Docker Management Design Patterns

Swarm Mode on Amazon Web Services — Deepak Vohra



Docker Management Design Patterns

Swarm Mode on Amazon Web Services



Deepak Vohra

Apress[®]

Docker Management Design Patterns: Swarm Mode on Amazon Web Services

Deepak Vohra White Rock, British Columbia, Canada

ISBN-13 (pbk): 978-1-4842-2972-9 ISBN-13 (electronic): 978-1-4842-2973-6 https://doi.org/10.1007/978-1-4842-2973-6

Library of Congress Control Number: 2017955383

Copyright © 2017 by Deepak Vohra

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Cover image by Freepik (www.freepik.com)

Managing Director: Welmoed Spahr Editorial Director: Todd Green Acquisitions Editor: Steve Anglin Development Editor: Matthew Moodie Technical Reviewers: Michael Irwin and Massimo Nardone Coordinating Editor: Mark Powers Copy Editor: Kezia Endsley

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit http://www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484229729. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

Contents at a Glance

About the Author xiii
About the Technical Reviewersxv
Introductionxvii
Chapter 1: Getting Started with Docker 1
Chapter 2: Using Docker in Swarm Mode9
Chapter 3: Using Docker for AWS to Create a Multi-Zone Swarm
Chapter 4: Docker Services 55
Chapter 5: Scaling Services85
Chapter 6: Using Mounts
Chapter 7: Configuring Resources115
Chapter 8: Scheduling 131
Chapter 9: Rolling Updates 155
Chapter 10: Networking 179
Chapter 11: Logging and Monitoring 201
Chapter 12: Load Balancing 219
Chapter 13: Developing a Highly Available Website
Chapter 14: Using Swarm Mode in Docker Cloud
Chapter 15: Using Service Stacks 297
Index

Contents

About the Author	xiii
About the Technical Reviewers	XV
Introduction	xvii
Chapter 1: Getting Started with Docker	1
Setting the Environment	1
Running a Docker Application	3
Summary	7
Chapter 2: Using Docker in Swarm Mode	9
The Problem	9
The Solution	10
Docker Swarm Mode	10
Nodes	
Service	11
Desired State of a Service	11
Manager Node and Raft Consensus	11
Worker Nodes	12
Quorum	12
Setting the Environment	14
Initializing the Docker Swarm Mode	14
Joining Nodes to the Swarm	18
Testing the Swarm	20
Promoting a Worker Node to Manager	24
Demoting a Manager Node to Worker	25

Making a Worker Node Leave the Swarm	25
Making a Manager Node Leave the Swarm	26
Reinitializing a Cluster	28
Modifying Node Availability	28
Removing a Node	30
Summary	30
Chapter 3: Using Docker for AWS to Create a Multi-Zone Swarm	31
The Problem	31
The Solution	32
Setting the Environment	33
Creating a AWS CloudFormation Stack for Docker Swarm	34
Connecting with the Swarm Manager	49
Using the Swarm	49
Deleting a Swarm	51
Summary	53
Summary Chapter 4: Docker Services	53 55
Summary Chapter 4: Docker Services The Problem	53 55 55
Summary Chapter 4: Docker Services The Problem The Solution	53 55 55
Summary Chapter 4: Docker Services The Problem The Solution Setting the Environment	53 55 55 55 57
Summary Chapter 4: Docker Services The Problem The Solution Setting the Environment The docker service Commands	53 55 55 57 59
Summary Chapter 4: Docker Services The Problem The Solution Setting the Environment The docker service Commands Types of Services	53 55 55 57 59 60
Summary Chapter 4: Docker Services The Problem The Solution Setting the Environment The docker service Commands Types of Services Creating a Service	53 55 55 57 59 60 60
Summary Chapter 4: Docker Services The Problem The Solution Setting the Environment The docker service Commands Types of Services Creating a Service Listing the Tasks of a Service	 53 55 55 57 59 60 60 61
Summary Chapter 4: Docker Services The Problem The Solution Setting the Environment The docker service Commands Types of Services Creating a Service Listing the Tasks of a Service Invoking a Hello World Service Task on the Command Line	 53 55 55 57 59 60 60 61 62
Summary Chapter 4: Docker Services The Problem The Solution Setting the Environment The docker service Commands Types of Services Creating a Service Listing the Tasks of a Service Invoking a Hello World Service Task on the Command Line Getting Detailed Information About a Service	 53 55 55 57 59 60 61 62 63
Summary Chapter 4: Docker Services The Problem The Solution Setting the Environment The docker service Commands Types of Services Creating a Service Listing the Tasks of a Service Invoking a Hello World Service Task on the Command Line Getting Detailed Information About a Service Invoking the Hello World Service in a Browser	 53 55 55 57 59 60 61 62 63 65
Summary Chapter 4: Docker Services The Problem The Solution Setting the Environment The docker service Commands Types of Services Creating a Service Listing the Tasks of a Service Invoking a Hello World Service Task on the Command Line Getting Detailed Information About a Service Invoking the Hello World Service in a Browser Creating a Service for a MySQL Database	 53 55 55 57 59 60 61 62 63 65 67

Listing Service Tasks	. <mark>68</mark>
Accessing a MySQL Database in a Docker Container	. 70
Updating a Service	. 73
Updating the Replicas	74
Updating the Docker Image Tag	75
Updating the Placement Constraints	79
Updating Environment Variables	<mark>80</mark>
Updating the Docker Image	<mark>81</mark>
Updating the Container Labels	82
Updating Resources Settings	82
Removing a Service	. 83
Creating a Global Service	. <mark>83</mark>
Summary	. <mark>84</mark>
Chapter 5: Scaling Services	. 85
The Problem	. 85
The Solution	. 86
Setting the Environment	. 87
Creating a Replicated Service	. 87
Scaling Up a Service	. 88
Scaling Down a Service	. 91
Removing a Service	. 92
Global Services Cannot Be Scaled	. <mark>92</mark>
Scaling Multiple Services Using the Same Command	. <mark>9</mark> 3
Service Tasks Replacement on a Node Leaving the Swarm	. <mark>9</mark> 5
Summary	. <mark>96</mark>
Chapter 6: Using Mounts	. 97
The Problem	. 97
The Solution	. 97

Volume Mounts97
Bind Mounts
Setting the Environment
Creating a Named Volume 100
Using a Volume Mount
Removing a Volume
Creating and Using a Bind Mount112
Summary114
Chapter 7: Configuring Resources
The Problem
The Solution
Setting the Environment118
Creating a Service Without Resource Specification
Reserving Resources
Setting Resource Limits 120
Creating a Service with Resource Specification121
Scaling and Resources121
Reserved Resources Must Not Be More Than Resource Limits 122
Rolling Update to Modify Resource Limits and Reserves 124
Resource Usage and Node Capacity 125
Scaling Up the Stack
Summary130
Chapter 8: Scheduling 131
The Problem
The Solution
Setting the Environment135
Creating and Scheduling a Service: The Spread Scheduling136
Desired State Reconciliation

Scheduling Tasks Limited by Node Resource Capacity	
Adding Service Scheduling Constraints	
Scheduling on a Specific Node	
Adding Multiple Scheduling Constraints	
Adding Node Labels for Scheduling	
Adding, Updating, and Removing Service Scheduling Constraints	
Spread Scheduling and Global Services	
Summary	
Chapter 9: Rolling Updates	
The Problem	
The Solution	
Setting the Environment	
Creating a Service with a Rolling Update Policy	157
Rolling Update to Increase the Number of Replicas	
Rolling Update to a Different Image Tag	1 <mark>6</mark> 1
Rolling Update to Add and Remove Environment Variables	1 <mark>6</mark> 2
Rolling Update to Set CPU and Memory Limits and Reserve	
Rolling Update to a Different Image	
Rolling Restart	
Rolling Update to Add and Remove Mounts	
Rolling Update Failure Action	
Roll Back to Previous Specification	175
Rolling Update on a Global Service	
Summary	
Chapter 10: Networking	
The Problem	
The Solution	
The Ingress Network	
Custom Overlay Networks	

х

The docker_gwbridge Network	181
The Bridge Network	181
Setting the Environment	182
Networking in Swarm Mode	183
Using the Default Bridge Network to Create a Service	1 <mark>86</mark>
Creating a Service in the Ingress Network	187
Creating a Custom Overlay Network	191
Using a Custom Overlay Network to Create a Service	194
Creating an Internal Overlay Network	195
Deleting a Network	1 <mark>98</mark>
Summary	199
Chapter 11: Logging and Monitoring	201
The Problem	201
The Solution	201
Setting the Environment	202
Creating a SPM Application	203
Creating a Logsene Application	205
Connecting the SPM and Logsene Apps	208
Deploying the Sematext Docker Agent as a Service	209
Creating a MySQL Database Service on a Docker Swarm	212
Monitoring the Docker Swarm Metrics	213
Getting Docker Swarm Logs in Logsene	214
Summary	217
Chapter 12: Load Balancing	219
Service Discovery	219
Custom Scheduling	219
Ingress Load Balancing	219
The Problem	219
The Solution	220

Setting the Environment	. 221
Creating a Hello World Service	. 222
Invoking the Hello World Service	. 224
Creating an External Elastic Load Balancer	. 227
Load Balancing in Docker for AWS	. 234
Summary	. 239
Chapter 13: Developing a Highly Available Website	241
The Problem	. 241
The Solution	. 242
Setting the Environment	. 243
Creating Multiple Docker Swarms	. 243
Deploying a Docker Swarm Service	. 246
Creating an Amazon Route 53	. 251
Creating a Hosted Zone	<mark>252</mark>
Configuring Name Servers	254
Creating Resource Record Sets	256
Testing High Availability	. <mark>26</mark> 3
Deleting a Hosted Zone	. <mark>266</mark>
Summary	. 269
Chapter 14: Using Swarm Mode in Docker Cloud	271
The Problem	. 271
The Solution	. 271
Setting the Environment	. 272
Creating an IAM Role	. 272
Creating a Docker Swarm in Docker Cloud	. 280
Connecting to the Docker Swarm from a Docker Host	. 289
Connecting to the Docker Swarm from a Swarm Manager	. 292
Bringing a Swarm into Docker Cloud	. 294
Summary	. <mark>296</mark>

CONTENTS

Chapter 15: Using Service Stacks 29	97
The Problem	97
The Solution	97
Setting the Environment29	99
Configuring a Service Stack	03
Creating a Stack	04
Listing Stacks	05
Listing Services	06
Listing Docker Containers	07
Using the Service Stack	80
Removing a Stack	14
Summary31	15
Index	17

About the Author



Deepak Vohra is an Oracle certified Java programmer and web component developer. Deepak has published in several journals, including *Oracle Magazine, OTN, IBM developerWorks, ONJava, DevSource, WebLogic Developer's Journal, XML Journal, Java Developer's Journal, FTPOnline,* and *devx.* Deepak has published three other books on Docker, and a dozen other books on other topics. Deepak is also a Docker Mentor.

About the Technical Reviewers

Michael Irwin is an Application Architect at Virginia Tech (Go Hokies!) where he's both a developer and evangelist for cutting-edge technologies. He is helping Virginia Tech adopt Docker, cloud services, single-page applications, CI/CD pipelines, and other current development practices. As a Docker Captain and a local meetup organizer, he is very active in the Docker community giving presentations and trainings to help others learn how to best utilize Docker in their organizations. Find him on Twitter at @mikesir87.



Massimo Nardone has more than 23 years of experience in security, web/mobile development, and cloud and IT architecture. His true IT passions are security and Android systems.

He has been programming and teaching people how to program with Android, Perl, PHP, Java, VB, Python, C/C++, and MySQL for more than 20 years.

He holds a Master's of Science degree in Computing Science from the University of Salerno, Italy.

He worked as a project manager, software engineer, research engineer, chief security architect, information security manager, PCI/SCADA auditor, and senior lead IT security/cloud/SCADA architect for many years.

His technical skills include security, Android, cloud, Java, MySQL,

Drupal, Cobol, Perl, web and mobile development, MongoDB, D3, Joomla, Couchbase, C/C++, WebGL, Python, Pro Rails, Django CMS, Jekyll, Scratch, and more.

He worked as a visiting lecturer and supervisor for exercises at the Networking Laboratory of the Helsinki University of Technology (Aalto University). He holds four international patents (in the PKI, SIP, SAML, and Proxy areas).

He currently works as the Chief Information Security Office (CISO) for Cargotec Oyj and is a member of ISACA, Finland Chapter Board.

Massimo has reviewed more than 40 IT books for different publishers and he is the coauthor of *Pro Android Games* (Apress, 2015).

Introduction

Docker, made available as open source in March 2013, has become the de facto containerization platform. The Docker Engine by itself does not provide functionality to create a distributed Docker container cluster or the ability to scale a cluster of containers, schedule containers on specific nodes, or mount a volume. The book is about orchestrating Docker containers with the Docker-native Swarm mode, which was introduced July 2016 with Docker 1.12. Docker Swarm mode should not be confused with the legacy standalone Docker Swarm, which is not discussed in the book. The book discusses all aspects of orchestrating/managing Docker, including creating a Swarm, using mounts, scheduling, scaling, resource management, rolling updates, load balancing, high availability, logging and monitoring, using multiple zones, and networking. The book also discusses the managed services for Docker Swarm: Docker for AWS and Docker Cloud Swarm mode.

Docker Swarm Design Patterns

"A software design pattern is a general reusable solution to a commonly occurring problem within a given context in software design." (Wikipedia)

Docker Swarm mode provides several features that are general-purpose solutions to issues inherent in a single Docker Engine. Each chapter starting with Chapter 2 introduces a problem and discusses a design pattern as a solution to the problem.

Why Docker Swarm Mode?

Why use the Docker Swarm mode when several container cluster managers are available? Docker Swarm mode is Docker-native and does not require the complex installation that some of the other orchestration frameworks do. A managed service Docker for AWS is available for Docker Swarm to provision a Swarm on production-ready AWS EC2 nodes. Docker Cloud may be linked to Docker for AWS to provision a new Swarm or connect to an existing Swarm. Docker 1.13 includes support for deploying a Docker Stack (collection of services) on Docker Swarm with Docker Compose.

What the Book Covers

Chapter 1 introduces running a Docker standalone container on CoreOS Linux. The chapter establishes the basis of the book and subsequent chapters discuss how the management design patterns provided by the Swarm mode solve problems inherent in a standalone Docker Engine.

Chapter 2 introduces the Swarm mode, including initializing a Swarm and joining worker nodes to the Swarm. Chapter 2 includes promoting/demoting a node, making a node (manager or worker) leave a Swarm, reinitializing a Swarm, and modifying node availability.

INTRODUCTION

Chapter 3 discusses the managed service Docker for AWS, which provisions a Docker Swarm by supplying the Swarm parameters, including the number of managers and workers and the type of EC2 instances to use. AWS uses an AWS CloudFormation to create the resources for a Swarm. Docker for AWS makes it feasible to create a Swarm across multiple AWS zones.

Chapter 4 is about Docker services. Two types of services are defined—replicated and global. Chapter 4 discusses creating a service (replicated and global), scaling a replicated service, listing service tasks, and updating a service.

Chapter 5 discusses scaling replicated services in more detail, including scaling multiple services simultaneously. Global services are not scalable.

In Chapter 6, two types of mounts are defined: a *bind* mount and *volume* mount. This chapter discusses creating and using each type of mount.

Chapter 7 is about configuring and using resources in a Swarm. Two types of resources are supported for configuration: memory and CPU. Two types of resource configurations are defined: *reserves* and *limits*. It discusses creating a service with and without resources specification.

Chapter 8 discusses scheduling service tasks with the default and custom scheduling. Scheduling constraints are also discussed.

Chapter 9 discusses rolling updates, including setting a rolling update policy. Different types of rolling updates are provisioned, including updating to a different Docker image tag, adding/removing environment variables, updating resource limits/reserves, and updating to a different Docker image.

Chapter 10 is about networking in Swarm mode, including the built-in overlay networking called *ingress* and support for creating a custom overlay network.

Chapter 11 is about logging and monitoring in a Swarm, which does not provide a built-in support for logging and monitoring. Logging and monitoring is provided in a Swarm with a Sematext Docker agent, which sends metrics to a SPM dashboard and logs to a Logsene user interface and Kibana.

Chapter 12 discusses load balancing across service tasks with *ingress* load balancing. An external AWS elastic load balancer may also be added for distributing client requests across the EC2 instances on which a Swarm is based.

Chapter 13 discusses developing a highly available website that uses an Amazon Route 53 to create a hosted zone with resource record sets configured in a Primary/Secondary failover mode.

Chapter 14 discusses another managed service, Docker Cloud, which may be used to provision a Docker Swarm or connect to an existing Swarm.

Chapter 15 discusses Docker service stacks. A *stack* is a collection of services that have dependencies among them and are defined in a single configuration file for deployment.

Who this Book Is For

The primary audience of this book includes Docker admins, Docker application developers, and Container as a Service (CaaS) admins and developers. Some knowledge of Linux and introductory knowledge of Docker—such as using a Docker image to run a Docker container, connecting to a container using a bash shell, and stopping and removing a Docker container—is required.

CHAPTER 1

Getting Started with Docker

Docker has become the de facto containerization platform. The main appeal of Docker over virtual machines is that it is lightweight. Whereas a virtual machine packages a complete OS in addition to the application binaries, a Docker container is a lightweight abstraction at the application layer, packaging only the code and dependencies required to run an application. Multiple Docker containers run as isolated processes on the same underlying OS kernel. Docker is supported on most commonly used OSes, including several Linux distributions, Windows, and MacOS. Installing Docker on any of these platforms involves running several commands and also setting a few parameters. CoreOS Linux has Docker installed out-of-the-box. We will get started with using Docker Engine on CoreOS in this chapter. This chapter sets the context of the subsequent chapters, which discuss design patterns for managing Docker Engine using the Swarm mode. This chapter does not use Swarm mode and provides a contrast to using the Swarm mode. This chapter includes the following sections:

- Setting the environment
- Running a Docker application

Setting the Environment

We will be using CoreOS on Amazon Web Services (AWS) EC2, which you can access at https://console. aws.amazon.com/ec2/v2/home?region=us-east-1#. Click on Launch Instance to lauch an EC2 instance. Next, choose an Amazon Machine Image (AMI) for CoreOS. Click on AWS Marketplace to find a CoreOS AMI. Type CoreOS in the search field to find a CoreOS AMI. Select the Container Linux by CoreOS (Stable), as shown in the EC2 wizard in Figure 1-1, to launch an instance.

2. Choose Instance Type 3	Configure Instance 4. Add St	orage 5. Add Tags	6. Configure Security Group	7. Review	
oose an Amazon ate that contains the softwar our user community, or the	e configuration (operating syst AWS Marketplace; or you can	AMI) em, application server, select one of your own	and applications) required to AMIs.	launch your instance	Cancel and Exit
Q coreos		×		I< < 1	to 6 of 6 Products > >
containe 🥲	rlinux Container Linux by	CoreOS (Stable)	oreOS		Select
MIS Free tier e	\$0.00/hr for software + AW ligible Linux/Unix, Other 1235.9.0	S usage fees 64-bit Amazon Machine Ima	ge (AMI) Updated: 2/8/17		
	CoreOS Container Linux and containers running More info	automates software up on large-scale clusters.	dates to ensure better security a Operating system updates	and reliability of machir	105
	2. Choose Instance Type 3 OOSE AN AMAZON ate that contains the softwar our user community, or the C, coreos ace Mis Free ter effective Sector (0)	2. Choose Instance Type 3. Configure Instance 4. Add Site OOSEE AN AMAZON MACHINE IMAGE (A ate that contains the software configuration (operating syste our user community, or the AWS Marketplace; or you can se Q coreos ace Mils Free Ver elgeber More info	2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags OOSE an AMAZON MAChine Image (AMI) ate that contains the software configuration (operating system, application server, our user community, or the AWS Marketplace; or you can select one of your own Q, coreos X ace Mis Free tier digits Free tier digits Free tier digits More info	2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group OOSee an Amazon Machine Image (AMI) ate that contains the software configuration (operating system, application server, and applications) required to our user community, or the AWS Marketplace; or you can select one of your own AMIs. Image: Image: <t< td=""><td>2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review OOSee an Amazon Machine Image (AMI) ate that contains the software configuration (operating system, application server, and applications) required to launch your instance our user community, or the AWS Marketplace; or you can select one of your own AMIs. Image: Image:</td></t<>	2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review OOSee an Amazon Machine Image (AMI) ate that contains the software configuration (operating system, application server, and applications) required to launch your instance our user community, or the AWS Marketplace; or you can select one of your own AMIs. Image: Image:

Figure 1-1. Selecting an AMI for CoreOS Linux

CHAPTER 1 GETTING STARTED WITH DOCKER

From Choose an Instance Type, choose the t2.micro Type and click on Next. In Configure Instance Details, specify the number of instances as 1. Select a network or click on Create New VPC to create a new VPC. Select a subnet or click on Create New Subnet to create a new subnet. Select Enable for Auto-Assign Public IP. Click on Next.

From Add Storage, select the default settings and click on Next. In Add Tags, no tags need to be added. Click on Next. From Configure Security Group, add a security group to allow all traffic of any protocol in all port ranges from any source (0.0.0.0/0). Click on Review and Launch and subsequently click on Launch.

Select a key pair and click on Launch Instances in the Select an Existing Key Pair or Create a New Key Pair dialog, as shown in Figure 1-2.

A key they a	pair consists of a public key that AWS stores, and a private k e allow you to connect to your instance securely. For Windows AM	ey file that you store. Together, Is, the private key file is required
secure	ely SSH into your instance.	, the private key file allows you t
Note:	The selected key pair will be added to the set of keys authorize	d for this instance. Learn more
about	removing existing key pairs from a public AMI.	
C	Choose an existing key pair	٣
Se	elect a key pair	
	oreos	
C		
C	I acknowledge that I have access to the selected private key fi	le (coreos pem) and that

Figure 1-2. Launch instances

An EC2 instance with CoreOS is launched. Obtain the public DNS or IPv4 public IP address of the EC2 instance from the EC2 Console, as shown in Figure 1-3, to SSH login into the instance.

Q search : i-	09dd4e6018d23214	47 💿 Add filter							0	K -	< 1 to 1	of 1	> >
Name	* 1	nstance ID		Instance Type	- Availabi	lity Zone -	Instance State -	Status Checks ~	Alarm Sta	atus	Public [DNS (IF	Pv4)
CoreOS	į	09dd4e6018d232	1	t2.micro	us-east-1	le	running	2/2 checks	None	>	ec2-54-1	196-51-	20.com
nstance: i- 09	dd4e6018d23214	7 (CoreOS)	Public	: DN \$: ec2-54	-196-51-20.co	mpute-1.am	azonaws.com					82	
nstance: i-09	dd4e6018d23214	7 (CoreOS)	Public	DNS: ec2-54	-196-51-20.co	mpute-1.an	nazonaws.com					88	
Description	dd4e6018d23214 Status Checks	7 (CoreOS)	Public	DNS: ec2-54 gs Usage	-196-51-20.co	mpute-1.an	azonaws.com					82	
nstance: i-09	dd4e6018d23214 Status Checks Instance ID	7 (CoreOS) I Monitoring i-09dd4e6018d	Public Ta	gs Usage	-196-51-20.co	mpute-1.an	Public DNS (IP	 (4) ec2-54-196-51 1.amazonaws. 	-20.compute	8-		88	
Instance: i-09 Description	dd4e6018d23214 Status Checks Instance ID Instance state	7 (CoreOS) Monitoring i-09dd4e6018d running	Public Ta	gs Usage	196-51-20.co	mpute-1.an	Public DNS (IP)	 ec2-54-196-51 1.amazonaws. IP 54.196.51.20 	-20.compute	8-		88	

Figure 1-3. Public DNS and public IPv4

SSH login into the EC2 instance as user "core".

```
ssh -i "coreos.pem" core@<public ip>
```

commit

Running a Docker Application

As mentioned earlier, Docker is pre-installed on CoreOS. Run the docker command to list its usage, as shown in the following bash shell:

```
core@ip-172-30-4-75 ~ $ docker
Usage: docker [OPTIONS] COMMAND [arg...]
       docker [ --help | -v | --version ]
A self-sufficient runtime for containers.
Options:
  --config=~/.docker
                                  Location of client config files
 -D, --debug
                                  Enable debug mode
                                  Daemon socket(s) to connect to
 -H, --host=[]
 -h, --help
                                  Print usage
 -l, --log-level=info
                                  Set the logging level
 --tls
                                  Use TLS; implied by --tlsverify
 --tlscacert=~/.docker/ca.pem
                                  Trust certs signed only by this CA
 --tlscert=~/.docker/cert.pem
                                  Path to TLS certificate file
 --tlskey=~/.docker/key.pem
                                  Path to TLS key file
 --tlsverify
                                  Use TLS and verify the remote
 -v, --version
                                  Print version information and quit
Commands:
   attach
             Attach to a running container
              Build an image from a Dockerfile
   build
```

Create a new image from a container's changes

ср	Copy files/folders between a container and the local filesystem
create	Create a new container
diff	Inspect changes on a container's filesystem

Output the Docker version using the docker version command. For native Docker Swarm support, the Docker version must be 1.12 or later as listed in the bash shell output.

```
core@ip-172-30-4-75 ~ $ docker version
Client:
Version:
            1.12.6
API version: 1.24
Go version: go1.7.5
Built: Mon Jun 19 23:04:34 2017
OS/Arch: linux/amd64
Server:
            1.12.6
Version:
API version: 1.24
Go version: go1.7.5
Git commit: a82d35e
 Built:
              Mon Jun 19 23:04:34 2017
OS/Arch:
             linux/amd64
```

Run a Hello World app with the tutum/hello-world Docker image.

docker run -d -p 8080:80 --name helloapp tutum/hello-world

The Docker image is pulled and a Docker container is created, as shown in the following listing.

```
core@ip-172-30-4-75 ~ $ docker run -d -p 8080:80 --name helloapp tutum/hello-world
Unable to find image 'tutum/hello-world:latest' locally
latest: Pulling from tutum/hello-world
658bc4dc7069: Pull complete
a3ed95caeb02: Pull complete
af3cc4b92fa1: Pull complete
d0034177ece9: Pull complete
983d35417974: Pull complete
Digest: sha256:0d57def8055178aafb4c7669cbc25ec17f0acdab97cc587f30150802da8f8d85
Status: Downloaded newer image for tutum/hello-world:latest
1b7a85df6006b41ea1260b5ab957113c9505521cc8732010d663a5e236097502
```

List the Docker container using the docker ps command.

core@ip-172-30	-4-75 ~ \$ docker ps	;		
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
PORTS	NAMES			
1b7a85df6006	tutum/hello-world	"/bin/sh -c 'php-fpm "	19 minutes ago	Up 19 minutes
0.0.0.0:8080->	80/tcp helloapp			

~ •

The port mapping for the Docker container is also listed using the docker ps command, but it may also be obtained using the docker port <container> command.

```
core@ip-172-30-4-75 ~ $ docker port helloapp
80/tcp -> 0.0.0.0:8080
```

Using the 8080 port and localhost, invoke the Hello World application with curl.

```
curl localhost:8080
```

The HTML markup for the Hello World application is output, as listed shown here.

```
core@ip-172-30-4-75 ~ $ curl localhost:8080
<html>
<head>
        <title>Hello world!</title>
        <link href='http://fonts.googleapis.com/css?family=Open+Sans:400,700'</pre>
         rel='stylesheet' type='text/css'>
        <style>
        body {
                background-color: white;
                text-align: center;
                padding: 50px;
                font-family: "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;
        }
        #logo {
                margin-bottom: 40px;
        }
        </style>
</head>
<body>
        <img id="logo" src="logo.png" />
        <h1>Hello world!</h1>
        <h3>My hostname is 1b7a85df6006</h3>
</bodv>
</html>
```

Using the public DNS for the EC2 instance, the Hello World application may also be invoked in a browser. This is shown in the web browser in Figure 1-4.

CHAPTER 1 GETTING STARTED WITH DOCKER



Figure 1-4. Invoking the Hello World application in a web browser

The docker stop <container> command stops a Docker container. The docker rm <container> command removes a Docker container. You can list Docker images using the docker images command. A Docker image may be removed using the docker rmi <image> command.

```
core@ip-172-30-4-75 ~ $ docker stop helloapp
helloapp
core@ip-172-30-4-75 ~ $ docker rm helloapp
helloapp
core@ip-172-30-4-75 ~ $ docker images
REPOSITORY
                                        IMAGE ID
                                                            CREATED
                                                                                 SIZE
                    TAG
tutum/hello-world
                   latest
                                        31e17b0746e4
                                                            19 months ago
                                                                                 17.79 MB
core@ip-172-30-4-75 ~ $ docker rmi tutum/hello-world
Untagged: tutum/hello-world:latest
Untagged: tutum/hello-world@sha256:0d57def8055178aafb4c7669cbc25ec17f0acdab97cc587f30150802da8f8d85
Deleted: sha256:31e17b0746e48958b27f1d3dd4fe179fbba7e8efe14ad7a51e964181a92847a6
Deleted: sha256:e1bc9d364d30cd2530cb673004dbcdf1eae0286e41a0fb217dd14397bf9debc8
Deleted: sha256:a1f3077d3071bd3eed5bbe5c9c036f15ce3f6b4b36bdd77601f8b8f03c6f874f
Deleted: sha256:ff7802c271f507dd79ad5661ef0e8c7321947c145f1e3cd434621fa869fa648d
Deleted: sha256:e38b71a2478cad712590a0eace1e08f100a293ee19a181d5f5d5a3cdb0663646
Deleted: sha256:5f27c27ccc6daedbc6ee05562f96f719d7f0bb38d8e95b1c1f23bb9696d39916
Deleted: sha256:fab20b60d8503ff0bc94ac3d25910d4a10f366d6da1f69ea53a05bdef469426b
Deleted: sha256:a58990fe25749e088fd9a9d2999c9a17b51921eb3f7df925a00205207a172b08
core@ip-172-30-4-75 ~ $
```

Summary

This chapter sets the basis for subsequent chapters by using a single Docker Engine on CoreOS. Subsequent chapters explore the different design patterns for managing distributed Docker applications in a cluster. The next chapter introduces the Docker Swarm mode.

CHAPTER 2

Using Docker in Swarm Mode

The Docker Engine is a containerization platform for running Docker containers. Multiple Docker containers run in isolation on the same underlying operating system kernel, with each container having its own network and filesystem. Each Docker container is an encapsulation of the software and dependencies required for an application and does not incur the overhead of packaging a complete OS, which could be several GB. Docker applications are run from Docker images in Docker containers, with each Docker image being specific to a particular application or software. A Docker image is built from a *Dockerfile*, with a Dockerfile defining the instruction set to be used to download and install software, set environment variables, and run commands.

The Problem

While the Docker Engine pre-1.12 (without native Swarm mode) is well designed for running applications in lightweight containers, it lacks some features, the following being the main ones.

- *No distributed computing*—No distributed computing is provided, as a Docker Engine is installed and runs on a single node or OS instance.
- *No fault tolerance*—As shown in the diagram in Figure 2-1, if the single node on which a Docker Engine is running fails, the Docker applications running on the Docker Engine fail as well.



Figure 2-1. Single node Docker cluster

The Solution

With Docker Engine version 1.12 onward, Docker container orchestration is built into the Docker Engine in *Swarm mode* and is native to the Docker Engine. Using the Swarm mode, a swarm (or cluster) of nodes distributed across multiple machines (OS instances) may be run in a master/worker/ pattern. Docker Swarm mode is not enabled in the Docker Engine by default and has to be initialized using a docker command. Next, as an introduction to the Docker Swarm mode, we introduce some terminology.

Docker Swarm Mode

Docker Swarm is a cluster of Docker hosts connected by an overlay networking for service discovery. A Docker Swarm includes one or more *manager nodes* and one or more *worker nodes*, as shown in Figure 2-2. In the Swarm mode, a Docker service is the unit of Docker containerization. Docker containers for a service created from a Manager node are deployed or scheduled across the cluster and the Swarm includes a built-in load balancing for scaling the services. The expected state for a service is declared on the manager, which then schedules the task to be run on a node. However, the worker node itself still pulls the image and starts the container.



Figure 2-2. Docker Swarm mode cluster

Nodes

An instance of a Docker host (a Docker Engine) is called a *node*. Two types of node roles are provided: *manager nodes* and *worker nodes*.

Service

A *service* is an abstraction for a collection of tasks (also called replicas or replica tasks) distributed across a Swarm. As an example, a service could be running three replicas of an Nginx server. Default scheduling, which is discussed in Chapter 7, uses the "spread" scheduling strategy, which spreads the tasks across the nodes of the cluster based on a computed node rank. A service consists of one or more tasks that run independent of each other, implying that stopping a task or starting a new task does not affect running other tasks. The Nginx service running on three nodes could consist of three replica tasks. Each task runs a Docker container for the service. One node could be running multiple tasks for a service. A task is an abstraction for the atomic unit of scheduling, a "slot" for the scheduler to run a Docker container.

Desired State of a Service

The "desired state" of a service refers to the service state as defined in the service definition when creating the service. As an example, a service definition could define a service as consisting of three replicas of an Nginx server.

Manager Node and Raft Consensus

When the Swarm is first created, the current node becomes the first manager node. By default, all manager nodes are also workers. The manager node performs the cluster orchestration and manages the Swarm, including the initial scheduling of service tasks and subsequent reconciliation, if any, between the desired state and the actual state of services. As an example, for a service definition consisting of three replicas of an Nginx server, the manager node would create three tasks and schedule the tasks on Swarm worker nodes in the Swarm. Subsequently, if a node running a task were to fail, the Swarm manager would start a new replacement task on the worker nodes still in the Swarm. The Swarm manager accepts the service definition when a service is created and schedules the service on one or more worker nodes as service tasks. The Swarm manager node also manages the scaling of service by adding/removing service tasks. The Swarm manager accepts the current as the service a unique DNS name and starts Docker containers via service replica tasks. The manager node monitors the cluster state. The Swarm manager is also a worker node by default, which is discussed in the next section.

To refer to "the manager node" is actually a simplification of the Swarm Manager, as a Swarm may consist of one or more manager nodes. Each manager node keeps the complete cluster state data, including which service replica tasks are running on which node and the node roles, and participates in Swarm management for the *Raft consensus*. The Raft consensus is merely an algorithm to create decisions/ agreements (consensus) within a group in a distributed fashion. Swarm uses it to make decisions such as leader elections, cluster membership, service changes, etc. In the Swarm mode, Raft consensus is an agreement among the manager nodes for a global cluster state parameter such as about the state of data value stored in a database. Swarm managers share data using Raft. Raft consensus is a protocol for implementing distributed consensus among all the reachable manager nodes in a Swarm. The Raft Consensus Algorithm has several implementations and its implementation in the Swarm mode has the properties typically found in distributed systems, such as the following:

- Agreement of values for fault tolerance
- Cluster membership management
- Leader election using mutual exclusion

Only one manager node, called the *leader*, performs all the cluster orchestration and management. Only the leader node performs the service scheduling, scaling, and restarting of service tasks. The other manager nodes are for the fault tolerance of Swarm manager, which implies that if the leader node were to fail, one of the other manager nodes would be elected as the new leader and take over the cluster management. Leader election is performed by a consensus from the majority of the manager nodes.

Worker Nodes

A worker node actually runs the service replica tasks and the associated Docker containers. The differentiation between node roles as manager nodes and worker nodes is not handled at service deployment time but is handled at runtime, as node roles may be promoted/demoted. Promoting/demoting a node is discussed in a later section. Worker nodes do not affect the manager Raft consensus. Worker nodes only increase the capacity of the Swarm to run service replica tasks. The worker nodes themselves do not contribute to the voting and state held in the raft, but the fact that they are worker nodes is held within the raft. As running a service task requires resources (CPU and memory) and a node has a certain fixed allocatable resources, the capacity of a Swarm is limited by the number of worker nodes in the Swarm.

Quorum

A *quorum* refers to agreement among the majority of Swarm manager nodes or managers. If a Swarm loses quorum it cannot perform any management or orchestration functions. The service tasks already scheduled are not affected and continue to run. The new service tasks are not scheduled and other management decisions requiring a consensus, such as adding or removing a node, are not performed. All Swarm managers are counted toward determining majority consensus for fault tolerance. For leader election only the reachable manager nodes are included for Raft consensus. Any Swarm update, such as the addition or removal of a node or the election of a new leader, requires a quorum. *Raft consensus* and *quorum* are the same. For high availability, three to five Swarm managers are recommended in production. An odd number of Swarm managers is recommended in general. *Fault tolerance* refers to the tolerance for failure of Swarm manager nodes or the number of Swarm managers that may fail without making a Swarm unavailable. Mathematically, "majority" refers to more than half, but for the Swarm mode Raft consensus algorithm, Raft tolerances (N-1)/2 failures and a majority for Raft consensus is determined by (N/2)+1. N refers to the Swarm size or the number of manager nodes in the Swarm.

```
Swarm Size = Majority + Fault Tolerance
```

As an example, Swarm sizes of 1 and 2 each have a fault tolerance of 0, as Raft consensus cannot be reached for the Swarm size if any of the Swarm managers were to fail. More manager nodes increase fault tolerance. For an odd number N, the fault tolerance is the same for a Swarm size N and N+1.

As an example, a Swarm with three managers has a fault tolerance of 1, as shown in Figure 2-3. Fault tolerance and Raft consensus do not apply to worker nodes, as Swarm capacity is based only on the worker nodes. Even if two of the three worker nodes were to fail, one Worker node, even if the manager nodes are manager-only nodes, would keep the Swarm available though a reduction in Swarm capacity and could transition some of the running tasks to non-running state.



Figure 2-3. Fault tolerance for a Swarm

This section covers the following topics:

- Setting the environment
- Initializing the Docker Swarm mode
- Joining nodes to the Swarm cluster
- Testing the Swarm cluster
- Promoting a worker node to manager
- Demoting a manager node to worker
- Making a worker node leave the Swarm cluster
- Making A worker node rejoin the Swarm cluster
- Making a manager node leave the Swarm cluster
- Reinitializing a Swarm
- Modifying node availability
- Removing a node

CHAPTER 2 USING DOCKER IN SWARM MODE

Setting the Environment

This chapter shows you how to create a three-node Swarm consisting of one manager node and two worker nodes. Create three Amazon EC2 instances using CoreOS Stable AMI, as shown in the EC2 console in Figure 2-4. Enable all traffic between the EC2 instances when configuring the security group for the EC2 instances. Obtain the IP address of the EC2 instance started for the Swarm manager.

									C+		_
Q Instance S	State : Running 💿	Add filter					0	< <	1 to 3 of 3	3 > 3	>1
Name	* h	nstance ID	Instance Type -	Availability Zone -	Instance State ~	Status Checks 👻	Alarm State	us	Public DN	5 (IPv4)	1
SwarmMar	nager i-	01b12315cb7c833be	t2.micro	us-east-1f	running	2/2 checks	None	20	ec2-34-204	-168-21	7.
SwarmWo	rker i-	053336322e12698	t2.micro	us-east-1f	🥥 running	2/2 checks	None		ec2-34-204	-199-45	i.c
SwarmWo	rker i-	08943beee6e430d	t2.micro	us-east-1f	running	2/2 checks	None	20	ec2-34-231	-70-10.0	co
nstance: i-01 Description	b12315cb7c833b Status Checks	e (SwarmManager) Monitoring	Public DNS: ec:	2-34-204-168-217.con	npute-1.amazonaw	s.com					3
nstance: i-01	b12315cb7c833b	e (SwarmManager)) Public DNS: ec:	2-34-204-168-217.con	npute-1.amazonaw	s.com					3
nstance: i i-01 Description	b12315cb7c833b Status Checks Instance ID	e (SwarmManager) Monitoring i-01b12315cb7c83) Public DNS: ec: Tags Usage Inst	2-34-204-168-217.con	npute-1.amazonaw	s.com	68-217.compu	te-	-		3
nstance: i-01 Description	b12315cb7c833b Status Checks Instance ID	e (SwarmManager) Monitoring i-01b12315cb7c83	Public DNS: ec: Tags Usage Inst 33be	2-34-204-168-217.con	Public DNS (IF	s.com Pv4) ec2-34-204-11 1.amazonaws	68-217.compu	te-	-		2
nstance: i i-01 Description	b12315cb7c833b Status Checks Instance ID Instance state	e (SwarmManager) Monitoring i-01b12315cb7c83 running	Public DNS: ec: Tags Usage Inst 33be	2-34-204-168-217.con	Public DNS (IF	s.com 2v4) ec2-34-204-11 1.amazonaws c IP 34.204.168.2*	68-217.compu .com 17	te-			•
nstance: i-01 Description	b12315cb7c833b Status Checks Instance ID Instance state Instance type Elastic IPs	e (SwarmManager) Monitoring i-01b12315cb7c83 running t2.micro	Public DNS: ec: Tags Usage Inst 33be	2-34-204-168-217.con	Public DNS (If Public DNS (If IPv4 Public IPv6 Private I	s.com ec2-34-204-11 1.amazonawa c IP 34.204.168.2: .IPs - .INS ip-172-30-5-7	68-217.compu .com 17 0.ec2.internal	te-			-
nstance: i-01	b12315cb7c833b Status Checks Instance ID Instance state Instance type Elastic IPs Availability zone	e (SwarmManager) Monitoring i-01b12315cb7c83 running t2.micro us-east-1f	Public DNS: ec: Tags Usage Inst 33be	2-34-204-168-217.con	Public DNS (If Public DNS (If Prote Public Private I Private Private	s.com ec2-34-204-11 1.amazonawa c IP 34.204.168.2 IPs - INS ip-172-30-5-7 IPs 172.30.5.70	68-217.compu .com 17 0.ec2.internal	te-			•
nstance: i-01	b12315cb7c833b Status Checks Instance ID Instance state Instance type Elastic IPs Availability zone Security groups	e (SwarmManager) Monitoring i-01b12315cb7c83 running t2.micro us-east-1f Container Linux b	Public DNS: ec: Tags Usage Inst 33be	2-34-204-168-217.com ructions	Public DNS (If Public DNS (If Prv4 Public IPv6 Private (Private Secondary private	 ec2-34-204-11 1.amazonaws c IP 34-204.168.2 IPs - NNS ip-172-30-5-7 IPs 172.30.5.70 IPs 	68-217.compu .com 17 0.ec2.internal	te-			-

Figure 2-4. EC2 instances

Initializing the Docker Swarm Mode

Docker Swarm mode is not enabled by default and needs to be enabled. SSH login to the EC2 instance started for the Swarm manager using the public IP address.

```
ssh -i "coreos.pem" core@34.204.168.217
```

Docker Swarm mode is available starting with Docker version 1.12. Verify that the Docker version is at least 1.12 using the docker --version command.

```
[root@localhost ~]# ssh -i "coreos.pem" core@34.204.168.217
Container Linux by CoreOS stable (1409.7.0)
core@ip-172-30-5-70 ~ $ docker --version
Docker version 1.12.6, build a82d35e
```

To initialize the Swarm, use the docker swarm init options command. Some of the options the command supports are listed in Table 2-1.

Option	Description	Default Value
advertise-addr	Advertised address in the format <ip interface>[:port]. The advertised address is the IP address at which other nodes may access the Swarm. If an IP address is not specified, the Docker ascertains if the system has a single IP address and, if it does, the IP address and port 2337 is used. If the system has multiple IP addresses, theadvertise-addr must be specified for inter-manager communication and overlay networking.</ip interface>	
availability	Availability of the node. Should be one of active/pause/drain.	active
force-new-cluster	Whether to force create a new cluster from the current state. We discuss why it may be required to force create and use the option in this chapter.	false
listen-addr	Listen address in the format <ip interface>[:port].</ip interface>	0.0.0.0:2377

Table 2-1. Command Swarm init Options

Use the default values for all options except the --advertise-addr for which a default value is not provided. Use the private address for the advertised address, which may be obtained from the EC2 console, as shown in Figure 2-5. If the EC2 instances on AWS were in different regions, the external public IP address should be used to access the manager node, which may also be obtained from the EC2 console.

									U	· •	
Q Instance S	State : Running	Add filter					0	< <	1 to 3 of	3 >	>
Name	*	Instance ID	Instance Type	Availability Zone 👻	Instance State	Status Checks 👻	Alarm Sta	tus	Public DN	IS (IPv4)
SwarmMa	inager	i-01b12315cb7c833b	e t2.micro	us-east-1f	running	2/2 checks	None	7	ec2-34-20	4-168-2	17.
SwarmWo	orker	i-053336322e12698	. t2.micro	us-east-1f	running	2/2 checks	None	20	ec2-34-20	4-199-4	5.c
SwarmWo	orker	-08943beee6e430d	t2.micro	us-east-1f	running	2/2 checks	None	10	ec2-34-23	1-70-10	.co
nstance: i i-01 Description	b12315cb7c833 Status Check	be (SwarmManager	r) Public DNS: ec: Tags Usage Inst	2-34-204-168-217.con	npute-1.amazona	ws.com			L		3
nstance: i i-01 Description	Status Check	Monitoring	r) Public DNS: ec: Tags Usage Inst	2-34-204-168-217.con	npute-1.amazona Public DNS (ws.com	58-217.comp	ute-			
nstance: i-01 Description	Status Check	be (SwarmManager Monitoring i-01b12315cb7c8	r) Public DNS: ec: Tags Usage Inst 33be	2-34-204-168-217.con	Public DNS	ws.com IPv4) ec2-34-204-10 1.amazonaws	68-217.comp	ute-			-
nstance: i-01 Description	Ib12315cb7c833 Status Check Instance I Instance stat	be (SwarmManager Monitoring i-01b12315cb7c8 running	r) Public DNS: ec: Tags Usage Inst 33be	2-34-204-168-217.con	npute-1.amazona Public DNS (IPv4 Put	VS.com (Pv4) ec2-34-204-10 1.amazonaws olic IP 34.204.168.21	68-217.comp .com 17	ute-			-
nstance: i i-01 Description	Ib12315cb7c833 Status Check Instance II Instance stat Instance typ	be (SwarmManager Monitoring i-01b12315cb7c8 running t2.micro	r) Public DNS: ec: Tags Usage Inst 33be	2-34-204-168-217.con	Public DNS Public DNS IPv4 Pub IPv	ws.com IPv4) ec2-34-204-10 1.amazonaws blic IP 34 204.168.21 6 IPs -	58-217.comp .com 17	ute-			
nstance: i-01	Status Check Instance I Instance stat Instance typ Elastic IP	be (SwarmManager Monitoring i-01b12315cb7c8 running t2 micro	r) Public DNS: ec: Tags Usage Inst 33be	2-34-204-168-217.con	npute-1.amazona Public DNS (IPv4 Put IPv Private	ws.com (Pv4) ec2-34-204-10 1.amazonaws blic IP 34.204.168.21 6 IPs - DNS ip-172-30-5-70	68-217.comp .com 17 D.ec2.interna	ute-			
nstance: i-01	Status Check Instance II Instance stat Instance typ Elastic IP Availability zon	be (SwarmManager Monitoring i-01b12315cb7c8 running t2 micro us-east-1f	r) Public DNS: ec: Tags Usage Inst 33be	2-34-204-168-217.con	Public DNS (Public DNS (IPv4 Put IPv Private Private	ws.com IPv4) ec2-34-204-11 1.amazonaws blic IP 34 204 168 2' 6 IPs - DNS ip-172-30-5-7/ ie IPs 172.30.5.70	68-217.comp .com 17 0.ec2.interna	ute-			-
nstance: i-01	Ib12315cb7c833 Status Check Instance I Instance stat Instance typ Elastic IP Availability zon Security group	be (SwarmManager Monitoring i-01b12315cb7c8 a running t2.micro b us-east-1f cOntainer Linux O-AutogenByAW3	r) Public DNS: ec: Tags Usage Inst 33be	2-34-204-168-217.con ructions	Public DNS Public DNS IPv4 Put IPv Private Secondary priva	ws.com IPv4) ec2-34-204-14 1.amazonaws bic IP 34.204.168.2* 6 IPs - DNS ip-172-30-5-70 ie IPs 172.30.5.70	58-217.comp .com 17 0.ec2.interna	ute-			-

Figure 2-5. Private IP

Run the following command to initialize Docker Swarm mode.

```
docker swarm init --advertise-addr 172.30.5.70
```

As the output in the following listing indicates, Swarm is initialized and the current node is a manager node. The command to add a worker node is also included in the output. The command to obtain the command to add a manager node is also output. Copy the docker swarm join command to add a worker node to the Swarm.

```
core@ip-172-30-5-70 ~ $ docker swarm init --advertise-addr 172.30.5.70
Swarm initialized: current node (bgzqx2cfsf05qdradxytmdcp3) is now a manager.
To add a worker to this swarm, run the following command:
    docker swarm join \
        --token SWMTKN-1-303zi1rxgkzy5gq5itr580yp9pbagxnkelinzh42ovrb7znt6f-
    dmgeg3veppor942vsavma3s47 \
        172.30.5.70:2377
To add a manager to this swarm, run 'docker swarm join-token manager' and follow the
    instructions.
```

Run the docker info command to get system-wide information about the Docker Engine. The command outputs the total number of Docker containers that are running, paused, or stopped; partial output is listed.

```
core@ip-172-30-5-70 ~ $ docker info
Containers: 0
 Running: 0
 Paused: 0
 Stopped: 0
Images: 0
Server Version: 1.12.6
Storage Driver: overlay
 Backing Filesystem: extfs
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: null host bridge overlay
Swarm: active
 NodeID: bgzqx2cfsf05qdradxytmdcp3
 Is Manager: true
 ClusterID: 056zm05kk6em6u7vlki8pbhc9
 Managers: 1
 Nodes: 1
CPUs: 1
Total Memory: 994.6 MiB
Name: ip-172-30-5-70.ec2.internal
Docker Root Dir: /var/lib/docker
```

The Storage Driver is overlay and the backing filesystem is extfs. The logging driver is json-file, which is covered in Chapter 11 on logging. The Swarm is shown to be active. Information about the node such as NodeID, whether the node is a manager, the number of managers in the Swarm, and the number of nodes in the Swarm, is also listed.

The resource capacity (CPU and memory) of the node is also listed. Chapter 7 discusses more about resource usage. The node name is the private DNS of the EC2 instance on which the Swarm is initialized.

List the nodes in the Swarm with the following command:

docker node ls

A single node gets listed including the node ID, which is the only unique parameter for a node. The hostname is also unique if a node has not been made to leave the Swarm and rejoined.

core@ip-172-30-5-70 ~ \$ docker node ls ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS bgzqx2cfsf05qdradxytmdcp3 * ip-172-30-5-70.ec2.internal Ready Active Leader

The * after the node ID indicates that this is the current node. The nodes in the Swarm also have a STATUS, AVAILABILITY, and MANAGER STATUS columns. STATUS can be one of the values listed in Table 2-2.

Table 2-2. Node Status

Status	Description
Ready	Ready for use
Down	Not ready for use
Unknown	Not known

AVAILABILITY can be one of the values listed in Table 2-3.

Table 2-3. AVAILABILITY Column

Availability	Description
Active	Scheduler may assign tasks to the node.
Pause	Scheduler does not assign new tasks to the node but existing tasks keep running.
Drain	Scheduler does not assign new tasks to the node and existing tasks are shut down. Replacement tasks are started on other nodes.

MANAGER STATUS can be one of the values listed in Table 2-4. If the MANAGER STATUS column has no value, it indicates a worker node.

CHAPTER 2 USING DOCKER IN SWARM MODE

Table 2-4.	Manager	Status
------------	---------	--------

Manager Status	Description
Reachable	The node participates in the Raft consensus quorum and, if the leader becomes unavailable, the node is eligible to be made the leader node.
Unreachable	The node was a manager node that was reachable but has become unreachable and is not able to communicate with the other manager nodes in the Swarm. An unreachable manager node could be made reachable though not guaranteed to be restored by doing one of the following:
	-Restart the machine
	-Restart the daemon
	If neither of the preceding restores a unreachable manager node, the following should be implemented. Demote and remove the failed node.
	docker node demote <node> and docker node rm <id-node></id-node></node>
	Add another manager node with docker swarm join. Or Promote a worker node to manager node with docker node promote
Leader	Primary manager node that performs all the Swarm management and orchestration.

Joining Nodes to the Swarm

Additional nodes, manager or worker, may be added or joined to the Swarm as required. By default, manager nodes are also worker nodes but not vice versa. The manager nodes are added for a different reason than the worker nodes. The manager nodes are added to make the Swarm more fault tolerant and the worker nodes are added to add capacity to the Swarm. The commands to add manager and worker nodes are also different. The command to add a worker node is output when the Swarm is initialized. The command to add a worker node is output when the Swarm is initialized. The command to add a worker node is not worker node.

docker swarm join-token worker

The command to add a manager node may be found using the following command.

docker swarm join-token manager

A reason for adding a worker node is that the service tasks scheduled on some of the nodes are not running and are in Allocated state. A reason for adding a manager node is that another manager node has become unreachable.

The node to join, manager or worker, must have Docker Engine version at least 1.12 installed. Next, you add two worker nodes. Obtain the public IP address of an EC2 instance started for a worker node. SSH login to the worker instance.

ssh -i "coreos.pem" core@34.204.199.

Run the docker swarm join command, which has the following syntax, to join the node to the Swarm as a worker node.

```
docker swarm join [OPTIONS] HOST:PORT
```

The options supported by the docker swarm join command are listed in Table 2-5.

Table 2-5. Options for docker swarm join Command

Option	Description	Default Value
advertise-addr	Advertised address in format <ip interface>[:port].</ip interface>	
availability	Availability of the node. One of active/pause/drain.	active
listen-addr	Listen address in format <ip interface>[:port].</ip interface>	0.0.0.0:2377
token	Token for entry into the Swarm.	

Run the docker swarm join command output during the initialization of the Swarm mode to join the worker instance with the Swarm. As the output message indicates, "The node joined the Swarm as a worker."

```
[root@localhost ~]# ssh -i "coreos.pem" core@34.204.199.45
Container Linux by CoreOS stable (1409.7.0)
core@ip-172-30-5-31 ~ $ docker swarm join \
> --token SWMTKN-1-303zi1rxgkzy5gq5itr580yp9pbagxnkelinzh42ovrb7znt6f-
dmgeg3veppor942vsavma3s47 \
> 172.30.5.70:2377
This node joined a swarm as a worker.
```

Similarly, SSH login to the other worker instance.

```
ssh -i "coreos.pem" core@34.231.70.10
```

Run the same docker swarm join command and the second nodes joins the Swarm as a worker node.

```
[root@localhost ~]# ssh -i "coreos.pem" core@34.231.70.10
Container Linux by CoreOS stable (1409.7.0)
core@ip-172-30-5-108 ~ $ docker swarm join \
> --token SWMTKN-1-303zi1rxgkzy5gq5itr580yp9pbagxnkelinzh42ovrb7znt6f-
dmgeg3veppor942vsavma3s47 \
> 172.30.5.70:2377
This node joined a swarm as a worker.
```

The following sequence of events takes place when the docker swarm join command runs to join a worker node to the Swarm.

- 1. The Swarm mode for the Docker Engine on the node is enabled.
- 2. A request for a TLS certificate is sent to the manager.
- 3. The node is named with the machine hostname.
- 4. The current node joins the Swarm at the manager listen address. Based on the token, the node is joined as a worker node or a manager node.

- 5. Sets the current node to Active availability.
- 6. The ingress overlay network is extended to the current node.

When a node is joined to the Swarm using the manager token, the node joins as a manager node. The new manager nodes should be Reachable and only the first manager node is the leader. Leader election to a different manager node occurs only if the initial leader node were to fail or be demoted.

The worker nodes differ from the manager nodes in another regard. A worker node cannot be used to view or modify the cluster state. Only the manager node can be used to view the cluster state such as the nodes in the Swarm. Only the manager node can be used to modify a cluster state such as remove a node. If the docker node ls command is run on a worker node, the following error message is generated.

```
core@ip-172-30-5-31 ~ $ docker node ls
```

Error response from daemon: This node is not a swarm manager. Worker nodes can't be used to view or modify cluster state. Please run this command on a manager node or promote the current node to a manager.

Testing the Swarm

Next, you deploy a simple Hello World service to the Swarm to test the cluster. List the nodes in the Swarm from the manager node with the following command.

docker node ls

The three nodes should be listed.

core@ip-172-30-5-70 ~ \$ doc	ker node ls			
ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS
9n5qmj4pp91f0n3s0n2jwjdv8	ip-172-30-5-108.ec2.internal	Ready	Active	
<pre>bgzqx2cfsf05qdradxytmdcp3 *</pre>	ip-172-30-5-70.ec2.internal	Ready	Active	Leader
bqq4bryuobylu0glm4p19tko4	ip-172-30-5-31.ec2.internal	Ready	Active	

How do you tell if a node is a manager node or a worker node? From the Manager Status column. If the Manager Status is empty, the node is a worker node and if the Manager Status has a value, which would be one of the values discussed in Table 2-4, the node is a manager node. Two worker nodes and one manager node are listed.

We already discussed that worker nodes can't be used to view or modify cluster state. Next, create a Docker service using the docker service create command, which becomes available only if the Swarm mode is enabled. Using Docker image alpine, which is a Linux distribution, create two replicas and ping the docker.com domain from the service containers.

docker service create --replicas 2 --name helloworld alpine ping docker.com

If the preceding command runs without an error, the Docker Swarm installed fine. The command returns the service ID.

```
core@ip-172-30-5-70 \sim $ docker service create --replicas 2 --name helloworld alpine ping docker.com bkwskfzqa173dp55j54erg5cg
```

Services may be listed with the following command.

```
docker service ls
```

The service helloworld is listed and the number of replicas is listed as 2/2, which implies that two replicas exist and meet the desired state of two replicas. The REPLICAS column output is ordered "actual/ desired". The Docker image is alpine and the command to run the service is ping docker.com.

core@ip-172-30-!	5-70 ~ \$ dockei	service ls		
ID	NAME	REPLICAS	IMAGE	COMMAND
bkwskfzqa173	helloworld	2/2	alpine	<pre>ping docker.com</pre>

The docker service inspect command is used to find more information about the service.

```
docker service inspect helloworld
```

The detailed information about the helloworld service—including the container spec, resources, restart policy, placement, mode, update config, and update status—is listed.

```
core@ip-172-30-5-70 ~ $ docker service inspect helloworld
[
    {
        "ID": "bkwskfzqa173dp55j54erg5cg",
        "Version": {
            "Index": 22
        },
        "CreatedAt": "2017-07-22T19:11:50.345823466Z",
        "UpdatedAt": "2017-07-22T19:11:50.345823466Z",
        "Spec": {
            "Name": "helloworld",
            "TaskTemplate": {
                 "ContainerSpec": {
                     "Image": "alpine",
                     "Args": [
                         "ping",
                         "docker.com"
                     1
                },
                 "Resources": {
                     "Limits": {},
                     "Reservations": {}
                },
                "RestartPolicy": {
                     "Condition": "any",
                     "MaxAttempts": 0
                },
                "Placement": {}
            },
             "Mode": {
                "Replicated": {
                     "Replicas": 2
                }
            },
```
```
"UpdateConfig": {
                 "Parallelism": 1,
                 "FailureAction": "pause"
            },
             "EndpointSpec": {
                 "Mode": "vip"
            }
        },
        "Endpoint": {
             "Spec": {}
        },
        "UpdateStatus": {
             "StartedAt": "0001-01-01T00:00:00Z",
             "CompletedAt": "0001-01-01T00:00:00Z"
        }
    }
]
```

The replicas and the nodes on which the replicas are placed may be listed with the following command syntax.

docker service ps <SERVICE

The <SERVICE> placeholder is either a service name (like helloworld) or the actual service ID (like bkwskfzqa173 for this example). For the helloworld service, the command becomes:

```
docker service ps helloworld
```

The preceding command also lists the node on which a replica is running. The Docker containers started for a service are listed with same command as before, the docker ps command.

```
core@ip-172-30-5-70 ~ $ docker service ps helloworld
ID
                           NAME
                                         IMAGE
                                                 NODE
                                                                                DESIRED STATE
CURRENT STATE
                        ERROR
2x8gqd2qbylpkug1kg0pxi1c2 helloworld.1 alpine ip-172-30-5-70.ec2.internal
                                                                                Running
Running 34 seconds ago
6twq1v0lr2gflnb6ae19hrpx9 helloworld.2 alpine ip-172-30-5-108.ec2.internal Running
Running 34 seconds ago
core@ip-172-30-5-70 ~ $ docker ps
CONTAINER ID
                   IMAGE
                                      COMMAND
                                                         CREATED
                                                                            STATUS
PORTS
                   NAMES
acbdaccad6ea
                   alpine:latest
                                      "ping docker.com"
                                                         47 seconds ago
                                                                            Up 46 seconds
helloworld.1.2x8gqd2qbylpkug1kg0pxi1c2
```

The docker ps command is not a Swarm mode command, but may be run on the worker nodes to find the service containers running on a worker node. The docker ps command gives you all containers running on a node, even if they are not service containers.

CHAPTER 2 USING DOCKER IN SWARM MODE

core@ip-172-30-5-108 ~ \$ docker ps CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES 74ea31054fb4 alpine:latest "ping docker.com" About a minute ago Up About a minute helloworld.2.6twq1v0lr2gflnb6ae19hrpx9

Only two nodes are listed by the docker service ps helloworld command on which replicas are scheduled, the manager node and one of the worker nodes. The docker ps command on the other worker node does not list any Docker containers.

```
core@ip-172-30-5-31 ~ $ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
```

The docker node inspect <node> command is used to get detailed information about a node, such as the node role, availability, hostname, resources capacity, plugins, and status.

```
core@ip-172-30-5-70 ~ $ docker node inspect ip-172-30-5-70.ec2.internal
[
    {
        "ID": "bgzqx2cfsf05qdradxytmdcp3",
        "Version": {
            "Index": 10
        },
        "CreatedAt": "2017-07-22T19:09:45.647701768Z",
        "UpdatedAt": "2017-07-22T19:09:45.68030039Z",
        "Spec": {
            "Role": "manager",
            "Availability": "active"
        },
        "Description": {
            "Hostname": "ip-172-30-5-70.ec2.internal",
            "Platform": {
                 "Architecture": "x86 64",
                 "OS": "linux"
            },
             "Resources": {
                 "NanoCPUs": 100000000,
                 "MemoryBytes": 1042935808
            },
"Engine": {
    "Cogine"
                 "EngineVersion": "1.12.6",
                 "Plugins": [
                     {
                         "Type": "Network",
                         "Name": "bridge"
                     },
                     ł
                         "Type": "Network",
                         "Name": "host"
                     },
```

```
{
                          "Type": "Network",
                          "Name": "null"
                     },
                     {
                          "Type": "Network",
                          "Name": "overlay"
                     },
                     {
                          "Type": "Volume",
                          "Name": "local"
                     }
                 ]
             }
        },
         'Status": {
             "State": "ready"
        },
        "ManagerStatus": {
             "Leader": true,
             "Reachability": "reachable",
             "Addr": "172.30.5.70:2377"
        }
    }
]
```

A service may be removed with the docker service rm <service> command. Subsequently, the docker service inspect <service> command should not list any replicas and running docker ps will show no more running Docker containers.

```
core@ip-172-30-5-70 ~ $ docker service rm helloworld
helloworld
core@ip-172-30-5-70 ~ $ docker service inspect helloworld
[]
Error: no such service: helloworld
```

Chapter 4 discusses more about services.

Promoting a Worker Node to Manager

As mentioned before, a manager node is also a worker node by default, but a worker node is only a worker node. But a worker node may be promoted to a manager node. The Docker command to promote one or more worker nodes to a manager node has the following syntax.

```
docker node promote NODE [NODE...]
```

The command must be run from the leader node. As an example, promote the node ip-172-30-5-108. ec2.internal. As the output indicates, the node gets promoted to a manager node. Subsequently list the nodes in the Swarm and the node promoted should have manager status as Reachable.

A worker node should preferably be promoted using the node ID; the reason for which is discussed subsequently. Promote another worker node using the node ID. Subsequently, both the worker nodes are listed as Reachable in the Manager Status column.

```
core@ip-172-30-5-70 ~ $ docker node promote ip-172-30-5-108.ec2.internal
Node ip-172-30-5-108.ec2.internal promoted to a manager in the swarm.
core@ip-172-30-5-70 ~ $ docker node ls
                                                          STATUS AVAILABILITY MANAGER STATUS
ID
                            HOSTNAME
9n5qmj4pp91f0n3s0n2jwjdv8
                            ip-172-30-5-108.ec2.internal Ready
                                                                               Reachable
                                                                 Active
bgzqx2cfsf05qdradxytmdcp3 * ip-172-30-5-70.ec2.internal
                                                          Ready
                                                                 Active
                                                                               Leader
bqq4bryuobylu0glm4p19tko4
                            ip-172-30-5-31.ec2.internal
                                                          Readv
                                                                 Active
```

Demoting a Manager Node to Worker

A manager node may be demoted to a worker node with the following Docker command.

docker node demote NODE [NODE...]

Any manager node, including the leader node, may be demoted. As an example, demote the manager node ip-172-30-5-108.ec2.internal.

```
core@ip-172-30-5-70 ~ $ docker node demote ip-172-30-5-108.ec2.internal
Manager ip-172-30-5-108.ec2.internal demoted in the swarm.
```

Once demoted, the commands such as docker node 1s that can be run only from a manager node cannot be run any more on the node. The docker node 1s command lists the demoted node as a worker node; no MANAGER STATUS is listed for a worker node.

core@ip-172-30-5-70 ~ \$ docker node ls						
ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS		
9n5qmj4pp91f0n3s0n2jwjdv8	ip-172-30-5-108.ec2.internal	Ready	Active			
<pre>bgzqx2cfsf05qdradxytmdcp3 *</pre>	ip-172-30-5-70.ec2.internal	Ready	Active	Leader		
bqq4bryuobylu0glm4p19tko4	ip-172-30-5-31.ec2.internal	Ready	Active			

A node should be preferably promoted/demoted and otherwise referred to in any command that is directed at the node using the node ID, which is unique to a node. The reason being that a demoted node, if promoted back, could be added with a different node ID and the docker node 1s command could list two node IDs for the same hostname. If the hostname is used to refer to a node, it could result in the node is ambiguous error message.

Making a Worker Node Leave the Swarm

Earlier you joined a node to the Swarm as a worker node. A worker node may also be made to leave the Swarm. As an example, make one of the worker nodes leave with the following command, which must be run from the node you want to remove from the Swarm.

docker swarm leave

As the message output indicates, the node has left the Swarm.

```
core@ip-172-30-5-31 \sim $ docker swarm leave Node left the swarm.
```

Similarly, make the other worker node leave the Swarm.

core@ip-172-30-5-108 \sim \$ docker swarm leave Node left the swarm.

After a worker node has left the Swarm, the node itself is not removed and continues to be listed with the docker node 1s command with a Down status.

core@ip-172-30-5-70 ~ \$ docker node lsIDHOSTNAMESTATUSAVAILABILITYMANAGER STATUS9n5qmj4pp91f0n3s0n2jwjdv8ip-172-30-5-108.ec2.internalDownActivebgzqx2cfsf05qdradxytmdcp3 *ip-172-30-5-70.ec2.internalReadyActiveLeaderbqq4bryuobylu0glm4p19tko4ip-172-30-5-31.ec2.internalDownActive

Making a Manager Node Leave the Swarm

While it is easier to make a worker node leave the Swarm, it is different when a manager node must leave the Swarm. Making a worker node leave the Swarm only lowers the Swarm capacity in terms of the service tasks that may be scheduled in the Swarm. But making a manager node leave the Swarm makes the Swarm less available. If the fault tolerance does not allow for a manager node to fail or be removed from the Swarm, the same docker swarm leave command that made a worker node leave the Swarm cannot be used to make a manager node leave the Swarm. If a Swarm has only one manager node, the docker swarm leave command generates the following error message.

```
core@ip-172-30-5-70 \sim $ docker swarm leave
Error response from daemon: You are attempting to leave the swarm on a node that is
participating as a manager. Removing the last manager erases all current state of the
swarm. Use `--force` to ignore this message.
```

Add the --force option to the docker swarm leave command on the manager node to cause the manager node to leave the Swarm.

```
core@ip-172-30-5-70 \sim $ docker swarm leave --force Node left the swarm.
```

If the only manager node is removed, the Swarm no longer exists. The Swarm must be initialized again if the Swarm mode is to be used.

```
core@ip-172-30-5-70 ~ $ docker swarm init --advertise-addr 172.30.5.70
Swarm initialized: current node (cnyc2w3n8q8zuxjujcd2s729k) is now a manager.
To add a worker to this swarm, run the following command:
    docker swarm join \
        --token SWMTKN-1-4lxmisvlszjgck4ly0swsxubejfxOphlne1xegho2fiq99amqf-
        11mpscd8gs6bsayzren8fa2ki \
        172.30.5.70:2377
To add a manager to this swarm, run 'docker swarm join-token manager' and follow the
    instructions.
```

A new Swarm is created with only the manager node and the Swarm has only one node initially.

core@ip-172-30-5-70 ~ \$ docker node ls					
ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS	
cnyc2w3n8q8zuxjujcd2s729k *	ip-172-30-5-70.ec2.internal	Ready	Active	Leader	

If a Swarm has two manager nodes, making one of the manager nodes leave the Swarm has a different effect. With two managers, the fault tolerance is 0, as discussed earlier. To create a Swarm with two manager nodes, start with a Swarm that has one manager node and two worker nodes.

core@ip-172-30-5-70 ~ \$ docker node ls					
ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS	
4z03hudbo3fz17q94leo24pvh	ip-172-30-5-108.ec2.internal	Ready	Active		
<pre>cnyc2w3n8q8zuxjujcd2s729k *</pre>	ip-172-30-5-70.ec2.internal	Ready	Active	Leader	
efsxwt43iskasa6poh2stkjeb	ip-172-30-5-31.ec2.internal	Ready	Active		

Promote one of the worker nodes to a manager node.

core@ip-172-30-5-70 ~ \$ docker node promote ip-172-30-5-108.ec2.internal Node ip-172-30-5-108.ec2.internal promoted to a manager in the swarm.

The Swarm will then have two manager nodes.

core@ip-172-30-5-70 ~ \$ docker node ls						
ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS		
4z03hudbo3fz17q94leo24pvh	ip-172-30-5-108.ec2.internal	Ready	Active	Reachable		
<pre>cnyc2w3n8q8zuxjujcd2s729k *</pre>	ip-172-30-5-70.ec2.internal	Ready	Active	Leader		
efsxwt43iskasa6poh2stkjeb	ip-172-30-5-31.ec2.internal	Ready	Active			

Run the docker swarm leave command from a manager node that's not the leader node. The following message is generated.

core@ip-172-30-5-108 ~ \$ docker swarm leave

The error response from the daemon is as follows:

You are attempting to leave the swarm on a node that is participating as a manager.

Removing this node leaves one manager out of two. Without a Raft quorum, your Swarm will be inaccessible. The only way to restore a Swarm that has lost consensus is to reinitialize it with --force-new-cluster. Use --force to suppress this message.

To make the manager node leave, you must add the --force option to the command.

core@ip-172-30-5-108 ~ \$ docker swarm leave --force Node left the swarm. When one of the two managers has left the Swarm, the Raft quorum is lost and the Swarm becomes inaccessible. As indicated, the Swarm must be reinitialized using the --force-new-cluster option.

Reinitializing a Cluster

A Swarm that has lost quorum cannot be reinitialized using the command used to initialize a Swarm. If the same command runs on a Swarm that has lost quorum, a message indicates that the node is already in the Swarm and first must be made to leave the Swarm:

```
core@ip-172-30-5-70 \sim $ docker swarm init --advertise-addr 172.30.5.70
Error response from daemon: This node is already part of a swarm. Use "docker swarm leave" to leave this swarm and join another one.
```

```
To reinitialize the Swarm the --force-new-cluster option must be added to the docker swarm
init command. core@ip-172-30-5-70 ~ $ docker swarm init --advertise-addr 172.30.5.70
--force-new-cluster
Swarm initialized: current node (cnyc2w3n8q8zuxjujcd2s729k) is now a manager.
To add a worker to this swarm, run the following command:
    docker swarm join \
    --token SWMTKN-1-4lxmisvlszjgck4ly0swsxubejfx0phlne1xegho2fiq99amqf-
    11mpscd8gs6bsayzren8fa2ki \
    172.30.5.70:2377
To add a manager to this swarm, run 'docker swarm join-token manager' and follow the
    instructions.
```

The Swarm is reinitialized and the docker swarm join command to add a worker node is output.

Modifying Node Availability

The availability of a node may be modified with the D command with the --availability option. One of the --availability options shown in Table 2-6 may be set.

Table 2-6. Ava	lability Options
----------------	------------------

Availability Option	Description
active	Restores a paused or drained node to active.
pause	Pauses a node so that it is not available to receive new tasks.
drain	With a worker node, the node becomes down and unavailable for scheduling new tasks. A manager node also becomes unavailable for scheduling new tasks but continues to perform Swarm management.

As an example, you can drain a worker node as follows.

core@ip-172-30-5-70 ~ \$ docker node update --availability drain ip-172-30-5-108.ec2.internal ip-172-30-5-108.ec2.internal The worker node is drained. All service tasks on the drained node are shut down and started on other nodes that are available. The output from the docker node ls command lists the node with the status set to Drain.

```
core@ip-172-30-5-70 ~ $ docker node ls
ID
                           HOSTNAME
                                                         STATUS AVAILABILITY MANAGER STATUS
bhuzgyqvb83dx0zvms54o0a58
                            ip-172-30-5-108.ec2.internal Ready
                                                                 Drain
cnyc2w3n8q8zuxjujcd2s729k * ip-172-30-5-70.ec2.internal
                                                                 Active
                                                         Ready
                                                                              Leader
efsxwt43iskasa6poh2stkjeb
                           ip-172-30-5-31.ec2.internal
                                                         Ready
                                                                 Active
The node detail (partial output is listed) for the drained worker node lists the node
availability as "drain".core@ip-172-30-5-70 ~ $ docker node inspect ip-172-30-5-108.ec2.
internal
[
    {
        "ID": "bhuzgyqvb83dx0zvms54o0a58",
        "Version": {
            "Index": 49
        },
        "CreatedAt": "2017-07-22T19:30:31.544403951Z",
        "UpdatedAt": "2017-07-22T19:33:37.45659544Z",
        "Spec": {
            "Role": "worker",
            "Availability": "drain"
        },
        "Description": {
            "Hostname": "ip-172-30-5-108.ec2.internal",
```

All service tasks on the drained node are shut down and started on other nodes that are available. The node availability with the docker node ls is listed as Drain.

A drained node can be made active again using the docker node update command with --availability set to Active.

```
core@ip-172-30-5-70 ~ $ docker node update --availability active ip-172-30-5-108.ec2.internal
ip-172-30-5-108.ec2.internal
```

The drained node becomes active and is listed with the status set to Active.

core@ip-172-30-5-70 ~ \$ docker node ls					
ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS	
bhuzgyqvb83dx0zvms54o0a58	ip-172-30-5-108.ec2.internal	Ready	Active		
cnyc2w3n8q8zuxjujcd2s729k *	ip-172-30-5-70.ec2.internal	Ready	Active	Leader	
efsxwt43iskasa6poh2stkjeb	ip-172-30-5-31.ec2.internal	Ready	Active		

Removing a Node

One or more nodes may be removed from the Swarm using the docker node rm command, which is run from any manager node.

```
docker node rm [OPTIONS] NODE [NODE...]
```

The difference between docker swarm leave and docker node rm is that the docker node rm may be run only from a manager node. A demoted node can only be removed from the Swarm with the docker node rm command. The sequence to remove a manager node without using the --force option is the following.

- 1. Demote the manager node, which makes it a worker node.
- 2. Drain the worker node.
- 3. Make the worker node leave the Swarm.
- 4. Remove the node.

Summary

This chapter discussed using Docker in Swarm mode. First, you initialized the Swarm mode with the docker swarm init command to make the current node the manager node in the Swarm. Subsequently, you joined worker nodes to the Swarm with the docker swarm join command. The chapter also discussed promoting a worker node to a manager node/demoting a manager node to a worker node, making a worker node leave a Swarm and then rejoin the Swarm, making a manager node leave a Swarm, reinitializing a Swarm, and modifying node availability and removing a node. The next chapter introduces Docker for AWS, which is a managed service for Docker Swarm mode.

CHAPTER 3

Using Docker for AWS to Create a Multi-Zone Swarm

Docker Swarm is provisioned by first initiating a Swarm to create a manager node and subsequently joining worker nodes to that manager node. Docker Swarm provides distributed service deployment for Docker applications.

The Problem

By default, a Docker Swarm is provisioned on a single zone on AWS, as illustrated in Figure 3-1. With the manager nodes and all the worker nodes in the same AWS zone, failure of the zone would make the zone unavailable. A single-zone Swarm is not a highly available Swarm and has no fault tolerance.



Figure 3-1. A single-zone Swarm

The Solution

Docker and AWS have partnered to create a Docker for AWS deployment platform that provisions a Docker Swarm across multiple zones on AWS. Docker for AWS does not require users to run any commands on a command line and is graphical user interface (GUI) based. With manager and worker nodes in multiple zones, failure of a single AWS zone does not make the Swarm unavailable, as illustrated in Figure 3-2. Docker for AWS provides fault tolerance to a Swarm.



Figure 3-2. A Multi-zone Swarm

Docker for AWS is a managed service for Docker Swarm on the AWS cloud platform. In addition to multiple zones, Docker for AWS has several other benefits:

- All the required infrastructure is provisioned automatically.
- Automatic upgrade to new software versions without service interruption.
- A custom Linux distribution optimized for Docker. The custom Linux distribution is not available separately on AWS and uses the overlay2 storage driver.
- Unused Docker resources are pruned automatically.
- Auto-scaling groups for managing nodes.

- Log rotation native to the host to avoid chatty logs consuming all the disk space.
- Centralized logging with AWS CloudWatch.
- A bug-reporting tool based on a docker-diagnose script.

Two editions of Docker for Swarm are available:

- Docker Enterprise Edition (EE) for AWS
- Docker Community Edition (CE) for AWS

We use the Docker Community Edition (CE) for AWS in this chapter to create a multi-zone Swarm. This chapter includes the following topics:

- Setting the environment
- Creating a AWS CloudFormation stack for the Docker Swarm
- Connecting with the Swarm manager
- Using the Swarm
- Deleting the Swarm

Setting the Environment

Two deployment options are available with Docker for AWS.

- Use a pre-existing VPC
- Use a new VPC created by Docker

Letting Docker create the VPC, subnets, and gateways is the easier option and the one used in this chapter.

Create an AWS account if you don't already have one at https://aws.amazon.com/resources/createaccount/. The AWS account must support EC2-VPC. Even though AWS services such as VPC are created automatically, the account must have permissions to create EC2 instances, including auto-scaling groups, IAM profiles, DynamoDB tables, SQS Queue, VPC (including subnets, gateways, and security groups), Elastic Load Balancer, and CloudWatch Log Group. The only user input other than creating an account with the required permissions is to create a SSH key pair in the AWS region in the Docker Swarm.

Select the EC2 AWS service and click on the Key Pairs link in the EC2 dashboard. Click on Create Key Pair to create and download a key pair. Specify a key pair name (docker for example) in the Create Key Pair dialog and click on Create. A key pair gets created, as shown in Figure 3-3. Copy the key pair file (docker.pem) to a local Linux machine.

Jervices +	Resource Groups V R	C Deepak voina + Onio + Support +
Elastic IPs	Create Key Pair Import Key Pair Delete	근 🕈 🔞
Rey Pairs	Q. Filter by attributes or search by keyword	② K ≤ 1 to 2 of 2 > >
Network Interfaces	Key pair name • Fingerprint	-
LOAD BALANCING	coreos 5f:18:ce:08:2b:2e:a9:d2:7f:7a	a:a5:6c:0e:38:07:e7:74:18:72:a1
Target Groups	docker 3e:a5:ea:08:30:40:b3:73:04:7	7c:15:a0:35:2c:d6:d4:4e:41:0c:44

Figure 3-3. A key pair

Set the permissions on the docker.pem to 400, which gives only read permissions and removes all other permissions.

chmod 400 docker.pem

Creating a AWS CloudFormation Stack for Docker Swarm

Navigate to https://docs.docker.com/docker-for-aws/ in a web browser and click on the Deploy Docker for AWS option, as as shown in Figure 3-4. The label could be different, such as Deploy Docker Community Edition [CE] for AWS [stable].



Figure 3-4. Deploy Docker for AWS

The Create Stack wizard is started with the provision to either design a new template or choose the default CloudFormation template for Docker on AWS. Select the Specify an Amazon S3 Template URL option for which a URL is pre-specified, as shown in Figure 3-5. Click on Next.

Create stack				
Select Template Specify Details	Select Template			
Options Review	Select the template that descrit	bes the stack that you want to create. A stack is a group of related res	ources that you manage as a single unit.	
	Design a template	Use AWS CloudFormation Designer to create or modify an existing to Design template	template. Learn more.	
	Choose a template	A template is a JSONY/AML-formatted text file that describes your s Select a sample template	tack's resources and their properties. Learn more.	
		Upload a template to Amazon S3 Choose File No file chosen Specify an Amazon S3 template URL		
		https://editions-us-east-1.s3.amazonaws.com/aws/stable/Docker.tr	View/Edit template in Designer	
			Cancel	ext 占

Figure 3-5. Selecting a template

In Specify Details, specify a stack name (DockerSwarm). The Swarm Parameters section has the fields listed in Table 3-1.

Table 3-1. Swarm Parameters

Parameter	Description
Number of Swarm managers?	Number of Swarm manager nodes. Valid values are 1, 3, and 5.
Number of Swarm worker nodes?	Number of worker nodes in the Swarm (0-1000).

Keep the default settings of 3 for Number of Swarm Managers and 5 for Number of Swarm Worker nodes, as shown in Figure 3-6.

CHAPTER 3 USING DOCKER FOR AWS TO CREATE A MULTI-ZONE SWARM

Create stack			
Select Template Specify Details Options Review	Specify Details		
	Specify a stack name and para	meter values. You can use or change the defaul	t parameter values, which are defined in the AWS CloudFormation template. Learn more.
	Stack name	DockerSwarm	
	Parameters		
	Swarm Size		
	Number of Swarm managers?	3 1 3	Number of Swarm manager nodes (1, 3, 5)
	Number of Swarm worker nodes?	5 v.	Number of worker nodes in the Swarm (0-1000).
	Swarm Properties		
	Which SSH key to use?	Search - Name of an existing FC2 KeyPair to enable SSH acc	es to the instances

Figure 3-6. Specifying a stack name

Next, specify the Swarm properties, as discussed in Table 3-2.

Table 3-2. Swarm Properties

Swarm Property	Description	Value Set
Which SSH key to use?	Name of an existing EC2 key pair to enable SSH access to the instances.	docker
Enable daily resource cleanup?	Cleans up unused images, containers, networks, and volumes.	no
Use CloudWatch for container logging?	Send all container logs to CloudWatch.	yes

In the Which SSH key to use? property, select the docker SSH key. The Swarm properties are shown in Figure 3-7.

Swarm Size		
Number of Swarm managers?	3	Number of Swarm manager nodes (1, 3, 5)
Number of Swarm worker nodes?	5	Number of worker nodes in the Swarm (0-1000).
Swarm Properties		
Which SSH key to use?	docker Name of an existing EC2 Key	vPair to enable SSH access to the instances
Enable daily resource cleanup?	no	Cleans up unused images, containers, networks and volume
Use Cloudwatch for container logging?	yes	Send all Container logs to CloudWatch

Figure 3-7. Swarm properties

Specify the Swarm Manager properties, as discussed in Table 3-3.

Swarm Property	Description	Value Set
Swarm manager instance type?	EC2 HVM instance type (t2.micro, m3.medium, etc.)	t2.micro
Manager ephemeral storage volume size?	Size of manager's ephemeral storage volume in GB	20
Manager ephemeral storage volume type?	Manager volume type	standard

The Swarm Manager properties are as shown in Figure 3-8. Specify the Swarm Worker properties, as discussed in Table 3-4.

Table 3-4. Swarm Worker Properties

Swarm Worker Property	Description	Value Set
Agent worker instance type?	EC2 HVM instance type (t2.micro, m3.medium, etc.)	t2.micro
Worker ephemeral storage volume size?	Size of worker's ephemeral storage volume in GB	20
Worker ephemeral storage volume type?	Worker volume type	standard

The Swarm Worker properties are shown in Figure 3-8. Click on Next.

Swarm Manager Proper	ties		
Swarm manager instance type?	t2.micro	*	EC2 HVM instance type (t2.micro, m3.medium, etc).
Manager ephemeral storage volume size?	20		Size of Manager's ephemeral storage volume in GiB
Manager ephemeral storage volume type	standard	•	Manager ephemeral storage volume type
Swarm Worker Propertie	es		
Agent worker instance type?	t2.micro	v	EC2 HVM instance type (t2.micro, m3.medium, etc).
Worker ephemeral storage volume size?	20		Size of Workers's ephemeral storage volume in GiB
Worker ephemeral storage volume type	standard	×	Worker ephemeral storage volume type
			Cancel Previous Next

Figure 3-8. Swarm worker properties

Next, specify the options for the stack. Tags (key-value pairs) may be specified for resources in a stack. For permissions, an IAM role for CloudFormation may be chosen. None of these options is required to be set, as shown in Figure 3-9.

Create stack								
Select Template Specify Details	Options							
Options	Tags							
Review	You can specify tags (key-value pairs) for resources in your stack. You can add up to 50 unique key-value pairs for each stack. Learn more.							
	Key (127 characters max	dmum)		Value (255 characters maximum)				
	1							
	Permissions							
	You can choose an IAM role that CloudFormation uses to create, modify, or delete resources in the stack. If you don't choose a role, CloudFormation uses the permissions defined in your account. Learn more.							
	IAM Role	Choose a role (optional)	Ŧ					
		Enter role am						
	Advanced							
	rou can set additional options	ior your seach, inte fiotilication o	puons driu a si	ack policy, cearl mole.				

Figure 3-9. Optional settings

For Advanced options, the Notification options are set to No Notification. Set Rollback on Failure to Yes, as shown in Figure 3-10. Click on Next.

Advanced

You can set additional options for your stack, like notification options and a stack policy. Learn more.

Notification options	IS						
	No notification						
	New Amazon SNS topic						
	Торіс						
	Email						
	Existing Amazon SNS topic						
	Ŧ						
	Existing topic ARN						
Timeout 🚯	Minutes						
Rollback on failure 🚯	 Yes No 						



Figure 3-10. Setting rollback on failure

Review the stack settings, as shown in Figure 3-11.

CloudFormation ~	Stacks > Create Sta	ack				
Create stack						
Select Template Specify Details	Review					
Options	Template					
Review	Template URL Description Estimate cost	https://editions-us-east-1.s3.amazonaws.com/aws/stable/Docker.tmpl Docker CE for AWS 17.06.0-ce (17.06.0-ce-aws2) Link is not available				
	Details					
	Stack name:	DockerSwarm				
	Swarm Size					
	ManagerSize ClusterSize	3 5				
	Swarm Properties					
	KeyName EnableSystemPrune	docker				
	EnableCloudWatchLogs EnableCloudStorEfs	yes no				
	Swarm Manager Properties					

Figure 3-11. Reviewing the stack settings

Select the acknowledgement checkbox and then click on Create, as shown in Figure 3-12.

Options
Tags
No tags provided
Advanced
Notification Timeout none Rollback on failure Yes
Capabilities
1 The following resource(s) require capabilities: [AWS::IAM::Role]
This template contains Identity and Access Management (IAM) resources that might provide entities access to make changes to your AWS account. Check that you want to create each of these resources and that they have the minimum required permissions. Learn more.
Cancel Previous Creaters

Figure 3-12. Creating the stack

A new stack begins to be created. Click on the Refresh button to refresh the stacks listed, as shown in Figure 3-13.

1 CloudFormation V Stacks		
Create Stack Actions De	sign template	C O
Filter: Active By Stack Name		Showi Refresh ks
	Create a stack AWS CloudFormation allows you to quickly and easily deploy your infrastructure resources and applications on AWS. You can use one of the templates we provide to get started quickly with applications like WordPress or Drupal, one of the many sample templates or create your own template. You do not currently have any stacks. Choose Create new stack below to create a new AWS CloudFormation stack. Create new stack	Î
	Design a template Templates tell AWS CloudFormation which AWS resources to provision and how to provision them. When you create a CloudFormation stack, you must submit a template. To build and view templates, you can use the drag-and drop tool called AWS CloudFormation Designer Vou drag-and-drop the resources that you want to add to your template and drag	



A new stack based on a CloudFormation template for Docker Swarm starts to be created, as indicated by the status CREATE_IN_PROGRESS shown in Figure 3-14.

4	CloudFormation	· Stacks			
C	treate Stack 🔹 Action	Design template			C ¢
Fi	Iter: Active - By Stack I	Name			Showing 1 stack
	Stack Name	Created Time	Status	Description	
6	DockerSwarm	2017-07-22 14:59:31 UTC-0700	CREATE_IN_PROGRESS	Docker CE for AWS 17.06.0-ce (17.06.0-ce-aws2)	

Figure 3-14. CloudFormation stack status

The different tabs are provided for the different stack details. The Resources tab shows the AWS resources created by the CloudFormation template, as shown in Figure 3-15.

1 Clou	dFormation	Stack	(S								
Create Star	ck - A	ctions -	Design	emplate							C ¢
Filter: Act	we • By Sta	ick Name									Showing 1 stack
Stack	Name	Cre	ated Time		Statu	ıs		Description			
Docke	erSwarm	201	7-07-22 14	59:31 UTC-070	O CRE	ATE_IN_F	ROGRESS	Docker CE for AW	S 17.06.0-ce (17.06.0-ce-aw	\$2)	
			0								
Overview	Outputs	Resources	Events	Template	Parameters	Tags	Stack Polic	y Change Sets			886
Logical ID		Physical	ID				Туре		Status	Status Reason	-
AZInfo	2017/07/22/[\$LATEST]8598/668a7274ebbb1579bd3219a 3937		3219a	Custom: AZInfo	2	CREATE_COMPLETE					
AZInfoFund	ction	DockerSw	arm-AZInfo	Function-16CF	7771HOCF0		AWS::Lambda:	Function	CREATE_COMPLETE		
AttachGate	way	Docke-Att	ac-1VS1AX	18UO1WW	AWS::EC2::VPCGatewayAttach		CREATE_COMPLETE				
CloudstorE	BSPolicy	Docke-Clo	u-1NNAR8	CS6KXLI			AWS: IAM: Policy		CREATE_COMPLETE		
DockerLog	Group	DockerSw	rarm-lg				AWS::Logs::Log	gGroup	CREATE_COMPLETE		
DynDBPolicies Docke-DynD-7ZLQO1H9AY1I			AWS::IAM::Poli	cy	CREATE_COMPLETE						
DynDBWorkerPolicies Docke-DynD-1AC97BWW06VB A		AWS::IAM::Poli	cy	CREATE_COMPLETE							
ExternalLo	adBalancer	DockerSw	a-External-	1HBH91HC4D9	00		AWS::ElasticLo	adBalancing::Lo	CREATE_COMPLETE		

Figure 3-15. CloudFormation stack resources

The Events tab shows the events that occur in creating a CloudFormation stack, as shown in Figure 3-16.

1 Clou	dFormation	n v Stac	ks							
Create Star	:k 🔹 A	ctions -	Design to	emplate						C O
Filter: Act	ve • By Sta	sck Name								Showing 1 stack
Stack	Name	Cr	eated Time		Statu	ıs		Description		
Docke	rSwarm	20	17-07-22 14:	59:31 UTC-070	0 CRE	ATE_IN_P	ROGRESS	Docker CE for AWS 17.0	06.0-ce (17.06.0-ce-aws2)	
Overview	Outputs	Resources	Events	Template	Parameters	Tags	Stack Polic	Change Sets		888
2017-07-22		Status	L2	Тур	•			Logical ID	Status reason	·
• 15:02:12	UTC-0700	CREATE_IN_	PROGRESS	AWA	S::AutoScaling:	AutoScal	ingGroup	ManagerAsg	Resource creation Initiated	
15:02:11	UTC-0700	CREATE_IN_	PROGRESS	AWA	S::AutoScaling:	AutoScal	ingGroup	ManagerAsg		
15:02:11	UTC-0700	CREATE_CO	MPLETE	AWA	S::IAM::Instance	eProfile		WorkerInstanceProfi	le	
• 15:02:07	UTC-0700	CREATE_CO	MPLETE	AW	S::AutoScaling:	:LaunchC	onfiguration	ManagerLaunchCon aws2	fig17060ce	
• 15:02:06	UTC-0700	CREATE_IN_	PROGRESS	AW	S::AutoScaling:	LaunchC	onfiguration	ManagerLaunchCon aws2	fig17060ce Resource creation Initiated	
 15:02:06 	UTC-0700	CREATE_IN_	PROGRESS	AW	S::AutoScaling:	LaunchC	onfiguration	ManagerLaunchCon aws2	fig17060ce	
• 15:02:02	UTC-0700	CREATE_CO	MPLETE	AW	S::IAM::Instance	eProfile		ProxyInstanceProfile		
• 15:01:43	UTC-0700	CREATE_CO	MPLETE	AWA	S.: IAM .: Policy			DynDBWorkerPolicie	25	
• 15:01:43	UTC-0700	CREATE_IN	PROGRESS	AWA	S::IAM::Policy			DynDBWorkerPolicie	Resource creation Initiated	
• 15:01:38	UTC-0700	CREATE_IN_	PROGRESS	AW	S.: IAM .: Policy			DynDBWorkerPolicie	rs	

Figure 3-16. CloudFormation stack events

When the stack creation completes, the status says CREATE_COMPLETE, as shown in Figure 3-17.

0	Cloud	Formation	n v Stack	(S						
Cr	eate Stac	ж - А	ctions -	Design te	mplate					C O
Fill	ter: Activ	ve • By St	ack Name							Showing 1 stack
	Stack	Name	Cre	ated Time		Stat	us		Description	
	Docker	rSwarm	201	17-07-22 14:5	9:31 UTC-070	0 CRE	ATE_COM	PLETE	Docker CE for AWS 17.06.	0-ce (17.06.0-ce-aws2)
Ov	erview	Outputs	Resources	Events	Template	Parameters	Tags	Stack Polic	y Change Sets	880
2017	-07-22		Status		Тур	•			Logical ID	Status reason
	15:08:33	UTC-0700	CREATE_CO	MPLETE	AW	S::CloudForma	tion::Stack		DockerSwarm	
	15:08:30	UTC-0700	CREATE_CO	MPLETE	AW	S::AutoScaling	LifecycleH	look	SwarmWorkerUpgradel	Hook
	15:08:29	UTC-0700	CREATE_IN_I	PROGRESS	AW	S::AutoScaling	LifecycleH	look	SwarmWorkerUpgradel	Hook Resource creation Initiated
	15:08:29	UTC-0700	CREATE_IN_I	PROGRESS	AW	S::AutoScaling	LifecycleH	look	SwarmWorkerUpgradel	Hook
	15:08:25	UTC-0700	CREATE_CO	MPLETE	AW	S::AutoScaling	AutoScali	ngGroup	NodeAsg	
•	15:08:23	UTC-0700	CREATE_IN_I	PROGRESS	AW	S::AutoScaling	AutoScali	ngGroup	NodeAsg	Received SUCCESS signal with Uniqueld i-Daac881e ac57f2bd3
	15:08:02	UTC-0700	CREATE_IN_I	PROGRESS	AW	S.:AutoScaling	AutoScali	ngGroup	NodeAsg	Received SUCCESS signal with Uniqueld I-0a4fe45a 2497dc3b6
,	15:08:01	UTC-0700	CREATE_IN_I	PROGRESS	AW	S::AutoScaling	AutoScali	ngGroup	NodeAsg	Received SUCCESS signal with UniqueId I-027111d2 804a125ba
•	15:08:01	UTC-0700	CREATE_IN_I	PROGRESS	AW	S::AutoScaling	AutoScali	ngGroup	NodeAsg	Received SUCCESS signal with Uniqueld I-04f6c727 e2c94384e

Figure 3-17. Stack status is CREATE_COMPLETE

All the required resources—including auto-scaling groups, EC2 Internet Gateway, EC2 security groups, Elastic Load Balancer, IAM policy, Log Group, and VPC Gateway—are created, as shown in Figure 3-18.

Cloud	dFormation	~ Stack	ks								
Create Stat	sk 🔹 Ar	ctions -	Design t	emplate							C O
Filter: Acti	ve • By Sta	ck Name									Showing 1 stack
Stack	Name	Cre	eated Time		Statu	s		Description			
Docke	rSwarm	201	17-07-22 14:	59:31 UTC-07	00 CREA	TE_CON	IPLETE	Docker CE for AW	S 17.06.0-ce (17.06.0-ce-av	vs2)	9
Overview	Outputs	Resources	Events	Template	Parameters	Tags	Stack Polic	cy Change Sets			885
Logical ID		Physical	ID				Туре		Status	Status Reason	*
AZInfo		2017/07/2 3937	22/[\$LATEST]8598f668a72	74ebbb1579bd3	219a	Custom::AZInfe	b	CREATE_COMPLETE		
AZInfoFund	tion	DockerSy	varm-AZInfo	Function-16CF	7771HOCF0		AWS::Lambda:	Function	CREATE_COMPLETE		
AttachGate	way	Docke-At	tac-1VS1AX	18UO1WW			AWS: EC2::VP	CGatewayAttach	CREATE_COMPLETE		
CloudstorE	BSPolicy	Docke-Cl	ou-1NNAR8	CS6KXLI			AWS::IAM::Pol	icy	CREATE_COMPLETE		
DockerLog	Group	DockerSv	varm-lg			,	AWS::Logs::Lo	gGroup	CREATE_COMPLETE		
DynDBPolk	cles	Docke-Dy	nD-7ZLQ01	H9AY1I			AWS::IAM::Pol	icy	CREATE_COMPLETE		
DynDBWor	kerPolicies	Docke-Dy	nD-1AC97B	WWO6VB			AWS: IAM Pol	icy	CREATE_COMPLETE		
ExternalLoa	adBalancer	DockerSv	va-External-	HBH91HC4D	eco		AWS.:ElasticLo	adBalancing::Lo	CREATE_COMPLETE		
ExternalLoa	adBalancerSG	sg-0b93b	e7a				AWS: EC2::Se	curityGroup	CREATE_COMPLETE		

Figure 3-18. Resources are created

The Outputs tab lists the Default DNS target, the zone availability comment about the number of availability zones, and the manager nodes, as shown in Figure 3-19.

	Cloud	Formation	n v Stac	ks									
(Create Star	ik 🝷 A	ctions -	Design	template								C O
	Filter: Acti	ve • By St	ack Name										Showing 1 stack
	Stack	Name	Cr	eated Time		State	is			Description			
	Docke	rSwarm	20	17-07-22 1	4:59:31 UTC-07	0 CRE	ATE_COM	IPLETE		Docker CE for AWS 17.0	6.0-ce (17.06.0-ce-a	ws2)	0
1	Overview	Outputs	Resources	Events	Template	Parameters	Tags	Stack F	Policy	Change Sets			888
1	Кеу				Value				Desc	ription		Export Name	
	DefaultDNS	Target			DockerSwa-Ex 68.us-east-1.el	temal-1HBH91 b.amazonaws.c	HC4D9CC om	-495913	Uset	his name to update your	DNS records		
	ZoneAvaila	bilityCommer	nt		This region has This is ideal to case you lose a	s at least 3 Avai ensure a fully fu an AZ.	lability Zor Inctional S	nes (AZ). Swarm in	Avail	abilty Zones Comment			
	Managers				https://us-east- 2/home?region oscaling:group g-1L8ET\$5Y21	1.console.aws =us-east-1#ins Name=DockerS /IC5A;sort=desc	amazon.co lances.tag warm-Ma ::dnsNam	om/ec2/v aws:aut nagerAs e	You	an see the manager nod	les associated wi		
	VPCID				vpc-055d3f7c				Uset	his as the VPC for config	uring Private Ho		
	ELBDNSZO	nelD			Z35SXDOTRO	7X7K			Usel	his zone ID to update yo	ur DNS records		

Figure 3-19. Outputs

To list the EC2 instances for the Swarm managers, click on the link in Managers, as shown in Figure 3-20.

Create Stac	ck • A	ctions -	Design	n template							C ¢
Filter: Activ	ve • By Sta	ck Name									Showing 1 stack
Stack	Name	Cre	ated Time		Statu	IS			Description		
Docker	rSwarm	201	17-07-22 1	4:59:31 UTC-070	0 CREA	ATE_COM	PLETE	0	Docker CE for AWS 17.06.0-ce	17.06.0-ce-aws2)	0
Overview	Outputs	Resources	Events	Template	Parameters	Tags	Stack P	olicy	Change Sets		888
Key				Value				Desci	ription	Export Name	
DefaultDNS	STarget			DockerSwa-Ex 68.us-east-1.el	ternal-1HBH91H b.amazonaws.c	HC4D9CO om	495913	Use th	is name to update your DNS re	cords	
ZoneAvaila	bilityCommen	t		This region has This is ideal to case you lose a	at least 3 Avail ensure a fully fu an AZ.	ability Zon Inctional S	es (AZ). warm in	Availa	bilty Zones Comment		
Managers				https://us-east- 2/home?region oscaling.group	1 console aws a =us-east-1#inst Name=DockerS	amazon.co ances.tag warm_Mar	m/ec2/v aws.aut lagerAs	You ca	an see the manager nodes asso	clated wi	
VPCID				vpc-055d3f7c	1#Insta	/us-east-1.co nces:tag:aw	onsole.aws.a s:autoscaling	mazon.c grouph	om/ec2/v2/home?region=us-east- lame=DockerSwarm-ManagerAsg-1L its as the VPC for coninguring P	8ETSSY2MCSA;sort=desc:dnsName	
ELBDNSZO	melD			Z35SXDOTRQ	7X7K			Use th	is zone ID to update your DNS	records	

Figure 3-20. The Managers link

The three manager instances are all in different availability zones. The public/private IP addresses and the public DNS name for each EC2 instance may be obtained from the EC2 console, as shown in Figure 3-21.

Q aws:auto	scaling:groupName	: DockerSwarm-Man	nagerAsg-1L8ETS5Y21	MC5A 🔿 Add filter				0	K 4	1 to 3 of	f3 >	>
Name	~ Ins	stance ID -	Instance Type 🔹	Availability Zone 👻	Instance State -	Status	Checks 👻	Alarm State	IS	Public DN	IS (IPv4	4)
DockerSw	arm-Manager i-0	339cf772ffbf2a18	t2.micro	us-east-1c	running	Ø 2/2	checks	None	>	ec2-54-89	-68-201	l.com
DockerSw	arm-Manager i-0	029a79d96d0fada2	t2.micro	us-east-1b	running	2/2	checks	None	20	ec2-34-22	6-138-1	97.c
DockerSw	arm-Manager i-0	af871b372a1151ad	t2.micro	us-east-1a	running	2/2	checks	None	20	ec2-34-20	0-226-2	246.cr
nstance: i-03	339cf772ffbf2a18 (Status Checks	Monitoring	Tags	NS: ec2-54-89-68-201.	.compute-1.amazo	naws.co	m	201		I	88	
nstance: i-03	Status Checks	DockerSwarm-Mar Monitoring	Tags	\$\$\$ \$\$: ec2•54•89•68•201.	compute-1.amazo	naws.co	m	-201 compute		1	88	
nstance: I I-03 Description	Status Checks	Docker Swarm-Mar Monitoring i-0339cf772ffbf2a1	Tags	45: ec2-54-89-68-201.	compute-1.amazo	Pv4) e	m c2-54-89-68 .amazonawa	I-201.compute s.com	-	l	88	
nstance: i-03	339cf772ffbf2a18 (I Status Checks Instance ID Instance state	Docker Swarm-Mar Monitoring i-0339cf772ffbf2a1 running	Tags 8	NS: ec2-54-89-68-201.	Public DNS (IPv4) e Ilic IP 5	m c2-54-89-68 .amazonaw: 44.89.68.201	3-201.compute s.com		1	88	
nstance: i-0	Status Checks Instance ID Instance state	Docker Swarm-Mar Monitoring i-0339cf772ftbf2a1 running t2.micro	nager) Public DM Tags	NS: ec2-54-89-68-201.	Public DNS (IPv4) e 1 lic IP 5 6 IPs -	m c2-54-89-68 .amazonaw: 4.89.68.201	-201.compute s.com		1	88	
nstance: i-03	Status Checks Instance ID Instance state Instance type Elastic IPs	Monitoring i-0339ef772ffbf2a1 running t2.micro	nager) Public DM Tags	NS: ec2-54-89-68-201.	Public DNS (Public DNS (Private Private	IPv4) e lic IP 5 6 IPs - DNS ii	m c2-54-89-68 .amazonawa 4.89.68.201 p-172-31-33- 72.21.33.25	-201.compute s.com -35.ec2.interna	-	1	88	

Figure 3-21. Manager instances on EC2

The AMI used for the EC2 instances may be found using the AMI ID, as shown in Figure 3-22. A Moby Linux AMI is used for this Swarm, but the AMI could be different for different users and in different AWS regions.

Laur	ch instance Connect	Actions ¥								0	¢	0
Q,	aws:autoscaling:groupName	e : DockerSwarm-Mana	gerAsg-1L8ETS5Y2N	IC5A 🕥 Add filter				0	K <	1 to 3 of 3	>	>
	Name - In	stance ID 👻	Instance Type 👻	Availability Zone 👻	Instance State -	Sta	tus Checks 👻	Alarm Status	5	Public DNS	(IPv4)	
	DockerSwarm-Manager i-0	0339cf772ffbf2a18	t2.micro	us-east-1c	running	0	2/2 checks	None	>	ec2-54-89-68	-201.c	
	DockerSwarm-Manager i-0	0029a79d96d0fada2	t2.micro	us-east-1b	running	0	2/2 checks	None	>	ec2-34-226-1	38-197	7.co
	DockerSwarm-Manager i-0	Daf871b372a1151ad	t2.micro	us-east-1a	running	0	2/2 checks	None	7	ec2-34-200-2	26-246	5.co
4												*
	Instance ID	i-0339cf772ffbf2a18		6 G Ø	Public DNS (II	Pv4)	ec2-54-89-68 1.amazonaws	-201.compute-				•
	Instance state	running			IPv4 Publi	ic IP	54.89.68.201					
	Instance type	t2.micro			IPv6	5 IPs	-					
	Elastic IPs				Private L	DNS	ip-1/2-31-33-	35.ec2.internal				
	Availability zone	us-east-1c			Private	Ps	1/2.31.33.35					
	Security groups	DockerSwarm-Mana 1UAGHYRA351UQ SwarmWideSG-QW inbound rules	agerVpcSG- , DockerSwarm- DRB6Q4F087 . view		Secondary private	e IPs						
	Scheduled events	No scheduled event	s		VP	CID	vpc-055d3f7c					
	₿ ^{AMI ID}	Moby Linux 17.06.0 a25c51b4)	-ce-aws2 stable (ami-		Subne	et ID	subnet-d1326	099				
	Platform	G			Network interfa	aces	eth0					-

Figure 3-22. Moby Linux AMI

You can list all the EC2 instances by setting Instance State to Running. The Docker Swarm manager nodes (three) and worker nodes (five) are listed, as shown in Figure 3-23. The manager and worker nodes are in three different availability zones.

Q Instance	State : Running	Add filter					0	K	1 to 8 of 8 > >
Name	-	Instance ID	Instance Type 👻	Availability Zone 👻	Instance State -	Status Checks 👻	Alarm Stat	us	Public DNS (IPv4)
DockerSw	arm-worker	i-0a4fe45a2497dc3b6	t2.micro	us-east-1a	running	2/2 checks	None	>>	ec2-107-23-82-165.co.
DockerSw	arm-worker	i-04f6c727e2c94384e	t2.micro	us-east-1b	running	2/2 checks	None	7	ec2-54-173-99-220.co.
DockerSw	arm-worker	i-0aac881eac57f2bd3	t2.micro	us-east-1c	running	2/2 checks	None	20	ec2-54-144-50-220.co.
DockerSw	arm-Manager	i-0339cf772ffbf2a18	t2.micro	us-east-1c	running	2/2 checks	None	20	ec2-54-89-68-201.com
DockerSw	arm-worker	i-027111d2804a125ba	t2.micro	us-east-1c	running	2/2 checks	None	>	ec2-52-23-223-212.co.
DockerSw	arm-Manager	i-0029a79d96d0fada2	t2.micro	us-east-1b	running	2/2 checks	None	>>	ec2-34-226-138-197.co
DockerSw	arm-worker	i-01bb468a3a5bab84c	t2.micro	us-east-1a	running	2/2 checks	None	>>	ec2-34-205-53-11.com
DockerSw	arm-Manager	i-0af871b372a1151ad	t2.micro	us-east-1a	running	2/2 checks	None	20	ec2-34-200-226-246.cc
nstance: i-0a	Afe45a2497dc	3b6 (DockerSwarm-w	orker) Public DM	IS: ec2-107-23-82-16	5.compute-1.amazo	onaws.com			880
beschphon	Instance	D i-0a4fe45a2497dd	3b6		Public DNS (I	Pv4) ec2-107-23-8 1.amazonaws	2-165.comput	te-	
	Instance st	ate running			Pv4 Publ	ic IP 107.23.82.16	5		
	instance t	ype uz.micro			IPvt	o IPS -			

Figure 3-23. Swarm managers and workers in three different availability zones

Select Load Balancers in the EC2 dashboard and the provisioned Elastic Load Balancer is listed, as shown in Figure 3-24. Click on the Instances tab to list the instances. All instances should have a status set to InService, as shown in Figure 3-24.

					-	
Iter: Q Search	×			< < 1 to	1 of 1	>
Name	- DNS name	- State -	VPC ID	Availability Zones ·	Туре	
DockerSwa-External-1HBH	H9 DockerSwa-External-1HBH9		vpc-055d3f7c	us-east-1a, us-east-1b,	classic	
ad balancer: DockerSwa-	External-1HBH91HC4D9CO	0.0.6				
Description Instances	Health Check Listeners	Monitoring Tags				
Connection Draining: Disab	and (Edit)					
connection braining: Disau	neu (cuit)					
Edit Instances	Ned (Edit)					
Edit Instances	(Euit)					
Edit Instances	Name	Availability Zone	Status	Actions		
Edit Instances Instance ID i-0339cf772ffbf2a18	Name DockerSwarm-Manager	Availability Zone us-east-1c	Status InService (j)	Actions Remove from Load Balancer		
Edit Instances Instance ID I-0339cf772fbf2a18 I-0af871b372a1151ad	Name DockerSwarm-Manager DockerSwarm-Manager	Availability Zone us-east-1c us-east-1a	Status InService () InService ()	Actions Remove from Load Balancer Remove from Load Balancer		
Edit Instances Instance ID i-0339cf772fbf2a18 i-0af871b372a1151ad i-04f6c727e2c94384e	Name DockerSwarm-Manager DockerSwarm-Manager DockerSwarm-worker	Availability Zone us-east-1c us-east-1a us-east-1b	Status InService () InService ()	Actions Remove from Load Balancer Remove from Load Balancer Remove from Load Balancer		
Edit Instances Instance ID i-0339cf772fbf2a18 i-0af871b372a1151ad i-04f6c727e2c94384e i-01bb468a3a5bab84c	Name DockerSwarm-Manager DockerSwarm-Manager DockerSwarm-worker DockerSwarm-worker	Availability Zone us-east-1c us-east-1a us-east-1b us-east-1a	Status InService () InService () InService ()	Actions Remove from Load Balancer Remove from Load Balancer Remove from Load Balancer Remove from Load Balancer		
Edit Instances Instance ID i-0339cf772fbf2a18 i-0af871b372a1151ad i-04f6c727e2c94384e i-01bb468a3a5bab84c i-01bb468a3a5bab84c	Name DockerSwarm-Manager DockerSwarm-Manager DockerSwarm-worker DockerSwarm-worker DockerSwarm-worker	Availability Zone us-east-1c us-east-1a us-east-1b us-east-1a us-east-1a	Status InService () InService () InService () InService ()	Actions Remove from Load Balancer Remove from Load Balancer Remove from Load Balancer Remove from Load Balancer Remove from Load Balancer		
Edit Instances Instance ID i-0339cf772fbf2a18 i-0af871b372a1151ad i-04f6c727e2c94384e i-01bb468a3a5bab84c i-0a4fe45a2497dc3b6 i-0029a79d96d0fada2	Name DockerSwarm-Manager DockerSwarm-Manager DockerSwarm-worker DockerSwarm-worker DockerSwarm-worker DockerSwarm-worker	Availability Zone us-east-1c us-east-1a us-east-1b us-east-1a us-east-1a us-east-1a	Status InService () InService () InService () InService () InService ()	Actions Remove from Load Balancer Remove from Load Balancer		

Figure 3-24. Elastic Load Balancer

Select Launch Configurations from the EC2 dashboard. The two launch configurations—one for the managers and one for the worker nodes—will be listed, as shown in Figure 3-25.

4	Cr	eate	launch configuration Create Auto Scaling grou	up A	ctio	ns ~						0	¢	0
	Filt	ter:	Q, Filter launch configurations X						K	<	1 to 2 of 2 Launch Config	urations	>	>
			lame		•	AMI ID	Ŧ	Instance Type 👻	Spot Price	Ŧ	Creation Time	*		
		0	lockerSwarm-NodeLaunchConfig17060ceaws2-RYA1B9VJK	KDF3		ami-a25c51b4		t2.micro			July 22, 2017 3:05:29 PM UT	C-7		
G			ockerSwarm-ManagerLaunchConfig17060ceaws2-1VOMPF	PGK1X	1	ami-a25c51b4		t2.micro			July 22, 2017 3:02:06 PM UT	C-7		

Figure 3-25. Launch configurations

Select Auto Scaling Groups in the EC2 dashboard. The two auto-scaling groups—one for the managers and one for the worker nodes—will be listed, as shown in Figure 3-26.

•	Crea	te Auto Scaling group Actions *							Ð	¢	0
	Filter	Q. Filter Auto Scaling groups	×				ŀ	1 to 2 of 2 Auto Scaling	Grou	os 🔅	>
		Name	Launch Configuration 👻	Instances -	Desired ~	Min ~	Max -	Availability Zones -	Defa	ult Co	oldow
1		DockerSwarm-NodeAsg-1PWVVETKVWXMJ	DockerSwarm-NodeLau	5	5	0	1,000	us-east-1a, us-east-1b, us-e	300		
B		DockerSwarm-ManagerAsg-1L8ETS5Y2MC5A	DockerSwarm-Manager	3	3	0	6	us-east-1a, us-east-1b, us-e	300		

Figure 3-26. Auto-scaling groups

Connecting with the Swarm Manager

Next, connect to a Swarm manager node from the local machine on which the key pair docker.pem is copied. Using the public IP address of a manager EC2 instance, SSH login into the instance with user as "docker".

```
ssh -i "docker.pem" docker@54.89.68.201
```

The command prompt for the manager node is displayed.

[root@localhost ~]# ssh -i "docker.pem" docker@54.89.68.201

Welcome to Docker!

The Docker version of the Swarm node may be listed using docker --version. The version will be 17.06 or greater. Swarm mode is supported on Docker 1.12 or greater.

```
~ $ docker --version
Docker version 17.06.0-ce, build 02c1d87
```

Using the Swarm

List the Swarm nodes with docker node 1s and the three manager nodes and five worker nodes will be listed.

~ \$ docker node ls

ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS

255llm8729rns82bmloaxs6usl ip-172-31-8-37.ec2.internal Ready Active

ikyskl4ysocymoe4pbrj3qnh3 ip-172-31-4-154.ec2.înternal Ready Active Reachable

p2ky6meej8tnph5wyuw59xtmr ip-172-31-21-30.ec2.internal Ready Active Leader

r56kkltfgc4zzzfbslinrun2d1 ip-172-31-24-185.ec2.internal Ready Active

soggz5qplcihk8y2y58uj9md4 ip-172-31-1-33.ec2.internal Ready Active

xbdeo8qp9jhi398h478wl2zrv * ip-172-31-33-35.ec2.internal Ready Active Reachable

ykk4odpjps6t6eqc9mriqvo4a ip-172-31-47-162.ec2.internal Ready Active

zrlrmijyj5vklxl3ag7gayb3w ip-172-31-39-210.ec2.internal Ready Active

The leader node and two other manager nodes indicated by Manager Status of Leader and Reachable are listed. The worker nodes are all available, as indicated by Active in the Availability column.

Docker services are introduced in the next chapter, but you can run the following docker service create command to create an example Docker service for a MySQL database.

```
docker service create \
    --env MYSQL_ROOT_PASSWORD='mysql'\
    --replicas 1 \
    --name mysql \
    --update-delay 10s \
    --update-parallelism 1 \
    mysql
```

A service gets created:

```
~ $ docker service create \
> --env MYSQL_ROOT_PASSWORD='mysql'\
> --replicas 1 \
> --name mysql \
> --update-delay 10s \
> --update-parallelism 1 \
> mysql
12hg71a3vy793quv14uems5gk
```

List the service with the docker service 1s command, which is also discussed in the next chapter, and the service ID, mode, replicas, and image are listed.

```
~S docker service ls
```

```
ID NAME MODE REPLICAS IMAGE
```

```
n2tomumtl9sbniysql replicated 1/1 mysql:latest
```

Scale the service to three replicas with the docker service scale command. The three replicas are scheduled—one on the leader manager node and two on the worker nodes. The docker service ps command to list service replicas is also discussed in more detail in the next chapter.

```
~ S docker service scale mysql=3
mysql scaled to 3
~ S docker service ps mysql
ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR PORTS
slqtuf9l4hxo mysq1.1 mysql:latest ip-172-31-35-3.us-east-2.compute.internal
Running Running about a minute ago
exqsthrgszzc mysql.2 mysql:latest ip-172-31-27-83.us-east-2.compute.internal
Running Preparing 8 seconds ago
vtuhsl6mya85 mysql.3 mysql:1atest ip-172-31-29-199.us-east-2.compute.internal Running
Preparing 8 seconds ago
```

Deleting a Swarm

To delete a Swarm, choose Actions ➤ Delete Stack from the CloudFormation console, as shown in Figure 3-27.

Cloud	dFormation	n 🗸 Stac	ks							
Create Stac	ck - A	actions -	Design t	emplate						C Q
Filter: Activ	ve - 🕞	Create Change	Set For Curr	ent Stack						Showing 1 stack
Stack	Name	Update Stack		_	Statu	IS		Description		
Docke	rSwarm	Delete Stack			00 CREA	ATE_CON	IPLETE	Docker CE for AWS 1	7.06.0-ce (17.06.0-ce-aws2)	
Overview	Outputs	Resources	Events	Template	Parameters	Tags	Stack Policy	Change Sets		888
S	Stack name:	DockerSwarr	m							
	Stack ID:	arn:aws:clou	dformation;us	east-1:67259	3526685:stack/E	DockerSw	arm/09ce3260-6	f29-11e7-9c71-500c28t	b27a35	
	Status:	CREATE_CC	OMPLETE							
Star	tus reason:									
	IAM Role:									
	Description	Docker CE fo	or AWS 17.06	.0-ce (17.06.0	-ce-aws2)					

Figure 3-27. Choosing Actions ➤ *Delete Stack*

In the Delete Stack confirmation dialog, click on Yes, Delete, as shown in Figure 3-28.



Figure 3-28. Delete stack confirmation dialog

The stack's status becomes DELETE_IN_PROGRESS, as shown in Figure 3-29.

_								
c	reate Stack 🔹 Action	ns • Design template			C O			
Fil	Create Stack Actions Design template Filter: Active By Stack Name							
	Stack Name	Created Time	Status	Description				
	otaon Hame							

Figure 3-29. Delete in progress

As each of the stack's resources is deleted, its status becomes DELETE_COMPLETE, as shown for some of the resources on the Events tab in Figure 3-30.

1 Clou	dFormation	n 🛩 Stacks									
Create Sta	ack - A	ctions -	Design te	mplate						c	٥
Filter: Ac	tive • By St	ack Name								Showing	1 stack
Stack	k Name	Create	d Time		Status			Description			
Dock	erSwarm	2017-0	7-22 14:5	9:31 UTC-070	DELE		ROGRESS	Docker CE for AWS 17.06.0	l-ce (17.06.0-ce-aws2)		
Overview	Outputs	Resources	Events	Template	Parameters	Tags	Stack Policy	Change Sets			
2017-07-22		Status		Тур				Logical ID	Status reason		^
• 15:38:24	4 UTC-0700	DELETE_IN_PRO	GRESS	AW	S::EC2::Subnet			PubSubnetAz2			
• 15:38:24	4 UTC-0700	DELETE_IN_PRO	GRESS	AW	S::EC2::VPCGat	ewayAtta	chment	AttachGateway			
• 15:38:24	4 UTC-0700	DELETE_IN_PRO	GRESS	AW	S::EC2::Subnet			PubSubnetAz1			
• 15:38:24	4 UTC-0700	DELETE_IN_PRO	OGRESS	AW	S::EC2::Subnet			PubSubnetAz3			
• 15:38:22	2 UTC-0700	DELETE_COMPL	ETE	AW	S::ElasticLoadBa	lancing::	LoadBalancer	ExternalLoadBalancer			
• 15:38:22	2 UTC-0700	DELETE_IN_PRO	OGRESS	AW	S::ElasticLoadBa	lancing	LoadBalancer	ExternalLoadBalancer			
 15:38:20 	0 UTC-0700	DELETE_COMPL	ETE	AW	S::DynamoDB::T	able		SwarmDynDBTable			
• 15:37:54	4 UTC-0700	DELETE_COMPL	ETE	AW	S::IAM::Role			ProxyRole			
• 15:37:53	3 UTC-0700	DELETE_IN_PRO	OGRESS	AW	S.:IAM::Role			ProxyRole			
• 15:37:53	3 UTC-0700	DELETE_COMPL	ETE	AW	S.:EC2.:Security	Group		NodeVpcSG			
• 15:37:53	2 UTC-0700	DELETE_IN_PRO	GRESS	AW	S::EC2::Security	Group		NodeVpcSG			

Figure 3-30. Events list some of the resources with a status of DELETE_COMPLETE

When the EC2 instances have been deleted, the EC2 console lists their status as terminated, as shown in Figure 3-31.

Q,	Filter by tags and attributes	or search by keyword				0	< <	1 to 8 of 8	> >	1
	Name -	Instance ID *	Instance Type 👻	Availability Zone -	Instance State - Status Checks -	Alarm Statu	5	Public DNS	(IPv4)	
	DockerSwarm-Manager	i-0029a79d96d0fada2	t2.micro	us-east-1b	terminated	None	>			
	DockerSwarm-worker	i-01bb468a3a5bab84c	t2.micro	us-east-1a	terminated	None	10			
	DockerSwarm-worker	i-027111d2804a125ba	12.micro	us-east-1c	terminated	None	10			
	DockerSwarm-Manager	i-0339cf772ffbf2a18	t2.micro	us-east-1c	🥥 terminated	None	20			
	DockerSwarm-worker	i-04f6c727e2c94384e	t2.micro	us-east-1b	🧼 terminated	None	10			
	DockerSwarm-worker	i-0a4fe45a2497dc3b6	t2.micro	us-east-1a	🥚 terminated	None	20			
	DockerSwarm-worker	i-0aac881eac57f2bd3	t2.micro	us-east-1c	terminated	None	10			
	DockerSwarm-Manager	i-0af871b372a1151ad	t2.micro	us-east-1a	terminated	None	10			

Figure 3-31. EC2 instances with status set to terminated

Summary

This chapter discussed creating a multi-zone Docker Swarm provisioned by a CloudFormation template using the Docker for AWS service. You learned how to connect to the Swarm manager to run docker service commands. The next chapter introduces Docker services.

CHAPTER 4

Docker Services

A Docker container contains all the binaries and dependencies required to run an application. A user only needs to run a Docker container to start and access an application. The CoreOS Linux operating system has Docker installed and the Docker commands may be run without even installing Docker.

The Problem

A Docker container, by default, is started only on a single node. However, for production environments, where uptime and redundancy matters, you need to run your applications on multiple hosts.

When a Docker container is started using the docker run command, the container starts only on a single host, as illustrated in Figure 4-1. Software is usually not designed to run on a single host only. A MySQL database in a production environment, for example, may need to run across a cluster of hosts for redundancy and high availability. Applications that are designed for a single host should be able to scale up to multiple hosts as needed. But distributed Docker applications cannot run on a single Docker Engine.



Figure 4-1. Docker container on a single host

The Solution

Docker Swarm mode enables a Docker application to run across a distributed cluster of Docker Engines connected by an overlay network, as illustrated in Figure 4-2. A Docker service may be created with a specific number of replicas, with each replica potentially running on a different host in a cluster. A Swarm consists of one or more manager nodes with a single leader for Swarm management and orchestration. Worker nodes run the actual service tasks with the manager nodes being worker nodes by default. A Docker service may be started only from the leader node. Service replicas scheduled on the worker nodes, as a result, run a distributed application. Distributed applications provide several benefits, such as fault tolerance, failover, increased capacity, and load balancing, to list a few.

CHAPTER 4 DOCKER SERVICES



Figure 4-2. Docker service tasks and containers spread across the nodes

This chapter covers the following topics:

- Setting the environment
- The Docker service commands
- Types of services
- Creating a service
- Listing the tasks of a service
- Invoking a Hello World service task on the command line
- Getting detailed information about a service
- Invoking the Hello World service in a browser
- Creating a service for a MySQL database
- Scaling a service
- Listing service tasks
- Accessing a MySQL database in a Docker container
- Updating a service
- Updating the replicas
- Updating the Docker image tag
- Updating the placement constraints

- Updating environment variables
- Updating the Docker image
- Updating the container labels
- Updating resources settings
- Removing a service

Setting the Environment

Create a Docker Swarm consisting of one manager and two worker nodes using the procedure discussed in Chapter 3. First, start three CoreOS instances—one for a Swarm manager and two for the Swarm workers. Obtain the public IP address of the Swarm manager, as shown in the EC2 console in Figure 4-3.

Q Instance	State : Running 🗇	Add filter					0	K (< 1 to 3 of 3	>>
Name	* Ir	nstance ID 🗸 👻	Instance Type 🔹	Availability Zone -	Instance State 👻	Status Checks 👻	Alarm Stat	tus	Public DNS	(IPv4)
DockerSv	varm-worker i-l	08fe3aa870ead7c3b	t2.micro	us-east-1c	running	2/2 checks	None	70	ec2-52-91-39	-226.co
DockerSv	varm-worker i-l	014d06a4d265e89	t2.micro	us-east-1a	running	2/2 checks	None	7	ec2-52-54-70	-201.co
DockerSv	varm-Manager i-	0436f9d57d0d950d9	t2.micro	us-east-1a	running	2/2 checks	None	20	ec2-34-200-2	25-39.0
Description	Status Checks	9 (DockerSwarm-Ma	anager) Public (Tags	DNS: ec2-34-200-225	-39.compute-1.ama	zonaws.com				80
nstance: i-0	Status Checks	9 (DockerSwarm-M: Monitoring	anager) Public (Tags 0d9	DNS: ec2-34-200-225	-39.compute-1.ama	zonaws.com	25-39.compu	te-		
Description	Status Checks	9 (DockerSwarm-M: Monitoring i-0436f9d57d0d950	anager) Public I Tags 0d9	DNS: ec2-34-200-225	-39.compute-1.ama Public DNS (I	zonaws.com Pv4) ec2-34-200-2 1.amazonaws	25-39.compu .com	te-		88
Description	436f9d57d0d950d Status Checks Instance ID	9 (DockerSwarm-M: Monitoring i-0436f9d57d0d950 running	anager) Public (Tags 0d9	DNS: ec2-34-200-225	-39.compute-1.ama Public DNS (I	zonaws.com Pv4) ec2-34-200-2 1.amazonaws ic IP 34.200.225.3	25-39.compu s.com	te-	•	
Description	436f9d57d0d950d Status Checks Instance ID Instance state Instance type	9 (DockerSwarm-M: Monitoring i-0436f9d57d0d950 running t2.micro	anager) Public (Tags 0d9	DNS: ec2-34-200-225	-39.compute-1.ama Public DNS (I Pv4 Public IPv4	zonaws.com Pv4) ec2-34-200-22 1.amazonaws ic IP 34.200.225.31 iPs -	25-39.compu .com 9	te-		88
Description	436f9d57d0d950d Status Checks Instance ID Instance state Instance type Elastic IPs	9 (DockerSwarm-Mi Monitoring i-0436f9d57d0d950 running t2.micro	anager) Public (Tags	DNS: ec2-34-200-225	-39.compute-1.ama Public DNS (I Pv4 Public IPv4 Private Denote	zonaws.com Pv4) ec2-34-200-2: 1.amazonaws ic IP 34-200-225.3: iPs - DNS ip-172-31-13- - 27-2 31-13-6:	25-39.compu .com 9 155.ec2.inter	te- nal		80

Figure 4-3. EC2 instances for Swarm

SSH login to the Swarm manager instance with user as "docker".

```
[root@localhost ~]# ssh -i "docker.pem" docker@34.200.225.39
Welcome to Docker!
```

Three nodes should get listed in the Swarm with the docker node 1s command—one manager node and two worker nodes.

~ \$ docker node ls				
ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS
ilru4f0i280w2tlsrg9hglwsj	ip-172-31-10-132.ec2.internal	Ready	Active	
w5to186ipblpcq390625wyq2e	ip-172-31-37-135.ec2.internal	Ready	Active	
<pre>zkxle7kafwcmt1sd93kh5cy5e *</pre>	ip-172-31-13-155.ec2.internal	Ready	Active	Leader

A worker node may be promoted to a manager node using the docker node promote <node ip> command.

```
~ $ docker node promote ilru4f0i280w2tlsrg9hglwsj
Node ilru4f0i280w2tlsrg9hglwsj promoted to a manager in the swarm.
```

If you list the nodes again, two manager nodes should be listed. A manager node is identified by a value in the Manager Status column. One node has a Manager Status of Reachable and the other says Leader.

a docket node is				
ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS
ilru4f0i280w2tlsrg9hglwsj	ip-172-31-10-132.ec2.internal	Ready	Active	Reachable
w5to186ipblpcq390625wyq2e	ip-172-31-37-135.ec2.internal	Ready	Active	
<pre>zkxle7kafwcmt1sd93kh5cy5e *</pre>	ip-172-31-13-155.ec2.internal	Ready	Active	Leader

The manager node that is the Leader performs all the swarm management and orchestration. The manager node that is Reachable participates in the raft consensus quorum and is eligible for election as the new leader if the current leader node becomes unavailable.

Having multiple manager nodes adds fault tolerance to the Swarm, but one or two Swarm managers provide the same fault tolerance. If required, one or more of the worker nodes could also be promoted to a manager node to increase fault tolerance.

For connectivity to the Swarm instances, modify the inbound rules of the security groups associated with the Swarm manager and worker instances to allow all traffic. The inbound rules for the security group associated with a Swarm node are shown in Figure 4-4.

Create Security	Group	Actions ¥								Ð	۰	0
Q, search : s	g-6c39101d	Add filter						0	< <	1 to 2 of 2	>	>
Name	- Grou	ıp ID	-	Group Name -	VPC ID	~	Description					*
	sg-6e	:39101d		DockerSwarm-NodeVpcSG	vpc-b9bed3c0		Node SecurityGroup					-
	sg-de	250cad		DockerSwarm-ManagerVpc	vpc-b9bed3c0		Manager SecurityGroup					+
Security Group	Description		Tags							1	88	
Edit												
Туре ()			Protoc	ol (j)	Port Range ()		Source	1				
All traffic			All		All		0.0.0/0					
All traffic			All		All		(0					

Figure 4-4. Setting inbound rules on a security group to allow all traffic
The outbound rules for the security group associated with the Swarm manager are shown in Figure 4-5.

C	search : s	g-dc250cad	3 Add filter						0		< <	1 to 1 of 1	>	>
	Name	- Gro	up ID	-	Group Name	*	VPC ID	-	Description					*
	1	sg-d	c250cad		DockerSwarm-ManagerVpc		vpc-b9bed3c0		Manager SecurityGroup					
Sec	curity Group	sg-dc250c	ad				a d a					1	88	
D	Description	Inbound	Outbound	Tags										
	Edit													
	Туре ()			Protoco	(i)		Port Range (j)		Destination	n (j)			
	All traffic			All			All		0.0.0/0					-

Figure 4-5. Setting outbound rules on a security group to allow all traffic

The docker service Commands

The docker service commands are used to manage Docker services. The docker service command provides the sub-commands listed in Table 4-1.

Table 4-1. The docker service Sub-Commands

Command			Description			
docker	service c	reate	Creates a new service.			
docker	service in	nspect	Displays detailed information on one or more services.			
docker	service lo	ogs	Fetches the logs of a service. The command was added in Docker 17.0.6.			
docker	service le	s	Lists services.			
docker	service p	s	Lists the tasks of one or more services.			
docker	service r	m	Removes one or more services.			
docker	service so	cale	Scales one or multiple replicated services.			
docker	service up	pdate	Updates a service.			

To run docker service commands, the following requirements must be met.

- The Docker Swarm mode must be enabled
- The docker service commands must be run from the Swarm manager node that is the Leader

The docker service commands are available only in Swarm mode and cannot be run outside the Swarm mode.

The docker service commands cannot be run from a worker node. Worker nodes cannot be used to view or modify Swarm cluster state.

Types of Services

Docker Swarm mode supports two types of services, also called service modes—*replicated services* and *global services*. Global services run one task only on every node in a Docker Swarm. Replicated services run as a configured number of tasks, which are also referred to as *replicas*, the default being one. The number of replicas may be specified when a new service is created and may be updated later. The default service type is a replicated service. A global service requires the --mode option to be set to global. Only replicated services may be scaled; global services cannot be scaled.

We start off by creating a replicated service. Later in the chapter, we also discuss creating a global service.

Creating a Service

The command syntax to create a Docker service is as follows.

```
docker service create [OPTIONS] IMAGE [COMMAND] [ARG...]
```

Some of the supported options are listed in Table 4-2.

Table 4-2. Supported Options for Creating a Service

Option	Description			
constraint	Placement constraints.			
container-label	Container labels.			
env, -e	Sets environment variables.			
env-file	Reads in a file of environment variables. Option not added until Docker 1.13.			
host	Sets one or more custom host-to-IP mappings. Option not added until Docker 1.13. Format is host:ip.			
hostname	Container hostname. Option not added until Docker 1.13.			
label, -l	Service labels.			
limit-cpu	Limits CPUs. Default value is 0.000.			
limit-memory	Limits memory. Default value is 0.			
log-driver	Logging driver for service.			
log-opt	Logging driver options.			
mode	Service mode. Value may be replicated or global. Default is replicated.			
mount	Attaches a filesystem mount to the service.			
name	Service name.			
network	Network attachments. By default, the "ingress" overlay network is used.			
publish, -p	Publishes a port as a node port.			
read-only	Mounts the container's root filesystem as read only. Option not added until Docker 17.03. Default is false.			

(continued)

Table 4-2. (continued)

Option	Description
replicas	Number of tasks.
reserve-cpu	Reserves CPUs. Default is 0.000.
reserve-memory	Reserves memory. Default is 0.
restart-condition	Restarts when condition is met. Value may be none, on-failure, or any.
restart-delay	Delays between restart attempts (ns us ms s m h).
restart-max-attempts	Maximum number of restarts before giving up.
tty, -t	Whether to allocate a pseudo-TTY. Option not added until Docker 1.13. Default is false.
update-delay	Delays between updates (ns us ms s m h). Default is 0s.
update-failure-action	Action on update failure. Value may be pause or continue. Default value is pause.
update-monitor	Duration after each task update to monitor for failure (ns us ms s m h). Default is 0s.
update-parallelism	Maximum number of tasks updated simultaneously. A value of 0 to updates all at once. Default value is 1.
user, -u	Username or UID in format: <name uid>[:<group gid>].</group gid></name uid>
workdir, -w	Working directory inside the container.

As an example, create a service called hello-world with Docker image tutum/hello-world consisting of two replicas. Expose the service on port 8080 on the host. The docker service create command outputs a service ID if successful.

```
~ $ docker service create \
> --name hello-world \
> --publish 8080:80 \
> --replicas 2 \
> tutum/hello-world
vyxnpstt351124h12niqm7s64
```

A service gets created.

Listing the Tasks of a Service

You can list the service tasks, also called replicas in the context of a replicated service, with the following command.

```
docker service ps hello-world
```

The two service tasks are listed.

~ \$ docker servi	ice ps hello-woi	rld	
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
zjmO3bjsqyhp	hello-world.1	<pre>tutum/hello-world:latest</pre>	ip-172-31-10-132.ec2.internal
Running	Running 41 seco	onds ago	
kezidi82ol5c	hello-world.2	<pre>tutum/hello-world:latest</pre>	ip-172-31-13-155.ec2.internal
Running	Running 41 seco	onds ago	

The ID column lists the task ID. The task name is in the format servicename.n; hello-world.1 and hello-world.2 for the two replicas. The Docker image is also listed. The NODE column lists the private DNS of the node on which the task is scheduled. The DESIRED STATE is the state that is desired as defined in the service definition. The CURRENT STATE is the actual state of the task. At times, a task could be in a pending state because of lack of resource capacity in terms of CPU and memory.

A service task is a slot for running a Docker container. On each node on which a task is running, a Docker container should also be running. Docker containers may be listed with the docker ps command.

~ \$ docker ps			
CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
0ccdcde64e7d	<pre>tutum/hello-world:latest</pre>	"/bin/sh -c 'php-f"	2 minutes ago
Up 2 minutes	80/tcp	hello-world.2.kezidi82	2ol5ct81u59jpgfhs1

Invoking a Hello World Service Task on the Command Line

Invoke the hello-world service using curl at <hostname>:8080. The curl command output is the HTML markup for the service.

```
~ $ curl ec2-34-200-225-39.compute-1.amazonaws.com:8080
<html>
<head>
        <title>Hello world!</title>
        <link href='http://fonts.googleapis.com/css?family=Open+Sans:400,700' rel='stylesheet'</pre>
        type='text/css'>
        <style>
        body {
                 background-color: white;
                 text-align: center;
                 padding: 50px;
                 font-family: "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif;
        }
        #logo {
                 margin-bottom: 40px;
        </style>
</head>
<body>
```

Getting Detailed Information About a Service

To get detailed information about the hello-world service, run the docker service inspect command.

```
docker service inspect hello-world
```

The detailed information includes the container specification, resources, restart policy, placement, mode, update config, ports (target port and published port), virtual IPs, and update status.

```
~ $ docker service inspect hello-world
[
    {
        "ID": "vyxnpstt351124h12niqm7s64",
        "Version": {
            "Index": 30
        },
        "CreatedAt": "2017-07-23T19:00:09.98992017Z",
        "UpdatedAt": "2017-07-23T19:00:09.993001487Z",
        "Spec": {
            "Name": "hello-world",
            "Labels": {},
            "TaskTemplate": {
                "ContainerSpec": {
                     "Image": "tutum/hello-world:latest@sha256:0d57def8055178aafb4c7669cbc25e
                    c17f0acdab97cc587f30150802da8f8d85",
                     "StopGracePeriod": 1000000000,
                    "DNSConfig": {}
                },
                "Resources": {
                    "Limits": {},
                    "Reservations": {}
                },
                "RestartPolicy": {
                     "Condition": "any",
                     "Delay": 500000000,
                    "MaxAttempts": 0
                },
                "Placement": {
                     "Platforms": [
                         {
                             "Architecture": "amd64",
                             "OS": "linux"
                         }
                    ]
                },
```

```
"ForceUpdate": 0,
         "Runtime": "container"
    },
    "Mode": {
        "Replicated": {
             "Replicas": 2
        }
    },
    "UpdateConfig": {
        "Parallelism": 1,
"FailureAction": "pause",
        "Monitor": 500000000,
        "MaxFailureRatio": 0,
        "Order": "stop-first"
    },
    "RollbackConfig": {
        "Parallelism": 1,
        "FailureAction": "pause",
        "Monitor": 500000000,
        "MaxFailureRatio": 0,
        "Order": "stop-first"
    },
    "EndpointSpec": {
        "Mode": "vip",
         "Ports": [
             {
                 "Protocol": "tcp",
                 "TargetPort": 80,
                 "PublishedPort": 8080,
                 "PublishMode": "ingress"
             }
        ]
    }
},
"Endpoint": {
    "Croc": {
    "Spec": {
         "Mode": "vip",
         "Ports": [
             {
                 "Protocol": "tcp",
                 "TargetPort": 80,
                  "PublishedPort": 8080,
                 "PublishMode": "ingress"
             }
        ]
    },
    "Ports": [
        {
             "Protocol": "tcp",
             "TargetPort": 80,
```

Invoking the Hello World Service in a Browser

The Hello World service may be invoked in a browser using the public DNS of a EC2 instance on which a Swarm node is hosted. A service replica does not have to be running on a node to invoke the service from the node. You obtain the public DNS of a manager node from the EC2 console, as shown in Figure 4-3. Invoke the Hello World service with <Public DNS>:<Published Port> URL. As the Hello World service is exposed or published on port 8080, the URL to invoke in a browser becomes <Public DNS>:8080. The service is invoked and the service output is displayed in the browser, as shown in Figure 4-6.



Figure 4-6. Invoking a service in a browser

CHAPTER 4 DOCKER SERVICES

Similarly, you can obtain the public DNS of a EC2 instance on which a Swarm worker node is hosted, as shown in Figure 4-7.

Q,	Instance State : Running	Add filter					0	< <	1 to 3 of 3	> >
	Name -	Instance ID *	Instance Type -	Availability Zone -	Instance State ~	Status Checks 👻	Alarm State	JS	Public DNS (IP	v4)
	DockerSwarm-worker	i-08fe3aa870ead7c3b	t2.micro	us-east-1c	running	2/2 checks	None	7	ec2-52-91-39-2	26.com.
	DockerSwarm-worker	i-014d06a4d265e89	t2.micro	us-east-1a	running	2/2 checks	None	2	ec2-52-54-70-2	D1.com.
	DockerSwarm-Manager	i-0436f9d57d0d950d9	12.micro	us-east-1a	running	2/2 checks	None	20	ec2-34-200-225	-39.co
-										,

Figure 4-7. Obtaining the public DNS for a EC2 instance on which a Swarm worker node is hosted

Invoke the service using the PublicDNS: 8080 URL in a browser, as shown in Figure 4-8.



Figure 4-8. Invoking a service in a browser using public DNS for a EC2 instance on which a Swarm worker node is hosted

A manager node is also a worker node by default and service tasks also run on the manager node.

Creating a Service for a MySQL Database

Next, we create a service for a MySQL database. Using the mysql Docker image is different than using the tutum/hello-world Docker image in two respects.

- The mysql Docker image has a mandatory environment variable called MYSQL_ROOT_ PASSWORD.
- The mysql Docker image is based on a Debian Linux and starts the MySQL database server in Docker container, while the tutum/hello-world image is based on Alpine Linux and starts Apache Server to run PHP applications.

Run the following docker service create command to create one replica of the MySQL database service. Supply a root password with the MYSQL_ROOT_PASSWORD environment variable. Include some other options for the restart condition, the restart max attempts, the update delay, and the update failure action. Remove any previously running Docker service called mysql with the docker service rm mysql command.

```
~ $ docker service create \
    --env MYSQL_ROOT_PASSWORD='mysql'\
    --replicas 1 \
    --restart-condition none \
    --restart-max-attempts 5 \
    --update-failure-action continue \
    --name mysql \
    --update-delay 10s \
    mysql
```

A service gets created for MySQL database and the service ID gets output.

```
~ $ docker service create \
> --env MYSQL_ROOT_PASSWORD='mysql'\
> --replicas 1 \
> --restart-condition none \
> --restart-max-attempts 5 \
> --update-failure-action continue \
> --name mysql \
> --update-delay 10s \
> mysql
gzl8k1wy8kf3ms1nu5zwlfxm6
```

List the services with the docker service 1s command; the mysql service should be listed.

~ \$ docker sei	vice ls				
ID	NAME	MODE	REPLICAS	IMAGE	PORTS
gzl8k1wy8kf3	mysql	replicated	1/1	mysql:latest	
vyxnpstt3511	hello-world	replicated	2/2	<pre>tutum/hello-world:latest</pre>	*:8080->80/tcp

List the service tasks/replicas with the docker service ps mysql command. One task is running on the manager worker node.

CHAPTER 4 DOCKER SERVICES

~ \$ docker service ps mysql ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR PORTS mfw76m4rxbhp mysql.1 mysql:latest ip-172-31-37-135.ec2.internal Running Running 16 seconds ago

How service tasks are scheduled, including node selection based on node ranking, is discussed in Chapter 8, which covers scheduling.

Scaling a Service

Next, we scale the mysql service. Only replicated services can be scaled and the command syntax to scale one or more services is as follows.

```
docker service scale SERVICE=REPLICAS [SERVICE=REPLICAS...]
```

To scale the mysql service to three tasks, run the following command.

docker service scale mysql=3

The mysql service gets scaled to three, as indicated by the command output.

```
~ $ docker service scale mysql=3
mysql scaled to 3
```

Listing Service Tasks

The docker service ps command syntax to list service tasks is as follows.

```
docker service ps [OPTIONS] SERVICE [SERVICE...]
```

The command supports the options listed in Table 4-3.

Table 4-3. Options for the docker service ps Command

Option	Description				
filter, -f	Filters output based on conditions provided. The following filters are supported:				
	id= <task id=""> name=<task name=""> node=<node id="" name="" or=""></node></task></task>				
	<pre>desired-state=(running shutdown accepted)</pre>				
no-resolve	Whether to map IDs to names. Default value is false.				
no-trunc	Whether to truncate output. Option not added until Docker 1.13. Default value is false.				
quiet, -q	Whether to only display task IDs. Option not added until Docker 1.13. Default value is false.				

As an example, you can list only the service tasks that are running.

docker service ps -f desired-state=running mysql

Only the running tasks are listed.

~ \$ docker service	os -f desired-state=runnir	ng mysql	
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
mfw76m4rxbhp	mysql.1	mysql:latest	ip-172-31-37-135.ec2.internal
Running	Running 46 seconds ago		
s4flvtode8od	mysql.2	mysql:latest	ip-172-31-13-155.ec2.internal
Running	Running 8 seconds ago		
j0jd92p5dmd8	mysql.3	mysql:latest	ip-172-31-10-132.ec2.internal
Running	Running 9 seconds ago		

All tasks are running; therefore, the effect of using the filter is not very apparent. But, in a subsequent example, you'll list running service tasks when some tasks are not running.

Not all worker nodes are utilized for running service tasks if the number of nodes is more than the number of tasks, as when the hello-world and mysql services had fewer than three tasks running. A node could have more than one service task running if the number of replicas is more than the number of nodes in a Swarm. Scaling up to five replicas starts more than one replica on two of the nodes.

```
~ $ docker service scale mysql=5
```

ps mysql		
NAME	IMAGE	NODE
CURRENT STATE	ERROR	PORTS
mysql.1	mysql:latest	ip-172-31-37-135.ec2.internal
Running about a minute ago		
mysql.2	mysql:latest	ip-172-31-13-155.ec2.internal
Running 44 seconds ago		
mysql.3	mysql:latest	ip-172-31-10-132.ec2.internal
Running 45 seconds ago		
mysql.4	mysql:latest	ip-172-31-37-135.ec2.internal
Running 26 seconds ago		
mysql.5	mysql:latest	ip-172-31-10-132.ec2.internal
Running 26 seconds ago		
	ps mysql NAME CURRENT STATE mysql.1 Running about a minute ago mysql.2 Running 44 seconds ago mysql.3 Running 45 seconds ago mysql.4 Running 26 seconds ago mysql.5 Running 26 seconds ago	ps mysql NAME IMAGE CURRENT STATE ERROR mysql.1 mysql:latest Running about a minute ago mysql.2 mysql:latest Running 44 seconds ago mysql.3 mysql:latest Running 45 seconds ago mysql.4 mysql:latest Running 26 seconds ago mysql.5 mysql:latest Running 26 seconds ago

Only one mysql service replica is running on the manager node; therefore, only one Docker container for the mysql service is running on the manager node.

~ \$ docker ps			
CONTAINER ID	IMAGE	COMMAND	
CREATED	STATUS	PORTS	NAMES
6bbe40000874	mysql:latest	"docker-entrypoint	п
About a minute ago	Up About a minute	3306/tcp	mysql.2.s4flvtode8odjjere2z si9gdx

Scaling to 10 tasks starts multiple tasks on each of the Swarm nodes.

~ \$ docker service	scale mysql=10			
mysql scaled to 10				
~ \$ docker service	ps -f desired-state	=running mysql		
ID	NAME	IMAGE	NODE	
DESIRED STATE	CURRENT STATE		ERROR	PORTS
s4flvtode8od	mysql.2	mysql:latest	ip-172-31-13-155.ec2	2.internal
Running	Running about a min	nute ago		
j0jd92p5dmd8	mysql.3	<pre>mysql:latest</pre>	ip-172-31-10-132.ec	2.internal
Running	Running 2 minutes a	ago		
6jtkvstssnkf	mysql.5	mysql:latest	ip-172-31-10-132.ec2	2.internal
Running	Running about a min	nute ago		
jxunbdec3fnj	mysql.6	mysql:latest	ip-172-31-37-135.ec2	2.internal
Running	Running 14 seconds	ago		
t1nz59dyoi2s	mysql.7	mysql:latest	ip-172-31-10-132.ec2	2.internal
Running	Running 14 seconds	ago		
lousvchdirn9	mysql.8	mysql:latest	ip-172-31-13-155.ec2	2.internal
Running	Running 14 seconds	ago		
94ml0f52344d	mysql.9	mysql:latest	ip-172-31-37-135.ec2	2.internal
Running	Running 14 seconds	ago		
pd40sd7qlk3j	mysql.10	mysql:latest	ip-172-31-13-155.ec2	2.internal
Running	Running 14 seconds	ago		

The number of Docker containers for the mysql service on the manager node increases to three for the three tasks running on the manager node.

~ \$ docker ps		
CONTAINER ID	IMAGE	COMMAND CREATED
STATUS	PORTS	NAMES
15e3253f69f1	mysql:latest	"docker-entrypoint" 50 seconds ago
Up 49 seconds	3306/tcp	mysql.8.lousvchdirn9fv8wot5vivk6d
cca7ab20c914	mysql:latest	"docker-entrypoint" 50 seconds ago
Up 49 seconds	3306/tcp	mysql.10.pd40sd7qlk3jc0i73huop8e4r
6bbe40000874	mysql:latest	"docker-entrypoint" 2 minutes ago
Up 2 minutes	3306/tcp	mysql.2.s4flvtode8odjjere2zsi9gdx

Because you'll learn more about Docker services with the MySQL database service example in later sections, and also for completeness, next we discuss using a Docker container for MySQL database to create a database table.

Accessing a MySQL Database in a Docker Container

Next, we access MySQL database in a Docker container. The docker ps command, when run on each instance, lists Docker containers for the mysql service on the instance. Start a bash shell for a Docker container with the docker exec -it <containerid> bash command. The root prompt gets displayed for the Docker container.

```
~ $ docker exec -it 15e3253f69f1 bash
root@15e3253f69f1:/#
```

Start the MySQL CLI with the mysql command as user root. Specify the password when prompted; the password used to create the service was specified in the --env option to the docker service create command using environment variable MYSQL_ROOT_PASSWORD. The mysql> CLI command prompt is displayed.

```
root@15e3253f69f1:/# mysql -u root -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.7.19 MySQL Community Server (GPL)
Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql>
```

Set the database to use as mysql with the use mysql command.

```
mysql> use mysql;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
```

Database changed

Create a database table with the following SQL script.

```
CREATE TABLE wlslog(time_stamp VARCHAR(45) PRIMARY KEY,category VARCHAR(25),type
VARCHAR(25),servername VARCHAR(25),code VARCHAR(25),msg VARCHAR(45));
```

The wlslog table is created.

```
mysql> CREATE TABLE wlslog(time_stamp VARCHAR(45) PRIMARY KEY,category VARCHAR(25),type
VARCHAR(25),servername VARCHAR(25),code VARCHAR(25),msg VARCHAR(45));
Query OK, 0 rows affected (0.06 sec)
```

Add some data to the wlslog table with the following SQL commands run from the MySQL CLI.

mysql> INSERT INTO wlslog VALUES('Apr-8-2014-7:06:16-PM-PDT','Notice','WebLogicServer', 'AdminServer','BEA-000365','Server state changed to STANDBY'); Query OK, 1 row affected (0.02 sec)

```
mysql> INSERT INTO wlslog VALUES('Apr-8-2014-7:06:17-PM-PDT','Notice','WebLogicServer',
'AdminServer','BEA-000365','Server state changed to STARTING');
Query OK, 1 row affected (0.01 sec)
```

CHAPTER 4 DOCKER SERVICES

mysql> INSERT INTO wlslog VALUES('Apr-8-2014-7:06:18-PM-PDT','Notice','WebLogicServer', 'AdminServer','BEA-000365','Server state changed to ADMIN'); Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO wlslog VALUES('Apr-8-2014-7:06:19-PM-PDT','Notice','WebLogicServer', 'AdminServer','BEA-000365','Server state changed to RESUMING'); Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO wlslog VALUES('Apr-8-2014-7:06:20-PM-PDT','Notice','WebLogicServer', 'AdminServer','BEA-000331','Started WebLogic AdminServer'); Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO wlslog VALUES('Apr-8-2014-7:06:21-PM-PDT','Notice','WebLogicServer', 'AdminServer','BEA-000365','Server state changed to RUNNING'); Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO wlslog VALUES('Apr-8-2014-7:06:22-PM-PDT','Notice','WebLogicServer', 'AdminServer','BEA-000360','Server started in RUNNING mode'); Query OK, 1 row affected (0.00 sec)

Run a SQL query to list the database table data.

mysql> SELECT * FROM wlslog;

Apr-8-2014-7:06:16-PM-PDTNoticeWebLogicServerAdminServerBEA-000365Server state changed to STANDBYApr-8-2014-7:06:17-PM-PDTNoticeWebLogicServerAdminServerBEA-000365Server state changed to STARTINGApr-8-2014-7:06:18-PM-PDTNoticeWebLogicServerAdminServerBEA-000365Server state changed to ADMINApr-8-2014-7:06:19-PM-PDTNoticeWebLogicServerAdminServerBEA-000365Server state changed to ADMINApr-8-2014-7:06:20-PM-PDTNoticeWebLogicServerAdminServerBEA-000365Server state changed to RESUMINGApr-8-2014-7:06:21-PM-PDTNoticeWebLogicServerAdminServerBEA-000365Server state changed to RUNNINGApr-8-2014-7:06:22-PM-PDTNoticeWebLogicServerAdminServerBEA-000365Server state changed to RUNNINGApr-8-2014-7:06:22-PM-PDTNoticeWebLogicServerAdminServerBEA-000365Server state changed to RUNNING	time_stamp	category	type	servername	code	msg
	Apr-8-2014-7:06:16-PM-PDT	Notice	WebLogicServer	AdminServer	BEA-000365	Server state changed to STANDBY
	Apr-8-2014-7:06:17-PM-PDT	Notice	WebLogicServer	AdminServer	BEA-000365	Server state changed to STARTING
	Apr-8-2014-7:06:13-PM-PDT	Notice	WebLogicServer	AdminServer	BEA-000365	Server state changed to ADMIN
	Apr-8-2014-7:06:19-PM-PDT	Notice	WebLogicServer	AdminServer	BEA-000365	Server state changed to RESUMING
	Apr-8-2014-7:06:20-PM-PDT	Notice	WebLogicServer	AdminServer	BEA-000331	Started WebLogic AdminServer
	Apr-8-2014-7:06:21-PM-PDT	Notice	WebLogicServer	AdminServer	BEA-000365	Server state changed to RUNNING
	Apr-8-2014-7:06:22-PM-PDT	Notice	WebLogicServer	AdminServer	BEA-000365	Server started in RUNNING mode

7 rows in set (0.00 sec)

Exit the MySQL CLI and the bash shell using the exit command.

mysql> exit Bye root@15e3253f69f1:/# exit exit

Updating a Service

A service may be updated subsequent to being created with the docker service update command, which has the following syntax:

docker service update [OPTIONS] SERVICE

Some of the supported options are listed in Table 4-4.

Table 4-4. Options for the docker service update Command

Option	Description
args	Args for the command.
constraint-add	Adds or updates a placement constraint.
constraint-rm	Removes a placement constraint.
container-label-add	Adds or updates a Docker container label.
container-label-rm	Removes a container label by its key.
env-add	Adds or updates an environment variable.
env-rm	Removes an environment variable.
force	Whether to force an update even if no changes require it. Option added in Docker 1.13. Default is false.
group-add	Adds an additional supplementary user group to the container. Option added in Docker 1.13.
group-rm	Removes a previously added supplementary user group from the container. Option added in Docker 1.13.
host-add	Adds or updates a custom host-to-IP mapping (host:ip). Option added in Docker 1.13.
host-rm	Removes a custom host-to-IP mapping (host:ip). Option added in Docker 1.13.
hostname	Updates the container hostname. Option added in Docker 1.13.
image	Updates the service image tag.
label-add	Adds or updates a service label.
label-rm	Removes a label by its key.
limit-cpu	Updates the limit CPUs. Default value is 0.000.
limit-memory	Updates the limit memory. Default value is 0.
log-driver	Updates logging driver for service.
log-opt	Updates logging driver options.
mount-add	Adds or updates a mount on a service.
mount-rm	Removes a mount by its target path.
publish-add	Adds or updates a published port.
publish-rm	Removes a published port by its target port.

(continued)

Option	Description
read-only	Mounts the container's root filesystem as read only. Option added in Docker 17.06. Default is false.
replicas	Updates the number of tasks.
reserve-cpu	Updates the reserve CPUs. Default is 0.000.
reserve-memory	Updates the reserve memory. Default is 0.
restart-condition	Updates the restart when condition is met (none, on-failure, or any).
restart-delay	Updates the delay between restart attempts (ns us ms s m h).
restart-max-attempts	Updates the maximum number of restarts before giving up.
rollback	Whether to roll back to a previous specification. Option added in Docker 1.13. Default is false.
tty, -t	Whether to allocate a pseudo-TTY. Option added in Docker 1.13. Default is false.
update-delay	Updates delay between updates (ns us ms s m h). Default is 0s.
update-failure-action	Updates action on update failure (pause continue). Default is pause.
update-monitor	Duration after each task update to monitor for failure (ns us ms s m h). Option added in Docker 1.13. Default 0s.
update-parallelism	Updates the maximum number of tasks updated simultaneously (0 to update all at once). Default is 1.
user, -u	Adds the username or UID (format: <name uid>[:<group gid>]).</group gid></name uid>
workdir, -w	Updates the working directory inside the container.

Next, we update some of the parameters of a deployed service.

Updating the Replicas

First, create a mysql service to update.

```
docker service create \
    --env MYSQL_ROOT_PASSWORD='mysql'\
    --replicas 1 \
    --restart-condition on-failure \
    --restart-max-attempts 5 \
    -update-failure-action continue \
    -name mysql \
    -update-delay 10s \
    mysql:5.6
```

A service from Docker image mysql:5.6 is created and the service ID is output.

```
~ $ docker service rm mysql
mysql
~ $ docker service create \
    --env MYSQL ROOT PASSWORD='mysql'\
>
    --replicas 1 \
>
   --restart-condition on-failure \
>
   --restart-max-attempts 5 \
>
   --update-failure-action continue \
>
    --name mvsal \
>
   --update-delay 10s \
>
> mysql:5.6
mecdt3zluvlvxqc3hdpw8edg1
```

Update the number of replicas to five using the docker service update command. If the command is successful, the service name is output from the command.

```
~ $ docker service update --replicas 5 mysql mysql
```

Setting replicas to five does not just start four new tasks to make a total of five tasks. When a service is updated to change the number of replicas, all the service tasks are shut down and new tasks are started. Subsequently listing the service tasks lists the first task as being shut down and five new tasks as being started.

~ \$ docker service	e ps mysql			
ID	NAME	IMAGE		NODE
DESIRED STATE	CURRENT STATE		ERROR	PORTS
jenOfmkjj13k	mysql.1	mysql:5.6		ip-172-31-37-135.ec2.internal
Running	Starting less th	an a second ago		
r616gx588opd	_ mysql.1	mysql:5.6		ip-172-31-37-135.ec2.internal
Shutdown	Failed 5 seconds	ago	"task:	: non-zero exit (137)"
y350n4e8furo	mysql.2	mysql:5.6		ip-172-31-13-155.ec2.internal
Running	Running 7 second	s ago		
ktrwxnn13fug	mysql.3	mysql:5.6		ip-172-31-37-135.ec2.internal
Running	Running 14 secon	ds ago		
2t8j1zd8uts1	mysql.4	mysql:5.6		ip-172-31-10-132.ec2.internal
Running	Running 10 secon	ds ago		
8tf0uuwb8i31	mysql.5	mysql:5.6		ip-172-31-10-132.ec2.internal
Running	Running 10 secon	ds ago		

Updating the Docker Image Tag

Starting with a MySQL database service called mysql for Docker image mysql:5.6, next we update the service to a different Docker image tag—the mysql:latest Docker image. Run the following command to update the Docker image; the service name is output to indicate that the update is successful.

```
~ $ docker service update --image mysql:latest mysql
mysql
```

You can list detailed information about the service with the docker service inspect command. The image listed in the ContainerSpec is mysql:latest. The PreviousSpec is also listed.

```
~ $ docker service inspect mysql
ſ
    {
         "Spec": {
            "Name": "mysql",
            "Labels": {},
            "TaskTemplate": {
                "ContainerSpec": {
                    "Image": "mysql:latest@sha256:75c563c474f1adc149978011fedfe2e6670483d133
                    b22b07ee32789b626f8de3",
                    "Env": [
                         "MYSOL ROOT PASSWORD=mysql"
                    ],
        "PreviousSpec": {
            "Name": "mysql",
            "Labels": {},
            "TaskTemplate": {
                "ContainerSpec": {
                    "Image": "mysql:5.6@sha256:6ad5bd392c9190fa92e65fd21f6debc8b2a76fc54f139
                    49f9b5bc6a0096a5285",
```

]

The update does not get completed immediately even though the docker service update command does. While the service is being updated, the UpdateStatus for the service is listed with State set to "updating" and the Message of "update in progress".

```
"UpdateStatus": {
    "State": "updating",
    "StartedAt": "2017-07-23T19:24:15.539042747Z",
    "Message": "update in progress"
    }
```

When the update completes, the UpdateStatus State becomes "completed" and the Message becomes "update completed".

```
"UpdateStatus": {
    "State": "completed",
    "StartedAt": "2017-07-23T19:24:15.539042747Z",
    "CompletedAt": "2017-07-23T19:25:25.660907984Z",
    "Message": "update completed"
}
```

While the service is updating, the service tasks are shutting down and the new service tasks are starting. When the update is starting, some of the running tasks might be based on the previous image mysql:5.6 whereas others could be based on the new image mysql:latest.

~ \$ docker service	ps mysql		
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
jenOfmkjj13k	mysql.1	mysql:5.6	ip-172-31-37-135.ec2.internal
Running	Running 38 seconds a	ago	
r616gx588opd	_ mysql.1	mysql:5.6	ip-172-31-37-135.ec2.internal
Shutdown	Failed 43 seconds ag	go "task: non-zero	o exit (137)"
y350n4e8furo	mysql.2	mysql:5.6	ip-172-31-13-155.ec2.internal
Running	Running 45 seconds a	ago	
bswz4sm8e3vj	mysql.3	mysql:5.6	ip-172-31-37-135.ec2.internal
Running	Running 6 seconds ag	go	
ktrwxnn13fug	_ mysql.3	mysql:5.6	ip-172-31-37-135.ec2.internal
Shutdown	Failed 12 seconds ag	go "task: non-zero	p exit (1)"
wj1x26wvp0pt	mysql.4	mysql:latest	ip-172-31-13-155.ec2.internal
Running	Running 7 seconds ag	go	
2t8j1zd8uts1	_ mysql.4	mysql:5.6	ip-172-31-10-132.ec2.internal
Shutdown	Shutdown 7 seconds a	ago	
hppq840ekrh7	mysql.5	mysql:latest	ip-172-31-10-132.ec2.internal
Running	Running 2 seconds ag	go	
8tfOuuwb8i31	_ mysql.5	mysql:5.6	ip-172-31-10-132.ec2.internal
Shutdown	Failed 8 seconds age	o "task: non-zero	o exit (1)"

The desired state of the tasks with image mysql:5.6 is set to Shutdown. Gradually, all the new service tasks based on the new image mysql:latest are started.

_		
ps mysql		
NAME	IMAGE	NODE
CURRENT STATE	ERROR	PORTS
mysql.1	mysql:latest	ip-172-31-37-135.ec2.internal
Running 30 seconds a	ago	
_ mysql.1	mysql:5.6	ip-172-31-37-135.ec2.internal
Failed 36 seconds ag	go "task: non-z	zero exit (137)"
_ mysql.1	mysql:5.6	ip-172-31-37-135.ec2.internal
Failed about a minut	te ago "task: non-z	zero exit (137)"
mysql.2	<pre>mysql:latest</pre>	ip-172-31-13-155.ec2.internal
Ready 3 seconds ago		
_ mysql.2	mysql:5.6	ip-172-31-13-155.ec2.internal
Failed 4 seconds ago	o "task: non-z	zero exit (137)"
mysql.3	mysql:latest	ip-172-31-37-135.ec2.internal
Running 12 seconds a	ago	
_ mysql.3	mysql:5.6	ip-172-31-37-135.ec2.internal
Shutdown 12 seconds	ago	
_ mysql.3	mysql:5.6	ip-172-31-37-135.ec2.internal
Failed 48 seconds ag	go "task: non-z	zero exit (1)"
mysql.4	mysql:latest	ip-172-31-13-155.ec2.internal
Running 44 seconds a	ago	
	os mysql NAME CURRENT STATE mysql.1 Running 30 seconds a _mysql.1 Failed 36 seconds ag _mysql.2 Ready 3 seconds ago _mysql.2 Failed 4 seconds ag mysql.3 Running 12 seconds a _mysql.3 Shutdown 12 seconds a _mysql.3 Failed 48 seconds ag mysql.4 Running 44 seconds ag	<pre>bs mysql NAME IMAGE CURRENT STATE ERROR mysql.1 mysql:latest Running 30 seconds ago _mysql.1 mysql:5.6 Failed 36 seconds ago "task: non-2 mysql.1 mysql:5.6 Failed about a minute ago "task: non-2 mysql.2 mysql:latest Ready 3 seconds ago _mysql.2 mysql:5.6 Failed 4 seconds ago "task: non-2 mysql.3 mysql:latest Running 12 seconds ago _mysql.3 mysql:5.6 Failed 48 seconds ago "task: non-2 mysql.4 mysql:latest Running 44 seconds ago</pre>

2t8j1zd8uts1	_ mysql.4	mysql:5.6	ip-172-31-10-132.ec2.internal
Shutdown	Shutdown 44 seco	nds ago	
hppq840ekrh7	mysql.5	mysql:latest	ip-172-31-10-132.ec2.internal
Running	Running 39 secon	ds ago	
8tf0uuwb8i31	_ mysql.5	mysql:5.6	ip-172-31-10-132.ec2.internal
Shutdown	Failed 44 second	s ago "task: ı	non-zero exit (1)"

Filtering the service tasks with the -f option was introduced earlier. To find which, if any, tasks are scheduled on a particular node, you run the docker service ps command with the filter set to the node. Filtered tasks, both Running and Shutdown, are then listed.

~ \$ docker service	ps -f node=ip-172-3:	1-13-155.ec2.internal	L mysql
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
mkv95bvx3sl1	mysql.2	mysql:latest	ip-172-31-13-155.ec2.internal
Running	Running about a min	ute ago	
y350n4e8furo	_ mysql.2	mysql:5.6	ip-172-31-13-155.ec2.internal
Shutdown	Failed about a minut	te ago "task: non-	-zero exit (137)"
oksssg7gsh79	mysql.4	mysql:latest	ip-172-31-13-155.ec2.internal
Running	Running 50 seconds a	ago	
wj1x26wvp0pt	_ mysql.4	<pre>mysql:latest</pre>	ip-172-31-13-155.ec2.internal
Shutdown	Failed 55 seconds ag	go "task: non-	-zero exit (1)"

Service tasks may also be filtered by desired state. To list only running tasks, set the desired-state filter to running.

~ \$ docker service	e ps -f desired-stat	te=running mysql	
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
2uafxtcbj9qj	mysql.1	<pre>mysql:latest</pre>	ip-172-31-37-135.ec2.internal
Running	Running 3 minutes	s ago	
mkv95bvx3sl1	mysql.2	<pre>mysql:latest</pre>	ip-172-31-13-155.ec2.internal
Running	Running 2 minutes	s ago	
yevunzer12vm	mysql.3	<pre>mysql:latest</pre>	ip-172-31-37-135.ec2.internal
Running	Running 2 minutes	s ago	
oksssg7gsh79	mysql.4	mysql:latest	ip-172-31-13-155.ec2.internal
Running	Running 2 minutes	s ago	
hppq840ekrh7	mysql.5	mysql:latest	ip-172-31-10-132.ec2.internal
Running	Running 3 minutes	s ago	

Likewise, only the shutdown tasks are listed by setting the desired-state filter to shutdown.

~ \$ docker service	os -f desired-stat	e=shutdown myso	l
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
jen0fmkjj13k	mysql.1	mysql:5.6	ip-172-31-37-135.ec2.internal
Shutdown	Failed 3 minutes	ago "task:	non-zero exit (137)"
r616gx588opd	_ mysql.1	mysql:5.6	ip-172-31-37-135.ec2.internal
Shutdown	Failed 3 minutes	ago "task:	non-zero exit (137)"
y350n4e8furo	mysql.2	mysql:5.6	ip-172-31-13-155.ec2.internal
Shutdown	Failed 2 minutes	ago "task:	non-zero exit (137)"

bswz4sm8e3vj	mysql.3	mysql:5.6	ip-172-31-37-135.ec2.internal
Shutdown	Shutdown 2 minu	ites ago	
ktrwxnn13fug	_ mysql.3	mysql:5.6	ip-172-31-37-135.ec2.internal
Shutdown	Failed 3 minute	s ago "task: non-	-zero exit (1)"
wj1x26wvp0pt	mysql.4	mysql:latest	ip-172-31-13-155.ec2.internal
Shutdown	Failed 2 minute	s ago "task: non-	-zero exit (1)"
2t8j1zd8uts1	_ mysql.4	mysql:5.6	ip-172-31-10-132.ec2.internal
Shutdown	Shutdown 3 minu	ites ago	
8tfOuuwb8i31	mysql.5	mysql:5.6	ip-172-31-10-132.ec2.internal
Shutdown	Failed 3 minute	s ago "task: non-	-zero exit (1)"

Updating the Placement Constraints

The placement constraints may be added/removed with the --constraint-add and --constraint-rm options. We started with a Swarm consisting of three nodes—one manager and two worker nodes. We then promoted a worker node to a manager, resulting in a Swarm with two manager nodes and one worker node.

Starting with service replicas running across the Swarm nodes, the replicas may be constrained to run on only worker nodes with the following command. The docker service update command outputs the service name if successful.

```
~ $ docker service update --constraint-add "node.role==worker" mysql
mysql
```

It may take a while (a few seconds or minutes) for the desired state of a service to be reconciled, during which time tasks could be running on manager nodes even though the node.role is set to worker or less than the required number of tasks could be running. When the update has completed (the update status may be found from the docker service inspect command), listing the running tasks for the mysql service indicates that the tasks are running only on the worker nodes.

~ \$ docker servic	e ps -f desired-sta	te=running mysql	
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
smk5q4nhu1rw	mysql.1	mysql:latest	ip-172-31-37-135.ec2.internal
Running	Running about a	minute ago	
wzmou8f6r2tg	mysql.2	mysql:latest	ip-172-31-37-135.ec2.internal
Running	Running 23 secon	ds ago	
byavev89hukv	mysql.3	mysql:latest	ip-172-31-37-135.ec2.internal
Running	Running 23 secon	ds ago	
erx409p0sgcc	mysql.4	mysql:latest	ip-172-31-37-135.ec2.internal
Running	Running 53 secon	ds ago	
q7eqw8jlqig8	mysql.5	mysql:latest	ip-172-31-37-135.ec2.internal
Running	Running 46 secon	ds ago	

As another example, service tasks for the mysql service may be constrained to run on only manager nodes. Starting with service tasks running on both manager and worker nodes and with no other constraints added, run the following command to place all tasks on the manager nodes.

~ \$ docker service update --constraint-add 'node.role==manager' mysql
mysql

The tasks are not shut down on worker nodes and started on manager nodes immediately and initially may continue to be running on worker nodes.

List the service replicas again after a while. You'll see that all the tasks are listed as running on the manager nodes.

~ \$ docker service	<pre>ps -f desired-state=</pre>	running mysql	
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
7tj8bck4jr5n	mysql.1	mysql:latest	ip-172-31-13-155.ec2.internal
Running	Running 14 seconds	ago	
uyeu3y67v2rt	mysql.2	<pre>mysql:latest</pre>	ip-172-31-10-132.ec2.internal
Running	Running about a min	ute ago	
lt9p7479lkta	mysql.3	mysql:latest	ip-172-31-10-132.ec2.internal
Running	Running 1 second ag	0	
t7d9c4viuo5y	mysql.4	mysql:latest	ip-172-31-13-155.ec2.internal
Running	Running 40 seconds	ago	
8xufz871yx1x	mysql.5	mysql:latest	ip-172-31-13-155.ec2.internal
Running	Running 27 seconds	ago	

Updating Environment Variables

The --env-add and --env-rm options are used to add/remove environment variables to/from a service. The mysql service we created includes only one environment variable—the mandatory MYSQL_ROOT_PASSWORD variable. You can use the docker service update command to add the environment variables MYSQL_ DATABASE, MYSQL_PASSWORD, and MYSQL_ALLOW_EMPTY_PASSWORD and to update MYSQL_ROOT_PASSWORD in the same command to an empty password. The command outputs the service name if successful.

```
~ $ docker service update --env-add 'MYSQL _DATABASE=mysql' --env-add 'MYSQL_
PASSWORD=mysql' --env-add 'MYSQL_ALLOW_EMPTY_PASSWORD=yes' --env-add 'MYSQL_ROOT_
PASSWORD=yes' mysql
mysql
```

When the update has completed, the docker service inspect command lists the environment variables added.

Updating the environment variables causes the containers to restart. So, simply adding environment variables doesn't cause the new database to be created in the same container. A new container is started with the updated environment variables.

Updating the Docker Image

The Docker image may also be updated, not just the image tag. As an example, update the Docker image for a MySQL database service to use the postgres Docker image, which is for the PostgreSQL database. The command outputs the service name if the update is successful.

```
~ $ docker service update --image postgres mysql
mysql
```

After the update has completed, showing the running service tasks lists new tasks for the postgres image. The service name stays the same and the Docker image is updated to postgres.

~ \$ docker service	e ps -f desired-sta	te=running mysql	
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
hmk7128ls19a	mysql.1	postgres:latest	ip-172-31-13-155.ec2.internal
Running	Running 18 second	ds ago	
5ofbkc82gp0i	mysql.2	<pre>postgres:latest</pre>	ip-172-31-10-132.ec2.internal
Running	Running about a r	minute ago	
vOgfc65lhw62	mysql.3	<pre>postgres:latest</pre>	ip-172-31-13-155.ec2.internal
Running	Running 31 second	ds ago	
miscjf9n66qq	mysql.4	<pre>postgres:latest</pre>	ip-172-31-13-155.ec2.internal
Running	Running 45 second	ds ago	
g5viy8jyzpi1	mysql.5	postgres:latest	ip-172-31-10-132.ec2.internal
Running	Running about a r	minute ago	

Updating the Docker image does not remove the environment variables associated with the mysql Docker image, which are still listed in the service detail.

```
~ $ docker service inspect mysql
[
...
"Spec": {
    "Name": "mysql",
...
"ContainerSpec": {
    "Env": [
    "MYSQL_ROOT_PASSWORD=yes",
    "MYSQL_DATABASE=mysql",
    "MYSQL_PASSWORD=mysql",
    "MYSQL_ALLOW_EMPTY_PASSWORD=yes"
    ],
...
```

```
]
```

The added environment variables for the MySQL database need to be removed, as the PostgreSQL database Docker image postgres does not use the same environment variables. Remove all the environment variables from the mysql service with the --env-rm option to the docker service update command. To remove only the env variable, the name needs to be specified, not the env value.

```
docker service update --env-rm 'MYSQL_DATABASE' --env-rm 'MYSQL_PASSWORD' --env-rm
'MYSQL_ALLOW_EMPTY_PASSWORD' --env-rm 'MYSQL_ROOT_PASSWORD' mysql
```

Updating the Container Labels

The --container-label-add and --container-label-rm options are used to update the Docker container labels for a service. To add a container label to the mysql service, run a docker service update command, which outputs the service name if successful.

```
~ $ docker service update --container-label-add 'com.docker.swarm.service.version=latest'
mysql
mysql
```

шузчт

On listing detailed information about the service, the added label is listed in the ContainerSpec labels.

```
~ $ docker service inspect mysql
[
...
"ContainerSpec": {
    "Labels": {
        "com.docker.swarm.service.version": "latest"
        },
...
]
```

The label added may be removed with the --container-label-rm option. To remove only the label, the key needs to be specified, not the label value.

```
~ $ docker service update --container-label-rm 'com.docker.swarm.service.version' mysql
mysql
```

Updating Resources Settings

The --limit-cpu, --limit-memory, --reserve-cpu, and --reserve-memory options of the docker service update command are used to update the resource settings for a service. As an example, update the resource limits and reserves. The command outputs the service name if successful.

```
~ $ docker service update --limit-cpu 0.5 --limit-memory 1GB --reserve-cpu
"0.5" --reserve-memory "1GB" mysql
mysql
```

The resources settings are updated. Service detail lists the updated resource settings in the Resources JSON object.

```
"NanoCPUs": 50000000,
"MemoryBytes": 1073741824
}
},
....
```

Removing a Service

The docker service rm command removes a service. If the output of the command is the service name, the service has been removed. All the associated service tasks and Docker containers also are removed.

```
~ $ docker service rm mysql
mysql
```

Creating a Global Service

As discussed earlier, a service has two modes—*replicated* or *global*. The default mode is replicated. The mode may also be explicitly set to replicated with the --mode option of the docker service create command. The service mode cannot be updated after a service has been created, with the docker service update command for example. Create a replicated service for nginx using the --mode option.

```
~ $ docker service create --mode replicated --name nginx nginx
no177eh3gxsyemb1gfzc99mmd
```

A replicated mode service is created with the default number of replicas, which is 1. List the services with the docker service 1s command. The nginx service is listed with one replica.

~ \$ docker servi	ce ls				
ID	NAME	MODE	REPLICAS	IMAGE	PORTS
no177eh3gxsy	nginx	replicated	1/1	nginx:latest	

A global service runs one task on each node in a Swarm by default. A global service may be required at times such as for an agent (logging/monitoring) that needs to run on each node. A global service is used for logging in Chapter 11. Next, we create a nginx Docker image-based service that's global. Remove the replicated service nginx with the docker service rm nginx command. A service name must be unique even if different services are of different modes. Next, create a global mode nginx service with the same command as for the replicated service, except that the --mode option is set to global instead of replicated.

```
~ $ docker service create --mode global --name nginx nginx
5prj6c4v4be6ga0odnb22qa4n
```

A global mode service is created. The docker service 1s command lists the service. The REPLICAS column for a global service does not list the number of replicas, as no replicas are created. Instead global is listed in the REPLICAS column.

~ \$ docker service]	ls				
ID	NAME	MODE	REPLICAS	IMAGE	PORTS
5prj6c4v4be6	nginx	global	3/3	nginx:latest	

A service task is created for a global service on each node in the Swarm on which a task can run. Scheduling constraints may be used with a global service to prevent running a task on each node. Scheduling is discussed in Chapter 8. Global services cannot be scaled.

Summary

This chapter introduced Docker services running on a Docker Swarm. A service consists of service tasks or replicas. A Docker Swarm supports two types of services—*replicated services* and *global services*. A replicated service has the assigned number of replicas and is scalable. A global service has a task on each node in a Swarm. The term "replica" is used in the context of a replicated service to refer to the service tasks that are run across the nodes in a Swarm. A replicated service could run a specified number of tasks for a service, which could imply running no tasks or running multiple tasks on a particular node. The term "replica" is generally not used in the context of a global service, which runs only one task on each node in the Swarm. Each task (replica) is associated with a Docker container. We started with a Hello World service and invoked the service with curl on the command line and in a browser. Subsequently, we discussed a service for a MySQL database. We started a bash shell for a MySQL service container and created a database table. Scaling, updating, and removing a service are some of the other service features this chapter covered. The chapter concluded by creating a global service. The next chapter covers the Docker Swarm scaling service in more detail.

CHAPTER 5

Scaling Services

Docker Engine is suitable for developing lightweight applications that run in Docker containers that are isolated from each other. Docker containers are able to provide their own networking and filesystem.

The Problem

Docker Engine (prior to native Swarm mode) was designed to run Docker containers that must be started separately. Consider the use case that multiple replicas or instances of a service need to be created. As client load on an application running in a Docker container increases, the application may need to be run on multiple nodes. A limitation of Docker Engine is that the docker run command must be run each time a Docker container is to be started for a Docker image. If a Docker application must run on three nodes, the docker run command must run on each of the nodes as well, as illustrated in Figure 5-1. No provision to scale an application or run multiple replicas is provided in the Docker Engine (prior to Docker 1.12 native Swarm mode support).



Figure 5-1. Docker engine without provision for scaling

The Solution

The Docker Swarm mode has the provision to scale a Docker service. A service abstraction is associated with zero or more replicas (tasks) and each task starts a Docker container for the service. The service may be scaled up or down to run more/fewer replicas, as required. With a single docker service scale <svc>=<replicas> command, a service can run the required number of replicas, as illustrated in Figure 5-2. If 10 service replicas are to be started across a distributed cluster, a single command is able to provision scaling.



Figure 5-2. Docker Swarm mode with provision for scaling

Scaling is supported only for replicated services. A global service runs one service task on each node in a Swarm. Scaling services was introduced in Chapter 3 and, in this chapter, we discuss some of the other aspects of scaling services not discussed in Chapter 3. This chapter covers the following topics:

- Setting the environment
- Creating a replicated service
- Scaling up a service
- Scaling down a service
- Removing a service
- Global services cannot be scaled
- Scaling multiple services in the same command
- Service replicas replacement on a node leaving the Swarm

www.allitebooks.com

Setting the Environment

Create a three-node Swarm on Docker for Swarm, which is discussed in Chapter 3. A Docker for AWS Swarm you created in another chapter may be used in this chapter. Obtain the public IP address of the EC2 instance for the Swarm manager.

SSH login to the Swarm manager EC2 instance with user "docker".

[root@localhost ~]# ssh -i "docker.pem" docker@34.200.225.39
Welcome to Docker!

The docker node 1s command lists the nodes in the Swarm.

~ \$ docker node ls

IDHOSTNAMESTATUSAVAILABILITYMANAGERSTATUSilru4f0i280w2tlsrg9hglwsjip-172-31-10-132.ec2.internalReadyActiveActivew5t0186ipblpcq390625wyq2eip-172-31-37-135.ec2.internalReadyActiveLeaderzkxle7kafwcmt1sd93kh5cy5e *ip-172-31-13-155.ec2.internalReadyActiveLeader

Creating a Replicated Service

As discussed in Chapter 4, Docker Swarm mode supports two types of services—global and replicated. The default is the replicated mode. Only the replicated service can be scaled. Next, create a replicated service for MySQL database using the docker service create command, initially consisting of one replica, as specified in the --replicas option. The default number of replicas if the --replicas option is not specified is also one.

```
~ $ docker service create \
> --env MYSQL_ROOT_PASSWORD='mysql'\
> --replicas 1 \
> --name mysql \
> mysql
ndu4kwqk9ol7e7wxvv5bremr4
```

List the services using docker service ls.

~ \$ docker serv:	ice ls				
ID	NAME	MODE	REPLICAS	IMAGE	PORTS
ndu4kwqk9ol7	mysql	replicated	1/1	mysql:latest	

As service replicas take a while (albeit a few seconds) to start, initially 0/1 replicas could be listed in the REPLICAS column, which implies that the desired state of running one service replica has not been achieved yet. Run the same command after a few seconds and 1/1 REPLICAS should be listed as running.

Optionally, the docker service create command may also be run by setting the --mode option. Remove the mysql service if it was created previously and use the --mode option as follows.

```
~ $ docker service rm mysql
mysql
~ $ docker service create \
> --mode replicated \
> --env MYSQL_ROOT_PASSWORD='mysql'\
> --replicas 1 \
> --name mysql \
> mysql
rl2s2ptgbs9z2t7fy5e63wf2j
```

The mysql service is created as without the --mode replicated option. List the service replicas or tasks with docker service ps mysql. A single replica is listed.

~ \$ docker service ps mysql ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR PORTS yrikmh7mciv7 mysql.1 mysql: ip-172-31-13- Running Running 21 latest 155.ec2.internal seconds ago

One service replica is created by default if the --replicas option is omitted. It should be mentioned that running multiple replicas of the MySQL database does not automatically imply that they are sharing data, so accessing one replica will not give you the same data as another replica. Sharing data using mounts is discussed in Chapter 6.

Scaling Up a Service

The docker service scale command, which has the following syntax, may be used to scale up/down a service, which changes the desired state of the service.

```
docker service scale SERVICE=REPLICAS [SERVICE=REPLICAS...]
```

First, scale up the service to three replicas.

```
~ $ docker service scale mysql=3
mysql scaled to 3
```

Subsequently, three tasks are listed as scheduled on the three nodes in the Swarm.

```
~ $ docker service ps mysql
             NAME
                      IMAGE
                              NODE
                                               DESIRED STATE CURRENT STATE
                                                                              ERROR
                                                                                      PORTS
TD
yrikmh7mciv7 mysql.1 mysql:
                              ip-172-31-13-
                                                             Running 37
                                                Running
                      latest 155.ec2.internal
                                                             seconds ago
3zxmotmy6n2t
             mysql.2 mysql: ip-172-31-37-
                                               Running
                                                             Running 7
                                                             seconds ago
                       latest 135.ec2.internal
rdfsowttd3b9 mysql.3
                      mysql: ip-172-31-10-
                                               Running
                                                             Running 7
                       latest 132.ec2.internal
                                                             seconds ago
```

In addition to one replica on the manager node, one replica each is started on each of the two worker nodes. If the docker ps command is run on the manager node, only one Docker container for the mysql Docker image is listed.

~ \$ docker ps						
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
6d2161a3b282	mysql: latest	"docker- entrypoint"	50 seconds ago	Up 49 seconds	3306/tcp	mysql.1.yrikmh7mci v7dsmql1nhdi62l

A service may also be scaled using the docker service update command with the --replicas option. As an example, scale it to 50 replicas.

```
\sim $ docker service update --replicas=50 mysql mysql
```

The service is scaled to 50 replicas and, subsequently, 50 service tasks are listed.

~ \$ docker service	ps -f desired-state	=running mysql	NODE
ID	NAME	IMAGE	NODE
DESTRED STATE		EKKUK	PURIS
t026kjbsgzmq	mysql.1	mysql:latest	1p-172-31-37-135.ec2.internal
Running	Running 11 seconds	ago	
+3tx2kbe55dh	mysq1.2	mysql:latest	1p-172-31-10-132.ec2.internal
Running	Running 20 seconds	ago	
5mzej75us115	mysql.3	mysql:latest	1p-172-31-10-132.ec2.internal
Running	Running 13 seconds	ago	
wluix1b3z863	mysql.4	_mysql:latest	ip-172-31-13-155.ec2.internal
Running	Preparing 13 second	is ago	
91d8smvahk9g	mysql.5	mysql:latest	ip-172-31-13-155.ec2.internal
Running	Running 47 seconds	ago	
3tgw8ni5mfi1	mysql.6	mysql:latest	ip-172-31-10-132.ec2.internal
Running	Running 46 seconds	ago	
1gm8e7pxkg0o	mysql.7	mysql:latest	ip-172-31-13-155.ec2.internal
Running	Running 46 seconds	ago	
iq5p2g48oagq	mysql.8	mysql:latest	ip-172-31-37-135.ec2.internal
Running	Running 45 seconds	ago	
i4yh072h1gs6	mysql.9	mysql:latest	ip-172-31-13-155.ec2.internal
Running	Running 46 seconds	ago	
r1z5tgu0dg13	mysql.10	mysql:latest	ip-172-31-13-155.ec2.internal
Running	Running 45 seconds	ago	
mekfjvxi9pds	mysql.11	mysql:latest	ip-172-31-10-132.ec2.internal
Running	Running 46 seconds	ago	
nd8f2pr4oivc	mysql.12	mysql:latest	ip-172-31-13-155.ec2.internal
Running	Running 45 seconds	ago	
xou9hztlj637	mysql.13	mysql:latest	ip-172-31-13-155.ec2.internal
Running	Running 45 seconds	ago	
t95flokvca2y	mysql.14	mysql:latest	ip-172-31-37-135.ec2.internal
Running	Running 45 seconds	ago	
rda5shwwfmsc	mysql.15	mysql:latest	ip-172-31-37-135.ec2.internal
Running	Running 45 seconds	ago	
ibb2fk2llm3w	mysql.16	mysql:latest	ip-172-31-13-155.ec2.internal
Running	Running 47 seconds	ago	
st4ofpvrfaip	mysql.17	mysql:latest	ip-172-31-13-155.ec2.internal
Running	Running 45 seconds	ago	
iw4daunt6s63	mysql.18	mysql:latest	ip-172-31-37-135.ec2.internal
Running	Running 47 seconds	ago	
vk4nzq7utyl2	mysql.19	mysql:latest	ip-172-31-10-132.ec2.internal
Running	Running 46 seconds	ago	
oj59qjcy51qw	mysql.20	mysql:latest	ip-172-31-37-135.ec2.internal
Running	Running 45 seconds	ago	

wiou769z8xeh Running 5exwimn64w94 Running agqongnh9uu3 Running ynkvjwgqqqlx Running yf87kbsn1cga Running xxqj62007cxd Running 50ym9i8tjwd5 Running 7btl2pga1l5o Running 62dqj60q1ol8 Running psn7zl4th2zb Running khsj2an2f5gk Running rzpndzjpmuj7 Running 9zrcga93u5fi Running x565ry5ugj8m Running o1os5dievj37 Running dritgxq0zrua Running n8hs01m8picr Running dk5w0qnkfb63 Running joii103na4ao Running db5hz7m2vac1 Running ghk6s12eeo48 Running jbi8aksksozs Running rx3rded30oa4 Running c3zaacke440s Running 16ppiurx4306 Running

mysql.21 mysql:latest Running 47 seconds ago mysql.22 mysql:latest Running 48 seconds ago mysql.23 mysql:latest Running 45 seconds ago mysql.24 mysql:latest Running 47 seconds ago mysql.25 mysql:latest Running 10 seconds ago mysql.26 mysql:latest Running 45 seconds ago mysql.27 mysql:latest Running 45 seconds ago mysql.28 mysql:latest Running 46 seconds ago mysql.29 mysql:latest Running 45 seconds ago mysql.30 mysql:latest Preparing 16 seconds ago mysql:latest mysql.31 Running 45 seconds ago mysql.32 mysql:latest Running 45 seconds ago mysql.33 mysql:latest Running 45 seconds ago mysql:latest mysql.34 Running 48 seconds ago mysql.35 mysql:latest Running 46 seconds ago mysql.36 mysql:latest Running 45 seconds ago mysql.37 mysql:latest Running 47 seconds ago mysql.38 mysql:latest Running 45 seconds ago mysql.39 mysql:latest Running 45 seconds ago mysql:latest mysql.40 Running 46 seconds ago mysql.41 mysql:latest Running 45 seconds ago mysql.42 mysql:latest Running 47 seconds ago mysql.43 mysql:latest Running 47 seconds ago mysql.44 mysql:latest Running 45 seconds ago mysql.46 mysql:latest Running 46 seconds ago

ip-172-31-10-132.ec2.internal ip-172-31-10-132.ec2.internal ip-172-31-37-135.ec2.internal ip-172-31-37-135.ec2.internal ip-172-31-13-155.ec2.internal ip-172-31-37-135.ec2.internal ip-172-31-37-135.ec2.internal ip-172-31-10-132.ec2.internal ip-172-31-13-155.ec2.internal ip-172-31-37-135.ec2.internal ip-172-31-37-135.ec2.internal ip-172-31-13-155.ec2.internal ip-172-31-13-155.ec2.internal ip-172-31-10-132.ec2.internal ip-172-31-10-132.ec2.internal ip-172-31-37-135.ec2.internal ip-172-31-37-135.ec2.internal ip-172-31-13-155.ec2.internal ip-172-31-37-135.ec2.internal ip-172-31-13-155.ec2.internal ip-172-31-37-135.ec2.internal ip-172-31-13-155.ec2.internal ip-172-31-37-135.ec2.internal ip-172-31-13-155.ec2.internal ip-172-31-10-132.ec2.internal

CHAPTER 5 SCALING SERVICES

of06zibtlsum	mysql.47	mysql:latest	ip-172-31-10-132.ec2.internal
Running	Running 46 second	s ago	
kgjjwlc9zmp8	mysql.48	mysql:latest	ip-172-31-10-132.ec2.internal
Running	Running 46 second	s ago	
rw1icgkyw61u	mysql.49	mysql:latest	ip-172-31-10-132.ec2.internal
Running	Running 46 second	s ago	
j5jpl9a5jgbj	mysql.50	mysql:latest	ip-172-31-10-132.ec2.internal
Running	Running 47 second	s ago	

A small-scale MySQL database service probably wouldn't benefit from scaling to 50 replicas, but an enterprise-scale application could use 50 or even more replicas.

Scaling Down a Service

A service may be scaled down just as it is scaled up. A service may even be scaled down to no replicas. The mysql service may be scaled down to no replicas by setting the number of replicas to 0 using the docker service update or docker service scale command.

```
~ $ docker service scale mysql=0
mysql scaled to 0
```

The service gets scaled down to no replicas. No service replicas that are running are listed.

~ \$ dock	er service ps -f	desired-state=running mysql		
ID	NAME	IMAGE	NODE	DESIRED STATE
CURRENT	STATE ERROR	PORTS		

The actual service tasks could take a while to shut down, but the desired state of all tasks is set to Shutdown.

Scaling a service to no tasks does not run any tasks, but the service is not removed. The mysql service may be scaled back up again from none to three tasks as an example.

```
~ $ docker service scale mysql=3
mysql scaled to 3
```

Three service tasks start running.

~ \$ docker servio	ce ps -f	desire	d-state=runn	ing mysql	
ID	NAME		IMAGE	NODE	
DESIRED STATE	CURRENT	STATE	ERF	ROR	PORTS
py7aqwy2reku	mysql.1		mysql:latest	t ip-172	-31-37-135.ec2.internal
Running	Running	9 seco	nds ago		
re1l3q3iwmvo	mysql.2		mysql:latest	t ip-172	-31-37-135.ec2.internal
Running	Running	9 seco	nds ago		
h7my2ucpfz3u	mysql.3		mysql:latest	t ip-172	-31-37-135.ec2.internal
Running	Running	9 seco	nds ago		

Removing a Service

A service may be removed using the docker service rm command.

```
~ $ docker service rm mysql
mysql
```

The mysql service is not listed after having been removed.

~ \$ 0	docker service ls				
ID	NAME	MODE	REPLICAS	IMAGE	PORTS

Multiple services may be removed using the docker service rm command. To demonstrate, you can create two services, hello-world and nginx.

```
~ $ docker service create \
    --name hello-world \
>
    --publish 8080:80 \
>
>
    --replicas 2 \
    tutum/hello-world
>
t3msb25rc8b6xcm30k0zoh4ws
~ $ docker service create --name nginx nginx
ncn4aqkgzrcjc8w1uorjo5jrd
~ $ docker service ls
TD
               NAMF
                            MODF
                                         REPLICAS
                                                    TMAGE
                                                                              PORTS
ncn4aqkgzrcj
               nginx
                            replicated
                                         1/1
                                                    nginx:latest
t3msb25rc8b6
              hello-world
                            replicated
                                         2/2
                                                    tutum/hello-world:latest *:8080->80/tcp
```

Subsequently, remove both the services with one docker service rm command. The services removed are output if the command is successful.

```
~ $ docker service rm nginx hello-world
nginx
hello-world
```

Global Services Cannot Be Scaled

A global service creates a service task on each node in the Swarm and cannot be scaled. Create a global service for a MySQL database using the docker service create command. Notable differences in the command are that the --mode is set to global and the --replicas option is not included.

```
~ $ docker service create \
> --mode global \
> --env MYSQL_ROOT_PASSWORD='mysql'\
> --name mysql-global \
> mysql
nxhnrsiulymd9n4171cie9a8j
```

The global service is created and listing the service should indicate a Mode set to global.

~ \$ docker service ls
ID NAME MODE REPLICAS IMAGE PORTS
nxhnrsiulymd mysql-global global 3/3 mysql:latest

One service task is created on each node in the Swarm.

```
~ $ docker service ps mysql-global
ID
                  NAME
                                                          IMAGE
NODE
                             DESIRED STATE
                                             CURRENT STATE
                                                                               PORTS
                                                                    FRROR
nfbmkqdh46k0 mysql-global.zkxle7kafwcmt1sd93kh5cy5e
                                                          mysql:latest
ip-172-31-13-155.ec2.internal Running
                                             Running 22 seconds ago
t55ba3bobwzf mysql-global.w5to186ipblpcq390625wyq2e
                                                          mysql:latest
ip-172-31-37-135.ec2.internal Running
                                             Running 22 seconds ago
kqg656m30lj3
                  mysql-global.ilru4f0i280w2tlsrg9hglwsj
                                                          mysql:latest
ip-172-31-10-132.ec2.internal Running
                                             Running 22 seconds ago
```

If another node is added to the Swarm, a service task automatically starts on the new node. If the docker service scale command is run for the global service, the service does not get scaled. Instead, the following message is output.

```
~ $ docker service scale mysql-global=5
mysql-global: scale can only be used with replicated mode
```

A global service may be removed just as a replicated service, using the docker service rm command.

```
~ $ docker service rm mysql-global
mysql-global
```

Scaling Multiple Services Using the Same Command

Multiple services may be scaled using a single docker service scale command. To demonstrate, create two services: nginx and mysql.

```
~ $ docker service create \
> --replicas 1 \
> --name nginx \
> nginx
u6i4e8eg720dwzz425inhxqrp
~ $ docker service create \
> --env MYSQL_ROOT_PASSWORD='mysql'\
> --name mysql \
> mysql
1umb7e2gr68s54utujr6khjgd
```

List the two services. One replica for each service should be running.

~ \$ docker service ls						
ID	NAME	MODE	REPLICAS	IMAGE	PORTS	
1umb7e2gr68s	mysql	replicated	1/1	mysql:latest		
u6i4e8eg720d	nginx	replicated	1/1	nginx:latest		

CHAPTER 5 SCALING SERVICES

Scale the nginx service and the mysql service with a single command. Different services may be scaled to a different number of replicas.

```
~ $ docker service scale mysql=5 nginx=10
mysql scaled to 5
nginx scaled to 10
```

The mysql service gets scaled to five tasks and the nginx service gets scaled to 10 replicas. Initially, some of the new tasks for a service may not have started, as for the nginx service, which lists only 8 of the 10 tasks as running.

~ \$ docker servi	ice ls				
ID	NAME	MODE	REPLICAS	IMAGE	PORTS
1umb7e2gr68s	mysql	replicated	5/5	<pre>mysql:latest</pre>	
u6i4e8eg720d	nginx	replicated	8/10	nginx:latest	

After a while, all service tasks should be listed as running, as indicated by 10/10 for the nginx service.

~ \$ docker servic	e ls				
ID	NAME	MODE	REPLICAS	IMAGE	PORTS
1umb7e2gr68s	mysql	replicated	5/5	<pre>mysql:latest</pre>	
u6i4e8eg720d	nginx	replicated	10/10	nginx:latest	

The service tasks for the two services may be listed using a single docker service ps command.

~ \$ docker service	ps nginx	mysql			
ID	NAME		IMAGE		NODE
DESIRED STATE	CURRENT	STATE	ER	ROR	PORTS
f9g1tw88nppk	mysql.1		mysql:lates	t	ip-172-31-26-234.ec2.internal
Running	Running	about a mi	nute ago		
zcl1qfdiqrvu	nginx.1		nginx:lates	t	ip-172-31-10-132.ec2.internal
Running	Running	about a mi	nute ago		
vu4xo99xr0y4	nginx.2		nginx:lates	t	ip-172-31-13-155.ec2.internal
Running	Running	40 seconds	ago		
xvxgfoacxjos	mysql.2		mysql:lates	t	ip-172-31-37-135.ec2.internal
Running	Running	41 seconds	ago		
yw0opq5y0x20	nginx.3		nginx:lates	t	ip-172-31-13-155.ec2.internal
Running	Running	41 seconds	ago		
vb92hkua6eyo	mysql.3		mysql:lates	t	ip-172-31-13-155.ec2.internal
Running	Running	40 seconds	ago		
1cnqwtb24zvy	nginx.4		nginx:lates	t	ip-172-31-13-155.ec2.internal
Running	Running	41 seconds	ago		
hclu53xkosva	mysql.4		mysql:lates	t	ip-172-31-26-234.ec2.internal
Running	Running	40 seconds	ago		
2xjcw4i9xw89	nginx.5		nginx:lates	t	ip-172-31-10-132.ec2.internal
Running	Running	41 seconds	ago		
ocvb2qctuids	mysql.5		mysql:lates	t	ip-172-31-10-132.ec2.internal
Running	Running	41 seconds	ago		
l8mlu3jpp9cx	nginx.6		nginx:lates	t	ip-172-31-10-132.ec2.internal
Running	Running	41 seconds	ago		
p84m8yh5if5t	nginx.7 ng	ginx:latest	ip-172-31-37-135.ec2.internal		
--------------	------------------------	-------------	-------------------------------		
Running	Running 41 seconds ago)			
7yp8m7ytt7z4	nginx.8 ng	;inx:latest	ip-172-31-26-234.ec2.internal		
Running	Running 24 seconds ago)			
zegs90r015nn	nginx.9 ng	ginx:latest	ip-172-31-37-135.ec2.internal		
Running	Running 41 seconds ago)			
qfkpvy28g1g6	nginx.10 ng	ginx:latest	ip-172-31-26-234.ec2.internal		
Running	Running 24 seconds ago)	-		

Service Tasks Replacement on a Node Leaving the Swarm

The desired state reconciliation in Docker Swarm mode ensures that the desired number of replicas are running if resources are available. If a node is made to leave a Swarm, the replicas running on the node are scheduled on another node. Starting with a mysql service replica running on each node in a three-node Swarm, you can make one worker node leave the Swarm.

~ docker swarm 1eave Node left the swarm.

A replacement service task for the service task running on the shutdown node gets scheduled on another node.

~ s docker service ps mysql

NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR

6zu7a59ejdxip3y9oeu548hv5 mysql.l mysql ip-10-0-0-46.ec2.internal Running Running 3 minutes ago 441cuufa7sa9möeatqbiq7vi3 mysql.2 mysql ip-10-0-0-28.ec2.internal Running Running about a minute ago blcdm8Bh6v86gl..pwp6zx3janv mysql.3 mysql ip-10-0-0-28.ec2.internal Running Running 4 seconds ago Or3oki4acf3d6ils5iazmg425 _ mysql.3 mysql ip-10-0-0-106.ec2.internal Shutdown Running about a minute ago

Make the other worker node also leave the Swarm. The service replicas on the other worker node also get shut down and scheduled on the only remaining node in the Swarm.

~ s docker service ps mysql

NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR

```
6zu7a59ejdxip3y9oeu548hv5 mysql.1 mysql ip-10-0-0-46.ec2. internal Running Running 5 minutes ago
dbdaxvl6lohlxrsxh5aobjxi8 mysq.2 mysql ip-10-0-0-46.ec2.internal Running Running 7 seconds ago
44tcuufa7sa9m6eatqbiq7vi3 \_ mysql.2 mysql ip-10-0-0-28.ec2.internal Shutdown Running 2 minutes ago
216iu28xh5hztm3bgtvy7ttk8 mysql.3 mysql ip-10-0-0-46.ec2.internal Running Running 7 seconds ago
blcdm88h6v86gLpwp6zx3janv \_ mysql.3 mysql ip-10-0-0-28.ec2.internal Shutdown Running about a minute ago
0r3oki4acf3d6ils5iazmg425 \_ mysql.3 mysql ip-10-0-0-106.ec2.internal Shutdown Running 2 minutes ago
```

If only the replicas with desired state as running are listed, all replicas are listed as running on the manager node.

~s docker service ps -f desired-state=running mysql

ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR

6zu7a59ejdxip3y9oeu548hv5 mysql.1 mysql ip-10-0-0-46.ec2.internal Running Running 7 minutes ago

dbdaxvl6lohlxrsxh5aobjxi8 mysql.2 mysql ip-10-0-0-46.ec2.internal Running Running 2 minutes ago

216iu28xh5hztm3bgtvy7ttk8 mysql.3 mysql ip-10-0-0-46.ec2.internal Running Running 2 minutes ago

Summary

This chapter discussed service scaling in Swarm mode. Only a replicated service can be scaled and not a global service. A service may be scaled up to as many replicas as resources can support and can be scaled down to no replicas. Multiple services may be scaled using the same command. Desire state reconciliation ensures that the desired number of service replicas are running. The next chapter covers Docker service mounts.

CHAPTER 6

Using Mounts

A service task container in a Swarm has access to the filesystem inherited from its Docker image. The data is made integral to a Docker container via its Docker image. At times, a Docker container may need to store or access data on a persistent filesystem. While a container has a filesystem, it is removed once the container exits. In order to store data across container restarts, that data must be persisted somewhere outside the container.

The Problem

Data stored only within a container could result in the following issues:

- The data is not persistent. The data is removed when a Docker container is stopped.
- The data cannot be shared with other Docker containers or with the host filesystem.

The Solution

Modular design based on the Single Responsibility Principle (SRP) recommends that data be decoupled from the Docker container. Docker Swarm mode provides *mounts* for sharing data and making data persistent across a container startup and shutdown. Docker Swarm mode provides two types of mounts for services:

- Volume mounts
- Bind mounts

The default is the volume mount. A mount for a service is created using the --mount option of the docker service create command.

Volume Mounts

Volume mounts are named volumes on the host mounted into a service task's container. The named volumes on the host persist even after a container has been stopped and removed. The named volume may be created before creating the service in which the volume is to be used or the volume may be created at service deployment time. Named volumes created at deployment time are created just prior to starting a service task's container. If created at service deployment time, the named volume is given an auto-generated name if a volume name is not specified. An example of a volume mount is shown in Figure 6-1, in which a named volume mysql-scripts, which exists prior to creating a service, is mounted into service task containers at the directory path /etc/mysql/scripts.

CHAPTER 6 USING MOUNTS



Figure 6-1. Volume mount

Each container in the service has access to the same named volume on the host on which the container is running, but the host named volume could store the same or different data.

When using volume mounts, contents are not replicated across the cluster. For example, if you put something into the mysql-scripts directory you're using, those new files will only be accessible to other tasks running on that same node. Replicas running on other nodes will not have access to those files.

Bind Mounts

Bind mounts are filesystem paths on the host on which the service task is to be scheduled. The host filesystem path is mounted into a service task's container at the specified directory path. The host filesystem path must exist on each host in the Swarm on which a task may be scheduled prior to a service being created. If certain nodes are to be excluded for service deployment, using node constraints, the bind mount host filesystem does not have to exist on those nodes. When using bind mounts, keep in mind that the service using a bind mount is not portable as such. If the service is to be deployed in production, the host directory path must exist on each host in the Swarm in the production cluster.

The host filesystem path does not have to be the same as the destination directory path in a task container. As an example, the host path /db/mysql/data is mounted as a bind mount into a service's containers at directory path /etc/mysql/data in Figure 6-2. A bind mount is read-write by default, but could be made read-only at service deployment time. Each container in the service has access to the same directory path on the host on which the container is running, but the host directory path could store different or the same data.



Figure 6-2. Bind mount

Swarm mode mounts provide shareable named volumes and filesystem paths on the host that persist across a service task startup and shutdown. A Docker image's filesystem is still at the root of the filesystem hierarchy and a mount can only be mounted on a directory path within the root filesystem.

This chapter covers the following topics:

- Setting the environment
- Types of mounts
- Creating a named volume
- Using a volume mount to get detailed info about a volume
- Removing a volume
- Creating and using a bind mount

Setting the Environment

Create a Docker for AWS-based Swarm consisting of one manager node and two worker nodes, as discussed in Chapter 3. The Docker for AWS Swarm will be used for one type of mount, the *volume* mount. For the *bind* mount, create a three-node Swarm consisting of one manager and two worker nodes on CoreOS instances. Creating a Swarm on CoreOS instances is discussed in Chapter 2. A CoreOS-based Swarm is used because Docker for AWS Swarm does not support bind mounts out-of-the-box. Obtain the public IP address of the manager instance for the Docker for AWS Swarm from the EC2 console, as shown in Figure 6-3.

CHAPTER 6 USING MOUNTS

-		anna ann ann ann ann a						0			
Q Filter by tag	s and attributes or se	earch by keyword						0	< <	1 to 3 of 3	> >
Name	* Ins	stance ID 🗸	Instance Type 🔹	Availability Zone +	Instance State ~	Status	s Checks 🔹	Alarm State	IS	Public DNS (IPv4)
Docker-we	orker i-00	83e010a9e00271	t2.micro	us-east-1b	running	O 2/2	2 checks	None	>	ec2-52-91-20	0-241.
Docker-M	anager i-0-	4812ba54e249c99c	t2.micro	us-east-1b	running	O 2/2	2 checks	None	10	ec2-52-91-115	5-180.0
Docker-we	orker i-0a	a0d7ba92b6a454	t2.micro	us-east-1c	running	2/2	2 checks	None	10	ec2-34-229-8	5-64.cd
Description	Status Checks	Monitoring) Public DNS: ed	c2-52-91-115-180.com	npute-1.amazonaw	s.com					8 8
Description	Status Checks	Monitoring) Public DNS: ed	c2-52-91-115-180.com	npute-1.amazonaw	s.com				8	
Description	Status Checks	Monitoring i-04812ba54e249c) Public DNS: ed Tags 99c	c2-52-91-115-180.com	pute-1.amazonaw Public DNS (II	s.com Pv4)	ec2-52-91-11 1.amazonaws	5-180.compute	9-		
Description	Status Checks Instance ID	Monitoring i-04812ba54e249c running) Public DNS: ed Tags 99c	c2-52-91-115-180.com	Public DNS (II	s.com Pv4) ic IP	ec2-52-91-11 1.amazonaws 52.91.115.18	5-180.compute s.com 0	9-		
Description	Status Checks Instance ID Instance state Instance type	Monitoring i-04812ba54e249c running t2.micro) Public DNS: ed Tags 99c	c2-52-91-115-180.com	Public DNS (II	Pv4) ic IP i IPs	ec2-52-91-11 1.amazonaws 52.91.115.18	5-180.compute s.com 0	9-		
Description	Status Checks Instance ID Instance state Instance type Elastic IPs	Monitoring i-04812ba54e249c running t2.micro	99c	c2-52-91-115-180.com	Public DNS (I Public DNS (I Pv4 Public IPv6 Private I	Pv4) ic IP ic IPs DNS	ec2-52-91-11: 1.amazonaws 52.91.115.18 - ip-172-31-16-	5-180.compute s.com 0 11.ec2.interna	ə- 1	-	
Description	Status Checks Instance ID Instance state Instance type Elastic IPs Availability zone	Monitoring i-04812ba54e249c running t2.micro us-east-1b) Public DNS: ed Tags 99c	c2-52-91-115-180.com	Public DNS (II Public DNS (II Pv4 Public IPv6 Private Private	Pv4) ic IP ic IPs DNS e IPs	ec2-52-91-11: 1.amazonaws 52.91.115.18i - ip-172-31-16- 172.31.16.11	5-180.compute s.com 0 -11.ec2.interna	ə- 1	-	

Figure 6-3. EC2 instances for Docker for AWS Swarm nodes

SSH login into the manager instance.

```
[root@localhost ~]# ssh -i "docker.pem" docker@52.91.115.180
Welcome to Docker!
```

List the nodes in the Swarm. A manager node and two worker nodes are listed.

```
~ $ docker node ls
ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS
8ynq7exfo5v74ymoe7hrsghxh ip-172-31-33-230.ec2.internal Ready Active
o0h7o09a61ico7n1t8ooe281g * ip-172-31-16-11.ec2.internal Ready Active Leader
yzlv7c3qwcwozhxz439dbknj4 ip-172-31-25-163.ec2.internal Ready Active
```

Creating a Named Volume

A named volume to be used in a service as a mount of type volume may either be created prior to creating the service or at deployment time. A new named volume is created with the following command syntax.

```
docker volume create [OPTIONS] [VOLUME]
```

The options discussed in Table 6-1 are supported.

Option	Description	Туре	Default Value
driver, -d	Specifies volume driver name	string	local
label	Sets metadata for a volume	value	[]
name	Specifies volume name	string	
opt, -o	Sets driver specific options	value	map[]

 Table 6-1. Options for the docker volume create Command for a Named Volume

Create a named volume called hello using the docker volume create command.

 \sim \$ docker volume create --name hello hello

Subsequently, list the volumes with the docker volume 1s command. The hello volume is listed in addition to other named volumes that may exist.

~	\$	docker	volume	ls	
DF	RI۱	/ER		VOLUME	NAME
10	oca	al		hello	

You can find detailed info about the volume using the following command.

docker volume inspect hello

In addition to the volume name and driver, the mountpoint of the volume also is listed.

The scope of a local driver volume is local. The other supported scope is global. A local volume is created on a single Docker host and a global volume is created on each Docker host in the cluster.

Using a Volume Mount

Use the hello volume in the docker service create command with the --mount option. The options discussed in Table 6-2 may be used both with bind mounts and volume mounts.

Option	Required	Description	Default
type	No	Specifies the type of mount. One of three values may be specified:	volume
		volume-Mounts is a named volume in a container. bind-Bind-mounts is a directory or file from the host into a container. tmpfs-Mounts is a tmpfs into a container.	
src or source	Yes for type=bind only. No for type=volume	The source directory or volume. The option has different meanings for different types of mounts.	
		type=volume: src specifies the name of the volume. If the named volume does not exist, it is created. If src is omitted, the named volume is created with an auto-generated name, which is unique on the host but may not be unique cluster-wide. An auto-generated named volume is removed when the container using the volume is removed. The docker service update command shuts down task containers and starts new task containers and so does scaling a service. volume source must not be an absolute path.	
		<pre>type=bind: src specifies the absolute path to the directory or file to bind-mount. The directory path must be an absolute and not a relative path. The src option is required for a mount of type bind and an error is generated if it's not specified. type=tmpfs: is not supported.</pre>	
dst or destination or target	Yes	Specifies the mount path inside a container. If the path does not exist in a container's filesystem, the Docker engine creates the mount path before mounting the bind or volume mount. The volume target must be a relative path.	
readonly or ro	No	A boolean (true/false) or (1/0) to indicate whether the Docker Engine should mount volumes and bind read-write or read-only. If the option is not specified, the engine mounts the bind or volume read-write. If the option is specified with a value of true or 1 or no value, the engine mounts the volume or bind read-only. If the option is specified with a value of false or 0, the engine mounts the volume or bind read-write.	

Table 6-2. Options for Volume and Bind Mounts

Some of the mount options are only supported for volume mounts and are discussed in Table 6-3.

Option	Required	Description	Default Value
volume-driver	No	Specifies the name of the volume-driver plugin to use for the volume. If a named volume is not specified in src, the volume-driver is used to create a named volume.	local
volume-label	No	Specifies one or more comma-separated metadata labels to apply to the volume. Example: volume- label=label-1=hello-world,label-2=hello.	
volume-nocopy	No	Applies to an empty volume that is mounted in a container at a mount path at which files and directories already existed. Specifies whether a container's filesystem files and directories at the mount path (dst) are to be copied to the volume. A host is able to access the files and directories copied from the container to the named volume. A value of true or 1 disables copying of files from the container's filesystem to the host volume. A value of false or 0 enables copying.	true or 1
volume-opt	No	Specifies the options to be supplied to the volume- driver in creating a named volume if one does not exist. The volume-opt options are specified as a comma-separated list of key/value pairs. Example: volume-opt-1=option-1=value1,option-2=value2.	
		A named volume has to exist on each host on which a mount of type volume is to be mounted. Creating a named volume on the Swarm manager does not also create the named volume on the worker nodes. The volume-driver and volume-opt options are used to create the named volume on the worker nodes.	

Table 6-3. Options for Volume Mounts

The options discussed in Table 6-4 are supported only with a mount of type tmpfs.

 Table 6-4.
 Options for the tmpfs Mount

Option	Required	Description	Default Value
tmpfs-size	No	Size of the tmpfs mount in bytes	Unlimited value on Linux
tmpfs-mode	No	Specifies the file mode of the tmpfs in octal	1777 in Linux

Next, we will use the named volume hello in a service created with Docker image tutum/hello-world. In the following docker service create command, the --mount option specifies the src as hello and includes some volume-label labels for the volume.

```
~ $ docker service create \
    --name hello-world \
    --mount src=hello,dst=/hello,volume-label="msg=hello",volume-label="msg2=world" \
    --publish 8080:80 \
    --replicas 2 \
    tutum/hello-world
```

The service is created and the service ID is output.

```
~ $ docker service create \
> --name hello-world \
> --mount src=hello,dst=/hello,volume-label="msg=hello",volume-label="msg2=world" \
> --publish 8080:80 \
> --replicas 2 \
> tutum/hello-world
8ily37072wyxkyw2jt60kdqoz
```

Two service replicas are created.

~ \$ docker service ls ID NAME MODE REPLICAS IMAGE PORTS 8ily37o72wyx hello-world replicated 2/2 tutum/hello-world:latest *:8080->80/tcp ~ \$ docker service ps hello-world NODE NAME IMAGE TD CURRENT STATE DESIRED STATE ERROR PORTS uw6coztxwqhf hello-world.1 tutum/hello-world:latest ip-172-31-25-163.ec2.internal Running 20 seconds ago Running cfkwefwadkki hello-world.2 tutum/hello-world:latest ip-172-31-16-11.ec2.internal Running 21 seconds ago Running

The named volume is mounted in each task container in the service. The service definition lists the mounts, including the mount labels.

```
~ $ docker service inspect hello-world
ſ
        "Spec": {
                 "ContainerSpec": {
                     "Image": "tutum/hello-world:latest@sha256:0d57def8055178aafb4c7669cbc25e
                     c17f0acdab97cc587f30150802da8f8d85",
                     "Mounts": [
                         {
                             "Type": "volume",
                             "Source": "hello",
                             "Target": "/hello",
                             "VolumeOptions": {
                                  "Labels": {
                                      "msg": "hello",
                                      "msg2": "world"
                                 },
. . .
1
```

In the preceding example, a named volume is created before using the volume in a volume mount. As another example, create a named volume at deployment time. In the following docker service create command, the --mount option is set to type=volume with the source set to nginx-root. The named volume nginx-root does not exist prior to creating the service.

```
~ $ docker service create \
> --name nginx-service \
> --replicas 3 \
> --mount type=volume,source="nginx-root",destination="/var/lib/nginx",volume-
label="type=nginx root dir" \
> nginx:alpine
rtz1ldok405mr03uhdk1htlnk
```

When the command is run, a service is created. Service description includes the volume mount in mounts.

```
~ $ docker service inspect nginx-service
[
. . .
         "Spec": {
             "Name": "nginx-service",
. . .
                      "Mounts": [
                          {
                              "Type": "volume",
                              "Source": "nginx-root",
                              "Target": "/var/lib/nginx",
                              "VolumeOptions": {
                                  "Labels": {
                                       "type": "nginx root dir"
                                  },
. . .
1
```

The named volume nginx-root was not created prior to creating the service and is therefore created before starting containers for service tasks. The named volume nginx-root is created only on nodes on which a task is scheduled. One service task is scheduled on each of the three nodes.

~ \$ docker service	ps nginx-service		
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
pfqinizqmgur	nginx-service.1	nginx:alpine	ip-172-31-33-230.ec2.internal
Running	Running 19 seconds	ago	
mn8h3p40chgs	nginx-service.2	nginx:alpine	ip-172-31-25-163.ec2.internal
Running	Running 19 seconds	ago	
k8n5zzlnn46s	nginx-service.3	nginx:alpine	ip-172-31-16-11.ec2.internal
Running	Running 18 seconds	ago	

CHAPTER 6 USING MOUNTS

As a task is scheduled on the manager node, a named volume called nginx-root is created on the manager node, as listed in the output of the docker volume ls command.

~	\$	docker	volume	ls	
DF	۲I	/ER		VOLUME	NAME
10	оса	al		hello	
10	Ca	al		nginx-1	root

Service tasks and task containers are started on each of the two worker nodes. A nginx-root named volume is created on each of the worker nodes. Listing the volumes on the worker nodes lists the nginx-root volume.

```
[root@localhost ~]# ssh -i "docker.pem" docker@34.229.86.64
Welcome to Docker!
~ $ docker volume ls
DRIVER VOLUME NAME
local hello
local nginx-root
[root@localhost ~]# ssh -i "docker.pem" docker@52.91.200.241
```

```
Welcome to Docker!

~ $ docker volume ls

DRIVER VOLUME NAME

local hello

local nginx-root
```

A named volume was specified in src in the preceding example. The named volume may be omitted as in the following service definition.

```
~ $ docker service create \
> --name nginx-service-2 \
> --replicas 3 \
> --mount type=volume,destination=/var/lib/nginx \
> nginx:alpine
q8ordkmkwqrwiwhmaemvcypc3
```

The service is created with a replica and is scheduled on each of the Swarm nodes.

~ \$ docker service	ps nginx-service-2		
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
kz8d8k6bxp7u	nginx-service-2.1	nginx:alpine	ip-172-31-25-163.ec2.internal
Running	Running 27 seconds	ago	
wd65qsmqixpg	nginx-service-2.2	nginx:alpine	ip-172-31-16-11.ec2.internal
Running	Running 27 seconds	ago	
mbnmzldtaaed	nginx-service-2.3	nginx:alpine	ip-172-31-33-230.ec2.internal
Running	Running 26 seconds	ago	

The service definition does not list a named volume.

Named volumes with auto-generated names are created when a volume name is not specified explicitly. One auto-generated named volume with an auto-generated name is created on each node on which a service task is run. One of the named volumes listed on the manager node is an auto-generated named volume with an auto-generated name.

~ \$ docker vol	ume ls
DRIVER	VOLUME NAME
local	305f1fa3673e811b3b320fad0e2dd5786567bcec49b3e66480eab2309101e23
local	hello
local	nginx-root

As another example of using named volumes as mounts in a service, create a named volume called mysql-scripts for a MySQL database service.

```
~ $ docker volume create --name mysql-scripts mysql-scripts
```

The named volume is created and listed.

~ \$ docker volume	ls
DRIVER	VOLUME NAME
local	305f1fa3673e811b3b320fad0e2dd5786567bcec49b3e66480eab2309101e233
local	hello
local	mysql-scripts
local	nginx-root

The volume description lists the scope as local and lists the mountpoint.

Next, create a service that uses the named volume in a volume mount.

```
~ $ docker service create \
> --env MYSQL_ROOT_PASSWORD='mysql'\
> --mount type=volume,src="mysql-scripts",dst="/etc/mysql/scripts",
        el="msg=mysql",volume-label="msg2=scripts" \
> --publish 3306:3306\
> --replicas 2 \
> --name mysql \
> mysql
cghaz4zoxurpyqil5iknqf4c1
```

The service is created and listed.

~ \$ docker service ls									
ID	NAME	MODE	REPLICAS	IMAGE	PORTS				
8ily37o72wyx	hello-world	replicated	2/2	<pre>tutum/hello-world:latest</pre>	*:8080->80/tcp				
cghaz4zoxurp	ysql	replicated	1/2	mysql:latest	*:3306->3306/tcp				

Listing the service tasks indicates that the tasks are scheduled on the manager node and one of the worker nodes.

~ \$ docker service	ps mysql			
ID	NAME	IMAGE		NODE
DESIRED STATE	CURRENT STATE		ERROR	PORTS
y59yhzwch2fj	mysql.1	mysql:latest		ip-172-31-33-230.ec2.internal
Running	Preparing 12 second	ls ago		
zg7wrludkr84	mysql.2	<pre>mysql:latest</pre>		ip-172-31-16-11.ec2.internal
Running	Running less than a	second ago		

The destination directory for the named volume is created in the Docker container. The Docker container on the manager node may be listed with docker ps and a bash shell on the container may be started with the docker exec -it <containerid> bash command.

IMAGE	
STATUS	PORTS
mysql:latest	
Up 21 seconds	3306/tcp
	IMAGE STATUS mysql:latest Up 21 seconds

COMMAND NAMES "docker-entrypoint..." mysql.2.zg7wrludkr84zf 8vhdkf8wnlh

```
~ $ docker exec -it a855826cdc75 bash
root@a855826cd75:/#
```

Change the directory to /etc/mysql/scripts in the container. Initially, the directory is empty.

```
root@a855826cdc75:/# cd /etc/mysql/scripts
root@a855826cdc75:/etc/mysql/scripts# ls -l
total 0
root@a855826cdc75:/etc/mysql/scripts# exit
exit
```

A task container for the service is created on one of the worker nodes and may be listed on the worker node.

~ \$ docker ps			
CONTAINER ID	IMAGE		COMMAND
CREATED	STATUS	PORTS	NAMES
eb8d59cc2dff	<pre>mysql:latest</pre>		<pre>"docker-entrypoint"</pre>
8 minutes ago	Up 8 minutes	3306/tcp	mysql.1.xjmx7qviihyq2so7n0oxi1muq

Start a bash shell for the Docker container on the worker node. The /etc/mysql/scripts directory on which the named volume is mounted is created in the Docker container.

```
~ $ docker exec -it eb8d59cc2dff bash
root@eb8d59cc2dff:/# cd /etc/mysql/scripts
root@eb8d59cc2dff:/etc/mysql/scripts# exit
exit
```

If a service using an auto-generated named volume is scaled to run a task on nodes on which a task was not running previously, named volumes are auto-generated on those nodes also. As an example of finding the effect of scaling a service when using an auto-generated named volume as a mount in the service, create a MySQL database service with a volume mount. The volume mysql-scripts does not exist prior to creating the service; remove the mysql-scripts volume if it exists.

```
~ $ docker service create \
> --env MYSQL_ROOT_PASSWORD='mysql'\
> --replicas 1 \
> --mount type=volume,src="mysql-scripts",dst="/etc/mysql/scripts"\
> --name mysql \
> mysql
088ddf5pt4yb3yvr5s7elyhpn
```

CHAPTER 6 USING MOUNTS

The service task is scheduled on a node.

~ \$ docker service p	os mysql		
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
xlix91njbaq0	mysql.1	mysql:latest	ip-172-31-13-122.ec2.internal
Running	Preparing 12 seconds	s ago	

List the nodes; the node on which the service task is scheduled is the manager node.

~ \$ docker node ls				
ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS
o5hyue3hzuds8vtyughswbosl	ip-172-31-11-41.ec2.internal	Ready	Active	
p6uuzp8pmoahlcwexr3wdulxv	ip-172-31-23-247.ec2.internal	Ready	Active	
<pre>qnk35m0141lx8jljp87ggnsnq *</pre>	ip-172-31-13-122.ec2.internal	Ready	Active	Leader

A named volume mysql-scripts and an ancillary named volume with an auto-generated name are created on the manager node on which a task is scheduled.

~ \$ docker	volume ls
DRIVER	VOLUME NAME
local	a2bc631f1b1da354d30aaea37935c65f9d99c5f084d92341c6506f1e2aab1d55
local	mysql-scripts

The worker nodes do not list the mysql-scripts named volume, as a task is not scheduled on the worker nodes.

~ \$ docker volume ls DRIVER VOLUME NAME

Scale the service to three replicas. A replica is scheduled on each of the three nodes.

```
~ $ docker service scale mysql=3
mysql scaled to 3
```

~ \$ docker servic	e ps mysql			
ID	NAME	IMAGE		NODE
DESIRED STATE	CURRENT STAT	E	ERROR	PORTS
xlix91njbaq0	mysql.1	<pre>mysql:latest</pre>		ip-172-31-13-122.ec2.internal
Running	Running abou	t a minute ago		-
ifk7xuvfp9p2	mysql.2	mysql:latest		ip-172-31-23-247.ec2.internal
Running	Running less	than a second ago		-
3c53fxgcjqyt	mysql.3	<pre>mysql:latest</pre>		ip-172-31-11-41.ec2.internal
Running	Running less	than a second ago		-

A named volume mysql-scripts and an ancillary named volume with an auto-generated name are created on the worker nodes because a replica is scheduled.

```
[root@localhost ~]# ssh -i "docker.pem" docker@54.165.69.9
Welcome to Docker!
```

~ \$ docker volume ls	5
DRIVER	VOLUME NAME
local	431a792646d0b04b5ace49a32e6c0631ec5e92f3dda57008b1987e4fe2a1b561
local	mysql-scripts
<pre>[root@localhost ~]#</pre>	ssh -i "docker.pem" docker@34.232.95.243
Welcome to Docker!	
~ \$ docker volume ls	;
DRIVER	VOLUME NAME
local	afb2401a9a916a365304b8aa0cc96b1be0c161462d375745c9829f2b6f180873
local	mysql-scripts

The auto-generated named volumes are persistent and do not get removed when a service replica is shut down. The named volumes with auto-generated names are not persistent volumes. As an example, scale the service back to one replica. Two of the replicas shut down, including the replica on the manager node.

~ \$ docker servic	ce scale mysql=1		
mysql scaled to 1	L		
~ \$ docker servic	ce ps mysql		
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
3c53fxgcjqyt	mysql.3	mysql:latest	ip-172-31-11-41.ec2.internal
Running	Running 2 minute	s ago	

But the named volume mysql-scripts on the manager node is not removed even though no Docker container using the volume is running.

~	\$	docker	volume	ls	
DF	RΙ	/ER		VOLUME	NAME
10	оса	1		mysql-s	scripts

Similarly, the named volume on a worker node on which a service replica is shut down also does not get removed even though no Docker container using the named volume is running. The named volume with the auto-generated name is removed when no container is using it, but the mysql-scripts named volume does not.

Remove the volume mysql-scripts still does not get removed.

```
~ $ docker service rm mysql
mysql
~ $ docker volume ls
DRIVER VOLUME NAME
local mysql-scripts
```

Removing a Volume

A named volume may be removed using the following command.

```
docker volume rm <VOL>
```

As an example, remove the named volume mysql-scripts.

```
~ $ docker volume rm mysql-scripts
mysql-scripts
```

If the volume you try to delete is used in a Docker container, an error is generated instead and the volume will not be removed. Even a named volume with an auto-generated name cannot be removed if it's being used in a container.

Creating and Using a Bind Mount

In this section, we create a mount of type *bind*. Bind mounts are suitable if data in directories that already exist on the host needs to be accessed from within Docker containers. type=bind must be specified with the --mount option when creating a service with mount of type *bind*. The host source directory and the volume target must both be absolute paths. The host source directory must exist prior to creating a service. The target directory within each Docker container of the service is created automatically. Create a directory on the manager node and then add a file called createtable.sql to the directory.

```
core@ip-10-0-0-143 ~ $ sudo mkdir -p /etc/mysql/scripts
core@ip-10-0-0-143 ~ $ cd /etc/mysql/scripts
core@ip-10-0-0-143 /etc/mysql/scripts $ sudo vi createtable.sql
```

Save a SQL script in the sample SQL file, as shown in Figure 6-4.

Σ				ro	oot@loca	lhost:	~			-		x
File	Edit ۱	/iew	Search	Terminal	Help							
CREATE INSERT INSERT INSERT INSERT INSERT	TABL INTO INTO INTO INTO INTO	E wls wlsl wlsl wlsl wlsl wlsl	log(tim og(time og(time og(time og(time og(time	e_stamp \ _stamp,ca _stamp,ca _stamp,ca _stamp,ca _stamp,ca	/ARCHAR(2 ategory,1 ategory,1 ategory,1 ategory,1 ategory,1	255) P type,s type,s type,s type,s	RIMARY erverna erverna erverna erverna	KEY, catego ime, code, msg ime, code, msg ime, code, msg ime, code, msg ime, code, msg ime, code, msg	y VARCHA) VALUES) VALUES) VALUES) VALUES) VALUES) VALUES	R(255 ('Apr ('Apr ('Apr ('Apr ('Apr),t -8- -8- -8- -8- -8-	2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
INSERT	INTO	wlsl	log(time	_stamp,ca	ategory,1	type,s	erverna	ime, code, ms) VALUES	('Apr	-8-	2

Figure 6-4. Adding a SQL script to the host directory

Similarly, create a directory and add a SQL script to the worker nodes.

Create a service with a bind mount that's using the host directory. The destination directory is specified as /scripts.

```
core@ip-10-0-0-143 ~ $ docker service create \
>    --env MYSQL_ROOT_PASSWORD='mysql' \
>    --replicas 3 \
>    --mount type=bind,src="/etc/mysql/scripts",dst="/scripts" \
>        --name mysql \
>         mysql
Okvk2hk2qigqyeem8x1r8qkvk
```

Start a bash shell for the service container from the node on which a task is scheduled. The destination directory /scripts is listed.

core@ip-10-0-0-143 ^	°\$ docker ps		
CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
e71275e6c65c Up 4 seconds	mysql:latest 3306/tcp	"docker-entrypoint.sh" mysql.1.btqfrx7uffym2xvc4	5 seconds ago 141pubaza
core@ip-10-0-0-143 ^ root@e71275e6c65c:/#	∽\$ docker exec -it e ‡ ls -l	271275e6c65c bash	

drwxr-xr-x. 2 root root 4096 Jul 24 20:44 scripts

Change the directory (cd) to the destination mount path /scripts. The createtable.sql script is listed in the destination mount path of the bind mount.

```
root@e71275e6c65c:/# cd /scripts
root@e71275e6c65c:/scripts# ls -l
-rw-r--r-. 1 root root 1478 Jul 24 20:44 createtable.sql
```

Each service task Docker container has its own copy of the file on the host. Because, by default, the mount is read-write, the files in the mount path may be modified or removed. As an example, remove the createtable.sql script from a container.

```
core@ip-10-0-0-137 ~ $ docker exec -it 995b9455aff2 bash
root@995b9455aff2:/# cd /scripts
root@995b9455aff2:/scripts# ls -l
total 8
-rw-r--r--. 1 root root 1478 Jul 24 20:45 createtable.sql
root@995b9455aff2:/scripts# rm createtable.sql
root@995b9455aff2:/scripts# ls -l
total 0
root@995b9455aff2:/scripts#
```

A mount may be made read-only by including an additional option in the --mount arg, as discussed earlier. To demonstrate a readonly mount, first remove the mysql service that's already running. Create a service and mount a readonly bind with the same command as before, except include an additional readonly option.

```
core@ip-10-0-0-143 ~ $ docker service create \
>     --env MYSQL_ROOT_PASSWORD='mysql' \
>     --replicas 3 \
>     --mount type=bind,src="/etc/mysql/scripts",dst="/scripts",readonly \
>     --name mysql \
>         mysql
c27se8vfygk2z57rtswentrix
```

A bind of type mount which is readonly is mounted.

Access the container on a node on which a task is scheduled and list the sample script from the host directory.

```
core@ip-10-0-0-143 ~ $ docker exec -it 3bf9cf777d25 bash
root@3bf9cf777d25:/# cd /scripts
root@3bf9cf777d25:/scripts# ls -l
-rw-r--r-. 1 root root 1478 Jul 24 20:44 createtable.sql
```

Remove, or try to remove, the sample script. An error is generated.

```
root@3bf9cf777d25:/scripts# rm createtable.sql
rm: cannot remove 'createtable.sql': Read-only file system
```

Summary

This chapter introduced mounts in Swarm mode. Two types of mounts are supported—*bind* mount and *volume* mount. A bind mount mounts a pre-existing directory or file from the host into each container of a service. A volume mount mounts a named volume, which may or may not exist prior to creating a service, into each container in a service. The next chapter discusses configuring resources.

CHAPTER 7

Configuring Resources

Docker containers run in isolation on the underlying OS kernel and require resources to run. Docker Swarm mode supports two types of resources—CPU and memory—as illustrated in Figure 7-1.



Figure 7-1. Types of resources supported by Docker Swarm mode

The Problem

By default, Docker Swarm mode does not impose any limit on how many resources (CPU cycles or memory) a service task may consume. Nor does Swarm mode guarantee minimum resources. Two issues can result if no resource configuration is specified in Docker Swarm mode.

Some of the service tasks could consume a disproportionate amount of resources, while the other service tasks are not able to get scheduled due to lack of resources. As an example, consider a node with resource capacity of 3GB and 3 CPUs. Without any resource guarantees and limits, one service task container could consume most of the resources (2.8GB and 2.8 CPUs), while two other service task containers each have only 0.1GB and 0.1 CPU of resources remaining to be used and do not get scheduled, as illustrated in Figure 7-2. A Docker service task that does not have enough resources to get scheduled is put in Pending state.

CHAPTER 7 CONFIGURING RESOURCES



Figure 7-2. Unequal allocation of resources

The second issue that can result is that the resource capacity of a node can get fully used up without any provision to schedule any more service tasks. As an example, a node with a resource capacity of 9GB and 9 CPUs has three service task containers running, with each using 3GB and 3 CPUs, as illustrated in Figure 7-3. If a new service task is created for the same or another service, it does not have any available resources on the node.



Figure 7-3. Fully resource-utilized node

The Solution

Docker Swarm mode has a provision to set resource guarantees (or reserves) and resource limits, as illustrated in Figure 7-4. A *resource reserve* is the minimum amount of a resource that is guaranteed or reserved for a service task. A *resource limit* is the maximum amount of a resource that a service task can use regardless of how much of a resource is available.



Figure 7-4. Managing Swarm resources with resource reserves and limits

With resource reserves, each service task container can be guaranteed 1 CPU and 1GB in the issue discussed previously, as illustrated in Figure 7-5.



Figure 7-5. Resource allocation with resource reserves set

And, if resource limits are implemented for service task containers, excess resources would be available to start new service task containers. In the example discussed previously, a limit of 2GB and 2 CPUs per service task would keep the excess resources of 3GB and 3 CPUs available for new service task containers, as illustrated in Figure 7-6.

CHAPTER 7 CONFIGURING RESOURCES



Figure 7-6. Resource allocation with resource limits set

This chapter covers the following topics:

- Setting the environment
- Creating a service without resource specification
- Reserving resources
- Setting resource limits
- Creating a service with resource specification
- Scaling and resources
- Reserved resources must be less than resource limits
- Rolling update to set resource limits and reserves
- Resource usage and node capacity

Setting the Environment

Create a three-node Swarm on Docker for AWS with one manager node and two worker nodes. Creating a Swarm on Docker for AWS is discussed in Chapter 3. We use the three-node Swarm created in Chapter 6 for this chapter also. Obtain the public IP address of the Swarm manager instance, as shown in Figure 7-7.

CHAPTER 7 CONFIGURING RESOURCES

Q. Filter by tags	s and attributes or	search by keyword					0	< <	< 1 to 3 of 3	> >
Name	*	nstance ID	• Instance Type •	Availability Zone •	Instance State 👻	Status Checks 👻	Alarm Statu	IS	Public DNS (II	Pv4)
Docker-wo	rker	083e010a9e00271	t2.micro	us-east-1b	running	2/2 checks	None	>	ec2-52-91-200	-241.c
Docker-Ma	anager	04812ba54e249c99	t2.micro	us-east-1b	running	2/2 checks	None	20	ec2-52-91-115	-180.c
Docker-wo	rker	0a0d7ba92b6a454	t2.micro	us-east-1c	running	2/2 checks	None	10	ec2-34-229-86	-64.co
nstance: i i-04 Description	812ba54e249c9 Status Check	e (Docker-Manage	er) Public DNS: ed	c2-52-91-115-180.com	npute-1.amazonaw	s.com				3 6
nstance: i-04	812ba54e249c9 Status Check	e (Docker-Managi Monitoring	er) Public DNS: ea	c2-52-91-115-180.com	npute-1.amazonaw	s.com				
nstance: i i-04 Description	812ba54e249c9 Status Check Instance I	C (Docker-Manage Monitoring	er) Public DNS: er Tags 90:990:	2-52-91-115-180.com	npute-1.amazonaw Public DNS (I	s.com Pv4) ec2-52-91-11	5-180.compute	÷-	8	
nstance: i i-04	812ba54e249c9 Status Check Instance I	Bc (Docker-Manage Monitoring) i-04812ba54e24	Tags	c2-52-91-115-180.com	npute-1.amazonaw Public DNS (I	s.com Pv4) ec2-52-91-11 1.amazonawa	5-180.compute	þ.		
nstance: i-04	812ba54e249c9 Status Check Instance I	C (Docker-Manage Monitoring i-04812ba54e24	er) Public DNS: er Tags 90990	c2-52-91-115-180.com	Public DNS (II	s.com Pv4) ec2-52-91-11 1.amazonawa 52.91.115.18	5-180.compute s.com	ð-	8	
nstance: I-04	812ba54e249c9 Status Check Instance I Instance stat	C (Docker-Manage Monitoring i-04812ba54e24 running t2.micro	er) Public DNS: en Tags 90990	22-52-91-115-180.com	Public DNS (II	s.com Pv4) ec2-52-91-11 1.amazonaw: ic IP 52.91.115.18 i IPs - DVB in 172.31.15	5-180.compute a.com D	ð-		3 8
nstance: 1-04 Description	812ba54e249c9 Status Check Instance I Instance stat Instance typ Elastic IP	C (Docker-Manage Monitoring i-04812ba54e24 running t2.micro	er) Public DNS: en Tags 39c99c	:2-52-91-115-180.com	Public DNS (I Public DNS (I Pv4 Publi IPv4 Private I Private I	s.com Pv4) ec2-52-91-11 1.amazonaw: ic IP 52.91.115.18 i IPs - DNS ip-172-31-16 IP- 172-31.16 11	5-180.compute s.com 0 11.ec2.internal	9- 1	= :	

Figure 7-7. EC2 instances for Swarm nodes

SSH login into the manager instance with user as "docker".

```
[root@localhost ~]# ssh -i "docker.pem" docker@52.91.115.180
Welcome to Docker!
```

List the Swarm nodes; a manager node and two worker nodes are listed.

~ \$ docker node ls					
ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER	STATUS
8ynq7exfo5v74ymoe7hrsghxh	ip-172-31-33-230.ec2.internal	Ready	Active		
o0h7o09a61ico7n1t8ooe281g *	ip-172-31-16-11.ec2.internal	Ready	Active	Leader	
yzlv7c3qwcwozhxz439dbknj4	ip-172-31-25-163.ec2.internal	Ready	Active		

Creating a Service Without Resource Specification

We start by creating a service without any resource specification. Create a MySQL database service without setting any resource reserves or limits.

```
docker service create \
    --env MYSQL_ROOT_PASSWORD='mysql'\
    --replicas 1 \
    --name mysql \
    mysql
```

A single service replica is created. The output of the command is the service ID (shown in italics).

```
~ $ docker service create \
> --env MYSQL_ROOT_PASSWORD='mysql'\
> --replicas 1 \
> --name mysql \
> mysql
2kcq6cf72t4wu94o00k3sax41
```

List the services; the mysql service is listed.

~ \$ docker service .	ls				
ID	NAME	MODE	REPLICAS	IMAGE	PORTS
2kcq6cf72t4w	mysql	replicated	1/1	mysql:latest	

List the service tasks. The only service task is running on a worker node.

~ \$ docker service	e ps mysql		
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
sccqv4k9r22h	mysql.1	mysql:latest	ip-172-31-33-230.ec2.internal
Running	Running 10 seconds a	go	

On inspecting the service, the container spec does not include any resources, limits, or reserves. The single service task may use all of the available resources on the node on which it's scheduled.

```
~ $ docker service inspect mysql
[
                "Resources": {
                "Limits": {},
                "Reservations": {}
                },
]
```

Reserving Resources

Swarm mode provides two options for resource reserves in the docker service create and docker service update commands, as listed in Table 7-1.

Table 7-1. Options for Resource Reserves

Option	Description	Default Value
reserve-cpu	Reserve CPUs. A value of 0.000 implies no reserves are set.	0.000
reserve-memory	Reserve memory. A value of 0 implies no reserves are set.	0

Setting Resource Limits

Swarm mode provides two options for resource limits in the docker service create and docker service update commands, as discussed in Table 7-2.

 Table 7-2.
 Options for Resource Limits

Option	Description	Default Value
limit-cpu	Limit CPUs	0.000
limit-memory	Limit Memory	0

Creating a Service with Resource Specification

Next, create a service using resource specification. Set resource reserves of 0.25 CPUs and 128MB and resource limits of 1 CPU and 256MB. Remove the mysql service previously created before creating a new service with resources defined. The output of the command is the service ID (shown in italics).

```
~ $ docker service rm mysql
mysql
~ $ docker service create \
> --env MYSQL_ROOT_PASSWORD='mysql'\
> --replicas 1 \
> --name mysql \
> --reserve-cpu .25 --limit-cpu 1 --reserve-memory 128mb --limit-memory 256mb \
> mysql
abwq9budo7joyd00u32z2b047
```

On inspecting the service, the resources limits and reserves are listed, which contrasts with the empty settings for resources when a service is created without the resources definition.

Scaling and Resources

Before scaling up a service, it may be suitable to determine the node capacity in terms of CPU and memory resources. As all three nodes in the Swarm are identical, the node capacity on one node is the same as on the other nodes. The node capacity is 1 CPU and 1GB, as listed in the output of the docker node inspect command.

The CPU limit on each service task created in the preceding section is also 1 CPU. When scaling, the total of the resource limits for all service tasks on a node may exceed the node's capacity. However, the total of resource reserves must not exceed node capacity.

As an example, scale to five replicas.

```
~ $ docker service scale mysql=5
mysql scaled to 5
```

. . . .

Scaling to five schedules two replicas on the manager node, two replicas on one of the worker nodes, and one replica on the other worker node. The aggregate of the resource limits on the worker nodes is exceeded but the aggregate of resource reserves are within the node's capacity.

~ \$ docker servi	ce ps mysq⊥			
ID	NAME	IMAGE	NODE	
DESIRED STATE	CURRENT STATE	ERROR	PORTS	
npc5r7xf98fg	mysql.1	<pre>mysql:latest</pre>	ip-172-31-16-11.ec2.interna	L
Running	Running 2 minu	ites ago		
xokdhowntp0w	mysql.2	<pre>mysql:latest</pre>	ip-172-31-25-163.ec2.interna	31
Running	Running 13 sec	onds ago		
b6h4bsf7xzdc	mysql.3	<pre>mysql:latest</pre>	ip-172-31-16-11.ec2.interna	Ĺ
Running	Running 12 sec	onds ago		
j1d7ti7nb80u	mysql.4	<pre>mysql:latest</pre>	ip-172-31-33-230.ec2.interna	31
Running	Running 13 sec	onds ago		
w6to9pxcdbm5	mysql.5	mysql:latest	ip-172-31-25-163.ec2.interna	31
Running	Running 13 sec	onds ago		

Reserved Resources Must Not Be More Than Resource Limits

The resource limits are not taken into consideration when scheduling a service task, only the resource reserves are. Not setting the reserves (whether limits are set or not and whether limits exceed node capacity) schedules the service task if the resources required to run a task are within the node capacity. Resource reserves must not exceed resource limits or a service task may not get scheduled or might fail after a while. As an example, delete the mysql service and create a new service where the resource reserves exceed resource limits. The output of the command is the service ID (shown in italics).

```
~ $ docker service rm mysql
mysql
~ $ docker service create \
> --env MYSQL_ROOT_PASSWORD='mysql'\
> --replicas 1 \
> --name mysql \
> --reserve-cpu .75 --limit-cpu .5 --reserve-memory 256mb --limit-memory 128mb \
> mysql
srot5vr8x7v7iml2awc3fxb1u
```

The service is created and even scheduled.

~ \$ docker service	ps mysql		
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
pmcjrj6p3wfp	mysql.1	mysql:latest	ip-172-31-16-11.ec2.internal
Running	Running 20 seconds	ago	-

The service configuration has the resource reserves exceeding the resource limits.

The resource reserves are within the node capacity, but because the resource limits are less than the resource reserves, the newly started service task fails and is shut down. The service task keeps getting restarted and shut down.

~ \$ docker service	ps mysql		
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
vjcnjkwfdfkb	mysql.1	mysql:latest	ip-172-31-16-11.ec2.internal
Running	Running 16 second	ds ago	
pxdku8pxviyn	_ mysql.1	mysql:latest	ip-172-31-16-11.ec2.internal
Shutdown	Failed 21 seconds	s ago "task: n	on-zero exit (1)"
pmcjrj6p3wfp	_ mysql.1	mysql:latest	ip-172-31-16-11.ec2.internal
Shutdown	Failed about a mi	inute ago "task: n	on-zero exit (1)"

The service task resource limits can be the same as the resource reserves. Remove the mysql service and create it again with the resource limits the same as the resource reserves. The output of the command is the service ID (shown in italics).

```
~ $ docker service rm mysql
mysql
~ $ docker service create \
> --env MYSQL_ROOT_PASSWORD='mysql'\
> --replicas 1 \
> --name mysql \
> --reserve-cpu .5 --limit-cpu .5 --reserve-memory 256mb --limit-memory 256mb \
> mysql
81bu63v97p9rm81xfyxv9k11e
```

The service is created and the single task is scheduled. The service task does not fail as when the resource reserves exceeded the resource limit.

CHAPTER 7 CONFIGURING RESOURCES

~ \$ docker servi	ice ps mysql				
ID	NAME	IMAGE		NODE	
DESIRED STATE	CURRENT ST	ATE ERROR	PORTS		
4i1fpha53abs	mysql.1	mysql:1	atest	ip-172-31	-16-11.ec2.internal
Running	Running 33	seconds ago			
And a Docker	container is start	ted.			
~ \$ docker ps					
CONTAINER ID	IMAGE	COMMAND	CF	REATED	STATUS

PORTSNAMES14d5553f0393mysql:latest"docker-entrypoint..."34 seconds agoUp 33 seconds3306/tcpmysql.1.4i1fpha53absl4qky9dgafo8t

Rolling Update to Modify Resource Limits and Reserves

This section demonstrates a rolling update to set new CPU and memory limits and reserves. The service created in the previous section is used for updating in this section. Using the docker service update command, update the CPU and memory reserves and limits. The output of the command is the service name mysql (shown in italics).

```
~ $ docker service update --reserve-cpu 1 --limit-cpu 2 --reserve-memory 256mb
--limit-memory 512mb mysql
mysql
```

The resources are updated. Updating the resource specification for a service shuts down the service replica and starts a new replica with the new resource specification.

~ \$ docker servi	ice ls				
ID	NAME	MODE	REPLICAS	IMAGE	PORTS
81bu63v97p9r	mysql	replicated	1/1	<pre>mysql:latest</pre>	
~ \$ docker servi	ice ps mysql				
ID	NAME	IMAGE	NODE		
DESIRED STATE	CURRENT STATE	ERROR	PORTS		
xkis4mirgbtv	mysql.1	<pre>mysql:latest</pre>	ip-172-3	1-33-230.ec2.inte	rnal
Running	Running 14 seco	nds ago			
4i1fpha53abs	_ mysql.1	<pre>mysql:latest</pre>	ip-172-3	1-16-11.ec2.inter	nal
Shutdown	Shutdown 15 sec	onds ago			

The service resources configuration is updated.

```
"Reservations": {
          "NanoCPUs": 100000000,
          "MemoryBytes": 268435456
        }
     },
]
```

Resource Usage and Node Capacity

Resource usage cannot exceed node capacity. On the three-node Swarm (one manager and two worker nodes), recall that the node capacity is 1GB and 1 CPU.

Remove the mysql service that's already running and create a mysql service with three replicas that requests 4GB of memory. The service is created. The output of the command is the service ID (shown in italics).

```
~ $ docker service rm mysql
mysql
~ $ docker service create \
> --env MYSQL_ROOT_PASSWORD='mysql'\
> --replicas 3 \
> --name mysql \
> --reserve-memory=4GB\
> mysql
cqrihwij2znn4jkfe6hswxqr7
```

None of the service replicas is scheduled, as indicated by the Replicas column value of 0/3, because the requested capacity is more than the node capacity of a single node.

~ \$ docker servio	ce ls				
ID	NAME	MODE	REPLICAS	IMAGE	PORTS
cgrihwij2znn	mysql	replicated	0/3	mysql:latest	

The Current State of the replicas is listed as Pending.

~ \$ docker servi	ce ps mysql		
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
vm7z20krx3j6	mysql.1	mysql:latest	
Running	Pending 19 seconds ago		
exmsheo144ef	mysql.2	mysql:latest	
Running	Pending 19 seconds ago		
kiset9poqz2s	mysql.3	mysql:latest	
Running	Pending 19 seconds ago		

If a service that was previously running with all replicas is scaled up, some or all of the replicas could get de-scheduled. This happens if the resources required to run the new replicas exceed the available node capacity. As an example, remove the mysql service and create a new mysql service with resource settings within the provision of a node. The output of the command is the service ID (shown in italics).

```
CHAPTER 7 CONFIGURING RESOURCES

~ $ docker service rm mysql

mysql

~ $

~ $ docker service create \

> --env MYSQL_ROOT_PASSWORD='mysql'\

> --replicas 1 \

> --reserve-cpu .5 --reserve-memory 512mb \

> mysql

ysef8n02mhuwa7sxerc9jwjqx
```

The service is created and the single replica is running as indicated by the Replicas column value of 1/1.

~ \$ docker servio	ce ls				
ID	NAME	MODE	REPLICAS	IMAGE	PORTS
ysef8n02mhuw	mysql	replicated	1/1	mysql:latest	

Incrementally scale up the service to determine if all of the service replicas are scheduled. First, scale up to three replicas.

```
~ $ docker service scale mysql=3
mysql scaled to 3
```

The service description lists 3/3 Replicas as running.

~ \$ docker ser	vice ls				
ID	NAME	MODE	REPLICAS	IMAGE	PORTS
ysef8n02mhuw	mysql	replicated	3/3	mysql:latest	

The service replicas are scheduled, one replica on each node in the Swarm, using the *spread* scheduling strategy, which is discussed in more detail in Chapter 8.

~ \$ docker service	ps mysql			
ID .	NAME		IMAGE	NODE
DESIRED STATE	CURRENT	STATE	ERROR	PORTS
8kkkdns01690	mysql.1		mysql:latest	ip-172-31-16-11.ec2.internal
Running	Running	51 seconds	ago	
k209uge36bih	mysql.2		<pre>mysql:latest</pre>	ip-172-31-25-163.ec2.internal
Running	Running	16 seconds	ago	
oiublpclz9eu	mysql.3		mysql:latest	ip-172-31-33-230.ec2.internal
Running	Running	16 seconds	ago	

Scale the mysql service further up to replicas.

~ \$ docker service scale mysql=10
mysql scaled to 10

Only 3/10 of the replicas are listed as running.

~ \$ docker service]	ls				
ID	NAME	MODE	REPLICAS	IMAGE	PORTS
ysef8n02mhuw	mysql	replicated	3/10	mysql:latest	

Some of the replicas are Allocated but not scheduled for running on any node due to insufficient resources. The service replicas not running are listed with Current State set to Pending.

~ \$ docker service ps	mysql		
ID N	AME	IMAGE	NODE
DESIRED STATE C	URRENT STATE	ERROR	PORTS
8kkkdns01690 m	ysql.1	mysql:latest	ip-172-31-16-11.ec2.internal
Running Ru	unning about a mir	nute ago	
k209uge36bih my	ysql.2	mysql:latest	ip-172-31-25-163.ec2.internal
Running R	unning 35 seconds	ago	
oiublpclz9eu my	ysql.3	mysql:latest	ip-172-31-33-230.ec2.internal
Running R	unning 35 seconds	ago	
u807b7h0qvqc m	ysql.4	mysql:latest	
Running Po	ending 7 seconds a	igo	
jh2ep10sonxy my	ysql.5	mysql:latest	
Running Po	ending 7 seconds a	igo	
8d19osxa4fwf my	ysql.6	mysql:latest	
Running Po	ending 7 seconds a	igo	
k8hba8j5o9vi m	ysql.7	mysql:latest	
Running Po	ending 7 seconds a	igo	
ettk65bpin3b m	ysql.8	mysql:latest	
Running Po	ending 7 seconds a	igo	
i3otbqfsfvr7 m	ysql.9	mysql:latest	
Running Po	ending 7 seconds a	ago	
sxdi970o6d3b m	ysql.10	mysql:latest	
Running Po	ending 7 seconds a	ago	

Adding one or more new worker nodes could make the service reconcile its desired state and cause all the replicas to run. To demonstrate next, we scale up the CloudFormation stack to increase the number of worker nodes.

Scaling Up the Stack

To scale up the CloudFormation stack, select the Docker stack in the CloudFormation \succ Stacks table and choose Actions \succ Update Stack, as shown in Figure 7-8.

Create	Stack •	Actions •	Design temptate				C O
Filter:	Active - B	Create Change	Set For Current Stack				Showing 1 stack
St	ack Name	Update Stack		_	Status	Description	
	ocker	Delete Stack		700	CREATE COMPLETE	Docker CE for AWS 17.06.0-ce (17.06.0-ce-aws2)	

Figure 7-8. Choosing Actions ➤ *Update Stack*

The Update Docker Stack wizard starts. It's similar to the Create Stack wizard. In the Select Template, click on Next without modifying any settings. In Specify Details, increase Number of Swarm Worker Nodes? to 10, as shown in Figure 7-9. Click on Next.

CHAPTER 7 CONFIGURING RESOURCES

CloudFormation	✓ Stacks → Stack Deta	ill > Update Stack	
Update Docke	er stack		
Select Template	Specify Details		
Options Review	Specify parameter values. You	can use or change the default parameter values	, which are defined in the AWS CloudFormation template. Learn more.
	Stack name	Docker	1
	Parameters		
	Swarm Size		
	Number of Swarm managers?	1	Number of Swarm manager nodes (1, 3, 5)
	Number of Swarm worker nodes?	10	Number of worker nodes in the Swarm (0-1000).

Figure 7-9. Increasing the number of worker nodes to 10

In Preview Your Changes, click on Update, as shown in Figure 7-10.

Preview your changes

Based on yo	Based on your input, CloudFormation will change the following resources. For more information, choose View change set details.							
Action	Logical ID	Physical ID	Resource type	Replacement				
Modify	NodeAsg	Docker-NodeAsg-1OUA0XESLI58J	AWS::AutoScaling::AutoScalingGroup	False				
				Cancel Previous Update				
				40				

Figure 7-10. Click Update to preview your changes

When the update completes, the stack's status becomes UPDATE_COMPLETE, as shown in Figure 7-11.

CloudFormation	✓ Stacks			
Create Stack - Ad	tions • Design template			C O
Filter: Active . By Sta	ck Name			Showing 1 stack
Stack Name	Created Time	Status	Description	
Docker	2017-07-24 09:52:37 UTC-0700	UPDATE COMPLETE	Docker CE for AWS 17.06.0-ce (17.06.0-ce-aws2)	

Figure 7-11. Stack update is complete

The Swarm gets eight new worker nodes, for a total of 10 worker nodes. List the service description periodically (after an interval of few seconds) and, as new worker nodes are created, new replicas start to reconcile the current state with the desired state. The number of replicas in the Replicas column increases gradually within a few seconds. All the replicas for the mysql service start running, as indicated by 10/10 in the service listing.

ce ls				
NAME	MODE	REPLICAS	IMAGE	PORTS
mysql	replicated	3/10	mysql:latest	
ce ls				
NAME	MODE	REPLICAS	IMAGE	PORTS
mysql	replicated	6/10	mysql:latest	
ce ls				
NAME	MODE	REPLICAS	IMAGE	PORTS
mysql	replicated	9/10	mysql:latest	
ce ls				
NAME	MODE	REPLICAS	IMAGE	PORTS
mysql	replicated	10/10	mysql:latest	
	te ls MAME mysql te ls MAME mysql te ls NAME mysql te ls NAME mysql	te ls MODE MODE mysql replicated s NAME mysql replicate s NAME mysql replicate s NAME mysql replicate s NAME mysql repli	te ls NAME MODE REPLICAS mysql replicated 3/10 te ls NAME MODE REPLICAS mysql replicated 6/10 te ls NAME MODE REPLICAS mysql replicated 9/10 te ls NAME MODE REPLICAS mysql replicated 10/10	te ls NAME MODE REPLICAS IMAGE mysql replicated 3/10 mysql:latest te ls NAME MODE REPLICAS IMAGE mysql replicated 6/10 mysql:latest te ls NAME MODE REPLICAS IMAGE mysql replicated 9/10 mysql:latest te ls NAME MODE REPLICAS IMAGE mysql replicated 10/10 mysql:latest

Listing the service replicas lists all replicas as Running. The previously Pending replicas are scheduled on the new nodes.

~ \$ docker service	ps mysql					
ID	NAME		IMAGE	NODE		
DESIRED STATE	CURRENT	STATE	ERROR	PORTS		
8kkkdns01690	mysql.1		mysql:latest	ip-172-31-16-11.ec2.internal		
Running	Running	7 minutes ag	go			
k209uge36bih	mysql.2		mysql:latest	ip-172-31-25-163.ec2.internal		
Running	Running	6 minutes ag	<u>go</u>			
oiublpclz9eu	mysql.3		mysql:latest	ip-172-31-33-230.ec2.internal		
Running	Running	6 minutes ag	<u>go</u>			
u807b7h0qvqc	mysql.4		mysql:latest	ip-172-31-11-105.ec2.internal		
Running	Running	Running 45 seconds ago				
jh2ep10sonxy	mysql.5		mysql:latest	ip-172-31-13-141.ec2.internal		
Running	Running	about a minu	ite ago			
8d19osxa4fwf	mysql.6		mysql:latest	ip-172-31-24-10.ec2.internal		
Running	Running	about a minu	ite ago			
k8hba8j5o9vi	mysql.7		mysql:latest	ip-172-31-0-114.ec2.internal		
Running	Running	55 seconds a	igo			
ettk65bpin3b	mysql.8		mysql:latest	ip-172-31-5-127.ec2.internal		
Running	Running	Running about a minute ago				
i3otbqfsfvr7	mysql.9		mysql:latest	ip-172-31-35-209.ec2.internal		
Running	Running	24 seconds a	igo			
sxdi970o6d3b	mysql.10		mysql:latest	ip-172-31-21-57.ec2.internal		
Running	Running	49 seconds a	igo			

If the stack is updated again to decrease the number of worker nodes, some of the replicas shut down and are de-scheduled. After decreasing the number of worker nodes, the Replicas column lists only 5/10 replicas as running.

~ \$ docker service ls								
ID	NAME	MODE	REPLICAS	IMAGE	PORTS			
ysef8n02mhuw	mysql	replicated	5/10	mysql:latest				

Some of the service tasks are listed as Shutdown because some of the worker nodes have been removed from the Swarm.

~ \$ docker service	ps mysql				
ID	NAME	IMAGE	NODE		
DESIRED STATE	CURRENT STATE	ERROR	PORTS		
8kkkdns01690	mysql.1	mysql:latest	ip-172-31-16-11.ec2.internal		
Running	Running 10 minutes	ago			
ulknt3e5zxy1	mysql.2	mysql:latest			
Ready	Pending 3 seconds	ago			
k209uge36bih	_ mysql.2	mysql:latest	ip-172-31-25-163.ec2.internal		
Shutdown	Running 14 seconds	ago			
oiublpclz9eu	mysql.3	mysql:latest	ip-172-31-33-230.ec2.internal		
Running	Running 9 minutes	ago			
mh2fpioi441k	mysql.4	mysql:latest			
Running	Pending 3 seconds	ago			
u807b7h0qvqc	_ mysql.4	mysql:latest	v53huw84hskqsb3e8o0a2pmun		
Shutdown	Running about a mi	nute ago			
jzghd72nk0zc	mysql.5	mysql:latest			
Ready	Pending 3 seconds	ago			
jh2ep10sonxy	_ mysql.5	mysql:latest	ip-172-31-13-141.ec2.internal		
Shutdown	Running 14 seconds	ago			
8d19osxa4fwf	mysql.6	<pre>mysql:latest</pre>	ip-172-31-24-10.ec2.internal		
Running	Running 4 minutes	ago			
dlcgstxxkd9t	mysql.7	mysql:latest			
Running	Pending 3 seconds	ago			
ziqslz7u9d9l	_ mysql.7	mysql:latest	ip-172-31-43-179.ec2.internal		
Shutdown	Assigned 57 second	s ago			
k8hba8j5o9vi	_ mysql.7	<pre>mysql:latest</pre>	op1dzvmt5eyc74l6pcl5ut64p		
Shutdown	Running about a mi	nute ago			
ettk65bpin3b	mysql.8	mysql:latest	ip-172-31-5-127.ec2.internal		
Running	Running 4 minutes	ago			
i3otbqfsfvr7	mysql.9	mysql:latest	ip-172-31-35-209.ec2.internal		
Running	Running 3 minutes ago				
sxdi970o6d3b	mysql.10	mysql:latest	ip-172-31-21-57.ec2.internal		
Running	Running 12 seconds	ago			
		-			

Summary

This chapter discussed the resources model of Docker Swarm mode, which is based on resource reserves and resource limits. Reserved resources cannot be more than resource limits and resource allocation to service tasks is limited by the node capacity. The next chapter discusses scheduling in Docker Swarm mode.
CHAPTER 8

Scheduling

In Chapter 2, the Docker Swarm was introduced. In Chapter 4, Docker Swarm services were introduced. A service consists of zero or more service tasks (replicas), which it schedules on the nodes in a Swarm. The desired state of a service includes the number of tasks that must be run. Scheduling is defined as the process of placing a service task that is required to be run on a node in the Swarm to keep the desired state of a service, as illustrated in Figure 8-1. A service task may only be scheduled on a worker node. A manager node is also a worker node by default.



Figure 8-1. Scheduling

The Problem

Without a scheduling policy, the service tasks could get scheduled on a subset of nodes in a Swarm. As an example, all three tasks in a service could get scheduled on the same node in a Swarm, as illustrated in Figure 8-2.



Figure 8-2. Avoid scheduling all tasks on one node

Not using a scheduling policy could lead to the following problems:

- *Underutilization of resources in a Swarm*—If all the tasks are scheduled on a single node or a subset of nodes, the resource capacity of the other nodes is not utilized.
- Unbalanced utilization of resources—If all the tasks are scheduled on a single node or a subset of nodes, the resources on the nodes on which the tasks are scheduled are over-utilized and the tasks could even use up all the resource capacity without any scope for scaling the replicas.
- *Lack of locality*—Clients access a service's tasks based on node location. If all the service tasks are scheduled on a single node, the external clients that are accessing the service on other nodes cannot access the service locally, thereby incurring a network overhead in accessing a relatively remote task.
- *Single point of failure*—If all services are running on one node and that node has a problem, it results in downtime. Increasing redundancy across nodes obviates that problem.

The Solution

To overcome the issues discussed in the preceding section, service task scheduling in a Docker Swarm is based on a built-in scheduling policy. Docker Swarm mode uses the *spread* scheduling strategy to rank nodes for placement of a service task (replica). Node ranking is computed for scheduling of each task and a task is

scheduled on the node with the highest computed ranking. The *spread* scheduling strategy computes node rank based on the node's available CPU, RAM, and the number of containers already running on the node. The spread strategy optimizes for the node with the least number of containers. Load sharing is the objective of the spread strategy and results in tasks (containers) spread thinly and evenly over several machines in the Swarm. The expected outcome of the spread strategy is that if a single node or a small subset of nodes go down or become available, only a few tasks are lost and a majority of tasks in the Swarm continue to be available.

Note Because a container consumes resources during all states, including when it is exited, the spread strategy does not take into consideration the state of a container. It is recommended that a user remove stopped containers, because a node that would otherwise be eligible and suitable for scheduling a new task becomes unsuitable if it has several stopped containers.

The spread scheduling strategy does not take into consideration for which service a task is scheduled. Only the available and requested resources are used to schedule a new task. Scheduling using the spread scheduling policy is illustrated in Figure 8-3.



Figure 8-3. Using the spread scheduling policy

As a hypothetical example:

1. Start with three nodes, each with a capacity of 3GB and 3 CPUs and no containers running.

- 2. Create a mysql service with one replica, which requests resources of 1GB and 1 CPU. The first replica gets scheduled randomly on one of the three nodes in the Swarm as all nodes have the same ranking. If all the nodes have the same ranking, a new task gets scheduled randomly on one of the nodes.
- **3.** Scale the mysql service to three tasks. As one of the nodes is already loaded, the two new tasks are scheduled on the other two nodes, spreading one task to each node.
- 4. Scale the mysql service to five tasks. Two new tasks must be started and all the nodes have the same ranking because they have the same available resource capacity and the same number of containers running. The two new tasks are scheduled randomly on two of the nodes. As a result, two nodes have two tasks each and one node has one task.
- 5. Create another service for the nginx server with a desired state of two tasks, with each task requesting 0.5GB and 0.5 CPU. Both the tasks are scheduled on the node that has only the task of the mysql service, as it is the least loaded. As a result, two nodes have two tasks of mysql service and an available capacity of 1GB and 1 CPU, and one node has two tasks of nginx service and one task of mysql service and also an available resource capacity of 1GB and 1 CPU.
- 6. Scale the nginx service to three. Even though all nodes have the same available CPU and RAM, the new task is not scheduled randomly on one of the three nodes, but is scheduled on the node with the least number of containers. As a result, the new nginx task gets scheduled randomly on one of the nodes, with two tasks of mysql each. If the nodes have the same available CPU and RAM, the node with fewer containers (running or stopped) is selected for scheduling the new task.

This chapter covers the following topics:

- Setting the environment
- Creating and scheduling a service—the spread scheduling
- Desired state reconciliation
- Scheduling tasks limited by node resource capacity
- Adding service scheduling constraints
- Scheduling on a specific node
- Adding multiple scheduling constraints
- Adding node labels for scheduling
- Adding, updating, and removing service scheduling constraints
- Spread scheduling and global services

Setting the Environment

Create a CloudFormation stack using Docker for AWS consisting of one manager node and two worker nodes. Docker for AWS was introduced in Chapter 3. The stack is shown in Figure 8-4.

A Introducing Sta	ckSets			0
AWS StackSet is	a container for a set of AWS CloudFormation	stacks and allows you to create start	stacks across multiple AWS Accounts and AWS Regions. Open the St ed.	ackSets console to get
Citcuic Church	ocogn compare			
and the second				
Filter: Active - By Stace	Name			Showing 1 stack
Filter: Active - By Stac) Stack Name	Name Created Time	Status	Description	Showing 1 stack

Figure 8-4. CloudFormation stack

The three EC2 instances in the stack are shown in Figure 8-5.

Q Filter by tags	s and attributes or	search by keyword					0	< <	1 to 3 of 3	>
Name	*	Instance ID	Instance Type	Availability Zone -	Instance State 👻	Status Checks 👻	Alarm Stat	us	Public DNS	(IPv4)
Docker-wo	rker	i-01a9e249cb984dc6	b t2.micro	us-east-1a	running	2/2 checks	None	70	ec2-52-203-	21-60.
Docker-wo	rker	i-03a1baacb6feca3c0	t2.micro	us-east-1b	running	2/2 checks	None	.0	ec2-34-205-	143-27
Docker-Ma	nager	i-06f95103949862c4a	a t2.micro	us-east-1b	running	2/2 checks	None		ec2-54-84-1	33-157
_									_	
nstance: i-06	f95103949862c	la (Docker-Manage	r) Public DNS: ec2	2-54-84-133-157.com	pute-1.amazonaws.	com				
nstance: i-06 Description	Status Check	a (Docker-Manage Monitoring	r) Public DNS: ec2 Tags	2-54-84-133-157.com	pute-1.amazonaws.	com				
Description	Status Check	s Monitoring i-06f9510394986	r) Public DNS: ec2 Tags 2c4a	2-54-84-133-157.com	pute-1.amazonaws. Public DNS (IP	(4) ec2-54-84-133 1.amazonaws	3-157.compute)-	-	
Description	f95103949862cd Status Check Instance I Instance stal	Gocker-Manage Monitoring i-06/9510394986 e running	r) Public DNS: ec2 Tags 2c4a	-54-84-133-157.com	Public DNS (IP	 ec2-54-84-133 1.amazonaws IP 54.84.133.157 	3-157.compute .com)-	-	
Description	55103949862c Status Check Instance I Instance stat	Konitoring Monitoring i-06f9510394986 running t2.micro	r) Public DNS: ec2 Tags 2c4a	-54-84-133-157.com	Public DNS (IP)	ec2-54-84-133 1.amazonaws IP 54.84.133.157 Ps -	3-157.compute .com)+		
nstance: i-06	f95103949862c Status Check Instance I Instance stal Instance typ Elastic IF	Ia (Docker-Manage) s Monitoring D i-06/9510394986 e running e t2.micro s 12	r) Public DNS: ec2 Tags 2c4a	2-54-84-133-157.com	Public DNS (IP Public DNS (IP Public DNS (IP IPv6 I Private DI	 ec2-54-84-133 1.amazonaws IP 54.84.133.157 Ps - IP-172-31-25-1 	3-157.compute .com 7 121.ec2.intern)- al	-	
nstance: 1-06	Status Check Status Check Instance I Instance stal Instance typ Elastic IF Availability zon	Ia (Docker-Manage s Monitoring p i-0619510394986 e running e t2.micro s us-east-1b	r) Public DNS: ec2 Tags 2c4a	2-54-84-133-157.com	Public DNS (IP Public DNS (IP Pv4 Public IPv6 I Private D Private D	 ec2-54-84-133 1.amazonaws IP 54.84.133.157 Ps - IP 172-31-25-1 172.31.25.121 	3-157.compute .com 7 121.ec2.intern	a.		

Figure 8-5. EC2 instances for the Docker swarm

SSH Login to the Swarm manager using the public IP address, which may be obtained from the EC2 console, as shown in Figure 8-5.

```
[root@localhost ~]# ssh -i "docker.pem" docker@54.84.133.157
Welcome to Docker!
```

List the nodes in the Swarm; three nodes should be listed.

~ \$ docker node ls				
ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER
				STATUS
Owaa5g3b6j641xtwsygvjvwc1	ip-172-31-0-147.ec2.internal	Ready	Active	
e7vigin0luuo1kynjnl33v9pa	ip-172-31-29-67.ec2.internal	Ready	Active	
ptm7e0p346zwypos7wnpcm72d *	ip-172-31-25-121.ec2.internal	Ready	Active	Leader

Creating and Scheduling a Service: The Spread Scheduling

First, we discuss the default spread scheduling using a MySQL database service as an example. From the Swarm manager node, run the following command to create a five-replica service for MySQL. The output is the service ID (shown in italics).

```
~ $ docker service create \
> --env MYSQL_ROOT_PASSWORD='mysql'\
> --replicas 5 \
> --name mysql \
> mysql
10npemnoz4x1lh3sv5umab8uo
```

Subsequently, list the services using docker service 1s. Initially, the REPLICAS column could be 0/5, indicating that none of the replicas are scheduled and running yet.

~ \$ docker set	rvice ls				
ID	NAME	MODE	REPLICAS	IMAGE	PORTS
1onpemnoz4x1	mysql	replicated	0/5	mysql:latest	

Run the command again after a while; all the replicas should be running as indicated by a 5/5 in the REPLICAS column. List the service replicas using the docker service ps mysql command. The tasks should be running or preparing to run.

~ \$ docker serv	ice ps mysq	1		
ID	NAME	IMAGE	NODE	
DESIRED STATE	CURRENT S	TATE	ERROR	PORTS
fwjbu3gt2zn0	mysql.1	mysql:latest	ip-172-	31-0-147.ec2.internal
Running	Preparing	8 seconds ago		
w0521ik1awjf	mysql.2	<pre>mysql:latest</pre>	ip-172-	31-29-67.ec2.internal
Running	Preparing	8 seconds ago		
z9wn2nrzfzt8	mysql.3	mysql:latest	ip-172-	31-0-147.ec2.internal
Running	Preparing	8 seconds ago		
tm8jbque3xbb	mysql.4	mysql:latest	ip-172-	31-25-121.ec2.internal
Running	Preparing	8 seconds ago		
7drxfy3vbmp5	mysql.5	mysql:latest	ip-172-	31-29-67.ec2.internal
Running	Preparing	8 seconds ago		

Following the spread scheduling strategy, two of the replicas are listed as scheduled on one of the worker nodes, two on the other worker node, and one on the manager node. Because of the odd number of replicas, the placement cannot be completely evenly distributed, but a single node does not have more than two replicas.

To see how the spread scheduling strategy distributes the replicas evenly across a Swarm, scale the service to six replicas. The output of the docker service scale command is in italics.

~ \$ docker service scale mysql=6
mysql scaled to 6

Subsequently, list the replicas. Each node has two replicas scheduled on it, as the spread scheduling policy is designed to schedule.

e ps mysql			
AME :	IMAGE	NODE	
CURRENT ST	ATE	ERROR	PORTS
ysql.1 r	mysql:latest	ip-172-31-	0-147.ec2.internal
Running 13	seconds ago		
ysql.2 r	mysql:latest	ip-172-31-	29-67.ec2.internal
Running 12	seconds ago		
ysql.3 r	mysql:latest	ip-172-31-	0-147.ec2.internal
Running 13	seconds ago		
ysql.4 r	mysql:latest	ip-172-31-	25-121.ec2.internal
Running 8 :	seconds ago		
ysql.5 r	mysql:latest	ip-172-31-	29-67.ec2.internal
Running 12	seconds ago		
ysql.6 r	mysql:latest	ip-172-31-	25-121.ec2.internal
Running 5 :	seconds ago		
	e ps mysql AME CURRENT ST ysql.1 1 Running 13 ysql.2 1 Running 12 ysql.3 1 Running 13 ysql.4 1 Running 8 Sysql.5 1 Running 12 ysql.6 1 Running 5	e ps mysql AME IMAGE CURRENT STATE ysql.1 mysql:latest Running 13 seconds ago ysql.2 mysql:latest Running 12 seconds ago ysql.3 mysql:latest Running 13 seconds ago ysql.4 mysql:latest Running 8 seconds ago ysql.5 mysql:latest Running 12 seconds ago ysql.6 mysql:latest Running 5 seconds ago	e ps mysqlAMEIMAGENODECURRENT STATEERRORysql.1mysql:latestip-172-31-Running 13 seconds agoysql.2mysql:latestysql.2mysql:latestip-172-31-Running 12 seconds agoysql.3mysql:latestysql.3mysql:latestip-172-31-Running 13 seconds agoysql.4mysql:latestysql.4mysql:latestip-172-31-Running 8 seconds agoysql.5mysql:latestysql.5mysql:latestip-172-31-Running 12 seconds agoysql.6mysql:latestysql.6mysql:latestip-172-31-Running 5 seconds agoip-172-31-

As a service replica or task is nothing but a slot to run a container, each node runs two containers for the mysql service.

To further demonstrate spread scheduling, scale down the service to three tasks. The command output is in italics.

~ \$ docker service scale mysql=3
mysql scaled to 3

List the service tasks. Each node has one task running on it, which again is an evenly spread scheduling of tasks.

~ \$ docker serv	vice ps mys	sql		
ID	NAME	IMAGE	NODE	
DESIRED STATE	CURRENT	STATE	ERROR	PORTS
w0521ik1awjf	mysql.2	mysql:latest	ip-172-31	L-29-67.ec2.internal
Running	Running	40 seconds ago		
z9wn2nrzfzt8	mysql.3	mysql:latest	ip-172-31	L-0-147.ec2.internal
Running	Running	41 seconds ago		
utjo8lwbtzf7	mysql.6	mysql:latest	ip-172-31	L-25-121.ec2.internal
Running	Running	33 seconds ago		

Desired State Reconciliation

When a service is created or is scaled up or down, the service initially has a discrepancy between the *current state* and the *desired state*. The different values for the desired state are ready, running, shutdown, and accepted. Docker services are designed for desired state reconciliation, which implies that the Swarm manager continuously monitors the cluster state to reconcile any differences between the desired state of a service and the current state. The current state of a task can be assigned, preparing, ready, running, shutdown, or pending. A task that has been assigned to a node but is not currently running is in the assigned state. A task that has desired state as running and is preparing to run is in the preparing current state. A task is in the pending state if no node in the Swarm can run the task.

In the following task listing, some tasks have a desired state and current state of running. These tasks have reconciled their desired state. One task has a desired state set to running, but the current state is pending. Another task has a desired state set to shutdown and a current state set to assigned.

~ \$ docker serv	ice ps mysql			
ID	NAME	IMAGE	NODE	
DESIRED STATE	CURRENT STA	ATE	ERROR	PORTS
opxf4ne7iyy6	mysql.1	mysql:latest	ip-172	-31-25-121.ec2.internal
Running	Running 9 m	ninutes ago		
x30y3jlea047	mysql.2	mysql:latest	ip-172	-31-29-67.ec2.internal
Running	Running 8 m	ninutes ago		
w4ivsbvwqqzq	mysql.3	mysql:latest	ip-172	-31-2-177.ec2.internal
Running	Running 4 m	ninutes ago		
j9lp08ojofj7	mysql.4	mysql:latest		
Running	Pending 28	seconds ago		
ph1zpsjsvp69	_ mysql.4	mysql:latest	ip-172	-31-7-137.ec2.internal
Shutdown	Assigned 33	seconds ago		
d3oxy6hxfjh3	_ mysql.4	mysql:latest	ip-172	-31-40-70.ec2.internal
Shutdown	Running 43	seconds ago		
ic331aasjpdm	mysql.5	mysql:latest	ip-172	-31-44-104.ec2.internal
Running	Running 8 m	ninutes ago		

In an earlier task listing, all tasks were in the current state preparing and the desired state running. Swarm mode is designed to reconcile the desired state as much as feasible, implying that if node resources are available, the desired number of replicas runs. To demonstrate, update the Docker for AWS CloudFormation stack by choosing Actions ➤ Update Stack, as shown in Figure 8-6.



Figure 8-6. Updating a stack

CloudFormation	✓ Stacks → Stack Deta	il > Update Stack			
Update Docker	stack				
Select Template	Specify Details				
Options Review	Specify parameter values. You can use or change the default parameter values, which are defined in the AWS CloudFormation template. Learn more				
	Stack name	Docker			
	Parameters				
	Swarm Size				
	Number of Swarm managers?	Number of Swarm manager nodes (1, 3, 5)			
	Number of Swarm worker	1 Number of worker nodes in the Swarm (0-1000).			

Decrease the number of worker nodes from two to one, as shown in Figure 8-7.

Figure 8-7. Decreasing the number of worker nodes to one

Subsequently, list the service replicas from the Swarm manager node.

docker service ps mysql

The service replicas running on the Swarm worker node that was made to leave the Swarm are listed as shutdown. New replicas are started on the remaining two nodes in the Swarm to reconcile the desired state.

~ \$ docker serv	/ice ps mysql			
ID	NAME	IMAGE	NODE	
DESIRED STATE	CURRENT ST	ATE	ERROR PORTS	
p14bbk7ij1mt	mysql.1	mysql:latest	ip-172-31-29-67.ec	2.internal
Running	Running 5	minutes ago		
w0521ik1awjf	mysql.2	mysql:latest	ip-172-31-29-67.ec	2.internal
Running	Running 7	minutes ago		
uatsaay7axlc	mysql.3	mysql:latest	ip-172-31-25-121.e	c2.internal
Running	Running ab	out a minute ago		
z9wn2nrzfzt8	_ mysql.3	mysql:latest	0waa5g3b6j641xtwsy	gvjvwc1
Shutdown	Running 2	minutes ago		
w1tlw0fom42q	mysql.4	mysql:latest	ip-172-31-29-67.ec	2.internal
Running	Running ab	out a minute ago		
qc75buhzzct3	_ mysql.4	mysql:latest	0waa5g3b6j641xtwsy	gvjvwc1
Shutdown	Running 2	minutes ago		
s09ts9s8np3d	mysql.5	mysql:latest	ip-172-31-25-121.e	c2.internal
Running	Running 5	minutes ago		
utjo8lwbtzf7	mysql.6	mysql:latest	ip-172-31-25-121.e	c2.internal
Running	Running 7	minutes ago		

Listing only the replicas with a desired state of running, the six replicas are listed as scheduled evenly between the two nodes—three replicas on the manager node and three replicas on the worker node.

~ \$ docker serv	/ice ps -f	<pre>desired-state=re</pre>	unning mys	sql
ID	NAME	IMAGE	NODE	
DESIRED STATE	CURRENT	STATE	ERROR	PORTS
p14bbk7ij1mt	mysql.1	mysql:latest	ip-172-	-31-29-67.ec2.internal
Running	Running	6 minutes ago		
w0521ik1awjf	mysql.2	mysql:latest	ip-172-	-31-29-67.ec2.internal
Running	Running	8 minutes ago		
uatsaay7axlc	mysql.3	mysql:latest	ip-172-	-31-25-121.ec2.internal
Running	Running	2 minutes ago		
w1tlw0fom42q	mysql.4	mysql:latest	ip-172-	-31-29-67.ec2.internal
Running	Running	2 minutes ago		
s09ts9s8np3d	mysql.5	mysql:latest	ip-172-	-31-25-121.ec2.internal
Running	Running	6 minutes ago		
utjo8lwbtzf7	mysql.6	mysql:latest	ip-172-	-31-25-121.ec2.internal
Running	Running	8 minutes ago		

The spread scheduling strategy does not reschedule already running replicas to achieve even spread across a Swarm if new nodes are added to the Swarm. To demonstrate this, we increase the number of worker nodes back to two, as shown in Figure 8-8.

Update Docker stack

Select Template	Specify Details	Specify Details				
Options Review	Specify parameter values. You	can use or change the default parameter values	, which are defined in the AWS CloudFormation template. Learn more.			
	Stack name	Docker	1			
	Parameters					
	Swarm Size					
	Number of Swarm managers?	1	Number of Swarm manager nodes (1, 3, 5)			
	Number of Swarm worker nodes?	2	Number of worker nodes in the Swarm (0-1000).			

Figure 8-8. Re-adding a worker node to Swarm

Adding a node to a swarm does not shut down replicas on other nodes and start replicas on the new node. Listing the running replicas does not indicate a replacement of the service replicas. Service replicas continue to run on the nodes they were running on before the new node was added—three on the manager node and three on the worker node.

~ \$ docker serv	ice ps mysql				
ID	NAME	IMAGE	NODE		
DESIRED STATE	CURRENT ST	ATE	ERROR	PORTS	
p14bbk7ij1mt	mysql.1	mysql:latest	ip-17	2-31-29-67.ec2.inte	rnal
Running	Running 15	minutes ago			
w0521ik1awjf	mysql.2	<pre>mysql:latest</pre>	ip-17	2-31-29-67.ec2.inte	rnal
Running	Running 17	minutes ago			
uatsaay7axlc	mysql.3	mysql:latest	ip-17	2-31-25-121.ec2.int	ernal
Running	Running 12	minutes ago			
z9wn2nrzfzt8	_ mysql.3	mysql:latest	Owaa5	g3b6j641xtwsygvjvwc	1
Shutdown	Running 13	minutes ago			
w1tlw0fom42q	mysql.4	mysql:latest	ip-17	2-31-29-67.ec2.inte	rnal
Running	Running 12	minutes ago			
qc75buhzzct3	_ mysql.4	mysql:latest	Owaa5	g3b6j641xtwsygvjvwc	1
Shutdown	Running 13	minutes ago			
s09ts9s8np3d	mysql.5	mysql:latest	ip-17	2-31-25-121.ec2.int	ernal
Running	Running 15	minutes ago			
utjo8lwbtzf7	mysql.6	mysql:latest	ip-17	2-31-25-121.ec2.int	ernal
Running	Running 17	minutes ago			

Scheduling Tasks Limited by Node Resource Capacity

The scheduling policy is limited by the available node resources, implying that service replicas cannot be made to run if not enough node resources in terms of CPU and memory are available. Resource usage cannot exceed node capacity. The replicas are still allocated to the service to define the desired state but may not be running due to insufficient resources. To demonstrate this, we remove the service mysql and create the service again with the specified resource requests and limits. Command outputs are shown in italics.

```
~ $ docker service rm mysql
mysql
~ $ docker service create \
> --env MYSQL_ROOT_PASSWORD='mysql'\
> --replicas 1 \
> --name mysql \
> --reserve-cpu 1 --limit-cpu 2 --reserve-memory 256mb --limit-memory 512mb mysql
Oqe2thyOdlviroli6k8thist1
```

Listing the services indicates that one replica of the service is created.

~ \$ docker ser	vice ls				
ID	NAME	MODE	REPLICAS	IMAGE	PORTS
0qe2thy0dlvi	mysql	replicated	1/1	mysql:latest	

The single replica is scheduled on the manager node, which is chosen randomly if all nodes in a Swarm have the same node ranking.

~ \$ docker service ps mysql ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR PORTS opxf4ne7iyy6 mysql.1 mysql:latest ip-172-31-25-121.ec2.internal Running Running 8 seconds ago

Next, to potentially make the service replicas consume more resources than available, scale the service to five replicas.

```
~ $ docker service scale mysql=5
mysql scaled to 5
```

Listing the services indicates that 3/5 Replicas are running.

~ \$ docker ser	vice ls				
ID	NAME	MODE	REPLICAS	IMAGE	PORTS
0qe2thy0dlvi	mysql	replicated	3/5	mysql:latest	

Listing the service replicas indicates that some of the replicas are pending instead of running.

~ \$ docker serv	ice ps mys	ql		
ID	NAME	IMAGE	NODE	
DESIRED STATE	CURRENT	STATE	ERROR	PORTS
opxf4ne7iyy6	mysql.1	mysql:latest	ip-172-	-31-25-121.ec2.internal
Running	Running	4 minutes ago		
x30y3jlea047	mysql.2	mysql:latest	ip-172-	-31-29-67.ec2.internal
Running	Running	3 minutes ago		
w4ivsbvwqqzq	mysql.3	mysql:latest		
Running	Pending	3 minutes ago		
d3oxy6hxfjh3	mysql.4	mysql:latest		
Running	Pending	3 minutes ago		
ic331aasjpdm	mysql.5	mysql:latest	ip-172-	-31-44-104.ec2.internal
Running	Running	3 minutes ago		

The pending state implies that the replicas are allocated to the service but not scheduled on any node yet. Only three replicas could run based on the requested resources and available node resources, one on each node.

Because the replicas are not scheduled due to lack of resources, we add one or more new worker nodes to potentially schedule the replicas to reconcile the desired state. Increase the number of worker nodes to five, as shown in Figure 8-9.

Update Docker stack

Select Template	Specify Details	Specify Details					
Specify Details Options Review	Specify parameter values. You	can use or change the default parameter valu	es, which are defined in the AWS CloudFormation template. Learn more.				
	Stack name	Docker					
	Parameters						
	Swarm Size						
	Number of Swarm managers?	1	Number of Swarm manager nodes (1, 3, 5)				
	Number of Swarm worker nodes?	5	Number of worker nodes in the Swarm (0-1000).				

Figure 8-9. Increasing the number of worker nodes to five

The Swarm should list six nodes after a new node is added. As resources became available for the pending tasks, the tasks get scheduled and start running.

~ \$ docker serv	/ice ps mys	ql		
ID	NAME	IMAGE	NODE	
DESIRED STATE	CURRE	NT STATE	ERROR	PORTS
opxf4ne7iyy6	mysql.1	mysql:latest	ip-172-31-25-121	.ec2.internal
Running	Runni	ng 5 minutes ago		
x30y3jlea047	mysql.2	<pre>mysql:latest</pre>	ip-172-31-29-67.	ec2.internal
Running	Runni	ng 4 minutes ago		
w4ivsbvwqqzq	mysql.3	mysql:latest	ip-172-31-2-177.	ec2.internal
Running	Runni	ng 21 seconds ago	0	
d3oxy6hxfjh3	mysql.4	<pre>mysql:latest</pre>	ip-172-31-40-70.	ec2.internal
Running	Prepa	ring 30 seconds a	ago	
ic331aasjpdm	mysql.5	mysql:latest	ip-172-31-44-104	.ec2.internal
Running	Runni	ng 4 minutes ago		

If the number of worker nodes is decreased, some of the tasks are descheduled, as indicated by the shutdown desired state.

~ \$ docker serv:	ice ps mys	ql		
ID	NAME	IMAGE	NODE	
DESIRED STATE	CURRENT	STATE	ERROR	PORTS
opxf4ne7iyy6	mysql.1	mysql:latest	ip-17	2-31-25-121.ec2.internal
Running	Running	9 minutes ago		
x30y3jlea047	mysql.2	<pre>mysql:latest</pre>	ip-17	2-31-29-67.ec2.internal
Running	Running	8 minutes ago		
w4ivsbvwqqzq	mysql.3	mysql:latest	ip-17	2-31-2-177.ec2.internal
Running	Running	4 minutes ago		
j9lp08ojofj7	mysql.4	mysql:latest		
Running	Pending	28 seconds ago		
ph1zpsjsvp69	_ mysql	.4 mysql:latest	ip-17	2-31-7-137.ec2.internal
Shutdown	Assigned	33 seconds ago		
d3oxy6hxfjh3	_ mysql	.4 mysql:latest	ip-17	2-31-40-70.ec2.internal
Shutdown	Running	43 seconds ago		
ic331aasjpdm	mysql.5	mysql:latest	ip-17	2-31-44-104.ec2.internal
Running	Running	8 minutes ago		

Updating the service to lower CPU and memory resource usage reserved only updates the UpdateConfig for the service. This does not lower the resource usage of the already running tasks or make pending or shutdown tasks run. As an example, lower the resource reserves and limits for the mysql service when some of the tasks are pending or shutdown due to lack of resources.

```
~ $ docker service update --reserve-cpu .1 --limit-cpu .5 --reserve-memory 64mb
--limit-memory 128mb mysql
mysql
```

The UpdateConfig gets modified, but only applies to new replicas created after that point.

Only three of the replicas in the mysql service are actually running.

~ \$ docker ser	vice ps -f	<pre>desired-state=r</pre>	running mysql
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT	STATE	ERROR PORTS
opxf4ne7iyy6	mysql.1	mysql:latest	ip-172-31-25-121.ec2.interna
Running	Running	10 minutes ago	
x30y3jlea047	mysql.2	mysql:latest	ip-172-31-29-67.ec2.internal
Running	Running	10 minutes ago	
w4ivsbvwqqzq	mysql.3	<pre>mysql:latest</pre>	ip-172-31-2-177.ec2.internal
Running	Running	5 minutes ago	
rm9uj4qevt5b	mysql.5	<pre>mysql:latest</pre>	
Running	Pending	33 seconds ago	

To force the service tasks to use the new resource settings, scale down the service to one task and then scale back up to five tasks.

~ \$ docker service scale mysql=1 mysql scaled to 1 ~ \$ docker service scale mysql=5 mysql scaled to 5

All five tasks are now running.

~ \$ docker serv	ice ps mysql			
ID	NAME	IMAGE	NODE	
DESIRED STATE	CURRENT ST	ATE	ERROR	PORTS
anai3mptbnkp	mysql.1	<pre>mysql:latest</pre>	ip-172	-31-2-177.ec2.internal
Running	Running 17	seconds ago		
opxf4ne7iyy6	_ mysql.1	<pre>mysql:latest</pre>	ip-172	-31-25-121.ec2.internal
Shutdown	Shutdown 18	8 seconds ago		
lmkn8150t334	mysql.2	mysql:latest	ip-172	-31-25-121.ec2.internal
Running	Running 10	seconds ago		
7uz7q86wnzn4	mysql.3	mysql:latest	ip-172	-31-2-177.ec2.internal
Running	Running 11	seconds ago		
ubh4m39aw8m9	mysql.4	mysql:latest	ip-172	-31-29-67.ec2.internal
Running	Running 11	seconds ago		
56pnrzajogvs	mysql.5	mysql:latest	ip-172	-31-25-121.ec2.internal
Running	Running 10	seconds ago		

Adding Service Scheduling Constraints

Docker Swarm supports placement or scheduling constraints for scheduling new tasks. Service placement constraints are additional criteria for placement of service tasks and could be based on node attributes, metadata, and engine metadata. The Swarm scheduler uses the following sequence to schedule a service task.

- Does the node satisfy all the placement constraints? 1.
- 2. Does a node meet the scheduling policy requirements of an even spread?
- Does the node have sufficient resources to schedule a task? 3.

A placement constraint may be added using the --constraint option with the docker service create command. For an already running service, constraints may be added and removed with the --constraint-add and --constraint-rm options, respectively, with the docker service update command. The node attributes discussed in Table 8-1 may be used to specify constraints.

Node Attribute	Description	Example
node.id	Specifies the node ID. Node IDs are listed using the docker node 1s command.	node.id==a3r56hj7y
node.hostname	Specifies the node's hostname. The node's hostname is listed with the docker node 1s command.	node.hostname!=ip-10-0-0- ec2.internal
node.role	Specifies the node role, which is one of worker or manager.	node.role==manager
node.labels	Specifies the node labels added by a user. A label is a key-value pair. When adding a node label, the node.labels. prefix is to be omitted and gets added automatically. Adding and using node labels is discussed in a subsequent section.	node.labels.db==mysql
engine.labels	Docker Engine labels such as drivers, operating system, version.	engine.labels.os==coreos

Next, we discuss some examples of using scheduling constraints.

Scheduling on a Specific Node

In this section we schedule service replicas on specific nodes in a Swarm. List the node IDs with the docker node 1s command. The Swarm has the following three nodes available for scheduling.

~ \$ docker node ls ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS
81h6uvu8uq0emnovzkg6v7mzg	ip-172-31-2-177.ec2.internal	Ready	Active	
e7vigin0luuo1kynjnl33v9pa	ip-172-31-29-67.ec2.internal	Ready	Active	
ptm7e0p346zwypos7wnpcm72d *	ip-172-31-25-121.ec2.internal	Ready	Active	Leader

We can schedule a service by node role. Create a mysql service with the placement constraint that the service tasks be scheduled on worker nodes only. First, remove the mysql service if it's already running

```
~ $ docker service rm mysql
mysql
~ $ docker service create \
> --env MYSQL_ROOT_PASSWORD='mysql'\
> --replicas 3 \
> --constraint node.role==worker \
> --name mysql \
> mysql
nzqte4zac1x8itx6t98y5gi42
```

The service is created and three tasks are scheduled only on the two worker nodes, as listed in the running service tasks.

~ \$ docker set	rvice ps -f	desired-state=ru	unning mysql	
ID	NAME	IMAGE	NODE	
DESIRED STATE	CURRENT	STATE	ERROR P	ORTS
f5t15mnrftOh	mysql.1	mysql:latest	ip-172-31-2	9-67.ec2.internal
Running	Running	19 seconds ago		
oxvq4ljuq6yz	mysql.2	mysql:latest	ip-172-31-2	-177.ec2.internal
Running	Running	19 seconds ago		
k5jo862lvsxf	mysql.3	<pre>mysql:latest</pre>	ip-172-31-2	-177.ec2.internal
Running	Running	19 seconds ago		

Next, we use the node ID to schedule a service's tasks. Copy the node ID for the manager node, which is also the leader in the Swarm being the only manager node. Substitute the node ID in the following command to create a service for the MySQL database and schedule replicas only on the manager node.

```
docker service create \
    --env MYSQL_ROOT_PASSWORD='mysql'\
    --replicas 3 \
    --constraint node.id ==<nodeid>
    --name mysql \
    mysql
```

A service is created with three tasks. Command output is shown in italics.

```
~ $ docker service create \
> --env MYSQL_ROOT_PASSWORD='mysql'\
> --replicas 3 \
> --constraint node.id==ptm7e0p346zwypos7wnpcm72d\
> --name mysql \
> mysql
u1qi6zqnch9hn7x6k516axq7h
```

All the three replicas of the service are scheduled on the manager node only.

~ \$ docker serv	ice ps -f	desired-state=run	ning mysql	
ID	NAME	IMAGE	NODE	
DESIRED STATE	CURRENT	STATE	ERROR	PORTS
lbttu95qdjvy	mysql.1	mysql:latest	ip-172-31-	-25-121.ec2.internal
Running	Running	21 seconds ago		
89x0z94on0fb	mysql.2	<pre>mysql:latest</pre>	ip-172-31-	-25-121.ec2.internal
Running	Running	21 seconds ago		
3s6508aimdaj	mysql.3	mysql:latest	ip-172-31-	-25-121.ec2.internal
Running	Running	22 seconds ago		

Adding Multiple Scheduling Constraints

Multiple node constraints may also be specified and every constraint expression must be met using AND for the scheduler to schedule a replica on a node. As an example, we create a service with two roles, one that constrains the node role to worker and the other constrains the node hostname not to be a specific hostname ip-172-31-2-177.ec2.internal.

```
~ $ docker service create \
> --env MYSQL_ROOT_PASSWORD='mysql'\
> --replicas 3 \
> --constraint node.role==worker \
> --constraint node.hostname!=ip-172-31-2-177.ec2.internal\
> --name mysql \
> mysql
87g0c8kauhz8yb4wv2ryc2vqr
```

A service gets created. Listing the services lists 3/3 replicas as running.

~ \$ docker serv	vice ls				
ID	NAME	MODE	REPLICAS	IMAGE	PORTS
87g0c8kauhz8	mysql	replicated	3/3	mysql:latest	

Listing the service tasks indicates that all tasks are scheduled on a single worker node. The two constraints are met: the node is a worker node and not the worker node with hostname ip-172-31-2-177.ec2. internal.

~ \$ docker serv	vice ps mys	ql		
ID	NAME	IMAGE	NODE	
DESIRED STATE	CURRENT	STATE	ERROR	PORTS
jlfk79mb6m6a	mysql.1	mysql:latest	ip-172-31	1-29-67.ec2.internal
Running	Running	13 seconds ago		
if5y39ky884q	mysql.2	mysql:latest	ip-172-31	1-29-67.ec2.internal
Running	Running	13 seconds ago		
zctm6mzbl4du	mysql.3	mysql:latest	ip-172-31	1-29-67.ec2.internal
Running	Running	13 seconds ago		

If the mysql service is updated to remove the constraints, the spread scheduling strategy reschedules the tasks based on node ranking. As an example, update the service to remove the two placement constraints added. A constraint is removed with the -constraint-rm option of the docker service update command.

```
~ $ docker service update \
> --constraint-rm node.role==worker \
> --constraint-rm node.hostname!=ip-172-31-2-177.ec2.internal\
> mysql
mysql
```

When a service is updated to remove constraints, all the service tasks are shut down and new service tasks are started. The new service tasks are started, one each on the three nodes in the Swarm.

~ \$ docker serv	vice ps mysql			
ID	NAME	IMAGE	NODE	
DESIRED STATE	CURRENT STAT	ΓE	ERROR	PORTS
d22bkgteivot	mysql.1	mysql:latest	ip-172-31-29	9-67.ec2.internal
Ready	Ready less t	than a second ago		
jlfk79mb6m6a	_ mysql.1	mysql:latest	ip-172-31-29	9-67.ec2.internal
Shutdown	Running 1 se	econd ago		
mp757499j3io	mysql.2	mysql:latest	ip-172-31-2-	-177.ec2.internal
Running	Running 1 se	econd ago		
if5y39ky884q	_ mysql.2	mysql:latest	ip-172-31-29	9-67.ec2.internal
Shutdown	Shutdown 2 s	seconds ago		
jtdxucteb0fl	mysql.3	mysql:latest	ip-172-31-2	5-121.ec2.internal
Running	Running 4 se	econds ago		
zctm6mzbl4du	_ mysql.3	mysql:latest	ip-172-31-29	9-67.ec2.internal
Shutdown	Shutdown 5 s	seconds ago		

List only the running tasks. One task is listed running on each node.

~ \$ docker serv	ice ps -f	desired-state=rur	ning mysql	
ID	NAME	IMAGE	NODE	
DESIRED STATE	CURRENT	STATE	ERROR	PORTS
d22bkgteivot	mysql.1	mysql:latest	ip-172-31-	-29-67.ec2.internal
Running	Running	46 seconds ago		
mp757499j3io	mysql.2	mysql:latest	ip-172-31-	-2-177.ec2.internal
Running	Running	49 seconds ago		
jtdxucteb0fl	mysql.3	mysql:latest	ip-172-31-	25-121.ec2.internal
Running	Running	53 seconds ago		

Similarly, multiple node constraints could be used to run replicas only on a manager node. Next, we update the mysql service to run on a specific manager node. First, promote one of the worker nodes to manager.

~ \$ docker node promote ip-172-31-2-177.ec2.internal Node ip-172-31-2-177.ec2.internal promoted to a manager in the swarm.

Subsequently, two manager nodes are listed as indicated by the Manager Status for two of the nodes.

~ \$ docker node ls ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS
81h6uvu8uq0emnovzkg6v7mzg e7vigin0luuo1kynjnl33v9pa	ip-172-31-2-177.ec2.internal ip-172-31-29-67.ec2.internal	Ready Ready	Active Active	Reachable
ptm7e0p346zwypos7wnpcm72d *	ip-172-31-25-121.ec2.internal	Ready	Active	Leader

Update the mysql service to add multiple node constraints to run replicas only on a specific manager node. Constraints are added using the --constraint-add option of the docker service update command.

```
~ $ docker service update \
> --constraint-add node.role==manager \
> --constraint-add node.hostname==ip-172-31-2-177.ec2.internal\
> mysql
mysql
```

Again, all service tasks are shut down and new tasks are started, all on the specified manager node that was promoted from the worker node.

```
~ $ docker service ps -f desired-state=running mysql
               NAME
                                           NODE
TD
                           IMAGE
DESIRED STATE
                CURRENT STATE
                                            ERROR
                                                     PORTS
eghm1or6yg5g
                mysql.1
                          mysql:latest
                                           ip-172-31-2-177.ec2.internal
                Running 28 seconds ago
Running
bhfngac5ssm7
                mysql.2 mysql:latest
                                           ip-172-31-2-177.ec2.internal
Running
                Running 22 seconds ago
ts3fgvq900os
                mysql.3
                          mysql:latest
                                           ip-172-31-2-177.ec2.internal
Running
                 Running 25 seconds ago
```

Adding Node Labels for Scheduling

Next, we discuss how node labels can be used to specify service placement constraints. Labels may be added to a node with the following command syntax, in which variables are <LABELKEY>, <LABELVALUE>, and <NODE>. The <NODE> is the node ID or hostname.

docker node update --label-add <LABELKEY>=<LABELVALUE> <NODE>

As an example, add the label db=mysql to the node with a hostname set to ip-172-31-25-121.ec2. internal, which is the leader node.

```
~ $ docker node update --label-add db=mysql ip-172-31-25-121.ec2.internal
ip-172-31-25-121.ec2.internal
```

A node label is added. On inspecting the node, the label is listed in the Labels field.

```
~ $ docker node inspect ip-172-31-25-121.ec2.internal
[
          "Spec": {
              "Labels": {
                  "db": "mysql"
              },
              "Role": "manager",
              "Availability": "active"
              },
]
```

Next, create a service that uses the node label to add a placement constraint. The --constraint option for the label must include the prefix node.labels.

```
~ $ docker service rm mysql
mysql
~ $ docker service create \
> --env MYSQL_ROOT_PASSWORD='mysql'\
> --replicas 3 \
> --constraint node.labels.db==mysql \
> --name mysql \
> mysql
2hhccmj9senseazbet11dekoa
150
```

The service is created. Listing the tasks lists all the tasks on the Leader manager node, which is what the node label constraint specified.

′\$ docker service ps -f desired-state=running mysql				
ID	NAME	IMAGE	NODE	
DESIRED STATE	CURRENT	STATE	ERROR	PORTS
g5jz9im3fufv	mysql.1	mysql:latest	ip-172-31-	25-121.ec2.internal
Running	Running	18 seconds ago		
bupr27bs57h1	mysql.2	<pre>mysql:latest</pre>	ip-172-31-	25-121.ec2.internal
Running	Running	18 seconds ago		
5bb2yf8aehqn	mysql.3	mysql:latest	ip-172-31-	25-121.ec2.internal
Running	Running	18 seconds ago		

The label added may be removed with the --label-rm option of the docker node update command in which the only the label key is specified.

docker node update --label-rm db ip-172-31-25-121.ec2.internal

Adding, Updating, and Removing Service Scheduling Constraints

In an earlier section, we discussed adding placement constraints when creating a service with docker service create. Placement constraints may be added/removed with the docker service update command using the --constraint-add and --constraint-rm options. To discuss an example of updating placement constraints, we create a mysql service with three replicas and no placement constraints to start with.

```
~ $ docker service rm mysql
mysql
~ $ docker service create \
> --env MYSQL_ROOT_PASSWORD='mysql'\
> --replicas 3 \
> --name mysql \
> mysql
az3cq6sxwrrk4mxkksdu21i25
```

A mysql service gets created with three replicas scheduled on the three nodes in the Swarm, using the spread policy.

Next, update the service with the docker service update command to add a constraint for the service replicas to run only on the manager nodes.

```
~ $ docker service update \
> --constraint-add node.role==manager \
> mysql
mysql
```

In a Swarm with two manager nodes, all the service tasks are shut down and new tasks are started only on the manager nodes.

~ \$ docker serv	\$ docker service ps mysql					
ID	NAME	IMAGE	NODE			
DESIRED STATE	CURRENT STAT	E	ERROR	PORTS		
pjwseruvy4rj	mysql.1	mysql:latest	ip-172-31	-2-177.ec2.internal		
Running	Running 4 se	conds ago				
s66g9stz9af5	_ mysql.1	mysql:latest	ip-172-31	-2-177.ec2.internal		
Shutdown	Shutdown 4 s	econds ago				
yqco9zd0vq79	mysql.2	mysql:latest	ip-172-31	-25-121.ec2.internal		
Running	Running 9 se	conds ago				
8muu6gbghhnd	_ mysql.2	mysql:latest	ip-172-31	-25-121.ec2.internal		
Shutdown	Shutdown 10	seconds ago				
8x7xlavcxdau	mysql.3	mysql:latest	ip-172-31	-25-121.ec2.internal		
Running	Running 7 se	conds ago				
qx95vwi2h547	_ mysql.3	mysql:latest	ip-172-31	-29-67.ec2.internal		
Shutdown	Shutdown 7 s	econds ago				

Scheduling constraints may be added and removed in the same docker service update command. As an example, remove the constraint for the node to be a manager and add a constraint for the node to be a worker.

```
~ $ docker service update \
> --constraint-rm node.role==manager \
> --constraint-add node.role==worker \
> mysql
mysql
```

Again. all the service tasks are shut down and new tasks are started only on the worker nodes.

~ \$ docker serv	/ice ps -f	desired-state=ru	unning mysql	
ID	NAME	IMAGE	NODE	
DESIRED STATE	CURRENT	STATE	ERROR	PORTS
6ppgmvw9lv75	mysql.1	mysql:latest	ip-172-31-	29-67.ec2.internal
Running	Running	9 seconds ago		
qmOloki65v9s	mysql.2	mysql:latest	ip-172-31-	29-67.ec2.internal
Running	Running	17 seconds ago		
ypl0tc1ft92o	mysql.3	mysql:latest	ip-172-31-	29-67.ec2.internal
Running	Running			

If the only scheduling constraint that specifies the node role as worker is removed, the spread scheduling strategy starts new tasks spread evenly across the Swarm. To demonstrate, remove the constraint for the node role to be a worker.

```
~ $ docker service update --constraint-rm node.role==worker mysql
mysql
```

Subsequently, new tasks are spread across the nodes in the Swarm.

```
~ $ docker service ps -f desired-state=running mysql
ID
                NAME
                           IMAGE
                                            NODE
DESIRED STATE
                 CURRENT STATE
                                            FRROR
                                                      PORTS
jpx4jjw6l9d5
                mysql.1
                           mysql:latest
                                            ip-172-31-29-67.ec2.internal
Running
                 Running 5 seconds ago
                           mysql:latest
                                            ip-172-31-25-121.ec2.internal
ngajiik1hugb
                mysql.2
                 Running 12 seconds ago
Running
                                            ip-172-31-2-177.ec2.internal
40eaujzlux88
                mysql.3
                           mysql:latest
Running
                 Running 8 seconds ago
```

Spread Scheduling and Global Services

A global service runs one task on every node in a Swarm. A global service cannot be scaled to create more/ fewer tasks. As a result, the spread scheduling policy concept does not apply to global services. However, node constraints may be applied to global services. As an example, we create a global service for the mysql database. Apply a placement constraint that the service should be available only on worker nodes.

```
~ $ docker service create \
> --mode global \
> --env MYSQL_ROOT_PASSWORD='mysql'\
> --constraint node.role==worker \
> --name mysql \
> mysql
jtzcwatp001q9r26n1uubd8me
```

The global service is created. Listing the service tasks for the tasks with desired state as running lists only the tasks on the worker nodes.

```
~ $ docker service ps -f desired-state=running mysql
ID NAME IMAGE NODE
DESIRED STATE CURRENT STATE ERROR PORTS
o5nskzpv27j9 mysql.e7vigin0luuo1kynjnl33v9pa mysql:latest ip-172-31-29-67.ec2.internal
Running Running 17 seconds ago
```

If created without the constraint to schedule on worker nodes only, a global service schedules one task on each node, as demonstrated by the following example.

```
~ $ docker service rm mysql
mysql
~ $ docker service create \
> --mode global \
> --env MYSQL_ROOT_PASSWORD='mysql'\
> --name mysql \
> mysql
mv9yzyyntdhzz41zssbutcsvw
```

~ \$ docker set	vice ps -f desired-state=runni	ng mysql.	
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE ERROR	PORTS	
mc87btddhmpl	<pre>mysql.e7vigin0luuo1kynjnl33v9p</pre>	a mysql:latest	ip-172-31-29-67.ec2.internal
Running	Running 19 seconds ago		
oOwfdq9sd8yt	mysql.ptm7e0p346zwypos7wnpcm72	d mysql:latest	ip-172-31-25-121.ec2.internal
Running	Running 19 seconds ago		
wt2q5k2dhqjt	mysql.81h6uvu8uq0emnovzkg6v7mz	g mysql:latest	ip-172-31-2-177.ec2.internal
Running	Running 19 seconds ago		

Summary

This chapter discussed the scheduling policy of spread used in the Docker Swarm mode, whereby service replicas are spread evenly across nodes in a Swarm based on node ranking; a higher node ranking gets a service replica placement priority. We also discussed the effect of limited node resource capacity and how to alleviate it by adding new nodes to the Swarm. We discussed placement constraints for scheduling new replicas. The spread scheduling policy is not relevant for global services as they create one service task on each node by default. However, scheduling constraints may be used with global services. In the next chapter we discuss rolling updates to Docker services.

CHAPTER 9

Rolling Updates

The Docker Swarm mode provisions services consisting of replicas that run across the nodes in the Swarm. A service definition is created when a service is first created/defined. A service definition is created with the docker service create command. That command provides several options, including those for adding placement constraints, container labels, service labels, DNS options, environment variables, resource reserves and limits, logging driver, mounts, number of replicas, restart condition and delay, update delay, failure action, max failure ratio, and parallelism, most of which were discussed in Chapter 4.

The Problem

Once a service definition has been created, it may be required to update some of the service options such as increase/decrease the number of replicas, add/remove placement constraints, update resource reserves and limits, add/remove mounts, add/remove environment variables, add/remove container and service labels, add/remove DNS options, and modify restart and update parameters. If a service is required to be shut down as a whole to update service definition options, an interruption of service is the result.

The Solution

Docker Swarm mode includes the provision for rolling updates. In a rolling update, the service is not shut down, but individual replicas/tasks in the service are shut down one at a time and new service replicas/ tasks based on the new service definition are started one at a time, as illustrated in Figure 9-1. As a result the service continues to be available during the rolling update. The service tasks that are served to a client could be from both old and new service definitions during a rolling update. As an example, if the rolling update performs an update to a more recent image tag, some of the tasks served to external clients during the rolling update could be from a mix of old image tag and new image tag.

CHAPTER 9 ROLLING UPDATES



Figure 9-1. Rolling update

Rolling update creates a new service definition and a new desired state for a service. Rolling update involves shutting down all service replicas and starting all new service replicas and does not apply to service replicas that have not yet been scheduled, due to lack of resources for example. Even updating just the number of replicas in a rolling update shuts down or fails all the old replicas and starts all new replicas.

The following sequence is used by the scheduler during a rolling update.

- 1. The first task is stopped.
- 2. An update for the stopped task is scheduled.
- 3. A Docker container for the updated task is started.
- 4. If the update to a task returns RUNNING, wait for the duration specified in --update-delay and start the update to the next task.

- 5. If during the update, a task returns FAILED, perform the --update-failure-action, which is to pause the update by default.
- 6. Restart a paused update with docker service update <SERVICE-ID>.
- 7. If an update failure is repeated, find the cause of the failure and reconfigure the service by supplying other options to the docker service update.

Setting the Environment

Create a Docker Swarm consisting of a manager node and two worker nodes using Docker for AWS, as discussed in Chapter 3. Obtain the public IP address of the manager instance from the EC2 console and then SSH login to the instance.

```
[root@localhost ~]# ssh -i "docker.pem" docker@54.84.133.157
Welcome to Docker!
```

List the Swarm nodes.

~ \$ docker node ls					
ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER	STATUS
81h6uvu8uq0emnovzkg6v7mzg	ip-172-31-2-177.ec2.internal	Ready	Active		
e7vigin0luuo1kynjnl33v9pa	ip-172-31-29-67.ec2.internal	Ready	Active		
ptm7e0p346zwypos7wnpcm72d *	ip-172-31-25-121.ec2.internal	Ready	Active	Leader	

Creating a Service with a Rolling Update Policy

A rolling update policy or update config consists of the service definition options discussed in Table 9-1.

Option	Description	Default Value
update-delay	Delay between updates (ns us ms s m h).	0 seconds
update-failure-action	Action on update failure. Value may be pause or continue.	pause
update-max-failure-ratio		
update-monitor	Duration after each task update to monitor for failure (ns us ms s m h).	0 seconds
update-parallelism	Maximum number of tasks updated simultaneously. A value of 0 updates all at once.	1

 Table 9-1.
 Rolling Update Options

To configure the rolling update policy at service deployment time, the options to be configured must be supplied when the service is created. As an example, create a service for MySQL database and specify the update policy options --update-delay and --update-parallelism.

```
~ $ docker service create \
> --env MYSQL_ROOT_PASSWORD='mysql'\
> --replicas 1 \
> --name mysql \
> --update-delay 10s \
> --update-parallelism 1 \
> mysql:5.6
wr0z48v1uquk1c40pa42vwrpn
```

The service is created. Listing the services may not list all replicas as running initially, as indicated by 0/1 in the REPLICAS column.

~ \$ docker serv	ice ls				
ID	NAME	MODE	REPLICAS	IMAGE	PORTS
wr0z48v1uguk	mysql	replicated	0/1	mysql:5.6	

Running the same command after a while should list all replicas as running, as indicated by 1/1 in REPLICAS column.

~ \$ docker set	rvice ls				
ID	NAME	MODE	REPLICAS	IMAGE	PORTS
wr0z48v1uguk	mysql	replicated	1/1	mysql:5.6	

The single service replica is scheduled on the manager node itself and the Docker container for the replica is started.

~ \$ docker service p	os mysql		
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
38dm9gm6cmvk	mysql.1	mysql:5.6	ip-172-31-25-121.ec2.internal
Running	Running 13 seconds a	ago	

Creating a service using rolling update options does not by itself demonstrate a rolling update. It only defines the UpdateConfig settings of the service. In the next section we perform a rolling update.

Rolling Update to Increase the Number of Replicas

A rolling update could be used to update the number of replicas with the --replicas option to the docker service update command. A rolling update updates the UpdateConfig policy applied when the service is first deployed. Next, we update the number of replicas for the mysql:5.6 image based service from the one replica created in the preceding section. Run the following command to update the service definition to five replicas from one replica. The --update-delay and --update-parallelism options modify the UpdateConfig of the service definition. The docker service update command outputs the service name if the update is successful.

```
~ $ docker service update \
> --replicas 5 \
> --update-delay 20s \
> --update-parallelism 1 \
> mysql
mysql
```

Subsequently, the services listing may list some of the replicas as not started yet in the output to the docker service 1s command. But, running the command again after a while should list all replicas as running.

~ \$ docker service	e ls				
ID	NAME	MODE	REPLICAS	IMAGE	PORTS
wr0z48v1uguk	mysql	replicated	5/5	mysql:5.6	

During the rolling update, all the running tasks are shut down and new tasks are started. The desired state of the mysql.1 task gets updated to shutdown and the current state is set to failed. A new task mysql.1 is started.

~ \$ docker service	ps mysql		
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
ydqj6vf9rsgw	mysql.1	mysql:5.6	ip-172-31-25-121.ec2.internal
Running	Running 26 seconds	ago	
38dm9gm6cmvk	_ mysql.1	mysql:5.6	ip-172-31-25-121.ec2.internal
Shutdown	Failed 31 seconds a	go "task: non-zei	ro exit (137)"
7bns96iu8ygz	mysql.2	mysql:5.6	ip-172-31-29-67.ec2.internal
Running	Running 32 seconds	ago	
62wfdbcv3cr4	mysql.3	mysql:5.6	ip-172-31-2-177.ec2.internal
Running	Running 33 seconds	ago	
ql66z5x0a2lf	mysql.4	mysql:5.6	ip-172-31-25-121.ec2.internal
Running	Running 14 seconds	ago	
3n3b1j7ey732	_ mysql.4	mysql:5.6	ip-172-31-25-121.ec2.internal
Shutdown	Failed 19 seconds a	go "task: non-zei	ro exit (137)"
bl1365y60vuu	mysql.5	mysql:5.6	ip-172-31-2-177.ec2.internal
Running	Running 33 second	s ago	

When scaling from one to five replicas, first a few new tasks are started and then the task running initially is shut down so that the service continues to be available during the rolling update. If the only task in the service were to be shut down first before starting any new tasks, the service wouldn't have any running tasks for a short while.

The desired state of running five replicas is not immediately reconciled during a rolling update. Fewer than five tasks could be running while the rolling update is in progress. Listing the running service tasks lists only three tasks as running.

~ \$ docker servi	ice ps -f des	ired-state=running	mysql	
ID	NAME	IMAGE		NODE
DESIRED STATE	CURRENT	STATE	ERROR	PORTS
ydqj6vf9rsgw	mysql.1	mysql:5.6		ip-172-31-25-121.ec2.internal
Running	Running	35 seconds ago		
7bns96iu8ygz	mysql.2	mysql:5.6		ip-172-31-29-67.ec2.internal
Running	Running	40 seconds ago		
ql66z5x0a2lf	mysql.4	mysql:5.6		ip-172-31-25-121.ec2.internal
Running	Running	22 seconds ago		

When the rolling update has completed, five tasks are running.

~ \$ docker servi	ce ps -f desired-stat	te=running mysql	
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
u8falo7q95cq	mysql.1	mysql:5.6	ip-172-31-25-121.ec2.internal
Running	Running 20 second	ds ago	
luabknwzwqoj	mysql.2	mysql:5.6	ip-172-31-29-67.ec2.internal
Running	Running 13 second	ds ago	
ce4l2qvtcanv	mysql.3	mysql:5.6	ip-172-31-2-177.ec2.internal
Running	Running 25 second	ds ago	
iw8vwsxq3tjz	mysql.4	mysql:5.6	ip-172-31-25-121.ec2.internal
Running	Running 6 seconds	s ago	
qfi5fionjt2v	mysql.5	mysql:5.6	ip-172-31-29-67.ec2.internal
Running	Running 25 second	ds ago	

Inspecting the service should list the updated number of replicas. The UpdateConfig is also listed with the docker service inspect command.

```
~ $ docker service inspect mysql
[
        "Spec": {
             "Name": "mysql",
. . .
            },
"Mode": {
                 "Replicated": {
                      "Replicas": 5
                 }
             },
             "UpdateConfig": {
                 "Parallelism": 1,
                 "Delay": 2000000000,
                 "FailureAction": "pause",
                 "Monitor": 500000000,
                 "MaxFailureRatio": 0,
                 "Order": "stop-first"
             },
             "RollbackConfig": {
                 "Parallelism":`1,
"FailureAction": "pause",
                 "Monitor": 500000000,
                 "MaxFailureRatio": 0,
                 "Order": "stop-first"
             },
• • •
1
```

Rolling Update to a Different Image Tag

A use case for a rolling update is to update to a newer image tag. As an example, perform a rolling update to update to Docker image mysql:latest from mysql:5.6 for the mysql service. Update parallelism is set to 2 to update two replicas at a time.

~ \$ docker service update --image mysql:latest --update-parallelism 2 mysql
mysql

The service rolling update gets started. Listing the service replicas lists mysql:5.6 image-based replicas as shutting down, as indicated by the shutdown desired state and mysql:latest image-based replicas as starting, as indicated by the running desired state.

~ \$ docker service	os mysql		
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
vqc6rhzw5uxz	mysql.1	mysql:latest	ip-172-31-2-177.ec2.internal
Ready	Ready 7 seconds ago		
80kswuu4d5gc	_ mysql.1	mysql:5.6	ip-172-31-2-177.ec2.internal
Shutdown	Running 7 seconds ag	go	
u8falo7q95cq	_ mysql.1	mysql:5.6	ip-172-31-25-121.ec2.internal
Shutdown	Failed 12 seconds ag	go "task: non-z	ero exit (1)"
ydqj6vf9rsgw	_ mysql.1	mysql:5.6	ip-172-31-25-121.ec2.internal
Shutdown	Failed 56 seconds ag	go "task: non-z	ero exit (1)"
38dm9gm6cmvk	_ mysql.1	mysql:5.6	ip-172-31-25-121.ec2.internal
Shutdown	Failed about a minut	te ago "task: non-z	ero exit (137)"
tvxjmahy08uh	mysql.2	mysql:5.6	ip-172-31-29-67.ec2.internal
Running	Running 2 seconds ag	go	
luabknwzwqoj	_ mysql.2	mysql:5.6	ip-172-31-29-67.ec2.internal
Shutdown	Failed 8 seconds ago	o "task: non-z	ero exit (137)"
7bns96iu8ygz	_ mysql.2	mysql:5.6	ip-172-31-29-67.ec2.internal
Shutdown	Failed 50 seconds	ago "task: nor	n-zero exit (137)"
u2ea4xq4yx6t	mysql.3	mysql:latest	ip-172-31-2-177.ec2.internal
Running	Running 4 seconds ag	go	
ce4l2qvtcanv	_ mysql.3	mysql:5.6	ip-172-31-2-177.ec2.internal
Shutdown	Shutdown 4 seconds a	ago	
62wfdbcv3cr4	_ mysql.3	mysql:5.6	ip-172-31-2-177.ec2.internal
Shutdown	Failed about a minut	te ago "task: non-z	ero exit (1)"
iw8vwsxq3tjz	mysql.4	mysql:5.6	ip-172-31-25-121.ec2.internal
Running	Running 37 seconds a	ago	
ql66z5x0a2lf	_ mysql.4	mysql:5.6	ip-172-31-25-121.ec2.internal
Shutdown	Failed 43 seconds ag	go "task: non-z	ero exit (137)"
3n3b1j7ey732	_ mysql.4	mysql:5.6	ip-172-31-25-121.ec2.internal
Shutdown	Failed about a minut	te ago "task: non-z	ero exit (137)"
f5vcf9mgluqe	mysql.5	mysql:5.6	ip-172-31-29-67.ec2.internal
Running	Running 14 seconds a	ago	
qfi5fionjt2v	_ mysql.5	mysql:5.6	ip-172-31-29-67.ec2.internal
Shutdown	Failed 19 seconds ag	go "task: non-z	ero exit (1)"
bl1365y60vuu	_ mysql.5	mysql:5.6	ip-172-31-2-177.ec2.internal
Shutdown	Failed about a minut	te ago "task: non-z	ero exit (1)"

While the rolling update is in progress, some of the running tasks could be based on the previous service specification (mysql:5.6), while others are based on the new service specification (mysql:latest).

~ \$ docker servi	.ce ps -f desired-st	ate=running mysql	
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
vqc6rhzw5uxz	mysql.1	<pre>mysql:latest</pre>	ip-172-31-2-177.ec2.internal
Running	Running 4 secon	ds ago	
tvxjmahy08uh	mysql.2	mysql:5.6	ip-172-31-29-67.ec2.internal
Running	Running 11 seco	nds ago	
u2ea4xq4yx6t	mysql.3	mysql:latest	ip-172-31-2-177.ec2.internal
Running	Running 13 seco	nds ago	
iw8vwsxq3tjz	mysql.4	mysql:5.6	ip-172-31-25-121.ec2.internal
Running	Running 46 seco	nds ago	
f5vcf9mgluqe	mysql.5	mysql:5.6	ip-172-31-29-67.ec2.internal
Running	Running 23 seco	nds ago	

When the rolling update has completed, all running tasks are based on the new service specification.

~ \$ docker service	<pre>ps -f desired-state=</pre>	=running mysql		
ID	NAME	IMAGE		NODE
DESIRED STATE	CURRENT STATE		ERROR	PORTS
vqc6rhzw5uxz	mysql.1	mysql:latest		ip-172-31-2-177.ec2.internal
Running	Running 45 seconds	ago		
53choz0dd967	mysql.2	mysql:latest		ip-172-31-29-67.ec2.internal
Running	Running less than a	a second ago		
u2ea4xq4yx6t	mysql.3	mysql:latest		ip-172-31-2-177.ec2.internal
Running	Running 53 seconds	ago		
tyo6v0yen7ev	mysql.4	mysql:latest		ip-172-31-29-67.ec2.internal
Running	Running 21 seconds	ago		
upt212osx7au	mysql.5	mysql:latest		ip-172-31-29-67.ec2.internal
Running	Running 25 seconds	ago		

Rolling Update to Add and Remove Environment Variables

The Docker image mysql requires one mandatory environment variable MYSQL_ROOT_PASSWORD for the root password and supports some other environment variables that may also be specified. The other environment variables are MYSQL_DATABASE for the MySQL database, MYSQL_USER for the MYSQL user, MYSQL_PASSWORD for the MySQL password, and MYSQL_ALLOW_EMPTY_PASSWORD for whether to allow the root password to be empty. The MYSQL_ROOT_PASSWORD was already set when the mysql service was created. Using the --env-add option to the docker service update command, we can add the other environment variables.

```
~ $ docker service update --env-add MYSQL_DATABASE='mysqldb' --env-add MYSQL_USER='mysql'
--env-add MYSQL_PASSWORD='mysql' --env-add MYSQL_ALLOW_EMPTY_PASSWORD='no' --update-
parallelism 1 mysql
mysql
```

An output of mysql implies the command ran successfully.

The rolling update status is found with the docker service inspect command, which in addition to listing the env variables added in the Env JSON object, lists the UpdateStatus. The State of the update status is updating and the message is "update in progress".

```
~ $ docker service inspect mysql
ſ
    {...
       "Spec": {
            "Name": "mysql",
                 "ContainerSpec": {
. . .
                     "Env": [
                         "MYSOL ROOT PASSWORD=mysql",
                         "MYSQL_DATABASE=mysqldb",
                         "MYSQL_USER=mysql",
                         "MYSOL PASSWORD=mysql",
                         "MYSQL ALLOW EMPTY PASSWORD=no"
                     ],
           },
. . .
        "UpdateStatus": {
            "State": "updating",
            "StartedAt": "2017-07-25T19:18:11.44139778Z",
            "Message": "update in progress"
        }
    }
]
```

When the update has completed, the UpdateStatus state becomes "completed" and the Message becomes "update completed".

```
~ $ docker service inspect mysql
[
... },
    "UpdateStatus": {
        "State": "completed",
        "StartedAt": "2017-07-25T19:18:11.44139778Z",
        "CompletedAt": "2017-07-25T19:20:37.912993431Z",
        "Message": "update completed"
        }
    }
}
```

As indicated by the StartedAt and CompletedAt timestamp, the rolling update takes about two minutes. Listing only tasks with desired state of running indicates that one task has been running for 21 seconds and another task has been running for two minutes.

~ \$ docker service	ps -f desired-state=:	running mysql	
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
3zhf94kklu6r	mysql.1	mysql:latest	ip-172-31-29-67.ec2.internal
Running	Running 21 seconds a	ago	
ta16ch5kjlr9	mysql.2	mysql:latest	ip-172-31-29-67.ec2.internal
Running	Running 2 minutes ag	go	
fc7uxvwvcmk3	mysql.3	mysql:latest	ip-172-31-2-177.ec2.internal
Running	Running about a min	ute ago	

jir97p344kol	mysql.4	mysql:latest	ip-172-31-29-67.ec2.internal
Running	Running about	a minute ago	
5rly53mcc8yq	mysql.5	mysql:latest	ip-172-31-2-177.ec2.internal
Running	Running 45 se	conds ago	

The environment variables added may be removed with another docker service update command and the --env-rm options for each environment variable to remove. Only the env variable name is to be specified in --env-rm, not the env value.

```
~ $ docker service update --env-rm MYSQL_DATABASE --env-rm MYSQL_USER --env-rm
MYSQL_PASSWORD --env-rm MYSQL_ALLOW_EMPTY_PASSWORD mysql
mysql
```

Another rolling update gets performed. All service tasks get shut down and new service tasks based on the new service specification are started. The service definition lists only the mandatory environment variable MYSQL_ROOT_PASSWORD.

Rolling Update to Set CPU and Memory Limits and Reserve

A rolling update may be used to set new resource limits and reserves.

```
~ $ docker service update --reserve-cpu 1 --limit-cpu 2 --reserve-memory 256mb -
-limit-memory 512mb mysql
mysql
```

New resource limits and reserves are configured, as listed in the service specification. The PreviousSpec indicates that no Resources Limits and Reservations are configured to start with.

```
~ $ docker service inspect mysql
[
...
"Spec": {
    "Name": "mysql",
...
"ContainerSpec": {
...
},
```

```
"Resources": {
                     "Limits": {
                         "NanoCPUs": 200000000,
                         "MemoryBytes": 536870912
                     },
                     "Reservations": {
                         "NanoCPUs": 100000000,
                         "MemoryBytes": 268435456
                     }
                },
           },
. . .
        "PreviousSpec": {
. . .
            "Name": "mysql",
                 "Resources": {
                     "Limits": {},
                     "Reservations": {}
                },
        "UpdateStatus": {
            "State": "updating",
            "StartedAt": "2017-07-25T19:23:44.004458295Z",
            "Message": "update in progress"
        }
    }
]
```

Setting new resource limits and reserves are subject to node capacity limits. If requested resources exceed the node capacity the rolling update may continue to run and not get completed, with some tasks in the pending current state.

~ \$ docker servi	.ce ps -f desired-sta	te=running mysql	
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
5u7zifw15n7t	mysql.1	mysql:latest	ip-172-31-25-121.ec2.internal
Running	Running about an	hour ago	
2kgsb16c8m8u	mysql.2	<pre>mysql:latest</pre>	
Running	Pending about an	hour ago	
muO8iu9qzqlh	mysql.3	mysql:latest	ip-172-31-29-67.ec2.internal
Running	Running about an	hour ago	
aakxr8dw5s15	mysql.4	mysql:latest	ip-172-31-2-177.ec2.internal
Running	Running about an	hour ago	
z6045639f20p	mysql.5	mysql:latest	ip-172-31-25-121.ec2.internal
Running	Running about an	hour ago	

If some tasks are pending, adding resources to the Swarm could make the pending tasks run. We can update the CloudFormation stack to increase the number of worker nodes from 2 to 3, as shown in Figure 9-2.

CHAPTER 9 ROLLING UPDATES

CloudFormatio	n v	Stacks > Stack Deta	ill > Update Stack			
Update Dock	er si	tack				
Select Template		Specify Details				
Specify Details Options Review		Specify parameter values. You can use or change the default parameter values, which are defined in the AWS CloudFormation template. Learn more.				
		Stack name	Docker			
		Parameters				
	Swarm Size					
		Number of Swarm managers?	1	Number of Swarm manager nodes (1, 3,	5)	
	ß	Number of Swarm worker nodes?	3	Number of worker nodes in the Swarm (0	D-1000).	

Figure 9-2. Increasing the number of worker nodes in the Swarm

Subsequently, the Swarm should list four nodes.

~ \$ docker node ls				
ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS
81h6uvu8uq0emnovzkg6v7mzg	ip-172-31-2-177.ec2.internal	Ready	Active	
e7vigin0luuo1kynjnl33v9pa	ip-172-31-29-67.ec2.internal	Ready	Active	
ptm7e0p346zwypos7wnpcm72d *	ip-172-31-25-121.ec2.internal	Ready	Active	Leader
t4d0aq9w2a6avjx94zgkwc557	ip-172-31-42-198.ec2.internal	Ready	Active	

With increased resources in the Swarm, the pending tasks also start to run.

~ \$ docker service	e ps -f desired-sta	te=running mysql		
ID	NAME	IMAGE	NODE	
DESIRED STATE	CURRENT STATE	ERROR	PORTS	
5u7zifw15n7t	mysql.1	mysql:latest	ip-172-31-25-121.ec2.internal	
Running	Running about an	hour ago		
2kgsb16c8m8u	mysql.2	mysql:latest	ip-172-31-2-177.ec2.internal	
Running	Running 7 minutes ago			
muO8iu9qzqlh	mysql.3	mysql:latest	ip-172-31-29-67.ec2.internal	
Running	Running about an	hour ago		
i5j2drlcm75f	mysql.4	<pre>mysql:latest</pre>	ip-172-31-42-198.ec2.internal	
Running	Running 4 seconds ago			
z6045639f20p	mysql.5	mysql:latest	ip-172-31-25-121.ec2.internal	
Running	Running about an	hour ago		
Rolling Update to a Different Image

Rolling update may also be used to update to a completely different Docker image. As an example, perform a rolling update to the mysql service to use Docker image postgres instead of the mysql image it is using. Other options such as --update-parallelism may also be set.

~ \$ docker service update --image postgres --update-parallelism 1 mysql
mysql

The mysql:latest image-based tasks start to get shut down and postgres image-based replacement tasks begin to get started one task at a time. The rolling update does not get completed immediately and listing the service tasks with the desired state as running lists some tasks based on the postgres:latest image, while other tasks are still using the mysql:latest image.

~ \$ docker servi	ce ps -f desired-sta	te=running mysql	
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
9tzm5pa6pcyx	mysql.1	postgres:latest	ip-172-31-2-177.ec2.internal
Running	Running 39 secon	ds ago	
xj23fu5svv9d	mysql.2	<pre>postgres:latest</pre>	ip-172-31-42-198.ec2.internal
Running	Running about a	minute ago	
mu08iu9qzqlh	mysql.3	mysql:latest	ip-172-31-29-67.ec2.internal
Running	Running about an	hour ago	
skzxi33c606o	mysql.4	postgres:latest	ip-172-31-2-177.ec2.internal
Running	Running 13 secon	ds ago	
z6045639f20p	mysql.5	<pre>mysql:latest</pre>	ip-172-31-25-121.ec2.internal
Running	Running about an	hour ago	

One replica at a time, the mysql image-based replicas are shut down and postgres image-based replicas are started. After about two minutes, all tasks have updated to the postgres:latest image.

<pre>-f desired-state=r</pre>	unning mysql	
ME	IMAGE	NODE
RRENT STATE	ERROR	PORTS
sql.1	postgres:latest	ip-172-31-2-177.ec2.internal
nning about a minu [.]	te ago	
sql.2	postgres:latest	ip-172-31-42-198.ec2.internal
nning about a minu [.]	te ago	
sql.3	postgres:latest	ip-172-31-42-198.ec2.internal
nning 35 seconds a	go	
sql.4	postgres:latest	ip-172-31-2-177.ec2.internal
nning 59 seconds a	go	
sql.5	postgres:latest	ip-172-31-25-121.ec2.internal
nning 8 seconds ag	0	
	-+ desired-state=r ME RRENT STATE sql.1 nning about a minu sql.2 nning about a minu sql.3 sql.3 sql.4 nning 59 seconds a sql.5 nning 8 seconds ag	-+ desired-state=running mysqlMEIMAGERRENT STATEERRORsql.1postgres:latestnning about a minute agosql.2postgres:latestnning about a minute agosql.3postgres:latestnning 35 seconds agosql.4postgres:latestnning 59 seconds agosql.5postgres:latestnning 8 seconds ago

The service name continues to be the same and the replica names also include the mysql prefix. The mysql service definition ContainerSpec lists the image as postgres. Updating the image to postgres does not imply that all other service definition settings are updated for the new image. The postgres image does not use the MYSQL_ROOT_PASSWORD, but the environment variable continues to be in the service specification.

```
~ $ docker service inspect mysql
ſ
        "Spec": {
            "Name": "mysql",
                "ContainerSpec": {
                    "Image": "postgres:latest@sha256:e92fe21f695d27be7050284229a1c8c63ac10d8
                    8cba58d779c243566e125aa34",
                    "Env": [
                         "MYSOL ROOT PASSWORD=mvsal"
                    ],
        "PreviousSpec": {
            "Name": "mysql",
                "ContainerSpec": {
                     "Image": "mysql:latest@sha256:75c563c474f1adc149978011fedfe2e6670483d133
                    b22b07ee32789b626f8de3",
                    "Env": [
                         "MYSOL ROOT PASSWORD=mysql"
           },
. . .
        "UpdateStatus": {
            "State": "completed",
            "StartedAt": "2017-07-25T20:39:45.230997671Z",
            "CompletedAt": "2017-07-25T20:42:04.186537673Z",
            "Message": "update completed"
        }
    }
]
```

 $The {\tt MYSQL_ROOT_PASSWORD}\ environment\ variable\ may\ be\ removed\ with\ another\ update\ command.$

```
~ $ docker service update --env-rm MYSQL_ROOT_PASSWORD mysql
mysql
```

 $Subsequently, the \ {\tt ContainerSpec}\ does\ not\ include\ the\ {\tt MYSQL_ROOT_PASSWORD\ environment\ variable}.$

```
~ $ docker service inspect mysql
[
. . .
        "Spec": {
            "Name": "mysql",
            . . .
                "ContainerSpec": {
                     "Image": "postgres:latest@sha256:e92fe21f695d27be7050284229a1c8c63ac10d8
                     8cba58d779c243566e125aa34",
                     "StopGracePeriod": 1000000000,
                     "DNSConfig": {}
                },
           },
. . .
        "PreviousSpec": {
                "ContainerSpec": {
                     "Image": "postgres:latest@sha256:e92fe21f695d27be7050284229a1c8c63ac10d8
                     8cba58d779c243566e125aa34",
```

A rolling update to remove an environment variable involves shutting down all service tasks and starting all new tasks. The update takes about two minutes to complete.

Listing the running tasks indicates that tasks have only been running two minutes at the maximum.

~ \$ docker service	ps -f des	ired-state=r	unning mysql		
ID	NAME		IMAGE		NODE
DESIRED STATE	CURRENT S	STATE	ERROR	PORTS	5
menpo2zgit5u	mysql.1		postgres:latest	I.	ip-172-31-2-177.ec2.internal
Running	Running a	about a minu	te ago		
adnid3t69sue	mysql.2		postgres:latest		ip-172-31-25-121.ec2.internal
Running	Running a	about a minu	te ago		
we92apfuivil	mysql.3		postgres:latest		ip-172-31-42-198.ec2.internal
Running	Running 4	46 seconds a	go		
ed7vh4ozefm5	mysql.4		postgres:latest	2	ip-172-31-29-67.ec2.internal
Running	Running 2	2 minutes ag	0		
i2x2377ad7u0	mysql.5		postgres:latest	Ē.	ip-172-31-25-121.ec2.internal
Running	Running a	about a minu	te ago		

By removing the env variable MYSQL_ROOT_PASSWORD the mysql service gets updated to use Docker image postgres. The service name itself cannot be updated. The service may be updated back to the mysql image and the mandatory environment variable MYSQL_ROOT_PASSWORD added with another rolling update.

~ \$ docker service update --image mysql --env-add MYSQL_ROOT_PASSWORD='mysql' mysql
mysql

Again, listing the replicas with a desired state as running lists the postgres image-based replicas being replaced by mysql image-based replicas. One replica at a time, the postgres image-based replicas are replaced by mysql image-based replicas.

~ \$ docker service	ns _f desired_st	tate-running mysal	
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
menpo2zgit5u	mysql.1	<pre>postgres:latest</pre>	ip-172-31-2-177.ec2.internal
Running	Running 2 minut	tes ago	
adnid3t69sue	mysql.2	postgres:latest	ip-172-31-25-121.ec2.internal
Running	Running 2 minut	tes ago	
we92apfuivil	mysql.3	postgres:latest	ip-172-31-42-198.ec2.internal
Running	Running about a	a minute ago	
pjvj50j822xr	mysql.4	mysql:latest	ip-172-31-29-67.ec2.internal
Running	Running 12 seco	onds ago	
i2x2377ad7u0	mysql.5	postgres:latest	ip-172-31-25-121.ec2.internal
Running	Running 2 minut	tes ago	

Within a minute or two, all the postgres image replicas are replaced by mysql image-based replicas.

~ \$ docker servic	e ps -f desired-sta	te=running mysql	
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
sobd90v7gbmz	mysql.1	mysql:latest	ip-172-31-25-121.ec2.internal
Running	Running about a	minute ago	
st5t7y8rdgg1	mysql.2	mysql:latest	ip-172-31-29-67.ec2.internal
Running	Running 57 secon	ds ago	
upekevrlbmgo	mysql.3	<pre>mysql:latest</pre>	ip-172-31-42-198.ec2.internal
Running	Running about a	minute ago	
pjvj50j822xr	mysql.4	mysql:latest	ip-172-31-29-67.ec2.internal
Running	Running 2 minute	s ago	
nmrmdug87cy0	mysql.5	<pre>mysql:latest</pre>	ip-172-31-2-177.ec2.internal
Running	Running 2 minute	s ago	

The service specification is updated to the mysql image and the mandatory environment variable MYSQL ROOT PASSWORD is added. When the update has completed, the UpdateStatus State becomes completed.

```
"Image": "postgres:latest@sha256:e92fe21f695d27be7050284229a1c8c63ac10d8
8cba58d779c243566e125aa34",
... },
"UpdateStatus": {
    "State": "completed",
    "StartedAt": "2017-07-25T20:45:54.104241339Z",
    "CompletedAt": "2017-07-25T20:47:47.996420791Z",
    "Message": "update completed"
    }
}
```

Rolling Restart

Docker 1.13 added a new option to perform a rolling restart even when no update is required based on the update options. As an example starting with the mysql service with update config as --update-parallelism 1 and --update-delay 20s, the following update command won't perform any rolling update, as no changes are being made to the service.

```
\sim $ docker service update --update-parallelism 1 --update-delay 20s mysql mysql
```

To force a rolling restart, include the --force option.

. .

```
\sim $ docker service update --force --update-parallelism 1 --update-delay 20s mysql mysql
```

Service tasks begin to get shut down and new service tasks are started even though no update is made to the service specification. Some tasks are listed as having started a few seconds ago.

~ \$ docker servic	e ps -f desired-stat	e=running mysql	
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERRO	R PORTS
sobd90v7gbmz	mysql.1	mysql:latest	ip-172-31-25-121.ec2.internal
Running	Running 3 minutes	ago	
trye9chir91l	mysql.2	mysql:latest	ip-172-31-25-121.ec2.internal
Running	Running 23 second	ls ago	
uu7sfp147xnu	mysql.3	mysql:latest	ip-172-31-42-198.ec2.internal
Running	Running less than	a second ago	
pjvj50j822xr	mysql.4	mysql:latest	ip-172-31-29-67.ec2.internal
Running	Running 4 minutes	ago	
nmrmdug87cy0	mysql.5	<pre>mysql:latest</pre>	ip-172-31-2-177.ec2.internal
Running	Running 3 minutes	ago	

A rolling restart could take 1-2 minutes to complete.

```
~ $ docker service inspect mysql
[
....
},
```

```
CHAPTER 9 ■ ROLLING UPDATES

"UpdateStatus": {

    "State": "completed",

    "StartedAt": "2017-07-25T20:49:34.716535081Z",

    "CompletedAt": "2017-07-25T20:51:36.880045931Z",

    "Message": "update completed"

    }

}
```

After the rolling restart has completed, the service has all new service tasks as shown.

~ \$ docker service	e ps -f desired-stat	te=running mysql	
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
z2n2qcgfsbke	mysql.1	<pre>mysql:latest</pre>	ip-172-31-29-67.ec2.internal
Running	Running 6 seconds	s ago	
trye9chir91l	mysql.2	mysql:latest	ip-172-31-25-121.ec2.internal
Running	Running about a r	ninute ago	
uu7sfp147xnu	mysql.3	mysql:latest	ip-172-31-42-198.ec2.internal
Running	Running about a r	ninute ago	
1aovurxkteq1	mysql.4	mysql:latest	ip-172-31-29-67.ec2.internal
Running	Running 29 second	ds ago	
r0lslq6jibvp	mysql.5	<pre>mysql:latest</pre>	ip-172-31-2-177.ec2.internal
Running	Running 52 second	ds ago	

Rolling Update to Add and Remove Mounts

Rolling update can also be used to add and remove mounts. As an example, we add a mount of type volume with the source volume specified with src and the destination directory specified with dst.

```
~ $ docker service update \
> --mount-add type=volume,src=mysql-scripts,dst=/etc/mysql/scripts \
> mysql
mysql
```

A mount is added to the service and is listed in the service definition. Adding a mount involves shutting down all service tasks and starting new tasks. The rolling update could take 1-2 minutes.

```
"UpdateStatus": {
    "State": "completed",
    "StartedAt": "2017-07-25T20:51:55.205456644Z",
    "CompletedAt": "2017-07-25T20:53:56.451313826Z",
    "Message": "update completed"
    }
}
```

The mount added may be removed with the --mount-rm option of the docker service update command and by supplying only the mount destination directory as an argument.

```
~ $ docker service update \
>         --mount-rm /etc/mysql/scripts \
>        mysql
mysql
```

Another rolling update is performed and the mount is removed. It does not get listed in the service definition. The PreviousSpec lists the mount. The UpdateStatus indicates the status of the rolling update.

```
~ $ docker service inspect mysql
[
        "Spec": {
            "Name": "mysql",
                 "ContainerSpec": {
. . .
        "PreviousSpec": {
            "Name": "mysql",
. . .
                     "Mounts": [
                         {
                             "Type": "volume",
                             "Source": "mysql-scripts",
                             "Target": "/etc/mysql/scripts"
        "UpdateStatus": {
            "State": "completed",
            "StartedAt": "2017-07-25T20:55:56.30844324Z",
            "CompletedAt": "2017-07-25T20:57:58.489349432Z",
            "Message": "update completed"
        }
    }
1
```

Rolling Update Failure Action

The --update-failure-action option of the docker service create and docker service update commands specifies the follow-up action to take if the update to a task fails and returns FAILED. We set the UpdateConfig for the mysql service to include a --update-failure-action of pause (the default). The other option setting is continue, which does not pause a rolling update but continues with the update of the next task. To demonstrate a update failure action, specify a Docker image that does not exist, such as mysql:5.9.

```
CHAPTER 9 ROLLING UPDATES
~ $ docker service update \
    --replicas 10 \
>
    --image mysql:5.9 \
>
>
    --update-delay 10s \
>
    --update-failure-action pause \
> mysql
image mysql:5.9 could not be accessed on a registry to record
its digest. Each node will access mysql:5.9 independently,
possibly leading to different nodes running different
versions of the image.
mysql
```

The rolling update is still started and the update status indicates that the update is paused. The update status message indicates "update paused due to failure or early termination of task".

```
~ $ docker service inspect mysql
[
        "Spec": {
            "Name": "mysql",
            },
            "UpdateConfig": {
                "Parallelism": 1,
                "Delay": 1000000000,
                "FailureAction": "pause",
                "Monitor": 500000000,
                "MaxFailureRatio": 0,
                "Order": "stop-first"
            },
            "RollbackConfig": {
                "Parallelism": 1,
                "FailureAction": "pause",
                "Monitor": 500000000,
                "MaxFailureRatio": 0,
                "Order": "stop-first"
            },
           },
. . .
        "UpdateStatus": {
            "State": "paused",
            "StartedAt": "2017-07-25T20:58:51.695333064Z",
            "Message": "update paused due to failure or early termination of task
            s1p1n0x3k67uwpoj7qxg13747"
        }
    }
]
```

Two options are available if a rolling update is paused due to update to a task having failed.

- Restart a paused update using docker service update <SERVICE-ID>.
- If an update failure is repeated, find the cause of the failure and reconfigure the service by supplying other options to the docker service update <SERVICE-ID> command.

Roll Back to Previous Specification

Docker 1.13 Swarm mode added the feature to roll back to the previous service definition. As an example, perform a rolling update to update the image of the mysql service to postgres. The mysql-based replicas begin to be shut down and postgres-based replicas are started. At any time during the rolling update from the mysql image to the postgres image or after the update to the postgres image has completed, if it is ascertained that the rolling update should not have been started or performed, the rolling update may be rolled back with the following command. To demonstrate a rollback, we first start a mysql service.

```
~ $ docker service rm mysql
mysql
~ $ docker service create \
> --env MYSQL_ROOT_PASSWORD='mysql'\
> --replicas 5 \
> --name mysql \
> --update-delay 10s \
> --update-parallelism 1 \
> mysql:5.6
xkmrhnkOa444zambp9yh1mk9h
```

We start a rolling update to the postgres image from the mysql image.

```
~ $ docker service update --image postgres mysql
mysql
```

Subsequently, some of the tasks are based on the postgres image and some on the mysql image.

ps mysql		
NAME	IMAGE	NODE
CURRENT STATE	ERROR	PORTS
mysql.1	mysql:5.6	ip-172-31-25-121.ec2.internal
Running 58 seconds a	ago	
mysql.2	postgres:latest	ip-172-31-2-177.ec2.internal
Ready 2 seconds ago		
_ mysql.2	mysql:5.6	ip-172-31-2-177.ec2.internal
Running 2 seconds ag	go	
mysql.3	mysql:5.6	ip-172-31-42-198.ec2.internal
Running 22 seconds a	ago	
mysql.4	<pre>postgres:latest</pre>	ip-172-31-29-67.ec2.internal
Running 12 seconds a	ago	
_ mysql.4	mysql:5.6	ip-172-31-29-67.ec2.internal
Shutdown 13 seconds	ago	
mysql.5	mysql:5.6	ip-172-31-42-198.ec2.internal
Running 22 seconds a	ago	
	os mysql NAME CURRENT STATE mysql.1 Running 58 seconds ago _ mysql.2 Ready 2 seconds ago _ mysql.2 Running 2 seconds ag mysql.3 Running 22 seconds a mysql.4 Running 12 seconds a _ mysql.4 Shutdown 13 seconds mysql.5 Running 22 seconds a	ps mysqlNAMEIMAGECURRENT STATEERRORmysql.1mysql:5.6Running 58 seconds agomysql.2postgres:latestReady 2 seconds ago_mysql.2mysql:5.6Running 2 seconds agomysql.3mysql:5.6Running 22 seconds agomysql.4postgres:latestRunning 12 seconds ago_mysql.4mysql:5.6Shutdown 13 seconds agomysql.5mysql:5.6Running 22 seconds ago

Start a rollback to revert to the mysql image.

```
~ $ docker service update --rollback mysql
mysql
```

The postgres image-based tasks start to get shut down and the mysql image-based tasks are started.

~ \$ docker service	e ps mysql		
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
mnm5pg9ha61u	mysql.1	mysql:5.6	ip-172-31-25-121.ec2.internal
Running	Running about a	minute ago	
gyqgtoc4ix3y	mysql.2	mysql:5.6	ip-172-31-2-177.ec2.internal
Running	Running 14 secon	ds ago	
9yOfzn4sgivO	_ mysql.2	postgres:latest	ip-172-31-2-177.ec2.internal
Shutdown	Shutdown 15 seco	nds ago	
ewl7zxwi07gc	_ mysql.2	mysql:5.6	ip-172-31-2-177.ec2.internal
Shutdown	Shutdown 23 seco	nds ago	
l3ock28cmtzx	mysql.3	mysql:5.6	ip-172-31-42-198.ec2.internal
Running	Running 46 secon	ds ago	
ecvh8fd5308k	mysql.4	mysql:5.6	ip-172-31-29-67.ec2.internal
Running	Running 16 secon	ds ago	
1vqs3lcqvbt5	_ mysql.4	postgres:latest	ip-172-31-29-67.ec2.internal
Shutdown	Shutdown 16 seco	nds ago	
wu11jjbszesy	_ mysql.4	mysql:5.6	ip-172-31-29-67.ec2.internal
Shutdown	Shutdown 37 seco	nds ago	
m27d3gz4g6dy	mysql.5	mysql:5.6	ip-172-31-25-121.ec2.internal
Running	Running 1 second	ago	
g3tr6z9l5vzx	_ mysql.5	mysql:5.6	ip-172-31-42-198.ec2.internal
Shutdown	Failed 6 seconds	ago "task: no	on-zero exit (1)"

The rolling update from mysql to postgres is rolled back. When the rollback has completed, all replicas are mysql image-based, which is the desired state of the service to start with.

ps -f desired-state=	running mysql	
NAME	IMAGE	NODE
CURRENT STATE	ERROR	PORTS
mysql.1	mysql:5.6	ip-172-31-25-121.ec2.internal
Running 30 seconds	ago	
mysql.2	mysql:5.6	ip-172-31-2-177.ec2.internal
Running 56 seconds	ago	
mysql.3	mysql:5.6	ip-172-31-42-198.ec2.internal
Running about a min	ute ago	
mysql.4	mysql:5.6	ip-172-31-29-67.ec2.internal
Running 58 seconds	ago	
	<pre>ps -f desired-state= NAME CURRENT STATE mysql.1 Running 30 seconds mysql.2 Running 56 seconds mysql.3 Running about a min mysql.4 Running 58 seconds</pre>	ps -f desired-state=running mysql NAME IMAGE CURRENT STATE ERROR mysql.1 mysql:5.6 Running 30 seconds ago mysql.2 mysql:5.6 Running 56 seconds ago mysql.3 mysql:5.6 Running about a minute ago mysql.4 mysql:5.6 Running 58 seconds ago

Rolling Update on a Global Service

A rolling update may also be performed on a global service. To demonstrate, we create a global service for the mysql:latest image.

```
~ $ docker service rm mysql
mysql
~ $ docker service create \
> --mode global \
> --env MYSQL_ROOT_PASSWORD='mysql'\
```

```
> --name mysql \
> mysql
7nokncnti3izud08gfdovwxwa
```

```
Start a rolling update to Docker image mysql:5.6. ~ $ docker service update \
>    --image mysql:5.6 \
>    --update-delay 10s \
>    mysql
mysql
```

The service is updated. The Spec>ContainerSpec>Image is updated to mysql:5.6 from the PreviousSpec> ContainerSpec>Image of mysql:latest.

```
~ $ docker service inspect mysql
[
        "Spec": {
            "Name": "mysql",
                "ContainerSpec": {
                    "Image": "mysql:5.6@sha256:6ad5bd392c9190fa92e65fd21f6debc8b2a76fc54f139
                    49f9b5bc6a0096a5285",
            },
        "PreviousSpec": {
            "Name": "mysql",
                "ContainerSpec": {
                    "Image": "mysql:latest@sha256:75c563c474f1adc149978011fedfe2e6670483d133
                    b22b07ee32789b626f8de3",
        "UpdateStatus": {
            "State": "completed",
            "StartedAt": "2017-07-25T21:06:46.973666693Z",
            "CompletedAt": "2017-07-25T21:07:46.656023733Z",
            "Message": "update completed"
        }
    }
1
```

Within a minute, all the new service tasks based on mysql:5.6 are started.

~ \$ docker servi	<pre>ce ps -f desired-state=running</pre>	mysql	
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR PO	RTS
ybf4xpofte8l	mysql.81h6uvu8uq0emnovzkg6v7m	zg mysql:5.6	ip-172-31-2-177.ec2.internal
Running	Running 46 seconds ago		
7nq99jeil9n0	mysql.t4d0aq9w2a6avjx94zgkwc55	7 mysql:5.6	ip-172-31-42-198.ec2.internal
Running	Running about a minute ago		
wcng24mq7e8m	mysql.e7vigin0luuo1kynjnl33v9	pa mysql:5.6	ip-172-31-29-67.ec2.internal
Running	Running about a minute ago		
q14t2pyhra3w	mysql.ptm7e0p346zwypos7wnpcm7	2d mysql:5.6	ip-172-31-25-121.ec2.internal
Running	Running about a minute ago		

A rolling update cannot be performed on a global service to set replicas with the --replicas option, as indicated by the message in the following docker service update command.

```
~ $ docker service update \
> --image mysql \
> --replicas 1 \
> mysql
replicas can only be used with replicated mode
```

As the output indicates, while replicas are set on a replicated service <code>mysql</code>, replicas are not set on the global service.

Summary

This chapter discussed rolling updates on a service. A rolling update on a service involves shutting down previous service tasks and updating the service definition to start new tasks. In the next chapter, we discuss configuring networking in Swarm mode.

CHAPTER 10

Networking

Networking on a Docker Engine is provided by a *bridge* network, the docker0 bridge. The docker0 bridge is local in scope to a Docker host and is installed by default when Docker is installed. All Docker containers run on a Docker host and are connected to the docker0 bridge network. They communicate with each other over the network.

The Problem

The default docker0 bridge network has the following limitations:

- The bridge network is limited in scope to the local Docker host to provide containerto-container networking and not for multi-host networking.
- The bridge network isolates the Docker containers on the host from external access. A Docker container may expose a port or multiple ports and the ports may be published on the host for an external client host access, as illustrated in Figure 10-1, but by default the docker0 bridge does not provide any external client access outside the network.



Figure 10-1. The default docker0 bridge network

The Solution

The Swarm mode (Docker Engine >=1.12) creates an overlay network called *ingress* for the nodes in the Swarm. The ingress overlay network is a multi-host network to route ingress traffic to the Swarm; external clients use it to access Swarm services. Services are added to the ingress network if they publish a port. The ingress overlay network has a default gateway and a subnet and all services in the ingress network are exposed on all nodes in the Swarm, whether a service has a task scheduled on each node or not. In addition to the ingress network, custom overlay networks may be created using the overlay driver. Custom overlay networks provide network connectivity between the Docker daemons in the Swarm and are used for service-to-service communication. Ingress is a special type of overlay network and is not for network traffic between services or tasks. Swarm mode networking is illustrated in Figure 10-2.



Figure 10-2. The Swarm overlay networks

The following Docker networks are used or could be used in Swarm mode.

The Ingress Network

The *ingress* network is created automatically when Swarm mode is initialized. On Docker for AWS, the ingress network is available out-of-the-box because the managed service has the Swarm mode enabled by default. The default overlay network called ingress extends to all nodes in the Swarm, whether the node has a service task scheduled or not. The ingress provides load balancing among a service's tasks. All services that publish a port are added to the ingress network. Even a service created in an internal network is added to ingress if the service publishes a port. If a service does not publish a port, it is not added to the ingress network. A service publishes a port with the --publish or -p option using the following docker service create command syntax.

```
docker service create \
    --name <SERVICE-NAME> \
    --publish <PUBLISHED-PORT>:<TARGET-PORT> \
    <IMAGE>
```

If the <PUBLISHED-PORT> is omitted, the Swarm manager selects a port in the range 30000-32767 to publish the service.

The following ports must be open between the Swarm nodes to use the ingress network.

- Port 7946 TCP/UDP is used for the container network discovery
- Port 4789 UDP is used for the container ingress network

Custom Overlay Networks

Custom overlay networks are created using the overlay driver and services may be created in the overlay networks. A service is created in an overlay network using the --network option of the docker service create command. Overlay networks provide service-to-service communication. One Docker container in the overlay network can communicate directly with another Docker container in the network, whether the container is on the same node or a different node. Only Docker containers for Swarm service tasks can connect with each using the overlay network and not just any Docker containers running on the hosts in a Swarm. Docker containers started with the docker run command, for instance, cannot connect to a Swarm overlay network, using docker network connect
overlay network> <container> for instance.

Nor are Docker containers on Docker hosts that are not in a Swarm able to connect and communicate with Docker containers in the Swarm directly. Docker containers in different Swarm overlay networks cannot communicate with each other directly, as each Swarm overlay network is isolated from other networks.

While the default overlay network in a Swarm, ingress, extends to all nodes in the Swarm whether a service task is running on it or not, a custom overlay network whose scope is also the Swarm does not extend to all nodes in the Swarm by default. A custom Swarm overlay network extends to only those nodes in the Swarm on which a service task created with the custom Swarm overlay network is running.

An "overlay" network overlays the underlay network of the hosts and the scope of the overlay network is the Swarm. Service containers in an overlay network have different IP addresses and each overlay network has a different range of IP addresses assigned. On modern kernels, the overlay networks are allowed to overlap with the underlay network, and as a result, multiple networks can have the same IP addresses.

The docker_gwbridge Network

Another network that is created automatically (in addition to the ingress network) when the Swarm mode is initialized is the docker_gwbridge network. The docker_gwbridge network is a bridge network that connects all the overlay networks, including the ingress network, to a Docker daemon's host network. Each service container is connected to the local Docker daemon host's docker_gwbridge network.

The Bridge Network

A *bridge* network is a network on a host that is managed by Docker. Docker containers on the host communicate with each other over the bridge network. A Swarm mode service that does not publish a port is also created in the bridge network. So are the Docker containers started with the docker run command. This implies that a Swarm mode Docker service that does not publish a port is in the same network as Docker containers started with the docker run command.

CHAPTER 10 NETWORKING

This chapter covers the following topics:

- Setting the environment
- Networking in Swarm mode
- Using the default overlay network ingress to create a service
- Creating a custom overlay network
- Using a custom overlay network to create a service
- Connecting to another Docker container in the same overlay network
- Creating an internal network
- Deleting a network

Setting the Environment

Create a three-node Docker Swarm on Docker for AWS, as discussed in Chapter 3. An AWS CloudFormation stack, shown in Figure 10-3, is used to create a Swarm.



Figure 10-3. AWS CloudFormation stack

Obtain the public IP address of the Swarm manager node, as shown in Figure 10-4.

Q Filter by tags	s and attributes or se	earch by keyword						0	< ·	< 1 to 3 of 3	3 > >	F
Name	* Ins	stance ID ×	Instance Type 👻	Availability Zone -	Instance State	- Stat	us Checks 👻	Alarm Stat	us	Public DNS	(IPv4)	
Docker-Ma	anager i-08	83eb8f9739e82ee1	t2.micro	us-east-1c	running	0	2/2 checks	None	7	ec2-174-129	-48-148	co
Docker-wor	rker i-Oc	d260eb7b8b1b49	t2.micro	us-east-1c	running	0	2/2 checks	None	70	ec2-54-209-1	159-170.	co
Docker-wo	rker i-09	9717d6dc4f309b4e	t2.micro	us-east-1a	running	0	2/2 checks	None	10	ec2-34-205-4	41-131.c	0.
Description	Status Checks	(Docker-Manager)	Public DNS: ec	2-174-129-48-148.co	ompute-1.amazo	naws.co	m	40 440			88	
Description	Status Checks	(Docker-Manager) Monitoring	Public DNS: ec	2-174-129-48-148.co	Public Df	IS (IPv4)	ec2-174-129-	48-148.comp	ute-	-	88	
Description	Status Checks Instance ID	(Docker-Manager) Monitoring	Public DNS: ec	22-174-129-48-148.co	Public DI	IS (IPv4)	m ec2-174-129- 1.amazonaw	48-148.comp s.com	ute-		88	
Description	Status Checks Instance ID	(Docker-Manager) Monitoring	Public DNS: ec	22-174-129-48-148.co	Public Di	IS (IPv4) Public IP	ec2-174-129- 1.amazonawa 174.129.48.1	48-148.comp s.com 48	ute-		88	
Description	Status Checks Instance ID Instance state Instance type	(Docker-Manager) Monitoring 1 i-083eb8f9739e82e running t2.micro	Public DNS: ec	22-174-129-48-148.co	Public Di	IS (IPv4) Public IP IPv6 IPs	m ec2-174-129- 1.amazonaw: 174.129.48.1 - ip.172.31.47	48-148.comp s.com 48	ute-	-	88	
Description	Status Checks Instance ID Instance state Elastic IPs Autibility zone	(Docker-Manager) Monitoring 1 i-083eb8f9739e82e running t2.micro	Public DNS: ec	12-174-129-48-148.co	Public Df	IS (IPv4) Public IP IPv6 IPs ate DNS	m ec2-174-129- 1.amazonawi 174.129.48.1 - ip-172-31-47- 172-31-47-5	48-148.comp s.com 48 -15.ec2.intern	ute- al		80	
Description	Status Checks Instance ID Instance state Instance type Elastic IPs Availability zone Security groups	(Docker-Manager) Monitoring 1 i-083eb8f9739e82e running t2.micro us-east-1c Docker-ManagerVy Docker-SwarmWid view inbound rules	Public DNS: ec Tags le1 bcSG-1C8V48XMFIH eSG-1UW0UXM4BN	12-174-129-48-148.co	Public DI	IS (IPv4) Public IP IPv6 IPs ate DNS ivate IPs ivate IPs	ec2-174-129- 1.amazonaw: 174.129.48.1 - ip-172-31-47- 172.31.47.15	48-148.comp s.com 48 15.ec2.intem	ute-	8	80	
Description	Status Checks Instance ID Instance state Instance type Elastic IPs Availability zone Security groups Scheduled events	(Docker-Manager) Monitoring 1 i-083eb8f9739e82e running t2.micro us-east-1c Docker-ManagerVy Docker-SwarmWid viow inbound rules No scheduled even	Public DNS: ec Tags le1 ccSG-1C8V48XMFIH eSG-1UW0UXM4BN8	22-174-129-48-148.co	Public DI	IS (IPv4) Public IP IPv6 IPs ate DNS ivate IPs ivate IPs	m ec2-174-129- 1.amazonaw 174.129.48.1 - ip-172-31-47- 172.31.47.15 vpc-b35b30c:	48-148.comp s.com 48 15.ec2.intern a	ute- al			

Figure 10-4. Obtaining the public IP address of a Swarm manager node instance

SSH login into the Swarm manager instance.

```
[root@localhost ~]# ssh -i "docker.pem" docker@174.129.48.148
Welcome to Docker!
```

List the Swarm nodes-one manager and two worker nodes.

~ \$ docker node ls				
ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS
npz2akark8etv4ib9biob5yyk	ip-172-31-47-123.ec2.internal	Ready	Active	
p6wat4lxq6a1o3h4fp2ikgw6r	ip-172-31-3-168.ec2.internal	Ready	Active	
tb5agvzbi0rupq7b83tk00cx3 *	ip-172-31-47-15.ec2.internal	Ready	Active	Leader

Networking in Swarm Mode

The Swarm mode provides some default networks, which may be listed with the docker network 1s command. These networks are available not just on Docker for AWS but on any platform (such as CoreOS) in Swarm mode.

~ \$ docker network	ls		
NETWORK ID	NAME	DRIVER	SCOPE
34a5f77de8cf	bridge	bridge	local
0e06b811a613	docker_gwbridge	bridge	local
6763ebad69cf	host	host	local
e41an60iwval	ingress	overlay	swarm
eb7399d3ffdd	none	null	local

We discussed most of these networks in a preceding section. The "host" network is the networking stack of the host. The "none" network provides no networking between a Docker container and the host networking stack and creates a container without network access.

The default networks are available on a Swarm manager node and Swarm worker nodes even before any service task is scheduled.

The listed networks may be filtered using the driver filter set to overlay.

```
docker network ls --filter driver=overlay
```

Only the ingress network is listed. No other overlay network is provisioned by default.

~ \$ docker netwo	ork lsfilter di	river=overlay	
NETWORK ID	NAME	DRIVER	SCOPE
e41an60iwval	ingress	overlay	swarm

The network of interest is the overlay network called ingress, but all the default networks are discussed in Table 10-1 in addition to being discussed in the chapter introduction.

Table 10-1. Docker Networks	able 10-1.
------------------------------------	------------

Network	Description
bridge	The bridge network is the docker0 network created on all Docker hosts. The Docker daemon connects containers to the docker0 network by default. Any Docker container started with the docker run command, even on a Swarm node, connects to the docker0 bridge network.
docker_gwbridge	Used for communication among Swarm nodes on different hosts. The network is used to provide external connectivity to a container that lacks an alternative network for connectivity to external networks and other Swarm nodes. When a container is connected to multiple networks, its external connectivity is provided via the first non- internal network, in lexical order.
host	Adds a container to the host's network stack. The network configuration inside the container is the same as the host's.
ingress	The overlay network used by the Swarm for ingress, which is external access. The ingress network is only for the routing mesh/ingress traffic.
none	Adds a container to a container specific network stack and the container lacks a network interface.

The default networks cannot be removed and, other than the ingress network, a user does not need to connect directly or use the other networks. To find detailed information about the ingress network, run the following command.

docker network inspect ingress

The ingress network's scope is the Swarm and the driver used is overlay. The subnet and gateway are 10.255.0.0/16 and 10.255.0.1, respectively. The ingress network is not an internal network as indicated by the internal setting of false, which implies that the network is connected to external networks. The ingress network has an IPv4 address and the network is not IPv6 enabled.

ſ

```
~ $ docker network inspect ingress
   {
        "Name": "ingress",
        "Id": "e41an60iwvalbeq5y3stdfem9",
        "Created": "2017-07-26T18:38:29.753424199Z",
        "Scope": "swarm",
        "Driver": "overlay",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                     "Subnet": "10.255.0.0/16",
                     "Gateway": "10.255.0.1"
                }
            1
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": true,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "ingress-sbox": {
                "Name": "ingress-endpoint",
                "EndpointID": "f646b5cc4316994b8f9e5041ae7c82550bc7ce733db70df3f
                               66b8d771d0f53c4",
                "MacAddress": "02:42:0a:ff:00:02",
                "IPv4Address": "10.255.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {
            "com.docker.network.driver.overlay.vxlanid list": "4096"
        },
        "Labels": {},
        "Peers": [
            {
                "Name": "ip-172-31-47-15.ec2.internal-17c7f752fb1a",
                "IP": "172.31.47.15"
            },
            {
```

```
"Name": "ip-172-31-47-123.ec2.internal-d6ebe8111adf",
        "IP": "172.31.47.123"
        },
        {
            "Name": "ip-172-31-3-168.ec2.internal-99510f4855ce",
               "IP": "172.31.3.168"
        }
        ]
        }
]
```

Using the Default Bridge Network to Create a Service

To create a service in Swarm mode using the default bridge network, no special option needs to be specified. The --publish or -p option must not be specified. Create a service for the mysql database.

```
~ $ docker service create \
> --env MYSQL_ROOT_PASSWORD='mysql'\
> --replicas 1 \
> --name mysql \
> mysql
likujs72e46ti5go1xjtksnky
```

The service is created and the service task is scheduled on one of the nodes.

~ \$ docker servi	ice ls				
ID	NAME	MODE	REPLICAS	IMAGE	PORTS
likujs72e46t	mysql	replicated	1/1	mysql:latest	

The service may be scaled to run tasks across the Swarm.

~ \$ docker servi	ce scale mysql=3		
mysqi scarca co	-		
~ \$ docker servi	ce ps mysql		
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
v4bn24seygc6	mysql.1	<pre>mysql:latest</pre>	ip-172-31-47-15.ec2.internal
Running	Running 2 minut	tes ago	
29702ebj52gs	mysql.2	mysql:latest	ip-172-31-47-123.ec2.internal
Running	Running 3 secor	nds ago	
c7b8v16msudl	mysql.3	mysql:latest	ip-172-31-3-168.ec2.internal
Running	Running 3 secor	nds ago	

The mysql service created is not added to the ingress network, as it does not publish a port.

Creating a Service in the Ingress Network

In this section, we create a Docker service in the ingress network. The ingress network is not to be specified using the --network option of docker service create. A service must publish a port to be created in the ingress network. Create a Hello World service published (exposed) on port 8080.

```
~ $ docker service rm hello-world
hello-world
~ $ docker service create \
> --name hello-world \
> -p 8080:80\
> --replicas 3 \
>
   tutum/hello-world
176ukzrctq22mn97dmg0oatup
```

The service creates three tasks, one on each node in the Swarm.

~ \$ docker sei	vice ls					
ID	NAME	MODE	REPLICAS	IMAGE		PORTS
176ukzrctq22	hello-world	replicated	3/3	tutum/he	ello-world:latest	*:8080->80/tcp
~ \$ docker ser	vice ps hell	o-world				
ID	NAME	IMAGE			NODE	
DESIRED STATE	CURRENT STA	TE			ERROR	PORTS
5ownzdjdt1yu	hello-world	.1 tutum/h	nello-world:	latest	ip-172-31-14-234.	ec2.internal
Running	Running 33	seconds ago				
csgofrbrznhq	hello-world	.2 tutum/h	nello-world:l	atest	ip-172-31-47-203.	ec2.internal
Running	Running 33	seconds ago				
sctlt9rvn571	hello-world	.3 tutum/h	nello-world:l	atest	ip-172-31-35-44.e	c2.internal
Running	Running 32	seconds ago				

CHAPTER 10 NETWORKING

The service may be accessed on any node instance in the Swarm on port 8080 using the <Public DNS>: <8080> URL. If an elastic load balancer is created, as for Docker for AWS, the service may be accessed at <LoadBalancer DNS>: <8080>, as shown in Figure 10-5.



Figure 10-5. Invoking a Docker service in the ingress network using EC2 elastic load balancer public DNS

The <PublishedPort> 8080 may be omitted in the docker service create command.

```
~ $ docker service create \
> --name hello-world \
> -p 80\
> --replicas 3 \
> tutum/hello-world
pbjcjhx163wm37d5cc5au2fog
```

Three service tasks are started across the Swarm.

~ \$ docker ser	vice ls				
ID	NAME	MODE	REPLICAS	IMAGE	PORTS
pbjcjhx163wm	hello-world	replicated	3/3	<pre>tutum/hello-world:latest</pre>	*:0->80/tcp

```
~ $ docker service ps hello-world
               NAME
                               IMAGE
                                                          NODE
ID
DESIRED STATE
               CURRENT STATE
                                         ERROR
                                                 PORTS
xotbpv10508n
               hello-world.1
                               tutum/hello-world:latest
                                                          ip-172-31-37-130.ec2.internal
Running
                Running 13 seconds ago
nvdn3j5pzuqi
               hello-world.2
                               tutum/hello-world:latest
                                                          ip-172-31-44-205.ec2.internal
Running
                Running 13 seconds ago
uuveltc5izpl
               hello-world.3
                               tutum/hello-world:latest
                                                          ip-172-31-15-233.ec2.internal
                Running 14 seconds ago
Running
```

The Swarm manager automatically assigns a published port (30000), as listed in the docker service inspect command.

```
~ $ docker service inspect hello-world
[
        "Spec": {
            "Name": "hello-world",
. . .
            "EndpointSpec": {
                 "Mode": "vip",
                 "Ports": [
                     {
                         "Protocol": "tcp",
                         "TargetPort": 80,
                          "PublishMode": "ingress"
                     }
                 ]
            }
        },
        "Endpoint": {
             "Spec": {
                 "Mode": "vip",
                 "Ports": [
                     {
                         "Protocol": "tcp",
                          "TargetPort": 80,
                         "PublishMode": "ingress"
                     }
                 ]
            },
"Ports": [
                 ł
                     "Protocol": "tcp",
                     "TargetPort": 80,
                     "PublishedPort": 30000,
                     "PublishMode": "ingress"
                 }
            ],
```

Even though the service publishes a port (30000 or other available port in the range 30000-32767), the AWS elastic load balancer for the Docker for AWS Swarm does not add a listener for the published port (30000 or other available port in the range 30000-32767). We add a listener with <Load Balancer Port:Instance Port> mapping of 30000:30000, as shown in Figure 10-6.

< < 1 to 1 of 1 > ⇒			×		C, Search	Filter: Q, Se
• Availability Zones • T	- VPC ID	- State	S name	~ DI	ne	Name
us-east-1a, us-east-1b, cl	vpc-1dc7b464		ker-ExternalLoa-LPH5A	AXKEV140 Do	ker-ExternalLoa-LPH	Docker-8
			140	rnalLoa-LPH5AXKE	ncer: Docker-Exte	Load balance
88		Tags	isteners Monitoring	malLoa-LPH5AXKE	incer: Docker-Extension Instances	Load balancer
88		Tags	140 Isteners Monitoring his load balancer:	rnalLoa-LPH5AXKE Health Check	Incer: Docker-Extension Instances	Load balancer Description The following
88	SSL Certificate	Tags	140 Isteners Monitoring his load balancer. Instance Protocol	malLoa-LPH5AXKE Health Check	Incer: Docker-Extension Instances	Load balancer Description The following Load Balar

Figure 10-6. Adding a load balancer listener

Invoke the service at the <Load Balancer DNS>:<30000> URL, as shown in Figure 10-7.



Figure 10-7. Invoking a Hello World service on port 30000

Creating a Custom Overlay Network

We used the default overlay network ingress provisioned in Swarm mode. The ingress network is only for the Swarm mode routing mesh in which all nodes are included. The Swarm routing mesh is provided so that each node in the Swarm may accept connections on published ports for services in the Swarm even if a service does not run a task on a node. The ingress network is not for service-to-service communication.

A custom overlay network may be used in Swarm mode for service-to-service communication. Next, create an overlay network using some advanced options, including setting subnets with the --subnet option and the default gateway with the --gateway option, as well as the IP range with the --ip-range option. The --driver option must be set to overlay and the network must be created in Swarm mode. A matching subnet for the specified IP range must be available. A subnet is a logical subdivision of an IP network. The gateway is a router that links a host's subnet to other networks. The following command must be run from a manager node.

```
~ $ docker network create \
```

- > --subnet=192.168.0.0/16 \
- > --subnet=192.170.0.0/16 \
- > --gateway=192.168.0.100 \
- > --gateway=192.170.0.100 \
- > --ip-range=192.168.1.0/24 \

> --driver overlay \
> mysql-network
mkileuo6ve329jx5xbd1m6r10

The custom overlay network is created and listed in networks as an overlay network with Swarm scope.

~ \$ docker network 1	ls		
NETWORK ID	NAME	DRIVER	SCOPE
34a5f77de8cf	bridge	bridge	local
0e06b811a613	docker_gwbridge	bridge	local
6763ebad69cf	host	host	local
e41an60iwval	ingress	overlay	swarm
mkileuo6ve32	mysql-network	overlay	swarm
eb7399d3ffdd	none	null	local

Listing only the overlay networks should list the ingress network and the custom mysql-network.

~ \$ docker network]	lsfilter driver=o\	/erlay	
NETWORK ID	NAME	DRIVER	SCOPE
e41an60iwval	ingress	overlay	swarm
mkileuo6ve32	mysql-network	overlay	swarm

The detailed information about the custom overlay network <code>mysql-network</code> lists the subnets and gateways.

```
~ $ docker network inspect mysql-network
[
    {
        "Name": "mysql-network",
        "Id": "mkileuo6ve329jx5xbd1m6r1o",
        "Created": "0001-01-01T00:00:00Z",
        "Scope": "swarm",
        "Driver": "overlay",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                     "Subnet": "192.168.0.0/16",
                     "IPRange": "192.168.1.0/24",
                     "Gateway": "192.168.0.100"
                },
                {
                    "Subnet": "192.170.0.0/16",
                    "Gateway": "192.170.0.100"
                }
            1
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
```

Only a single overlay network can be created for specific subnets, gateways, and IP ranges. Using a different subnet, gateway, or IP range, a different overlay network may be created.

```
~ $ docker network create \
> --subnet=10.0.0.0/16 \
> --gateway=10.0.0.100 \
> --ip-range=10.0.1.0/24 \
> --driver overlay \
> mysql-network-2
qwgb1lwycgvogoq9t62ea4ny1
```

The mysql-network-2 is created and added to the list of networks.

ls		
NAME	DRIVER	SCOPE
bridge	bridge	local
docker_gwbridge	bridge	local
host	host	local
ingress	overlay	swarm
mysql-network	overlay	swarm
mysql-network-2	overlay	swarm
none	null	local
	ls NAME bridge docker_gwbridge host ingress mysql-network mysql-network-2 none	ls NAME DRIVER bridge bridge docker_gwbridge bridge host host ingress overlay mysql-network overlay mysql-network-2 overlay none null

New overlay networks are only made available to worker nodes that have containers using the overlay. While the new overlay networks mysql-network and mysql-network-2 are available on the manager node, the network is not extended to the two worker nodes. SSH login to a worker node.

[root@localhost ~]# ssh -i "docker.pem" docker@54.209.159.170
Welcome to Docker!

The mysql-network and mysql-network-2 networks are not listed on the worker node.

ls		
NAME	DRIVER	SCOPE
bridge	bridge	local
docker_gwbridge	bridge	local
host	host	local
ingress	overlay	swarm
none	null	local
	ls NAME bridge docker_gwbridge host ingress none	ls DRIVER bridge Dridge docker_gwbridge bridge host host ingress overlay none null

To extend the custom overlay network to worker nodes, create a service in the network that runs a task on the worker nodes, as we discuss in the next section.

The Swarm mode overlay networking is secure by default. The gossip protocol is used to exchange overlay network information between Swarm nodes. The nodes encrypt and authenticate the information exchanged using the AES algorithm in GCM mode. Manager nodes rotate the encryption key for gossip data every 12 hours by default. Data exchanged between containers on different nodes on the overlay network may also be encrypted using the --opt encrypted option, which creates IPSEC tunnels between all the nodes on which tasks are scheduled. The IPSEC tunnels also use the AES algorithm in GCM mode and rotate the encryption key for gossip data every 12 hours. The following command creates an encrypted network.

```
~ $ docker network create \
> --driver overlay \
> --opt encrypted \
> overlay-network-2
aqppoe3qpy6mzln46q5tunecr
```

A Swarm scoped network that is encrypted is created.

~ \$ docker network] c		
NETWORK ID	NAME	DRIVER	SCOPE
34a5f77de8cf	bridge	bridge	local
0e06b811a613	docker gwbridge	bridge	local
6763ebad69cf	host	host	local
e41an60iwval	ingress	overlay	swarm
mkileuo6ve32	mysql-network	overlay	swarm
qwgb1lwycgvo	mysql-network-2	overlay	swarm
eb7399d3ffdd	none	null	local
аqррое3qру6m	overlay-network-2	overlay	swarm

Using a Custom Overlay Network to Create a Service

If a custom overlay network is used to create a service, the --network must be specified. The following command creates a MySQL database service in Swarm mode using the custom Swarm scoped overlay network mysql-network.

```
~ $ docker service create \
> --env MYSQL_ROOT_PASSWORD='mysql'\
> --replicas 1 \
> --network mysql-network \
> --name mysql-2\
> mysql
ocd9sz8qqp2becf0ww2rj5p5n
```

The mysql-2 service is created. Scale the mysql-2 service to three replicas and lists the service tasks for the service.

```
~ $ docker service scale mysql-2=3
mysql-2 scaled to 3
```

Docker containers in two different networks for the two services—mysql (bridge network) and mysql-2 (mysql-network overlay network)—are running simultaneously on the same node.

A custom overlay network is not extended to all nodes in the Swarm until the nodes have service tasks that use the custom network. The mysql-network does not get extended to and get listed on a worker node until after a service task for mysql-2 has been scheduled on the node.

A Docker container managed by the default Docker Engine bridge network docker0 cannot connect with a Docker container in a Swarm scoped overlay network. Using a Swarm overlay network in a docker run command, connecting with a Swarm overlay network with a docker network connect command, or linking a Docker container with a Swarm overlay network using the --link option of the docker network connect command is not supported. The overlay networks in Swarm scope can only be used by a Docker service in the Swarm.

For connecting between service containers:

- Docker containers for the same or different services in the same Swarm scoped overlay network are able to connect with each other.
- Docker containers for the same or different services in different Swarm scoped overlay networks are not able to connect with each other.

In the next section, we discuss an internal network, but before we do so, the external network should be introduced. The Docker containers we have created as of yet are external network containers. The ingress network and the custom overlay network mysql-network are external networks. External networks provide a default route to the gateway. The host and the wider Internet network may connect to a Docker container in the ingress or custom overlay networks. As an example, run the following command to ping google.com from a Docker container's bash shell; the Docker container should be in the ingress overlay network or a custom Swarm overlay network.

```
docker exec -it <containerid> ping -c 1 google.com
```

A connection is established and data is exchanged. The command output is shown in italics.

```
~ $ docker exec -it 3762d7c4ea68 ping -c 1 google.com
PING google.com (172.217.7.142): 56 data bytes
64 bytes from 172.217.7.142: icmp_seq=0 ttl=47 time=0.703 ms
--- google.com ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.703/0.703/0.703/0.000 ms
```

Creating an Internal Overlay Network

In this section, we discuss creating and using an internal overlay network. An internal network does not provide external connectivity. What makes a network internal is that a default route to a gateway is not provided for external connectivity from the host or the wider Internet.

First, create an internal overlay network using the --internal option of the docker network create command. Add some other options, such as --label, which have no bearing on the internal network. It's configured with the --internal option of the docker network create command.

```
~ $ docker network create \
```

```
> --subnet=10.0.0/16 \
```

```
> --gateway=10.0.0.100 \
```

```
> --internal \
```

```
> --label HelloWorldService \
```

```
> --ip-range=10.0.1.0/24 \
```

```
> --driver overlay \
> hello-world-network
pfwsrjeakomplo5zm6t4p19a9
```

The internal network is created and listed just the same as an external network would be.

~ \$ docker network	ls		
NETWORK ID	NAME	DRIVER	SCOPE
194d51d460e6	bridge	bridge	local
a0674c5f1a4d	docker_gwbridge	bridge	local
pfwsrjeakomp	hello-world-network	overlay	swarm
03a68475552f	host	host	local
tozyadp06rxr	ingress	overlay	swarm
3dbd3c3ef439	none	null	local

In the network description, the internal is set to true.

```
core@ip-172-30-2-7 ~ $ docker network inspect hello-world-network
ſ
    {
        "Name": "hello-world-network",
        "Id": "58fzvj4arudk2053q6k2t8rrk",
        "Scope": "swarm",
        "Driver": "overlay",
        "EnableIPv6": false,
        "IPAM": {
             "Driver": "default",
             "Options": null,
             "Config": [
                 {
                     "Subnet": "10.0.0.0/16",
"IPRange": "10.0.1.0/24",
                      "Gateway": "10.0.0.100"
                 }
             ]
        },
        "Internal": true,
        "Containers": null,
        "Options": {
             "com.docker.network.driver.overlay.vxlanid_list": "257"
        },
        "Labels": {
             "HelloWorldService": ""
        }
    }
]
```

Create a service that uses the internal network with the --network option.

~ \$ docker service create \
> --name hello-world \
> --network hello-world-network \
> --replicas 3 \
> tutum/hello-world
hm5pf6ftcvphdrd2zm3pp4lpj

The service is created and the replicas are scheduled. Obtain the container ID for one of the service tasks, d365d4a5ff4c.

~ \$ docker ps CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES d365d4a5ff4c tutum/hello-world:latest "/bin/sh -c 'php-f..." About a minute ago Up About a minute hello-world.3.r759ddnl1de11spo0zdi7xj4z

As before, ping google.com from the Docker container.

docker exec -it <containerid> ping -c 1 google.com

A connection is not established, which is because the container is in an internal overlay network.

```
~ $ docker exec -it d365d4a5ff4c ping -c 1 google.com
ping: bad address 'google.com'
```

Connection is established between containers in the same internal network, as the limitation is only on external connectivity. To demonstrate, obtain the container ID for another container in the same internal network.

~ \$ docker ps						
CONTAINER ID	IMAGE		COMMAND		CREATED	
STATUS	PORTS	NAMES				
b7b505f5eb8d	tutum/hello	-world:latest	"/bin/sh -c	'php-f"	3 seconds	ago
Up 2 seconds		hello-world.6.i	60ezt6da2t1o	dwdjvecb75fx		
57e612f35a38	tutum/hello-	-world:latest	"/bin/sh -c	'php-f"	3 seconds	ago
Up 2 seconds		hello-world.7.6	5ltqnybn8twht	blpqjtvulkup		
d365d4a5ff4c	tutum/hello-	-world:latest	"/bin/sh -c	'php-f"	7 minutes	ago
Up 7 minutes		hello-world.3.1	759ddnl1de11	spo0zdi7xj4z		

Connect between two containers in the same internal network. A connection is established.

~ \$ docker exec -it d365d4a5ff4c ping -c 1 57e612f35a38
PING 57e612f35a38 (10.0.1.7): 56 data bytes
64 bytes from 10.0.1.7: seq=0 ttl=64 time=0.288 ms

```
--- 57e612f35a38 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.288/0.288/0.288 ms
```

If a service created in an internal network publishes (exposes) a port, the service gets added to the ingress network and, even though the service is in an internal network, external connectivity is provisioned. As an example, we add the --publish option of the docker service create command to publish the service on port 8080.

```
~ $ docker service create \
> --name hello-world \
> --network hello-world-network \
> --publish 8080:80 \
> --replicas 3 \
> tutum/hello-world
mqgek4umisgycagy4qa206f9c
```

Find a Docker container ID for a service task.

```
~ $ docker ps
CONTAINER ID IMAGE COMMAND CREATED
STATUS PORTS NAMES
1c52804dc256 tutum/hello-world:latest "/bin/sh -c 'php-f..." 28 seconds ago
Up 27 seconds 80/tcp hello-world.1.20152n01ng3t6uaiahpex9n4f
```

Connect from the container in the internal network to the wider external network at google.com, as an example. A connection is established. Command output is shown in italics.

```
~ $ docker exec -it 1c52804dc256 ping -c 1 google.com
PING google.com (172.217.7.238): 56 data bytes
64 bytes from 172.217.7.238: seg=0 ttl=47 time=1.076 ms
```

```
--- google.com ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 1.076/1.076/1.076 ms
```

Deleting a Network

A network that is not in use may be removed with the docker network rm <networkid> command. Multiple networks may be removed in the same command. As an example, we can list and remove multiple networks.

~ \$ docker network	ls		
NETWORK ID	NAME	DRIVER	SCOPE
34a5f77de8cf	bridge	bridge	local
0e06b811a613	docker_gwbridge	bridge	local
wozpfgo8vbmh	hello-world-network		swarm
6763ebad69cf	host	host	local
e41an60iwval	ingress	overlay	swarm
mkileuo6ve32	mysql-network	overlay	swarm
qwgb1lwycgvo	mysql-network-2	overlay	swarm
eb7399d3ffdd	none	null	local
aqppoe3qpy6m	overlay-network-2	overlay	swarm

Networks that are being used by a service are not removed. The command output is shown in italics.

~ \$ docker network rm hello-world-network mkileuo6ve32 qwgb1lwycgvo overlay-network-2 hello-world-network Error response from daemon: rpc error: code = 9 desc = network mkileuo6ve329jx5xbd1m6r1o is in use by service ocd9sz8qqp2becf0ww2rj5p5nqwgb1lwycgvo overlay-network-2

Summary

This chapter discussed the networking used by the Docker Swarm mode. The default networking used in Swarm mode is the overlay network ingress, which is a multi-host network spanning all Docker nodes in the same Swarm to provide a routing mesh for each node to be able to accept ingress connections for services on published ports. Custom overlay network may be used to create a Docker service with the difference that a custom overlay network provides service-to-service communication instead of ingress communication and extends to a Swarm worker node only if a service task using the network is scheduled on the node. The chapter also discussed the difference between an internal and an external network. In the next chapter, we discuss logging and monitoring in Docker Swarm mode.

CHAPTER 11

Logging and Monitoring

Docker includes several built-in logging drivers for containers, such as json-file, syslog, journald, gelf, fluentd, and awslogs. Docker also provides the docker logs command to get the logs for a container. Docker 1.13 includes an experimental feature for getting a Docker service log using the docker service logs command.

The Problem

Docker Swarm mode does not include a native monitoring service for Docker services and containers. Also the experimental feature to get service logs is a command-line feature and required to be run per service. A logging service with which all the services' logs and metrics could be collected and viewed in a dashboard is lacking.

The Solution

Sematext is an integrated data analytics platform that provides SPM performance monitoring for metrics and events collection, and Logsene for log collection, including correlation between performance metrics, logs, and events. Logsene is a hosted ELK (Elasticsearch, Logtash, Kibana) stack. Sematext Docker Agent is required to be installed on each Swarm node in the Swarm for continuously collecting logs, metrics, and events, as illustrated in Figure 11-1.

CHAPTER 11 LOGGING AND MONITORING



Figure 11-1. Sematext Docker agent on each Swarm node

This chapter covers the following topics:

- Setting the environment
- Creating a SPM application
- Creating a Logsene application
- Deploying the Sematext Docker agent as a service
- Creating a MySQL database deployment on Docker Swarm
- Monitoring the Docker Swarm metrics
- Getting Docker Swarm logs in Logsene

Setting the Environment

Start a three-node Swarm consisting of one manager and two worker nodes using Docker for AWS. (This is discussed in Chapter 3.) Obtain the public IP address of the manager node instance from the EC2 console and SSH login into the instance.

```
[root@localhost ~]# ssh -i "docker.pem" docker@54.227.123.67
Welcome to Docker!
```

The procedure to use Sematext SPM and Logsene for logging and monitoring with a Docker Swarm is as follows.

- 1. Create an account at https://apps.sematext.com/ui/registration.
- 2. Log in to the user account at https://apps.sematext.com/ui/login.
- 3. Select the integrations (Logsene app and SPM Docker app) from https://apps. sematext.com/ui/integrations?newUser, as listed in Steps 4 and 5.
- **4.** Create a SPM (a performance monitoring app). An app is like a namespace for data. A SPM token is generated that is to be used to install a Sematext agent on each Swarm node.
- 5. Create a Logsene app. A Logsene token is generated that is also used to install a Sematext agent on each Swarm node.
- 6. Install a Sematext agent on each Swarm node. Docker Swarm metrics, logs, and events start getting collected in the SPM dashboard and the Logsene dashboard.

Creating a SPM Application

Log in to a Sematext account at https://apps.sematext.com/ui/integrations?newUser to display the Integrations page. For a SPM Docker app, select Docker from Infrastructure and Application Performance Monitoring. In the Add SPM Docker App dialog, specify an application name (DockerSwarmSPM), as shown in Figure 11-2. Click on Create App.



Figure 11-2. Adding a SPM Docker app
An SPM App is created, as shown in Figure 11-3. Several client configurations are listed.



Figure 11-3. SPM app is created

Click on the Client Configuration tab for Docker Swarm, as shown in Figure 11-4. The Docker Swarm tab displays the docker service create command to create a service for a Sematext Docker agent; copy the command. The command includes a SPM_TOKEN, which is unique for each SPM app.

Requirement: Docker v1.6 or newer installed (or v1.12 in case of Docker Swarm).								
Linux / Mac OS X / Windows	CoreOS Kubernetes	RancherOS	Mesos	Docker Swarm				
1. The following configuration docker service create restart-condition any name sematext-agent-do	n will activate Sematext Do mode global \ \ scker \ rar/run/docker.sock,dst=/va	ocker Agent on eve r/run/docker.sock	ery node in	the Swarm Cluster cluster				
mount type=bind, src=/v		and the second second second second						

Figure 11-4. Docker Swarm configuration

The SPM app is added to the dashboard, as shown in Figure 11-5. Click on the App link to navigate to App Reports, which shows the monitoring data, metrics, and events collected by the SPM app and the charts generated from the data.

<		A11	4000							
2 Dashboards		AU	Apps 0							
Infrastructure	\rightarrow									
Monitoring		10	• of 1			Search or filter using tags				*
≡ Logs		Туре	Application 1	Token 1	PL	an I	State 1	Your Role	Data Received 1	Actions
X Correlations		٠	DockerSwarmSPM	9b5552fd-xxxxxxxxxxxx76046d4a3413	Pr	o Silver SPM Docker	0	OWNER		
🏳 Alerts & Events			Go to	App Reports						
🚳 Integrations	-									
Overview										
Apps										
Connected Apps										
Notification Hooks										

Figure 11-5. DockerSwarmSPM app on the dashboard

As the message in Figure 11-6 indicates, the app has not received any data yet. All the metrics graphs are empty initially, but they will display the graphs when data starts getting received.

The DockerSwarmSPM App has not received any data yet, wa Ensure you can connect to SPM Receiver https://sm- missing required entries. See how to check network co Make sure the disk where SPM agent is installed is not check if you missed any of the steps in installation ins Please see known issues and troublemoting will i If nothing else helped, please contact us directly via live	It a few more minutes or check troubleshooting tips: ecciver.seatest.com/receiver/vs/_bulk i Proxy settings may b neetivity, full pructions e chat or via email.	be needed, firew	vall may be blocking outbound traffic, or your DNS could be
🕍 Host CPU 🛛 🤤 🧭	Le Container CPU	0 = 2 × 1	🛎 Container Memory 0 🕳
Host CPU O = 2 <	Image: Container CPU C	0 = x ² ∨ 1 1ns 0 8ns 0 7ns 0 0ns 0 5ns 0 5ns	Container Memory C T Container Memory C T There is no data for this chart. Check SPM troubleshooting guide

Figure 11-6. The DockerSwarmSPM app has not received any data

Creating a Logsene Application

To create a Logsene app, select Logs App from the integrations page at https://apps.sematext.com/ui/integrations?newUser, as shown in Figure 11-7.

<		
🕐 Dashboards	You have no Apps for your logs yet. Create a new App and enjoy your logs!	
Infrastructure		
Monitoring		
All Monitoring Apps	+	
DockerSwarmSPM	Logs App	
Docker	t	
OS	Create new Logsene app	
Custom Metrics		
E Logs		
X Correlations		
🏳 Alerts & Events		
lntegrations		

Figure 11-7. Selecting the Logs app

In the Add Logsene App dialog, specify an application name (DockerSwarmLogsene) and click on Create App, as shown in Figure 11-8.

<		Add Locence Ann
2 Dashboards		Add Logsene App
Infrastructure	×	Application name
Monitoring	-	DockerSwarmLogsene
All Monitoring Apps DockerSwarmSPM Docker OS Custom Metrics	•	Discount code Educational institution? Non-profit? Small startup? Get in touch later for d Invite team members Enter emails separated by comma
≡ Logs		6
X Correlations		
Alerts & Events	Þ	Create App
& Integrations		V

Figure 11-8. Adding the Logsene app

A new Logsene application called DockerSwarmLogsene is created, as shown in Figure 11-9. Copy the LOGSENE_TOKEN that's generated, which we will use to create a Sematext Docker agent service in a Docker Swarm.

•	Overview		
-	Programming Languages		🞗 Waiting for data. Configure your log shipping.
=	Log Shippers	*	Docker
	os	Σ	We made it super simple to collect logs with Docker.
=	Docker	-	Setup
×	Docker		Add *-e LOGSENE_TOKEN=81ac5395-fe8f-47d9-93b2-dc00c649116a* to the Docker run command of the Sematext Docker Agent to get all container and host logs into Logsene.
а	Kubernetes		The following will work with any Docker host. This will work with RancherOS as well.
8	Mesos Marathon Cloud Iaas / PaaS Mobile	÷	sudo docker pull senatext/senatext-agent-docker sudo docker run -dname senatext-agentrestart=alxays \ -e LOGSINE_TOC(N=BacSJ95-FeB-r4-269-902-coBeC691166 \ -V_VMPT_UMPdSHer_Lock(rVMPTUMPdSHer_Lock \
	ios		-v /etc/localtime:/etc/localtime:ro \ sematext/sematext-agent-docker
	Android		At the same time, you can also collect Docker Metrics and Events and send them to SPM for Docker monitoring and alerting by adding*-e SPM_TOKEN= <your_spm_token>* to the Docker run command. To get an SPM Token create SPM app for Docker.</your_spm_token>

Figure 11-9. The Logsene app is added and LOGSENE_TOKEN is generated

A new Logsene application called DockerSwarmLogsene is added to the dashboard, as shown in Figure 11-10.

< ✿ Dashboards ■ Infrastructure	All	Apps O						
Monitoring	10	* of 2		Search or filter using t	ags			٣
E Logs	Туре	Application I	Token [Plan I	State [Your Role 1	Data Received 1	Actions
X Correlations	=	DockerSwarmLogsene	81ac5395-xxxxxxxxxxxxxx-dc00c649116a	Pro, 7 days, 10 GB/day	0	OWNER		
Alerts & Events	*	DockerSwarmSPM	9b5552fd-xxxxxxxxxxxxx76046d4a3413	Pro Silver SPM Docker	0	OWNER		
🚳 Integrations								

Figure 11-10. The DockerSwarmLogsene app

Click on the DockerSwarmLogsene app link to display the log data collected by the app. Initially, the app does not receive any data, as indicated by a message in Figure 11-11, because we have not yet configured a Sematext Docker agent service on the Docker Swarm. The Logsene UI is integrated with the Kibana dashboard.

>		8	*	>
B	The DockerSwarmLogsene App has not received any data yet, wait a few more minutes or check troubleshooting tips:		×	Type to filter fields
	 Double-check your log shipping configuration, esp. any errors messages log shipper itself may be logging to its own log files or console Try running your log shipper in debug and/or verbose mode Check if there are any network connectivity issues, see our viki page Check that your servers' clocks are correct and check that your logs have accurate timestamps and not timestamps from the future Check that your servers' clocks are correct and check that your servers' clocks are correct and check that your acceptable timestamps: Elasticescarch API: 2001-06-08108:000:011232 (ISO8601) or Oct 11 2214:15 (RFC3164 timestamp - UTC) For more details around timestamp formats see wiki page Check integrations guide once more or consider using a different log shipper See more details on how to troubleshoot Logsene apps on Logsene troubleshooting wiki If after all of the above you still don't see your logs please contact us via mail or via our live chat. So we can help you faster, please let u which log shipping method you are using. 	s know		_source _index _type _id _version TAGS
	Search your logs Q Log Counts	8 0 = 1	*	
	No data found for the selected time period	0 mate	hes	

Figure 11-11. The app does not receive any data at first

Connecting the SPM and Logsene Apps

Next, connect the SPM and Logsene apps so that the metrics and events collected by the SPM are integrated with the Logsene app. Choose Integrations \succ Connected Apps, as shown in Figure 11-12.

<	First app		6		
DockerSwarmSPM	Select		You can co	nnect any two apps you have access to, regardless	of their type.
Docker	, Second app		A single ap	p can be connected to any number of other apps.	
OS Custom Metrics	Select		• Why con	nect apps? ecting a logs app to a monitoring app will pre-sele	ct that logs app when you
E Logs	Connect apps		decid	de to correlate metrics from the connected monitor	ing app, and thus save you
All Logs Apps			auto	matically include information (e.g. charts) from con-	nected apps, and thus provide
DockerSwarmLogsene	1200000000000		more	montation and context for you.	
X Correlations	Connected apps				
P Alerts & Events	, 5 ▼ of 0			Search or filter using tags	¥
💫 Integrations	First App 1	Second App 1	Created On 1	Created By	Actions
Overview	There is no data availa	ble			
Apps					
Connected Apps					
Notification Hooks					

Figure 11-12. Choosing Integrations ➤ *Connected Apps*

Select DockerSwarmSPM as the first app and DockerSwarmLogsene as the second app, as shown in Figure 11-13. Then click on Connect Apps.

<		First app			A		
DockerSwarmSPM		DockerSwarmSPM		× *	You can co	of their type.	
Docker	,	Second app			A single ap	op can be connected to any number of other apps.	
OS Custom Metrics		DockerSwarmLogsene		× *	Why con • conr	nect apps? necting a logs app to a monitoring app will pre-selec	t that logs app when you
≡ Logs		Connect apps			time	de to correlate metrics from the connected monitori h.	ng app, and thus save you
All Logs Apps DockerSwarmLogsene		C)			• whe auto more	n you receive an alert notification for an app, the ale matically include information (e.g. charts) from conn e information and context for you.	rt notification will ected apps, and thus provide
X Correlations		Connected apps					
🏳 Alerts & Events	,	5 • of 0				Search or filter using tags	•
lntegrations		First App [Second App 1		Created On [Created By 1	Actions
Overview		There is no data available					
Apps							
Connected Apps							

Figure 11-13. DockerSwarmLogsene

The connected apps are listed, as shown in Figure 11-14.

<		First app		6			
DockerSwarmSPM		Select		- You can con	nnect any two ap	ps you have access to, regardless of their	type.
Docker OS	ь 1	Second app		Why con	p can be connect nect apps?	ed to any number of other apps.	
Custom Metrics		Connect apps		 conn decid time. 	ecting a logs app ie to correlate me	to a monitoring app will pre-select that l atrics from the connected monitoring app	ogs app when you , and thus save you
All Logs Apps DockerSwarmLogsene				• when autor more	nyou receive an a matically include information and	lert notification for an app, the alert notif information (e.g. charts) from connected a context for you.	ication will apps, and thus provide
X Correlations		Connected apps					
🏳 Alerts & Events	-	5 * of 1			Search or filter	using tags	*
🗞 Integrations	-	First App 1	Second App 1	Created O	n I	Created By [Actions
Overview	- 1	DockerSwarmSPM	DockerSwarmLogsene	2017-07-2	7 00:18:17	dvohra17@yahoo.com	
Apps	C	è					
Connected Apps	- 1						

Figure 11-14. The connected apps

Deploying the Sematext Docker Agent as a Service

The docker service create command copied earlier includes just the SPM_TOKEN token. Add -e LOGSENE_TOKEN obtained from the Logsene app. Run the docker service create command on the Swarm manager node.

```
~ $ docker service create --mode global \
```

```
> --restart-condition any \
```

```
> --name sematext-agent-docker \
```

```
> --mount type=bind,src=/var/run/docker.sock,dst=/var/run/docker.sock \
```

```
> --mount type=bind,src=/,dst=/rootfs,readonly=true \
```

```
> -e SPM_TOKEN=9b5552fd-001d-44f0-9452-76046d4a3413 \
```

```
> -e LOGSENE_TOKEN=81ac5395-fe8f-47d9-93b2-dc00c649116a \
```

```
> sematext/sematext-agent-docker
```

```
oubjk53mpdnjgak5dgfdxs4ft
```

A service for the Sematext Docker agent is created; it's listed using docker service 1s.

~ \$ docker service ls ID NAME MODE REPLICAS IMAGE PORTS sematext-agent-docker global 3/3 sematext/sematext-agent-docker:latest oubjk53mpdnj List the service tasks. As this is a global service, one task gets started on each node. ~ \$ docker service ps sematext-agent-docker ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR PORTS sematext-agent-docker.8d0qv1epqu8xop4o2f94i8j40 5jvl7gnvl0te sematext/sematext-agentdocker:latest Running 2 minutes ago ip-172-31-8-4.ec2.internal Running y53f20d3kknh sematext-agent-docker.xks3sw6qgwbcuacyypemfbxyj sematext/sematext-agentdocker:latest ip-172-31-31-117.ec2.internal Running Running 2 minutes ago t5w2pxy4fc9l sematext-agent-docker.r02ftwtp3n4m0cl7v2llw4gi8 sematext/sematext-agentdocker:latest ip-172-31-44-8.ec2.internal Running 2 minutes ago Running

If additional nodes are added to the Swarm, the Sematext Docker agent starts a service task on the new nodes. As an example, update the CloudFormation stack to increase the number of manager nodes to three and worker nodes to five, as shown in Figure 11-15.

1 CloudFormation	✓ Stacks → Stack Deta	all > Update Stack
Update Docke	er stack	
Select Template	Specify Details	
Specify Details Options Review	Specify parameter values. You	can use or change the default parameter values, which are defined in the AWS CloudFormation template. Learn more.
	Stack name	Docker
	Parameters	
	Swarm Size	
	Number of Swarm managers?	3 v Number of Swarm manager nodes (1, 3, 5)
	Number of Swarm worker nodes?	5 Number of worker nodes in the Swarm (0-1000).

Figure 11-15. Increasing the number of worker nodes

The Swarm nodes are increased to three manager nodes and five worker nodes when the Stack update is complete.

```
~ $ docker node ls
                           HOSTNAME
                                                        STATUS AVAILABILITY MANAGER STATUS
ID
8d0qv1epqu8xop4o2f94i8j40
                           ip-172-31-8-4.ec2.internal
                                                        Ready Active
9rvieyqnndgecagbuf73r9gs5
                           ip-172-31-35-125.ec2.internal Ready Active
                                                                            Reachable
j4mg3fyzjtsdcnmr7rkiytltj
                           ip-172-31-18-156.ec2.internal Ready Active
mhbbunhl358chah1dmr0y6i71
                                                                            Reachable
                           ip-172-31-7-78.ec2.internal Ready Active
r02ftwtp3n4m0cl7v2llw4gi8
                                                        Ready Active
                           ip-172-31-44-8.ec2.internal
vdamjjjrz7a3ri3prv9fjngvy ip-172-31-6-92.ec2.internal
                                                        Ready Active
xks3sw6qgwbcuacyypemfbxyj * ip-172-31-31-117.ec2.internal Ready Active
                                                                            Leader
xxyy4ys4oo30bb4l5daoicsr2
                           ip-172-31-21-138.ec2.internal Ready Active
```

Adding nodes to the Swarm starts a Sematext agent on the nodes that were added.

~ \$ docker service	ps sematext-agent-docl	ker	
ID	NAME		
IMAGE		NODE	DESIRED STATE
CURRENT STATE	ERROR	PORTS	
cgaturw05p59	sematext-agent-docker	.xxyy4ys4oo30bb4l5daoicsr2	
<pre>sematext/sematext-a</pre>	gent-docker:latest	ip-172-31-21-138.ec2.internal	Running
Running 2 minutes a	go		
lj4f46q3ydv1	sematext-agent-docker	r.j4mg3fyzjtsdcnmr7rkiytltj	
<pre>sematext/sematext-a</pre>	<pre>gent-docker:latest</pre>	ip-172-31-18-156.ec2.internal	Running
Running 2 minutes a	go		
v54bjs3c8u5r	sematext-agent-docker	r.vdamjjjrz7a3ri3prv9fjngvy	
<pre>sematext/sematext-a</pre>	<pre>gent-docker:latest</pre>	ip-172-31-6-92.ec2.internal	Running
Running 2 minutes a	go		
s7arohbeoake	sematext-agent-docker.	.9rvieyqnndgecagbuf73r9gs5	
<pre>sematext/sematext-a</pre>	<pre>gent-docker:latest</pre>	ip-172-31-35-125.ec2.internal	Running
Running 3 minutes a	go		
ixpri65xwpds	sematext-agent-docker	.mhbbunhl358chah1dmr0y6i71	
<pre>sematext/sematext-a</pre>	gent-docker:latest	ip-172-31-7-78.ec2.internal	Running
Running 4 minutes a	go		
5jvl7gnvl0te	sematext-agent-docker	.8d0qv1epqu8xop4o2f94i8j40	
<pre>sematext/sematext-a</pre>	gent-docker:latest	ip-172-31-8-4.ec2.internal	Running
Running 15 minutes	ago		
y53f20d3kknh	sematext-agent-docker	.xks3sw6qgwbcuacyypemfbxyj	
<pre>sematext/sematext-a</pre>	gent-docker:latest	ip-172-31-31-117.ec2.internal	Running
Running 15 minutes	ago		
t5w2pxy4fc9l	sematext-agent-docker	.r02ftwtp3n4m0cl7v2llw4gi8	
<pre>sematext/sematext-a</pre>	gent-docker:latest	ip-172-31-44-8.ec2.internal	Running
Running 15 minutes	ago		

Creating a MySQL Database Service on a Docker Swarm

In this section, we create a MySQL database service from which metrics, logs, and events can be collected with Sematext SCM and Logsene using the Sematext Docker Agent, which we installed. To start, run the following command to create a mysql service with 10 replicas.

```
~ $ docker service create \
> --env MYSQL_ROOT_PASSWORD='mysql'\
> --replicas 10 \
> --name mysql \
> mysql
rmy45fpa31twkyb3dowzpc74a
```

The service is created and listed in addition to the Sematext Docker agent service.

~ \$ docker se	ervice ls				
ID	NAME	MODE	REPLICAS	IMAGE	PORTS
oubjk53mpdnj	<pre>sematext-agent-docker</pre>	global	8/8	<pre>sematext/sematext-agent- docker:latest</pre>	
rmy45fpa31tw	mysql	replicated	10/10	mysql:latest	

The service tasks for the mysql service are also listed.

~ \$ docker servi	ce ps mysq	1		
ID	NAME	IMAGE	NODE	DESIRED STATE
CURRENT STATE	ERR	OR PORTS		
x8j221ws4kx2	mysql.1	mysql:latest	ip-172-31-21-138.ec2.internal	Running
Running 13 secon	ds ago			
98rbd6nwspqz	mysql.2	mysql:latest	ip-172-31-44-8.ec2.internal	Running
Running 11 secon	ds ago			
vmq0lylni8or	mysql.3	mysql:latest	ip-172-31-8-4.ec2.internal	Running
Running 24 secon	ds ago			
0vb6oda3yh3d	mysql.4	mysql:latest	ip-172-31-7-78.ec2.internal	Running
Running 23 secon	ds ago			
vdpplkyxy1uy	mysql.5	mysql:latest	ip-172-31-6-92.ec2.internal	Running
Running 23 secon	ds ago			
9ser7fwz6998	mysql.6	mysql:latest	ip-172-31-18-156.ec2.internal	Running
Running 17 secon	ds ago			
vfsfvanghns0	mysql.7	mysql:latest	ip-172-31-18-156.ec2.internal	Running
Running 17 secon	ds ago			
v71qwpvjhhzn	mysql.8	mysql:latest	ip-172-31-6-92.ec2.internal	Running
Running 23 secon	ds ago			
j7172i5ml43d	mysq1.9	mysql:latest	ip-172-31-31-117.ec2.internal	Running
Running 24 secon	ds ago			
5p5mg2wnbb0o	mysql.10	mysql:latest	ip-172-31-35-125.ec2.internal	Running
Running 20 secon	ds ago			

After the Sematext Docker agent service has been started on the Swarm and a MySQL database service has been started, both the SPM and Logsene apps start receiving data, as indicated by the Data Received column in the dashboard. See Figure 11-16.

< Monitoring All Monitoring Apps	All	Apps			Provide an Alternative International				
DockerSwarmSPM	10	• 012			search or filter using ta	8			
Docker	Туре	Application	Token [Pla	in 1	State]	Your Role	Data Received	Actions
os	=	DockerSwarmLogsene	81ac5395-xxxxxxxxxxxx-dc00c649116a	Ba	sic, 7 days, 500 MB/day	~	OWNER	2017-07-27	
Custom Metrics	*	DockerSwarmSPM	9b5552fd-xxxxxxxxxxxxxx76046d4a3413	Ba	sic SPM Docker	~	OWNER	2017-07-27	
≡ Logs									
All Logs Apps									
DockerSwarmLogsene									

Figure 11-16. DockerSwarmSPM overview

Monitoring the Docker Swarm Metrics

After the mysql service is started on the Swarm, the metrics for the service start getting loaded into the SPM – Performance Monitoring dashboard. This happens as soon as the Sematext Docker agent is installed and new metrics from a deployment become available. Graphs for different metrics—including Host CPU, Container CPU, Container Memory, Container Count, Container Memory Failed Counter, Container Swap, Container I/O Throughput, Container Network Traffic, and Container Network Errors—are displayed, as shown in Figure 11-17.



Figure 11-17. Docker Swarm SPM overview

The Docker container metrics—including Container Count, Container CPU, Container Disk, Container Memory, and Container Network—may be displayed by selecting Docker in the navigation. The Docker Container Count metrics are shown in Figure 11-18.



Figure 11-18. Docker metrics

The Docker \succ Container Network selection displays the network traffic received and transmitted, the receive rate, and the transmit rate. The OS Disk Space Used may be displayed by choosing OS \succ Disk. The metrics collection granularity may be set to auto granularity (default), by month, by week, by day, by hour, by 5 minutes, or by 1 minute. The Logs Overview may be displayed using the Logs button.

Click the Refresh Charts button to refresh the charts if they are not set to auto-refresh, which is the default.

Detailed logs are displayed using Logsene UI or Kibana 4, which we discuss in the next section.

Getting Docker Swarm Logs in Logsene

Select Logs > DockerSwarmLogsene in the margin navigation to display the logs collected by Logsene. The Log Counts, Log Events, and Filter fields are displayed, as shown in Figure 11-19. To search for logs generated by the mysql service, add "mysql" to the search field and click on the Search button. The logs generated by the mysql Docker service are displayed, including status messages such as "mysqld ready for connections". Click on the Refresh button to refresh the logs.

<						£	*	>	
2 Dashboards	mysql			×	Q 🖺	1	*	Type to filter fields	
Infrastructure	Log Counts				0	- 2		Otimestamo	
Monitoring	tk				1,910	match	es	> container_hostname	0
All Monitoring Apps	000 600							> container_id	0
DockerSwarmSPM	200							> container_name	0
Docker	ento en	P BSP Set 5	10 10 10 10 10 10 10 10 10 10 10 10 10 1	15 ¹⁰ 15 ¹⁰	10 10 10 M		and and	+ dockerEventAction	0
05								+ dockerEventFrom	0
Custom Nation	I of Events				2	_ /		+ dockerEventHost	0
custom metrics	Otimestamp	source			C	Taris		> dockerEventImageName	0
Logs	10:39:19.689	severity: info logType: docker	swarm_service_name: mysal	ip: 172.17.0.5 logSource:	mysql.2.96rbd	0		> dockerEventType	0
All Logs Apps	2m S95 ago	e7-27722:39:19.82916e2 e [Not tables container_hostname: cd	te] End of list of non-nat 339d54cf31 swarm_task_name	ively partitioned 98rbd6nwspqzrf7fryl4relp	t inage_name:			> host	
DockerSwarmLogsene		e7- 27722:30:19.0092 container_na 172.21.44.8.er2.internal_hors	ane: mysql.2.90rbd6nwspqzri	7fryl4relpt swarm_node_in	rezftirtpan4r			image_name	0
X Correlations	10:39:19.681	severity: info logType: docker	swarm_service_name: mysql	ip: 172.17.0.5 logSource:	m/sq1.2.98rbd	0		⇒ ip	0
Alerts & Events	3m 59s ago	<pre>'5.7.19' socket: '/var/run/my (GPL) container_hostname: cd</pre>	ysqld/mysqld.sock' port: 3 3339d54cf31 swarm_task_nam	386 MSQL Community Server :98rbd6nwspqzrf7fry14re1p	t inage_name:			+ logSource	0
		07- 27722:39:19.881Z container_na	me: mysql.2.98rbd6nuspqzri	7/ryl4relpt swarm_node_in	: re2ftitp3n4r			+ logType	0
A Team	14-30-10 001	severity: info Instant docker	terre service care: and	in: 172 17 0 5 lostourse	muni 1 stold	0		+ logsene_orig_log	
- lean	3m 55s ago	07-27722:39:19.8005852 0 [Not	te] mysqld: ready for	name: Districtions registrates	Acales (mana)	×.		> logsene_original_type	0
Account		07- 27722:39:19.8812 container_na	me: msql.2.98rbd6rwspgzrf	7fryl4relpt swarm_node_i	: réžftutp3n4e			> message	_

Figure 11-19. Logs generated by the mysql Docker Service

The Logsene collects all the Docker events, such as the Docker pull event for the mysql:latest image, as shown in Figure 11-20.



Figure 11-20. Logs for Docker event for mysql image pull

Logs for another Docker event, a volume mount, are shown in Figure 11-21.





Figure 11-21. Logs for Docker event volume mount

Summary

This chapter discussed continuous logging and monitoring of a Docker Swarm with Sematext SPM performance monitoring and Logsene log management. First, you learned how to create a SPM app and a Logsene app. Then you installed a Sematext agent service on each of the Swarm nodes and monitored the metrics and events in a SPM dashboard. You also learned how to monitor the logs in the Logsene UI or a Kibana 4 dashboard. The next chapter discusses load balancing in a Docker Swarm.

CHAPTER 12

Load Balancing

A Docker Swarm mode service provides a distributed application that may be scaled across a cluster of nodes. Swarm mode provides internal load balancing among the different services in the Swarm based on the DNS name of a service. Swarm mode also provides ingress load balancing among a service's different tasks if the service is published on a host port. Additionally, service tasks may be scheduled on specific nodes using placement constraints.

Service Discovery

A Swarm has a DNS server embedded in it. Service discovery is based on the DNS name. Swarm manager assigns each service in the Swarm a unique DNS name entry. Swarm manager uses internal load balancing to distribute requests for the different services in the Swarm based on the DNS name for a service.

Custom Scheduling

Service replicas are scheduled on the nodes in a Swarm using the *spread* scheduling strategy by default. A user may configure placement constraints for a service so that replicas are scheduled on specific nodes. Scheduling using constraints is discussed in Chapter 6.

Ingress Load Balancing

By default, each service that's exposed on a published port for external access is added to the ingress overlay network. A user may specify any available port to expose a service by using the --publish, or -p, option. The syntax for the --publish (-p) option is --publish <PublishedPort>:<TargetPort> in which the <PublishedPort> variable is for the published port on the host and the <TargetPort> variable is for the container port. If the --publish, -p option does not specify a <PublishedPort> port to publish the service on the Swarm, the manager automatically exposes the service on a published port chosen from the range 30000-32767.

The Problem

Ingress load balancing is for distributing the load among the service tasks and is used even if a Swarm consists of a single node. Ingress load balancing for a multi-node Swarm is illustrated in Figure 12-1. A client may access any node in the Swarm, whether the node has a service task scheduled or not, and the client request is forwarded to one of the service tasks using ingress load balancing.

CHAPTER 12 LOAD BALANCING



Figure 12-1. Ingress load balancing

A single client accesses a single node and, as a result, the Swarm is under-utilized in terms of distributing external client load across the Swarm nodes. The client load is not balanced across the Swarm nodes. A single node does not provide any fault tolerance. If the node fails, the service becomes unavailable to an external client accessing the service at the node.

The Solution

An AWS Elastic Load Balancer (ELB) is used to distribute client load across multiple EC2 instances. When used for Docker Swarm mode an AWS Elastic Load Balancer distributes client load across the different EC2 instances, which are hosting the Swarm nodes. The external load balancer accesses (listens to) the Swarm on each EC2 instance at the published ports for the services running in the Swarm using LB listeners. Each LB listener has an LB port mapped to an instance port (a published port for a service) on each EC2 instance. An ELB on a Swarm is illustrated in Figure 12-2.



Figure 12-2. External load balancer

As a client is not accessing the service at a single host even if a single node goes down or becomes unavailable, the Swarm does not become unavailable as the external load balancer directs the client request to a different node in the Swarm. Even when all the nodes are available, the client traffic is distributed among the different nodes. As an example, a client could be being served from one node at a particular time and from a different node shortly thereafter. Thus, an external load balancer serves two functions: load balancing and fault tolerance. Additionally the cloud provider on which a Swarm is hosted may provide additional features such as a secure and elastic external load balancing. Elastic load balancing, as provided by AWS Elastic Load Balancer, scales the request handling capacity based on the client traffic.

This chapter discusses load balancing with a user-created Swarm on CoreOS. It also discusses the automatically provisioned elastic load balancer on Docker for AWS managed services.

Setting the Environment

Start three CoreOS instances—one for the manager node and two for the worker nodes—as shown in Figure 12-3. Obtain the public IP address of the manager instance from the EC2 dashboard, as shown in Figure 12-3.

CHAPTER 12 LOAD BALANCING

Q, Ke	ey Name : co	reos 🔿 Ad	d filter							0	K <	1 to 3 of 3	>	×
N	ame	~	Instance ID	▲ Ir	stance Type 👻	Availability Zone 👻	Instance State +	Statu	s Checks 👻	Alarm State	IS	Public DNS	(IPv4)
C	oreOSManag	er	i-0a9e7ea5e9dfa6e	04 12	micro	us-east-1b	running	O 2/	/2 checks	None	10	ec2-52-91-2	12-27	co
C	oreOSWorker	r	i-0c1067b1fbc0310	Of t2	micro	us-east-1b	running	2/	/2 checks	None	10	ec2-54-162-	11-21	3.c
C	oreOSWorker	r	i-Odd912caa9fa1b1	of t2	micro	us-east-1b	running	2/	2 checks	None	2	ec2-34-201-	118-1	46.
istanc	e: I-0a9e76	ea5e9dfa6e	04 (CoreOSManag	ler)	Public DNS: ec:	2-52-91-212-27.comp	ute-1.amazonaws.c	om					8	
nstanc Descri	e: i-0a9e7	ea5e9dfa6e itatus Check	04 (CoreOSManag s Monitoring	ier) Tags	Public DNS: ec	2-52-91-212-27.comp	ute-1.amazonaws.c	om					8	-
nstanc Descri	iption S	ea5e9dfa6e Status Check Instance I	04 (CoreOSManag s Monitoring D i-0a9e7ea5e9d	Tags fa6e04	Public DNS: ec:	2-52-91-212-27.comp	ute-1.amazonaws.c	∞om ∿4)	ec2-52-91-212 1.amazonaws.	-27.compute-		8	8	-
Descri	iption s	ea5e9dfa6e itatus Check Instance I Instance stal	04 (CoreOSManag s Monitoring D i-0a9e7ea5e9d e running	Tags fa6e04	Public DNS: ec:	2-52-91-212-27.comp	ute-1.amazonaws.c Public DNS (IP	*v4) (ec2-52-91-212 1.amazonaws 52.91.212.27	-27.compute- com			8	•
Descri	iption S	ea5e9dfa6ei itatus Check Instance I Instance stai Instance typ	04 (CoreO SManag S Monitoring D i-0a9e7ea5e9d e running e t2.micro	Tags	Public DNS: ec:	2-52-91-212-27.comp	ute-1.amazonaws.c Public DNS (IP QIPv4 Public IPv6	>v4) 0 cIP €	ec2-52-91-212 1.amazonaws 52.91.212.27 -	-27.compute- com		-	8	-
Descri	iption S	ea5e9dfa6el Itatus Check Instance I Instance stal Instance typ Elastic IF	04 (CoreO SManag S Monitoring D i-Da9e7ea5e9d e running e t2.micro	Tags	Public DNS: ec/	2-52-91-212-27.comp	ute-1.amazonaws.c Public DNS (IP PV4 Public IPv6 Private D	v4) cIP IPs iNS i	ec2-52-91-212 1.amazonaws. 52.91.212.27 - ip-10-0-0-226.0	-27.compute- com		8		-
Descri	e: I-0a9e70	ea5e9dfa6ei itatus Check Instance I Instance stal Instance typ Elastic IF vailability zon	A (Core O SManages) Monitoring D i-0a9e7ea5e9d running t 2.micro s us-east-1b	fa6e04	Public DNS: ec:	2-52-91-212-27.comp	Public DNS (IP Public DNS (IP PV4 Public IPv6 Private D Private D	v4) (IP (IPs (IPs (IPs (ec2-52-91-212 1.amazonaws. 52.91.212.27 - ip-10-0-0-226.0 10.0.0.226	-27.compute- com ec2.internal		8	8	-

Figure 12-3. CoreOS instances on EC2 for a manager and two worker nodes

SSH login into the manager node to initiate the Swarm mode. Initializing a Swarm mode on CoreOS and joining worker nodes to the Swarm is discussed in Chapter 2. Copy the docker swarm join command output to join the worker nodes to the Swarm. List the Swarm nodes with the docker node 1s command.

```
core@ip-10-0-0-226 ~ $ docker node lsIDHOSTNAMESTATUSAVAILABILITYMANAGER STATUS9iqh5tg7hxy8u43tlifd1ri0qip-10-0-0-203.ec2.internalReadyActiveaoe1b2623qj03852mrc5cax97ip-10-0-0-198.ec2.internalReadyActivedsyo3b6553ueishozhfb1apad *ip-10-0-0-226.ec2.internalReadyActiveLeader
```

Creating a Hello World Service

Next, create a hello world service with the docker service create command. Expose the service at port 8080 using the --publish option. The syntax to publish a service using --publish or -p is as follows.

```
docker service create \
    --name <SERVICE-NAME> \
    --publish <PUBLISHED-PORT>:<TARGET-PORT> \
    <IMAGE>
```

The <PUBLISHED-PORT> is the port exposed on the hosts and the <TARGET-PORT> is the port on which the Docker container exposes the service. Using the tutum/hello-world Docker image, <PUBLISHED-PORT> as 8080, <TARGET-PORT> as 80, and <SERVICE-NAME> as hello-world, run the following command to create the service.

```
core@ip-10-0-0-226 ~ $ docker service create \
> --name hello-world \
> --publish 8080:80 \
> --replicas 3 \
> tutum/hello-world
Ogk3wom7z91fpm509e6optmb5
```

The service is added to the ingress overlay network and the service is exposed at each node on the Swarm, whether a service task is running on the node or not. The hello-world service lists 3/3 replicas.

core@ip-10-0-	0-226 ~ \$ doc	ker servic	e ls	
ID	NAME	REPLICAS	IMAGE	COMMAND
0gk3wom7z91f	hello-world	3/3	tutum/hello-world	

List the service tasks using the docker service ps hello-world command and the three tasks are listed as scheduled, one on each node.

core@ip-10-0-0-226 ~ \$ docker service ps hello-world NODE NAME IMAGE ID DESIRED STATE CURRENT STATE ERROR di5oilh96jmr6fd5haevkkt2 hello-world.1 tutum/hello-world ip-10-0-0-198.ec2.internal Running Running 24 seconds ago 5g5d075yib2td8466mh7c01cz hello-world.2 tutum/hello-world ip-10-0-0-226.ec2.internal Running Running 24 seconds ago 5saarf4ngju3xr7uh7ninho0o hello-world.3 tutum/hello-world ip-10-0-0-203.ec2.internal Running Running 23 seconds ago

One Docker container is running on the manager node.

core@ip-10-0-0-	-226 ~ \$ docker ps		
CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
b73cbcd0c37e	<pre>tutum/hello-world:latest</pre>	"/bin/sh -c 'php-fpm "	34 seconds ago
Up 32 seconds	80/tcp	hello-world.2.5g5d075yib2te	d8466mh7c01cz

One Docker container is running on one of the worker nodes.

core@ip-10-0-0-	-198 ~ \$ docker ps		
CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
8bf11f2df213	<pre>tutum/hello-world:latest</pre>	"/bin/sh -c 'php-fpm "	38 seconds ago
Up 36 seconds	80/tcp	hello-world.1.di5oilh96j	mr6fd5haevkkkt2

And the third Docker container is running on the other worker node.

core@ip-10-0-0-	203 ~ \$ docker ps		
CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
a461bfc8d4f9	<pre>tutum/hello-world:latest</pre>	"/bin/sh -c 'php-fpm "	40 seconds ago
Up 38 seconds	80/tcp	hello-world.3.5saarf4ngju3	xr7uh7ninho0o

Invoking the Hello World Service

Without an external load balancer, an ingress connection may be made at each of the nodes at the published port. To invoke the service at the manager node, obtain the public DNS of the Swarm manager instance from the EC2 console, as shown in Figure 12-3.

Invoke the service in a web browser at the <PublicDNS>:<PublishedPort> URL, as shown in Figure 12-4.



Figure 12-4. Invoking the service in a browser

Similarly, to invoke the service at a worker node, obtain the public DNS of the worker instance from the EC2 console and invoke the service in a web browser at the <PublicDNS>:<PublishedPort> URL, as shown in Figure 12-5.



Figure 12-5. Invoking the service at a worker node

CHAPTER 12 LOAD BALANCING

Similarly, to invoke the service at the other worker node, obtain the public DNS of the worker instance from the EC2 console and invoke the service in a web browser at the <PublicDNS>:<PublishedPort> URL, as shown in Figure 12-6.



Figure 12-6. Invoking the service at the other worker node

While the external AWS Elastic Load Balancer distributes the load among the EC2 instances, the ingress load balancer distributes the load among the service tasks. In the preceding example, the same service task is invoked when the service is invoked at the Swarm manager instance and at a Swarm worker instance, as indicated by the same hostname (Figures 12-4 and 12-6). This demonstrates the ingress load balancing.

A different service task could get invoked if the service is invoked at the same host. As an example, invoke the service at the Swarm manager instance again. A different service task is served, as indicated by a different hostname in Figure 12-7. This is in comparison to the hostname served earlier in Figure 12-4, again demonstrating the ingress load balancing.



Figure 12-7. Different hostname served when invoking the service at the manager node again

Creating an External Elastic Load Balancer

In this section, we create an external elastic load balancer on the AWS cloud. Click on Load Balancers in the EC2 dashboard. Then click on Create Load Balancer to create a new load balancer, as shown in Figure 12-8.



Figure 12-8. Creating a new load balancer

CHAPTER 12 LOAD BALANCING

AWS Elastic Load Balancing offers two types of load balancers—classic load balancers and application load balancers. The classic load balancer routes traffic based on either application or network level information whereas the application load balancer routes traffic based on advanced application-level information. The classic load balancer should suffice for most simple load balancing of traffic to multiple EC2 instances and is the one we use for Docker Swarm instances. Select the Classic Load Balancer and then click on Continue, as shown in Figure 12-9.



Figure 12-9. Selecting the classic load balancer option

In the Define Load Balancer dialog, specify a load balancer name (HelloWorldLoadBalancer) and select a VPC to create the load balancer in, as shown in Figure 12-10. The VPC must exist prior to creating the load balancer and must be where the EC2 instances to be load balanced are created. The load balancer protocol is HTTP and so is the instance protocol, by default. Keeping the default setting of HTTP protocol, specify the load balancer port and the instance port as 8080, because the Hello World service is exposed at port 8080.

1. Define Load Balancer 2. Assign 5	Security Groups 3. Configure Security Settings	4. Configure Health Check 5. Add EC	2 Instances 6. Add Tags	7. Review	
Step 1: Define Load B	alancer				
Basic Configuration					í
This wizard will walk you through setti configure ports and protocols for your standard web server on port 80.	ing up a new load balancer. Begin by giving yo load balancer. Traffic from your clients can be	ur new load balancer a unique name so routed from any load balancer port to a	that you can identify it from ny port on your EC2 instan	n other load balancers you r ces. By default, we've config	night create. You will also need to jured your load balancer with a
Load Balancer name:	HelloWorldLoadBalancer				
Create LB Inside:	vpc-18c6a261 (10.0.0.0/24) redshift-vpc	•			
Create an internal load balancer:	(what's this?)				
Listener Configuration:					
Load Balancer Protocol	Load Balancer Port	Instance Protoco		Instance Port	
HTTP	8080	HTTP	•	8080	0
Add					
Select Subnets					
You will need to select a Subnet for ea different Availability Zones to provide	ach Availability Zone where you wish traffic to th higher availability for your load balancer.	be routed by your load balancer. If you	have instances in only one a	Availability Zone, please sel	ect at least two Subnets in
				Cancel	Next: Assign Security Groups

Figure 12-10. Selecting the load balancer protocol

In the Select Subnets tab, click on one or more subnets listed in the Available Subnets table. The subnets are added to the selected subnets, as shown in Figure 12-11. Click on Next. To provide high availability, select at least two subnets in different availability zones.

A sea of the law sea the state of	- ABL - BA				
Load Balancer Protocol	Load Balancer Port	Instance Pr	rotocol	Instance Port	
HTTP *	8080	HTTP	٣	8080	۲
Add					
Select Subnets					
You will need to select a Subnet for different Availability Zones to provid	each Availability Zone where you wish te higher availability for your load baland	traffic to be routed by your load balancer. cer.	If you have instances in only one Avai	ability Zone, please select at least	two Subnets in
VPC vpc-18c6a261 (10.0.0.0/24)	redshift-vpc				
Available subnets					
Actions	Availability Zone	Subnet ID	Subnet CIDR	Name	
Selected subnets				Mana	
Selected subnets Actions	Availability Zone	Subnet ID	Subnet CIDR	Name	
Selected subnets Actions	Availability Zone	Subnet ID subnet-4531d221	Subnet CIDR 10.0.0/25	subnet-1	
Actions	Avallability Zone us-east-1a us-east-1b	Subnet ID subnet-4531d221 subnet-ecdd41c0	Subnet CIDR 10.0.0.0/25 10.0.0.128/25	subnet-1 subnet-2	

Figure 12-11. Selecting subnets

In the Assign Security Groups tab, select Create a New Security Group, as shown in Figure 12-12. In Type, select Custom TCP Rule. Choose the TCP protocol and the port range as 8080. Select Anywhere for the source and its value as 0.0.0/0. Click on Next.

1. Define Load Balancer 2. A Step 2: Assign Ser You have selected the option of This can be changed at any tim	ssign Security Groups 2. Configure Security Settings Curity Groups (having your Elastic Load Balancer inside of a VPC, withe)	4. Configure Health Check 5. Add EC2 Instances	6. Add Tage 7. Review	id balancer.
Assign a security group:	Create a new security group			
Security group name:	quick-create-1			
Description:	quick-create-1 created on Thursday, July 27, 2017 a	t 12:23:46 PM UTC-		
Туре ()	Protocol (j)	Port Range (1)	Source (j)	
Custom TCP Rule *	TCP	8080	Anywhere • 0.0.0.0/0	8
Add Rule				
			Cancel Previous Next: Configure Securit	y Settings

Figure 12-12. Assigning security groups

Click on Next in Configure Security Settings, as we have not used the HTTPS or the SSL protocol. In the Configure Health Check tab, select HTTP for the ping protocol and 8080 for the ping port. Specify the ping path as /, as shown in Figure 12-13. Keep the defaults as is in the Advanced Details area and then click on Next.

CHAPTER 12 LOAD BALANCING

Step 4: Configure Health Check

Vour load balancer will automatically perform health checks on your EC2 instances and only route traffic to instances that pass the health check. If an instance fails the health check, it is automatically removed from the load balancer. Customize the health check to meet your specific needs.

Ping Proto Ping I Ping F	Port Path	HTTP 8080	•					
Advanced Details								
Response Timeout	1	5	seconds					
Interval	1	30	seconds					
Unhealthy threshold	1	2	*					
Healthy threshold	1	10	•					
						Cancel	Previous	Next: Add EC2 Instances

Figure 12-13. Configuring a health check

Select the three Swarm instances listed, as shown in Figure 12-14. Also select Enable Cross-Zone Load Balancing, which distributes traffic evenly across all backend instances in all availability zones. Click on Next.

VPC v	00-18068261 (10	0.0.0.0/24) redshift-vpc							
•	Instance *	Name	*	State -	Security groups	Zone		Subnet ID	 Subnet CIDR
	i-0c1067b1	CoreOSWorker		running	Container Linux by CoreOS -Stable1409-7-0-AutogenBy	us-east-1b	\$	subnet-ecdd41c0	10.0.0.128/25
	i-0a9e7ea5	CoreOSManager		running	Container Linux by CoreOS -Stable1409-7-0-AutogenBy	us-east-1b	5	subnet-ecdd41c0	10.0.0.128/25
	i-0dd912ca	CoreOSWorker		running	Container Linux by CoreOS -Stable1409-7-0-AutogenBy	us-east-1b	s	subnet-ecdd41c0	10.0.0.128/25
Availa 3 insta	bility Zone Dist	tribution 1b							
Availa 3 insta 🕑 Ena	bility Zone Dist nces in us-east- ble Cross-Zone	tribution 1b 2 Load Balancing (j)							
Availa 3 insta 2 Ena 2 Ena	bility Zone Dist inces in us-east- ble Cross-Zone ble Connection	tribution 1b Load Balancing (j) Draining (j) 300 seconds							

Figure 12-14. Adding EC2 instances

In the Add Tags tab, no tags need to be added. In the Review tab, click on Create, as shown in Figure 12-15. As indicated, the load balancer is an Internet-facing type.

Step 7: Review		
▼ Define Load Balancer		Edit load balancer definition
Load Balancer name: Scheme: Port Configuration:	HelloWorldLoadBatancer internet-facing 8060 (HTTP) forwarding to 8080 (HTTP)	
▼ Configure Health Check		Edit health check
Ping Target: Timeout: Interval: Unhealthy threshold: Healthy threshold:	HTTP:8080' 5 seconds 30 seconds 2 10	
 Add EC2 Instances 		Edit instances
Cross-Zone Load Balancing: Connection Draining: Instances:	Enabled Enabled, 300 seconds i-0dd912caa9fa1b1cf (CoreOSWorker), i-0a9e7ea5e9dfa6e04 (CoreOSManager), i-0c1067b1fbc03100f (CoreOSWorker)	
 VPC Information 		Edit subnets
		Cancel Previous Create

Figure 12-15. Review your settings then create the load balancer

A load balancer is created, as shown in Figure 12-16.

Load Balancer Creation Status

0	Successfully created load balancer Load balancer HelloWorldLoadBalancer was successfully created. Nole: It may take a few minutes for your instances to become active in the new load balancer.

Figure 12-16. The load balancer has been created

CHAPTER 12 LOAD BALANCING

Obtain the DNS name of the load balancer from the EC2 console, as shown in Figure 12-17. Initially, the status will be "0 of 3 instances in service" because the registration is still in progress.

×		< 1 to 2 of 2 > >
✓ DNS name	- VPC ID	- Availability Zones - Ty
PS1IGIG5GT Docker-ExternalLoa-10WPS	vpc-f06f1a89	us-east-1a, us-east-1b, cl
HelloWorldLoadBalancer-13	vpc-18c6a261	us-east-1a, us-east-1b cla
HelloWorldLoadBalancer	Creation time: July 27, 20	017 at 12:25:19 PM UTC-7
HelloWorldLoadBalancer-1388543093.us-east-	Hosted zone: Z35SXDO	TRQ7X7K
1.elb.amazonaws.com (A Record)	Status: 0 of 3 inst	ances in service
internet-facing	VPC: vpc-18c6a	261

Figure 12-17. Obtaining the DNS name of the load balancer

After a while, the status should become "3 of 3 instances in service" and all the instance should be InService, as shown in Figure 12-18.

Filter: Q Sea	rch		×						< < 1	to 2 of 2	>	Ы
Name		*	DNS name	Ţ	State	*	VPC ID	+	Availability Zo	ones	*	Туре
Docker-Ex	ternalLoa-10W	PS1IGIG5GT	Docker-Externa	alLoa-10WPS			vpc-f06f1a89		us-east-1a, us-	east-1b		class
HelloWorld	dLoadBalancer		HelloWorldLoa	dBalancer-13			vpc-18c6a261		us-east-1a, us-	east-1b	4	class
Description	Instances	Health Check	Listeners	Monitoring	Tags							
Description Connection D	Instances raining: Enab	Health Check	Listeners (Edit)	Monitoring	Tags							_
Description Connection D Edit Instanc	Instances graining: Enab ges	Health Check	Edit)	Monitoring	Tags							
Description Connection D Edit Instance Instance ID	Instances Irraining: Enab	Health Check Med, 300 seconds (Name	(Edit)	Monitoring	Tags	Status	\$	Actions				
Description Connection D Edit Instance Instance ID i-0c1067b1fbc	Instances Fraining: Enab	Health Check iled, 300 seconds (Name CoreOSN	(Edit) Worker	Availab us-east-1	Tags Ility Zone	Status	(j)	Actions Remove from Lo	ad Balancer			
Description Connection D Edit Instance Instance ID i-0c1067b1fbc i-0a9e7ea5e9	Instances raining: Enab es :03100f kdfa6e04	Health Check iled, 300 seconds (Name CoreOSI CoreOSI	Listeners (Edit) Worker Manager	Availabi us-east-1 us-east-1	Tags Ility Zone b b	Status InService InService) () ()	Actions Remove from Los Remove from Los	ad Balancer ad Balancer			

Figure 12-18. Status indicates three of three instances InService

The Hello World service may be invoked from the <DNSname>:<LoadBalancerPort> URL in a web browser, as shown in Figure 12-19.



Figure 12-19. Invoking the Hello World service

The external elastic load balancer balances the load among the EC2 instances in the Swarm. Because the ingress load balancer balances the load among the different service tasks, a different service task could get invoked if the service is invoked at the ELB DNS name again, as shown in Figure 12-20.



Figure 12-20. Different service task served

Load Balancing in Docker for AWS

While an external elastic load balancer had to be created when creating a Docker Swarm using the command line (by first initiating the Swarm mode and subsequently joining the worker nodes to the Swarm), the Docker for AWS managed service, which was introduced in Chapter 3, automatically creates an elastic load balancer.

Create a Swarm (a Swarm created earlier may be updated) with three manager nodes and five worker nodes using Docker for AWS, as shown in Figure 12-21. An external elastic load balancer is created as one of the Swarm resources, as listed in the Resources tab in Figure 12-21.

Create Star	A A	tions -	Design t	emplate							C O		
Filter: Acti	ve - By Sta	ck Name	besign	emplate							Showing 1 stack		
Stack	Name	Cre	ated Time		Statu	s		Description					
Docke	Docker 2017-07-27 11:30:37 UTC-0700				00 UPD	UPDATE_COMPLETE Docker C			Docker CE for AWS 17.06.0-ce (17.06.0-ce-aws2)				
Overview	Outputs	Resources	Events	Template	Parameters	Tags	Stack Policy	Change Sets			880		
Logical ID		Physical	D				Туре		Status	Status Reason			
AZInfo		2017/07/2 a30	7/ <mark>[\$LATEST</mark>	']2162563a0b8	949e9adca2dbe	5b4f1	Custom::AZInfo		CREATE_COMPLETE				
AZInfoFund	tion:	Docker-A	ZinfoFunctio	n-16Y219QN	LO06		AWS::Lambda::F	unction	CREATE_COMPLETE				
AttachGate	way	Docke-Att	ac-1KP10X	E64DPD4			AWS.:EC2::VPC	GatewayAttach	CREATE_COMPLETE				
CloudstorE	BSPolicy	Docke-Clo	u-1GKBPB	2MOUKUE			AWS::IAM::Polic	y	CREATE_COMPLETE				
DockerLog	Group	Docker-Ig					AWS::Logs::Log	Group	CREATE_COMPLETE				
DynDBPoli	cies	Docke-Dy	nD-QB8AQ	R6HT4YY			AWS: IAM Polic	y	CREATE_COMPLETE				
DynDBWor	kerPolicies	Docke-Dy	nD-179NNV	27QWUYC			AWS::IAM::Polic	y.	CREATE_COMPLETE				
ExternalLog	adBalancer	Docker-Ex	ternalLoa-1	OWPS1IGIG5	GT		AWS::ElasticLoa	dBalancing::Lo	CREATE_COMPLETE				
ExternalLoa	adBalancerSG	sg-18797e	969				AWS::EC2::Secu	rityGroup	CREATE_COMPLETE				

Figure 12-21. CloudFormation stack for a Docker Swarm

An Internet-facing Elastic Load Balancer is created, as shown in Figure 12-22. The public DNS for the load balancer may be used to access the Swarm, as discussed later.

Filter: Q, S	earch		×					K < 1 to 1 of 1 >	×
Name		-	DNS name	~	State	- VF	PC ID	- Availability Zones -	Тур
Docker-	ExternalLoa-10W	PS1IGIG5GT	Docker-Externa	alLoa-10WPS		vp	c-f06f1a89	us-east-1a, us-east-1b,	class
1	12000 A. 10000								
Load balance	r: Docker-Exte	rnalLoa-10WPS1	IGIG5GT					88	
Description	Instances	Health Check	Listeners	Monitoring	Tags				
Basic Cor	nfiguration								
	Name:	Docker-External	Loa-10WPS1	GIG5GT		Creation time:	July 27, 2017 at	11:31:49 AM UTC-7	
	* DNS name:	Docker-External	Loa-10WPS1	GIG5GT-		Hosted zone:	Z35SXDOTRQ7	хлк	
		449932458.us-e Record)	ast-1.elb.amaz	zonaws.com (A		Status:	8 of 8 instances	in service	
	Scheme:	internet-facing				VPC:	vpc-f06f1a89		
	ability Zones	subnet-2542230	9 - us-east-1b	65					

Figure 12-22. Load balancer for the Swarm created with Docker for AWS

Select the Instances tab. All the instances in the Swarm, manager or worker, are listed. All the instances should be InService, as shown in Figure 12-23.

CHAPTER 12 LOAD BALANCING

ilter: Q, Search	×			< < 1 to 1 of 1 > >
Name	- DNS name	- State	- VPC ID	- Availability Zones - Type
Docker-ExternalLoa-10WP	S1IGIG5GT Docker-Extern	alLoa-10WPS	vpc-f06f1a89	us-east-1a, us-east-1b, class
Description Instances	Health Check Listeners	Monitoring Tags		
Connection Draining: Disabl	led (Edit)			
(- - - - - - - - - -				
Edit Instances				
Edit Instances	Name	Availability Zone	Status 🔓	Actions
Edit Instances Instance ID i-01b4f402d0a425099	Name Docker-Manager	Availability Zone	Status by	Actions Remove from Load Balancer
Edit Instances Instance ID i-01b4f402d0a425099 i-015d1c3f6e46fd811	Name Docker-Manager Docker-Manager	Availability Zone us-east-1b us-east-1a	Status by InService ()	Actions Remove from Load Balancer Remove from Load Balancer
Edit Instances Instance ID i-01b4f402d0a425099 i-015d1c3f6e46fd811 i-066ca22a93404261d	Name Docker-Manager Docker-Manager Docker-worker	Availability Zone us-east-1b us-east-1a us-east-1a	Status InService () InService () InService ()	Actions Remove from Load Balancer Remove from Load Balancer Remove from Load Balancer
Edit Instances Instance ID I-01b4f402d0a425099 I-015d1c3f6e46fd811 I-066ca22a93304261d I-0ac60724422253013	Name Docker-Manager Docker-Manager Docker-worker Docker-worker	Availability Zone us-east-1b us-east-1a us-east-1a us-east-1c	Status InService () InService () InService ()	Actions Remove from Load Balancer
Edit Instances Instance ID I-01b4f402d0a425099 I-015d1c3f6e46fd811 I-066ea22a93404251d I-066ea22a93404251d I-0ac60724422253013 I-0ea799c5a529bd0f6	Name Docker-Manager Docker-Manager Docker-worker Docker-worker Docker-worker	Availability Zone us-east-1b us-east-1a us-east-1a us-east-1c us-east-1c	Status InService () InService () InService () InService ()	Actions Remove from Load Balancer
Edit Instances	Name Docker-Manager Docker-Manager Docker-worker Docker-worker Docker-worker Docker-Manager	Availability Zone us-east-1b us-east-1a us-east-1a us-east-1c us-east-1c us-east-1c	Status InService () InService () InService () InService () InService ()	Actions Remove from Load Balancer
Edit Instances	Name Docker-Manager Docker-Manager Docker-worker Docker-worker Docker-worker Docker-worker Docker-worker	Availability Zone us-east-1b us-east-1a us-east-1a us-east-1c us-east-1c us-east-1c us-east-1c us-east-1b	Status InService () InService () InService () InService () InService ()	Actions Remove from Load Balancer

Figure 12-23. Instances status is InService

Update the load balancer listeners in the Listeners tab to add/modify a listener with a load balancer port set to 8080 and an instance port set to 8080, which is the published port for the Hello World service we create, as shown in Figure 12-24.

< < 1 to 1 of 1 > >					×		arch	Filter: Q, Sea
- Availability Zones - Type	- VPC ID		- State		DNS name	*		Name
us-east-1a, us-east-1b, classi	vpc-f06f1a89		•	alLoa-10WPS	Docker-Externa	PS1IGIG5GT	xternalLoa-10W	Docker-E
880			Tags	Monitoring alancer:	Listeners	Health Check	Instances	Description
	SSL Certificate	Cipher	Instance Port	ce Protocol	Port Instan	Load Balancer	cer Protocol	Load Balan
	N/A	N/A	8080		TCP	8080		TCP

Figure 12-24. The Listeners tab

Obtain the public IP address of one of the manager nodes from the EC2 console. SSH login to the manager node.

```
[root@localhost ~]# ssh -i "docker.pem" docker@34.205.43.53
Welcome to Docker!
```

List the Swarm nodes.

~ \$ docker node ls					
ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER	STATUS
8d0qv1epqu8xop4o2f94i8j40	ip-172-31-8-4.ec2.internal	Ready	Active		
8eckb0twpbuoslfr58lbibplh	ip-172-31-32-133.ec2.internal	Ready	Active		
b6f18h4f3o44gkf5dhkzavoy3	ip-172-31-2-148.ec2.internal	Ready	Active		
k9nl2zcmjzobbqu5c5bkd829g	ip-172-31-21-41.ec2.internal	Ready	Active		
pOd70jwh5vpjwximc1cpjfjkp *	ip-172-31-1-130.ec2.internal	Ready	Active	Leader	
r02ftwtp3n4m0cl7v2llw4gi8	ip-172-31-44-8.ec2.internal	Ready	Active		
rd8d0kksuts3aa07orhgkri3i	ip-172-31-41-86.ec2.internal	Ready	Active	Reachab]	e
xks3sw6qgwbcuacyypemfbxyj	ip-172-31-31-117.ec2.internal	Ready	Active	Reachab]	le

Create a Hello World service and expose the service at port 8080 (published port).

```
~ $ docker service create \
> --name hello-world \
> --publish 8080:80 \
> --replicas 10 \
> tutum/hello-world
n4hmfognhjrasf5nhukr55krb
```

Service tasks are scheduled across the Swarm.

~ \$ docker servio	ce ps hello-world		
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
y1fetn3kpwwn	hello-world.1	<pre>tutum/hello-world:latest</pre>	ip-172-31-2-148.ec2.internal
Running	Running 15 sec	conds ago	
5i15zl9dickd	hello-world.2	<pre>tutum/hello-world:latest</pre>	ip-172-31-44-8.ec2.internal
Running	Running 17 sec	conds ago	
k9glaavn0gzg	hello-world.3	<pre>tutum/hello-world:latest</pre>	ip-172-31-8-4.ec2.internal
Running	Running 17 sec	conds ago	
n83f89ijlokn	hello-world.4	<pre>tutum/hello-world:latest</pre>	ip-172-31-41-86.ec2.internal
Running	Running 17 sec	conds ago	
nelf275h9tp1	hello-world.5	<pre>tutum/hello-world:latest</pre>	ip-172-31-8-4.ec2.internal
Running	Running 16 sec	conds ago	
w4c8zcvlq5v7	hello-world.6	<pre>tutum/hello-world:latest</pre>	ip-172-31-32-133.ec2.internal
Running	Running 17 sec	conds ago	
b5qvbbgkrpd5	hello-world.7	<pre>tutum/hello-world:latest</pre>	ip-172-31-21-41.ec2.internal
Running	Running 16 sec	conds ago	
qlm8dt9fuv92	hello-world.8	tutum/hello-world:latest	ip-172-31-31-117.ec2.internal
Running	Running 17 sec	conds ago	
t3tenhpahh7g	hello-world.9	tutum/hello-world:latest	ip-172-31-44-8.ec2.internal
Running	Running 17 sec	conds ago	
up64ekxqeftk	hello-world.10	<pre>tutum/hello-world:latest</pre>	ip-172-31-1-130.ec2.internal
Running	Running 17 sec	conds ago	

The hello-world service may be created without explicitly specifying a published port.

```
~ $ docker service create \
```

- > --name hello-world \
- > --publish 80 \
- > --replicas 3 \
- > tutum/hello-world

The Swarm manager automatically assigns a published port in the range 30000-32767; the default being port 30000 if it's available. The listener in the load balancer for the Docker for AWS Swarm may need to be modified to add a mapping for the LoadBalancerPort:ServiceInstancePort, such as 30000:30000.

Obtain the public DNS for the elastic load balancer, which gets created automatically, as shown in Figure 12-25.

Filter: Q, Sear	rch		×					< < 1 to 1 of 1 > >	
Name		*	DNS name	÷	State	~ VF	PC ID	- Availability Zones - Type	
Docker-ExternalLoa-10WPS1IGIG5GT Docker-ExternalLoa-10WPS				vpc-f06f1a89		us-east-1a, us-east-1b, classi			
.oad balancer:	Docker-Exte	rnalLoa-10WPS1	GIG5GT					880	
Description	Instances	Health Check	Listeners	Monitoring	Tags				
Basic Confi	guration								
	Name:	Docker-ExternalLoa-10WPS1IGIG5GT			Creation time:	July 27, 2017 at 11:31:49 AM UTC-7			
* DNS name: Docker-E: 44993245		Docker-ExternalL	ternalLoa-10WPS1IGIG5GT-			Hosted zone:	Z35SXDOTRQ7X7K		
		449932458.us-east-1.elb.amazonaws.com (A			Status:	8 of 8 instances in service			
	Scheme:	internet-facing			VPC:	vpc-f06f1a89			
Availabi	lity Zones:	subnet-2542230 subnet-72608810 subnet-d67b209	9 - us-east-1b, 6 - us-east-1a, e - us-east-1c						

Figure 12-25. Obtaining the public DNS of the ELB

Access the service at <PublicDNS>:<PublishedPort> in a web browser, as shown in Figure 12-26. The request is forwarded to the ingress load balancer on one of the instances in the Swarm. The instance that the external request is forwarded to does not have to be hosting a service task. Finding a service task is what the ingress load balancer does.



Figure 12-26. Accessing a Docker service at the elastic load balancer DNS

Summary

This chapter discussed load balancing in Swarm mode. An ingress load balancer is used to distribute the load among a service's tasks. Each service in a Swarm is assigned a DNS name and an internal load balancer balances service requests among the services based on DNS name. We also created an external load balancer for AWS EC2 instances to distribute load among the EC2 instances. Docker for AWS creates an external load balancer automatically on AWS. In the next chapter we discuss developing a Docker Swarm based highly available website.

CHAPTER 13

Developing a Highly Available Website

High availability of a website refers to a website being available continuously without service interruption. A website is made highly available by provisioning fault tolerance into the Docker Swarm application. High availability is provided at various levels. The ingress load balancer balances incoming client requests across the multiple service tasks and provides fault tolerance at the tasks level. If one service task fails, client traffic is routed to another service task. Using an external load balancer for a Docker Swarm hosted across multiple availability zones is another method for providing high availability. An external load balancer provides fault tolerance at the node level. If one node fails, client traffic is routed to Swarm nodes on another node.

The Problem

Using an external load balancer such as an AWS Elastic Load Balancer provides fault tolerance across multiple availability zones in an AWS region. The elastic load balancer may be accessed at its DNS name by a client host, as illustrated in Figure 13-1. The Swarm is not highly available, as failure of a single AWS region would cause a website to become unavailable.



Figure 13-1. The elastic load balancer may be accessed at its DNS name by a client host
The Solution

Amazon Route 53 provides high availability with various DNS failover options, including active-active and active-passive failover using alias resource record sets. Amazon Route 53 provides DNS failover across AWS regions that are geographically spread, as illustrated in Figure 13-2. We use the Amazon Route 53 active-passive failover configuration based on the primary-secondary architectural patter for load balancer DNSes.



Figure 13-2. Amazon Route 53 provides DNS failover across AWS regions

This chapter covers the following topics:

- Setting the environment
- Creating multiple Docker swarms
- Deploying a Docker Swarm service
- Creating a AWS Route 53
- Creating a hosted zone
- Configuring name servers
- Creating record sets
- Testing high availability
- Deleting a hosted zone

Setting the Environment

We use two Docker for AWS managed Swarms for providing two DNS for active-passive DNS failover configuration. A Route 53 provides the primary-secondary architectural pattern for the two DNSes. The only prerequisite is an AWS account, which may be created at https://aws.amazon.com/resources/create-account/. Create a key pair (Swarm) that is to be used for SSH login to Swarm manager nodes, as shown in Figure 13-3. Set the permissions on the key pair to read-only by the owner only with the chmod 400 swarm. pem command.

Create Key Pair Import H	Key Pair Delete	÷ • (2
Q. Filter by attributes or searc	h by keyword	② K < 1 to 1 of 1 > >	
Key pair name	Fingerprint	Ŧ	
swarm	ae:ed:95:29:3c:0f:00:da:fd:5f:d7:a1:o	c9:a0:29:be:78:6f:9b:e4	Ī

Figure 13-3. Key pair

A domain name must be registered to be used for creating an Amazon Route 53 hosted zone.

Creating Multiple Docker Swarms

Create two Docker Swarms using the Docker for AWS managed service at https://docs.docker.com/ docker-for-aws/. The two Docker Swarms must be in two different AWS regions to use the high availability provided by geographically distributed AWS regions. Create one Docker Swarm Oregon region as an example, as shown in Figure 13-4.

🧊 Services 🗸 R	esource Groups 👻 🔭		🇘 🛛 Deepak Vohra 🗸	Oregon - Support -
CloudFormation	✓ Stacks			
Create Stack - Act	ions ▼ Design template			C O
Filter: Active - By Stac	k Name			Showing 1 stack
Stack Name	Created Time	Status	Description	
DockerSwarm-1	2017-02-12 14:01:27 UTC-0800	CREATE_COMPLETE	Docker for AWS 1.13.1 (ga-2)	

Figure 13-4. CloudFormation stack for Docker Swarm

Each Docker Swarm has manager and worker nodes spread across the AWS availability zones in an AWS region. The public IP of a manager node may be obtained from the EC2 console, as shown in Figure 13-5.

CHAPTER 13 DEVELOPING A HIGHLY AVAILABLE WEBSITE

aunch instand	Connect	Actions Y						
	Connect	Actions					0 4	2
Q Filter by tags	and attributes or s	earch by keyword			0	I< < 1 to 8	of 8 >	- 3
Name	÷	Instance ID	Instance Type 👻	Availability Zone 👻	Instance State	- Status Che	cks -	Al
DockerSwa	arm-1-worker	i-009e516538a4603d2	t2.micro	us-west-2c	running	2/2 che	cks	No
DockerSwa	arm-1-worker	i-031913742454e9b90	t2.micro	us-west-2c	running	2/2 che	cks	No
DockerSwa	arm-1-Manager	i-054193cb964bbbce5	t2.micro	us-west-2c	running	2/2 che	cks	No
DockerSwa	arm-1-Manager	i-06c7242048df94d06	t2.micro	us-west-2b	running	2/2 che	cks	No
DaskarSur	um 1 undere	1 0C-07640-4304961	17 miero	ue uset 3b	a ninnina	@ 212 cho	eke	Ale
Instance: i-05 .compute.ama:	4193cb964bbbce zonaws.com	e5 (DockerSwarm-1-Ma	nager) Public [DNS: ec2-54-149-86-1	148.us-west-			E
Description	Status Checks	Monitoring Tags	S					
	Instance ID	i-054193cb964bbbce5		Public DNS (IF	2.compute)-86-148.us-west .amazonaws.cor	- n	
	Instance state	running		Pv4 Public	c IP 54.149.86.	148		
	Instance type	t2.micro		IPv6	IPs -			
	Elactic IDe			Private [INS in-172-31-4	14-164 us-west-		

Figure 13-5. Obtaining the public IP of the Swarm manager node

Using the public IP address for a manager node in the first Docker Swarm, SSH login to the manager node EC2 instance.

```
[root@localhost ~]# ssh -i "swarm.pem" docker@54. 149.86.148
```

```
Welcome to Docker!
```

~\$

Create the other Docker Swarm in the Ohio AWS region as an example, as shown in Figure 13-6. The regions may be different for different users.

Û	Services - Res	ource Groups 👻 🕈		🗘 🛛 Deepak Vohra 🗸	Ohio - Support -
٥	CloudFormation ~	Stacks			
c	Create Stack - Action	Design template			C O
Fi	iter: Active - By Stack N	lame			Showing 1 stack
	Stack Name	Created Time	Status	Description	
8	DockerSwarm-2	2017-02-12 14:14:06 UTC-0800	CREATE_COMPLETE	Docker for AWS 1.13.1 (ga-2)	

Figure 13-6. CloudFormation stack for the Docker Swarm in one region

The Swarm node EC2 instances for the second Docker Swarm are also spread across the AWS availability zones in the second AWS region, as shown in Figure 13-7. Obtain the public IP for a manager node.

Q Filter by tag	s and attributes or se	earch by keyword					0	K	1 to 8 of 8	> >
Name	~	Instance ID	•	Instance Type 👻	Availabili	ity Zone 👻	nstance State	~ St	tatus Checks 👻	Alarm
DockerSw	arm-2-Manager	i-009e6f31f68a14	05a	t2.micro	us-east-2a	a (running	0	2/2 checks	None
DockerSw	arm-2-Manager	i-01baf37172925c	5a6	t2.micro	us-east-2b	b (orunning	0	2/2 checks	None
DockerSw	arm-2-worker	i-0431b8d8e0f034	3b5	t2.micro	us-east-2a	а (orunning	0	2/2 checks	None
DockerSw	arm-2-worker	i-05cf4e01e90967	1f9	t2.micro	us-east-20	. (orunning	0	2/2 checks	None
DockerSw	arm-2-worker	i-08609eb056b862	22f7	t2.micro	us-east-2a	a (running	0	2/2 checks	None
h nution			1.65	in anti-					20 shashe	
Instance: i-00 .compute.ama Description	09e6f31f68a1405a izonaws.com Status Checks	(DockerSwarm-2	2-Mana Tags	ager) Public DM	IS: ec2-52	-14-23-163.u	ıs-east-			
	Instance ID	i-009e6f31f68a14	05a		Pu	iblic DNS (IPv	4) ec2-52-14 2.compute	-23-163	us-east-	
	Instance state	running			De	IPv4 Public	P 52.14.23.	163		
	Instance type	t2 micro				IPv6 IP	e .			

Figure 13-7. The Availability Zone column lists multiple zones

SSH login to the instance.

List the Swarm nodes in a Docker Swarm with the Docker node.

~ \$ docker node ls

ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS

```
fncv7ducej3ind4u2sy9xtwi7 ip-172-31-34-223.us-east-2.compute.internal. Ready Active
Reachable
grdeu2x49yi2fmvuy9lmoogqg ip-172-31-43-174.us-east-2.compute.internal Ready Active
keOd75qef9bg8t22eqv9spdpm ip-172-31-30-180.us-east-2.compute.internal. Ready Active
Reachable
m2mmifbrnjbdriub5r36zxyjc * ip-172-31-8-11.us-east-2.compute.internal Ready Active Leader
qenbfrms0xv7wom6wpw9yspw4 ip-172-31-27-178.us-east-2.compute.internal Ready Active
tipzy29hgh3m6og5bzkgsego8 ip-172-31-12-37.us-east-2.compute.internal Ready Active
v4xdl4jvthovrzsamujoxy3ju ip-172-31-7-219.us-east-2.compute.internal Ready Active
vuq68yex58vzgx3audj3sm23a ip-172-31-28-182.us-east-2.compute.internal Ready Active
```

Deploying a Docker Swarm Service

Next, we deploy a Hello World service that will be hosted on a website. Run the following command on a manager instance for the DockerSwarm-1 Swarm to create a tutum/hello-world service with two replicas exposed at port 8080 on the host nodes.

```
docker service create \
    --name hello-world \
    --publish 8080:80 \
    --replicas 2 \
    tutum/hello-world
```

A Docker service with two service tasks is created.

~ \$ docker service create $\$

```
> --name hello-world \
> --publish 8080:80 \
> -- replicas 2 \
> tutum/hello-world
vn5fl8h7t65sjwk54dwcoklhu
```

~ \$ docker service 1s

ID NAME MODE REPLICAS IMAGE

vn5tl8h7t65s hello-world replicated 2/2 tutum/hello-world:latest

~ \$ docker service ps hello-world

ID NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR PORTS

ac9ks5y9duni2 hello-world.l tutum/hello-world:latest ip-172-31-19-220.us-west-2.compute. internal Running Running 13 seconds ago 8s6r48wUui9 hello-world.2 tutum/hello-world:latest ip-172-31-24-250.us-west-2.compute. internal Running Running 13 seconds ago

Scale the service to 10 replicas to provide load distribution. Subsequently, list the services to list 10/10 replicas as running.~ \$ docker service scale hello-world=10

hello-world scaled to 10
~ \$ docker service ls

ID NAME MODE REPLICAS IMAGE

vn5U8h7t65s hello-world replicated 10/10 tutum/hello-world:latest

~ \$

The 10 service task replicas are scheduled across the Swarm nodes, as shown in Figure 13-8.

~ \$ docker service	ps hello-world				
ID NAME	IMA	GE		NODE	
1	DESIRED STAT	E CURRENT	STATE	ERROR	PORTS
ac9ks5y9dum2 hello	-world.1 tut	um/hello-w	orld:latest	ip-172-31-19-2	20.us-west
-2.compute.internal	Running	Running	about a minu	ite ago	
8s6r48wltui9 hello	-world.2 tut	um/hello-w	orld:latest	ip-172-31-24-2	50.us-west
-2.compute.internal	Running	Running	about a minu	ite ago	
k4r20unv1xxs hello	-world.3 tut	um/hello-w	orld:latest	ip-172-31-19-2	20.us-west
-2.compute.internal	Running	Running	49 seconds a	go	
imz825y6j5ya hello	-world.4 tut	um/hello-w	orld:latest	ip-172-31-6-10	9.us-west-
2.compute.internal	Running	Running	46 seconds a	igo	
m2lz9wpsbtea hello	-world.5 tut	um/hello-w	orld:latest	ip-172-31-44-1	64.us-west
-2.compute.internal	Running	Running	45 seconds a	igo	
4kxqmmc3ux1w hello	-world.6 tut	um/hello-w	orld:latest	ip-172-31-42-2	45.us-west
-2.compute.internal	Running	Running	46 seconds a	igo	
o3ychxa9p1y8 hello	-world.7 tut	um/hello-w	orld:latest	ip-172-31-36-2	49.us-west
-2.compute.internal	Running	Running	46 seconds a	igo	
uriyh6hh9o26 hello	-world.8 tut	um/hello-w	orld:latest	ip-172-31-20-2	51.us-west
-2.compute.internal	Running	Running	46 seconds a	igo	
nmrlnzv17yj5 hello	-world.9 tut	um/hello-w	orld:latest	ip-172-31-3-20	9.us-west-
2.compute.internal	Running	Running	46 seconds a	igo	
ghfduagrtgqg hello	-world.10 tut	um/hello-w	orld:latest	ip-172-31-3-20	9.us-west-
2.compute.internal	Running	Running	45 seconds a	igo	

Figure 13-8. Service tasks scheduled across the Swarm nodes

Obtain the load balancer DNS for the first Docker Swarm from the EC2 dashboard, as shown in Figure 13-9.

Resource Groups 👻 🔸	í.				Δ	Deepak Vohra 🗸	Oregon 🕶	Supp	ort •	
Create Load Balancer	Actions 💙							0	¢	0
Filter: Q Search		×				K	< 1 to 1	of 1	> >	
Name	 DNS name 		- State		×	VPC ID		Ava	ilabili	ty Z
DockerSwarm-1-EL	B DockerSwarn	n-1-ELB-15626	5			vpc-287ce04f		US-W	est-2	a, us
Load balancer: Docke Description Instar	erSwarm-1-ELB	Listeners	Monitoring	Tags						-
Basic Configuration	on									
Name:	DockerSwarm-1-ELB		Cr	eation time:	Feb	oruary 12, 2017 at	2:02:30 PM	UTC-8		
* DNS name:	DockerSwarm-1-ELB-1 west-2.elb.amazonaws	1562658885.u s.com (A Reco	s- H rd)	osted zone: Status:	Z1H 8 o	H1FL5HABSF5 f 8 instances in se	rvice			
Availability	subnet-18aa9e40 - us	-west-2c,		VPC:	vpc	-287ce04f				•

Figure 13-9. Docker Swarm load balancer

Access the service at <DNS>:<LoadBalancerPort> in a web browser, as shown in Figure 13-10; the load balancer port is set to 8080, the port at which the service is exposed.



Figure 13-10. Accessing the service in a browser

Similarly for the second Docker Swarm, create a tutum/hello-world service with a published port set to 8080. Scale the service to 10 replicas for load distribution across the Swarm.

S docker service create \
> --name hello-world \

> --publish 8080:80 \
,> --replicas 2 \
> tutum/hello-world

woqx2ltuibv53ctmuvssrsq8j

~ \$ docker service ls

ID NAME MODE REPLICAS IMAGE

woqx2ltuibv5 hello-world replicated 2/2 tutum/hello-world:latest

~ \$ docker service ps hello-world NAME IMAGE NODE DESIRED STATE CURRENT STATE ERROR PORTS ny9ermdgb7a4 hello-world.1 tutum/hello-world:latest ip-172-31-34-223.us-east-2.compute. internal Running Running 15 seconds ago

5w3thlgleinme hello-world.2 tutum/hello-world:latest ip-172-31-30-180.us-east-2.compute. internal Running Running 15 seconds ago

~ \$ docker service scale hello-world=10

hello-world scaled to 10

The service replicas are distributed across the Swarm nodes, as shown in Figure 13-11.

~ \$ docker service ps hello-	world	
ID NAME	IMAGE	NODE
DESIRED	STATE CURRENT STATE	ERROR PORTS
ny9ermdgb7a4 hello-world.1	<pre>tutum/hello-world:latest</pre>	ip-172-31-34-223.us-east
-2.compute.internal Running	Running about a min	ute ago
5w3th1g10mme hello-world.2	<pre>tutum/hello-world:latest</pre>	ip-172-31-30-180.us-east
-2.compute.internal Running	Running about a min	ute ago
1warzpstn2wk hello-world.3	tutum/hello-world:latest	ip-172-31-8-11.us-east-2
.compute.internal Running	Running 25 seconds	ago
ff58q94ij91m hello-world.4	tutum/hello-world:latest	ip-172-31-8-11.us-east-2
.compute.internal Running	Running 25 seconds	ago
wtjskvuwhmu4 hello-world.5	tutum/hello-world:latest	ip-172-31-30-180.us-east
-2.compute.internal Running	Running 27 seconds	ago
8vtnxa7yktxu hello-world.6	tutum/hello-world:latest	ip-172-31-43-174.us-east
-2.compute.internal Running	Running 26 seconds	ago
Ombtqclkqwax hello-world.7	tutum/hello-world:latest	ip-172-31-12-37.us-east-
2.compute.internal Running	Running 26 seconds	ago
twfj8nn881u7 hello-world.8	tutum/hello-world:latest	ip-172-31-7-219.us-east-
2.compute.internal Running	Running 26 seconds	ago
98sykb376wn7 hello-world.9	tutum/hello-world:latest	ip-172-31-27-178.us-east
-2.compute.internal Running	Running 25 seconds	ago
g6cyck25jd08 hello-world.10	<pre>tutum/hello-world:latest</pre>	ip-172-31-28-182.us-east
-2.compute.internal Running ~ \$ ∎	Running 26 seconds	ago

Figure 13-11. Service replicas distributed across the Swarm

Obtain the DNS of the elastic load balancer for the second Swarm, as shown in Figure 13-12.

ource Groups 👻 1					Δ	Deepak Vohra 🗸	Ohio 🕶	Suppo	ort +
Create Load Balance	r Actions 🛩							0	¢ 0
Filter: Q Search		×				К	< 1 to 1	of 1	> >
Name	- DNS name	*	State		~ VI	PC ID	~	Avail	ability Z
DockerSwarm-2-E	LB DockerSwarm-	-2-ELB-823944			vp	c-01cf6f68		us-ea	st-2a, us
				_					•
oad balancer: Doc	kerSwarm-2-ELB		000						
Description Inst	ances Health Check	Listeners	Monitoring	Tags					
Basic Configurat	ion								
Name:	DockerSwarm-2-ELB		Cre	eation time:	Februa	ary 12, 2017 at 2:	15:09 PM	UTC-8	
* DNS name:	DockerSwarm-2-ELB-82	:3944047.us-	н	osted zone:	Z3AAD	JGX6KTTL2			
13	east-2.elb.amazonaws.c	om (A Record)		Status:	8 of 8 i	instances in servi	ce		
Scheme:	internet-facing			VPC:	vpc-01	cf6f68			
Availability	subnet-00259569 - us-e	ast-2a,							
Zones:	subnet-0487437f - us-ea	ast-2b,							-

Figure 13-12. Obtaining the DNS name for the Swarm ELB

Access the service at <DNS>:<LoadBalancerPort> in a web browser, as shown in Figure 13-13.



Figure 13-13. Accessing the service in a browser 250

Creating an Amazon Route 53

Amazon Route 53 is a highly available and scalable cloud Domain Name Service (DNS) web service that connects user requests to infrastructure running on the AWS, including Amazon EC2 instances, load balancers, and Amazon S3 buckets. We already created two Docker Swarms hosting the same Docker service using the Docker AWS managed service, which automatically creates an AWS ELB for each Docker Swarm.

In this section, we create an Amazon Route 53 to route user requests to the nosqlsearch.com domain to the elastic load balancers for the two Docker Swarms. In Amazon Route 53, we create two resource record sets pointing to the two different ELBs configured for failover, with one of the ELBs being the primary resource record set and the other being the secondary resource record set.

When the nosqlsearch.com domain is opened in a web browser, the Route 53 routes the request to the primary resource record set. If the primary record set fails, Route 53 routes the user request to the secondary record set, in effect providing high availability of the Hello World Docker service hosted on the nosqlsearch.com domain. To create an AWS Route 53, select Route 53 from the AWS services, as shown in Figure 13-14.



Figure 13-14. Selecting the Amazon Route 53 service

Creating a Hosted Zone

A hosted zone is a configuration that determines how traffic to a domain on the Internet will be routed. To create a hosted zone, open https://console.aws.amazon.com/route53/ in a web browser and click on Create Hosted Zone in the DNS management, as shown in Figure 13-15.



Figure 13-15. Creating the hosted zone

Alternatively, select Hosted Zones or open https://console.aws.amazon.com/route53/home#hosted-zones in a browser and click on Create Hosted Zone, as shown in Figure 13-16.



Figure 13-16. Creating a hosted zone



Click on Create Hosted Zone again, as shown in Figure 13-17.

Figure 13-17. Creating a hosted zone

In the Create Hosted Zone dialog, specify a domain name (nosqlsearch.com). The domain name must be registered with the user. Select Public Hosted Zone for the type, as shown in Figure 13-18.

		Create Hosted Zone	Types •	×	Q Search all fields
ation about how y	ntainer that holds information	A hosted zone is a co want to route traffic f	Zones to display 🔰 渊	< < No He	
		subdomains.	Count- Comment	pe - Record	Domain Name - Typ
	nosqlsearch.com	Domain Name:			
		Comment:	nosted zones	have no	You
		Type:			
	Public Hosted Zone				

Figure 13-18. Configuring the hosted zone

A new public hosted zone is created, as shown in Figure 13-19. The name servers for the hosted zone (by default, there are four) are assigned.

Q Record Set Name		X Any Type Aliases On	nly	Edit Reco	rd Set			
Weighted Only	>۱	S Displaying 1 to 2 out of 2 Record Sets	>1	Name: r Type: N	NS - Name ser	m. ver		
Name	Туре	Value		TTL (Se	conds):	172800 1n	n 5m 1h	1d
nosqlsearch.com.	NS	ns-1293.awsdns-33.org. ns-1929.awsdns-49.co.uk. ns-175.awsdns-21.com. ns-538.awsdns-03.net.		Value:	ns-1293.aws ns-1929.aws ns-175.awsd The domain na	dns-33.org. dns-49.co.uk ns-21.com. me of a name s		
nosqlsearch.com.	SOA	ns-175.awsdns-21.com. awsdns-hostmas	ter.am;		Enter multiple separate line Example: ns1.amazon.	e name servers s. com	on	

Figure 13-19. The new public hosted zone

Configuring Name Servers

Next, we need to configure the name servers for the domain with the domain registrar. The procedure to configure name servers is different for different domain registrars, but an option to add a zone record for a domain should be provided.

Specify the record type as Nameserver, as shown in Figure 13-20. Specify the host as @. Each zone record should point to a single name server, which may be obtained from the public hosted zone we created earlier.

Add Zone Record

Record type: *	View current
NS (Nameserver)	Ţ
Host: * 🕜	
@	
Points to: * 🥡	
ns-1293.awsdns-33.org	
TTL: * 🕢	Seconds: *
Custom	• 600

Figure 13-20. Adding a name server record

Add four name servers (collectively called a *delegation set*), as shown in Figure 13-21, for the domain for which a hosted zone is to be created.

Renew	Created: 2012-07-28	3 Expires: <u>2017</u> 5 Buy & Sell -	07-28 Folder: <u>None</u> Profile: <u>None</u>
Settings	DNS Zone File	Contacts	
Auto-Renew ()	Standard: Extended: Manage	Off Off	
Lock ()	On Manage		
Nameservers ()	NS-1293, NS-1929, NS-175,AV NS-538,AV Updated 2 Manage	WSDNS-33.ORG WSDNS-49.CO.UI VSDNS-21.COM VSDNS-03.NET 017-02-12	<

Figure 13-21. Name servers configured on a domain

Creating Resource Record Sets

After creating and configuring a hosted zone, create one or more resource record sets. A resource record set is a Domain Name System (DNS) configuration for routing traffic to a domain. Click on Create Record Set to create a resource record set, as shown in Figure 13-22.

CHAPTER 13 DEVELOPING A HIGHLY AVAILABLE WEBSITE

Record Set Name		Any Type 🔹	Aliases Only	Edit Reco	rd Set			
Weighted Only	K	Solution State	of 2 Record Sets 📏 🔌	Name: Type:	nosqlsearch NS – Name	server		
Name	Type	Value		Alias:	Yes No	0		
nosqlsearch.com.	NS	ns-1293.awsdns-33.or ns-1929.awsdns-49.co ns-175.awsdns-21.con ns-538.awsdns-03.net	g. .uk. n.	Value:	ns-1293.awsdns-33.org. ns-1929.awsdns-49.co.uk. ns-175.awsdns-21.com. The domain name of a name server. Enter multiple name servers on recorded lines			
nosqlsearch.com.	SOA	ns-176.awsdns-21.con	n. awsdns-hostmaster.am:		separate Example: ns1.ama: ns2.ama: ns3.ama: ns4.ama:	lines. zon.com zon.org zon.net zon.co.uk	4	

Figure 13-22. Creating a record set

In the Create Record Set tab, the type should be set to A –IPv4 address, as shown in Figure 13-23. The name of each record set ends with the domain name. Select Yes for Alias.

Q	Record Set Name		X Any Type •	Aliases Only	Create R	ecord Se	t	
	Weighted Only	L/			Name: Type:	A - IPv	4 address	nosqlsearch.co
	Name	Туре	Value	2 Record Sets 🦻 🦓	Alias:	• Yes) No Enter target nam	e
	nosqlsearch.com.	NS	ns-1293.awsdns-33.org ns-1929.awsdns-49.co. ns-175.awsdns-21.com ns-538.awsdns-03.net.). uk.	You can - Cloudf - Elastic - ELB lo - S3 wel	also type th Front distribute Beanstalk ad balancer bsite endpo	e domain name for ti ution domain name: o environment CNAME r DNS name: example int: example.s3-webs	he resource. Examples: 1111111abcdef6.cloudfront. : example.elasticbeanstalk.i e-1.us-east-1.elb.amazonaws.ci
	nosqlsearch.com.	SOA	ns-175.awsdns-21.com	. awsdns-hostmaster.am	- Resou Learn M	rce record s lore	set in this hosted zone	e: www.example.com
					Routin	g Policy:	Simple	*
					Route 53 More	3 responds t	to queries based only	on the values in this record
					Evalua	te Targe	t Health: 🔘 Yes	• No

Figure 13-23. Configuring a record set

Next, select the alias target as the AWS Elastic Load Balancer DNS for one of the Docker Swarms, as shown in Figure 13-24.

Q	Record Set Name		X Any Type •	Aliases Only	Create Re	cord Se	et		
	Weighted Only				Name: Type:	A-IP	v4 address		nosqlsearch.com
		14	S Displaying 1 to 2 out of	2 Record Sets 🤌 🔌	Alias: ()	Yes (O No		
	nosqlsearch.com.	NS	ns-1293.awsdns-33.org ns-1929.awsdns-49.co. ns-176.awsdns-21.com ns-538.awsdns-03.net. ns-176.awsdns-21.com	uk. . awsdns-hostmaster.am	Alias Ta You can al - CloudFro - Elastic B - ELB load - S3 webs - Resourc Learn Mor	Iso type ont distr Beanstal d balanc site end; ce record re	— <i>S3 web</i> : No Targets — <i>ELB Ap</i> , No Targets — <i>ELB Cla</i> HelloWorld	site endpoints Available blication load Available Issic load bala LB-13835901	— balancers — ancers — 2.us-east-1.elb.an
					Routing Route 53 r More	Policy respond	DockerSwa DockerSwa CloudFr et Health:	Irm 2-ELB-82 Irm 1-ELB-15 Dat distribution	3944047.us-east- 62658885.us-wes ns 0



Next, select the routing policy, as shown in Figure 13-25.

QR	Record Set Name		X Any Type Aliases Only	Create R	ecord Set		
	Weighted Only			Name: Type:	A - IPv4 a	ddress	nosqlsearch.com
	Name	I≪ Type	✓ Displaying 1 to 2 out of 2 Record Sets >> > Value	Alias:	• Yes ON	0	
	nosqlsearch.com.	NS SOA	ns-1293.awsdns-33.org. ns-1929.awsdns-49.co.uk. ns-175.awsdns-21.com. ns-538.awsdns-03.net. ns-175.awsdns-21.com. awsdns-hostmaster.a	Alias I Alias I You can - Cloudi - Elastic - ELB lo n: - S3 wei - Resou Learn M	Instead Zone also type the d Front distribution Beanstalk environd bazonaws.com bsite endpoint: rece record set in lore	ISTACK DOCKETSW D: Z3AADJGX omain name for the r n domain name for the r n domain name: d111 romment CNAME: ex. IS name: example.1.1 example.s3-website-to this hosted zone: w	skTTL2 source. Examples: 111abcdef8.cloudfront.ne ample.elasticbeanstalk.co us-east- us-east-1.amazonaws.com ww.example.com
				Routin	g Policy:	Simple	Th
				Route 53 Learn M	3 responds to q lore	Simple Welghted Latency	in this record.
_						Failover Geolocation	2 9

Figure 13-25. Selecting a routing policy 258

Select Failover for the routing policy. This configures DNS failover, as shown in Figure 13-26. Select Failover Record Type as Primary.

Q Rec	ord Set Name		X	Any Type *	Aliases Only	Create	Record S	iet		
We	eighted Only	12	Dien	leving 1 to 2 out of	2 Decord Sate	Name Type:	A – IF	^o v4 addres	5	nosqlsearch.com
		14	< Diap	aying 1 to 2 out of	2 100010 3003 # #1	Alias	: Yes	O No		
no no	sqlsearch.com. sqlsearch.com.	NS SOA	ns-129 ns-192 ns-175 ns-538 ns-175	3.awsdns-33.org 9.awsdns-49.co. .awsdns-21.com. .awsdns-03.net. .awsdns-21.com.	uk. awsdns-hostmaster.am:	Alias Target: dualstack.Docker Alias Hosted Zone ID: Z3AADJG You can also type the domain name for th - CloudFront distribution domain name: d - Elastic Beanstalk environment CNAME: - ELB load balancer DNS name: example 1.elb.amazonaws.com - S3 webaite endpoint: example.s3-webait - Resource record set in this hosted zone Learn More		DockerSwarr Z3AADJGX6KT ame for the resound in name: d111111 tt CNAME: example e: example-1.us-e e: s3-website-us-e osted zone: www.	n-2-ELB-82394 TL2 rce. Examples: abcdef8.cloudfront.ne le.elasticbeanstalk.cor ast- ast-1.amazonaws.com example.com	
						Route Route health	ing Policy 53 respond by, or using s	y: Failo is to queries u secondary rec ord Type:	iver sing primary reco ord sets otherwise Primary	rd sets if any are a. Learn More

Figure 13-26. Selecting failover record type

For Evaluate Target Health, select Yes, as shown in Figure 13-27.

QReco	ord Set Name		X Any Type T	Aliases Only	Alias Targe	et: dua	Istack.DockerS	warm-2-ELB-82394
We	eighted Only	K	Control of the provide state of the providence of the providenc	2 Record Sets >>	Alias Hoste You can also t - CloudFront e - Elastic Bean - ELB load ba	ed Zone ype the do distribution stalk envir lancer DN	ID: Z3AADJG main name for the domain name: d1 ronment CNAME: o S name: example-	X6KTTL2 resource. Examples: 11111abcdef8.cloudfront.ne example.elasticbeanstalk.co 1.us-east-
Na	ime	Туре	Value ns-1293.awsdns-33.org		1.elb.amazona - S3 website e - Resource re Learn More	endpoint: e cord set in	example.s3-website this hosted zone:	e-us-east-1.amazonaws.com www.example.com
no	sqlsearch.com.	NS	ns-1929.awsdns-49.co. ns-175.awsdns-21.com ns-538.awsdns-03.net.	uk.	Routing Po Route 53 resp	licy:	Failover veries using primar	v record sets if any are
no	sqlsearch.com.	SOA	ns-175.awsdns-21.com	. awsdns-hostmaster.am	healthy, or usi	ecord T	ary record sets oth	v 🔘 Secondary
					Set ID: F	rimary		
					Evaluate Ta	arget He	ealth: Yes	© No
					Associate v	vith Hea	alth Check: 🤇	Yes No
							_	

Figure 13-27. Selecting the Evaluate Target Health option

For Associate with Health Check, select No. Click on Create, as shown in Figure 13-28.

Q Record Set Name			Only	Alias Target: dualstad	ck.DockerSwarm-2-ELB-82394
Weighted Only	[≮	Displaying 1 to 2 out of 2 Record Sets	> >1	Alias Hosted Zone ID: You can also type the domain - CloudFront distribution dom - Elastic Beanstalk environm - ELB load balancer DNS nar	Z3AADJGX6KTTL2 in name for the resource. Examples: lain name: d11111abcdef8 cloudfront.net ent CNAME: example.elasticbeanstalk.com me: example-1.us-east-
Name	Туре	Value		1.elb.amazonaws.com - S3 website endpoint: examp - Resource record set in this	ple.s3-website-us-east-1.amazonaws.com hosted zone: www.example.com
nosqlsearch.com.	NS	ns-1293.awsdns-33.org. ns-1929.awsdns-49.co.uk. ns-175.awsdns-21.com. ns-538.awsdns-03.net.		Learn More Routing Policy: Fai	lover 🔻
nosqlsearch.com.	SOA	ns-175.awsdns-21.com. awsdns-hostn	naster.am:	Failover Record Type: Set ID: Primary	using primary record sets if any are ecord sets otherwise. Learn More
				Evaluate Target Healt	h: ● Yes ○ No

Figure 13-28. Creating a record set

A primary record set is created, as shown in Figure 13-29; "primary" implies that website traffic will be first routed to the record set.

Record Set Name		X Any Type Aliases Only		
Weighted Only			O get started, click Cre existing record set.	eate Record Set button or click ar
	≮	Solution of the second Sets > >		
Name	Туре	Value		
nosqlsearch.com.	А	ALIAS dualstack.dockerswarm-2-elb-823944047		
nosqlsearch.com.	NS	ns-1293.awsdns-33.org. ns-1929.awsdns-49.co.uk. ns-175.awsdns-21.com. ns-538.awsdns-03.net.		
nosalsearch com	SOA	ns-175 awsdns-21 com awsdns-hostmaster am		

Figure 13-29. Primary record set

Record Set Name		X Any Type •	Aliases Only		
Weighted Only				To get started, click Cre existing record set.	eate Record Set button or click an
	<	Solution Control Co	3 Record Sets > >		
Name	Туре	Value			
nosqlsearch.com.	A	ALIAS dualstack.docker	swarm-2-elb-823944047		
nosqlsearch.com.	NS	ns-1293.awsdns-33.org ns-1929.awsdns-49.co.t ns-175.awsdns-21.com ns-538.awsdns-03.net.	uk		
nosqlsearch.com.	SOA	ns-175.awsdns-21.com.	awsdns-hostmaster.am		

To create a secondary record set, click on Create Record Set again, as shown in Figure 13-30.

Figure 13-30. Creating another record set

Select the type as A –IPv4 address and choose Yes for Alias. Select Alias Target as the second ELB DNS, as shown in Figure 13-31.

	et	ecord Set	Create R	Any Type Aliases Only		Record Set Name	Q
nosqlsearch.com	v4 address	A - IPv4 addres	Name: Type:	5 SI	12	Weighted Only	
	🗩 No	Yes 🔘 No	Alias: (ue	Туре	Name	
points —	— 53 website endpoints —	arget:	Alias T	AS dualstack.dockerswarm-2-elb-823944047	A	nosqlsearch.com.	
le <i>load balancers</i> — le <i>d balancers</i> — 859012 us-east-1 elb am	No Targets Available — ELB Application load baland No Targets Available — ELB Classic load balancers HelloWorld B-138359012 use	ront distr Beanstal Id balanc Ce record Ore HelloWor	- CloudF - Elastic - ELB loc - S3 web - Resour Learn Mo	1293.awsdns-33.org. 1929.awsdns-49.co.uk. 175.awsdns-21.com. 138.awsdns-03.net.	NS	nosqlsearch.com.	
.B-823944047.us-east-2	DockerSwarm-2-ELB-8239440	Policy DockerSy	Routing	75.awsdns-21.com. awsdns-hostmaster.am	SOA	nosqlsearch.com.	
ihutions	ClaudFrant distributions -	respond Cloud	Route 53 More				
No	t Health: 🔾 Yes 💿 No	e Target Health	Evaluat				
No	et Health: O Yes I No	e Target Health	Evaluat				

Figure 13-31. Selecting an alias target

Select the Failover routing policy and the secondary Failover Record Type, as shown in Figure 13-32.

Q	Record Set Name		X Any Type Aliases Only	Create R	ecord Set	£	
	Weighted Only	14		Name: Type:	A-IPv	4 address	nosqlsearch.com
	Name	Туре	Value	Alias:	• Yes C) No	
	nosqlsearch.com.	A	ALIAS dualstack.dockerswarm-2-elb-823944047	Alias T Alias H	arget: c	iualstack.Docker one ID: Z1H1FL	Swarm-1-ELB-15626
	nosqlsearch.com.	NS	ns-1293.awsdns-33.org. ns-1929.awsdns-49.co.uk. ns-175.awsdns-21.com. ns-538.awsdns-03.net.	You can - CloudF - Elastic - ELB lo 1.elb.am - S3 wel	also type the ront distribute Beanstalk of ad balancer azonaws.co osite endpoi	e domain name for ti ution domain name: o environment CNAME DNS name: example m int: example s3-webs	he resource. Examples: i111111abcdef8.cloudfront.net example.elasticbeanstalk.cor e-1.us-east- ite-us-east-1.amazonaws.com
	nosqlsearch.com.	SOA	ns-175.awsdns-21.com. awsdns-hostmaster.am;	- Resou Learn M	rce record s ore	et in this hosted zone	a: www.example.com
				Routing	Policy:	Failover	•
				Route 53 healthy,	responds to or using sec	o queries using prim condary record sets o	any record sets if any are therwise. Learn More
							ary sociality up

Figure 13-32. Selecting failover record type as secondary

Choose Yes for the Evaluate Target Health and No for the Associate with Health Check. Click on Create, as shown in Figure 13-33.

ype the domain name for the resource. Examples: distribution domain name: d11111abcdef8 cloudfront.net istalk environment CNAME: example.elasticbeanstalk.com lancer DNS name: example-1.us-east- two.com endpoint: example s3-website-us-east-1.amazonaws.com cord set in this hosted zone: www.example.com
aws.com endpoint: example s3-website-us-east-1.amazonaws.com cord set in this hosted zone: www.example.com
licy: Failover
econd Type: Primary Secondary
arget Health: Yes No
vith Health Check: 🔘 Yes 💿 No

Figure 13-33. Creating a secondary record set

The secondary record set is created; "secondary" implies that traffic is routed to the record set if the primary record set fails, as shown in Figure 13-34. Click on Back to Hosted Zones.

Bac	k to Hosted Zones		Create Record Set	Import Zone File	Delete Record Set	Test Record Set
Q, F	Record Set Name		X Any Type *	Aliases Only		
	Weighted Only				To get started, click Cre existing record set.	eate Record Set button or click an
		K	Comparing 1 to 4 out of the second	f 4 Record Sets 🔰 🔰		
	Name	Туре	Value			
	nosqlsearch.com.	A	ALIAS dualstack.dockerswarm-2-elb-823944047			
	nosqlsearch.com.	A	ALIAS dualstack.docke	erswarm-1-elb-156265888	12	
			ns-1293.awsdns-33.org	g. "		
	nosolsearch.com.	NS	ns-1929.awsdns-49.co	.uk.		
			ns-175.awsdns-21.com	1.		
			ns-538.awsdns-03.net.			
	nosolsearch com	SOA	ns-175 awsdns-21 com	awsdns-hostmaster am:		

Figure 13-34. Secondary record set is created

The domain (nosqlsearch.com) is configured with four record sets, as shown in Figure 13-35.

0	Search all fields		X All Types			Hosted Zone Details
~	Search an lields		K K Displaying 1	to 1 out of 1 Ho	sted Zones 🗦 🗲	Domain Name: nosqlsearch.com. Type: Public Hosted Zone
	Domain Name *	Туре ~	Record Set Count*	Comment	Hosted Zone ID	Hosted Zone ID: Z2WEJDWBFEX8U3
۲	nosqlsearch.com.	Public	4		Z2WEJDWBFEX8U	Record Set Count: <u>4</u> Comment: Ø
						Name Servers *: ns-175.awsdns-21.com ns-538.awsdns-03.net ns-1293.awsdns-33.org ns-1929.awsdns-49.co.uk
						* Before the Domain Name System will start to route queries for this domain to Route 53 name servers, you must update the name server records either with the current DNS service or with the registrar for the domain, as applicable. For more information, click the ? icon above

Figure 13-35. Hosted zone created

Testing High Availability

Next, we test the high availability we configured. Open the domain, including the service published port (nosqlsearch.com:8080), in a web browser, as shown in Figure 13-36. The Docker service output should be displayed.



Figure 13-36. Invoking a service in a browser

To test high availability, delete the CloudFormation stack for the Docker Swarm associated with the primary record set, as shown in Figure 13-37.

0	CloudFormat	tion v Stad	cks				
Cr	reate Stack -	Actions •	Design template				C O
Filt	ter: Active - B	Create Change	e Set For Current Stack				Showing 1 stack
	Stack Name	Update Stack		Status		Description	
	DockerSwarm-2	Delete Stack	•	800 CREAT	E COMPLETE	Docker for AWS 1 13 1 (ga-2)	
20	o o cital o numita	View/Edit temp	late in Designer	GILLI	C_CONTRACTO	500000 101 101 100 1.10. 1 (gu-z)	

Figure 13-37. Deleting a stack

Click on Yes, Delete in the Delete Stack dialog. The stack should start to be deleted, as indicated by the DELETE_IN_PROGRESS status shown in Figure 13-38.

0	CloudFormation ~	Stacks			
Cr	eate Stack - Actions	Design template			C O
Filt	ter: Active - By Stack Nar	ne			Showing 1 stack
	Stack Name	Created Time	Status	Description	
	DockerSwarm-2	2017-02-12 14:14:06 UTC-0800	DELETE_IN_PROGRESS	Docker for AWS 1 13 1 (ga-2)	

Figure 13-38. The delete is in progress

The DNS fails over to the secondary resource record set and the domain continues to serve the Docker service, as shown in Figure 13-39.

) Hello world! X	·	
← → X ③ nosqlsearch.com:8080		☆ eí O :
	Hello world!	
	My hostname is 1ed90aee6c33	

Figure 13-39. Domain continues to serve

The hostname in the browser could become different if the request is forwarded to a different service task replica, as shown in Figure 13-40. But the hostname could also become different regardless of whether failover has been initiated, because the ingress load balancer distributes traffic among the different service replicas.



Figure 13-40. Different hostname

Deleting a Hosted Zone

Before a hosted zone can be deleted, all the resource record sets associated with the hosted zone must be deleted. Select the resource record sets to delete and click on Delete Record Set, as shown in Figure 13-41.

Back to Hosted Zones	0	Create Record Set	Import Zone File	Delete Record Set	Test Record Set	
Record Set Name		X Any Type •	Aliases Only	2 record sets selected		
Weighted Only	I<	Comparing 1 to 4 out of the second	If 4 Record Sets 🔌	Selected resource Name nosgisearch.com nosgisearch.com	records:	Type A A
Name *	Турет	Value		and the second second second		
nosqlsearch.com.	A	ALIAS dualstack.dock	erswarm-2-elb-82394404			
nosqlsearch.com.	A	ALIAS dualstack.dock	erswarm-1-elb-15626588			
nosqlsearch.com.	NS	ns-1293.awsdns-33.o ns-1929.awsdns-49.co ns-175.awsdns-21.co ns-538.awsdns-03.ne	rg. " o.uk. m. t.			
nosqlsearch.com.	SOA	ns-175.awsdns-21.co	m. awsdns-hostmaster.am			

Figure 13-41. Deleting the record sets 266

Click on Confirm in the Confirm dialog, as shown in Figure 13-42.

Are you sure you want to delete the	following 2 record
sets?	
 nosqlsearch.com. 	
 nosqlsearch.com. 	

Figure 13-42. Confirmation dialog

Click on Back to Hosted Zones, as shown in Figure 13-43.



Figure 13-43. Going back to the hosted zones

Select the hosted zone to delete and click on Delete Hosted Zone, as shown in Figure 13-44.

CHAPTER 13 DEVELOPING A HIGHLY AVAILABLE WEBSITE

Q Search a	all fields	×	All Types	,		Hosted Zone Details
	≪ ≪ Disp	laying 1 to	1 out of 1 Hosted	Zones ≽ 🖇	-1	Domain Name: nosqlsearch.com. Type: Public Hosted Zone
Domai	Name • Type	* Reco	rd Set Count *	Comment	1	Hosted Zone ID: Z2WEJDWBFEX8U3
nosqls	arch.com. Publi	c 2			z	Record Set Count: 2 Comment: 8
						Name Servers *: ns-175.awsdns-21.com ns-538.awsdns-03.net ns-1293.awsdns-33.org ns-1929.awsdns-49.co.uk
						* Before the Domain Name System will start to route queries for this domain to Route 53 name servers, yo must update the name server records either with the current DNS service or with the registrar for the dom as applicable. For more information, click the ? icon

Figure 13-44. Deleting a hosted zone

Click on Confirm in the Confirm dialog, as shown in Figure 13-45.

Confirm	Cancel 🗙
The name servers for this hosted a servers for the nosqlsearch.com de	cone are still the name omain.
If you want to use this domain name recommend that you either keep the transfer DNS service to another pro- DNS queries from possibly being mi Learn more	e in the future, we hosted zone or vider to prevent future isrouted.
Are you sure you want to delete the hosted zone?	nosqlsearch.com
	Confirm Cancel

Figure 13-45. Confirmation dialog for deleting a hosted zone

The hosted zone is deleted.

Summary

This chapter developed a highly available website using an Amazon Route 53 hosted zone. First, we created two Docker Swarms using the Docker for AWS managed service and deployed the same Docker service on each. Each Docker Swarm service may be accessed using the AWS Elastic Load Balancer for the Docker Swarm created automatically by the Docker for AWS. The Route 53 hosted zone is to create a hosted zone for a domain to route traffic to DNSes configured in the primary/secondary failover pattern. Subsequently, we tested that if the Docker Swarm for the primary record set is shut down, the website is still available, as the hosted zone routes the traffic to the secondary ELB DNS. In the next chapter we discuss using the Docker Swarm mode in Docker Cloud.

CHAPTER 14

Using Swarm Mode in Docker Cloud

Docker for AWS is a managed service for Docker Swarm based on a custom Linux distribution, and hosted on AWS with all the benefits inherent with being integrated with the AWS Cloud platform, such as centralized logging with CloudWatch, custom debugging, auto-scaling groups, elastic load balancing, and a DynamoDB database.

The Problem

While AWS is a managed cloud platform, it is not a managed service for Docker containers, images, and services per se. Docker's builds and tests still need to be integrated.

The Solution

Docker Cloud is a managed service to test code and build Docker images and to create and manage Docker image repositories in the Docker Cloud registry. Docker Cloud also manages Docker containers, services, stacks, nodes, and node clusters. A *stack* is a collection of services and a *service* is a collection of containers. Docker Cloud is an integrated cloud service that manages builds and images, infrastructure, and nodes and apps.

Docker Cloud also introduced a Swarm mode to manage Docker Swarms. In Swarm mode, Docker Cloud is integrated with Docker for AWS. As a result, Docker Cloud Swarm mode is an integration of two managed services—Docker for AWS and Docker Cloud.

Docker Cloud provides some Docker images to interact between a Docker Swarm and a Docker host client, as discussed in Table 14-1.

Docker Image	Description
dockercloud/client	Used on the client side to start an interactive shell to connect to a remote docker Swarm cluster using Docker ID credentials.
dockercloud/client-proxy	Used on the client side to forward local docker API calls to a remote swarm cluster by injecting Docker ID authorization information on each request.
dockercloud/server-proxy	Authenticates and authorizes incoming Docker API calls and forwards them to the local Docker engine.
dockercloud/registration	Registers a Swarm cluster to Docker Cloud and launches a server proxy.

Table 14-1. Docker Images for Docker Swarm

In this chapter, we discuss the Docker Cloud Swarm mode to provision a Docker Swarm with infrastructure hosted on AWS. This chapter covers the following topics:

- Setting the environment
- Creating an IAM role
- Creating a Docker Swarm in Docker Cloud
- Connecting to the Docker Swarm from a Docker host
- Connecting to the Docker Swarm from a Swarm manager
- Bringing a Swarm into Docker Cloud

Setting the Environment

As Docker Cloud is a managed service, all that is required is an account, which may be created at https://cloud.docker.com/. An AWS account is also required and may be created at https://aws.amazon.com/resources/create-account/. Also create a key pair in the region in which the EC2 instances for the Docker Swarm will run, as shown in Figure 14-1.

			Ð	¢	0
0	K <	1 to :	2 of 2	>	>
	0	ØK			

Figure 14-1. Creating a key pair on AWS EC2

Creating an IAM Role

The Docker Cloud Swarm mode requires an AWS role with a new policy, an embedded policy for Docker for AWS. To create the IAM role, navigate to https://console.aws.amazon.com/iam/home?#roles in a web browser. Click on Create New Role, as shown in Figure 14-2.



Figure 14-2. Creating a new role

Specify a role name (dockercloud-swarm-role), as shown in Figure 14-3, and click on Next Step.

Services - Resour	ce Groups 👻 🥆		4	Deepak Vohra 👻	Global +	Support •
Create Role	Set Role Nam	e				
Step 1 : Set Role Name	Enter a role name. You can	not edit the role name after the role is created.				
Step 2 : Select Role Type	Role Name	dockercloud-swarm-role				
Step 3 : Establish Trust		Maximum 64 characters. Use alphanumeric and '+=,@' characters				
Step 4 : Attach Policy						
Step 5 : Review						

Figure 14-3. Specifying a role name

The Select Role Type page is displayed, as shown in Figure 14-4. As we are linking two services—Docker Cloud and Docker for AWS—we do not need to select an AWS service role.

reate Role	Select Role Type	
tep 1 : Set Role Name	AWS Service Roles	
tep 2 : Select Role Type tep 3 : Establish Trust	Amazon EC2 Allows EC2 instances to call AWS services on your behalf.	Select
tep 4 : Attach Policy tep 5 : Review	AWS Directory Service Allows AWS Directory Service to manage access for existing directory users and groups to AWS services.	Select
	AWS Lambda Allows Lambda Function to call AWS services on your behalf.	Select
	Amazon Redshift Allows Amazon Redshift Clusters to call AWS services on your behalf	Select
	Amazon API Gateway Allows API Gateway to call AWS resources on your behalf.	Select
	© Role for Cross-Account Access	
	© Role for Identity Provider Access	

Figure 14-4. Select the role type

Select Role for Cross-Account Access, as shown in Figure 14-5, and select the sub-choice called Provide Access Between Your AWS Account and a 3rd Party AWS Account using the Select button.

CHAPTER 14 USING SWARM MODE IN DOCKER CLOUD

Create Role	Select Role Type			
Step 1 : Set Role Name	O AWS Service Roles			
Step 2 : Select Role Type Step 3 : Establish Trust	Role for Cross-Account Access			
Step 4 : Attach Policy	Provide access between AWS accounts you own Allows IAM users from one of your other AWS accounts to access this account.		Se	lect
	Provide access between your AWS account and a 3rd party AWS account Allows IAM users from a 3rd party AWS account to access this account and enforces use of External ID.		Se	lect

Figure 14-5. Role for cross-account access

Next, specify the account ID of the third party AWS account whose IAM users will access the AWS account. A third-party AWS account has been set up for the Docker Cloud service and has an account ID of 689684103426, which may be used by anyone (AWS user) linking Docker Cloud service to their AWS account. Specify the account ID as 689684103426, as shown in Figure 14-6. The external ID is a user's Docker ID for the Docker Cloud service account created at https://cloud.docker.com/. While the account ID will be the same (689684103426) for everyone, the external ID will be different for different users. Keep the Require MFA checkbox unchecked. Click on Next Step.

🧊 Services - Resour	ce Groups 👻 🕏					Φ	Deepak Vohra 🗸	Global +	Support +
Services Resour Create Role Step 1: Set Role Name Step 2: Select Role Type Step 3: Establish Trust Step 4: Attach Policy Step 5: Review	ce Groups 🔹 🖈 Enter the ID of the 3rd party A About the External ID. Account ID: External ID: Require MFA:	WS account whose IAM user 689684103426 dvohra	rs will be able	to access this accou	int. Enter the externa	A I ID prot	Deepsk Vohra -	Global •	Support -
							Cancel Pres	vious	Next Step I-

Figure 14-6. Specifying account and external IDs

As we are embedding a custom policy, do not select from any of the listed policies in Attach Policy. Click on Next Step, as shown in Figure 14-7.

Create Role	Atta	ch P	Policy			
step 1 : Set Role Name step 2 : Select Role Type	Select	one o	r more policies to attach. Each role c	an have up to 10 policies attache	ed.	
tep 3 : Establish Trust	Filte	r: Po	olicy Type + Filter			Showing 269 result
tep 5 : Review			Policy Name ¢	Attached Entities \$	Creation Time \$	Edited Time ¢
	0	Ū.	AdministratorAccess	0	2015-02-06 10:39 PDT	2015-02-06 10:39 PDT
		Û	AmazonAPIGatewayAdministr	0	2015-07-09 10:34 PDT	2015-07-09 10:34 PDT
		Û	AmazonAPIGatewayInvokeFul	0	2015-07-09 10:36 PDT	2015-07-09 10.36 PDT
		ũ	AmazonAPIGatewayPushToCI	0	2015-11-11 15:41 PDT	2015-11-11 15:41 PDT
		Û	AmazonAppStreamFullAccess	0	2015-02-06 10:40 PDT	2015-02-06 10:40 PDT
		0	AmazonAppStreamReadOnlyA	0	2015-02-06 10:40 PDT	2016-12-07 13:00 PDT
		Û	AmazonAppStreamServiceAcc	0	2016-11-18 20:17 PDT	2016-11-18 20:17 PDT
		Ũ	AmazonAlhenaFullAccess	0	2016-11-30 08:46 PDT	2016-11-30 08:46 PDT
		Û	AmazonCloudDirectoryFullAcc	0	2017-02-24 16:41 PDT	2017-02-24 16:41 PDT
	0	-	AmazanClaudDirastan/DaadO	n	2017 02 29 45-42 007	2017 02 29 15-12 DOT

Figure 14-7. Do not select a policy

On the Review page, click on Create Role, as shown in Figure 14-8.

Create Role	Review				
Step 1 : Set Role Name Step 2 : Select Role Type	Review the following ro finish. Role Name	le information. To edit the role, click an edit link, or cl dockercloud-swarm-role	ick Create Role to Edit Role Name		
step 3 : Establish Trust Step 4 : Attach Policy	Role ARN	arn.aws.iam.:672593526685.role/dockercloud -swarm-role			
tep 5 : Review	Trusted Entities	The account 689684103426			
	Policies		Change Policies		
	Give this link to users who can switch roles in the console	https://signin.aws.amazon.com/switchrole? account=672593526685&roleName=Gockercl oud-swarm-role	Copy Link		

Figure 14-8. Creating a role

A new AWS IAM role called dockercloud-swarm-role is created, as shown in Figure 14-9. Click on the **dockercloud-swarm-role** role name.

🎁 Services ~	R	esource Groups 👻 🕏	Д. Deepak Vohra → Global → Support →
Search IAM	Ċ	Create New Role Role Actions -	C 0 0
Dashboard		Filler	Showing 1 results
Groups Users		Role Name \$	Creation Time ©
Roles	N	dockercloud-swarm-role	2017-03-16 12:22 PDT
Policies	45		
Identity providers			
Account settings			
Credential report			
Encryption keys			

Figure 14-9. New role

Next, we will add an embedded (also called an inline) policy. The Permissions tab should be selected by default. Click on the v icon to expand the Inline Policies section, as shown in Figure 14-10.

🔰 Services 🗸	Resource Groups 👻 🛠		🗘 Deepak Vohra 🗸 Global 🗸 Support
Search IAM	IAM > Roles > dockercloud-swarm-r	ble	
Dashboard	Role ARN	arn:aws:lam::672593526685:role/dockercloud-swarm-role	
Groups	Instance Profile ARN(s)		
Jsers	Path	1	
toles	Creation Time	2017-03-16 12:22 PDT	
Policies dentity providers	Give this link to users who can switch roles in the console	https://signin.aws.amazon.com/switchrole? account=672593526685&roleName=dockercloud-swarm-role	Copy Link
ccount settings redential report	Permissions Trust Relationsh	ps Access Advisor Revoke Sessions	
ncryption keys	Managed Policies		^
and a second first	There are no managed policies a Attach Policy	tached to this role.	
	Inline Policies		X

Figure 14-10. Expanding the inline policies

To start, no inline policies are listed. Click on the Click Here link to add an inline policy, as shown in Figure 14-11.

🎁 Services 🗸	Resource Groups	s ~ 1€			¢	Deepak Vohra -	Global +	Support	*
Search IAM	- Summary								
Dashboard Groups Users Roles Policies	Role ARN Instance Profile Path Creation Time Give this link to switch roles in	: ARN(s) o users who can the console	arn:aws:lam::672593 / 2017-03-16 12:22 P https://signin.aws.an account=672593526	3526685:role/dockercloud-swarm-role DT nazon.com/switchrole? 6855aroleName=dockercloud-swarm-role		Copy Link			
Account settings	Permissions	Trust Relationships	Access Advisor	Revoke Sessions					
Encryption keys	Managed There are r Attach F	Policies no managed policies attach Policy	ed to this role.					^	
	Inline Poli	cies						^	
	There are r	no inline policies to show. T	o create one, click her	<u>e</u> ,					

Figure 14-11. Click on the Click Here link to add an inline policy

In Set Permissions, select Custom Policy using the Select button, as shown in Figure 14-12.



Figure 14-12. Selecting a custom policy

A policy document lists some permissions and the policy document for an IAM role to use Docker for AWS may be obtained from https://docs.docker.com/docker-for-aws/iam-permissions/. Click on Validate Policy to validate the policy, as shown in Figure 14-13.

Manage Role Permissions	Review F	olicy
	Customize per IAM guide. To Policy Name	missions by editing the following policy document. For more information about the access policy language, see Overview of Policies in the Usir test the effects of this policy before applying your changes, use the IAM Policy Simulator.
	dockercloud	swarm-policy
	Policy Docum	ient .
	2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17	<pre>"Version": "2012-10-17", "Statement": [</pre>

Figure 14-13. Validating the policy

Click on Apply Policy, as shown in Figure 14-14.

anage Role Permissions	Review Policy
	Customize permissions by editing the following policy document. For more information about the access policy language, see Overview of Policies in the Usi IAM guide. To test the effects of this policy before applying your changes, use the IAM Policy Simulator.
	This policy is valid.
	Policy Name
	dockercloud-swarm-policy
	Policy Document
	<pre>1. {</pre>
	15 "cloudformation:DescribeChangeSet",

Figure 14-14. Applying the policy
A new inline policy is added for the dockercloud-swarm-role role, as shown in Figure 14-15.

	Cr	eation Time		2017-03-16 12:22 F	TOT		
Search IAM	G	ve this link to	users who can	https://signin.aws.ar	mazon com/switchrole?	Conviliate	
Dashboard	sv	itch roles in t	the console	account=67259352	6685&roleName=dockercloud-swarm-role	CODY LINK	
Groups	-		1				
Users	1	Permissions	Trust Relationships	Access Advisor	Revoke Sessions		
loles							
olicies		Managed H	Policies		^		
ientity providers		There are n	o managed policies attact	ned to this role.			
Account settings		Attach P	olicy				
Credential report							
		Inline Polic	cies				^
ncryption keys		This view sh	nows all inline policies that	are embedded in this	role		
		Create R	ala Policy				
		Greate R	ore Policy				
		Policy Nar	me	Action	5		

Figure 14-15. The new inline policy is added

Copy the Role ARN String listed in Figure 14-16, as we need the ARN string to connect to the AWS Cloud provider from Docker Cloud.

earch IAM	IAM > Roles > dockercloud-swarm-ro • Summary	le				
Dashboard Bro <mark>up</mark> s	Role ARN Instance Profile ARN(s)	arn:aws:iam::672593526685:role/dockercloud-swarm-role				
Jeers	Path					
Roles	Creation Time	2017-03-16 12:22 PDT				
Policies dentity providers	Give this link to users who can switch roles in the console	https://signin.aws.amazon.com/switchrole? account=672593526685&roleName=dockercloud-swarm-role	Copy Link le			
ccount settings credential report	Permissions Trust Relationship	s Access Advisor Revoke Sessions				
non-millen have	Managed Policies			^		
ncrypuon keys	There are no managed policies att Attach Policy	ached to this role.				
	Inline Policies			^		
	This view shaws all inline policies t	ant are embedded in this rate				

Figure 14-16. Role ARN

Creating a Docker Swarm in Docker Cloud

In this section, we create a Docker Swarm from the Docker Cloud service. Log in to the Docker Cloud service at https://cloud.docker.com/. The Cloud registry page should be displayed at https://cloud.docker.com/. The Cloud registry page should be displayed at https://cloud.docker.com/. The Cloud registry page should be displayed at https://cloud.docker.com/. The Cloud registry page should be displayed at https://cloud.docker.com/. The Cloud registry page should be displayed at https://cloud.docker.com/ app/dvohra/dashboard/onboarding/cloud-registry. A Swarm Mode option is available in the margin and it's off by default, as shown in Figure 14-17.



Figure 14-17. The Swarm Mode slider

Click on the Swarm Mode slider; the Swarm mode should be enabled, as shown in Figure 14-18.

\leftrightarrow \times	Secure https://cloud.docker.com/app/dvohra/dashboard/onboarding/cloud-registry	e o :
=	DOCKER + Get Help - Q dvot	nra 👻
Swarm Mode	U Switching to Swarm mode	
Ξ	Cloud registry Continuous Integration Application deployment Continuous deployment Teams & Organizations	

Figure 14-18. Switching to Swarm mode

A Swarms toolbar option is added, as shown in Figure 14-19.

← → C Secure http://www.com/actions/actio	Docker Cloud × vs://cloud.docker.com/swarm/dvohra/dashboard/or	nboarding/cloud-registry					☆ ⊘
	Swarm mode	+	Repositories	Swarms	Get Help	- 0	dvohra 👻
Welcomet							
	Welco Let's get you famil	me to Dock	er Cloud!	oud.			

Figure 14-19. Swarms toolbar option

Two options are available—Bring Your Own Swarm or Create a New Swarm. Click on Create to create a new Swarm, as shown in Figure 14-20.

The Docker Platform X Docker Cloud X						
← → C Secure https://cloud.docker.com/swarm/dvohra/swar	/list/1?page_size=10					★ eí O i
DOCKER Swarm mode		Repositories	Swarms	Get Help 👻	Q dvohra	•
Swarms						
				Polosia	Const.	
				Bring your own s	swarm	
A Swarm is a clu: <u>Click here to learn</u>	er of Docker Engines where you deploy s nore	ervices.				

Figure 14-20. Creating a new Swarm

Next, we will configure the Swarm, including specifying a Swarm name, selecting a cloud provider, and selecting cloud provider options. Two Cloud service providers are supported: Amazon Web Services (AWS) and Microsoft Azure (not yet available). We use AWS in this chapter. We need to configure the cloud settings for AWS with the ARN string we copied earlier. Cloud settings may be configured with one of the two options. One option is to select Cloud Settings from the account, as shown in Figure 14-21.



Figure 14-21. Cloud settings

In the Cloud Settings page, click on the plug icon that says Connect Provider for the Amazon Web Services provider, as shown in Figure 14-22.

		+ 8	epositories	Swarms	Get Help	* Q	dvohra 👻
Cloud Settings							
General Service providers Source providers Notifications	dvohra Member since Oct 0	2, 2015					
Default Privacy Billing Plan	Service providers Amazon Web Services Add	new credentials			Connect p	rovider Free I	er
	Source providers Provider Github	Account		¢	5		
	Bitbucket	No account linked		¢	<i>14</i>		

Figure 14-22. Connecting the provider

The Add AWS Credentials dialog is displayed, as shown in Figure 14-23.

Aud AVVS Credentials	
Click here for instructions.	
Role delegation ARN	

Figure 14-23. Adding AWS credentials

The other option to configure the Cloud settings is to click on the Amazon Web Service Provider icon, as shown in Figure 14-24, which also displays the Add AWS Credentials dialog.

🖌 🔹 The Docker P	Istform X 🗸 👉 Docker Cloud	×							The second second	ينصالغا	a 🗾 🗙 🚽
€ ∋ C 🛢	Secure https://cloud.docker.com	/swarm/dvohra/swarm/wizard								* 🖆	0 :
	DOCKER Swar				Repositories	Swarms	Get Help 👻	Q	dvohra 👻		
	Swarms Create					6					
	Swarm Name										
	dvohra										
	Service Provider										
	webservices-	Microsoft Azure									
	 not connected 	Coming soon									
	Powered by Docker CE for AWS										
	Docker Community Edition										
	Docker Community Edition (CE) is i Docker and experimenting with co	ideal for developers and small tea ntainer-based apps.	ms looking to get started	d with							
	Learn More										
								Cancel	Create	í.	

Figure 14-24. Connecting to an Amazon web services provider

Specify the ARN string copied earlier from the Add AWS Credentials dialog and click on Save, as shown in Figure 14-25.



Figure 14-25. Saving the AWS credentials

With either option, the service provider Amazon Web Services should be connected, as indicated by the Connect Provider icon turning to Connected, as shown in Figure 14-26.

	Swarm mode	+	Repositories	Swarms BETA	Get Help	0	dvohra 👻
Cloud Settings							
General							
Service providers	dvohra Member since Oct 02, 2015						
Source providers	0						
Notifications							
Default Privacy	Service providers						
Billing	Amazon Web Services arraysticiam::672502526	695 rola/do	ckarcloud swarm ro			Free Tier	
Plan	annavian.orzazaza	1005.1016/00	ener croud-smallin-to			1155 1161	

Figure 14-26. Amazon Web Services provider in connected mode

The Amazon Web Services option should indicate connected, as shown in Figure 14-27.

DOCKER SW		+	Repositories	Swarms	Get Help 🔫	🚺 dvohra
Swarms Create))			
Swarm Name						
dvohra	/ Name					
Service Provider						
wamazon	Microsoft Azure					
webservices™ connected	Coming soon					
Powered by Docker CE for AWS						
Docker Community Edition						
Docker Community Edition (CE) i Docker and experimenting with	s ideal for developers and small teams looking container-based apps.	to get started with				
Learn More						

Figure 14-27. Amazon Web Services provider connected

Specify a Swarm name. That name should not include any spaces, capitalized letters, or special characters other than ",", "-" and "_", as shown in Figure 14-28.

UDCKER Swarm mod	je
Swarms Create	
Swarm Name	
dvohra:	/ DockerCloudSwarm
	No spaces, capitalized letters, and specie characters other than ., _, or - are allowe

Figure 14-28. Specifying a Swarm name

Specify a valid Swarm name (docker-cloud-swarm), select the Amazon Web Services Service provider, which is already connected, and click on Create, as shown in Figure 14-29.

	DOCKER Sw	arm mode	+	Repositories	Swarms	Get Help 🔻	🚺 dvohra 👻	
	Swarms Create							
	Swarm Name							
	dvohra							
	Service Provider							
De	webservices~	Microsoft Azure						
	Powered by Docker CE for AWS							
	Docker Community Edition							
	Docker Community Edition (CE) is Docker and experimenting with o	i ideal for developers and small teams lookin ontainer-based apps.	ng to get started with					

Figure 14-29. Creating a Docker Swarm using the AWS service provider

CHAPTER 14 USING SWARM MODE IN DOCKER CLOUD

In the region, select a region (us-east-2), the number of Swarm managers (3), the number of Swarm workers (5), the Swarm manager instance type (t2.micro), the agent worker instance type (t2.micro), and the SSH key. Click on Create, as shown in Figure 14-30.

Region	
us-east-2 👻	
Region where your swarm is provisioned	
Swarm Size	
Number of Swarm managers?	Number of Swarm worker nodes?
Number of swarm manager nodes needs to be an odd number	Number of swarm worker nodes (0 - 1000)
Swarm Properties	
Swarm manager instance type?	Agent worker instance type?
t2.micro 👻	t2.micro 👻
EC2 HVM instance type (t2.micro, m3.medium, etc)	EC2 HVM instance type (t2.micro, m3.medium, etc)
Which ssh key to use?	
docker 👻	
Name of an existing EC2 KeyPair to enable SSH access to the instance	
	Cancel Create

Figure 14-30. Configuring and creating a Swarm

The Swarm should start to get deployed, as indicated by the DEPLOYING message shown in Figure 14-31.

UDCKER Swarm mode	+	Repositories	Swarms BETA	Get Help	- 6	dvohra 👻
Swarms						
				Bring your	own swarm	Create
dvohra/docker-cloud-swarm DEPLOYING			O a few secor	ids ago		ï
	1					

Figure 14-31. Deploying a Swarm

When the Swarm has been deployed, the message becomes Deployed, as shown in Figure 14-32.

ä	DOCKER Swarm mode	+	Repositories	Swarms	Get Help	- 6	dvohra 👻
	Swarms						
					Bring your	own swarm	Create
Ð	dvohra/docker-cloud-swarm			⊙ 5 minutes a	igo		/ 11

Figure 14-32. The Swarm is now deployed

The AWS infrastructure for the Swarm is created and configured. A CloudFormation stack is created, as shown in Figure 14-33.

Cr	reate Stack Actions	Design template			C O
Fil	ter: Active - By Stack Nar	ne			Showing 1 stack
	Stack Name	Created Time	Status	Description	
	docker-cloud-swarm	2017-03-16 12:38:07 UTC-0700	CREATE COMPLETE	Docker for AWS 1 13 1 (ga-2)	

Figure 14-33. CloudFormation stack for the created Swarm

A new proxy AWS IAM role for the Swarm is added, as shown in Figure 14-34.

Search IAM	Crea	Role Actions +		C	٥
Dashboard	Filte			Show	ing 2 resu
Groups Users		Role Name ©	Creation Time ©		
Roles		docker-cloud-swarm-ProxyRole-MI55EFAJQQ1R	2017-03-16 12:38 PDT		
Policies		dockercloud-swarm-role	2017-03-16 12:22 PDT		
Identity providers Account settings Credential report					

Figure 14-34. Proxy role and Docker Cloud Swarm AWS role

CHAPTER 14 USING SWARM MODE IN DOCKER CLOUD

EC2 instances for the Swarm manager and worker nodes are started. Each EC2 instance is started with the proxy IAM role created automatically, as shown for a manager node in Figure 14-35.

Q,	Filter by tags and attributes or se	earch by keyword				0	K < 1	to 8 of 8	> :	>1
	Name 👻	Instance ID 🔺	Instance Type 👻	Availability Zone +	Instance State ~	Statu	s Checks	Alarm	Status	
	docker-cloud-swarm-worker	i-00bd2442d55f04ede	t2.micro	us-east-2b	running	o 2/	2 checks	None		7
	docker-cloud-swarm-Manag	i-03f1027044d22efb9	t2.micro	us-east-2a	running	2	2 checks	None		>
	docker-cloud-swarm-Manag	i-04f4ab6cf8f51672d	t2.micro	us-east-2b	running	O 21	2 checks	None		2
	docker-cloud-swarm-worker	i-05169d85603d643d6	t2.micro	us-east-2a	running	2/	2 checks	None		>
	docker-cloud-swarm-worker	i-067a12b88c06044f0	t2.micro	us-east-2c	running	O 21	2 checks	None		2
	docker-cloud-swarm-worker	i-0ae14983fce64b264	t2.micro	us-east-2b	running	O 2/	2 checks	None		10
	docker-cloud-swarm-worker	i-0bc7fcf948af91805	t2.micro	us-east-2c	running	O 2/	2 checks	None		>
	docker-cloud-swarm-Manag	i-0fd1a015ad15203c2	t2.micro	us-east-2c	running	O 2/	2 checks	None		20
										,
	Platform			Network	interfaces eth0					-
	IAM role	docker-cloud-swarm-Pro MI55EFAJQQ1R	xyRole-	Source/de	est. check True					

Figure 14-35. IAM role for EC2 instances

Each Docker Cloud account namespace must be associated with only one AWS IAM role. If multiple Docker Cloud accounts are to access the same AWS account, multiple roles must be created for each Docker Cloud account or Docker Cloud account namespace. Each AWS IAM role for Docker Cloud to access AWS is associated with an ARN string. The ARN string for a deployed Swarm may be edited with the Edit Endpoint link, as shown in Figure 14-36.

UDCKER Swarm mode	+ Repositories Swarms Get Help + 💽 dvohra +
Swarms	
	Bring your own swarm Create
dvohra/docker-cloud-swarm	© 23 minutes ago

Figure 14-36. Edit Endpoint link

If the Swarm endpoint is to be modified, specify a new ARN string (for a different IAM role associated with a different Docker Cloud namespace) in the Edit Endpoint dialog. Click on Save, as shown in Figure 14-37.

dit endpoint fo	r dvohra/docker-cloud	-swarm	
and entrapolitient			
f left empty, Docker ('ou can override this	loud uses the auto-detected publi value by specifying an IP address o	lic IP of the manager nodes to connect to the Sw or a hostname in the field below.	arm.
locker-cloud-swarm	-ELB-226413314.us-east-2.elb.a	amazonaws.com	
		Cancel	ave

Figure 14-37. Editing the endpoint

Next, we connect to the Docker Swarm. There are two ways to do so:

- Connect directly from any Docker host
- Obtain the public IP address of a Swarm manager from the EC2 dashboard and SSH login to the Swarm manager

We discuss each of these options.

Connecting to the Docker Swarm from a Docker Host

Click on the Docker Swarm in the Docker Cloud dashboard. The Connect To dialog should be displayed with a docker run command, as shown in Figure 14-38. Copy the docker run command.



Figure 14-38. Listing and copying the docker run command to connect to the Swarm

Start an EC2 instance with CoreOS AMI, which has Docker pre-installed, as shown in Figure 14-39.

	a starting		g	accord croop		
tep 1: Choose an A AMI is a template that contain NS, our user community, or the	Amazon Mac s the software config AWS Marketplace; o	chine Image juration (operating s or you can select on	e (AMI) system, application server e of your own AMIs.	, and applications) required to	launch your instance	Cancel and Exit
Quick Start	O coreos		×			$ \langle \langle 1 \text{ to 6 of 6 Products} \rangle \rangle $
My AMIs	Ci corcos		~			
AWS Marketplace	Container linux	Container Linux	by CoreOS (Stable) 0 Previous versions Sold by C	oreOS		Select
Community AMIs	Free tier eligible	\$0.00/hr for software - Linux/Unix, Other 1298	+ AWS usage fees 5.0 64-bit Amazon Machine Ima	ge (AMI) Updated: 3/2/17		
Categories		CoreOS Container L containers running o	inux automates software up on large-scale clusters. Oper	dates to ensure better security a rating system updates	and reliability of machin	es and
Software Infrastructure (6)		More info				

Figure 14-39. Creating an EC2 instance with CoreOS AMI

Obtain the public IP address of the CoreOS instance from the EC2 console, as shown in Figure 14-40.

Launc	h Instance	Connect	Actions V							The second se	₽, 4	F.
Q, s	search : i-0e5	525ed8b79fc52b0	O Add filter					0	< <	1 to 1 o	f1 >	×
	Name			▲ Instan	ce ID 🔹	Instance Type 🔹	Availability	Zone - Insta	nce State	- Stat	us Che	cks
	CoreOS			i-0e525	ed8b79fc52b0	t2.micro	us-east-1c	🥥 n	unning	×	Initiali	zing
					_							
Instan	ice: i-0e52	25ed8b79fc52b	0 (CoreOS)	Public DNS	: ec2-34-207-2	20-127.compute-1.	amazonaws.	.com				
Instan	ription	25ed8b79fc52b Status Checks	0 (CoreOS) Monitoring	Public DNS Tags	: ec2-34-207-2 Usage Instruc	20-127.compute-1.	amazonaws	.com			88	
Instan Desci	ription	25ed8b79fc52b Status Checks Instance ID	0 (CoreOS) Monitoring i-0e525ed8b79f	Public DNS Tags fc52b0	: ec2-34-207-2 Usage Instruc	20-127.compute-1.	amazonaws. DNS (IPv4)	.com ec2-34-207-220 1.amazonaws.c)-127.compu com	I te-	88	

Figure 14-40. Displaying EC2 instance detail

SSH login to the CoreOS instance.

ssh -i "coreos.pem" core@34.207.220.127

Run the command copied earlier to connect to the Docker Swarm.

docker run --rm -ti -v /var/run/docker.sock:/var/run/docker.sock -e DOCKER_HOST dockercloud/ client dvohra/docker-cloud-swarm The dockercloud/client Docker image that's used to connect to Docker Cloud is downloaded. A username and password prompt should be displayed. Specify the username and password for the Docker Cloud account in which the Swarm was created.

```
Container Linux by CoreOS stable (1298.5.0)

$ docker run --rm -ti -v /var/run/docker.sock:/var/run/docker.sock -e DOCKER_HOST

dockercloud/client dvohra/docker-cloud-swarm

Unable to find image 'dockercloud/client:latest' locally

latest: Pulling from dockercloud/client

b7f33ccOb4Be: Pull complete

91b7430c5c68: Pull complete

19aaa3obba7a: Pull complete

19aaa3obba7a: Pull complete

Digest: sha2S6: 11d3cc5e1a62c7324]2a6e038]ccffi9]53tc91d0b1c69c8D1d3b68629337558a6

Status: Downloaded newer image for dockercloud/client:latest

Use your Docker ID credentials to authenticate:

Username: dvohra

Password:
```

A export command is output to connect to the Swarm. Copy the command.

```
Use your Docker ID credentials to authenticate:
Username: dvohra
Password:
=> You can now start using the swarm dvohra/docker-cloud-swarm by executing:
export DOCKER HOST=tcp://127.0.0.1:32768
```

Run the command. The Swarm is connected to the CoreOS Docker host. List the Swarm nodes using the docker node 1s command.

>export DOCKER HOST=tcp://127.0.0.1:32768

```
>docker node ls
ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS
```

liuomlmb6n6xtq4apxayumsx3 ip-172-31-0-251.us-east-2.cornpute.internal. Ready Active bchea5x85m82jtzoq336trn8y ip-172-31-47-61.us-east-2.compute.internat. Ready Active e2b1785z5pqouakdceomdpsbi ip-172-31-42-130.us-east-2.compute.internal. Ready Active hzxb8choml.7gylaqtrjrh6phx ip-172-31-26-90.us-east-2.compute.internal. Ready Active pcnple9l29w88ueonhdwUcoc ip-172-31-27-18.us-east-2.compute.internal. Ready Active rupjaojommfchjgcshffdobhf * ip-172-31-10-153.us-east-2.compute.internal Ready Active Leader uyl5xv7mhb6c8jam5ofncplyh ip-172-31-25-137.us-east-2.compute.internal. Ready Active Reachable wi6zurda4nawf9mgku3enf6io ip-172-31-34-33.us-east-2.compute.internal Ready Active Reachable

Connecting to the Docker Swarm from a Swarm Manager

The other option is to connect to a Swarm manager using its public IP address. First, we obtain the public IP address of a Swarm manager from the EC2 console, as shown in Figure 14-41.

Q Filter by tag	s and attributes or se	arch by keyword				0 K < 1	to 8 of 8	> >
Name	*	Instance ID 🔺	Instance Type 🔹	Availability Zone -	Instance State	Status Checks ~	Alarm St	atus
docker-clo	ud-swarm-worker	i-00bd2442d55f04ede	t2.micro	us-east-2b	running	2/2 checks	None	1
docker-clo	ud-swarm-Manag	i-03f1027044d22efb9	t2.micro	us-east-2a	running	2/2 checks	None	1
docker-clo	ud-swarm-Manag	i-04f4ab6cf8f51672d	t2.micro	us-east-2b	running	2/2 checks	None	1
docker-clo	ud-swarm-worker	i-05169d85603d643d6	t2.micro	us-east-2a	running	2/2 checks	None	1
docker-clo	ud-swarm-worker	i-067a12b88c06044f0	t2.micro	us-east-2c	running	2/2 checks	None	1
docker-clo	ud-swarm-worker	i-0ae14983fce64b264	t2.micro	us-east-2b	running	2/2 checks	None	1
docker-clo	ud-swarm-worker	i-0bc7fcf948af91805	t2.micro	us-east-2c	running	2/2 checks	None	1
docker-clo	ud-swarm-Manag	i-0fd1a015ad15203c2	t2 micro	us-east-2c	🥥 running	2/2 checks	None	1
Description	Status Checks	Monitoring Tage	5					
	Instance ID	i-03f1027044d22efb9		Public D	NS (IPv4) ec2-5	2-14-146-223.us-east-		
					2.com	pute.amazonaws.com		
	Instance state	running		De IPv4	Public IP 52.14	.146.223		
	Instance state Instance type	running t2.micro		₽ IPv4	Public IP 52.14 IPv6 IPs -	146.223		
	Elastic IPs			Pri	vate DNS ip-172	-31-10-153.us-east-		

Figure 14-41. Obtaining the public IP of a Swarm manager

SSH login into the Swarm manager.

ssh -i "docker.pem" docker@52.14.146.223

The Swarm manager is logged in and the Swarm command prompt is displayed.

```
[root@1ocathost -]# ssh -i "docker.pem" docker@52.14.146.223
The authenticity of host 52.14.146.223 (52.14.146.223)1 cant be established.
RSA key fingerprint is e9:7f:d2:3c:de:6d:5d:94:06:e2:09:56:b7:2a:c6:9a.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '52.14.146.223 (RSA) to the list of known hosts.
Welcome to Docker!
```

List the Swarm nodes using the docker node 1s command.

Welcome to Docker!

~ \$ docker node l.s

ID HOSTNAME STATUS

AVAILABILITY MANAGER STATUS

liuomlmb6n6xtq4apxayumsx3 ip-172-31-0-251.us-east-2.compute.internal Ready Active bchea5x85m82jtzoq336trn8y ip-172-31-47-61.us-east-2.compute.internal Ready Active e2b1785z5pqouakdceonidpsbi ip-172-31-42-130.us-east-2.compute.internal Ready Active hzxb8chomt7gyl.aqtrj rh6phx ip-172-31-26-90.us-east-2.compute.internal Ready Active pcnple9l29w88ueenhdwflcoc ip-172-31-27-18.us-east-2.compute.internal Ready Active rupjaejommfchjgcshffdobhf * ip-172-31-10-153.us-east-2.compute.internal. Ready Active Leader uyl5xv7mhb6c8jain5ofncplyh ip-172-31-25-137.us-east-2.compute.internal. Ready Active Reachable wi6zurda4nawf9mgku3enf6ie ip-172-31-34-33.us-east-2.compute.internal Ready Active Reachable

Create a service using the docker service create command and list the service with docker service 1s.

```
docker service create \
    --name hello-world \
    --publish 8080:80 \
    --replicas 1 \
    tutum/hello-world
```

The hello-world service is created. A Docker Cloud server proxy service is also listed.

~ \$ docker service create \

- > --name hello-world \
- > --publish 8080:80 \
- > - replicas 1 \
- > tutum/hello-world

hbiejbua8u5øskabun3dzkxk4

~ \$ docker service 1s

ID NAME MODE REPLICAS IMAGE

Ogzua3p56myx dockerdoud-server-proxy global 3/3 dockercioud/server-proxy:latest

hbiejbua8u50 hello-world replicated 1/1 tutum/hello-world:latest

Bringing a Swarm into Docker Cloud

Docker Cloud Swarm mode also has the provision to import an existing Swarm into Docker Cloud. The Swarm to be imported must have the following prerequisites:

- Based on Docker Engine 1.13 or later nodes
- Swarm manager incoming port 2376 unblocked

In this section, we create a Swarm and import the Swarm into Docker Cloud. First, run the docker --version command to determine if the Docker host version is 1.13 or later. One of the EC2 instances provisioned by Docker for AWS may be used to create and import a Swarm, as the Docker version on the custom Linux distribution is > Docker 1.13; the node must be made to leave the Swarm before creating a new Swarm. Using the private IP address of the EC2 instance, initiate a new Swarm.

```
docker swarm init --advertise-addr 172.31.23.196
```

Copy the docker swarm join command output to join the worker nodes.

~ \$ docker --version

Docker version 17.03.0-ce, build 60ccb22

~ \$ docker swarm init --advertise-addr 172.31.23.196

Swarm initialized: current node (ylzc3h3slxO5ztbujtl3yf86p) is now a manager.

To add a worker to this swarm, run the following command:

docker swarm join \

--token SWMTKN-1-23snf1iuieafnyd1zzgf37ucwuz1.khg9atqsmysmvv6iw1.arw0-do29n83jptkkdwss5fjsd3rt \

172.31.23.196:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

Join a worker node on another EC2 instance with Docker 1.13 or later.

docker swarm join ∖

--token SWMTKN-1-61gcsgkr1ildxz580ftdl3rq0s9p7h30n12byktgvbd6y3dk7r-cpes7ofdsq8abhxtznh92tjrz \ 10.0.0.176:2377

The worker node joins the Swarm.

A Swarm with two nodes is created, as listed in the output to the docker node 1s command, which runs on the Swarm manager node.

~\$ docker node 1s

HOSTNAME STATUS

AVAILABILITY MANAGER STATUS

trgb2t4ehs2gp3cjbrnqhs7a5 ip-172-31-6-64.us-east-2.compute.internal. Ready Active

yl.ic3h3stxo5ztbujtl3yf86p ip-172-31-23-196.us-east-2.compute.internal Ready Active Leader

~\$

Next, import the Swarm into Docker Cloud. From the Swarm manager node, run the following command.

docker run -ti --rm -v /var/run/docker.sock:/var/run/docker.sock dockercloud/registration

Specify the Docker ID at the username prompt and the password at the password prompt.

```
~ S docker run -ti --rm -v /var/run/docker.sock:/var/run/docker.sock dockercloud/
registration
Unable to find image dockercloud/registration:latest' locally
latest: Pulling from dockercloud/registration
b7f33cc0b48e: Pull complete
b52875cf8fd4: Pull complete
23f82c866468: Pull complete
Digest: sha256: a3f39de96d2763b957e7bel22ce99b8lfbba03fbd6b2e54bd607lcafbelcabcl
Status: Downloaded newer image for dockercloud/registration:latest
Use your Docker ID credentials to authenticate:
Username: dvohra
Password:
```

Specify a cluster name for the Swarm imported into Docker Cloud, or use the default. Specify cluster as dvohra/dockercloudswarm. The Swarm is registered with Docker Cloud. As for a Swarm created in the Docker Cloud Swarm mode, the Swarm may be accessed from any Docker host for which a command is output.

Enter name for the new cluster [dvohra/wkhøtlq8cw5u44x22qp6r4eau]: dvohra/dockercloudswarm

You can now access this cluster using the following command in any Docker Engine

docker run -rm -ti -v /var/run/docker.sock:/var/run/docker.sock -e DOCKER HOST dockerctoud/
client dvohra/dockerctoudswarm

To bring the Swarm into Docker Cloud, click on the Bring Your Own Swarm button in Swarm mode, as shown in Figure 14-42.

CHAPTER 14 USING SWARM MODE IN DOCKER CLOUD

dvohra/docker-cloud-swarm © 2 hours ago	UDCKER Swarm mode	+ Repositories Swarms Get Help - 👥 dvohra
dvohra/docker-cloud-swarm © 2 hours ago	Swarms	
dvohra/docker-cloud-swarm © 2 hours ago REMOVED		Bring your own swarm Create
dvohra/docker-cloud-swarm © 2 hours ago REMOVED		
REMOVED	dvohra/docker-cloud-swarm	© 2 hours ago
	REMOVED	O 2 hours ago

Figure 14-42. Bring your own Swarm

The Swarm registered with Docker Cloud is added to the Docker Cloud Swarms, as shown in Figure 14-43.

UDOCKER Swarm mode	+ Repositories Swarms Get Help + 👤	dvohra 👻
Swarms		
	Bring your own swarm	Create
dvohra/swarm-coreos UNAVAILABLE	© Z hours ago	/ 11
dvohra/dockercloudswarm	O a few seconds ago	/ 1

Figure 14-43. Docker Cloud Swarms, including the imported Swarm

Summary

This chapter introduced the Docker Cloud Swarm mode, which is a managed service for linking the Docker Cloud managed service to a AWS service provider account and provisioning a Swarm from Docker Cloud. A Swarm created on the command line can be imported into Docker Cloud. In the next chapter we discuss Docker service stacks.

CHAPTER 15

Using Service Stacks

The Docker Swarm mode is Docker-native as of Docker 1.12 and is used to create distributed and scalable services for developing Docker applications.

The Problem

While single Docker image applications are also commonly used, a vast majority of Docker enterprise applications are comprised of multiple images that have dependencies between them. Docker Compose (standalone in v1 and v2) could be used to declare dependencies between microservices using the links and depends_on options, but Compose (standalone) is archaic, other than the format for defining services, in the context of Swarm mode services.

The Solution

Docker Swarm mode has introduced service *stacks* to define a collection of services (Swarm mode services) that are automatically linked with each other to provide a logical grouping of services with dependencies between them. Stacks use stack files that are YAML files in a format very much like the docker-compose.yml format. There are a few differences such as the absence of links and depends_on options that were used to define dependencies between microservices in Docker Compose (standalone). YAML (http://www.yaml.org/) is a data serialization format commonly used for configuration files.

As of Docker v1.13, the docker stack subset of commands has been introduced to create a Docker stack. Using a *stack file* that defines multiple services, including services' configuration such as environment variables, labels, number of containers, and volumes, a single docker stack deploy command creates a service stack, as illustrated in Figure 15-1. The services are automatically linked to each other.

CHAPTER 15 USING SERVICE STACKS



Figure 15-1. Service stack created with the docker stack deploy command

Docker Compose versions 3.x and later are fully Docker Swarm mode compatible, which implies that a Docker Compose v3.x docker-compose.yml file could be used as a Stack file except for a few suboptions (including build, container_name, external_links, and links) that are not supported in a stack file. Docker Compose 3.x could still be used standalone to develop non-Swarm mode services, but those microservices are not usable or scalable with the Docker Swarm mode docker service group of commands. To use stacks to manage Swarm mode services, the following requirements must be applied.

- Docker version must be 1.13 or later
- Swarm mode must be enabled
- Stack file YAML format must be based on Docker Compose v3.x file format

To use service stacks, the Docker Compose version 3 YAML file format is used, but Docker Compose is not required to be installed.

When using Docker Swarm mode, the Docker version requirement for Swarm mode is 1.12 or later. Before developing stacks to manage Swarm mode services, verify that the Docker version is at least 1.13. The Docker version used in this chapter is 17.0x. The docker stack group of commands listed in Table 15-1 becomes available in Docker v1.13 and later.

Description
Deploys a service stack or updates an existing stack
Lists the stacks
Lists the Swarm mode tasks in a stack
Removes a stack
Lists the Swarm mode services in a stack

Table 15-1. The docker stack Commands

Run the docker --version command to list the Docker version. To list the commands for stack usage, run the docker stack command.

```
[root@localhost ~]# ssh -i "docker.pem" docker@34.205.43.53
Welcome to Docker!
~ $ docker --version
Docker version 17.06.0-ce, build 02c1d87
~ $ docker stack
Usage:
          docker stack COMMAND
Manage Docker stacks
Options:
      --help
               Print usage
Commands:
  deploy
              Deploy a new stack or update an existing stack
  ls
              List stacks
              List the tasks in the stack
  ps
              Remove one or more stacks
  rm
  services
              List the services in the stack
```

To use stacks, the following procedure is used.

- 1. Install Docker version 1.13 or later (not Docker version 1.12, which is used in several of the earlier chapters).
- 2. Enable Swarm mode.
- 3. Create a Stack file using Docker Compose (version 3.x) YAML format.
- 4. Use the docker stack group of commands to create and manage the stack.

The chapter creates a service stack consisting of two services, one for a WordPress blog and another for a MySQL database to store the data in the WordPress blog.

Setting the Environment

We use Docker for AWS available at https://docs.docker.com/docker-for-aws/ to launch a Docker Swarm mode cluster of nodes. Docker for AWS uses the AWS CloudFormation template to create a Docker Swarm mode cluster. Click on the Deploy Docker Community Edition (stable), shown in Figure 15-2, to launch a Create CloudFormation Stack wizard to create a Docker Swarm mode cluster.



Deploy Docker Community Edition [CE] for AWS [edge] Deploy Docker Community Edition [CE] for AWS [test]

Figure 15-2. Deploying the Docker Community Edition for AWS (stable)

CHAPTER 15 USING SERVICE STACKS

Configure a Swarm using the Create Stack wizard as discussed in Chapter 3. You can specify the number of swarm managers to be 1, 3, or 5 and the number of Swarm worker nodes to be 1-1000. We used one Swarm manager node and two Swarm worker nodes, as shown in Figure 15-3.

Ciodal cimatici		
Create stack	Specify Details	
Options Review	Specify a stack name and parameter values. You can use or change the default parameter values, which are defined in the AWS CloudFormation template.	Learn more.
	Parameters	
	Swarm Size	
	Number of Swarm 1 vumber of Swarm manager nodes (1, 3, 5) managers?	
	Number of Swarm worker 2 Number of worker nodes in the Swarm (0-1000). nodes?	
	Swarm Properties	

Figure 15-3. Configuring a CloudFormation stack

The CloudFormation stack is created, as shown in Figure 15-4.

	Clou	udFormation	Stack:	S							
1	Create St	ack 🔸 A	ctions +	Design to	emplate						C O
	Filter: A	tive - By Sta	ick Name								Showing 1 stack
	Stac	k Name	Crea	ated Time		Status		De	escription		
	Dock	kerSwarm	2017	7-06-16 16:	49:42 UTC-07	00 CREAT	E_COMP	LETE DO	ocker for AWS 17.0	03.1-ce (aws2)	
		Stack name:	DockerSwarm								
		Stack ID:	arn:aws:cloudf	ormation:us	east-1:67259	3526685:stack/E	DockerSwa	arm/776fa8f0-52e	e-11e7-9480-500c	217dbe62	
		Status:	CREATE_COM	IPLETE							
	St	atus reason:									
		IAM Role:									
		Description:	Docker for AW	S 17.03.1-0	e (aws2)						

Figure 15-4. CloudFormation Stack for Docker on AWS

Three EC2 instances—one for Docker Swarm manager node and two for the Swarm worker nodes—are launched, as shown in Figure 15-5. The Linux distribution used by the CloudFormation stack is Moby Linux, as shown in Figure 15-5.

DNS (IPv4)									Add filter	: Running 💮	Instance State :	4
	Public DNS (tus	Alarm Sta	tatus Checks 👻	Sta	Instance State +	Availability Zone -	Instance Type 👻	nstance ID 🔹	× 1	Name	
227-151-152.co.	ec 2-34-227-15	6	None	2/2 checks	۲	running	us-east-1b	t2.micro	01242202c7b019e28	vorker i-	DockerSwarm-wo	
205-48-154.com	ec2-54-205-48	70	None	2/2 checks	0	running	us-east-1b	t2.micro	Obfc 9384db35835cd	Aanager i-	DockerSwarm-Ma	
227-191-199.co.	ec 2-34-227-19	6	None	2/2 checks	0	running	us-east-1c	t2.micro	0d89f7a7e2121a09e	vorker i	DockerSwarm-wo	
+						2						
-							0.0.0		inbound rules			
				vpc-4ccc6535	CID	VPC		5	No scheduled event	eduled events	Sched	
			54b) vpc-4ccc6535) subnet-671cb6	C ID et ID	VPC Subne		s 7.03.1-ce-aws2 (ami-	inbound rules No scheduled event Moby Linux aws-v1 3a81f12c)	eduled events AMI ID	Schee	

Figure 15-5. The Moby Linux AMI used for Docker on AWS

Before being able to use Docker on AWS, enable all inbound/outbound traffic between the EC2 instances in the security groups used by the EC2 instances. This is shown for the security group for Swarm manager node instance inbound rules in Figure 15-6.

									-					
-	Q search : :	sg-1bbf7a6a	Add filter						0	K	< 1 to 2	of 2	> >	
	Name	+ Grou	p ID	•	Group Name +	VPC ID	¥	Description		*				
		sg-1b	bf7a6a		DockerSwarm-NodeVpcSG-I	vpc-4ccc6535		Node SecurityGroup						
(sg-c1	bc 79b0		DockerSwarm-ManagerVpcS	vpc-4ccc6535		Manager SecurityGroup						
S	ecurity Group Description	inbound	Outbound	Tags										
	Type (i)			Protocol	0	Port Range (i)		Source (l
8	All traffic			All		All		0.0.0.0/0						
	All traffic			All		All		::/0						
						1000 I								

Figure 15-6. The security group inbound rules are enabled for all traffic

CHAPTER 15 USING SERVICE STACKS

SSH login into the Swarm manager EC2 instance and obtain the public IP address from the AWS management console, as shown in Figure 15-7.

									0	-
Q Instance S	State : Running 💿	Add filter					0	< <	1 to 3 of 3	> >
Name	* I	Instance ID •	Instance Type +	Availability Zone +	Instance State +	Status Checks -	Alarm Status	• 1	Public DNS	(IPv4)
DockerSw	varm-worker i	-01242202c7b019e28	t2.micro	us-east-1b	running	2/2 checks	None	>	ec2-34-227-1	51-152
DockerSw	arm-Manager i	-0bfc9384db35835cd	t2.micro	us-east-1b	running	2/2 checks	None	>	ec2-54-205-4	8-154.c
DockerSw	arm-worker i	-0d89f7a7e2121a09e	t2.micro	us-east-1c	running	2/2 checks	None	>	ec2-34-227-1	91-199.
instance: i-OL	bfc9384db35835	cd (DockerSwarm-N	lanager) Public	DNS: ec2-54-205-48	-154.compute-1.ar	nazonaws.com				
Instance: i-OL	bfc9384db35835	cd (DockerSwarm-N	lanager) Public	DNS: ec2-54-205-48	-154.compute-1.ar	nazonaws.com				88
Description	Status Checks	cd (DockerSwarm-N s Monitoring	Manager) Public Tags	DNS: ec2-54-205-48	-154.compute-1.ar	nazonaws.com				88
Description	Status Checks	cd (DockerSwarm-N Monitoring i-Obfc9384db35835	Manager) Public Tags cd	DNS: ec2-54-205-48	Public DNS (IF	v4) ec2-54-205-48 1.amazonaws.	-154.compute-		-	88
nstance: 1-0	bfc9384db35835 Status Checks Instance ID Instance state	cd (DockerSwarm-N Monitoring 0 i-0bfc9384db35835 a running	Manager) Public Tags cd	DNS: ec2-54-205-48	Public DNS (IF	v4) ec2-54-205-48 1.amazonaws. IP 54.205.48.154	-154.compute- com			
nstance: 1-01	bfc9384db35835 Status Checks Instance ID Instance state Instance type	cd (DockerSwarm-N Monitoring i-Obfc9384db35835 running t2.micro	Manager) Public Tags cd	DNS: ec2-54-205-48	Public DNS (IF	v4) ec2-54-205-48 1.amazonaws. IP 54.205.48.154 IPs -	-154.compute- com		-	88
Description	Status Checks Instance ID Instance state Instance type Elastic IPs	d (DockerSwarm-N Monitoring i-Obfc 9384db36836 running t2.micro	Manager) Public Tags cd	DNS: ec2-54-205-48	Public DNS (IF Public DNS (IF IPv4 Public IPv6 IPv6 Private D	v4) ec2-54-205-48 1.amazonaws. IP 54-205.48.154 IPs - NS ip-172-31-19-1	-154.compute- com 38.ec2.internal		-	88
Description	bfc9384db35835 Status Checks Instance ID Instance state Instance type Elastic IPs Availability zone	d (DockerSwarm-N Monitoring) i-Obfc 9384db35835 running t2.micro us-east-1b	Manager) Public Tags cd	DNS: ec2-54-205-48	Public DNS (IF IPv4 Public IPv4 Public IPv6 Private D Private	Azonaws.com v4) ec2-54-205-48 1.amazonaws. IP 54-205.48.154 IPs - NS ip-172-31-19-11 Ps 172.31.19.138	-154.compute- com 38.ec2.internal			88
Instance: 1-01	btc9384db35835 Status Checks Instance ID Instance state Instance type Elastic IPs Availability zone Security groups	d (DockerSwarm-M Monitoring i-Obfc 9384db36835 running t2.micro us-east-1b DockerSwarm-Mar	Manager) Public Tags cd nagerVpcSG-	DNS: ec2-54-205-48	Public DNS (IF IPv4 Public IPv6 IPv6 Private D Private Secondary private	v4) ec2-54-205-48 1.amazonaws. IP 54-205-48 154 IPs - NS ip-172-31-19-1 Ps 172-31.19.138 Ps	-154.compute- com 38.ec2.internal			

Figure 15-7. Public IP address

Using the key pair used to create the CloudFormation stack SSH login into the Swarm manager instance.

```
ssh -i "docker.pem" docker@54.205.48.154
```

The command prompt for the Swarm manager node is displayed.

```
[root@localhost ~]# ssh -i "docker.pem" docker@54.205.48.154
Welcome to Docker!
```

List the nodes in the Swarm mode.

docker node ls

Three nodes, one manager and two workers, are listed.

~ \$ docker node ls ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS
bf4ifhh86sivqp03ofzhk6c46	ip-172-31-21-175.ec2.internal	Ready	Active	Leader
ozdhl0jtnricny1y95xbnhwtq	ip-172-31-37-108.ec2.internal	Ready	Active	
ud2js50r4livrqf3f4l30fv9r *	ip-172-31-19-138.ec2.internal	Ready	Active	

Test the Swarm mode by creating and listing a Hello World service.

docker service create --replicas 2 --name helloworld alpine ping docker.com

docker service ls

The docker service commands output indicates a Docker Swarm service, so it's created and listed.

~ \$ docker service create --replicas 2 --name helloworld alpine ping docker.com q05fef2a7cf98cv4r2ziyccnv

~ \$ docker service ls
ID NAME MODE REPLICAS IMAGE PORTS
q05fef2a7cf9 helloworld replicated 2/2 alpine:latest
~ \$

Configuring a Service Stack

To create a service stack consisting of two services, one for a WordPress blog and another for MySQL database, create a stack file using the Docker Compose version 3 YAML format (https://docs.docker.com/compose/ compose-file/). Create a docker-cloud.yml stack file (the filename is arbitrary) to specify two services (web and mysql) using Docker images wordpress and mysql respectively. Set the environment variables for the Docker image. The only environment variable required to be set is MYSQL_ROOT_PASSWORD for the mysql Docker image. The WORDPRESS_DB_PASSWORD environment variable for the wordpress Docker image defaults to the MYSQL_ROOT_PASSWORD, but may also be set explicitly to the same value as the MYSQL_ROOT_PASSWORD. Some of the other environment variables used by the wordpress Docker image are listed in Table 15-2.

Environment Variable	Description	Default Value
WORDPRESS_DB_HOST	The linked database host, which is assumed to be MySQL database by default.	The IP and port of the linked mysql Docker container
WORDPRESS_DB_USER	The database user.	root
WORDPRESS_DB_PASSWORD	The database password.	MYSQL_ROOT_PASSWORD
WORDPRESS_DB_NAME	The database name. The database is created if it does not already exist.	wordpress
WORDPRESS_TABLE_PREFIX	Table prefix.	«»

Table 15-2. Environment Variables for the Docker Image WordPress

If we were to create a WordPress blog using the wordpress and mysql images with the docker run command, we would create Docker containers for each of the Docker images separately and link the containers using the -link option. If we were to use Docker Compose (standalone), we would need to add a links or depends_on sub-option in the Docker Compose file.

Next, specify the Docker images and environment variables to the stack file for creating a service stack. To use the Docker Compose YAML file format for Swarm mode stacks, specify the version in the stack file as 3 or a later version such as 3.1. The docker-cloud.yml file is listed:

```
version: '3'
services:
web:
image: wordpress
links:
    - mysql
environment:
    - WORDPRESS_DB_PASSWORD="mysql"
ports:
    - "8080:80"
mysql:
image: mysql:latest
environment:
    - MYSQL_ROOT_PASSWORD="mysql"
    - MYSQL_DATABASE="mysqldb"
```

The ports mapping of 8080:80 maps the WordPress Docker container port 80 to the host port 8080. Any stack file options, such as links that are included in the preceding listing that are not supported by docker stack deploy, are ignored when creating a stack. Store the preceding listing as docker-cloud.yml in the Swarm manager EC2 instance. Listing the files in Swarm manager should list the docker-cloud.yml file.

```
~ $ ls -1
total 4
-rwxr-x--- 1 docker docker 265 Jun 17 00:07 docker-cloud.yml
```

Having configured a stack file with two services, next we will create a service stack.

Creating a Stack

The docker stack deploy command is used to create and deploy a stack. It has the following syntax.

docker stack deploy [OPTIONS] STACK

The supported options are discussed in Table 15-3.

Option	Description	Default Value
bundle-file	Path to a Distributed Application Bundle file. An application bundle is created from a Docker Compose file just as a Docker image is created from a Dockerfile. An application bundle may be used to create stacks. Application bundles are an experimental feature at the time the chapter was developed and are not discussed in this chapter.	
compose-file, -c	Path to stack file.	
with-registry-auth	Whether to send registry authentication information to Swarm agents.	False

Table 15-3. Options for the docker stack deploy Command

Using the stack file docker-cloud.yml, create a Docker stack called mysql with the docker stack deploy command.

```
docker stack deploy --compose-file docker-cloud.yml mysql
```

A Docker stack is created and the links option, which is not supported in Swarm mode, is ignored. Two Swarm services—mysql_mysql and mysql_web—are created in addition to a network mysql_default.

```
~ $ docker stack deploy --compose-file docker-cloud.yml mysql
Ignoring unsupported options: links
```

```
Creating network mysql_default
Creating service mysql_mysql
Creating service mysql_web
```

Listing Stacks

List the stacks with the following command.

docker stack ls

The mysql stack is listed. The number of services in the stack also are listed.

```
~ $ docker stack ls
NAME SERVICES
mysql 2
```

Listing Services

List the services in the mysql stack using the docker stack services command, which has the following syntax.

docker stack services [OPTIONS] STACK

The supported options are listed in Table 15-4.

Table 15-4. Options for the docker stack services Command

Option	Description	Default Value
filter, -f	Filters output based on filters (or conditions) provided	
quiet, -q	Whether to display only the IDs of the services	false

To list all services, run the following command.

docker stack services mysql

The two services—mysql_mysql and mysql_web—are listed.

~ \$ docker st	ack services	mysql		
ID	NAME	MODE	REPLICAS	IMAGE
ixv0ykhuo14c vl7ph81hfxan	mysql_mysql mysql_web	replicated replicated	1/1 1/1	mysql:latest wordpress:latest

To filter the services, add the --filter option. To filter multiple services, add multiple --filter options, as shown in the following command.

docker stack services --filter name=mysql web --filter name=mysql mysql mysql

The filtered stack services are listed. As both services are specified using -filter, both services are listed.

```
~ $ docker stack services --filter name=mysql_web --filter name=mysql_mysql mysql
l
ID NAME MODE REPLICAS IMAGE
ixv0ykhuo14c mysql_mysql replicated 1/1 mysql:latest
vl7ph81hfxan mysql_web replicated 1/1 wordpress:latest
```

The services created by a stack are Swarm services and may also be listed using the following command.

docker service ls

The same two services are listed.

```
~ $ docker service ls
ID NAME MODE REPLICAS IMAGE
ixv0ykhuo14c mysql_mysql replicated 1/1 mysql:latest
sl2jmsat30ex helloworld replicated 2/2 alpine:latest
vl7ph81hfxan mysql_web replicated 1/1 wordpress:latest
```

Listing Docker Containers

The docker stack ps command is used to list the Docker containers in a stack and has the following syntax; output the command usage with the --help option.

```
~ $ docker stack ps --help
Usage: docker stack ps [OPTIONS] STACK
List the tasks in the stack
Options:
   -f, --filter filter Filter output based on conditions provided
        --help Print usage
        --no-resolve Do not map IDs to Names
        --no-trunc Do not truncate output
```

To list all Docker containers in the mysql stack, run the following command.

docker stack ps mysql

By default, one replica is created for each service, so one Docker container for each service in the stack is listed. Both Docker containers are running on a Swarm worker node.

~ \$ docker sta	ck ps mysql		
ID	NAME	IMAGE	NODE
DESIRED STATE	CURRENT STATE	ERROR	PORTS
n9oqwaikd61g	mysql_web.1	wordpress:latest	ip-172-31-37-108.ec2.internal
Running	Running 3 minu	ites ago	
infzi7kxg9g9	mysql_mysql.1	mysql:latest	ip-172-31-37-108.ec2.internal
Running	Running 3 minu	ites ago	

Using the -f option to filter the Docker containers to list only the mysql_web.1 container.

~ \$ docker sta	ick ps -f name=	mysql_web.1	1 mysql	
ID	NAME	IMAGE		NODE
DESIRED STATE	CURRENT STAT	E	ERROR	PORTS
n9oqwaikd61g	mysql_web.1	wordpress	latest	ip-172-31-37-108.ec2.internal
Running	Running 9 mi	nutes ago		

List all the running containers by setting the desired-state filter to running.

```
~ $ docker stack ps -f desired-state=running mysql
ID
              NAME
                              IMAGE
                                                 NODE
DESIRED STATE
                                        ERROR
               CURRENT STATE
                                                PORTS
n9oqwaikd61g mysql_web.1
                              wordpress:latest ip-172-31-37-108.ec2.internal
Running
              Running 10 minutes ago
infzi7kxg9g9
              mysql mysql.1
                              mysql:latest
                                                 ip-172-31-37-108.ec2.internal
               Running 10 minutes ago
Running
```

Using the Service Stack

Next, we use the stack to create a WordPress blog. The stack service called web may be accessed on port 8080 on the Swarm manager host. Obtain the public DNS of the Swarm manager node EC2 instance, as shown in Figure 15-8.

Laun	ich Instanc	e Connect	t Actions Y							0	۰	6
Q,	Instance St	ate : Running	Add filter					ØK	<	1 to 3 of 3	>	>
	Name	~	Instance ID	Instance Type	Availability Zone +	Instance State ~	Status Checks ~	Alarm Status		Public DNS	(IPv4	4)
	DockerSwar	m-worker	i-01242202c7b019e2	8 t2.micro	us-east-1b	running	2/2 checks	None		ec2-34-227-	151-18	52.co.
	DockerSwar	m-Manager	i-0bfc9384db35835cd	t t2.micro	us-east-1b	running	2/2 checks	None		ec 2-54-205-	48-154	4.com
	DockerSwar	m-worker	i-0d89f7a7e2121a09e	t2.micro	us-east-1c	running	2/2 checks	None		ec2-34-227-	191-19	99.co.
4												,
Insta	ince: i-0bf	c9384db3583	5cd (DockerSwarm	-Manager) Public	C DNS: ec2-54-205-48	-154.compute-1.am	azonaws.com				= (
Des	cription	Status Chec	ks Monitoring	Tags								
		Instance	D i-Obfc 9384db358	35cd		Public DNS (IP	v4) ec2-54-205-48 1 amazonaws	-154.compute-				
		Instance sta	te running			IPv4 Public	IP 54.205.48.154					

Figure 15-8. Public DNS of Swarm manager

Open the <public dns>:8080 URL in a browser. The <public dns>:8080/wp-admin/install.php URL is displayed to start the WordPress installation. Select Continue. Specify a subtitle, username, password, e-mail, and whether to discourage search engines from indexing the website. Then click on Install WordPress, as shown in Figure 15-9.

Welcome					
Welcome to the famous five-minute WordPress installation process! Just fill in the information below and you'll be on your way to using the most extendable and powerful personal publishing platform in the world.					
Information needed					
Please provide the following information. Don't worry, you can always change these settings later.					
Site Title	Docker Swarm Stacks				
Username	dvohra				
	Usernames can have only alphanumeric symbol.	characters, spaces, underscores, hyphens, periods, and the @			
Password		(Show			
	Strong				
	Important: You will need this passw	ord to log in. Please store it in a secure location.			
Your Email	dvohra17@yahoo.com				
	Double-check your email address befor	e continuing.			
Search Engine	Discourage search engines	from indexing this site			
Visibility	It is up to search engines to honor this	equest			

Figure 15-9. Installing WordPress

WordPress is installed, as shown in Figure 15-10. Click on Log In.

Success!	
WordPress has been installed. Thank you, and enjoy!	
Username dvohra	
Password Your chosen password.	

Figure 15-10. WordPress is installed

Specify a username and password and click on Log In, as shown in Figure 15-11.

→ C ① Not secure ec2-54-205-48-154.comp	ute-1.amazonaws.com:8080/wp-login.php
	Username or Email Address
	Username or Email Address
	Username or Email Address dvohra Password
	Username or Email Address dvohra Password
	Username or Email Address dvohra Password Remember Me

Figure 15-11. Logging in

Dashboard Docker	Swe ×				ö (=) =
← → C ① ec2-5	4-205-48-154.compute-1.amazonaws.com:8080/wp-admin/				F 🛧 🖸
🔞 者 Docker Swarm	Stacks 🌹 0 🕂 New				Howdy, dvohra
Dashboard	Dashboard				Screen Options • Help •
Home Updates	Welcome to WordPress				() Dismiss
📌 Posts	We've assembled some links to get you started:				
91 Media	Get Started	Next Steps		More Actions	
📕 Pages		Write your first blo	og post	Manage widgets or	r menus
Comments	Customize Your Site	+ Add an About pag	e	Turn comments on	oroff
Appearance	or, change your theme completely	View your site		R Learn more about	getting started
Plugins					
👗 Users	At a Glance	*	Quick Draft		*
🖋 Tools	📌 1 Post 📕 1 Page		Title		
Settings	📮 1 Comment		1.00		
Collapse menu	WordPress 4.8 running Twenty Seventeen theme. Search Engines Discouraged		what's on your mind?		
	Activity	*	Save Draft		
	Recently Published				
	Today, 12:34 am Hello world!		WordPress Events and N	ews	

The WordPress blog dashboard is displayed, as shown in Figure 15-12.

Figure 15-12. The WordPress dashboard

To add a new post, select Posts and click on Add New, as shown in Figure 15-13.

Posts - Docker Swar	m St. × \					8 - 8
← → C ① ec2-5	54-205-48-154.compute-1.amazonaws.com:8080/wp-admin/edit.php					☆ 🖸 :
🚯 👩 Docker Swarm	Stacks 👎 0 🕂 New					Howdy, dvohra 📗
2 Dashboard	Posts Add New				Screen Op	tions • Help •
🖈 Posts	All (1) Published (1)					Search Posts
All Posts	Bulk Actions Apply All dates All Categories Filter					1 item
Add New	Title	Author	Categories	Tags		Date
Categories Tags	Hello world!	dvohra	Uncategorized	-	Ģ	Published 2 mins ago
91 Media	Title	Author	Categories	Tags		Date
 Pages Pomments Appearance Plugins Users Tools Settings Collapse menu 	Bulk Actions • Apply					1 item
ec2-54-205-48-154.compute	e-Lamazonawas.com/8080/wp-admin/post-new.php					Version 4.8

Figure 15-13. Adding a new post

In the Add New Post dialog, specify a title and add a blog entry. Click on Publish, as shown in Figure 15-14.

← → C ① ec2-	54-205-48-154.compute-1.amazonaws.com:8080/wp-admin/post-new.php	☆ 🔞
🔞 📸 Docker Swam	Stacks 🛡 0 🕂 New	Howdy, dvohra 🔲
2 Dashboard	Add New Post	Screen Options ¥ Help ¥
📌 Posts	Wordpress Blog with MySQL. On Docker Swarm Mode Stack	Publish .
All Posts	Permalink: http://ec2-54-205-48-154.compute-1.amazonaws.com/8080/2017/06/17/wordpress-blog-wcker-swarm-stack/	Save Draft Preview
Categories	9 Status: Draft Edit	
Tags	Paragraph ▼ B I ⊟ ⊟ 44 ≣ Ξ Ξ Ø 22 Ξ 🛛	Visibility: Public Edit
Pages The wor Comments "Stacks"	The <u>wordpress</u> blog is created using the Docker Swam Mode feature called "Stacks", which is available in Docker <u>v1.13</u> and later.	Move to Trash Publish
Appearance		Format
 Plugins Users Tools Settings Collapse menu 		Aside Standard Sta
	Word count: 21 Draft saved at 12:37:08 am.	

Figure 15-14. Publishing a new post

The new post is added. Click on View Post, as shown in Figure 15-15, to display the post.

/ C Edit Post + Docker S	warr x			800
← → × 0 ec2-!	54-205-48-154.compute-1.amazonaws.com:8080/wp-admin/post.php?post=4&action=edit			☆ 🖸 :
🚯 🔗 Docker Swarm	i Stacks 📕 0 🕂 New View Post			Howdy, dvohra 📃
Dashboard	Edit Post Add New		Screen Option	s 🔻 Help 🔻
📌 Posts	Post published. <u>View post</u>			0
All Posts Add New Categories Tags	Wordpress Blog with MySQL On Docker Swarm Mode Stack Permalink: http://cc2-54-205-48-154.compute-Lamazonaws.com/8080/2017/05/17/wordpress-blog-wswarm-mode	le-stack/ Edit	Publish	▲ Preview Changes
 Media Pages Comments 	93 Add Media Paragraph ▼ B I II	Visual Text	Status: Published Edit Visibility: Public Edit Dished on: Jun 17,	2017 @ 00:38 Edit
Appearance	"Stacks", which is available in Docker v1.13 and later.		Move to Trash Format	Update
Collapse manual			 ● ★ Standard ○ ● Aside ○ ● Image ○ ● Image 	
Waiting for ec2-54-205-48-	154.compute-Lamzonaws.com	on June 17, 2017 at 12:38 am	Gallery	

Figure 15-15. Viewing the new post

The blog post is displayed, as shown in Figure 15-16.

Wordpress Blog with M/ ×		
← → C ① ec2-54-205-48-154.compute-1.amazonaws.com/8080/2017/06/17/wordpress-blog-with-mysql-on-docke	r-swarm-mode-stack/	\$
🔞 🕸 Docker Swarm Stacks 🖋 Customize 🌹 0 🕂 New 🖉 Edit Post		Howdy, dvohra 📃 🔍
DOCKER SWARM STACKS		
WordPress Blog with MySQL On Docker	Search Q	
SWarm Mode Stack The wordpress blog is created using the Docker Swam Mode feature called "Stacks", which is available in Docker v1.13 and later.	RECENT POSTS WordPress Blog with MySQL On Docker Swarm	
	Mode Stack Hello world!	
Edit	RECENT COMMENTS	
	A WordPress Commenter on Hello world!	

Figure 15-16. Displaying a blog post

Scroll down and add a comment, as shown in Figure 15-17.

🚯 🍘 Docker Swarm Stacks 🖌 Customize 👎 0 🕂 New 🖉 Edit Post		Howdy, dvohra 🔲 🔍
Leave a Reply	ARCHIVES	
Logged in as dvohra. Log out?	June 2017	
Comment	CATEGORIES	
The Docker Swarm mode "Stacks" feature is available only in Swarm mode and in Docker v1.13 and later.	Uncategorized	
	META	
	Site Admin	
	Log out	
	Entries RSS	
Post Comment	Comments RSS	
U	WordPress.org	
PREVIOUS		
← Hello world!		

Figure 15-17. Adding a comment

The comment is added, as shown in Figure 15-18.

D Wordpress Blog with My x			
← → C ① ec2-54-205-	48-154.compute-1.amazonaws.com:8080/2017/06/17/wordpress-blog-with-mysql-on-docker	-swarm-mode-stack/#comment-2	☆ 🖸 🗄
🚯 🍙 Docker Swarm Stacks	🖋 Customize 🌹 0 🕂 New 🖉 Edit Post		Howdy, dvohra 🔲 🔍
	JUNE 17, 2017 BY DVOHRA WordPress Blog with MySQL On Docker	Search Q	
	Swarm Mode Stack The wordpress blog is created using the Docker Swam Mode feature called "Stacks", which is available in Docker v1.13 and later.	RECENT POSTS	
		WordPress Blog with MySQL On Docker Swarm Mode Stack	
		Hello world!	
	Edit	RECENT COMMENTS	
		dvohra on WordPress Blog with MySQL On Docker Swarm Mode Stack	
		A WordPress Commenter on Helio world!	
	One Reply to "WordPress Blog with MySQL On Docker Swarm Mode Stack"	ARCHIVES	
	dvohra	June 2017	
<u> /</u>	JUNE 17, 2017 AT 12:41 AM EDIT The Docker Swarm mode "Stacks" feature is available only in Swarm mode and in	CATEGORIES	

Figure 15-18. The comment has been added

Removing a Stack

The docker stack rm STACK command is used to remove a stack. Remove the mysql stack using the following command.

docker stack rm mysql

The mysql stack is removed and the docker stack service mysql command does not list the stack, as shown in the output from the command.

~\$ docker stack rm mysql

Removing service mysql_mysql

Removing service mysql_web

Removing network mysql_default

~\$ docker stack services mysql

Nothing found in stack: mysql
Summary

This chapter introduced stacks, a Docker-native feature added in Docker 1.13. A stack is a collection of related services and is created using a stack file, which is defined in YAML format similar to the Docker Compose v3.x YAML syntax. This chapter concludes this book about Docker management design patterns. As new features are added to Docker, other design patterns may be used for developing Docker-native applications.

Index

A

Amazon Route 53 service, 242, 251 Amazon Web Services (AWS), 281-284 CloudFormation Deploy Docker, 34 EC2, 46 Elastic Load Balancer, 48 launch configurations, 48 Moby Linux AMI, 47 stacks, 41-46 Swarm parameters, 35 Swarm properties, 36-38 delete stack, 51-53 editions, 33 key pair, 33 manager and worker nodes, 49-50 option, 284 single/multi zone Swarm, 31-33 Application load balancers, 228 Availability zone column, 245 AWS credentials dialog, 282 AWS elastic load balancer, 241 AWS service provider, 285

B

Bind mounts, 98–99, 112–114 Bridge network AWS CloudFormation stack, 182 create service, 186 description, 184 docker0 limitations, 179 Swarm manager instance, 183

C

Classic load balancers, 228 CloudFormation stack, 235, 244, 287 Cloud settings page, 282 Cross-account access, 273 Current state reconciliation, 138 Custom overlay network gossip protocol, 194 IP range, 191 IPSEC tunnels, 194 mysql-network, 192–193 MySQL database service, creation, 194–195 service containers, 181

D

Desired state reconciliation, 138-139 Docker Cloud, 271-272 CoreOS, 1 DNS/IPv4, 3 execution, 3-6 launch instances, 2 service, 239, 246 Docker Cloud dashboard, 289 Docker Cloud Swarm AWS role, 287 Docker Cloud Swarm mode, 285, 294, 296 docker_gwbridge network, 181, 184 Docker docker stack deploy command, 297 Docker Swarm load balancer. 247 Docker Swarm mode service, 219 Domain Name Service (DNS), 251, 256 DynamoDB database, 271

E

Edit Endpoint link, 288 Elastic load balancer (ELB), 220, 238, 250 External load balancer, 221

F

Failover record type, 259 Failover routing policy, 262 INDEX

G

Global service, 83 rolling update, 176, 178 spread scheduling policy, 153 Gossip protocol, 194

H

Hello World service, 222, 233, 237, 246, 293 Highly available website alias target, 258, 261 Amazon Route 53, 242 AWS elastic load balancer, 258 AWS region, 241 confirmation dialog, 267 DNSes, 243 Docker Swarm, 243 domain name, 243 failover record type, 259 hosted zone, 252, 266–268 hostname, 266 name servers, 254–255 Host network, 184

🛛 I, J, K

IAM role, 288 Ingress network, 180, 185 add load balancer listener, 190–191 create service, 189 description, 184 docker service create command, 188 inspect command, 189, 191 ports, 181 Ingress load balancing, 219 Internal overlay network, 195–198

L

Listeners tab, 236 Load balancer listeners, 236 Load balancing CoreOS, 221 custom scheduling, 219 dialog, 228 DNS name, 232 Docker container, 223 EC2 instances, 230 ELB, 220 external elastic load balancer, 227 HTTP, 228 invoke, service at worker node, 226 new security group, 229 protocol, 228 security settings, 229 service discovery, 219 service task, 226, 234 SSH login, 222 types, 228 Logging and monitoring connecting, SPM and Logsene apps, 208–209 Docker Swarm logs, Logsene, 214–216 Docker Swarm metrics, 213 Logsene application, 205–206, 208 MySQL database service, 212 Sematext Docker agent, 209, 211 Sematext SPM and Logsene, 203 SPM application creation, 203–205

M

Mounts bind, 98–99, 112–114 data stored, 97 EC2 instances, AWS Swarm nodes, 100 named volume, 100–101 options, 102–103 tmpfs mount options, 103 volume (*see* Volume mount) Multiple Docker Swarms, 243 MySQL database Docker container, 70–72 service, creation, 67

■ N, O

Network custom overlay, 181 delete, 198–199 docker0 bridge limitations, 179 docker_gwbridge, 181 ingress, 180 internal overlay, 195–198 Swarm mode, 183–184

■ P, Q

Postgres image, 167, 170 Primary record set, 260 Public hosted zone, 254

R

Replicated services, 60 Resources configuration allocation, resource reserves set, 117–118 CPU and memory limits, 124 EC2 instances, Swarm nodes, 119

options, resource reserves, 120 reserved resources, 117, 122-123 resource-utilized node, 116 scale up, CloudFormation stack, 127-128, 130 scaling, 121 service creation, 119-121 service resources configuration, 124 setting resource limits, 120 types, 115 unequal allocation, 116 usage and node capacity, 125-126 Rolling update configuration, 158 ContainerSpec, 168-169 desired state. 159 environment variables, 162 failure action, 173-174 global service, 176, 178 image tag, 161-162 mounts, 172-173 mysql service to Postgres, 175 nodes, 157 options, 157 Postgres image, 167, 170 resource limits and reserves, 164-165, 167 restart, 171 running, 163 sequences, 156-157 shutting down, 155-156 status, 162 updates, 158, 160-161

S, T, U

Scaling services docker node ls command, 87 Docker Swarm mode, 86 global service, 92 multiple services, 93-95 removing service, 92 replacement service task, 95-96 replicated service, creation, 87 scaling down, 91 scaling up, 88-91 Scheduling add multiple constraints, 148, 150 add node labels, 150 definition. 131 lack of locality, 132 limited, node resource capacity, 141-143, 145 node attributes, constraints, 146 pending state, 142-143 remove constraints, 152 shutdown desired state, 143 single point of failure, 132

specific nodes, 146-147 unbalanced utilization of resource, 132 underutilization of resources, 132 updating placement constraints, 151 Secondary record set, 261-263 Sematext Docker agent, 201, 202 Services command line, 62-63 creation, 60-61 docker service inspect, 63-65 docker service ps, 68-70 EC2 instances, 57-59, 66 global, 60 MySQL database, 67 removing, 83 replicated, 60 scaling, 68 sub-commands, 59 tasks and containers, 55-57, 61 updation container labels, 82 environment variable, 80 image, 75-78, 81 mysql, 74-75 options, 73-74 placement constraints, 79-80 resources settings, 82 Service stacks CloudFormation stack, 300 configuration, 303-304 creation, 304-305 docker stack commands, 298-299 docker stack ps command, 307 listing stacks, 305 Moby Linux AMI, 301 options, docker stack services command, 306 public IP address, 302 removing, 314 security group inbound rules, 301 Swarm mode services, 298 WordPress add new post, 311 comment adding, 313 dashboard, 311 displaying blog post, 313 installation, 309 logging in, 310 public DNS, Swarm manager, 308 publishing post, 312 viewing post, 312 Single responsibility principle (SRP), 97 Spread scheduling policy, global service, 153 Spread scheduling strategy CloudFormation stack, 135 current state reconciliation, 138

INDEX

Spread scheduling strategy (cont.) desired state reconciliation, 138-141 docker service scale command, 137 EC2 instances, 135 mysql service, 134 nginx server, 134 node ranking, 133 re-adding worker node, 140-141 using MySQL database, 136-137 Swarm manager node, 295 Swarm mode, 183-184 account and external IDs, 274 apply policy, 278 AWS infrastructure, 287 Cloud service, 281 CoreOS instance, 290 cross-account access, 274 deploying, 286 desired state, 11 Docker Cloud service, 280 dockercloud-swarm-role, 276 Docker images, 271 EC2 console, 292 EC2 instances, 14, 290 endpoint, 289 features, 9 IAM role, 272 initializing, 14-15, 17-18 inline policy, 276, 277, 279 joining nodes, 18-19 manager nodes, 10, 11, 24-27

node availability, 28–29 quorum, 12–13 Raft consensus, 11 reinitializing, 28 removing, 30 role name, 273 service, 11 testing, 20–24 worker nodes, 10, 12, 25 Swarm nodes, 220, 249 Swarms toolbar option, 280

🛛 V, W, X, Y, Z

Volume mount auto-generated named volumes, 107, 109, 111 container, 98 destination directory, 108 docker service create command, 102 mysql-scripts, 107, 110-111 named volume, 97, 103-105, 107, 111 nginx-root, 105 options, 102-103 removing, 112 service creation, 105, 108 service definition, 104, 107 service replicas, 104 service task, 110 task container, service, 109 tmpfs mount, 103 volume-label, 103