

THE EXPERT'S VOICE® IN SQL

SQL Server AlwaysOn Revealed

Supporting 24 x 7 applications
with continuous uptime

Second Edition

Peter A. Carter

Apress®

www.allitebooks.com

SQL Server AlwaysOn Revealed

Second Edition



Peter A. Carter

Apress®

SQL Server AlwaysOn Revealed, 2nd Edition

Peter A. Carter
Botley, United Kingdom

ISBN-13 (pbk): 978-1-4842-2396-3
DOI 10.1007/978-1-4842-2397-0

ISBN-13 (electronic): 978-1-4842-2397-0

Library of Congress Control Number: 2016960322

Copyright © 2016 by Peter A. Carter

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.

Managing Director: Welmoed Spahr

Lead Editor: Jonathan Gennick

Development Editor: Laura Berendson

Technical Reviewer: Bradley Beard

Editorial Board: Steve Anglin, Pramila Balan, Laura Berendson, Aaron Black, Louise Corrigan,

Jonathan Gennick, Todd Green, Robert Hutchinson, Celestin Suresh John, Nikhil Karkal,

James Markham, Susan McDermott, Matthew Moodie, Natalie Pao, Gwenan Spearing

Coordinating Editor: Jill Balzano

Copy Editor: Brendan Frost

Compositor: SPi Global

Indexer: SPi Global

Artist: SPi Global

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.springer.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 www.apress.com.

Apress and friends of ED books 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 Special Bulk Sales-eBook Licensing web page at www.apress.com/bulk-sales.

Any source code or other supplementary materials referenced by the author in this text are available to readers at www.apress.com. For detailed information about how to locate your book's source code, go to www.apress.com/source-code/. Readers can also access source code at SpringerLink in the Supplementary Material section for each chapter.

Printed on acid-free paper

*This book is dedicated to the beloved memory of
Margaret Carter (1938–2015).*

Contents at a Glance

About the Author	xi
About the Technical Reviewer	xiii
Acknowledgments	xv
■ Chapter 1: High Availability and Disaster Recovery Concepts	1
■ Chapter 2: Understanding High Availability and Disaster Recovery Technologies.....	9
■ Chapter 3: Implementing a Cluster.....	29
■ Chapter 4: Implementing an AlwaysOn Failover Clustered Instance.....	59
■ Chapter 5: Implementing HA with AlwaysOn Availability Groups	83
■ Chapter 6: Implementing DR with AlwaysOn Availability Groups	121
■ Chapter 7: Administering AlwaysOn.....	149
■ Chapter 8: Monitoring AlwaysOn Availability Groups.....	167
■ Chapter 9: Troubleshooting AlwaysOn.....	191
Index.....	207

Contents

About the Author	xi
About the Technical Reviewer	xiii
Acknowledgments	xv
■ Chapter 1: High Availability and Disaster Recovery Concepts	1
Level of Availability.....	1
Service-Level Agreements and Service-Level Objectives	3
Proactive Maintenance	4
Recovery Point Objective and Recovery Time Objective	4
Cost of Downtime	5
Classification of Standby Servers.....	6
Summary	7
■ Chapter 2: Understanding High Availability and Disaster Recovery Technologies.....	9
AlwaysOn Failover Clustering.....	9
Active/Active Configuration	11
Three-Plus Node Configurations.....	12
Quorum	13
Database Mirroring.....	15
AlwaysOn Availability Groups	18
Automatic Page Repair	22

Log Shipping	23
Recovery Modes	24
Remote Monitor Server.....	25
Failover	25
Combining Technologies.....	25
Summary	27
■ Chapter 3: Implementing a Cluster.....	29
Building the Cluster.....	30
Installing the Failover Cluster Feature.....	31
Creating the Cluster.....	37
Configuring the Cluster	48
Changing the Quorum.....	48
Configuring MSDTC.....	52
Configuring a Role	55
Summary	58
■ Chapter 4: Implementing an AlwaysOn Failover Clustered Instance.....	59
Building the Instance.....	59
Installing the Instance with PowerShell	76
Adding a Node	77
Adding a Node Using PowerShell	80
Summary	82
■ Chapter 5: Implementing HA with AlwaysOn Availability Groups	83
Preparing for Availability Groups	84
Configuring SQL Server	89
Creating the Availability Group	90

Using the New Availability Group Wizard	90
Scripting the Availability Group.....	100
Using the New Availability Group Dialog Box.....	109
Performance Considerations for Synchronous Commit Mode.....	112
Summary	119
■ Chapter 6: Implementing DR with AlwaysOn	
Availability Groups	121
Configuring the Cluster	121
Adding a Node	122
Modifying the Quorum	124
Adding an IP Address.....	128
Configuring the Availability Group	130
Adding and Configuring a Replica	131
Add an IP Address.....	139
Improving Connection Times	141
Distributed Availability Groups	142
Configuring Readable Secondary Replicas	144
Summary	147
■ Chapter 7: Administering AlwaysOn.....	149
Managing a Cluster	149
Moving the Instance Between Nodes	149
Rolling Patch Upgrade	151
Removing a Node from the Cluster.....	153
Managing AlwaysOn Availability Groups.....	154
Failover	154
Synchronizing Uncontained Objects.....	161
Adding Multiple Listeners	161
Other Administrative Considerations	163
Summary	164

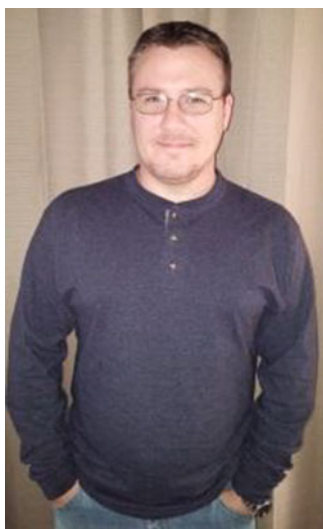
■ Chapter 8: Monitoring AlwaysOn Availability Groups.....	167
AlwaysOn Dashboard	167
AlwaysOn Health Trace	170
Monitoring AlwaysOn with Extended Events	171
Extended Events Concepts	171
Creating an Event Session to Monitor Availability Group	183
Summary	189
■ Chapter 9: Troubleshooting AlwaysOn.....	191
AlwaysOn Failover Clustered Instance Metadata	191
Discovering the Node That Hosts an Instance	191
Viewing Health Check Configuration	192
AlwaysOn Availability Group Metadata	196
Determining the Last Failover Reason.....	196
Assessing the State of Availability Databases	199
Summary	205
Index.....	207

About the Author



Peter A. Carter is a SQL Server expert with over a decade of experience in developing, administering, and architecting SQL Server platforms, data-tier applications, and ETL solutions. Peter has a passion for SQL Server and hopes that his enthusiasm for this technology helps or inspires others.

About the Technical Reviewer



Bradley Beard is a software engineer with more than 15 years experience writing dynamic, interactive websites using ColdFusion and SQL Server. He graduated from Florida Institute of Technology in 2007 with a Master of Science in Computer Information Systems, and studied for his undergraduate degrees in CIS and Technology Management at Herzing University. In 2013, he earned the MCSA: SQL Server 2012 certification from Microsoft, and in 2016, he earned the MCSE: Business Intelligence certification as well. His continual quest for learning has earned him shelves full of books at home and at work, most of which are about SQL Server, ColdFusion, or general web architectures or frameworks.

He lives in Palm Bay, Florida with his wife, Jessica, and children, Josh, Kaylee, Matthew, and Emma. He also apparently runs an animal shelter made up of his dogs, Lady and Bella, and cats, Spice, Simba, Mercury, and Dobby. In his free time, he enjoys fishing and spending time with his wife and kids.

Bradley is available for consultation and third-shift remote employment on ColdFusion and SQL Server by contacting bradley.beard@gmail.com.

Acknowledgments

I would like to thank Ian Stirk and Paul Grubb, both of whom provided helpful and constructive feedback on the first edition of this book. I have attempted to incorporate their suggestions into this edition.

I would also like to offer special thanks to Lawrence You, who helped me resolve routing issues between the subnets, within the lab environment that I used in this book.

CHAPTER 1



High Availability and Disaster Recovery Concepts

In today's 24x7 environments that are running mission-critical applications, businesses rely heavily on the availability of their data. Although servers and their software are generally reliable, there is always the risk of a hardware failure or a software bug, each of which could bring a server down. To mitigate these risks, business-critical applications often rely on redundant hardware to provide fault tolerance. If the primary system fails, then the application can automatically fail over to the redundant system. This is the underlying principle of high availability (HA).

Even with the implementation of HA technologies, there is always a small risk of an event that causes the application to become unavailable. This could be due to a major incident, such as the loss of a data center, due to a natural disaster, or due to an act of terrorism. It could also be caused by data corruption or human error, resulting in the application's data becoming lost or damaged beyond repair.

In these situations, some applications may rely on restoring the latest backup to recover as much data as possible. However, more critical applications may require a redundant server to hold a synchronized copy of the data in a secondary location. This is the underpinning concept of disaster recovery (DR). This chapter discusses the concepts behind HA and DR.

Level of Availability

The amount of time that a solution is available to end users is known as the *level of availability*, or *uptime*. To provide a true picture of uptime, a company should measure the availability of a solution from a user's desktop. In other words, even if your SQL Server has been running uninterrupted for over a month, users may still experience outages to their solution caused by other factors. These factors can include network outages or an application server failure.

Electronic supplementary material The online version of this chapter (doi:[10.1007/978-1-4842-2397-0_1](https://doi.org/10.1007/978-1-4842-2397-0_1)) contains supplementary material, which is available to authorized users.

In some instances, however, you have no choice but to measure the level of availability at the SQL Server level. This may be because you lack holistic monitoring tools within the Enterprise. Most often, however, the requirement to measure the level of availability at the instance level is political, as opposed to technical. In the IT industry, it has become a trend to outsource the management of data centers to third-party providers. In such cases, the provider responsible for managing the SQL servers may not necessarily be the provider responsible for the network or application servers. In this scenario, you need to monitor uptime at the SQL Server level to accurately judge the performance of the service provider.

The level of availability is measured as a percentage of the time that the application or server is available. Companies often strive to achieve 99%, 99.9%, 99.99%, or 99.999% availability. As a result, the level of availability is often referred to in 9s. For example, five 9s of availability means 99.999% uptime and three 9s means 99.9% uptime.

Table 1-1 details the amount of acceptable downtime per week, per month, and per year for each level of availability.

Table 1-1. *Levels of Availability*

Level of Availability	Downtime per Week	Downtime per Month	Downtime per Year
99%	1 hour, 40 minutes, 48 seconds	7 hours, 18 minutes, 17 seconds	3 days, 15 hours, 39 minutes, 28 seconds
99.9%	10 minutes, 4 seconds	43 minutes, 49 seconds	8 hours, 45 minutes, 56 seconds
99.99%	1 minute	4 minutes, 23 seconds	52 minutes, 35 seconds
99.999%	6 seconds	26 seconds	5 minutes, 15 seconds

All values are rounded down to the nearest second.

To calculate other levels of availability, you can use the script in Listing 1-1. Before running this script, replace the value of @Uptime to represent the level of uptime that you wish to calculate. You should also replace the value of @UptimeInterval to reflect uptime per week, month, or year.

Listing 1-1. Calculating the Level of Availability

```

DECLARE @Uptime    DECIMAL(5,3) ;

--Specify the uptime level to calculate

SET @Uptime = 99.9 ;

DECLARE @UptimeInterval VARCHAR(5) ;

```

```

--Specify WEEK, MONTH, or YEAR

SET @UptimeInterval = 'YEAR' ;

DECLARE @SecondsPerInterval FLOAT ;

--Calculate seconds per interval

SET @SecondsPerInterval =
(
SELECT CASE
    WHEN @UptimeInterval = 'YEAR'
        THEN 60*60*24*365.243
    WHEN @UptimeInterval = 'MONTH'
        THEN 60*60*24*30.437
    WHEN @UptimeInterval = 'WEEK'
        THEN 60*60*24*7
    END
) ;

DECLARE @UptimeSeconds DECIMAL(12,4) ;

--Calculate uptime

SET @UptimeSeconds = @SecondsPerInterval * (100-@Uptime) / 100 ;

--Format results
SELECT
    CONVERT(VARCHAR(12), FLOOR(@UptimeSeconds / 60 / 60 / 24)) + ' Day(s), '
+ CONVERT(VARCHAR(12), FLOOR(@UptimeSeconds / 60 / 60 % 24)) + ' Hour(s), '
+ CONVERT(VARCHAR(12), FLOOR(@UptimeSeconds / 60 % 60)) + ' Minute(s), '
+ CONVERT(VARCHAR(12), FLOOR(@UptimeSeconds % 60)) + ' Second(s).' ;

```

Service-Level Agreements and Service-Level Objectives

When a third-party provider is responsible for managing servers, the contract usually includes service-level agreements (SLAs). These SLAs define many parameters, including how much downtime is acceptable, the maximum length of time a server can be down in the event of failure, and how much data loss is acceptable if failure occurs. Normally, there are financial penalties for the provider if these SLAs are not met.

In the event that servers are managed in-house, DBAs still have the concept of customers. These are usually the end users of the application, with the primary contact being the business owner. An application's business owner is the stakeholder within the business who commissioned the application and who is responsible for signing off on funding enhancements, among other things.

In an in-house scenario, it is still possible to define SLAs, and in such a case, the IT Infrastructure or Platform departments may be liable for charge-back to the business teams if these SLAs are not being met. However, in internal scenarios, it is much more common for IT departments to negotiate service-level objectives (SLOs) with the business teams, as opposed to SLAs. SLOs are very similar in nature to SLAs, but their use implies that the business does not impose financial penalties on the IT department in the event that they are not met.

Proactive Maintenance

It is important to remember that downtime is caused not only by failure, but also by proactive maintenance. For example, if you need to patch the operating system, or SQL Server itself, with the latest service pack, then you must have some downtime during installation.

Depending on the upgrade you are applying, the downtime in such a scenario could be substantial—several hours for a stand-alone server. In this situation, high availability is essential for many business-critical applications—not to protect against unplanned downtime, but to avoid prolonged outages during planned maintenance.

Recovery Point Objective and Recovery Time Objective

The recovery point objective (RPO) of an application indicates how much data loss is acceptable in the event of a failure. For a data warehouse that supports a reporting application, for example, this may be an extended period, such as 24 hours, given that it may only be updated once per day by an ETL (Extract Transform and Load) process and all other activity is read-only reporting. For highly transactional systems, however, such as an OLTP (Online Transaction Processing) database supporting trading platforms or web applications, the RPO will be zero. An RPO of zero means that no data loss is acceptable.

Applications may have different RPOs for high availability and for disaster recovery. For example, for reasons of cost or application performance, an RPO of zero may be required for a failover within the site. If the same application fails over to a DR data center, however, five or ten minutes of data loss may be acceptable. This is because of technology differences used to implement intrasite availability and intersite recovery.

The recovery time objective (RTO) for an application specifies the maximum amount of time an application can be down before recovery is complete and users can reconnect. When calculating the achievable RTO for an application, you need to consider many aspects. For example, it may take less than a minute for a cluster to fail over from one node to another and for the SQL Server service to come back up; however, it may take far longer for the databases to recover. The time it takes for databases to recover depends on many factors, including the size of the databases, the quantity of databases within an instance, and how many transactions were in-flight when the failover occurred. This is because all noncommitted transactions need to be rolled back.

Just like RPO, it is common for there to be different RTOs depending on whether you have an intrasite or intersite failover. Again, this is primarily due to differences in technologies, but it also factors in the amount of time you need to bring up the entire estate in the DR data center if the primary data center is lost.

The RPO and RTO of an application may also vary in the event of data corruption. Depending on the nature of the corruption and the HA/DR technologies that have been implemented, data corruption may result in you needing to restore a database from a backup.

If you must restore a database, the worst-case scenario is that the achievable point of recovery may be the time of the last backup. This means that you must factor a hard business requirement for a specific RPO into your backup strategy. If only part of the database is corrupt, however, you may be able to salvage some data from the live database and restore only the corrupt data from the restored database.

Data corruption is also likely to have an impact on the RTO. One of the biggest influencing factors is if backups are stored locally on the server, or if you need to retrieve them from tape. Retrieving backup files from tape, or even from off-site locations, is likely to add significant time to the recovery process.

Another influencing factor is what caused the corruption. If it is caused by a faulty IO subsystem, then you may need to factor in time for the Windows administrators to run the check disk command (CHKDSK) against the volume and potentially more time for disks to be replaced. If the corruption is caused by a user accidentally truncating a table or deleting a data file, however, then this is not of concern.

Cost of Downtime

If you ask any business owners how much downtime is acceptable for their applications and how much data loss is acceptable, the answers invariably come back as zero and zero, respectively. Of course, it is never possible to guarantee zero downtime, and once you begin to explain the costs associated with the different levels of availability, it starts to get easier to negotiate a mutually acceptable level of service.

The key factor in deciding how many 9s you should try to achieve is the cost of downtime. Two categories of cost are associated with downtime: tangible costs and intangible costs. Tangible costs are usually fairly straightforward to calculate. Let's use a sales application as an example. In this case, the most obvious tangible cost is lost revenue because the sales staff cannot take orders. Intangible costs are more difficult to quantify but can be far more expensive. For example, if a customer is unable to place an order with your company, they may place their order with a rival company and never return. Other intangible costs can include loss of staff morale, which leads to higher staff turnover, or even loss of company reputation. Because intangible costs, by their very nature, can only be estimated, the industry rule of thumb is to multiply the tangible costs by three and use this figure to represent your intangible costs.

Once you have an hourly figure for the total cost of downtime for your application, you can scale this figure out, across the predicted lifecycle of your application, and compare the costs of implementing different availability levels. For example, imagine that you calculate that your total cost of downtime is \$2,000/hour and the predicted lifecycle

of your application is three years. Table 1-2 illustrates the cost of downtime for your application, comparing the costs that you have calculated for implementing each level of availability, after you have factored in hardware, licenses, power, cabling, additional storage, and additional supporting equipment, such as new racks, administrative costs, and so on. This is known as the total cost of ownership (TCO) of a solution.

Table 1-2. *Cost of Downtime*

Level of Availability	Cost of Downtime (Three Years)	Cost of Availability Solution
99%	\$525,600	\$108,000
99.9%	\$52,560	\$224,000
99.99%	\$5,256	\$462,000
99.999%	\$526	\$910,000

In this table, you can see that implementing five 9s of availability saves \$525,074 over a two-9s solution, but the cost of implementing the solution is an additional \$802,000, meaning that it is not economical to implement. Four 9s of availability saves \$520,344 over a two-9s solution and only costs an additional \$354,000 to implement. Therefore, for this particular application, a four-9s solution is the most appropriate level of service to design for.

Classification of Standby Servers

There are three classes of standby solution. You can implement each using different technologies, although you can use some technologies to implement multiple classes of standby server. Table 1-3 outlines the different classes of standby that you can implement.

Table 1-3. *Standby Classifications*

Class	Description	Example Technologies
Hot	A synchronized solution where failover can occur automatically or manually. Often used for high availability.	Clustering, AlwaysOn Availability Groups (Synchronous)
Warm	A synchronized solution where failover can only occur manually. Often used for disaster recovery.	Log Shipping, AlwaysOn Availability Groups (Asynchronous)
Cold	An unsynchronized solution where failover can only occur manually. This is only suitable for read-only data, which is never modified.	-

■ **Note** Cold standby does not show an example technology because no synchronization is required and, thus, no technology implementation is required.

Summary

Your application's level of availability is measured as a percentage of time that the application is available to users. The level of availability is often referred to in 9s. For example 99.9% uptime requirement is known as three 9s of availability. The higher the uptime requirement, the higher the cost of implementing the solution. Therefore, the level of uptime that you strive to achieve should be driven by SLAs and the cost of downtime.

Recovery point objective is a measure of how much data it is acceptable to lose in the event of a disaster. For example, if your only DR solution is backups and backups are scheduled to be taken every hour, you can achieve a recovery point objective of one hour. Recovery time objective is a measure of how long it will take to recover a solution after a failure. For example if you have a recovery time objective of 30 minutes, then you must be able to restore service with half an hour.

It is important to determine the cost of downtime for your application, as this is one of the main drivers to determine your level of availability. The cost of downtime consists of both tangible and intangible costs. Tangible costs can be calculated, whereas intangible costs need to be estimated.

Redundant infrastructure helps you to maintain availability of your applications and services. A redundant server will be classified as hot, warm, or cold. A hot standby server is one which is kept synchronized with the live server and configured to allow automatic failover. This is suitable for HA scenarios. A warm standby server is one which is kept synchronized with the live server, but is not configured to fail over automatically. Instead, an engineer must perform the failover manually. This is suitable for DR scenarios. A cold standby server is not kept synchronized with the live server and therefore cannot be failed over automatically. A cold standby server is suitable for DR scenarios where all data is read-only and never modified.



Understanding High Availability and Disaster Recovery Technologies

SQL Server provides a full suite of technologies for implementing high availability and disaster recovery. This chapter provides an overview of these technologies and discusses their most appropriate uses.

AlwaysOn Failover Clustering

A Windows cluster is a technology for providing high availability in which a group of up to 64 servers works together to provide redundancy. An AlwaysOn Failover Clustered Instance (FCI) is an instance of SQL Server that spans the servers within this group. If one of the servers within this group fails, another server takes ownership of the instance. Its most appropriate usage is for high availability scenarios where the databases are large or have high write profiles. This is because clustering relies on shared storage, meaning the data is only written to disk once. With SQL Server-level HA technologies, write operations occur on the primary database, and then again on all secondary databases, before the commit on the primary completes. This can cause performance issues.

Even though it is possible to stretch a cluster across multiple sites, in Windows Server 2012 R2 and prior, this involves SAN replication, which means that a cluster is normally configured within a single site. When configured with SAN replication, failover to a secondary site is not automatic (unless you create custom scripts to automate the process). This is because SAN replication must be stopped, and the LUNs in the SAN at the DR site need to be manually made writable. They will be presented to the DR server as read-only, while SAN replication is active.

Windows Server 2016 addresses this issue, by introducing Storage Replica (SR). SR technology aims to provide a geo-cluster solution that does not rely on SAN replication, by performing storage-agnostic, block level data synchronization. SR works in either synchronous or asynchronous modes, with a performance overhead when used in synchronous mode being traded off against a potential for data loss in asynchronous mode.

Windows Server 2016 also introduces site-aware cluster functionality, which improves the manageability of multisite clusters by enhancing operations such as heartbeat between nodes and failover behavior. The reliance on all cluster nodes being within the same domain is also removed in this version of Windows, meaning that a cluster can span multiple domains, or even exist within a workgroup.

Each server within a cluster is called a node. Therefore, if a cluster consists of three servers, it is known as a three-node cluster. Each node within a cluster has the SQL Server binaries installed, but the SQL Server service is only started on one of the nodes, which is known as the active node. Each node within the cluster also shares the same storage for the SQL Server data and log files. The storage, however, is only attached to the active node.

If the active node fails, then the SQL Server service is stopped and the storage is detached. The storage is then reattached to one of the other nodes in the cluster, and the SQL Server service is started on this node, which is now the active node. The instance is also assigned its own network name and IP Address, which are also bound to the active node. This means that applications can connect seamlessly to the instance, regardless of which node has ownership.

The diagram in Figure 2-1 illustrates a two-node cluster. It shows that although the databases are stored on a shared storage array, each node still has a dedicated system volume. This volume contains the SQL Server binaries. It also illustrates how the shared storage, IP Address, and network name are rebound to the passive node in the event of failover.

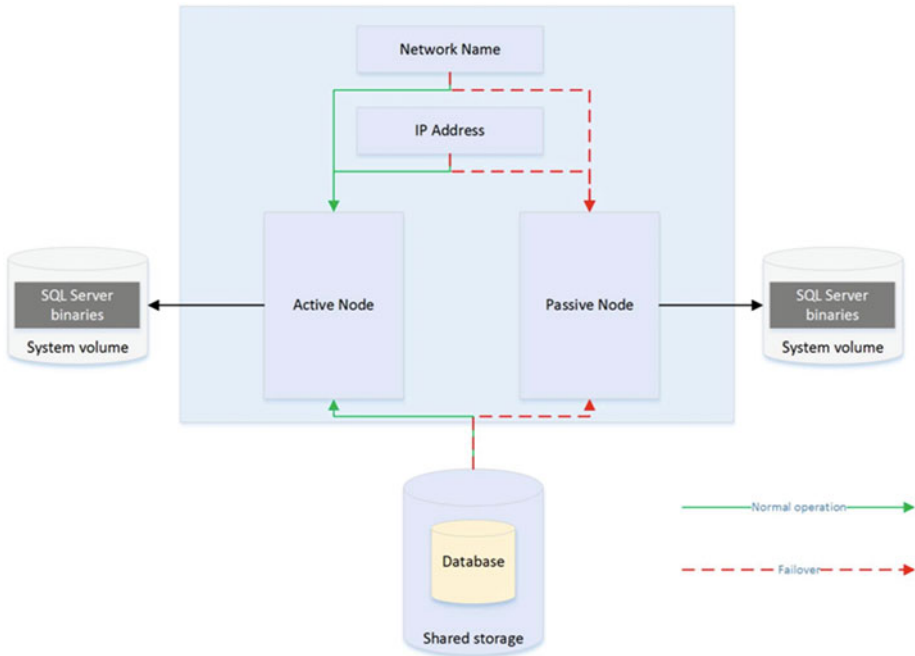


Figure 2-1. Two-node cluster

Active/Active Configuration

Although the diagram in Figure 2-1 illustrates an active/passive configuration, it is also possible to have an active/active configuration. Although it is not possible for more than one node at a time to own a single instance, and therefore it is not possible to implement load-balancing, it is possible to install multiple instances on a cluster, and a different node may own each instance. In this scenario, each node has its own unique network name and IP Address. Each instance's shared storage also consists of a unique set of volumes.

Therefore, in an active/active configuration, during normal operations, Node1 may host Instance1 and Node2 may host Instance2. If Node1 fails, both instances are then hosted by Node2, and vice versa. The diagram in Figure 2-2 illustrates a two-node active/active cluster.

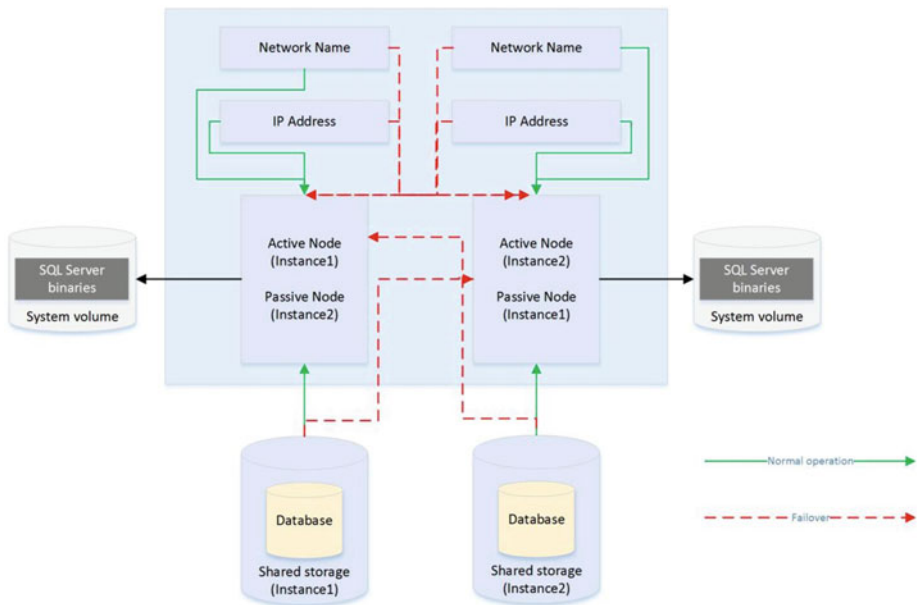


Figure 2-2. Active/active cluster

■ **Caution** In an active/active cluster, it is important to consider resources in the event of failover. For example, if each node has 128GB of RAM and the instance hosted on each node is using 96GB of RAM and locking pages in memory, then when one node fails over to the other node, this node fails as well, because it does not have enough memory to allocate to both instances. Make sure you plan both memory and processor requirements as if the two nodes are a single server. For this reason, active/active clusters are not generally recommended for SQL Server.

Three-Plus Node Configurations

As previously mentioned, it is possible to have up to 64 nodes in a cluster. When you have three or more nodes, it is unlikely that you will want to have a single active node and two redundant nodes, due to the associated costs. Instead, you can choose to implement an N+1 or N+M configuration.

In an N+1 configuration, you have multiple active nodes and a single passive node. If a failure occurs on any of the active nodes, they fail over to the passive node. The diagram in Figure 2-3 depicts a three-node N+1 cluster.

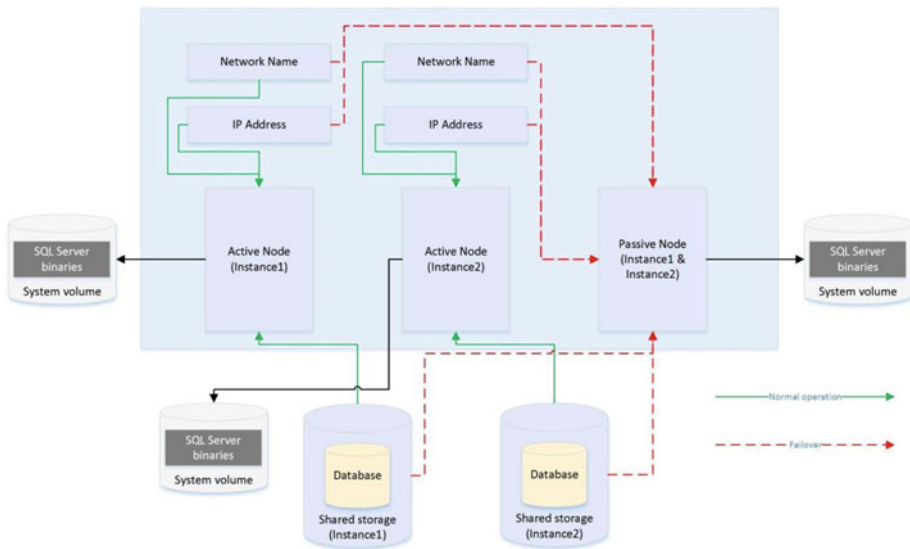


Figure 2-3. Three-node N+1 configuration

In an N+1 configuration, in a multifailure scenario, multiple nodes may fail over to the passive node. For this reason, you must be very careful when you plan resources to ensure that the passive node is able to support multiple instances. However, you can mitigate this issue by using an N+M configuration.

Whereas an N+1 configuration has multiple active nodes and a single passive node, an N+M cluster has multiple active nodes and multiple passive nodes, although there are usually fewer passive nodes than there are active nodes. The diagram in Figure 2-4 shows a five-node N+M configuration. The diagram shows that Instance3 is configured to always fail over to one of the passive nodes, whereas Instance1 and Instance2 are configured to always fail over to the other passive node. This gives you the flexibility to control resources on the passive nodes, but you can also configure the cluster to allow any of the active nodes to fail over to either of the passive nodes, if this is a more appropriate design for your environment.

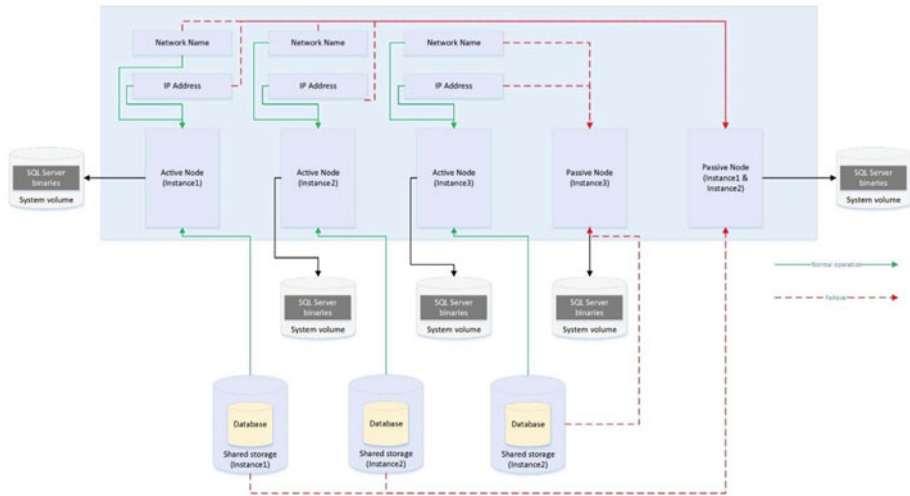


Figure 2-4. Five-node N+M configuration

Quorum

So that automatic failover can occur, the cluster service needs to know if a node goes down. In order to achieve this, you must form a quorum. The definition of a quorum is “*The minimum number of members required in order for business to be carried out.*” In terms of high availability, this means that each node within a cluster, and optionally a witness device (which may be a cluster disk or a file share that is external to the cluster), receives a vote. If more than half of the voting members are unable to communicate with a node, then the cluster service knows that it has gone down and any cluster-aware applications on the server fail over to another node. The reason that more than half of the voting members need to be unable to communicate with the node is to avoid a situation known as a *split brain*.

To explain a split-brain scenario, imagine that you have three nodes in Data Center 1 and three nodes in Data Center 2. Now imagine that you lose network connectivity between the two data centers, yet all six nodes remain online. The three nodes in Data Center 1 believe that all of the nodes in Data Center 2 are unavailable. Conversely, the nodes in Data Center 2 believe that the nodes in Data Center 1 are unavailable. This leaves both sides (known as partitions) of the cluster thinking that they should take control. This can have unpredictable and undesirable consequences for any application that successfully connects to one or the other partition. The *Quorum* = $(\text{Voting Members} / 2) + 1$ formula protects against this scenario.

■ **Tip** If your cluster loses quorum, then you can force one partition online, by starting the cluster service using the /fq switch. If you are using Windows Server 2012 R2 or higher, then the partition that you force online is considered the *authoritative partition*. This means that other partitions can automatically rejoin the cluster when connectivity is re-established.

Various quorum models are available and the most appropriate model depends on your environment. Table 2-1 lists the models that you can utilize and details the most appropriate way to use them.

Table 2-1. *Quorum Models*

Quorum Model	Appropriate Usage
Node Majority	When you have an odd number of nodes in the cluster
Node + Disk Witness Majority	When you have an even number of nodes in the cluster
Node + File Share Witness Majority	When you have nodes split across multiple sites or when you have an even number of nodes and are required to avoid shared disks*
Node + Cloud Witness Majority	When you have nodes split across multiple sites, but there is not a third data center available to host a file share quorum. This quorum model is new in Windows Server 2016

**Reasons for needing to avoid shared disks due to virtualization are discussed later in this chapter.*

Although the default option is one node, one vote, it is possible to manually remove a node's vote by changing the NodeWeight property to zero. This is useful if you have a *multi-subnet cluster* (a cluster in which the nodes are split across multiple sites). In this scenario, it is recommended that you use a file-share witness in a third site. This helps you avoid a cluster outage as a result of network failure between data centers. If you have an odd number of nodes in the quorum, however, then adding a file-share witness leaves you with an even number of votes, which is dangerous. Removing the vote from one of the nodes in the secondary data center eliminates this issue.

Caution A file-share witness and quorum witness do not store a full copy of the quorum database. This means that a two-node cluster with a file-share witness is vulnerable to a scenario known as *partition in time*. In this scenario, if one node fails while you are in the process of patching or altering the cluster service on the second node, then there is no up-to-date copy of the quorum database. This leaves you in a position in which you need to destroy and rebuild the cluster.

Windows Server 2012 R2 introduced the concepts of Dynamic Quorum and Tie Breaker for 50% Node Split. When Dynamic Quorum is enabled, the cluster service automatically decides whether or not to give the quorum witness a vote, depending on the number of nodes in the cluster. If you have an even number of nodes, then it is

assigned a vote. If you have an odd number of nodes, it is not assigned a vote. Tie Breaker for 50% Node Split expands on this concept. If you have an even number of nodes and a witness and the witness fails, then the cluster service automatically removes a vote from one random node within the cluster. This maintains an odd number of votes in the quorum and reduces the risk of a cluster going offline, due to a witness failure.

■ **Note** Clustering is discussed in more depth in Chapter 3 and Chapter 4.

Database Mirroring

Database mirroring is a technology that can provide configurations for both high availability and disaster recovery. As opposed to relying on the Windows cluster service, Database mirroring is implemented entirely within SQL Server and provides availability at the database level, as opposed to the instance level. It works by compressing transaction log records and sending them to the secondary server via a TCP endpoint. A database mirroring topology consists of precisely one primary server, precisely one secondary server, and an optional witness server.

Database mirroring is a deprecated technology, which means that it will be removed in a future version of SQL Server. In SQL Server 2014, however, it can still prove useful. For instance, if you are upgrading a data-tier application from SQL Server 2008, where AlwaysOn Availability Groups were not supported and database mirroring had been implemented, and also assuming your expectation is that the lifecycle of the application will end before the next major release of SQL Server, then you can continue to use database mirroring. Some organizations, especially where there is disconnect between the Windows administration team and the SQL Server DBA team, are also choosing not to implement AlwaysOn Availability Groups, especially for DR, until database mirroring has been removed; this is because of the relative complexity and multiteam effort involved in managing an AlwaysOn environment. Database mirroring can also be useful when you upgrade data-tier applications from older versions of SQL Server in a side-by-side migration. This is because you can synchronize the databases and fail them over with minimal downtime. If the upgrade is unsuccessful, then you can move them back to the original servers with minimal effort and downtime.

Database mirroring can be configured to run in three different modes: High Performance, High Safety, and High Safety with Automatic Failover. When running in High Performance mode, database mirroring works in an asynchronous manor. Data is committed on the primary database and is then sent to the secondary database, where it is subsequently committed. This means that it is possible to lose data in the event of a failure. If data is lost, the recovery point is the beginning of the oldest open transaction. This means that you cannot guarantee an RPO that relies on asynchronous mirroring for availability, since it will be nondeterministic. There is also no support for automatic failover in this configuration. Therefore, asynchronous mirroring offers a DR solution, as opposed to a high availability solution. The diagram in Figure 2-5 illustrates a mirroring topology, configured in High Performance mode.

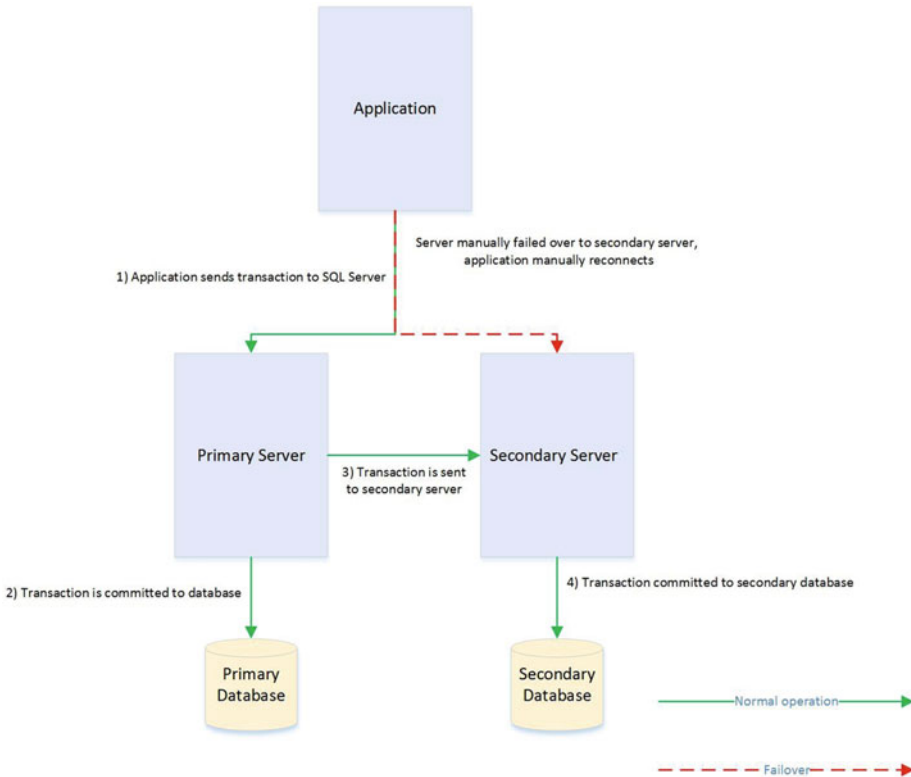


Figure 2-5. Database mirroring in High Performance mode

When running in High Safety with Automatic Failover mode, data is committed at the secondary server using a synchronous method, as opposed to an asynchronous method. This means that the data is committed on the secondary server before it is committed on the primary server. This can cause performance degradation and requires a fast network link between the two servers. The network latency should be less than 3 milliseconds.

In order to support automatic failover, the database mirroring topology needs to form a quorum. In order to achieve quorum, it needs a third server. This server is known as the witness server and it is used to arbitrate in the event that the primary and secondary servers lose network connectivity. For this reason, if the primary and secondary servers are in separate sites, it is good practice to place the witness server in the same data center as the primary server, as opposed to with the secondary server. This can reduce the likelihood of a failover caused by a network outage between the data centers, which makes them become isolated. The diagram in Figure 2-6 illustrates a database mirroring topology configured in High Protection with Automatic Failover mode.

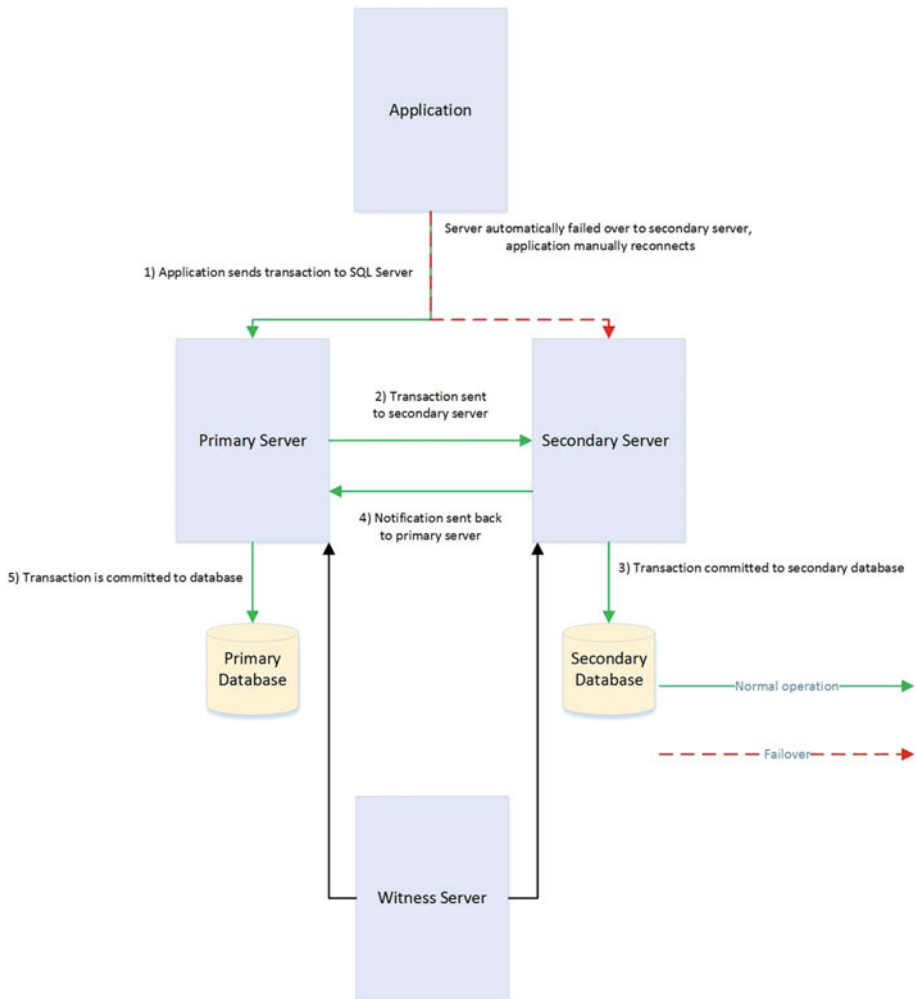


Figure 2-6. Database mirroring in High Safety with Automatic Failover mode

High Safety mode combines the negative aspects of the other two modes. You have the same performance degradation that you expect with High Safety with Automatic Failover, but you also have the manual server failover associated with High Performance mode. The benefit that High Safety mode offers is resilience in the event that the witness goes offline. If database mirroring loses the witness server, instead of suspending the mirroring session to avoid a split-brain scenario, it switches to High Safety mode. This means that database mirroring continues to function, but without automatic failover. High Safety mode is also useful in planned failover scenarios. If your primary server is online, but you need to fail over for maintenance, then you can change to High Safety mode. This essentially puts the database in a safe state, where there is no possibility of

data loss, without you needing to configure a witness server. You can then fail over the database. After the maintenance work is complete and you have failed the database back, then you can revert to High Performance mode.

■ **Tip** Database mirroring is not supported on databases that use In-Memory OLTP. You will be unable to configure database mirroring, if your database contains a memory-optimized filegroup.

AlwaysOn Availability Groups

AlwaysOn Availability Groups (AOAG) replaces database mirroring and is essentially a merger of database mirroring and clustering technologies. SQL Server is installed as a stand-alone instance (as opposed to an AlwaysOn Failover Clustered Instance) on each node of a cluster. A cluster-aware application, called an Availability Group Listener, is then installed on the cluster; it is used to direct traffic to the correct node. Instead of relying on shared disks, however, AOAG compresses the log stream and sends it to the other nodes, in a similar fashion to database mirroring.

Until SQL Server 2016, AlwaysOn Availability Groups were only supported in Enterprise Edition of SQL Server. In SQL Server 2016, however, Standard Edition supports AlwaysOn Availability Groups, with basic functionality only. The basic functionality in Standard Edition supports two replicas only. This is intended as a replacement for Standard Edition support for Database mirroring, in High Safety mode only.

AOAG is the most appropriate technology for high availability in scenarios where you have small databases with low write profiles. This is because, when used synchronously, it requires that the data is committed on all synchronous replicas before it is committed on the primary database. Unlike with database mirroring, however, you can have up to eight replicas, including two synchronous replicas. AOAG may also be the most appropriate technology for implementing high availability in a virtualized environment. This is because the shared disk required by clustering may not be compatible with some features of the virtual estate. As an example, VMware does not support the use of vMotion, which is used to manually move virtual machines (VMs) between physical servers, and the Distributed Resource Scheduler (DRS), which is used to automatically move VMs between physical servers, based on resource utilization, when the VMs use shared disks, presented over Fiber Channel.

■ **Tip** The limitations surrounding shared disks with VMware features can be worked around by presenting the storage directly to the guest OS over an iSCSI connection. This is at the expense of performance degradation, however.

AOAG is the most appropriate technology for DR when you have a proactive failover requirement but when you do not need to implement a load delay. AOAG may also be suitable for disaster recovery in scenarios where you wish to utilize your DR server for offloading reporting. When used for disaster recovery, AOAG works in an asynchronous mode. This means that it is possible to lose data in the event of a failover. The RPO is nondeterministic and is based on the time of the last uncommitted transaction.

When you use database mirroring, the secondary database is always offline. This means that you cannot use the secondary database to offload any reporting or other read-only activity. It is possible to work around this by creating a database snapshot against the secondary database and pointing read-only activity to the snapshot. This can still be complicated, however, because you must configure your application to issue read-only statements against a different network name and IP Address. Availability Groups, on the other hand, allow you to configure one or more replicas as readable. The only limitation is that readable replicas and automatic failover cannot be configured on the same secondaries. The norm, however, would be to configure readable secondary replicas in Asynchronous Commit mode so that they do not impair performance.

To further simplify this, the Availability Group Replica checks for the read-only or read-intent properties in an application's connection string and points the application to the appropriate node. This means that you can easily scale reporting and database maintenance routines horizontally with very little development effort and with the applications being able to use a single connection string.

In SQL Server 2016, load balancing for readable secondary replicas has been introduced. This functionality allows you to specify groups of readable secondaries that the read-only workload will be balanced across. This is in contrast to previous versions, where traffic was routed to the first available replica.

Because AOAG allows you to combine synchronous replicas (with or without automatic failover), asynchronous replicas, and replicas for read-only access, it allows you to satisfy high availability, disaster recovery, and reporting scale-out requirements using a single technology.

When you are using AOAG, failover does not occur at the database level, or at the instance level. Instead, failover occurs at the level of the Availability Group. The Availability Group is a concept that allows you to group similar databases together so that they can fail over as an atomic unit. This is particularly useful in consolidated environments, because it allows you to group together the databases that map to a single application. You can then fail over this application to another replica for the purposes of DR testing, among other reasons, without having an impact on the other data-tier applications that are hosted on the instance. SQL Server 2016 extends the Availability Group concept, to allow you to group databases from separate clusters into a single Availability Group. This is especially useful if you are implementing DR for a data-tier application that is dispersed across separate clusters, but where all databases need to fail over atomically, to the DR site. This functionality is known as a Distributed Availability Group.

No hard limits are imposed for the number of Availability Groups you can configure on an instance, nor are there any hard limits for the number of databases on an instance that can take part in AOAG. Microsoft, however, has tested up to, and officially recommends, a maximum of 100 databases and 10 Availability Groups per instance. The main limiting factor in scaling the number of databases is that AOAG uses a database mirroring endpoint and there can only be one per instance. This means that the log stream for all data modifications is sent over the same endpoint.

Figure 2-7 depicts how you can map data-tier applications to Availability Groups for independent failover. In this example, a single instance hosts two data-tier applications. Each application has been added to a separate Availability Group. The first Availability Group has failed over to Node2. Therefore, the Availability Group Listeners point traffic for Application1 to Node2 and traffic for Application2 to Node1. Because each Availability Group has its own network name and IP Address, and because these resources fail over with the AOAG, the application is able to seamlessly reconnect to the databases after failover.

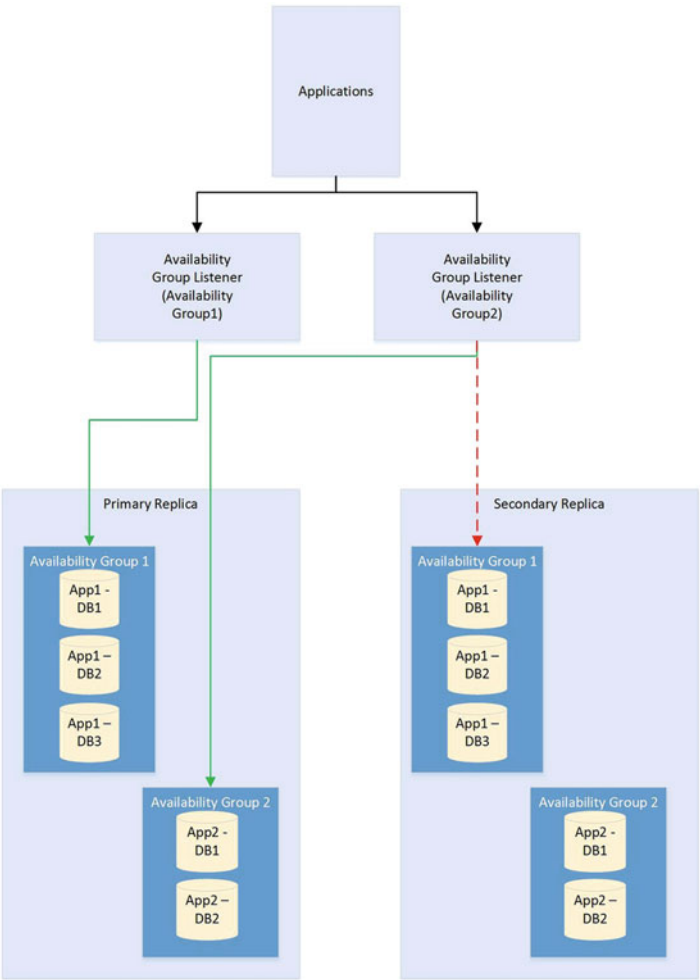


Figure 2-7. Availability groups failover

The diagram in Figure 2-8 depicts an AlwaysOn Availability Group topology. In this example, there are four nodes in the cluster and a disk witness. Node1 is hosting the primary replicas of the databases, Node2 is being used for automatic failover, Node3 is being used to offload reporting, and Node4 is being used for DR. Because the cluster is stretched across two data centers, multi-subnet clustering has been implemented. Because there is no shared storage, however, there is no need for SAN replication between the sites.

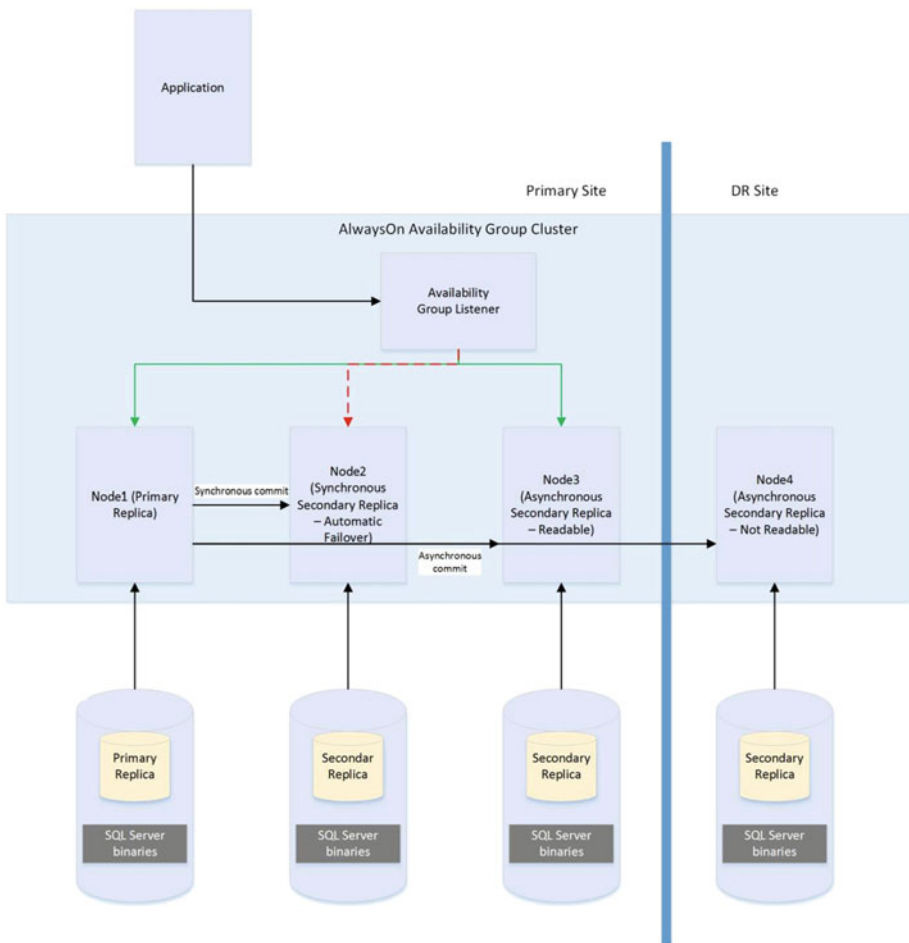


Figure 2-8. AlwaysOn Availability Group topology

■ **Note** AlwaysOn Availability Groups are discussed in more detail in Chapter 5 and Chapter 6.

Automatic Page Repair

If a page becomes corrupt in a database configured as a replica in an AlwaysOn Availability Group topology, then SQL Server attempts to fix the corruption by obtaining a copy of the pages from one of the secondary replicas. This means that a logical corruption can be resolved without you needing to perform a restore or for you to run DBCC CHECKDB with a repair option. However, automatic page repair does not work for the following page types:

- File Header page
- Database Boot page
- Allocation pages
- GAM (Global Allocation Map)
- SGAM (Shared Global Allocation Map)
- PFS (Page Free Space)

If the primary replica fails to read a page because it is corrupt, it first logs the page in the `MSDB.dbo.suspect_pages` table. It then checks that at least one replica is in the `SYNCHRONIZED` state and that transactions are still being sent to the replica. If these conditions are met, then the primary sends a broadcast to all replicas, specifying the `PageID` and `LSN` (log sequence number) at the end of the flushed log. The page is then marked as `restore pending`, meaning that any attempts to access it will fail, with error code 829.

After receiving the broadcast, the secondary replicas wait, until they have redone transactions up to the `LSN` specified in the broadcast message. At this point, they try to access the page. If they cannot access it, they return an error. If they can access the page, they send the page back to the primary replica. The primary replica accepts the page from the first secondary to respond.

The primary replica will then replace the corrupt copy of the page with the version that it received from the secondary replica. When this process completes, it updates the page in the `MSDB.dbo.suspect_pages` table to reflect that it has been repaired by setting the `event_type` column to a value of 5 (Repaired).

If the secondary replica fails to read a page while redoing the log because it is corrupt, it places the secondary into the `SUSPENDED` state. It then logs the page in the `MSDB.dbo.suspect_pages` table and requests a copy of the page from the primary replica. The primary replica attempts to access the page. If it is inaccessible, then it returns an error and the secondary replica remains in the `SUSPENDED` state.

If it can access the page, then it sends it to the secondary replica that requested it. The secondary replica replaces the corrupt page with the version that it obtained from the primary replica. It then updates the `MSDB.dbo.suspect_pages` table with an `event_id` of 5. Finally, it attempts to resume the AOAG session.

Note It is possible to manually resume the session, but if you do, the corrupt page is hit again during the synchronization. Make sure you repair or restore the page on the primary replica first.

Log Shipping

Log shipping is a technology that you can use to implement disaster recovery. It works by backing up the transaction log on the principal server, copying it to the secondary server, and then restoring it. It is most appropriate to use log shipping in DR scenarios in which you require a load delay, because this is not possible with AOAG. As an example of where a load delay may be useful, consider a scenario in which a user accidentally deletes all of the data from a table. If there is a delay before the database on the DR server is updated, then it is possible to recover the data for this table, from the DR server, and then repopulate the production server. This means that you do not need to restore a backup to recover the data. Log shipping is not appropriate for high availability, since there is no automatic failover functionality. The diagram in Figure 2-9 illustrates a log shipping topology.

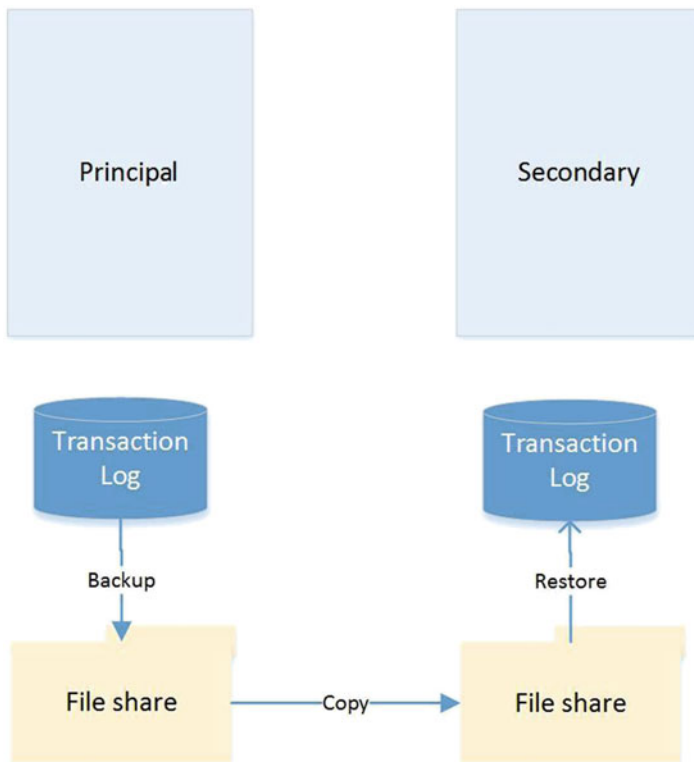


Figure 2-9. Log shipping topology

Recovery Modes

In a log shipping topology, there is always exactly one principal server, which is the production server. It is possible to have multiple secondary servers, however, and these servers can be a mix of DR servers and servers used to offload reporting.

When you restore a transaction log, you can specify three recovery modes: Recovery, NoRecovery, and Standby. The Recovery mode brings the database online, which is not supported with Log Shipping. The NoRecovery mode keeps the database offline so that more backups can be restored. This is the normal configuration for log shipping and is the appropriate choice for DR scenarios.

The Standby option brings the database online, but in a read-only state so that you can restore further backups. This functionality works by maintaining a TUF (Transaction Undo File). The TUF file records any uncommitted transactions in the transaction log. This means that you can roll back these uncommitted transactions in the transaction log, which allows the database to be more accessible (although it is read-only). The next time a restore needs to be applied, you can reapply the uncommitted transaction in the TUF file to the log before the redo phase of the next log restore begins.

Figure 2-10 illustrates a log shipping topology that uses both a DR server and a reporting server.

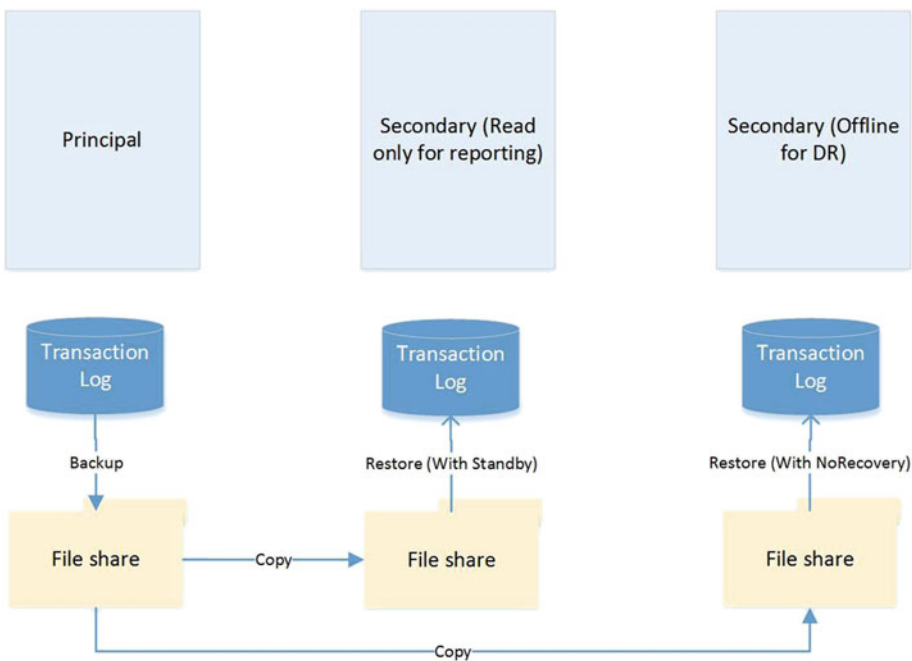


Figure 2-10. Log shipping with DR and reporting servers

Remote Monitor Server

Optionally, you can configure a monitor server in your log shipping topology. This helps you centralize monitoring and alerting. When you implement a monitor server, the history and status of all backup, copy, and restore operations are stored on the monitor server. A monitor server also allows you to have a single alert job, which is configured to monitor the backup, copy, and restore operations on all servers, as opposed to it needing separate alerts on each server in the topology.

■ **Caution** If you wish to use a monitor server, it is important to configure it when you set up log shipping. After log shipping has been configured, the only way to add a monitor server is to tear down and reconfigure log shipping.

Failover

Unlike other high availability and disaster recovery technologies, an amount of administrative effort is associated with failing over log shipping. To fail over log shipping, you must back up the tail end of the transaction log and copy it, along with any other uncopied backup files, to the secondary server.

You now need to apply the remaining transaction log backups to the secondary server in sequence, finishing with the tail-log backup. You apply the final restore using the `WITH RECOVERY` option to bring the database back online in a consistent state. If you are not planning to fail back, you can reconfigure log shipping with the secondary server as the new primary server.

Combining Technologies

To meet your business objectives and nonfunctional requirements (NFRs), you need to combine multiple high availability and disaster recovery technologies together to create a reliable, scalable platform. A classic example of this is the requirement to combine an AlwaysOn Failover Cluster with AlwaysOn Availability Groups.

The reason you may need to combine these technologies is that when you use AlwaysOn Availability Groups in synchronous mode, which you must do for automatic failover, it can cause a performance impediment. As discussed earlier in this chapter, the performance issue is caused by the transaction being committed on the secondary server before being committed on the primary server. Clustering does not suffer from this issue, however, because it relies on a shared disk resource, and therefore the transaction is only committed once.

Therefore, it is common practice to first use a cluster to achieve high availability and then use AlwaysOn Availability Groups to perform DR and/or offload reporting. The diagram in Figure 2-11 illustrates a HA/DR topology that combines clustering and AOAG to achieve high availability and disaster recovery, respectively.

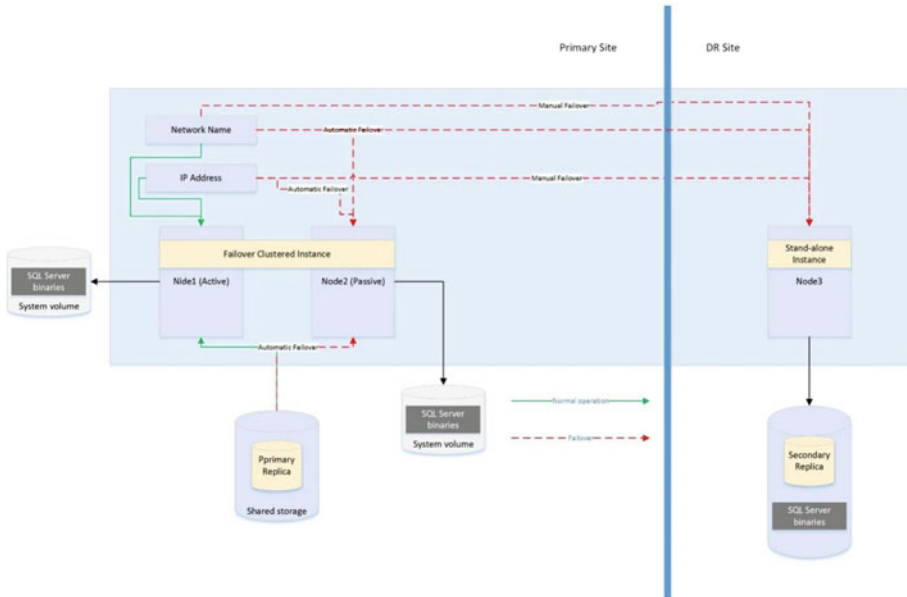


Figure 2-11. Clustering and AlwaysOn Availability Groups combined

The diagram in Figure 2-11 shows that the primary replica of the database is hosted on a two-node active/passive cluster. If the active node fails, the rules of clustering apply, and the shared storage, network name, and IP Address are reattached to the passive node, which then becomes the active node. If both nodes are inaccessible, however, the Availability Group Listener points the traffic to the third node of the cluster, which is situated in the DR site and is synchronized using log stream replication. Of course, when asynchronous mode is used, the database must be failed over manually by a DBA.

Another common scenario is the combination of a cluster and log shipping to achieve high availability and disaster recovery, respectively. This combination works in much the same way as clustering combined with AlwaysOn Availability Groups and is illustrated in Figure 2-12.

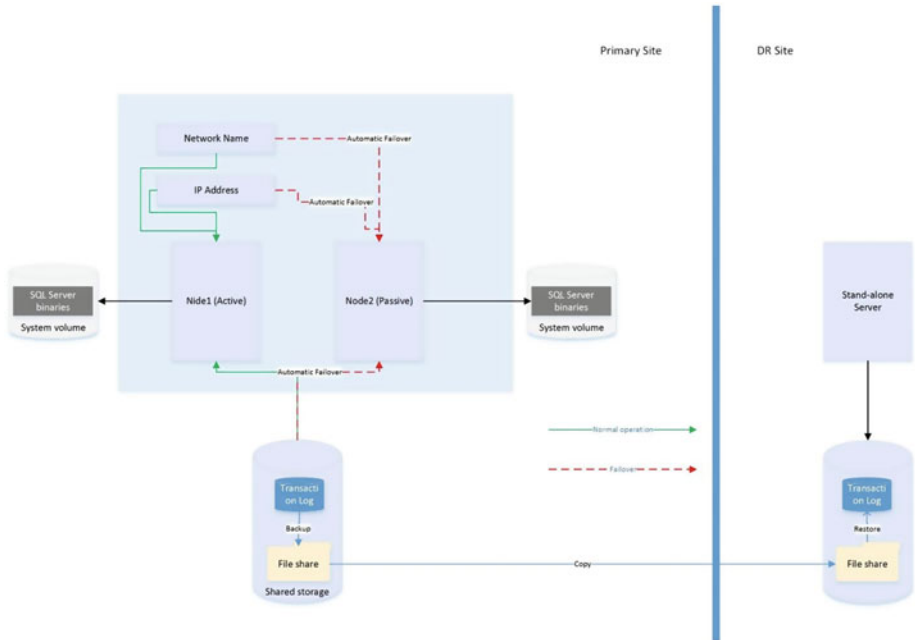


Figure 2-12. Clustering combined with log shipping

The diagram shows that a two-node active/passive cluster has been configured in the primary data center. The transaction log(s) of the database(s) hosted on this instance are then shipped to a stand-alone server in the DR data center. Because the cluster uses shared storage, you should also use shared storage for the backup volume and add the backup volume as a resource in the role. This means that when the instance fails over to the other node, the backup share also fails over, and log shipping continues to synchronize, uninterrupted.

■ **Caution** If failover occurs while the log shipping backup or copy jobs are in progress, then log shipping may become unsynchronized and require manual intervention. This means that after a failover, you should check the health of your log shipping jobs.

Summary

SQL Server provides a full suite of high availability and disaster recovery technologies, giving you the flexibility to implement a solution that best fits the needs of your data-tier applications. For high availability, you can implement either an AlwaysOn cluster or AlwaysOn Availability Groups (AOAG). Clustering uses a shared disk resource and failover occurs at the instance level. AOAG, on the other hand, synchronizes data at the

database level by maintaining a redundant copy of the database with a synchronous log stream. Database mirroring is also available in SQL Server 2014, but it is a deprecated feature and will be removed in a future version of SQL Server.

To implement disaster recovery, you can choose to implement AOAG or log shipping. Log shipping works by backing up, copying, and restoring the transaction logs of the databases, whereas AOAG synchronizes the data using an asynchronous log stream.

It is also possible to combine multiple HA and DR technologies together in order to implement the most appropriate availability strategy. Common examples of this are combining clustering for high availability with AOAG or log shipping to provide DR.



Implementing a Cluster

Engineers may find the process of building and configuring a cluster to be complex and that they can implement many variations of the pattern. Although DBAs may not always need to build a cluster themselves, they do need to be comfortable with the technology and often need to provide their input into the process. They may also take part in troubleshooting issues discovered with the cluster.

For these reasons, this chapter discusses how to build a cluster at the Windows level and discusses some of the possible configurations. The demonstrations in this chapter use a prebuilt environment, consisting of two servers: `ClustNode1` and `ClustNode2`. Both servers reside in a domain named `AlwaysOnRevealed.com`. Four volumes have been presented to the nodes from a SAN, and have been brought online and formatted on `ClusterNode1`, with the configuration detailed in Table 3-1.

Table 3-1. *Disk Configuration*

Drive Letter	Volume Label	Size	Comments
F	Data	4.88GB	Host data and log files
G	MSDTC	972MB	Host files associated with the MSDTC Role
H	Quorum	461MB	Host a disk-based quorum witness
I	TempDB	1.96GB	Host the TempDB data and log files

Tip You may be surprised that there is a single volume allocated for data and log files, as a DBA's natural instinct is to separate these files onto separate drives. The important thing to remember here is that we are working with a SAN, and there is a very strong chance that even if we used separate volumes, those volumes would reside on the same physical spindles, meaning that separation is logical only. Also, if SAN snapshots are to be used, some SANs may require the data and log files to be stored on the same volume, to ensure data consistency.

The scenario in this chapter requires us to build a two-node failover cluster, with a disk witness. Before we can do this, we will need to configure the Windows Cluster Service (WCS). We also need to configure an MSDTC (Microsoft Distributed Transaction Coordinator) Cluster Role, which will provide distributed transaction coordination for SSIS (SQL Server Integration Services). Additionally, we need to configure the cluster to MSDTC role, so that failovers occur with High priority (compared to other roles on the same cluster), that three failovers are allowed within any 24-hour period, and that immediate failback is permitted.

Therefore, the complete list of tasks that we will perform is as follows:

- Install the Failover Cluster feature
- Build a Windows Cluster, called ALWAYSON-C
- Correctly configure the Quorum
- Create a Cluster role for MSDTC, called ALWAYSON-MSDTC-C
- Configure the properties of the MSDTC role
- Configure the Failover properties of the MSDTC role

■ **Tip** If you wish to build a cluster for learning purposes, but you do not have access to a domain, or a SAN, then the newer features of clustering allow you to simulate a very similar topology. Two virtual machines can be used as the cluster nodes. A third virtual machine, running the iSCSI Target feature of Windows can be used to present shared storage to each of these nodes. Even better, Windows Server 2016 allows a cluster to be created on a workgroup, meaning that there is no need to create an additional VM to use as a domain controller. Be warned, however, that creating a cluster within a workgroup is only supported in PowerShell, and not through Failover Cluster Manager (Correct in Windows Server 2016 CTP5). It is also important to be aware that from a SQL Server perspective, Availability Groups are supported on a workgroup cluster, but failover clustered instances are not.

Building the Cluster

Before you install a SQL Server AlwaysOn failover cluster instance, you must prepare the servers that form the cluster (known as nodes) and build a Windows cluster across them. The following sections demonstrate how to perform these activities.

Installing the Failover Cluster Feature

In order to build the cluster, the first thing we need to do is install the Failover Cluster feature on each of the nodes. To do this, we need to select the Add Roles and Features option in Server Manager. This causes the Add Roles and Features Wizard to display. The first page of this wizard offers guidance on prerequisites, as shown in Figure 3-1.

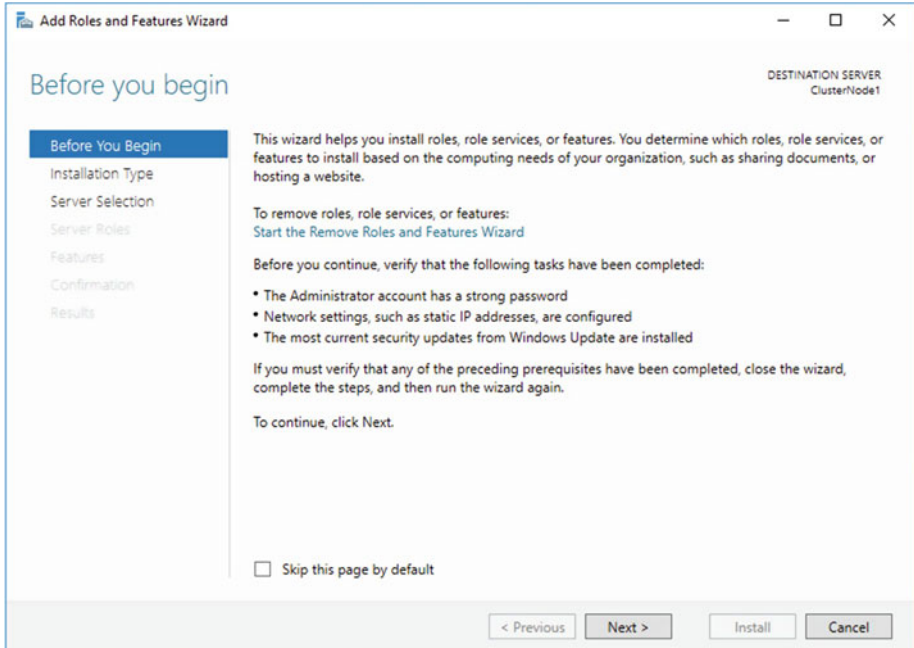


Figure 3-1. The *Before You Begin* page

On the Installation Type page, ensure that Role-Based or Feature-Based Installation is selected, as illustrated in Figure 3-2.

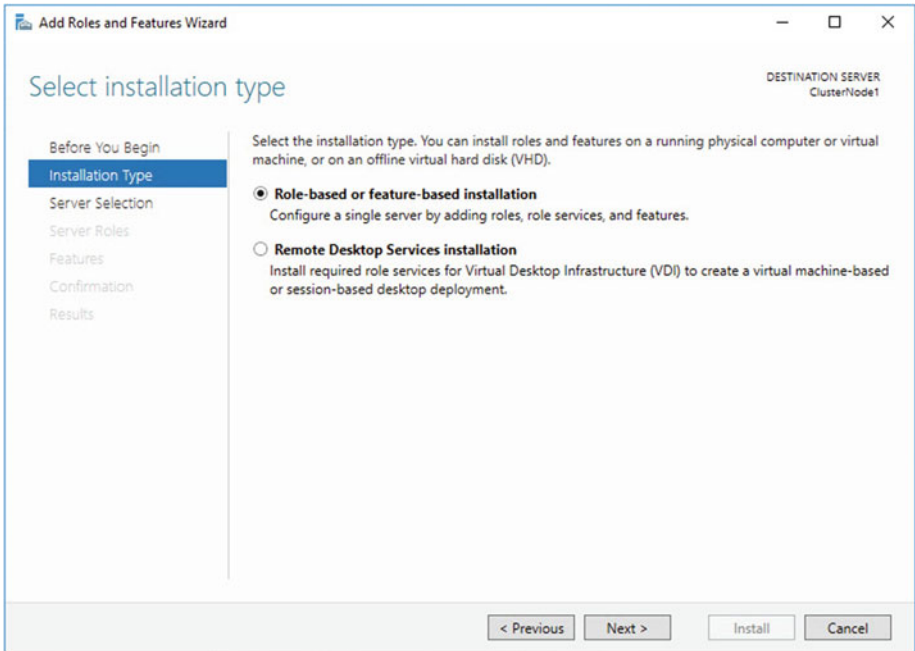


Figure 3-2. *The Installation Type page*

On the Server Selection page, ensure that the cluster node that you are currently configuring is selected. This is illustrated in Figure 3-3.

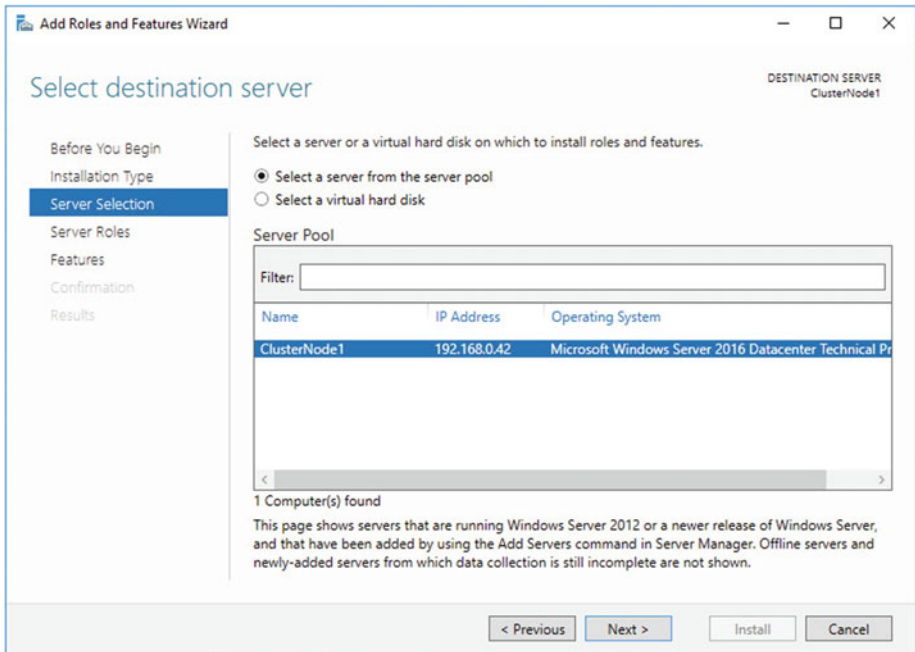


Figure 3-3. The Server Selection page

The Server Roles page of the wizard allows you to select any server roles that you wish to configure. As shown in Figure 3-4, this can include roles such as Application Server or DNS Server, but in our case, this is not appropriate, so we simply move to the next screen.

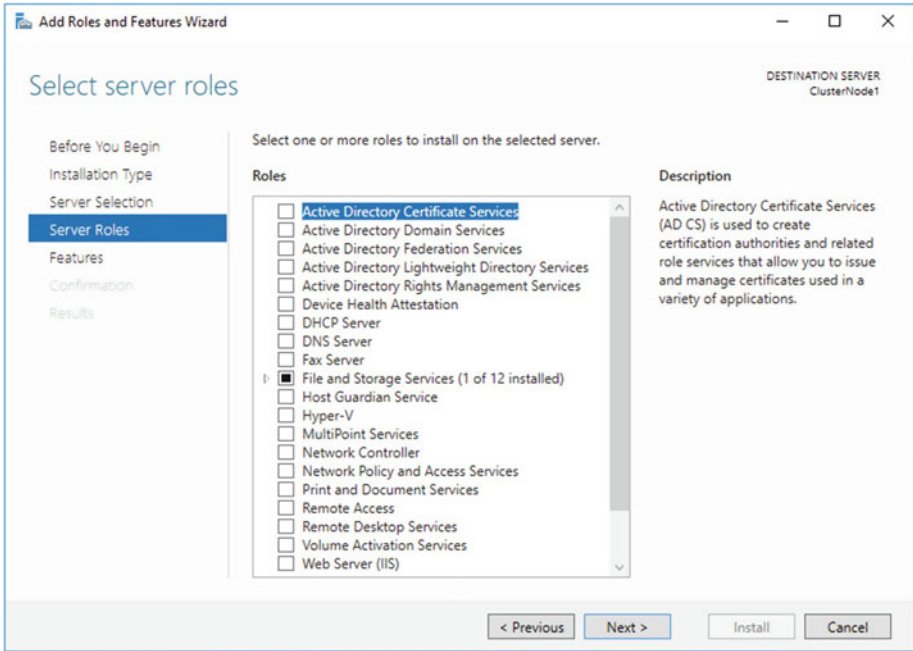


Figure 3-4. The Server Roles page

On the Features page of the wizard, we need to select Failover Clustering, as shown in Figure 3-5. This satisfies the prerequisites for building the Windows cluster.

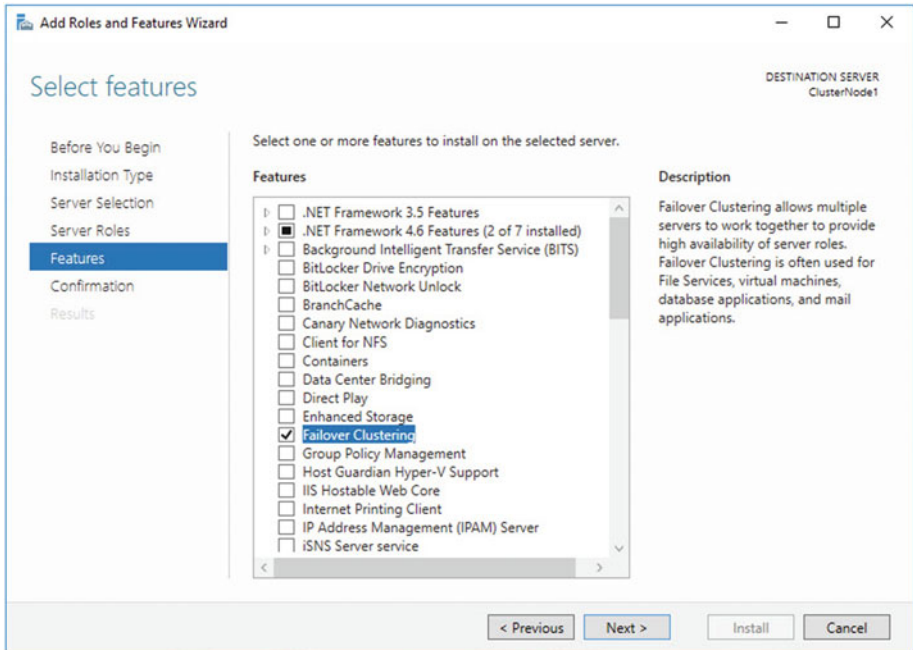


Figure 3-5. The Features page

When you select Failover Clustering, the wizard presents you with a screen (Figure 3-6) that asks if you want to install the management tools in the form of a checkbox. If you are planning to manage the cluster directly from the nodes, check this option.

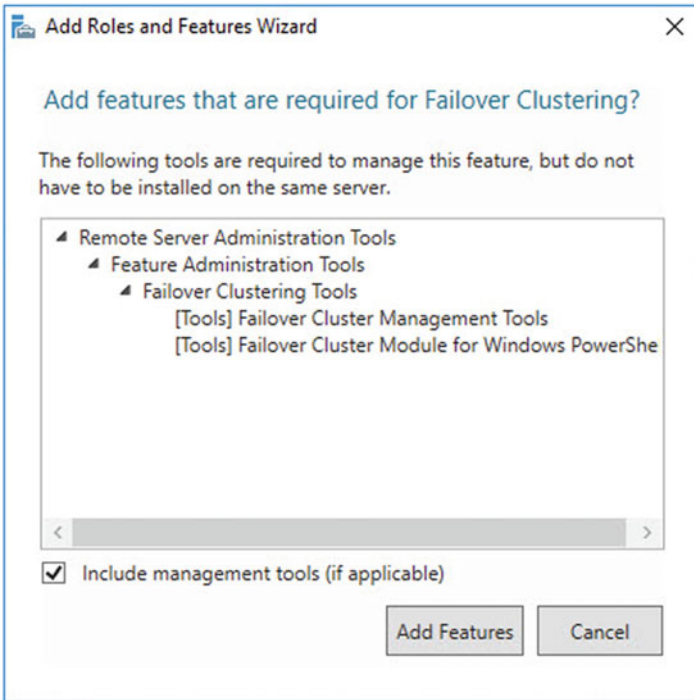


Figure 3-6. *Selecting management tools*

On the final page of the wizard, you see a summary of the features that are to be installed, as shown in Figure 3-7. Here, you can specify the location of the Windows media if you need to. You can also choose whether the server should automatically restart, if required. If you are building out a new server, it makes sense to check this box. However, if the server is already in production when you add the feature, make sure you consider what is currently running on the box, and whether you should wait for a maintenance window to perform a restart if one is needed.

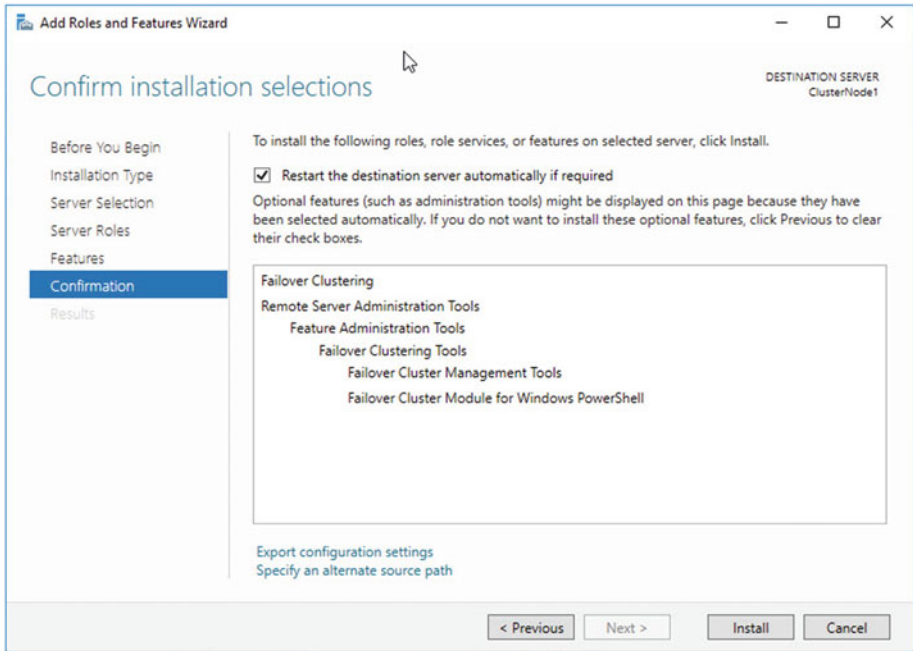


Figure 3-7. The Confirmation page

As well as installation through Server Manager, cluster services can also be installed from PowerShell. The PowerShell command in Listing 3-1 achieves the same result as the preceding steps.

Listing 3-1. Installing Cluster Services

```
Install-WindowsFeature -Name Failover-Clustering -IncludeManagementTools
```

Creating the Cluster

Once clustering has been installed on both nodes, you can begin building the cluster. To do this, connect to the server that you intended to be the active node, and run Failover Cluster Manager from Administrative Tools.

The Before You Begin page of the Create Cluster Wizard warns that Microsoft only supports clusters that pass all verification tests, as shown in Figure 3-8. The message also warns that you must be a local administrator on each node of the cluster. In previous versions of Windows Server, this meant that you must use a domain account that has local administrator rights on each server that will participate in the cluster. In Windows Server 2016, however, the reliance on domain authentication has been removed, and the only requirement is that an account with local administrator rights exists on each node, that has a consistent name and password. This allows the creation of a cluster on a workgroup, or across multiple domains. Neither of these options were available in previous versions of Windows Server.

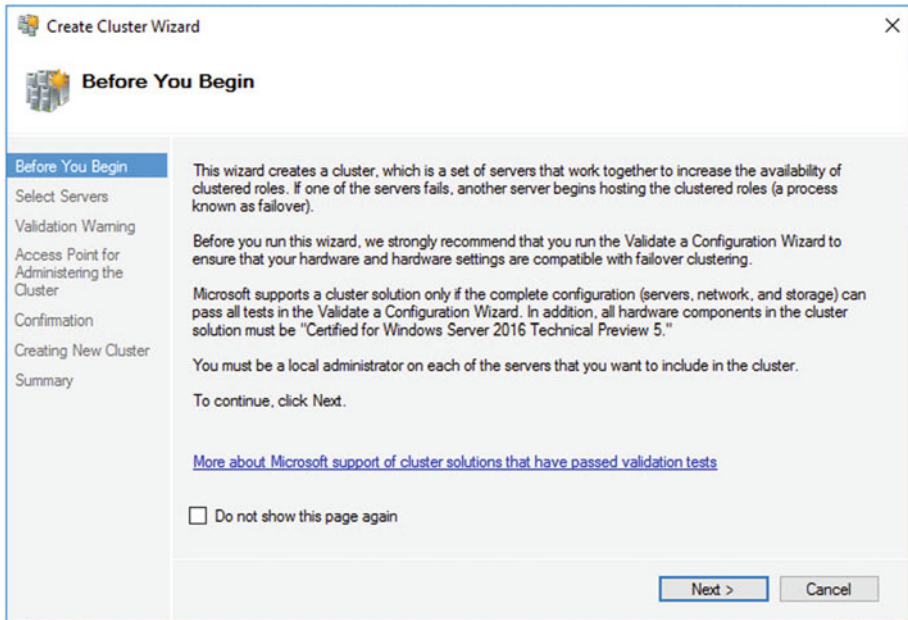


Figure 3-8. The *Before You Begin* page

On the Select Servers screen of the wizard, you need to enter the names of the cluster nodes. This is illustrated in Figure 3-9. In our case, our cluster nodes are named ClusterNode1 and ClusterNode2, respectively. If they were part of a domain, however, then the domain name and suffix would be appended to the server name.

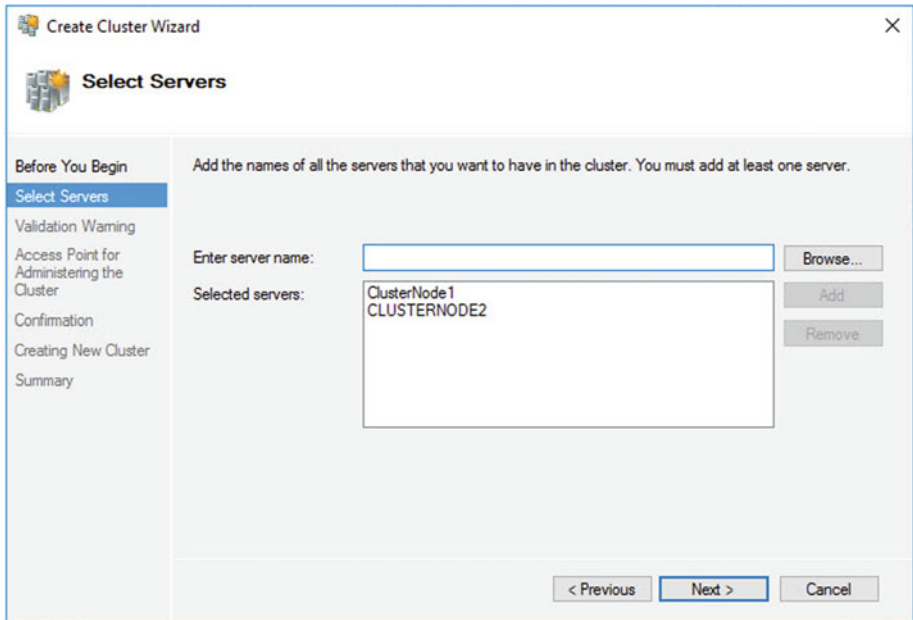


Figure 3-9. *The Select Servers page*

On the Validation Warnings page, you are asked if you wish to run the validation tests against the cluster. You should always choose to run this validation for production servers, because Microsoft will not offer support for the cluster unless it has been validated. Choosing to run the validation tests invokes the Validate a Configuration Wizard. You can also run this wizard independently from the Management pane of Failover Cluster Manager. The Validation Warnings page is shown in Figure 3-10.

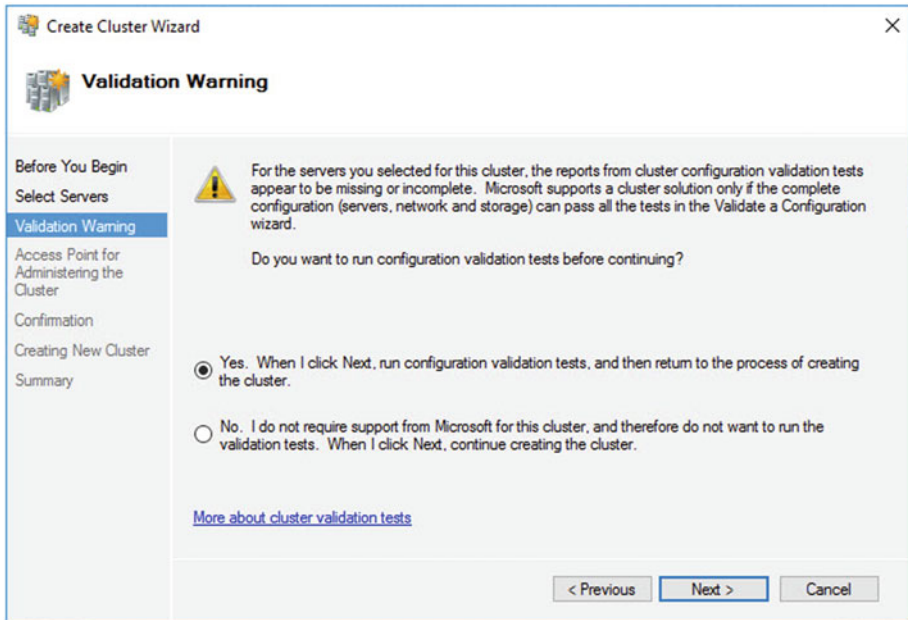


Figure 3-10. *The Validation Warning page*

■ **Tip** There are some situations in which validation is not possible, and in these instances, you need to select the No, I Do Not Require Support... option. For example, some DBAs choose to install one-node clusters instead of stand-alone instances so that they can be scaled up to full clusters in the future, if need be. This approach can cause operational challenges for Windows administrators, however, so use it with extreme caution.

After you pass through the Before You Begin page of the Validate a Configuration Wizard, you see the Testing Options page. Here, you are given the option of either running all validation tests or selecting a subset of tests to run, as illustrated in Figure 3-11. Normally when you are installing a new cluster, you want to run all validation tests, but it is useful to be able to select a subset of tests if you invoke the Validate a Configuration Wizard independently after you make a configuration change to the cluster.

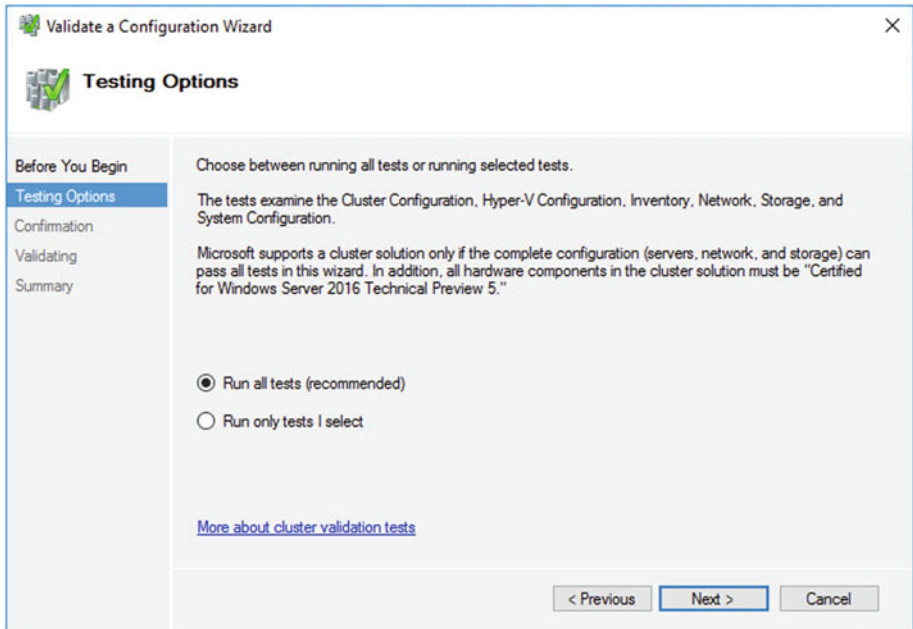


Figure 3-11. *The Testing Options page*

On the Confirmation page of the wizard, illustrated in Figure 3-12, you are presented with a summary of tests that will run and the cluster nodes that they will run against. The list of tests is comprehensive and includes the following categories:

- Inventory (such as identifying any unsigned drivers)
- Network (such as checking for a valid IP configuration)
- Storage (such as validating the ability to fail disks over, between nodes)
- System Configuration (such as validating the configuration of Active Directory)

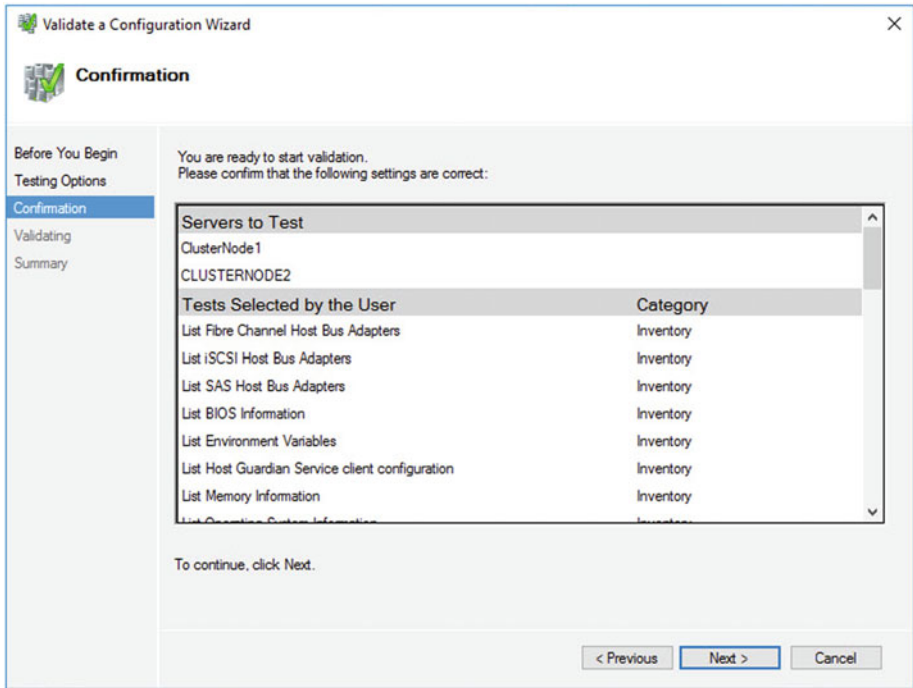


Figure 3-12. The Confirmation page

The Summary page, shown in Figure 3-13, provides the results of the tests and also a link to an HTML version of the report. Make sure to examine the results for any errors or warnings. You should always resolve errors before continuing, but some warnings may be acceptable. For example, if you are building your cluster to host AlwaysOn Availability Groups, you may not have any shared storage. This will generate a warning but is not an issue in this scenario. Configuring AlwaysOn Availability Groups are discussed in further detail in Chapters 5 and 6.

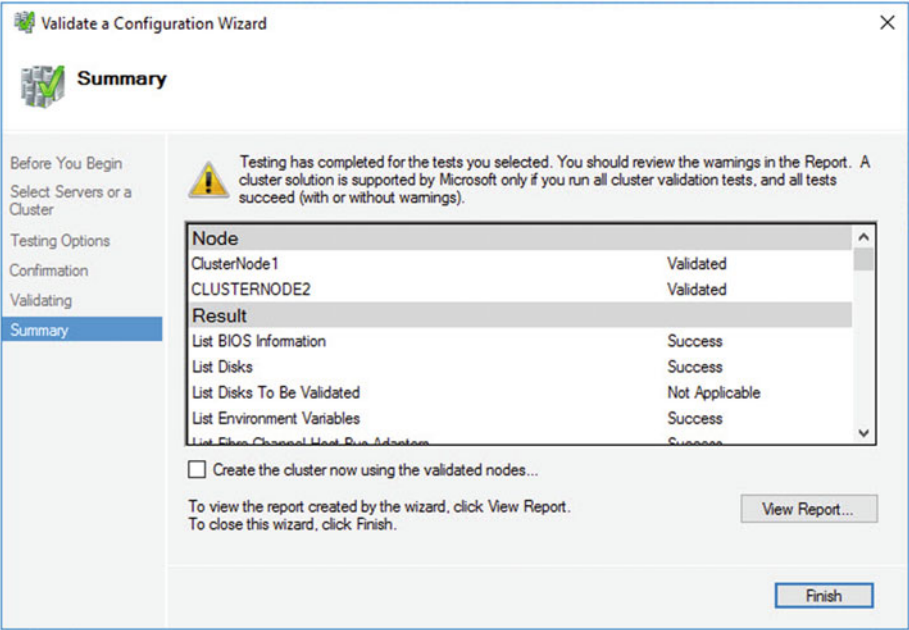


Figure 3-13. The Summary page

The View Report button displays the full version of the validation report, as shown in Figure 3-14. The hyperlinks take you to a specific category within the report, where further hyperlinks are available for each test. These allow you to drill down to messages generated for the specific test, making it easy to identify errors.

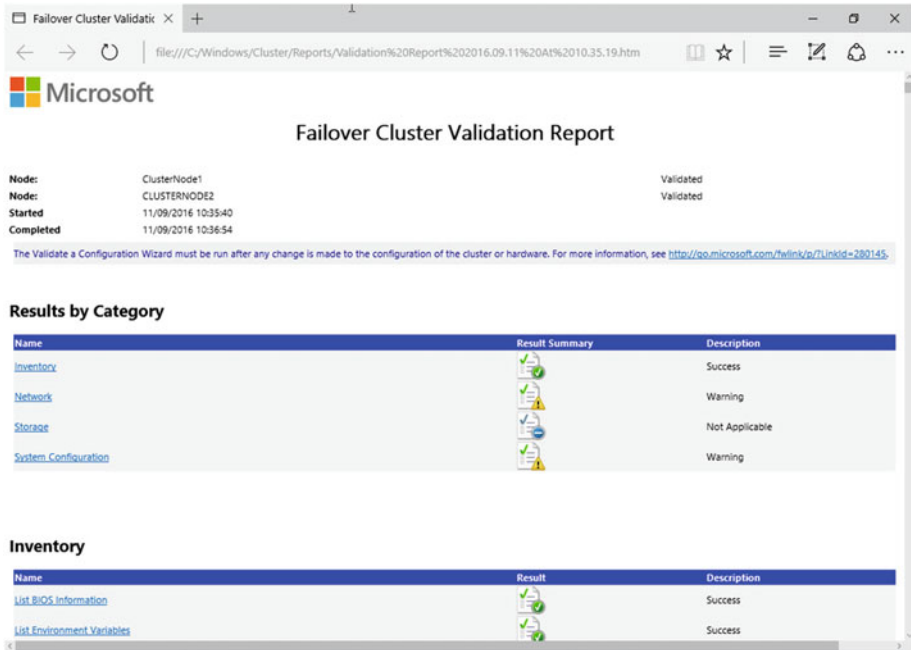


Figure 3-14. *The Failover Cluster Validation Report*

Clicking Finish on the Summary page returns you to the Create Cluster Wizard, where you are greeted with the Access Point for Administering the Cluster page, illustrated in Figure 3-15. On this page, you need to enter the virtual name of your cluster. We will name our cluster ALWAYS-ON-C. If the network card is configured to acquire an IP Address automatically, then an IP Address will be assigned using DHCP. Otherwise, you will be required to enter an IP Address manually. This is known as a static IP, as it will remain constant, whereas an IP Address assigned through DHCP may change. I strongly recommend using a static IP for cluster access points, to avoid dynamic routing issues, but this can be changed after the cluster has been created.

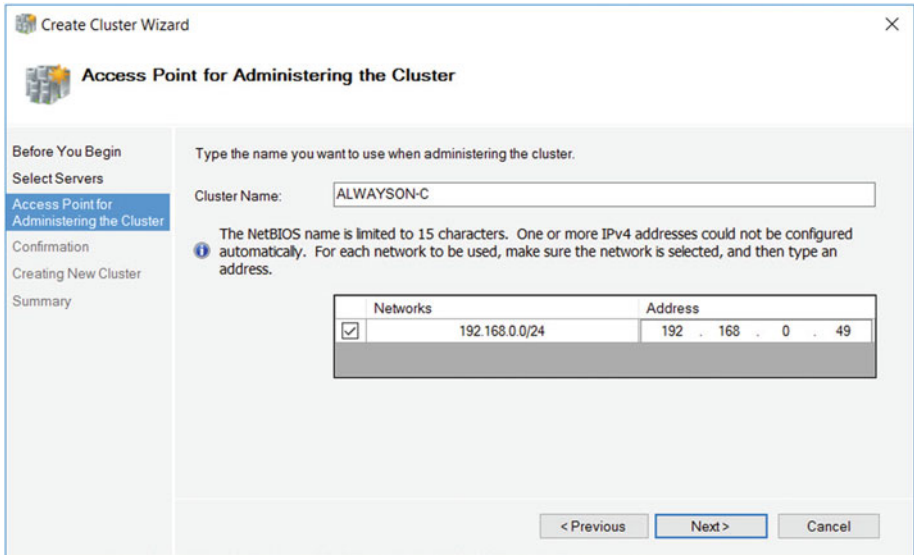


Figure 3-15. The Access Point for Administering the Cluster page

■ **Note** The virtual name and IP Address are bound to whichever node is active, meaning that the cluster is always accessible in the event of failover.

In our case, the cluster resides within a simple domain, a single site and single subnet. If you are configuring a multi-subnet cluster, however, then the wizard detects this, and IP Addresses will be required for each subnet. In this scenario, you need to enter an IP Address for each subnet.

■ **Note** Each of the two NICs within a node is configured on a separate subnet so that the heartbeat between the nodes is segregated from the public network. However, a cluster is only regarded as multi-subnet if the data NICs of the cluster nodes reside in different subnets.

■ **Tip** If your cluster will reside within a domain, and if you do not have permissions to create AD (Active Directory) objects in the OU (organizational unit) that contains your cluster, then the VCO (virtual computer object) for the cluster must already exist and you must have the Full Control permission assigned.

The Confirmation page displays a summary of the cluster that is created. You can also use this screen to specify whether or not all eligible storage should be added to the cluster, which is generally a useful feature. This screen is displayed in Figure 3-16.

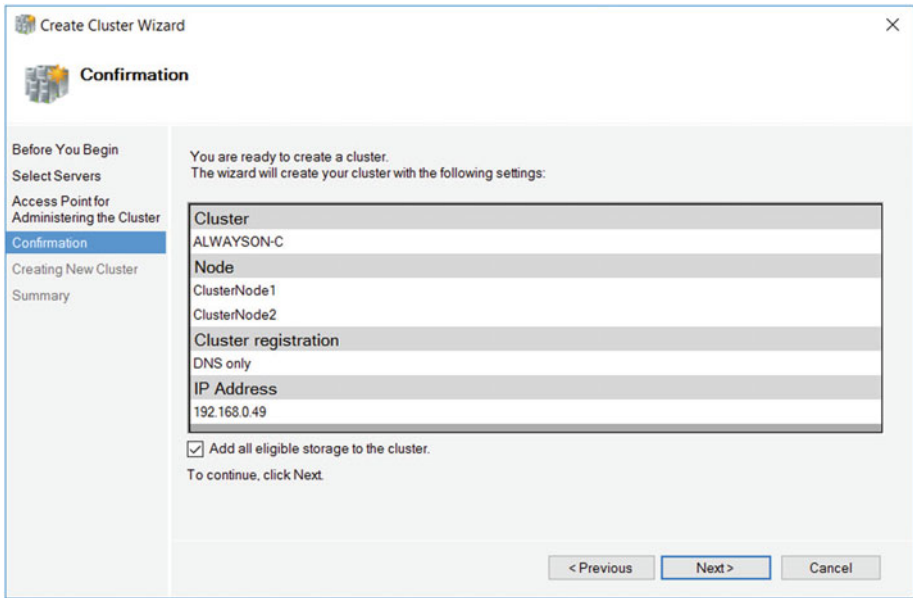


Figure 3-16. The Confirmation page

After the cluster has been built, the Summary page shown in Figure 3-17 displays. This screen summarizes the cluster name, IP Address, nodes, the quorum model that has been configured, and details of any warnings regarding the cluster. It also provides a link to an HTML (Hypertext Markup Language) version of the report.

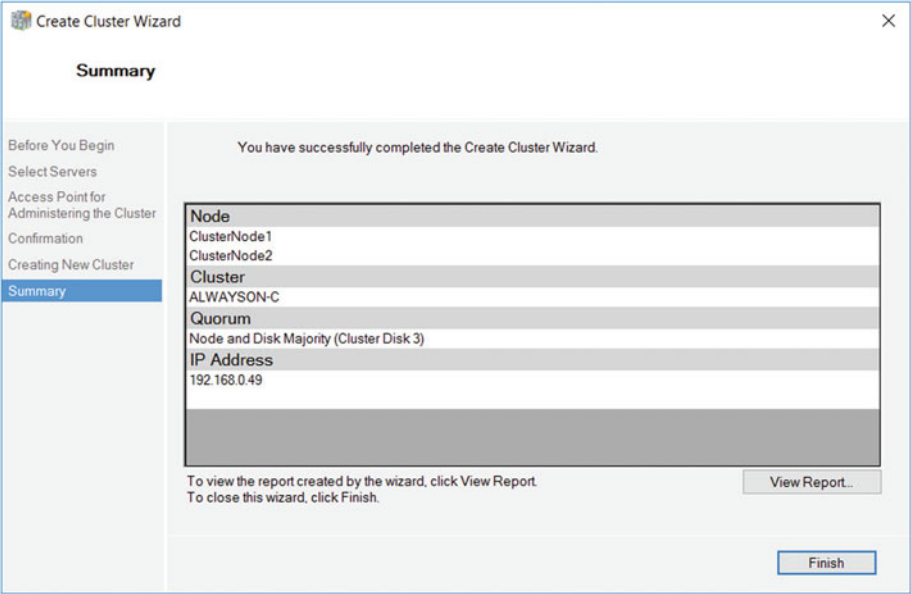


Figure 3-17. The Summary page

The Create Cluster report displays a complete list of tasks that have been completed during the cluster build, as shown in Figure 3-18.

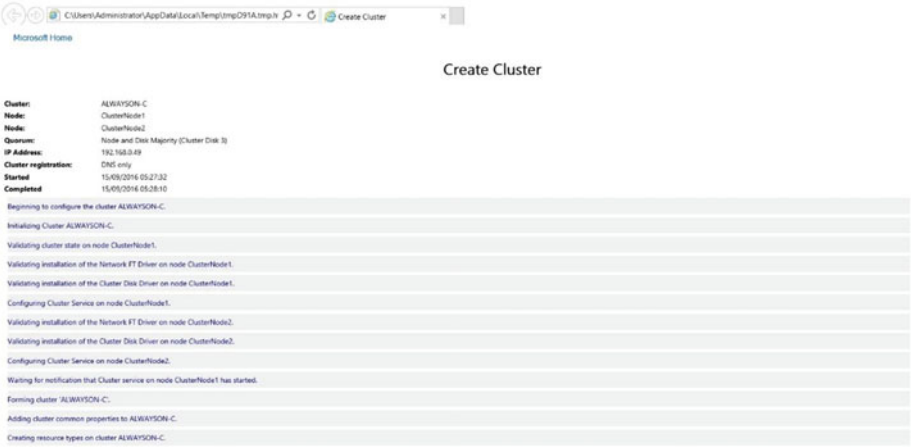


Figure 3-18. The Create Cluster report

We could also have used PowerShell to create the cluster. The script in Listing 3-2 runs the cluster validation tests using the `Test-Cluster` cmdlet, before using the `New-Cluster` cmdlet to configure the cluster.

Listing 3-2. Validating and Creating the Cluster

```
#Run the validation tests
```

```
Test-Cluster -Node Clusternode1,Clusternode2
```

```
#Create the cluster
```

```
New-Cluster -Node ClusterNode1,ClusterNode2 -Name ALWAYSON-C
```

Configuring the Cluster

Many cluster configurations can be altered, depending on the needs of your environment. This section demonstrates how to change some of the more common configurations.

Changing the Quorum

If we examine our cluster storage in the Failover Cluster Manager, by drilling through `ALWAYSON-C | Storage | Disks` and highlighting the disk assigned to quorum, we can see that the witness has been incorrectly configured as the drive for MSDTC. This is illustrated in Figure 3-19.

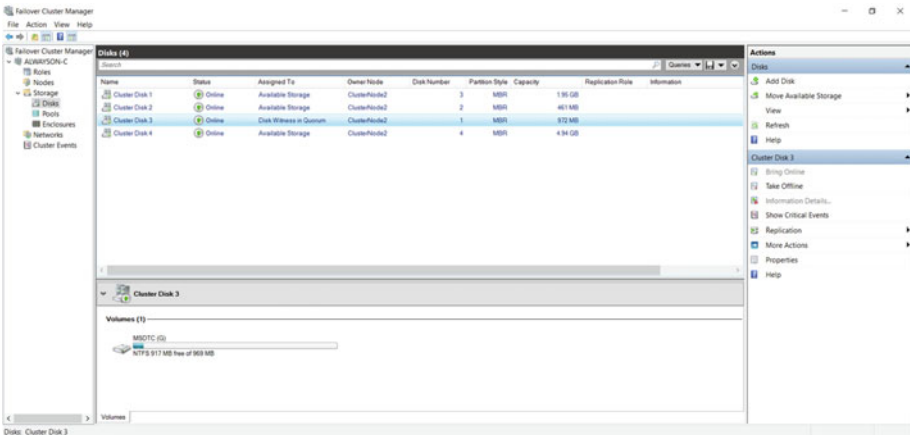


Figure 3-19. Cluster summary

We can modify this by entering the context menu of the cluster and by selecting More Actions | Configure Cluster Quorum Settings, which causes the Configure Cluster Quorum Wizard to be invoked. On the Select Quorum Configuration Option page, shown in Figure 3-20, we choose the Select the Quorum Witness option.

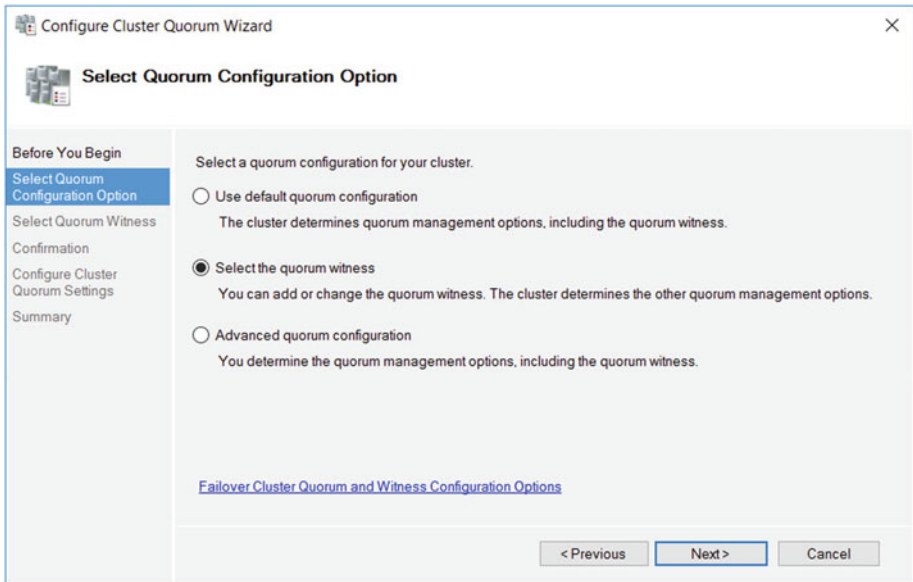


Figure 3-20. The Select Quorum Configuration Option page

On the Select Quorum Witness page, we select the type of quorum that we want to configure. A disk witness is most appropriate when there is an even number of nodes in the cluster and all nodes reside in the same data center, or when there is an even number of nodes in the primary data center and another node in a secondary data center.

A fileshare witness is most appropriate when there are nodes split across two data centers, and you have access to a third data center, where a fileshare is available to act as a quorum.

A cloud witness is a new feature of Windows 2016 and is most appropriate when there are nodes spread across two data centers and there is not a third data center available to set up a fileshare witness. To use a cloud witness, you must have an Azure storage account, as the witness will be created in Azure BLOB storage.

It is most appropriate to not configure a witness where there is an odd number of nodes in a single data center.

For our scenario, we will select the option to configure a disk witness. This is illustrated in Figure 3-21.

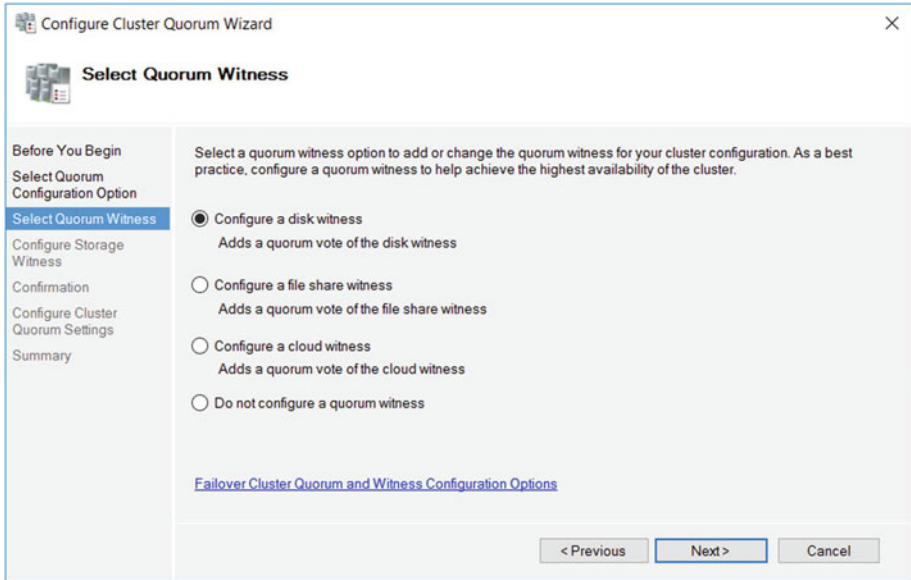


Figure 3-21. The Select Quorum Witness page

On the Configure Storage Witness page of the wizard, we can select the correct disk to use as a quorum. In our case, this is Disk 2, as illustrated in Figure 3-22.

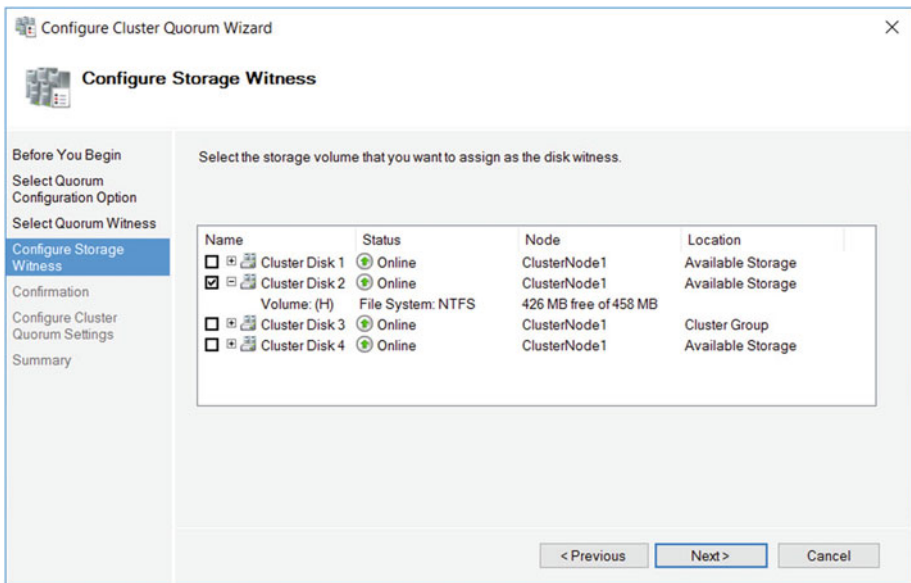


Figure 3-22. The Configure Storage Witness page

The Summary page of the wizard, shown in Figure 3-23, details the configuration changes that will be made to the cluster. It also highlights that dynamic quorum management is enabled and that all nodes, plus the quorum disk, have a vote in the quorum. Advanced quorum configurations are discussed further in Chapter 4.

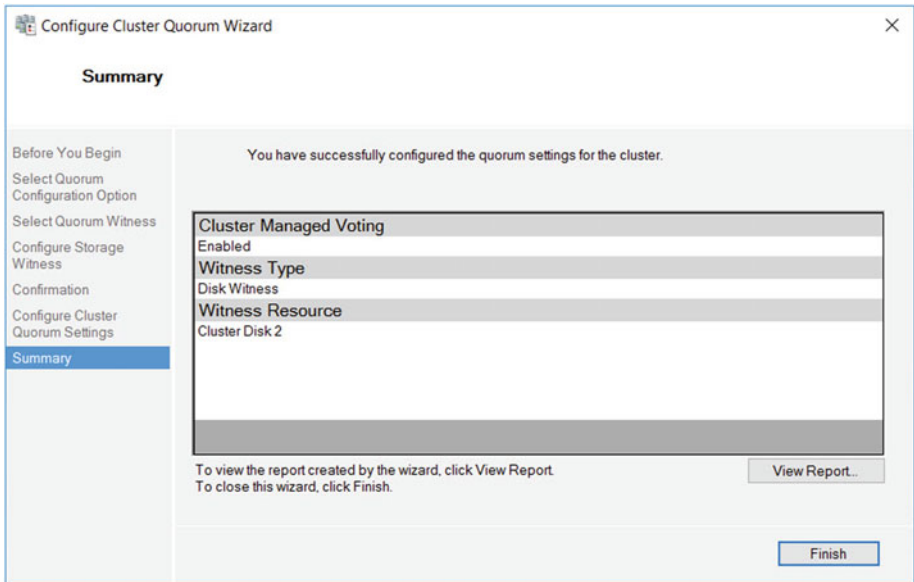


Figure 3-23. The Summary page

We can also perform this configuration from the command line by using the PowerShell command in Listing 3-3. Here, we use the `Set-ClusterQuorum` cmdlet and pass in the name of the cluster, followed by the quorum type that we wish to configure. Because disk is included in this quorum type, we can also pass in the name of the cluster disk that we plan to use, and it is this aspect that allows us to change the quorum disk.

■ **Tip** If following the demos using PowerShell, remember to change the disk number to match your own configuration.

Listing 3-3. Configuring the Quorum Disk

```
Set-ClusterQuorum -Cluster ALWAYSON-C -NodeAndDiskMajority "Cluster Disk 2"
```

Configuring MSDTC

If your instance of SQL Server uses distributed transactions, or if you are installing SQL Server Integration Services (SSIS), then it relies on MSDTC (Microsoft Distributed Transaction Coordinator). If your instance will use MSDTC, then you need to ensure that it is properly configured. If it is not, then setup will succeed, but transactions that rely on it may fail.

When installed on a cluster, SQL Server automatically uses the instance of MSDTC that is installed in the same role, if one exists. If it does not, then it uses the instance of MSDTC to which it has been mapped (if this mapping has been performed). If there is no mapping, it uses the cluster's default instance of MSDTC, and if there is not one, it uses the local machine's instance of MSDTC.

Many DBAs choose to install MSDTC within the same role as SQL Server; however, this introduces a problem. If MSDTC fails, it can also bring down the instance of SQL Server. Of course, the cluster attempts to bring both of the applications up on a different node, but this still involves downtime, including the time it takes to recover the databases on the new node, which takes a nondeterministic duration. For this reason, I recommend installing MSDTC in a separate role. If you do, the SQL Server instance still utilizes MSDTC, since it is the cluster's default instance, and it removes the possibility of MSDTC causing an outage to SQL Server. This is also preferable to using a mapped instance or the local machine instance since it avoids unnecessary configuration, and the MSDTC instance should be clustered when a clustered instance of SQL Server is using it.

To create an MSDTC role, start by selecting the Configure Role option from the Roles context menu in Failover Cluster Manager. This invokes the High Availability Wizard. On the Select a Role page of the wizard, select the Distributed Transaction Coordinator (DTC) role type, as shown in Figure 3-24.

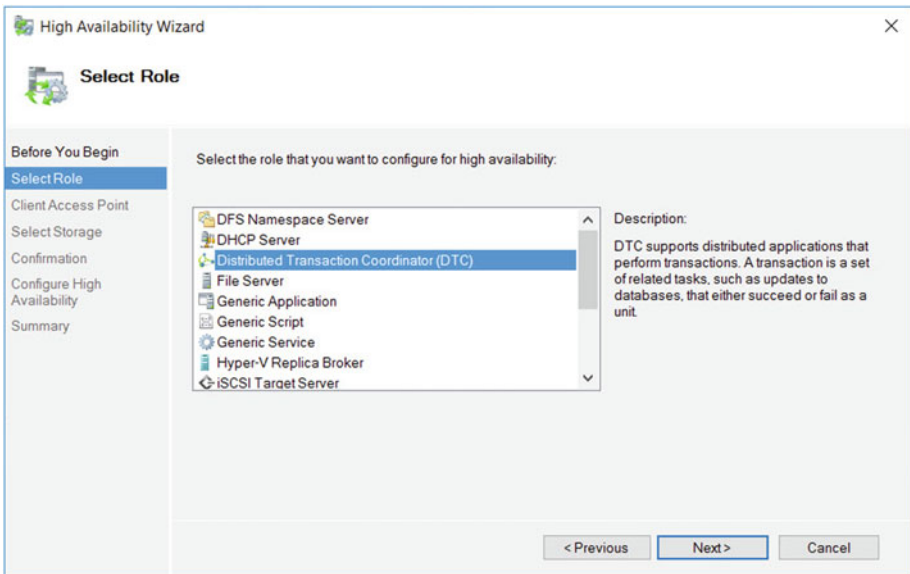


Figure 3-24. The Select Role page

On the Client Access Point page, illustrated in Figure 3-25, you need to enter a virtual name and IP Address for MSDTC. In our case, we name it ALWAYSON-MSDTC-C and assign 192.168.0.50 as the IP Address. On a multi-subnet cluster, you need to provide an IP Address for each network.

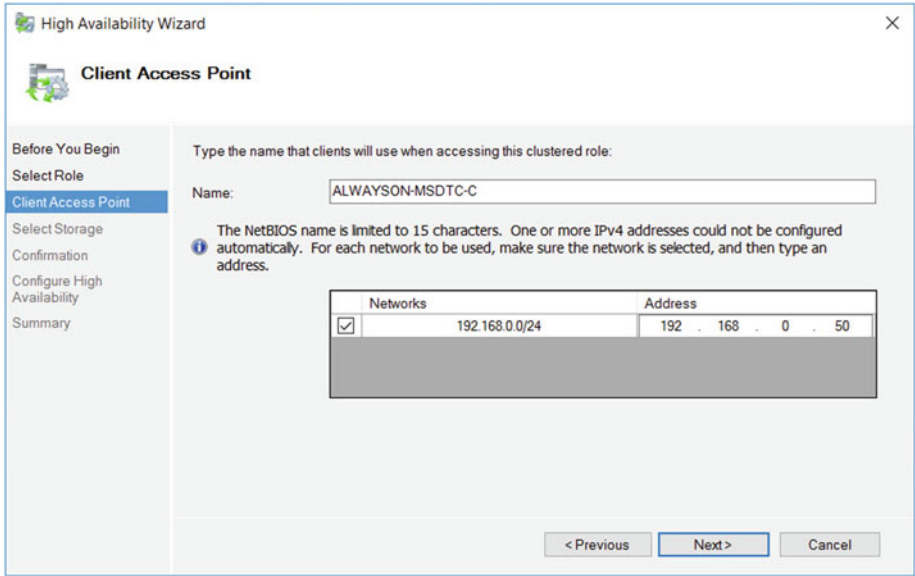


Figure 3-25. The Client Access Point page

On the Select Storage page of the wizard, select the cluster disk on which you plan to store the MSDTC files, as shown in Figure 3-26. In our case, this is Disk 4.

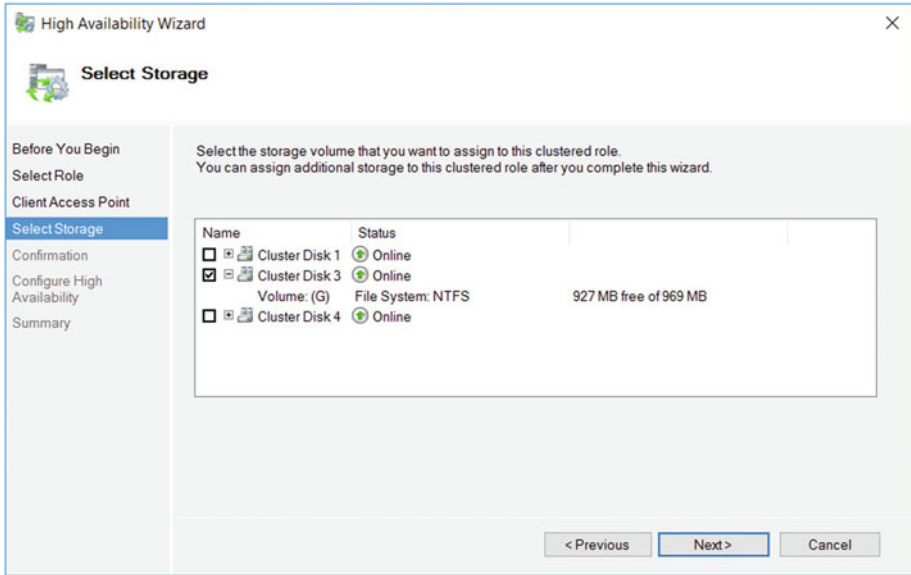


Figure 3-26. The Select Storage page

The Confirmation page displays an overview of the role that is about to be created, as shown in Figure 3-27.

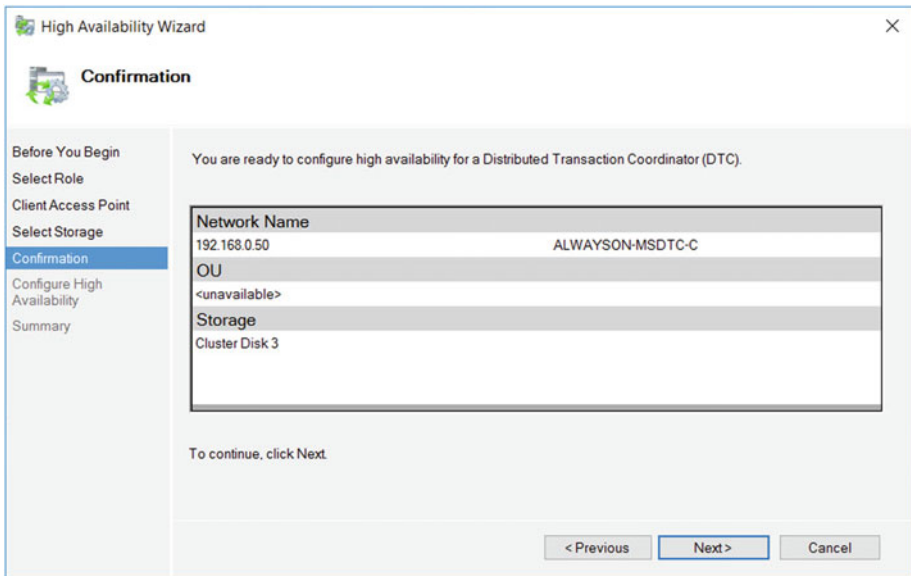


Figure 3-27. The Confirmation page

Alternatively, we could create this role in PowerShell. The script in Listing 3-4 first uses the `Add-ClusterServerRole` cmdlet to create the role. We pass the virtual name to use for the role into the `Name` parameter, the name of the cluster disk to use into the `Storage` parameter, and the IP Address for the role into the `StaticAddress` parameter.

We then use the `Add-ClusterResource` cmdlet to add the DTC resource. The `Name` parameter names the resource and the `ResourceType` parameter specifies that it is a DTC resource. We then need to create the dependencies between the resources within the role. We did not need to do this when using the GUI, as the dependencies were created for us automatically. Resource dependencies specify the resource or resources on which other resources depend. A resource failing propagates through the chain and could take a role offline. For example, in the case of our `ALWAYSON-MSDTC-C` role, if either the disk or the virtual name becomes unavailable, the DTC resource goes offline. Windows Server supports multiple dependencies with both `AND` and `OR` constraints. It is the `OR` constraints that make multi-subnet clustering possible, because a resource can be dependent on IP Address A `OR` IP Address B. Finally, we need to bring the role online by using the `Start-ClusterGroup` cmdlet.

Listing 3-4. Creating an MSDTC Role

```
#Create the Role

Add-ClusterServerRole -Name ALWAYSON-MSDTC-C -Storage "Cluster Disk 3"
-StaticAddress 192.168.0.50

#Create the DTC Resource

Add-ClusterResource -Name MSDTC-ALWAYSON-MSDTC-C -ResourceType "Distributed
Transaction Coordinator" -Group ALWAYSON-MSDTC-C

#Create the dependencies

Add-ClusterResourceDependency MSDTC-ALWAYSON-MSDTC-C ALWAYSON-MSDTC-C

Add-ClusterResourceDependency MSDTC-ALWAYSON-MSDTC-C "Cluster Disk 3"

#Bring the Role online

Start-ClusterGroup ALWAYSON-MSDTC-C
```

Configuring a Role

After creating a role, you may wish to configure it to alter the failover policy or configure nodes as preferred owners. To configure a role, select `Properties` from the role's context menu. On the `General` tab of the `Properties` dialog box, which is shown in Figure 3-28, you can configure a node as the preferred owner of the role. You can also change the order of precedence of node preference by moving nodes above or below others in the `Preferred Owners` window.

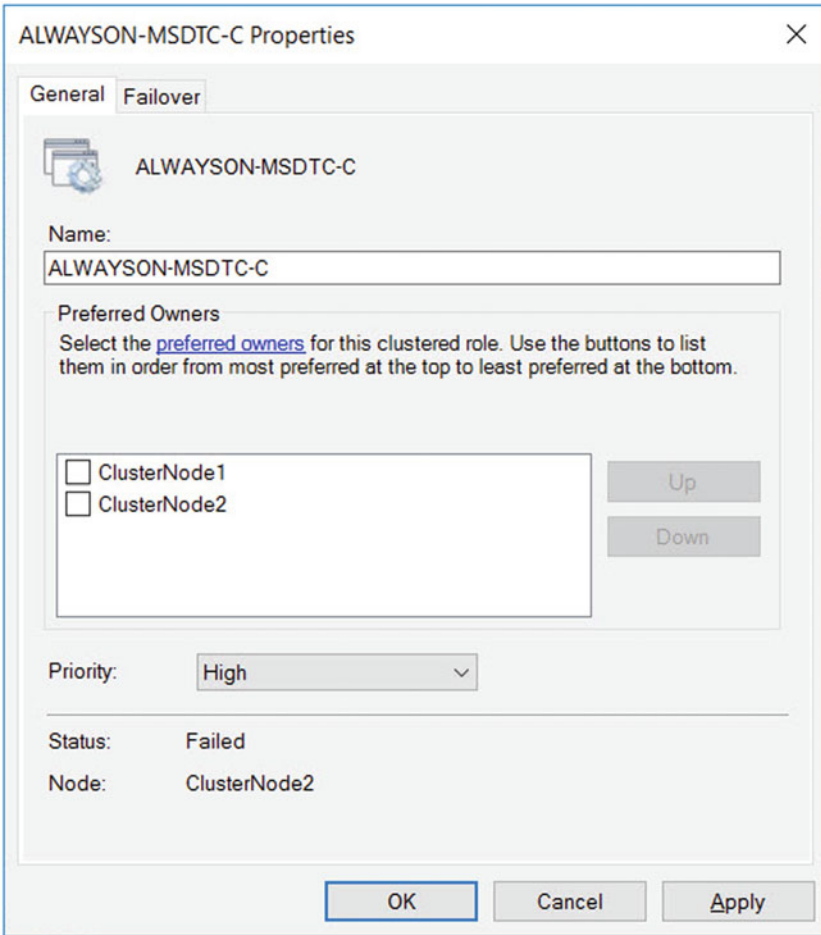


Figure 3-28. The General tab

You can also select the priority for the role in the event that multiple roles fail over to another node at the same time. The options for this setting are as follows:

- High
- Medium
- Low
- No Auto Start

We will configure the ALWAYSON-MSDTC-C role to failover with High priority.

On the Failover tab of the Properties dialog box, you can configure the number of times that the role can fail over within a given period before the role is left offline. The default value for this is one failure within 6 hours. The issue with this is that if a role fails over, and after you fix the issue on the original node, you fail the role back, no more failovers are allowed within the 6-hour window. This is obviously a risk, and I generally advise that you change this setting. In our case, we have configured the role to allow a maximum of three failovers within a 24-hour time window, as illustrated in Figure 3-29. We have also configured the role to fail back to the most preferred owner if it becomes available again. Remember, when setting automatic failback, that failback also causes downtime in the same way that a failover does. If you aspire to a very high level of availability, such as five 9s, then this option may not be appropriate. We will configure the ALWAYSON-MSDTC-C Role to allow three failovers within a 24-hour period. We will also configure the role to allow immediate failback.

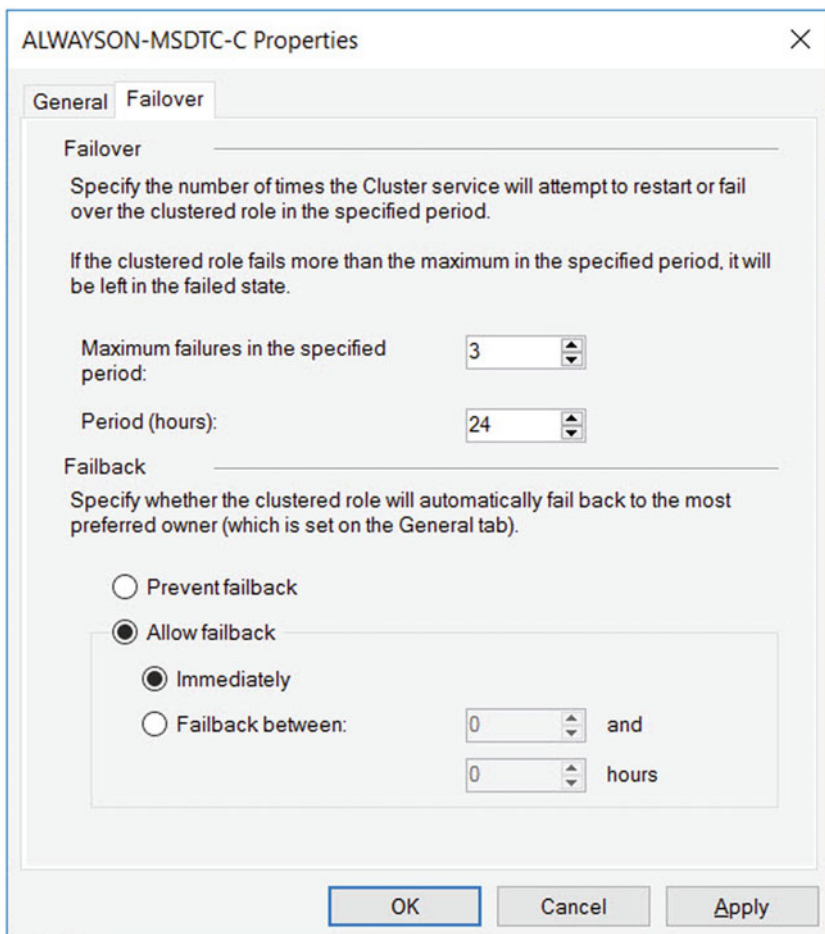


Figure 3-29. The Failover tab

Summary

Before creating the cluster, the Microsoft Cluster Service (MCS) must be installed on all nodes. This can be achieved by installing the Cluster feature, using the Add Roles and Features wizard.

Once the Cluster feature has been installed, clustering can be configured on each node by using the Create Cluster Wizard. Before building the cluster, this wizard will prompt you to run the Cluster Validation Wizard. The Cluster Validation Wizard will validate that environment meets the requirements for a cluster. If you find that your environment does not meet the requirements, you can continue to building the cluster, but the installation will not be supported by Microsoft.

Once the cluster has been built, it will also need to be configured. This will include configuring the quorum mode and may also include configuring MSDTC. After creating a role on the cluster, you may also wish to configure the role with failover policies or preferred owners.



Implementing an AlwaysOn Failover Clustered Instance

Once the cluster has been built and configured, it is time to install the SQL Server AlwaysOn failover cluster instance. In our scenario, we want to build a clustered instance, which spans both nodes of the Windows cluster that we built in Chapter 3. We will also discuss how to build the failover clustered instance using PowerShell. To do this, we will need to use the Install a SQL Server Failover Cluster wizard on the primary node of the cluster. We will then need to run the Add Node wizard, to allow the passive node to host the instance in the event of a failover. Therefore, in this chapter, we will perform the following tasks:

- Install a SQL Server failover clustered instance on the active cluster node
- Configure the passive node of the wizard to support the failover clustered instance

Building the Instance

An AlwaysOn Failover Clustered Instance can be built using the Install a SQL Server Failover Cluster wizard, which can be invoked by opening the SQL Server Installation Center, on the node hosting the cluster core resources, and selecting the New SQL Server Failover Cluster Installation option from the Installation. The installation tab of the SQL Server Installation Center is illustrated in Figure 4-1.

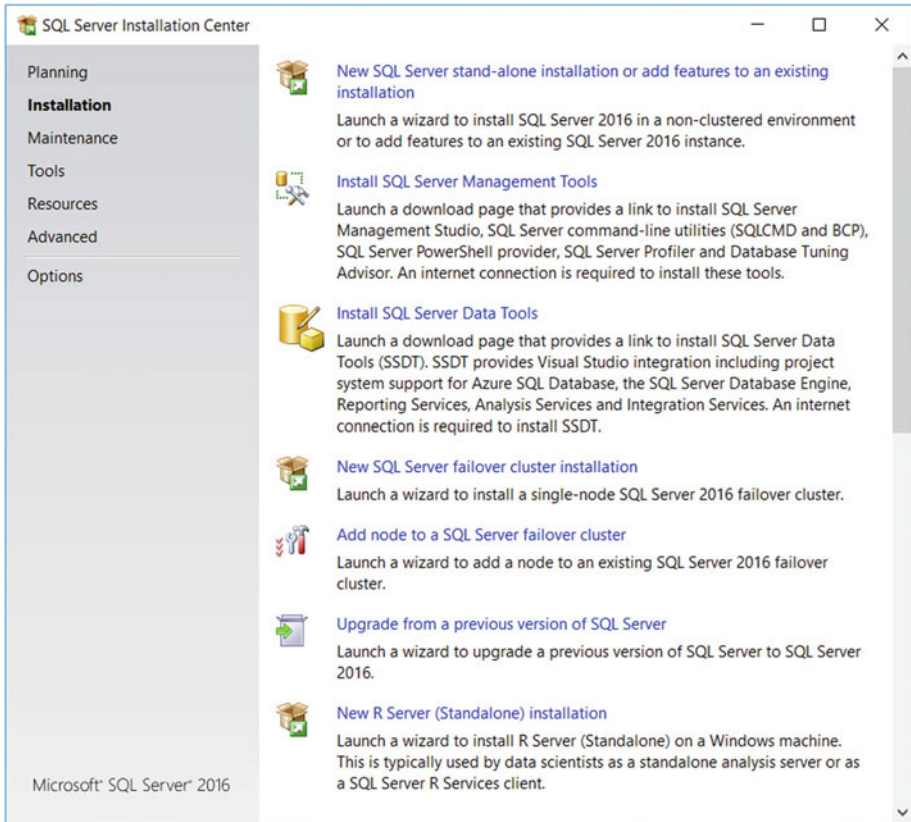


Figure 4-1. SQL Server Installation Center—Installation tab

The Product Key page of the wizard is the first to be displayed, and is illustrated in Figure 4-2. On this page of the wizard, you will either select one of the free versions of SQL Server to install, or enter a product key, or volume licensing key, which will automatically determine the correct version of SQL Server to install.

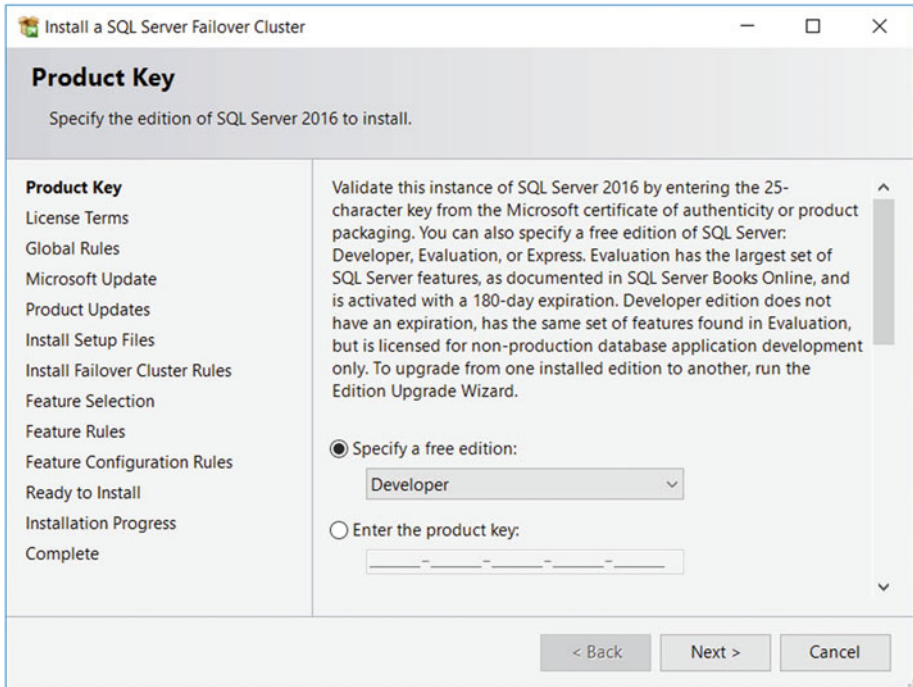


Figure 4-2. *Install a SQL Server Failover Cluster Wizard—Product Key page*

■ **Tip** In SQL Server 2016, Developer Edition, which is functionally equivalent to Enterprise Edition, is free. In previous SQL Server versions, there was a nominal charge associated with this noncommercial license.

On the License Terms page of the wizard (Figure 4-3), you will be invited to accept Microsoft's license terms, via a check box. The installation cannot proceed without agreement.

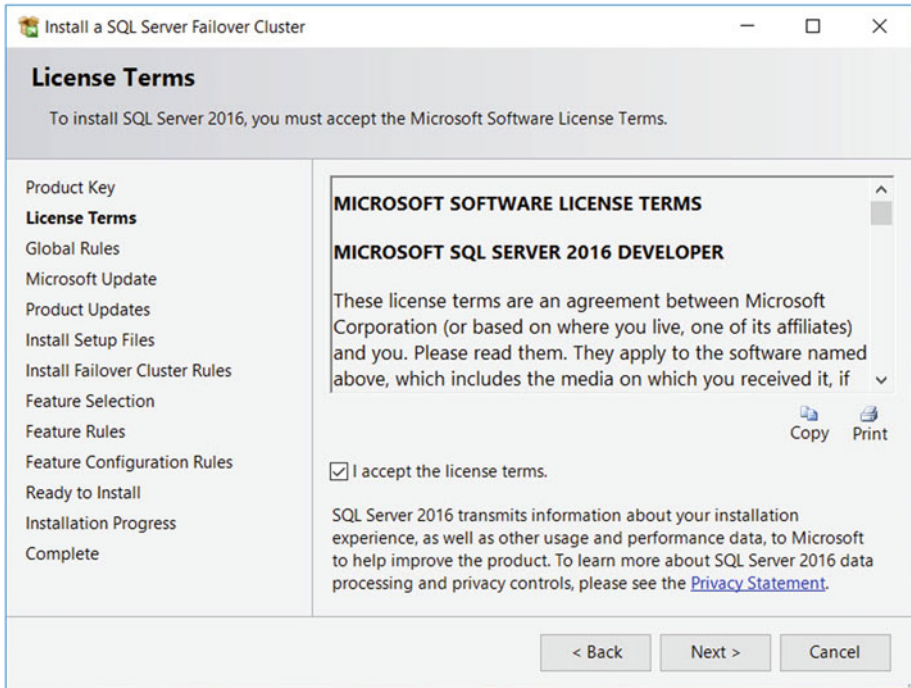


Figure 4-3. *Install a SQL Server Failover Cluster Wizard—License Terms page*

Global Rules will now be checked, to ensure that the setup support files can be successfully installed. When all checks are passed, the Microsoft Update page of the wizard, illustrated in Figure 4-4, will prompt you to choose if you want Windows Update to check for SQL Server patches and hotfixes. The choice here will depend on your organization’s patching policy. Some organizations implement a ridged patching regime for the testing and acceptance of patches, followed by a patching cycle, which is often supported with software such as WSUS (Windows Server Update Services). If such a regime exists in your organization, then you should not select this option.

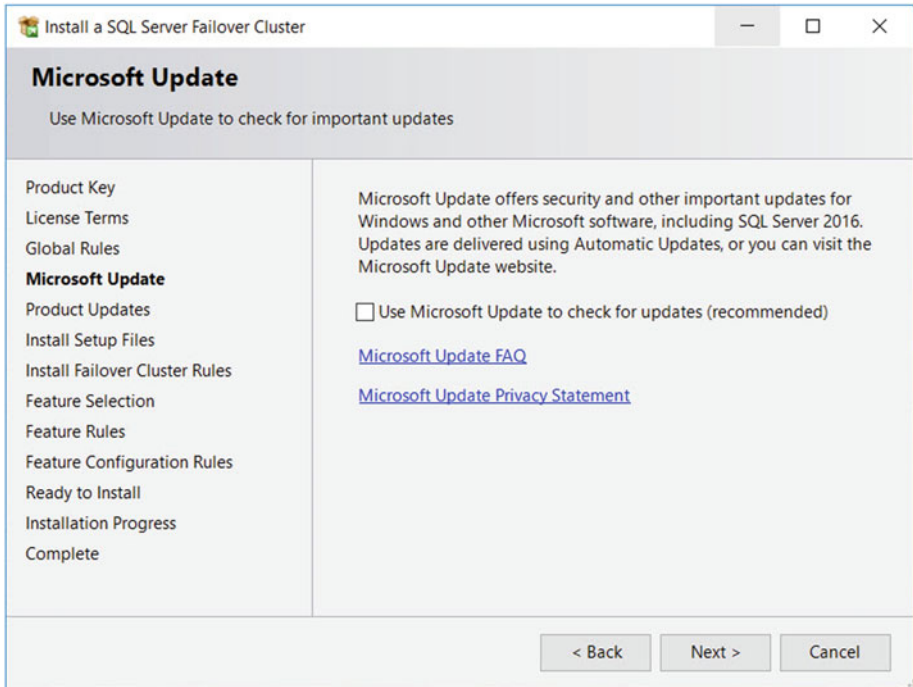


Figure 4-4. *Install a SQL Server Failover Cluster Wizard—Microsoft Update page*

If you have chosen to check for updates, and if any available updates are discovered, then the Product Updates page of the wizard, which is illustrated in Figure 4-5, will list any available updates that have been found. You should confirm if they should be installed or not, using the check box.

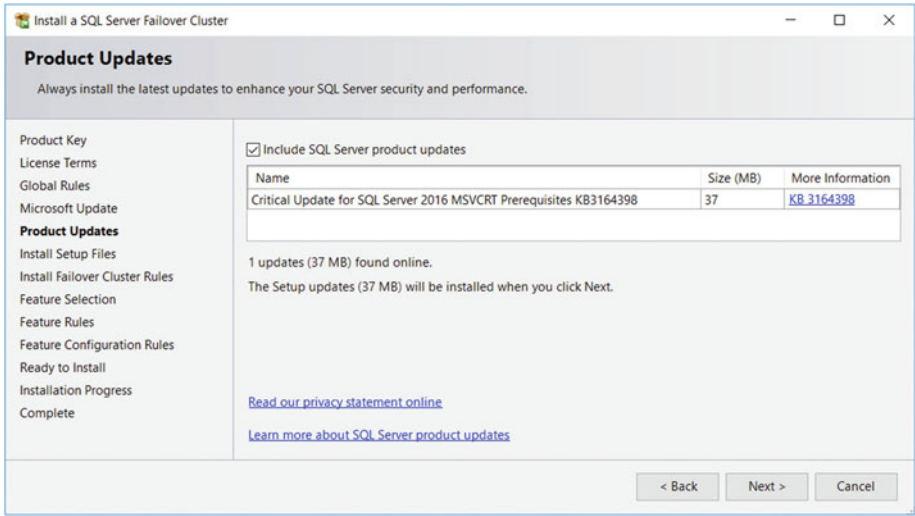


Figure 4-5. *Install a SQL Server Failover Cluster Wizard—Product Updates page*

After the setup support files and any product updates have been downloaded (if applicable), extracted, and installed, installation rules for the installation of a failover clustered instance will be checked, and the results displayed on the Install Failover Cluster Rules page of the wizard, which is shown in Figure 4-6.

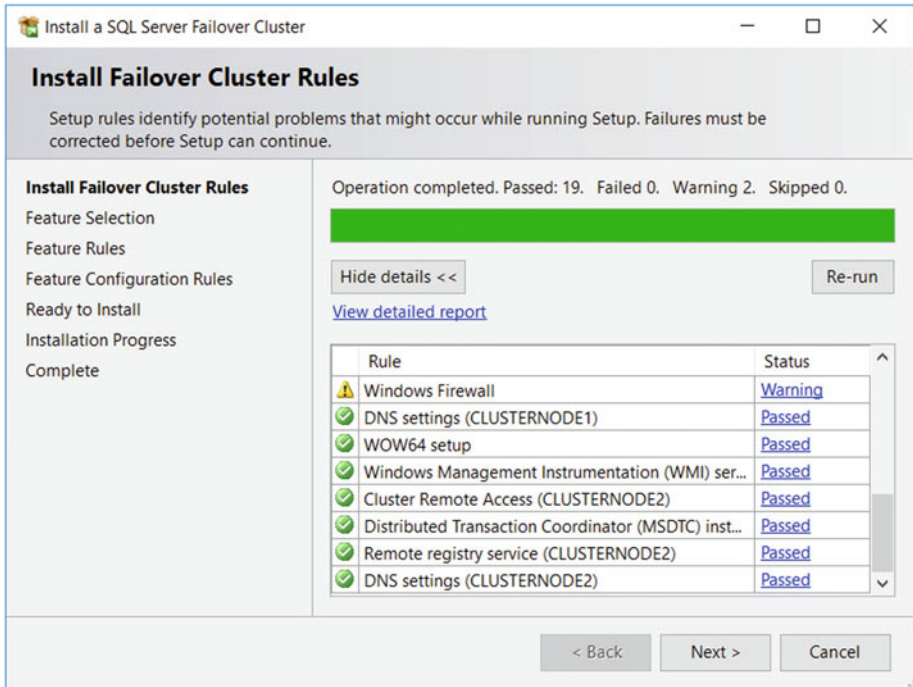


Figure 4-6. *Install a SQL Server Failover Cluster Wizard—Install Failover Cluster Rules*

You will notice that there is a warning regarding the Windows Firewall. This is displayed, simply because the Firewall is turned on. It does not indicate that the required ports are not open. For further information on configuring firewalls for SQL Server, I recommend the Apress book *Securing SQL Server: DBAs Defending the Database*, which is available from www.apress.com/9781484222645.

On the Feature Selection page of the wizard, which is shown in Figure 4-7, you will select the features of the SQL Server 2016 product suite that you wish to install. For the purpose of this book, we will choose to install the Database Engine and SQL Server Integration Services (SSIS).

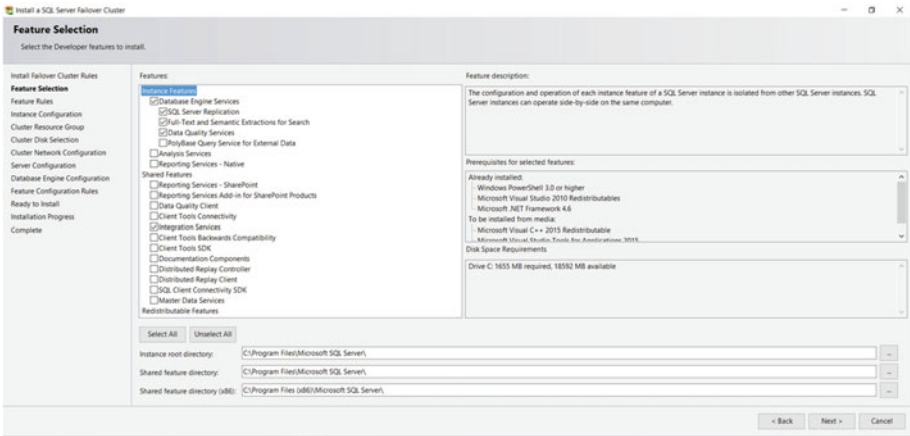


Figure 4-7. *Install a SQL Server Failover Cluster Wizard—Feature Selection page*

It is worthy of note that SSIS is not cluster-aware, and is not designed to be clustered. There are some circumstances where you may decide to cluster the Integration Services service, and if this is the case, then you can work around the limitation by creating a Cluster Role, with the Generic type, and adding the Integration Services service as a dependency. In most scenarios, however, the most appropriate way to manage SSIS on a clustered instance is simply to install the Integration Services service, as a stand-alone service, on each node of the cluster.

Additionally, the Feature Selection page of the wizard requires you to specify folder locations for the instance root folder and the shared features folder. You may want to move these to a different drive in order to leave the C:\ drive for the operating system. This may be a consideration for space reasons, or just to isolate the SQL Server binaries from other applications.

The instance root directory will typically contain a folder for each instance that you create on the server, and there will be separate folders for the Database Engine, SSAS, and SSRS installations. A folder associated with the Database Engine will be called MSSQL13.[InstanceName], where instance name is either the name of your instance, or MSSQLSERVER for a default instance. The number 13 in the folder name relates to the version of SQL Server, which is 13 for SQL Server 2016.

This folder will contain a subfolder called MSSQL, which in turn will contain folders that will store files associated with your instance, including a folder called Binn, which will contain the application files, application extensions, and XML configurations associated with your instance; a folder called Backup, which will be the default location for backups of databases; and a folder called Data, which will be the default location of the system databases.

The default folders for TempDB, user databases, and backups can be modified later in the installation process, and splitting these databases onto separate volumes often a good practice, but may not be necessary (or even possible) if your data will be located on a SAN, as discussed earlier in this chapter. Other folders will also be created here, including a folder called LOGS, which will be the default location for the files for both the Error Logs and the default Extended Event health trace.

If you are installing SQL Server in a 64-bit environment, you will be asked to enter folders for both 32-bit and 64-bit versions of the shared features directory. This is because some SQL Server components are always installed as 32-bit processes. The 32-bit and 64-bit components cannot share a directory, so for installation to continue, you must specify different folders for each of these options. The Shared Features directory becomes a root level directory for features that are shared by all instances of SQL Server, such as SDKs and management tools.

The rules for installing the features that you have selected, and if all rules pass, then the Instance Configuration page will be displayed, as illustrated in Figure 4-8. On this page of the wizard, you will specify a name for the instance. Because the instance will be clustered, this page will also ask you to specify the network name for the instance. In this scenario, we will install a default instance of SQL Server, meaning that we do not need to specify an instance name. We will assign ALWAYSON-SQL-C as the network name.

Install a SQL Server Failover Cluster

Instance Configuration

Specify the name and instance ID for the instance of SQL Server. Instance ID becomes part of the installation path.

Install Failover Cluster Rules
Feature Selection
Feature Rules
Instance Configuration
Cluster Resource Group
Cluster Disk Selection
Cluster Network Configuration
Server Configuration
Database Engine Configuration
Feature Configuration Rules
Ready to Install
Installation Progress
Complete

Specify a network name for the new SQL Server failover cluster. This will be the name used to identify your failover cluster on the network.

SQL Server Network Name:

☒ Default instance
☐ Named instance:

Instance ID:

SQL Server directory:

Detected SQL Server instances and features on this computer:

Instance	Cluster Network Name	Features	Edition	Version	Instance ID

< Back Next > Cancel

Figure 4-8. *Install a SQL Server Failover Cluster Wizard—Instance Configuration page*

■ **Tip** SQL Server uses the term cluster resource group to describe a cluster role. A cluster role is Microsoft’s newer term and within this chapter, the terms should be treated synonymously.

On the Cluster Resource Group page of the wizard, we have the option of either selecting an existing cluster resource group (which gives the option of a Windows administrator precreating the resource group), or entering the name of a new resource group, which will then be created by setup. In our case, we will specify `ALWAYSON-SQL-C` as a name for a new resource group. This is illustrated in Figure 4-9.

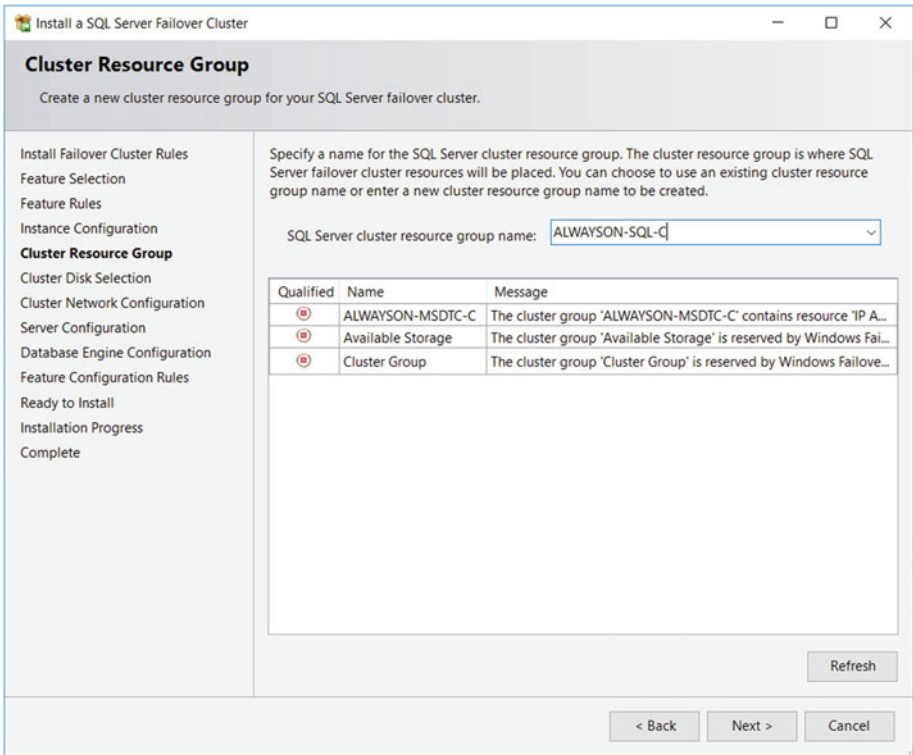


Figure 4-9. *Install a SQL Server Failover Cluster Wizard—Cluster Resource Group page*

■ **Tip** Existing resource groups will be marked with a red or green indicator in the Qualified column. If the marker is red, it means that it is not possible to use that resource group for the SQL Server instance. In this case, the Message column will indicate the reason.

On the Cluster Disk Selection page, illustrated in Figure 4-10, you can select the disk resources that should be associated with the resource group. The page will list all disks that are associated with the cluster and indicate which disks can be selected, with red or green indicators in the Qualified column. Disks that are already associated with other resource groups cannot be selected, because a disk can only be associated with a single resource group. We will specify that both available disks (Data and TempDB) should be associated with the ALWAYSON-SQL-C resource group.

Cluster Disk Selection
Select shared cluster disk resources for your SQL Server failover cluster.

Install Failover Cluster Rules
Feature Selection
Feature Rules
Instance Configuration
Cluster Resource Group
Cluster Disk Selection
Cluster Network Configuration
Server Configuration
Database Engine Configuration
Feature Configuration Rules
Ready to Install
Installation Progress
Complete

Specify the shared disks to be included in the SQL Server resource cluster group. The first drive will be used as the default drive for all databases, but this can be changed on the Database Engine or Analysis Services configuration pages.

☒ Cluster Disk 1
☒ Cluster Disk 4

Available shared disks:

Qualified	Disk	Message
✓	Cluster Disk 1	
✗	Cluster Disk 2	The disk resource 'Cluster Disk 2' cannot be used because it is a cluster q...
✗	Cluster Disk 3	The disk resource 'Cluster Disk 3' is already in use by resource 'MSDTC-AL...
✓	Cluster Disk 4	

Refresh

< Back Next > Cancel

Figure 4-10. Install a SQL Server Failover Cluster Wizard—Cluster Disk Selection page

On the Cluster Network Configuration page (Figure 4-11) we can configure the IP Address of the cluster role. We can either choose DHCP, which means that the IP Address will be attained automatically, or we can specify a static IP Address. In our case, we will specify a static IP Address. If our cluster were to be a stretch cluster (spread across multiple subnets), we would need to specify an IP Address for each subnet.

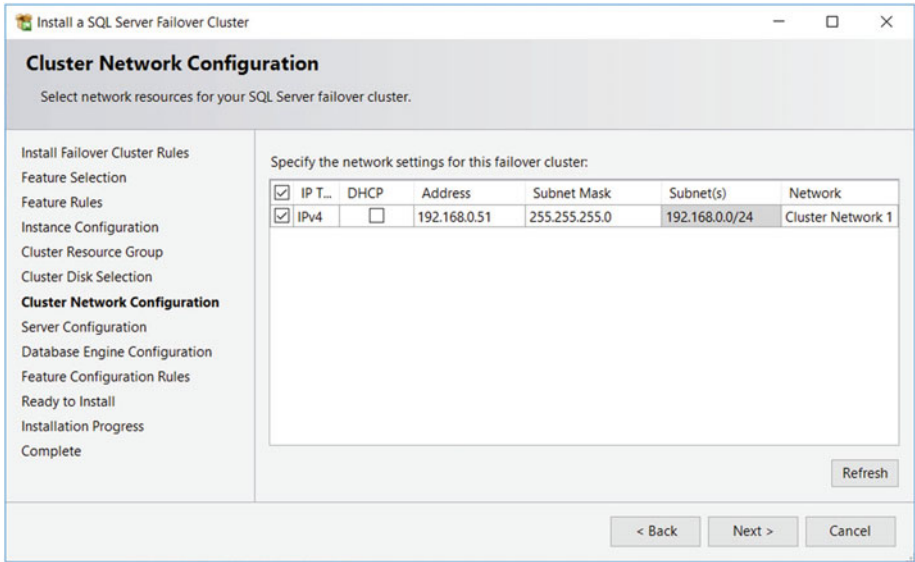


Figure 4-11. *Install a SQL Server Failover Cluster—Cluster Disk Selection page*

The Server Configuration page has two tabs. The first tab is Service Accounts. On this tab, which is illustrated in Figure 4-12, we will specify the service account that will be used as the security context for each SQL Server service, and also specify the startup mode for each of the services. This is worthy of note, because when installing a stand-alone instance, you will usually set each service to start automatically. When installing a cluster, however, then cluster-aware services should be configured to start manually. This is because they will be managed by the cluster service.

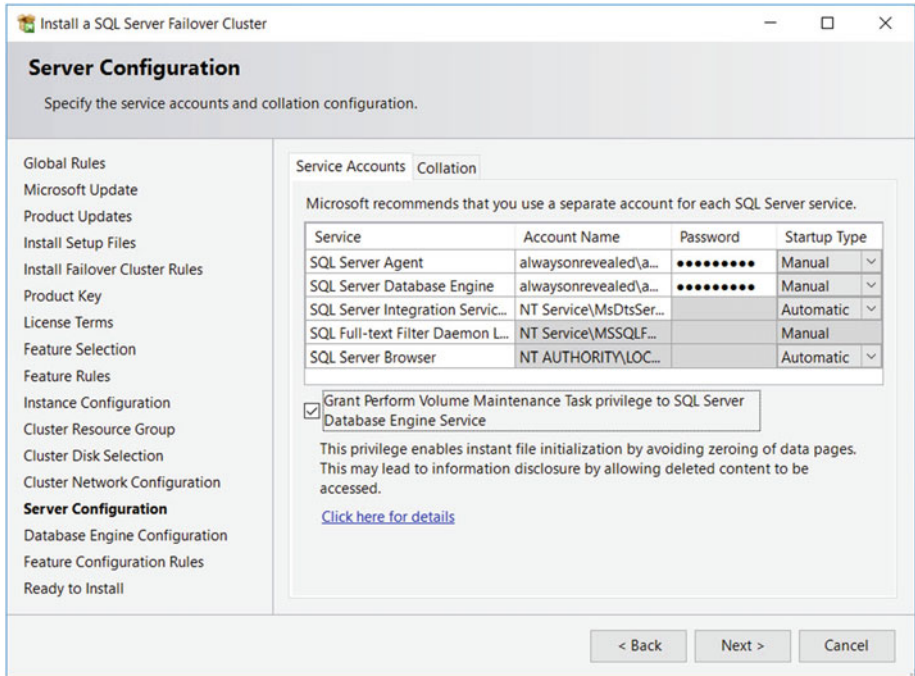


Figure 4-12. *Install a SQL Server Failover Cluster—Service Accounts tab*

A new feature of SQL Server 2016 is to enable Perform Volume Maintenance Tasks during setup, and this is implemented as a simple check box of the Service Account tab. The consideration here is security versus performance. If you choose to perform volume maintenance tasks, then data files will not be zeroed out when they are created or expanded; however, an attacker with specialist software could potentially retrieve data that was previously stored on the allocated disk blocks. Perform Volume Maintenance Tasks does not apply to transaction log files.

■ **Tip** For a full discussion on SQL Server security considerations, I recommend the Apress title *Securing SQL Server: DBAs Defending the Database*, which can be purchased from www.apress.com/9781484222645.

On the Collation tab, shown in Figure 4-13, you can specify the collation that will be configured for the instance. Wherever possible, it is a good idea to use a consistent collation throughout the enterprise, or at a minimum, throughout the instances that make up a data-tier application.

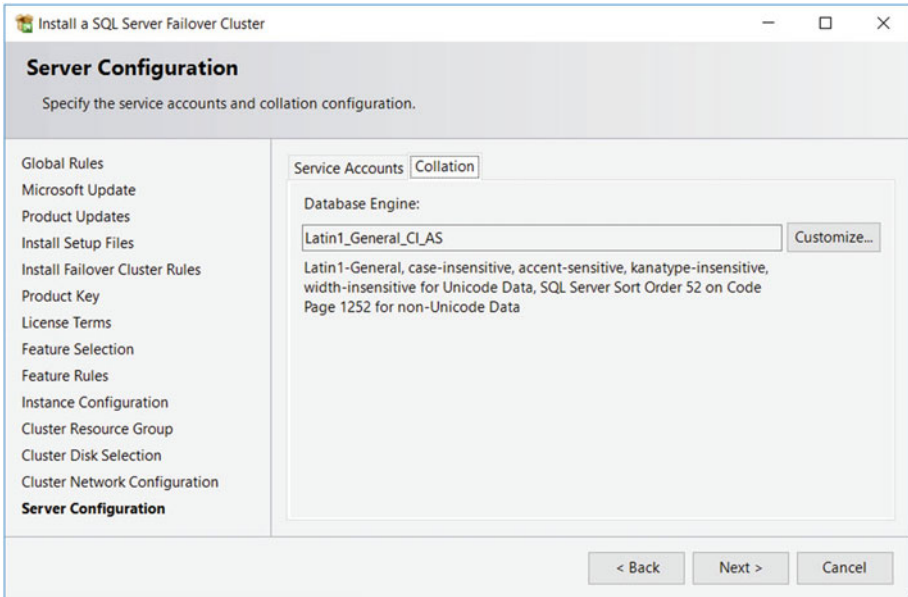


Figure 4-13. *Install a SQL Server Failover Cluster—Collation tab*

The Database Engine Configuration page consists of four tabs. The first of these is the Server Configuration tab, which is illustrated in Figure 4-14. Here, we specify the authentication mode that the instance will use.

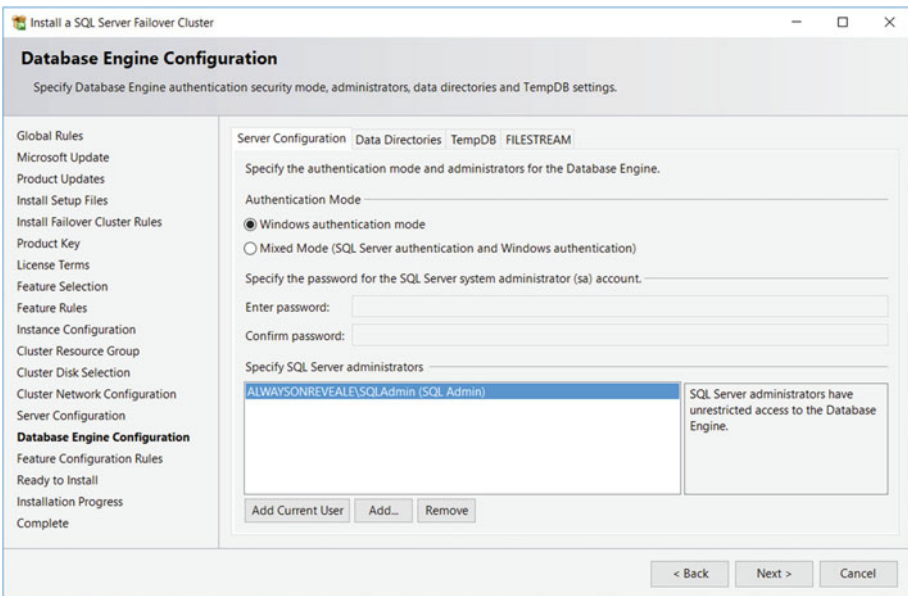


Figure 4-14. *Install a SQL Server Failover Cluster—Server Configuration tab*

Windows Authentication Mode means that the credentials that a user supplies when logging into Windows will be passed to SQL Server, and the user does not require any additional credentials to gain access to the instance. With Mixed Mode, although Windows credentials can still be used to access the instance, users can also be given second-tier credentials. If this option is selected, then SQL Server will hold its own Login names and passwords for Login created inside the instance, and users can supply these in order to gain access, even if their Windows identity does not have permissions.

For security best practice, it is a good idea to only allow Windows authentication to your instance. This is for two reasons. First, with Windows authentication only, if an attacker were to gain access to your network, then they would still not be able to access SQL Server, since they would not have a valid Windows account with the correct permissions. With mixed-mode authentication, however, once inside the network, attackers could use brute force attacks or other hacking methodologies to attempt to gain access via a second-tier SQL Server Login. Secondly, if you specify mixed-mode authentication, then you are required to create an SA account. The SA account is a SQL Server user account that has administrative privileges over the instance. If the password for this account became compromised, then an attacker could gain administrative control over SQL Server. If mixed-mode authentication must be used, it is a good idea to disable the SA account.

Mixed-mode authentication is a necessity in some cases, however. For example, you may have a legacy application that does not support Windows authentication, or a third-party application that has a hardcoded connection that uses second-tier authentication. These would be two valid reasons why mixed mode authentication may be required. Another valid reason would be if you have users that need to access the instance from a nontrusted domain.

Additionally, on the Server Configuration tab, you can specify Windows users that will be added to the sysadmin fixed server role, giving them unrestricted access to the instance. In our scenario, we will add the AlwaysOnRevealed\SQLAdmin user as an instance administrator and use Windows Authentication only.

On the Data Directories tab, illustrated in Figure 4-15, you can alter the default location of the data root directory. On this screen, you can also change the default location for user databases and their log files. Finally, this tab allows you to specify a default location for backups of databases that will be taken. In our scenario, we must ensure that the data root is pointing at the Data volume.

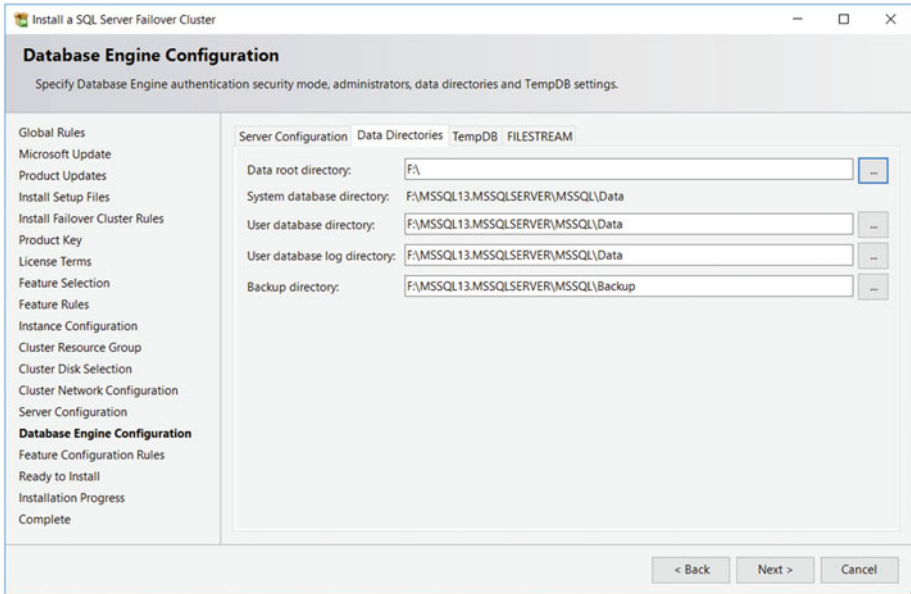


Figure 4-15. *Install a SQL Server Failover Cluster—Data Directories tab*

The TempDB tab is shown in Figure 4-16. This is a new tab, in the SQL Server 2016 installation wizard, and allows you to configure the properties of the TempDB database. This is important as TempDB requires the correct size and processor settings to avoid becoming a bottleneck for the instance. The settings will default to the recommended configuration, which is one file per processor core, to a maximum of eight. This is considered to be the best number of files to avoid contention on system pages, such as GAM, SGAM, and PFS pages. The correct size of TempDB should be estimated through a capacity planning exercise. We will configure TempDB to have an initial size of 60MB per file (240MB in total). We have also configured TempDB to reside on the TempDB volume.

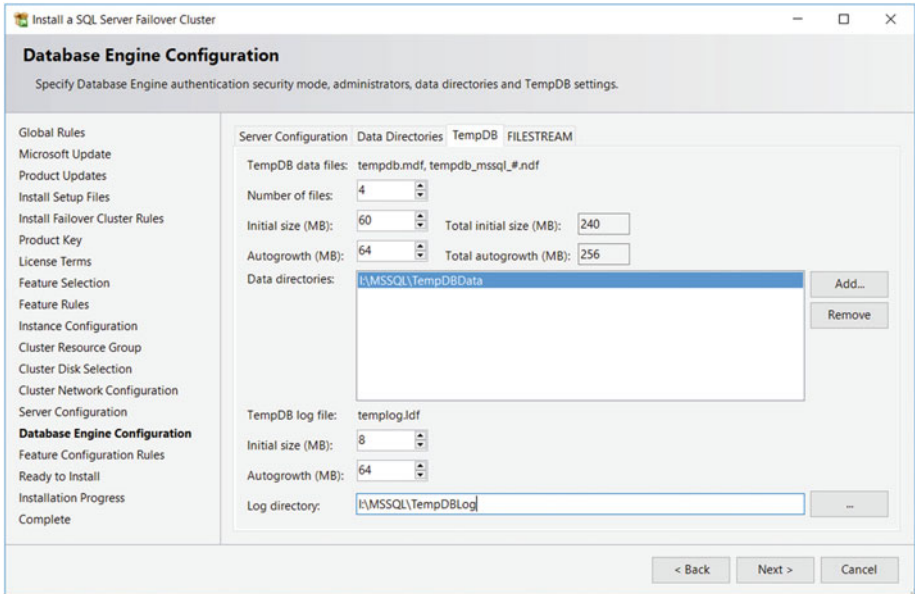


Figure 4-16. *Install a SQL Server Failover Cluster—TempDB tab*

The FILESTREAM tab of the Database Engine Configuration page allows you to enable and configure the level of access for SQL Server FILESTREAM functionality, as illustrated in Figure 4-17. FILESTREAM must also be enabled if you wish to use the FileTable feature of SQL Server. FILESTREAM and FileTable provide the ability to store data in an unstructured manner within the Windows folder structure, while retaining the ability to manage and interrogate this data from SQL Server.

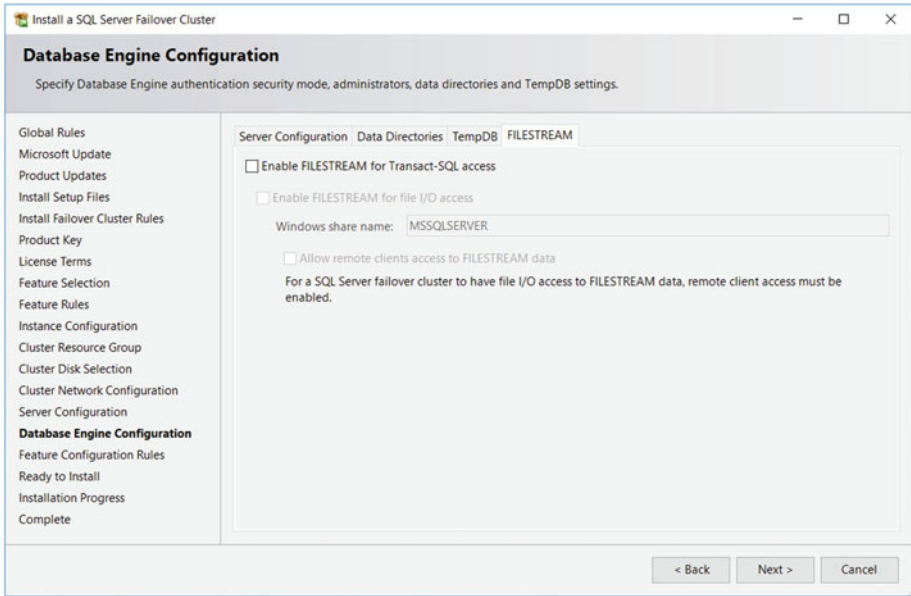


Figure 4-17. *Install a SQL Server Failover Clustered Instance—FILESTREAM tab*

After the Feature Configuration Rules have been checked, the Ready to Install page of the wizard will be displayed. This page of the wizard provides a summary of actions that will be performed by the setup utility. Selecting Install on this page will cause the instance installation to begin. After installation is complete, the Complete page should be reviewed.

Installing the Instance with PowerShell

Of course, we can use PowerShell to install the AlwaysOn failover cluster instance instead of using the GUI. To install an AlwaysOn failover cluster instance from PowerShell, we can use SQL Server’s `setup.exe` application with the `InstallFailoverCluster` action specified.

When you perform a command-line installation of a clustered instance, you need the parameters in Table 4-1, in addition to the parameters that are mandatory when you install a stand-alone instance of SQL Server.

Table 4-1. Required Parameters for the Installation of a Clustered Instance

Parameter	Usage
/FAILOVERCLUSTERIPADDRESSES	Specifies the IP Address(s) to use for the instance in the format <IP Type>;<address>;<network name>;<subnet mask>. For multi-subnet clusters, the IP Addresses are space delimited.
/FAILOVERCLUSTERNETWORKNAME	The virtual name of the clustered instance.
/INSTALLSQLDATADIR	The folder in which to place SQL Server data files. This must be a cluster disk.

The script in Listing 4-1 performs the same installation that has just been demonstrated when you run it from the root directory of the installation media.

Listing 4-1. Installing an AlwaysOn Failover Cluster Instance with PowerShell

```
.\SETUP.EXE /IACCEPTSQLSERVERLICENSETERMS /ACTION="InstallFailoverCluster"
/FEATURES=SQL,IS /INSTANCENAME="MSSQLSERVER"
/SQLSVCACCOUNT="ALWAYSONREVEALED\SQLAdmin" /SQLSVCPASSWORD="Pa$$w0rd"
/AGTSVCACCOUNT="ALWAYSONREVEALED\SQLAdmin" /AGTSVCPASSWORD="Pa$$w0rd"
/SQLSYSADMINACCOUNTS="ALWAYSONREVEALED\SQLAdmin"
/FAILOVERCLUSTERIPADDRESSES="IPv4;192.168.0.51;Cluster Network 2;255.255.255.0"
/FAILOVERCLUSTERNETWORKNAME="ALWAYSON-SQL-C" /INSTALLSQLDATADIR="F:\\" /qs
```

Adding a Node

The next step you should take when installing the cluster is to add the second node. Failure to add the second node results in the instance staying online, but with no high availability, since the second node is unable to take ownership of the role. To configure the second node, you need to log in to the passive cluster node and select the Add Node to SQL Server Failover Cluster option from the Installation tab of SQL Server Installation Center. This invokes the Add a Failover Cluster Node Wizard. The first page of this wizard is the Product Key page. Just like when you install an instance, you need to use this screen to provide the product key for SQL Server. Not specifying a product key only leaves you the option of installing the Evaluation edition, and since this expires after 180 days, it's probably not the wisest choice for high availability.

The following License Terms page of the wizard asks you to read and accept the license terms of SQL Server. Additionally, you need to specify if you wish to participate in Microsoft's Customer Experience Improvement Program. If you select this option, then error reporting is captured and sent to Microsoft.

After you accept the license terms, a rules check runs to ensure that all of the conditions are met so you can continue with the installation. After the wizard checks for Microsoft updates and installing the setup files required for installation, another rules check is carried out to ensure that the rules for adding the node to the cluster are met.

On the Cluster Node Configuration page, illustrated in Figure 4-18, you are asked to confirm the instance name to which you are adding a node. If you have multiple instances on the cluster, then you can use the drop-down box to select the appropriate instance.

Add a Failover Cluster Node

Cluster Node Configuration

Add a node to an existing SQL Server failover cluster.

Product Key
License Terms
Global Rules
Microsoft Update
Install Setup Files
Add Node Rules
Cluster Node Configuration
Cluster Network Configuration
Service Accounts
Feature Rules
Ready to Add Node
Add Node Progress
Complete

SQL Server instance name: MSSQLSERVER

Name of this node: CLUSTERNODE2

Disk Space Requirements: Drive C: 1655 MB required, 19783 MB available

Instance Name	Cluster Network Name	Features	Nodes
MSSQLSERVER	ALWAYSON-SQ...	SQLEngine, SQ...	CLUSTERNODE1

< Back Next > Cancel

Figure 4-18. Add a Failover Cluster Node Wizard—Cluster Node Configuration page

On the Cluster Network Configuration page, shown in Figure 4-19, you confirm the network details. These should be identical to the first node in the cluster, including the same IP Address, since this is, of course, shared between the two nodes.

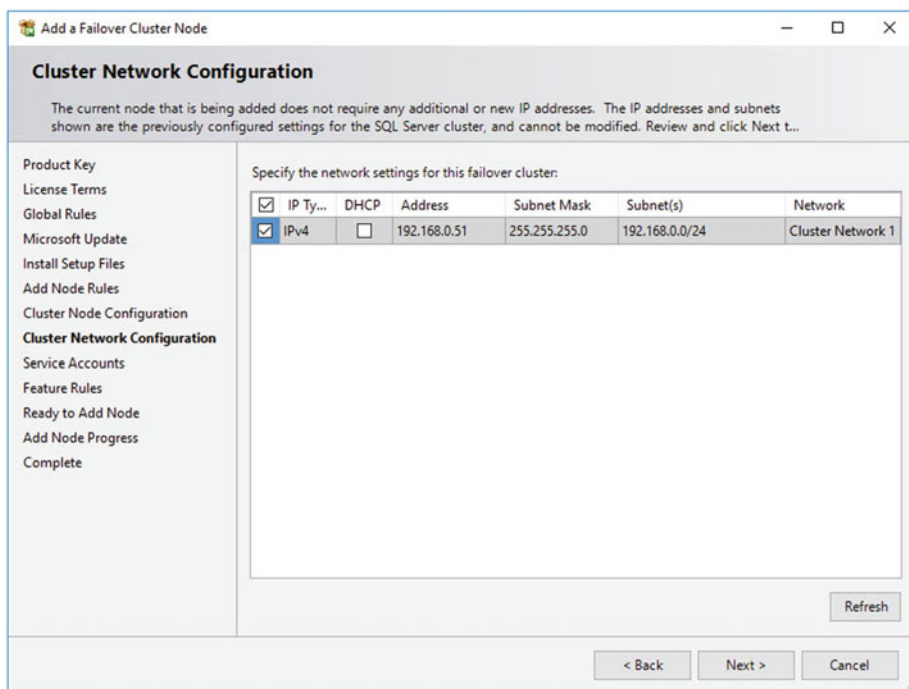


Figure 4-19. The Cluster Network Configuration page

On the Service Accounts page of the wizard, most of the information is in read-only mode and you are not able to modify it. This is because the service accounts you use must be the same for each node of the cluster. You need to re-enter the service account passwords, however. This page is shown in Figure 4-20.

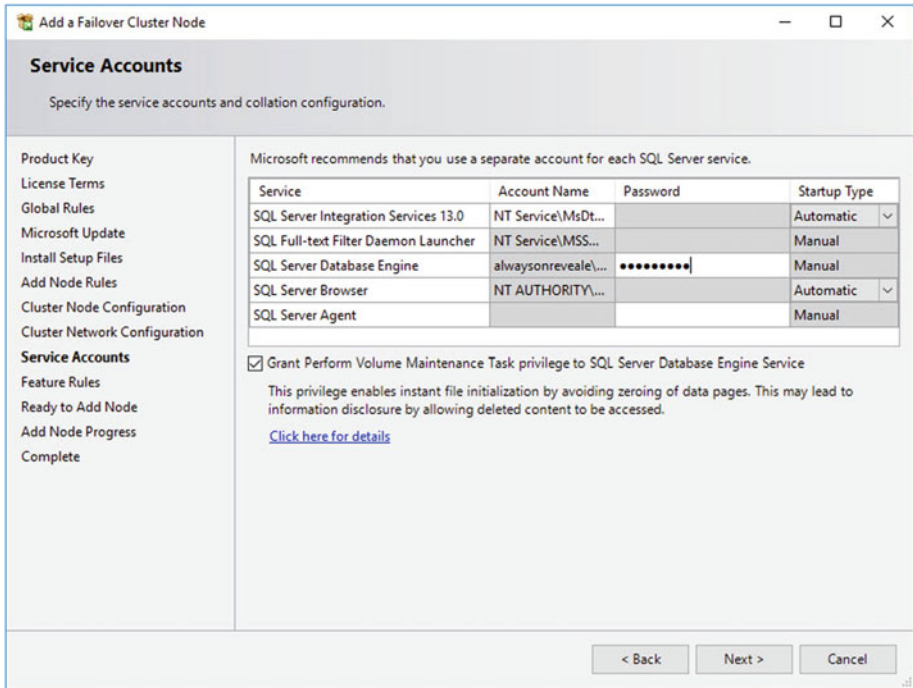


Figure 4-20. The Service Accounts page

Now that the wizard has all of the required information, an additional rules check is carried out before the summary page displays. The summary page is known as the Ready to Add Node page, which provides a summary of the activities that take place during the installation.

Adding a Node Using PowerShell

To add a node using PowerShell instead of the GUI, you can run SQL Server's `setup.exe` application with an `AddNode` action. When you add a node from the command line, the parameters detailed in Table 4-2 are mandatory.

Table 4-2. *Mandatory Parameters for the AddNode Action*

Parameter	Usage
/ACTION	Must be configured as AddNode.
/IACCEPTSQLSERVERLICENSETERMS	Mandatory when installing on Windows Server Core, since the /qs switch must be specified on Windows Server Core.
/INSTANCENAME	The instance that you are adding the extra node to support.
/CONFIRMIPDEPENDENCYCHANGE	Allows multiple IP Addresses to be specified for multi-subnet clusters. Pass in a value of 1 for True or 0 for False.
/FAILOVERCLUSTERIPADDRESSES	Specifies the IP Address(es) to use for the instance in the format <IP Type>;<address>;<network name>;<subnet mask>. For multi-subnet clusters, the IP Addresses are space delimited.
/FAILOVERCLUSTERNETWORKNAME	The virtual name of the clustered instance.
/INSTALLSQLDATADIR	The folder in which to place SQL Server data files. This must be a cluster disk.
/SQLSVCAccount	The service account that is used to run the Database Engine.
/SQLSVCPASSWORD	The password of the service account that is used to run the Database Engine.
/AGTSVCAccount	The service account that issued to run SQL Server Agent.
/AGTSVCPASSWORD	The password of the service account that is used to run SQL Server Agent.

The script in Listing 4-2 adds ClusterNode2 to the role when you run it from the root folder of the install media.

Listing 4-2. Adding a Node Using PowerShell

```
.\setup.exe /IACCEPTSQLSERVERLICENSETERMS /ACTION="AddNode"
/INSTANCENAME="MSSQLSERVER" /SQLSVCAccount="ALWAYSONREVEALED\SQLAdmin"
/SQLSVCPASSWORD="Pa$$w0rd" /AGTSVCAccount="ALWAYSONREVEALED\SQLAdmin"
/AGTSVCPASSWORD="Pa$$w0rd" /FAILOVERCLUSTERIPADDRESSES="IPv4;192.168.0.51;
Cluster Network 2;255.255.255.0" /CONFIRMIPDEPENDENCYCHANGE=0 /qs
```


Summary

An Always On Failover Clustered Instance can be installed using SQL Server Installation Center or via PowerShell. When using the SQL Server Installation Center, the process is very similar to the installation of a stand-alone instance; however, you will need to specify additional details, such as the network name, IP Address, and resource group configuration.

When using PowerShell to install the Instance, you will `InstallFailoverCluster` action, specifying the `/FAILOVERCLUSTERIPADDRESSES`, `/FAILOVERCLUSTERNETWORKNAME`, and `/INSTALLSQLDATADIR` parameters, in addition the parameters required for a stand-alone instance build.



Implementing HA with AlwaysOn Availability Groups

AlwaysOn Availability Groups provide a flexible option for achieving high availability, recovering from disasters, and scaling out read-only workloads. The technology synchronizes data at the database level, but health monitoring and quorum are provided by a Windows cluster.

This chapter demonstrates how to build and configure Availability Groups for both high availability (HA) and disaster recovery (DR). We also discuss aspects such as performance considerations and maintenance, along with using Availability Groups to scale out read-only workloads.

Demonstrations in this chapter make use of the cluster that was built in Chapter 3. Each node has been preconfigured with a stand-alone instance. CLUSTERNODE1 hosts an instance called PRIMARYREPLICA and CLUSTERNODE2 hosts an instance called SYNCHA.

In this chapter, we will create two Availability Groups. The first will be called App1 and contain the App1Customers and App2Sales databases. The second will be called App2 and contain the App2Customers database. Therefore, the tasks that we will perform in this chapter are as follows:

- Create the App1Customers, App1Sales, and App2Customers databases
- Configure the stand-alone instances to support Availability Groups
- Use the New Availability Groups wizard to create the App1 Availability Group and Listener
- Use the New Availability Group dialog box to create the App2 Availability Group
- Use the New Listener dialog box to create the App2 Listener

Preparing for Availability Groups

Before implementing AlwaysOn Availability Groups, we first create three databases, which we will use during the demonstrations in this chapter. Two of the databases relate to the fictional application App1, and the third database relates to the fictional application App2. Each contains a single table, which we populate with data. Each database is configured with Recovery mode set to FULL. This is a hard requirement for a database to use AlwaysOn Availability Groups because data is synchronized via a log stream. The script in Listing 5-1 creates these databases.

Listing 5-1. Creating Databases

```
CREATE DATABASE App1Customers ;
GO

ALTER DATABASE App1Customers SET RECOVERY FULL ;
GO

USE App1Customers
GO

CREATE TABLE App1Customers
(
  ID                INT                PRIMARY KEY          IDENTITY,
  FirstName          NVARCHAR(30),
  LastName           NVARCHAR(30),
  CreditCardNumber  VARBINARY(8000)
) ;
GO

--Populate the table

DECLARE @Numbers TABLE
(
    Number          INT
)

;WITH CTE(Number)
AS
(
    SELECT 1 Number
    UNION ALL
    SELECT Number + 1
    FROM CTE
    WHERE Number < 100
)
INSERT INTO @Numbers
SELECT Number FROM CTE
```

```

DECLARE @Names TABLE
(
    FirstName    VARCHAR(30),
    LastName     VARCHAR(30)
) ;

INSERT INTO @Names
VALUES('Peter', 'Carter'),
      ('Michael', 'Smith'),
      ('Danielle', 'Mead'),
      ('Reuben', 'Roberts'),
      ('Iris', 'Jones'),
      ('Sylvia', 'Davies'),
      ('Finola', 'Wright'),
      ('Edward', 'James'),
      ('Marie', 'Andrews'),
      ('Jennifer', 'Abraham'),
      ('Margaret', 'Jones')

INSERT INTO App1Customers(FirstName, LastName, CreditCardNumber)
SELECT  FirstName, LastName, CreditCardNumber FROM
        (SELECT
            (SELECT TOP 1 FirstName FROM @Names ORDER BY NEWID()) FirstName
          , (SELECT TOP 1 LastName FROM @Names ORDER BY NEWID()) LastName
          , (SELECT CONVERT(VARBINARY(8000)
            (SELECT TOP 1 CAST(Number * 100 AS CHAR(4))
              FROM @Numbers
              WHERE Number BETWEEN 10 AND 99 ORDER BY NEWID()) + '-' +
              (SELECT TOP 1 CAST(Number * 100 AS CHAR(4))
                FROM @Numbers
                WHERE Number BETWEEN 10 AND 99 ORDER BY NEWID())
              + '-' +
              (SELECT TOP 1 CAST(Number * 100 AS CHAR(4))
                FROM @Numbers
                WHERE Number BETWEEN 10 AND 99 ORDER BY NEWID())
              + '-' +
              (SELECT TOP 1 CAST(Number * 100 AS CHAR(4))
                FROM @Numbers
                WHERE Number BETWEEN 10 AND 99 ORDER BY NEWID()))))
          CreditCardNumber
        FROM @Numbers a
        CROSS JOIN @Numbers b
        CROSS JOIN @Numbers c
        ) d ;

CREATE DATABASE App1Sales ;
GO

```

```
ALTER DATABASE App1Sales SET RECOVERY FULL ;
GO
```

```
USE App1Sales
GO
```

```
CREATE TABLE dbo.Orders(
    OrderNumber          int          NOT NULL IDENTITY(1,1) PRIMARY KEY
                           CLUSTERED,
    OrderDate            date         NOT NULL,
    CustomerID           int          NOT NULL,
    ProductID            int          NOT NULL,
    Quantity             int          NOT NULL,
    NetAmount            money        NOT NULL,
    TaxAmount            money        NOT NULL,
    InvoiceAddressID      int          NOT NULL,
    DeliveryAddressID     int          NOT NULL,
    DeliveryDate          date         NULL,
) ;
```

```
DECLARE @Numbers TABLE
(
    Number              INT
)
```

```
;WITH CTE(Number)
AS
(
    SELECT 1 Number
    UNION ALL
    SELECT Number + 1
    FROM CTE
    WHERE Number < 100
)
```

```
INSERT INTO @Numbers
SELECT Number FROM CTE
```

```
--Populate ExistingOrders with data
```

```
INSERT INTO Orders
SELECT
    (SELECT CAST(DATEADD(dd,(SELECT TOP 1 Number
                           FROM @Numbers
                           ORDER BY NEWID()),getdate())as DATE)),
    (SELECT TOP 1 Number -10 FROM @Numbers ORDER BY NEWID()),
    (SELECT TOP 1 Number FROM @Numbers ORDER BY NEWID()),
    (SELECT TOP 1 Number FROM @Numbers ORDER BY NEWID()),
```

```

500,
100,
(SELECT TOP 1 Number FROM @Numbers ORDER BY NEWID()),
(SELECT TOP 1 Number FROM @Numbers ORDER BY NEWID()),
(SELECT CAST DATEADD(dd,(SELECT TOP 1 Number - 10
FROM @Numbers
ORDER BY NEWID()),getdate()) as DATE))
FROM @Numbers a
CROSS JOIN @Numbers b
CROSS JOIN @Numbers c ;

CREATE DATABASE App2Customers ;
GO

ALTER DATABASE App2Customers SET RECOVERY FULL ;
GO

USE App2Customers
GO

CREATE TABLE App2Customers
(
ID INT PRIMARY KEY IDENTITY,
FirstName NVARCHAR(30),
LastName NVARCHAR(30),
CreditCardNumber VARBINARY(8000)
) ;
GO

--Populate the table

DECLARE @Numbers TABLE
(
Number INT
) ;

;WITH CTE(Number)
AS
(
SELECT 1 Number
UNION ALL
SELECT Number + 1
FROM CTE
WHERE Number < 100
)
INSERT INTO @Numbers
SELECT Number FROM CTE ;

```

```

DECLARE @Names TABLE
(
    FirstName      VARCHAR(30),
    LastName       VARCHAR(30)
) ;

INSERT INTO @Names
VALUES('Peter', 'Carter'),
      ('Michael', 'Smith'),
      ('Danielle', 'Mead'),
      ('Reuben', 'Roberts'),
      ('Iris', 'Jones'),
      ('Sylvia', 'Davies'),
      ('Finola', 'Wright'),
      ('Edward', 'James'),
      ('Marie', 'Andrews'),
      ('Jennifer', 'Abraham'),
      ('Margaret', 'Jones')

INSERT INTO App2Customers(Firstname, LastName, CreditCardNumber)
SELECT FirstName, LastName, CreditCardNumber FROM
    (SELECT
        (SELECT TOP 1 FirstName FROM @Names ORDER BY NEWID()) FirstName
      ,(SELECT TOP 1 LastName FROM @Names ORDER BY NEWID()) LastName
      ,(SELECT CONVERT(VARBINARY(8000)
      ,(SELECT TOP 1 CAST(Number * 100 AS CHAR(4))
      FROM @Numbers
      WHERE Number BETWEEN 10 AND 99 ORDER BY NEWID()) + '-' +
        (SELECT TOP 1 CAST(Number * 100 AS CHAR(4))
        FROM @Numbers
        WHERE Number BETWEEN 10 AND 99 ORDER BY NEWID())
        + '-' +
        (SELECT TOP 1 CAST(Number * 100 AS CHAR(4))
        FROM @Numbers
        WHERE Number BETWEEN 10 AND 99 ORDER BY NEWID())
        + '-' +
        (SELECT TOP 1 CAST(Number * 100 AS CHAR(4))
        FROM @Numbers
        WHERE Number BETWEEN 10 AND 99 ORDER BY NEWID()))))
    CreditCardNumber
FROM @Numbers a
CROSS JOIN @Numbers b
CROSS JOIN @Numbers c
) d ;

```

Configuring SQL Server

The first step in configuring AlwaysOn Availability Groups is enabling this feature on the SQL Server service. To enable the feature from the GUI, we open SQL Server Configuration Manager, drill through SQL Server Services, and select Properties from the context menu of the SQL Server service. When we do this, the service properties display and we navigate to the AlwaysOn High Availability tab, shown in Figure 5-1.

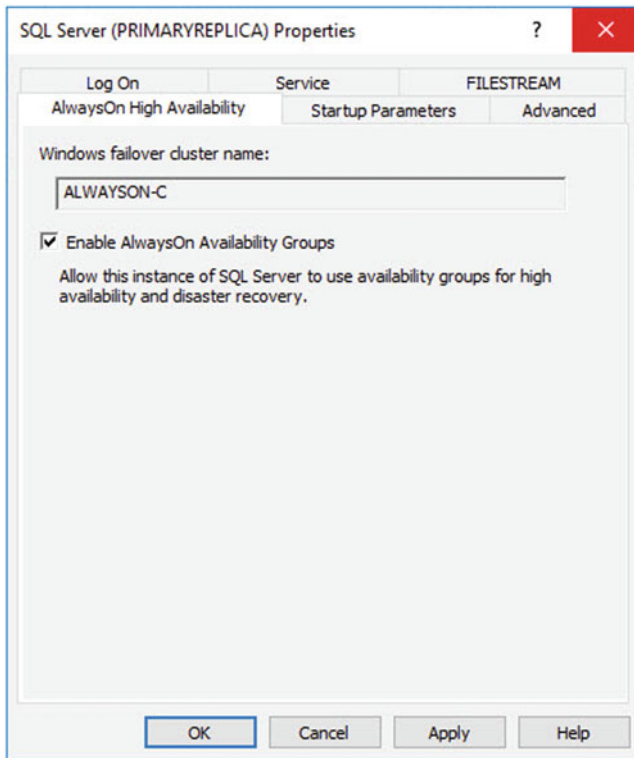


Figure 5-1. The AlwaysOn High Availability tab

On this tab, we check the Enable AlwaysOn Availability Groups box and ensure that the cluster name displayed in the Windows Failover Cluster Name box is correct. We then need to restart the SQL Server service. Because AlwaysOn Availability Groups uses stand-alone instances, which are installed locally on each cluster node, as opposed to a failover clustered instance, which spans multiple nodes, we need to repeat these steps for each stand-alone instance hosted on the cluster.

We can also use PowerShell to enable AlwaysOn Availability Groups. To do this, we use the PowerShell command in Listing 5-2. The script assumes that CLUSTERNODE1 is the name of the server and that PRIMARYREPLICA is the name of the SQL Server instance.

Listing 5-2. Enabling AlwaysOn Availability Groups

```
Enable-SqlAlwaysOn -Path SQLSERVER:\SQL\CLUSTERNODE1\PRIMARYREPLICA
```

The next step is to take a full backup of all databases that will be part of the Availability Group. We create separate Availability Groups for App1 and App2, respectively, so to create an Availability Group for App1, we need to back up the App1Customers and App1Sales databases. We do this by running the script in Listing 5-3.

Listing 5-3. Backing Up the Databases

```
BACKUP DATABASE App1Customers
TO DISK = N'C:\Backups\App1Customers.bak'
WITH NAME = N'App1Customers-Full Database Backup' ;
GO
```

```
BACKUP DATABASE App1Sales
TO DISK = N'C:\Backups\App1Sales.bak'
WITH NAME = N'App1Sales-Full Database Backup' ;
GO
```

Creating the Availability Group

You can create an Availability Group topology in SQL Server in several ways. It can be created manually, predominantly through dialog boxes, via T-SQL, or through a wizard. The following sections explore each of these options.

Using the New Availability Group Wizard

When the backups complete successfully, we invoke the New Availability Group wizard by drilling through AlwaysOn High Availability in Object Explorer and selecting the New Availability Group wizard from the context menu of the Availability Groups folder. The Introduction page of the wizard, displayed in Figure 5-2, now displays, giving us an overview of the steps that we need to undertake.

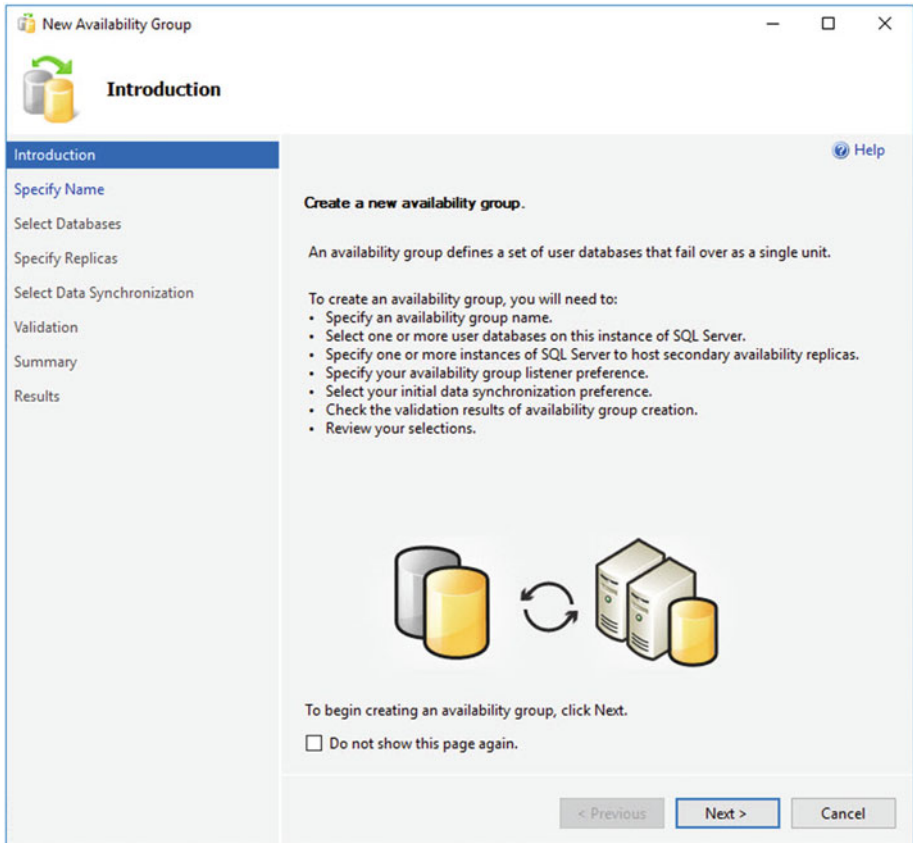


Figure 5-2. The Introduction page

On the Specify Name page (see Figure 5-3), we are prompted to enter a name for our Availability Group. We also need to specify if Database Level Health Detection will be used for the Availability Group. This is a new feature of SQL Server 2016, and if used, the Availability Group will fail over, if any of the availability databases within the group transition out of the ONLINE state.

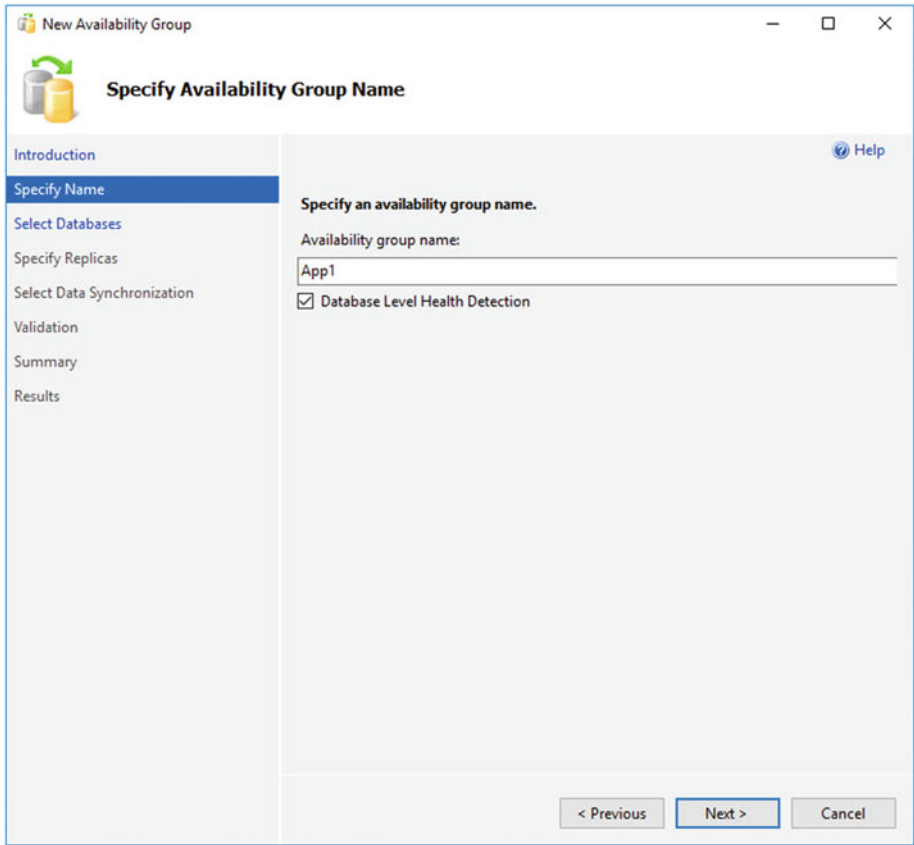


Figure 5-3. The Specify Name page

On the Select Databases page, we are prompted to select the database(s) that we wish to participate in the Availability Group, as illustrated in Figure 5-4. On this screen, notice that we cannot select the App2Customers database, because we have not yet taken a full backup of the database.

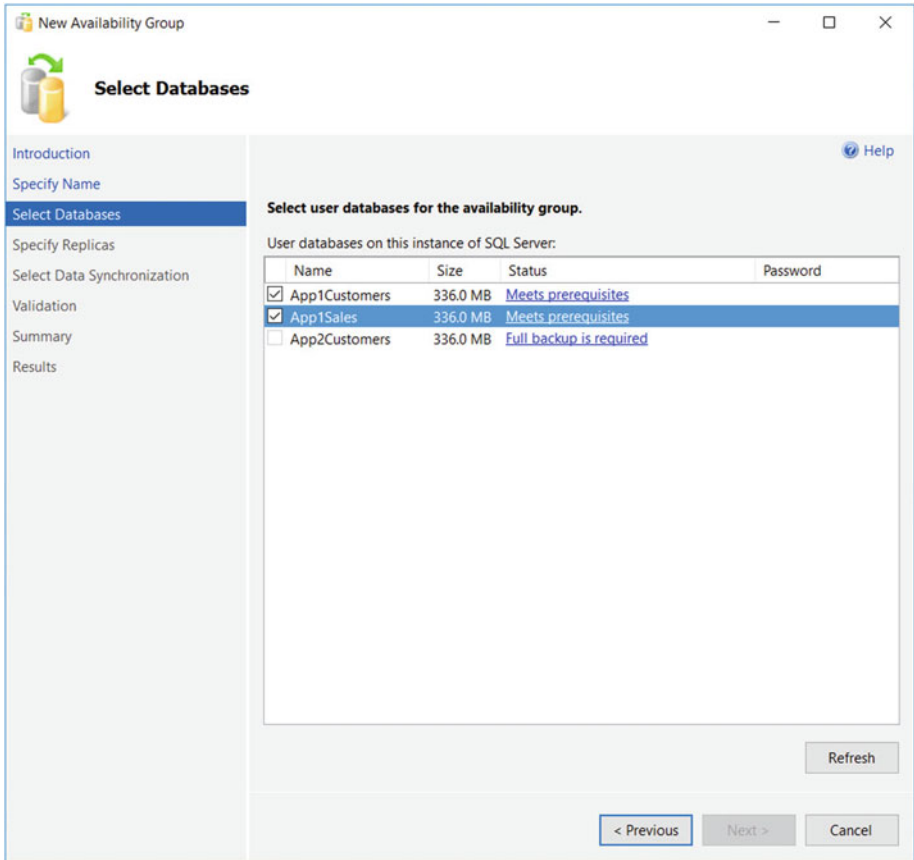


Figure 5-4. The Select Database page

The Specify Replicas page consists of four tabs. We use the first tab, Replicas, to add the secondary replicas to the topology. Checking the Synchronous Commit option causes data to be committed on the secondary replica before it is committed on the primary replica. (This is also referred to as *hardening the log* on the secondary before the primary.) This means that, in the event of a failover, data loss is not possible, meaning that we can meet an SLA (service level agreement) with an RPO (recovery point objective) of 0 (zero). It also means that there is a performance impediment, however. If we choose not to check the option for Synchronous Commit, then the replica operates in Asynchronous Commit mode. This means that data is committed on the primary replica before being committed on the secondary replica. This stops us from suffering a performance impediment, but it also means that, in the event of failover, the RPO is nondeterministic. Performance considerations for synchronous replicas are discussed later in this chapter.

Tip SQL Server 2016 Enterprise Edition allows three replicas to be configured for automatic failover. Previous versions of SQL Server only allowed two synchronous replicas.

When we check the Automatic Failover option, the Synchronous Commit option is also selected automatically if we have not already selected it. This is because automatic failover is only possible in Synchronous Commit mode. We can set the Readable Secondary drop-down to No, Yes, or Read-intent. When we set it to No, the database is not accessible on replicas that are in a secondary role. When we set it to read-intent, the Availability Group Listener is able to redirect read-only workloads to this secondary replica, but only if the application has specified `Application Intent=Read-only` in the connection string. Setting it to Yes enables the listener to redirect read-only traffic, regardless of whether the `Application Intent` parameter is present in the application's connection string. Although we can change the value of Readable Secondary through the GUI while at the same time configuring a replica for automatic failover without error, this is simply a quirk of the wizard. In fact, the replica is not accessible, since active secondaries are not supported when configured for automatic failover. The Replicas tab is illustrated in Figure 5-5.

Specify Replicas

Introduction
Specify Name
Select Databases
Specify Replicas
Select Data Synchronization
Validation
Summary
Results

Help

Specify an instance of SQL Server to host a secondary replica.

Replicas | Endpoints | Backup Preferences | Listener

Availability Replicas:

Server Instance	Initial Role	Automatic Failover (Up to 3)	Synchronous Commit (Up to 3)	Readable Secondary
CLUSTERNODE1\PRIMARYREPLICA	Primary	<input type="checkbox"/>	<input checked="" type="checkbox"/>	No
CLUSTERNODE2\SYNCHA	Secondary	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	No

Add Replica... Add Azure Replica... Remove Replica

Summary for the replica hosted by CLUSTERNODE2\SYNCHA

Replica mode: Synchronous commit with automatic failover
This replica will use synchronous-commit availability mode and support both automatic failover and manual failover.

Readable secondary: No
In the secondary role, this availability replica will not allow any connections.

< Previous Next > Cancel

Figure 5-5. The Replicas tab

Note Using secondary replicas for read-only workloads is discussed in more depth in chapter 6.

On the Endpoints tab of the Specify Replicas page, illustrated in Figure 5-6, we specify the port number for each endpoint. The default port is 5022, but we can specify a different port if we need to. On this tab, we also specify if data should be encrypted when it is sent between the endpoints. It is usually a good idea to check this option, and if we do, then AES (Advanced Encryption Standard) is used as the encryption algorithm.

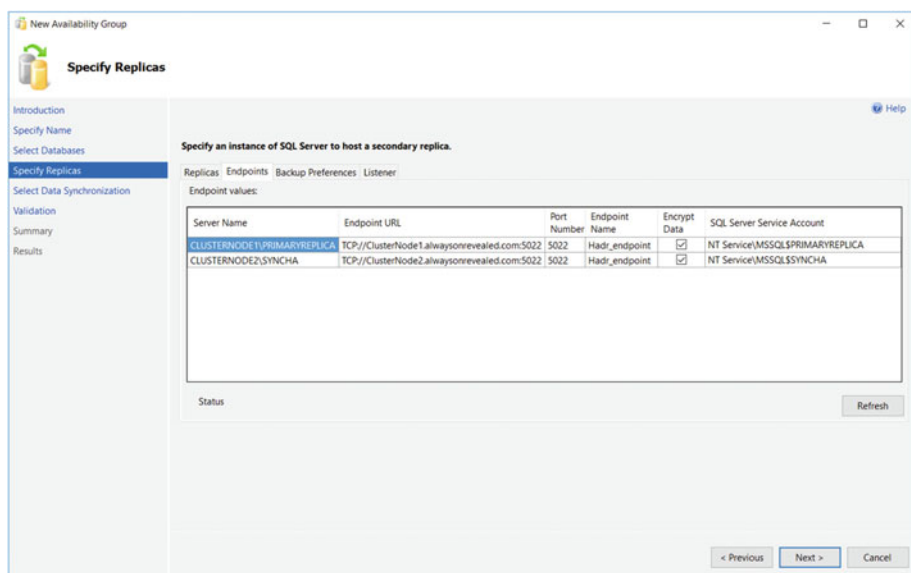


Figure 5-6. The Endpoints tab

Optionally, you can also change the name of the endpoint that is created. Because only one database mirroring endpoint is allowed per instance, however, and because the default name is fairly descriptive, there is not always a reason to change it. Some DBAs choose to rename it to include the name of the instance, since this can simplify the management of multiple servers. This is a good idea if your enterprise has many Availability Group clusters.

The service account each instance uses is displayed for informational purposes. It simplifies security administration if you ensure that the same service account is used by both instances. If you fail to do this, you will need to grant each instance permissions to each service account. This means that instead of reducing the security footprint of each service account by using it for one instance only, you simply push the footprint up to the SQL Server level instead of the Operating System level.

The endpoint URL specifies the URL of the endpoint that Availability Groups will use to communicate. The format of the URL is [Transport Protocol]://[Path]:[Port]. The transport protocol for a database mirroring endpoint is always TCP (Transmission Control Protocol). The path can either be the fully qualified domain name (FQDN) of the server, the server name on its own, or an IP Address, which is unique across the network. I recommend using the FQDN of the server, because this is always guaranteed to work. It is also the default value populated. The port should match the port number that you specify for the endpoint.

Note Availability groups communicate with a database mirroring endpoint. Although database mirroring is deprecated, the endpoints are not.

On the Backup Preferences tab (see Figure 5-7), we can specify the replica on which automated backups will be taken. One of the big advantages of AlwaysOn Availability Groups is that when you use them, you can scale out maintenance tasks, such as backups, to secondary servers. Therefore, automated backups can seamlessly be directed to active secondaries. The possible options are Prefer Secondary, Secondary Only, Primary, or Any Replica. It is also possible to set priorities for each replica. When determining which replica to run the backup job against, SQL Server evaluates the backup priorities of each node and is more likely to choose the replica with the highest priority.

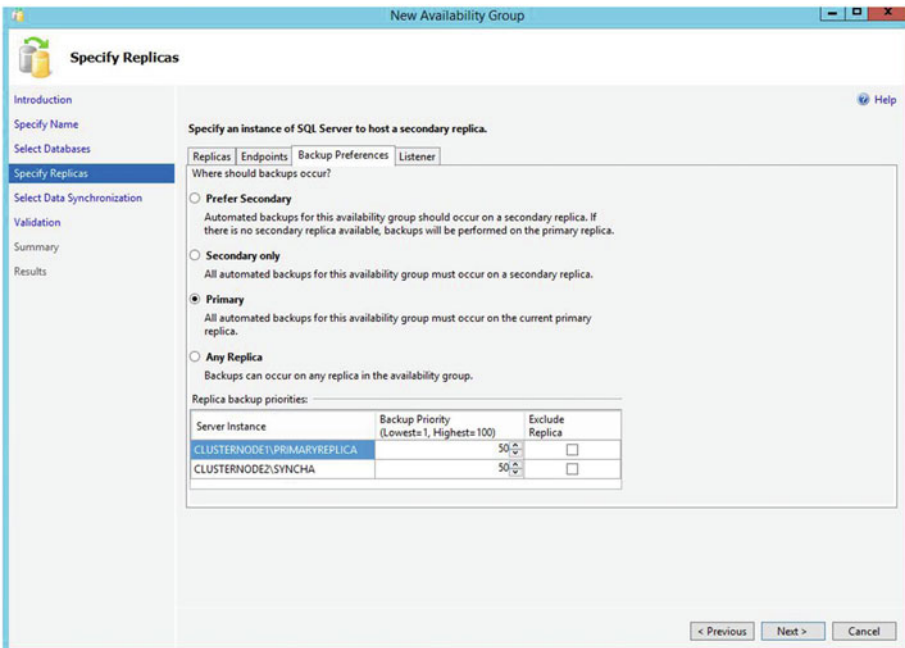


Figure 5-7. The Backup Preferences tab

Although the advantages of reducing IO on the primary replica are obvious, I, somewhat controversially, recommend against scaling automated backups to secondary replicas in many cases. This is especially the case when RTO (recovery time objective) is a priority for the application because of operational supportability issues. Imagine a scenario in which backups are being taken against a secondary replica and a user calls to say that they have accidentally deleted all data from a critical table. You now need to restore a copy of the database and repopulate the table. The backup files, however, sit on the secondary replica. As a result, you need to copy the backup files over to the primary replica before you can begin to restore the database (or perform the restore over the network). This instantly increases your RTO.

Also, when configured to allow backups against multiple servers, SQL Server still only maintains the backup history on the instance where the backup was taken. This means that you may be scrambling between servers, trying to retrieve all of your backup files, not knowing where each one resides. This becomes even worse if one of the servers has a complete system outage. You can find yourself in a scenario in which you have a broken log chain.

The workaround for most of the issues that I just mentioned is to use a share on a file server and configure each instance to back up to the same share. The problem with this, however, is that by setting things up in this manner, you are now sending all of your backups across the network rather than backing them up locally. This can increase the duration of your backups as well as increase network traffic.

On the Listener tab, shown in Figure 5-8, we choose if we want to create an Availability Group Listener or if we want to defer this task until later. If we choose to create the listener, then we need to specify the listener's name, the port that it should listen on, and the IP Address(es) that it should use. We specify one address for each subnet, in multi-subnet clusters. The details provided here are used to create the client access point resource in the Availability Group's cluster role. You may notice that we have specified port 1433 for the listener, although our instance is also running on port 1433. This is a valid configuration, because the listener is configured on a different IP Address than the SQL Server instance. It is also not mandatory to use the same port number, but it can be beneficial, if you are implementing AlwaysOn Availability Groups on an existing instance, because applications that specify the port number to connect may need fewer application changes. Remember that the server name will still be different, however, because applications will be connecting to the virtual name of the listener, as opposed to the name of the physical server\instance. In our example, applications connect to APP1LISTEN\PRIMARYREPLICA instead of CLUSTERNODE1\PRIMARYREPLICA. Although connections via CLUSTERNODE1 are still permitted, they do not benefit from high availability or scale our reporting.

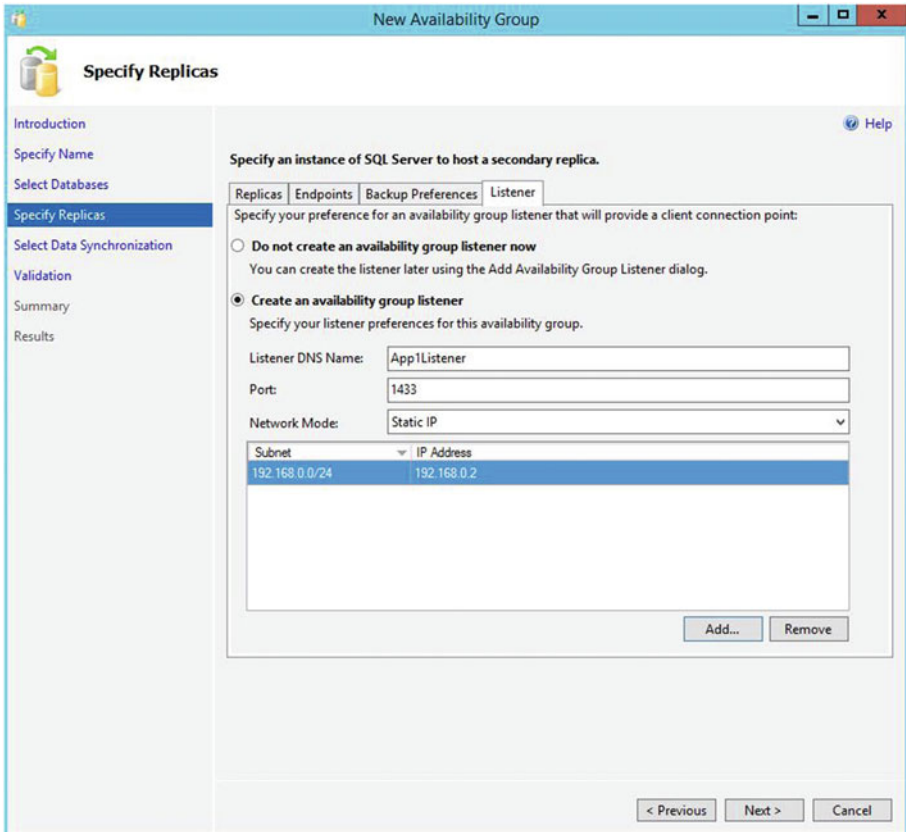


Figure 5-8. The Listener tab

Tip If you do not have Create Computer Objects permission within the OU, then the listener's VCO (virtual computer object) must be prestaged in AD and you must be assigned Full Control permissions on the object.

On the Select Initial Data Synchronization screen, shown in Figure 5-9, we choose how the initial data synchronization of the replicas is performed. If you choose Full, then each database that participates in the Availability Group is subject to a full backup, followed by a log backup. The backup files are backed up to a share, which you specify, before they are restored to the secondary servers. After the restore is complete, data synchronization, via log stream, commences.

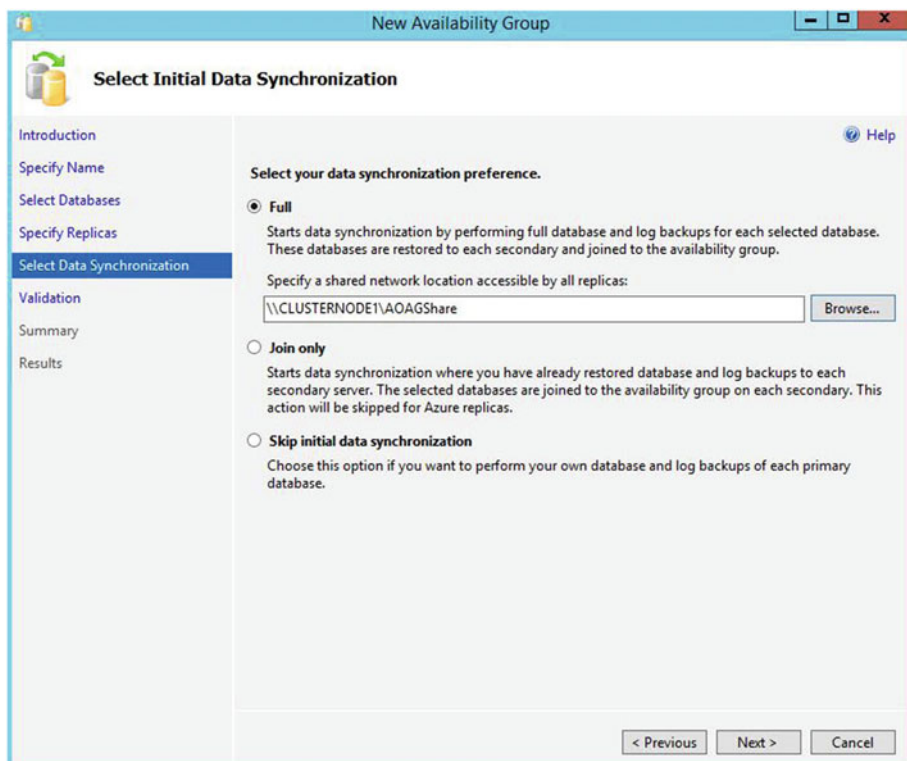


Figure 5-9. The Select Data Synchronization page

If you have already backed up your databases and restored them onto the secondaries, then you can select the Join Only option. This starts the data synchronization, via log stream, on the databases within the Availability Group. Selecting Skip Initial Data Synchronization allows you to back up and restore the databases yourself after you complete the setup.

Tip If your Availability Group will contain many databases, then it may be best to perform the backup/restore yourself. This is because the inbuilt utility will perform the actions sequentially, and therefore it may take a long time to complete.

On the Validation page, rules that may cause the setup to fail are checked, as illustrated in Figure 5-10. If any of the results come back as Failed, then you need to resolve them before you attempt to continue.

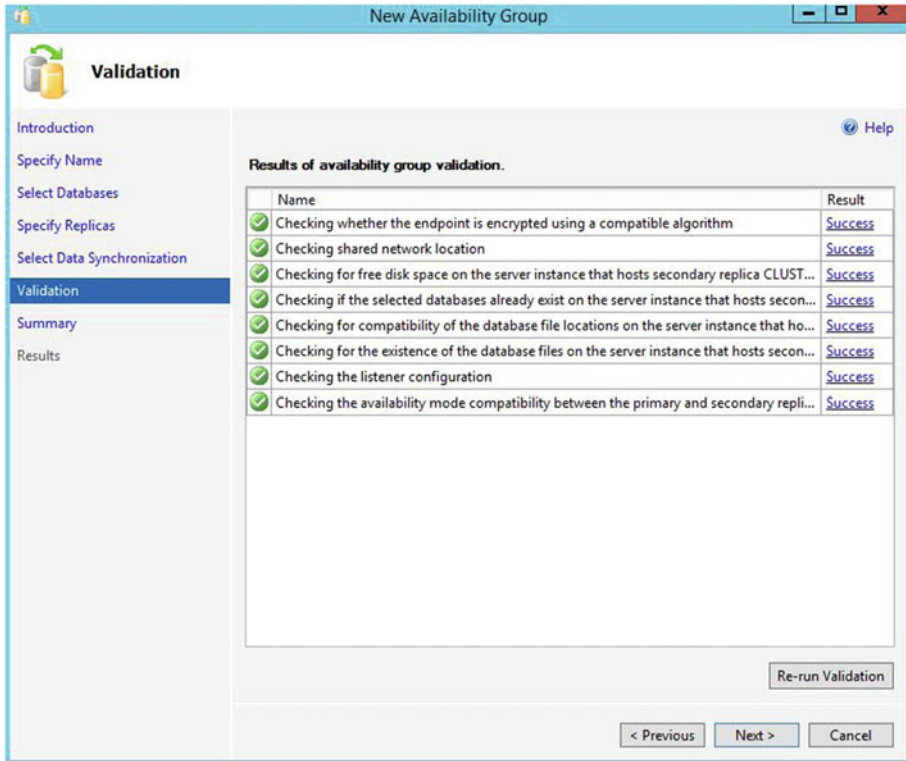


Figure 5-10. The Validation page

Once validation tests are complete and we move to the Summary page, we are presented with a list of the tasks that are to be carried out during the setup. As setup progresses, the results of each configuration task display on the Results page. If any errors occur on this page, be sure to investigate them, but this does not necessarily mean that the entire Availability Group needs to be reconfigured. For example, if the creation of the Availability Group listener fails because the VCO had not been prestaged in AD, then you can re-create the listener without needing to re-create the entire Availability Group.

As an alternative to using the New Availability Group wizard, you can perform the configuration of the Availability Group using the New Availability Group dialog box, followed by the Add Listener dialog box. This method of creating an Availability Group is examined later in this chapter.

Scripting the Availability Group

We can also script the activity by using the script in Listing 5-4. This script connects to both instances within the cluster, meaning that it can only be run in SQLCMD mode. First, the script creates a login for the service account on each instance. It then creates

the TCP endpoint, assigns the connect permission to the service account, and starts the health trace for AlwaysOn Availability Groups (which we discuss later in this chapter). The script then creates the Availability Group on the primary and joins the secondary to the group. Next, we perform a full and log backup and a restore of each database that will participate in the Availability Group before we add the databases to the group. Note that the databases are backed up, restored, and added to the group in a serial manner. If you have many databases, then you may want to parallelize this process.

Listing 5-4. Creating Availability Group

```
--Create Logins for the Service Account,
--create Endpoints and assign Service Account permissions
--to the Endpoint on Primary Replica

:Connect CLUSTERNODE1\PRIMARYREPLICA

USE master
GO

CREATE LOGIN [prosqladmin\clusteradmin] FROM WINDOWS ;
GO

CREATE ENDPOINT [Hadr_endpoint]
    AS TCP (LISTENER_PORT = 5022)
    FOR DATA_MIRRORING (ROLE = ALL, ENCRYPTION = REQUIRED ALGORITHM AES) ;
GO

ALTER ENDPOINT [Hadr_endpoint] STATE = STARTED ;
GO

GRANT CONNECT ON ENDPOINT::[Hadr_endpoint] TO [prosqladmin\clusteradmin] ;
GO

IF EXISTS(SELECT * FROM sys.server_event_sessions WHERE name='AlwaysOn_
health')
BEGIN
    ALTER EVENT SESSION AlwaysOn_health ON SERVER WITH (STARTUP_STATE=ON);
END
IF NOT EXISTS(SELECT * FROM sys.dm_xe_sessions WHERE name='AlwaysOn_health')
BEGIN
    ALTER EVENT SESSION AlwaysOn_health ON SERVER STATE=START;
END
GO

--Create Logins for the Service Account,
--create Endpoints and assign Service Account permissions
--to the Endpoint on Secondary Replica
```

```
:Connect CLUSTERNODE2\SYNCHA
```

```
USE master
GO
```

```
CREATE LOGIN [prosqladmin\ClusterAdmin] FROM WINDOWS ;
GO
```

```
CREATE ENDPOINT [Hadr_endpoint]
    AS TCP (LISTENER_PORT = 5022)
    FOR DATA_MIRRORING (ROLE = ALL, ENCRYPTION = REQUIRED ALGORITHM AES) ;
GO
```

```
ALTER ENDPOINT [Hadr_endpoint] STATE = STARTED ;
GO
```

```
GRANT CONNECT ON ENDPOINT::[Hadr_endpoint] TO [prosqladmin\ClusterAdmin] ;
GO
```

```
IF EXISTS(SELECT * FROM sys.server_event_sessions WHERE name='AlwaysOn_health')
BEGIN
    ALTER EVENT SESSION AlwaysOn_health ON SERVER WITH (STARTUP_STATE=ON);
END
IF NOT EXISTS(SELECT * FROM sys.dm_xe_sessions WHERE name='AlwaysOn_health')
BEGIN
    ALTER EVENT SESSION AlwaysOn_health ON SERVER STATE=START;
END
GO
```

```
--Create Availability Group
```

```
:Connect CLUSTERNODE1\PRIMARYREPLICA
```

```
USE master
GO
```

```
CREATE AVAILABILITY GROUP App1
WITH (AUTOMATED_BACKUP_PREFERENCE = PRIMARY)
FOR DATABASE App1Customers, App1Sales
REPLICA ON N'CLUSTERNODE1\PRIMARYREPLICA'
WITH (ENDPOINT_URL = N'TCP://ClusterNode1.PROSQLADMIN.COM:5022',
    FAILOVER_MODE = AUTOMATIC, AVAILABILITY_MODE = SYNCHRONOUS_COMMIT, BACKUP_
    PRIORITY = 50,
    SECONDARY_ROLE(ALLOW_CONNECTIONS = NO)),
    N'CLUSTERNODE2\SYNCHA'
    WITH (ENDPOINT_URL = N'TCP://ClusterNode2.PROSQLADMIN.COM:5022',
    FAILOVER_MODE = AUTOMATIC, AVAILABILITY_MODE = SYNCHRONOUS_COMMIT,
    BACKUP_PRIORITY = 50, SECONDARY_ROLE(ALLOW_CONNECTIONS = NO));
GO
```

--Create the Listener (Use an IP Address applicable to your environment)

```
ALTER AVAILABILITY GROUP App1
ADD LISTENER N'App1Listen' (
WITH IP
((N'192.168.0.4', N'255.255.255.0')
)
, PORT=1433);
GO
```

--Join the Secondary Replica

```
:Connect CLUSTERNODE2\SYNCHA
```

```
ALTER AVAILABILITY GROUP App1 JOIN;
GO
```

--Back Up Database and Log (First database)

```
:Connect CLUSTERNODE1\PRIMARYREPLICA
```

```
BACKUP DATABASE App1Customers
TO DISK = N'\\CLUSTERNODE1\Backups\App1Customers.bak'
WITH COPY_ONLY, FORMAT, INIT, REWIND, COMPRESSION, STATS = 5 ;
GO
```

```
BACKUP LOG App1Customers
TO DISK = N'\\CLUSTERNODE1\Backups\App1Customers.trn'
WITH NOSKIP, REWIND, COMPRESSION, STATS = 5 ;
GO
```

--Restore Database and Log (First database)

```
:Connect CLUSTERNODE2\SYNCHA
```

```
RESTORE DATABASE App1Customers
FROM DISK = N'\\CLUSTERNODE1\Backups\App1Customers.bak'
WITH NORECOVERY, STATS = 5 ;
GO
```

```
RESTORE LOG App1Customers
FROM DISK = N'\\CLUSTERNODE1\Backups\App1Customers.trn'
WITH NORECOVERY, STATS = 5 ;
GO
```

```

--Wait for replica to start communicating

DECLARE @connection BIT

DECLARE @replica_id UNIQUEIDENTIFIER
DECLARE @group_id UNIQUEIDENTIFIER

SET @connection = 0

WHILE @Connection = 0
BEGIN
    SET @group_id = (SELECT group_id
                     FROM Master.sys.availability_groups
                     WHERE name = N'App1')
    SET @replica_id = (SELECT replica_id
                       FROM Master.sys.availability_replicas
                       WHERE UPPER(replica_server_name COLLATE Latin1_
                                   General_CI_AS) =
                             UPPER(@SERVERNAME COLLATE Latin1_
                                   General_CI_AS)
                       AND group_id = @group_id)

    SET @connection = ISNULL((SELECT connected_state
                              FROM Master.sys.dm_hadr_availability_
                               replica_states
                              WHERE replica_id = @replica_id), 1)

    WAITFOR DELAY '00:00:10'
END

--Add first Database to the Availability Group

ALTER DATABASE App1Customers SET HADR AVAILABILITY GROUP = App1;

GO

--Back Up Database and Log (Second database)

:Connect CLUSTERNODE1\PRIMARYREPLICA

BACKUP DATABASE App1Sales
TO DISK = N'\\CLUSTERNODE1\Backups\App1Sales.bak'
WITH COPY_ONLY, FORMAT, INIT, REWIND, COMPRESSION, STATS = 5 ;
GO

BACKUP LOG App1Sales
TO DISK = N'\\CLUSTERNODE1\Backups\App1Sales.trn'
WITH NOSKIP, REWIND, COMPRESSION, STATS = 5 ;
GO

```

```

--Restore Database and Log (Second database)

:Connect CLUSTERNODE2\SYNCHA

RESTORE DATABASE App1Sales
FROM DISK = N'\\CLUSTERNODE1\Backups\App1Sales.bak'
WITH NORECOVERY, STATS = 5 ;
GO

RESTORE LOG App1Sales
FROM DISK = N'\\CLUSTERNODE1\Backups\App1Sales.trn'
WITH NORECOVERY, STATS = 5 ;
GO

--Wait for replica to start communicating
DECLARE @connection BIT

DECLARE @replica_id UNIQUEIDENTIFIER
DECLARE @group_id UNIQUEIDENTIFIER

SET @connection = 0

WHILE @Connection = 0
BEGIN
    SET @group_id = (SELECT group_id
                     FROM Master.sys.availability_groups
                     WHERE name = N'App1')
    SET @replica_id = (SELECT replica_id
                       FROM Master.sys.availability_replicas
                       WHERE UPPER(replica_server_name COLLATE Latin1_
                                   General_CI_AS) =
                           UPPER(@SERVERNAME COLLATE Latin1_
                                   General_CI_AS)
                       AND group_id = @group_id)

    SET @connection = ISNULL((SELECT connected_state
                              FROM Master.sys.dm_hadr_availability_
                               replica_states
                              WHERE replica_id = @replica_id), 1)

    WAITFOR DELAY '00:00:10'
END

--Add Second database to the Availaility Group

ALTER DATABASE App1Sales SET HADR AVAILABILITY GROUP = App1;
GO

```


Creating the Availability Group via T-SQL gives you the most flexibility in terms of configuration. Table 5-1 contains a complete list of arguments, along with their explanation.

Table 5-1. *The CREATE AVAILABILITY GROUP Arguments*

Argument	Description	Acceptable Values
AUTOMATED_BACKUP_PREFERENCE	Defines where backups run from automated jobs should be taken.	PRIMARY SECONDARY_ONLY SECONDARY NONE
FAILURE_CONDITION_LEVEL	Specifies how sensitive the failover will be. Further details in Table 5-2.	1 through 5
HEALTH_CHECK_TIMEOUT	Configures the amount of time, in milliseconds, that SQL Server has to return health check information to the cluster before the cluster assumes that the instance is not responding, which triggers a failover when FAILOVER_MODE is set to AUTOMATIC.	15000ms through 4294967295ms
DB_FAILOVER	Specifies if an Availability Group should fail over when a database within the Availability Group on the primary replica transitions out of the ONLINE state. Acceptable values are ON and OFF.	
DTC_SUPPORT	Specifies if cross-database transactions are supported. Acceptable values are PER_DB and NONE. This setting can only be configured when creating a new Availability Group. It cannot be set on existing Availability Groups.	
BASIC	Specifies that a basic Availability Group should be supported. This is the only option available for SQL Server Standard Edition and limits the Availability Group to one database and two replicas.	

(continued)

Table 5-1. (continued)

Argument	Description	Acceptable Values
DISTRIBUTED	Specifies that the Availability Group will contain databases which are stretched across multiple Windows Failover Clusters.	
DATABASE	A comma-separated list of databases that will join the Availability Group.	-
REPLICA ON	A comma-separated list of server\instance names that will be replicas within the group. The following arguments in this table form the WITH clause of the REPLICA ON argument.	-
ENDPOINT_URL	The URL of the TCP endpoint that the replica will use to communicate.	-
AVAILABILITY_MODE	Determines if the replica operates in synchronous or asynchronous mode.	SYNCHRONOUS_COMMIT ASYNCHRONOUS_COMMIT
FAILOVER_MODE	When the AVAILABILITY_MODE is set to synchronous, determines if automatic failover should be allowed.	AUTOMATIC MANUAL
BACKUP_PRIORITY	Gives the replica a weight when SQL Server is deciding where an automated backup job should run.	0 through 100
SECONDARY_ROLE	Specifies properties that only apply to the replica when it is in a secondary role. ALLOW_CONNECTIONS specifies if the replica is readable, and if so, by all read_only connections or only those that specify read-intent in the connection string. READ_ONLY_ROUTING_URL specifies the URL for applications to connect to it, for read-only operations, in the following format: TCP://ServerName:Port.	-

(continued)

Table 5-1. (continued)

Argument	Description	Acceptable Values
PRIMARY_ROLE	Specifies properties that only apply to the replica when it is in the primary role. ALLOW_CONNECTIONS can be configured as All to allow any connection, or Read_Write, to disallow read-only connections. READ_ONLY_ROUTING_LIST is a comma-separated list of server\instance names that have been configured as read-only replicas.	-
AVAILABILITY GROUP ON	If the Availability Group is DISTRIBUTED, use the AVAILABILITY GROUP ON argument to specify the two Availability Groups (on different clusters). that will be used	
SESSION_TIMEOUT	Specifies how long replicas can survive without receiving a ping before they enter the DISCONNECTED state.	5 to 2147483647 seconds

The FAILOVER_CONDITION_LEVEL argument determines the group's sensitivity to failover. Table 5-2 provides a description of each of the five levels.

Table 5-2. The FAILOVER_CONDITION_LEVEL Argument

Level	Failover Triggered By
1	Instance down. AOAG lease expires.
2	Conditions of level 1 plus: HEALTH_CHECK_TIMEOUT is exceeded. The replica has a state of FAILED.
3 (Default)	Conditions of level 2 plus: SQL Server experiences critical internal errors.
4	Conditions of level 3 plus: SQL Server experiences moderate internal errors.
5	Failover initialed on any qualifying condition.

Using the New Availability Group Dialog Box

Now that we have successfully created our first Availability Group, let's create a second Availability Group for App2. This time, we use the New Availability Group and Add Listener Dialog boxes. We begin this process by backing up the App2Customers database. Just like when we created the App1 Availability Group, the databases are not selectable until we perform the backup. Unlike when we used the wizard, however, we have no way to make SQL Server perform the initial database synchronization for us. Therefore, we back up the database to the share that we created during the previous demonstration and then restore the backup, along with a transaction log backup, to the secondary instance. We do this by using the script in Listing 5-5, which must be run in SQLCMD mode for it to work. This is because it connects to both instances.

Listing 5-5. Backing Up and Restoring the Database

```
--Back Up Database and Log
```

```
:Connect CLUSTERNODE1\PRIMARYREPLICA
```

```
BACKUP DATABASE App2Customers TO DISK = N'\\CLUSTERNODE1\
Backups\App2Customers.bak' WITH COPY_ONLY, FORMAT, INIT, REWIND,
COMPRESSION, STATS = 5 ;
GO
```

```
BACKUP LOG App2Customers TO DISK = N'\\CLUSTERNODE1\Backups\App2Customers.
trn' WITH NOSKIP, REWIND, COMPRESSION, STATS = 5 ;
GO
```

```
--Restore Database and Log
```

```
:Connect CLUSTERNODE2\SYNCHA
```

```
RESTORE DATABASE App2Customers FROM DISK = N'\\CLUSTERNODE1\Backups\
App2Customers.bak' WITH NORECOVERY, STATS = 5 ;
GO
```

```
RESTORE LOG App2Customers FROM DISK = N'\\CLUSTERNODE1\Backups\
App2Customers.trn' WITH NORECOVERY, STATS = 5 ;
GO
```

If we had not already created an Availability Group, then our next job would be to create a TCP endpoint so the instances could communicate. We would then need to create a login for the service account on each instance and grant it the connect permissions on the endpoints. Because we can only ever have one database mirroring endpoint per instance, however, we are not required to create a new one, and obviously we have no reason to grant the service account additional privileges. Therefore, we continue by creating the Availability Group. To do this, we drill through AlwaysOn High Availability in Object Explorer and select New Availability Group from the context menu of Availability Groups.

This causes the General tab of the New Availability Group dialog box to display, as illustrated in Figure 5-11. On this screen, we type the name of the Availability Group in the first field. Then we click the Add button under the Availability Databases window before we type the name of the database that we wish to add to the group. We then need to click the Add button under the Availability Replicas window before we type the server\instance name of the secondary replica in the new row.

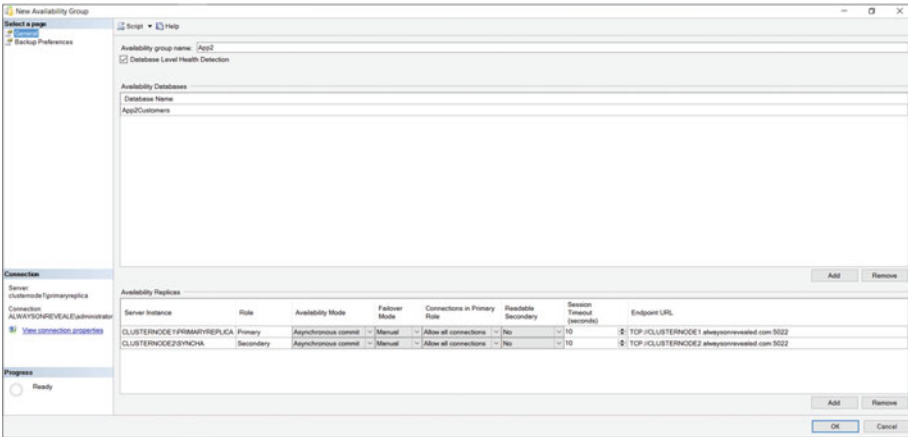


Figure 5-11. The New Availability Group dialog box

Now we can begin to set the replica properties. We discussed the Role, Availability Mode, Failover Mode, Readable Secondary, and Endpoint URL properties when we created the App1 Availability Group. The Connection in Primary Role property defines what connections can be made to the replica if the replica is in the primary role. You can configure this as either Allow All Connections, or Allow Read/Write Connections. When Read/Write is specified, applications using the Application Intent = Read-only parameter in their connection string will not be able to connect to the replica.

The Session Timeout property sets how long the replicas can go without receiving a ping from one another before they enter the DISCONNECTED state and the session ends. Although it is possible to set this value to as low as 5 seconds, it is usually a good idea to keep the setting at or above 10 seconds, otherwise you run the risk of a false positive response, resulting in unnecessary failover. If a replica times out, it needs to be resynchronized, since transactions on the primary will no longer wait for the secondary, even if the secondary is running in Synchronous Commit mode.

Note You may have noticed that we have configured the replica in Asynchronous Commit mode. This is for the benefit of a later demonstration. For HA, we would always configure Synchronous Commit mode, since otherwise, automatic failover is not possible.

On the Backup Preferences tab of the dialog box, we define the preferred replica to use for automated backup jobs, as shown in Figure 5-12. Just like when using the wizard, we can specify Primary, or we can choose between enforcing and preferring backups to occur on a secondary replica. We can also configure a weight, between 0 and 100 for each replica, and use the Exclude Replica check box to avoid backups being taken on a specific node.

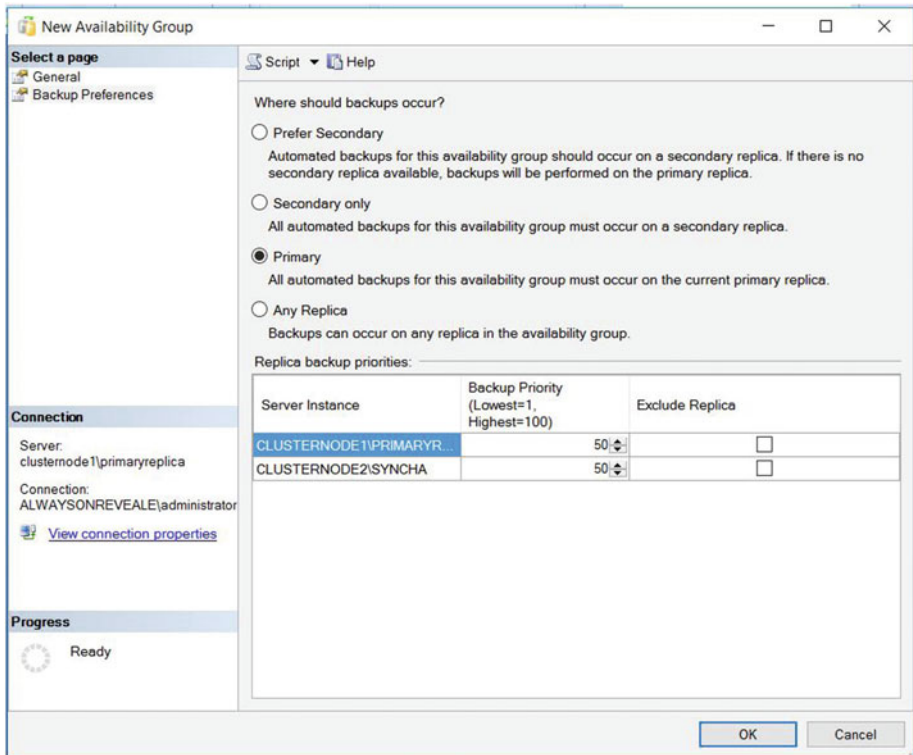


Figure 5-12. The Backup Preferences tab

Once we have created the Availability Group, we need to create the Availability Group Listener. To do this, we select New Listener from the context menu of the App2 Availability Group, which should now be visible in Object Explorer. This invokes the New Availability Group Listener dialog box, which can be seen in Figure 5-13.

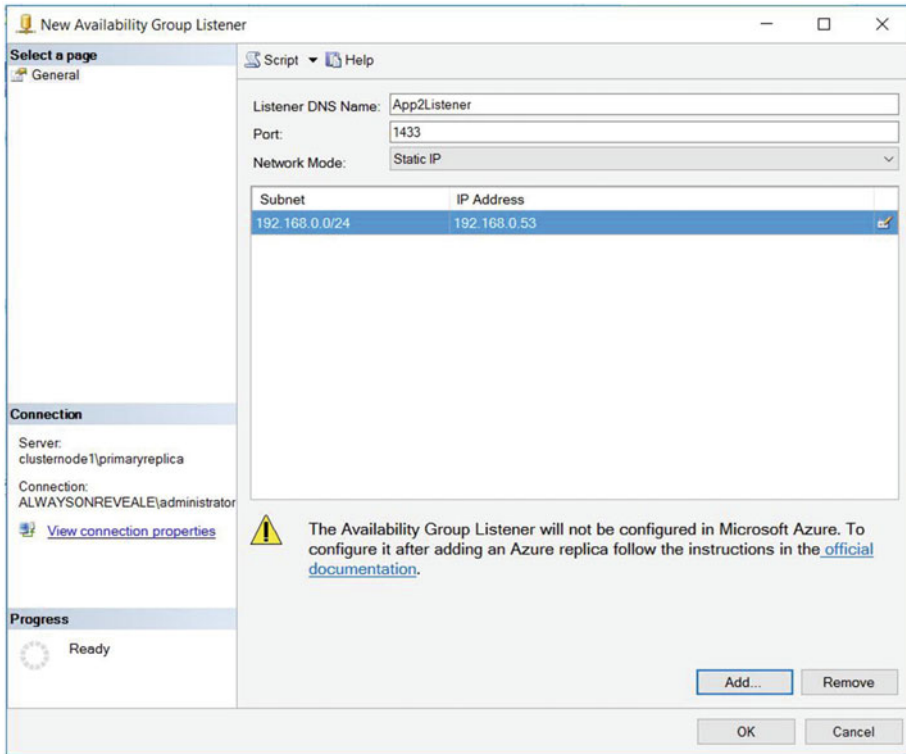


Figure 5-13. The New Availability Group Listener dialog box

In this dialog box, we start by entering the virtual name for the listener. We then define the port that it will listen on and the IP Address that will be assigned to it. We are able to use the same port for both of the listeners, as well as the SQL Server instance, because all three use different IP Addresses.

Performance Considerations for Synchronous Commit Mode

Unlike traditional clustering, Availability Group topology does not have any shared disk resources. Therefore, data must be replicated on two devices, which of course, has an overhead. This overhead varies depending on various aspects of your environment, such as network latency and disk performance, as well as the application profile. However, the script in Listing 5-6 runs some write-intensive tests against the App2Customers database (which is in Asynchronous Commit mode) and then against the App1Customers database (which is Synchronous Commit mode). This indicates the overhead that you can expect to witness.

■ **Tip** It is important to remember that there is no overhead on read performance. Also, despite the overhead associated with writes, some of this is offset by distributing read-only workloads if you implement readable secondary replicas.

Listing 5-6. Performance Benchmark with Availability Groups

```
DBCC FREEPROCCACHE
DBCC DROPCLEANBUFFERS

SET STATISTICS TIME ON

PRINT 'Begin asynchronous commit benchmark'

USE App2Customers
GO

PRINT 'Build a nonclustered index'

CREATE NONCLUSTERED INDEX NIX_FirstName_LastName ON App2Customers(FirstName,
LastName) ;

PRINT 'Delete from table'

DELETE FROM [dbo].[App2Customers] ;

PRINT 'Insert into table'

DECLARE @Numbers TABLE
(
    Number      INT
)

;WITH CTE(Number)
AS
(
    SELECT 1 Number
    UNION ALL
    SELECT Number + 1
    FROM CTE
    WHERE Number < 100
)
INSERT INTO @Numbers
SELECT Number FROM CTE ;
```



```

DECLARE @Names TABLE
(
    FirstName      VARCHAR(30),
    LastName       VARCHAR(30)
) ;

INSERT INTO @Names
VALUES('Peter', 'Carter'),
      ('Michael', 'Smith'),
      ('Danielle', 'Mead'),
      ('Reuben', 'Roberts'),
      ('Iris', 'Jones'),
      ('Sylvia', 'Davies'),
      ('Finola', 'Wright'),
      ('Edward', 'James'),
      ('Marie', 'Andrews'),
      ('Jennifer', 'Abraham'),
      ('Margaret', 'Jones')

INSERT INTO App2Customers(FirstName, LastName, CreditCardNumber)
SELECT  FirstName, LastName, CreditCardNumber FROM
      (SELECT
        (SELECT TOP 1 FirstName FROM @Names ORDER BY NEWID()) FirstName
        ,(SELECT TOP 1 LastName FROM @Names ORDER BY NEWID()) LastName
        ,(SELECT CONVERT(VARBINARY(8000)
          , (SELECT TOP 1 CAST(Number * 100 AS CHAR(4))
            FROM @Numbers
            WHERE Number BETWEEN 10 AND 99 ORDER BY NEWID()) + '-' +
          (SELECT TOP 1 CAST(Number * 100 AS CHAR(4))
            FROM @Numbers
            WHERE Number BETWEEN 10 AND 99 ORDER BY NEWID())
          + '-' +
          (SELECT TOP 1 CAST(Number * 100 AS CHAR(4))
            FROM @Numbers
            WHERE Number BETWEEN 10 AND 99 ORDER BY NEWID())
          + '-' +
          (SELECT TOP 1 CAST(Number * 100 AS CHAR(4))
            FROM @Numbers
            WHERE Number BETWEEN 10 AND 99 ORDER BY NEWID()))))
      CreditCardNumber
FROM @Numbers a
CROSS JOIN @Numbers b
CROSS JOIN @Numbers c
) d ;
GO

```

```

PRINT 'Begin synchronous commit benchmark'

USE App1Customers
GO

PRINT 'Build a nonclustered index'

CREATE NONCLUSTERED INDEX NIX_FirstName_LastName ON App1Customers(FirstName,
LastName) ;

PRINT 'Delete from table'

DELETE FROM dbo.App1Customers ;

PRINT 'Insert into table'

DECLARE @Numbers TABLE
(
    Number          INT
)

;WITH CTE(Number)
AS
(
    SELECT 1 Number
    UNION ALL
    SELECT Number + 1
    FROM CTE
    WHERE Number < 100
)
INSERT INTO @Numbers
SELECT Number FROM CTE ;

DECLARE @Names TABLE
(
    FirstName      VARCHAR(30),
    LastName       VARCHAR(30)
) ;

INSERT INTO @Names
VALUES('Peter', 'Carter'),
      ('Michael', 'Smith'),
      ('Danielle', 'Mead'),
      ('Reuben', 'Roberts'),
      ('Iris', 'Jones'),
      ('Sylvia', 'Davies'),

```

```

        ('Finola', 'Wright'),
        ('Edward', 'James'),
        ('Marie', 'Andrews'),
        ('Jennifer', 'Abraham'),
        ('Margaret', 'Jones') ;

INSERT INTO App1Customers(FirstName, LastName, CreditCardNumber)
SELECT  FirstName, LastName, CreditCardNumber FROM
        (SELECT
            (SELECT TOP 1 FirstName FROM @Names ORDER BY NEWID()) FirstName
        ,(SELECT TOP 1 LastName FROM @Names ORDER BY NEWID()) LastName
        ,(SELECT CONVERT(VARBINARY(8000)
        ,(SELECT TOP 1 CAST(Number * 100 AS CHAR(4))
        FROM @Numbers
        WHERE Number BETWEEN 10 AND 99 ORDER BY NEWID()) + '-' +
        (SELECT TOP 1 CAST(Number * 100 AS CHAR(4))
        FROM @Numbers
        WHERE Number BETWEEN 10 AND 99 ORDER BY NEWID())
        + '-' +
        (SELECT TOP 1 CAST(Number * 100 AS CHAR(4))
        FROM @Numbers
        WHERE Number BETWEEN 10 AND 99 ORDER BY NEWID())
        + '-' +
        (SELECT TOP 1 CAST(Number * 100 AS CHAR(4))
        FROM @Numbers
        WHERE Number BETWEEN 10 AND 99 ORDER BY NEWID()))))
CreditCardNumber
FROM @Numbers a
CROSS JOIN @Numbers b
CROSS JOIN @Numbers c
) d ;

```

GO

SET STATISTICS TIME OFF

GO

The relevant parts of the results of this query are displayed in Listing 5-7. You can see that the index rebuild was three times slower when the Availability Group was operating in Synchronous Commit mode, the insert was nearly twice as slow, and the delete was also marginally slower.

Listing 5-7. SQL Server 2016 Results of Performance Test

Begin asynchronous commit benchmark

Build a nonclustered index

SQL Server Execution Times:

CPU time = 4157 ms, elapsed time = 4948 ms.

Delete from table

SQL Server Execution Times:

CPU time = 12500 ms, elapsed time = 21671 ms.

Insert into table

SQL Server Execution Times:

CPU time = 6454 ms, elapsed time = 8771 ms.

Begin synchronous commit benchmark

Build a nonclustered index

SQL Server Execution Times:

CPU time = 4610 ms, elapsed time = 15709 ms.

Delete from table

SQL Server Execution Times:

CPU time = 11468 ms, elapsed time = 27389 ms.

Insert into table

SQL Server Execution Times:

CPU time = 7562 ms, elapsed time = 14364 ms.

Let's compare this to the same results witnessed using SQL Server 2014, which are shown in Listing 5-8. Here, you can see that the insert was six times slower when using Synchronous Commit mode. This is testament to the performance improvements that Microsoft has made in the newer version.

Listing 5-8. SQL Server 2014 Results of Performance Test

Begin asynchronous commit benchmark

Build a nonclustered index

SQL Server Execution Times:

CPU time = 1109 ms, elapsed time = 4316 ms.

Delete from table

SQL Server Execution Times:

CPU time = 6938 ms, elapsed time = 69652 ms.
(1000000 row(s) affected)

Insert into table

SQL Server Execution Times:

CPU time = 13656 ms, elapsed time = 61372 ms.
(1000000 row(s) affected)

Begin synchronous commit benchmark

Build a nonclustered index

SQL Server Execution Times:

CPU time = 1516 ms, elapsed time = 12437 ms.

Delete from table

SQL Server Execution Times:

CPU time = 8563 ms, elapsed time = 77273 ms.
(1000000 row(s) affected)

Insert into table

SQL Server Execution Times:

CPU time = 23141 ms, elapsed time = 372161 ms.
(1000000 row(s) affected)

Caution The performance tests in this section are based on VMs running on a laptop. The tests are intended to illustrate the performance impediment caused by Synchronous Commit mode, and the performance improvements within SQL Server 2016. It is not intended as an accurate benchmark. The actual performance difference in your environment will depend on various factors, including infrastructure and database workload profile

Because of the performance challenges associated with Synchronous Commit mode, many DBAs decide to implement high availability and disaster recovery by using a three-node cluster, with two nodes in the primary data center and one node in the DR data center. Instead of having two synchronous replicas within the primary data center, however, they stretch the primary replica across a failover clustered instance and configure the cluster to be able to host the instance only on these two nodes, and not on the third node in the DR data center. This is important, because it means that we don't need to implement SAN replication between the data centers. The DR node is synchronized using Availability Groups in Asynchronous Commit mode. If you combine an AlwaysOn failover clustered instance with AlwaysOn Availability Groups in this way, then automatic failover is not supported between the clustered instance

and the replica. It can only be configured for manual failover. There is also no need for Availability Groups to fail over between the two nodes hosting the clustered instance, because this failover is managed by the cluster service. This configuration can prove to be a highly powerful and flexible way to achieve your continuity requirements.

Summary

AlwaysOn Availability Groups can be implemented with up to eight secondary replicas, combining both Synchronous and Asynchronous Commit modes. When implementing high availability with Availability Groups, you always use Synchronous Commit mode, because Asynchronous Commit mode does not support automatic failover. When implementing Synchronous Commit mode, however, you must be aware of the associated performance penalty caused by committing the transaction on the secondary replica before it is committed on the primary replica. For disaster recovery, you will normally choose to implement Asynchronous Commit mode.

The Availability Group can be created via the New Availability Group Wizard, though dialog boxes, through T-SQL, or even through PowerShell. If you create an Availability Group using dialog boxes, then some aspects, such as the endpoint and associated permissions, must be scripted using T-SQL or PowerShell.



Implementing DR with AlwaysOn Availability Groups

In Chapter 5, we successfully implemented high availability for the App1Customers and App1Sales databases through the App1 Availability Group. In this chapter, we will discuss how we can also implement disaster recovery for these databases. To do this, we first need to build out a new server in our second site and install a stand-alone instance of SQL Server. Because the cluster now spans two sites, we need to reconfigure it as a multi-subnet cluster. We also need to reconfigure the quorum model to remove its dependency on the shared storage, which we currently have for the quorum. Once this is complete, we are able to add the instance on the new node to our Availability Group. The following sections assume that you have already built out a third server with a SQL Server instance called CLUSTERNODE3\ASYNCDR, and they demonstrate how to reconfigure the cluster as well as the Availability Group. The chapter will also discuss Distributed Availability Groups and readable secondary replicas. Therefore, the tasks that we will perform in this chapter are as follows:

- Use the Add Node Wizard, to add a third node to the cluster
- Modify the quorum, to use a file share, and exclude the DR node from voting
- Add an IP Address to the quorum
- Configure an OR constraint on the IP Address dependencies
- Configure the cluster's RegisterAllProvidersIP and HostRecordTTL settings

Configuring the Cluster

We need to perform several cluster configuration steps before we begin to alter our Availability Group. These include adding the new node, reconfiguring the quorum, and adding a new IP to the cluster's client access point.

Adding a Node

The first task in adding DR capability to our Availability Group is to add the third node to the cluster. To do this, we select Add Node from the context menu of nodes in Failover Cluster Manager. This causes the Add Node Wizard to be invoked. After passing through the Before You Begin page of this wizard, you are presented with the Select Servers page, which is illustrated in Figure 6-1. On this page, you need to enter the server name of the node that you plan to add to the cluster.

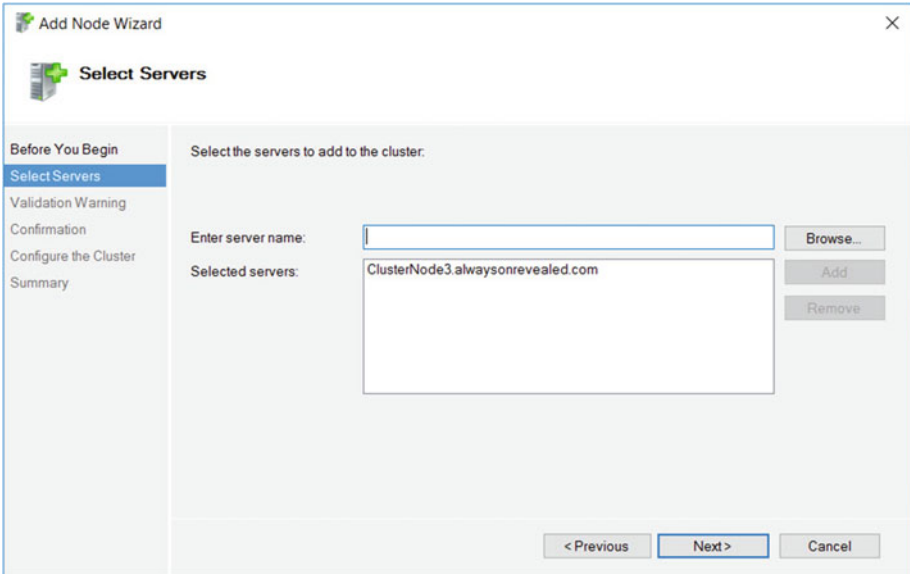


Figure 6-1. The Select Servers page

On the Validation Warning page, you are invited to run the Cluster Validation Wizard. You should always run this wizard in a production environment when making changes of this nature; otherwise, you will not be able to receive support from Microsoft for the cluster. Details of running the Cluster Validation wizard can be found in Chapter 3. Running the Cluster Validation wizard in our scenario is likely to throw up some warnings, which are detailed in Table 6-1.

Table 6-1. *Cluster Validation Warnings*

Warning	Reason	Resolution
This resource does not have all the nodes of the cluster listed as Possible Owners. The clustered role that this resource is a member of will not be able to start on any node that is not listed as a Possible Owner.	This warning has been displayed because we have not yet configured our Availability Group to use the new node.	Configuring the Availability Group to use the new node is discussed later in this chapter.
The RegisterAllProvidersIP property for network name 'Name: App1Listen' is set to 1. For the current cluster configuration this value should be set to 0.	Setting the RegisterAllProvidersIP to 1 will cause all IP Addresses to be registered, regardless of whether they are online or not. When we created the Availability Group Listener through SSMS, this setting was automatically configured to allow clients to fail over faster, and this warning should always be ignored. If we had created the Listener through Failover Cluster Manager, the property would have been set to 0 by default.	No resolution is required, but RegisterAllProvidersIP is discussed in more detail later in this chapter.

On the Confirmation page, we are given a summary of the tasks that will be performed. On this page, we deselect the option to add eligible storage, since one of our aims is to remove the dependency on shared storage. The Confirmation page is displayed in Figure 6-2.

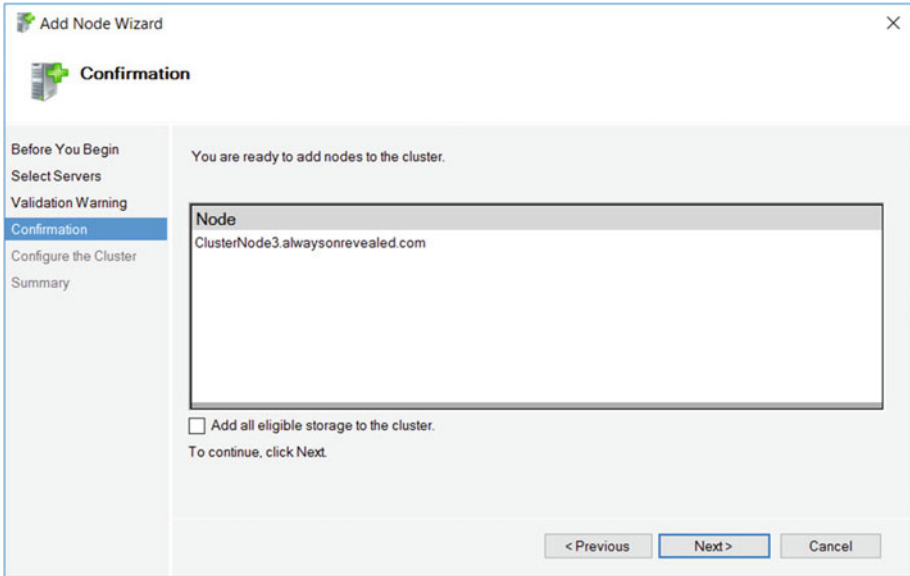


Figure 6-2. Confirmation page

On the Configure the Cluster page, the progress on the tasks displays until it is complete. The Summary page then displays, giving an overview of the actions and their success.

Modifying the Quorum

Our next step in configuring the cluster will be to modify the quorum. As mentioned earlier, we would like to remove our current dependency on shared storage. Therefore, we need to make a choice. Since we now have three nodes in the cluster, one possibility is to remove the disk witness and form a node majority quorum. The issue with this is that one of our nodes is in a different location. Therefore, if we lose network connectivity between the two sites for an extended period, then we have no fault tolerance in our primary site. If one of the nodes goes down, we lose quorum and the cluster goes offline. On the other hand, if we have an additional witness in the primary location, then we are not maintaining best practice, since there is an even number of votes. Again, if we lose one voting member, we lose resilience.

Therefore, the approach that we take is to replace the disk witness with a file share witness, thus removing the shared disk dependency. We then remove the vote from the node in the DR site. This means that we have three voting members of the quorum, and all of them are within the same site. This mitigates the risk of an intersite network issue causing loss of redundancy in our HA solution.

In order to invoke the Configure Cluster Quorum Wizard, we select Configure Cluster Quorum from the More Actions submenu within the context menu of our cluster in Failover Cluster Manager. After moving through the Before You Begin page of this wizard, you are asked to select the configuration that you wish to make on the Select Quorum Configuration Option page, which is shown in Figure 6-3. We select the Advanced Quorum Configuration option.

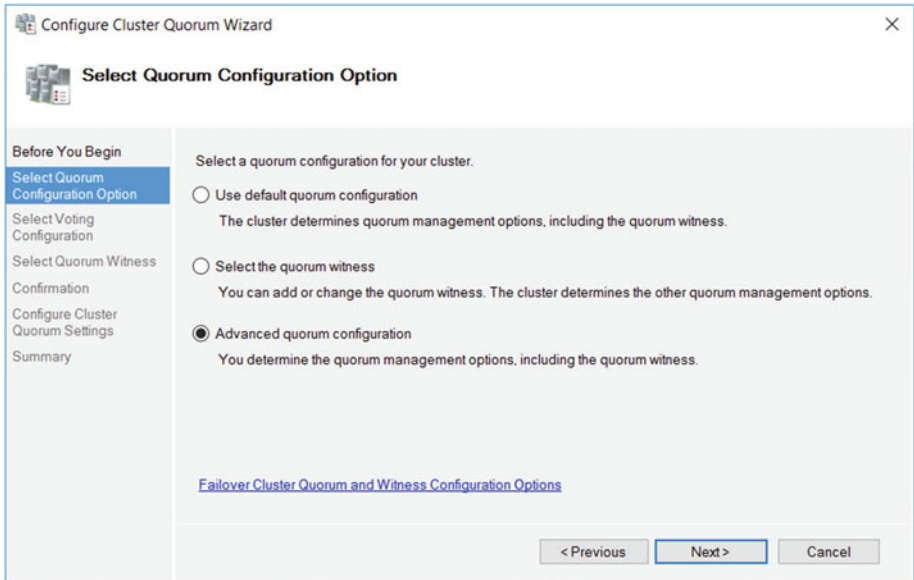


Figure 6-3. The Select Quorum Configuration Option page

On the Select Voting Configuration page, we choose to select nodes and remove the vote from CLUSTER NODE3. This is demonstrated in Figure 6-4.

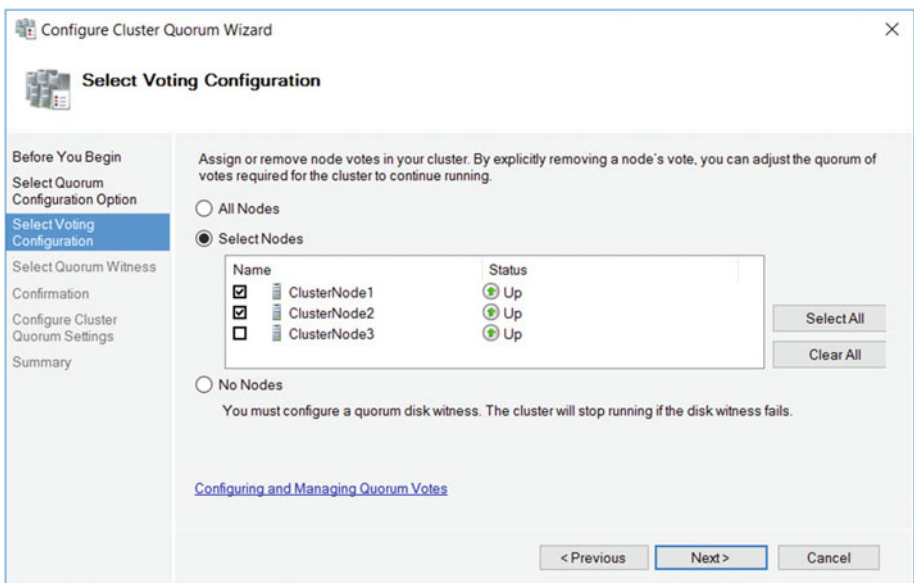


Figure 6-4. The Select Voting Configuration page

On the Select Quorum Witness page, we choose the Configure a File Share Witness option, as shown in Figure 6-5.

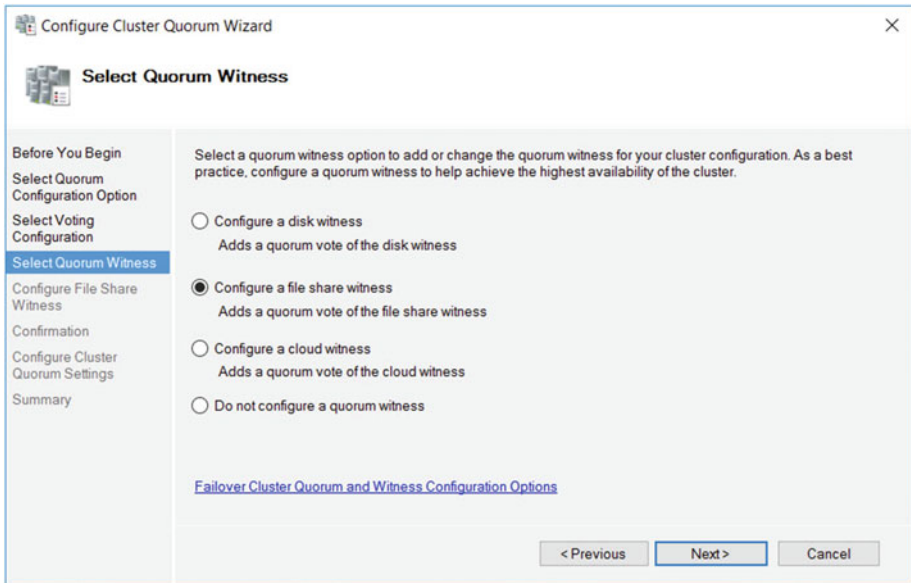


Figure 6-5. The Select Quorum Witness page

On the Configure File Share Witness page, which is illustrated in Figure 6-6, we enter the UNC of the share that we will use for the quorum. This file share must reside outside of the cluster and must be an SMB file share on a machine running Windows Server.

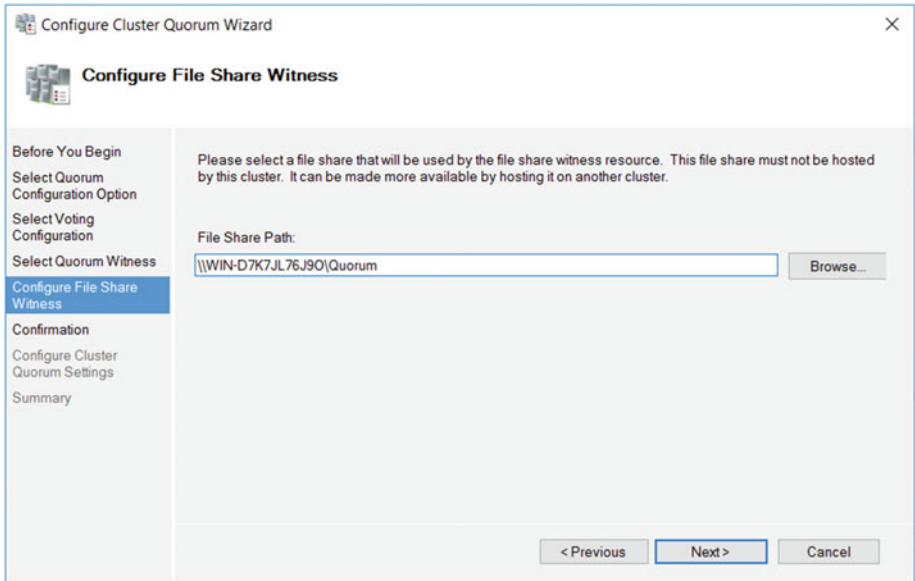


Figure 6-6. The *Configure File Share Witness* page

■ **Tip** Although many non-Windows-based NAS (network-attached storage) devices have support for SMB 3, I have experienced real-world implementations of a file share quorum on a NAS device work only intermittently, without resolution by either vendor.

■ **Caution** Remember that using a file share witness, with only two other voting nodes, can lead to a partition-in-time scenario. For more information, please refer to Chapter 3.

On the Confirmation page of the wizard, you are given a summary of the configuration changes that will be made, as shown in Figure 6-7.

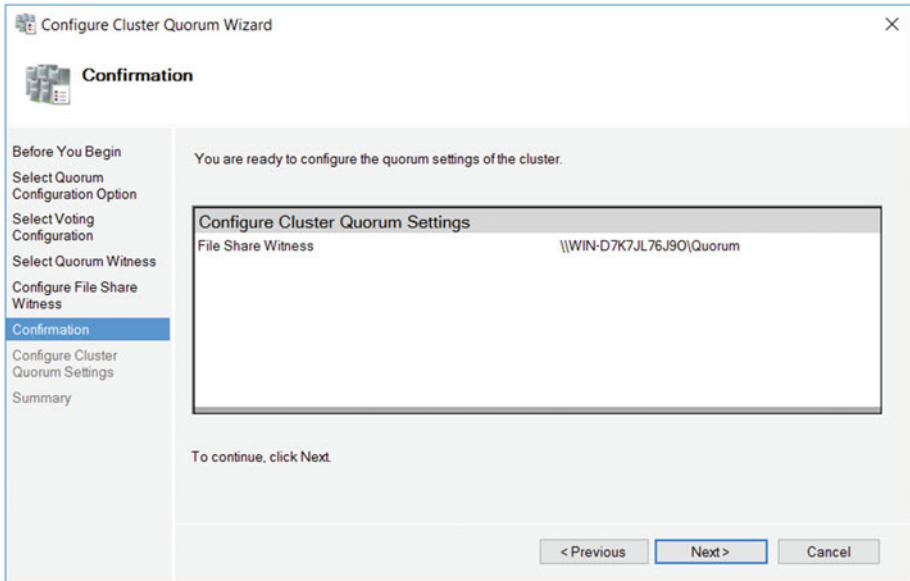


Figure 6-7. The Confirmation page

On the Configure Cluster Quorum Settings page, a progress bar displays. Once the configuration is complete, the Summary page appears. This page provides a summary of the configuration changes and a link to the report.

Adding an IP Address

Our next task is to add a second IP Address to the cluster’s client access point. We do not add an extra IP Address for the Availability Group Listener yet. We perform this task in the “Configuring the Availability Group” section later in this chapter.

In order to add the second IP Address, we select Properties from the context menu of Server Name in the Core Cluster Resources window of Failover Cluster Manager. On the General tab of the Cluster Properties dialog box, we add the IP Address for administrative clients, following failover to DR, as illustrated in Figure 6-8.

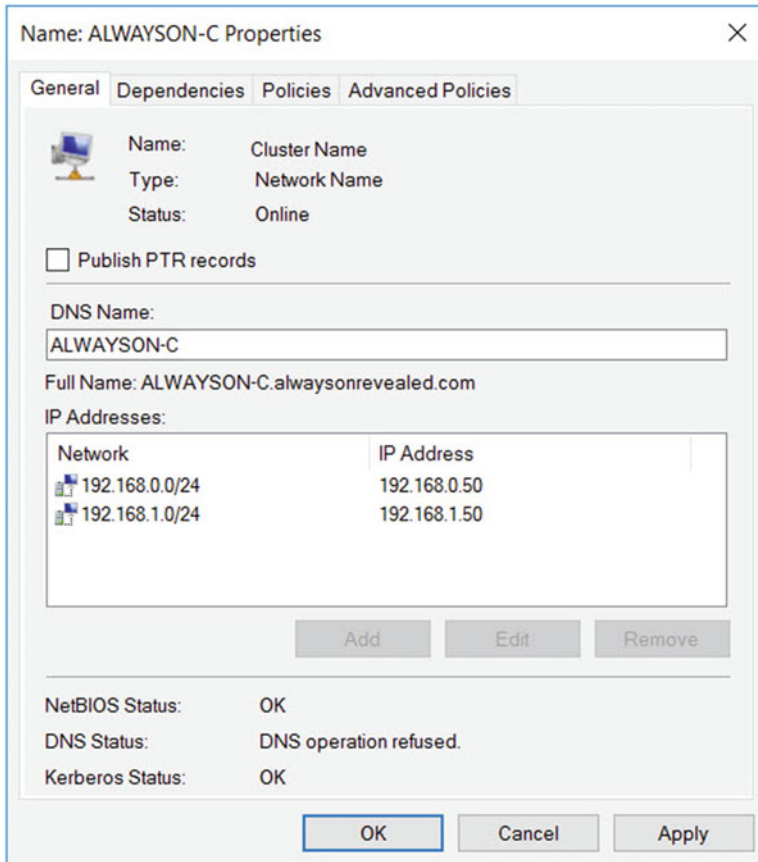


Figure 6-8. The General tab

When we apply the change, we receive a warning saying that administrative clients will temporarily be disconnected from the cluster. This does not include any clients connected to our Availability Group. If we choose to proceed, we then navigate to the Dependencies tab of the dialog box and ensure that an OR dependency has been created between our two IP Addresses, as shown in Figure 6-9.

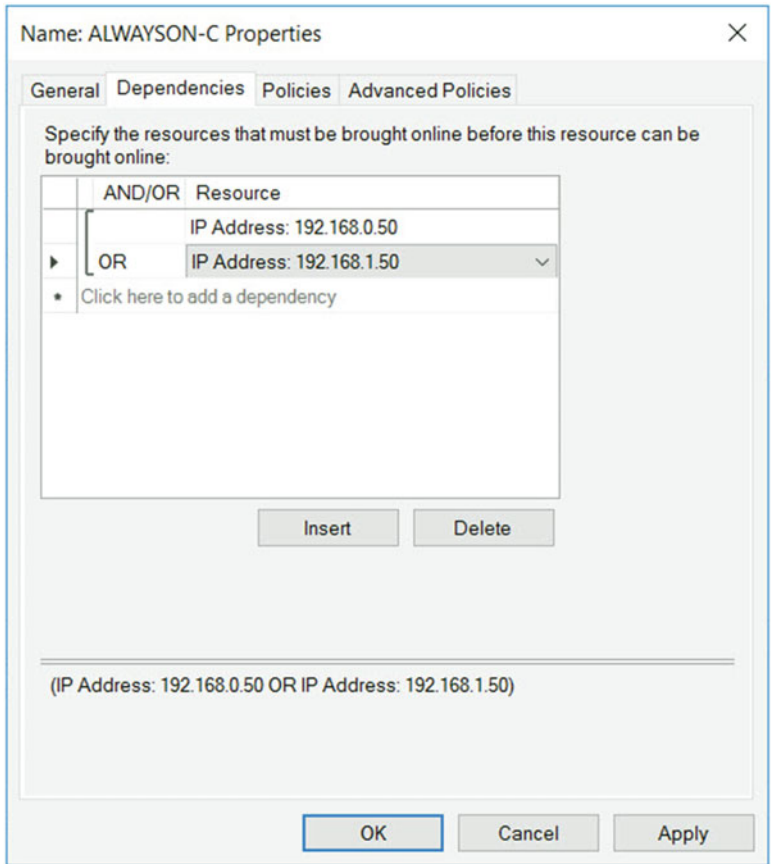


Figure 6-9. The Dependencies tab

After this process is complete, the second IP Address resource in the Cluster Core Resources group shows up as offline. This is normal. In the event of failover to the server in the second subnet, this IP Address comes online, and the IP Address of the subnet in the primary site goes offline. This is why the OR dependency (as opposed to an AND dependency) is critical. Without it, the Server Name resource could never be online.

Configuring the Availability Group

To configure the Availability Group, we first have to add the new node as a replica and configure its properties. We then add a new IP Address to our listener for the second subnet. Finally, we look at improving the connection times for clients.

Adding and Configuring a Replica

In SQL Server Management Studio (SSMS), on the primary replica, we drill through Availability Groups | App1 and select Add Replica from the context menu of the Availability Replicas node. This causes the Add Replica to Availability Group wizard to be displayed. After passing through the Introduction page of the wizard, you see the Connect to Replicas page, as displayed in Figure 6-10. On this page, you are invited to connect to the other replicas in the Availability Group.

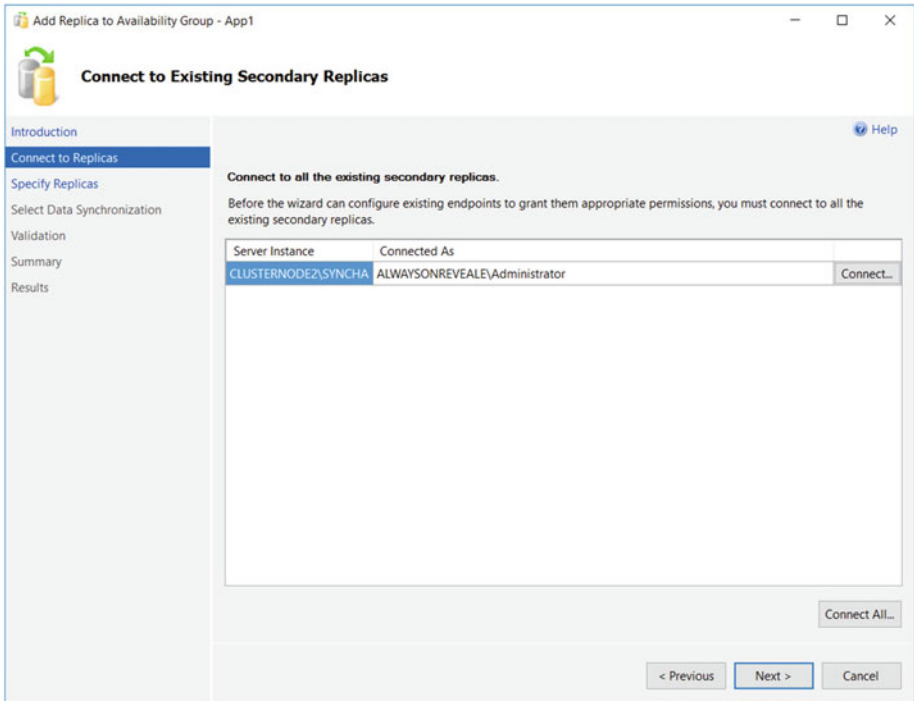


Figure 6-10. The Connect to Replicas page

On the Replicas tab of the Specify Replicas page, shown in Figure 6-11, we first use the Add Replica button to connect to the DR instance. After we have connected to the new replica, we specify the properties for that replica. In this case, we leave them as-is because it will be a DR replica. Therefore, we want it to be asynchronous and we do not want it to be readable.

Specify Replicas

Introduction
Connect to Replicas
Specify Replicas
Select Data Synchronization
Validation
Summary
Results

Help

Specify an instance of SQL Server to host a secondary replica.

Replicas | Endpoints | Backup Preferences | Listener

Availability Replicas:

Server Instance	Initial Role	Automatic Failover (Up to 3)	Synchronous Commit (Up to 3)	Readable Secondary
CLUSTERNODE1\PRIMARYREPLICA	Primary	<input type="checkbox"/>	<input checked="" type="checkbox"/>	No
CLUSTERNODE2\SYNCHA	Second...	<input type="checkbox"/>	<input checked="" type="checkbox"/>	No
CLUSTERNODE3\ASYNCDR	Second...	<input type="checkbox"/>	<input type="checkbox"/>	No

Add Replica... Add Azure Replica... Remove Replica

Summary for the replica hosted by CLUSTERNODE1\PRIMARYREPLICA

Replica mode: Synchronous commit
This replica will use synchronous-commit availability mode and support only manual failover.

Readable secondary: No
In the secondary role, this availability replica will not allow any connections.

< Previous Next > Cancel

Figure 6-11. The Replicas tab

On the Endpoints tab, shown in Figure 6-12, we ensure that the default settings are correct and acceptable.

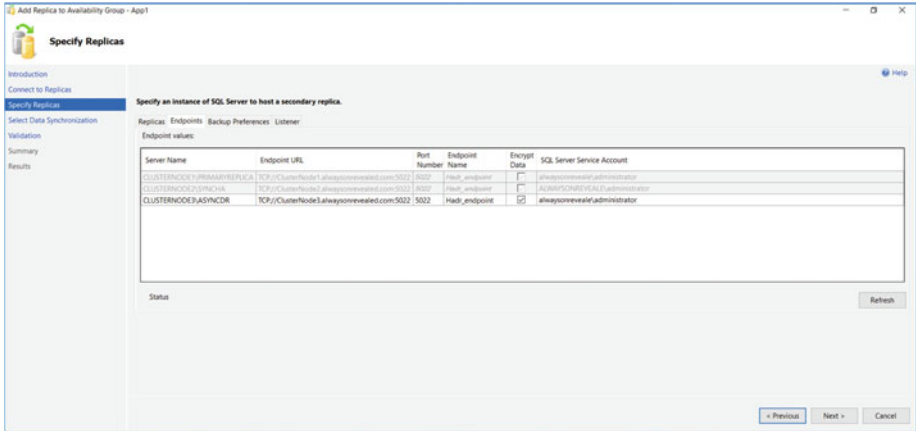


Figure 6-12. The Endpoints tab

On the Backup Preferences tab, the option for specifying the preferred backup replica is read-only. We are, however, able to specifically exclude our new replica as a candidate for backups or change its backup priority. This page is displayed in Figure 6-13.

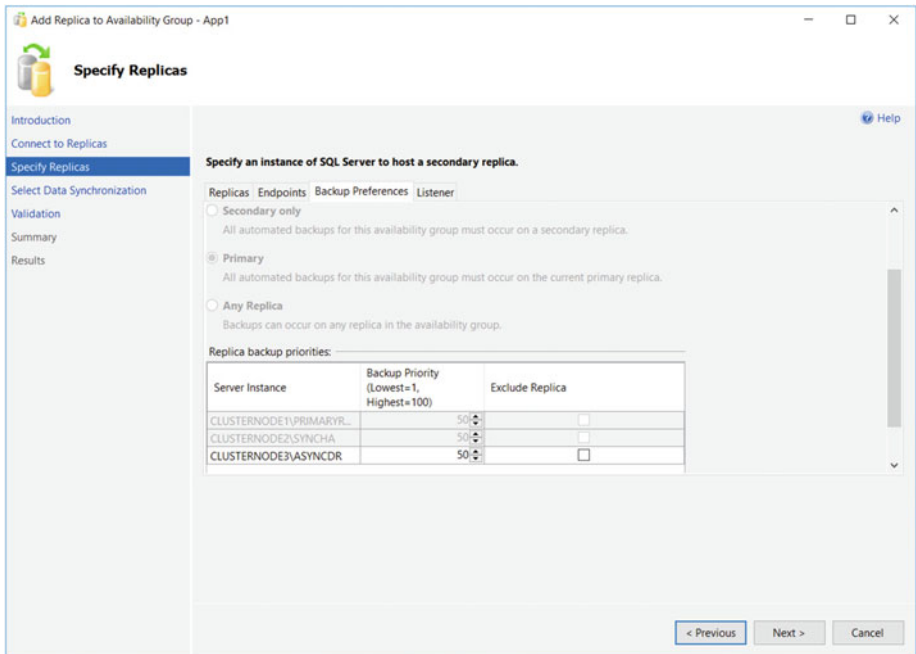


Figure 6-13. The Backup Preferences tab

On the Listener tab, illustrated in Figure 6-14, we can decide if we will create a new listener. This is a strange option, since SQL Server only allows us to create a single listener for an Availability Group, and we already have one. Therefore, we leave the default choice of Do Not Create an Availability Group Listener selected. It is possible to create a second listener, directly from Failover Cluster Manager, but you would only want a second listener for the same Availability Group in very rare, special cases, which we discuss later in this chapter.

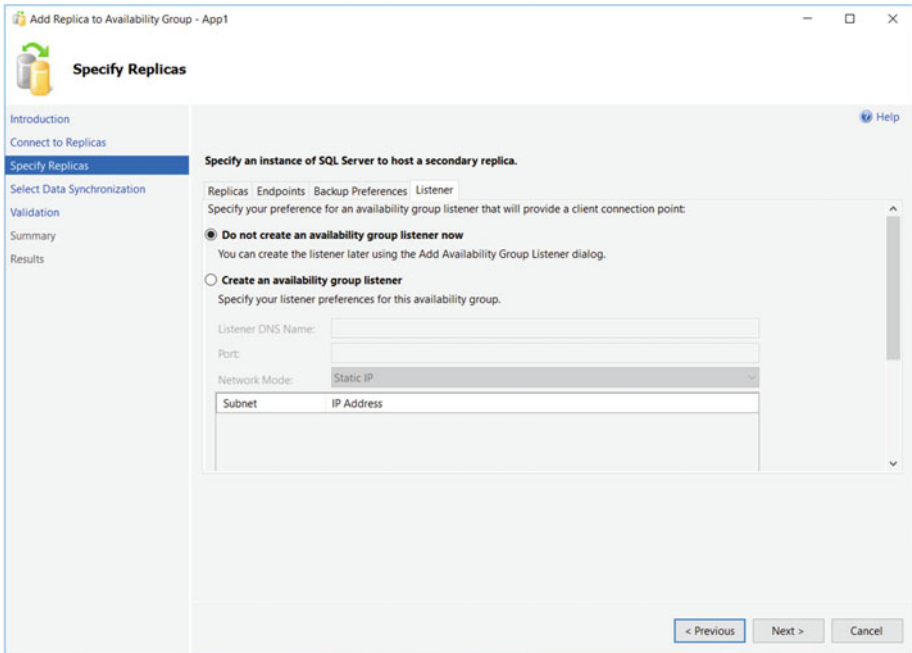


Figure 6-14. The Listener tab

On the Select Data Synchronization page, we choose how we want to perform the initial synchronization of the replica. The options are the same as they were when we created the Availability Group, except that the file share will be prepopulated, assuming that we choose the Full synchronization when creating the Availability Group. This screen is shown in Figure 6-15.

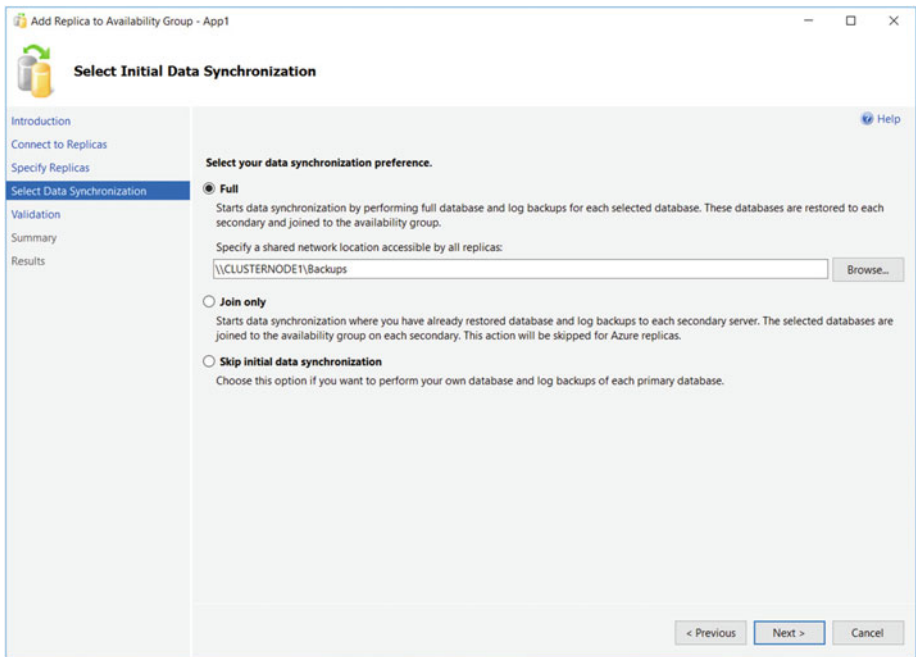


Figure 6-15. The Select Data Synchronization page

On the Validation page, we should review any warnings or errors and resolve them before continuing. Finally, on the Summary page, we are presented with a summary of the configurations that will be carried out.

After the reconfiguration completes, our new replica is added to the cluster. We should then review the results and respond to any warnings or errors. We could also have used T-SQL to add the replica to the Availability Group. The script in Listing 6-1 performs the same actions just demonstrated. You must run this script in SQLCMD Mode since it connects to multiple instances.

■ **Tip** You will notice that the following script (and other scripts throughout the book) drops the final character from the domain name. This is because `alwaysonrevealed` exceeds the 15 characters allowed as a NETBIOS name. Therefore, SQL Server does not recognize the domain name. The work-around for this in SQL Server is to only specify the first 15 characters. While this seems a little strange, as SQL Server actually stores the data as the `sysname` data type, which is a synonym for `NVARCHAR(128)`, it is a better situation than we are faced with when dealing login names. Here, the maximum length is 16 characters, and while longer names are fine if the login maps to a Windows group, if the login maps to a Windows user, then the login can be created, but cannot log in.

Listing 6-1. Adding a Replica

```

:Connect CLUSTERNODE3\ASYNC DR

--Create Login for Service Account

USE master
GO

CREATE LOGIN [alwaysonreveale\SQLAdmin] FROM WINDOWS ;
GO

--Create the Endpoint

CREATE ENDPOINT Hadr_endpoint
    AS TCP (LISTENER_PORT = 5022)
    FOR DATA_MIRRORING (ROLE = ALL, ENCRYPTION = REQUIRED ALGORITHM AES) ;
GO

ALTER ENDPOINT Hadr_endpoint STATE = STARTED ;
GO

--Grant the Service Account permissions to the Endpoint

GRANT CONNECT ON ENDPOINT::[Hadr_endpoint] TO [alwaysonreveale\SQLAdmin] ;
GO

--Start the AOAG Health Trace

IF EXISTS(SELECT * FROM sys.server_event_sessions WHERE name='AlwaysOn_health')
BEGIN
    ALTER EVENT SESSION AlwaysOn_health ON SERVER WITH (STARTUP_STATE=ON);
END
IF NOT EXISTS(SELECT * FROM sys.dm_xe_sessions WHERE name='AlwaysOn_health')
BEGIN
    ALTER EVENT SESSION AlwaysOn_health ON SERVER STATE=START;
END
GO

:Connect CLUSTERNODE1\PRIMARYREPLICA

USE master
GO

```

```

--Add the replica to the Availability Group

ALTER AVAILABILITY GROUP App1
ADD REPLICA ON N'CLUSTERNODE3\ASYNCDR'
WITH (ENDPOINT_URL = N'TCP://CLUSTERNODE3.ALWAYSONREVEALE.COM:5022',
      FAILOVER_MODE = MANUAL, AVAILABILITY_MODE = ASYNCHRONOUS_COMMIT,
      BACKUP_PRIORITY = 50,
      SECONDARY_ROLE(ALLOW_CONNECTIONS = NO));

GO

--Back up and restore the first database and log

BACKUP DATABASE App1Customers TO DISK = N'\\CLUSTERNODE1\Backups\
App1Customers.bak'
WITH COPY_ONLY, FORMAT, INIT, REWIND, COMPRESSION, STATS = 5 ;
GO

BACKUP LOG App1Customers
TO DISK = N'\\CLUSTERNODE1\Backups\App1Customers.trn'
WITH NOSKIP, REWIND, COMPRESSION, STATS = 5 ;
GO

:Connect CLUSTERNODE3\ASYNCDR

ALTER AVAILABILITY GROUP App1 JOIN;
GO

RESTORE DATABASE App1Customers
FROM DISK = N'\\CLUSTERNODE1\Backups\App1Customers.bak'
WITH NORECOVERY, STATS = 5 ;
GO

RESTORE LOG App1Customers
FROM DISK = N'\\CLUSTERNODE1\Backups\App1Customers.trn'
WITH NORECOVERY, STATS = 5 ;
GO

-- Wait for the replica to start communicating
DECLARE @connection BIT

DECLARE @replica_id UNIQUEIDENTIFIER
DECLARE @group_id UNIQUEIDENTIFIER

SET @connection = 0

```

```

WHILE @Connection = 0
BEGIN
    SET @group_id = (SELECT group_id
                     FROM Master.sys.availability_groups
                     WHERE name = N'App1')
    SET @replica_id = (SELECT replica_id
                      FROM Master.sys.availability_replicas
                      WHERE UPPER(replica_server_name COLLATE Latin1_
                                General_CI_AS) =
                          UPPER(@@SERVERNAME COLLATE Latin1_General_CI_AS)
                          AND group_id = @group_id)

    SET @connection = ISNULL((SELECT connected_state
                              FROM Master.sys.dm_hadr_availability_
                               replica_states
                              WHERE replica_id = @replica_id), 1)

    WAITFOR DELAY '00:00:10'
END

--Add the first Database to the Availability Group on the new replica

ALTER DATABASE App1Customers SET HADR AVAILABILITY GROUP = [App1];
GO

--Back up and restore the second database and log

:Connect CLUSTERNODE1\PRIMARYREPLICA

BACKUP DATABASE App1Sales
TO DISK = N'\\CLUSTERNODE1\Backups\App1Sales.bak'
WITH COPY_ONLY, FORMAT, INIT, REWIND, COMPRESSION, STATS = 5 ;
GO

BACKUP LOG App1Sales
TO DISK = N'\\CLUSTERNODE1\Backups\App1Sales.trn'
WITH NOSKIP, REWIND, COMPRESSION, STATS = 5 ;
GO

:Connect CLUSTERNODE3\ASYNCDR

ALTER AVAILABILITY GROUP [App1] JOIN;
GO

RESTORE DATABASE App1Sales
FROM DISK = N'\\CLUSTERNODE1\Backups\App1Sales.bak'
WITH NORECOVERY, STATS = 5 ;
GO

```



```

RESTORE LOG App1Sales
FROM DISK = N'\\CLUSTERNODE1\Backups\App1Sales.trn'
WITH NORECOVERY, STATS = 5 ;
GO

-- Wait for the replica to start communicating
DECLARE @connection BIT

DECLARE @replica_id UNIQUEIDENTIFIER
DECLARE @group_id UNIQUEIDENTIFIER

SET @connection = 0

WHILE @Connection = 0
BEGIN
    SET @group_id = (SELECT group_id
                     FROM Master.sys.availability_groups
                     WHERE name = N'App1')
    SET @replica_id = (SELECT replica_id
                       FROM Master.sys.availability_replicas
                       WHERE UPPER(replica_server_name COLLATE Latin1_
                                   General_CI_AS) =
                           UPPER(@@SERVERNAME COLLATE Latin1_
                                   General_CI_AS)
                       AND group_id = @group_id)

    SET @connection = ISNULL((SELECT connected_state
                              FROM Master.sys.dm_hadr_availability_
                               replica_states
                              WHERE replica_id = @replica_id), 1)

    WAITFOR DELAY '00:00:10'
END

--Add the second database to the Availability Group on the new replica

ALTER DATABASE App1Sales SET HADR AVAILABILITY GROUP = App1;
GO

```

Add an IP Address

Even though the replica has been added to the Availability Group and we are able to fail over to this replica, our clients are still not able to connect to it in the DR site using the Availability Group Listener. This is because we need to add an IP Address resource, which resides in the second subnet. To do this, we can select Properties from the context menu of App1Listen in Object Explorer, which causes the Availability Group Listener Properties dialog box to be displayed, as in Figure 6-16. Here, we add the listener's second IP Address.

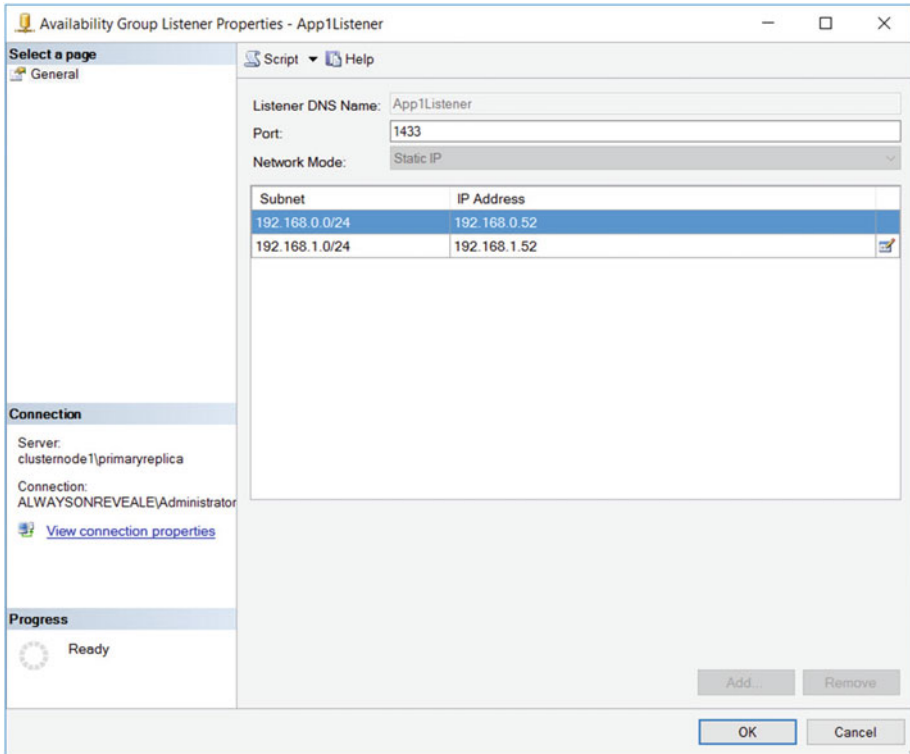


Figure 6-16. The Availability Group Listener Properties

We can also achieve this through T-SQL by running the script in Listing 6-2.

Listing 6-2. Adding an IP Address to the Listener

```
ALTER AVAILABILITY GROUP App1
MODIFY LISTENER 'App1Listen'
(ADD IP (N'192.168.1.52', N'255.255.255.0')) ;
```

SQL Server now adds the IP Address as a resource in the App1 role, and also configures the OR dependency on the Name resource. You can view this by running the dependency report against the Name resource in Failover Cluster Manager, as illustrated in Figure 6-17.

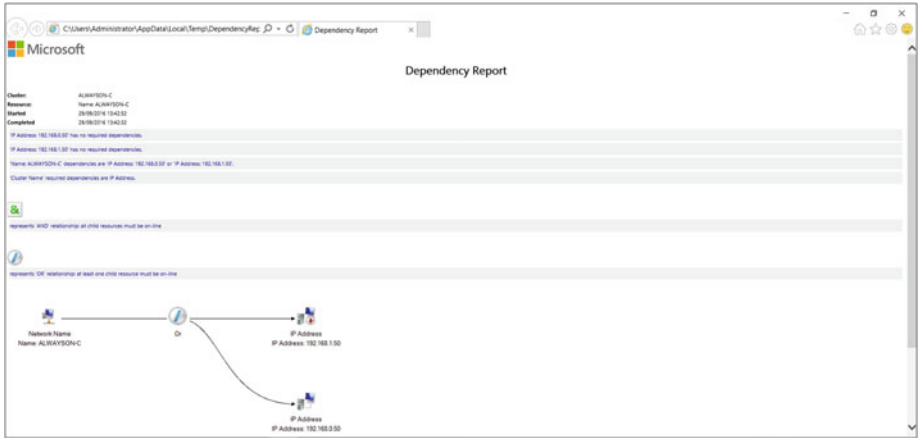


Figure 6-17. The dependency report

Improving Connection Times

Clients using .NET 4 or higher are able to specify the new `MultiSubnetFailover=True` property in their connecting strings when connecting to an AlwaysOn Availability Group. This improves connection times by retrying TCP connections more aggressively. If clients are using older versions of .NET, however, then there is a high risk of their connections timing out.

There are two workarounds for this issue. The first is to set the `RegisterAllProvidersIP` property to 0. This is the recommended approach, but the problem with it is that failover to the DR site can take up to 15 minutes. This is because the IP Address resource for the second subnet is offline until failover occurs. It can then take up to 15 minutes for the PTR record to be published. In order to reduce this risk, it is recommended that you also lower the `HostRecordTTL`. This property defines how often the resource records for the cluster name are published.

The script in Listing 6-3 demonstrates how to disable `RegisterAllProvidersIP` and then reduce the `HostRecordTTL` to 300 seconds.

Listing 6-3. Configuring Connection Properties

```
Get-ClusterResource "App1_App1Listen" | Set-ClusterParameter
RegisterAllProvidersIP 0
```

```
Get-ClusterResource "App1_App1Listen" | Set-ClusterParameter HostRecordTTL 300
```

The alternative workaround is to simply increase the timeout value for connections to 30 seconds. However, this solution accepts that a large volume of connections will take up to 30 seconds. This may not be acceptable to the business.

Distributed Availability Groups

Distributed Availability Groups are a new feature of SQL Server 2016, which offer an alternative DR configuration, which can reduce network traffic and mitigate the risk of cluster health at the DR site causing issues in the primary site, while still being able to fully monitor quorum in the DR site.

Instead of a single cluster being stretched across two sites, distributed Availability Groups allow you to join together Availability Groups which reside on multiple clusters. Each cluster maintains its own quorum, so the DR site becoming unavailable will have no impact on the primary site. Additionally, network traffic is reduced between the sites, if there are multiple replicas in the secondary site, because the data from the primary site is only replicated once, as opposed to being replicated to each individual replica, as it would be with a traditional stretch cluster configuration. Other clustering rules are also relaxed. For example, the cluster nodes in the DR site cluster can optimally run a different version of the operating system, to the cluster nodes in the primary site cluster.

■ **Tip** While many rules are relaxed, the configuration and folder locations of the databases within the primary and secondary Availability Groups must be the same, just as if you used a single stretch cluster. The same port number must also be used for the database mirroring endpoint.

A typical Distributed Availability Group topology is illustrated in Figure 6-18.

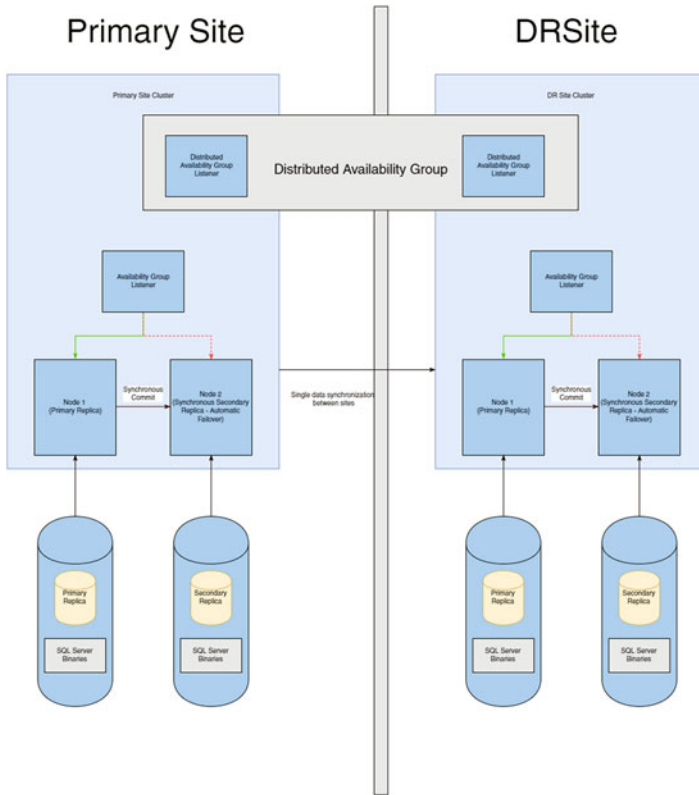


Figure 6-18. Distributed Availability Group topology

Distributed Availability Groups can also be configured with a SQL Server AlwaysOn Failover Clustered Instance. In this scenario, a cluster in the primary site would host a failover clustered instance, with an Availability Group in the DR site providing DR, without the need for a stretch cluster.

The script in Listing 6-4 demonstrates how to create a distributed Availability Group, for our App2 Availability Group. The script assumes that you have already created an Availability Group on a second cluster, which matches the configuration of the App2 Availability Group, which resides on the ALWAYS ON-C cluster. It assumes that you have named the Availability Group listener on the DR cluster App2DR. The script then joins the Availability Group on the second server, to the distributed Availability Group.

■ **Tip** Once you have joined the secondary Availability Group to the Distributed Availability Group, it will automatically become read only, and updates will only be allowed, via the synchronization from the primary Availability Group.

Listing 6-4. Creating a Distributed Availability Group

```

CREATE AVAILABILITY GROUP App2Distributed
WITH (DISTRIBUTED)
AVAILABILITY GROUP ON
    'App2' WITH
    (
        LISTENER_URL = 'tcp://App2_App2Listen:5022',
        AVAILABILITY_MODE = ASYNCHRONOUS_COMMIT,
        FAILOVER_MODE = MANUAL,
        SEEDING_MODE = AUTOMATIC
    ),
    'App2DR' WITH
    (
        LISTENER_URL = 'tcp://App2_App2Listen:5022',
        AVAILABILITY_MODE = ASYNCHRONOUS_COMMIT,
        FAILOVER_MODE = MANUAL,
        SEEDING_MODE = AUTOMATIC
    );
GO

```

■ **Note** Do not follow the demonstrations in this section, if you wish to follow the demonstrations in later chapters.

Configuring Readable Secondary Replicas

It can be very useful to add readable secondary replicas to an AlwaysOn Availability Group topology in order to implement vertically scaled reporting. When you use this strategy, the databases are kept synchronized, with variable, but typically low latency, using log streaming. The additional advantage of readable secondary replicas from SQL Server 2014 onward is that they stay online, even if the primary replica is offline. The limitation when using this availability feature, however, is that users must connect directly to the instance, as opposed to the Availability Group Listener.

You can further improve read performance in readable secondary replicas by using temporary statistics, which you can also use to optimize read-only workloads. Also, snapshot isolation is also used exclusively on readable secondary replicas, even if other isolation levels or locking hints are explicitly requested. This helps avoid contention, but it also means that TempDB should be suitably scaled and on a fast disk array.

The main risk of using readable secondary replicas is that implementing snapshot isolation on the secondary replica can actually cause deleted records not to be cleaned up on the primary replica. This is because the ghost record clean-up task only removes rows from the primary once they are no longer required at the secondary. In this scenario, log truncation is also delayed on the primary replica. This means that you potentially risk having to kill long-running queries that are being satisfied against the readable

secondary. This issue can also occur if the secondary replica becomes disconnected from the primary. Therefore, there is a risk that you may need to remove the secondary replica from the Availability Group and subsequently read it.

To make a secondary replica readable, you need to perform three tasks. First configure the secondary replica to allow read-only connections. Second, specify a read-only URL for reporting. The Availability Group Listener then directs appropriate traffic to this URL. The final task is to update the read-only routing list on the primary replica. These tasks are performed by the script in Listing 6-5.

Listing 6-5. Configuring Read-Only Routing

```
--Configure the ASYNCDR Replica to allow read-only connections

ALTER AVAILABILITY GROUP App1
  MODIFY REPLICATION ON N'CLUSTERNODE3\ASYNCDR' WITH
  (SECONDARY_ROLE (ALLOW_CONNECTIONS = READ_ONLY)) ;

--Configure the read-only URL for the ASYNCDR Replica

ALTER AVAILABILITY GROUP App1
  MODIFY REPLICATION ON N'CLUSTERNODE3\ASYNCDR' WITH
  (SECONDARY_ROLE (READ_ONLY_ROUTING_URL = N'TCP://CLUSTERNODE3.
  ALWAYSONREVEALE.com:1433')) ;

--Configure the read-only routing list on the Primary Replica

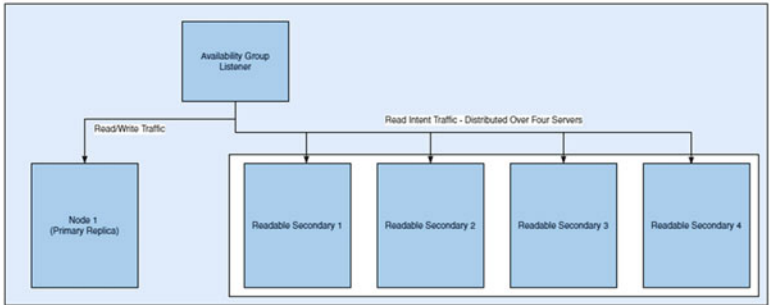
ALTER AVAILABILITY GROUP App1
  MODIFY REPLICATION ON N'CLUSTERNODE1\PRIMARYREPLICA' WITH
  (PRIMARY_ROLE (READ_ONLY_ROUTING_LIST=('CLUSTERNODE3\ASYNCDR')));
```

Prior to SQL Server 2016, if you specified multiple readable secondary replicas in the routing list, then the listener would attempt to direct read-intent traffic to the first replica in the routing list. If this replica was not available, it would attempt to write to the second replica in the routing list, and so on. The result of this, was that while you could scale reporting to a secondary server, you could not load balance those reports across multiple secondary servers.

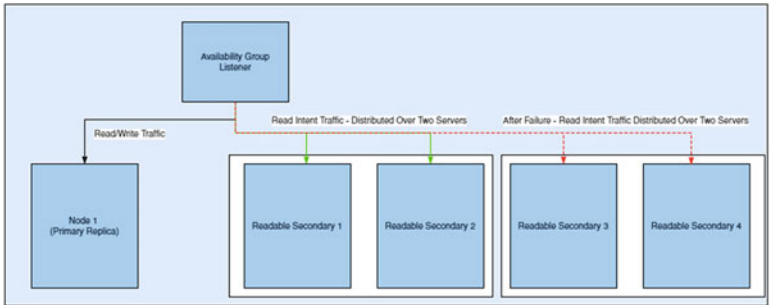
SQL Server 2016 introduces load balancing for active secondary replicas, however. When using load balancing for active secondaries, the listener will balance the read-intent workload, by distributing it across the specified replicas with a round-robin algorithm.

Additionally, you can specify groups of balanced replicas. For example, imagine that we added four additional replicas to the App1 Availability Group, all of which were intended for the offloading of reporting. Figure 6-19 shows several different topologies that could be used, depending on your requirements for consistency of performance.

Example 1



Example 2



Example 3

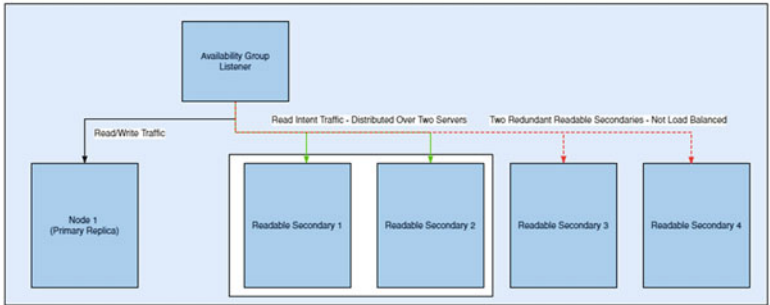


Figure 6-19. Possible load balancing topologies

Tip Often, business users will value consistency of performance over the performance itself.

In the first example, traffic will be balanced across all four active secondaries. The second example illustrates that the replicas have been split into two separate groups. The listener will first try to route read-intent request to the first group of servers and will balance the load between the two. If the first group of servers is unavailable, then the listener will route the read-intent traffic to the second group of servers, and balance the load between the servers within this group. In the final example, the listener will attempt to route traffic to the two server group. If this group of servers is unavailable, then it will route all read-intent traffic to the third server, with no load balancing. If the third server also goes offline, then all read-intent traffic will be routed to the fourth server, again with no load balancing.

The script in Listing 6-6 demonstrates how to modify the read-only routing list on the primary replica, to configure load balancing as illustrated in the second example. The script assumes that the read-only routing URLs have already been configured on the active secondaries. Notice that each server group has been enclosed in nested parenthesis.

Listing 6-6. Configure the Read-Only Routing List for Load Balancing

```
ALTER AVAILABILITY GROUP App1
MODIFY REPLICA ON N'CLUSTERNODE1\PRIMARYREPLICA' WITH
(PRIMARY_ROLE (READ_ONLY_ROUTING_LIST=((('CLUSTERNODE4\READ01','CLUSTERNODE5\
READ02'),('CLUSTERNODE6\READ03','CLUSTERNODE7\READ04'))))) ;
```

Summary

If you implement disaster recovery with Availability Groups, then you need to configure a multi-subnet cluster. This does not mean that you must have SAN replication between the sites, however, since Availability Groups do not rely on shared storage. What you do need to do is add additional IP Addresses for the administrative cluster access point and also for the Availability Group Listener. You also need to pay attention to the properties of the cluster that support client reconnection to ensure that clients do not experience a high number of timeouts.

CHAPTER 7



Administering AlwaysOn

This chapter will discuss how to administer AlwaysOn features. We will first look at cluster maintenance, including rolling patch upgrades and removing an instance. We will then discuss managing Availability Groups, including how to fail over synchronously and asynchronously. We will also examine how to fail over a Distributed Availability Group.

Additional maintenance tasks, such as synchronizing instance-level objects, safe-stating an application, and adding multiple Listeners to an Availability Group, will also be discussed. The chapter will end by discussing how to suspend data movement and how to remove a database from an Availability Group.

Managing a Cluster

Installing the cluster is not the end of the road from an administrative perspective. You still need to periodically perform maintenance tasks. The following sections describe some of the most common maintenance tasks.

Moving the Instance Between Nodes

Other than protecting against unplanned outages, one of the benefits of implementing high availability technologies is that doing so significantly reduces downtime for maintenance tasks, such as patching. This can be at the operating system level or the SQL Server level.

If you have a two-node cluster, apply the patch to the passive node first. Once you are happy that the update was successful, fail over the instance and then apply the patch to the other node. At this point, you may or may not wish to fail back to the original node, depending on the needs of your environment. For example, if the overriding priority is the level of availability of the instance, then you will probably not wish to fail back, because this will incur another short outage.

On the other hand, if your instance is less critical and you have licensed SQL Server with Software Assurance, then you may not be paying for the SQL Server license on the passive node. In this scenario, you only have a limited time period in which to fail the instance back to avoid needing to purchase an additional license for the passive node.

■ **Note** For versions of SQL Server prior to SQL Server 2014, Software Assurance is not required in order to have a passive node without a license.

To move an instance to a different node using Failover Cluster Manager, select Move | Select Node from the context menu of the role that contains the instance. This causes the Move Clustered Role dialog box to display. Here, you can select the node to which you wish to move the role, as illustrated in Figure 7-1.

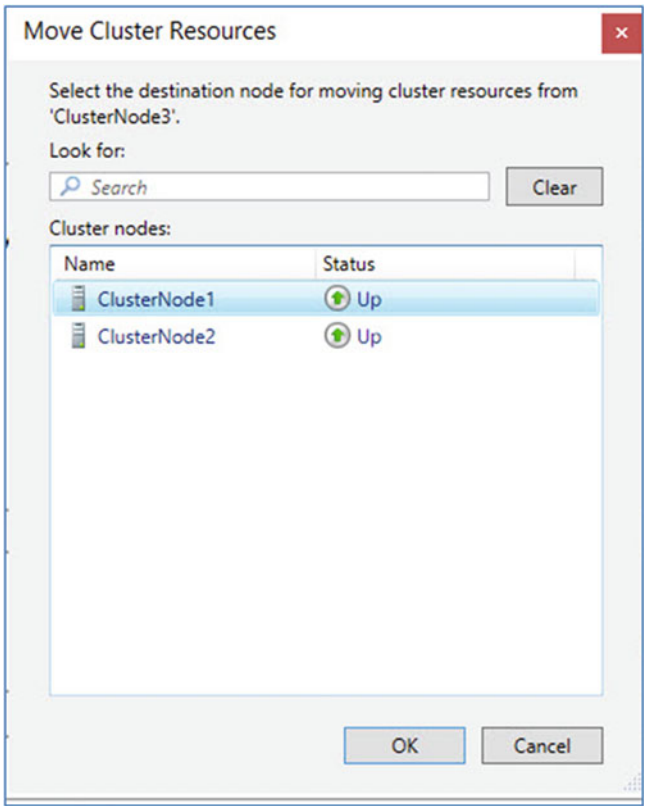


Figure 7-1. The Move Clustered Role dialog box

The role is then moved to the new node. If you watch the role's resources window in Failover Cluster Manager, then you see each resource move through the states of Online ► Offline Pending ► Offline. The new node is now displayed as the owner before the resources move in turn through the states of Offline ► Online Pending ► Online. The resources are taken offline and placed back online in order of their dependencies.

We can also fail over a role using PowerShell. To do this, we need to use the `Move-ClusterGroup` cmdlet. Listing 7-1 demonstrates this by using the cmdlet to fail back the instance to `ClusterNode1`. We use the `-Name` parameter to specify the role that we wish to move and the `-Node` parameter to specify the node to which we wish to move it.

Listing 7-1. Moving the Role Between Nodes

```
Move-ClusterGroup -Name "SQL Server (ALWAYSON-C)" -Node ClusterNode1
```

Rolling Patch Upgrade

If you have a cluster with more than two nodes, then consider performing a rolling patch upgrade when you are applying updates for SQL Server. In this scenario, you mitigate the risk of having different nodes, which are possible owners of the role, running different versions or patch levels of SQL Server, which could lead to data corruption.

The first thing that you should do is make a list of all nodes that are possible owners of the role. Then select 50% of these nodes and remove them from the Possible Owners list. You can do this by selecting Properties from the context menu of the Name resource, and then, in the Advanced Policies tab, unchecking the nodes in the possible owners list, as illustrated in Figure 7-2.

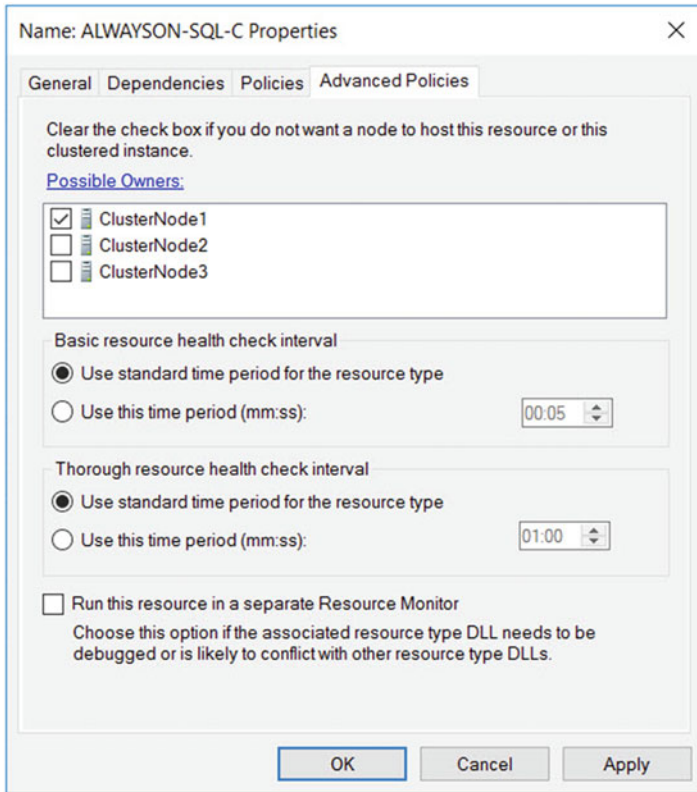


Figure 7-2. Remove possible owners.

To achieve the same result using PowerShell, we can use the `Get-Resource` cmdlet to navigate to the name resource and then pipe in the `Set-ClusterOwnerNode` to configure the possible owners list. This is demonstrated in Listing 7-2. The possible owners list is comma separated in the event that you are configuring multiple possible owners.

Listing 7-2. Configuring Possible Owners

```
Get-ClusterResource "SQL Network Name (ALWAYS ON-SQL-C)" | Set-ClusterOwnerNode -Owners clusternode1
```

Once 50% of the nodes have been removed as possible owners, you should apply the update to these nodes. After the update has been verified on this half of the nodes, you should reconfigure them to allow them to be possible owners once more.

The next step is to move the role to one of the nodes that you have upgraded. After failover has successfully completed, remove the other half of the nodes from the preferred owners list before applying the update to these nodes. Once the update has been verified on this half of the nodes, you can return them to the possible owners list.

■ **Tip** The possible owners can only be set on a resource. If you run `Set-ClusterOwnerNode` against a role using the `-Group` parameter, then you are configuring preferred owners rather than possible owners.

■ **Note** Please do not follow the demonstration which removes a node from the cluster, if you wish to follow later demonstrations within this book.

Removing a Node from the Cluster

If you wish to uninstall an AlwaysOn failover cluster instance, then you cannot perform this action from Control Panel as you would a stand-alone instance. Instead, you must run the Remove Node Wizard on each of the nodes of the cluster. You can invoke this wizard by selecting Remove Node from a SQL Server Failover Cluster option from the Maintenance tab in SQL Server Installation Center.

The wizard starts by running a global rules check, followed by a rules check for removing a node. Then, on the Cluster Node Configuration page shown in Figure 7-3, you are asked to confirm the instance for which you wish to remove a node. If the cluster hosts multiple instances, you can select the appropriate instance from the drop-down box.

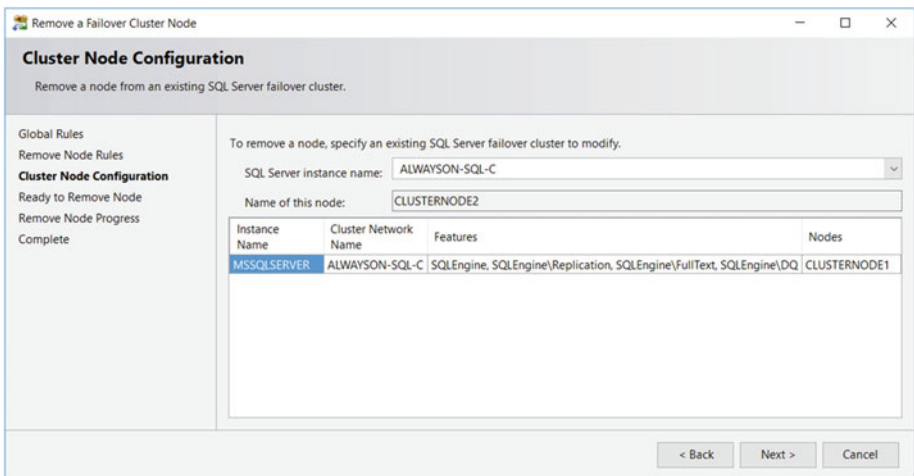


Figure 7-3. The Cluster Node Configuration page

On the Ready to Remove Node page, you are given a summary of the tasks that will be performed. After confirming the details, the instance is removed. This process should be repeated on all passive nodes, and then finally on the active node. When the instance is removed from the final node, the cluster role is also removed.

To remove a node using PowerShell, we need to run SQL Server's `setup.exe` application, with the action parameter configured as `RemoveNode`. When you use PowerShell to remove a node, the parameters in Table 7-1 are mandatory.

Table 7-1. *Mandatory Parameters When Removing a Node from a Cluster*

Parameter	Usage
/ACTION	Must be configured as <code>AddNode</code> .
/INSTANCENAME	The instance that you are adding the extra node to support.
/CONFIRMIPDEPENDENCYCHANGE	Allows multiple IP Addresses to be specified for multi-subnet clusters. Pass in a value of 1 for <code>True</code> or 0 for <code>False</code> .

The script in Listing 7-3 removes a node from our cluster when we run it from the root directory of the SQL Server installation media.

Listing 7-3. Removing a Node

```
.\setup.exe /ACTION="RemoveNode" /INSTANCENAME="ALWAYSON-SQL-C"  
/CONFIRMIPDEPENDENCYCHANGE=0 /qs
```

Managing AlwaysOn Availability Groups

Once the initial setup of your Availability Group is complete, you still need to perform administrative tasks. These include failing over the Availability Group, monitoring, and on rare occasions, adding additional listeners. These topics are discussed in the following sections.

Failover

If a replica is in Synchronous Commit mode and is configured for automatic failover, then the Availability Group automatically moves to a redundant replica in the event of an error condition being met on the primary replica. There are occasions, however, when you will want to manually fail over an Availability Group. This could be because of DR testing, proactive maintenance, or because you need to bring up an asynchronous replica following a failure of the primary replica or the primary data center.

Synchronous Failover

If you wish to fail over a replica that is in Synchronous Commit mode, launch the Failover Availability Group wizard by selecting Failover from the context menu of your Availability Group in Object Explorer. After moving past the Introduction page, you find the Select New Primary Replica page (see Figure 7-4). On this page, check the box of the replica to which you want to fail over. Before doing so, however, review the Failover Readiness column to ensure that the replicas are synchronized and that no data loss will occur.

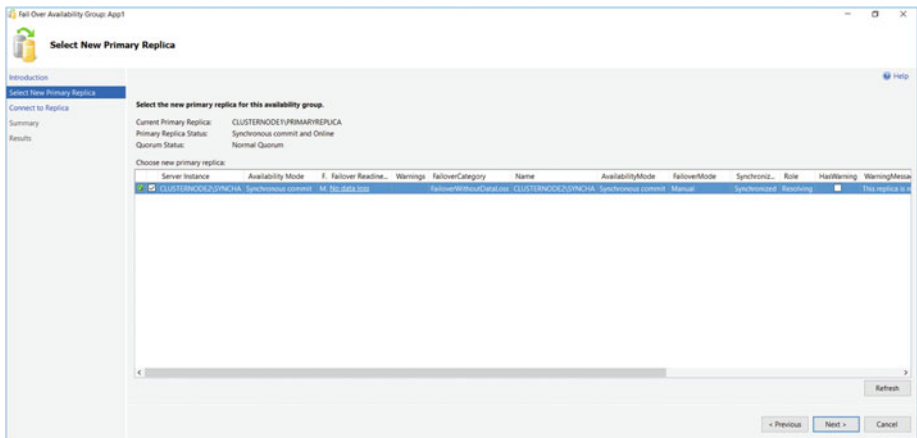


Figure 7-4. The Select New Primary Replica page

On the Connect to Replica page, illustrated in Figure 7-5, use the Connect button to establish a connection to the new primary replica.

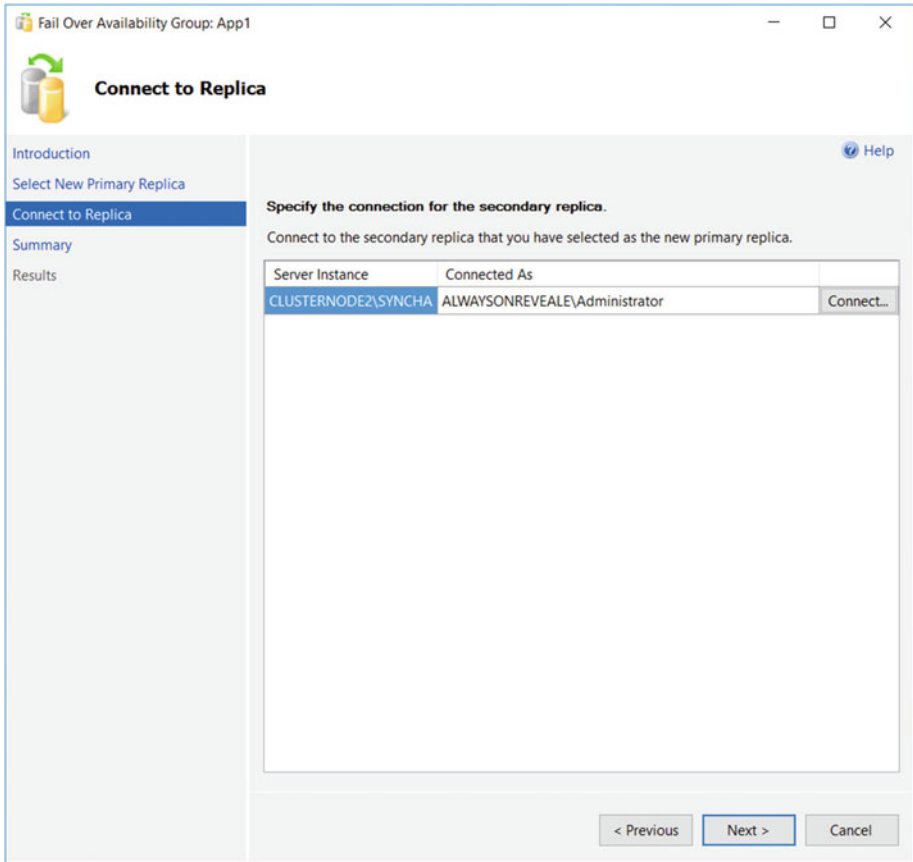


Figure 7-5. The Connect to Replica page

On the Summary page, you are given details of the task to be performed, followed by a progress indicator on the Results page. Once the failover completes, check that all tasks were successful, and investigate any errors or warnings that you receive.

We can also use T-SQL to fail over the Availability Group. The command in Listing 7-4 achieves the same results. Make sure to run this script from the replica that will be the new primary replica. If you run it from the current primary replica, use SQLCMD mode and connect to the new primary within the script.

Listing 7-4. Failing Over an Availability Group

```
ALTER AVAILABILITY GROUP App1 FAILOVER ;  
GO
```

Asynchronous Failover

If your Availability Group is in Asynchronous Commit mode, then from a technical standpoint, you can fail over in a similar way to the way you can for a replica running in Synchronous Commit mode, except for the fact that you need to force the failover, thereby accepting the risk of data loss. You can force failover by using the command in Listing 7-5. You should run this script on the instance that will be the new primary. For it to work, the cluster must have quorum. If it doesn't, then you need to force the cluster online before you force the Availability Group online.

Listing 7-5. Forcing Failover

```
ALTER AVAILABILITY GROUP App1 FORCE_FAILOVER_ALLOW_DATA_LOSS ;
```

From a process perspective, you should only ever do this if your primary site is completely unavailable. If this is not the case, first put the application into a safe state. This avoids any possibility of data loss. The way that I normally achieve this in a production environment is by performing the following steps:

1. Disable logins.
2. Change the mode of the replica to Synchronous Commit mode.
3. Fail over.
4. Change the replica back to Asynchronous Commit mode.
5. Enable the logins.

You can perform these steps with the script in Listing 7-6. When run from the DR instance, this script places the databases in App1 into a safe state before failing over, and then it reconfigures the application to work under normal operations.

Listing 7-6. Safe-Stating an Application and Failing Over

```
--DISABLE LOGINS

DECLARE @AOAGDBs TABLE
(
  DBName NVARCHAR(128)
) ;

INSERT INTO @AOAGDBs
SELECT database_name
FROM sys.availability_groups AG
INNER JOIN sys.availability_databases_cluster ADC
  ON AG.group_id = ADC.group_id
WHERE AG.name = 'App1' ;
```

```

DECLARE @Mappings TABLE
(
    LoginName NVARCHAR(128),
    DBname NVARCHAR(128),
    UserName NVARCHAR(128),
    AliasName NVARCHAR(128)
) ;

INSERT INTO @Mappings
EXEC sp_msloginmappings ;

DECLARE @SQL NVARCHAR(MAX)

SELECT DISTINCT @SQL =
(
    SELECT 'ALTER LOGIN [' + LoginName + '] DISABLE; ' AS [data()]
    FROM @Mappings M
    INNER JOIN @AOAGDBs A
        ON M.DBname = A.DBName
    WHERE LoginName <> SUSER_NAME()
    FOR XML PATH ('')
)

EXEC(@SQL)
GO

--SWITCH TO SYNCHRONOUS COMMIT MODE

ALTER AVAILABILITY GROUP App1
MODIFY REPLICA ON N'CLUSTERNODE3\ASYNCDR' WITH (AVAILABILITY_MODE =
SYNCHRONOUS_COMMIT) ;
GO

--FAIL OVER

ALTER AVAILABILITY GROUP App1 FAILOVER
GO

--SWITCH BACK TO ASYNCHRONOUS COMMIT MODE

ALTER AVAILABILITY GROUP App1
MODIFY REPLICA ON N'CLUSTERNODE3\ASYNCDR' WITH (AVAILABILITY_MODE =
ASYNCHRONOUS_COMMIT) ;
GO

```

```
--ENABLE LOGINS

DECLARE @AOAGDBs TABLE
(
  DBName NVARCHAR(128)
) ;

INSERT INTO @AOAGDBs
SELECT database_name
FROM sys.availability_groups AG
INNER JOIN sys.availability_databases_cluster ADC
    ON AG.group_id = ADC.group_id
WHERE AG.name = 'App1' ;

DECLARE @Mappings TABLE
(
  LoginName NVARCHAR(128),
  DBname NVARCHAR(128),
  Username NVARCHAR(128),
  AliasName NVARCHAR(128)
) ;

INSERT INTO @Mappings
EXEC sp_msloginmappings

DECLARE @SQL NVARCHAR(MAX)

SELECT DISTINCT @SQL =
(
  SELECT 'ALTER LOGIN [' + LoginName + '] ENABLE; ' AS [data()]
  FROM @Mappings M
  INNER JOIN @AOAGDBs A
      ON M.DBname = A.DBName
  WHERE LoginName <> SUSER_NAME()
  FOR XML PATH ('')
) ;

EXEC(@SQL)
```

Failing Over a Distributed Availability Group

Distributed Availability Groups do not support automatic failover; only manual failover is supported. When you need to fail over to a secondary Availability Group, within a Distributed Availability Group, you should perform the following steps:

- Set the synchronization mode to Synchronous Commit
- Wait for the secondary Availability Group to become synchronized

- Set the primary Availability Group to take the role of the secondary
- Force failover

The script in Listing 7-7 will force failover for the Distributed Availability Group discussed in Chapter 6.

Listing 7-7. Failover for a Distributed Availability Group

--Set the secondary Availability Group to synchronous commit mode

```
ALTER AVAILABILITY GROUP App2Distributed
MODIFY
AVAILABILITY GROUP ON
'ag1' WITH
(
    LISTENER_URL = 'tcp://App2_App2Listen:5022',
    AVAILABILITY_MODE = ASYNCHRONOUS_COMMIT,
    FAILOVER_MODE = MANUAL,
    SEEDING_MODE = MANUAL
),
'ag2' WITH
(
    LISTENER_URL = 'tcp://App2_App2Listen:5022',
    AVAILABILITY_MODE = SYNCHRONOUS_COMMIT,
    FAILOVER_MODE = MANUAL,
    SEEDING_MODE = MANUAL
);

--Wait until the Availability Groups are synchronized

WHILE (SELECT COUNT(DISTINCT synchronization_state_desc)
FROM (
    SELECT
        ag.name
        , drs.database_id
    , drs.group_id
    , drs.replica_id
    , drs.synchronization_state_desc
    , drs.end_of_log_lsn
    FROM sys.dm_hadr_database_replica_states drs
    INNER JOIN sys.availability_groups ag
        ON drs.group_id = ag.group_id
    WHERE ag.name = 'App2'
        AND synchronization_state_desc = 'synchronized'
    ) a
) > 1
BEGIN
    WAITFOR DELAY '00:00:05' ;
END
```

```
--Assign the primary Availability Group, the secondary role

ALTER AVAILABILITY GROUP App2Distributed SET (ROLE = SECONDARY) ;

--Force the failover

ALTER AVAILABILITY GROUP App2Distributed FORCE_FAILOVER_ALLOW_DATA_LOSS ;
```

Synchronizing Uncontained Objects

Regardless of the method you use to fail over, assuming that all of the databases within the Availability Group are not contained, then you need to ensure that instance-level objects are synchronized. The most straightforward way to keep your instance-level objects synchronized is by implementing an SSIS package which is scheduled to run on a periodic basis.

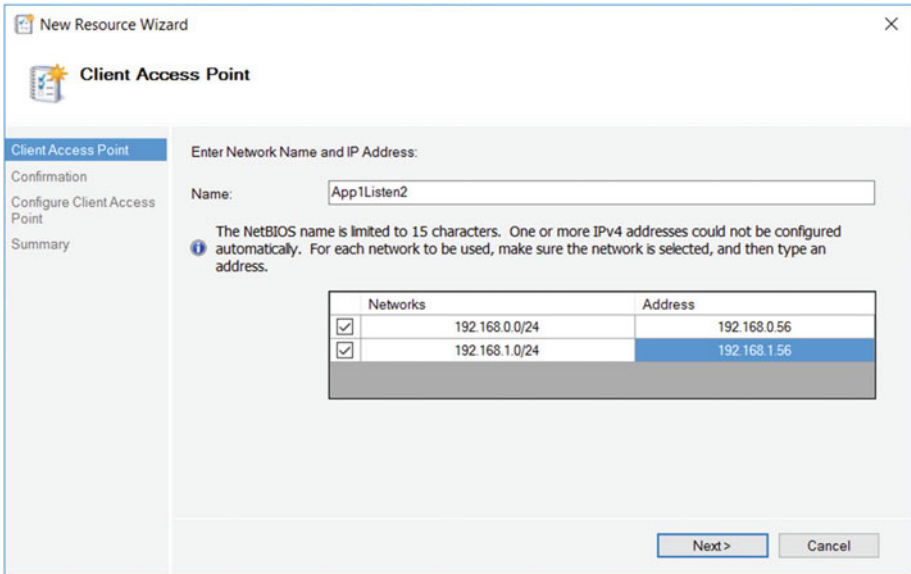
Whether you choose to schedule a SSIS package to execute, or you choose a different approach, such as a SQL Server Agent job that scripts and re-creates the objects on the secondary servers, these are the objects that you should consider synchronizing:

- Logins
- Credentials
- SQL Server Agent jobs
- Custom error messages
- Linked servers
- Server-level event notifications
- Stored procedures in Master
- Server-level triggers
- Encryption keys and certificates

Adding Multiple Listeners

Usually, each Availability Group has a single Availability Group Listener, but there are some rare instances in which you may need to create multiple listeners for the same Availability Group. One scenario in which this may be required is if you have legacy applications with hard-coded connection strings. Here, you can create an extra listener with a client access point that matches the name of the hard-coded connection string.

As mentioned earlier in this chapter, it is not possible to create a second Availability Group Listener through SQL Server Management Studio, T-SQL, or even PowerShell. Instead, we must use Failover Cluster Manager. Here, we create a new Client Access Point resource within our App1 role. To do this, we select Add Resource from the context menu of the App1 role, and then select Client Access Point. This causes the New Resource Wizard to be invoked. The Client Access Point page of the wizard is illustrated in Figure 7-6. You can see that we have entered the DNS name for the client access point and specified an IP Address from each subnet.

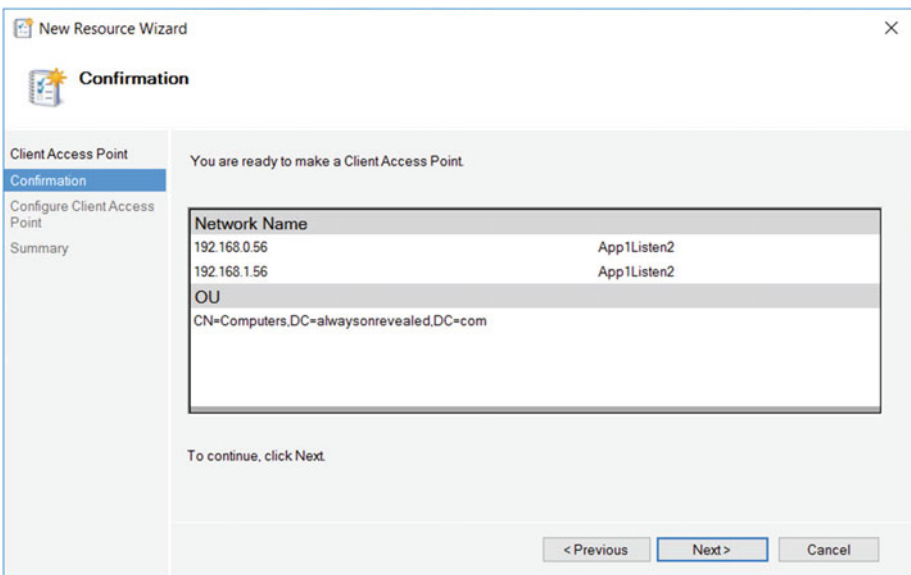


The screenshot shows the 'New Resource Wizard' window, specifically the 'Client Access Point' page. The left sidebar contains a tree view with 'Client Access Point' selected, and other options like 'Confirmation', 'Configure Client Access Point', and 'Summary'. The main area is titled 'Enter Network Name and IP Address:'. It features a 'Name:' text box containing 'App1Listen2'. Below this, an information icon (i) is followed by a note: 'The NetBIOS name is limited to 15 characters. One or more IPv4 addresses could not be configured automatically. For each network to be used, make sure the network is selected, and then type an address.' A table with two columns, 'Networks' and 'Address', is displayed. The 'Networks' column has two rows, both with a checked checkbox. The first row shows '192.168.0.0/24' and the second row shows '192.168.1.0/24'. The 'Address' column has two rows: the first row shows '192.168.0.56' and the second row shows '192.168.1.56'. At the bottom right, there are 'Next >' and 'Cancel' buttons.

Networks	Address
<input checked="" type="checkbox"/> 192.168.0.0/24	192.168.0.56
<input checked="" type="checkbox"/> 192.168.1.0/24	192.168.1.56

Figure 7-6. The Client Access Point page

On the Confirmation page, we are shown a summary of the configuration that will be performed. On the Configure Client Access Point page, we see a progress indicator, before we are finally shown a completion summary on the Summary page, which is illustrated in Figure 7-7.



The screenshot shows the 'New Resource Wizard' window, specifically the 'Confirmation' page. The left sidebar contains a tree view with 'Confirmation' selected, and other options like 'Client Access Point', 'Configure Client Access Point', and 'Summary'. The main area is titled 'You are ready to make a Client Access Point.' Below this, a table with two columns, 'Network Name' and 'Address', is displayed. The first row shows '192.168.0.56' and 'App1Listen2'. The second row shows '192.168.1.56' and 'App1Listen2'. Below the table, the text 'OU' is displayed, followed by 'CN=Computers,DC=alwaysonrevealed,DC=com'. At the bottom, the text 'To continue, click Next.' is displayed. At the bottom right, there are '< Previous', 'Next >', and 'Cancel' buttons.

Network Name	Address
192.168.0.56	App1Listen2
192.168.1.56	App1Listen2

OU
CN=Computers,DC=alwaysonrevealed,DC=com

Figure 7-7. The Confirmation page

Now we need to configure the Availability Group resource to be dependent upon the new client access point. To do this, we select Properties from the context menu of the App1 resource and then navigate to the Dependencies tab. Here, we add the new client access point as a dependency and configure an OR constraint between the two listeners, as illustrated in Figure 7-8. Once we apply this change, clients are able to connect using either of the two listener names.

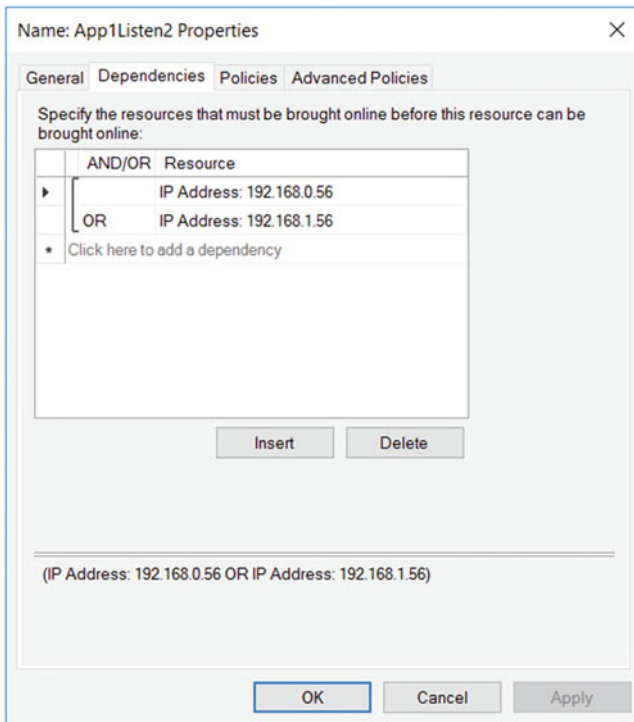


Figure 7-8. The Dependencies tab

Other Administrative Considerations

When databases are made highly available with AlwaysOn Availability Groups, several limitations are imposed. One of the most restrictive of these is that databases cannot be placed in single-user mode or be made read only. This can have an impact when you need to safe-state your application for maintenance. This is why, in the “Failover” section of this chapter, we disabled the logins that have users mapped to the databases. If you must place your database in single-user mode, then you must first remove it from the Availability Group.

A database can be removed from an Availability Group by running the command in Listing 7-8. This command removes the App1Customers database from the Availability Group.

Listing 7-8. Removing a Database from an Availability Group

```
ALTER DATABASE App1Customers SET HADR OFF ;
```

There may also be occasions in which you want a database to remain in an Availability Group, but you wish to suspend data movement to other replicas. This is usually because the Availability Group is in Synchronous Commit mode and you have a period of high utilization, where you need a performance improvement. You can suspend the data movement to a database by using the command in Listing 7-9, which suspends data movement for the App1Sales database and then resumes it.

■ **Caution** If you suspend data movement, the transaction log on the primary replica continues to grow, and you are not able to truncate it until data movement resumes and the databases are synchronized.

Listing 7-9. Suspending Data Movement

```
ALTER DATABASE App1Customers SET HADR SUSPEND ;  
GO
```

```
ALTER DATABASE App1Customers SET HADR RESUME ;  
GO
```

Another important consideration is the placement of database and log files. These files must be in the same location on each replica. This means that if you use named instances, it is a hard technical requirement that you change the default file locations for data and logs, because the default location includes the name of the instance. This is assuming, of course, that you do not use the same instance name on each node, which would defy many of the benefits of having a named instance.

Summary

Failover to a synchronous replica in the event of a failure of the primary replica is automatic. There are instances, however, in which you will also need to fail over manually. This could be because of a disaster that requires failover to the DR site, or it could be for proactive maintenance. Although it is possible to fail over to an asynchronous replica with the possibility of data loss, it is good practice to place the databases in a safe-state first. Because you cannot place a database in read-only or single-user mode, if it is participating in an Availability Group, safe-stating usually consists of disabling the logins and then switching to Synchronous Commit mode before failover.

To monitor Availability Groups throughout the enterprise, you need to use a monitoring tool, such as Systems Operation Center. If you need to monitor a small number of Availability Groups or troubleshoot a specific issue, however, use one of the tools included with SQL Server, such as a dashboard for monitoring the health of the topology, and an extended events session, called the AlwaysOn Health Trace.

One benefit of achieving high availability for SQL Server is that doing so allows you to minimize downtime during planned maintenance. On a two-node cluster, you can upgrade the passive node, fail over, and then upgrade the active node. For larger clusters, you can perform a rolling patch upgrade, which involves removing half of the nodes from the possible owners list and upgrading them. You then fail over the instance to one of the upgraded nodes and repeat the process for the remaining nodes. This mitigates the risk of mixed versions, across the possible owners.



Monitoring AlwaysOn Availability Groups

Once you have implemented Availability Groups, you need to monitor them and respond to any errors or warnings that could affect the availability of your data. If you have many Availability Groups implemented throughout the enterprise, then the only way to monitor them effectively and holistically is by using an enterprise monitoring tool, such as SOC (Systems Operations Center). If you only have a small number of Availability Groups, however, or if you are troubleshooting a specific issue, then SQL Server provides the AlwaysOn Dashboard and the AlwaysOn Health Trace. You can also create your own Extended Events sessions to monitor Availability Groups. This chapter will discuss each of these monitoring possibilities.

AlwaysOn Dashboard

The AlwaysOn Dashboard is an interactive report that allows you to view the health of your AlwaysOn environment and drill through, or roll up elements within the topology. You can invoke the report from the context menu of the Availability Groups folder in Object Explorer, or from the context menu of the Availability Group itself. Figure 8-1 shows the report that is generated from the context menu of the App1 Availability Group. You can see that currently, synchronization of both replicas is in a healthy state.

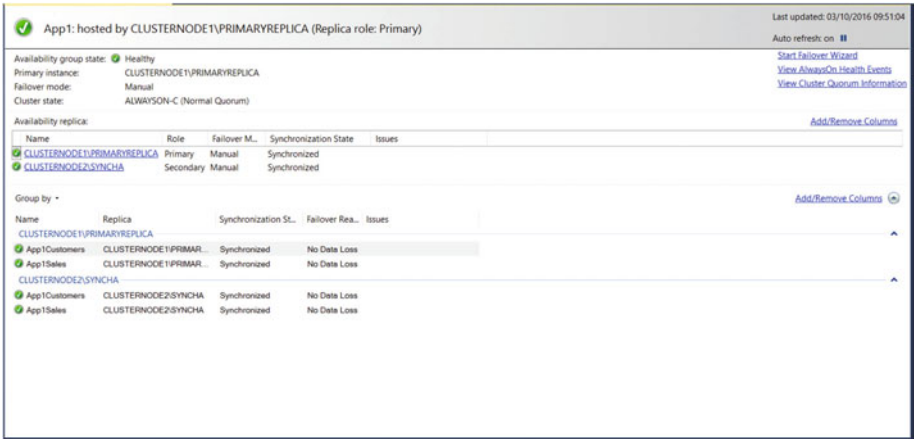


Figure 8-1. The Availability Group Dashboard

The three possible synchronization states that a database can be in are SYNCHRONIZED, SYNCHRONIZING, and NOT SYNCHRONIZING. A synchronous replica should be in the SYNCHRONIZED state, and any other state is unhealthy. An asynchronous replica, however, will never be in the SYNCHRONIZED state, and a state of SYNCHRONIZING is considered healthy. Regardless of the mode, NOT SYNCHRONIZING indicates that the replica is not connected.

Note In addition to the synchronization states, a replica also has one of the following operational states: PENDING_FAILOVER, PENDING, ONLINE, OFFLINE, FAILED, FAILED_NO_QUORUM, and NULL (when the replica is disconnected). The operational state of a replica can be viewed using the `sys.dm_hadr_availability_replica_states` DMV.

At the top right of the report, there are links to the failover wizard, which we discussed earlier in this chapter; the AlwaysOn Health events, which we discussed in the next section; and also a link to view cluster quorum information. The Cluster Quorum Information screen, which is invoked by this link, is displayed in Figure 8-2.

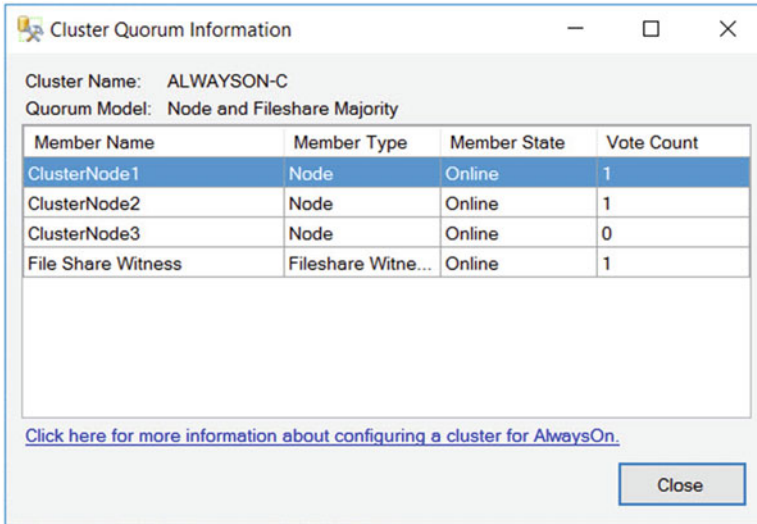


Figure 8-2. The Cluster Quorum Information screen

The Add/Remove Columns link will display a context menu, where you can dynamically add or remove columns from the display. Figure 8-3 shows that Availability Mode and Member State have been added.

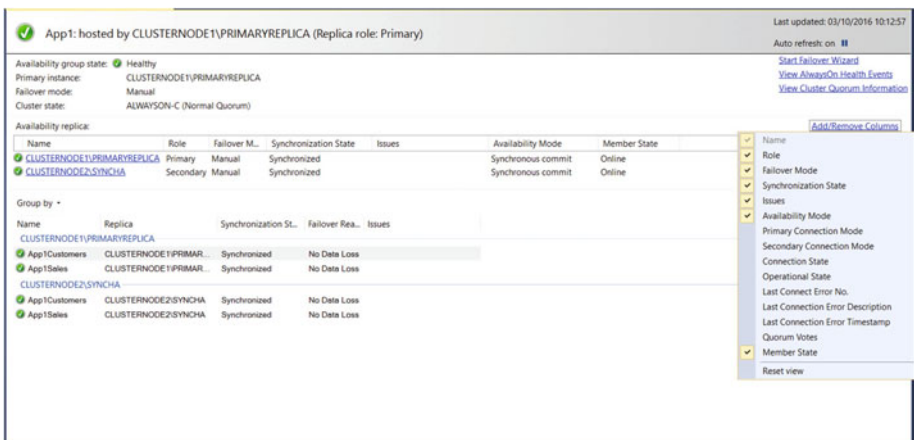


Figure 8-3. Add/Remove Columns

You can also drill through each replica in the Availability Replicas window to see replica-specific details. The Group By button will allow you to group Availability Databases by Replica, Database, Synchronization state, Failover Readiness, or Issue.

AlwaysOn Health Trace

The AlwaysOn Health Trace is an Extended Events session, which is created when you create your first Availability Group. It can be located in SQL Server Management Studio, under Extended Events | Sessions, and via its context menu, you can view live data that is being captured, or you can enter the session's properties to change the configuration of the events that are captured. It can also be accessed by using the View AlwaysOn Health Events link in the AlwaysOn Dashboard.

Drilling through the session exposes the session's package, and from the context menu of the package, you can view previously captured events. Figure 8-4 shows that the latest event captured, was Database 5 (in our case, App1Customers), was waiting for the log to be hardened on the synchronous replica.

Displaying 7139 Events	
name	timestamp
hadr_db_partner_set_sync_state	2016-09-27 05:35:55...
Event: hadr_db_partner_set_sync_state (2016-09-27 05:35:55.6387982)	
Details	
Field	Value
ag_database_id	82536CB0-3C92-4CD6-85E1-C9E7845ABE6A
commit_policy	WaitForHarden
commit_policy_...	WaitForHarden
database_id	5
group_id	94860D5A-FAFE-420D-87A2-09D74CA571C3
replica_id	680FB41E-F997-44AC-A847-0ABB3E46A705
sync_log_block	360777345306
sync_state	LOG

Figure 8-4. The target data

Right-clicking a column header in the top pane of the window will expose a context menu, which allows you to search for text or a value in a specific column, group by the values within a column, or sort the results sets by a specific column. You can also use the context menu to add or remove columns from the result set.

Monitoring AlwaysOn with Extended Events

Extended Events are a lightweight monitoring system in SQL Server, which captures events using WMI. Because the architecture uses so few system resources, they scale very well and allow you to monitor instances, with minimal impact on user activity. They are also highly configurable, giving you a wide range of options for capturing details from a very fine grain, such as page splits, to coarser-grain information, such as CPU utilization. You can also correlate Extended Events with operating system data to provide a holistic picture when troubleshooting issues. The predecessor to Extended Events was a T-SQL based tool called SQL Trace, and its GUI, which was called Profiler.

Extended Events Concepts

Extended Events have a rich architecture, which consists of Events, Targets, Actions, Predicates, Types, Maps, and Sessions. These artifacts are stored within a package, which is, in turn, stored within a module, which can be either a .dll or an executable. We discuss these concepts in the following sections.

Packages

A package is a container for the objects used within Extended Events. Here are the four types of SQL Server package:

- **Package0:** The default package, used for Extended Events system objects.
- **Sqlserver:** Used for SQL Server-related objects.
- **Sqlos:** Used for SQLOS-related objects.
- **SecAudit:** Used by SQL Audit; however, its objects are not exposed.

Events

An event is an occurrence of interest that you can trace. It may be a SQL batch completing, a cache miss, or a page split, or virtually anything else that can happen within the Database Engine, depending on the nature of the trace that you are configuring. Each event is categorized by channel and keyword (also known as category). A channel is a high-level categorization, and all events in SQL Server 2016 fall into one of the channels described in Table 8-1.

Table 8-1. *Channels*

Channel	Description
Admin	Well-known events with well-known resolutions. For example, deadlocks, server starts, CPU thresholds being exceeded, and the use of deprecated features.
Operational	Used for troubleshooting issues. For example, bad memory being detected, an Always On Availability Group replica changing its state, and a long IO being detected are all events that fall within the Operational channel.
Analytic	High-volume events that you can use for troubleshooting issues such as performance. For example, a transaction beginning, a lock being acquired, and a file read completing are all events that fall within the Analytic channel.
Debug	Used by developers to diagnose issues by returning internal data. The events in the Debug channel are subject to change in future versions of SQL Server, so you should avoid them when possible.

Keywords (or categories) are much more fine-grain. All events relating to Always On fall into the Always On and HARD categories. SQL Server exposes 122 events relating to Always On. These events are listed in Table 8-2.

Table 8-2. *Always On Events*

Event	Description
hadr_ddl_failover_execution_state	Raised when a DDL command alters the Availability Group failover state
hadr_transport_dump_message	Traces HADR transport messages throughout the system
hadr_transport_dump_config_message	Traces HADR configuration messages
hadr_transport_dump_failure_message	Traces HADR failure messages
hadr_transport_dump_preconfig_message	Traces HADR preconfig messages
hadr_transport_dump_dropped_message	Traces trace dropped HADR transport messages throughout the system
hadr_transport_session_state	Raised when a HADR transport session changes states
hadr_transport_configuration_state	Raised when session state changes
hadr_transport_ucs_registration	Raised when UCS registration state changes

(continued)

Table 8-2. *(continued)*

Event	Description
hadr_transport_ucs_connection_info	Raised when the USC connection ID associated with the AlwaysOn transport replica is registered or changes
hadr_transport_flow_control_action	Raised when a flow control action has occurred for a particular replica
hadr_database_flow_control_action	Raised when a flow control action has occurred for a particular replica
hadr_db_manager_state	Raised when the state of db_manager state
hadr_db_manager_lsn_sync_msg	Traces Log Sequence Number synchronization messages
hadr_db_manager_establish_db_msg	Raised when a DB message is established
hadr_db_manager_status_change	Traces DBReplicaStatusChange messages
hadr_db_manager_redo	Traces redo processing on secondary
hadr_db_manager_undo	Traces undo processing on secondary
hadr_db_manager_db_queue_restart	Fires in response to the queue restart hadron database
hadr_db_manager_db_startdb	Fires in response to start hadron database
hadr_db_manager_db_shutdown	Fires in response to shutdown hadron database
hadr_db_manager_user_control	Fires in response to a change in user status for an AlwaysOn controlled database
hadr_db_manager_redo_control	Traces change log scan status for an AlwaysOn controlled database
hadr_db_manager_scan_control	Traces change log scan status for an AlwaysOn controlled database
hadr_db_manager_suspend_resume	Fires in response to a change in suspend/resume status for an AlwaysOn controlled database
hadr_db_manager_db_restart	Fires in response to a restart of an AlwaysOn controlled database
hadr_worker_pool_thread	Traces AlwaysOn worker pool thread actions
hadr_worker_pool_task	Traces AlwaysOn worker pool task actions
hadr_thread_pool_worker_start	Traces AlwaysOn thread pool worker thread, start actions

(continued)

Table 8-2. (continued)

Event	Description
hadr_db_manager_page_request	Traces page Request/Response between servers
hadr_db_commit_mgr_update_harden	Fires in response to the update of the hardened Log Sequence Number for an AlwaysOn controlled database
hadr_db_commit_mgr_harden_still_waiting	Traces transaction Commit harden, still waiting for AlwaysOn Commit management
hadr_db_commit_mgr_harden	Traces transaction Commit harden result from AlwaysOn Commit management
hadr_db_commit_mgr_set_policy	Fires in response to a transaction Commit manager policy update
hadr_db_partner_set_policy	Fires in response to an AlwaysOn partner commit policy update
hadr_db_partner_set_sync_state	Fires in response to a synchronization state change of an AlwaysOn partner
hadr_apr_added_corrupted_page	Fires when auto page repair added a corrupted page
hadr_apr_repaired_page	Fires when auto page repair repaired a corrupted page
hadr_apr_skipped_page_repair	Fires when auto page repair skipped a page repair
hadr_apr_failed_page_repair	Fires when auto page repair added a corrupted page
hadr_apr_sent_repair_request_for_page	Fires when auto page repair sent a page repair request
hadr_apr_received_page_repair_request	Fires when auto page repair received a page repair request
hadr_apr_deferring_page_repair_request	Fires when auto page repair is deferring the page repair request
hadr_apr_page_repair_failed	Fires when auto page repair failed to repair page
hadr_undo_of_redo_log_scan	Traces the amount of log scanned in Undo of Redo, and the total log needing to be scanned
hadr_db_manager_filemetadata_request	Fires in response to a File Metadata Request/Response between servers

(continued)

Table 8-2. (continued)

Event	Description
hadr_capture_compressed_log_cache	Traces the hit/miss ratio for the compressed log block cache
hadr_db_manager_backup_sync_msg	Fires in response to a backup synchronization message
hadr_db_manager_backup_info_msg	Fires in response to a backup informational message
hadr_db_manager_primary_replica_file_list_msg	Fires in response to a Primary replica file list message
hadr_db_manager_seeding_request_msg	Fires in response to a seeding request message
hadr_physical_seeding_backup_state_change	Fires in response to a change in the state of a physical seeding, on the backup side
hadr_physical_seeding_restore_state_change	Fires in response to a change in the state of a physical seeding, on the restore side
hadr_physical_seeding_forwarder_state_change	Fires in response to a change in the state of a physical seeding, on the forwarder side
hadr_physical_seeding_forwarder_target_state_change	Fires in response to a change in the state of a physical seeding, on the forwarder target side
hadr_physical_seeding_submit_callback	Fires in response to a physical seeding submit callback
hadr_physical_seeding_failure	Fires in response to a physical Seeding Failure
hadr_physical_seeding_progress	Fires in response to a physical Seeding Progress
hadr_physical_seeding_schedule_long_task_failure	Fires in response to a physical Seeding Schedule Long Task Failure
hadr_automatic_seeding_start	Fires when an automatic seeding operation is submitted
hadr_automatic_seeding_state_transition	Fires when an automatic seeding operation changes state
hadr_automatic_seeding_success	Fires when an automatic seeding operation succeeds
hadr_automatic_seeding_failure	Fires when an automatic seeding operation fails
hadr_automatic_seeding_timeout	Fires when an automatic seeding operation times out

(continued)

Table 8-2. (continued)

Event	Description
hadr_filestream_file_open	Fires when AlwaysOn FileStream transport opens a file
hadr_filestream_file_close	Fires when AlwaysOn FileStream transport closes a file
hadr_filestream_log_interpreter	Fires when AlwaysOn FileStream transport finds relevant log records when interpreting log
hadr_filestream_processed_block	Fires when AlwaysOn FileStream transport has completed processing a log block
hadr_filestream_directory_create	Fires when AlwaysOn FileStream transport creates a directory
hadr_filestream_corrupt_message	Fires when AlwaysOn FileStream transport detects message corruption
hadr_filestream_message_block_end	Fires when AlwaysOn FileStream transport traces a block end message
hadr_filestream_message_dir_create	Fires when AlwaysOn FileStream transport traces a directory create message
hadr_filestream_message_file_write	Fires when AlwaysOn FileStream transport traces a file write message
hadr_filestream_file_flush	Fires when AlwaysOn FileStream transport flushes a file
hadr_filestream_file_set_eof	Fires when AlwaysOn FileStream transport sets end of a file
hadr_filestream_undo_inplace_update	Fires when AlwaysOn FileStream transport detects in-place update to undo
hadr_filestream_message_file_request	Fires when HADR FileStream transport traces a file write message
hadr_wsfc_change_notifier_status	Fires when Windows Server Failover Clustering change notifier status changes
hadr_wsfc_change_notifier_start_ag_specific_notifications	Fires when Windows Server Failover Clustering change notifier starts receiving Availability Group-specific notifications
hadr_wsfc_change_notifier_severe_error	Fires when Windows Server Failover Clustering change notifier encountered a severe error and will terminate

(continued)

Table 8-2. (continued)

Event	Description
hadr_tds_synchronizer_payload_skip	Fires when an AlwaysOn TDS Listener Synchronizer skipped a listener payload because there were no changes since the previous payload
hadr_sql_instance_to_node_map_entry_deleted	Fires at the end of an API call that deletes a SQL Server instance to cluster node map entry
hadr_wsfc_change_notifier_node_not_online	Fires when Windows Server Failover Clustering change notifier detected that the local cluster node is not online
hadr_online_availability_group_first_attempt_failure	Fires if the first attempt to bring an AlwaysOn Availability Group resource online failed
hadr_online_availability_group_retry_end	Fires when SQL Server has either exhausted all retry attempts, or Windows Server Failover Cluster has accepted the command to bring an AlwaysOn Availability Group resource online
hadr_ar_api_call	Fires when an API call is made to an Availability replica
hadr_ar_manager_starting	Fires when the Availability Group replica manager is starting
hadr_ag_wsfc_resource_state	Fires in response to a state change of an Availability Group in the Windows Server Failover Cluster
hadr_ag_database_api_call	Fires in response to an API call to an Availability Group database replica
hadr_ag_lease_renewal	Fires in response to an Availability Group Lease Renewal
hadr_ar_manager_mutex_acquisition_state	Fires in response to an Availability Replica mutex acquisition state for synchronization of Availability Group manager startup and shutdown operations
hadr_ar_critical_section_entry_state	Fires in response to an Availability Replica critical section entry state
hadr_ag_config_data_mutex_acquisition_state	Fires in response to an Availability Group mutex acquisition state

(continued)

Table 8-2. (continued)

Event	Description
hadr_database_replica_disjoin_completion	Fires when a Database Replica has been fully unjoined from the Availability Group
hadr_ar_controller_debug	Fires when a replica controller outputs a debug message
hadr_apply_log_block	Fires when a secondary is going to append a log block to the log manager
hadr_capture_log_block	Fires when the primary has captured a log block
hadr_capture_vlfheader	Fires when the primary has captured a log block which starts new virtual file
hadr_apply_vlfheader	Fires when a secondary is going to apply a Virtual Log File header
hadr_scan_state	Fires when primary or secondary Database Replica is changing state
hadr_dump_log_block	Fires when a primary sends or secondary receives a logblock message
hadr_log_block_send_complete	Fires after a log block message has been sent
hadr_dump_vlf_header	Fires when a primary sends or secondary receives a vlfheader message
hadr_dump_log_progress	Fires when a secondary sends a progress message
hadr_dump_primary_progress	Fires when a primary sends progress message
hadr_dump_sync_primary_progress	Fires when a synchronous secondary sends a progress message
hadr_send_harden_lsn_message	This event should not be used; it is for Microsoft internal testing
hadr_evaluate_readonly_routing_info	Fires when evaluating read-only routing information on a local primary database replica
hadr_db_log_throttle	Fires when a database log generation throttle changes
hadr_db_log_throttle_input	Fires when the Fabric log management component updates the log throttle

(continued)

Table 8-2. (continued)

Event	Description
hadr_db_marked_for_reseed	Fires when a secondary database falls too far behind the primary and is marked for reseed
hadr_db_log_management_configuration_parameters	Occurs when automatic log management configurations are read.
hadr_db_long_running_xact_aborted	Fires when a long-running transaction is forced to terminate by the system to avoid log becoming full
hadr_db_remote_harden_failure	Fires when a harden request, which was part of a commit or prepare, failed due to a remote failure
hadr_partner_log_send_transition	Fires in response to a log send transition between the log writer and the log capture
hadr_partner_restart_scan	Fires when a replica scans for its partner, on restart
hadr_transport_sync_send_failure	Fires when a synchronous send fails in transport
hadr_xrf_deleteAllXrf_beforeEntry	Fires immediately before all extended recovery forks are deleted
hadr_xrf_deleteRecLsn_beforeEntry	Fires immediately before the recovery Log Sequence Number is deleted from the metadata
hadr_xrf_updateXrf_partialUpdate	Fires during an updating secondary's recovery forks stack; specifically, it fires after deleting extra entries in the secondary stack, but before copying new entries from primary
hadr_xrf_updateXrf_before_recoveryLsn_update	Fires during an updating secondary's recovery forks stack; specifically, it fires after updating the stack but before saving the recovery Log Sequence Number in the metadata
hadr_xrf_copyXrf_partialCopy	Fires after deleting a secondary's stack entries, but before copying primary's entries
alwayson_ddl_executed	Fires when AlwaysOn DDL statement is executed

(continued)

Table 8-2. (continued)

Event	Description
availability_replica_state	Fires when an Availability Replica is starting or shutting down
availability_replica_state_change	Fires when the state of the Availability Replica has changed
availability_replica_manager_state_change	Fires when the state of the Availability Replica Manager has changed
availability_group_lease_expired	Fires when there is a connectivity issue between the cluster and the Availability Group, which has caused a failure to renew the lease
availability_replica_automatic_failover_validation	Fires when the failover validates the readiness of replica as a primary
availability_replica_database_fault_reporting	Fires when a database reports a fault to the availability replica manager
before_redo_lsn_update	Fires immediately before the update of the EOL Log Sequence Number
read_only_route_complete	Fires when a read-only routing operation successfully completed
read_only_route_fail	Fires when a read-only routing operation failed

Targets

A target is the consumer of the events; essentially, it is the device to which the trace data will be written. The targets available within SQL Server 2016 are detailed in [Table 8-3](#).

Table 8-3. *Targets*

Target	Synchronous/Asynchronous	Description
Event counter	Synchronous	Counts the number of events that occur during a session
Event file	Asynchronous	Writes the event output to memory buffers and then flushes them to disk
Event pairing	Asynchronous	Determines if a paired event occurs without its matching event, for example, if a statement started but never completed
ETW (Event Tracking for Windows)	Synchronous	Used to correlate Extended Events with operating system data
Histogram	Asynchronous	Counts the number of events that occur during a session, based on an action or event column
Ring buffer	Asynchronous	Stores data in a memory buffer, using First-In First-Out (FIFO) methodology

Actions

Also known as Global Fields, Actions are commands that allow additional information to be captured when an event fires. An action is fired synchronously when an event occurs and the event is unaware of the action. There are 50 actions available that allow you to capture a rich array of information, including the statement that caused the event to fire, the security context under which the statement ran, the transaction ID, the CPU ID, and the call stack.

Predicates

Predicates are filter conditions that you can apply before the system sends events to the target. It is possible to create simple predicates, such as filtering statements completing based on a database ID, but you can also create more complex predicates, such as only capturing the role change of an AlwaysOn Availability Group replica if it happens more than twice.

Predicates also fully support short-circuiting. This means that if you use multiple conditions within a predicate, then the order of predicates is important, because if the evaluation of the first predicate fails, the second predicate will not be evaluated. Because predicates are evaluated synchronously, this can have an impact on performance. Therefore, it is sensible to design your predicates so that predicates which are least likely to evaluate to true are placed before predicates that are very likely to evaluate to true.

For example, imagine that you are planning to filter on a specific database (with a database ID of 6), but this database accounts for a high percentage of the activity on the instance. You also plan to filter on a specific user ID (UserA), which is responsible for a low percentage of the activity. In this scenario, you would use the `WHERE ([sqlserver].[username]='UserA') AND ([sqlserver].[database_id]=6))` predicate to first filter out activity that does not relate to UserA, before then filtering out activity that does not relate to database ID 6.

Types

All objects within a package are assigned a type. This type is used to interpret the data stored within the byte collection of an object. Objects are assigned one of the following types:

- Action
- Event
- Pred_compare (retrieve data from events)
- Pred_source (compare data types)
- Target
- Type

Maps

A map is a dictionary that maps internal ID values to strings that DBAs can understand. Map keys are only unique within their context and are repeated between contexts. For example, within the `statement_recompile_cause` context, a `map_key` of 1 relates to a `map_value` of Schema Changed. Within the context of a `database_sql_statement` type, however, a `map_key` of 1 relates to a `map_value` of CREATE DATABASE. You can find a complete list of mappings by using the `sys.dm_xe_map_values` DMV.

Sessions

A session is essentially a trace. It can contain events from multiple packages, actions, targets, and predicates. When you start or stop a session, you are turning the trace on or off. When a session starts, events are written to memory buffers and have predicates applied before they are sent to the target. Therefore, when creating a session, you need to configure properties, such as how much memory the session can use for buffering, what events can be dropped if the session experiences memory pressure, and the maximum latency before the events are sent to the target.

Creating an Event Session to Monitor Availability Group

You can create an event session using either the New Session Wizard, the New Session Dialog Box, or via TSQL. To create a session using the New Session wizard, drill through Management | Extended Events in Object Explorer, and select New Session Wizard, from the context menu of Sessions. This will cause the Introduction page of the New Session Wizard to be displayed.

After passing through the Introduction page, you will find the Set Session Properties page, as displayed in Figure 8-5. Here, you can configure a name for the Session, and also specify if the Session should automatically be started on creation.

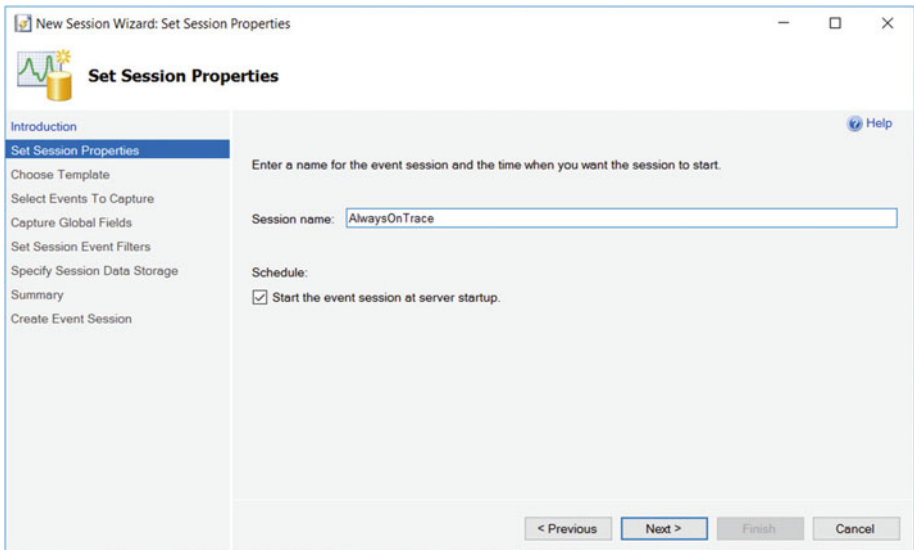


Figure 8-5. Set Properties page

On the Choose Template page of the wizard, which is illustrated in Figure 8-6, you can either select a predefined template, which will give you a starting point for commonly required sessions, or start with a blank canvas and define the entire session manually. We will choose the latter option.

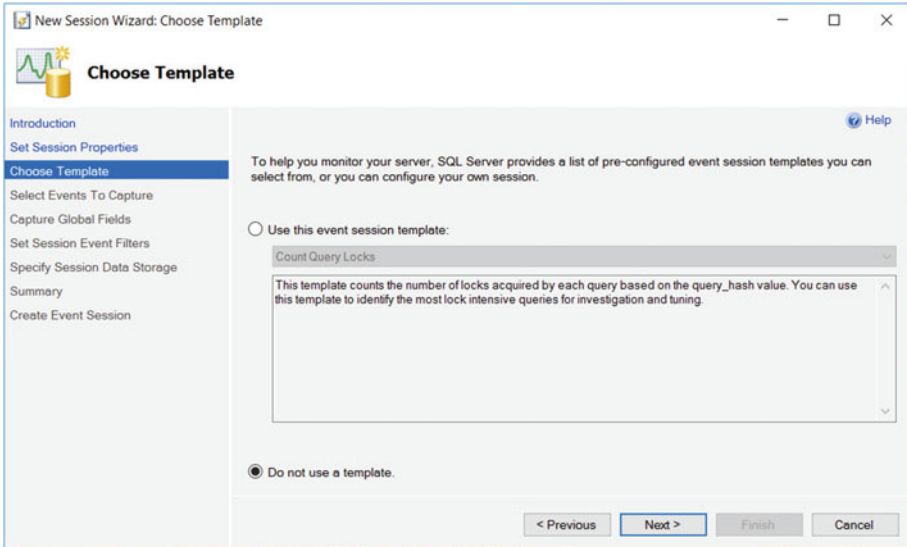


Figure 8-6. Choose Template page

Figure 8-7 shows the Select Events to Capture page. Here, we can choose what events we want to include in our session. For the purposes of this demonstration, imagine that we are frequently seeing the secondary fall behind the primary, and we are trying to determine the cause. Specifically, do we have an IO bottleneck? Because we are trying to answer a very narrow question, the choice of events to select is clear. We will need the `hadr_db_marked_for_reseed` event, to determine when the secondary falls behind, and we will need the `long_io_detected` event, so that we can correlate the times, and see if there is a pattern.

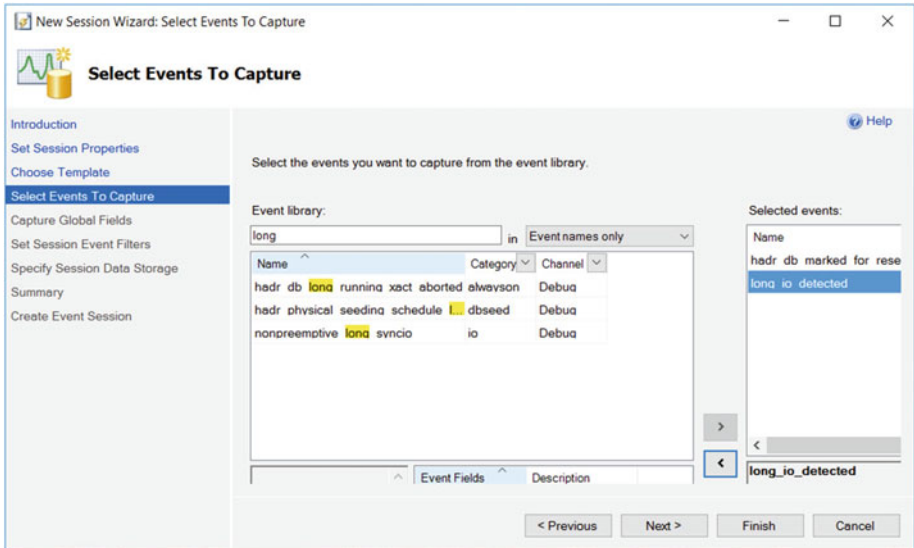


Figure 8-7. Select Events To Capture page

The Capture Global Fields page will allow us to specify any Actions that we wish to capture. In our scenario, we will capture the NT Username and SQLText actions. This will allow us to trace any long IOs back through, to see if they are caused by an inefficient query. The Capture Global Fields page is illustrated in Figure 8-8.

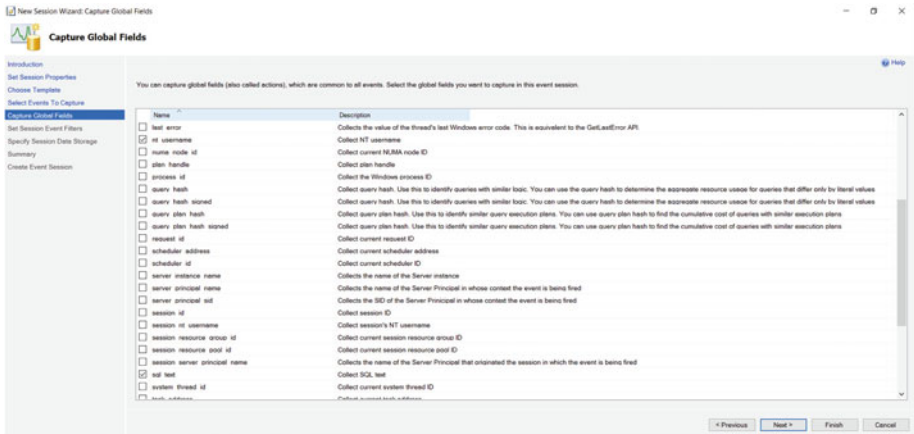


Figure 8-8. Capture Global Fields page

The Set Session Event Filters page, shown in Figure 8-9, allows you to configure Predicates on the Session. We will configure a Predicate which filters operations on system databases.

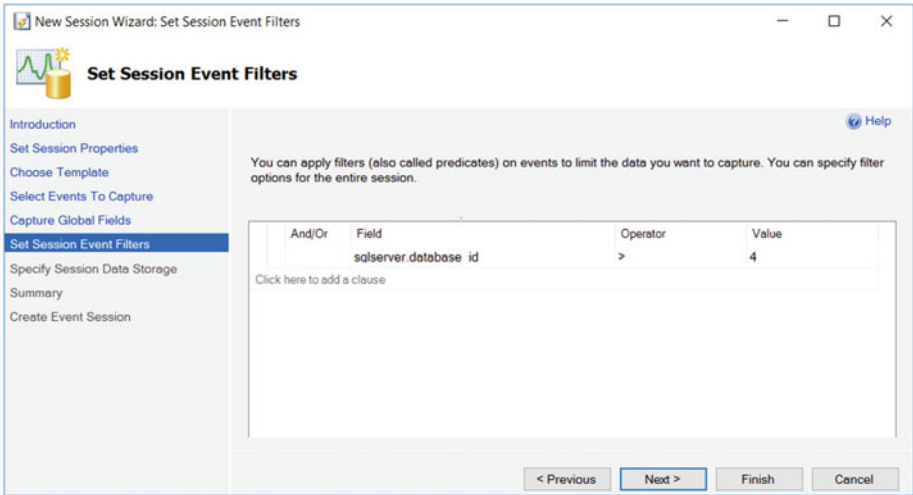


Figure 8-9. Set Session Event Filters page

The Specify Session Data Storage page of the wizard is where we can configure the Target. The wizard provides the choice of a file or ring buffer target, along with the option to specify size and rollover options. We will configure a file target, as illustrated in Figure 8-10.

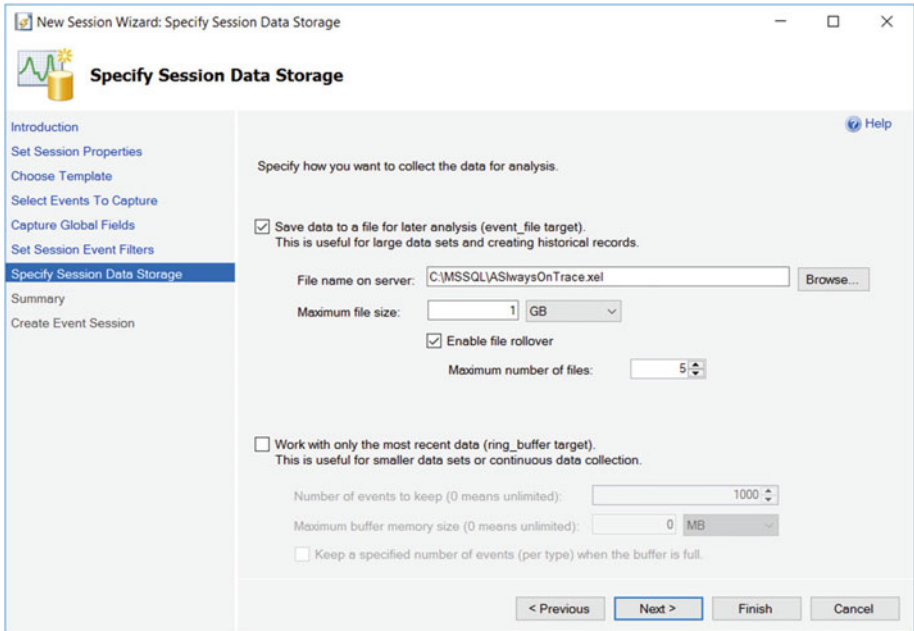


Figure 8-10. Specify Session Data Storage

The Summary page of the wizard will confirm the actions that the wizard will perform. After the Session has been created, the Completion page will provide the option of watching live data upon exit. To create the same Session using T-SQL, you could use the script in Listing 8-1.

Listing 8-1. Create an Event Session

```
CREATE EVENT SESSION AlwaysOnTrace ON SERVER
ADD EVENT sqlserver.hadr_db_marked_for_reseed(
    ACTION(sqlserver.nt_username,sqlserver.sql_text)
    WHERE (sqlserver.database_id>(4))),
ADD EVENT sqlserver.long_io_detected(
    ACTION(sqlserver.nt_username,sqlserver.sql_text)
    WHERE (sqlserver.database_id>(4)))
ADD TARGET package0.event_file(SET filename='C:\MSSQL\ASlwaysOnTrace.xel')
WITH (MAX_MEMORY=4096 KB,EVENT_RETENTION_MODE=ALLOW_SINGLE_EVENT_LOSS,MAX_
DISPATCH_LATENCY=30 SECONDS,MAX_EVENT_SIZE=0 KB,MEMORY_PARTITION_
MODE=NONE,TRACK_CAUSALITY=OFF,STARTUP_STATE=ON) ;
```

The CREATE EVENT SESSION DDL statement accepts the arguments detailed in Table 8-4.

Table 8-4. *CREATE EVENT SESSION Arguments*

Argument	Description
event_session_name	Specifies the name of the event session that you are creating
ADD EVENT SET	Repeating for every event that is added to the session, followed by the name of the event, in the format package.event; you can use the SET statement to set event-specific customizations, such as including nonmandatory event fields
ACTION	Specified after each ADD EVENT argument if there are global fields which should be captured for that event
WHERE	Specified after each ADD EVENT argument if the event should have a predicate associated with it
ADD TARGET SET	Specified for each target that will be added to the session; you can use the SET statement to populate target-specific parameters, such as the filename parameter for the event_file target

The CREATE EVENT SESSION statement also accepts a number of WITH options, which are detailed in Table 8-5.

Table 8-5. *CREATE EVENT SESSION WITH Options*

WITH Option	Description
MAX_MEMORY	Specifies the maximum amount of memory that the event session can use for buffering events before dispatching them to the target(s)
EVENT_RETENTION_MODE	Specifies the behavior if the buffers become full: acceptable values are ALLOW_SINGLE_EVENT_LOSS, which indicates that a single event can be dropped if all buffers are full; ALLOW_MULTIPLE_EVENT_LOSS, which indicates that an entire buffer can be dropped if all buffers are full; and NO_EVENT_LOSS, which indicates that tasks that cause events to fire are to wait until there is space in the buffer
MAX_DISPATCH_LATENCY	Specifies the maximum amount of time that events can reside in the sessions buffers before being flushed to the target(s), specified in seconds
MAX_EVENT_SIZE	Specifies the maximum possible size for event data from any single event; it can be specified in kilobytes or megabytes and should only be configured to allow events that are larger than the MAX_MEMORY setting

(continued)

Table 8-5. (continued)

WITH Option	Description
MEMORY_PARTITION_MODE	Specifies where event buffers are created; acceptable values are as follows: <ul style="list-style-type: none"> • NONE - which indicates that the buffers will be created within the instance • PER_NODE - which indicates that the buffers will be created for each NUMA node • PER_CPU - which means that buffers will be created for each CPU
TRACK_CAUSALITY	Specifies that an additional GUID and sequence number will be stored with each event so that events can be correlated
STARTUP_STATE	<ul style="list-style-type: none"> • Specifies if the session automatically starts when the instance starts; ON indicates it does • OFF indicates it does not

■ **Tip** For a deeper discussion around Extended Events, I highly recommend the Apress book *Pro SQL Server Administration*, which can be purchased from www.apress.com/9781484207116.

Summary

SQL Server provides rich tools for monitoring the health of AlwaysOn Availability Groups. The AlwaysOn Dashboard is an interactive report, within SQL Server Management Studio, which will allow you to assess the health of your Availability Groups and Replicas. It also provides links to view quorum configuration information and live health data.

Live Health Data is captured by an extended events session, which is created when you create the first Availability Group on an instance, and runs in the background, capturing preconfigured events. It is possible to customize this trace; I would recommend leaving it with default configurations and creating a new Event Session, if you require a custom capture.

Creating an Event Session allows you to capture either very fine-grain points of interest, or just coarser-grain information, depending on your requirements. Extended Events are implemented using WMI, and are a very lightweight framework, meaning you can identify issues and trend, without compromising the performance of your instance.



Troubleshooting AlwaysOn

SQL Server exposes a wealth of metadata, pertaining to high availability and disaster recovery objects, especially around the AlwaysOn feature set. This metadata can be used to quickly identify a configuration, find the root cause of an issue, or script automated responses to events that may occur. The following sections will discuss the metadata that is available, and provide examples of how it can be used.

AlwaysOn Failover Clustered Instance Metadata

From inside the database engine, it is possible to view a large amount of metadata regarding a clustered instance, and the Windows Cluster that hosts it. This information can prove invaluable to a DBA. The following section will introduce some of the most useful and interesting metadata objects.

Discovering the Node That Hosts an Instance

Naturally, a DBA will need to know which node within a cluster is hosting a failover clustered instance, especially when attempting to diagnose connectivity or performance issues. If your organization has a policy that DBAs are not allowed operating system access, however, then Failover Manager can't be used. Luckily, there is a DMV (Dynamic Management View) within SQL Server that will expose this information. The `sys.dm_os_cluster_nodes` DMV will return the columns detailed in Table 9-1.

Table 9-1. *sys.dm_os_cluster_nodes* Columns

Column	Description
NodeName	The name of the cluster node
Status	The current status of the node. Possible values are <ul style="list-style-type: none"> • 0 - Indicates the node is up • 1 - Indicates the node is down • 2 - Indicates the node is paused • 3 - Indicates the node is currently joining the cluster • 4 - Indicates that the status is unknown
status_description	A textual description of the status. Possible values are <ul style="list-style-type: none"> • Up • Down • Paused • Joining • Unknown
is_current_owner	Indicates if the instance is currently hosted by the node. Possible values are <ul style="list-style-type: none"> • 0 - Indicates the node does not own the instance • 1 - Indicates that the node does own the instance

The query in Listing 9-1 will return the name of the cluster node that currently hosts the instance.

Listing 9-1. Discover the Node That Hosts the Instance

```
SELECT NodeName
FROM sys.dm_os_cluster_nodes
WHERE is_current_owner = 1 ;
```

Viewing Health Check Configuration

If assisting the Windows administration team with the repeated failover of a clustered instance, a DBA may wish to expose details of what conditions can cause a failover, to ensure that an appropriate level is configured. This can be achieved by using the `sys.dm_os_cluster_properties` DMV, which returns the columns detailed in Table 9-2.

Table 9-2. *sys.dm_os_cluster_properties* Columns

Column	Description
VerboseLogging	Indicates the logging level used by the cluster. Possible values are <ul style="list-style-type: none"> • 0 - Indicates that logging is turned off • 1 - Indicates that only errors are logged • 2 - Indicates that errors and warning are logged
SQLDumperDumpFlags	Specifies the type of dump file that SQLDumper will generate. Possible values are <ul style="list-style-type: none"> • 0x0120 - Indicates a Minidump • 0x0110 - Indicates a Full Dump • 0x8100 - Indicates a Filtered Dump
SQLDumperDumpPath	Specifies the file path where SQLDumper will output the dump files
SQLDumperDumpTimeOut	The timeout value for SQLDumper, when creating a dump file. Specified in milliseconds
FailureConditionLevel	The level of failure that will cause a failover to occur. A full description of failure condition levels can be found in Table 9-3.
HealthCheckTimeout	The duration that the database engine will wait for health information to be returned, before it will decide that the instance is unresponsive

The possible failure condition levels, returned by the `FailureConditionLevel` column are detailed in Table 9-3.

Table 9-3. *Failure Condition Levels*

Condition Level	Description
0	Automatic failover does not occur
1	Automatic failover occurs when the SQL Server service is down
2	Automatic failover will occur when <ul style="list-style-type: none"> • Level 1 conditions are met • The HealthCheckTimeout value is exceeded
3	Automatic failover will occur when <ul style="list-style-type: none"> • Level 2 conditions are met • The health check returns System Error
4	Automatic failover will occur when <ul style="list-style-type: none"> • Level 3 conditions are met • The health check returns Resource Error
5	Automatic failover will occur when <ul style="list-style-type: none"> • Level 4 conditions are met • The health check returns Query_Processing_Error

The query in Listing 9-2 will return the current failover condition level and the current health check timeout value.

Listing 9-2. Return Health Check Configuration

```

SELECT
    FailureConditionLevel
    , HealthCheckTimeout
FROM sys.dm_os_cluster_properties ;

```

The current health of the instance can be determined manually by using the `sp_server_diagnostics` system stored procedure. The procedure accepts a single parameter, `@repeat_interval`, which specifies how often the procedure should return results, specified in seconds. If the parameter is omitted, then results will only be returned once. If a value is passed for the parameter, then it must be greater than 5. The procedure returns the result set detailed in Table 9-4.

Table 9-4. Columns Returned by *sp_server_diagnostics*

Column	Description
creation_time	Indicates the time that the row was created
component_type	Indicates the type of component. Possible values are <ul style="list-style-type: none"> • Instance • AlwaysOn: Availability Group
component_name	Indicates the name of the component. Possible values are <ul style="list-style-type: none"> • system • resource • query_processing • io_subsystem • events • [Availability Group name]
State	The health status of the component. Possible values are <ul style="list-style-type: none"> • 0 - Indicates that the state is unknown • 1 - Indicates that the state is clean (meaning healthy) • 2 - Indicates that there are warnings • 3 - Indicates that there are errors
state_desc	A textual description of the component's state. Possible values are <ul style="list-style-type: none"> • Unknown • Clean • Warnings • Errors
Data	An XML representation of component specific data. For example, the resource component includes element specifying the available physical and available virtual memory. It also includes attributes including a count of out-of-memory exceptions.

The script in Listing 9-3 will return the complete result set of the *sp_server_diagnostics* system stored procedure, alongside values which have been shredded from the XML column, to provide a quick view of the overall server CPU utilization, the CPU utilization of the instance, and a count of any out-of-memory exceptions that may have occurred.

■ **Tip** A discussion around shredding XML is beyond the scope of this book. However, I recommend the Apress title *Expert Scripting and Automation for SQL Server DBAs*, where a discussion around working with XML for administrative purposes can be found. The book can be purchased at www.apress.com/9781484219423.

Listing 9-3. Retrieving Diagnostic Information

```
CREATE TABLE ##Server_Diagnostics
(
  creation_time    DATETIME,
  component_type   NVARCHAR(8),
  component_name   NVARCHAR(128),
  [state]          TINYINT,
  state_desc       NVARCHAR(8),
  [data]           XML
) ;

INSERT INTO ##Server_Diagnostics
EXEC sp_server_diagnostics ;

SELECT *,
data.value('/system/@systemCpuUtilization)[1]', 'int') AS
SystemCPUUtilization
, data.value('/system/@sqlCpuUtilization)[1]', 'int') AS SQLServerCPU
, data.value('/resource/@outOfMemoryExceptions)[1]', 'int') AS
OutOfMemoryExceptions
FROM ##Server_Diagnostics ;

DROP TABLE ##Server_Diagnostics ;
```

AlwaysOn Availability Group Metadata

Metadata can also be used to troubleshoot issues with Availability Groups. The following sections will discuss some of the most useful and interesting metadata objects, relating the Availability Groups.

Determining the Last Failover Reason

If an Availability Group has failed over, the first questions you are likely to want to answer are “when?” and “why?” These questions can be answered using the `sys.dm_hadr_availability_replica_states` DMV. This object returns the columns detailed in Table 9-5.

Table 9-5. *sys.dm_hadr_availability_replica_states*

Column	Description
<code>replica_id</code>	The GUID of the Replica
<code>group_id</code>	The GUID of the Availability Group
<code>is_local</code>	Indicates if the Replica is local or remote. Possible values are <ul style="list-style-type: none"> • 0 - Indicates a remote secondary • 1 - Indicates a local Replica
<code>role</code>	Indicates the role that is currently assigned to the Replica. Possible values are <ul style="list-style-type: none"> • 0 - Indicates that the role is currently being resolved • 1 - Indicates that the Replica has the Primary role • 2 - Indicates that the Replica currently has the Secondary role
<code>role_desc</code>	A textual description of the Replica's current role. Possible values are <ul style="list-style-type: none"> • RESOLVING • PRIMARY • SECONDARY
<code>operational_state</code>	Indicates the current operational state of the replica. Possible values are <ul style="list-style-type: none"> • 0 - Indicates a failover is pending • 1 - Indicates the state is pending • 2 - Indicates online • 3 - Indicates offline • 4 - Indicates failed • 5 - Indicates failed, with no quorum • NULL - Indicates the Replica is not local
<code>operational_state_desc</code>	A textual description of the operational state. Possible values are <ul style="list-style-type: none"> • PENDING_FAILOVER • PENDING • ONLINE • OFFLINE • FAILED • FAILED_NO_QUORUM • NULL

(continued)

Table 9-5. (continued)

Column	Description
connected_state	Indicates if a Secondary Replica is currently connected to the Primary Replica. Possible values are <ul style="list-style-type: none"> • 0 - Indicates the Replica is disconnected from the Primary • 1 - Indicates that the Replica is connected to the Primary
connected_state_desc	A textual description of the connected state. Possible values are <ul style="list-style-type: none"> • DISCONNECTED • CONNECTED
recovery_health	Indicates if databases within the Availability Group are online. Possible values are <ul style="list-style-type: none"> • 0 - Indicating at least one of the databases is not online • 1 - Indicates that all of the databases are online • NULL - Indicates the Availability Group is not local
recovery_health_desc	A textual description of the recovery_health. Possible values are <ul style="list-style-type: none"> • ONLINE_IN_PROGRESS • ONLINE • NULL
synchronization_health	Indicates the synchronization state of the Availability Group's databases. Possible values are <ul style="list-style-type: none"> • 0 - Indicates that at least one database is in the NOT SYNCHRONIZING state. This is known as Not Healthy • 1 - Indicates that at least one database is not in the ideal synchronization state. This is known as Partially Healthy. The ideal state will be <ul style="list-style-type: none"> • SYNCHRONIZED - For Synchronous Commit Replicas • SYNCHRONIZING - For Asynchronous Commit Replicas • 2 - Indicates that all databases are in the ideal state. This is known as Healthy
synchronization_health_desc	A textual description of the synchronization health state. Possible values are <ul style="list-style-type: none"> • NOT_HEALTHY • PARTIALLY_HEALTHY • HEALTHY

(continued)

Table 9-5. (continued)

Column	Description
last_connect_error_number	The error number of the last connection error
last_connect_error_description	The description of the last connection error
last_connect_error_timestamp	The date and time of the last connection error

The query in Listing 9-4 demonstrates how to return the time and reason of the last connection error. This will indicate when and why failover occurred. One row will be returned for each combination of Replica and Availability Group. You will notice that we join the `sys.dm_hadr_availability_replica_states` DMV to the `sys.availability_replicas` and `sys.availability_groups` DMVs, to retrieve the names of the nodes that host the Replicas, and the names of the Availability Groups.

Listing 9-4. Determine Last Failover Time and Reason

```

SELECT
    ar.replica_server_name
    ,ag.name
    ,ars.last_connect_error_description
    ,ars.last_connect_error_timestamp
FROM sys.dm_hadr_availability_replica_states ars
INNER JOIN sys.availability_replicas ar
    ON ar.group_id = ars.group_id
    AND ars.replica_id = ar.replica_id
INNER JOIN sys.availability_groups ag
    ON ag.group_id = ar.group_id ;

```

Assessing the State of Availability Databases

You may have noticed that the `sys.dm_hadr_availability_replica_states` DMV will provide details of Availability Groups that contain databases that are not in a healthy state. The results are not granular enough, however, for you to discover which databases are not healthy. This information can be retrieved from the `sys.dm_hadr_database_replica_states` DMV, which returns the columns detailed in Table 9-6.

Table 9-6. *sys.dm_hadr_database_replica_states* Columns

Column	Description
database_id	The ID of the database
group_id	The Availability Group GUID
replica_id	The Availability Replica GUID
group_database_id	The ID of the database, within the Availability Group
is_local	Indicates if the database is local or remote. Possible values are <ul style="list-style-type: none"> • 0 - Indicates that the database is not local to the instance • 1 - Indicates that the database is local to the instance
is_primary_replica	Indicates if the database replica currently has the role of primary or secondary. Possible values are <ul style="list-style-type: none"> • 0 - Indicates a Secondary Database Replica • 1 - Indicates a Primary Database Replica
synchronization_state	Indicates the state of the database synchronization. Possible values are <ul style="list-style-type: none"> • 0 - Indicates Not Synchronizing • 1 - Indicates Synchronizing • 2 - Indicates Synchronized • 3 - Indicates that the state is Reverting. This means that the secondary is at the part of the undo stage, where it is retrieving pages from the primary • 4 - Indicates that the state is Initializing. This means that the secondary is at the part of the undo phase, where required log records are currently being shipped and hardened
synchronization_state_desc	A textual description of the synchronization state. Possible values are <ul style="list-style-type: none"> • NOT SYNCHRONIZING • SYNCHRONIZING • SYNCHRONIZED • REVERTING • INITIALIZING

(continued)

Table 9-6. *(continued)*

Column	Description
<code>is_commit_participant</code>	<p>Indicates if transaction commits are synchronized. Databases on asynchronous replicas will always report 0 and the value is only accurate for databases on synchronous replicas, for the primary database. Possible values are</p> <ul style="list-style-type: none"> • 0 - Indicates that transaction commit is not synchronized • 1 - Indicates that transaction commit is synchronized
<code>synchronization_health</code>	<p>Indicates the synchronization state of the database. Possible values are</p> <ul style="list-style-type: none"> • 0 - Indicates Not Healthy. This means that the database is not synchronizing • 1 - Indicates Partially Healthy. This means that the database is synchronizing • 2 - Indicates Healthy. This means that the database is synchronized
<code>synchronization_health_desc</code>	<p>A textual description of the synchronization health. Possible values are</p> <ul style="list-style-type: none"> • NOT_HEALTHY • PARTIALLY_HEALTHY • HEALTHY
<code>database_state</code>	<p>Indicates that current state of the database. The value reflects the value in the <code>sys.databases</code> catalog view. Possible values are</p> <ul style="list-style-type: none"> • 0 - Indicates that the database is Online • 1 - Indicates that the database is Restoring • 2 - Indicates that the database is Recovering • 3 - Indicates that the database has a state of Recovery pending • 4 - Indicates that the database is Suspect • 5 - Indicates that the database is in Emergency mode • 6 - Indicates that the database is Offline

(continued)

Table 9-6. (continued)

Column	Description
database_state_desc	<p>A textual description of the database state. Possible values are</p> <ul style="list-style-type: none"> • ONLINE • RESTORING • RECOVERING • RECOVERY_PENDING • SUSPECT • EMERGENCY • OFFLINE
is_suspended	<p>Indicates if the database is suspended. Possible values are</p> <ul style="list-style-type: none"> • 0 - Indicates resumed • 1 - Indicates suspended
suspend_reason	<p>If the database is suspended, the suspend_reason column indicates the reason. Possible values are</p> <ul style="list-style-type: none"> • 0 - Indicates a user manually suspended the data movement • 1 - Indicates a suspension following a forced failover • 2 - Indicates that an error occurred during the redo phase • 3 - Indicates that there was an error during the log capture • 4 - Indicates that there was an error when writing the log • 5 - Indicates the database was suspended prior to a restart • 6 - Indicates that there was an error during the undo phase • 7 - Indicates a log chain mismatch error • 8 - Indicates that there was an error in the calculation of the secondary replica's synchronization point

(continued)

Table 9-6. (continued)

Column	Description
suspend_reason_desc	A textual description of the suspend_reason column. Possible values are <ul style="list-style-type: none"> • SUSPEND_FROM_USER • SUSPEND_FROM_PARTNER • SUSPEND_FROM_REDO • SUSPEND_FROM_CAPTURE • SUSPEND_FROM_APPLY • SUSPEND_FROM_RESTART • SUSPEND_FROM_UNDO • SUSPEND_FROM_REVALIDATION • SUSPEND_FROM_XRF_UPDATE
recovery_lsn	On the primary replica, recovery_lsn indicates the end of the transaction log (the final point in the transaction log for point-in-time recovery). On a secondary replica, the column indicates the point to which the resynchronization would be required. If the value is equal or greater than last_hardened_lsn, however, then it indicates that resynchronization would not be required
truncation_lsn	For a primary replica, the column indicates the minimum log truncation LSN across all secondaries. For a secondary replica, the column indicates the log truncation point for that specific database replica
last_sent_lsn	Indicates the end of the last log block that has been sent
last_sent_time	The date and time that the last log block was sent
last_recieved_lsn	Indicates the end of the last log block to be received
last_hardened_lsn	Indicates the start of the last log block to be hardened. The value will be NULL for Asynchronous Commit replicas
last_hardened_time	The date and time of the hardened LSN
last_redone_lsn	The LSN of the last log record to be redone on the secondary
last_redone_time	The timestamp of the last LSN to be redone on the secondary
log_send_queue_size	The size of the log records that have not yet been sent to the secondary, specified in kilobytes

(continued)

Column	Description
log_send_rate	The speed at which log records are being sent to the secondary, specified in kilobytes/sec
filestream_send_rate	The speed at which FILESTREAM files are being sent to the secondary, specified in kilobytes/sec
end_of_log_lsn	The LSN of the final log record within the log cache
last_commit_lsn	The LSN of the last committed transaction in the transaction log
last_commit_time	The timestamp of the last committed LSN in the transaction log
low_water_mark_for_ghosts	The ghost cleanup task (which physically deletes rows that have already been logically deleted), uses the minimum value of this column, across all replicas of the database, to determine where to start cleaning up records
secondary_log_seconds	The number of seconds that the secondary replica is behind the primary replica

The script in Listing 9-5 demonstrates how to assess the health availability database, within the App1 Availability Group. You will notice that we use the `DB_NAME()` function to return the name of the database, and join the `sys.dm_hadr_database_replica_states` DMV to the `sys.availability_groups` and `sys.availability_replicas` catalog views, to return the names of the Availability Groups, and Replicas.

Listing 9-5. Assessing the State of an Availability Database

```

SELECT
    DB_NAME(database_id)
    ,ag.name
    ,ar.replica_server_name
    ,is_primary_replica
    ,synchronization_state_desc
    ,synchronization_health_desc
    ,database_state_desc
FROM sys.dm_hadr_database_replica_states drs
INNER JOIN sys.availability_groups ag
    ON drs.group_id = ag.group_id
INNER JOIN sys.availability_replicas ar
    ON drs.replica_id = ar.replica_id
WHERE ag.name = 'App1';

```

Summary

SQL Server exposes a large amount of metadata that can help you troubleshoot issues and audit your configuration. While this chapter discusses some of the most helpful metadata, I strongly encourage you to explore the `hadr` DMVs further.

For failover clustered instances, the `sys.dm_os_cluster_nodes` DMV exposes the health status of nodes within a cluster. Further troubleshooting detail can be found by calling the `sp_server_diagnostics` system stored procedure.

■ **Tip** The `sp_server_diagnostics` system stored procedure can also be used for troubleshooting AlwaysOn Availability Groups, and returns a row for each Availability Group hosted on the instance.

The `sys.dm_hadr_availability_replica_states` DMV exposes details of the health status of Replicas, which host AlwaysOn Availability Groups. To drill down to view the health status of databases that participate within an Availability Group, the `sys.dm_hadr_database_replica_states` DMV can be queried.

Both of the DMVs mentioned in the preceding can be joined to the `sys.availability_groups` and `sys.availability_replicas` DMVs, to obtain the textual information regarding the configuration, as opposed to GUIDs. The `sys.dm_hadr_database_replica_states` DMV can also be joined to `sys.databases`, to obtain further information, regarding database configuration.

Index

■ A

- Active node, 10
- Advanced Encryption Standard (AES), 95
- AlwaysOn administration
 - Availability Group (*see* AlwaysOn Availability Groups)
 - cluster maintenance, 149
 - Cluster Node Configuration page, 153
 - configuring possible owners, 152
 - Failover Cluster Manager, 150
 - move clustered role dialog box, 150
 - move instance between nodes, 149, 151
 - overriding priority, 149
 - PowerShell, 151, 154
 - Remove Node wizard, 153–154
 - remove possible owners, 151–152
 - rolling patch upgrade, 151–152
 - Software Assurance, 149
- AlwaysOn Availability Groups (AOAG)
 - active/passive cluster, 26–27
 - App1 and App2, 83
 - App1Customers and App1Sales databases, 121
 - asynchronous failover, 157
 - automatic page repair, 22
 - backing up database, 90, 27
 - CLUSTERNODE3\ASYNCADR, 26–27, 121
 - creation
 - Application Intent parameter, 94
 - arguments, 106–108
 - Backup Preferences tab, 96
 - database mirroring endpoint, 95
 - database page, 92–93
 - data synchronizaton page, 98–99
 - Endpoints tab, 95
 - FAILOVER_CONDITION_LEVEL argument, 108
 - introduction page, 90–91
 - Listener dialog box, 100
 - Listener tab, 97–98
 - multi-subnet clusters, 97
 - network traffic, 97
 - replicas page, 93–94
 - RTO, 97
 - script, 101
 - service account, 95
 - summary page, 100
 - Synchronous Commit option, 93–94
 - validation page, 99–100
 - database and log files, 164
 - database creation, 84
 - clustering technologies, 18
 - specify name page, 91–92
 - data-tier applications, 19–20
 - disaster recovery (*see* Disaster recovery)
 - HA/DR topology, 25, 83
 - High Availability tab, 89
 - Last Failover Reason
 - return the time and reason, 199
 - sys.dm_hadr_availability_replica_states, 197–199
 - listener dialog box
 - App2Customers database, 109
 - backing up and restoring database, 109
 - Backup Preferences tab, 111
 - general tab, 110

AlwaysOn Availability Groups (*cont.*)

- IP Addresses, 112
- Primary Role property, 110
- replica properties, 110
- Session Timeout property, 110
- TCP endpoint, 109
- transaction log backup, 109
- load balancing, 19
- log stream replication, 26
- monitoring tool
 - AlwaysOn Dashboard, 167
 - AlwaysOn Health Trace, 170
- multiple listeners
 - client access page point, 161–162
 - confirmation page, 162
 - dependencies tab, 163
 - hard-coded connection strings, 161
- multi-subnet cluster, 21, 121
- performance Benchmark, 117
- PRIMARYREPLICA, 83
- production environment steps, 157
- quorum model, 121
- readable secondary replicas, 144
- remove database, 163–164
- safe-stating application and
 - failing over, 157, 159
- scale-out requirements, 19
- shared disk resource, 25
- single connection string, 19
- single-user mode, 163
- SQL Server configuration, 89
- stand-alone instances, 89
- State of Availability Databases
 - assessing health availability database, 204
 - DB_NAME() function, 204
 - sys.dm_hadr_database_replica_states Columns, 200–203
- suspend data movement, 164
- SYNCHA, 83
- Synchronous Commit mode, 112
- synchronous failover
 - introduction page, 155
 - Primary Replica page, 155
 - Replica page, 155–156
 - summary page, 156
- synchronous replicas, 18
- tasks, 83, 121
- uncontained objects,
 - synchronizing, 161

AlwaysOn Dashboard

- add/remove columns, 169
- App1 Availability Group, 167–168
- Cluster Quorum Information
 - screen, 168–169
- Group By button, 169
- synchronization states, 168
- AlwaysOn Failover Cluster Instance (FCI)
 - active/active configuration, 11
 - active node, 10
 - Cluster Disk Selection page, 69–70
 - Cluster Network Configuration page, 69
 - Cluster Resource Group page, 68
 - Collation tab, 71–72
 - Database Engine Configuration page, 72
 - Data Directories tab, 73–74
 - Error Logs and default Extended Event health trace, 66
 - Feature Selection page, 65–66
 - FILESTREAM tab, 75–76
 - five-node N+M configuration, 12–13
 - high availability, 9
 - hacking methodologies, 73
 - Install Failover Cluster Rules, 64–65
 - Instance Configuration page, 67
 - Integration Services service, 66
 - License Terms page, 61–62
 - Metadata
 - DBA, 191
 - DMV, 191
 - failure condition levels, 194
 - hosts the instance, 192
 - retrieving diagnostic information, 196
 - return, 194
 - sp_server_diagnostics, 194–195
 - sys.dm_os_cluster_nodes
 - Columns, 192
 - sys.dm_os_cluster_properties
 - Columns, 193
 - viewing health check configuration
 - Windows administration team, 192
- Microsoft Update page, 62–63
- mixed-mode authentication, 73
- MSSQL13.[InstanceName], 66
- nodes
 - Cluster Network Configuration page, 78–79
 - Cluster Node Configuration page, 78

- License Terms page, 77
- parameters, 81
- PowerShell, 80–81
- Product Key page, 77
- Ready to Add Node page, 80
- Service Accounts page, 79–80
- parameters installation, 77
- perform volume maintenance tasks, 71
- PowerShell installation, 59, 76–77
- Product Key page, 60–61
- Product Updates page, 63–64
- quorum
 - data centers, 13
 - definition, 13
 - high availability, 13
 - models, 14
 - multi-subnet cluster, 14
 - partitions, 13
 - split brain, 13
- SAN replication, 9
- SDKs and management tools, 67
- Server Configuration page, 70, 72–73
- Service Accounts tab, 70–71
- site-aware cluster functionality, 10
- SQL Server Installation
 - Center—Installation, 59–60
- SR technology, 9
- system databases, 66
- tasks, 59
- TempDB tab, 74–75
- three-node cluster, 10
- three-plus node configurations, 12
- two-node cluster, 10
- Windows Authentication Mode, 73
- Windows Firewall, 65
- AlwaysOn Health Trace, 165
 - Extended Events session, 170
 - target data, 170
- Asynchronous mirroring, 15
- Availability Group Listener, 18
- Availability Groups failover, 20

■ B

Business-critical applications, 1

■ C

- Check disk command (CHKDSK), 5
- Cluster
 - ClustNode1 and ClustNode2, 29

- creation
 - admin access point, 44–45
 - begin page, 37–38
 - confirmation page, 41–42, 46
 - DHCP, 44
 - PowerShell, 48
 - report, 47
 - server page, 38–39
 - summary page, 42–43, 46–47
 - testing options page, 40–41
 - validation report, 43–44, 48
 - validation warning page, 39–40

- Disk Configuration, 29
- installation, failover features
 - begin page, 31
 - confirmation page, 36–37
 - features page, 34–35
 - Installation type page, 31–32
 - management tools, 35–36
 - server roles page, 33–34
 - server selection page, 32–33
 - services, PowerShell
 - command, 37

- MSDTC configuration
 - client access point page, 53
 - confirmation page, 54
 - creation, 55
 - downtime, 52
 - DTC resource, 55
 - High Availability Wizard, 52
 - role page, 52
 - SQL Server, 52
 - storage page, 53–54
 - Windows Server, 55

- quorum configuration
 - cloud witness, 49
 - disks, 48, 51
 - Failover Cluster Manager, 48
 - fileshare witness, 49
 - option page, 49
 - PowerShell command, 51
 - storage witness page, 50
 - summary page, 51
 - witness page, 49, 50

- role configuration
 - Failover tab, 57
 - general tab, 55–56
 - options, 56

- tasks, 30
- troubleshooting issues, 29
- Cluster Validation wizard, 58, 122

Configure Cluster Quorum Wizard, 124
 Cost of downtime
 intangible costs, 5
 levels of availability, 5–6
 predicted lifecycle, 5
 tangible costs, 5

■ D

Database mirroring
 AlwaysOn Availability Groups, 15
 data-tier application, 15
 deprecated technology, 15
 DR solution, 15
 high performance mode, 15–16
 high safety, automatic
 failover mode, 16–17
 modes, 15
 network latency, 16
 primary and secondary servers, 16
 synchronous and asynchronous
 method, 16
 TCP endpoint, 15
 Windows cluster service, 15
 witness server, 16
 Data corruption, 5
 Disaster recovery (DR), 1
 cluster configuration
 confirmation page, 123–124
 Failover Cluster Manager, 122
 IP Address, 128–130
 quorum, 124, 126–128
 servers page, 122
 validation warning page, 122–123
 replica configuration
 Backup Preferenes tab, 133
 code implementation, 135–139
 connection times, 141
 data synchronization
 page, 134–135
 Endpoints tab, 132–133
 IP Address, 139–140
 listener tab, 134
 Replicas page, 131
 SQLCMD Mode, 135
 SSMS, 131
 summary page, 135
 validation page, 135

Distributed Availability Groups, 19, 149
 App2 Availability Group, 143
 cluster, 142
 coding, 144
 DR site, 142
 network traffic, 142
 script, failover, 160
 steps, 159
 topology, 142–143
 Distributed Resource Scheduler (DRS), 18
 Distributed Transaction
 Coordinator (DTC), 52
 Dynamic Management View (DMV), 191
 Dynamic Quorum, 14

■ E

Extended events
 actions, 181
 AlwaysOn Events, 172–180
 channels, 172
 CPU utilization, 171
 keywords/category, 171–172
 maps, 182
 monitor Availability Group sessions
 Capture Global Fields page, 185
 Capture page, 184–185
 CREATE EVENT SESSION
 WITH options, 188–189
 creation, 187
 data storage, 186–187
 Filters page, 186
 Properties page, 183
 Summary page, 187
 Template page, 183–184
 packages, 171
 predecessor, 171
 predicates, 181–182
 profiler, 171
 sessions, 182
 targets, 180–181
 types, 182
 WMI, 171

■ F

Fully qualified domain
 name (FQDN), 96

■ **G**

Global Fields, 181

■ **H**

High availability (HA)
 data corruption/human error, 1
 implementation, 1
 Hypertext Markup Language (HTML), 46

■ **I, J, K**

IP Address
 Availability Group Listener
 Properties, 139–140
 Configuring the Availability
 Group, 128
 Core Cluster Resources
 window, 128
 dependencies tab, 129–130
 dependency report, 140–141
 general tab, 129
 OR dependency, 130
 script, 140

■ **L**

Level of availability
 calculation, 2–3
 downtime, 2
 holistic monitoring tools, 2
 network/application servers, 1–2
 proactive maintenance, 4
 SLAs and SLOs, 3–4
 uptime, 1
 Log sequence number (LSN), 22
 Log shipping
 disaster recovery, 23
 DR and reporting servers, 23–24
 failover, 25
 recovery modes, 24
 remote monitor server, 25
 topology, 23

■ **M**

Microsoft Cluster Service (MCS), 58
 Microsoft Distributed Transaction
 Coordinator (MSDTC), 30, 52

Microsoft's Customer Experience
 Improvement Program, 77
 Mixed-mode authentication, 73

■ **N, O, P**

Node, 10
 Nonfunctional requirements (NFRs), 25

■ **Q**

Quorum
 Configuration Option
 page, 124–125
 Configure File Share Witness page,
 126–127
 Confirmation page, 127–128
 Voting Configuration
 page, 125
 Witness page, 126

■ **R**

Readable secondary replicas
 Availability Group Listener, 145
 load balancing topologies, 145–146
 log streaming, 144
 log truncation, 144
 read-intent traffic, 147
 read-only routing
 configuration, 145, 147
 round-robin algorithm, 145
 snapshot isolation, 144
 temporary statistics, 144
 vertically scaled
 reporting, 144
 Recovery point objective (RPO), 93
 applications, 4
 data corruption, 5
 data warehouse, 4
 intrasite availability and intersite
 recovery, 4
 OLTP (Online Transaction Processing)
 database, 4
 Recovery time objective (RTO), 97
 data corruption, 5
 intrasite/intersite failover, 5
 noncommitted
 transactions, 4
 Redundant infrastructure, 7

■ S

- Service-level agreements(SLAs), 3–4, 93
- Service-level objectives (SLOs), 3–4
- SQLCMD mode, 100
- SQL Server Integration Services (SSIS), 30, 52, 65
- SQL Server Management Studio (SSMS), 131
- Standby server classifications, 6
- Storage Replica (SR), 9
- Synchronous Commit mode
 - Availability Group topology, 112
 - network latency and disk performance, 112
 - performance test results
 - SQL Server 2014, 117
 - SQL Server 2016, 117
 - SAN replication, 118
 - script, 112–116
 - three-node cluster, 118
- Systems Operations Center (SOC), 167

■ T, U

- TempDB database, 74
- Tie Breaker, 15
- Total cost of
 - ownership (TCO), 6
- Transaction Undo
 - File (TUF), 24
- Transmission Control
 - Protocol (TCP), 96

■ V

- Virtual computer object (VCO), 98
- Virtual machines (VMs), 18

■ W, X, Y, Z

- Windows Cluster Service (WCS), 9, 30
- Windows Server Update
 - Services (WSUS), 62