THE EXPERT'S VOICE® IN SQL SERVER

SQL Server AlwaysOn Revealed

Supporting 24 x 7 operations with continuous uptime

Peter A Carter

Apress®

SQL Server AlwaysOn Revealed

Peter A Carter

Apress®

SQL Server AlwaysOn Revealed

ISBN-13 (pbk): 978-1-4842-1762-7

ISBN-13 (electronic): 978-1-4842-1763-4

Managing Director: Welmoed Spahr
Lead Editor: Jonathan Gennick
Technical Reviewer: Alex Grinberg and Louis Davidson
Editorial Board: Steve Anglin, Louise Corrigan, Jim DeWolf, Jonathan Gennick, Robert Hutchinson, Michelle Lowman, James Markham, Susan McDermott, Matthew Moodie, Jeffrey Pepper, Douglas Pundick, Ben Renow-Clarke, Gwenan Spearing
Coordinating Editor: Jill Balzano
Copy Editor: Rebecca Rider
Compositor: SPi Global
Indexer: SPi Global
Artist: SPi Global
Cover Designer: Anna Ishchenko

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 material referenced by the author in this text is 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/.

This book is dedicated to the beloved memory of Margaret Carter (1938-2015)

Contents at a Glance

Contents

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 Reviewers

Louis Davidson has been in the IT industry for more than 15 years as a corporate database developer and architect. He has spent the majority of his career working with Microsoft SQL Server, beginning in the early days of version 1.0. He has a bachelor's degree from the University of Tennessee at Chattanooga in computer science, with a minor in mathematics. Louis is the data architect for Compass Technology (Compass.net) in Chesapeake, Virginia, leading database development on their suite of nonprofit-oriented customer relationship management (CRM) products, which are built on the Microsoft CRM platform and SQL Server technologies.

Alex Grinberg is a senior SQL Server database administrator (DBA) with more than 20 years of IT experience. He has been working on Microsoft SQL Server products since version 6.5. Alex currently works in the Pennsylvania branch of Cox Automotive, headquartered in Atlanta, GA. His primary duties are to provide architecture, tuning, optimization, analysis, operational, and development services; to create new applications; to convert legacy technologies (SQL Server, VB.NET, and C#); and to provide on-site training with the latest Microsoft technologies including .NET (VB and C#), SSRS, and SSIS. Alex is a frequent speaker at professional IT events, including SQLSaturdays, Code Camps, SQL Server User Groups, and other industry seminars, where he shares his cumulative knowledge. He is the guest author for SQLServerSentral.com and is also the cofounder of HexaArt Inc., an IT consulting services company for small and mid-size corporations. For any questions or consulting needs, Alex can be reached at hexaart@gmail.com.

CHAPTER 1

High Availability and Disaster Recovery Concepts

In today's 24×7 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.

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 percent, 99.9 percent, 99.99 percent, or 99.999 percent availability. As a result, the level of availability is often referred to in 9s. For example, five 9s of availability means 99.999 percent uptime and three 9s means 99.9 percent 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 do 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 not only caused 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 process and all other activity is read-only reporting. For highly transactional systems, however, such as an OLTP 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 intra-site availability and inter-site 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 intra-site or inter-site 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 you 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 accidently 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 $525,474 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 $520,334 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 in nines. 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.

Recover Point Objective is a measure of how much data it is acceptable to loose 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 mane 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 warn standby server is one which is kept synchronized with the live server, but is not configured to failover 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.

CHAPTER 2

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 discuss 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, this involves SAN replication, which means that a cluster is normally configured within a single site.

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.

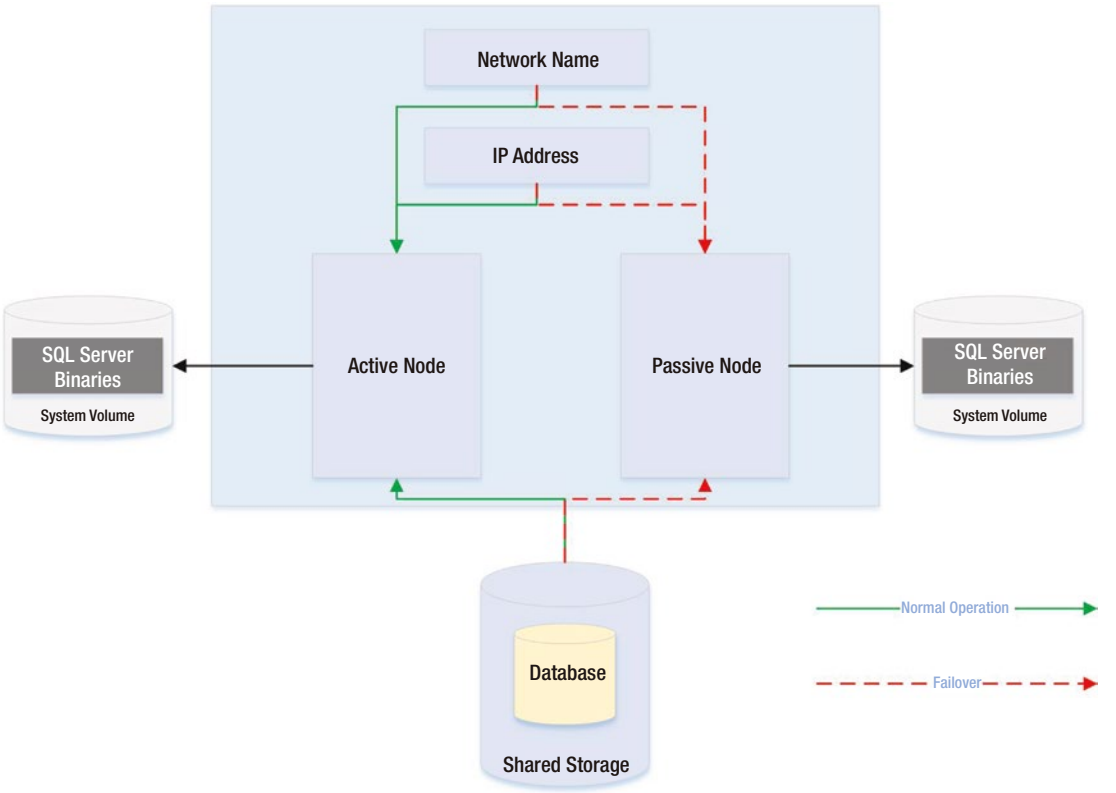

Figure 2-1. *Two-node cluster*

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.

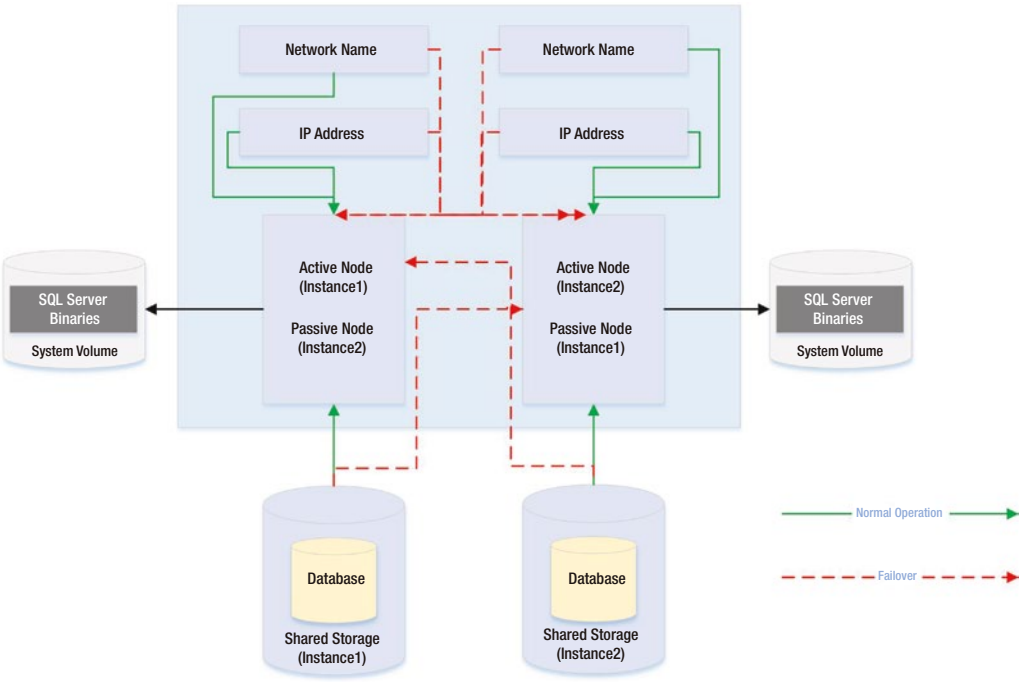

Figure 2-2. *Active/Active cluster*

■ **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 3 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.

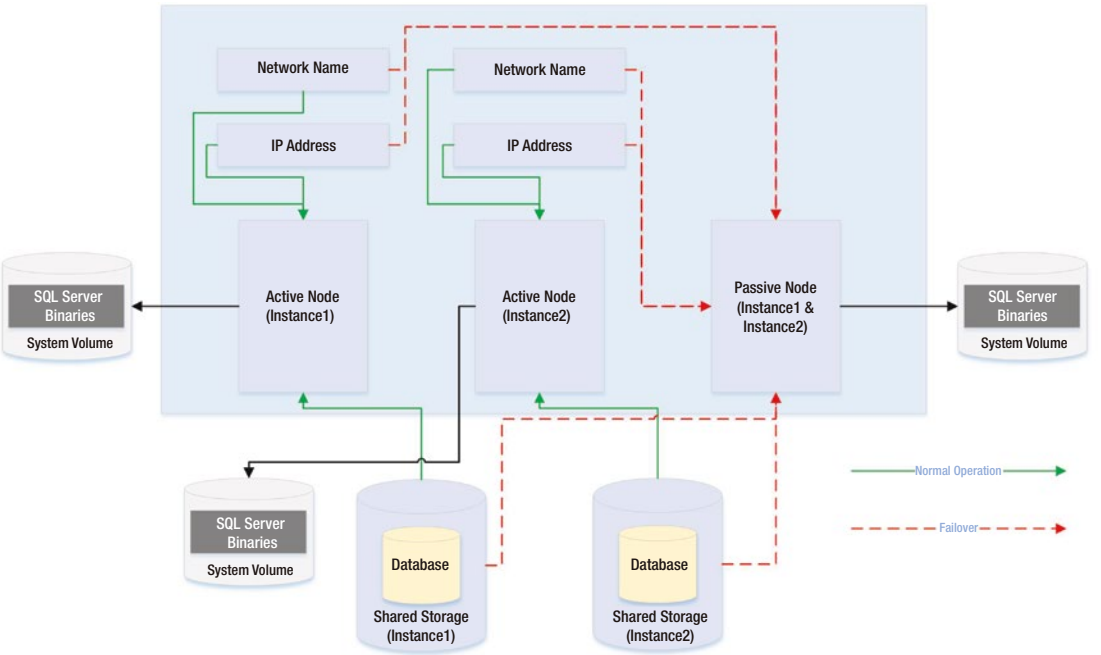

Figure 2-3. *Three-node N+1 configuration*

In an N+1 configuration, in a multi-failure 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.

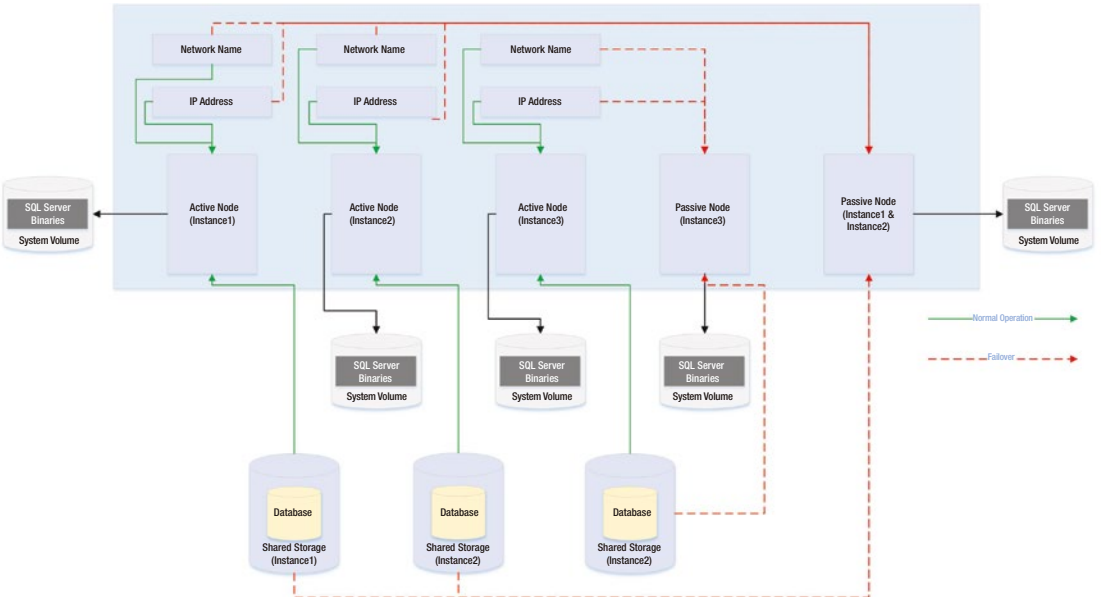

Figure 2-4. *Five-node N+M configuration*

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 = (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 re-join 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*

**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 possibly to manually remove a nodes 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 does 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 know 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 also introduces 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 multi-team 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.

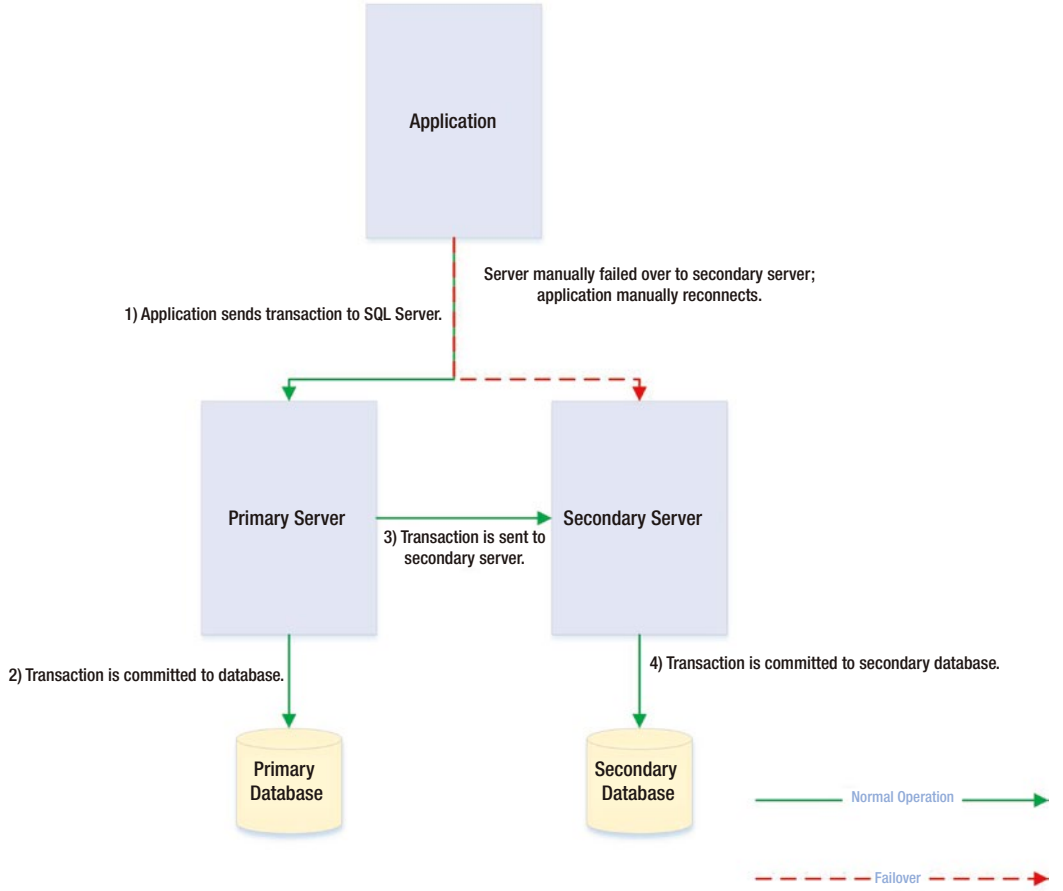

Figure 2-5. *Database mirroring in High Performance mode*

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 loose 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.

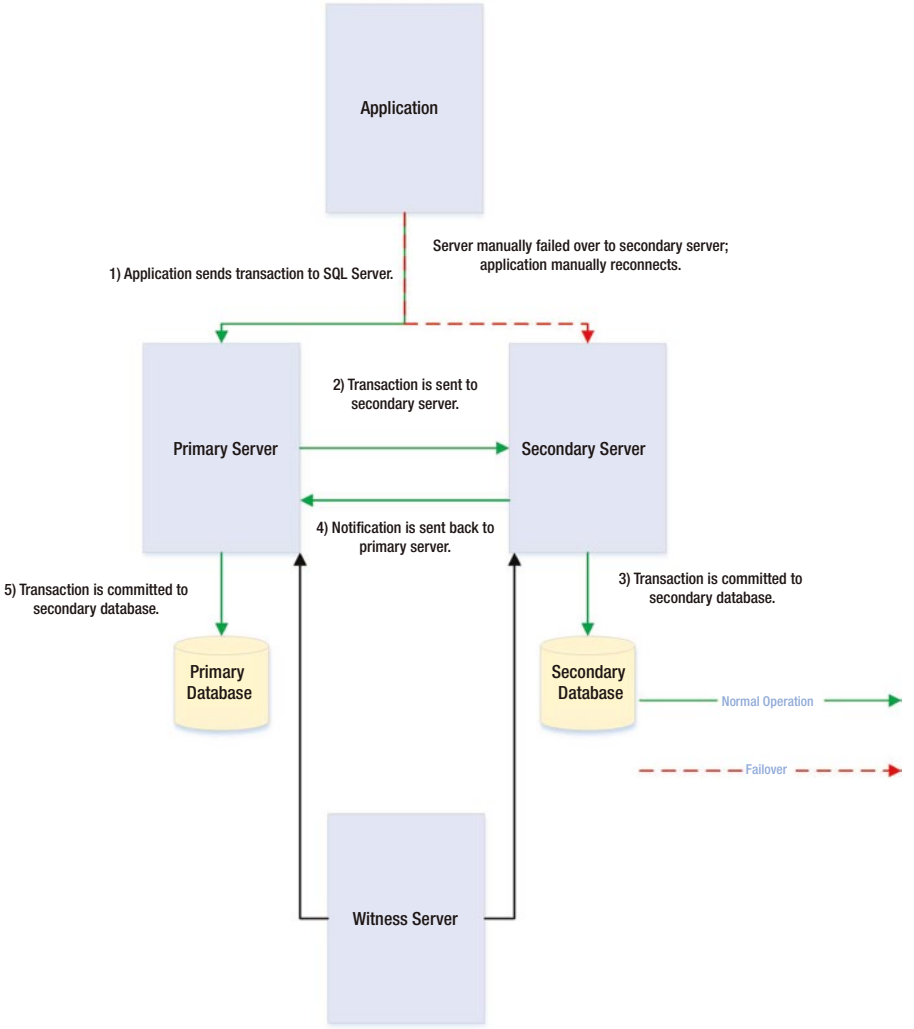

Figure 2-6. *Database mirroring in High Safety with Automatic Failover mode*

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.

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 at the expense of performance degradation.

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 applications 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.

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, nor 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.

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.

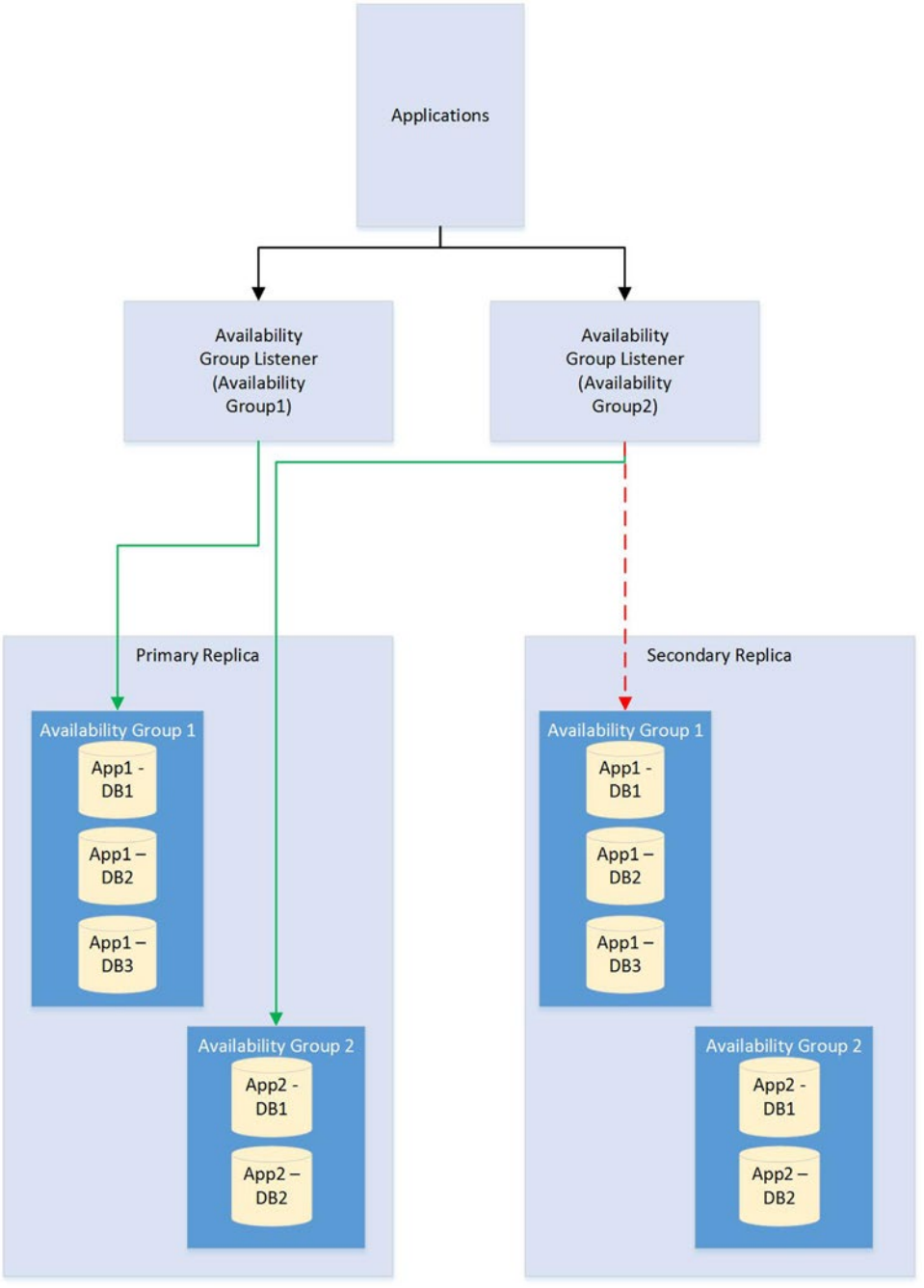

Figure 2-7. *Availability groups failover*

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.

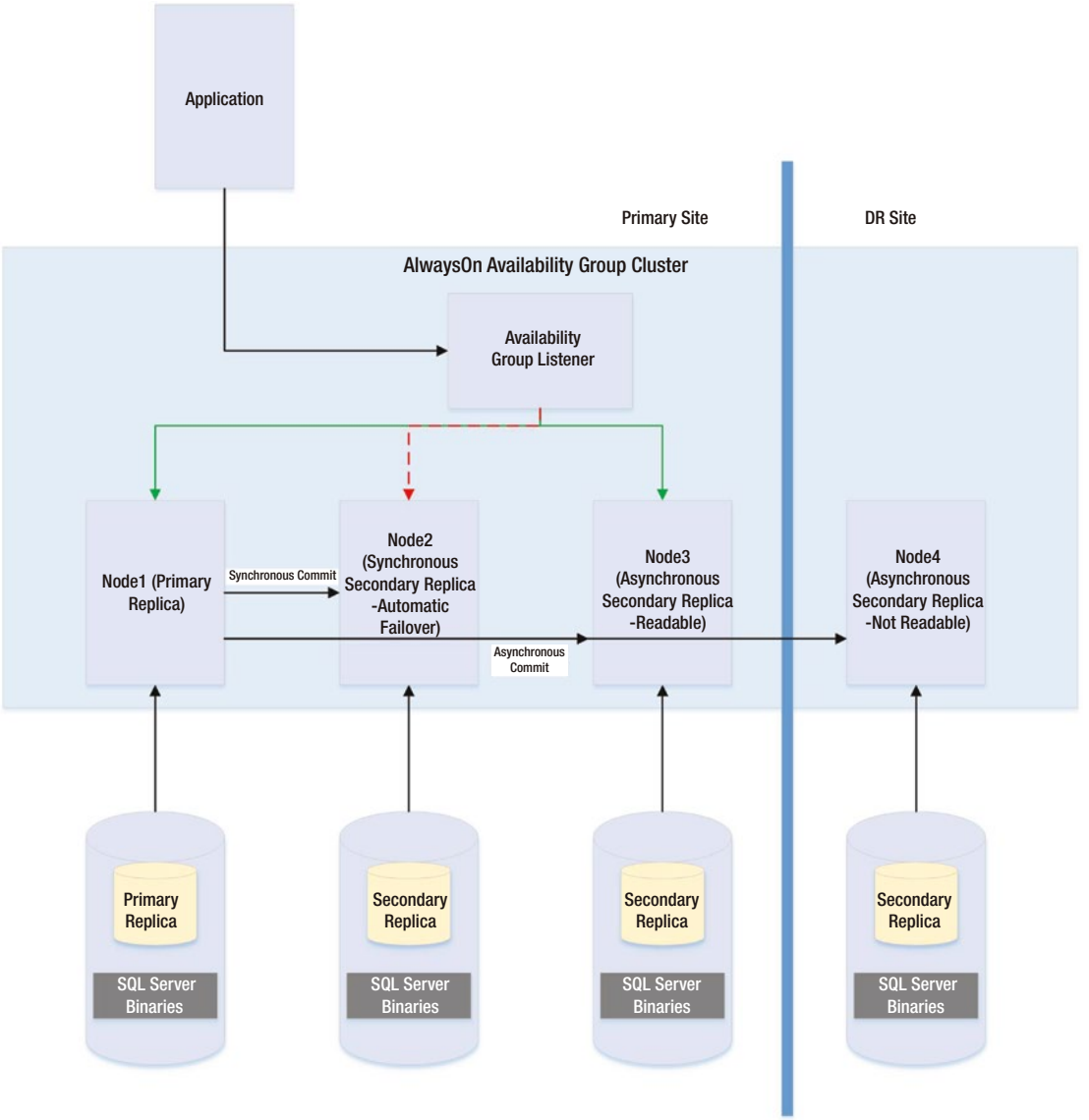

Figure 2-8. *AlwaysOn Availability Group topology*

■ **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 principle 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 accidently 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.

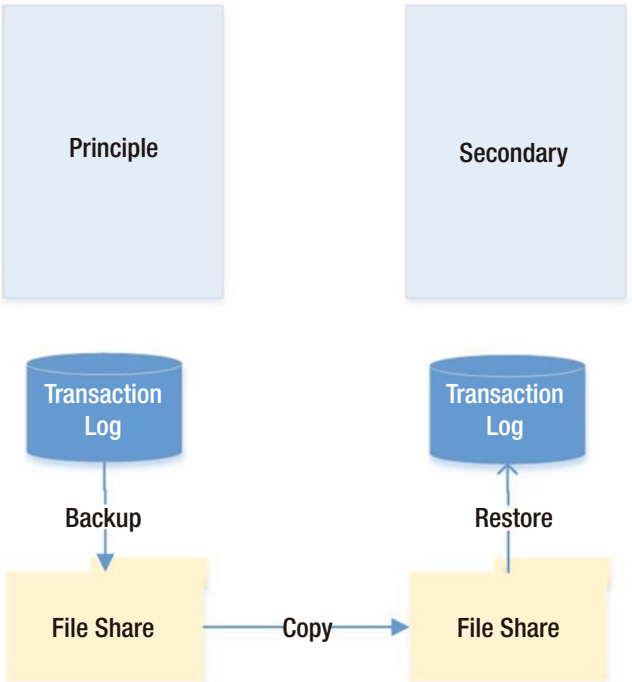

Figure 2-9. *Log Shipping topology*

Recovery Modes

In a log shipping topology, there is always exactly one principle 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.

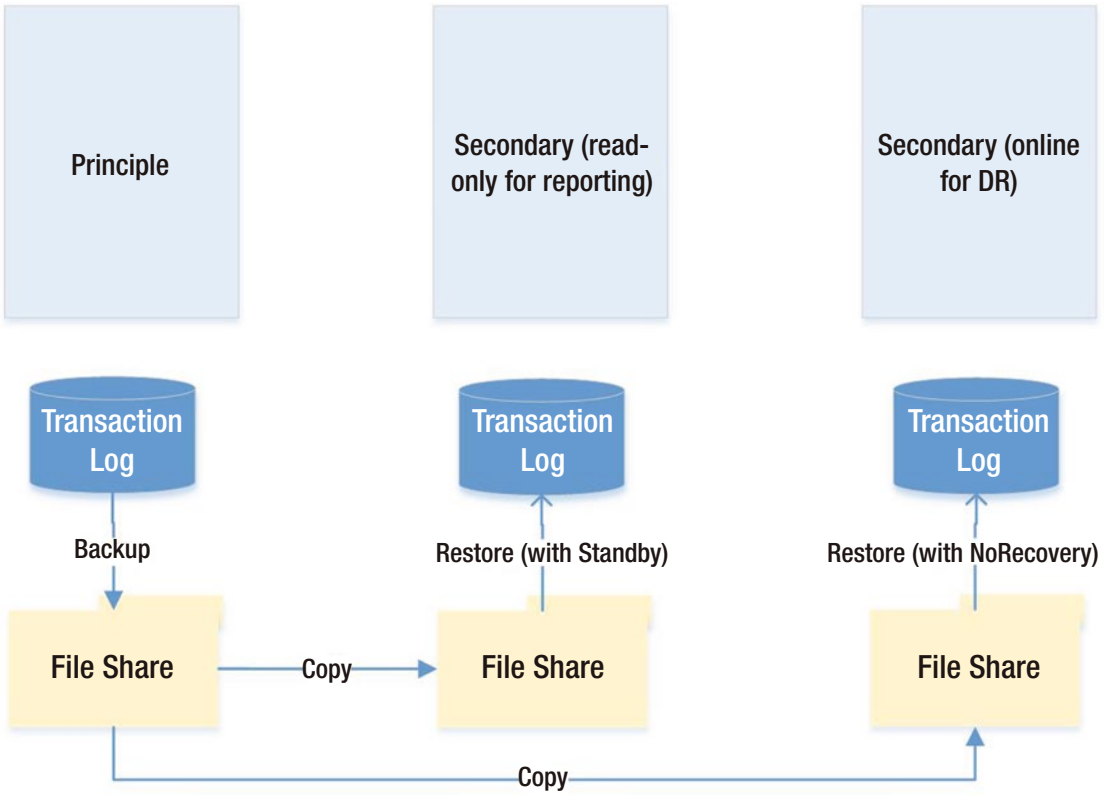

Figure 2-10. *Log shipping with DR and reporting servers*

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 non-functional 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.

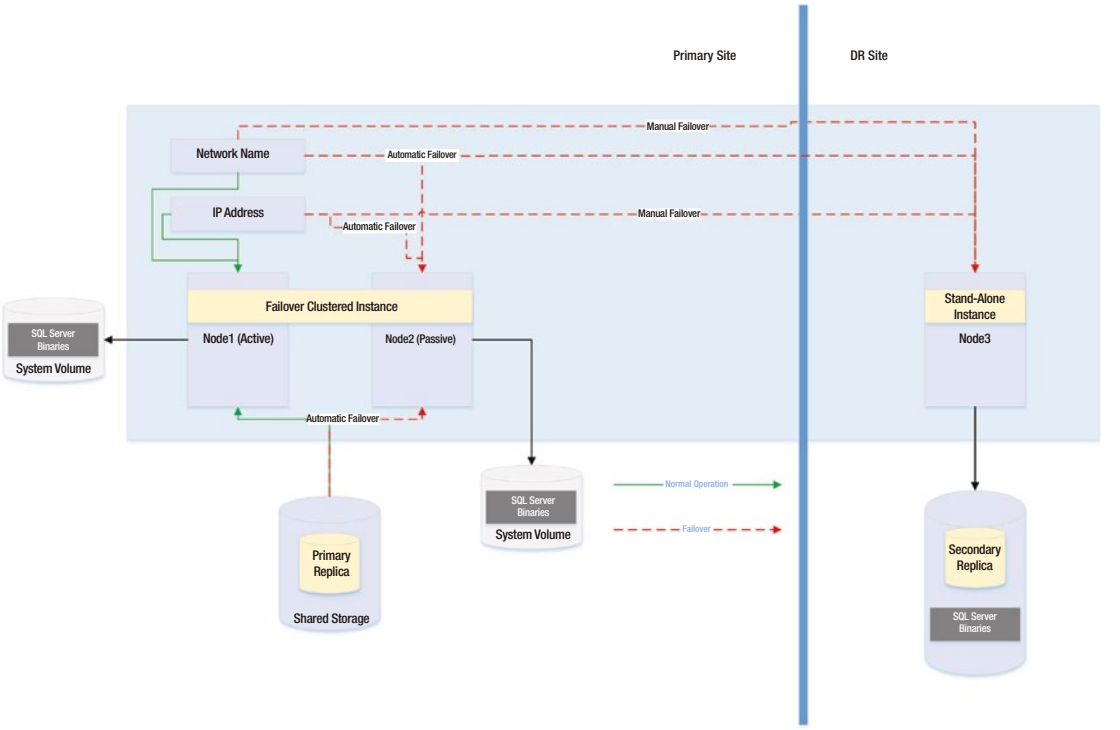

Figure 2-11. *Clustering and AlwaysOn Availability Groups combined*

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.

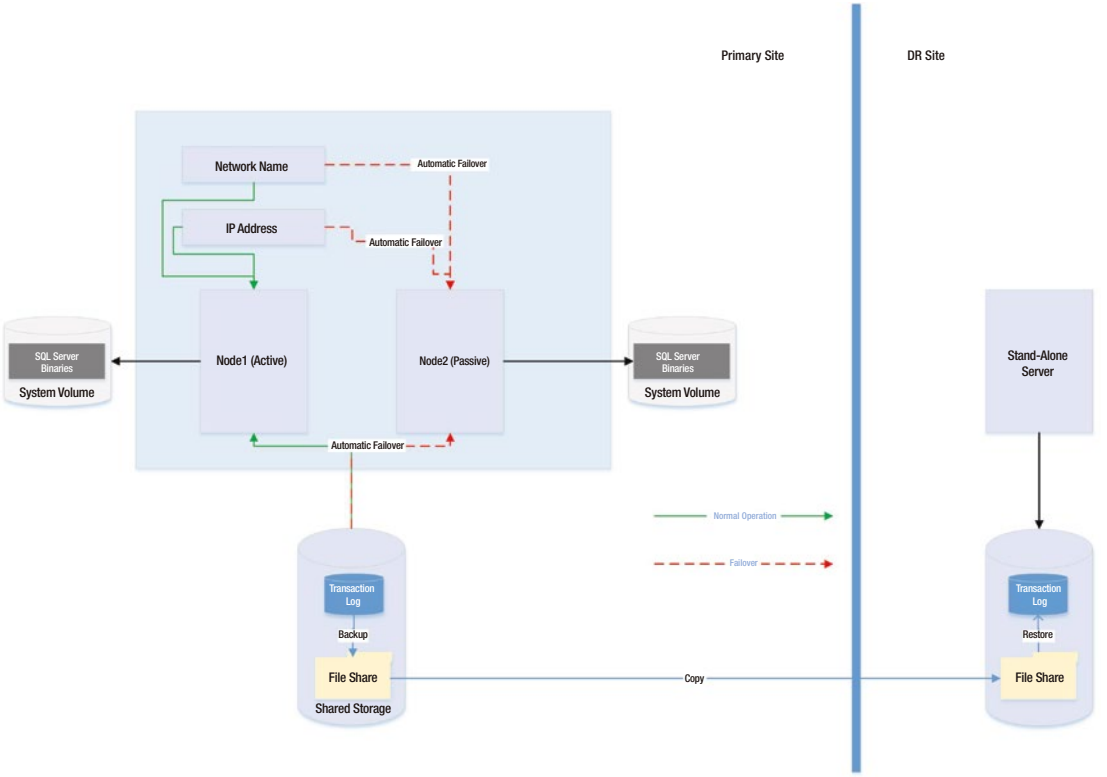

Figure 2-12. *Clustering combined with log shipping*

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.

CHAPTER 3

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 begins by looking at how to build a cluster at the Windows level and discusses some of the possible configurations.

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.

Note To support demonstrations in this chapter, we use a domain called `PROSQLADMIN`. The domain contains a domain controller, which is also configured to serve up five iSCSI disks (Data, Logs, TempDB, MSDTC, and Quorum). Two servers also act as cluster nodes named `ClusterNode1` and `ClusterNode2`. Each server has two NICs (network interface cards) on different subnets; one will be used for data and the other for the cluster heartbeat.

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.

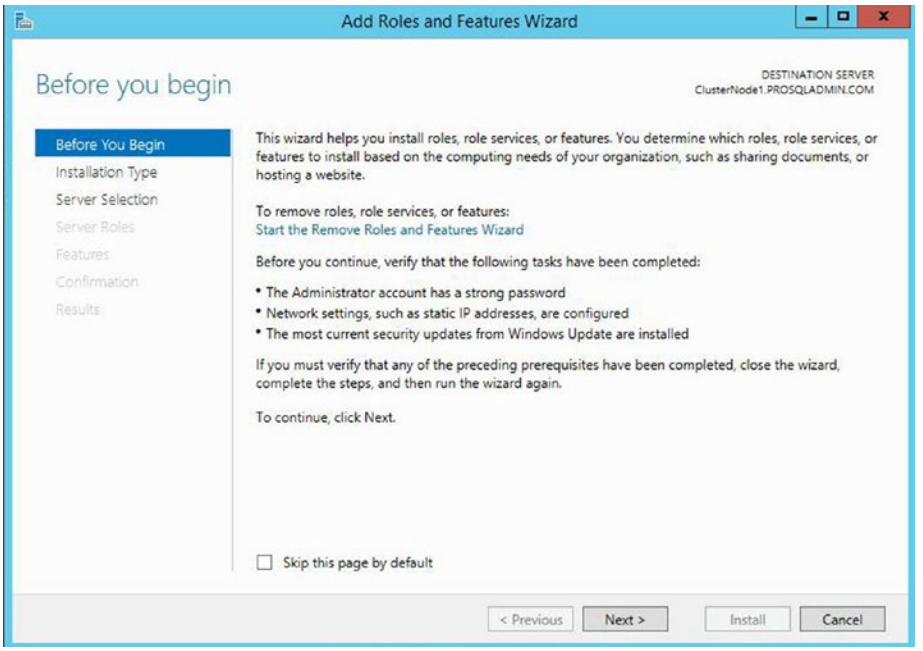

Figure 3-1. *The Before You Begin page*

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

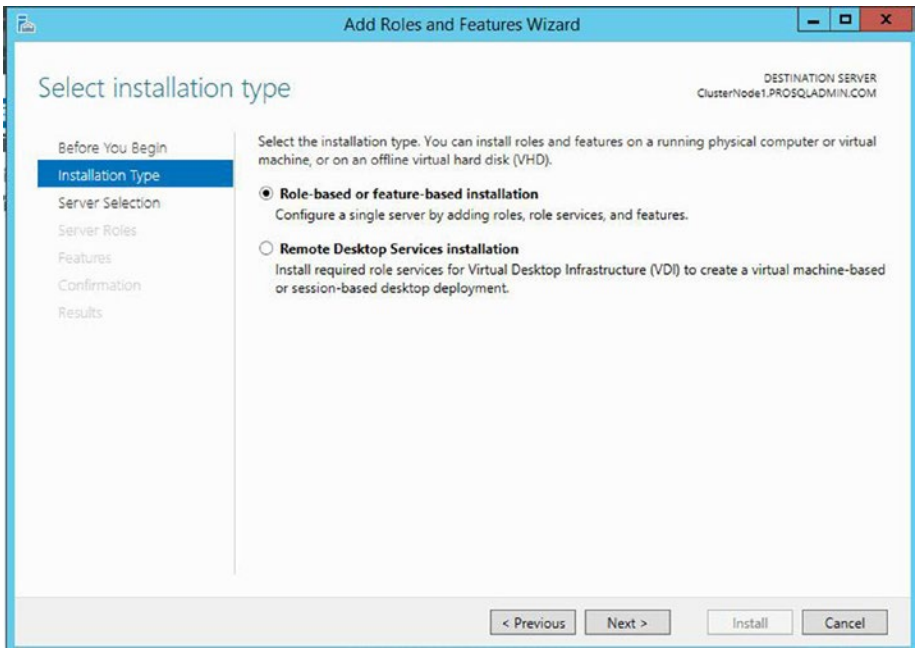

Figure 3-2. *The Installation Type page*

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

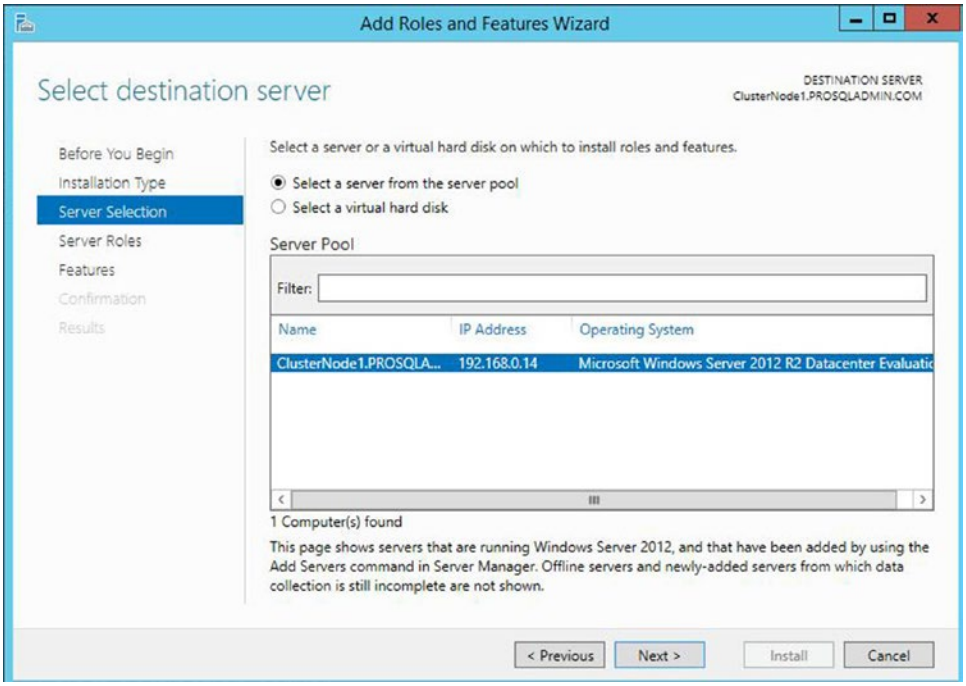

Figure 3-3. *The Server Selection page*

The Server Roles page of the wizard allows you to select any server roles that you want configured. 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.

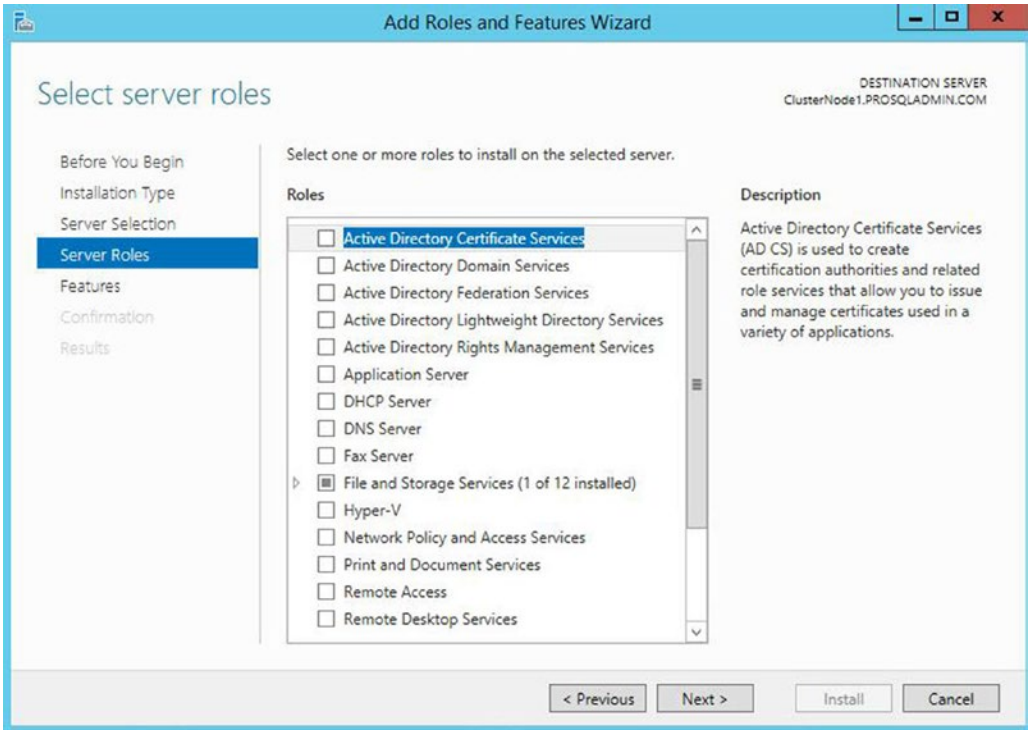

Figure 3-4. *The Server Roles page*

On the Features page of the wizard, we need to select both .NET Framework 3.5 Features and Failover Clustering, as shown in Figure 3-5. This satisfies the prerequisites for building the Windows cluster and also the AlwaysOn failover cluster instance.

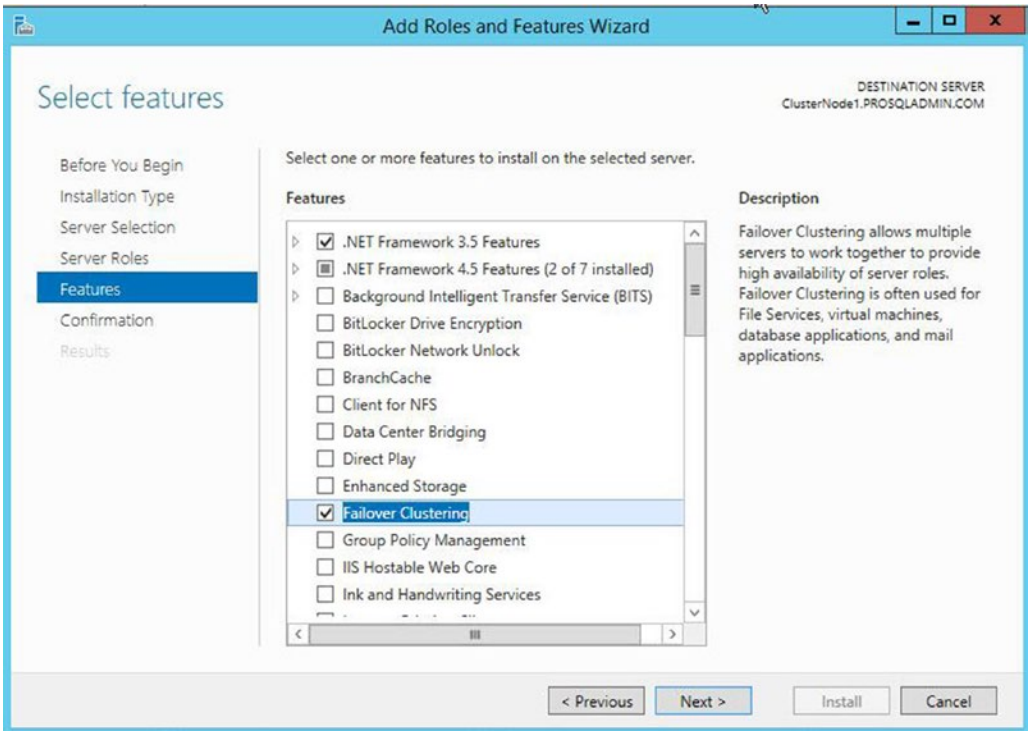

Figure 3-5. *The Features page*

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 managing the cluster directly from the nodes, check this option.

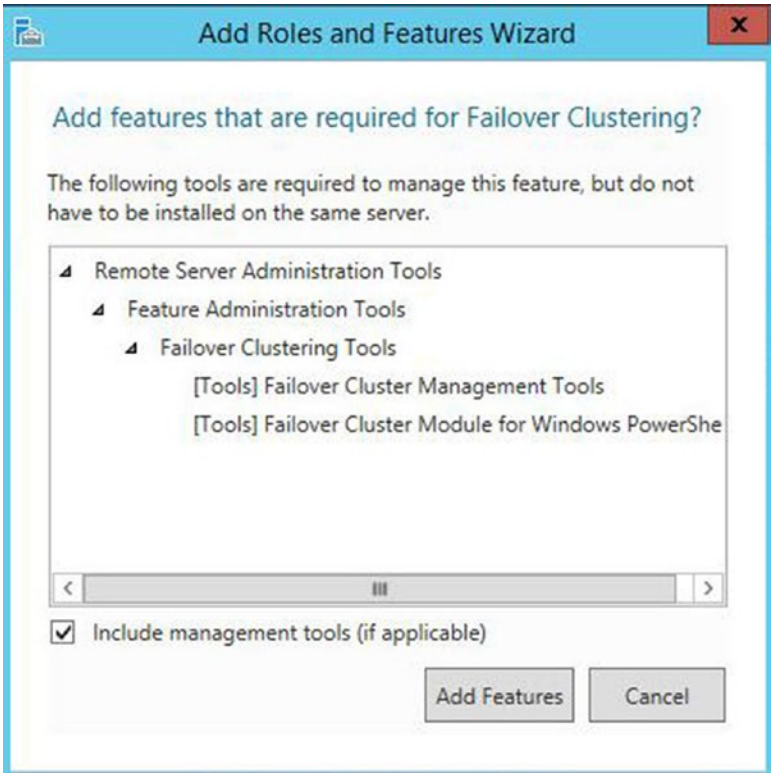

Figure 3-6. *Selecting management tools*

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 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.

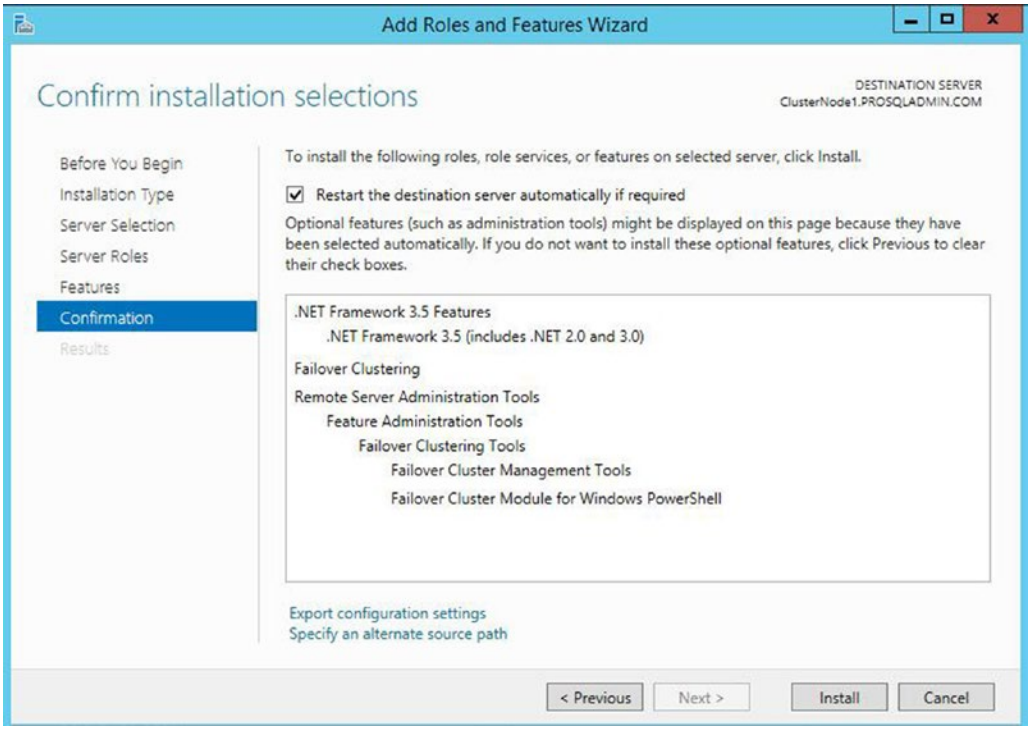

Figure 3-7. *The Confirmation page*

Instead of installing the cluster services through Server Manager, you can install them from PowerShell. The script in Listing 3-1 achieves the same results as the preceding steps.

Listing 3-1. Installing Cluster Services

```
Install-WindowsFeature –name NET-Framework-Core

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 using a domain account, and then run Failover Cluster Manager from Administrative Tools.

The Before You Begin page of the Create Cluster Wizard warns that the domain account you use to install the cluster must be an administrator of the cluster nodes and that Microsoft only supports clusters that pass all verification tests, as shown in Figure 3-8.

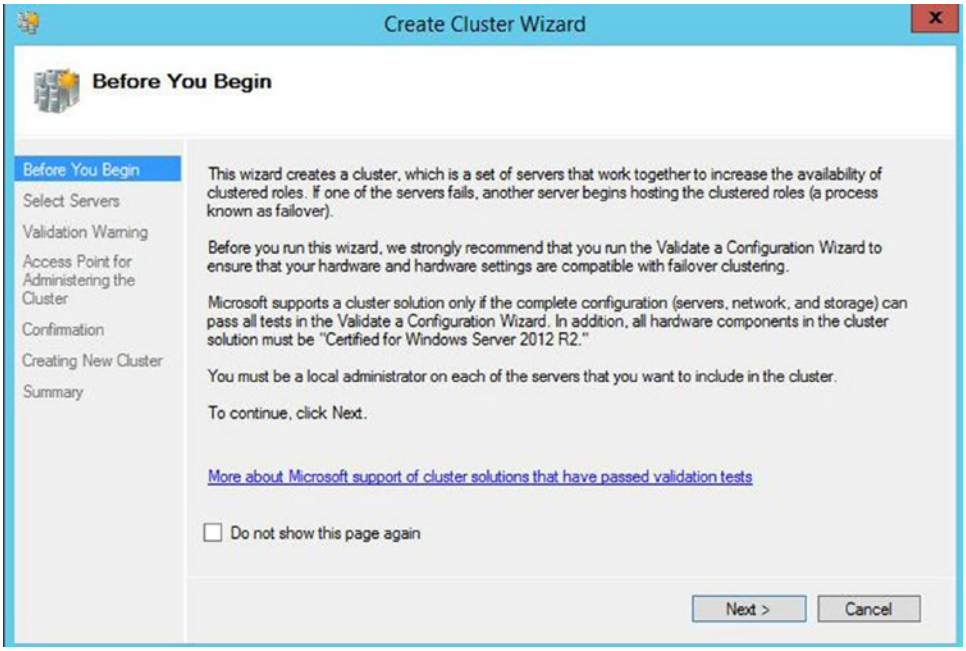

Figure 3-8. *The Before You Begin page*

On the Select Servers screen of the wizard, you need to enter the names of the cluster nodes. Even if you enter just the short names of the servers, they will be converted to fully qualified names in the format `server.domain`. This is illustrated in Figure 3-9. In our case, our cluster nodes are named `ClusterNode1` and `ClusterNode2`, respectively.

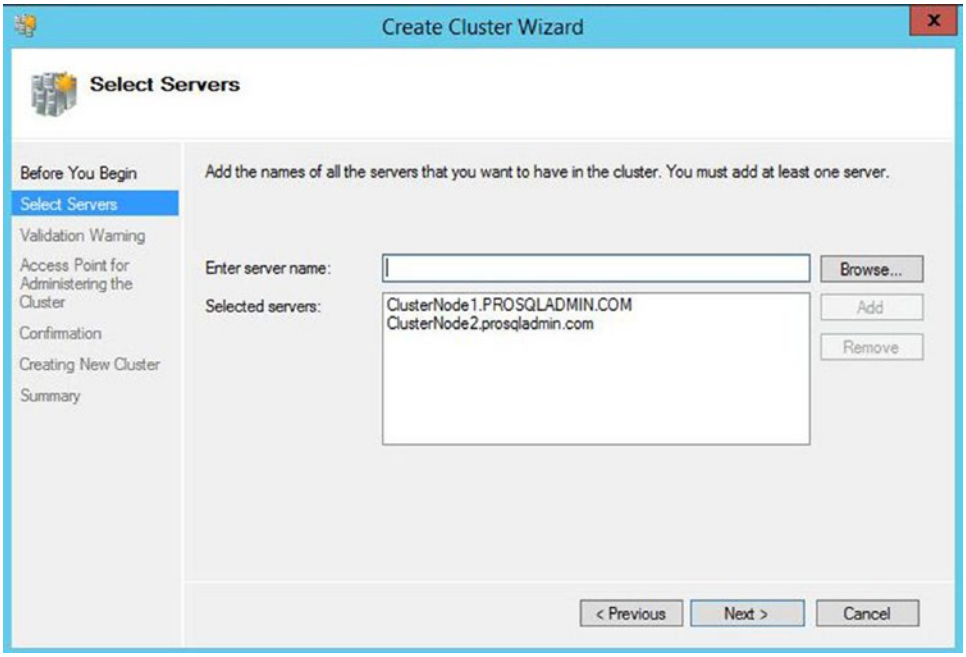

Figure 3-9. *The Select Servers page*

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.

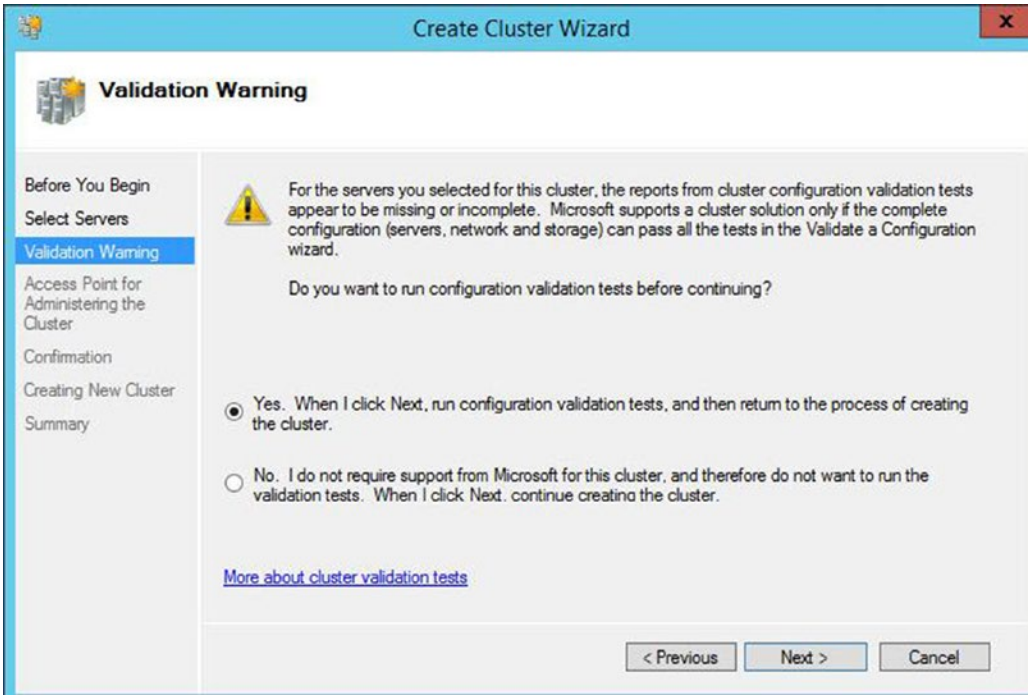

Figure 3-10. *The Validation Warning page*

■ **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 complete 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.

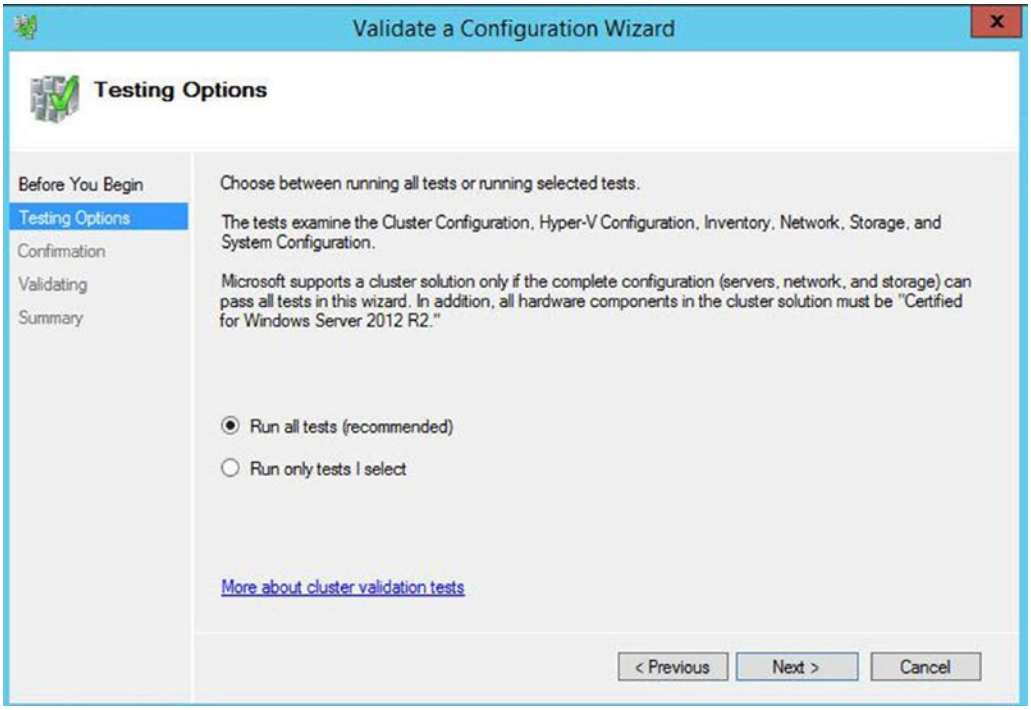

Figure 3-11. *The Testing Options page*

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)

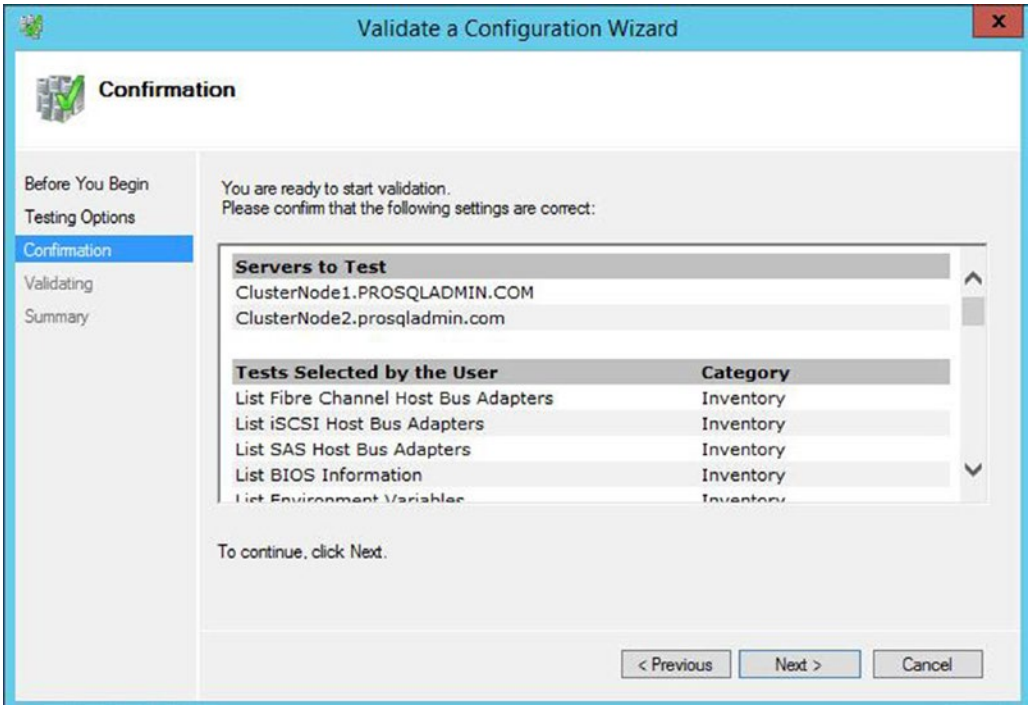

Figure 3-12. *The Confirmation page*

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. AlwaysOn Availability Groups are discussed in further detail in Chapter 5.

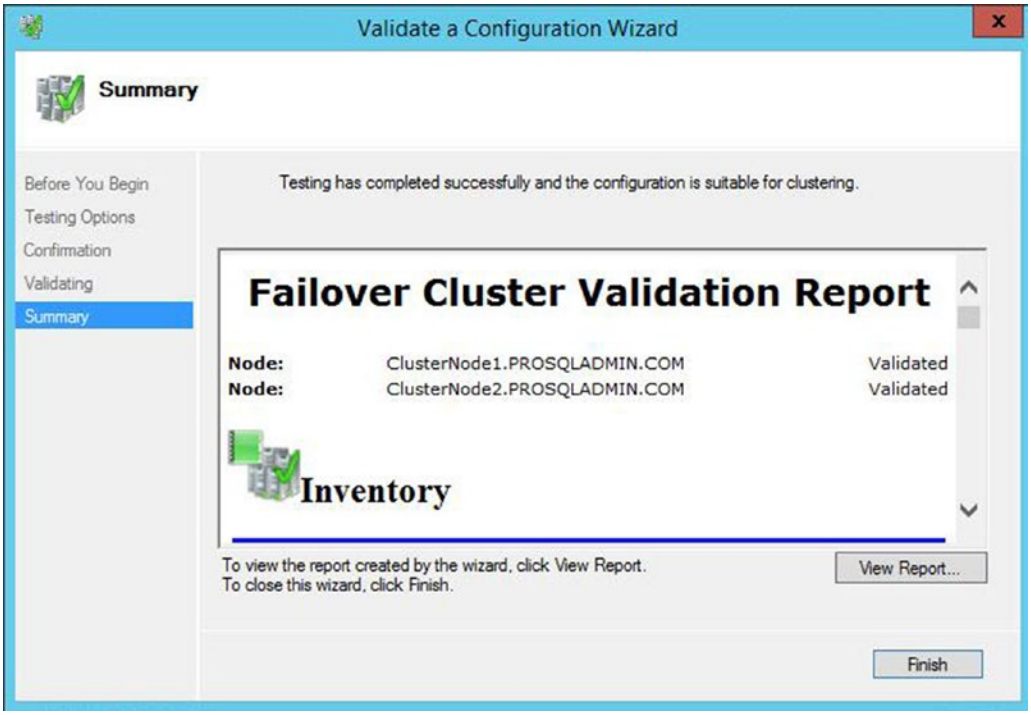

Figure 3-13. *The Summary page*

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.

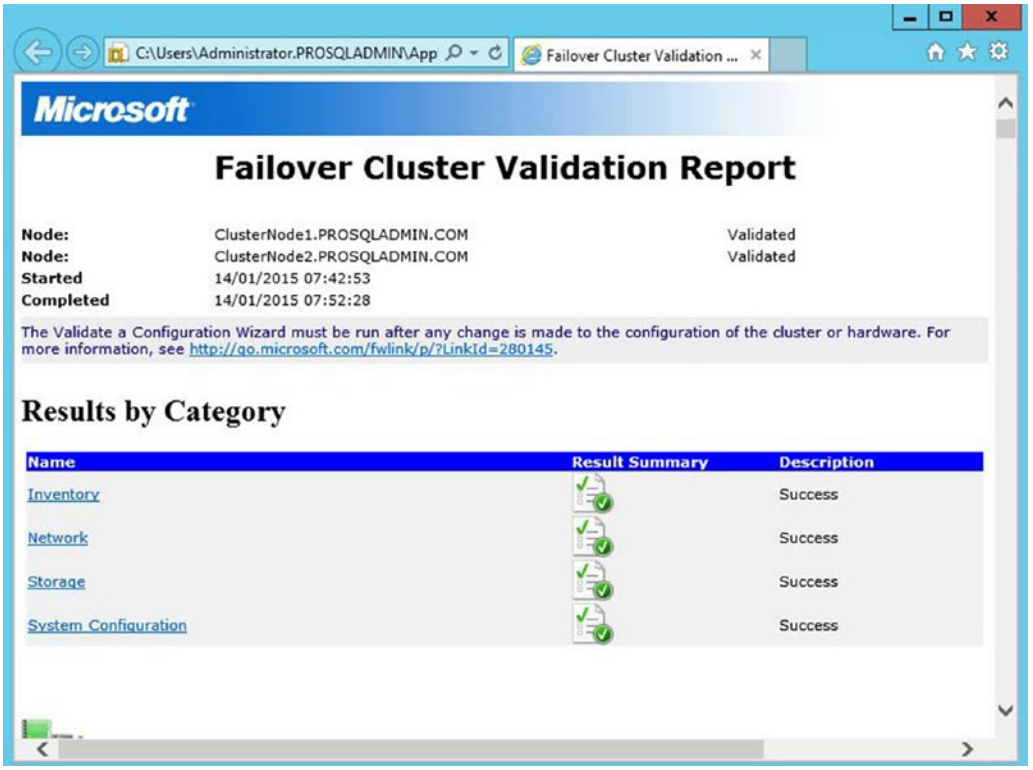

Figure 3-14. *The Failover Cluster Validation Report*

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. On this screen, illustrated in Figure 3-15, you need to enter the virtual name of your cluster and the IP address for administering the cluster. We name our cluster `PROSQLADMIN-C` and assign an IP address of `192.168.0.20`.

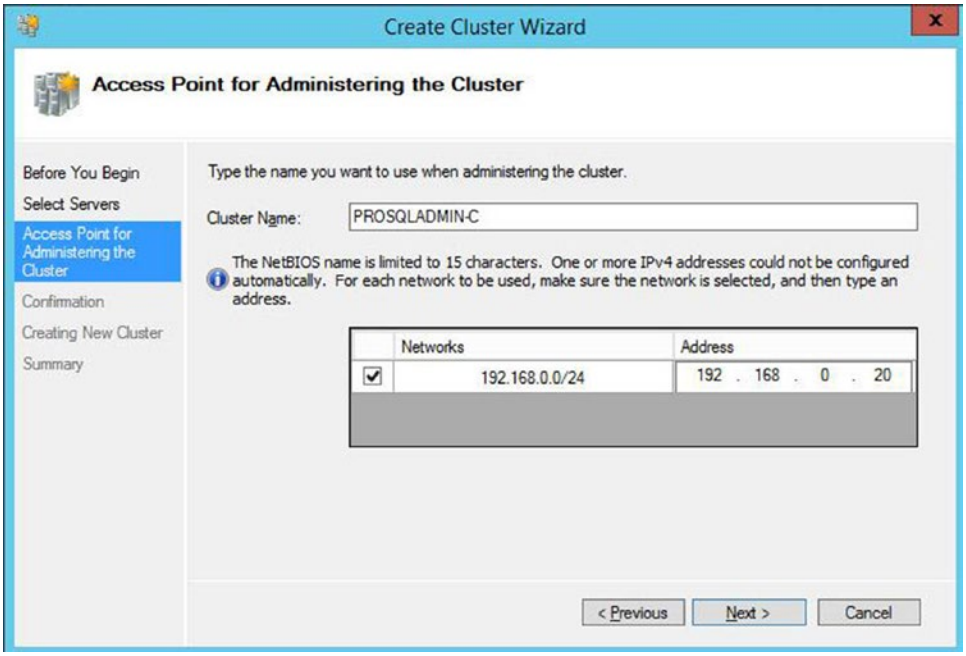

Figure 3-15. *The Access Point For Administering The Cluster page*

■ **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 single site and single subnet, so only one network range is displayed. If you are configuring a multi-subnet cluster, however, then the wizard detects this, and multiple networks display. 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 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.

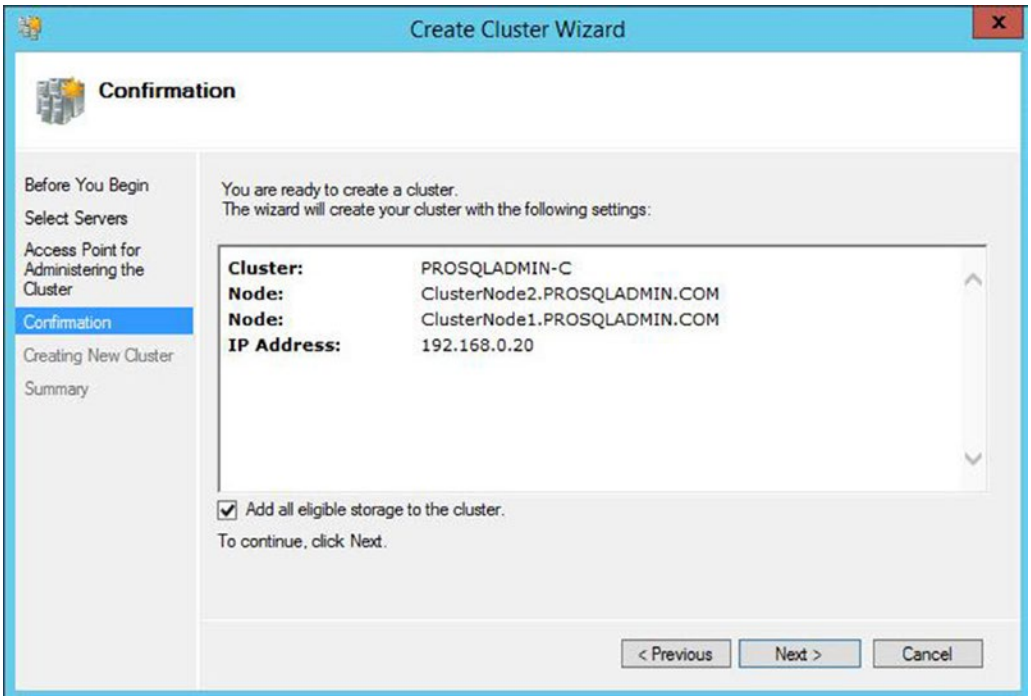

Figure 3-16. *The Confirmation page*

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

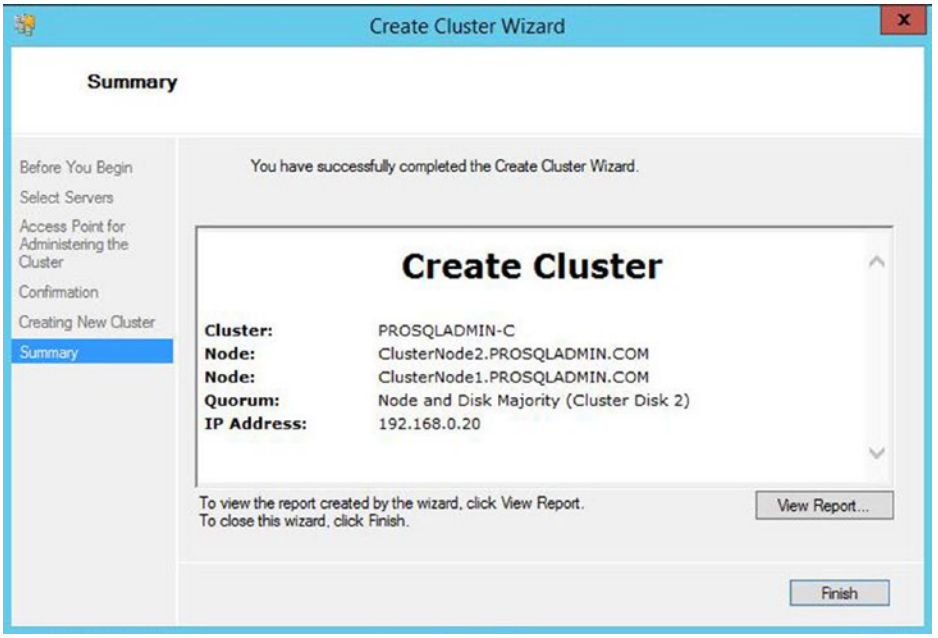

Figure 3-17. *The Summary page*

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


Microsoft

Create Cluster

Cluster:	PROSQLADMIN-C
Node:	ClusterNode2.PROSQLADMIN.COM
Node:	ClusterNode1.PROSQLADMIN.COM
Quorum:	Node and Disk Majority (Cluster Disk 2)
IP Address:	192.168.0.20
Started	14/01/2015 09:48:24
Completed	14/01/2015 09:48:59

Beginning to configure the cluster PROSQLADMIN-C.

Initializing Cluster PROSQLADMIN-C.

Validating cluster state on node ClusterNode2.PROSQLADMIN.COM.

Find a suitable domain controller for node ClusterNode2.PROSQLADMIN.COM.

Searching the domain for computer object 'PROSQLADMIN-C'.

Bind to domain controller \\WIN-19J5OCCCPHP.PROSQLADMIN.COM.

Check whether the computer object PROSQLADMIN-C for node ClusterNode2.PROSQLADMIN.COM exists in the domain. Domain controller \\WIN-19J5OCCCPHP.PROSQLADMIN.COM.

Computer object for node ClusterNode2.PROSQLADMIN.COM does not exist in the domain.

Creating a new computer account (object) for 'PROSQLADMIN-C' in the domain.

Check whether the computer object ClusterNode2 for node ClusterNode2.PROSQLADMIN.COM exists in the domain. Domain controller \\WIN-19J5OCCCPHP.PROSQLADMIN.COM.

Creating computer object in organizational unit CN=Computers,DC=PROSQLADMIN,DC=COM where node ClusterNode2.PROSQLADMIN.COM exists.

Create computer object PROSQLADMIN-C on domain controller \\WIN-19J5OCCCPHP.PROSQLADMIN.COM in organizational unit CN=Computers,DC=PROSQLADMIN,DC=COM.

Check whether the computer object PROSQLADMIN-C for node ClusterNode2.PROSQLADMIN.COM exists in the domain. Domain controller \\WIN-19J5OCCCPHP.PROSQLADMIN.COM.

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.prosqladmin.com,Clusternode2.prosqladmin.com

#Create the cluster

New-Cluster -Node ClusterNode1.prosqladmin.com,ClusterNode2.prosqladmin.com -StaticAddress
192.168.0.20 -Name PROSQLADMIN-C
```

Figure 3-19 shows the results of running this script. The first part of the output provides the details of the validation report that has been generated. The second part confirms the name of the cluster that has been created.

```
PS C:\Users\Administrator.PROSQLADMIN> Test-Cluster -Node Clusternode1.prosqladmin.com,Clusternode2.prosqladmin.com
New-Cluster -Node ClusterNode1.prosqladmin.com,ClusterNode2.prosqladmin.com -StaticAddress 192.168.0.20 -Name PROSQLADMIN-C

Mode                LastWriteTime     Length Name
----                -------------     ------ ----
-a---         14/01/2015     11:27     442532 Validation Report 2015.01.14 At 11.22.10.xml.mht

Name : PROSQLADMIN-C


PS C:\Users\Administrator.PROSQLADMIN>
```

Figure 3-19. *Validate and create cluster output*

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 in the Failover Cluster Manager, we can instantly see one configuration change that we need to make. The cluster has chosen the Logs volume as the quorum drive, as shown in Figure 3-20.

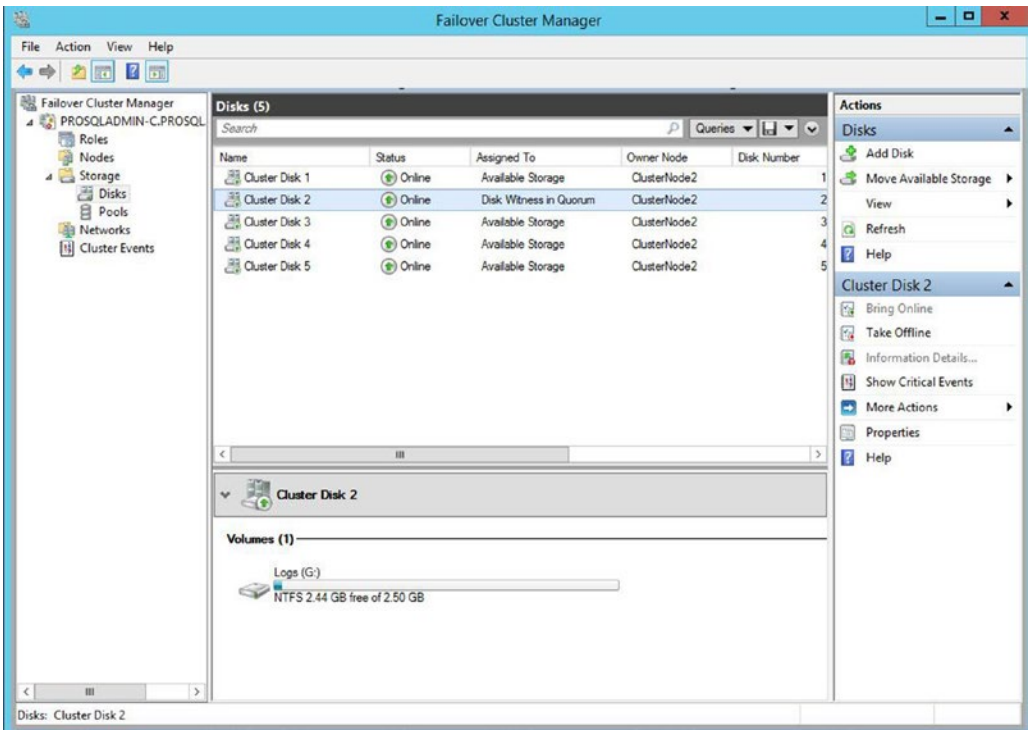

Figure 3-20. *Cluster disks*

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 3-21, we choose the Select The Quorum Witness option.

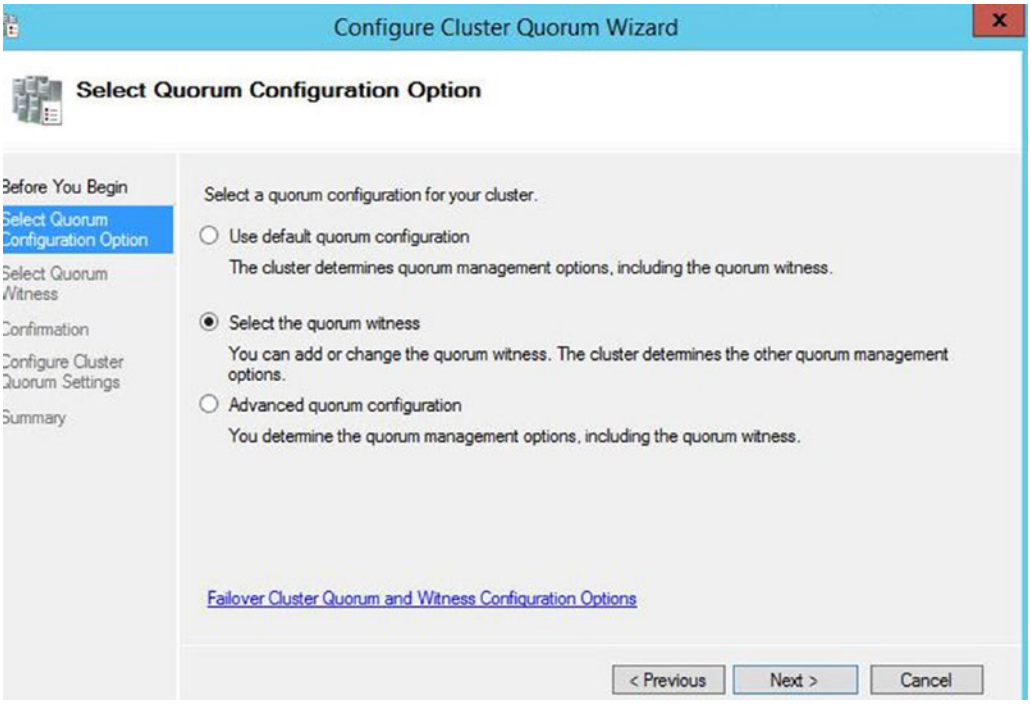

Figure 3-21. *The Select Quorum Configuration Option page*

On the Select Quorum Witness page, we select the option to configure a disk witness. This is illustrated in Figure 3-22.

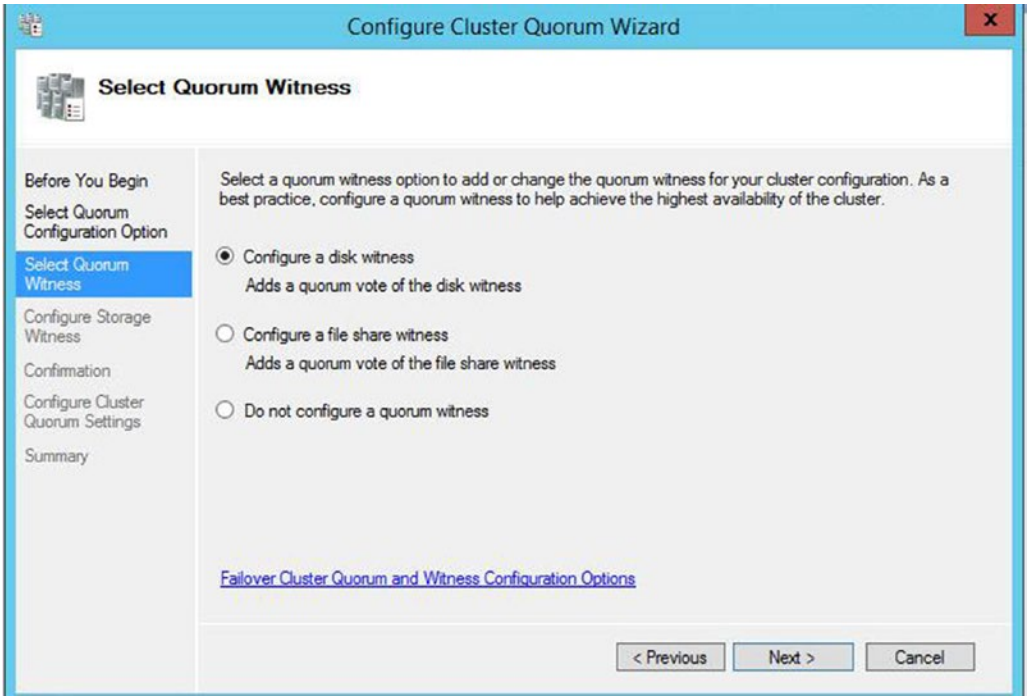

Figure 3-22. *The Select Quorum Witness page*

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 5, as illustrated in Figure 3-23.

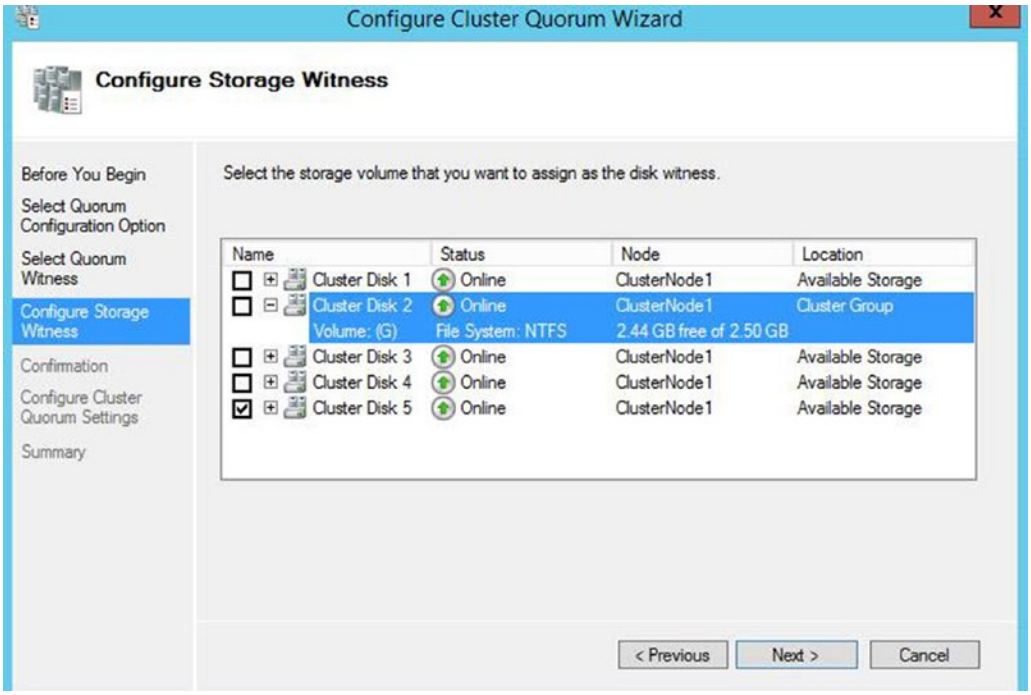

Figure 3-23. *The Configure Storage Witness page*

The Summary page of the wizard, shown in Figure 3-24, 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 in Chapter 4.

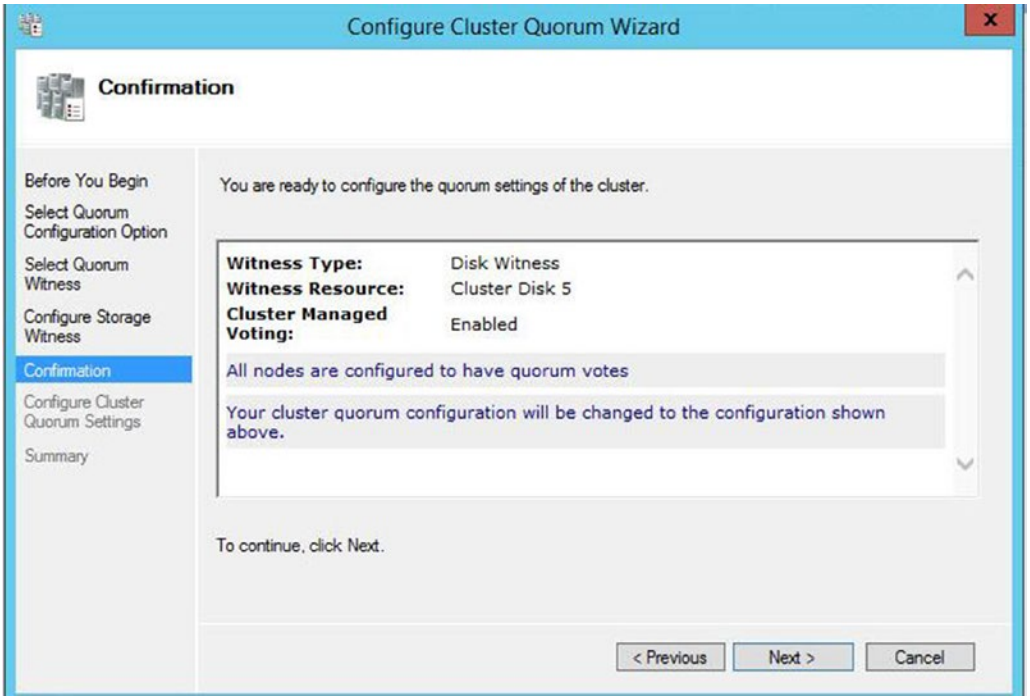

Figure 3-24. *The Summary Page*

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.

Listing 3-3. Configuring the Quorum Disk

```
Set-ClusterQuorum -Cluster PROSQLADMIN-C -NodeAndDiskMajority "Cluster Disk 5"
```

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 non-deterministic 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 3-25.

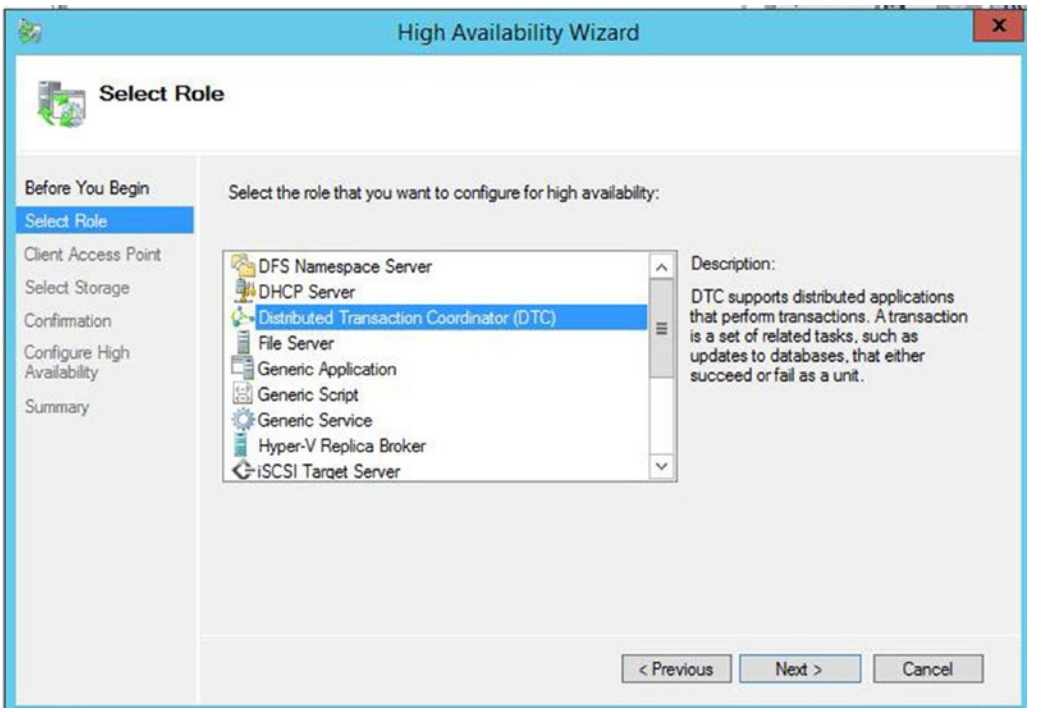

Figure 3-25. *The Select Role page*

On the Client Access Point page, illustrated in Figure 3-26, you need to enter a virtual name and IP address for MSDTC. In our case, we name it `PROSQLMSDTC-C` and assign `192.168.0.21` as the IP address. On a multi-subnet cluster, you need to provide an IP address for each network.

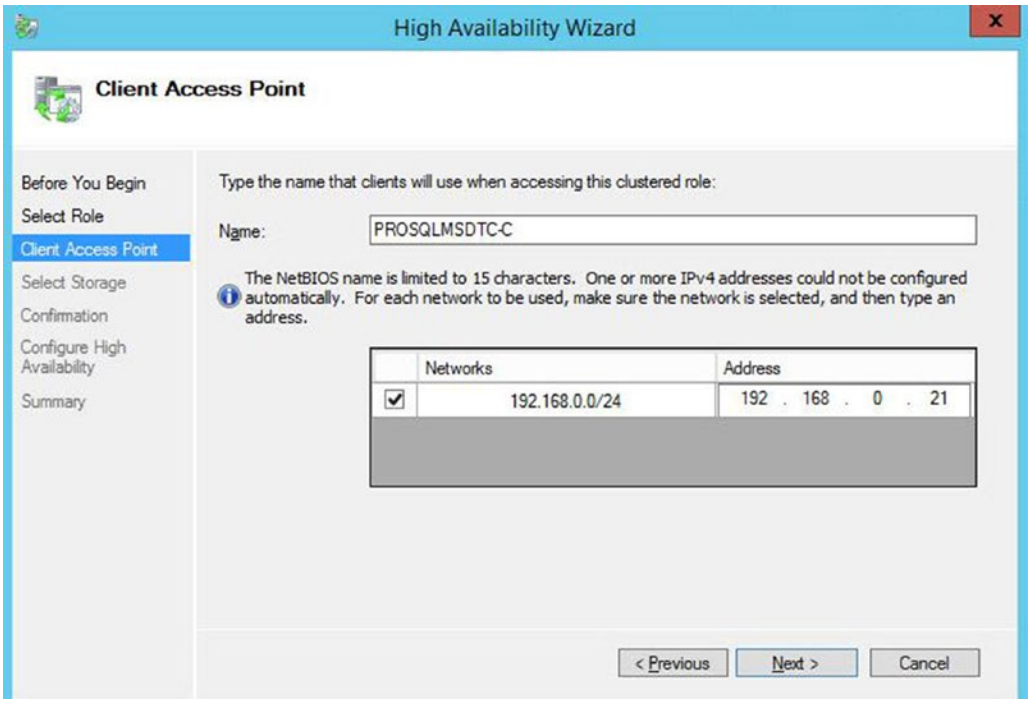

Figure 3-26. *The Client Access Point page*

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 3-27. In our case, this is Disk 4.

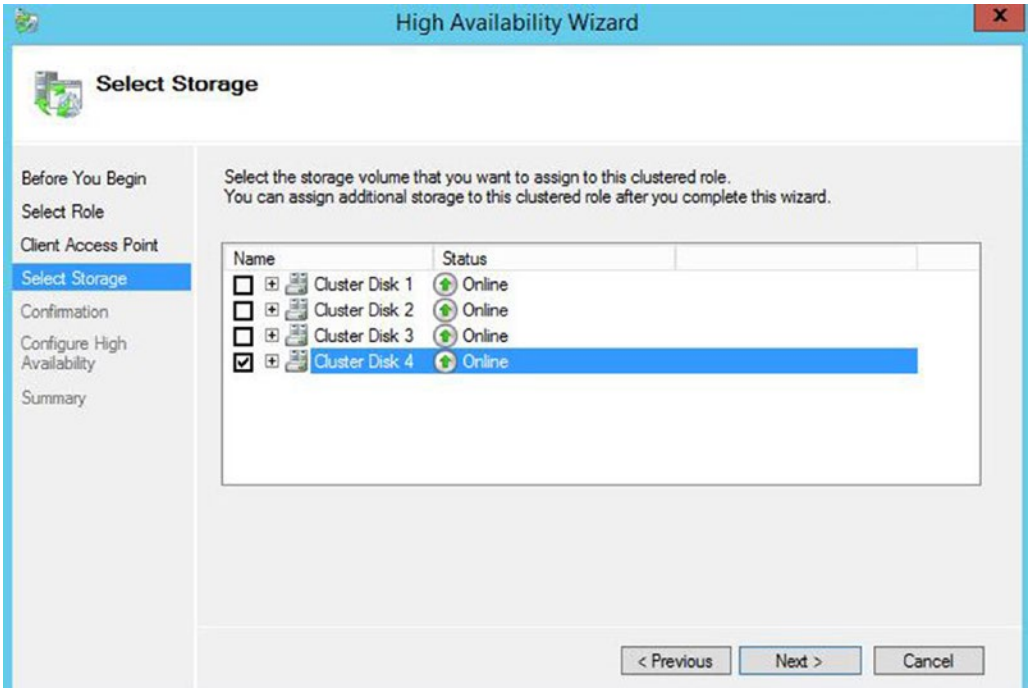

Figure 3-27. *The Select Storage page*

The Confirmation page displays an overview of the role that is about to be created, as shown in Figure 3-28.

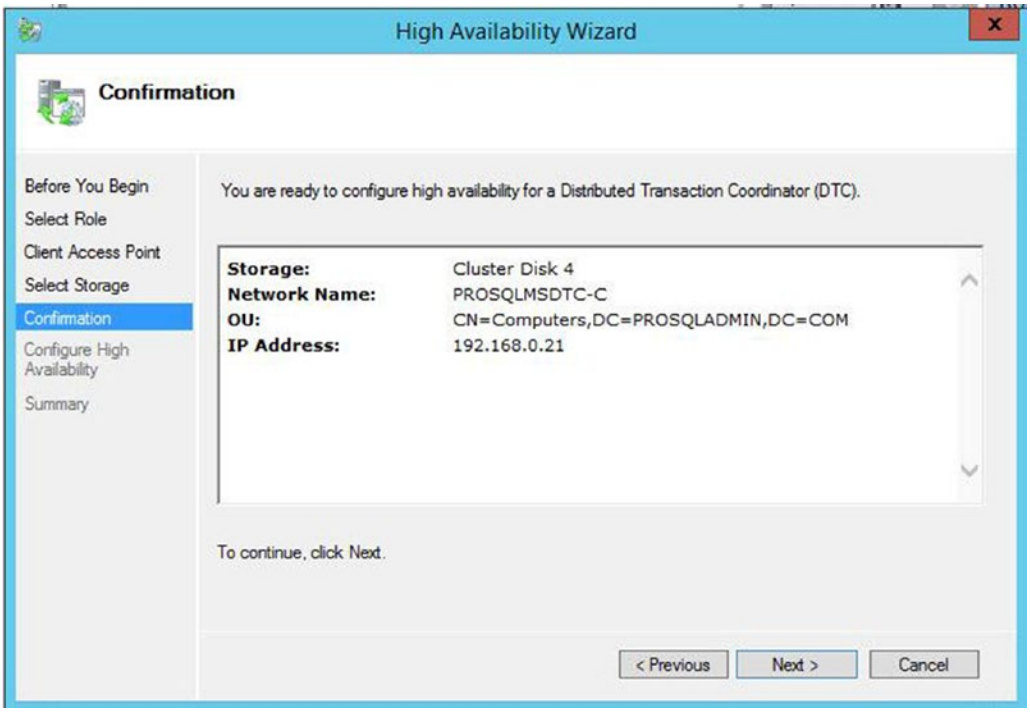

Figure 3-28. *The Confirmation page*

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.

Listing 3-4. Creating an MSDTC Role

```
#Create the Role

Add-ClusterServerRole -Name PROSQLMSDTC-C -Storage "Cluster Disk 4" -StaticAddress
192.168.0.21

#Create the DTC Resource

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

#Create the dependencies

Add-ClusterResourceDependency MSDTC-PROSQLMSDTC-C PROSQLMSDTC-C

Add-ClusterResourceDependency MSDTC-PROSQLMSDTC-C "Cluster Disk 4"

#Bring the Role online

Start-ClusterGroup PROSQLMSDTC-C
```

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 `PROSQLMSDTC-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.

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 3-29, 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.

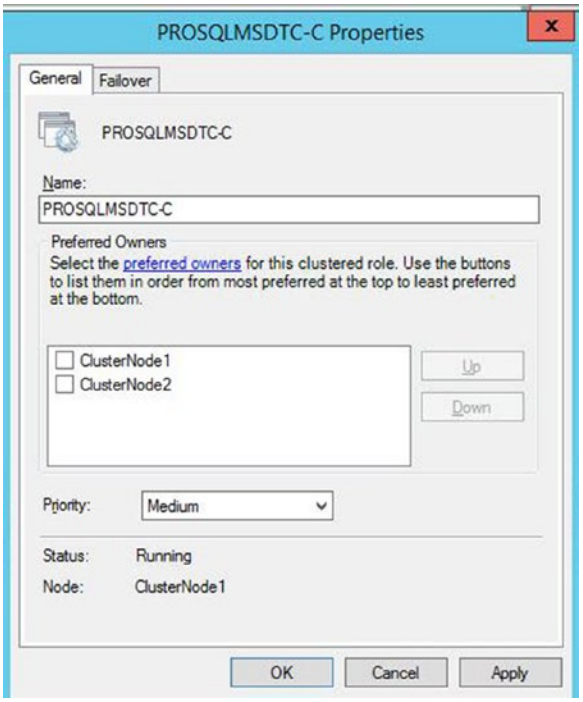

Figure 3-29. *The General tab*

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

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 3-30. 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.

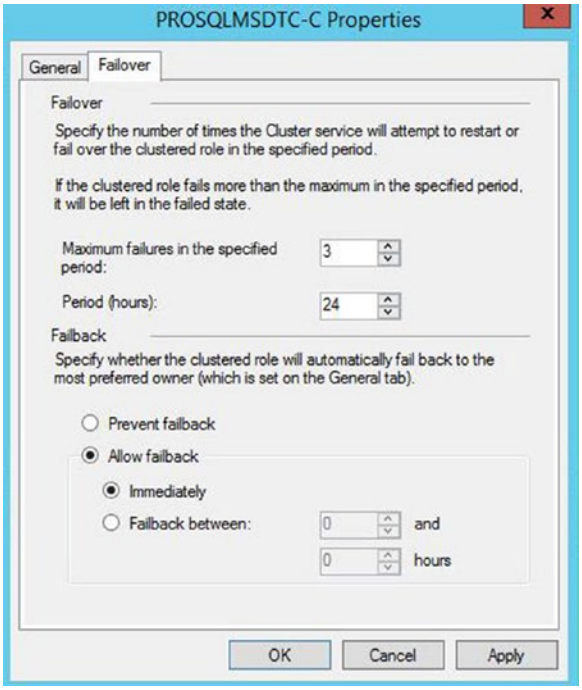

Figure 3-30. *The Failover tab*

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.

CHAPTER 4

Implementing an AlwaysOn Failover Clustered Instance

Once the cluster has been built and configured, it is time to install the AlwaysOn failover cluster instance of SQL Server. This chapter will discuss how to create an AlwaysOn Failover Clustered Instance and also demonstrate how to add a node to the cluster.

Building the Instance

To build an AlwaysOn Failover Clustered Instance, select the New SQL Server Failover Cluster Installation option from the Installation tab of the SQL Server Installation Center.

Preparation Steps

When you select this option, you invoke the Install A SQL Server Failover Cluster Wizard. The majority of the pages in this wizard are identical to the pages in the SQL Server 2014 Setup Wizard.

Cluster-Specific Steps

The first deviation from a stand-alone installation of SQL Server is the Instance Configuration page, which is illustrated in Figure 4-1. On this screen, you are prompted to enter the SQL Server Network Name as well as choose between a default instance and a named instance. The SQL Server Network Name is the virtual name, which is configured as a resource in the role. This allows client applications to connect to a server with a consistent name, regardless of the node that owns the instance.

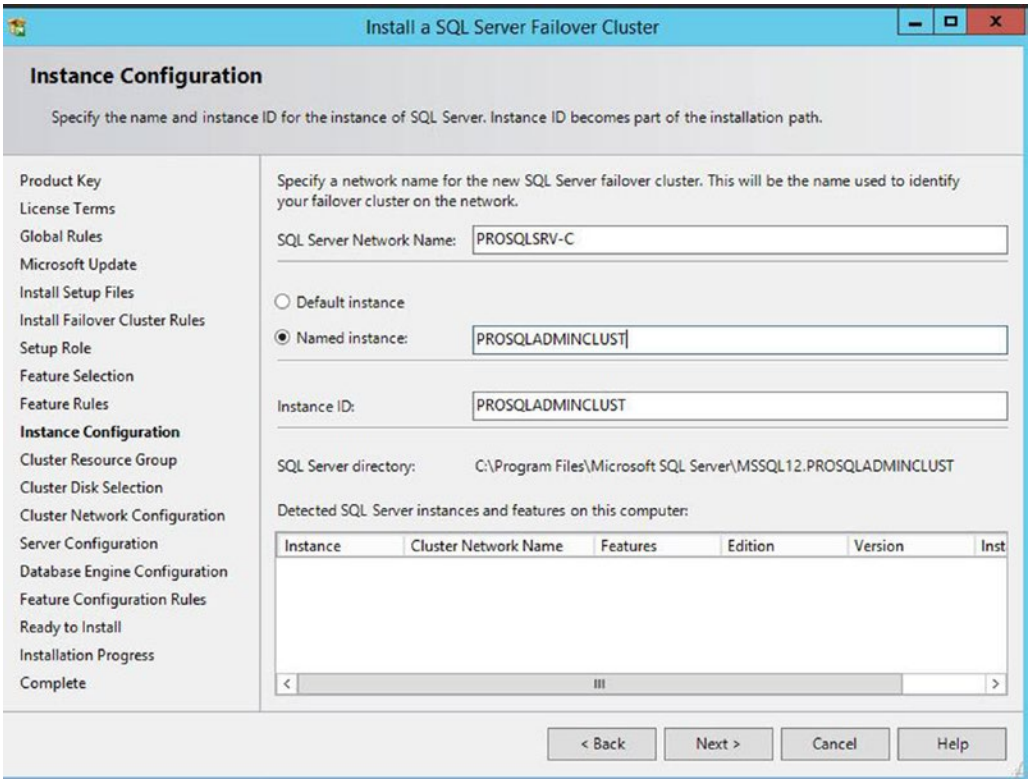

Figure 4-1. *The Instance Configuration page*

■ **Tip** If you do not have permissions to create AD objects in the OU that contains your cluster, then the VCO for the instance must already exist, and you must have the Full Control permission assigned.

On the Cluster Resource Group page, displayed in Figure 4-2, a list of roles on the local cluster is displayed. It is possible to create an empty role prior to running the installation of the instance, and if you have done so, then you can select it from the list. Alternatively, you can modify the default name supplied in the SQL Server cluster resource group name box to create a new resource group. In this demonstration, we leave the default name.

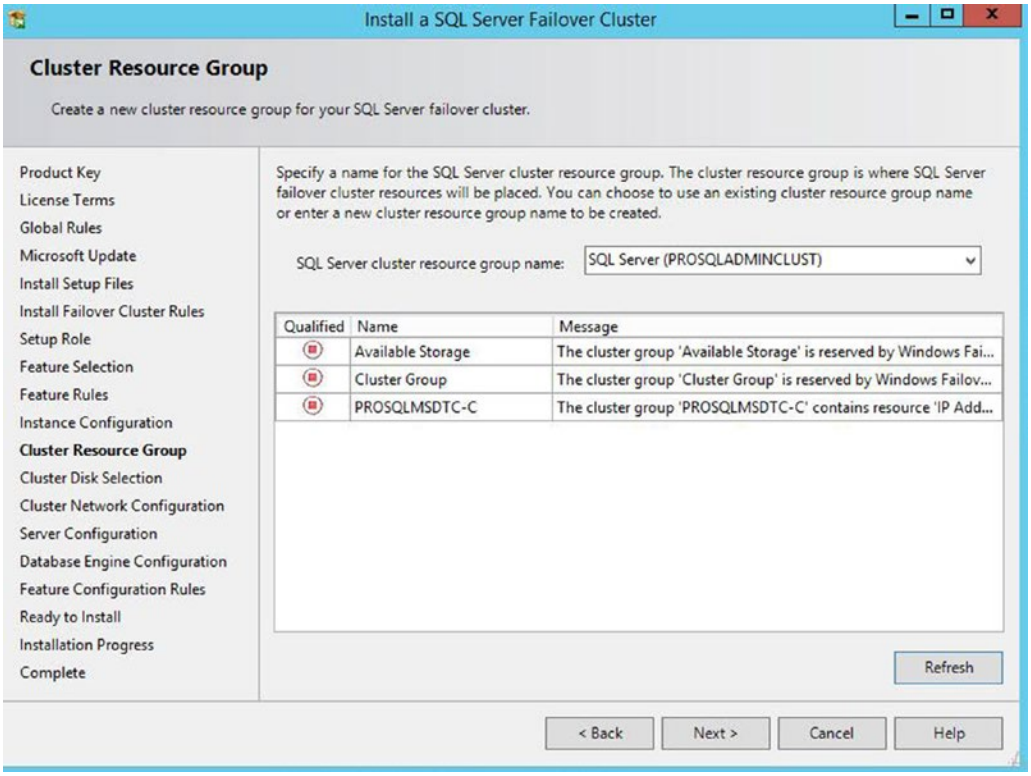

Figure 4-2. *The Cluster Resource Group page*

On the Cluster Disk Selection page of the wizard, shown in Figure 4-3, a list of cluster disks displays in the lower pane, with warnings next to any disks that you cannot select. In the top pane, you can check the disks that you want to add to the role. In our case, we select all of them, since there is one for data, one for logs, and one for TempDB.

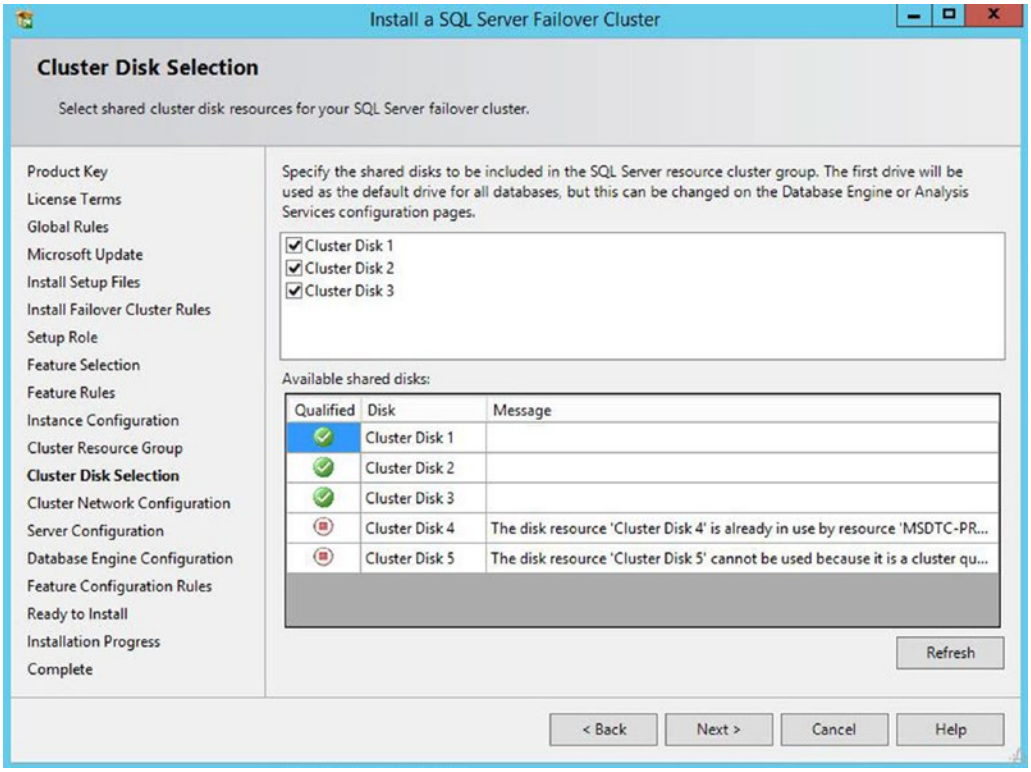

Figure 4-3. *The Cluster Disk Selection page*

On the Cluster Network Configuration page, we add an IP address for the role. In a multi-subnet cluster, we would add multiple IP addresses, one for each subnet. The Cluster Network Configuration page is illustrated in Figure 4-4.

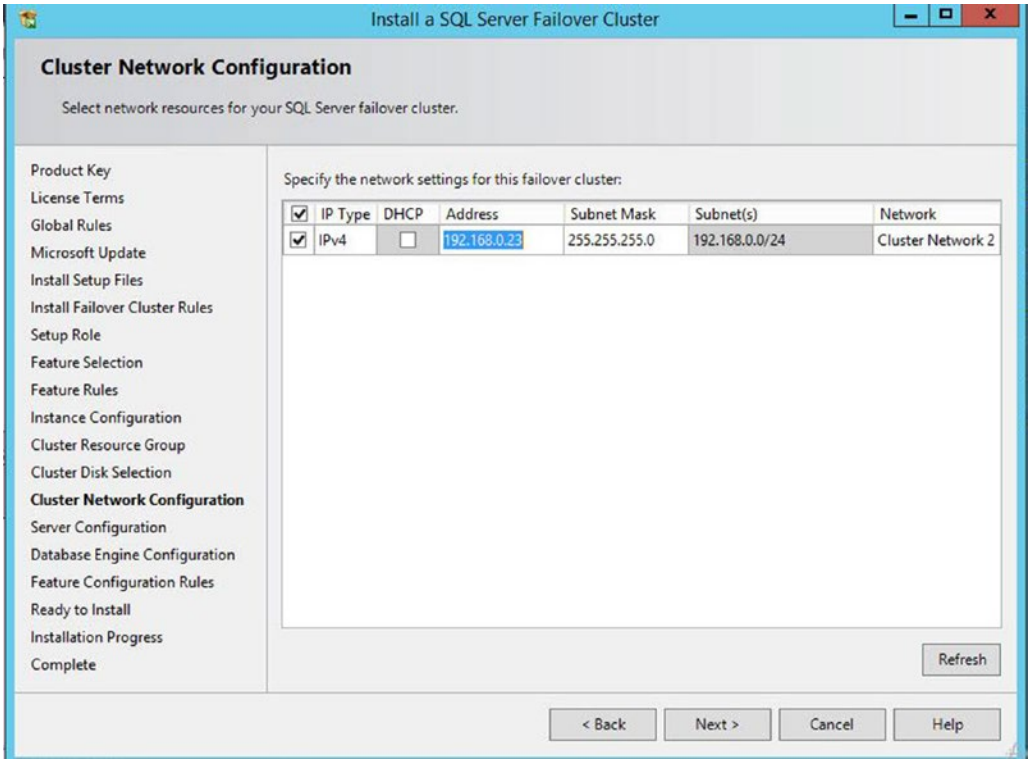

Figure 4-4. *The Cluster Network Configuration page*

On the Service Accounts tab of the Server Configuration page, the startup type for the SQL Server service and the Server Agent service can only be configured as manual, since startup is controlled by the cluster during failover. The other required service accounts are in a read-only display and cannot be configured, as shown in Figure 4-5.

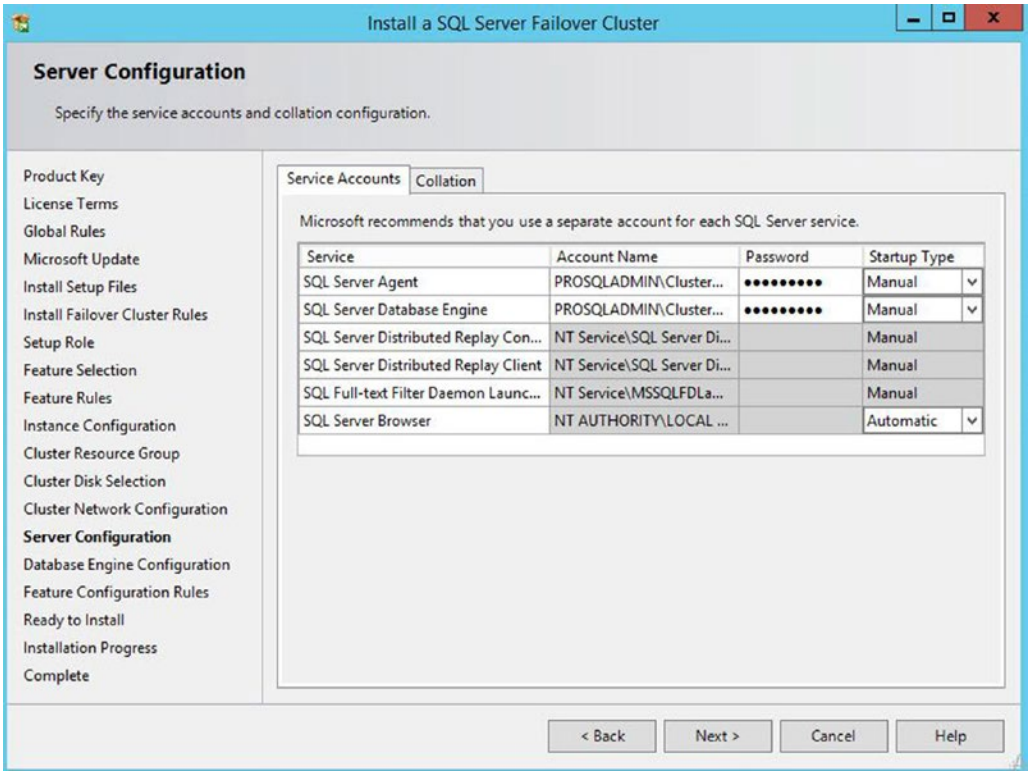

Figure 4-5. *The Sevice Account tab*

After instance installation is complete, the role is brought online and the instance is visible in Failover Cluster Manager, as shown in Figure 4-6.

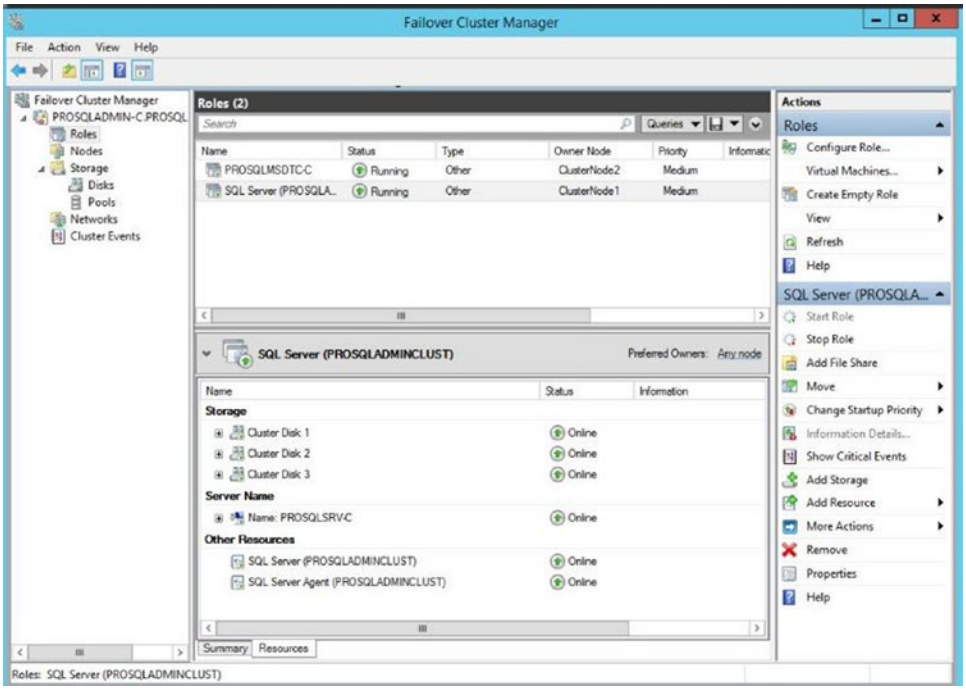

Figure 4-6. *The SQL Server role in Failover Cluster Manager*

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,Conn,ADV_SSMS,DREPLAY_CTLR,DREPLAY_CLT
/INSTANCENAME="PROSQLADMINCLUST" /SQLSVCACCOUNT="PROSQLADMIN\ClusterAdmin"
/SQLSVCPASSWORD="Pa$$w0rd" /AGTSVCACCOUNT="PROSQLADMIN\ClusterAdmin"
/AGTSVCPASSWORD="Pa$$w0rd" /SQLSYSADMINACCOUNTS="PROSQLADMIN\SQLAdmin"
/FAILOVERCLUSTERIPADDRESSES="IPv4;192.168.0.23;Cluster Network 2;255.255.255.0"
/FAILOVERCLUSTERNETWORKNAME="PROSQLSRV-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 4-7, 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.

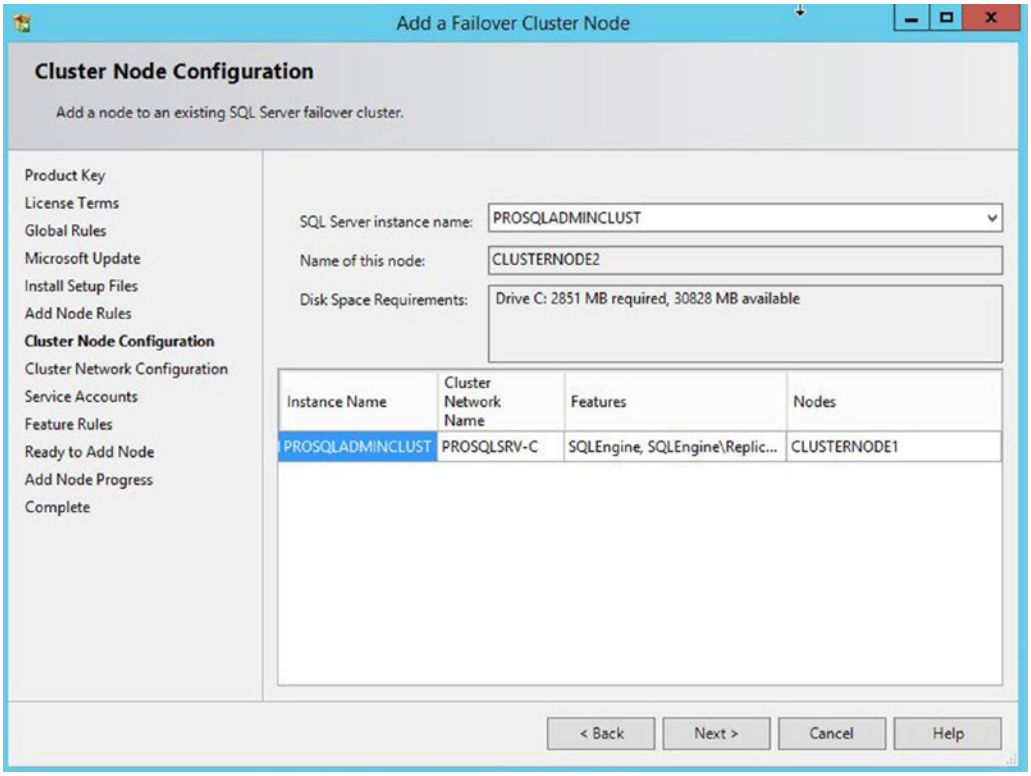

Figure 4-7. *The Cluster Node Configuration page*

On the Cluster Network Configuration page, shown in Figure 4-8, 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.

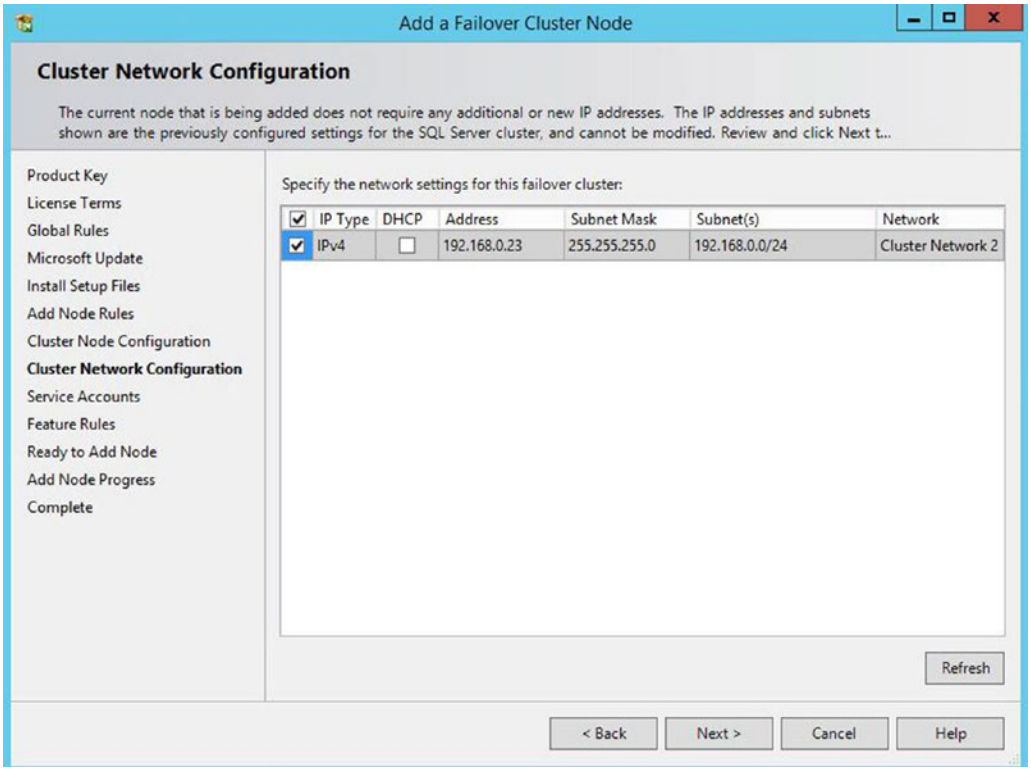

Figure 4-8. *The Cluster Network Configuration page*

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 4-9.

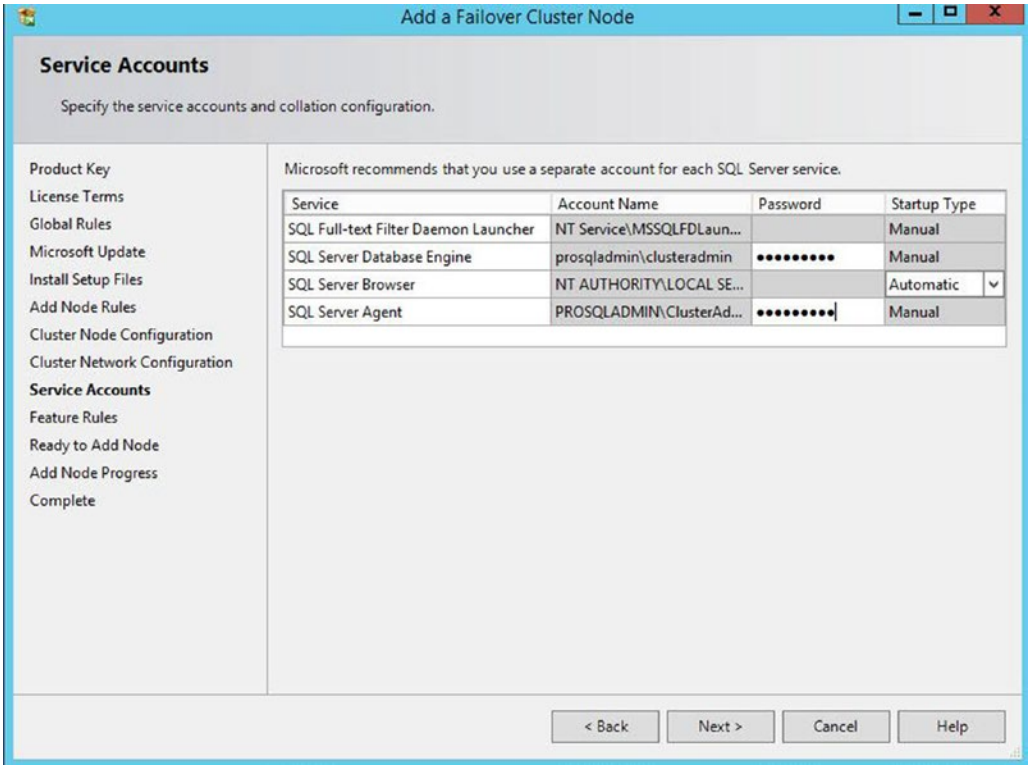

Figure 4-9. *The Service Accounts page*

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, known as the Ready To Add Node page, is illustrated in Figure 4-10. It provides a summary of the activities that take place during the installation.

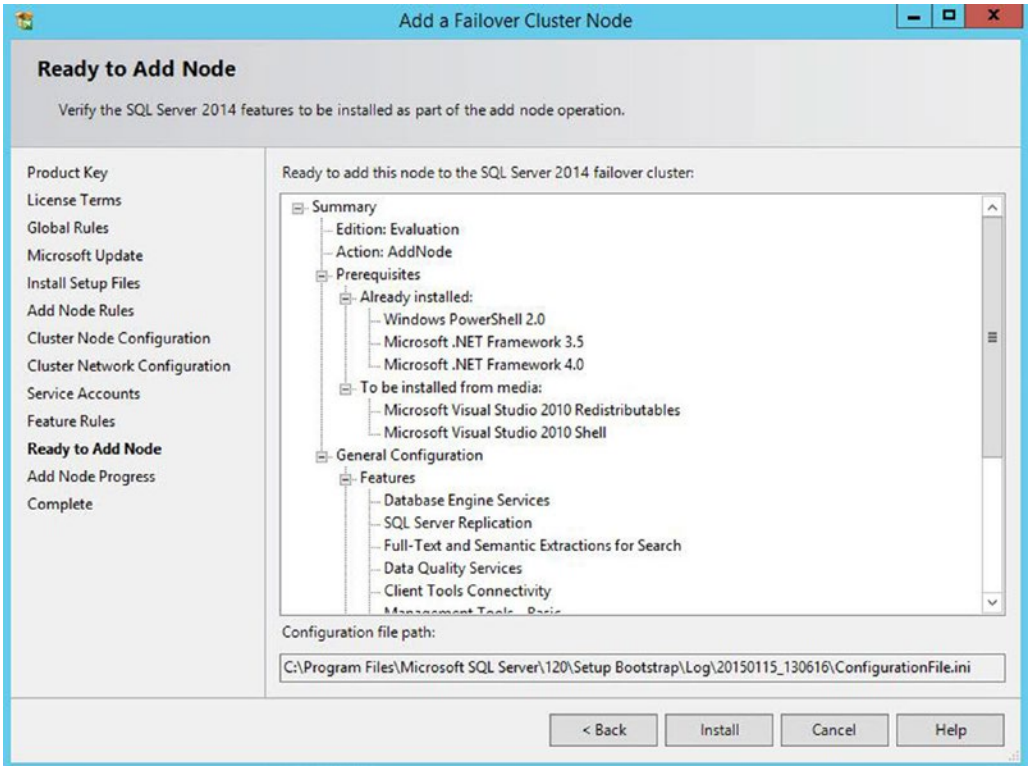

Figure 4-10. *The Read To Add Node page*

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="PROSQLADMINCLUST" /SQLSVCACCOUNT="PROSQLADMIN\ClusterAdmin"
/SQLSVCPASSWORD="Pa$$wOrd" /AGTSVCACCOUNT="PROSQLADMIN\ClusterAdmin"
/AGTSVCPASSWORD="Pa$$wOrd"
/FAILOVERCLUSTERIPADDRESSES="IPv4;192.168.0.23;Cluster Network
2;255.255.255.0" /CONFIRMIPDEPENDENCYCHANGE=0 /qs
```

Summary

An AlwaysOn 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 InstalFailoverCluster action, specifying the /FAILOVERCLUSTERIPADDRESSES, /FAILOVERCLUSTERNETWORKNAME, and /INSTALLSQLDATADIR parameters, in addition the parameters required for a stand-alone instance build.

CHAPTER 5

Implementing 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. We also discuss using availability groups to scale out read-only workloads.

Note For the demonstrations in this chapter, we use a domain that contains a domain controller and a two-node cluster. These servers are in a site called `Site1`. A second site, called `Site2`, contains a third server. During the course of the chapter, we add this as a node to the cluster. The cluster has no shared storage for data and there is no AlwaysOn failover clustered instance. Each node has a stand-alone instance of SQL Server installed on it named `ClusterNode1\PrimaryReplica`, `ClusterNode2\SyncHA`, and `ClusterNode3\AsyncDR`, respectively.

Implementing High Availability with AlwaysOn 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 Chapter5App1Customers ;
GO

ALTER DATABASE Chapter5App1Customers SET RECOVERY FULL ;
GO

USE Chapter5App1Customers
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 Chapter5App1Sales ;
GO

ALTER DATABASE Chapter5App1Sales SET RECOVERY FULL ;
GO

USE Chapter5App1Sales
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 Chapter5App2Customers ;
GO

ALTER DATABASE Chapter5App2Customers SET RECOVERY FULL ;
GO

USE Chapter5App2Customers
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.

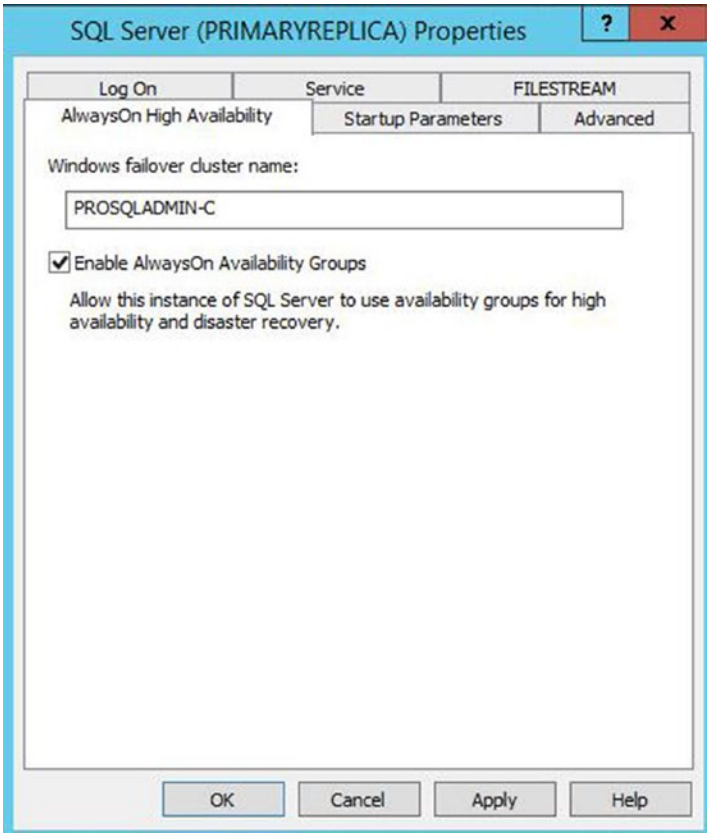

Figure 5-1. *The AlwaysOn High Avaiability tab*

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 `Chapter5App1Customers` and `Chapter5App1Sales` databases. We do this by running the script in Listing 5-3.

Listing 5-3. Backing Up the Databases

```
BACKUP DATABASE Chapter5App1Customers
TO  DISK = N'C:\Backups\Chapter5App1Customers.bak'
WITH NAME = N'Chapter5App1Customers-Full Database Backup' ;
GO

BACKUP DATABASE Chapter5App1Sales
TO  DISK = N'C:\Backups\Chapter5App1Sales.bak'
WITH NAME = N'Chapter5App1Sales-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.

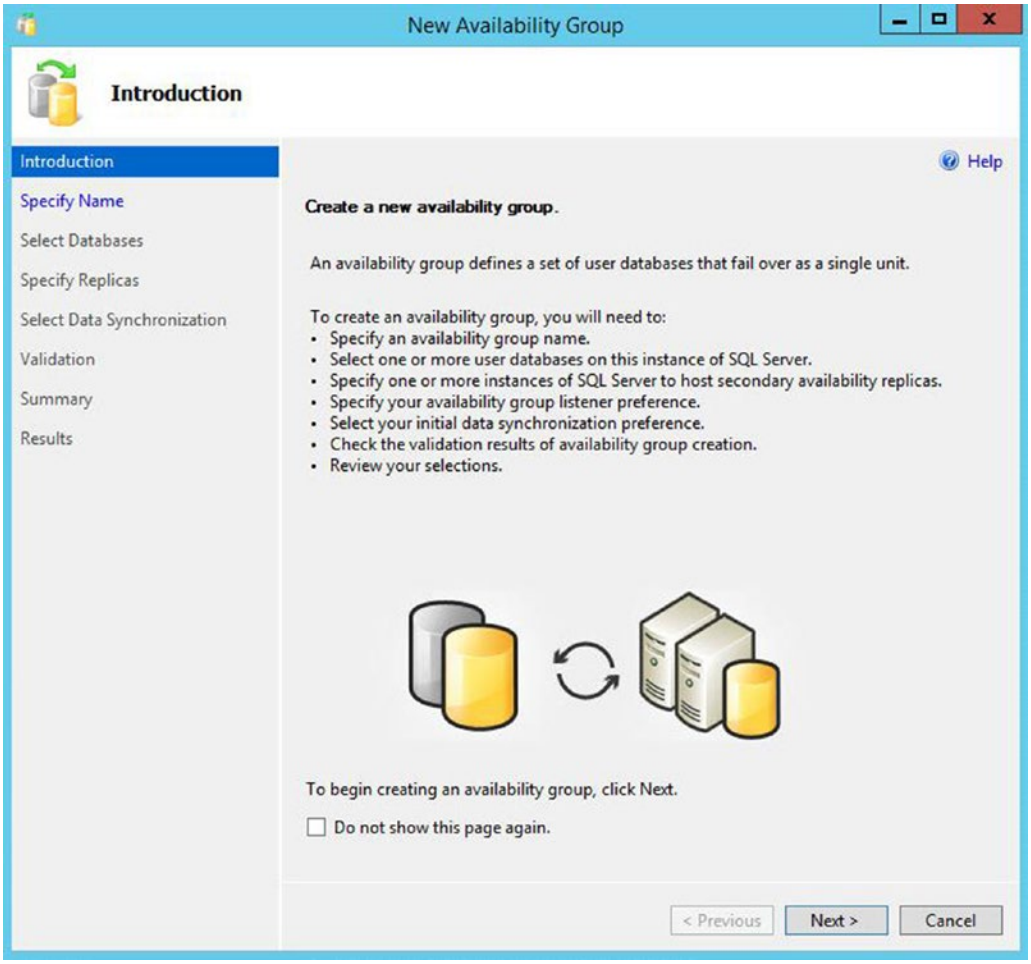

Figure 5-2. *The Introduction page*

On the Specify Name page (see Figure 5-3), we are prompted to enter a name for our availability group.

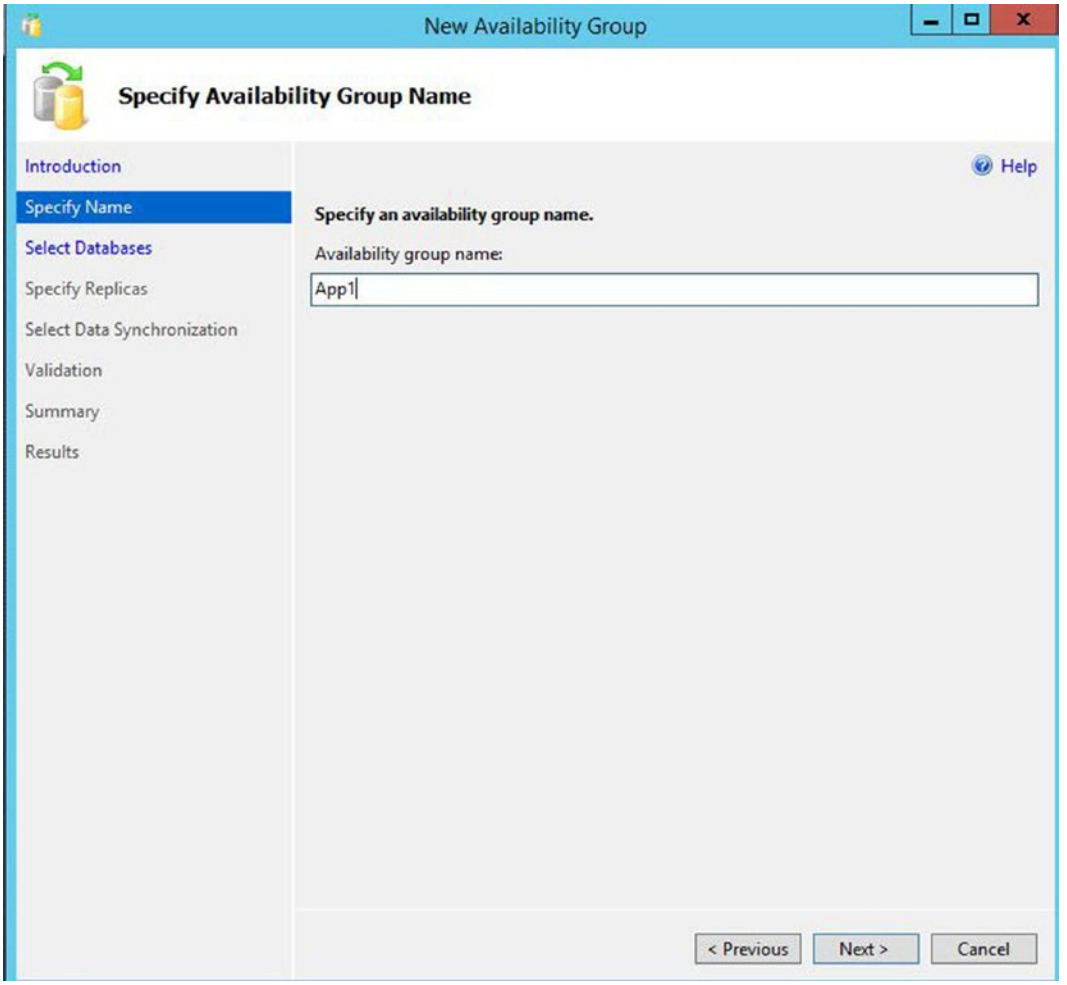

Figure 5-3. *The Specify Name page*

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 `Chapter5App2Customers` database, because we have not yet taken a full backup of the database.

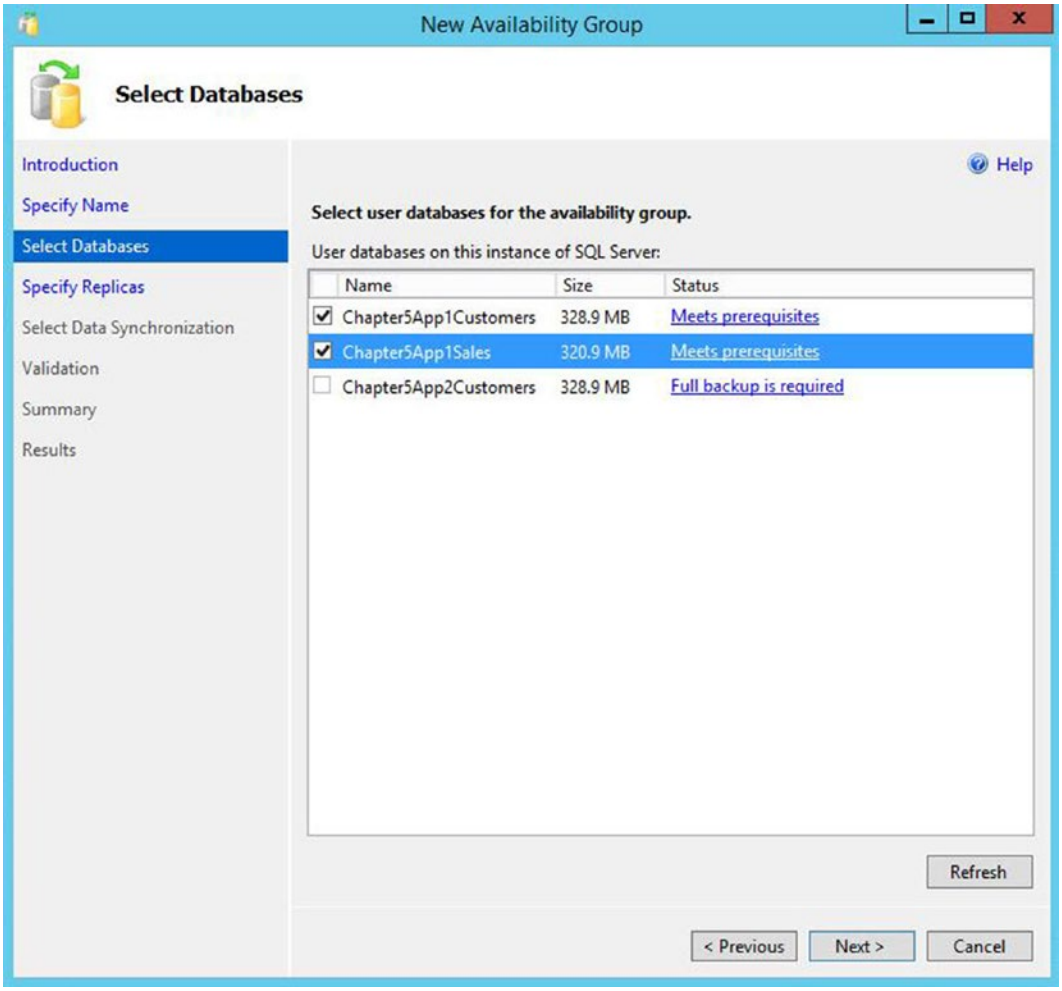

Figure 5-4. *The Select Database page*

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.

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.

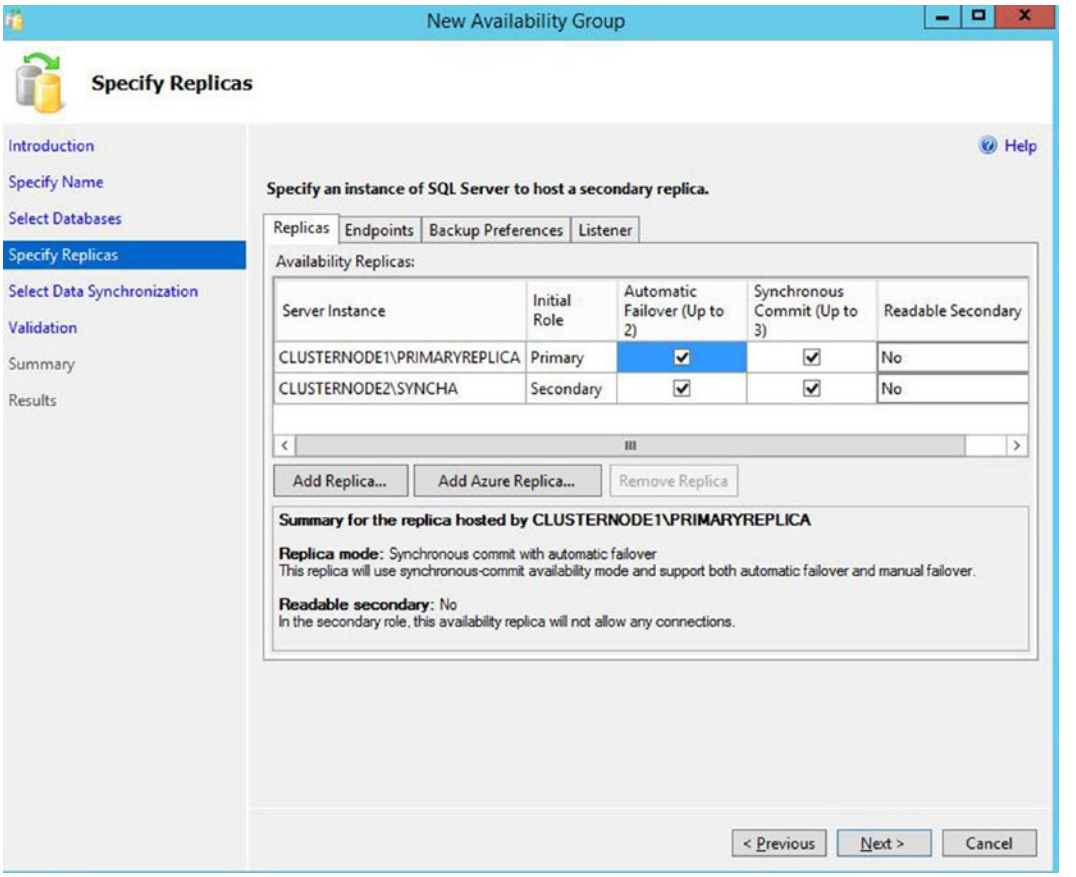

Figure 5-5. *The Replicas tab*

■ **Note** Using secondary replicas for read-only workloads is discussed in more depth in the Adding AlwaysOn Readable Secondary Replicas section of this chapter.

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.

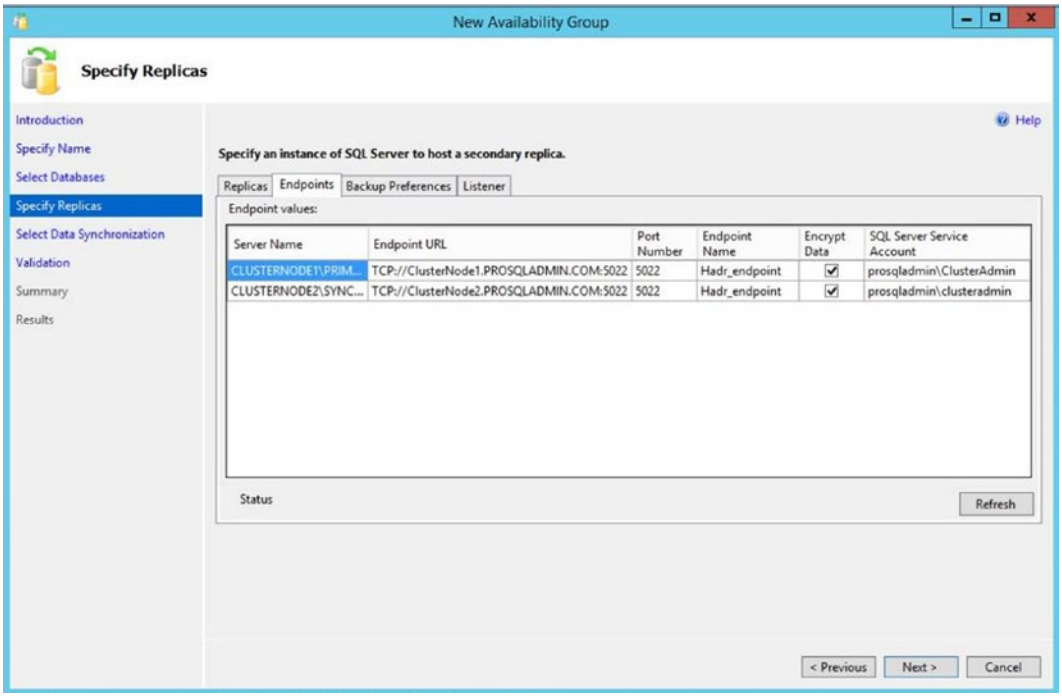

Figure 5-6. *The Endpoints tab*

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.

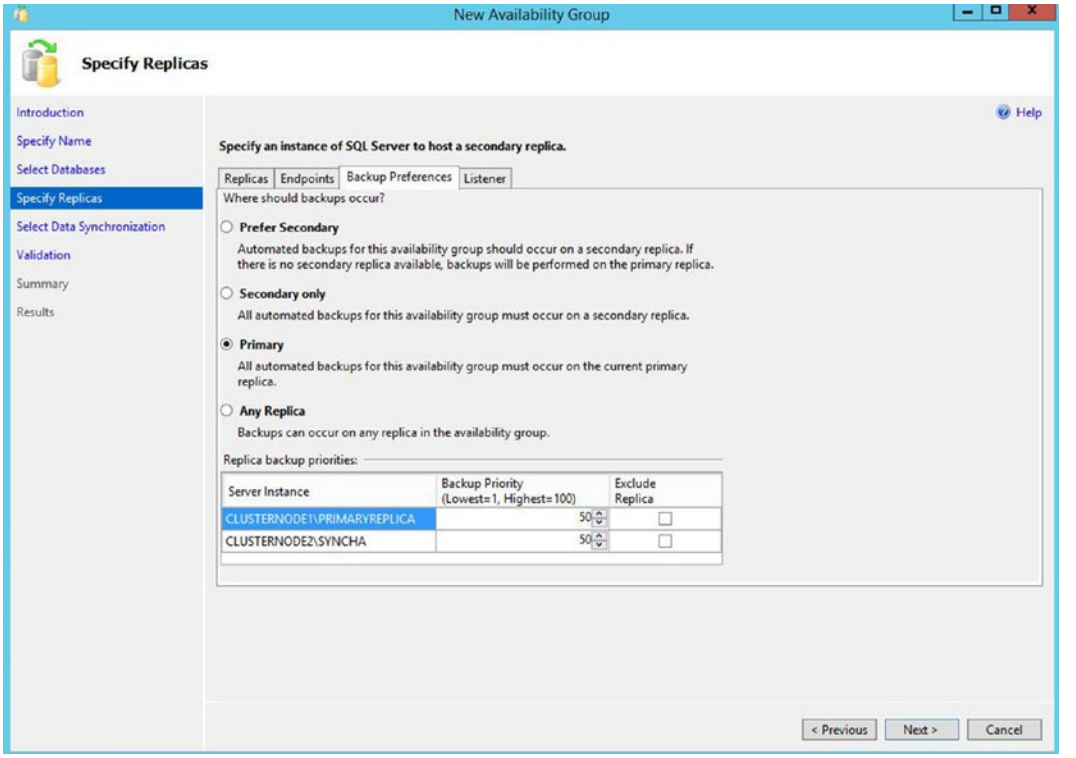

Figure 5-7. *The Backup Preferences tab*

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 accidently 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.

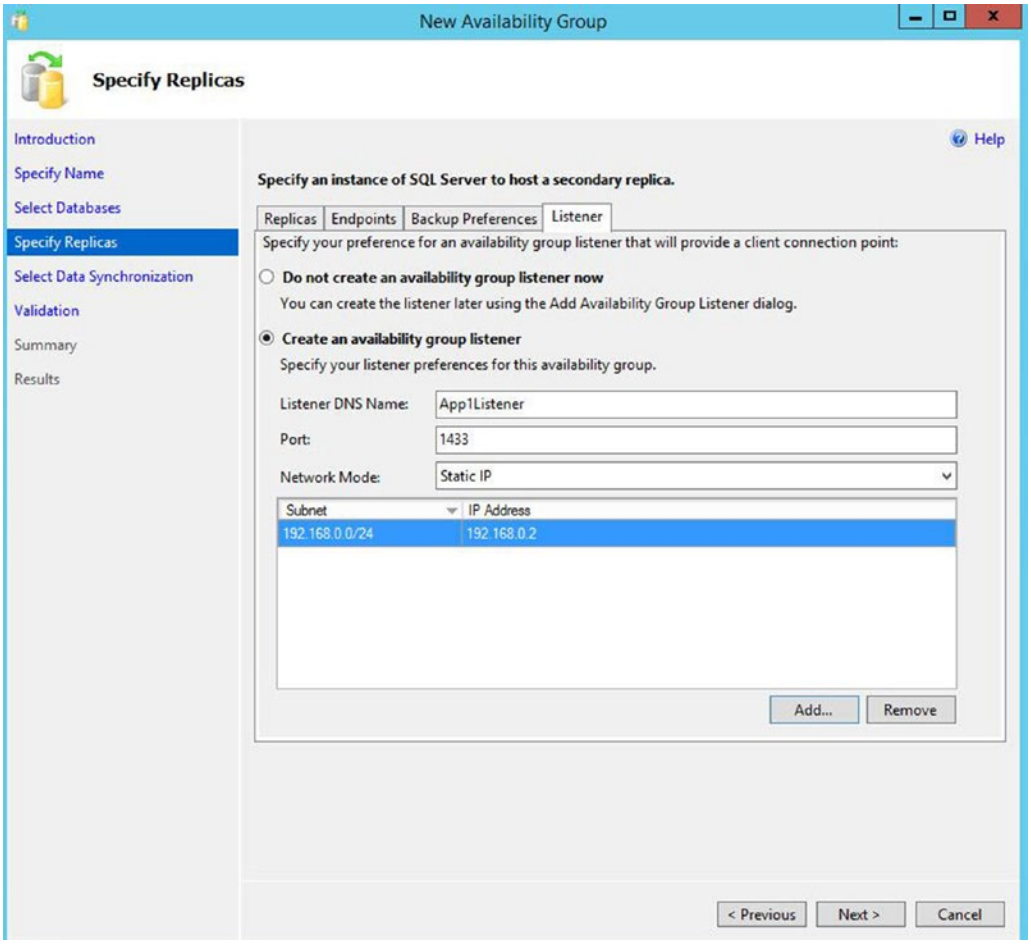

Figure 5-8. *The Listener tab*

■ **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.

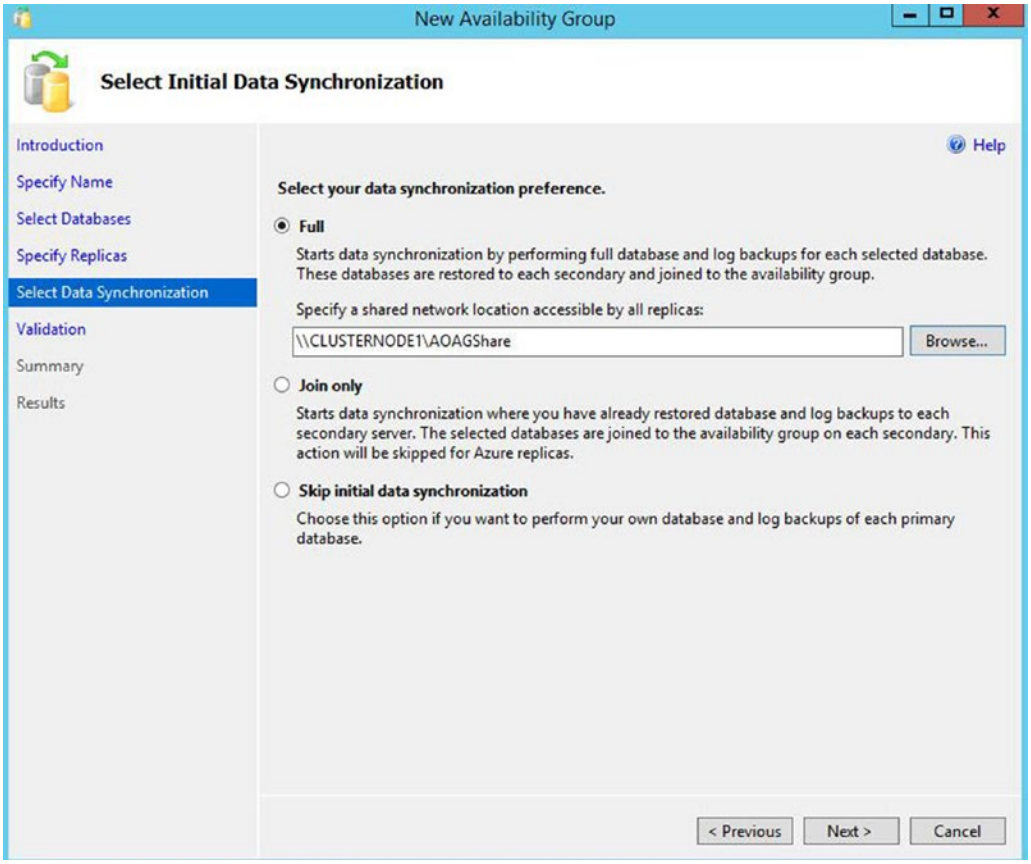

Figure 5-9. *The Select Data Synchronization page*

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.

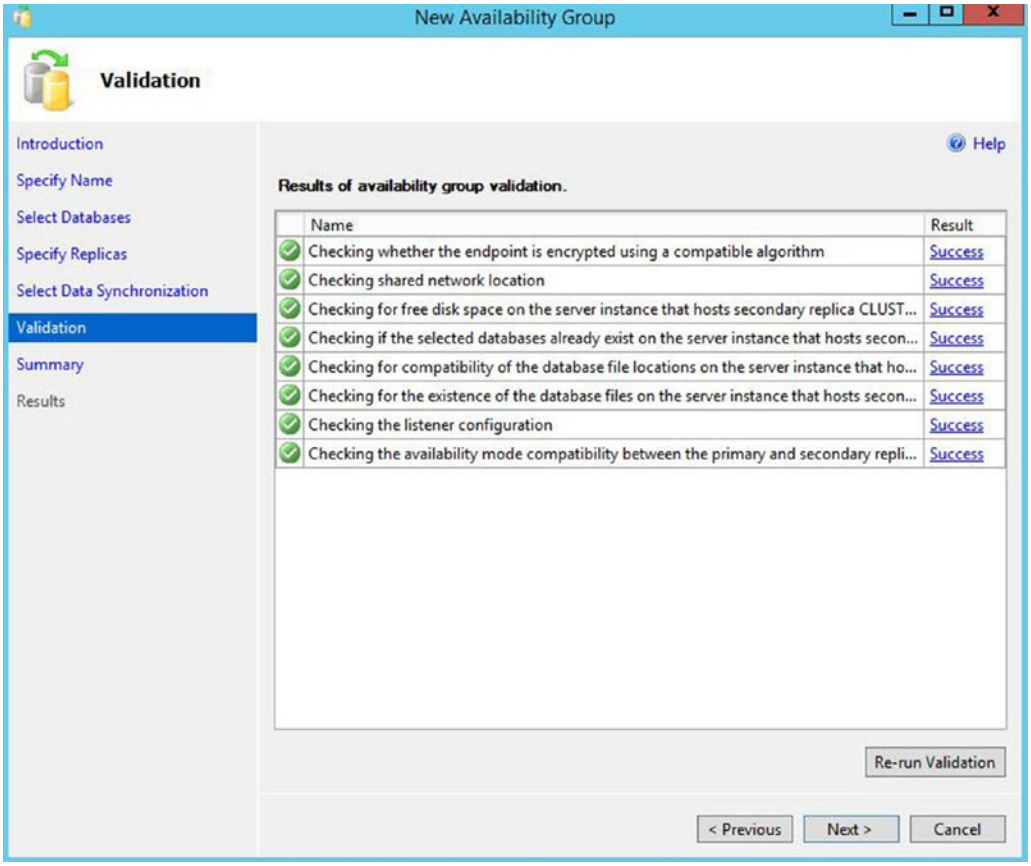

Figure 5-10. *The Validation page*

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 investigated 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 Avaiability Group

:Connect CLUSTERNODE1\PRIMARYREPLICA

USE master
GO

CREATE AVAILABILITY GROUP App1
WITH (AUTOMATED_BACKUP_PREFERENCE = PRIMARY)
FOR DATABASE Chapter5App1Customers, Chapter5App1Sales
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 Chapter5App1Customers
TO  DISK = N'\\CLUSTERNODE1\AOAGShare\Chapter5App1Customers.bak'
WITH  COPY_ONLY, FORMAT, INIT, REWIND, COMPRESSION,  STATS = 5 ;
GO

BACKUP LOG Chapter5App1Customers
TO  DISK = N'\\CLUSTERNODE1\AOAGShare\Chapter5App1Customers.trn'
WITH NOSKIP, REWIND, COMPRESSION,  STATS = 5 ;
GO

--Restore Database and Log (First database)

:Connect CLUSTERNODE2\SYNCHA

RESTORE DATABASE Chapter5App1Customers
FROM  DISK = N'\\CLUSTERNODE1\AOAGShare\Chapter5App1Customers.bak'
WITH  NORECOVERY,  STATS = 5 ;
GO

RESTORE LOG Chapter5App1Customers
FROM  DISK = N'\\CLUSTERNODE1\AOAGShare\Chapter5App1Customers.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 Chapter5App1Customers SET HADR AVAILABILITY GROUP = App1;

GO

--Back Up Database and Log (Second database)

:Connect CLUSTERNODE1\PRIMARYREPLICA

BACKUP DATABASE Chapter5App1Sales
TO  DISK = N'\\CLUSTERNODE1\AOAGShare\Chapter5App1Sales.bak'
WITH  COPY_ONLY, FORMAT, INIT, REWIND, COMPRESSION,  STATS = 5 ;
GO

BACKUP LOG Chapter5App1Sales
TO  DISK = N'\\CLUSTERNODE1\AOAGShare\Chapter5App1Sales.trn'
WITH NOSKIP, REWIND, COMPRESSION,  STATS = 5 ;
GO

--Restore Database and Log (Second database)

:Connect CLUSTERNODE2\SYNCHA

RESTORE DATABASE Chapter5App1Sales
FROM  DISK = N'\\CLUSTERNODE1\AOAGShare\Chapter5App1Sales.bak'
WITH  NORECOVERY, STATS = 5 ;
GO
```

```
RESTORE LOG Chapter5App1Sales
FROM  DISK = N'\\CLUSTERNODE1\AOAGShare\Chapter5App1Sales.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 Chapter5App1Sales 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 AVAIABILITY 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
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 format: TCP://ServerName:Port.	-
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.	-
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 Arguement*

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 Chapter5App2Customers 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 Chapter5App2Customers TO  DISK = N'\\CLUSTERNODE1\AOAGShare\
Chapter5App2Customers.bak' WITH  COPY_ONLY, FORMAT, INIT, REWIND, COMPRESSION,  STATS = 5 ;
GO

BACKUP LOG Chapter5App2Customers TO  DISK = N'\\CLUSTERNODE1\AOAGShare\
Chapter5App2Customers.trn' WITH NOSKIP, REWIND, COMPRESSION,  STATS = 5 ;
GO

--Restore Database and Log

:Connect CLUSTERNODE2\SYNCHA
```

```
RESTORE DATABASE Chapter5App2Customers FROM  DISK = N'\\CLUSTERNODE1\AOAGShare\
Chapter5App2Customers.bak' WITH  NORECOVERY, STATS = 5 ;
GO

RESTORE LOG Chapter5App2Customers FROM  DISK = N'\\CLUSTERNODE1\AOAGShare\
Chapter5App2Customers.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.

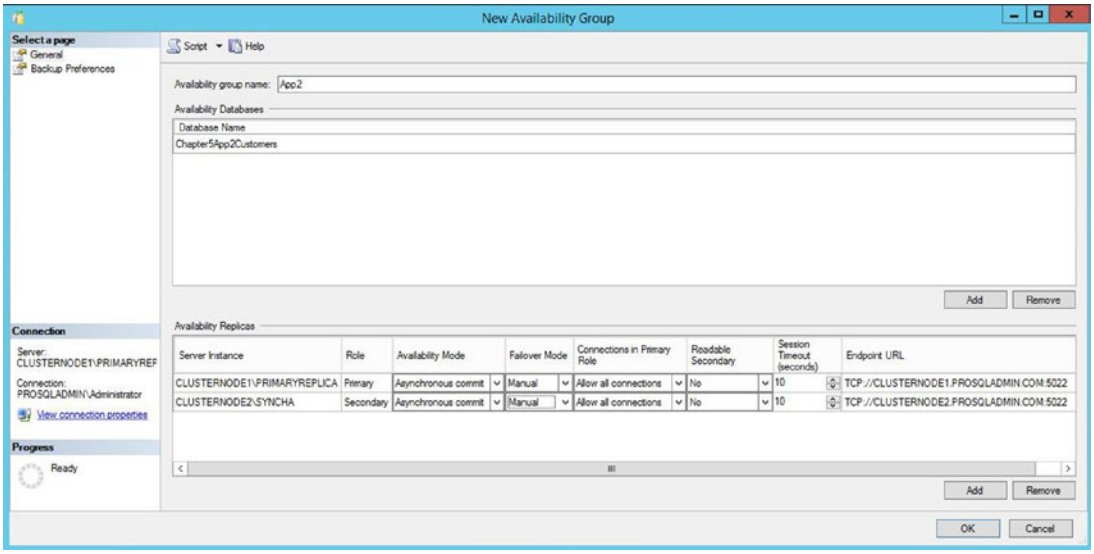

Figure 5-11. *The New Availability Group dialog box*

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.

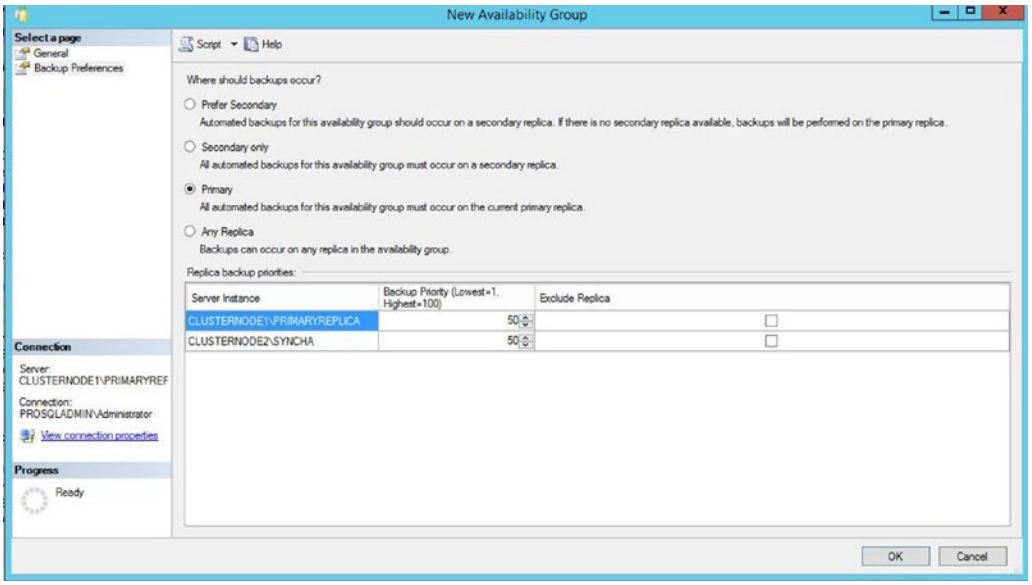

Figure 5-12. *The Backup Preferences tab*

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.

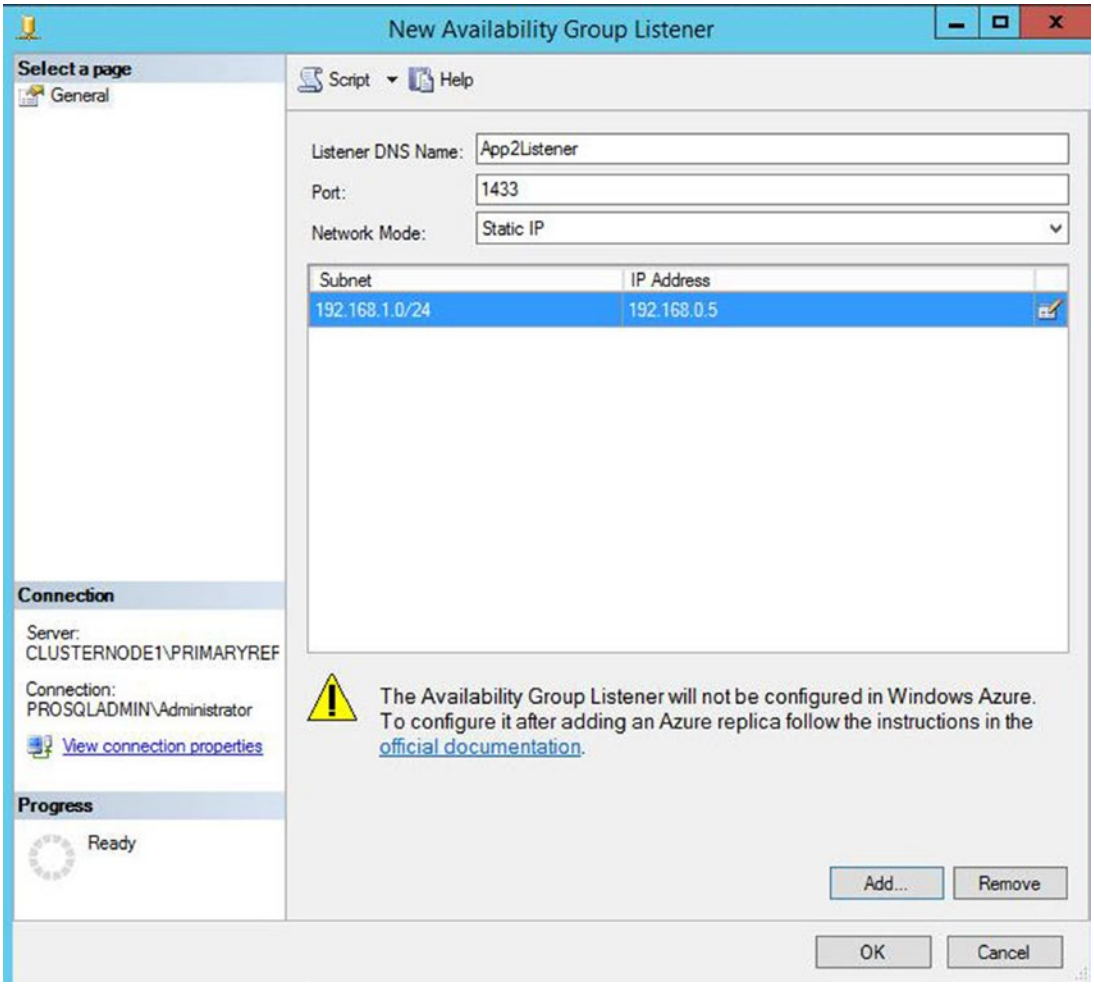

Figure 5-13. *The New Availability Group Listener dialog box*

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 `Chapter5App2Customers` database (which is in Asynchronous Commit mode) and then against the `Chapter5App1Customers` 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 Chapter5App2Customers
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 Chapter5App1Customers
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 almost three times slower when the Availability Group was operating in Synchronous Commit mode, the `insert` was over six times slower, and the `delete` was also marginally slower. In productions environments in which I have implemented availability groups in Synchronous Commit mode, they have generally been around two times slower than in Asynchronous Commit mode.

Listing 5-7. Results of Performance Benchmark

```
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)
```

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 only be able to host the instance 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.

Implementing Disaster Recovery with Availability Group

Now that we have successfully implemented high availability for the `Chapter5App1Customers` and `Chapter5App1Sales` databases through the `App1` availability group, we need to 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.

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 5-14. On this page, you need to enter the server name of the node that you plan to add to the cluster.

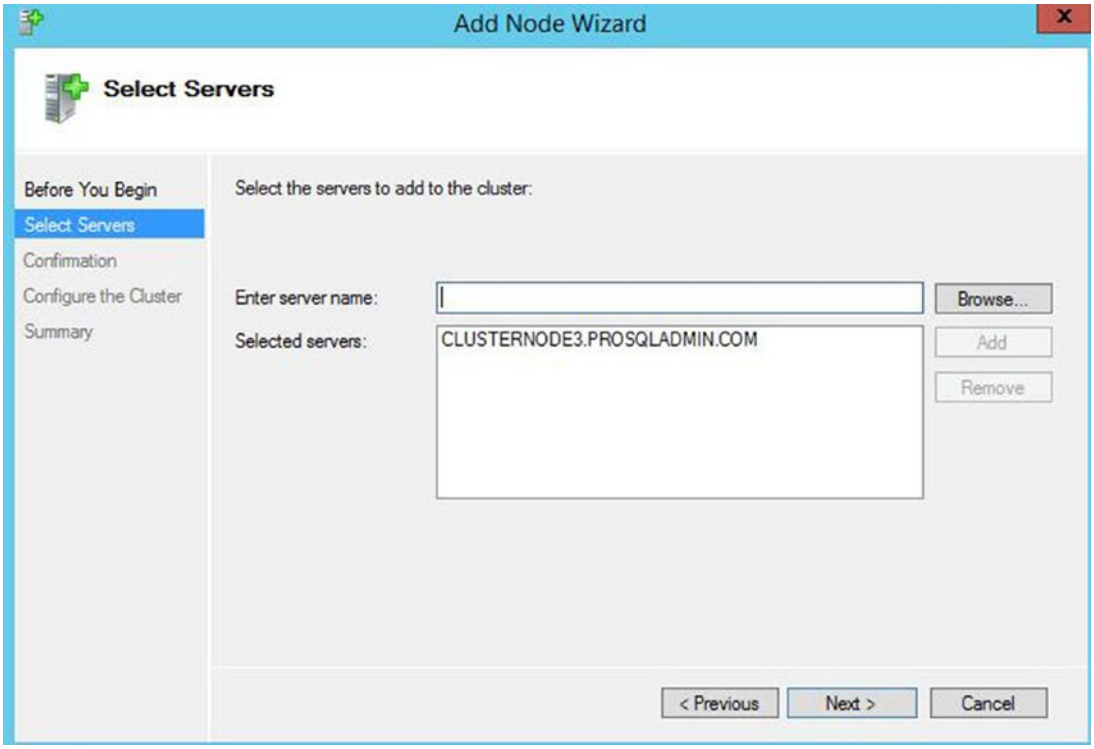

Figure 5-14. *The Select Servers page*

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 5-3.

Table 5-3. *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 5-15.

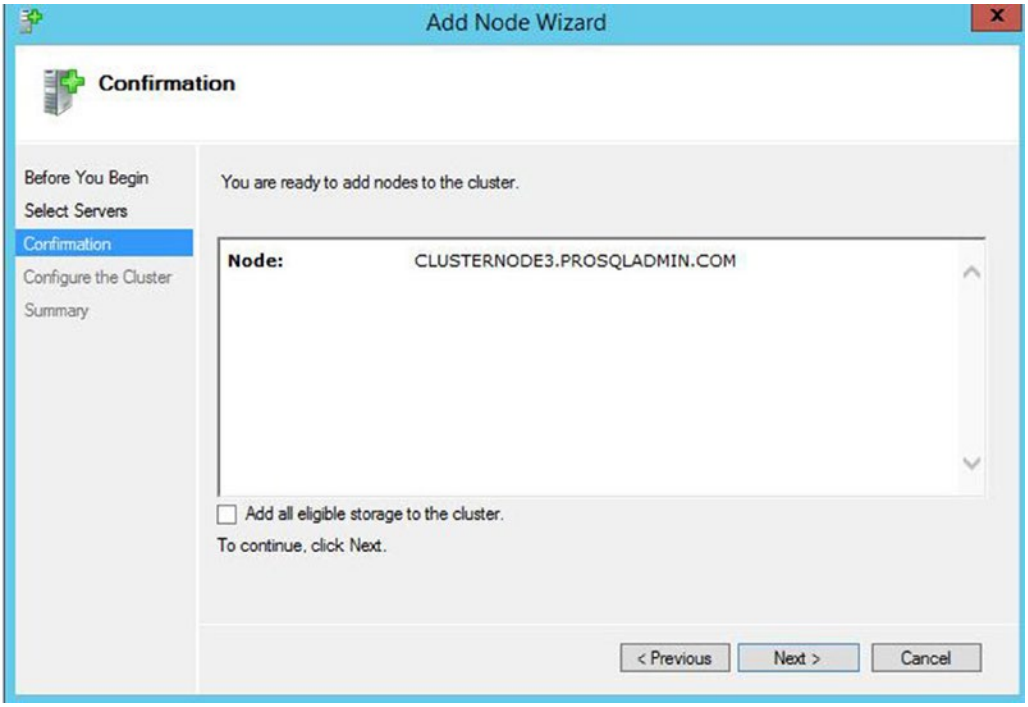

Figure 5-15. *Confirmation page*

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 are 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 inter-site 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 5-16. We select the Advanced Quorum Configuration option.

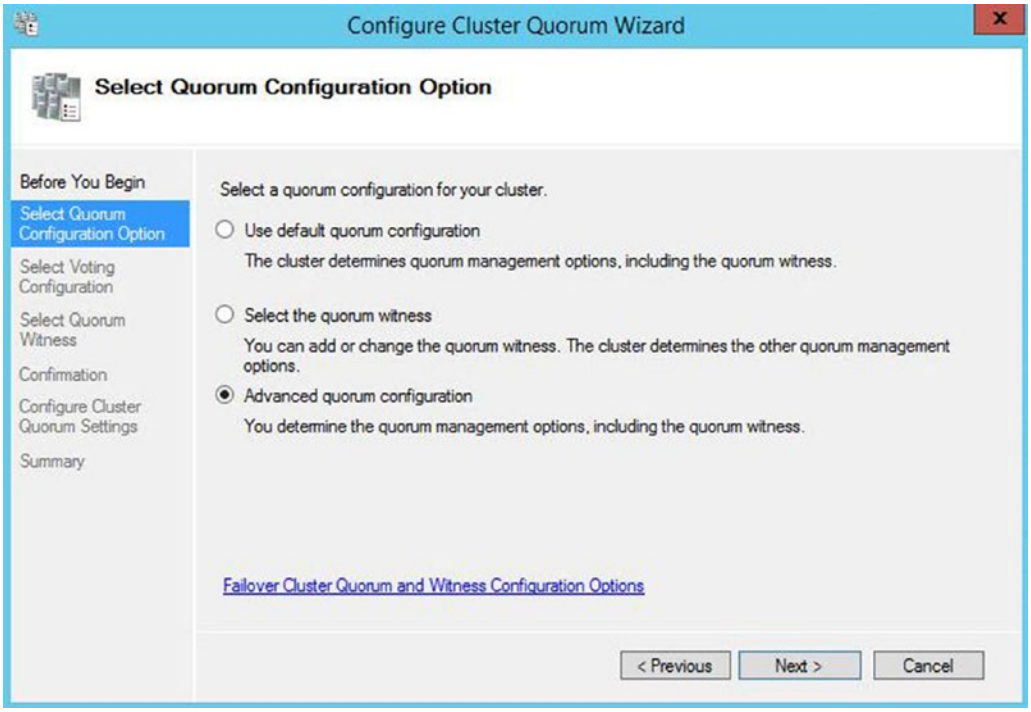

Figure 5-16. *The Select Quorum Configuration Option page*

On the Select Voting Configuration page, we choose to select nodes and remove the vote from `CLUSTERNODE3`. This is demonstrated in Figure 5-17.

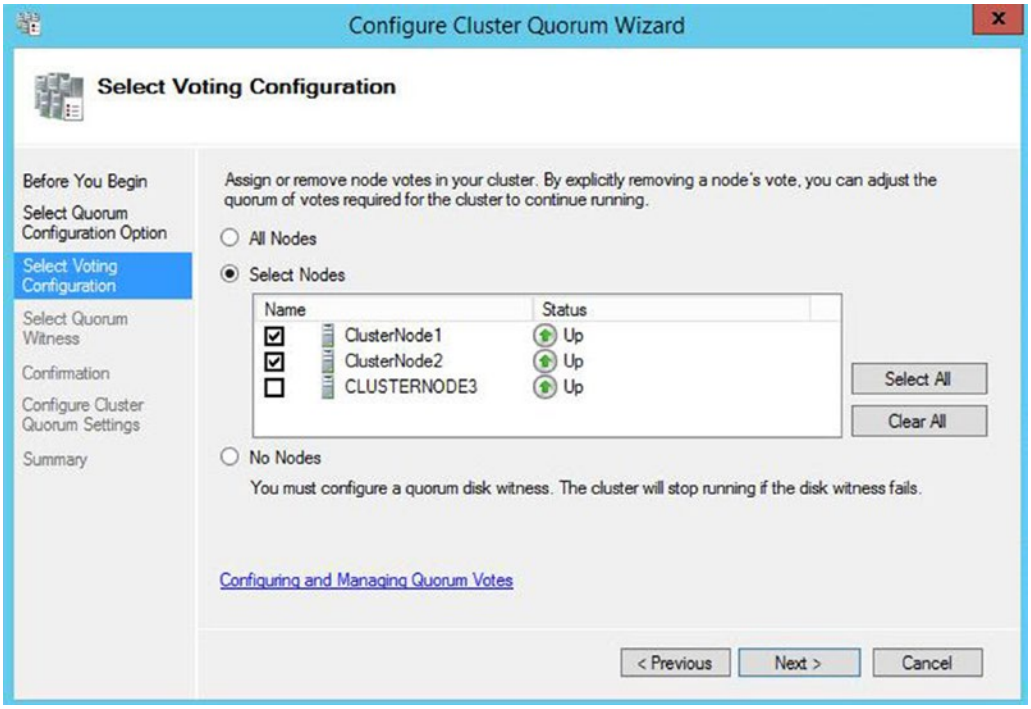

Figure 5-17. *The Select Voting Configuration page*

On the Select Quorum Witness page, we choose the Configure A File Share Witness option, as shown in Figure 5-18.

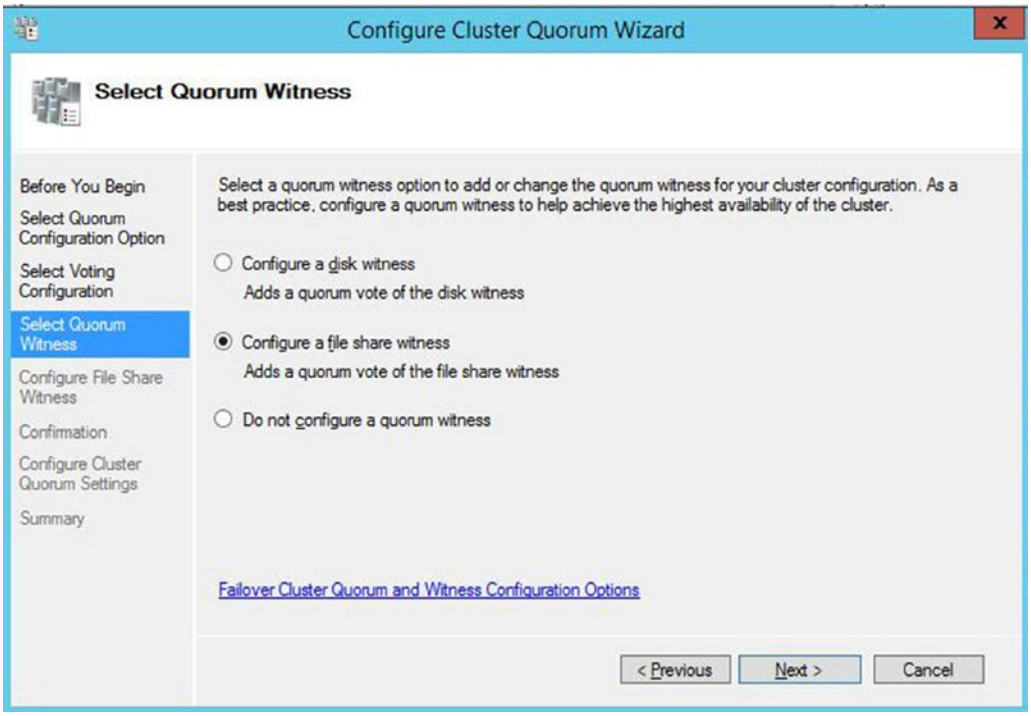

Figure 5-18. *The Select Quorum Witness page*

On the Configure File Share Witness page, which is illustrated in Figure 5-19, 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.

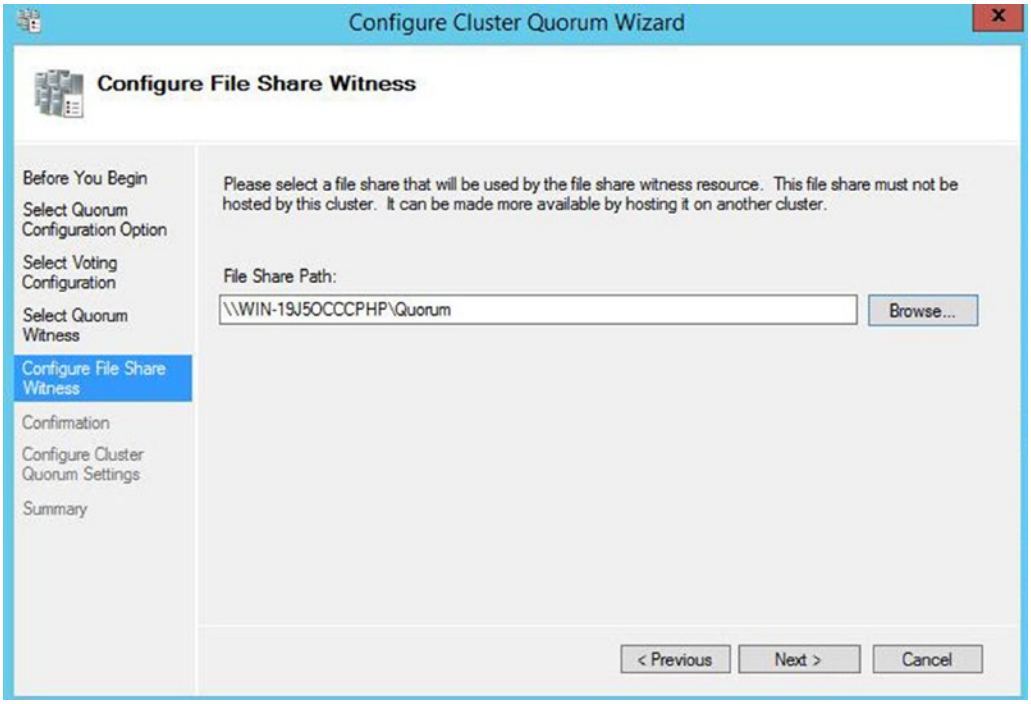

Figure 5-19. *The Configure File Share Witness page*

■ **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 5-20.

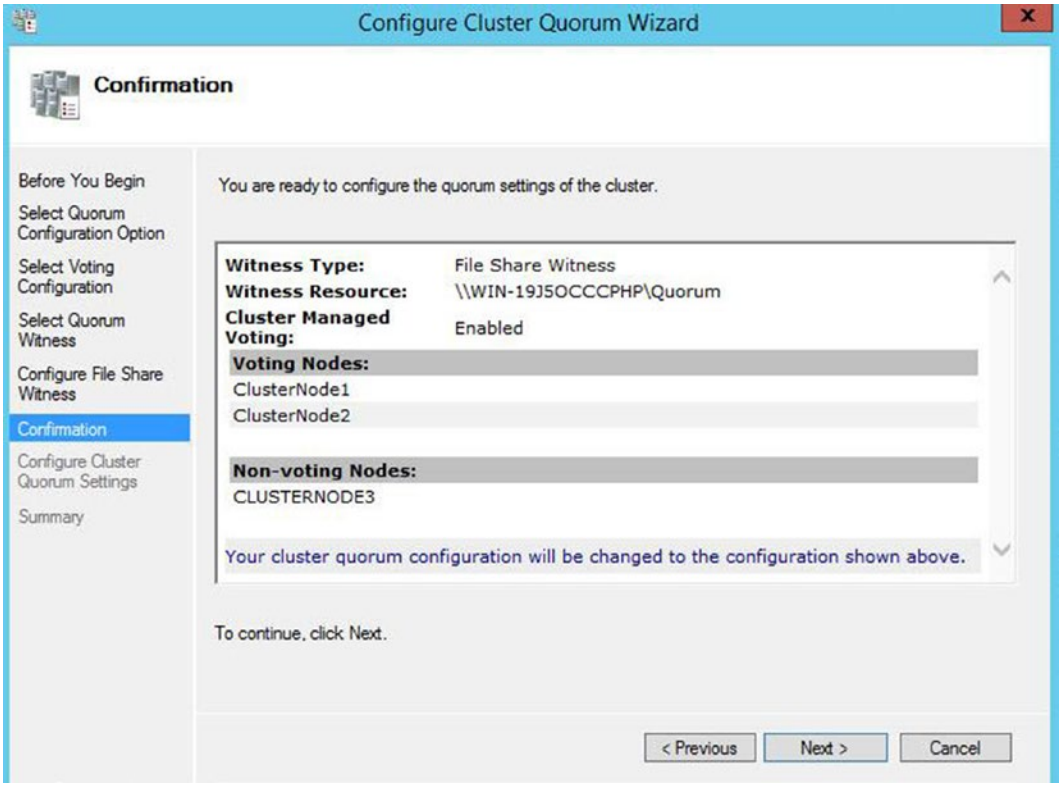

Figure 5-20. *The Confirmation page*

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 5-21.

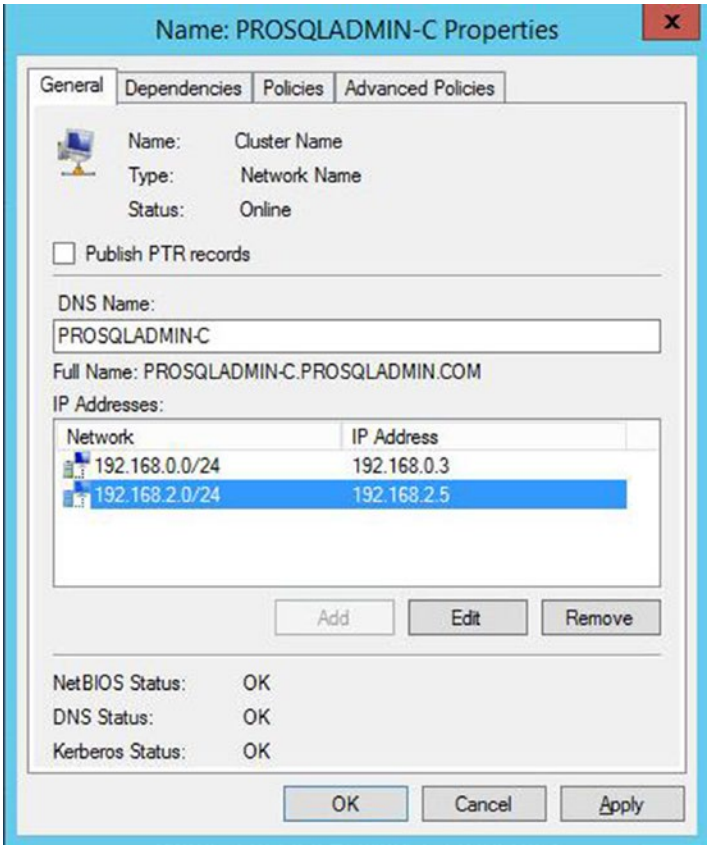

Figure 5-21. *The General tab*

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 5-22.

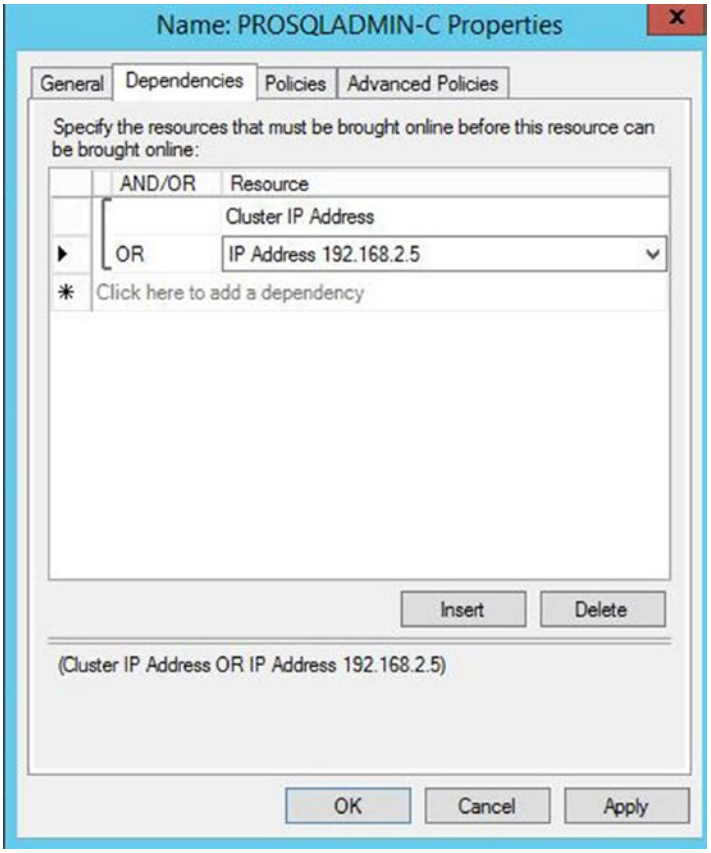

Figure 5-22. *The Dependencies tab*

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 5-23. On this page, you are invited to connect to the other replicas in the availability group.

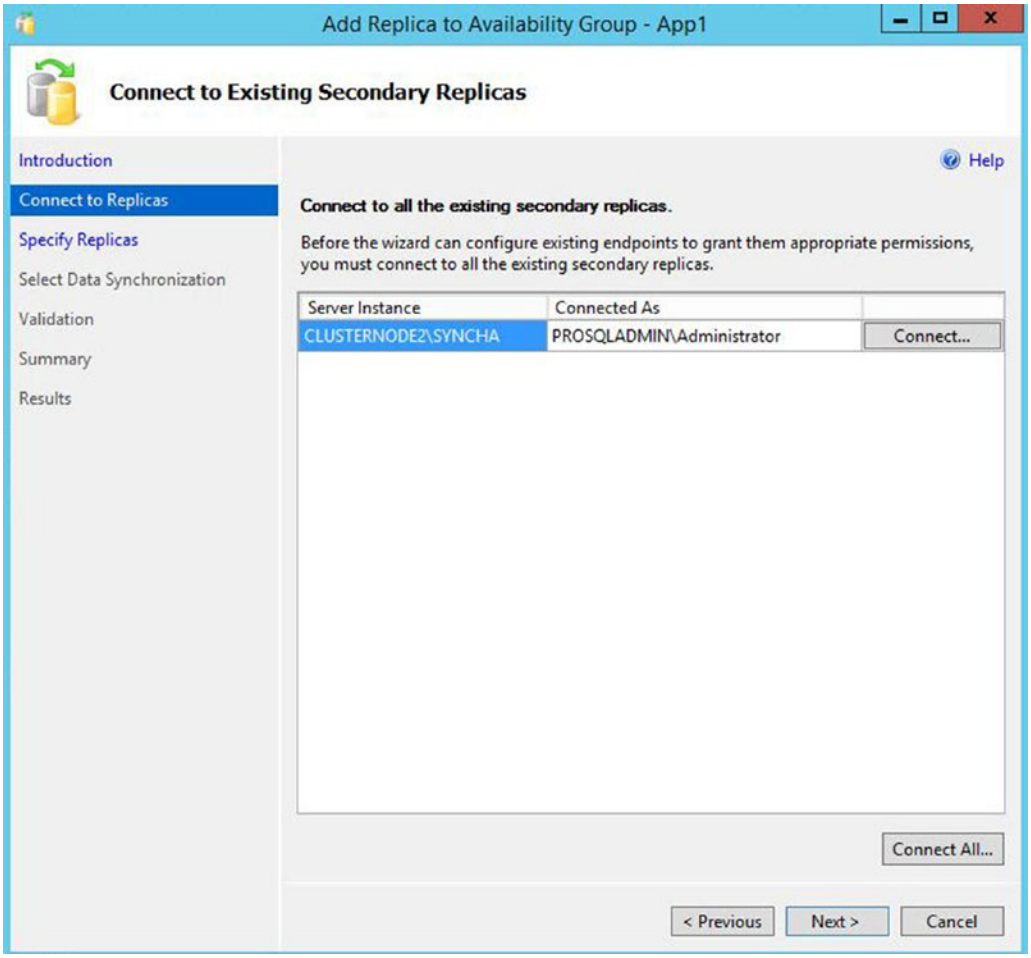

Figure 5-23. *The Connect To Replicas page*

On the Replicas tab of the Specify Replicas page, shown in Figure 5-24, 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.

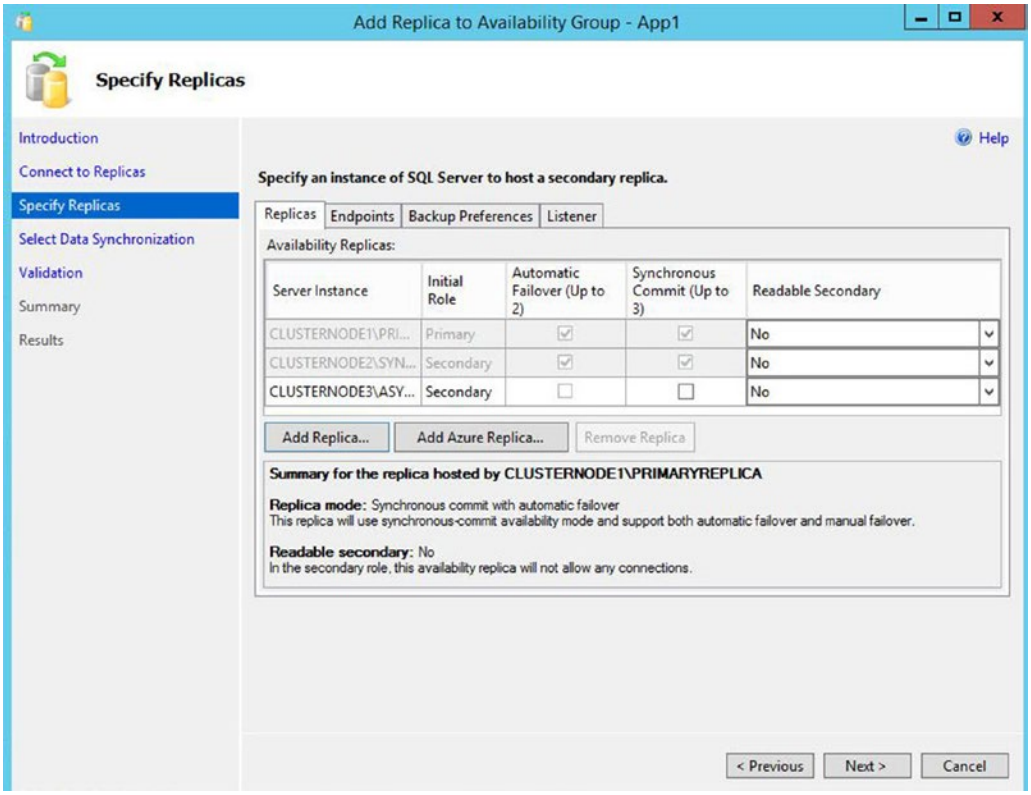

Figure 5-24. *The Replicas tab*

On the Endpoints tab, shown in Figure 5-25, we ensure that the default settings are correct and acceptable.

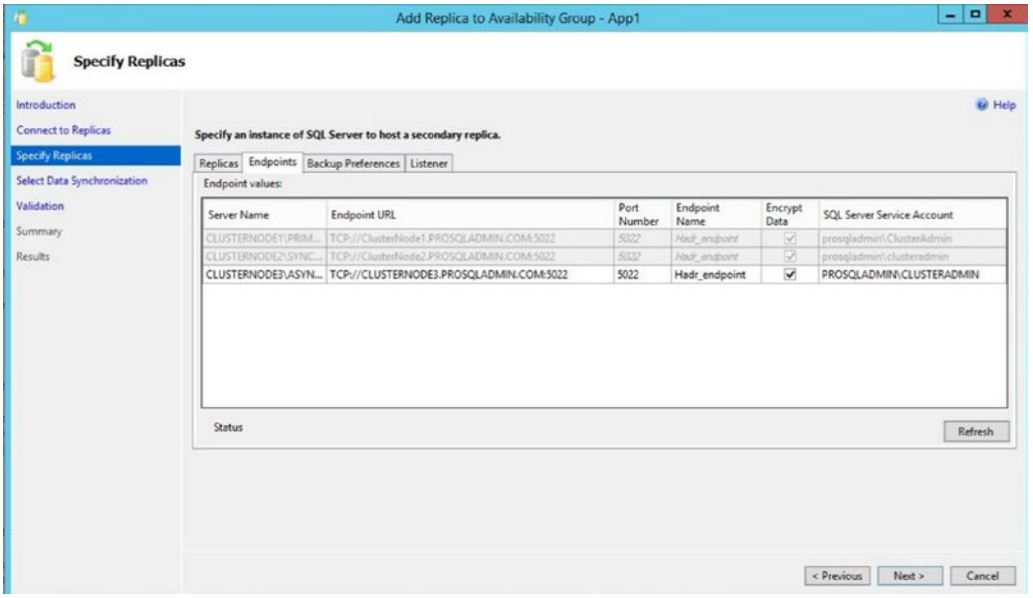

Figure 5-25. *The Endpoints tab*

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 5-26.

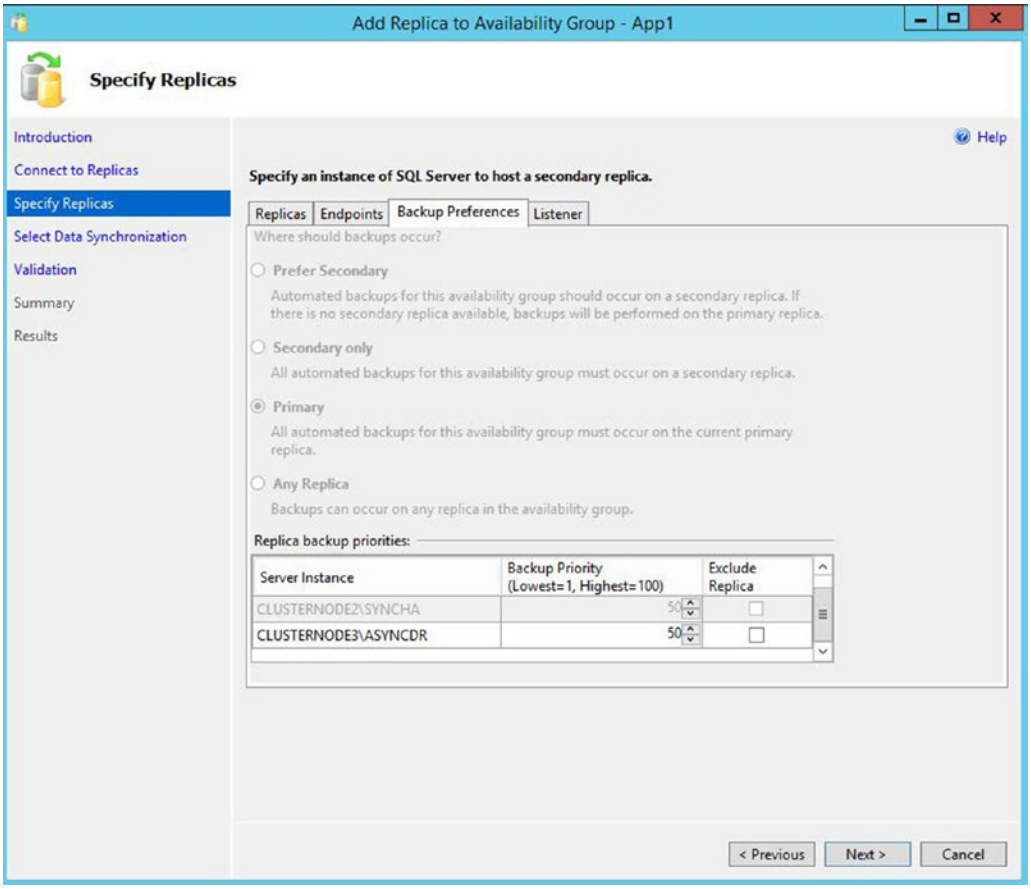

Figure 5-26. *The Backup Preferences tab*

On the Listener tab, illustrated in Figure 5-27, 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.

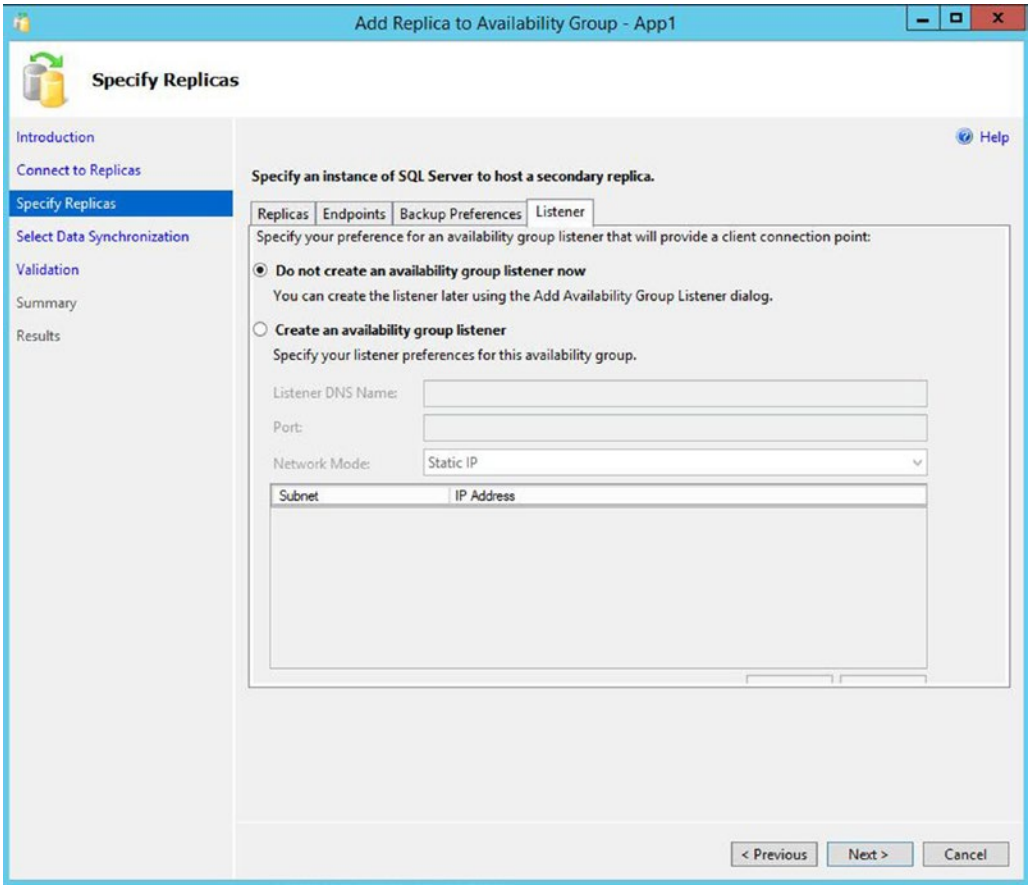

Figure 5-27. *The Listener tab*

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 5-28.

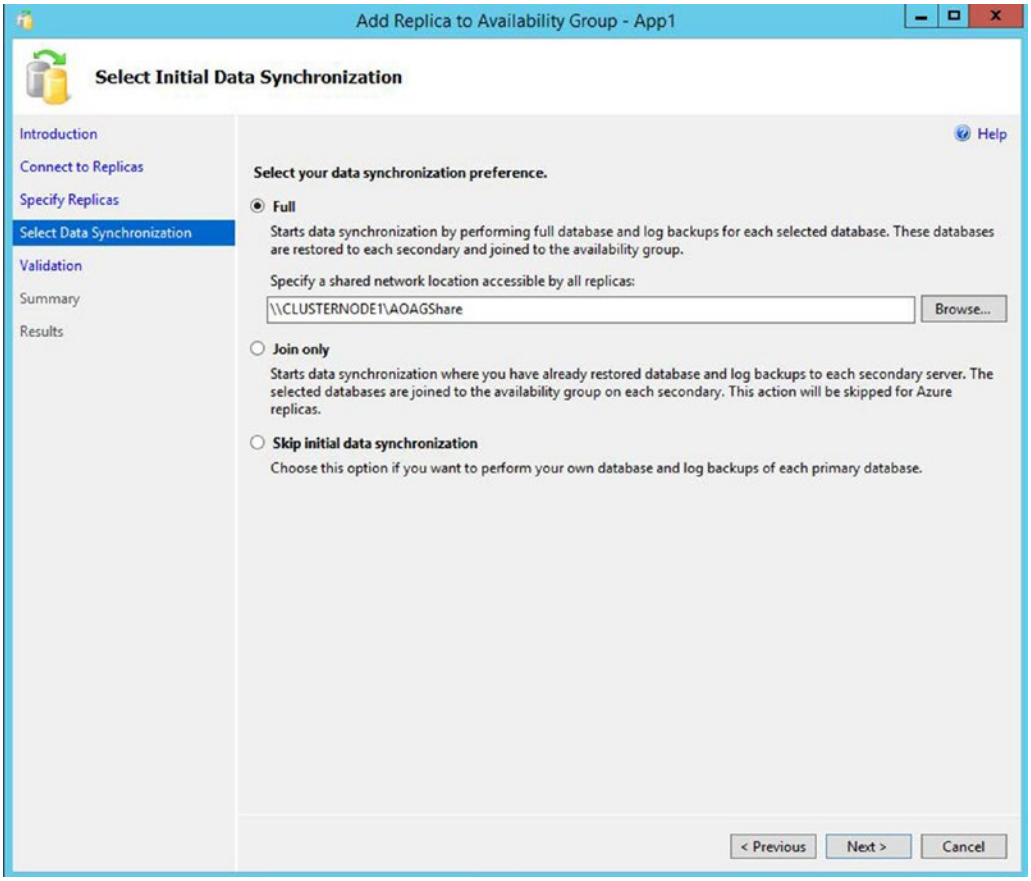

Figure 5-28. *The Select Data Synchronization page*

On the Validation page, which is illustrated in Figure 5-29, we should review any warnings or errors and resolve them before continuing.

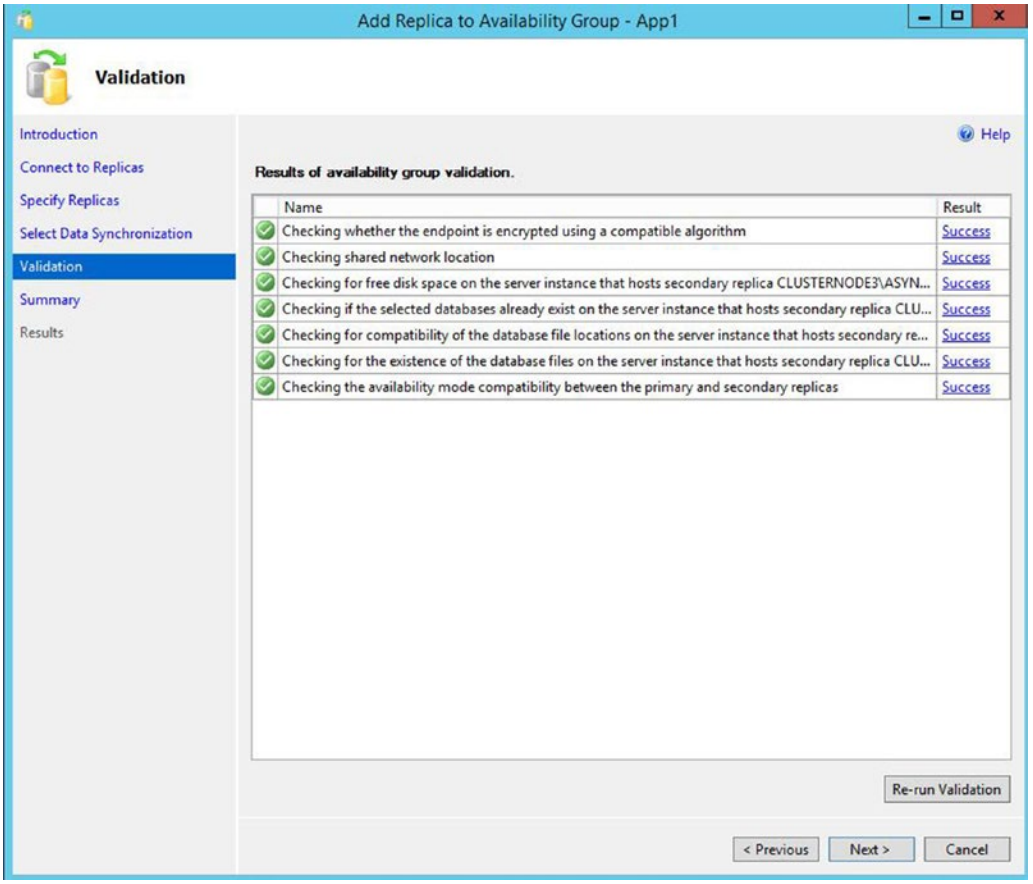

Figure 5-29. *The Validation page*

On the Summary page, displayed in Figure 5-30, we are presented with a summary of the configurations that will be carried out.

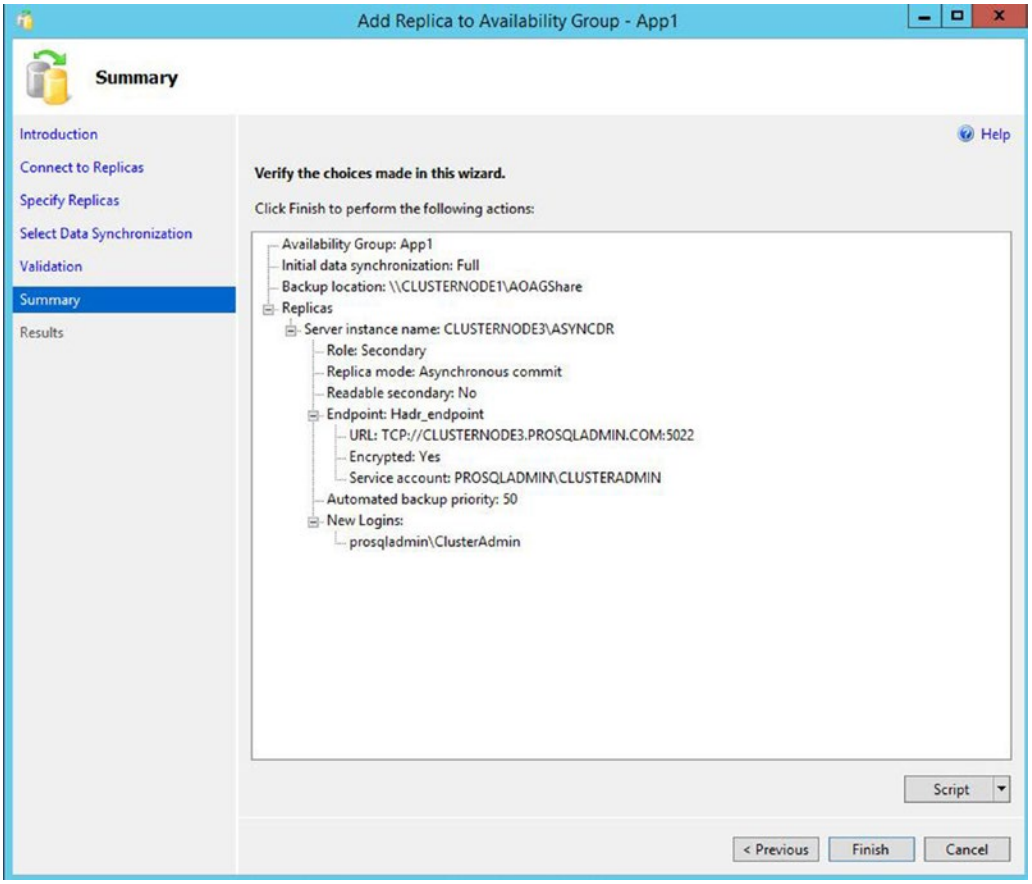

Figure 5-30. *The Summary page*

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 5-8 performs the same actions just demonstrated. You must run this script in SQLCMD Mode since it connects to multiple instances.

Listing 5-8. Adding a Replica

```
:Connect CLUSTERNODE3\ASYNCDR

--Create Login for Service Account

USE master
GO

CREATE LOGIN [prosqladmin\ClusterAdmin] 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 [prosqladmin\ClusterAdmin] ;
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 Avaialability Group

ALTER AVAILABILITY GROUP App1
ADD REPLICA ON N'CLUSTERNODE3\ASYNCDR'
WITH (ENDPOINT_URL = N'TCP://CLUSTERNODE3.PROSQLADMIN.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 Chapter5App1Customers TO  DISK = N'\\CLUSTERNODE1\AOAGShare\
Chapter5App1Customers.bak'
WITH  COPY_ONLY, FORMAT, INIT, REWIND, COMPRESSION,  STATS = 5 ;
GO
```

```
BACKUP LOG Chapter5App1Customers
TO  DISK = N'\\CLUSTERNODE1\AOAGShare\Chapter5App1Customers.trn'
WITH NOSKIP, REWIND, COMPRESSION,  STATS = 5 ;
GO

:Connect CLUSTERNODE3\ASYNCDR

ALTER AVAILABILITY GROUP App1 JOIN;
GO

RESTORE DATABASE Chapter5App1Customers
FROM  DISK = N'\\CLUSTERNODE1\AOAGShare\Chapter5App1Customers.bak'
WITH  NORECOVERY,  STATS = 5 ;
GO

RESTORE LOG Chapter5App1Customers
FROM  DISK = N'\\CLUSTERNODE1\AOAGShare\Chapter5App1Customers.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 Chapter5App1Customers SET HADR AVAILABILITY GROUP = [App1];
GO
```

```
--Back up and restore the second database and log

:Connect CLUSTERNODE1\PRIMARYREPLICA

BACKUP DATABASE Chapter5App1Sales
TO  DISK = N'\\CLUSTERNODE1\AOAGShare\Chapter5App1Sales.bak'
WITH  COPY_ONLY, FORMAT, INIT, REWIND, COMPRESSION,  STATS = 5 ;
GO

BACKUP LOG Chapter5App1Sales
TO  DISK = N'\\CLUSTERNODE1\AOAGShare\Chapter5App1Sales.trn'
WITH NOSKIP, REWIND, COMPRESSION,  STATS = 5 ;
GO

:Connect CLUSTERNODE3\ASYNCDR

ALTER AVAILABILITY GROUP [App1] JOIN;
GO

RESTORE DATABASE Chapter5App1Sales
FROM  DISK = N'\\CLUSTERNODE1\AOAGShare\Chapter5App1Sales.bak'
WITH  NORECOVERY,  STATS = 5 ;
GO

RESTORE LOG Chapter5App1Sales
FROM  DISK = N'\\CLUSTERNODE1\AOAGShare\Chapter5App1Sales.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 Avaiability Group on the new replica

ALTER DATABASE Chapter5App1Sales 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 5-31. Here, we add the listener's second IP address.

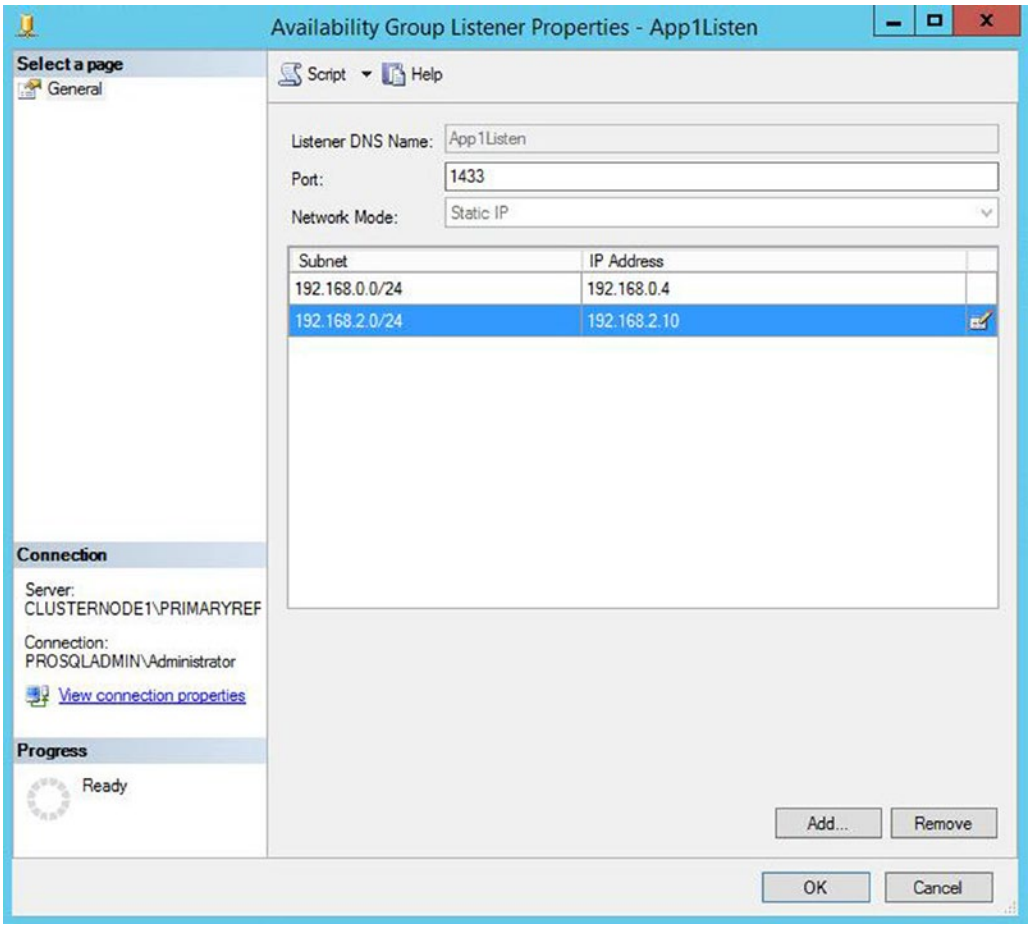

Figure 5-31. *The Availability Group Listener Properties*

We can also achieve this through T-SQL by running the script in Listing 5-9.

Listing 5-9. Adding an IP Address to the Listener

```
ALTER AVAILABILITY GROUP App1
MODIFY LISTENER 'App1Listen'
(ADD IP (N'192.168.2.10', 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 5-32.

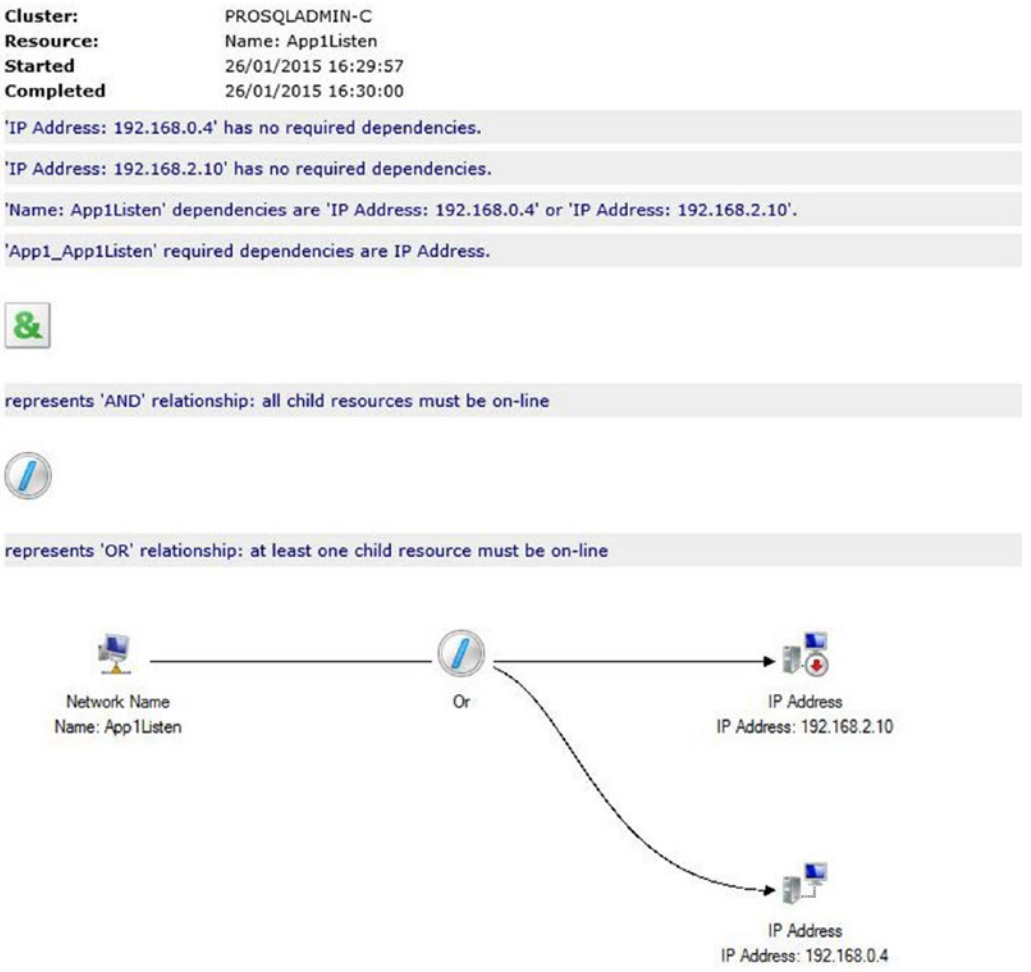

Figure 5-32. *The dependency report*

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 5-10 demonstrates how to disable `RegisterAllProvidersIP` and then reduce the `HostRecordTTL` to 300 seconds.

Listing 5-10. 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 time-out 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.

Adding AlwaysOn 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 here, 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 cleanup task only remove 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 re-add 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 5-11.

Listing 5-11. Configuring Read-only Routing

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

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

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

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

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.

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 6

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 monitoring with the AlwaysOn Dashboard and a demonstration of how to failover.

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 6-1.

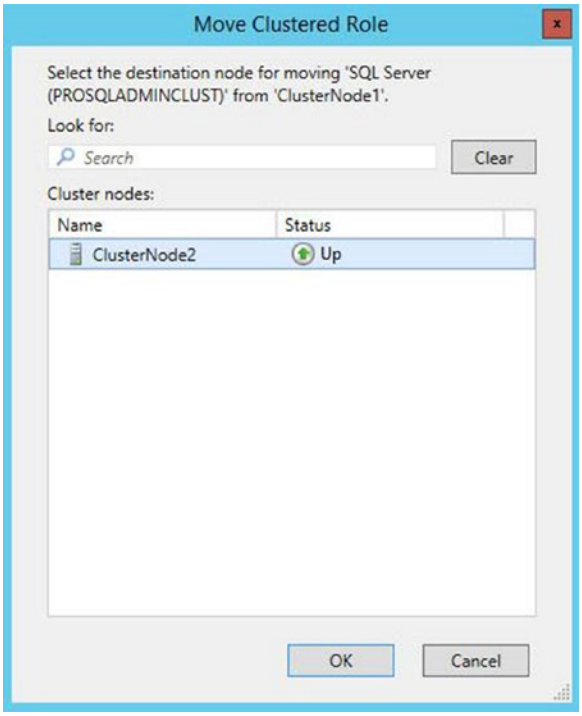

Figure 6-1. *The Move Clustered Role dialog box*

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, as illustrated in Figure 6-2. The resources are taken offline and placed back online in order of their dependencies.

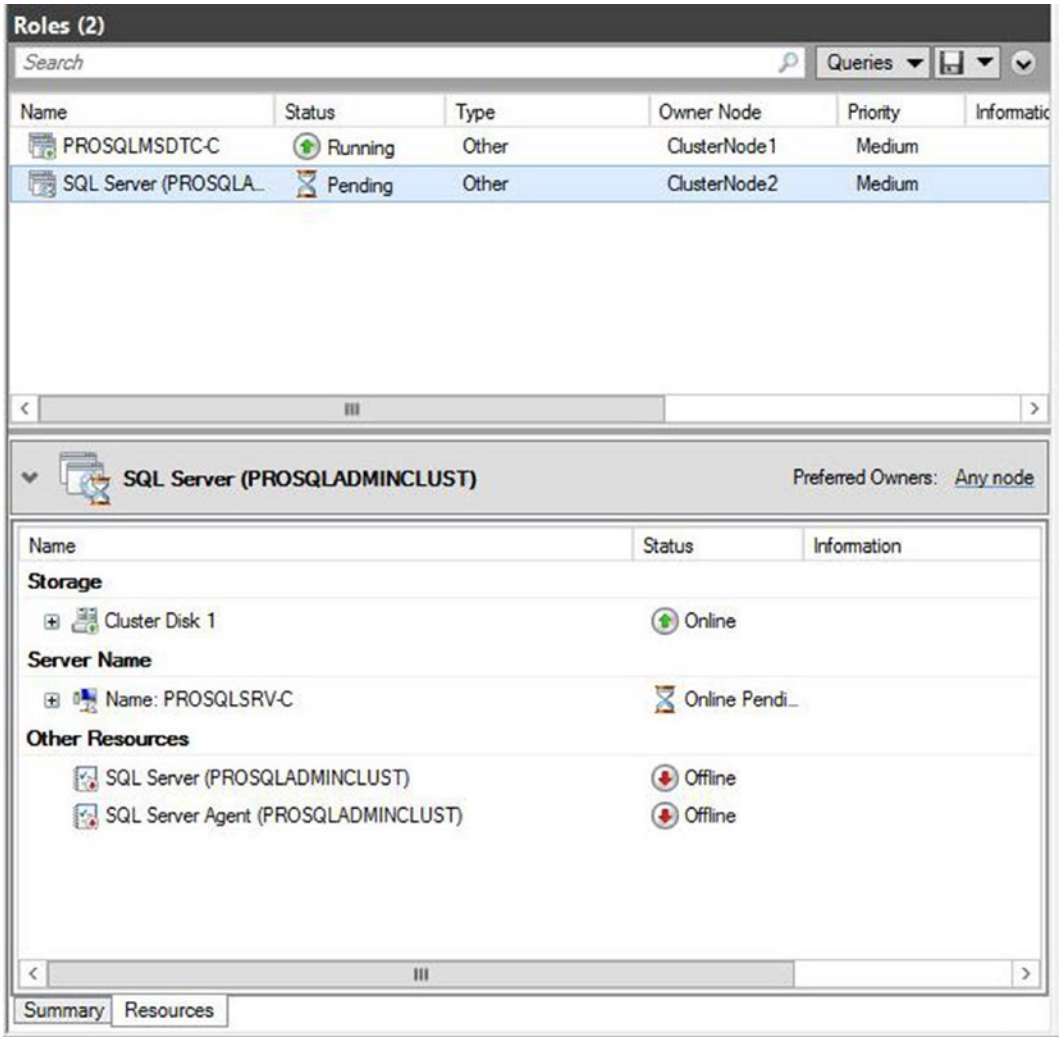

Figure 6-2. *Resources coming onlinie on passive node*

We can also fail over a role using PowerShell. To do this, we need to use the `Move--ClusterGroup` cmdlet. Listing 6-1 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 6-1. Moving the Role Between Nodes

```
Move-ClusterGroup -Name "SQL Server (PROSQLADMINCLUST)" -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 percent 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 6-3.

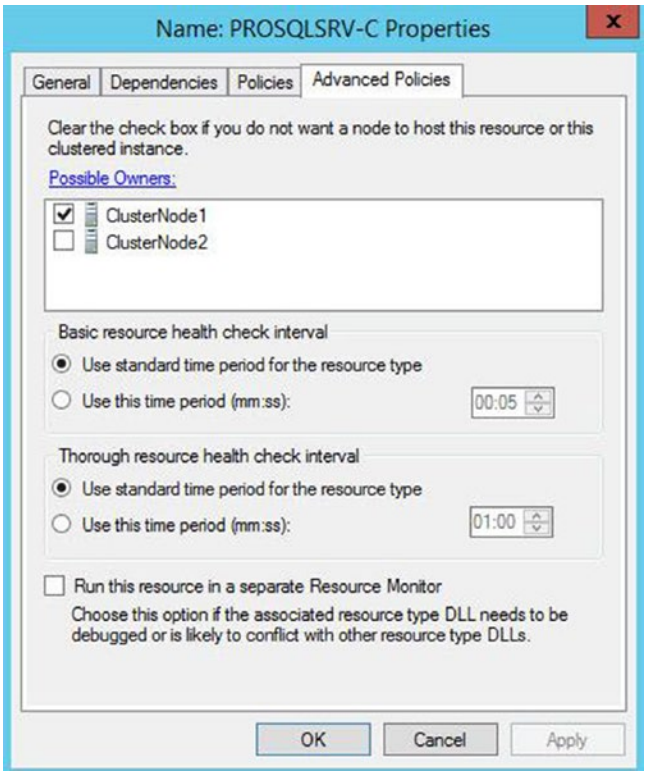

Figure 6-3. *Remove possible owners.*

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 6-2. The possible owners list is comma separated in the event that you are configuring multiple possible owners.

Listing 6-2. Configuring Possible Owners

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

Once 50 percent 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.

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 6-4, 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.

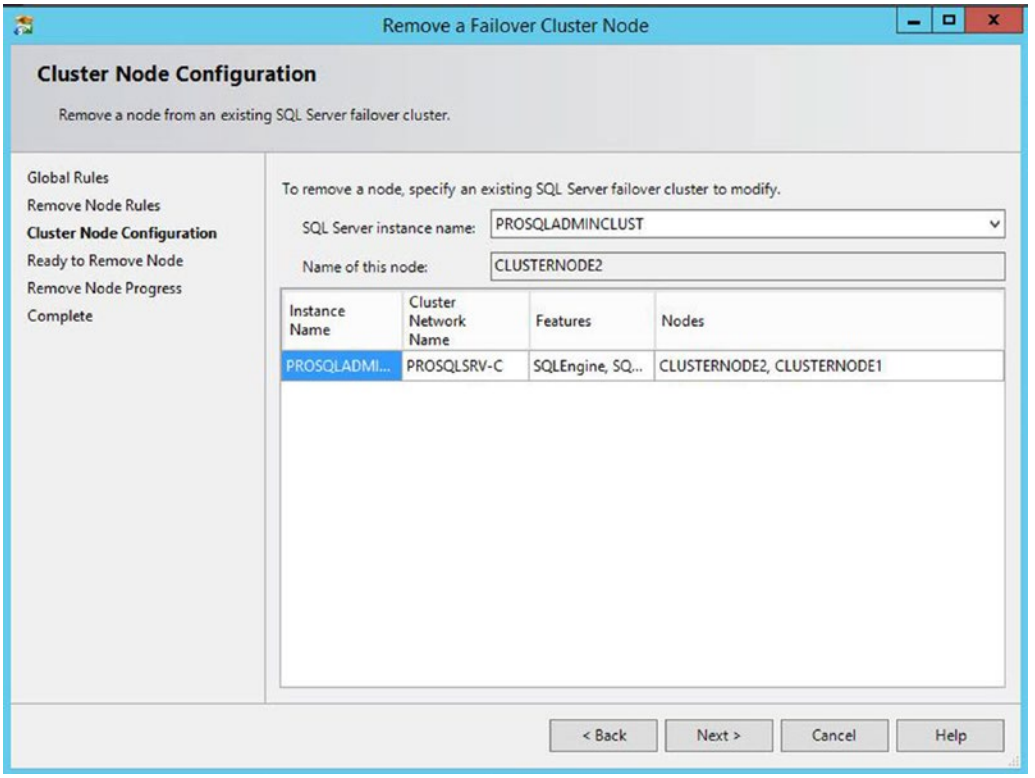

Figure 6-4. *The Cluster Node Configuration page*

On the Ready To Remove Node page, shown in Figure 6-5, 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.

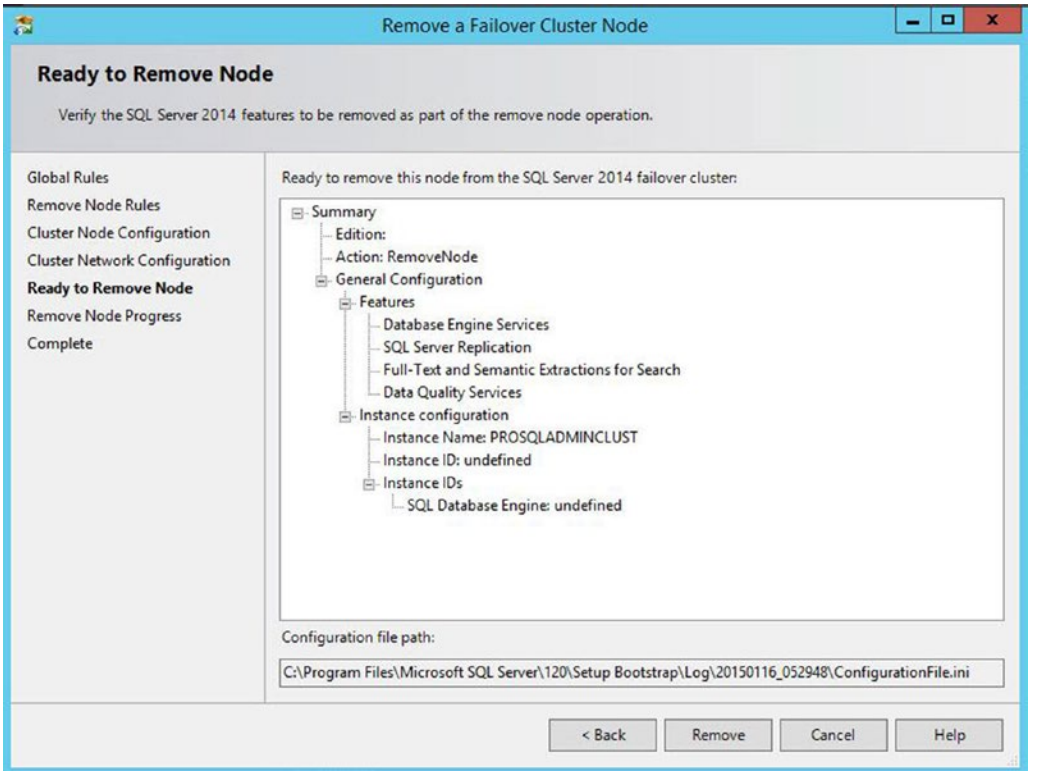

Figure 6-5. *The Ready To Remove Node page*

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 6-1 are mandatory.

Table 6-1. *Mandatory Parameters When Removing a Node from a Cluster*

Parameter	Usage
`/ACTION`	Must be configured as `AddNode`.
`/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`.

The script in Listing 6-3 removes a node from our cluster when we run it from the root directory of the SQL Server installation media.

Listing 6-3. Removing a Node

```
.\setup.exe /ACTION="RemoveNode" /INSTANCENAME="PROSQLADMINCLUST" /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 6-6). 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.

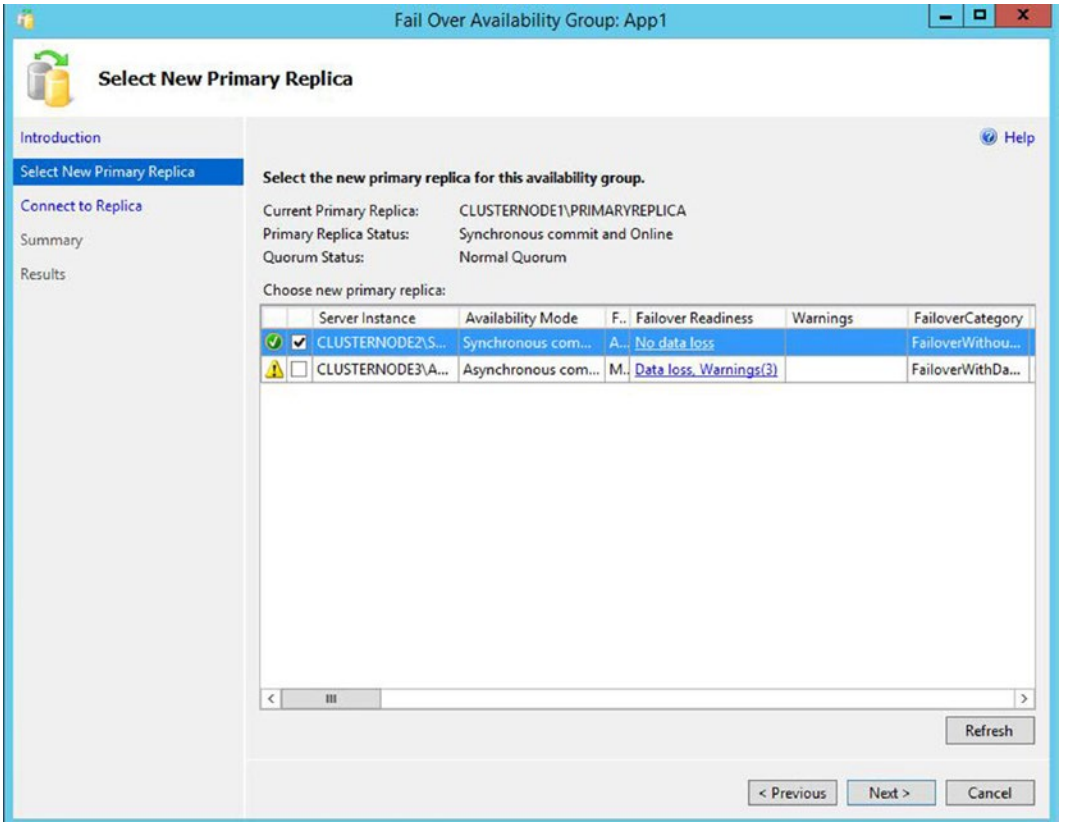

Figure 6-6. *The Select New Primary Replica page*

On the Connect To Replica page, illustrated in Figure 6-7, use the Connect button to establish a connection to the new primary replica.

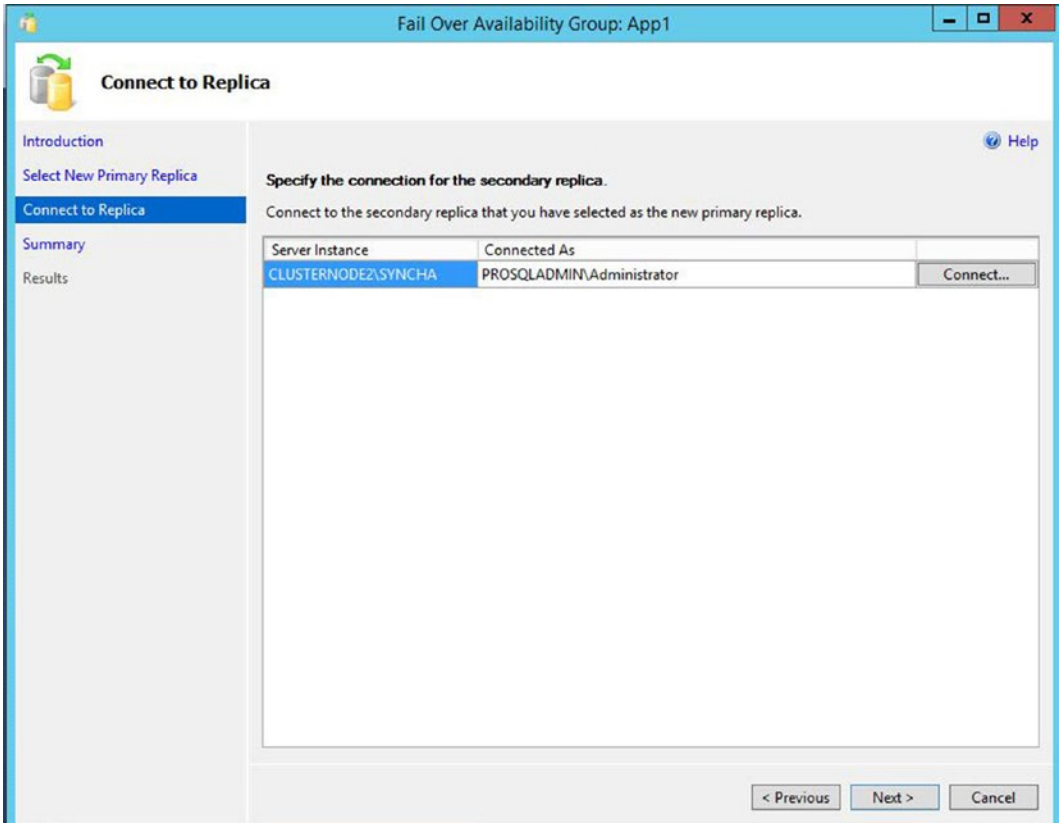

Figure 6-7. *The Connect To Replica page*

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 used T-SQL to fail over the availability group. The command in Listing 6-4 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 6-4. 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 6-5. 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 6-5. 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 6-6. 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 6-6. 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)
```

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

Monitoring

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. The following sections examine these two features.

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 6-8 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.

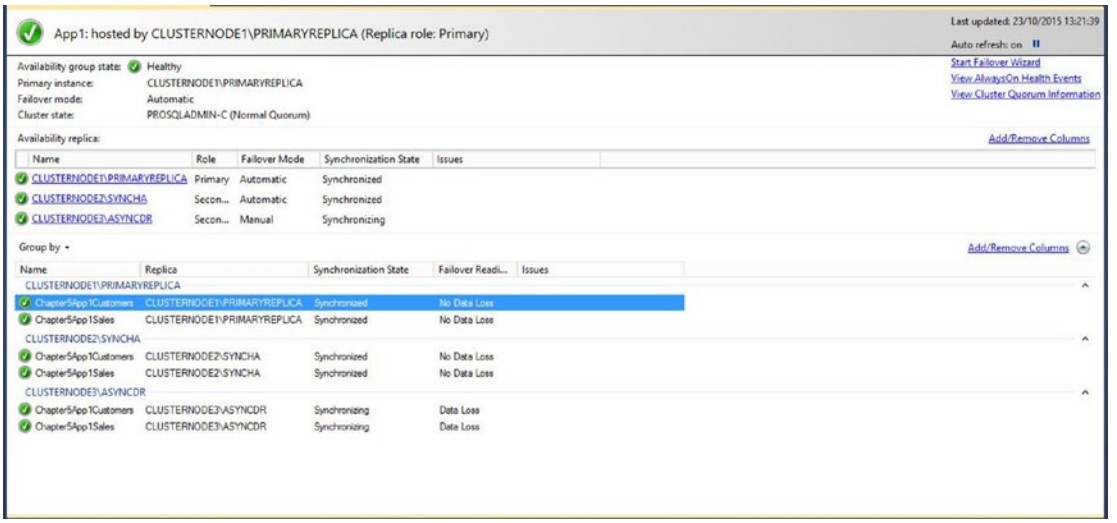

Figure 6-8. *The availability group dashboard*

The three possible synchronization states that a database can be in are `SYNCHRONIZED`, `SYNCRONIZING`, 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 6-9. You can also drill through each replica in the Availability Replicas window to see replica-specific details.

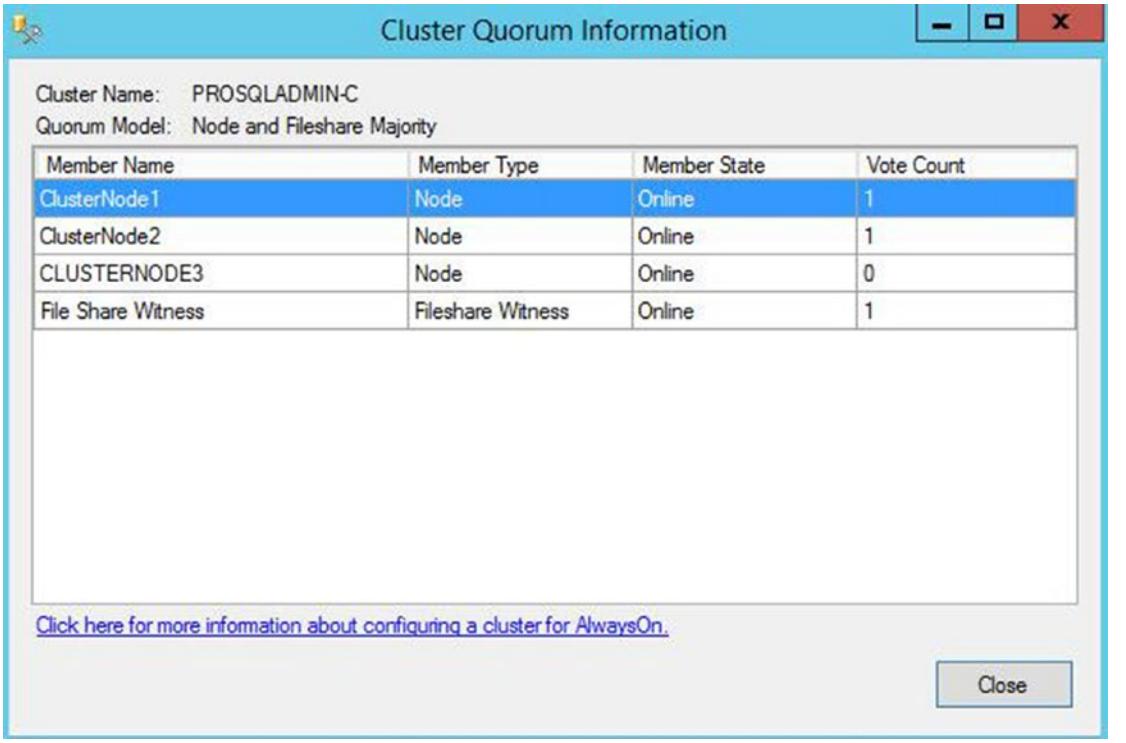

Figure 6-9. *The Cluster Quorum Information screen*

AlwaysOn Health Trace

The AlwaysOn Health Trace is an Extended Events session, which is created when you create you 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.

Drilling through the session exposes the session's package, and from the context menu of the package, you can view previously captured events. Figure 6-10 shows that the latest event captured, was Database 5 (which, in our case, is `Chapter5App1Customers`), was waiting for the log to be hardened on the synchronous replica.

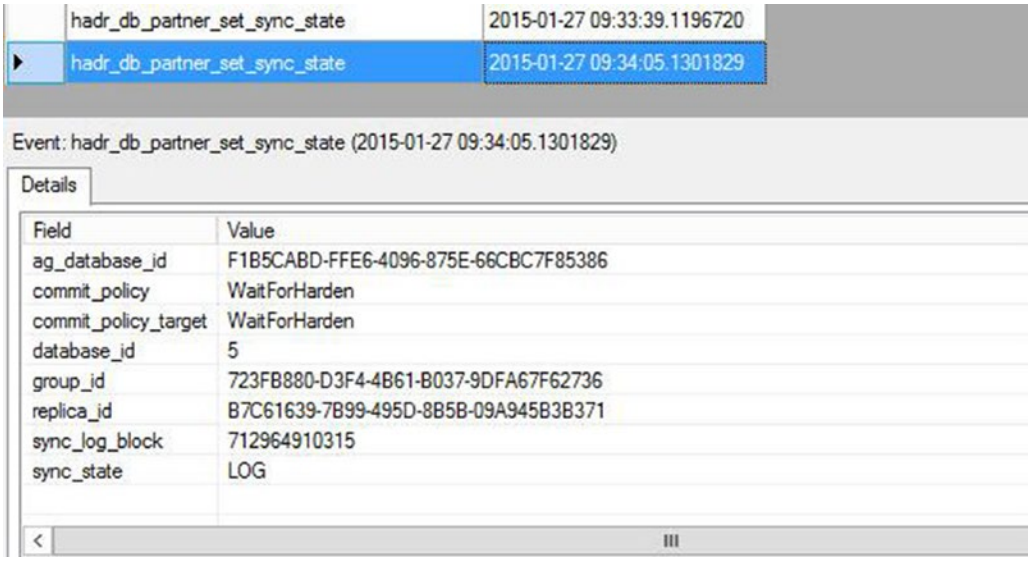

Figure 6-10. *The target data*

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 6-11. You can see that we have entered the DNS name for the client access point and specified an IP address from each subnet.

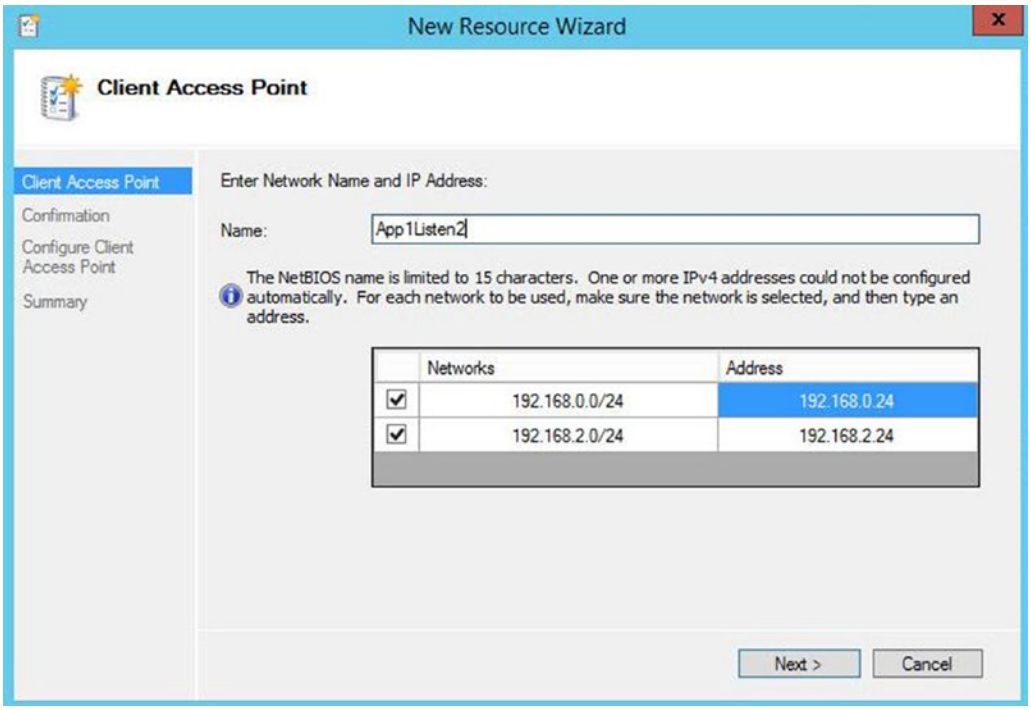

Figure 6-11. *The Client Access Point page*

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 6-12.

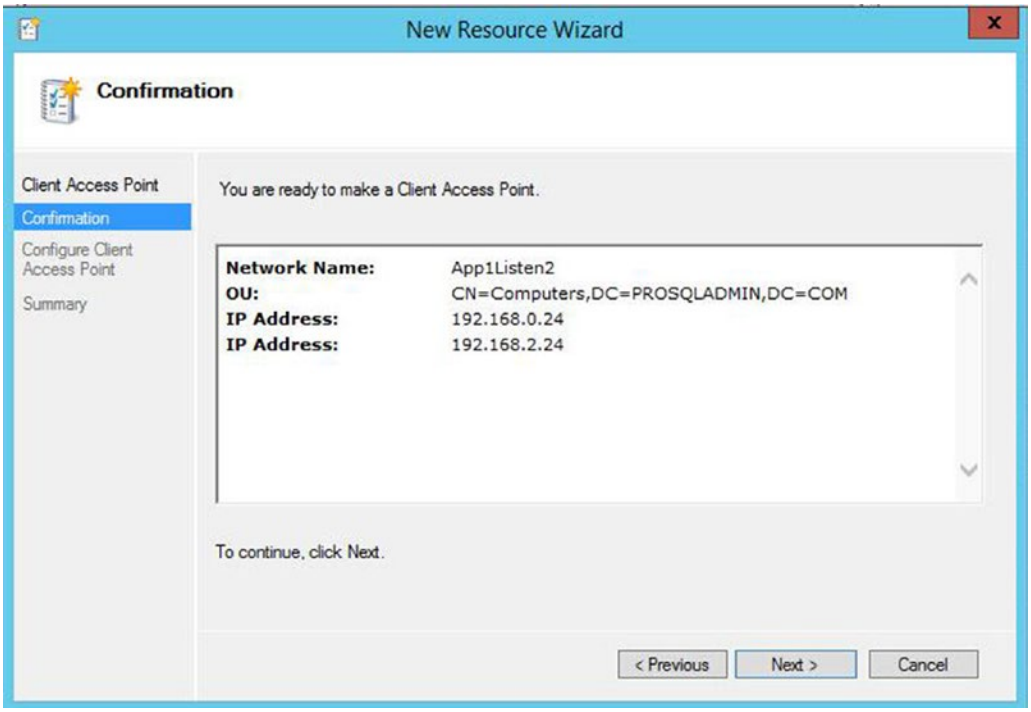

Figure 6-12. *The Confirmation page*

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 6-13. Once we apply this change, clients are able to connect using either of the two listener names.

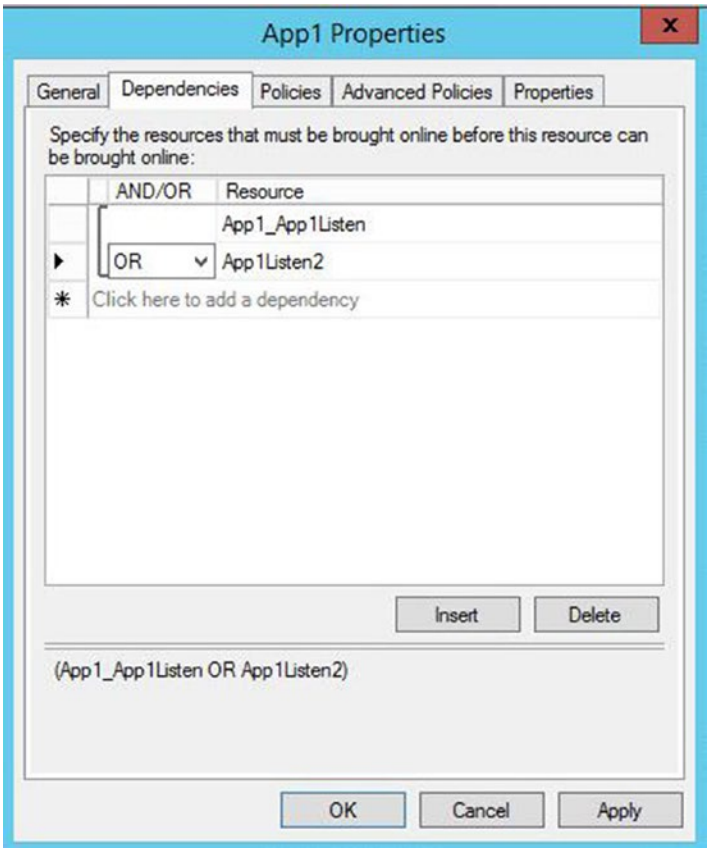

Figure 6-13. *The Dependencies tab*

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 6-7. This command removes the `Chapter5App1Customers` database from the availability group.

Listing 6-7. Removing a Database from an Availability Group

```
ALTER DATABASE Chapter5App1Customers 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 6-8, which suspends data movement for the `Chapter5App1Sales` 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 6-8. Suspending Data Movement

```
ALTER DATABASE Chapter5App1Customers SET HADR SUSPEND ;
GO

ALTER DATABASE Chapter5App1Customers 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 failover 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 failover 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 version, across the possible owners.

Index

A, B

C

D, E

F, G, H

I, J, K

L, M, N, O, P

Q

R

S

T, U

V, W, X, Y, Z