



**BRENT OZAR**  
UNLIMITED®

# How to Build a Microsoft SQL Server Always On Availability Group in Google Compute Engine



Authors: Tara Kizer, Brent Ozar  
© 2017 Brent Ozar Unlimited.

Version: v1.01, 2017-03-20. To get the latest version free, visit: <https://BrentOzar.com/go/gce>

# Table of Contents

## [Table of Contents](#)

### [Introduction: About You, Us, and This Build](#)

[About You](#)

[About Us](#)

[About What We'll Build Together](#)

[About the Licensing](#)

### [How to Build a 2-Node Availability Group in GCE](#)

[Networks: or, Why This Document is Long](#)

[Creating the VMs Using the UI](#)

[Creating the VMs Using the Command Line](#)

[Connecting to the VMs via Remote Desktop](#)

[Configuring the VMs' Security and Networks](#)

[Installing and Configuring the Windows Clustering Services](#)

[Setting up the Availability Group](#)

### [Testing Your Availability Group](#)

[Test #1: Failing Over Manually - A planned failover with zero data loss](#)

[Test #2: Failing Over Automatically - An unplanned failover with zero data loss](#)

[Test #3: Failing Over Manually with Data Loss - An unplanned failover with data loss](#)

[Test #4: Failing Back Manually - A planned failback with zero data loss](#)

[Testing Recap](#)

### [Replacing a Server with a Bigger/Lesser Server](#)

### [Dealing With Availability Group Design Gotchas](#)

[The Big Showstopper: Cross-Database Transactions](#)

[Database Objects That Aren't Protected by Availability Groups](#)

[Synchronizing Logins Across Replicas](#)

[Synchronizing Agent Jobs Across Replicas](#)

[Services That Aren't Protected by Availability Groups](#)

### [Recap, Next Steps, and Learning Resources](#)

[Change Log](#)

# Introduction: About You, Us, and This Build

## About You

You're a database administrator, Windows admin, or developer. You want to build a Microsoft SQL Server environment that's highly available, and you've chosen to use Always On Availability Groups.

In this white paper, we'll show you:

- How to build your first Availability Group in Google Compute Engine
- How to test your work with four failure simulations
- How to tell whether your databases will work well in GCE

We're going to skip a lot of common SQL Server setup tasks that aren't really any different for the cloud. For those, check out our SQL Server Setup Checklist in [our free First Responder Kit](#).

We'll be using PowerShell to accomplish some of the setup and configuration tasks, but you don't need to be familiar with PowerShell in order to follow along.

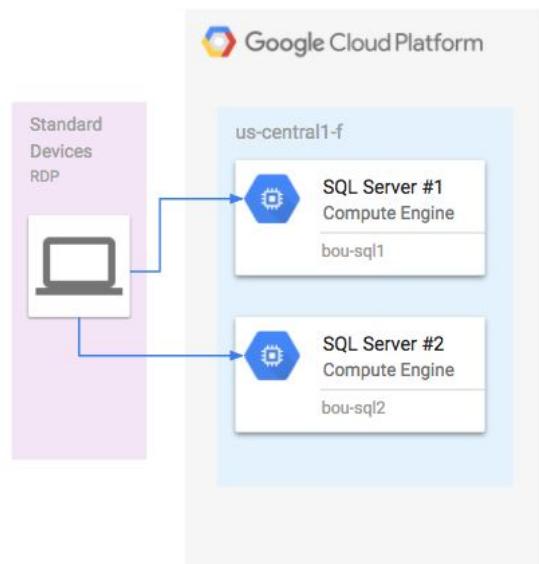
## About Us

We're your tour guides: Tara Kizer and Brent Ozar. We're from Brent Ozar Unlimited®, a small boutique consulting firm that makes SQL Server faster and more reliable. You can find more of our work at <https://www.BrentOzar.com>.

## About What We'll Build Together

We're going to build a 2-node Always On Availability Group using two SQL Server virtual machines in the same zone as shown at right.

We're going to keep this one simple: we're not going to deal with network security, VPNs, application servers, or anything like that, and we're not going to expose SQL Server to the Internet. (That's a whole 'nother security discussion.)



Always On Availability Groups (AGs) make a lot of sense in the cloud for a few reasons:

**They offer automatic failover of multiple databases.** This is a common high availability requirement, and on premises, DBAs usually solve this with failover clustered instances (FCIs). Unfortunately, that requires shared storage, which isn't usually available in today's infrastructure-as-a-service (IaaS) cloud. There are some workarounds involving third party software or UNC paths, but neither of those are great options in IaaS - especially when we're taught to [treat our servers like cattle, not like pets](#). Always On Availability Groups can pull this off without shared storage.

**Each replica can automatically repair corrupt data pages.** Each replica has its own copy of the database's data pages - only logged transactions are sent from one replica to another, not data pages. When a replica encounters a corrupt data page, it can request a clean copy of the page from another replica, and heal itself live. It doesn't protect from every kind of corruption, but it's a real lifesaver in many situations. [Learn more about automatic page repair in SQL Server Books Online.](#)

**Lower downtime for patching.** Patch the secondary replica, make sure it's working successfully, then take a brief outage to fail over from the primary replica to the newly patched secondary. Presto, minimal downtime. After that, you can patch the former primary. If something goes wrong when patching either replica, simply build a new one to replace it and join it into the Availability Group. (Remember, treat servers like cattle, not like pets.)

**Easier scalability with lower downtime** - when we want to switch to a bigger instance type, we can provision a new one, add it into the Availability Group, fail over to it, and then remove the previous instance. You can scale up and down to handle load with this approach, although we don't commonly see this used on a scripted basis to handle daily peaks and valleys. It's more of a seasonal approach.

The 2-node Availability Group design does come with a few drawbacks:

- Higher cost - having a warm standby means we're basically doubling our GCE costs
- More complexity - this is a little harder to manage than a conventional single SQL Server VM and requires Windows clustering knowledge
- No disaster recovery - if we want the ability to fail over to another zone or region, we'll need to add additional VMs, but we can do that later

Assumptions:

- We're using SQL Server 2016 for this project. (Setup is different for prior versions.)
- We're using Windows Server 2016. (Future versions of SQL Server will support running Linux on the OS, but the AG setup will be quite different. If you're worried about the licensing costs of Windows as a base OS, that pales next to SQL Server's licensing.)
- We've already got domain controllers and DNS set up.

- Our application servers are in Google Compute Engine. (App servers in other places would require more complex security and network configuration that's out of scope for this white paper.)

## About the Licensing

In Google Compute Engine, there are two ways to license your SQL Server.

**With pay-per-use licensing**, your GCE VM hourly cost includes licensing. Google manages the licensing logistics with Microsoft. Your hourly costs are higher, but you have total flexibility to ramp your costs up and down whenever you want.

**With bring-your-own-licensing (BYOL)**, your GCE VM costs are lower because the licensing isn't included. You'll need to purchase your own SQL Server licensing from Microsoft, which means paying up front, and you don't have much flexibility here. However, for companies with very stable usage needs, or with free/discounted licensing through Microsoft licensing agreements, this can end up cheaper. If you're installing Developer Edition, use this approach.

Generally:

- If your SQL Server needs are **unpredictable**, use pay-per-use
- If your SQL Server needs are **predictable for at least a year**, buy them and bring them

We'll give you instructions for both in methods in this white paper. If you're not sure which one to use, try pay-per-use because it's easier to get started. (Just make sure to shut down your instances when you don't need 'em!)

# How to Build a 2-Node Availability Group in GCE

## Networks: or, Why This Document is Long

We're going to build two virtual machines (VMs), each running Microsoft SQL Server, but it's a little bit more complicated than you might be used to doing on-premises.

Here's the computer names we're going to be using:

- **bou-sql1** is a Windows VM running SQL Server
- **bou-sql2** is a Windows VM running SQL Server

Both of those VMs will be in a Windows cluster. If you've used Windows failover clustering in the past, you may have used a cluster that required a shared quorum drive (like maybe the "Q" drive) that was used for voting, determining which node was the primary at any given time.

Always On Availability Groups is built atop Windows clustering. Even if you don't have a shared drive (and we won't), Windows still needs to determine which node is the primary, and which nodes are the readable secondaries. Clustering is outside of the scope of this white paper, but there's one thing you need to know: clustering requires some networking and virtual server name infrastructure.

Later in this white paper, we're going to have a couple other names and IP addresses:

- **bou-cluster1** is a virtual name that points to the node that owns the cluster infrastructure (the Windows plumbing that determines who's the primary)
- **TestListener** is a virtual name that points to the node that owns the databases, the writable primary

These two names will fail over back and forth between nodes.

If you've built labs at home before, you might have put your two SQL Server VMs right next to each other in the same subnet. For example, maybe you used 192.168.1.10 and 192.168.1.11. Due to some complex networking issues, that doesn't fly here.

However, for this failover process to work in Google Compute Engine, our two SQL Server virtual machines have to be in *different network subnets*, like this:

- bou-sql1 - IP address 10.0.1.4, subnet 10.0.0.0/16
- bou-sql2 - IP address 10.1.1.4, subnet 10.1.0.0/16

Catch that difference? It's going to be pretty important. We're going to have to:

- Create the VMs in different subnets
- Setup network rules between the subnets so that Google will route traffic between them

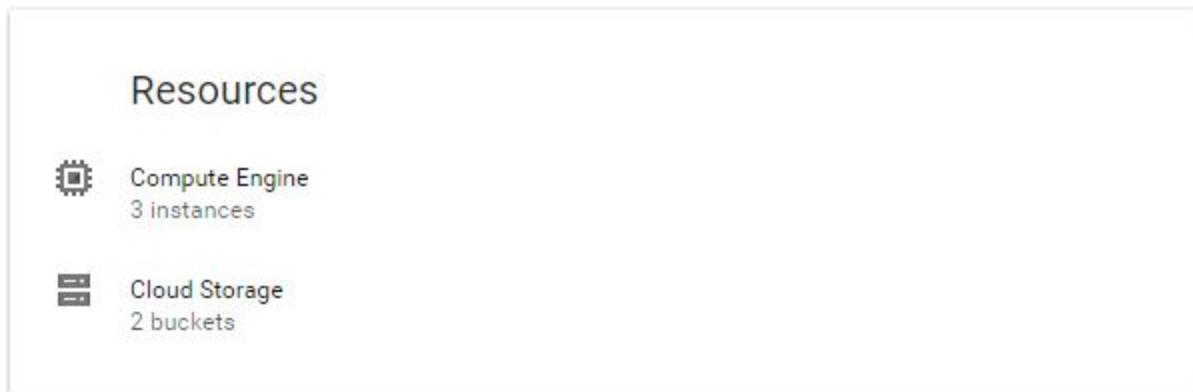
- Test to make sure that the communication works

In this white paper, we're going to give you an exact design complete with IP addresses. It's important that you not deviate from these - if you even fat-finger a single number here, your setup will fail.

When you go to do this in the real world, you'll want to work closely with your network team to pick the right IP addresses, subnets, and network security rules so that your app servers and end users can connect to SQL Server. For now, though, let's plow ahead.

## Creating the VMs Using the UI

From the GCP Dashboard, select *Compute Engine* in the *Resources* box.



Click *CREATE INSTANCE*.

[CREATE INSTANCE](#)

Give it a name and then specify which zone it should be in.

**Don't Overthink It:** for your first test AG, don't worry too much about the zone. When it's time to design your production infrastructure, you'll want to put your database servers in the same zone as your application servers in order to keep latency low. To learn more about what's available in each zone, head over to [Google's list of regions and zones](#) which lists what machine types are available in each location.

For *Machine type*, we don't recommend going any smaller than 4 vCPUs with 15 GB of memory (n1-standard-4), and it's likely that you'll need larger for production. Later on in this white paper, we'll talk about how to choose your production instance size based on your current workloads - but you can just start here for your staging or development environment.

Name 

Zone   

Machine type

 15 GB memory

Change *Boot disk* to a custom image by clicking *Change* and then either *OS images* or *Custom images*, depending on your licensing.

We will be using SQL Server 2016 on Windows 2016 with a 200GB SSD. (Smaller boot drives run the risk of filling up due to Windows Updates, or that idiotic sysadmin you work with who keeps saving ISO files to his desktop, not realizing that it goes to a folder on the C drive.)

Boot disk 

 New 10 GB standard persistent disk  
Image  
Debian GNU/Linux 8 (jessie)

**Decision Point:** picking your licensing model is important because you can't change this quickly later on. If you pick the wrong licensing model, you'll need to build new SQL Server VMs using the correct OS instance, join them into the Availability Group, fail over to the new VMs, and then decommission the old VMs.

Pay-per-use:

## Boot disk

Select an image or snapshot to create a boot disk; or attach an existing disk.

OS images    **Application images**    Custom images    Snapshots    Existing disks

- SQL Server 2012 Enterprise on Windows Server 2012 R2  
x64 built on 2017-01-17
- SQL Server 2012 Standard on Windows Server 2012 R2  
x64 built on 2017-01-17
- SQL Server 2012 Web on Windows Server 2012 R2  
x64 built on 2017-01-17
- SQL Server 2014 Enterprise on Windows Server 2012 R2  
x64 built on 2017-01-17
- SQL Server 2014 Standard on Windows Server 2012 R2  
x64 built on 2017-01-17
- SQL Server 2014 Web on Windows Server 2012 R2  
x64 built on 2017-01-17
- SQL Server 2016 Enterprise on Windows Server 2012 R2  
x64 built on 2017-01-17
- SQL Server 2016 Enterprise on Windows Server 2016  
x64 built on 2017-01-17
- SQL Server 2016 Express on Windows Server 2012 R2  
x64 built on 2017-01-17
- SQL Server 2016 Express on Windows Server 2016  
x64 built on 2017-01-17
- SQL Server 2016 Standard on Windows Server 2012 R2  
x64 built on 2017-01-17
- SQL Server 2016 Standard on Windows Server 2016  
x64 built on 2017-01-17
- SQL Server 2016 Web on Windows Server 2012 R2  
x64 built on 2017-01-17
- SQL Server 2016 Web on Windows Server 2016  
x64 built on 2017-01-17

Boot disk type 	Size (GB) 
SSD persistent disk	200

Bring your own license:

- Debian GNU/Linux 8 (jessie)  
amd64 built on 2016-12-15
- CentOS 6  
x86\_64 built on 2016-12-12
- CentOS 7  
x86\_64 built on 2016-12-12
- CoreOS alpha 1262.0.0  
amd64-usr published on 2016-12-15
- CoreOS beta 1235.2.0  
amd64-usr published on 2016-12-07
- CoreOS stable 1185.5.0  
amd64-usr published on 2016-12-07
- Ubuntu 12.04 LTS  
amd64 precise image built on 2016-12-05
- Ubuntu 14.04 LTS  
amd64 trusty image built on 2016-12-13
- Ubuntu 16.04 LTS  
amd64 xenial image built on 2016-12-21
- Ubuntu 16.10  
amd64 yakkety image built on 2016-12-05
- Red Hat Enterprise Linux 6  
x86\_64 built on 2016-12-12
- Red Hat Enterprise Linux 7  
x86\_64 built on 2016-12-12
- SUSE Linux Enterprise Server 11 SP4  
x86\_64 built on 2016-12-21
- SUSE Linux Enterprise Server 12 SP2  
x86\_64 built on 2016-12-14
- Windows Server 2008 R2  
Server with Desktop Experience, x64 built on 2016-12-13
- Windows Server 2012 R2  
Server with Desktop Experience, x64 built on 2016-12-13
- Windows Server 2016  
Server with Desktop Experience, x64 built on 2016-12-13

In this same screen, you'll configure the type and size of your SSD:

Boot disk type 	Size (GB) 
SSD persistent disk 	200

**Don't Overthink It:** for this lab, we're going to keep the storage simple. In our separate performance tuning white paper, we talk about how to design the disk layouts for your data, logs, and TempDB. Even if you stand up your production VMs using this checklist, you can always add additional disks later, and put your user databases, log files, and TempDB over there.

Expand *Management, disk, networking, SSH keys* and then select *Networking*.

▼ Management, disk, networking, SSH keys

Specify the *Network* you created and which *Subnetwork* this VM should be in. Use a custom unused *Internal IP address* with no *External IP*. (We're not going to expose this SQL Server to the evil darkness that is the public Internet.) Enable *IP forwarding*.

The screenshot shows the Networking configuration page for a virtual machine. At the top, there are tabs for Management, Disks, Networking (which is selected and underlined), and SSH Keys. Below the tabs, there are several configuration fields:

- Network**: A dropdown menu showing "wsfcnet".
- Subnetwork**: A dropdown menu showing "wsfcsubnet1".
- Internal IP**: A dropdown menu showing "Custom".
- Internal IP address**: An input field containing "10.0.0.14".
- External IP**: A dropdown menu showing "None".
- IP forwarding**: A dropdown menu showing "On".

It's time to *Create* the instance.

You will be billed for this instance. [Learn more](#)

**Create**

**Cancel**

Now create your next VM. It's going to be in a different subnet - remember how we talked earlier about the complexities of networking? That's why this document starts to get a little long. Sorry about that. (Not really. We get paid by the word. We tried adding in another ten pages with animated GIFs, but Google didn't agree with my suggestion that if pictures were worth a thousand words, then animated GIFs were surely worth ten thousand. They countered that they could do a tl;dr with "LOL," and there went my retirement plan.)

Pay-per-use:

Name 

 bou-sql2

Zone 

 us-central1-f

Machine type

 4 vCPUs 15 GB memory[Customize](#)

Containers

 Create VM instance running Docker containers

Boot disk 



New 200 GB SSD persistent disk

Image

SQL Server 2016 Enterprise on Windo...

[Change](#)

Identity and API access 

Service account 

 Compute Engine default service account

Access scopes 

- Allow default access
- Allow full access to all Cloud APIs
- Set access for each API

Firewall 

Add tags and firewall rules to allow specific network traffic from the Internet

 Allow HTTP traffic Allow HTTPS traffic[Management](#)[Disks](#)[Networking](#)[SSH Keys](#)

Network 

 wsfcnet

Subnetwork 

 wsfcsubnet2

Internal IP 

 Custom

Internal IP address

© 2017 Brent Ozar Unlimited®. For comments, questions, and the latest version free, visit: <https://www.BrentOzar.com/go/gce>  
10.1.0.14

External IP 

 None

Bring your own licenses:

Name   
bou-sql2

Zone   
us-central1-f

Machine type  
4 vCPUs  15 GB memory [Customize](#)

Boot disk   
 New 200 GB SSD persistent disk  
Image  
Windows Server 2016 [Change](#)

Identity and API access   
Service account   
Compute Engine default service account 

Access scopes   
 Allow default access  
 Allow full access to all Cloud APIs  
 Set access for each API

Firewall   
Add tags and firewall rules to allow specific network traffic from the Internet

Allow HTTP traffic  
 Allow HTTPS traffic

Management Disks [Networking](#) [SSH Keys](#)

Network   
wsfcnet

Subnetwork   
wsfcsubnet2

Internal IP   
Custom

Internal IP address  
10.1.0.14

©2017 Brent Ozar Unlimited®. For comments, questions, and the latest version free, visit: <https://www.BrentOzar.com/go/gce>

None

IP forwarding   
On

Your VM Instances should now look similar to this:

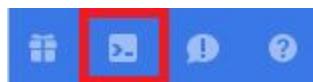
VM instances							
		CREATE INSTANCE		CREATE INSTANCE GROUP		REFRESH	START
				Columns	Labels	Recommendations	
Name	Zone	Machine type	Recommendation	In use by	Internal IP	External IP	Connect
bou-sql1	us-central1-f	4 vCPUs, 15 GB			10.0.0.14	None	RDP
bou-sql2	us-central1-f	4 vCPUs, 15 GB			10.1.0.14	None	RDP
dc-windows	us-central1-f	1 vCPU, 3.75 GB			10.2.0.100	104.154.31.198	RDP

## Creating the VMs Using the Command Line

With Google, there are two ways you can run commands:

- The Google Cloud Shell, an online terminal: <https://cloud.google.com/shell/>
- Your own command prompt (Windows, Linux, Mac) by installing the [Google Cloud SDK](#)

We're not going to walk you through installing the SDK, but if you're going to do a lot of cloud work long term, you'll want to do that. Long term, it's way easier to just fire up your local command prompt and run a few scripts to configure your cloud. If you're just playing around for now, though, use the Google Cloud Shell in your browser and you won't have to worry about network connectivity and firewalls.



Pay-per-use:

```
gcloud compute instances create bou-sql1 --machine-type n1-standard-4 --boot-disk-type pd-ssd --boot-disk-size 200GB --image-project windows-sql-cloud --image-family sql-ent-2016-win-2016 --zone us-central1-f --subnet wsfcsubnet1 --private-network-ip=10.0.0.4 --can-ip-forward
```

```
gcloud compute instances create bou-sql2 --machine-type n1-standard-4 --boot-disk-type pd-ssd --boot-disk-size 200GB --image-project windows-sql-cloud --image-family sql-ent-2016-win-2016 --zone us-central1-f --subnet wsfcsubnet2 --private-network-ip=10.1.0.4 --can-ip-forward
```

Bring your own licenses:

First, create a disk and a Windows image with the *guestOSFeatures* enabled.

```
gcloud compute disks create windows-server-2016-disk --size 200 --zone us-central1-f --type pd-ssd --image-family windows-2016 --image-project windows-cloud
```

```
gcloud beta compute images create windows-server-2016 --source-disk  
windows-server-2016-disk --source-disk-zone us-central1-f --guest-os-features  
MULTI_IP_SUBNET
```

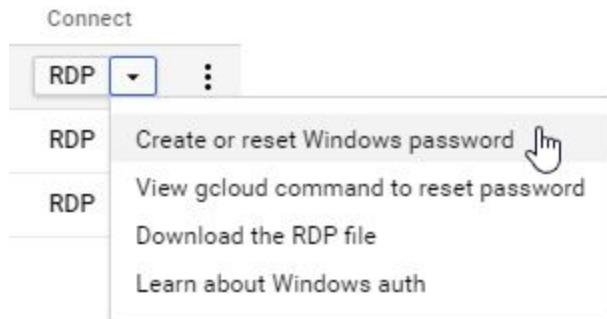
Then create the VMs using the image just created.

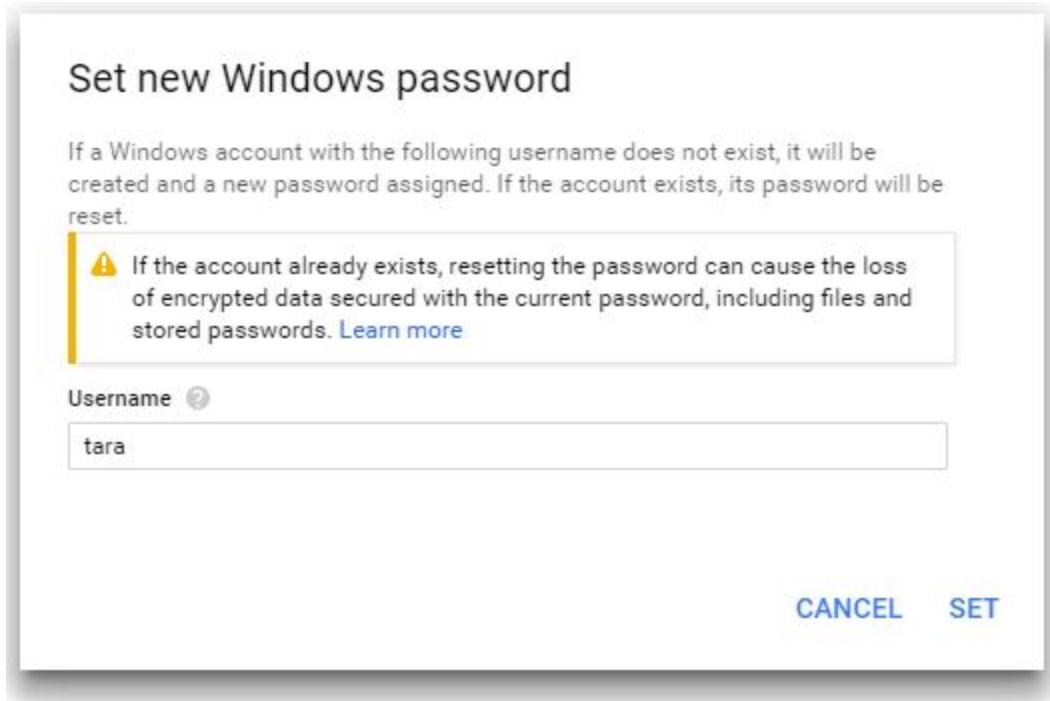
```
gcloud compute instances create bou-sql1 --machine-type n1-standard-4 --boot-disk-type  
pd-ssd --boot-disk-size 200GB --image windows-server-2016 --zone us-central1-f --subnet  
wsfcsubnet1 --private-network-ip=10.0.0.4 --can-ip-forward
```

```
gcloud compute instances create bou-sql2 --machine-type n1-standard-4 --boot-disk-type  
pd-ssd --boot-disk-size 200GB --image windows-server-2016 --zone us-central1-f --subnet  
wsfcsubnet2 --private-network-ip=10.1.0.4 --can-ip-forward
```

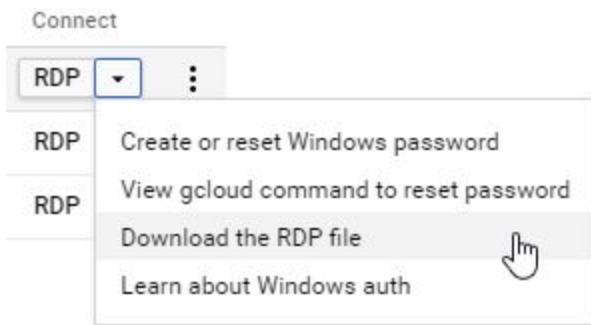
## Connecting to the VMs via Remote Desktop

The VMs have not yet been joined to the domain (a requirement for most Always On setups), so you'll need to create a local user on each and then RDP to them.





To RDP to a VM, you'll need to download the RDP file and then use that to connect. (Mac users can open this file with the free [Microsoft Remote Desktop app](#) from the Mac App Store.)



Note: If you restart the server, you may need to download a new file since the external IP may have changed.

Double click on the downloaded RDP file and use the local account you created to connect.

If you get a timeout error, most likely you're dealing with a network firewall somewhere between you and your cherished virtual machines. Check with your local sysadmins to make sure port 3389 is open out of your own environment, then if another team is managing your Google Cloud networking, make sure they haven't blocked 3389 there.

## Configuring the VMs' Security and Networks

RDP to each VM and add the appropriate domain users/groups to the Administrators group. You'll need to use a domain account to setup the cluster, so be sure to do this step before you get to the next section. We cover more details about this in our SQL Server Setup Guide in our First Responder Kit: <https://www.brentozar.com/first-aid/>

If you are using *Bring your own licenses*, update all components to the latest version, such as .NET Framework and then install SQL Server. If you are using *Pay-per-use*, this has already been done.

Change to static IP by running the following commands in an admin command prompt. If you haven't used netsh before, it's a command-line way of reconfiguring your network. Here's where to learn more about other things it can do:

[https://technet.microsoft.com/en-us/library/cc754516\(v=ws.10\).aspx](https://technet.microsoft.com/en-us/library/cc754516(v=ws.10).aspx)

Server1:

```
netsh interface ip set address name=Ethernet static 10.0.0.4 255.255.0.0 10.0.0.1 1  
netsh interface ip set dns Ethernet static 10.2.0.100
```

```
C:\windows\system32>netsh interface ip set address name=Ethernet static 10.0.0.4 255.255.0.0 10.0.0.1 1  
C:\windows\system32>netsh interface ip set dns Ethernet static 10.2.0.100
```

Server2:

```
netsh interface ip set address name=Ethernet static 10.1.0.4 255.255.0.0 10.1.0.1 1  
netsh interface ip set dns Ethernet static 10.2.0.100
```

```
C:\windows\system32>netsh interface ip set address name=Ethernet static 10.1.0.4 255.255.0.0 10.1.0.1 1  
C:\windows\system32>netsh interface ip set dns Ethernet static 10.2.0.100
```

Join the servers to the domain by executing the following command on both servers:

```
netdom join %computername% /domain:dbeng.com /userd:Administrator /passwordd * /reboot
```

```
C:\windows\system32>netdom join %computername% /domain:dbeng.com /userd:Administrator /passwordd * /reboot  
Type the password associated with the domain user:  
The command completed successfully.
```

In Google Cloud Shell, run the following commands to add the routes needed for Availability Groups.

```
gcloud compute routes create bou-sql1-route --network wsfcnet --destination-range 10.0.1.4/32  
--next-hop-instance bou-sql1 --next-hop-instance-zone us-central1-f --priority 1
```

```
gcloud compute routes create bou-sql2-route --network wsfcnet --destination-range 10.1.1.4/32  
--next-hop-instance bou-sql2 --next-hop-instance-zone us-central1-f --priority 1
```

```
gcloud compute routes create bou-sql1-route-listener --network wsfcnet --destination-range  
10.0.1.5/32 --next-hop-instance bou-sql1 --next-hop-instance-zone us-central1-f --priority 1
```

```
gcloud compute routes create bou-sql2-route-listener --network wsfcnet --destination-range  
10.1.1.5/32 --next-hop-instance bou-sql2 --next-hop-instance-zone us-central1-f --priority 1
```

```
gcloud compute routes list --filter="network:wsfcnet"
```

## Installing and Configuring the Windows Clustering Services

Always On Availability Groups, while not being your grandma's failover cluster with a shared quorum disk, still relies on Windows Failover Clustering Services in order to determine who owns stuff. We have to do two things: install these additional services, then configure them to group our two virtual machines together into a blood bond known as a cluster.

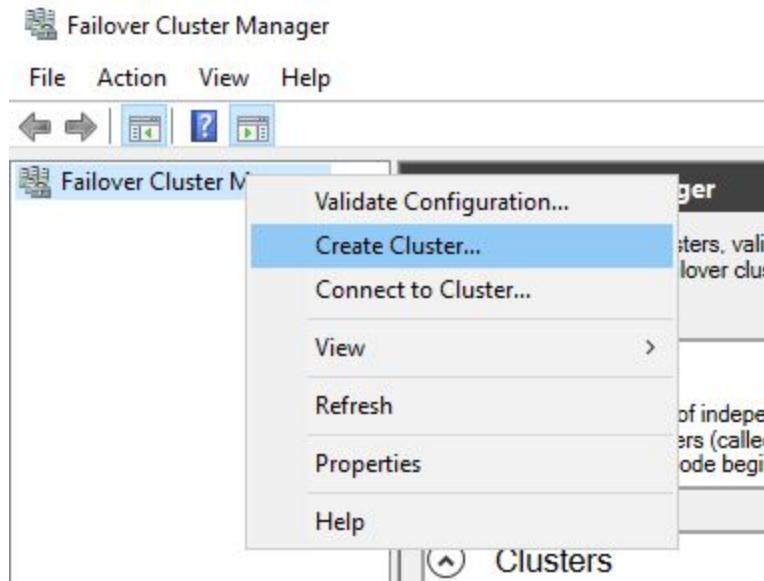
First, add the clustering tools to both servers by running the following command in an admin Powershell window:

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

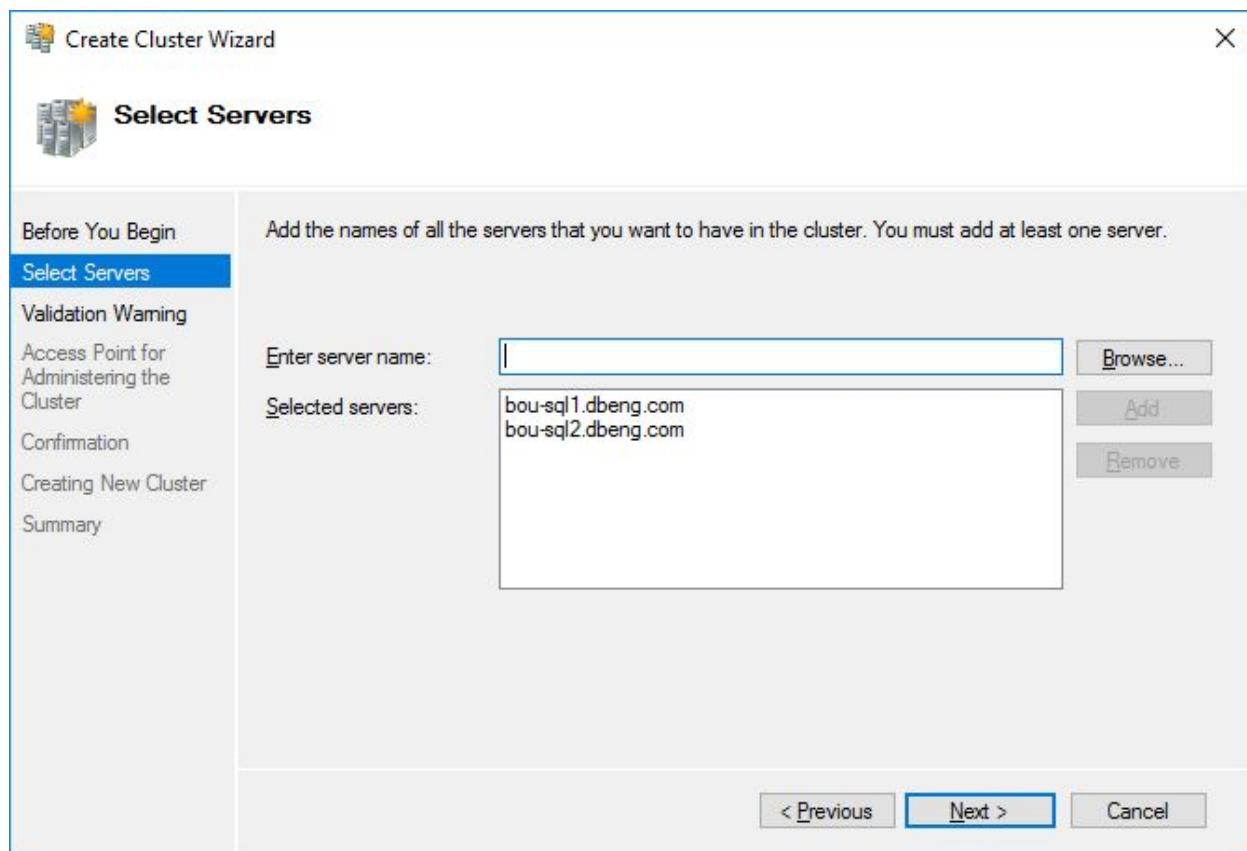
PS C:\windows\system32> <b>Install-WindowsFeature</b> Failover-Clustering -IncludeManagementTools				
Success	Restart Needed	Exit Code	-----	Feature Result
True	No	Success	-----	{Failover Clustering, Remote Server Admini...

RDP to one of the servers using a domain account that has administrative privileges.

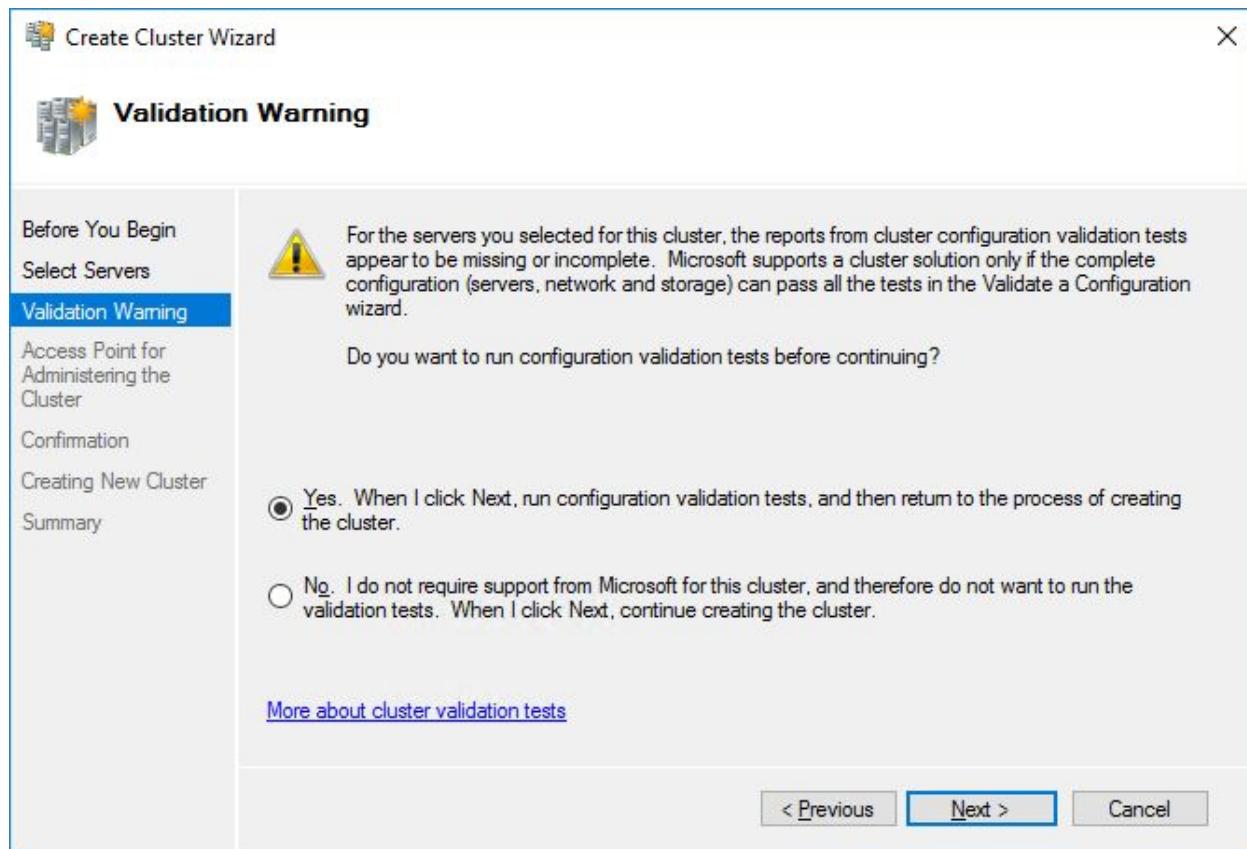
Open Failover Cluster Manager. Right click on *Failover Cluster Manager* and select *Create Cluster...*



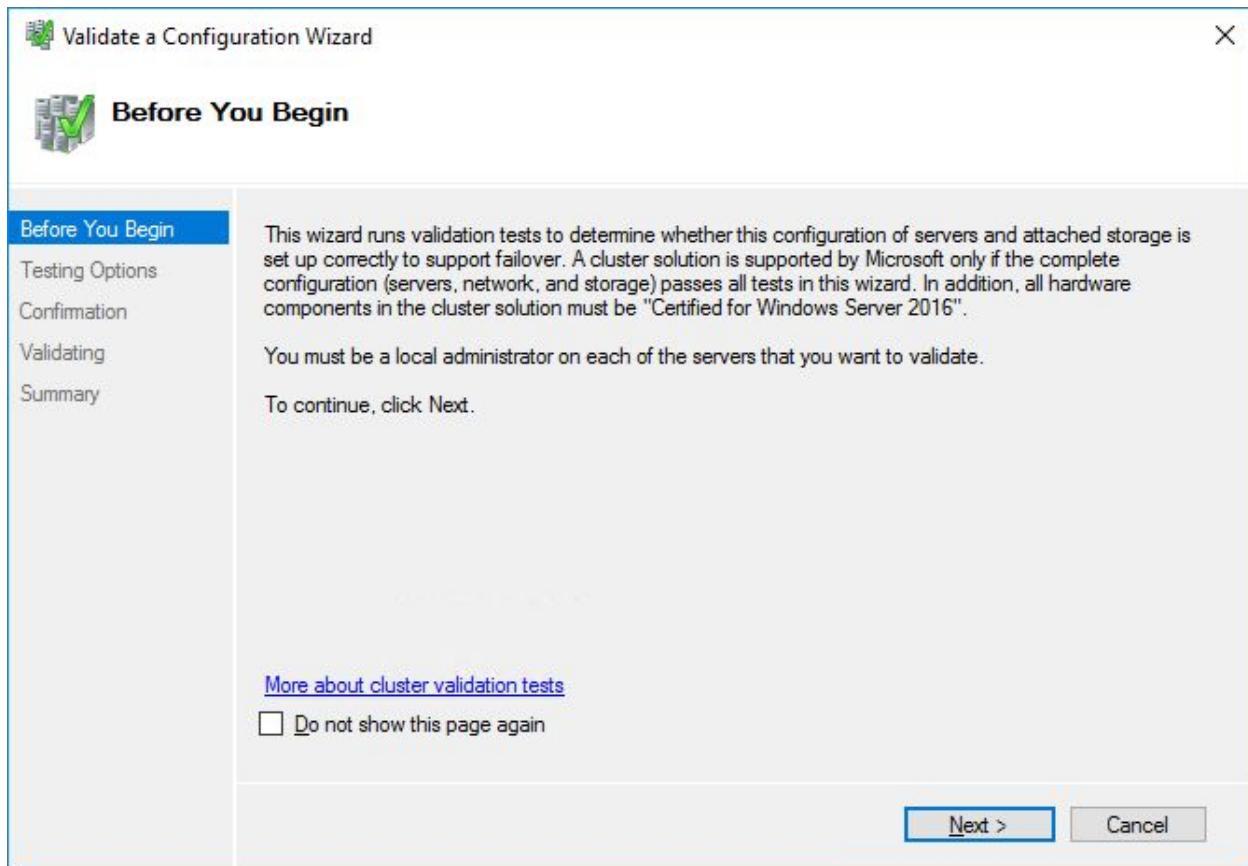
Click *Next* to go to the *Select Servers* page and add both servers.



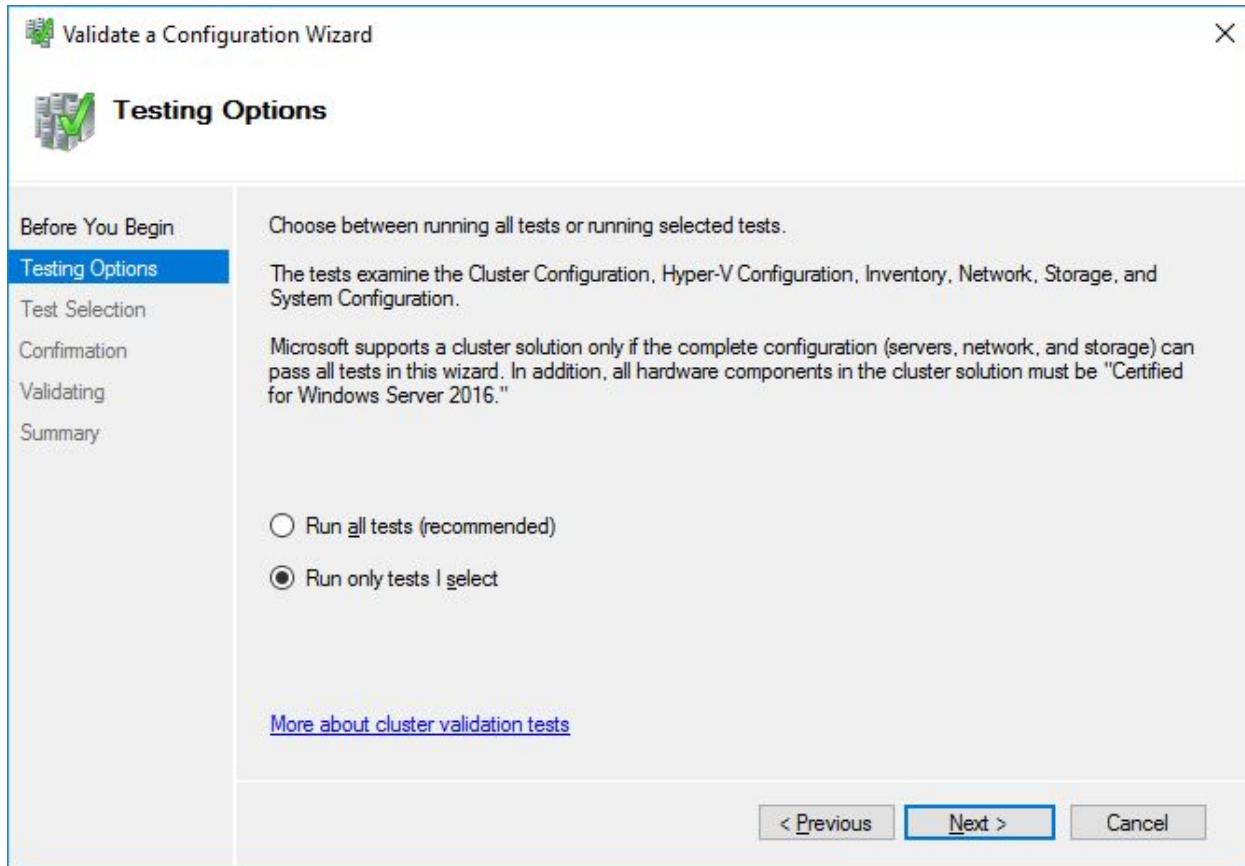
Click Next and keep the option to run configuration validation tests.



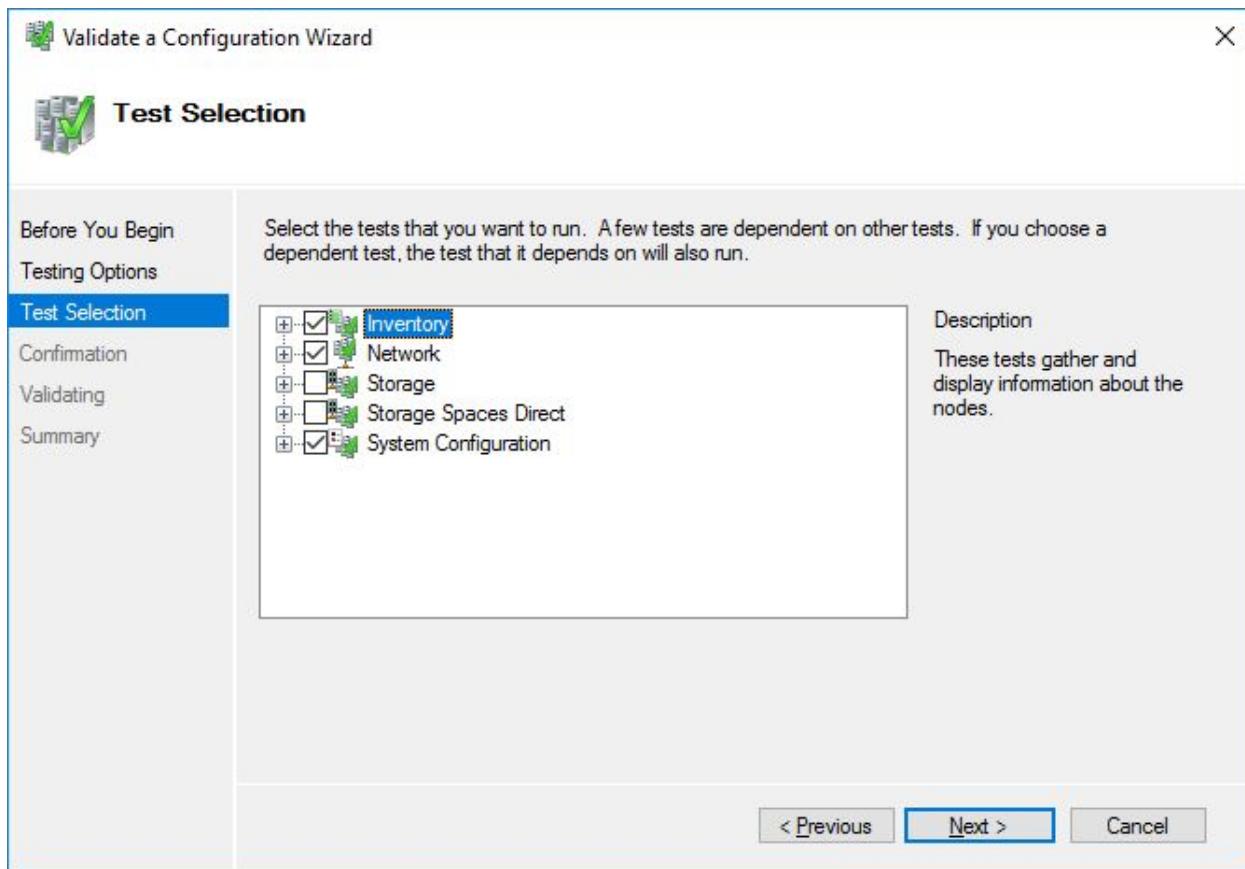
Click *Next* to get to the *Validate a Configuration Wizard* screen.



On the *Testing Options* page, switch it to *Run only tests I select* and click *Next*.



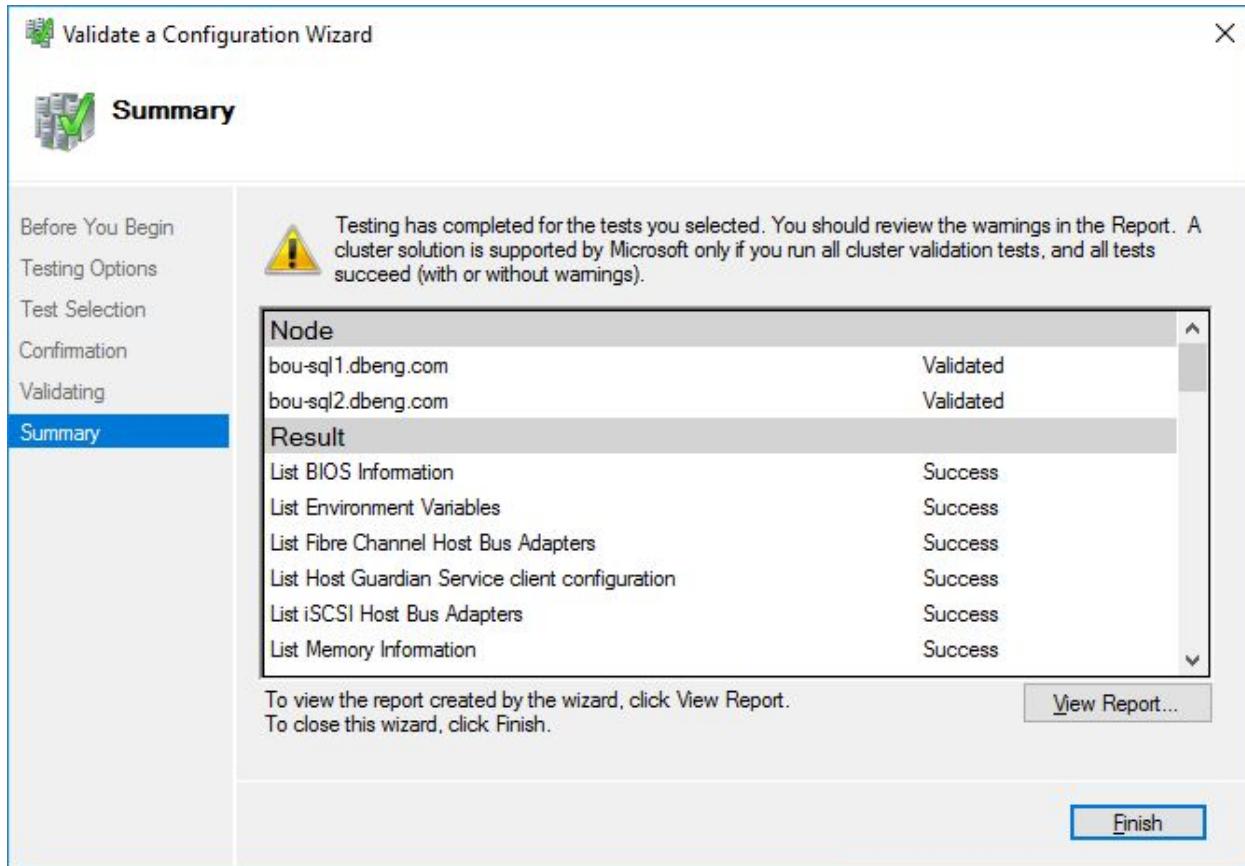
Unselect *Storage* on the *Test Selection* page as the *Storage* option will fail in GCE (as it would for separate standalone physical servers, too). It is not needed for Availability Groups - it's really for traditional failover clustered instances (FCIs) where every node needs to see the shared storage locations where data and log files reside.



Click *Next* twice, and it'll run the tests. Make sure none of the tests have failed.

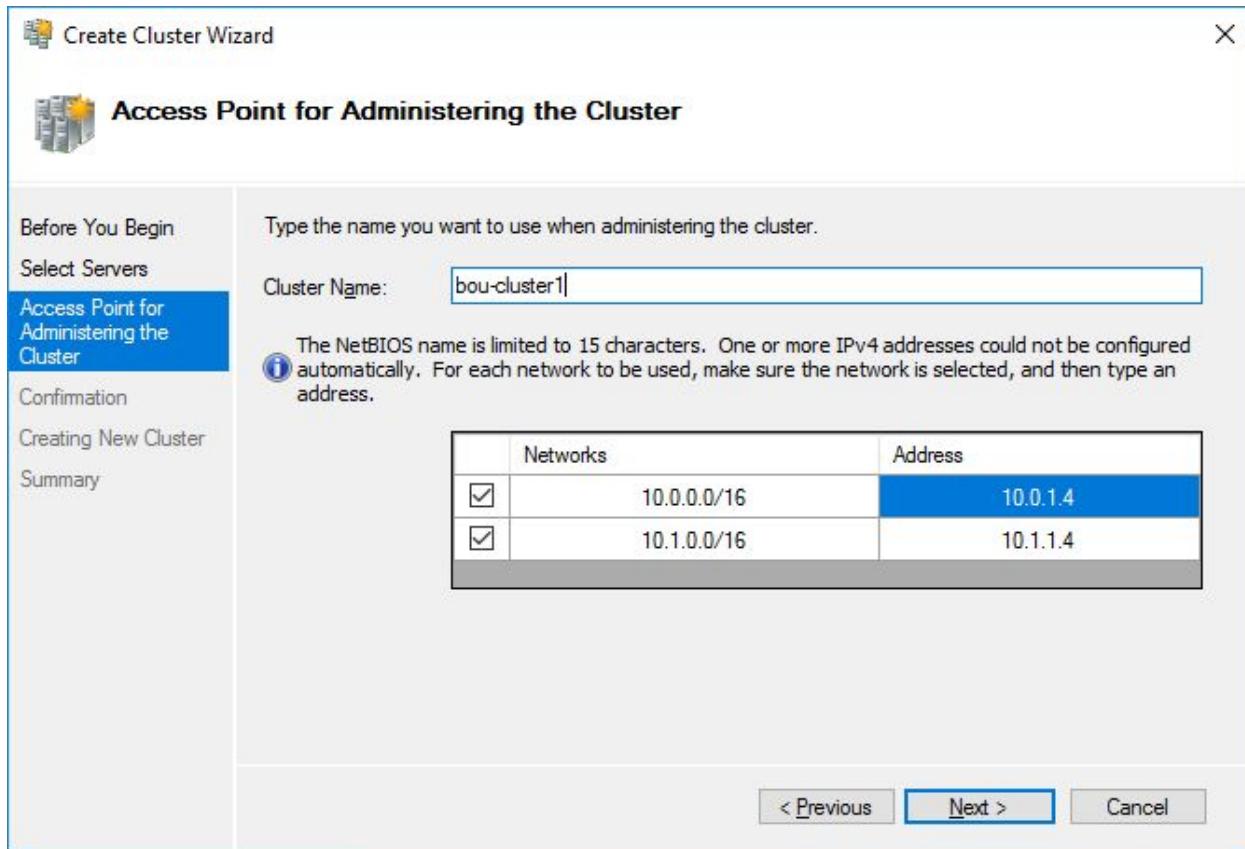
Common issues found during cluster validation include:

- Only one network path between replicas - in the old days, you would build a separate cluster heartbeat network. That's not the case today in the cloud, and you can ignore this one.
- Windows Updates not the same on both replicas - if you configured them to apply updates automatically, one of them might have applied updates that the other hasn't downloaded yet. Generally speaking, you don't want to automatically apply updates to SQL Server anyway.
- Pending reboot - you've made changes to one of the servers, and it needs a reboot to apply. Don't ignore this one.



**Decision Point:** before you click Finish, you really want these tests to succeed. Whenever we look at a cluster, we check Failover Cluster Validation to see if the tests have passed. If they haven't, we start to ask questions about whether this thing was actually tested, ever.

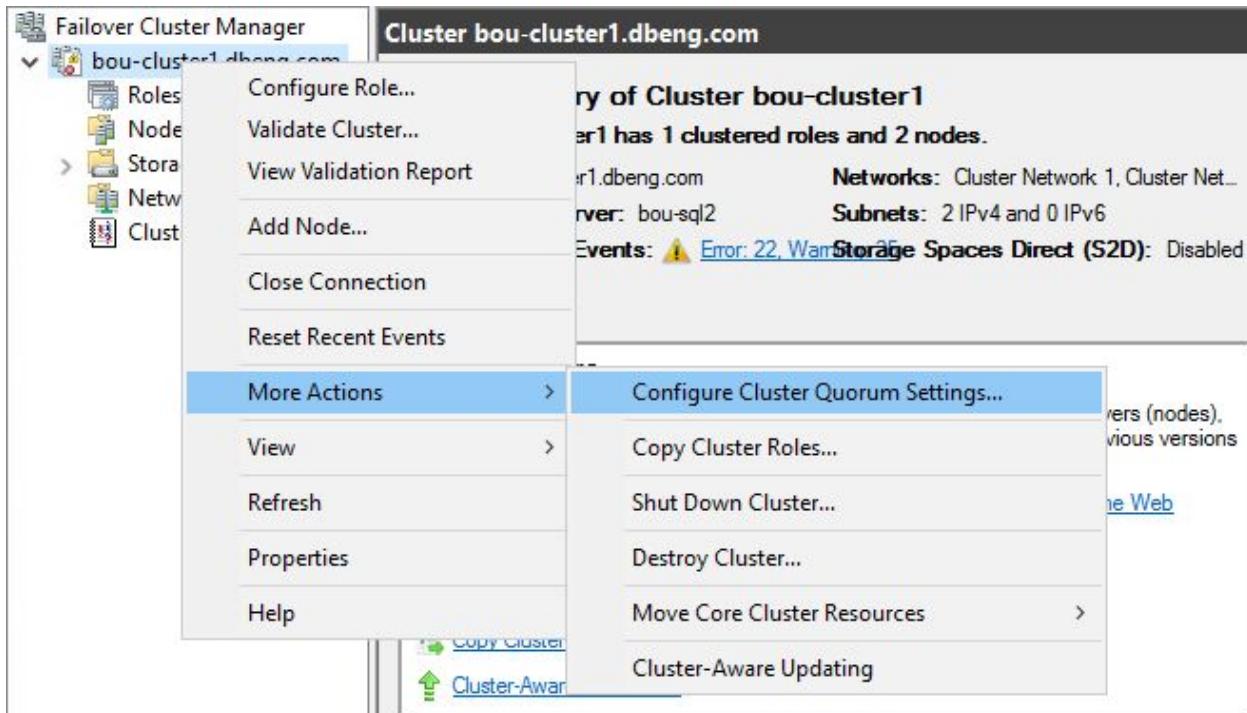
Click *Finish* to get back to *Create Cluster Wizard*. Name the cluster on the *Access Point for Administering the Cluster* page and specify the addresses that were used when you created the routes.



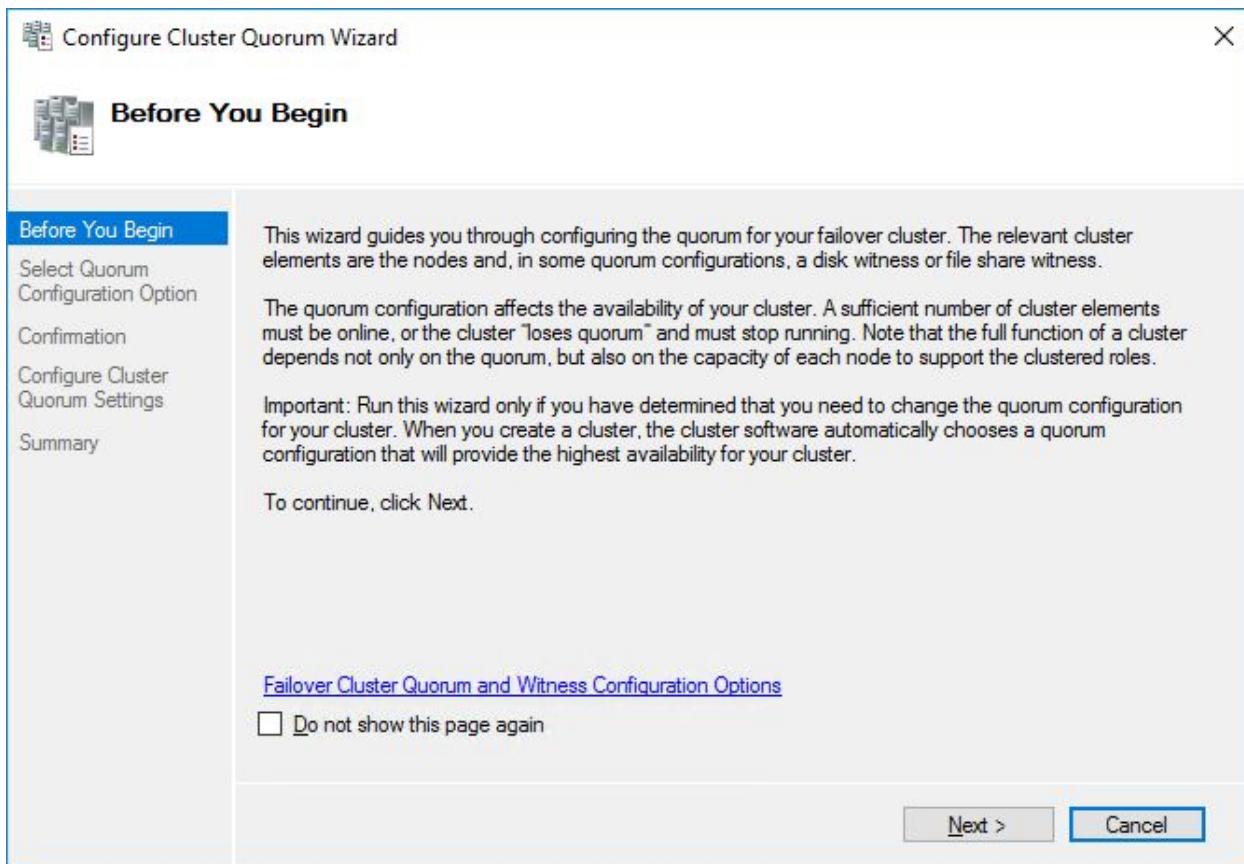
Click *Next* twice to create the cluster and then *Finish* to complete the wizard.

Setup a file share witness to achieve quorum. First create a file share on a server that is not part of this cluster. We used our domain controller: \\DC-WINDOWS\\Witness.

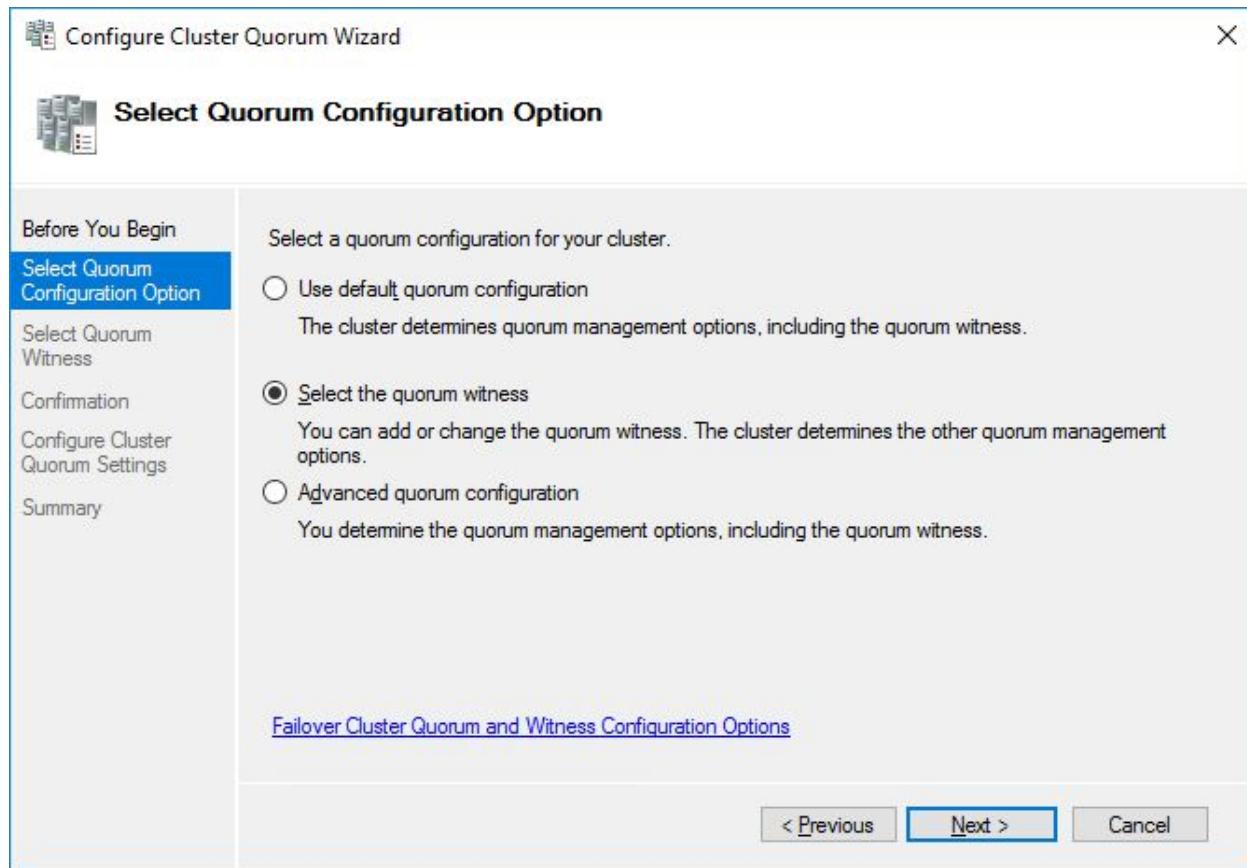
Right click on the cluster, select *More Actions* and then *Configure Cluster Quorum Settings*.



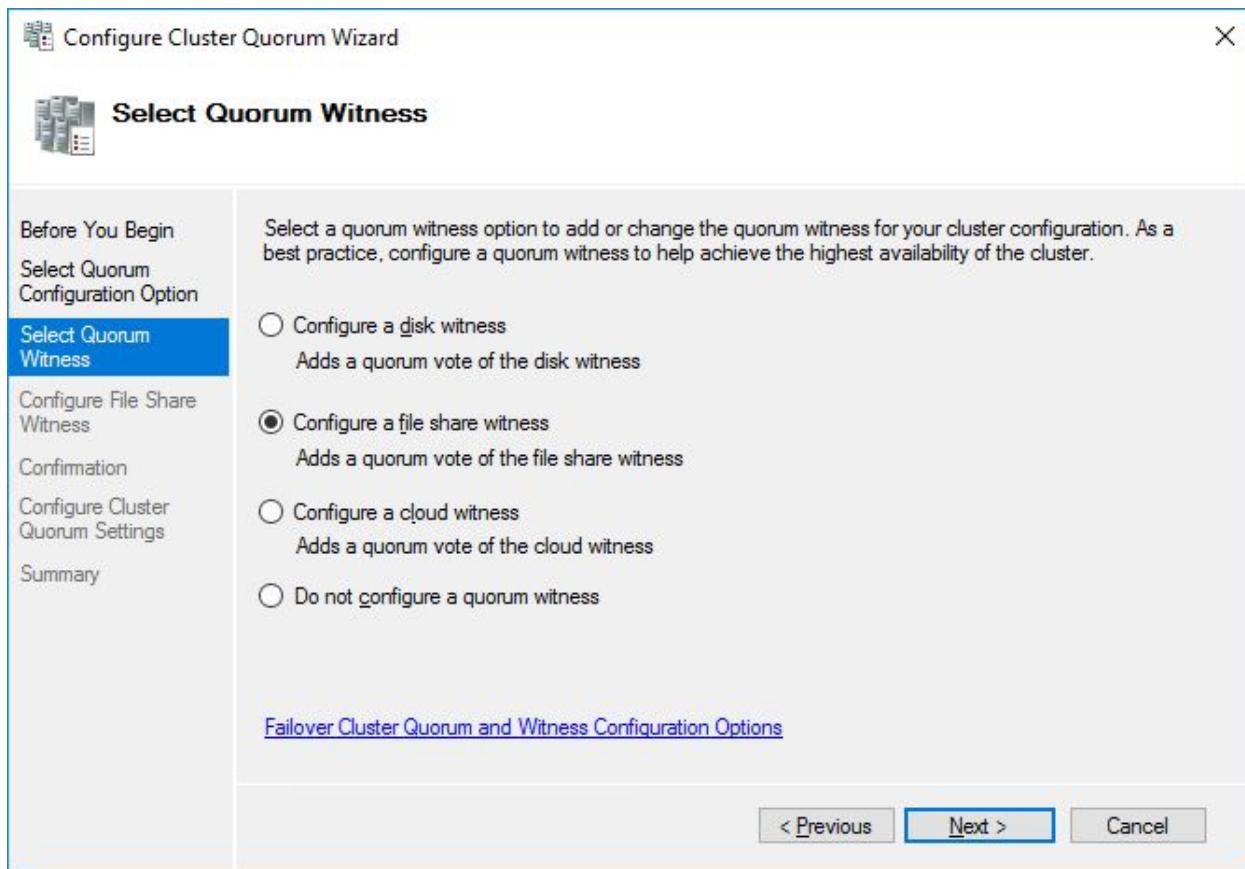
Click Next.



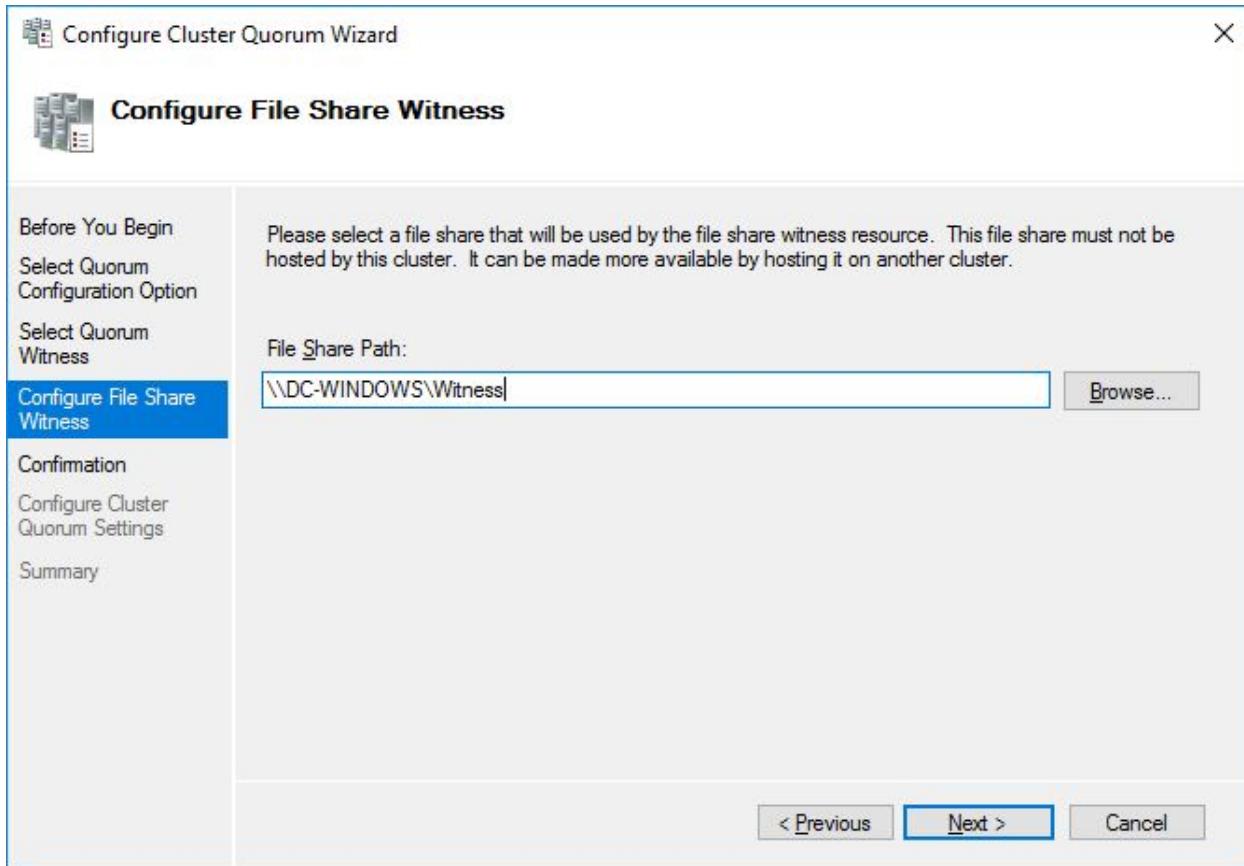
Select the option for *Select the quorum witness* and then click *Next*.



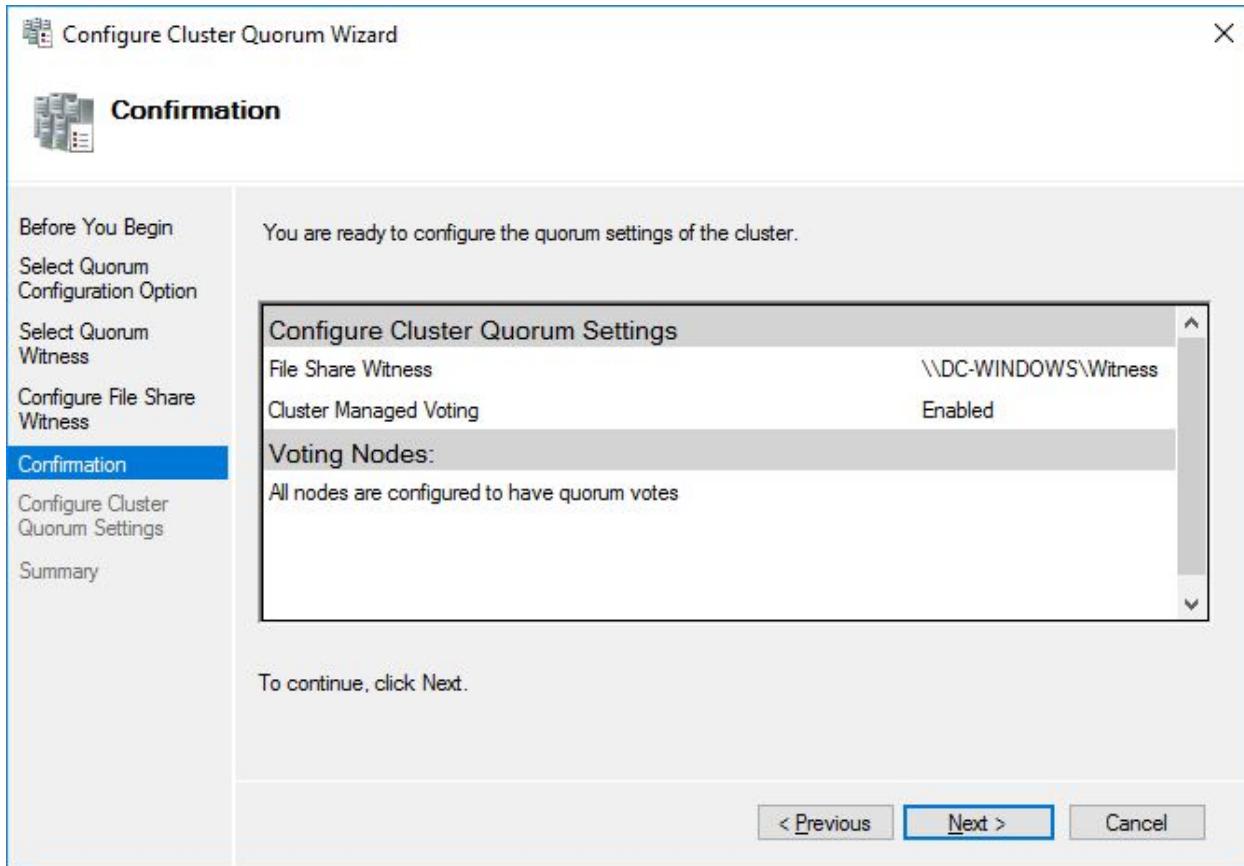
Select the option for *Configura a file share witness*.



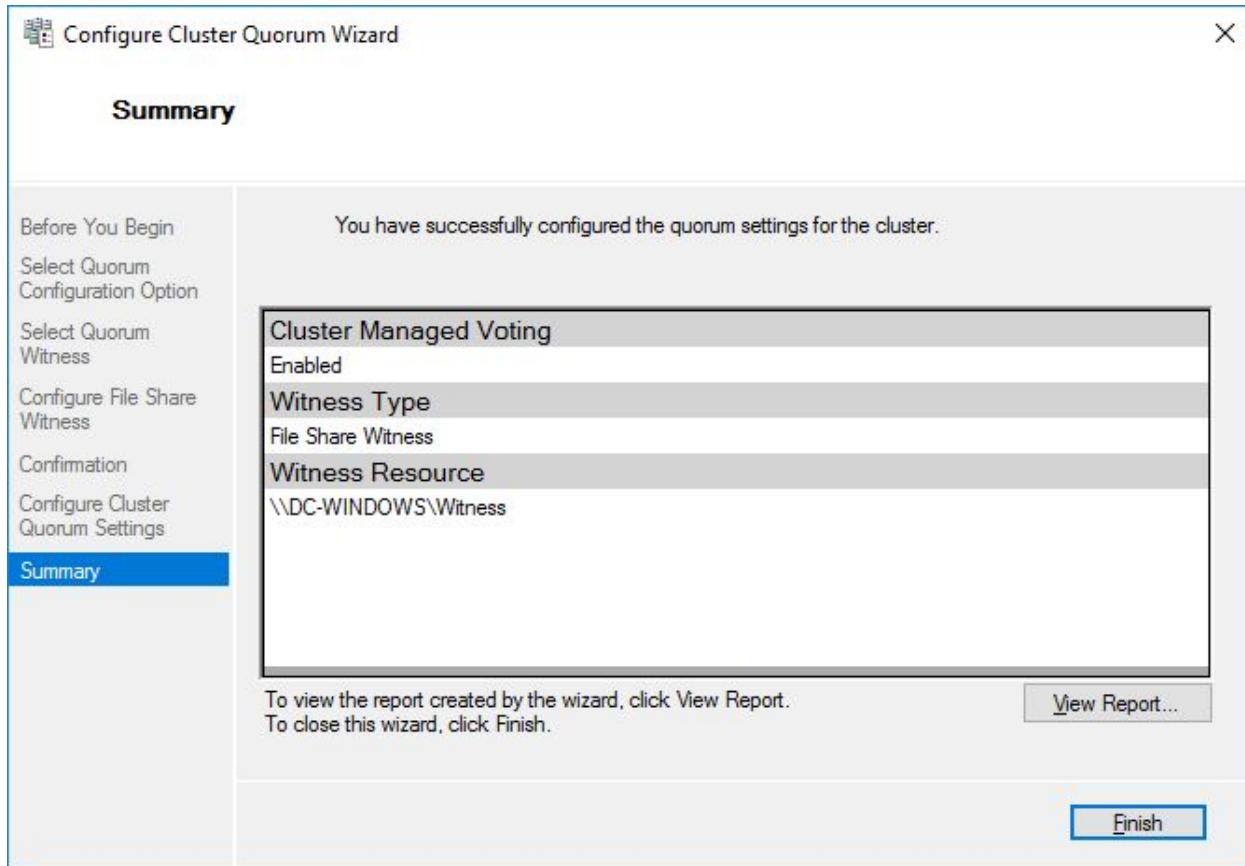
Input the path to the file share and then click **Next**.



Click *Next* after confirming the settings.

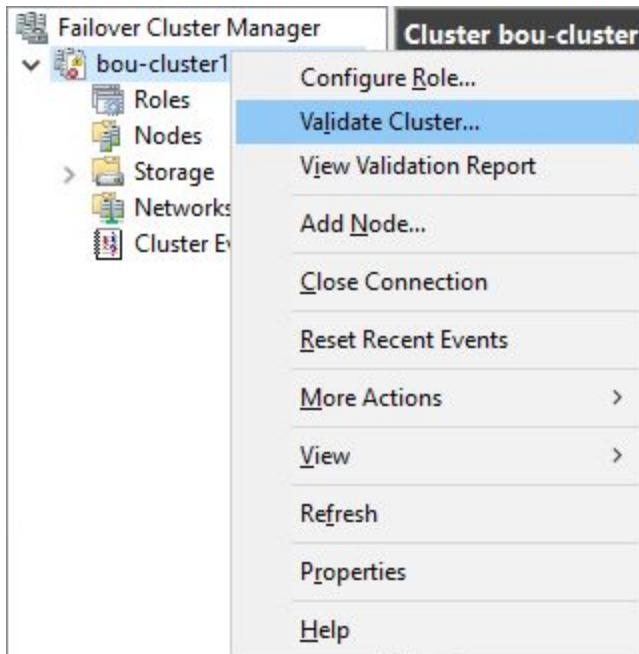


Click *Finish* to end the wizard.



It is recommended to run cluster validation tests after every cluster change.

Right click on the cluster and select *Validate Cluster*. Run through the wizard, ensuring all tests passed.



At this point, you've built the Windows clustering plumbing. Windows can now manage voting to understand which nodes own a particular service or function - and now, we can configure one of those functions, an Availability Group.

To be clear, we're not doing justice to all the complexities of Windows clustering. For production, you'd also want to consider things like:

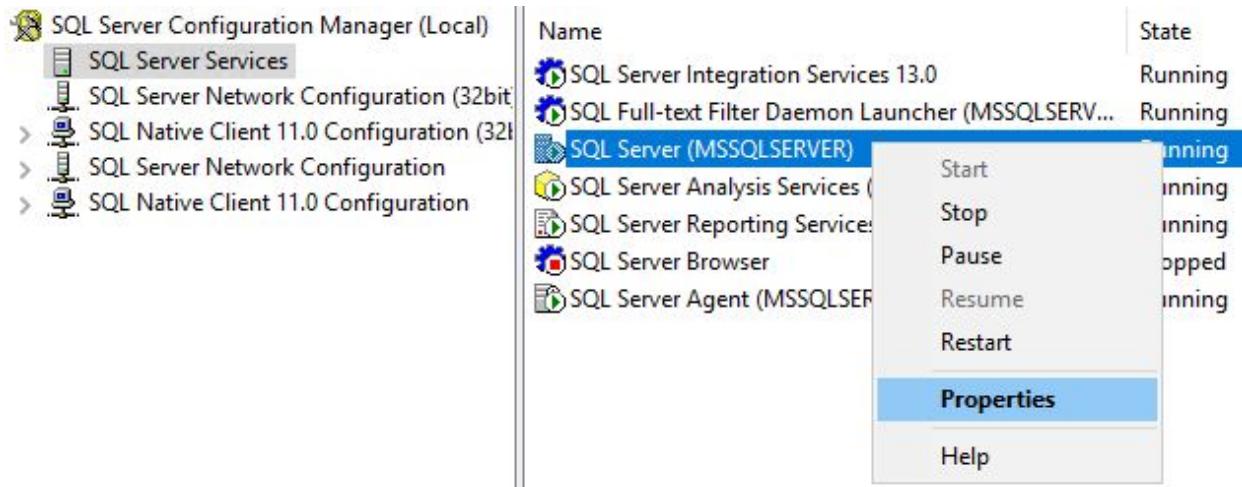
- Witness design - this Books Online page discusses Windows 2012, but the basics still apply to Win2016: <https://technet.microsoft.com/library/jj612870.aspx>
- Distributed transactions - newly supported with Always On Availability Groups in Windows 2016 & SQL 2016, but there's a huge catch that we'll discover later

We cover some of these in our SQL Server Setup Guide, and we go into more detail in our online training classes. For now, on with the application setup.

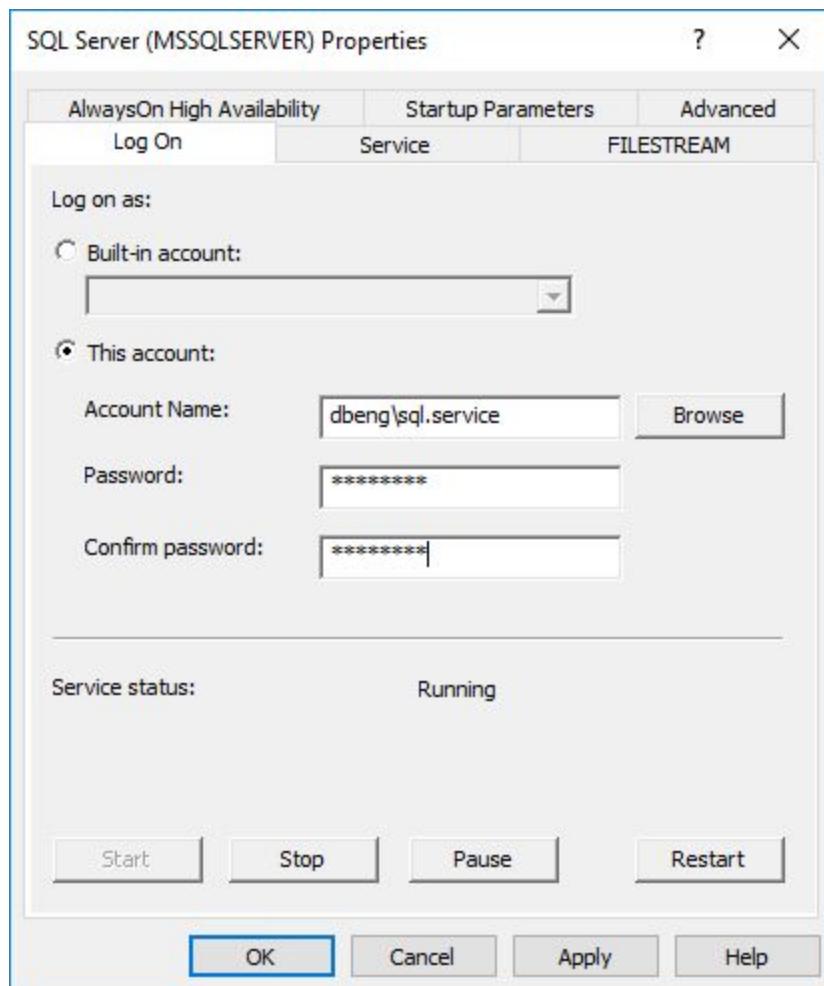
## Setting up the Availability Group

Do these steps on both nodes to enable the AG feature:

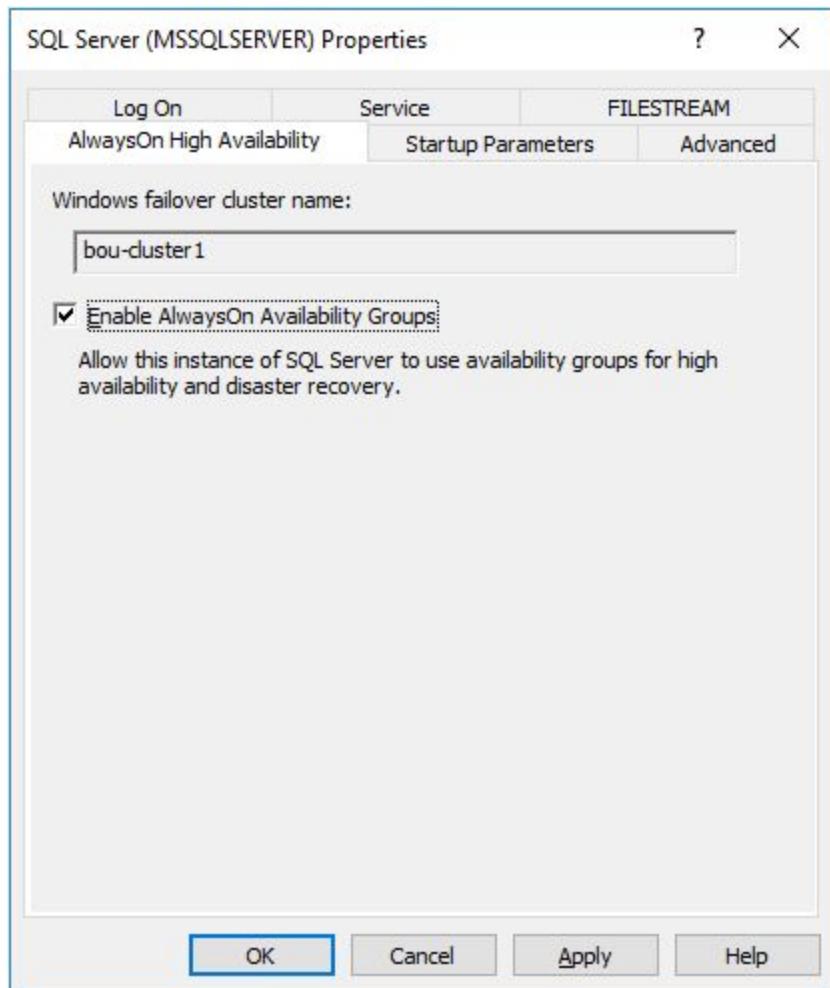
1. Open SQL Server Configuration Manager
2. Click *SQL Server Services* on the left
3. Right click on the SQL Server service and go to *Properties* or just double click on the SQL Server service



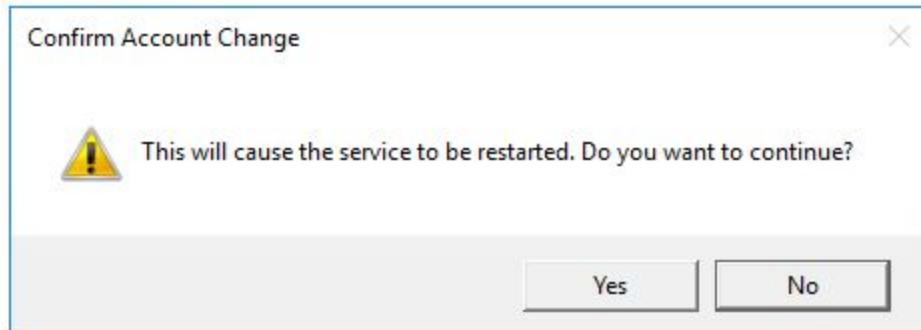
4. Change the account to dbeng\sql.service and provide the password



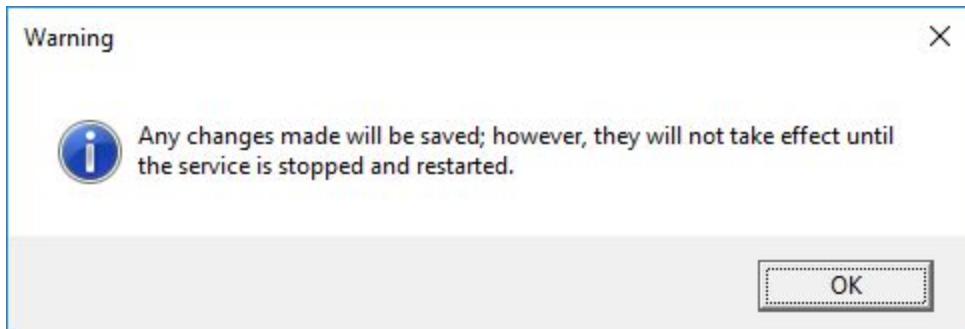
5. On the AlwaysOn High Availability tab, click the checkbox for *Enable AlwaysOn Availability Groups*



6. Click OK and then Yes on the restart warning



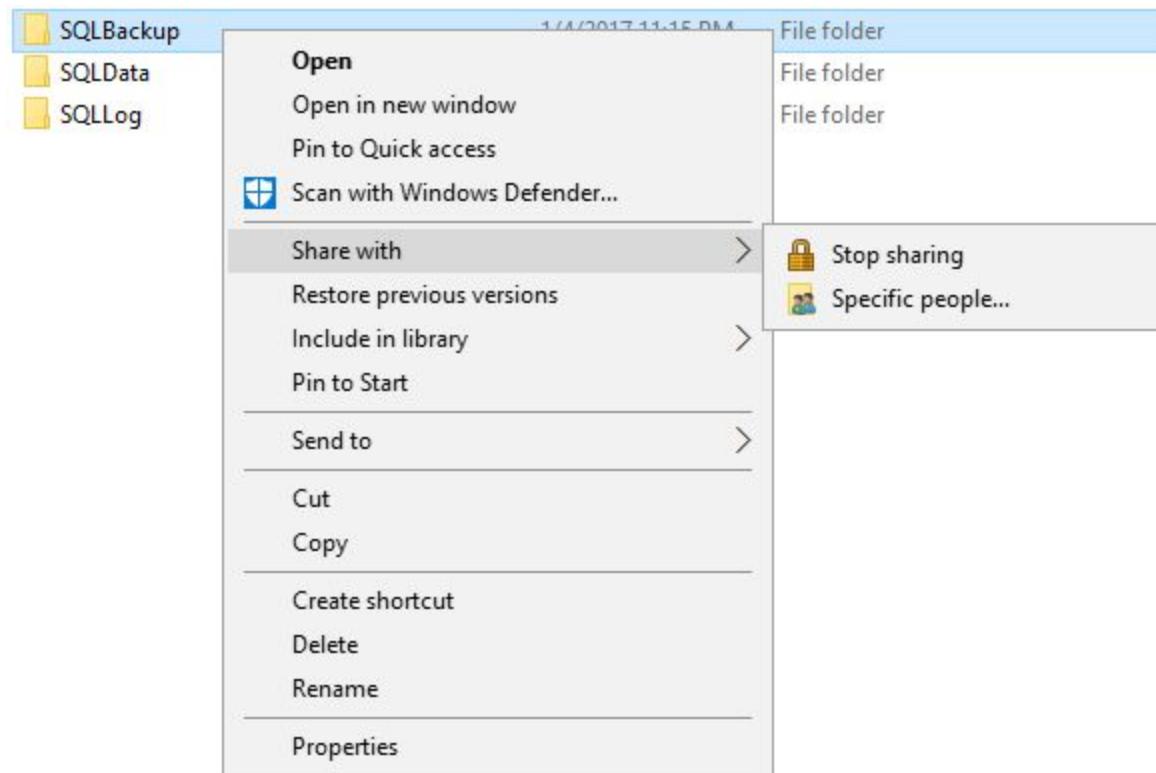
7. Click OK on the next warning

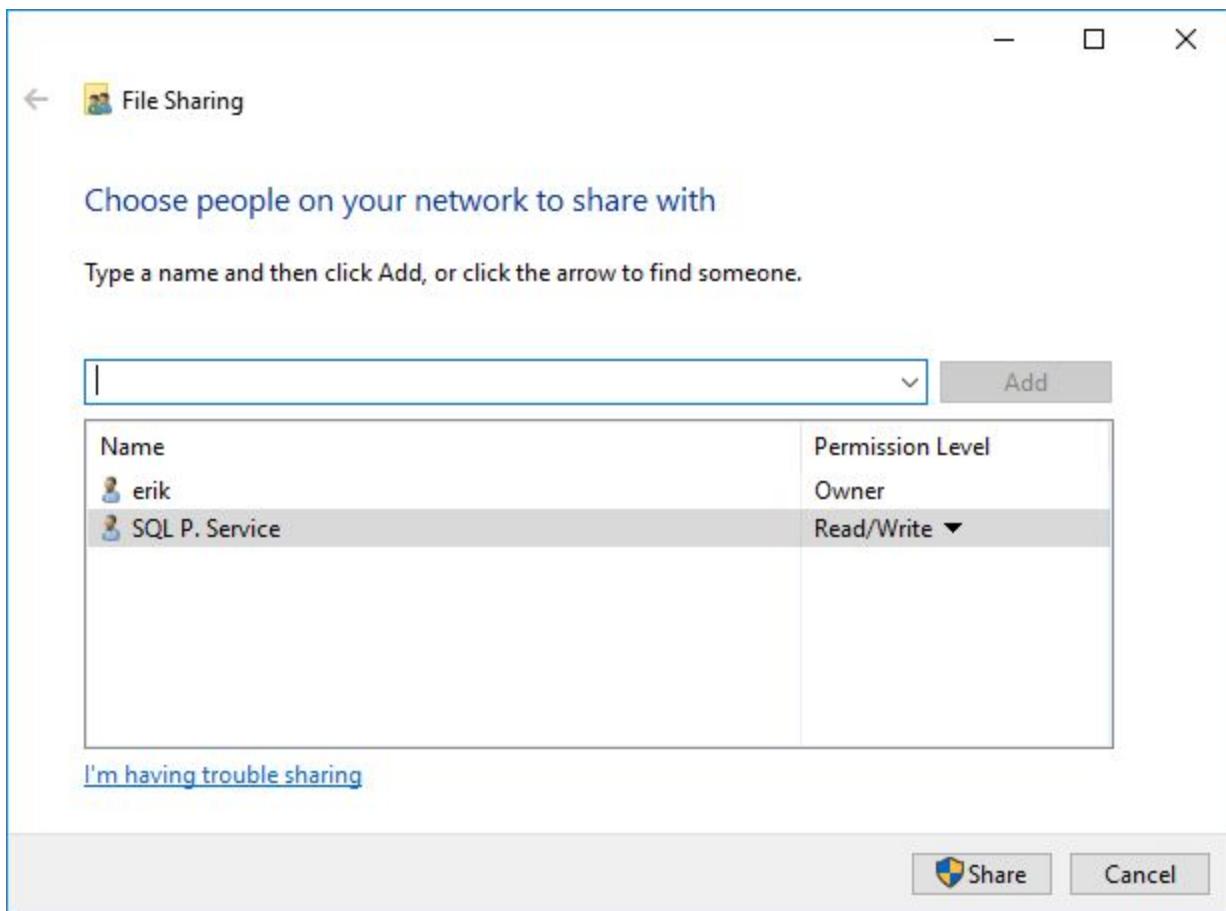


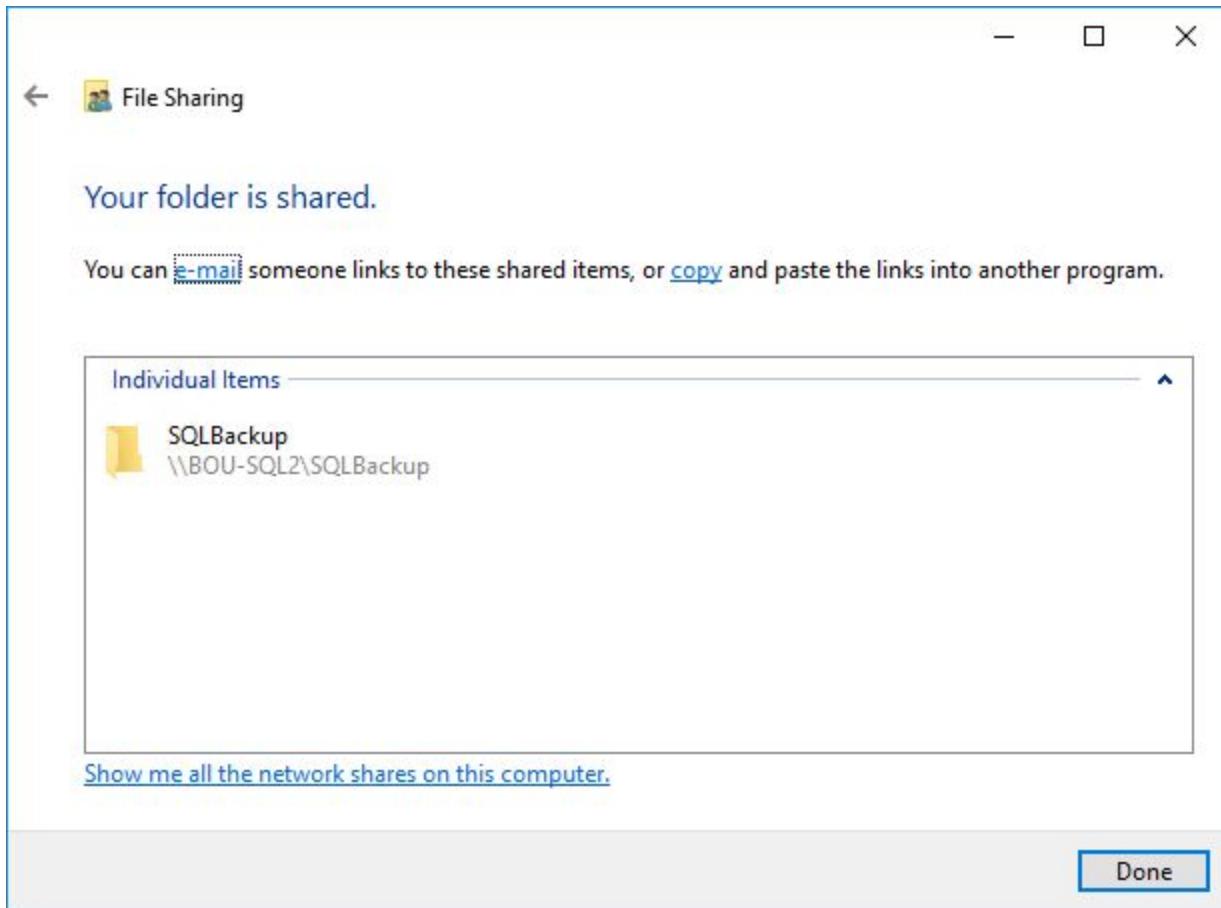
## 8. Restart the SQL Server service

Now that the SQL Server instances are ready for Availability Groups, setup a backup share that the SQL Server service account can read from and write to. You can do this on both nodes, but it is recommended that this share be a remote resource so that you have access to all of the backup files in the case of a restore.

For this white paper, we used C:\SQLBackup for simplicity on node2 and shared it as SQLBackup.







**Don't Overthink It:** Look, we told you this was for simplicity. Yes, you're going to draft us an ugly email right away condemning us because:

- The backups won't be available if this node goes down
- If backups aren't cleaned up regularly, the C drive will fill up, and the party will stop
- Backups will be slower on node1 than node2, since it'll be writing the files across the network
- If node1 is doing a backup, it will slow down node2 because it's pushing data into its network card
- If we run a backup app that sweeps these files to somewhere else for safekeeping, the SQL Server will slow down when that happens, because the backup app will be burning up bandwidth
- Somebody could download or delete these backups if security isn't locked down tight

We totally understand. You're right, and we're wrong.

Now let's carry on.

Run these commands in an admin command prompt on both nodes to open up ports 1433 and 5022 if you are using Windows Firewall (recommended).

```
netsh advfirewall firewall add rule name="Open Port 5022 for Availability Groups" dir=in  
action=allow protocol=TCP localport=5022
```

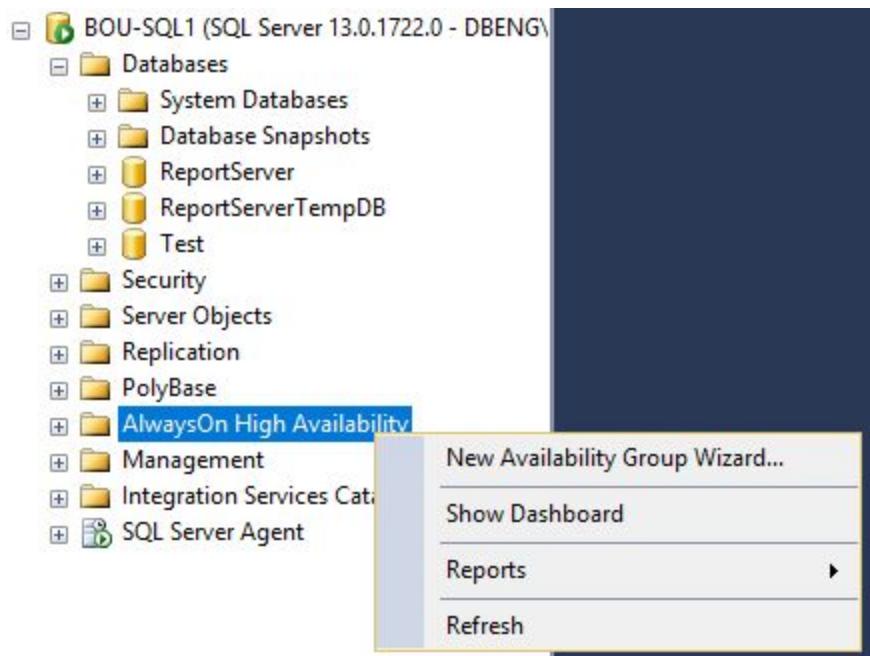
```
netsh advfirewall firewall add rule name="Open Port 1433 for SQL Server" dir=in action=allow  
protocol=TCP localport=1433
```

```
C:\windows\system32>netsh advfirewall firewall add rule name="Open Port 5022 for Availability Groups" dir=in action=allow  
protocol=TCP localport=5022  
Ok.  
  
C:\windows\system32>netsh advfirewall firewall add rule name="Open Port 1433 for SQL Server" dir=in action=allow protocol=  
TCP localport=1433  
Ok.
```

On node1, create a test database or restore your database. The recovery model must be set to Full. Perform a FULL backup as this is a prerequisite.

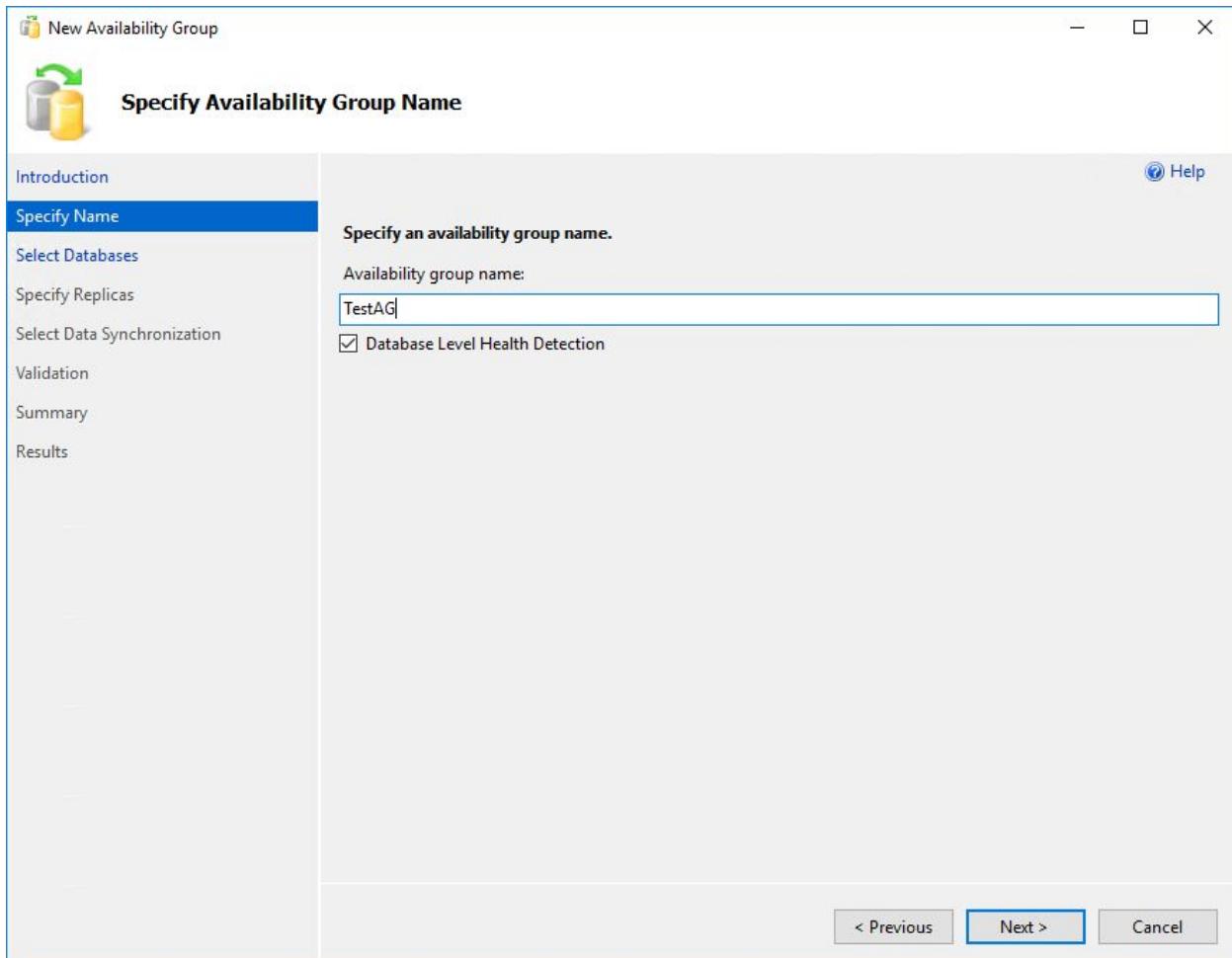
Now it's time to create the Availability Group. We love using TSQL commands - especially gems like KILL and SHUTDOWN WITH NOWAIT - but we would recommend using the UI to create your first AG. You can script it out at the end of the wizard to see what it takes to create an AG and reuse that script in the future.

Right click on *AlwaysOn High Availability* and select *New Availability Group Wizard...*

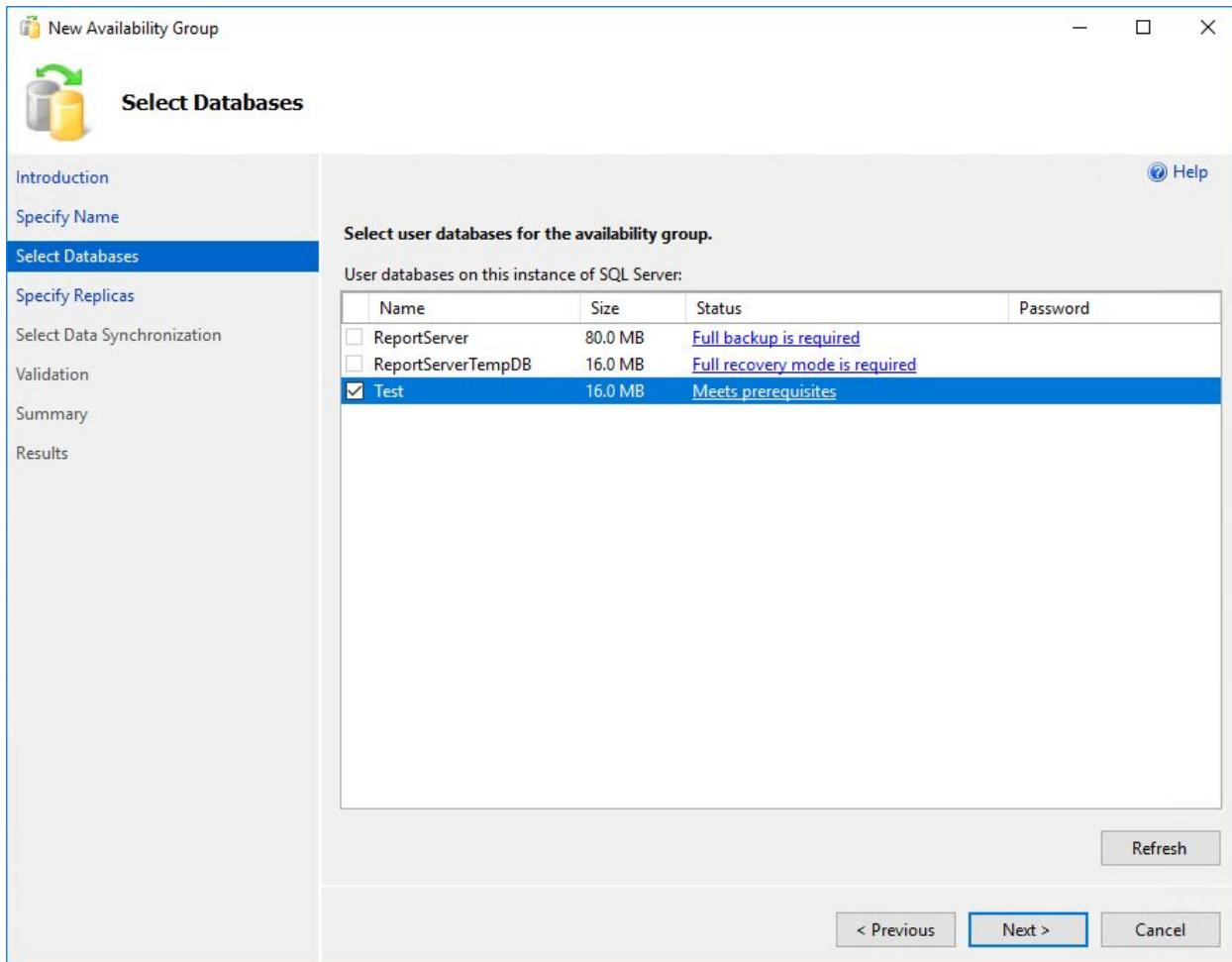


On the *Specify Name* page, specify a name for your AG and check the box for *Database Health Detection*. In SQL Server 2012 and 2014, a failover would not occur if the database became

inaccessible. In SQL Server 2016, a failover will occur if the database becomes inaccessible and if you have enabled this feature.



Click *Next* and then select the database you want to add to the AG.



Notice the prerequisites have not been met for ReportServer and ReportServerTempDB.

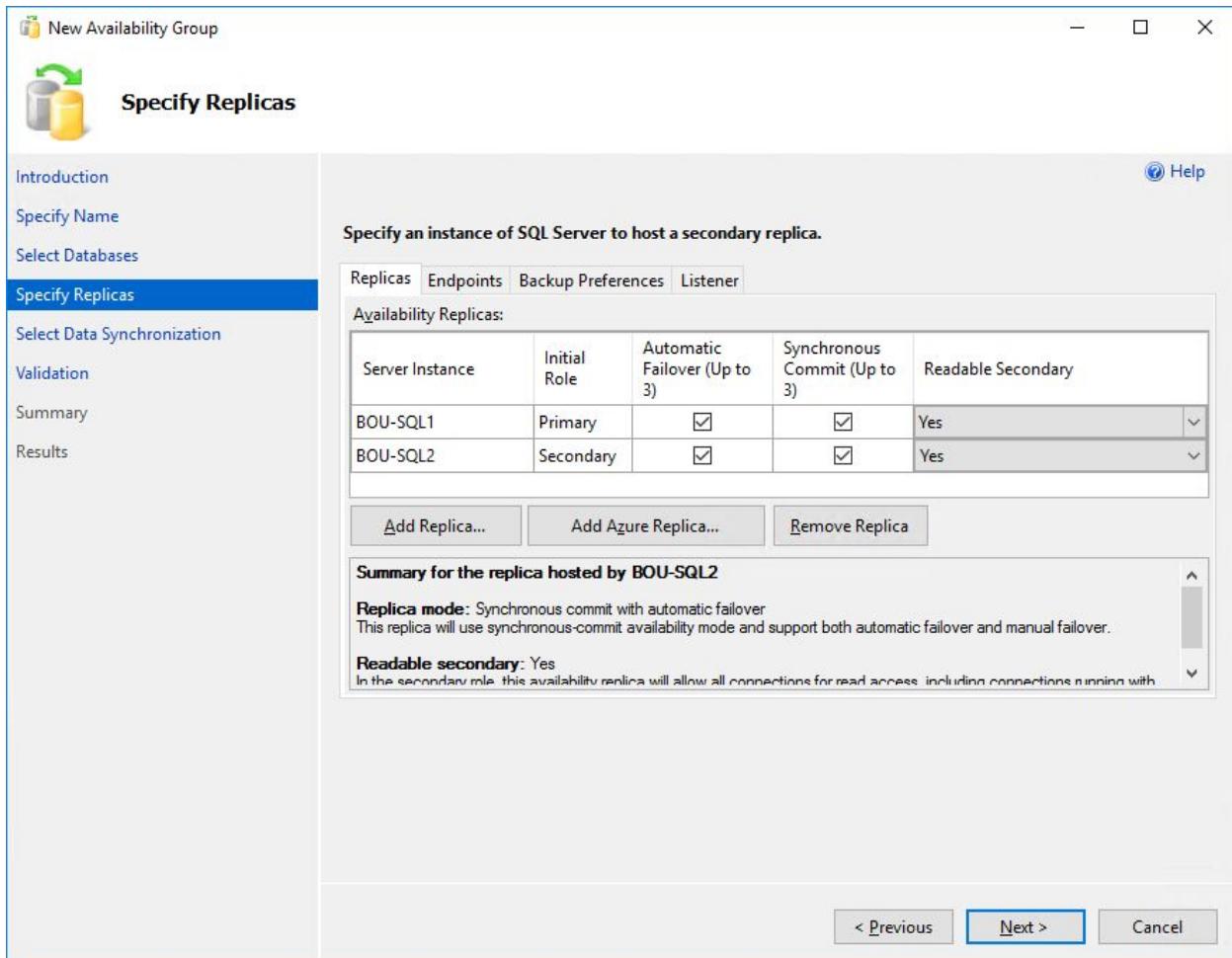
You might wonder why a full backup is required in order to put a database into an Availability Group. It turns out that SQL Server doesn't consider a database to really be in full recovery until the first full backup has been taken.

You might then wonder why that matters, since this very wizard we're using will actually take a full backup for you as part of the AG initialization process, and then restore that backup over on the secondary replica to seed it. We certainly wonder that.

We also wonder why the above screenshot says "full recovery mode" when it's really called full recovery model. Seriously, here's the Books Online page called Recovery Models, not Modes: <https://msdn.microsoft.com/en-us/library/ms189275.aspx>

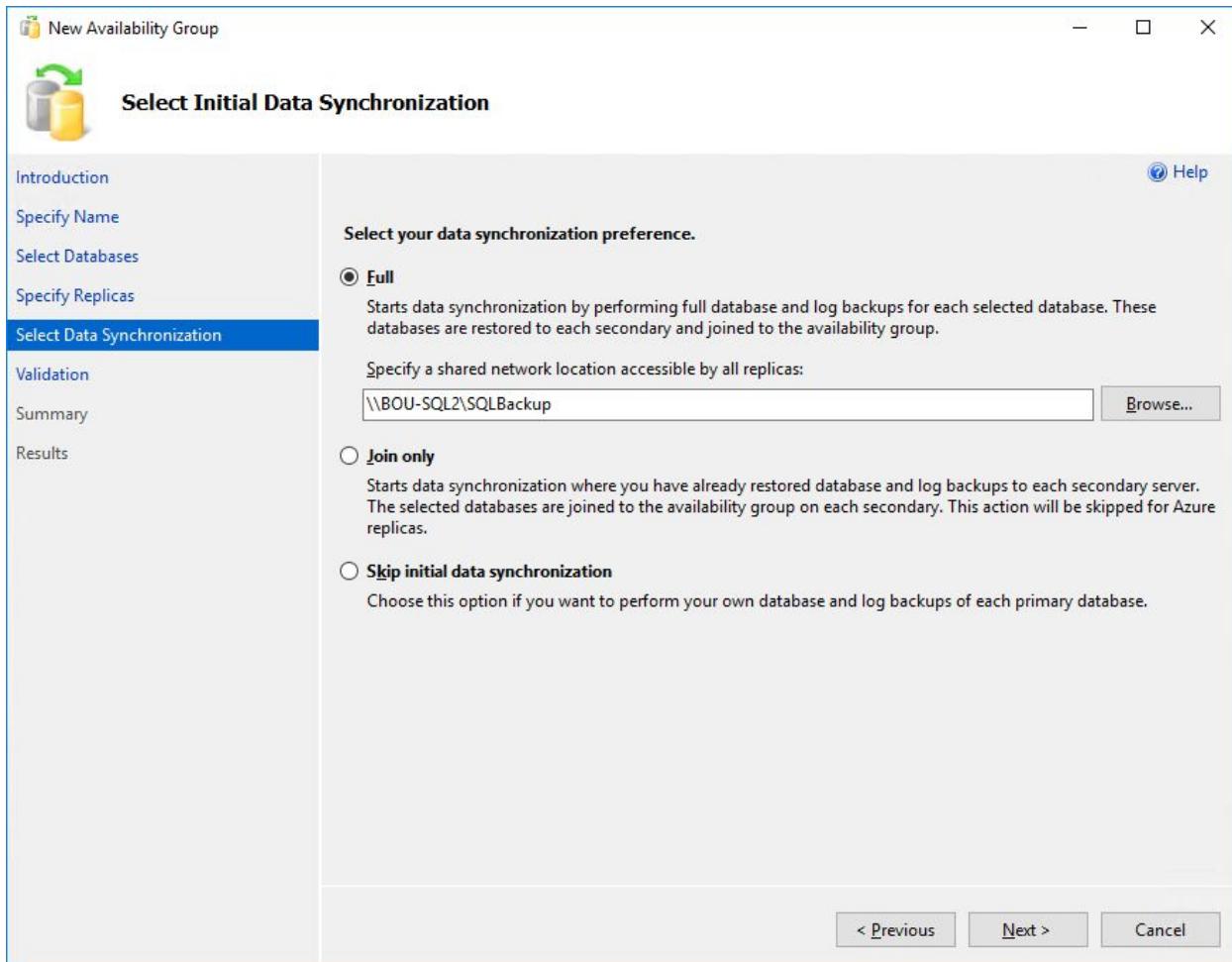
When we work with SQL Server, we're full of wonder. Sometimes it's the good kind of wonder. Sometimes it's not. Anyhoo, let's carry on.

Click *Next* and add node2, node1 will already be listed. Enable *Automatic Failover*, *Synchronous Commit*, and *Readable Secondary*. You can use *Yes* or *Read-intent only* for *Readable Secondary*. *Read-intent only* allows only read-intent connections, whereas *Yes* allows all connections. *Yes* is needed if you have applications using database drivers that do not support the *ApplicationIntent=ReadOnly* connection parameter.



**Don't Overthink It:** Don't worry about the other tabs for Endpoints, Backup Preferences, and Listener. If you're really bored, you could change the option in the *Backup Preferences* tab but don't add a listener as that will fail in GCE. The listener will be added later.

Click *Next* and have it perform a FULL backup to the backup share on node2.

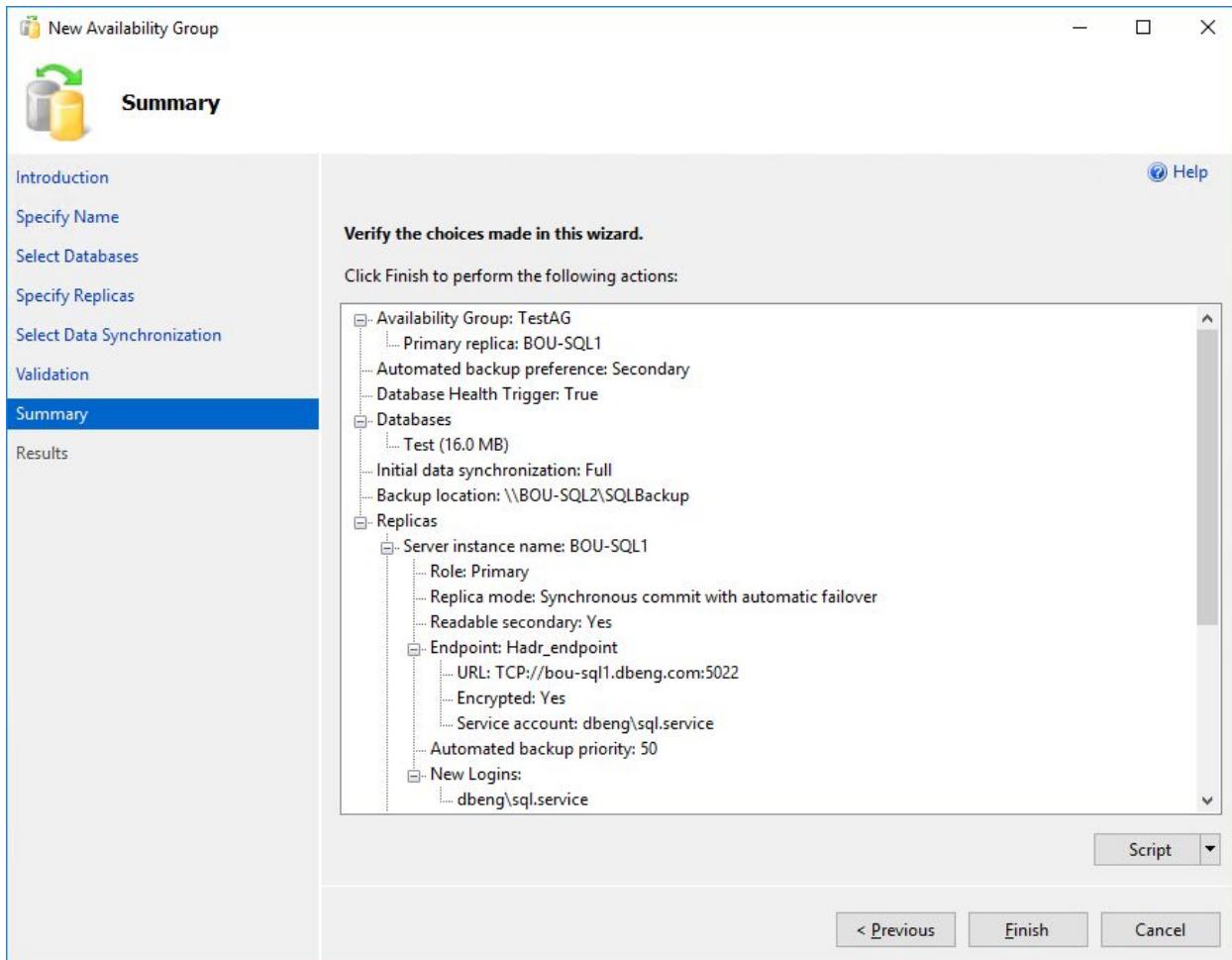


If you restored your own database and it is sizable, you could opt to use the previous FULL backup and the LOG backup chain. If you do this, you would restore the database and log chain and then select the *Join only* option.

SQL Server 2016 also adds Direct Seeding, the ability to seed replicas automatically without taking backups. That sounds great in theory, but comes with some gotchas that Erik Darling has blogged about: <https://www.brentozar.com/archive/tag/directseeding/>

Click *Next* and make sure all test passed with *Success*, with the exception of the listener test which will show *Warning*. The warning can be ignored as we will be creating the listener later as previously mentioned. If any tests failed, resolve those so that you can continue.

Click *Next* and then *Finish*. You could also script it out from this final page using the *Script* dropdown.



We like the passive language here: “Verify the choices made in this wizard.” Translation: if you got any of this wrong, it’s your own fault.

New Availability Group

Results

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

The wizard completed successfully.

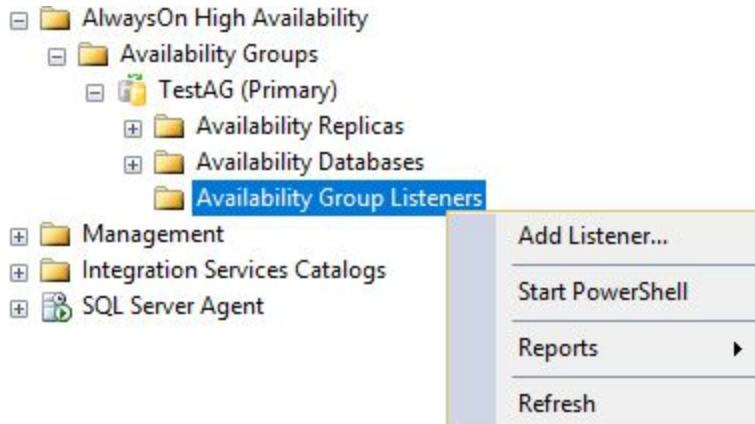
Summary:

Name	Result
Creating Login on replica 'BOU-SQL1'.	Success
Configuring endpoints.	Success
Starting the 'AlwaysOn_health' extended events session on 'BOU-SQL1'.	Success
Creating Login on replica 'BOU-SQL2'.	Success
Configuring endpoints.	Success
Starting the 'AlwaysOn_health' extended events session on 'BOU-SQL2'.	Success
Creating availability group 'TestAG'.	Success
Waiting for availability group 'TestAG' to come online.	Success
Joining secondaries to availability group 'TestAG'.	Success
Validating Windows Failover Cluster quorum vote configuration	Success
Creating a full backup for 'Test'.	Success
Restoring 'Test' on 'BOU-SQL2'.	Success
Backing up log for 'Test'.	Success
Restoring 'Test' log on 'BOU-SQL2'.	Success
Joining 'Test' to availability group 'TestAG' on 'BOU-SQL2'.	Success

< Previous      Next >      Close

After the AG is successfully created, click *Close*.

The listener can now be added. Expand the Availability Group and right click on *Availability Group Listeners*. Select *Add Listener...*



Give the listener a name. This will be the name that you use in connection strings, so keep that in mind when naming it. You can change it later if needed.

**Decision Point:** When you go to do this for real, you might consider using your old server's name as your listener name. For example, if you're migrating away from SQL2012B, then when you decommission that old junker, you could add a new listener with that name. Your applications wouldn't even know that their databases were in a fancy new Availability Group. However, if you do that, you won't be able to build & test the listener ahead of time.

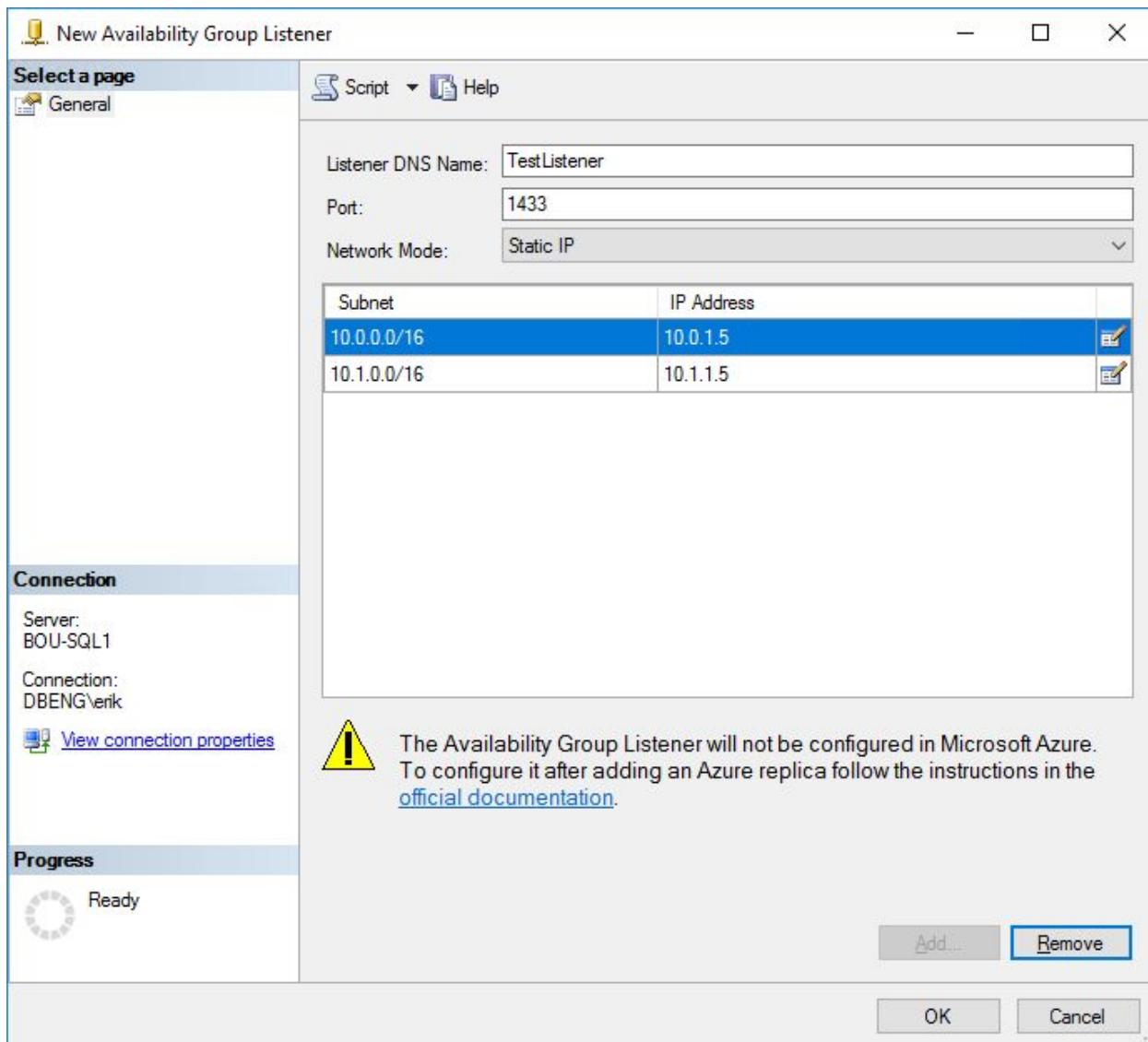
Instead, use a new name for the listener, but when it's time to decommission SQL2012B, turn that into a DNS CNAME. Rename the old SQL2012B to a different name (assuming that you have to keep the box around for a while for safekeeping), and then work with your network admins to create a new DNS CNAME for SQL2012B that points to your new listener name. This way, if someone forgets to change the connection string in their application, it'll still work.

To learn more about CNAMEs, check out this MSSQLTips post:

<https://www.mssqltips.com/sqlservertip/2663/using-friendly-names-for-sql-servers-via-dns/>

Now, back to our setup.

Use 1433 for *Port* and select *Static IP* for *Network Mode*. Add the listener IP addresses using the *Add* button.



Click OK to create the listener.

It's now time to test the AG. No, not the Attorney General, silly - vetting that AG is done by the United States Senate. You should know that, it's important.

# Testing Your Availability Group

Whenever you implement a high availability feature, you need to test 4 things:

1. **A planned failover with zero data loss** - like how you'll deal with patching, or a planned migration from one instance type to another
2. **An unplanned failover with zero data loss** - like when the primary replica goes offline
3. **An unplanned failover with data loss** - like when replication from the primary to the secondary falls behind, and then the primary fails
4. **A planned fallback with zero data loss** - after scenario #2 happens, and we bring the former primary back online, we need to re-establish synchronization and deal with the data differences between the replicas

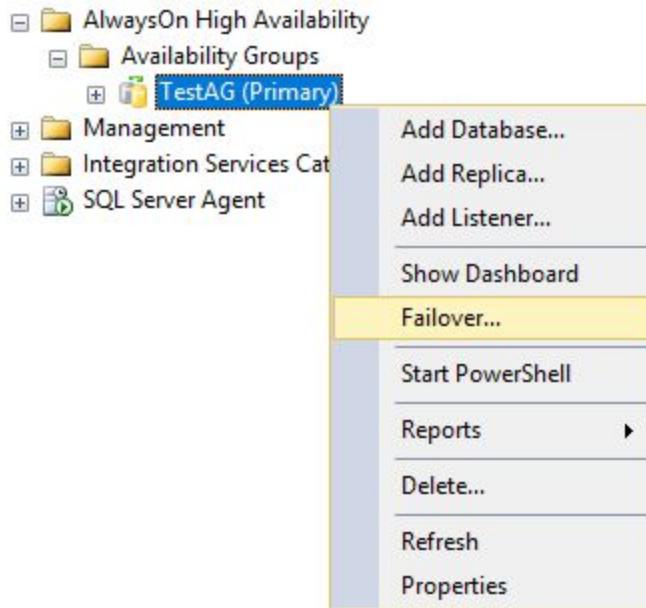
Let's tackle these one at a time.

## Test #1: Failing Over Manually - A planned failover with zero data loss

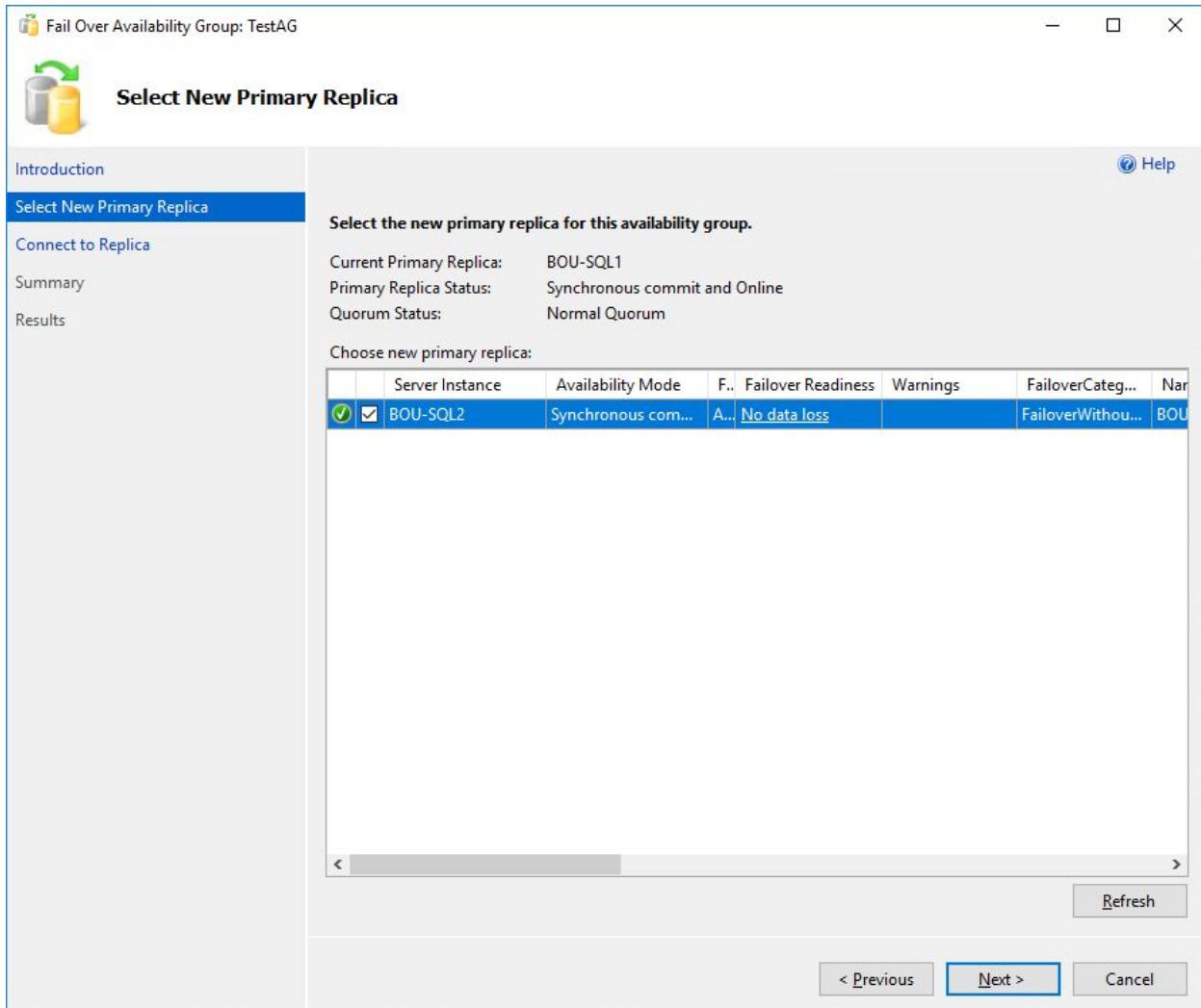
Never use *Failover Cluster Manager* to perform failovers for Availability Groups. It is not aware of the synchronization status of the replicas and can lead to an outage. Make sure all sysadmins are aware of this. It's especially diabolical because it *works*, in the sense that it doesn't throw an error, but that doesn't mean SQL Server won't complain when you fail over under duress.

To perform failovers, use the *Fail Over Availability Group* wizard or TSQL. You can initiate it from a primary replica or a secondary replica.

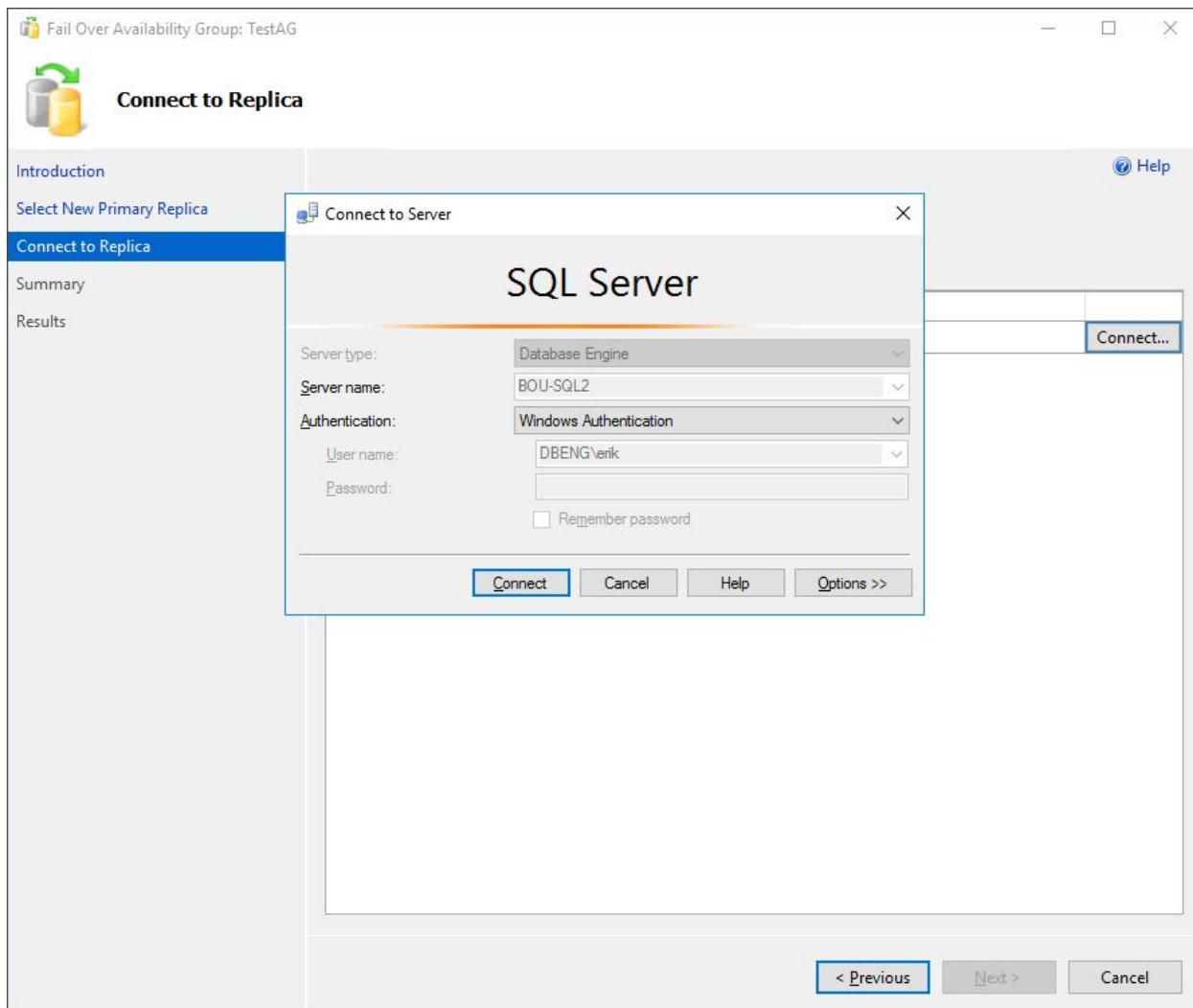
In SQL Server Management Studio, right click on the AG and select *Failover...*



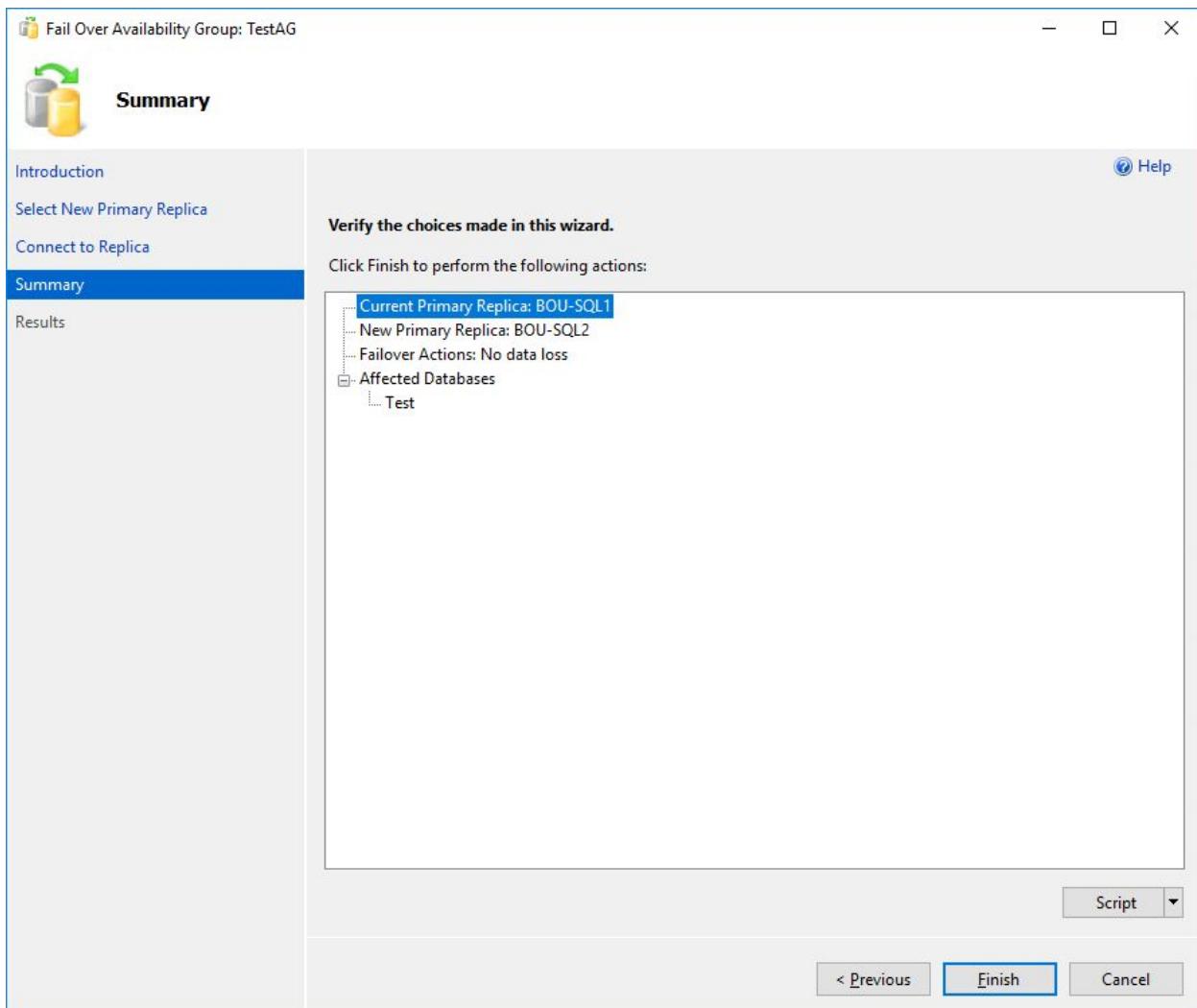
On the *Select New Primary Replica* page, verify that the secondary replica is selected for the server to failover to and then click *Next*.

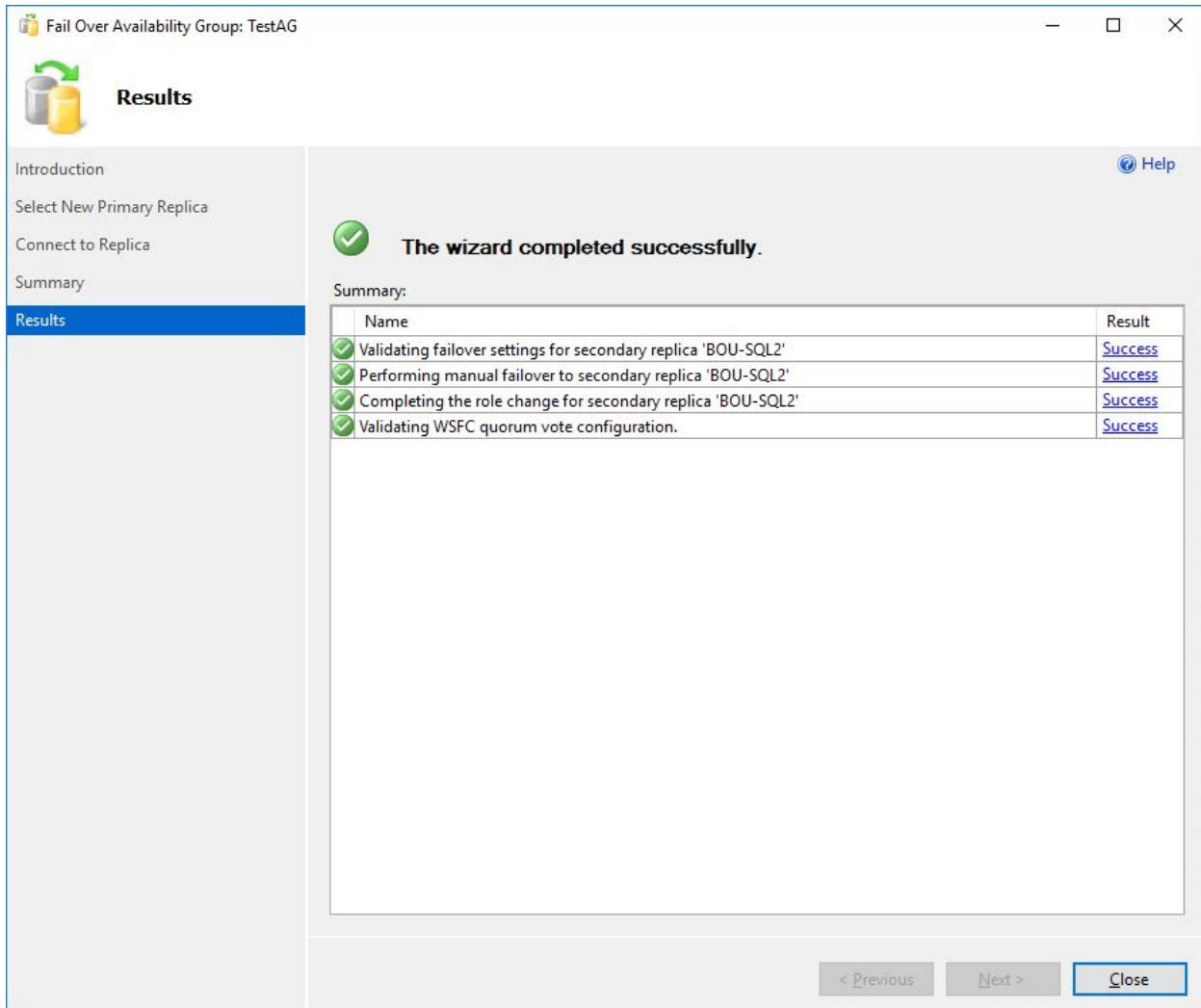


Connect to the secondary server using the *Connect...* button, click *Connect* and then *Next*.

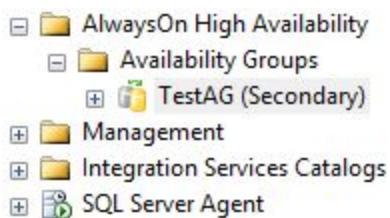


Click *Finish* to perform the failover and then *Close* when it finishes. You could click *Script* to script out the command, but I'll show that command next.





Management Studio will refresh and show the new replica role.

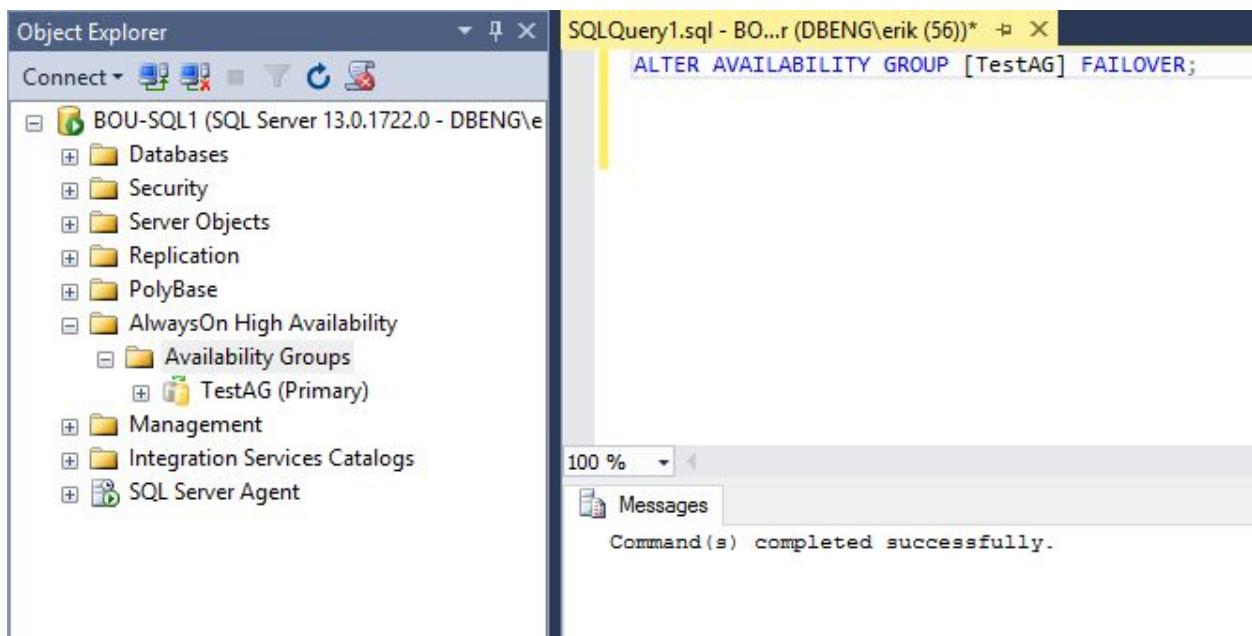


When there's a planned, graceful failover, SQL Server automatically starts synchronizing data over to the other replica (the former primary). You can fail over back and forth, no data loss, and the replicas are just magically kept in sync.

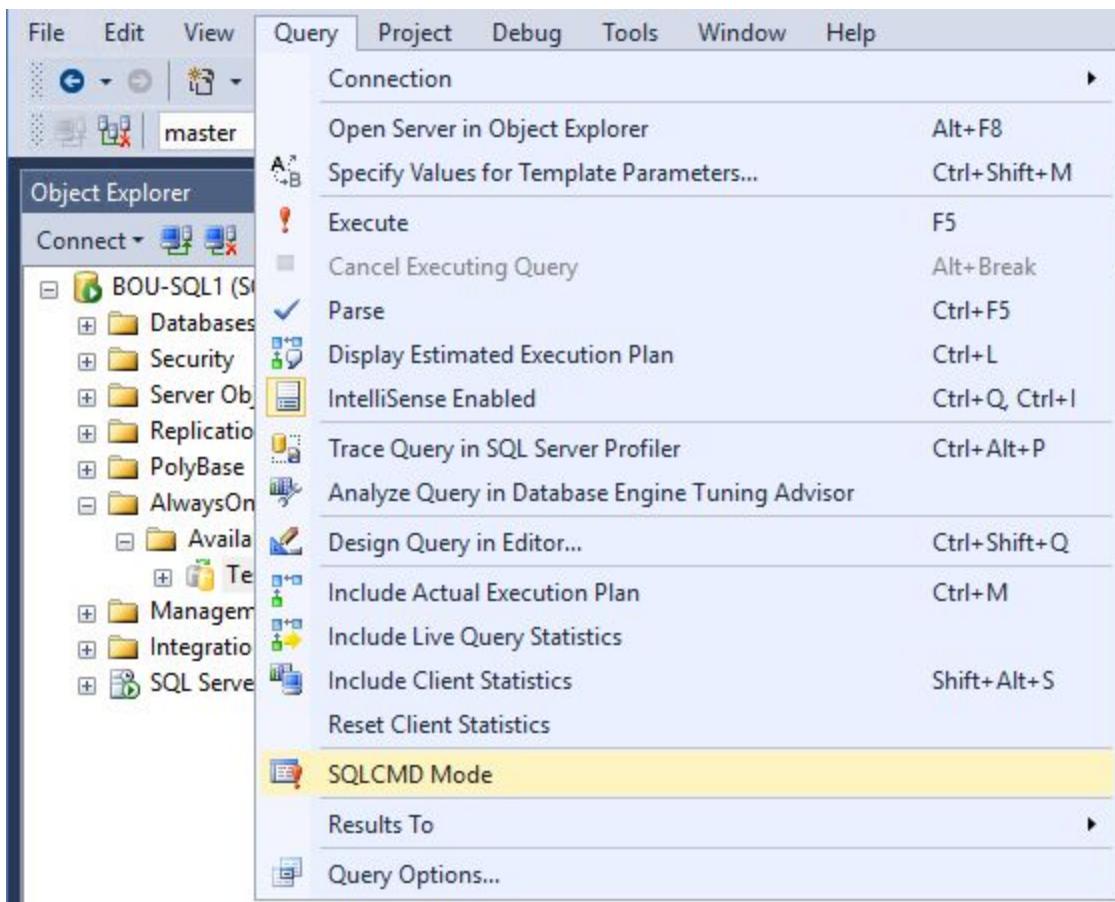
To perform the failover using a TSQL command, connect to the secondary replica that you want to failover to and then run the below command.

```
ALTER AVAILABILITY GROUP [TestAG] FAILOVER;
```

Refresh *Object Explorer* for the server you failed over to and see that the role has changed.



You could also use sqlcmd mode in Management Studio. I prefer this method when I don't already have a Query Editor window open for the server I'm failing over to. I'm lazy like that.



Use :CONNECT syntax to run the commands underneath it on another server.

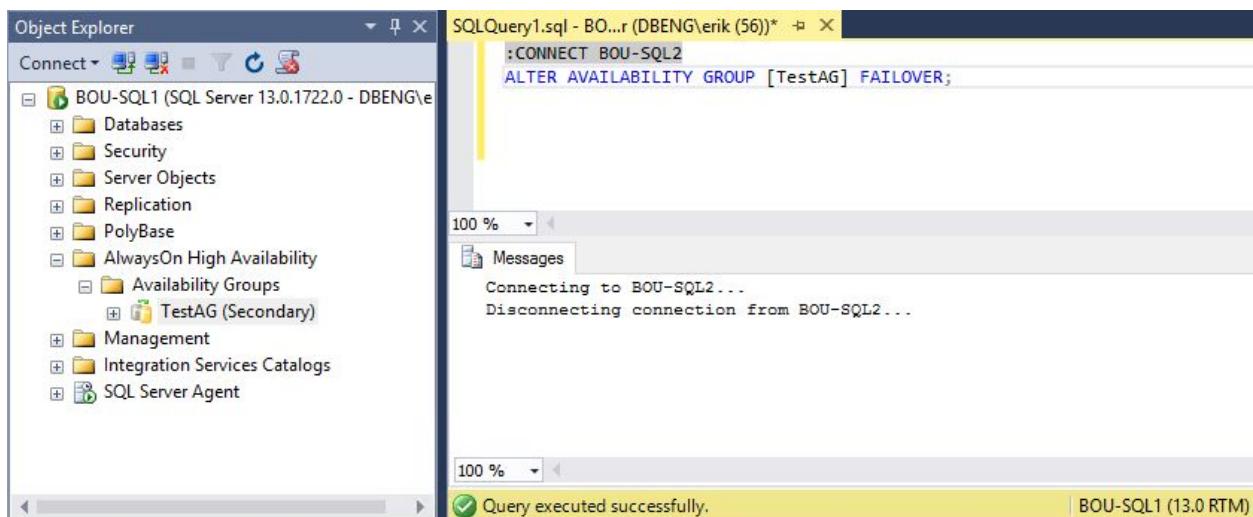
The screenshot shows the SSMS Query Editor window with the following content:

```
SQLQuery1.sql - BOU-SQL1 (DBENG\erik (56)) * → X
:CONNECT BOU-SQL2
ALTER AVAILABILITY GROUP [TestAG] FAILOVER;
```

The status bar at the bottom indicates "Query executed successfully." and "BOU-SQL1 (13.0 RTM) | DBENG\erik (56) | master | 00:00:05 | 0 rows".

Notice that the Query Editor window is connected to BOU-SQL1 but that the failover command will run on BOU-SQL2.

Execute both commands and then refresh *Object Explorer*.



Remember that all sysadmins should know not to use Failover Cluster Manager to perform an AG failover and instead know how to do them in Management Studio. This is really important when an organization has a DBA team and a different team that performs reboots such as for Microsoft security patching. Server admins with clustering experience know to failover SQL Server instances off the server that they are going to reboot, so be sure that they know how to do an AG failover.

## Test #2: Failing Over Automatically - An unplanned failover with zero data loss

Initiate an automatic failover by stopping the SQL Server service on the primary replica.

Wait a few seconds, typically takes 15-30 seconds, and then check *Object Explorer* for the new primary replica. Assuming everything went well, the AG failed over with no data loss since the replica it failed over to is configured for synchronous-commit with automatic failovers.

In this scenario, our failover wasn't planned - but as long as the two replicas were in synchronous commit mode, no data is lost. When the stopped SQL Server service is started back up again, it will automatically start fetching data from the new primary in a matter of minutes, and catch back up to sync and allow automatic failovers.

However, while one of the two replicas is down, you have no safety net.

Folks often design a two-node AG with synchronous commits and say, "We're aiming for zero data loss." That's only true as long as both replicas are up. SQL Server doesn't automatically go down with a minimum number of replicas - your users keep right on inserting data into the one remaining replica, thinking that their data is safe, when it's not. You have a single point of failure at this point.

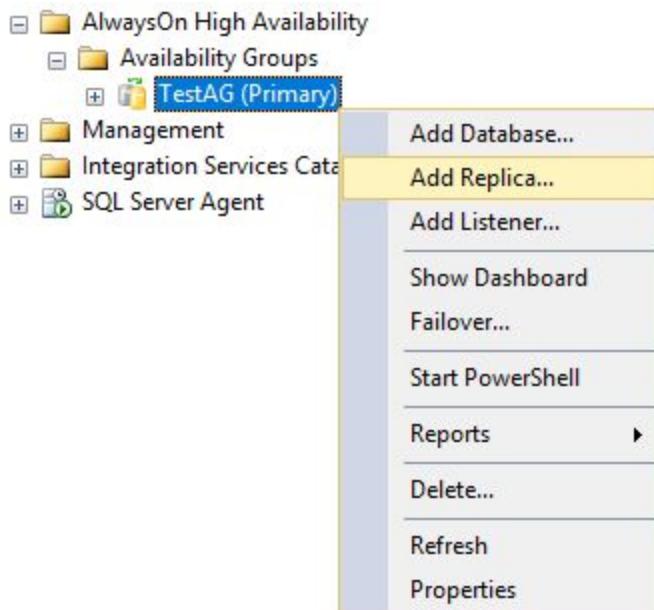
That's why monitoring is so important. By default, SQL Server doesn't tell you that you're down to a single point of failure, or that a replica has stopped synchronizing with other replicas.

Speaking of which, let's hit the third test.

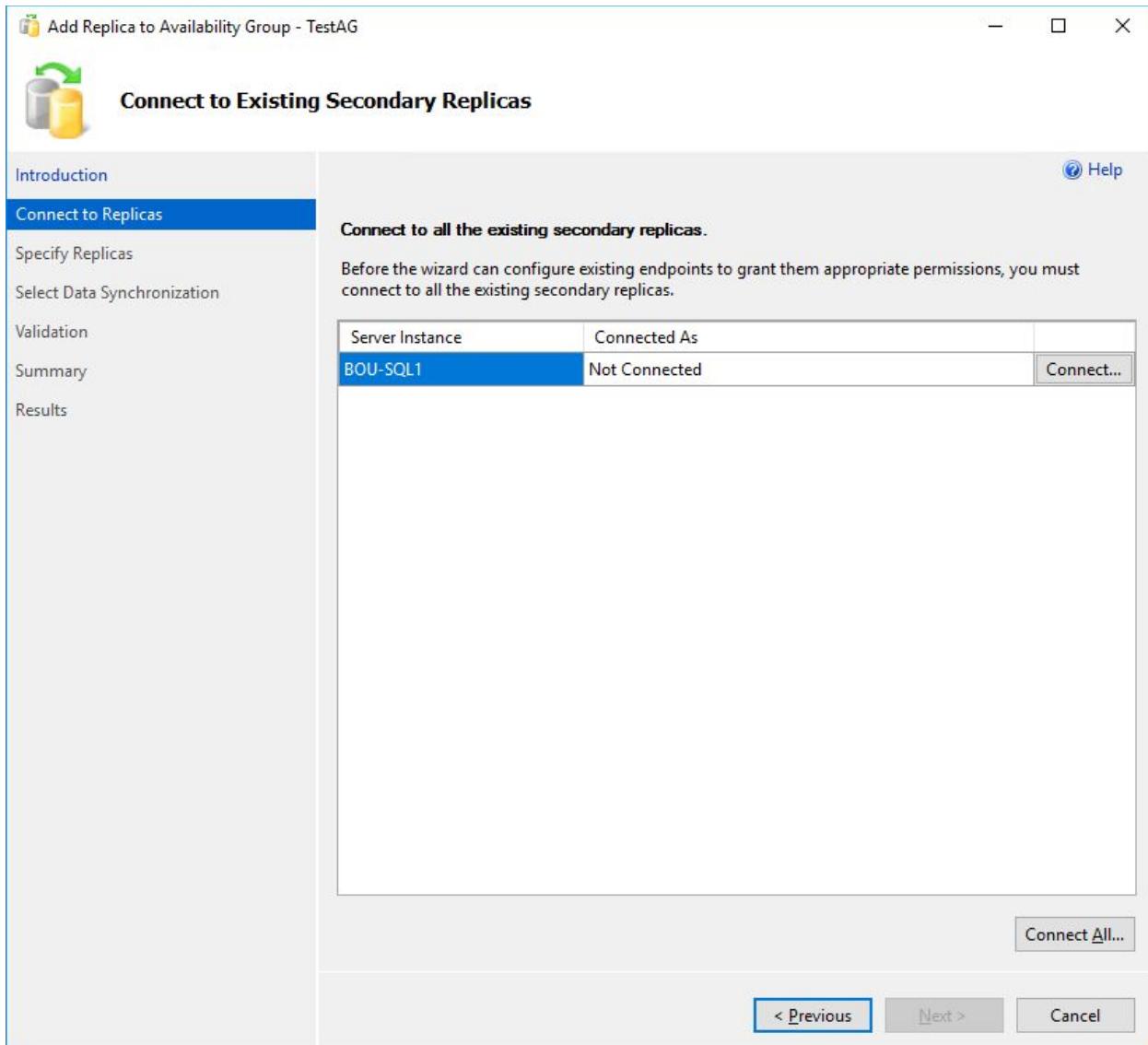
## Test #3: Failing Over Manually with Data Loss - An unplanned failover with data loss

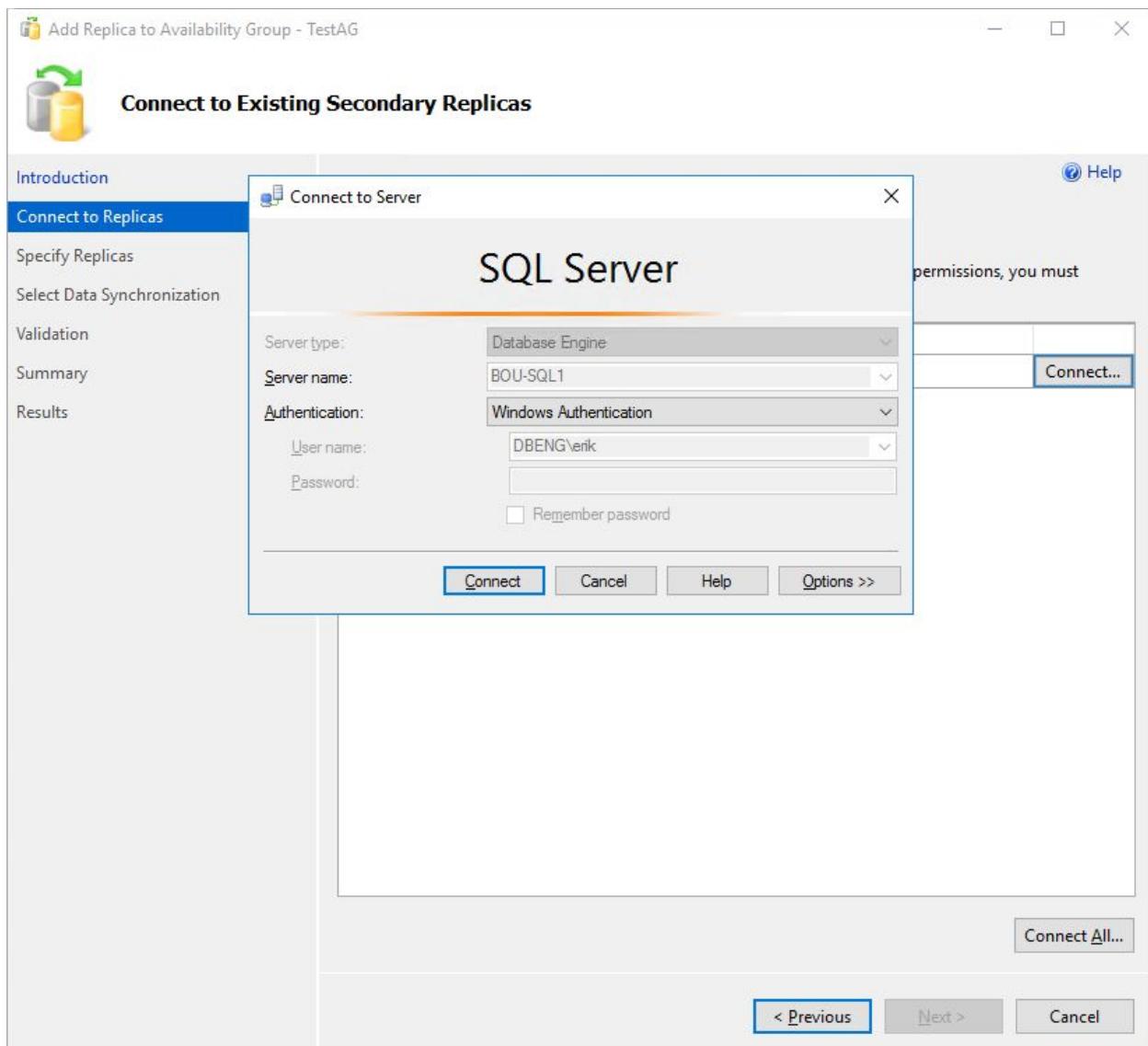
We could use the 2 servers to test out a manual failover with data loss by switching server2 to be asynchronous-commit and then stopping SQL Server on server1, but let's take it one step further and add a third asynchronous server to the mix.

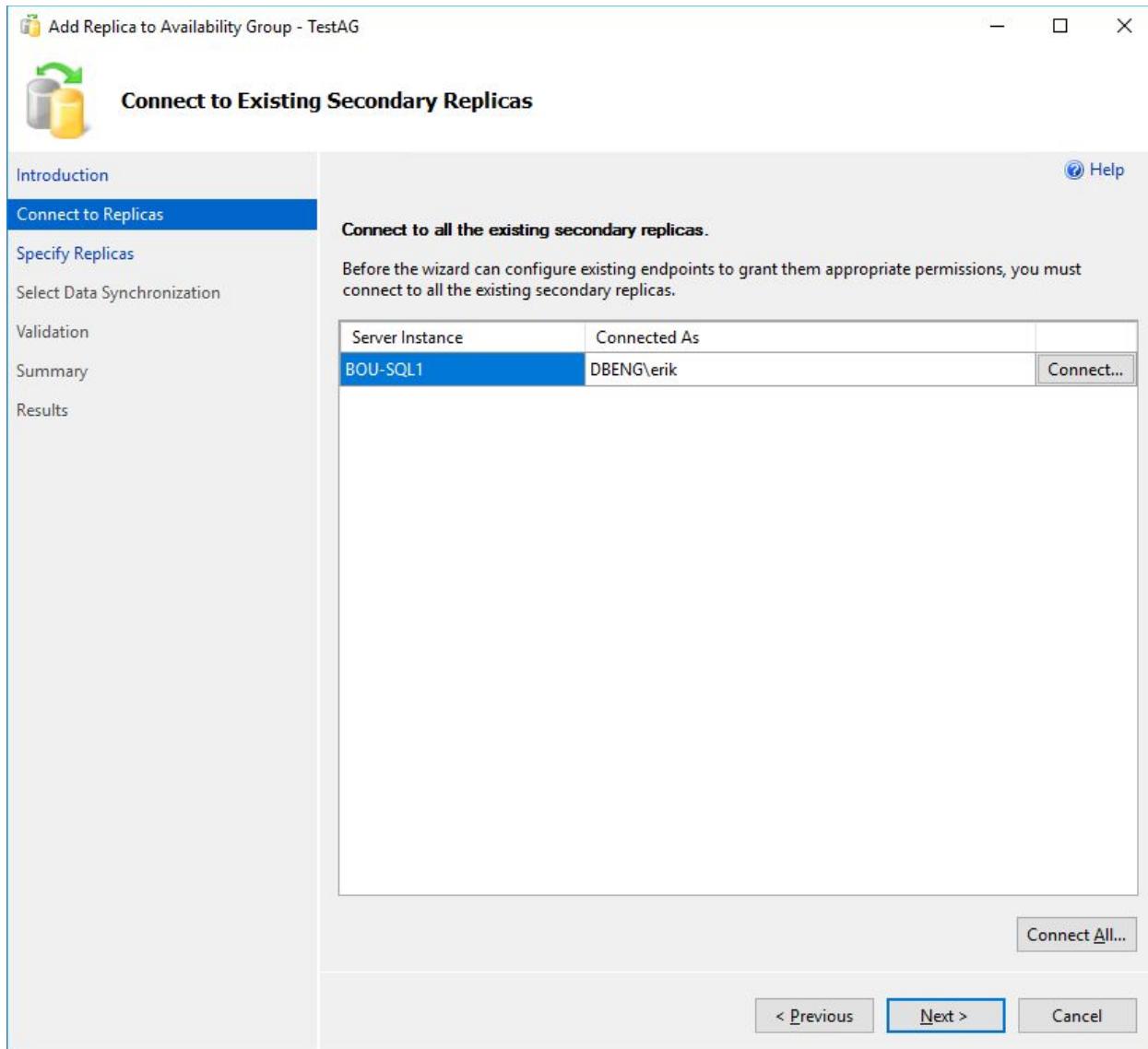
Create a 3rd VM, join it to the domain, add it to the cluster (use *Add Node* in Failover Cluster Manager), enable the AG feature and then add it to the AG by right clicking on the AG and then clicking *Add Replica...*



On the *Connect to Replicas* page, connect to the secondary replica. For me, that's now BOU-SQL1 as BOU-SQL2 is primary. Click *Connect...*, *Connect* and then *Next*.







On the *Specify Replicas* page, click *Add Replica...*, enter the 3rd server and then click *Connect*.

Add Replica to Availability Group - TestAG

## Specify Replicas

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

Help

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

Replicas Endpoints Backup Preferences Listener

Availability Replicas:

Server Instance	Initial Role	Automatic Failover (Up to 3)	Synchronous Commit (Up to 3)	Readable Secondary
BOU-SQL2	Primary	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Yes
BOU-SQL1	Secondary	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Yes

< >

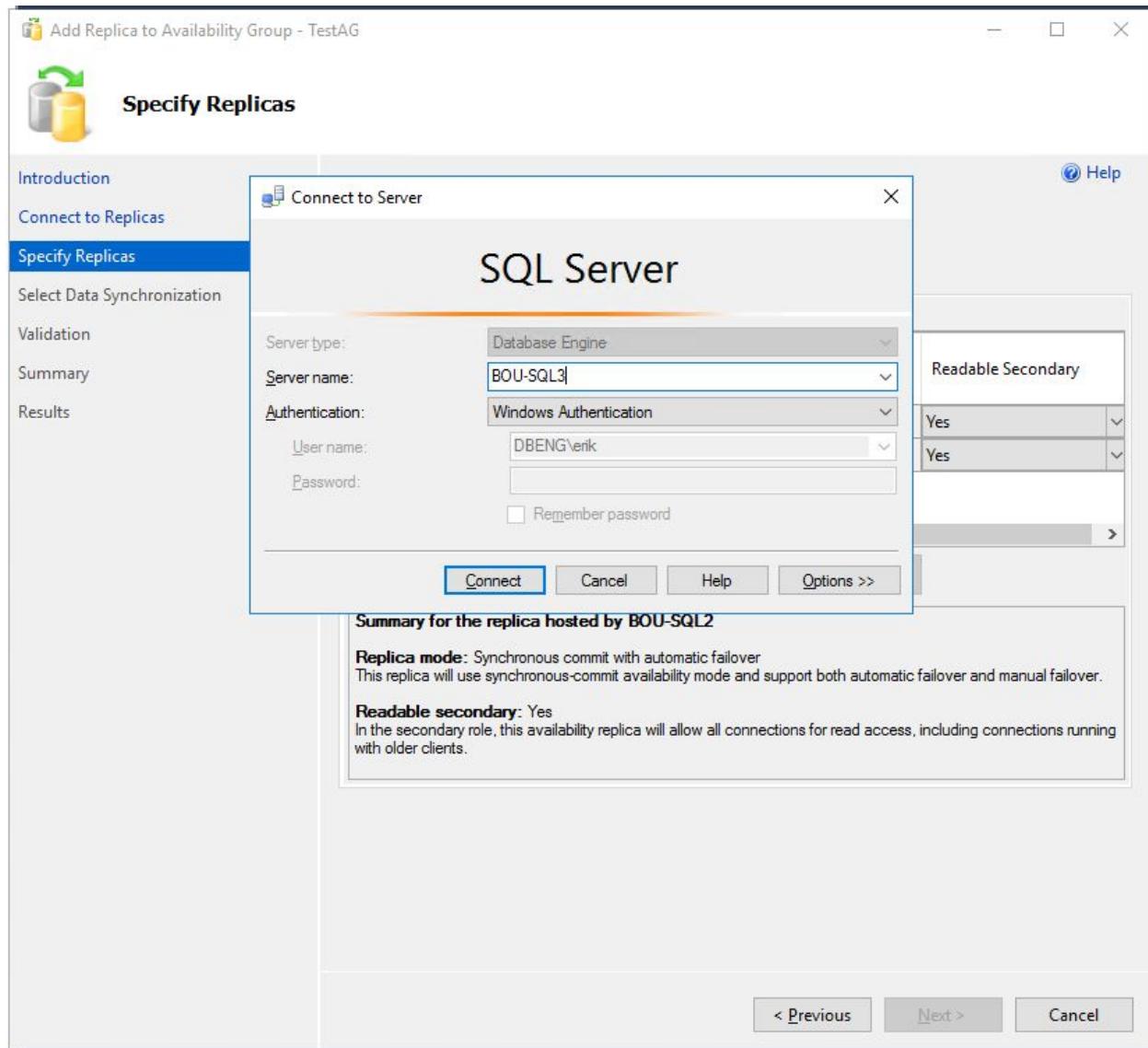
Add Replica... Add Azure Replica... Remove Replica

**Summary for the replica hosted by BOU-SQL2**

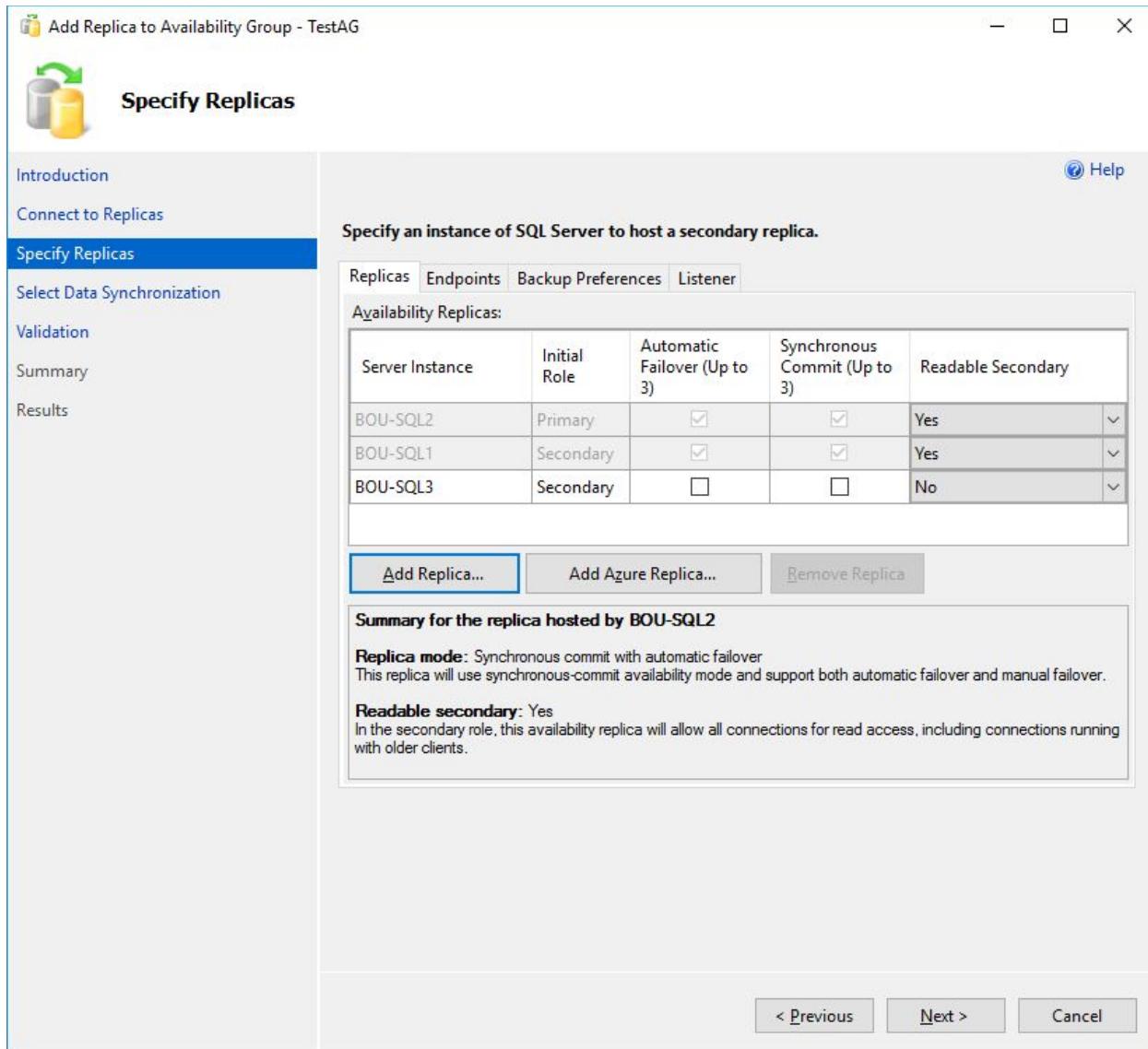
**Replica mode:** Synchronous commit with automatic failover  
This replica will use synchronous-commit availability mode and support both automatic failover and manual failover.

**Readable secondary:** Yes  
In the secondary role, this availability replica will allow all connections for read access, including connections running with older clients.

< Previous Next > Cancel



Configure it as asynchronous-commit with manual failovers. There isn't an option to use automatic failovers with asynchronous-commit. Someone has to initiate the failover if there could be data loss, because Availability Groups will never automatically fail over in a way that data will be lost. Availability Groups: watching out for you since 2012. (Well, that's not technically true - they tried to kill us several times in 2012. But they're better now.)



Decide how you'd like to synchronize the data. The Test database that I'm using is empty, so a FULL backup is easy. Point it to the backup share if it's not already filled in and then click *Next*.

Add Replica to Availability Group - TestAG

## Select Initial Data Synchronization

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

Select your data synchronization preference.

**Full**  
Starts data synchronization by performing full database and log backups for each selected database. These databases are restored to each secondary and joined to the availability group.

Specify a shared network location accessible by all replicas:

**Join only**  
Starts data synchronization where you have already restored database and log backups to each secondary server. The selected databases are joined to the availability group on each secondary. This action will be skipped for Azure replicas.

**Skip initial data synchronization**  
Choose this option if you want to perform your own database and log backups of each primary database.

< Previous  Cancel

Add Replica to Availability Group - TestAG

**Validation**

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

Help

**Results of availability group validation.**

Name	Result
Checking whether the endpoint is encrypted using a compatible algorithm	Success
Checking shared network location	Success
Checking replica availability mode	Success
Checking for free disk space on the server instance that hosts secondary replica BOU-SQL3	Success
AddReplicaDatabaseExistenceValidator	Success
AddReplicaDatabaseFileCompatibilityValidator	Success
AddReplicaDatabaseFileExistenceValidator	Success
⚠️ Checking the listener configuration	Warning

Re-run Validation

< Previous    **Next >**    Cancel

If you get an error for the *AddReplicaDatabaseFileCompatibilityValidator* check on the *Validation* page like I did, create the folders where the database files reside on the 3rd server. My database uses the SQLData and SQLLog folders on the C: drive. I had to create those on the 3rd server in order to pass this validation check. It makes sense to get an error since a RESTORE command will not create folders for you. Once the folders have been created, click *Re-run Validation*. It should pass the check now.

Add Replica to Availability Group - TestAG

**Validation**

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

Help

**Results of availability group validation.**

Name	Result
Checking whether the endpoint is encrypted using a compatible algorithm	Success
Checking shared network location	Success
Checking replica availability mode	Success
Checking for free disk space on the server instance that hosts secondary replica BOU-SQL3	Success
AddReplicaDatabaseExistenceValidator	Success
AddReplicaDatabaseFileCompatibilityValidator	Error
AddReplicaDatabaseFileExistenceValidator	Success
Checking the listener configuration	Warning

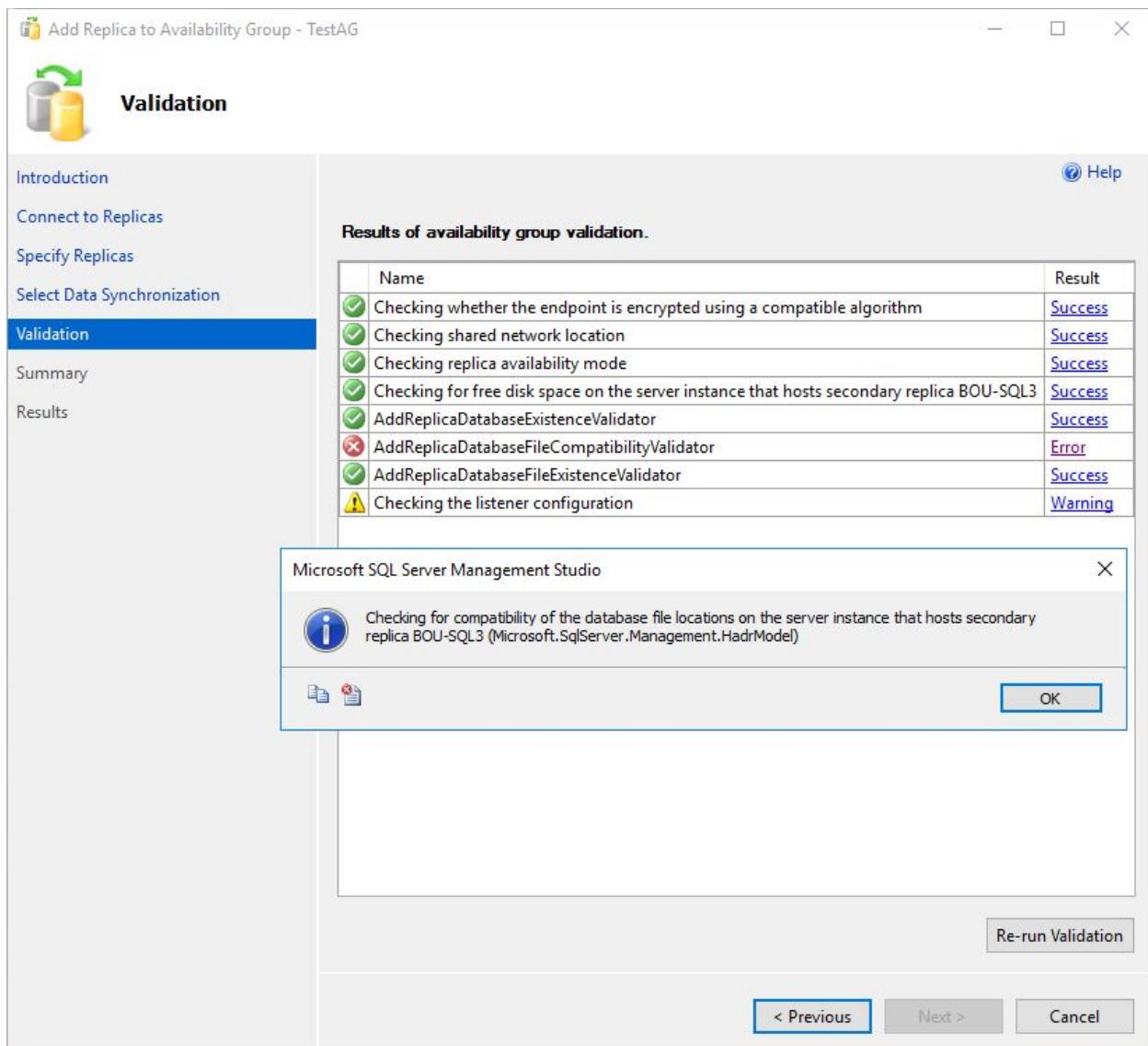
Microsoft SQL Server Management Studio

Checking for compatibility of the database file locations on the server instance that hosts secondary replica BOU-SQL3 (Microsoft.SqlServer.Management.HadrModel)

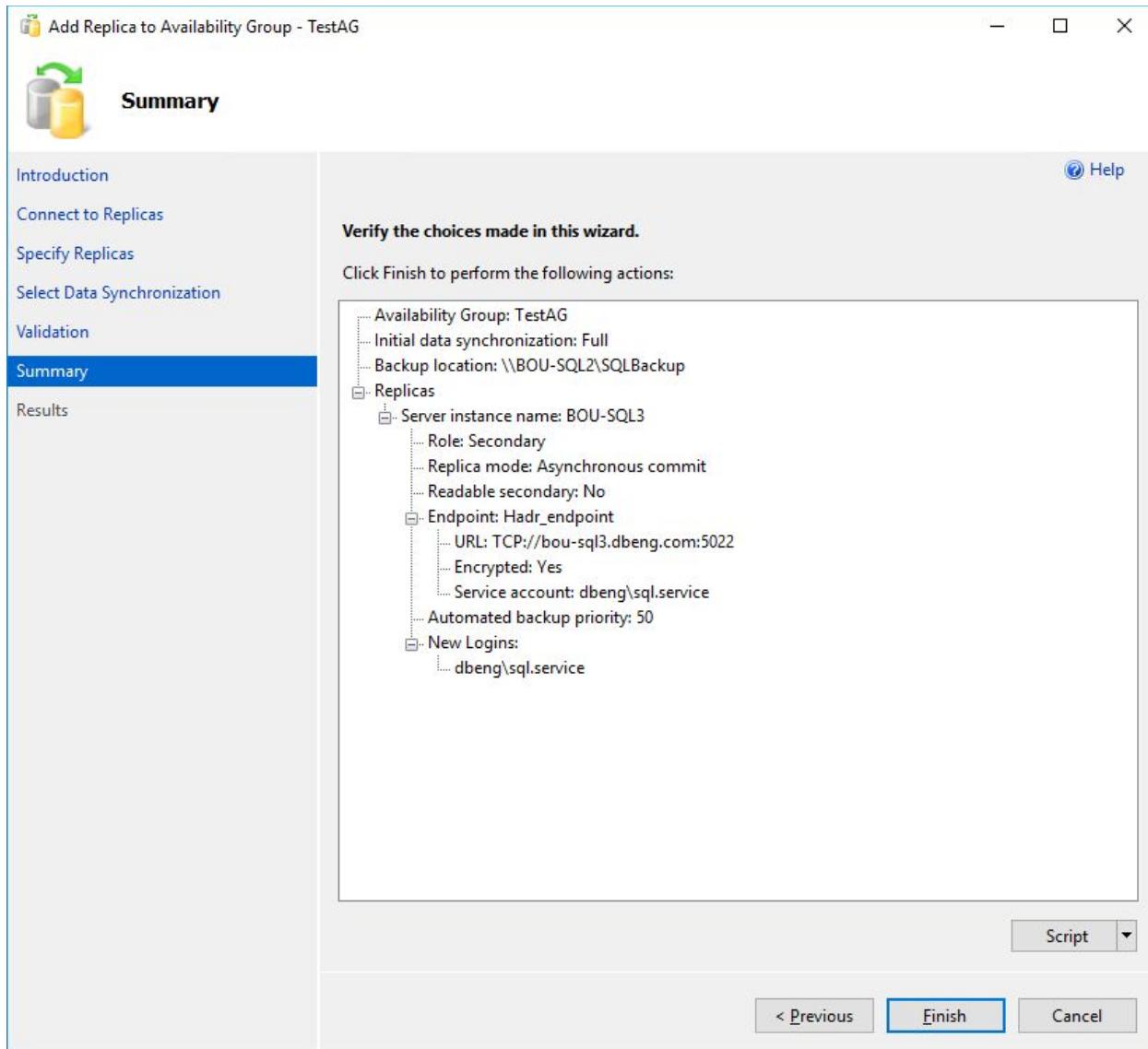
OK

Re-run Validation

< Previous Next > Cancel



Click *Next* and then *Finish*.



Click *Close* when it is done.

Add Replica to Availability Group - TestAG

 Results

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

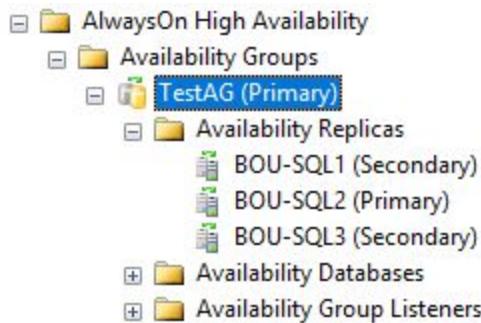
The wizard completed successfully.

Summary:

Name	Result
Configuring endpoints.	Success
Configuring endpoints.	Success
Creating Login on replica 'BOU-SQL3'.	Success
Configuring endpoints.	Success
Starting the 'AlwaysOn_health' extended events session on 'BOU-SQL3'.	Success
Adding replica to availability group 'TestAG'.	Success
Joining secondaries to availability group 'TestAG'.	Success
Creating a full backup for 'Test'.	Success
Restoring 'Test' on 'BOU-SQL3'.	Success
Backing up log for 'Test'.	Success
Restoring 'Test' log on 'BOU-SQL3'.	Success
Joining 'Test' to availability group 'TestAG' on 'BOU-SQL3'.	Success

< Previous      Next >      Close

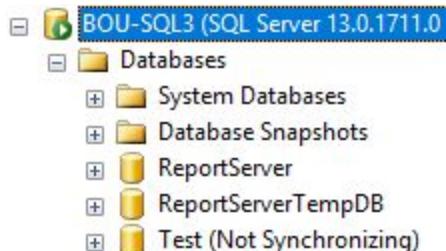
Refresh *Object Explorer* and expand the AG to see that the 3rd replica has been added.



If you encountered any errors along the way for this 3rd VM, go back in the doc and make sure you've ran all the necessary commands. I missed a few steps, such as the Powershell command to add the cluster tools and opening up ports 1433 and 5022, which caused errors at various stages. Run the same commands with necessary changes and do the same steps as was done for the other two VMs. Yours probably passed the first time, though. You're lucky. Not to mention attractive. I like the way you're holding this white paper.

Stop the SQL Server service on both synchronous-commit replicas (the first two VMs). This partially simulates a region or zone failure where all of your synchronous replicas have gone down. (It isn't a perfect simulation, because Windows is still up, but we're making this test a little easier for you.)

Connect to the 3rd VM in SSMS. Notice that the database is in a "Not Synchronizing" state because it's unable to reach the other two replicas.



Run the ALTER AVAILABILITY GROUP command to fail the AG over to the 3rd server, but use the option to allow data loss. This replica is asynchronous and might not be completely up to date with all of the data changes that occurred on the primary replica when both synchronous replicas went down. Failing over to it simulates what would happen in the case of a disaster where you lose the primary data center and need to get production online at the Disaster Recovery site.

```
ALTER AVAILABILITY GROUP TestAG FORCE_FAILOVER_ALLOW_DATA_LOSS;
```

The screenshot shows the SSMS interface. In the top window, a query is being run:

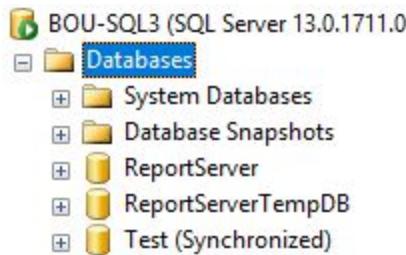
```
ALTER AVAILABILITY GROUP TestAG FORCE_FAILOVER_ALLOW_DATA_LOSS;
```

The bottom window, titled "Messages", displays the result:

Command(s) completed successfully.

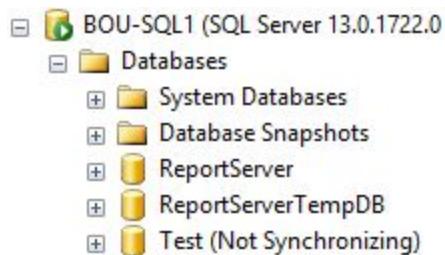
In the status bar at the bottom, there is a green checkmark icon followed by the text "Query executed successfully.", and the server information "BOU-SQL3 (13.0 RTM) | DBENG\erik (52) | master | 00:00:05 | 0 rows".

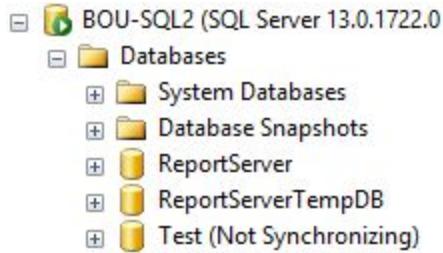
Refresh *Object Explorer* for the 3rd server.



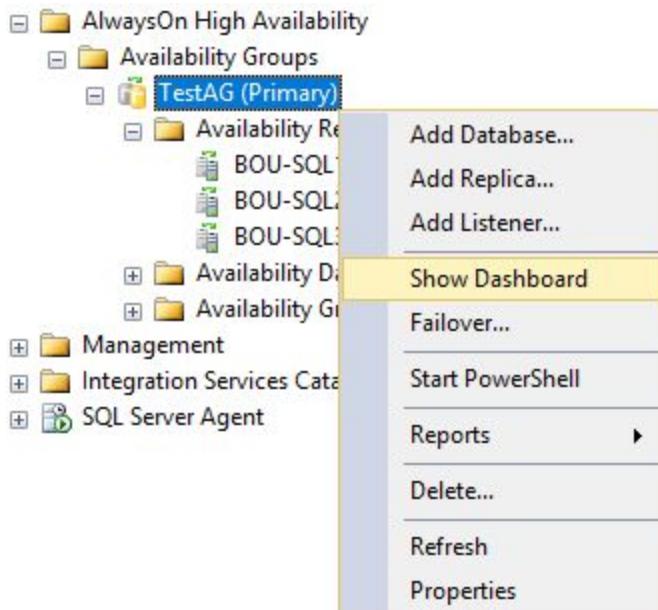
Start SQL Server on the first 2 VMs.

Refresh *Object Explorer* for the first 2 servers.





Check the Always On Dashboard to confirm that the 3rd server is in a good state, whereas the first 2 servers are not. Right-click on the AG and select *Show Dashboard*.



TestAG:BOU-SQL3 X

! TestAG: hosted by BOU-SQL3 (Replica role: Primary)

Availability group state:	<span style="color: yellow;">!</span> <a href="#">Warning --- Warnings (2)</a>
Primary instance:	BOU-SQL3
Failover mode:	Manual
Cluster state:	bou-cluster1 (Normal Quorum)

Availability replica:

Name	Role	Failover Mode	Synchronization State	Issues
<span style="color: yellow;">!</span> <a href="#">BOU-SQL1</a>	Secon...	Automatic	Not Synchronizing	<a href="#">Warnings (1)</a>
<span style="color: yellow;">!</span> <a href="#">BOU-SQL2</a>	Secon...	Automatic	Not Synchronizing	<a href="#">Warnings (1)</a>
<span style="color: green;">✓</span> <a href="#">BOU-SQL3</a>	Primary	Manual	Synchronized	

Group by ▾

Name	Replica	Synchronization State	Failover Readi...	Issues
BOU-SQL1				
<span style="color: yellow;">!</span> Test	BOU-SQL1	Not Synchronizing	Data Loss	<a href="#">Warnings (2)</a>
BOU-SQL2				
<span style="color: yellow;">!</span> Test	BOU-SQL2	Not Synchronizing	Data Loss	<a href="#">Warnings (2)</a>
BOU-SQL3				
<span style="color: green;">✓</span> Test	BOU-SQL3	Synchronized	No Data Loss	

The “Not Synchronizing” state for BOU-SQL1 and BOU-SQL2 isn’t temporary: once you perform a manual failover with data loss, SQL Server knows that there’s data on SQL1 and SQL2 that you might want to recover.

Books Online has a great checklist for tasks you need to perform after a forced failover:  
<https://msdn.microsoft.com/en-us/library/ff877957.aspx#FollowUp>

It talks about fixing cluster quorum, creating database snapshots in case you want to recover the lost data from the former primaries (1 & 2), enabling synchronization from the new primary, and what might happen if the offline nodes restart with their own quorum. Seriously, it’s important if you plan on implementing a multi-subnet AG.

## Test #4: Failing Back Manually - A planned failback with zero data loss

There are at least 2 ways to get the first 2 replicas (BOU-SQL1 & 2) back to a healthy state without rebuilding the AG:

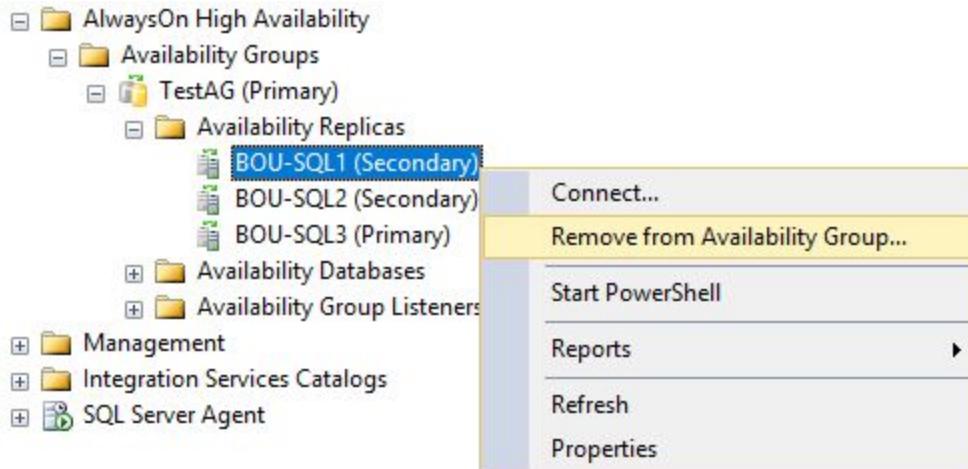
1. Remove the first 2 servers from the AG, restore the LOG backup chain and then add the first 2 servers back to the AG
2. Remove the first 2 servers from the AG, restore a FULL backup and the LOG backup chain, and then add the first 2 servers back to the AG

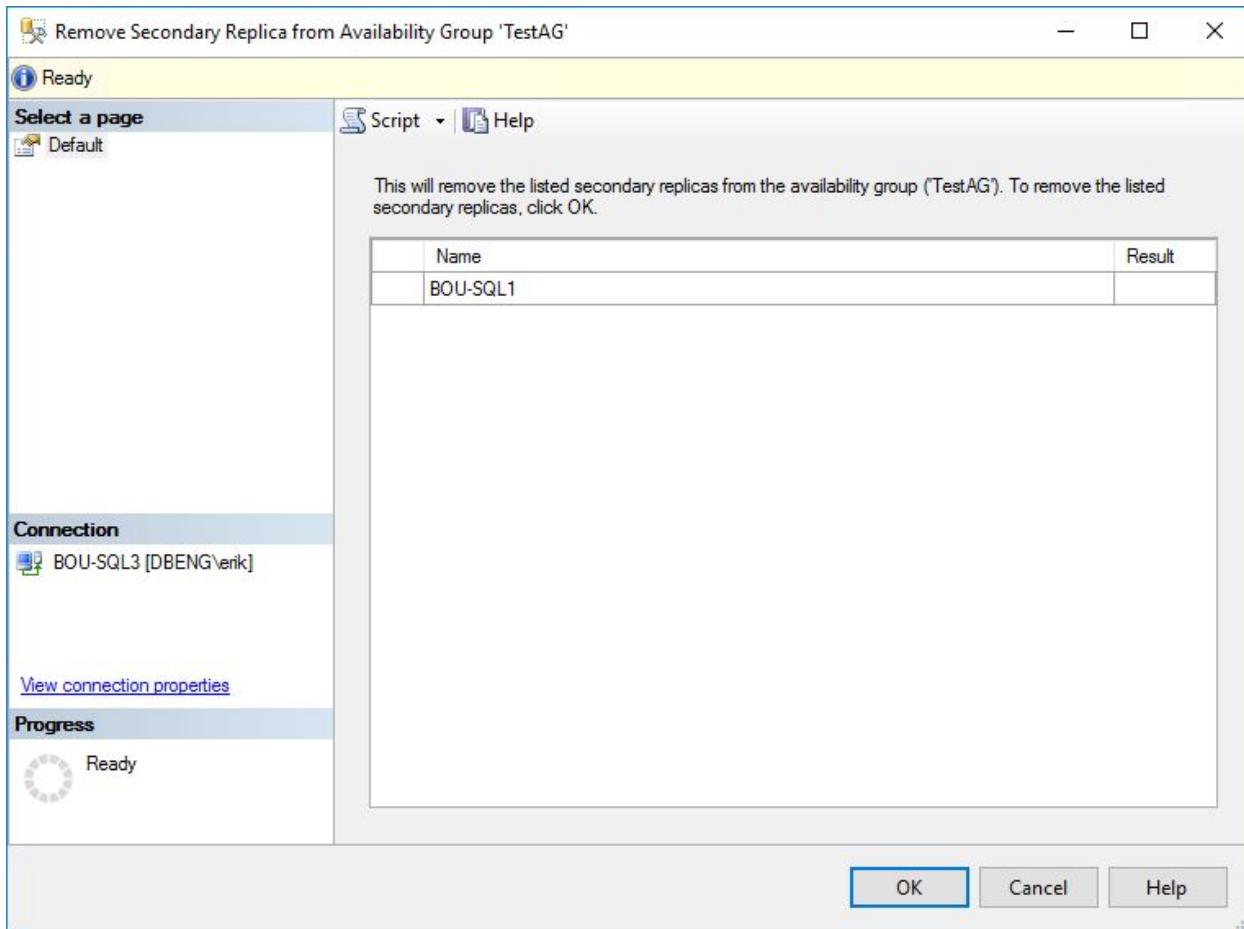
There can be more depending on factors like how little behind the AG was at the time of the failover, how small the databases are, how fast the network is between sites, etc. (This is where 2016's Direct Seeding can come in handy, but you have to be careful with large databases flooding your network connection unchecked.)

Let's use Option 1 for the 1st server and Option 2 for the 2nd server.

Remove the first 2 servers from the AG using the UI or the TSQL command.

UI:



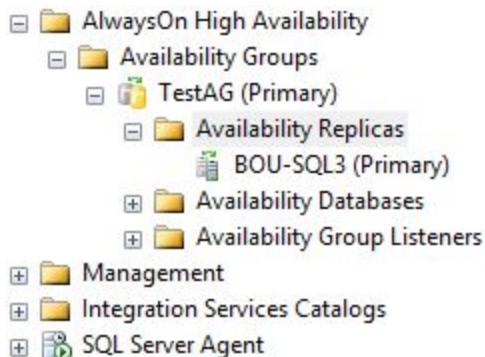


TSQL:

On the 3rd server, use the REMOVE REPLICA option of the ALTER AVAILABILITY GROUP command to remove a replica.

```
ALTER AVAILABILITY GROUP [TestAG] REMOVE REPLICA ON N'BOU-SQL2';
```

Refresh *Object Explorer* on the 3rd server to confirm that only the 3rd replica is in the AG.



Take a LOG backup on the 3rd server and then restore it on the first server. If any other LOG backups have run on the 3rd server since the first 2 servers have been offline, restore those LOG backups first.

Use the WITH NORECOVERY option when restoring the LOG backups.

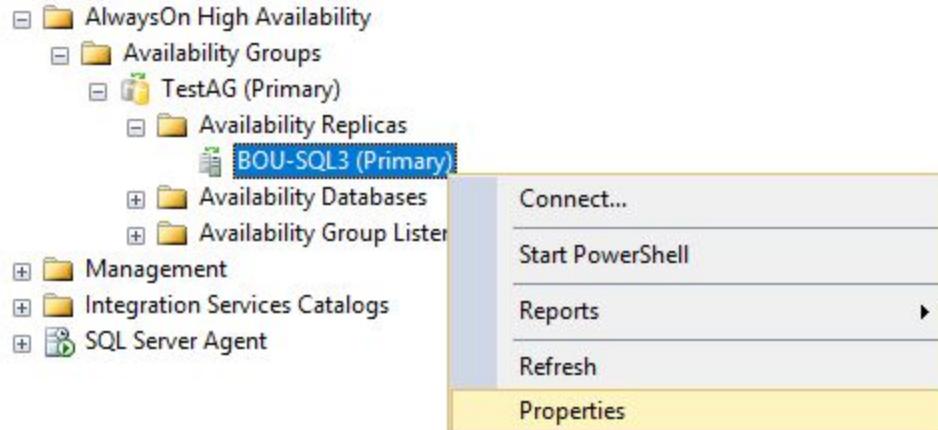
3rd server:

```
BACKUP LOG Test  
TO DISK = 'C:\SQLBackup\Test_20170909_1022.trn'  
WITH INIT;
```

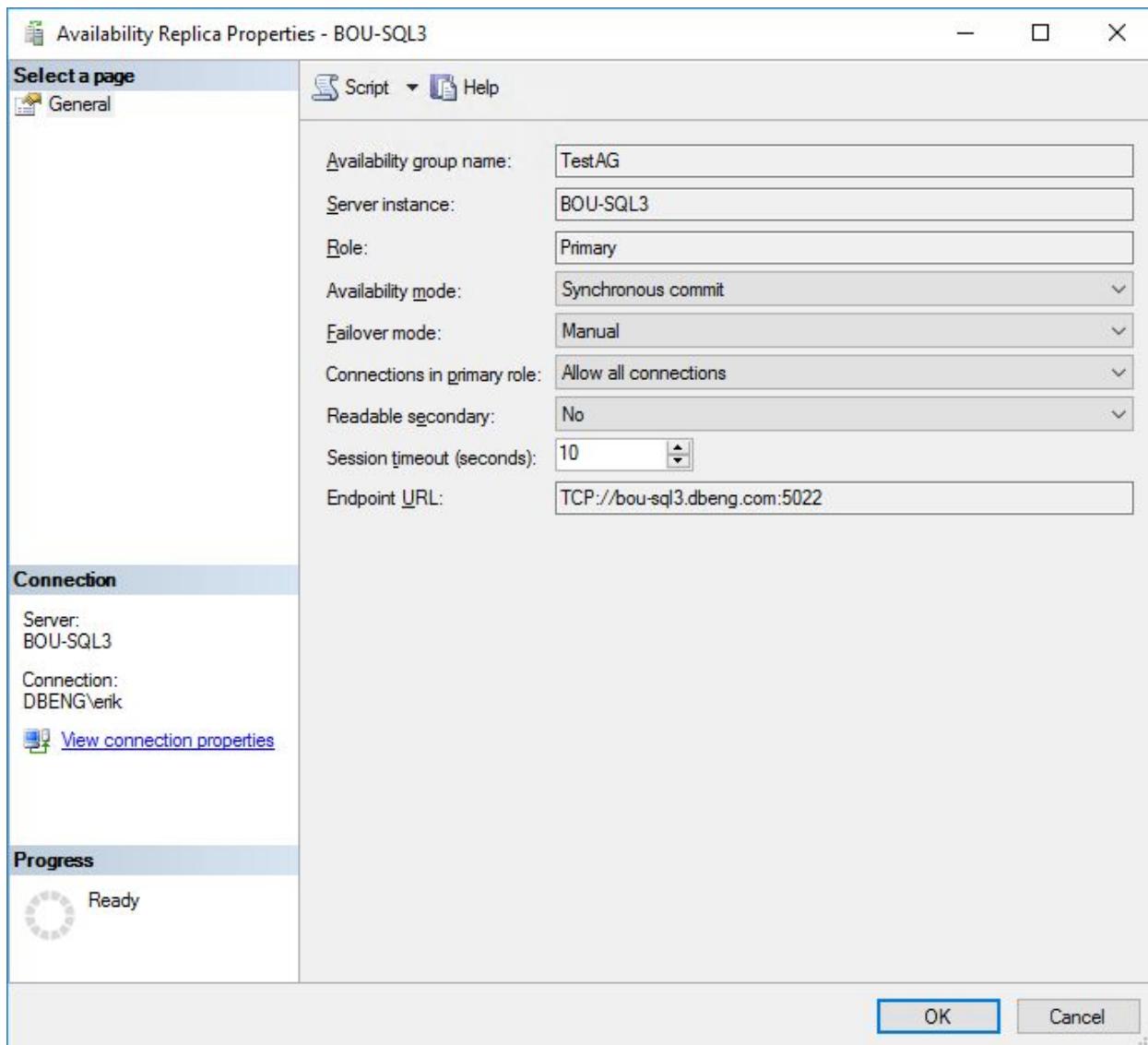
1st server (copy trn file to the 1st server or use a UNC path to reach the file on the 3rd server):

```
RESTORE LOG Test  
FROM DISK = 'C:\SQLBackup\Test_20170909_1022.trn'  
WITH NORECOVERY;
```

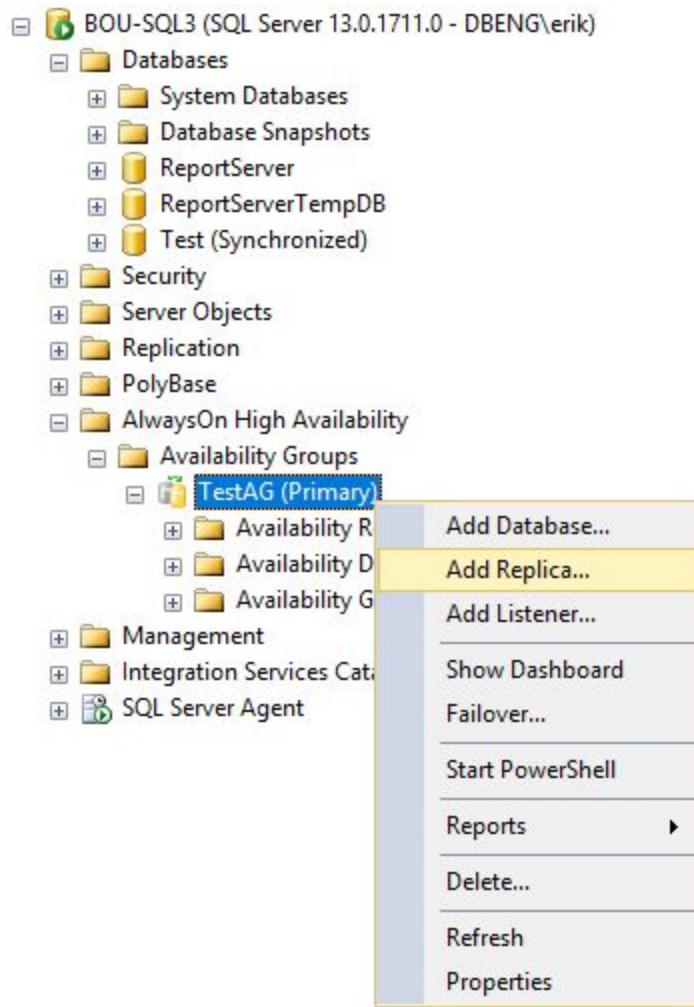
Set the 3rd server to synchronous-commit by right clicking on the replica and selecting *Properties*.



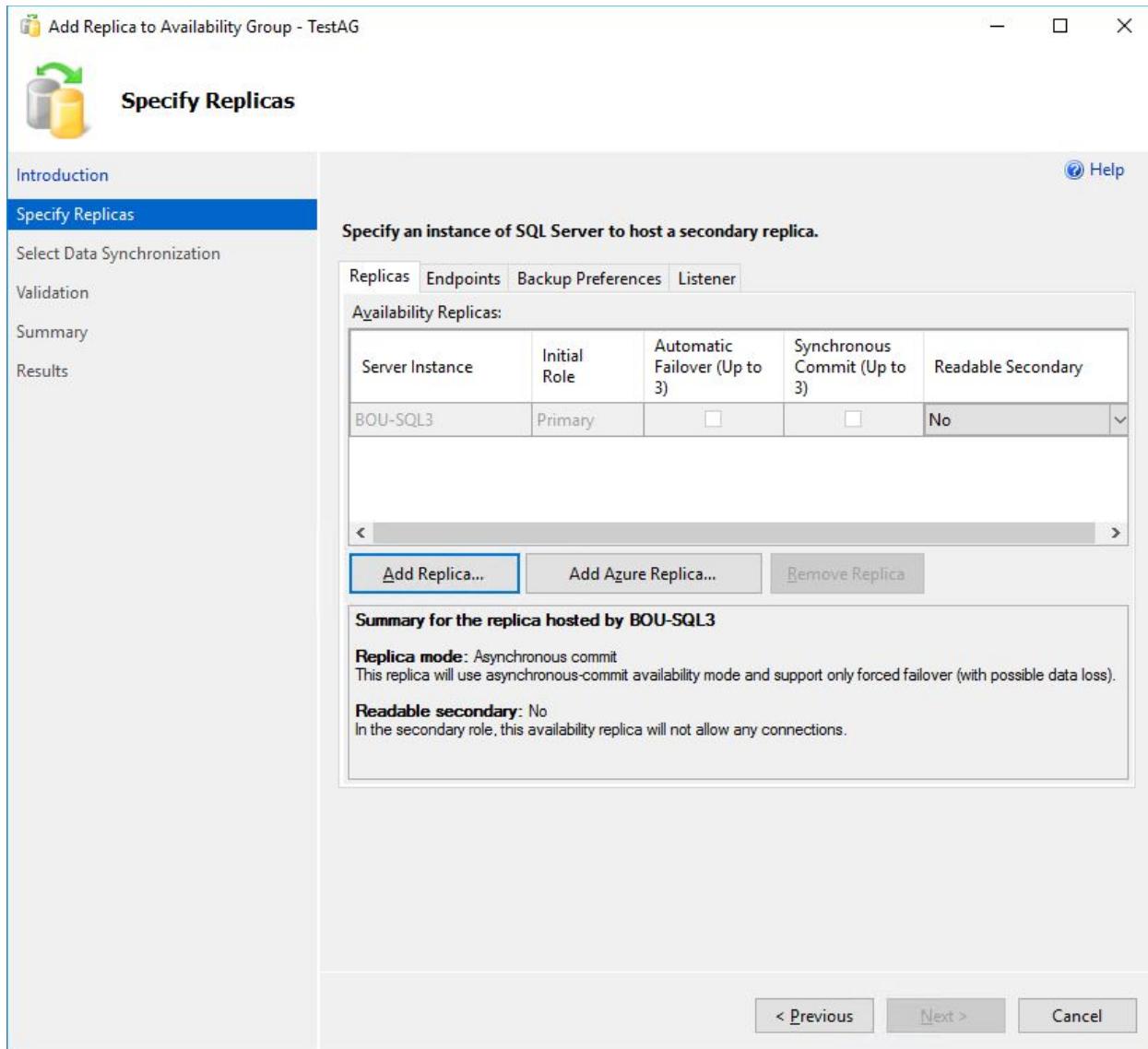
Change *Availability mode* to *Synchronous commit*.



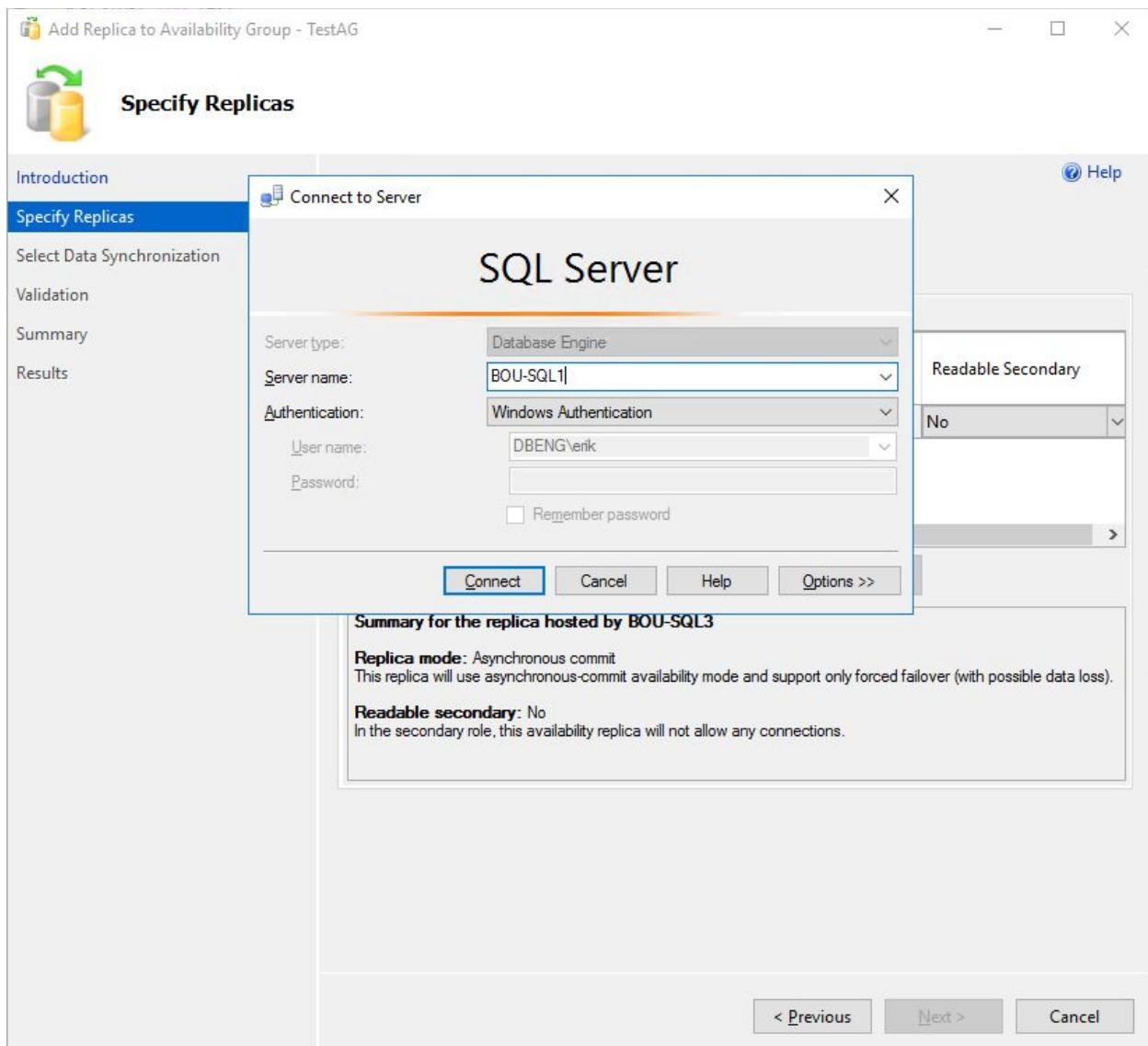
Add the 1st server back to the AG by right clicking on the AG and selecting *Add Replica...*



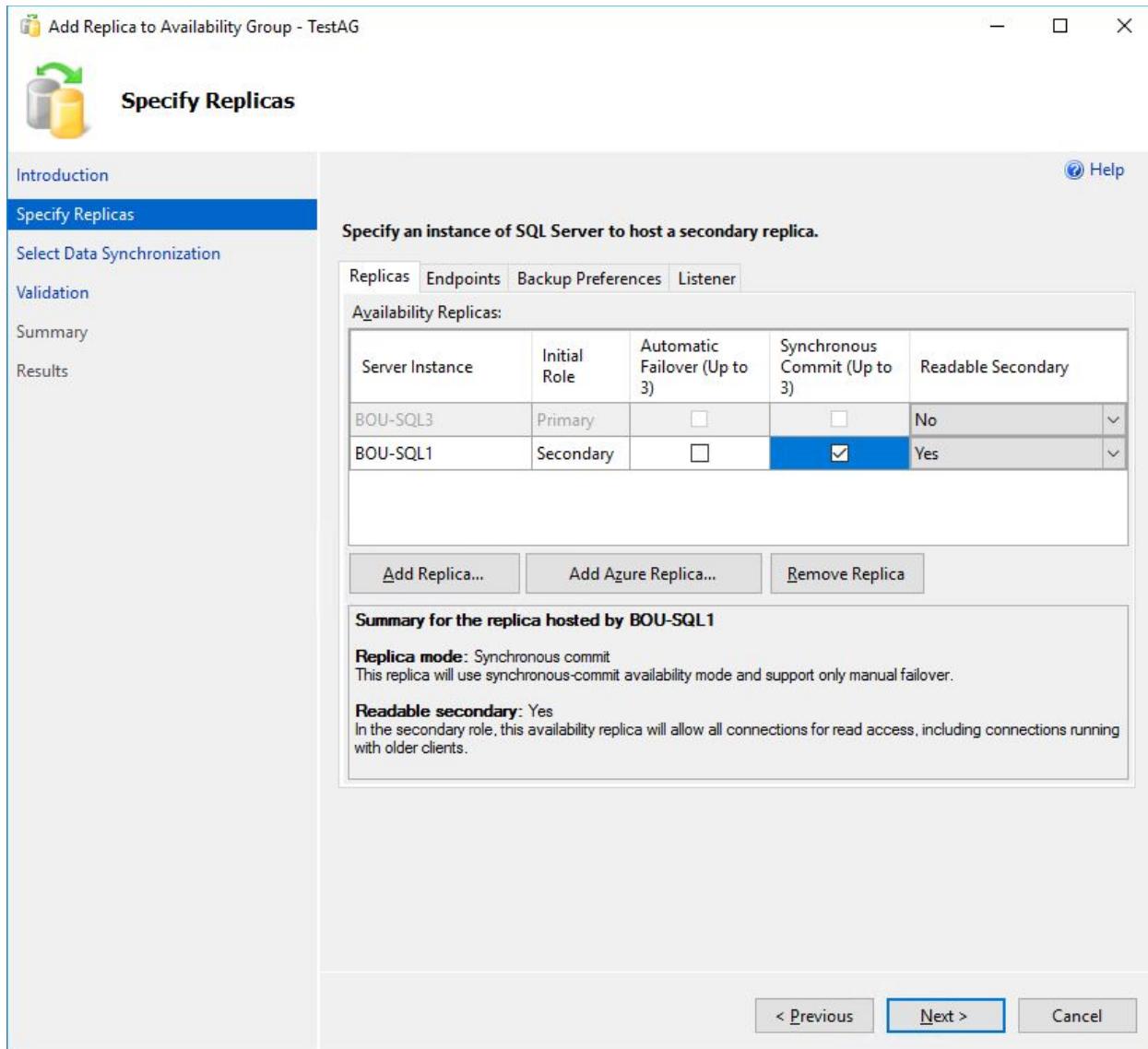
On the *Specify Replicas* page, click *Add Replica...*



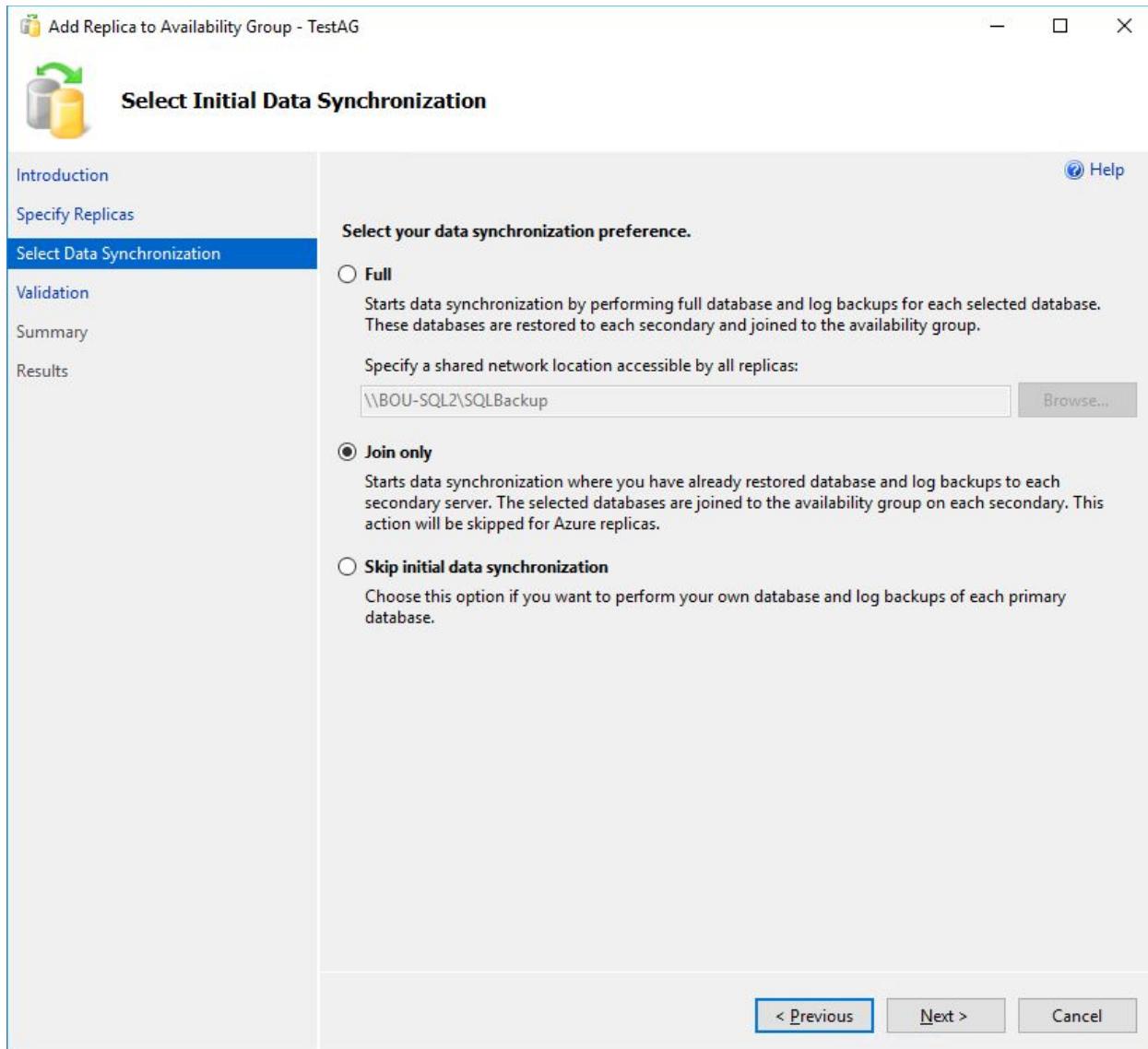
Specify the 1st server and click *Connect*.



Check the box for it to be a synchronous-commit replica as we are going to fail over to it without further data loss. We'll go back and switch the 3rd server to asynchronous-commit after the first 2 servers are back to synchronizing with each other.



Click *Next*, select *Join only* on the *Select Data Synchronization* page and then click *Next* again.



Ignore the listener warning and then click *Next*.

Add Replica to Availability Group - TestAG

## Validation

Introduction      Help

Specify Replicas

Select Data Synchronization

Validation

Summary

Results

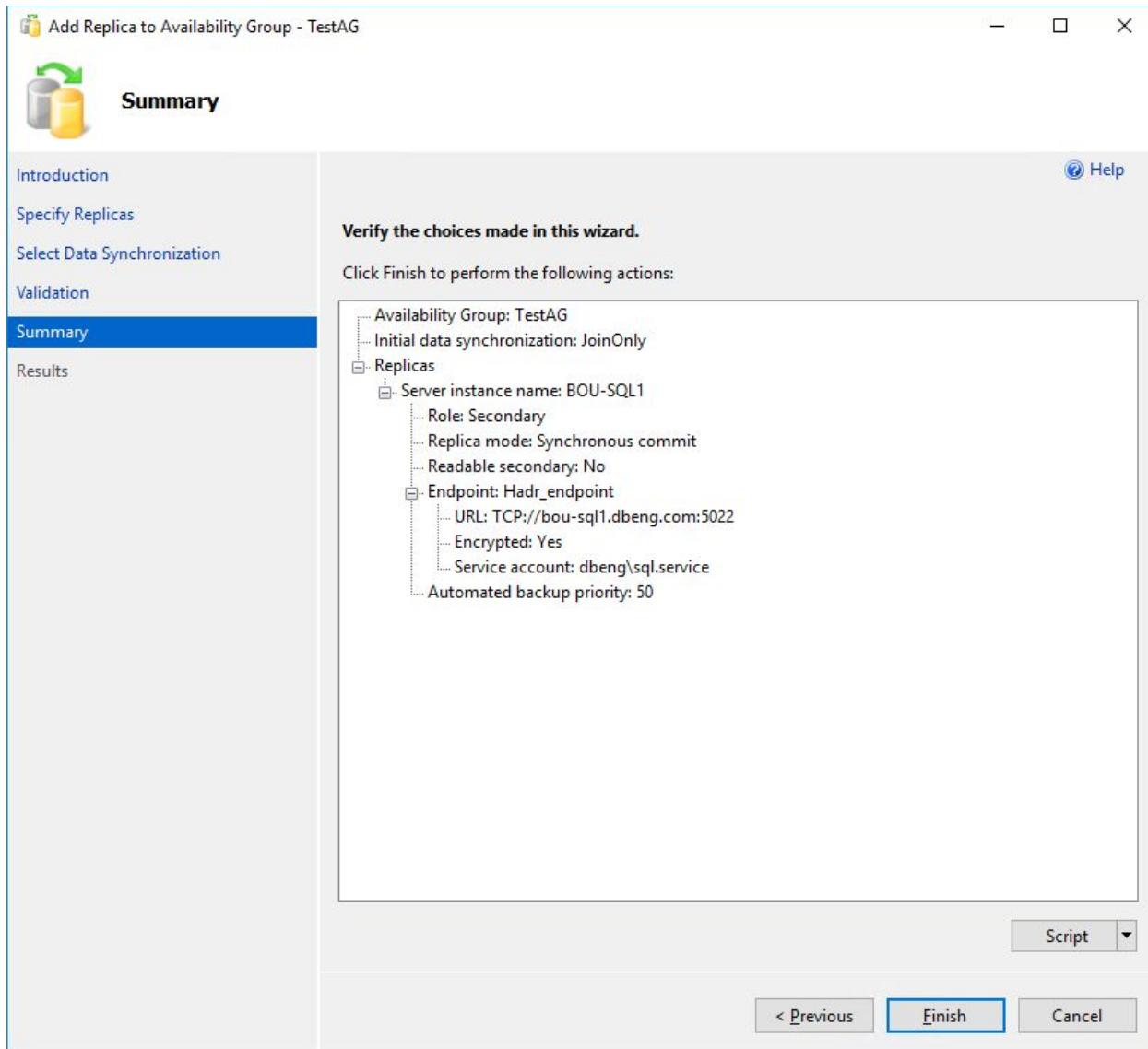
Results of availability group validation.

Name	Result
Checking whether the endpoint is encrypted using a compatible algorithm	Success
Checking shared network location	Skipped
Checking replica availability mode	Success
Checking for free disk space on the server instance that hosts secondary replica BOU-SQL1	Skipped
AddReplicaDatabaseExistenceValidator	Skipped
AddReplicaDatabaseFileCompatibilityValidator	Skipped
AddReplicaDatabaseFileExistenceValidator	Skipped
Checking the listener configuration	Warning

Re-run Validation

< Previous    Next >    Cancel

Click *Finish* to add the server as a replica to the AG.



Click Close to close out the *Add Replica to Availability Group* wizard.

The screenshot shows the 'Results' page of the 'Add Replica to Availability Group - TestAG' wizard. The title bar says 'Add Replica to Availability Group - TestAG'. On the left, a sidebar lists steps: Introduction, Specify Replicas, Select Data Synchronization, Validation, Summary, and Results (which is selected). The main area has a green checkmark icon and the message 'The wizard completed successfully.' Below this is a 'Summary:' table:

Name	Result
Configuring endpoints.	Success
Configuring endpoints.	Success
Starting the 'AlwaysOn_health' extended events session on 'BOU-SQL1'.	Success
Adding replica to availability group 'TestAG'.	Success
Joining secondaries to availability group 'TestAG'.	Success
Creating a full backup for 'Test'.	Skipped
Restoring 'Test' on 'BOU-SQL1'.	Skipped
Backing up log for 'Test'.	Skipped
Restoring 'Test' log on 'BOU-SQL1'.	Skipped
Joining 'Test' to availability group 'TestAG' on 'BOU-SQL1'.	Success

At the bottom are buttons: '< Previous', 'Next >', and 'Close'.

Refresh *Object Explorer* to see that server1 and server3 are replicas in the AG.

Run a FULL backup and a LOG backup on server3 and then restore those on server2 using WITH NORECOVERY.

Server3:

```
BACKUP DATABASE Test  
TO DISK = 'C:\SQLBackup\Test_20170909_1105.bak'  
WITH INIT;
```

```
BACKUP LOG Test  
TO DISK = 'C:\SQLBackup\Test_20170909_1105.trn'
```

```
WITH INIT;
```

Server2:

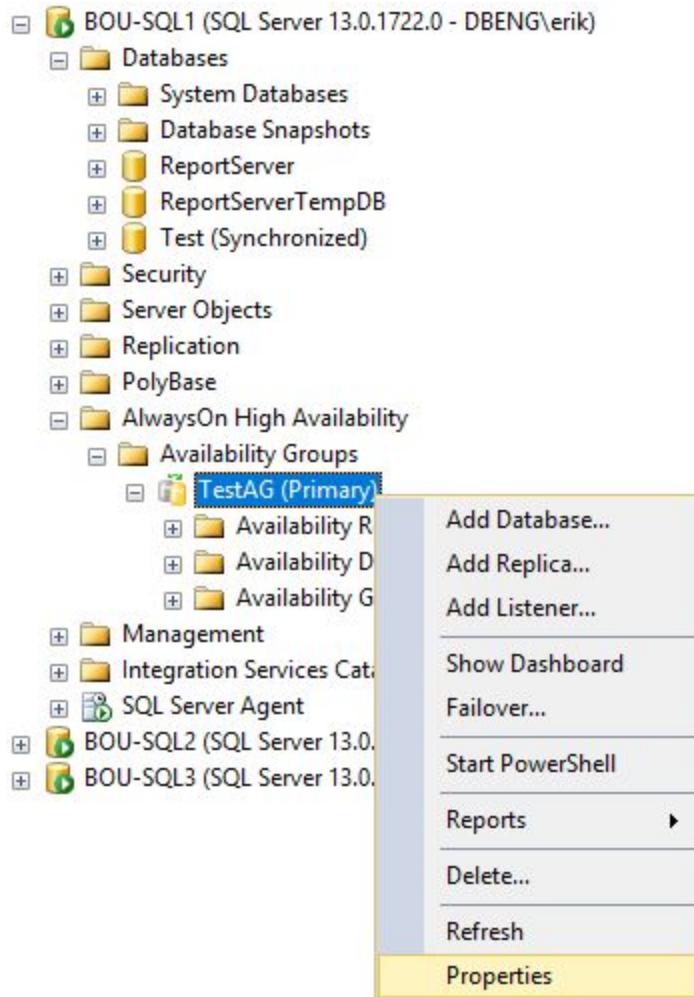
```
RESTORE DATABASE Test  
FROM DISK = 'C:\SQLBackup\Test_20170909_1105.bak'  
WITH NORECOVERY, REPLACE;
```

```
RESTORE LOG Test  
FROM DISK = 'C:\SQLBackup\Test_20170909_1105.trn'  
WITH NORECOVERY;
```

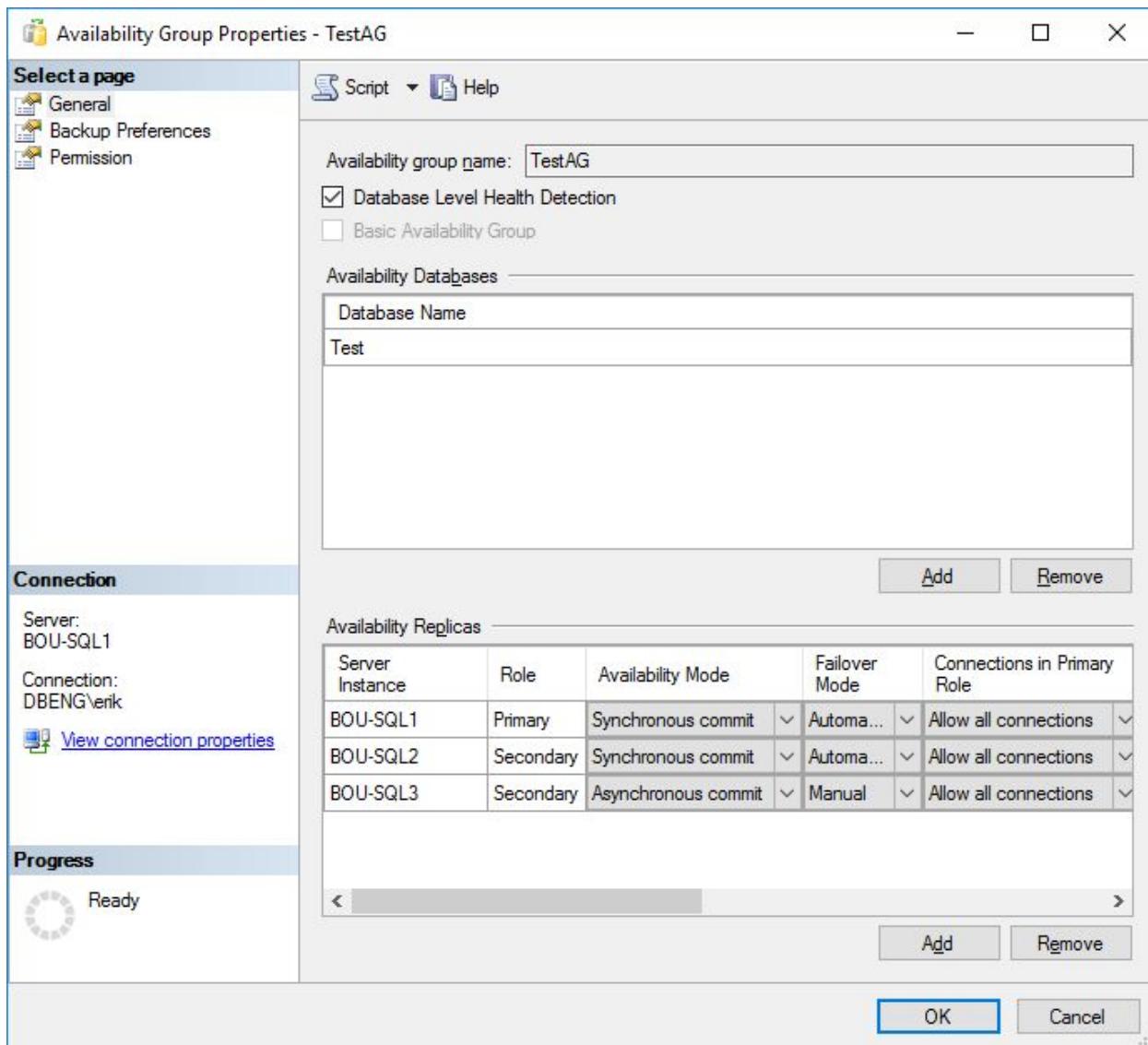
Add server2 as a replica to the AG using the same steps as was done for server1.

Failover to server1.

Set server3 to asynchronous-commit by right clicking on the AG from the new primary replica, server1 and then selecting *Properties*.



Set server3 to *Asynchronous commit* for *Availability Mode* and switch to *Automatic* for *Failover Mode* for servers 1 and 2.



Click OK. The AG is now back to the original settings before the first 2 servers were taken offline.

## Testing Recap

These tests aren't just about making sure your Always On Availability Group will be Always On.

It's also about making sure *you* will be Always On your game.

When you build a real production Availability Group, you'll want to step through these exact same four tests while taking screenshots and writing documentation just like we did. That way,

when disaster strikes, you - or any member of your team, including the mouth-breathers - will be able to handle the emergency simply by reading through your checklist.

When real disasters strike, like when you lose an entire zone, you're going to have your hands full. You're going to be trying to fix multiple servers at once, with lots of managers standing over your shoulder, asking when things will be back up and running. That's not the time you want to be Googling for answers for the first time, because that's when you'll make mistakes. Your pre-written checklist complete with screenshots will make sure you look like a rock star. No, not like Elvis in the bathroom - the good kind of rock star, like catching a beer thrown at him, and drinking it.

# Replacing a Server with a Bigger/Lesser Server

You can easily replace a server in GCE when using an Availability Group. Maybe you want to do this because of one of these scenarios:

- You have outgrown your hardware and need a bigger server
- You started off with too big of a server and want to downsize it
- The workload is higher during certain peak times of the year and you want to temporarily scale up

The AG already has 3 servers in it if you followed the steps in the testing section. Let's replace server1 with server3.

1. Failover to server2
2. Set server3 to synchronous-commit with automatic failovers
3. Remove server1 from the AG

That's it!

# Dealing With Availability Group Design Gotchas

In the immortal words of a timeless philosopher:

*“Your scientists were so preoccupied with whether or not they could, they didn’t stop to think if they should.”*

So now that we’ve shown you how to build and test your own dinosaur, let’s step back and ask: are Always On Availability Groups a good fit for your SQL Server infrastructure, and what are you going to have to work around?

## The Big Showstopper: Cross-Database Transactions

This one catches a lot of folks by surprise:

- Yes, Availability Groups fail over multiple databases together
- No, they are not guaranteed to be at the same transactional point in time
- Yes, this is true even in synchronous mode, even if all of the databases are in the same Availability Group

For details, read the fine print in Microsoft’s Books Online page about cross-database transactions. They do a great job of showing an example scenario with database mirroring, too, explaining why it’s just not doable today:

<https://msdn.microsoft.com/en-us/library/ms366279.aspx>

This is confusing because you may have read that SQL Server 2016 supports distributed transactions (DTC) with Always On Availability Groups, and that is technically true. However, they’re only supported between two databases on *two different servers*, not two databases on the same server. That’s kinda like me saying that shirts and pants are finally supported together, but only when they’re worn by two different people at any given time. I dunno how long you’d like to hang out with that support policy. And maybe hang out wasn’t the right term.

Anyway, the point is that if your application does cross-database transactions, Microsoft can’t guarantee that a transaction will be consistent when we fail over from one replica to another. A transaction’s data may be present in one database, but missing in another database.

Before choosing AGs, talk that Books Online page over with your management. The risk is fairly low, and in most cases, businesses are okay with the slight risks involved. However, sometimes they’re not, and it’s going to be different on a case-by-case basis.

If your application and business needs absolutely require cross-database transaction support, and you want automatic failover with zero data loss, your only option today is a failover clustered instance (FCI). We’re not covering that in this white paper, though.

## Database Objects That Aren't Protected by Availability Groups

AGs focus on copying data inside specific user databases: the ones you configure.

If you add more user databases later - especially if you're using apps like Microsoft SharePoint or kCura Relativity that let users add their own databases at any time - they won't be protected by default. You'll have to take manual actions to add them into your Availability Group, or build a script or app to add them.

The system databases aren't synchronized between replicas, which presents a few challenges.

- **master** - this database holds your logins, plus utility scripts you might have added like `sp_Blitz` or `sp_WhoIsActive`.
- **msdb** - holds your Agent jobs, backup history.
- **model** - the template database that SQL Server uses to create new databases. It's fairly unusual to see this, but it's possible that your app relies on putting data in here, and then seeing it show up in every newly created database.

Microsoft has published a huge Books Online page describing the metadata you'll need to keep synchronized:

<https://msdn.microsoft.com/en-us/library/ms187580.aspx>

We'll touch on just a few of the most important items here, but you'll want to read that page carefully if you use features like encryption, extended stored procedures, replication, and Service Broker.

### Synchronizing Logins Across Replicas

If you use Windows logins, you're best served by creating Active Directory groups, and then only synchronizing those groups across the replicas. This way, you'll have less changes to deal with. Instead of touching every server every time you need to give a new person access, just add them to the AD group centrally.

Windows login passwords are stored centrally in Active Directory, which means you don't have to lift a finger to keep those synchronized across replicas.

SQL logins are a lot more painful here. You have to touch every replica whenever you add a user, and that's rough. But much worse, whenever a user changes their own password, it's not changed across all replicas.

With these complexities, scripts have popped up to help you manage logins. Here are a couple of the more popular options:

- <http://www.sqlsoldier.com/wp/sqlserver/transferring-logins-to-a-database-mirror>
- [http://blogs.microsoft.co.il/yaniv\\_etrogi/2015/07/26/synchronize-alwayson-replicas-configuration-and-server-level-objects/](http://blogs.microsoft.co.il/yaniv_etrogi/2015/07/26/synchronize-alwayson-replicas-configuration-and-server-level-objects/)

There have been others, but before you try one, make sure it works with SQL Server 2016.  
(Many have been abandoned along the way.)

## Synchronizing Agent Jobs Across Replicas

Don't.

It sounds really tempting - just copy all of your Agent jobs across all of your replicas - but each Agent job can have:

- Different file paths - for example, in production, you may write your backups to one UNC path, but a different one in your DR site
- Different statuses and schedules - you may want to have some jobs run on a specific readable secondary, and stop running if it becomes the primary - or you may want some jobs to always run, like backing up the system databases
- Different versions of the truth - sometimes, human beings accidentally change a job on a single replica, and we're not sure which replica contains the "right" copy of a job

There are ways to work around these challenges using dynamic code in jobs. For example, if you use Ola Hallengren's backup scripts (<http://ola.hallengren.com>), they run on every replica every time, and then they examine which databases they should back up based on a few factors.

The problem is, all of your jobs have to be as smart as Ola's - and in most cases, they're not today, and this represents a new development task that somebody will have to start working on.

**Short term, use source control for your Agent jobs.** Script them out, check them into source control, and then deploy them across all servers. If your jobs aren't identical across all of your replicas, consider using different repos for the different servers. (Remember, think cattle, not pets.)

**Long term, build logic into your Agent jobs.** The jobs should be identical across all replicas, and be driven by DMVs like sys.fn\_hadr\_backup\_is\_preferred\_replica and sys.dm\_hadr\_availability\_replica\_states. The challenge here, and why we say that it's a long term goal, is that you may need to connect directly to various replicas using linked servers. There are known issues with the DMVs where they stop updating, like this one:

<https://connect.microsoft.com/SQLServer/feedback/details/779206/sql-server-2012-alwayson-availability-groups-dashboard-stops-updating-after-thread-exhaustion>

## Services That Aren't Protected by Availability Groups

SQL Server Analysis Services, Integration Services, and Reporting Services are popular business intelligence tools that come free in the box with SQL Server. All three of them query the SQL Server engine, and they're installed with the same installer as the engine.

Thing is, they don't really have that much in common with the SQL Server engine.

They use completely different high availability and disaster recovery methods. Sure, you can protect the SSIS & SSRS database repositories by putting that database into an Availability Group. However, the SSAS/SSIS/SSRS services themselves don't fail over together when the Availability Group databases and listener fail over.

For high availability of these other services, we'd suggest starting with these resources:

SSAS High Availability and Scalability: Books Online talks about why you can't put the SSAS data in an Always On Availability Group, and why you probably want multiple SSAS servers.  
<https://msdn.microsoft.com/en-us/library/mt668924.aspx>

SSIS in a Cluster: Books Online explains why you probably shouldn't do it.  
<https://msdn.microsoft.com/en-us/library/hh213127.aspx>

Hosting SSISDB in Always On Availability Groups: Microsoft discusses issues you can see when failing over or patching.

<https://msdn.microsoft.com/en-us/library/mt163864.aspx>

SSRS High Availability: Microsoft points out that SSRS is stateless, so the easiest way to protect it is to build multiple SSRS servers and point them at the same repository.

<https://msdn.microsoft.com/en-us/library/bb522745.aspx>

Database administrators may be frustrated that these tools don't have a great HA/DR story.

Cloud-savvy sysadmins, however, will be right at home because these services are a perfect fit for the new cloud administration mentality: treat your servers like cattle, not like pets.

This represents a new way of thinking for DBAs, but it's the way that sysadmins have been scaling out web application servers. Now, it's possible to do with SQL Server as well - but as with your application servers, it means new automation tasks for you to learn. Here's how it works:

1. Put your group of SSAS/SSIS/SSRS servers behind a load balancer like Google Cloud Load Balancing
2. Script the build & configuration of your SSAS/SSIS/SSRS servers just once

3. Check those scripts into source control
4. If the business is willing to run standby servers, use the scripts to spin up standby servers in another region or zone
5. When you need new servers (either due to failures or performance boosts), just run scripts, and add these new servers into your instance group

This gives you new flexibility not just to handle outages, but also to handle scaling issues. Need faster servers? Just spin up new ones with the horsepower you want, and decommission the old ones.

So it's probably time for us to start talking about SQL Server performance capacity design and management in Google Compute Engine, and we've got that covered in a separate white paper.

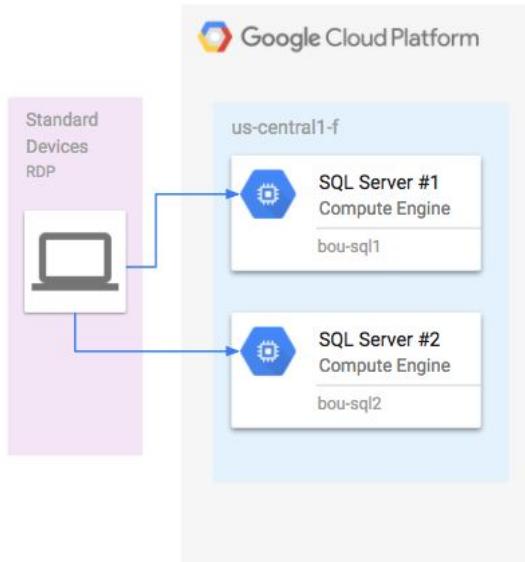
# Recap, Next Steps, and Learning Resources

Whew.

We built a 2-node Always On Availability Group, tested it in four different ways, and talked about the parts of SQL Server that aren't protected by AGs.

Now, just from reading this, you're already better prepared than a lot of the AG implementations we've seen. Most folks just step through the wizards and hope for the best.

When you're ready to build your go-live plan and migrate to a new Always On Availability Groups infrastructure, here's the big-picture next steps:



- Set your RPO and RTO goals in writing, which will drive the design of your infrastructure
- Review your existing licensing to see if you can leverage it in the cloud
- Design your production infrastructure - how many replicas you're going to have in each region and zone, which ones will be synchronous vs asynchronous
- Size each VM - we've got a separate white paper on that, [SQL Server Performance Tuning in Google Compute Engine](#)
- Determine your licensing strategy - bring-your-own-licensing or renting from Google
- Build the staging infrastructure - identical to production, but an environment that you'll be comfortable breaking over and over while you learn
- Build the production infrastructure
- Design your migration plan - how you're going to get your databases into the primary replica, and fail your applications over to their new connection string
- Rehearse the migration plan
- Go live

Obviously, this is a lot of work, and your learning journey isn't over.

If you'd like to continue your learning, check out our online video course, [DBA's Guide to SQL Server High Availability and Disaster Recovery](#). We explain how to choose the right HA/DR features for your needs, then implement and test them.

If you'd like our help, drop us a line:

- Learn how our SQL Critical Care® process works:  
<https://www.brentozar.com/sql-critical-care/>
- Email us at [Help@BrentOzar.com](mailto:Help@BrentOzar.com), or use our contact form:  
<https://www.brentozar.com/contact/>

## Change Log

v1.01, 2017-03-20:

- Updated syntax for “gcloud compute disks create” commands to reflect new OS image names.

v1.0, 2017-03-03:

- Initial public release.