Exercise - Create a website hosted in Azure

As a developer for Tailwind Traders, you likely have expertise creating applications. As you migrate to Azure, many of the steps that you'll follow to set up a website in the cloud will parallel the steps that you followed when you created websites in your company's datacenter. For example, you need to choose where you'll create your website, and then allocate the necessary resources. In Azure, the physical hardware is managed for you, so your tasks are to choose where your website will be located and which resources to provide.

In this exercise, you'll create an Azure App Service instance to host a WordPress website.

Activate a sandbox to complete this exercise

This exercise requires you to use a sandbox on Microsoft Learn to complete. A **sandbox** gives you access to resources. Your Azure subscription will not be charged. The sandbox may only be used to complete training Learn. Use for any other reason is prohibited, and may result in permanent loss of access to the sandbox. Sig
activate sandbox

Azure terminology and concepts

Before you get started, let's review and discuss some basic terms and concepts that you'll need to know when you create your website.

What is App Service?

App Service is an HTTP-based service that enables you to build and host many types of web-based solutions without managing infrastructure. For example, you can host web apps, mobile back ends, and RESTful APIs in several supported programming languages. Applications developed in .NET, .NET Core, Java, Ruby, Node.js, PHP, or Python can run in and scale with ease on both Windows- and Linux-based environments.

For this exercise, we want to create a website in less than the time it takes to eat lunch. So, we're not going to write any code. Instead, you'll deploy a predefined application from Azure Marketplace.

What is Azure Marketplace?

Azure Marketplace is an online store that hosts applications that are certified and optimized to run in Azure. Many types of applications are available, ranging from AI and machine learning to web applications. As you'll see in a couple of minutes, deployments from the store are done via the Azure portal by using a wizard-style user interface. This user interface makes evaluating different solutions easy.

We're going to use one of the WordPress application options from Azure Marketplace for our website.

Create resources in Azure

Typically, the first thing we'd do is to create a *resource group* to hold all the things that we need to create. The resource group allows us to administer all the services, disks, network interfaces, and other elements that potentially make up our solution as a unit. We can use the

Azure portal to create and manage our solution's resource groups. Keep in mind that you can also manage resources via a command line by using the Azure CLI. The Azure CLI is a useful option if you need to automate the process in the future.

In the free Azure sandbox environment, you'll use the pre-created resource group **[sandbox resource group name]**, and you don't need to do this step.

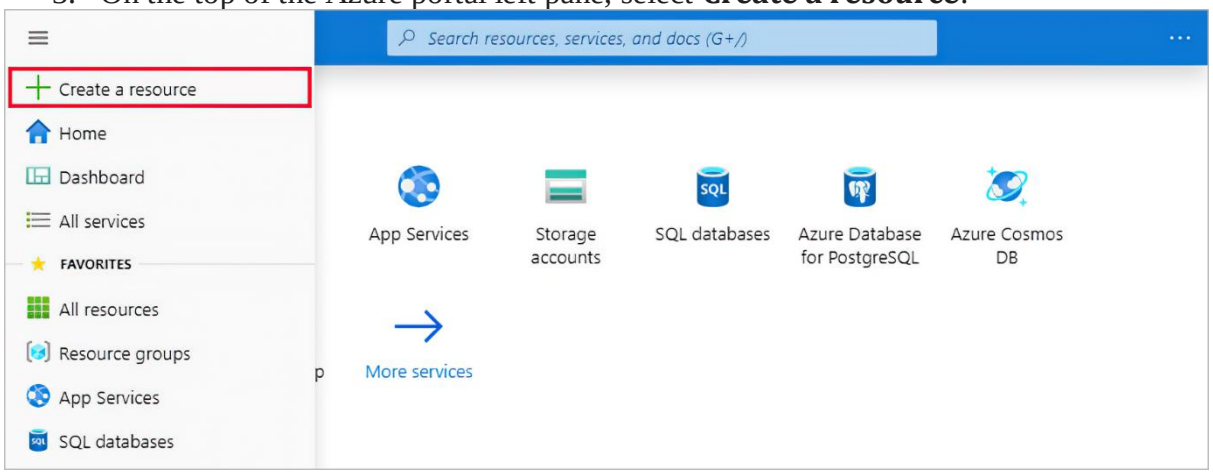
Choose a location

The free sandbox allows you to create resources in a subset of the Azure global regions. Select a region from this list when you create resources:

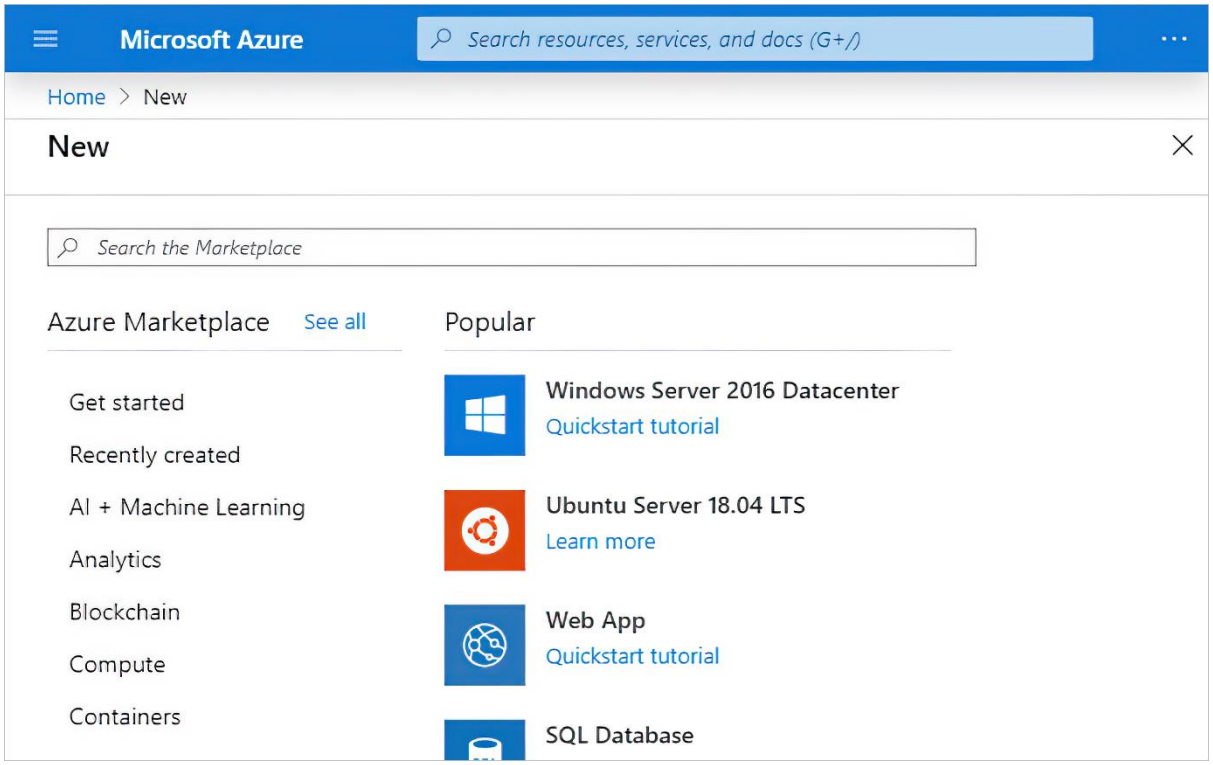
- westus2
- southeastasia
- japaneast
- brazilsouth
- australiasoutheast
- centralindia
- southcentralus
- centralus
- eastus
- westeurope

Create a WordPress website

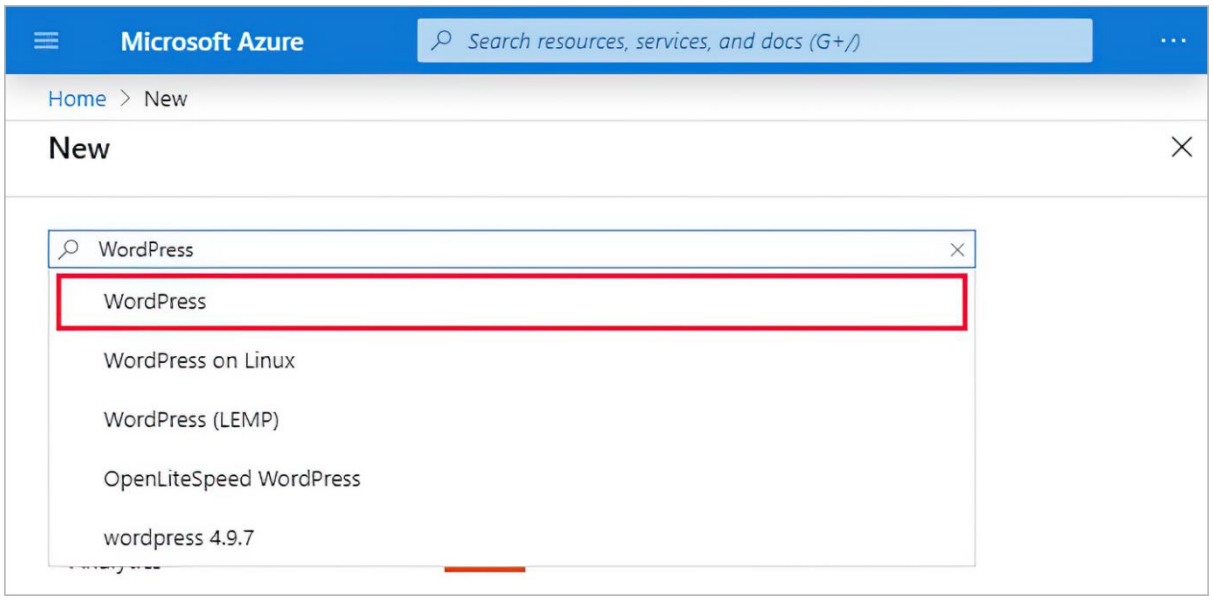
1. If you haven't done so already, verify that you've activated the sandbox. Activating the sandbox allocates the subscription and resource group you'll use in this exercise. This step is required for any Microsoft Learn exercises that use a sandbox.
2. Sign in to the Azure portal by using the same account you used to activate the sandbox.
3. On the top of the Azure portal left pane, select **Create a resource**.



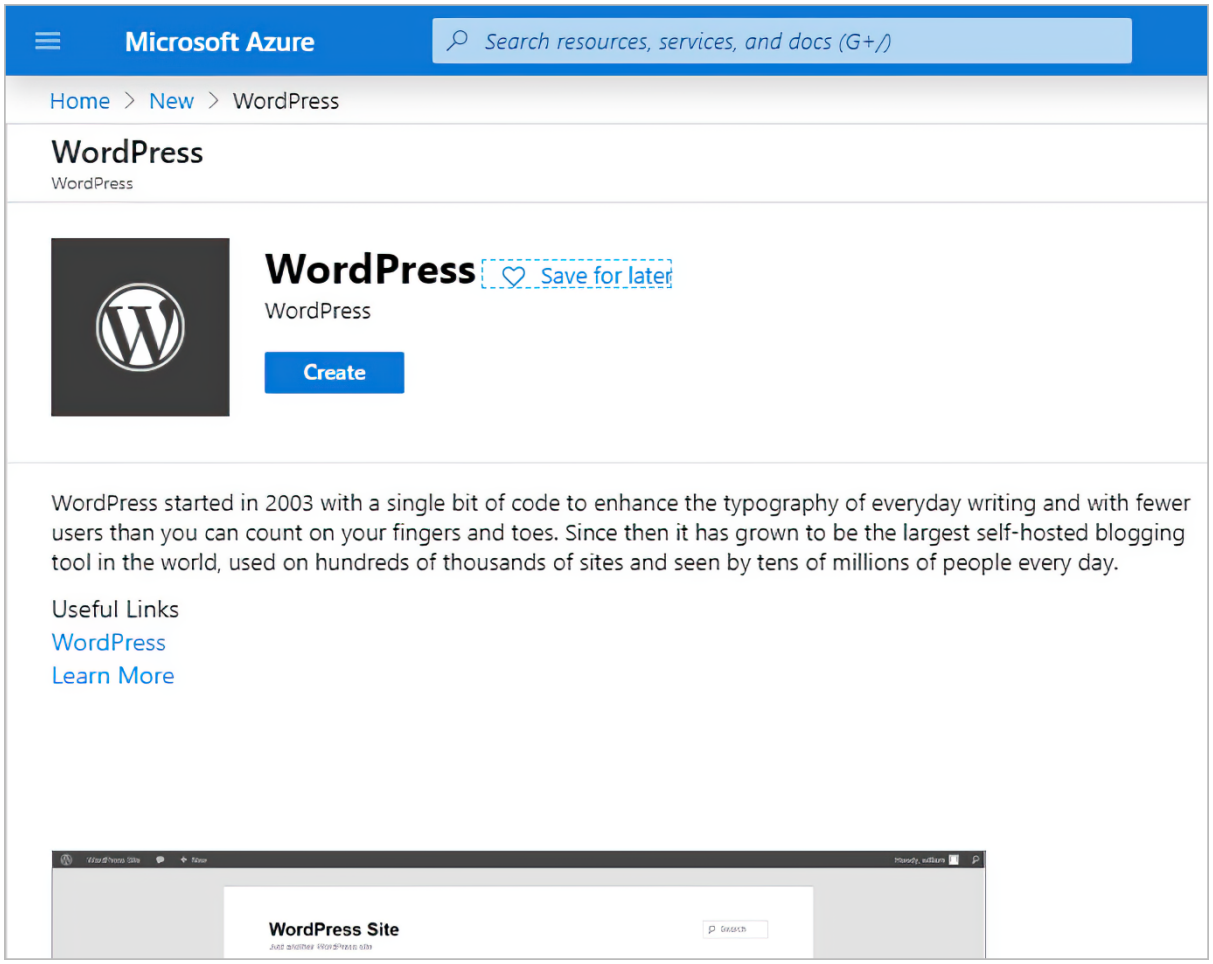
This option takes you to **Azure Marketplace**.



4. Azure Marketplace has many services, solutions, and resources available for you to use. We know that we want to install WordPress, so we can do a quick search for it. In the **Search the Marketplace** box with the listed application options, enter **WordPress**. Select the default **WordPress** option from the list of options available.



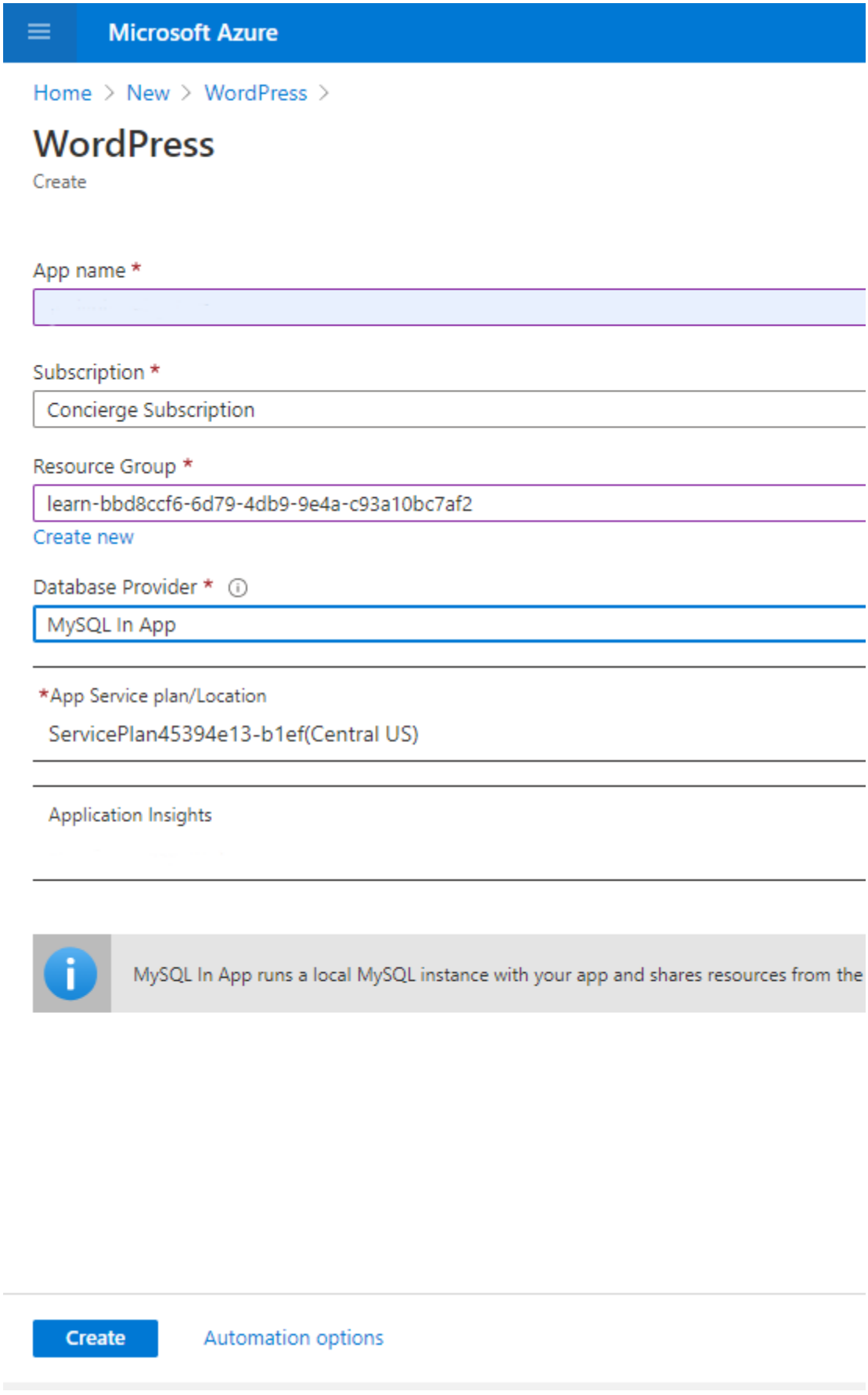
5. In the pane that appears, you'll typically find more information about the item you're about to install, such as the publisher, a brief description of the resource, and links to more information. Make sure to review this information. Select **Create** to begin the process to create a WordPress app.



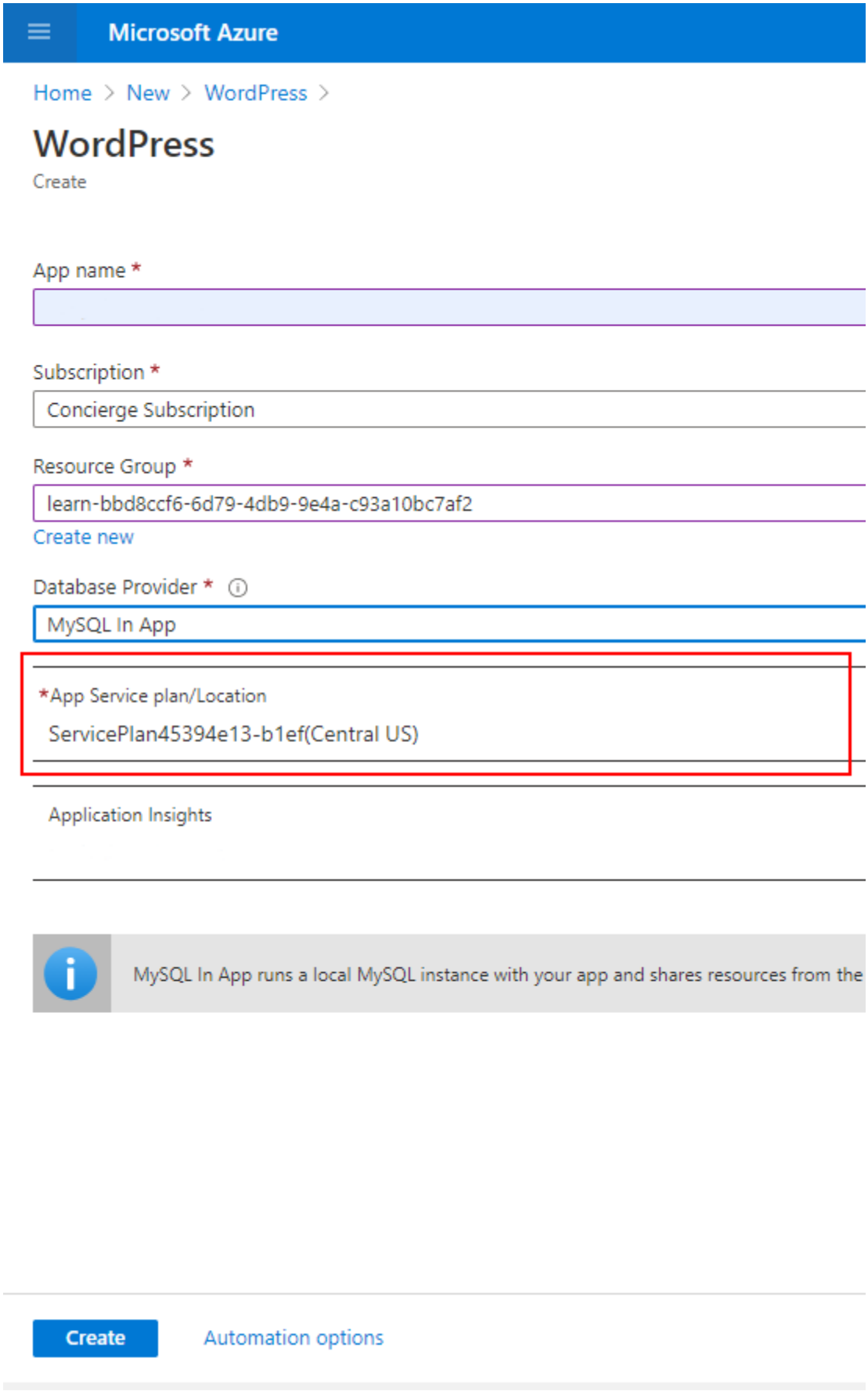
6. Several options to configure your deployment appear. Enter the following information:

Property	Value
App name	Choose a unique value for the app name. It will form part of a fully qualified domain (FQDN).
Subscription	Make sure **Concierge Subscription** is selected.
Resource Group	Select the **Use existing** option, and then select the **[sandbox resource group nar** group from the dropdown.
Database Provider	From the dropdown, select **MySQL in App**.
App Service plan/Location	You'll change the App Service plan in the next step.
Application Insights	Leave at the default configuration.

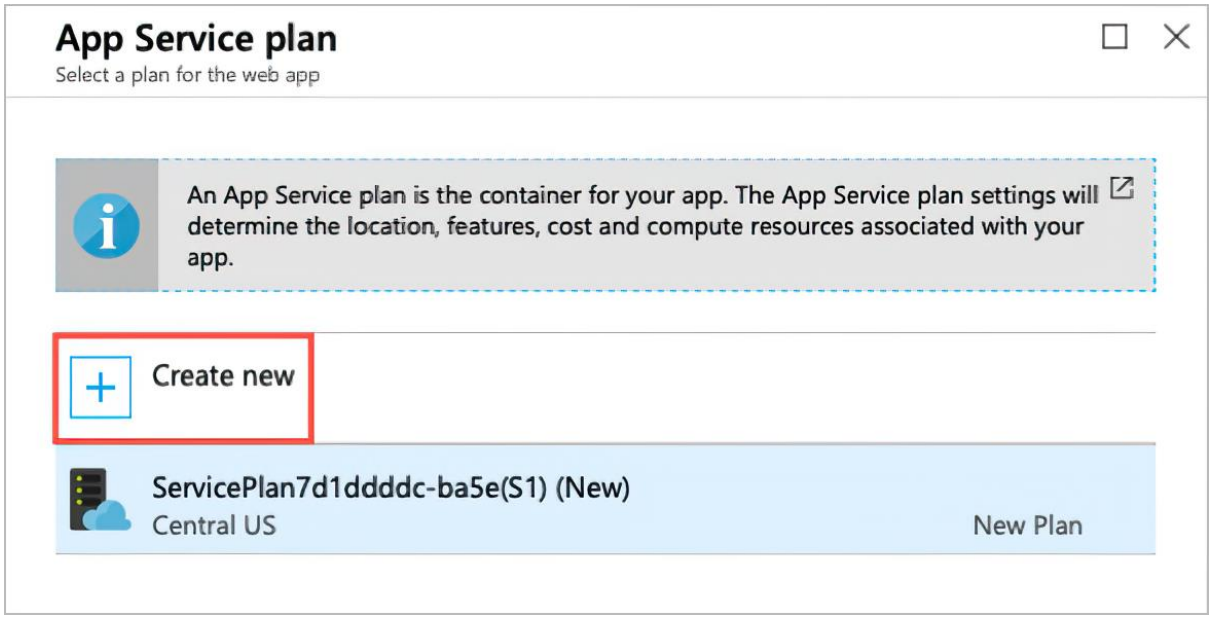
Your configuration should look like this example:


Microsoft Azure
Home > New > WordPress >
WordPress
Create
App name *
Subscription *
Concierge Subscription
Resource Group *
learn-bbd8ccf6-6d79-4db9-9e4a-c93a10bc7af2
Create new
Database Provider *
MySQL In App
*App Service plan/Location
ServicePlan45394e13-b1ef(Central US)
Application Insights
MySQL In App runs a local MySQL instance with your app and shares resources from the
Create
Automation options

7. Now let's configure the App Service plan to use a specific pricing tier. The App Service plan specifies the compute resources and location for the web app. Select **App Service plan/Location**.



8. In the **App Service plan** pane, select **Create new**.

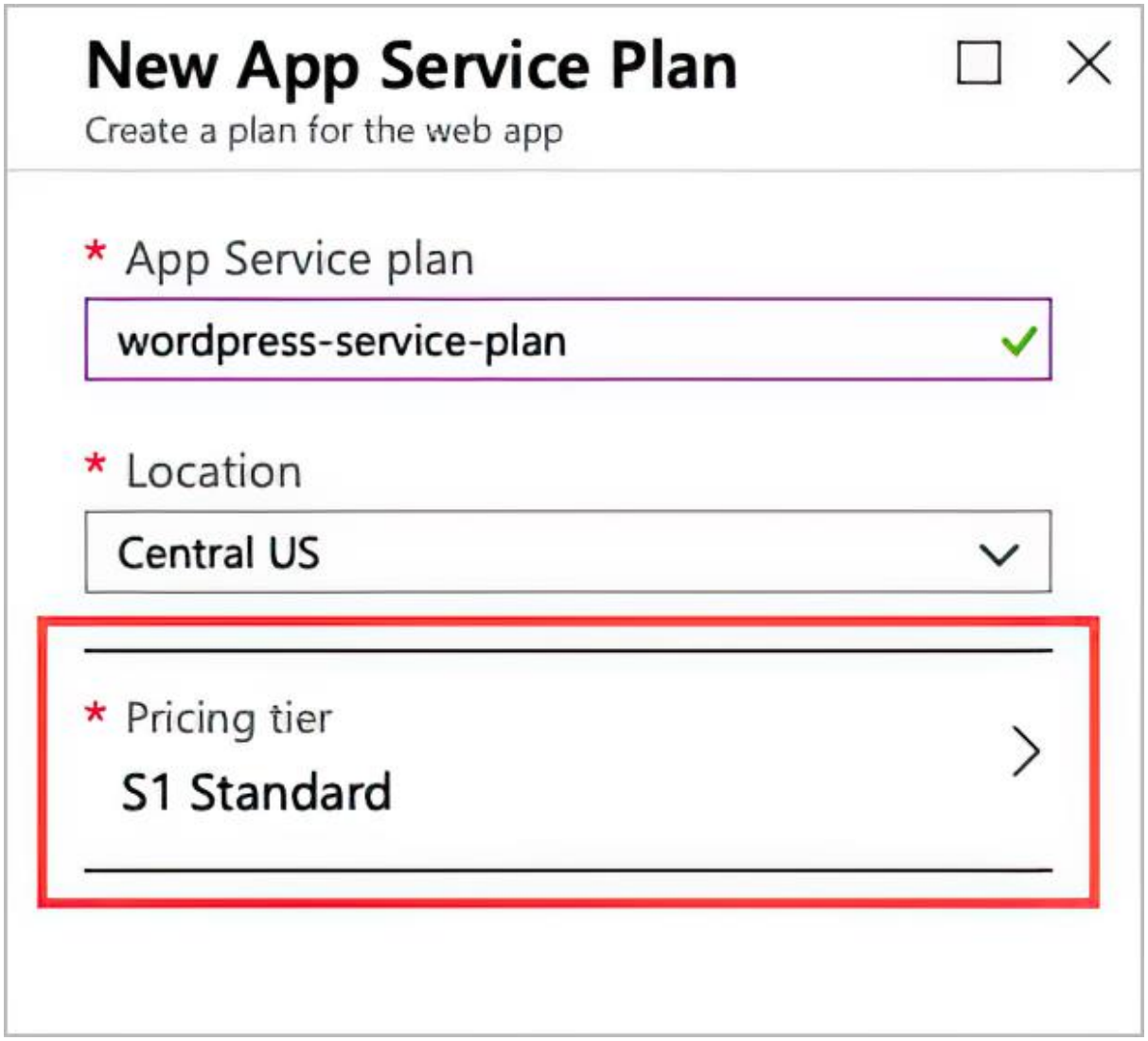


Screenshot of the Azure portal showing the App Service plan pane with the Create new button highlighted

9. In the **New App Service plan** pane, enter a name for the new service plan.

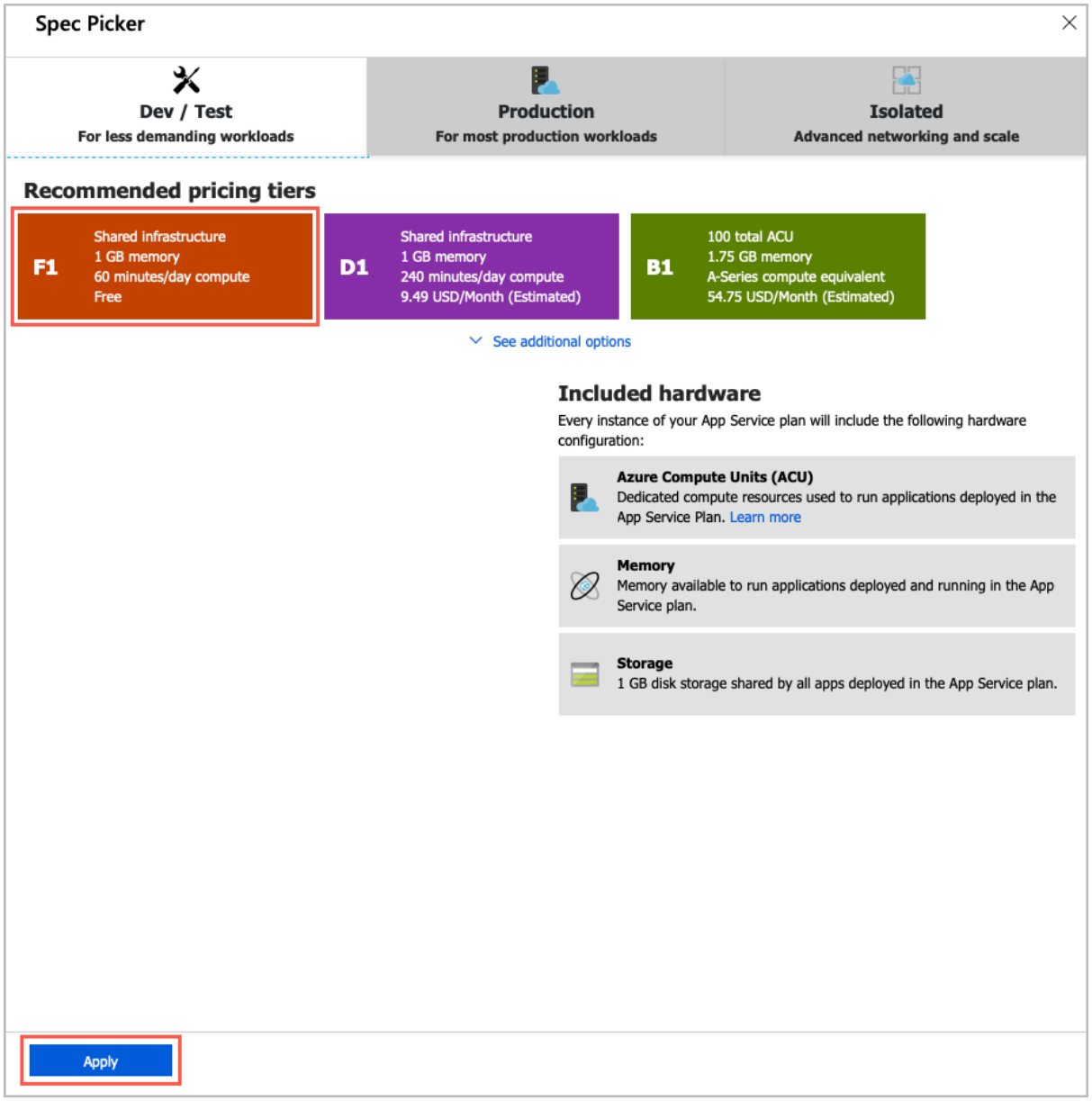
10. For **Location**, select **Central US** to make sure we choose a region that allows the service plan you'll choose. Normally, you'll select the region that's closest to your customers while offering the services you need.

11. Select **Pricing tier** to see the performance and feature options of the various types of service plans.



12. The **Spec Picker** allows us to select a new pricing tier for our application. This screen opens to the **Production** tab, with the S1 pricing tier selected. We'll select a new pricing tier from the **Dev / Test** tab for our website.

Select the **Dev / Test** tab, then select the **F1** pricing tier, and then select **Apply**.



Test section selected and the free F1 tier and the Apply button highlighted

13. Back on the **New App Service plan** pane, select **OK** to create the new plan, and close the pane.

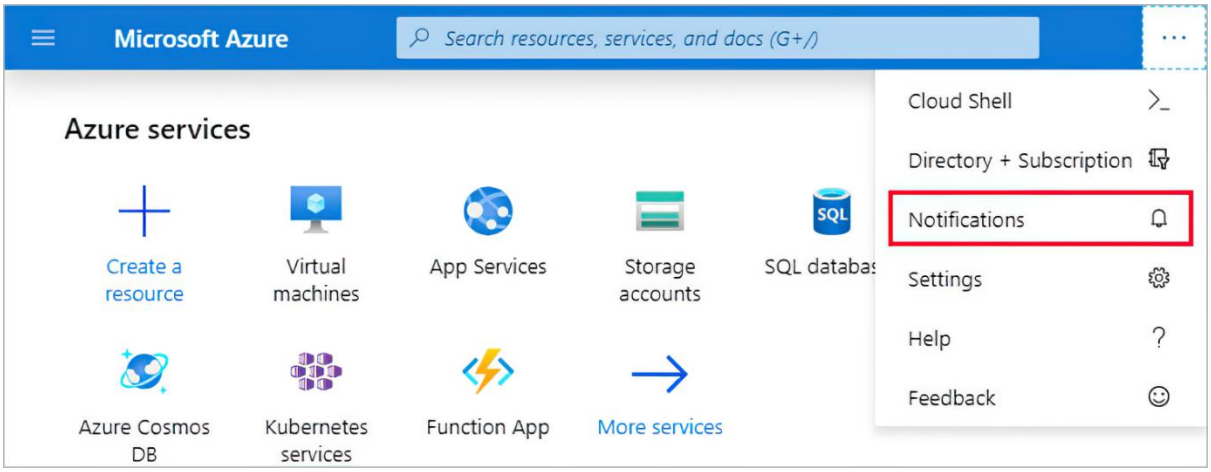
14. Finally, select the **Create** button to start the deployment of your new site.

Verify your website is running

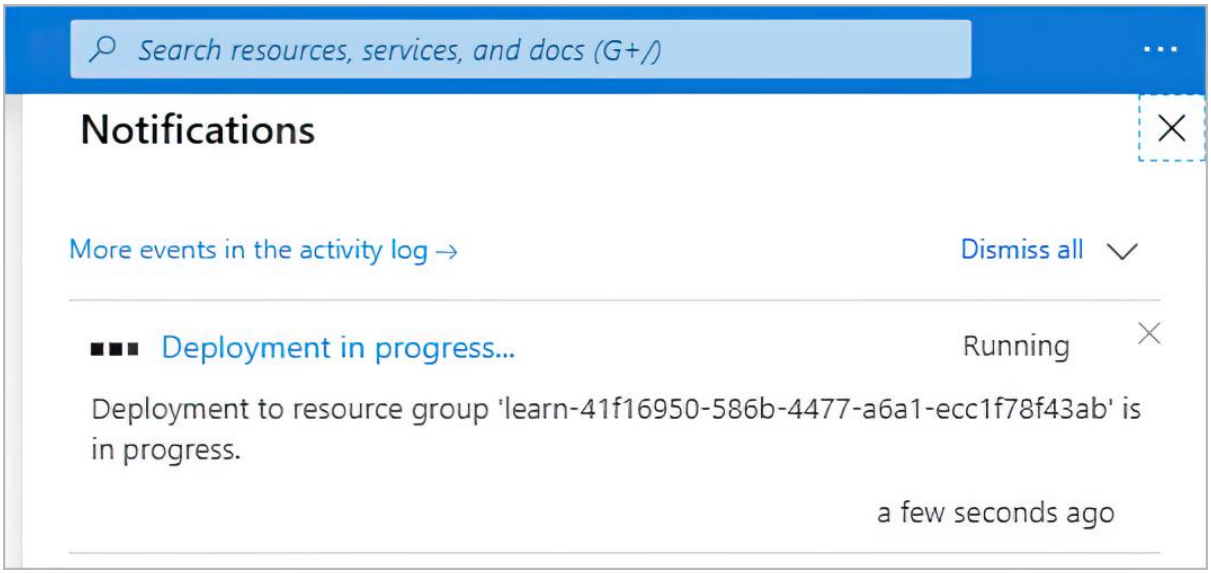
The deployment of the new website can take a few minutes to complete. You're welcome to explore the portal further on your own.

We can track the progress of the deployment at any time.

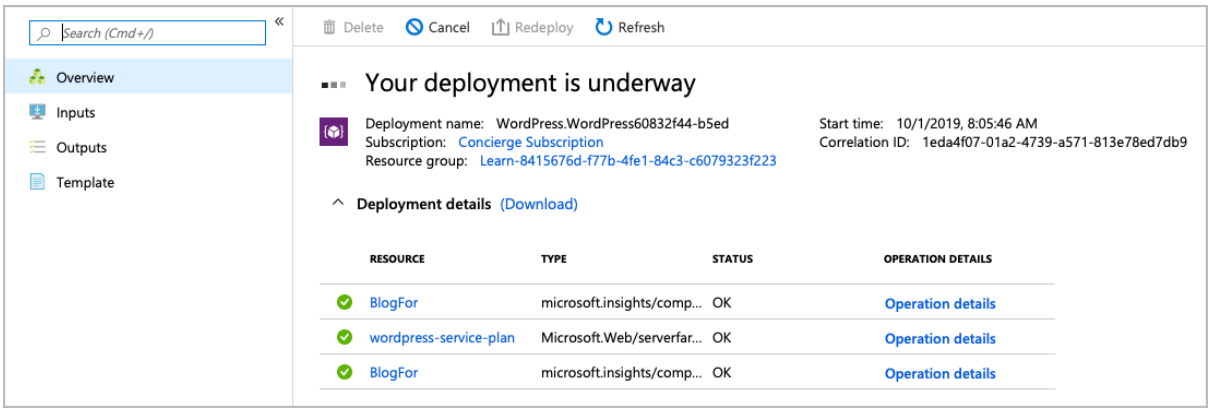
1. Select the Notifications bell icon at the top of the portal. If your browser window width is smaller, it might be shown when you select the ellipsis (…) icon in the upper-right corner.



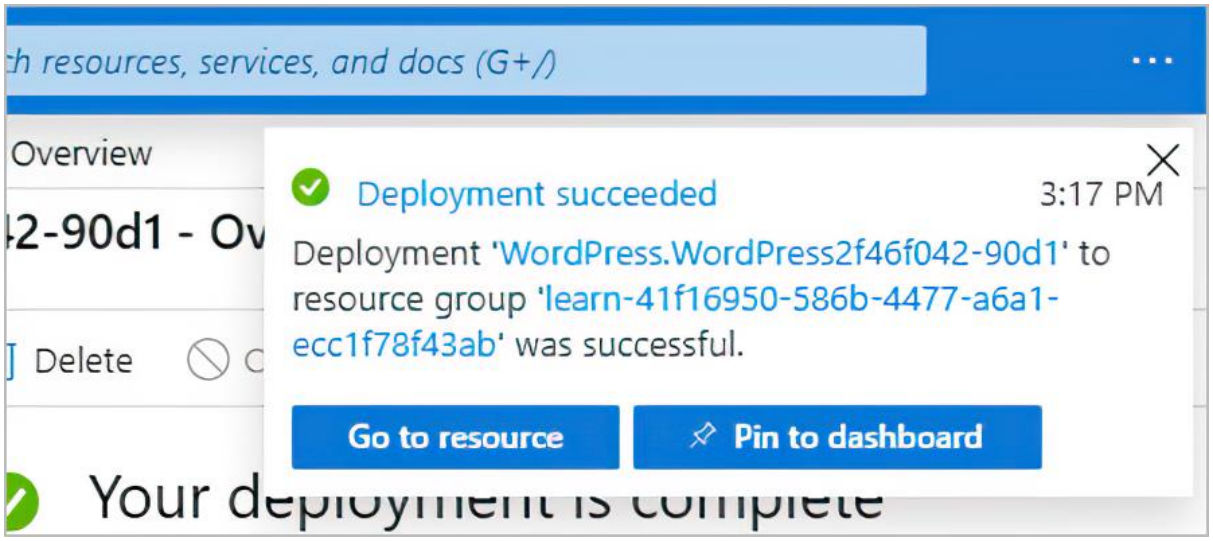
2. Select Deployment in progress to see the details about all the resources that are created.



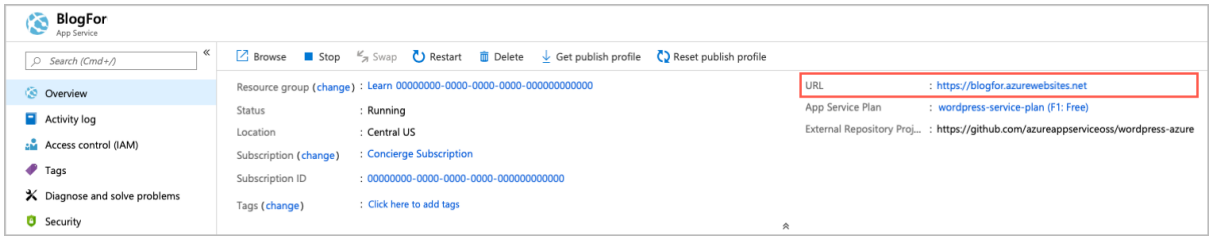
Notice how resources are listed as they're created and the status changes to a green check mark as each component in the deployment completes.



3. After the deployment status message changes to **Your deployment is complete**, you'll notice the status in the **Notifications** dialog box changes to **Deployment succeeded**. Select **Go to resource** to go to the App Service overview.

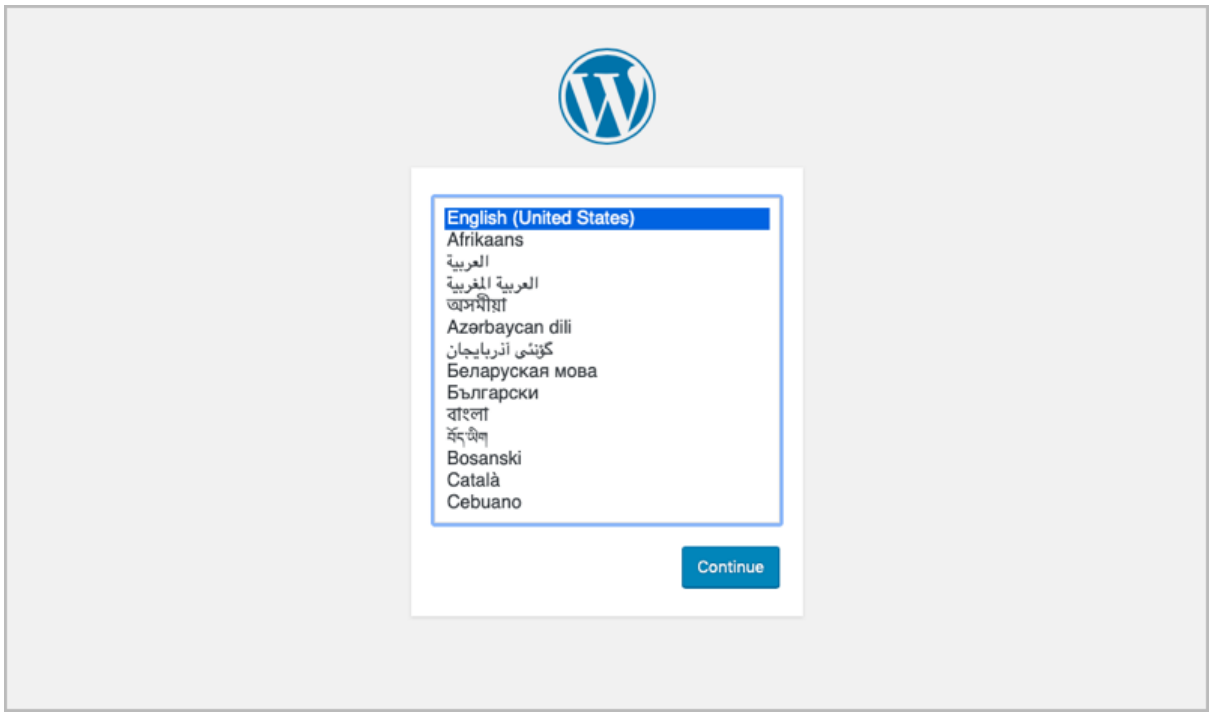


4. Find the **URL** in the **Overview** section.



5. Copy the **URL** information by selecting the **Copy to clipboard** icon at the end of URL.

6. Open a new tab in your browser, paste this URL, and press Enter to browse to your new WordPress site. You can now configure your WordPress site, and add content.



Exercise - Protect a storage account from accidental deletion by using a resource lock

Completed100 XP

- 8 minutes

In this exercise, you see how resource locks help prevent accidental deletion of your Azure resources.

To do so, you create a resource group from the Azure portal. Think of a resource group as a container for related Azure resources. Then you add a lock to your resource group and verify that you can't delete the resource group.

You then add a storage account to your resource group and see how the lock from the parent resource group prevents the storage account from being deleted. A storage account is a container that groups a set of Azure Storage services together.

Important

You need your own **Azure subscription** to complete the exercises in this module. If you don't have an Azure subscription, you can still read along.

Create the resource group

Here you create a resource group that's named **my-test-rg**.

1. Go to the Azure portal and sign in.
2. At the top of the page, select **Resource groups**.
3. Select **+ New**. The **Create a resource group** page appears.
4. In the **Basics** tab, fill in the following fields.

   Setting

   Value

   Project details

   Subscription

   Your Azure subscription

Resource group

my-test-rg

Resource details

Region

(US) East US

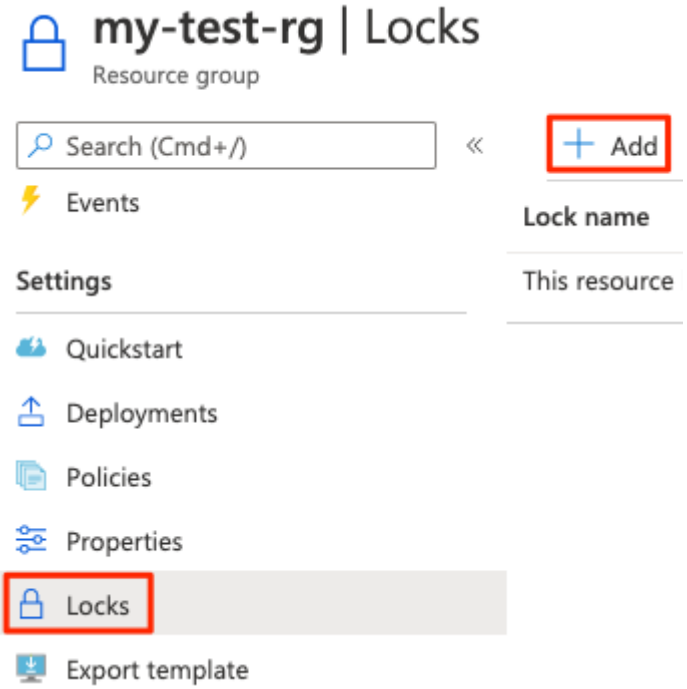
You can also select a region that's closer to you.

5. Select **Review + create**, and then select **Create**.

Add a lock to the resource group

Add a resource lock to the resource group. To do so:

1. From the Azure portal, select your resource group, **my-test-rg**.
2. Under **Settings**, select **Locks**, and then select **Add**.



3. Fill in these fields.

Setting

Value

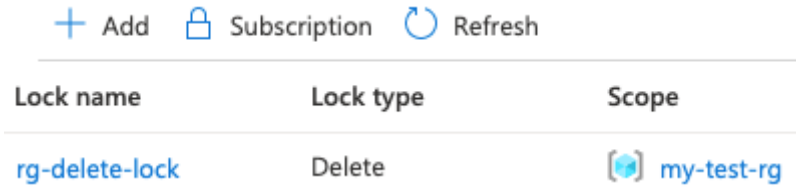
Lock name

rg-delete-lock

Lock type

Delete

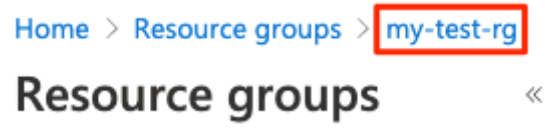
4. Select **OK**. You see that the resource lock is applied to your resource group.



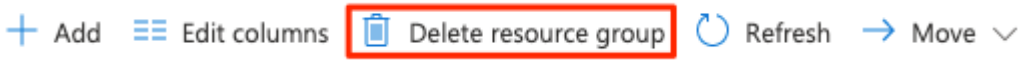
Verify that the resource group is protected from deletion

Here, you verify protection by attempting to delete the resource group.

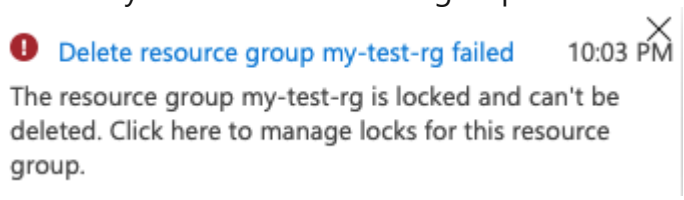
1. From the top of the page, select **my-test-rg** to go to your resource group's overview page.



2. Select **Delete resource group**.



3. At the prompt, enter **my-test-rg**, and then select **OK**. You see a message that tells you that the resource group is locked and can't be deleted.



Protect a storage account from accidental deletion

Here, you add a storage account to your resource group and see how the lock from the parent resource group prevents the storage account from being deleted. To do so:

1. From the Azure portal, at the top of the page, select **Home** to return to the start page.

2. Select **Storage accounts**. Then select **+ New**. The **Create storage account** page appears.
3. In the **Basics** tab, fill in the following fields.

   Note

   Replace **NNN** with a series of numbers. The numbers help to ensure that your storage account name is unique.

   Setting

   Value

   Project details

   Subscription

   Your Azure subscription

   Resource group

   my-test-rg

   Instance details

   Storage account name

   mysaNNN

   Location

   (US) East US

   Performance

   Standard

   Account kind

   StorageV2 (general purpose v2)

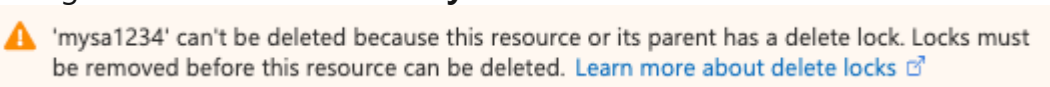
   Replication

   Locally redundant storage (LRS)

   As before, you can also select a region that's closer to you.

4. Select **Review + create**, and then select **Create**. The deployment might take a few moments to complete.
5. Select **Go to resource**.
6. At the top of the page, select **Delete**.

You see a message that tells you the resource or its parent is locked and can't be deleted. Here's an example that shows the error message for a storage account that's named **mysa1234**.



Although you didn't create a lock specifically for the storage account, the lock you created for the parent resource group prevents you from deleting the resource. In other words, the storage account inherits the lock from the parent resource group.

Delete the resource group and the storage account

You no longer need your resource group or storage account. Here you remove both.

When you delete a resource group, you also delete its child resources, such as the storage account you previously created.

To delete the resource group, you first need to remove the resource lock.

1. From the Azure portal, select **Home** > **Resource groups** > **my-test-rg** to go to your resource group.
2. Under **Settings**, select **Locks**.
3. Locate **rg-delete-lock**, and select **Delete** on that same row.
4. Select **Overview**, and then select **Delete resource group**.
5. At the prompt, enter **my-test-rg**, and then select **OK**. The deletion operation might take a few moments to complete.
6. When the operation completes, select **Home** > **Resource groups**. You see that the **my-test-rg** resource group no longer exists in your account. Your storage account is also deleted.

Nice work. You can now apply resource locks to help prevent the accidental deletion of your Azure resources.

Exercise - Restrict deployments to a specific location by using Azure Policy

Completed 100 XP

- 8 minutes

In this exercise, you create a policy in Azure Policy that restricts the deployment of Azure resources to a specific location. You verify the policy by attempting to create a storage account in a location that violates the policy.

Tailwind Traders wants to limit the location where resources can be deployed to the **East US** region. It has two reasons:

- **Improved cost tracking** To track costs, Tailwind Traders uses different subscriptions to track deployments to each of its regional locations. The policy will ensure that all resources are deployed to the **East US** region.
- **Adhere to data residency and security compliance** Tailwind Traders must adhere to a compliance rule that states where customer data can be stored. Here, customer data must be stored in the **East US** region.

Recall that you can assign a policy to a management group, a single subscription, or a resource group. Here, you assign the policy to a resource group so that policy doesn't affect any other resources in your Azure subscription.

Important

You need your own **Azure subscription** to complete the exercises in this module. If you don't have an Azure subscription, you can still read along.

Create the resource group

Here you create a resource group that's named **my-test-rg**. This is the resource group to which you'll apply your location policy.

For learning purposes, you use the same resource group name that you used in the previous exercise. You can use the same name because you deleted the previous resource group.

1. Go to the Azure portal, and sign in.
2. Select **Create a resource**.
3. Enter **resource group** in the search box, and press Enter.
4. If you're taken to a search results pane, select **Resource group** from the results.

5. Select **Create**. Then, enter the following values for each setting.

   Setting

   Value

   Subscription

   (Your Azure subscription)

   Subscription > Resource group

   my-test-rg

   Region

   (US) East US

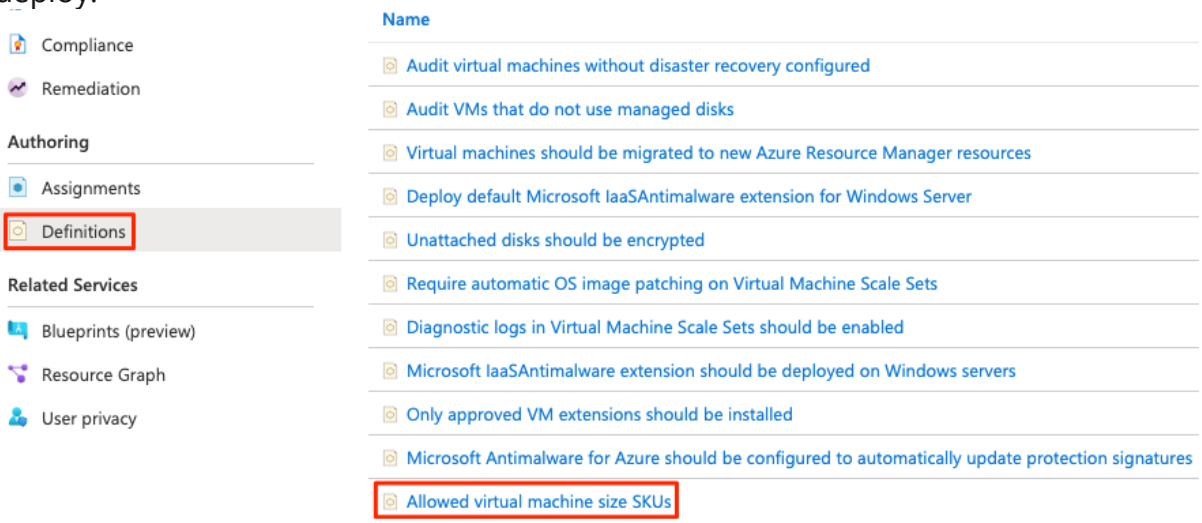
6. Select **Review + create**, and then select **Create**.

Explore predefined policies

Before you configure your location policy, let's take a brief look at some predefined policies. As an example, you'll look at policies that relate to Azure Compute services.

1. From the Azure portal, at the top of the page, select **Home** to return to the start page.
2. At the top of the page, enter **policy** in the search bar. Then, select **Policy** from the list of results to access Azure Policy.
3. Under **Authoring**, select **Definitions**.
4. From the **Category** dropdown list, select only **Compute**. Notice that the **Allowed virtual machine SKUs** definition enables you to specify a set of virtual machine SKUs that your organization can

deploy.

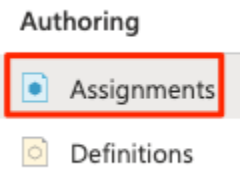


As an optional step, explore any other policies or categories that interest you.

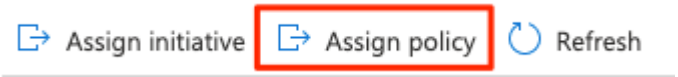
Configure the location policy

Here you configure the allowed location policy by using Azure Policy. Then you assign that policy to your resource group. To do so:

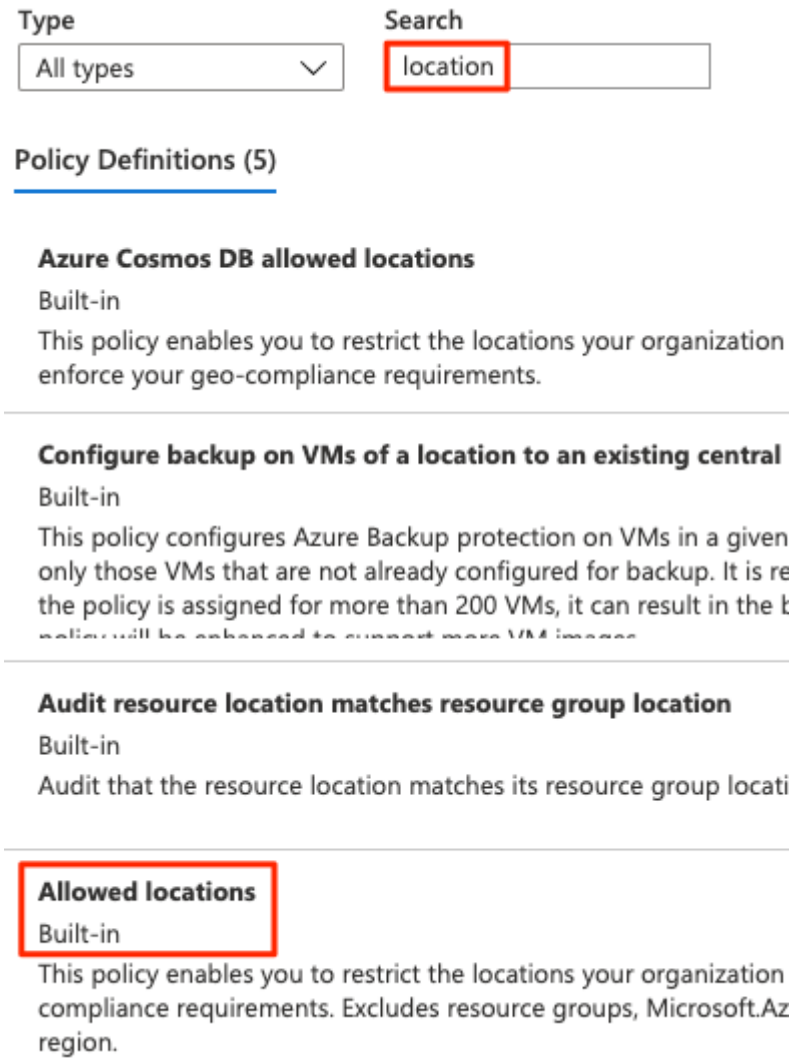
1. From the **Policy** pane, under **Authoring**, select **Assignments**.

   An assignment is a policy that has been assigned to take place within a specific scope. For example, a definition could be assigned to the subscription scope.
2. Select **Assign Policy**.

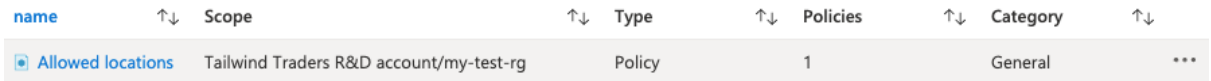
   

   You're taken to the **Assign policy** pane.
3. Under **Scope**, select the ellipsis.
   From the dialog box that appears, set:
   a. **Subscription** field to your Azure subscription.
   b. **Resource Group** field to **my-test-rg**.
   c. Select **Select**.
4. Under **Policy definition**, select the ellipsis.
   a. In the search bar, enter *location*.
   b. Select the **Allowed locations** definition.
   c. Select **Select**.



This policy definition specifies the location into which all resources must be deployed. If a different location is chosen, deployment will fail.

5. Select **Next** to move to the **Parameters** tab.
6. From the **Allowed locations** dropdown list, select **East US**.
7. Select **Review + create**, and then select **Create**.
   You see that the **Allowed locations** policy assignment is now listed on the **Policy | Assignments** pane. It enforces the policy on the **my-test-rg** resource group.



Verify the location policy

Here you attempt to add a storage account to your resource group at a location that violates your location policy.

1. From the Azure portal, at the top of the page, select **Home** to return to the start page.
2. Select **Create a resource**.
3. Enter **storage account** in the search box, and press Enter.
4. If you're taken to a search results pane, select **Storage account** from the results.
5. Select **Create**. Then, enter the following values for each setting.

   Note

   Replace **NNN** with a series of numbers. Numbers help to ensure that your storage account name is unique.

   Setting

   Value

   Subscription

   (Your Azure subscription)

   Subscription > Resource group

   my-test-rg

   Storage account name

   mysaNNN

   Location

   (Asia Pacific) Japan East

   Performance

   Standard

   Account kind

   StorageV2 (general purpose v2)

   Redundancy

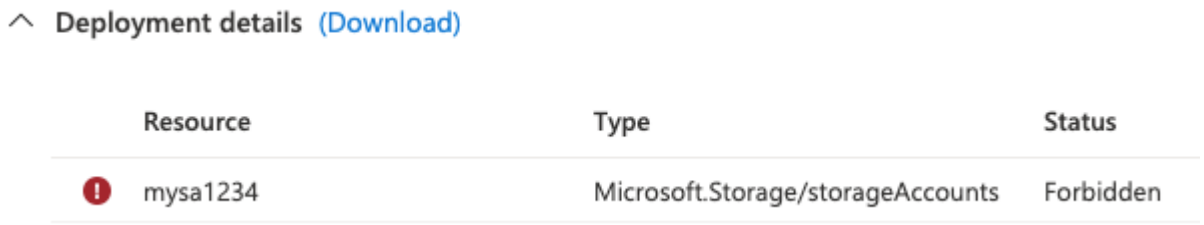
   Locally redundant storage (LRS)

   Access tier (default)

Hot

If you previously selected **Japan East** in your location policy, select a different region from the list.

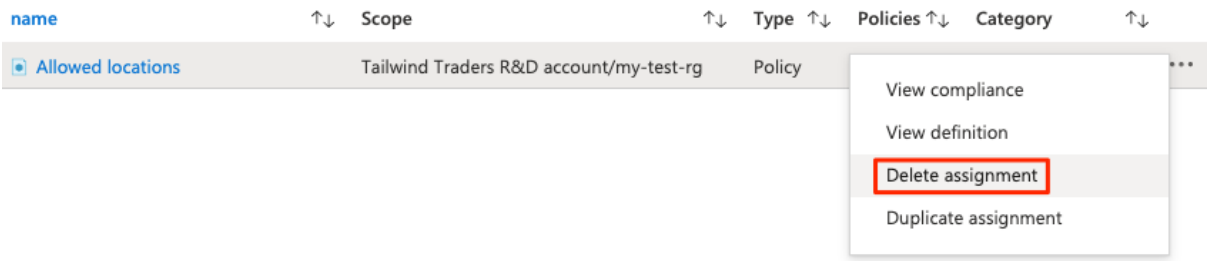
6. Select **Review + create**, and then select **Create**.
   You see a message that states that the deployment failed because of the policy violation. You also see the deployment details.
   Here's an example that shows the deployment details for a storage account named **mysa1234**.



Delete the policy assignment

You no longer need your policy assignment. Here you remove it from your subscription.

1. From the Azure portal, select **Home** > **Policy**.
2. Under **Authoring**, select **Assignments**.
3. On the **Allowed locations** row, select the ellipsis. Then, select **Delete assignment**. When prompted,
   select **Yes**.



   You see that the **Allowed locations** policy assignment no longer exists.

As an optional step, you can try to create the storage account a second time to verify that the policy is no longer in effect.

Delete the resource group

You no longer need your resource group. Here you remove it from your subscription.

1. From the Azure portal, select **Home** > **Resource groups** > **my-test-rg** to go to your resource group.
2. Select **Overview**, and then select **Delete resource group**.
3. At the prompt, enter **my-test-rg**, and then select **OK**. The deletion operation might take a few moments to complete.
4. After the operation completes, select **Home** > **Resource groups**. You see that the **my-test-rg** resource group no longer exists in your account.

Great work! You've successfully applied a policy by using Azure Policy to restrict deployments of Azure resources to specific locations. You can now apply the policies that you need at the management group, subscription, or resource group level.

Quickstart: Create a management group

- Article
- 12/14/2021
- 2 contributors

Feedback

In this article

Management groups are containers that help you manage access, policy, and compliance across multiple subscriptions. Create these containers to build an effective and efficient hierarchy that can be used with Azure Policy and Azure Role Based Access Controls. For more information on management groups, see Organize your resources with Azure management groups.

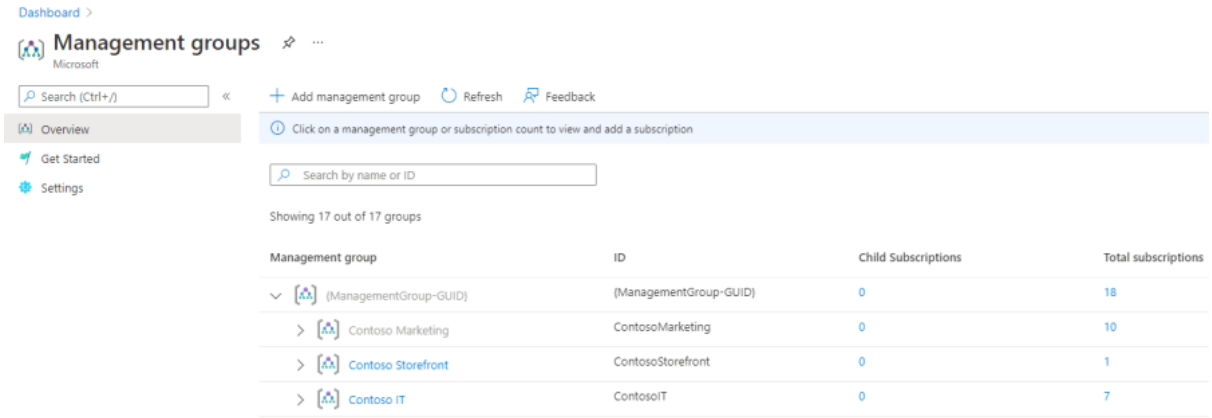
The first management group created in the directory could take up to 15 minutes to complete. There are processes that run the first time to set up the management groups service within Azure for your directory. You receive a notification when the process is complete. For more information, see initial setup of management groups.

Prerequisites

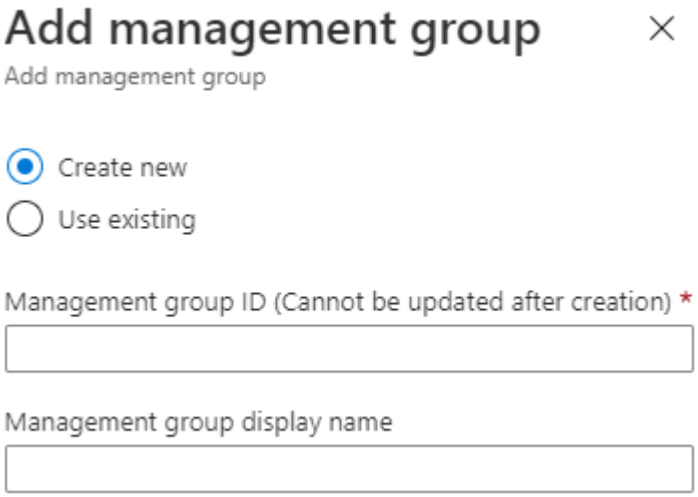
- If you don't have an Azure subscription, create a free account before you begin.
- Any Azure AD user in the tenant can create a management group without the management group write permission assigned to that user if hierarchy protection isn't enabled. This new management group becomes a child of the Root Management Group or the default management group and the creator is given an "Owner" role assignment. Management group service allows this ability so that role assignments aren't needed at the root level. No users have access to the Root Management Group when it's created. To avoid the hurdle of finding the Azure AD Global Admins to start using management groups, we allow the creation of the initial management groups at the root level.

Create in portal

1. Log into the Azure portal.
2. Select **All services** > **Management + governance**.
3. Select **Management Groups**.
4. Select **+ Add management group**.



5. Leave **Create new** selected and fill in the management group ID field.
    - The **Management Group ID** is the directory unique identifier that is used to submit commands on this management group. This identifier isn't editable after creation as it's used throughout the Azure system to identify this group. The root management group is automatically created with an ID that is the Azure Active Directory ID. For all other management groups, assign a unique ID.
    - The display name field is the name that is displayed within the Azure portal. A separate display name is an optional field when creating the management group and can be changed at any time.



6. Select **Save**.

Clean up resources

To remove the management group created, follow these steps:

1. Select **All services** > **Management + governance**.
2. Select **Management Groups**.
3. Find the management group created above, select it, then select **Details** next to the name. Then select **Delete** and confirm the prompt.

Quickstart: Create a management group with the Azure CLI

- Article
- 04/13/2023
- 4 contributors

Feedback

In this article

Management groups are containers that help you manage access, policy, and compliance across multiple subscriptions. Create these containers to build an effective and efficient hierarchy that can be used with Azure Policy and Azure Role Based Access Controls. For more information on management groups, see Organize your resources with Azure management groups.

The first management group created in the directory could take up to 15 minutes to complete. There are processes that run the first time to set up the management groups service within Azure for your directory. You receive a notification when the process is complete. For more information, see initial setup of management groups.

Prerequisites

- If you don't have an Azure subscription, create a free account before you begin.
- This quickstart requires that you run Azure CLI version 2.0.76 or later to install and use the CLI locally. To find the version, run `az --version`. If you need to install or upgrade, see Install Azure CLI.
- Any Azure AD user in the tenant can create a management group without the management group write permission assigned to that user if hierarchy protection isn't enabled. This new management group becomes a child of the Root Management Group or the default management group and the creator is given an "Owner" role assignment. Management group service allows this ability so that role assignments aren't needed at the root level. No users have access to the Root Management Group when it's created. To avoid the hurdle of finding the Azure AD Global Admins to start using management groups, we allow the creation of the initial management groups at the root level.

Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article, without having to install anything on your local environment.

To start Azure Cloud Shell:

Option	Example/Link
Select **Try It** in the upper-right corner of a code or command block. Selecting **Try It** doesn't automatically	

Option	Example/Link
copy the code or command to Cloud Shell.	
Go to https://shell.azure.com, or select the **Launch Cloud Shell** button to open Cloud Shell in your browser.	
Select the **Cloud Shell** button on the menu bar at the upper right in the Azure portal.	

To use Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block (or command block) to copy the code or command.
3. Paste the code or command into the Cloud Shell session by selecting **Ctrl**+**Shift**+**V** on Windows and Linux, or by selecting **Cmd**+**Shift**+**V** on macOS.
4. Select **Enter** to run the code or command.

Create in the Azure CLI

For Azure CLI, use the az account management-group create command to create a new management group. In this example, the management group **name** is *Contoso*.

Azure CLICopy
Open Cloudshell

```
az account management-group create --name 'Contoso'
```

The **name** is a unique identifier being created. This ID is used by other commands to reference this group and it can't be changed later.

If you want the management group to show a different name within the Azure portal, add the **display-name** parameter. For example, to create a management group with the GroupName of Contoso and the display name of "Contoso Group", use the following command:

Azure CLICopy
Open Cloudshell

```
az account management-group create --name 'Contoso' --display-name 'Contoso Group'
```

In the preceding examples, the new management group is created under the root management group. To specify a different management group as the parent, use the **parent** parameter and provide the name of the parent group.

Azure CLICopy
Open Cloudshell

```
az account management-group create --name 'ContosoSubGroup' --parent 'Contoso'
```

Clean up resources

To remove the management group created above, use the az account management-group delete command:

Azure CLICopy
Open Cloudshell

```
az account management-group delete --name 'Contoso'
```

Next steps

In this quickstart, you created a management group to organize your resource hierarchy. The management group can hold subscriptions or other management groups.

Quickstart: Create a management group with Azure PowerShell

- Article
- 05/11/2023
- 3 contributors

Feedback

In this article

Management groups are containers that help you manage access, policy, and compliance across multiple subscriptions. Create these containers to build an effective and efficient hierarchy that can be used with Azure Policy and Azure Role Based Access Controls. For more information on management groups, see Organize your resources with Azure management groups.

The first management group created in the directory could take up to 15 minutes to complete. There are processes that run the first time to set up the management groups service within Azure for your directory. You receive a notification when the process is complete. For more information, see initial setup of management groups.

Prerequisites

- If you don't have an Azure subscription, create a free account before you begin.
- Before you start, make sure that the latest version of Azure PowerShell is installed. See Install Azure PowerShell module for detailed information.
- Any Azure AD user in the tenant can create a management group without the management group write permission assigned to that user if hierarchy protection isn't enabled. This new management group becomes a child of the Root Management Group or the default management group and the creator is given an "Owner" role assignment. Management group service allows this ability so that role assignments aren't needed at the root level. No users have access to the Root Management Group when it's created. To avoid the hurdle of finding the Azure AD Global Admins to start using management groups, we allow the creation of the initial management groups at the root level.

Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article, without having to install anything on your local environment.

To start Azure Cloud Shell:

Option	Example/Link
Select **Try It** in the upper-right corner of a code or command block. Selecting **Try It** doesn't automatically copy the code or command to Cloud Shell.	
Go to https://shell.azure.com, or select the **Launch Cloud Shell** button to open Cloud Shell in your browser.	
Select the **Cloud Shell** button on the menu bar at the upper right in the Azure portal.	

To use Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block (or command block) to copy the code or command.
3. Paste the code or command into the Cloud Shell session by selecting **Ctrl**+**Shift**+**V** on Windows and Linux, or by selecting **Cmd**+**Shift**+**V** on macOS.
4. Select **Enter** to run the code or command.

Create in Azure PowerShell

For PowerShell, use the New-AzManagementGroup cmdlet to create a new management group. In this example, the management group **GroupName** is *Contoso*.

Azure PowerShellCopy
Open Cloudshell

```
New-AzManagementGroup -GroupName 'Contoso'
```

The **GroupName** is a unique identifier being created. This ID is used by other commands to reference this group and it can't be changed later.

If you want the management group to show a different name within the Azure portal, add the **DisplayName** parameter. For example, to create a management group with the GroupName of Contoso and the display name of "Contoso Group", use the following cmdlet:

Azure PowerShellCopy
Open Cloudshell

```
New-AzManagementGroup -GroupName 'Contoso' -DisplayName 'Contoso Group'
```

In the preceding examples, the new management group is created under the root management group. To specify a different management group as the parent, use the **ParentId** parameter.

Azure PowerShellCopy
Open Cloudshell

```
$parentGroup = Get-AzManagementGroup -GroupName Contoso
New-AzManagementGroup -GroupName 'ContosoSubGroup' -ParentId $parentGroup.id
```

Clean up resources

To remove the management group created above, use the Remove-AzManagementGroup cmdlet:

Azure PowerShellCopy
Open Cloudshell

```
Remove-AzManagementGroup -GroupName 'Contoso'
```

Next steps

In this quickstart, you created a management group to organize your resource hierarchy. The management group can hold subscriptions or other management groups.

Quickstart: Create a management group with .NET Core

- Article
- 08/18/2021
- 2 contributors

Feedback

In this article

Management groups are containers that help you manage access, policy, and compliance across multiple subscriptions. Create these containers to build an effective and efficient hierarchy that can be used with Azure Policy and Azure Role Based Access Controls. For more information on management groups, see Organize your resources with Azure management groups.

The first management group created in the directory could take up to 15 minutes to complete. There are processes that run the first time to set up the management groups service within Azure for your directory. You receive a notification when the process is complete. For more information, see initial setup of management groups.

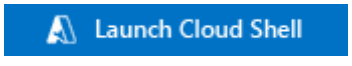
Prerequisites

- If you don't have an Azure subscription, create a free account before you begin.
- An Azure service principal, including the *clientId* and *clientSecret*. If you don't have a service principal for use with Azure Policy or want to create a new one, see Azure management libraries for .NET authentication. Skip the step to install the .NET Core packages as we'll do that in the next steps.
- Any Azure AD user in the tenant can create a management group without the management group write permission assigned to that user if hierarchy protection isn't enabled. This new management group becomes a child of the Root Management Group or the default management group and the creator is given an "Owner" role assignment. Management group service allows this ability so that role assignments aren't needed at the root level. No users have access to the Root Management Group when it's created. To avoid the hurdle of finding the Azure AD Global Admins to start using management groups, we allow the creation of the initial management groups at the root level.

Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article, without having to install anything on your local environment.

To start Azure Cloud Shell:

Option	Example/Link
Select **Try It** in the upper-right corner of a code or command block. Selecting **Try It** doesn't automatically copy the code or command to Cloud Shell.	
Go to https://shell.azure.com, or select the **Launch Cloud Shell** button to open Cloud Shell in your browser.	
Select the **Cloud Shell** button on the menu bar at the upper right in the Azure portal.	

To use Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block (or command block) to copy the code or command.
3. Paste the code or command into the Cloud Shell session by selecting **Ctrl**+**Shift**+**V** on Windows and Linux, or by selecting **Cmd**+**Shift**+**V** on macOS.
4. Select **Enter** to run the code or command.

Application setup

To enable .NET Core to manage management groups, create a new console application and install the required packages.

1. Check that the latest .NET Core is installed (at least **3.1.8**). If it isn't yet installed, download it at dotnet.microsoft.com.
2. Initialize a new .NET Core console application named "mgCreate":

   .NET CLICopy
   ```
   dotnet new console --name "mgCreate"
   ```

3. Change directories into the new project folder and install the required packages for Azure Policy:

   .NET CLICopy
   ```
   # Add the Azure Policy package for .NET Core
   dotnet add package Microsoft.Azure.Management.ManagementGroups --
   version 1.1.1-preview

   # Add the Azure app auth package for .NET Core
   dotnet add package Microsoft.Azure.Services.AppAuthentication --
   version 1.6.1
   ```

4. Replace the default `program.cs` with the following code and save the updated file:

   C#Copy
   ```
   using System;
   using System.Collections.Generic;
   using System.Threading.Tasks;
   using Microsoft.IdentityModel.Clients.ActiveDirectory;
   using Microsoft.Rest;
   using Microsoft.Azure.Management.ManagementGroups;
   using Microsoft.Azure.Management.ManagementGroups.Models;

   namespace mgCreate
   {
       class Program
       {
           static async Task Main(string[] args)
   ```

```
        {
            string strTenant = args[0];
            string strClientId = args[1];
            string strClientSecret = args[2];
            string strGroupId = args[3];
            string strDisplayName = args[4];

            var authContext = new
AuthenticationContext($"https://login.microsoftonline.com/{strTenant}"
);
            var authResult = await authContext.AcquireTokenAsync(
                   "https://management.core.windows.net",
                   new ClientCredential(strClientId,
strClientSecret));

            using (var client = new ManagementGroupsAPIClient(new
TokenCredentials(authResult.AccessToken)))
            {
                var mgRequest = new CreateManagementGroupRequest
                {
                    DisplayName = strDisplayName
                };
                var response = await
client.ManagementGroups.CreateOrUpdateAsync(strGroupId, mgRequest);
            }
        }
    }
}
```

5. Build and publish the `mgCreate` console application:

   .NET CLICopy
   ```
   dotnet build
   dotnet publish -o {run-folder}
   ```

Create the management group

In this quickstart, you create a new management group in the root management group.

1. Change directories to the `{run-folder}` you defined with the previous `dotnet publish` command.
2. Enter the following command in the terminal:

   BashCopy
   ```
   mgCreate.exe `
      "{tenantId}" `
      "{clientId}" `
      "{clientSecret}" `
      "{groupID}" `
      "{displayName}"
   ```

The preceding commands use the following information:

- `{tenantId}` - Replace with your tenant ID
- `{clientId}` - Replace with the client ID of your service principal
- `{clientSecret}` - Replace with the client secret of your service principal
- `{groupID}` - Replace with the ID for your new management group
- `{displayName}` - Replace with the friendly name for your new management group

The result is a new management group in the root management group.

Clean up resources

- Delete the new management group through the portal.
- If you wish to remove the .NET Core console application and installed packages, delete the `mgCreate` project folder.

Next steps

In this quickstart, you created a management group to organize your resource hierarchy. The management group can hold subscriptions or other management groups.

Quickstart: Create a management group with Go

- Article
- 04/19/2023
- 5 contributors

Feedback

In this article

Management groups are containers that help you manage access, policy, and compliance across multiple subscriptions. Create these containers to build an

effective and efficient hierarchy that can be used with Azure Policy and Azure Role Based Access Controls. For more information on management groups, see Organize your resources with Azure management groups.

The first management group created in the directory could take up to 15 minutes to complete. There are processes that run the first time to set up the management groups service within Azure for your directory. You receive a notification when the process is complete. For more information, see initial setup of management groups.

Prerequisites

- If you don't have an Azure subscription, create a free account before you begin.
- An Azure service principal, including the *clientId* and *clientSecret*. If you don't have a service principal for use with Azure Policy or want to create a new one, see Azure management libraries for .NET authentication. Skip the step to install the .NET Core packages as we'll do that in the next steps.
- Any Azure AD user in the tenant can create a management group without the management group write permission assigned to that user if hierarchy protection isn't enabled. This new management group becomes a child of the Root Management Group or the default management group and the creator is given an "Owner" role assignment. Management group service allows this ability so that role assignments aren't needed at the root level. No users have access to the Root Management Group when it's created. To avoid the hurdle of finding the Azure AD Global Admins to start using management groups, we allow the creation of the initial management groups at the root level.

Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article, without having to install anything on your local environment.

To start Azure Cloud Shell:

Option	Example/Link
Select **Try It** in the upper-right corner of a code or command block. Selecting **Try It** doesn't automatically	

Option	Example/Link
copy the code or command to Cloud Shell.	
Go to https://shell.azure.com, or select the **Launch Cloud Shell** button to open Cloud Shell in your browser.	
Select the **Cloud Shell** button on the menu bar at the upper right in the Azure portal.	

To use Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block (or command block) to copy the code or command.
3. Paste the code or command into the Cloud Shell session by selecting **Ctrl**+**Shift**+**V** on Windows and Linux, or by selecting **Cmd**+**Shift**+**V** on macOS.
4. Select **Enter** to run the code or command.

Add the management group package

To enable Go to manage management groups, the package must be added. This package works wherever Go can be used, including bash on Windows 10 or locally installed.

1. Check that the latest Go is installed (at least **1.15**). If it isn't yet installed, download it at Golang.org.
2. Check that the latest Azure CLI is installed (at least **2.5.1**). If it isn't yet installed, see Install the Azure CLI.

   Note

   Azure CLI is required to enable Go to use the `auth.NewAuthorizerFromCLI()` method in the following example. For information about other options, see **Azure SDK for Go - More authentication details**.

3. Authenticate through Azure CLI.

   Azure CLICopy

   ```
   az login
   ```

4. In your Go environment of choice, install the required packages for management groups:

   BashCopy
   ```
   # Add the management group package for Go
   go install github.com/Azure/azure-sdk-for-
   go/services/resources/mgmt/2020-05-01/managementgroups@latest

   # Add the Azure auth package for Go
   go install github.com/Azure/go-autorest/autorest/azure/auth@latest
   ```

Application setup

With the Go packages added to your environment of choice, it's time to set up the Go application that can create a management group.

1. Create the Go application and save the following source as `mgCreate.go`:

   GoCopy
   ```
   package main

   import (
    "context"
    "fmt"
    "os"

    mg "github.com/Azure/azure-sdk-for-go/services/resources/mgmt/2020-
   05-01/managementgroups"
    "github.com/Azure/go-autorest/autorest/azure/auth"
   )

   func main() {
    // Get variables from command line arguments
    var mgName = os.Args[1]

    // Create and authorize a client
    mgClient := mg.NewClient()
    authorizer, err := auth.NewAuthorizerFromCLI()
    if err == nil {
        mgClient.Authorizer = authorizer
    } else {
        fmt.Printf(err.Error())
    }

    // Create the request
    Request := mg.CreateManagementGroupRequest{
        Name: &mgName,
    }

    // Run the query and get the results
    var results, queryErr = mgClient.CreateOrUpdate(context.Background(),
   mgName, Request, "no-cache")
    if queryErr == nil {
        fmt.Printf("Results: " + fmt.Sprint(results) + "\n")
   ```

```
 } else {
     fmt.Printf(queryErr.Error())
 }
}
```

2. Build the Go application:

   BashCopy
   ```
   go build mgCreate.go
   ```

3. Create a management group using the compiled Go application. Replace `<Name>` with the name of your new management group:

   BashCopy
   ```
   mgCreate "<Name>"
   ```

The result is a new management group in the root management group.

Clean up resources

If you wish to remove the installed packages from your Go environment, you can do so by using the following command:

BashCopy
```
# Remove the installed packages from the Go environment
go clean -i github.com/Azure/azure-sdk-for-go/services/resources/mgmt/2020-05-01/managementgroups
go clean -i github.com/Azure/go-autorest/autorest/azure/auth
```

Next steps

In this quickstart, you created a management group to organize your resource hierarchy. The management group can hold subscriptions or other management groups.

Quickstart: Create a management group with JavaScript

- Article
- 06/20/2022
- 4 contributors

Feedback

In this article

Management groups are containers that help you manage access, policy, and compliance across multiple subscriptions. Create these containers to build an effective and efficient hierarchy that can be used with Azure Policy and Azure Role Based Access Controls. For more information on management groups, see Organize your resources with Azure management groups.

The first management group created in the directory could take up to 15 minutes to complete. There are processes that run the first time to set up the management groups service within Azure for your directory. You receive a notification when the process is complete. For more information, see initial setup of management groups.

Prerequisites

- If you don't have an Azure subscription, create a free account before you begin.
- Before you start, make sure that at least version 12 of Node.js is installed.
- Any Azure AD user in the tenant can create a management group without the management group write permission assigned to that user if hierarchy protection isn't enabled. This new management group becomes a child of the Root Management Group or the default management group and the creator is given an "Owner" role assignment. Management group service allows this ability so that role assignments aren't needed at the root level. No users have access to the Root Management Group when it's created. To avoid the hurdle of finding the Azure AD Global Admins to start using management groups, we allow the creation of the initial management groups at the root level.

Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article, without having to install anything on your local environment.

To start Azure Cloud Shell:

Option	Example/Link
Select **Try It** in the upper-right corner of a code or command block. Selecting **Try It** doesn't automatically copy the code or command to Cloud Shell.	
Go to https://shell.azure.com, or select the **Launch Cloud Shell** button to open Cloud Shell in your browser.	
Select the **Cloud Shell** button on the menu bar at the upper right in the Azure portal.	

To use Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block (or command block) to copy the code or command.
3. Paste the code or command into the Cloud Shell session by selecting **Ctrl**+**Shift**+**V** on Windows and Linux, or by selecting **Cmd**+**Shift**+**V** on macOS.
4. Select **Enter** to run the code or command.

Application setup

To enable JavaScript to manage management groups, the environment must be set up. This setup works wherever JavaScript can be used, including bash on Windows 10.

1. Set up a new Node.js project by running the following command.

   BashCopy
   ```
   npm init -y
   ```

2. Add a reference to the yargs module.

   BashCopy
   ```
   npm install yargs
   ```

3. Add a reference to the Azure Resource Graph module.

   BashCopy
   ```
   npm install @azure/arm-managementgroups
   ```

4. Add a reference to the Azure authentication library.

   BashCopy
   ```
   npm install @azure/identity
   ```
   Note

   Verify in *package.json* `@azure/arm-managementgroups` is version **2.0.1** or higher and `@azure/identity` is version **2.0.4** or higher.

Create the management group

1. Create a new file named *index.js* and enter the following code.

   JavaScriptCopy
   ```
      const argv = require("yargs").argv;
      const { InteractiveBrowserCredential } =
   require("@azure/identity");
      const { ManagementGroupsAPI } = require("@azure/arm-
   managementgroups");

      if (argv.groupID && argv.displayName) {
         const createMG = async () => {
            const credentials = new InteractiveBrowserCredential();
            const client = new ManagementGroupsAPI(credentials);
            const result = await
   client.managementGroups.beginCreateOrUpdateAndWait(
               argv.groupID,
               {
                  displayName: argv.displayName
               }
            );
            console.log(result);
         };

      createMG();
   }
   ```

2. Enter the following command in the terminal:

   BashCopy
   ```
   node index.js --groupID "<NEW_MG_GROUP_ID>" --displayName
   "<NEW_MG_FRIENDLY_NAME>"
   ```

   Make sure to replace each token <> placeholder with your *management group ID* and *management group friendly name*, respectively.

   As the script attempts to authenticate, a message similar to the following message is displayed in the terminal:

To sign in, use a web browser to open the page https://microsoft.com/devicelogin and enter the code FGB56WJUGK to authenticate.

Once you authenticate in the browser, then the script continues to run.

The result of creating the management group is output to the console.

Clean up resources

If you wish to remove the installed libraries from your application, run the following command.

BashCopy

```
npm uninstall @azure/arm-managementgroups @azure/identity yargs
```

Next steps

In this quickstart, you created a management group to organize your resource hierarchy. The management group can hold subscriptions or other management groups.

Quickstart: Create a management group with Python

- Article
- 04/29/2022
- 3 contributors

Feedback

In this article

Management groups are containers that help you manage access, policy, and compliance across multiple subscriptions. Create these containers to build an effective and efficient hierarchy that can be used with Azure Policy and Azure Role Based Access Controls. For more information on management groups, see Organize your resources with Azure management groups.

The first management group created in the directory could take up to 15 minutes to complete. There are processes that run the first time to set up the management groups service within Azure for your directory. You receive a notification when the process is complete. For more information, see initial setup of management groups.

Prerequisites

- If you don't have an Azure subscription, create a free account before you begin.
- Any Azure AD user in the tenant can create a management group without the management group write permission assigned to that user if hierarchy protection isn't enabled. This new management group becomes a child of the Root Management Group or the default management group and the creator is given an "Owner" role assignment. Management group service allows this ability so that role assignments aren't needed at the root level. No users have access to the Root Management Group when it's created. To avoid the hurdle of finding the Azure AD Global Admins to start using management groups, we allow the creation of the initial management groups at the root level.

Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article, without having to install anything on your local environment.

To start Azure Cloud Shell:

Option	Example/Link
Select **Try It** in the upper-right corner of a code or command block. Selecting **Try It** doesn't automatically copy the code or command to Cloud Shell.	

Option	Example/Link
Go to https://shell.azure.com, or select the **Launch Cloud Shell** button to open Cloud Shell in your browser.	
Select the **Cloud Shell** button on the menu bar at the upper right in the Azure portal.	

To use Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block (or command block) to copy the code or command.
3. Paste the code or command into the Cloud Shell session by selecting **Ctrl**+**Shift**+**V** on Windows and Linux, or by selecting **Cmd**+**Shift**+**V** on macOS.
4. Select **Enter** to run the code or command.

Add the Resource Graph library

To enable Python to manage management groups, the library must be added. This library works wherever Python can be used, including bash on Windows 10 or locally installed.

1. Check that the latest Python is installed (at least **3.8**). If it isn't yet installed, download it at Python.org.
2. Check that the latest Azure CLI is installed (at least **2.5.1**). If it isn't yet installed, see Install the Azure CLI.

   Note

   Azure CLI is required to enable Python to use the **CLI-based authentication** in the following examples. For information about other options, see **Authenticate using the Azure management libraries for Python**.

3. Authenticate through Azure CLI.

   Azure CLICopy
   ```
   az login
   ```

4. In your Python environment of choice, install the required libraries for management groups:

BashCopy

```
# Add the management groups library for Python
pip install azure-mgmt-managementgroups

# Add the Resources library for Python
pip install azure-mgmt-resource

# Add the CLI Core library for Python for authentication (development
only!)
pip install azure-cli-core
```

Note

If Python is installed for all users, these commands must be run from an elevated console.

5. Validate that the libraries have been installed. `azure-mgmt-managementgroups` should be **0.2.0** or higher, `azure-mgmt-resource` should be **9.0.0** or higher, and `azure-cli-core` should be **2.5.0** or higher.

   BashCopy

   ```
   # Check each installed library
   pip show azure-mgmt-managementgroups azure-mgmt-resource azure-cli-
   core
   ```

Create the management group

1. Create the Python script and save the following source as `mgCreate.py`:

   PythonCopy

   ```
   # Import management group classes
   from azure.mgmt.managementgroups import ManagementGroupsAPI

   # Import specific methods and models from other libraries
   from azure.common.credentials import get_azure_cli_credentials
   from azure.common.client_factory import get_client_from_cli_profile
   from azure.mgmt.resource import ResourceManagementClient,
   SubscriptionClient

   # Wrap all the work in a function
   def createmanagementgroup( strName ):
       # Get your credentials from Azure CLI (development only!) and get
   your subscription list
       subsClient = get_client_from_cli_profile(SubscriptionClient)
       subsRaw = []
       for sub in subsClient.subscriptions.list():
           subsRaw.append(sub.as_dict())
       subsList = []
       for sub in subsRaw:
           subsList.append(sub.get('subscription_id'))

       # Create management group client and set options
       mgClient = get_client_from_cli_profile(ManagementGroupsAPI)
   ```

```
        mg_request = {'name': strName, 'display_name': strName}

        # Create management group
        mg =
    mgClient.management_groups.create_or_update(group_id=strName,create_ma
    nagement_group_request=mg_request)

        # Show results
        print(mg)

    createmanagementgroup("MyNewMG")
```

2. Authenticate with Azure CLI with `az login`.
3. Enter the following command in the terminal:

   BashCopy
   ```
   py mgCreate.py
   ```

The result of creating the management group is output to the console as an `LROPoller` object.

Clean up resources

If you wish to remove the installed libraries from your Python environment, you can do so by using the following command:

BashCopy
```
# Remove the installed libraries from the Python environment
pip uninstall azure-mgmt-managementgroups azure-mgmt-resource azure-cli-core
```

Next steps

In this quickstart, you created a management group to organize your resource hierarchy. The management group can hold subscriptions or other management groups.

Quickstart: Create a management group with REST API

- Article
- 08/18/2021
- 3 contributors

Feedback

In this article

Management groups are containers that help you manage access, policy, and compliance across multiple subscriptions. Create these containers to build an effective and efficient hierarchy that can be used with Azure Policy and Azure Role Based Access Controls. For more information on management groups, see Organize your resources with Azure management groups.

The first management group created in the directory could take up to 15 minutes to complete. There are processes that run the first time to set up the management groups service within Azure for your directory. You receive a notification when the process is complete. For more information, see initial setup of management groups.

Prerequisites

- If you don't have an Azure subscription, create a free account before you begin.
- If you haven't already, install ARMClient. It's a tool that sends HTTP requests to Azure Resource Manager-based REST APIs. Instead, you can use the "Try It" feature in REST documentation or tooling like PowerShell's Invoke-RestMethod or Postman.
- Any Azure AD user in the tenant can create a management group without the management group write permission assigned to that user if hierarchy protection isn't enabled. This new management group becomes a child of the Root Management Group or the default management group and the creator is given an "Owner" role assignment. Management group service allows this ability so that role assignments aren't needed at the root level. No users have access to the Root Management Group when it's created. To avoid the hurdle of finding the Azure AD Global Admins to start using management groups, we allow the creation of the initial management groups at the root level.

Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to

work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article, without having to install anything on your local environment.

To start Azure Cloud Shell:

Option	Example/Link
Select **Try It** in the upper-right corner of a code or command block. Selecting **Try It** doesn't automatically copy the code or command to Cloud Shell.	
Go to https://shell.azure.com, or select the **Launch Cloud Shell** button to open Cloud Shell in your browser.	
Select the **Cloud Shell** button on the menu bar at the upper right in the Azure portal.	

To use Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block (or command block) to copy the code or command.
3. Paste the code or command into the Cloud Shell session by selecting **Ctrl**+**Shift**+**V** on Windows and Linux, or by selecting **Cmd**+**Shift**+**V** on macOS.
4. Select **Enter** to run the code or command.

Create in REST API

For REST API, use the Management Groups - Create or Update endpoint to create a new management group. In this example, the management group **groupId** is *Contoso*.

- REST API URI

  HTTPCopy
  ```
  PUT
  https://management.azure.com/providers/Microsoft.Management/managementGroups/Contoso?api-version=2020-05-01
  ```

- No Request Body

The **groupId** is a unique identifier being created. This ID is used by other commands to reference this group and it can't be changed later.

If you want the management group to show a different name within the Azure portal, add the **properties.displayName** property in the request body. For example, to create a management group with the **groupId** of *Contoso* and the display name of *Contoso Group*, use the following endpoint and request body:

- REST API URI

  HTTPCopy
  ```
  PUT
  https://management.azure.com/providers/Microsoft.Management/managementGroups/Contoso?api-version=2020-05-01
  ```

- Request Body

  JSONCopy
  ```
  {
    "properties": {
      "displayName": "Contoso Group"
    }
  }
  ```

In the preceding examples, the new management group is created under the root management group. To specify a different management group as the parent, use the **properties.parent.id** property.

- REST API URI

  HTTPCopy
  ```
  PUT
  https://management.azure.com/providers/Microsoft.Management/managementGroups/Contoso?api-version=2020-05-01
  ```

- Request Body

  JSONCopy
  ```
  {
    "properties": {
      "displayName": "Contoso Group",
      "parent": {
        "id":
  "/providers/Microsoft.Management/managementGroups/HoldingGroup"
      }
    }
  }
  ```

Clean up resources

To remove the management group created above, use the Management Groups - Delete endpoint:

- REST API URI

  HTTPCopy
  ```
  DELETE
  https://management.azure.com/providers/Microsoft.Management/managementGroups/Contoso?api-version=2020-05-01
  ```

- No Request Body

Next steps

In this quickstart, you created a management group to organize your resource hierarchy. The management group can hold subscriptions or other management groups.

Exercise - Create an Azure SQL Database

Tailwind Traders has chosen Azure SQL Database for part of its migration. You've been tasked with creating the database.

In this exercise, you'll create a SQL database in Azure and then query the data in that database.

Task 1: Create the database

In this task, you create a SQL database based on the *AdventureWorksLT* sample database.

Task 1: Create the database

In this task, you create a SQL database based on the *AdventureWorksLT* sample database.

1. Sign in to the Azure portal.

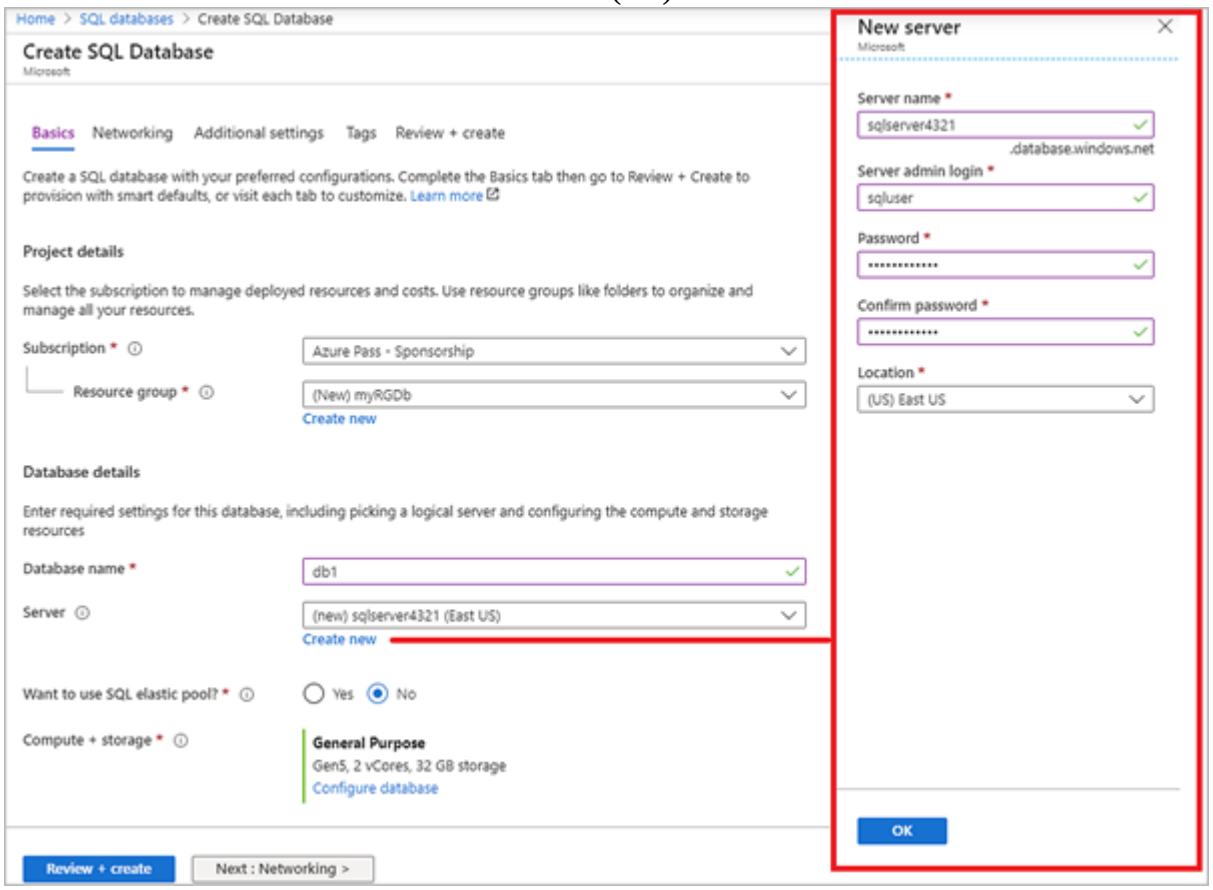
2. Select **Create a resource > Databases > SQL database**.

3. On the **Basics** tab, fill in the following information:

Setting	**Value**
Subscription	Choose Concierge Subscription
Resource Group	Choose [sandbox resource group name]
Database name	db1

4. For the **Server**, select **Create new**. i. Enter the following information (replace **nnnn** in the name of the server with letters and digits, such that the name is globally unique)

Setting	Value
Server Name	sqlserver**nnn** (must be unique)
Server admin login	sqluser
Password	Pa$$w0rd1234
Location	(US) East US



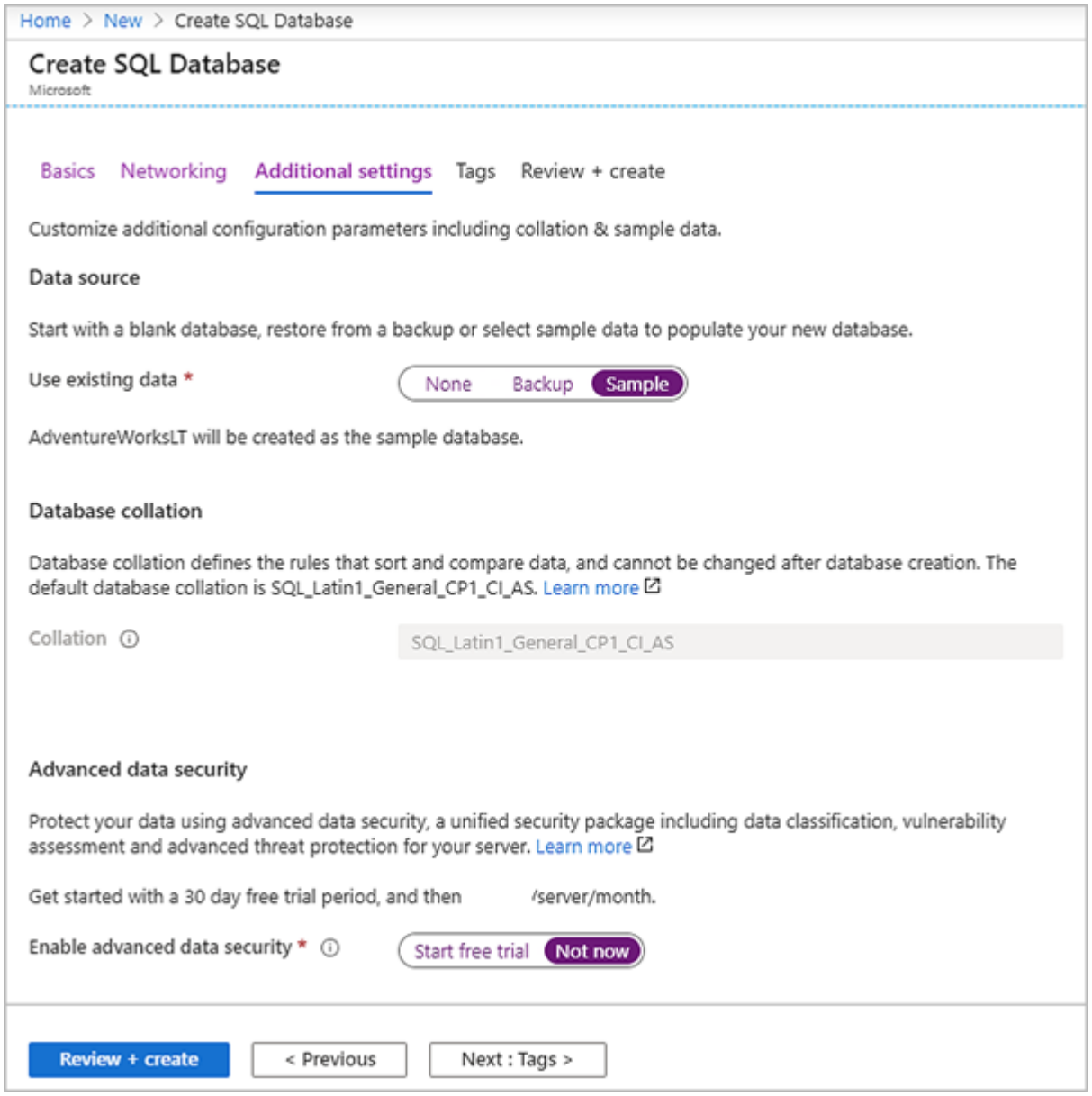
ii. Select OK when you have finished.

5. Select the **Next: Networking >** at the bottom, and configure the following settings (leave others with their defaults):

Setting	Value
Server Name	sqlserver**nnn** (must be unique)
Server admin login	sqluser
Password	Pa$$w0rd1234
Location	(US) East US

6. Select the **Next: Networking >** at the bottom, and configure the following settings (leave others with their defaults):

Setting	Value
Connectivity Method	Public Endpoint (Default)



7. Select **Review + create > Create** to deploy the server and database. i. It can take approximately 2 to 5 minutes to create the server and deploy the sample database.

8. Select **Go to resource**.

9. Select **Set server firewall** and Allow Azure services and resources to access this server = **Yes**.

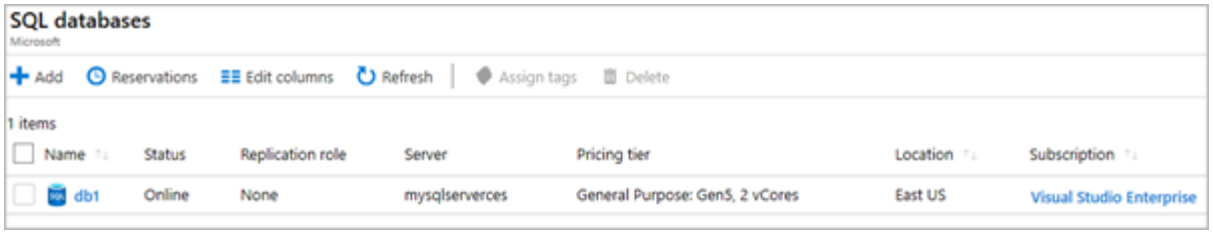
10. Select **Save.**

11. Select **OK.**

Task 2: Test the database

In this task, you configure the server and run a SQL query.

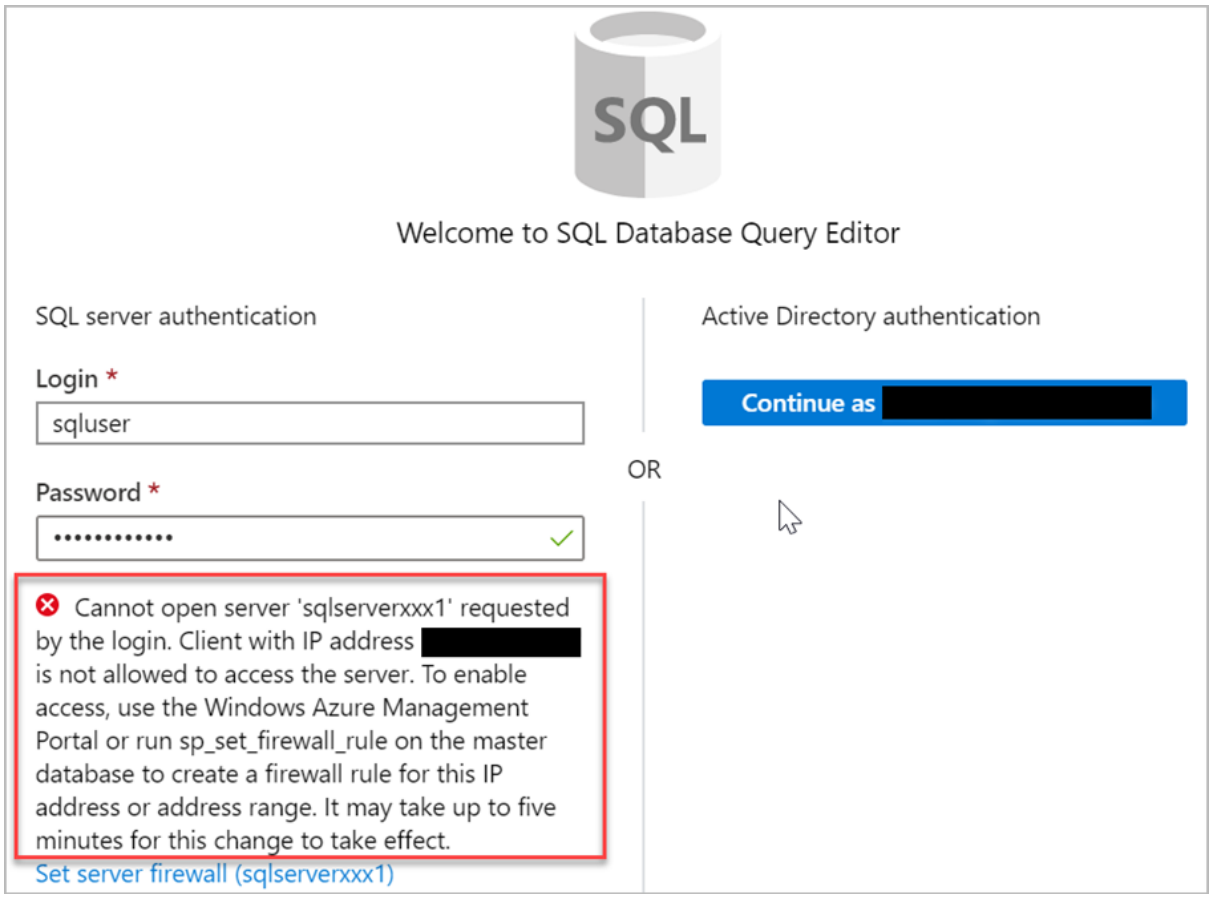
1. From the All resources pan, search and select SQL databases and ensure that your new database was created. You might need to refresh the page.



2. Select the **db1** entry representing the SQL database you created, and then select **Query editor (preview)** on the left side

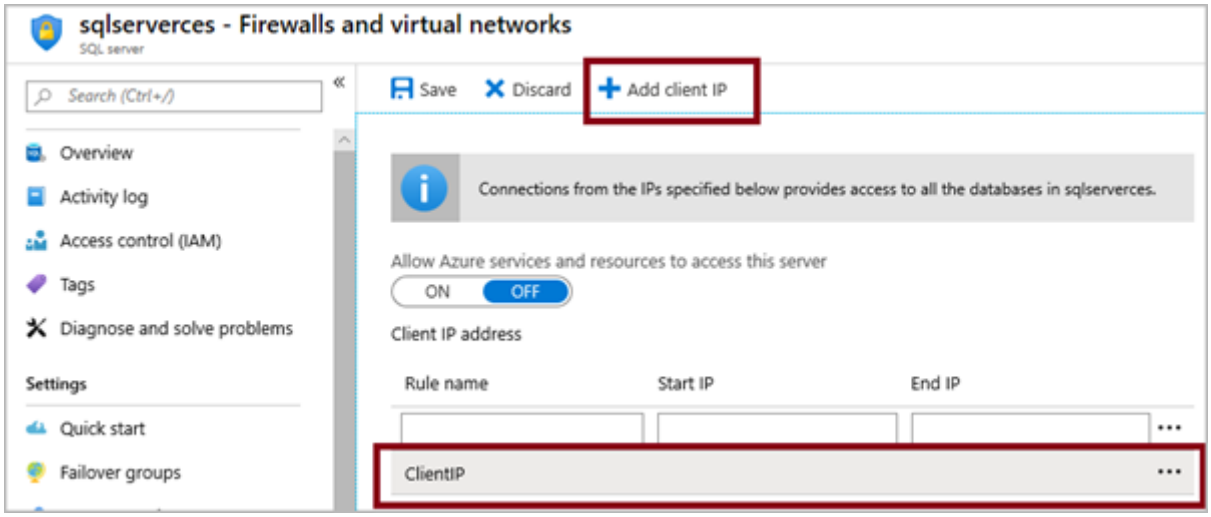
3. Sign in as **sqluser**, with the password **Pa$$w0rd1234**.

4. You will not be able to sign in. Read the error closely and make note of the IP address that needs to be allowed through the firewall.



5. Select **Overview > Set server firewall**.

6. In **Client IP address** your IP will be shown, create a **Rule name** > Add your IP in both **Start IP and END IP** and then select **Save**.

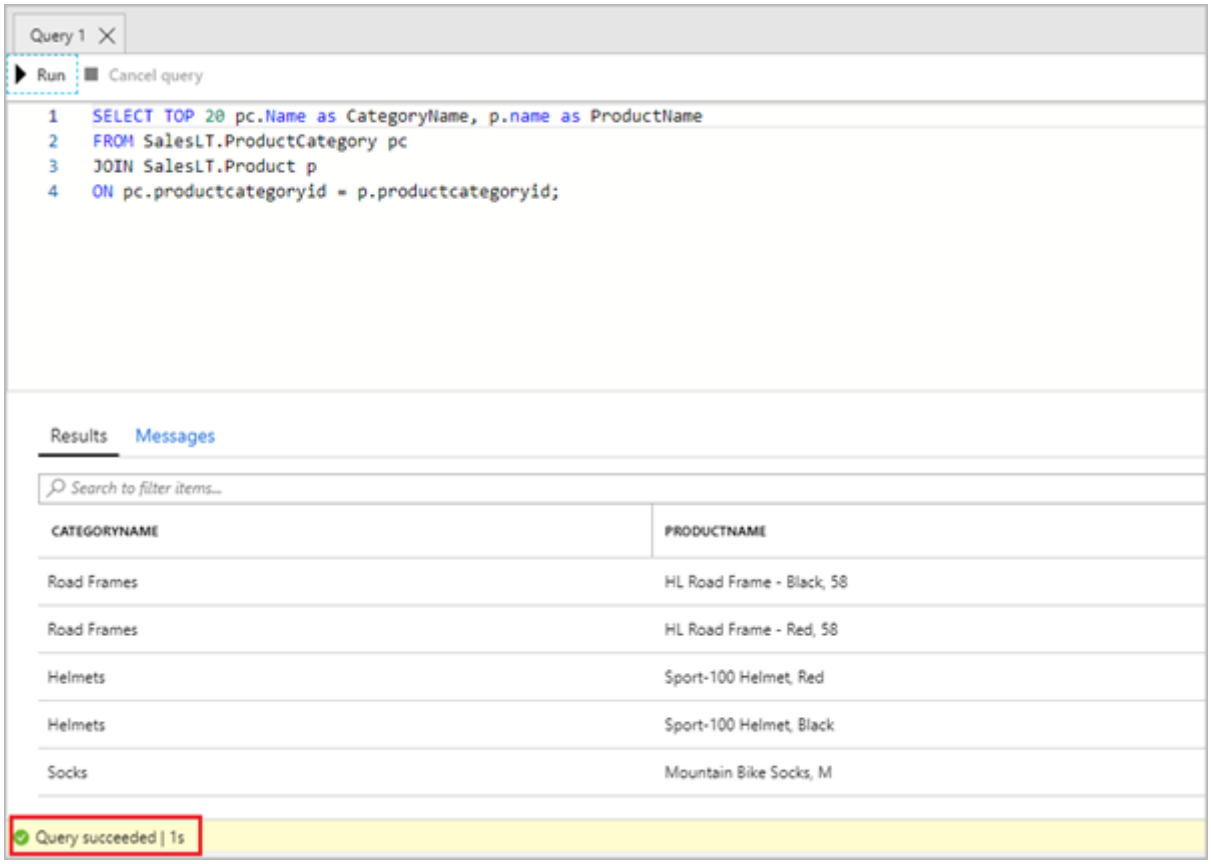


7. Return to your SQL database and the Query Editor sign-in page. Try to sign in again as **sqluser**, with the password **Pa$$w0rd1234**. This time you should succeed. It might take a couple of minutes for the new firewall rule to be deployed. If you wait and still get an error, try selecting **Firewall settings >** again.

8. After you sign in successfully, the query pane appears. Enter the following query into the editor pane:

```
SELECT TOP 20 pc.Name as CategoryName, p.name as ProductName
FROM SalesLT.ProductCategory pc
JOIN SalesLT.Product p
ON pc.productcategoryid = p.productcategoryid;
```

9. Select **Run**, and then review the query results in the **Results** pane. The query should run successfully.



Try Azure Cosmos DB free

- Article
- 11/29/2022
- 3 contributors

Feedback

In this article

APPLIES TO: NoSQL MongoDB Cassandra Gremlin Table PostgreSQL

Try Azure Cosmos DB makes it easy to try out Azure Cosmos DB for free before you commit. There's no credit card required to get started. Your account is free for 30 days. After expiration, a new sandbox account can be created. You can extend beyond 30 days for 24 hours. You can upgrade your active Try Azure Cosmos DB account at any time during the 30 day trial period.

If you're using the API for NoSQL or PostgreSQL, you can also migrate your Try Azure Cosmos DB data to your upgraded account before the trial ends.

This article walks you through how to create your account, limits, and upgrading your account. This article also walks through how to migrate your data from your Try Azure Cosmos DB sandbox to your own account using the API for NoSQL.

Limits to free account

- NoSQL / Cassandra/ Gremlin / Table
- MongoDB
- PostgreSQL

The following table lists the limits for the Try Azure Cosmos DB for Free trial.

Resource	Limit
Duration of the trial	30 days[1][2]
Maximum containers per subscription	1
Maximum throughput per container	5,000
Maximum throughput per shared-throughput database	20,000
Maximum total storage per account	10 GB

[1] A new trial can be requested after expiration. [2] After expiration, the information stored in your account is deleted. You can upgrade your account prior to expiration and migrate the information stored.

Note

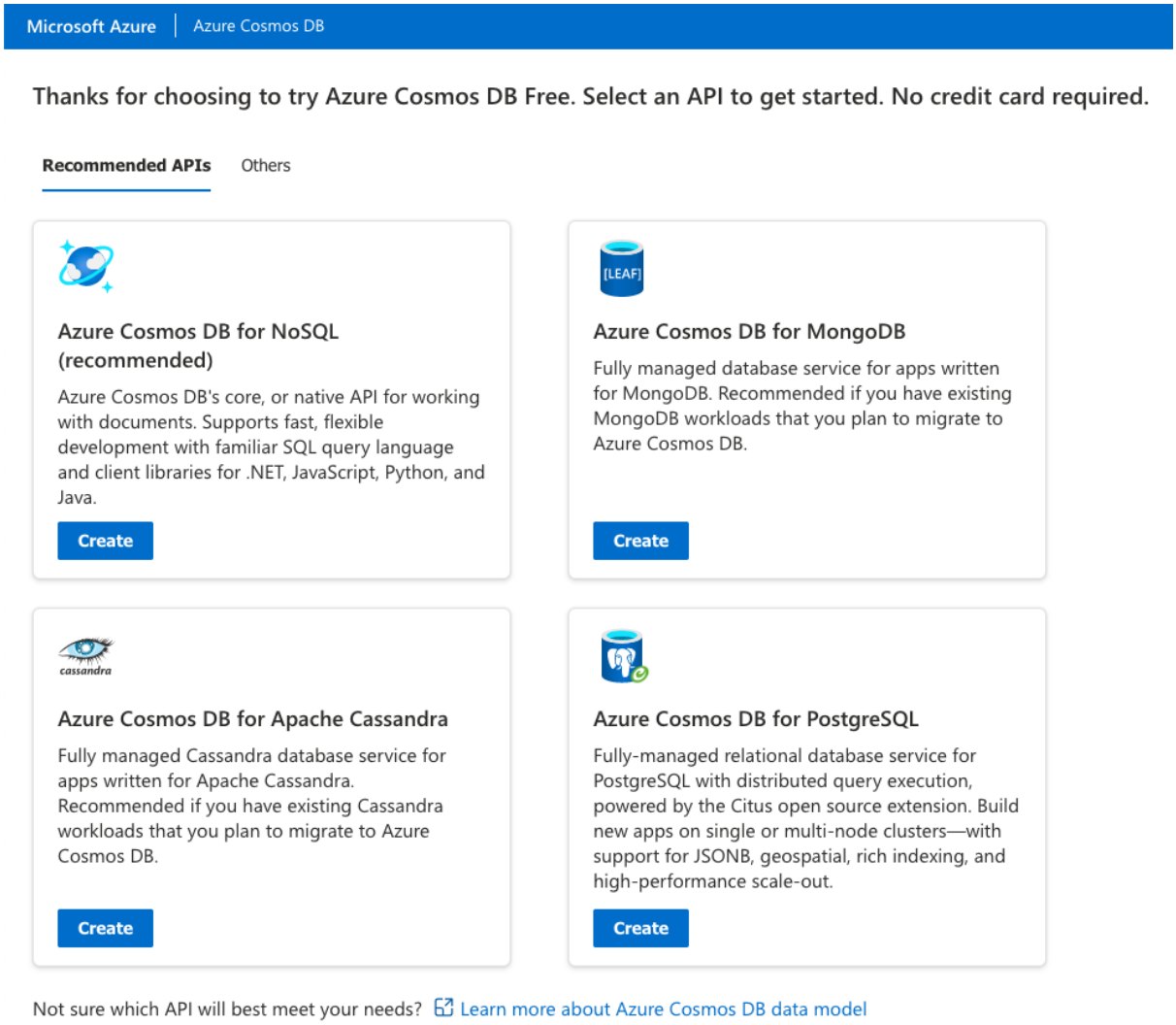
Try Azure Cosmos DB supports global distribution in only the **East US**, **North Europe**, **Southeast Asia**, and **North Central US** regions. Azure support tickets can't be created for Try Azure Cosmos DB accounts. However, support is provided for subscribers with existing support plans.

Create your Try Azure Cosmos DB account

From the Try Azure Cosmos DB home page, select an API. Azure Cosmos DB provides five APIs: NoSQL and MongoDB for document data, Gremlin for graph data, Azure Table, and Cassandra.

Note

Not sure which API will best meet your needs? To learn more about the APIs for Azure Cosmos DB, see **Choose an API in Azure Cosmos DB**.

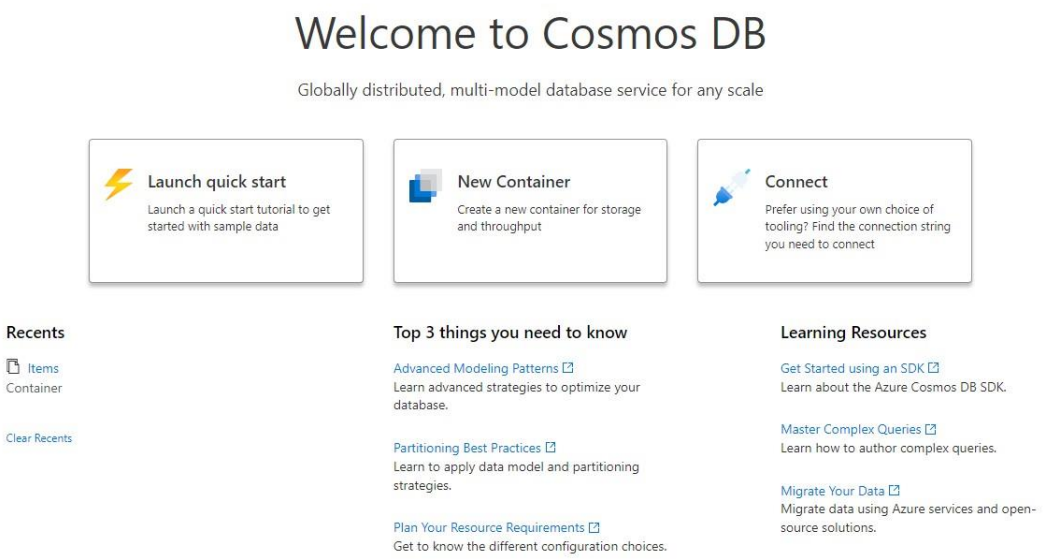


Launch a Quick Start

Launch the Quickstart in Data Explorer in Azure portal to start using Azure Cosmos DB or get started with our documentation.

- API for NoSQL
- API for PostgreSQL
- API for MongoDB
- API for Apache Cassandra
- API for Apache Gremlin
- API for Table

You can also get started with one of the learning resources in the Data Explorer.

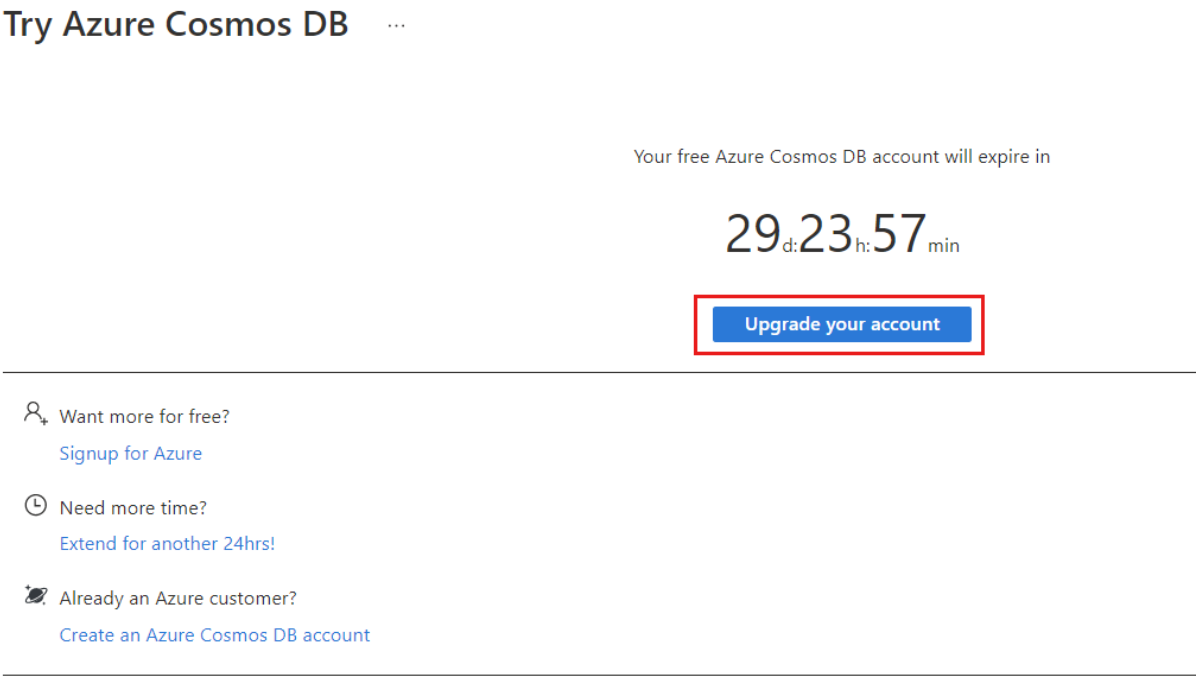


Upgrade your account

Your account is free for 30 days. After expiration, a new sandbox account can be created. You can upgrade your active Try Azure Cosmos DB account at any time during the 30 day trial period. Here are the steps to start an upgrade.

Start upgrade

1. From either the Azure portal or the Try Azure Cosmos DB free page, select the option to **Upgrade** your account.



2. Choose to either **Sign up for an Azure account** or **Sign in** and create a new Azure Cosmos DB account following the instructions in the next section.

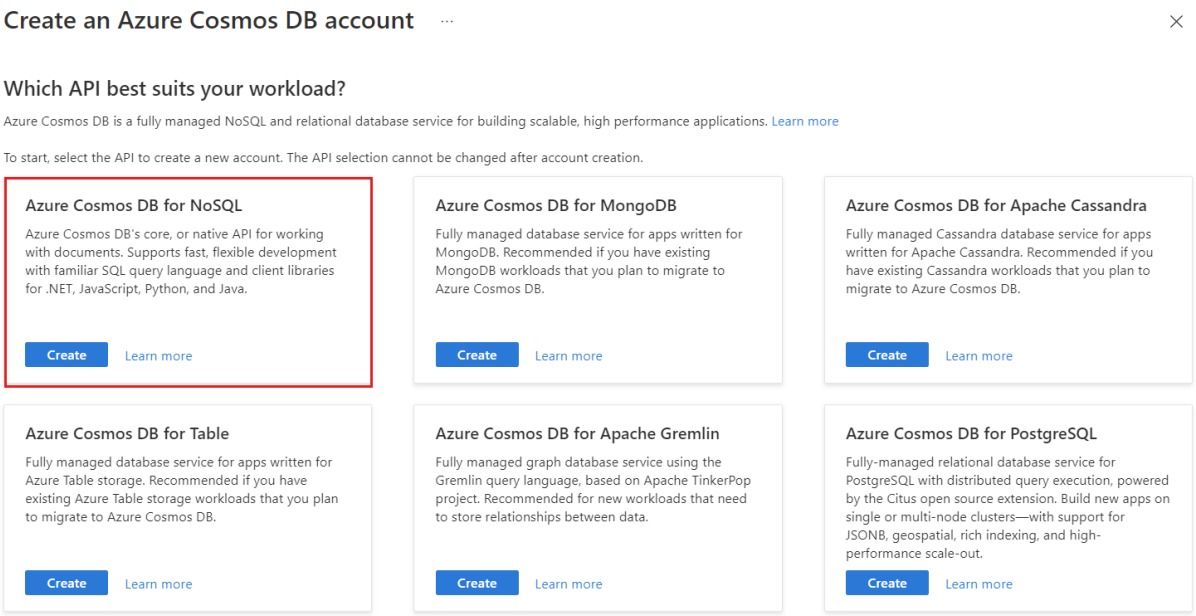
Create a new account

- NoSQL / MongoDB / Cassandra / Gremlin / Table
- PostgreSQL

Note

While this example uses API for NoSQL, the steps are similar for the APIs for MongoDB, Cassandra, Gremlin, or Table.

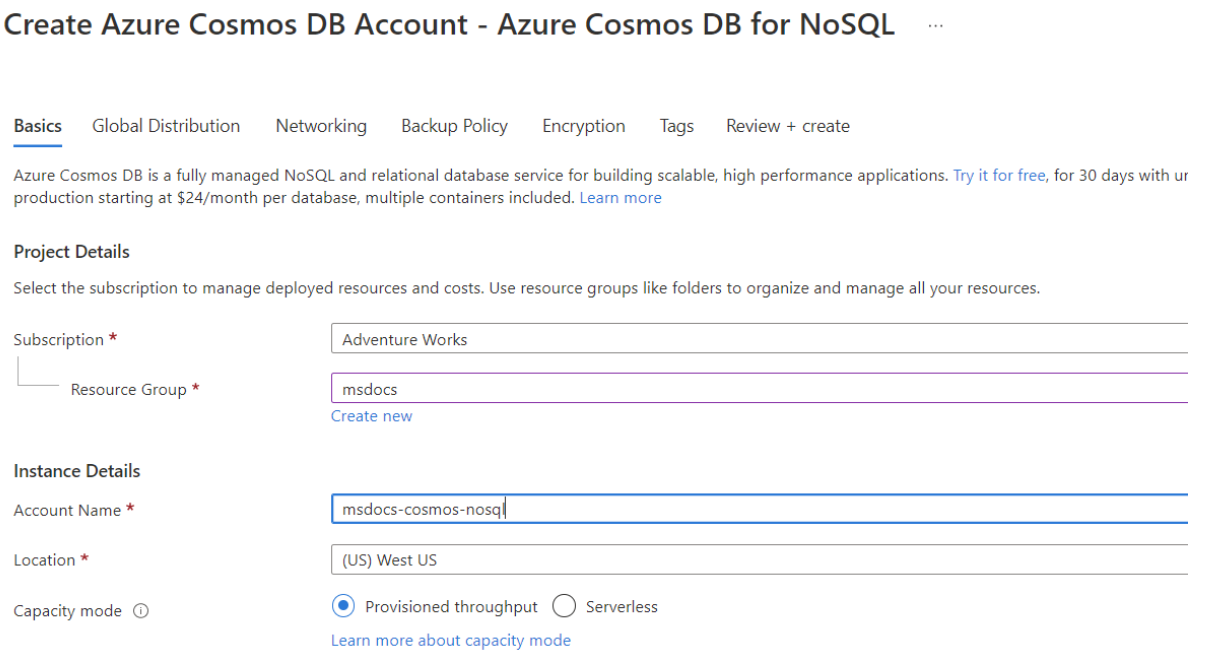
1. Sign in to the Azure portal.
2. From the Azure portal menu or the **Home page**, select **Create a resource**.
3. On the **New** page, search for and select **Azure Cosmos DB**.
4. On the **Select API option** page, select the **Create** option within the **NoSQL** section. Azure Cosmos DB has six APIs: NoSQL, MongoDB, PostgreSQL, Apache Cassandra, Apache Gremlin, and Table. Learn more about the API for NoSQL.



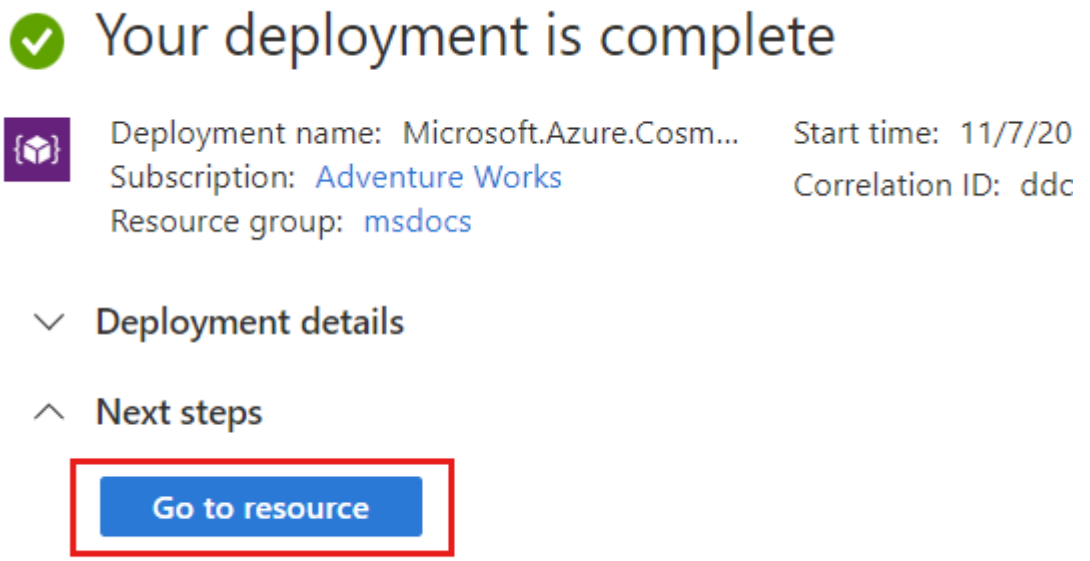
5. On the **Create Azure Cosmos DB Account - Azure Cosmos DB for NoSQL** page, enter the following information:

Setting	Value
Subscription	Select the Azure subscription that you wish to use for this Azure Cosmos account.
Resource Group	Select a resource group, or select **Create new**, then enter a unique name for the new resource group.
Account Name	Enter a name to identify your Azure Cosmos account. The name will be used as part of a fully qualified domain name (FQDN) with a suffix of *documents.azure.com*, so the name must be globally unique. The name can only contain lowercase letters, numbers, and the hyphen (-) character. The name must also be between 3-44 characters in length.
Location	Select a geographic location to host your Azure Cosmos DB account. Use the location that is closest to your users to give them the fastest access to the data.
Capacity mode	Select **Provisioned throughput** to create an account in provisioned throughput mode.
Apply Azure Cosmos DB free tier discount	Choose whether you wish to enable Azure Cosmos DB free tier. With Azure Cosmos DB free tier, you'll get the first 1000 RU/s and 25 GB of storage for free in an account. Learn more about free tier.

6. **Note**
7. You can have up to one free tier Azure Cosmos DB account per Azure subscription and must opt-in when creating the account. If you do not see the option to apply the free tier discount, this means another account in the subscription has already been enabled with free tier.



8.
9. Select **Review + create**.
10. Review the settings you provide, and then select **Create**. It takes a few minutes to create the account. Wait for the portal page to display **Your deployment is complete** before moving on.
11. Select **Go to resource** to go to the Azure Cosmos DB for NoSQL account page.

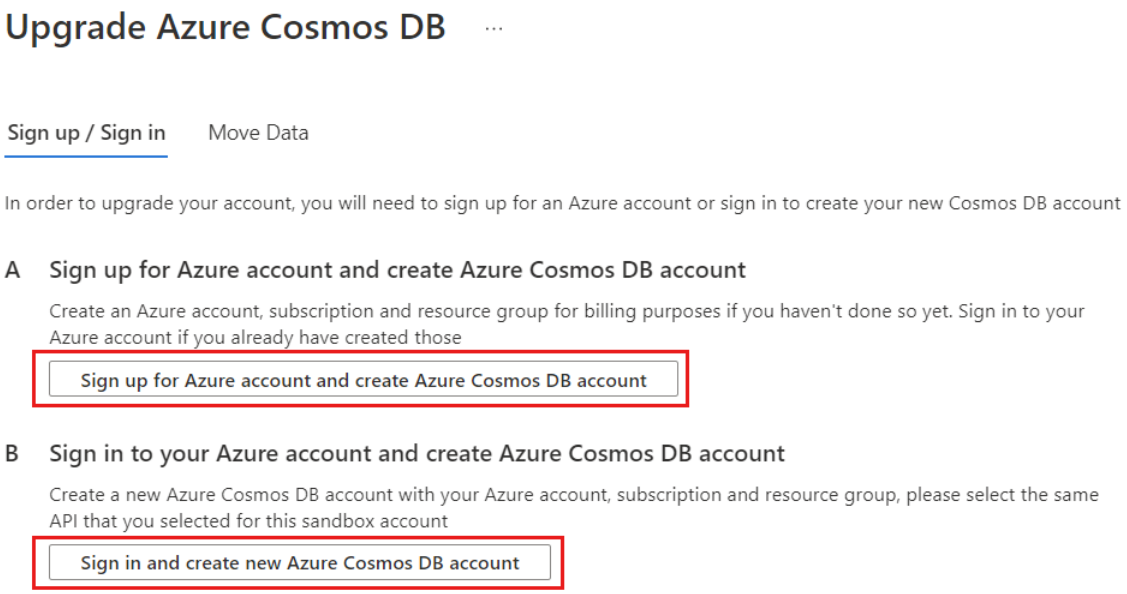


Move data to new account

If you desire, you can migrate your existing data from the free account to the newly created account.

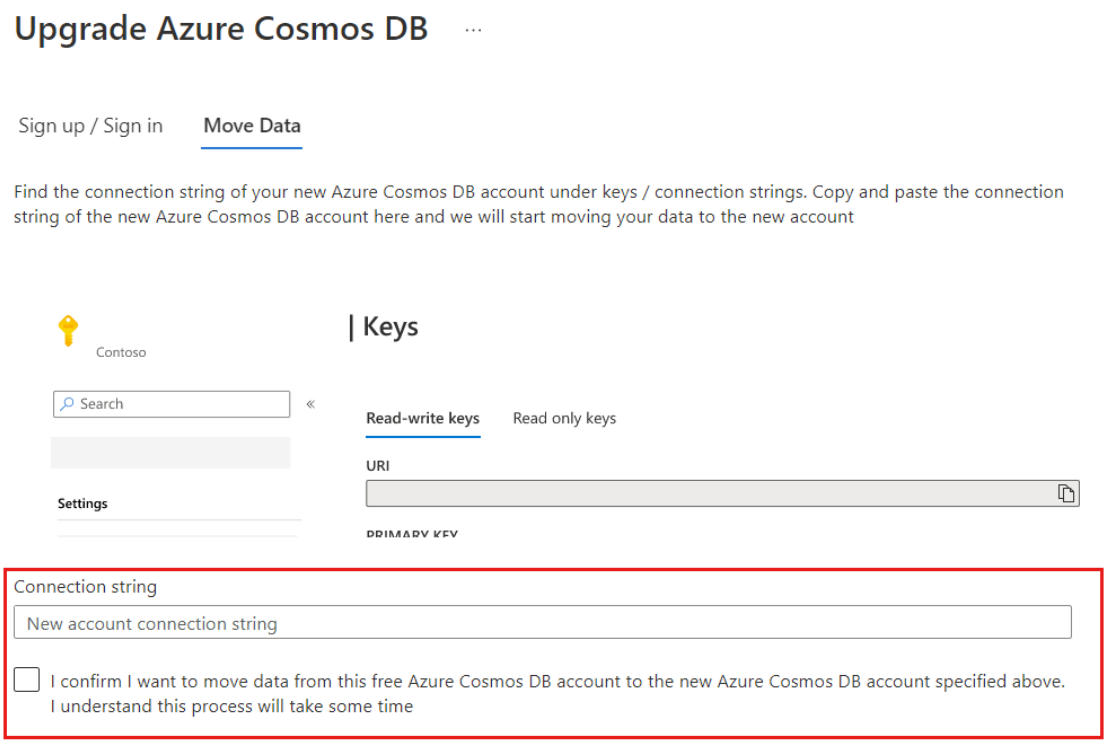
- NoSQL
- PostgreSQL
- MongoDB / Cassandra / Gremlin / Table

1. Navigate back to the **Upgrade** page from the Start upgrade section of this guide. Select **Next** to move on to the third step and move your data.



2. Locate your **Primary Connection string** for the Azure Cosmos DB account you created for your data. This information can be found within the **Keys** page of your new account.

3. Back in the **Upgrade** page from the Start upgrade section of this guide, insert the connection string of the new Azure Cosmos DB account in the **Connection string** field.

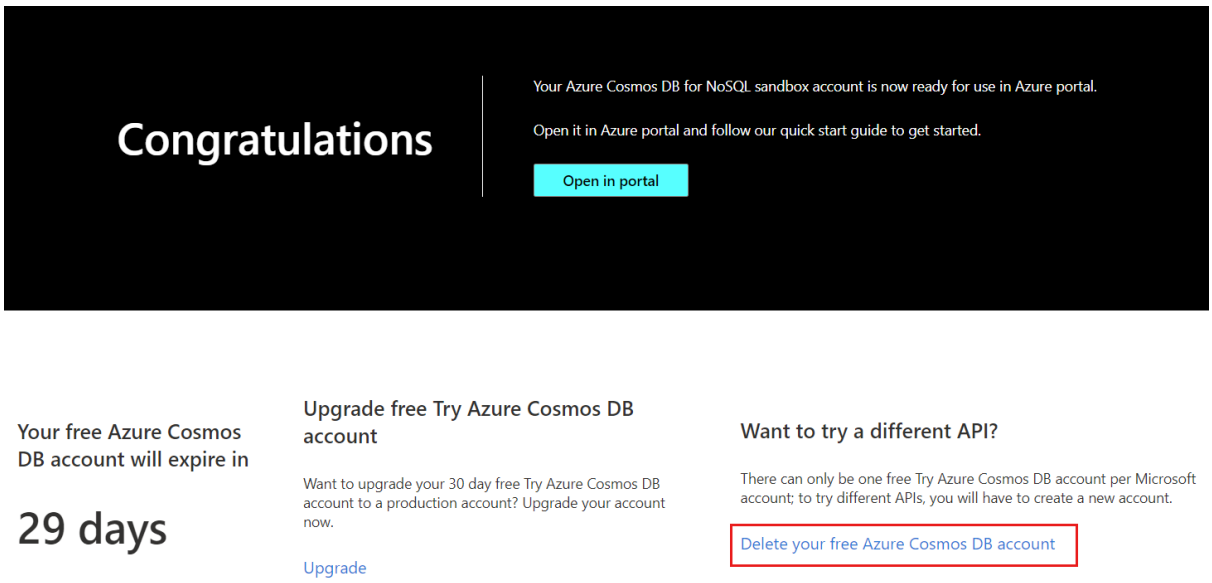


4. Select **Next** to move the data to your account. Provide your email address to be notified by email once the migration has been completed.

Delete your account

There can only be one free Try Azure Cosmos DB account per Microsoft account. You may want to delete your account or to try different APIs, you'll have to create a new account. Here's how to delete your account.

1. Go to the Try Azure Cosmos DB page.
2. Select **Delete my account**.



Next steps

After you create a Try Azure Cosmos DB sandbox account, you can start building apps with Azure Cosmos DB with the following articles:

- Use API for NoSQL to build a console app using .NET to manage data in Azure Cosmos DB.
- Use API for MongoDB to build a sample app using Python to manage data in Azure Cosmos DB.
- Create a Jupyter notebook and analyze your data.
- Learn more about understanding your Azure Cosmos DB bill
- Get started with Azure Cosmos DB with one of our quickstarts:
    - Get started with Azure Cosmos DB for NoSQL
    - Get started with Azure Cosmos DB for PostgreSQL
    - Get started with Azure Cosmos DB for MongoDB
    - Get started with Azure Cosmos DB for Cassandra
    - Get started with Azure Cosmos DB for Gremlin
    - Get started with Azure Cosmos DB for Table

- Trying to do capacity planning for a migration to Azure Cosmos DB? You can use information about your existing database cluster for capacity planning.
- If all you know is the number of vCores and servers in your existing database cluster, see estimating request units using vCores or vCPUs.
- If you know typical request rates for your current database workload, see estimating request units using Azure Cosmos DB capacity planner.

Quickstart: Create a Linux virtual machine with the Azure CLI

- Article
- 03/23/2023
- 20 contributors

Feedback

In this article

Applies to: ✔ Linux VMs

This quickstart shows you how to use the Azure CLI to deploy a Linux virtual machine (VM) in Azure. The Azure CLI is used to create and manage Azure resources via either the command line or scripts.

In this tutorial, we will be installing the latest Debian image. To show the VM in action, you'll connect to it using SSH and install the NGINX web server.

If you don't have an Azure subscription, create a free account before you begin.

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also open Cloud Shell in a separate browser tab by going to https://shell.azure.com/bash. Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and select **Enter** to run it.

If you prefer to install and use the CLI locally, this quickstart requires Azure CLI version 2.0.30 or later. Run `az --version` to find the version. If you need to install or upgrade, see Install Azure CLI.

Define Environment Variables

Environment variables are commonly used in Linux to centralize configuration data to improve consistency and maintainability of the system. Create the following environment variables to specify the names of resources that will be created later in this tutorial:

Azure CLICopy
Open Cloudshell

```
export RESOURCE_GROUP_NAME=myResourceGroup
export LOCATION=eastus
export VM_NAME=myVM
export VM_IMAGE=debian
export ADMIN_USERNAME=azureuser
```

Create a resource group

Create a resource group with the az group create command. An Azure resource group is a logical container into which Azure resources are deployed and managed.

Azure CLICopy
Open Cloudshell

```
az group create --name $RESOURCE_GROUP_NAME --location $LOCATION
```

Create virtual machine

Create a VM with the az vm create command.

The following example creates a VM and adds a user account. The `--generate-ssh-keys` parameter is used to automatically generate an SSH key, and put it in the default key location (*~/.ssh*). To use a specific set of keys instead, use the `--ssh-key-values` option.

Azure CLICopy

Open Cloudshell

```
az vm create \
  --resource-group $RESOURCE_GROUP_NAME \
  --name $VM_NAME \
  --image $VM_IMAGE \
  --admin-username $ADMIN_USERNAME \
  --generate-ssh-keys \
  --public-ip-sku Standard
```

It takes a few minutes to create the VM and supporting resources. The following example output shows the VM create operation was successful.

JSONCopy

```
{
  "fqdns": "",
  "id":
"/subscriptions/<guid>/resourceGroups/myResourceGroup/providers/Microsoft.Compute/
virtualMachines/myVM",
  "location": "eastus",
  "macAddress": "00-0D-3A-23-9A-49",
  "powerState": "VM running",
  "privateIpAddress": "10.0.0.4",
  "publicIpAddress": "40.68.254.142",
  "resourceGroup": "myResourceGroup"
}
```

Make a note of the `publicIpAddress` to use later.

You can retrieve and store the IP address in the variable IP_ADDRESS with the following command:

Azure CLICopy
Open Cloudshell

```
export IP_ADDRESS=$(az vm show --show-details --resource-group
$RESOURCE_GROUP_NAME --name $VM_NAME --query publicIps --output tsv)
```

Install web server

To see your VM in action, install the NGINX web server. Update your package sources and then install the latest NGINX package. The following command uses run-command to run `sudo apt-get update && sudo apt-get install -y nginx` on the VM:

Azure CLICopy
Open Cloudshell

```
az vm run-command invoke \
   --resource-group $RESOURCE_GROUP_NAME \
   --name $VM_NAME \
   --command-id RunShellScript \
   --scripts "sudo apt-get update && sudo apt-get install -y nginx"
```

Open port 80 for web traffic

By default, only SSH connections are opened when you create a Linux VM in Azure. Use az vm open-port to open TCP port 80 for use with the NGINX web server:

Azure CLICopy
Open Cloudshell

```
az vm open-port --port 80 --resource-group $RESOURCE_GROUP_NAME --name $VM_NAME
```

View the web server in action

Use a web browser of your choice to view the default NGINX welcome page. Use the public IP address of your VM as the web address. The following example shows the default NGINX web site:

Alternatively, run the following command to see the NGINX welcome page in the terminal

Azure CLICopy
Open Cloudshell

```
 curl $IP_ADDRESS
```

The following example shows the default NGINX web site in the terminal as successful output:

HTMLCopy

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
```

```
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Clean up resources

When no longer needed, you can use the az group delete command to remove the resource group, VM, and all related resources.

Azure CLICopy
Open Cloudshell

```
az group delete --name $RESOURCE_GROUP_NAME --no-wait --yes --verbose
```

Next steps

In this quickstart, you deployed a simple virtual machine, opened a network port for web traffic, and installed a basic web server. To learn more about Azure virtual machines, continue to the tutorial for Linux VMs.

Quickstart: Create a Linux virtual machine in the Azure portal

- Article
- 03/29/2023
- 26 contributors

Feedback

In this article

Applies to: ✔ Linux VMs

Azure virtual machines (VMs) can be created through the Azure portal. The Azure portal is a browser-based user interface to create Azure resources. This quickstart

shows you how to use the Azure portal to deploy a Linux virtual machine (VM) running Ubuntu Server 22.04 LTS. To see your VM in action, you also SSH to the VM and install the NGINX web server.

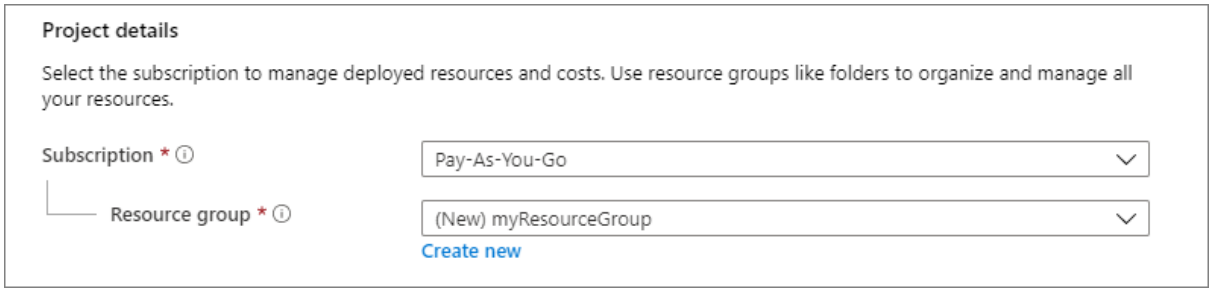
If you don't have an Azure subscription, create a free account before you begin.

Sign in to Azure

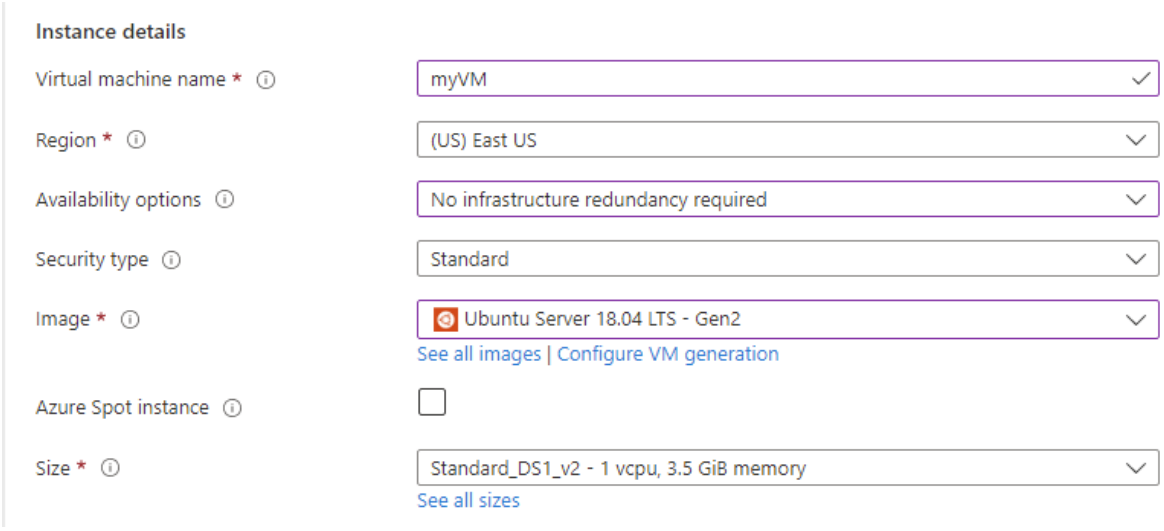
Sign in to the Azure portal.

Create virtual machine

1. Enter *virtual machines* in the search.
2. Under **Services**, select **Virtual machines**.
3. In the **Virtual machines** page, select **Create** and then **Virtual machine**. The **Create a virtual machine** page opens.
4. In the **Basics** tab, under **Project details**, make sure the correct subscription is selected and then choose to **Create new** resource group. Enter *myResourceGroup* for the name.*.

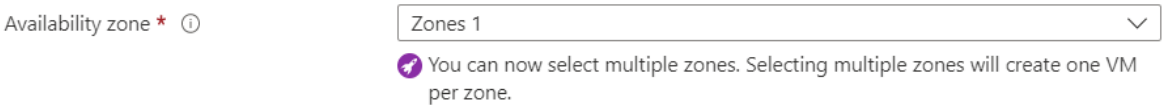


5. Under **Instance details**, enter *myVM* for the **Virtual machine name**, and choose *Ubuntu Server 22.04 LTS - Gen2* for your **Image**. Leave the other defaults. The default size and pricing is only shown as an example. Size availability and pricing are dependent on your region and subscription.

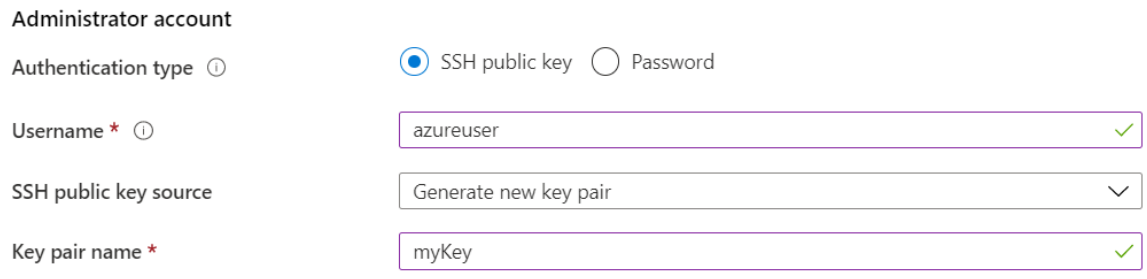


Note

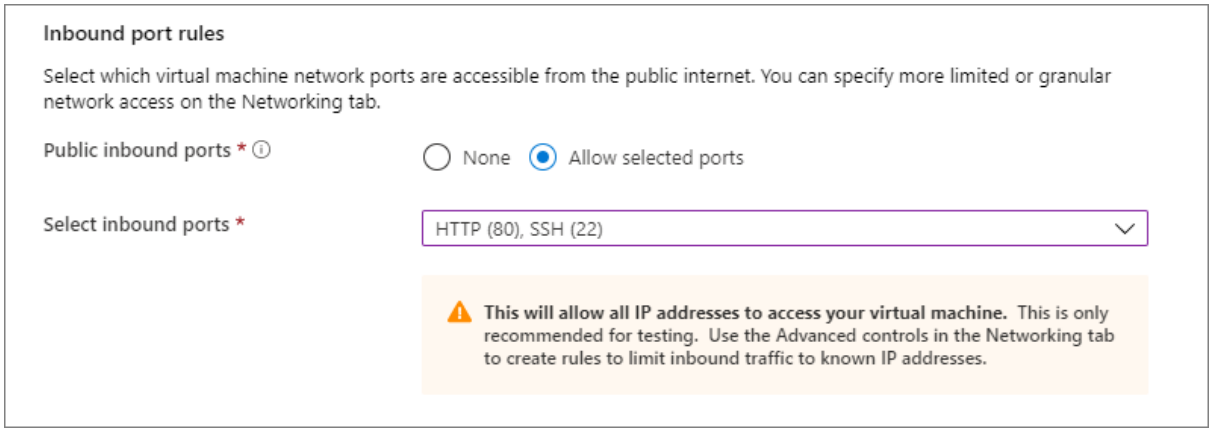
Some users will now see the option to create VMs in multiple zones. To learn more about this new capability, see **Create virtual machines in an availability zone**.



6. Under **Administrator account**, select **SSH public key**.
7. In **Username** enter *azureuser*.
8. For **SSH public key source**, leave the default of **Generate new key pair**, and then enter *myKey* for the **Key pair name**.



9. Under **Inbound port rules** > **Public inbound ports**, choose **Allow selected ports** and then select **SSH (22)** and **HTTP (80)** from the drop-down.



10. Leave the remaining defaults and then select the **Review + create** button at the bottom of the page.
11. On the **Create a virtual machine** page, you can see the details about the VM you are about to create. When you are ready, select **Create**.
12. When the **Generate new key pair** window opens, select **Download private key and create resource**. Your key file will be download as **myKey.pem**. Make sure you know where the `.pem` file was downloaded; you will need the path to it in the next step.
13. When the deployment is finished, select **Go to resource**.
14. On the page for your new VM, select the public IP address and copy it to your clipboard.

Connect to virtual machine

Create an SSH connection with the VM.

1. If you are on a Mac or Linux machine, open a Bash prompt and set read-only permission on the .pem file using `chmod 400 ~/Downloads/myKey.pem`. If you are on a Windows machine, open a PowerShell prompt.
2. At your prompt, open an SSH connection to your virtual machine. Replace the IP address with the one from your VM, and replace the path to the `.pem` with the path to where the key file was downloaded.

ConsoleCopy

```
ssh -i ~/Downloads/myKey.pem azureuser@10.111.12.123
```

Tip

The SSH key you created can be used the next time your create a VM in Azure. Just select the **Use a key stored in Azure** for **SSH public key source** the next time you

create a VM. You already have the private key on your computer, so you won't need to download anything.

Install web server

To see your VM in action, install the NGINX web server. From your SSH session, update your package sources and then install the latest NGINX package.

- Ubuntu
- SUSE Linux (SLES)
- Red Hat Enterprise Linux (RHEL)

BashCopy

```
sudo apt-get -y update
sudo apt-get -y install nginx
```

When done, type `exit` to leave the SSH session.

View the web server in action

Use a web browser of your choice to view the default NGINX welcome page. Type the public IP address of the VM as the web address. The public IP address can be found on the VM overview page or as part of the SSH connection string you used earlier.

Clean up resources

When no longer needed, you can delete the resource group, virtual machine, and all related resources. To do so, select the resource group for the virtual machine, select **Delete**, then confirm the name of the resource group to delete.

Next steps

In this quickstart, you deployed a virtual machine, created a Network Security Group and rule, and installed a basic web server.

Quickstart: Create a Linux virtual machine in Azure with PowerShell

- Article
- 10/22/2022
- 18 contributors

Feedback

In this article

Applies to: ✔ Linux VMs

The Azure PowerShell module is used to create and manage Azure resources from the PowerShell command line or in scripts. This quickstart shows you how to use the Azure PowerShell module to deploy a Linux virtual machine (VM) in Azure. This quickstart uses the latest Debian marketplace image. To see your VM in action, you'll also SSH to the VM and install the NGINX web server.

If you don't have an Azure subscription, create a free account before you begin.

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press enter to run it.

Create a resource group

Create an Azure resource group with New-AzResourceGroup. A resource group is a logical container into which Azure resources are deployed and managed:

Azure PowerShellCopy
Open Cloudshell

```
New-AzResourceGroup -Name 'myResourceGroup' -Location 'EastUS'
```

Create a virtual machine

We will be automatically generating an SSH key pair to use for connecting to the VM. The public key that is created using `-GenerateSshKey` will be stored in Azure as a resource, using the name you provide as `SshKeyName`. The SSH key resource can be reused for creating additional VMs. Both the public and private keys will also be downloaded for you. When you create your SSH key pair using the Cloud Shell, the keys are stored in a storage account that is automatically created by Cloud Shell. Don't delete the storage account, or the file share in it, until after you have retrieved your keys or you will lose access to the VM.

You will be prompted for a user name that will be used when you connect to the VM. You will also be asked for a password, which you can leave blank. Password log in for the VM is disabled when using an SSH key.

In this example, you create a VM named *myVM*, in *East US*, using the *Standard_B2s* VM size.

Azure PowerShellCopy
Open Cloudshell

```
New-AzVm `
    -ResourceGroupName 'myResourceGroup' `
    -Name 'myVM' `
    -Location 'East US' `
    -Image Debian `
    -size Standard_B2s `
    -PublicIpAddressName myPubIP `
```

```
    -OpenPorts 80 `
    -GenerateSshKey `
    -SshKeyName mySSHKey
```

The output will give you the location of the local copy of the SSH key. For example:

OutputCopy

```
Private key is saved to /home/user/.ssh/1234567891
Public key is saved to /home/user/.ssh/1234567891.pub
```

It will take a few minutes for your VM to be deployed. When the deployment is finished, move on to the next section.

Install NGINX

To see your VM in action, install the NGINX web server.

Azure PowerShellCopy
Open Cloudshell

```
Invoke-AzVMRunCommand `
   -ResourceGroupName 'myResourceGroup' `
   -Name 'myVM' `
   -CommandId 'RunShellScript' `
   -ScriptString 'sudo apt-get update && sudo apt-get install -y nginx'
```

The `-ScriptString` parameter requires version `4.27.0` or later of the `Az.Compute` module.

View the web server in action

Get the public IP address of your VM:

Azure PowerShellCopy
Open Cloudshell

```
Get-AzPublicIpAddress -Name myPubIP -ResourceGroupName myResourceGroup | select
"IpAddress"
```

Use a web browser of your choice to view the default NGINX welcome page. Enter the public IP address of the VM as the web address.

Clean up resources

When no longer needed, you can use the Remove-AzResourceGroup cmdlet to remove the resource group, VM, and all related resources:

Azure PowerShellCopy
Open Cloudshell

```
Remove-AzResourceGroup -Name 'myResourceGroup'
```

Next steps

In this quickstart, you deployed a simple virtual machine, created a Network Security Group and rule, and installed a basic web server. To learn more about Azure virtual machines, continue to the tutorial for Linux VMs.

Quickstart: Use Terraform to create a Linux VM

- Article
- 09/01/2022
- 1 contributor

Feedback

In this article

1. Prerequisites
2. Implement the Terraform code
3. Initialize Terraform
4. Create a Terraform execution plan

Show 5 more

Applies to: ✔ Linux VMs

Article tested with the following Terraform and Terraform provider versions:

- Terraform v1.2.7
- AzureRM Provider v.3.20.0

This article shows you how to create a complete Linux environment and supporting resources with Terraform. Those resources include a virtual network, subnet, public IP address, and more.

Terraform enables the definition, preview, and deployment of cloud infrastructure. Using Terraform, you create configuration files using HCL syntax. The HCL syntax allows you to specify the cloud provider - such as Azure - and the elements that make up your cloud infrastructure. After you create your configuration files, you create an *execution plan* that allows you to preview your infrastructure changes before they're deployed. Once you verify the changes, you apply the execution plan to deploy the infrastructure.

In this article, you learn how to:

- Create a virtual network
- Create a subnet
- Create a public IP address
- Create a network security group and SSH inbound rule
- Create a virtual network interface card
- Connect the network security group to the network interface
- Create a storage account for boot diagnostics
- Create SSH key
- Create a virtual machine
- Use SSH to connect to virtual machine

Note

The example code in this article is located in the **Microsoft Terraform GitHub repo**. See more **articles and sample code showing how to use Terraform to manage Azure resources**

Prerequisites

- **Azure subscription**: If you don't have an Azure subscription, create a free account before you begin.
- Install and configure Terraform

Implement the Terraform code

1. Create a directory in which to test the sample Terraform code and make it the current directory.
2. Create a file named `providers.tf` and insert the following code:

   TerraformCopy

```
terraform {
  required_version = ">=0.12"

  required_providers {
```

```
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "~>2.0"
    }
    random = {
      source  = "hashicorp/random"
      version = "~>3.0"
    }
    tls = {
      source = "hashicorp/tls"
      version = "~>4.0"
    }
  }
}

provider "azurerm" {
  features {}
}
```

3. Create a file named `main.tf` and insert the following code:

TerraformCopy

```
resource "random_pet" "rg_name" {
  prefix = var.resource_group_name_prefix
}

resource "azurerm_resource_group" "rg" {
  location = var.resource_group_location
  name     = random_pet.rg_name.id
}

# Create virtual network
resource "azurerm_virtual_network" "my_terraform_network" {
  name                = "myVnet"
  address_space       = ["10.0.0.0/16"]
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
}

# Create subnet
resource "azurerm_subnet" "my_terraform_subnet" {
  name                 = "mySubnet"
  resource_group_name  = azurerm_resource_group.rg.name
  virtual_network_name =
azurerm_virtual_network.my_terraform_network.name
  address_prefixes     = ["10.0.1.0/24"]
}

# Create public IPs
resource "azurerm_public_ip" "my_terraform_public_ip" {
  name                = "myPublicIP"
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
  allocation_method   = "Dynamic"
}

# Create Network Security Group and rule
resource "azurerm_network_security_group" "my_terraform_nsg" {
```

```
  name                = "myNetworkSecurityGroup"
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name

  security_rule {
    name                       = "SSH"
    priority                   = 1001
    direction                  = "Inbound"
    access                     = "Allow"
    protocol                   = "Tcp"
    source_port_range          = "*"
    destination_port_range     = "22"
    source_address_prefix      = "*"
    destination_address_prefix = "*"
  }
}

# Create network interface
resource "azurerm_network_interface" "my_terraform_nic" {
  name                = "myNIC"
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name

  ip_configuration {
    name                          = "my_nic_configuration"
    subnet_id                     =
azurerm_subnet.my_terraform_subnet.id
    private_ip_address_allocation = "Dynamic"
    public_ip_address_id          =
azurerm_public_ip.my_terraform_public_ip.id
  }
}

# Connect the security group to the network interface
resource "azurerm_network_interface_security_group_association"
"example" {
  network_interface_id      =
azurerm_network_interface.my_terraform_nic.id
  network_security_group_id =
azurerm_network_security_group.my_terraform_nsg.id
}

# Generate random text for a unique storage account name
resource "random_id" "random_id" {
  keepers = {
    # Generate a new ID only when a new resource group is defined
    resource_group = azurerm_resource_group.rg.name
  }

  byte_length = 8
}

# Create storage account for boot diagnostics
resource "azurerm_storage_account" "my_storage_account" {
  name                     = "diag${random_id.random_id.hex}"
  location                 = azurerm_resource_group.rg.location
  resource_group_name      = azurerm_resource_group.rg.name
  account_tier             = "Standard"
  account_replication_type = "LRS"
```

```
}

# Create (and display) an SSH key
resource "tls_private_key" "example_ssh" {
  algorithm = "RSA"
  rsa_bits  = 4096
}

# Create virtual machine
resource "azurerm_linux_virtual_machine" "my_terraform_vm" {
  name                  = "myVM"
  location              = azurerm_resource_group.rg.location
  resource_group_name   = azurerm_resource_group.rg.name
  network_interface_ids =
[azurerm_network_interface.my_terraform_nic.id]
  size                  = "Standard_DS1_v2"

  os_disk {
    name                 = "myOsDisk"
    caching              = "ReadWrite"
    storage_account_type = "Premium_LRS"
  }

  source_image_reference {
    publisher = "Canonical"
    offer     = "0001-com-ubuntu-server-jammy"
    sku       = "22_04-lts-gen2"
    version   = "latest"
  }

  computer_name                   = "myvm"
  admin_username                  = "azureuser"
  disable_password_authentication = true

  admin_ssh_key {
    username   = "azureuser"
    public_key = tls_private_key.example_ssh.public_key_openssh
  }

  boot_diagnostics {
    storage_account_uri =
azurerm_storage_account.my_storage_account.primary_blob_endpoint
  }
}
```

4. Create a file named `variables.tf` and insert the following code:

   TerraformCopy

```
variable "resource_group_location" {
  type        = string
  default     = "eastus"
  description = "Location of the resource group."
}

variable "resource_group_name_prefix" {
  type        = string
  default     = "rg"
```

```
  description = "Prefix of the resource group name that's combined
with a random ID so name is unique in your Azure subscription."
}
```

5. Create a file named `outputs.tf` and insert the following code:

   TerraformCopy

```
output "resource_group_name" {
  value = azurerm_resource_group.rg.name
}

output "public_ip_address" {
  value =
azurerm_linux_virtual_machine.my_terraform_vm.public_ip_address
}

output "tls_private_key" {
  value     = tls_private_key.example_ssh.private_key_pem
  sensitive = true
}
```

Initialize Terraform

Run terraform init to initialize the Terraform deployment. This command downloads the Azure provider required to manage your Azure resources.

ConsoleCopy

```
terraform init -upgrade
```

Key points:

- The `-upgrade` parameter upgrades the necessary provider plugins to the newest version that complies with the configuration's version constraints.

Create a Terraform execution plan

Run terraform plan to create an execution plan.

ConsoleCopy

```
terraform plan -out main.tfplan
```

Key points:

- The `terraform plan` command creates an execution plan, but doesn't execute it. Instead, it determines what actions are necessary to create the configuration specified in your configuration files. This pattern allows

you to verify whether the execution plan matches your expectations before making any changes to actual resources.
- The optional `-out` parameter allows you to specify an output file for the plan. Using the `-out` parameter ensures that the plan you reviewed is exactly what is applied.
- To read more about persisting execution plans and security, see the security warning section.

Apply a Terraform execution plan

Run terraform apply to apply the execution plan to your cloud infrastructure.

ConsoleCopy
```
terraform apply main.tfplan
```

Key points:

- The example `terraform apply` command assumes you previously ran `terraform plan -out main.tfplan`.
- If you specified a different filename for the `-out` parameter, use that same filename in the call to `terraform apply`.
- If you didn't use the `-out` parameter, call `terraform apply` without any parameters.

Verify the results

To use SSH to connect to the virtual machine, do the following steps:

1. Run terraform output to get the SSH private key and save it to a file.

   ConsoleCopy
   ```
   terraform output -raw tls_private_key > id_rsa
   ```

2. Run terraform output to get the virtual machine public IP address.

   ConsoleCopy
   ```
   terraform output public_ip_address
   ```

3. Use SSH to connect to the virtual machine.

   ConsoleCopy
   ```
   ssh -i id_rsa azureuser@<public_ip_address>
   ```

   Key points:

- Depending on the permissions of your environment, you might get an error when trying to ssh into the virtual machine using the `id_rsa` key file. If you get an error stating that the private key file is unprotected and can't be used, try running the following command: `chmod 600 id_rsa`, which will restrict read and write access to the owner of the file.

Clean up resources

When you no longer need the resources created via Terraform, do the following steps:

1. Run terraform plan and specify the `destroy` flag.

   ConsoleCopy
   ```
   terraform plan -destroy -out main.destroy.tfplan
   ```

   Key points:

   - The `terraform plan` command creates an execution plan, but doesn't execute it. Instead, it determines what actions are necessary to create the configuration specified in your configuration files. This pattern allows you to verify whether the execution plan matches your expectations before making any changes to actual resources.
   - The optional `-out` parameter allows you to specify an output file for the plan. Using the `-out` parameter ensures that the plan you reviewed is exactly what is applied.
   - To read more about persisting execution plans and security, see the security warning section.
2. Run terraform apply to apply the execution plan.

   ConsoleCopy
   ```
   terraform apply main.destroy.tfplan
   ```

Troubleshoot Terraform on Azure

Troubleshoot common problems when using Terraform on Azure

Next steps

In this quickstart, you deployed a simple virtual machine using Terraform. To learn more about Azure virtual machines, continue to the tutorial for Linux VMs.

Quickstart: Create an Ubuntu Linux virtual machine using a Bicep file

- Article
- 03/31/2023
- 2 contributors

Feedback

In this article

Applies to: ✔ Linux VMs

This quickstart shows you how to use a Bicep file to deploy an Ubuntu Linux virtual machine (VM) in Azure.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

If you don't have an Azure subscription, create a free account before you begin.

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

BicepCopy

```
@description('The name of you Virtual Machine.')
param vmName string = 'simpleLinuxVM'

@description('Username for the Virtual Machine.')
param adminUsername string

@description('Type of authentication to use on the Virtual Machine. SSH key is
recommended.')
@allowed([
  'sshPublicKey'
  'password'
])
param authenticationType string = 'password'

@description('SSH Key or password for the Virtual Machine. SSH key is
recommended.')
@secure()
param adminPasswordOrKey string

@description('Unique DNS Name for the Public IP used to access the Virtual
Machine.')
param dnsLabelPrefix string = toLower('${vmName}-
${uniqueString(resourceGroup().id)}')

@description('The Ubuntu version for the VM. This will pick a fully patched image
of this given Ubuntu version.')
@allowed([
  'Ubuntu-1804'
  'Ubuntu-2004'
  'Ubuntu-2204'
])
param ubuntuOSVersion string = 'Ubuntu-2004'

@description('Location for all resources.')
param location string = resourceGroup().location

@description('The size of the VM')
param vmSize string = 'Standard_D2s_v3'

@description('Name of the VNET')
param virtualNetworkName string = 'vNet'

@description('Name of the subnet in the virtual network')
param subnetName string = 'Subnet'

@description('Name of the Network Security Group')
param networkSecurityGroupName string = 'SecGroupNet'

@description('Security Type of the Virtual Machine.')
@allowed([
  'Standard'
  'TrustedLaunch'
])
param securityType string = 'TrustedLaunch'

var imageReference = {
  'Ubuntu-1804': {
    publisher: 'Canonical'
    offer: 'UbuntuServer'
```

```
    sku: '18_04-lts-gen2'
    version: 'latest'
  }
  'Ubuntu-2004': {
    publisher: 'Canonical'
    offer: '0001-com-ubuntu-server-focal'
    sku: '20_04-lts-gen2'
    version: 'latest'
  }
  'Ubuntu-2204': {
    publisher: 'Canonical'
    offer: '0001-com-ubuntu-server-jammy'
    sku: '22_04-lts-gen2'
    version: 'latest'
  }
}
var publicIPAddressName = '${vmName}PublicIP'
var networkInterfaceName = '${vmName}NetInt'
var osDiskType = 'Standard_LRS'
var subnetAddressPrefix = '10.1.0.0/24'
var addressPrefix = '10.1.0.0/16'
var linuxConfiguration = {
  disablePasswordAuthentication: true
  ssh: {
    publicKeys: [
      {
        path: '/home/${adminUsername}/.ssh/authorized_keys'
        keyData: adminPasswordOrKey
      }
    ]
  }
}
var securityProfileJson = {
  uefiSettings: {
    secureBootEnabled: true
    vTpmEnabled: true
  }
  securityType: securityType
}
var extensionName = 'GuestAttestation'
var extensionPublisher = 'Microsoft.Azure.Security.LinuxAttestation'
var extensionVersion = '1.0'
var maaTenantName = 'GuestAttestation'
var maaEndpoint = substring('emptystring', 0, 0)

resource networkInterface 'Microsoft.Network/networkInterfaces@2021-05-01' = {
  name: networkInterfaceName
  location: location
  properties: {
    ipConfigurations: [
      {
        name: 'ipconfig1'
        properties: {
          subnet: {
            id: subnet.id
          }
          privateIPAllocationMethod: 'Dynamic'
          publicIPAddress: {
            id: publicIPAddress.id
```

```
          }
        }
      }
    ]
    networkSecurityGroup: {
      id: networkSecurityGroup.id
    }
  }
}

resource networkSecurityGroup 'Microsoft.Network/networkSecurityGroups@2021-05-01'
= {
  name: networkSecurityGroupName
  location: location
  properties: {
    securityRules: [
      {
        name: 'SSH'
        properties: {
          priority: 1000
          protocol: 'Tcp'
          access: 'Allow'
          direction: 'Inbound'
          sourceAddressPrefix: '*'
          sourcePortRange: '*'
          destinationAddressPrefix: '*'
          destinationPortRange: '22'
        }
      }
    ]
  }
}

resource virtualNetwork 'Microsoft.Network/virtualNetworks@2021-05-01' = {
  name: virtualNetworkName
  location: location
  properties: {
    addressSpace: {
      addressPrefixes: [
        addressPrefix
      ]
    }
  }
}

resource subnet 'Microsoft.Network/virtualNetworks/subnets@2021-05-01' = {
  parent: virtualNetwork
  name: subnetName
  properties: {
    addressPrefix: subnetAddressPrefix
    privateEndpointNetworkPolicies: 'Enabled'
    privateLinkServiceNetworkPolicies: 'Enabled'
  }
}

resource publicIPAddress 'Microsoft.Network/publicIPAddresses@2021-05-01' = {
  name: publicIPAddressName
  location: location
  sku: {
```

```
    name: 'Basic'
  }
  properties: {
    publicIPAllocationMethod: 'Dynamic'
    publicIPAddressVersion: 'IPv4'
    dnsSettings: {
      domainNameLabel: dnsLabelPrefix
    }
    idleTimeoutInMinutes: 4
  }
}

resource vm 'Microsoft.Compute/virtualMachines@2021-11-01' = {
  name: vmName
  location: location
  properties: {
    hardwareProfile: {
      vmSize: vmSize
    }
    storageProfile: {
      osDisk: {
        createOption: 'FromImage'
        managedDisk: {
          storageAccountType: osDiskType
        }
      }
      imageReference: imageReference[ubuntuOSVersion]
    }
    networkProfile: {
      networkInterfaces: [
        {
          id: networkInterface.id
        }
      ]
    }
    osProfile: {
      computerName: vmName
      adminUsername: adminUsername
      adminPassword: adminPasswordOrKey
      linuxConfiguration: ((authenticationType == 'password') ? null :
linuxConfiguration)
    }
    securityProfile: ((securityType == 'TrustedLaunch') ? securityProfileJson :
null)
  }
}

resource vmExtension 'Microsoft.Compute/virtualMachines/extensions@2022-03-01' =
if ((securityType == 'TrustedLaunch') &&
((securityProfileJson.uefiSettings.secureBootEnabled == true) &&
(securityProfileJson.uefiSettings.vTpmEnabled == true))) {
  parent: vm
  name: extensionName
  location: location
  properties: {
    publisher: extensionPublisher
    type: extensionName
    typeHandlerVersion: extensionVersion
    autoUpgradeMinorVersion: true
```

```
    enableAutomaticUpgrade: true
    settings: {
      AttestationConfig: {
        MaaSettings: {
          maaEndpoint: maaEndpoint
          maaTenantName: maaTenantName
        }
      }
    }
  }
}

output adminUsername string = adminUsername
output hostname string = publicIPAddress.properties.dnsSettings.fqdn
output sshCommand string = 'ssh
${adminUsername}@${publicIPAddress.properties.dnsSettings.fqdn}'
```

Several resources are defined in the Bicep file:

- **Microsoft.Network/virtualNetworks/subnets**: create a subnet.
- **Microsoft.Storage/storageAccounts**: create a storage account.
- **Microsoft.Network/networkInterfaces**: create a NIC.
- **Microsoft.Network/networkSecurityGroups**: create a network security group.
- **Microsoft.Network/virtualNetworks**: create a virtual network.
- **Microsoft.Network/publicIPAddresses**: create a public IP address.
- **Microsoft.Compute/virtualMachines**: create a virtual machine.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.
   - CLI
   - PowerShell

   Azure CLICopy

   ```
   az group create --name exampleRG --location eastus

   az deployment group create --resource-group exampleRG --template-file
   main.bicep --parameters adminUsername=<admin-username>
   ```

   Note

   Replace **<admin-username>** with a unique username. You'll also be prompted to enter adminPasswordOrKey.

   When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

Azure CLICopy
Open Cloudshell

```
az resource list --resource-group exampleRG
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the VM and all of the resources in the resource group.

- CLI
- PowerShell

Azure CLICopy
Open Cloudshell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you deployed a simple virtual machine using a Bicep file. To learn more about Azure virtual machines, continue to the tutorial for Linux VMs.

Quickstart: Create an Ubuntu Linux virtual machine by using an ARM template

- Article
- 04/18/2023
- 8 contributors

Feedback

In this article

Applies to: ✔ Linux VMs

This quickstart shows you how to use an Azure Resource Manager template (ARM template) to deploy an Ubuntu Linux virtual machine (VM) in Azure.

A resource manager template is a JavaScript Object Notation (JSON) file that defines the infrastructure and configuration for your project. The template uses declarative syntax. In declarative syntax, you describe your intended deployment without writing the sequence of programming commands to create the deployment.

If your environment meets the prerequisites and you're familiar with ARM templates, select the **Deploy to Azure** button. The template opens in the Azure portal.

Prerequisites

If you don't have an Azure subscription, create a free account before you begin.

Review the template

For more information on this template, see Deploy a simple Ubuntu Linux VM 18.04-LTS.

JSONCopy


```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-
01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "metadata": {
    "_generator": {
      "name": "bicep",
      "version": "0.17.1.54307",
      "templateHash": "11453007703022839041"
    }
  },
  "parameters": {
```

```
    "vmName": {
      "type": "string",
      "defaultValue": "simpleLinuxVM",
      "metadata": {
        "description": "The name of you Virtual Machine."
      }
    },
    "adminUsername": {
      "type": "string",
      "metadata": {
        "description": "Username for the Virtual Machine."
      }
    },
    "authenticationType": {
      "type": "string",
      "defaultValue": "password",
      "allowedValues": [
        "sshPublicKey",
        "password"
      ],
      "metadata": {
        "description": "Type of authentication to use on the Virtual Machine. SSH
key is recommended."
      }
    },
    "adminPasswordOrKey": {
      "type": "securestring",
      "metadata": {
        "description": "SSH Key or password for the Virtual Machine. SSH key is
recommended."
      }
    },
    "dnsLabelPrefix": {
      "type": "string",
      "defaultValue": "[toLower(format('{0}-{1}', parameters('vmName'),
uniqueString(resourceGroup().id)))]",
      "metadata": {
        "description": "Unique DNS Name for the Public IP used to access the
Virtual Machine."
      }
    },
    "ubuntuOSVersion": {
      "type": "string",
      "defaultValue": "Ubuntu-2004",
      "allowedValues": [
        "Ubuntu-1804",
        "Ubuntu-2004",
        "Ubuntu-2204"
      ],
      "metadata": {
        "description": "The Ubuntu version for the VM. This will pick a fully
patched image of this given Ubuntu version."
      }
    },
    "location": {
      "type": "string",
      "defaultValue": "[resourceGroup().location]",
      "metadata": {
        "description": "Location for all resources."
```

```
      }
    },
    "vmSize": {
      "type": "string",
      "defaultValue": "Standard_D2s_v3",
      "metadata": {
        "description": "The size of the VM"
      }
    },
    "virtualNetworkName": {
      "type": "string",
      "defaultValue": "vNet",
      "metadata": {
        "description": "Name of the VNET"
      }
    },
    "subnetName": {
      "type": "string",
      "defaultValue": "Subnet",
      "metadata": {
        "description": "Name of the subnet in the virtual network"
      }
    },
    "networkSecurityGroupName": {
      "type": "string",
      "defaultValue": "SecGroupNet",
      "metadata": {
        "description": "Name of the Network Security Group"
      }
    },
    "securityType": {
      "type": "string",
      "defaultValue": "TrustedLaunch",
      "allowedValues": [
        "Standard",
        "TrustedLaunch"
      ],
      "metadata": {
        "description": "Security Type of the Virtual Machine."
      }
    }
  },
  "variables": {
    "imageReference": {
      "Ubuntu-1804": {
        "publisher": "Canonical",
        "offer": "UbuntuServer",
        "sku": "18_04-lts-gen2",
        "version": "latest"
      },
      "Ubuntu-2004": {
        "publisher": "Canonical",
        "offer": "0001-com-ubuntu-server-focal",
        "sku": "20_04-lts-gen2",
        "version": "latest"
      },
      "Ubuntu-2204": {
        "publisher": "Canonical",
        "offer": "0001-com-ubuntu-server-jammy",
```

```
        "sku": "22_04-lts-gen2",
        "version": "latest"
      }
    },
    "publicIPAddressName": "[format('{0}PublicIP', parameters('vmName'))]",
    "networkInterfaceName": "[format('{0}NetInt', parameters('vmName'))]",
    "osDiskType": "Standard_LRS",
    "subnetAddressPrefix": "10.1.0.0/24",
    "addressPrefix": "10.1.0.0/16",
    "linuxConfiguration": {
      "disablePasswordAuthentication": true,
      "ssh": {
        "publicKeys": [
          {
            "path": "[format('/home/{0}/.ssh/authorized_keys',
parameters('adminUsername'))]",
            "keyData": "[parameters('adminPasswordOrKey')]"
          }
        ]
      }
    },
    "securityProfileJson": {
      "uefiSettings": {
        "secureBootEnabled": true,
        "vTpmEnabled": true
      },
      "securityType": "[parameters('securityType')]"
    },
    "extensionName": "GuestAttestation",
    "extensionPublisher": "Microsoft.Azure.Security.LinuxAttestation",
    "extensionVersion": "1.0",
    "maaTenantName": "GuestAttestation",
    "maaEndpoint": "[substring('emptystring', 0, 0)]"
  },
  "resources": [
    {
      "type": "Microsoft.Network/networkInterfaces",
      "apiVersion": "2021-05-01",
      "name": "[variables('networkInterfaceName')]",
      "location": "[parameters('location')]",
      "properties": {
        "ipConfigurations": [
          {
            "name": "ipconfig1",
            "properties": {
              "subnet": {
                "id": "[resourceId('Microsoft.Network/virtualNetworks/subnets',
parameters('virtualNetworkName'), parameters('subnetName'))]"
              },
              "privateIPAllocationMethod": "Dynamic",
              "publicIPAddress": {
                "id": "[resourceId('Microsoft.Network/publicIPAddresses',
variables('publicIPAddressName'))]"
              }
            }
          }
        ],
        "networkSecurityGroup": {
```

```
          "id": "[resourceId('Microsoft.Network/networkSecurityGroups',
parameters('networkSecurityGroupName'))]"
        }
      },
      "dependsOn": [
        "[resourceId('Microsoft.Network/networkSecurityGroups',
parameters('networkSecurityGroupName'))]",
        "[resourceId('Microsoft.Network/publicIPAddresses',
variables('publicIPAddressName'))]",
        "[resourceId('Microsoft.Network/virtualNetworks/subnets',
parameters('virtualNetworkName'), parameters('subnetName'))]"
      ]
    },
    {
      "type": "Microsoft.Network/networkSecurityGroups",
      "apiVersion": "2021-05-01",
      "name": "[parameters('networkSecurityGroupName')]",
      "location": "[parameters('location')]",
      "properties": {
        "securityRules": [
          {
            "name": "SSH",
            "properties": {
              "priority": 1000,
              "protocol": "Tcp",
              "access": "Allow",
              "direction": "Inbound",
              "sourceAddressPrefix": "*",
              "sourcePortRange": "*",
              "destinationAddressPrefix": "*",
              "destinationPortRange": "22"
            }
          }
        ]
      }
    },
    {
      "type": "Microsoft.Network/virtualNetworks",
      "apiVersion": "2021-05-01",
      "name": "[parameters('virtualNetworkName')]",
      "location": "[parameters('location')]",
      "properties": {
        "addressSpace": {
          "addressPrefixes": [
            "[variables('addressPrefix')]"
          ]
        }
      }
    },
    {
      "type": "Microsoft.Network/virtualNetworks/subnets",
      "apiVersion": "2021-05-01",
      "name": "[format('{0}/{1}', parameters('virtualNetworkName'),
parameters('subnetName'))]",
      "properties": {
        "addressPrefix": "[variables('subnetAddressPrefix')]",
        "privateEndpointNetworkPolicies": "Enabled",
        "privateLinkServiceNetworkPolicies": "Enabled"
      },
```

```
      "dependsOn": [
        "[resourceId('Microsoft.Network/virtualNetworks',
parameters('virtualNetworkName'))]"
      ]
    },
    {
      "type": "Microsoft.Network/publicIPAddresses",
      "apiVersion": "2021-05-01",
      "name": "[variables('publicIPAddressName')]",
      "location": "[parameters('location')]",
      "sku": {
        "name": "Basic"
      },
      "properties": {
        "publicIPAllocationMethod": "Dynamic",
        "publicIPAddressVersion": "IPv4",
        "dnsSettings": {
          "domainNameLabel": "[parameters('dnsLabelPrefix')]"
        },
        "idleTimeoutInMinutes": 4
      }
    },
    {
      "type": "Microsoft.Compute/virtualMachines",
      "apiVersion": "2021-11-01",
      "name": "[parameters('vmName')]",
      "location": "[parameters('location')]",
      "properties": {
        "hardwareProfile": {
          "vmSize": "[parameters('vmSize')]"
        },
        "storageProfile": {
          "osDisk": {
            "createOption": "FromImage",
            "managedDisk": {
              "storageAccountType": "[variables('osDiskType')]"
            }
          },
          "imageReference":
"[variables('imageReference')[parameters('ubuntuOSVersion')]]"
        },
        "networkProfile": {
          "networkInterfaces": [
            {
              "id": "[resourceId('Microsoft.Network/networkInterfaces',
variables('networkInterfaceName'))]"
            }
          ]
        },
        "osProfile": {
          "computerName": "[parameters('vmName')]",
          "adminUsername": "[parameters('adminUsername')]",
          "adminPassword": "[parameters('adminPasswordOrKey')]",
          "linuxConfiguration": "[if(equals(parameters('authenticationType'),
'password'), null(), variables('linuxConfiguration'))]"
        },
        "securityProfile": "[if(equals(parameters('securityType'),
'TrustedLaunch'), variables('securityProfileJson'), null())]"
      },
```

```
      "dependsOn": [
        "[resourceId('Microsoft.Network/networkInterfaces',
variables('networkInterfaceName'))]"
      ]
    },
    {
      "condition": "[and(equals(parameters('securityType'), 'TrustedLaunch'),
and(equals(variables('securityProfileJson').uefiSettings.secureBootEnabled,
true()), equals(variables('securityProfileJson').uefiSettings.vTpmEnabled,
true())))]",
      "type": "Microsoft.Compute/virtualMachines/extensions",
      "apiVersion": "2022-03-01",
      "name": "[format('{0}/{1}', parameters('vmName'),
variables('extensionName'))]",
      "location": "[parameters('location')]",
      "properties": {
        "publisher": "[variables('extensionPublisher')]",
        "type": "[variables('extensionName')]",
        "typeHandlerVersion": "[variables('extensionVersion')]",
        "autoUpgradeMinorVersion": true,
        "enableAutomaticUpgrade": true,
        "settings": {
          "AttestationConfig": {
            "MaaSettings": {
              "maaEndpoint": "[variables('maaEndpoint')]",
              "maaTenantName": "[variables('maaTenantName')]"
            }
          }
        }
      },
      "dependsOn": [
        "[resourceId('Microsoft.Compute/virtualMachines', parameters('vmName'))]"
      ]
    }
  ],
  "outputs": {
    "adminUsername": {
      "type": "string",
      "value": "[parameters('adminUsername')]"
    },
    "hostname": {
      "type": "string",
      "value": "[reference(resourceId('Microsoft.Network/publicIPAddresses',
variables('publicIPAddressName')), '2021-05-01').dnsSettings.fqdn]"
    },
    "sshCommand": {
      "type": "string",
      "value": "[format('ssh {0}@{1}', parameters('adminUsername'),
reference(resourceId('Microsoft.Network/publicIPAddresses',
variables('publicIPAddressName')), '2021-05-01').dnsSettings.fqdn)]"
    }
  }
}
```

Several resources are defined in the template:

- Microsoft.Network/virtualNetworks/subnets: create a subnet.
- Microsoft.Storage/storageAccounts: create a storage account.

- Microsoft.Network/networkInterfaces: create a NIC.
- Microsoft.Network/networkSecurityGroups: create a network security group.
- Microsoft.Network/virtualNetworks: create a virtual network.
- Microsoft.Network/publicIPAddresses: create a public IP address.
- Microsoft.Compute/virtualMachines: create a virtual machine.

Deploy the template

1. Select the following image to sign in to Azure and open a template. The template creates a key vault and a secret.

2. Select or enter the following values. Use the default values, when available.
    - **Subscription**: select an Azure subscription.
    - **Resource group**: select an existing resource group from the drop-down, or select **Create new**, enter a unique name for the resource group, and select **OK**.
    - **Region**: select a region. For example, **Central US**.
    - **Admin username**: provide a username, such as *azureuser*.
    - **Authentication type**: You can choose between an SSH key or a password.
    - **Admin Password Or Key** depending on what you choose for authentication type:
        - If you choose **password**, the password must be at least 12 characters long and meet the defined complexity requirements.
        - If you choose **sshPublicKey**, paste in the contents of your public key.
    - **DNS label prefix**: enter a unique identifier to use as part of the DNS label.
    - **Ubuntu OS version**: select which version of Ubuntu you want to run on the VM.
    - **Location**: the default is the same location as the resource group, if it already exists.
    - **VM size**: select the size to use for the VM.
    - **Virtual Network Name**: name to be used for the vNet.
    - **Subnet Name**: name for the subnet the VM should use.
    - **Network Security Group Name**: name for the NSG.

3. Select **Review + create**. After validation completes, select **Create** to create and deploy the VM.

The Azure portal is used to deploy the template. In addition to the Azure portal, you can also use the Azure CLI, Azure PowerShell, and REST API. To learn other deployment methods, see Deploy templates.

Review deployed resources

You can use the Azure portal to check on the VM and other resource that were created. After the deployment is finished, select **Resource groups** to see the VM and other resources.

Clean up resources

When no longer needed, delete the resource group, which deletes the VM and all of the resources in the resource group.

1. Select the **Resource group**.
2. On the page for the resource group, select **Delete resource group**.
3. When prompted, type the name of the resource group and then select **Delete**.

Next steps

In this quickstart, you deployed a virtual machine by using an ARM template. To learn more about Azure virtual machines, continue to the tutorial for Linux VMs.

Quickstart: Create a Windows virtual machine with the Azure CLI

- Article
- 02/24/2023
- 19 contributors

Feedback

In this article

1. Launch Azure Cloud Shell
2. Create a resource group
3. Create virtual machine
4. Install web server

Show 4 more

Applies to: ✔ Windows VMs

The Azure CLI is used to create and manage Azure resources from the command line or in scripts. This quickstart shows you how to use the Azure CLI to deploy a virtual machine (VM) in Azure that runs Windows Server 2019. To see your VM in action, you then RDP to the VM and install the IIS web server.

If you don't have an Azure subscription, create a free account before you begin.

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to https://shell.azure.com/bash. Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Create a resource group

Create a resource group with the az group create command. An Azure resource group is a logical container into which Azure resources are deployed and managed. The following example creates a resource group named *myResourceGroup* in the *West US 3* location. Replace the value of the variables as needed.

Azure CLICopy
Open Cloudshell

```
resourcegroup="myResourceGroupCLI"
location="westus3"
az group create --name $resourcegroup --location $location
```

Create virtual machine

Create a VM with az vm create. The following example creates a VM named *myVM*. This example uses *azureuser* for an administrative user name. Replace the values of the variables as needed.

You'll be prompted to supply a password that meets the password requirements for Azure VMs.

Using the example below, you'll be prompted to enter a password at the command line. You could also add the `--admin-password` parameter with a value for your password. The user name and password will be used when you connect to the VM.

Azure CLICopy
Open Cloudshell

```
vmname="myVM"
username="azureuser"
az vm create \
    --resource-group $resourcegroup \
    --name $vmname \
    --image Win2022AzureEditionCore \
    --public-ip-sku Standard \
    --admin-username $username
```

It takes a few minutes to create the VM and supporting resources. The following example output shows the VM create operation was successful.

OutputCopy

```
{
  "fqdns": "",
  "id":
"/subscriptions/<guid>/resourceGroups/myResourceGroup/providers/Microsoft.Compute/
virtualMachines/myVM",
  "location": "westus3",
  "macAddress": "00-0D-3A-23-9A-49",
  "powerState": "VM running",
  "privateIpAddress": "10.0.0.4",
  "publicIpAddress": "52.174.34.95",
  "resourceGroup": "myResourceGroupCLI"
  "zones": ""
}
```

Take a note your own `publicIpAddress` in the output when you create your VM. This IP address is used to access the VM later in this article.

Install web server

To see your VM in action, install the IIS web server.

Azure CLICopy
Open Cloudshell

```
az vm run-command invoke -g $resourcegroup \
   -n $vmname \
   --command-id RunPowerShellScript \
   --scripts "Install-WindowsFeature -name Web-Server -IncludeManagementTools"
```

Open port 80 for web traffic

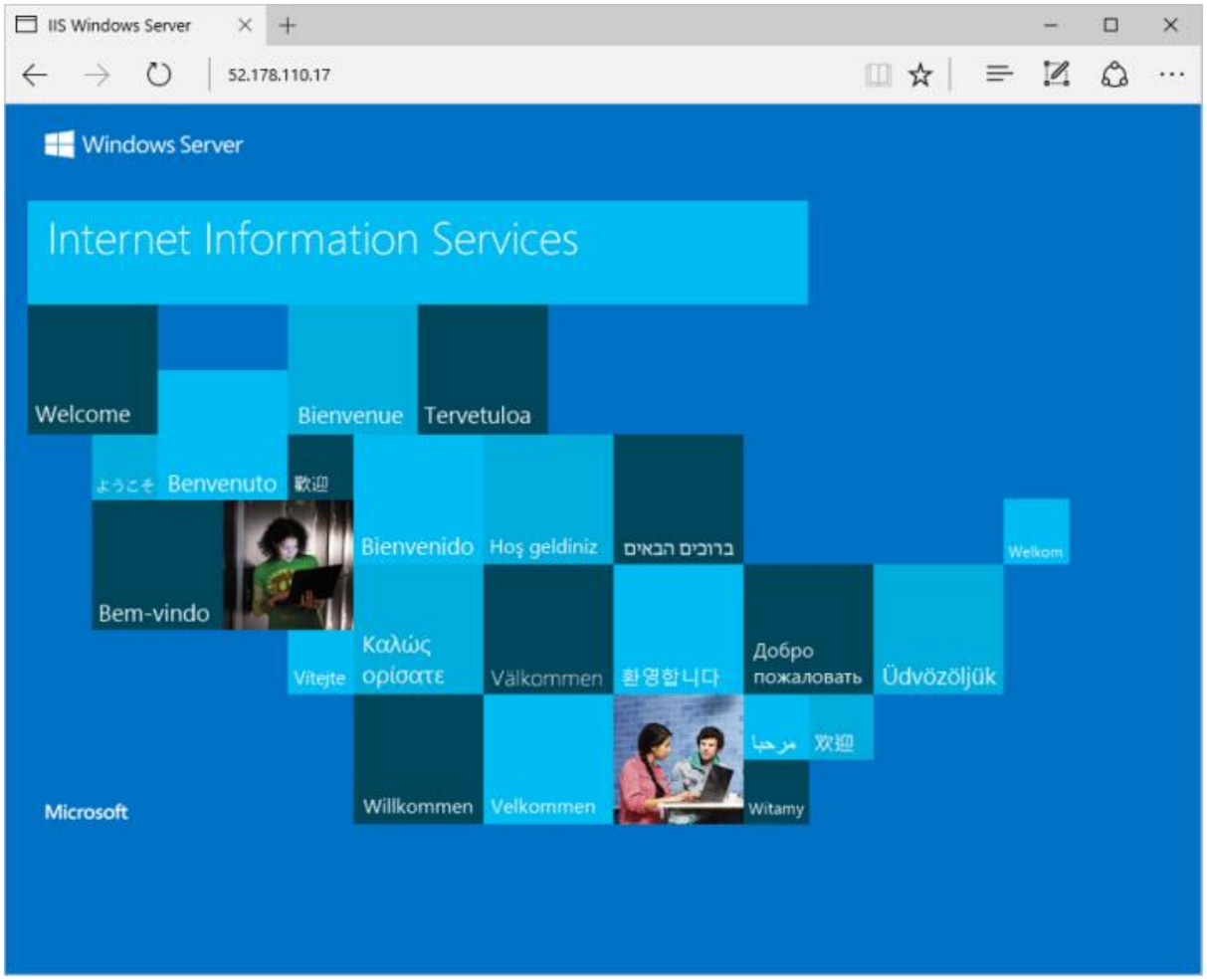
By default, only RDP connections are opened when you create a Windows VM in Azure. Use az vm open-port to open TCP port 80 for use with the IIS web server:

Azure CLICopy
Open Cloudshell

```
az vm open-port --port 80 --resource-group $resourcegroup --name $vmname
```

View the web server in action

With IIS installed and port 80 now open on your VM from the Internet, use a web browser of your choice to view the default IIS welcome page. Use the public IP address of your VM obtained in a previous step. The following example shows the default IIS web site:



Clean up resources

When no longer needed, you can use the az group delete command to remove the resource group, VM, and all related resources:

Azure CLICopy
Open Cloudshell

```
az group delete --name $resourcegroup
```

Next steps

In this quickstart, you deployed a simple virtual machine, open a network port for web traffic, and installed a basic web server. To learn more about Azure virtual machines, continue to the tutorial for Windows VMs.

Quickstart: Create a Windows virtual machine in the Azure portal

- Article
- 09/01/2022
- 20 contributors

Feedback

In this article

Applies to: ✔ Windows VMs

Azure virtual machines (VMs) can be created through the Azure portal. This method provides a browser-based user interface to create VMs and their associated resources. This quickstart shows you how to use the Azure portal to deploy a virtual machine (VM) in Azure that runs Windows Server 2019. To see your VM in action, you then RDP to the VM and install the IIS web server.

If you don't have an Azure subscription, create a free account before you begin.

Sign in to Azure

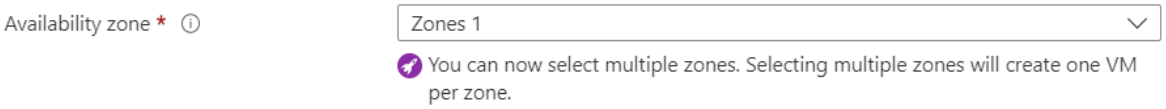
Sign in to the Azure portal at https://portal.azure.com.

Create virtual machine

1. Enter *virtual machines* in the search.
2. Under **Services**, select **Virtual machines**.
3. In the **Virtual machines** page, select **Create** and then **Azure virtual machine**. The **Create a virtual machine** page opens.
4. Under **Instance details**, enter *myVM* for the **Virtual machine name** and choose *Windows Server 2022 Datacenter - Gen 2* for the **Image**. Leave the other defaults.

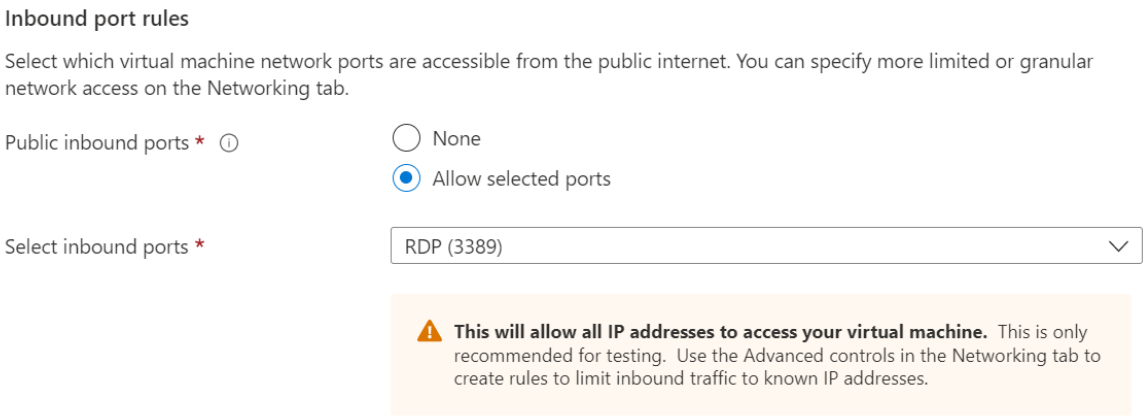
Note

Some users will now see the option to create VMs in multiple zones. To learn more about this new capability, see **Create virtual machines in an availability zone**.



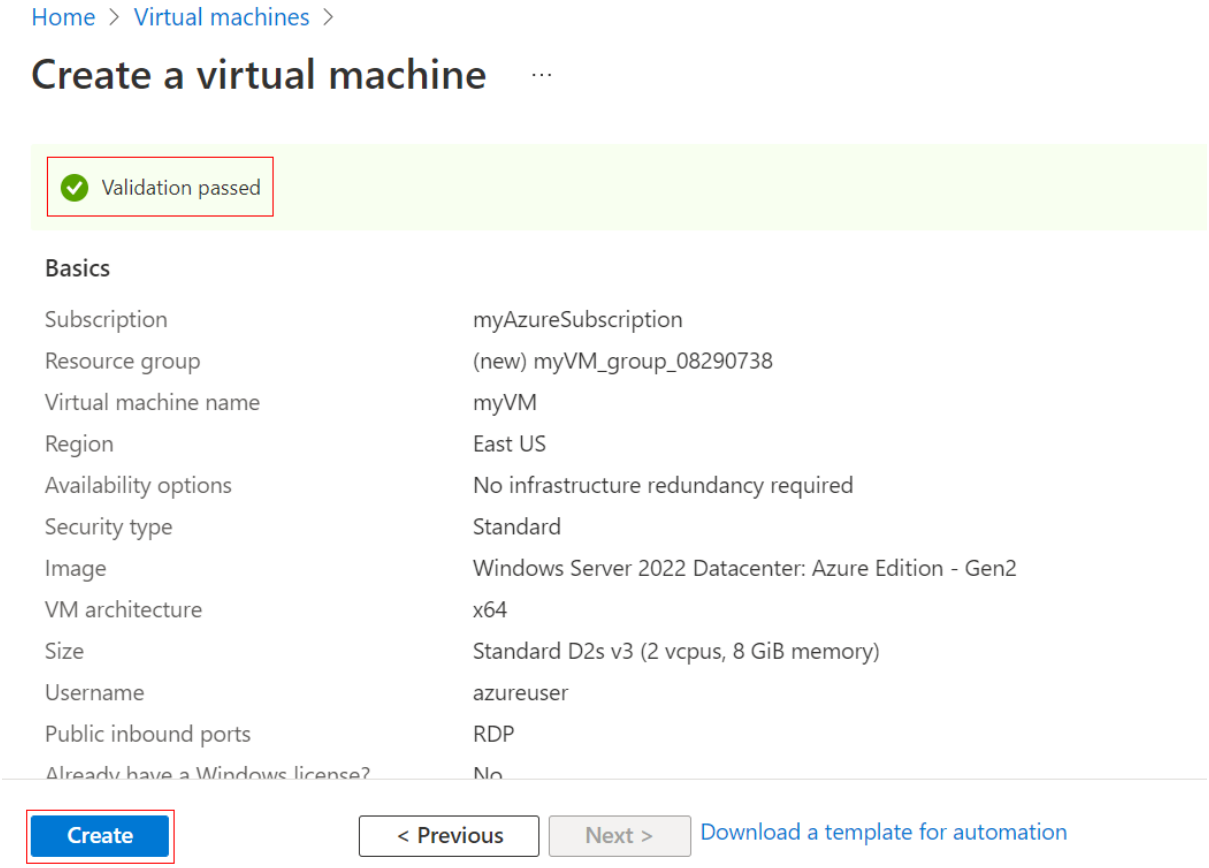
5. Under **Administrator account**, provide a username, such as *azureuser* and a password. The password must be at least 12 characters long and meet the defined complexity requirements.

6. Under **Inbound port rules**, choose **Allow selected ports** and then select **RDP (3389)** and **HTTP (80)** from the drop-down.



7. Leave the remaining defaults and then select the **Review + create** button at the bottom of the page.

8. After validation runs, select the **Create** button at the bottom of the page.



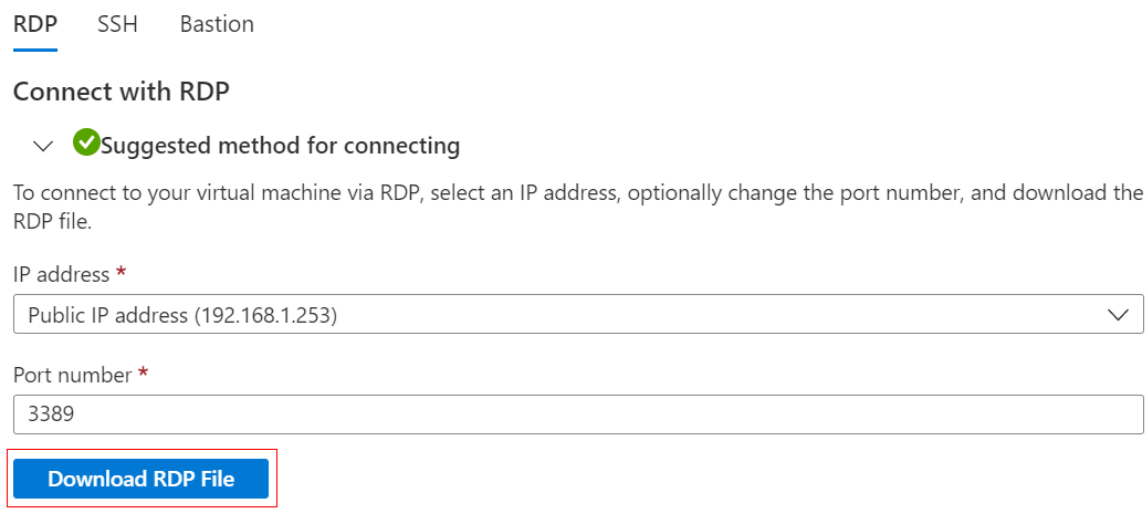
9. After deployment is complete, select **Go to resource**.

Connect to virtual machine

Create a remote desktop connection to the virtual machine. These directions tell you how to connect to your VM from a Windows computer. On a Mac, you need an RDP client such as this Remote Desktop Client from the Mac App Store.

1. On the overview page for your virtual machine, select the **Connect** > **RDP**.

2. In the **Connect with RDP** tab, keep the default options to connect by IP address, over port 3389, and click **Download RDP file**.



3. Open the downloaded RDP file and click **Connect** when prompted.
4. In the **Windows Security** window, select **More choices** and then **Use a different account**. Type the username as **localhost***username*, enter the password you created for the virtual machine, and then click **OK**.
5. You may receive a certificate warning during the sign-in process. Click **Yes** or **Continue** to create the connection.

Install web server

To see your VM in action, install the IIS web server. Open a PowerShell prompt on the VM and run the following command:

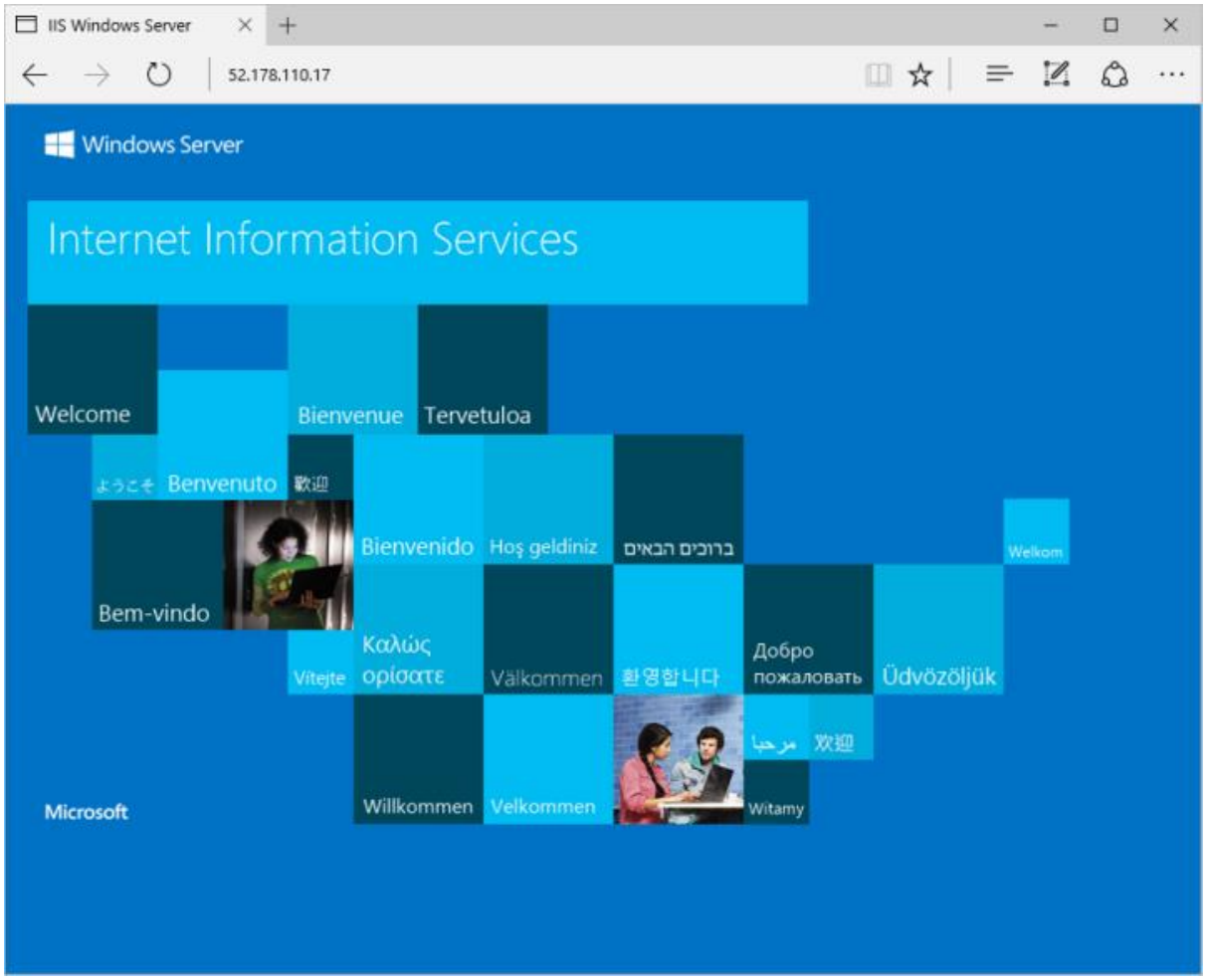
PowerShellCopy

```
Install-WindowsFeature -name Web-Server -IncludeManagementTools
```

When done, close the RDP connection to the VM.

View the IIS welcome page

In the portal, select the VM and in the overview of the VM, hover over the IP address to show **Copy to clipboard**. Copy the IP address and paste it into a browser tab. The default IIS welcome page will open, and should look like this:



Clean up resources

When no longer needed, you can delete the resource group, virtual machine, and all related resources.

1. On the Overview page for the VM, select the **Resource group** link.
2. At the top of the page for the resource group, select **Delete resource group**.
3. A page will open warning you that you are about to delete resources. Type the name of the resource group and select **Delete** to finish deleting the resources and the resource group.

Next steps

In this quickstart, you deployed a simple virtual machine, opened a network port for web traffic, and installed a basic web server. To learn more about Azure virtual machines, continue to the tutorial for Windows VMs.

Quickstart: Create a Windows virtual machine in Azure with PowerShell

- Article
- 04/29/2023
- 16 contributors

Feedback

In this article

Applies to: ✔ Windows VMs

The Azure PowerShell module is used to create and manage Azure resources from the PowerShell command line or in scripts. This quickstart shows you how to use the Azure PowerShell module to deploy a virtual machine (VM) in Azure that runs Windows Server 2016. You also bring Remote Desktop Portal (RDP) to the VM and install the IIS web server, to show the VM in action.

If you don't have an Azure subscription, create a free account before you begin.

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, just select **Open Cloudshell** from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to https://shell.azure.com/powershell. Select **Copy** to copy the code blocks, paste them into the Cloud Shell, and press enter to run the them.

Create resource group

Create an Azure resource group with New-AzResourceGroup. A resource group is a logical container into which Azure resources are deployed and managed.

Azure PowerShellCopy
Open Cloudshell

```
New-AzResourceGroup -Name 'myResourceGroup' -Location 'EastUS'
```

Create virtual machine

Create a VM with New-AzVM. Provide names for each of the resources and the `New-AzVM` cmdlet creates if they don't already exist.

When prompted, provide a username and password to be used as the sign-in credentials for the VM:

Azure PowerShellCopy
Open Cloudshell

```
New-AzVm `
    -ResourceGroupName 'myResourceGroup' `
    -Name 'myVM' `
    -Location 'East US' `
    -VirtualNetworkName 'myVnet' `
    -SubnetName 'mySubnet' `
    -SecurityGroupName 'myNetworkSecurityGroup' `
    -PublicIpAddressName 'myPublicIpAddress' `
    -OpenPorts 80,3389
```

Install web server

To see your VM in action, install the IIS web server. Open a PowerShell prompt on the VM and run the following command:

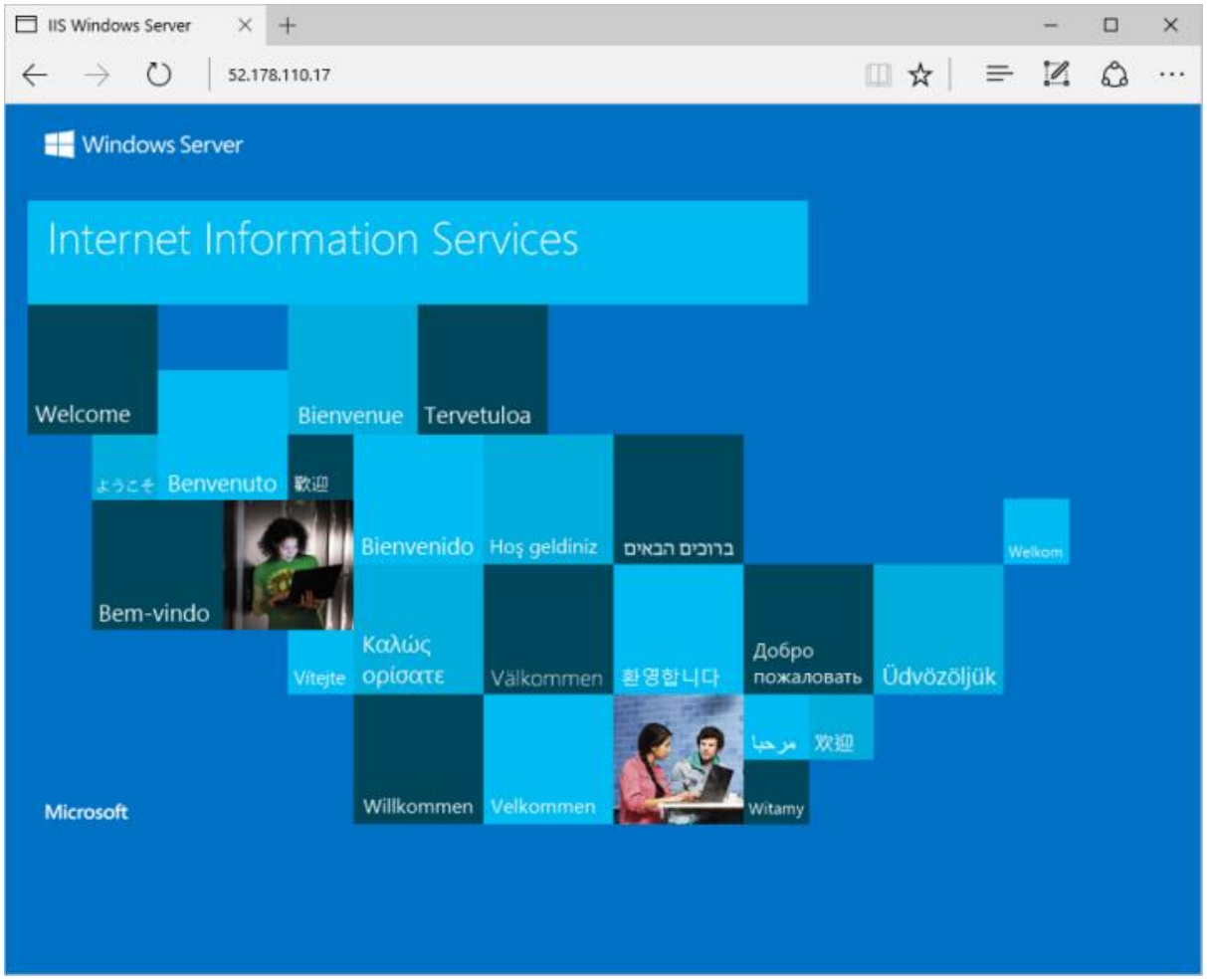
Azure PowerShellCopy
Open Cloudshell

```
Invoke-AzVMRunCommand -ResourceGroupName 'myResourceGroup' -VMName 'myVM' -
CommandId 'RunPowerShellScript' -ScriptString 'Install-WindowsFeature -Name Web-
Server -IncludeManagementTools'
```

The `-ScriptString` parameter requires version `4.27.0` or later of the `Az.Compute` module.

View the web server in action

With IIS installed and port 80 now open on your VM from the Internet, use a web browser of your choice to view the default IIS welcome page. Use the public IP address of the VM that you created. The following example shows the default IIS web site:



Clean up resources

When no longer needed, you can use the Remove-AzResourceGroup cmdlet to remove the resource group, VM, and all related resources:

Azure PowerShellCopy
Open Cloudshell

```
Remove-AzResourceGroup -Name 'myResourceGroup'
```

Next steps

In this quickstart, you deployed a simple virtual machine, opened a network port for web traffic, and installed a basic web server. To learn more about Azure virtual machines, continue to the tutorial for Windows VMs.

Quickstart: Use Terraform to create a Windows VM

- Article
- 02/08/2023
- 2 contributors

Feedback

In this article

Applies to: ✔ Windows VMs

Article tested with the following Terraform and Terraform provider versions:

- Terraform v1.2.7
- AzureRM Provider v.3.20.0

This article shows you how to create a complete Windows environment and supporting resources with Terraform. Those resources include a virtual network, subnet, public IP address, and more.

Terraform enables the definition, preview, and deployment of cloud infrastructure. Using Terraform, you create configuration files using HCL syntax. The HCL syntax allows you to specify the cloud provider - such as Azure - and the elements that make up your cloud infrastructure. After you create your configuration files, you create an *execution plan* that allows you to preview your infrastructure changes before they're deployed. Once you verify the changes, you apply the execution plan to deploy the infrastructure.

In this article, you learn how to:

- Create a virtual network
- Create a subnet
- Create a public IP address
- Create a network security group and SSH inbound rule
- Create a virtual network interface card

- Connect the network security group to the network interface
- Create a storage account for boot diagnostics
- Create a virtual machine with an IIS web server
- View the web server page

Note

The example code in this article is located in the **Microsoft Terraform GitHub repo**. See more **articles and sample code showing how to use Terraform to manage Azure resources**

Prerequisites

- **Azure subscription**: If you don't have an Azure subscription, create a free account before you begin.
- Install and configure Terraform

Implement the Terraform code

1. Create a directory in which to test the sample Terraform code and make it the current directory.
2. Create a file named `providers.tf` and insert the following code:

   TerraformCopy

```
terraform {
  required_version = ">=1.0"

  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "~>3.0"
    }
    random = {
      source  = "hashicorp/random"
      version = "~>3.0"
    }
  }
}

provider "azurerm" {
  features {}
}
```

3. Create a file named `main.tf` and insert the following code:

   TerraformCopy

```
resource "azurerm_resource_group" "rg" {
  location = var.resource_group_location
```

```
  name     = "${random_pet.prefix.id}-rg"
}

# Create virtual network
resource "azurerm_virtual_network" "my_terraform_network" {
  name                = "${random_pet.prefix.id}-vnet"
  address_space       = ["10.0.0.0/16"]
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
}

# Create subnet
resource "azurerm_subnet" "my_terraform_subnet" {
  name                 = "${random_pet.prefix.id}-subnet"
  resource_group_name  = azurerm_resource_group.rg.name
  virtual_network_name =
azurerm_virtual_network.my_terraform_network.name
  address_prefixes     = ["10.0.1.0/24"]
}

# Create public IPs
resource "azurerm_public_ip" "my_terraform_public_ip" {
  name                = "${random_pet.prefix.id}-public-ip"
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
  allocation_method   = "Dynamic"
}

# Create Network Security Group and rules
resource "azurerm_network_security_group" "my_terraform_nsg" {
  name                = "${random_pet.prefix.id}-nsg"
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name

  security_rule {
    name                       = "RDP"
    priority                   = 1000
    direction                  = "Inbound"
    access                     = "Allow"
    protocol                   = "*"
    source_port_range          = "*"
    destination_port_range     = "3389"
    source_address_prefix      = "*"
    destination_address_prefix = "*"
  }
  security_rule {
    name                       = "web"
    priority                   = 1001
    direction                  = "Inbound"
    access                     = "Allow"
    protocol                   = "Tcp"
    source_port_range          = "*"
    destination_port_range     = "80"
    source_address_prefix      = "*"
    destination_address_prefix = "*"
  }
}

# Create network interface
```

```
resource "azurerm_network_interface" "my_terraform_nic" {
  name                = "${random_pet.prefix.id}-nic"
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name

  ip_configuration {
    name                          = "my_nic_configuration"
    subnet_id                     =
azurerm_subnet.my_terraform_subnet.id
    private_ip_address_allocation = "Dynamic"
    public_ip_address_id          =
azurerm_public_ip.my_terraform_public_ip.id
  }
}

# Connect the security group to the network interface
resource "azurerm_network_interface_security_group_association"
"example" {
  network_interface_id      =
azurerm_network_interface.my_terraform_nic.id
  network_security_group_id =
azurerm_network_security_group.my_terraform_nsg.id
}

# Create storage account for boot diagnostics
resource "azurerm_storage_account" "my_storage_account" {
  name                     = "diag${random_id.random_id.hex}"
  location                 = azurerm_resource_group.rg.location
  resource_group_name      = azurerm_resource_group.rg.name
  account_tier             = "Standard"
  account_replication_type = "LRS"
}


# Create virtual machine
resource "azurerm_windows_virtual_machine" "main" {
  name                  = "${var.prefix}-vm"
  admin_username        = "azureuser"
  admin_password        = random_password.password.result
  location              = azurerm_resource_group.rg.location
  resource_group_name   = azurerm_resource_group.rg.name
  network_interface_ids =
[azurerm_network_interface.my_terraform_nic.id]
  size                  = "Standard_DS1_v2"

  os_disk {
    name                 = "myOsDisk"
    caching              = "ReadWrite"
    storage_account_type = "Premium_LRS"
  }

  source_image_reference {
    publisher = "MicrosoftWindowsServer"
    offer     = "WindowsServer"
    sku       = "2022-datacenter-azure-edition"
    version   = "latest"
  }
```

```
  boot_diagnostics {
    storage_account_uri =
azurerm_storage_account.my_storage_account.primary_blob_endpoint
  }
}

# Install IIS web server to the virtual machine
resource "azurerm_virtual_machine_extension" "web_server_install" {
  name                       = "${random_pet.prefix.id}-wsi"
  virtual_machine_id         = azurerm_windows_virtual_machine.main.id
  publisher                  = "Microsoft.Compute"
  type                       = "CustomScriptExtension"
  type_handler_version       = "1.8"
  auto_upgrade_minor_version = true

  settings = <<SETTINGS
    {
      "commandToExecute": "powershell -ExecutionPolicy Unrestricted
Install-WindowsFeature -Name Web-Server -IncludeAllSubFeature -
IncludeManagementTools"
    }
  SETTINGS
}

# Generate random text for a unique storage account name
resource "random_id" "random_id" {
  keepers = {
    # Generate a new ID only when a new resource group is defined
    resource_group = azurerm_resource_group.rg.name
  }

  byte_length = 8
}

resource "random_password" "password" {
  length      = 20
  min_lower   = 1
  min_upper   = 1
  min_numeric = 1
  min_special = 1
  special     = true
}

resource "random_pet" "prefix" {
  prefix = var.prefix
  length = 1
}
```

4. Create a file named `variables.tf` and insert the following code:

```
TerraformCopy
variable "resource_group_location" {
  default     = "eastus"
  description = "Location of the resource group."
}

variable "prefix" {
  type        = string
```

```
  default     = "win-vm-iis"
  description = "Prefix of the resource name"
}
```

5. Create a file named `outputs.tf` and insert the following code:

   TerraformCopy
   ```
   output "resource_group_name" {
     value = azurerm_resource_group.rg.name
   }

   output "public_ip_address" {
     value = azurerm_windows_virtual_machine.main.public_ip_address
   }

   output "admin_password" {
     sensitive = true
     value     = azurerm_windows_virtual_machine.main.admin_password
   }
   ```

Initialize Terraform

Run terraform init to initialize the Terraform deployment. This command downloads the Azure provider required to manage your Azure resources.

ConsoleCopy
```
terraform init -upgrade
```

Key points:

- The `-upgrade` parameter upgrades the necessary provider plugins to the newest version that complies with the configuration's version constraints.

Create a Terraform execution plan

Run terraform plan to create an execution plan.

ConsoleCopy
```
terraform plan -out main.tfplan
```

Key points:

- The `terraform plan` command creates an execution plan, but doesn't execute it. Instead, it determines what actions are necessary to create the configuration specified in your configuration files. This pattern allows you to verify whether the execution plan matches your expectations before making any changes to actual resources.

- The optional `-out` parameter allows you to specify an output file for the plan. Using the `-out` parameter ensures that the plan you reviewed is exactly what is applied.
- To read more about persisting execution plans and security, see the security warning section.

Apply a Terraform execution plan

Run terraform apply to apply the execution plan to your cloud infrastructure.

ConsoleCopy
```
terraform apply main.tfplan
```

Key points:

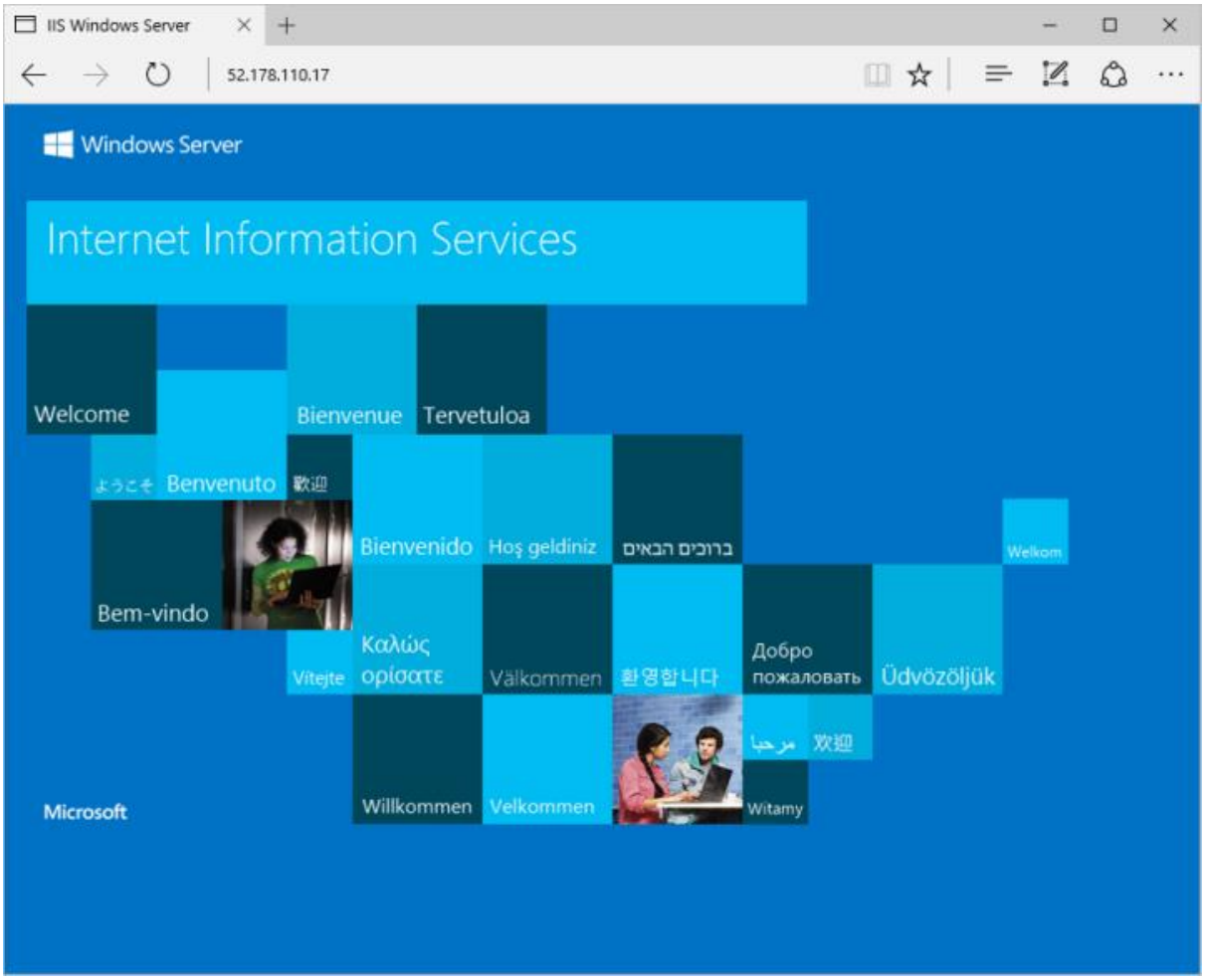
- The example `terraform apply` command assumes you previously ran `terraform plan -out main.tfplan`.
- If you specified a different filename for the `-out` parameter, use that same filename in the call to `terraform apply`.
- If you didn't use the `-out` parameter, call `terraform apply` without any parameters.

Verify the results

1. Run the following command to get the VM's public IP address and make note of it:

   Azure CLICopy
   Open Cloudshell
   ```
   echo $(terraform output -raw public_ip_address)
   ```

2. With IIS installed and port 80 now open on your VM from the Internet, use a web browser of your choice to view the default IIS welcome page. Use the public IP address of your VM obtained from the previous command. The following example shows the default IIS web site:



Clean up resources

When you no longer need the resources created via Terraform, do the following steps:

1. Run terraform plan and specify the `destroy` flag.

   ConsoleCopy
   ```
   terraform plan -destroy -out main.destroy.tfplan
   ```

   Key points:

   - The `terraform plan` command creates an execution plan, but doesn't execute it. Instead, it determines what actions are necessary to create the configuration specified in your configuration files. This pattern allows you to verify whether the execution plan matches your expectations before making any changes to actual resources.

- The optional `-out` parameter allows you to specify an output file for the plan. Using the `-out` parameter ensures that the plan you reviewed is exactly what is applied.
- To read more about persisting execution plans and security, see the security warning section.

2. Run terraform apply to apply the execution plan.

   ConsoleCopy
   ```
   terraform apply main.destroy.tfplan
   ```

Troubleshoot Terraform on Azure

Troubleshoot common problems when using Terraform on Azure

Next steps

In this quickstart, you deployed a simple virtual machine using Terraform. To learn more about Azure virtual machines, continue to the tutorial for Linux VMs.

Quickstart: Create a Windows virtual machine using a Bicep file

- Article
- 03/31/2023
- 2 contributors

Feedback

In this article

Applies to: ✔ Windows VMs

This quickstart shows you how to use a Bicep file to deploy a Windows virtual machine (VM) in Azure.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

If you don't have an Azure subscription, create a free account before you begin.

Review the Bicep file

The Bicep file used in this quickstart is from Azure Quickstart Templates.

BicepCopy

```
@description('Username for the Virtual Machine.')
param adminUsername string

@description('Password for the Virtual Machine.')
@minLength(12)
@secure()
param adminPassword string

@description('Unique DNS Name for the Public IP used to access the Virtual
Machine.')
param dnsLabelPrefix string = toLower('${vmName}-
${uniqueString(resourceGroup().id, vmName)}')

@description('Name for the Public IP used to access the Virtual Machine.')
param publicIpName string = 'myPublicIP'

@description('Allocation method for the Public IP used to access the Virtual
Machine.')
@allowed([
  'Dynamic'
  'Static'
])
param publicIPAllocationMethod string = 'Dynamic'

@description('SKU for the Public IP used to access the Virtual Machine.')
@allowed([
  'Basic'
  'Standard'
])
param publicIpSku string = 'Basic'

@description('The Windows version for the VM. This will pick a fully patched image
of this given Windows version.')
@allowed([
```

```
  '2016-datacenter-gensecond'
  '2016-datacenter-server-core-g2'
  '2016-datacenter-server-core-smalldisk-g2'
  '2016-datacenter-smalldisk-g2'
  '2016-datacenter-with-containers-g2'
  '2016-datacenter-zhcn-g2'
  '2019-datacenter-core-g2'
  '2019-datacenter-core-smalldisk-g2'
  '2019-datacenter-core-with-containers-g2'
  '2019-datacenter-core-with-containers-smalldisk-g2'
  '2019-datacenter-gensecond'
  '2019-datacenter-smalldisk-g2'
  '2019-datacenter-with-containers-g2'
  '2019-datacenter-with-containers-smalldisk-g2'
  '2019-datacenter-zhcn-g2'
  '2022-datacenter-azure-edition'
  '2022-datacenter-azure-edition-core'
  '2022-datacenter-azure-edition-core-smalldisk'
  '2022-datacenter-azure-edition-smalldisk'
  '2022-datacenter-core-g2'
  '2022-datacenter-core-smalldisk-g2'
  '2022-datacenter-g2'
  '2022-datacenter-smalldisk-g2'
])
param OSVersion string = '2022-datacenter-azure-edition'

@description('Size of the virtual machine.')
param vmSize string = 'Standard_D2s_v5'

@description('Location for all resources.')
param location string = resourceGroup().location

@description('Name of the virtual machine.')
param vmName string = 'simple-vm'

@description('Security Type of the Virtual Machine.')
@allowed([
  'Standard'
  'TrustedLaunch'
])
param securityType string = 'TrustedLaunch'

var storageAccountName = 'bootdiags${uniqueString(resourceGroup().id)}'
var nicName = 'myVMNic'
var addressPrefix = '10.0.0.0/16'
var subnetName = 'Subnet'
var subnetPrefix = '10.0.0.0/24'
var virtualNetworkName = 'MyVNET'
var networkSecurityGroupName = 'default-NSG'
var securityProfileJson = {
  uefiSettings: {
    secureBootEnabled: true
    vTpmEnabled: true
  }
  securityType: securityType
}
var extensionName = 'GuestAttestation'
var extensionPublisher = 'Microsoft.Azure.Security.WindowsAttestation'
var extensionVersion = '1.0'
```

```
var maaTenantName = 'GuestAttestation'
var maaEndpoint = substring('emptyString', 0, 0)

resource storageAccount 'Microsoft.Storage/storageAccounts@2022-05-01' = {
  name: storageAccountName
  location: location
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'Storage'
}

resource publicIp 'Microsoft.Network/publicIPAddresses@2022-05-01' = {
  name: publicIpName
  location: location
  sku: {
    name: publicIpSku
  }
  properties: {
    publicIPAllocationMethod: publicIPAllocationMethod
    dnsSettings: {
      domainNameLabel: dnsLabelPrefix
    }
  }
}

resource networkSecurityGroup 'Microsoft.Network/networkSecurityGroups@2022-05-01'
= {
  name: networkSecurityGroupName
  location: location
  properties: {
    securityRules: [
      {
        name: 'default-allow-3389'
        properties: {
          priority: 1000
          access: 'Allow'
          direction: 'Inbound'
          destinationPortRange: '3389'
          protocol: 'Tcp'
          sourcePortRange: '*'
          sourceAddressPrefix: '*'
          destinationAddressPrefix: '*'
        }
      }
    ]
  }
}

resource virtualNetwork 'Microsoft.Network/virtualNetworks@2022-05-01' = {
  name: virtualNetworkName
  location: location
  properties: {
    addressSpace: {
      addressPrefixes: [
        addressPrefix
      ]
    }
    subnets: [
```

```
      {
        name: subnetName
        properties: {
          addressPrefix: subnetPrefix
          networkSecurityGroup: {
            id: networkSecurityGroup.id
          }
        }
      }
    ]
  }
}

resource nic 'Microsoft.Network/networkInterfaces@2022-05-01' = {
  name: nicName
  location: location
  properties: {
    ipConfigurations: [
      {
        name: 'ipconfig1'
        properties: {
          privateIPAllocationMethod: 'Dynamic'
          publicIPAddress: {
            id: publicIp.id
          }
          subnet: {
            id: resourceId('Microsoft.Network/virtualNetworks/subnets',
virtualNetworkName, subnetName)
          }
        }
      }
    ]
  }
  dependsOn: [

    virtualNetwork
  ]
}

resource vm 'Microsoft.Compute/virtualMachines@2022-03-01' = {
  name: vmName
  location: location
  properties: {
    hardwareProfile: {
      vmSize: vmSize
    }
    osProfile: {
      computerName: vmName
      adminUsername: adminUsername
      adminPassword: adminPassword
    }
    storageProfile: {
      imageReference: {
        publisher: 'MicrosoftWindowsServer'
        offer: 'WindowsServer'
        sku: OSVersion
        version: 'latest'
      }
      osDisk: {
```

```
        createOption: 'FromImage'
        managedDisk: {
          storageAccountType: 'StandardSSD_LRS'
        }
      }
      dataDisks: [
        {
          diskSizeGB: 1023
          lun: 0
          createOption: 'Empty'
        }
      ]
    }
    networkProfile: {
      networkInterfaces: [
        {
          id: nic.id
        }
      ]
    }
    diagnosticsProfile: {
      bootDiagnostics: {
        enabled: true
        storageUri: storageAccount.properties.primaryEndpoints.blob
      }
    }
    securityProfile: ((securityType == 'TrustedLaunch') ? securityProfileJson :
null)
  }
}

resource vmExtension 'Microsoft.Compute/virtualMachines/extensions@2022-03-01' =
if ((securityType == 'TrustedLaunch') &&
((securityProfileJson.uefiSettings.secureBootEnabled == true) &&
(securityProfileJson.uefiSettings.vTpmEnabled == true))) {
  parent: vm
  name: extensionName
  location: location
  properties: {
    publisher: extensionPublisher
    type: extensionName
    typeHandlerVersion: extensionVersion
    autoUpgradeMinorVersion: true
    enableAutomaticUpgrade: true
    settings: {
      AttestationConfig: {
        MaaSettings: {
          maaEndpoint: maaEndpoint
          maaTenantName: maaTenantName
        }
      }
    }
  }
}

output hostname string = publicIp.properties.dnsSettings.fqdn
```

Several resources are defined in the Bicep file:

- **Microsoft.Network/virtualNetworks/subnets**: create a subnet.
- **Microsoft.Storage/storageAccounts**: create a storage account.
- **Microsoft.Network/publicIPAddresses**: create a public IP address.
- **Microsoft.Network/networkSecurityGroups**: create a network security group.
- **Microsoft.Network/virtualNetworks**: create a virtual network.
- **Microsoft.Network/networkInterfaces**: create a NIC.
- **Microsoft.Compute/virtualMachines**: create a virtual machine.

Deploy the Bicep file

1. Save the Bicep file as **main.bicep** to your local computer.
2. Deploy the Bicep file using either Azure CLI or Azure PowerShell.
   - CLI
   - PowerShell

   Azure CLICopy

```
az group create --name exampleRG --location eastus
az deployment group create --resource-group exampleRG --template-file
main.bicep --parameters adminUsername=<admin-username>
```

Note

Replace **<admin-username>** with a unique username. You'll also be prompted to enter adminPassword. The minimum password length is 12 characters.

When the deployment finishes, you should see a messaged indicating the deployment succeeded.

Review deployed resources

Use the Azure portal, Azure CLI, or Azure PowerShell to list the deployed resources in the resource group.

- CLI
- PowerShell

Azure CLICopy

Open Cloudshell

```
az resource list --resource-group exampleRG
```

Clean up resources

When no longer needed, use the Azure portal, Azure CLI, or Azure PowerShell to delete the VM and all of the resources in the resource group.

- CLI
- PowerShell

Azure CLICopy
Open Cloudshell

```
az group delete --name exampleRG
```

Next steps

In this quickstart, you deployed a simple virtual machine using a Bicep file. To learn more about Azure virtual machines, continue to the tutorial for Linux VMs.

Quickstart: Create a Windows virtual machine using a template

- Article
- 04/05/2023
- 8 contributors

Feedback

In this article

1. Prerequisites
2. Review the template
3. Deploy the template
4. Review deployed resources

Show 2 more

Applies to: ✔ Windows VMs

This quickstart shows you how to use an Azure Resource Manager template to deploy a Windows virtual machine (VM) in Azure.

A resource manager template is a JavaScript Object Notation (JSON) file that defines the infrastructure and configuration for your project. The template uses declarative syntax. In declarative syntax, you describe your intended deployment without writing the sequence of programming commands to create the deployment.

If your environment meets the prerequisites and you're familiar with using templates, select the **Deploy to Azure** button. The template opens in the Azure portal.

Prerequisites

If you don't have an Azure subscription, create a free account before you begin.

Review the template

The template used in this quickstart is from Azure Quickstart Templates.

JSONCopy

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-
01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "metadata": {
    "_generator": {
      "name": "bicep",
      "version": "0.16.2.56959",
      "templateHash": "14427937023370378081"
    }
  },
  "parameters": {
    "adminUsername": {
      "type": "string",
      "metadata": {
        "description": "Username for the Virtual Machine."
      }
    },
    "adminPassword": {
      "type": "securestring",
      "minLength": 12,
      "metadata": {
        "description": "Password for the Virtual Machine."
      }
    },
    "dnsLabelPrefix": {
      "type": "string",
      "defaultValue": "[toLower(format('{0}-{1}', parameters('vmName'),
uniqueString(resourceGroup().id, parameters('vmName'))))]",
      "metadata": {
        "description": "Unique DNS Name for the Public IP used to access the
Virtual Machine."
      }
    },
    "publicIpName": {
      "type": "string",
      "defaultValue": "myPublicIP",
```

```
      "metadata": {
        "description": "Name for the Public IP used to access the Virtual
Machine."
      }
    },
    "publicIPAllocationMethod": {
      "type": "string",
      "defaultValue": "Dynamic",
      "allowedValues": [
        "Dynamic",
        "Static"
      ],
      "metadata": {
        "description": "Allocation method for the Public IP used to access the
Virtual Machine."
      }
    },
    "publicIpSku": {
      "type": "string",
      "defaultValue": "Basic",
      "allowedValues": [
        "Basic",
        "Standard"
      ],
      "metadata": {
        "description": "SKU for the Public IP used to access the Virtual Machine."
      }
    },
    "OSVersion": {
      "type": "string",
      "defaultValue": "2022-datacenter-azure-edition",
      "allowedValues": [
        "2016-datacenter-gensecond",
        "2016-datacenter-server-core-g2",
        "2016-datacenter-server-core-smalldisk-g2",
        "2016-datacenter-smalldisk-g2",
        "2016-datacenter-with-containers-g2",
        "2016-datacenter-zhcn-g2",
        "2019-datacenter-core-g2",
        "2019-datacenter-core-smalldisk-g2",
        "2019-datacenter-core-with-containers-g2",
        "2019-datacenter-core-with-containers-smalldisk-g2",
        "2019-datacenter-gensecond",
        "2019-datacenter-smalldisk-g2",
        "2019-datacenter-with-containers-g2",
        "2019-datacenter-with-containers-smalldisk-g2",
        "2019-datacenter-zhcn-g2",
        "2022-datacenter-azure-edition",
        "2022-datacenter-azure-edition-core",
        "2022-datacenter-azure-edition-core-smalldisk",
        "2022-datacenter-azure-edition-smalldisk",
        "2022-datacenter-core-g2",
        "2022-datacenter-core-smalldisk-g2",
        "2022-datacenter-g2",
        "2022-datacenter-smalldisk-g2"
      ],
      "metadata": {
        "description": "The Windows version for the VM. This will pick a fully
patched image of this given Windows version."
```

```
      }
    },
    "vmSize": {
      "type": "string",
      "defaultValue": "Standard_D2s_v5",
      "metadata": {
        "description": "Size of the virtual machine."
      }
    },
    "location": {
      "type": "string",
      "defaultValue": "[resourceGroup().location]",
      "metadata": {
        "description": "Location for all resources."
      }
    },
    "vmName": {
      "type": "string",
      "defaultValue": "simple-vm",
      "metadata": {
        "description": "Name of the virtual machine."
      }
    },
    "securityType": {
      "type": "string",
      "defaultValue": "TrustedLaunch",
      "allowedValues": [
        "Standard",
        "TrustedLaunch"
      ],
      "metadata": {
        "description": "Security Type of the Virtual Machine."
      }
    }
  },
  "variables": {
    "storageAccountName": "[format('bootdiags{0}',
uniqueString(resourceGroup().id))]",
    "nicName": "myVMNic",
    "addressPrefix": "10.0.0.0/16",
    "subnetName": "Subnet",
    "subnetPrefix": "10.0.0.0/24",
    "virtualNetworkName": "MyVNET",
    "networkSecurityGroupName": "default-NSG",
    "securityProfileJson": {
      "uefiSettings": {
        "secureBootEnabled": true,
        "vTpmEnabled": true
      },
      "securityType": "[parameters('securityType')]"
    },
    "extensionName": "GuestAttestation",
    "extensionPublisher": "Microsoft.Azure.Security.WindowsAttestation",
    "extensionVersion": "1.0",
    "maaTenantName": "GuestAttestation",
    "maaEndpoint": "[substring('emptyString', 0, 0)]"
  },
  "resources": [
    {
```

```
    "type": "Microsoft.Storage/storageAccounts",
    "apiVersion": "2022-05-01",
    "name": "[variables('storageAccountName')]",
    "location": "[parameters('location')]",
    "sku": {
      "name": "Standard_LRS"
    },
    "kind": "Storage"
  },
  {
    "type": "Microsoft.Network/publicIPAddresses",
    "apiVersion": "2022-05-01",
    "name": "[parameters('publicIpName')]",
    "location": "[parameters('location')]",
    "sku": {
      "name": "[parameters('publicIpSku')]"
    },
    "properties": {
      "publicIPAllocationMethod": "[parameters('publicIPAllocationMethod')]",
      "dnsSettings": {
        "domainNameLabel": "[parameters('dnsLabelPrefix')]"
      }
    }
  },
  {
    "type": "Microsoft.Network/networkSecurityGroups",
    "apiVersion": "2022-05-01",
    "name": "[variables('networkSecurityGroupName')]",
    "location": "[parameters('location')]",
    "properties": {
      "securityRules": [
        {
          "name": "default-allow-3389",
          "properties": {
            "priority": 1000,
            "access": "Allow",
            "direction": "Inbound",
            "destinationPortRange": "3389",
            "protocol": "Tcp",
            "sourcePortRange": "*",
            "sourceAddressPrefix": "*",
            "destinationAddressPrefix": "*"
          }
        }
      ]
    }
  },
  {
    "type": "Microsoft.Network/virtualNetworks",
    "apiVersion": "2022-05-01",
    "name": "[variables('virtualNetworkName')]",
    "location": "[parameters('location')]",
    "properties": {
      "addressSpace": {
        "addressPrefixes": [
          "[variables('addressPrefix')]"
        ]
      },
      "subnets": [
```

```
          {
            "name": "[variables('subnetName')]",
            "properties": {
              "addressPrefix": "[variables('subnetPrefix')]",
              "networkSecurityGroup": {
                "id": "[resourceId('Microsoft.Network/networkSecurityGroups',
variables('networkSecurityGroupName'))]"
              }
            }
          }
        ]
      },
      "dependsOn": [
        "[resourceId('Microsoft.Network/networkSecurityGroups',
variables('networkSecurityGroupName'))]"
      ]
    },
    {
      "type": "Microsoft.Network/networkInterfaces",
      "apiVersion": "2022-05-01",
      "name": "[variables('nicName')]",
      "location": "[parameters('location')]",
      "properties": {
        "ipConfigurations": [
          {
            "name": "ipconfig1",
            "properties": {
              "privateIPAllocationMethod": "Dynamic",
              "publicIPAddress": {
                "id": "[resourceId('Microsoft.Network/publicIPAddresses',
parameters('publicIpName'))]"
              },
              "subnet": {
                "id": "[resourceId('Microsoft.Network/virtualNetworks/subnets',
variables('virtualNetworkName'), variables('subnetName'))]"
              }
            }
          }
        ]
      },
      "dependsOn": [
        "[resourceId('Microsoft.Network/publicIPAddresses',
parameters('publicIpName'))]",
        "[resourceId('Microsoft.Network/virtualNetworks',
variables('virtualNetworkName'))]"
      ]
    },
    {
      "type": "Microsoft.Compute/virtualMachines",
      "apiVersion": "2022-03-01",
      "name": "[parameters('vmName')]",
      "location": "[parameters('location')]",
      "properties": {
        "hardwareProfile": {
          "vmSize": "[parameters('vmSize')]"
        },
        "osProfile": {
          "computerName": "[parameters('vmName')]",
          "adminUsername": "[parameters('adminUsername')]",
```

```
          "adminPassword": "[parameters('adminPassword')]"
        },
        "storageProfile": {
          "imageReference": {
            "publisher": "MicrosoftWindowsServer",
            "offer": "WindowsServer",
            "sku": "[parameters('OSVersion')]",
            "version": "latest"
          },
          "osDisk": {
            "createOption": "FromImage",
            "managedDisk": {
              "storageAccountType": "StandardSSD_LRS"
            }
          },
          "dataDisks": [
            {
              "diskSizeGB": 1023,
              "lun": 0,
              "createOption": "Empty"
            }
          ]
        },
        "networkProfile": {
          "networkInterfaces": [
            {
              "id": "[resourceId('Microsoft.Network/networkInterfaces',
variables('nicName'))]"
            }
          ]
        },
        "diagnosticsProfile": {
          "bootDiagnostics": {
            "enabled": true,
            "storageUri":
"[reference(resourceId('Microsoft.Storage/storageAccounts',
variables('storageAccountName')), '2022-05-01').primaryEndpoints.blob]"
          }
        },
        "securityProfile": "[if(equals(parameters('securityType'),
'TrustedLaunch'), variables('securityProfileJson'), null())]"
      },
      "dependsOn": [
        "[resourceId('Microsoft.Network/networkInterfaces',
variables('nicName'))]",
        "[resourceId('Microsoft.Storage/storageAccounts',
variables('storageAccountName'))]"
      ]
    },
    {
      "condition": "[and(equals(parameters('securityType'), 'TrustedLaunch'),
and(equals(variables('securityProfileJson').uefiSettings.secureBootEnabled,
true()), equals(variables('securityProfileJson').uefiSettings.vTpmEnabled,
true())))]",
      "type": "Microsoft.Compute/virtualMachines/extensions",
      "apiVersion": "2022-03-01",
      "name": "[format('{0}/{1}', parameters('vmName'),
variables('extensionName'))]",
      "location": "[parameters('location')]",
```

```
      "properties": {
        "publisher": "[variables('extensionPublisher')]",
        "type": "[variables('extensionName')]",
        "typeHandlerVersion": "[variables('extensionVersion')]",
        "autoUpgradeMinorVersion": true,
        "enableAutomaticUpgrade": true,
        "settings": {
          "AttestationConfig": {
            "MaaSettings": {
              "maaEndpoint": "[variables('maaEndpoint')]",
              "maaTenantName": "[variables('maaTenantName')]"
            }
          }
        }
      },
      "dependsOn": [
        "[resourceId('Microsoft.Compute/virtualMachines', parameters('vmName'))]"
      ]
    }
  ],
  "outputs": {
    "hostname": {
      "type": "string",
      "value": "[reference(resourceId('Microsoft.Network/publicIPAddresses',
parameters('publicIpName')), '2022-05-01').dnsSettings.fqdn]"
    }
  }
}
```

Several resources are defined in the template:

- **Microsoft.Network/virtualNetworks/subnets**: create a subnet.
- **Microsoft.Storage/storageAccounts**: create a storage account.
- **Microsoft.Network/publicIPAddresses**: create a public IP address.
- **Microsoft.Network/networkSecurityGroups**: create a network security group.
- **Microsoft.Network/virtualNetworks**: create a virtual network.
- **Microsoft.Network/networkInterfaces**: create a NIC.
- **Microsoft.Compute/virtualMachines**: create a virtual machine.

Deploy the template

1. Select the following image to sign in to Azure and open a template. The template creates a key vault and a secret.

2. Select or enter the following values. Use the default values, when available.
    - **Subscription**: select an Azure subscription.

- **Resource group**: select an existing resource group from the drop-down, or select **Create new**, enter a unique name for the resource group, and then select **OK**.
- **Region**: select a region. For example, **Central US**.
- **Admin username**: provide a username, such as *azureuser*.
- **Admin password**: provide a password to use for the admin account. The password must be at least 12 characters long and meet the defined complexity requirements.
- **DNS label prefix**: enter a unique identifier to use as part of the DNS label.
- **OS version**: select which OS version you want to run on the VM.
- **VM size**: select the size to use for the VM.
- **Location**: the default is the same location as the resource group, if it already exists.

3. Select **Review + create**. After validation completes, select **Create** to create and deploy the VM.

The Azure portal is used to deploy the template. In addition to the Azure portal, you can also use the Azure PowerShell, Azure CLI, and REST API. To learn other deployment methods, see Deploy templates.

Review deployed resources

You can use the Azure portal to check on the VM and other resource that were created. After the deployment is finished, select **Resource groups** to see the VM and other resources.

Clean up resources

When no longer needed, delete the resource group, which deletes the VM and all of the resources in the resource group.

1. Select the **Resource group**.
2. On the page for the resource group, select **Delete resource group**.
3. When prompted, type the name of the resource group and then select **Delete**.

Next steps

In this quickstart, you deployed a simple virtual machine using a Resource Manager template. To learn more about Azure virtual machines, continue to the tutorial for managing VMs.

Back up a virtual machine in Azure with the Azure CLI

- Article
- 02/02/2023
- 15 contributors

Feedback

In this article

The Azure CLI is used to create and manage Azure resources from the command line or in scripts. You can protect your data by taking backups at regular intervals. Azure Backup creates recovery points that can be stored in geo-redundant recovery vaults. This article details how to back up a virtual machine (VM) in Azure with the Azure CLI. You can also perform these steps with Azure PowerShell or in the Azure portal.

This quickstart enables backup on an existing Azure VM. If you need to create a VM, you can create a VM with the Azure CLI.

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Quickstart for Bash in Azure Cloud Shell.

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.

    - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
    - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
    - Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.

- This quickstart requires version 2.0.18 or later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

Create a Recovery Services vault

A Recovery Services vault is a logical container that stores the backup data for each protected resource, such as Azure VMs. When the backup job for a protected resource runs, it creates a recovery point inside the Recovery Services vault. You can then use one of these recovery points to restore data to a given point in time.

Create a Recovery Services vault with az backup vault create. Specify the same resource group and location as the VM you wish to protect. If you used the VM quickstart, then you created:

- a resource group named *myResourceGroup*,
- a VM named *myVM*,
- resources in the *eastus* location.

Azure CLICopy
Open Cloudshell

```
az backup vault create --resource-group myResourceGroup \
    --name myRecoveryServicesVault \
    --location eastus
```

By default, the Recovery Services vault is set for Geo-Redundant storage. Geo-Redundant storage ensures your backup data is replicated to a secondary Azure region that's hundreds of miles away from the primary region. If the storage redundancy setting needs to be modified, use az backup vault backup-properties set cmdlet.

Azure CLICopy

```
az backup vault backup-properties set \
    --name myRecoveryServicesVault  \
    --resource-group myResourceGroup \
    --backup-storage-redundancy "LocallyRedundant/GeoRedundant"
```

Enable backup for an Azure VM

Create a protection policy to define: when a backup job runs, and how long the recovery points are stored. The default protection policy runs a backup job each day and retains recovery points for 30 days. You can use these default policy values to quickly protect your VM. To enable backup protection for a VM, use az backup protection enable-for-vm. Specify the resource group and VM to protect, then the policy to use:

Azure CLICopy
Open Cloudshell

```
az backup protection enable-for-vm \
    --resource-group myResourceGroup \
    --vault-name myRecoveryServicesVault \
    --vm myVM \
    --policy-name DefaultPolicy
```

Note

If the VM isn't in the same resource group as that of the vault, then myResourceGroup refers to the resource group where vault was created. Instead of VM name, provide the VM ID as indicated below.

Azure CLICopy
Open Cloudshell

```
az backup protection enable-for-vm \
    --resource-group myResourceGroup \
    --vault-name myRecoveryServicesVault \
    --vm $(az vm show -g VMResourceGroup -n MyVm --query id | tr -d '"') \
    --policy-name DefaultPolicy
```

Important

While using CLI to enable backup for multiple VMs at once, ensure that a single policy doesn't have more than 100 VMs associated with it. This is a **recommended best practice**. Currently, the PowerShell client doesn't explicitly block if there are more than 100 VMs, but the check is planned to be added in the future.

Prerequisites to backup encrypted VMs

To enable protection on encrypted VMs (encrypted using BEK and KEK), you must provide the Azure Backup service permission to read keys and secrets from the key vault. To do so, set a *keyvault* access policy with the required permissions, as demonstrated below:

Azure CLICopy
Open Cloudshell

```
# Enter the name of the resource group where the key vault is located on this
variable
AZ_KEYVAULT_RGROUP=TestKeyVaultRG

# Enter the name of the key vault on this variable
AZ_KEYVAULT_NAME=TestKeyVault

# Get the object id for the Backup Management Service on your subscription
AZ_ABM_OBJECT_ID=$( az ad sp list --display-name "Backup Management Service" --
query '[].objectId' -o tsv --only-show-errors )

# This command will grant the permissions required by the Backup Management
Service to access the key vault
az keyvault set-policy --key-permissions get list backup --secret-permissions get
list backup \
  --resource-group $AZ_KEYVAULT_RGROUP --name $AZ_KEYVAULT_NAME --object-id
$AZ_ABM_OBJECT_ID
```

Start a backup job

To start a backup now rather than wait for the default policy to run the job at the scheduled time, use az backup protection backup-now. This first backup job creates a full recovery point. Each backup job after this initial backup creates incremental recovery points. Incremental recovery points are storage and time-efficient, as they only transfer changes made since the last backup.

The following parameters are used to back up the VM:

- `--container-name` is the name of your VM
- `--item-name` is the name of your VM
- `--retain-until` value should be set to the last available date, in UTC time format (**dd-mm-yyyy**), that you wish the recovery point to be available

The following example backs up the VM named *myVM* and sets the expiration of the recovery point to October 18, 2017:

Azure CLICopy
Open Cloudshell

```
az backup protection backup-now \
    --resource-group myResourceGroup \
    --vault-name myRecoveryServicesVault \
    --container-name myVM \
    --item-name myVM \
    --backup-management-type AzureIaaSVM
    --retain-until 18-10-2017
```

Monitor the backup job

To monitor the status of backup jobs, use az backup job list:

Azure CLICopy
Open Cloudshell
```
az backup job list \
    --resource-group myResourceGroup \
    --vault-name myRecoveryServicesVault \
    --output table
```

The output is similar to the following example, which shows the backup job is *InProgress*:

OutputCopy
```
Name      Operation        Status      Item Name    Start Time UTC       Duration
--------  ---------------  ----------  -----------  -------------------  ---------
-----
a0a8e5e6  Backup           InProgress  myvm         2017-09-19T03:09:21
0:00:48.718366
fe5d0414  ConfigureBackup  Completed   myvm         2017-09-19T03:03:57
0:00:31.191807
```

When the *Status* of the backup job reports *Completed*, your VM is protected with Recovery Services and has a full recovery point stored.

Clean up deployment

When no longer needed, you can disable protection on the VM, remove the restore points and Recovery Services vault, then delete the resource group and associated VM resources. If you used an existing VM, you can skip the final az group delete command to leave the resource group and VM in place.

If you want to try a Backup tutorial that explains how to restore data for your VM, go to Next steps.

Azure CLICopy
Open Cloudshell
```
az backup protection disable \
    --resource-group myResourceGroup \
    --vault-name myRecoveryServicesVault \
    --container-name myVM \
    --item-name myVM \
    --backup-management-type AzureIaaSVM
    --delete-backup-data true
az backup vault delete \
    --resource-group myResourceGroup \
    --name myRecoveryServicesVault \
az group delete --name myResourceGroup
```

Next steps

In this quickstart, you created a Recovery Services vault, enabled protection on a VM, and created the initial recovery point. To learn more about Azure Backup and Recovery Services, continue to the tutorials.

Back up a virtual machine in Azure with PowerShell

- Article
- 05/11/2023
- 10 contributors

Feedback

In this article

The Azure PowerShell AZ module is used to create and manage Azure resources from the command line or in scripts.

Azure Backup backs up on-premises machines and apps, and Azure VMs. This article shows you how to back up an Azure VM with the AZ module. Alternatively, you can back up a VM using the Azure CLI, or in the Azure portal.

This quickstart enables backup on an existing Azure VM. If you need to create a VM, you can create a VM with Azure PowerShell.

This quickstart requires the Azure PowerShell AZ module version 1.0.0 or later. Run `Get-Module -ListAvailable Az` to find the version. If you need to install or upgrade, see Install Azure PowerShell module.

Note

We recommend that you use the Azure Az PowerShell module to interact with Azure. See **Install Azure PowerShell** to get started. To learn how to migrate to the Az PowerShell module, see **Migrate Azure PowerShell from AzureRM to Az**.

Sign in and register

1. Sign in to your Azure subscription with the `Connect-AzAccount` command and follow the on-screen directions.

   PowerShellCopy
   ```
   Connect-AzAccount
   ```

2. The first time you use Azure Backup, you must register the Azure Recovery Service provider in your subscription with Register-AzResourceProvider, as follows:

   PowerShellCopy
   ```
   Register-AzResourceProvider -ProviderNamespace
   "Microsoft.RecoveryServices"
   ```

Create a Recovery Services vault

A Recovery Services vault is a logical container that stores backup data for protected resources, such as Azure VMs. When a backup job runs, it creates a recovery point inside the Recovery Services vault. You can then use one of these recovery points to restore data to a given point in time.

When you create the vault:

- For the resource group and location, specify the resource group and location of the VM you want to back up.
- If you used this sample script to create the VM, the resource group is **myResourceGroup**, the VM is ***myVM**, and the resources are in the **WestEurope** region.
- Azure Backup automatically handles storage for backed up data. By default the vault uses Geo-Redundant Storage (GRS). Geo-redundancy ensures that backed up data is replicated to a secondary Azure region, hundreds of miles away from the primary region.

Now create a vault:

1. Use the New-AzRecoveryServicesVault to create the vault:

   PowerShellCopy
   ```
   New-AzRecoveryServicesVault `
       -ResourceGroupName "myResourceGroup" `
       -Name "myRecoveryServicesVault" `
   -Location "WestEurope"
   ```

2. Set the vault context with Set-AzRecoveryServicesVaultContext, as follows:

   PowerShellCopy
   ```
   Get-AzRecoveryServicesVault `
       -Name "myRecoveryServicesVault" | Set-
   AzRecoveryServicesVaultContext
   ```

3. Change the storage redundancy configuration (LRS/GRS) of the vault with Set-AzRecoveryServicesBackupProperty, as follows:

   PowerShellCopy
   ```
   Get-AzRecoveryServicesVault `
       -Name "myRecoveryServicesVault" | Set-
   AzRecoveryServicesBackupProperty -BackupStorageRedundancy
   LocallyRedundant/GeoRedundant
   ```
   Note

   Storage Redundancy can be modified only if there are no backup items protected to the vault.

Enable backup for an Azure VM

You enable backup for an Azure VM, and specify a backup policy.

- The policy defines when backups run, and how long recovery points created by the backups should be retained.
- The default protection policy runs a backup once a day for the VM, and retains the created recovery points for 30 days. You can use this default policy to quickly protect your VM.

Enable backup as follows:

1. First, set the default policy with Get-AzRecoveryServicesBackupProtectionPolicy:

   PowerShellCopy
   ```
   $policy = Get-AzRecoveryServicesBackupProtectionPolicy     -Name
   "DefaultPolicy"
   ```

2. Enable VM backup with Enable-AzRecoveryServicesBackupProtection. Specify the policy, the resource group, and the VM name.

   PowerShellCopy
   ```
   Enable-AzRecoveryServicesBackupProtection `
       -ResourceGroupName "myResourceGroup" `
       -Name "myVM" `
   ```

```
    -Policy $policy
```

Start a backup job

Backups run according to the schedule specified in the backup policy. You can also run an on-demand backup:

- The first initial backup job creates a full recovery point.
- After the initial backup, each backup job creates incremental recovery points.
- Incremental recovery points are storage and time-efficient, as they only transfer changes made since the last backup.

To run an on-demand backup, you use the Backup-AzRecoveryServicesBackupItem.

- You specify a container in the vault that holds your backup data with Get-AzRecoveryServicesBackupContainer.
- Each VM to back up is treated as an item. To start a backup job, you obtain information about the VM with Get-AzRecoveryServicesBackupItem.

Run an on-demand backup job as follows:

1. Specify the container, obtain VM information, and run the backup.

   PowerShellCopy

   ```
   $backupcontainer = Get-AzRecoveryServicesBackupContainer `
       -ContainerType "AzureVM" `
       -FriendlyName "myVM"

   $item = Get-AzRecoveryServicesBackupItem `
       -Container $backupcontainer `
       -WorkloadType "AzureVM"

   Backup-AzRecoveryServicesBackupItem -Item $item
   ```

2. You might need to wait up to 20 minutes, since the first backup job creates a full recovery point. Monitor the job as described in the next procedure.

Monitor the backup job

1. Run Get-AzRecoveryservicesBackupJob to monitor the job status.

   PowerShellCopy

```
Get-AzRecoveryservicesBackupJob
```

Output is similar to the following example, which shows the job as **InProgress**:

OutputCopy

```
WorkloadName   Operation         Status       StartTime
EndTime                JobID
------------   ---------         ------       ---------                -
------                 -----
myvm           Backup            InProgress   9/18/2017 9:38:02 PM
9f9e8f14
myvm           ConfigureBackup   Completed    9/18/2017 9:33:18 PM
9/18/2017 9:33:51 PM   fe79c739
```

2. When the job status is **Completed**, the VM is protected and has a full recovery point stored.

Manage VM backups

If you want to perform more actions such as change policy, edit policy etc.. refer to the manage VM backups section.

Clean up the deployment

If you no longer need to back up the VM, you can clean it up.

- If you want to try out restoring the VM, skip the clean-up.
- If you used an existing VM, you can skip the final Remove-AzResourceGroup cmdlet to leave the resource group and VM in place.

Disable protection, remove the restore points and vault. Then delete the resource group and associated VM resources, as follows:

PowerShellCopy

```
Disable-AzRecoveryServicesBackupProtection -Item $item -RemoveRecoveryPoints
$vault = Get-AzRecoveryServicesVault -Name "myRecoveryServicesVault"
Remove-AzRecoveryServicesVault -Vault $vault
Remove-AzResourceGroup -Name "myResourceGroup"
```

Next steps

In this quickstart, you created a Recovery Services vault, enabled protection on a VM, and created the initial recovery point.

Back up a virtual machine in Azure

- Article
- 02/27/2023
- 8 contributors

Feedback

In this article

Azure backups can be created through the Azure portal. This method provides a browser-based user interface to create and configure Azure backups and all related resources. You can protect your data by taking backups at regular intervals. Azure Backup creates recovery points that can be stored in geo-redundant recovery vaults. This article details how to back up a virtual machine (VM) with the Azure portal.

This quickstart enables backup on an existing Azure VM. If you need to create a VM, you can create a VM with the Azure portal.

Sign in to Azure

Sign in to the Azure portal.

Note

The functionality described in the following sections can also be accessed via **Backup center**. Backup center is a single unified management experience in Azure. It enables enterprises to govern, monitor, operate, and analyze backups at scale. With this solution, you can perform most of the key backup management operations without being limited to the scope of an individual vault.

Create a Recovery Services vault

A Recovery Services vault is a management entity that stores recovery points that are created over time, and it provides an interface to perform backup-related operations. These operations include taking on-demand backups, performing restores, and creating backup policies.

To create a Recovery Services vault:

1. Sign in to the Azure portal.
2. Search for **Backup center**, and then go to the **Backup center** dashboard.

3. On the **Overview** pane, select **Vault**.

4. Select **Recovery Services vault** > **Continue**.

5. On the **Recovery Services vault** pane, enter the following values:
    - **Subscription**: Select the subscription to use. If you're a member of only one subscription, you'll see that name. If you're not sure which subscription to use, use the default subscription. There are multiple choices only if your work or school account is associated with more than one Azure subscription.
    - **Resource group**: Use an existing resource group or create a new one. To view a list of available resource groups in your subscription, select **Use existing**, and then select a resource in the dropdown list. To create a new resource group, select **Create new**, and then enter the name. For more information about resource groups, see Azure Resource Manager overview.
    - **Vault name**: Enter a friendly name to identify the vault. The name must be unique to the Azure subscription. Specify a name that has at least 2 but not more than 50 characters. The name must start with a letter and consist only of letters, numbers, and hyphens.
    - **Region**: Select the geographic region for the vault. For you to create a vault to help protect any data source, the vault *must* be in the same region as the data source.

      Important

      If you're not sure of the location of your data source, close the window. Go to the list of your resources in the portal. If

you have data sources in multiple regions, create a Recovery Services vault for each region. Create the vault in the first location before you create a vault in another location. There's no need to specify storage accounts to store the backup data. The Recovery Services vault and Azure Backup handle that automatically.

6.
7. After providing the values, select **Review + create**.
8. To finish creating the Recovery Services vault, select **Create**.

   It can take a while to create the Recovery Services vault. Monitor the status notifications in the **Notifications** area at the upper right. After the vault is created, it appears in the list of Recovery Services vaults. If the vault doesn't appear, select **Refresh**.

Note

Azure Backup now supports immutable vaults that help you ensure that recovery points once created can't be deleted before their expiry as per the backup policy. You can make the immutability irreversible for maximum protection to your backup data from various threats, including ransomware attacks and malicious actors. **Learn more**.

Apply a backup policy

To apply a backup policy to your Azure VMs, follow these steps:

1. Go to **Backup center** and click **+Backup** from the **Overview** tab.

2. Select **Azure Virtual machines** as the **Datasource type** and select the vault you have created. Then click **Continue**.

3. Assign a Backup policy.

- The default policy backs up the VM once a day. The daily backups are retained for *30 days*. Instant recovery snapshots are retained for two days.

- If you don't want to use the default policy, select **Create New**, and create a custom policy as described in the next procedure.

Note

With Enhanced policy, you can now back up Azure VMs multiple times a day that helps to perform hourly backups. **Learn more**.

Select a VM to back up

Create a simple scheduled daily backup to a Recovery Services vault.

1. Under **Virtual Machines**, select **Add**.

2. The **Select virtual machines** pane will open. Select the VMs you want to back up using the policy. Then select **OK**.
    - The selected VMs are validated.
    - You can only select VMs in the same region as the vault.
    - VMs can only be backed up in a single vault.

3. **Note**
4. All the VMs in the same region and subscription as that of the vault are available to configure backup. When configuring backup, you can browse to the virtual machine name and its resource group, even though you don’t have the required permission on those VMs. If your VM is in soft deleted state, then it won't be visible in this list. If you need to re-protect the VM, then you need to wait for the soft delete period to expire or undelete the VM from the soft deleted list. For more information, see **the soft delete for VMs article**.

Enable backup on a VM

A Recovery Services vault is a logical container that stores the backup data for each protected resource, such as Azure VMs. When the backup job for a protected resource runs, it creates a recovery point inside the Recovery Services vault. You can then use one of these recovery points to restore data to a given point in time.

To enable VM backup, in **Backup**, select **Enable backup**. This deploys the policy to the vault and to the VMs, and installs the backup extension on the VM agent running on the Azure VM.

After enabling backup:

- The Backup service installs the backup extension whether or not the VM is running.
- An initial backup will run in accordance with your backup schedule.
- When backups run, note that:
    - A VM that's running has the greatest chance for capturing an application-consistent recovery point.
    - However, even if the VM is turned off, it's backed up. Such a VM is known as an offline VM. In this case, the recovery point will be crash-consistent.
- Explicit outbound connectivity isn't required to allow backup of Azure VMs.

Create a custom policy

If you selected to create a new backup policy, fill in the policy settings.

1. In **Policy name**, specify a meaningful name.
2. In **Backup schedule**, specify when backups should be taken. You can take daily or weekly backups for Azure VMs.
3. In **Instant Restore**, specify how long you want to retain snapshots locally for instant restore.
    - When you restore, backed up VM disks are copied from storage, across the network to the recovery storage location. With instant restore, you can leverage locally stored snapshots taken during a backup job, without waiting for backup data to be transferred to the vault.
    - You can retain snapshots for instant restore for between one to five days. The default value is two days.
4. In **Retention range**, specify how long you want to keep your daily or weekly backup points.

5. In **Retention of monthly backup point** and **Retention of yearly backup point**, specify whether you want to keep a monthly or yearly backup of your daily or weekly backups.
6. Select **OK** to save the policy.

Note

To store the restore point collection (RPC), the Backup service creates a separate resource group (RG). This RG is different than RG of the VM. **Learn more**.

Note

Azure Backup doesn't support automatic clock adjustment for daylight-saving changes for Azure VM backups. As time changes occur, modify backup policies manually as required.

Start a backup job

The initial backup will run in accordance with the schedule, but you can run it immediately as follows:

1. Go to **Backup center** and select the **Backup Instances** menu item.
2. Select **Azure Virtual machines** as the **Datasource type**. Then search for the VM that you have configured for backup.
3. Right-click the relevant row or select the more icon (...), and then click **Backup Now**.
4. In **Backup Now**, use the calendar control to select the last day that the recovery point should be retained. Then select **OK**.
5. Monitor the portal notifications. To monitor the job progress, go to **Backup center** > **Backup Jobs** and filter the list for **In progress** jobs. Depending on the size of your VM, creating the initial backup may take a while.

Monitor the backup job

The Backup job details for each VM backup consist of two phases, the **Snapshot** phase followed by the **Transfer data to vault** phase.

The snapshot phase guarantees the availability of a recovery point stored along with the disks for **Instant Restores** and are available for a maximum of five days depending on the snapshot retention configured by the user. Transfer data to vault creates a recovery point in the vault for long-term retention. Transfer data to vault only starts after the snapshot phase is completed.

There are two **Sub Tasks** running at the backend, one for front-end backup job that can be checked from the **Backup Job** details pane as given below:

The **Transfer data to vault** phase can take multiple days to complete depending on the size of the disks, churn per disk and several other factors.

Job status can vary depending on the following scenarios:

Snapshot	Transfer data to vault	Job Status
Completed	In progress	In progress
Completed	Skipped	Completed
Completed	Completed	Completed
Completed	Failed	Completed with warning
Failed	Failed	Failed

Now with this capability, for the same VM, two backups can run in parallel, but in either phase (snapshot, transfer data to vault) only one sub task can be running. So in scenarios where a backup job in progress resulted in the next day's backup to fail, it will be avoided with this decoupling functionality. Subsequent days' backups can have the snapshot completed, while **Transfer data to vault** is skipped if an earlier day's backup job is in progress state. The incremental recovery point created in the vault will capture all the churn from the most recent recovery point created in the vault. There's no cost impact on the user.

Optional steps

Install the VM agent

Azure Backup backs up Azure VMs by installing an extension to the Azure VM agent running on the machine. If your VM was created from an Azure Marketplace image, the agent is installed and running. If you create a custom VM, or you migrate an on-

premises machine, you might need to install the agent manually, as summarized in the table.

VM	Details
Windows	1. Download and install the agent MSI file. 2. Install with admin permissions on the machine. 3. Verify the installation. In *C:\WindowsAzure\Packages* on the VM, right-click **WaAppAgent.exe** > **Properties**. On the **Details** tab, **Product Version** should be 2.6.1198.718 or higher. If you're updating the agent, make sure that no backup operations are running, and reinstall the agent.
Linux	Install by using an RPM or a DEB package from your distribution's package repository. This is the preferred method for installing and upgrading the Azure Linux agent. All the endorsed distribution providers integrate the Azure Linux agent package into their images and repositories. The agent is available on GitHub, but we don't recommend installing from there. If you're updating the agent, make sure no backup operations are running, and update the binaries.

Clean up deployment

When no longer needed, you can disable protection on the VM, remove the restore points and Recovery Services vault, then delete the resource group and associated VM resources

If you're going to continue on to a Backup tutorial that explains how to restore data for your VM, skip the steps in this section and go to Next steps.

1. Select the **Backup** option for your VM.
2. Choose **Stop backup**.

3. Select **Delete Backup Data** from the drop-down menu.
4. In the **Type the name of the Backup item** dialog, enter your VM name, such as *myVM*. Select **Stop Backup**.

   Once the VM backup has been stopped and recovery points removed, you can delete the resource group. If you used an existing VM, you may wish to leave the resource group and VM in place.

5. In the menu on the left, select **Resource groups**.
6. From the list, choose your resource group. If you used the sample VM quickstart commands, the resource group is named *myResourceGroup*.
7. Select **Delete resource group**. To confirm, enter the resource group name, then select **Delete**.

Next steps

In this quickstart, you created a Recovery Services vault, enabled protection on a VM, and created the initial recovery point. To learn more about Azure Backup and Recovery Services, continue to the tutorials.

Back up a virtual machine in Azure with an ARM template

- Article
- 03/08/2023
- 10 contributors

Feedback

In this article

Azure Backup backs up on-premises machines and apps, and Azure VMs. This article shows you how to back up an Azure VM with an Azure Resource Manager template (ARM template) and Azure PowerShell. This quickstart focuses on the process of deploying an ARM template to create a Recovery Services vault. For more information on developing ARM templates, see the Azure Resource Manager documentation and the template reference.

A resource manager template is a JavaScript Object Notation (JSON) file that defines the infrastructure and configuration for your project. The template uses declarative

syntax. In declarative syntax, you describe your intended deployment without writing the sequence of programming commands to create the deployment.

A Recovery Services vault is a logical container that stores backup data for protected resources, such as Azure VMs. When a backup job runs, it creates a recovery point inside the Recovery Services vault. You can then use one of these recovery points to restore data to a given point in time. Alternatively, you can back up a VM using Azure PowerShell, the Azure CLI, or in the Azure portal.

If your environment meets the prerequisites and you're familiar with using ARM templates, select the **Deploy to Azure** button. The template will open in the Azure portal.

Review the template

The template used in this quickstart is from Azure quickstart Templates. This template allows you to deploy simple Windows VM and Recovery Services vault configured with the *DefaultPolicy* for *Protection*.

JSONCopy

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-
01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "metadata": {
    "_generator": {
      "name": "bicep",
      "version": "0.5.6.12127",
      "templateHash": "12431143174203400392"
    }
  },
  "parameters": {
    "projectName": {
      "type": "string",
      "maxLength": 8,
      "metadata": {
        "description": "Specifies a name for generating resource names."
      }
    },
    "location": {
      "type": "string",
      "defaultValue": "[resourceGroup().location]",
      "metadata": {
        "description": "Specifies the location for all resources."
      }
    },
    "adminUsername": {
      "type": "string",
```

```
      "metadata": {
        "description": "Specifies the administrator username for the Virtual
Machine."
      }
    },
    "adminPassword": {
      "type": "secureString",
      "metadata": {
        "description": "Specifies the administrator password for the Virtual
Machine."
      }
    },
    "dnsLabelPrefix": {
      "type": "string",
      "metadata": {
        "description": "Specifies the unique DNS Name for the Public IP used to
access the Virtual Machine."
      }
    },
    "vmSize": {
      "type": "string",
      "defaultValue": "Standard_A2",
      "metadata": {
        "description": "Virtual machine size."
      }
    },
    "windowsOSVersion": {
      "type": "string",
      "defaultValue": "2016-Datacenter",
      "allowedValues": [
        "2008-R2-SP1",
        "2012-Datacenter",
        "2012-R2-Datacenter",
        "2016-Nano-Server",
        "2016-Datacenter-with-Containers",
        "2016-Datacenter",
        "2019-Datacenter",
        "2019-Datacenter-Core",
        "2019-Datacenter-Core-smalldisk",
        "2019-Datacenter-Core-with-Containers",
        "2019-Datacenter-Core-with-Containers-smalldisk",
        "2019-Datacenter-smalldisk",
        "2019-Datacenter-with-Containers",
        "2019-Datacenter-with-Containers-smalldisk"
      ],
      "metadata": {
        "description": "Specifies the Windows version for the VM. This will pick a
fully patched image of this given Windows version."
      }
    }
  },
  "variables": {
    "storageAccountName": "[format('{0}store', parameters('projectName'))]",
    "networkInterfaceName": "[format('{0}-nic', parameters('projectName'))]",
    "vNetAddressPrefix": "10.0.0.0/16",
    "vNetSubnetName": "default",
    "vNetSubnetAddressPrefix": "10.0.0.0/24",
    "publicIPAddressName": "[format('{0}-ip', parameters('projectName'))]",
    "vmName": "[format('{0}-vm', parameters('projectName'))]",
```

```
    "vNetName": "[format('{0}-vnet', parameters('projectName'))]",
    "vaultName": "[format('{0}-vault', parameters('projectName'))]",
    "backupFabric": "Azure",
    "backupPolicyName": "DefaultPolicy",
    "protectionContainer": "[format('iaasvmcontainer;iaasvmcontainerv2;{0};{1}',
resourceGroup().name, variables('vmName'))]",
    "protectedItem": "[format('vm;iaasvmcontainerv2;{0};{1}',
resourceGroup().name, variables('vmName'))]",
    "networkSecurityGroupName": "default-NSG"
  },
  "resources": [
    {
      "type": "Microsoft.Storage/storageAccounts",
      "apiVersion": "2021-08-01",
      "name": "[variables('storageAccountName')]",
      "location": "[parameters('location')]",
      "sku": {
        "name": "Standard_LRS"
      },
      "kind": "Storage",
      "properties": {}
    },
    {
      "type": "Microsoft.Network/publicIPAddresses",
      "apiVersion": "2021-05-01",
      "name": "[variables('publicIPAddressName')]",
      "location": "[parameters('location')]",
      "properties": {
        "publicIPAllocationMethod": "Dynamic",
        "dnsSettings": {
          "domainNameLabel": "[parameters('dnsLabelPrefix')]"
        }
      }
    },
    {
      "type": "Microsoft.Network/networkSecurityGroups",
      "apiVersion": "2021-05-01",
      "name": "[variables('networkSecurityGroupName')]",
      "location": "[parameters('location')]",
      "properties": {
        "securityRules": [
          {
            "name": "default-allow-3389",
            "properties": {
              "priority": 1000,
              "access": "Allow",
              "direction": "Inbound",
              "destinationPortRange": "3389",
              "protocol": "Tcp",
              "sourceAddressPrefix": "*",
              "sourcePortRange": "*",
              "destinationAddressPrefix": "*"
            }
          }
        ]
      }
    },
    {
      "type": "Microsoft.Network/virtualNetworks",
```

```
      "apiVersion": "2021-05-01",
      "name": "[variables('vNetName')]",
      "location": "[parameters('location')]",
      "properties": {
        "addressSpace": {
          "addressPrefixes": [
            "[variables('vNetAddressPrefix')]"
          ]
        },
        "subnets": [
          {
            "name": "[variables('vNetSubnetName')]",
            "properties": {
              "addressPrefix": "[variables('vNetSubnetAddressPrefix')]",
              "networkSecurityGroup": {
                "id": "[resourceId('Microsoft.Network/networkSecurityGroups',
variables('networkSecurityGroupName'))]"
              }
            }
          }
        ]
      },
      "dependsOn": [
        "[resourceId('Microsoft.Network/networkSecurityGroups',
variables('networkSecurityGroupName'))]"
      ]
    },
    {
      "type": "Microsoft.Network/networkInterfaces",
      "apiVersion": "2021-05-01",
      "name": "[variables('networkInterfaceName')]",
      "location": "[parameters('location')]",
      "properties": {
        "ipConfigurations": [
          {
            "name": "ipconfig1",
            "properties": {
              "privateIPAllocationMethod": "Dynamic",
              "publicIPAddress": {
                "id": "[resourceId('Microsoft.Network/publicIPAddresses',
variables('publicIPAddressName'))]"
              },
              "subnet": {
                "id": "[format('{0}/subnets/{1}',
resourceId('Microsoft.Network/virtualNetworks', variables('vNetName')),
variables('vNetSubnetName'))]"
              }
            }
          }
        ]
      },
      "dependsOn": [
        "[resourceId('Microsoft.Network/publicIPAddresses',
variables('publicIPAddressName'))]",
        "[resourceId('Microsoft.Network/virtualNetworks', variables('vNetName'))]"
      ]
    },
    {
      "type": "Microsoft.Compute/virtualMachines",
```

```
      "apiVersion": "2021-11-01",
      "name": "[variables('vmName')]",
      "location": "[parameters('location')]",
      "properties": {
        "hardwareProfile": {
          "vmSize": "[parameters('vmSize')]"
        },
        "osProfile": {
          "computerName": "[variables('vmName')]",
          "adminUsername": "[parameters('adminUsername')]",
          "adminPassword": "[parameters('adminPassword')]"
        },
        "storageProfile": {
          "imageReference": {
            "publisher": "MicrosoftWindowsServer",
            "offer": "WindowsServer",
            "sku": "[parameters('windowsOSVersion')]",
            "version": "latest"
          },
          "osDisk": {
            "createOption": "FromImage"
          },
          "dataDisks": [
            {
              "diskSizeGB": 1023,
              "lun": 0,
              "createOption": "Empty"
            }
          ]
        },
        "networkProfile": {
          "networkInterfaces": [
            {
              "id": "[resourceId('Microsoft.Network/networkInterfaces',
variables('networkInterfaceName'))]"
            }
          ]
        },
        "diagnosticsProfile": {
          "bootDiagnostics": {
            "enabled": true,
            "storageUri":
"[reference(resourceId('Microsoft.Storage/storageAccounts',
variables('storageAccountName'))).primaryEndpoints.blob]"
          }
        }
      },
      "dependsOn": [
        "[resourceId('Microsoft.Network/networkInterfaces',
variables('networkInterfaceName'))]",
        "[resourceId('Microsoft.Storage/storageAccounts',
variables('storageAccountName'))]"
      ]
    },
    {
      "type": "Microsoft.RecoveryServices/vaults",
      "apiVersion": "2022-01-01",
      "name": "[variables('vaultName')]",
      "location": "[parameters('location')]",
```

```
      "sku": {
        "name": "RS0",
        "tier": "Standard"
      },
      "properties": {}
    },
    {
      "type":
"Microsoft.RecoveryServices/vaults/backupFabrics/protectionContainers/protectedIte
ms",
      "apiVersion": "2022-01-01",
      "name": "[format('{0}/{1}/{2}/{3}', variables('vaultName'),
variables('backupFabric'), variables('protectionContainer'),
variables('protectedItem'))]",
      "properties": {
        "protectedItemType": "Microsoft.Compute/virtualMachines",
        "policyId": "[format('{0}/backupPolicies/{1}',
resourceId('Microsoft.RecoveryServices/vaults', variables('vaultName')),
variables('backupPolicyName'))]",
        "sourceResourceId": "[resourceId('Microsoft.Compute/virtualMachines',
variables('vmName'))]"
      },
      "dependsOn": [
        "[resourceId('Microsoft.RecoveryServices/vaults',
variables('vaultName'))]",
        "[resourceId('Microsoft.Compute/virtualMachines', variables('vmName'))]"
      ]
    }
  ]
}
```

The resources defined in the template are:

- **Microsoft.Storage/storageAccounts**
- **Microsoft.Network/publicIPAddresses**
- **Microsoft.Network/networkSecurityGroups**
- **Microsoft.Network/virtualNetworks**
- **Microsoft.Network/networkInterfaces**
- **Microsoft.Compute/virtualMachines**
- **Microsoft.RecoveryServices/vaults**
- **Microsoft.RecoveryServices/vaults/backupFabrics/protectionContainers/protectedItems**

Deploy the template

To deploy the template, select **Try it** to open the Azure Cloud Shell, and then paste the following PowerShell script into the shell window. To paste the code, right-click the shell window and then select **Paste**.

Azure PowerShellCopy
Open Cloudshell

```
$projectName = Read-Host -Prompt "Enter a project name (limited to eight
characters) that is used to generate Azure resource names"
$location = Read-Host -Prompt "Enter the location (for example, centralus)"
$adminUsername = Read-Host -Prompt "Enter the administrator username for the
virtual machine"
$adminPassword = Read-Host -Prompt "Enter the administrator password for the
virtual machine" -AsSecureString
$dnsPrefix = Read-Host -Prompt "Enter the unique DNS Name for the Public IP used
to access the virtual machine"

$resourceGroupName = "${projectName}rg"
$templateUri = "https://raw.githubusercontent.com/Azure/azure-quickstart-
templates/master/quickstarts/microsoft.recoveryservices/recovery-services-create-
vm-and-configure-backup/azuredeploy.json"

New-AzResourceGroup -Name $resourceGroupName -Location $location
New-AzResourceGroupDeployment -ResourceGroupName $resourceGroupName -TemplateUri
$templateUri -projectName $projectName -adminUsername $adminUsername -
adminPassword $adminPassword -dnsLabelPrefix $dnsPrefix
```

Azure PowerShell is used to deploy the ARM template in this quickstart. The Azure portal, Azure CLI, and REST API can also be used to deploy templates.

Validate the deployment

Start a backup job

The template creates a VM and enables backup on the VM. After you deploy the template, you need to start a backup job. For more information, see Start a backup job.

Monitor the backup job

To monitor the backup job, see Monitor the backup job.

Clean up resources

If you no longer need to back up the VM, you can clean it up.

- If you want to try out restoring the VM, skip the cleanup.
- If you used an existing VM, you can skip the final Remove-AzResourceGroup cmdlet to leave the resource group and VM in place.

Disable protection, remove the restore points and vault. Then delete the resource group and associated VM resources, as follows:

PowerShellCopy

```
Disable-AzRecoveryServicesBackupProtection -Item $item -RemoveRecoveryPoints
$vault = Get-AzRecoveryServicesVault -Name "myRecoveryServicesVault"
Remove-AzRecoveryServicesVault -Vault $vault
Remove-AzResourceGroup -Name "myResourceGroup"
```

Next steps

In this quickstart, you created a Recovery Services vault, enabled protection on a VM, and created the initial recovery point.

Tutorial: Set up disaster recovery for Linux virtual machines

- Article
- 03/16/2023
- 3 contributors

Feedback

In this article

Applies to: ✔ Linux VMs ✔ Flexible scale sets

This tutorial shows you how to set up disaster recovery for Azure VMs running Linux. In this article, learn how to:

- Enable disaster recovery for a Linux VM
- Run a disaster recovery drill to check it works as expected
- Stop replicating the VM after the drill

When you enable replication for a VM, the Site Recovery Mobility service extension installs on the VM, and registers it with Azure Site Recovery. During replication, VM disk writes are sent to a cache storage account in the source VM region. Data is sent from there to the target region, and recovery points are generated from the data.

When you fail a VM over to another region during disaster recovery, a recovery point is used to create a VM in the target region.

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

1. Check that your Azure subscription allows you to create a VM in the target region. If you just created your free Azure account, you're the administrator of the subscription, and you have the permissions you need.
2. If you're not the subscription administrator, work with the administrator to assign you:
    - Either the Virtual Machine Contributor built-in role, or specific permissions to:
        - Create a VM in the selected virtual network.
        - Write to an Azure storage account.
        - Write to an Azure managed disk.
    - The Site Recovery Contributor built-in role, to manage Site Recovery operations in the vault.
3. Check that the Linux VM is running a supported operating system.
4. If VM outbound connections use a URL-based proxy, make sure it can access these URLs. Using an authenticated proxy isn't supported.

Name	Public cloud	Government cloud
Storage	`*.blob.core.windows.net`	`*.blob.core.usgovcloudapi.net`
Azure AD	`login.microsoftonline.com`	`login.microsoftonline.us`
Replication	`*.hypervrecoverymanager.windowsazure.com`	`*.hypervrecoverymanager.windowsazure.com`
Service Bus	`*.servicebus.windows.net`	`*.servicebus.usgovcloudapi.net`

5. If you're using network security groups (NSGs) to limit network traffic for VMs, create NSG rules that allow outbound connectivity (HTTPS 443) for

the VM using these service tags (groups of IP addresses). Try out the rules on a test NSG first.

Tag	Allow
Storage tag	Allows data to be written from the VM to the cache storage account.
Azure AD tag	Allows access to all IP addresses that correspond to Azure AD.
Events Hub tag	Allows access to Site Recovery monitoring.
Azure Site Recovery tag	Allows access to the Site Recovery service in any region.
GuestAndHybridManagement	Use if you want to automatically upgrade the Site Recovery Mobility agent that's running on VMs enabled for replication.

6. Make sure VMs have the latest root certificates. On Linux VMs, follow the guidance provided by your Linux distributor, to get the latest trusted root certificates and certificate revocation list on the VM.

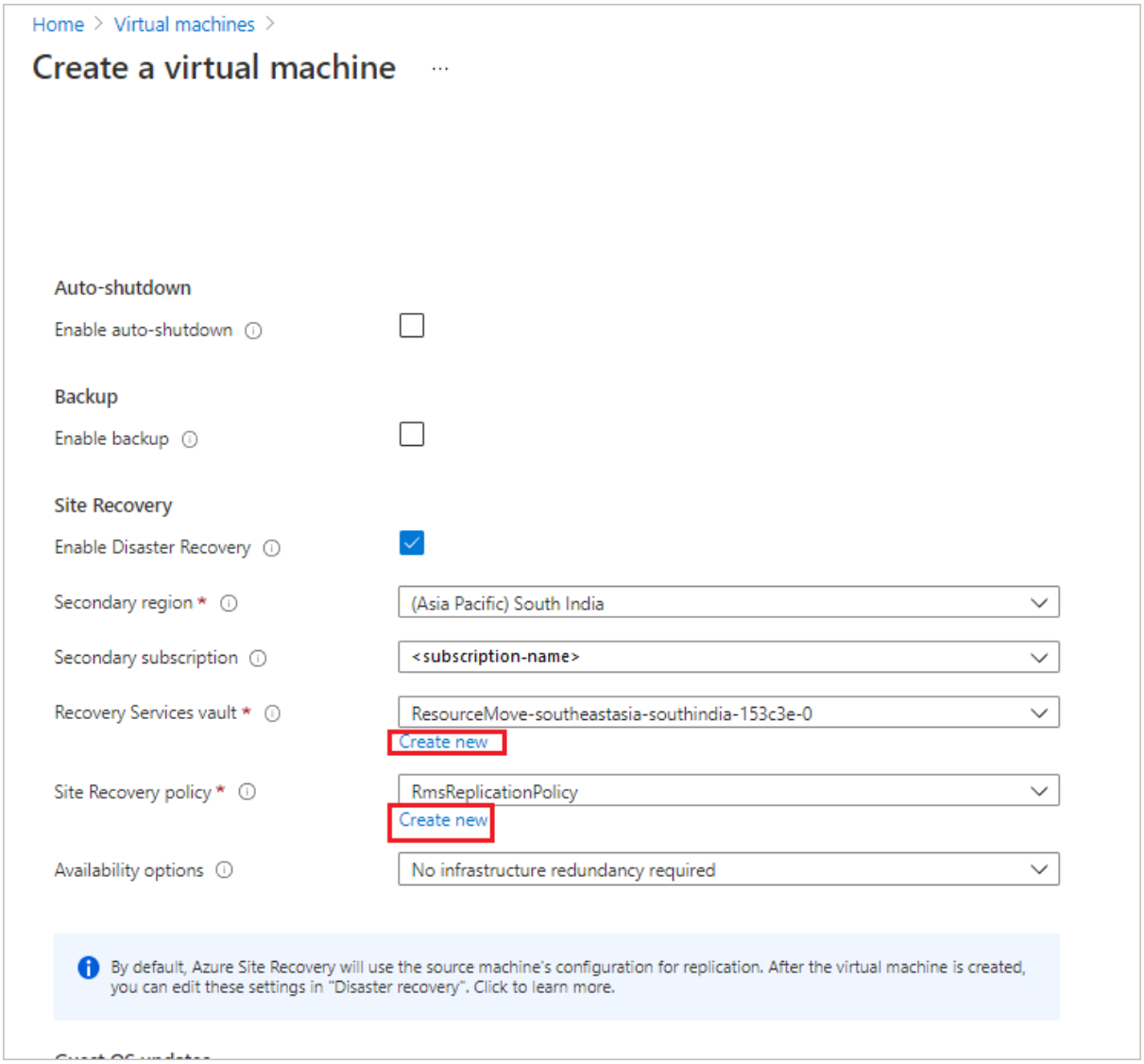
Create a VM and enable disaster recovery

You can optionally enable disaster recovery when you create a VM.

1. Create a Linux VM.
2. On the **Management** tab, under **Site Recovery** select **Enable disaster recovery**.
3. In **Secondary region**, select the target region to which you want to replicate the VM for disaster recovery.
4. In **Secondary subscription**, select the target subscription in which the target VM will be created. The target VM is created when you fail over the source VM from the source region to the target region.
5. In **Recovery Services vault**, select the vault you want to use for the replication. If you don't have a vault, select **Create new**. Select a resource group in which to place the vault, and a vault name.
6. In **Site Recovery policy**, leave the default policy, or select **Create new** to set custom values.
    - Recovery points are created from snapshots of VM disks taken at a specific point in time. When you fail over a VM, you use a recovery point to restore the VM in the target region.
    - A crash-consistent recovery point is created every five minutes. This setting can't be modified. A crash-consistent snapshot captures data that was on the disk when the snapshot was taken. It doesn't include anything in memory.

- By default Site Recovery keeps crash-consistent recovery points for 24 hours. You can set a custom value between 0 and 72 hours.
- An app-consistent snapshot is taken every 4 hours.
- By default Site Recovery stores recovery points for 24 hours.

7. In **Availability options**, specify whether the VM will deploy as standalone, in an availability zone, or in an availability set.



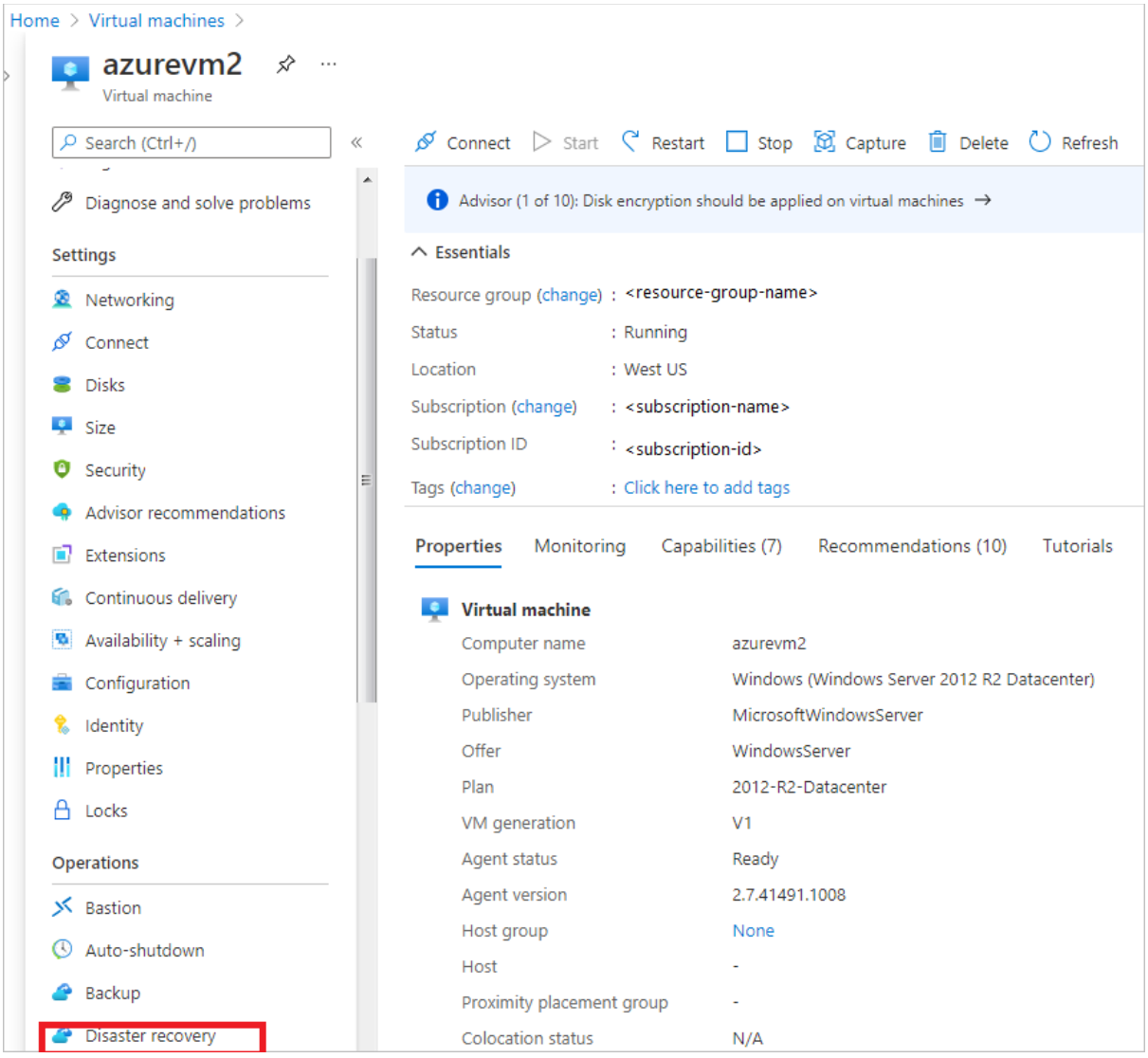
8. Finish creating the VM.

Enable disaster recovery for an existing VM

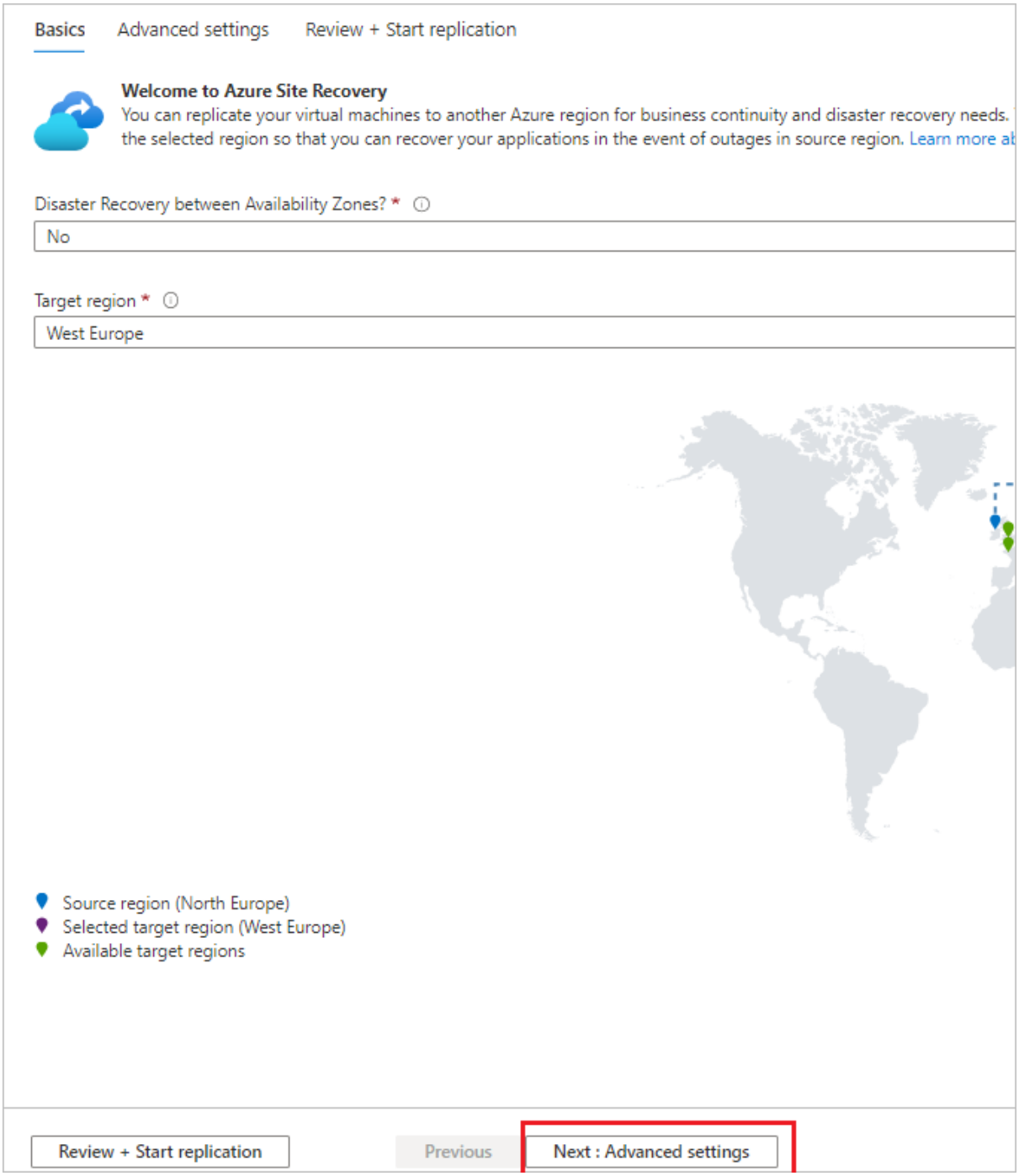
If you want to enable disaster recovery on an existing VM, use this procedure.

1. In the Azure portal, open the VM properties page.

2. In **Operations**, select **Disaster recovery**.

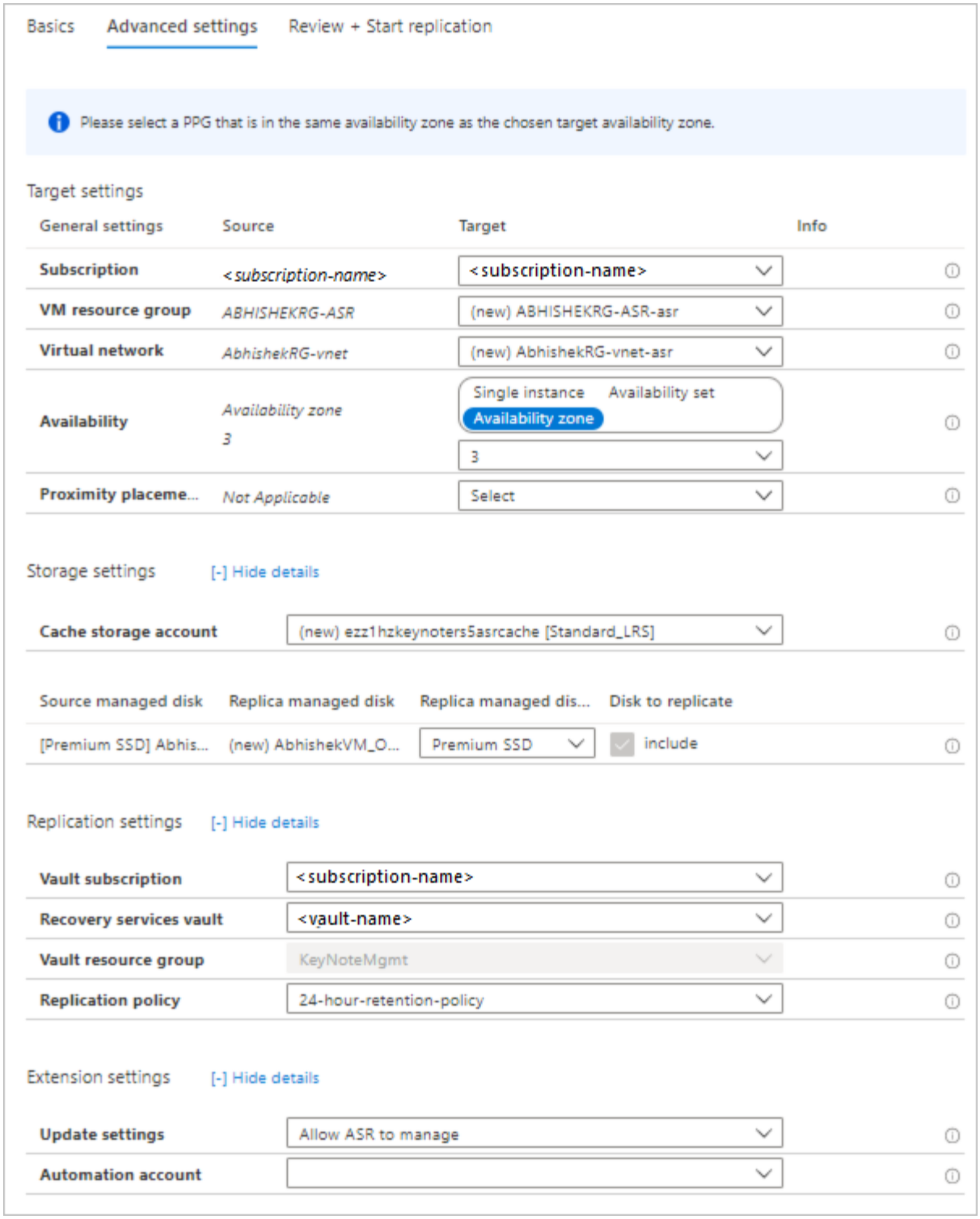


3. In **Basics**, if the VM is deployed in an availability zone, you can select disaster recovery between availability zones.
4. In **Target region**, select the region to which you want to replicate the VM. The source and target regions must be in the same Azure Active Directory tenant.

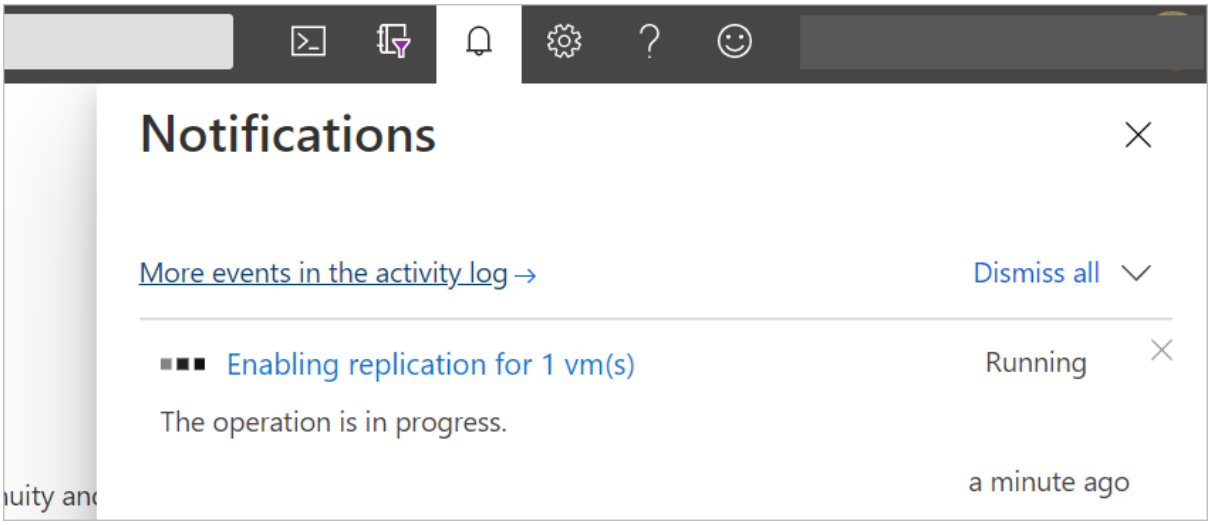


5. Select **Next: Advanced settings**.
6. In **Advanced settings**, you can review settings, and modify values to custom settings. By default, Site Recovery mirrors the source settings to create target resources.
    - **Target subscription**. The subscription in which the target VM is created after failover.
    - **Target VM resource group**. The resource group in which the target VM is created after failover.
    - **Target virtual network**. The Azure virtual network in which the target VM is located when it's created after failover.

- **Target availability**. When the target VM is created as a single instance, in an availability set, or availability zone.
- **Proximity placement**. If applicable, select the proximity placement group in which the target VM is located after failover.
- **Storage settings-Cache storage account**. Recovery uses a storage account in the source region as a temporary data store. Source VM changes are cached in this account, before being replicated to the target location.
    - By default one cache storage account is created per vault and reused.
    - You can select a different storage account if you want to customize the cache account for the VM.
- **Storage settings-Replica managed disk**. By default, Site Recovery creates replica managed disks in the target region.
    - By default the target managed disk mirror the source VM managed disks, using the same storage type (standard HDD/SSD, or premium SSD).
    - You can customize the storage type as needed.
- **Replication settings**. Shows the vault in which the VM is located, and the replication policy used for the VM. By default, recovery points created by Site Recovery for the VM are kept for 24 hours.
- **Extension settings**. Indicates that Site Recovery manages updates to the Site Recovery Mobility Service extension that's installed on VMs you replicate.
    - The indicated Azure Automation account manages the update process.
    - You can customize the automation account.



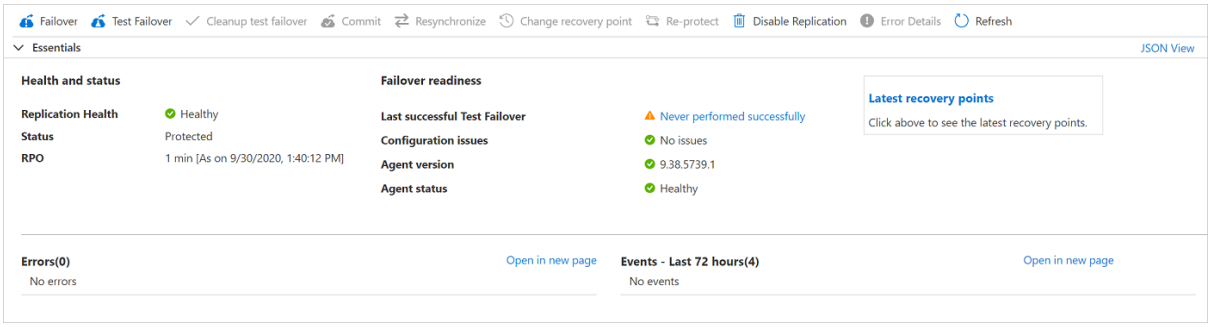
7. Select **Review + Start replication**.
8. Select **Start replication**. Deployment starts, and Site Recovery starts creating target resources. You can monitor replication progress in the notifications.



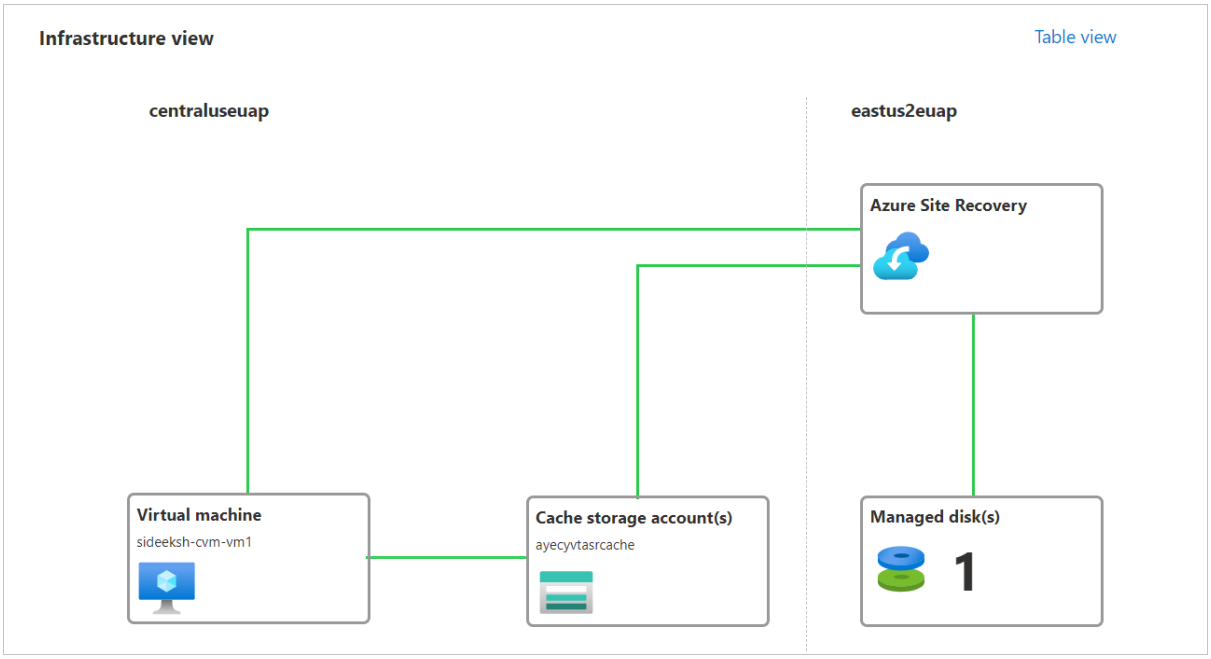
Check VM status

After the replication job finishes, you can check the VM replication status.

1. Open the VM properties page.
2. In **Operations**, select **Disaster recovery**.
3. Expand the **Essentials** section to review defaults about the vault, replication policy, and target settings.
4. In **Health and status**, get information about replication state for the VM, the agent version, failover readiness, and the latest recovery points.



5. In **Infrastructure view**, get a visual overview of source and target VMs, managed disks, and the cache storage account.



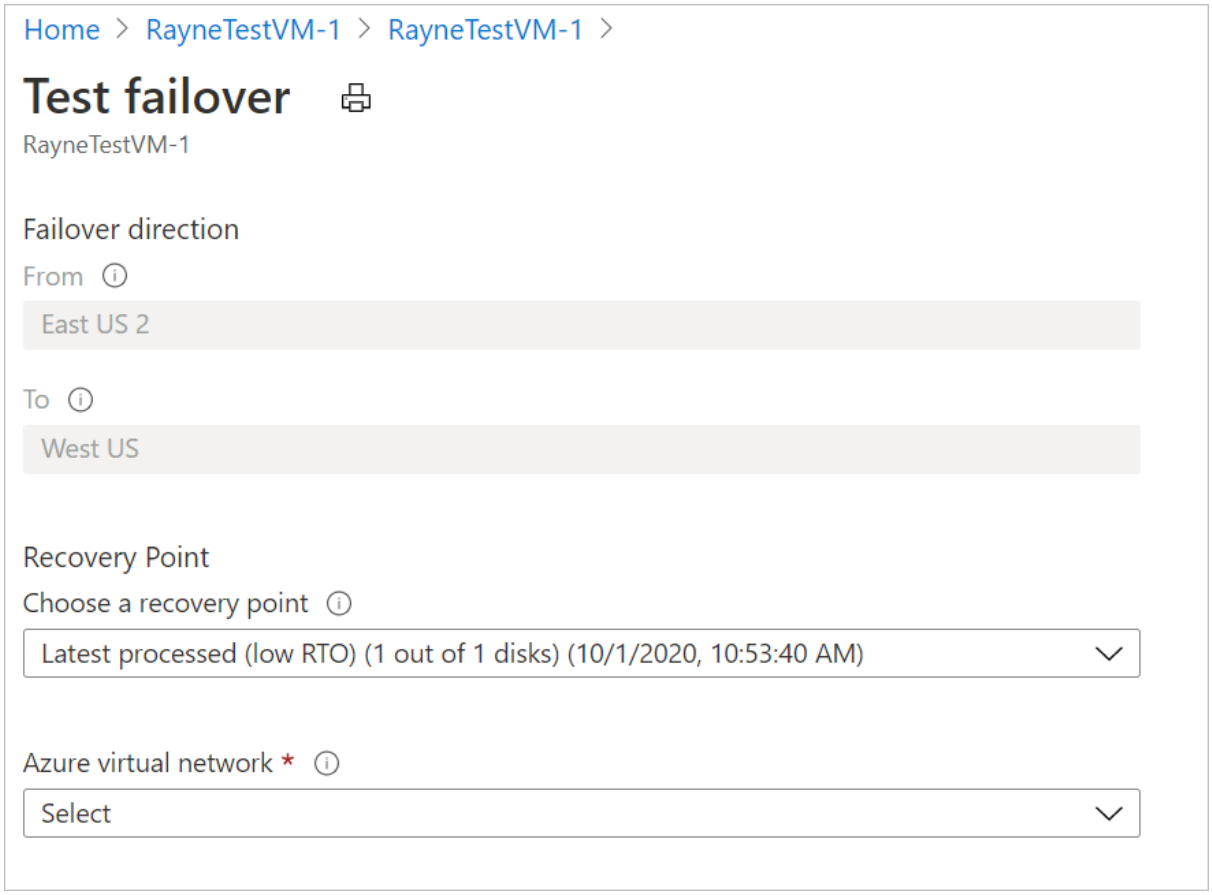
Run a drill

Run a drill to make sure disaster recovery works as expected. When you run a test failover, it creates a copy of the VM, with no impact on ongoing replication, or on your production environment.

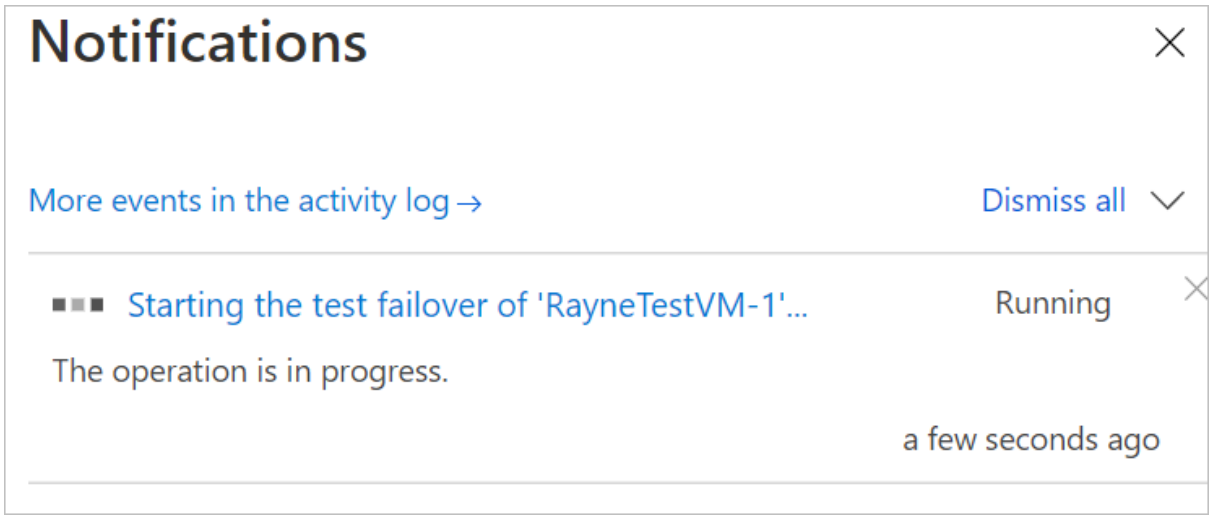
1. In the VM disaster recovery page, select **Test failover**.
2. In **Test failover**, leave the default **Latest processed (low RPO)** setting for the recovery point.

   This option provides the lowest recovery point objective (RPO), and generally spins up the VM most quickly in the target region. It first processes all the data that has been sent to Site Recovery service, to create a recovery point for each VM, before failing over to it. This recovery point has all the data replicated to Site Recovery when the failover was triggered.

3. Select the virtual network in which the VM will be located after failover.



4. The test failover process begins. You can monitor the progress in notifications.

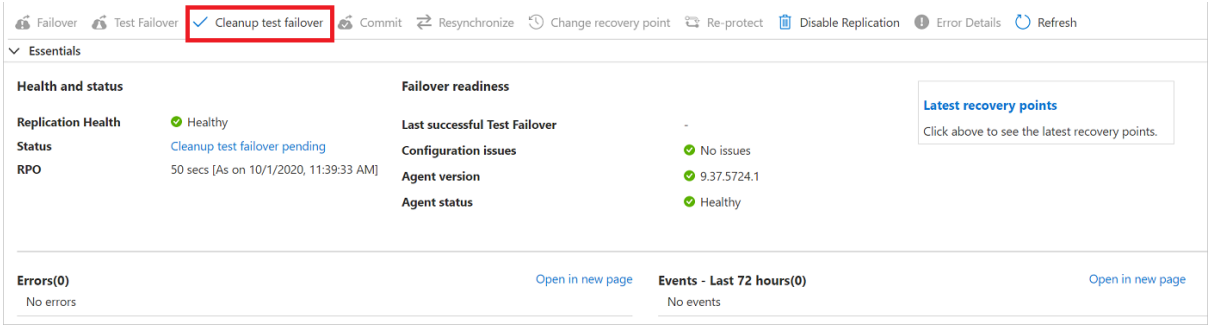


After the test failover completes, the VM is in the *Cleanup test failover pending* state on the **Essentials** page.

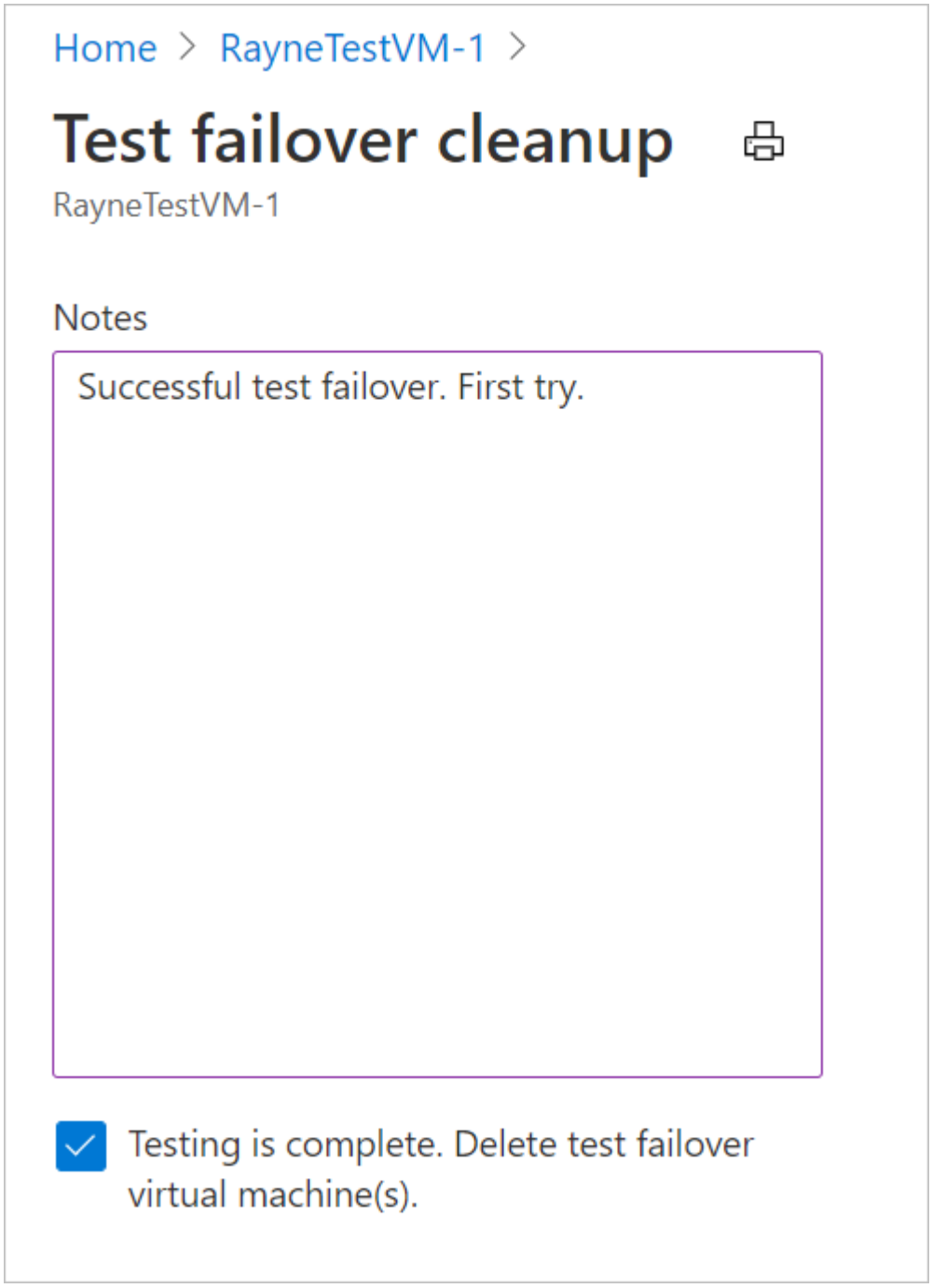
Clean up resources

The VM is automatically cleaned up by Site Recovery after the drill.

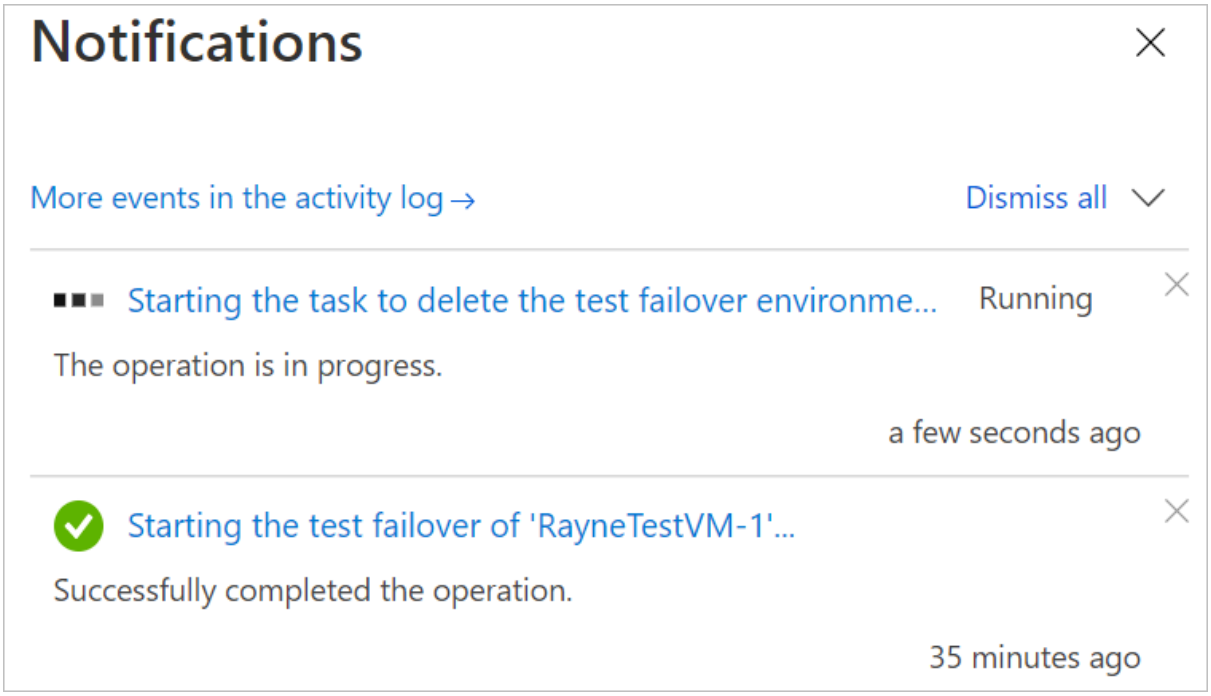
1. To begin automatic cleanup, select **Cleanup test failover**.



2. In **Test failover cleanup**, type in any notes you want to record for the failover, and then select **Testing is complete. Delete test failover virtual machine**. Then select **OK**.



3. The delete process begins. You can monitor progress in notifications.



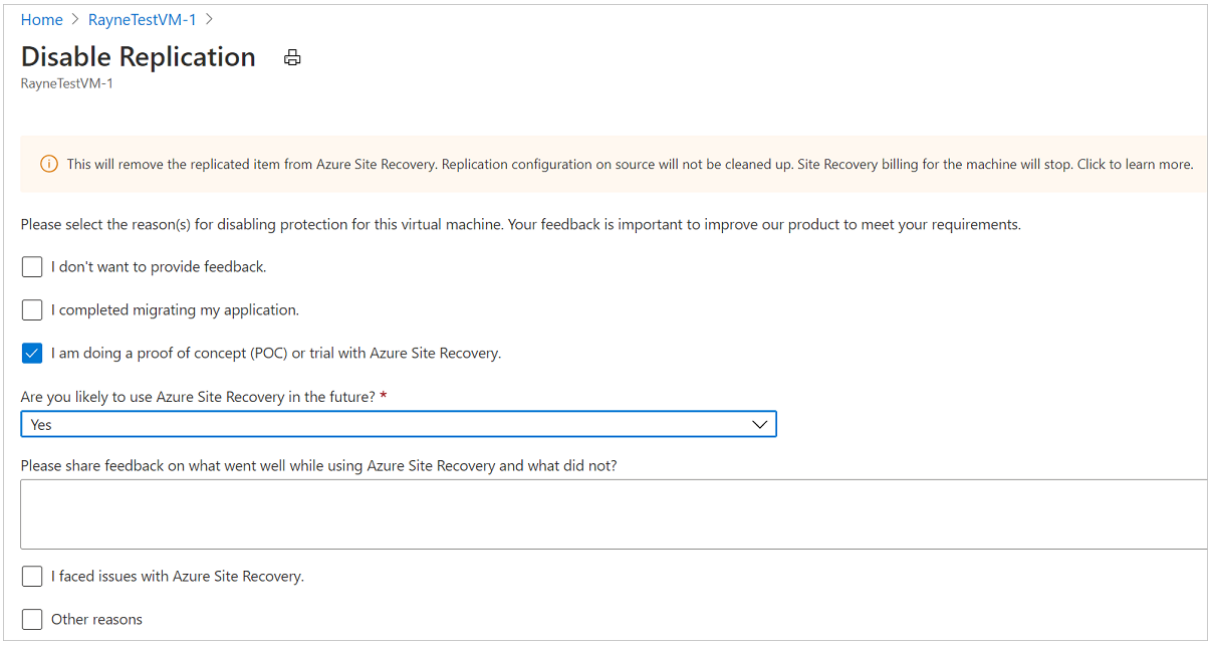
Stop replicating the VM

After completing a disaster recovery drill, we suggest you continue to try out a full failover. If you don't want to do a full failover, you can disable replication. Disabling replication will:

- Remove the VM from the Site Recovery list of replicated machines.
- Stop Site Recovery billing for the VM.
- Automatically clean up source replication settings.

Stop replication as follows:

1. In the VM disaster recovery page, select **Disable Replication**.
2. In **Disable Replication**, select the reasons that you want to disable replication. Then select **OK**.



The Site Recovery extension installed on the VM during replication isn't removed automatically. If you disable replication for the VM, and you don't want to replicate it again at a later time, you can remove the Site Recovery extension manually, as follows:

1. Go to the VM > **Settings** > **Extensions**.
2. In the **Extensions** page, select each *Microsoft.Azure.RecoveryServices* entry for Linux.
3. In the properties page for the extension, select **Uninstall**.

Tutorial: Enable disaster recovery for Windows VMs

- Article
- 03/16/2023
- 4 contributors

Feedback

In this article

Applies to: ✔ Windows VMs ✔ Flexible scale sets

This tutorial shows you how to set up disaster recovery for Azure VMs running Windows. In this article, learn how to:

- Enable disaster recovery for a Windows VM
- Run a disaster recovery drill to check it works as expected
- Stop replicating the VM after the drill

When you enable replication for a VM, the Site Recovery Mobility service extension installs on the VM, and registers it with Azure Site Recovery. During replication, VM disk writes are sent to a cache storage account in the source region. Data is sent from there to the target region, and recovery points are generated from the data. When you fail over a VM during disaster recovery, a recovery point is used to create a VM in the target region.

If you don't have an Azure subscription, create a free account before you begin.

Prerequisites

1. Check that your Azure subscription allows you to create a VM in the target region. If you just created your free Azure account, you're the administrator of the subscription, and you have the permissions you need.
2. If you're not the subscription administrator, work with the administrator to assign you:
    - Either the Virtual Machine Contributor built-in role, or specific permissions to:
        - Create a VM in the selected virtual network.
        - Write to an Azure storage account.
        - Write to an Azure-managed disk.
    - The Site Recovery Contributor built-in role, to manage Site Recovery operations in the vault.
3. We recommend you use a Windows VM running Windows Server 2012 or later. The VM disk shouldn't be encrypted for the purpose of this tutorial.
4. If VM outbound connections use a URL-based proxy, make sure it can access these URLs. Using an authenticated proxy isn't supported.

Name	Public cloud	Government cloud
Storage	`*.blob.core.windows.net`	`*.blob.core.usgovcloudapi.net`

Name	Public cloud	Government cloud
Azure AD	`login.microsoftonline.com`	`login.microsoftonline.us`
Replication	`*.hypervrecoverymanager.windowsazure.com`	`*.hypervrecoverymanager.windowsazure.com`
Service Bus	`*.servicebus.windows.net`	`*.servicebus.usgovcloudapi.net`

5. If you're using network security groups (NSGs) to limit network traffic for VMs, create NSG rules that allow outbound connectivity (HTTPS 443) for the VM using these service tags (groups of IP addresses). Try out the rules on a test NSG first.

Tag	Allow
Storage tag	Allows data to be written from the VM to the cache storage account.
Azure AD tag	Allows access to all IP addresses that correspond to Azure AD.
EventsHub tag	Allows access to Site Recovery monitoring.
AzureSiteRecovery tag	Allows access to the Site Recovery service in any region.
GuestAndHybridManagement	Use if you want to automatically upgrade the Site Recovery Mobility agent that's running on VMs enabled for replication.

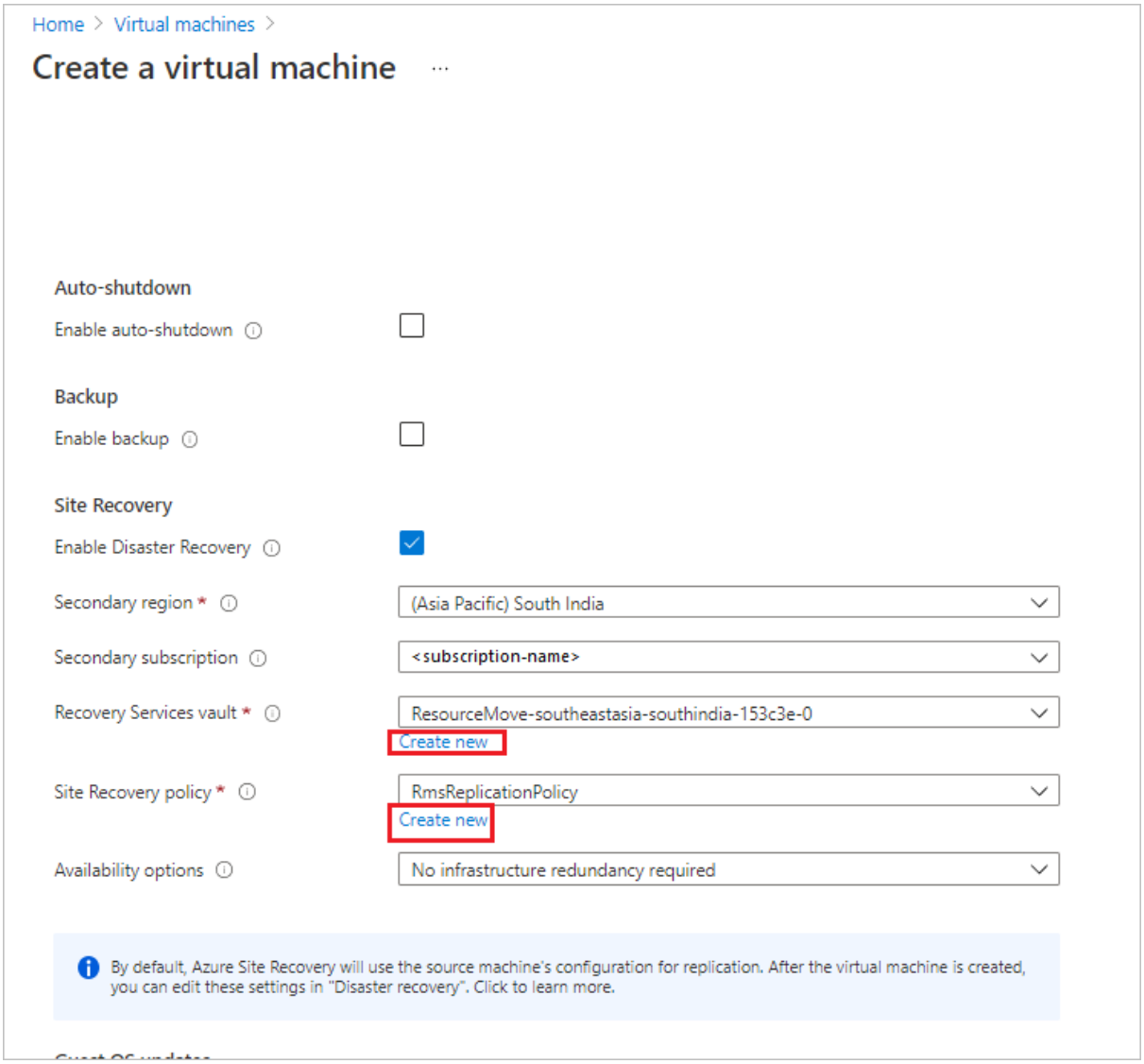
6. On Windows VMs, install the latest Windows updates, to make sure that VMs have the latest root certificates.

Create a VM and enable disaster recovery

You can optionally enable disaster recovery when you create a VM.

1. Create a VM.
2. On the **Management** tab, select **Enable disaster recovery**.
3. In **Secondary region**, select the target region to which you want to replicate a VM for disaster recovery.
4. In **Secondary subscription**, select the target subscription in which the target VM will be created. The target VM is created when you fail over the source VM from the source region to the target region.

5. In **Recovery Services vault**, select the vault you want to use for the replication. If you don't have a vault, select **Create new**. Select a resource group in which to place the vault, and a vault name.
6. In **Site Recovery policy**, leave the default policy, or select **Create new** to set custom values.
    - Recovery points are created from snapshots of VM disks taken at a specific point in time. When you fail over a VM, you use a recovery point to restore the VM in the target region.
    - A crash-consistent recovery point is created every five minutes. This setting can't be modified. A crash-consistent snapshot captures data that was on the disk when the snapshot was taken. It doesn't include anything in memory.
    - By default Site Recovery keeps crash-consistent recovery points for 24 hours. You can set a custom value between 0 and 72 hours.
    - An app-consistent snapshot is taken every 4 hours. An app-consistent snapshot
    - By default Site Recovery stores recovery points for 24 hours.
7. In **Availability options**, specify whether the VM is deploy as standalone, in an availability zone, or in an availability set.



8. Finish creating the VM.

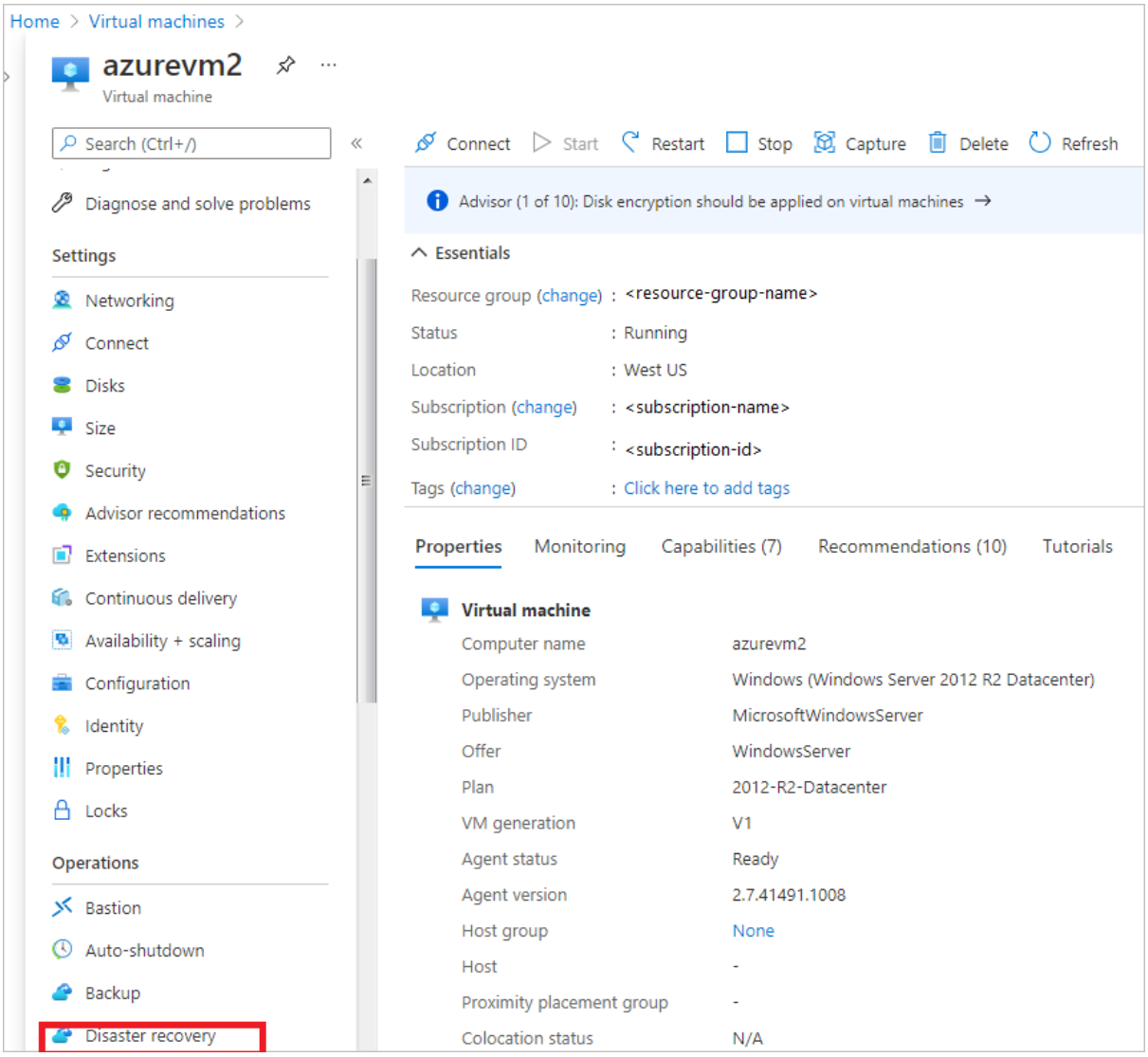
Note

When you enable replication while creating a Windows VM, only the OS disk gets replicated. Data disks need to be initialized by you, after which Azure Site Recovery automatically replicates them.

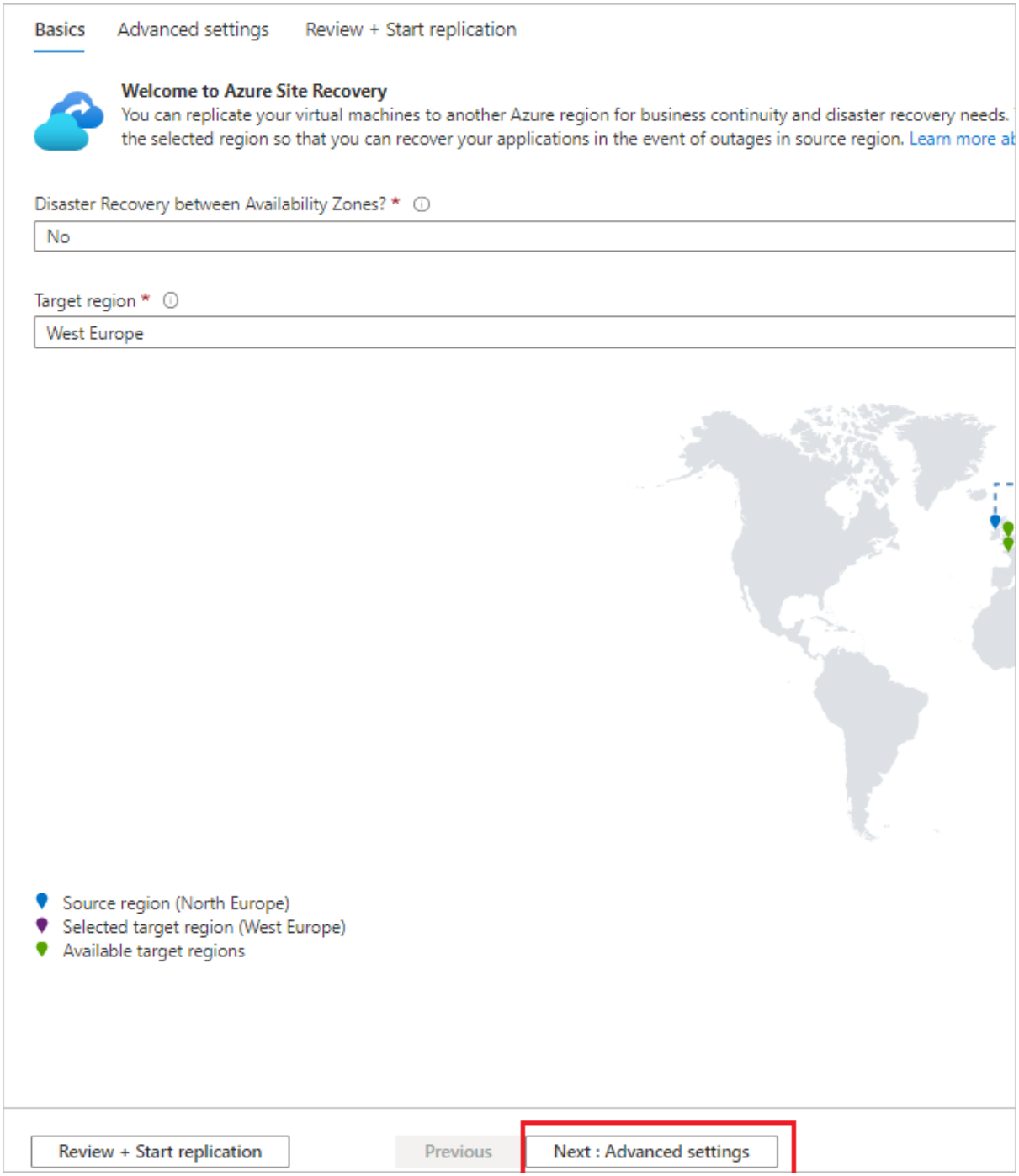
Enable disaster recovery for an existing VM

If you want to enable disaster recovery on an existing VM instead of for a new VM, use this procedure.

1. In the Azure portal, open the VM properties page.
2. In **Operations**, select **Disaster recovery**.

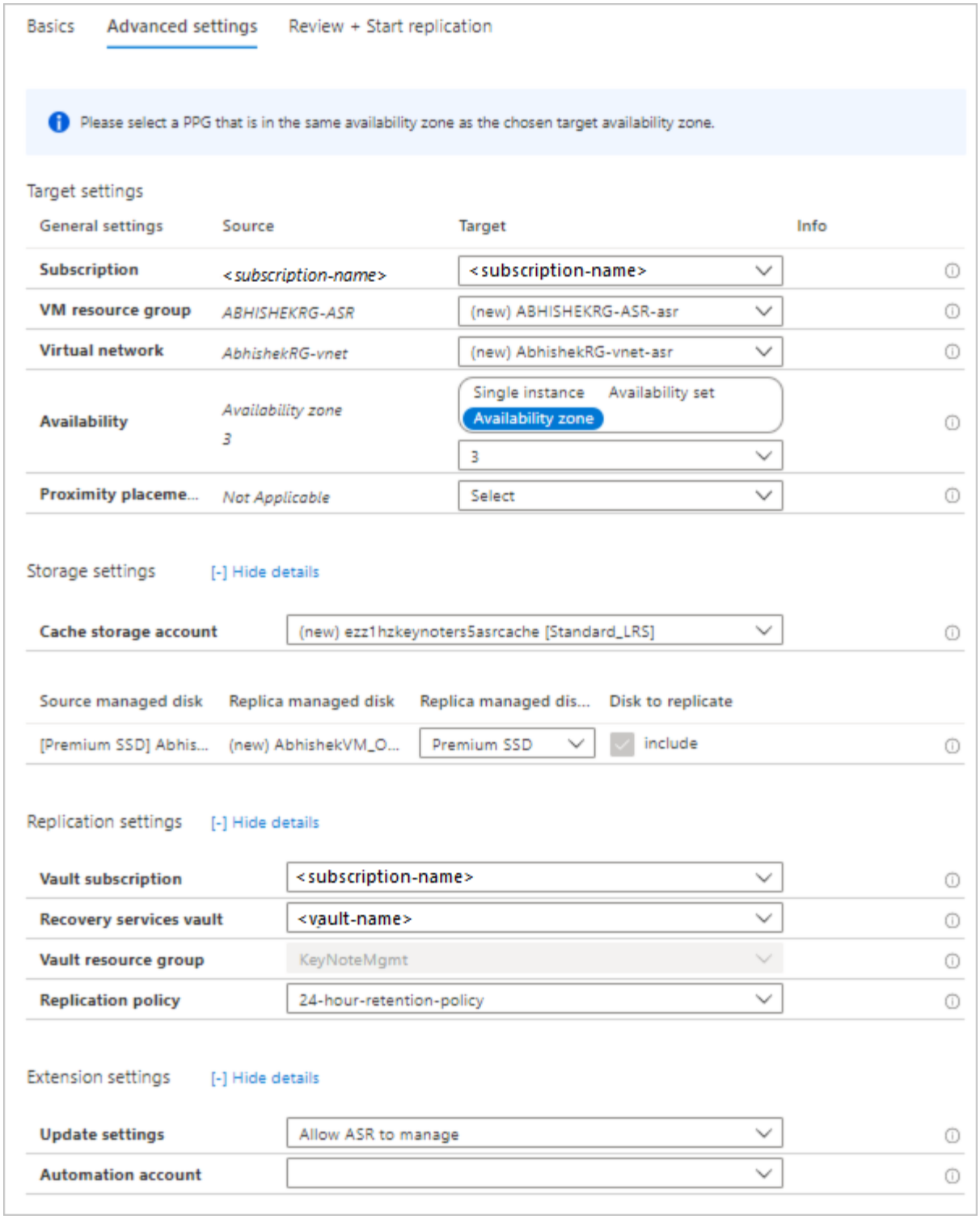


3. In **Basics**, if the VM is deployed in an availability zone, you can select disaster recovery between availability zones.
4. In **Target region**, select the region to which you want to replicate the VM. The source and target regions must be in the same Azure Active Directory tenant.

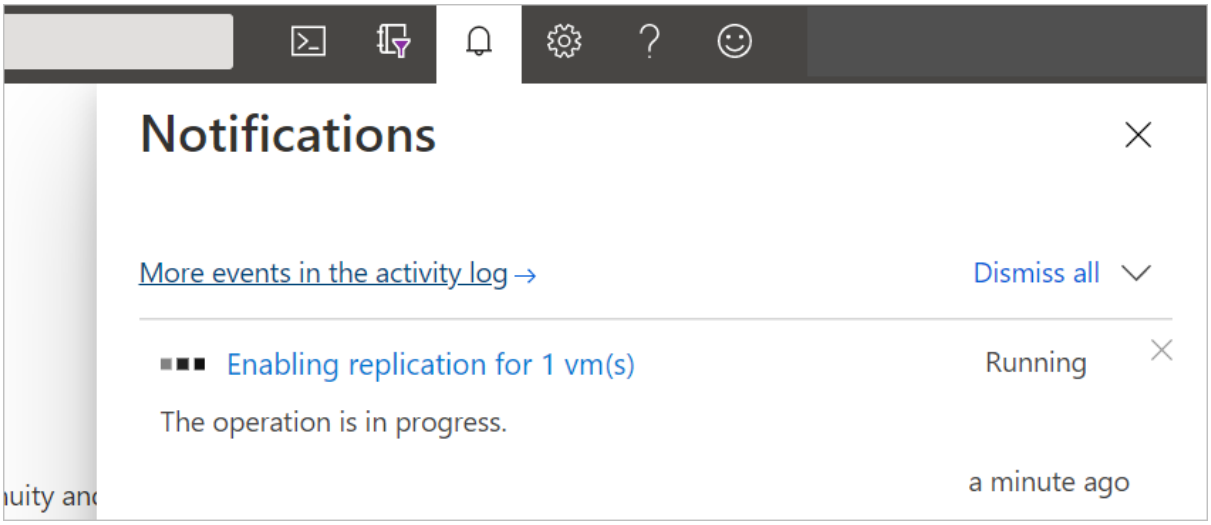


5. Select **Next: Advanced settings**.
6. In **Advanced settings**, you can review settings, and modify values to custom settings. By default, Site Recovery mirrors the source settings to create target resources.
    - **Target subscription**. The subscription in which the target VM is created after failover.
    - **Target VM resource group**. The resource group in which the target VM is created after failover.
    - **Target virtual network**. The Azure virtual network in which the target VM is located when it's created after failover.

- **Target availability**. When the target VM is created as a single instance, in an availability set, or availability zone.
- **Proximity placement**. If applicable, select the proximity placement group in which the target VM is located after failover.
- **Storage settings-Cache storage account**. Recovery uses a storage account in the source region as a temporary data store. Source VM changes are cached in this account, before being replicated to the target location.
    - By default one cache storage account is created per vault and reused.
    - You can select a different storage account if you want to customize the cache account for the VM.
- **Storage settings-Replica managed disk**. By default, Site Recovery creates replica managed disks in the target region.
    - By default the target managed disk mirror the source VM managed disks, using the same storage type (standard HDD/SSD, or premium SSD).
    - You can customize the storage type as needed.
- **Replication settings**. Shows the vault in which the VM is located, and the replication policy used for the VM. By default, recovery points created by Site Recovery for the VM are kept for 24 hours.
- **Extension settings**. Indicates that Site Recovery manages updates to the Site Recovery Mobility Service extension that's installed on VMs you replicate.
    - The indicated Azure automation account manages the update process.
    - You can customize the automation account.



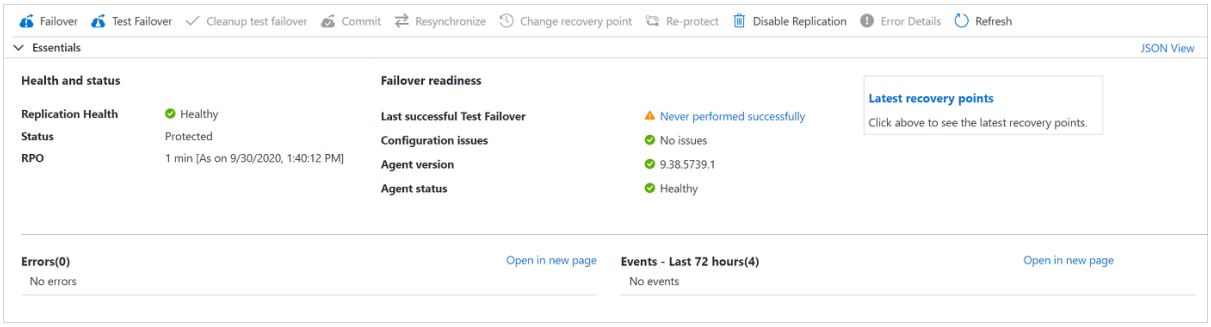
7. Select **Review + Start replication**.
8. Select **Start replication**. Deployment starts, and Site Recovery starts creating target resources. You can monitor replication progress in the notifications.



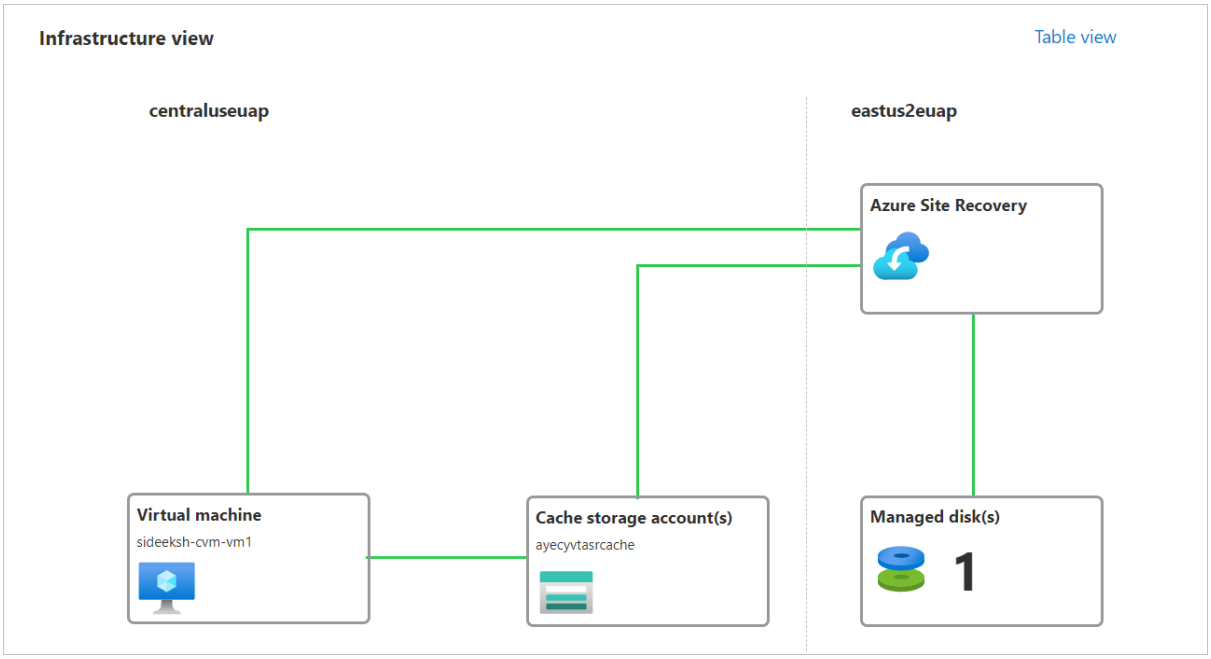
Check VM status

After the replication job finishes, you can check the VM replication status.

1. Open the VM properties page.
2. In **Operations**, select **Disaster recovery**.
3. Expand the **Essentials** section to review defaults about the vault, replication policy, and target settings.
4. In **Health and status**, get information about replication state for the VM, the agent version, failover readiness, and the latest recovery points.



5. In **Infrastructure view**, get a visual overview of source and target VMs, managed disks, and the cache storage account.



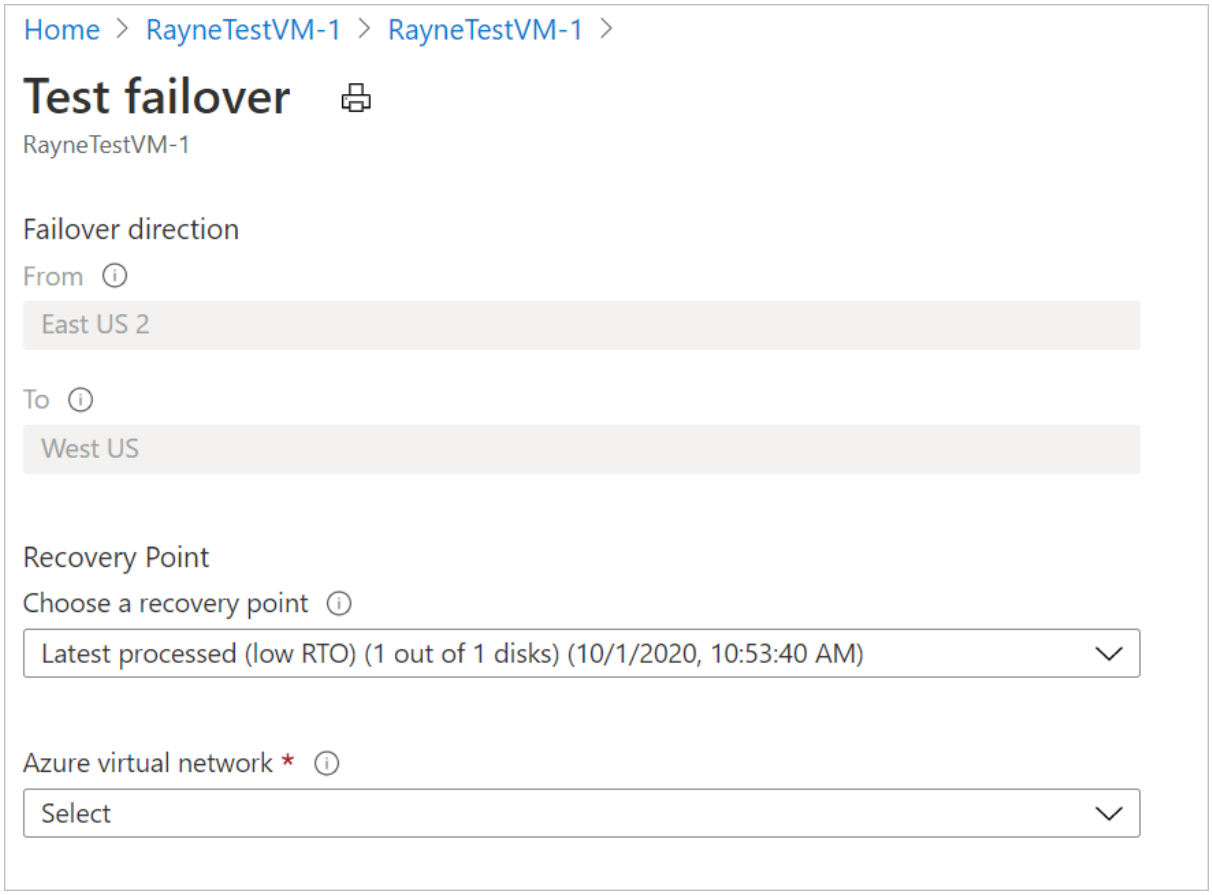
Run a drill

Run a drill to make sure disaster recovery works as expected. When you run a test failover, it creates a copy of the VM, with no impact on ongoing replication, or on your production environment.

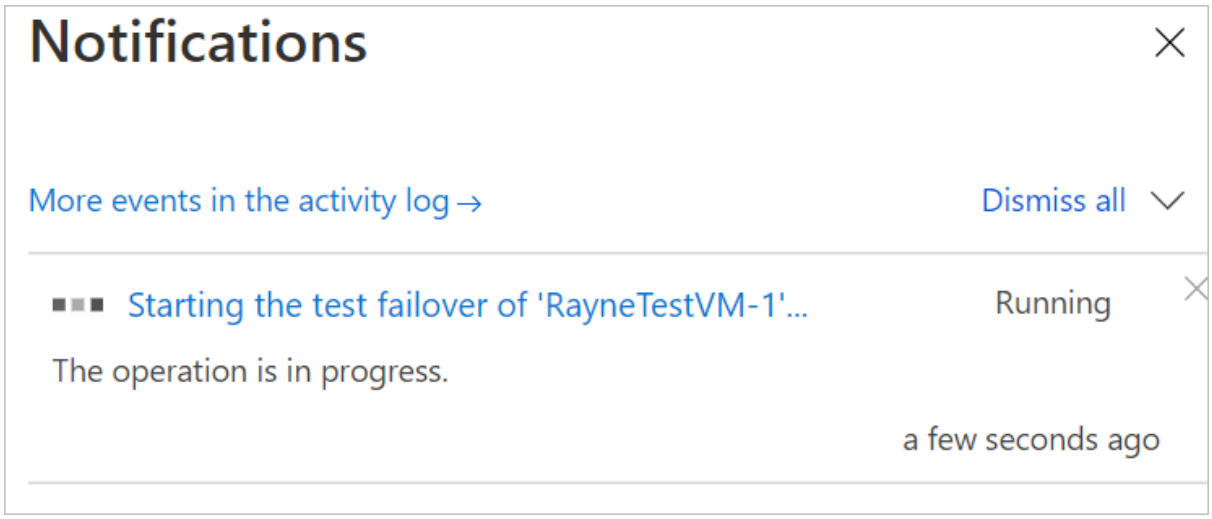
1. In the VM disaster recovery page, select **Test failover**.
2. In **Test failover**, leave the default **Latest processed (low RPO)** setting for the recovery point.

   This option provides the lowest recovery point objective (RPO), and generally the quickest spin up of the target VM. It first processes all the data that has been sent to Site Recovery service, to create a recovery point for each VM, before failing over to it. This recovery point has all the data replicated to Site Recovery when the failover was triggered.

3. Select the virtual network in which the VM will be located after failover.



4. The test failover process begins. You can monitor the progress in notifications.

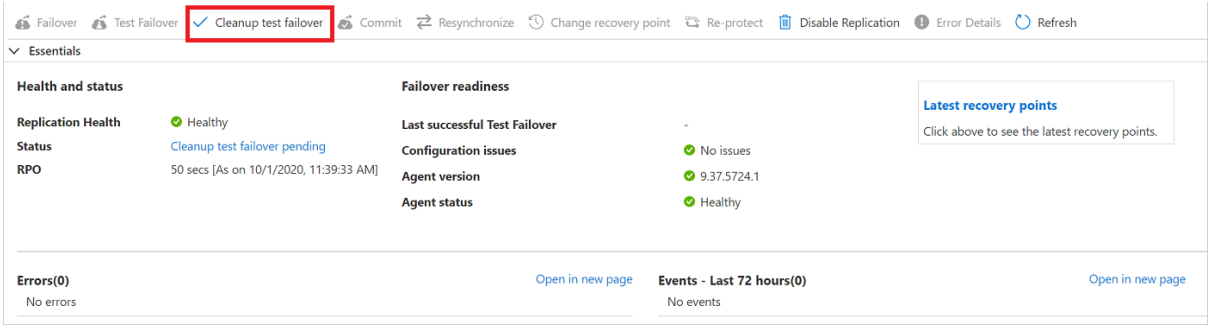


After the test failover completes, the VM is in the *Cleanup test failover pending* state on the **Essentials** page.

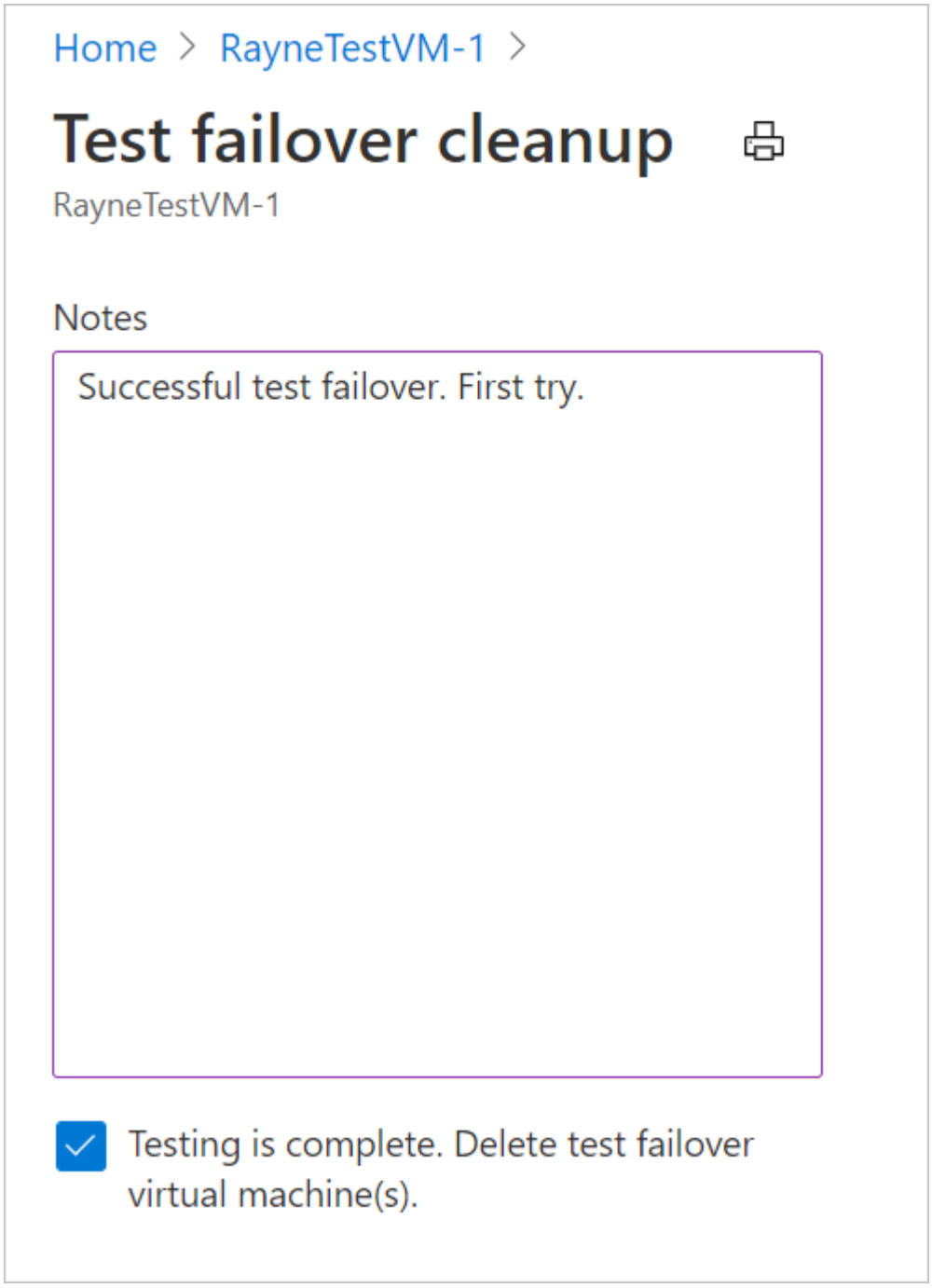
Clean up resources

The VM is automatically cleaned up by Site Recovery after the drill.

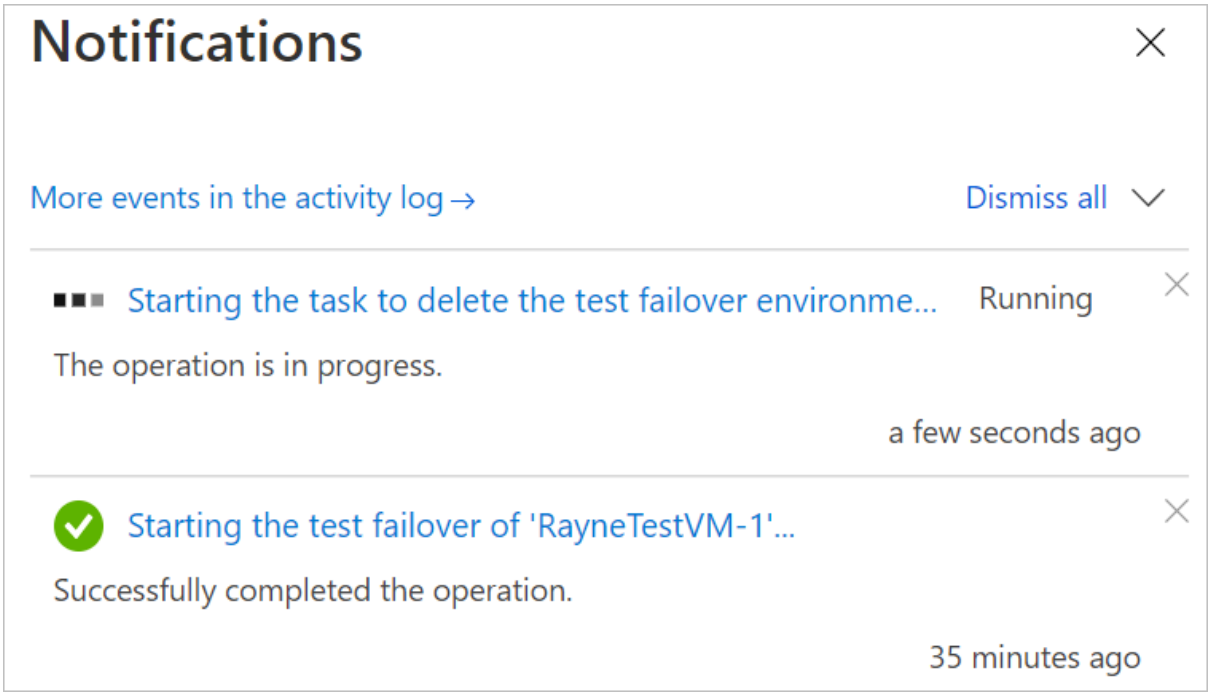
1. To begin automatic cleanup, select **Cleanup test failover**.



2. In **Test failover cleanup**, type in any notes you want to record for the failover, and then select **Testing is complete. Delete test failover virtual machine**. Then select **OK**.



3. The delete process begins. You can monitor progress in notifications.



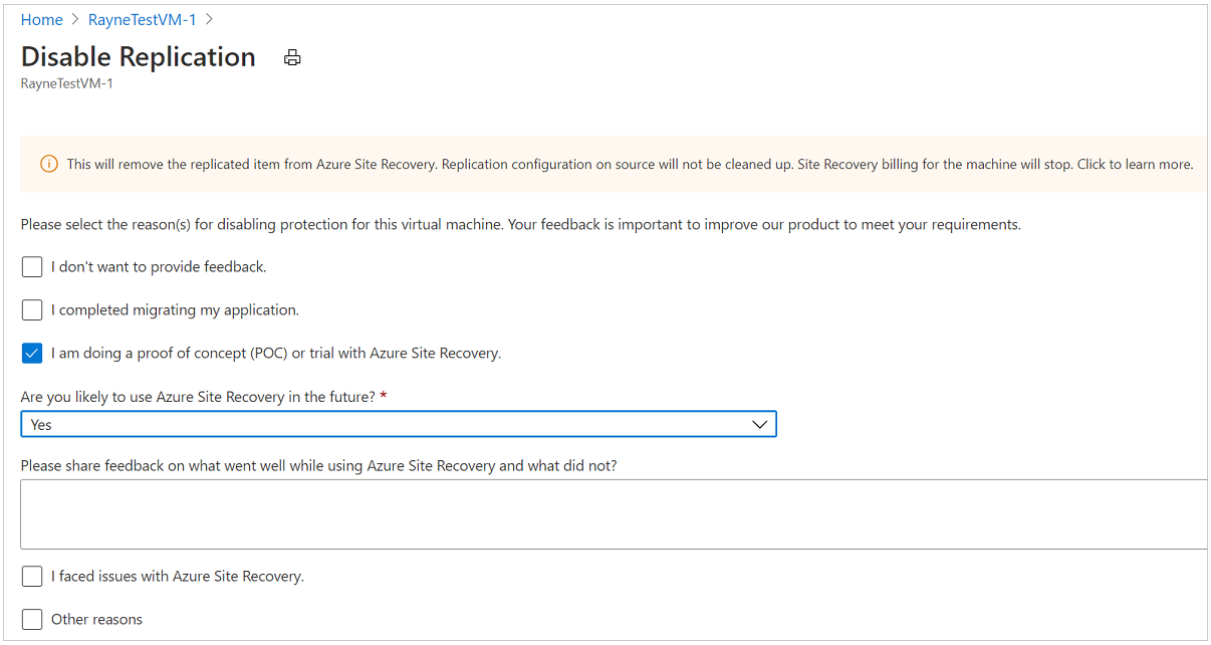
Stop replicating the VM

After completing a disaster recovery drill, we suggest you continue to try out a full failover. If you don't want to do a full failover, you can disable replication. This does the following:

- Removes the VM from the Site Recovery list of replicated machines.
- Stops Site Recovery billing for the VM.
- Automatically cleans up source replication settings.

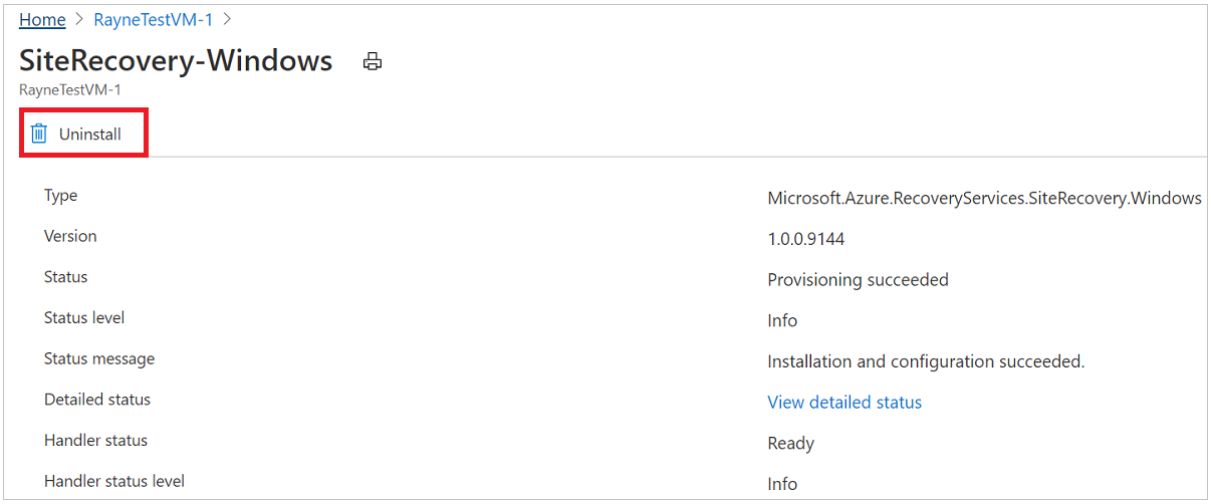
Stop replication as follows:

1. In the VM disaster recovery page, select **Disable Replication**.
2. In **Disable Replication**, select the reasons that you want to disable replication. Then select **OK**.



The Site Recovery extension installed on the VM during replication isn't removed automatically. If you disable replication for the VM, and you don't want to replicate it again at a later time, you can remove the Site Recovery extension manually, as follows:

1. Go to the VM > **Settings** > **Extensions**.
2. In the **Extensions** page, select each *Microsoft.Azure.RecoveryServices* entry for Linux.
3. In the properties page for the extension, select **Uninstall**.



Tutorial: Fail over Azure VMs to a secondary region

- Article
- 02/01/2023
- 6 contributors

Feedback

In this article

Learn how to fail over Azure VMs that are enabled for disaster recovery with Azure Site Recovery, to a secondary Azure region. After failover, you reprotect VMs in the target region so that they replicate back to the primary region. In this article, you learn how to:

- Check prerequisites
- Verify VM settings
- Run a failover to the secondary region
- Start replicating the VM back to the primary region.

Note

This tutorial shows you how to fail over VMs with minimal steps. If you want to run a failover with full settings, learn about Azure VM **networking**, **automation**, and **troubleshooting**.

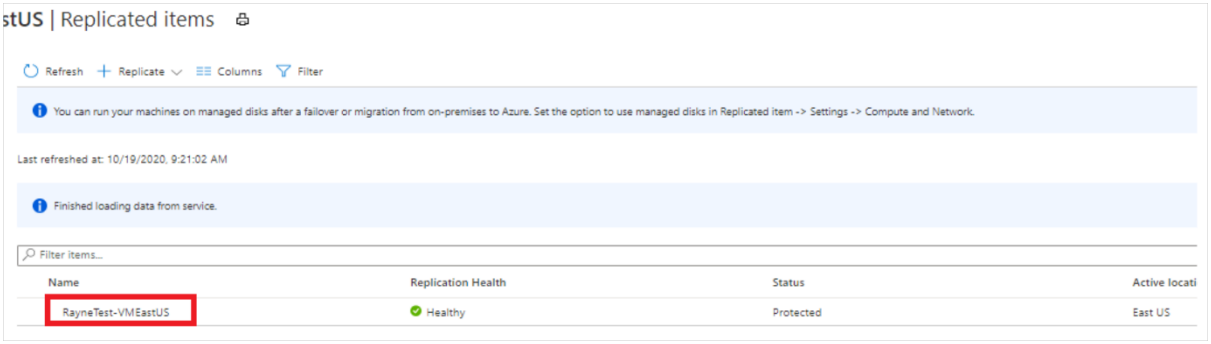
Prerequisites

Before you start this tutorial, you should have:

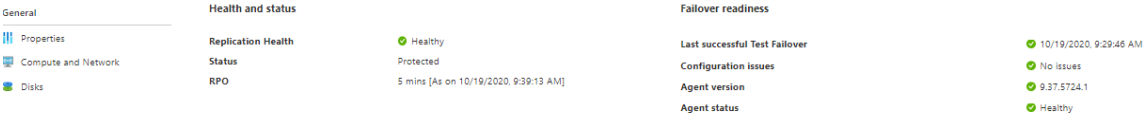
1. Set up replication for one or more Azure VMs. If you haven't, complete the first tutorial in this series to do that.
2. We recommend you run a disaster recovery drill for replicated VMs. Running a drill before you run a full failover helps ensure everything works as expected, without impacting your production environment.

Verify the VM settings

1. In the vault > **Replicated items**, select the VM.



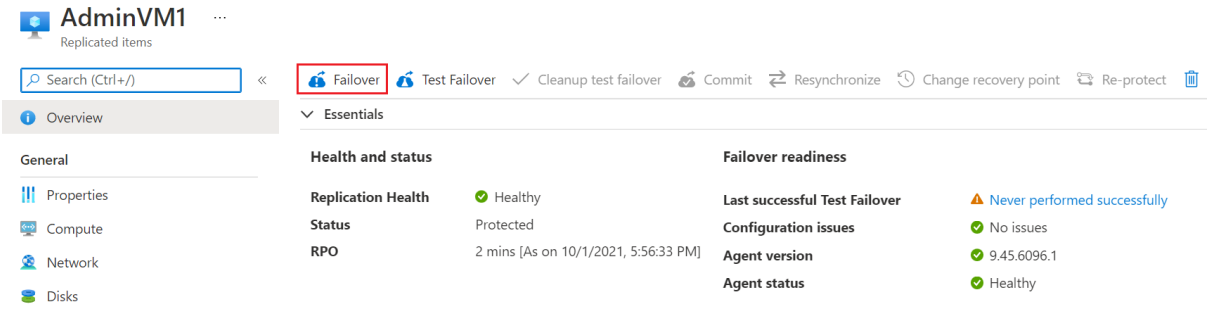
2. On the VM **Overview** page, check that the VM is protected and healthy, before you run a failover.



3. Before you fail over, check that:
    - The VM is running a supported Windows or Linux operating system.
    - The VM complies with compute, storage, and networking requirements.

Run a failover

1. On the VM **Overview** page, select **Failover**.



2. In **Failover**, choose a recovery point. The Azure VM in the target region is created using data from this recovery point.
    - **Latest processed**: Uses the latest recovery point processed by Site Recovery. The time stamp is shown. No time is spent

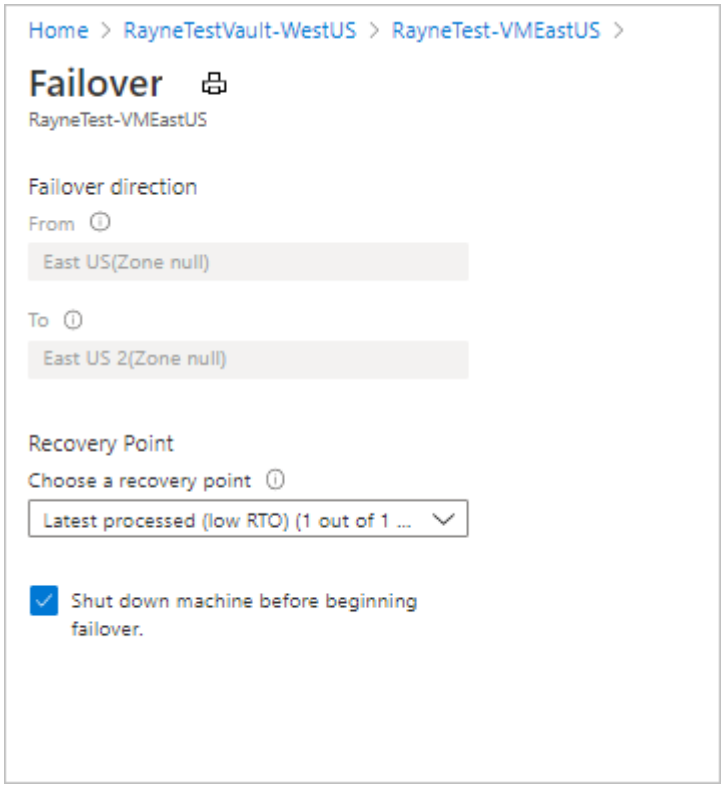
processing data, so it provides a low recovery time objective (RTO).

- **Latest**: Processes all the data sent to Site Recovery, to create a recovery point for each VM before failing over to it. Provides the lowest recovery point objective (RPO), because all data is replicated to Site Recovery when the failover is triggered.
- **Latest app-consistent**: This option fails over VMs to the latest app-consistent recovery point. The time stamp is shown.
- **Custom**: Fail over to particular recovery point. Custom is only available when you fail over a single VM, and don't use a recovery plan.

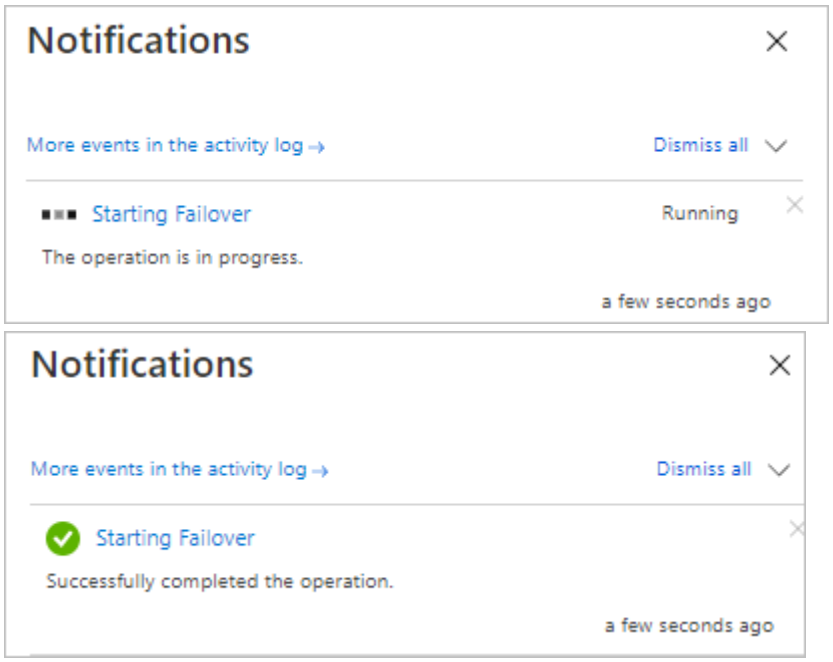
Note

If you added a disk to a VM after you enabled replication, replication points shows disks available for recovery. For example, a replication point created before you added a second disk will show as "1 of 2 disks".

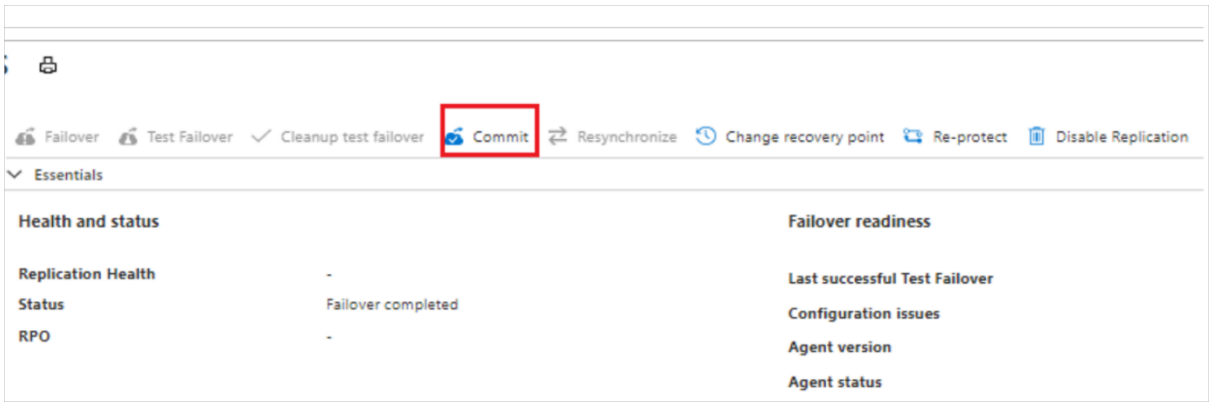
3. Select **Shut down machine before beginning failover** if you want Site Recovery to try to shut down the source VMs before starting failover. Shutdown helps to ensure no data loss. Failover continues even if shutdown fails.



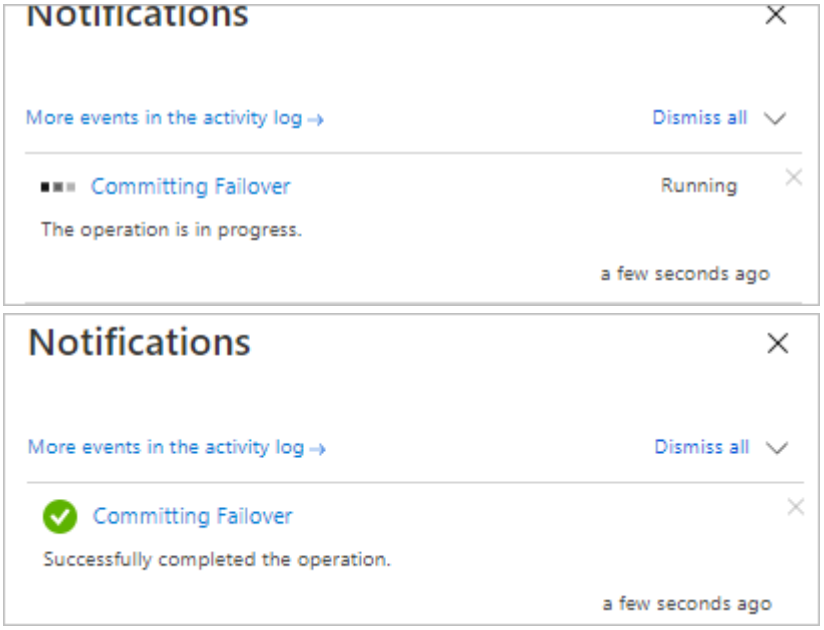
4. To start the failover, select **OK**.
5. Monitor the failover in notifications.



6. After the failover, the Azure VM created in the target region appears in **Virtual Machines**. Make sure that the VM is running, and sized appropriately. If you want to use a different recovery point for the VM, select **Change recovery point**, on the **Essentials** page.
7. When you're satisfied with the failed over VM, select **Commit** on the overview page, to finish the failover.



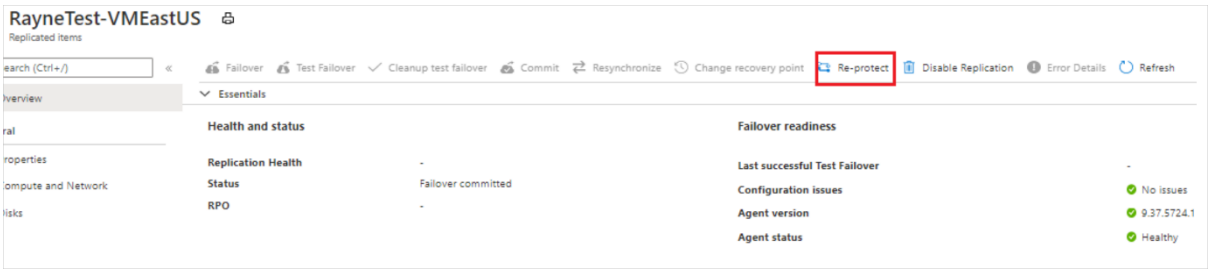
8. In **Commit**, select **OK** to confirm. Commit deletes all the available recovery points for the VM in Site Recovery, and you won't be able to change the recovery point.
9. Monitor the commit progress in notifications.



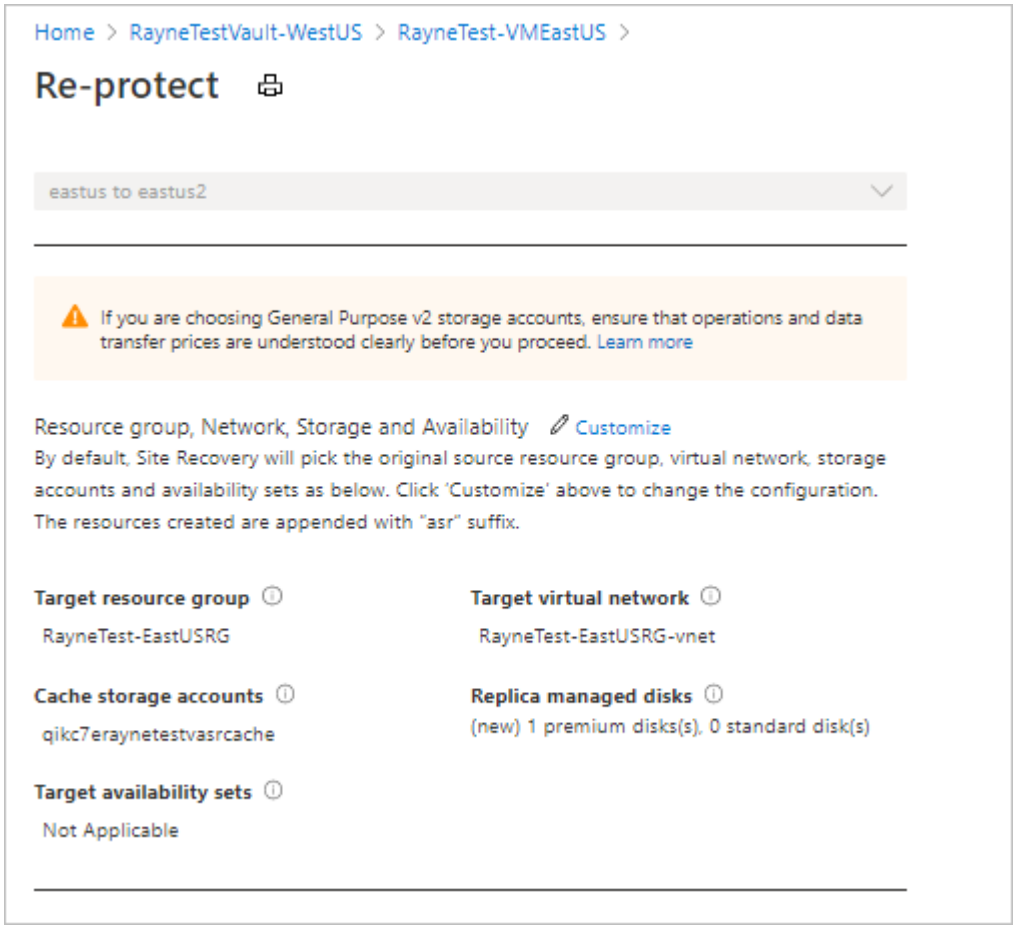
Reprotect the VM

After failover, you reprotect the VM in the secondary region, so that it replicates back to the primary region.

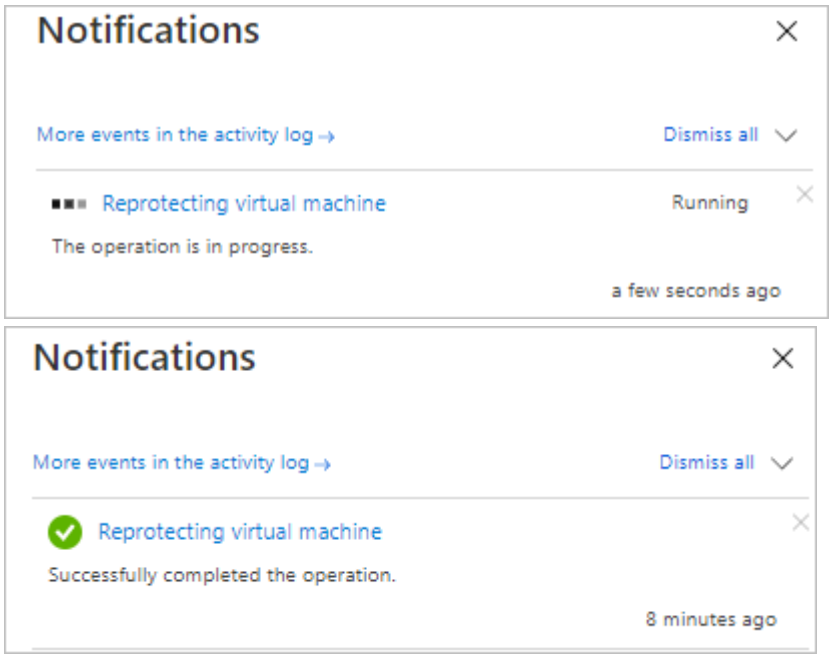
1. Make sure that VM **Status** is *Failover committed* before you start.
2. Check that you can access the primary region is available, and that you have permissions to create VMs in it.
3. On the VM **Overview** page, select **Re-Protect**.



4. In **Re-protect**, verify the replication direction (secondary to primary region), and review the target settings for the primary region. Resources marked as new are created by Site Recovery as part of the reprotect operation.



5. Select **OK** to start the reprotect process. The process sends initial data to the target location, and then replicates delta information for the VMs to the target.
6. Monitor reprotect progress in the notifications.



Tutorial: Fail back Azure VM to the primary region

- Article
- 04/21/2023
- 9 contributors

Feedback

In this article

After failing over an Azure VM to a secondary Azure region, follow this tutorial to fail the VM to the primary Azure region, using Azure Site Recovery. In this article, you learn how to:

- Review the prerequisites.
- Fail back the VM in the secondary region.
- Reprotect primary VMs back to the secondary region.

Note

This tutorial shows you how to fail back with minimal steps. If you want to run a failover with full settings, learn about Azure VM **networking**, **automation**, and **troubleshooting**.

Prerequisites

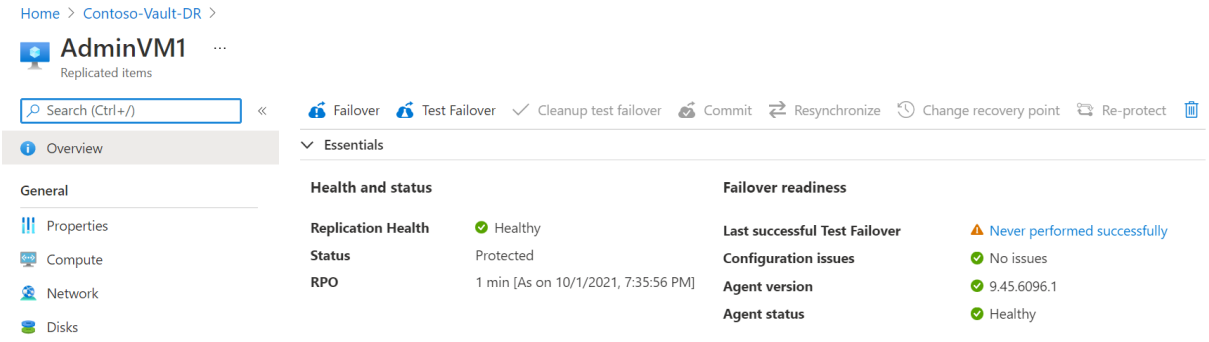
Before you start this tutorial, you should have:

1. Set up replication for at least one Azure VM, and tried out a disaster recovery drill for it.
2. Failed over the VM from the primary region to a secondary region, and reprotected it so that it replicates from the secondary region to the primary.
3. Check that the primary region is available, and that you're able to create and access new resources in it.

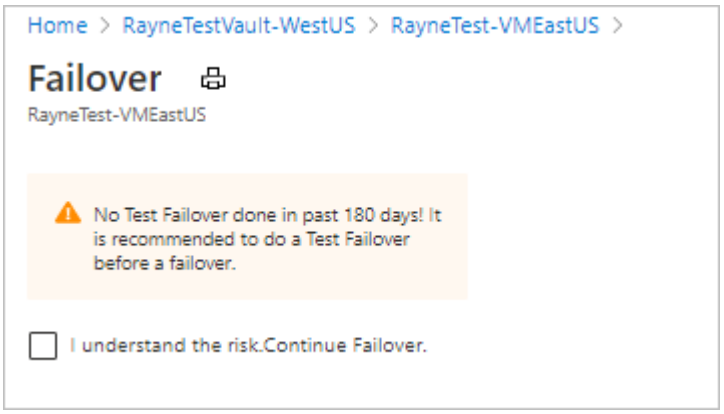
Fail back to the primary region

After VMs are reprotected, you can fail back to the primary region as needed.

1. In the vault > **Replicated items**, select the VM.
2. On the VM overview page, check that the VM is healthy, and that synchronization is complete, before you run a failover. The VM should be in a *Protected* state.



3. On the overview page, select **Failover**. Since we're not doing a test failover this time, we're prompted to verify.



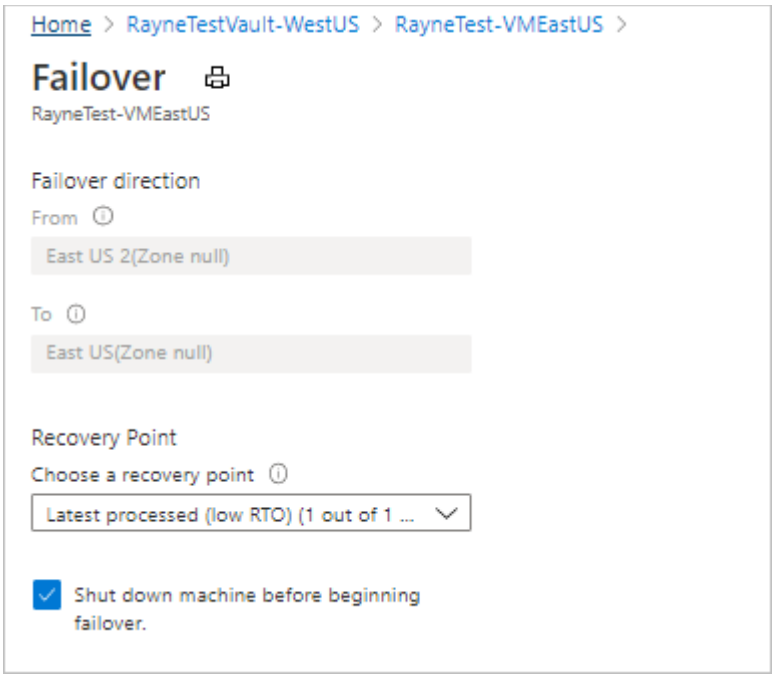
4. In **Failover**, note the direction from secondary to primary, and select a recovery point. The Azure VM in the target (primary region) is created using data from this point.
    - **Latest processed**: Uses the latest recovery point processed by Site Recovery. The time stamp is shown. No time is spent processing data, so it provides a low recovery time objective (RTO).
    - **Latest**: Processes all the data sent to Site Recovery, to create a recovery point for each VM before failing over to it. Provides the lowest recovery point objective (RPO), because all data is replicated to Site Recovery when the failover is triggered.

- **Latest app-consistent**: This option fails over VMs to the latest app-consistent recovery point. The time stamp is shown.
- **Custom**: Fail over to particular recovery point. Custom is only available when you fail over a single VM, and don't use a recovery plan.

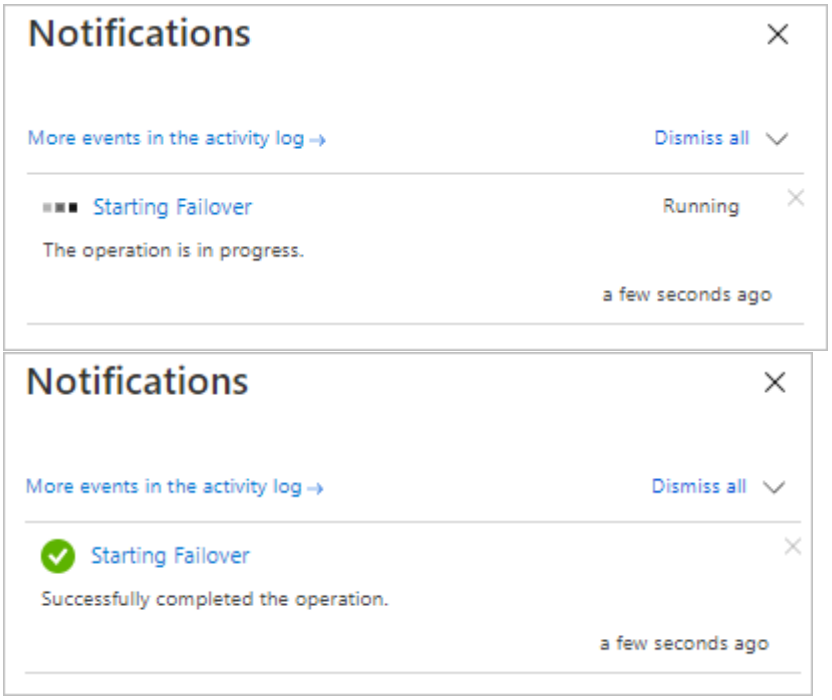
Note

If you fail over a VM to which you added a disk after you enabled replication for the VM, replication points will show the disks available for recovery. For example, a replication point that was created before you added a second disk will show as "1 of 2 disks".

5. Select **Shut down machine before beginning failover** if you want Site Recovery to attempt to shut down the source VMs before starting failover. Shutdown helps to ensure no data loss. Failover continues even if shutdown fails.



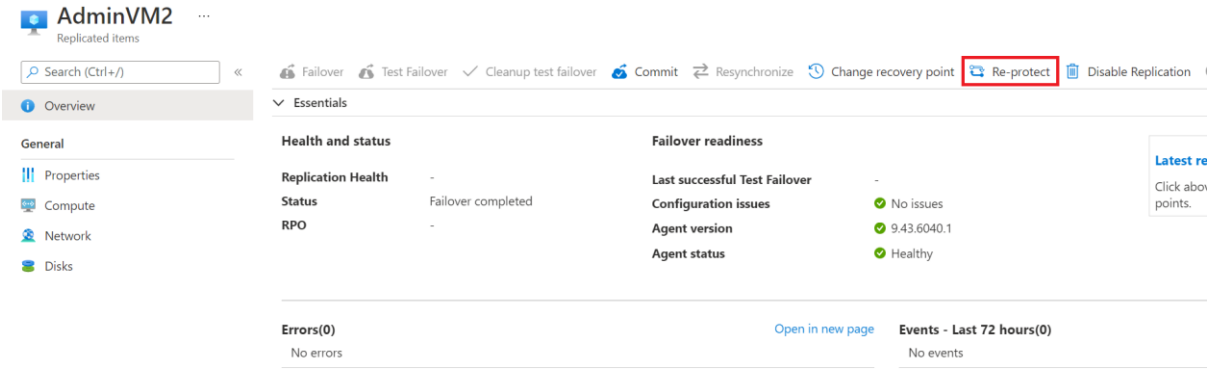
6. To start the failover, select **OK**.
7. Monitor the failover in notifications.



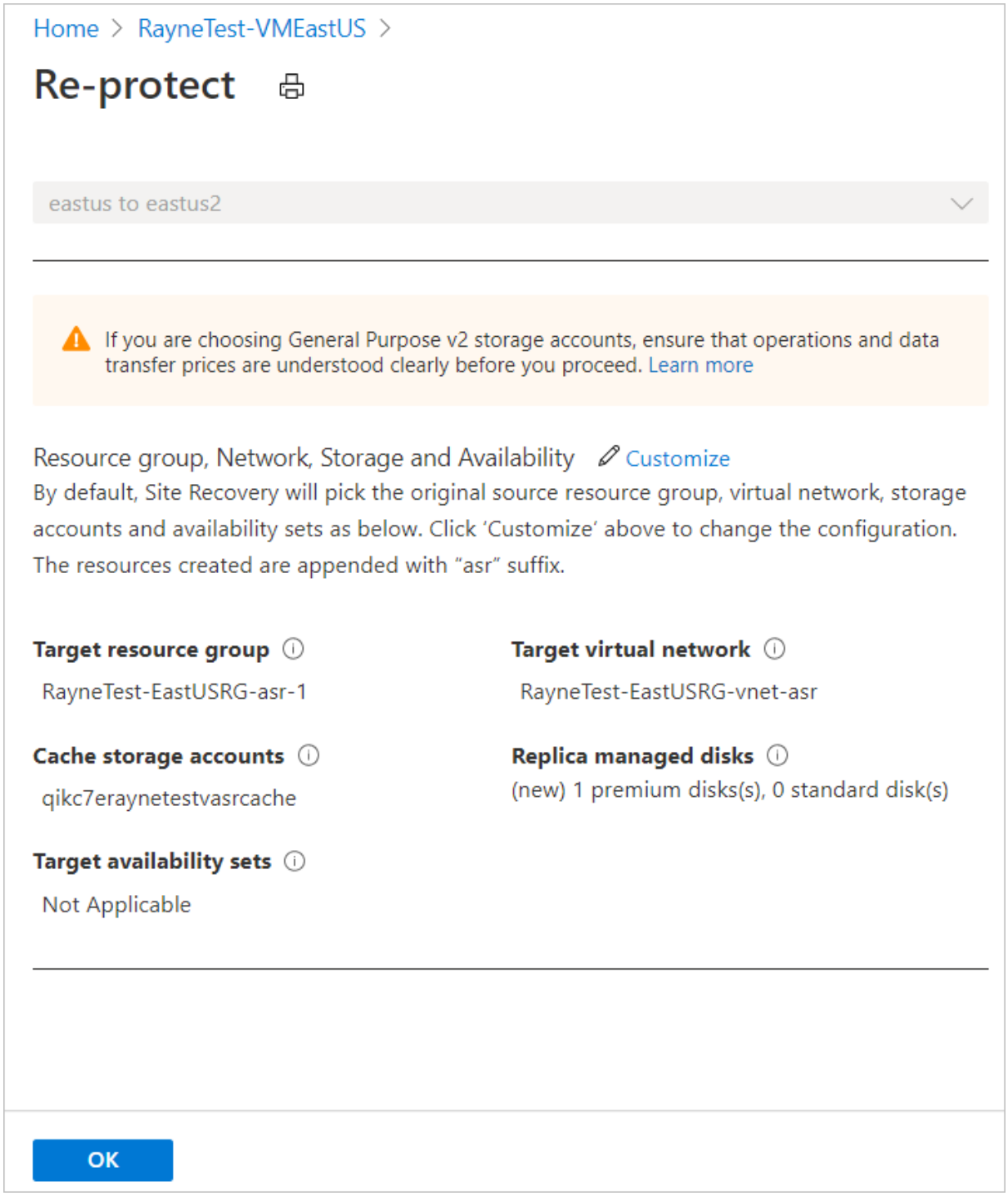
Reprotect VMs

After failing back VMs to the primary region, you need to reprotect them, so that they start replicating to the secondary region again.

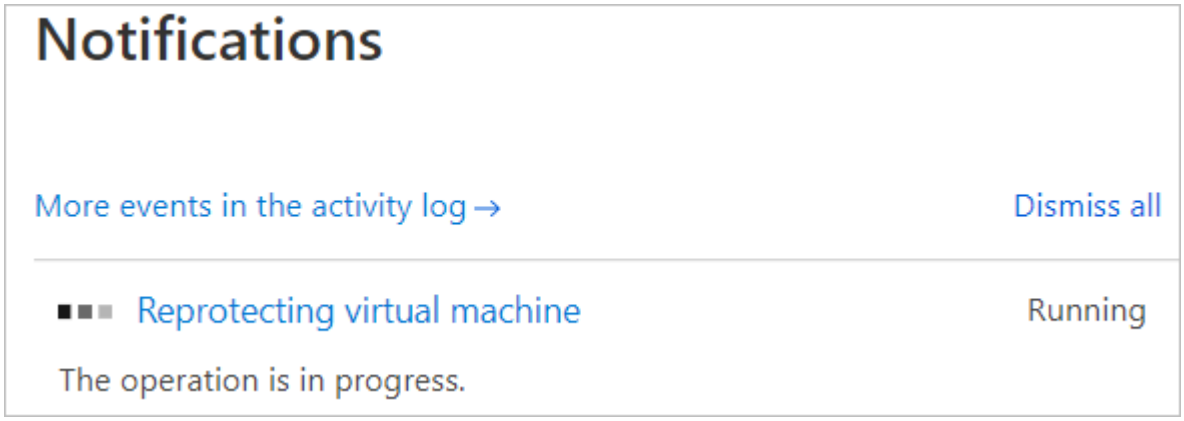
1. In the **Overview** page for the VM, select **Re-protect**.



2. Review the target settings for the primary region. Resources marked as new are created by Site Recovery as part of the reprotect operation.
3. Select **OK** to start the reprotect process. The process sends initial data to the target location, and then replicates delta information for the VMs to the target.



4. Monitor reprotect progress in notifications.



Reprotect progress notification

Clean up resources

For VMs with managed disks, after failback is complete and VMs are reprotected for replication from primary to secondary, Site Recovery automatically cleans up machines in the secondary disaster recovery region. You don't need to manually delete VMs and NICs in the secondary region. VMs with unmanaged disks aren't cleaned up.

If you completely disable replication after failing back, Site Recovery cleans up machines protected by it. In this case, it also cleans up disks for VMs that don't use managed disks.

Quickstart: Create VM restore points using APIs

- Article
- 11/01/2022
- 2 contributors

Feedback

In this article

You can protect your data by taking backups at regular intervals. Azure VM restore point APIs are a lightweight option you can use to implement granular backup and retention policies. VM restore points support application consistency for VMs running Windows operating systems and support file system consistency for VMs running Linux operating system.

You can use the APIs to create restore points for your source VM in either the same region, or in other regions. You can also copy existing VM restore points between regions.

Prerequisites

- Learn more about the requirements for a VM restore point.
- Consider the limitations before creating a restore point.

Create VM restore points

The following sections outline the steps you need to take to create VM restore points with the Azure Compute REST APIs.

You can find more information in the Restore Points, PowerShell, and Restore Point Collections API documentation.

Step 1: Create a VM restore point collection

Before you create VM restore points, you must create a restore point collection. A restore point collection holds all the restore points for a specific VM. Depending on your needs, you can create VM restore points in the same region as the VM, or in a different region. To create a restore point collection, call the restore point collection's Create or Update API.

- If you're creating restore point collection in the same region as the VM, then specify the VM's region in the location property of the request body.
- If you're creating the restore point collection in a different region than the VM, specify the target region for the collection in the location property, but also specify the source restore point collection ARM resource ID in the request body.

To create a restore point collection, call the restore point collection's Create or Update API.

Step 2: Create a VM restore point

After you create the restore point collection, the next step is to create a VM restore point within the restore point collection. For more information about restore point creation, see the Restore Points - Create API documentation.

Tip

To save space and costs, you can exclude any disk from either local region or cross-region VM restore points. To exclude a disk, add its identifier to the `excludeDisks` property in the request body.

Step 3: Track the status of the VM restore point creation

Restore point creation in your local region will be completed within a few seconds. Scenarios, which involve the creation of cross-region restore points will take considerably longer. To track the status of the creation operation, follow the guidance in Get restore point copy or replication status. This is only applicable for scenarios where the restore points are created in a different region than the source VM.

Get restore point copy or replication status

Creation of a cross-region VM restore point is a long running operation. The VM restore point can be used to restore a VM only after the operation is completed for all disk restore points. To track the operation's status, call the Restore Point - Get API on the target VM restore point and include the `instanceView` parameter. The return will include the percentage of data that has been copied at the time of the request.

During restore point creation, the `ProvisioningState` will appear as `Creating` in the response. If creation fails, `ProvisioningState` is set to `Failed`.

Create virtual machine restore points using Azure CLI

- Article
- 03/31/2023
- 5 contributors

Feedback

In this article

You can protect your data and guard against extended downtime by creating VM restore points at regular intervals. You can create VM restore points, and exclude disks while creating the restore point, using Azure CLI. Azure CLI is used to create and manage Azure resources using command line or scripts. Alternatively, you can create VM restore points using the Azure portal or using PowerShell.

The az restore-point module is used to create and manage restore points from the command line or in scripts.

In this tutorial, you learn how to:

- Create a VM restore point collection
- Create a VM restore point
- Track the progress of Copy operation
- Restore a VM

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Quickstart for Bash in Azure Cloud Shell.

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.

    - Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.

- Learn more about the support requirements and limitations before creating a restore point.

Step 1: Create a VM restore point collection

Use the az restore-point collection create command to create a VM restore point collection, as shown below:

Copy
```
az restore-point collection create --location "norwayeast" --source-id
"/subscriptions/{subscription-
id}/resourceGroups/ExampleRg/providers/Microsoft.Compute/virtualMachines/ExampleVM
" --tags myTag1="tagValue1" --resource-group "ExampleRg" --collection-name
"ExampleRpc"
```

Step 2: Create a VM restore point

Create a VM restore point with the az restore-point create command as follows:

Copy
```
az restore-point create --resource-group "ExampleRg" --collection-name
"ExampleRpc" --name "ExampleRp"
```

Exclude disks when creating a restore point

Exclude the disks that you do not want to be a part of the restore point with the `--exclude-disks` parameter, as follows:

Copy
```
az restore-point create --exclude-disks "/subscriptions/{subscription-
id}/resourceGroups/ExampleRg/providers/Microsoft.Compute/disks/ExampleDisk1" --
resource-group "ExampleRg" --collection-name "ExampleRpc" --name "ExampleRp"
```

Step 3: Track the status of the VM restore point creation

Use the az restore-point show command to track the progress of the VM restore point creation.

Copy

```
az restore-point show --resource-group "ExampleRg" --collection-name "ExampleRpc"
--name "ExampleRp"
```

Restore a VM from VM restore point

To restore a VM from a VM restore point, first restore individual disks from each disk restore point. You can also use the ARM template to restore a full VM along with all the disks.

Copy

```
# Create Disks from disk restore points
$osDiskRestorePoint = az restore-point show --resource-group "ExampleRg" --
collection-name "ExampleRpc" --name "ExampleRp" --query
"sourceMetadata.storageProfile.dataDisks[0].diskRestorePoint.id"
$dataDisk1RestorePoint = az restore-point show --resource-group "ExampleRg" --
collection-name "ExampleRpcTarget" --name "ExampleRpTarget" –query
"sourceMetadata.storageProfile.dataDisks[0].diskRestorePoint.id"
$dataDisk2RestorePoint = az restore-point show --resource-group "ExampleRg" --
collection-name "ExampleRpcTarget" --name "ExampleRpTarget" –query
"sourceMetadata.storageProfile.dataDisks[0].diskRestorePoint.id"

az disk create --resource-group “ExampleRg” --name “ExampleOSDisk” --sku
Premium_LRS --size-gb 128 --source $osDiskRestorePoint

az disk create --resource-group “ExampleRg” --name “ExampleDataDisk1” --sku
Premium_LRS --size-gb 128 --source $dataDisk1RestorePoint

az disk create --resource-group “ExampleRg” --name “ExampleDataDisk1” --sku
Premium_LRS --size-gb 128 --source $dataDisk2RestorePoint
```

Once you have created the disks, create a new VM and attach these restored disks to the newly created VM.

Create virtual machine restore points using PowerShell

- Article
- 11/01/2022
- 4 contributors

Feedback

In this article

3. Step 2: Create a VM restore point
4. Step 3: Track the status of the VM restore point creation

Show 2 more

Note

We recommend that you use the Azure Az PowerShell module to interact with Azure. See **Install Azure PowerShell** to get started. To learn how to migrate to the Az PowerShell module, see **Migrate Azure PowerShell from AzureRM to Az**.

You can create Virtual Machine restore points using PowerShell scripts. The Azure PowerShell Az module is used to create and manage Azure resources from the command line or in scripts.

You can protect your data and guard against extended downtime by creating VM restore points at regular intervals. This article shows you how to create VM restore points, and exclude disks from the restore point, using the Az.Compute module. Alternatively, you can create VM restore points using the Azure CLI or in the Azure portal.

In this tutorial, you learn how to:

- Create a VM restore point collection
- Create a VM restore point
- Track the progress of Copy operation
- Restore a VM

Prerequisites

- Learn more about the support requirements and limitations before creating a restore point.

Step 1: Create a VM restore point collection

Use the New-AzRestorePointCollection cmdlet to create a VM restore point collection.

Copy

```
New-AzRestorePointCollection -ResourceGroupName ExampleRG -Name ExampleRPC -VmId
“/subscriptions/{SubscriptionId}/resourcegroups/
ExampleRG/providers/microsoft.compute/virtualmachines/Example-vm-1” -Location
“WestEurope”
```

Step 2: Create a VM restore point

Create a VM restore point with the New-AzRestorePoint cmdlet as shown below:

Copy
```
New-AzRestorePoint -ResourceGroupName ExampleRG -RestorePointCollectionName
ExampleRPC -Name ExampleRP
```

Exclude disks from the restore point

Exclude certain disks that you do not want to be a part of the restore point with the `-DisksToExclude` parameter, as follows:

Copy
```
New-AzRestorePoint -ResourceGroupName ExampleRG -RestorePointCollectionName
ExampleRPC -Name ExampleRP -DisksToExclude
“/subscriptions/{SubscriptionId}/resourcegroups/
ExampleRG/providers/Microsoft.Compute/disks/example-vm-1-data_disk_1”
```

Step 3: Track the status of the VM restore point creation

You can track the progress of the VM restore point creation using the Get-AzRestorePoint cmdlet, as follows:

Copy
```
Get-AzRestorePoint -ResourceGroupName ExampleRG -RestorePointCollectionName
ExampleRPC -Name ExampleRP
```

Restore a VM from VM restore point

To restore a VM from a VM restore point, first restore individual disks from each disk restore point. You can also use the ARM template to restore a full VM along with all the disks.

Copy
```
# Create Disks from disk restore points
$restorePoint = Get-AzRestorePoint -ResourceGroupName ExampleRG -
RestorePointCollectionName ExampleRPC -Name ExampleRP

$osDiskRestorePoint =
$restorePoint.SourceMetadata.StorageProfile.OsDisk.DiskRestorePoint.Id
$dataDisk1RestorePoint =
$restorePoint.sourceMetadata.storageProfile.dataDisks[0].diskRestorePoint.id
$dataDisk2RestorePoint =
$restorePoint.sourceMetadata.storageProfile.dataDisks[1].diskRestorePoint.id
```

```
New-AzDisk -DiskName “ExampleOSDisk” (New-AzDiskConfig  -Location eastus -
CreateOption Restore -SourceResourceId $osDiskRestorePoint) -ResourceGroupName
ExampleRg

New-AzDisk -DiskName “ExampleDataDisk1” (New-AzDiskConfig  -Location eastus -
CreateOption Restore -SourceResourceId $dataDisk1RestorePoint) -ResourceGroupName
ExampleRg

New-AzDisk -DiskName “ExampleDataDisk2” (New-AzDiskConfig  -Location eastus -
CreateOption Restore -SourceResourceId $dataDisk2RestorePoint) -ResourceGroupName
ExampleRg
```

After you create the disks, create a new VM and attach these restored disks to the newly created VM.

Create virtual machine restore points using Azure portal

- Article
- 11/01/2022
- 4 contributors

Feedback

In this article

You can create virtual machine restore points through the Azure portal. You can protect your data and guard against extended downtime by creating VM restore points at regular intervals. This article shows you how to create VM restore points using the Azure portal. Alternatively, you can create VM restore points using the Azure CLI or using PowerShell.

In this tutorial, you learn how to:

- Create a VM restore point collection
- Create a VM restore point
- Track the progress of Copy operation
- Restore a VM

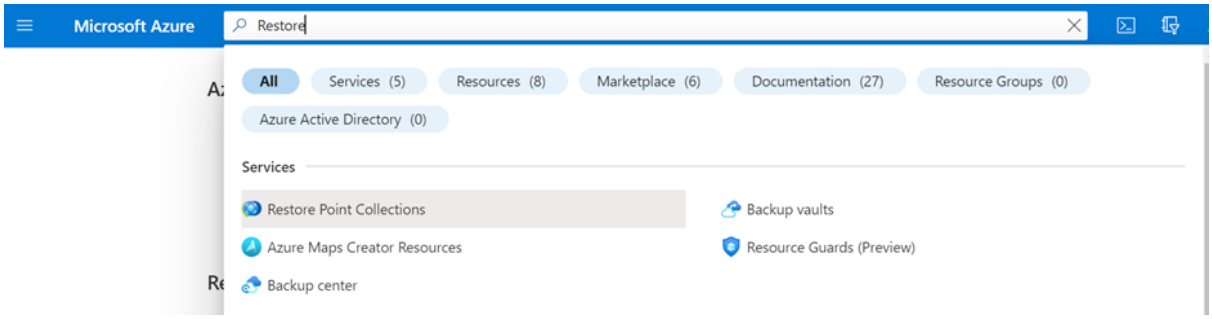
Prerequisites

- Learn more about the support requirements and limitations before creating a restore point.

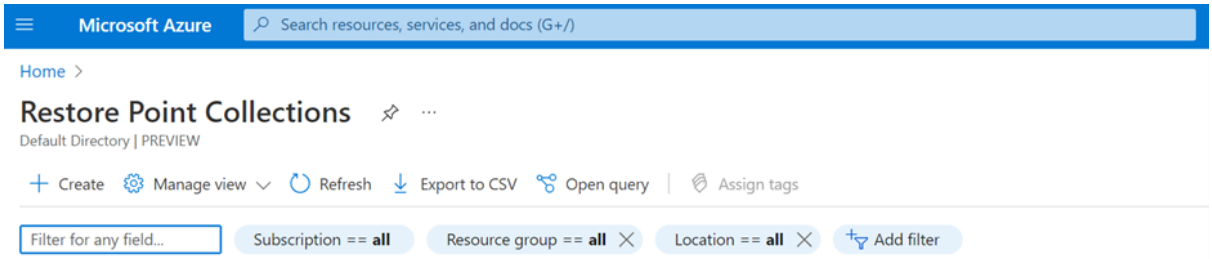
Step 1: Create a VM restore point collection

Use the following steps to create a VM restore points collection:

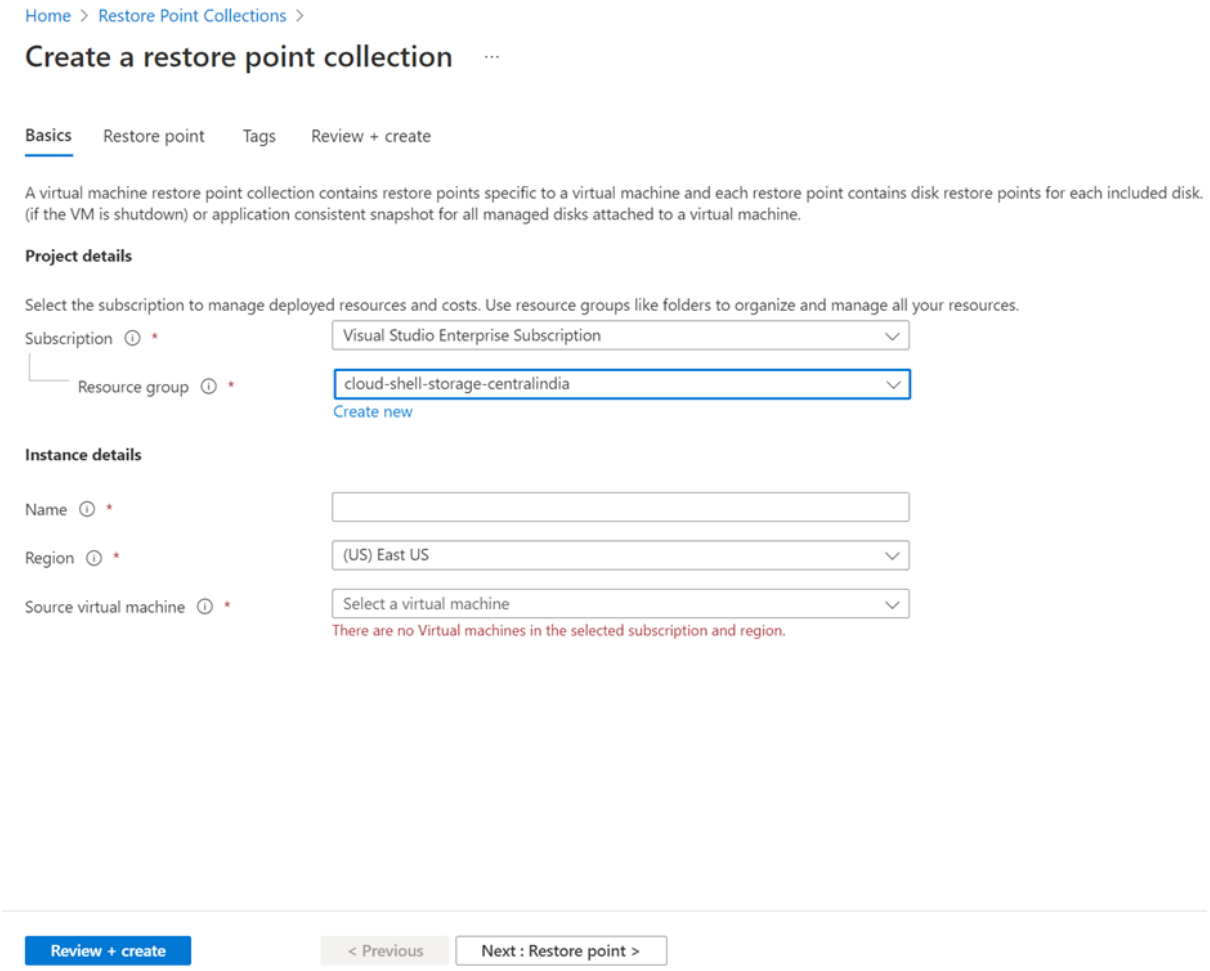
1. Sign in to the Azure portal.
2. In the Search box, enter **Restore Point Collections**.



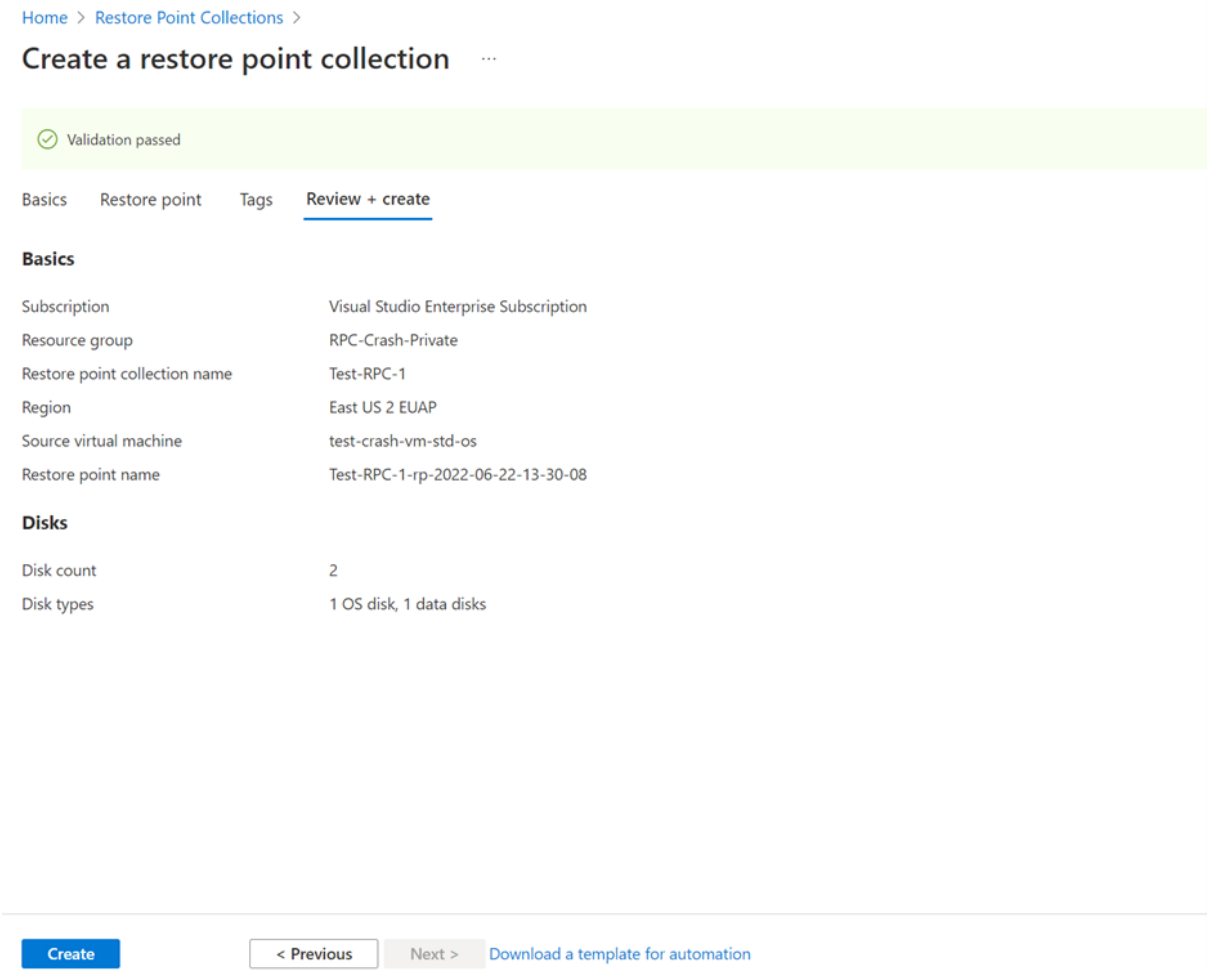
3. Select **+ Create** to create a new Restore Point Collection.



4. Enter the details and select the VM for which you want to create a restore point collection.



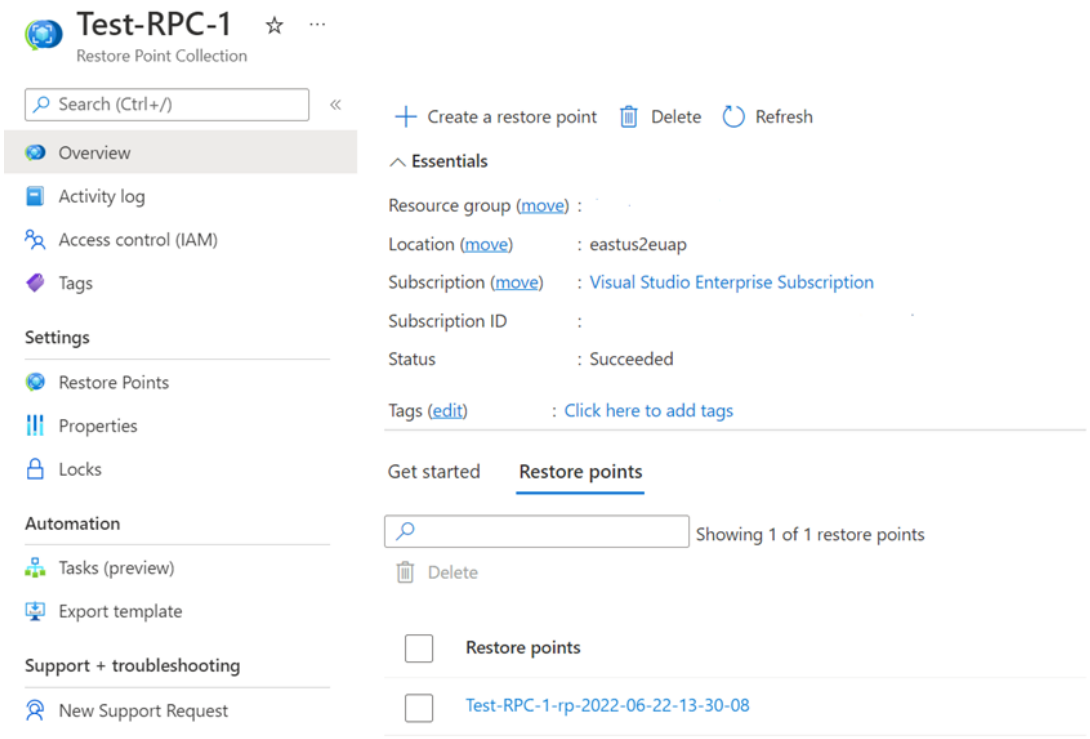
5. Select **Next: Restore Point** to create your first restore point or select **Review + Create** to create an empty restore point collection.



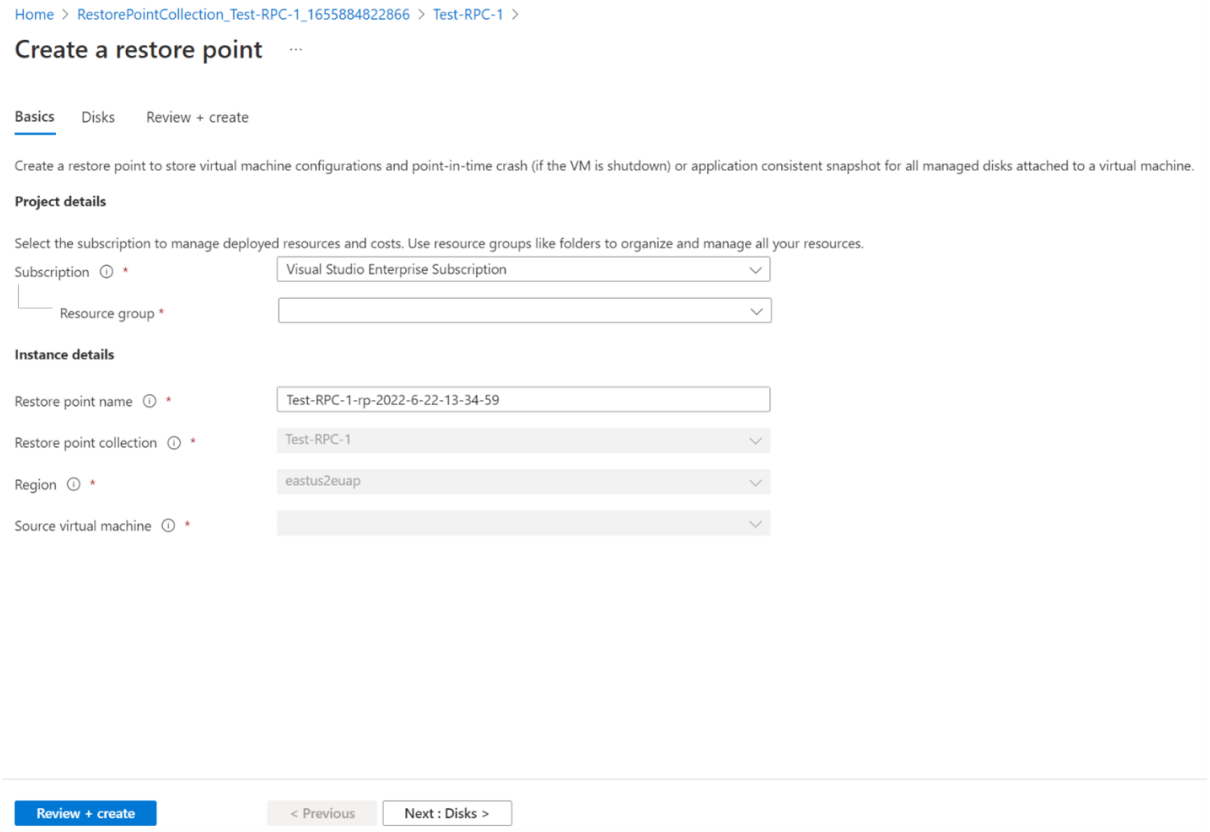
Step 2: Create a VM restore point

Use the following steps to create a VM restore point:

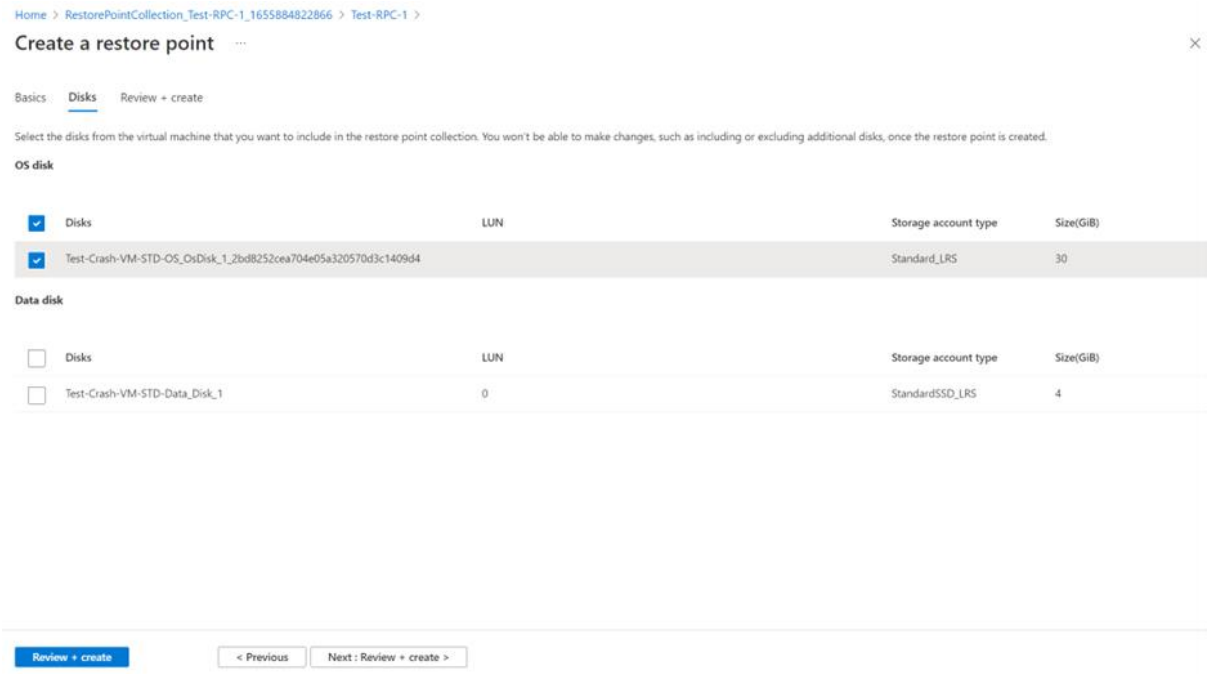
1. Navigate to the restore point collection where you want to create restore points and select **+ Create a restore point** to create new restore point for the VM.



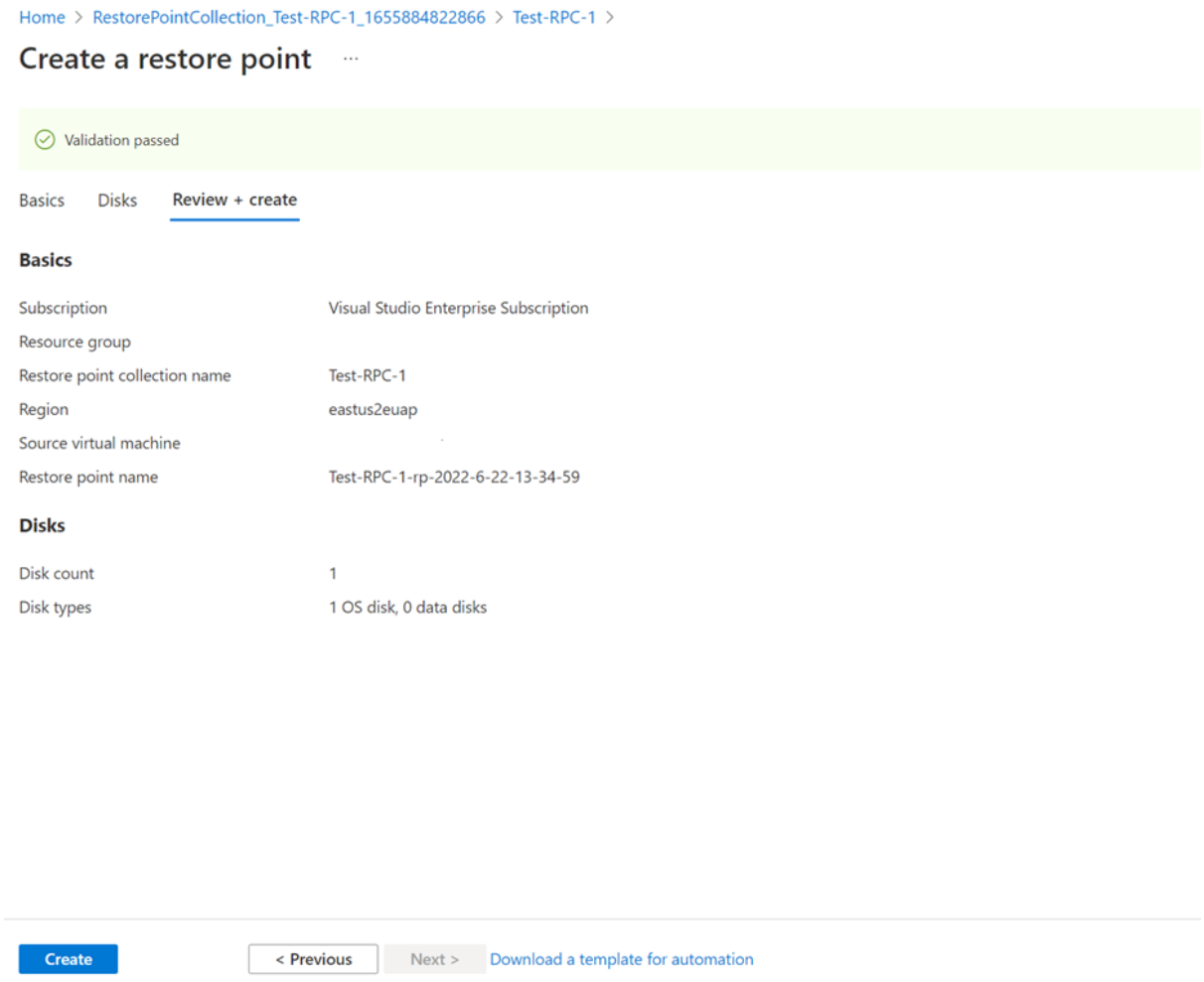
2. Enter a name for the restore point and other required details and select **Next: Disks >**.



3. Select the disks to be included in the restore point.

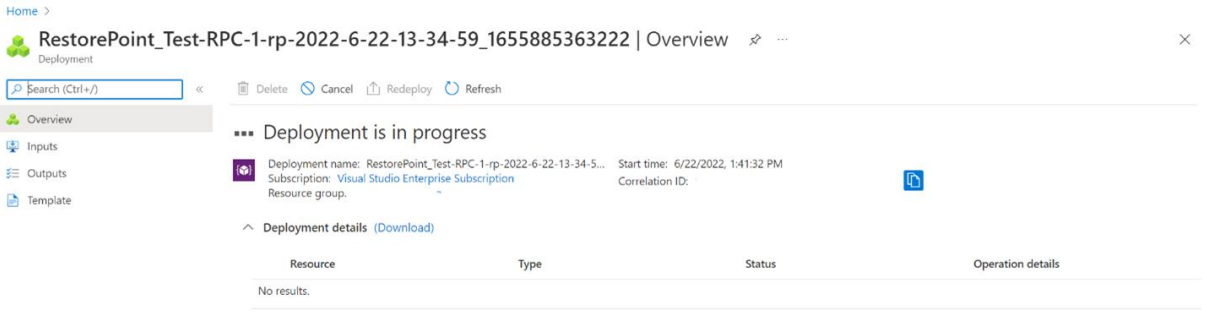


4. Select **Review + create** to validate the settings. Once validation is completed, select **Create** to create the restore point.



Step 3: Track the status of the VM restore point creation

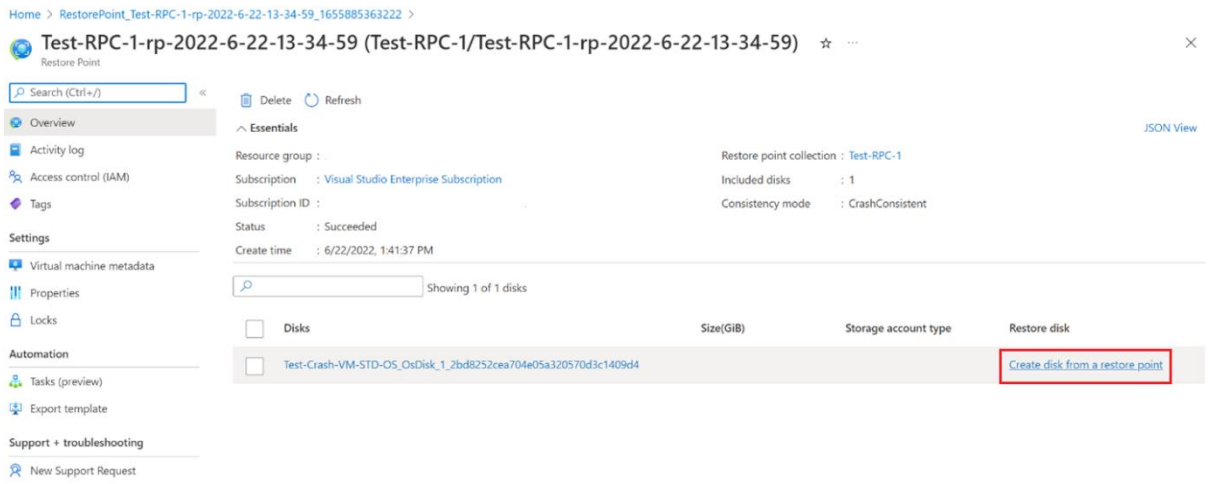
1. Select the notification to track the progress of the restore point creation.



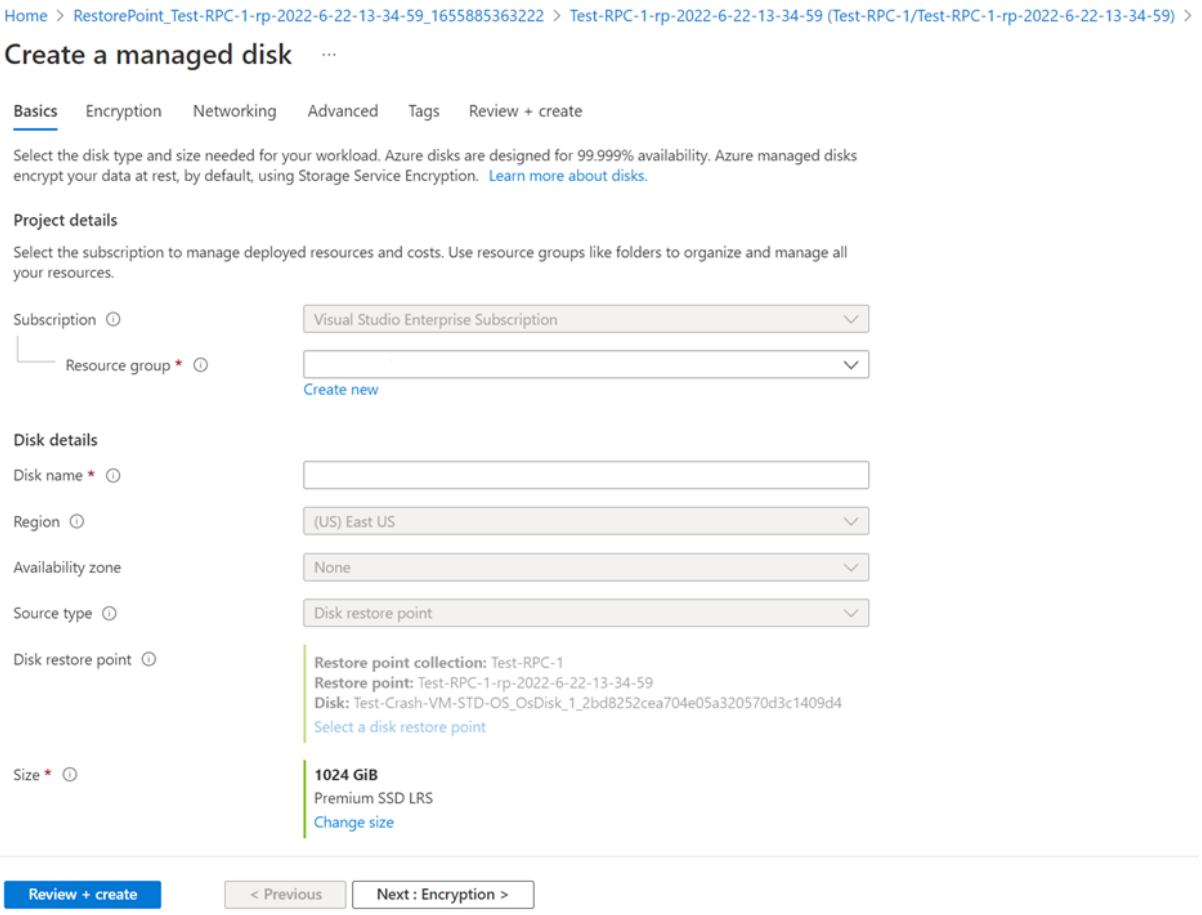
Restore a VM from a restore point

To restore a VM from a VM restore point, first restore individual disks from each disk restore point. You can also use the ARM template to restore a VM along with all the disks.

1. Select **Create a disk from a restore point** to restore a disk from a disk restore point. Do this for all the disks that you want to restore.



2. Enter the details in the **Create a managed disk** dialog to create disks from the restore points. Once the disks are created, create a new VM and attach these restored disks to the newly created VM.



Quickstart: Create a private container registry using the Azure CLI

- Article
- 10/12/2022
- 16 contributors

Feedback

In this article

1. Create a resource group
2. Create a container registry
3. Log in to registry
4. Push image to registry

Show 4 more

Azure Container Registry is a private registry service for building, storing, and managing container images and related artifacts. In this quickstart, you create an

Azure container registry instance with the Azure CLI. Then, use Docker commands to push a container image into the registry, and finally pull and run the image from your registry.

This quickstart requires that you are running the Azure CLI (version 2.0.55 or later recommended). Run `az --version` to find the version. If you need to install or upgrade, see Install Azure CLI.

You must also have Docker installed locally. Docker provides packages that easily configure Docker on any macOS, Windows, or Linux system.

Because the Azure Cloud Shell doesn't include all required Docker components (the `dockerd` daemon), you can't use the Cloud Shell for this quickstart.

Create a resource group

Create a resource group with the az group create command. An Azure resource group is a logical container into which Azure resources are deployed and managed.

The following example creates a resource group named *myResourceGroup* in the *eastus* location.

Azure CLICopy

```
az group create --name myResourceGroup --location eastus
```

Create a container registry

In this quickstart you create a *Basic* registry, which is a cost-optimized option for developers learning about Azure Container Registry. For details on available service tiers, see Container registry service tiers.

Create an ACR instance using the az acr create command. The registry name must be unique within Azure, and contain 5-50 lowercase alphanumeric characters. In the following example, *mycontainerregistry* is used. Update this to a unique value.

Azure CLICopy

```
az acr create --resource-group myResourceGroup \
  --name mycontainerregistry --sku Basic
```

When the registry is created, the output is similar to the following:

JSONCopy

```
{
  "adminUserEnabled": false,
```

```
  "creationDate": "2019-01-08T22:32:13.175925+00:00",
  "id": "/subscriptions/00000000-0000-0000-0000-
000000000000/resourceGroups/myResourceGroup/providers/Microsoft.ContainerRegistry/
registries/mycontainerregistry",
  "location": "eastus",
  "loginServer": "mycontainerregistry.azurecr.io",
  "name": "mycontainerregistry",
  "provisioningState": "Succeeded",
  "resourceGroup": "myResourceGroup",
  "sku": {
    "name": "Basic",
    "tier": "Basic"
  },
  "status": null,
  "storageAccount": null,
  "tags": {},
  "type": "Microsoft.ContainerRegistry/registries"
}
```


Take note of `loginServer` in the output, which is the fully qualified registry name (all lowercase). Throughout the rest of this quickstart `<registry-name>` is a placeholder for the container registry name, and `<login-server>` is a placeholder for the registry's login server name.

Tip

In this quickstart, you create a *Basic* registry, which is a cost-optimized option for developers learning about Azure Container Registry. Choose other tiers for increased storage and image throughput, and capabilities such as connection using a **private endpoint**. For details on available service tiers (SKUs), see **Container registry service tiers**.

Log in to registry

Before pushing and pulling container images, you must log in to the registry. To do so, use the az acr login command. Specify only the registry resource name when logging in with the Azure CLI. Don't use the fully qualified login server name.

Azure CLICopy

```
az acr login --name <registry-name>
```

Example:

Azure CLICopy

```
az acr login --name mycontainerregistry
```

The command returns a `Login Succeeded` message once completed.

Push image to registry

To push an image to an Azure Container registry, you must first have an image. If you don't yet have any local container images, run the following docker pull command to pull an existing public image. For this example, pull the `hello-world` image from Microsoft Container Registry.

Copy
```
docker pull mcr.microsoft.com/hello-world
```

Before you can push an image to your registry, you must tag it with the fully qualified name of your registry login server. The login server name is in the format *<registry-name>.azurecr.io* (must be all lowercase), for example, *mycontainerregistry.azurecr.io*.

Tag the image using the docker tag command. Replace `<login-server>` with the login server name of your ACR instance.

Copy
```
docker tag mcr.microsoft.com/hello-world <login-server>/hello-world:v1
```

Example:

Copy
```
docker tag mcr.microsoft.com/hello-world mycontainerregistry.azurecr.io/hello-world:v1
```

Finally, use docker push to push the image to the registry instance. Replace `<login-server>` with the login server name of your registry instance. This example creates the **hello-world** repository, containing the `hello-world:v1` image.

Copy
```
docker push <login-server>/hello-world:v1
```

After pushing the image to your container registry, remove the `hello-world:v1` image from your local Docker environment. (Note that this docker rmi command does not remove the image from the **hello-world** repository in your Azure container registry.)

Copy
```
docker rmi <login-server>/hello-world:v1
```

List container images

The following example lists the repositories in your registry:

Azure CLICopy

```
az acr repository list --name <registry-name> --output table
```

Output:

Copy
```
Result
----------------
hello-world
```

The following example lists the tags on the **hello-world** repository.

Azure CLICopy
```
az acr repository show-tags --name <registry-name> --repository hello-world --
output table
```

Output:

Copy
```
Result
--------
v1
```

Run image from registry

Now, you can pull and run the `hello-world:v1` container image from your container registry by using docker run:

Copy
```
docker run <login-server>/hello-world:v1
```

Example output:

Copy
```
Unable to find image 'mycontainerregistry.azurecr.io/hello-world:v1' locally
v1: Pulling from hello-world
Digest: sha256:662dd8e65ef7ccf13f417962c2f77567d3b132f12c95909de6c85ac3c326a345
Status: Downloaded newer image for mycontainerregistry.azurecr.io/hello-world:v1

Hello from Docker!
This message shows that your installation appears to be working correctly.

[...]
```

Clean up resources

When no longer needed, you can use the az group delete command to remove the resource group, the container registry, and the container images stored there.

Azure CLICopy

```
az group delete --name myResourceGroup
```

Quickstart: Create an Azure container registry using the Azure portal

- Article
- 10/12/2022
- 20 contributors

Feedback

In this article

Azure Container Registry is a private registry service for building, storing, and managing container images and related artifacts. In this quickstart, you create an Azure container registry instance with the Azure portal. Then, use Docker commands to push a container image into the registry, and finally pull and run the image from your registry.

- Azure CLI
- Azure PowerShell

To log in to the registry to work with container images, this quickstart requires that you are running the Azure CLI (version 2.0.55 or later recommended). Run `az --version` to find the version. If you need to install or upgrade, see Install Azure CLI.

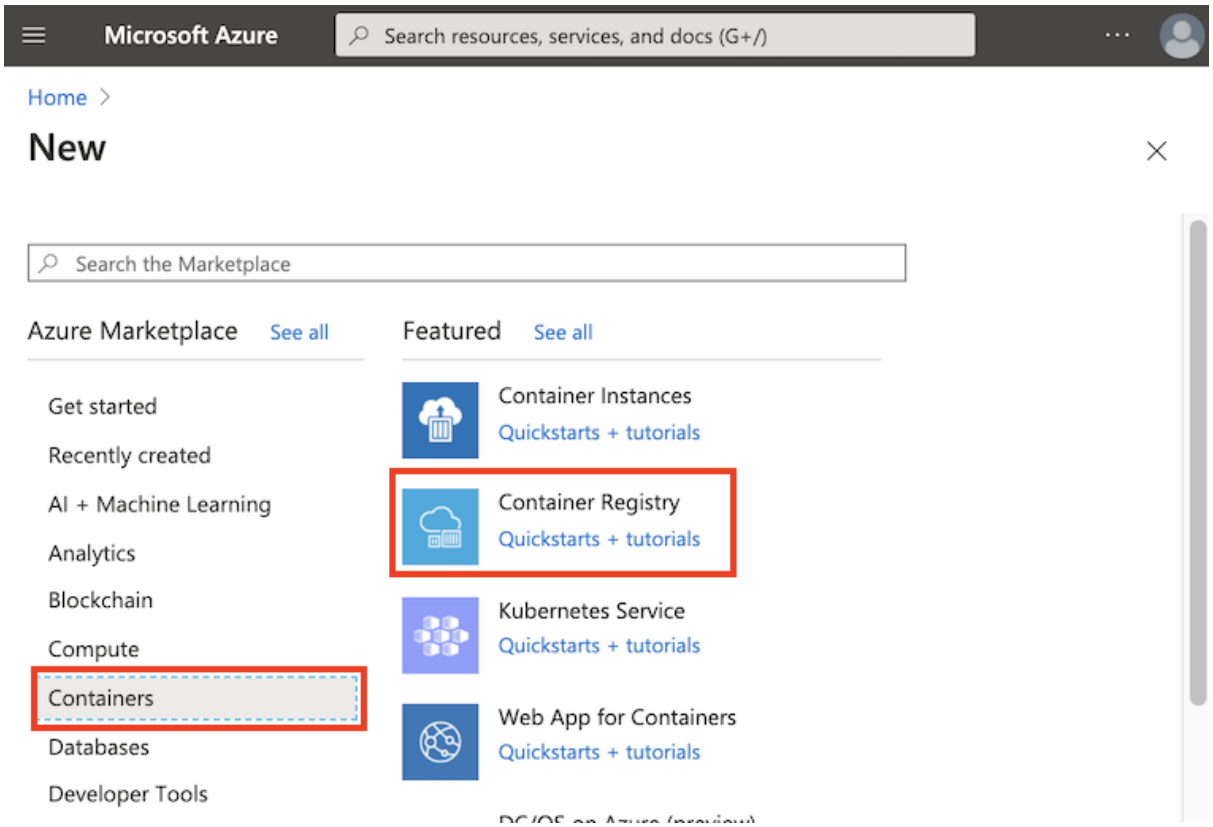
You must also have Docker installed locally with the daemon running. Docker provides packages that easily configure Docker on any Mac, Windows, or Linux system.

Sign in to Azure

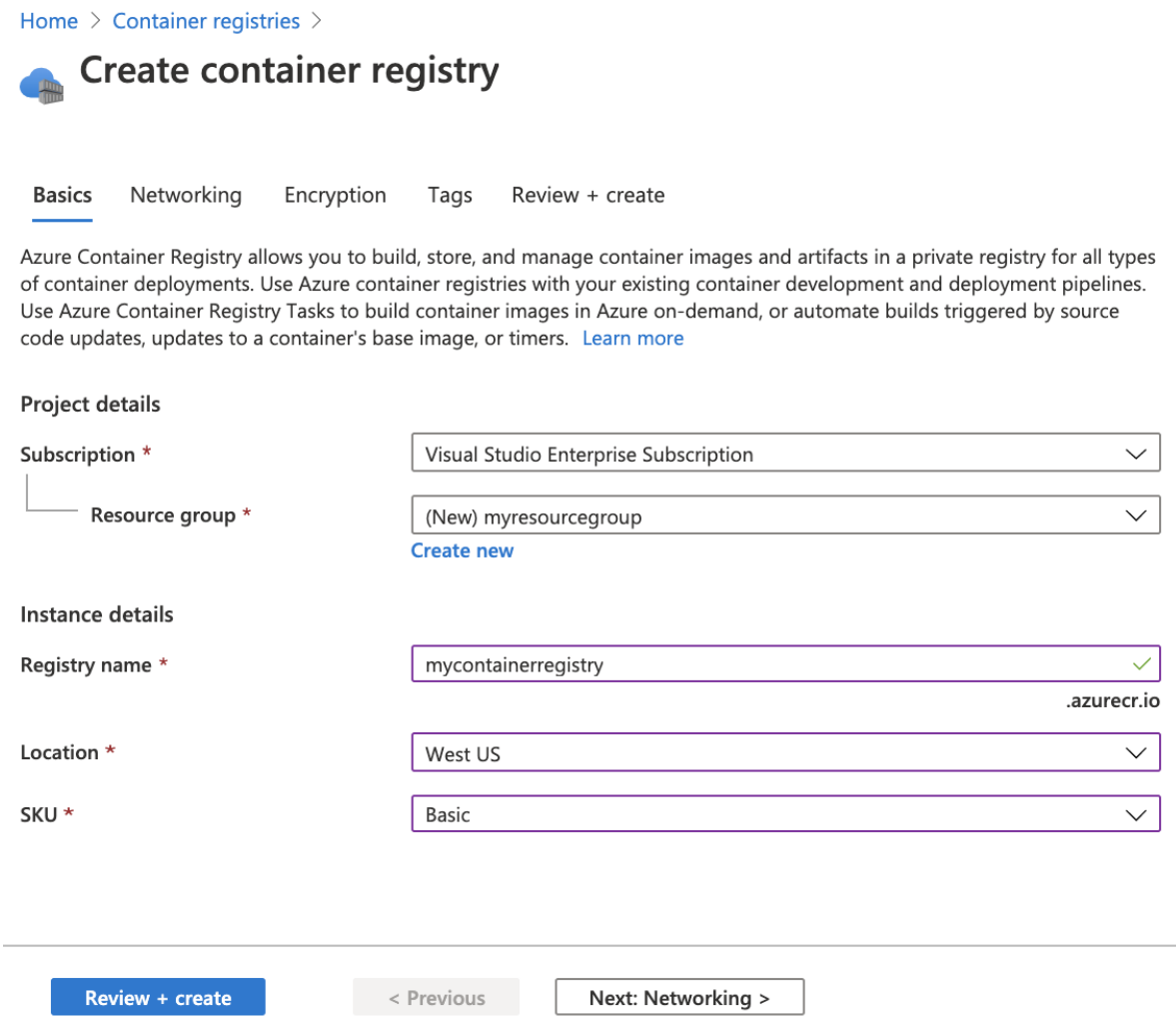
Sign in to the Azure portal at https://portal.azure.com.

Create a container registry

Select **Create a resource** > **Containers** > **Container Registry**.



In the **Basics** tab, enter values for **Resource group** and **Registry name**. The registry name must be unique within Azure, and contain 5-50 alphanumeric characters. For this quickstart create a new resource group in the `West US` location named `myResourceGroup`, and for **SKU**, select 'Basic'.

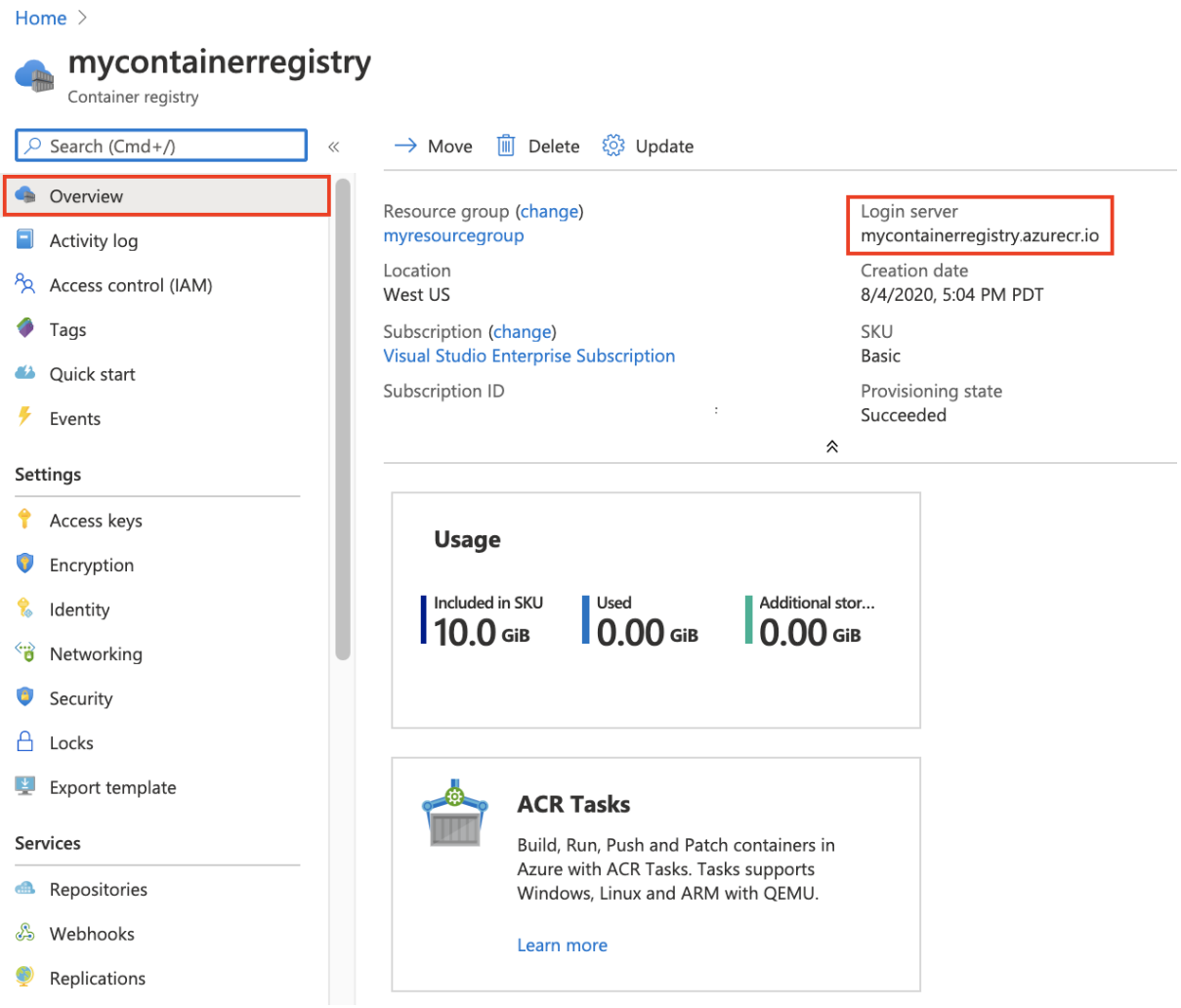


Accept default values for the remaining settings. Then select **Review + create**. After reviewing the settings, select **Create**.

Tip

In this quickstart, you create a *Basic* registry, which is a cost-optimized option for developers learning about Azure Container Registry. Choose other tiers for increased storage and image throughput, and capabilities such as connection using a **private endpoint**. For details on available service tiers (SKUs), see **Container registry service tiers**.

When the **Deployment succeeded** message appears, select the container registry in the portal.



Take note of the registry name and the value of the **Login server**, which is a fully qualified name ending with `azurecr.io` in the Azure cloud. You use these values in the following steps when you push and pull images with Docker.

Log in to registry

- Azure CLI
- Azure PowerShell

Before pushing and pulling container images, you must log in to the registry instance. Sign into the Azure CLI on your local machine, then run the az acr login command. Specify only the registry resource name when logging in with the Azure CLI. Don't use the fully qualified login server name.

Azure CLICopy

```
az acr login --name <registry-name>
```

Example:

Azure CLICopy
```
az acr login --name mycontainerregistry
```

The command returns `Login Succeeded` once completed.

Push image to registry

To push an image to an Azure Container registry, you must first have an image. If you don't yet have any local container images, run the following docker pull command to pull an existing public image. For this example, pull the `hello-world` image from Microsoft Container Registry.

Copy
```
docker pull mcr.microsoft.com/hello-world
```

Before you can push an image to your registry, you must tag it with the fully qualified name of your registry login server. The login server name is in the format *<registry-name>.azurecr.io* (must be all lowercase), for example, *mycontainerregistry.azurecr.io*.

Tag the image using the docker tag command. Replace `<login-server>` with the login server name of your ACR instance.

Copy
```
docker tag mcr.microsoft.com/hello-world <login-server>/hello-world:v1
```

Example:

Copy
```
docker tag mcr.microsoft.com/hello-world mycontainerregistry.azurecr.io/hello-world:v1
```

Finally, use docker push to push the image to the registry instance. Replace `<login-server>` with the login server name of your registry instance. This example creates the **hello-world** repository, containing the `hello-world:v1` image.

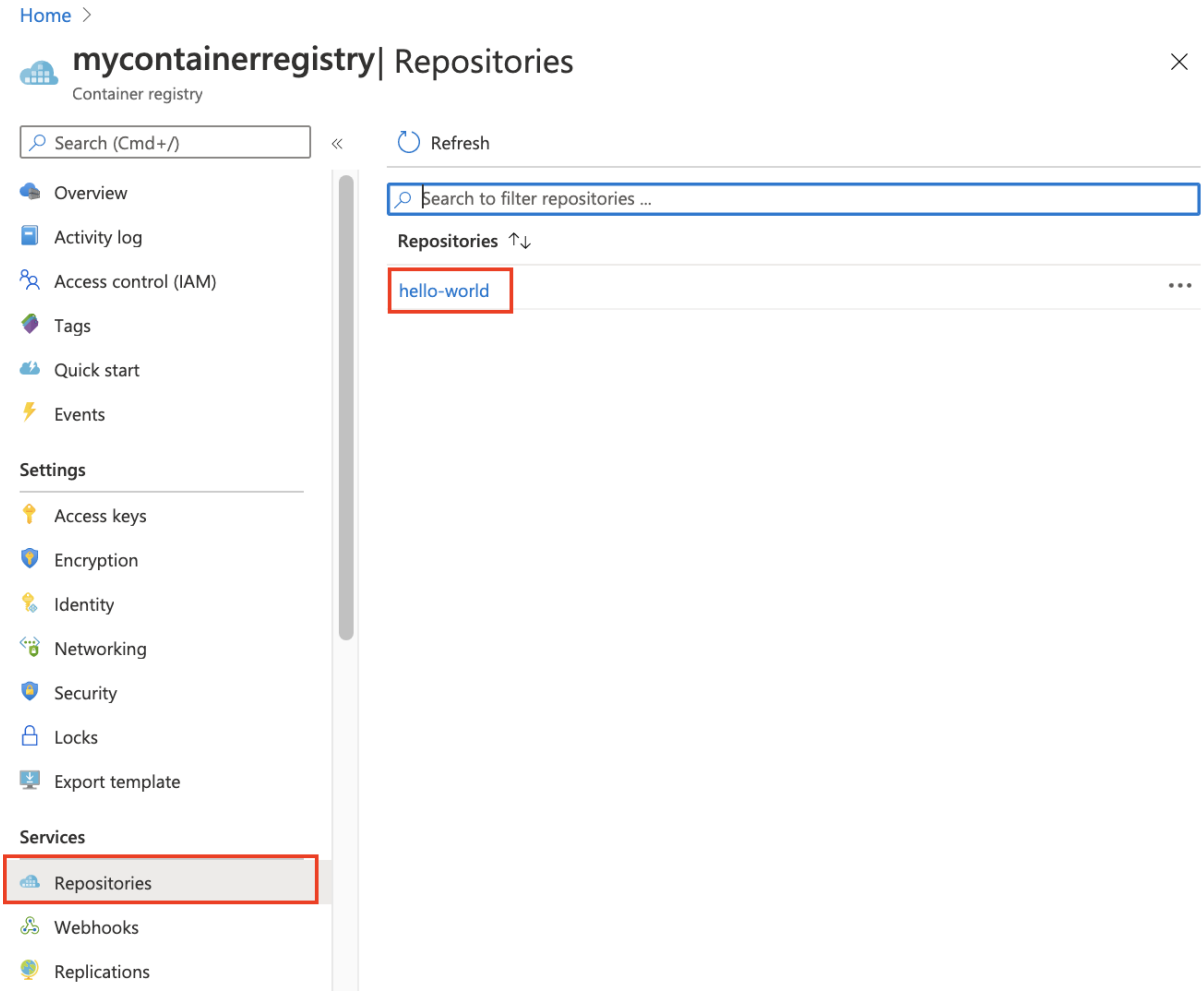
Copy
```
docker push <login-server>/hello-world:v1
```

After pushing the image to your container registry, remove the `hello-world:v1` image from your local Docker environment. (Note that this docker rmi command does not remove the image from the **hello-world** repository in your Azure container registry.)

Copy
```
docker rmi <login-server>/hello-world:v1
```

List container images

To list the images in your registry, navigate to your registry in the portal and select **Repositories**, then select the **hello-world** repository you created with `docker push`.



By selecting the **hello-world** repository, you see the `v1`-tagged image under **Tags**.

Run image from registry

Now, you can pull and run the `hello-world:v1` container image from your container registry by using docker run:

Copy
```
docker run <login-server>/hello-world:v1
```

Example output:

Copy

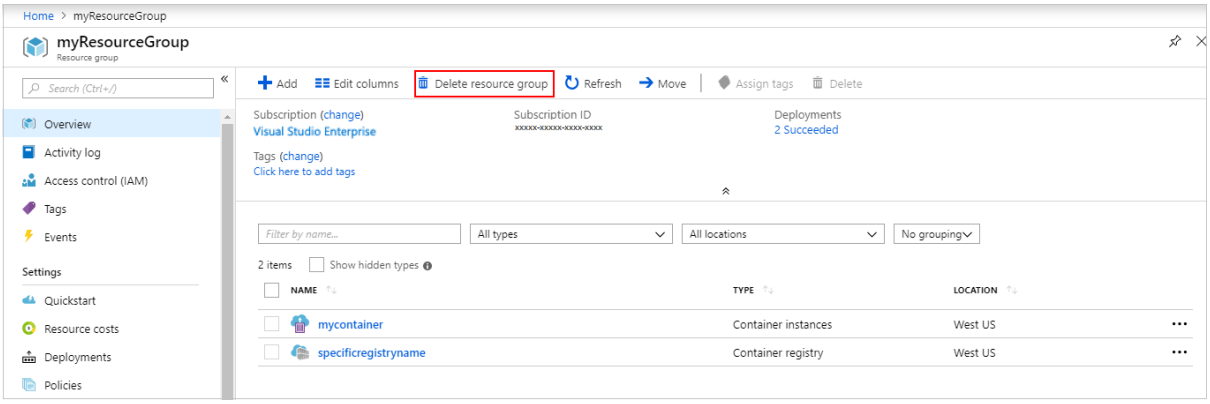
```
Unable to find image 'mycontainerregistry.azurecr.io/hello-world:v1' locally
v1: Pulling from hello-world
Digest: sha256:662dd8e65ef7ccf13f417962c2f77567d3b132f12c95909de6c85ac3c326a345
Status: Downloaded newer image for mycontainerregistry.azurecr.io/hello-world:v1

Hello from Docker!
This message shows that your installation appears to be working correctly.

[...]
```

Clean up resources

To clean up your resources, navigate to the **myResourceGroup** resource group in the portal. Once the resource group is loaded, click on **Delete resource group** to remove the resource group, the container registry, and the container images stored there.



Quickstart: Create a private container registry using Azure PowerShell

- Article
- 05/11/2023
- 11 contributors

Feedback

In this article

1. Prerequisites
2. Sign in to Azure
3. Create resource group
4. Create container registry

Show 5 more

Azure Container Registry is a private registry service for building, storing, and managing container images and related artifacts. In this quickstart, you create an Azure container registry instance with Azure PowerShell. Then, use Docker commands to push a container image into the registry, and finally pull and run the image from your registry.

Prerequisites

Note

We recommend that you use the Azure Az PowerShell module to interact with Azure. See **Install Azure PowerShell** to get started. To learn how to migrate to the Az PowerShell module, see **Migrate Azure PowerShell from AzureRM to Az**.

This quickstart requires Azure PowerShell module. Run `Get-Module -ListAvailable Az` to determine your installed version. If you need to install or upgrade, see Install Azure PowerShell module.

You must also have Docker installed locally. Docker provides packages for macOS, Windows, and Linux systems.

Because the Azure Cloud Shell doesn't include all required Docker components (the `dockerd` daemon), you can't use the Cloud Shell for this quickstart.

Sign in to Azure

Sign in to your Azure subscription with the Connect-AzAccount command, and follow the on-screen directions.

PowerShellCopy

```
Connect-AzAccount
```

Create resource group

Once you're authenticated with Azure, create a resource group with New-AzResourceGroup. A resource group is a logical container in which you deploy and manage your Azure resources.

PowerShellCopy

```
New-AzResourceGroup -Name myResourceGroup -Location EastUS
```

Create container registry

Next, create a container registry in your new resource group with the New-AzContainerRegistry command.

The registry name must be unique within Azure, and contain 5-50 alphanumeric characters. The following example creates a registry named "mycontainerregistry." Replace *mycontainerregistry* in the following command, then run it to create the registry:

PowerShellCopy
```
$registry = New-AzContainerRegistry -ResourceGroupName "myResourceGroup" -Name
"mycontainerregistry" -EnableAdminUser -Sku Basic
```
Tip

In this quickstart, you create a *Basic* registry, which is a cost-optimized option for developers learning about Azure Container Registry. Choose other tiers for increased storage and image throughput, and capabilities such as connection using a **private endpoint**. For details on available service tiers (SKUs), see **Container registry service tiers**.

Log in to registry

Before pushing and pulling container images, you must log in to your registry with the Connect-AzContainerRegistry cmdlet. The following example uses the same credentials you logged in with when authenticating to Azure with the `Connect-AzAccount` cmdlet.

Note

In the following example, the value of `$registry.Name` is the resource name, not the fully qualified registry name.

PowerShellCopy
```
Connect-AzContainerRegistry -Name $registry.Name
```

The command returns `Login Succeeded` once completed.

Push image to registry

To push an image to an Azure Container registry, you must first have an image. If you don't yet have any local container images, run the following docker pull command to

pull an existing public image. For this example, pull the `hello-world` image from Microsoft Container Registry.

Copy
```
docker pull mcr.microsoft.com/hello-world
```

Before you can push an image to your registry, you must tag it with the fully qualified name of your registry login server. The login server name is in the format *<registry-name>.azurecr.io* (must be all lowercase), for example, *mycontainerregistry.azurecr.io*.

Tag the image using the docker tag command. Replace `<login-server>` with the login server name of your ACR instance.

Copy
```
docker tag mcr.microsoft.com/hello-world <login-server>/hello-world:v1
```

Example:

Copy
```
docker tag mcr.microsoft.com/hello-world mycontainerregistry.azurecr.io/hello-world:v1
```

Finally, use docker push to push the image to the registry instance. Replace `<login-server>` with the login server name of your registry instance. This example creates the **hello-world** repository, containing the `hello-world:v1` image.

Copy
```
docker push <login-server>/hello-world:v1
```

After pushing the image to your container registry, remove the `hello-world:v1` image from your local Docker environment. (Note that this docker rmi command does not remove the image from the **hello-world** repository in your Azure container registry.)

Copy
```
docker rmi <login-server>/hello-world:v1
```

Run image from registry

Now, you can pull and run the `hello-world:v1` container image from your container registry by using docker run:

Copy
```
docker run <login-server>/hello-world:v1
```

Example output:

Copy

```
Unable to find image 'mycontainerregistry.azurecr.io/hello-world:v1' locally
v1: Pulling from hello-world
Digest: sha256:662dd8e65ef7ccf13f417962c2f77567d3b132f12c95909de6c85ac3c326a345
Status: Downloaded newer image for mycontainerregistry.azurecr.io/hello-world:v1

Hello from Docker!
This message shows that your installation appears to be working correctly.

[...]
```

Clean up resources

Once you're done working with the resources you created in this quickstart, use the Remove-AzResourceGroup command to remove the resource group, the container registry, and the container images stored there:

PowerShellCopy

```
Remove-AzResourceGroup -Name myResourceGroup
```

Quickstart: Create a geo-replicated container registry by using an ARM template

- Article
- 04/09/2023
- 6 contributors

Feedback

In this article

This quickstart shows how to create an Azure Container Registry instance by using an Azure Resource Manager template (ARM template). The template sets up a geo-replicated registry, which automatically synchronizes registry content across more than one Azure region. Geo-replication enables network-close access to images from

regional deployments, while providing a single management experience. It's a feature of the Premium registry service tier.

A resource manager template is a JavaScript Object Notation (JSON) file that defines the infrastructure and configuration for your project. The template uses declarative syntax. In declarative syntax, you describe your intended deployment without writing the sequence of programming commands to create the deployment.

The registry with replications does not support the ARM/Bicep template Complete mode deployments.

If your environment meets the prerequisites and you're familiar with using ARM templates, select the **Deploy to Azure** button. The template will open in the Azure portal.

Prerequisites

If you don't have an Azure subscription, create a free account before you begin.

Review the template

The template used in this quickstart is from Azure Quickstart Templates. The template sets up a registry and an additional regional replica.

JSONCopy

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-
01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "metadata": {
    "_generator": {
      "name": "bicep",
      "version": "0.5.6.12127",
      "templateHash": "12610175857982700190"
    }
  },
  "parameters": {
    "acrName": {
      "type": "string",
      "defaultValue": "[format('acr{0}', uniqueString(resourceGroup().id))]",
      "maxLength": 50,
      "minLength": 5,
      "metadata": {
        "description": "Globally unique name of your Azure Container Registry"
      }
```

```
    },
    "acrAdminUserEnabled": {
      "type": "bool",
      "defaultValue": false,
      "metadata": {
        "description": "Enable admin user that has push / pull permission to the
registry."
      }
    },
    "location": {
      "type": "string",
      "defaultValue": "[resourceGroup().location]",
      "metadata": {
        "description": "Location for registry home replica."
      }
    },
    "acrSku": {
      "type": "string",
      "defaultValue": "Premium",
      "allowedValues": [
        "Premium"
      ],
      "metadata": {
        "description": "Tier of your Azure Container Registry. Geo-replication
requires Premium SKU."
      }
    },
    "acrReplicaLocation": {
      "type": "string",
      "metadata": {
        "description": "Short name for registry replica location."
      }
    }
  },
  "resources": [
    {
      "type": "Microsoft.ContainerRegistry/registries",
      "apiVersion": "2019-12-01-preview",
      "name": "[parameters('acrName')]",
      "location": "[parameters('location')]",
      "sku": {
        "name": "[parameters('acrSku')]"
      },
      "tags": {
        "displayName": "Container Registry",
        "container.registry": "[parameters('acrName')]"
      },
      "properties": {
        "adminUserEnabled": "[parameters('acrAdminUserEnabled')]"
      }
    },
    {
      "type": "Microsoft.ContainerRegistry/registries/replications",
      "apiVersion": "2019-12-01-preview",
      "name": "[format('{0}/{1}', parameters('acrName'),
parameters('acrReplicaLocation'))]",
      "location": "[parameters('acrReplicaLocation')]",
      "properties": {},
      "dependsOn": [
```

```
        "[resourceId('Microsoft.ContainerRegistry/registries',
parameters('acrName'))]"
      ]
    }
  ],
  "outputs": {
    "acrLoginServer": {
      "type": "string",
      "value": "[reference(resourceId('Microsoft.ContainerRegistry/registries',
parameters('acrName'))).loginServer]"
    }
  }
}
```

The following resources are defined in the template:

- **Microsoft.ContainerRegistry/registries**: create an Azure container registry
- **Microsoft.ContainerRegistry/registries/replications**: create a container registry replica

More Azure Container Registry template samples can be found in the quickstart template gallery.

Deploy the template

1. Select the following image to sign in to Azure and open a template.

2. Select or enter the following values.
    - **Subscription**: select an Azure subscription.
    - **Resource group**: select **Create new**, enter a unique name for the resource group, and then select **OK**.
    - **Region**: select a location for the resource group. Example: **Central US**.
    - **Acr Name**: accept the generated name for the registry, or enter a name. It must be globally unique.
    - **Acr Admin User Enabled**: accept the default value.
    - **Location**: accept the generated location for the registry's home replica, or enter a location such as **Central US**.
    - **Acr Sku**: accept the default value.
    - **Acr Replica Location**: enter a location for the registry replica, using the region's short name. It must be different from the home registry location. Example: **westeurope**.

3. Select **Review + Create**, then review the terms and conditions. If you agree, select **Create**.
4. After the registry has been created successfully, you get a notification:

The Azure portal is used to deploy the template. In addition to the Azure portal, you can use the Azure PowerShell, Azure CLI, and REST API. To learn other deployment methods, see Deploy templates.

Review deployed resources

Use the Azure portal or a tool such as the Azure CLI to review the properties of the container registry.

1. In the portal, search for Container Registries, and select the container registry you created.
2. On the **Overview** page, note the **Login server** of the registry. Use this URI when you use Docker to tag and push images to your registry. For information, see Push your first image using the Docker CLI.

3. On the **Replications** page, confirm the locations of the home replica and the replica added through the template. If desired, add more replicas on this page.

Clean up resources

When you no longer need them, delete the resource group, the registry, and the registry replica. To do so, go to the Azure portal, select the resource group that contains the registry, and then select **Delete resource group**.

Quickstart: Create a container registry by using a Bicep file

- Article
- 04/09/2023
- 3 contributors

Feedback

In this article

This quickstart shows how to create an Azure Container Registry instance by using a Bicep file.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. Bicep offers the best authoring experience for your infrastructure-as-code solutions in Azure.

Prerequisites

If you don't have an Azure subscription, create a free account before you begin.

Review the Bicep file

Use Visual studio code or your favorite editor to create a file with the following content and name it **main.bicep**:

BicepCopy

```
@minLength(5)
@maxLength(50)
@description('Provide a globally unique name of your Azure Container Registry')
param acrName string = 'acr${uniqueString(resourceGroup().id)}'

@description('Provide a location for the registry.')
param location string = resourceGroup().location

@description('Provide a tier of your Azure Container Registry.')
```

```
param acrSku string = 'Basic'

resource acrResource 'Microsoft.ContainerRegistry/registries@2023-01-01-preview' =
{
  name: acrName
  location: location
  sku: {
    name: acrSku
  }
  properties: {
    adminUserEnabled: false
  }
}

@description('Output the login server property for later use')
output loginServer string = acrResource.properties.loginServer
```

The following resource is defined in the Bicep file:

- **Microsoft.ContainerRegistry/registries**: create an Azure container registry

More Azure Container Registry template samples can be found in the quickstart template gallery.

Deploy the Bicep file

To deploy the file you've created, open PowerShell or Azure CLI. If you want to use the integrated Visual Studio Code terminal, select the `ctrl + ` key combination. Change the current directory to where the Bicep file is located.

- CLI
- PowerShell

Azure CLICopy

```
az group create --name myContainerRegRG --location centralus

az deployment group create --resource-group myContainerRegRG --template-file
main.bicep --parameters acrName={your-unique-name}
```

Note

Replace **{your-unique-name}**, including the curly braces, with a unique container registry name.

When the deployment finishes, you should see a message indicating the deployment succeeded.

Review deployed resources

Use the Azure portal or a tool such as the Azure CLI to review the properties of the container registry.

1. In the portal, search for **Container Registries**, and select the container registry you created.
2. On the **Overview** page, note the **Login server** of the registry. Use this URI when you use Docker to tag and push images to your registry. For information, see Push your first image using the Docker CLI.

Clean up resources

When you no longer need the resource, delete the resource group, and the registry. To do so, go to the Azure portal, select the resource group that contains the registry, and then select **Delete resource group**.

Quickstart: Send events from private container registry to Event Grid

- Article
- 03/15/2023
- 12 contributors

Feedback

In this article

Azure Event Grid is a fully managed event routing service that provides uniform event consumption using a publish-subscribe model. In this quickstart, you use the Azure CLI to create a container registry, subscribe to registry events, then deploy a

sample web application to receive the events. Finally, you trigger container image `push` and `delete` events and view the event payload in the sample application.

After you complete the steps in this article, events sent from your container registry to Event Grid appear in the sample web app:

If you don't have an Azure subscription, create an Azure free account before you begin.

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Quickstart for Bash in Azure Cloud Shell.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
  - Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.

- The Azure CLI commands in this article are formatted for the **Bash** shell. If you're using a different shell like PowerShell or Command Prompt, you may need to adjust line continuation characters or variable assignment lines accordingly. This article uses variables to minimize the amount of command editing required.

Create a resource group

An Azure resource group is a logical container in which you deploy and manage your Azure resources. The following az group create command creates a resource group named *myResourceGroup* in the *eastus* region. If you want to use a different name for your resource group, set `RESOURCE_GROUP_NAME` to a different value.

Azure CLICopy
Open Cloudshell

```
RESOURCE_GROUP_NAME=myResourceGroup

az group create --name $RESOURCE_GROUP_NAME --location eastus
```

Create a container registry

Next, deploy a container registry into the resource group with the following commands. Before you run the az acr create command, set `ACR_NAME` to a name for your registry. The name must be unique within Azure, and is restricted to 5-50 alphanumeric characters.

Azure CLICopy
Open Cloudshell

```
ACR_NAME=<acrName>

az acr create --resource-group $RESOURCE_GROUP_NAME --name $ACR_NAME --sku Basic
```

Once the registry has been created, the Azure CLI returns output similar to the following:

JSONCopy

```
{
  "adminUserEnabled": false,
  "creationDate": "2018-08-16T20:02:46.569509+00:00",
  "id": "/subscriptions/<Subscription
ID>/resourceGroups/myResourceGroup/providers/Microsoft.ContainerRegistry/registrie
s/myregistry",
  "location": "eastus",
  "loginServer": "myregistry.azurecr.io",
  "name": "myregistry",
  "provisioningState": "Succeeded",
  "resourceGroup": "myResourceGroup",
  "sku": {
    "name": "Basic",
    "tier": "Basic"
  },
  "status": null,
  "storageAccount": null,
  "tags": {},
  "type": "Microsoft.ContainerRegistry/registries"
}
```

Create an event endpoint

In this section, you use a Resource Manager template located in a GitHub repository to deploy a prebuilt sample web application to Azure App Service. Later, you subscribe to your registry's Event Grid events and specify this app as the endpoint to which the events are sent.

To deploy the sample app, set `SITE_NAME` to a unique name for your web app, and execute the following commands. The site name must be unique within Azure because it forms part of the fully qualified domain name (FQDN) of the web app. In a later section, you navigate to the app's FQDN in a web browser to view your registry's events.

Azure CLICopy
Open Cloudshell

```
SITE_NAME=<your-site-name>

az deployment group create \
    --resource-group $RESOURCE_GROUP_NAME \
    --template-uri "https://raw.githubusercontent.com/Azure-Samples/azure-event-grid-viewer/master/azuredeploy.json" \
    --parameters siteName=$SITE_NAME hostingPlanName=$SITE_NAME-plan
```

Once the deployment has succeeded (it might take a few minutes), open a browser and navigate to your web app to make sure it's running:

```
http://<your-site-name>.azurewebsites.net
```

You should see the sample app rendered with no event messages displayed:

Enable the Event Grid resource provider

If you haven't previously used Event Grid in your Azure subscription, you might need to register the Event Grid resource provider. Run the following command to register the provider:

Azure CLICopy
Open Cloudshell

```
az provider register --namespace Microsoft.EventGrid
```

It might take a moment for the registration to finish. To check the status, run:

Azure CLICopy

Open Cloudshell

```
az provider show --namespace Microsoft.EventGrid --query "registrationState"
```

When `registrationState` is `Registered`, you're ready to continue.

Subscribe to registry events

In Event Grid, you subscribe to a *topic* to tell it which events you want to track, and where to send them. The following az eventgrid event-subscription create command subscribes to the container registry you created, and specifies your web app's URL as the endpoint to which it should send events. The environment variables you populated in earlier sections are reused here, so no edits are required.

Azure CLICopy
Open Cloudshell

```
ACR_REGISTRY_ID=$(az acr show --name $ACR_NAME --query id --output tsv)
APP_ENDPOINT=https://$SITE_NAME.azurewebsites.net/api/updates

az eventgrid event-subscription create \
    --name event-sub-acr \
    --source-resource-id $ACR_REGISTRY_ID \
    --endpoint $APP_ENDPOINT
```

When the subscription is completed, you should see output similar to the following:

JSONCopy

```
{
  "destination": {
    "endpointBaseUrl": "https://eventgridviewer.azurewebsites.net/api/updates",
    "endpointType": "WebHook",
    "endpointUrl": null
  },
  "filter": {
    "includedEventTypes": [
      "All"
    ],
    "isSubjectCaseSensitive": null,
    "subjectBeginsWith": "",
    "subjectEndsWith": ""
  },
  "id": "/subscriptions/<Subscription
ID>/resourceGroups/myResourceGroup/providers/Microsoft.ContainerRegistry/registrie
s/myregistry/providers/Microsoft.EventGrid/eventSubscriptions/event-sub-acr",
  "labels": null,
  "name": "event-sub-acr",
  "provisioningState": "Succeeded",
  "resourceGroup": "myResourceGroup",
  "topic": "/subscriptions/<Subscription
ID>/resourceGroups/myresourcegroup/providers/microsoft.containerregistry/registrie
s/myregistry",
  "type": "Microsoft.EventGrid/eventSubscriptions"
}
```

Trigger registry events

Now that the sample app is up and running and you've subscribed to your registry with Event Grid, you're ready to generate some events. In this section, you use ACR Tasks to build and push a container image to your registry. ACR Tasks is a feature of Azure Container Registry that allows you to build container images in the cloud, without needing the Docker Engine installed on your local machine.

Build and push image

Execute the following Azure CLI command to build a container image from the contents of a GitHub repository. By default, ACR Tasks automatically pushes a successfully built image to your registry, which generates the `ImagePushed` event.

Note

The Dockerfile used in the following example depends on a public base container image from Docker Hub. To improve reliability when using public content, import and manage the image in a private Azure container registry, and update your Dockerfile to use your privately managed base image. **Learn more about working with public images**.

Azure CLICopy
Open Cloudshell

```
az acr build --registry $ACR_NAME --image myimage:v1 -f Dockerfile
https://github.com/Azure-Samples/acr-build-helloworld-node.git#main
```

You should see output similar to the following while ACR Tasks build and then pushes your image. The following sample output has been truncated for brevity.

OutputCopy

```
Sending build context to ACR...
Queued a build with build ID: aa2
Waiting for build agent...
2018/08/16 22:19:38 Using acb_vol_27a2afa6-27dc-4ae4-9e52-6d6c8b7455b2 as the home
volume
2018/08/16 22:19:38 Setting up Docker configuration...
2018/08/16 22:19:39 Successfully set up Docker configuration
2018/08/16 22:19:39 Logging in to registry: myregistry.azurecr.io
2018/08/16 22:19:55 Successfully logged in
Sending build context to Docker daemon  94.72kB
Step 1/5 : FROM node:9-alpine
...
```

To verify that the built image is in your registry, execute the following command to view the tags in the `myimage` repository:

Azure CLICopy
Open Cloudshell

```
az acr repository show-tags --name $ACR_NAME --repository myimage
```

The "v1" tag of the image you built should appear in the output, similar to the following:

OutputCopy

```
[
  "v1"
]
```

Delete the image

Now, generate an `ImageDeleted` event by deleting the image with the az acr repository delete command:

Azure CLICopy
Open Cloudshell

```
az acr repository delete --name $ACR_NAME --image myimage:v1
```

You should see output similar to the following, asking for confirmation to delete the manifest and associated images:

OutputCopy

```
This operation will delete the manifest
'sha256:f15fa9d0a69081ba93eee308b0e475a54fac9c682196721e294b2bc20ab23a1b' and all
the following images: 'myimage:v1'.
Are you sure you want to continue? (y/n):
```

View registry events

You've now pushed an image to your registry and then deleted it. Navigate to your Event Grid Viewer web app, and you should see both `ImageDeleted` and `ImagePushed` events. You might also see a subscription validation event generated by executing the command in the Subscribe to registry events section.

The following screenshot shows the sample app with the three events, and the `ImageDeleted` event is expanded to show its details.

Congratulations! If you see the `ImagePushed` and `ImageDeleted` events, your registry is sending events to Event Grid, and Event Grid is forwarding those events to your web app endpoint.

Clean up resources

Once you're done with the resources you created in this quickstart, you can delete them all with the following Azure CLI command. When you delete a resource group, all of the resources it contains are permanently deleted.

WARNING: This operation is irreversible. Be sure you no longer need any of the resources in the group before running the command.

Azure CLICopy
Open Cloudshell

```
az group delete --name $RESOURCE_GROUP_NAME
```

Event Grid event schema

You can find the Azure Container Registry event message schema reference in the Event Grid documentation:

Azure Event Grid event schema for Container Registry

Quickstart: Use the Azure Container Registry client libraries

- Article
- 06/19/2023
- 3 contributors

Feedback
Choose a programming language
C# Java JavaScript Python Go

In this article

4. Authenticate the client

Show 3 more

Use this article to get started with the client library for Azure Container Registry. Follow these steps to try out example code for data-plane operations on images and artifacts.

Use the client library for Azure Container Registry to:

- List images or artifacts in a registry
- Obtain metadata for images and artifacts, repositories, and tags
- Set read/write/delete properties on registry items
- Delete images and artifacts, repositories, and tags

Azure Container Registry also has a management library for control-plane operations including registry creation and updates.

Prerequisites

- You need an Azure subscription and an Azure container registry to use this library.

  To create a new Azure container registry, you can use the Azure portal, Azure PowerShell, or the Azure CLI. Here's an example using the Azure CLI:

  Azure CLICopy

  ```
  az acr create --name MyContainerRegistry --resource-group
  MyResourceGroup \
      --location westus --sku Basic
  ```

- Push one or more container images to your registry. For steps, see Push your first image to your Azure container registry using the Docker CLI.

Key concepts

- An Azure container registry stores *container images* and OCI artifacts.
- An image or artifact consists of a *manifest* and *layers*.
- A manifest describes the layers that make up the image or artifact. It is uniquely identified by its *digest*.
- An image or artifact can also be *tagged* to give it a human-readable alias. An image or artifact can have zero or more tags associated with it, and each tag uniquely identifies the image.

- A collection of images or artifacts that share the same name, but have different tags, is a *repository*.

For more information, see About registries, repositories, and artifacts.

Get started

Source code | Package (NuGet) | API reference | Samples

To develop .NET application code that can connect to an Azure Container Registry instance, you will need the `Azure.Containers.ContainerRegistry` library.

Install the package

Install the Azure Container Registry client library for .NET with NuGet:

PowershellCopy
```
dotnet add package Azure.Containers.ContainerRegistry --prerelease
```

Authenticate the client

For your application to connect to your registry, you'll need to create a `ContainerRegistryClient` that can authenticate with it. Use the Azure Identity library to add Azure Active Directory support for authenticating Azure SDK clients with their corresponding Azure services.

When you're developing and debugging your application locally, you can use your own user to authenticate with your registry. One way to accomplish this is to authenticate your user with the Azure CLI and run your application from this environment. If your application is using a client that has been constructed to authenticate with `DefaultAzureCredential`, it will correctly authenticate with the registry at the specified endpoint.

C#Copy
```
// Create a ContainerRegistryClient that will authenticate to your registry
through Azure Active Directory
Uri endpoint = new Uri("https://myregistry.azurecr.io");
ContainerRegistryClient client = new ContainerRegistryClient(endpoint, new
DefaultAzureCredential(),
    new ContainerRegistryClientOptions()
    {
        Audience = ContainerRegistryAudience.AzureResourceManagerPublicCloud
    });
```

See the Azure Identity README for more approaches to authenticating with `DefaultAzureCredential`, both locally and in deployment environments. To connect to registries in non-public Azure clouds, see the API reference.

For more information on using Azure AD with Azure Container Registry, see the authentication overview.

Examples

Each sample assumes there is a `REGISTRY_ENDPOINT` environment variable set to a string containing the `https://` prefix and the name of the login server, for example "https://myregistry.azurecr.io".

The following samples use asynchronous APIs that return a task. Synchronous APIs are also available.

List repositories asynchronously

Iterate through the collection of repositories in the registry.

C#Copy
```
// Get the service endpoint from the environment
Uri endpoint = new Uri(Environment.GetEnvironmentVariable("REGISTRY_ENDPOINT"));

// Create a new ContainerRegistryClient
ContainerRegistryClient client = new ContainerRegistryClient(endpoint, new
DefaultAzureCredential(),
    new ContainerRegistryClientOptions()
    {
        Audience = ContainerRegistryAudience.AzureResourceManagerPublicCloud
    });

// Get the collection of repository names from the registry
AsyncPageable<string> repositories = client.GetRepositoryNamesAsync();
await foreach (string repository in repositories)
{
    Console.WriteLine(repository);
}
```

Set artifact properties asynchronously

C#Copy
```
// Get the service endpoint from the environment
Uri endpoint = new Uri(Environment.GetEnvironmentVariable("REGISTRY_ENDPOINT"));

// Create a new ContainerRegistryClient
ContainerRegistryClient client = new ContainerRegistryClient(endpoint, new
DefaultAzureCredential(),
```

```
    new ContainerRegistryClientOptions()
    {
        Audience = ContainerRegistryAudience.AzureResourceManagerPublicCloud
    });

// Get the collection of repository names from the registry
AsyncPageable<string> repositories = client.GetRepositoryNamesAsync();
await foreach (string repository in repositories)
{
    Console.WriteLine(repository);
}
```

Delete images asynchronously

C#Copy

```
using System.Linq;
using Azure.Containers.ContainerRegistry;
using Azure.Identity;

// Get the service endpoint from the environment
Uri endpoint = new Uri(Environment.GetEnvironmentVariable("REGISTRY_ENDPOINT"));

// Create a new ContainerRegistryClient
ContainerRegistryClient client = new ContainerRegistryClient(endpoint, new
DefaultAzureCredential(),
    new ContainerRegistryClientOptions()
    {
        Audience = ContainerRegistryAudience.AzureResourceManagerPublicCloud
    });

// Iterate through repositories
AsyncPageable<string> repositoryNames = client.GetRepositoryNamesAsync();
await foreach (string repositoryName in repositoryNames)
{
    ContainerRepository repository = client.GetRepository(repositoryName);

    // Obtain the images ordered from newest to oldest
    AsyncPageable<ArtifactManifestProperties> imageManifests =
        repository.GetManifestPropertiesCollectionAsync(orderBy:
ArtifactManifestOrderBy.LastUpdatedOnDescending);

    // Delete images older than the first three.
    await foreach (ArtifactManifestProperties imageManifest in
imageManifests.Skip(3))
    {
        RegistryArtifact image = repository.GetArtifact(imageManifest.Digest);
        Console.WriteLine($"Deleting image with digest {imageManifest.Digest}.");
        Console.WriteLine($"   Deleting the following tags from the image: ");
        foreach (var tagName in imageManifest.Tags)
        {
            Console.WriteLine($"
{imageManifest.RepositoryName}:{tagName}");
            await image.DeleteTagAsync(tagName);
        }
        await image.DeleteAsync();
    }
}
```

Clean up resources

If you want to clean up and remove an Azure container registry, you can delete the resource or resource group. Deleting the resource group also deletes any other resources associated with it.

- Portal
- Azure CLI

Quickstart: Build and push container images of the Java Spring Boot app to Azure Container Registry

- Article
- 02/21/2023
- 13 contributors

Feedback

In this article

You can use this Quickstart to build container images of Java Spring Boot app and push it to Azure Container Registry using Maven and Jib. Maven and Jib are one way of using developer tooling to interact with an Azure container registry.

Prerequisites

- An Azure subscription; Sign up for a free Azure account or activate MSDN subscriber benefits if you don't already have an Azure subscription.
- A supported Java Development Kit (JDK); For more information on available JDKs when developing on Azure, see Java support on Azure and Azure Stack.
- The Azure CLI.

- The Apache's Maven build tool (Version 3 or above).
- A Git client.
- A Docker client.
- The ACR Docker credential helper.

Create and build a Spring Boot application on Docker

The following steps walk you through building a containerized Java Spring Boot web application and testing it locally.

1. From the command prompt, use the following command to clone the Spring Boot on Docker Getting Started sample project.

   BashCopy
   ```
   git clone https://github.com/spring-guides/gs-spring-boot-docker.git
   ```

2. Change directory to the complete project.

   BashCopy
   ```
   cd gs-spring-boot-docker/complete
   ```

3. Use Maven to build and run the sample app.

   BashCopy
   ```
   mvn package spring-boot:run
   ```

4. Test the web app by browsing to `http://localhost:8080`, or with the following `curl` command:

   BashCopy
   ```
   curl http://localhost:8080
   ```

You should see the following message displayed: **Hello Docker World**

Create an Azure Container Registry using the Azure CLI

Next, you'll create an Azure resource group and your ACR using the following steps:

1. Log in to your Azure account by using the following command:

   Azure CLICopy

```
az login
```

2. Specify the Azure subscription to use:

   Azure CLICopy
   ```
   az account set -s <subscription ID>
   ```

3. Create a resource group for the Azure resources used in this tutorial. In the following command, be sure to replace the placeholders with your own resource name and a location such as `eastus`.

   Azure CLICopy
   ```
   az group create \
       --name=<your resource group name> \
       --location=<location>
   ```

4. Create a private Azure container registry in the resource group, using the following command. Be sure to replace the placeholders with actual values. The tutorial pushes the sample app as a Docker image to this registry in later steps.

   Azure CLICopy
   ```
   az acr create \
       --resource-group <your resource group name> \
       --location <location> \
       --name <your registry name> \
       --sku Basic
   ```

Push your app to the container registry via Jib

Finally, you'll update your project configuration and use the command prompt to build and deploy your image.

Note

To log in the Azure container registry that you just created, you will need to have the Docker daemon running. To install Docker on your machine, **here is the official Docker documentation**.

1. Log in to your Azure Container Registry from the Azure CLI using the following command. Be sure to replace the placeholder with your own registry name.

   Azure CLICopy
   ```
   az config set defaults.acr=<your registry name>
   az acr login
   ```

The `az config` command sets the default registry name to use with `az acr` commands.

2. Navigate to the completed project directory for your Spring Boot application (for example, "*C:\SpringBoot\gs-spring-boot-docker\complete*" or "*/users/robert/SpringBoot/gs-spring-boot-docker/complete*"), and open the *pom.xml* file with a text editor.
3. Update the `<properties>` collection in the *pom.xml* file with the following XML. Replace the placeholder with your registry name, and add a `<jib-maven-plugin.version>` property with value `2.2.0`, or a newer version of the jib-maven-plugin.

   XMLCopy
   ```
   <properties>
      <docker.image.prefix><your registry
   name>.azurecr.io</docker.image.prefix>
      <java.version>1.8</java.version>
      <jib-maven-plugin.version>2.2.0</jib-maven-plugin.version>
   </properties>
   ```

4. Update the `<plugins>` collection in the *pom.xml* file so that the `<plugin>` element contains and an entry for the `jib-maven-plugin`, as shown in the following example. Note that we are using a base image from the Microsoft Container Registry (MCR): `mcr.microsoft.com/java/jdk:8-zulu-alpine`, which contains an officially supported JDK for Azure. For other MCR base images with officially supported JDKs, see Java SE JDK, Java SE JRE, Java SE Headless JRE, and Java SE JDK and Maven.

   XMLCopy
   ```
   <plugin>
     <artifactId>jib-maven-plugin</artifactId>
     <groupId>com.google.cloud.tools</groupId>
     <version>${jib-maven-plugin.version}</version>
     <configuration>
        <from>
            <image>mcr.microsoft.com/java/jdk:8-zulu-alpine</image>
        </from>
        <to>
            <image>${docker.image.prefix}/${project.artifactId}</image>
        </to>
     </configuration>
   </plugin>
   ```

5. Navigate to the complete project directory for your Spring Boot application and run the following command to build the image and push the image to the registry:

   Azure CLICopy

```
az acr login && mvn compile jib:build
```

Note

For security reasons, the credential created by `az acr login` is valid for 1 hour only. If you receive a *401 Unauthorized* error, you can run the `az acr login -n <your registry name>` command again to reauthenticate.

Verify your container image

Congratulations! Now you have your containerized Java App build on Azure supported JDK pushed to your ACR. You can now test the image by deploying it to Azure App Service, or pulling it to local with command (replacing the placeholder):

BashCopy

```
docker pull <your registry name>.azurecr.io/gs-spring-boot-docker
```

Quickstart: Create a connected registry using the Azure CLI

- Article
- 01/13/2023
- 4 contributors

Feedback

In this article

In this quickstart, you use the Azure CLI to create a connected registry resource in Azure. The connected registry feature of Azure Container Registry allows you to deploy a registry remotely or on your premises and synchronize images and other artifacts with the cloud registry.

Here you create two connected registry resources for a cloud registry: one connected registry allows read and write (artifact pull and push) functionality and one allows read-only functionality.

After creating a connected registry, you can follow other guides to deploy and use it on your on-premises or remote infrastructure.

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Quickstart for Bash in Azure Cloud Shell.

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
  - Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.

- Azure Container registry - If you don't already have a container registry, create one (Premium tier required) in a region that supports connected registries.

Enable the dedicated data endpoint for the cloud registry

Enable the dedicated data endpoint for the Azure container registry in the cloud by using the az acr update command. This step is needed for a connected registry to communicate with the cloud registry.

Azure CLICopy

```
# Set the REGISTRY_NAME environment variable to identify the existing cloud registry
REGISTRY_NAME=<container-registry-name>

az acr update --name $REGISTRY_NAME \
  --data-endpoint-enabled
```

Import images to your cloud registry

Import the following container images to your cloud registry using the az acr import command. Skip this step if you already imported these images.

Connected registry image

To support nested IoT Edge scenarios, the container image for the connected registry runtime must be available in your private Azure container registry. Use the az acr import command to import the connected registry image into your private registry.

Azure CLICopy

```
# Use the REGISTRY_NAME variable in the following Azure CLI commands to identify the registry
REGISTRY_NAME=<container-registry-name>

az acr import \
  --name $REGISTRY_NAME \
  --source mcr.microsoft.com/acr/connected-registry:0.8.0
```

IoT Edge and API proxy images

To support the connected registry on nested IoT Edge, you need to deploy modules for the IoT Edge and API proxy. Import these images into your private registry.

The IoT Edge API proxy module allows an IoT Edge device to expose multiple services using the HTTPS protocol on the same port such as 443.

Azure CLICopy

```
az acr import \
  --name $REGISTRY_NAME \
  --source mcr.microsoft.com/azureiotedge-agent:1.2.4

az acr import \
  --name $REGISTRY_NAME \
  --source mcr.microsoft.com/azureiotedge-hub:1.2.4

az acr import \
  --name $REGISTRY_NAME \
  --source mcr.microsoft.com/azureiotedge-api-proxy:1.1.2

az acr import \
  --name $REGISTRY_NAME \
  --source mcr.microsoft.com/azureiotedge-diagnostics:1.2.4
```

Hello-world image

For testing the connected registry, import the `hello-world` image. This repository will be synchronized to the connected registry and pulled by the connected registry clients.

Azure CLICopy

```
az acr import \
  --name $REGISTRY_NAME \
  --source mcr.microsoft.com/hello-world:1.1.2
```

Create a connected registry resource for read and write functionality

Create a connected registry using the az acr connected-registry create command. The connected registry name must start with a letter and contain only alphanumeric characters. It must be 5 to 40 characters long and unique in the hierarchy for this Azure container registry.

Azure CLICopy

```
# Set the CONNECTED_REGISTRY_RW environment variable to provide a name for the
connected registry with read/write functionality
CONNECTED_REGISTRY_RW=<connnected-registry-name>

az acr connected-registry create --registry $REGISTRY_NAME \
  --name $CONNECTED_REGISTRY_RW \
  --repository "hello-world" "acr/connected-registry" "azureiotedge-agent"
"azureiotedge-hub" "azureiotedge-api-proxy"
```

This command creates a connected registry resource whose name is the value of *$CONNECTED_REGISTRY_RW* and links it to the cloud registry whose name is the value of *$REGISTRY_NAME*. In later quickstart guides, you learn about options to deploy the connected registry.

- The specified repositories will be synchronized between the cloud registry and the connected registry once it is deployed.
- Because no `--mode` option is specified for the connected registry, it is created in the default ReadWrite mode.
- Because there is no synchronization schedule defined for this connected registry, the repositories will be synchronized between the cloud registry and the connected registry without interruptions.

Important

To support nested scenarios where lower layers have no Internet access, you must always allow synchronization of the `acr/connected-registry` repository. This repository contains the image for the connected registry runtime.

Create a connected registry resource for read-only functionality

You can also use the az acr connected-registry create command to create a connected registry with read-only functionality.

Azure CLICopy

```
# Set the CONNECTED_REGISTRY_READ environment variable to provide a name for the
connected registry with read-only functionality
CONNECTED_REGISTRY_RO=<connnected-registry-name>
az acr connected-registry create --registry $REGISTRY_NAME \
  --parent $CONNECTED_REGISTRY_RW \
  --name $CONNECTED_REGISTRY_RO \
  --repository "hello-world" "acr/connected-registry" "azureiotedge-agent"
"azureiotedge-hub" "azureiotedge-api-proxy" \
  --mode ReadOnly
```

This command creates a connected registry resource whose name is the value of *$CONNECTED_REGISTRY_RO* and links it to the cloud registry named with the value of *$REGISTRY_NAME*.

- The specified repositories will be synchronized between the parent registry named with the value of *$CONNECTED_REGISTRY_RW* and the connected registry once deployed.
- This resource is created in the ReadOnly mode, which enables read-only (artifact pull) functionality once deployed.
- Because there is no synchronization schedule defined for this connected registry, the repositories will be synchronized between the parent registry and the connected registry without interruptions.

Verify that the resources are created

You can use the connected registry az acr connected-registry list command to verify that the resources are created.

Azure CLICopy

```
az acr connected-registry list \
  --registry $REGISTRY_NAME \
  --output table
```

You should see a response as follows. Because the connected registries are not yet deployed, the connection state of "Offline" indicates that they are currently disconnected from the cloud.

Copy

```
NAME                 MODE        CONNECTION STATE    PARENT               LOGIN
SERVER    LAST SYNC (UTC)
-------------------  --------    ------------------  -------------------  --------
------  -----------------
myconnectedregrw     ReadWrite   Offline
myconnectedregro     ReadOnly    Offline             myconnectedregrw
```

Quickstart: Create a connected registry using the Azure portal

- Article
- 01/13/2023
- 4 contributors

Feedback

In this article

In this quickstart, you use the Azure portal to create a connected registry resource in Azure. The connected registry feature of Azure Container Registry allows you to deploy a registry remotely or on your premises and synchronize images and other artifacts with the cloud registry.

Here you create two connected registry resources for a cloud registry: one connected registry allows read and write (artifact pull and push) functionality and one allows read-only functionality.

After creating a connected registry, you can follow other guides to deploy and use it on your on-premises or remote infrastructure.

Prerequisites

- Azure Container registry - If you don't already have a container registry, create one (Premium tier required) in a region that supports connected registries.

To import images to the container registry, use the Azure CLI:

- Use the Bash environment in Azure Cloud Shell. For more information, see Quickstart for Bash in Azure Cloud Shell.

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
  - Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.

Enable the dedicated data endpoint for the cloud registry

Enable the dedicated data endpoint for the Azure container registry in the cloud. This step is needed for a connected registry to communicate with the cloud registry.

1. In the Azure portal, navigate to your container registry.
2. Select **Networking > Public access**. Select the **Enable dedicated data endpoint** checkbox.
3. Select **Save**.

Import images to your cloud registry

Import the following container images to your cloud registry using the az acr import command. Skip this step if you already imported these images.

Connected registry image

To support nested IoT Edge scenarios, the container image for the connected registry runtime must be available in your private Azure container registry. Use the az acr import command to import the connected registry image into your private registry.

Azure CLICopy

```
# Use the REGISTRY_NAME variable in the following Azure CLI commands to identify
the registry
REGISTRY_NAME=<container-registry-name>

az acr import \
  --name $REGISTRY_NAME \
  --source mcr.microsoft.com/acr/connected-registry:0.8.0
```

IoT Edge and API proxy images

To support the connected registry on nested IoT Edge, you need to deploy modules for the IoT Edge and API proxy. Import these images into your private registry.

The IoT Edge API proxy module allows an IoT Edge device to expose multiple services using the HTTPS protocol on the same port such as 443.

Azure CLICopy

```
az acr import \
  --name $REGISTRY_NAME \
  --source mcr.microsoft.com/azureiotedge-agent:1.2.4

az acr import \
  --name $REGISTRY_NAME \
  --source mcr.microsoft.com/azureiotedge-hub:1.2.4

az acr import \
  --name $REGISTRY_NAME \
  --source mcr.microsoft.com/azureiotedge-api-proxy:1.1.2

az acr import \
  --name $REGISTRY_NAME \
  --source mcr.microsoft.com/azureiotedge-diagnostics:1.2.4
```

Hello-world image

For testing the connected registry, import the `hello-world` image. This repository will be synchronized to the connected registry and pulled by the connected registry clients.

Azure CLICopy

```
az acr import \
  --name $REGISTRY_NAME \
  --source mcr.microsoft.com/hello-world:1.1.2
```

Create a connected registry resource for read and write functionality

The following steps create a connected registry in ReadWrite mode that is linked to the cloud registry.

1. In the Azure portal, navigate to your container registry.
2. Select **Connected registries (Preview) > + Create**.
3. Enter or select the values in the following table, and select **Save**.

Item	Description
Parent	Select **No parent** for a connected registry linked to the cloud registry.
Mode	Select **ReadWrite**.
Name	The connected registry name must start with a letter and contain only alphanumeric characters. It must be 5 to 40 characters long and unique in the hierarchy for this Azure container registry.
Logging properties	Accept the default settings.
Sync properties	Accept the default settings. Because there is no synchronization schedule defined by default, the repositories will be synchronized between the cloud registry and the connected registry without interruptions.
Repositories	Select or enter the names of the repositories you imported in the previous step. The specified repositories will be synchronized between the cloud registry and the connected registry once it is deployed.

Important

To support nested scenarios where lower layers have no Internet access, you must always allow synchronization of the `acr/connected-registry` repository. This repository contains the image for the connected registry runtime.

Create a connected registry resource for read-only functionality

The following steps create a connected registry in ReadOnly mode whose parent is the connected registry you created in the previous section. This connected registry enables read-only (artifact pull) functionality once deployed.

1. In the Azure portal, navigate to your container registry.
2. Select **Connected registries (Preview) > + Create**.
3. Enter or select the values in the following table, and select **Save**.

Item	Description
Parent	Select the connected registry you created previously.
Mode	Select **ReadOnly**.

Item	Description
Name	The connected registry name must start with a letter and contain only alphanumeric characters. It must be 5 to 40 characters long and unique in the hierarchy for this Azure container registry.
Logging properties	Accept the default settings.
Sync properties	Accept the default settings. Because there is no synchronization schedule defined by default, the repositories will be synchronized between the cloud registry and the connected registry without interruptions.
Repositories	Select or enter the names of the repositories you imported in the previous step. The specified repositories will be synchronized between the parent registry and the connected registry once it is deployed.

View connected registry properties

Select a connected registry in the portal to view its properties, such as its connection status (Offline, Online, or Unhealthy) and whether it has been activated (deployed on-premises). In the following example, the connected registry is not deployed. Its connection state of "Offline" indicates that it is currently disconnected from the cloud.

From this view, you can also generate a connection string and optionally generate passwords for the sync token. A connection string contains configuration settings used for deploying a connected registry and synchronizing content with a parent registry.

Next steps

In this quickstart, you used the Azure portal to create two connected registry resources in Azure. Those new connected registry resources are tied to your cloud registry and allow synchronization of artifacts with the cloud registry.

Continue to the connected registry deployment guides to learn how to deploy and use a connected registry on your IoT Edge infrastructure.

Quickstart: Deploy a connected registry to an IoT Edge device

- Article
- 01/13/2023
- 5 contributors

Feedback

In this article

In this quickstart, you use the Azure CLI to deploy a connected registry as a module on an Azure IoT Edge device. The IoT Edge device can access the parent Azure container registry in the cloud.

For an overview of using a connected registry with IoT Edge, see Using connected registry with Azure IoT Edge. This scenario corresponds to a device at the top layer of an IoT Edge hierarchy.

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Quickstart for Bash in Azure Cloud Shell.

  Launch Cloud Shell

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.

    - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
    - Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.

- Azure IoT Hub and IoT Edge device. For deployment steps, see Quickstart: Deploy your first IoT Edge module to a virtual Linux device.

  Important

  For later access to the modules deployed on the IoT Edge device, make sure that you open the ports 8000, 5671, and 8883 on the device. For configuration steps, see **How to open ports to a virtual machine with the Azure portal**.

- Connected registry resource in Azure. For deployment steps, see quickstarts using the Azure CLI or Azure portal.
    - A connected registry in either `ReadWrite` or `ReadOnly` mode can be used in this scenario.
    - In the commands in this article, the connected registry name is stored in the environment variable *CONNECTED_REGISTRY_RW*.

Import images to your cloud registry

Import the following container images to your cloud registry using the az acr import command. Skip this step if you already imported these images.

Connected registry image

To support nested IoT Edge scenarios, the container image for the connected registry runtime must be available in your private Azure container registry. Use the az acr import command to import the connected registry image into your private registry.

Azure CLICopy

```
# Use the REGISTRY_NAME variable in the following Azure CLI commands to identify the registry
REGISTRY_NAME=<container-registry-name>

az acr import \
  --name $REGISTRY_NAME \
  --source mcr.microsoft.com/acr/connected-registry:0.8.0
```

IoT Edge and API proxy images

To support the connected registry on nested IoT Edge, you need to deploy modules for the IoT Edge and API proxy. Import these images into your private registry.

The IoT Edge API proxy module allows an IoT Edge device to expose multiple services using the HTTPS protocol on the same port such as 443.

Azure CLICopy

```
az acr import \
  --name $REGISTRY_NAME \
  --source mcr.microsoft.com/azureiotedge-agent:1.2.4

az acr import \
  --name $REGISTRY_NAME \
  --source mcr.microsoft.com/azureiotedge-hub:1.2.4

az acr import \
  --name $REGISTRY_NAME \
  --source mcr.microsoft.com/azureiotedge-api-proxy:1.1.2

az acr import \
  --name $REGISTRY_NAME \
  --source mcr.microsoft.com/azureiotedge-diagnostics:1.2.4
```

Hello-world image

For testing the connected registry, import the `hello-world` image. This repository will be synchronized to the connected registry and pulled by the connected registry clients.

Azure CLICopy

```
az acr import \
  --name $REGISTRY_NAME \
  --source mcr.microsoft.com/hello-world:1.1.2
```

Retrieve connected registry configuration

Before deploying the connected registry to the IoT Edge device, you need to retrieve configuration settings from the connected registry resource in Azure.

Use the az acr connected-registry get-settings command to get the settings information required to install a connected registry. The following example specifies HTTPS as the parent protocol. This protocol is required when the parent registry is a cloud registry.

Azure CLICopy

```
az acr connected-registry get-settings \
```

```
  --registry $REGISTRY_NAME \
  --name $CONNECTED_REGISTRY_RW \
  --parent-protocol https
```

By default, the settings information doesn't include the sync token password, which is also needed to deploy the connected registry. Optionally, generate one of the passwords by passing the `--generate-password 1` or `generate-password 2` parameter. Save the generated password to a safe location. It cannot be retrieved again.

Warning

Regenerating a password rotates the sync token credentials. If you configured a device using the previous password, you need to update the configuration.

Command output includes the registry connection string and related settings. The following example output shows the connection string for the connected registry named *myconnectedregistry* with parent registry *contosoregistry*:

JSONCopy

```
{
  "ACR_REGISTRY_CONNECTION_STRING":
"ConnectedRegistryName=myconnectedregistry;SyncTokenName=myconnectedregistry-sync-token;SyncTokenPassword=xxxxxxxxxxxxxxx;ParentGatewayEndpoint=contosoregistry.eastus.data.azurecr.io;ParentEndpointProtocol=https"
}
```

Configure a deployment manifest for IoT Edge

A deployment manifest is a JSON document that describes which modules to deploy to the IoT Edge device. For more information, see Understand how IoT Edge modules can be used, configured, and reused.

To deploy the connected registry and API proxy modules using the Azure CLI, save the following deployment manifest locally as a `manifest.json` file. You will use the file path in the next section when you run the command to apply the configuration to your device.

Connected registry module settings

- Use the token credentials and connection string from the previous sections to update the relevant JSON values in the `env` node.
- The following environment variables are optional in the `env` node:

Variable	Description
ACR_REGISTRY_LOGIN_SERVER	Specifies a unique hostname or FQDN. If used, the connected registry only accepts requests made to this login server value. If no value is provided, then the connected registry can be accessed with any login server value.
ACR_REGISTRY_CERTIFICATE_VOLUME	If your connected registry will be accessible via HTTPS, points to the volume where the HTTPS certificates are stored. If not set, the default location is `/var/acr/certs`.
ACR_REGISTRY_DATA_VOLUME	Overwrites the default location `/var/acr/data` where the images will be stored by the connected registry. This location must match the volume bind for the container.

- **Important**
- If the connected registry listens on a port different from 80 and 443, the `ACR_REGISTRY_LOGIN_SERVER` value (if specified) must include the port. Example: `192.168.0.100:8080`.
- A `HostPort` binding for the connected registry should be set if the API proxy module isn't used. Example:

  Azure CLICopy

```
 "createOptions":
"{\"HostConfig\":{\"Binds\":[\"/home/azureuser/connected-
registry:/var/acr/data\"],\"PortBindings\":{\"8080/tcp\":[{\"HostPort\
":\"8080\"}]}}}"
```

API proxy module settings

- The API proxy will listen on port 8000 configured as `NGINX_DEFAULT_PORT`. For more information about the API proxy settings, see the [IoT Edge GitHub repo](#).

JSONCopy

```
{
    "modulesContent": {
        "$edgeAgent": {
            "properties.desired": {
                "modules": {
                    "connected-registry": {
                        "settings": {
                            "image":
"<REPLACE_WITH_CLOUD_REGISTRY_NAME>.azurecr.io/acr/connected-registry:0.8.0",
                            "createOptions":
"{\"HostConfig\":{\"Binds\":[\"/home/azureuser/connected-
registry:/var/acr/data\"]}}"
                        },
                        "type": "docker",
```

```
                        "env": {
                            "ACR_REGISTRY_CONNECTION_STRING": {
                                "value":
"ConnectedRegistryName=<REPLACE_WITH_CONNECTED_REGISTRY_NAME>;SyncTokenName=<REPLA
CE_WITH_SYNC_TOKEN_NAME>;SyncTokenPassword=REPLACE_WITH_SYNC_TOKEN_PASSWORD;Parent
GatewayEndpoint=<REPLACE_WITH_CLOUD_REGISTRY_NAME>.<REPLACE_WITH_CLOUD_REGISTRY_RE
GION>.data.azurecr.io;ParentEndpointProtocol=https"
                            }
                        },
                        "status": "running",
                        "restartPolicy": "always",
                        "version": "1.0"
                    },
                    "IoTEdgeAPIProxy": {
                        "settings": {
                            "image":
"<REPLACE_WITH_CLOUD_REGISTRY_NAME>.azurecr.io/azureiotedge-api-proxy:1.1.2",
                            "createOptions":
"{\"HostConfig\":{\"PortBindings\":{\"8000/tcp\":[{\"HostPort\":\"8000\"}]}}}"
                        },
                        "type": "docker",
                        "env": {
                            "NGINX_DEFAULT_PORT": {
                                "value": "8000"
                            },
                            "CONNECTED_ACR_ROUTE_ADDRESS": {
                                "value": "connected-registry:8080"
                            },
                            "BLOB_UPLOAD_ROUTE_ADDRESS": {
                                "value": "AzureBlobStorageonIoTEdge:11002"
                            }
                        },
                        "status": "running",
                        "restartPolicy": "always",
                        "version": "1.0"
                    }
                },
                "runtime": {
                    "settings": {
                        "minDockerVersion": "v1.25",
                        "registryCredentials": {
                            "cloudregistry": {
                                "address":
"<REPLACE_WITH_CLOUD_REGISTRY_NAME>.azurecr.io",
                                "password": "<REPLACE_WITH_SYNC_TOKEN_PASSWORD>",
                                "username": "<REPLACE_WITH_SYNC_TOKEN_NAME>"
                            }
                        }
                    },
                    "type": "docker"
                },
                "schemaVersion": "1.1",
                "systemModules": {
                    "edgeAgent": {
                        "settings": {
                            "image":
"<REPLACE_WITH_CLOUD_REGISTRY_NAME>.azurecr.io/azureiotedge-agent:1.2.4",
                            "createOptions": ""
                        },
```

```
                        "type": "docker",
                        "env": {
                            "SendRuntimeQualityTelemetry": {
                                "value": "false"
                            }
                        }
                    },
                    "edgeHub": {
                        "settings": {
                            "image":
"<REPLACE_WITH_CLOUD_REGISTRY_NAME>.azurecr.io/azureiotedge-hub:1.2.4",
                            "createOptions":
"{\"HostConfig\":{\"PortBindings\":{\"443/tcp\":[{\"HostPort\":\"443\"}],\"5671/tc
p\":[{\"HostPort\":\"5671\"}],\"8883/tcp\":[{\"HostPort\":\"8883\"}]}}}"
                        },
                        "type": "docker",
                        "status": "running",
                        "restartPolicy": "always"
                    }
                }
            }
        },
        "$edgeHub": {
            "properties.desired": {
                "routes": {
                    "route": "FROM /messages/* INTO $upstream"
                },
                "schemaVersion": "1.1",
                "storeAndForwardConfiguration": {
                    "timeToLiveSecs": 7200
                }
            }
        }
    }
}
```

Deploy the connected registry and API proxy modules on IoT Edge

Use the following command to deploy the connected registry and API proxy modules on the IoT Edge device, using the deployment manifest created in the previous section. Provide the ID of the IoT Edge top layer device and the name of the IoT Hub where indicated.

Azure CLICopy

```
# Set the IOT_EDGE_TOP_LAYER_DEVICE_ID and IOT_HUB_NAME environment variables for
use in the following Azure CLI command
IOT_EDGE_TOP_LAYER_DEVICE_ID=<device-id>
IOT_HUB_NAME=<hub-name>

az iot edge set-modules \
  --device-id $IOT_EDGE_TOP_LAYER_DEVICE_ID \
  --hub-name $IOT_HUB_NAME \
  --content manifest.json
```

For details, see Deploy Azure IoT Edge modules with Azure CLI.

To check the status of the connected registry, use the following az acr connected-registry show command. The name of the connected registry is the value of *$CONNECTED_REGISTRY_RW*.

Azure CLICopy

```
az acr connected-registry show \
  --registry $REGISTRY_NAME \
  --name $CONNECTED_REGISTRY_RW \
  --output table
```

After successful deployment, the connected registry shows a status of `Online`.

Create an App Service Environment

- Article
- 03/09/2023
- 7 contributors

Feedback

In this article

App Service Environment is a single-tenant deployment of Azure App Service. You use it with an Azure virtual network. You need one subnet for a deployment of App Service Environment, and this subnet can't be used for anything else.

Note

This article is about App Service Environment v3, which is used with isolated v2 App Service plans.

Before you create your App Service Environment

After you create your App Service Environment, you can't change any of the following:

- Location

- Subscription
- Resource group
- Azure virtual network
- Subnets
- Subnet size
- Name of your App Service Environment

Make your subnet large enough to hold the maximum size that you'll scale your App Service Environment. The recommended size is a /24 with 256 addresses.

Deployment considerations

Before you deploy your App Service Environment, think about the virtual IP (VIP) type and the deployment type.

With an *internal VIP*, an address in your App Service Environment subnet reaches your apps. Your apps aren't on a public DNS. When you create your App Service Environment in the Azure portal, you have an option to create an Azure private DNS zone for your App Service Environment. With an *external VIP*, your apps are on an address facing the public internet, and they're in a public DNS. For both *internal VIP* and *external VIP* you can specify *Inbound IP address*, you can select *Automatic* or *Manual* options. If you want to use the *Manual* option for an *external VIP*, you must first create a standard *Public IP address* in Azure.

For the deployment type, you can choose *single zone*, *zone redundant*, or *host group*. The single zone is available in all regions where App Service Environment v3 is available. With the single zone deployment type, you have a minimum charge in your App Service plan of one instance of Windows Isolated v2. As soon as you use one or more instances, then that charge goes away. It isn't an additive charge.

In a zone redundant App Service Environment, your apps spread across three zones in the same region. Zone redundant is available in regions that support availability zones. With this deployment type, the smallest size for your App Service plan is three instances. That ensures that there's an instance in each availability zone. App Service plans can be scaled up one or more instances at a time. Scaling doesn't need to be in units of three, but the app is only balanced across all availability zones when the total instances are multiples of three.

A zone redundant deployment has triple the infrastructure, and ensures that even if two of the three zones go down, your workloads remain available. Due to the increased system need, the minimum charge for a zone redundant App Service Environment is nine instances. If you've fewer than this number of instances, the difference is charged as Windows I1v2. If you've nine or more instances, there's no

added charge to have a zone redundant App Service Environment. To learn more about zone redundancy, see Regions and availability zones.

In a host group deployment, your apps are deployed onto a dedicated host group. The dedicated host group isn't zone redundant. With this type of deployment, you can install and use your App Service Environment on dedicated hardware. There's no minimum instance charge for using App Service Environment on a dedicated host group, but you do have to pay for the host group when you're provisioning the App Service Environment. You also pay a discounted App Service plan rate as you create your plans and scale out.

With a dedicated host group deployment, there are a finite number of cores available that are used by both the App Service plans and the infrastructure roles. This type of deployment can't reach the 200 total instance count normally available in App Service Environment. The number of total instances possible is related to the total number of App Service plan instances, plus the load-based number of infrastructure roles.

Create an App Service Environment in the portal

Here's how:

1. Search Azure Marketplace for *App Service Environment v3*.
2. From the **Basics** tab, for **Subscription**, select the subscription.
   For **Resource Group**, select or create the resource group, and enter the name of your App Service Environment. For **Virtual IP**, select **Internal** if you want your inbound address to be an address in your subnet.
   Select **External** if you want your inbound address to face the public internet. For **App Service Environment Name**, enter a name. The name must be no more than 36 characters. The name you choose will also be used for the domain suffix. For example, if the name you choose is *contoso*, and you have an internal VIP, the domain suffix will be `contoso.appserviceenvironment.net`. If the name you choose is *contoso*, and you have an external VIP, the domain suffix will be `contoso.p.azurewebsites.net`.

3. From the **Hosting** tab, for **Physical hardware isolation**, select **Enabled** or **Disabled**. If you enable this option, you can deploy onto dedicated hardware. With a dedicated host deployment, you're charged for two dedicated hosts per our pricing when you create the

App Service Environment v3 and then, as you scale, you're charged a specialized Isolated v2 rate per vCore. I1v2 uses two vCores, I2v2 uses four vCores, and I3v2 uses eight vCores per instance. For **Zone redundancy**, select **Enabled** or **Disabled**.

4. From the **Networking** tab, for **Virtual Network**, select or create your virtual network. For **Subnet**, select or create your subnet. If you're creating an App Service Environment with an internal VIP, you can configure Azure DNS private zones to point your domain suffix to your App Service Environment. For more information, see the DNS section in Use an App Service Environment. If you're creating an App Service Environment with an internal VIP, you can specify private IP address using **Manual** option for **Inbound IP address**.

If you're creating an App Service Environment with an external VIP, you can select public IP address using **Manual** option for **Inbound IP address**.

5. From the **Review + create** tab, check that your configuration is correct, and select **Create**. Your App Service Environment can take more than one hour to create.

When your App Service Environment has been successfully created, you can select it as a location when you're creating your apps.