

12. Connecting an application to an Azure database

In previous chapters, you stored the state of your application in your cluster, either on a Redis cluster or on MariaDB. You might remember that both had some issues when it came to high availability. This chapter will take you through the process of connecting to a MySQL database managed by Azure.

We will discuss the benefits of using a hosted database rather than running **StatefulSets** on Kubernetes. To create this hosted and managed database, you will make use of **Azure Service Operator (ASO)**. ASO is a way to create Azure resources, such as a managed MySQL database, from within a Kubernetes cluster. In this chapter, you will learn more details about the ASO

project, and you will set up and configure ASO on your cluster.

You will then make use of ASO to create a MySQL database in Azure. You will use this managed database as part of a WordPress application. This will show you how you can connect an application to a managed database. This chapter is broken down into the following topics:

- Azure Service Operator
- Installing ASO on your cluster
- Creating a MySQL database using ASO
- Creating an application using the MySQL database

Let's start by exploring ASO.

Azure Service Operator

In this section, you will learn more about ASO. We will start by exploring the benefits of using a hosted database versus running StatefulSets on Kubernetes itself, and then learn more details about ASO.

All the examples that you have gone through so far have been self-contained; that is, everything ran inside the Kubernetes cluster. Almost any production application has a state, which is generally stored in a database. While there is a great advantage to being mostly cloud-agnostic, this has a huge disadvantage when it comes to managing a stateful workload such as a database.

When you are running your own database on top of a Kubernetes cluster, you need to take care of scalability, security, high availability, DR, and backup. Managed database services offered by cloud providers can offload you or your team from having to execute these tasks. For example, Azure Database for MySQL comes with enterprise-grade security and compliance, built-in high availability, and automated backups. The service scales within seconds. Finally, you also have the option to configure DR to a secondary region.

It is a lot simpler to consume a production-grade database from Azure than it is to set up and

manage your own on Kubernetes. In the next section, you will explore a way that Kubernetes can be used to create these databases on Azure.

What is ASO?

As with most applications these days, much of the hard work has already been done for us by the open-source community (including those who work for Microsoft). Microsoft has realized that many users would like to use their managed services from Kubernetes and that they require an easier way of using the same methodologies that are used for Kubernetes deployment. The ASO project was created to solve this problem.

ASO is a new project started in 2020 that succeeds the **Open Service Broker for Azure (OSBA)** project. OSBA was Microsoft's original implementation that allowed you to create Azure resources from within Kubernetes, but this project is no longer maintained and has been deprecated. ASO serves the same purpose and is actively maintained and developed.

There are two parts to ASO: a set of **CustomResourceDefinitions (CRDs)** and a controller that manages those CRDs. The CRDs are a set of API extensions for Kubernetes that allow you to specify which Azure resources you want to create. There are CRDs for resource groups, virtual machines, MySQL databases, and more.

Most APIs in ASO are still in either the alpha or beta stage, meaning they might change in the future. Please refer to the documentation at <https://github.com/Azure/azure-service-operator> for an up-to-date resource definition, as the definitions used in this chapter might have changed.

The controller is a pod that runs on your cluster and monitors the Kubernetes API for any objects that are created using these CRDs. It's this controller that will interface with the Azure API and create the resource you create using ASO.

ASO depends on two other projects that you have already learned about in this book, namely

Azure Active Directory (Azure AD) pod-managed identities and cert-manager. ASO uses Azure AD pod-managed identities to link a managed identity to the ASO pod. This also means that this managed identity needs to have permissions to create those resources. ASO uses cert-manager to get access to a certificate for the ASO pod to use.

By default, ASO will store secrets such as connection strings in Kubernetes secrets. As you have learned in the preceding chapters, it's better to store secrets in Key Vault rather than in Kubernetes. ASO has the option to store secrets in Key Vault as well, and during the setup, you will configure ASO to store secrets in Key Vault.

For a user perspective using ASO, *Figure 12.1* describes what happens when you create a resource:

1. As a user, you submit a YAML definition for an Azure resource to the Kubernetes API. The Azure resources are defined in a CRD.

2. The ASO pod is monitoring the Kubernetes API for changes to the Azure CRD objects.
3. When changes are detected, ASO will create the resources in Azure.
4. If a connection string was created as part of the resource creation, this connection string will be stored either as a Kubernetes secret (default) or in Key Vault (if configured).

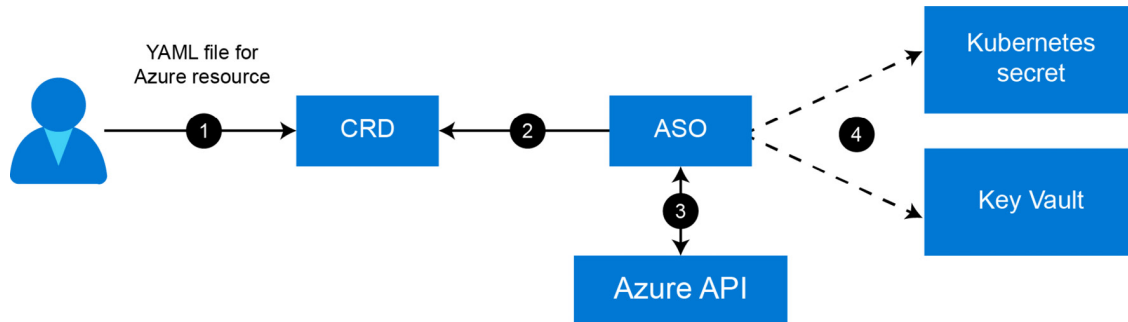


Figure 12.1: High-level process diagram of resource creation using ASO

In this section, you've learned the basics of the ASO project. In the next section, you will go ahead and install ASO on your cluster.

Installing ASO on your cluster

To install ASO on your cluster, you will need a cluster. At the end of the previous chapter, you

deleted your cluster, so you will create a new one here. After that, you will need to create a managed identity and Key Vault. Both are best practices when setting up ASO, which is why this chapter will explain how to set up ASO this way. After the creation of these resources, you need to ensure that cert-manager is set up in your cluster. Once that is confirmed, you can install ASO using a Helm chart.

Let's start with the first step, creating a new AKS cluster.

Creating a new AKS cluster

Since you deleted your cluster at the end of the previous chapter, let's start by creating a new cluster. You can do all these steps using Cloud Shell. Let's get started:

1. First, you will create a new cluster. Since you will be making use of pod identities for the authorization of ASO, you will also enable the pod identity add-on on this new cluster. At the

time of this writing, the pod identity add-on is in preview.

If you haven't registered for your subscription for this preview as explained in *Chapter 9, Azure Active Directory pod-managed identities in AKS*, please do so now using the following commands:

```
az feature register --name  
EnablePodIdentityPreview \  
--namespace Microsoft.ContainerService
```

You will also need a preview extension of the Azure CLI, which you can install using the following command:

```
az extension add --name aks-preview
```

You will have to wait until the pod identity preview is registered on your subscription. You can use the following command to verify this status:

```
az feature show --name  
EnablePodIdentityPreview \  
--namespace Microsoft.ContainerService -o  
table
```

Wait until the status shows as registered, as shown in *Figure 12.2*:

```

user@Azure:~/Hands-On-Kubernetes-on-Azure$ az feature show --name EnablePodIdentityPreview \
> --namespace Microsoft.ContainerService -o table
Name                                RegistrationState
-----
Microsoft.ContainerService/EnablePodIdentityPreview  Registering
user@Azure:~/Hands-On-Kubernetes-on-Azure$ az feature show --name EnablePodIdentityPreview \
> --namespace Microsoft.ContainerService -o table
Name                                RegistrationState
-----
Microsoft.ContainerService/EnablePodIdentityPreview  Registered

```

Figure 12.2: Waiting for the feature to be registered

Once the feature is registered, you need to refresh the registration of the namespace before creating a new cluster. Let's first refresh the registration of the namespace:

```

az provider register --namespace
Microsoft.ContainerService

```

2. Once you registered the preview provider, or if you had already done so as part of *Chapter 9, Azure Active Directory pod-managed identities in AKS*, you can create a new cluster using the following command:

```

az aks create -g rg-handsonaks -n handsonaks \
--enable-managed-identity --enable-pod-identity \

```

```
--network-plugin azure --node-vm-size  
Standard_DS2_v2 \  
--node-count 2 --generate-ssh-keys
```

3. Once the command is finished, get the credentials to get access to your cluster using the following command:

```
az aks get-credentials -g rg-handsonaks \  
-n handsonaks --overwrite-existing
```

You now have a new Kubernetes cluster with pod identities enabled. To continue the setup of ASO, let's now create a managed identity.

Creating a managed identity

In this section, you will use the Azure portal to create a managed identity. You will then give permission to your AKS cluster to manage this managed identity and give the managed identity access to your subscription to create the resources. Let's start:

1. In the Azure search bar, look for *Managed Identities*, as shown in *Figure 12.3*:

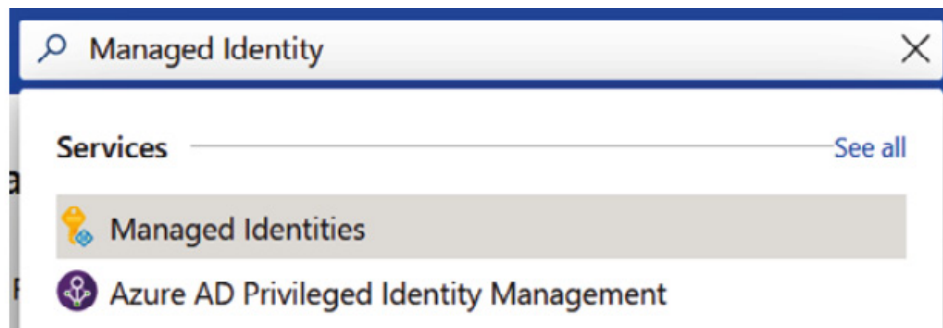


Figure 12.3: Searching for Managed Identities

2. In the resulting screen, click on + New to create a new managed identity, as shown in

Figure 12.4:

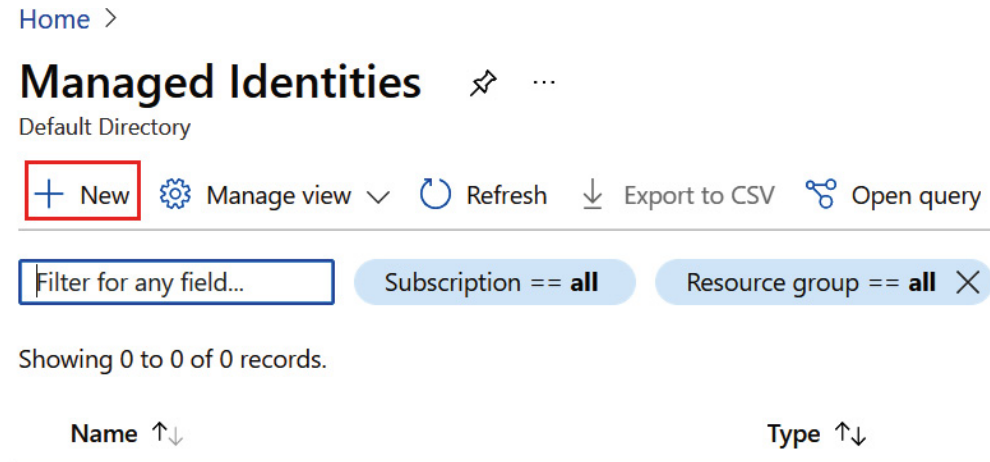


Figure 12.4: Creating a new managed identity

3. To organize the resources for this chapter together, create a new resource group called ASO, as shown in *Figure 12.5:*

Subscription * ⓘ Azure subscription 1

Resource group * ⓘ rg-handsonaks

Create new

Instance details

Region * ⓘ

Name * ⓘ

A resource group is a container that holds related resources for an Azure solution.

Name *

ASO

OK Cancel

Figure 12.5: Creating a new resource group

4. Provide the location and a name for your managed identity; use the name **aso-mi** as shown in *Figure 12.6* if you wish to follow the example here. Make sure to select the same region as the region of your cluster:

Create User Assigned Managed Identity

Basics Tags Review + create

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ Azure subscription 1

Resource group * ⓘ (New) ASO

Create new

Instance details

Region * ⓘ West US 2

Name * ⓘ aso-mi

Figure 12.6: Providing Project and Instance details for creating the managed identity

5. Click Review + create at the bottom of the screen and create the managed identity.
6. Once the managed identity is created, you need to capture the client ID and resource ID for later use. Copy and paste this information in a location where you can access it later. You can get the client ID in the Overview pane, as shown in *Figure 12.7*:

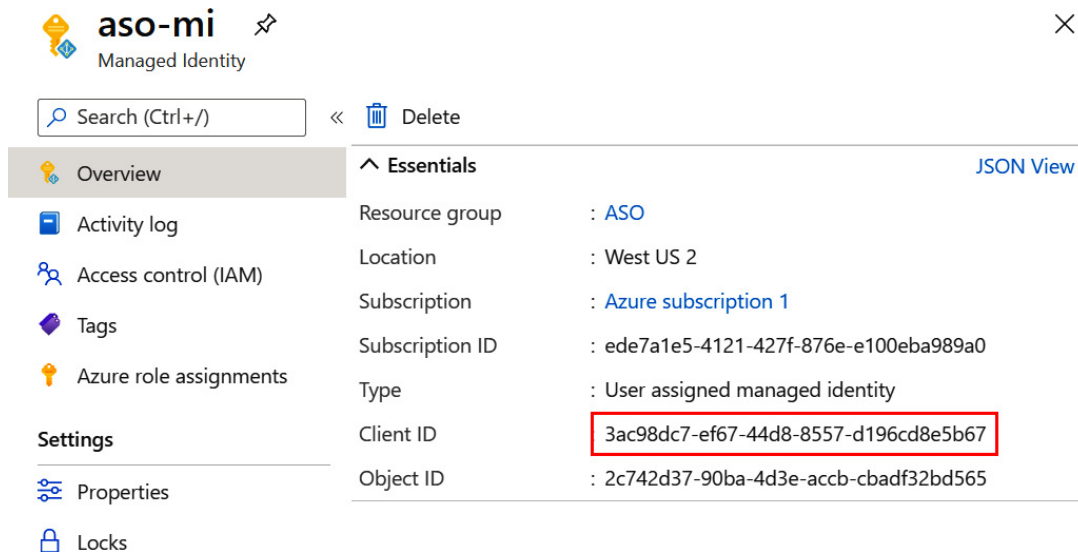


Figure 12.7: Getting the client ID from the managed identity

You can get the resource ID in the Properties pane, as shown in *Figure 12.8*:

The screenshot displays the 'aso-mi' Managed Identity Properties page in the Azure portal. The left-hand navigation pane includes sections for Overview, Activity log, Access control (IAM), Tags, and Azure role assignments. Below these are sections for Settings (Properties, Locks), Monitoring (Advisor recommendations), Automation (Tasks (preview), Export template), and Support + troubleshooting (New support request). The main content area on the right lists various properties for the managed identity, including Name, Resource type, Location, Location ID, Resource ID, Resource group, Resource group ID, Subscription, and Subscription ID. The 'Resource ID' field, which contains the value '/subscriptions/ede7a1e5-4121-427f-876e-e100eba989a...', is highlighted with a red rectangular box.

Property	Value
Name	aso-mi
Resource type	Microsoft.ManagedIdentity/userAssignedIdentities
Location	West US 2
Location ID	westus2
Resource ID	/subscriptions/ede7a1e5-4121-427f-876e-e100eba989a...
Resource group	ASO
Resource group ID	/subscriptions/ede7a1e5-4121-427f-876e-e100eba989a...
Subscription	Azure subscription 1
Subscription ID	/subscriptions/ede7a1e5-4121-427f-876e-e100eba989a0

Figure 12.8: Getting the resource ID of the managed identity

7. The next thing to do on the managed identity is to give our AKS cluster permissions to it. To do this, click on Access control (IAM) in the left pane, click on the + Add button at the top of the screen, click Add role assignment from the dropdown menu, select the Managed Identity Operator role, select User assigned managed

identity from the Assign access to dropdown menu, and select the handsonaks-agentpool identity and save. This process is shown in

Figure 12.9:

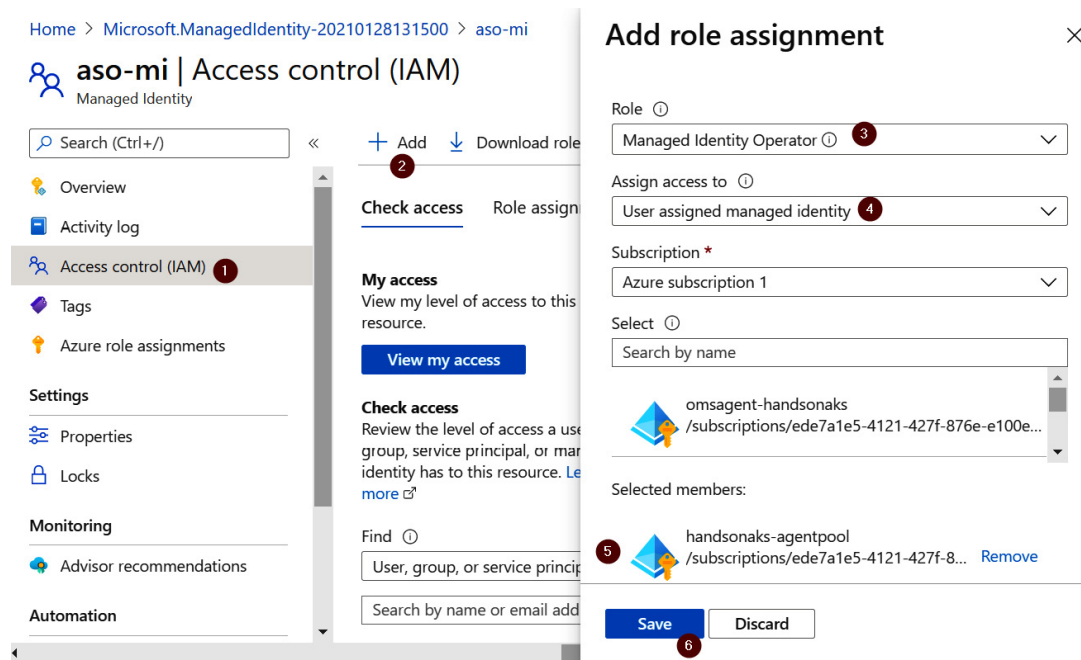


Figure 12.9: Giving AKS access to the managed identity

8. You will now give Managed Identities permission to create resources on your subscription. To do this, look for Subscriptions in the Azure search bar, as shown in *Figure 12.10*, and then select your subscription:

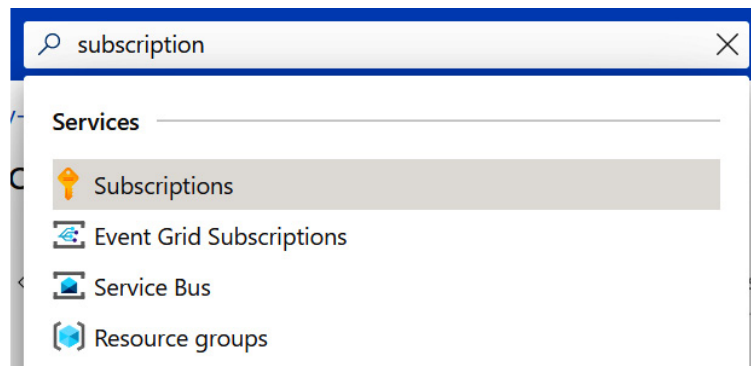


Figure 12.10: Looking for subscriptions in the Azure search bar

9. In the Subscription pane, click on Access control (IAM), click on the + Add button at the top of the screen, click Add role assignment, select the Contributor role, select User assigned managed identity from the Assign access to dropdown menu, and select the aso-mi identity and save. This process is shown in *Figure 12.11*:

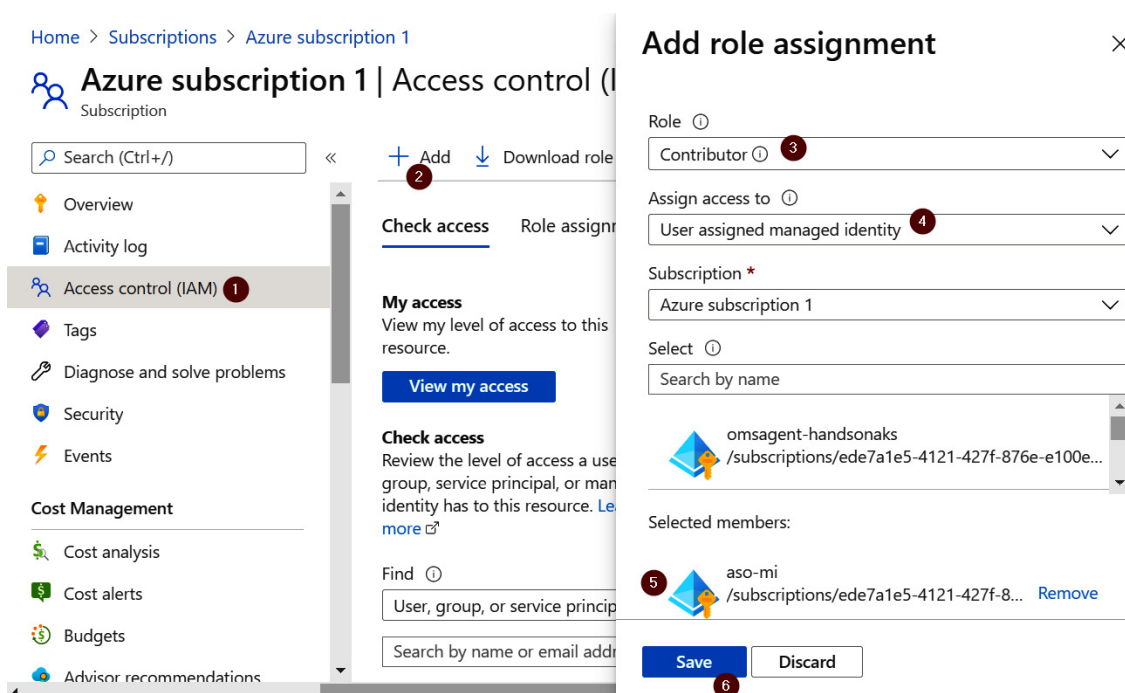


Figure 12.11: Giving the aso-mi permissions to your subscription

This completes the setup of the managed identity. In the next section, you will create a key vault and allow the managed identity you just created to create and read secrets.

Creating a key vault

In this section, you will create the key vault that ASO will use to store connection strings and secrets. This is optional in the ASO setup process but recommended.

1. To start, look for key vaults in the Azure search bar, as shown in *Figure 12.12*:

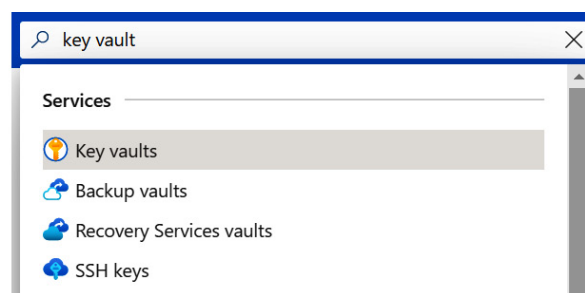


Figure 12.12: Looking for key vaults in the Azure search bar

2. Click the + New button at the top of the screen to create a new key vault. Select the ASO resource group you created earlier and give your key vault a name. Please note that your key vault name has to be unique, so consider adding extra characters to the name if it is not unique. Also, make sure to create the key vault in the same region as your AKS cluster. The resulting configuration is shown in *Figure 12.13: Create key vault*

Create key vault

Basics Access policy Networking Tags Review + create

Azure Key Vault is a cloud service used to manage keys, secrets, and certificates. Key Vault eliminates the need for developers to store security information in their code. It allows you to centralize the storage of your application secrets which greatly reduces the chances that secrets may be leaked. Key Vault also allows you to securely store secrets and keys backed by Hardware Security Modules or HSMs. The HSMs used are Federal Information Processing Standards (FIPS) 140-2 Level 2 validated. In addition, key vault provides logs of all access and usage attempts of your secrets so you have a complete audit trail for compliance.

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *	Azure subscription 1
Resource group *	ASO
	Create new
Instance details	
Key vault name * ⓘ	handsonaks-aso
Region *	West US 2
Pricing tier * ⓘ	Standard

Review + create

< Previous

Next : Access policy >

Figure 12.13: Key vault configuration

3. Now select Next: Access policy > to configure a new access policy. Here you will give the aso-mi managed identity you created in the previ-

ous section permission to do secret management in this key vault. To do this, start by clicking the + Add Access Policy button, as shown in *Figure 12.14*:

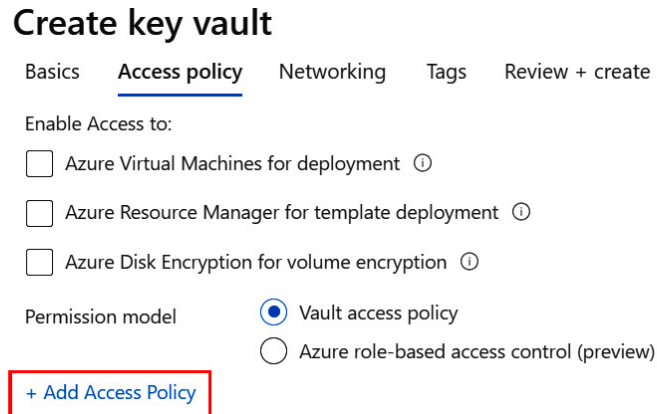


Figure 12.14: Clicking the + Add Access Policy button

4. In the resulting popup, select the Secret Management template, then click on None selected to select your managed identity. In the resulting popup, look for the aso-mi managed identity, select it, and then click Select followed by clicking on Add, as shown in *Figure 12.15*:

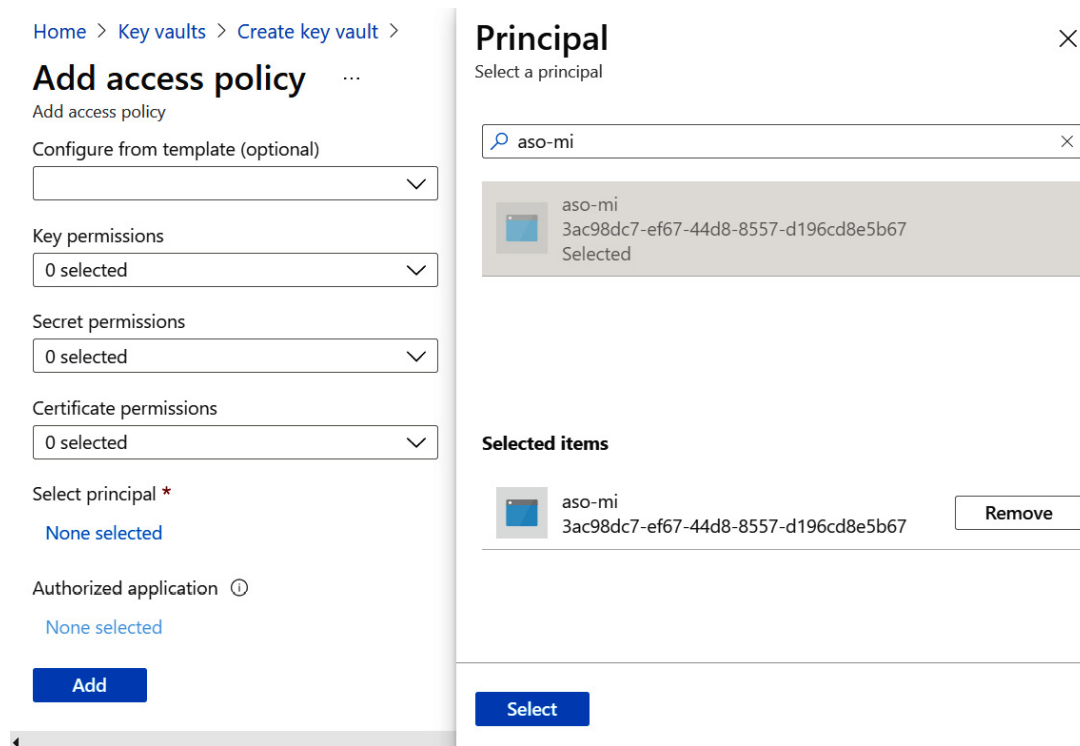


Figure 12.15: Adding the secret management permissions to the managed identity

5. This has configured the access policy in Key Vault. Now click the Review + create button, and in the last window hit Create to create the key vault. This should take a couple of minutes to complete.

Once your key vault has been deployed, you are ready to start installing ASO, which will be explained in the next section.

Setting up ASO on your cluster

Now that you have the required managed identity and Key Vault, you are ready to start deploying ASO on your cluster. You can do all these steps using Cloud Shell. Let's get started:

1. You created a new cluster in the *Creating a new AKS cluster* section. You will need to link the managed identity you created earlier to the cluster. The ASO components will be created in their own namespace, so you will also create a new namespace for this:

```
kubectl create namespace azureoperator-system
```

```
az aks pod-identity add --resource-group rg-handsonaks \
```

```
  --cluster-name handsonaks --namespace  
  azureoperator-system \
```

```
  --name aso-identity-binding \  
  --identity-resource-id <resource ID of managed identity>
```

2. Now you can install cert-manager on your cluster. You've done this once before in *Chapter 6, Securing your application with*

HTTPS, but at the end of the chapter you were asked to remove this component. You can install it again using the following command:

`kubectl apply -f`

<https://github.com/jetstack/cert-manager/releases/download/v1.1.0/cert-manager.yaml>

3. Track the deployment status of cert-manager using the following command:

`kubectl rollout status \`

`-n cert-manager deploy cert-manager-webhook`

Wait until the rollout shows that it's successfully rolled out, as shown in *Figure 12.16*:

```
user@Azure:~/Hands-On-Kubernetes-on-Azure/Chapter12$ kubectl rollout status \
> -n cert-manager deploy cert-manager-webhook
Waiting for deployment "cert-manager-webhook" rollout to finish: 0 of 1 updated replicas are available...
deployment "cert-manager-webhook" successfully rolled out
```

Figure 12.16: Checking the rollout status of cert-manager

4. Once cert-manager has fully rolled out, you can start the ASO installation. Start by adding the Helm repo for ASO using the following command:

```
helm repo add azureserviceoperator \
```

<https://raw.githubusercontent.com/Azure/azure-service-operator/master/charts>

5. Next, you need to provide configuration values for your ASO installation. Open the **values.yaml** file that is part of the code sample that comes with this chapter using the following command:

```
code values.yaml
```

Fill in all the required values in that file, as shown here:

```
1  azureTenantID: "<tenant ID>"
2  azureSubscriptionID: "<subscription ID>"
3  azureOperatorKeyvault: "<key vault name>"
4  azureClientID: "<client ID>"
5  cloudEnvironment: AzurePublicCloud
6  azureUseMI: true
7  image:
8    repository:
mcr.microsoft.com/k8s/azureserviceoperator:0.1.16800
9  installAadPodIdentity: true
10 aad-pod-identity:
11  azureIdentityBinding:
```



```
12  name: aso-identity-binding
13  selector: aso_manager_binding
14  azureIdentity:
15    enabled: True
16    name: aso-identity
17    type: 0
18    resourceID: "<resource ID>"
19    clientID: "<client ID>"
```

As shown in the previous code sample, you will need to provide your tenant ID, subscription ID, key vault name, client ID of the managed identity (twice), and resource ID of the managed identity. You can find the tenant ID and subscription ID with the following command:

```
az account show
```

This will return an output similar to *Figure 12.17*, in which the tenant ID and subscription ID have been highlighted:

```

user@Azure:~/Hands-On-Kubernetes-on-Azure/Chapter12$ az account show
{
  "environmentName": "AzureCloud",
  "homeTenantId": "1cf4b872-ae04-44c8-8318-2ba43e95f591",
  "id": "ede7a1e5-4121-427f-876e-e100eba989a0",
  "isDefault": true,
  "managedByTenants": [],
  "name": "Azure subscription 1",
  "state": "Enabled",
  "tenantId": "1cf4b872-ae04-44c8-8318-2ba43e95f591",
  "user": {
    "cloudShellID": true,
    "name": "live.com#handson-aks-book@outlook.com",
    "type": "user"
  }
}

```

Figure 12.17: Getting the subscription ID and tenant ID

6. Once you have the values filled in, you can install ASO using the following command:

```

helm upgrade --install aso \
  azureserviceoperator/azure-service-operator \
  -n azureoperator-system --create-namespace \
  -f values.yaml

```

7. The installation process takes a couple of minutes. Wait until the following command returns a successful rollout:

```

kubectl rollout status deploy \
  -n azureoperator-system azureoperator-controller-manager

```

The output should look similar to *Figure 12.18*:

```

user@Azure:~/Hands-On-Kubernetes-on-Azure/Chapter12$ kubectl rollout status deploy \
> -n azureoperator-system azureoperator-controller-manager
Waiting for deployment "azureoperator-controller-manager" rollout to finish: 0 of 1 updated replicas are available...
deployment "azureoperator-controller-manager" successfully rolled out

```

Figure 12.18: Checking the status of the deployments for ASO

8. At the time of writing, there was an issue with the **aadpodidbinding** label on the deployment of **azureoperator-controller-manager**. This can, however, be fixed by applying a patch, to apply a new label to that deployment. The patch has been provided in the files for the chapter, specifically in the **patch.yaml** file:
- spec:

```
template:
```

```
  metadata:
```

```
    labels:
```

```
      aadpodidbinding: aso-identity-binding
```

As you can see, the patch itself applies a new label to the pods in the deployment. You can apply the patch using the following command:

```
kubectl patch deployment \
  azureoperator-controller-manager \
  -n azureoperator-system \
  --patch "$(cat patch.yaml)"
```

This will ensure that you can use ASO in the next section.

Now that ASO has been deployed on your cluster, you are ready to start deploying Azure resources using Kubernetes and ASO. You will do that in the next section.

Deploying Azure Database for MySQL using ASO

In the previous section, you deployed ASO on your Kubernetes cluster. This means that now you can use the Kubernetes API to deploy Azure resources. In this section, you will create a MySQL database running on the Azure Database for MySQL service using YAML files that you will submit to Kubernetes using **kubectl**. Let's get started:

1. First, you need to create a resource group. The code for the resource group definition is also available in the code samples with this chapter. Create this file and save it as **rg.yaml**:
apiVersion: azure.microsoft.com/v1alpha1
kind: ResourceGroup
metadata:

name: aso-resources

spec:

location: <cluster location>

As you can see in the code for the resource, **apiVersion** refers to **azure.microsoft.com** and **kind** is **ResourceGroup**. Furthermore, you provide the details for the resource group, being its name and its location. Make sure to change **location** to the location of your cluster.

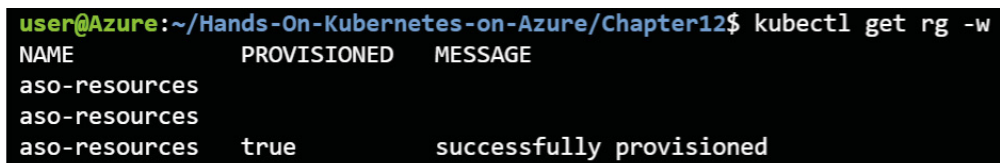
You can create this resource group using the following command:

```
kubectl create -f rg.yaml
```

To monitor the process of the resource group creation, you can use the following command:

```
kubectl get resourcegroup -w
```

This returns an output similar to *Figure 12.19*:

A terminal window with a black background and green text. The prompt is 'user@Azure: ~/Hands-On-Kubernetes-on-Azure/Chapter12\$'. The command entered is 'kubectl get rg -w'. The output shows three rows: the first two rows show 'NAME' as 'aso-resources' and 'PROVISIONED' as an empty field; the third row shows 'NAME' as 'aso-resources', 'PROVISIONED' as 'true', and 'MESSAGE' as 'successfully provisioned'.

NAME	PROVISIONED	MESSAGE
aso-resources		
aso-resources		
aso-resources	true	successfully provisioned

Figure 12.19: Monitoring the creation of a new resource group

2. Let's also verify that the resource group was created in Azure. To do so, look for the resource group name (**aso-resources**, in this example) in the Azure search bar, as shown in *Figure 12.20*:

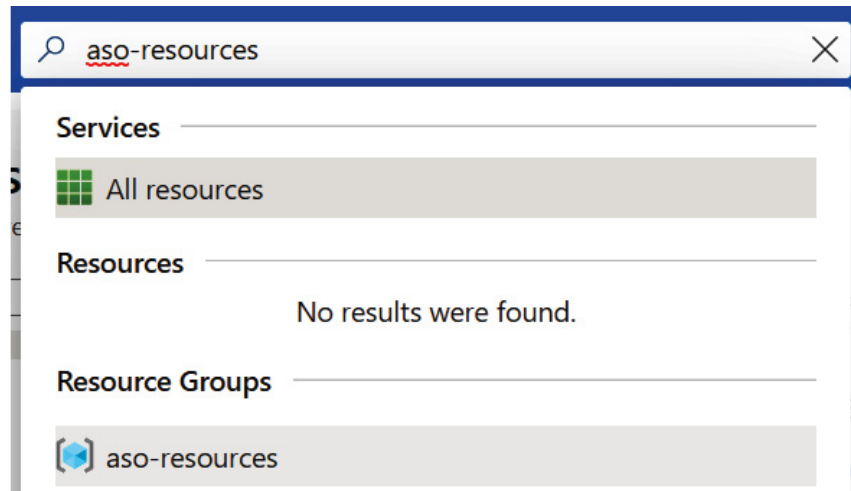


Figure 12.20: Searching for the resource group in the Azure portal

As you can see, the resource group is returned in the search results, meaning the resource group was successfully created.

3. Now you can create the MySQL server. You won't create a virtual machine to run MySQL, but rather create a managed MySQL server on Azure. To create this, you can use the **mysql-server.yaml** file that is provided for you:

```
1  apiVersion: azure.microsoft.com/v1alpha1
2  kind: MySQLServer
3  metadata:
4    name: <mysql-server-name>
5  spec:
6    location: <cluster location>
7    resourceGroup: aso-resources
8    serverVersion: "8.0"
9    sslEnforcement: Disabled
10   createMode: Default
11   sku:
12     name: B_Gen5_1
13     tier: Basic
14     family: Gen5
15     size: "5120"
16     capacity: 1
```

This file contains specific configurations for the MySQL server. A number of elements are worth pointing out:

1. **Line 2:** Here you define that you will create a **MySQLServer** instance.
2. **Line 4:** Here you give the server a name.

This name has to be globally unique, so con-

sider appending your initials to the server name.

3. **Line 6:** The location of the MySQL server you will create. Make sure to change **location** to the location of your cluster.
4. **Line 9: `sslEnforcement`** is disabled for this demo. This has been done to make the demo easier to follow. If you create a production cluster, it is highly recommended to enable **`sslEnforcement`**.
5. **Line 11-16:** Here you define the size of the MySQL server. In this case, you are creating a basic server with 5 GB of capacity. If you plan to use this for production use cases, you will likely need a larger server.

You can create the MySQL server using the following command:

```
kubectl create -f mysql-server.yaml
```

This will take a couple of minutes to complete.

You can follow the progress using the following command:

```
kubectl get mysqlserver -w
```


This will return an output similar to *Figure 12.21*:

```
user@Azure:~/Hands-On-Kubernetes-on-Azure/Chapter12$ kubectl get mysqlserver -w
NAME          PROVISIONED  MESSAGE
wp-helm-mysql
wp-helm-mysql
wp-helm-mysql
wp-helm-mysql
wp-helm-mysql  true         request submitted to Azure
wp-helm-mysql  true         successfully provisioned
```

Figure 12.21: Monitoring the creation of the MySQL server

If you were to run into errors when creating the MySQL server, please refer to the ASO documentation at

<https://github.com/Azure/azure-service-operator/blob/master/docs/troubleshooting.md>.

Once you get the message that the server has successfully been provisioned, you can exit out of this command by pressing *Ctrl + C*.

4. After the MySQL server, you can create the MySQL database. The definition of the MySQL database has been provided in the `mysql-database.yaml` file:

```
1  apiVersion: azure.microsoft.com/v1alpha1
2  kind: MySQLDatabase
3  metadata:
```

```
4 name: wordpress-db
5 spec:
6   resourceGroup: aso-resources
7   server: <mysql-server-name>
```

The definition of the database is providing a name and referring to the server you created earlier. To create the database, you can use the following command:

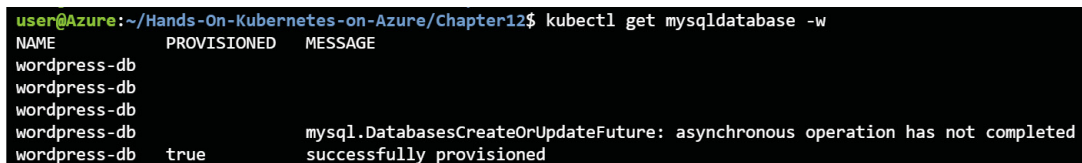
```
kubectl create -f mysql-database.yaml
```

This will take a couple of seconds to complete.

You can follow the progress using the following command:

```
kubectl get mysqldatabase -w
```

This will return an output similar to *Figure 12.22*:



```
user@Azure:~/Hands-On-Kubernetes-on-Azure/Chapter12$ kubectl get mysqldatabase -w
NAME          PROVISIONED  MESSAGE
wordpress-db  false        mysql.DatabasesCreateOrUpdateFuture: asynchronous operation has not completed
wordpress-db  false        mysql.DatabasesCreateOrUpdateFuture: asynchronous operation has not completed
wordpress-db  false        mysql.DatabasesCreateOrUpdateFuture: asynchronous operation has not completed
wordpress-db  true         successfully provisioned
```

Figure 12.22: Monitoring the creation of the MySQL database

Once you get the message that the database has successfully been provisioned, you can exit

out of this command by pressing *Ctrl + C*.

5. You can create a firewall rule that will allow traffic to your database. In this example, you will create a rule that will allow traffic from all sources. In a production environment, this is not recommended. For the recommended networking configurations for Azure Database for MySQL, please refer to the documentation:

<https://docs.microsoft.com/azure/mysql/flexible-server/concepts-networking>. The configuration for the firewall rule has been provided in the **mysql-firewall.yaml** file:

```
1  apiVersion: azure.microsoft.com/v1alpha1
2  kind: MySQLFirewallRule
3  metadata:
4    name: allow-all-mysql
5  spec:
6    resourceGroup: aso-resources
7    server: <mysql-server-name>
8    startIpAddress: 0.0.0.0
9    endIpAddress: 255.255.255.255
```

As you can see, we refer to the MySQL server that was created earlier and allow traffic from

all IP addresses (meaning from **0.0.0.0** to **255.255.255.255**).

To create the firewall rule, you can use the following command:

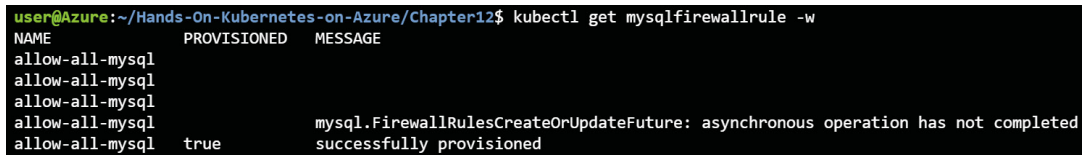
```
kubectl create -f mysql-firewall.yaml
```

This will take a couple of seconds to complete.

You can follow the progress using the following command:

```
kubectl get mysqlfirewallrule -w
```

This will return an output similar to *Figure 12.23*:



NAME	PROVISIONED	MESSAGE
allow-all-mysql	false	mysql.FirewallRulesCreateOrUpdateFuture: asynchronous operation has not completed
allow-all-mysql	false	mysql.FirewallRulesCreateOrUpdateFuture: asynchronous operation has not completed
allow-all-mysql	false	mysql.FirewallRulesCreateOrUpdateFuture: asynchronous operation has not completed
allow-all-mysql	false	mysql.FirewallRulesCreateOrUpdateFuture: asynchronous operation has not completed
allow-all-mysql	true	successfully provisioned

Figure 12.23: Monitoring the creation of the MySQL firewall rule

Once you get the message that the firewall rule has successfully been provisioned, you can exit out of this command by pressing *Ctrl + C*.

6. Let's verify that all of this was successfully created in the Azure portal. To do so, start by searching for the MySQL server name (**wp-**

helm-mysql in this example) in the Azure search bar as shown in *Figure 12.24*. Click on the server to go to the details:

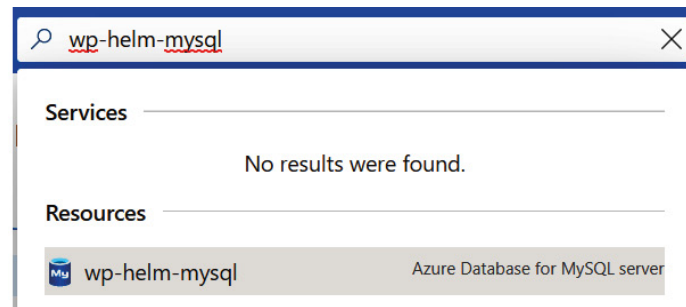


Figure 12.24: Searching for the MySQL server in the Azure portal

7. This will take you to the Overview pane of the MySQL server. Scroll down in this pane and expand the Available resources section. Here you should see that **wordpress-db** was created, as shown in *Figure 12.25*:

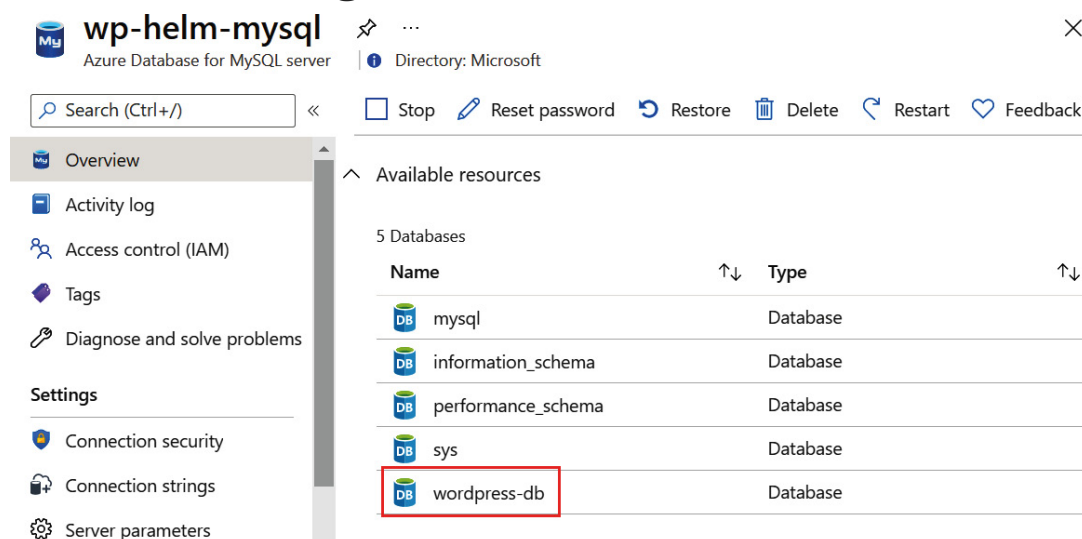


Figure 12.25: The database created through ASO is shown in the Azure portal

8. From the MySQL server pane, click on Connection security in the left-hand navigation to verify the firewall rule. You should see the firewall rule you created through ASO on this pane, as shown in *Figure 12.26*:

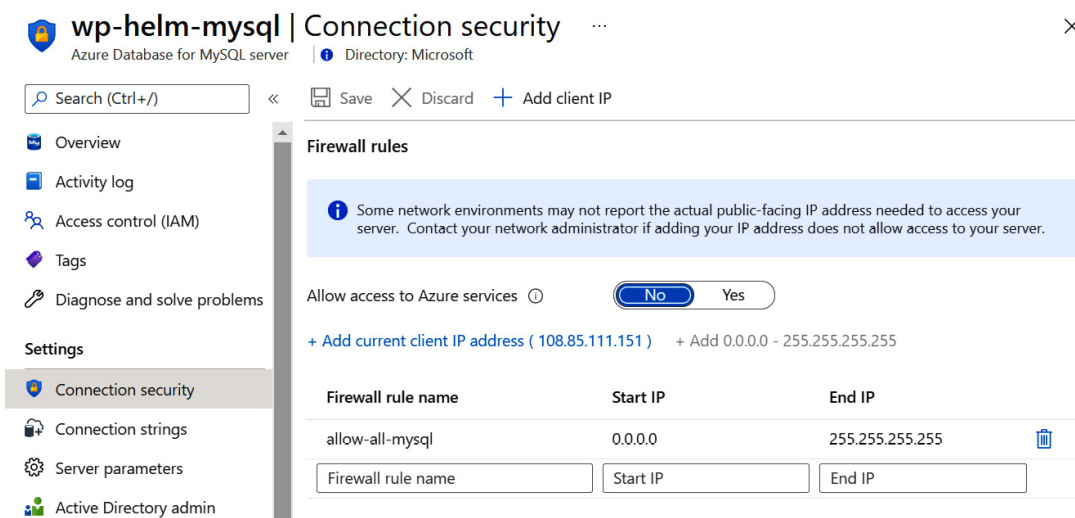


Figure 12.26: The firewall rule created through ASO is set on the MySQL server

This verifies that you were able to create a MySQL server with a database in Azure and configure its firewall settings.

In this section, you've used ASO to create a MySQL server, as well as a database on that

server, and then finally configured its firewall. You were able to do all of this using Kubernetes YAML files. ASO translated those YAML files to Azure and created the resources for you. Finally, you were able to confirm everything was created and configured in the Azure portal.

In the next and final section, you will use this database to support the WordPress application.

Creating an application using the MySQL database

You now have a MySQL database. To showcase that you can use this database to configure an application, you will use the WordPress application. You can install this using Helm and provide the connection information to your database in the Helm configuration:

1. To start, you will need the connection information to your database server. When you installed ASO on your cluster, you configured it to use Key Vault as a secret store rather than

Kubernetes secrets. You will need this connection information to connect WordPress to your Azure MySQL database. Search for **Key Vaults** in the Azure search bar, as shown in *Figure 12.27*, click on Key vaults, and then select the key vault you created earlier in the chapter:

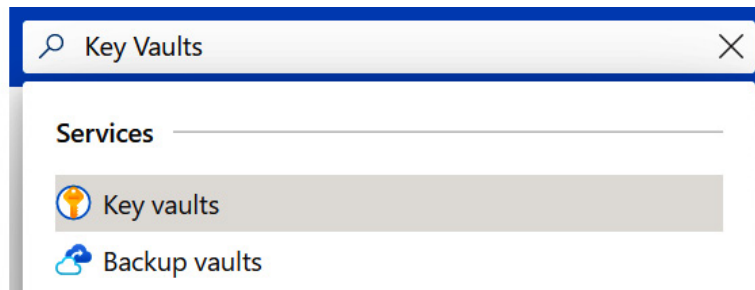


Figure 12.27: Searching for key vaults in the Azure portal

2. In the resulting pane, click on Secrets in the left-hand navigation and then click on the secret, as shown in *Figure 12.28*. The name of this secret follows the naming convention **<object type>-<Kubernetes namespace>-<object name>**.

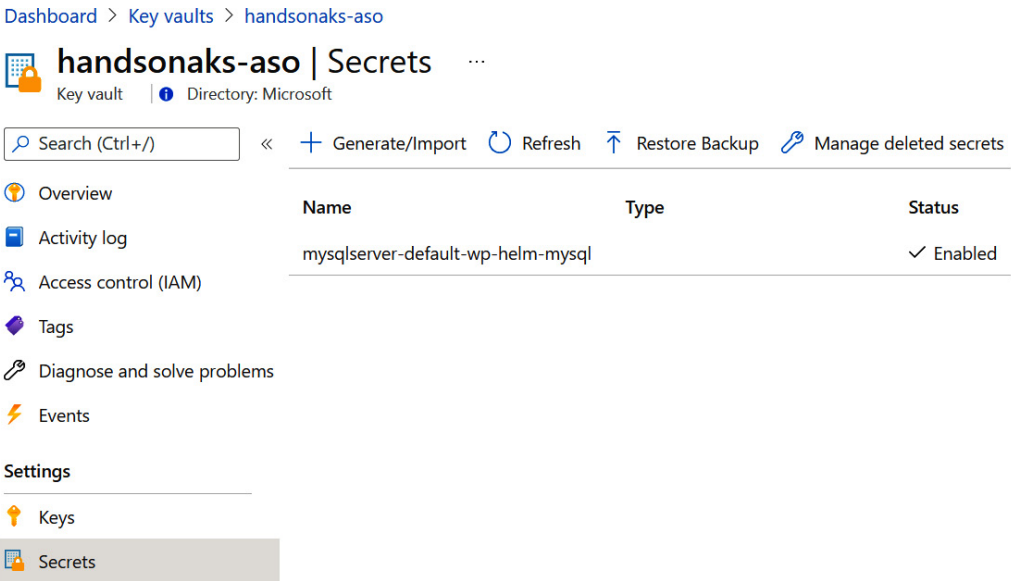


Figure 12.28: The MySQL secret in the Azure portal

3. You will then get a view with multiple versions of your secret; click the current version as shown in *Figure 12.29*:

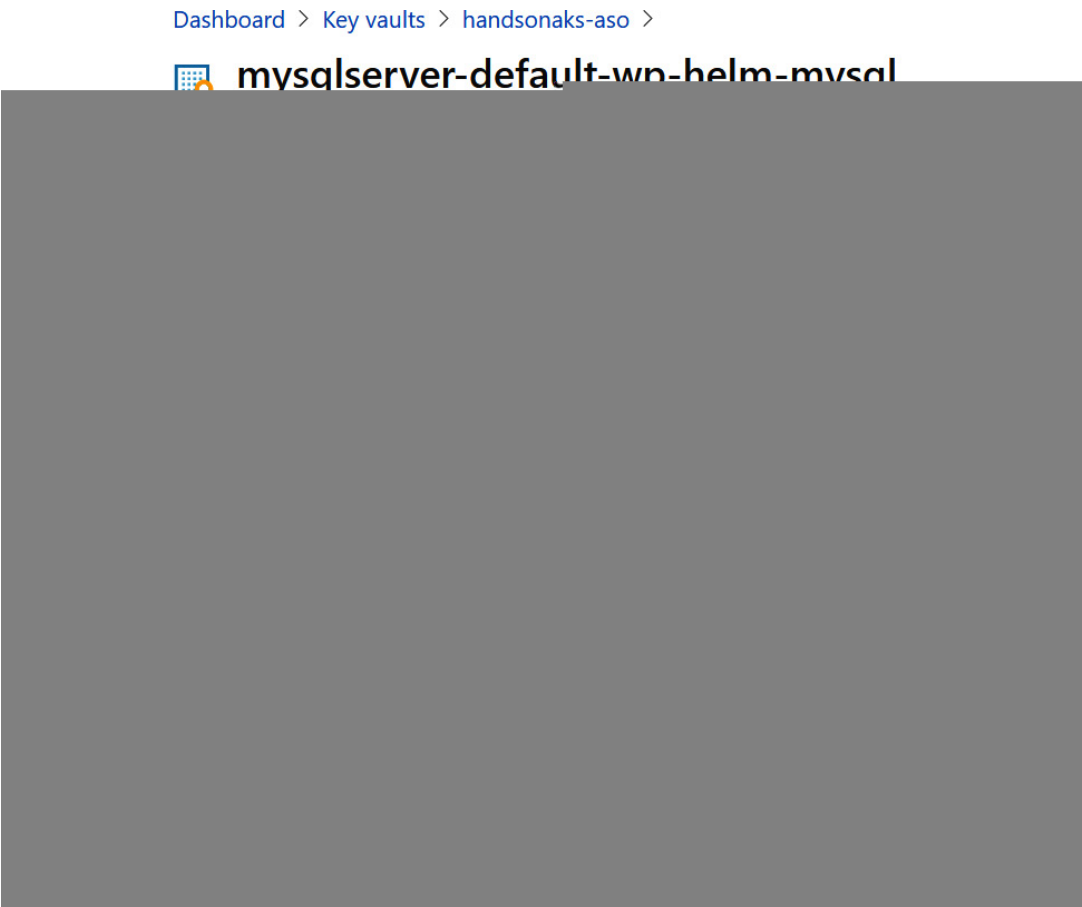


Figure 12.29: Different secret versions in your key vault

Now, copy the value of the secret, as shown in *Figure 12.30*:

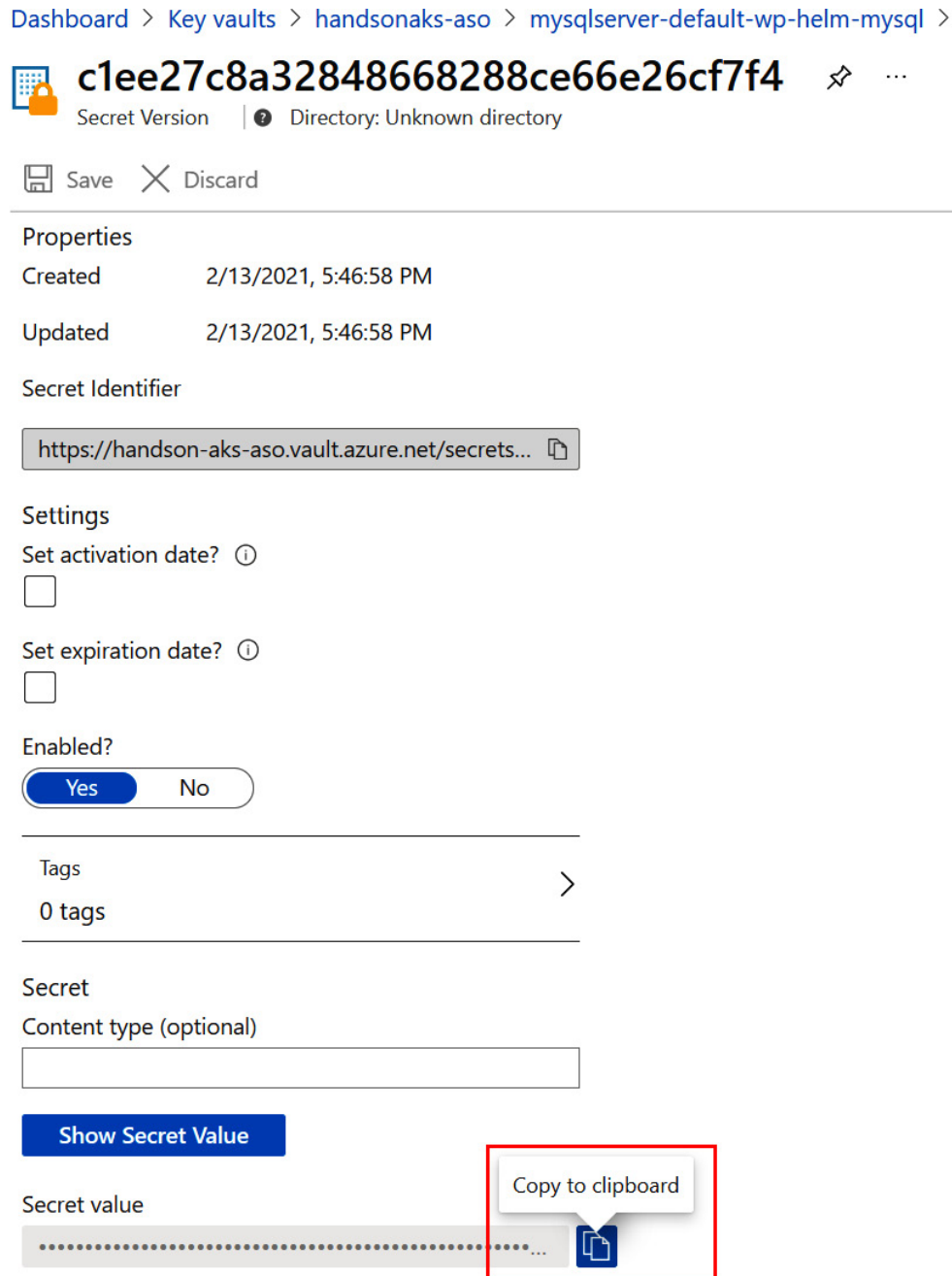


Figure 12.30: Copying the value of the secret to clipboard

4. The secret contains several pieces of information related to your database connection that you will need for the Helm installation. It contains the fully qualified server name, the username, and the password. The values in the secret are Base64 encoded. To make working with this secret easier, a shell script has been provided that will give you the required decoded values. To run this script, use the following command:

`sh decode-secret.sh <secret value>`

An example is shown in *Figure 12.31*:

```
user@Azure:~/Hands-On-Kubernetes-on-Azure/Chapter12$ sh decode-secret.sh '{"fullyQualifiedServerName":"d3AtaGVsbS1teXNxbC5teXNxbC5kYXRhYmFzZS5henVyZS5jb20=","fullyQualifiedUsername":"bUFGdW5WeERiekB3cC1oZWxtLW15c3Fs","mysqlServerName":"d3AtaGVsbS1teXNxbA=","password":"MWYzeVR2YmsqRyQwcFF2bg==" ,"username":"bUFGdW5WeERieg=="}'
externalDatabase.host="wp-helm-mysql.mysql.database.azure.com"
externalDatabase.user="mAFunVxDbz@wp-helm-mysql"
externalDatabase.password="1f3yTvbK*G$0pQvn"
```

Figure 12.31: Decoding the secret

5. You can use the values outputted by the previous step to configure Helm to use your Azure MySQL database. The following Helm command will set up WordPress on your cluster, but use an external database:

```
helm repo add bitnami
https://charts.bitnami.com/bitnami
helm install wp bitnami/wordpress \
  --set mariadb.enabled=false \
  --set externalDatabase.host='<decoded host
value>' \
  --set externalDatabase.user='<decoded user
value>' \
  --set externalDatabase.password='<decoded
password value>' \
  --set externalDatabase.database='wordpress-
db' \
  --set externalDatabase.port='3306'
```

As you can see, with this command, you disabled the MariaDB installation by setting the **mariadb.enabled** value to **false** and then provided the connection information to the external database.

To monitor the setup of WordPress, you can use the following command:

```
kubectl get pods -w
```

This will take a couple of minutes to fully set up, and finally, you should see the WordPress

pod in a running state and ready, as shown in *Figure 12.32*:

```
user@Azure:~/Hands-On-Kubernetes-on-Azure/Chapter12$ kubectl get pods -w
```

NAME	READY	STATUS	RESTARTS	AGE
wp-wordpress-99f458496-pslcz	0/1	Pending	0	3s
wp-wordpress-99f458496-pslcz	0/1	Pending	0	10s
wp-wordpress-99f458496-pslcz	0/1	ContainerCreating	0	10s
wp-wordpress-99f458496-pslcz	0/1	Running	0	78s
wp-wordpress-99f458496-pslcz	1/1	Running	0	2m17s

Figure 12.32: WordPress pod in a running state

Once the pod is running and ready, you can stop this command by pressing *Ctrl + C*. If you remember the WordPress deployment in *Chapter 3, Application deployment on AKS*, there was a second pod present in the WordPress installation hosting a MariaDB database. This pod is no longer there since we replaced it with an Azure MySQL database.

6. Let's now finally connect to this WordPress application. You can get the public IP address of the WordPress website using the following command:

```
kubectl get service
```

This will show you the public IP, as shown in *Figure 12.33*:

```
user@Azure:~/Hands-On-Kubernetes-on-Azure/Chapter12$ kubectl get service
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.0.0.1	<none>	443/TCP	4h39m
wp-wordpress	LoadBalancer	10.0.174.94	51.143.111.39	80:32393/TCP,443:32765/TCP	12m

Figure 12.33: Getting the public IP of the WordPress website

Enter this IP address in your web browser's address bar and hit *Enter*. You should be able to see the WordPress landing page with the default demo post, as shown in *Figure 12.34*:



Figure 12.34: Browsing to the WordPress website

You now have a fully functional WordPress website hosted on Kubernetes, with the database being backed by Azure Database for MySQL.

7. This concluded the examples from this chapter. You created a number of resources and in-

stalled a number of cluster components. Let's also clean them up from the cluster using the following commands:

```
helm uninstall wp
kubectl delete -f mysql-firewall.yaml
kubectl delete -f mysql-database.yaml
kubectl delete -f mysql-server.yaml
kubectl delete -f rg.yaml
helm uninstall aso -n azureoperator-system
az aks pod-identity delete --resource-group rg-
handsonaks \
  --cluster-name handsonaks --namespace
azureoperator-system \
  --name aso-identity-binding
kubectl delete namespace azureoperator-
system
kubectl delete -f
https://github.com/jetstack/cert-
manager/releases/download/v1.1.0/cert-
manager.yaml
az group delete -n aso --yes
```

You've been able to connect an application on Kubernetes to an Azure-managed MySQL database. You used the WordPress Helm chart and provided custom values to configure this Helm chart to make it connect to the managed database.

Summary

This chapter introduced **Azure Service Operator (ASO)**. ASO is an open-source project that makes it possible to create Azure services using Kubernetes. This allows you as the user to not have to switch between the Azure portal or CLI and Kubernetes resource definitions.

In this chapter, you created a new AKS cluster and then installed ASO on this cluster. You then created a MySQL database on Azure using ASO. You verified that this database was available in Azure using the Azure portal.

Finally, you created a WordPress application on your Kubernetes cluster that connected to the ex-

ternal database. You verified that the application was running and available as you've seen in previous chapters.

In the next chapter, you will learn about other Azure integrations with AKS, namely Azure Security Center and Azure Defender for Kubernetes, which are used to monitor the security configuration of your cluster and mitigate threats.