Handling common failures in AKS

Kubernetes is a distributed system with many working parts. AKS abstracts most of it for you, but it is still your responsibility to know where to look and how to respond when bad things happen. Much of the failure handling is done automatically by Kubernetes; however, you will encounter situations where manual intervention is required.

There are two areas where things can go wrong in an application that is deployed on top of AKS. Either the cluster itself has issues, or the application deployed on top of the cluster has issues. This chapter focuses specifically on cluster issues. There are several things that can go wrong with a cluster.

The first thing that can go wrong is a node in the cluster can become unavailable. This can happen either due to an Azure infrastructure outage or due to an issue with the virtual machine itself, such as an operating system crash. Either way, Kubernetes monitors the cluster for node failures and will recover automatically. You will see this process in action in this chapter.

A second common issue in a Kubernetes cluster is out-of-resource failures. This means that the workload you are trying to deploy requires more resources than are available on your cluster. You will learn how to monitor these signals and how you can solve them.

Another common issue is problems with mounting storage, which happens when a node becomes unavailable. When a node in Kubernetes becomes unavailable, Kubernetes will not detach the disks attached to this failed node. This means that those disks cannot be used by workloads on other nodes. You will see a practical example of this and learn how to recover from this failure.

We will look into the following topics in depth in this chapter:

- Handling node failures
- Solving out-of-resource failures
- Handling storage mount issues

In this chapter, you will learn about common failure scenarios, as well as solutions to those scenarios. To start, we will introduce node failures.

> **Note:**
>
> Refer to Kubernetes the Hard Way (https://github.com/kelseyhightower/kubernetes-the-hard-way), an excellent tutorial, to get an idea about the blocks on which Kubernetes is built. For the Azure version, refer to Kubernetes the Hard Way – Azure Translation (https://github.com/ivanfioravanti/kubernetes-the-hard-way-on-azure).

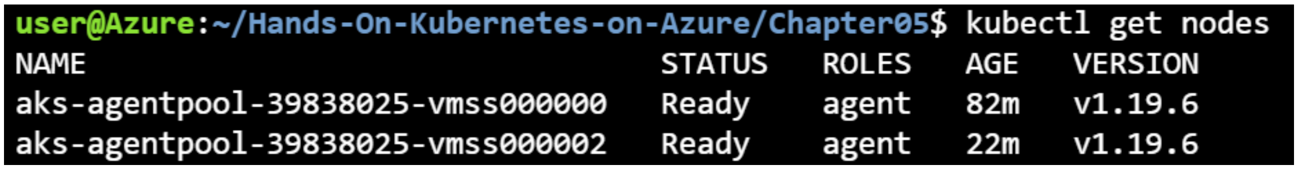
Handling node failures

Intentionally (to save costs) or unintentionally, nodes can go down. When that happens, you don't want to get the proverbial 3 a.m. call that your system is down. Kubernetes can handle moving workloads on failed nodes automatically for you instead. In this exercise, you are going to deploy the guestbook application and bring a node down in your cluster to see what Kubernetes does in response:

1. Ensure that your cluster has at least two nodes:

   ```
   kubectl get nodes
   ```

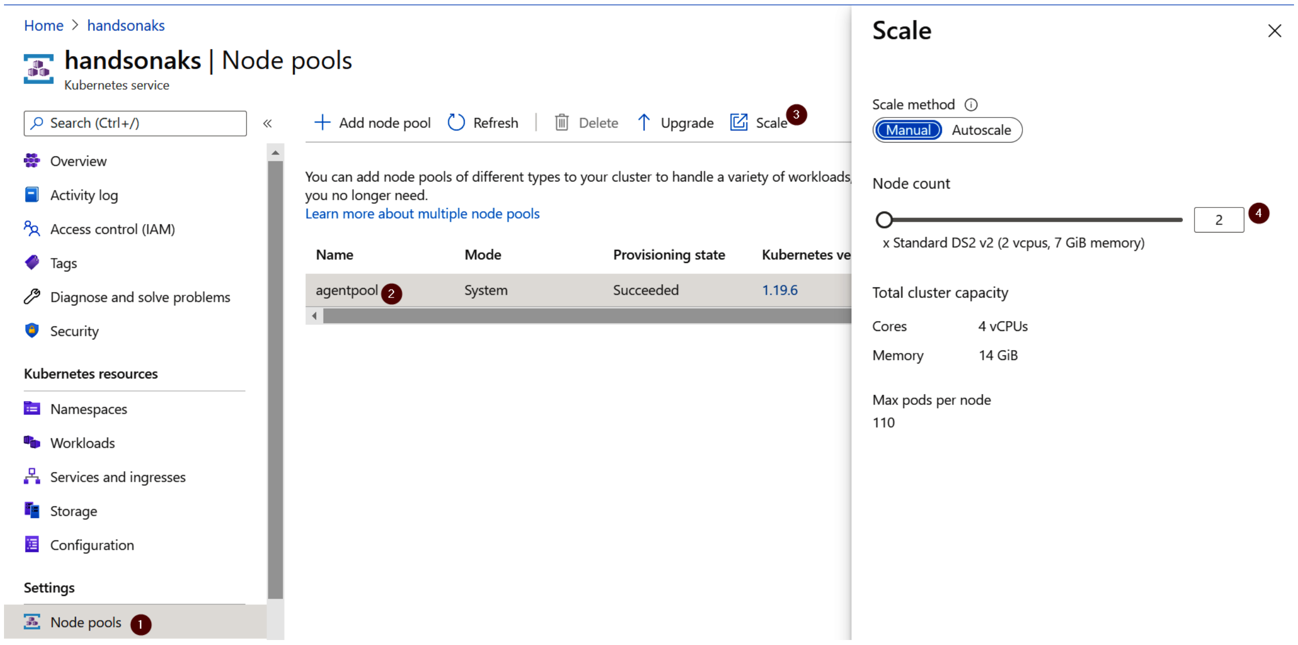
   This should generate an output as shown in *Figure 5.1*:



```
user@Azure:~/Hands-On-Kubernetes-on-Azure/Chapter05$ kubectl get nodes
NAME                                STATUS   ROLES   AGE   VERSION
aks-agentpool-39838025-vmss000000   Ready    agent   82m   v1.19.6
aks-agentpool-39838025-vmss000002   Ready    agent   22m   v1.19.6
```

Figure 5.1: List of nodes in the cluster

   If you don't have two nodes in your cluster, look for your cluster in the Azure portal, navigate to **Node pools**, select the pool you wish to scale, and click on **Scale**. You can then scale **Node count** to **2** nodes as shown in *Figure 5.2*:



You can add node pools of different types to your cluster to handle a variety of workloads, you no longer need.
Learn more about multiple node pools

Name	Mode	Provisioning state	Kubernetes ve
agentpool 2	System	Succeeded	1.19.6

Scale method: [✓ Manual] Autoscale

Node count: 2

x Standard DS2 v2 (2 vcpus, 7 GiB memory)

Total cluster capacity

Cores	4 vCPUs
Memory	14 GiB

Max pods per node
110

Figure 5.2: Scaling the cluster

2. As an example application in this section, deploy the guestbook application. The YAML file to deploy this has been provided in the source code for this chapter (`guestbook-all-in-one.yaml`). To deploy the guestbook application, use the following command:

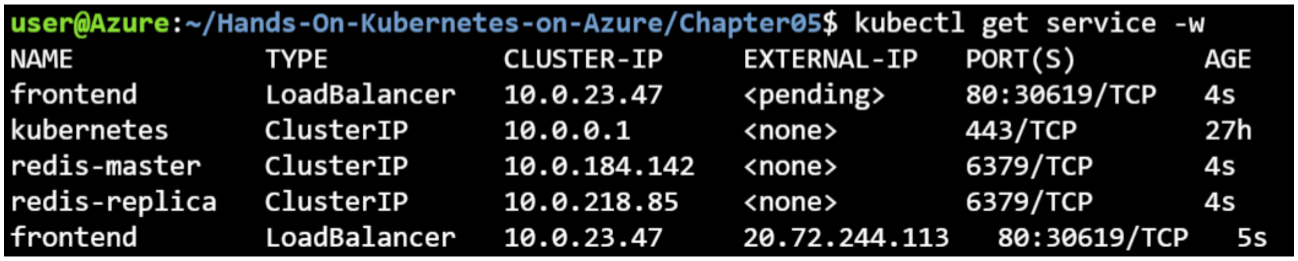
   ```
   kubectl create -f guestbook-all-in-one.yaml
   ```

3. Watch the `service` object until the public IP becomes available. To do this, type the following:

```
kubectl get service -w
```

> **Note**
>
> You can also get services in Kubernetes by using `kubectl get svc` rather than the full `kubectl get service`.

4. This will take a couple of seconds to show you the updated external IP. *Figure* 5.3 shows the service's public IP. Once you see the public IP appear (**20.72.244.113** in this case), you can exit the watch command by hitting *Ctrl* + C:



```
user@Azure:~/Hands-On-Kubernetes-on-Azure/Chapter05$ kubectl get service -w
NAME             TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)        AGE
frontend         LoadBalancer   10.0.23.47     <pending>       80:30619/TCP   4s
kubernetes       ClusterIP      10.0.0.1       <none>          443/TCP        27h
redis-master     ClusterIP      10.0.184.142   <none>          6379/TCP       4s
redis-replica    ClusterIP      10.0.218.85    <none>          6379/TCP       4s
frontend         LoadBalancer   10.0.23.47     20.72.244.113     80:30619/TCP   5s
```

Figure 5.3: The external IP of the frontend service changes from <pending> to an actual IP address

5. Go to `http://<EXTERNAL-IP>` (`http://20.72.244.113` in this case) as shown in *Figure* 5.4:

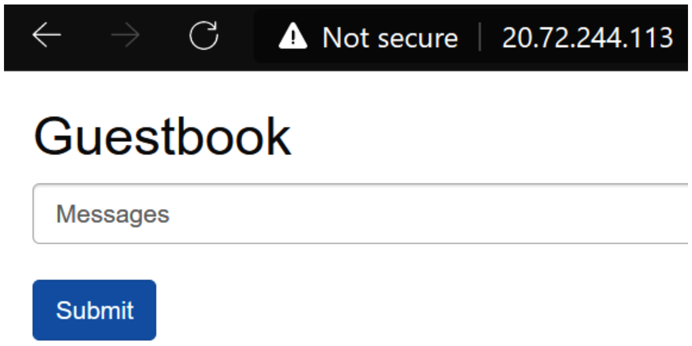
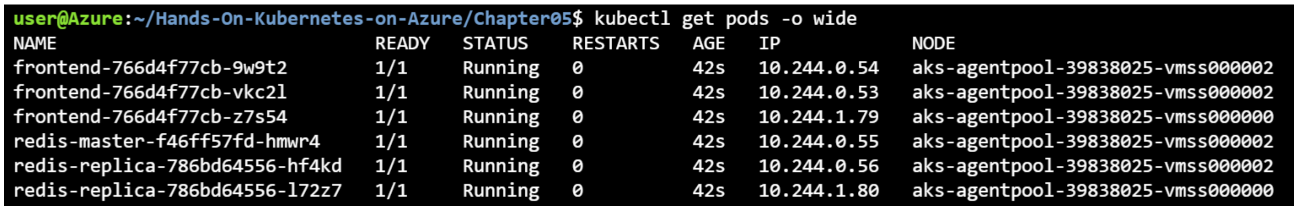


Figure 5.4: Browsing to the guestbook application

6. Let's see where the pods are currently running using the following command:

```
kubectl get pods -o wide
```

This will generate an output as shown in *Figure* 5.5:



```
user@Azure:~/Hands-On-Kubernetes-on-Azure/Chapter05$ kubectl get pods -o wide
NAME                             READY   STATUS    RESTARTS   AGE   IP            NODE
frontend-766d4f77cb-9w9t2        1/1     Running   0          42s   10.244.0.54   aks-agentpool-39838025-vmss000002
frontend-766d4f77cb-vkc2l        1/1     Running   0          42s   10.244.0.53   aks-agentpool-39838025-vmss000002
frontend-766d4f77cb-z7s54        1/1     Running   0          42s   10.244.1.79   aks-agentpool-39838025-vmss000000
redis-master-f46ff57fd-hmwr4     1/1     Running   0          42s   10.244.0.55   aks-agentpool-39838025-vmss000002
redis-replica-786bd64556-hf4kd   1/1     Running   0          42s   10.244.0.56   aks-agentpool-39838025-vmss000002
redis-replica-786bd64556-l72z7   1/1     Running   0          42s   10.244.1.80   aks-agentpool-39838025-vmss000000
```

Figure 5.5: The pods are spread between node 0 and node 2

This shows you that you should have the workload spread between node 0 and node 2.

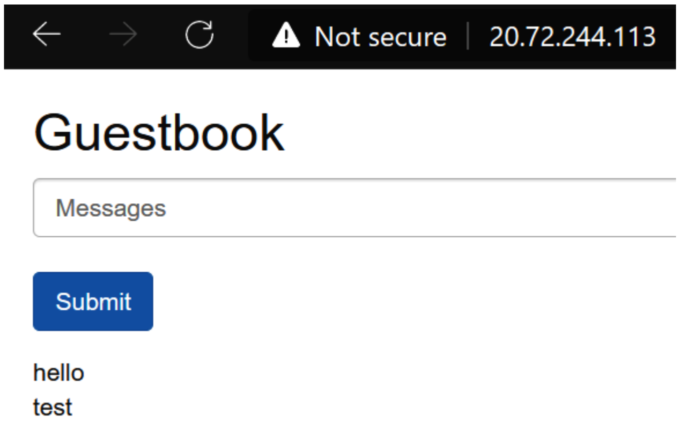
> **Note**
>
> In the example shown in *Figure 5.5*, the workload is spread between nodes 0 and 2. You might notice that node 1 is missing here. If you followed the example in *Chapter 4, Building scalable applications*, your cluster should be in a similar state. The reason for this is that as Azure removes old nodes and adds new nodes to a cluster (as you did in *Chapter 4, Building scalable applications*), it keeps incrementing the node counter.

7. Before introducing the node failures, there are two optional steps you can take to verify whether your application can continue to run. You can run the following command to hit the guestbook front end every 5 seconds and get the HTML. It's recommended to open this in a new Cloud Shell window:

```
while true; do
  curl -m 1 http://<EXTERNAl-IP>/;
  sleep 5;
done
```

> **Note**
>
> The preceding command will keep calling your application till you press *Ctrl* + *C*. There might be intermittent times where you don't get a reply, which is to be expected as Kubernetes takes a couple of minutes to rebalance the system.

You can also add some guestbook entries to see what happens to them when you cause the node to shut down. This will display an output as shown in *Figure 5.6*:

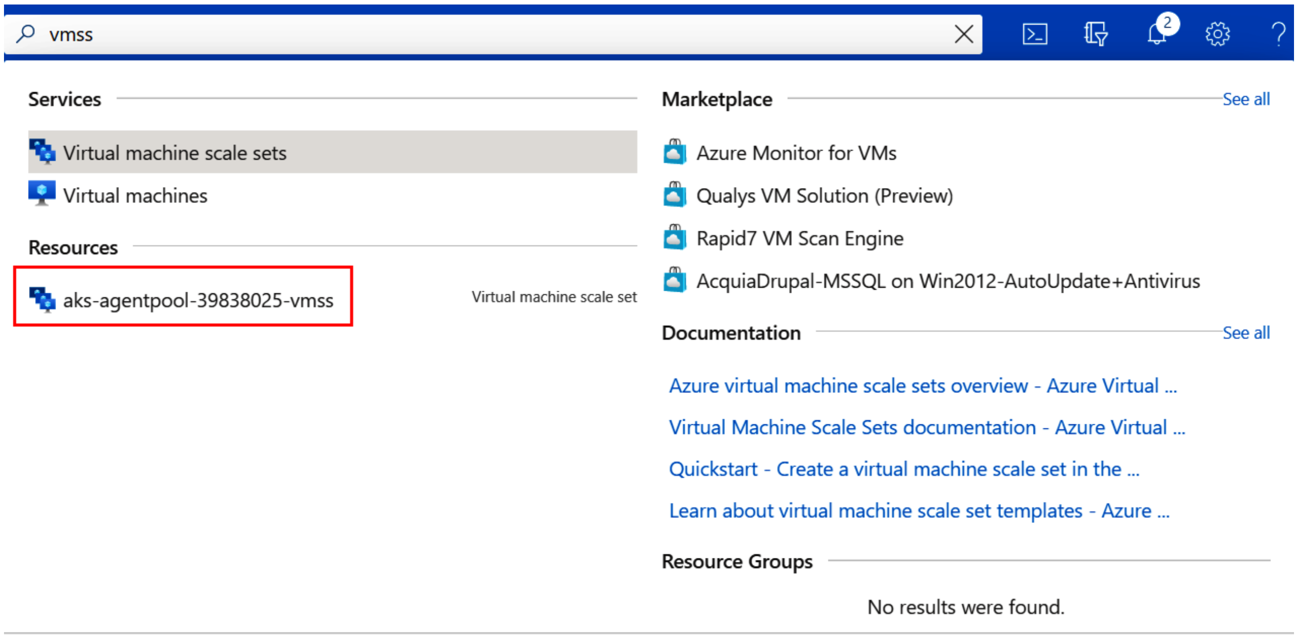


hello
test

Figure 5.6: Writing a couple of messages in the guestbook

8. In this example, you are exploring how Kubernetes handles a node failure. To demonstrate this, shut down a node in the cluster. You can shut down either node, although for maximum impact it is recommended you shut down the node from *step* 6 that hosted the most pods. In the case of the example shown, node 2 will be shut down.

   To shut down this node, look for **VMSS** (**virtual machine scale sets**) in the Azure search bar, and select the scale set used by your cluster, as shown in *Figure* 5.7. If you have multiple scale sets in your subscription, select the one whose name corresponds to the node names shown in *Figure* 5.5:


vmss

Services

Virtual machine scale sets

Virtual machines

Resources

aks-agentpool-39838025-vmss Virtual machine scale set

Marketplace See all

Azure Monitor for VMs

Qualys VM Solution (Preview)

Rapid7 VM Scan Engine

AcquiaDrupal-MSSQL on Win2012-AutoUpdate+Antivirus

Documentation See all

Azure virtual machine scale sets overview - Azure Virtual ...

Virtual Machine Scale Sets documentation - Azure Virtual ...

Quickstart - Create a virtual machine scale set in the ...

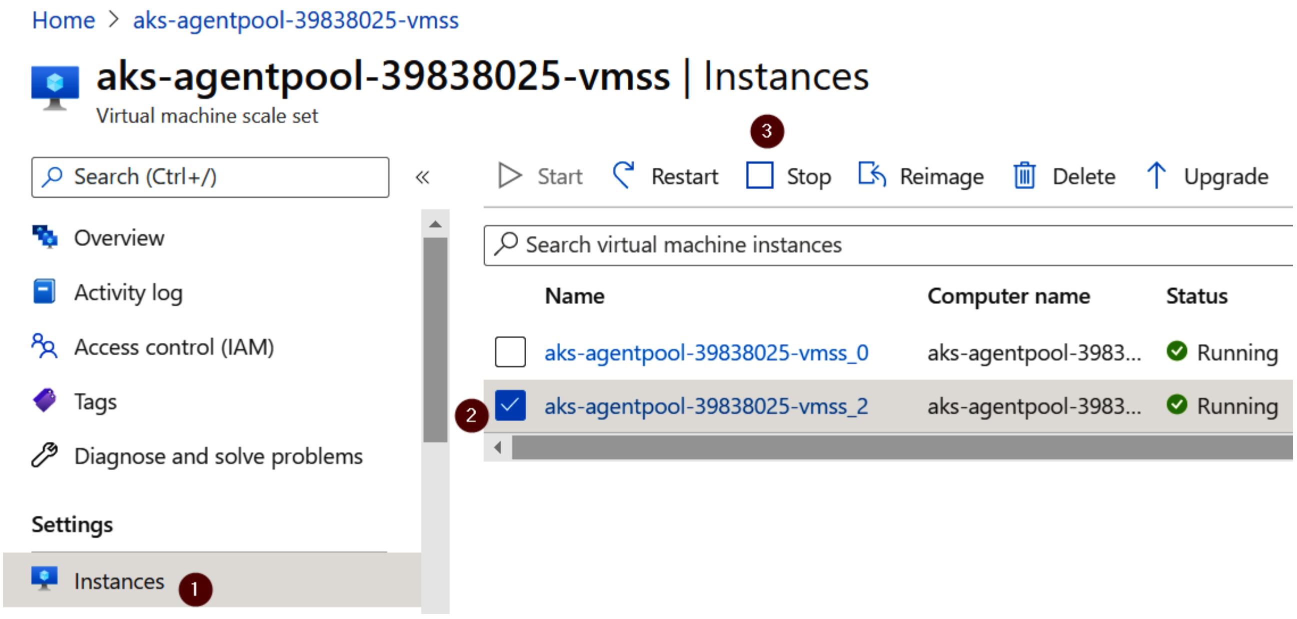
Learn about virtual machine scale set templates - Azure ...

Resource Groups

No results were found.

Figure 5.7: Looking for the scale set hosting your cluster

After navigating to the pane of the scale set, go to the **Instances** view, select the instance you want to shut down, and then hit the **Stop** button, as shown in *Figure 5.8*:


Home > aks-agentpool-39838025-vmss

aks-agentpool-39838025-vmss | Instances

Virtual machine scale set

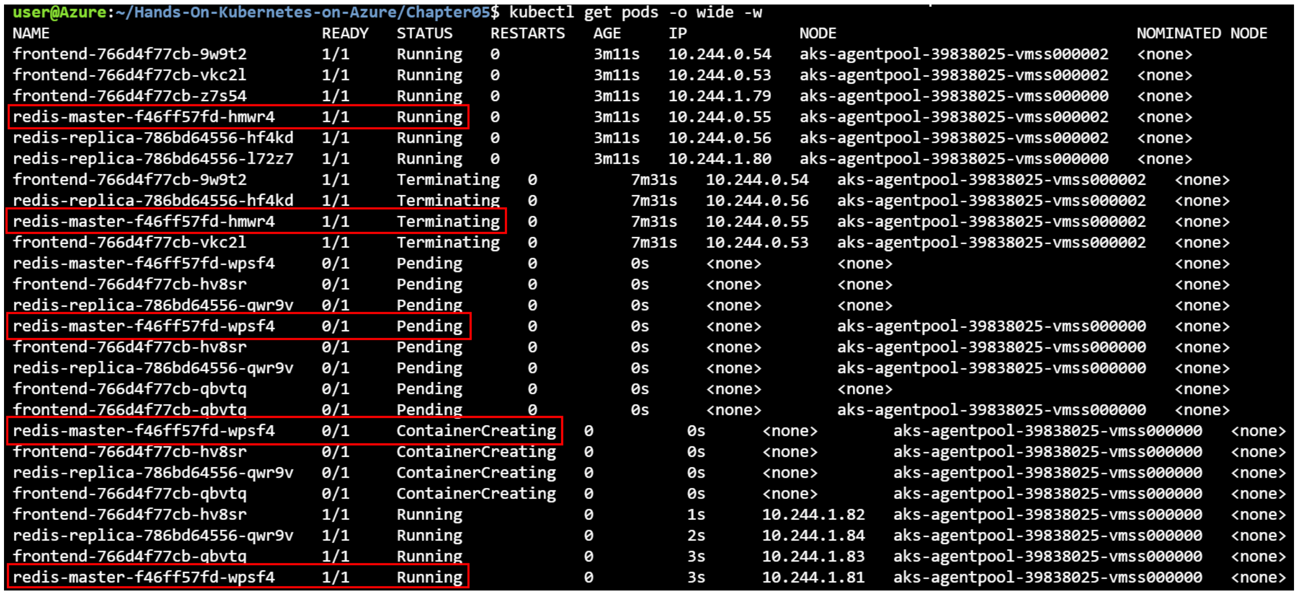
Name	Computer name	Status
aks-agentpool-39838025-vmss_0	aks-agentpool-3983...	Running
aks-agentpool-39838025-vmss_2	aks-agentpool-3983...	Running

Figure 5.8: Shutting down node 2

This will shut down the node. To see how Kubernetes will react with your pods, you can watch the pods in your cluster via the following command:

```
kubectl get pods -o wide -w
```

After a while, you should notice additional output, showing you that the pods got rescheduled on the healthy host, as shown in *Figure* 5.9:



```
user@Azure:~/Hands-On-Kubernetes-on-Azure/Chapter05$ kubectl get pods -o wide -w
NAME                             READY   STATUS    RESTARTS   AGE     IP            NODE                                NOMINATED NODE
frontend-766d4f77cb-9w9t2        1/1     Running   0          3m11s   10.244.0.54   aks-agentpool-39838025-vmss000002   <none>
frontend-766d4f77cb-vkc2l        1/1     Running   0          3m11s   10.244.0.53   aks-agentpool-39838025-vmss000002   <none>
frontend-766d4f77cb-z7s54        1/1     Running   0          3m11s   10.244.1.79   aks-agentpool-39838025-vmss000000   <none>
redis-master-f46ff57fd-hmwr4     1/1     Running   0          3m11s   10.244.0.55   aks-agentpool-39838025-vmss000002   <none>
redis-replica-786bd64556-hf4kd   1/1     Running   0          3m11s   10.244.0.56   aks-agentpool-39838025-vmss000002   <none>
redis-replica-786bd64556-l72z7   1/1     Running   0          3m11s   10.244.1.80   aks-agentpool-39838025-vmss000000   <none>
frontend-766d4f77cb-9w9t2        1/1     Terminating   0          7m31s   10.244.0.54   aks-agentpool-39838025-vmss000002   <none>
redis-replica-786bd64556-hf4kd   1/1     Terminating   0          7m31s   10.244.0.56   aks-agentpool-39838025-vmss000002   <none>
redis-master-f46ff57fd-hmwr4     1/1     Terminating   0          7m31s   10.244.0.55   aks-agentpool-39838025-vmss000002   <none>
frontend-766d4f77cb-vkc2l        1/1     Terminating   0          7m31s   10.244.0.53   aks-agentpool-39838025-vmss000002   <none>
redis-master-f46ff57fd-wpsf4     0/1     Pending       0          0s      <none>        <none>                              <none>
frontend-766d4f77cb-hv8sr        0/1     Pending       0          0s      <none>        <none>                              <none>
redis-replica-786bd64556-qwr9v   0/1     Pending       0          0s      <none>        <none>                              <none>
redis-master-f46ff57fd-wpsf4     0/1     Pending       0          0s      <none>        aks-agentpool-39838025-vmss000000   <none>
frontend-766d4f77cb-hv8sr        0/1     Pending       0          0s      <none>        aks-agentpool-39838025-vmss000000   <none>
redis-replica-786bd64556-qwr9v   0/1     Pending       0          0s      <none>        aks-agentpool-39838025-vmss000000   <none>
frontend-766d4f77cb-qbvtq        0/1     Pending       0          0s      <none>        <none>                              <none>
frontend-766d4f77cb-qbvtq        0/1     Pending       0          0s      <none>        aks-agentpool-39838025-vmss000000   <none>
redis-master-f46ff57fd-wpsf4     0/1     ContainerCreating   0          0s      <none>        aks-agentpool-39838025-vmss000000   <none>
frontend-766d4f77cb-hv8sr        0/1     ContainerCreating   0          0s      <none>        aks-agentpool-39838025-vmss000000   <none>
redis-replica-786bd64556-qwr9v   0/1     ContainerCreating   0          0s      <none>        aks-agentpool-39838025-vmss000000   <none>
frontend-766d4f77cb-qbvtq        0/1     ContainerCreating   0          0s      <none>        aks-agentpool-39838025-vmss000000   <none>
frontend-766d4f77cb-hv8sr        1/1     Running             0          1s      10.244.1.82   aks-agentpool-39838025-vmss000000   <none>
redis-replica-786bd64556-qwr9v   1/1     Running             0          2s      10.244.1.84   aks-agentpool-39838025-vmss000000   <none>
frontend-766d4f77cb-qbvtq        1/1     Running             0          3s      10.244.1.83   aks-agentpool-39838025-vmss000000   <none>
redis-master-f46ff57fd-wpsf4     1/1     Running             0          3s      10.244.1.81   aks-agentpool-39838025-vmss000000   <none>
```

Figure 5.9: The pods from the failed node getting recreated on a healthy node

What you see here is the following:

- The Redis master pod running on **node 2** got terminated as the host became unhealthy.
- A new Redis master pod got created, on host **0**. This went through the stages **Pending**, **ContainerCreating**, and then **Running**.

Note

In the preceding example, Kubernetes picked up that the host was unhealthy before it rescheduled the pods. If you were to do `kubectl get nodes`, you would see node **2** is in a **NotReady** state. There is a configuration in Kubernetes called `pod-eviction-timeout` that defines how long the system will wait to reschedule pods on a healthy host. The default is 5 minutes.

9. If you recorded a number of messages in the guestbook during *step* 7, browse back to the guestbook application on its public IP. What you can see is that all your precious messages are gone! This shows the importance of having **PersistentVolumeClaims** (**PVCs**) for any data that you want to survive in the case of a node failure, which is not the case in our application here. You will see an example of this in the last section of this chapter.

In this section, you learned how Kubernetes automatically handles node failures by recreating pods on healthy nodes. In the next section, you will learn how you can diagnose and solve out-of-resource issues.

Solving out-of-resource failures

Another common issue that can come up with Kubernetes clusters is the cluster running out of resources. When the cluster doesn't have enough CPU power or memory to schedule additional pods, pods will become stuck in a `Pending` state. You have seen this behavior in *Chapter* 4, *Building scalable applications*, as well.

Kubernetes uses requests to calculate how much CPU power or memory a certain pod requires. The guestbook application has requests defined for all the deployments. If you open the `guestbook-all-in-one.yaml` file in the folder `Chapter05`, you'll see the following for the `redis-replica` deployment:

```
kind: Deployment
metadata:
  name: redis-replica
...
        resources:
          requests:
            cpu: 200m
            memory: 100Mi
```

This section explains that every pod for the `redis-replica` deployment requires `200m` of a CPU core (`200` milli or `20%`) and `100MiB` (Mebibyte) of memory. In your 2 CPU clusters (with node 1 shut down), scaling this to 10 pods will cause issues with the available resources. Let's look into this:

Note

In Kubernetes, you can use either the binary prefix notation or the base 10 notation to specify memory and storage. Binary prefix notation means using KiB (kibibyte) to represent 1,024 bytes, MiB (mebibyte) to represent 1,024 KiB, and Gib (gibibyte) to represent 1,024 MiB. Base 10 notation means using kB (kilobyte) to represent 1,000 bytes, MB (megabyte) to represent 1,000 kB, and GB (gigabyte) represents 1,000 MB.

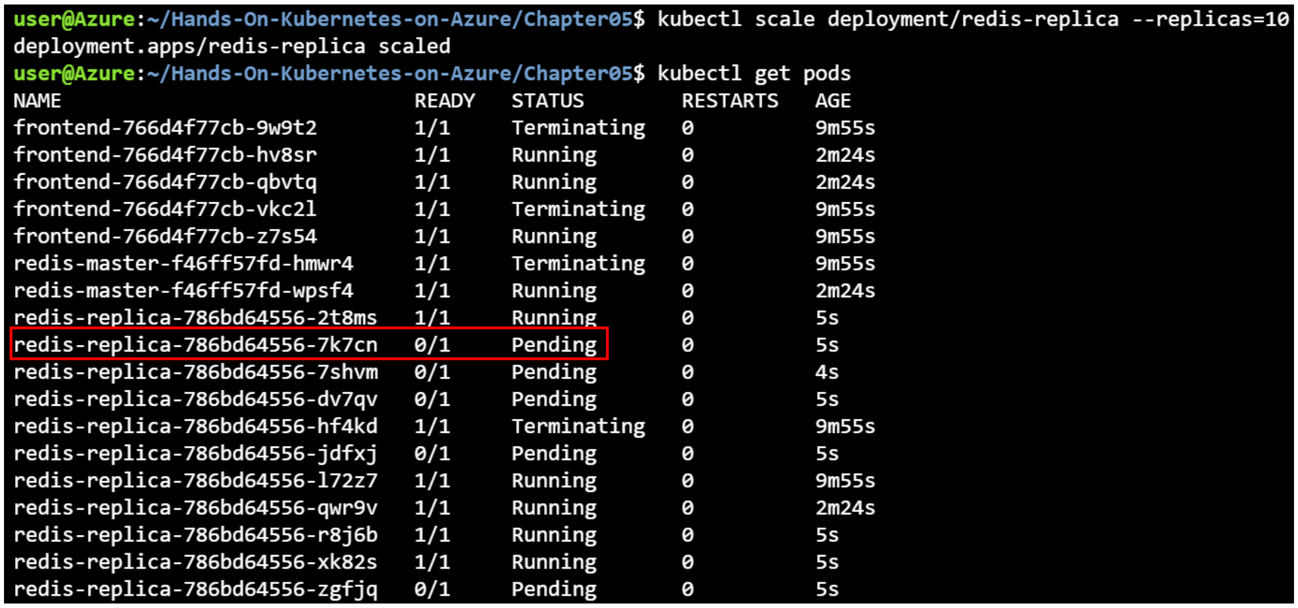
1. Let's start by scaling the `redis-replica` deployment to 10 pods:

    ```
    kubectl scale deployment/redis-replica --replicas=10
    ```

2. This will cause a couple of new pods to be created. We can check our pods using the following:

    ```
    kubectl get pods
    ```

    This will generate an output as shown in *Figure 5.10*:



```
user@Azure:~/Hands-On-Kubernetes-on-Azure/Chapter05$ kubectl scale deployment/redis-replica --replicas=10
deployment.apps/redis-replica scaled
user@Azure:~/Hands-On-Kubernetes-on-Azure/Chapter05$ kubectl get pods
NAME                              READY   STATUS        RESTARTS   AGE
frontend-766d4f77cb-9w9t2         1/1     Terminating   0          9m55s
frontend-766d4f77cb-hv8sr         1/1     Running       0          2m24s
frontend-766d4f77cb-qbvtq         1/1     Running       0          2m24s
frontend-766d4f77cb-vkc2l         1/1     Terminating   0          9m55s
frontend-766d4f77cb-z7s54         1/1     Running       0          9m55s
redis-master-f46ff57fd-hmwr4      1/1     Terminating   0          9m55s
redis-master-f46ff57fd-wpsf4      1/1     Running       0          2m24s
redis-replica-786bd64556-2t8ms    1/1     Running       0          5s
redis-replica-786bd64556-7k7cn    0/1     Pending       0          5s
redis-replica-786bd64556-7shvm    0/1     Pending       0          4s
redis-replica-786bd64556-dv7qv    0/1     Pending       0          5s
redis-replica-786bd64556-hf4kd    1/1     Terminating   0          9m55s
redis-replica-786bd64556-jdfxj    0/1     Pending       0          5s
redis-replica-786bd64556-l72z7    1/1     Running       0          9m55s
redis-replica-786bd64556-qwr9v    1/1     Running       0          2m24s
redis-replica-786bd64556-r8j6b    1/1     Running       0          5s
redis-replica-786bd64556-xk82s    1/1     Running       0          5s
redis-replica-786bd64556-zgfjq    0/1     Pending       0          5s
```

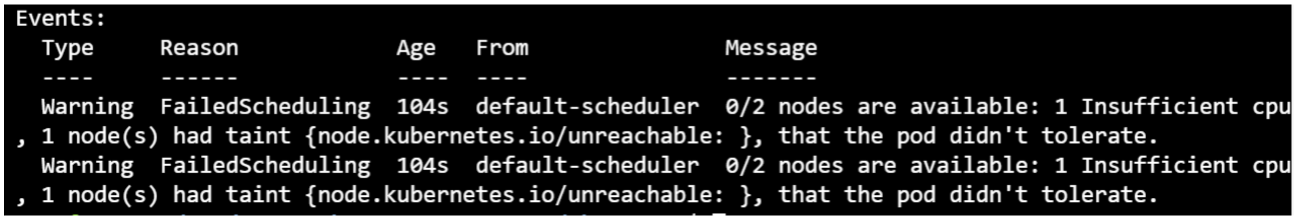
Figure 5.10: Some pods are in the Pending state

Highlighted here is one of the pods that are in the **Pending** state. This occurs if the cluster is out of resources.

3. We can get more information about these pending pods using the following command:

   ```
   kubectl describe pod redis-replica-<pod-id>
   ```

   This will show you more details. At the bottom of the describe command, you should see something like what's shown in *Figure* 5.11:



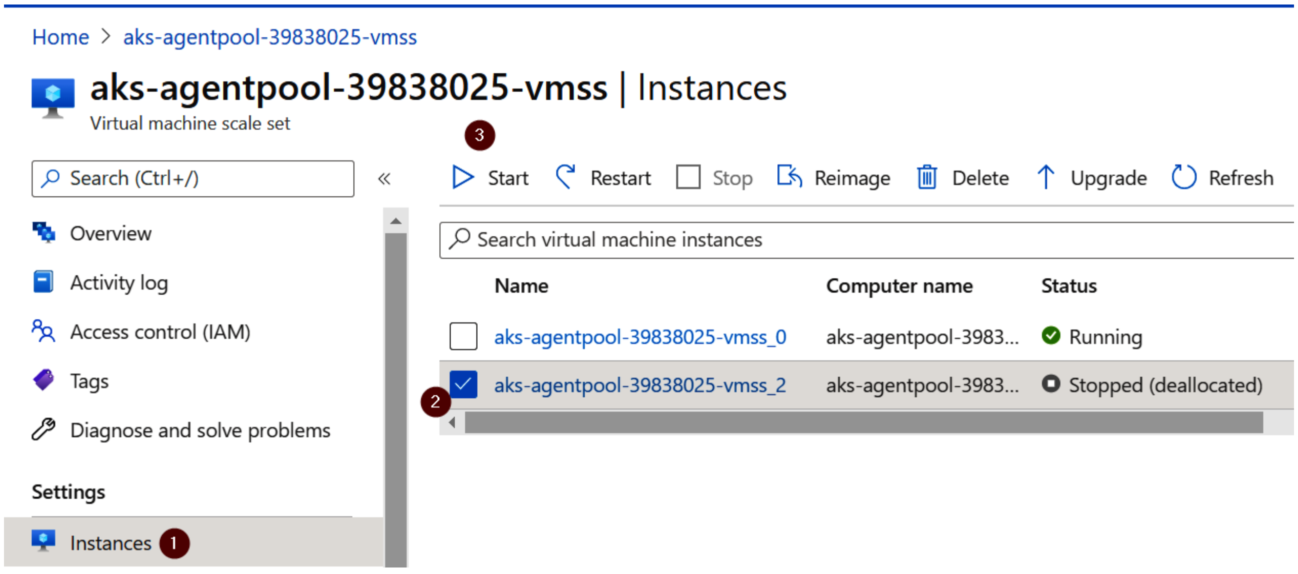
```
Events:
  Type     Reason            Age   From               Message
  ----     ------            ----  ----               -------
  Warning  FailedScheduling  104s  default-scheduler  0/2 nodes are available: 1 Insufficient cpu
, 1 node(s) had taint {node.kubernetes.io/unreachable: }, that the pod didn't tolerate.
  Warning  FailedScheduling  104s  default-scheduler  0/2 nodes are available: 1 Insufficient cpu
, 1 node(s) had taint {node.kubernetes.io/unreachable: }, that the pod didn't tolerate.
```

Figure 5.11: Kubernetes is unable to schedule this pod

   It explains two things:

   - One of the nodes is out of CPU resources.
   - One of the nodes has a taint (node.kubernetes.io/unreachable) that the pod didn't tolerate. This means that the node that is NotReady can't accept pods.

4. We can solve this capacity issue by starting up node **2** as shown in *Figure* 5.12. This can be done in a way similar to the shutdown process:



Home > aks-agentpool-39838025-vmss

aks-agentpool-39838025-vmss | Instances

Virtual machine scale set

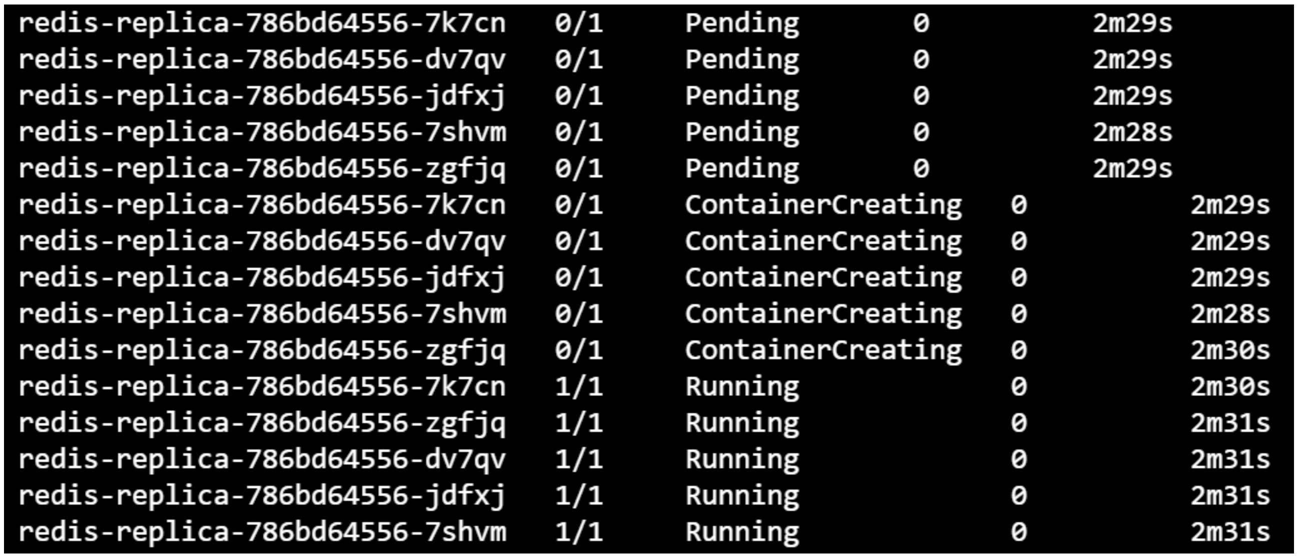
	Name	Computer name	Status
☐	aks-agentpool-39838025-vmss_0	aks-agentpool-3983...	Running
☑	aks-agentpool-39838025-vmss_2	aks-agentpool-3983...	Stopped (deallocated)

Figure 5.12: Start node 2 again

5. It will take a couple of minutes for the other node to become available again in Kubernetes. You can monitor the progress on the pods by executing the following command:

   ```
   kubectl get pods -w
   ```

   This will show you an output after a couple of minutes similar to *Figure* 5.13:

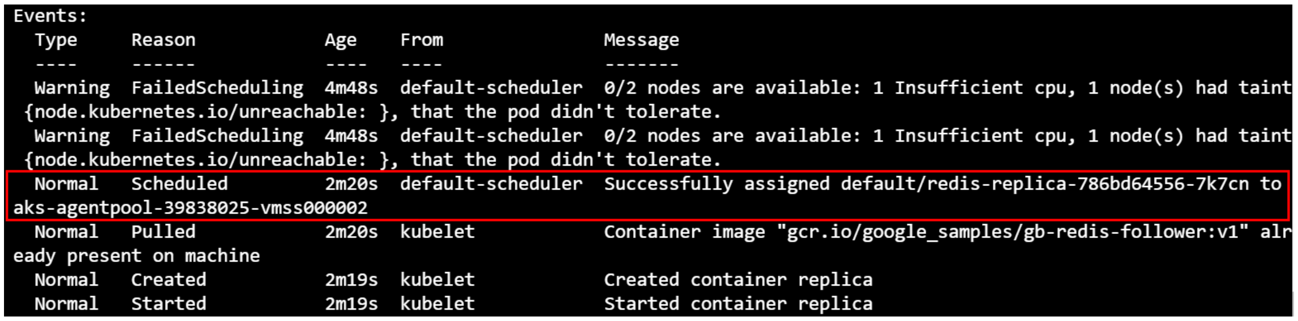


```
redis-replica-786bd64556-7k7cn   0/1     Pending       0          2m29s
redis-replica-786bd64556-dv7qv   0/1     Pending       0          2m29s
redis-replica-786bd64556-jdfxj   0/1     Pending       0          2m29s
redis-replica-786bd64556-7shvm   0/1     Pending       0          2m28s
redis-replica-786bd64556-zgfjq   0/1     Pending       0          2m29s
redis-replica-786bd64556-7k7cn   0/1     ContainerCreating   0          2m29s
redis-replica-786bd64556-dv7qv   0/1     ContainerCreating   0          2m29s
redis-replica-786bd64556-jdfxj   0/1     ContainerCreating   0          2m29s
redis-replica-786bd64556-7shvm   0/1     ContainerCreating   0          2m28s
redis-replica-786bd64556-zgfjq   0/1     ContainerCreating   0          2m30s
redis-replica-786bd64556-7k7cn   1/1     Running             0          2m30s
redis-replica-786bd64556-zgfjq   1/1     Running             0          2m31s
redis-replica-786bd64556-dv7qv   1/1     Running             0          2m31s
redis-replica-786bd64556-jdfxj   1/1     Running             0          2m31s
redis-replica-786bd64556-7shvm   1/1     Running             0          2m31s
```

Figure 5.13: Pods move from a Pending state to ContainerCreating to Running

   Here again, you see the container status change from **Pending**, to **ContainerCreating**, to finally **Running**.

6. If you re-execute the `describe` command on the previous pod, you'll see an output like what's shown in *Figure* 5.14:



```
Events:
  Type     Reason            Age    From               Message
  ----     ------            ----   ----               -------
  Warning  FailedScheduling  4m48s  default-scheduler  0/2 nodes are available: 1 Insufficient cpu, 1 node(s) had taint
 {node.kubernetes.io/unreachable: }, that the pod didn't tolerate.
  Warning  FailedScheduling  4m48s  default-scheduler  0/2 nodes are available: 1 Insufficient cpu, 1 node(s) had taint
 {node.kubernetes.io/unreachable: }, that the pod didn't tolerate.
  Normal   Scheduled         2m20s  default-scheduler  Successfully assigned default/redis-replica-786bd64556-7k7cn to
aks-agentpool-39838025-vmss000002
  Normal   Pulled            2m20s  kubelet            Container image "gcr.io/google_samples/gb-redis-follower:v1" alr
eady present on machine
  Normal   Created           2m19s  kubelet            Created container replica
  Normal   Started           2m19s  kubelet            Started container replica
```

Figure 5.14: When the node is available again, the Pending pods are assigned to that node

   This shows that after node 2 became available, Kubernetes scheduled the pod on that node, and then started the container.

In this section, you learned how to diagnose out-of-resource errors. You were able to solve the error by adding another node to the cluster. Before moving on to the final failure mode, clean up the guestbook deployment.

> **Note**
>
> In *Chapter 4, Building scalable applications*, the **cluster autoscaler** was introduced. The cluster autoscaler will monitor out-of-resource errors and add new nodes to the cluster automatically.

Let's clean up the guestbook deployment by running the following `delete` command:

```
kubectl delete -f guestbook-all-in-one.yaml
```

It is now also safe to close the other Cloud Shell window you opened earlier.

So far, you have learned how to recover from two failure modes for nodes in a Kubernetes cluster. First, you saw how Kubernetes handles a node going offline and how the system reschedules pods to a working node. After that, you saw how Kubernetes uses requests to manage the scheduling of pods on a node, and what happens when a cluster is out of resources. In the next section, you'll learn about another failure mode in Kubernetes, namely what happens when Kubernetes encounters storage mounting issues.

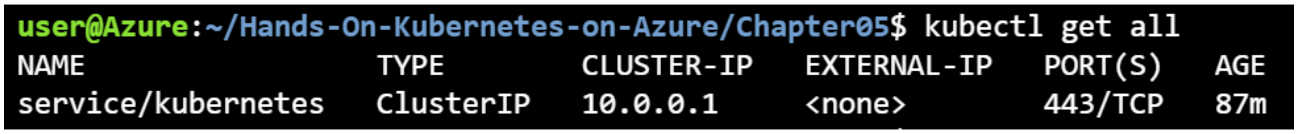
Fixing storage mount issues

Earlier in this chapter, you noticed how the guestbook application lost data when the Redis master was moved to another node. This happened because that sample application didn't use any persistent storage. In this section, you'll see an example of how PVCs can be used to prevent data loss when Kubernetes moves a pod to another node. You will see a common error that occurs when Kubernetes moves pods with PVCs attached, and you'll learn how to fix this.

For this, you will reuse the WordPress example from the previous chapter. Before starting, let's make sure that the cluster is in a clean state:

```
kubectl get all
```

This should show you just the one Kubernetes service, as in *Figure 5.15*:



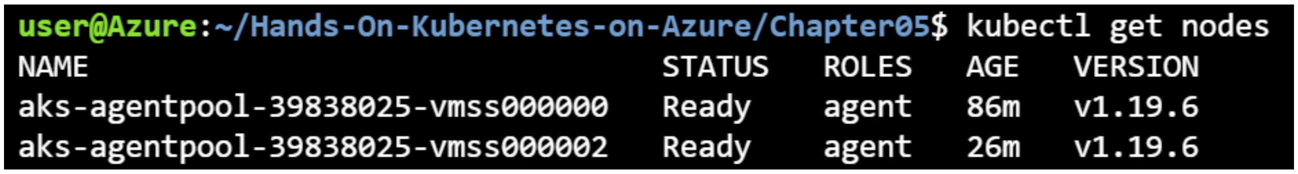
```
user@Azure:~/Hands-On-Kubernetes-on-Azure/Chapter05$ kubectl get all
NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.0.0.1     <none>        443/TCP   87m
```

Figure 5.15: You should only have the one Kubernetes service running for now

Let's also ensure that both nodes are running and **Ready**:

```
kubectl get nodes
```

This should show us both nodes in a **Ready** state, as in *Figure 5.16*:



```
user@Azure:~/Hands-On-Kubernetes-on-Azure/Chapter05$ kubectl get nodes
NAME                                STATUS   ROLES   AGE   VERSION
aks-agentpool-39838025-vmss000000   Ready    agent   86m   v1.19.6
aks-agentpool-39838025-vmss000002   Ready    agent   26m   v1.19.6
```

Figure 5.16: You should have two nodes available in your cluster

In the previous example, under the *Handling node failures* section, you saw that the messages stored in `redis-master` are lost if the pod gets restarted. The reason for this is that `redis-master` stores all data in its container, and whenever it is restarted, it uses the clean image without the data. In order to survive reboots, the data has to be stored outside. Kubernetes uses PVCs to abstract the underlying storage provider to provide this external storage.

To start this example, set up the WordPress installation.

Starting the WordPress installation

Let's start by installing WordPress. We will demonstrate how it works and then verify that storage is still present after a reboot.

If you have not done so yet in a previous chapter, add the Helm repository for Bitnami:

```
helm repo add bitnami https://charts.bitnami.com/bitnami
```

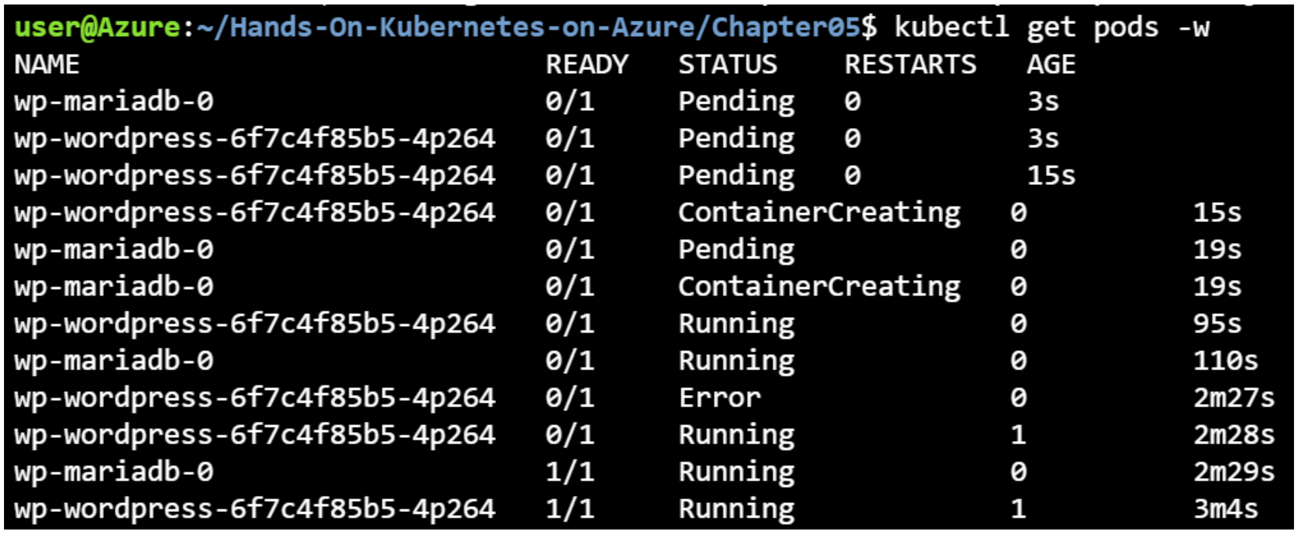
Begin reinstallation by using the following command:

```
helm install wp bitnami/wordpress
```

This will take a couple of minutes to process. You can follow the status of this installation by executing the following command:

```
kubectl get pods -w
```

After a couple of minutes, this should show you two pods with a status of **Running** and with a ready status of **1/1** for both pods, as shown in *Figure* 5.17:



```
user@Azure:~/Hands-On-Kubernetes-on-Azure/Chapter05$ kubectl get pods -w
NAME                              READY   STATUS    RESTARTS   AGE
wp-mariadb-0                      0/1     Pending   0          3s
wp-wordpress-6f7c4f85b5-4p264     0/1     Pending   0          3s
wp-wordpress-6f7c4f85b5-4p264     0/1     Pending   0          15s
wp-wordpress-6f7c4f85b5-4p264     0/1     ContainerCreating   0          15s
wp-mariadb-0                      0/1     Pending             0          19s
wp-mariadb-0                      0/1     ContainerCreating   0          19s
wp-wordpress-6f7c4f85b5-4p264     0/1     Running             0          95s
wp-mariadb-0                      0/1     Running             0          110s
wp-wordpress-6f7c4f85b5-4p264     0/1     Error               0          2m27s
wp-wordpress-6f7c4f85b5-4p264     0/1     Running             1          2m28s
wp-mariadb-0                      1/1     Running             0          2m29s
wp-wordpress-6f7c4f85b5-4p264     1/1     Running             1          3m4s
```

Figure 5.17: All pods will have the status of Running after a couple of minutes

You might notice that the `wp-wordpress` pod went through an Error status and was restarted afterward. This is because the `wp-mariadb` pod was not ready in time, and `wp-wordpress` went through a restart. You will learn more about readiness and how this can influence pod restarts in *Chapter* 7, *Monitoring the AKS cluster and the application.*

In this section, you saw how to install WordPress. Now, you will see how to avoid data loss using persistent volumes.

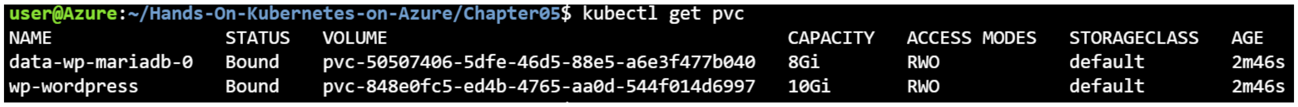
Using persistent volumes to avoid data loss

A **persistent volume** (**PV**) is the way to store persistent data in the cluster with Kubernetes. PVs were discussed in more detail in *Chapter 3*, *Application deployment on AKS*. Let's explore the PVs created for the WordPress deployment:

1. You can get the PersistentVolumeClaims using the following command:

   ```
   kubectl get pvc
   ```

   This will generate an output as shown in *Figure 5.18*:



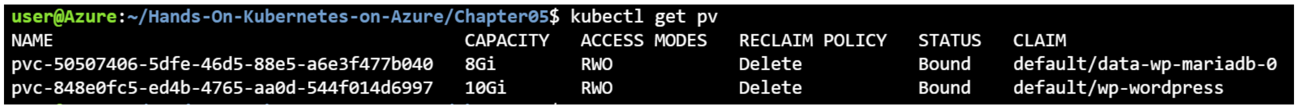
```
user@Azure:~/Hands-On-Kubernetes-on-Azure/Chapter05$ kubectl get pvc
NAME                 STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
data-wp-mariadb-0    Bound    pvc-50507406-5dfe-46d5-88e5-a6e3f477b040   8Gi        RWO            default        2m46s
wp-wordpress         Bound    pvc-848e0fc5-ed4b-4765-aa0d-544f014d6997   10Gi       RWO            default        2m46s
```

Figure 5.18: Two PVCs are created by the WordPress deployment

   A PersistentVolumeClaim will result in the creation of a PersistentVolume. The PersistentVolume is the link to the physical resource created, which is an Azure disk in this case. The following command shows the actual PVs that are created:

   ```
   kubectl get pv
   ```

   This will show you the two PersistentVolumes:



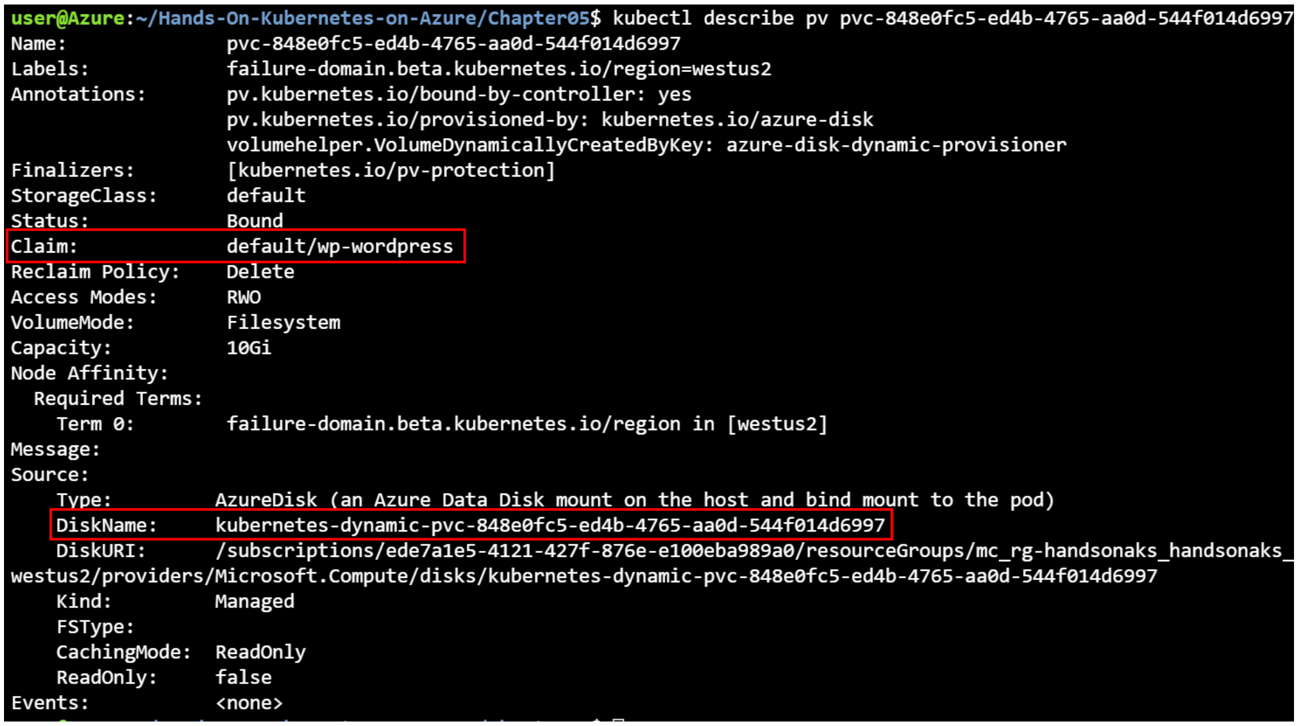
```
user@Azure:~/Hands-On-Kubernetes-on-Azure/Chapter05$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM
pvc-50507406-5dfe-46d5-88e5-a6e3f477b040   8Gi        RWO            Delete           Bound    default/data-wp-mariadb-0
pvc-848e0fc5-ed4b-4765-aa0d-544f014d6997   10Gi       RWO            Delete           Bound    default/wp-wordpress
```

Figure 5.19: Two PVs are created to store the data of the PVCs

   You can get more details about the specific PersistentVolumes that were created. Copy the name of one of the PVs, and run the following command:

   ```
   kubectl describe pv <pv name>
   ```

This will show you the details of that volume, as in *Figure* 5.20:



```
user@Azure:~/Hands-On-Kubernetes-on-Azure/Chapter05$ kubectl describe pv pvc-848e0fc5-ed4b-4765-aa0d-544f014d6997
Name:              pvc-848e0fc5-ed4b-4765-aa0d-544f014d6997
Labels:            failure-domain.beta.kubernetes.io/region=westus2
Annotations:       pv.kubernetes.io/bound-by-controller: yes
                   pv.kubernetes.io/provisioned-by: kubernetes.io/azure-disk
                   volumehelper.VolumeDynamicallyCreatedByKey: azure-disk-dynamic-provisioner
Finalizers:        [kubernetes.io/pv-protection]
StorageClass:      default
Status:            Bound
Claim:             default/wp-wordpress
Reclaim Policy:    Delete
Access Modes:      RWO
VolumeMode:        Filesystem
Capacity:          10Gi
Node Affinity:
  Required Terms:
    Term 0:        failure-domain.beta.kubernetes.io/region in [westus2]
Message:
Source:
    Type:         AzureDisk (an Azure Data Disk mount on the host and bind mount to the pod)
    DiskName:     kubernetes-dynamic-pvc-848e0fc5-ed4b-4765-aa0d-544f014d6997
    DiskURI:      /subscriptions/ede7a1e5-4121-427f-876e-e100eba989a0/resourceGroups/mc_rg-handsonaks_handsonaks_
westus2/providers/Microsoft.Compute/disks/kubernetes-dynamic-pvc-848e0fc5-ed4b-4765-aa0d-544f014d6997
    Kind:         Managed
    FSType:
    CachingMode:  ReadOnly
    ReadOnly:     false
Events:           <none>
```

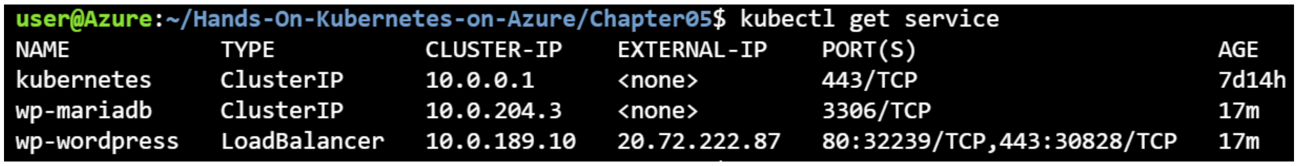
Figure 5.20: The details of one of the PVs

Here, you can see which PVC has claimed this volume and what the **DiskName** is in Azure.

2. Verify that your site is working:

```
kubectl get service
```

This will show us the public IP of our WordPress site, as seen in *Figure* 5.21:



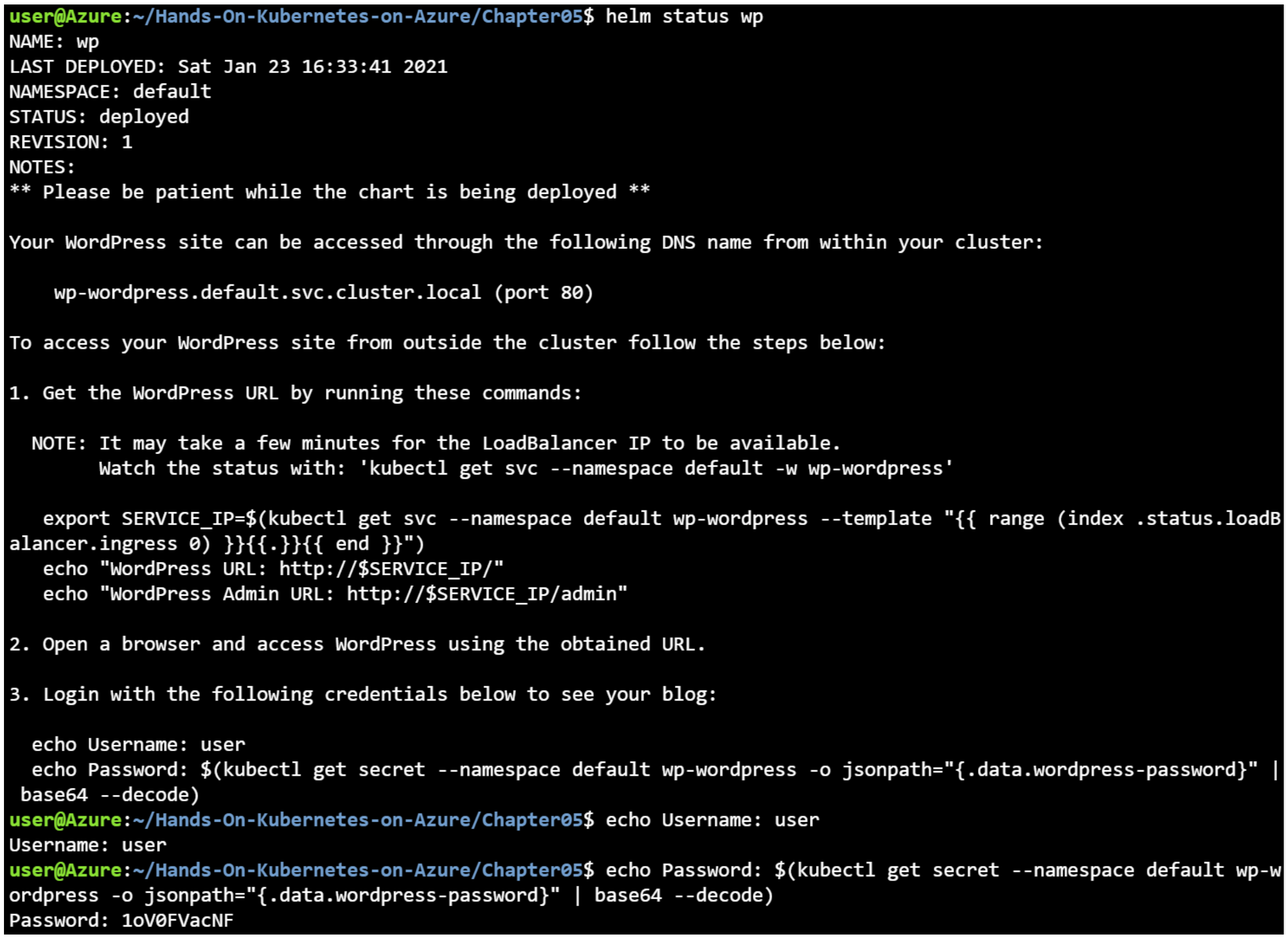
```
user@Azure:~/Hands-On-Kubernetes-on-Azure/Chapter05$ kubectl get service
NAME           TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)                      AGE
kubernetes     ClusterIP      10.0.0.1       <none>         443/TCP                      7d14h
wp-mariadb     ClusterIP      10.0.204.3     <none>         3306/TCP                     17m
wp-wordpress   LoadBalancer   10.0.189.10    20.72.222.87   80:32239/TCP,443:30828/TCP   17m
```

Figure 5.21: Public IP of the WordPress site

3. If you remember from *Chapter 3*, *Application deployment of AKS*, Helm showed you the commands you need to get the admin credentials for our WordPress site. Let's grab those commands and execute them to log on to the site as follows:

   ```
   helm status wp
   echo Username: user
   echo Password: $(kubectl get secret --namespace default wp-wordpress
   -o jsonpath="{.data.wordpress-password}" | base64 -d)
   ```

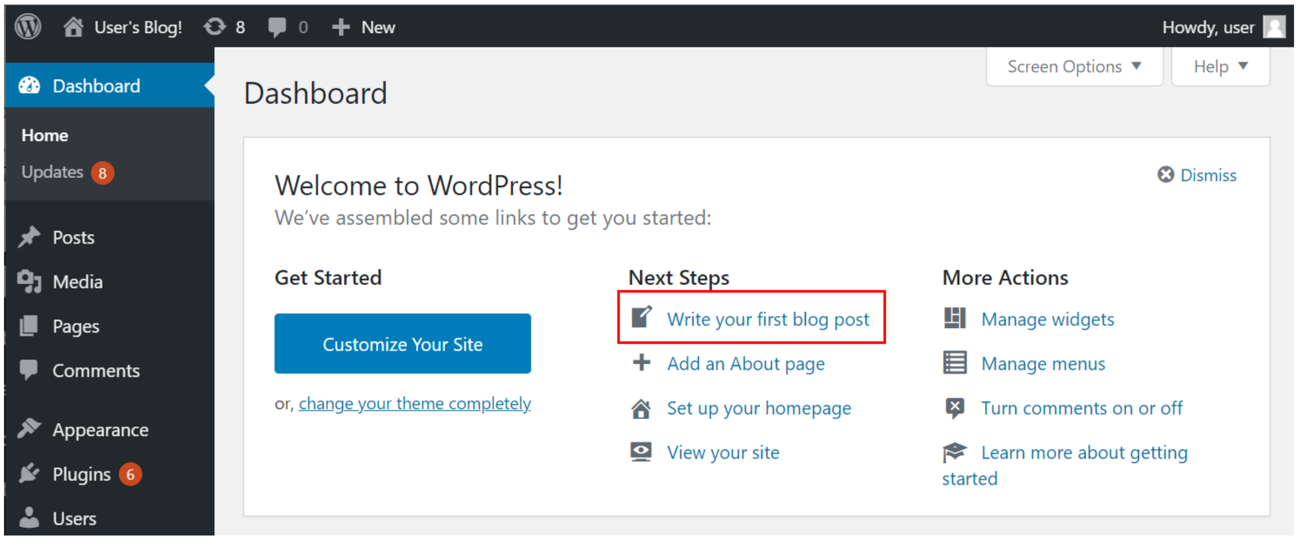
   This will show you the **username** and **password**, as displayed in *Figure 5.22*:



```
user@Azure:~/Hands-On-Kubernetes-on-Azure/Chapter05$ helm status wp
NAME: wp
LAST DEPLOYED: Sat Jan 23 16:33:41 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
** Please be patient while the chart is being deployed **

Your WordPress site can be accessed through the following DNS name from within your cluster:

    wp-wordpress.default.svc.cluster.local (port 80)

To access your WordPress site from outside the cluster follow the steps below:

1. Get the WordPress URL by running these commands:

  NOTE: It may take a few minutes for the LoadBalancer IP to be available.
        Watch the status with: 'kubectl get svc --namespace default -w wp-wordpress'

   export SERVICE_IP=$(kubectl get svc --namespace default wp-wordpress --template "{{ range (index .status.loadB
alancer.ingress 0) }}{{.}}{{ end }}")
   echo "WordPress URL: http://$SERVICE_IP/"
   echo "WordPress Admin URL: http://$SERVICE_IP/admin"

2. Open a browser and access WordPress using the obtained URL.

3. Login with the following credentials below to see your blog:

  echo Username: user
  echo Password: $(kubectl get secret --namespace default wp-wordpress -o jsonpath="{.data.wordpress-password}" |
 base64 --decode)
user@Azure:~/Hands-On-Kubernetes-on-Azure/Chapter05$ echo Username: user
Username: user
user@Azure:~/Hands-On-Kubernetes-on-Azure/Chapter05$ echo Password: $(kubectl get secret --namespace default wp-w
ordpress -o jsonpath="{.data.wordpress-password}" | base64 --decode)
Password: 1oV0FVacNF
```

Figure 5.22: Getting the username and password for the WordPress application

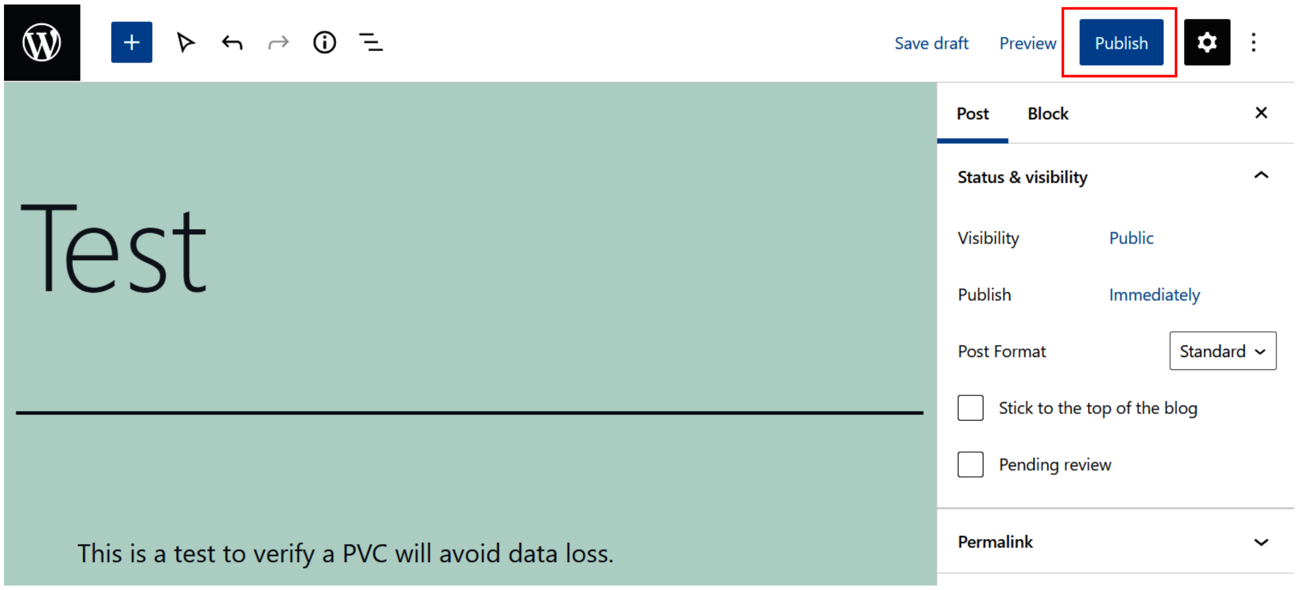
You can log in to our site via the following address: `http://<external-ip>/admin`. Log in here with the credentials from the previous step. Then you can go ahead and add a post to your website. Click the **Write your first blog post** button, and then create a short post, as shown in *Figure 5.23*:



Welcome to WordPress!
We've assembled some links to get you started:

Figure 5.23: Writing your first blog post

Type some text now and hit the **Publish** button, as shown in *Figure 5.24*. The text itself isn't important; you are writing this to verify that data is indeed persisted to disk:



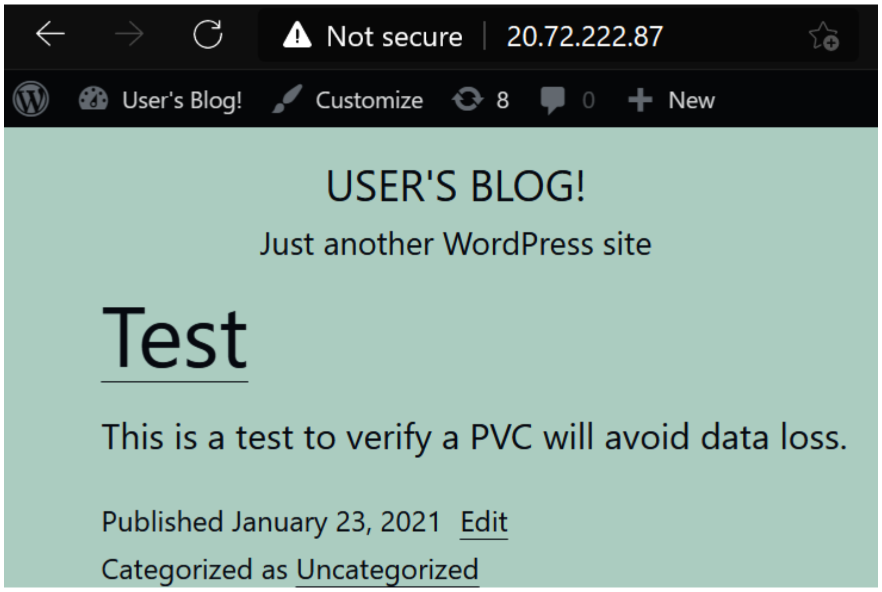
Test

This is a test to verify a PVC will avoid data loss.

Visibility Public
Publish Immediately
Post Format Standard

Figure 5.24: Publishing a post with random text

If you now head over to the main page of your website at `http://<external-ip>`, you'll see your test post as shown in *Figure* 5.25:



USER'S BLOG!

Just another WordPress site

Test

This is a test to verify a PVC will avoid data loss.

Published January 23, 2021 Edit

Categorized as Uncategorized

Figure 5.25: The published blog post appears on the home page

In this section, you deployed a WordPress site, you logged in to your WordPress site, and you created a post. You will verify whether this post survives a node failure in the next section.

Handling pod failure with PVC involvement

The first test you'll do with the PVCs is to kill the pods and verify whether the data has indeed persisted. To do this, let's do two things:

1. **Watch the pods in your application**: To do this, use the current Cloud Shell and execute the following command:

   ```
   kubectl get pods -w
   ```

2. **Kill the two pods that have the PVC mounted**: To do this, create a new Cloud Shell window by clicking on the icon shown in *Figure* 5.26:

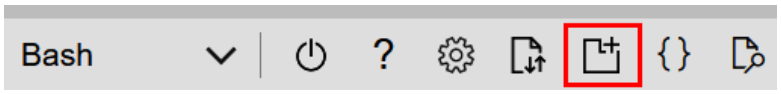
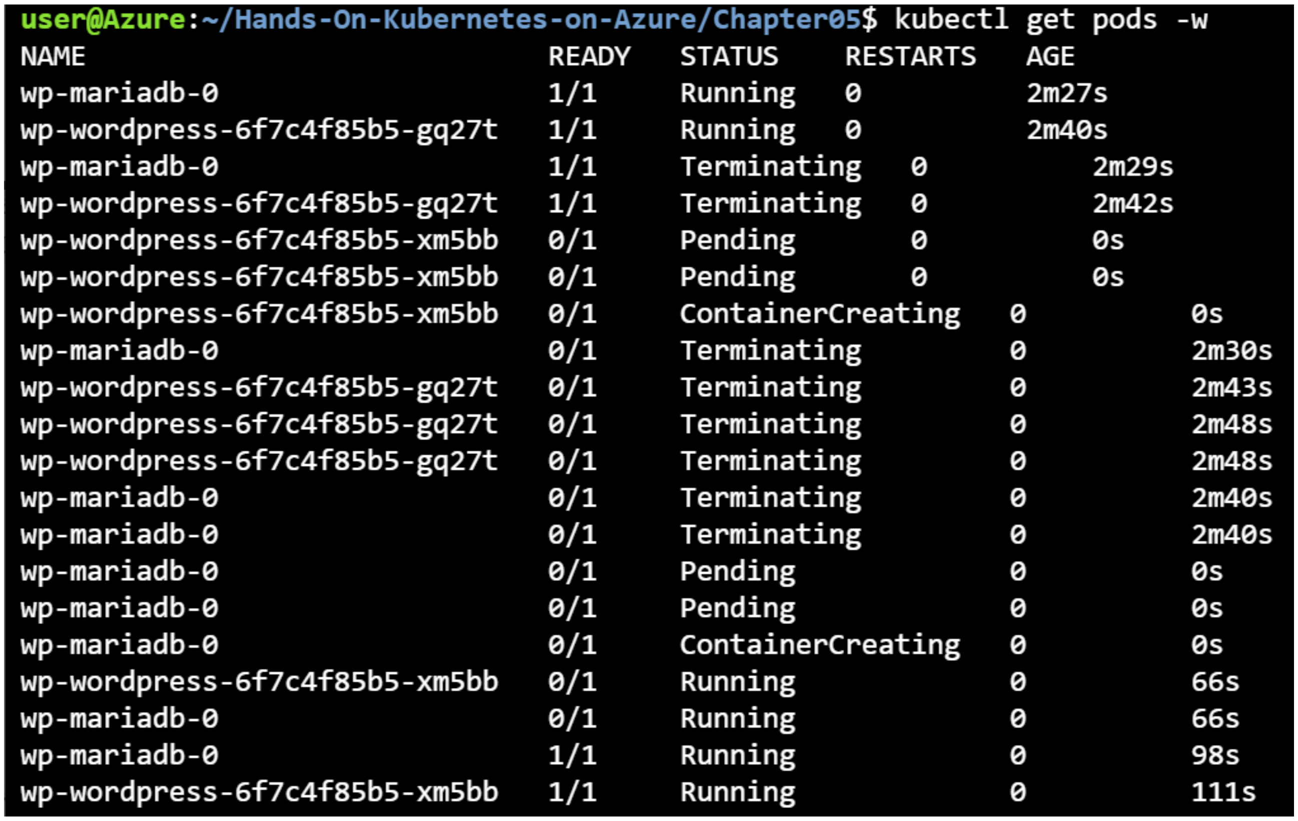


Figure 5.26: Opening a new Cloud Shell instance

Once you open a new Cloud Shell, execute the following command:

```
kubectl delete pod --all
```

In the original Cloud Shell, follow along with the `watch` command that you executed earlier. You should see an output like what's shown in *Figure 5.27*:



```
user@Azure:~/Hands-On-Kubernetes-on-Azure/Chapter05$ kubectl get pods -w
NAME                              READY   STATUS    RESTARTS   AGE
wp-mariadb-0                      1/1     Running   0          2m27s
wp-wordpress-6f7c4f85b5-gq27t     1/1     Running   0          2m40s
wp-mariadb-0                      1/1     Terminating   0          2m29s
wp-wordpress-6f7c4f85b5-gq27t     1/1     Terminating   0          2m42s
wp-wordpress-6f7c4f85b5-xm5bb     0/1     Pending       0          0s
wp-wordpress-6f7c4f85b5-xm5bb     0/1     Pending       0          0s
wp-wordpress-6f7c4f85b5-xm5bb     0/1     ContainerCreating   0          0s
wp-mariadb-0                      0/1     Terminating         0          2m30s
wp-wordpress-6f7c4f85b5-gq27t     0/1     Terminating         0          2m43s
wp-wordpress-6f7c4f85b5-gq27t     0/1     Terminating         0          2m48s
wp-wordpress-6f7c4f85b5-gq27t     0/1     Terminating         0          2m48s
wp-mariadb-0                      0/1     Terminating         0          2m40s
wp-mariadb-0                      0/1     Terminating         0          2m40s
wp-mariadb-0                      0/1     Pending             0          0s
wp-mariadb-0                      0/1     Pending             0          0s
wp-mariadb-0                      0/1     ContainerCreating   0          0s
wp-wordpress-6f7c4f85b5-xm5bb     0/1     Running             0          66s
wp-mariadb-0                      0/1     Running             0          66s
wp-mariadb-0                      1/1     Running             0          98s
wp-wordpress-6f7c4f85b5-xm5bb     1/1     Running             0          111s
```

Figure 5.27: After deleting the pods, Kubernetes will automatically recreate both pods

As you can see, the two original pods went into a **Terminating** state. Kubernetes quickly started creating new pods to recover from the pod outage. The pods went through a similar life cycle as the original ones, going from **Pending** to **ContainerCreating** to **Running**.

3. If you head on over to your website, you should see that your demo post has been persisted. This is how PVCs can help you prevent data loss, as they persist data that would not have been persisted in the pod itself.

In this section, you've learned how PVCs can help when pods get recreated on the same node. In the next section, you'll see how PVCs are used when a node has a failure.

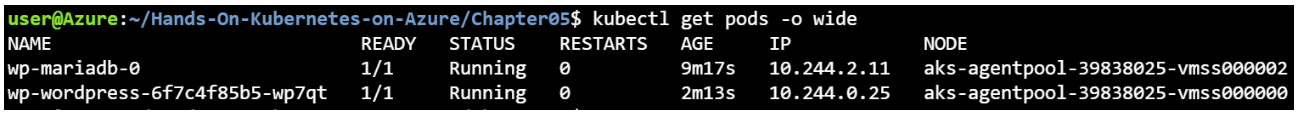
Handling node failure with PVC involvement

In the previous example, you saw how Kubernetes can handle pod failures when those pods have a PV attached. In this example, you'll learn how Kubernetes handles node failures when a volume is attached:

1. Let's first check which node is hosting your application, using the following command:

   ```
   kubectl get pods -o wide
   ```

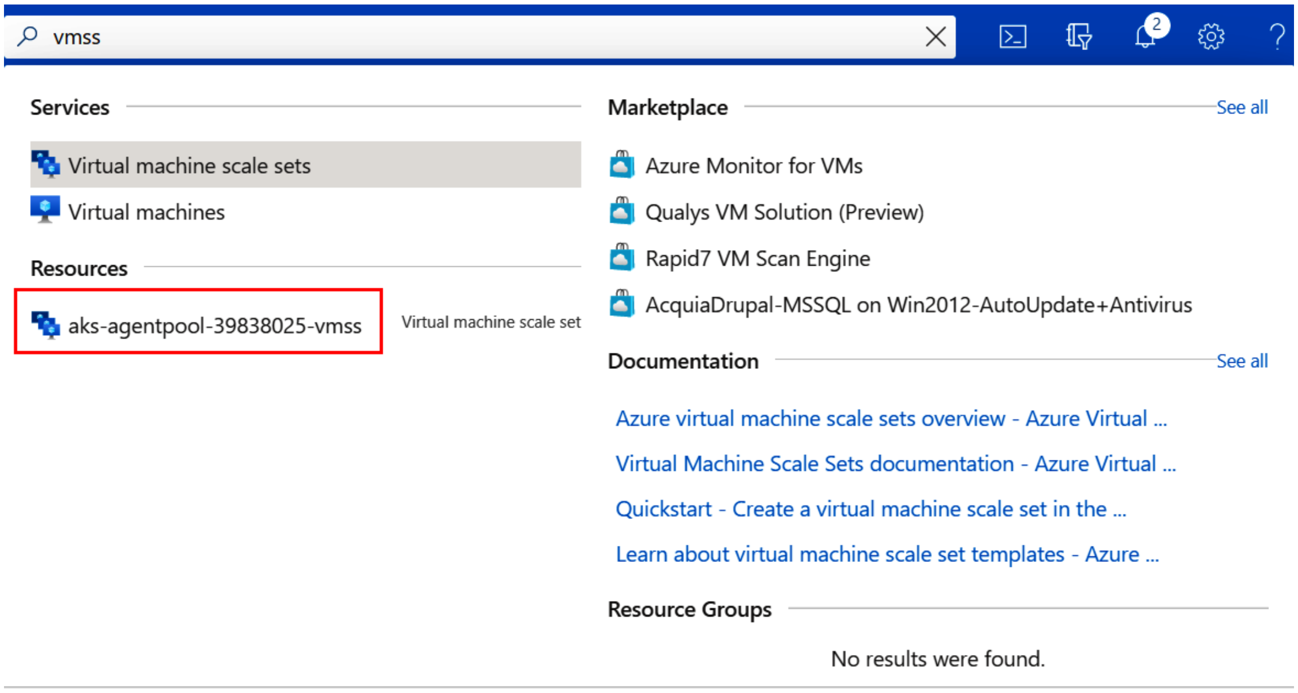
   In the example shown in *Figure* 5.28, node **2** was hosting **MariaDB**, and node **0** was hosting the **WordPress** site:



```
user@Azure:~/Hands-On-Kubernetes-on-Azure/Chapter05$ kubectl get pods -o wide
NAME                            READY   STATUS    RESTARTS   AGE     IP            NODE
wp-mariadb-0                    1/1     Running   0          9m17s   10.244.2.11   aks-agentpool-39838025-vmss000002
wp-wordpress-6f7c4f85b5-wp7qt   1/1     Running   0          2m13s   10.244.0.25   aks-agentpool-39838025-vmss000000
```

Figure 5.28: Check which node hosts the WordPress site

2. Introduce a failure and stop the node that is hosting the **WordPress** pod using the Azure portal. You can do this in the same way as in the earlier example. First, look for the scale set backing your cluster, as shown in *Figure* 5.29:



vmss

Services
Virtual machine scale sets
Virtual machines

Resources
aks-agentpool-39838025-vmss   Virtual machine scale set

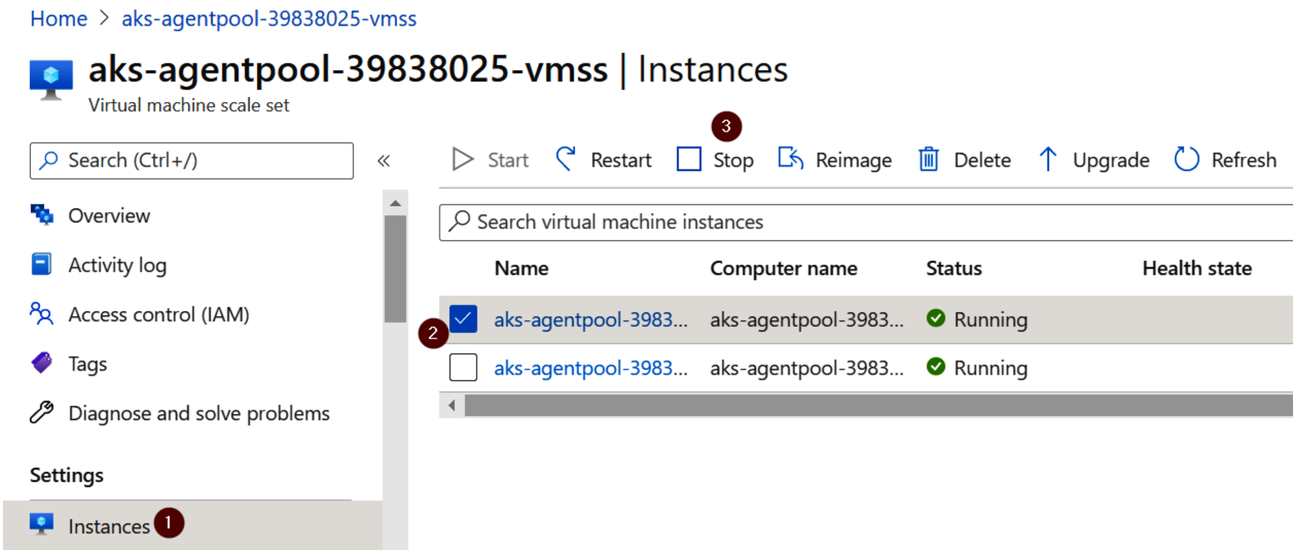
Marketplace   See all
Azure Monitor for VMs
Qualys VM Solution (Preview)
Rapid7 VM Scan Engine
AcquiaDrupal-MSSQL on Win2012-AutoUpdate+Antivirus

Documentation   See all
Azure virtual machine scale sets overview - Azure Virtual ...
Virtual Machine Scale Sets documentation - Azure Virtual ...
Quickstart - Create a virtual machine scale set in the ...
Learn about virtual machine scale set templates - Azure ...

Resource Groups
No results were found.

Figure 5.29: Looking for the scale set hosting your cluster

3. Then shut down the node, by clicking on **Instances** in the left-hand menu, then selecting the node you need to shut down and clicking the **Stop** button, as shown in *Figure 5.30*:


Home > aks-agentpool-39838025-vmss
aks-agentpool-39838025-vmss | Instances
Virtual machine scale set
Search (Ctrl+/)
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Settings
Instances
1
Start
Restart
3
Stop
Reimage
Delete
Upgrade
Refresh
Search virtual machine instances
2

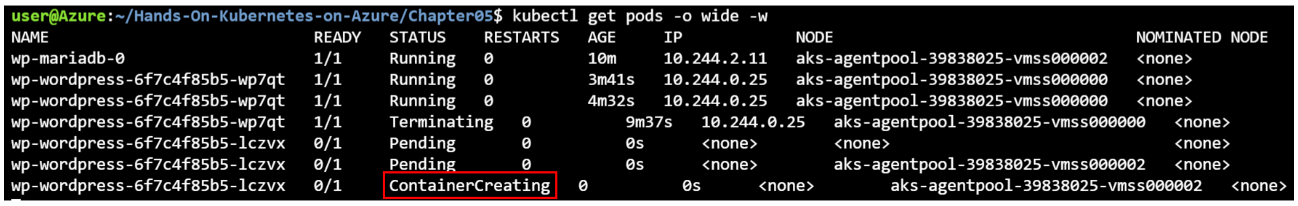
Name	Computer name	Status	Health state
✓ aks-agentpool-3983...	aks-agentpool-3983...	Running	
aks-agentpool-3983...	aks-agentpool-3983...	Running	

Figure 5.30: Shutting down the node

4. After this action, once again, watch the pods to see what is happening in the cluster:

```
kubectl get pods -o wide -w
```

As in the previous example, it is going to take 5 minutes before Kubernetes will start taking action against the failed node. You can see that happening in *Figure 5.31*:



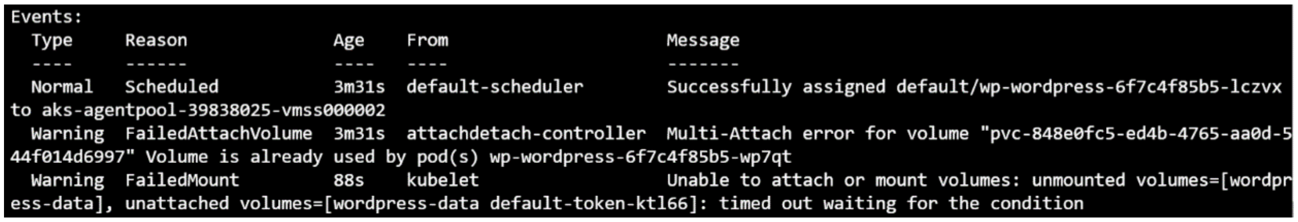
```
user@Azure:~/Hands-On-Kubernetes-on-Azure/Chapter05$ kubectl get pods -o wide -w
NAME                          READY   STATUS    RESTARTS   AGE     IP            NODE                                NOMINATED NODE
wp-mariadb-0                  1/1     Running   0          10m     10.244.2.11   aks-agentpool-39838025-vmss000002   <none>
wp-wordpress-6f7c4f85b5-wp7qt 1/1     Running   0          3m41s   10.244.0.25   aks-agentpool-39838025-vmss000000   <none>
wp-wordpress-6f7c4f85b5-wp7qt 1/1     Running   0          4m32s   10.244.0.25   aks-agentpool-39838025-vmss000000   <none>
wp-wordpress-6f7c4f85b5-wp7qt 1/1     Terminating   0          9m37s   10.244.0.25   aks-agentpool-39838025-vmss000000   <none>
wp-wordpress-6f7c4f85b5-lczvx 0/1     Pending       0          0s      <none>        <none>                              <none>
wp-wordpress-6f7c4f85b5-lczvx 0/1     Pending       0          0s      <none>        aks-agentpool-39838025-vmss000002   <none>
wp-wordpress-6f7c4f85b5-lczvx 0/1     ContainerCreating   0          0s      <none>        aks-agentpool-39838025-vmss000002   <none>
```

Figure 5.31: A pod in a ContainerCreating state

5. You are seeing a new issue here. The new pod is stuck in a **ContainerCreating** state. Let's figure out what is happening here. First, describe that pod:

```
kubectl describe pods/wp-wordpress-<pod-id>
```

You will get an output as shown in *Figure* 5.32:



```
Events:
  Type     Reason              Age    From                      Message
  ----     ------              ----   ----                      -------
  Normal   Scheduled           3m31s  default-scheduler         Successfully assigned default/wp-wordpress-6f7c4f85b5-lczvx
to aks-agentpool-39838025-vmss000002
  Warning  FailedAttachVolume  3m31s  attachdetach-controller  Multi-Attach error for volume "pvc-848e0fc5-ed4b-4765-aa0d-5
44f014d6997" Volume is already used by pod(s) wp-wordpress-6f7c4f85b5-wp7qt
  Warning  FailedMount         88s    kubelet                   Unable to attach or mount volumes: unmounted volumes=[wordpr
ess-data], unattached volumes=[wordpress-data default-token-ktl66]: timed out waiting for the condition
```

Figure 5.32: Output explaining why the pod is in a ContainerCreating state

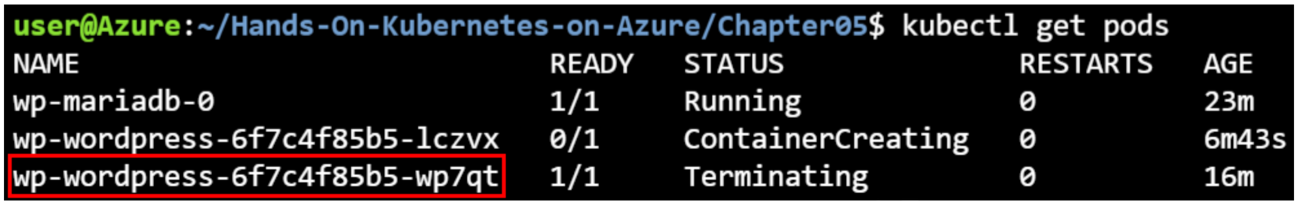
This tells you that there is a problem with the volume. You see two errors related to that volume: the `FailedAttachVolume` error explains that the volume is already used by another pod, and `FailedMount` explains that the current pod cannot mount the volume. You can solve this by manually forcefully removing the old pod stuck in the `Terminating` state.

> **Note**
>
> The behavior of the pod stuck in the `Terminating` state is not a bug. This is default Kubernetes behavior. The Kubernetes documentation states the following: *"Kubernetes (versions 1.5 or newer) will not delete pods just because a Node is unreachable. The pods running on an unreachable Node enter the Terminating or Unknown state after a timeout. Pods may also enter these states when the user attempts the graceful deletion of a pod on an unreachable Node."* You can read more at https://kubernetes.io/docs/tasks/run-application/force-delete-stateful-set-pod/.

6. To forcefully remove the terminating pod from the cluster, get the full pod name using the following command:

```
kubectl get pods
```

This will show you an output similar to *Figure* 5.33:



```
user@Azure:~/Hands-On-Kubernetes-on-Azure/Chapter05$ kubectl get pods
NAME                            READY   STATUS              RESTARTS   AGE
wp-mariadb-0                    1/1     Running             0          23m
wp-wordpress-6f7c4f85b5-lczvx   0/1     ContainerCreating   0          6m43s
wp-wordpress-6f7c4f85b5-wp7qt   1/1     Terminating         0          16m
```

Figure 5.33: Getting the name of the pod stuck in the Terminating state

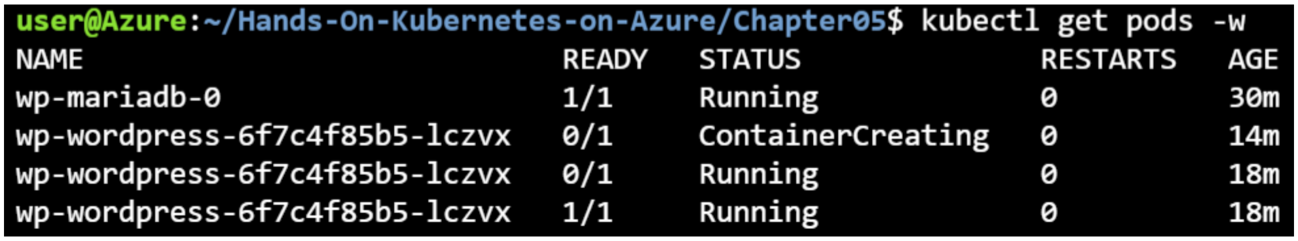
7. Use the pod's name to force the deletion of this pod:

   ```
   kubectl delete pod wordpress-wp-<pod-id> --force
   ```

8. After the pod has been deleted, it will take a couple of minutes for the other pod to enter a Running state. You can monitor the state of the pod using the following command:

   ```
   kubectl get pods -w
   ```

   This will return an output similar to *Figure* 5.34:



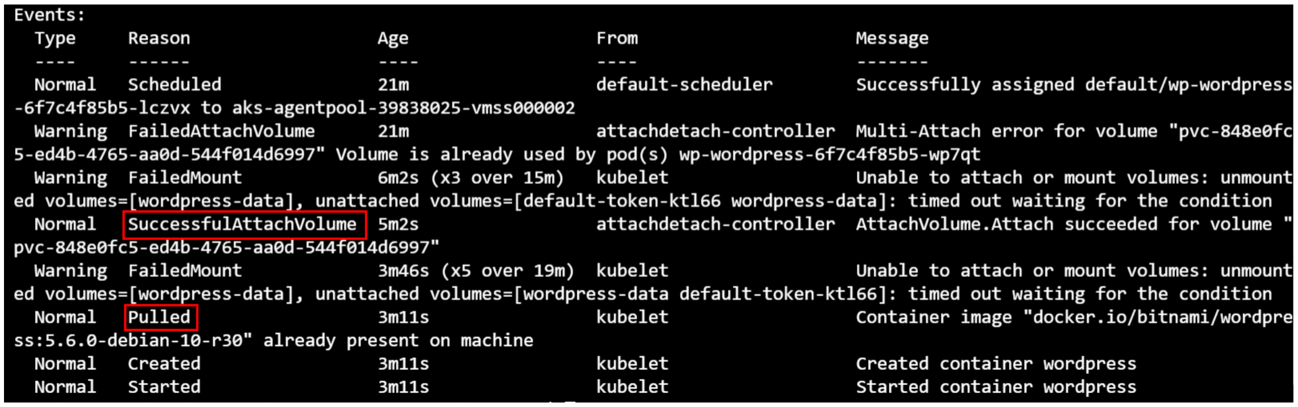
```
user@Azure:~/Hands-On-Kubernetes-on-Azure/Chapter05$ kubectl get pods -w
NAME                             READY   STATUS              RESTARTS   AGE
wp-mariadb-0                     1/1     Running             0          30m
wp-wordpress-6f7c4f85b5-lczvx    0/1     ContainerCreating   0          14m
wp-wordpress-6f7c4f85b5-lczvx    0/1     Running             0          18m
wp-wordpress-6f7c4f85b5-lczvx    1/1     Running             0          18m
```

Figure 5.34: The new WordPress pod returning to a Running state

9. As you can see, this brought the new pod to a healthy state. It did take a couple of minutes for the system to pick up the changes and then mount the volume to the new pod. Let's get the details of the pod again using the following command:

   ```
   kubectl describe pod wp-wordpress-<pod-id>
   ```

   This will generate an output as follows:



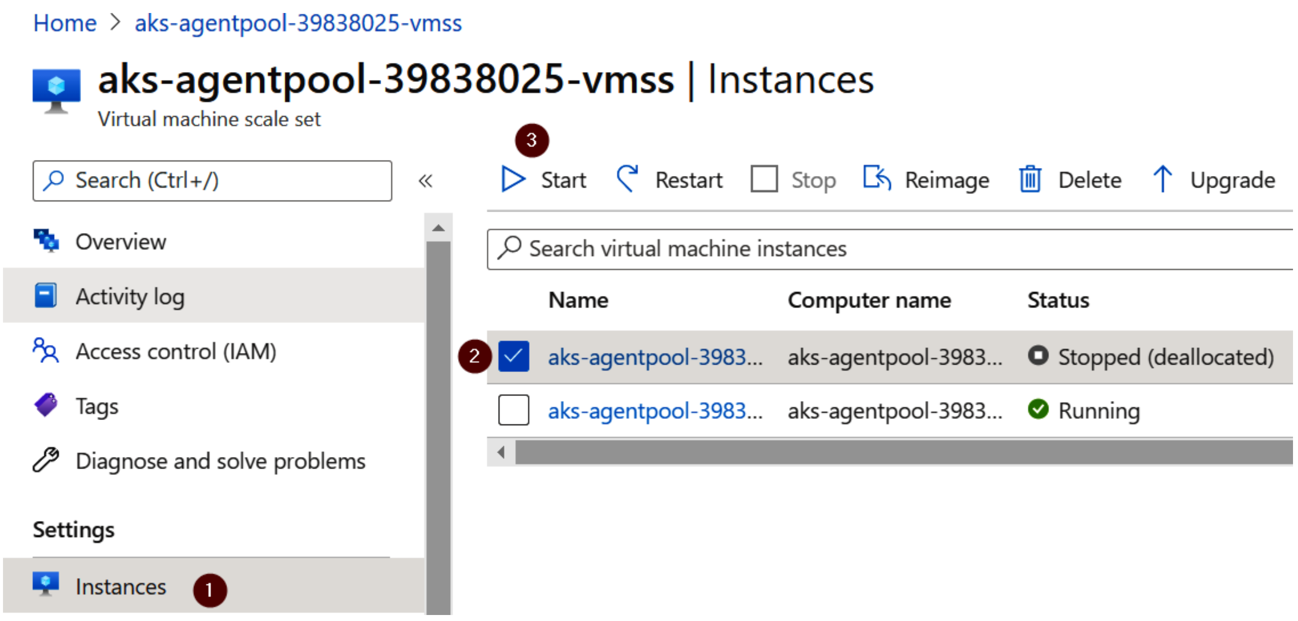
```
Events:
  Type     Reason                  Age                 From                      Message
  ----     ------                  ----                ----                      -------
  Normal   Scheduled               21m                 default-scheduler         Successfully assigned default/wp-wordpress
-6f7c4f85b5-lczvx to aks-agentpool-39838025-vmss000002
  Warning  FailedAttachVolume      21m                 attachdetach-controller   Multi-Attach error for volume "pvc-848e0fc
5-ed4b-4765-aa0d-544f014d6997" Volume is already used by pod(s) wp-wordpress-6f7c4f85b5-wp7qt
  Warning  FailedMount             6m2s (x3 over 15m)  kubelet                   Unable to attach or mount volumes: unmount
ed volumes=[wordpress-data], unattached volumes=[default-token-ktl66 wordpress-data]: timed out waiting for the condition
  Normal   SuccessfulAttachVolume  5m2s                attachdetach-controller   AttachVolume.Attach succeeded for volume "
pvc-848e0fc5-ed4b-4765-aa0d-544f014d6997"
  Warning  FailedMount             3m46s (x5 over 19m)  kubelet                  Unable to attach or mount volumes: unmount
ed volumes=[wordpress-data], unattached volumes=[wordpress-data default-token-ktl66]: timed out waiting for the condition
  Normal   Pulled                  3m11s               kubelet                   Container image "docker.io/bitnami/wordpre
ss:5.6.0-debian-10-r30" already present on machine
  Normal   Created                 3m11s               kubelet                   Created container wordpress
  Normal   Started                 3m11s               kubelet                   Started container wordpress
```

Figure 5.35: The new pod is now attaching the volume and pulling the container image

10. This shows you that the new pod successfully got the volume attached and that the container image got pulled. This also made your WordPress website available again, which you can verify by browsing to the public IP. Before continuing to the next chapter, clean up the application using the following command:

    ```
    helm delete wp
    kubectl delete pvc --all
    kubectl delete pv --all
    ```

11. Let's also start the node that was shut down: go back to the scale set pane in the Azure portal, click **Instances** in the left-hand menu, select the node you need to start, and click on the **Start** button, as shown in *Figure* 5.36:


Home > aks-agentpool-39838025-vmss
aks-agentpool-39838025-vmss | Instances
Virtual machine scale set
Search (Ctrl+/)
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Settings
Instances
1
2
3
Start
Restart
Stop
Reimage
Delete
Upgrade
Search virtual machine instances

	Name	Computer name	Status
✓	aks-agentpool-3983...	aks-agentpool-3983...	Stopped (deallocated)
	aks-agentpool-3983...	aks-agentpool-3983...	Running

Figure 5.36: Starting node 0 again

In this section, you learned how you can recover from a node failure when PVCs aren't mounting to new pods. All you needed to do was forcefully delete the pod that was stuck in the `Terminating` state.

Summary

In this chapter, you learned about common Kubernetes failure modes and how you can recover from them. This chapter started with an example of how Kubernetes automatically detects node failures and how it will start new pods to recover the workload. After that, you scaled out your workload and had your cluster run out of resources. You recovered from that situation by starting the failed node again to add new resources to the cluster.

Next, you saw how PVs are useful to store data outside of a pod. You deleted all pods on the cluster and saw how the PV ensured that no data was lost in your application. In the final example in this chapter, you saw how you can recover from a node failure when PVs are attached. You were able to recover the workload by forcefully deleting the terminating pod. This brought your workload back to a healthy state.

This chapter has explained common failure modes in Kubernetes. In the next chapter, we will introduce HTTPS support to our services and introduce authentication with Azure Active Directory.