Getting Started with Volumes

In this lesson, you will be introduced to Kubernetes Volumes and create a Minikube cluster.

WE'LL COVER THE FOLLOWING

State Preservation

> Having a system without a state is impossible. Even though there is a tendency to develop stateless applications, we still need to deal with the state. There are databases and other stateful third-party applications. No matter what we do, we need to make sure that the state is preserved no matter what happens to containers, Pods, or even whole nodes.

Most of the time, stateful applications store their state on disk. That leaves us with a problem. If a container crashes, `kubelet` will restart it. The problem is that it will create a new container based on the same image. All data accumulated inside a container that crashed will be lost.

The Volumes

Kubernetes Volumes solve the need to preserve the state across container crashes. In essence, Volumes are references to files and directories made accessible to containers that form a Pod. The significant difference between different types of Kubernetes Volumes is in the way these files and directories are created.

While the primary use-case for Volumes is the preservation of state, there are quite a few others. For example, we might use Volumes to access Docker's socket running on a host. Or we might use them to access configuration residing in a file on the host file system.

> We can describe Volumes as a way to access a file system that might be running on the same host or somewhere else. No matter where that file system is, it is external to the containers that mount volumes. There can be many reasons why someone might mount a Volume, with state preservation being only one of them.

There are over **twenty-five** Volume types supported by Kubernetes. It would take us too much time to go through all of them. Besides, even if we'd like to do that, many Volume types are specific to a hosting vendor. For example, `awsElasticBlockStore` works only with AWS, `azureDisk` and `azureFile` work only with Azure, and so on and so forth.

We'll limit our exploration to Volume types that can be used within Minikube. You should be able to extrapolate that knowledge to Volume types applicable to your hosting vendor of choice.

Let's get down to it.

> ℹ All the commands from this chapter are available in the 08-volume.sh Gist.

Copying the Files

This time, we'll have an additional action we'll execute in preparation to create a Minikube cluster.

```
cd k8s-specs

git pull

cp volume/prometheus-conf.yml \
    ~/.minikube/files
```

We'll need the `volume/prometheus-conf.yml` file inside the soon-to-be-created

Minikube VM. When it starts, it will copy all the files from `~/.minikube/files`

on your host, into the `/files` directory in the VM.

> ⚠ Depending on your operating system, the `~/.minikube/files` directory might be somewhere else. If that's the case, please adapt the command above.

Creating a Cluster

Now that the files are copied to the shared directory, we can repeat the same process we did quite a few times before.

> 📝 Please note that we've added the step from the last chapter that enables the ingress addon.

```
minikube start --vm-driver=virtualbox
minikube addons enable ingress
kubectl config current-context
```

Our Minikube cluster is up and running.

Now that the Minikube cluster is up and running, we can explore the first Volume type in the next lesson.