Defining Pods through Declarative Syntax

In this lesson, we will create and run Pods using declarative syntax.

WE'LL COVER THE FOLLOWING

- ^
- Defining Pods Through Declarative Syntax
- Looking into a Pod's Definition

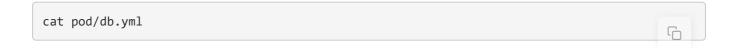
Defining Pods Through Declarative Syntax

Even though a Pod can contain any number of containers, the most common use case is to use the **single-container-in-a-Pod** model. In such a case, a Pod is a *wrapper* around one container. From Kubernetes' perspective, a Pod is the smallest unit.

We **cannot** tell Kubernetes to run a container. Instead, we ask it to create a Pod that wraps around a container.

Looking into a Pod's Definition

Let's take a look at a simple Pod definition by accessing the db.yml file from the cloned git repository.



The **output** is as follows.

```
apiVersion: v1
kind: Pod
metadata:
  name: db
  labels:
    type: db
    vendor: MongoLabs
spec:
  containers:
```

```
- name: db
  image: mongo:3.3

command: ["mongod"]
  args: ["--rest", "--httpinterface"]
```

Let's analyze the various sections in the output definition of a Pod.

- Line 1-2: We're using v1 of Kubernetes Pods API. Both apiVersion and kind are mandatory. That way, Kubernetes knows what we want to do (create a Pod) and which API version to use.
- Line 3-7: The next section is metadata. It provides information that does not influence how the Pod behaves. We used metadata to define the name of the Pod (db) and a few labels. Later on, when we move into *Controllers*, labels will have a practical purpose. For now, they are purely informational.
- Line 8: The last section is the spec in which we defined a single container. As you might have guessed, we can have multiple containers defined as a Pod. Otherwise, the section would be written in singular (container without s). We'll explore multi-container Pods later.
- Line 12: In our case, the container is defined with the name (db), the image (mongo), the command that should be executed when the container starts (mongod)
- **Line 13:** Finally, the set of arguments. The arguments are defined as an array with, in this case, two elements (--rest and --httpinterface).

We won't go into details of everything you can use to define a Pod. Throughout the course, you'll see quite a few other commonly (and not so commonly) used things we should define in Pods. Later on, when you decide to learn all the possible arguments you can apply, explore the official, and ever-changing, Pod v1 core documentation.

Let's create the Pod defined in the db.yml file.

```
kubectl create -f pod/db.yml
```

You'll notice that we did not need to specify pod in the command. The

command will create the kind of resource defined in the pod/db.yml file. Later

on, you'll see that a single YAML file can contain definitions of multiple resources.

Let's take a look at the Pods in the cluster.

```
kubectl get pods
```

The **output** is as follows.

```
NAME READY STATUS RESTARTS AGE
db 1/1 Running 0 11s
```

Our Pod named db is up and running!

In some cases, you might want to retrieve a bit more information by specifying wide output.

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

The **output** is as follows.

```
NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES db 1/1 Running 0 1m 172.17.0.4 minikube <none>
```

As you can see, we got two additional columns; the IP and the NODE.

If you'd like to parse the output, using json format is probably the best option.

```
kubectl get pods -o json
```

The output is too big to be presented here, especially since we won't go through all the information provided through the json output format.

When we want more information than provided with the default output, but still in a format that is human-friendly, yaml output is probably the best choice.

```
kubectl get pods -o yaml
```

Just as with the json output, we won't go into details of everything we got from Kubernetes. With time, you'll become familiar with all the information related to Pods. For now, we want to focus on the most important aspects.

Let's introduce a new kubectl sub-command.

```
kubectl describe pod db
```

The describe sub-command returned details of the specified resource. In this case, the resource is the Pod named db.

The output is too big for us to go into every detail. Besides, most of it should be self-explanatory if you're familiar with containers. Instead, we'll briefly comment on the last section called events.

```
Events:
 Type
       Reason
                               Age
                                    From
                                                       Message
 Normal Scheduled
                                    default-scheduler Successfully assigned db to minikut
                               2m
 Normal SuccessfulMountVolume 2m
                                    kubelet, minikube MountVolume.SetUp succeeded for vol
 Normal Pulling
                               2m
                                    kubelet, minikube pulling image "mongo:3.3"
                                    kubelet, minikube Successfully pulled image "mongo:3.
 Normal Pulled
                               2m
 Normal Created
                               2m
                                    kubelet, minikube Created container
 Normal Started
                                    kubelet, minikube Started container
```

The above output may appear a bit different and take some time to show up in its full form.

We can see that Pod was created. Even though the process was simple from a user's perspective, quite a few things happened in the background.

In the next lesson, we will go through the stages involved in a Pod's creation.