

Running Multiple Containers in a Single Pod

In this lesson, we will learn how to run a Pod having multiple containers.

WE'LL COVER THE FOLLOWING



- Anatomy of a Pod
- Formatting the Output
- Executing Commands Inside the Pod

Anatomy of a Pod

- **Pods** are designed to run multiple cooperative processes that should act as a cohesive unit. Those processes are wrapped in containers.
- All the containers that form a Pod are running on the same machine. A Pod cannot be split across multiple nodes.
- All the processes (containers) inside a Pod share the same set of resources, and they can communicate with each other through `localhost`. One of those shared resources is storage.
- A volume (think of it as a directory with shareable data) defined in a Pod can be accessed by all the containers thus allowing them all to share the same data.

We'll explore storage and volumes in more depth later on. For now, let's take a look at the `pod/go-demo-2.yml` specification.

```
cat pod/go-demo-2.yml
```



The **output** is as follows.

```
apiVersion: v1
```



```
kind: Pod
metadata:
  name: go-demo-2

  labels:
    type: stack
spec:
  containers:
    - name: db
      image: mongo:3.3
    - name: api
      image: vfarctic/go-demo-2
      env:
        - name: DB
          value: localhost
```

The YAML file defines a Pod with two containers named `db` and `api`. The service inside the `vfarctic/go-demo-2` image uses environment variable `DB` to know where the database is. The value is `localhost` since all the containers in the same Pod are reachable through it. Let's create the Pod.

```
kubectl create -f pod/go-demo-2.yml
kubectl get -f pod/go-demo-2.yml
```

We created a new Pod defined in the `go-demo-2.yml` file and retrieved its information from Kubernetes. The **output** of the latter command is as follows.

NAME	READY	STATUS	RESTARTS	AGE
go-demo-2	2/2	Running	0	2m

We can see from the `READY` column that, this time, the Pod has two containers (`2/2`).

This might be an excellent opportunity to introduce formatting to retrieve specific information.

Formatting the Output

Let's say that we want to retrieve the names of the containers in a Pod. The first thing we'd have to do is get familiar with Kubernetes API. We can do that by going to [Pod v1 core](#) documentation. While reading the documentation will become mandatory sooner or later, we'll use a simpler route and inspect the output from Kubernetes.

```
kubectl get -f pod/go-demo-2.yml -o json
```

The output is too big to be presented here, so we'll focus on the task at hand. We need to retrieve the names of the containers in the Pod. Therefore, the part of the **output** we're looking for is as follows.

```
{
  ...
  "spec": {
    "containers": [
      {
        ...
        "name": "db",
        ...
      },
      {
        ...
        "name": "api",
        ...
      }
    ],
    ...
  },
  ...
}
```

The **get** command that would filter the output and retrieve only the names of the containers is as follows.

```
kubectl get -f pod/go-demo-2.yml \
  -o jsonpath="{.spec.containers[*].name}"
```

The **output** is as follows.

```
db api
```

We used **jsonpath** as the output format and specified that we want to retrieve names of all the **containers** from the **spec**. The ability to filter and format information might not look that important right now but, once we move into more complex scenarios, it will prove to be invaluable. That will become especially evident when we try to automate the processes and requests sent to Kubernetes API.


Executing Commands Inside the Pod

How would we execute a command inside the Pod? Unlike the previous examples that did a similar task, this time we have two containers in the Pod, so we need to be more specific.

```
kubectl exec -it -c db go-demo-2 ps aux
```



The **output** should display the processes inside the `db` container. Namely, the `mongod` process.

 The file name `go-demo-2` is from the subsection `name` of the `metadata` section defined inside the `yaml` file.

How about logs from a container? As you might have guessed, we cannot execute something like `kubectl logs go-demo-2` since the Pod hosts multiple containers. Instead, we need to be specific and name the container from which we want to see the logs.

```
kubectl logs go-demo-2 -c db
```



How about scaling? How would we, for example, scale the service so that there are two containers of the API and one container for the database?

One option could be to define two containers in the Pod. Let's take a look at a Pod definition that might accomplish what we need.

```
cat pod/go-demo-2-scaled.yaml
```



The **output** is as follows.

```
apiVersion: v1
kind: Pod
metadata:
  name: go-demo-2
  labels:
    type: stack
spec:
  containers:
  - name: db
    image: mongo:3.3
  - name: api-1
    image: vfarcic/go-demo-2
```



```
env:  
  - name: DB  
    value: localhost  
- name: api-2  
  image: vfarcic/go-demo-2  
  env:  
    - name: DB  
      value: localhost
```

We defined two containers for the API and named them `api-1` and `api-2`. The only thing left is to create the Pod. But, we're not going to do that and the reason is given in the next lesson.

In the next lesson, we will discuss a Pod with a single container vs a Pod with multiple containers inside it.