Exercise 2.3: Create a Basic Pod

1. The smallest unit we directly control with Kubernetes is the pod. We will create a pod by creating a minimal YAML file. First we will get a list of current API objects and their `APIGROUP`. If value is not shown it may not exist, as with `SHORTNAMES`. Note that `pods` does not declare an `APIGROUP`. At the moment this indicates it is part of the stable `v1` group.

```
student@cp:~$ kubectl api-resources
```

```
NAME                SHORTNAMES   APIVERSION   NAMESPACED   KIND
bindings                         v1           true         Binding
componentstatuses   cs           v1           false        ComponentStatus
configmaps          cm           v1           true         ConfigMap
endpoints           ep           v1           true         Endpoints
.....
pods                po           v1           true         Pod
....
```

2. From the output we see most are `v1` which is used to denote a stable object. With that information we will add the other three required sections for pods such as `metadata`, with a `name`, and `spec` which declares which container image to use and a `name` for the container. We will create an eight line YAML file. White space and indentation matters. Don't use **Tab**s. There is a `basic.yaml` file available in the tarball, as well as `basic-later.yaml` which shows what the file will become and can be helpful for figuring out indentation.

YAML **basic.yaml**

```
1 apiVersion: v1
2 kind: Pod
3 metadata:
4   name: basicpod
5 spec:
6   containers:
7   - name: webcont
8     image: nginx
```

3. Create the new pod using the recently created YAML file.

```
student@cp:~$ kubectl create -f basic.yaml
```

```
pod/basicpod created
```

4. Make sure the pod has been created then use the **describe** sub-command to view the details. Among other values in the output you should be about to find the image and the container name.

```
student@cp:~$ kubectl get pod
```

```
NAME       READY   STATUS    RESTARTS   AGE
basicpod   1/1     Running   0          23s
```

```
student@cp:~$ kubectl describe pod basicpod
```

```
Name:               basicpod
Namespace:          default
```

```
Priority:           0
<output_omitted>
```

5. Shut down the pod and verify it is no longer running.

```
student@cp:~$ kubectl delete pod basicpod
```

```
pod "basicpod" deleted
```

```
student@cp:~$ kubectl get pod
```

```
No resources found in default namespace.
```

6. We will now configure the pod to expose port 80. This configuration does not interact with the container to determine what port to open. We have to know what port the process inside the container is using, in this case port 80 as a web server. Add two lines to the end of the file. Line up the indentation with the `image` declaration.

```
student@cp:~$ vim basic.yaml
```

basic.yaml

```
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: basicpod
5  spec:
6    containers:
7    - name: webcont
8      image: nginx
9      ports:                      #<--Add this and following line
10     - containerPort: 80
```

7. Create the pod and verify it is running. Use the `-o wide` option to see the internal IP assigned to the pod, as well as `NOMINATED NODE`, which is used by the scheduler and `READINESS GATES` which show if experimental features are enabled. Using **curl** and the pods IP address you should get the default nginx welcome web page.

```
student@cp:~$ kubectl create -f basic.yaml
```

```
pod/basicpod created
```

```
student@cp:~$ kubectl get pod -o wide
```

```
NAME     READY STATUS  RESTARTS AGE  IP           NODE
  NOMINATED NODE  READINESS GATES
basicpod 1/1   Running 0        9s   192.168.1.3  cp
  <none>          <none>
```

```
student@cp:~$ curl http://192.168.1.3
```

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>

<output_omitted>
```

```
student@cp:~$ kubectl delete pod basicpod
```

```
pod "basicpod" deleted
```

8. We will now create a simple service to expose the pod to other nodes and pods in the cluster. The `service` YAML will have the same four sections as a pod, but different `spec` configuration and the addition of a `selector`. Again, copy of the example from the tarball instead of typing the file by hand.

```
student@cp:~$ vim basicservice.yaml
```

basicservice.yaml

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: basicservice
5  spec:
6    selector:
7      type: webserver
8    ports:
9    - protocol: TCP
10     port: 80
```

9. We will also add a `label` to the pod and a `selector` to the service so it knows which object to communicate with.

```
student@cp:~$ vim basic.yaml
```

basic.yaml

```
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: basicpod
5    labels:                    #<-- Add this line
6      type: webserver          #<-- and this line which matches selector
7  spec:
8  ....
```

10. Create the new pod and service. Verify both have been created. We will learn details of the output in later chapters.

```
student@cp:~$ kubectl create -f basic.yaml
```

```
pod/basicpod created
```

```
student@cp:~$ kubectl create -f basicservice.yaml
```

```
service/basicservice created
```

```
student@cp:~$ kubectl get pod
```

```
NAME       READY   STATUS    RESTARTS   AGE
basicpod   1/1     Running   0          110s
```

```
student@cp:~$ kubectl get svc
```

```
NAME           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)     AGE
basicservice   ClusterIP   10.96.112.50    <none>        80/TCP      14s
```

```
kubernetes    ClusterIP  10.96.0.1       <none>       443/TCP     4h
```

11. Test access to the web server using the CLUSTER-IP for the basicservice.

```
student@cp:~$ curl http://10.96.112.50
```

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>

<output_omitted>
```

12. We will now expose the service to outside the cluster as well. Delete the service, edit the file and add a type declaration.

```
student@cp:~$ kubectl delete svc basicservice
```

```
service "basicservice" deleted
```

```
student@cp:~$  vim basicservice.yaml
```

basicservice.yaml

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4      name: basicservice
5  spec:
6    selector:
7      type: webserver
8    type: NodePort       #<--Add this line
9    ports:
10   - protocol: TCP
11     port: 80
```

13. Create the service again. Note there is a different TYPE and CLUSTER-IP and also a high-numbered port.

```
student@cp:~$  kubectl create -f basicservice.yaml
```

```
service/basicservice created
```

```
student@cp:~$ kubectl get svc
```

```
NAME         TYPE       CLUSTER-IP      EXTERNAL-IP PORT(S)         AGE
basicservice NodePort   10.100.139.155 <none>      80:31514/TCP   3s
kubernetes   ClusterIP  10.96.0.1       <none>      443/TCP         47h
```

14. Using the public IP address of the node and the high port you should be able to test access to the webserver. In the example below the public IP is 35.238.3.83, as reported by a **curl** to ifconfig.io. Your IP will be different. The high port will also probably be different. Note that testing from within a GCE or AWS node will not work. Use a local to you terminal or web browser to test.

```
student@cp:~$ curl ifconfig.io
```

```
35.238.3.83
```

```
local$ curl http://35.238.3.83:31514
```

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
<output_omitted>
```