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@ckad-1:~$ kubectl api-resources
   ```

   ```
   1  NAME               SHORTNAMES  APIGROUP    NAMESPACED    KIND
   2  bindings                                   true          Binding
   3  componentstatuses  cs                      false         ComponentStatus
   4  configmaps         cm                      true          ConfigMap
   5  endpoints          ep                      true          Endpoints
   6  .....
   7  pods               po                      true          Pod
   8  ....
   ```

2. Finding no declared `APIGROUP` we will use `v1` to denote a stable object. With that information we will add the other three required sections such as `metadata`, with a `name`, and `spec` which declares which **Docker** 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.

   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@ckad-1:~$ kubectl create -f basic.yaml
   ```

   ```
   1  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@ckad-1:~$ kubectl get pod
   ```

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

   ```
   student@ckad-1:~$ kubectl describe pod basicpod
   ```

```
1  Name:                 basicpod
2  Namespace:            default
3  Priority:             0
4  <output_omitted>
```

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

   student@ckad-1:~$ kubectl delete pod basicpod

```
1  pod "basicpod" deleted
```

   student@ckad-1:~$ kubectl get pod

```
1  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@ckad-1:~$ 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@ckad-1:~$ kubectl create -f basic.yaml

```
1  pod/basicpod created
```

   student@ckad-1:~$ kubectl get pod -o wide

```
1  NAME      READY STATUS  RESTARTS AGE   IP            NODE
2    NOMINATED NODE  READINESS GATES
3  basicpod 1/1   Running 0        9s    192.168.1.3   ckad-1
4    <none>          <none>
```

   student@ckad-1:~$ curl http://192.168.1.3

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>Welcome to nginx!</title>
5
6  <output_omitted>
```

THE
LINUX
FOUNDATION

student@ckad-1:~$ kubectl delete pod basicpod

```
1  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`.

student@ckad-1:~$ 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@ckad-1:~$ 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.

student@ckad-1:~$ kubectl create -f basic.yaml

```
1  pod/basicpod created
```

student@ckad-1:~$ kubectl create -f basicservice.yaml

```
1  service/basicservice created
```

student@ckad-1:~$ kubectl get pod

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

student@ckad-1:~$ kubectl get svc

THE LINUX FOUNDATION

```
1  NAME          TYPE        CLUSTER-IP      EXTERNAL-IP  PORT(S)    AGE
2  basicservice  ClusterIP   10.96.112.50    <none>       80/TCP     14s
3  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@ckad-1:~$ curl http://10.96.112.50

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>Welcome to nginx!</title>
5
6  <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@ckad-1:~$ kubectl delete svc basicservice

```
1  service "basicservice" deleted
```

    student@ckad-1:~$ 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@ckad-1:~$ kubectl create -f basicservice.yaml

```
1  service/basicservice created
```

    student@ckad-1:~$ kubectl get svc

```
1  NAME          TYPE        CLUSTER-IP      EXTERNAL-IP  PORT(S)        AGE
2  basicservice  NodePort    10.100.139.155  <none>       80:31514/TCP   3s
3  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`, yours 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.

    local$ curl http://35.238.3.83:31514

                                ****

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>Welcome to nginx!</title>
5  <style>
6  <output_omitted>
```

THE
LINUX
FOUNDATION