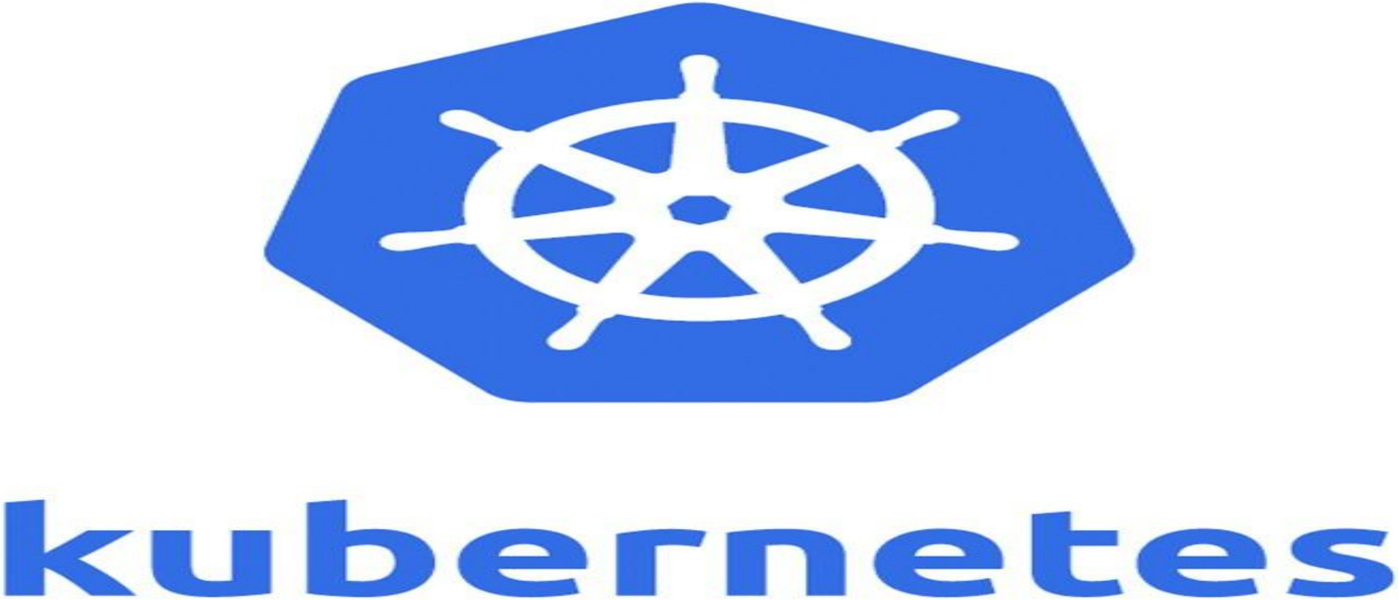
IN-DEPTH KUBERNETES PART III

INTRODUCTION

- Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications.
- It was originally designed by Google, and is now maintained by the Cloud Native Computing Foundation.
- First of all we need to setup kubernetes environment.
- You also need a good understanding of YAML language for creating configuration files in kubernetes and a basic understanding of what a master and worker nodes are and what Pods, Replica Sets and Deployments are.

INTRODUCTION

- There is no doubt about the fact that the adoption of kubernetes is expected to grow exponentially in the coming years as seen in the growth from Google Trends.
- So it is important for us to be prepared to establish credibility and value in the market.

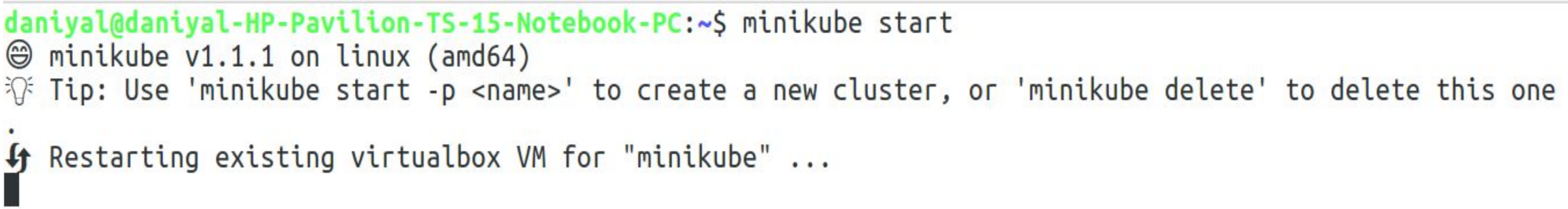


Let’s Start

MINIKUBE

MINIKUBE

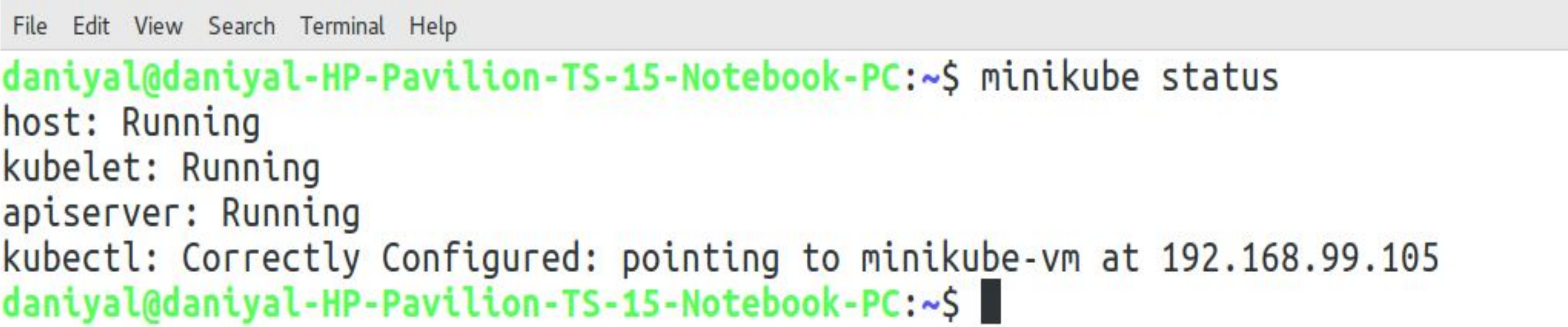
- **Minikube** is a tool that makes it easy to run **Kubernetes** locally.
- **Minikube** runs a single-node **Kubernetes** cluster on your laptop.
- Kubernetes architecture consist of Master node and worker nodes
- Minikube is a tool that makes it easy to run Kubernetes locally
- Minikube runs a single-node Kubernetes cluster on your laptop to use kubernetes for practice or development
- To start Kubernetes cluster using below command.



```
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~$ minikube start
😄  minikube v1.1.1 on linux (amd64)
💡  Tip: Use 'minikube start -p <name>' to create a new cluster, or 'minikube delete' to delete this one
.
🔄  Restarting existing virtualbox VM for "minikube" ...
█
```

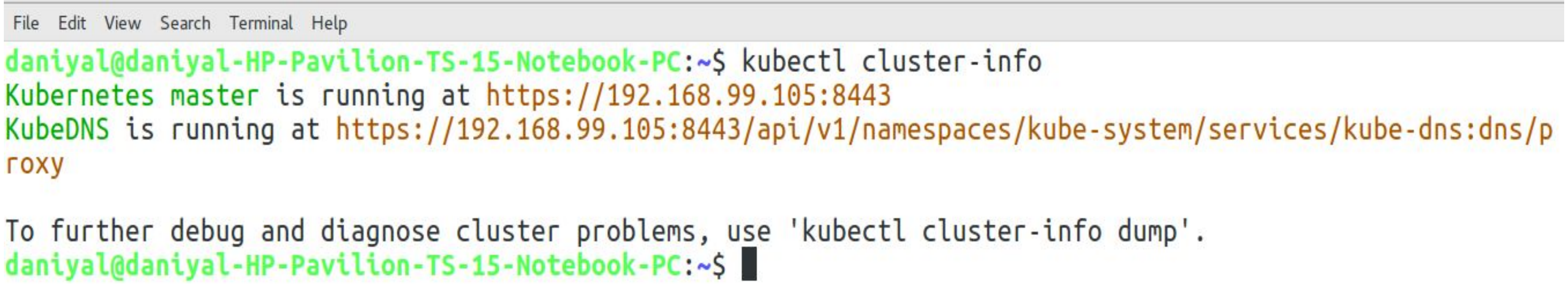
MINIKUBE COMMANDS

- To check the minikube status



```
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~$ minikube status
host: Running
kubelet: Running
apiserver: Running
kubectl: Correctly Configured: pointing to minikube-vm at 192.168.99.105
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~$ 
```

- To check the cluster info like addresses of the kubernetes master and services



```
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~$ kubectl cluster-info
Kubernetes master is running at https://192.168.99.105:8443
KubeDNS is running at https://192.168.99.105:8443/api/v1/namespaces/kube-system/services/kube-dns:dns/p
roxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~$ 
```

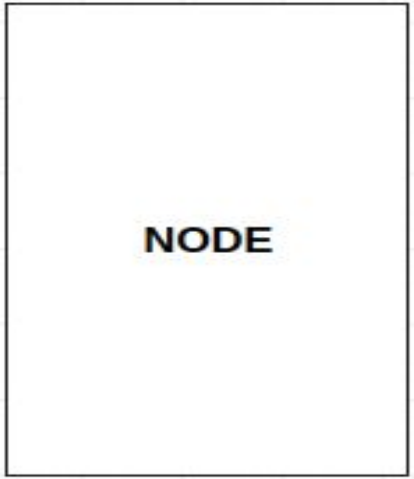
CORE CONCEPTS

KUBERNETES ARCHITECTURE

NODES

NODES

- A **node** is a machine physical or virtual one which kubernetes is installed.

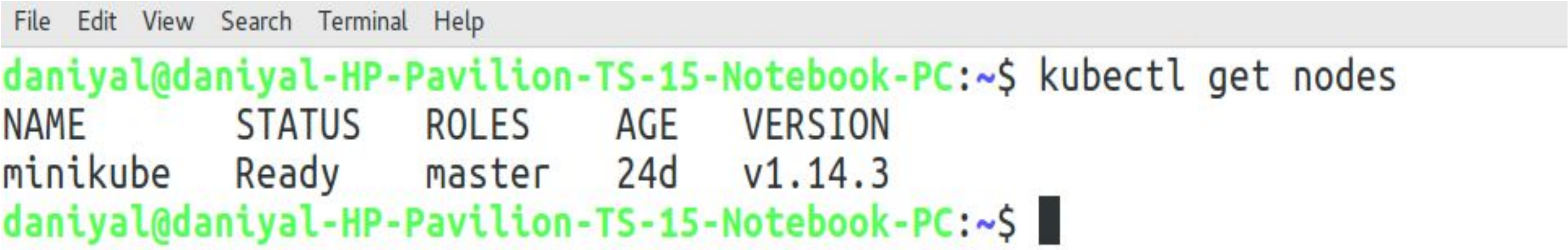


NODES

- A **node** is a worker machine and that is where containers will be launched by kubernetes.
- It was also known as “Minion’s” in the past.
- So you might hear these terms used interchangeably.
- But what if the node on which your application is running fails?
- Well Obviously our application goes down.
- So you need to have more than one Node.

NODES

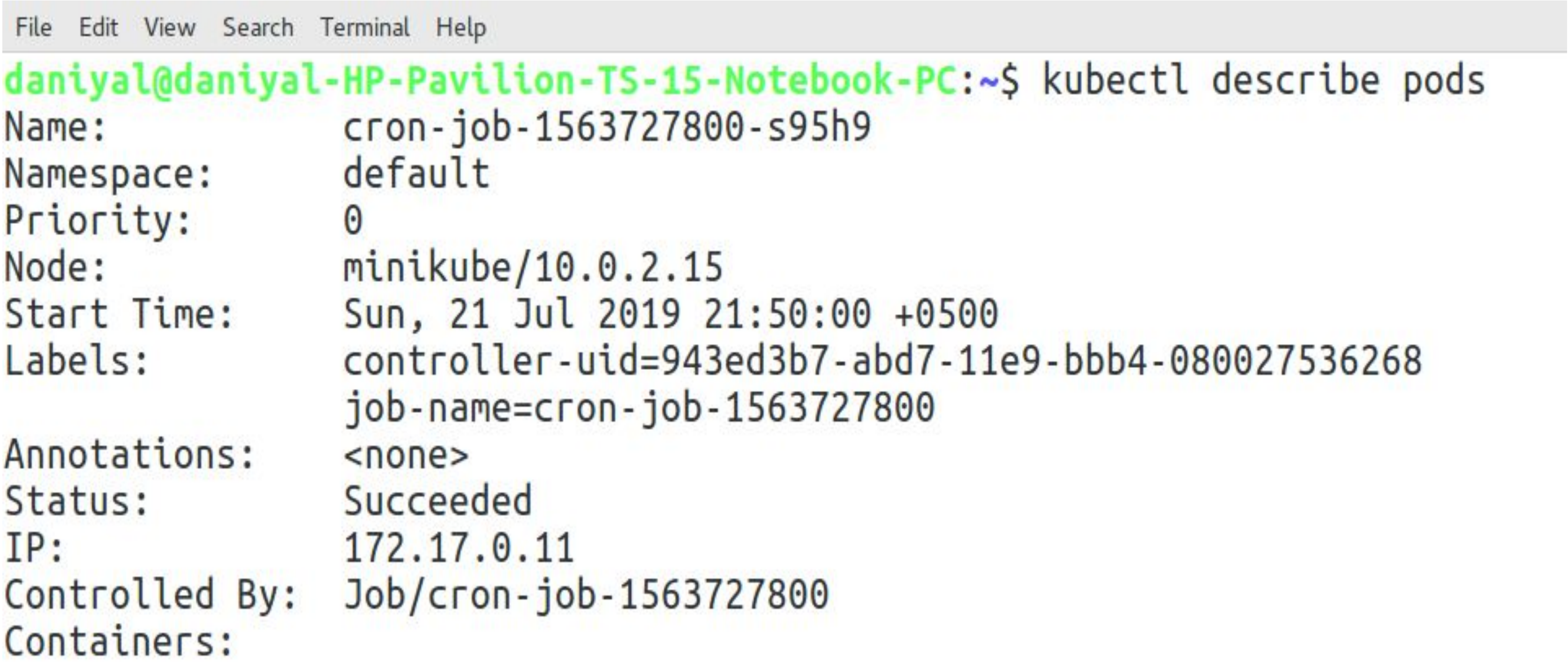
- The most common operations can be done with the following kubectl commands:
- **kubectl get** - list resources
- **kubectl describe** - show detailed information about a resource
- **kubectl logs** - print the logs from a container in a pod
- **kubectl exec** - execute a command on a container in a pod
- To view the list of nodes.



```
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~$ kubectl get nodes
NAME       STATUS   ROLES    AGE   VERSION
minikube   Ready    master   24d   v1.14.3
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~$ 
```

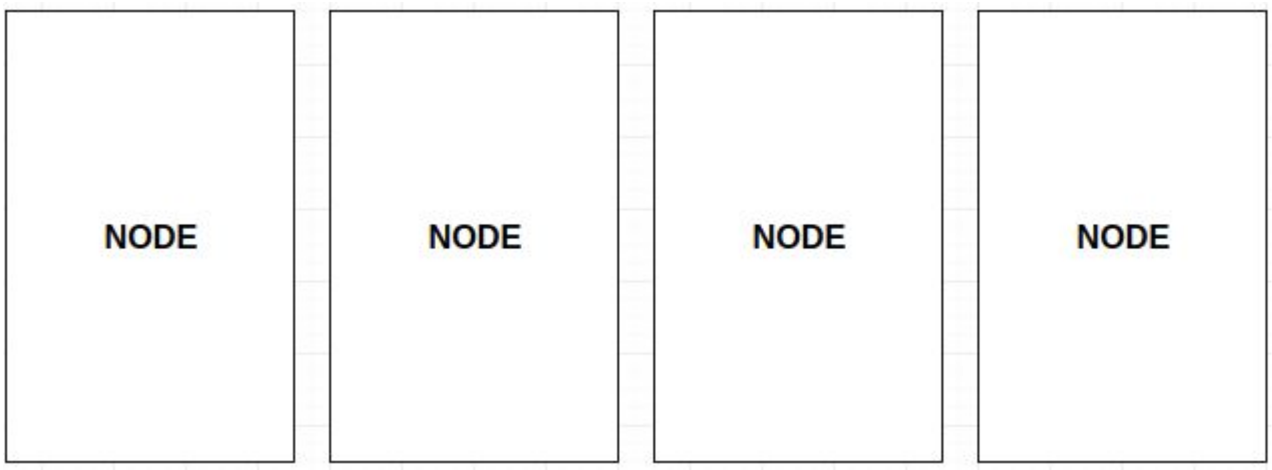
KUBECTL COMMAND

- To Display detailed state of one or more resources.



```
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~$ kubectl describe pods
Name:           cron-job-1563727800-s95h9
Namespace:      default
Priority:       0
Node:           minikube/10.0.2.15
Start Time:     Sun, 21 Jul 2019 21:50:00 +0500
Labels:         controller-uid=943ed3b7-abd7-11e9-bbb4-080027536268
                job-name=cron-job-1563727800
Annotations:    <none>
Status:         Succeeded
IP:             172.17.0.11
Controlled By:  Job/cron-job-1563727800
Containers:
```

CLUSTER



NODE

NODE

NODE

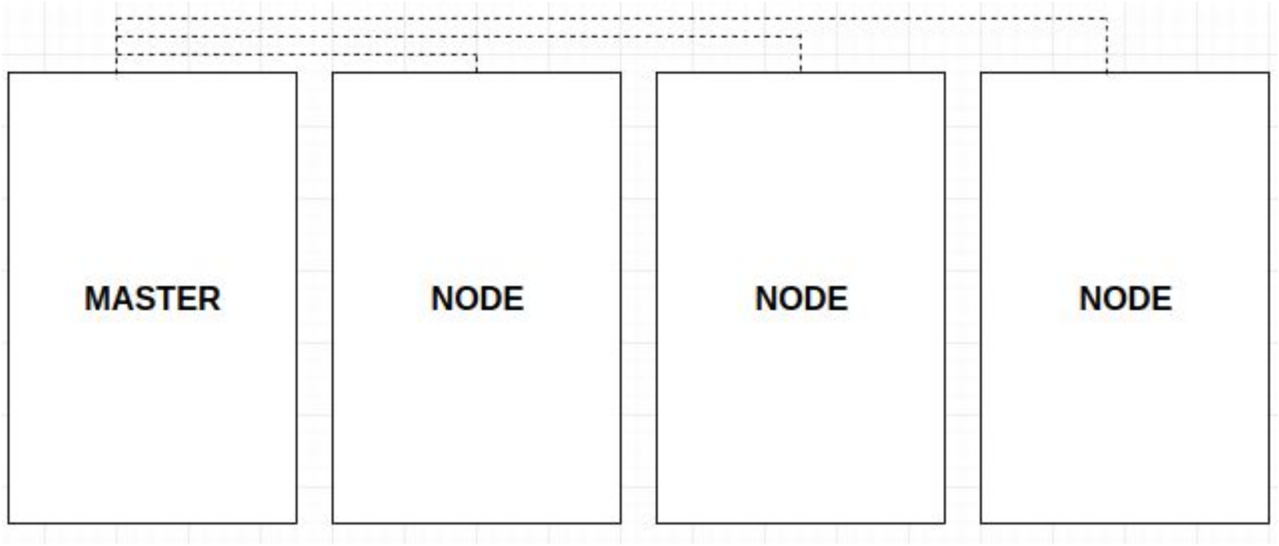
NODE

CLUSTER

- A cluster is a set of nodes grouped together.
- This way even if one node fails you have your application still accessible from the other Nodes.
- Moreover having multiple nodes helps in sharing load as well.
- Now we have a cluster but who is responsible for managing the cluster.
- Where is the information about members of the cluster stored.
- How were the Nodes monitored.
- When a node fails how do you move the workload of the failed node to another worker node?

HERE IS **MASTER** COMES IN.

MASTER

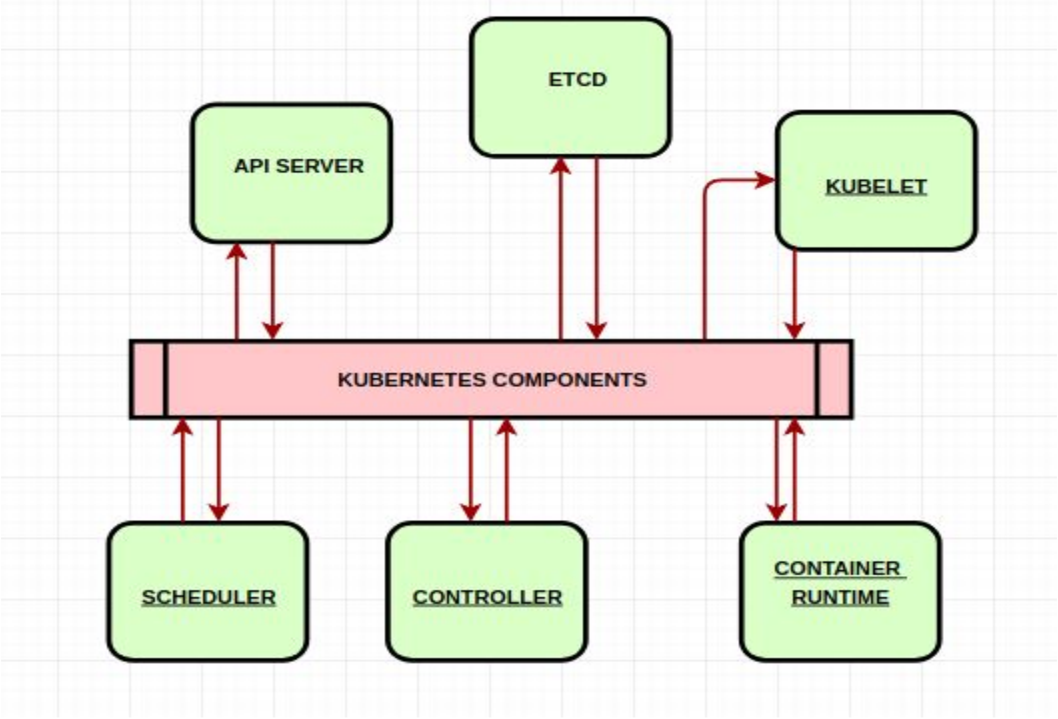


MASTER

- The Master is another node with kubernetes installed in it and is configured as a Master.
- The Master watches over the Nodes in the cluster and is responsible for the actual orchestration of containers on the worker Nodes.
- The “**master**” refers to a collection of processes managing the cluster state. Typically all these processes run on a single node in the cluster, and this node is also referred to as the **master**.
- The **master** can also be replicated for availability and redundancy.

COMPONENTS

When install kubernetes on a system the following components install with it.



COMPONENTS

- **API Server**
    - The API server acts as the frontend for kubernetes the users, management devices, command line interfaces all talk to the API Server to interact with the kubernetes cluster.
- **ETCD**
    - **Kubernetes** uses **etcd** to store all its data. Its configuration data, its state, and its metadata.
    - **Kubernetes** is a distributed system, so it needs a distributed data store like **etcd**.
    - **Etcd** lets any of the nodes in the **Kubernetes** cluster read and write data.
- **SCHEDULER**
    - The scheduler is responsible for distributing work or containers across multiple nodes. It looks for newly created containers and assigns them to nodes.

COMPONENTS

- **CONTROLLER**
  - The controllers are the brain behind orchestration. They are responsible for noticing and responding when nodes, containers and end-points goes down.
- **CONTAINER-RUNTIME**
  - The container runtime is the underline software that is use to run containers, in our case it happens to be docker but there are other options as well.
- **KUBELET**
  - The kubelet is the agent that runs on each node in the cluster.
  - The agent is responsible for making sure that the containers are running on the nodes as expected.

MASTER VS WORKER NODE’s

- So far we saw two types of servers.
- Master and Worker and a set of components that make up kubernetes.
- But how are these companies distributed across different types of servers.
- In other words how does one server become a master and the other slave.
- The worker node or minion as it is also known is where the containers are hosted.
- For example: Docker containers and to run docker containers on a system we need container runtime installed.
- And that’s where the container runtime falls.

MASTER VS WORKER NODE’s

- In this case it happens to be Docker. This doesn’t have to be docker.
- There are other container runtime alternatives available such as (rkt)Rocket or CRI-O.
- But throughout this course we are going to use Docker as our container runtime engine.
- The master server has the kube API server and that is what makes it a master.
- Similarly the worker nodes have to be kubelet agent that is responsible for interacting with a master to provide health information of the worker node and carry out actions requested by the master on the worker nodes.

MASTER VS WORKER NODE’s

- All the information gathered are stored in a key value store on the master.
- The key value store is based on the popular ETCD framework as we just discussed.
- The master also has the control manager and the scheduler.
- There are other components as well but we will stop there for now.
- The reason we went through this is to understand what components constitute the master and worker nodes.
- This will help us to install and configure the right components on different systems when we setup our infrastructure.

KUBECTL

KUBECTL

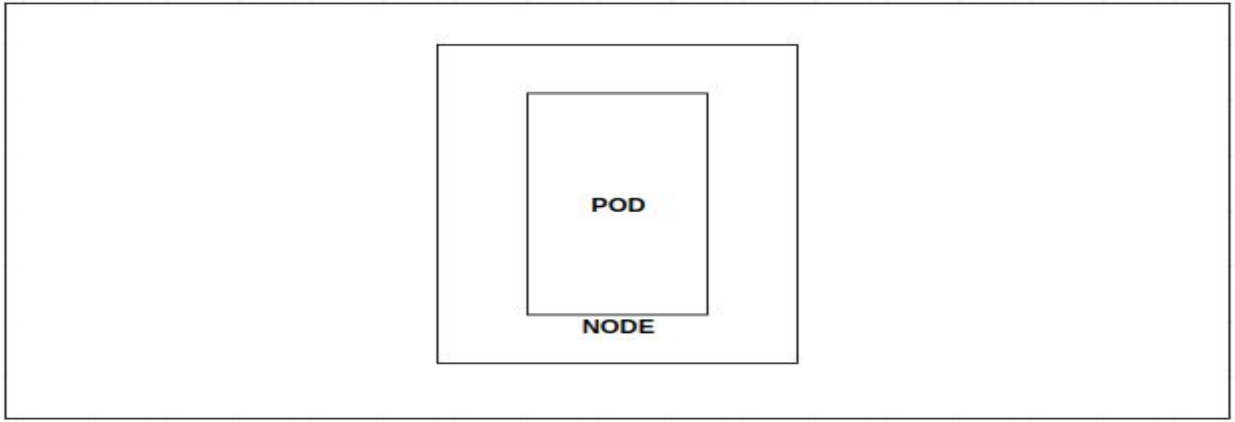
- The kubectl is the command line tool.
- The kube control tool is used to deploy and manage applications on a Kubernetes cluster to get cluster information, to get the status of other nodes in the cluster and to manage many other things.
- The kubectl run command is used to deploy an application on the cluster.
- The kubectl cluster info command is used to view information about the cluster and the **kubectl get nodes** command is used to list all the nodes part of the cluster.
- That’s all we need to know for now and we will keep learning more components throughout this course.

KUBERNETES PODS

- The containers are encapsulated into a kubernetes object known as PODs.
- A Pod is a single instance of an application.
- A Pod is the smallest object that you can create in Kubernetes.
- This could be a single node setup and or a multi node setup, doesn’t matter.
- Each pod is like a separate logical machine with its own IP, hostname, processes, and so on, running a single application
- A Pod with multiple containers will always run on a same worker node .

KUBERNETES PODS

- However kubernetes does not deploy containers directly on the worker nodes.
- It creates a POD and deploy it into the Node.

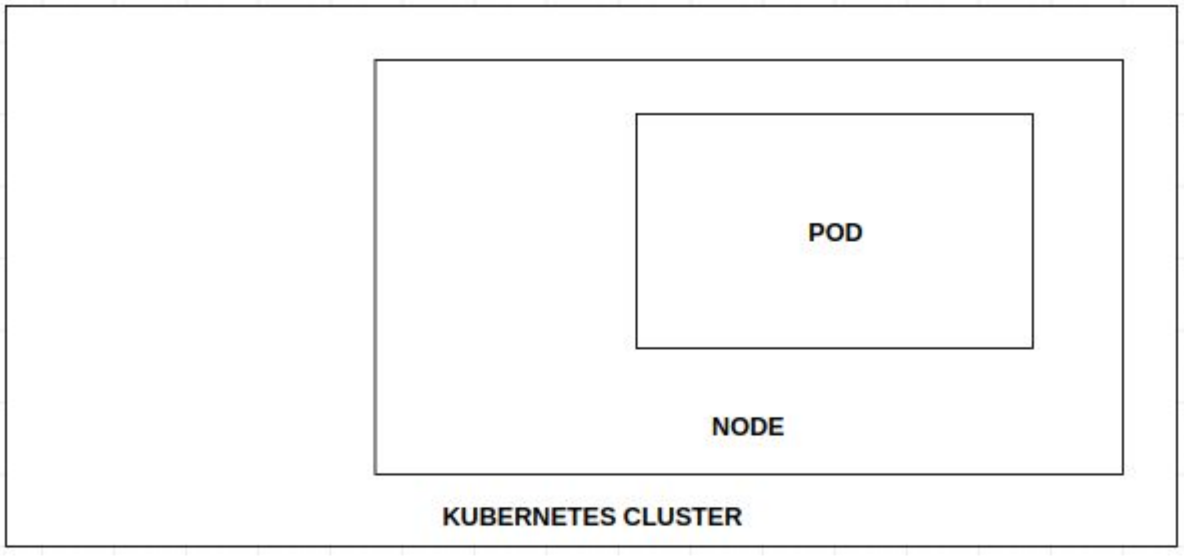


KUBERNETES PODS

- A pod of containers allows you to run closely related processes together.
- Kubernetes provide these containers with (almost) the same environment as if they were all running in a single container, while keeping them somewhat isolated.
- “Somewhat isolated”, this is because you want containers inside each group to share certain resources, although not all, so that they’re not fully isolated.
- For example if your main application’s container write logs in some file and you want another application to use some part of that log file and make some report.
- Thanks to kubernetes which provides volume concept by which we can share files directories between containers with in the pods
- This is also one of the reason why pods having multiple container does not spread over different nodes. They always co-located on same worker node.

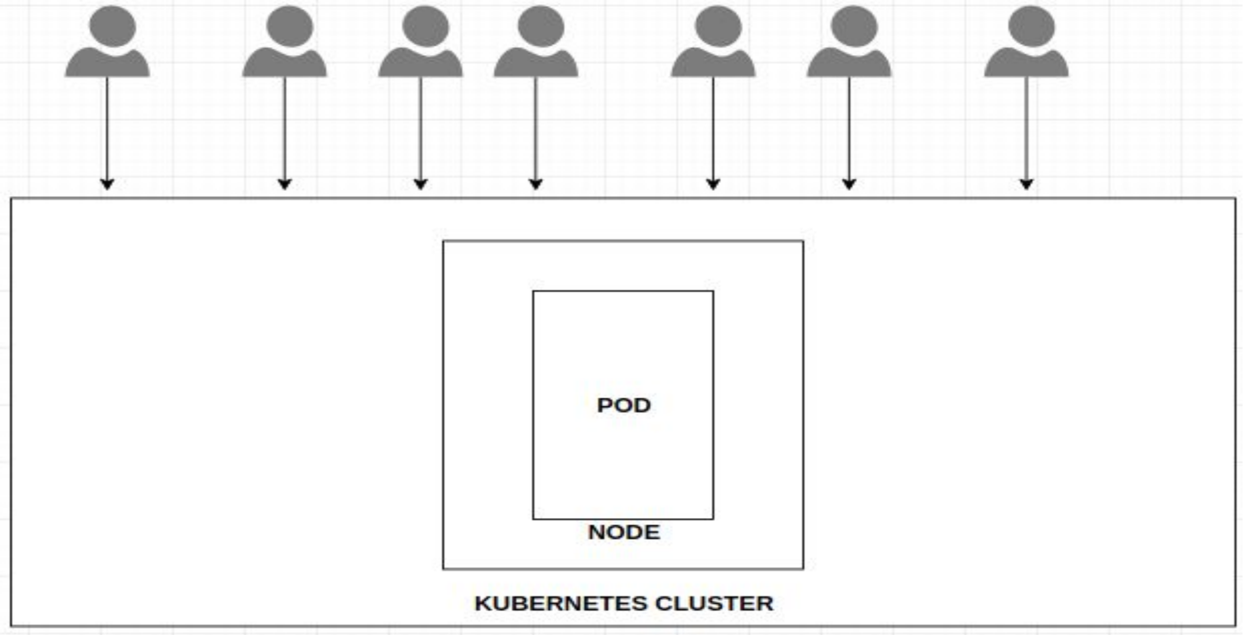
KUBERNETES PODS

- Here we see the simplest of simplest cases where you have a single node kubernetes cluster with a single instance of your application running in a single docker container encapsulated in a POD.



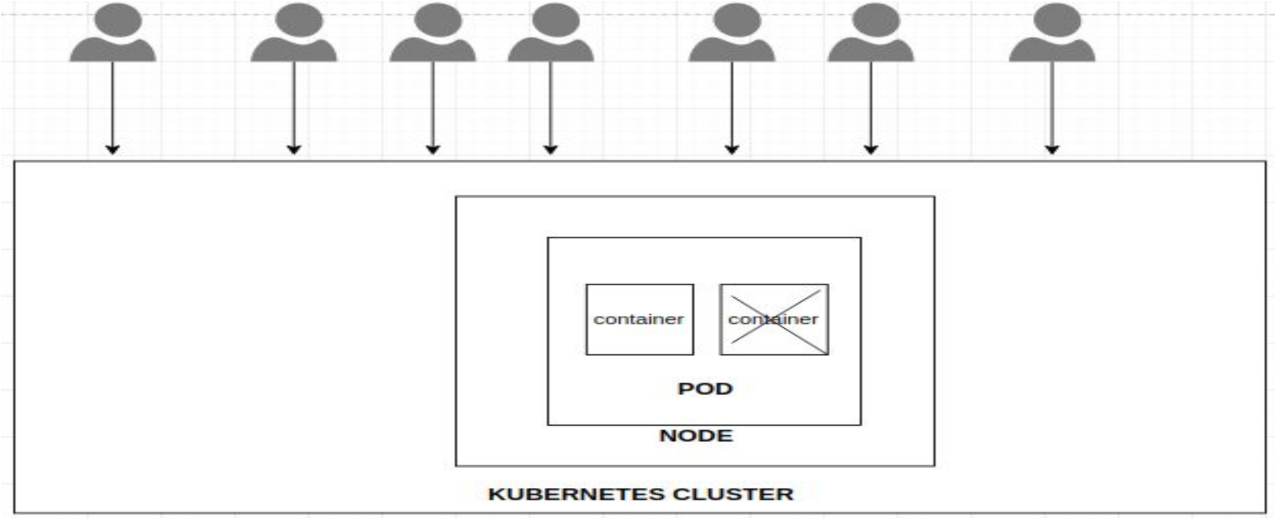
KUBERNETES PODS

- What if the number of users accessing your application increased and you need to scale your application.



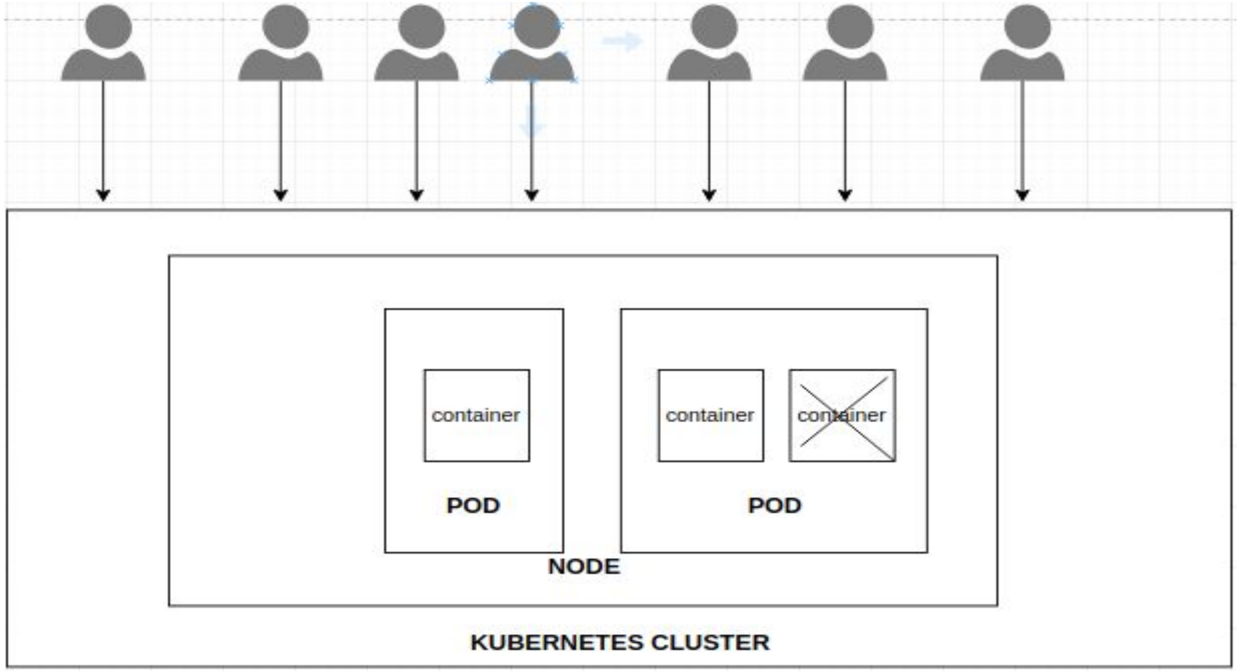
KUBERNETES PODS

- You need to add additional instances of your web application to share the load.
- Now where would you spin up additional instances.
- Do we bring up a new container instance within the same POD, No.

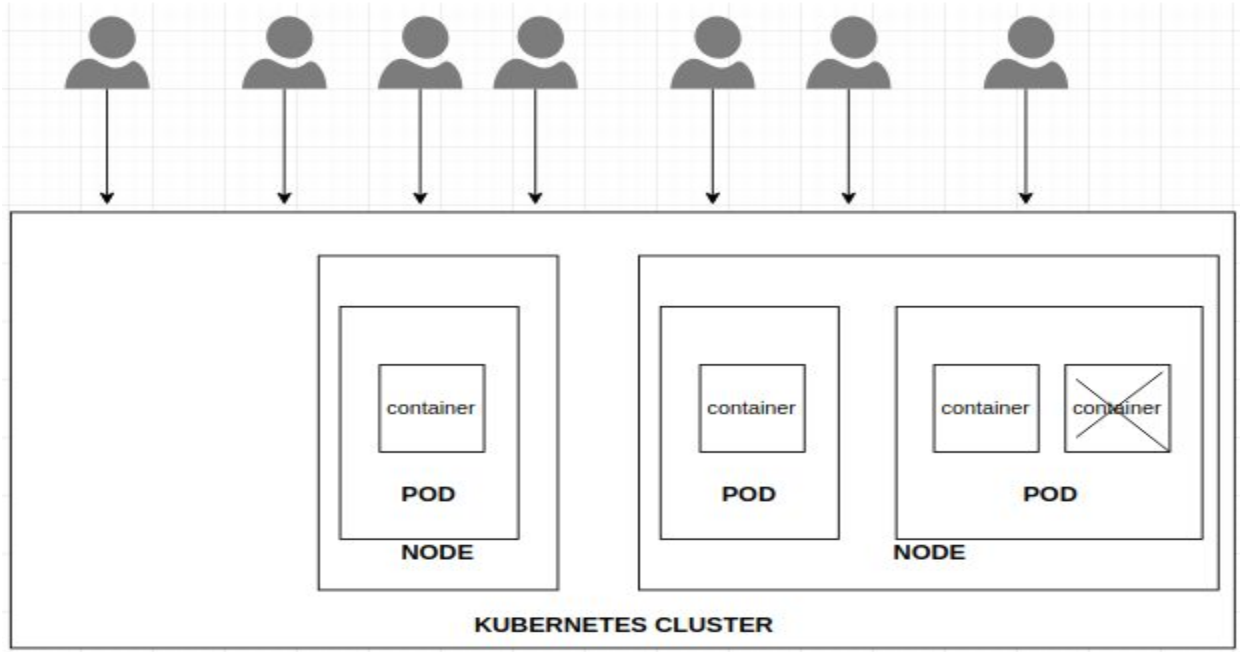


KUBERNETES PODS

- We create a new POD altogether with a new instance of the same application as you can see below image.



- We now have two instances of our web application running on two separate PODs on the same kubernetes system or node.
- What if the user base further increases and your current node has no sufficient capacity.
- Well then you always deploy additional PODs on a new node in the cluster.



KUBERNETES PODS

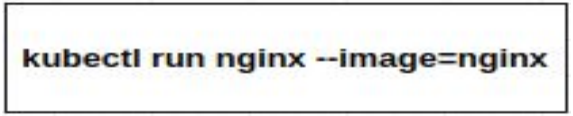
- You will have a new node added in the cluster to expand the clusters physical capacity.
- So what i am trying to illustrate in this slide is that PODs usually have a one to one relationship with containers running your application.
- To scale up, You create new PODs and to scale down you delete existing PODs.
- You do not add additional containers to an existing PODs to scale your application.
- Also if you’re wondering how we implement all of this and how we achieve load balancing between the containers etc.
- We will get into all of that in a later lecture.
- For now we are only trying to understand the basic concepts.

KUBERNETES PODS

- We just said that PODs usually have a one to one relationship with the containers. But are we restricted to having a single container in a single POD. NO
- A single POD can have multiple containers except for the fact that they’re usually not multiple containers of the same kind.
- As we discussed in the previous slide if our intention was to scale our application then we would need to create additional PODs.
- The two containers can also communicate with each other directly by referring to each other as local host since they share the same network space plus they can easily share the same storage space as well.

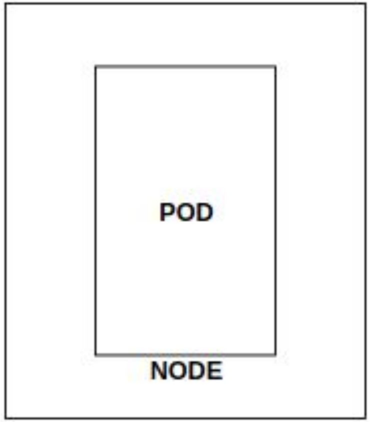
KUBECTL

- Let’s now look at how to deploy PODs.
- Earlier we learned about the kubectl run command.



```
kubectl run nginx --image=nginx
```

- What this command really does is It deploys a Docker container by creating a POD.

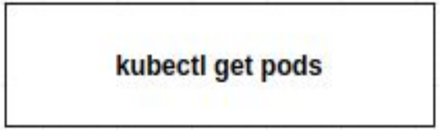


KUBECTL

- So it first creates a POD automatically and deploys an instance of the nginx next to Docker image.
- But where does it get the application image from. For that, you need to specify the image name using the --image parameter.
- The application image, In this case the nginx image is downloaded from the docker hub repository.
- Docker Hub as we discussed is a public repository where latest docker images of various applications are stored.
- You could configure kubernetes that is to pull the image from the public Docker hub or a private repository within the organization.
- Now that we have a POD created how do we see a list of PODs available.

KUBECTL

- The kubectl get pods command helps us see the list of pods in one cluster.



```
kubectl get pods
```

- Also remember that we haven’t really talked about the concepts on how a user can access the nginx web server.
- For now we will just see how to deploy a POD and in a later lecture once we learn about networking and services we will get to know how to service accessible to end users.

YAML

What is YAML?

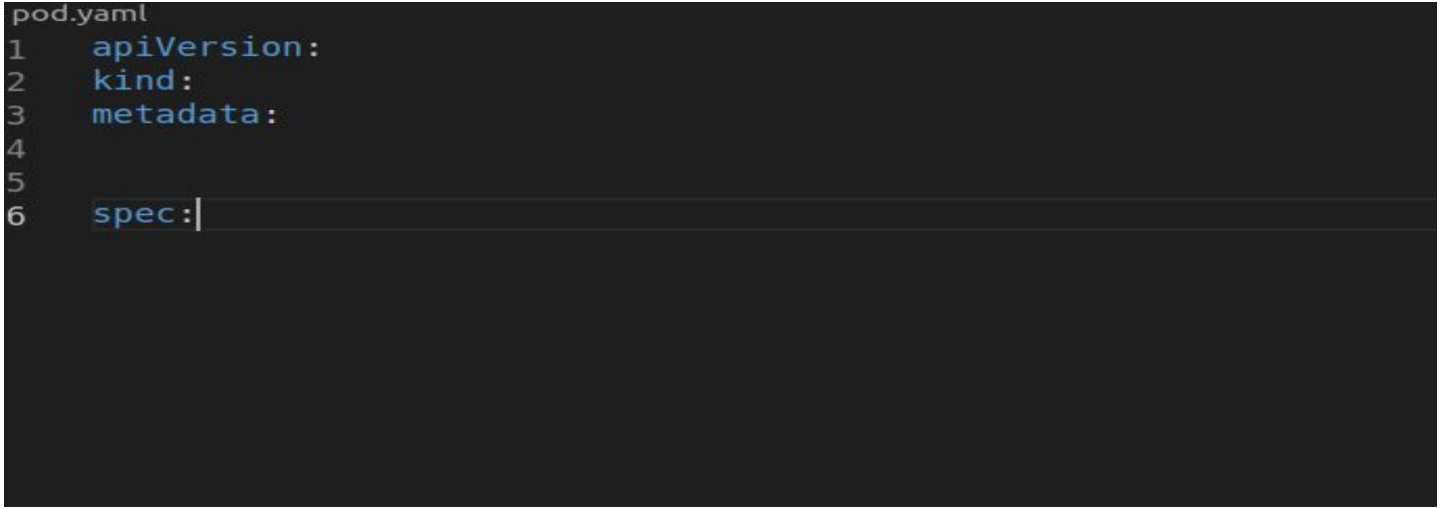
- YAML is a human-readable data-serialization language.
- It is commonly used for configuration files, but could be used in many applications where data is being stored or transmitted.
- When you are creating a file in YAML, you should remember the following basic rules:
    - YAML is case sensitive
    - The files should have .yaml as the extension
    - YAML does not allow the use of tabs while creating YAML files ; spaces are allowed instead.

PODs with YAML

- Now we will talk about creating a pod using a YAML based configuration file.
- We learnt about YAML files in detail.
- Now we will learn how to deploy YAML files specifically for kubernetes.
- Kubernetes uses YAML files as inputs for the creation of objects such as pods, replicasets, deployments, services etc. All of these follow a similar structure.
- Kubernetes definition file always contains four top level fields.
    - The API version
    - Kind
    - Metadata
    - Spec

PODs with YAML

- These are the top level or root level properties.
- These are also required fields. So you must have them in your configuration file.
- Let us look at each one of them.



pod.yaml

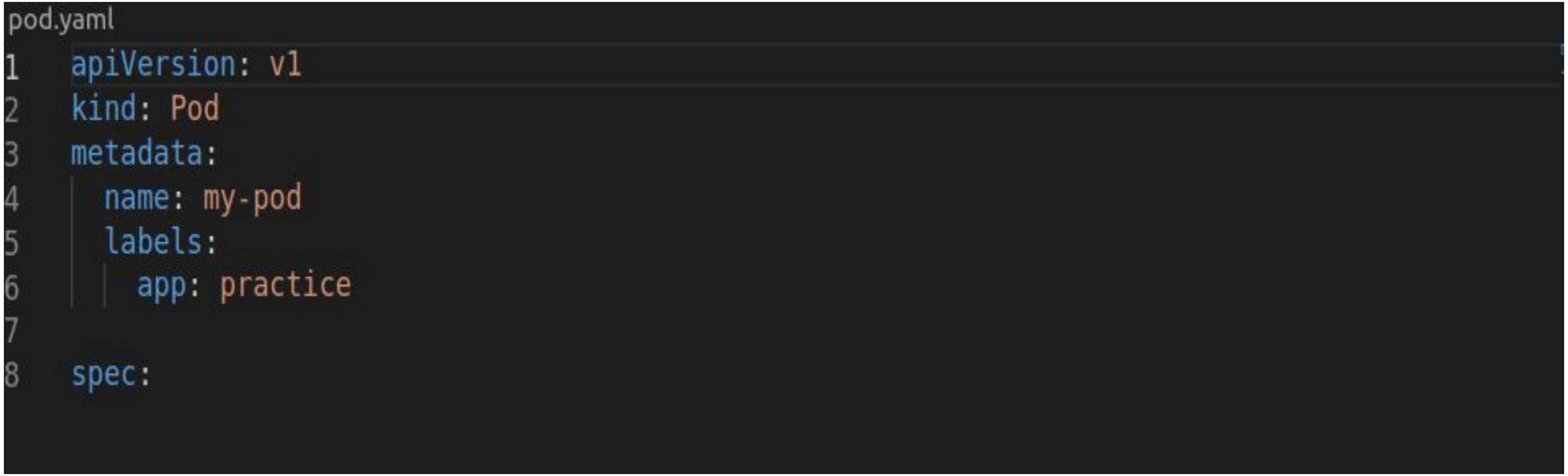
```yaml
apiVersion:
kind:
metadata:


spec:
```

PODs with YAML

- The first one is the api version. This is the version of the kubernetes API. we are using to create the object. Depending on what we are trying to create, we must use the right api version.
- For now, since we are working on PODs.
- We will set the api version as v1.
- Few other possible values for this field are apps/v1-beta etc.
- Next is the kind.
- The kind refers to the type of object we are trying to create which in this case happens to be a POD.
- So we will set it as Pod. Some other possible values here could be replica set or deployment or service.

PODs with YAML

- Next is metadata. The metadata is data about the object like its name, labels etc.

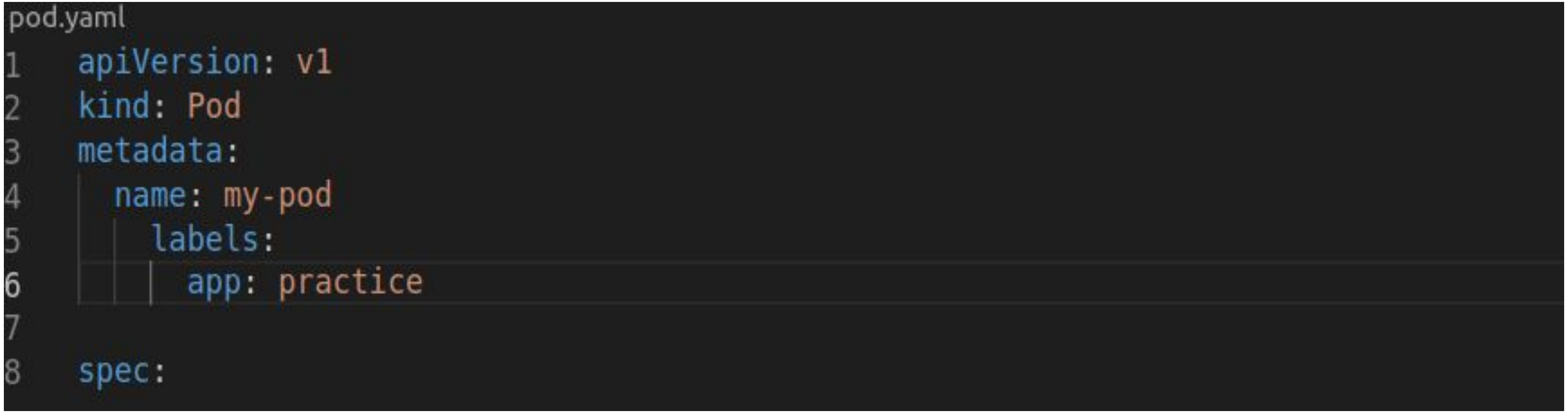


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  labels:
    app: practice

spec:
```

- So everything under metadata is indented to the right a little bit and so name and labels are children of metadata.
- The number of spaces before the two properties name and labels doesn’t matter but they should be the same as they are siblings.

PODs with YAML

- In this case labels has more spaces on the left name and so it is now a child of the name property instead of a sibling, which is incorrect.



pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
    labels:
      app: practice

spec:
```

- Under metadata the name is a string value so you can name your pod and the labels is a dictionary, so labels is a dictionary within the metadata dictionary

PODs with YAML

- It can have any key and value pairs as you wish.
- For now i have added a label with the value “practice”. Similarly you could add other labels as you see fit which will help you identify these objects at a point in time.
- It’s important to note that under metadata you can only specify name or labels or anything else that kubernetes expects to be under metadata.
- You cannot add any other property as you wish under this.
- However under labels you can have any kind of key or value pairs as you see fit.
- So it’s important to understand what each of these parameters expect.
- So far we have only mentioned the type and name of the object we need to create which happens to be a pod with a name my-pod but we haven’t really specified the container or image we need in the pod.

PODs with YAML

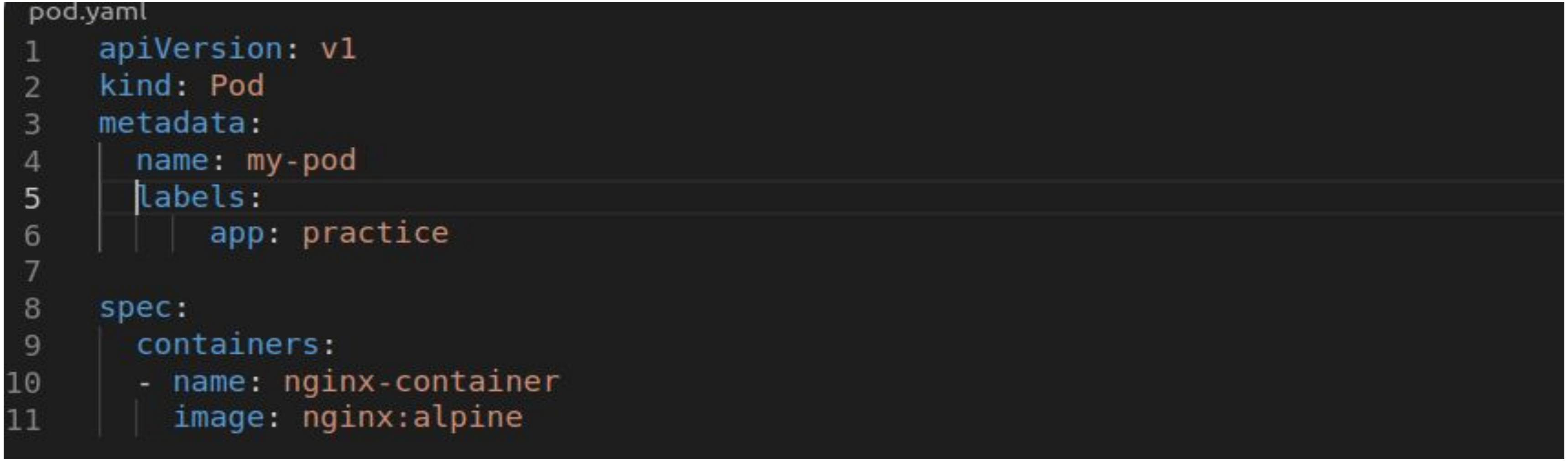
The last section in the configuration file is the specification section which is written as “spec” depending on the object we are going to create.

This is where we would provide additional information to kubernetes, as pertaining to that object.

This is going to be different for different objects so it’s important to understand or refer to the documentation section to get the right format for each. Since we are only creating a pod with a single container in it, it is easy.

Spec is a dictionary, so add a property under it called “containers”.

PODs with YAML



pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  labels:
      app: practice

spec:
  containers:
  - name: nginx-container
    image: nginx:alpine
```

- Containers is a list or an array. The reason this property is a list is because the pods can have multiple containers within them.

- As we learned in the lecture earlier. In this case though, we will only add a single item in the list. Since we plan to have only a single container in the pod.

PODs with YAML

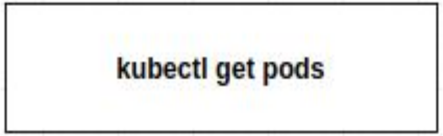
- The ‘-’ right before the name indicates that this is the first item in the list.
- The item in the list is a dictionary. So add a name and image property the value for image is nginx which is the name of the Docker image in the docker repository once the files created from the command kubectl create -f followed by the filename which is pod.yaml and kubernetes creates the pod.
- So to summarize remember the four top level properties.
  - API VERSION
  - KIND
  - METADATA
  - SPEC

PODs with YAML

- Then start by adding values to those depending on the object you are going to create.

  Once we create the pod, How do you see it?

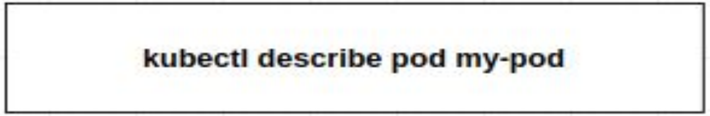
- Use this below command to see a list of pods available.



```
kubectl get pods
```

PODs with YAML

- To see the detailed information about the pod, in this case use below command.

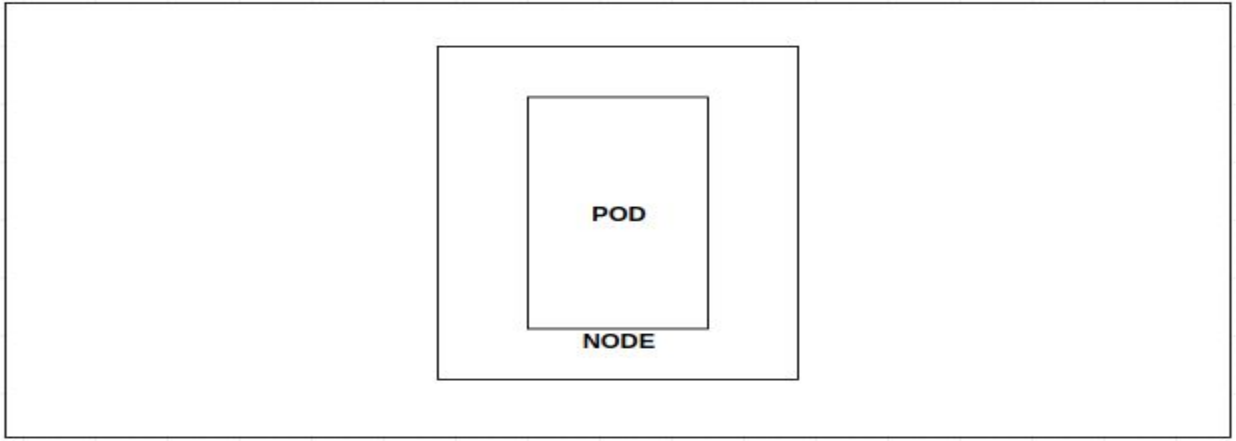


```
kubectl describe pod my-pod
```

- This will tell you information about the pod, when it was created, what labels are assigned to it, what docker containers are a part of it, and the events associated with that pod.

KUBERNETES CONTROLLERS

- Controllers are the brain behind kubernetes.
- They are the processes that monitor kubernetes objects and respond accordingly.
- We will discuss one controller now that is replication controller.
- So what is a replica and why do we need a replication controller.
- Let’s go back to our first scenario where we had a single POD running our application.
- What if for some reason our application crashes and the POD fails?



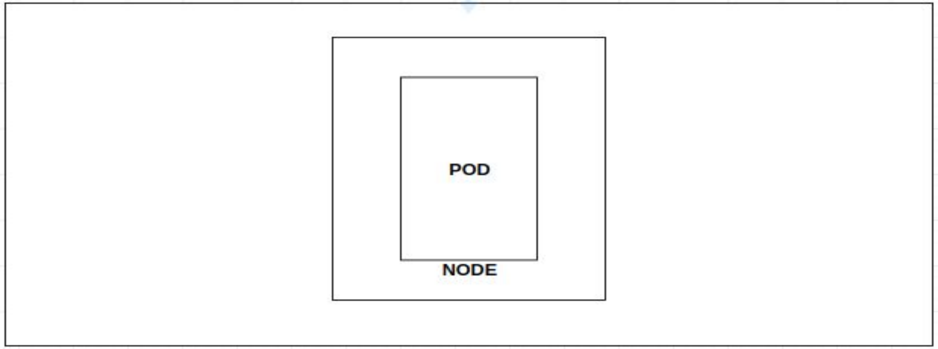
REPLICA SET

REPLICA SET

- Users will no longer be able to access our application. To prevent users from losing access to our application.
- We would like to have more than one instance or POD running at the same time. That way if one fails we still have our application running on the other one.
- The replica set helps us run multiple instances of a single POD in the kubernetes cluster thus providing high availability.
- So does that mean you can’t use a replica set if you plan to have a single Pod. No
- Even if you have a single POD, replica set can help by automatically bringing up a new POD when the existing one fails.

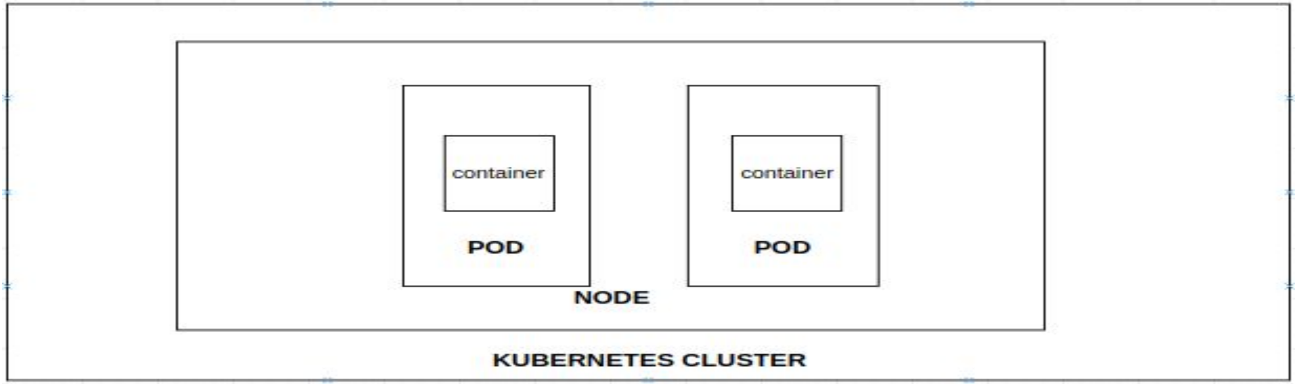
REPLICA SET

- The replica set ensures that the specified number of PODs are running at all times. Even if it is just one or hundred.
- Another reason we need a replica set is to create multiple pods to share the load across all them.



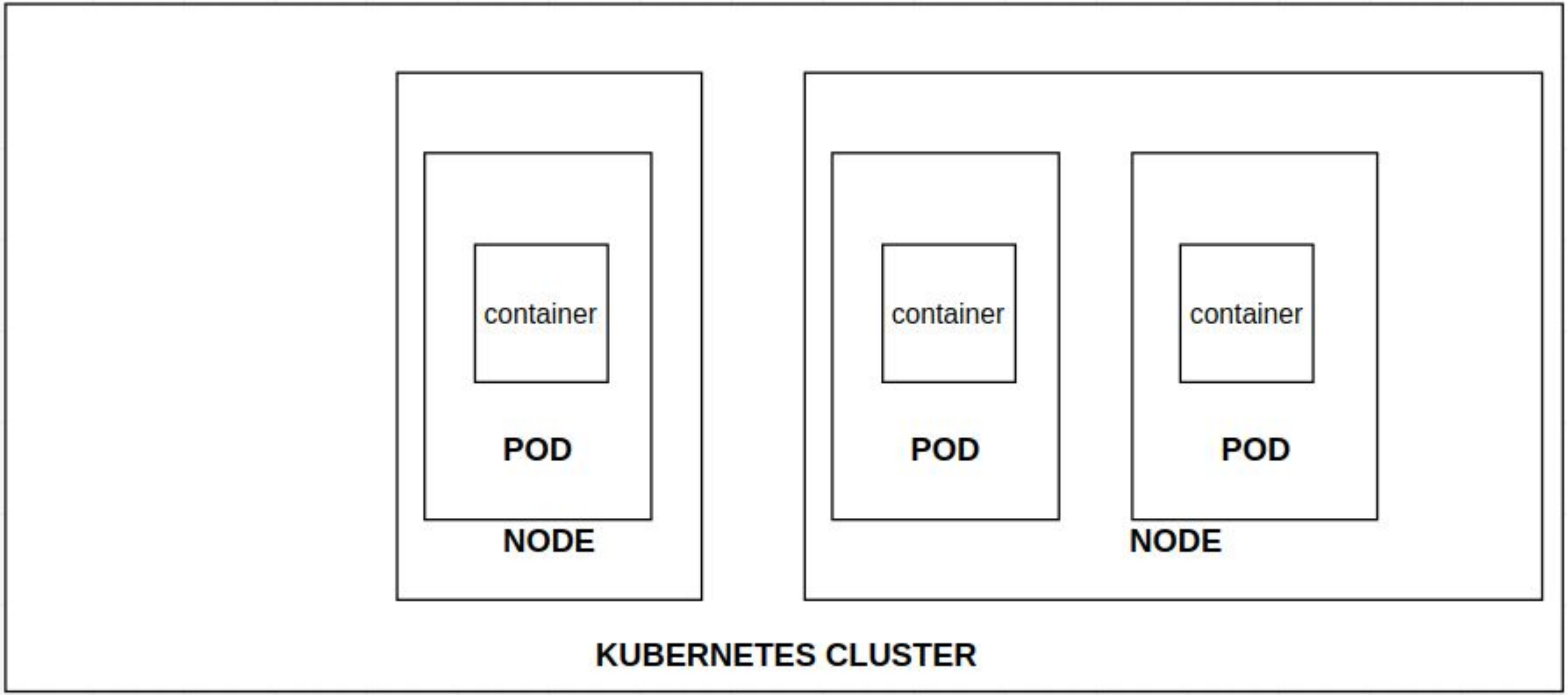
- For example in this simple scenario we have a single POD serving a set of users with the number of users increase we deploy additional POD to balance the load across the two PODs.

REPLICA SET



- If the demand further increases and if we were to run out of resources on the first node we could deploy additional PODs across the other nodes in the cluster.
- As you can see the replica set spans across multiple nodes in the cluster in the next slide.
- It helps us balance the load across multiple parts on different nodes as well as scale our application, When the demand increases.

REPLICA SET

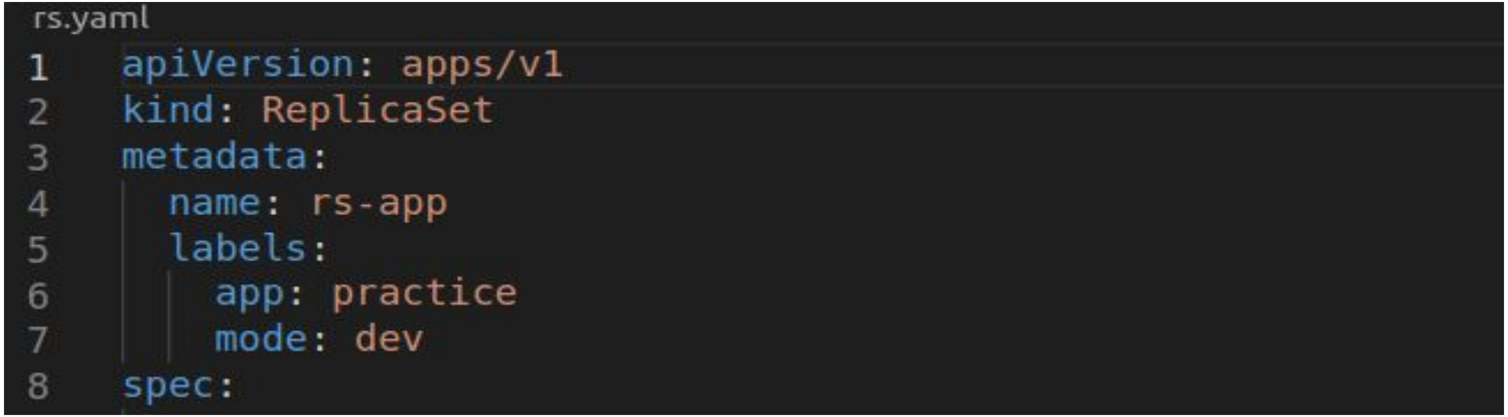

container
POD
NODE
container
POD
container
POD
NODE
KUBERNETES CLUSTER

REPLICA SET VS REPLICATION CONTROLLER

- It’s important to know that there are two similar terms Replication Controller and Replica Set.
- Both have the same purpose but they’re not the same. Replication controller is the older technology that is being by Replica Set.
- Replica Set is the new recommended way to setup replication. However whatever we discussed in the previous slides remain applicable to both these technologies.
- There are minor differences in the way each works and we will look at that in a bit.
- Let us now create a replica set.

REPLICA SET

- As with any kubernetes file, we have four sections.
  - API VERSION
  - KIND
  - METADATA
  - SPEC

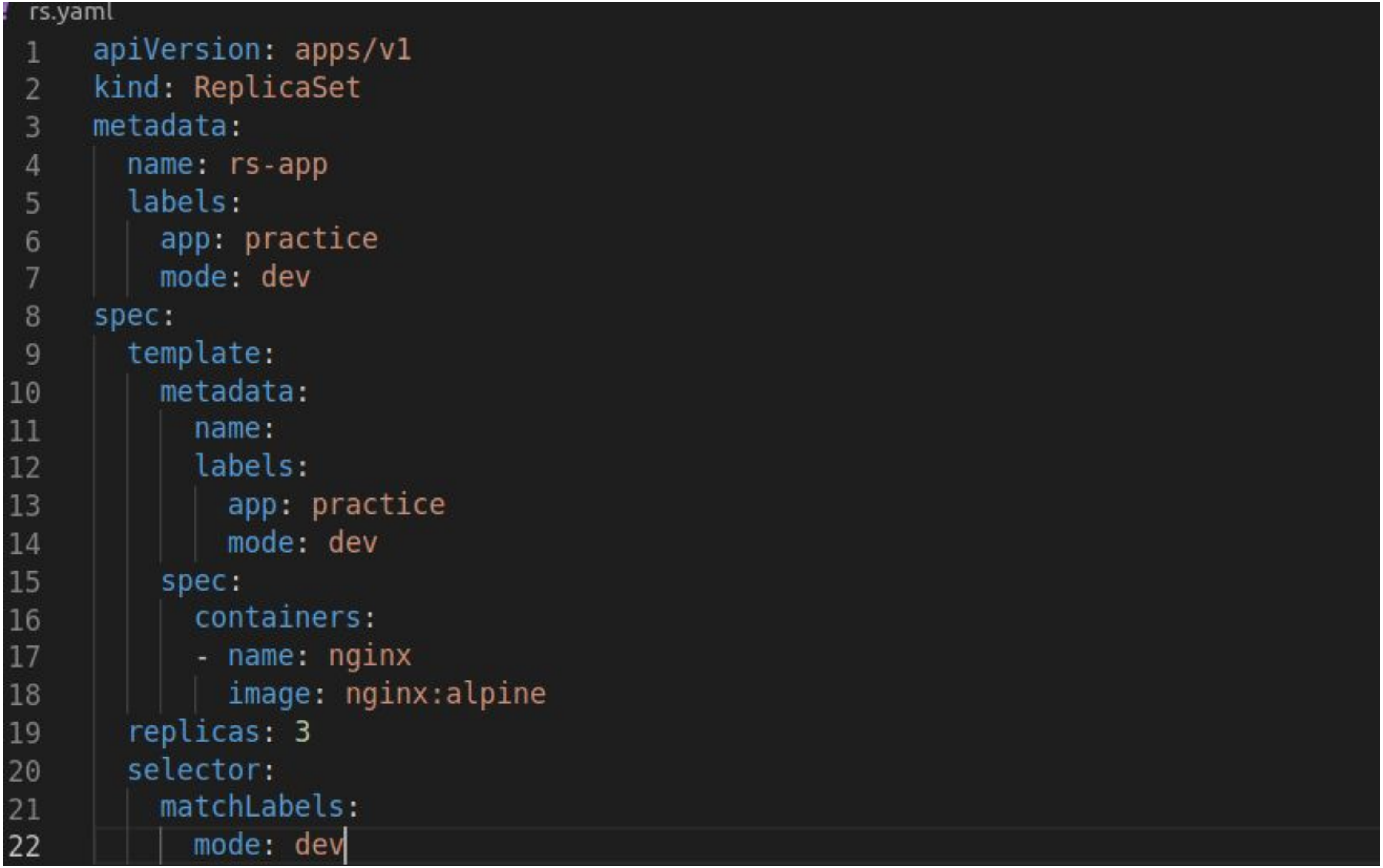


rs.yaml

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: rs-app
  labels:
    app: practice
    mode: dev
spec:
```

REPLICA SET

- The API version is specific to what we are creating. In this case replica set is supported in kubernetes API version apps/v1.
- The kind as we know is ReplicaSet.
- Under metadata we will add a name and we will call it “rs-app” and we will also add a few labels app and mode and assign values to them.
- So far it has been very similar to how we created a POD In the previous section.
- Last one is spec and it is the most crucial part of the definition file and that is the specification written as spec.
- The spec section defines what’s inside the object we’re creating.



rs.yaml

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: rs-app
  labels:
    app: practice
    mode: dev
spec:
  template:
    metadata:
      name:
      labels:
        app: practice
        mode: dev
    spec:
      containers:
      - name: nginx
        image: nginx:alpine
  replicas: 3
  selector:
    matchLabels:
      mode: dev
```

REPLICA SET

- In this case we know that the replica set creates multiple instances of a POD. but what POD?
- We create a template section under spec to provide a POD template to be used by the replica set to create replicas. Move all the contents of the POD file into the template section of the replica set.
- This should be intended to the right and have more spaces before them than the template line itself.
- They should be children of the templates section.
- Now we have two metadata sections, One is for the replica set and another for the pod.

REPLICA SET

- The API version though is a bit different. It’s apps/v1, which is different from what we had before for application controller which was just v1.
- If you get this wrong you will get an error.
- The kind would be replica set and we add name and labels in the metadata.
- The specification sections looks very similar to replication controller.
- However there is one major difference between replication controller and replica set. Replica Set requires a selector definition.
- The selector section helps the replica set identify what PODs fall under it. But why would you have to specify what PODs fall under it.

REPLICA SET

- If you have provided the contents of the POD definition file itself in the template.
- It’s because Replica Set can also manage PODs that were not created as part of the replica set creation. So for example there were PODs created before the creation of the replica set that match labels specified in the selector.
- The Replica set will also take those PODs into consideration when creating the replicas.
- The selector is the major difference between replication controller and replica set.
- The selector is the required property in the replica set and it has to be written in the form of match labels as shown above.
- The match label selector simply matches the labels specified under it to the labels on the POD.
- The replica set selector also provides many other options for matching labels that were not available in a replication controller.

REPLICA SET

- As always to create a replica set run **kubectl create** command providing the definition file as input and to see the created replicas see below command.

```
kubectl get replicaset
```

- To get the list of POD's simply run.

```
kubectl get pods
```

LABELS AND SELECTORS

What is labels and selectors?
WHY Do We Label OUR PODs And Objects in kubernetes?

LABELS AND SELECTORS

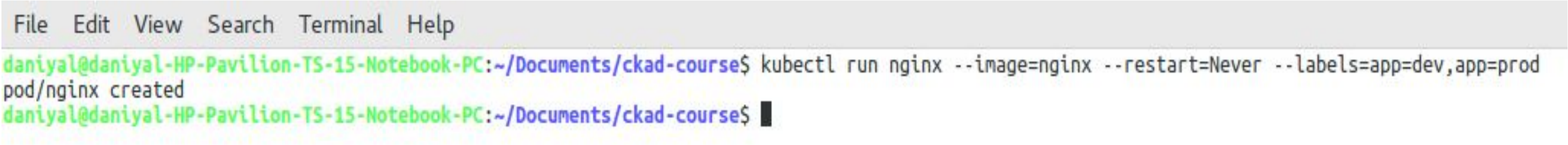
- Let us look at a simple scenario. Say we deployed three instances of our frontend application as three pods.
- We would like to create a replica set to ensure that we have three active pods at any time. And that is one of the use case of replica sets you can use it to monitor existing pods if you have them already created as it is in this example.
- In case they were not created the replica set will create them for you.
- The role of the replica set is to monitor the PODs and if any of them were to fall, deploy new ones.
- The replica set is in fact a process that monitor the PODs. Now how does the replica set know what PODs to monitor. There could be hundred of other pods in the cluster running different applications.

LABELS AND SELECTORS

- This is where labeling our PODs during creation comes in handy.
- We could now provide these labels as a filter for replica set. Under the selector section, we use to match labels filter and provide the same label that we used while creating the PODs.
- This way the replica set knows which POD to monitor.
- The same concept of labels and selectors is used many other places throughout kubernetes.

IMPERATIVE WAY TO CREATE A POD

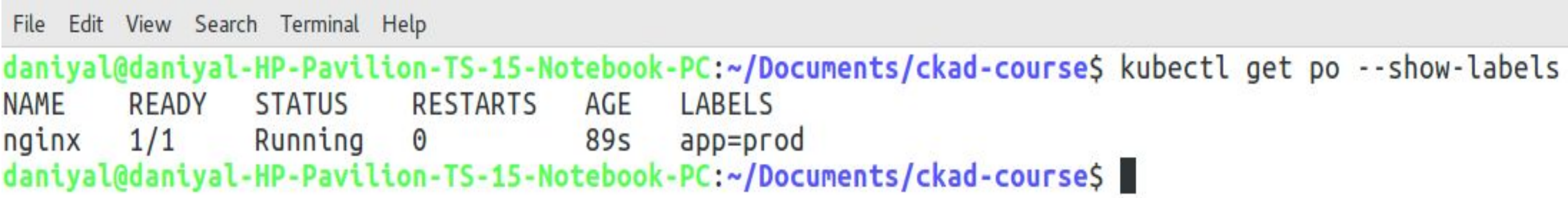
- To create a pod using command line.
  - kubectl run <pod_name> --image=<image_name> --restart=Never **--labels=key=value**



```
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ kubectl run nginx --image=nginx --restart=Never --labels=app=dev,app=prod
pod/nginx created
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ 
```

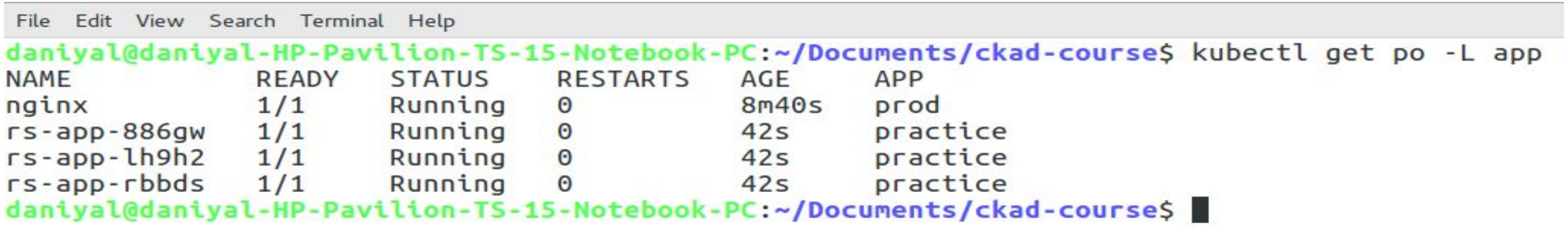
GET PODS WITH LABELS

- To get the labels in the listing.



```
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ kubectl get po --show-labels
NAME    READY   STATUS    RESTARTS   AGE   LABELS
nginx   1/1     Running   0          89s   app=prod
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$
```

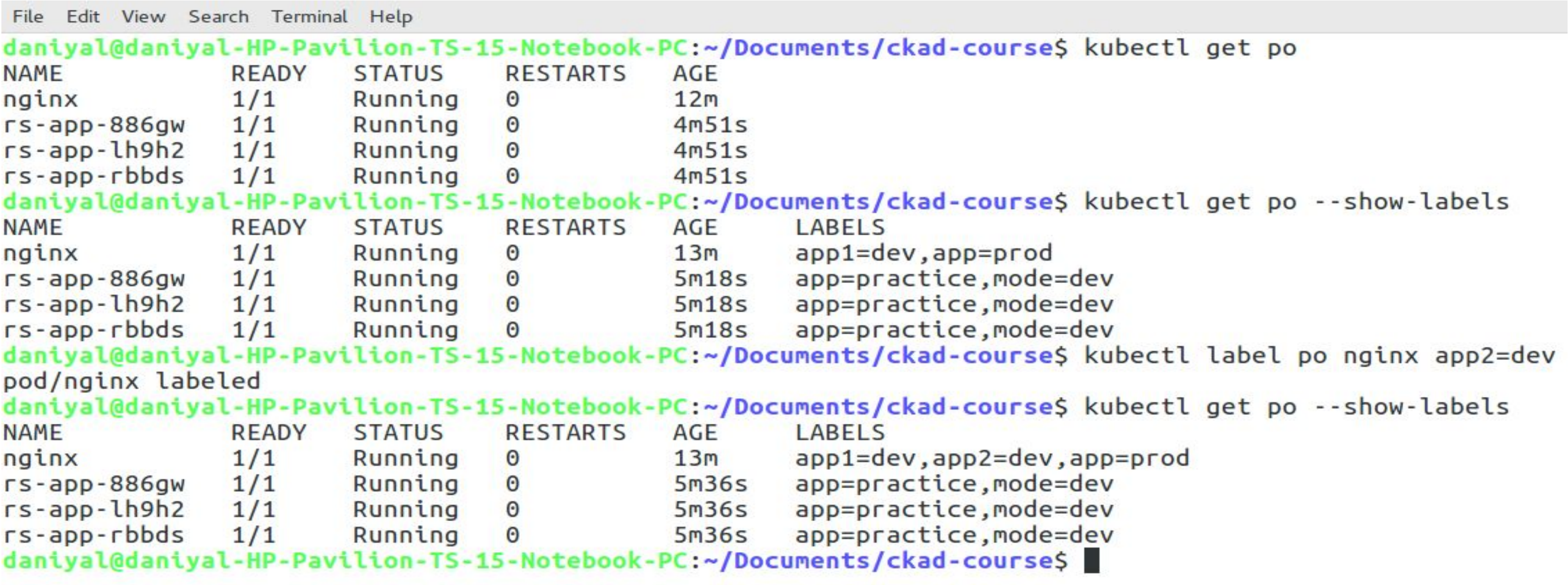
- To make label as a column.



```
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ kubectl get po -L app
NAME           READY   STATUS    RESTARTS   AGE     APP
nginx          1/1     Running   0          8m40s   prod
rs-app-886gw   1/1     Running   0          42s     practice
rs-app-lh9h2   1/1     Running   0          42s     practice
rs-app-rbbds   1/1     Running   0          42s     practice
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$
```

ASSIGN LABELS

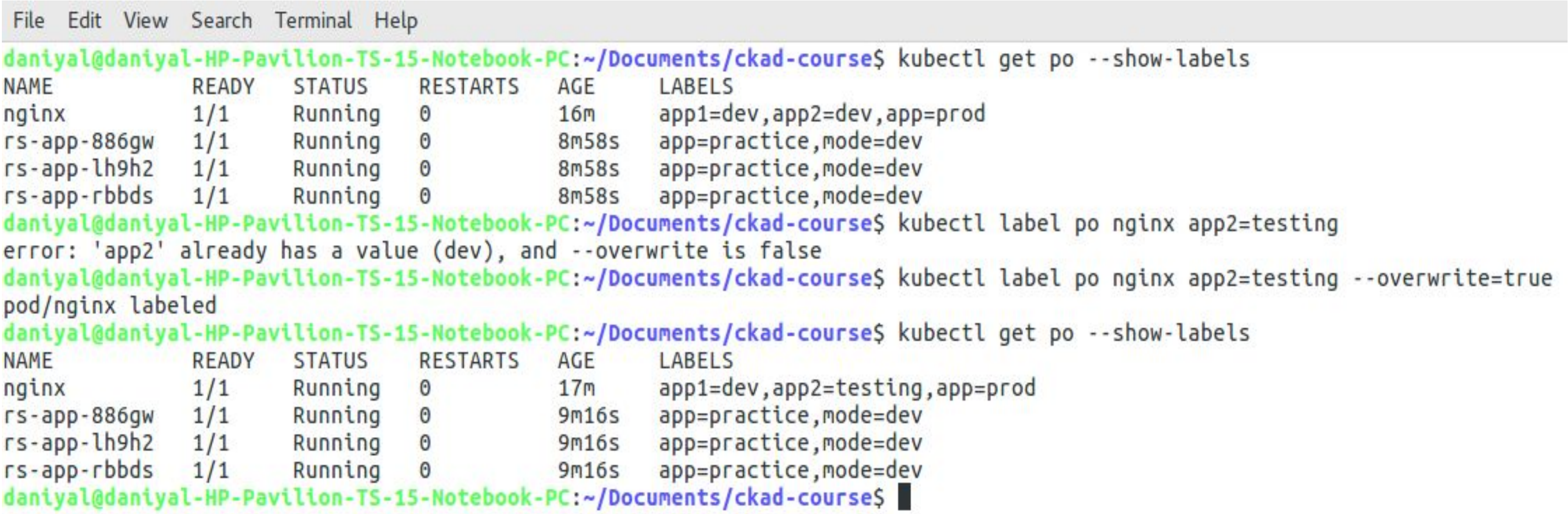
- Assigning labels to existing pods



```
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ kubectl get po
NAME           READY   STATUS    RESTARTS   AGE
nginx          1/1     Running   0          12m
rs-app-886gw   1/1     Running   0          4m51s
rs-app-lh9h2   1/1     Running   0          4m51s
rs-app-rbbds   1/1     Running   0          4m51s
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ kubectl get po --show-labels
NAME           READY   STATUS    RESTARTS   AGE     LABELS
nginx          1/1     Running   0          13m     app1=dev,app=prod
rs-app-886gw   1/1     Running   0          5m18s   app=practice,mode=dev
rs-app-lh9h2   1/1     Running   0          5m18s   app=practice,mode=dev
rs-app-rbbds   1/1     Running   0          5m18s   app=practice,mode=dev
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ kubectl label po nginx app2=dev
pod/nginx labeled
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ kubectl get po --show-labels
NAME           READY   STATUS    RESTARTS   AGE     LABELS
nginx          1/1     Running   0          13m     app1=dev,app2=dev,app=prod
rs-app-886gw   1/1     Running   0          5m36s   app=practice,mode=dev
rs-app-lh9h2   1/1     Running   0          5m36s   app=practice,mode=dev
rs-app-rbbds   1/1     Running   0          5m36s   app=practice,mode=dev
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ 
```

OVERWRITE EXISTING LABEL

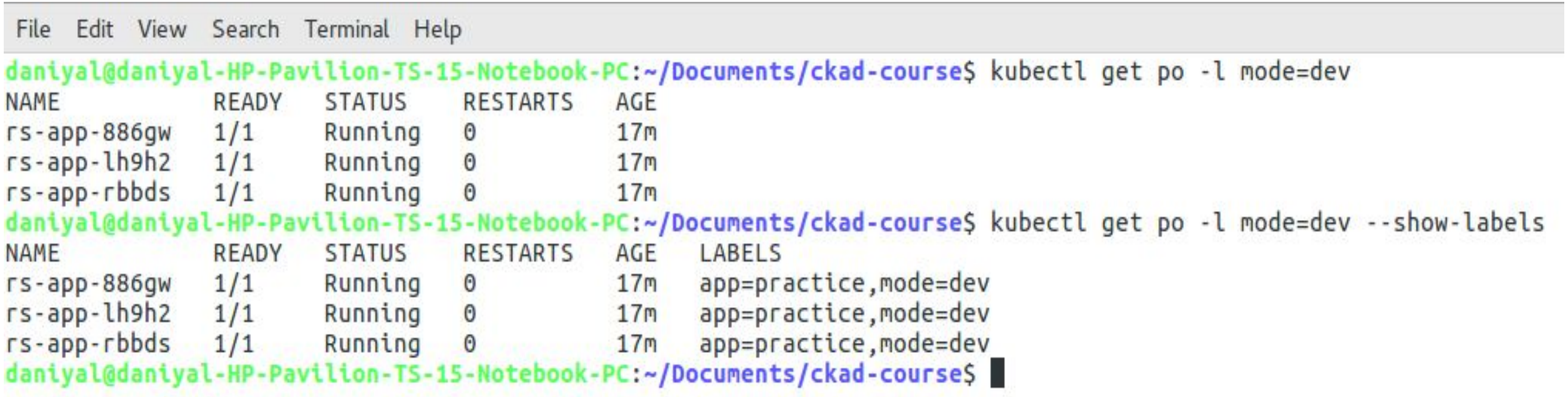
- Overwrite to existing labels



```
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ kubectl get po --show-labels
NAME           READY   STATUS    RESTARTS   AGE     LABELS
nginx          1/1     Running   0          16m     app1=dev,app2=dev,app=prod
rs-app-886gw   1/1     Running   0          8m58s   app=practice,mode=dev
rs-app-lh9h2   1/1     Running   0          8m58s   app=practice,mode=dev
rs-app-rbbds   1/1     Running   0          8m58s   app=practice,mode=dev
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ kubectl label po nginx app2=testing
error: 'app2' already has a value (dev), and --overwrite is false
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ kubectl label po nginx app2=testing --overwrite=true
pod/nginx labeled
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ kubectl get po --show-labels
NAME           READY   STATUS    RESTARTS   AGE     LABELS
nginx          1/1     Running   0          17m     app1=dev,app2=testing,app=prod
rs-app-886gw   1/1     Running   0          9m16s   app=practice,mode=dev
rs-app-lh9h2   1/1     Running   0          9m16s   app=practice,mode=dev
rs-app-rbbds   1/1     Running   0          9m16s   app=practice,mode=dev
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ 
```

LABELS

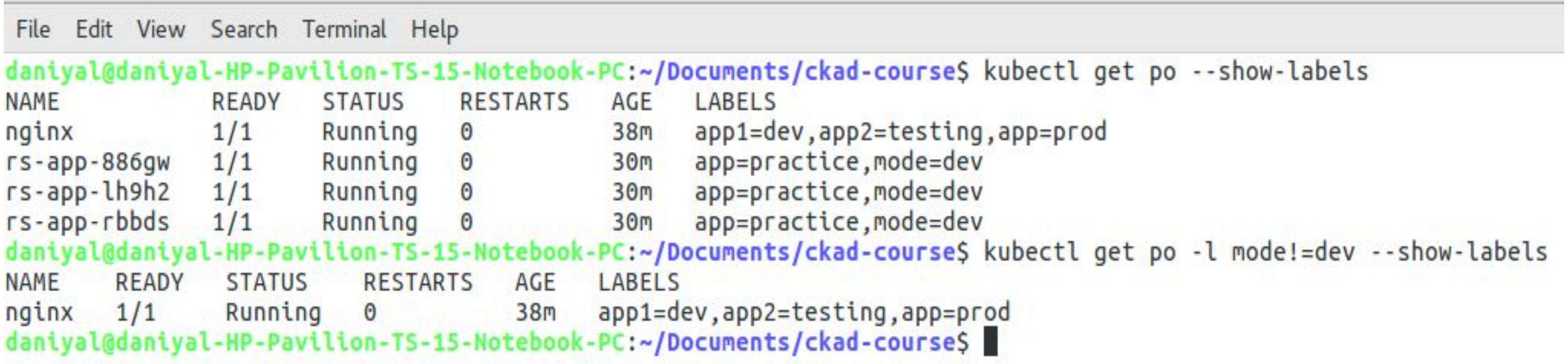
- Contains a multiple labels with a certain key/value with --show-labels flag and without flag.



```
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ kubectl get po -l mode=dev
NAME            READY   STATUS    RESTARTS   AGE
rs-app-886gw    1/1     Running   0          17m
rs-app-lh9h2    1/1     Running   0          17m
rs-app-rbbds    1/1     Running   0          17m
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ kubectl get po -l mode=dev --show-labels
NAME            READY   STATUS    RESTARTS   AGE   LABELS
rs-app-886gw    1/1     Running   0          17m   app=practice,mode=dev
rs-app-lh9h2    1/1     Running   0          17m   app=practice,mode=dev
rs-app-rbbds    1/1     Running   0          17m   app=practice,mode=dev
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ 
```

LABELS

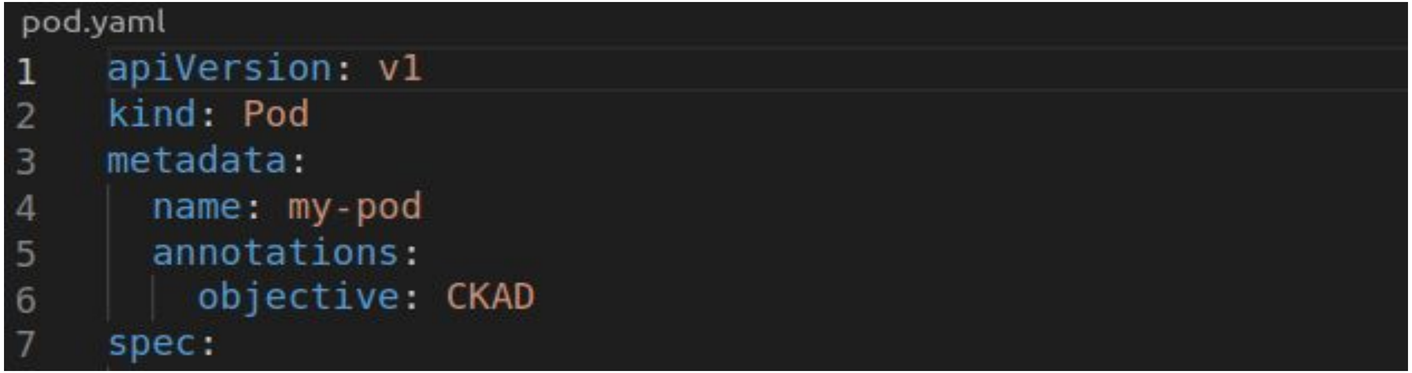
- Does NOT contains a label value against certain key.



```
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ kubectl get po --show-labels
NAME           READY   STATUS    RESTARTS   AGE   LABELS
nginx          1/1     Running   0          38m   app1=dev,app2=testing,app=prod
rs-app-886gw   1/1     Running   0          30m   app=practice,mode=dev
rs-app-lh9h2   1/1     Running   0          30m   app=practice,mode=dev
rs-app-rbbds   1/1     Running   0          30m   app=practice,mode=dev
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ kubectl get po -l mode!=dev --show-labels
NAME    READY   STATUS    RESTARTS   AGE   LABELS
nginx   1/1     Running   0          38m   app1=dev,app2=testing,app=prod
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ 
```

ANNOTATIONS

- Kubernetes annotations to attach arbitrary non-identifying metadata to objects.
- Annotations are used to record other details for informatory purpose.
- For example, tool details like name, version, build, information etc or contact details, phone numbers, emails, IDs etc that may be used for some kind of integration purpose.

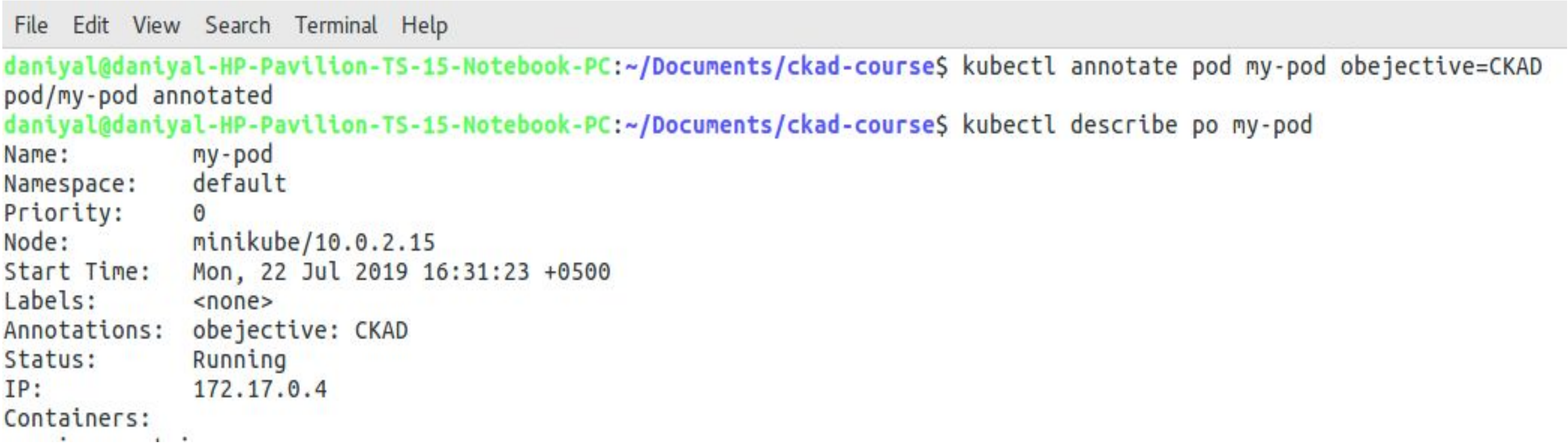


pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  annotations:
    objective: CKAD
spec:
```

ANNOTATIONS

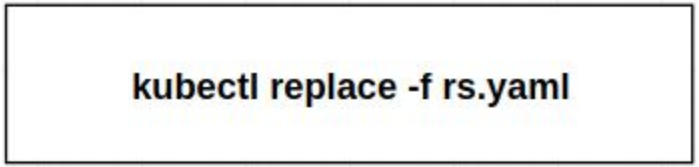
- Annotate the running POD using command line.



```
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ kubectl annotate pod my-pod obejective=CKAD
pod/my-pod annotated
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ kubectl describe po my-pod
Name:         my-pod
Namespace:    default
Priority:     0
Node:         minikube/10.0.2.15
Start Time:   Mon, 22 Jul 2019 16:31:23 +0500
Labels:       <none>
Annotations:  obejective: CKAD
Status:       Running
IP:           172.17.0.4
Containers:
```

SCALE THE REPLICA SET

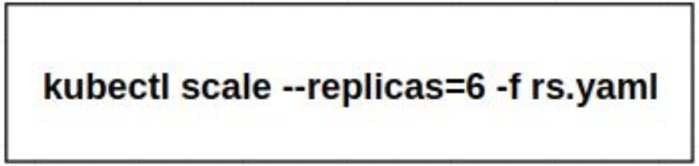
- Let’s now look at how we scale the Replica set, Say we started with three replicas and in the future we decided to scale to six.
- How do we update our replica set to scale to six replicas. Well there are multiple ways to do it.
- The first is to update the number of replicas in the definition file to 6.
- Then run the command that will update the replica set to have six replicas.



```
kubectl replace -f rs.yaml
```

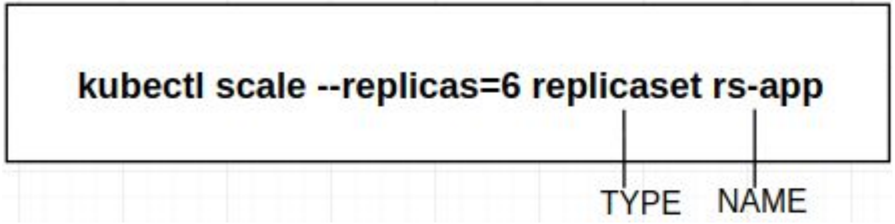
SCALE THE REPLICA SET

- The second way to do it is to run kubectl scale command use the replica's parameters to provide the new number of replicas and specify the same file as input.



```
kubectl scale --replicas=6 -f rs.yaml
```

- You may either input the definition file or provide the replica set name in the type name format.



```
kubectl scale --replicas=6 replicaset rs-app
```

NAMESPACES

What is namespace?

How to switch namespaces?

NAMESPACES

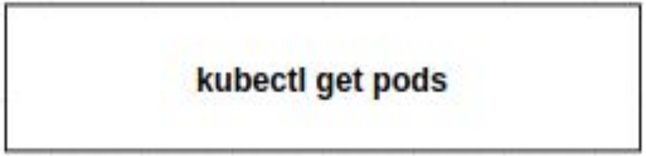
- In Kubernetes, all objects such as pods, services, volumes, etc… are part of a namespace.
- If you do not specify a namespace when creating or viewing your objects, they will be created in the “default” namespace.
- Whatever we have been doing, we have been doing within a namespace.
- When the cluster is first setup kubernetes automatically creates a namespace that is known as “default” namespace.
- Kubernetes creates a set of pods and services for its internal purpose such as those required by the networking solution the DNS service etc.
- To isolate from the user and to prevent you from accidentally deleting or modifying the services, kubernetes creates them under another namespace created at cluster startup named kube-system.

NAMESPACES

- The third namespace created by kubernetes automatically is called kube-public.
- This is where resources that should be made available to all users are created.
- You can create your own namespaces. If you wanted to use the same cluster for both Dev and Production environment.
- Each of these namespace can have its own set of policies that define who can do what.
- You can also assign quota of resources to each of these namespaces. That way each namespace is guaranteed a certain amount and does not use more than its allowed limit.
- Let us now look at some of the operational aspects of namespaces.

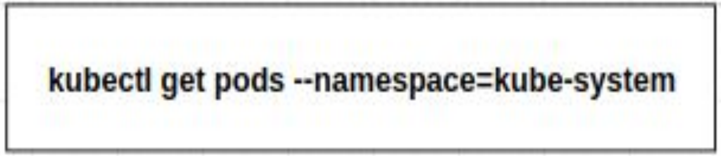
NAMESPACES

- Start with commands:



```
kubectl get pods
```

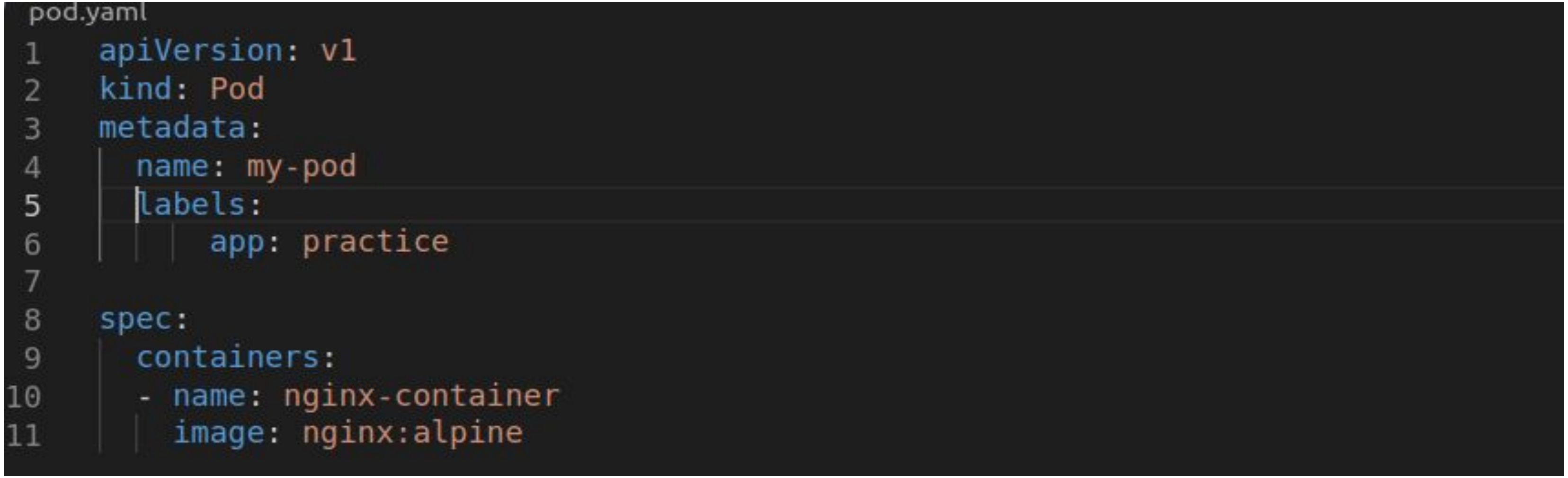
- The above is get the list of all the PODs. But it only lists the PODs in the “default” namespace.
- To list PODs in another namespace use the namespace option in the command along with the name of the namespace. In this case “kube-system”
- For example:



```
kubectl get pods --namespace=kube-system
```

NAMESPACES

- Here I have a POD definition file:



pod.yaml

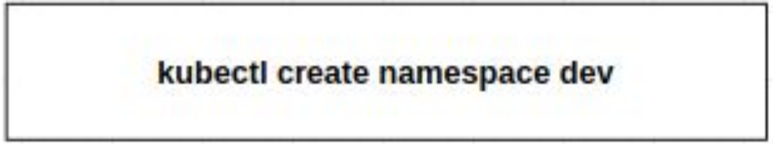
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  labels:
    app: practice

spec:
  containers:
  - name: nginx-container
    image: nginx:alpine
```

```
kubectl create -f pod.yaml
```

- When you create a POD using this file, the pod is created in the default namespace.

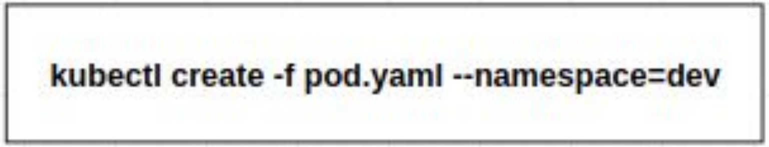
NAMESPACES

- To create a namespace use this command.



```
kubectl create namespace dev
```

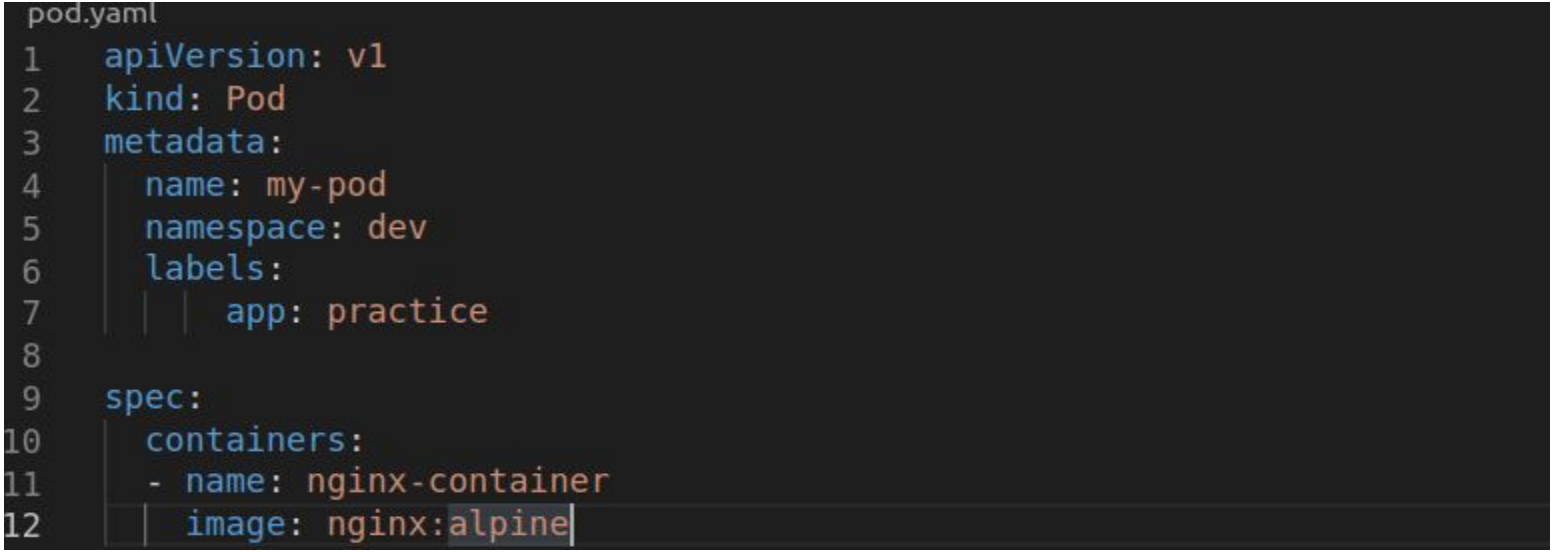
- To create a POD in another namespace. Use the namespace option using below command.



```
kubectl create -f pod.yaml --namespace=dev
```

NAMESPACES

- If you want to make sure that this POD gets created in that Dev Environment all the time even if you don’t specify the namespace in the command line for that you can move the namespace definition into the POD definition file.
- Like this under the metadata section:

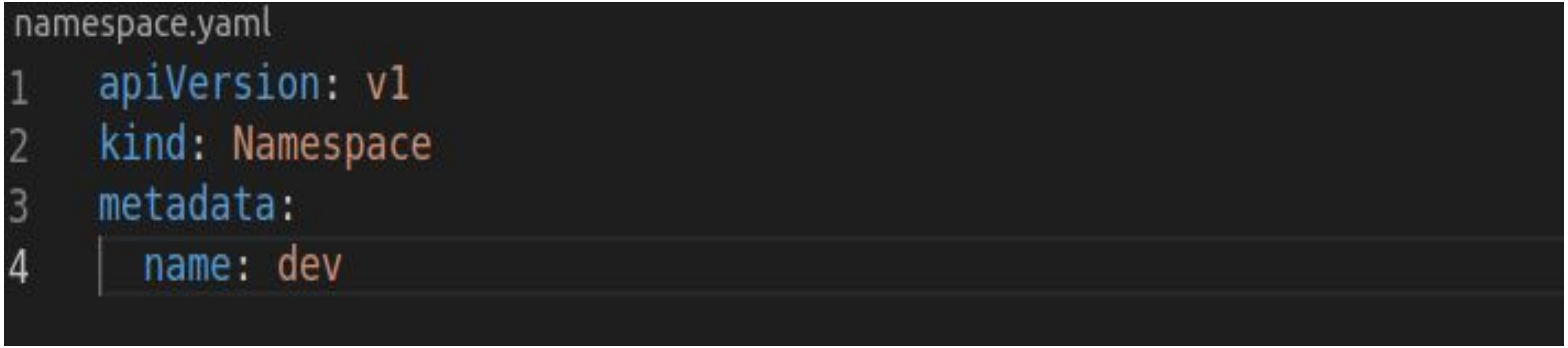


pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  namespace: dev
  labels:
      app: practice

spec:
  containers:
  - name: nginx-container
    image: nginx:alpine
```

NAMESPACES

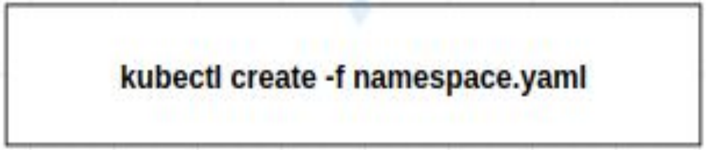
- This is a good way to ensure your resources are always created in the same namespace.
- Now create a namespace using declarative way.



namespace.yaml

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: dev
```

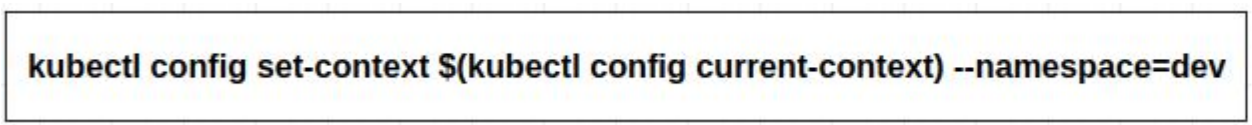
- The API version is v1, kind is namespace and under metadata specify the name. In this case dev. Run the below command to create a namespace.



```
kubectl create -f namespace.yaml
```

NAMESPACES

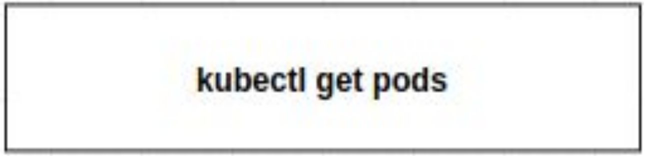
- Let suppose we have three namespaces.
- As we have discussed before by default we are in the default namespace, which is why we can see the resources inside the default namespace using the kubectl get pods command and to view those in dev namespace, We have to use the namespace option.
- But what if we want to switch to the dev namespace permanently so that we don't have to specify the namespace option anymore.
- Well in this case use the kubectl config command to set the namespace in the current context to dev. You can then simply run the kubectl get pods command without the namespace option to list pods in the dev environment.



```
kubectl config set-context $(kubectl config current-context) --namespace=dev
```

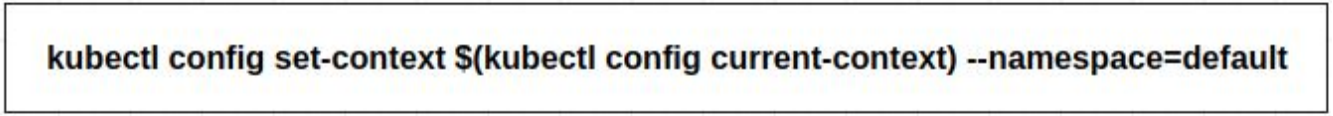
NAMESPACES

- Now simply run get pods command and you will get a list of pods in the dev namespace without specifying the name of the namespace in the command line.



```
kubectl get pods
```

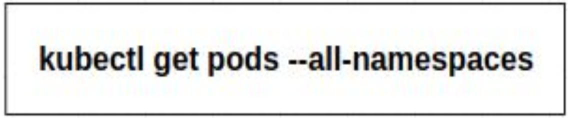
- But you will need to specify the option for other environments such as default.
- Similarly, you can switch to another namespace the same way.



```
kubectl config set-context $(kubectl config current-context) --namespace=default
```

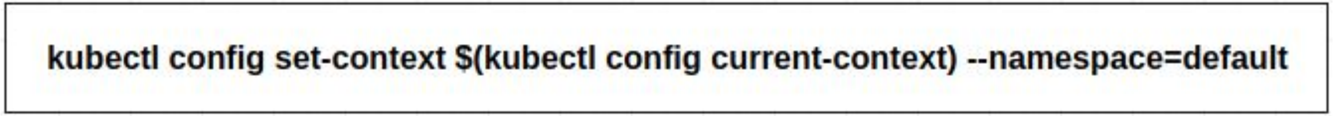
NAMESPACES

- Finally to view PODs in all namespaces use the all namespaces option in the command.
- This will list all the pods in all of the namespaces.



```
kubectl get pods --all-namespaces
```

Taking a closer look at the below command this command first identifies the current context and then sets the namespace to the desired one for that current context.

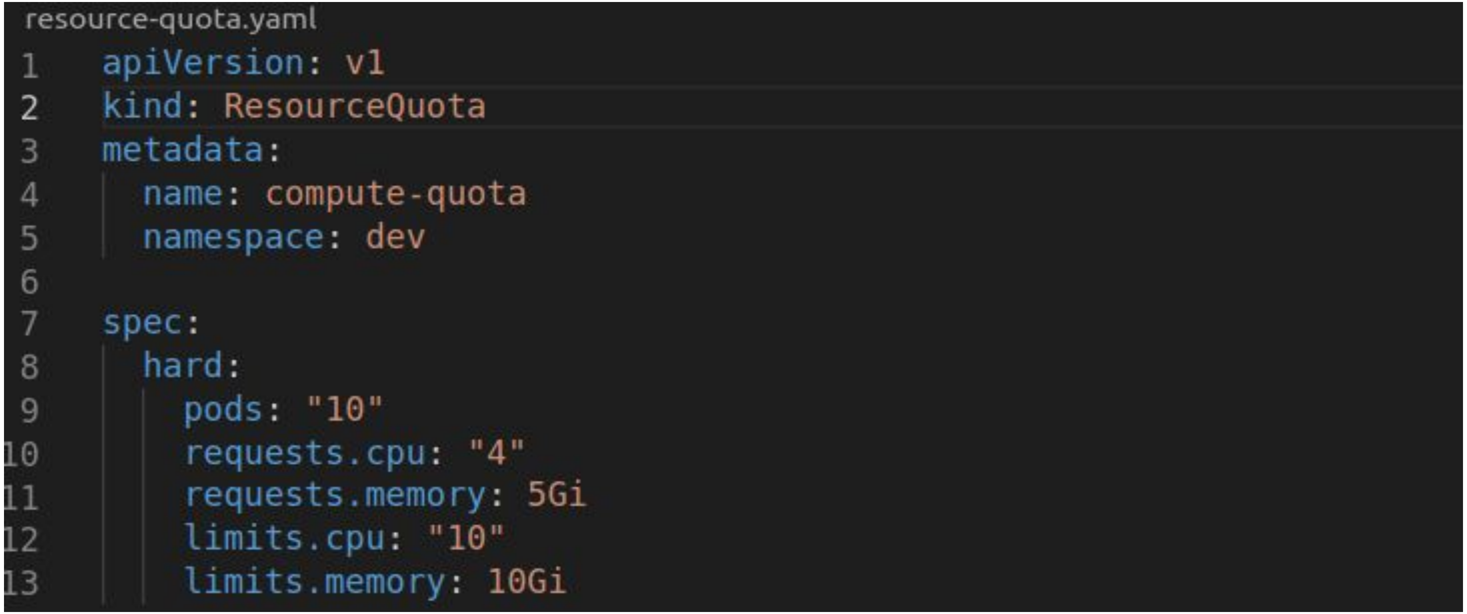


```
kubectl config set-context $(kubectl config current-context) --namespace=default
```

RESOURCE QUOTA

RESOURCE QUOTA

- To limit resources in a namespace, create a resource quota.
- To create one, Start with a definition file for resource quota.



resource-quota.yaml

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-quota
  namespace: dev

spec:
  hard:
    pods: "10"
    requests.cpu: "4"
    requests.memory: 5Gi
    limits.cpu: "10"
    limits.memory: 10Gi
```

RESOURCE QUOTA

- Specify the namespace for which you want to create a quota and then under spec provide your limits such as 10 PODs, 10 CPU units, 10 GB of memory etc.

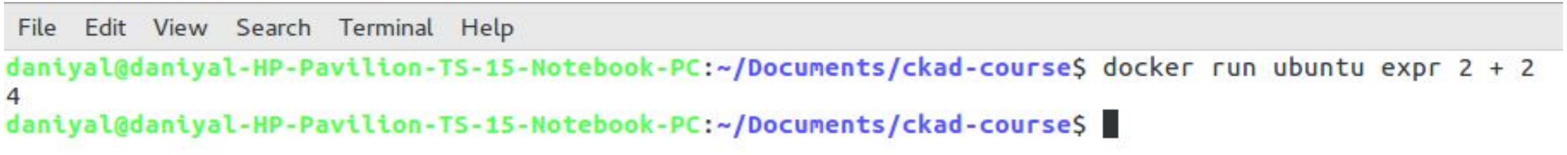
JOB

JOB

- There are different types of workloads that a container can serve.
- If we deploy an application it runs for a long period of time until manually taken down.
- For example, performing a computation, processing an image, performing some kind of analytics on a large data set, generating a report and sending an email etc.
- These are workloads that are meant to live for a short period of time.
- Performs a set of tasks and then finish. Let us first see how such a workload works in docker and then we will relate the same concept to kubernetes.

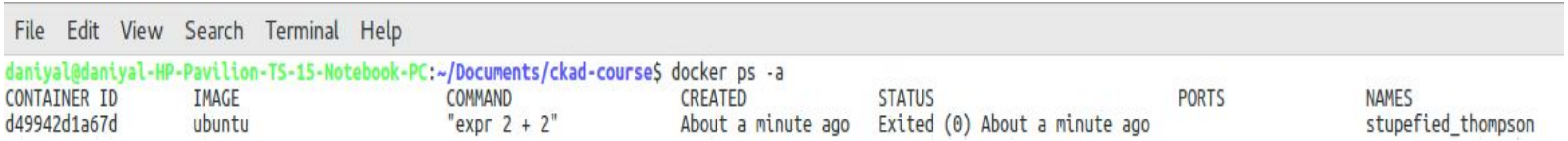
JOB

- So i am going to run a docker container using command **docker run ubuntu expr 2 + 2.**



```
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ docker run ubuntu expr 2 + 2
4
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ 
```

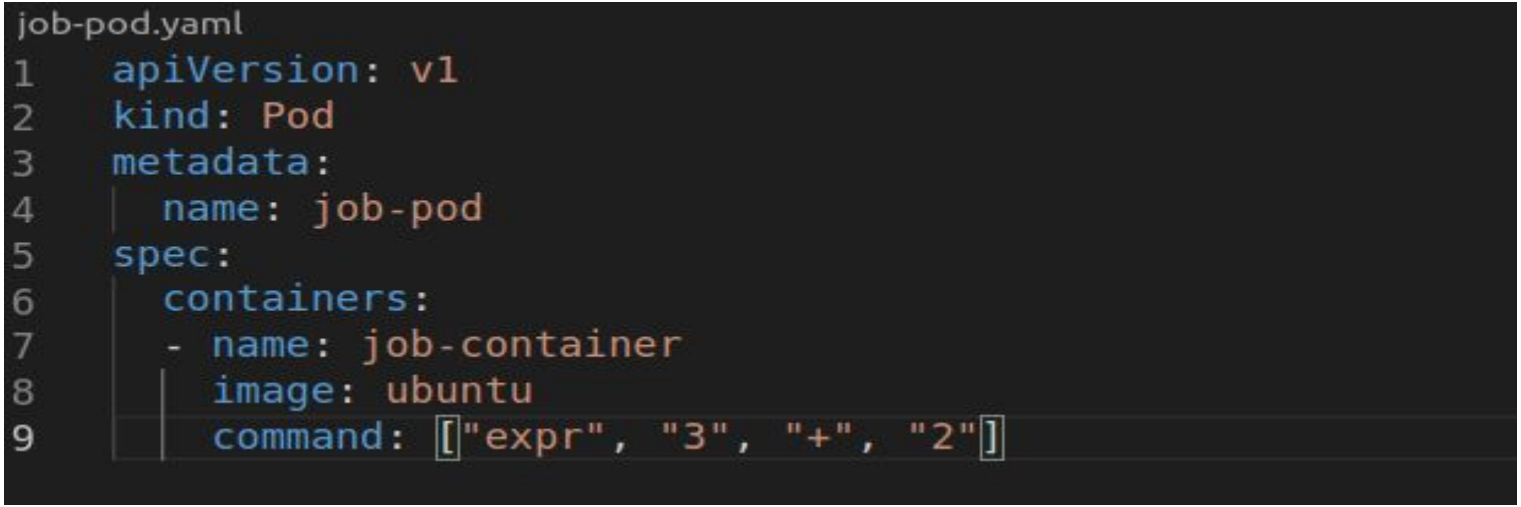
- The docker container comes up perform the requested operation prints the output and then exits.
- When you run the docker ps command you see the container exited state.



```
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED              STATUS                          PORTS               NAMES
d49942d1a67d        ubuntu              "expr 2 + 2"        About a minute ago   Exited (0) About a minute ago                       stupefied_thompson
```

JOB

- Lets replicate it in the kubernetes. We create a pod definition file to perform the same operation.



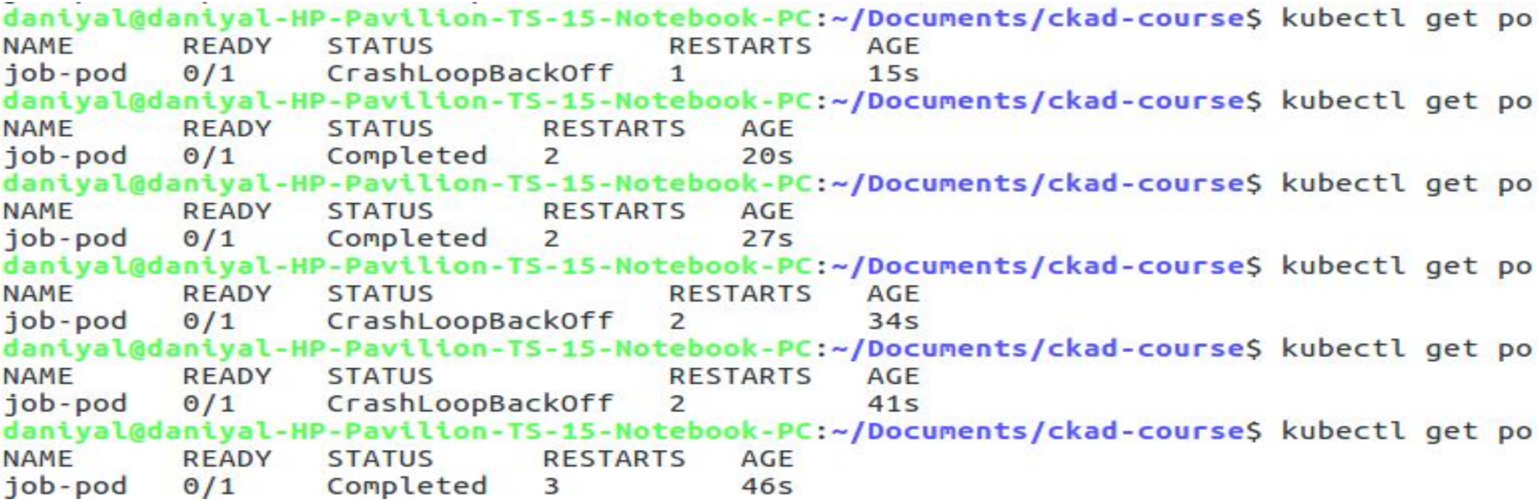
job-pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: job-pod
spec:
  containers:
  - name: job-container
    image: ubuntu
    command: ["expr", "3", "+", "2"]
```

JOB

- When the pod is created it runs the container perform the computation task and then exit and the pod goes into a completed state.

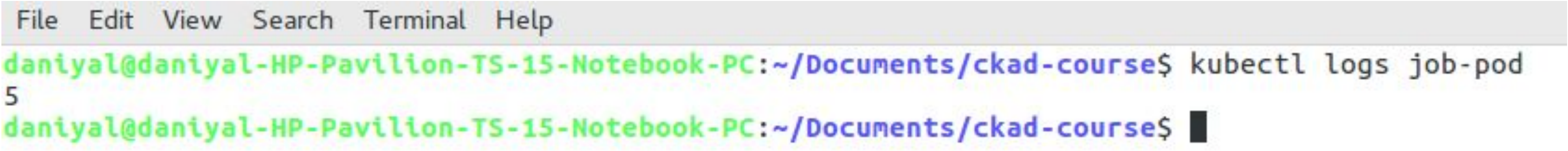
- But it then recreates the container in an attempt to leave it running. Again the container perform the computation task and then exits in the below screenshot you can see **RESTARTS** increased.



```
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ kubectl get po
NAME      READY   STATUS             RESTARTS   AGE
job-pod   0/1     CrashLoopBackOff   1          15s
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ kubectl get po
NAME      READY   STATUS      RESTARTS   AGE
job-pod   0/1     Completed   2          20s
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ kubectl get po
NAME      READY   STATUS      RESTARTS   AGE
job-pod   0/1     Completed   2          27s
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ kubectl get po
NAME      READY   STATUS             RESTARTS   AGE
job-pod   0/1     CrashLoopBackOff   2          34s
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ kubectl get po
NAME      READY   STATUS             RESTARTS   AGE
job-pod   0/1     CrashLoopBackOff   2          41s
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ kubectl get po
NAME      READY   STATUS      RESTARTS   AGE
job-pod   0/1     Completed   3          46s
```

JOB

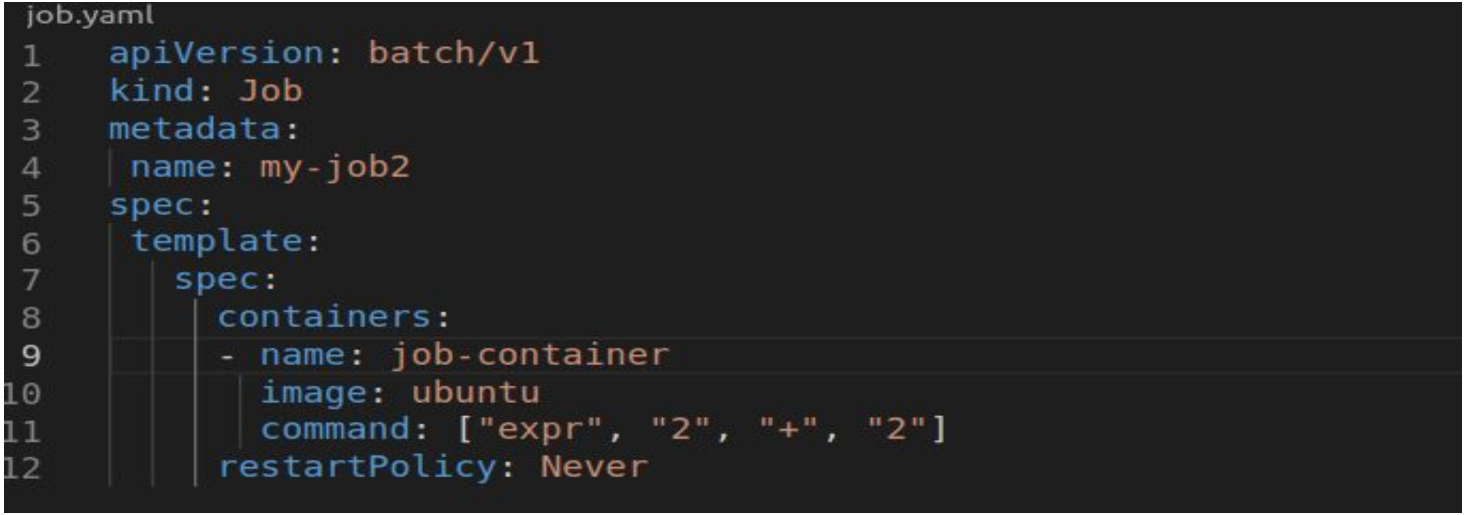
- This behaviour is defined by the property restart policy set on the pod which is default set to always and that is why pod always recreate the container when it exits.
- You can overwrite this property to never. Kubernetes does not recreate the container once the job is finished.
- You can see the logs using kubectl logs command with the name of the pod.



```
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ kubectl logs job-pod
5
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ 
```

JOB

- Let’s create a JOB using YAML.

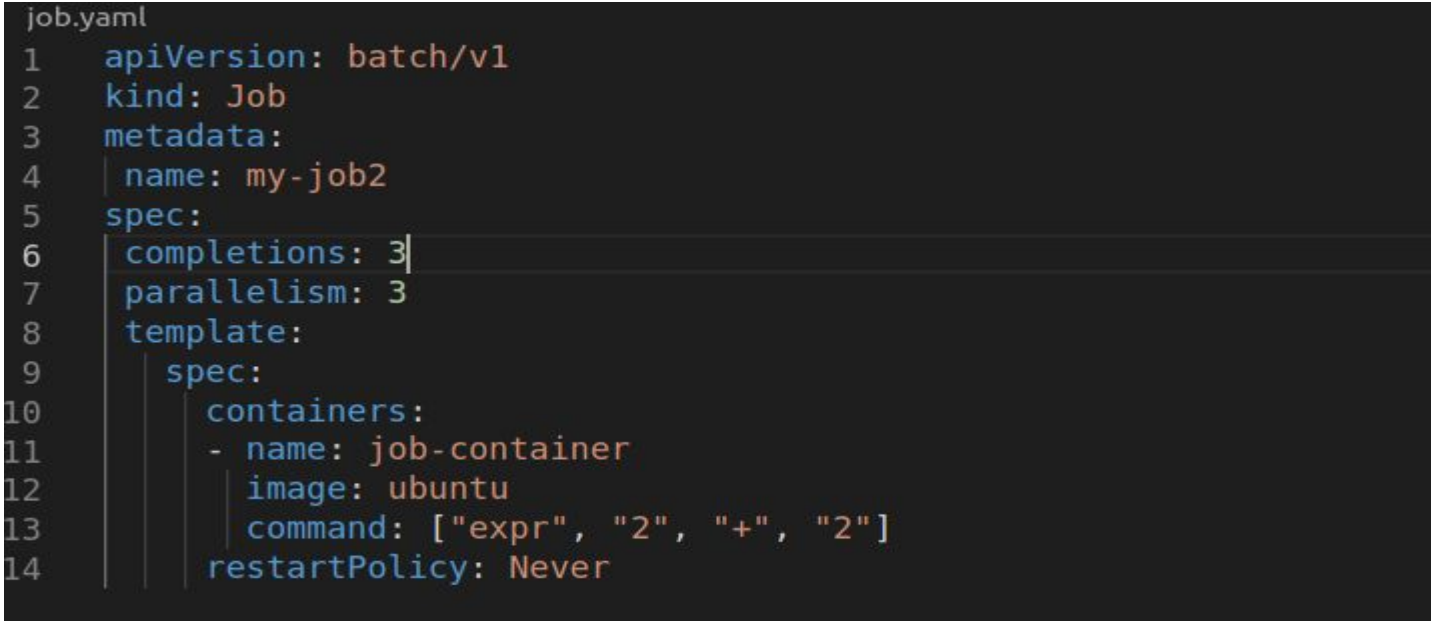


```yaml
apiVersion: batch/v1
kind: Job
metadata:
 name: my-job2
spec:
 template:
   spec:
     containers:
     - name: job-container
       image: ubuntu
       command: ["expr", "2", "+", "2"]
     restartPolicy: Never
```

- The **restartPolicy** property you can see in the above yaml this is mandatory property for job because by default **restartPolicy** is Always and JOB don’t support Always. JOB supported values: "OnFailure" and "Never”.

JOB

- Now add few more properties in the JOB YAML file.



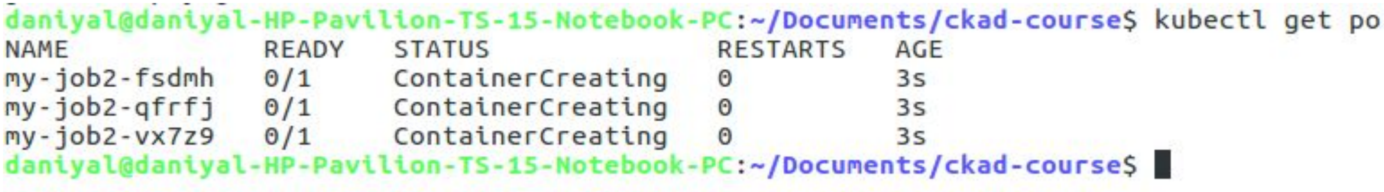
job.yaml

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: my-job2
spec:
  completions: 3
  parallelism: 3
  template:
    spec:
      containers:
      - name: job-container
        image: ubuntu
        command: ["expr", "2", "+", "2"]
      restartPolicy: Never
```

- Completions property means the job will execute three times once its completed then second one is execute.

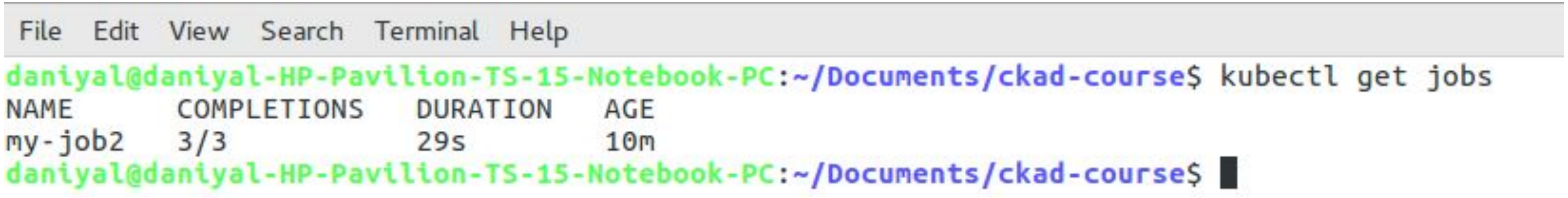
JOB

- Parallelism property means all three job executes at a same time.



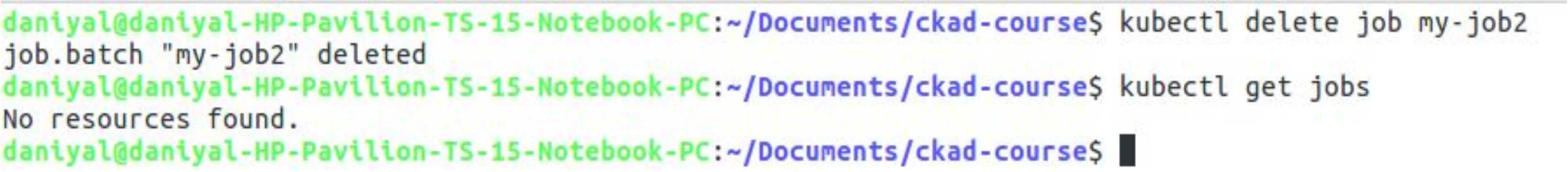
```
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ kubectl get po
NAME             READY   STATUS              RESTARTS   AGE
my-job2-fsdmh    0/1     ContainerCreating   0          3s
my-job2-qfrfj    0/1     ContainerCreating   0          3s
my-job2-vx7z9    0/1     ContainerCreating   0          3s
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$
```

- To view the list of the jobs.



```
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ kubectl get jobs
NAME      COMPLETIONS   DURATION   AGE
my-job2   3/3           29s        10m
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$
```

- To delete the jobs.



```
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ kubectl delete job my-job2
job.batch "my-job2" deleted
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ kubectl get jobs
No resources found.
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$
```

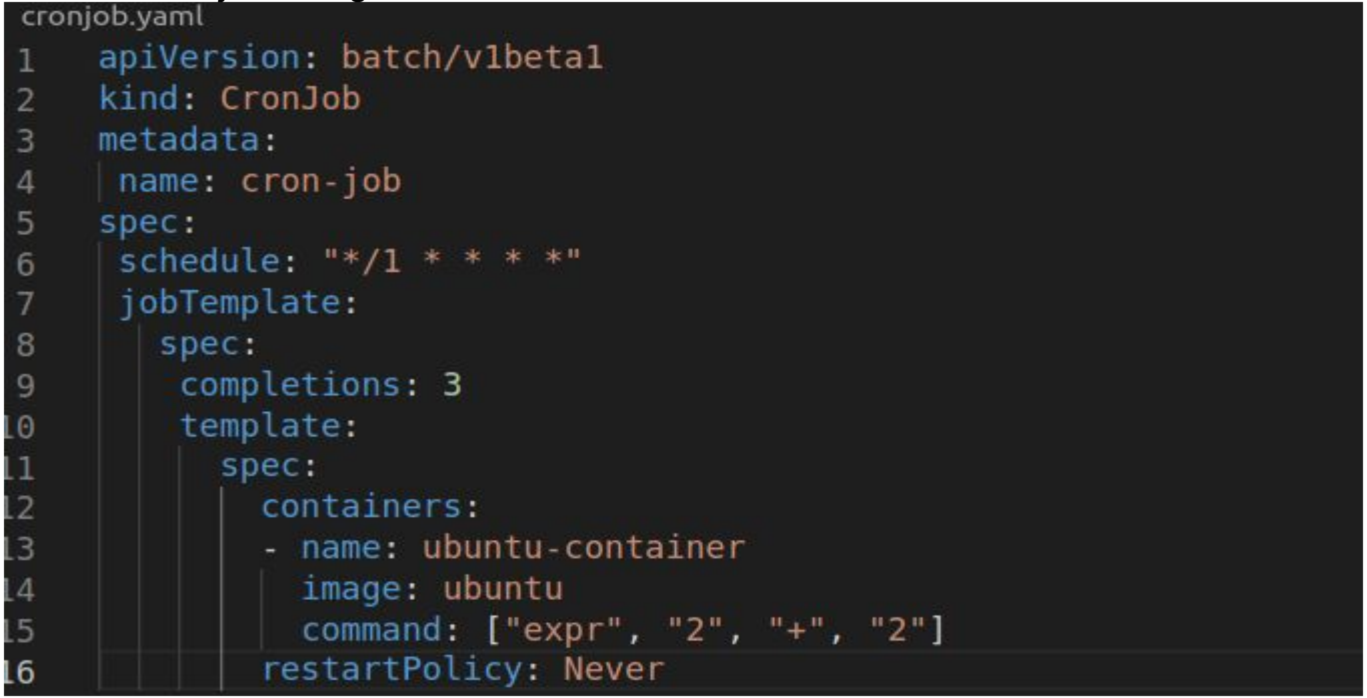
CRON JOB

CRON JOB

- CronJobs to run jobs on a time-based schedule. These automated jobs run like Cron tasks on a Linux or UNIX system.
- Cron jobs are useful for creating periodic and recurring tasks, like running backups or sending emails.
- Cron jobs can also schedule individual tasks for a specific time, such as if you want to schedule a job for a low activity period.
- It works almost similar to Kubernetes Job resource.

CRON JOB

- Let’s create a cron job using YAML definition file.



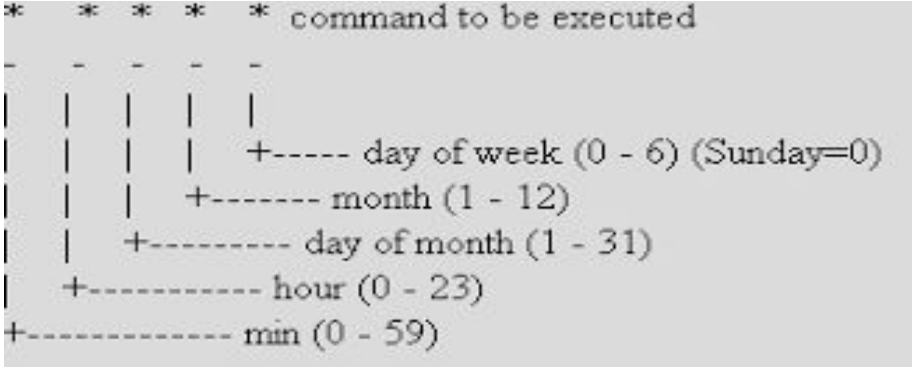
cronjob.yaml

```yaml
apiVersion: batch/v1beta1
kind: CronJob
metadata:
 name: cron-job
spec:
 schedule: "*/1 * * * *"
 jobTemplate:
   spec:
    completions: 3
    template:
      spec:
        containers:
        - name: ubuntu-container
          image: ubuntu
          command: ["expr", "2", "+", "2"]
        restartPolicy: Never
```

- According to the above YAML file the job will execute every minute.

CRON JOB

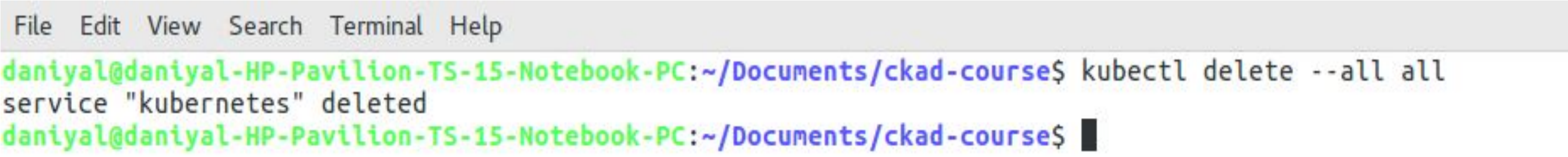
- The above YAML file is very similar to the Job definition file just we have two additional properties here.
- One is **schedule** and second one is **job template**.
- The **schedule** is a required field of the **spec**. It takes a Cron format string, such as **0** * * * *or **@hourly**, as schedule time of its jobs to be created and executed. Below is the format.



```
*    *   *   *    *  command to be executed
-    -   -   -    -
|    |   |   |    |
|    |   |   |    +----- day of week (0 - 6) (Sunday=0)
|    |   |   +------- month (1 - 12)
|    |   +--------- day of month (1 - 31)
|    +----------- hour (0 - 23)
+------------- min (0 - 59)
```

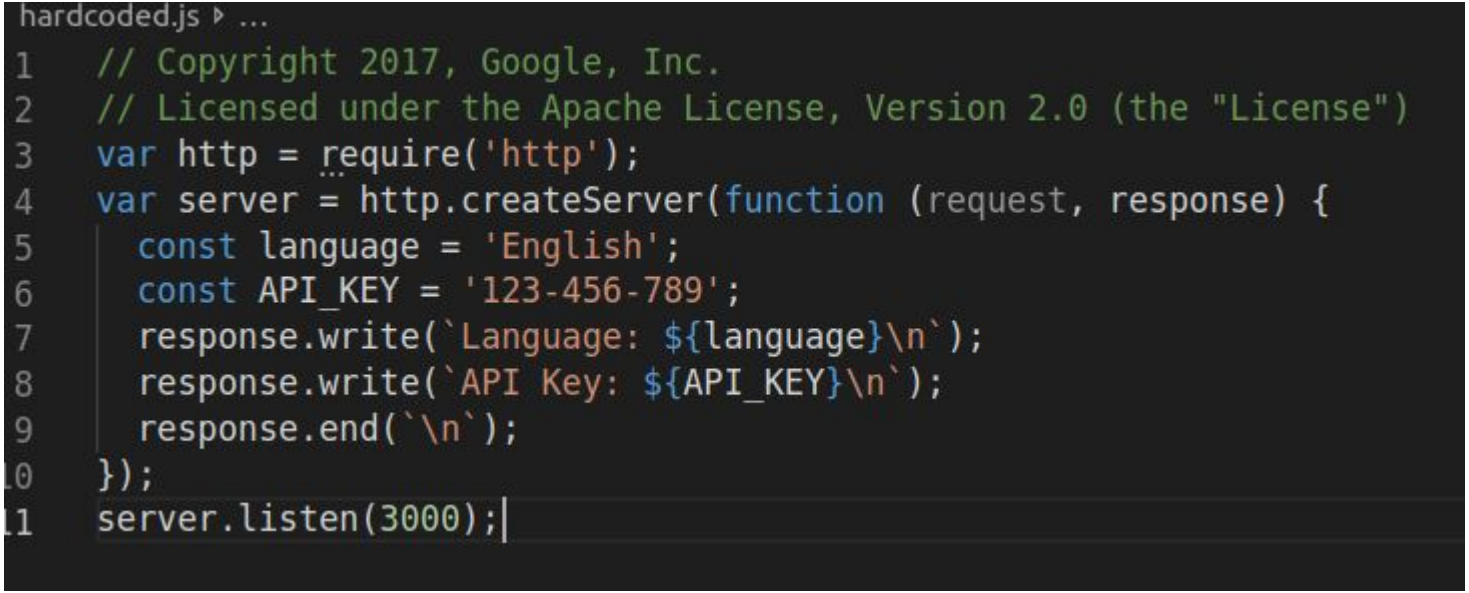
KUBECTL COMMANDS

- To delete all the resources in kubernetes. Run the below command.



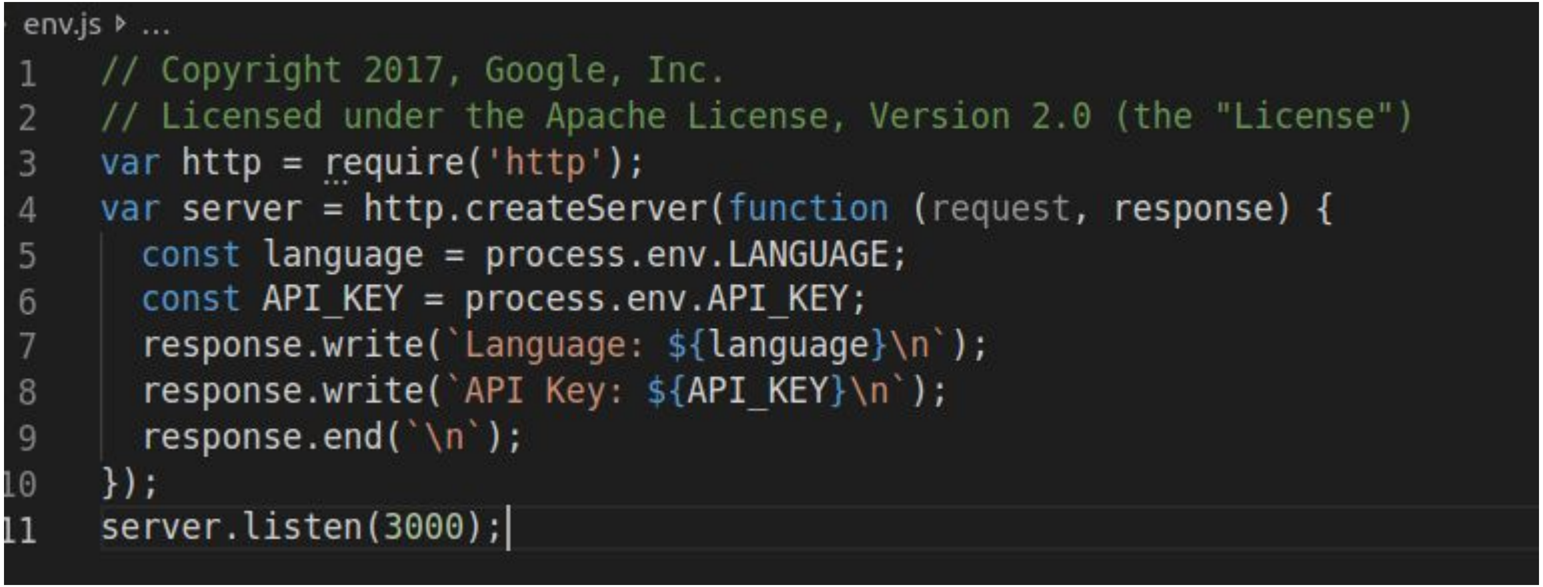
```
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ kubectl delete --all all
service "kubernetes" deleted
daniyal@daniyal-HP-Pavilion-TS-15-Notebook-PC:~/Documents/ckad-course$ 
```

CONFIG MAP AND SECRET

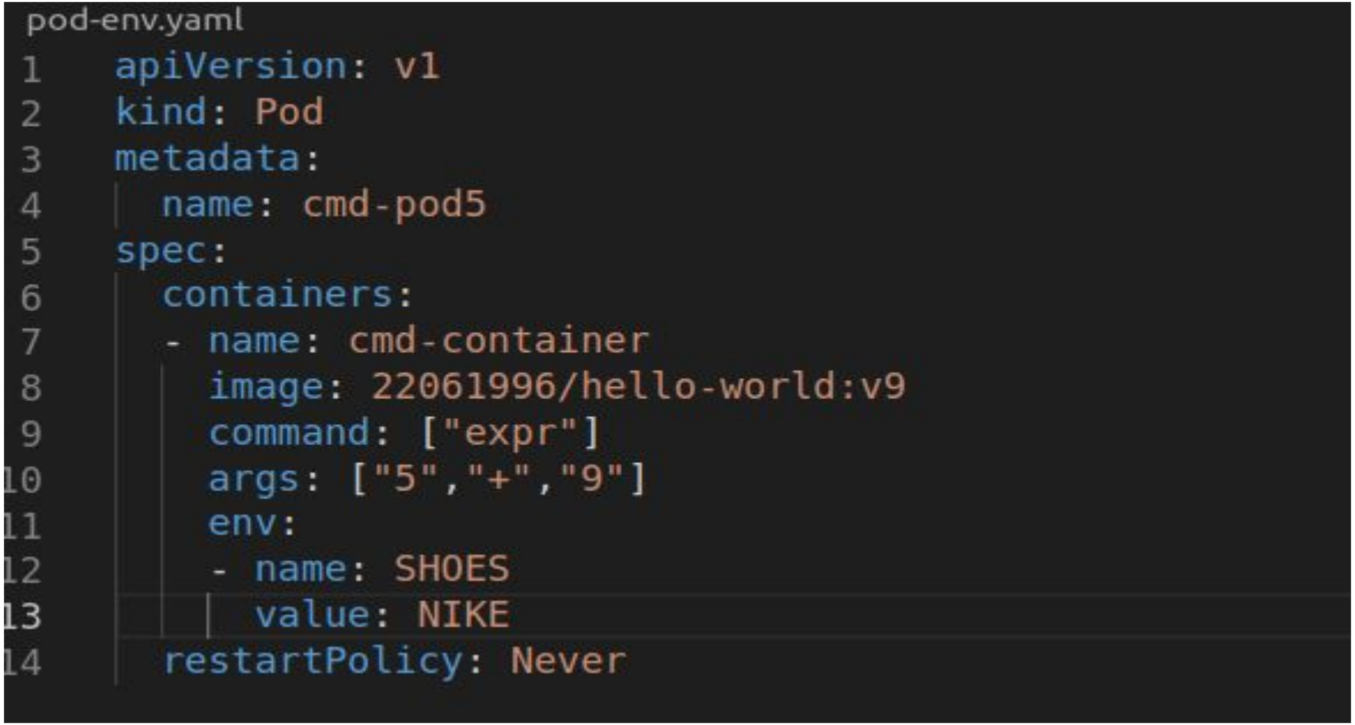


```
// Copyright 2017, Google, Inc.
// Licensed under the Apache License, Version 2.0 (the "License")
var http = require('http');
var server = http.createServer(function (request, response) {
  const language = 'English';
  const API_KEY = '123-456-789';
  response.write(`Language: ${language}\n`);
  response.write(`API Key: ${API_KEY}\n`);
  response.end(`\n`);
});
server.listen(3000);
```

CONFIG MAP AND SECRET



```javascript
// Copyright 2017, Google, Inc.
// Licensed under the Apache License, Version 2.0 (the "License")
var http = require('http');
var server = http.createServer(function (request, response) {
  const language = process.env.LANGUAGE;
  const API_KEY = process.env.API_KEY;
  response.write(`Language: ${language}\n`);
  response.write(`API Key: ${API_KEY}\n`);
  response.end(`\n`);
});
server.listen(3000);
```



pod-env.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: cmd-pod5
spec:
  containers:
  - name: cmd-container
    image: 22061996/hello-world:v9
    command: ["expr"]
    args: ["5","+","9"]
    env:
    - name: SHOES
      value: NIKE
  restartPolicy: Never
```

CONFIG MAP

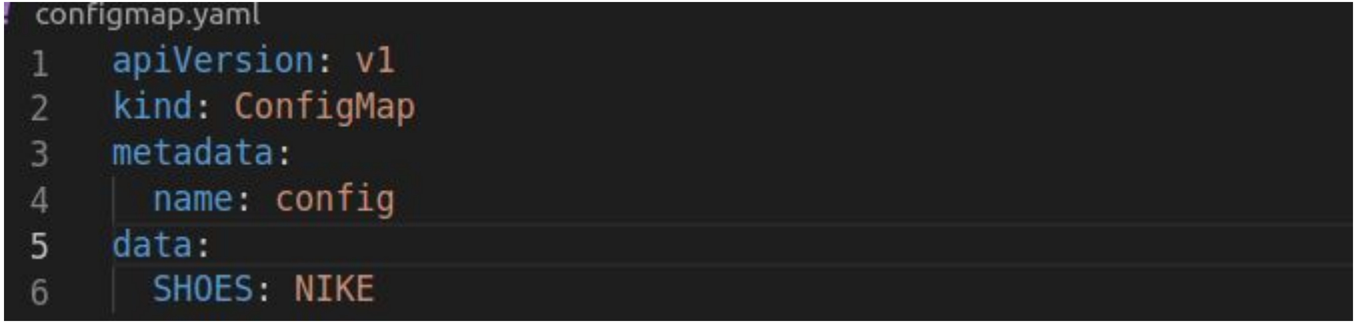
Create config map using imperative way:

```
kubectl create configmap <configmap-name> --from-literal=<key>=<value>
```

Another way using file:

```
kubectl create configmap <configmap-name> --from-literal=<file- path>
```

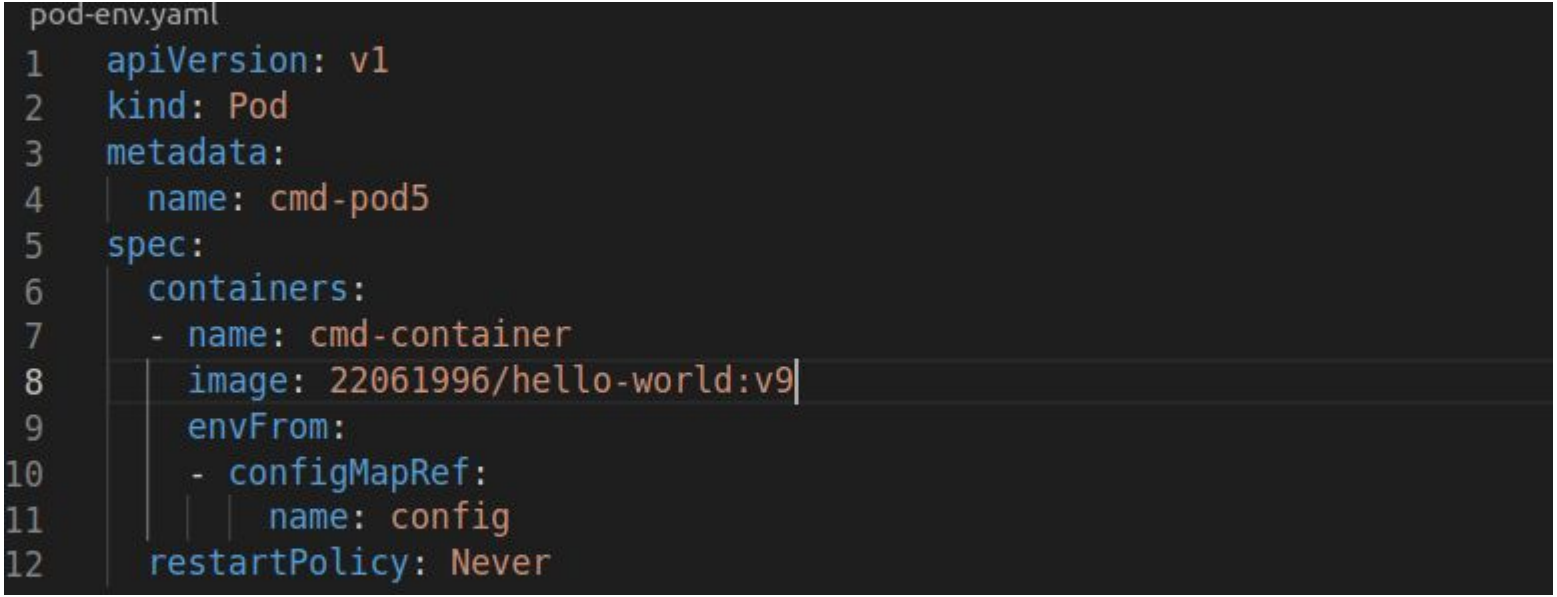
Using declarative way:



configmap.yaml

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: config
data:
  SHOES: NIKE
```

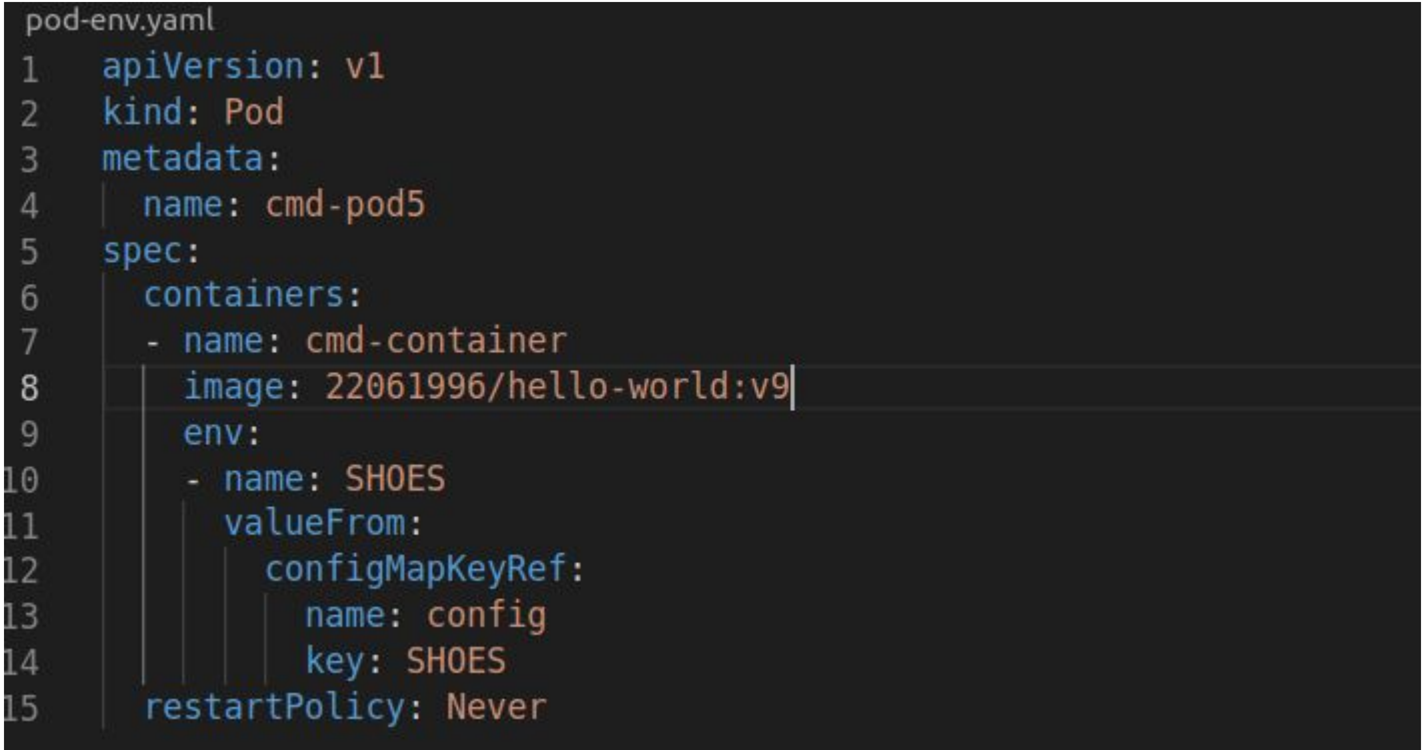
Inject config map environment into the pod.



pod-env.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: cmd-pod5
spec:
  containers:
  - name: cmd-container
    image: 22061996/hello-world:v9
    envFrom:
    - configMapRef:
        name: config
  restartPolicy: Never
```

Inject a single config map environment into the pod.



pod-env.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: cmd-pod5
spec:
  containers:
  - name: cmd-container
    image: 22061996/hello-world:v9
    env:
    - name: SHOES
      valueFrom:
        configMapKeyRef:
          name: config
          key: SHOES
  restartPolicy: Never
```

SECRETS

Secret stores data in encoded or hashed format:

```
echo -n "SHOES" | base6
```

- Run command:
- export LANGUAGE=ENGLISH
- Let’s save the API key as a Secret:

  -
    ```
    kubectl create secret generic apikey --from-literal=API_KEY=123-456
    ```
  -
- And the language as a ConfigMap:

  -
    ```
    kubectl create configmap language --from-literal=LANGUAGE=English
    ```
  -
  -

THANKS

:)