Kubernetes

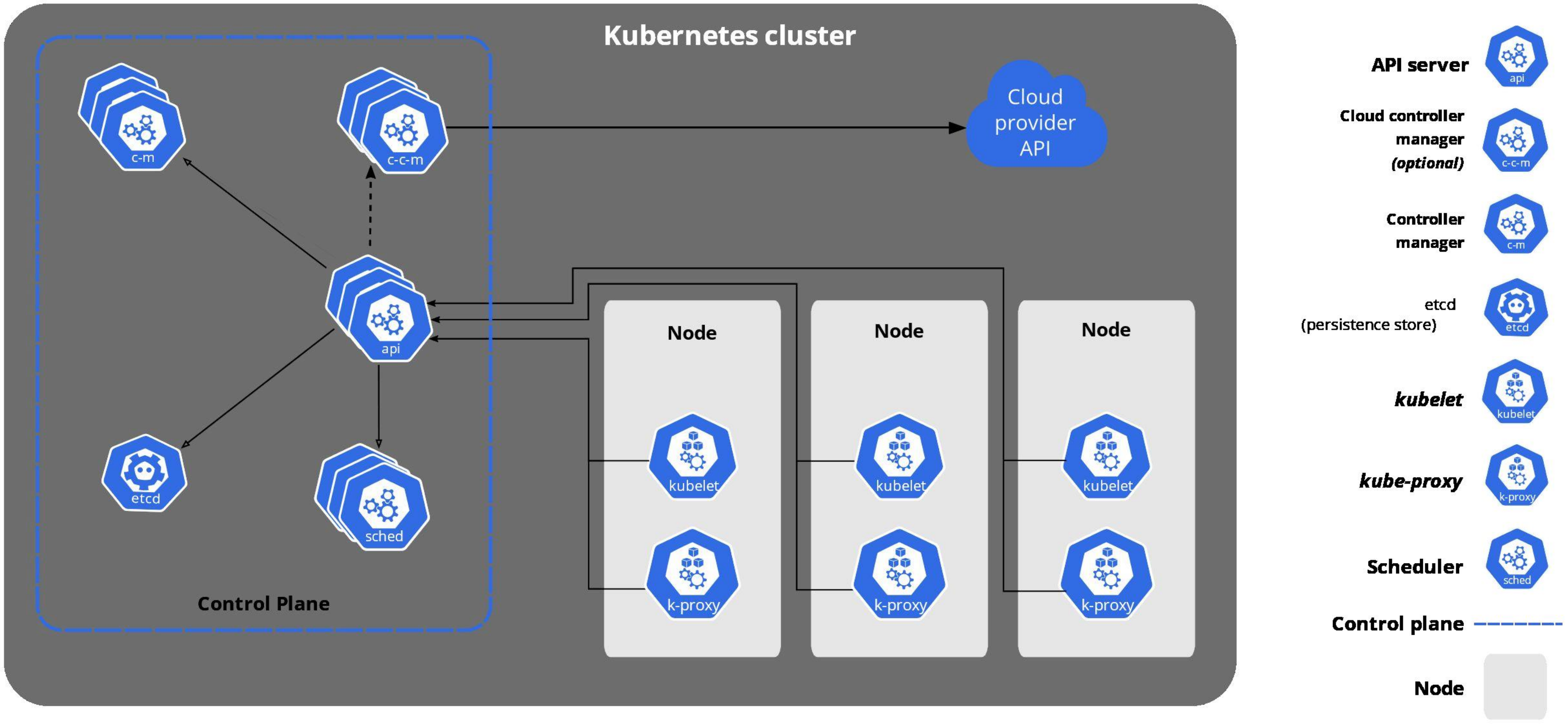
Repo that contains all the work,
https://github.com/james-jasvin/K8s-Repo

Refer for better understanding:
Techworld with Nana

What?

- Kubernetes is an open-source container orchestration system developed by **Google**
- lets you deploy, scale, and manage containerized applications.
- It is also known as **K8s**.

K8s Architecture



Node

- Node is a collection of Pods and it represents a single physical/virtual machine or a collection of IT resources that can run K8s.
- Nodes can be physical machines or virtual machines in a cloud environment, and they can be located in different data centers or regions to ensure high availability and disaster recovery.

Master-Worker Architecture

- Two types of Nodes :
  1. Master Node/Control Plane
  2. Worker Node
- There are Worker Nodes that do the actual work, i.e. run the deployed application and serve requests.
- And the Master Node or the Control Plane is responsible for managing the K8s
- cluster and allowing the user to interact with the cluster.
- Provisioning new Pods, restart of failed Pods, adding a new Node, monitoring, etc

Pod

- Smallest unit of K8s.
- Abstraction over container, so you can work with pods irrespective of the underlying container technology like Docker, AWS ECS, etc.
- You only interact with the Kubernetes components and not with the underlying container runtime.
- Usually you run 1 application per pod, so the main application container plus any sidecar containers (or general required containers).
- To recap, the K8s Cluster consists of Nodes run Pods.

Pods - Communication

- Pods communicate with each other via a virtual network that Kubernetes sets up.
- Each Pod has its **own IP address** via which it can communicate with the other Pods in the same Node.
- Pods are ephemeral (so they can die very quickly due to one reason or the other)
- and upon failure, K8s will automatically restart it but now it’ll have a new IP address assigned.

But how would communication happen via temporary IP addresses?

Service

- It is a permanent IP address that’s associated with a pod as the lifecycle of the Pod and its Service are not connected.
- There’ll be external and internal services depending on where you want the Pods to be accessible, outside the Node and inside the Node.
- There’s much more to discuss in Services but that’ll be done later.

Worker Node

- Must have the following 3 processes installed on each Pod for it to work,

1. Container Runtime: Docker, ECS, etc..
2. Kubelet
    - Interface between the K8s Node and Container Runtime.
    - It starts the Pod with the container inside and is responsible for assigning resources from the K8s Node to the container.
3. Kube Proxy
    - Responsible for forwarding requests from the Services to the Pods.
    - It has intelligent logic inside which ensures that the forwarding happens in a performant way to avoid overheads, such as network overhead, i.e. forward request to Pod on the same Node if it is free.

Master Node

- Will have the following 4 processes running to manage the cluster,

1. API Server
2. Scheduler
3. Controller Manager
4. etcd

Note: A Kubernetes cluster can have multiple master nodes for high availability and fault tolerance. Having multiple master nodes ensures that the cluster can continue to function even if one or more master nodes fail.

API Server

- It is the Cluster Gateway where the Client can query some data from the Cluster or perform some update on the Cluster via API endpoints.
- It also acts as a gatekeeper for authentication, so only authorized Clients can manage the K8s cluster.

Scheduler

- Once the Client request is validated, it is handed over to the Scheduler which will actually schedule the task to be executed. It has intelligent logic implemented for its scheduling.

- Note that the task of actually starting the new Pod is done by the Kubelet on the selected Node, Scheduler is simply responsible for selecting the Node on which the new Pod should be started up.

Controller Manager

- Detects state changes in the cluster like Pod shutdowns, failures, etc.
- When such a state change is detected it tries to recover the cluster state back to normal as soon as possible.
- For this it makes a request to the Scheduler to schedule the dead Pods and then the Scheduler does its task as specified before.

Optional Component: Cloud Controller Manager

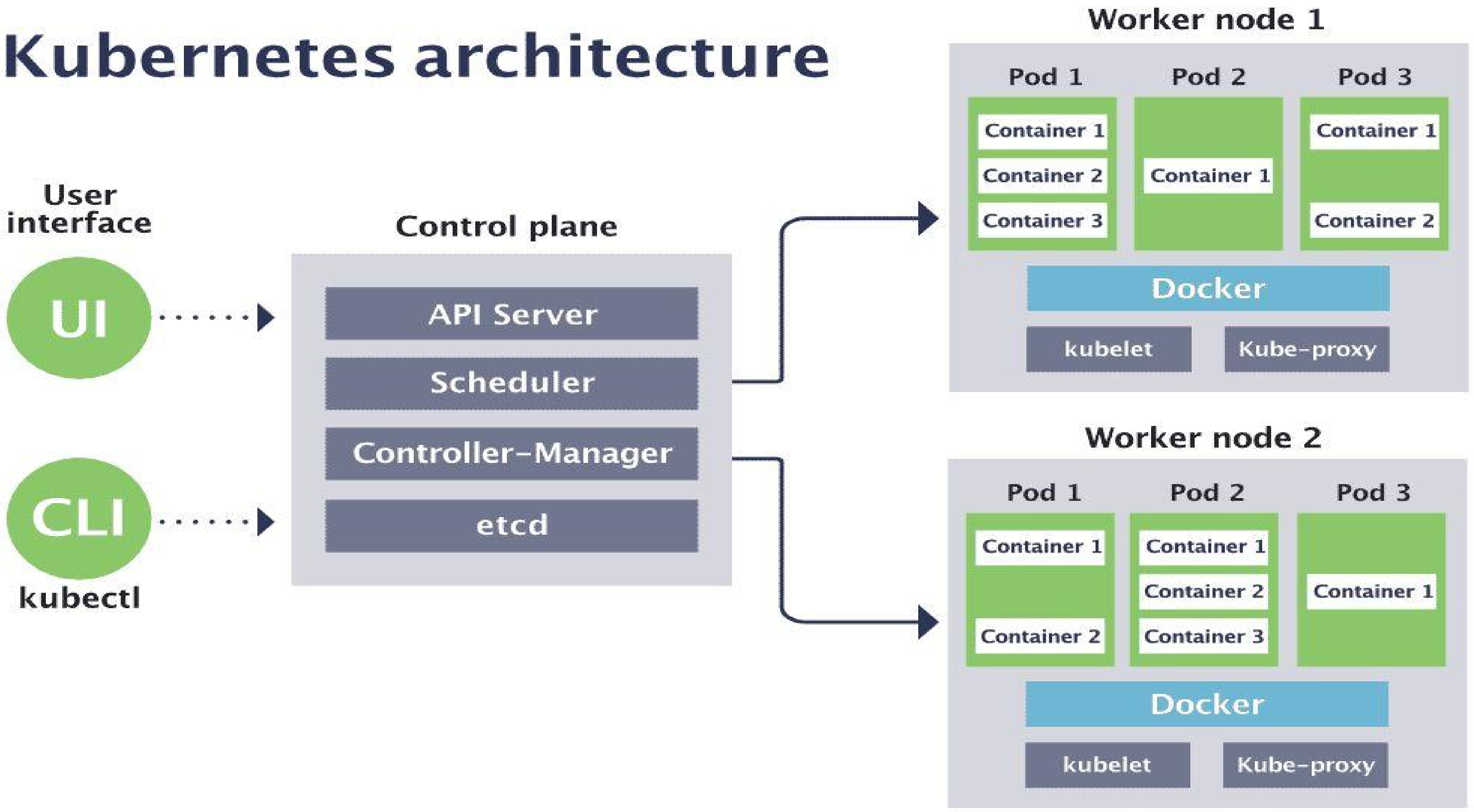
- A Kubernetes control plane component that embeds cloud-specific control logic.
- The cloud controller manager lets us link the cluster into your cloud provider's API, and separates out the components that interact with that cloud platform from components that only interact with your cluster.

etcd (‘et-cee-dee’)

- Key-value store of the Cluster state.
- Can be thought of as the Cluster Brain because all of the K8s Master Processes are dependent on the data that’s stored and maintained in the etcd for normal functioning.
- For example, node health, resource availability status, etc.

- In a multi-Master Node environment, note that the API Servers will be load balanced and the etcd will be a distributed storage.

Kubernetes architecture



Why?

- Scalability
- Availability
- Resource Optimization
- Service Discovery & Load Balancing
- Rolling Updates & Rollbacks

Scalability

- Kubernetes provides automatic scaling of application instances based on demand, allowing your application to handle sudden spikes in traffic or usage without manual intervention.
- Uses Horizontal Pod Autoscaler (HPA).

How?

- Define resource usage metric (e.g., CPU or memory usage) for your workload, along with a target value for that metric.
- If the resource usage exceeds the target value, Kubernetes will automatically increase the number of instances in the workload, up to a maximum limit that you specify.
- Same goes for downscaling as well, down to a minimum limit.

Resource Optimization

- Kubernetes can optimize the use of resources by scheduling application instances on the most suitable nodes based on their resource requirements, ensuring that resources are used efficiently.

Service Discovery & Load Balancing

- Kubernetes provides built-in service discovery and load balancing, making it easy to distribute traffic to the right instances and ensure that your application is highly available.

Rolling Updates & Rollbacks

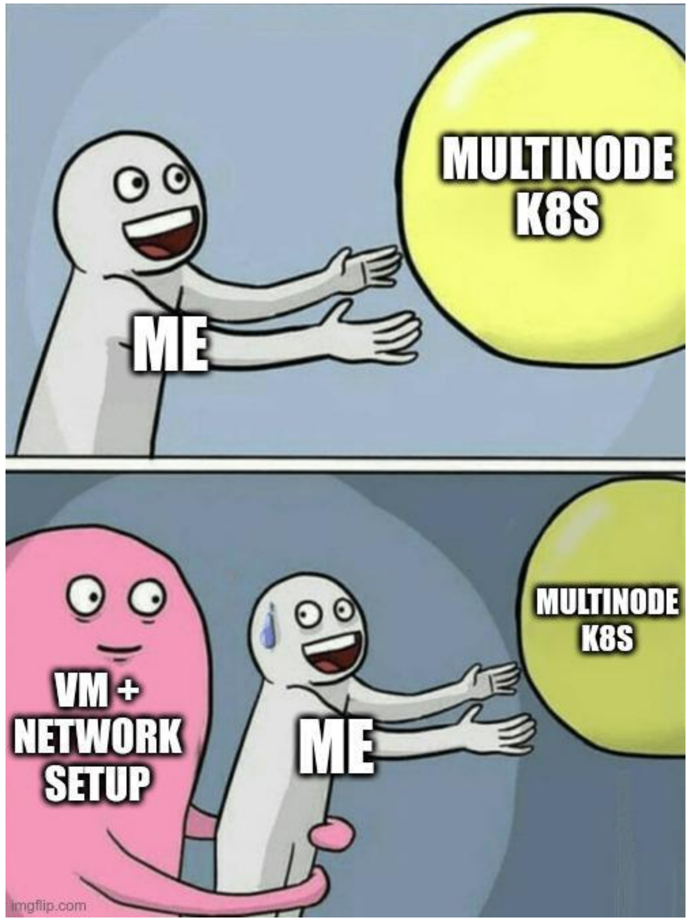
- Kubernetes can automate the process of rolling out new versions of your application while minimizing downtime and risk. If something goes wrong, it can also easily roll back to a previous version.

KUBERNETES
DOCKER COMPOSE
imgflip.com

Can’t Docker Compose do the same?

- In Docker Compose, if a container or a node fails, Docker Compose does not have built-in mechanisms to automatically replace the failed container or move it to a healthy node (but you can still have Restart Policies that can do this).
- In contrast, Kubernetes can automatically detect the failure and replace the failed instance with a new one (this is an in-built feature)

Multi-Node K8s Setup



“There are several tools available that can simplify the process of setting up a multi-node Kubernetes cluster.
For example, Kubernetes distributions like Kubernetes the Hard Way and **kubeadm** provide step-by-step guides for setting up a Kubernetes cluster **from scratch**.
Managed Kubernetes services **from cloud providers** such as Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure also offer easy ways to set up a Kubernetes cluster with a few clicks.

However, setting up a multi-node Kubernetes cluster requires a **good understanding of containerization, networking, and cloud infrastructure.** You'll need to configure and manage various components, such as the **API server, etcd, kube-scheduler, and kube-controller-manager**, and ensure that they are running correctly on each node. You'll also need to set up the **networking and security components**, such as load balancers and firewalls, to ensure that the cluster is accessible and secure.”

minikube

- So how will we test such complicated K8s clusters with just a single local machine?
- Using tools like minikube which **run the Master and Worker processes on a single machine** and the Docker container runtime is pre-installed.
- There’s also **kind** but that has a **stricter requirement on resource usage** so heavier clusters are bound to run out of memory space or not be able to access the processor, etc.
- It uses a Hypervisor under the hood to set up the Node and the virtual machines on your test machine to create a one Node K8s cluster.
- minikube CLI will be used for just the start-up/stop and delete of the cluster and everything else would be done via kubectl only.

kubectl

- Command line tool for interacting with the K8s cluster, i.e. send requests to create Pods, Deployments, Services, manage them, etc.
- API Server in the Control Plane has 3 ways of interacting with the cluster,
  1. UI
  2. API
  3. CLI (kubectl)

kubectl is the most powerful of the 3 clients.It will work with any type of K8s cluster setup, be it minikube, kind, Cloud Cluster, etc.

Exercise 1: K8s Installation

- You can refer to this link if you get stuck anywhere: https://kubernetes.io/docs/tasks/tools/
- **Step 1: Install kubectl**

1.1. Download the latest release with the command:

```
	curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
```

1.2. Install kubectl

- sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

1.3. Test to ensure the version you installed is up-to-date:

- kubectl version --client

Step 2: Install minikube

- To install the latest minikube stable release on x86-64 Linux using binary download:

curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

sudo install minikube-linux-amd64 /usr/local/bin/minikube

Step 3: Start minikube

- minikube start

minikube

Relevant Commands,

minikube start --vm-driver=hyperkit

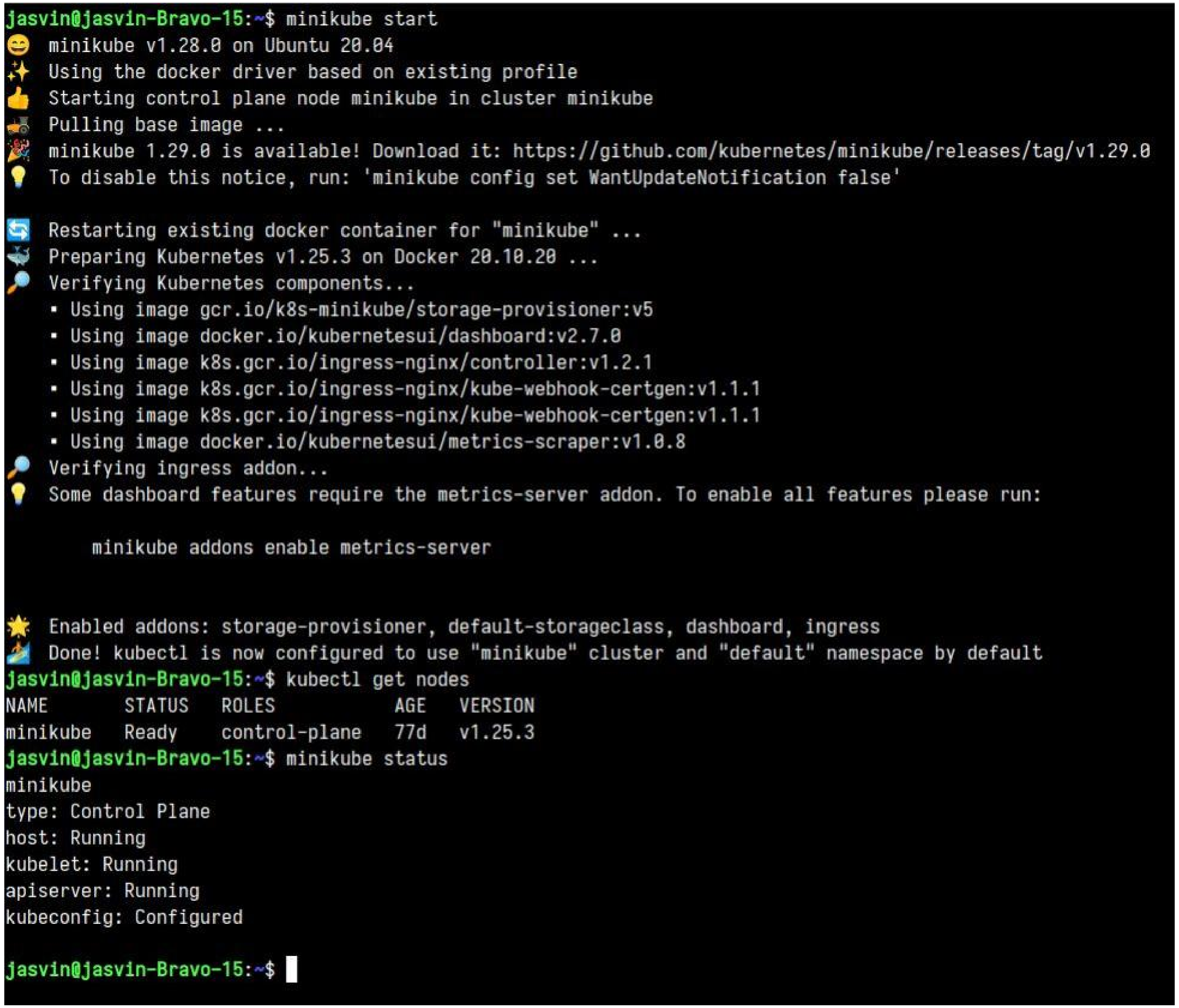
Start minikube cluster with specified VM driver (can also do it without specifying this option)

kubectl get nodes

Should show that the minikube node is ready

minikube status

Should show that all individual components are running



```
jasvin@jasvin-Bravo-15:~$ minikube start
😄  minikube v1.28.0 on Ubuntu 20.04
✨  Using the docker driver based on existing profile
👍  Starting control plane node minikube in cluster minikube
🚜  Pulling base image ...
🎉  minikube 1.29.0 is available! Download it: https://github.com/kubernetes/minikube/releases/tag/v1.29.0
💡  To disable this notice, run: 'minikube config set WantUpdateNotification false'

🔄  Restarting existing docker container for "minikube" ...
🐳  Preparing Kubernetes v1.25.3 on Docker 20.10.20 ...
🔎  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
    ▪ Using image docker.io/kubernetesui/dashboard:v2.7.0
    ▪ Using image k8s.gcr.io/ingress-nginx/controller:v1.2.1
    ▪ Using image k8s.gcr.io/ingress-nginx/kube-webhook-certgen:v1.1.1
    ▪ Using image k8s.gcr.io/ingress-nginx/kube-webhook-certgen:v1.1.1
    ▪ Using image docker.io/kubernetesui/metrics-scraper:v1.0.8
🔎  Verifying ingress addon...
💡  Some dashboard features require the metrics-server addon. To enable all features please run:

        minikube addons enable metrics-server


🌟  Enabled addons: storage-provisioner, default-storageclass, dashboard, ingress
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
jasvin@jasvin-Bravo-15:~$ kubectl get nodes
NAME       STATUS   ROLES           AGE   VERSION
minikube   Ready    control-plane   77d   v1.25.3
jasvin@jasvin-Bravo-15:~$ minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured

jasvin@jasvin-Bravo-15:~$ 
```

K8s Objects

Deployment

Service

ConfigMap

Secret

Ingress

Volume

StatefulSet

Deployment

- Recall: Pods run the app’s Docker images as containers and Services are used for the Pods to communicate with each other as well as with the outside world.
- The same Pods are replicated to ensure high availability.
- For example, 3 replicas of the backend Pod.
- But remember that a Service would only refer to a single entity to refer to all 3 of these replicas implying that there’s another layer of abstraction to a Pod’s replicas.
- Actually there’s two :)

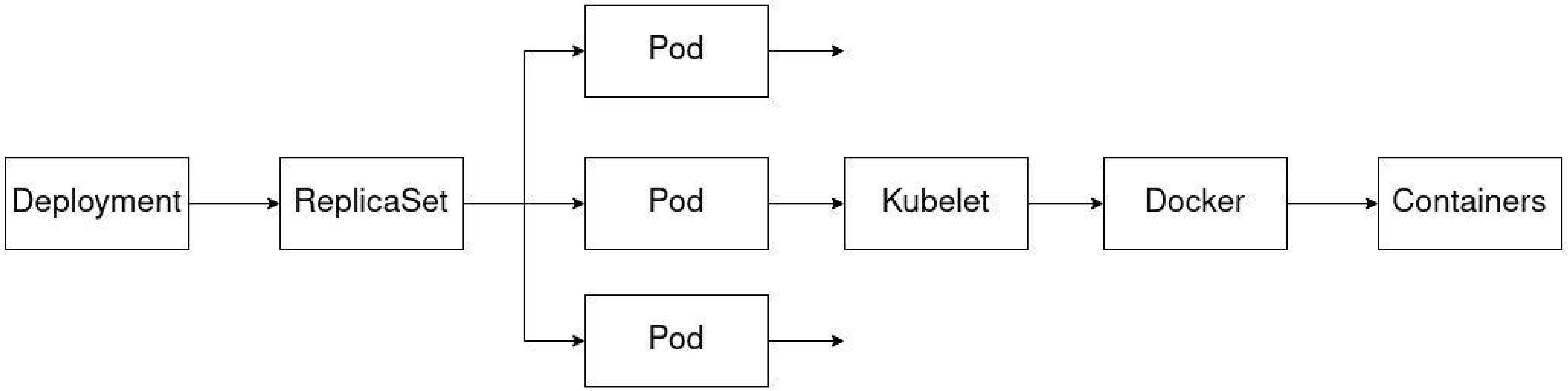
Deployment

Deployment: This is the overall entry into a Pod which defines everything there is to know about the Pod like the containers it'll run, it's network settings, environment variables, namespace, replicas, etc

ReplicaSet: To separately manage the replicas of a Pod specified in the Deployment, you have a ReplicaSet.

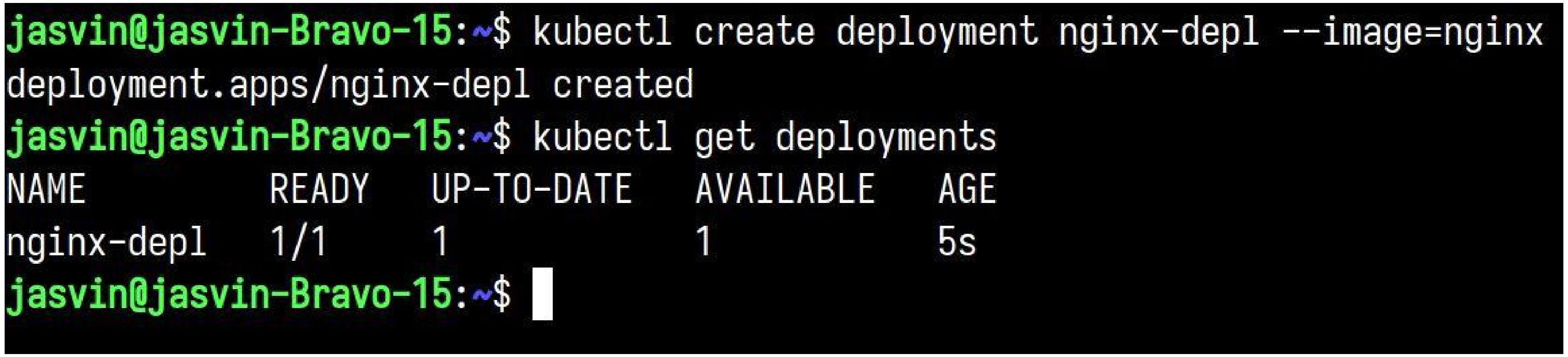
- This component primarily serves a granular level control over the replicas but we won't be dealing with this as our required settings can be set via the Deployment itself.

Deployment Abstraction Hierarchy



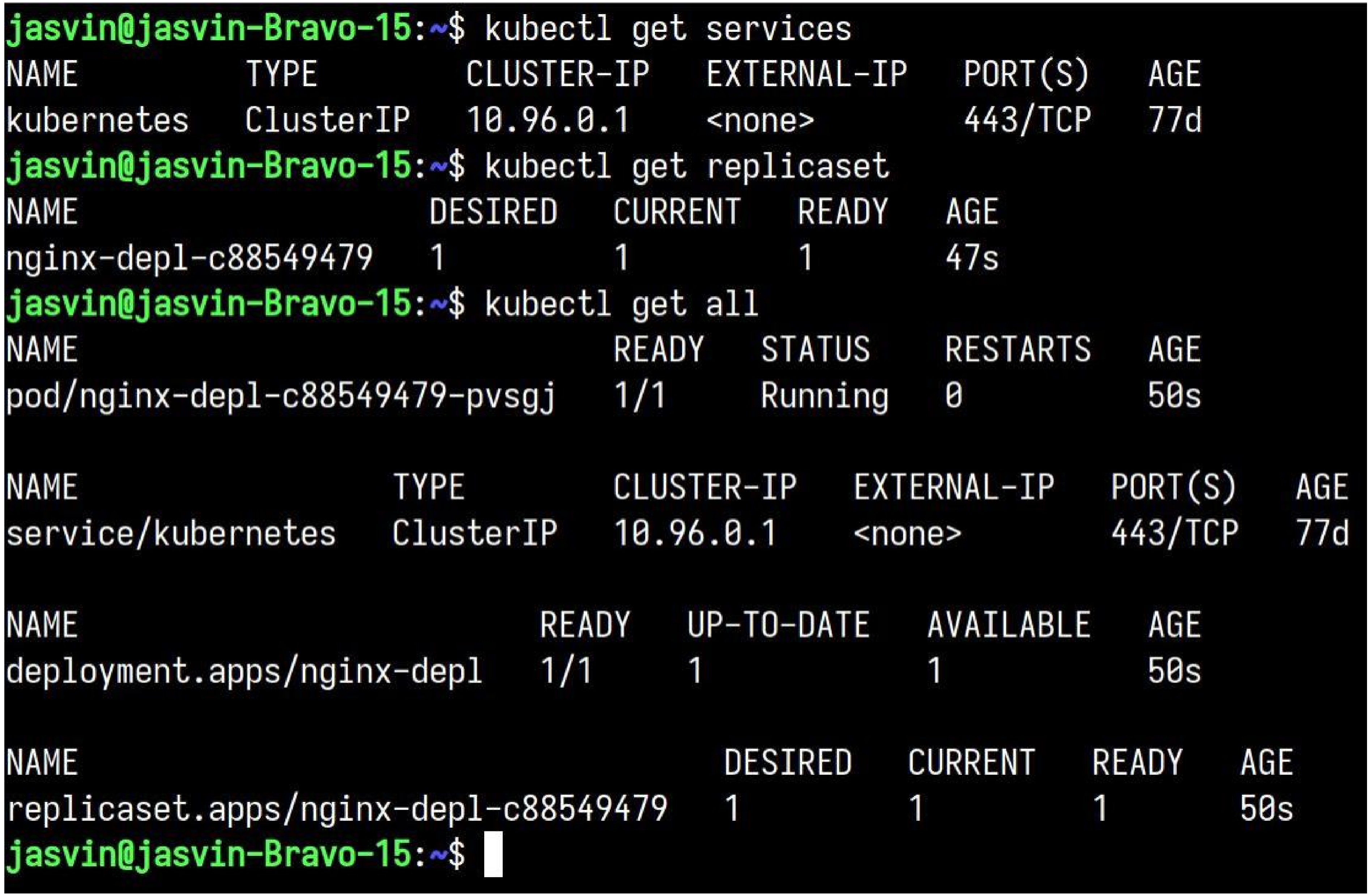
Exercise: kubectl Commands

- Exercise 3: Create Deployment (creates a deployment with the default configuration parameters)



```
jasvin@jasvin-Bravo-15:~$ kubectl create deployment nginx-depl --image=nginx
deployment.apps/nginx-depl created
jasvin@jasvin-Bravo-15:~$ kubectl get deployments
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
nginx-depl   1/1     1            1           5s
jasvin@jasvin-Bravo-15:~$ 
```

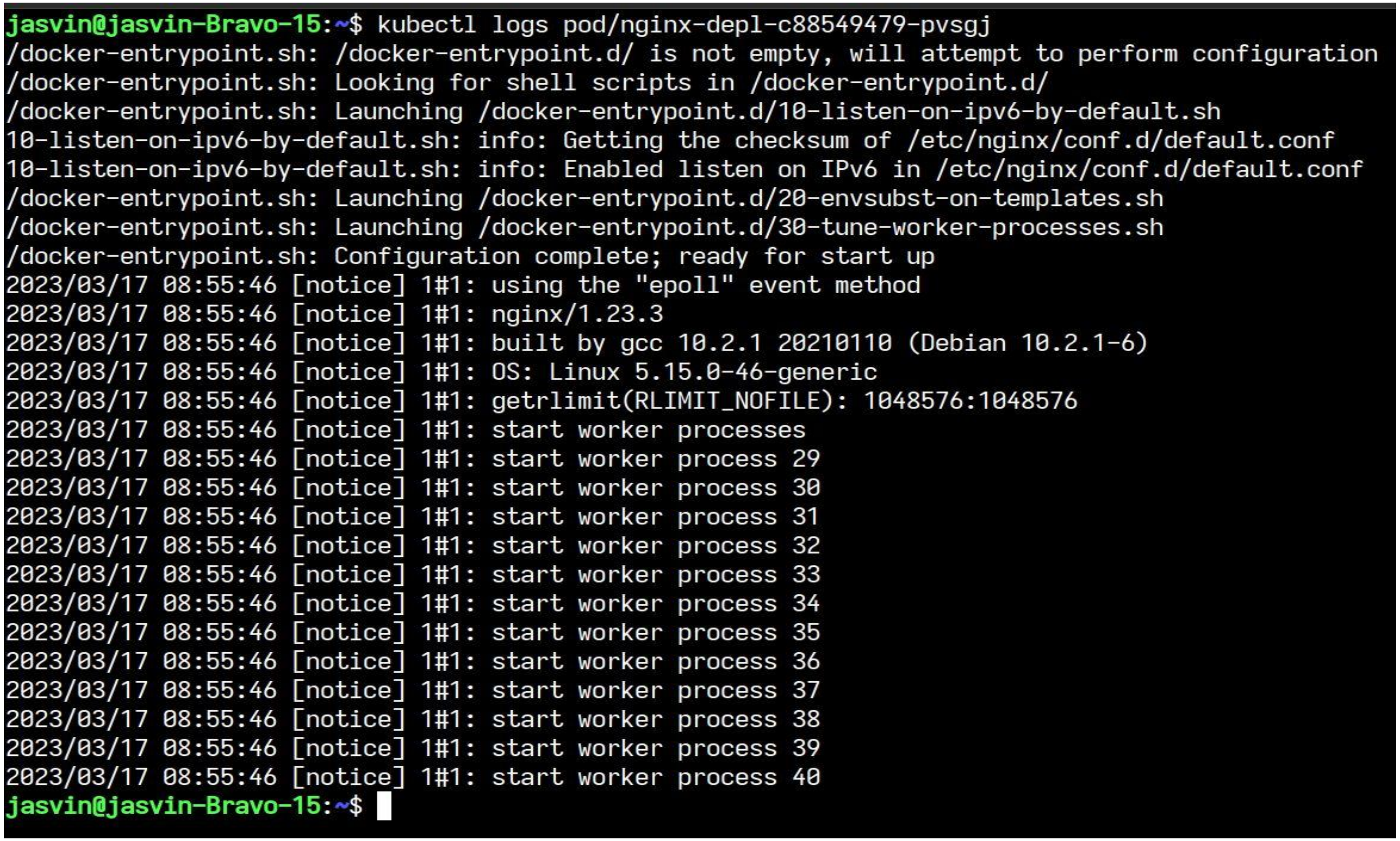
Exercise 4: kubectl get commands



```
jasvin@jasvin-Bravo-15:~$ kubectl get services
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   77d
jasvin@jasvin-Bravo-15:~$ kubectl get replicaset
NAME                   DESIRED   CURRENT   READY   AGE
nginx-depl-c88549479   1         1         1       47s
jasvin@jasvin-Bravo-15:~$ kubectl get all
NAME                             READY   STATUS    RESTARTS   AGE
pod/nginx-depl-c88549479-pvsgj   1/1     Running   0          50s

NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   77d

NAME                         READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx-depl   1/1     1            1           50s

NAME                                   DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-depl-c88549479   1         1         1       50s
jasvin@jasvin-Bravo-15:~$ 
```

Exercise 5: kubectl logs

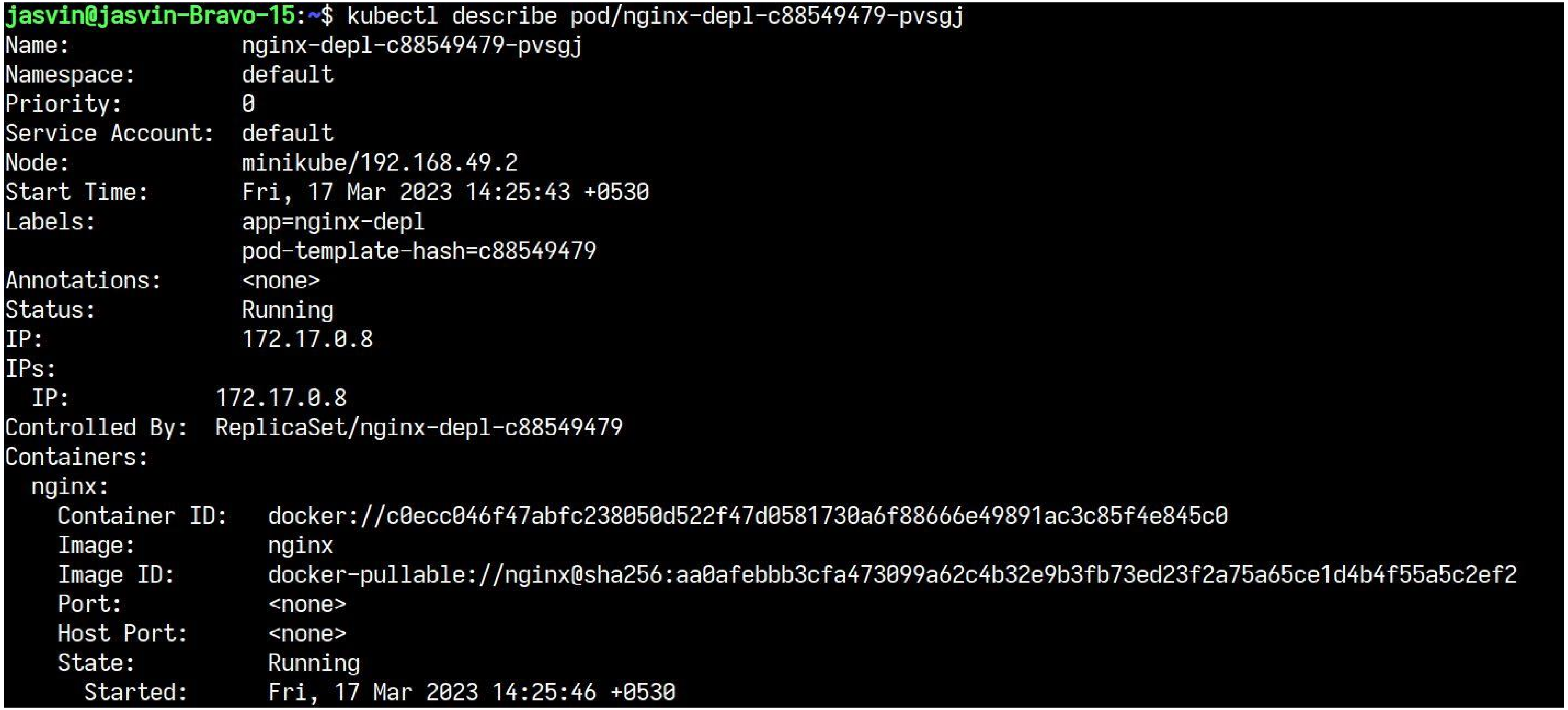
Inspect the logs of the specified K8s object



```
jasvin@jasvin-Bravo-15:~$ kubectl logs pod/nginx-depl-c88549479-pvsgj
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2023/03/17 08:55:46 [notice] 1#1: using the "epoll" event method
2023/03/17 08:55:46 [notice] 1#1: nginx/1.23.3
2023/03/17 08:55:46 [notice] 1#1: built by gcc 10.2.1 20210110 (Debian 10.2.1-6)
2023/03/17 08:55:46 [notice] 1#1: OS: Linux 5.15.0-46-generic
2023/03/17 08:55:46 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2023/03/17 08:55:46 [notice] 1#1: start worker processes
2023/03/17 08:55:46 [notice] 1#1: start worker process 29
2023/03/17 08:55:46 [notice] 1#1: start worker process 30
2023/03/17 08:55:46 [notice] 1#1: start worker process 31
2023/03/17 08:55:46 [notice] 1#1: start worker process 32
2023/03/17 08:55:46 [notice] 1#1: start worker process 33
2023/03/17 08:55:46 [notice] 1#1: start worker process 34
2023/03/17 08:55:46 [notice] 1#1: start worker process 35
2023/03/17 08:55:46 [notice] 1#1: start worker process 36
2023/03/17 08:55:46 [notice] 1#1: start worker process 37
2023/03/17 08:55:46 [notice] 1#1: start worker process 38
2023/03/17 08:55:46 [notice] 1#1: start worker process 39
2023/03/17 08:55:46 [notice] 1#1: start worker process 40
jasvin@jasvin-Bravo-15:~$ 
```

Exercise 6: kubectl describe

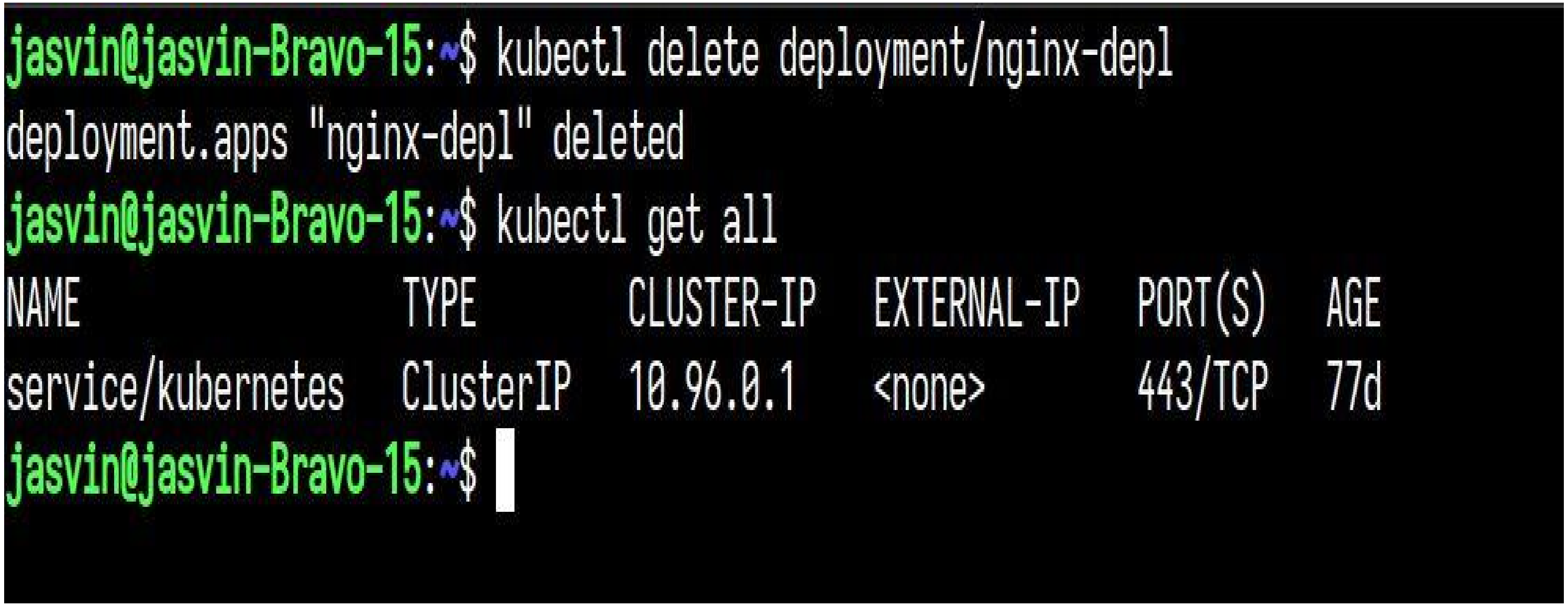
- Describe the specified K8s Object



```
jasvin@jasvin-Bravo-15:~$ kubectl describe pod/nginx-depl-c88549479-pvsgj
Name:             nginx-depl-c88549479-pvsgj
Namespace:        default
Priority:         0
Service Account:  default
Node:             minikube/192.168.49.2
Start Time:       Fri, 17 Mar 2023 14:25:43 +0530
Labels:           app=nginx-depl
                  pod-template-hash=c88549479
Annotations:      <none>
Status:           Running
IP:               172.17.0.8
IPs:
  IP:           172.17.0.8
Controlled By:  ReplicaSet/nginx-depl-c88549479
Containers:
  nginx:
    Container ID:   docker://c0ecc046f47abfc238050d522f47d0581730a6f88666e49891ac3c85f4e845c0
    Image:          nginx
    Image ID:       docker-pullable://nginx@sha256:aa0afebbb3cfa473099a62c4b32e9b3fb73ed23f2a75a65ce1d4b4f55a5c2ef2
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Fri, 17 Mar 2023 14:25:46 +0530
```

Exercise 7: kubectl delete

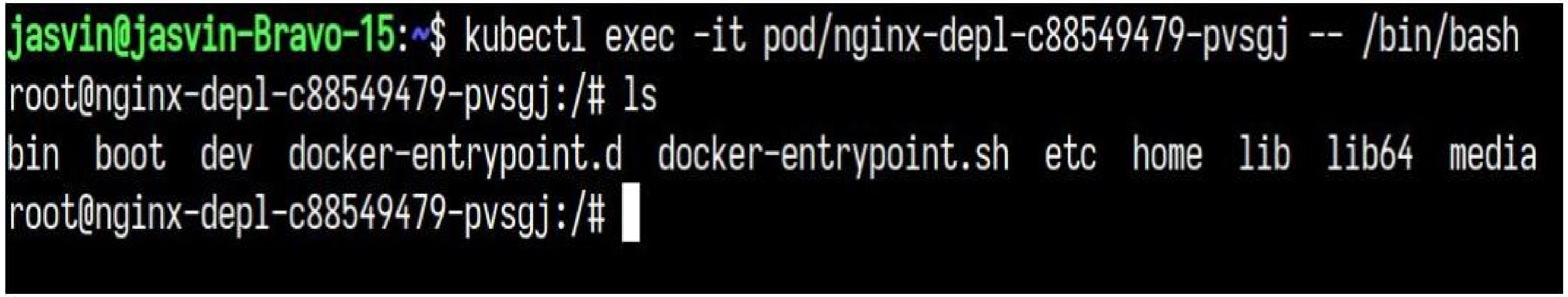
Delete a created K8s object



```
jasvin@jasvin-Bravo-15:~$ kubectl delete deployment/nginx-depl
deployment.apps "nginx-depl" deleted
jasvin@jasvin-Bravo-15:~$ kubectl get all
NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   77d
jasvin@jasvin-Bravo-15:~$
```

Exercise 8: kubectl exec

The equivalent to *docker exec* for K8s



```
jasvin@jasvin-Bravo-15:~$ kubectl exec -it pod/nginx-depl-c88549479-pvsgj -- /bin/bash
root@nginx-depl-c88549479-pvsgj:/# ls
bin  boot  dev  docker-entrypoint.d  docker-entrypoint.sh  etc  home  lib  lib64  media
root@nginx-depl-c88549479-pvsgj:/# 
```

-it = Interactive Terminal
/bin/bash = Which terminal to use

This command gives you a root access terminal window into the specified Pod which is again very helpful with debugging.