KUBERNETES

What is kubernetes

Kubernetes (popularly known as k8s) is a container orchestration engine, which as the name suggests lets you create, delete, and update containers

This is useful when:

You have your docker images in the docker registry and want to deploy it in a cloud native fashion

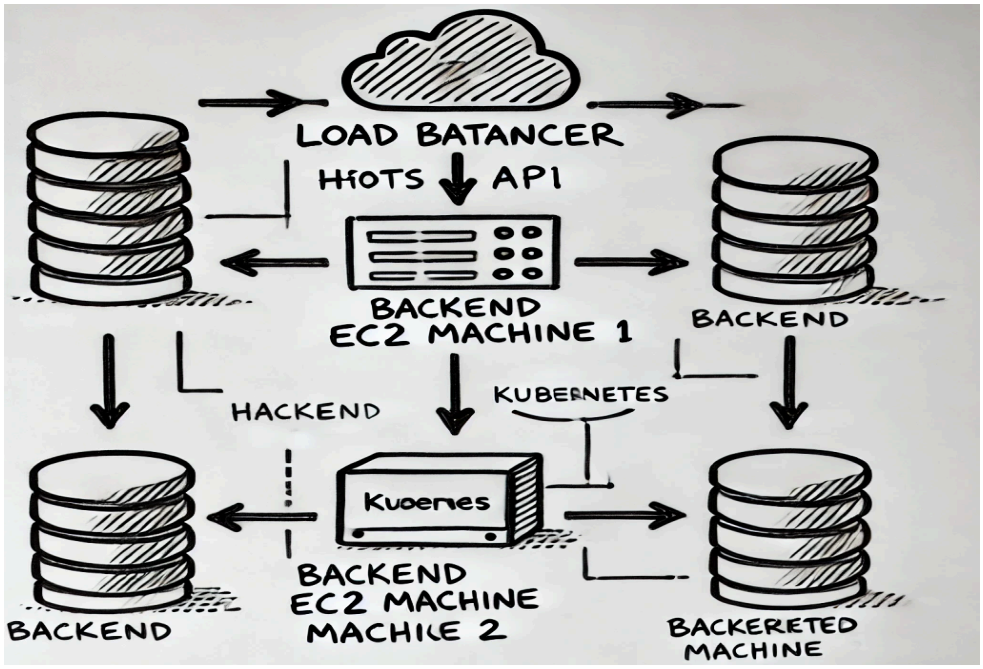
You want to not worry about patching, crashes. You want the system to auto heal

You want to autoscale with some simple constructs

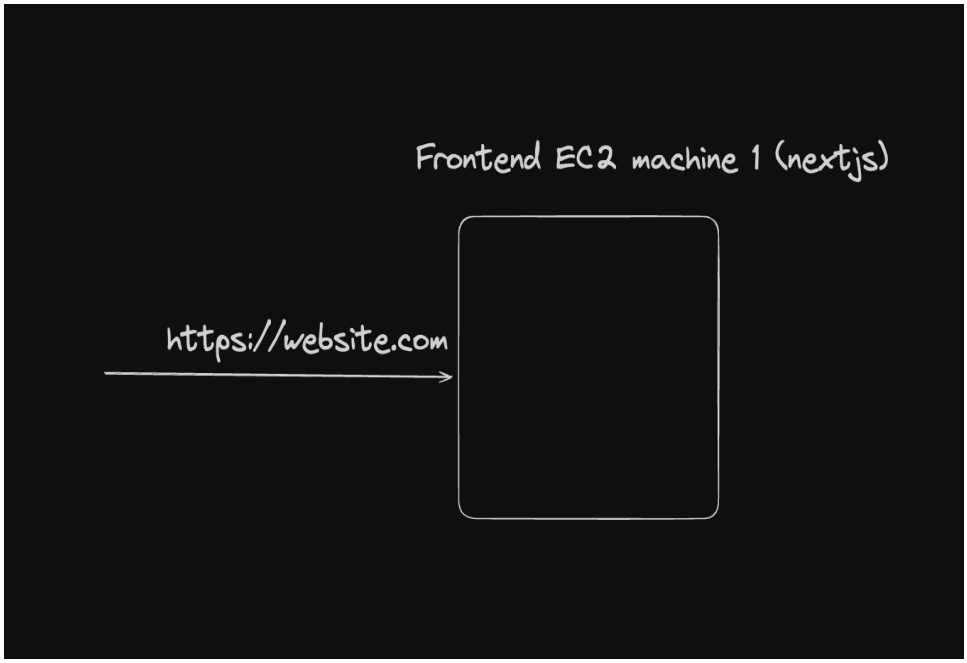
You want to observe your complete system in a simple dashboard

Before kubernetes

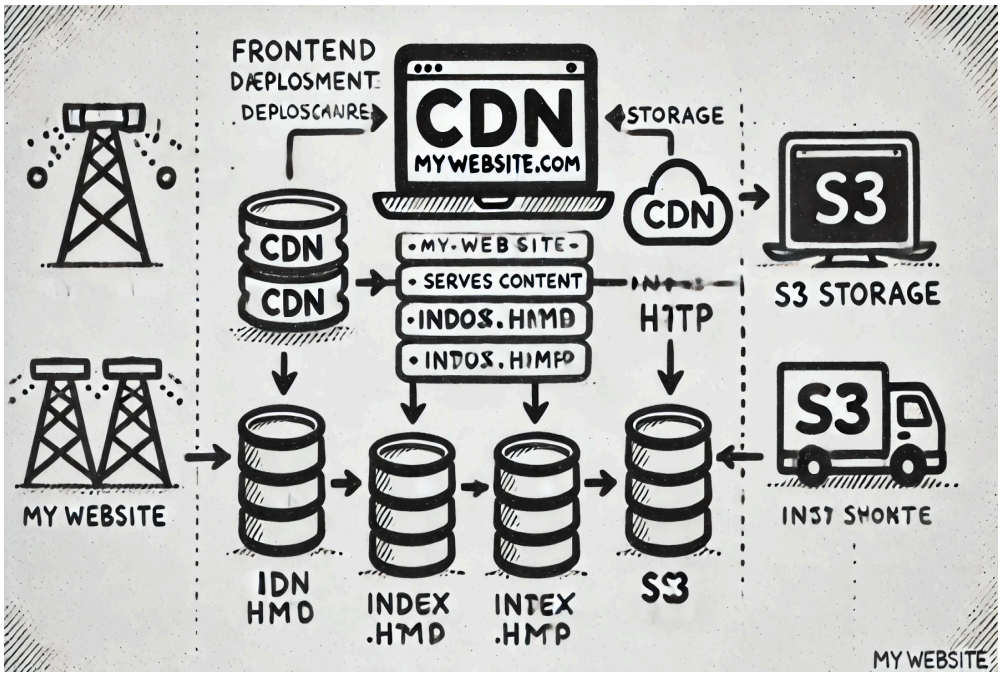
Backend



Frontend (Nextjs)

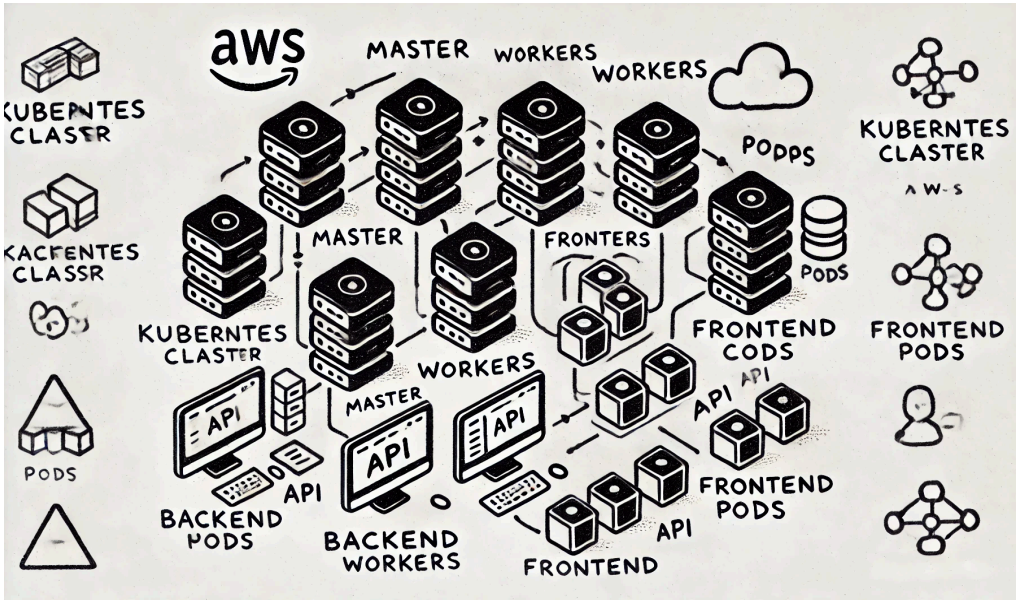


Frontend (React)



After kubernetes

Your frontend, backend are all pods in your kubernetes cluster



Nodes:-

In kubernetes, you can create and connect various machines together, all of which are running kubernetes. Every machine here is known as a node

There are two types of nodes:

a. Master Node (Control pane) - The node that takes care of deploying the containers/healing them/listening to the developer to understand what to deploy

b. Worker Nodes - The nodes that actually run your Backend/frontend

Cluster

A bunch of worker nodes + master nodes make up your kubernetes cluster . You can always add more / remove nodes from a cluster.

Images:-

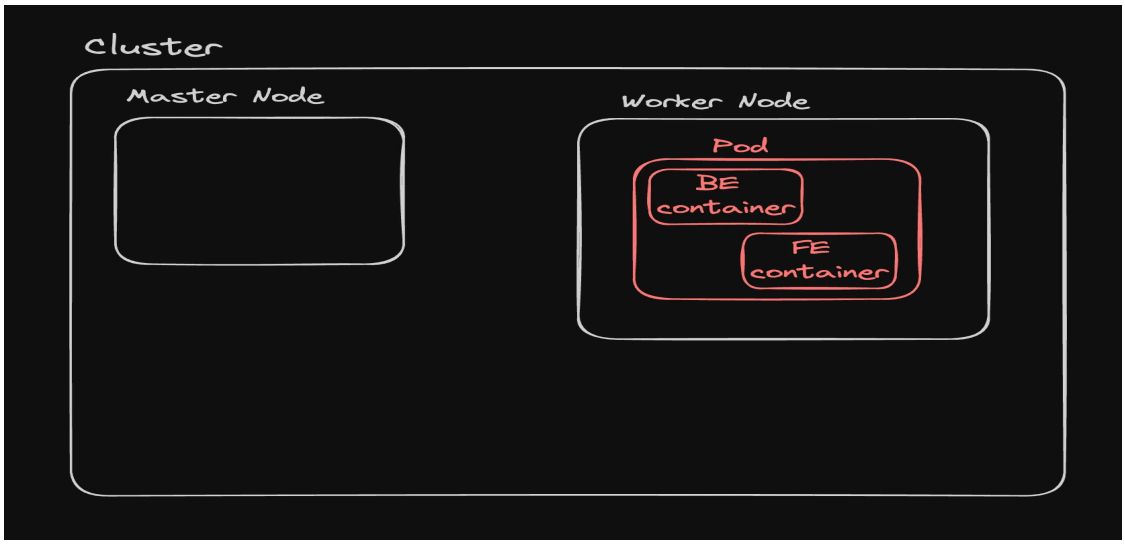
A Docker image is a lightweight, standalone, and executable software package that includes everything needed to run a piece of software, including the code, runtime, libraries, environment variables, and configuration files. Images are built from a set of instructions defined in a file called a Dockerfile.

Containers

A container is an image in execution. For example if you run

Pods:-

A pod is the smallest and simplest unit in the Kubernetes object model that you can create or deploy



Creating a k8s cluster

Locally (Make sure you have docker):-

minukube

kind

On cloud:-

GKE , AWS K8s , vultr

Single node setup:-

Create a 1 node cluster

```
kind create cluster --name local
```

Check the docker containers you have running

```
docker ps
```

You will notice a single container running (control-pane)

Delete the cluster

```
kind delete cluster -n local
```

Multi node setup

Create a `clusters.yml` file

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
- role: worker
- role: worker
```

Create the node setup

```
kind create cluster --config clusters.yml --name local
```

Check docker containers

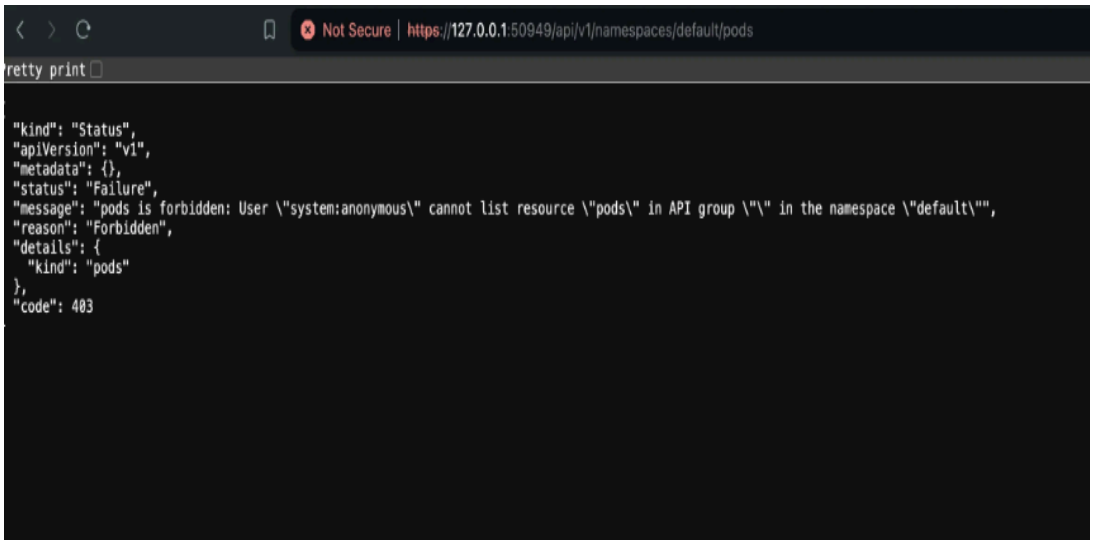
```
docker ps
```

Kubernetes API

The master node (control pane) exposes an API that a developer can use to start pods.

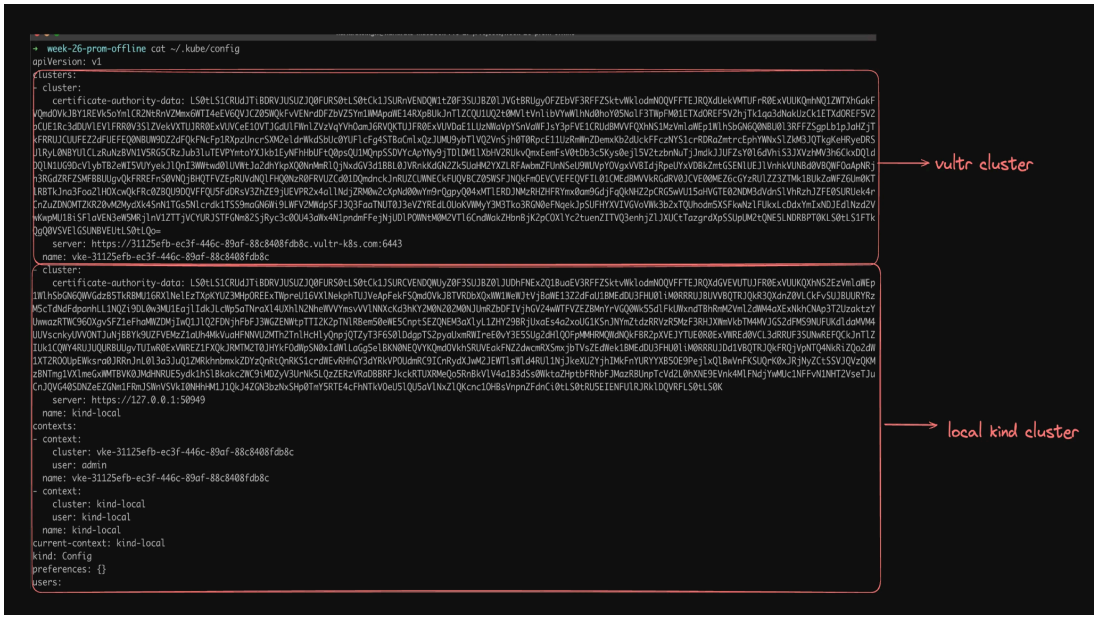
Try the API

Run docker ps to find where the control pane is running



Kubernetes API server does authentication checks and prevents you from getting in.

All of your authorization credentials are stored by kind in ~/.kube/config



kubectl

kubectl is a command-line tool for interacting with Kubernetes clusters. It provides a way to communicate with the Kubernetes API server and manage Kubernetes resources.

Ensure kubectl works fine

```
kubectl get nodes
kubectl get pods
```

If you want to see the exact HTTP request that goes out to the API server, you can add `--v=8` flag

```
kubectl get nodes --v=8
```

Creating a Pod

There were 5 jargons we learnt about:-

Cluster

Nodes

Images

Containers

Pods

Starting using docker

```
docker run -p 3005:80 nginx
```

```
docker (com.docker.cli)
➜  ~ docker run -p 3005:80 nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
24c63b8dcb66: Already exists
cfe05c174c17: Pull complete
f72850762d9b: Pull complete
82945d645e12: Pull complete
a6f455b832da: Pull complete
22fc067a16f6: Pull complete
4bd9fc5c1e9f: Pull complete
Digest: sha256:0f04e4f646a3f14bf31d8bc8d885b6c951fdcf42589d06845f64d18aec6a3c4d
Status: Downloaded newer image for nginx:latest
/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: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/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
2024/06/01 00:33:21 [notice] 1#1: using the "epoll" event method
2024/06/01 00:33:21 [notice] 1#1: nginx/1.27.0
2024/06/01 00:33:21 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2024/06/01 00:33:21 [notice] 1#1: OS: Linux 6.5.11-linuxkit
2024/06/01 00:33:21 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/06/01 00:33:21 [notice] 1#1: start worker processes
2024/06/01 00:33:21 [notice] 1#1: start worker process 29
2024/06/01 00:33:21 [notice] 1#1: start worker process 30
```

Try visiting localhost:3005

Starting a pod using k8s

Start a pod

```
kubectl run nginx --image=nginx --port=80
```

Check the status of the pod

```
kubectl get pods
```

Check the logs

```
kubectl logs nginx
```

```
→  week-26-prom-offline kubectl logs nginx
/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: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/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
2024/06/01 00:32:23 [notice] 1#1: using the "epoll" event method
2024/06/01 00:32:23 [notice] 1#1: nginx/1.27.0
2024/06/01 00:32:23 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2024/06/01 00:32:23 [notice] 1#1: OS: Linux 6.5.11-linuxkit
2024/06/01 00:32:23 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/06/01 00:32:23 [notice] 1#1: start worker processes
2024/06/01 00:32:23 [notice] 1#1: start worker process 32
2024/06/01 00:32:23 [notice] 1#1: start worker process 33
2024/06/01 00:32:23 [notice] 1#1: start worker process 34
2024/06/01 00:32:23 [notice] 1#1: start worker process 35
2024/06/01 00:32:23 [notice] 1#1: start worker process 36
2024/06/01 00:32:23 [notice] 1#1: start worker process 37
2024/06/01 00:32:23 [notice] 1#1: start worker process 38
2024/06/01 00:32:23 [notice] 1#1: start worker process 39
```

Describe the pod to see more details

```
→  week-26-prom-offline kubectl describe pod nginx
Name:             nginx
Namespace:        default
Priority:         0
Service Account:  default
Node:             local-worker/172.20.0.3        → worker 1
Start Time:       Sat, 01 Jun 2024 06:02:10 +0530
Labels:           run=nginx
Annotations:      <none>
Status:           Running
IP:               10.244.1.2
```

What our system looks like right now

Good questions to ask:-

How can I stop a pod?

How can I visit the pod? Which port is it available on?

How many pods can I start?

Stop the pod

Stop the pod by running

```
kubectl delete pod nginx
```

Check the current state of pods

```
kubectl get pods
```

Kubernetes manifest

A manifest defines the desired state for Kubernetes resources, such as Pods, Deployments, Services, etc., in a declarative manner.

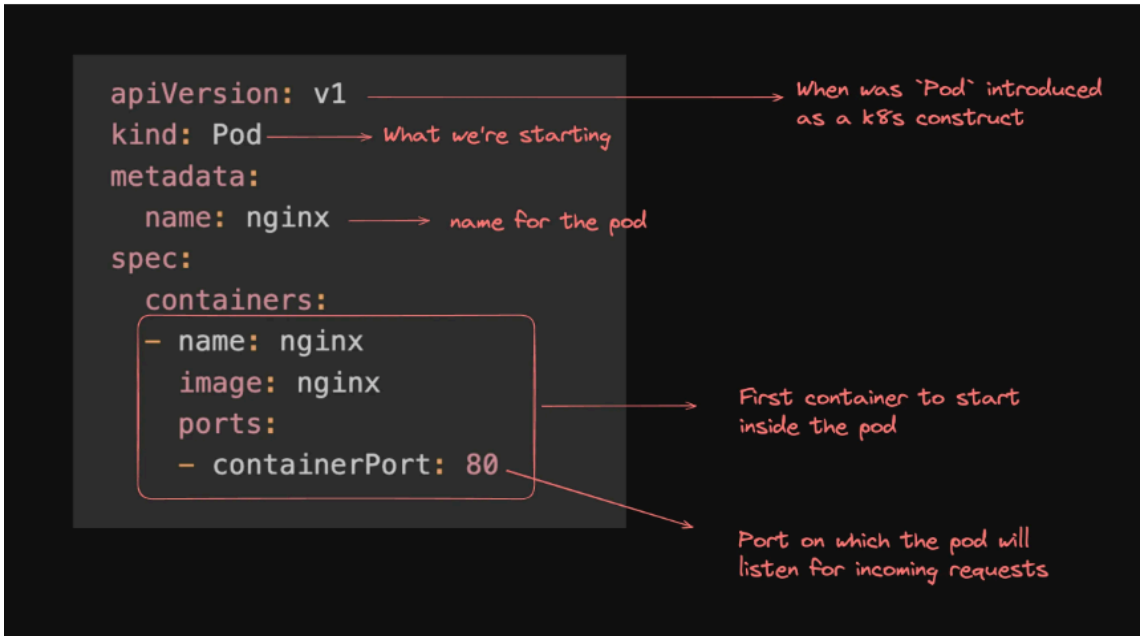
Original command

```
kubectl run nginx --image=nginx --port=80
```

Manifest

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
```

Breaking down the manifest



Applying the manifest

```
kubectl apply -f manifest.yml
```

Delete the pod

```
kubectl delete pod nginx
```

Checkpoint

We have created

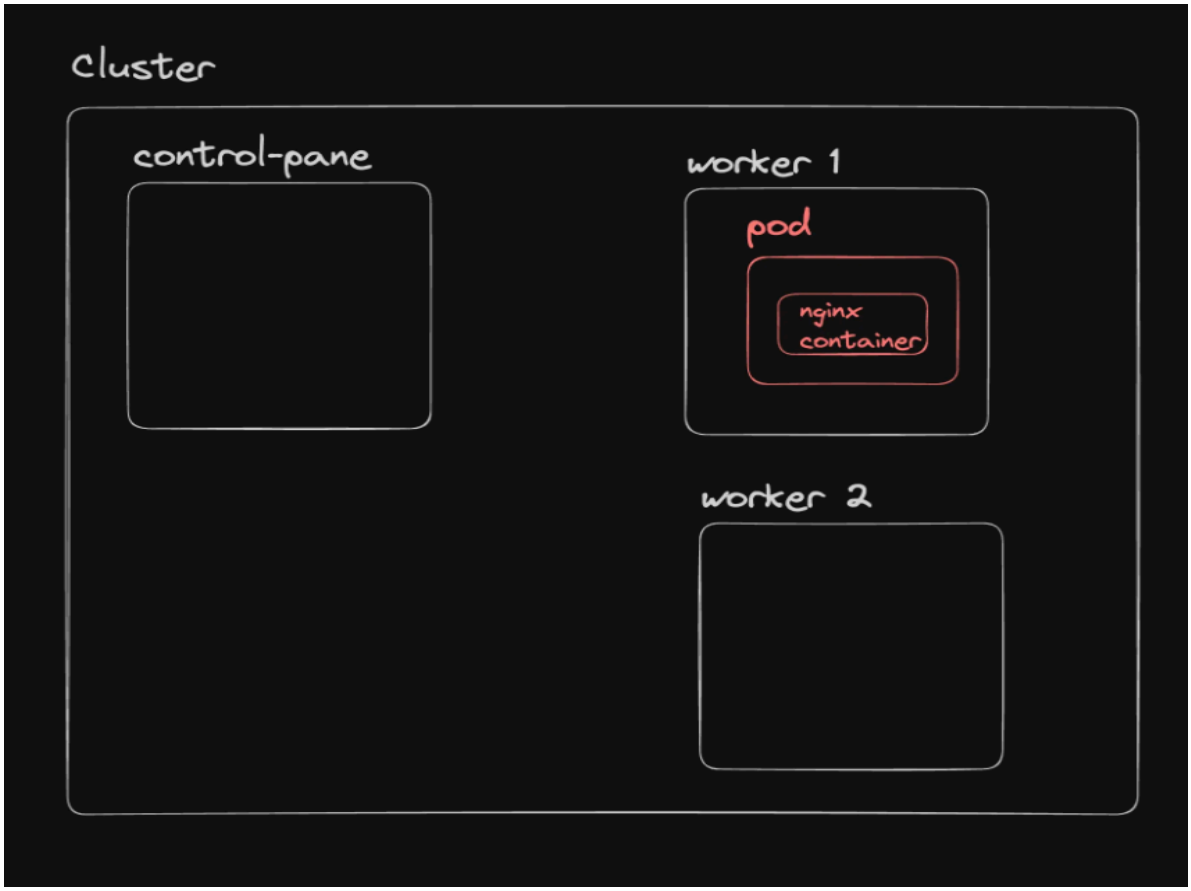
Cluster

Nodes

Images

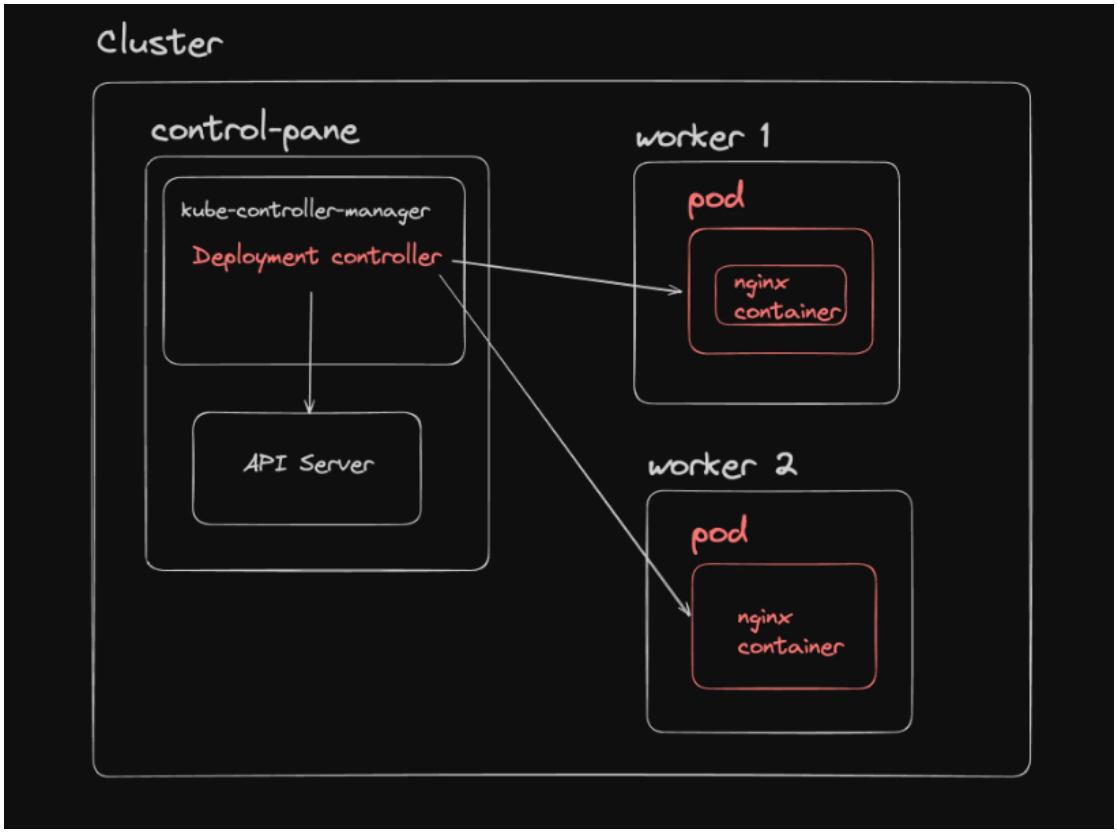
Containers

Pods



Deployment

A Deployment in Kubernetes is a higher-level abstraction that manages a set of Pods and provides declarative updates to them. It offers features like scaling, rolling updates, and rollback capabilities, making it easier to manage the lifecycle of applications.



Pod: A Pod is the smallest and simplest Kubernetes object. It represents a single instance of a running process in your cluster, typically containing one or more containers.

Deployment: A Deployment is a higher-level controller that manages a set of identical Pods. It ensures the desired number of Pods are running and provides declarative updates to the Pods it manages.

Key Differences Between Deployment and Pod:

Abstraction Level:

Pod: A Pod is the smallest and simplest Kubernetes object. It represents a single instance of a running process in your cluster, typically containing one or more containers.

Deployment: A Deployment is a higher-level controller that manages a set of identical Pods. It ensures the desired number of Pods are running and provides declarative updates to the Pods it manages.

Management:

Pod: They are ephemeral, meaning they can be created and destroyed frequently.

Deployment: Deployments manage Pods by ensuring the specified number of replicas are running at any given time. If a Pod fails, the Deployment controller replaces it automatically.

Updates:

Pod: Directly updating a Pod requires manual intervention and can lead to downtime.

Deployment: Supports rolling updates, allowing you to update the Pod template (e.g., new container image) and roll out changes gradually. If something goes wrong, you can roll back to a previous version.

Scaling:

Pod: Scaling Pods manually involves creating or deleting individual Pods.

Deployment: Allows easy scaling by specifying the desired number of replicas. The Deployment controller adjusts the number of Pods automatically.

Self-Healing:

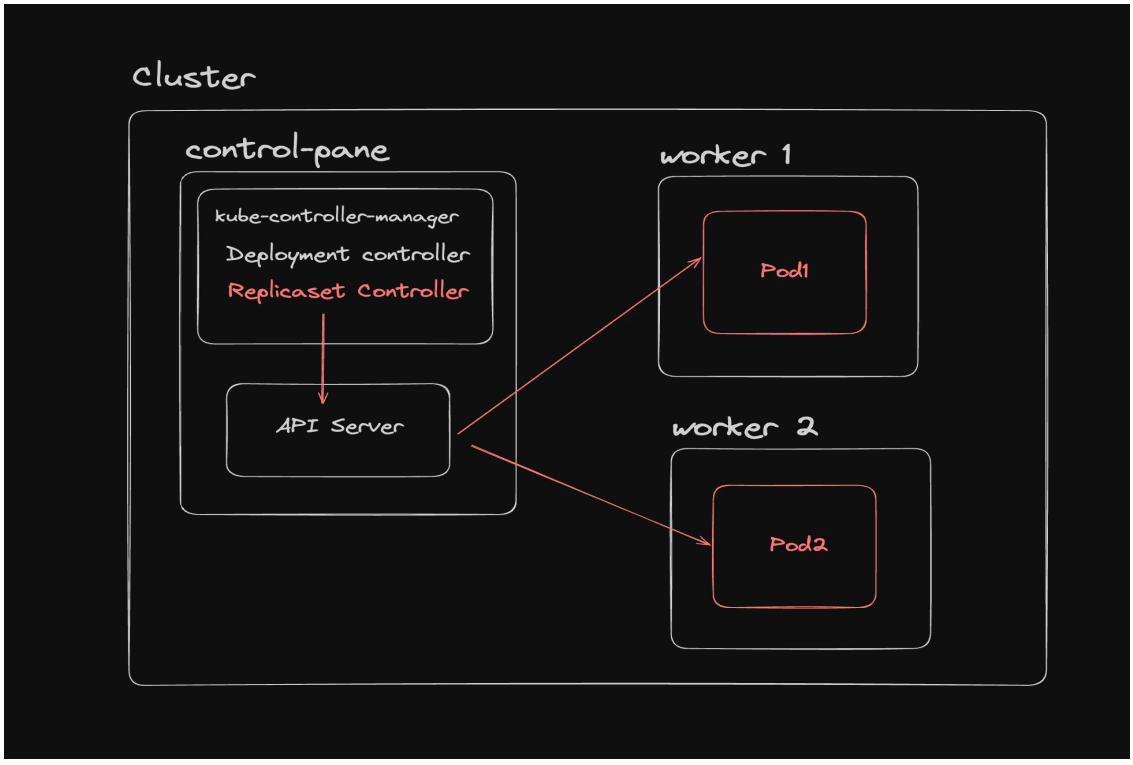
Pod: If a Pod crashes, it needs to be restarted manually unless managed by a higher-level controller like a Deployment.

Deployment: Automatically replaces failed Pods, ensuring the desired state is maintained.

Replicaset

A ReplicaSet in Kubernetes is a controller that ensures a specified number of pod replicas are running at any given time. It is used to maintain a stable set of replica Pods running in the cluster, even if some Pods fail or are deleted.

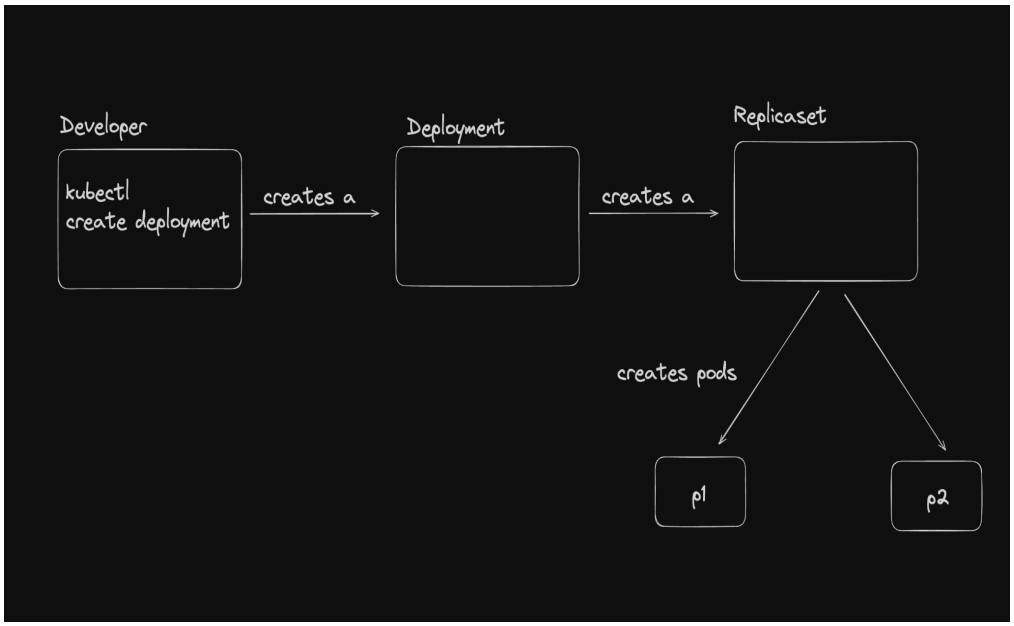
When you create a deployment, you mention the amount of replicas you want for this specific pod to run. The deployment then creates a new ReplicaSet that is responsible for creating X number of pods.



Series of events

User creates a deployment which creates a replicaset which creates pods

If pods go down, replicaset controller ensures to bring them back up



Series of events

When you run the following command, a bunch of things happen

```
kubectl create deployment \
  nginx-deployment \
  --image=nginx \
  --port=80 \
  --replicas=3
```

Step-by-Step Breakdown:-

1. Command Execution:

You execute the command on a machine with kubectl installed and configured to interact with your Kubernetes cluster.

2. API Request:

kubectl sends a request to the Kubernetes API server to create a Deployment resource with the specified parameters.

3. API Server Processing:

The API server receives the request, validates it, and then processes it. If the request is valid, the API server updates the desired state of the cluster stored in etcd. The desired state now includes the new Deployment resource.

4. Storage in etcd:

The Deployment definition is stored in etcd, the distributed key-value store used by Kubernetes to store all its configuration data and cluster state. etcd is the source of truth for the cluster's desired state.

5. Deployment Controller Monitoring:

The Deployment controller, which is part of the kube-controller-manager, continuously watches the API server for changes to Deployments. It detects the new Deployment you created.

6. ReplicaSet Creation:

The Deployment controller creates a ReplicaSet based on the Deployment's specification. The ReplicaSet is responsible for maintaining a stable set of replica Pods running at any given time.

7. Pod Creation:

The ReplicaSet controller (another part of the kube-controller-manager) ensures that the desired number of Pods (in this case, 3) are created and running. It sends requests to the API server to create these Pods.

8. Scheduler Assignment:

The Kubernetes scheduler watches for new Pods that are in the "Pending" state. It assigns these Pods to suitable nodes in the cluster based on available resources and scheduling policies.

9. Node and Kubelet:

The kubelet on the selected nodes receives the Pod specifications from the API server. It then pulls the necessary container images (nginx in this case) and starts the containers

Hierarchical Relationship in Kubernetes:-

1. Deployment (High-Level Manager): Manages application lifecycle, updates, scaling, and rollbacks by creating and managing ReplicaSets.
2. ReplicaSet (Mid-Level Manager): Ensures a specified number of Pods run, maintaining their desired state using label selectors.
3. Pod (Lowest-Level Unit): The smallest Kubernetes object, representing a running process with one or more containers.

Create a replicaset

Let's not worry about deployments, lets just create a replicaset that starts 3 pods

Create rs.yml

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-replicaset
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

Apply the manifest

```
kubectl apply -f rs.yml
```

Get the rs details

```
kubectl get rs

NAME            DESIRED   CURRENT   READY   AGE
nginx-replicaset  3         3         3       23s
```

Check the pods

```
kubectl get pods

NAME                  READY   STATUS    RESTARTS   AGE
nginx-replicaset-7zp2v  1/1     Running   0          35s
nginx-replicaset-q264f  1/1     Running   0          35s
nginx-replicaset-vj42z  1/1     Running   0          35s
```

Try deleting a pod and check if it self heals

```
kubectl delete pod nginx-replicaset-7zp2v
kubectl get pods
```

Try adding a pod with the `app=nginx`

```
kubectl run nginx-pod --image=nginx --labels="app=nginx"
```

Delete the replicaset

```
kubectl delete rs nginx-deployment-576c6b7b6
```

Create a deployment

Create deployment.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

Apply the deployment

```
kubectl apply -f deployment.yml
```

Get the deployment

```
kubectl get deployment

NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   3/3     3            3           18s
```

Get the rs

```
kubectl get rs
NAME                          DESIRED   CURRENT   READY   AGE
nginx-deployment-576c6b7b6    3         3         3       34s
```

Get the pod

```
kubectl get pod
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-576c6b7b6-b6kgk    1/1     Running   0          46s
nginx-deployment-576c6b7b6-m8ttl    1/1     Running   0          46s
nginx-deployment-576c6b7b6-n9cx4    1/1     Running   0          46s
```

Try deleting a pod

```
kubectl delete pod nginx-deployment-576c6b7b6-b6kgk
```

Ensure the pods are still up

```
kubectl get pods
```

Why do you need deployment?

If all that a deployment does is create a replicaset , why cant we just create rs ?

Experiment

Update the image to be nginx2 (an image that doesn't exist)

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx2:latest
        ports:
        - containerPort: 80
```

Apply the new deployment

```
kubectl apply -f deployment.yml
```

Check the new `rs` now

```
kubectl get rs
NAME                          DESIRED   CURRENT   READY   AGE
nginx-deployment-576c6b7b6    3         3         3       14m
nginx-deployment-5fbd4799cb   1         1         0       10m
```

Check the pods

```
kubectl get pods
NAME                                READY   STATUS             RESTARTS   AGE
nginx-deployment-576c6b7b6-9nlnq    1/1     Running            0          15m
nginx-deployment-576c6b7b6-m8ttl    1/1     Running            0          16m
nginx-deployment-576c6b7b6-n9cx4    1/1     Running            0          16m
nginx-deployment-5fbd4799cb-fmt4f   0/1     ImagePullBackOff   0          12m
```

Role of deployment

Deployment ensures that there is a smooth deployment, and if the new image fails for some reason, the old replicaset is maintained.

Even though the rs is what does pod management , deployment is what does rs management

Rollbacks

- Check the history of deployment

```
kubectl rollout history deployment/nginx-deployment
```

- Undo the last deployment

```
kubectl rollout undo deployment/nginx-deployment
```

Create a new deployment

Replace the image to be postgres

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: postgres:latest
        ports:
        - containerPort: 80
```

- Check the new set or `rs`

```
→  kubernetes kubectl get rs
NAME                          DESIRED   CURRENT   READY   AGE
nginx-deployment-576c6b7b6    3         3         3       21m
nginx-deployment-5fbd4799cb   0         0         0       17m
nginx-deployment-7cdb767447   1         1         0       4s
→  kubernetes kubectl get pods
NAME                                READY   STATUS              RESTARTS   AGE
nginx-deployment-576c6b7b6-9nlnq    1/1     Running             0          20m
nginx-deployment-576c6b7b6-m8ttl    1/1     Running             0          21m
nginx-deployment-576c6b7b6-n9cx4    1/1     Running             0          21m
nginx-deployment-7cdb767447-4d5dr   0/1     ContainerCreating   0          7s
```

- Check the pods

```
→  kubernetes kubectl get pods
NAME                                READY   STATUS              RESTARTS   AGE
nginx-deployment-576c6b7b6-9nlnq    1/1     Running             0          20m
nginx-deployment-576c6b7b6-m8ttl    1/1     Running             0          21m
nginx-deployment-576c6b7b6-n9cx4    1/1     Running             0          21m
nginx-deployment-7cdb767447-4d5dr   0/1     ContainerCreating   0          7s
```

- Check pods after some time

```
→ kubernetes kubectl get pods
NAME                                READY   STATUS             RESTARTS      AGE
nginx-deployment-576c6b7b6-9nlnq    1/1     Running            0             23m
nginx-deployment-576c6b7b6-m8ttl    1/1     Running            0             24m
nginx-deployment-576c6b7b6-n9cx4    1/1     Running            0             24m
nginx-deployment-7cdb767447-4d5dr   0/1     CrashLoopBackOff   4 (68s ago)   3m8s
```

- Update the manifest to pass `POSTGRES_PASSWORD`

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: postgres:latest
        ports:
        - containerPort: 80
        env:
        - name: POSTGRES_PASSWORD
          value: "yourpassword"
```

- Check pods now

```
kubectl get pods
```

```
→ kubernetes kubectl get pods
NAME                                READY   STATUS              RESTARTS   AGE
nginx-deployment-576c6b7b6-9nlnq    1/1     Running             0          25m
nginx-deployment-576c6b7b6-m8ttl    1/1     Running             0          26m
nginx-deployment-576c6b7b6-n9cx4    1/1     Running             0          26m
nginx-deployment-58ff5599d8-hq9r2   0/1     ContainerCreating   0          14s
```

- Try after some time

```
→  kubernetes kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-58ff5599d8-hq9r2   1/1     Running   0          44s
nginx-deployment-58ff5599d8-n4zxh   1/1     Running   0          25s
nginx-deployment-58ff5599d8-vbpv9   1/1     Running   0          22s
```

Check the rs

```
→  kubernetes kubectl get rs
NAME                          DESIRED   CURRENT   READY   AGE
nginx-deployment-576c6b7b6    0         0         0       26m
nginx-deployment-58ff5599d8   3         3         3       57s
nginx-deployment-5fbd4799cb   0         0         0       22m
nginx-deployment-7cdb767447   0         0         0       5m38s
```

How to expose the app?

Let's delete all resources and restart a `deployment` for `nginx` with 3 replicas

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

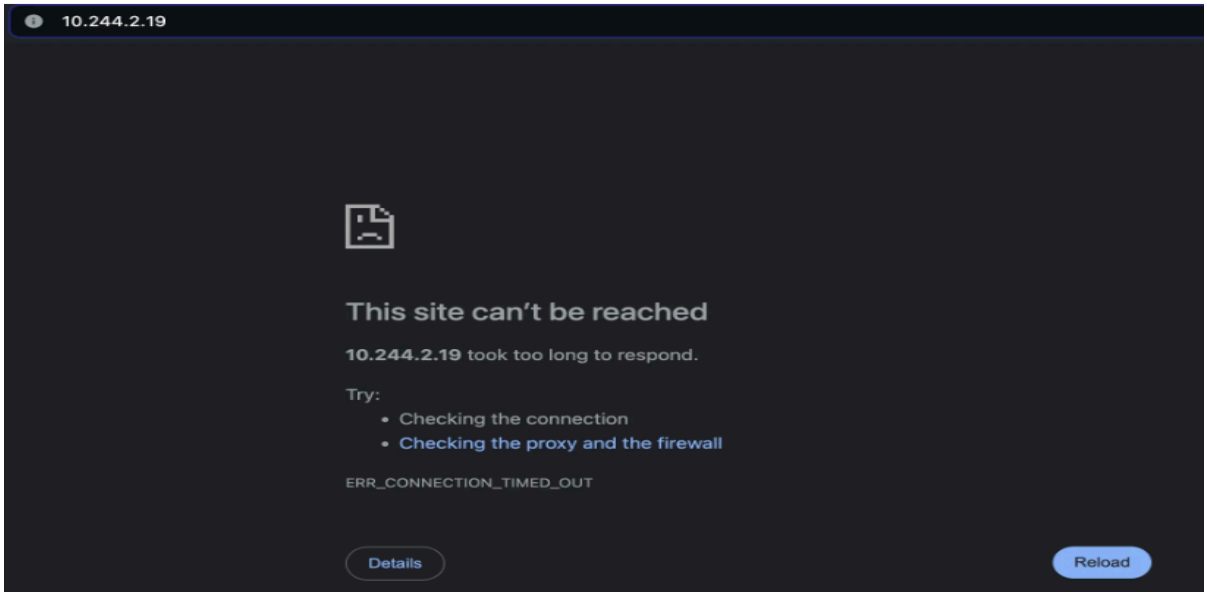
- Apply the configuration

```
kubectl apply -f deployment.yml
```

- Get all pods

```
kubectl get pods -owide
NAME                                READY   STATUS    RESTARTS   AGE    IP           NODE
nginx-deployment-576c6b7b6-7jrn5    1/1     Running   0          2m19s  10.244.2.1
nginx-deployment-576c6b7b6-88fkh    1/1     Running   0          2m22s  10.244.1.
nginx-deployment-576c6b7b6-zf8ff    1/1     Running   0          2m25s  10.244.2.1
```

The IPs that you see are `private IPs` . You wont be able to access the app on it



Services

In Kubernetes, a "Service" is an abstraction that defines a logical set of Pods and a policy by which to access them. Kubernetes Services provide a way to expose applications running on a set of Pods as network services. Here are the key points about Services in

- Create service.yml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
      nodePort: 30007  # This port can be any valid port within the NodePort rar
  type: NodePort
```

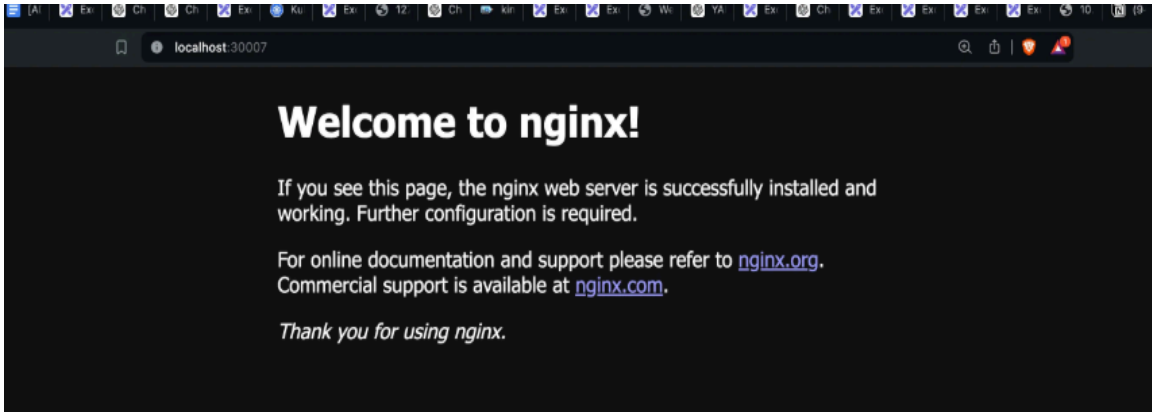
- Restart the cluster with a few extra ports exposed (create kind.yml)

```yaml
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
  extraPortMappings:
  - containerPort: 30007
    hostPort: 30007
- role: worker
- role: worker
```

kind create cluster --config kind.yml

Re apply the deployment and the service

Visit localhost:30007



Types of services

ClusterIP

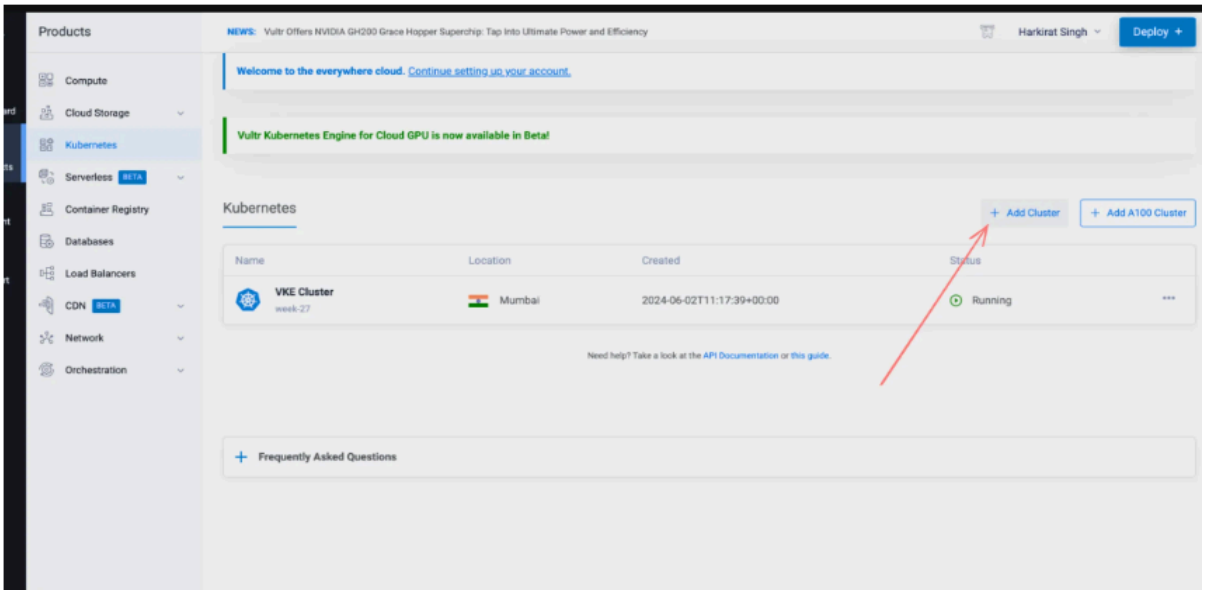
NodePort

Loadbalancer

Loadbalancer service

In Kubernetes, a LoadBalancer service type is a way to expose a service to external clients. When you create a Service of type LoadBalancer, Kubernetes will automatically provision an external load balancer from your cloud provider (e.g., AWS, Google Cloud, Azure) to route traffic to your Kubernetes service

Creating a kubernetes cluster in vultr



Create deployment.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

Apply deployment

```
kubectl apply -f deployment.yml
```

Create `service-lb.yml`

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: LoadBalancer
```

Apply the service

```
kubectl apply -f service-lb.yml
```

Series of events

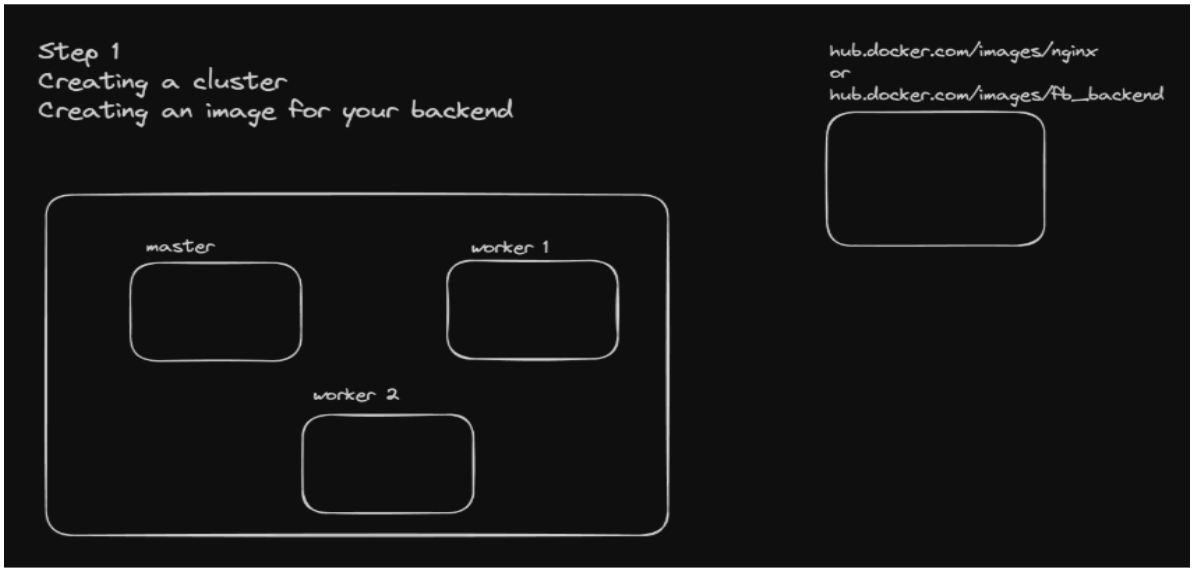
Step 1 - Create your cluster

Create `kind.yml`

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
  extraPortMappings:
  - containerPort: 30007
    hostPort: 30007
- role: worker
  extraPortMappings:
  - containerPort: 30007
    hostPort: 30008
- role: worker
```

- Create cluster

```
kind create cluster --config kind.yml --name local
```



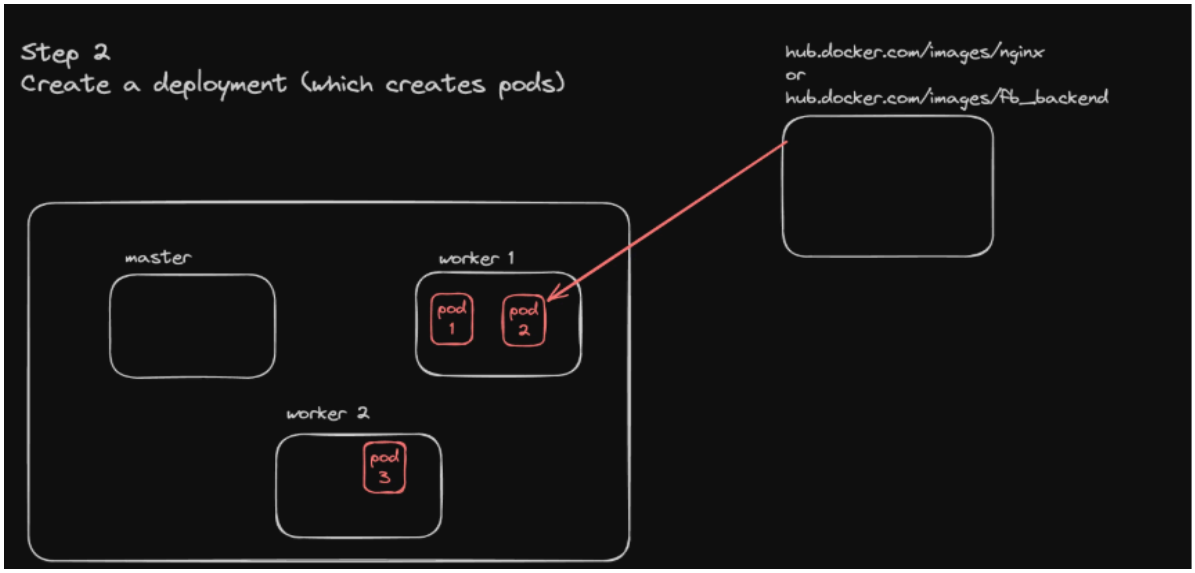
Step 2 - Deploy your pod

Create `deployment.yml`

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
```

Apply the deployment

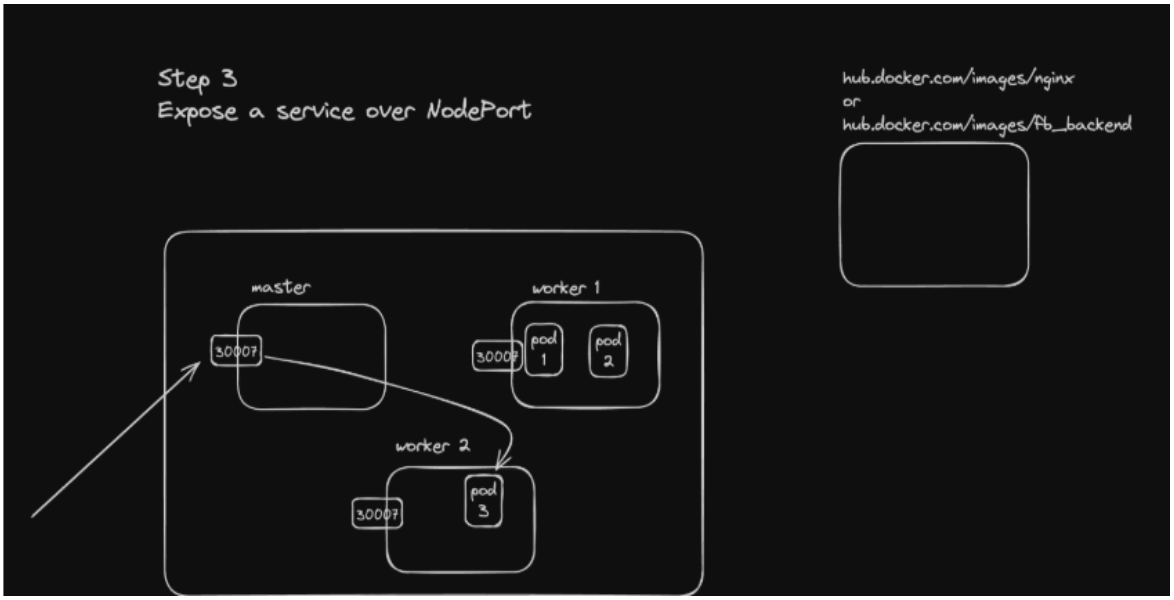
```
kubectl apply -f deployment.yml
```



Step 3 - Expose your app over a NodePort

Create service.yml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
      nodePort: 30007  # This port can be any valid port within the NodePort rang
  type: NodePort
```

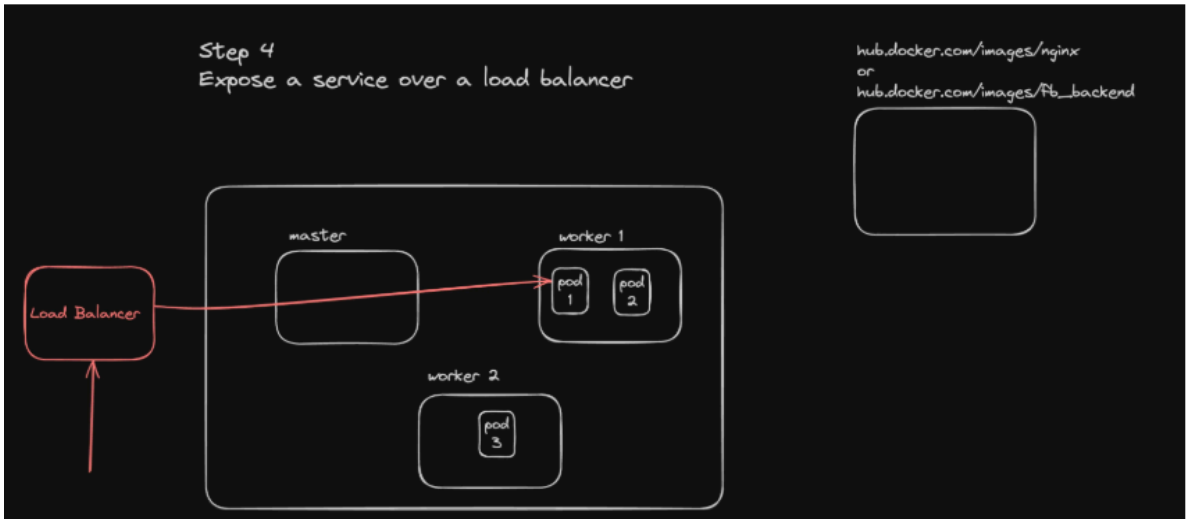


Step 4 - Expose it over a LoadBalancer

Create a load balancer service (service-lb.yml)

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: LoadBalancer
```

kubectl apply service-lb.yml



Check the cloud dashboard