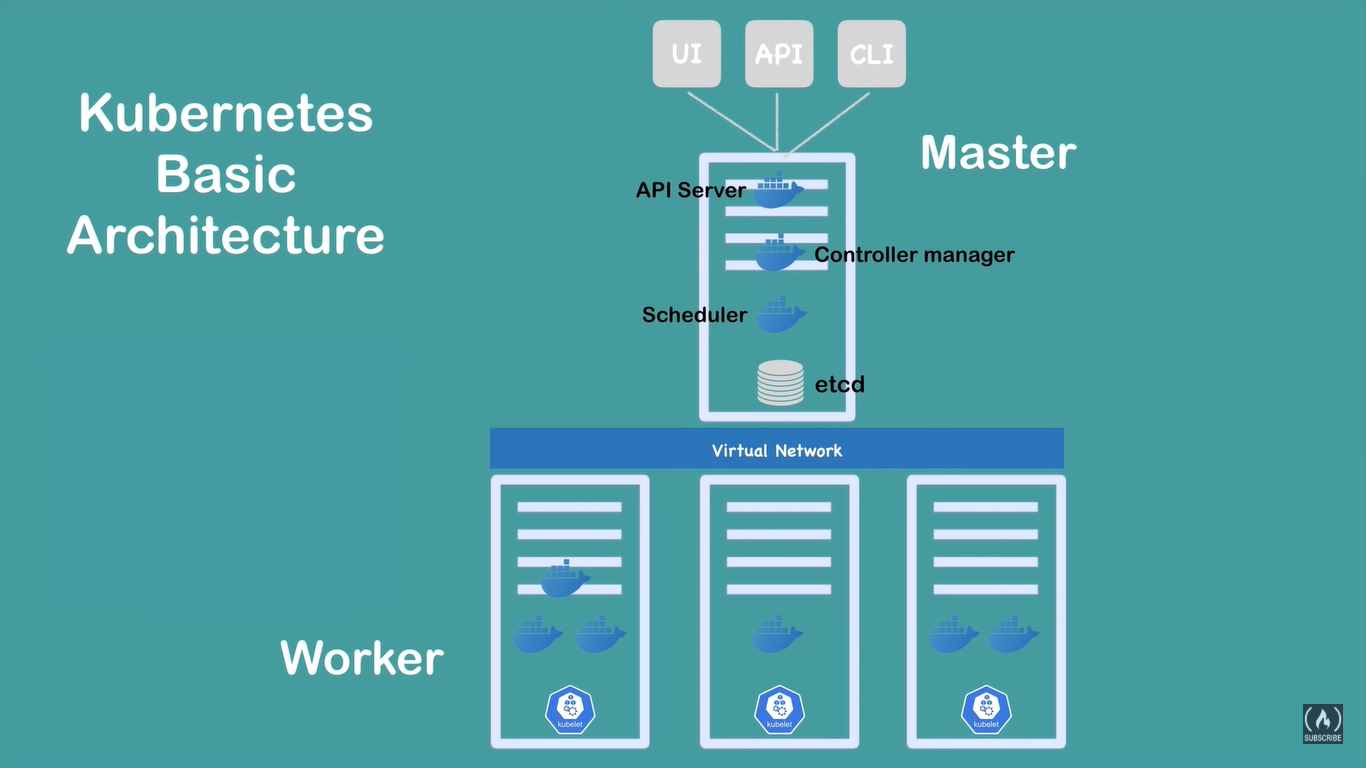
# Kubernetes Basic Architecture



Kubernetes is a container orchestration platform that automates the deployment, scaling, and management of containerized applications. A Kubernetes cluster is a group of nodes that work together to run containerized applications. It consists of two main components:

**Control Plane (Master Node)**

The control plane manages the cluster and facilitates communication between worker nodes. It includes the following key components:

* API Server: Acts as the entry point to the Kubernetes cluster, processing REST requests and updating the cluster state.
* Controller Manager: Ensures the desired state of the cluster by managing controllers that handle tasks like replication and node monitoring.
* Scheduler: Assigns workloads (pods) to worker nodes based on resource availability and constraints.
* etcd: A distributed key-value store that holds the cluster's metadata, such as the state of all components. It's critical for backup and restoration.

**Worker Nodes**

Worker nodes are responsible for running application workloads. Each worker node contains:

* Kubelet: A component that communicates with the control plane to ensure containers are running as expected.
* Kube-proxy: Manages network rules to allow communication between pods and services within and outside the cluster.
* Container Runtime: The software responsible for running containers (e.g., Docker, containerd).

Worker nodes can run multiple pods (the smallest deployable unit in Kubernetes), each of which encapsulates/ abstract one or more containers.

The cluster is designed to appear as a single, unified virtual network, allowing seamless communication between services across the worker nodes.

## Worker Node Architecture

### ****Pods****

Pods are the smallest deployable units in Kubernetes, representing one or more tightly coupled containers. They encapsulate application containers, storage resources, a unique network IP, and configuration options. Pods run on worker nodes and are managed by the **kubelet**, which ensures they are in the desired state. Containers within a pod share the same network namespace and can communicate with each other using “localhost”.

### ****Service****

A Service provides a stable, abstracted network endpoint to expose a set of Pods, ensuring that other components can reliably access them, even if Pods are replaced or restarted. Enables communication between different parts of the application (e.g., frontend accessing backend Pods). Services ensure Pods running on different worker nodes can communicate. The **kube-proxy** on each node routes traffic to the correct Pod(s).

### ****Ingress****

An Ingress is an API object that manages external access to services, typically HTTP or HTTPS traffic. Provides rules for routing requests from outside the cluster to specific services. It also supports load balancing, SSL termination, and URL path-based routing. Ingress controllers run on worker nodes to process and route external traffic based on the Ingress rules defined by the user.

### ****ConfigMap****

A ConfigMap is a key-value store used to manage application configuration data separately from the application code. Allows Pods to dynamically consume configuration data via environment variables, command-line arguments, or mounted files. ConfigMaps are mounted into Pods running on worker nodes, ensuring applications can access the required configuration without needing a redeployment.

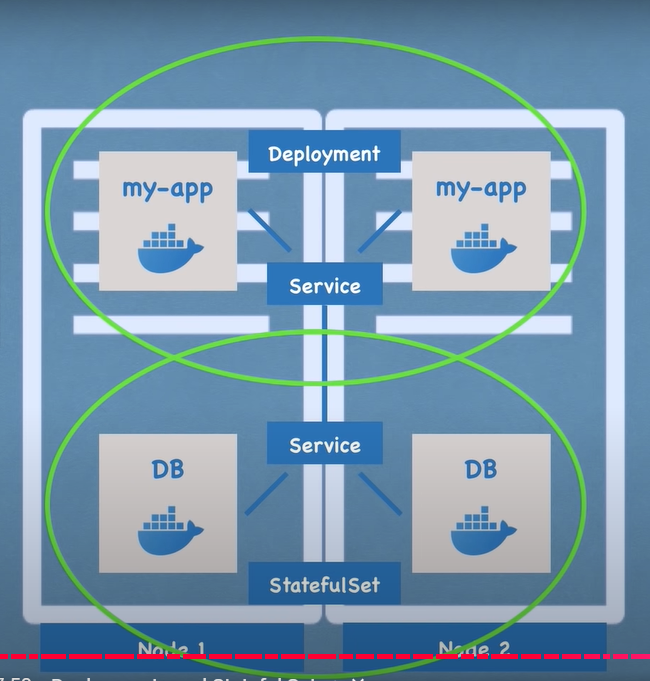
### ****Secrets****

Secrets are similar to ConfigMaps but are specifically designed to store sensitive information, such as API keys, passwords, and certificates, in an encoded format. Provides a secure mechanism for injecting sensitive data into Pods. Secrets are mounted into Pods running on worker nodes, ensuring secure access to sensitive information while keeping it abstracted from the application code.

### Volumes

In Kubernetes, **volumes** provide a way to persist and share data across containers within a Pod. Unlike the ephemeral storage of a container, which is lost when the container restarts, volumes enable data to persist beyond the lifecycle of individual containers. Volumes are mounted into Pods and can be used by one or more containers within the Pod. Kubernetes supports various volume types, including **emptyDir** (temporary storage), **hostPath** (host machine directory), **persistentVolumeClaim (PVC)** (dynamically provisioned storage), and cloud-specific options like **AWS EBS** or **Azure Disks**. Volumes ensure data availability and consistency, especially for stateful applications running in the cluster.

## Deployments and StatefulSets



### Deployments

Deployements are another abstraction above pods. They are the blueprint that is used to handle replication of pods such that instead of managing the lifecycle of a pod manually, the same can be specified within this deployment and the replication is then done manually.

They are designed for stateless applications.

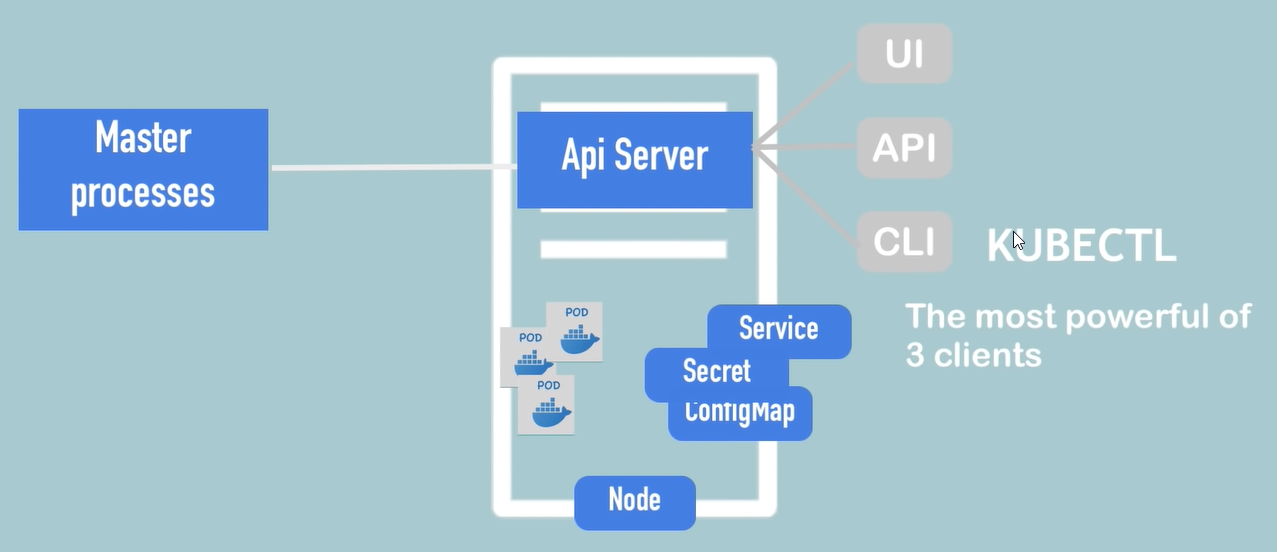
### StatefulSets

These are similar to deployments except they are meant for stateful applications such as database servers. Data in a database need to be persisted in spite of a crash in a pod running a database. The replication of the pods is therefore necessary but this pods need to be synchronized for the sake of consistency of the database. This is done in the statefulsets. However, they are difficult to work with and therefore most production-level applications use an external database.

# Minikube and Kubectl

Minikube is a 1-node kubernetes (k8s) cluster that is used for local testing purposes. This cluster is created in a virtual machine like virtual box and this VM has a docker runtime already installed. This 1 node contains both the master and worker processes.

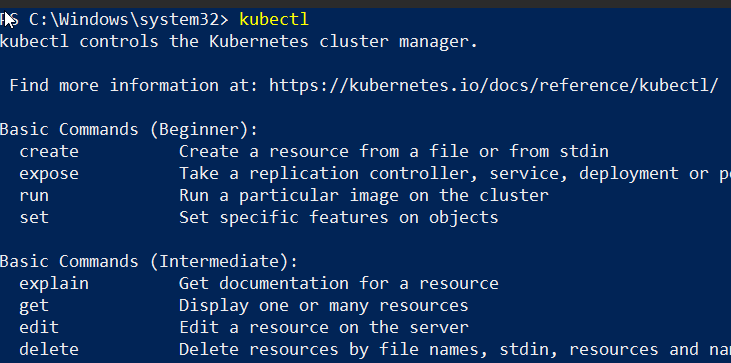
Kubectl is the command line tool for interacting with the kubernetes cluster. It communicates with the API Server of the master inside the node, which in turn communicates with the worker to create and destroy pods as well as create services.

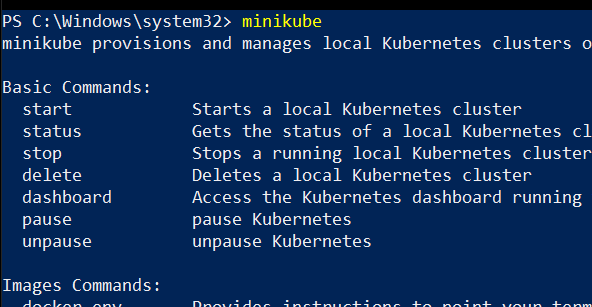


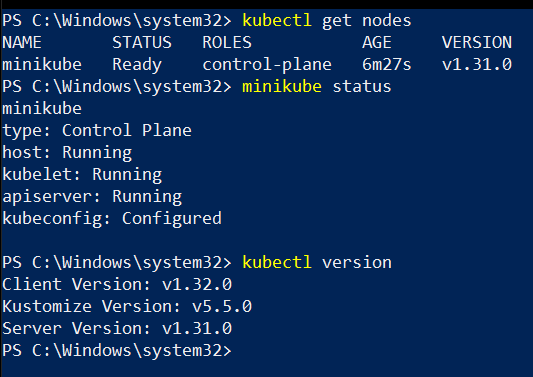
## Installing Minikube and Kubectl

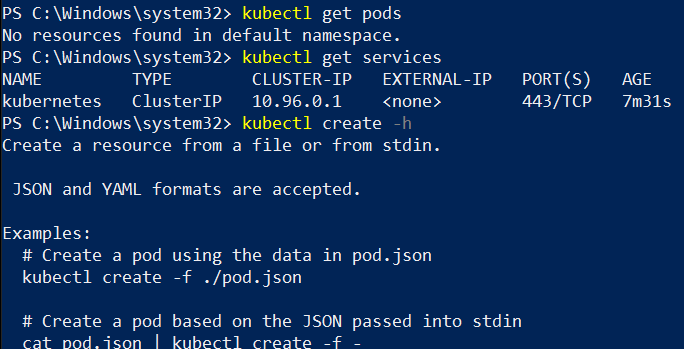
* Download the minikube installer and run it.
* Download kubectl.exe and the other associated files. Add the directory holding this to the system environments variables.

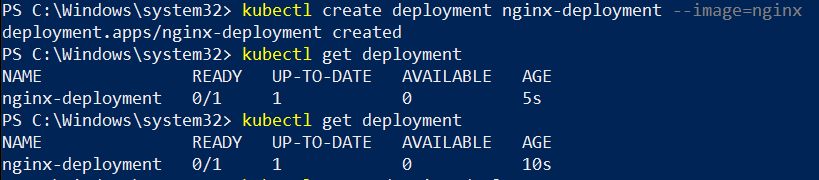
## Minikube and Kubectl commands

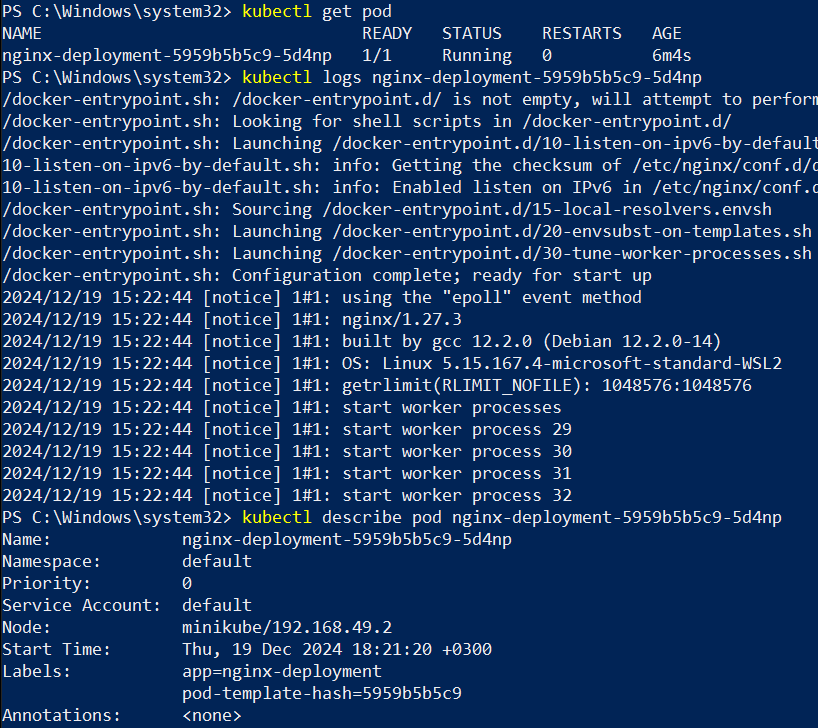


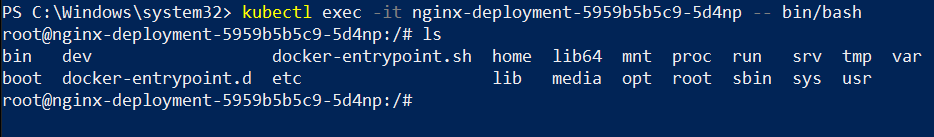


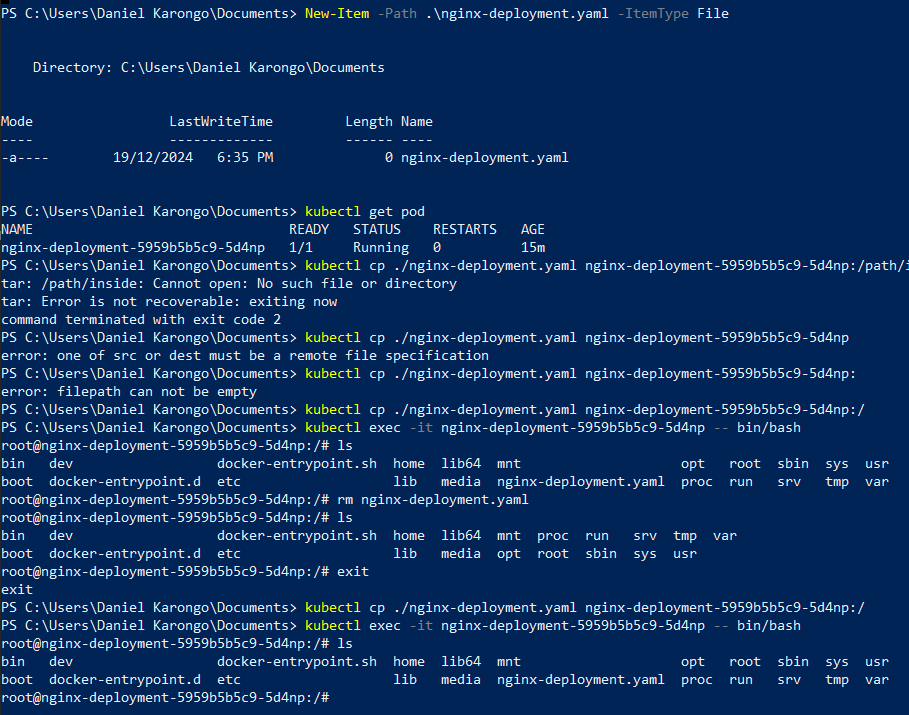


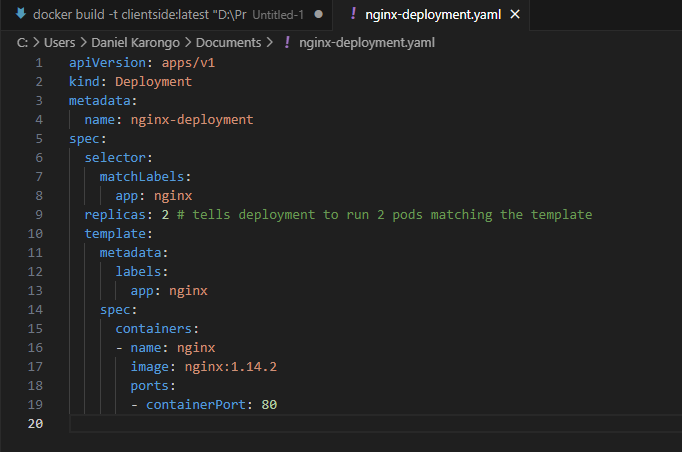


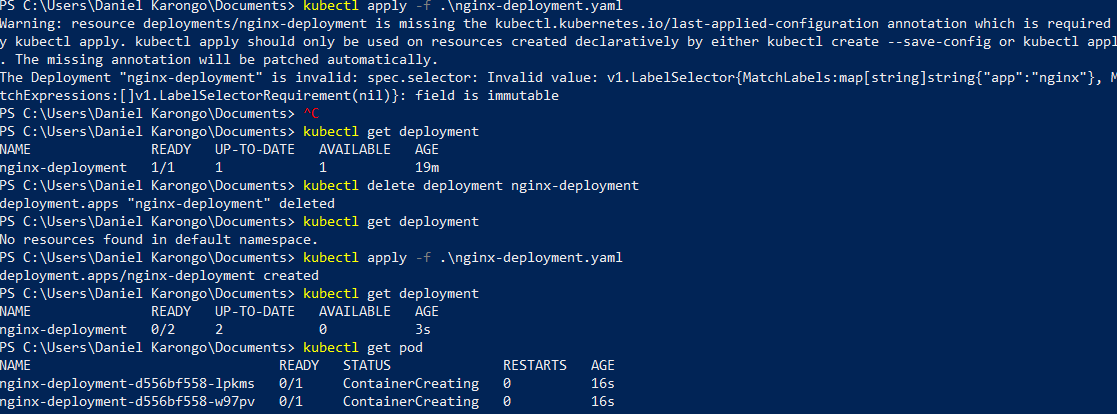




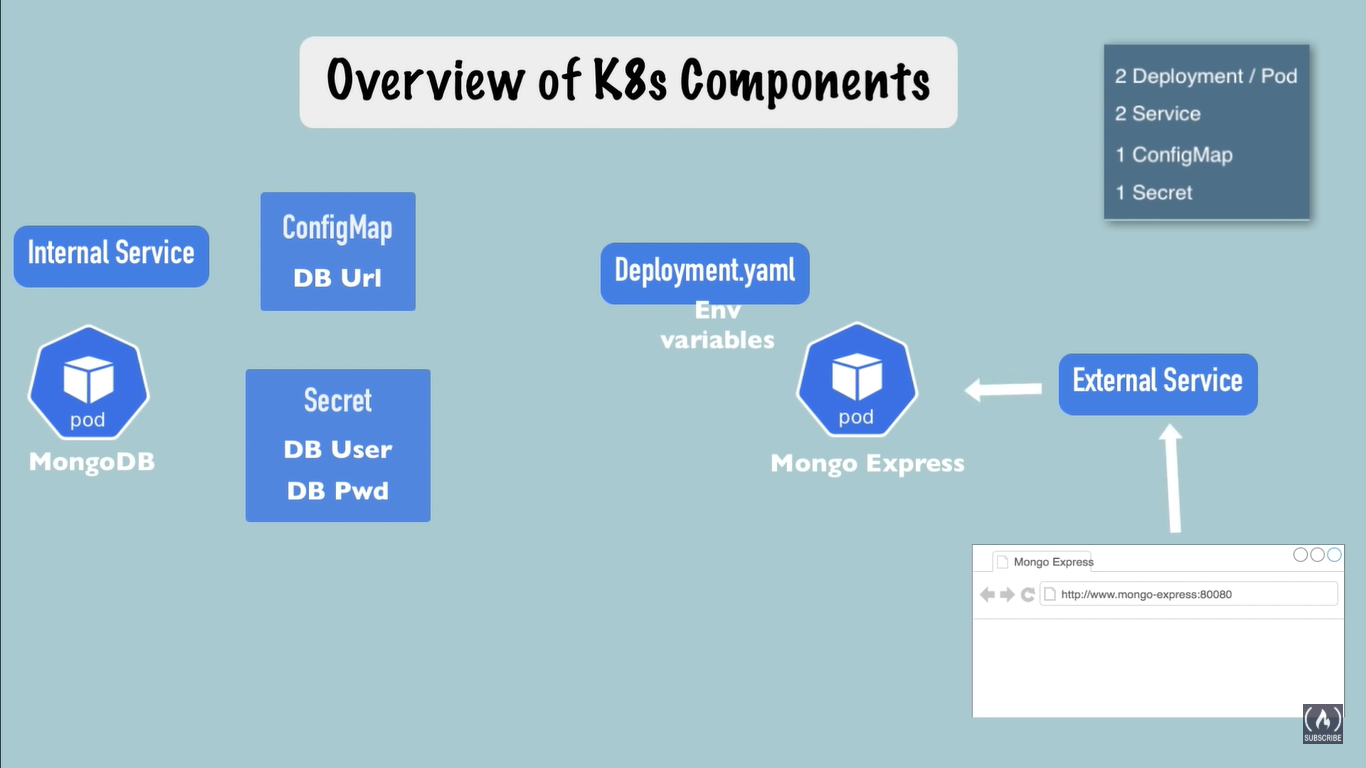


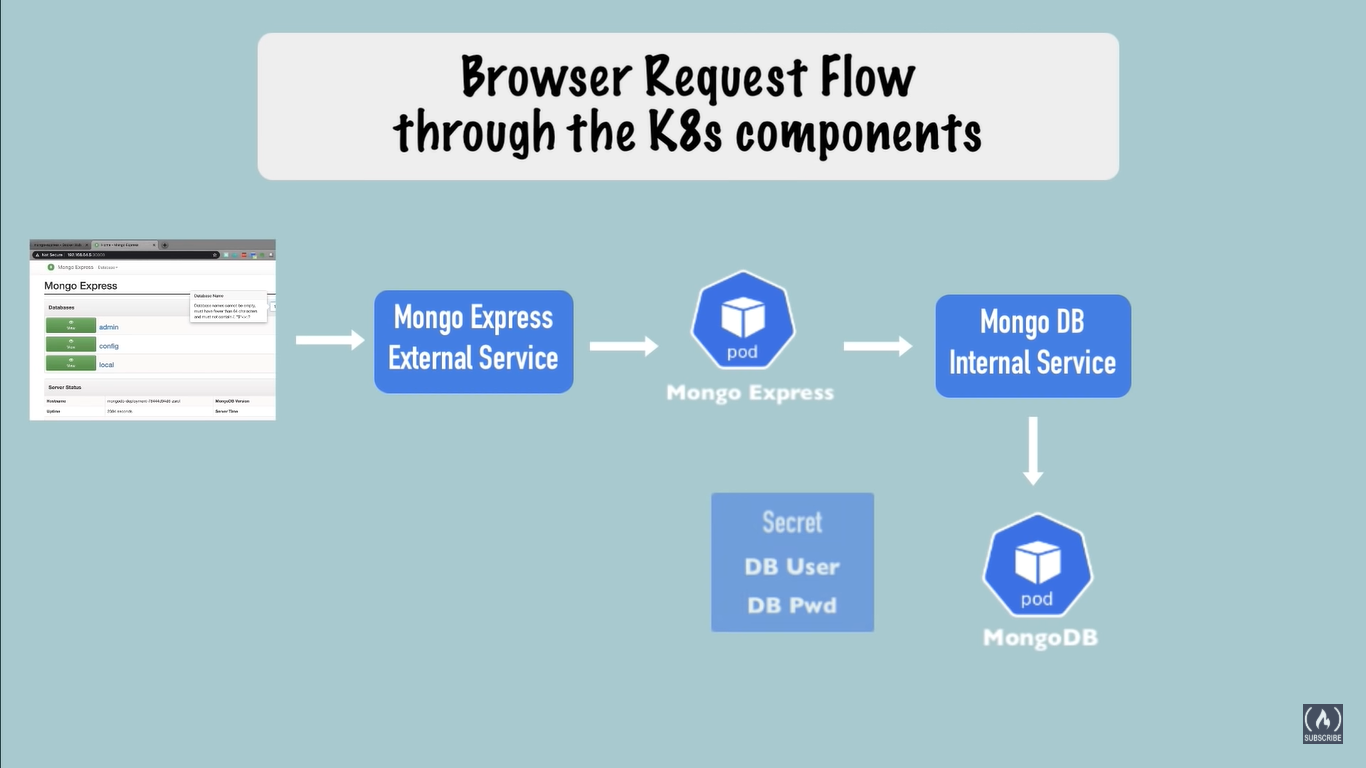






## Deploying Mongo db and Mongo express applications





Consider the following files:

mongodb.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

  labels:

    app: mongo

  name: mongo

spec:

  replicas: 1

  selector:

    matchLabels:

      app: mongo

  strategy: {}

  template:

    metadata:

      labels:

        app: mongo

    spec:

      containers:

      - image: mongo

        name: mongo

        args: ["--dbpath","/data/db"]

        ports:

        - containerPort: 27017

        livenessProbe:

          exec:

            command:

              - mongosh

              - --eval

              - "db.adminCommand('ping')"

          initialDelaySeconds: 30

          periodSeconds: 10

          timeoutSeconds: 5

          successThreshold: 1

          failureThreshold: 6

        readinessProbe:

          exec:

            command:

              - mongosh

              - --eval

              - "db.adminCommand('ping')"

          initialDelaySeconds: 30

          periodSeconds: 10

          timeoutSeconds: 5

          successThreshold: 1

          failureThreshold: 6

        env:

        - name: MONGO\_INITDB\_ROOT\_USERNAME

          valueFrom:

            secretKeyRef:

              name: mongo-creds

              key: username

        - name: MONGO\_INITDB\_ROOT\_PASSWORD

          valueFrom:

            secretKeyRef:

              name: mongo-creds

              key: password

        volumeMounts:

        - name: "mongo-data-dir"

          mountPath: "/data/db"

      volumes:

      - name: "mongo-data-dir"

        persistentVolumeClaim:

          claimName: "mongo-data"

mongo-secret.yaml

apiVersion: v1

kind: Secret

metadata:

  # creationTimestamp: null

  name: mongo-secret

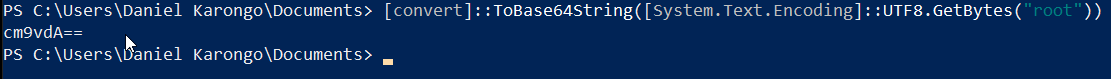
type: Opaque

data:

  password: cm9vdA== #root

  username: cm9vdA== #root

This password is the base64 version gotten as:



Create a deployment of the same: