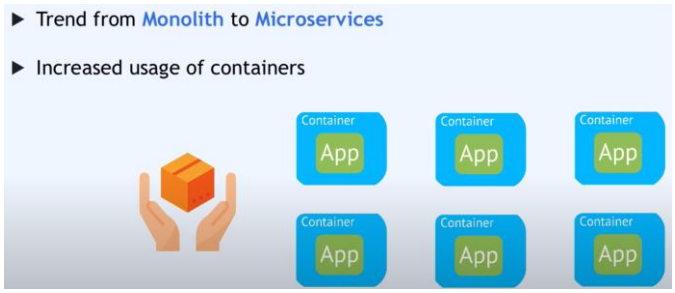
What is Kubernetes?

Kubernetes Architecture

Kubernetes Components

Kubernetes: Open-Source Orchestration tool, helps to **manage containerized applications** in different **deployment environments**.

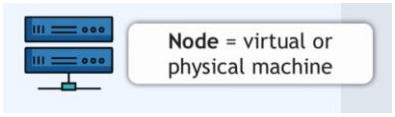
Need for container orchestration tool:

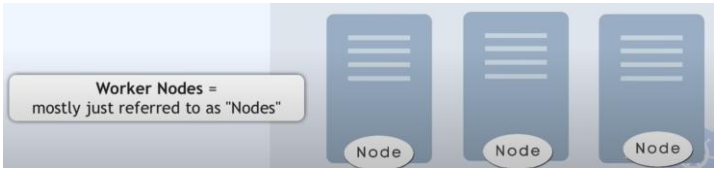


- Demand for a proper way of managing those hundreds of containers.

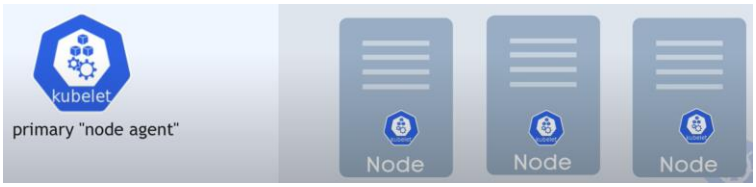
 # Kubernetes Architecture:

- Kubernetes cluster is made up of at least one **master node** and connected to couple of **worker nodes** where each node has a **cubelet** process running on it.

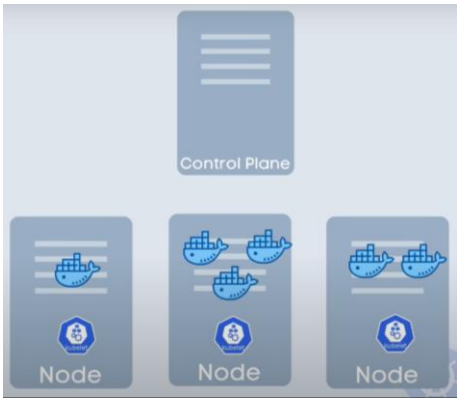




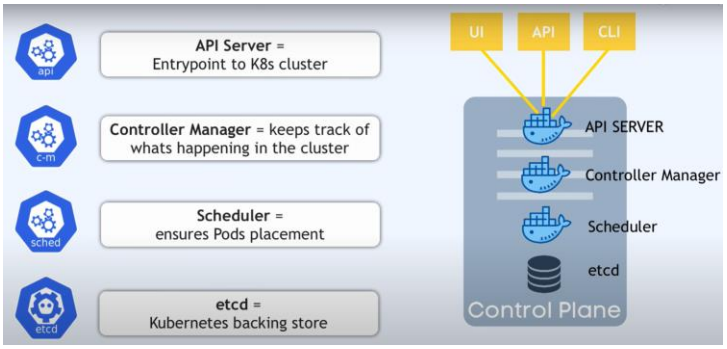
- **Cubelet** is a Kubernetes process that makes it possible for **cluster to communicate with each other** and execute some tasks on those nodes like running application processes.



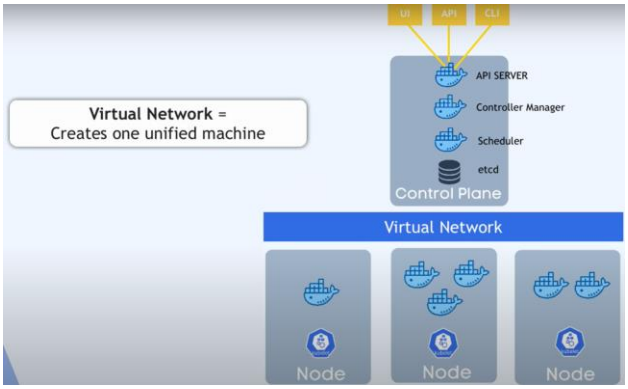
- Each **worker node** has containers of different application deployed on it | Here the actual work is happening i.e., the applications are running.



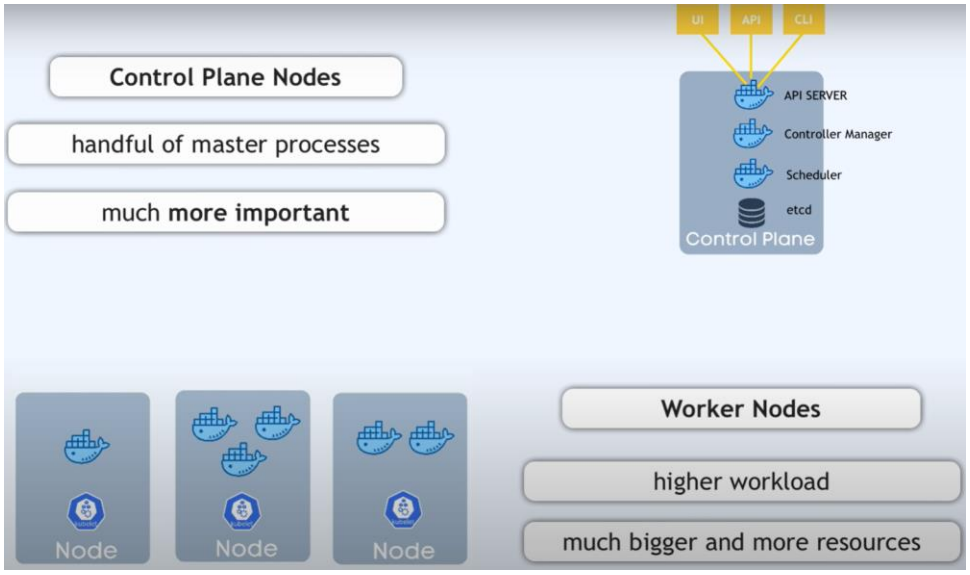
- **Master node** runs the several **Kubernetes processes** that are necessary to run and manage the cluster properly.

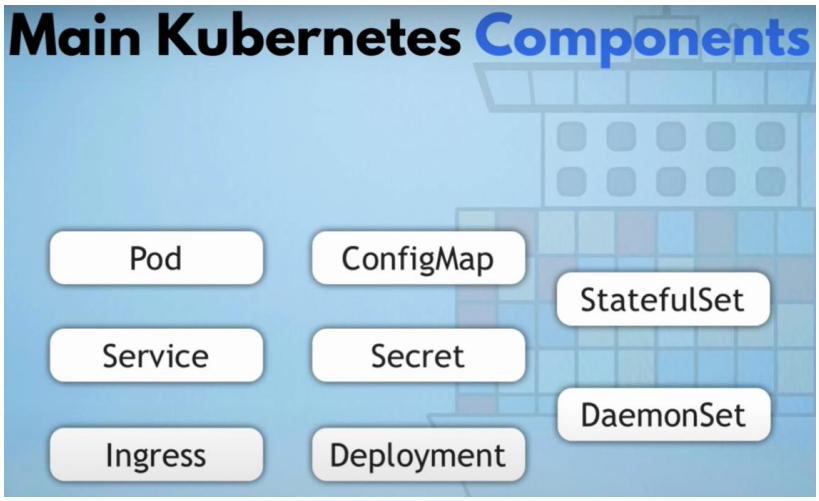


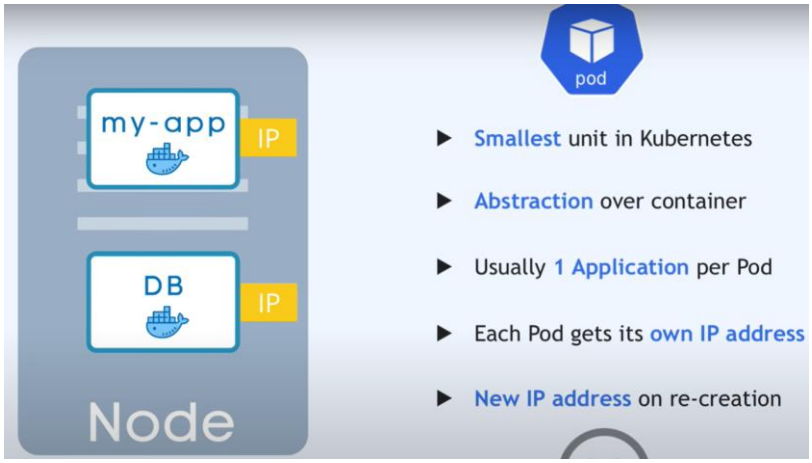
  - One of such processes is **API server** (also a container) which is the **entry point** to the Kubernetes cluster.
  - Another process that is running on the Master Node is **Controller Manager** which keeps an overview of what's happening in the cluster.
  - Another one is **Scheduler** which is responsible for scheduling containers on different nodes based on the workload and the availability of sever resources on each node.
- Another key component of whole cluster is **etcd** key value storage which basically **holds** at **any time the current state of the Kubernetes cluster**.
  - It has all the **configuration data** inside and all the status data of each node and each container inside of that node.
  - **Backup and restore** are made from the **etcd snapshots**.
- Another component is the **Virtual Network** which turns all the nodes inside of a cluster into one powerful machine that has the sum of all the resources of individual nodes.



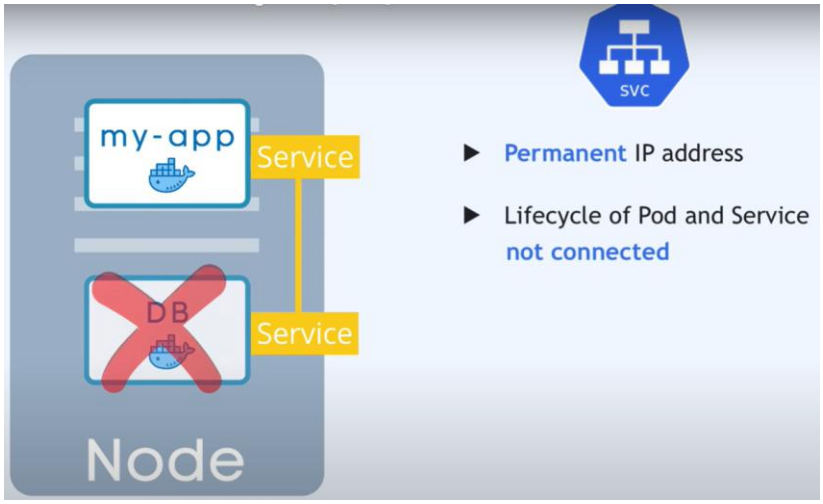
- It basically enables the worker nodes, master node talk to each other.







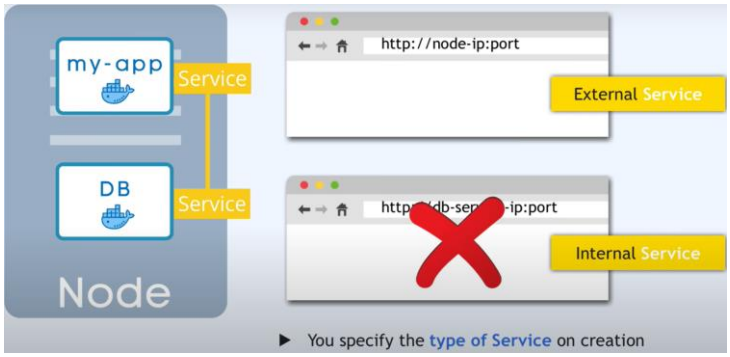
- It creates a **running environment** or a layer on top of the container.
- Each can communicate with each other using IP address (Internal)
- Pods are **ephemeral**, it means that pods can **die very easily**, and a new pod is created and replace the dead one.
- The newly created pod will get assigned with a new IP address.
- Another component: "**service"** is used to resolve the issue of communication through IP address.



- It is basically a **static IP address** or permanent IP address that can be attached to each pod.
- If the pod dies the service and its IP address will stay.

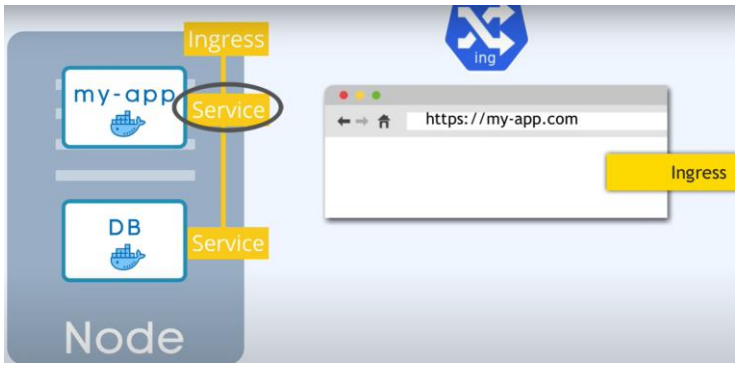
The Application should be accessible through browser.

- To do so you need to create an **External service**: A service that opens the communication from external sources.
- But the database used in our application should not be open to the public.
- So, we create an **Internal service**: A type of service you specify when creating one.

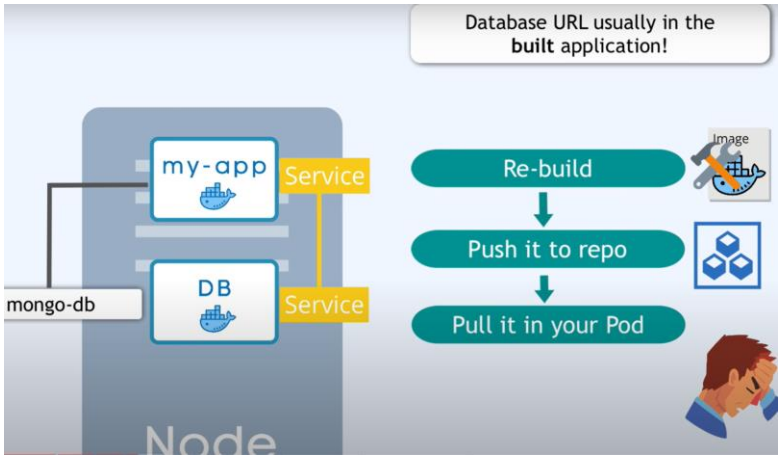


- Here the URL of External Service is not practical.

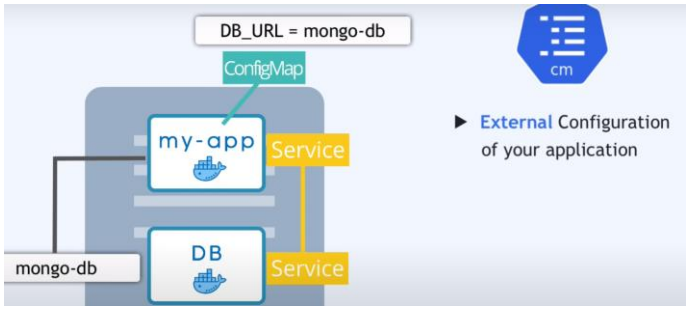
To handle that, there is **Ingress**. So instead of service the **request goes to Ingress** then **forwarded** to **service**.



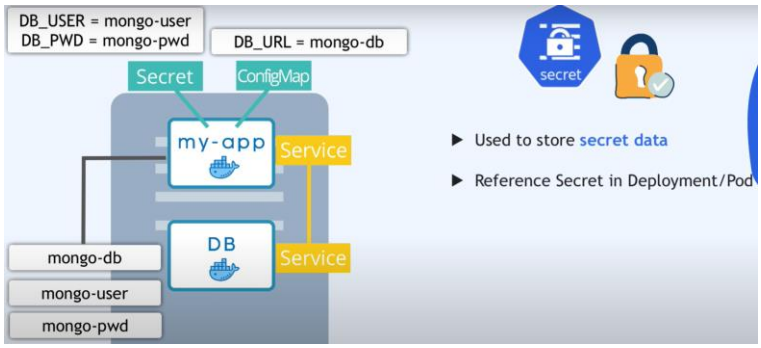
Pods communicate with each other using service.



- If the communication end point names are changes then you must adjust that URL in the application.
- **Configmap**: External Configuration of your application
- It basically contains **configuration data** like URLs of a database and some other services and in Kubernetes you just **connect it to the pod**.
- The Pod gets the data that **configmap** contains.
- If the End point of the service is changed then you need to just adjust the configmap
- ConfigMap is for **non-confidential data only**.

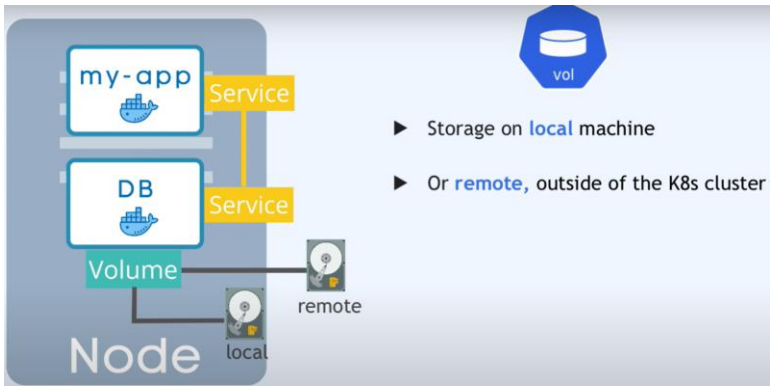


In order to store the **confidential data**, we use **secret**.

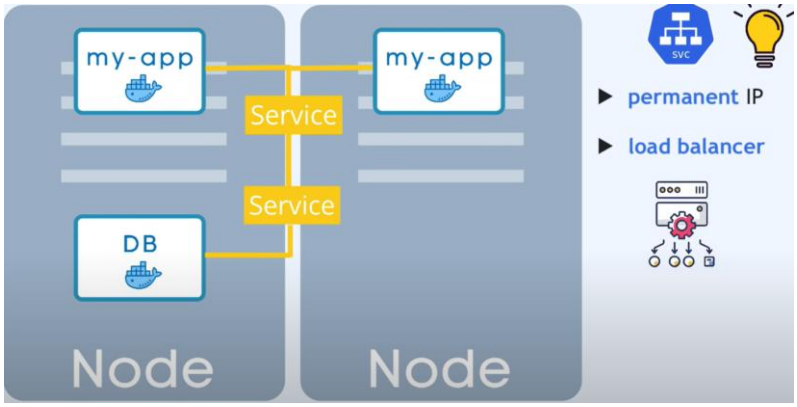


- The data stored in base **64bit encrypted** format.
- It stores the credentials.
- Just need connect the pod to secret.

If the database container or pod is restarted then all the data stored in it would be gone, so to handle this issue we use **Volumes**.



- It basically attaches a **physical storage** on a hard drive to your pod that storage could be either on a **local machine** (same server) or on a **remote storage** (not part of the cluster)
- Kubernetes doesn’t manage data persistence.



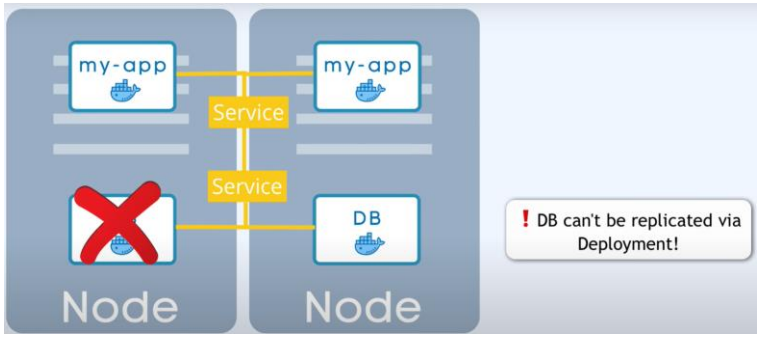
If the application pod crashes or dies, then there will a downtime to users to access the application.

- **Distributed Systems**: Instead of relying on one pod or server we **replicate** everything on different servers.
- The **replica** is **connected** to the **same service**.

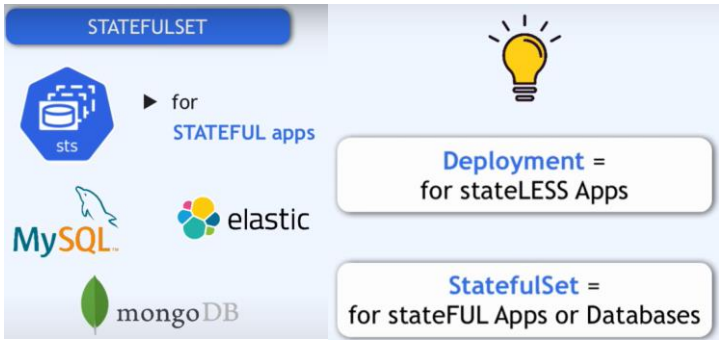
- In order to create the replica of pod we will define a **blueprint** for the Pods.
- **Blueprint**: Specify how many replicas you want to have.
- This blueprint is known as **deployments**.
- So instead of creating pods you will be creating **Deployments**.

Pod is a layer of abstraction on top of containers. Deployment is another abstraction on top of pods.

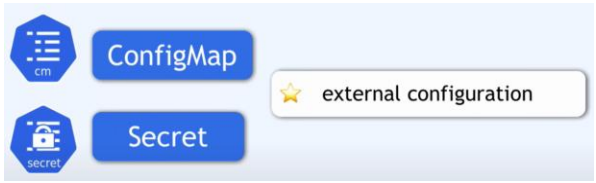
If one of replicas of your application pod will die, then the service will forward the request to another one.

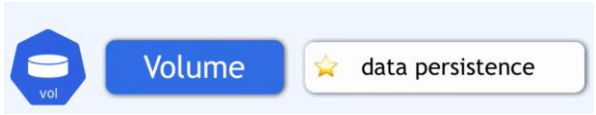


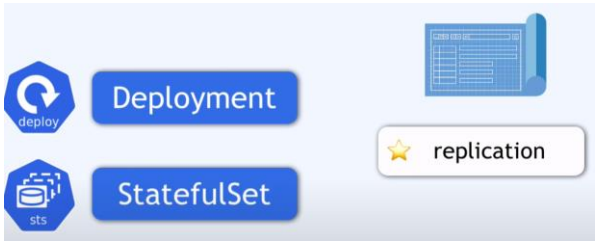
- The **database can't be replicated via Deployment**. Reason: The database has a state.
- All the replicas will have access to the database.



- In order to **replicate the database**, we use **SATATEFULLSET**.
- Deploying StatefulSet is not easy.
- Do avoid any issues it advised to **host the Database outside the Kubernetes cluster**.





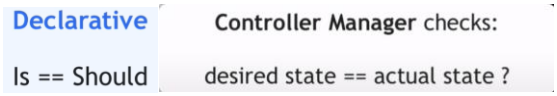


Kubernetes Configuration:

All the **configuration** in cluster goes **through a master node** with the process called **API server**.

- UI, API, CLI send their configuration requests to API server which is the main or only entry point into the cluster.
- These requests must be in **YAML format or Json format**.

- Configuration requests in Kubernetes are in **Declarative form**.



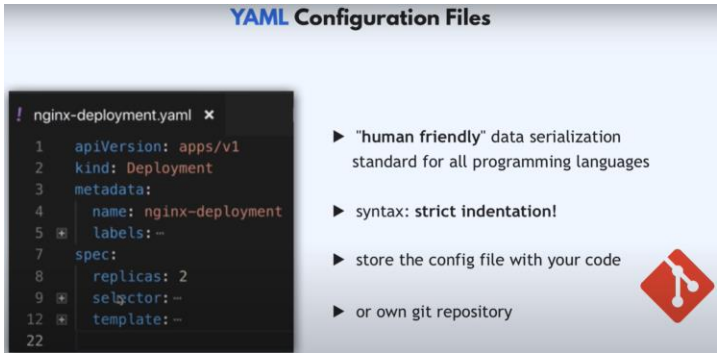
There are 3 parts of a K8s Configuration File:

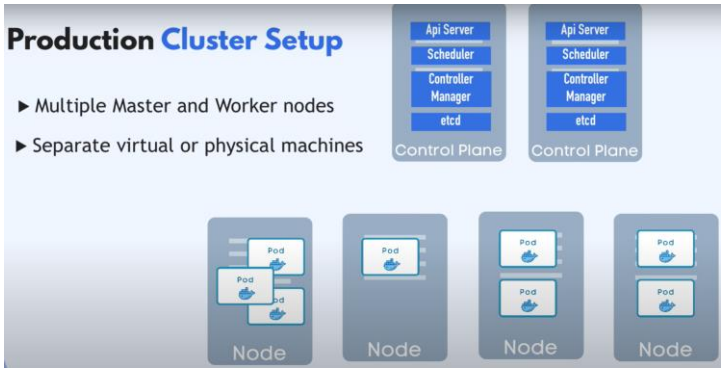
1. **Metadata** of the component
2. **Specification**: Kind of configuration you want to apply
    i. The attributes of “spec” are specific to the kind.

The Highlighted one is the Declaration.

3. **Status**: Automatically generated and added by Kubernetes.

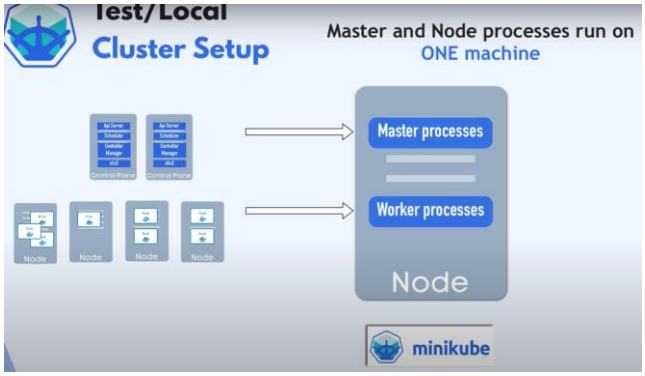
- K8s update the status continuously.
- Where does K8s get this **status data**?
    - **Etcd** holds the current status of any K8s components.





In order to **deploy or test some application on your local machine**, you need to have a **cluster**. To setup this much big cluster is difficult due to lack of resources.

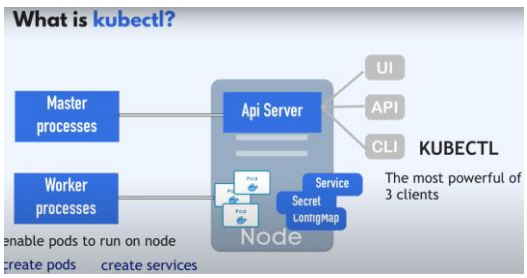
To resolve this problem, we use **minikube**.



Minikube: It is a **one node cluster** where master processes and worker processes run on one node.

- This node will have **docker container runtime pre-installed**.

Now you have **virtual node on your local machine** that represents **minikube** you need some way to **interact** with the **cluster**, you need to create pods and other Kubernetes components on the node.

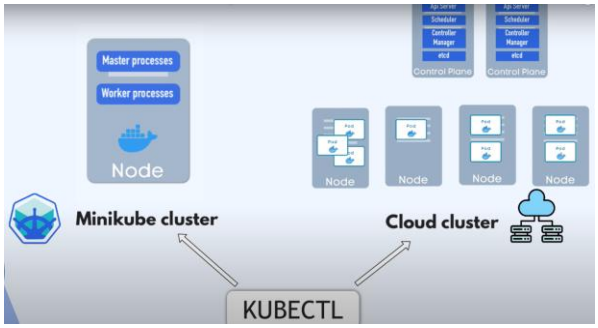


This can be done using **kubectl**: It's a **Command line tool** for K8s cluster.

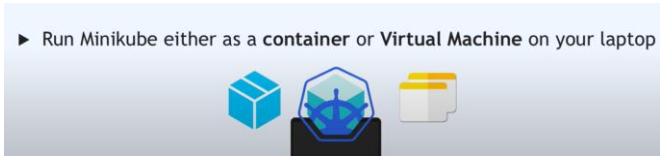
- Minikube is a tool for running a local, single-node Kubernetes cluster, while kubectl is the CLI tool used to interact with Kubernetes clusters, allowing you to manage and control various aspects of your cluster.
- Minikube and kubectl are two separate tools that work together to provide a local Kubernetes development and management environment.
- Minikube is responsible for setting up and running a single-node Kubernetes cluster on your local machine.
- It creates a virtual machine (VM) or uses a container runtime, such as Docker, to simulate a Kubernetes cluster environment.
- Minikube handles the installation and configuration of the necessary components, such as the Kubernetes API server, ... and etcd, to create a functional cluster.

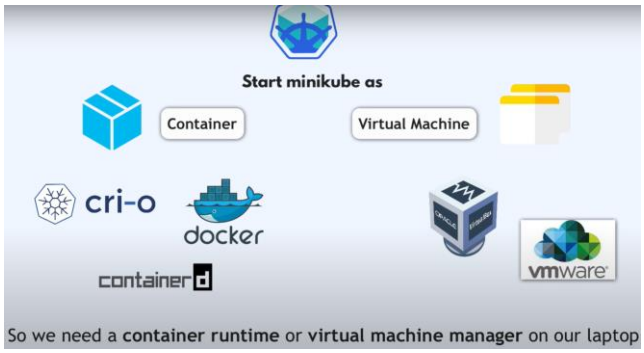
In order to configure anything, you need to talk to API server and this can be done in three ways.

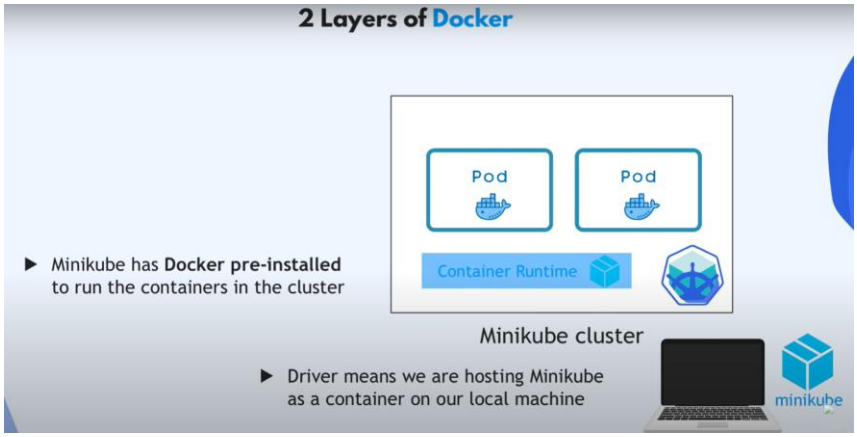
1. API
2. UI
3. CLI – kubectl (most powerful)

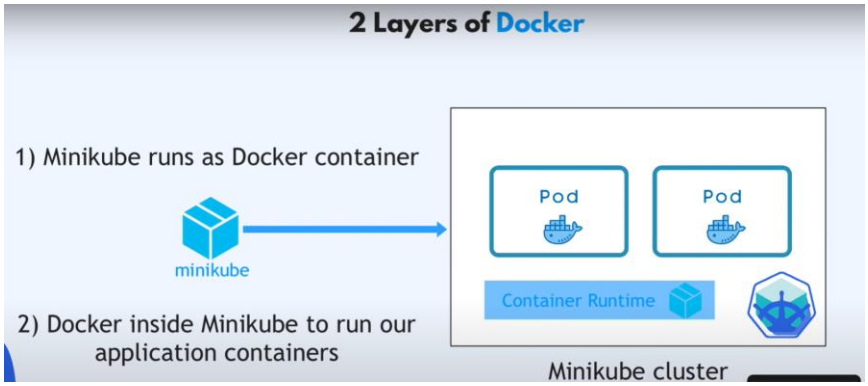


- Kubectl is used to interact with any type of cluster.







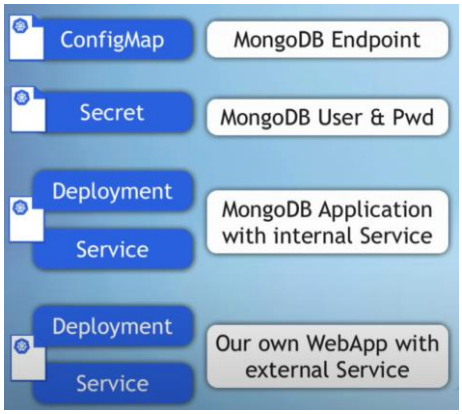


- Kubectl is installed as dependency when we install minikube, so no need to install separately.

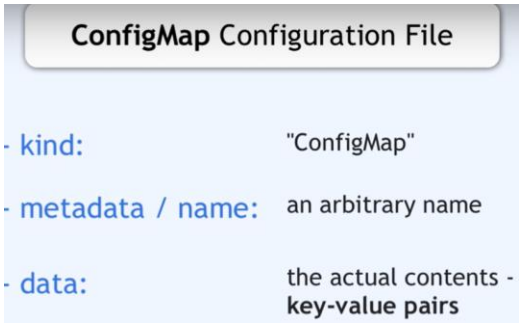
Kubectl CLI ...for configuring the Minikube cluster

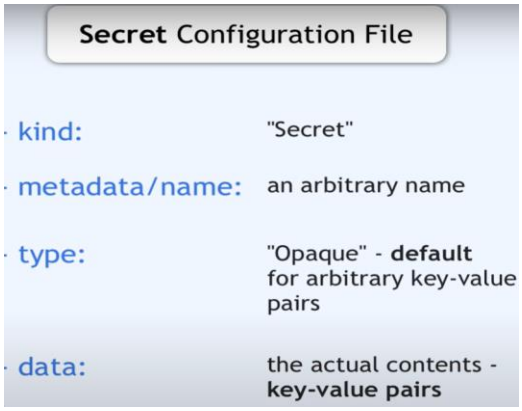
Minikube CLI ...for start up/deleting the cluster

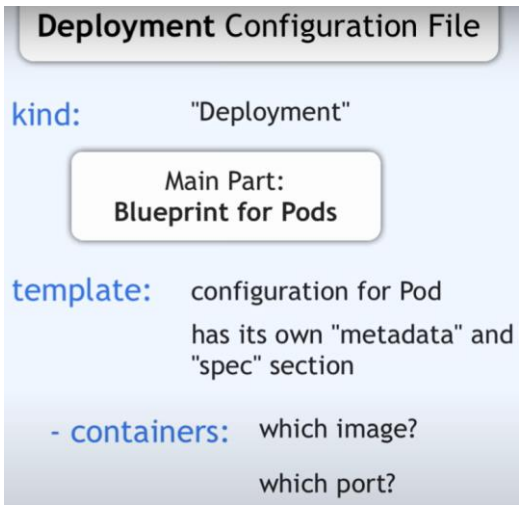
Create and start the cluster: minikube start –driver docker



- Deployment and Service in 1 file, because they belong together.
- Deployment manages Pod.

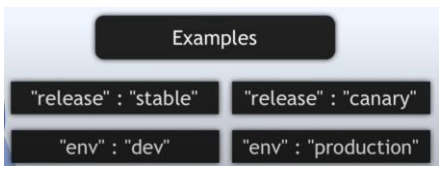






Labels:

- You can give any K8s component a label.
- Labels are key/value pairs that are attached to K8s resources.
- Labels are additional identifiers of the components in addition to name.
- Identify and address specific components using their labels.
- Identifier, which should be meaningful and relevant to users.



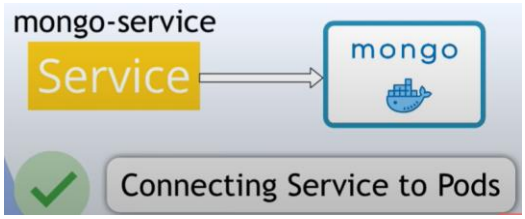
- Labels do not provide uniqueness (all Pod replicas will have the same label).

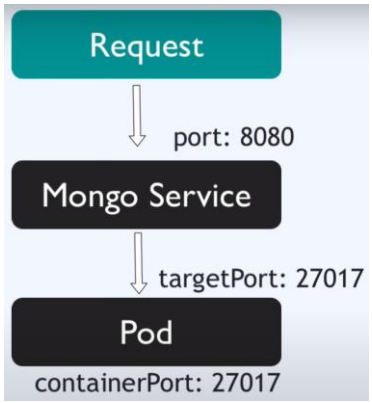
- With the help of labels, we can identify all the part replicas of the same application.

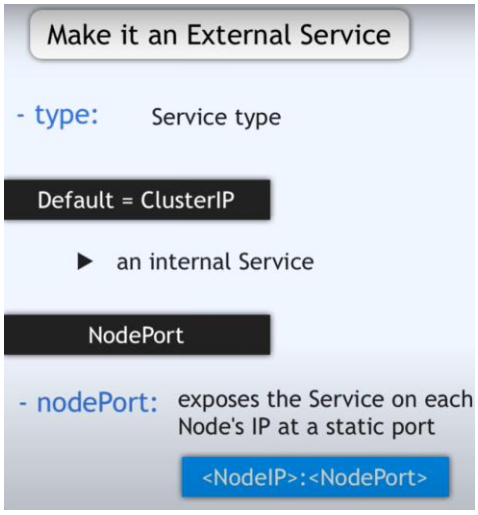
When we create the replicas, how does deployment know which pod belongs to it?

- Label Selector: Identify a set of resources.
    - Match app pods with label (app:nginx)

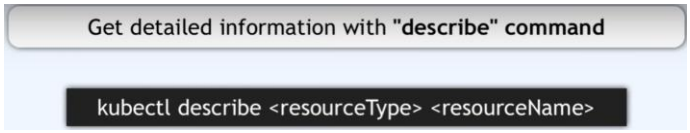
"---" =
You can have multiple YAML configurations within 1 file







- The port range must define between 30000 – 32767.
- ConfigMap and Secret must exist before Deployments.



Issues:

Install the minikube and kubectl using the official documentation and try to switch the internet connection.

Links:

How To Install Minikube on Ubuntu 22.04 LTS (Linux) (2023) - YouTube