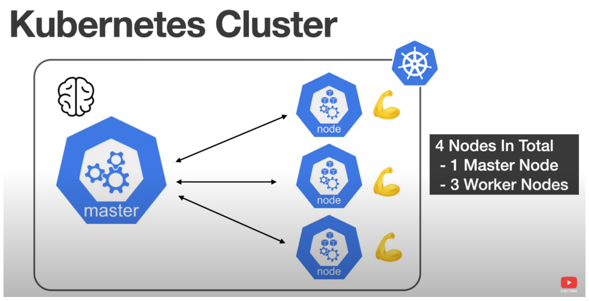
Kuberenetes:

- Also known as K8S because of the name Kuberenetes, the underlined characters are 8.
- Its an application orchestrator i.e it orchestrates all of the applications.
- It deploys and manages our applications (containers).
- Scales up & down according to demand
- Zero Downtime Deployments
- Rollbacks
- And more.

Important Terms:

1. Cluster:
    - A set of nodes.
    - A node could be a physical machine or a VM
    - These machines can be running on cloud such as AWS, Azure, Google Cloud or even on premises.
    - The cluster consists of master and worker node
    - The master node is brain of the cluster where all decisions are made & worker node is the one like muscles where all the heavy lifting happens such as running our applications.
    - Both the master & worker node communicate with each other using kubelet.
    - Generally there are many worker nodes but only 1 or 2 master node.
    - For this example: we have 4 nodes classified into 1 master node & 3 worker nodes:



Kubernetes Cluster

4 Nodes In Total
- 1 Master Node
- 3 Worker Nodes

    - To create local cluster we are going to use minkube.(Need docker + minkube)
2. Master Node:
    - It contains control plane.
    - In Control plane there are several components such as api server, scheduler, cluster store, controller manager & cloud controller manager (which communicates with cloud api such as google cloud,aws,azure) .
    - All these components communicate through api server

3. API SERVER:
    - Frontend to K8S control plane

- All communication go through API Server whether its External communication & Internal communication
- Exposes RestFul API on Port 443.
- Authentication & AUthorization checks are also done

4.

5. Kubectl:
    - Once you’ve local cluster running up with minikube, then to interact with it, We need kubectl.
    - Its a K8S command line tool

    - Run commands against our cluster
        1. Deploy applications
        2. Inspect
        3. Edit resources
        4. Debug our cluster
        5. View Logs

3. Pods:
    - A pod is the smallest deployable unit in K8S.
    - It contains main container which represents your application and may/may not have init container which runs before main container, and may/may not have side container to support main container.
    - Also it contains volumes to share data among themselves(containers)
    - The containers inside pod,communicate to each other using localhost ports
    - The pod also has a unique ip, it means another pod wants to talk to this pod then it uses this unique ip address.

    Commands:

    1. minikube start –node=2 -> This will create a cluster with 2 nodes.
    2. minikube config set driver docker -> set minikube driver for docker by default.
    3. minikube status -> returns all minikube nodes which were created with detailed description such as their type like control-plane and worker.
    4. kubectl get nodes -> returns all nodes in tabular format
    5. kubectl get pods -A -> gets all pods in all namespaces

Kuberenetes Nana Course

Kuberenetes:
It is a Open source container orchestration tool. Developed by google. Helps you managed containerized applications in different deployment environments.

Need for container orchestration tool:

1. Trend From Monolith to Microservcies
2. Increased usage of containers.
3. Demand a proper way of managing those hundred of containers.

Features of container orchestration tool:

1. High Availablilty / no Downtime.
2. Scalability or High Performance. i.e If there are more no. of users are trying to acces it i.e more load then scale it up fast & scale it down if there is less load.
3. Disaster recovery - restore & backup.

Kuberenetes Architechture:
It consists of a **master node** & several worker nodes.This collection of nodes is called as cluster.
The master node has following **4** components:

1. API Server: Entrypoint to k8 cluster i.e its is a cluster gateway
2. Controller Manager: Keeps tracks of what's happening in cluster.
3. Scheduler: ensures pods placement i.e it checks which worker node to allocate container for running.
4. etcd: Kind of key-Value Stor.A master process storing cluster data,holds currents status of of any k8's components. It has handful of master processes & is important. It is called cluster brain. **Application data is not stored in etcd**

The **worker nodes** have higher workload & is much bigger & has more resources.
The master & worker nodes communicate each other over virtual n/w.
Note: Have 2 master nodes in production environment as they we lose acces to one of them, 2nd one as a backup should be there.

MAIN K8s Components
Abstraction of containers

1. **Pod:**

1. Smallest unit of container.
2. Abstraction over container.
3. Usually one application per pod.
4. Each pod gets its own IP.The master & worker nodes communicate each other over virtual n/w.
5. New IP address on re-creation if older pod gets destroyed & thus another component named service is used instead of IP address.

Communication

2. Service:

1. Permanent IP address.
2. Lifecycle of Pod & Service are not connected=>even if pod dies then service & its pod stays.

Route traffic into cluster

3. **Ingress :**

1. When a request is made to my-app service then it shall first go to Ingres & not to the service.
2. It’s like domain name mapped to service which is an ip i.e static ip.

External Configuration to the application

4. Config Map: Will contain DB_URL(Database URL), so app can connect to the db.

5. Secret:

1. Will contain username & password to connect the db. It’s stored in base 64 in encoded format.
2. The secrets components are meant to be encrypted using third party tools in kuberenetes , as kuberenets doesn’t encrypt them out of the box.
3. Reference secret in Deployment/Pod.

Data Persistence

Volumes:

1. Storage on local machine or
2. Storage on remote machine, outside of K8s clusters.

Note: Kuberenets doesn’t manage data persistence.

Replication

Deployment:

1. Blueprint for “my-app”(application) pods .i.e deployment for creating pods.
2. We will create deployments rather than pods, incase a pod has downtime or dies.
3. It is abstraction of pods.

Stateful set:

1. Used when, worker node contains database pod.
2. Database read & write are synchronized, so that database inconsistencies are offered.
3. Deploying database apps using stateful state is not easy in k8’s cluster,so DB are often hosted outside of the k8’s cluster

Note: -

1. Deployment = for stateLESS apps.
4. Stateful Set= for stateFUL Apps or databases.
5. DB are often hosted outside kuberentes cluster.

K8’s Architecture:

1. We have several worker nodes
2. Each node contains pods,container runtime & kubelet.
3. Kubelet is responsible for starting the pod with a container inside.
4. Also kubelet assigns resources(cpu,ram & storage) from that node to the container
5. During replication of a node the app should not send the request from my-app in node 1 to database in another node, it should send the request to the same database which is associated with it i.e. in same node.
6. 3 Node processes :Kubelet,Kube Proxy & Container runtime should be there on each node

Removing Minikube:

```
  minikube stop; minikube delete && docker stop $(docker ps -aq) && rm -rf ~/.kube ~/.minikube && sudo rm -rf
/usr/local/bin/localkube /usr/local/bin/minikube && launchctl stop '*kubelet*.mount' && launchctl stop
localkube.service && launchctl disable localkube.service && sudo rm -rf /etc/kubernetes/ && docker system prune
-af --volumes
```

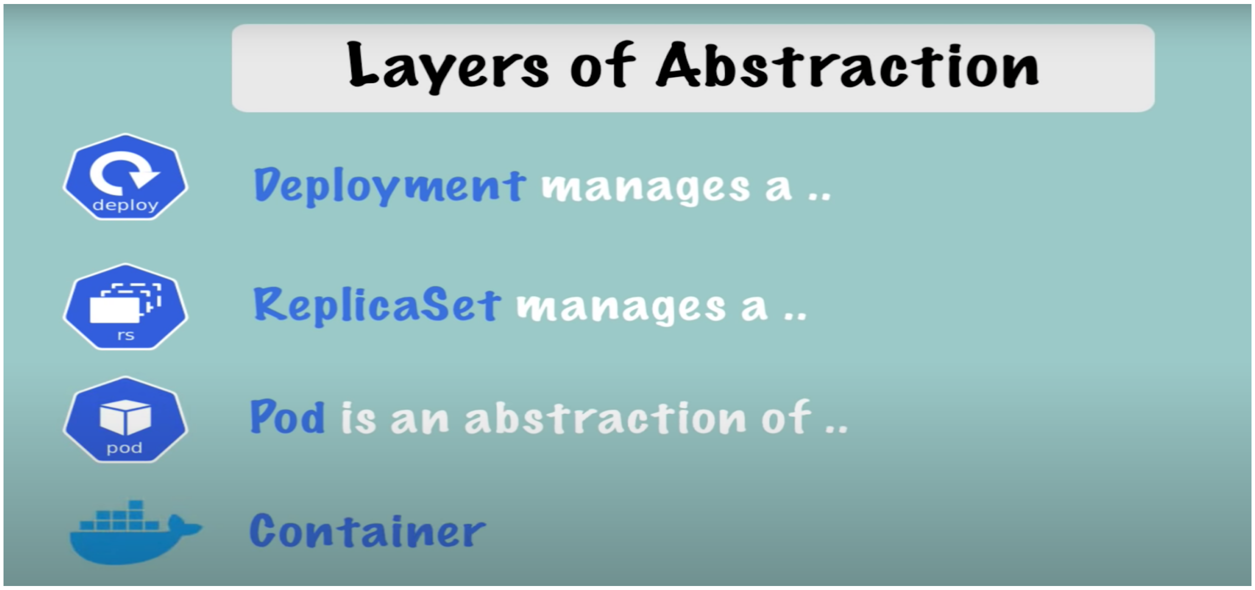
Ref: https://stackoverflow.com/questions/73600518/how-do-i-uninstall-minikube-on-a-mac

Kubectl commands:

1. kubectl get nodes: Get nodes i,e minikube ,it’s roles & version.
2. kubectl create deployment nginx-depl --image=nginx -> bluprint for creating pods,most basic configuration for deployment.(name & image to use). C**reate,Delete& Update happens at deployment level**
3. kubectl get replicaset-> gets replicaset, manages replicas of pod

```
Shubhams-MacBook-Air:~ shubhamphansekar$ kubectl get pod
        NAME                      READY   STATUS    RESTARTS   AGE
nginx-depl-c88549479-jv5f8   1/1        Running          0        75s
Shubhams-MacBook-Air:~ shubhamphansekar$ kubectl get replicaset
        NAME                 DESIRED   CURRENT   READY   AGE
        nginx-depl-c88549479   1            1            1      3m48s
```

NOTE: the replicaset name has prefix & middle part is same as pod name



Layers of Abstraction

Deployment manages a ..

ReplicaSet manages a ..

Pod is an abstraction of ..

Container

Everything below Deployment is handled by Kubernetes

4. kubectl apply -f nginx-deployment.yaml -> used to create/update deployments. If you already created a pod using using this, and then change replica to 2 it will update the pods to 2 rather than creating one.

   In general we can creat/update various components such as deployment/service/volumes.

Kuberenetes Configuration:

1. They are declarative.
2. Is == Should

Each Configuration File has 3 parts:

1) Metadata
2) Specification: Attributes of specification are specific to kind.
3) Status: it is automatically generated & added by Kubernetes. Checks for desired state(spcification) & actual status(status) i.e Is == Should ?. If they don’t match then kubernetes knows there’s something to be fixed. So it tries to fix it.

YAML:

1. “human friendly” data serialization standard for all programming languages.
2. has strict indentation
3. store the config file with your code.
4. You can also have git repo for it.

Minikube & Kubectl:

Production Cluster Setup:

1. Multiple Master and Worker node
2. Seperate virtual & physical machines

MiniKube:

1. A node cluster where master processes & worker processes both run on one node
2. This node will have docker-container pre-installed.
3. SO we will be able to run the containers or pods with containers on this node.

Kubectl:

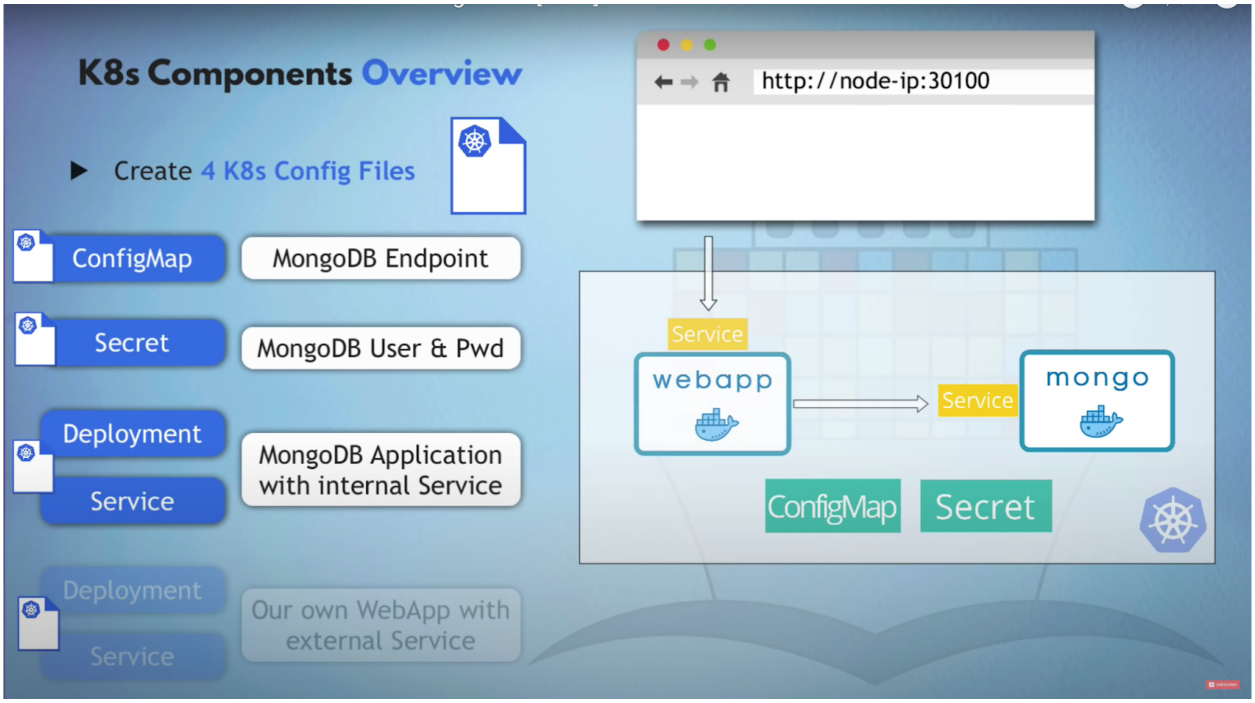
1. Used to create pods & other kuberenetes components on the node.
2. Command line tool for kubernetes cluster.
3. Most powerful of 3 clients to interact with api server, other 2 being UI & API
4. It submits commands to api server to create, delete components etc

5. Then the worker processes on minikube will actually make it happen i.e. worker process will actually executing commands. Eg: creates pods, destroys pods & create services.
6. Kubectl is used to interact with any type of the cluster setup i.e Minikube cluster,cloud cluster or hybrid cluster

Note: Kubectl CLI for configuring the MiniKube cluster
Minikube CLI for start up/deleting the cluster.

Commands:

1. minikube start --driver docker => creates & starts local kubernetes cluster
2. minikube status => gives status of component
3. kubectl get node=> gets status of nodes i.e



K8s Components Overview

▶ Create 4 K8s Config Files

ConfigMap	MongoDB Endpoint
Secret	MongoDB User & Pwd
Deployment Service	MongoDB Application with internal Service
Deployment Service	Our own WebApp with external Service

Config Map: Configuration file(**mongo-config.yaml**)

- kind : “Config Map”
- metadata / name : an arbitrary name (name: mongo-config)
- data : the actual contents- key-pairs (mongo-url: mongo-service)

Secret : Configuration file(**mongo-secret.yaml**)

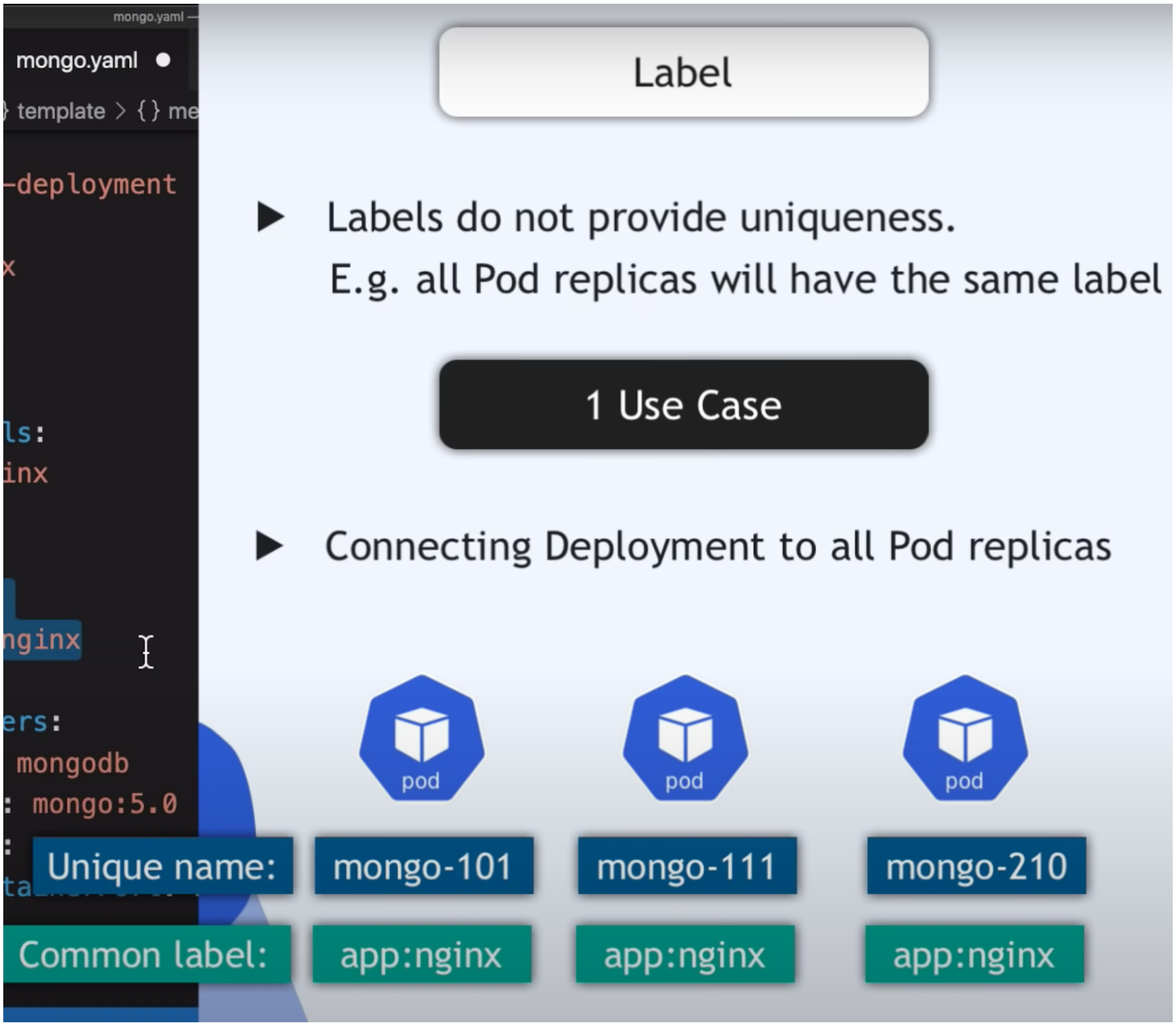
- kind : “Secret”
- metadata / name : an arbitrary name (name: mongo-secret)
- type : “Opaque” - default for arbitrary key-value pair
- data : the actual contents- key-pairs (mongo-user : bW9uZ291c2Vy & mongo-password : bW9uZ29wYXNzd29yZA==)

The above file Config & Secret can be referenced by different deployments file.

Note: Deployment & Service should be in 1 file because they belong together as deployment need services so you have grouped them in 1 yaml file.

Deployment : Configuration file (generally has blueprint for defining pods as deployment manages pod) (part of **mongo.yaml**)

- kind : “Deployment”
- template : configuration for Pod, has its own **“metadata”** and **“spec”** section
- The spec section of template has containers section which indicates about image by which containers can be created for eg: mongo 5.0 is the img by which we’ll create a container.
- labels are the key-value pairs, It’s an identifier which should be meaningful & relevant to users.



```
-deployment

x

ls:
inx

nginx

ers:
 mongodb
: mongo:5.0
:
ta
```

Label

- Labels do not provide uniqueness.
  E.g. all Pod replicas will have the same label

1 Use Case

- Connecting Deployment to all Pod replicas

Unique name:	mongo-101	mongo-111	mongo-210
Common label:	app:nginx	app:nginx	app:nginx

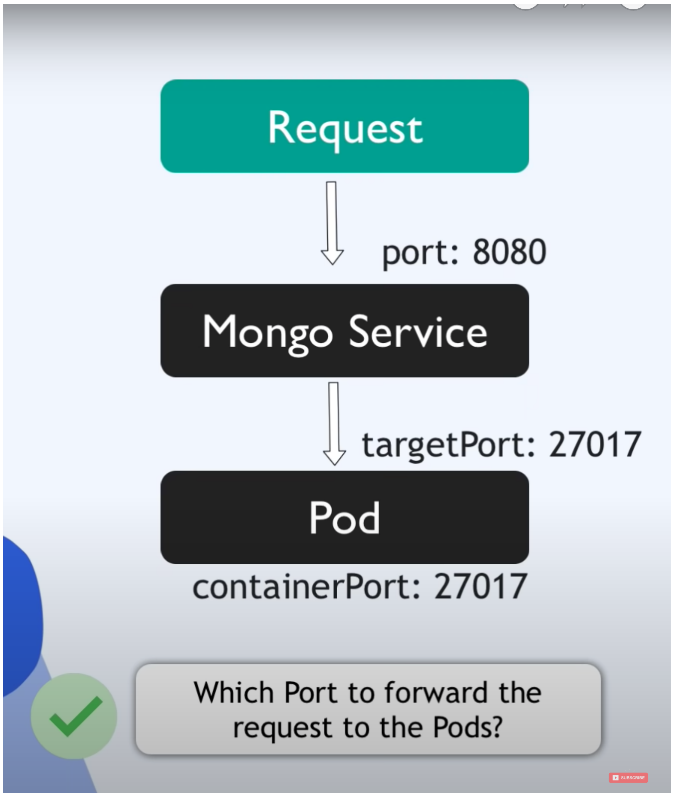
- Label Selectors allows kubernetes to know which pod belong to deployment. By seeing this,

```
selector:
    matchLabels:
                app: mongo
```

Thus app:nginx is a label identifier which belongs to the above deployment

- Replicas indicate how many pods you want to create.

Service: Configuration(added in deployment file as a different section) (part of **mongo.yaml**)

- kind : “Service”
- name: an arbitrary name(This name should be same as the name given in config file)
- selector: selects pods to forward the request to
- port: indicates service port i.e service is accessible within its cluster using its own port
- targetPort: It is the port of the pod,should be same as containerPort of Deployment that belong to the service.
- 

Request

port: 8080

Mongo Service

targetPort: 27017

Pod

containerPort: 27017

Which Port to forward the request to the Pods?

Passing secret data to Mongo Deployment:

- env should have secret data i.e username & password.

Passing Config data to WebApp Deployment

- WebApp needs information about:
    1. Database Endpoint => Refer from ConfigMap
    2. Database Username => Refer from Secret
    3. Database Password => Refer from Secret

WebApp should be accessible from browser,Use **external service** for that & changes would be:

- type: NodePort (Default is Cluster IP)
- nodePort: exposes the Service on Node's IP at a static port.(must be between 30000 - 32767)

kubectl apply -f *.yaml : Do this for all yaml files but do it first for mongo-config & mongo-secret file.

Interacting with K8 clusters

1. kubectl get all : return with all components created in cluster. The components include pod,svc & deploy.

   ```
   Shubhams-MacBook-Air:K8S-DEMO shubhamphansekar$ kubectl get all
   NAME                              READY   STATUS    RESTARTS   AGE
   pod/mongo-deployment-65ffdd9df6-74pf6    1/1     Running   0          2m11s
   pod/webapp-deployment-65d4754f9d-p64xd   1/1     Running   0          103s

   NAME                  TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
   service/kubernetes       ClusterIP   10.96.0.1       <none>        443/TCP          7m43s
   service/mongo-service    ClusterIP   10.100.67.191   <none>        27017/TCP        2m11s
   service/webapp-service   NodePort    10.109.16.122   <none>        3000:30100/TCP
   103s

   NAME                               READY   UP-TO-DATE   AVAILABLE   AGE
   deployment.apps/mongo-deployment    1/1     1            1           2m11s
   deployment.apps/webapp-deployment   1/1     1            1           103s

   NAME                                          DESIRED   CURRENT   READY   AGE
   replicaset.apps/mongo-deployment-65ffdd9df6    1         1         1       2m11s
   replicaset.apps/webapp-deployment-65d4754f9d   1         1         1       103s
   ```

2. kubectl get configmap: returns configmap

   ```
   NAME               DATA   AGE
   kube-root-ca.crt   1      9m46s
   mongo-config       1      5m2s
   ```

3. kubectl get secret: returns secret

   ```
   NAME           TYPE     DATA   AGE
   mongo-secret   Opaque   2      5m13s
   ```

4. Kubectl get pod: returns pod

```
NAME                                READY   STATUS    RESTARTS   AGE
mongo-deployment-65ffdd9df6-74pf6    1/1     Running   0          7m42s
webapp-deployment-65d4754f9d-p64xd   1/1     Running   0          7m14s
```

5. kubectl describe service webapp-service=> describe keyword is used to return with more information, for that specific component. Here it is service.

   ```
   Name:                     webapp-service
   Namespace:                default
   Labels:                   <none>
   Annotations:              <none>
   Selector:                 app=webapp
   Type:                     NodePort
   IP Family Policy:         SingleStack
   IP Families:              IPv4
   IP:                       10.109.16.122
   IPs:                      10.109.16.122
   Port:                     <unset>  3000/TCP
   TargetPort:               3000/TCP
   NodePort:                 <unset>  30100/TCP
   Endpoints:                172.17.0.4:3000
   Session Affinity:         None
   External Traffic Policy:  Cluster
   Events:                   <none>
   ```

6. kubectl describe service webapp-service=>returns with description of webapp-service

   ```
   Name:                     webapp-service
   Namespace:                default
   Labels:                   <none>
   Annotations:              <none>
   Selector:                 app=webapp
   Type:                     NodePort
   IP Family Policy:         SingleStack
   IP Families:              IPv4
   IP:                       10.109.16.122
   IPs:                      10.109.16.122
   Port:                     <unset>  3000/TCP
   TargetPort:               3000/TCP
   NodePort:                 <unset>  30100/TCP
   Endpoints:                172.17.0.4:3000
   Session Affinity:         None
   ```

```
External Traffic Policy:  Cluster
Events:                   <none>
```

7. kubectl get pod=> returns all pods.
8. Kubectl get service/svc=> returns all service

   ```
   NAME             TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
   kubernetes       ClusterIP   10.96.0.1       <none>        443/TCP          23m
   mongo-service    ClusterIP   10.100.67.191   <none>        27017/TCP        17m
   webapp-service   NodePort    10.109.16.122   <none>        3000:30100/TCP   17m
   ```

   The above web-app service is displaying 30100 port but on which IP?
   The NodePort Service is accessible on each worker node's ip address. In our case it's only one i.e minikube.

9. minikube ip => returns the ip => 192.168.49.2
10. kubectl get node =>

    ```
    NAME       STATUS   ROLES           AGE   VERSION
    minikube   Ready    control-plane   30m   v1.25.0
    ```

11. kubectl get node -o wide=> returns wide output of node.

    ```
    NAME       STATUS   ROLES           AGE   VERSION   INTERNAL-IP    EXTERNAL-IP
    OS-IMAGE             KERNEL-VERSION      CONTAINER-RUNTIME
    minikube   Ready    control-plane   31m   v1.25.0   192.168.49.2   <none>        Ubuntu
    20.04.5 LTS   5.10.124-linuxkit   docker://20.10.17
    ```