# Kubernetes

* Kubernetes was designed by google for container management.
* It is open sources.
* Helps manage applications with up hundreds of thousands of containers.

**Problems it solves.**

* Containers are perfect for microservices.
  + The rise of microservices resulted in Kubernetes so that we can manage it properly
  + Container orchestration
* Kubernetes guarantees.
  + High availability
    - App has no downtime.
    - Always accessible by users
  + Scalable
    - High performance
    - Loads fast.
    - Users have high response rate.
  + Disaster recovery
    - If something happens or service terminates, it will restore to the latest state

**Kubernetes Components**

* Nodes and Pods
  + Worker node
    - A simple server
    - Physical or virtual machine
  + Pod
    - Smallest unit in Kubernetes
    - Abstraction over a container
    - Creates a running environment/layer on top of container.
    - Only interact with Kubernetes layer
    - Pod is meant to run ONE APP CONTAINER inside.
    - Each pod gets its own IP
      * Not the container
    - Each pod can communicate with the IP(internal)
    - Affirmer
      * Can die easily
      * New IP gets assigned
* Services
  + Permanent IP address that can be attached to a pod
  + Lifecycles of service and pod are not connected
    - If pod dies, the service and IP PERSIST
* External Service
  + Opens communication from external sources
* Internal Service
  + Service that’s private

**ConfigMap and Secret**

* Pods communicate with each other with services
* External configuration to the application
* Contains:
  + url of a databases you are using in app
  + connect it to the pod so it gets the data in the config map
  + if you change the end point of the service you just have to change the map
  + don’t put credentials in here
    - not secure
* Secrets
  + Like configmap but used to store secret data such as credentials
  + Stored in base 64 format
  + Certificates, passwords
  + Connect to pod and the pod can read it

**Data Storage**

* How to have data persist in a down pod
  + Volumes!
    - Attaches a physical storage to your pod
    - Could be on a local machine, or in remote outside of the cluster
      * Cloud, etc
    - “external hardrive plugin”
    - Cluster does not manage data persistence
      * Our responsibility

**Deployment and Stateful Set**

* Everything is replicating everything
  + if app pod dies
    - downtime occurs
      * bad
  + on multiple servers
  + another node would have a replica of our app
    - with another service
    - service is also a load balancer.
      * this mean it will forward thew request to the pod that is less busy aka able to take on the client.
      * define blueprint for pod.
        + this is called deployment.
  + **Deployment**
    - Specify how many replicas.
    - Scale up replicas of pods
    - Pod is a layer of abstraction
    - Deployment is another abstraction above pods

**If a pod dies, the service will forward the request to the replica and the run that**

What about the database?

* We cannot replicate a database with a deployment because of STATE
* If we had clones they would all need to access the same shared data storage
  + Need to know whos writing, whos reading etc data inconsistencies
* **Stateful set**
  + Meant for databases
    - MySQL, MongoDB
    - Databases should be made with stateful sets
  + Replicates the pods and scales them up or down
  + Deploying database apps with stateful sets can be tedius

**Kubernetes Architecture**

* Master nodes
* Slave nodes

Node Processes

* One node, two app pods running
* Worker server/node
* Each node can have multiple pods on it
* 3 processes that must be installed
  + Container runtime
    - Ie. Docker
  + Kublet
    - Interacts with container runtime and the node itself
    - Responsible for taking the configuration and starting the pod with acontainer inside
    - And assigning resources for the node in that container
  + Kube Proxy
    - Intrelligent forwarding logic inside
    - Communication works in a performant way with no overhead
* Nodes do the work

How do we interact with the cluster

* Schedule pod?
* Monitor pod?
* Reschedule restart pod?
* Join a new node?

Done with MASTER NODES

Master nodes

* Have 4 processes running inside
  + Control cluster state and worker nodes
  + 1. API server
    - Interact with API server to deploy
    - Cluster gateway
    - Gets initial request/queries etc
    - Acts as gatekeeper for authentication
    - Make sure only authorized requests fget through to the cluster
  + 2. Scheduler
    - If you send API to schedule a new pod
      * When validated it will pass it to the scheduler to start the pod on a worker node
      * Knows which worker node it will be scheduled on
        + Sees how much resources the app needs
        + Checks the worker nodes available resources etc
        + If one node has more resources, the pod will go there
        + Scheduler decides on where it will go
  + 3. Controller Manager
    - Detects state changes
      * Crashing of pods
      * Pod dies, tries to recover the cluster state ASAP
      * Makes request to scheduler
        + Scheduler does the resource calculation etc…
  + ETCD
    - Key value store of cluster state
    - Cluster brain
    - Every change in cluster gets saved or updated into ETCD
    - Scheduler and controller manager work because of the data in this
      * How does it know resources or cluster state changed etd
      * It is stored in ETCD

# Minikube and Kubectl

Production cluster setup

* Multiple masters
* Multiple worker nodes
* These will have separate repspnsibilities

Minikube

* 1 node cluster
* Master and worker processes run on one node
* Runs through vm
* This node will run in the VM
* Smmary: 1 node cluster running in VM
* Great for testing
* How to interact wit the cluster?
* Kubectl

Kubectl

* API Server is the main entry point to cluster
* If you want to do anything in the cluster you HAVE to talk to API server
  + Do this through different clients
  + You can do this through kubectl
* It is a client
* Can do anything in the cluster
* Submit commands to api server to create/delete components
* Worker processes on minkube node will do all of thise
* NOTE: not just for minikube cluster, can be used for hybrid or cloud cluster

**Minikube**

* Minikube start
  + Starts the cluster
* Minikube status
  + Gets the status
* **Minikube is for the startup and deletion, kubectl will run our program**

# Main kubectl commands

* Kubectl get nodes
  + Gets all the nodes
* Kubectl get pod
  + Gets the pods
* Kubectl get services
  + Gets services

Pod is smallest unit, you are not creating or working with pods, deployment is the abstraction layer above

* Deployment actually makes the pods
* To create a deployment
  + **Kubectl create deployment <name> --image=<imagename>**
  + When we create a deployment, it has a blueprint for creating a pod, most basic config for deployment
* kubectl get deployment
  + will get the deplotments
* kubectl get replicaset
  + will get the replicas
  + replicaset is managing the replicas of a pod
* kubectl edit deployment <name>
  + this will bring up the configuration YAML file
  + when you edit a pod, the pod will slowly depreciate and a new one will take its place
  + the new pod will exist at the same time as the previous as it dies off
* kubectl logs
  + gets the logs of the pod
* kubectl apply -f {filename}