**Need for K8s**

* High Availability (HA):
  + When we run our applications in docker container and if the container fails, we need to manually start the container
  + If the node i.e., the machine fails all the containers running on the machine should be re-created on other machine
  + K8s can do both of the above
* Autoscaling
  + Containers don’t scale on their own.
  + Scaling is of two types
    - Vertical Scaling and Horizontal Scaling
  + K8s can do both horizontal and vertical scaling of containers
* Zero-Down time Deployments
  + K8s can handle deployments with near zero-down time deployments
  + K8s can handle rollout (new version) and roll back (undo new version => previous version)
* K8s is described as Production grade Container management

**History**

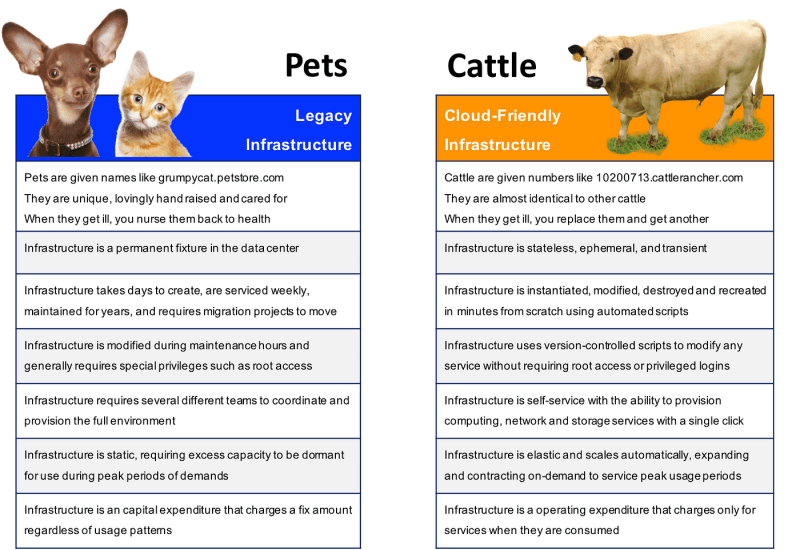
* Google had a history of running everything on containers.
* To manage these containers, Google has developed container management tools (inhouse)
  + Borg
  + Omega
* With Docker publicizing containers, With the experience in running and managing containers, Google has started a project Kubernetes (developed in Go) and then handed it over to Cloud Native Container Foundation (CNCF)

**Competitors**

* Apache Mesos
* HashiCorp Nomad
* Docker Swarm
* But K8s is clear winner

**Terms**

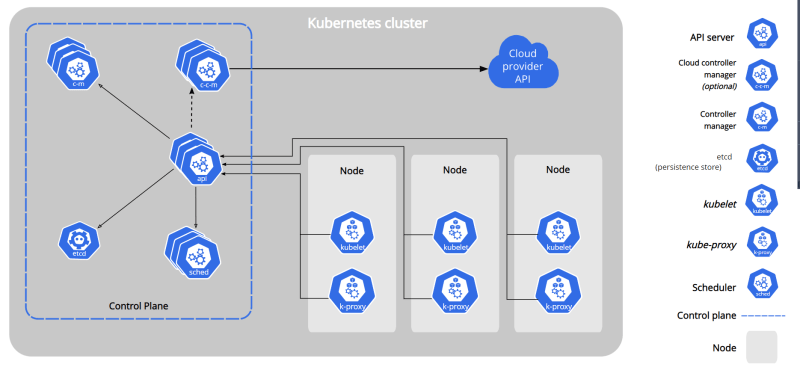
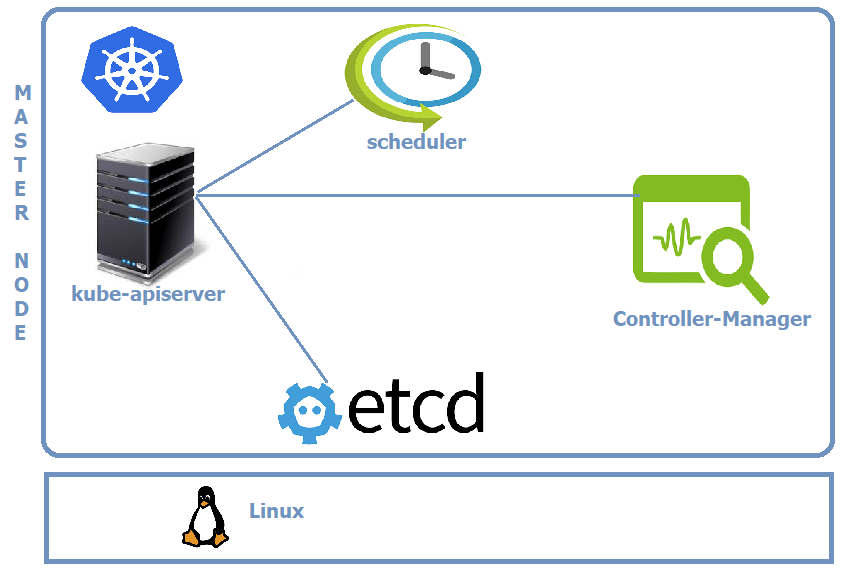
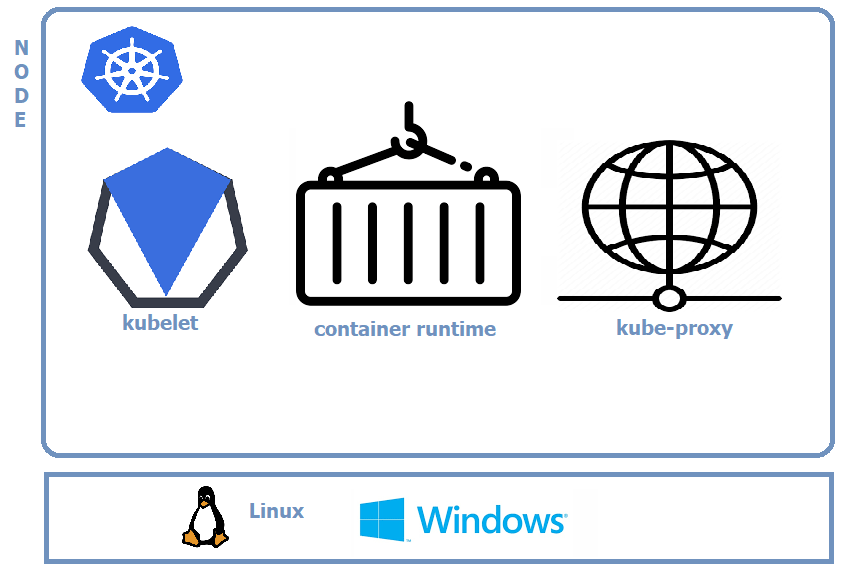
* Distributed System
* Node
* Cluster [Group of server work together for same purpose.]
* Desire State [Needful state]
* **Stateful Applications**: Application that store its data locally. [ELK]
* **Stateless Applications:** Application that need external storage.
* Monolith
* Microservices
* Desired State
* Declarative vs Imperative

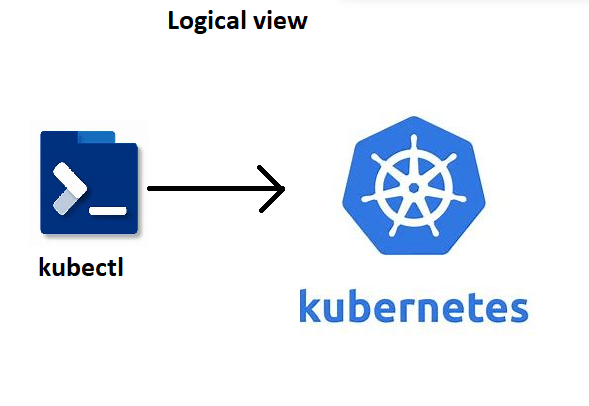
Pet Vs Cattle mindset  


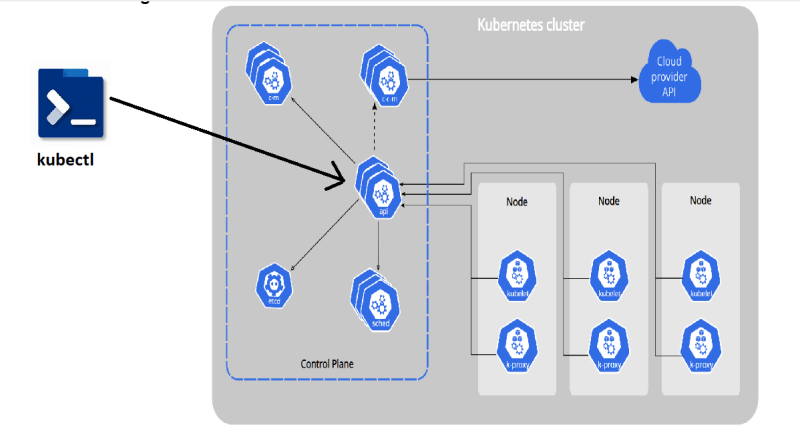
**K8s is not designed only for Docker**

* Initially k8s used docker as a main container platform and docker used to get special treatment, from k8s 1.24 special treatment is stopped.
* k8s is designed to run any container technology, for this k8s expects container technology to follow k8s interfaces.

**K8s Architecture**

* Official Architecture image  
  
* Other easier representations
* Master Node  
  
* Node  
  
* Clients
  + kubectl
  + any rest-based client

Logical view  


* Actual view  
  

**Kubernetes Components**

* <https://directdevops.blog/2019/10/10/kubernetes-master-and-node-components/> for the k8s components article
* Control plane components (Master Node Components)
  + kube-api server
  + etcd (\*)
  + kube-scheduler
  + controller manager
  + cloud controller manager
* Node Components
  + kubelet
  + kube-proxy
  + Container run time (\*)

**kube-api server**

* Handles all the communication of k8s cluster
* Let it be internal or external
* kube-api server exposes functionality over HTTP(s) protocol and provides REST API

**etcd**

* <https://etcd.io/> for etcd
* This is memory of k8s cluster

**scheduler**

* Scheduler is responsible for creating k8s objects and scheduling them on right node

**Controller**

* Controller Manger is responsible for maintaining desired state
* This reconcilation loop that checks for desired state and if it mis matches doing the necessary steps is done by controller

**Kubelet**

* This is an agent of the control plane

**Container Runtime**

* Container technology to be used in k8s cluster
* in our case it is docker.

**Kube-Proxy**

* This component is responsible for networking for containers on the node

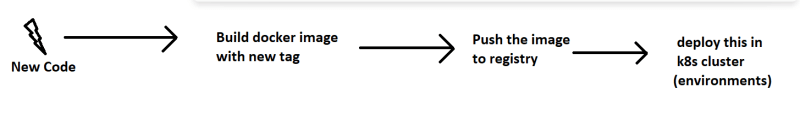
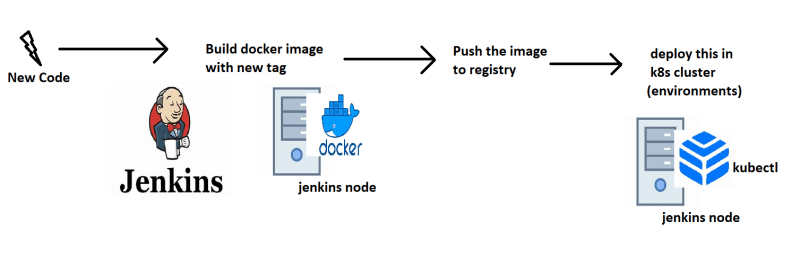
**kubectl**

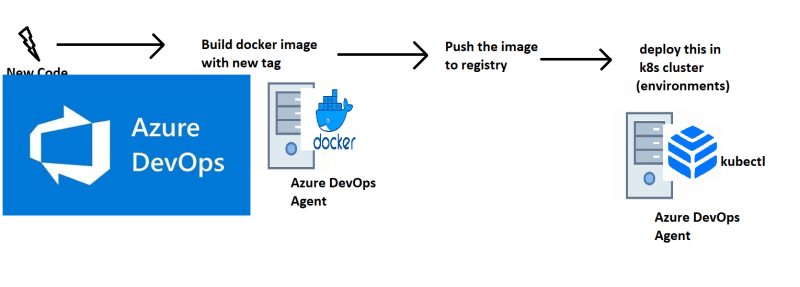
* This is command line that can be installed on the machine from which you communicate to k8s cluster.
* This tool is created to make communication with api-server simplified.
* Kubectl has a config file (KUBECONFIG) which contains
  + api-server information
  + keys to communicate with api server
* Kubectl allows to communication with cluster to create resources
  + imperatively: Type commands
  + declartively: Write manifests (YAML files)
* Reads manifests and connects to api server. Converts the manifest into REST API calls over JSON

**What is k8s manifest**

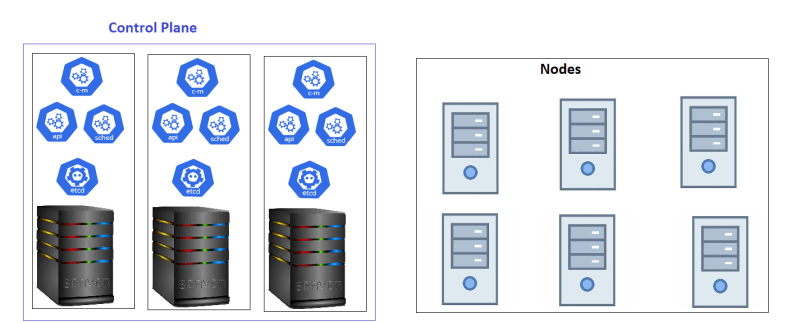
* This is a yaml file which describes the desired state of what you want in/using k8s cluster

**CI/CD Workflow**

* Basic Workflow  
  
* Jenkins  
  

Azure DevOps  


**IDEAL K8s HA-Cluster**



**Kubernetes as a Service**

* All popular clouds are offering k8s as a service
  + AKS (Azure K8s service)
  + EKS (Elastic K8s Service)
  + GKE (Google K8s engine)
* All cloud providers manage control plane for you and they charge hourly. For nodes we pay the similar costs of virtual machines

**K8s Installations**

* Single Node Installations
  + minikube
  + kind
* On-prem installations
  + kube-admin
* k8s as a Service
  + AKS
  + EKS
  + GKE
* Playground (for learning): <https://labs.play-with-k8s.com/>

**Installing k8s cluster on ubuntu vms**

* Create 3 ubuntu vms which are accesible to each other with atlest 2 vCPUS and 4 GB RAM
* Installation method (kubeadm) which is something we will be using in on-premises k8s.
* <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/> for kubeadm installation on single master node

**Steps**

* Install docker on all nodes
* Install CRI-Dockerd <https://github.com/Mirantis/cri-dockerd>
* Run the below commands as root user in all the nodes

# Run these commands as root

###Install GO###

wget https://storage.googleapis.com/golang/getgo/installer\_linux

chmod +x ./installer\_linux

./installer\_linux

source ~/.bash\_profile

git clone https://github.com/Mirantis/cri-dockerd.git

cd cri-dockerd

mkdir bin

go build -o bin/cri-dockerd

mkdir -p /usr/local/bin

install -o root -g root -m 0755 bin/cri-dockerd /usr/local/bin/cri-dockerd

cp -a packaging/systemd/\* /etc/systemd/system

sed -i -e 's,/usr/bin/cri-dockerd,/usr/local/bin/cri-dockerd,' /etc/systemd/system/cri-docker.service

systemctl daemon-reload

systemctl enable cri-docker.service

systemctl enable --now cri-docker.socket

* Installing kubadm, kubectl, kubelet <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/#installing-kubeadm-kubelet-and-kubectl>
* Now create a cluster from a master node <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/>
* use the command kubeadm init --pod-network-cidr "10.244.0.0/16" --cri-socket "unix:///var/run/cri-dockerd.sock"
* Setup kubeconfig
* install flannel kubectl apply -f <https://github.com/flannel-io/flannel/releases/latest/download/kube-flannel.yml>
* AS a root user run kubeadm join commands (need to pass crisocket)
* Now from manager execute kubectl get nodes

