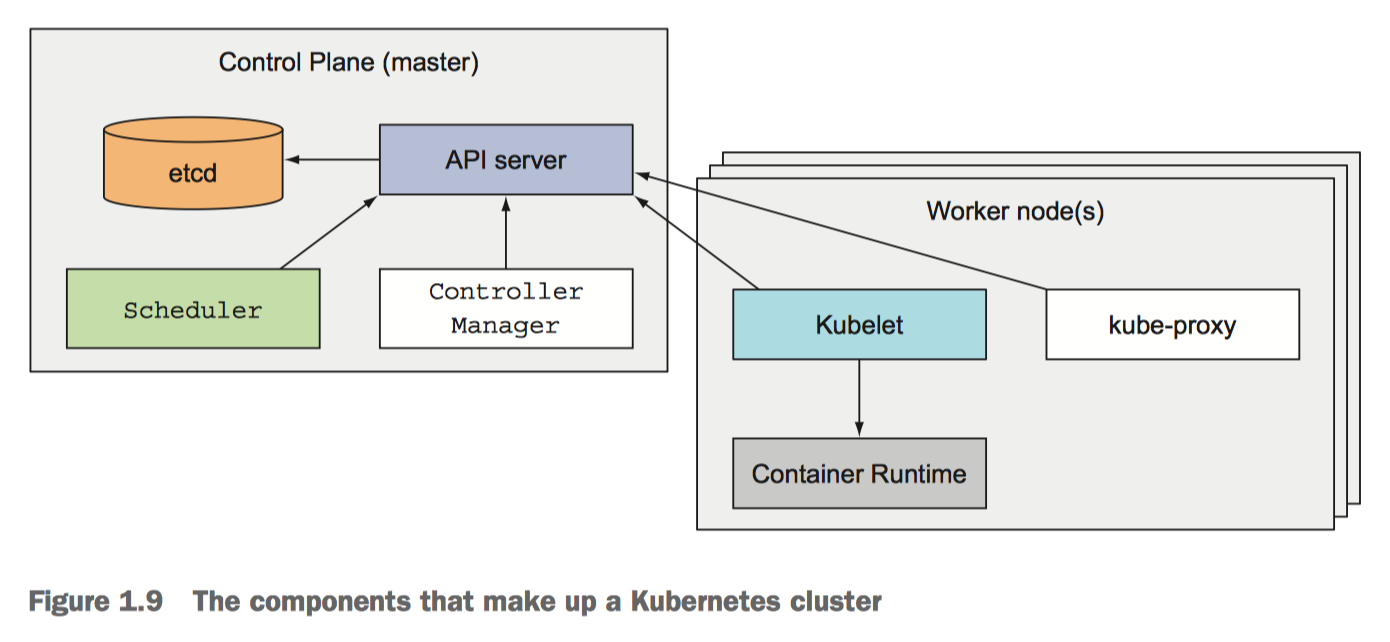
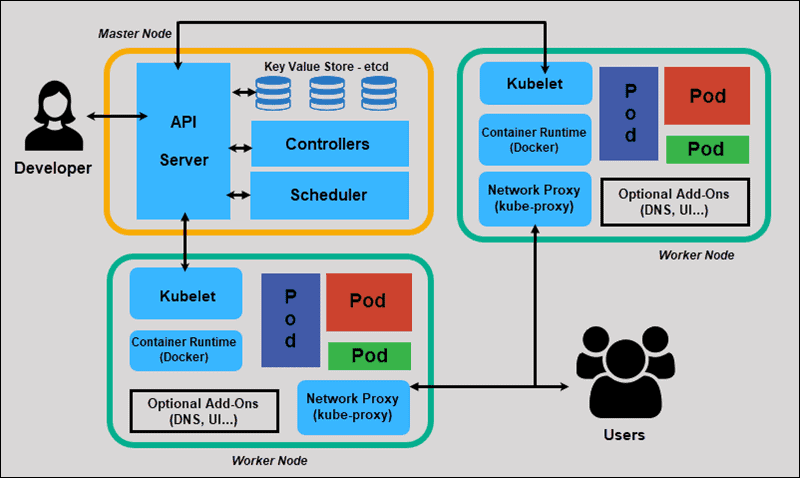
Kubernetes :- CKA ( Certified Kubernetes Associates ) – 01/03/2021

<https://carltsuis-blog.readthedocs.io/en/latest/kubernetes/Architecture%20of%20a%20Kubernetes%20cluster/>

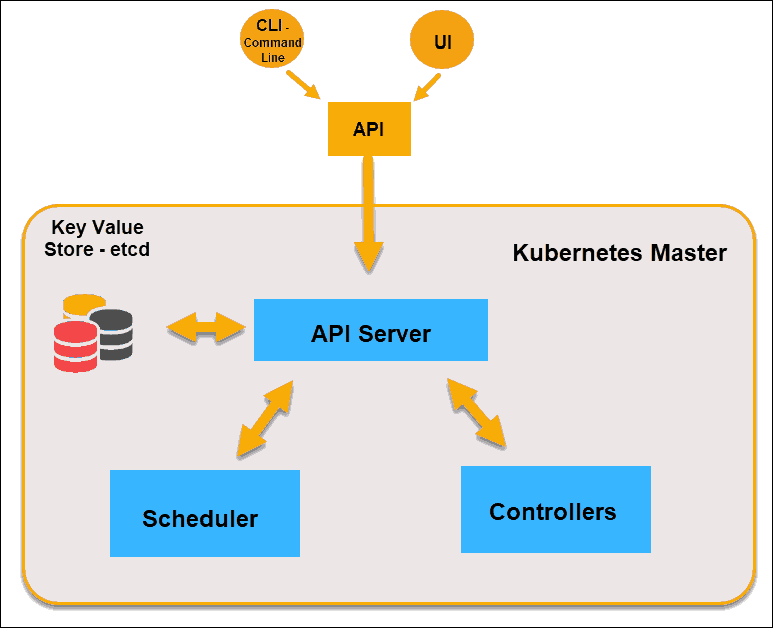
Architecture :-



<https://phoenixnap.com/kb/understanding-kubernetes-architecture-diagrams>



**Kubernetes Master Node:-**

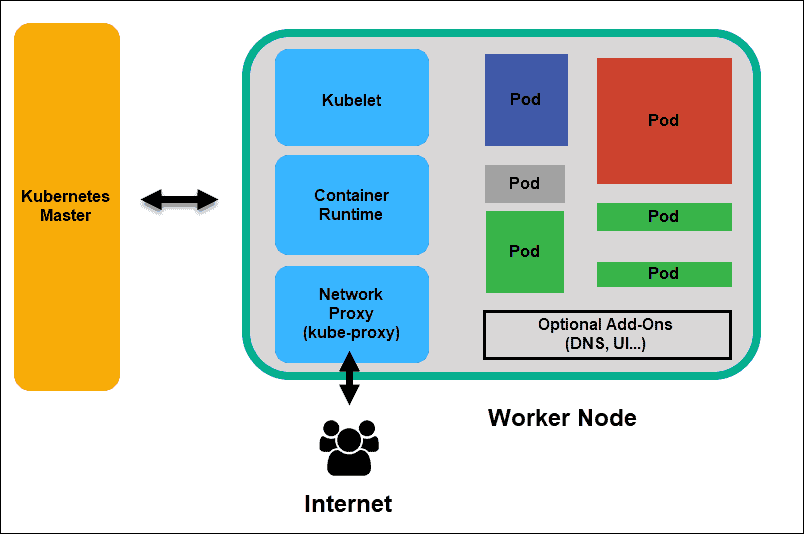


* Hosts the Kubernetes control plane that controls and manages the whole Kubernetes systems
* Receives input from CLI/UI via an API-server
* User defines – Replica, Pods and services

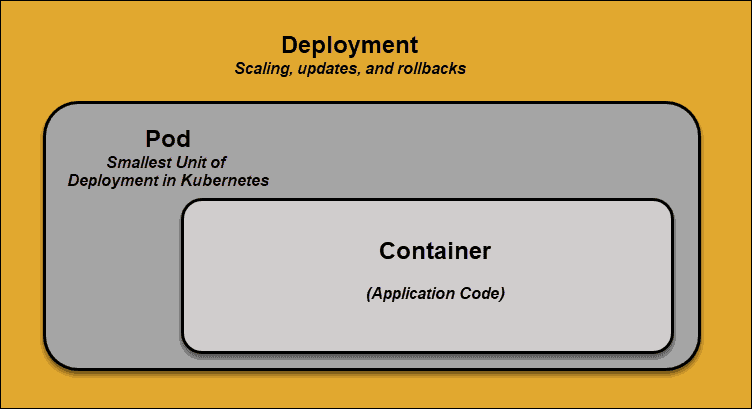
Control Planes :-

* They don’t run any applications
* Controls its cluster and makes it function
* It runs on single master node or multiple to support high availability
* It contains :-
  + **API-Server**: - A user/front-end or other control plane components communicates with
    - Authenticate user
    - Validate request
    - Retrieve Data
    - Update ETCD
    - Scheduler
    - Communicate with kubelet
    - kubectl get pods -n <pod name >
    - ps -ef | grep kube-apiserver – Shows the process
    - /etc/systemd/system/kube-api-server.service – Non – kubeadmin setup
  + **Scheduler** – Schedules the apps. Watches the new request from the API server. It ranks the quality of nodes and deploys pods to the best suited node ( Which pod goes to which node )
    - It goes in two phases – Filter Nodes ( Discard the nodes don’t fit the purpose ) Rank Nodes ( Assign a score as per node and choose the best one )
    - Labels & Selectors
    - Daemon Sets
    - Resource Limits
    - Manual Scheduling
    - Multiple Schedulers
    - Scheduler Events
    - Configure Kubernetes Scheduler
    - Download and Install – from Kubernetes-releases
  + **Control Manager/Controller**: - Used for replicating components, watching worker nodes and handling node failures
    - It monitors the nodes though kube-api-server
    - Node monitor period – 5sec
    - Waits for 40 secs before marking it as dead ( Grace Period )
    - POD eviction time 5 mins
    - All these are packaged in Kube-Controller-Manager
    - Download and install from Kubernetes release
    - kubectl get pods -n <pod name >
    - ps -ef | grep kube-controller-manager – To see the process status
  + **Etcd (Key Value Store)**: -
    - A data storage (database) which stores the cluster configurations (Nodes, Pods, Configs, Secrets, Accounts, Roles, Bindings, etc.)
    - Master node queries etcd to retrieve parameter for the state of nodes, pods and containers
    - Download, Install and Run (./etcd)
    - Listens on port – 2379
    - Default client comes with ETCD is - etcdctl ( etcd control clients ) **[./etcdctl set key1 value1**]
    - To retrieve data run - ./etcdctl get key1
    - Run etcdctl – To get more help options
    - Etcdctl is the CLI tool to interact with ETCD – It has two API versions – V2 (Default) and V3
    - V2 supports following commands
      * etcdctl backup
      * etcdctl cluster-health
      * etcdctl mk
      * etcdctl mkdir
      * etcdctl set
    - V3 supports following commands
      * etcdctl snapshot save
      * etcdctl endpoint health
      * etcdctl get
      * etcdctl put
    - To set the right version of API - export ETCDCTL\_API=3
    - ETCD certificates are placed here :
      * --cacert /etc/kubernetes/pki/etcd/ca.crt
      * --cert /etc/kubernetes/pki/etcd/server.crt
      * --key /etc/kubernetes/pki/etcd/server.key
    - User can set-up this by running this :-
      * kubectl exec etcd-master -n kube-system -- sh -c "ETCDCTL\_API=3 etcdctl get / --prefix --keys-only --limit=10 --cacert /etc/kubernetes/pki/etcd/ca.crt --cert /etc/kubernetes/pki/etcd/server.crt --key /etc/kubernetes/pki/etcd/server.key"

**Worker Nodes/Nodes :-**



* + Listens to API server for work assignments. Executes the work and report the result back to Master Node
  + Runs the actual applications user deploys
  + **Kubelets :- ( Captain of the ship )** 
    - Runs on each worker node of the cluster
    - Registers the Worker Node to Master Node
    - It is the principal Kubernetes agent
    - Talks to API server and manages containers on its node
    - Also monitors pods health and reports to control plane if not functional
    - Download, Install as a service
    - Kubeadm doesn’t deploy kubelets
    - ps -ef | grep kubelet
  + **Container Runtime:-**
    - Pulls images from a container image registry
    - Starts and Stops Container
    - A 3rd party software ( Docker ) does this function
  + **Kube-proxy :-** 
    - A Kubernetes service – runs on each worker node
    - Load-balances network traffic between application components
    - Assign ip-address – on each node
    - Assign iptables – on each node
    - Assign Rules – on each node
    - Download, install and run as a service - wget <https://storage.googleapis.com/kubernetes-release/release/v1.13.0/bin/linux/amd64/kube-proxy>
    - kubectl get pods – n <pod name>
    - kubectl get daemonset -n <pod name>
  + **POD :-**



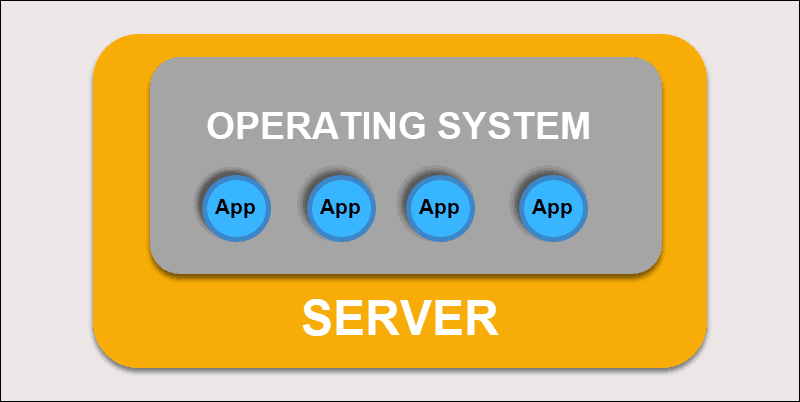
* + - The smallest element of scheduling in Kubernetes
    - A container can’t be part of cluster without a POD
    - If scaling app needed, can be only done by adding and removing pods
    - If any pod fails, Kubernetes doesn’t fix it, it creates a new pod

Kubernetes Services :-

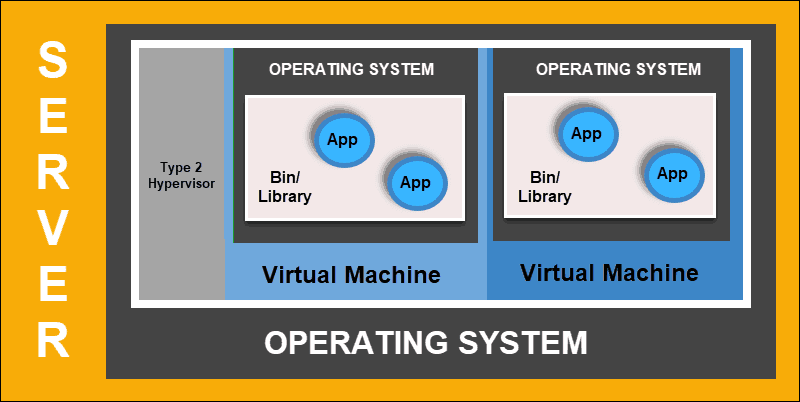
* If a POD fails, Kubernetes creates a new POD instead of fixing this. The new pod will have a new ip address which is unreliable.
* Services are introduced to provide reliable networking by bringing stable IP addresses, DNS and ports
* By this user can add/remove any pods
* How does this service work ?-
  + Each pod associated with labels and selectors
  + A new pod automatically discovers labels that matches with selector
  + This process seamlessly adds new pods and removes terminated pods from cluster
  + The same functionalities apply for replicating/scaling pods into cluster

Container Deployments (Kubernetes Orchestrates ) :-

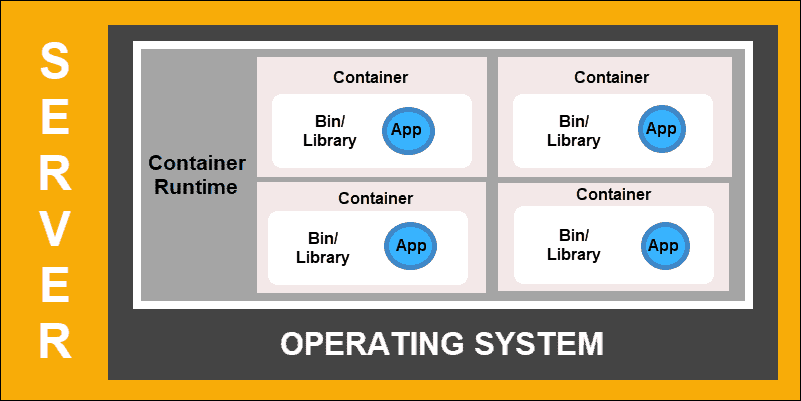
* Traditional Deployments



* + Deploying applications on individual physical servers
  + Problem – Applications can take up most of the processing power, limiting the performance of applications on same machine
  + Takes long time to extend hardware capacity which turns increase in cost
* Virtualized Deployments



* + This deployment allows user to create isolated VM on a single physical server
  + Isolates applications within VM, limits the use of resources and increase security
  + An application can no longer freely access the information processed by another application
  + It allows to scale quickly and spread resources of a single physical server, update at will
  + It keeps hardware cost at a check
* Container Deployments



* + Most flexible and efficient model
  + Container has its own – memory, system files and processing space
  + Multiple applications can share same operating system
  + These are portable across clouds, different device and almost any OS distribution
  + It allows to run applications a smaller and independent parts which can be deployed dynamically on multiple machines
* Creating a POD :-
  + Through CLI :-
    - kubectl run <pod name> --image <image Name>
    - kubectl run <pod name> --image=<image Name> - > 1.18 + Versions
    - kubectl run nginx –image nginx
  + Using yaml file :-
    - cat pod-defination.yaml

#Deploying a pod

apiVersion: v1

kind: Pod

metadata:

name: nginx-test

labels:

app: myapp

type: front-end

spec:

containers: #- > It creates two containers

- name: nginx-container

image: nginx

resources:

limits:

memory: “128Mi”

cpu: “500m”

- name: backend-container

image: redis

resources:

limits:

memory: “128Mi”

cpu: “500m”

* Creating ReplicaSet