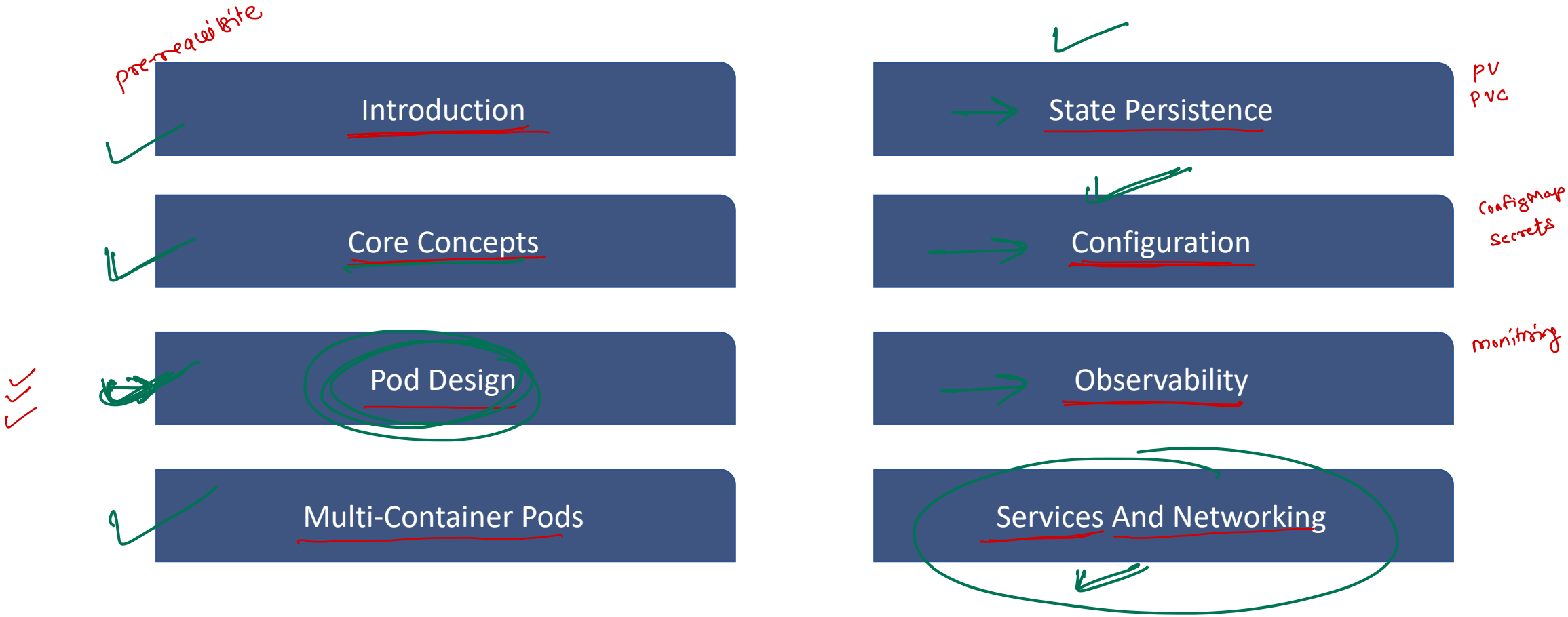
kubernetes

CKAD


Introduction
Core Concepts
Pod Design
Multi-Container Pods
State Persistence
Configuration
Observability
Services And Networking

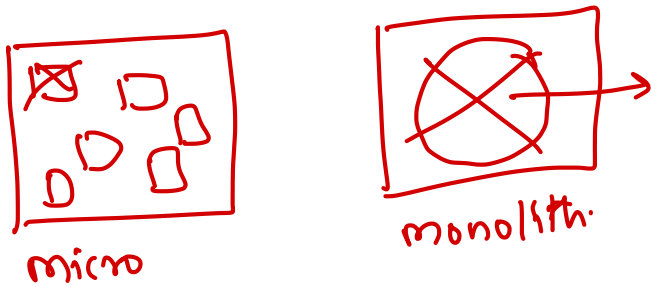
CKA

- Cluster Installation & Configuration
- Storage
- Cluster Administration
- Troubleshooting
- Workloads and Scheduling
- Services & Networking

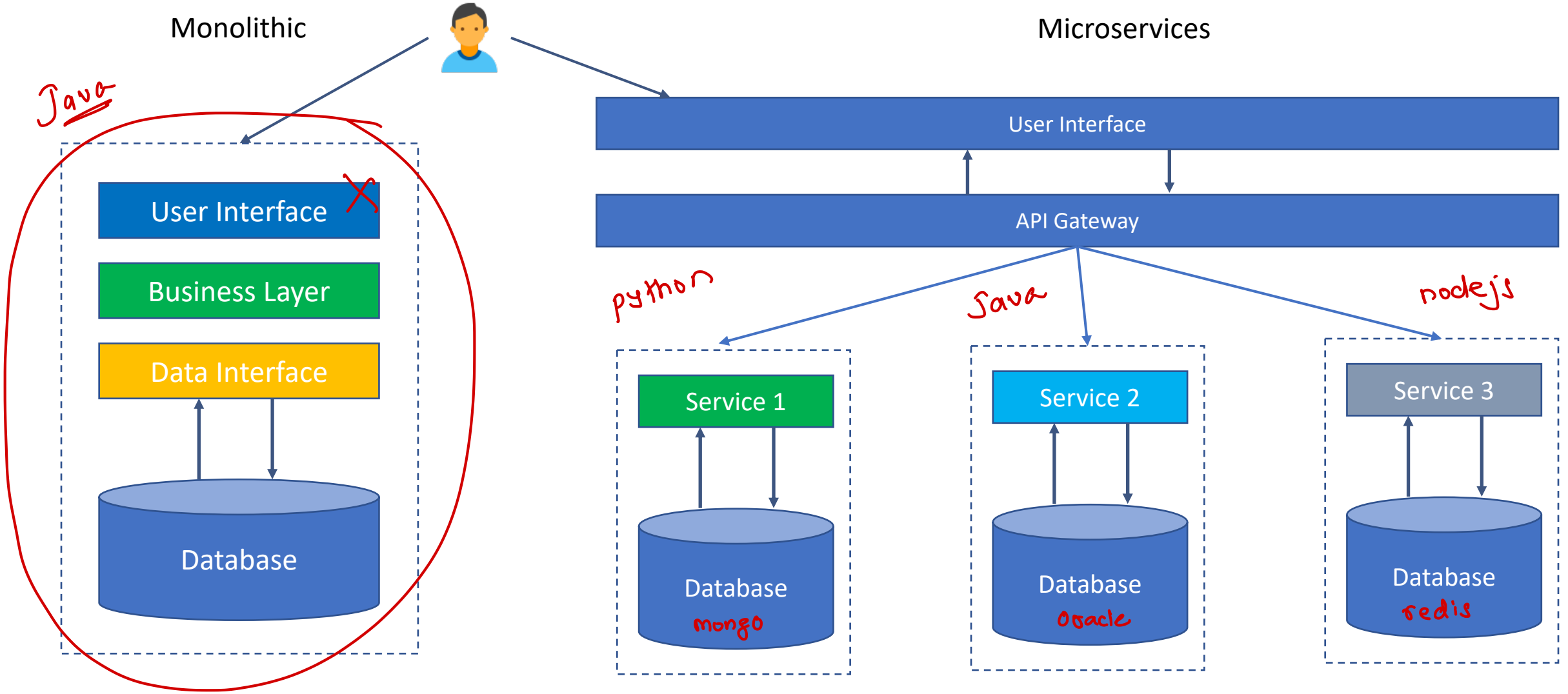
Pre-requisites

Microservice

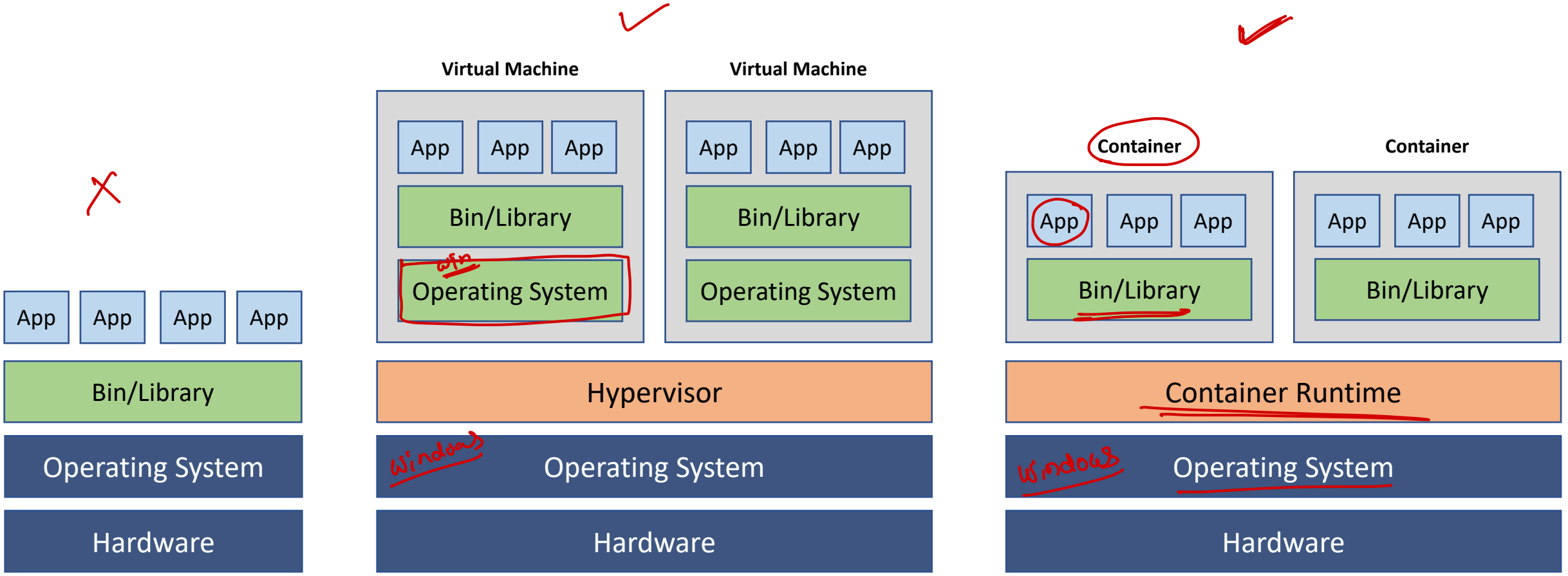
- Distinctive method of developing software systems that tries to focus on building single-function modules with well-defined interfaces and operations
- Is an architectural style that structures an application as a collection of services that are
  - Highly maintainable and testable
  - Loosely coupled
  - Independently deployable
  - Organized around business capabilities



Monolithic vs Microservice



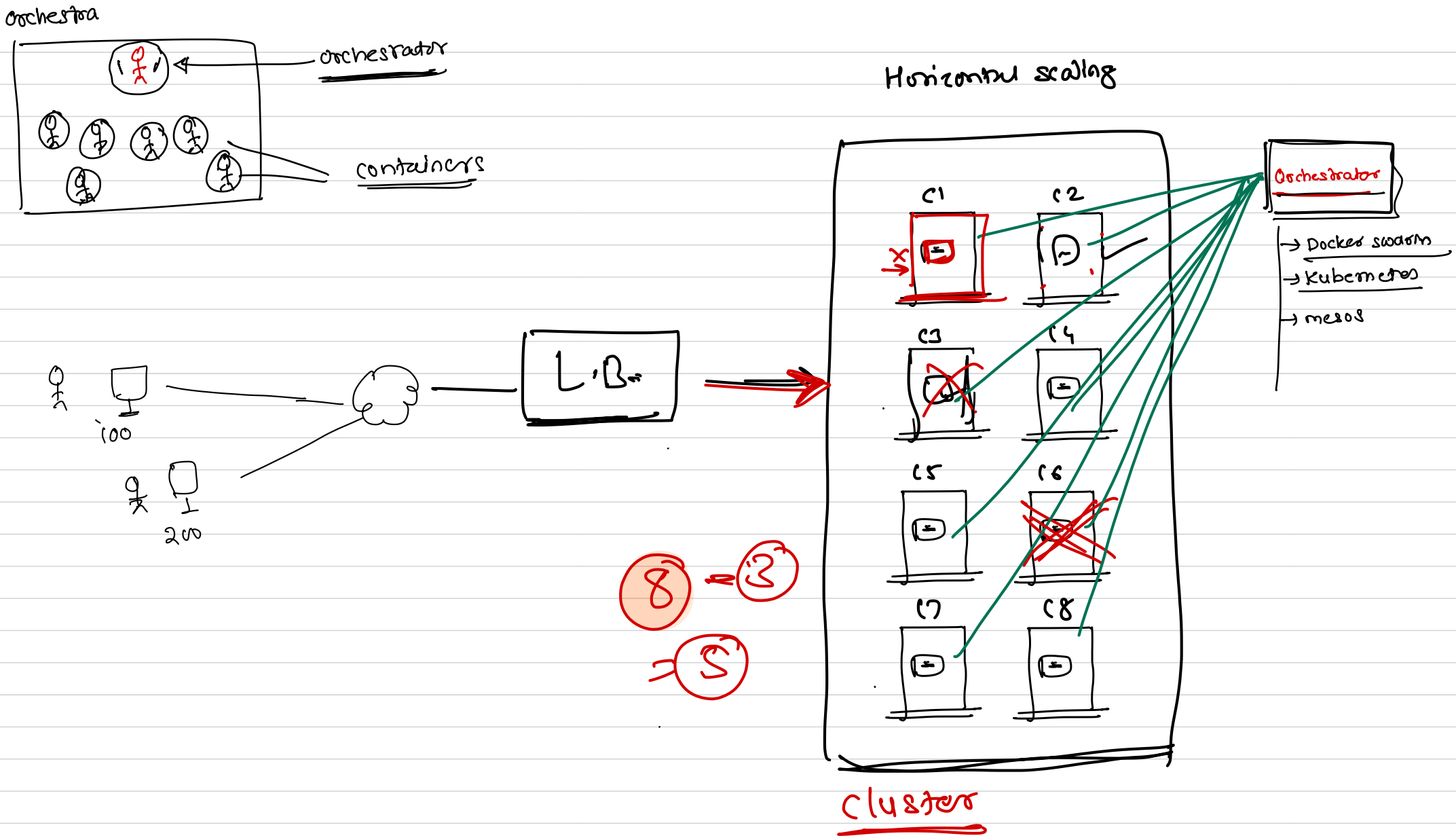
Deployment options

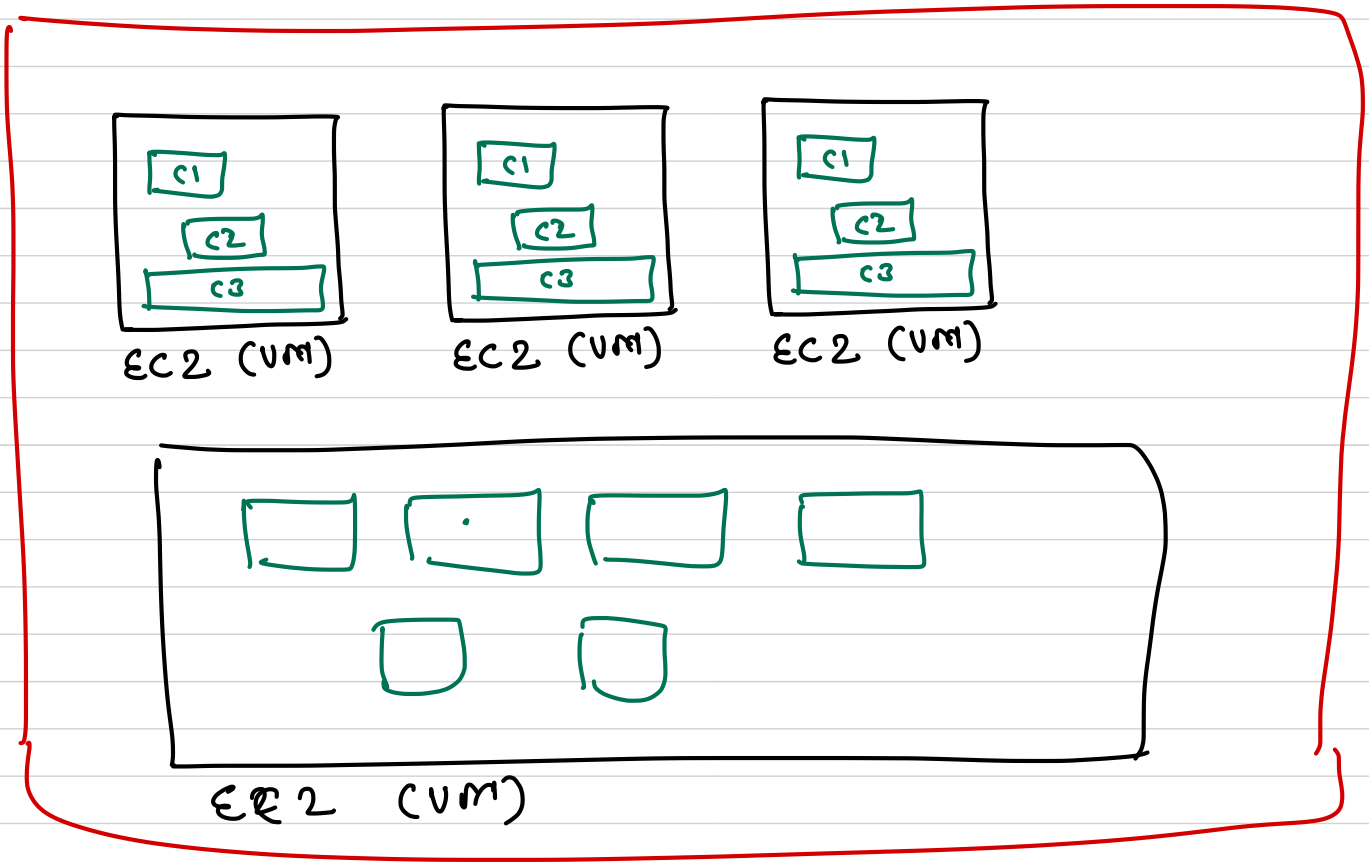


Traditional Deployment

Virtual Machine Deployment

Containerized Deployment


orchestra
orchestrator
containers
Horizontal scaling
L.B.
100
200
C1
C2
C3
C4
C5
C6
C7
C8
Orchestrator
→ Docker swarm
→ Kubernetes
→ mesos
cluster


C1
C2
C3
EC2 (VM)
C1
C2
C3
EC2 (VM)
C1
C2
C3
EC2 (VM)
EC2 (VM)

Container orchestration

- Container orchestration is all about managing the lifecycles of containers
- Software teams use container orchestration to control and automate many tasks
  - Provisioning and deployment of containers
  - Redundancy and availability of containers
  - Scaling up or removing containers to spread application load evenly across host infrastructure
  - Allocation of resources between containers
  - External exposure of services running in a container with the outside world
  - Load balancing of service discovery between containers
  - Health monitoring of containers and hosts
  - Configuration of an application in relation to the containers running it
  - Movement of containers from one host to another if there is a shortage of resources, or if a host dies
- Tools
  - Docker Swarm
  - Kubernetes
  - Apache Mesos and Marathon

Introduction

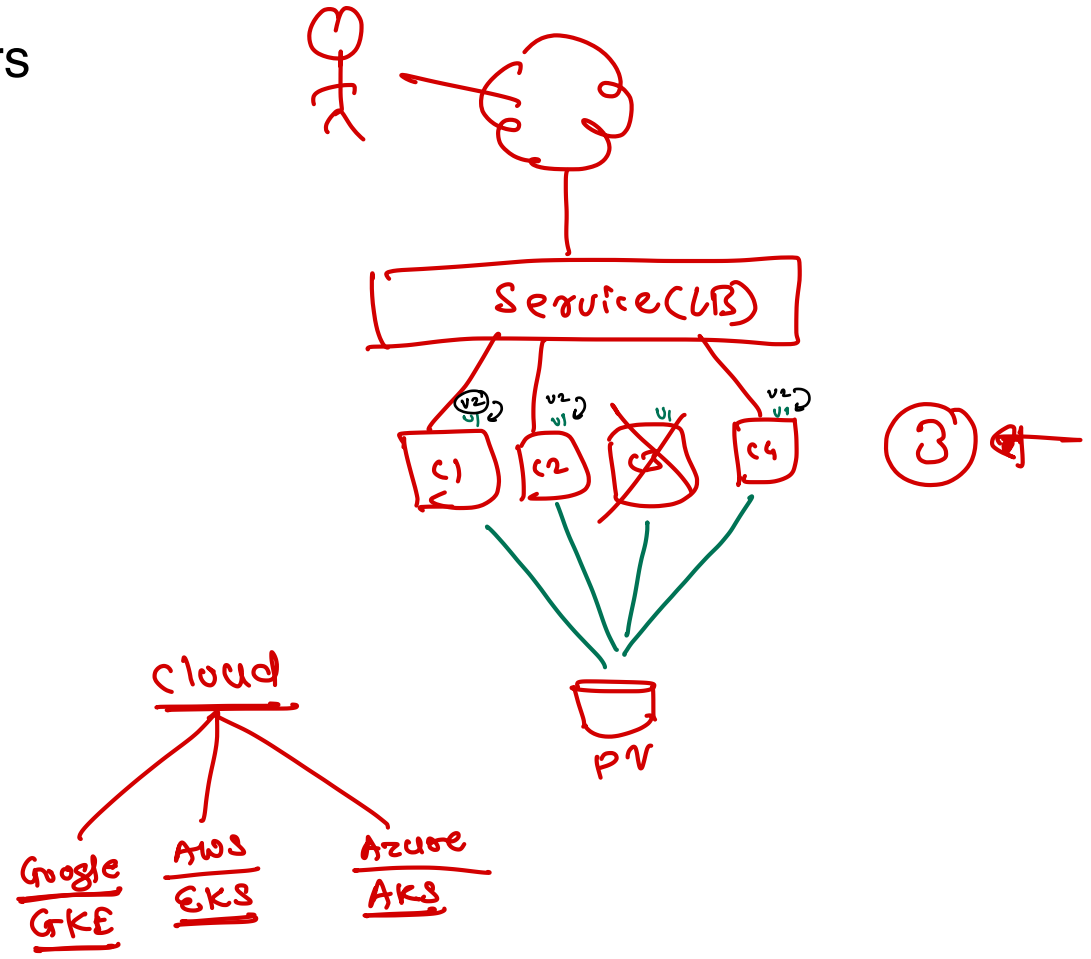
- Kubernetes is an open source system for automating deployments, scaling and management of containerized applications
- It is a container orchestration system
- It enables organizations to automate deployment and manage the containers, thus helping them to streamline and simply the day to day routines
- If you are using Kubernetes that means your application is following the microservices architecture and is already containerized

History

- Kubernetes was originally developed at Google having emerged from project Borg
- Now it has been taken over by Cloud Native Computing Foundation (CNCF)
- It is not only open source but it is managed by open community

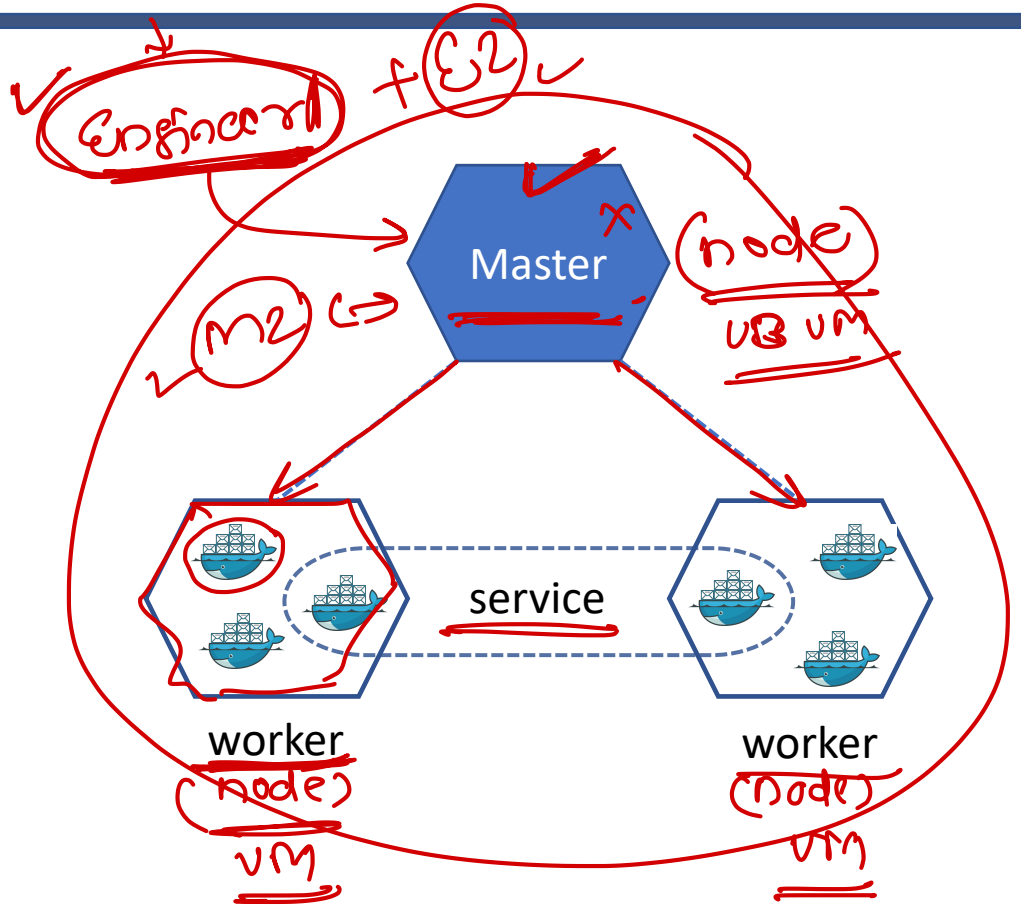
Kubernetes Features

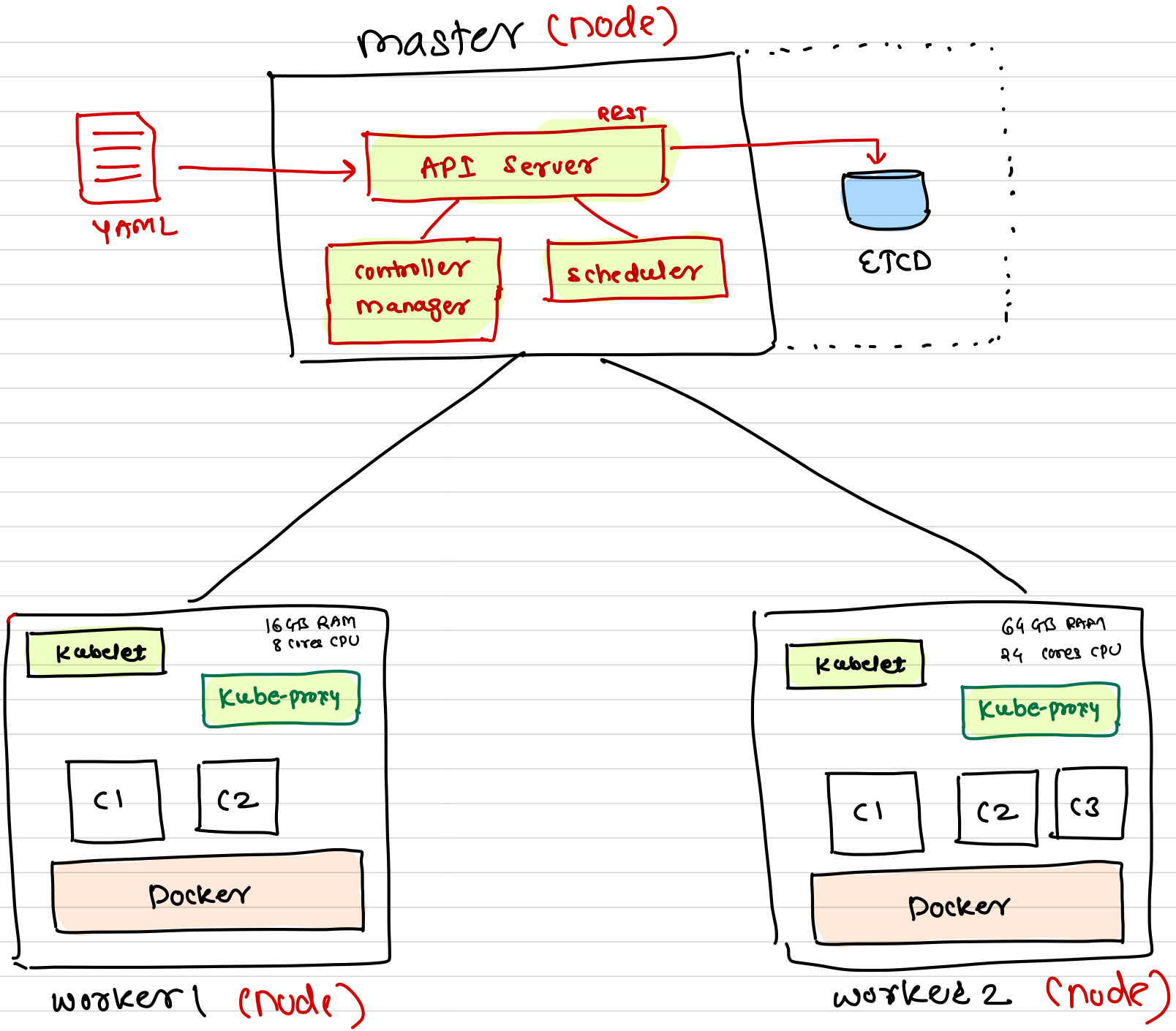
- Provisioning and deployment of containers
- Self healing
- Service discovery and load balancing
- Storage orchestration
- Auto scaling
- Run anywhere
- Automated rollouts and rollbacks
- Secrets and configuration management
- Scale
- REST APIs at its core
- Security



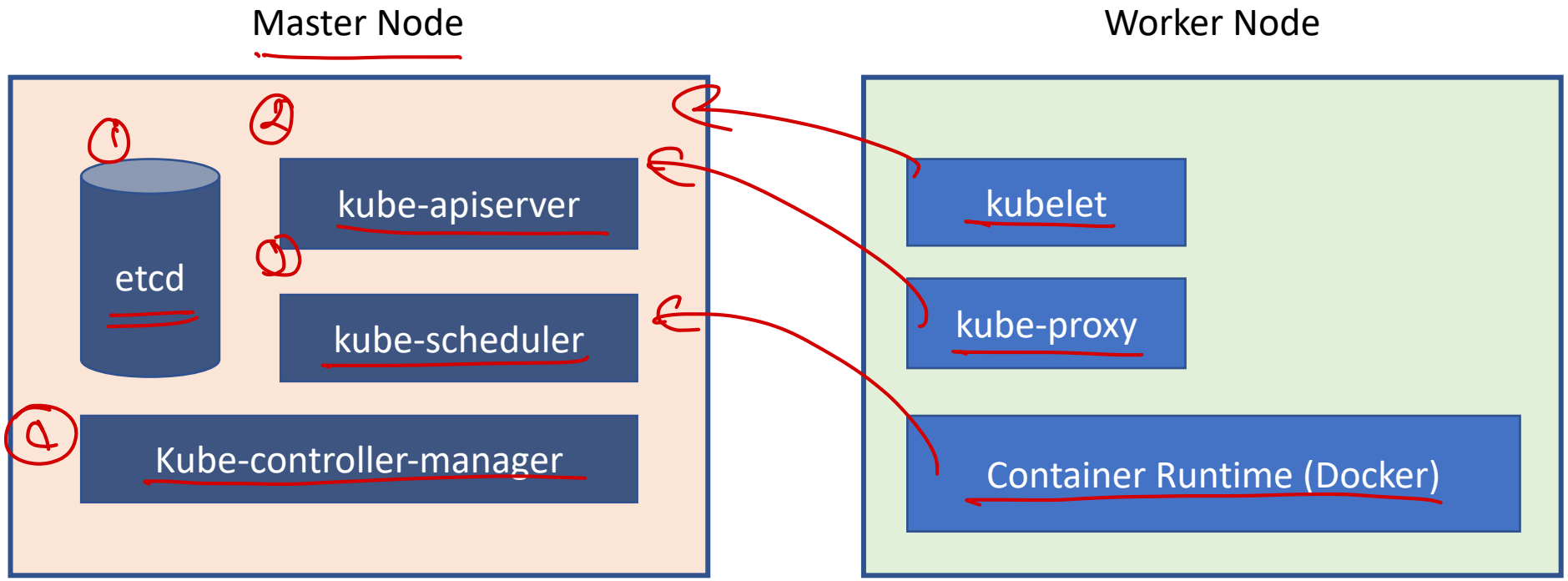
Kubernetes Cluster

- When you deploy Kubernetes, you get a cluster.
- A cluster is a set of machines (nodes), that run containerized applications managed by Kubernetes
- A cluster has at least one worker node and at least one master node
- The worker node(s) host the pods that are the components of the application
- The master node(s) manages the worker nodes and the pods in the cluster
- Multiple master nodes are used to provide a cluster with failover and high availability




master (node)
REST
API Server
YAML
controller manager
scheduler
ETCD
Kubelet
16 GB RAM
8 cores CPU
Kube-proxy
C1
C2
Docker
worker1 (node)
Kubelet
64 GB RAM
24 cores CPU
Kube-proxy
C1
C2
C3
Docker
worker2 (node)

Kubernetes Components



Master Components

- Master components make global decisions about the and they detect and respond to cluster events
- Master components can be run on any machine in the cluster
- **kube-apiserver**
  - The API server is a component that exposes the Kubernetes REST API
  - The API server is the front end for the Kubernetes
- **etcd**
  - Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data
  - The configuration data the etcd stores represents the cluster state
  - e.g. which nodes are running the containers, which applications are running etc
- **kube-scheduler**
  - Component on the master that watches newly created pods that have no node assigned, and selects a node for them to run on

Master Components

- **kube-controller-manager**
  - Component on the master that runs controllers
  - Logically, each controller is a separate process, but to reduce complexity, they are all compiled into a single binary and run in a single process
  - Types
    - **Node Controller**: Responsible for noticing and responding when nodes go down.
    - **Replication Controller**: Responsible for maintaining the correct number of pods for every replication controller object in the system
    - **Endpoints Controller**: Populates the Endpoints object (that is, joins Services & Pods)
    - **Service Account & Token Controllers**: Create default accounts and API access tokens for new namespaces
- **cloud-controller-manager**
  - Runs controllers that interact with the underlying cloud providers
  - The cloud-controller-manager binary is an alpha feature introduced in Kubernetes release 1.6

Node Components

- Node components run on every node, maintaining running pods and providing the Kubernetes runtime environment
- **kubelet**
  - An agent that runs on each node in the cluster
  - It makes sure that containers are running in a pod
- **kube-proxy**
  - Network proxy that runs on each node in your cluster, implementing part of the Kubernetes service concept
  - kube-proxy maintains network rules on nodes
  - These network rules allow network communication to your Pods from network sessions inside or outside of your cluster
- **Container Runtime**
  - The container runtime is the software that is responsible for running containers
  - Kubernetes supports several container runtimes: Docker, containerd, rktlet, cri-o etc.

Designing a cluster

- Application components and the number of replicas required to run the application smoothly
- Kubernetes v1.16 has following restrictions
  - 5000 nodes
  - 150000 pods
  - 300000 containers
  - 100 pods per node
- Resource requirements of the pods
  - Processor, Memory and HDD requirements for the pods
  - Application Data Storage
- Co-located or external etcd
- Cluster monitoring
- Infrastructure: on-premises, bare metal or managed cloud cluster
- Security

Designing a cluster

- Worker nodes requirements
- Number of workers required to run the pods
- Configuration of worker nodes
- https://kubernetes.io/docs/setup/best-practices/cluster-large/

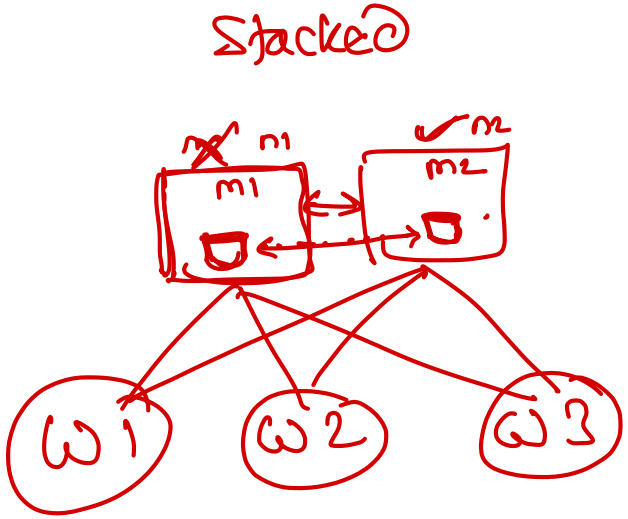
Nodes	Configuration
1-5 nodes	2 vCPU / 4 GB RAM
6-10 nodes	2 vCPU / 7.5 GB RAM
11-100 nodes	4 vCPU / 15 GB RAM
101-250 nodes	8 vCPU / 30 GB RAM
251-500 nodes	16 vCPU / 30 GB RAM
More than nodes	36 vCPU / 60 GB RAM

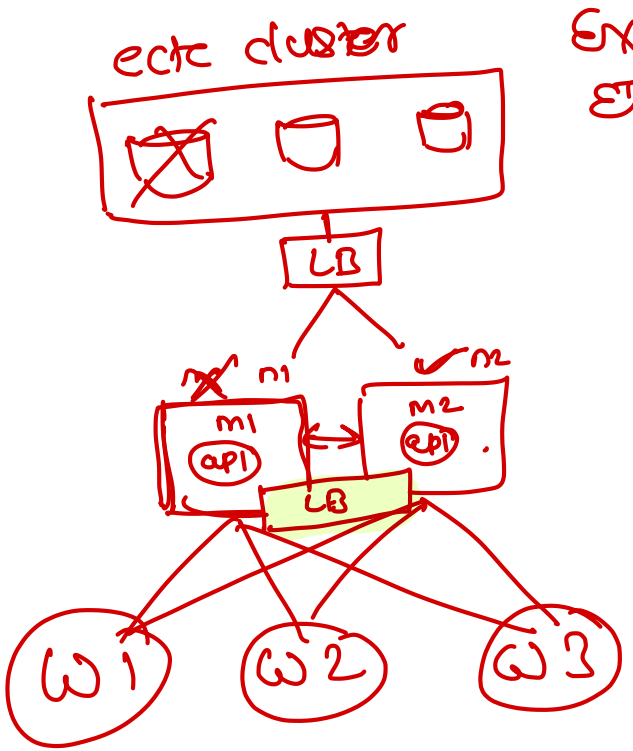
Cluster node requirements

- Cluster can be installed on physical or virtual machine
- Following operating systems are supported with v1.16
  - Ubuntu 16.04+
  - Debian 9+
  - Centos 7+
  - RHEL 7+
  - Fedora 25+
  - Container Linux
  - HypriotOS 1.0.1+
- Minimum configuration on each node is 2CPU and 2 GB RAM
- Control plane ports needed to open
  - 6443, 2379-2380, 10250, 10251, 10252
- Worker nodes ports needed to open
  - 10250, 30000-32767

High Availability Cluster

- No single point of failure
- Kube-apiserver is exposed to worker nodes using a load balancer
- Stacked etcd or External etcd





Create Cluster

- Use following commands on both master and worker nodes

> sudo apt-get update && sudo apt-get install -y apt-transport-https curl

> curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg I sudo apt-key add -

> cat <<EOF I sudo tee /etc/apt/sources.list.d/kubernetes.list deb https://apt.kubernetes.io/kubernetes-xenial main EOF

> sudo apt-get update
> sudo apt-get install -y kubelet kubeadm kubectl
> sudo apt-mark hold kubelet kubeadm kubectl

Initialize Cluster Master Node

- Execute following commands on master node

> kubeadm init --apiserver-advertise-address=<ip-address> --pod-network-cidr=10.244.0.0/16

> mkdir -p $HOME/.kube

> sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

> sudo chown $(id -u):$(id -g) $HOME/.kube/config

- Install pod network add-on

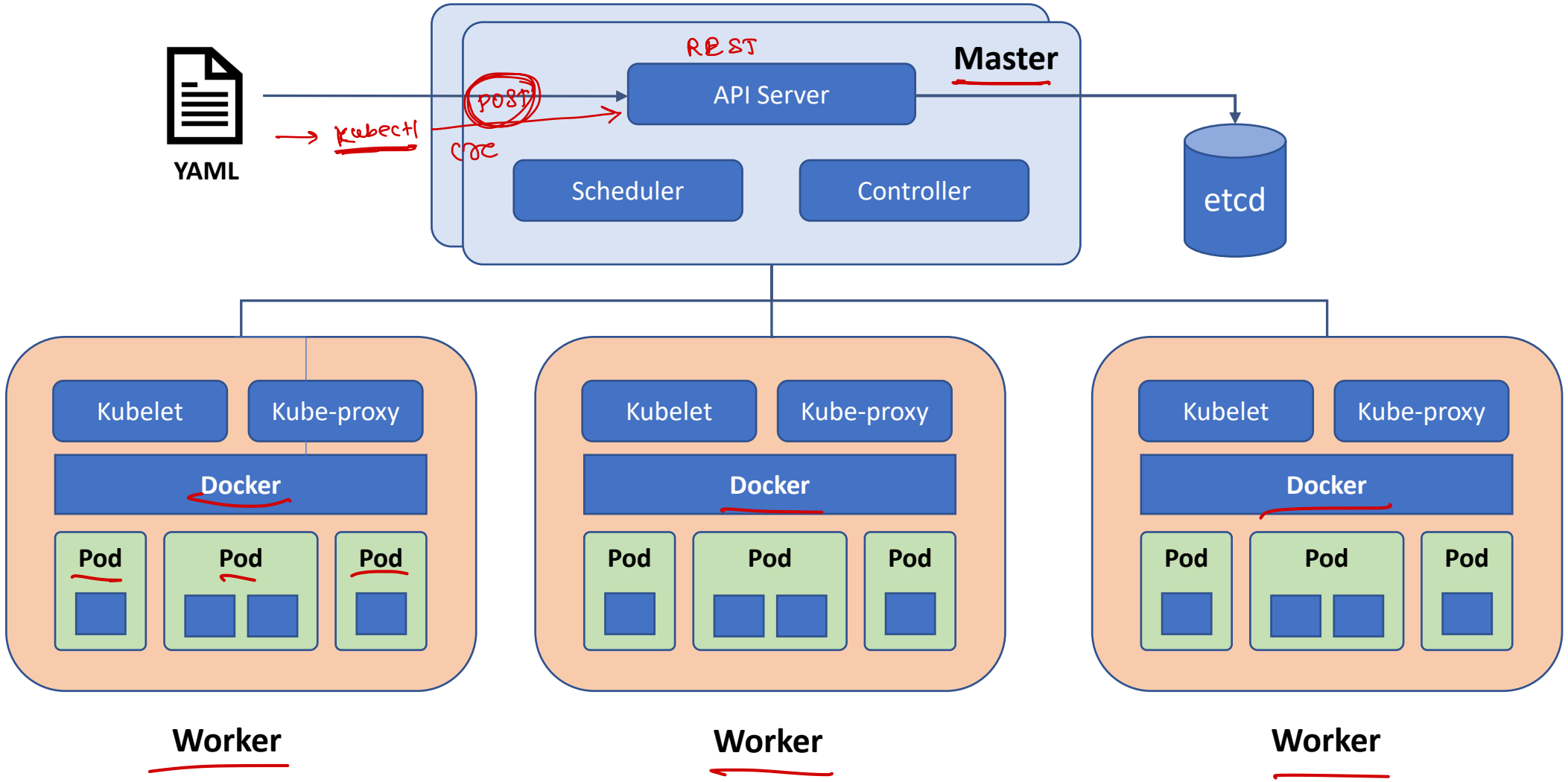
> kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/2140ac876ef134e0ed5af15c65e414cf26827915/Documentation/kube-flannel.yml

Add worker nodes

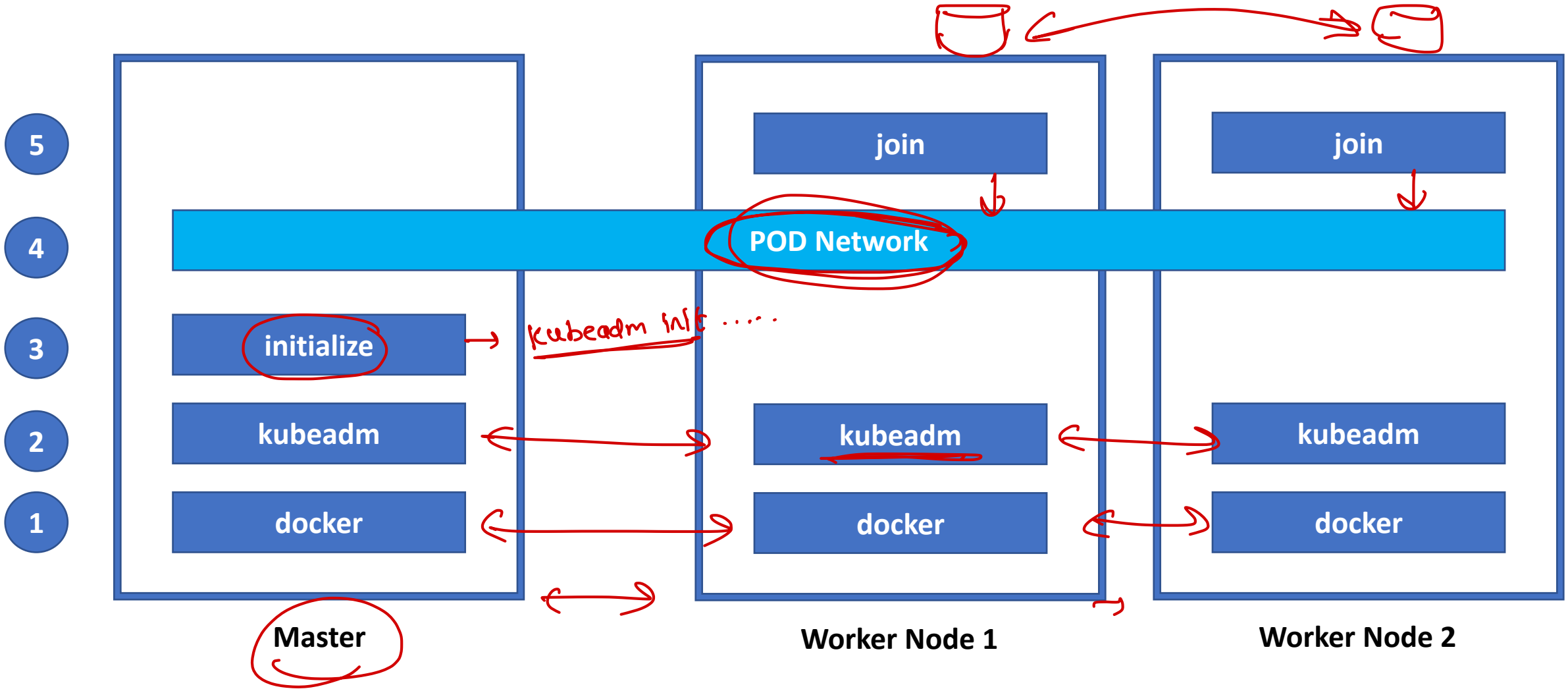
- Execute following command on every worker node

> kubeadm join --token <token> <control-plane-host>:<control-plane-port> --discovery-token-ca-cert-hash sha256:<hash>

Kubernetes Architecture



Steps to install Kubernetes



Nodes

- Node is not created by Kubernetes. It is created separately and then added to the cluster
- When Kubernetes creates a node, it creates an object to represent the node
- Node validation is done by checking the health of the node
- Valid nodes are eligible for running pods, otherwise they will be ignored while any scheduling activity
- Kubernetes keeps the node is invalid status and keeps checking to see if it becomes valid
- If the node is not required anymore to be the part of cluster, you need to delete it explicitly
- Tasks
  - Get the current cluster info
  - Get the list of nodes in the cluster
  - Find the information about any node

Core Concepts (13%)

Contents

- Understand Kubernetes API primitives
- Create and configure basic Pods

API Model

- Kubernetes API is resource-based (RESTful) programmatic interface provided via HTTP
- REST APIs are the fundamental fabric of kubenetes
- All operations, communications between components and commands are handled by the APIs
- Standard HTTP methods (POST, PUT, PATH, DELETE, GET)
- Change notifications on resources via watches
- Everything in the Kubernetes platform is treated as an API object
- Kubernetes provides a command like tool (kubectl) which consumes the APIs
- You can directly call the APIs programmatically

- Tasks
  - Get the list of APIs

Resource categories

- Workloads
  - Manage and run the application containers on the cluster
  - Common controllers: Deployments, StatefulSets and Jobs
- Discovery and load-balancer
  - Externally accessible and load balanced services
  - E.g. Services and Ingress
- Config and storage
  - Inject data into the application and keep the data persistent
  - E.g. ConfigMaps, Secrets and Volumes
- Cluster Resources
  - Configuration of the cluster
  - E.g. Cluster Roles, Role Binding and Certificate Signing
- Metadata resources
  - Configuring behavior of the resources within the cluster
  - E.g. HorizontalPodAutoscaler (HPA), PodDisruptionBudget and Notification events

API Terminology

- Resource Type: Name used in the URL
- Resource Kind: Representation of the resource type
- Collection: Instance list of a resource type
- Resource: Single instance of a resource type

API Versioning

- Version is set at the API level rather than at the resource or field level
- Alpha
  - Version name contains alpha e.g. v1alpha1
  - Recommended for use only in short-lived testing clusters
- Beta
  - Version name contains beta e.g. v2beta2
  - Recommended for only non-business-critical uses
- Stable
  - Stable versions e.g. v1, v2, v3 etc
- Tasks
  - Get the list of api versions

API Groups

- Core Group
  - Also known as legacy group
  - apiVesion is used as it is e.g. v1
- Named group
  - apiVersion as $GROUP_NAME/$VERSION
  - E.g. batch/v1

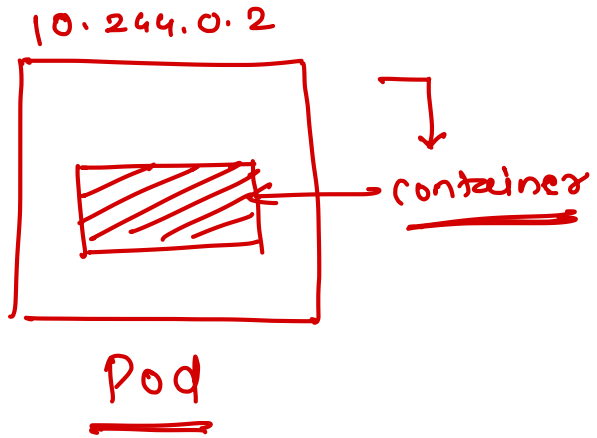
- Tasks
  - Get the API groups

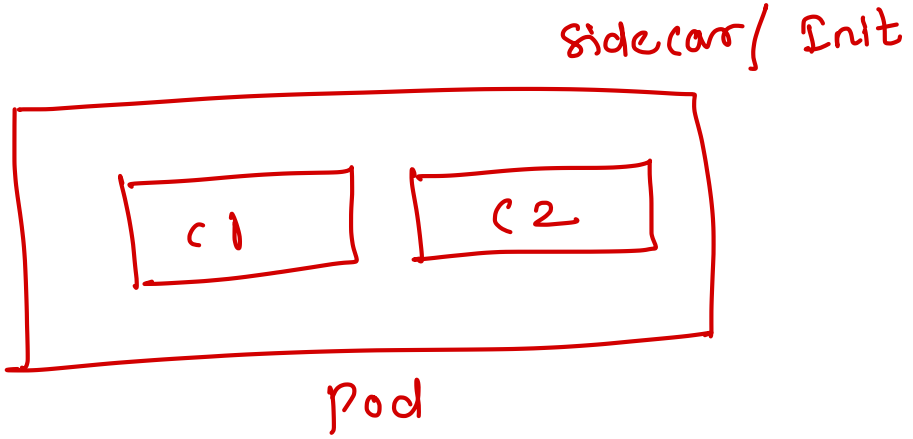
Kubernetes objects

- Persistent entities expressed in YAML format
- Represent the desired state of the cluster
- Describes the following
  - Containerized application running in the cluster
  - Resources made available to these applications
  - Policies for these applications
- Objects can be created using command line utility (kubectl)
- You can also use client libraries to make the API calls direcly

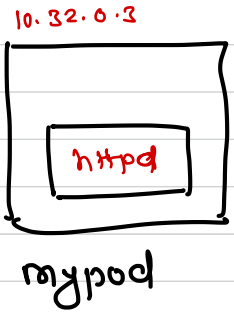
Pods

- Smallest deployable unit in the Kubernetes
- A pod encapsulates
  - Single or multiple containers (Init or App containers)
  - Storage resources
  - Unique Network IP
  - Options governing how the containers should run
  - Ephemeral and disposable entities
- Task
  - Create your first pod using yaml file





> kubectl run --image http mypod



YAML

YAML

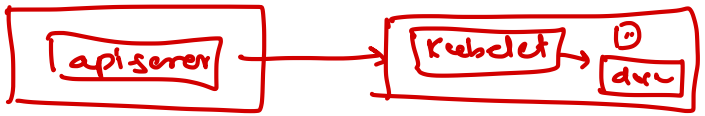
kubectl

1. create : kubectl create -f <file>
2. get : kubectl get <resource type>
3. describe : kubectl describe <resource> <name>
4. delete : kubectl delete <resource> <name>

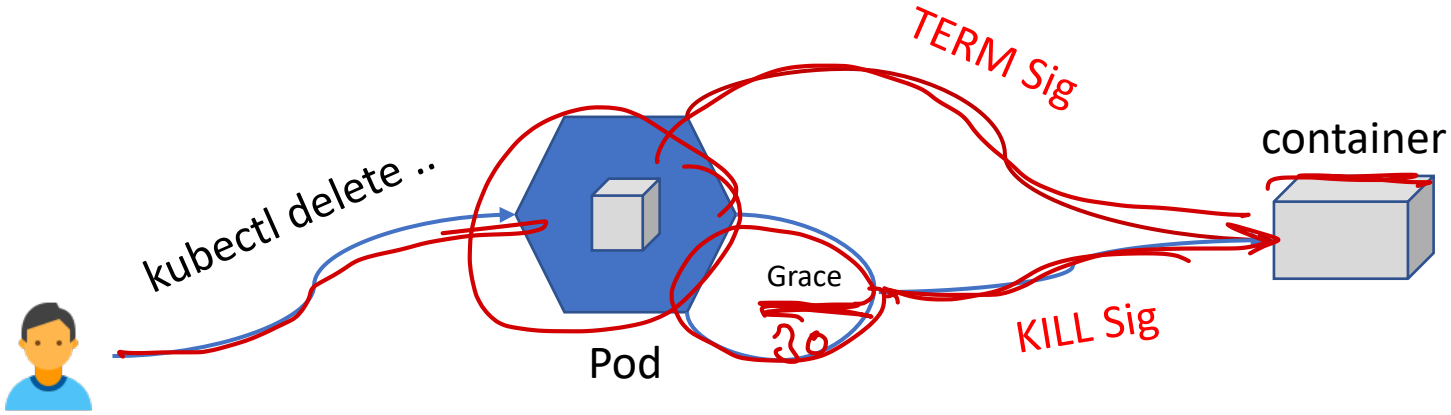
Phases of Pod

- Phase of a Pod is a simple high level summary showing what state the Pod is in
- Pending
  - Accepted by the cluster
  - Waiting to be scheduled
- Running
  - Scheduled on one of the worker nodes
  - In running/starting/restarting state
- Succeeded
  - All containers in the Pod are terminated successfully and will not be restarted
- Failed
  - At least one container terminated with failure state
- Unknown
  - For some reason the state is not known (may be due to the node communication error)

Container States



- Once the Pod is assigned to a node by scheduler, kubelet starts creating containers using container runtime (like Docker)
- There are three states
  - Waiting
  - Running
  - Terminated



- **Terminating Pod**
  - By default, all deletes are graceful within 30 seconds
  - kubectl delete commands supports the – –grace-period=<seconds> option
  - Value of grace-period zero will forcefully delete the pod

Kubernetes Object Management

- The kubectl supports 3 ways to manage the objects
- Imperative commands
  - Operates directly on the live objects
  - Simple with single step commands
  - Do not provide template for creating new objects
  - E.g. kubectl run nginx --image nginx
- Imperative object configurations
  - Specify the file containing the full definitions of obejcts
  - Configuration is simple to understand and can be source controlled
  - Requires additional step of creating YAML file
  - E.g. kubectl create -f config1.yml
- Declarative object configuration
  - User does not define the operations to be taken
  - Create, update and delete operations are automatically detected per-object by kubectl
  - Harder to debug and understand results
  - E.g. kubectl apply –f <directory>/

Kubernetes Objects - Names

- Every Kubernetes resource requires a unique name within a namespace
- Two different namespaces can have a pod with same name
- Resources have UID as well, which is unique across the whole cluster
- Resource can have names up to 253 characters long
- Allowed characters: digits (0-9), lower case letters (a-z) and dot (.)
- Tasks
  - Find the name of running pod
  - Find uid of a running pod

Namespaces

- Intended for use in environments with many users spread across multiple teams, or projects
- Namespaces provide a scope for resource names
- Names of resources need to be unique within a namespace, but not across namespaces
- Namespaces can not be nested inside one another and each Kubernetes resource can only be in one namespace
- Namespaces are a way to divide cluster resources between multiple users
- When a namespace is deleted, all the resources within the namespace will also get deleted
- Default namespaces
  - default: for objects with no specified namespace
  - kube-public: reserved for cluster usage
  - kube-node-lease: contains lease objects
  - kube-system: objects created by Kubernetes system

Namespaces

- Tasks
  - List all the namespaces
  - Create a new namespace using command
  - Create a new namespace using yaml file

Labels

- Formulate system objects in a structured manner
- They are optionally present in the configuration
- If present
  - They are key-value pars that are attached to the API objects
  - Each key must be unique in the key-value pairs list
- Tasks
  - Add a pod with labels attached to it