Day 1

What is Hypevisor?

- virtualization technology
- we can run multiple OS on the same laptop/desktop/workstation/server
- there are 2 types of hypervisors
    - Type 1 - aka Bare Metal Hypervisors (Used in wokstation/servers)
        - Examples - VMWare vSphere/vCenter
    - Type 2 - Laptop/Desktop/Workstation
        - Examples
            - VMWare Fusion (Mac OS-X)
            - VMWare Workstation - Linux & Windows
            - Oracle VirtualBox (Mac, Linux & Windows) - Free
            - Parallels (Mac OS-X)
            - KVM - Opensource (Linux)
    - heavy weight virtualization
    - each Virtual Machine has to allocated with dedicated hardware resources
        - CPU Cores
        - RAM
        - Storage
        - Network (virtual)
        - Graphics (virtual)
    - each VM represents 1 Operating System

Processor

- Processor Packaging
    1. SCM (Single Chip Module)
    - In 1 IC only 1 Processor will be there
    2. MCM (Multiple Chip Module)
    - In 1 IC 2/4/8 Processors will be there

Multi-core Processors

- Server grade Processors supports 128/256/512 CPU cores per Processor

What is Hyperthreading?

- each Physical CPU Core supports running 2 parallel threads simultaneously
- each Physical core is seen as 2 virtual cores by the hypervisor software
- Example
    - Assume we have a server with 8 Processor Sockets
    - Assume we have installed a MCM Processor on each Processor Socket
    - Assume the MCM supports 4 Processor/IC
    - Assume each Processors supports 256 CPU Cores

  - Assume each Physical core supports 2 virtual cores
  - Total Processors - 32 Processors
  - Total Physical CPU Cores - 32 x 256 = 8192
  - Total Virtual CPU Cores = 8192 x 2 = 16384 virtual Cores

What is the minimal number of servers to support 1000 Virtual Machines?

- 1 Physical server is enough to support 1000 Virtual Machines
  - Motherboard with 8 Processor Sockets
  - Each Processor Socket is installed with MCM Processor
  - Each Processor supports 256 CPU Cores
  - Each Physical CPU core supports 2 Virtual Cores
  - with 16384 virtual cores, we could easily support 1000 virtual Machines
- Virtualization helps in consolidating many physical server with 1/2 Physical server

Why do you think Developers/QA require so many virtual machines or Operating System?

- Let's assume the dev team is working on a software product and source code is cross-platform
- the same product is supported in Windows, Mac and several Linux Distributions
- the same product is supported in 32/64 bit OS

Containerization

- light weight virtualization technology
- Linux Kernel features behind containererization
  1. Namespace
  2. Control Groups aka CGroup
- containers running on the same servers shares the hardwares resources on the underlying host os
- containers are not Operating System
- container does not have OS Kernel
- container has one application and its dependent libraries
- application virtualization technology
- each container represents one application process
- the reason why many of us tend to compare a container with a virtual machine
  - just like virtual machines acquire one or more IP addresses, containers also gets its own dedicated IP address
  - just like virtual machine has file system, containers also has its own file system
  - just like virtual machine with linux distributions supports package managers, linux containers also has their own package managers (apt/apt-get,rpm,yum,dnf)
  - just like virtual machine, the containers also has their own port range (0 - 65535)

What is a container runtime?

- is a low-level software that manages the container images and containers
- not user-friendly, hence normally engineers won't directly use the container runtime
- examples

  - runC container runtime
  - CRI-O container runtime

What is a container engine?

- high-level softwares
- very user-friendly, offers easy to use commands to manage container images and container life-cycle
- internally they depend on container runtime to manage container images and container life-cycle
- examples
  - Docker Container Engine depends on containerd which in turn depends on runc container runtime
  - Podman Container engine depends on CRI-O Container Runtime

What is Container Image?

- Container Image is similar to OS Image - Ubuntu-16-04.iso
- Using container Images we can create multiple containers
- Container Images comes with some pre-installed software tools like package managers, unix tools like ls, cp,rm, etc.,
- though container images are named as OS names, they don't represent an Operating System
- through container images we can only create containers which are application process that runs in a separate namespace

What are the Control Plane Components?

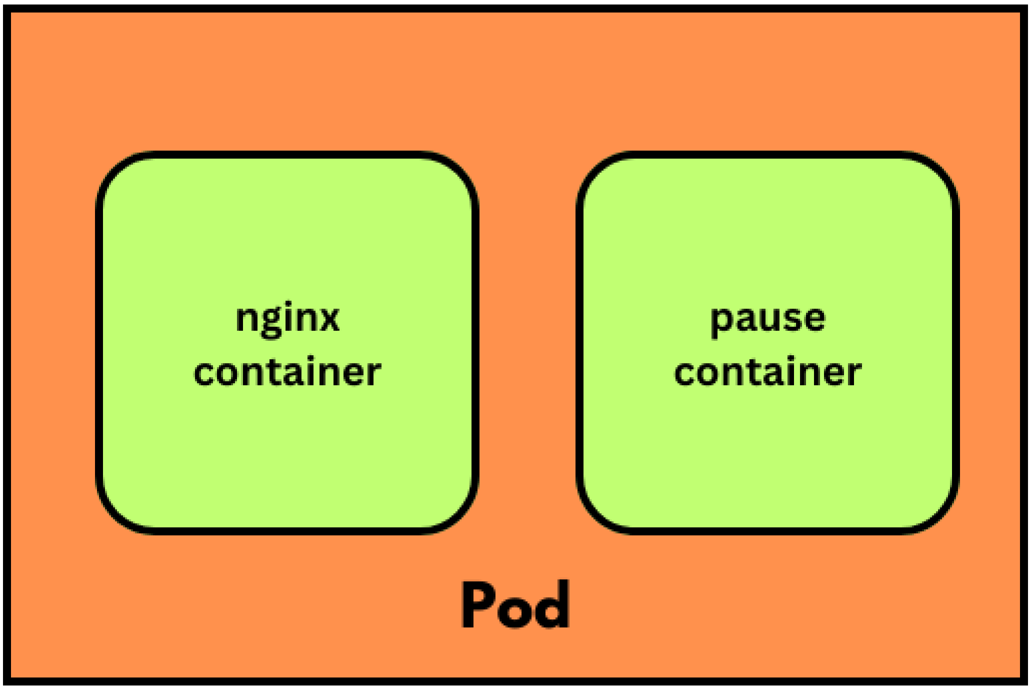
1. API Server
2. etcd key-value data-store (database)
3. scheduler
4. controller managers

- collection of many controllers
- Examples
  - Deployment Controller
  - ReplicaSet Controller
  - Job Controller
  - DaemonSet Controller
  - StatefulSet Controller
  - CronJob Controller
  - EndPoint Controller
- the control plane components runs only in master node
- the control plane components supports the Container Orchestration features
- the control plane helps in deploying our containerized applications
- the control plane helps in monitoring the health of our deployed application, repairs them on-demand, replaces with new applications instances when required
- the control plane also makes our application High Availability (HA)
- control planes supports scaling up/down our application based on user-traffic
- control plane supports rolling updates
  - upgrading our appliction from one version to other without any downtime

- control plane with the help of kube-proxy supports in-built load balancing to our application workloads
- control plane with the help of core-dns supports service discovery

Pod Overview

- a group of related containers
- within the Pod container, application will be running
- is a JSON/YAML Definition which is stored in the etcd database
- is a Kubernetes/Openshift resource
- the smallest unit that can be deployed within Kubernetes/Openshift
- Pod gets it own IP address
- all the containers that are part of a single Pod, shares the same IP Address and ports
- recommended best practice is one main application per Pod



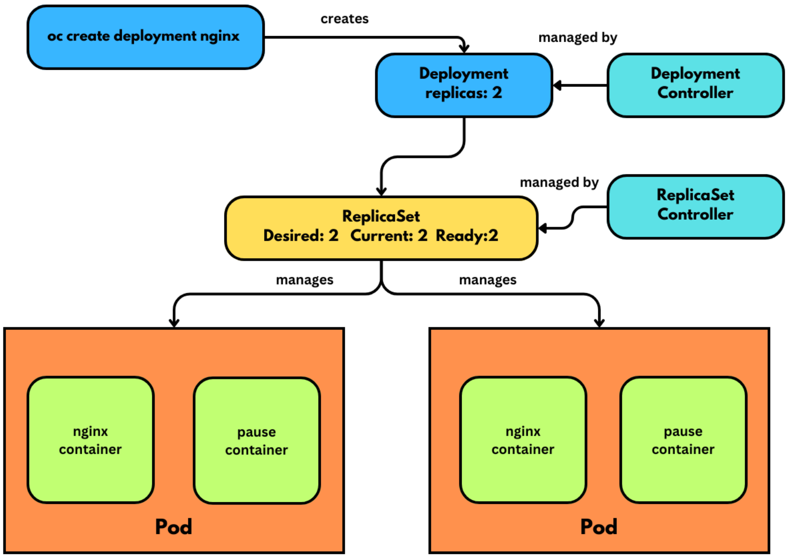
ReplicaSet Overview

- Let's say we wish to run many instances of our application
- supports scale up (running many instances of same application)
- supports scale down (deleting unwanted extra pod instance of aour application)
- is a Kubernetes/Openshift resource managed by ReplicaSet Controller
- ReplicaSet has one to many Pods

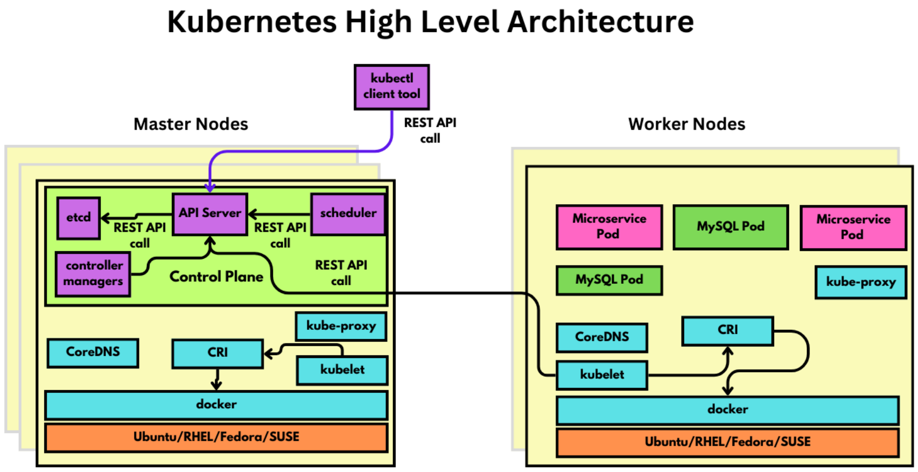
Deployment Overview

- This represents our application
- the deployment will have an unique name and id
- the name is user-defined, the id is auto-assigned by Deployment Controller
- Deployment is Kubernetes/Openshift resource managed by Deployment Controller

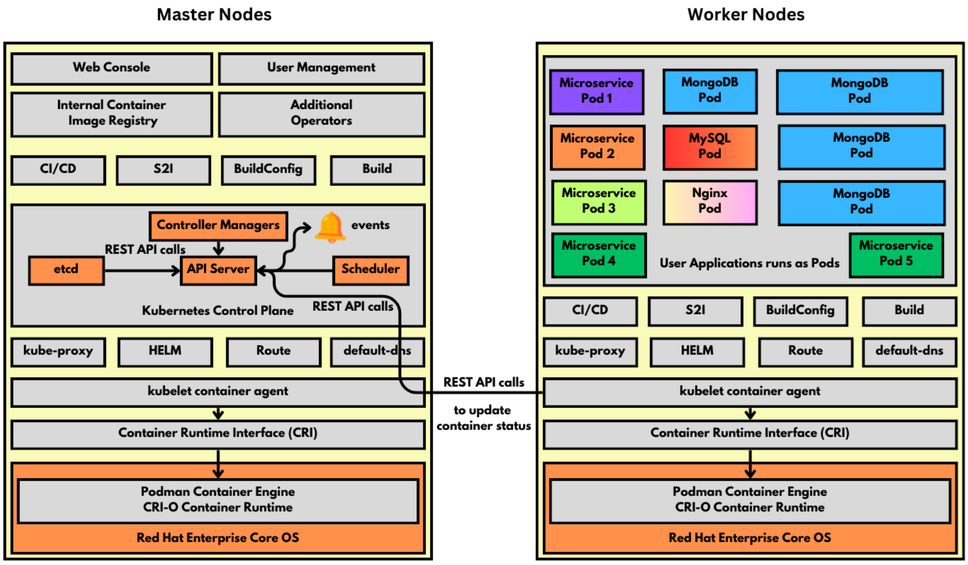
- Deployment has one to many ReplicaSets
- For each version of a container image, one ReplicaSet will be created within Deployment



Info - Kubernetes High-Level Architecture



Info - Red Hat OpenShift Architecture



Lab - Listing the nodes in the Openshift cluster

```
oc get nodes
```

Expected output

```
jegan@tektutor.org $ oc get nodes
NAME                              STATUS   ROLES
AGE    VERSION
master-1.ocp4.tektutor.org.labs   Ready    control-plane,master,worker
7d5h   v1.28.9+416ecaf
master-2.ocp4.tektutor.org.labs   Ready    control-plane,master,worker
7d5h   v1.28.9+416ecaf
master-3.ocp4.tektutor.org.labs   Ready    control-plane,master,worker
7d5h   v1.28.9+416ecaf
worker-1.ocp4.tektutor.org.labs   Ready    worker
7d5h   v1.28.9+416ecaf
worker-2.ocp4.tektutor.org.labs   Ready    worker
7d5h   v1.28.9+416ecaf
```

In the command above, oc is the openshift's client tool.

We could use the kubernetes client in openshift

```
kubectl get nodes
```

Expected output

```
jegan@tektutor.org $ kubectl get nodes
NAME                              STATUS   ROLES
AGE    VERSION
master-1.ocp4.tektutor.org.labs   Ready    control-plane,master,worker
7d5h   v1.28.9+416ecaf
master-2.ocp4.tektutor.org.labs   Ready    control-plane,master,worker
7d5h   v1.28.9+416ecaf
master-3.ocp4.tektutor.org.labs   Ready    control-plane,master,worker
7d5h   v1.28.9+416ecaf
worker-1.ocp4.tektutor.org.labs   Ready    worker
7d5h   v1.28.9+416ecaf
worker-2.ocp4.tektutor.org.labs   Ready    worker
7d5h   v1.28.9+416ecaf
```

Lab - How does the kubectl and oc client know how to communicate with openshift cluster

```
oc get nodes
kubectl get nodes
```

oc and kubectl clients makes a REST API Call to API Server running in the master nodes (master-1, master2 and master3)