Terraform

Terraform:
Terraform is a product by Hashicorp that uses Infrastructure as Code (IaC) to provision cloud infrastructure.
It lets you define resources and infrastructure in human-readable, declarative configuration files, and manages your infrastructure's lifecycle.

What is Terraform?
Terraform is an IaC tool that allows users to provision and manage infrastructure resources across various cloud platforms and on-premises environments. It uses declarative configuration files, written in HashiCorp Configuration Language (HCL) or JSON, to define and automate the lifecycle of resources, ensuring predictability and consistency. Terraform's extensible plugin-based architecture supports a wide range of providers, enabling seamless integration and management of diverse infrastructure environments.

What is Infrastructure as Code with Terraform?
Infrastructure as Code (IaC) tools allow you to manage infrastructure with configuration files rather than through a graphical user interface. IaC allows you to build, change, and manage your infrastructure in a safe, consistent, and repeatable way by defining resource configurations that you can version, reuse, and share.

- Terraform can manage infrastructure on multiple cloud platforms.
- The human-readable configuration language helps you write infrastructure code quickly.
- Terraform's state allows you to track resource changes throughout your deployments.
- You can commit your configurations to version control to safely collaborate on infrastructure.

Popular IaC Tools:

1. **Terraform** An open-source declarative tool that offers pre-written modules to build and manage an infrastructure.
   2. Chef: A configuration management tool that uses cookbooks and recipes to deploy the desired environment. Best used for Deploying and configuring applications using a pull-based approach.
   3. Puppet: Popular tool for configuration management that follows a Client-Server Model. Puppet needs agents to be deployed on the target machines before the puppet can start managing them.
   4. Ansible: Ansible is used for building infrastructure as well as deploying and configuring applications on top of them. Best used for Ad hoc analysis.
   5. Packer: Unique tool that generates VM images (not running VMs) based on steps you provide. Best used for Baking compute images.
   6. Vagrant: Builds VMs using a workflow. Best used for Creating pre-configured developer VMs within VirtualBox.

Types of IAC Tools

Configuration Management

Server Templating

Provisioning Tools

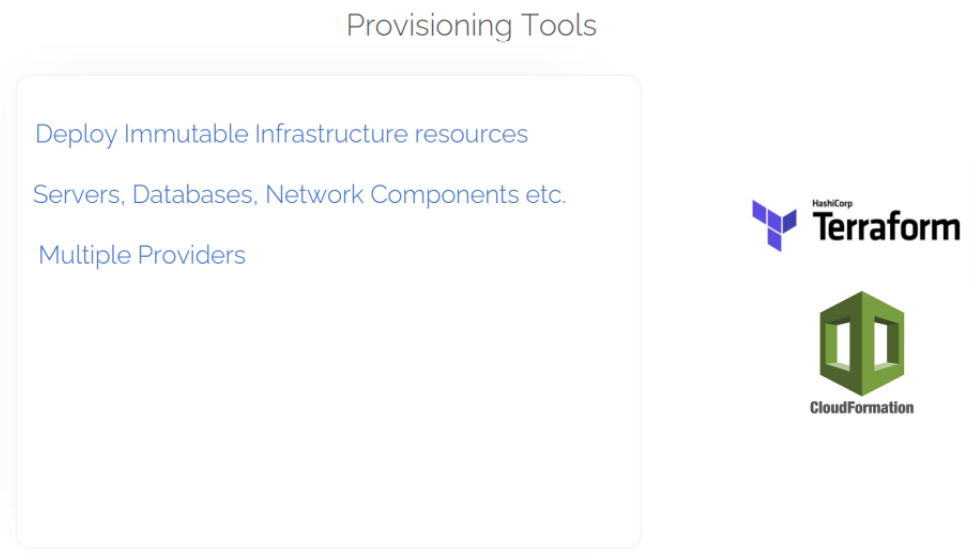
Types of IAC Tools

Configuration Management

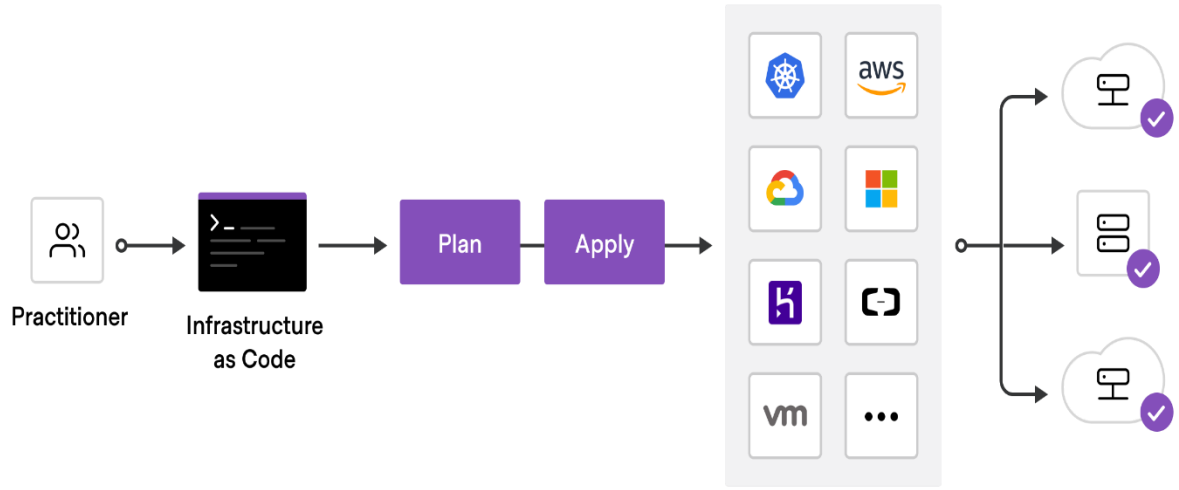
- Designed to Install and Manage Software
- Maintains Standard Structure
- Version Control
- Idempotent

Server Templating Tools

- Pre Installed Software and Dependencies
- Virtual Machine or Docker Images
- Immutable Infrastructure



Terraform deployment workflow:



Terraform Lifecycle:

Terraform lifecycle consists of – init, plan, apply, and destroy.

1. **Terraform init** initializes the (local) Terraform environment. Usually executed only once per session.
2. **Terraform plan** compares the Terraform state with the as-is state in the cloud, build and display an execution plan. This does not change the deployment (read-only).
3. **Terraform apply** executes the plan. This potentially changes the deployment.
4. **Terraform destroy** deletes all resources that are governed by this specific terraform environment.

Terraform Core Concepts

1. Variables: Terraform has input and output variables, it is a key-value pair. Input variables are used as parameters to input values at run time to customize our deployments. Output variables

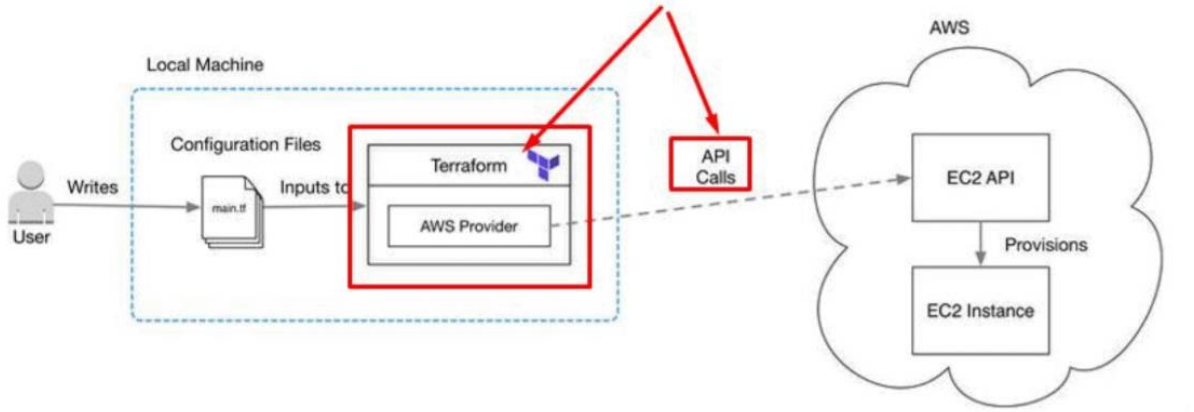
are return values of a terraform module that can be used by other configurations.
Read our blog on **Terraform Variables**
2. Provider: Terraform users provision their infrastructure on the major cloud providers such as AWS, Azure, OCI, and others. A *provider* is a plugin that interacts with the various APIs required to create, update, and delete various resources.
Read our blog to know more about **Terraform Providers**
3. Module: Any set of Terraform configuration files in a folder is a *module*. Every Terraform configuration has at least one module, known as its ***root module.***
4. State: Terraform records information about what infrastructure is created in a Terraform *state* file. With the state file, Terraform is able to find the resources it created previously, supposed to manage and update them accordingly.
5. Resources: Cloud Providers provides various services in their offerings, they are referenced as Resources in Terraform. Terraform resources can be anything from compute instances, virtual networks to higher-level components such as DNS records. Each resource has its own attributes to define that resource.
6. Data Source: Data source performs a read-only operation. It allows data to be fetched or computed from resources/entities that are not defined or managed by Terraform or the current Terraform configuration.
7. Plan: It is one of the stages in the Terraform lifecycle where it determines what needs to be created, updated, or destroyed to move from the real/current state of the infrastructure to the desired state.
8. Apply: It is one of the stages in the Terraform lifecycle where it applies the changes real/current state of the infrastructure in order to achieve the desired state.
Check Out: Our previous blog post on **Terraform Cheat Sheet**.

Terraform Providers:

Aws Provider

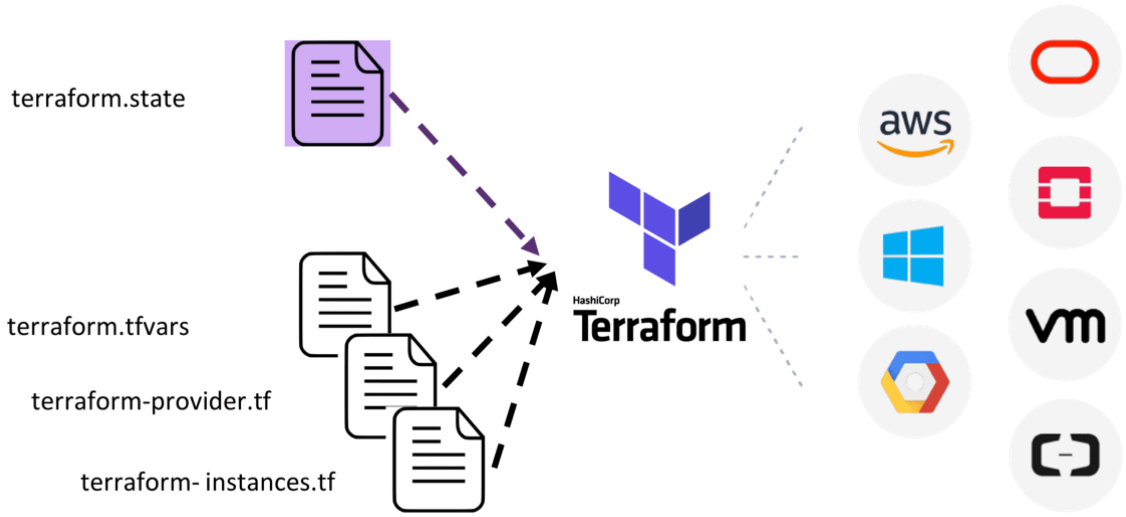
```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "5.63.1"
    }
  }
}

provider "aws" {
  # Configuration options
}
```

A provider is responsible for understanding API interactions and exposing resources. It is an executable plug-in that contains code necessary to interact with the API of the service. Terraform configurations must declare which providers they require so that Terraform can install and use them.



Terraform has over a hundred providers for different technologies, and each provider then gives terraform user access to its resources. So through AWS provider, for example, you have access to hundreds of AWS resources like EC2 instances, the AWS users, etc.
Read More: About [Terraform Workflow].

Terraform Configuration Files
Configuration files are a set of files used to describe infrastructure in Terraform and have the file extensions **.tf** and **.tf.json**. Terraform uses a declarative model for defining infrastructure. Configuration files let you write a configuration that declares your desired state. Configuration files are made up of resources with settings and values representing the desired state of your infrastructure.



A Terraform configuration is made up of one or more files in a directory, provider binaries, plan files, and state files once Terraform has run the configuration.
1. Configuration file (*.tf files): Here we declare the provider and resources to be deployed along with the type of resource and all resources specific settings
2. Variable declaration file (variables.tf or variables.tf.json): Here we declare the input variables required to provision resources
3. Variable definition files (terraform.tfvars): Here we assign values to the input variables
4. State file (terraform.tfstate): a state file is created once after Terraform is run. It stores state about our managed infrastructure.
Also Read: Our blog post on [Terraform Create VM].

For more information:
https://k21academy.com/terraform-iac/terraform-beginners-guide/

Terraform resources:

```
resource "aws_instance" "my_vm" {
 ami                    = "ami-065deacbcaac64cf2" //Ubuntu AMI
 instance_type          = "t2.micro"

 tags = {
   Name = "My EC2 instance",
 }
}
```

Terraform variables:
Variables introduce the flexibility and dynamism needed to manage larger sets of infrastructure.
There are mainly three types of variables in Terraform – local, input, and output.

Local variables:
These are locally declared variables.
They provide a way to name any attribute value that needs to be used throughout the Terraform code. Local variables are particularly useful when you need to reference the same values in multiple places within your Terraform configuration.

variables.tf

```
locals {
 ami      = "ami-065deacbcaac64cf2"
 type     = "t2.micro"
 name_tag = "My EC2 Instance"
}
```

main.tf

```
resource "aws_instance" "my_vm" {
 ami           = local.ami //Ubuntu AMI
 instance_type = local.type

 tags = {
   Name = local.name_tag,
 }
}
```

Input variables

variables.tf

```
variable "ami" {
   type        = string
   description = "Ubuntu AMI ID in N. Virginia Region"
   default     = "ami-065deacbcaac64cf2"
}

variable "instance_type" {
   type        = string
   description = "Instance type"
   default     = "t2.micro"
}

variable "name_tag" {
   type        = string
   description = "Name of the EC2 instance"
   default     = "My EC2 Instance"
}
```

main.tf

```
resource "aws_instance" "my_vm" {
 ami           = var.ami //Ubuntu AMI
 instance_type = var.instance_type

 tags = {
   Name = var.name_tag,
 }
}
```

Terraform.tfvars:

A better way to manage these default values is to create another file named terraform.tfvars. Terraform automatically interprets tfvars as a group of key-value pairs and maps them with the declared variables in the variables.tf file.

terraform.tfvars

```
filename = "/root/pets.txt"
content = "We love pets!"
prefix = "Mrs"
separator = "."
length = "2"
```

```
>_
$ terraform apply -var-file variables.tfvars
```

Automatically Loaded

terraform.tfvars	terraform.tfvars.json
*.auto.tfvars	*.auto.tfvars.json

Variable Definition Precedence

main.tf

```
resource local_file pet {
  filename = var.filename
}
```

variables.tf

```
variable filename {
  type    = string
}
```

```
>_
$ export TF_VAR_filename="/root/cats.txt" ?
```

terraform.tfvars

```
filename = "/root/pets.txt" ?
```

variable.auto.tfvars

```
filename = "/root/mypet.txt" ?
```

```
>_
$ terraform apply -var "filename=/root/best-pet.txt" ?
```

Variable Definition Precedence

Order	Option
1	Environment Variables
2	terraform.tfvars
3	*.auto.tfvars (alphabetical order)
4	-var or –var-file (command-line flags)

```
>_
$ export TF_VAR_filename="/root/cats.txt" 1
```

terraform.tfvars

```
filename = "/root/pets.txt" 2
```

variable.auto.tfvars

```
filename = "/root/mypet.txt" 3
```

```
>_
$ terraform apply -var "filename=/root/best-pet.txt" 4
```

Output Variables

Output Variables provide a way to retrieve the details we are interested in directly in the CLI terminal. Additionally, output variables "return" values to the parent modules for further processing.

```
output "public_ip" {
 value       = aws_instance.my_vm.public_ip
 description = "Public IP Address of EC2 instance"
}

output "instance_id" {
 value       = aws_instance.my_vm.id
 description = "Instance ID"
}
```

Terraform Commands:

1. **terraform validate**
2. **terraform fmt**
3. **terraform show**
4. **terraform providers**
5. **terraform output**
6. **terraform refresh**
7. **terraform graph**

Terraform State management:

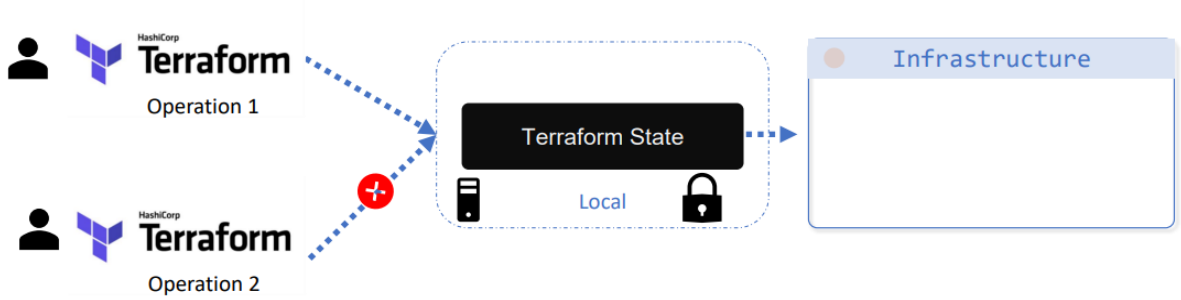
State management in Terraform is one of the crucial things to know and learn about especially when working with teams

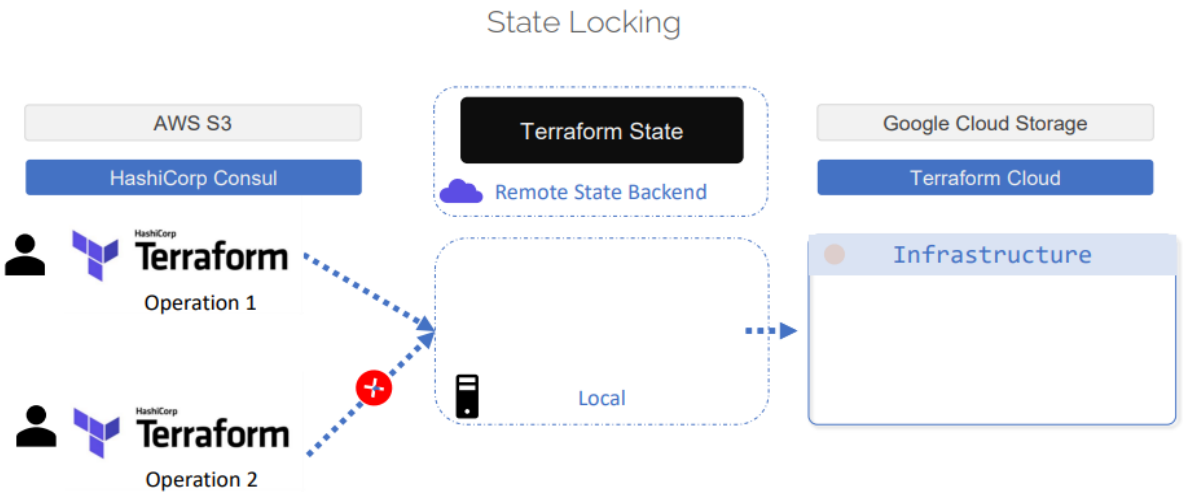
1. terraform.tfstate
2. terraform.tfstate.backup

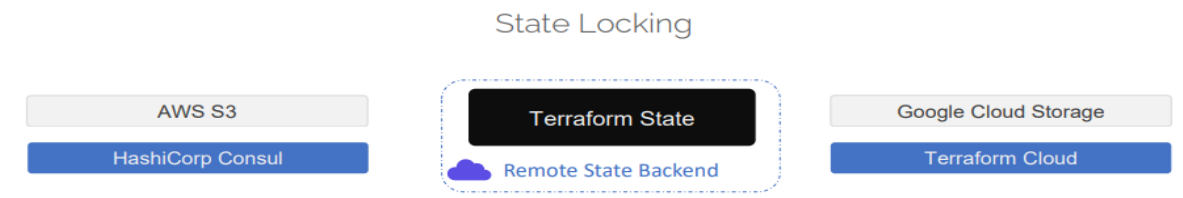
State lock file

There is one more file involved in all of this – .terraform.tfstate.lock.info.

State Locking



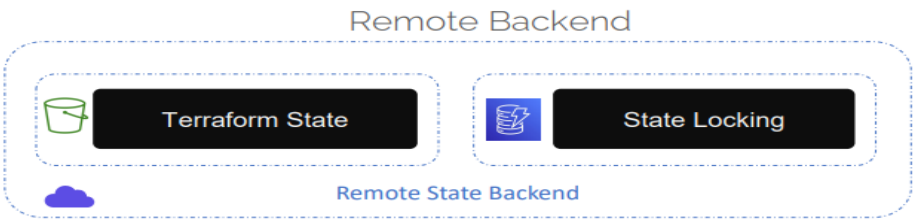


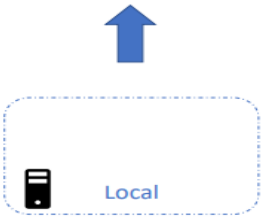


Automatically Load and Upload State File

Many Backends Support State Locking

Security





Object	Value
Bucket	kodekloud-terraform-state-bucket01
Key	finance/terraform.tfstate
Region	us-west-1
DynamoDB Table	state-locking

main.tf

```
resource "local_file" "pet" {
  filename = "/root/pets.txt"
  content = "We love pets!"
}


terraform {
  backend "s3" {
    bucket         = "kodekloud-terraform-state-bucket01"
    key            = "finance/terraform.tfstate"
    region         = "us-west-1"
    dynamodb_table = "state-locking"
  }
}
```

```
$ ls
main.tf  terraform.tfstate
```

Object	Value
Bucket	kodekloud-terraform-state-bucket01
Key	finance/terraform.tfstate
Region	us-west-1
DynamoDB Table	state-locking

main.tf

```
resource "local_file" "pet" {
  filename = "/root/pets.txt"
  content = "We love pets!"
}
```

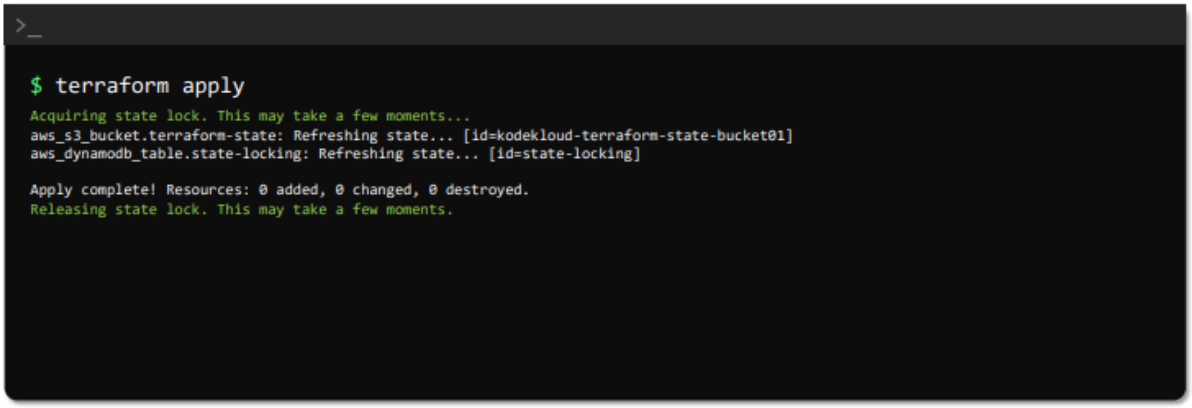
terraform.tf

```
terraform {
  backend "s3" {
    bucket         = "kodekloud-terraform-state-bucket01"
    key            = "finance/terraform.tfstate"
    region         = "us-west-1"
    dynamodb_table = "state-locking"
  }
}
```

```
$ terraform apply
Backend reinitialization required. Please run "terraform init". Reason: Initial configuration of the requested backend "s3"

The "backend" is the interface that Terraform uses to store state, perform operations, etc. If this message is showing up, it means
that the Terraform configuration you're using is using a custom configuration for the Terraform backend.

Changes to backend configurations require reinitialization. This allows Terraform to setup the new configuration, copy existing
state, etc. This is only done during "terraform init". Please run that command now then try again.

Error: Initialization required. Please see the error message above.
```

```
$ terraform init
 Initializing the backend...
 Do you want to copy existing state to the new backend?
   Pre-existing state was found while migrating the previous "local" backend to the newly configured "s3" backend. No existing state
 was found in the newly configured "s3" backend. Do you want to copy this state to the new "s3"
   backend? Enter "yes" to copy and "no" to start with an empty state.

   Enter a value: yes

 Successfully configured the backend "s3"! Terraform will automatically use this backend unless the backend configuration changes.

 Initializing provider plugins...
 - Using previously-installed hashicorp/aws v3.7.0
 .
 .[Output Truncated]
```

```
$ rm -rf terraform.tfstate
```



Terraform Taint

Debugging:

Log Levels
#export TF_LOG=TRACE

Terraform Import

Data Source:

Data Source

main.tf

```
.
.
data "aws_instance" "newserver" {
    instance_id = "i-026e13be10d5326f7"
}
output newserver {
  value       = data.aws_instance.newserver.public_ip
 }
```

```
>_

$ terraform apply

$ data.aws_instance.newserver: Refreshing state... [id=i-026e13be10d5326f7]
aws_key_pair.web: Refreshing state... [id=terraform-20201015013048509100000001]
aws_security_group.ssh-access: Refreshing state... [id=sg-0a543f25009e14628]
aws_instance.webserver: Refreshing state... [id=i-068fad300d9df27ac]

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.

Outputs:

newserver = 15.223.1.176
```

Terraform Modules:

Modules are containers for multiple resources that are used together. A module consists of a collection of .tf and/or .tf.json files kept together in a directory.
Modules are the main way to package and reuse resource configurations with Terraform.

Terraform AWS modules

https://registry.terraform.io/namespaces/terraform-aws-modules
https://github.com/terraform-aws-modules

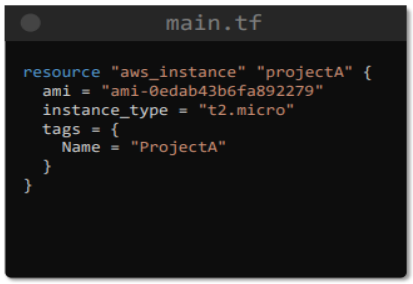
Terraform Functions:

Function Type	Description
String	String related operations (format, join, split)
Numeric	Numeric related operations (min, max, pow)
Collection	Functions that manipulate lists, tuples, sets and maps (length, lookup, merge)
Date and Time	Manipulate date and time (formatdate, timestamp)

Crypto and Hash	**Crypto and Hash functions (base64sha512, bcrypt)**
Filesystem	**File system operations (file, filexists, abspath)**
Ip Network	**Network Cidr functions (cidrsubnet, cidrhost)**
Encoding	**Encoding and decoding functions (base64decode, base64encode, jsonencode)**
Type Conversion	**Functions that convert data types (tobool, tomap, tolist)**

Please follow the below link for more details.
https://spacelift.io/blog/terraform-functions-expressions-loops
https://developer.hashicorp.com/terraform/language/functions

Terraform Workspaces:



Refer the below link for more information:
https://spacelift.io/blog/terraform-workspaces

Terraform Project

1. Terraform Project with Aws resources.

Q. Create AWS EC2 instance by supplying arguments with variables and use data source to grab the AMI (Latest ami of amazon linux) from the aws. Access the html file in the shell scripting and run it(add data file to access shell script) and also add security groups with inbound rules as http?

A.

main.tf:

```
#terraform provider.
terraform {
 required_providers {
  aws = {
   source = "hashicorp/aws"
   version = "5.64.0"
  }
 }
}

provider "aws" {
  region = var.region

}

#aws ec2 instance block
resource "aws_instance" "web" {
 ami         = data.aws_ami.linux.id
 instance_type = var.Instance_type

 tags = {
  Name = "Linux"
 }
}

#Data resource block to grab the latest ami
data "aws_ami" "linux" {
 most_recent = true
 owners     = ["amazon"]
 filter {
  name   = "architecture"
  values = ["arm64"]
 }
 filter {
  name   = "name"
  values = ["amzn2-ami-hvm-*-x86_64-gp2"]
 filter {
  name = “virtualization”
  values= [“hvm”]
 }
}

#allow security groups with inbound rules
resource "aws_security_group" "example" {
 # ... other configuration ...
  ingress {

  description = "HTTPS ingress"
```

```
  from_port      = 443
  to_port        = 443
  protocol       = " tcp"
  cidr_blocks     = ["0.0.0.0/0"]
  ipv6_cidr_blocks = ["::/0"]

 }

 egress {
  from_port      = 0
  to_port        = 0
  protocol       = "-1"
  cidr_blocks     = ["0.0.0.0/0"]
  ipv6_cidr_blocks = ["::/0"]

 }
}

#local file to call shell script.
resource "local_file" "foo" {
 content  = "foo!"
 filename = "${path.module}/foo.bar"
}
```

variables.tf:

```
#variable block
variable " region" {
 type       = string
 default    = “us-east-1 "
}
variable “Instance_type” {
type= string
default=”t2.micro”
}
```

shellscript.sh:

```
#shell script
```

Q. Create S3 buckets with policy and enable the static website host and enable versioning.

A.

```
#create s3 bucket
resource "aws_s3_bucket" "example" {
 bucket = "my-tf-test-bucket"
 }
```

}

Q. Create VPC using modules.
Add 2 availability zone and 2 public and private subnets and create database subnets
Attach internet and nat gateway and also use tfvars file for giving the variable?

Q. Create RDS and use null resources and also use locals and use environment variables?

Docker

Docker is containerization technology tool.
Kubernetes is container orchestration tool.

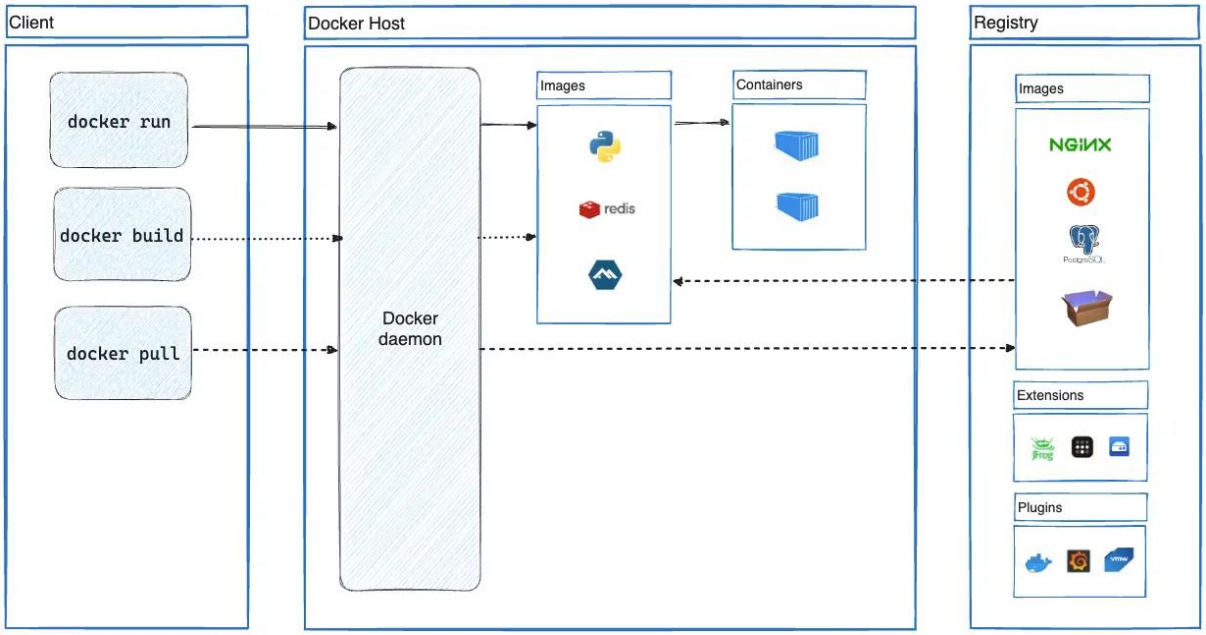
Docker is a container runtime.
Containerization:
Its all about deploy application with required dependencies is known as containerization.

Kubernetes:
Kubernetes is a platform for running and managing container for many(100's of) containers runtimes.

Virtualization:

Container Engine:
Software which helps to implement containerization on a machine or server is caller container Engine.
Tool:
Docker, Jail, Crio.

Docker Architecture:



Docker CLI/Client:

Docker client is used to interact with docker hub to interact with images or containers.

- **Docker pull :** It helps to get the images from docker hub into the server.
- **Docker build:** To create docker images.
- **Docker run:** To create the containers from the images.

Docker Daemon (Docker service):
Docker daemon manages all the services by communicating with other daemons. It manages docker objects such as images, containers, networks, and volumes with the help of the API requests of Docker.

(The Docker daemon listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes.)

Docker Host
A Docker host is a type of machine that is responsible for running more than one container. It comprises the Docker daemon, Images, Containers, Networks, and Storage.

Docker Registry
All the docker images are stored in the docker registry.
Registry : It is a collection of all dependencies and docker container images
Docker pull: It helps to download the images or dependencies from registry on to the docker host.
Docker pull <image name:version>

Docker Images
An image contains instructions for creating a docker container. It helps to create container from docker images and it is used to store and ship applications.

Docker Containers
Containers are created from docker images. With the help of Docker API or CLI, we can start, stop, delete, or move a container.

Docker Services :

- ➔ After docker install, we have to follow 3 steps (or) activities.
    1. Start the docker services.
       Ex: service docker start
    2. Enable docker services at boot time.
       chkconfig docker on
    3. Add the user account to the docker root group.
       Ex: usermod -a -G dockerroot <username>
- ➔ After installing docker, it will create group called dockerroot. We must add user account to the docker root group.

Docker commands:

How to check docker client and engine version ?
docker version

How to check docker server configuration ?
docker info

How to find the container image ?
Docker images

How to check the container information?
Docker ps -a
Or
Docker ps -> it will shows only live or running containers

How to create images ?
Docker build –tag <image Name> <file path to docker file>

How to create a container ?
Docker run -it <image name or Id> <container shell ex: sh, bash>
Note : -it is used for create and log into the container

How to close a container and come out ?
Type exit in container and click on enter.

How to come out from container without closing session ?
Ctrl + pq

How to stop container ?
Docker container stop <container id or name>

How to start container ?
Docker container start <container id or name>

How to log into the container or docker exec command ?
Docker exec -it <container name or id > <container shell>

Note : the container should be running and up to log into the container
If container is stopped or not running we have to start and log in.

How to start and log into a container?
Docker start -ai <container name or id >
Note : Here we can't change the shell, default shell is sh

Create or update container names ?
Docker rename <old container name > <new container name>

How to name a container while creating ?
Docker run –name <container name> -it <image id> <container shell>

How to delete containers ?
Docker rm <container id or name>
Note: always delete a container when status is showing as exited.
docker rm <container id or name> --force (to remove container forcefully)

How to delete multiple containers ?
Docker rm < container name1> name2 name3 etc...

How to delete docker images ?
Docker rmi <image name or id >
Note : it will only delete images not the container running from those images.

Image tagging :
Two types of image tags are available.

1. Local tag
   Local tag helps to create an alias name use case on docker server or host. We can't upload local tag images to docker registry
   Docker image tag <source image name or id > <new image name >

2. Remote tag
   We can generate image alias both for use case or keeping images on docker host and upload to the docker registry.
   Docker tag <source image name or id > <docker hub id>/<new image name>:<image version>

How to login to docker hub from cli mode ?
Docker login

How to push images to docker registry or hub ?
Docker push <image name>
Note: **only use image names to push images to docker hub**

How to logout from docker hub ?
Docker logout

Docker image inspect:
Inspect command shows the properties of the image.

How to get the image meta data info or properties ?
Docker image inspect <image id or name>

How to get the container meta data info or properties ?
Docker container inspect <container name or id>

Docker image history :
This command helps to check the layers of the image or code of the image.
Docker image history

Docker layer:
Docker build follows the process of interpretation it will check the code line by line.
If we have 4 lines in a code
First line code is correct it will create hashtag or commit id. If the hashtag crated this will generate a layer.

Image is a collection of layers
In dockerfile output layers are arranged in descending order
The last line of the layer in an image automatically considered as image id.
Each line of code in dockerfile we have to call as instruction.
If instruction executed correctly docker will create layer, all these instructions are written in a file is called dockerfile. For docker file no extension.

Docker build is process of converting instructions into layers if all layers are successfully executed then it will generate a docker image else if one of the instructions fails then layer will not generate.
If one of the layer fails then docker image fails to generate.

Dockerfile :

- Dockerfile helps to create images,
- File contains instructions.
- These instructions more look like Linux commands.

Work flow of docker image process:

1. Crete project folder
2. Cd into folder, create dockerfile. Name is Dockerfile.
3. Open the dockerfile, write the code and save it.
   Eg: FROM python:3.12
   WORKDIR /usr/local/app
   # Install the application dependencies
   COPY requirements.txt ./
   RUN pip install --no-cache-dir -r requirements.txt
   # Copy in the source code
   COPY src ./src
   EXPOSE 5000
   # Setup an app user so the container doesn't run as the root user
   RUN useradd app
   USER app
   CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8080"]
4. Excute it (Image build process)
   Commad:
   Docker image build –tag <image_name > <docker file path>
5. Result: docker ps
6. Then push to docker hub.

Docker instructions:

- **FROM <image>** - this specifies the base image that the build will extend.
- **WORKDIR <path>** - this instruction specifies the "working directory" or the path in the image where files will be copied and commands will be executed.
- **COPY <host-path> <image-path>** - this instruction tells the builder to copy files from the host and put them into the container image.
- **RUN <command>** - this instruction tells the builder to run the specified command.
- **ENV <name> <value>** - this instruction sets an environment variable that a running container will use.
- **EXPOSE <port-number> -** this instruction sets configuration on the image that indicates a port the image would like to expose.
- **USER <user-or-uid>** - this instruction sets the default user for all subsequent instructions.
- **CMD ["<command>", "<arg1>"]** - this instruction sets the default command a container using this image will run.

For more information about docker installations:
https://docs.docker.com/get-started/docker-concepts/building-images/writing-a-dockerfile/

Docker Networking

Docker Networking allows you to create a Network of Docker Containers managed by a master node called the manager.
A network is a group of two or more devices that can communicate with each other either physically or virtually.
The Docker network is a virtual network created by Docker to enable communication between Docker containers.
If two containers are running on the same host they can communicate with each other without the need for ports to be exposed to the host machine.
Network Drivers:

There are several default network drivers available in Docker.
Command to check docker networks list.
docker network ls

Types of Network Drivers:

bridge: If you build a container without specifying the kind of driver, the container will only be created in the bridge network, which is the default network.

host: Containers will not have any IP address they will be directly created in the system network which will remove isolation between the docker host and containers.

none: IP addresses won't be assigned to containers. These containments are not accessible to us from the outside or from any other container.

overlay: overlay network will enable the connection between multiple Docker demons and make different Docker swarm services communicate with each other.

ipvlan: Users have complete control over both IPv4 and IPv6 addressing by using the IPvlan driver.

macvlan: macvlan driver makes it possible to assign MAC addresses to a container.
Network Drivers

The Docker Network command is the main command that would allow you to create, manage, and configure your Docker Network. Let's see what the sub-commands can be used with the Docker Network command. to know more about Creating a Network in Docker and Connecting a Container to That Network.

sudo docker network

How to create a docker network.

sudo docker network create --driver <driver-name> <bridge-name>

Using the “Connect” command, you can connect a running Docker Container to an existing Network.

sudo docker network connect <network-name> <container-name or id>

Using the Network Inspect command, you can find out the details of a Docker Network.
You can also find the list of Containers that are connected to the Network.

sudo docker network inspect <network-name>

The disconnect command can be used to remove a Container from the Network.

sudo docker network disconnect <network-name> <container-name>

You can remove a Docker Network using the rm command.
Note that if you want to remove a network, you need to make sure that no container is currently referencing the network.

sudo docker network rm <network-name>

To remove all the unused Docker Networks, you can use the prune command.

sudo docker network prune

Kubernetes

Kubernetes:

Docker is container platform
Kubernetes is a container orchestration platform

Docker
Docker has container runtime, which will allows you to run container or manage the life cycle of the container.

Container is ephemeral - short life, containers can die or revive anytime.

Auto healing - Auto-healing refers to the capability of a system to automatically detect and recover from failures, ensuring continuous operation with minimal manual intervention.

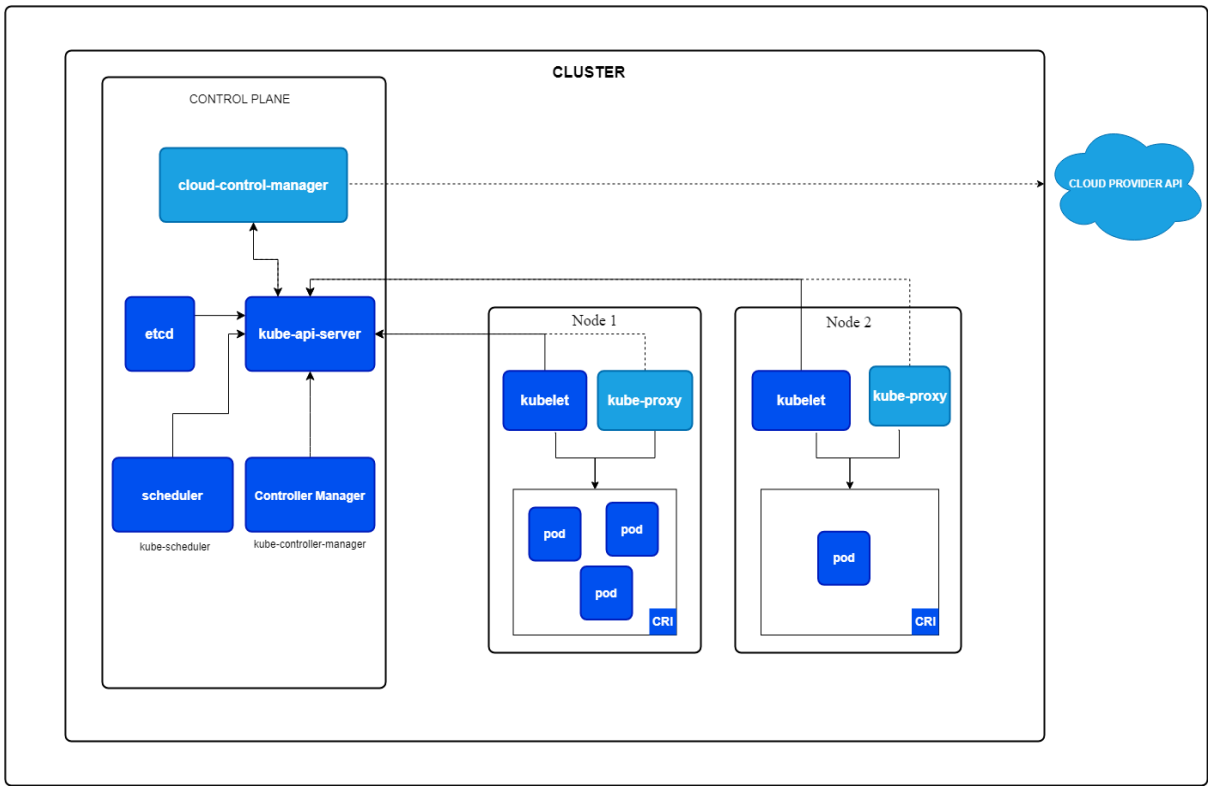
1. **Single host nature of docker container** - it might causes containers will not respond or die
2. **Auto healing** is not available in docker
3. **Auto scaling** - missing
4. **Enterprise level support**.
    - Auto scaling
    - Auto healing
    - Load balancer support
    - Firewall support
    - Api support gateways
    - White listing
    - Black listing

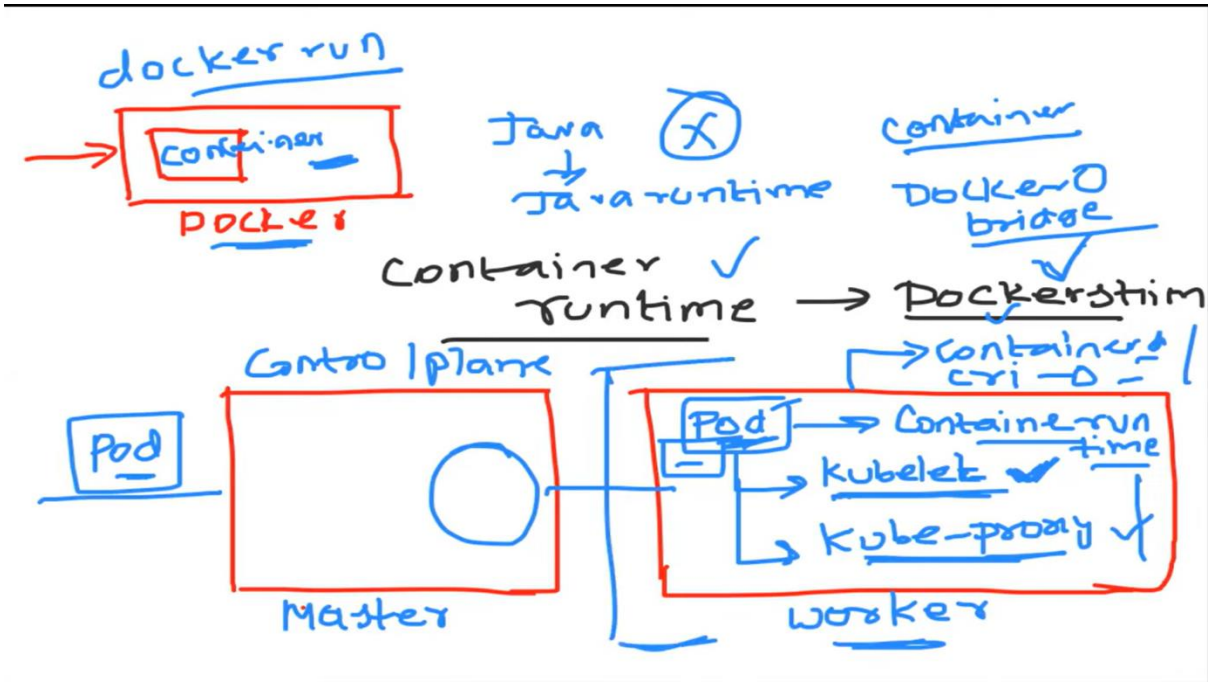
Kubernetes
Kubernetes is a cluster.
Cluster is a group of nodes.

In production Kubernetes is installed as cluster.

1. If 1 pod or container is affecting because of another node, kubernetes will put the affected node into another node.
2. Using replicas set, we can achieve the auto scaling, it can be horizontal or vertical scaling.
3. Auto healing - if one container does down, whenever api server received the signal container is going down, immediately api server will create new container or roll out the new container.
4. Enterprise level support.

Kubernetes architecture:





Node Components

Kubelet:
This is responsible for creation of pods and ensuring that pods are in running state.

An agent that runs on each node, ensuring that containers are running in a pod.

Container Runtime:
It will run the containers or pods
Manages the lifecycle of containers. Kubernetes supports container runtimes like containerd and CRI-O or docker shim (Docker runtime)

kube-proxy:
It is similar like docker default network- bridge network. It will update the IP tables rules.
Kube-proxy is basically about configureing the network rules on each of the node in cluster.

Kube-proxy works by maintaining a set of network rules on nodes, allowing network communication to your pods.
It provides networking, IP address and load balancing capabilities to pods
This component will distribute to pods

Linux has concept called IP tables.
We can configure kube-proxy in different modes, but by default one is kube-proxy updates the IP tables, so when ever somebody access the application, let's say your service is on node port mode.
So if they access the URL or if they hit the URL, node port : (colon) port number. the kube proxy because it has configured the IP tables. The request is sent from that specific node IP colon port to the pod. Okay. So this entire routing is done using the kernel and the IP tables. So you can also use ipvs and other things. But by default mode is ip tables in Kubernetes.

Control Plane Components

The brain of Kubernetes, managing the whole system.

1. **kube-apiserver**: The central management entity that exposes the Kubernetes API. It handles all REST operations to manage the cluster.
   It will exposes to external world. It will take the request from external.
   It will decided in which node pod should create.
   It will gives the information.
   For eg: user want to create a pod,
   First API server will receive the request and it will decide in which node pod should create or it will check which node is free and it inform to create a specific node.

2. **etcd**: A consistent and highly-available key-value store used for all cluster data.
   It will store the entire cluster information as objects or key value pair.

3. **kube-scheduler**: Assigns newly created pods to nodes based on resource requirements and other constraints.
   It is responsible for scheduling the resources or pods in the cluster.
   For eg: user want to create a pod,
   First API server will receive the request and it will decide in which node pod should create or it will check which node is free and it inform to create a specific node.
   Second Scheduler will do the action on specific node.

4. **kube-controller-manager**: Runs controller processes to regulate the state of the cluster, such as node and job controllers.
   To automate it has controllers.
   For eg: Replica set - it is maintaining state of pods
   Controller will ensure that the actual state in the yaml manifest is same as desired state.

   Controller-manager type:
   - Service controller
   - Pod controller
   - Deployment controller
   - Replica set controller

5. **cloud-controller-manager**: Integrates with cloud provider APIs to manage cloud-specific resources.

Kubernetes pods:

What is pod?
Definition of how to run containers.

Pod contains single container or multiple containers.

Multiple containers
1. Init container.
2. Side-car container

Consider 2 containers inside the pod.

If multiple containers are inside the pod, the kubernetes will ensure that both of the containers have the advantages

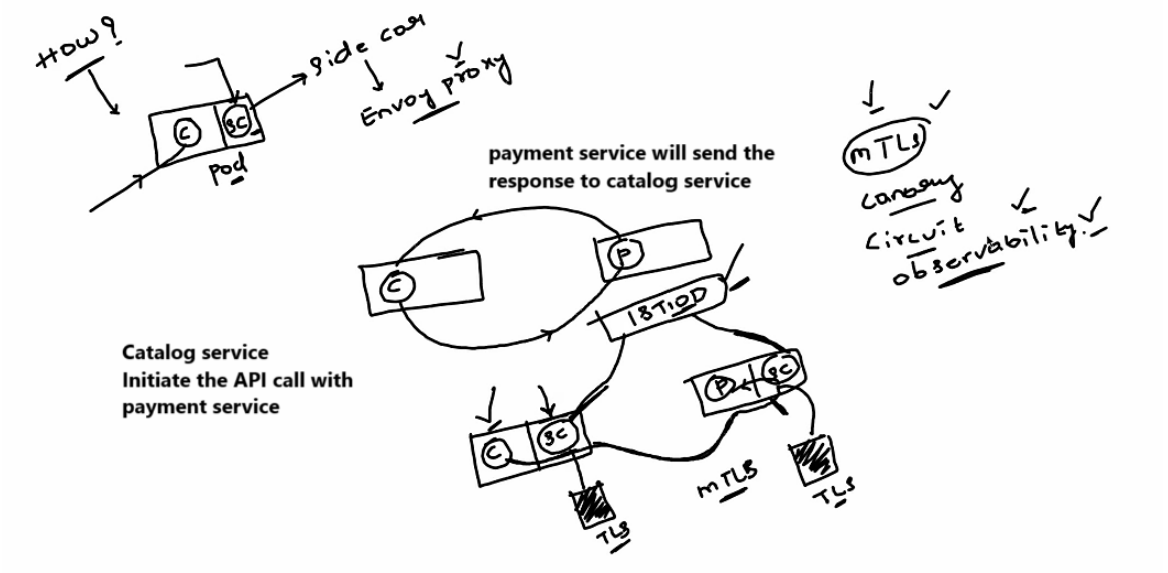
Advantages:
1. Shared the network.
2. Shared storage

With the help of cluster IP address, we can access containers inside pods.

Kube proxy will assign the cluster IP address to pod.

1. **Init container.**

2. **Side-car container**
   Side car container contains envoy proxy application. It is proxy server it will handles the traffic management of pods.



Interview:

1. How do you debug pod or applications issue in Kubernetes
   - Kubectl describe pod <pod_name>
   - Kubectl logs <pod_name>
2. Why do we need deployment?
   - Pod yaml files doesn't supports the auto healing and auto scaling features.
   - Deployment has replica set it will helps to auto healing and auto scaling features.
3. What is difference container, pod and deployment?
4. What is difference between deployment and replica set?
   - Replica set is Kubernetes controller that is the one implementing the auto scaling and auto healing feature of pods by saying the actual state in the deployment yaml manifest are the desired state in the deployment yaml manifest should be same on the cluster.
5. How to list out all the resources that are available in a particular namespace.
   - Kubectl get all
   - Kubectl get all -A (To list all namespaces.)

Kubernetes Service:

https://www.harness.io/blog/kubernetes-services-explained

An abstract way to expose an application running on a set of Pods as a network service.

Kubernetes Services are API objects that enable network exposure for one or more cluster Pods.

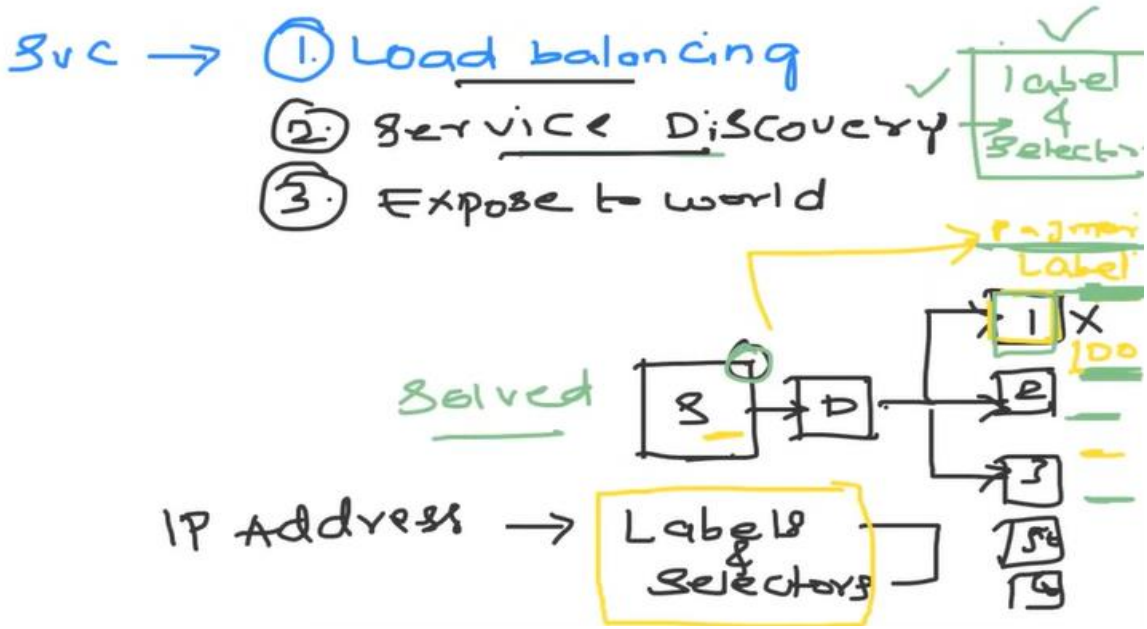
Kubernetes Services are resources that map network traffic to the Pods in your cluster. You need to create a Service each time you expose a set of Pods over the network, whether within your cluster or externally.

For each deployment yaml file we will create the service yaml file.

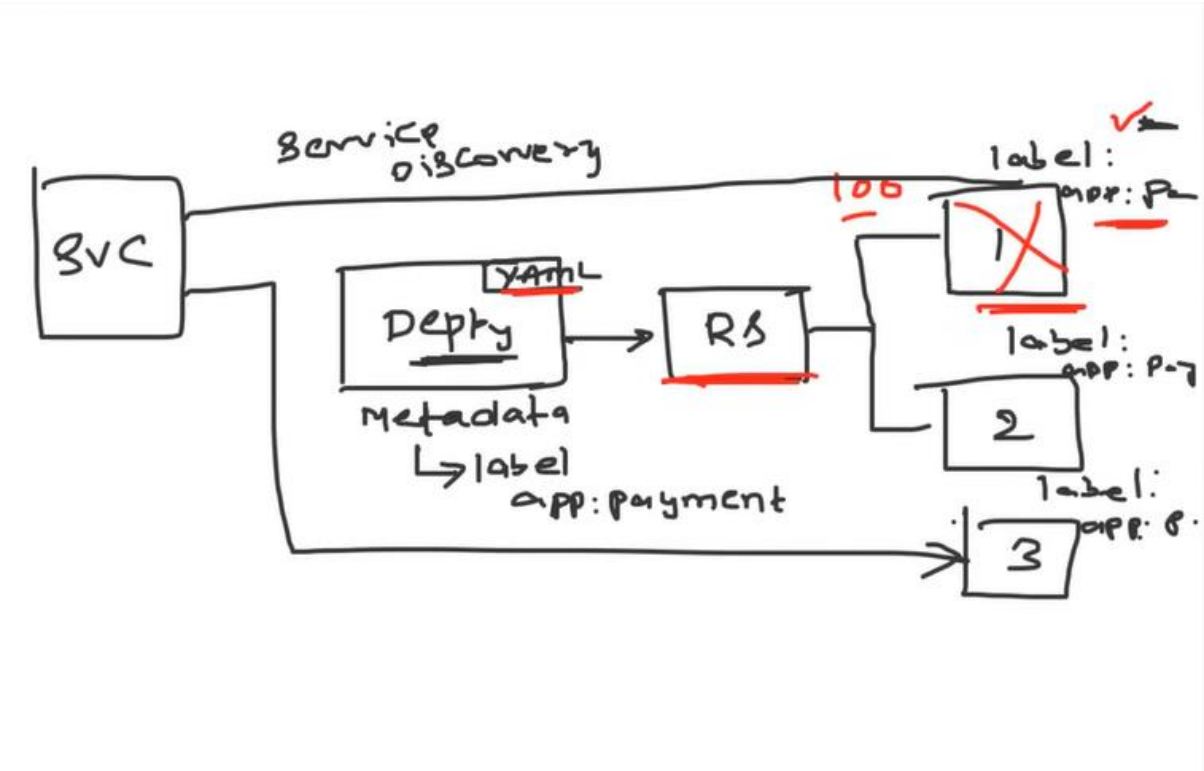
Why service is required in the kubernates?
If service is not available, due some issues, pod has deleted and created again becuaseof auto healing. Pod ip address will change automatically, if ip address changes, we will not able to access the application with old ip address. In order to overcome this problem, we will create a service yaml file, it will interact with pods using labess and selectors. If any requested came, serive file with comunication with pods using the labels and selectors.

Service responsibilities.

- Load balancing
- Service descovery machanism (using lables and selectors)
- Expose to the external world.



Service discovery



Networking in K8s

1. ClusterIP
2. NodePort Pull up for precise seeking
3. Load Balancer

ClusterIP

1. ClusterIP is the default type of service in Kubernetes.
2. This service type allows communication between different components inside the cluster.

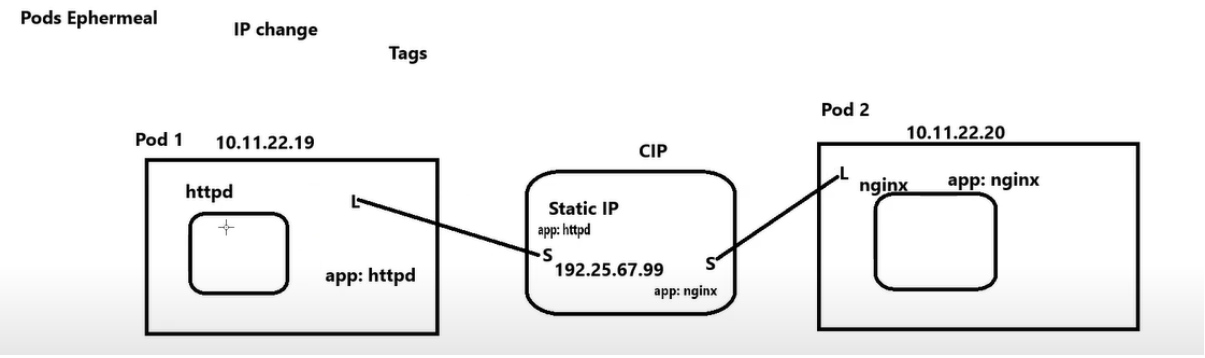
- **Internal-only:** Not accessible from outside the cluster.
- **Automatic load balancing:** ClusterIP service automatically balances traffic between the pods it manages.
- **Default service type:** If no other type is specified, Kubernetes creates a ClusterIP service.

ClusterIP Use Cases

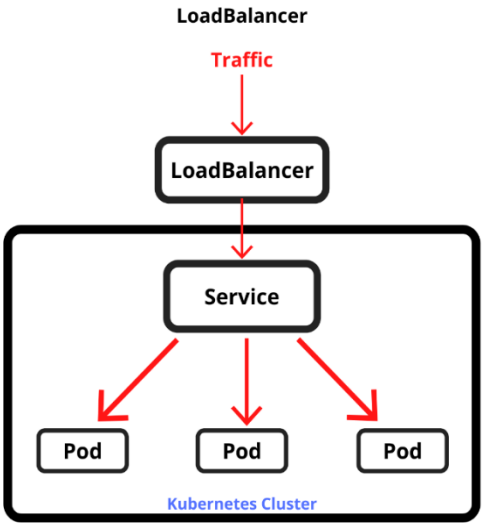
- Internal Communication Between Microservices
- **Example**: In an e-commerce application, a front-end service might need to communicate with a back-end API, and they can do so securely using ClusterIP.
- Database Access Within the Cluster
- Communication Between Pods in Different Namespaces

How ClusterIP works

Only internal, not external
For external we have NodePort and Load Balancer



LoadBalancer:



LoadBalancer is typically used in cloudenvironments to expose a serviceexternally

- LoadBalancer automatically provisions the external address to access your services outside the cluster.
- It works on AWS, GCP and Azure where you provision as external LoadBalancer
- On-premises may need provision manually.

LoadBalancer Use-cases in K8s

- Expose your applications to the public internet.
- Traffic distribution across Nodes and Pods

How Load Balancer works in K8s?

- Configure LoadBalancer as Service
- Distribute traffic to external IP

Eg:

```
apiVersion: v1
kind: Service
metadata:
  name: my-loadbalancer-service
spec:
  type: LoadBalancer
  selector:
    app: my-app
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
```

Advanatages of LoadBalancer:

- **Automatic provisioning:** Simplifies exposing services externally.
- **Traffic distribution:** Automatically balances traffic across pods.
- **Managed service:** Especially useful in cloud environments where you don't need to manually configure or maintain the load balancer.

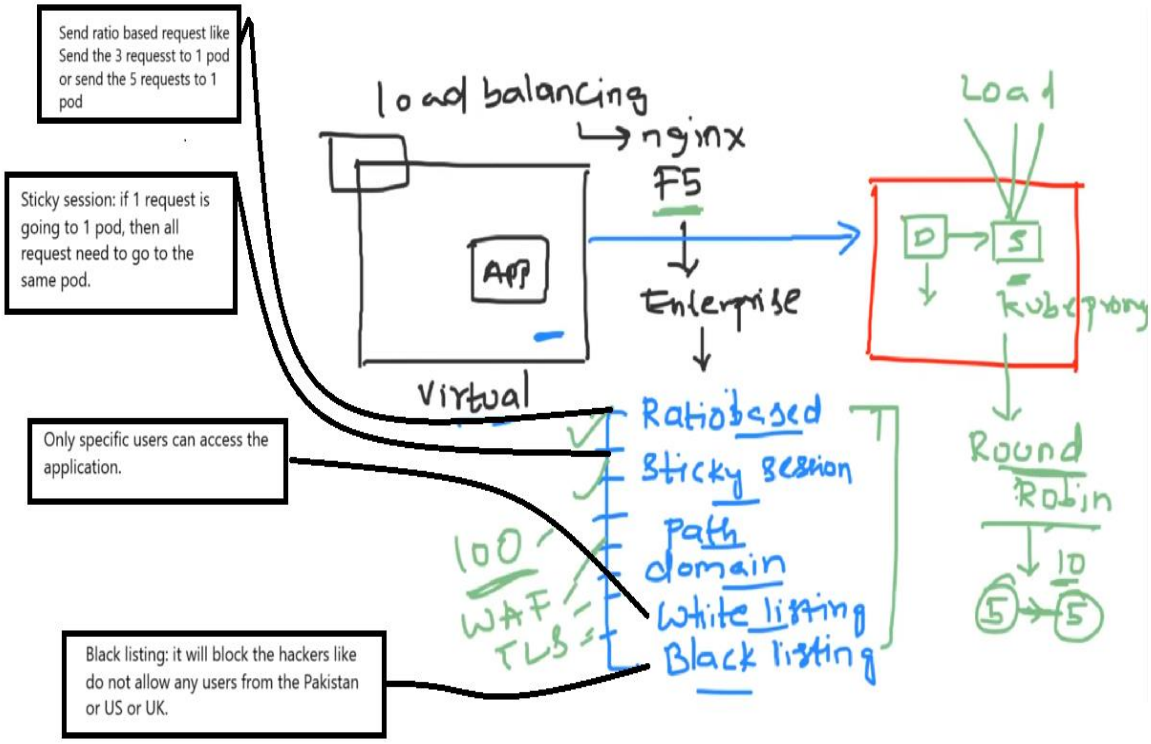
Ingress, Routes and Ingress Controllers:

Load Balancer Type Service

Q: Is Load Balancer service type only restricted to Cloud providers ?
A: Bare Metal LB Implementation - https://github.com/metallb/metallb
Q: If Load Balancer service type can do the thing for you, why use an Ingress resource?
A. 1. If you have the large-scale ecommerce application, consider it has 200 to 300 services, If you want to user cloud provider load balancer, then you have to create 200 to 300 static external IP addresses, which will cloud provider will charge huge amount. To overcome you can create ingress resource, it will manage all services with one static external IP address.
2. It defines routing for specific service using host base routing and path-based routing, session-based routing or cookies-based routing etc.

Before 2015, in K8s v1.1, it provides only basic load balancer support. Ingress concept is not there.
Loadbalancing mechanism is providing simple round robin mechanism, If 10 request come, if you have 2 pods, then round robin mechanism distribute it equal to 2 to pods. 5 request has to go to 1 pod and another request has to go to another pod.

In Virtual machines



Disadvantages of loadbalancer.

1. Enterprise level support and TLS Loadbalancer capabilities.
   - Ratio Based routing, Sticky, path, domain, writing listing and black listing based routing
2. Loadbalancer static IP address will be charged by cloud provider. if we created more loadbalancer static IP address

What is a Kubernetes Ingress?

Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. Traffic routing is controlled by rules defined on the Ingress resource.

Once install the ingress controller on Kubernetes cluster, it will continuously watch for ingress resources (nginx.conf, the place where nginx.conf updates all its routing details) on this specific cluster. Along with your ingress controller nginx applications is also deployed.

Ingress:

- **Ingress** as a resource that manages external HTTP(S) access to services within a Kubernetes cluster
- Unlike NodePort or LoadBalancer services, Ingress offers more advanced features like path-based or host-based routing.
- Expose multiple services using a single external IP or domain
- Loadbalancer + API Gateway

Why do we use Ingress?

- Expose multiple microservices with domains and subdomains
- Centralized routing
- So ingress is best compared to LoadBalancer and NodePort

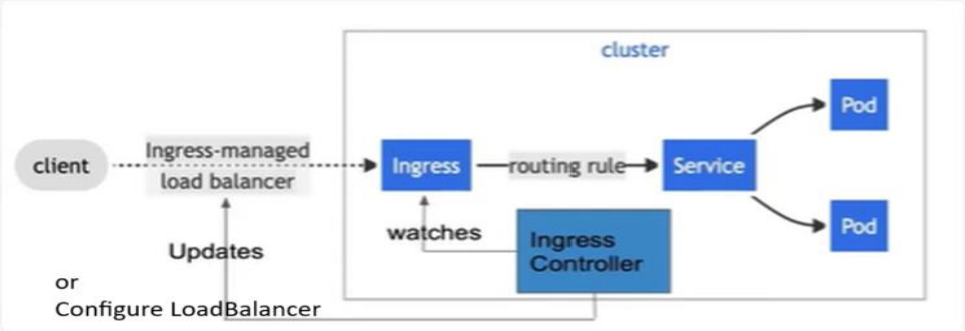
How ingress works?

- **The ingress controller** is responsible for routing traffic based on defined rules.
- Routes external traffic to the correct internal service based on hostnames or paths.

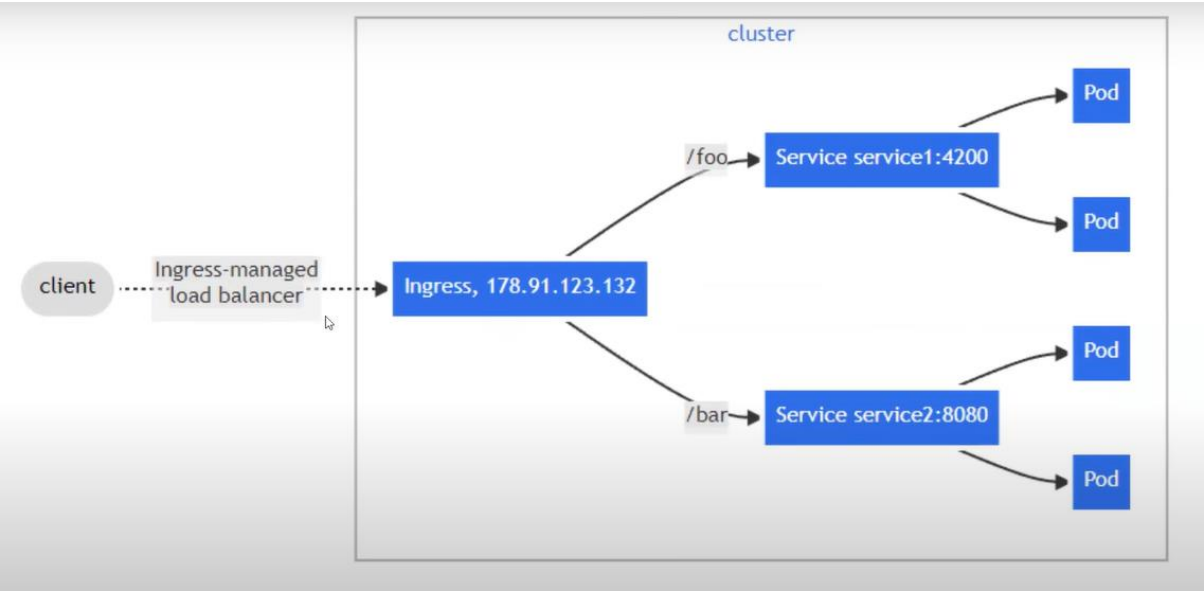
- NGINX Ingress Controller, Traefik, and AWS ALB Ingress

Ingress Controller

In order for the Ingress resource to work, the cluster must have an ingress controller running.



.



Reference:
https://raw.githubusercontent.com/kubernetes/website/main/content/en/examples/service/networking/test-i ngress.yaml

Sample Ingress

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: test-ingress
spec:
  defaultBackend:
    service:
      name: test
      port:
        number: 80
```

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-no-auth
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: http-svc
            port:
              number: 80
~
~
```

Host Based Routing

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-with-auth
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: http-svc
            port:
              number: 80
  - host: example.bar.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: meow-svc
            port:
              number: 80
```

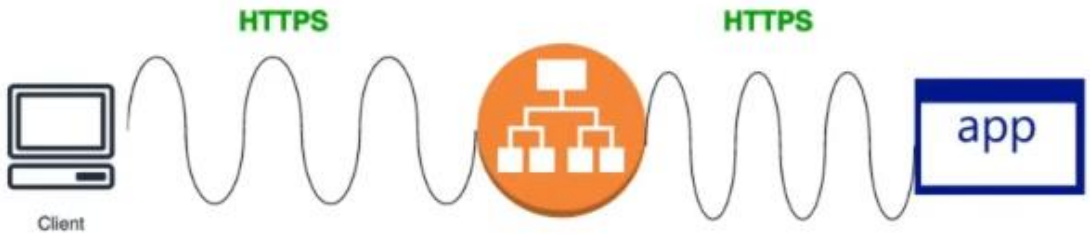
Path Based Routing

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-with-auth
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /first
        pathType: Prefix
        backend:
          service:
            name: http-svc
            port:
              number: 80
      paths:
      - path: /second
        pathType: Prefix
        backend:
          service:
            name: meow-svc
            port:
              number: 80
```

TLS:

SSL Passthrough

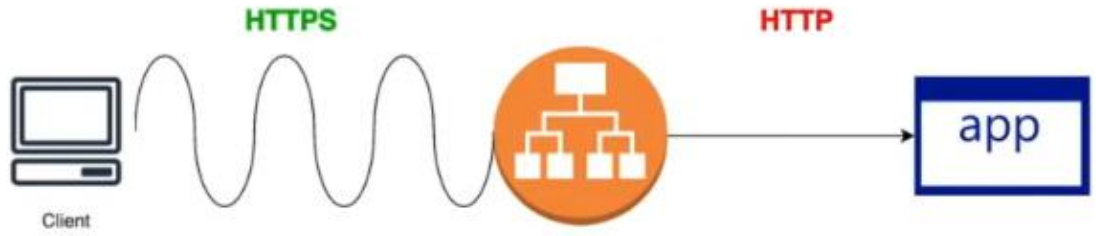
SSL passthrough passes encrypted HTTPS traffic directly to the backend servers without decrypting the traffics at the load balancer.



Load Balancer capabilities are merely used.
Attacker can pass hacking codes in the traffic and will be directly passed to the backend server.
SSL Passthrough is also a costly process. Might require more CPU.

SSL Offloading

SSL Offloading (SSL Termination) decrypts all HTTPS traffics when it arrives at the load balancer and the data is sent to the destination server as plain HTTP traffic.



Vulnerable to data theft, man-in-the-middle attacks.
Faster

SSL Bridging

SSL Bridging decrypts all HTTPS traffics when it arrives at the load balancer and the data is sent to the destination server as HTTPS traffic by re-encrypting.

E2E encrypted and validated for malware attacks by Load Balancer.
More secure more costly in processing as server has to decrypt the traffic.

Pros and Cons !!

SSL Passthrough	SSL-Offloading/Termination	SSL-Bridge/Re-encrypt
Costly in processing for Server	Fast in processing	Costly in processing for Server
Secure in many cases	Insecure	Secure
L4 (TCP) Load Balancing	L7 Load Balancing	L7 Balancing
Choose when you don't bother about access rules, blocking, cookie e.t.c.,.	When you want less latency and can compromise on security.	When you need security and advanced load balancer capabilities.
No Load Balancer Inspection.	Load Balancer Inspects the packets.	Load Balancer Inspects the packets.
Recommended*	Highly unrecommended.	Recommended

OpenShift Routes

SSL Offloading == Edge Termination

SSL Bridge == Re-encrypt Termination

SSL Passthrough == Passthrough Termination

```yaml
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: hello-openshift
spec:
  host: hello-openshift-hello-openshift.
  port:
    targetPort: 8080
  to:
    kind: Service
    name: hello-openshift
```

Secure Routes

*Routes does not support storing the TLS certs in secrets - Issue

Routes are simple, you cannot add multiple services, paths or hosts in a single route.

```yaml
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: route-passthrough-secured (1)
spec:
  host: www.example.com
  port:
    targetPort: 8080
  tls:
    termination: passthrough (2)
    insecureEdgeTerminationPolicy: None
  to:
    kind: Service
    name: frontend
```

```yaml
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: frontend
spec:
  host: www.example.com
  to:
    kind: Service
    name: frontend
  tls:
    termination: edge
    key: |-
      -----BEGIN PRIVATE KEY-----
      [...]
      -----END PRIVATE KEY-----
    certificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
    caCertificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
```

```yaml
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: frontend
spec:
  host: www.example.com
  to:
    kind: Service
    name: frontend
  tls:
    termination: reencrypt
    key: |-
      -----BEGIN PRIVATE KEY-----
      [...]
      -----END PRIVATE KEY-----
    certificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
    caCertificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
    destinationCACertificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
```

Service mesh

Service mesh
It will help to manage the traffic between services in cluster. Mainly (East - West)

Why service mesh needs?

- Service mesh will enhance the capability to service to service in cluster. **MTLS (Mutual TLS)**
- It will also add the advanced capabilities such as **deployment strategies** (Canary / A-B / Blue Green)
- **Observabilities** – Kiali – it will keeps a track of service to service communication information. It will helps to understand how services are behaving and metrics health of services.
- **Circuit breaking** – it can helps to split traffic splitting

Traffic Management

Tasks that demonstrate Istio's traffic routing features.

Request Routing
This task shows you how to configure dynamic request routing to multiple versions of a microservice.

Fault Injection
This task shows you how to inject faults to test the resiliency of your application.

Traffi
Shows
new ve

TCP Traffic Shifting
Shows you how to migrate TCP traffic from an old to new version of a TCP service.

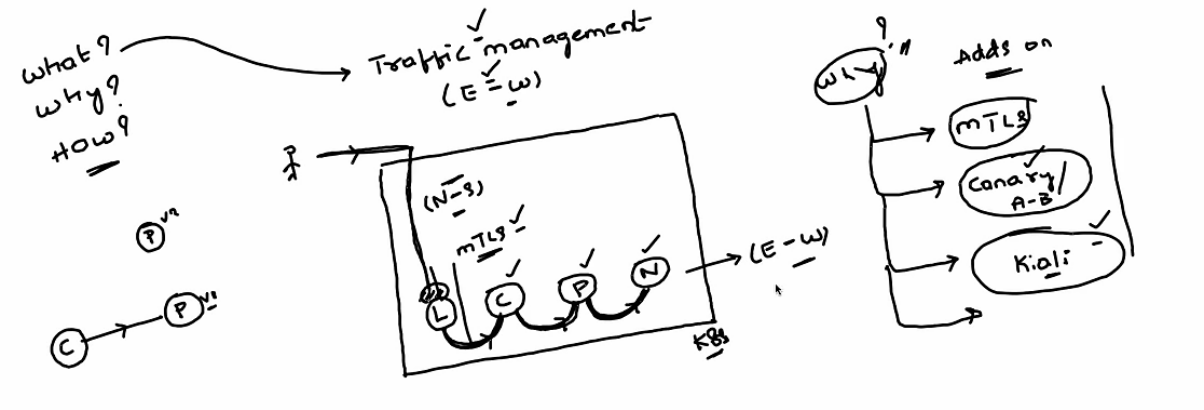
Request Timeouts
This task shows you how to set up request timeouts in Envoy using Istio.

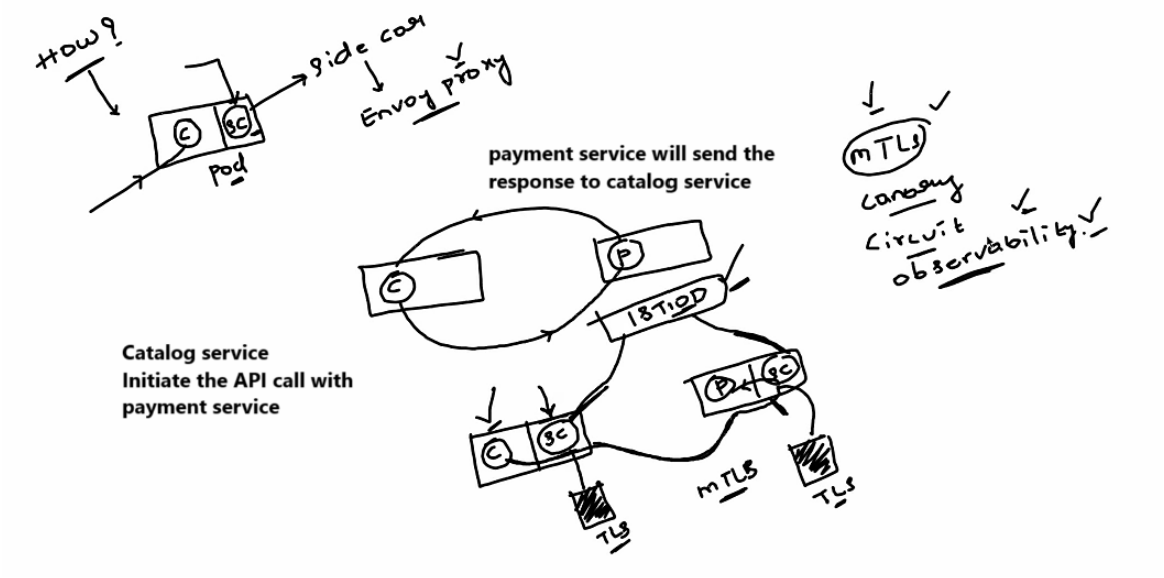
Circui
This ta
breaki
detect

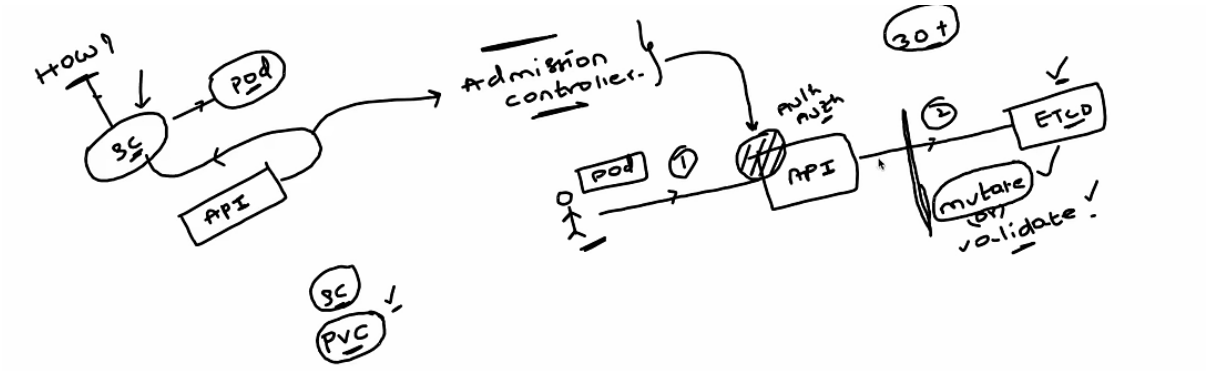
Mirroring
This task demonstrates the traffic mirroring/shadowing capabilities of Istio.

Locality Load Balancing
This series of tasks demonstrate how to configure locality load balancing in Istio.

Ingres
Contro
mesh.







Admission controller

It will validate and mutate the authenticated and authenticated user request

- Mutate
- Validate

Role Based Access Control (RBAC)

What is RBAC?

- **2 Primary responsibilities of RBAC is User management as well as managing the access of the services that are running in the cluster.**

To Manage RBAC in Production we have 3 major things in Kubernetes.

1. **Service account / User**
2. **Kubernetes roles / cluster roles**
3. **Role Binding / cluster role binding.**

How to create users in cluster?

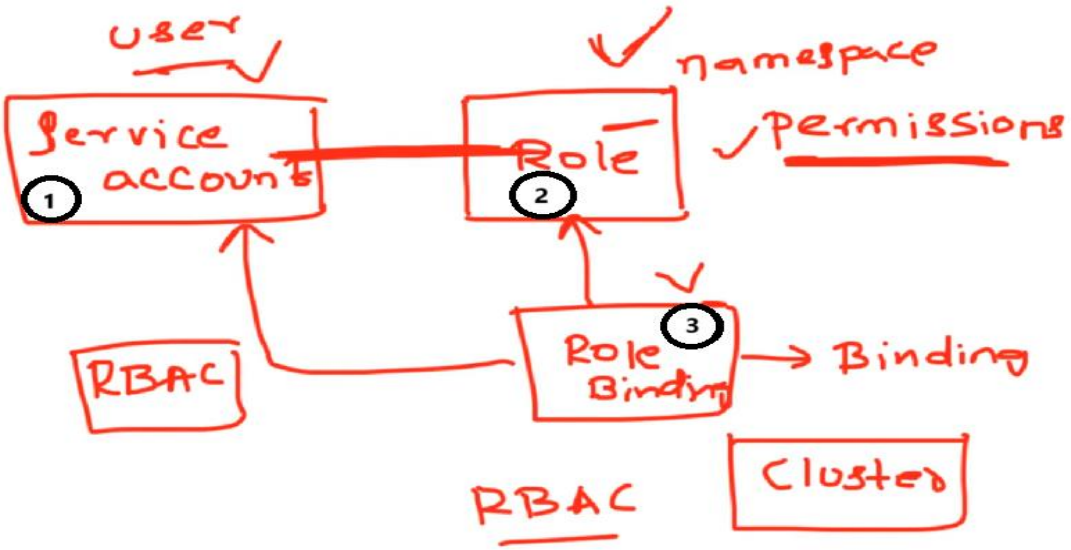
Kubernetes will not manage the user accounts or users. It will offload the user management to Identity providers.
Kubernetes API server works as Oauth server, IAM Oauth Identity provider
What is Oauth server?

Once login into K8s cluster as user after creating the service account and user. How do you manage the rules or configure?
To define access kubernetes supports 2 parts.

1. **Kubernetes roles / cluster roles**
2. **Role Binding / cluster role binding.**

Once application is running as service account or you logged in as user, then the next part is how to grant the access to it?
A. First create the role and assign the role to the users as they should have access to pods, config maps, secretes within the same namespace. Or if you want to access the across the cluster, then create the cluster role. Now we have to attach this role to user using role binding.

What is the difference be role and cluster role and role binding and cluster role binding?



Kubernetes Custom Resource Definitions, Custom Resources and Custom Resources Controller.

Custom Resource Definitions (CRD)

It will extend the capabilities of kubernetes or API of Kubernetes
We can introduce new resources or extend the API of kubernetes to introduce new resource in this case we are using **CRD, CR, and CRC.**

1. Devops Engineers: Responsible to deploy **CRD and CRC. CR (Optional)**
2. Users: Deploy **Custom Resources**

CNCF is all about the Kubernetes controllers like custom Kubernetes controllers.

There is different resource are giving the multiple advanced capabilities to kubernetes cluster.
ISTIO which adds the service mesh capabilities to cluster.
ArgoCD – Gitops
Keyclock – it will provide strong identity and access management Oauth or OIDC capabilities to cluster
Security..... etc,.

To extend the capabilities of kubernetes or API of Kubernetes, there are 3 Resources are there

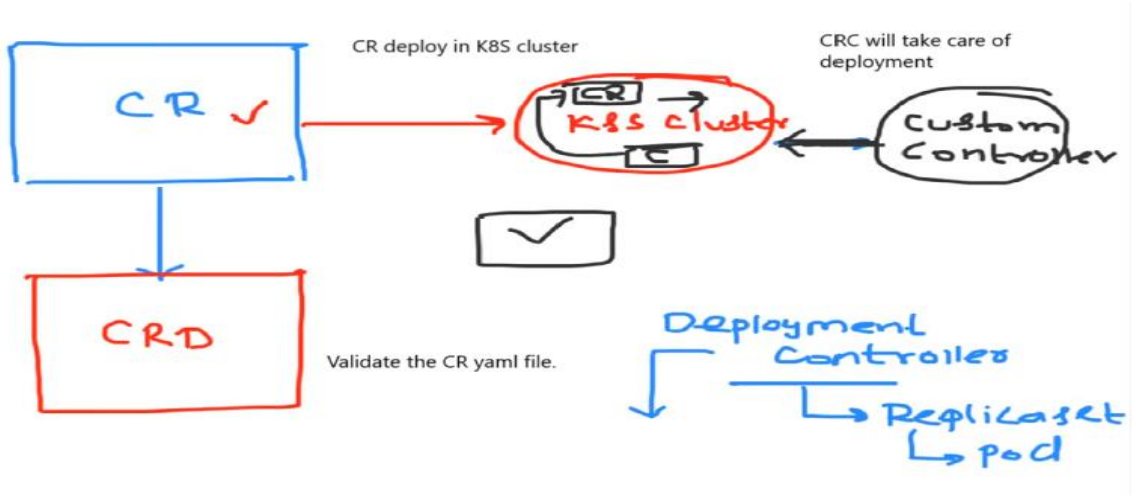
1. **Custom Resource Definitions**
2. **Custom Resources**
3. **Custom Resources Controller.**

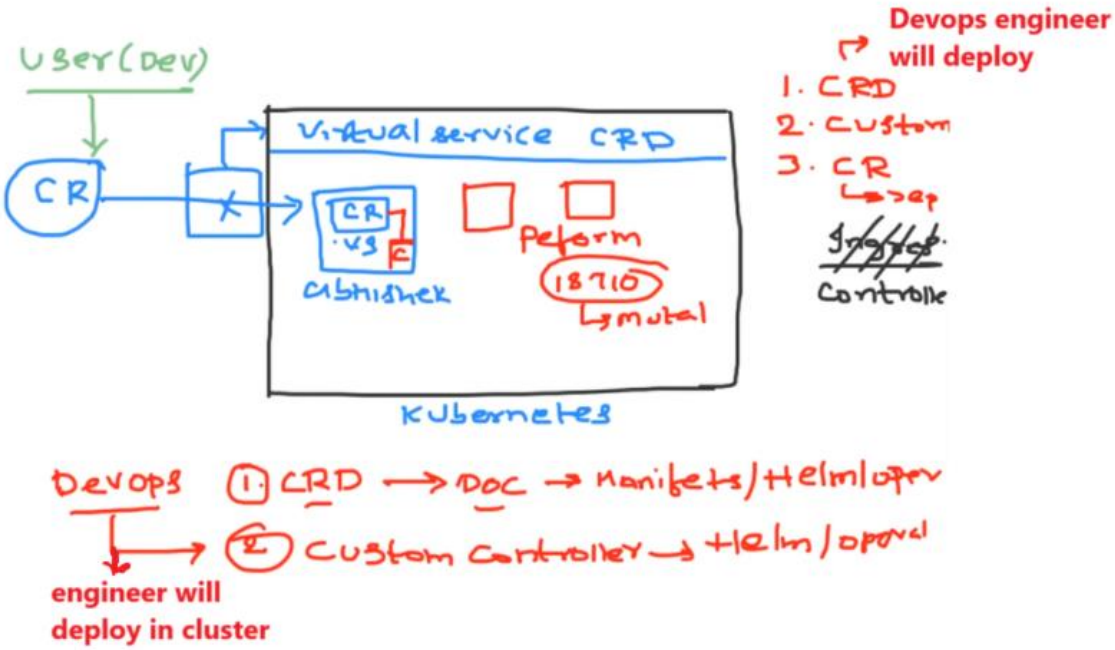
1. Custom Resource Definitions

If a company wants to enhance the capability of kubernetes**,** In order to do this we will create CRD. CRD means defining a new type of API to Kubernetes.

Submit the custom resource definition to Kubernetes.

Custom Resources
If user create the CR yml file, CRD will validate the CR file if it is correct or not.





ConfigMaps and Secretes

ConfigMaps:
It is used store the data or information.

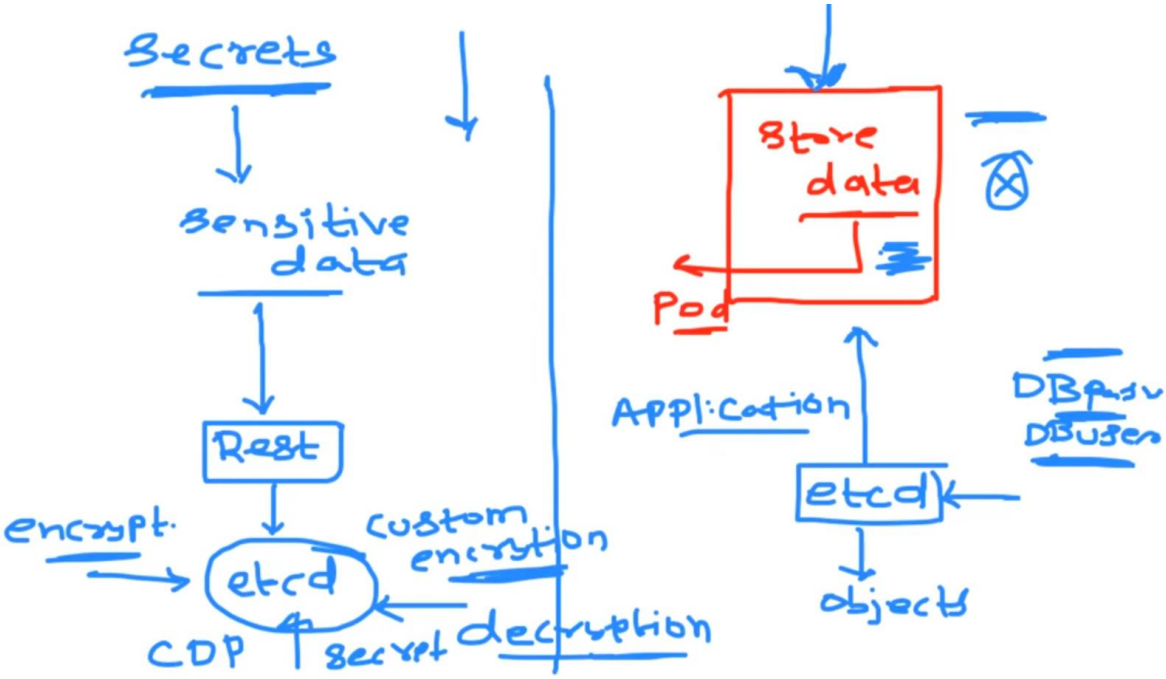
In K8s whenever create the recourse, this information gets saved in the etcd as a object.
If we store the password or sensitive information in ConfigMaps, there are high changes to get the entire application hack.

Secretes:
It is used to store the sensitive data or information.

Advantages

1. Encryption the data
2. Secrete – RBAC – we can provide the least privileges to secrets file.

If we saved the sensitive data in Secrete file, it will encrypt the data at the rest before the object is saved in etcd.
By default, kubernetes uses a basic encryption. However, it will also allow to use our own encryption mechanism like the custom encryption. It ensures that no security
To decrypt the data, it required the decryption key. Without decryption key it is not possible to get the retrieve the data.



Interview Question:

1. **How do you use the configMap inside your Kubernetes pod?**

A. There 2 ways to use the configMap inside pods.

1. Environment variables
2. Volume mounts.

We can mount the configmap file or we can use the details of config map inside kubernetes pods. The information can be sorted inside pod as environment variable or can be stored as a file inside pod on container file system but this information has to be retrieved.

If you want to change the DB port number 3306 to 3308 in ConfigMap, you can login into pod and then change the port number in ConfigMap, But the application will run on 3306 port number only, it will fails due to port number changed.

Deployment file will not understand as DB port number has changed and it needs to be upgrade or run the deployment file once again.

if the information is keeps changing, the changing environment variables is not possible inside containers or kubernetes. you can't update environment variable values because container does not allow changing the environment variable.

You have to recreate the container but, in the production, you cannot restart the containers because it might lead to some traffic loss if you are deleting the deployment and recreating the deployment you might incur some traffic loss which is not expected in production environment.

In this case Kubernetes has solution,
if the information is keeps changing, there is concept called **volume mounts**.

Using the volume mounts we can do the same thing but instead of using them as environment variables. The ConfigMap information will be saved inside a file. Whenever update and apply the ConfigMap file, without restarting the pod, port number will automatically change.

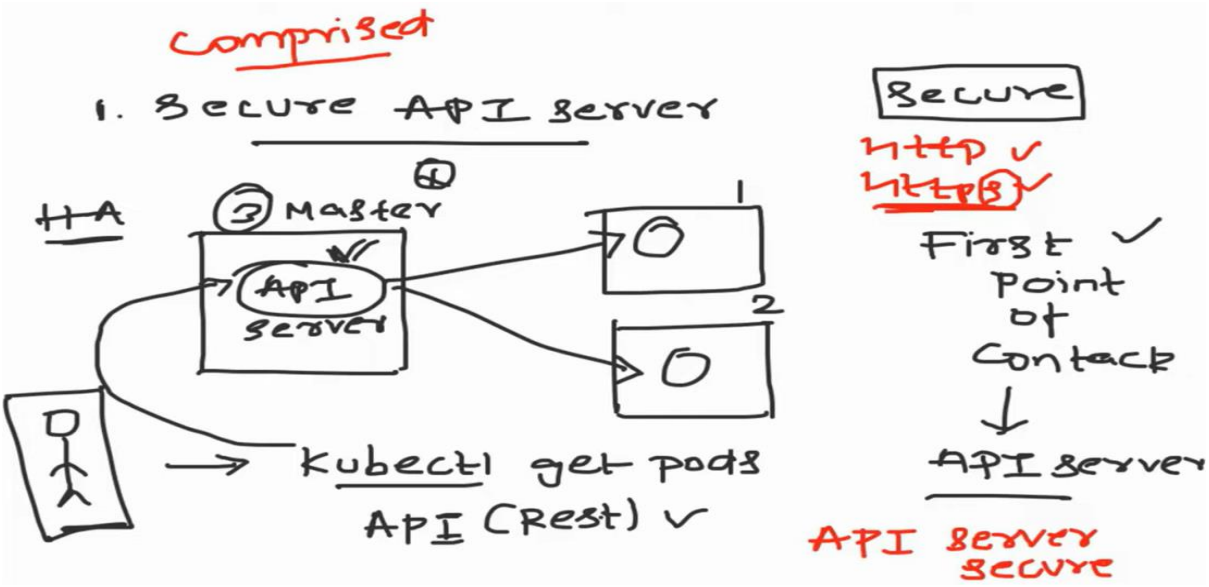
2. What is the deference between ConfigMaps and Secretes?
A.

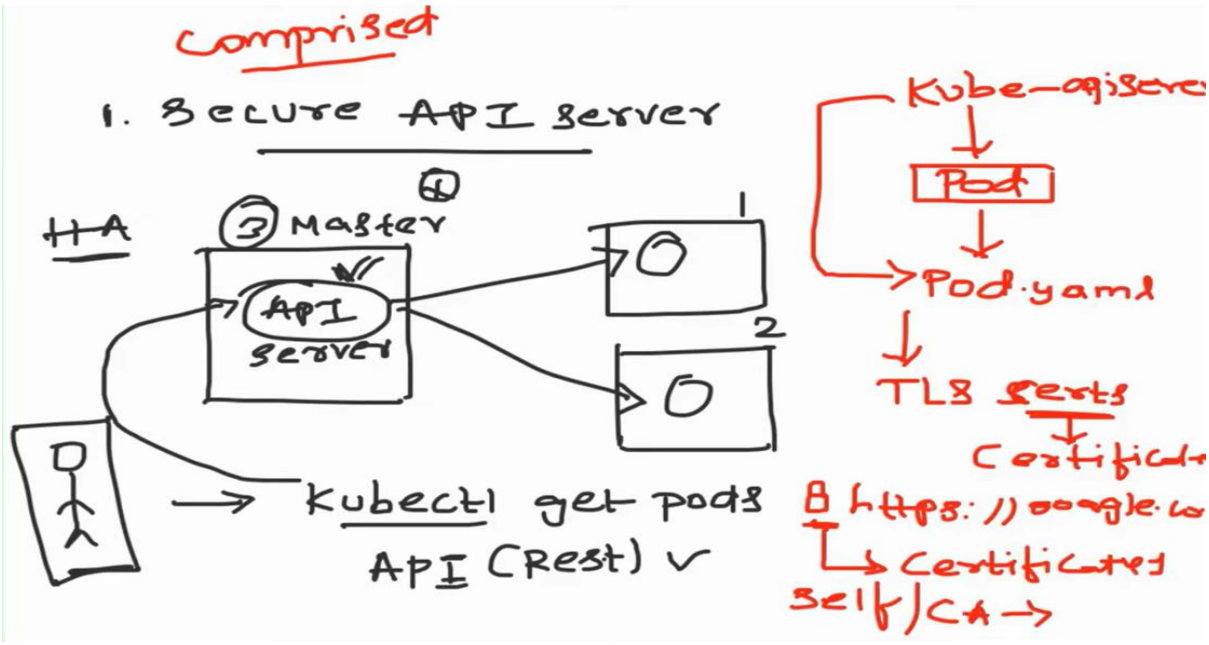
SECURE KUBERNETES

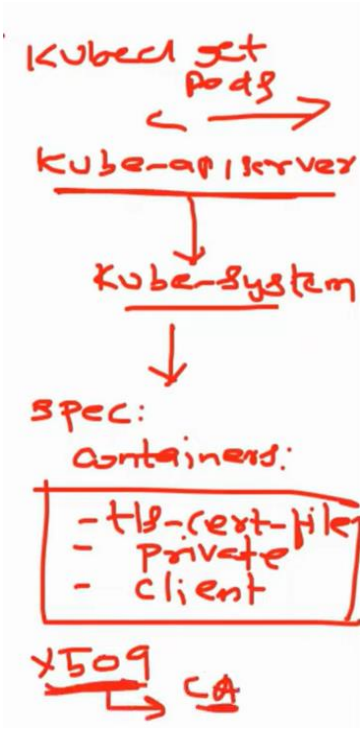
1. Secure API server
2. RBAC
3. Network policies
4. Encrypted at rest (ETCD)
5. Secure Container Images
6. Cluster monitoring
7. upgrades.

1. Secure API server

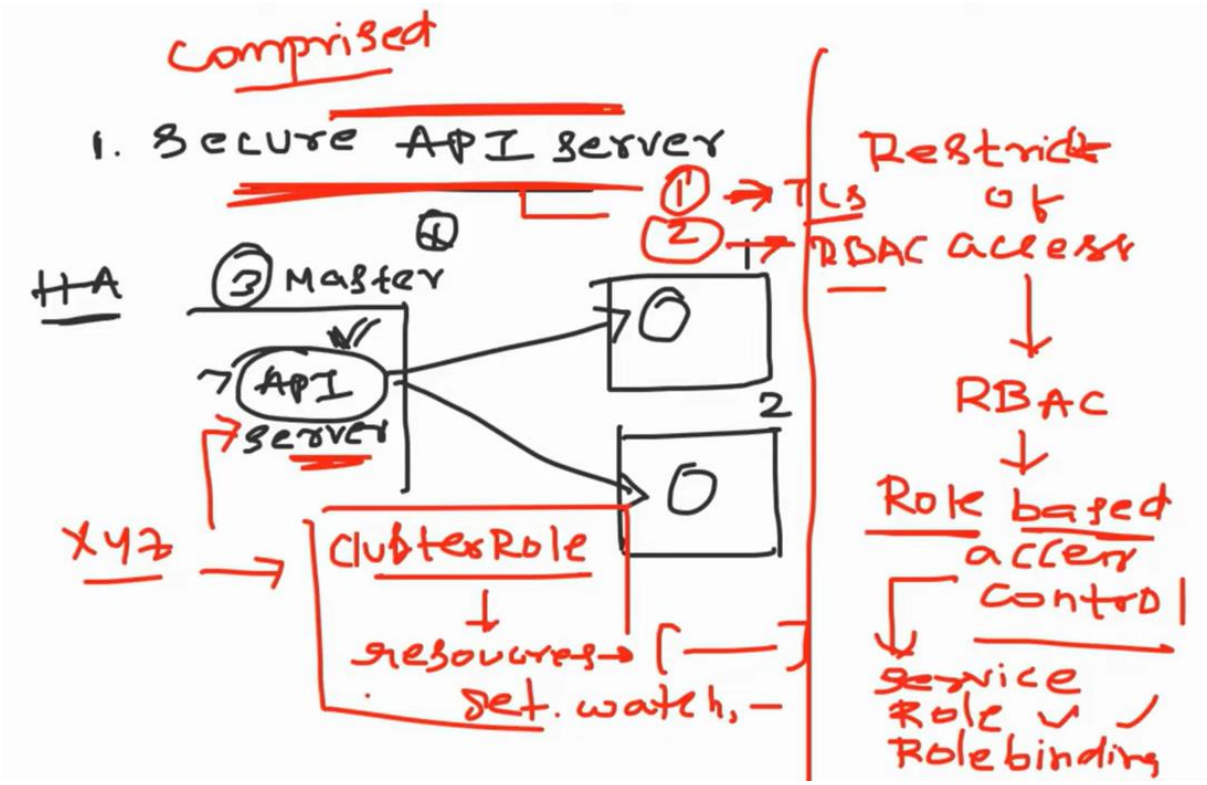
- Secure API server
- How to secure the API server?
  There are 2 methods
  1. Add the certificate in to the Kube-api server
  2. Restrict the access. (RBAC).


Compriseed
1. Secure API server
HA
3 Master
API server
1
2
Kubectl get pods
API (Rest) ✓
Secure
http ✓
https ✓
First ✓ Point of Contact
API server
API server secure


Compriseed
1. Secure API server
HA
3 Master
API server
1
2
Kubectl get pods
API (Rest) ✓
Kube-apiserver
Pod
Pod.yaml
TLS certs
Certificate
https://google.com
Certificate
self/CA

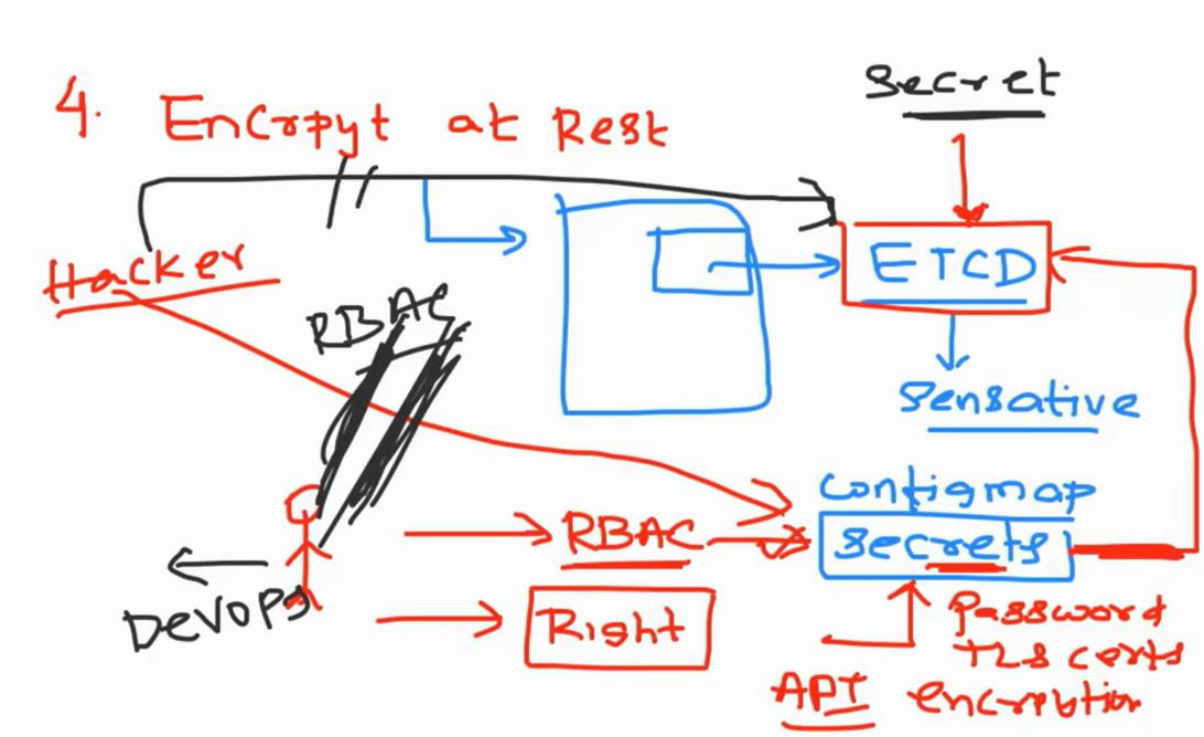


2. Role Based Access Control (RBAC)

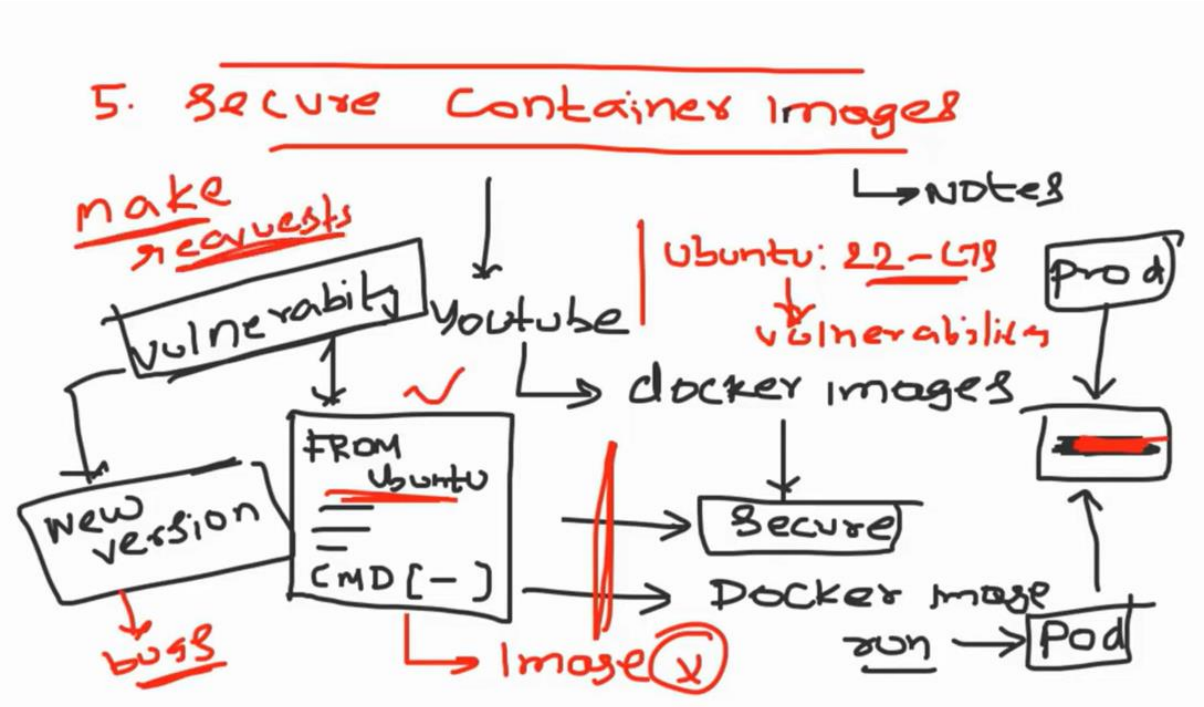


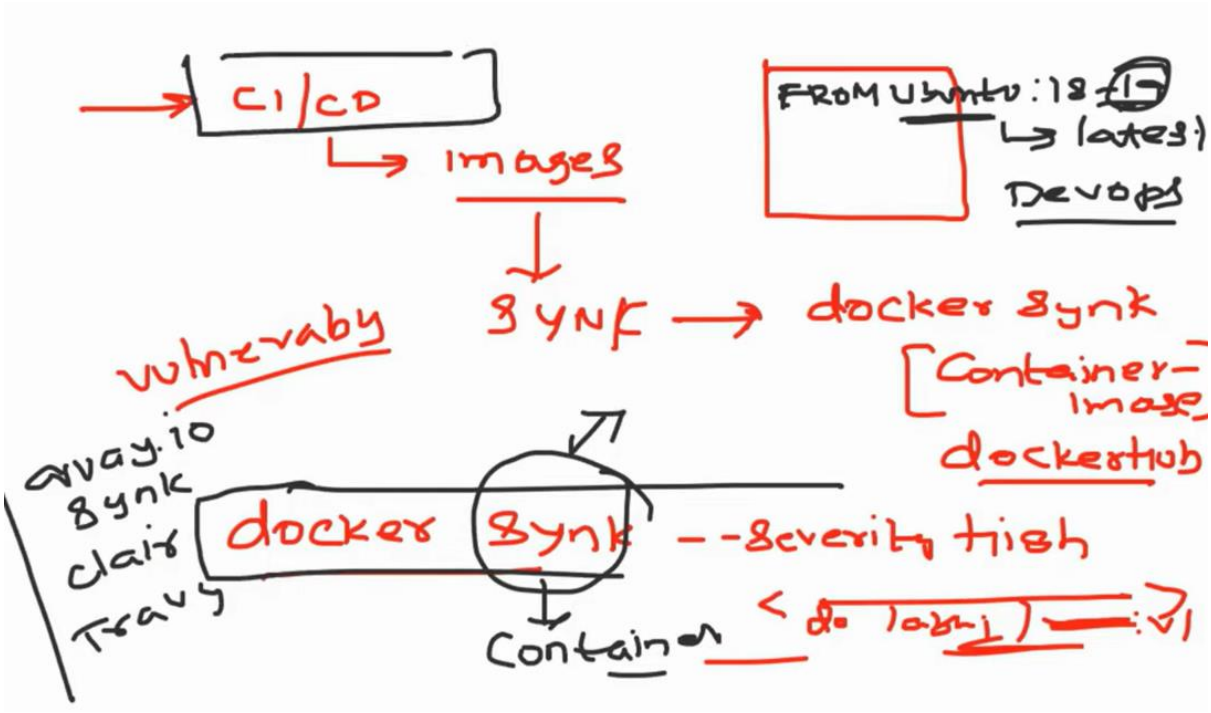
3. Network policies

4. Encrypted at rest (ETCD)

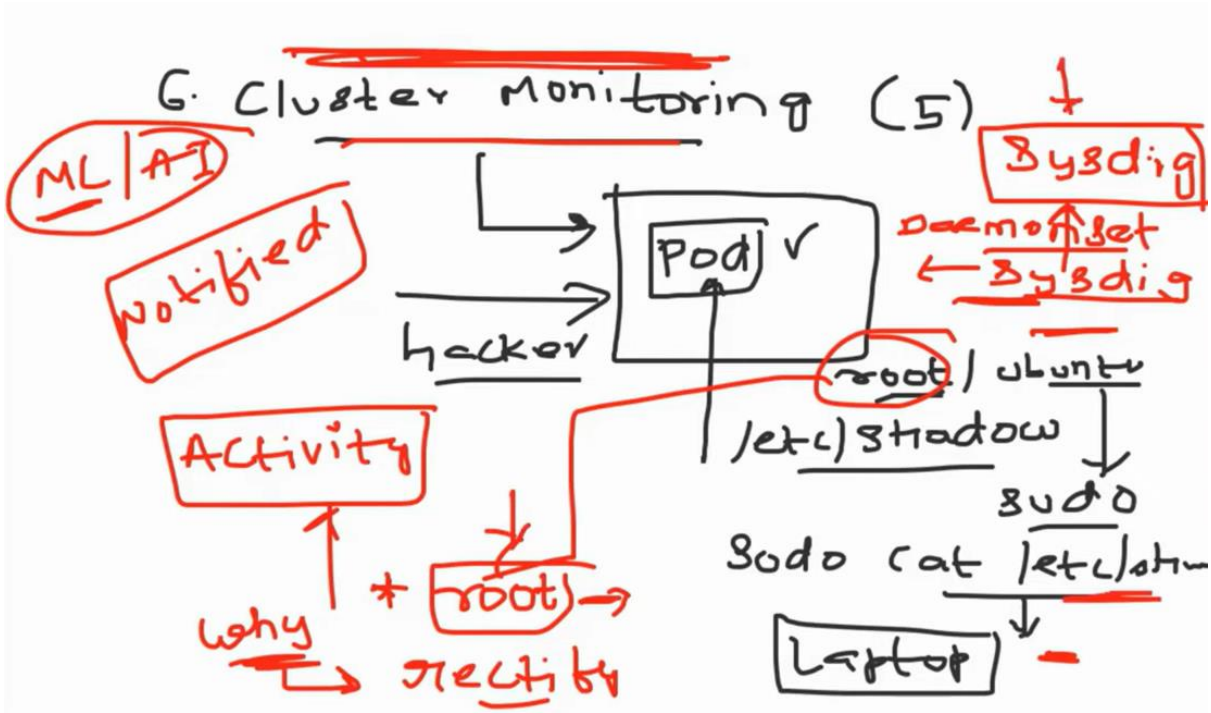


5. Secure Container Images





6. Cluster monitoring



7. upgrades.

Kubernetes Monitoring

Prometheus and Grafana

Prometheus:

Kube state metrics
Kubernetes API server exposes few metrics of kubernetes. It gives the information about the Kubernetes API server, default installations,

Kube state metric controller.

We can create a service for kube state metric and use this kube state metrics when you call the kube state metrics on the metric end point, so it will give you the lots of the information about the existing kubernetes cluster. This is information beyond the information kubernetes API server is providing.

AWS Elastic Kubernetes Service (EKS cluster)

OpenID connect provider URL.
We can integrate any identity providers.

Deployment Strategies in Kubernetes

Kubernetes offers several deployment strategies to manage how updates are applied to your applications. Here are some common ones:

1. **Recreate Deployment**: This strategy terminates all existing pods and replaces them with new ones. It can cause downtime but ensures a clean state
2. **Rolling Deployment**: This is the default strategy in Kubernetes. It gradually replaces old pods with new ones without downtime, ensuring that the application remains available

3. **Blue/Green Deployment**: This strategy involves running two environments, one with the current version (blue) and one with the new version (green). Traffic is switched to the green environment once it's verified to be stable
4. **Canary Deployment**: This strategy releases the new version to a small subset of users before rolling it out to the entire user base. It allows for testing in a real-world environment with minimal risk
5. **A/B Testing**: Similar to canary deployments, but it involves running two versions simultaneously to compare their performance and user acceptance
6. **Ramped Deployment**: This is a slow rollout where the new version is gradually introduced while monitoring its performance
7. **Shadow Deployment**: The new version receives a copy of the live traffic without affecting the actual user experience. This helps in testing the new version under real-world conditions

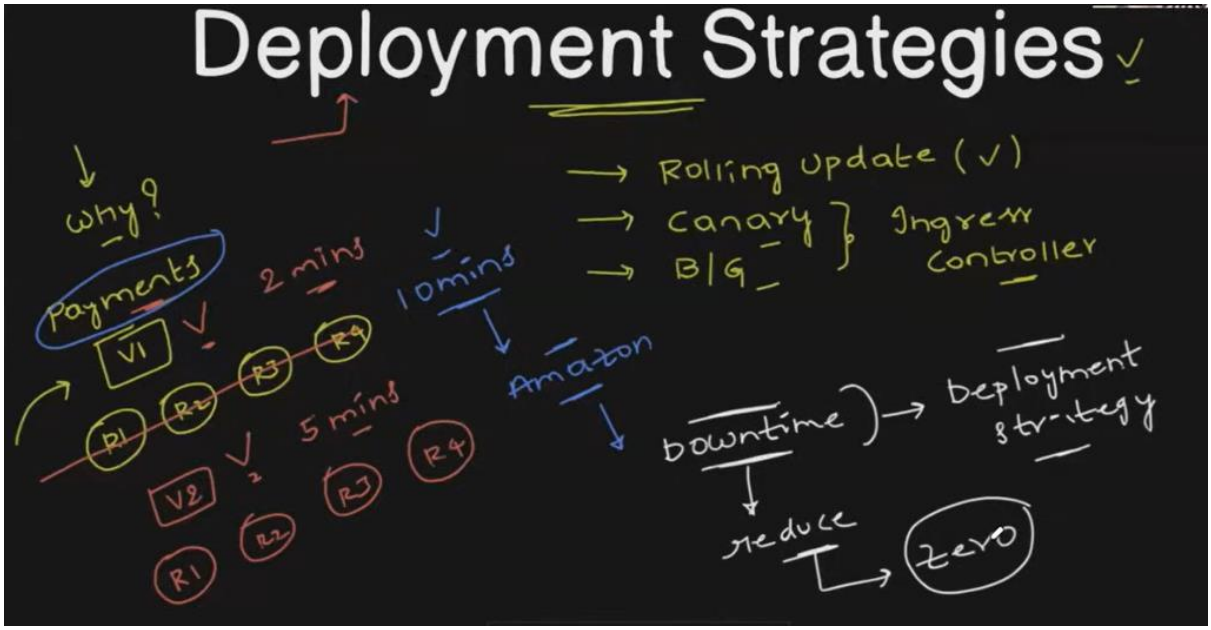
Each strategy has its advantages and use cases, depending on your application's requirements and the level of risk you're willing to take during updates.

Rolling Update (Default strategy)
Canary Deployment
Blue Green Deployment

Canary and Blue Green Deployment is not supported out of the Kubernetes box. We will set up ingress controller on kubernetes cluster and using this ingress controller we will configure Canary and Blue Green Deployment.

Why do you need Deployment strategy?
Consider we have 4 pods of payment application V1 version and if you want to upgrade the payment application pods to V2 version, in order to do this, we have to delete and create new pods with V2 version. These a downtime while upgrading the versions. It caused revenue loss, organization reputation and User experience.

Primary purpose is to reduce the downtime to Zero using deployment strategy.



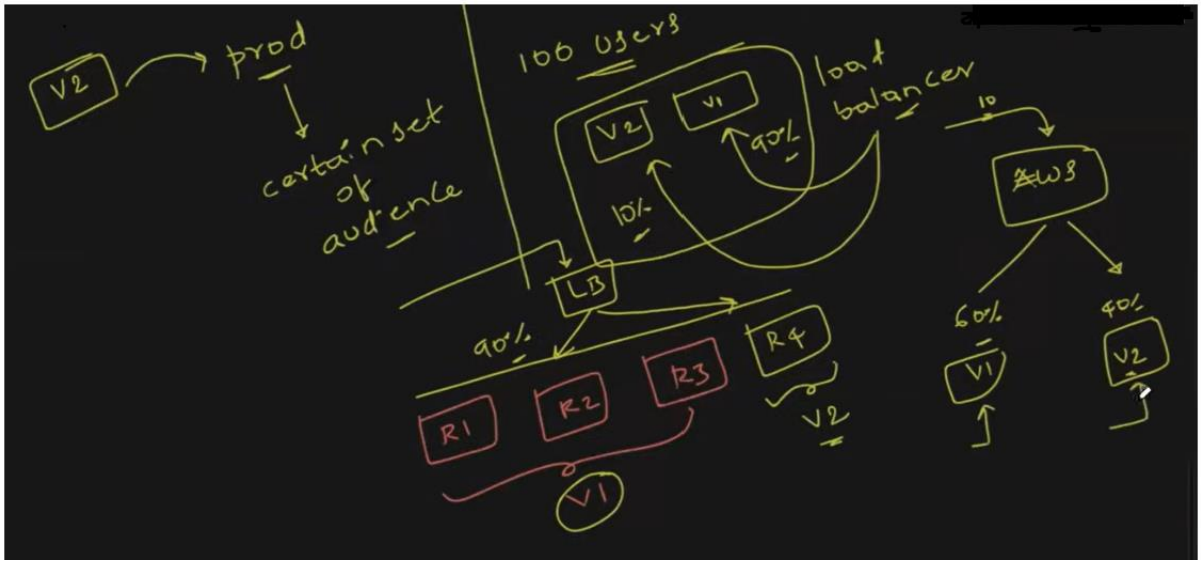
Rolling Update

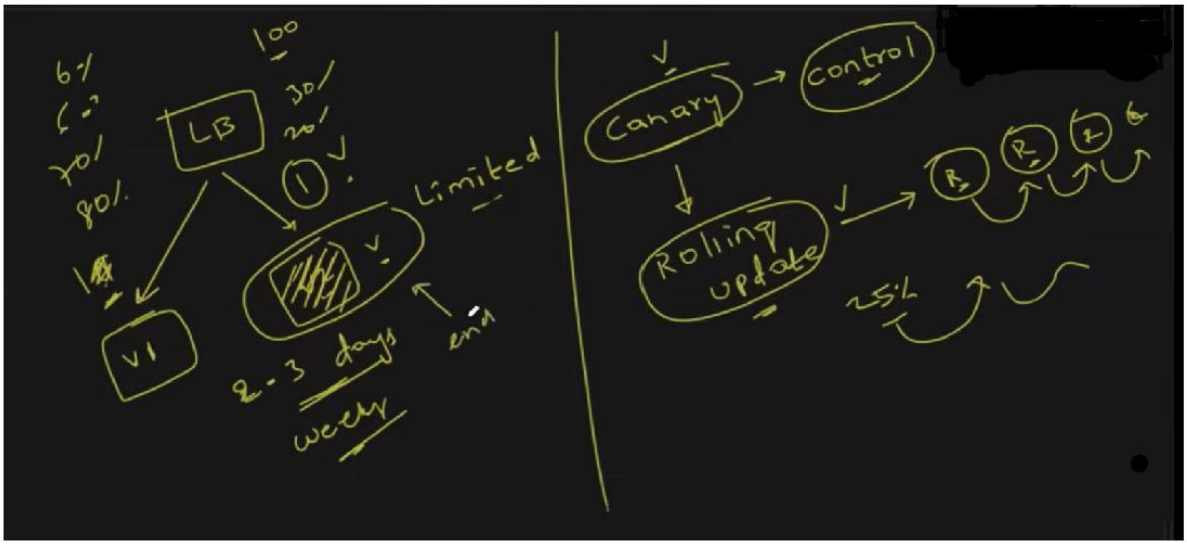
This is the default strategy in Kubernetes. It gradually or incrementally or one by one replaces old pods with new ones without downtime, ensuring that the application remains available
It will replace old pods with 25% first. If it is successful, then gradually replaces old pods with new ones without downtime.

Important Note: Readiness probe configured with Pods. If it is not configured, the problem is that kubernetes can't understand that the new pod or new replica set of new version is up and running or not.

Canary Deployment

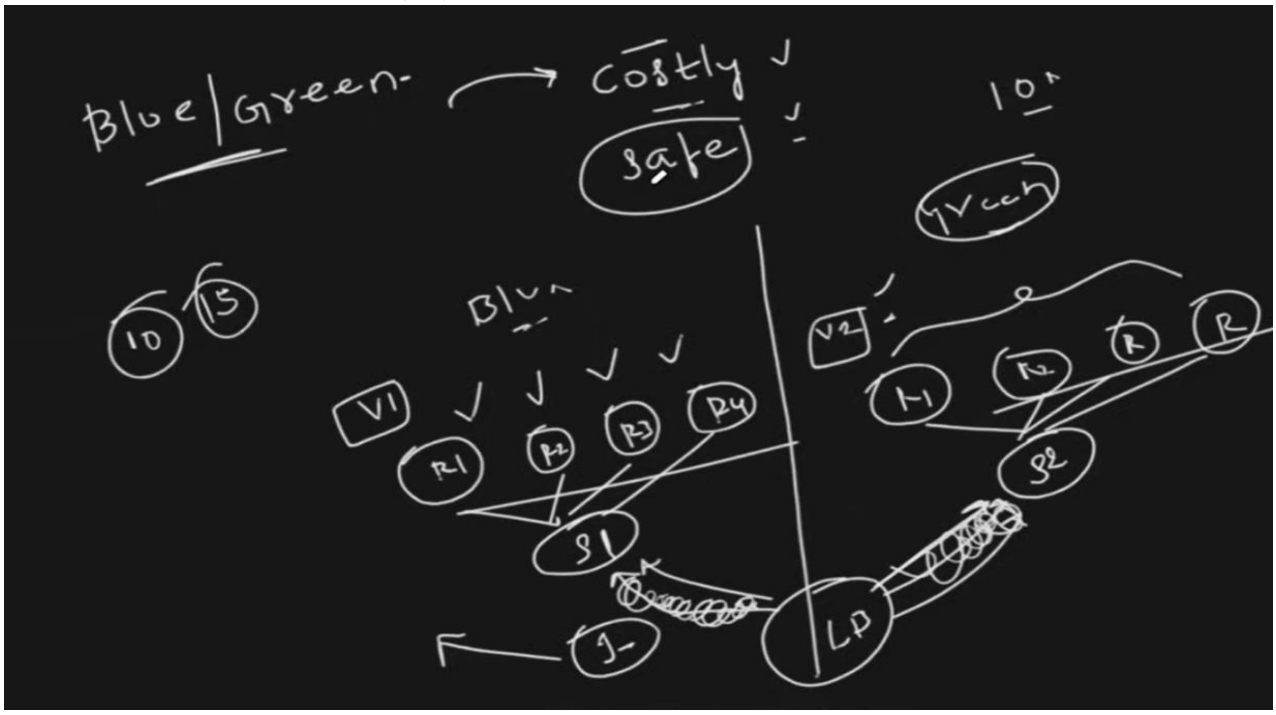
This strategy releases the new version to a small subset of users before rolling it out to the entire user base. It allows for testing in a real-world environment with minimal risk





Blue/Green Deployment:

This strategy involves running two environments, one with the current version (blue) and one with the new version (green). Traffic is switched to the green environment once it's verified to be stable
It is a safe and cost-effective deployment.



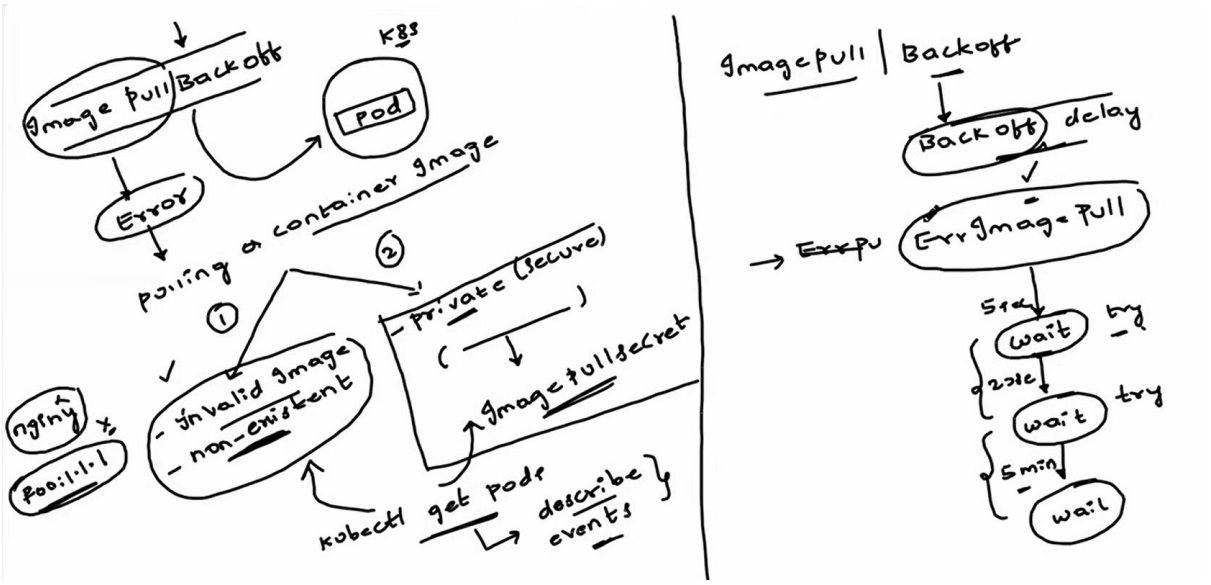
Kubernetes Troubleshooting

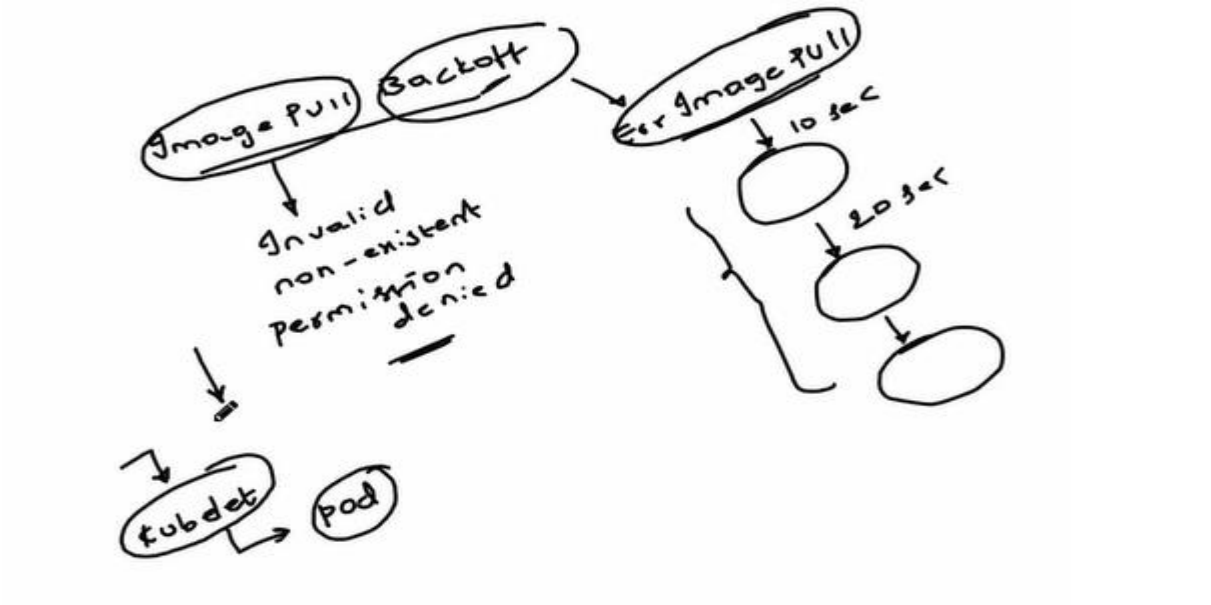
1. **ImagePullBackOff**

   This error is related to pulling a container image into the kubernetes cluster. When you deploy application on to the kubernetes cluster, it can be through a pod directly or a deployment, stateful set, demon set or replica set. You might get this error imagepullbackoff error.

   This error is getting because mainly 3 reasons

   3. Image name is incorrect
   4. Image doesn't exist
   5. Private images – permission denied.





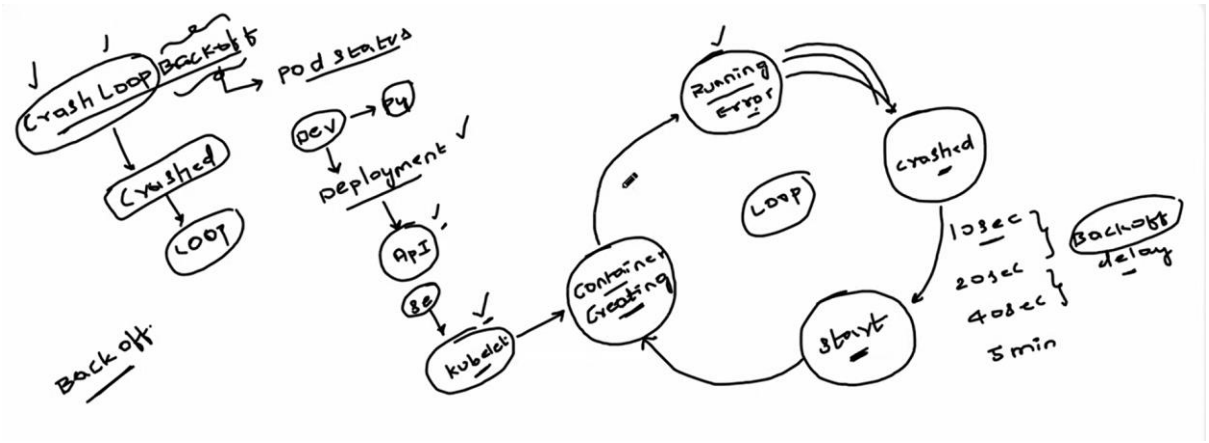
2. CrashLoopBackOff

If pods is running initially but after sometime or immediately, application is not running, then pod will crashes.

There are few reasons are there

1. Misconfigurations
2. Errors in the liveness Probes
3. The memory limits are very low – OOMkilled out of memory
4. Wrong command line arguments.
5. Bugs and Exceptions

6.



Container restart policy

The `spec` of a Pod has a `restartPolicy` field with possible values Always, OnFailure, and Never. The default value is Always.

The `restartPolicy` for a Pod applies to app containers in the Pod and to regular init containers. Sidecar containers ignore the Pod-level `restartPolicy` field: in Kubernetes, a sidecar is defined as an entry inside `initContainers` that has its container-level `restartPolicy` set to `Always`. For init containers that exit with an error, the kubelet restarts the init container if the Pod level `restartPolicy` is either `OnFailure` or `Always`.

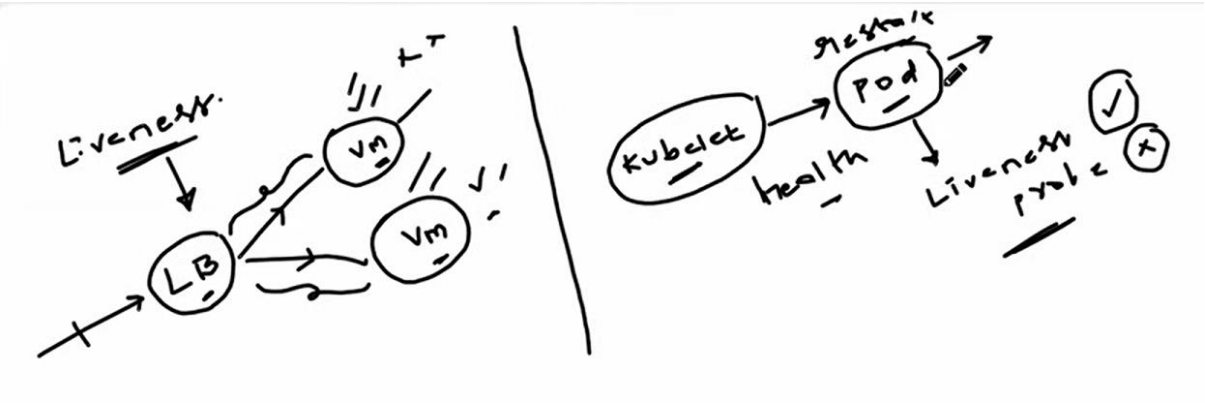
There are 3 probes.

1. Liveness Probe
2. Readyness Probe
3. Startup Probe

Liveness Probe

In Load balancer, it will always check the pods heath status and pass the request accordingly.

In the same way, Kubelet will check the pods heath status. How it will check the health status of pods.
If you configure the liveness probe, kubelet will understand if the liveness probe is passing, then it will understand pod is in the healthy state. If liveness probe fails, immediately kubelet will restart the pod.

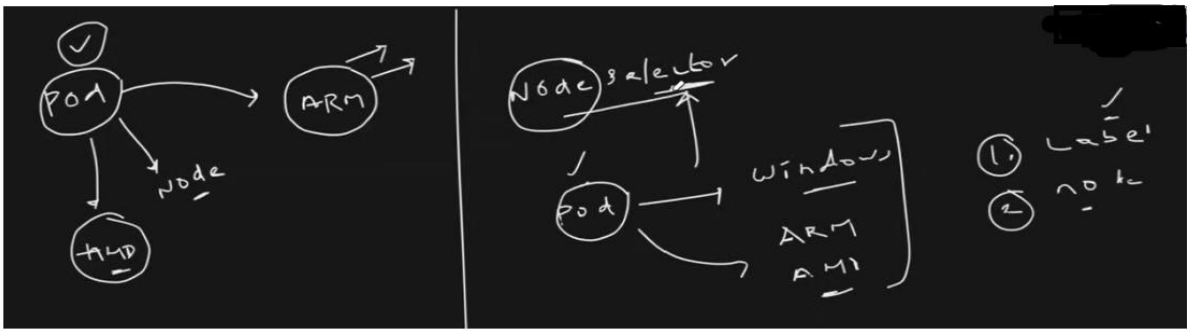


3. **Pod not schedulable.**
    1. **Node Selector**
    2. **Node Affinity**
    3. **Taints**
    4. **Tolerations**

1. **Node Selector**

    Node Selector is a simple way to constrain pods to be scheduled on nodes with specific labels. You specify a key-value pair in the pod's configuration, and the pod will only be scheduled on nodes that have a matching label

    1. FailedScheduling
        - If you didn't update the node name properly, then pod will fail to create.



2. **Node Affinity**

    Node Affinity is similar to Node Selector but offers more flexibility. It allows you to specify rules about which nodes your pod can be scheduled on, based on node labels.

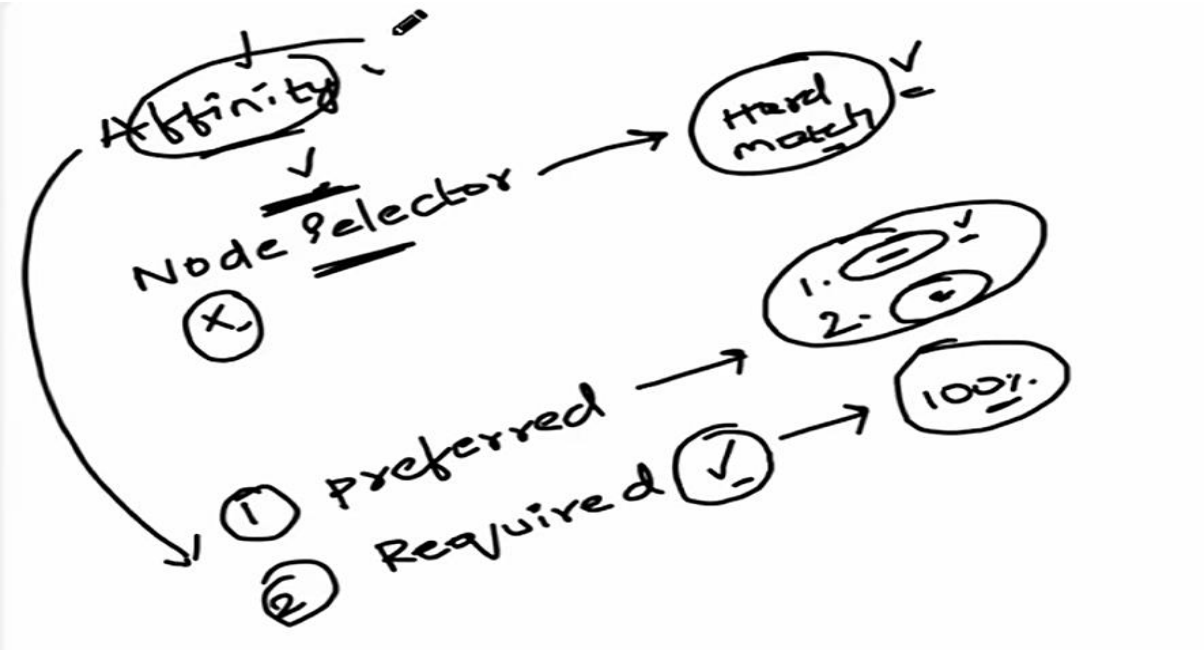
There are two types of node affinity:

requiredDuringSchedulingIgnoredDuringExecution:

- The pod will only be scheduled on nodes that meet the specified criteria.

preferredDuringSchedulingIgnoredDuringExecution:

- The pod prefers to be scheduled on nodes that meet the criteria but can still be scheduled on other nodes if necessary

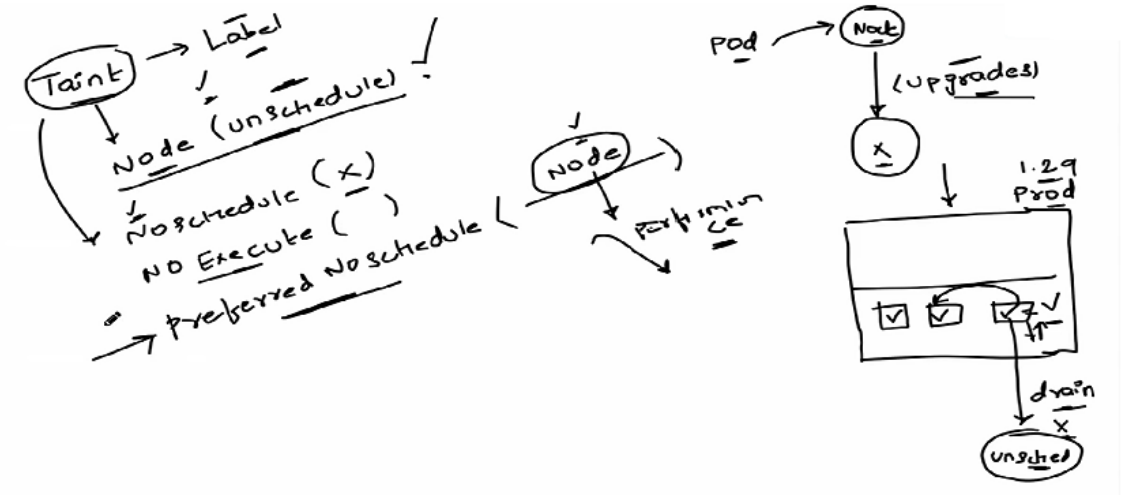


3. **Taints:**

Taints are applied to nodes to prevent pods from being scheduled on them unless the pods have a matching toleration. Taints are used to repel a set of pods from a node. For example, you might taint a node to indicate that it should only run certain types of workloads.

- **Purpose**: Taints are used to mark nodes so that pods are not scheduled on them unless the pods have a matching toleration. This helps to repel certain pods from being scheduled on specific nodes.
- **Usage**: Taints are applied to nodes to indicate that they should not accept certain pods. For example, you might taint a node to indicate that it is reserved for high-priority workloads or that it is undergoing maintenance.
- **Format**: A taint consists of three parts: a key, a value, and an effect. The effect can be one of three types:
    - **NoSchedule**: Pods that do not tolerate the taint will not be scheduled on the node.
    - **PreferNoSchedule**: The system will try to avoid scheduling pods that do not tolerate the taint on the node, but it is not guaranteed.
    - **NoExecute**: Pods that do not tolerate the taint will be evicted from the node if they are already running there.

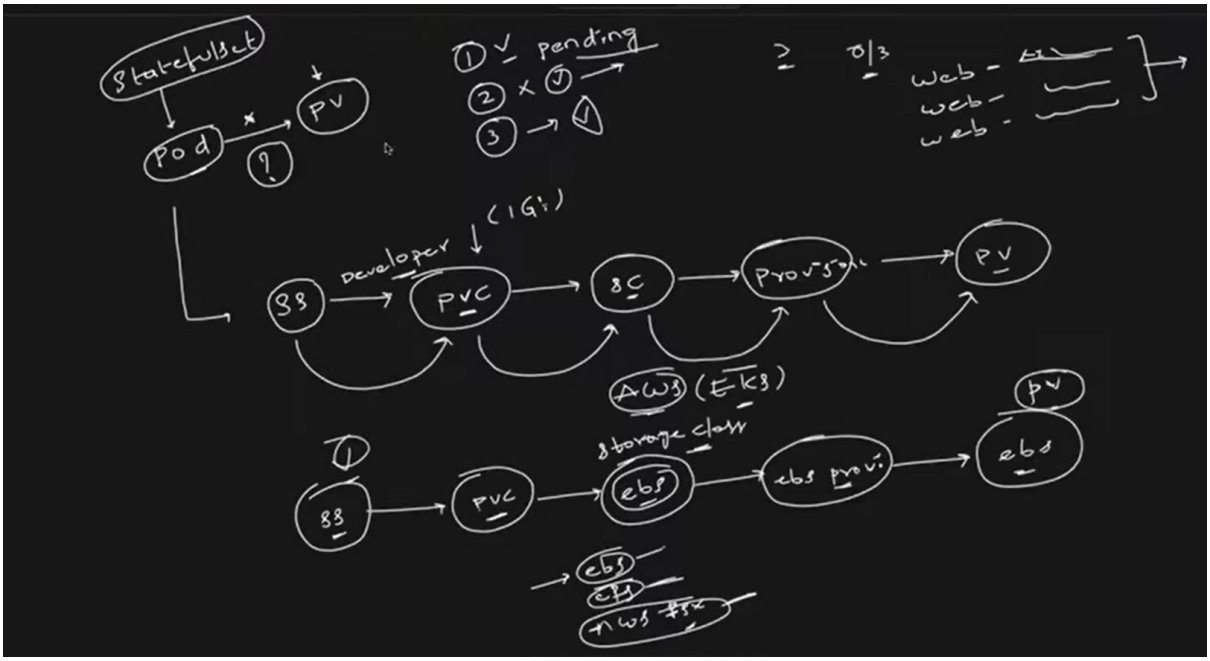
Example:
$ kubectl taint nodes node1 key=value:NoSchedule
This command applies a taint to node1 with the key key, value value, and effect NoSchedule.
Taints are a powerful tool for controlling pod placement and ensuring that nodes are used appropriately within a Kubernetes cluster.



4. **Tolerations:**

Tolerations are applied to pods to allow them to be scheduled on nodes with matching taints. Tolerations do not guarantee scheduling but allow the scheduler to consider the pod for nodes with the specified taints

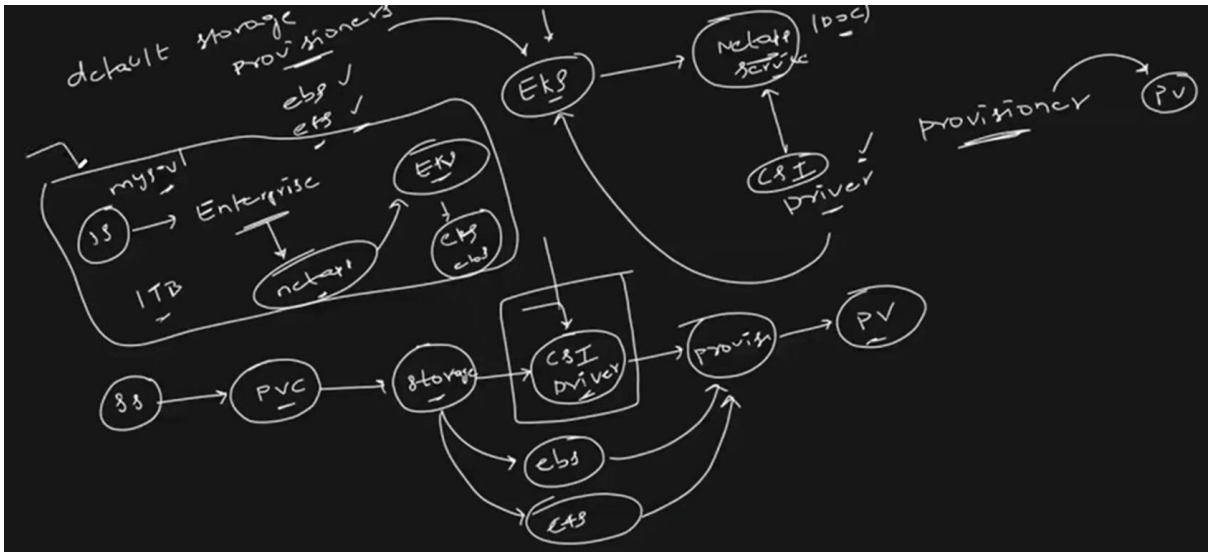
4. **Failedschedule due persistence volume and PVC issue.**



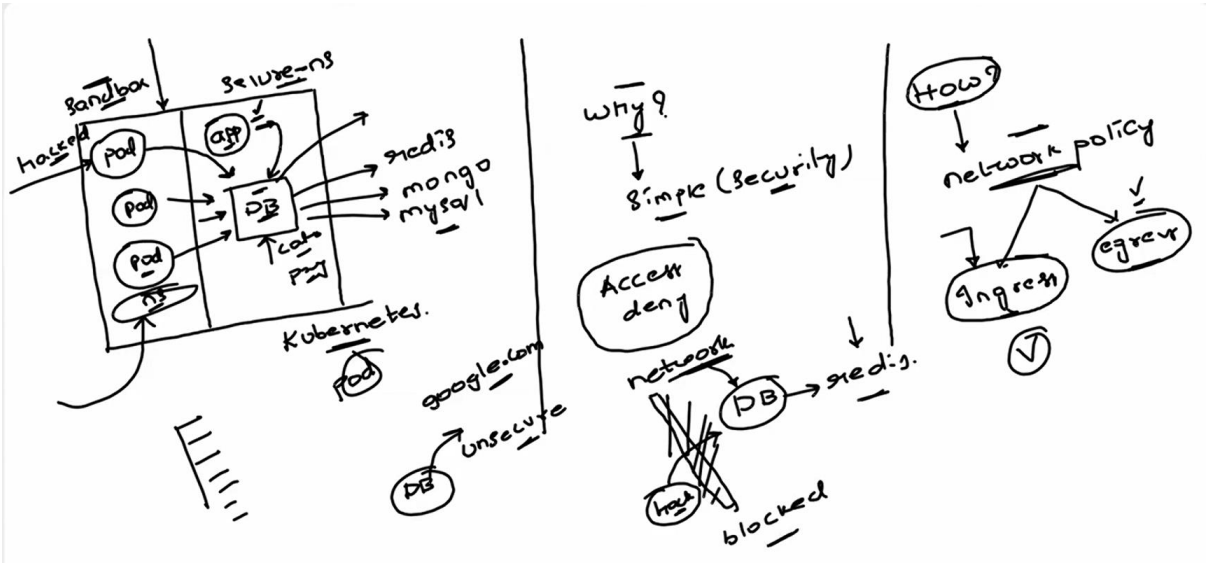
We have provide right storage class yml file

CSI Driver

Whenever you want to use the external storage provisioner, you need to install the CSI driver (Container storage interface) which will act as a provisioner and it will create the PV.

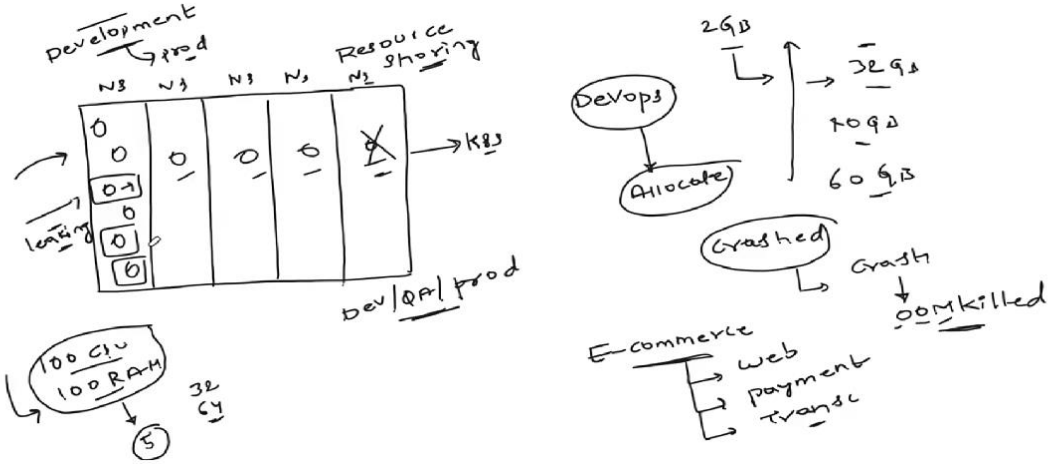


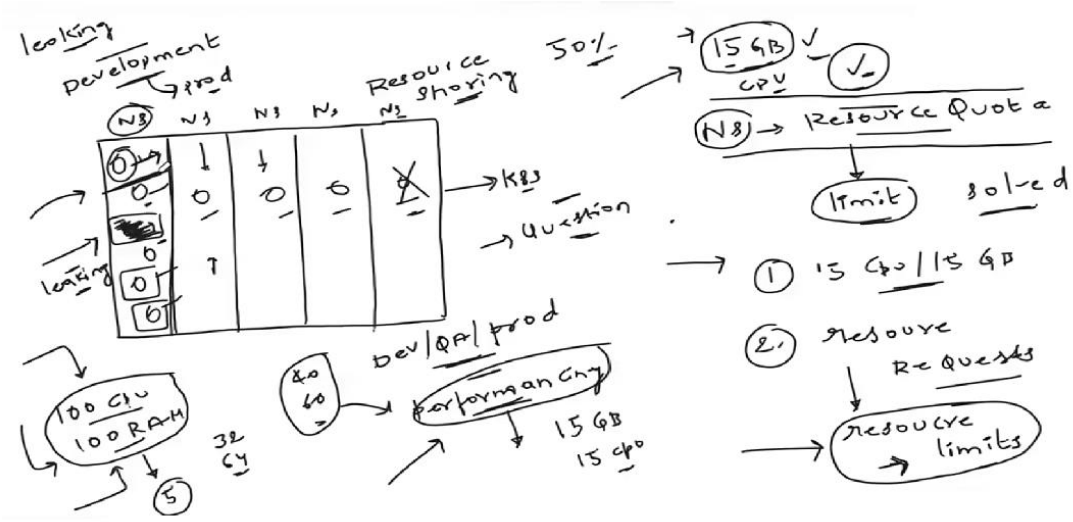
5. **Network policy: Pod security access issue.**

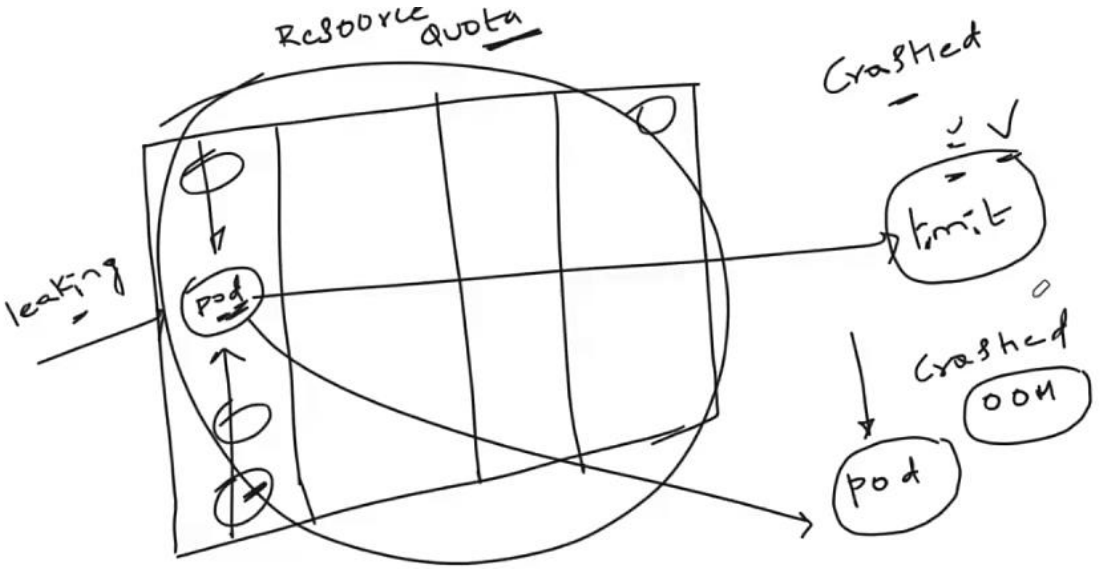


Real Time challenges in Kubernetes.

1. **Resource allocation (Resource Sharing) – Solution use resource quota**
   - Leaking memory – consuming more memory in a pods or services due to application or something in a particular namespace then other resource will not deploy.
   - If we use resource quota, 50 % issue with will resolve. We reduced the blast radius but still issue in a particular namespace.
   - **Solution: Part 1 - Resource quota**
     Part 2 – Resource limits or Resource Requests.

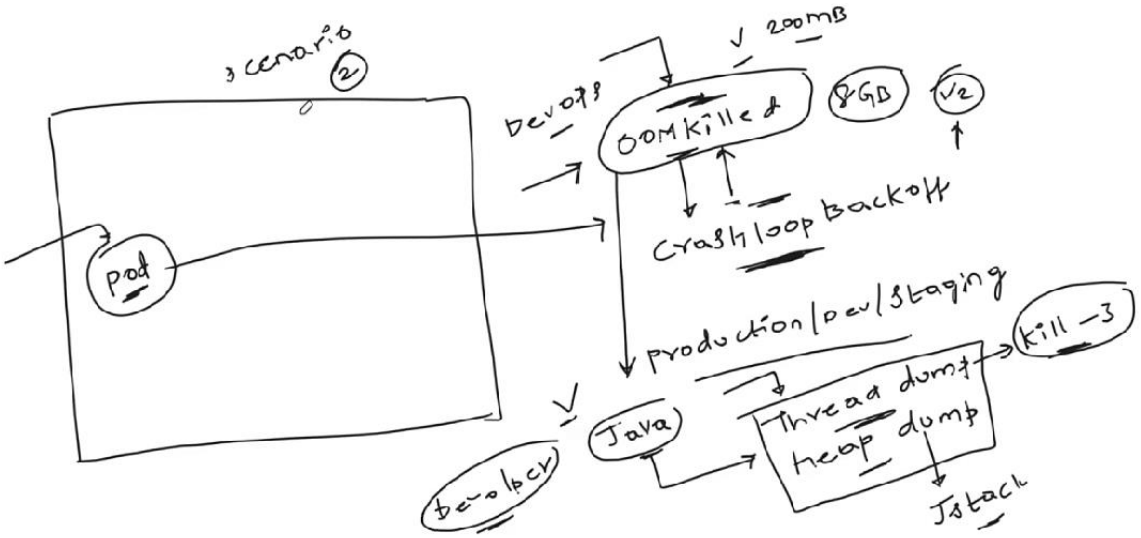






2. OOM Killed issue with pod

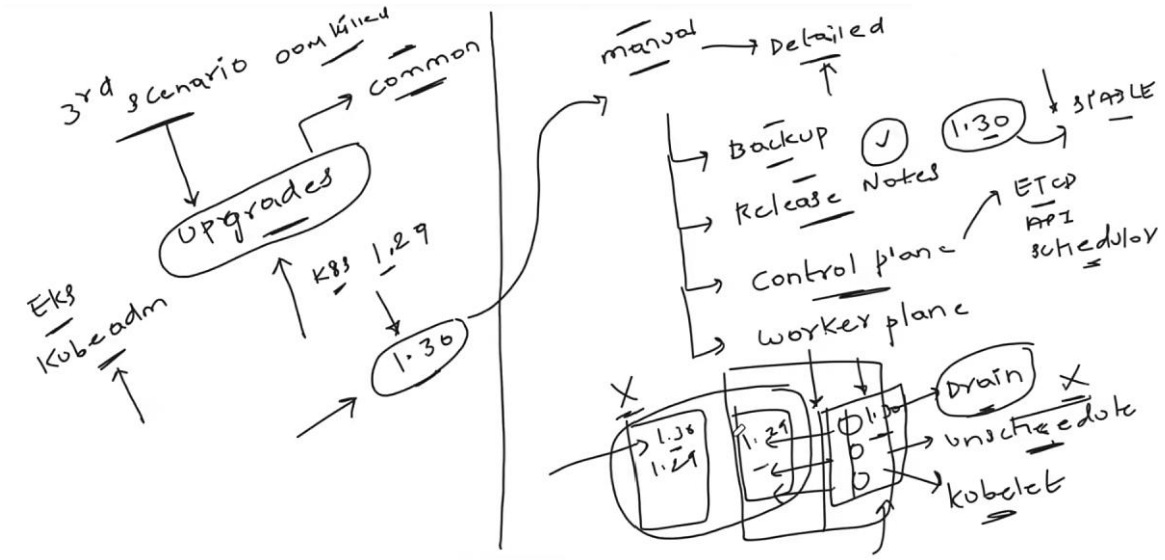
1. If OOM Killed issue occurs in pod, first you have to set the resource quota and resource limits. If still the same issue, based programming language collect thread and heap dumps and share it with developer.
2. Java - collect thread dump (CMD: kill 3) and heap dump (CMD: Jstack) and share it with developer.

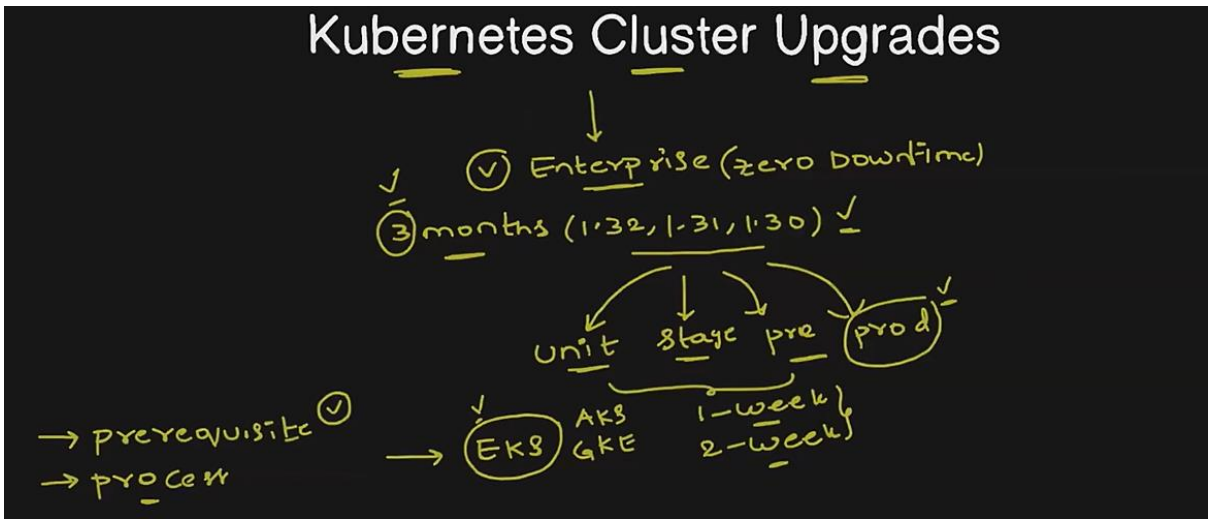


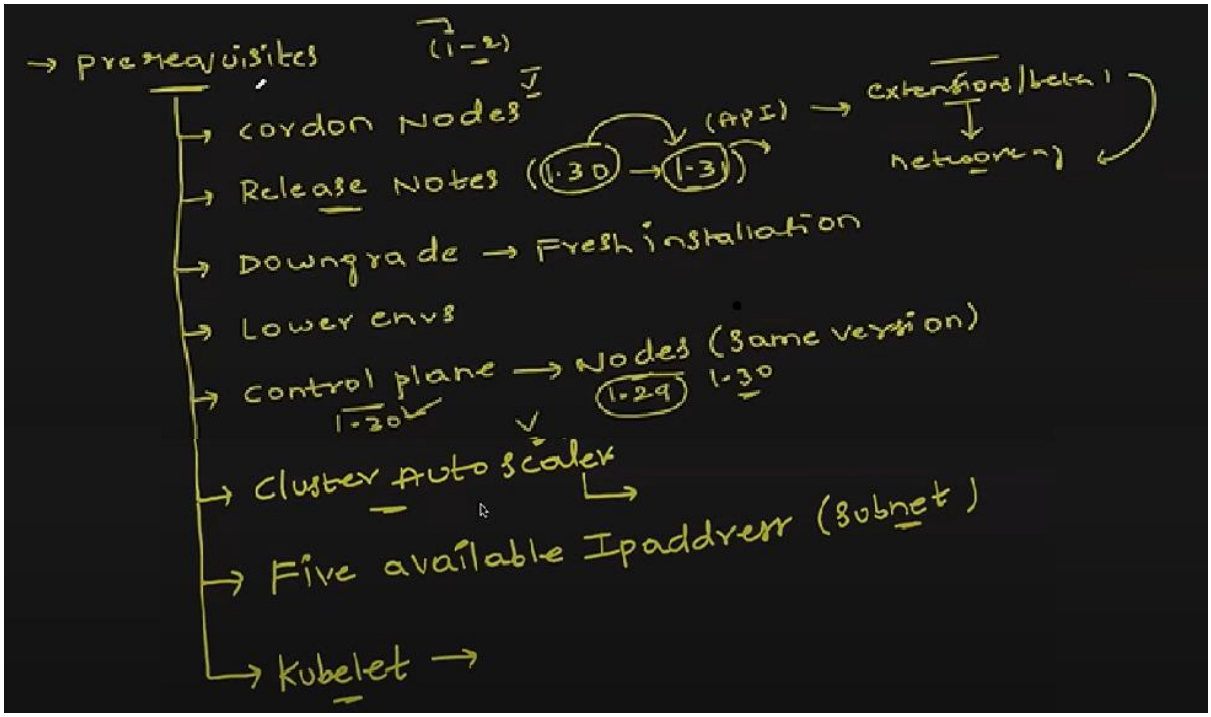
3. Upgrades

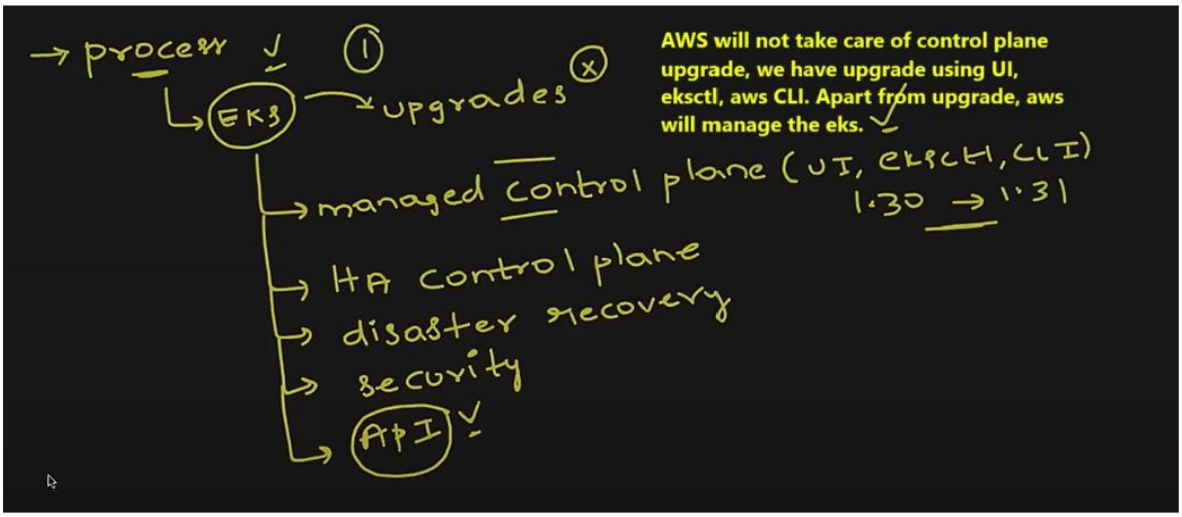
- If we want to upgrade the cluster from 1.29 to 1.30, follow the below steps
- Create a manual with detailed steps documentation.
- How to take the backup of resources
- How to read release notes, what are the exact points in the release notes that might impact the cluster
- Divide the steps for control plane and worker plane.
- Control plane – list the components to upgrade, ETCD, API, Schedular etc.
- Worker plane
  If you have 3 nodes, we must choose one node and make it drain and unscheduled (taint the node or cordon node). Then upgrade the node and once done, remove the unscheduled taints and any other taints that are applied to the node. Similarly update the other nodes as well.

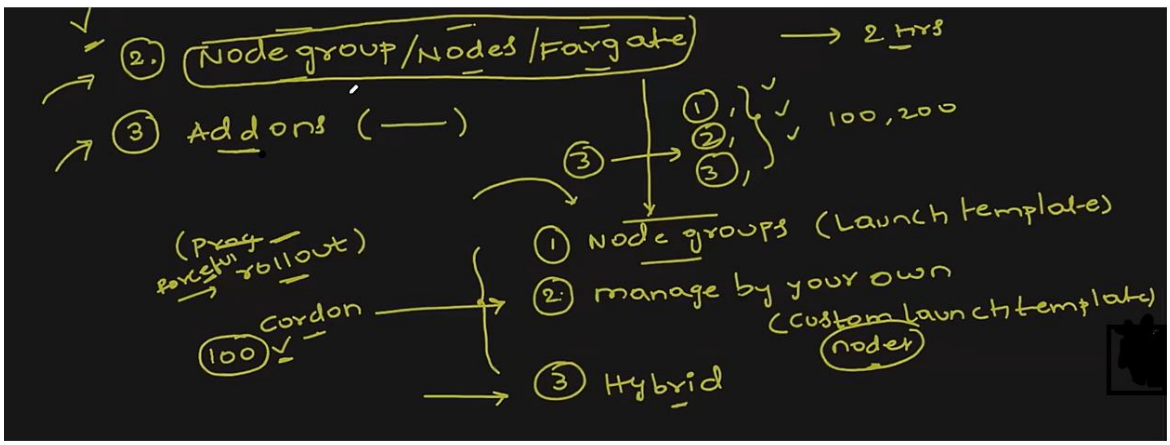
  Drain means you will give the particular node time or scheduler time to move all the running pods to another node.


3rd scenario oomkilled
common
Upgrades
K8s 1.29
1.30
Eks
Kubeadm
manual → Detailed
Backup
Release Notes
Control plane
ETCD
API
scheduler
worker plane
1.30
STABLE
Drain
Uncheedule
kubelet

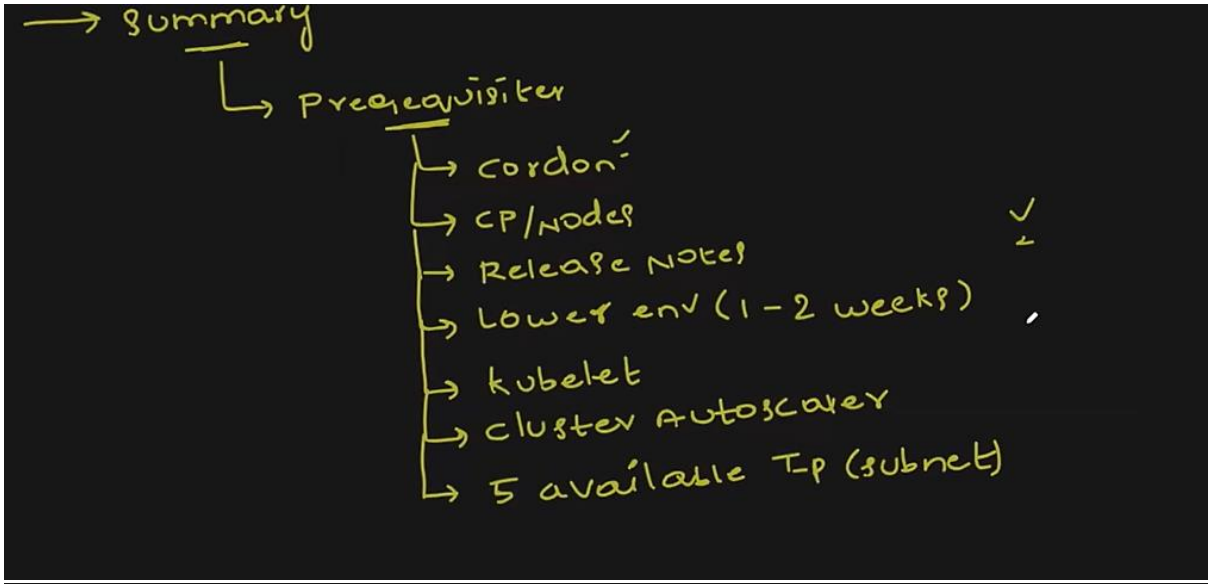

Kubernetes Cluster Upgrades
Enterprise (zero Downtime)
3 months (1.32, 1.31, 1.30)
unit
stage
pre
prod
→ prerequisite
→ process
Eks
AKS
GKE
1-week
2-week

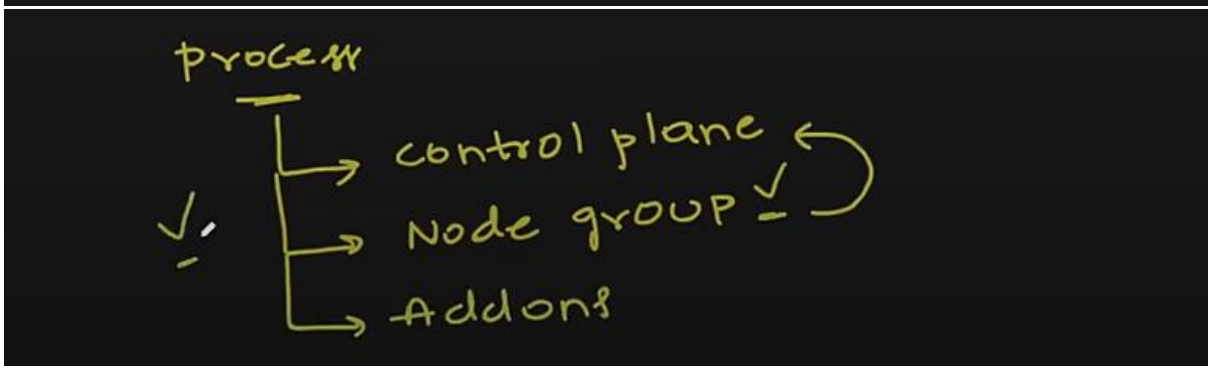

→ Pre-requisites (1-2)
→ cordon Nodes
→ Release Notes (1.30 → 1.31) (API) → Extensions/helm / Networking
→ Downgrade → Fresh installation
→ Lower envs
→ control plane → Nodes (same version)
1.30 1.29 1.30
→ Cluster Auto scaler
→ Five available Ipaddress (subnet)
→ Kubelet →


→ process (1)
EKS → upgrades (X)
AWS will not take care of control plane upgrade, we have upgrade using UI, eksctl, aws CLI. Apart from upgrade, aws will manage the eks.
→ managed control plane (UI, eksctl, CLI)
1.30 → 1.31
→ HA control plane
→ disaster recovery
→ security
→ API


2. Node group / Nodes / Fargate → 2 hrs
3 Addons (——)
3 → 1, 2, 3, 100, 200
(Proxy forceful rollout)
Cordon
100
1 Node groups (Launch template)
2 manage by your own (Custom Launch template)
node
3 Hybrid

Rolling update with zero down time upgrade





1. **what is the difference between the node affinity and node selector?**
   Both **Node Affinity** and **Node Selector** are used to control which nodes a pod can be scheduled on, but they have some key differences:
   Node Selector

- **Simplicity**: Node Selector is simpler and more straightforward. You specify a key-value pair in the pod's configuration, and the pod will only be scheduled on nodes that have a matching label.
- **Usage**: It's a basic way to constrain pods to specific nodes based on labels.
- **Example**: If you have a node labeled disktype=ssd, you can use Node Selector to ensure that a pod is only scheduled on nodes with this label.

Node Affinity

- **Flexibility**: Node Affinity offers more flexibility and advanced scheduling options compared to Node Selector.
- **Types**: There are two types of node affinity:
  requiredDuringSchedulingIgnoredDuringExecution:
  The pod will only be scheduled on nodes that meet the specified criteria.
  preferredDuringSchedulingIgnoredDuringExecution:
  The pod prefers to be scheduled on nodes that meet the criteria but can still be scheduled on other nodes if necessary.
  Usage: It allows for more complex rules and preferences, making it suitable for more sophisticated scheduling requirements.
  Example: You can specify that a pod should preferably be scheduled on nodes with disktype=ssd, but if no such nodes are available, it can still be scheduled on other nodes.

  In summary, **Node Selector** is simpler and more restrictive, while **Node Affinity** provides more flexibility and control over pod scheduling. If you need basic scheduling constraints, Node Selector is sufficient. For more complex scheduling needs, Node Affinity is the better choice.

Day to day activities

Managed the kubernetes cluster like scaling the cluster or pods.
Governance the kubernetes cluster.

Managing the aws cluster or kubernetes cluster according to the compliance of organisation.

Container storage interface CSI
Container network interface CNI
Difference b/w PV and PVC and storage class.
Pod should claim the PV or post pod and PV create
Which one should create one by one Pod >> PVC >> PV need to create.

Do we use the Kube admin in EKS?
Cluster upgradation
Secreat types.
Core DNS.
EKS Add ons.
Trains, Affinity, init container, side car container
Overly
How to upgrade the existing deployment in Kubernetes.
How many type of API's?
K8s monitoring API,
K8s API's?
Readiness probe and liveness probes
Why do you use Kubernetes Jobs and crone jobs?
Awk in linux

Helm
Helmfile
Helmfile doesn't come along with the installation.
We need to install helmfile using utilities separately.

Objects in kubernetes

Kubernetes objects are persistence entities in kubernetes system. Means if any pods will die, it will create new pods. Kubernetes will check every time whether the object or resources are running or not using API server.

Bottleneck

1. Production one cluster, for lower evn we were using the one cluster and different name space. There was issue with service file or resource issue,
2. Our node group does support for required container that is about to be deployed, it should deploy in high end server, so we deployed in different high end server.
3. Restart policy in docker file or Kubernetes

Interview Questions

2. What is the **difference Docker and Kubernetes?**
   a. Docker is a container platform where as Kubernetes is a container orchestration environment that offers capabilities like Auto healing, Auto Scaling, Clustering and Enterprise level support like Load balancing.

3. What are the main **components of Kubernetes architecture?**
   A. On a broad level, you can divide the kubernetes components in two parts
   1. Control Plane (API SERVER, SCHEDULER, Controller Manager, C-CM, ETCD)
   2. Data Plane (Kubelet, Kube-proxy, Container Runtime)

4. What are the main **differences between the Docker Swarm and Kubernetes?**
   A. Kubernetes is better suited for large organisations as it offers more scalability, networking capabilities like policies and huge third party ecosystem support.

5. What is the **difference between Docker container and a Kubernetes pod ?**
   A. A pod in kubernetes is a runtime specification of a container in docker. A pod provides more declarative way of defining using YAML and you can run more than one container in a pod.

6. What is a **namespace in Kubernetes**?
   A. In Kubernetes namespace is a logical isolation of resources, so that multiple project teams in a company can work on the same kubernetes cluster, but each of them will have a dedicated namespace, so that noboady will interrupt the work of the other people or other projects.

   In Kubernetes namespace is a logical isolation of resources, network policies, rbac and everything. For example, there are two projects using same k8s cluster. One project can use ns1 and other project can use ns2 without any overlap and authentication problems.

7. What is the **role of kube proxy**?
   A. Kube-proxy works by maintaining a set of network rules on each node in the cluster, which are updated dynamically as services are added or removed. When a client sends a request to a service, the request is intercepted by kube-proxy on the node where it was received. Kube-proxy then looks up the destination endpoint for the service and routes the request accordingly.

Kube-proxy is an essential component of a Kubernetes cluster, as it ensures that services can communicate with each other

8. What are the **different types of services within Kubernetes**?
   A. There are three different types of services that a user can create.
   1. Cluster IP Mode
   2. Node Port Mode
   3. Load Balancer Mode

9. What is the **difference between NodePort and Load Balancer type service**?
   A. When a **service is created a NodePort type**, The kube-proxy updates the IP Tables with Node IP address and port that is chosen in the service configuration to access the pods.

   Where as if you create a **Service as type Load Balance**r, the cloud control manager creates a external load balancer IP using the underlying cloud provider logic in the C-CM. Users can access services using the external IP

10. What is the role of **Kubelet**?
    A. Kubelet manages the containers that are scheduled to run on that node. It ensures that the containers are running and healthy, and that the resources they need are available.

    Kubelet communicates with the Kubernetes API server to get information about the containers that should be running on the node, and then starts and stops the containers as needed to maintain the desired state. It also monitors the containers to ensure that they are running correctly, and restarts them if necessary.

11. **What are the day to day activities on Kubernetes?**
    A. As part of the devops engineer role we manage kubernetes clusters for our organization and we also ensure that you know the applications are deployed onto the kubernetes cluster and there are no issues with the application so we have set up monitoring on our Kubernetes cluster we ensure that whenever there are bugs on the kubernetes cluster for example uh the developers are not able to troubleshoot some issue with respect to pods developers are not able to troubleshoot with respect to Services they are not able to you know uh route the traffic in inside the Kubernetes cluster so in such cases as subject matter expertise on the Kubernetes Clusters we coming to picture and we solve their problems but apart from that we also do a lot of Maintenance activities for example uh we have kubernetes clusters with three Master nodes and 10 worker nodes so we have to do some continuous maintenance activities on this worker nodes probably uh you know upgrading the versions of this worker nodes or installing some default mandatory packages ensuring that these worker nodes are not uh security uh exposed to security vulnerabilities so all of these things are our day-to-day activities on Kubernetes apart from that we also serve as subject matter expertise on kubernetes so if anyone in the organization has any issues with kubernetes they create a jeera items for us or you know they create tickets for us and we will help them in solving or making them understand the concept of kubernetes so this is how you can explain so it is a very simple answer it's a very straightforward answer you don't have to you know get

scared about this question so these are the 10 questions that I have for today and let us see how many people were able to get all the 10 questions correct because you know most of the questions we have covered I think eight questions we already covered in the previous videos so let us see what is the uh scorecard and uh yeah in future videos we will learn about Ingress we will learn about the Practical implementation of services we'll also talk about custom resources definitions we will see a few things about Helm so it's going to be four or five videos more on kubernetes and after that we'll also do a Kubernetes interview questions part two so if you like the video click on the like button and if you feel that someone who is not following our 45 days of devops course please share these videos with them so that they'll also get benefit out of the videos thank you so much I'll see in the next video take care everyone bye

12. **what is the difference between the node affinity and node selector?**

Both **Node Affinity** and **Node Selector** are used to control which nodes a pod can be scheduled on, but they have some key differences:

Node Selector

- **Simplicity**: Node Selector is simpler and more straightforward. You specify a key-value pair in the pod's configuration, and the pod will only be scheduled on nodes that have a matching label.
- **Usage**: It's a basic way to constrain pods to specific nodes based on labels.
- **Example**: If you have a node labeled disktype=ssd, you can use Node Selector to ensure that a pod is only scheduled on nodes with this label.

Node Affinity

- **Flexibility**: Node Affinity offers more flexibility and advanced scheduling options compared to Node Selector.
- **Types**: There are two types of node affinity:
  requiredDuringSchedulingIgnoredDuringExecution:
  The pod will only be scheduled on nodes that meet the specified criteria.
  preferredDuringSchedulingIgnoredDuringExecution:
  The pod prefers to be scheduled on nodes that meet the criteria but can still be scheduled on other nodes if necessary.
  Usage: It allows for more complex rules and preferences, making it suitable for more sophisticated scheduling requirements.
  Example: You can specify that a pod should preferably be scheduled on nodes with disktype=ssd, but if no such nodes are available, it can still be scheduled on other nodes.

  In summary, **Node Selector** is simpler and more restrictive, while **Node Affinity** provides more flexibility and control over pod scheduling. If you need basic scheduling constraints, Node Selector is sufficient. For more complex scheduling needs, Node Affinity is the better choice.

13.
14.

15.
16.
17.
18.
19.
20.
21.
22.
23.
24.
25.