**Course content:**

* Docker
* Kubernetes on Google Cloud, AWS with spring boot micro-service applications
* Micro-Services Design patterns

# Devops

**Containerization**:- **Containerization** using **Docker**. **Docker** is the **containerization** platform which is used to package your application and all its dependencies together in the form of containers so to make sure that your application works seamlessly in any environment which can be development or test or production.

* Containerization makes the deployment process easy, independent of development language and their environment.
* A blue box with black text

  Description automatically generated

**Step1:-** Install docker desktop for windows from docker site, For Docker version check **docker – version** type u will get docker version.   
FYI: <https://docs.docker.com/desktop/windows/install>

**Note: -**

* Docker file is set of instructions.
* Docker needs a set of instructions to run the any type of applications, it doesn’t worry about what are the dependencies, run time environment about applications. It will do the same job for all the images, just it need runtime docker environment. It will convert docker images to docker container but running the image.

**Docker-Image:-**

A **Docker image** is an immutable (unchangeable) file that contains the source code, libraries, dependencies, tools, and other files needed for an application to run.

**Docker-Container:-**

whereas [**Docker Container**](https://go4hosting.in/knowledgebase/docker/what-is-docker-container) is the instantiation of Docker Image. In other words, Docker Container is the run time instance of images.

#### Difference b/w kill and stop:

Kill command will stop container abnormally (Without stopping resources) Stop command will stop smoothly.

**Docker Container Ports Differentiating Table :**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S N0** | **Application/Service port** | **Docker File Expose Port** | **Container/Bridge network port** | **Host port** | **Outcome** |
| 1 | 9090 | N/A | 9090 | any | **working** - http://localhost:any/ |
| 2 | 9090 | 9090 | 9090 | any | **working** - http://localhost:any/ |
| 3 | 9090 | 9091 | 9090 | any | **working** - http://localhost:any/ |
| 4 | 9091 | 9090 | 9090 | any | not working - http://localhost:any/ |
| 5 | 9999 | 8989 | 8991 | any | not working - http://localhost:any/ |

**Note:**

* **Container/bridge network port** will be given in docker run command.
* **Application/Service port** will be given through app(spring boot service application.properties file)
* **Docker File Expose Port** will be given **dockerfile.**
* **Application/Service port** will be exposed from docker container only if we write either docker file export port or container/bridge network port.
* **Application/Service port** and **Container bridge network port** should match to link/find your application in docker container world and to expose outside world(Host Server Port)

**Docker inspect JSON file ports property structure:**

Case 1:



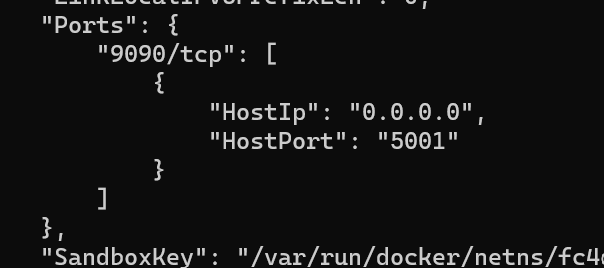
Case 2:



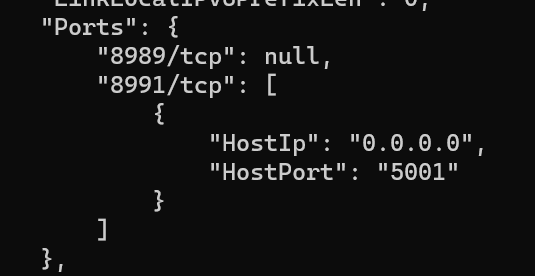
Case 3:



Case 4:



Case 5:



FYI:

<https://github.com/in28minutes/devops-master-class/tree/master/docker#commands>

**Note: -** Run the container in detached mode (if we close terminal will not stop running container) command

docker run –d –p 5000:5000 path-of-image

if u want to see the logs of the container then use following command

**docker logs container-id**

Command to check docker-images

#### docker images

If u want to see docker containers

#### docker container ls

If u want to see docker container wich are also killed and other status

docker container ls –a

docker logs –d –f container-id(first 3 letters)

docker --version

docker run -p 5000:5000 in28min/hello-world-python:0.0.1.RELEASE docker run -p 5000:5000 in28min/hello-world-java:0.0.1.RELEASE docker run -p 5000:5000 in28min/hello-world-nodejs:0.0.1.RELEASE docker run -d -p 5000:5000 in28min/hello-world-nodejs:0.0.1.RELEASE

* - This will follow the docker container logs continuously .

docker run -d -p 5001:5000 in28min/hello-world-python:0.0.1.RELEASE docker logs 04e52ff9270f5810eefe1f77222852dc1461c22440d4ecd6228b5c38f09d838e docker logs c2ba

docker images docker container ls

docker container ls -a

docker container stop f708b7ee1a8b

docker run -d -p 5001:8080 in28min/hello-world-rest-api:0.0.1.RELEASE

docker container pause 832 docker container unpause 832

docker system df---To see disk space after using all containers and images docker system events---will be able to see all exiguted events

docker container prune---will remove all unused containers

docker system prune –a ---will remove the all images which are not associated with container and un used containers aswell

docker stats container-id--- will give all the statistics of the container docker container run -p 5000:5000 -d -m 512m in28min/hello-world- java:0.0.1.RELEASE --- run the container with given amount of memory docker container run -p 5000:5000 -d -m 512m --cpu-quota=50000 in28min/hello-world-java:0.0.1.RELEASE --- With given cpu Quota

docker image history

docker image inspect

docker image rm –note: Only unused images can be removed (should not contained started or stopped containers)

docker conatiner pause {container-id}

docker conatiner unpause {container-id}

docker conatiner stop {container-id}

docker conatiner kill {container-id}

Difference between stop and kill command:

**stop** - will make sure to close all teh resources

- it will stop conatiner gracefully

**kill** - will never worry about any open resources and it will exist immediately

sig comamnd will pasue to docker deamanon

docker conatiner inspect id

docker conatiner prune id

docker system

docker system df

docker system events

docker conatiner -a

docker system prune -a

docker system prune -a -f

docker conatiner ls

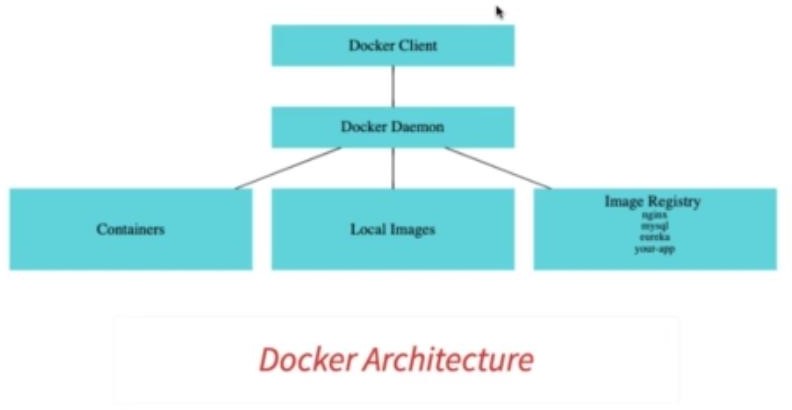
docker stats {conatiner-id}

docker run -p 5000:5000 -d -m 512m

--cpu-quota=50000

{image-name}

Docker Archicture:



Why docker is so popular

* Docker containers are light weight and isolated each other.
* **Lightweight**: Containers share the machine's OS system kernel and therefore do not require an OS per application, driving higher server efficiencies and reducing server and licensing costs. Secure: Applications are safer in containers and Docker provides the strongest default isolation capabilities in the industry.



**Docker file:-** The place where we will write all instructions to create docker-image to docker engine

Let’s download some projects from the in28min Git. Go to the current project directory and do docker build image as following command

**Step1:** Build docker image

#### docker build –t tag-name(image-tag-name) build-context

Ex: docker build –t docker-python-sample .

**Step2:** Now run docker container with respective image

#### docker container run –p 5000:5000 –d docker-python-sample

**Let’s have a java application to run using Docker** 1)For a java application we have 2 steps in docker file Step1:- create a jar file

Step2:- create image with generated jar file

# Build a JAR File

FROM maven:3.6.3-jdk-8-slim AS stage1 WORKDIR /home/app

COPY . /home/app/

RUN mvn -f /home/app/pom.xml clean package

# Create an Image

FROM openjdk:8-jdk-alpine EXPOSE 5000

COPY --from=stage1 /home/app/target/hello-world-java.jar hello-world-java.jar

ENTRYPOINT ["sh", "-c", "java -jar /hello-world-java.jar"]

**FROM** – base images

**WORKDIR** – it creates folder in container

**COPY** – it copies current folders from source path to destination path

1. Here we are referring up stage1 to down stage2, ENTRYPOINT is command to execute java commands

Token: 042012f3-63f9-4243-97c0-ac7a291215ed

|  |
| --- |
| 1)Each line of your Dockerfile creates an intermediate container to execute the Dockerfile directive for that line.  2)If the directive succeeds, that will create an intermediate image, which will be the base for the next container to be launch (to execute the next line of your Dockerfile)  3)  To speed up the build process the intermediate container will be used as cache. Those containers will be deleted after a succesful build but when you try to build for multiple times the 'old' intermediate containers and image will still be on your system  4)  FROM is the previous layer for first instruction runs the current instruction inside the container and commits this modified container as new image layer.  5) When we use FROM, previous container of previous command will not be used for next command container until unless you give reference(--from=alias) |

#### Difference b/w ENTRYPOINT vs CMD:

To lunch up java applications we used **ENTRYPOINT**, To lunch up NodeJS and Python applications we used CMD (Which one should be used when?)

#### docker run -d -p 5001:5000 in28min/hello-world-java:0.0.1.RELEASE ping google.com

If you observe above command arguments you pass(ping google.com) will replaces in CMD instruction

CMD [ "node","index.js"]

Now ping google.com will replace the node index.js

But even you pass arguments will not replace in ENTRYPOINT

ENTRYPOINT ["sh", "-c", "java -jar /hello-world-java.jar"]

|  |
| --- |
| 1) The ENTRYPOINT specifies a command that will always be executed when the container starts.  2) The CMD specifies arguments that will be fed to the ENTRYPOINT.  3) If you want to make an image dedicated to a specific command you will use ENTRYPOINT ["/path/dedicated\_command"]     Otherwise, if you want to make an image for general purpose, you can leave ENTRYPOINT unspecified and use CMD ["/path/dedicated\_command"]     as you will be able to override the setting by supplying arguments to docker run. |

**Deploying microservices in docker:**

1. Docker applications cannot communicate each other generally.
2. To communicate docker containers either we need to make them in same network or we need to provide the link between them.

How to establish the communication between multiple microservices in docker containers:?

1. **–link**

  docker container run -d -p 5000:9090 --name=irctc-microservice harikrishnathopugunta805/irctc-microservice:0.0.1-RELEASE

docker container run -d -p 5001:9091 --env=IRCTC\_SERVICE\_HOST=https://irctc-microservice --name=makemytrip-microservice --link=irctc-microservice harikrishnathopugunta805/makemytrip-microservice:0.0.1-RELEASE

docker container run -d -p 5002:9092 --env=IRCTC\_SERVICE\_HOST=https://irctc-microservice --name=paytm-microservice --link=irctc-microservice harikrishnathopugunta805/paytm-microservice:0.0.1-RELEASE

1. **--network**

 docker container run -d -p 5000:9090 --name=irctc-microservice --network=tickets-network  harikrishnathopugunta805/irctc-microservice:0.0.1-RELEASE

docker container run -d -p 5001:9091 --name=makemytrip-microservice --network=tickets-network harikrishnathopugunta805/makemytrip-microservice:0.0.1-RELEASE

docker container run -d -p 5002:9092 --name=paytm-microservice --network=tickets-network harikrishnathopugunta805/paytm-microservice:0.0.1-RELEASE

**Docker Compose:**

1. The dash(-) in YAML denotes an array of elements (values only). The other syntax in YAML, of key: value is a dictionary (key-value pairs).

When defined like this:

environment:KEY: valueKEY2: value2

it is a YAML dictionary of key-value pairs.

When defined like this:

environment:- KEY=value- KEY2=value2

it is an array of values, and once again, the = here has no significance from YAML standpoint, it is passed as is to the consumer that processes it (docker-compose binary in this case).

1. Dashes/hyphens denote (unnamed) list elements, thus in the format -<value>
2. Entries without dashes are named child elements, format <name>: <value>

Note:

Dash refers it has value only there is no key value properties

And also refers multiple array vales without child properties

docker-compose up

docker-compose up –d

docker-compose up -d

docker-compose down

docker-compose events

docker-compose images

docker-compose ps -- lists Docker containers.

docker-compose stop

docker-compose pause

docker-compose unpasue

# Docker-Kubernetes-GoogleCloud-AWS-

**SpringBoot-Microservices**

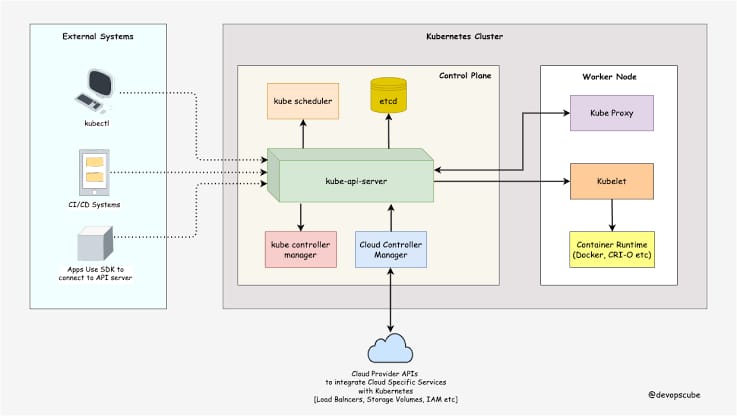
****

**What is Kubernetes?**

Kubernetes is an open-source system that automates the deployment, scaling, and management of containerized applications. Containers are lightweight and portable units of software that run on a shared operating system, isolating the application code and dependencies from the underlying infrastructure. Kubernetes orchestrates the containers across multiple nodes, or machines, in a cluster, and provides features such as service discovery, load balancing, networking, storage, security, and monitoring.

**Why use Kubernetes for microservices?**

* Kubernetes is designed to support distributed and dynamic applications, such as microservices, by providing a declarative and consistent way of defining and updating the desired state of the system. It offers several advantages for microservices deployment, such as scalability, resilience, flexibility, and service discovery and networking. With Kubernetes, you can automatically scale up or down the number of pods based on demand and resources. It also handles failures gracefully by restarting or replacing pods, and implements health checks to monitor the status of services. Additionally, it supports multiple languages, frameworks, and tools so you can choose the best fit for your microservices.
* Furthermore, Kubernetes assigns a unique IP address and DNS name to each pod, creates a virtual network to connect all the pods in the cluster, and allows you to define services that group a set of pods and expose them to other pods or external clients. This enables you to discover and communicate with your microservices regardless of their location or configuration**.**



Create Google Cloud Free Trail Account

What is Clusters?

Actually we will have virtual-servers in cloud for running our applications, Different cloud providers

calls these virtual –servers in Different Terminology

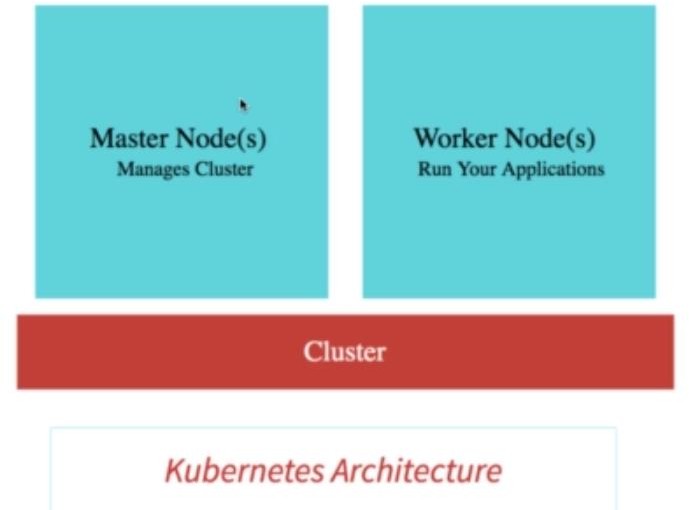
AWS Elastic Compute Cloud Azure  Virtual Servers Google  Compute Engine Kubernetes  Nodes

There are **Master Nodes** also there which will manages the group of nodes

**Cluster** - Master nodes + Nodes

* **Nodes**: Automated node provisioning, scaling, and maintenance
* **Networking**: VPC-native traffic routing for public or private clusters
* **Security**: Shielded GKE Nodes and Workload Identity
* **Telemetry**: Cloud Operations logging and monitoring

**So Cluster is combination Nodes and Master Nodes**



After creating Account, now we need to **enable the Kubernetes Engine** in Google Cloud Account.

create a **Cluster** with proper details(select-area: us-1a and give name of cluster)

**Deployee First Spring boot to Kubernetes Cluster**

First we need to connect to Kubernetes cluster using Google Cloud Shell in Google-Cloud console top right

Click Connect and copy command and paste in Cloud shell, so that you are connected to your Cluster

Kubectl  Kuernetes Control

To create deployment of of our image we need to execute below command

kubectl create deployment hello-world-rest-api --image=in28min/hello-world- rest-api:0.0.1.RELEASE

After deployment created we need to expose this to outside-world using below command

kubectl expose deployment hello-world-rest-api --type=LoadBalancer --port=8080

To check the status go to Services and Ingress and hit end-point

Let’s see some of the commands

The complete command list is also present on the github repository.

<https://github.com/in28minutes/devops-master-class/tree/master/kubernetes#commands>

kubectl get events

This will show all the events like **creation of node,pods,ReplicaSet,Deployment and services**

kubectl get pods

This command will give all the **pods** available in that cluster.

kubectl get replicaset

This command will give all the **ReplicaSets** available in that cluster.

kubectl get services

This command will give all the **Services**available in that cluster.

kubectl get deployment

This command will give all the **deployments** available in that cluster.

When we exigute commands

kubectl create deployment ----- It will create deployments,pods,replicaSet

kubectl expose deployment I t will create the service

These pods,services,ReplicaSet,deployments will helps Kubernetes to achieve all it’s functionalities.

* kubectl logs {makemytrip-microservice-5bc64fd88-2dmls}/podname

**Pods:-**Pods are smallest deployable units, Pod contains multiple containers in it. If u type below command u will get details like pod belongs to which node, it contains how many containers and IP address of the Pod.

* can we run multiple application in single docker container?

-It's ok to have multiple processes, but to get the most benefit out of Docker, avoid one container being responsible for multiple aspects of your overall application. You can connect multiple containers using user-defined networks and shared volumes.

* Kubernetes pods contains one or more containers. A container is a package of software dependencies and resources needed to run an application. The resources include code, libraries, tools, and settings. Pods create an abstraction layer over the containers providing dependencies and resources that allows Kubernetes to manage the containers efficiently.

**Types of Pods:**

**Pods run containers in two ways:**

* **Single containers:** One Kubernetes pod can run a single container. In this case, the container represents an entire application, including all dependencies and resources needed to run the application. Pods with single containers are simple and easy to run.
* **Multiple containers:** Equally, a pod in Kubernetes can run multiple containers on one server. Such cases occur when an application’s programs depend on one another and need to share resources such as files, volume, or data. Kubernetes pods enable this co-dependency by forming an abstraction layer over the containers where they can easily share resources in a controlled environment.
* **Node** – Is nothing but Virtual Machine (server) and node can have multiple pods.
* **Namespace** will provide isolation from pods of the cluster to other pods of the cluster. eg: separate from DEV, TST and PROD

**Relationship between Cluster and Node in kubernetes:**

* Cluster: a collection of one or masters + one or more nodes.

kubectl get pods -o wide

If u want to see the more details of pod then run below command

kubectl describe pod pod-name

**ReplicaSet:-** ReplicaSet ensures that specific number of working at all the times. If u type below command u will get details

kubectl get pods -o wide

Suppose of u delete one pode, then ReplicaSet create one more pode for u in a minute. So the role of the replicaSet is that keep monitoring the Pods are working goog or not.

Suppose if u want 3 pods then u need to tell to ReplicateSet that I need 3 pods as below command

kubectl scale deployment name --replicas=3

**Deployment:-**

While updating from one-version of our application to another version then to see there is **no down-time** we will use Deployments

kubectl get rs -o wide – This will give more details of ReplicaSet

If want to do **new version of hello-world-rest-api** the command to do that is shown below

kubectl set image deployment deployment-name container-name=image-name(new- image)

Example:-

kubectl set image deployment hello-world-rest-api hello-world-rest-api=new- image:TEST

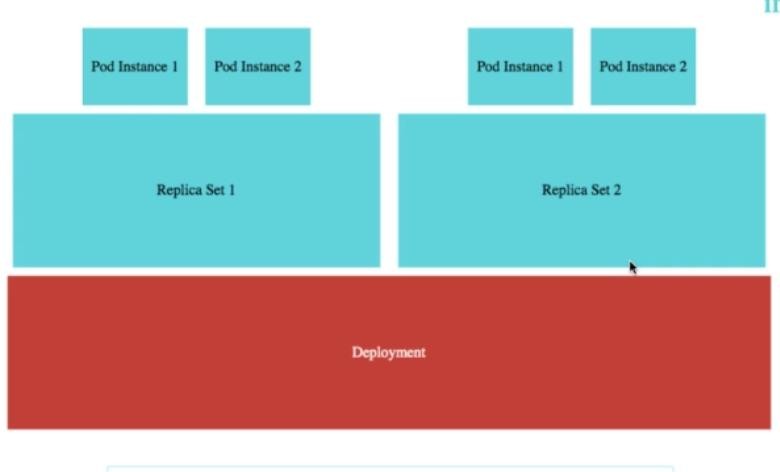
Above I gave wrong image still, old application not went down, Eventhought we made

mistacke in deployment deployment will help in monitoring this.

Correct command for updating to new version

kubectl set image deployment hello-world-rest-api hello-world-rest- api=in28min/hello-world-rest-api:0.0.2.RELEASE

* kubectl scale deployment irctc-microservice --replicas=3



Above diagram explains the Deployment maintains replicas, replicas maintains pods.

#### Service:-

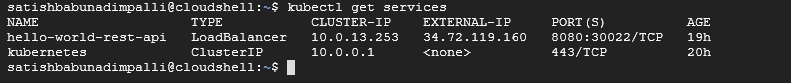
1. Pods are thowable components means we can create and delete the pods, every pod has an IP Address, but when we access the application with one URL only, this maintaining can be done by **Service**(permanent life time url for an application)
2. When we do expose command kubectl expose deployment hello-world-rest-api

Service will create a parmanent IP Address(Domain-IP) and load-balance with Pod IP Addresses

1. When we do expose command, service will create even Load-Balance also.

kubectl get services

This command will give the all the services running in kubernates.



ClusterIP is default service, which can only access from inside cluster, not from external.

-> Rolling updates - v1 -- 50% v2 – 50%, zero downtime, etc.

-> **delete** command will apply for all components pod, replica set, deployments

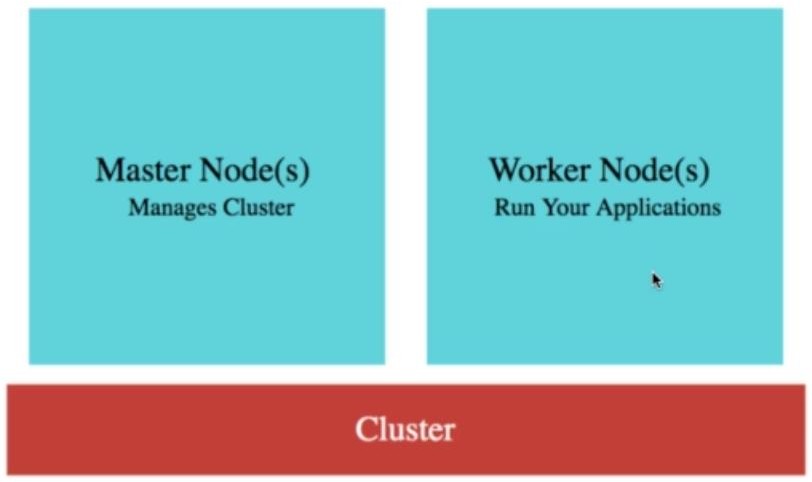
#### GUI For Kubernetes Operation:-

There is **WorkLoads** Tab there, we can perform all the operation which we have done in CommandLine.

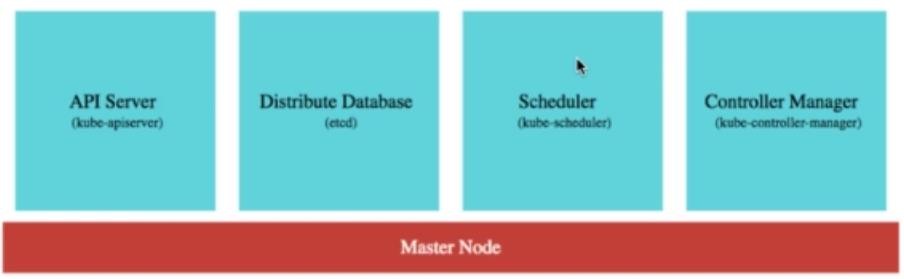
Till now we are discussing all except Nodes, Let’s discuss about nodes now

#### Nodes:-

On top of cluster we have 2 types of Nodes 1)Master Node(s)—Manages Cluster 2)Worker Node(s)—Runs the Applcation.

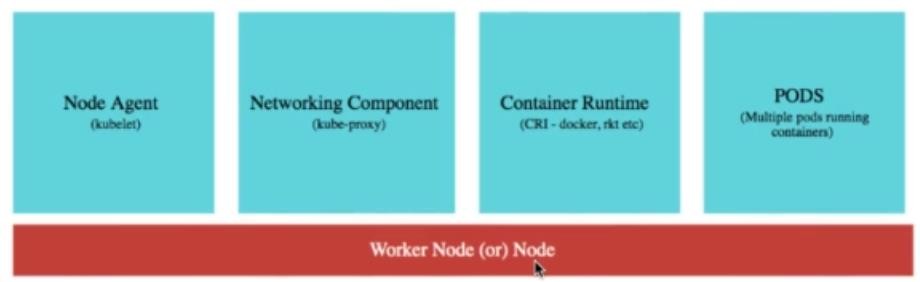


Master Node important components



kubectl get componentstatuses Will give status of master nodes

Worker Nodes



Create simple application and create docker-image for that and push docker-image to Docker-hub

Previously we used to take image directly from Docker-Hub into Kubernateeees Engine, We can even take from local as well, To do that we need to install 2 tools **GCloud** and **kubectl**

After GCloud install successfully, If u want to Login

https://cloud.google.com/blog/products/containers-kubernetes/kubectl-auth-changes-in-gke

gcloud components install gke-gcloud-auth-plugin

gcloud auth login

After install completes of kubectl, We have to deploy our local application to kubernates engine using above installed tools.

Connect to the GCloud (Copy connect code from kubernetes to paste in cmd)

To send our local-image execute below command

kubectl set image deployment deployment-nm container-nm=docker-image-nm

Ex:- kubectl set image deployment hello-world-rest-api hello-world-rest- api=satish4228/hello-world-rest-api:0.0.4-SNAPSHOT

Below command will give the History of deployment

kubectl rollout history deployment deployment-nm

You can undo the deployment to specific revision using below command

kubectl rollout undo deployment deployment-nm –-to-revision=3

If you want to see the logs of the application, then use below command

kubectl get pods

kubectl logs pod-nm

**Kubernetes YAML Configuration for deployment:**

Till now we have used multiple commands to make deployment, Rather Kubernetes provides the YAML Configuration for deployment.

In real world application can be deployed through YAML Configuration, let’s see YAML Configuration

kubectl get deployment deployment-nm

Will give respective deployment details.

You can get deployment information in the form of YAML using below command

kubectl get deployment deployment-nm -o yaml

kubectl get deployment deployment-nm -o yaml > deployment.yaml ---This will store deployment info in deployment.yaml file into local(pwd)

You can update this yaml file so modifications will be done in kubernetes by below command after the changes has been done.

kubectl apply –f deployment.yaml

You can get the service information also in the form of YAML

kubectl get service service-nm -o yaml

kubectl get service service-nm -o yaml > service.yaml ---This will store Service info in service.yaml file into local(pwd)

You can update this yaml file so modifications will be done in kubernetes by below

command after the changes has been done.

kubectl apply –f service.yaml

We can combine then to One file supported by ---(3dashes)

apiVersion: extensions/v1beta1 kind: Deployment

metadata: annotations:

deployment.kubernetes.io/revision: "3" creationTimestamp: "2020-05-08T16:47:55Z" generation: 4

labels:

app: hello-world-rest-api name: hello-world-rest-api namespace: default resourceVersion: "395478"

selfLink: /apis/extensions/v1beta1/namespaces/default/deployments/hello- world-rest-api

uid: aa28bbe2-914b-11ea-89e7-42010a8000a4 spec:

progressDeadlineSeconds: 600

replicas: 3

revisionHistoryLimit: 10 selector:

matchLabels:

app: hello-world-rest-api strategy:

rollingUpdate: maxSurge: 25%

maxUnavailable: 25% type: RollingUpdate

template: metadata:

creationTimestamp: null labels:

app: hello-world-rest-api spec:

containers:

- image: satish4228/hello-world-rest-api:0.0.4-SNAPSHOT imagePullPolicy: IfNotPresent

name: hello-world-rest-api resources: {}

terminationMessagePath: /dev/termination-log terminationMessagePolicy: File

dnsPolicy: ClusterFirst restartPolicy: Always schedulerName: default-scheduler securityContext: {} terminationGracePeriodSeconds: 30

status: availableReplicas: 3 conditions:

- lastTransitionTime: "2020-05-09T09:41:08Z" lastUpdateTime: "2020-05-09T09:41:08Z" message: Deployment has minimum availability. reason: MinimumReplicasAvailable

status: "True" type: Available

- lastTransitionTime: "2020-05-08T16:47:55Z" lastUpdateTime: "2020-05-09T22:53:23Z" message: ReplicaSet "hello-world-rest-api-

55cf764d97" has successfully progressed. reason: NewReplicaSetAvailable status: "True"

type: Progressing observedGeneration: 4

readyReplicas: 3

replicas: 3

updatedReplicas: 3

---

apiVersion: v1 kind: Service metadata:

creationTimestamp: "2020-05-08T16:55:15Z" labels:

app: hello-world-rest-api name: hello-world-rest-api namespace: default resourceVersion: "19771"

selfLink: /api/v1/namespaces/default/services/hello-world-rest-api uid: b08bad4b-914c-11ea-89e7-42010a8000a4

spec:

clusterIP: 10.0.13.253 externalTrafficPolicy: Cluster ports:

- nodePort: 30022

port: 8080 protocol: TCP targetPort: 8080

selector:

app: hello-world-rest-api sessionAffinity: None type: LoadBalancer

status: loadBalancer:

ingress:

- ip: 34.72.119.160

In Above there are some un-necessary tags are there we will remove those, finally yaml

file looks below

apiVersion: extensions/v1beta1 kind: Deployment

metadata: labels:

app: hello-world-rest-api name: hello-world-rest-api namespace: default

spec:

replicas: 3 selector:

matchLabels:

app: hello-world-rest-api strategy:

rollingUpdate: maxSurge: 25%

maxUnavailable: 25% type: RollingUpdate

template: metadata:

labels:

app: hello-world-rest-api

spec:

containers:

- image: satish4228/hello-world-rest-api:0.0.4-SNAPSHOT imagePullPolicy: IfNotPresent

name: hello-world-rest-api restartPolicy: Always terminationGracePeriodSeconds: 30

---

apiVersion: v1 kind: Service metadata:

labels:

app: hello-world-rest-api name: hello-world-rest-api namespace: default

spec:

ports:

- nodePort: 30022

port: 8080 protocol: TCP targetPort: 8080

selector:

app: hello-world-rest-api sessionAffinity: None

type: LoadBalancer

Suppose If want to delete all which are having tag-name, use command

kubectl delete all -l app=hello-world-rest-api

Now the pods,services,deployments are delete with respective tag-name, now we will apply our deployment.yaml file to recreate the all the things again.

**Note:-** Please be in deployment.yaml file directory

kubectl apply –f deployment.yaml

Suppose If u want to know how many resources(pods,deployments,services, etc…) use below command

kubectl get all

#### Understanding Kubernetes YAML Configuration:-

Inside the specification(**spec)** of **deployment**, **template** is defining the **pod definition**.

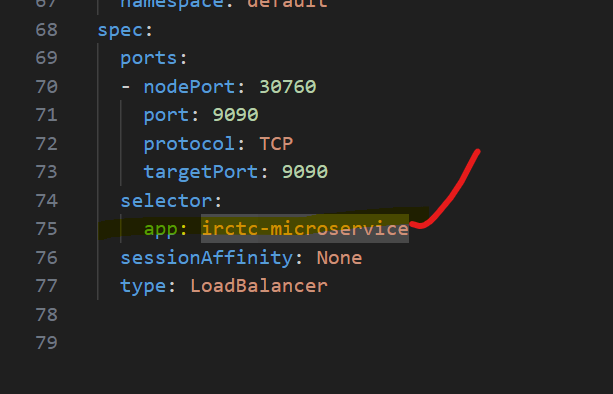
Labels are key/value pairs that are attached to objects such as Pods. Labels are intended to be used to specify identifying attributes of objects

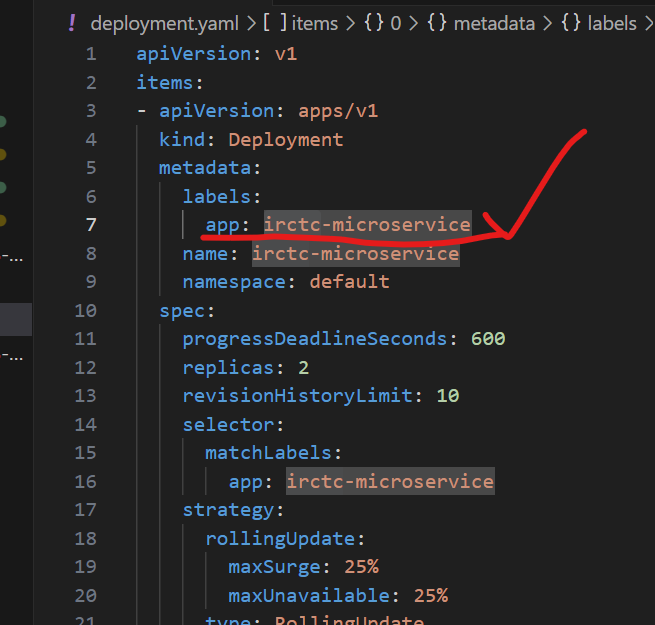
* **Template** is nothing but **pod**
* Pod is nothing but one virtual machine
* Services can talk to pods only not containers
* **Label** is used for referencing this **pod**(IRCTC) belongs to respective **deployment**.
* **Label** is unique name/ key name - which tell about object (pod, service, deployment) and selector is going to refer the label which should match to refer the parent one.

If they didn't match, any pod created wouldn't match the selector, and I'd imagine K8s would go on creating new pods until every node is full?

* Your deployment will not proceed, it will fail with error message "selector" does not match template "labels". No pod will be created.
* ...it feels like any label in the template metadata should automatically be in the selector matchLabels.
* Labels under template. metadata are used for many purposes and not only for deployment.
* Service ->**selector-match label** refers to deployment **label**.
* Selector refers the label where ever they matched it.

For example : service is referring the deployment label with help of selectors.





template:

metadata: labels:

app: hello-world-rest-api spec:

containers:

- image: satish4228/hello-world-rest-api:0.0.4-SNAPSHOT imagePullPolicy: IfNotPresent

name: hello-world-rest-api restartPolicy: Always

terminationGracePeriodSeconds: 30

app:hello-world-rest-api tells pod’s name

As we know that pod contains the **Containers.** Containers tells the info about container in the Pods.

- image: satish4228/hello-world-rest-api:0.0.4-SNAPSHOT imagePullPolicy: IfNotPresent

name: hello-world-rest-api

This – indicates array we can have multiple like below

* image: satish4228/hello-world-rest-api:0.0.4-SNAPSHOT imagePullPolicy: IfNotPresent

name: hello-world-rest-api

* image: satish4228/hello-world-rest-api:0.0.4-SNAPSHOT imagePullPolicy: IfNotPresent

name: hello-world-rest-api

ImagePullPolicy tells that pulling from docker-hub (**IfNotPresent**Then only pull,

**Always** Always check the Docker Repo)

**name**tells the name of the container, **teminationGracePeriodSeconds**  When container is stopped clean-up time is 30 sec

Replicastells how many replicas

maxSurge  Tells Max pod’s used during release maxUnavailiable  Max unavailable during release Service Tags:-

**sessionAffinity** – tells if we have session configuration then we need to one use/pod

otherwise session will be lost, So to main that this tag will be useful.

Suppose if I want reduce down-time, then we need to Use one property as part of deployment (minReadySeconds)

spec:

replicas: 3,

minReadySeconds: 45, selector:

matchLabels:

app: hello-world-rest-api

This **minReadySeconds** will make avaliable old pods 45 seconds after kickoff of new pods

This quick-fix only Actually we need to use readiness and liveness probes for perfect fix.

**Liveness probes**: These probes help you evaluate whether an application that is running in a container is in a healthy state

-> If you want to all pods, services, deployments specific to particular one app, we have to use below command where **–l** represents label --- as discussed it is SUR NAME for any person.

**kubectl delete all -l app=irctc-microservice**

#### ReplicaSet In Depth:-

ReplcaSet can have with Deployment, just replace deployment with ReplicaSet tag, which shown below

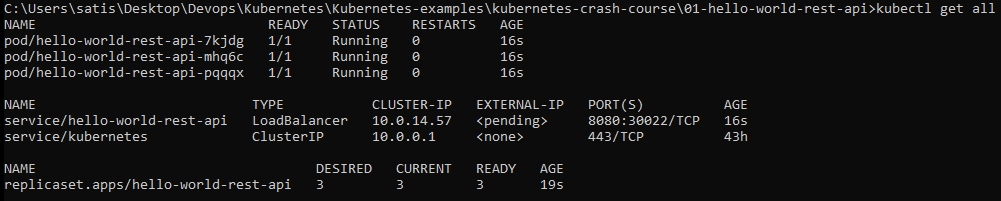
kind: ReplicaSet

Now delete existing deployments

Kubectl delete all –l app=hello-world-rest-api

Now Apply deployment with below command

Kubectl apply –f deployment.yaml Then it will tell Unknown property Strategy, As we know that deployment is responsible for rollingUpdate, but replicaSet is responsible only maintain pods not rollingUpdates. So delete it.And apply one-more time.



Above tells that replicaSet also creates **pods**

Suppose, If we go for new release then replicas will not update the pods, it just see that wheather pods are working fine or not, So we manually delete one of pod than replicaSet comes into picture and update to new release for that pod only not all the pods.

**So that is the reason we have go for Deployment rather than ReplicaSet**

**How do Pods communicate in Kubernetes? :**

* There are some basic facts about pod-to-pod communication, which generally hold true for all Kubernetes clusters:
* **In Kubernetes, each Pod has an IP address. A Pod can communicate with another Pod by directly addressing its IP address, but the recommended way is to use Services. A Service is a set of Pods, which can be reached by a single, fixed DNS name or IP address.**
* In reality, most applications on Kubernetes use Services as a way to communicate with each other. Using a Service is much more **flexible than addressing another Pod directly, because Pods can be restarted frequently**, which means that addressing them by name or
* <https://www.tutorialworks.com/kubernetes-pod-communication/>

**Overview:-** Create docker-image create yaml file  deploy to kubernates

Example of YAML file:-

apiVersion: extensions/v1beta1 kind: Deployment

metadata: labels:

app: todo-web-application-h2 name: todo-web-application-h2 namespace: default

spec:

replicas: 3 selector:

matchLabels:

app: todo-web-application-h2 strategy:

rollingUpdate: maxSurge: 25%

maxUnavailable: 25% type: RollingUpdate

template: metadata:

labels:

app: todo-web-application-h2 spec:

containers:

- image: satish4228/todo-web-application-h2:0.0.1-SNAPSHOT imagePullPolicy: IfNotPresent

name: todo-web-application-h2 restartPolicy: Always terminationGracePeriodSeconds: 30

---

apiVersion: v1 kind: Service metadata:

labels:

app: todo-web-application-h2 name: todo-web-application-h2 namespace: default

spec:

ports:

- nodePort: 30022

port: 8080 protocol: TCP targetPort: 8080

selector:

app: todo-web-application-h2

sessionAffinity: ClientIP type: LoadBalancer

**Understand environment variable create by Kubernetes for services:**

* By default, Kubernetes provides environment variables for each service to know about other services in cluster.

**Example:**

Name followed for each service:

* ServiceName\_SERVICE\_HOST
* ServiceName\_PORT

**Problem**:

* But we have some problem with above approach, that is if dependency app(irctc) is not available while starting other application(makemytrip/paytm) so it won’t register at the starting of application. So it is not recommending way.

**Kubernetes doesn’t register dependent environment variables after the service started.**

* So we can provide our own environment variable along with value through deployment.yaml file to kubernetes cluster.

If you observe we have entered the database-details like user-name and password in yaml file itself, which is not good. To make confidential Kubernetes provides the **Config Maps for Centralized Configuration.**

Before using Config Map, First we need to create it using below command, We want to store literals so **--from-literal**

kubectl create configmap to-do-web-application-config –from- literal=RDS\_DB\_NAME=todos

To see details which has been inserted in ConfigMap use below command

kubectl describe configmap/name-of-cinfig-map

To replace this value in the application.yaml file we need to do comment value section and write below code, So that it wil fetch the details from kubernetes configmap having name

#value :

valueFrom:

configMapKeyRef: key:RDS\_DB\_NAME

name:to-do-web-application-config

Apply changes to kubernetes

Kubectl get apply –f to-do-web-allication-deployment.yaml

Similarly modify all the configuration except **password.** Because password we will store in kubernetes secrets.

For creating the secret we need to use below command

kubectl create secret generic to-do-web-application-secret –from- literal=RDS\_PASSWORD=password-value

To see details which has been inserted in Secret use below command

kubectl describe configmap/name-secret

And modify the password value in yaml file as shown below

#value :

valueFrom:

secretKeyRef:

key:RDS\_PASSWORD

name:to-do-web-application-secret

Finally apply the changes to kubernetes with below command

Kubectl get apply –f to-do-web-allication-deployment.yaml

We have made the change **type=LoadBalancer** that means out-side people can access it, but always we may not want that feature

To do this we need to delete this service and create same one more time, So modify the **type:ClusterIP**

**Namespaces :**

A [Kubernetes](https://www.aquasec.com/cloud-native-academy/kubernetes-101/kubernetes-complete-guide/) namespace helps separate a cluster into logical units. It helps granularly organize, allocate, manage, and secure cluster resources. Here are two notable use cases for Kubernetes namespaces:

* **Apply policies to cluster segments**—Kubernetes namespaces let you apply policies to different parts of a cluster. For example, you can define resource policies to limit resource consumption. You can also use container network interfaces (CNIs) to apply network policies that define how communication is achieved between pods in each namespace. Learn more about [Kubernetes networking](https://www.aquasec.com/cloud-native-academy/kubernetes-101/kubernetes-networking/).
* **Apply access controls**—namespaces let you define role-based access control (RBAC). You can define a role object type and assign it using role binding. The role you define is applied to a namespace, and RoleBinding is applied to specific objects within this namespace. Using this technique can help you improve the security of your cluster.

**Default Kubernetes namespaces**

Here are the four default namespaces Kubernetes creates automatically:

* **default**—a default space for objects that do not have a specified namespace.
* **kube-system**—a default space for Kubernetes system objects, such as kube-dns and kube-proxy, and add-ons providing cluster-level features, such as web UI dashboards, ingresses, and cluster-level logging.
* **kube-public**—a default space for resources available to all users without authentication.
* **kube-node-lease**—a default space for objects related to cluster scaling.

**Custom Kubernetes namespaces**

Admins can create as many Kubernetes namespaces as necessary to isolate workloads or resources and limit access to specific users. Here is how to create a namespace using kubectl:

kubectl create ns mynamespace

* <https://www.aquasec.com/cloud-native-academy/kubernetes-101/kubernetes-namespace/>

**Ingress and egress​:**

**Ingress**

From the point of view of a Kubernetes pod, ingress is incoming traffic to he pod, and egress is outgoing traffic from the pod. In Kubernetes network policy, you create ingress and egress “allow” rules independently (egress, ingress, or both).

**Readiness:**

* Let’s imagine that your app takes a minute to warm up and start. Your service won’t work until it is up and running, even though the process has started. You will also have issues if you want to scale up this deployment to have multiple copies. A new copy shouldn’t receive traffic until it is fully ready, but by default Kubernetes starts sending it traffic as soon as the process inside the container starts. By using a readiness probe, Kubernetes waits until the app is fully started before it allows the service to send traffic to the new copy.
* <https://storage.googleapis.com/gweb-cloudblog-publish/original_images/google-kubernetes-probe-readiness6ktf.GIF>

**Liveness:**

* Let’s imagine another scenario where your app has a nasty case of deadlock, causing it to hang indefinitely and stop serving requests. Because the process continues to run, by default Kubernetes thinks that everything is fine and continues to send requests to the broken pod. By using a liveness probe, Kubernetes detects that the app is no longer serving requests and restarts the offending pod.
* <https://storage.googleapis.com/gweb-cloudblog-publish/original_images/google-kubernetes-probe-livenessae14.GIF>

**minReadySeconds: (deployment version image)**

To main old pods during new version deployment time

**Readiness vs Liveness:**

<https://cloud.google.com/blog/products/containers-kubernetes/kubernetes-best-practices-setting-up-health-checks-with-readiness-and-liveness-probes>

**SpringBoot-Microservices-Docker-Kubernetes**

Create images for both CES and CCS and push to Docker-Hub

Now deploy the application to kubernetes engine using deployment.yaml fie, first- currency-exchange, second-currency-conversion

**Note:-** Here interesting fact that how Service Discovery of currency-exchange happens by currency-conversion

That is kubernetes creates Environment variables for us by default with appending of service\_host

SERVICE\_NM\_SERVICE\_HOST, So if u used this in Feign then we can automatically call other service in the kubernates.

@FeignClient(name = "currency-exchange", url = "${CURRENCY\_EXCHANGE\_SERVICE\_H[OST:http://](http://localhost/)loca[lhost](http://localhost/)}:8000")

But with this we have a problem that by the time currency-conversion is ready if exchange is not ready, then URL goes wrong

So for this solution is **Service Discovery** provided by Kubernetes

Kubernetes service Discovery resolves url like [http://app-name](http://app-name/)

Ex: [http://currency-exchnge](http://currency-exchnge/) Then kubernetes automatically check for URL resolving based on DNS.

But to generate URL like this we can have some Environment variables and we can get name as shown below.

@FeignClient(name = "currency-

exchange", url = "${CURRENCY\_EXCHANGE\_URI:[http://localhost](http://localhost/)}:8000")//

At pod level give this

env:

- name: CURRENCY\_EXCHANGE\_URI

value: [http://currency-exchange](http://currency-exchange/)

We can solve this problem by Kubernetes-Spring-Profiles

1. create env like below
2. create property file like application-kubernetes.properties

CURRENCY\_EXCHANGE\_URI=[http://currency-exchange](http://currency-exchange/)

env:

- name: SPRING\_PPROFILES\_ACTIVE

value: kubernetes

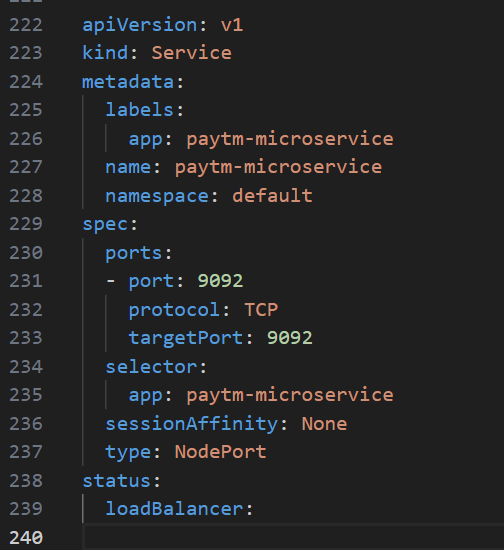
**Ingress :**

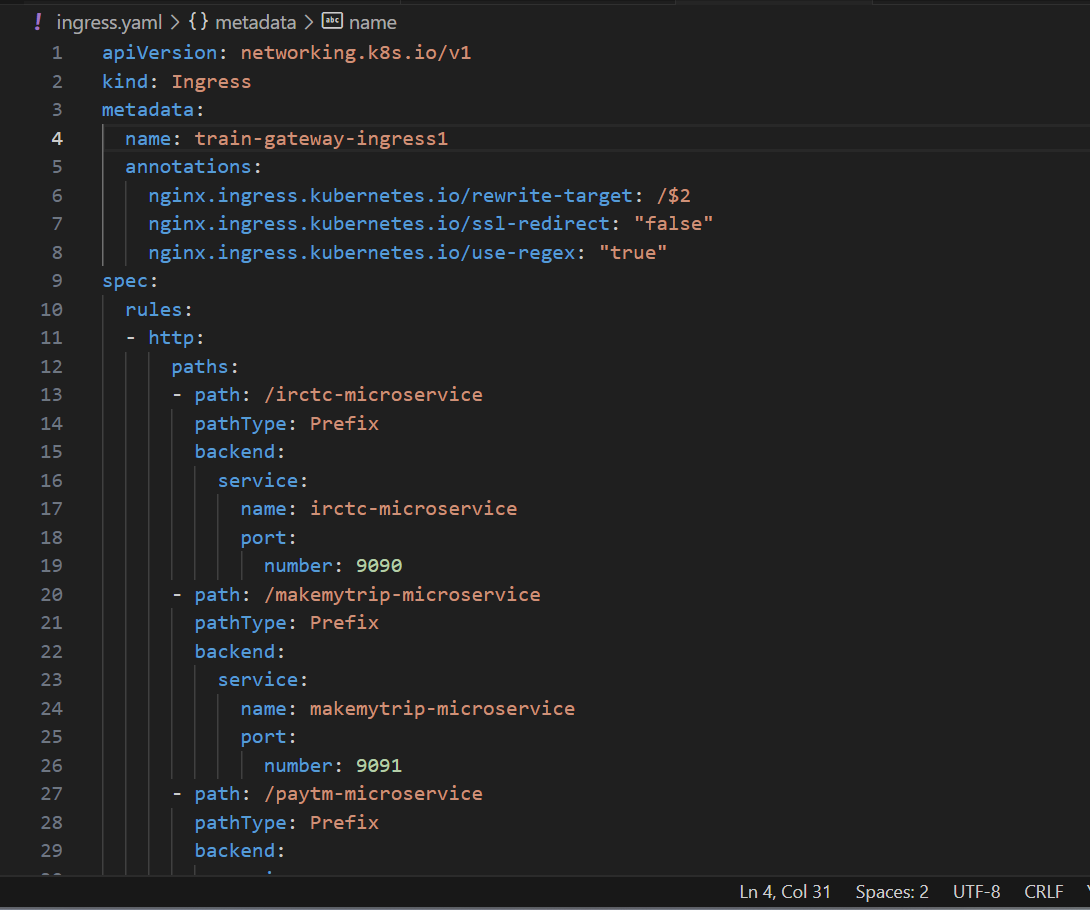
In kubernetes creating the LoadBalancer is very costlier operation (high availability), it always running monitoring etc…

So that kubernetes introduces concept called **Ingress**, in this LoadBalancer will be there this will distribute the load to multiple micro-services by redirecting to the actual services.

Ingress is also same like Service, For this we have **Ingress.yaml** file will be there apply to kubernetes(Ingress creation will take about 10 minutes)

Change type from **LoadBalaner to NodePort** in services(deployment.yaml),and apply to kubernetes, so Our ingress will handle the requests.





**Note**: Up to **here74** videos completed in devops course.

#### Spring-Cloud-Kubernetes

Now we can some of the features of Spring Cloud in-addition to kubernetes features like we can use Ribbon(Load Balancing)

Import 06-currency-conversion-microservice-cloud to workspace

<dependency> <!-- CHANGE -->

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-ribbon</artifactId>

</dependency>

<dependency> <!-- CHANGE -->

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-kubernetes-all</artifactId>

</dependency>

Modify the Feign URL to, remaining all comment

@RibbonClient(name = "currency-exchange")

Here Ribban Client used for Client-Side Load-Balancing and Feign Client is used for Calling

for othe service rest-api’s,

Create the image and push to docker-hub, and apply to kubernates

I will got error that

Error creating bean with name 'ribbonLoadBalancingHttpClient' defined in org.s pringframework.cloud.netflix.ribbon.apache.HttpClientRibbonConfiguration: Unsa tisfied dependency expressed through method 'ribbonLoadBalancingHttpClient' pa rameter 2; nested exception is org.springframework.beans.factory.BeanCreationE

xception

It telling that Ribbon is not able to talk to Kubernates and telling that configured service

account does not have access.

There is service account which already configured for us in K8s, To see service account use below command

Kubectl get serviceaccount

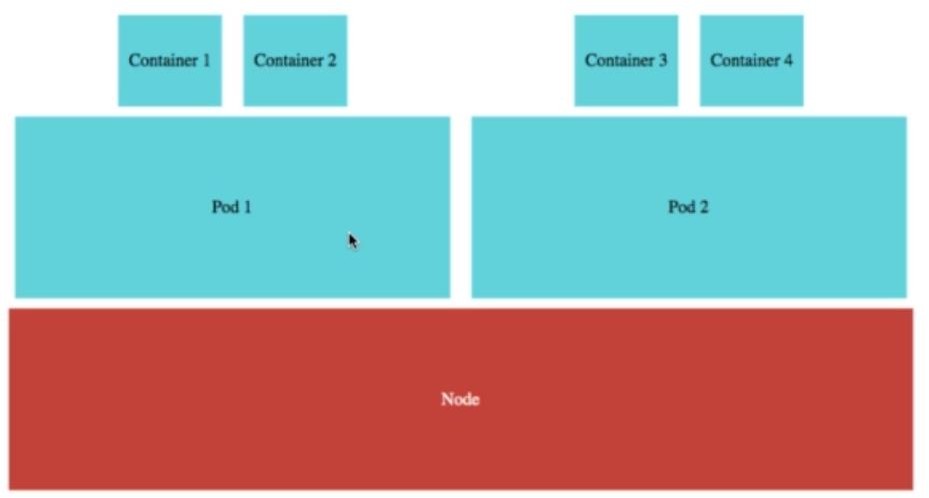
For Exposing our services to outside from kubernetes we need to give view permissions to service account by configuring **rbac.yaml** file.

Apply

Kubectl apply –f 02-rbac.yaml

#### Auto-Scaling:-

Till now we manually increasing or decreasing the no of instances throught replicas=2,3,etc…



Scaling in-sence that either increase Nodes if node capacity is over(**Cluster Auto-Scaling**), increase the no-of pods(**horizontal pod auto-scaling**), increase RAM or memory of pod(**vertical pod autoscaling**)

Now we will try **Horizantal pod Auto-Scaling** with currency-exchange-service

1)Have no-of replicas=1, for testing keep memory very less cpu 100m,500m 2)To check CPU usage of use below command

1. Then you will get cpu usage, u will check that if cpu usage >20 (In prod we will give >70 or 80) we will autoscale with below command.
2. Auto-scaling will kicked if cpu is more than given

Will give complete details of **horizontal-pod-autoscaling 5)**you can take the yaml configuration for hpa

kubectl get hpa

kubectl autoscale deployment currency-exchange –-min=1 –-max=3 –-cpu- percent=50

kubectl top pod-nm

How to delete Kubernetes Cluster?

Delete cluster from GUI of Kubernetes Web Site

#### Google-Stackdriver

As we know that we have Distributed tracing **Zipkin Distributed Tracing** for tracing the all micro-services, Similarly Google also has Popular Distrubuted Tracing **GoogleStackDriver**

For this we need to enable GoogleStackDriver option at the time of creating Cluster, So Let’s create a new cluster for with enablement of stackDriver.

Let’s import the project 7 and 8, we added the dependency

<dependency> <!-- CHANGE -->

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-gcp-starter-trace</artifactId>

</dependency>

<dependency> <!-- CHANGE -->

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-gcp-starter-logging</artifactId>

</dependency>

Write some application properties

#CHANGE - In production reduce sampling-rate to 0.01 spring.sleuth.sampler.probability=1.0

spring.cloud.gcp.trace.enabled=false

And enable Monitoring in Kubernetes in Deployment.yaml

env: #CHANGE

- name: SPRING\_CLOUD\_GCP\_TRACE\_ENABLED

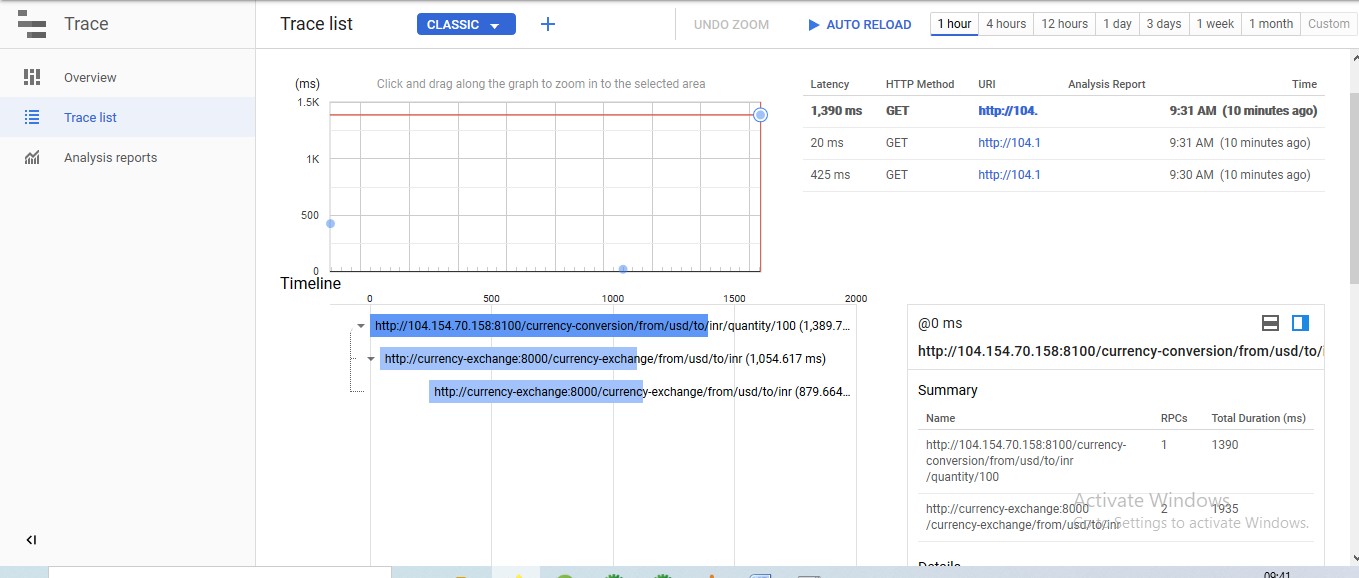
value: "true"

We need to Enable API and services in kubernetes(Serach in kubernetes search bar API & Services) >> Click Enable API AND SERVICES >> Make Sure all StackDriver API’s are enabled.

Now we need to connect new Cluster, Copy the connect Command and connect from CLI, And send deployment to Kubernetes

Kubectl apply –f deploymwnt.yaml

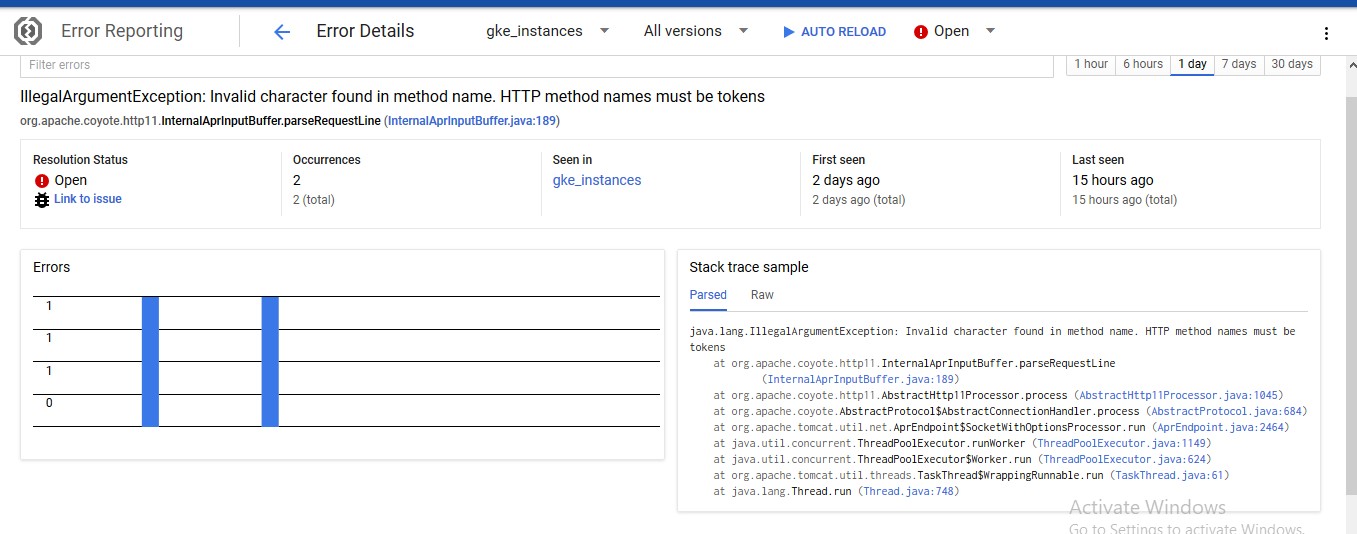
Now hit the CCS then, Search **stackDriver** in Kubernetes search >> Go to StackDriver >>Go to Trace >> TraceList



There we can find request trace, where it went and time all other info about each request. This StackDriver internally used ZipKin Distributed System only

#### Error Reporting

StackDriver can also report errors to us even. Serach Error Reporting Search bar



U can see even Logging of the application also In StackDriver. You can see logs in container-wise.

#### Istio:-

1. For Running istio let’s create a good powerful Cluster with more vCPU, GCloud provides the iStio but that is Beta version only all the features may not support. So we will use superate software for istio

#### What is istio ?

Istio is Service Mesh means suppose if we have 10 micro-services, if we need to

logging,tracing,monitoring etc… for 10 micro-services then we need to implement for ll the services, Rather we can have one service throught which only all the requests for all micro- services will goes, This service is called Service Mesh which is implemented by Istio.

Now open the new created cluster and open cloud terminal and connect new cluster, Now we will run some scripts in cloud shell

So no we need to install softwares related to istio in our cluster, for this we need to apply some commands

kubectl create namespace istio-system

curl -L https://git.io/getLatestIstio | ISTIO\_VERSION=1.2.2 sh - cd istio-1.2.2

for i in install/kubernetes/helm/istio-init/files/crd\*yaml; do kubectl apply - f $i; done

helm template install/kubernetes/helm/istio --name istio -- set global.mtls.enabled=false --set tracing.enabled=true --

set kiali.enabled=true --set grafana.enabled=true --namespace istio- system > istio.yaml

kubectl apply -f istio.yaml

Now let’s connect new cluster from the local

Move to 11-istio-script directory, Now apply to kubernates 01-hello.yaml. If u want u can change the image name in yaml file

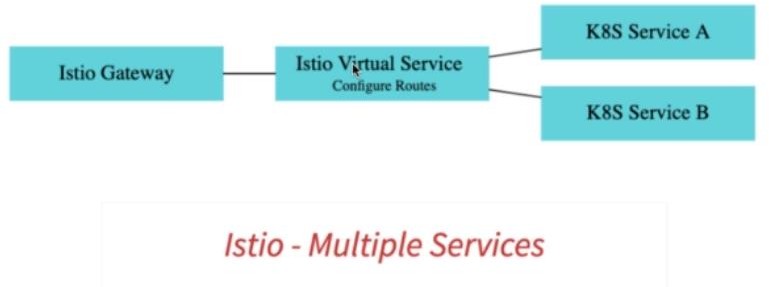
Now u will get error that enable istio in old created pods using below command

kubectl label namespace default istio-injection=enabled

Now apply to kubernetes one more time

But this we con’t access from outside, To access from outsie

For the access of allocation from out-side directly, let’s make user to hit these services first, bcz if we hit gate-way, virtual service first then we will have many features like tracing, monitoring etc…



In above diagram we have 2 service with istio configured and gateway also configured, We will get more benefits if we have these.Let’s configure them

We have gateway and virtual-service configuration, So go head aplly to kubernetes

kubectl apply -f 02-creating-http-gateway.yaml gateway.networking.istio.io/http-gateway created

kubectl apply -f 03-creating-virtualservice-external.yaml virtualservice.networking.istio.io/helloworld-virtual-services created

Access the External-Gateway of Istio, From there u will redirt to normal micro-serivce To get exteral IP address use below-command

kubectl get svc istio-ingressgateway --namespace istio-system

hit the IP

<http://35.193.18.35/hello-world>

Now we will have micro-service instead of rest-api and we will check distributed tracing using Istio.

Do mvn clean install and push CE service to docker-hub for 09 project. And apply to kubernetes.

We have kiali, Using which we can observe the service-mesh helath, how many micro- services does services-mesh has etc…..

#### Helm

Helm is The package manager for Kubernetes, means we have lot of yaml files that we exigutes as part of application deployment, helm makes the all yaml files to once place configuring, So it is easy and single click for applications deployment.

To use helm we need to install server in k8s and client server in local machine

### Kubernetes with AWS:-

Till now we worked on Kubrenetes with GCloud, Now we are working Kubenetes with AWS. As we know that kubenetes supports multiple cloud platforms

Manage Kubernetes on AWS is called **EKS**(Elastic Kubernetes Service). Running kubernetes with AWS is not comes under free tier. It is chargable.

 Install AWS CLI to connect to AWS-Console

After installing AWS-Clis, You need to install eksctl, To play with kubernetes on AWS.

After installing all the softwares, Run below command to configure a IAM User.

aws configure

Now we will create kubernetes cluster on AWS, To create kubernetes cluster on AWS u need to run below command.

eksctl create cluster --name devops-practise-cluster --nodegroup-name devops- practise-cluster-node-group --node-type t2.medium --nodes 3 --nodes-min 3 -- nodes-max 7 --managed --asg-access

--asg—access  is for enabling auto-scaling

--managed  if any new patch, to be updates kubernetes will manage those

Then your kubernetes cluster will get’s created. It will take around 15-20 minutes to create Cluster on AWS.

You can see same cluster will be available in AWS –EKS GUI

You can use same commands which u used at Gloud-Kubernetes to do all operrations on kubernetes.

Now we will deploy hello-world-allication to EKS, Same commands which use earlier.

kubectl create deployment name-of-deployment --image=image-name-from-docker- hub

ex:-

kubectl create deployment hello-world-rest-api --image=hello-world-rest- api:0.0.1:RELEASE

Expose deployment

kubectl expose deployment hello-world-rest-api –-type=LoadBalancer –-port=8080

Let’s deploy web application with MySQL into AWS-Kubernetes.

Create docker-compose for both mysql and web-app and generate deployment files using

#### Kompose

Let run the Mysql using below-command

kubectl apply -f mysql-database-data-volume-persistentvolumeclaim- aws.yaml,mysql-deployment.yaml,mysql-service.yaml

For MySQL Persistent volume- in AWS we will use StorageClass and we need to tell where it need to store(aws-ebs)

Run web-app with below command

kubectl apply -f config-map.yaml,secret.yaml,todo-web-application- deployment.yaml,todo-web-application-service.yaml

For deleting all the things which are having the tag-name using below command

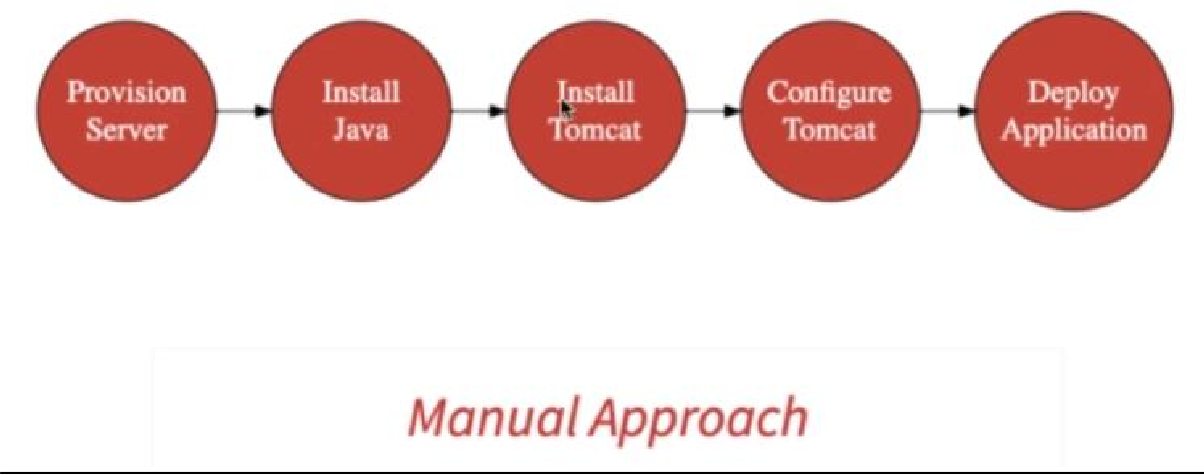
kubectl delete all -l app=hello-world-rest-api

Let’s deploy basic micro-services with below commands

kubectl apply -f 04-currency-exchange-microservice-basic/deployment.yaml kubectl apply -f 05-currency-conversion-microservice-basic/deployment.yaml

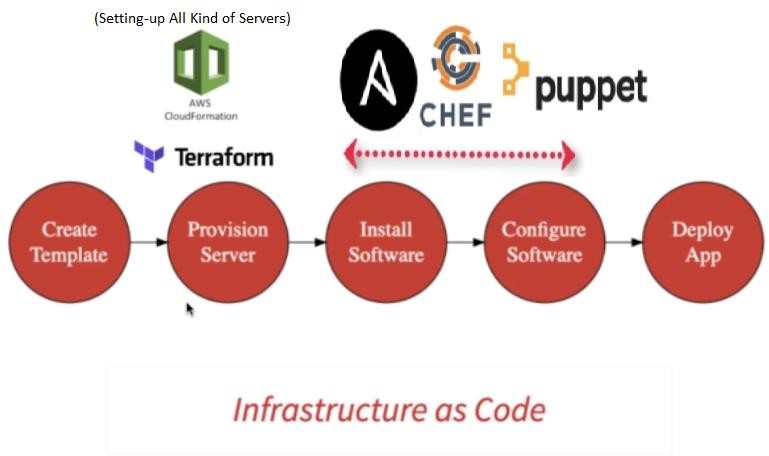
Remaining all configuration things are same for both Google and AWS.

#### Infrastructure As Code:



If u see manual Approach of deploying the Application will be very difficult, But If you can see Infrastructure as code all the deployment would be automated.

**Infrastructure as code: -** Making complete application deployment infrastructure in the form code means like code how we can run, similarly if we run code complete infrastructure for application deployment will come.



In infrastructure as code 2 things are very important, 1)Provisioning Server 2)Install & Configure Softwares

1. Provisioning the server means **Setting-up all the servers(kubernetes,aws,database- servers etc…).** Here teraraform is helps in creating the servers using some templates.
2. Install & Configure servers means installing software’s which require for applications run.

#### Terraform:-

Terraform is a infrastructure as code tool, Creating the servers on Kubernetes-Nodes and AWS-ES will be automated through this tool.

Pre-requesits :

* 1. AWS Account 2)Terraform Installed

To be able to connect AWS u need to have key and password, This key and password we can get from IAM(I am User)

Go to User >> add-user >> only-programatic-access >> existing policies (Administrative). Save your access key details, if u lost u con’t retrieve back, u have to create new user.

To connect AWS from Terraform we need to provide the details to terraform, we can provide the details to terraform in below no-of ways

The AWS provider offers a flexible means of providing credentials for authentication. The following methods are supported, in this order, and explained below:

* Static credentials
* Environment variables
* Shared credentials file
* EC2 Role

Using static credentials are good-practise, So we go and use Environment varibles

set AWS\_ACCESS\_KEY\_ID= AKIARZV6GAWYYJM54GPX

set AWS\_SECRET\_ACCESS\_KEY=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

#### Amazon S3:-

 Amazon S3 or Amazon Simple Storage Service is a service offered by Amazon Web Services that provides object storage through a web service interface. It is scalable storage infrastructure.

It can store files, like with keyValue mapping

Let’s create one bucket (bucket name should be unique across all the aws).

Now we will create the bucket from Terraform

1. Whenever if u want create anything from terraform u need to as **resource(This is the Object that u would want to maintain in the cloud),**for resource u need to provide 2 important things 1)type of resource 2) Terraform reference name for corresponding resource.
2. Inside of {} we need to give bucket-name

In terraform if u want to execute some code, you need to do 2 things 1)plan 2)execute **1)Plan:-** In plan, we will look **what will happen if execute (Command :** terraform plan**) 2)execute:-** In execute, we will **we will execute terraform code.( Command :** terraform apply**)**

If u are successfully able to create a bucket from terraform, then you are able to see

**terrafoem.tstate** file auto-generated.

**Terraform States:-**

Terraform has 3 states **1)DESIRED 2)KNOWN 3)ACTUAL**

There is auto generated file **terrafoem.tstate** in this terraform will save some state with some identification, which is called as **KNOWN** state.

Our **main.tf** file will have some state called **DESIRED.**

AWS resource will have one state(manually we might change resource) called **ACTUAL**

Terraform Work: I will go to AWS and takes the **ACTUAL** state and compare with **KNOWN** and **DESIRED,** If any changes Compared to ACTUAL state then it will ask confirmation, Do u want to apply changes to **ACTUAL** state.

Ex:-Suppose if modified tf file desired state is modified file, know state .tstate file and actual state is aws file state.

Let’s enable the versioning for AWS bucket.

resource "aws\_s3\_bucket" "my\_s3\_bucket" { bucket = "my-s3-bucket-devops-practise-002" versioning ={

enabled = true

}

}

There is terraform console, We can runqueries in terraform console, To enter into terraform console run below command

terraform console

Then you can get details through objects and references like below

**Ex:** aws\_s3\_bucket.my\_s3\_bucket

#### Creating AWS IAM User from Terraform:-

As we know that everything in the terraform to create is **resource**

resource "aws\_iam\_user" "my\_iam\_user" { name = "my\_iam\_user"

}

We can store terraform out-put to a file using below command

Terraform plan –out iam.tfplan

You can take gerenrated plan and apply also.

Terraform apply iam.tfplan

Terraform knows that bucket name con’t change(we need to re-create one more bucket), But terraform knows it can change the name of IAM User.

#### Understand tfstate and tfstate.backup:-

tfstate.backup is previous state file and tfstate is present state file

Actually we will not store terraform state file will be not be stored in Git Repository. Then we need to store in s3-bucket, from the s3 bucket we will take directly to terraform, So add these in the .gitignore file

If we have multiple resources and outputs then keeping all the resources in main.tf file is not good. So let’s write superate output to output.tf file

Suppose if we have 3 ts file, if we apply terraform plan (or) apply then terraform concat all the files and exigutes.

**terraform destroy** will delete all the resources that are there in the tf file.

Creating multiple IAM Users at once using below command

resource "aws\_iam\_user" "my\_iam\_user" { count =2

name = "my\_iam\_user\_${count.index}"

}

terraform validate

Terraform validate is the command, which will validate the wheather terraaform code is correct or not.

terraform fmt

Terraform fmt is the command, which will format the terraform code.

You create a varibles as part of terraform and use those inside the code as below, even you can give type of varible

variable "iam\_user\_name\_prefix" { type = string

default = "my\_iam\_user"

}

resource "aws\_iam\_user" "my\_iam\_user" { count =2

name = "${var.iam\_user\_name\_prefix}\_${count.index}"

}

If u don’t mention the default value then terraform ask at the time of validate. These terraform variable helps us in dynamic value taking at the time deployment(QA,DEV,PROD)

Creating terraform-list

variable "names" {

default = ["satish", "ram", "lakshman"]

}

resource "aws\_iam\_user" "my\_iam\_user" { count = length(var.names)

name = var.names[count.index]

}

#### Understand creation of EC2 instance from Terraform:

**What is EC2:-**

EC2 is virtual server in the cloud. A server maintained in the Cloud.

Let’s discuss Private and Public Networks, If u configure network as private then outside people can not access your VPC(MySQL DB we can give), If u configure network as public then outside people can access your VPC(All Our applications server).

Amazon by default it will create one default network for us.



Let’s create a instance manually. And note-down what are all u configured at the time of creation.

* 1. Amazon Machine Image -- ami-0323c3dd2da7fb37d
  2. Instance Type -- t2.micro
  3. Network (VPC) – vpc-a26073d8
  4. Configure Security Group – for accessing the protocols (SSH- for our connectivity,HTTP,HTTPS)

Let’s create a instance from terraform. Let’s configure the Security Group first, And Apply terraform

resource "aws\_security\_group" "http\_server\_sg" { name = "http\_server\_sg"

vpc\_id = "vpc-a26073d8"

ingress { from\_port = 80

to\_port = 80 protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

ingress { from\_port = 22

to\_port = 22 protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

egress { from\_port = 0

to\_port = 0

protocol = -1

cidr\_blocks = ["0.0.0.0/0"]

}

}

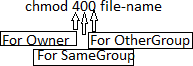
ingress is incoming network to be allowed for our VPC, name  what is the name of scg ,

Egress  It is out going network for VPC 0,0,-1 means all the protocols are allowed Vpcid  To tell for which VPC we are creating the SCG(security Groups)

Till now we create one of step, creating a security-group. Go and check wheather SCG get’s created or not.

For connecting to EC2, we need to have (Key,Pair). By taking one of Key,Pair we will associate this to connect our EC2.

Let’s create Key,Pair in EC2. Change permission of file to read-access only



Now let’s create a EC2 instance.

resource "aws\_instance" "http\_server" {

ami = "ami-0323c3dd2da7fb37d"

key\_name = "default-ec2"

instance\_type = "t2.micro"

vpc\_security\_group\_ids = [aws\_security\_group.http\_server\_sg.id] subnet\_id = "subnet-3129e96e"

}

Subnet\_id  You can get from VPC tab  subnet (By default there are some pre-created subnet, all are public subnets only)

SCG  I am taking with help of terraform query

Let’s deploy http server to EC2 instance and run some message

variable "aws\_key\_pair" {

default = "C:/Users/satis/Desktop/Devops/AWS/default-ec2.pem"

}

resource "aws\_security\_group" "http\_server\_sg" { name = "http\_server\_sg"

vpc\_id = "vpc-a26073d8"

ingress { from\_port = 80

to\_port = 80

protocol = "tcp" cidr\_blocks = ["0.0.0.0/0"]

}

ingress { from\_port = 22

to\_port = 22

protocol = "tcp" cidr\_blocks = ["0.0.0.0/0"]

}

egress { from\_port = 0

to\_port = 0

protocol = -1 cidr\_blocks = ["0.0.0.0/0"]

}

tags = {

name = "http\_server\_sg"

}

}

resource "aws\_instance" "http\_server" {

ami = "ami-0323c3dd2da7fb37d"

key\_name = "default-ec2"

instance\_type = "t2.micro"

vpc\_security\_group\_ids = [aws\_security\_group.http\_server\_sg.id] subnet\_id = "subnet-3129e96e"

connection {

type = "ssh"

host = self.public\_ip

user = "ec2-user"

private\_key = file(var.aws\_key\_pair)

}

#For executing the commands we will use provisioner provisioner "remote-exec" {

inline = [

"sudo yum install httpd -

y", //install httpd (which is http server)

"sudo service httpd start", //need to sta rt the http server

"echo Welcome Satish | sudo tee /var/www/html/index.html" //copy some h tml file for seeing output

]

}

}

As we want to connect to EC2 to perform operations we need to connect it. Let’s do this

with below Connection:-

Type  SSH

Host  default will be created(self refers present location, public \_ip will gets c reated)

User  Default value(Standerd) Private\_key  taking key\_pair from loca

Provisioner :-

To perform operation (or) run some commands we will use Provisiooner. Here we are installing the Httpd server and coping Welcome msg to index.html file (yum is package manager default available in vertical server)

After this perform terraform apply. Take instance public DNS and run in browser.You can take Public DNS using bellow command as well.

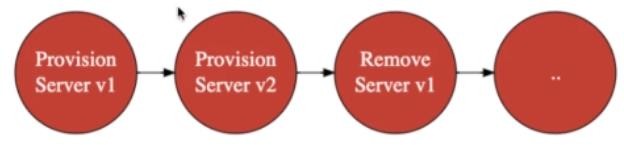
${self.public\_dns} inside the code

In terrafform Console: aws\_instance.your-ref-name.public\_dns

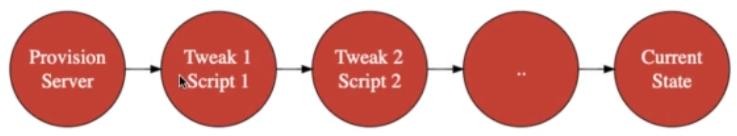
#### Immutable Server:-

Immutable server means creating new server for new-changes, after successful creation delete the old server.

If u apply new-changes directly on existing server then some time your thing might broke.



Immutable Implementing



Non-Immutable Implementing

Now we will remove hard-coding Default-VPC with below code.

#Using this terraform can able to get the details of default VPC, So that we can use in the code.

resource "aws\_default\_vpc" "default" {

}

#Here you can get vpc\_id

vpc\_id = aws\_defaulr\_vpc.default.id

Actually we can use **Data-Provider** to get the values from cloud rather hard-coding the values.

#For this VPC it will get the subnets data "aws\_subnet\_ids" "default\_subnets" {

vpc\_id = aws\_default\_vpc.default.id

}

Subnet\_id = tolist(data.aws\_subnet\_ids.default\_subnets.ids)[0]

data "aws\_ami" "aws-linux-2-latest" { most\_recent = true

owners = ["amazon"] filter {

name = "name"

values = ["amzn2-ami-hvm-\*"]

}

}

#Using

ami = data.aws\_ami.aws-linux-2-latest.id

Suppose you want particular resource only need to apply with AWS then use target as below

terraform apply -target=data.aws\_ami.aws-linux-2-latest(reference names)

You can create a graph for your terraform file, take generated output and get graph from any onine graphviz

terraform graph

Let’s create multiple EC2 instances with below code

variable "aws\_key\_pair" {

default = "C:/Users/satis/Desktop/Devops/AWS/default-ec2.pem"

}

resource "aws\_default\_vpc" "default" {

}

#For this VPC it will get the subnets data "aws\_subnet\_ids" "default\_subnets" {

vpc\_id = aws\_default\_vpc.default.id

}

data "aws\_ami" "aws-linux-2-latest" { most\_recent = true

owners = ["amazon"] filter {

name = "name"

values = ["amzn2-ami-hvm-\*"]

}

}

resource "aws\_security\_group" "http\_server\_sg" { name = "http\_server\_sg"

//vpc\_id = "vpc-a26073d8"

vpc\_id = aws\_default\_vpc.default.id

ingress { from\_port = 80

to\_port = 80

protocol = "tcp" cidr\_blocks = ["0.0.0.0/0"]

}

ingress { from\_port = 22

to\_port = 22

protocol = "tcp" cidr\_blocks = ["0.0.0.0/0"]

}

egress { from\_port = 0

to\_port = 0

protocol = -1 cidr\_blocks = ["0.0.0.0/0"]

}

tags = {

name = "http\_server\_sg"

}

}

resource "aws\_instance" "http\_server" {

#ami = "ami-0323c3dd2da7fb37d" ami = data.aws\_ami.aws-linux-2-latest.id

key\_name = "default-ec2"

instance\_type = "t2.micro"

vpc\_security\_group\_ids = [aws\_security\_group.http\_server\_sg.id] #subnet\_id = "subnet-3129e96e"

#subnet\_id = tolist(data.aws\_subnet\_ids.default\_subnets.ids)[0] for\_each = data.aws\_subnet\_ids.default\_subnets.ids

subnet\_id = each.value

tags = {

name : "http\_server\_${each.value}"

}

connection {

type = "ssh"

host = self.public\_ip

user = "ec2-user"

private\_key = file(var.aws\_key\_pair)

}

#For executing the commands we will use provisioner provisioner "remote-exec" {

inline = [

"sudo yum install httpd -

y", //install httpd (which is http server)

"sudo service httpd start", //need to sta rt the http server

"echo Welcome Satish | sudo tee /var/www/html/index.html" //copy some h tml file for seeing output

]

}

}

Above main change is for loop over all the subnets, creating how many subnets are there

those many we are istances

Let’s create load-balancer resource and we will have superate Security groups for load- balancer.

New load-balancer creating with below code

#For load-balancer

resource "aws\_security\_group" "elb\_sg" { name = "elb\_sg"

//vpc\_id = "vpc-a26073d8"

vpc\_id = aws\_default\_vpc.default.id

ingress { from\_port = 80

to\_port = 80

protocol = "tcp" cidr\_blocks = ["0.0.0.0/0"]

}

ingress { from\_port = 22

to\_port = 22

protocol = "tcp" cidr\_blocks = ["0.0.0.0/0"]

}

egress { from\_port = 0

to\_port = 0

protocol = -1 cidr\_blocks = ["0.0.0.0/0"]

}

}

New resource for Load-balancer

#load balancer resource resource "aws\_elb" "elb" {

name = "elb"

subnets = data.aws\_subnet\_ids.default\_subnets.ids security\_groups = [aws\_security\_group.elb\_sg.id] #Which are the instances under this load-balancer instances = values(aws\_instance.http\_server).\*.id #lister tells which port on load-

balancer should redirect to which port of instance listener {

instance\_port = 80 instance\_protocol = "http" lb\_port = 80

lb\_protocol = "http"

}

}

At last destroy all created resources using terraform destroy

Till now we have the state of terraform in local, but sharing the terraform details in GIT is good. So we will store in **Remote**.

As we are working in team storing tstate file in local in not correct. For this AWS provide us to store in the S3 bucket.

Let’s create superate project for backend-project for storing and taking the state from s3- bucket, remaining any other project will use back-end project for getting the state.

we need to create s3-resource and Locket the state file for update time, after update complete release the lock(For this Locking we will use Dynamo DB)

resource "aws\_s3\_bucket" "enterprise\_backend\_state" {

#This name is very important and here only u need to mention that this state for 1 application(or)multiple application (or) specific environment followed by some unique name

bucket = "dev-applications-backend-state-tstate-store" lifecycle {

#Even if do terraform destroy this bucket will not get's delete's prevent\_destroy = true

}

versioning {

enabled = true

}

#This will store the tstate in encrypted format server\_side\_encryption\_configuration {

rule {

apply\_server\_side\_encryption\_by\_default { sse\_algorithm = "AES256"

}

}

}

}

#Locing Code for MangoDB

resource "aws\_dynamodb\_table" "enterprise\_backend\_lock" { name = "dev\_application\_locks"

billing\_mode = "PAY\_PER\_REQUEST"

hash\_key = "LockID" attribute {

name = "LockID" type = "S"

}

}

Do terraform Apply

To use back-end application for storing state into S3 bucket we need to use below in main.tf

terraform {

backend "s3" {

bucket = "dev-applications-backend-state-in28minutes-abc" #key = "07-backend-state-users-dev"

#Here Key is very important app-nm-env

key = "dev/07-backend-state/users/backend-state" region = "us-east-1"

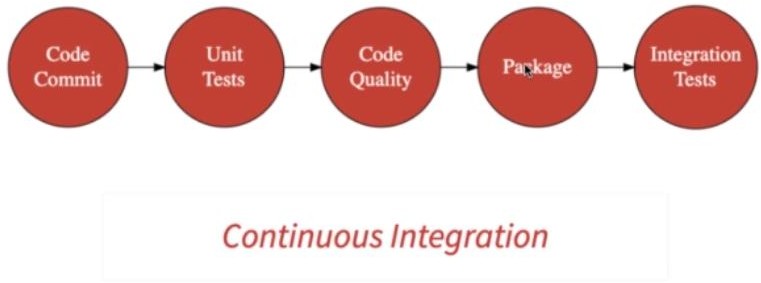
dynamodb\_table = "dev\_application\_locks" encrypt = true

}

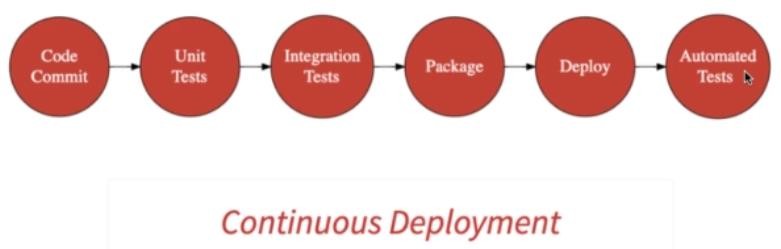
}

**Continuous Integration(CI) & Continuous Development(CD) & Continuous Delivery(CD):-**

Why we need CI ? Ans:- To get quick feedback (If u follow CI you will get immediate feedback)



What is extra over CI with CD(cont. Delivery). It have few more stages deploy and some automated tests.



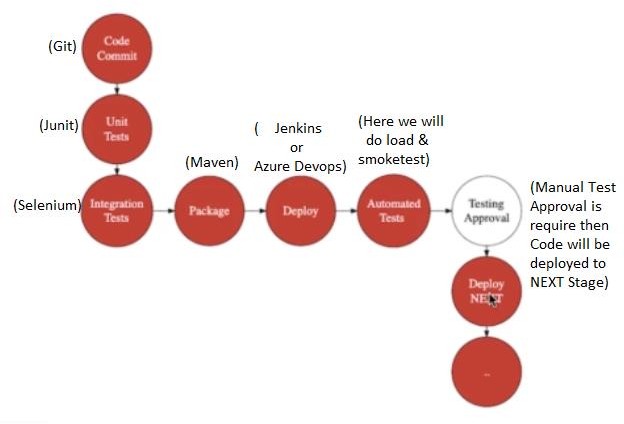
For Conti.. Delivery we will get feedback(Customer-Feedback (or) Testing feedback- from UAT) and will gove to Next stage as shown below.



What are the popular tools used for CI, CD & CD is Jenkins and Azure Devops

How do you implement these using Jenkins and Azure Devops? Ans:-you will create something called **pipelines.**

we will create CI-pipelines, CD-pipelines, CD-pipelines. Below diagram shows the tools used in the all steps



### Azure Devops:-

Azure devops is to develop the tool for achieving CI & CD. Where as Jenkins is tool for CI only.

To deployment you need to have Azure Account. After having Azure account, you u now start working on Azure Devops

Open Azure Devops and create a new project for a respective organization.

Now create a repository in the github and commit your code, Now we will connect this repository with Azure Devops

To do this Go to Pipeline Tab, Click **Create pipeline ,** Now it ask from u want to pick the code(Git, SVN, Azure Repo, etc….)

Connect to Git-Hub >> Take basic pipeline >> change the name of YAML file save and run, Then commit yaml file Git Repo from there itself.

You will find one more YAML file in u r git-repository. With this we created a pipeline

You can see that Job is success. Till now what we have done is we ran one pipeline, similarly you can new pipeline

These run can be Kicking either by click on **New Pipeline (or) Committing Something in the repository.**

Let’s check yaml file which is generated by Azure

trigger:

* master

pool:

vmImage: 'ubuntu-latest'

steps:

* script: echo Hello, world! displayName: 'Run a one-line script'
* script: |

echo Add other tasks to build, test, and deploy your project. echo See https://aka.ms/yaml

displayName: 'Run a multi-line script'

#### Trigger :-

Trigger tells when do you want to run the Jobs(On every change of which branch)

#### Pool:-

Pools that on which system(Windows,Ubentu etc…) this builds should happen. By default builds will happen in the Ubentu. You can change this in the project setting if u want.

#### Steps:-

1. These steps tells how u want to run the your pipes(What things u want to run as part of your pipeline)
2. Here u can run tasks,jobs here. Tasks is a most basic element in azure element in azure, each one is one step(or)task.

Task-Started

* script: echo Hello, world! displayName: 'Run a one-line script'

Task-ended

Task-started

* script: |

echo Add other tasks to build, test, and deploy your project. echo See https://aka.ms/yaml

displayName: 'Run a multi-line script'

Task-ended

Above pipe indicate run multi-line scripts.

In Azure pipelines, In pipelines we have multiple Stages(Dev,Prod), In stages we have multiple Jobs (build, run test-case,deploy), In jobs we have multiple steps(Tasks)

Pipelines >> Stages >> Jobs >> Tasks(Steps)

Each of these jobs will run on different machine, Unless until there was dependency theses will ran independently.

jobs:

* job: job1 steps:
  + script: echo Hello, world Job1!
  + script: |

echo Add other tasks to build, test, and deploy your project. echo See https://aka.ms/yaml

* job: job2

steps:

- script: echo Hello, world Job1! displayName: 'Run a one-line script'

If u want dependency of one Job with another job use **dependsOn** as shown below.

jobs:

* job: job1 steps:
  + script: echo Hello, world Job1!
  + script: |

echo Add other tasks to build, test, and deploy your project. echo See https://aka.ms/yaml

* job: job2 dependsOn: job1 steps:
  + script: echo Hello, world Job1!

displayName: 'Run a one-line script'

Now let’s see how to create stages, If u observe the stages will happen one after the another only. If u want u can give dependency of one stage to another using Depends on as shown delow.

stages:

* stage: Build jobs:
  + job: FirstJob steps:
    - bash: echo build FirstJob
  + job: SecondJob steps:
    - bash: echo build SecondJob
* stage: DevDeploy dependsOn: Build jobs:
  + job: DevDeployFirstJob steps:
    - bash: echo build FirstJob
  + job: DevDeploySecondJob steps:
    - bash: echo build SecondJob
* stage: QADeploy dependsOn: DevDeploy

jobs:

* job: QADeployFirstJob steps:
  + bash: echo build FirstJob
* job: QADeploySecondJob steps:
  + bash: echo build SecondJob
* stage: ProdDeploy jobs:
  + job: PrdDeployFirstJob steps:
    - bash: echo build FirstJob
  + job: ProdDeploySecondJob steps:
    - bash: echo build SecondJob

Whenever if don’t want pipeline to be run, then can **disable** the pipeline in Azure Devops.

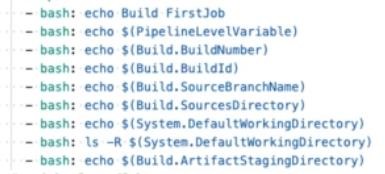
#### Varible Configure in Pipeline:-

Click variable at pipeline level >> New Varible >> Enter(Name,Value) >> Create variable

After creating the variable and use then in the following manner as part of your code.

$(PipelineLevelVarible)

There are some pre-define varibles are there as part of Azure Devops. Let’s see those varibles.



Now let’s try copy files from Source directory(**It is the Directory where our Source Code is available**) to Artifact Staging Directory(**This directory where our jar’s and all o/p relates stuff are available**)

Use can use existing tasks from UI and and pre-define varibles for getting Directory details.

- task: CopyFiles@2 inputs:

SourceFolder: '$(System.DefaultWorkingDirectory)' Contents: |

\*\*/\*.yaml

\*\*/\*.tf

TargetFolder: '$(Build.ArtifactStagingDirectory)'

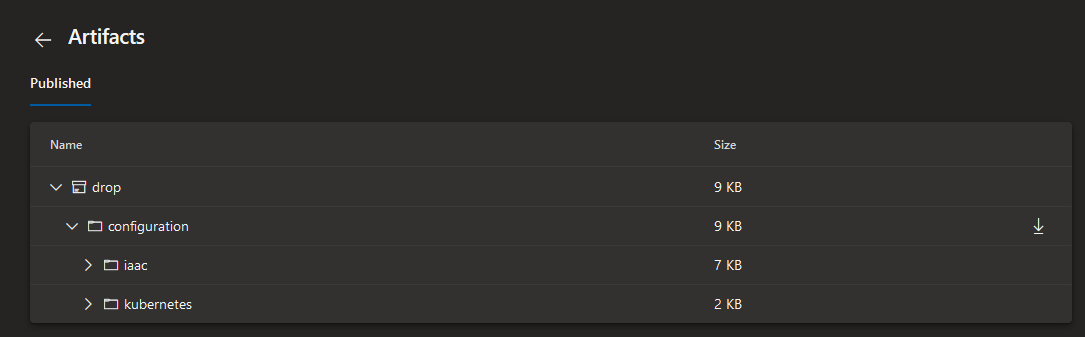
Suppose If u want one stage files to be used in another stage the way u can do by Publish build artifacts. Publish build artifacts are also available in redefine tasks.

- task: PublishBuildArtifacts@1 inputs:

PathtoPublish: '$(Build.ArtifactStagingDirectory)' ArtifactName: 'drop'

publishLocation: 'Container'

You will see one artifact is produced as part of output. If you click artifacts you will see below like



#### Running one operation in multiple agents:-

Suppose if I want to run same build operation in multiple environment like both in ubentu and mac os then I need to

for achieving this we need to Strategy

strategy: matrix:

linux:

operatingSystem: 'ubuntu-latest' mac:

operatingSystem: 'macos-latest'

pool:

vmImage: $(operatingSystem)

There is special type of jobs called deployment. Compare to normal Job, deployment job will have some more additional features let’see those

If u go-to environment tab u will find there are 2 environments. In each Environment you add Approvals and Checklist.

This means for this job will get wait until that approval person get’s approves. After approval deployment will be done seccussfully.

Here interesting thing is that whenever a code has been committed to a Git Repository, then Docker image shold get’s created and push to a docker hub.

Before this we need to have a connection for **Docker-Hub** from Azure Devops. For this **Go to Project Settings** >> **Select Service Connections** >> **New Service Connection** >> **Docker Registry** >> **Docker Hub** >> **Enter your docker details >> Save details**

Now at the time of creating pipeline as part of as part of **Configuration** Select **Docker**

rather Basic pipeline

In the yaml file click on settings to give details for docker container details(Container Registry and Container Repository details and command BuildAndPush) then click on Add.

Finally we have the YAML is below.

trigger:

* master

resources:

* repo: self

variables:

tag: '$(Build.BuildId)'

stages:

* stage: Build displayName: Build image jobs:

- job: Build displayName: Build pool:

vmImage: 'ubuntu-latest' steps:

- task: Docker@2 displayName: Build an image

inputs:

containerRegistry: 'devops-practise-docker-hub' repository: 'satish4228/currency-exchange-devops' command: 'buildAndPush'

Dockerfile: '\*\*/Dockerfile'

tags: '$(tag)'

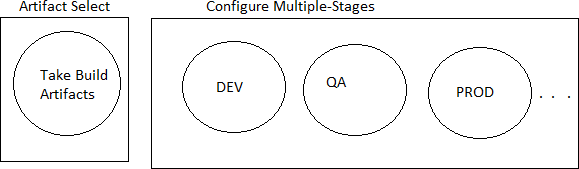
If u observe we have 9 steps only as part of Dockerfile, but some more steps will be Azure by default itself. Finally image will be pushed to Docker hub.

Now we will play Releases in the Azure Depos.(Remember)These Releases will make use of Build-Artifacts. Go to Replease tab click New PipeLine >> Select Empty Job >> Select ArtifactsNow Choose the Existing Artifacts >> Click Add >> Click Save for Artifacts

For Stag also add the task and save and Click on Create Release.

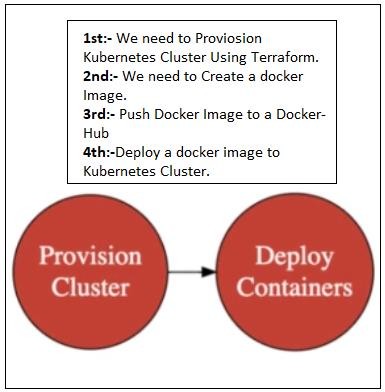
There you can sechudele the releases with custom dates and time. You can release for every build of selected BuildArtifacts.

It is like below



**Azure, AWS EKS, CI, CD and IAC with Docker and Terraform**

Before going to code wise overview let’s do a quick overview, what we have to do to achieve CI & CD



Creating a Kubernetes Cluster(1st step)- **Which is called IAAC(Infrastructure as Code)**

For creating Cluster we will use Terraform Code as below.

terraform { backend "s3" {

bucket = "mybucket" # Will be overridden from build

key = "path/to/my/key" # Will be overridden from build region = "us-east-1"

}

}

resource "aws\_default\_vpc" "default" {

}

data "aws\_subnet\_ids" "subnets" { vpc\_id = aws\_default\_vpc.default.id

}

provider "kubernetes" {

host = data.aws\_eks\_cluster.cluster.endpoint cluster\_ca\_certificate = base64decode(data.aws\_eks\_cluster.cluster.certifica

te\_authority.0.data)

token = data.aws\_eks\_cluster\_auth.cluster.token

load\_config\_file = false

version = "~> 1.9"

}

module "in28minutes-cluster" {

source = "terraform-aws-modules/eks/aws" cluster\_name = "in28minutes-cluster" cluster\_version = "1.14"

subnets = ["subnet-3f7b2563", "subnet-4a7d6a45"] #CHANGE #subnets = data.aws\_subnet\_ids.subnets.ids

vpc\_id = aws\_default\_vpc.default.id

#vpc\_id = "vpc-1234556abcdef"

node\_groups = [

{

instance\_type = "t2.micro" max\_capacity = 5

desired\_capacity = 3

min\_capacity = 3

}

]

}

data "aws\_eks\_cluster" "cluster" {

name = module.in28minutes-cluster.cluster\_id

}

data "aws\_eks\_cluster\_auth" "cluster" {

name = module.in28minutes-cluster.cluster\_id

}

# We will use ServiceAccount to connect to K8S Cluster in CI/CD mode # ServiceAccount needs permissions to create deployments

# and services in default namespace

resource "kubernetes\_cluster\_role\_binding" "example" { metadata {

name = "fabric8-rbac"

}

role\_ref {

api\_group = "rbac.authorization.k8s.io" kind = "ClusterRole"

name = "cluster-admin"

}

subject {

kind = "ServiceAccount"

name = "default" namespace = "default"

}

}

# Needed to set the default region provider "aws" {

region = "us-east-1"

}

To create AWS Kubernetes Cluster we are using Terraform-aws-module(To simplify our code we are using this module).

Our Azure Devops wants to talk to AWS for Accessing back-end-tstatefile and other, So take one of the IAM user details and connect from Azure Devops.

Create new Service Connection for AWS connection. If u are not aws then we need to add AWS plugin. Search in internet **AWS Toolkit for Microsoft Azure DevOps is an extension for Microsoft Azure DevOps (formerly VSTS)** and add extenstion in the market place.

Similarly you need to do for **Aws for Terraform** also.

We will create one pipeline for IAAC and another for deploy.

Create new pipeline for IAAC (Starter pipeline). Write a terraform init first and then terraform apply for creating the IAAC. I might take failure also after 25, 20 minutes try one more time it will success.

Next to connect to AWS kubernetes cluster we need to add Service Connection (Name : Kubernetes).

Here we need to choose service account and we need to enter details like ID and secret key

After service connection we need to Build docker-image and deploy to a kubernetes cluster.

For Second pipeline we will take 2 stages 1)Build Docker Image 2)Deploy to k8s

#### Stage1:

1)Build Docker Image 2)push to docker hub

3)copy files to staging directory 4)publish

#### Stage2:-

1)Download from Stagging Dir 2)Deploy image to k8s 3)kubectl apply

Finally Delete all clusters which you create.

#### Azure Devops Demo Generator:-

This is online tool for creating Projects. Let’s create one project. After creating the project you will see there are lot of pre-configured things are there.

**Jenkins**

Jenkins is most popular Continuous Integration(CI) Tool.

Let’s grab the Jenkins Image from the Docker-Hub.For Jenkins we have docker- compose.yaml as below.

version: '3.2' services:

jenkins:

image: jenkins/jenkins:lts privileged: true

user: root ports:

- 8081:8080

- 50000:50000

container\_name: jenkins volumes:

-

/tmp/jenkins:/var/jenkins\_home #Remember that, the tmp directory is designed to be wiped on system reboot.

* /var/run/docker.sock:/var/run/docker.sock
* /usr/local/bin/docker:/usr/local/bin/docker

Run docker-compose up.

When u ran above command you will get password. Pleases write-down the password. And lunch Jenkins by hitting http://localhost:8081 (If you using ToolBox use toolbaxIP:8081) There admin password will be asked.

Install the suggested Plugins, Meanwhile commit your code to Git Hub repository.

After installing required plugins, it will ask you create First Admin User. Make Note down the Details

Now we want to install Maven and Docker in Jenkins. This we are doing because we can use these across pipelines. So for this Go to **Manage Jenkins >> Global Tool Configuration**

#### >> Add Maven & Docker >> Save

Now Let’s create a new job, Give name >> Select pileline >> Ok

Next screen if u have lot of features if u want u can use. Select Poll SCM (means based version Control run the build). Give the schedule \* \* \* \* \*(mean build every minute.)

Next Pipeline, you can write simple pipeline script, but now we will script from Git-Hub Repo. Use drop-down (Pipeline-SCM) >> Git >> Paste the Repo URL. Carefull path for Jenkins file. Finally Click Save and Build Now.

Now your able to see Build and Test 2 stages which are going, Let’s see what are those.

We have a **no-extenxtion file** called **Jenkinsfile**

node {

stage('Build') { echo "Build"

}

stage('Test') { echo "Test"

}

}

The above is style of **scripted pipeline**. This is the **old way** of creating pipeline.

Here **node** refers Machine which executes pipeline. In Scripted pipeline writing stages are not mandatory you can write like below also.

node {

echo "Build" echo "Test"

}

There is **Declarative Pipeline** syntaxof creating pipeline, here there is node at all. In declarative pipeline will not be there **without stages**, so stages are mandatory and In **stages** having **steps** is also mandatory

Declarative pipeline will checkout automatically, whereas scripted pipeline manually we need to do checkout.

//Declarative Syntax pipeline {

agent any stages {

stage('Build') { steps {

echo "Build"

}

}

stage('Test') { steps {

echo "Test"

}

}

}

}

In declarative we can write what should if build is success and failure cases as below as shown below.

stages {

stage('Build') { steps {

echo "Build"

}

}

stage('Test') { steps {

echo "Test"

}

}

}

post {

always {

echo "Always I will run"

}

success {

echo "When Success"

}

failure {

echo "When failure"

}

Make sure space is require before post lone

We can use Docker image also as agent, Let’s try

agent { docker { image 'maven:3.6.3'}

Above tells that docker ehich installed maven will be act as a agent. To use shell command

use below

Sh ‘mvn --version’

All our stages relates o/p will be run by docker container which is having MVN. Similarly if you want any software installed u can use like docker image.

#### Pipeline Syntax:-

It is like a help for generating some code for us and To know more syntax for writing the pipeline code.

We have snippet generator, there are some default snippets that are availiable if u want u take these help.

There are some Globel Environment varibles as shown below

echo "$PATH"

echo "BUILD\_ID - $env.BUILD\_ID" echo "JOB\_NAME - $env.JOB\_NAME" echo "BUILD\_TAG - $env.BUILD\_TAG"

echo "BUILD\_URL - $env.BUILD\_URL"

Now Let’s build jar and docker-image

Initially we have created a globel configuration for maven and docker, So let’s use these in building.

After agent we need to have environment to access those globel configuration like below

agent any environment {

// We want to take Home Path for boot the tools

// Add to Our Environment path dockerHome = tool 'myDocker' mavenHome = tool 'myMaven'

// This will add both maven and docker bin folder path to the our env

path

PATH = "$dockerHome/bin:$mavenHome/bin:$PATH"

}

What is goal achieved is We have Docker and Maven in Globel Place, We used those using environment and we ran just version check, Previously We have taken maven installed image and we check the version

Now we will those docker and maven only to build and deply. The below code is for mvn build and then Tests and Integration tests

stage('Compile') {

steps {

sh 'mvn clean install'

}

}

stage('Test') { steps {

sh 'mvn test'

}

}

stage('Integration test') { steps {

sh 'mvn failsafe:integration-test failsafe:verify'

}

}

}

Now we need to create docker-image and push to Docker-Hub, but to push image to docker hub we need to Give credentials.

HomePage >> Credentials >> Click Jenkins >> Globel Credentials >> Add credentials >> Enter details >> ok

Now we want use these credential in pipeline as following manner.

Finnaly code is below.

stage('Build Docker Image') {

steps {

// This is the Declarative way of calling

// sh 'docker build -t satish4228/currency-exchange-service- from-jenkins:$env.BUILD\_TAG'

//The above can be achived by using the function aswell. script{

dockerImage = docker.build("satish4228/currency-exchange- service-from-jenkins:${env.BUILD\_TAG}")

}

}

}

//But Jenkins want to connect your docker hub stage ('package') {

steps{

script{

sh 'mvn package -DskipTests'

}

}

}

stage('Push Docker Image'){

steps {

script{

docker.withRegistry ('', 'dockerhub') {

dockerImage.push();

//For a latesh push use dockerImage.push('latest')

}

}

}

}

## Ansible

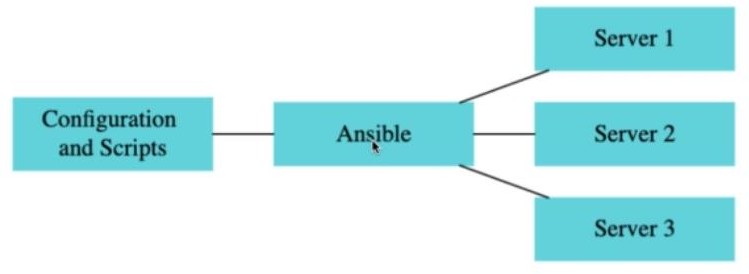
Ansible is IAAC Tool.

Terrafor is for creating the servers, Ansible is configuring the Softwares on the servers.

Ansible can also do the servers provisioning.

Windows is not supported for Ansible. If you have Azure account there you can use Ansible

Let’s create 3 EC2 instances using Terraform.



To start with ansible we need to have one file which is **ansible.cfg**. All configuration related to ansible we will keep in this.

Second file we will create is **ansible\_hosts.** All the server information we will keep in this.

Create a folder called **playbooks.** If u want to do any change in ansible you need work with play book in ansible. If u want to play anything with connected servers then u need to write playbooks.

**Inventort** Where are the all the server configuration presents.

Who is the remote user (remote\_user=ec2-user)

private key location path and host\_key\_check = false tells that no checking for do u want to believe.

If ansible fails then it will create retry\_files. False means don’t create those files.

[defaults] inventory=./ansible\_hosts remote\_user=ec2-user

private\_key\_file=C:/Users/satis/Desktop/Devops/AWS/default-ec2.pem

host\_key\_checking=False retry\_files\_enabled=False

Next we will configure the servers. Copy all the public IP of the aws servers.

[dev] 34.207.89.27

54.80.222.239

54.162.123.183

The command to ping all the servers is shown below.

ansible –m ping all

we can give name for all the hosts as shown below with Groupaswell.

[dev]

dev1 ansible\_host=3.83.104.44 dev2 ansible\_host=54.81.35.168

[qa]

qa1 ansible\_host=3.95.249.131

[first] dev1 qa1

[groupofgroups:children] dev

first

[devsubset] dev[1:2]

We can check server on each version contains using below command. Which will list all matching host.

ansible qa(group-nm) –a “python --version” or

ansible --list-host (group-nm)

These groups are helpful if we have multiple servers like for DB one server, app – one servers

Now we write playbooks, So create file **01-ping-server.yaml** . In playbook we can specify number of tasks and we need to tell where that specific task gets execute.

Three --- are actually indicating these is yaml file

---

- hosts: all tasks:

- name: Ping All Servers

action: ping

To execute play book we need to below command

Ansible-playbook yaml-file-nm

Now we will execute some shell commands using playbooks--

* hosts: qa tasks:
  + name: Execute shell commands shell: uname

register: uname\_result

* + debug: msg="{{ uname\_result }}"

We want get the o/p of this command using **register**

Now we will play with ansible varibles, we can declare the varivles as vars

---

* hosts: dev vars:

varname: "Play Value" tasks:

* + name: Execute shell commands
  + debug: msg="{{ varname }}"

You can pass these variable values through command-line aswell using below command.

ansible-playbook yaml-file-nm.yaml –e varible1=varibleValue

You can take variable from a different file using below code

---

- hosts: dev vars\_files:

- var-file-nm.yaml vars:

varname: "Play Value"

tasks:

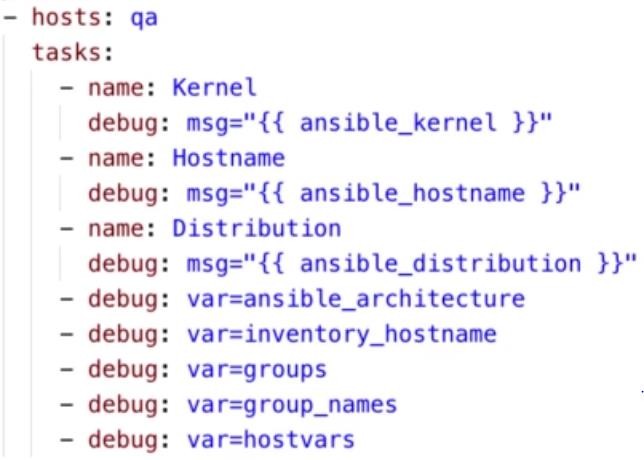
* name: Execute shell commands
* debug: msg="{{ varname }}"

#### Ansible Facts:-

When we run ansible-palybook we will see something called Gathering Facts, Now we will see what are these

Ansible facts does gathering of lot of servers information, ehich we can use in our playbooks.

The following are the some of facts gathred information



Now we will try to install the Apache server and in our instances. To install server in instance we would want to be root(become: true)

---

- hosts: dev become: true tasks:

- yum:

name:

- httpd state: present

* service: name=httpd state=started enabled=yes
* raw: "echo Welcome.... | sudo tee /var/www/html/index.html"

With that one simple play-book file we can control 100’s of server at once.

Let’s see how can we re-use existing play books and run multiple play-books at once.

* import\_playbook: file1.yaml
* import\_playbook: file2.yaml
* import\_playbook: file3.yaml
* import\_playbook: file4.yaml

Now u can execute this yaml file which will go execute all imported yaml files.

Now we will understand the Conditional and looping.

---

* hosts: qa vars:

system: "Windows" color: "Red"

tasks:

# - debug: var=hostvars

* + debug: var=ansible\_system
  + debug: var=color

when: system == 'Linux'

* + debug: var=item with\_items:
    - item1
    - item2
    - item3
    - item4
  + debug: var=item.name with\_items:
    - name: Ranga country: India
    - name: Jane country: US
    - name: Doe

country: Netherlands

As part of ansible\_hosts we are hot coding the IP address, we get those by using **Ansible Dynamic Inventory.**

For this we need to install boto3 and botocore. When it comes to real world our instances will get increase and decrese

create inventory folder and create a file which is must end with **aws\_ec2** and make the change of inventory location to present created file in **ansible\_config.**

plugin: aws\_ec2 region:

* us-east-1 keyed\_groups:
* prefix: arch

key: 'architecture'

* prefix: tag key: 'tags'
* prefix: aws\_region key: placement.region
* key: tags.Environment separator: ''
* key: instance\_type

prefix: instance\_type

Creating EC2 instances using Ansible

---

* hosts: localhost tasks:
  + ec2:

key\_name: default-ec2 instance\_type: t2.micro image: ami-062f7200baf2fa504 region: us-east-1

#count: 1

exact\_count: 2 count\_tag: {type: http}

vpc\_subnet\_id: subnet-75cb9912 assign\_public\_ip: yes

group: ["http\_server\_sg"]

instance\_tags: {type: http, Environment: QA} wait: yes

register: ec2\_output

* + debug: var=ec2\_output