Prblm:

It works on my machine.

Dockers are used to build containers which contains everything like (code,dependencies,configuration,process,os(chunks),networking etc)

We can bulid our application in the guest os with all necessary dependencies and all…



But every new application we need to install new Guest os and build our app in them so it takes burden to vms.



So we have to replacing the hypervisor…we can just install docker on top of os and build aour applications…



Docker\_basic\_cmds:

**1.Docker run hello-world**

It will run the hello-world image from docker hub…and prints hello world.

**2.docker images**

Shows images

**3.docker pull ubuntu**

It will pull docker image from hub

**4.docker run -it -d ubuntu**

It will run the ubuntu image as a container

**5.docker ps -a**

It will shows running containers

**6.docker exec -it container\_id bash**

It will access a running container…we will go within the container

**7.docker stop cont\_id**

For stopping docker container

**8.docker commit container\_id account\_name/ image\_name**

It will commit the new image to the local repository

**9.docker login**

For logging in to docker hub

TAG,book\_inv, .,

**10.docker push dockerhub\_username/image\_name**

Pushes image to local docker repository

**11.docker rm container\_id**

For deleting container

**12.docker rmi image\_id**

Deleting docker images

**Docker file is blue print for building images, image is a template for running container and a container is actual process.**

Building image with tag:

**docker build --tag flask-demo:1.0 .**

Every docker build sends a directory to the build server. Docker is a client/server application, and the build runs on the server which isn't necessarily where the docker command is run. Docker uses the build context as the source for files used in COPY and ADD steps. When you are in the current directory to run the build, you would pass a . for the context, aka the current directory. You could pass a completely different directory, even a git repo, and docker will perform the build using that as the context, e.g.:

docker build -t sudobmitch/base:alpine --target alpine-base \

'https://github.com/sudo-bmitch/docker-base.git#main'

**Docker -d**

**Detached mode if we use detached mode container will be run in background…and we can interact with terminal….if we don’t use -d then the log info will be shown in terminal and we can’t interact with terminal, if we terminate the process the container will not be in run state.**

**docker run -d -p 8080:5000 -t flask-demo:v1.0**

**🡪This will run container on 8080 port of localhost with random container name**

**docker run -p 5000:5000 -d --name v2\_container flask-demo:1.0**

**🡪This cmd create a container with name v2\_container of image flask:demo:1.0(tagname:1.0) on port 8080.**

**docker run --rm -it flask-demo:1.0**

**This command will run a container without detached mode and after exiting from it the container will be removed (--rm** there to tell the Docker Daemon to clean up the container and remove the file system after the container exits**) and (–it)** flag tells Docker to allocate a virtual terminal session within the container (i) is for interactive option which keeps STDIN open even if running in detached mode

Dot(.):

There is a “.” (dot) at the end of the command (which is **an alias for $PWD current directory**). This tells docker the that the resources for building the image are in the current directory. By default, when you run the docker build command, it looks for a file named Dockerfile in the current directory.

**docker tag flask-demo:1.0 venkatrobin/flask-demo:v1**

🡪To push the docker image first we need to tag the image and create new image with the tag…venkatrobin is my dockerhub id, flask-demo:v1 is my image name v1 is tag for new image.

**Docker build . -t my-app -f subfolder/Dockerfile**

**🡪Above command can build a image by specifying Dockerfile in another directory.**

**docker run -p 8080:8080 -p 50000:50000 -d -v jenkins\_home:/var/jenkins\_home jenkins/jenkins:lts**

🡪We are pulling Jenkins image and running….50000(master and slave ports) -v for creating volume for storing all Jenkins data…in var/Jenkins\_home path.

**Jenkins Pipeline**

Jenkins Pipeline is a suite of plugins which supports implementing and integrating continuous delivery pipelines into Jenkins

A Jenkinsfile can be written using two types of syntax

🡪 Declarative and Scripted

* Declarative provides richer syntactical features over Scripted Pipeline syntax, and is designed to make writing and reading Pipeline code easier

## Why Pipeline?

* Jenkins is, fundamentally, an automation engine which supports a number of automation patterns.
* Pipeline adds a powerful set of automation tools onto Jenkins.

 features of Pipeline:

**Code**: Pipelines are implemented in code and typically checked into source control.

**Durable**: Pipelines can survive both planned and unplanned restarts (\*\*)

**Pausable**: Pipelines can optionally stop and wait for human input or approval before continuing the Pipeline run.

**Versatile**: Pipelines support complex real-world CD requirements(fork,loop,join,parallelizam)

**Extensible**: The Pipeline plugin supports custom extensions to its DSL(**Domain Specific Language)**

Pipeline syntax:

### **Pipeline**

🡪 includes stages for building an application

🡪 Also, a pipeline block is a [key part of Declarative Pipeline syntax](https://www.jenkins.io/doc/book/pipeline/#declarative-pipeline-fundamentals).

### **Node:**

🡪Also, a node block is a [key part of Scripted Pipeline syntax](https://www.jenkins.io/doc/book/pipeline/#scripted-pipeline-fundamentals).

### **Stage:**

🡪present Jenkins Pipeline status/progress

### **Step:**

🡪 "step" in the process.( shell command make use the sh step: sh 'make')

### **Declarative Pipeline Syntax:**

pipeline {

agent any

stages {

stage('Build') {

steps {

//

}

}

stage('Test') {

steps {

//

}

}

stage('Deploy') {

steps {

//

}

}

}

}

|  |
| --- |
| 1. Execute this Pipeline or any of its stages, on any available agent(executer or a node). |
| 1. Define build stage 2. Perform steps related to build stage 3. Define test stage 4. 3(step)with test 5. Define Deploy stage 6. 3(step)with deploy  **Scripted Pipeline Syntax:** one or more node blocks do the core work throughout the entire Pipeline.(not mandatory) |
| inside of a node block does two things:  Schedules the steps contained within the block to run by adding an item to the Jenkins queue. As soon as an executor is free on a node, the steps will run.  Creates a workspace (a directory specific to that particular Pipeline) where work can be done on files checked out from source control.  Syntax:  node {  stage('Build') {  //  }  stage('Test') {  //  }  stage('Deploy') {  //  }  }  Execute this Pipeline or any of its stages, on any available agent.  Defines the "Build" stage. stage blocks are optional in Scripted Pipeline syntax. However, implementing stage  provides clearer visualization of each `stage’s subset of tasks/steps in the Jenkins UI.  Perform some steps related to the "Build" stage  Defines the "Test" stage.  Step(3)with test  Define deploy stage  Step(3)with stage  example of a Jenkinsfile using Declarative Pipeline syntax:  pipeline {  agent any  options {  skipStagesAfterUnstable()  }  stages {  stage('Build') {  steps {  sh 'make'  }  }  stage('Test'){  steps {  sh 'make check'  junit 'reports/\*\*/\*.xml'  }  }  stage('Deploy') {  steps {  sh 'make publish'  }  }  }  }  [**http://localhost:8080/env-vars.html/**](http://localhost:8080/env-vars.html/)  **🡪for getting all env variables of Jenkins**  **echo "nameserver 8.8.8.8" | sudo tee /etc/resolv.conf > /dev/null**  **For resetting dns prblm…**  **venkatlinux\_v20@IN-PF2N80T7:~$ cat /lib/systemd/system/docker.service**  **[Unit]**  **Description=Docker Application Container Engine**  **Documentation=https://docs.docker.com**  **After=network-online.target docker.socket firewalld.service containerd.service**  **Wants=network-online.target**  **Requires=docker.socket containerd.service**  **[Service]**  **Type=notify**  **# the default is not to use systemd for cgroups because the delegate issues still**  **# exists and systemd currently does not support the cgroup feature set required**  **# for containers run by docker**  **ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock**  **ExecReload=/bin/kill -s HUP $MAINPID**  **TimeoutSec=0**  **RestartSec=2**  **Restart=always**  **# Note that StartLimit\* options were moved from "Service" to "Unit" in systemd 229.**  **# Both the old, and new location are accepted by systemd 229 and up, so using the old location**  **# to make them work for either version of systemd.**  **StartLimitBurst=3**  **# Note that StartLimitInterval was renamed to StartLimitIntervalSec in systemd 230.**  **# Both the old, and new name are accepted by systemd 230 and up, so using the old name to make**  **# this option work for either version of systemd.**  **StartLimitInterval=60s**  **# Having non-zero Limit\*s causes performance problems due to accounting overhead**  **# in the kernel. We recommend using cgroups to do container-local accounting.**  **LimitNOFILE=infinity**  **LimitNPROC=infinity**  **LimitCORE=infinity**  **# Comment TasksMax if your systemd version does not support it.**  **# Only systemd 226 and above support this option.**  **TasksMax=infinity**  **# set delegate yes so that systemd does not reset the cgroups of docker containers**  **Delegate=yes**  **# kill only the docker process, not all processes in the cgroup**  **KillMode=process**  **OOMScoreAdjust=-500**  **[Install]**  **WantedBy=multi-user.target**  Error response from daemon: Get "https://registry-1.docker.io/v2/": dial tcp: lookup registry-1.docker.io on 172.22.224.1:53: read udp 172.22.236.210:40748->172.22.224.1:53: i/o timeout |
|  |
|  |

docker: Error response from daemon: create usr/bin/docker: "usr/bin/docker" includes invalid characters for a local volume name, only "[a-zA-Z0-9][a-zA-Z0-9\_.-]" are allowed. If you intended to pass a host directory, use absolute path.

See 'docker run --help'.

docker run -d --name jenkins -u root -p 8081:8080 -p 50001:50000 -v jenkins\_home:/var/jenkins\_home -v /var/run/docker.sock:/var/run/docker.sock --name jenkins jenkins/jenkins:latest

Possible way:

venkatlinux\_v20@IN-PF2N80T7:~$ docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

flask-demo 1.0 c6e0f88ae919 6 days ago 906MB

flask-demo 2.0 c6e0f88ae919 6 days ago 906MB

venkatrobin/flask-demo v1 c6e0f88ae919 6 days ago 906MB

jenkins/jenkins latest 57e97fdd24d7 6 days ago 460MB

portainer/portainer latest 12b0b8dced14 2 weeks ago 75.4MB

jenkins/jenkins lts 73264ace1394 2 weeks ago 464MB

ubuntu latest d2e4e1f51132 3 weeks ago 77.8MB

hello-world latest feb5d9fea6a5 8 months ago 13.3kB

python 3.8.10 a369814a9797 10 months ago 883MB

venkatlinux\_v20@IN-PF2N80T7:~$ docker rmi 57e97fdd24d7

Untagged: jenkins/jenkins:latest

Untagged: jenkins/jenkins@sha256:dea00045c93b16adc444c3211398ffb5dd34dacac5a52642c644fce36649d05b

Deleted: sha256:57e97fdd24d781dff3f100c5b18c801ab139edceac0bc19b5ca2bcc23ef9cb19

Deleted: sha256:7f99476932ffd2503ba94fc4d549a5a8d527746754c079f2c7c46c0571742926

venkatlinux\_v20@IN-PF2N80T7:~$ docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

flask-demo 1.0 c6e0f88ae919 6 days ago 906MB

flask-demo 2.0 c6e0f88ae919 6 days ago 906MB

venkatrobin/flask-demo v1 c6e0f88ae919 6 days ago 906MB

portainer/portainer latest 12b0b8dced14 2 weeks ago 75.4MB

jenkins/jenkins lts 73264ace1394 2 weeks ago 464MB

ubuntu latest d2e4e1f51132 3 weeks ago 77.8MB

hello-world latest feb5d9fea6a5 8 months ago 13.3kB

python 3.8.10 a369814a9797 10 months ago 883MB

venkatlinux\_v20@IN-PF2N80T7:~$ docker run -d --name jenkins -u root -p 8081:8080 -p 50001:50000 \

> -v jenkins\_home:/var/jenkins\_home \

> -v /var/run/docker.sock:/var/run/docker.sock \

me jenki> --name jenkins jenkins/jenkins:lts

a9df5f57cb852bf30f16989ae2c223de0d996f414d7fcd8082472767e14c332c

1.branch change

2.build

3.push

4.run

**Imp links:**

**->https://gist.github.com/afloesch/ea855b30cfb9f157dda8c207d40f05c0**

🡪<https://www.linkedin.com/video/event/urn:li:ugcPost:6925367528030396417/>

terraform training…

🡪<https://www.lambdatest.com/blog/jenkins-declarative-pipeline-examples/?utm_source=linkedin&utm_medium=group&utm_campaign=mar27_sb&utm_term=sb&utm_content=blog>

jenkins blog

🡪<https://www.youtube.com/watch?v=5a21SKsAvPw>

(jenkins-master-slave-video)

https://stackoverflow.com/questions/61044914/how-to-expose-minikube-in-gcp-vm

Nginx in kubernetes

**Kubernetes:**

Management tool for containers either Docker containers or linux container.(Scalability)

Can scale containers without having to do it manually.

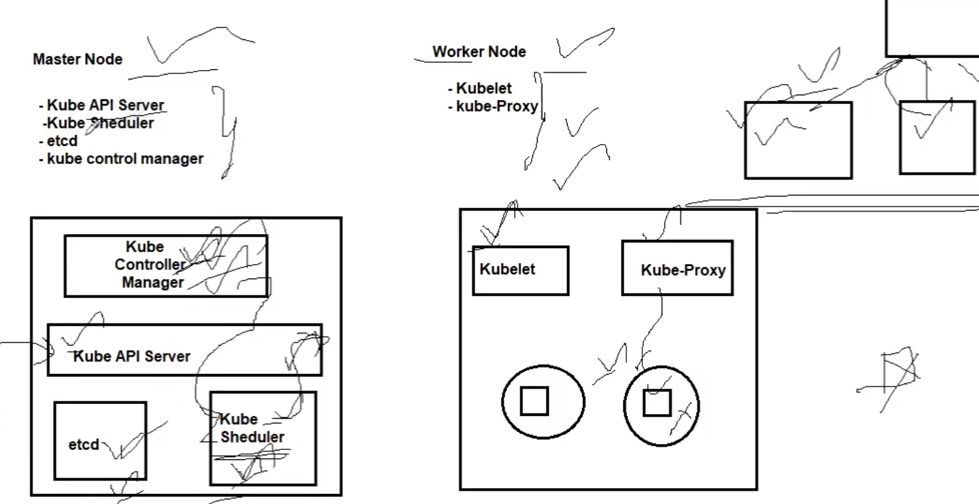
Use Case(PokemonGo,special sale days in e-commerce sotes, etc)

Structure:

🡪Pod is the smallest component in Kubernetes framework(it container one or more containers)

🡪Worker node contains pods, it’s like a vm, contains necessary components to run pods and it takes instr. From master node ,master node gets instr. From user yaml files.

🡪All these will be in Kubernetes cluster.



**Master node:**

Kube api server:- instr. From user through yaml files or json etc..

Kube scheduler :- all containers actions and info about containers, where to create container or where to manage…

Etcd :- “high available key storage” like backend store, it stores all master and worker configuration…

Kube controller manager :- checks container status with desired status in workers nodes…holds and notifies all info about worker nodes and containers status info whether working properly or not…

🡪if any nodes are failed or not working properly manager will notifies to kube scheduler to create new nodes or restart them…

**Worker Node:**

**Kubelet :-** it will creates pods having containers in the pods and make interaction between pods and container and sends info about pods and containers to control manager as well…

**Kube-Proxy** :- for networking configuration and communication between different worker nodes…

Create pod

Expose

Access

1.kuber objects

2.delpoy

3.expose

4.yaml

5.master and slave

Objects, deployment

🡪pod

🡪service

🡪replica set

🡪deployment

**Namespace**:

Default, kube-node-lease,kube-public,kube-system

Default :- when we don’t create any namespace with name it will created in default namespace.

Kube-node-lease :- it holds lease objects associates with each node,allow kublet allow heartbeat to send node failure to controller.

Kube-piblic :- for cluster usage

Kube-system :-contains all the components related k8s cluster

---

apiVersion: "apps/v1"

kind: "Deployment"

metadata:

  name: "demo-calc-app-v1"

  namespace: "default"

  labels:

    app: "demo-calc-app-v1"

spec:

  replicas: 3

  selector:

    matchLabels:

      app: "demo-calc-app-v1"

  template:

    metadata:

      labels:

        app: "demo-calc-app-v1"

    spec:

      containers:

      - name: "demo-sha256-1"

        image: "gcr.io/iron-gizmo-348614/demo@sha256:db50d725a54581b642ab94211ad7ec804a7c0ba7b20604344a5f368ee41c42c8"

---

apiVersion: "autoscaling/v2beta1"

kind: "HorizontalPodAutoscaler"

metadata:

  name: "demo-calc-app-v1-hpa-lni9"

  namespace: "default"

  labels:

    app: "demo-calc-app-v1"

spec:

  scaleTargetRef:

    kind: "Deployment"

    name: "demo-calc-app-v1"

    apiVersion: "apps/v1"

  minReplicas: 1

  maxReplicas: 5

  metrics:

  - type: "Resource"

    resource:

      name: "cpu"

      targetAverageUtilization: 80

sudo usermod -aG docker $USER && newgrp docker

k8s steps in gcp GUI: