**Profile:**

-----------

My self Charan.

Having overall 6+ years of experiences as a Dev Op’s and AWS Solution Architect and Cloud Automation Engineer.

Currently working as a Senior Software Engineer for R Systems International Pvt Ltd in Noida.

My Current Project is WebPT, i.e AWS Cloud Migration Project along with DevOp’s Implementation and Cloud Automation.

We are having Dot Net Silver light application i.e R2 and R4 for our on-premises.

So that we need to migrate to our on-premises application to aws cloud using with automation templates.

Our project is mainly bifurcate in to 3 components as Application, Database and File System migration

So that need to migrate every component over cloud to implementing along with CI/CD Pipeline,

**Tech Stack:**

--------------

AWS services,EC2,RDS,Cloudwatch,IAM,Terraform,Packer,Atlantis,Jira,slack,confluence,GitLab,gitlab ci,

**Responsibilities:**

==============

1. Involved the architecture and designing of the Aws cloud Migration Infrastructure.
2. Used **GitLab** as a Source code Management Repository
3. Using Terraform to provision the infrastructure over AWS Cloud
4. Created **Terraform** templates with modular basis to provision application, database and file system migrations
5. Configured **GitLab CI** pipeline to run Terraform jobs to create multiple environments and run the test cases
6. Used **Packer** to build AMI and publish aws cloud
7. Used **Atlantis** to run the text fixtures and Live Infra environment
8. Configured AutoScaling, Load Balancing, RDS,EC2,Storage gateways,cloud watch components
9. Used to written kubernetes configuration files for the creation of different objects
10. Used EKS and GitLab CI k8’s cluster to deploy application
11. Written no of docker files to creation of building images
12. Used Cloud formation to written cloud formation templates
13. In earlier project supposed to work with IOT,Analytics,Lambda functions,API Gateways,DynamoDB tables,docker,docker compose,Python Boto
14. Used Jenkins,docker,ansible and vagrant to configure ci/cd pipeline with automation
15. Follows agile and scrum methodologies
16. Used slack,confluence,JIRa for project management.

Tasks:

====

1. Using Developers Tools to configure CI/Cd Pipeline(**Code Commit/Build/Pipeline/deploy**)
2. IOT Core,GreenGrass,Kinesis,EMR,Elastic Search,Lambda,API Gateway,Cognito,DynamoDB

Docker-Compose,docker,Kubernetes

Python-BOTO

Cloud formation

ELK

1. AWS Migration Project (DONE)

(VPC,EC2,Auto Scaling, Load Balancer, IAM, RDS, DMs,S3,Glacier,storage Gateway,EFS,

SNS, Cloud watch,ECS,EKS,ECR,SystemsManager,Certificate Manager,Route 53)

Docker,Kubernetes

Terraform,Packer,atlantis,GitLab CI,GitLab,

1. Docker,Jenkins,Vagrant,Ansible,
2. Apache,Tomcat.Sql server,Jenkins,Maven,git,Nexus,git hub,Ansible,Shell Script

**Build & Release Engineer Tasks**

-----------------------------------------

1. Installations of Git,Maven,Jenkins,Nexus,Tomcat,Vagrant
2. Configurations of Git,Maven,Jenkins,Nexus,Tomcat,Vagrant
3. Provisioned Dev/Test/UAX environments using Vagrant
4. Configured Free Style Jobs using CI
5. Performed CI/CD operations

**Cloud Automation**

-------------------------

1. AWS Infrastructure has been provisioned using with Python Boto3 SDK
2. AWS Infrastructure provision with Ansible. Creation of AWS resources using yaml files.
3. Provisioning Of Development/Testing Environment with Ansible and Vagrant
4. Infrastructure automation using with Cloud Formation stack
5. Cloud Infrastructure provision using with Terraform Modules
6. Configured Jenkins Jobs to provision infrastructure using with Terraform.
7. Configured GitLab CI Jobs to provision infrastructure using with Terraform Modules.
8. CI/CD implementations using Jenkins pipeline jobs
9. CI/CD implementations using Docker-Compose files
10. CI/CD pipeline implemented by using AWS Devloper Tools(https://blog.pythian.com/build-ci-cd-pipeline-using-aws-developer-tools/)

**TIPS:**

=====

[**https://github.com/javahometech/docker-compose**](https://github.com/javahometech/docker-compose)

Automated CI/CD using Docker and Jenkins(https://medium.com/@vineetcloudops/a-tutorial-about-continuous-integration-and-continuous-delivery-by-dockerize-jenkins-pipeline-ca377b02889b)

Java home cloud

Versions:  
------------

Jenkins: 2.150.1

Ansible: 2.5

Docker:

Java : 1.8

Tomcat: 8.0

Git : 2.14

**Cloud Formation Templates:**

1. AWSTemplateFormatVersion, Description, Mappings, Outputs, Parameters, Resources
2. dry-run = noop mode : [Change Sets](https://aws.amazon.com/blogs/aws/new-change-sets-for-aws-cloudformation/).
3. Aws cloud formation create stack, create change-set,Create-stack, update-stack ,delete-stack describe-stack list-stacks

K8’s **Architecture**

---------------

**Master:** Managing the k8’s cluster. Entry point for admin tasks

**1. API server:** Admin tasks with in master node. Validates and process requests

**2. Scheduler:** Schedules the work in to different each worker node.

**3. Controller manager:** regulates the state of the k8s state. Current state is same as the desired state.

**4. Etcd:** Distributive key-value store to store the cluster state.

**Node:** Controlled by the master node

**1. pod:** logical collection of one or more container

**2. kubeproxy:** Network proxy runs on each pod

**3. kubelet:** Runs on each containers, and make sure all the containers are in healthy state.

**4. container runtime:** Containers

**Service** 🡪 type of k8s resource is configured to forward requests to a set of pod. Have an ip address and this ip address routes to a healthy pod.

1.**cluster ip** -- default service, reachable with in cluster ip

2.**node port** 🡪 each node ip at static port, outside the clusters

3. **load balancers** 🡪 exposes the services externally it will automatically created

4. **external name** : map services to the contents of the external name field by returning a cname record with its value , No proxy of any kind is set up

**Replication Controller:** Managing the pod life cycle.Supports EQUAITY based selectors**.**it looks only exact names/words

**Replica Sets:** replacement of RC**.** Supports SET based selectors

**Deployment Controllers**

1.Create Deployement2.Update Dep3.Rollback dep4.Scale Depl5.Pause Dep

K8’s used for Auto Scaling,Load Balncing,Healing,Scheduling,horizontal scaling,roll out’s and rollbacks

Kubectl is the platform using which you can pass commands to the cluster.

**Controller Manager Types:**

1. Node Controller: Manage status of nodes
2. Replication: Maintains no of pods for every replication object
3. Service account & Token Controller : create default ac & api token controller
4. Endpoint controller: Takes care of end point objects pods, services

Dedicated Host - You pay for the entire physical server

Dedicated Instances - You pay , but they get placed on whatever dedicated hardware Amazon decides.

**Instance Types:** On-demand, Reserved, Spot

Limits : EC2:20,Elastic IP’s :5,VPC:5,Subnets:200,ASG:200,LB:20

**Classic Load Balancer:** when you have an existing application running in the EC2-Classic network.http/s,tcp

**Application Load Balancer:** flexible feature set for your web applications with HTTP and HTTPS traffic.

**Network Load Balancer**: TCP, when you need ultra-high performance, handling millions of requests per second.

A **security group** is a set of firewall rules that control the traffic

**Auto Scaling** to manage Amazon EC2 capacity automatically,

**S3 Permissions :**

**1.Public access settings:** a)Manage public access control listsb) Manage public bucket policies

2. **Access Control List**: Access for u r ac, Access for other aws ac’s, Public Access,S3 log delivery group

3.**Bucket policies**

4.**CORS configuration**(Cross origin resource sharing)

**Storage Life Cycle**

Standard(Frequently access -- Standard-IA(InFrequently access) -- Reduced Redundancy (Frequently access,non critical data) – Glacier( data archiving)

**Storage Gateway** : connecting an on-premises with cloud to provide seamless and secure integration.

**route table** contains a set of rules, called **routes.** network traffic is directed

(**NAT**) **gateway** to enable instances in a private subnet to connect to the internet or other **AWS** services,

**network ACLs** with rules, IN Bound / Out Bound Rules/subnet Associations

**Route 53** to register new domains, transfer existing domains, route traffic for your domains to your AWS and external resources, and monitor the health of your resources

Using volume can able to create snapshot (Volume mandatory).

Using snapshot can able to create volume and Image

**Amazon Cognito** offers user pools and identity pools.

**Amazon Simple Queue Service (SQS)** is a reliable, scalable, fully-managed message queuing service.

A **Lambda function** can be triggered by any of a long list of event sources

**AWS API Gateway**, to invoke a simple Lambda function that handles GET, POST and DELETE requests

**Amazon ECS** to deploy, manage, and scale Docker containers running applications, services, and batch processes.

**Journeling :** It is a dedicated area in the file system where all the changes are tracked(tune2fs

to convert ext2 file system to ext3 file system)

journalctl(all the log files between two different timings and by default **/run/log** location)

auditctl(to see the audit report)

# sar(sar utility is to watch the system activity report like CPU, memory,...etc.,)

Increase the size of the logical volume by **# lvextend or # lvresize** commands.

Then finally update the file system by **# resize2fs or # xfs\_growfs** commands.

# stat <file name/directory name> (to see the statistics of the file or directory)

**How will you lock a user, if he enters wrong password 3 times?**

# vi /etc/pam.d/system-auth

auth required pam\_tally.so no\_magic\_root  
account required pam\_tally.so deny=3 no\_magic\_root lock\_time=180

|  |  |
| --- | --- |
| TCP/IP | UDP |
| Transmission Control Protocol | User Datagram Protocol |
| It is connection oriented | It is connection less |
| Reliable | Non-Reliable |
| TCP Acknowledgement will be sent / received | No Acknowledgement |
| Slow communication | Fast communication |
| Protocol No. for TCP is 6 | Protocol No. for UDP is 17 |
| HTTP, FTP, SMTP, ....etc., uses TCP | DNS, DHCP, ....etc., uses UDP |

A subnet mask allows the users to identify which part of an IP address is reserved

A Gateway is the network point that provides entrance into another network.

# nmap <IP address or hostname of the remote system>

**#**netstat -ntulpRunLevels: 0 Power off-1 Single user-2 Multi user without network-3 Multiuser with network-4Development purpose-5 GUI-6 Restart

**troubleshoot if the network is not reaching?**

# ethtool # ifconfig # uptime # service network # Chkconfig # route

# netstat -r (to check the default gateway and routing table)

# route (to check the default gateway with routing table)

# /etc/rc.local (is the last script to be run when the system is booting)

# netstat -r (to see all routing table information)

In Linux systems the booting is done in 6 stages.

* BIOS
* MBR
* GRUB
* Kernel
* Init
* Runlevel

# lsmod (to list all the currently loaded modules)

# modprobe -r <module name> (to remove the specified module)

**# ssh-keygen**

* /home/<user name>/.ssh
* id\_rsa (private key) and id\_rsa.pub (public key).
* # ssh-copy-id -i <user name>@<IP address of the remote system>
* Go to remote system and check whether the above key is copied or not by # cat /home/<user name>/.ssh/authorized\_keys file
* # ssh <user name>@<remote hostname or IP address

(i) Full backup (Entire file system backup)

(ii) Incremental backup (backup from the last full backup or incremental backup)

(iii) Cumulative or differential backup (backup from last full backup or cumulative backup)

# vmstat is the command to the complete information on virtual memory like no of processes, memory usage, paging memory, block I/O (input /output), traps, disk and CPU activity.

Mstat(a,d)

# iostat (to see the Input and Output statistics in the Linux system)

# dmidecode (to see the complete hardware information of the system)

# iptables -A INPUT -i eth0 -p tcp -m tcp --deport 80 -j ACCEPT

# iptables -A OUTPUT -i eth0 -p tcp -m tcp --deport 80 -j ACCEPT

# firewall-cmd --permanent --add-service=http

Take a backup of the database.

**# mysqldump -u root -p <database name>><file name with full path>**

ansible-galaxy [delete|import|info|init|install|list|login|remove|search|setup]

Setup ModuleTo get information about the network or hardware or OS version or memory related information

#### Command Module executes a specific command on the target machine and gives the output.

#### Shell Module To execute any command in the shell of your choice you can use the Shell module.

#### User Module Using this module one can create or delete users.

#### File Module This module is used to create files, directories, set, or change file permissions and ownership etc

Copy Module It is used for copying files to multiple target machines.

Yum

Apt

Service

**replace**module

**archive**module

**debug module**

**get\_url module,**

# lineinfile is used to add additional or append lines to existing files.

**Notify 🡪 Service 🡪 Handlers(Restart)**

**TAGS:** need to execute the tasks with the tag niranjan then the command to run would be:

$ ansible-playbook preposttagseg.yml --tags niranjan

Jinga:Ansible Templates can help us make our configuration files dynamic and re-usable.

Ansible supports running a playbook in dry run mode (also called Check Mode

# ansible-playbook --check playbook.yml

**Build Triggers**

Triggers build remotely -🡪 Authentication Token

Build after another projects are built 🡪 Projects to watch ,Trigger obnly if build is stable,unstable,fails

Build periodically 🡪 schedule

Github hook trigger for git scm poling

Poll scm 🡪 Schedule

**Env Variables :**

------------------

1.Jenkins\_home2. user4.Path5.PWD6.Logname9.WorkSpace directory 10.Build Record Root Directory :

11. Project Name12. BuildNo13. Build Status14. Build URL

Role Based authorization strategy

Jenkins Pipeline (or simply "Pipeline") is a suite of plugins which supports implementing and integrating continuous delivery pipelines into Jenkins.

*Jenkinsfile (Declarative Pipeline)*

pipeline {

agent any/none

stages {

stage('Deploy') {

steps {

options(timeout,retry)

}

}

environment {

DISABLE\_AUTH = 'true'

DB\_ENGINE = 'sqlite'

}

Args

}

Always,success,failure,stable,unstable

**Pipeline steps**

1. **Push the code to source code repo**
2. **Static analysis**
3. Pre deployment testing
4. Packaging and deployment to the test environment
5. **Post deployment testing**

Core concepts:

----------------

Build life cycle(default,clean,site)mvn clean package site

-🡪 build phases

--🡪goals

Generate-sources,Clean,compile,test compile,test,package,integration test,install,deploy

Install 🡪 copy artifact to the local repo

Deploy 🡪 copied to remote repo

Dependencies(external jars files)

build plugins(extra goals to achieve in build phase,add a plugin to the POM file)

common plugins: clean,compiler,surefire,jar,war,Javadoc,

build profiles(build project for diff env’s)

POM file should be located in ROOt directory of project belongs to

Settings.xml : Configure settings for maven across all maven pom files.configure:

1. Location of local repo
2. Active build Profile

Maven Installation dir : $M2\_Home/conf/settings.xml

User’s Home dir : ${User.Home}/.me/settings.xml(It over rides the other file)

Types

---------

1. Per Project -- > Defined in project pom.xml
2. Per user 🡪 defined maven settings.xml (.m2/)
3. Global 🡪 Defined maven global settings.xml file(/conf/settings.xml)

Profile Activation

----------------------

1. Explicitly using command console input
2. Through maven settings(.m2/)

<activeProfile>Test</activeProfile>

1. Based on default variable

<Profile><id>test</id></profile>

<activation>

<activeByDefault>true</activeByDefault>

</activation>

</profile>

1. Os settings

Activation element to include os details.this triggers when the system in windows

<activation>

<os>

<name>Windows</name>

<family>windows

</os>

</activation>

Repo:

====

3 types

1. Local 🡪 User Home Dir
2. Remote 🡪Developers custom repo,project jar’s ,lib’s

POM.xml

<Repositories>

<url>

1. Central 🡪 Maven community

Maven checks new **snapshot** version in remote repo for every build. Automatically fetches latest every time.

If once **version** is downloaded never try to download a newer 1.o.if it need to be downloaded it must be upgraded to 1.1

**Dependency Scope**

1. Compile : Dependency is available in the class path of the project .default scope
2. Provided: Dependency is provided by jdk/web server/container at runtime
3. Runtime: Dependency not required for compilation but needed for execution
4. Test: Dependency need only for test compilation and execution phases
5. System: Provide the system path

[Jenkins Thin Backup](https://wiki.jenkins.io/display/JENKINS/thinBackup) is a popular plugin for backing up Jenkins.

Stash:Stores unstaged changes

Role -- Which is used to provide access to any other aws services.

DB Parameter group to your instance, you are assigned a default Parameter group which has the default values.

backups in two ways: automated backups,DB Snapshots,

**read replicas**i.e. snapshots are created from a source DB and all the read traffic to the source database is distributed among the read replicas, this reduces the overall overhead on the source DB.

An IoT thing is a representation and record of your phyisical device in the cloud.

AWS IoT Greengrass is software that extends cloud capabilities to local devices

Amazon DynamoDB is the latest NoSQL offering from AWS

**Amazon Kinesis:**Easily collect, process, and analyze video and data streams in real time, so you can get timely insights and react quickly to new information

Amazon Elasticsearch Service (makes it easy to set up, operate, and scale an Elasticsearch cluster in the cloud.

Ansible playbook [cd.yml](https://github.com/vfarcic/jenkins-docker-ansible/blob/master/ansible/cd.yml) Role is run.

- hosts: localhost

  remote\_user: vagrant

  sudo: yes

  roles:

    - java

    - docker

    - registry

    - Jenkins

|  |
| --- |
| roles: |
| - etcd |
| - confd |
| - docker |
| - nginx |
| - bdd |

a service on GitHub to call the Jenkins Github webhook because we want to trigger the pipeline. To do this, go to Settings -> Integrations & services. The Jenkins Github plugin should be shown on the list of available services, as below.

**kubeadm join --token <token> <master-ip>:<master-port> --discovery-token-ca-cert-hash sha256:<hash>**

# lsof <device name> (to check the files which are open in that mount point)

# uptime(to see the information from how long the system is running, how many users login and load average)

Zombie Process: parent process will run with out knowing child process

Sar(p-cpu,q-load average,r-memory,s-swap,d-disk,b-i/o stat,

$# = number of arguments.

$@ = what parameters were passed.

$\* holds list of all arguments passed to the script

$? = was last command successful.

"$$" means current PID.

"$!" is the PID of the last program your shell ran in the background

**set command** to **set** or unset values of **shell** options and positional parameters.

The cut command in UNIX is a nice utility program which allows you to cut data from a text file.  
  
**Docker File**

* FROM – Image download from docker hub registry
* MAINTAINER – Specifies about the author
* LABEL – specify metadata about image
* ENV - setting environment variables with key and values.
* ARG - setting environment variables with key and values only applied over build image
* RUN - run commands over image and creates a new layer.
* CMD – to be executed when running a container, only one [ shell and Executable form]
* ENTRY POINT – to configure and run a container as an executable
* VOLUME – create a volume to the container
* USER – set uid,gid
* WORKDIR - set the working directory.
* EXPOSE - get your ports right.
* ADD – for downloading remote resources, extracting TAR files, etc
* COPY - for simply copying files and/or directories into the build context
* ONBUILD – the image in the docker file is used as a base image for another image

CMD is to provide defaults for an executing container.

ENTRYPOINT helps you to configure a container that you can run as an executable.

* CMD sets default command and/or parameters, which can be overwritten from command line when docker container runs.
* ENTRYPOINT command and parameters will not be overwritten from command line. Instead, all command line arguments will be added after ENTRYPOINT parameters.

HA in K8’s

Zero-DownTime-Deployment in K8’s

A list is a sequence of elements in a specific order.

A set has no order, contains an element is much faster, handy operations such as union and intersection.

A tuple is basically an immutable list, meaning you can't add, remove, or replace any elements.

A dictionary is a mapping from keys to values

Docker images are stored in two files as shown by following command

$ docker info

Data file: /var/lib/docker/devicemapper/devicemapper/data

Metadata file: /var/lib/docker/devicemapper/devicemapper/metadata

docker -D info(Debugging Info)

**Playbook:**

Hosts,Tasks,Vars,Handlers,MetaData,remote-user,Roles

propagate: false flag to move to next stage when previous stage fails.

ps –ef LIST ALL PROCESSES THAT ARE CURRENTLY RUNNING IN THE BACKGROUND

**Top/ps command** to display all running process: # ps -aux

**Python namespace** is a simple system to control the names in a program. It ensures that names are unique and won’t lead to any conflict.

**decode()** decodes the string using the codec registered for *encoding*

encode --- encryption format

decode – decryption format

A lambda function can take any number of arguments, but can only have one expression

**Browser Request**

**----------------------**

You type maps.google.com into the address bar of your browser.

The browser checks the cache for a DNS record to find the corresponding IP address of maps.google.com.

If the requested URL is not in the cache, ISP’s DNS server initiates a DNS query to find the IP address of the server that hosts maps.google.com.

Browser initiates a TCP connection with the server.

The browser sends an HTTP request to the web server.

The server handles the request and sends back a response.

The server sends out an HTTP response.

The browser displays the HTML content (for HTML responses which is the most common).

DNS is a list of URLs and their IP addresses just like how a phone book is a list of names and their corresponding phone numbers.

In order to find the DNS record, the browser checks four caches.

* First, it checks the browser cache.
* Second, the browser checks the OS cache.
* Third, it checks the router cache.
* Fourth, it checks the ISP cache.

The purpose of a DNS query is to search multiple DNS servers on the internet until it finds the correct IP address for the website.

ISP’s DNS server a DNS recurs or whose responsibility is to find the proper IP address of the intended domain name of Name Servers.

Root Domain : ., First level Domain : .com, Second-level-domain : google, third –level : www

Searches by .com fitst

Once the browser receives the correct IP address it will build a connection with the server that matches IP address to transfer information.

TCP is the most common protocol used for any type of HTTP request.

Client to Server communication establishment process called **TCP/IP three-way handshake**

1. Client send SYNC(Synchronize)
2. Server ACK(acknowledge server for new connections)
3. Client receive ACK packet and sent ACK packet to server

The browser will send a GET request asking for maps.google.com web page.

Entering credentials or submitting a form this could be a POST request.( as browser identification)

In server,The request handler is a program (written in ASP.NET, PHP, Ruby, etc.) that reads the request, it will assemble a response in a particular format (JSON, XML, HTML).

1xx indicates an informational message only● 2xx indicates success of some kind● 3xx redirects the client to another URL

● 4xx indicates an error on the client’s part● 5xx indicates an error on the server’s part

[mod\_evasive](http://www.zdziarski.com/blog/?page_id=442) is an Apache module for preventing DoS attacks.Essentially, you set some limits for how many times an IP address can make requests to your site over a given interval. It's useful in a variety of contexts, but not well documented.

DOSSiteCount 50

**/trunk** is your standard, bleeding edge development. We use CI so this must always build and pass tests.

## -x option to debug a bash shell script

Run a shell script with -x option.  
$ bash -x script-name  
$ bash -x domains.sh

* **set -x** : Display commands and their arguments as they are executed.(Turn ON) set +x(Turn OFF)
* **set -v** : Display shell input lines as they are read.

**Function :** Set \_DEBUG to ‘on’ when you need to debug a script: \_DEBUG="on"

Sonarqube scanner plugin : Configure sonarqube server

sshAgent plugin

Slack Plugin  
Quality Gate status : pass Build is success otherwise fails.

Having multiple branches in github,use parameterized branch selection from the given job i.e option1,option2..

Properties : choose check box this job is parameterized.

parmateres, select the choices ‘option1\noption2\noption3’

**FailFast : True (If any branch fails in parallel branches automatically comes out from the job)**

**Propagate : False**

Slack : Incoming/Outgoing Webhook

Docker-compose : describe micro services on single host using with single yaml file and create dev and test environments easily

Network types : bridge,host,null

Bridges : default, custom

Containers by default associates Default Bridge network(Only with IP Address not names)

Ping one to another container : docker attach conatiner1

#Ping container1

Custom bridge network : docker network create –driver=bridge javahome

Docker run –itd –name=alpine1&2 –network=javahome alpine

Docker network inspect 1,2

**Zero-downtime Deployment**

1. Blue-Green Deployment

Maintains IDENTICAL environment. So that one env is deploed then moves to the another env with out downtime

RDS –DB: follow static schema model, So that some down time is needed for the deployment

Mongo-DB: dynamic schema model. So that no downtime is needed for the data sync-up

1. Rolling Deployment (pick some servers and deploy then pick another some servers and deploy…..)

Postman -- > API Gateway 🡪 Lambda 🡪 DynamoDB Table

Query Parameter ?: Lambda accepts query parameters from api gateway

An AWS **Lambda alias** is like a pointer to a specific **Lambda** function version.

**Docker Run**

**=========**

**1.docker create**: creates a writable container layer over the specified image and prepares it for running the specified command. The container ID is then printed to STDOUT. This is similar to docker **run -d** except the container is never started. We can then use the **docker start <container\_id>** command to start the container at any point.

Docker create –it fedora bash

Docker start –ai container ID

**2.docker rename** allows the container to be renamed

3. **docker run** creates and starts a container in one operation.

4. **docker rm** deletes a container.

**5.docker update** updates a container's resource limits. Example : updating multiple resource configurations for multiple containers

docker update --cpu-shares 523 –m 300M container ID image name

docker logc Id

**6. docker inspect** looks at all the info on a container (including IP address)

**Docker Info**

==========

1. **docker events** gets events from container.
2. **docker port** shows public facing port of container.
3. **docker top** shows running processes in container.
4. **docker stats** shows containers' resource usage statistics.
5. **docker diff** shows changed files in the container's FS.
6. **docker ps** shows running containers.
7. **docker logs** gets logs from container.
8. **docker inspect** looks at all the info on a container

**Container start/stop**

1. **docker start** starts a container so it is running.
2. **docker stop** stops a running container.
3. **docker restart** stops and starts a container.
4. **docker pause** pauses a running container, "freezing" it in place.
5. **docker unpause** will unpause a running container.
6. **docker wait** blocks until running container stops.
7. **docker kill** sends a SIGKILL to a running container.
8. **docker attach** will connect to a running container.   
   **Image create/remove**
9. **docker images** : shows all images.
10. **docker import** : creates an image from a tarball.
11. **docker build** : creates image from Dockerfile.
12. **docker commit** : creates image from a container, by default, the container being committed and its processes will be paused while the image is committed.
13. **docker rmi** : removes an image
14. **docker load** : loads an image from a tar archive as STDIN, including images and tags
15. **docker save** : saves an image to a tar archive stream to STDOUT with all parent layers, tags & versions

**Container info**

Getting Docker Container's IP Address from host machine:

Docker inspect

**Image info**

1. **docker history** shows history of image.
2. **docker tag** tags an image to a name (local or registry).

**Network create/remove/info/connection**

1. **docker network create**
2. **docker network rm**
3. **docker network ls**
4. **docker network inspect**
5. **docker network connect**
6. **docker network disconnect**

**Docker Repo**

1. **docker login** to login to a registry. (see docker push)
2. **docker logout** to logout from a registry.
3. **docker search** searches registry for image.
4. **docker pull** pulls an image from registry to local machine.
5. **docker push** pushes an image to the registry from local machine. First, we'll install nginx on top of ubuntu,

**docker-export** - Export a container's filesystem as a tar archive and it removes the history, layers and entrypoint (flattening the image), run against a container.

docker export <container ID> > image-name.tar

**docker import:** Imports the tarball into a repo image.

**docker save:** Saves an image to a tarball, this preserves the history, layers and entrypoint, run against an image.  
**docker load:** Loads an image from a tarball into a repo image.  
**docker commit**: Creates an image from a modified container preserving entrypoint, layers and history.

**Continuous Delivery with AWS CodePipeline and CodeBuild**

<https://github.com/dishanrajapaksha/webinar-aws-codepipeline-codebuild>

**List : [1,2,4]**

Allows duplicates, Ordered collection, Mutable, Supports slicing

**Set :** { 1,3,’a’}

Doesn’t eliminates any duplicates, Un Ordered elements, doesn’t supports indexed based operations,Mutable

**Dictionary :** {Key:Value}

Mutable, nested lists or dict’s.

**Continue** – It stops the iteration, next go to the next iteration

**Break** – It stops completely the loop

**Git :** decentralized version control,commits possible offline,push/pull faster

SVN : centralized version control,only online commit,push/pull slow

**Git pull** command pulls new changes or commits from a particular branch from your central repository and updates your target branch

Git fetch it pulls all new commits from the desired branch and stores it in a new branch in your local repository.

Git pull = git fetch + git merge

To get a list files that has changed in a particular commit **git diff**

**Branching Strategies :**

**Featurebranching :** A feature branch model keeps all of the changes for a particular feature inside of a branch.

Task branching implemented on its own branch with the task key included in the branch name.

Release branching Once the develop branch has acquired enough features for a release, you can clone that branch to form a Release branch.

**revert previous commit in git?**

* To nuke changes in the commit: git reset --hard HEAD~1
* To undo the commit but keep your changes: git reset HEAD~1
* To undo your commit but leave your files and your index git reset --soft HEAD~1

Git rebase : merge, remove unnecessary commits, maintain linear history

Stash – stores unstaged changes in to a package and stored inmemory.

Git statsh list(list), git stash pop,git stash apply stash@{0},git stash drop stash(0}

NameSpace : Isolating resources per process(or group of processes)

Control Groups(C Group) : Limited amount of resources used for per process

Docker run = docker create + docker start

Docker system prune(Remove images,stopped container,iamges cache,dangling images)

NameSpaces : Isolating resources per process/group of processes(Processes,HD,Network,User,Hostnames)

CGroups : Limit amount of resources used per process(Memory,cpu,IO,Network)

Docker run = docker create + docker start

Docker system prune(stopped,network,un used)

Docker stop 🡪 sigterm pass message ,graceful

Docker kill -🡪 sigkill, no message has been send,forceful

Docker commit 🡪 Mannual image generation using with the running container

Exit code-0

**Restart Policies(Compose)**

----------------------------------

1. Restart : No
2. Restart : Always
3. Restart : OnFailure
4. Unless Stopped

**Volumes** : References has been made with the local host,Any kind of latest updates and deployments has been applied to over machine,automatically reflected over my container and deployed the changes with out performing any kind of builds

**Volume BookMark** -v Bookmark : Create the book mark over docker RUN command