Contents…

INTRODUCTION........ Error! Bookmark not defined.

DAILY TASK........ Error! Bookmark not defined.

Major challenges........ 2

Linux OS directory structure........ 3

GENERAL ISSUE: Cannot access the server........ 4

PERFORMANCE ISSUE........ 4

Virtual machine running slow........ Error! Bookmark not defined.

Backup & DISK SPACE FULL ISSUE........ 5

USER LOGIN ISSUES........ 5

SSH issues........ 5

KERNEL UPDATE........ 5

PATCHING PROCESS:........ 6

Building a server........ 6

FILE SYSTEM:........ 6

LVM........ 8

Apache Tomcat........ Error! Bookmark not defined.

Directory structure in tomcat........ Error! Bookmark not defined.

USER MANAGEMENT........ 8

DNS........ Error! Bookmark not defined.

Process management........ 9

SENDMAIL........ Error! Bookmark not defined.

NFS server........ 7

Network bonding........ 10

SUDO configuration........ 10

AWS........ Error! Bookmark not defined.

EC2........ 13

Unable to login to EC2 instance........ Error! Bookmark not defined.

Security in Amazon EC2........ Error! Bookmark not defined.

S3........ 14

IAM........ 16

VPC........ 17

Auto scaling........ 20

Load balancing........ 21

Route 53........ 21

SSL certificate(aws certificate manager)........ Error! Bookmark not defined.

CDN (Cloud front)........ Error! Bookmark not defined.

AWS backup........ 23

site migration........ Error! Bookmark not defined.

OS HARDENING........ Error! Bookmark not defined.

DISASTOR RECOVERY .... **Error! Bookmark not defined.**
GIT 32
MAVEN 35
JENKINS 39
ANSIBLE 56
DOCKER 44

glpat-HETa9yvesXbojEK8Axjb

Now,I am looking for an opportunity to expand my Skills and knowledge and learn more about Cloud computing and Linux administration.

Back to first page

Back to first page

Major challenges=

- **Analyzing and troubleshoot the issues used to take much time , that was one of the major challenge at the beginning of my career, eventually experiencing more and more issues it became much easier than what it used to be. And**
- **there r plenty of services in AWS, most challenges would be to chose right fit with most cost-effective way, this is the one challenge I am bettering and I always seek my seniors advice because it may effect end result of the company by choosing the wrong services.**

[Back to first page](#)

Boot Process

- When the computer is turned on, it goes the BIOS mode (Basic Input/Output System) is loaded. And then it performs a self-test of the hardware and initializes the system's components.
- Once the BIOS mode has completed its self-test, it looks for a boot loader on the bootable device. And then it loads the Linux kernel into memory and starts the processer
- And then the kernel detects and configures the system's hardware components, like CPU, memory, and storage devices.
- and Then init system is responsible for starting system services and processes. We have init0,1,2,3,4,5,6 of runlevels
- and then the user space is initialized. And then it starts user-level services, like networking and graphical interfaces.
- Once the user space has been initialized, the login screen appears, and the user can log in to the system.
- Runlevels:
- Init 0-shutdown the process
- Init 6- reboot the process
- Init 1-single user
- Init 2 – muilti user without networking
- Init 3 –muiltiuser with networking
- Init 4-multiuser without graphical user mode
- Init 5 –muiltiuser with gui mode

Linux OS directory structure.

  - **/bin** – binary or executable programs.
  - **/etc** – system configuration files.
  - **/home** – home directory. It is the default current directory.
  - **/opt** –third-party software.
  - **/tmp** – temporary space
  - **/usr** – User related programs.
  - **/var** – log files.
- **/boot-** It contains all the boot-related information files and folders.
- **/dev –** device files such as dev/sda1, dev/sda2, etc.
- **/lib –** It contains kernel modules and a shared library.
- **/media –** It contains subdirectories where removal media devices inserted.
- **/mnt –** It contains temporary mount directories for mounting the file system.
- **/proc** – It is a virtual and pseudo-file system to contains info about the running processes with a specific process ID or PID.
- **/run –** It stores volatile runtime data.
- **/sbin –** binary executable programs for an administrator.
- **/srv –** It contains server-specific and server-related files.
- **/sys –** It is a virtual filesystem for modern Linux distributions to store and allows modification of the devices connected to the system.

Back to first page

GENERAL ISSUE: Cannot access the server.

Whenever this issue is raised, we must see which is the source and from which ip they are trying to connect.

What is the target application and IP

And port number

Through which protocol they are trying to connect.

- **1st step is to ping the server**
- **Check the firewall whether they are blocking each other, like check for the security group rules whether they are allowed to communicate.**
- **Check whether the service they are trying connect is started.**
- **If severs are in 2 different VPCs, to communicate publicly we have to check for ICMP rule.for that, change the settings in the Security group inbound rule to allow the ICMP (all ports) to communicate.**
- **Then check the cables are properly connected for internet connectivity.**

SSH issue

- **Private key should have the permission of 400**
- **Check whether you entered username and ip properly, which you are trying to connect.**
- **Check whether user is allowed to ssh in config file** (/etc/ssh/sshd_config)
- **Check the ssh service started or not.**

Back to first page

PERFORMANCE ISSUE

- **Check the CPU performance & load avg using #top command**
- **Check the storage of the root files**
- **And check the Memory & Swap memory using the #free command**
- **Kill the zombie Process (top -b1 -n1 | grep Z**
- **Restart the application and check.**
- **And also check the server uptime and if more than 500 days then reboot the server by scheduling**
- **And check the recent Package update**
  #cat /var/log/yum.log

Back to first page

Virtual Machine running slow

- **Memory issue**
- **If application running in that vm is upgraded, then chances are more to get slow.**
- **Add the resources**
- **If All machines in hypervisor running same time, then vm may get slow.**

Back to first page

Backup &DISK SPACE FULL ISSUE

- Check the disk usage by #df -h command
- And list the file and check which files consuming more space #du -h
- Delete or archive the large files.
  #tar -cvf & gzip
- Extend the lvm to add extra storage.

Back to first page

USER LOGIN ISSUES

- If the user login shell is 'nologin' then login is not possible sochange the shell
- Reset the 'password and account expiry dates' in /etc/login.defs
- If the failed login is more then you cannot login so clear the pam_tally
  #pam_tally2 -u username –reset
- Lock and unlock the account using #usermod
- Delete the password and check

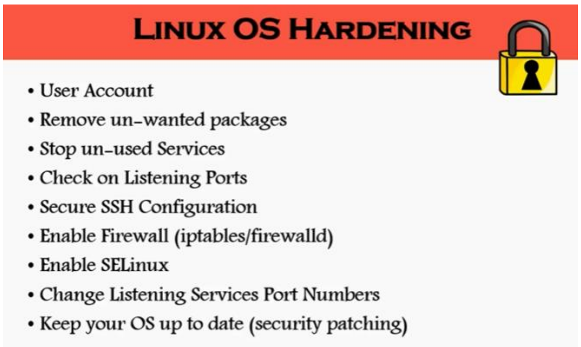
Back to first page

SSH issues

- Private key should have the permission of 400
- Check the username and ip
- Check whether user is allowed to ssh in config file (/etc/ssh/sshd_config)
- Check the ssh service started or not

Back to first page

OS HARDENING



LINUX OS HARDENING

- User Account
- Remove un-wanted packages
- Stop un-used Services
- Check on Listening Ports
- Secure SSH Configuration
- Enable Firewall (iptables/firewalld)
- Enable SELinux
- Change Listening Services Port Numbers
- Keep your OS up to date (security patching)

KERNEL UPDATE

- Take the backup of /boot files
- Or take the snapshot of the VM
- Download the repository using the link http://elrepo.org
- After downloading the repository enable the repo
- #yum –enablerepo=elrepo-kernel

- **Then install the package #yum install kernel-ml**
- **After installing reboot the machine.**

Back to first page

PATCHING PROCESS:

- **Get the downtime or schedule time from the client**
- **Raise the change request as per the schedule**
- **Collect the server pre configuration details like**
  ***Server uptime**
  ***Boot info**
  ***mountpoint info**
  ***disk info**
  ***Backup info**
- **Check with the application & Database team whether they need to exclude any package.**
- **After that submit the change request for CAB approval**
- **Once got the approval apply the patching as per the schedule time**
  #yum clean all
  #yum update
- **After that do the Post verification, check the kernel version & Notify to the client**
- **Close the change.**

Windows updates will be on 2nd Tuesday or every month .

We automated such a way that every third weekend aws system manager automatically scans the server check for pending updates and installs it with maintenance window of 2 hrs.

Back to first page

Building a server

- **Name of the server**
- **Version of the server**
- **Memory**
- **Harddisk**
- **Password**
- **Mountpoint**
- **Storage type (lvm or standard)**
- **Swap memory.**

Back to first page

FILE SYSTEM

- **#fsck = To Check and repair file system in ext4 or ext3 file system**
- **#xfs_repair= To Check and repair file system in xfs**

- Unmount the disk before running the #fsck
- #fsck -A = check all the filesystems except xfs

- Xfs is when application runs bigger files.
- Xfs is default filesystem in centOS

Convert ext2 to ext3

- Unmount the disk
- tune2fs -j /dev/vg/lv (enabling journaling)

Convert ext3 to ext4

- Unmount the disk
- tune2fs -O

Extend the xfs filesystem

- #xfs_growfs /dev/vg/lv

Wipe filesystem

- Unmount the disk
- #wipefs /dev/vg/lv

Back to first page

NFS server

```
# systemctl start nfs-server
```

NFS server is to share the mount point between 2 linux server.

#vi /etc/exports= is the configuration file for nfs

```
/mountpointpath 192.168.0.0/24(rw,no_root_squash)
```

#exportfs -av = it ll check whether mountpoint is shared or not.

NFS CLIENT:

```
#mount -t nfs 192.168.0.30:/mountpointoftheserver /locallycreatedmountpoint.
```

copy the files to the other user in different server.

```
#scp -pr <source> username@<server ip address>:/destination
```

Back to first page

LVM

*LVM used to manage block storage.
To Create the LVM: first Add a disk or volume to the machine then

```
#mkfs.ext4 /dev/sdb1 = format and assign new filesystem.

        #pvcreate /dev/sdb = initialize the disk for lvm use (physical volume create)
        # pvs

        #vgcreate -s 16 vgname /dev/sdb = it will create the volume group with the name
vgname , 16 is physical extend size in mbs.
        #vgs

        #lvcreate -l 12 -n lvnamevgname = -n is for custom name. -l is to feed size in logical extent

        #mount /vg/lv  /directory = it will mount the directory

        # lvrenamevolumegrouplvnameexisted newname
        # lvextend -L +100 /dev/volumegroup/LVname (+ sign is compulsory to increase size)
        #lvreduce -L 100 /dev/volumegroup/LVname
        # lvremove /dev/volumegroup/lvname
        # vgreduce /dev/sdb
        #vgremove vgname
        #pvremove /dev/sdb
```

Partitioning

```
#sudo parted /dev/sdbmkpart primary 1 101m = create a primary partition
        #sudo parted /dev/sdb print       =verify whether the partition is created or not.
        #lsblk /dev/sdb            = lists the disks and partitions.
        #mkfs.ext4 /dev/sdb1        = format and assign new filesystem.
        #mount /dev/sdb1 /directory   = it will mount the directory
```

Back to first page

```
/etc/fstab =  /dev/sdb1 /directory ext4 defaults 0 0
              /dev/VolGroup00/LogVol02 swap swap defaults 0 0    (swap memory)
```

Back to first page

USER MANAGEMENT

Passwd = username/password/uid/gid/description/home dir/shell

Shadow= 9 fields

Group= group name/passwd/GID/group members.

Gshadow= group name/passwd/admin/group members

```
#useradd -d = home directory

        -b= base dir

        -c = comment
```

-g= group

-G = secondary group

-s = shell

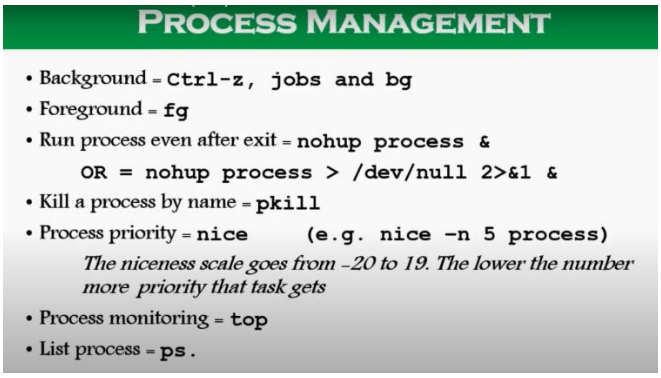
-l = lock account

-u = unlock

```
#setfacl -m u:username:rwx filename
```

[Back to first page]

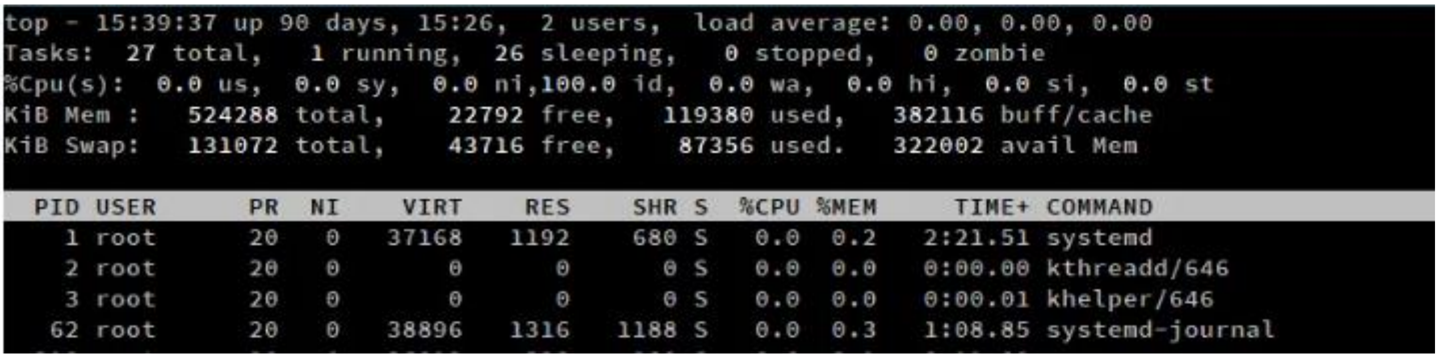
Process management



PROCESS MANAGEMENT

- Background = `Ctrl-z, jobs and bg`
- Foreground = `fg`
- Run process even after exit = `nohup process &`
  `OR = nohup process > /dev/null 2>&1 &`
- Kill a process by name = `pkill`
- Process priority = `nice` `(e.g. nice -n 5 process)`
  The niceness scale goes from -20 to 19. The lower the number more priority that task gets
- Process monitoring = `top`
- List process = `ps`.

#kill -9 process = kill the bg process.



```
top - 15:39:37 up 90 days, 15:26,  2 users,  load average: 0.00, 0.00, 0.00
Tasks:  27 total,   1 running,  26 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :   524288 total,    22792 free,   119380 used,   382116 buff/cache
KiB Swap:   131072 total,    43716 free,    87356 used.   322002 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
    1 root      20   0   37168   1192    680 S   0.0  0.2   2:21.51 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.00 kthreadd/646
    3 root      20   0       0      0      0 S   0.0  0.0   0:00.01 khelper/646
   62 root      20   0   38896   1316   1188 S   0.0  0.3   1:08.85 systemd-journal
```

[Back to first page]

#Crontab-e

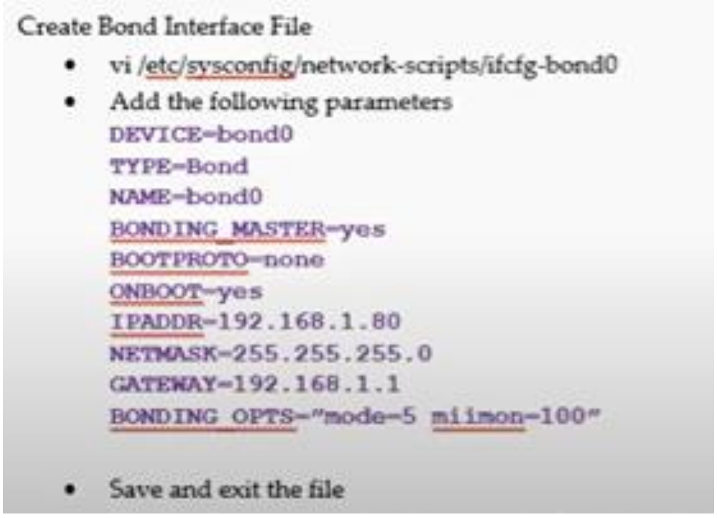
services are logged to the /var/log/cronfile.

This can be done by listing users in the allow or deny file located in the /etc

[Back to first page]

Network bonding

combing or joining two or more network interfaces together into a single interface

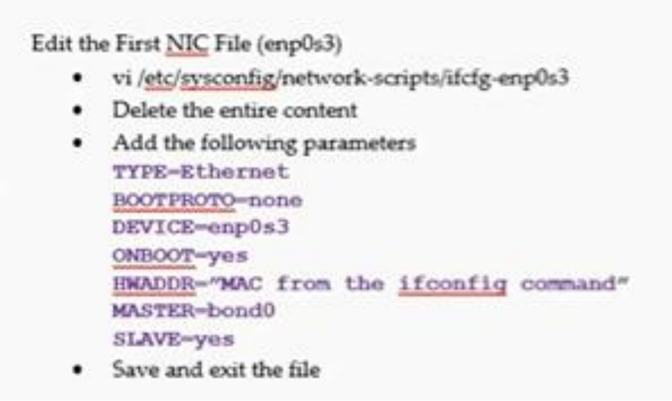


Create Bond Interface File

- vi /etc/sysconfig/network-scripts/ifcfg-bond0
- Add the following parameters

```
DEVICE=bond0
TYPE=Bond
NAME=bond0
BONDING_MASTER=yes
BOOTPROTO=none
ONBOOT=yes
IPADDR=192.168.1.80
NETMASK=255.255.255.0
GATEWAY=192.168.1.1
BONDING_OPTS="mode=5 miimon=100"
```

- Save and exit the file

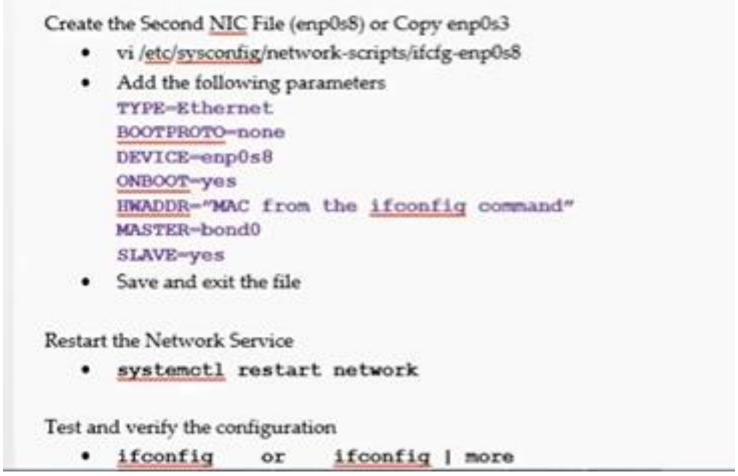


Edit the First NIC File (enp0s3)

- vi /etc/sysconfig/network-scripts/ifcfg-enp0s3
- Delete the entire content
- Add the following parameters

```
TYPE=Ethernet
BOOTPROTO=none
DEVICE=enp0s3
ONBOOT=yes
HWADDR="MAC from the ifconfig command"
MASTER=bond0
SLAVE=yes
```

- Save and exit the file



Create the Second NIC File (enp0s8) or Copy enp0s3

- vi /etc/sysconfig/network-scripts/ifcfg-enp0s8
- Add the following parameters

```
TYPE=Ethernet
BOOTPROTO=none
DEVICE=enp0s8
ONBOOT=yes
HWADDR="MAC from the ifconfig command"
MASTER=bond0
SLAVE=yes
```

- Save and exit the file

Restart the Network Service

- `systemctl restart network`

Test and verify the configuration

- `ifconfig`   or   `ifconfig | more`

Softlink and hardlink

A softlink is a pointer to a file or directory in a different location. It is essentially a shortcut to the original file or directory

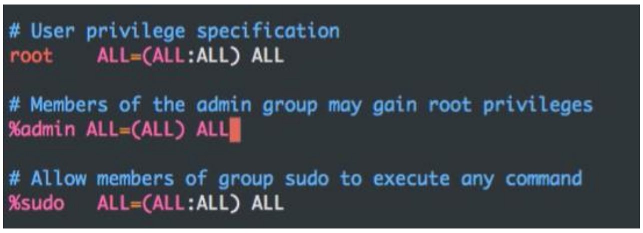
ln -s /path/to/original/file /path/to/softlink

A hardlink is essentially a duplicate of the original file or directory, and any changes made to either the original file or the hardlink will be reflected in both files.

ln /path/to/original/file /path/to/hardlink

SUDO configuration-------------------------------------

/etc/sudoers



```
# User privilege specification
root    ALL=(ALL:ALL) ALL

# Members of the admin group may gain root privileges
%admin ALL=(ALL) ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL
```

Back to first page

TOMCAT

Bin   conf   lib   logs   webapps

PERFORMANCE ISSUE

- **Check the CPU performance & load avg using #top command**
- **Check the storage of the root files**
- **And check the Memory & Swap memory using the #free command**
- **Kill the zombie Process**
- **Restart the application and check**
- **And check the server uptime and if more than 500 days then reboot the server by scheduling**
- **And check the recent Package update**
  #cat /var/log/yum.log
- /logs/Catalina.out here check for the logs.

  - **Status code 401 :** Make sure that the user name and password in your build.properties file match a user name and password with the role of manager in the tomcat-users.xml file
  - **Status code 403:** if a file misses some security context. Edit context.xml (/webapps/manager/META-INF/context.xml)
  - **Status code 404:** It refers to an error message that it is an HTTP response and the client was not able to communicate with the given server.
  - **Status code 500:** No Context Error, when attempting to run a deployed applicationThis error occurs, means that Tomcat is loaded, but it doesn't know about our application. If we have not deployed the application, then we have to deploy first.

    If Tomcat is loaded, but not yet loaded all contexts, we will get this error. For that Refresh on our browser until either the application loads or we get a different error message.

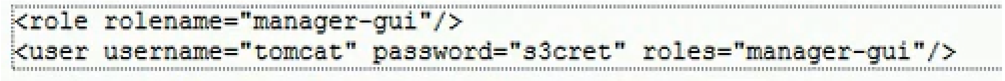
bin=(bin contains all the binaryfiles. **startup.sh shutdown.sh** etc…)

conf= Configuring the tomcat.

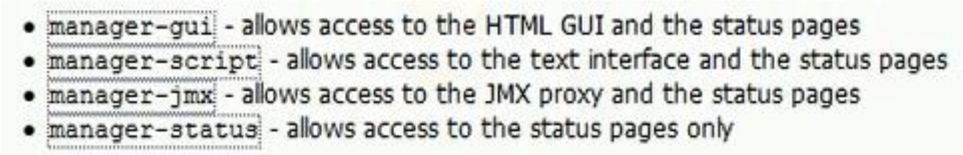
Context.xml ((/webapps/manager/META-INF/context.xml) = here comment out the allow localhost access only.

tomcat-users.xml (/conf/tomcat-user.xml) = here update the users and roles,**authorization** will be given to the users

server. xml (/conf/tomcat-user.xml) = to change the port restart the service if I make any changes in the configuration file.



```
<role rolename="manager-gui"/>
<user username="tomcat" password="s3cret" roles="manager-gui"/>
```



- `manager-gui` - allows access to the HTML GUI and the status pages
- `manager-script` - allows access to the text interface and the status pages
- `manager-jmx` - allows access to the JMX proxy and the status pages
- `manager-status` - allows access to the status pages only

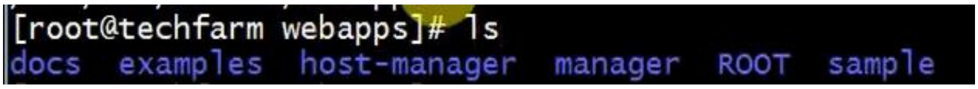
Lib= JAR files are deployed in this directory.(images or voices)

logs=(Catalina.out) **Back to first page**

readme.md=

running.txt=

webapps=it is the document root directory for application server. if you deploy any war files under webapps folder then data will be displayed under web pages.

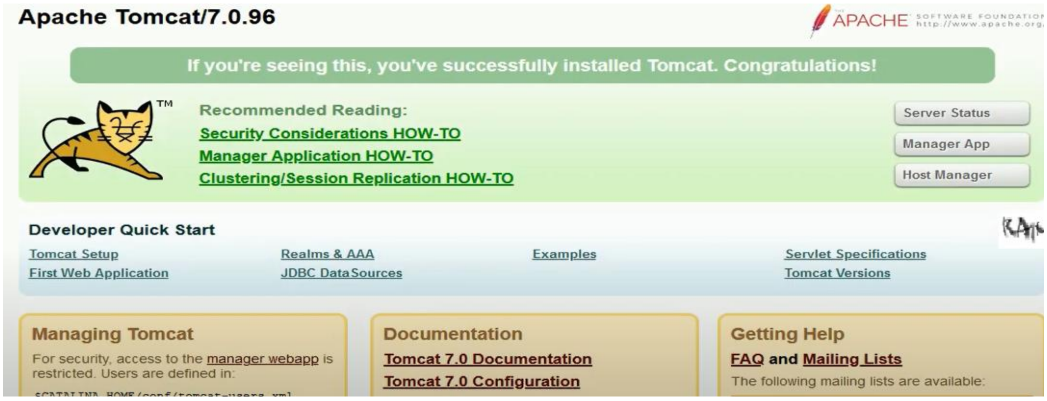


```
[root@techfarm webapps]# ls
docs  examples  host-manager  manager  ROOT  sample
```

to check the tomcat is running or not.

go to the browser and browse with ipaddress:8080

Back to first page



Apache Tomcat/7.0.96

If you're seeing this, you've successfully installed Tomcat. Congratulations!

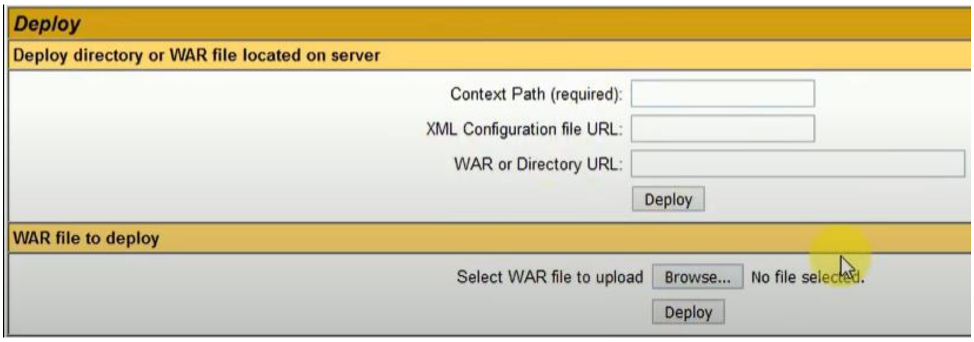
Recommended Reading:
Security Considerations HOW-TO
Manager Application HOW-TO
Clustering/Session Replication HOW-TO

Developer Quick Start
Tomcat Setup
First Web Application
Realms & AAA
JDBC DataSources
Examples
Servlet Specifications
Tomcat Versions

Managing Tomcat
For security, access to the manager webapp is restricted. Users are defined in:

Documentation
Tomcat 7.0 Documentation
Tomcat 7.0 Configuration

Getting Help
FAQ and Mailing Lists
The following mailing lists are available:



Deploy
Deploy directory or WAR file located on server
Context Path (required):
XML Configuration file URL:
WAR or Directory URL:
WAR file to deploy
Select WAR file to upload No file selected.

- Tomcat host the dynamic web applications.
- Apache Tomcat is basically a Web Server and Servlet system which is an open-source to run JSP and Servlets
- default port of Apache Tomcat is port 8080.
- name of the inbuilt Web Container in Tomcat is Catalina which is present in the bin directory

- There are basically two types of files with which we can Start and Stop the Server. Startup.bat Shutdown.bat
- configuration file that is used in Apache Tomcat is httpd.conf

AWS

- **We are using AWS for different different services. Mainly for, Computing, Storage, & networking.**
- **For computing we are using services like – EC2, Autoscaling & Load balancing.**
- **For storage we are using services like- For object storage we using S3 & For block storage we have EBS volumes.**
- **And in Networking we have- VPC**
- **And for access control and management we have IAM**
- **Other than that, in the production environment for CDN service we have Cloudfront.**
- **And for monitoring Purpose there'sCloudWatch.**
- **And ECR is used as a repository for our Docker Images.**
- **And ECS used to host the docker containers.**
- **For DNS service we have Route53**

EC2

- **AMI**
- **Select instance type**
- **Key pair**
- **Select VPC and the subnet**
- **Add the security group**
- **Configure the storage to the instance by adding the volume**

Amazon Machine Images (AMI)

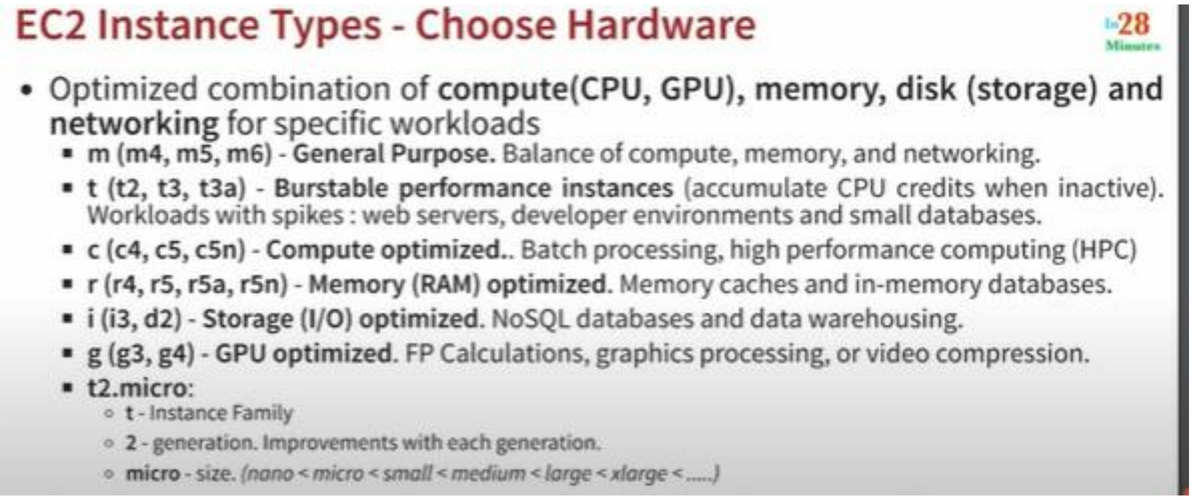
An Amazon Machine Image (AMI) provides the information required to launch an instance.

You must specify an AMI when you launch an instance.

You can launch multiple instances from a single AMI when you need multiple instances with the same configuration.

You can use different AMIs to launch instances when you need instances with different configuration

Back to first page



EC2 Instance Types - Choose Hardware

- Optimized combination of **compute(CPU, GPU), memory, disk (storage)** and **networking** for specific workloads
  - **m (m4, m5, m6) - General Purpose.** Balance of compute, memory, and networking.
  - **t (t2, t3, t3a) - Burstable performance instances** (accumulate CPU credits when inactive). Workloads with spikes : web servers, developer environments and small databases.
  - **c (c4, c5, c5n) - Compute optimized.**. Batch processing, high performance computing (HPC)
  - **r (r4, r5, r5a, r5n) - Memory (RAM) optimized.** Memory caches and in-memory databases.
  - **i (i3, d2) - Storage (I/O) optimized.** NoSQL databases and data warehousing.
  - **g (g3, g4) - GPU optimized.** FP Calculations, graphics processing, or video compression.
  - **t2.micro:**
    - **t** - Instance Family
    - **2** - generation. Improvements with each generation.
    - **micro** - size. *(nano < micro < small < medium < large < xlarge < .....)*

Back to first page.

Back to first page

UNABLE TO LOGIN TO EC2 INSTANCE

- **Verify the username for your instance**
- **Verify that your security group rules allow traffic**
- **Verify that your instance is ready.**

Security in AmazonEC2

- **Create individual IAM (Identity and Access Management) users to control access to your AWS recourses**
- **Secure the AWS Root account and its access keys.**
- **Harden EC2 instances by disabling unnecessary services and applications by installing only necessary software and tools on EC2 instances.**
- **Grant least privileges that are required to perform a specific task and not more than that. Additional permissions can be granted as required.**
- **Define and review the security group rules on a regular basis.**
- **Have a well-defined strong password policy for all the users.**

Back to first page

EBS Volumes

An Amazon EBS volume is **block-level storage device that you can attach to your instances**. After you attach a volume to an instance, you can use it as a physical hard drive.

types of EBS volumes available?

SSD-backed storage for workloads, such as databases, virtual desktops, and boot volumes, (SSD=solid state drive)

HDD-backed storage for intensive workloads, such as log processing (HDD= hard disk drive)

We attach ebs volume for multiple instance by creating new volume type in provisioned iops ssd by there is option to click multiple instance and after we can attach multiple instance in same availability zone.

S3

- **Create bucket.**

- S3 means (Simple Storage Service) is a cloud-based object storage service provided by Amazon Web Services (AWS). An S3 bucket is a container for storing objects, such as files, images, videos, and documents, within the Amazon S3 storage system.

- S3 buckets are used to store and retrieve any amount of data from anywhere on the web, at any time. They can be used to host static websites, store and share files across a distributed team, store backups of critical data, and host big data and analytics applications.

- They also provide secure access control to the stored data, by allowing you to set up permissions and policies for users and applications that access the data.

- **Bucket name.**
- **Region**
- **Allow or block public access.**
- **Bucket versioning**
- **Upload the files.**
- **Add storage class and lifecycle rules.**

S3 Bucket Permissions

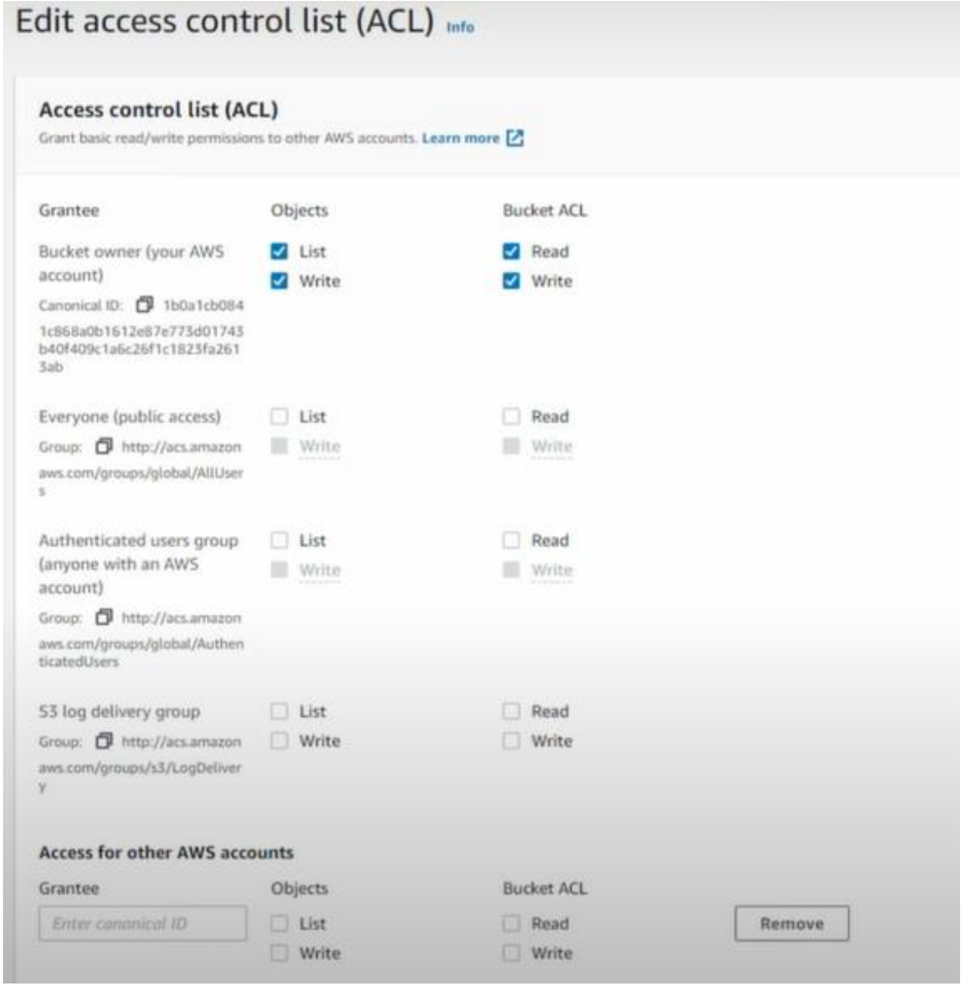
ACL (access control list)for object level permissions(it can be attached to the bucket or the objects as well),

Bucket policy for bucket level permissions**.**
IAM Policies also can be used but IAM policy can also control other services as well.

STORAGE CLASSES

- **Standard**
- **Standard IA**
- **Onezone**
- **Glacier**
- **Archive**

Back to first page



Edit access control list (ACL) Info

Access control list (ACL)

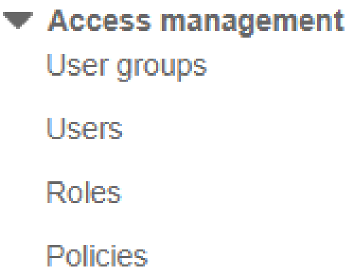
Grant basic read/write permissions to other AWS accounts. Learn more

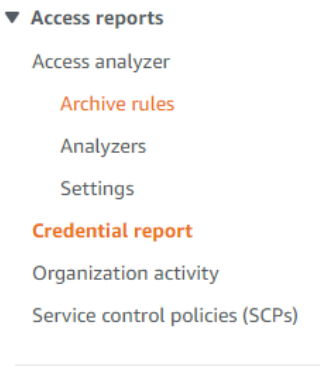
Grantee	Objects	Bucket ACL
Bucket owner (your AWS account) Canonical ID: 1b0a1cb084 1c868a0b1612e87e773d01743 b40f409c1a6c26f1c1823fa261 3ab	☑ List ☑ Write	☑ Read ☑ Write
Everyone (public access) Group: http://acs.amazon aws.com/groups/global/AllUsers	☐ List ☐ Write	☐ Read ☐ Write
Authenticated users group (anyone with an AWS account) Group: http://acs.amazon aws.com/groups/global/AuthenticatedUsers	☐ List ☐ Write	☐ Read ☐ Write
S3 log delivery group Group: http://acs.amazon aws.com/groups/s3/LogDelivery	☐ List ☐ Write	☐ Read ☐ Write

Access for other AWS accounts

Grantee	Objects	Bucket ACL	
Enter canonical ID	☐ List ☐ Write	☐ Read ☐ Write	Remove

IAM





- IAMhelps you securely control access to AWS resources & create and manages user accounts, you can add or give the limited accesses to the different users.

- mainly we have root account, rather than using the root account we created the administrator group Where all the people who have administrator access will added it.

- Similarly For examples if there are number of developers , then you can add them to the developer's group and you can directly give the required permissions to the groups rather than giving separately.

- While creating the user it will create access key and secrete key or password, by using this users can login

- Security credential details report will be available In the credential report section-
- **Back to first page**

Access to AWS

Back to first page

Roles =role has a set of permissions for making AWS service requests, and it is not associated with a specific user or group.it is similar to an IAM user. If an EC2 instance want to communicate with any other service, you can directly attach a role to it.

Giving access to the particular instance:

Create a policy+add the actions read, write or excecute+add the specific resources &instance id+add MFA if u want + add specific source ip if u want.+add name to the policy and create.

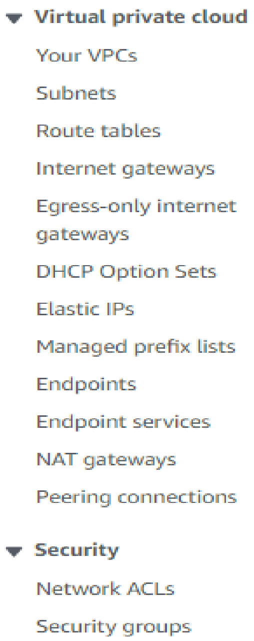
- **IAM does not require a region selection.**
- **In the IAM we are not using the root account instead we created the user with administrator access**

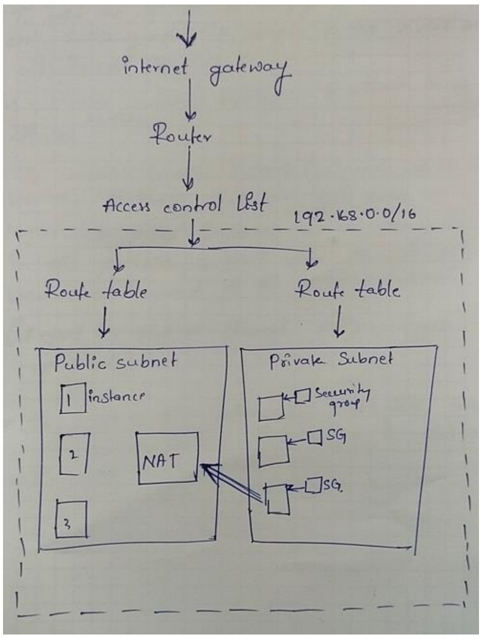
Polices:

- Policies are giving permissions to individual users, groups, or roles to control their access to AWS resources.
- Like individual resources, like S3 buckets or EC2 instances. and read-only access to a particular resource, controls, multi-factor authentication for certain operations.
- Policies can be created and managed through the AWS Management Console, .

Back to first page

VPC





Virtual private clouds (VPC)-- A VPC is a virtual network that closely resembles a traditional network that you'd operate in your own data center. After you create a VPC, you can add subnets.

Subnets---- A subnet is a range of IP addresses in your VPC. A subnet must reside in a single Availability Zone. After you add subnets, you can deploy AWS resources in our VPC.

IP addressing ---You can assign IPv4 addresses and IPv6 addresses to your VPCs and subnets. You can also bring your public IPv4 and IPv6 GUA addresses to AWS and allocate them to resources in our VPC, such as EC2 instances, NAT gateways, and Network Load Balancers.

Routing-- Use route tables to determine where network traffic from your subnet or gateway is directed.

Gateways and endpoints---- A gateway connects your VPC to another network. For example, use an internet gateway to connect your VPC to the internet. Use a VPC endpoint to connect to AWS services privately, without the use of an internet gateway or NAT device.

Peering connections---- Use a VPC peering connection to route traffic between the resources in two VPCs.

Traffic Mirroring ----Copy network traffic from network interfaces and send it to security and monitoring appliances for deep packet inspection.

Transit gateways-- Use a transit gateway, which acts as a central hub, to route traffic between your VPCs, VPN connections, and AWS Direct Connect connections.

VPC Flow Logs--- A flow log captures information about the IP traffic going to and from network interfaces in your VPC.

VPN connections--- Connect your VPCs to your on-premises networks using AWS Virtual Private Network (AWS VPN).

NAT devices

You can use a NAT device to allow instances in private subnets to connect to the internet

NAT devices are not supported for IPv6 traffic

EC2 instances communication?

* All the instances within the same VPC will communicate privately,

* if you want instance from the different vpc to communicate publicly, then Go to EC2 console and note down IP and security ID of both the linux instances, change the settings in the Security group inbound rule to allow the ICMP (all ports) to communicate.

* if you want instance from the different vpc to communicate privately , then do the VPC peering.

VPC peering?

VPC Peering is to build a private connection between your two VPCs.

It supports between different regions.

If you have VPC peering, network transfer will be in private.

If you dont want VPC peering, then communication will happen on the Public network means Instance must have Public IP to communicate

Instances in either VPC can communicate with each other as if they are within the same network.

Transit Gateway?

A transit gateway enables you to attach VPCs and VPN connections in the same Region and route traffic between them. A transit gateway works across AWS accounts

- **Security group= it acts at ec2 instance level**

  It is a firewall for ec2 instance.

  Not possible to block any ip in the security groups

Security group applies only inbound rule (traffic coming inside our network) . Will not apply outbound rule. This behavior is stateful behaviour.&also One instance can have multiple security group.

- **NACL = firewall for the subnet or vpc**
  You can block any IP in ACL, they are stateless.
  Only one ACL for one subnet.

- **If a key lost= we have a system manager service by using this service we can recover**

Networking commands

1. ifconfig
2. ip
3. traceroute
4. tracepath
5. ping
6. netstat
7. ss
8. dig
9. nslookup
10. route
11. host
12. arp
13. iwconfig
14. hostname
15. curl or wget
16. mtr
17. whois
18. ifplugstatus
19. iftop
20. tcpdum

Back to first page

ECR & ECS

- As written in the Pipeline code, Once the Image is built it is uploaded to the ECR , and that image is run as a container using the service called ECS
- For ECS -wehave environments like , staging and Production. These environments are running using EC2-instances and sometimes AWS fargate(serverless).

- Here in the ECS you have to create a task which runs our containers in the cluster.

While creating the task to run container we are mentioning the details like image url and container port numbers , etc…,

- After creating the task, you have to run the service using the tasks, which will run the containers. Same time it will load balance .

CLOUD WATCH

- **In aws to monitor the instances or services , there is service called cloud watch. Like memory matrics, disk utilization matrics , CPU utilization are monitored through it.**
- **we can configure the alarms, by using these alarms we can monitor the resources.**

  Here you can select at what intervals you have to monitor the resources. (like 1 minute , 5 mins etc.)

NAGIOS

/usr/local/nagios = here nagios is installed

Nagios is an open-source monitoring system that helps to keep an eye on the health and performance of IT infrastructure. It allows system administrators and IT professionals to monitor a wide variety of systems and services such as servers, network devices, applications, and services.

Nagios provides monitoring, alerting, and reporting functionalities that enable the early detection of problems and provide insight into the status of the monitored resources

Nagios supports a variety of notification options, including email, SMS, and phone calls, to alert system administrators and IT professionals when a problem is detected. It also provides historical reports and trending data that can be used to identify performance trends and plan capacity.

Bin etc include libexcec sbin share var

./etc/nrpe.cfg = here add the IP of the hosts which you want to moniter.

./etc/servers/hostname.cfg = here define the host & configure the services you want to moniter, and the notification interval, check attempts etc.. .

```
#systemctl restart nrpe.service
```

[Back to first page]

Auto scaling

Auto Scaling is the service automatically adjusts the number of instances in an Auto Scaling group based on user-defined policies. The policies like CPU utilization, network traffic, or any custom metrics scalling

the benefit of autoscaling is eliminates the manual interventions by creating automatically the services and resources and it has horizontal autoscaling and vertical autoscaling . horizontal means adding more servers like instances and vertical scaling group means adding more units like cpu and all

To configure Auto scalling

- Create a target group (group of instances with health check)
- Create aAuto scallingGroup–&select a launch template &attach a load balancer to the ASG (And also select the health check & select the scaling capacity)

Load balancing

Loadbancing automatically distributes the incoming traffic across the multiple target groups

add instances and ip adressess to that **target groups.**

And we have classic load balancer , application load balancer and networkload balancer
And classic load balancer routes the traffic between clients and backend servers
And application load balancer it works on l7 layours
And network load balancer it routes the to handle large amounts of traffic at the network level.
Gateway load balancer:

target group is used to route requests to one or more targets. When you create each listener rule, you specify a target group and conditions. When a rule condition is met, traffic is forwarded to the corresponding target group.

Application Load Balancer supports a **round-robin** load-balancing algorithm. Using this method, client requests are routed to available servers on a cycle. Round robin server load balancing works best when servers have identical computing capabilities and storage capacity.

A **listener**is a process that checks for connection requests

aws lambda

What is AWS Lambda?

AWS Lambda is a compute service that lets you run code without provisioning or managing servers.

Lambda runs your code only when needed and scales automatically, from a few requests per day to thousands per second.

Lambda runs your code on a high-availability compute infrastructure and performs all of the administration of the compute resources, including server and operating system maintenance, capacity provisioning
Lambda concepts
Function: A function is a resource that you can invoke to run your code in Lambda. A function has code to process the events that you pass into the function or that other AWS services send to the function.
Runtime: The runtime provides a language-specific environment that runs in an execution environment. The runtime relays invocation events, context information, and responses between Lambda and the function.
Concurrency: Concurrency is the number of requests that your function is serving at any given time.
Trigger:A trigger is a resource or configuration that invokes a Lambda function

Route 53

In AWS DNS service is called Route 53. Here you can buy a domain name and register .

Amazon Route 53 is a highly available and scalable cloud DNS service.

Its Services are

- Domain Name System (DNS),
- domain name registration

•

types of routing policies?

Simple routing policy

Failover routing policy

Geolocation routing policy

Geo proximity routing policy

Latency routing policy

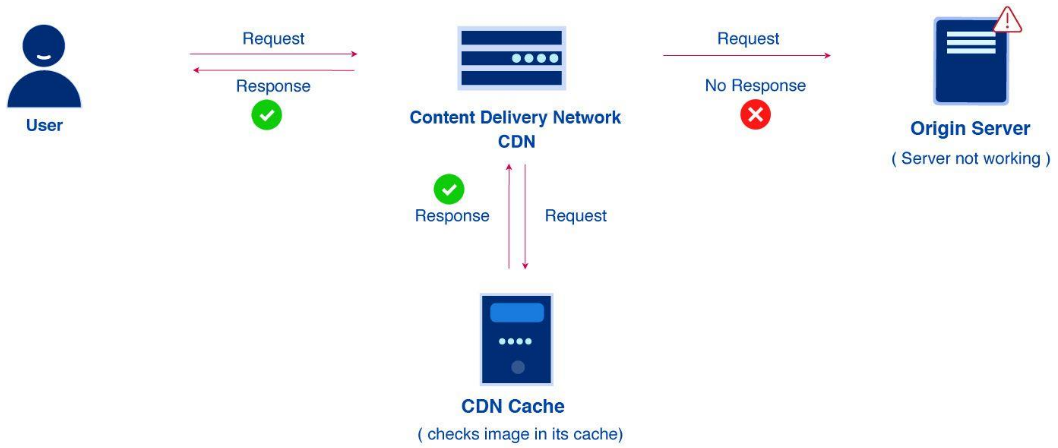
DNS Records

[Back to first page]

DNS Record	Description
A	Maps domain names to IPv4 addresses
AAAA	Maps domain names to IPv6 addresses
CNAME	Redirects a domain to a different domain
PTR	Resolves IPv4 or IPv6 addresses to domain names
NS	Provides a list of the authoritative name servers responsible for the domain
MX	Provides the domain names of mail servers that receive emails on behalf of a domain
SOA	Provides important details about a DNS zone; required for every DNS zone
TXT	Provides any type of descriptive information in text format

Content delivery network

CDN – content delivery network



A content delivery network (CDN) is a system of distributed servers that work together to deliver content, such as web pages, images, videos, and other files, to end-users. The primary purpose of a CDN is to improve the performance, scalability, and availability of websites and other online services by reducing the distance between the end-users and the content they are trying to access.

aws backup

AWS backup= ec2 instances are backed up by aws backup service, and the Volumes are backed up by taking the snapshots and its life managed by life cycle manager.

Encrypt the volume= take a snapshot + create a volume with that snapshot+ enable encryption while adding the volume+ then attach that volume to the instance.

If Ec2 reaches 90% CPU utilization = cloudwatch+createalarmas for 90%+ reboot if it reaches threshold.

How to achieve high availability in the ec2 = by running our intances across availability zones.

Back to first page

Server breakdown

6 Servers + 1 CDN

- server#1 = Load Balancer (balancing between server #1 and #2)
- server#2 = App / Web #1
- server#3 = App / Web #2 (replication of server#2)
- server#4 = Database Master (for reads and writes)
- server#5 = Database Slave (for read). This should be able to turn into the Master if the Master server goes down
- server#6 = media
- CDN#1 = pulls from server#6

- **Back to first page**

Site migration

A site migration is any event, a website undergoes major changes in areas that can significantly affect search engine— typically changes to the site's location, platform, structure, content, design

- The first step should be recording all the URLs of your current site and pages.
- Later in the migration process, you will use this information to redirect the old URLs to errors, but you will need to have all this information logged.
- Evaluate the current health of the site by running a site audit.
- Record the current traffic to your website for analyzing migration impact later.
- Track its performance against the old site’s.

aws security hub

AWS Security Hub is a fully managed security service that helps you centralize and prioritize security findings from across your AWS accounts. The following are the key components of AWS Security Hub:

1. **Security Hub Console**: This is the graphical user interface (GUI) that allows you to view and manage your security findings.
2. **Security Standards**: Security standards are a set of best practices and compliance frameworks that you can use to evaluate the security posture of your AWS environment. AWS Security Hub provides several built-in security standards, including CIS AWS Foundations Benchmark, AWS Foundational Security Best Practices, and Payment Card Industry Data Security Standard (PCI DSS).
3. **Compliance Checks**: Compliance checks are automated assessments that evaluate your environment against specific security controls required by a standard. AWS Security Hub uses AWS Config Rules to perform compliance checks.
4. **Security Findings**: Security findings are potential security issues that are identified by AWS Security Hub. These findings can come from a variety of sources, including AWS services, third-party tools, and custom integrations. Security findings are grouped into categories based on severity and type.
5. **Insights:** Insights are custom views that you can create to analyze and filter security findings. Insights allow you to create custom dashboards that show only the information you need to see.
6. **Integrations**: AWS Security Hub can integrate with other AWS services, third-party security tools, and custom integrations. Integrations allow you to bring in findings from a variety of sources and consolidate them in one place.
7. **EventBridge**: AWS Security Hub can send findings to EventBridge, which allows you to create automated workflows that respond to security events in real-time.

aws guard duty

AWS GuardDuty is a threat detection service that continuously monitors your AWS accounts and workloads for malicious activity and unauthorized behavior.

1. **Detector**: A detector is a logical container for the GuardDuty service. It represents the GuardDuty service deployment for a particular AWS account and is used to aggregate and organize the findings generated by the service.
2. **Master** Account: The Master account is the AWS account that has enabled GuardDuty and has created a detector. It is responsible for managing and configuring the GuardDuty service.
3. **Member Accounts:** Member accounts are other AWS accounts that have been invited to participate in GuardDuty by the Master account. They are monitored by the service and send findings to the Master account for aggregation and analysis.
4. **Threat Intelligence**: GuardDuty uses threat intelligence to detect potential threats to your AWS environment. The threat intelligence includes information from AWS internal sources, third-party feeds, and community-generated intelligence.
5. **CloudTrail Logs**: GuardDuty analyzes AWS CloudTrail logs to detect activity that might indicate a threat to your environment.
6. **VPC Flow Logs**: GuardDuty analyzes VPC Flow Logs to detect network traffic that might indicate a threat to your environment.
7. **IAM Access Logs**: GuardDuty analyzes IAM access logs to detect unauthorized or unusual activity related to AWS identity and access management.
8. **Findings**: GuardDuty generates findings that provide detailed information about potential threats to your environment. Findings include the type and severity of the threat, affected resources, and recommended remediation steps.

aws config

AWS Config is a fully managed service that provides you with an inventory of your AWS resources and the configuration history of those resources. It also continuously monitors and records any changes made to your resources, and alerts you when a resource is out of compliance with your desired configuration.

1. **Resources**: AWS Config supports a wide variety of AWS resources, including EC2 instances, S3 buckets, RDS instances, IAM users and roles, and many others. The service continuously monitors and records changes made to these resources, and stores this information in a configuration history.
2. **Configuration Items**: A configuration item is a record of the configuration of an AWS resource at a specific point in time. AWS Config creates a new configuration item every time a resource is created, updated, or deleted.
3. **Configuration History**: Configuration history is a record of all configuration items created for an AWS resource. The history allows you to track changes to the resource over time, and to roll back to a previous configuration if needed.
4. **Rules:** AWS Config Rules are pre-defined or custom-defined rules that are used to check for compliance with desired configurations. Rules can be used to enforce best practices, security policies, and regulatory compliance requirements.
5. **Aggregator**: AWS Config Aggregator is a service that allows you to aggregate data from multiple AWS accounts and regions into a single account and region. Aggregators allow you to monitor and evaluate the configuration of your resources across your entire organization, making it easier to identify compliance issues and security risks.

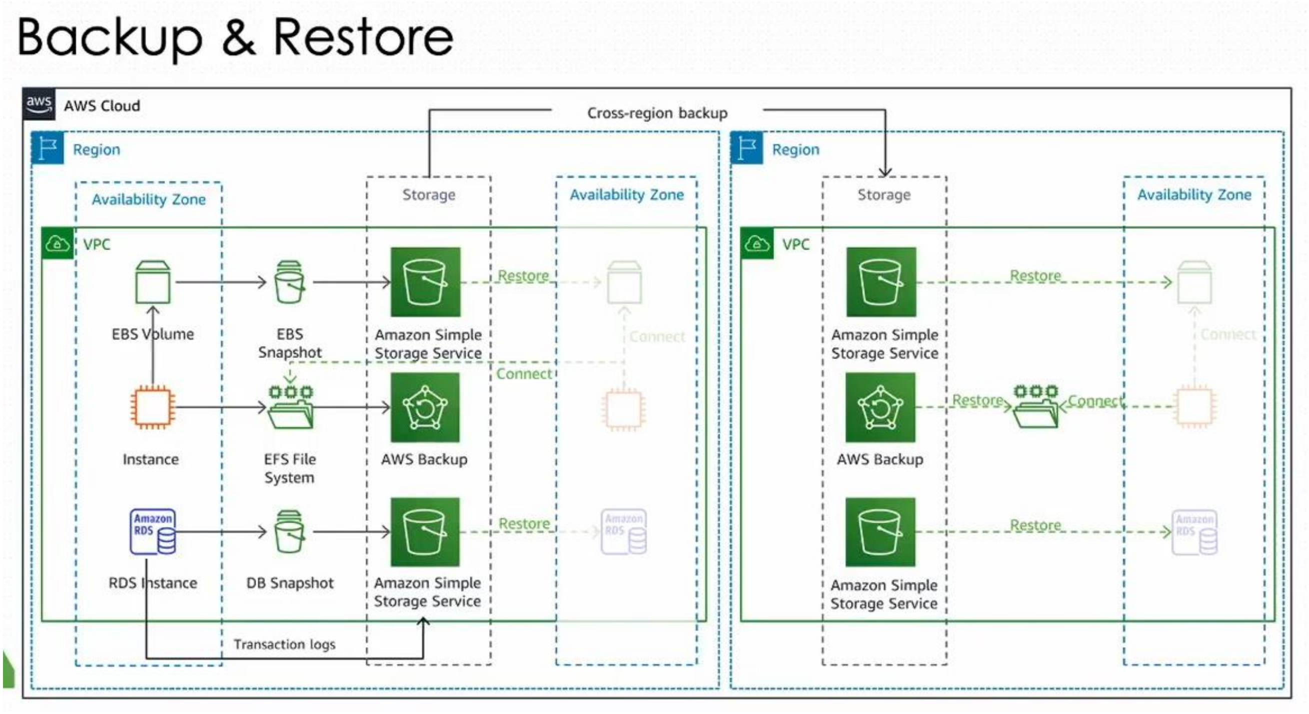
Delivery Channel: The AWS Config Delivery Channel is a configuration that enables you to specify where the configuration snapshots and change events are delivered. You can deliver them to an Amazon S3 bucket, an Amazon SNS topic, or both.
Dashboard: The AWS Config dashboard is a graphical interface that allows you to view the configuration items, configuration history, and compliance status of your resources. The dashboard provides an easy-to-use interface for tracking and managing the configuration of your AWS resources.

Aws web application firewall

web application firewall that helps protect your web applications from common web exploits that could affect application availability, compromise security, or consume excessive resources. AWS WAF has several key components, including:

Rules: AWS WAF provides a set of pre-configured rules that can be used to block common attack patterns, such as SQL injection and cross-site scripting (XSS). You can also create custom rules to match specific request patterns that are unique to your application.
Conditions: Conditions are used to specify the criteria for matching requests against your AWS WAF rules. You can use pre-defined conditions, such as IP addresses, HTTP headers, or query string parameters, or create custom conditions that match against specific request data.
Web ACLs: A Web ACL (Web Access Control List) is a collection of rules and conditions that are applied to a set of AWS resources, such as an Amazon CloudFront distribution, an Application Load Balancer, or an Amazon API Gateway API. You can create multiple Web ACLs, each with its own set of rules and conditions.
Logging: AWS WAF provides logging capabilities that allow you to capture detailed information about incoming requests, such as the source IP address, the requested resource, and whether the request was allowed or blocked. You can use these logs to perform real-time analysis or store them in Amazon S3 for long-term retention and analysis.
Rate-based rules: Rate-based rules enable you to protect against brute-force attacks, HTTP floods, and other types of volumetric attacks by limiting the number of requests that can be sent to your web application within a given time frame.
Managed rules: AWS WAF provides a set of managed rules that are maintained and updated by AWS to protect against common web exploits. You can choose to enable these managed rules on your web ACL to automatically block known threats.

Disaster recovery--



By typing lsblk –f linux machine their we get a uuid number and get that path there we can restore our Destroyed data by mounting the file system

Back to first page

HEALTH CHECK

- Check for Process Queue
- Check for Long Running Tasks with High CPU Usage
- Check for Excessive Physical Disk Input and Output
- Check/Clean the Cooling Fans, preventive maintenance

INTRODUCTION-devops

Thanks for giving opportunity, myself Prashanth having 5yrs of experience in IT domain industry .initially my carrier starts with system Engineer for 2yrs for windows and linux administration and where I need to write partial powershell and shell scripting to automate windows and linux patches updations and even in networking to configure routers ,switches and firewall are these things at initial period.

After that I moved carier called devops engineer for 3.5 yrs and cloud infrastucture as aws for different services like ec2,loadbalancer,autoscaling and vpc for networking, s3 bucket for storaging and Dns for route53 ,iam for control access management for different polices and permissions, lambda, and even cloud watch.

And i have intermediate level of terraform scripting to automate all these things.

And I have good experience in writing cicd pipeline scripts by using shared libraries and .we have integrating code from github to Jenkins through webhook integration for different different branching strategies and setting up Jenkins master slave architecture from scratch to distribute the work activity and most of our applications are java based applications we build the artifacts from maven repository and the generated artifacts were pushing to jfrog private repository from the jenkins server.

And for Code Quality & unit testing ,code coverage, Static Analysis i have used SonarQube So installing executable build Servers & passing environmental variables for unit testing, code coverage, passing it in the Sonarqube server, Generating the Reports and maintaining quality gates & publishing in to the SonarQube server dash board.

And Automation i have used Ansible for automating Build Configration Servers & Configration of differunt. different environment like stadging server. developing server & production server to automate through ansible .

And I have containlized many builds using Docker image. i have Created customized image to run the Docker container and Running builds inside that container.

And for orchestrstion we need to create a kubernetes cluster to deploy our application .and right know the deployment this happening on primeise servers and recently iam done poc es on ssl termination on nginx controller and ha of postgreql of patroni for statefullset .when fails the postgresql pod and setting up cpu pinned and cpu non pinned inside the pods and exploring argocd for Kubernetes cluster deployment.

These are my overall experience..

The role of a Linux administrator, also known as a system administrator or sysadmin, involves managing and maintaining Linux-based systems and networks. Their responsibilities can vary depending on the organization and the specific job requirements, but here are some common roles and responsibilities of a Linux administrator:

1. System Installation and Configuration: Linux administrators are responsible for installing, configuring, and maintaining Linux operating systems on servers, workstations, and other hardware platforms.
2. User and Group Management: They manage user accounts, permissions, and access control, ensuring that users have the appropriate privileges and restrictions.
3. System Monitoring and Performance Tuning: Linux administrators monitor system performance, identify bottlenecks, and optimize system resources such as CPU, memory, disk, and network usage. They also configure and analyze system logs and performance metrics.
4. System Security: Administrators implement and maintain security measures to protect Linux systems from unauthorized access, such as configuring firewalls, implementing encryption, and applying security patches and updates. They also conduct regular security audits and vulnerability assessments.
5. Backup and Recovery: They develop and implement backup strategies to ensure data integrity and system availability. Linux administrators perform regular backups and test the restoration process to ensure data can be recovered in case of system failures or data loss.
6. Package Management: Administrators use package managers like apt, yum, or dnf to install, upgrade, and remove software packages on Linux systems. They also manage software repositories and ensure that systems are up to date with the latest security patches and software updates.
7. Networking and DNS Configuration: Linux administrators configure and troubleshoot network interfaces, set up IP addressing, manage network services, and configure DNS (Domain Name System) settings to enable network connectivity and name resolution.
8. Scripting and Automation: They utilize scripting languages like Bash, Python, or Perl to automate routine tasks, create system monitoring scripts, and develop custom tools to streamline administrative processes.
9. Troubleshooting and Problem Resolution: Administrators diagnose and resolve system issues, including hardware and software failures, network connectivity problems, and performance bottlenecks. They use various tools and techniques to troubleshoot and fix problems promptly.
10. Documentation and Reporting: Linux administrators maintain documentation, including system configurations, procedures, and troubleshooting guides. They provide reports on system performance, security incidents, and maintenance activities.

It's important to note that the roles and responsibilities of Linux administrators can vary across organizations and may include additional tasks depending on the specific needs of the environment they are working in.

Devopsroles and responsibilities,

And my roles and responsibalities are My daily task starts with taking handover from the reliever, checking email notification of jira tickets

whatever the tasks or the tickets resolved or the pending tickets in the shifts are discussed in the handover meetings.

If any issues with the Machine server , Nagios or grafana will capture the issue and it will send the alerts.

Once the alerts come into our bucket, we try to resolve the issues, if not possible to resolve then I will forward to the L2 or L3 level of management team.

Other than that, providing support for developers and where issues in production servers and automation tasks and creating Jenkins jobs for deployment and deploying code on dev and production servers and checking the proper approvals before code release from development team QA team and user assurance test team and techteam for production process

. AND my Responsibalities are build and release process during code commits from any branch and creating and maintaining user accounts, giving access to the files, task scheduling the backup activities and manageging git branching activity and mainatance of cicd automation from Jenkins and generating project code quality html report by pushing the project code to sonarqube monitoring tool all these my responsibalities.

Roles and responsibalities.

Coming to my roles and responsibalities my day starts with checking mails and standup meeting as a Scrum Master , I manage a team of individuals, breaking down their stories and creating proper Jiras tickets for assigned tasks on a sprint basis, which typically lasts for two weeks.

I also contribute as an individual contributor, working to improve our established pipeline. Our team has already categorized alerts based on infrastructure and service level, and we analyze tickets before assigning them to the product team.

And also we assist the development team with their automation tool-related queries and maintaining the CI/CD pipeline to enhance their productivity. Setting up new pipelines or modifying existing ones to accommodate new environments are also part of our responsibilities.

I proactively to take the initiative to introduce new ways to enhance our pipeline. Recently, I suggested integrating SonarQube to improve code quality and detect security issues. like the challenges we face, our team is always striving to optimize our processes and deliver the best results possible.

1. **Challenges facing recent in my organisms:**

   In my organization, we used to build applications on a Jenkins server, but we encountered issues due to different applications having different requirements. For instance, installing dependencies for one application caused conflicts with other applications. So To resolve this issue, I recommended using a Docker-based virtual environment.

   In this approach, we use a base Docker image and install dependencies inside it. We then compile the code and create an image from it. This approach has helped us resolve many issues that were previously causing conflicts between applications.

Back to first page

Recently I am closely working with the Devops team ,

- So I have a experience in creating the CI/CD Pipeline Jobs, and also look in to releases.
- for newly onboarding projects they assign me a tasks like doing activities like- creating repository, giving access to the users and creating jobs in the Jenkins, integrating the different softwares
- Already existing project, I am supporting any build related issues or any access related issues. Aws administration. Related to EC2 instances and S3 buckets& experience in VPC as well.

Several deployments,

ChatGPT

In the context of software development and deployment, "several deployments" typically refers to the practice of deploying multiple instances or versions of an application or service. This approach is often used to support various use cases, such as testing, staging, production, or different environments (e.g., development, QA, production).

Here are a few scenarios where multiple deployments may be employed:

1. Development environments: In a development workflow, multiple deployments can be created to support individual developers or teams working on different features or bug fixes concurrently. Each deployment provides a dedicated environment where changes can be tested and validated independently.
2. Testing and staging environments: Multiple deployments can be used to create separate environments for testing new features or changes before promoting them to production. This allows for comprehensive testing and quality assurance without impacting the production environment.
3. Blue-Green or Canary deployments: Blue-Green and Canary deployment strategies involve creating multiple deployments to minimize risk during production releases. In Blue-Green deployments, two complete environments (blue and green) are set up, with one serving production traffic while the other is updated. Once the new deployment is verified, traffic is switched from the blue to the green environment. Canary deployments involve deploying a new version to a subset of users or servers to gather feedback and ensure stability before rolling out to the entire user base.
4. High availability and load balancing: Multiple deployments can be used to create redundant instances of an application to ensure high availability and fault tolerance. Load balancers can distribute traffic across these deployments, enabling horizontal scaling and improving performance.
5. Geographically distributed deployments: For applications that serve users in different regions, deploying multiple instances in geographically distributed locations can help reduce latency and improve user experience by serving requests from the closest deployment.

When managing multiple deployments, it is important to have proper orchestration, monitoring, and automation tools in place. These tools can assist in maintaining consistency, managing updates, and monitoring the health and performance of each deployment. Infrastructure as code approaches, such as using configuration management tools or container orchestration platforms like Kubernetes, can help streamline the management of multiple deployments and ensure consistency across environments.

Developing staging and production,

In Development phase, the Developers write a Code and make a minimal testing in local machine, they mainly use Version Control system to the manage the Code changes, finally team members are reviewed Code and merging the code into development branch to staging branch. like After that, stadging branch is nothing but pre-production environment. Realistic data used for testing like functional testting, integration testing, performance testing, security testing to identity the bugs and fixing the issues and then finally Software gets stable and meets the Require quality standards and then finally the code merging into the stadging branch to produ ction branch. In production environment, the Software is deployed for the end-uers and here we me monitoring tools to check the system health and performance and here mainly we implemented High-availabality ,Security and disaster recourey, these environments we implemented for software development life cycle.

My job is to create CI-CD pipeline from the Scratch we will, Connect to the product team and will the ask the how the build flow works to generate the artifact. suppose windows application are MSI (@r) ExE formate and for Linux application are jar. war file. and i installed the jenkins server and downloading the various plugins Required for the software and Creating the username and passwords in the manage credentials and we create CIcD multi branch pipeline for CI/CD for that build. In Every Repository is to write Jenkins file by stage by stage and every feature branch creates a jobs automatically by webhook contigration. and for unit testing, code coverage, Static analysis will publish in to the Sonar cube Server dashboard, if thresold crietria not meets means, the build build gets unstable. It shows in yellow in colour. after build gets stable by making some changes and then we generated artifact and pushing into the frog Repository. and writing customised Docker image and deployed into the Kubernetes duster.

developing environment in devops

ChatGPT

In DevOps, the development environment refers to the infrastructure and tools that enable software development, testing, and deployment. It aims to provide a consistent and controlled environment for developers to write code, collaborate, and validate their changes before deploying them to production.

Here are some key aspects and considerations for developing environments in DevOps:

1. Version Control System (VCS): A VCS such as Git is essential for managing source code, enabling collaboration, and tracking changes made by developers.
2. Continuous Integration (CI) Server: A CI server, like Jenkins or GitLab CI/CD, automatically builds and tests code changes from multiple developers in a shared environment. It helps catch integration issues early and promotes a consistent build process.
3. Development Environments: Developers need isolated environments to work on their code changes. This can be achieved through local development setups (e.g., local installations of development tools) or remote development environments (e.g., virtual machines or containers).
4. Infrastructure as Code (IaC): By using tools like Terraform or CloudFormation, infrastructure components such as servers, networking, and databases can be defined as code. This allows for versioning, automation, and consistent provisioning of environments.
5. Configuration Management: Tools like Ansible or Puppet help manage the configurations of various components in the development environment. They ensure consistency across environments and automate the setup and maintenance of required software and dependencies.
6. Testing Environments: DevOps promotes the use of dedicated testing environments that replicate the production environment as closely as possible. This includes similar hardware, software, and configurations. Automated testing frameworks, such as Selenium for web applications or JUnit for unit testing, help ensure the quality of code changes.
7. Continuous Delivery (CD) Pipelines: CD pipelines define the stages and workflows that code changes go through before reaching production. These pipelines typically include steps for building, testing, and deploying applications, as well as performing additional validation and approval processes.
8. Containerization and Orchestration: Containerization platforms like Docker enable developers to package applications and their dependencies into portable containers. Orchestration tools like Kubernetes provide a way to manage and scale containerized applications in a production-like environment.
9. Monitoring and Logging: Proper monitoring and logging solutions are essential for tracking the performance and health of applications in the development environment. Tools like Prometheus, Grafana, or ELK (Elasticsearch, Logstash, Kibana) stack can be used to collect, analyze, and visualize relevant metrics and logs.
10. Collaboration and Communication: DevOps emphasizes collaboration and effective communication between team members. Tools like Slack, Microsoft Teams, or Jira facilitate real-time communication, issue tracking, and project management.

These are some common elements of a development environment in DevOps. However, it's important to note that the specific tools and technologies used may vary depending on the organization, project requirements, and individual preferences.

staging environment in devops

ChatGPT

In DevOps, a staging environment is a pre-production environment that closely resembles the production environment. It serves as a final testing ground for software releases before they are deployed to the live production environment. The staging environment helps ensure that the software works correctly and performs well in an environment that closely mirrors the production environment.

Here are some key aspects and considerations for staging environments in DevOps:

1. Purpose: The staging environment is primarily used for final testing, validation, and quality assurance of software releases. It helps identify any issues or bugs that may have been missed during development and testing in lower environments.
2. Environment Setup: The staging environment should be designed to closely replicate the production environment, including similar infrastructure, network configurations, security measures, and dependencies. This allows for accurate testing and validation of software changes.
3. Data Management: In order to simulate real-world scenarios, the staging environment should ideally have a copy of production data or representative datasets. This helps ensure that the software performs correctly with actual data volumes and helps identify any data-related issues.
4. Version Control and Deployment: Similar to other environments in the DevOps pipeline, the staging environment should be version controlled to ensure traceability and repeatability. Automated deployment tools, such as CI/CD pipelines, can be used to deploy software releases to the staging environment.
5. Integration Testing: Staging environments are typically used for integration testing, where various components and services are tested together to ensure they work seamlessly. This includes testing interactions between different software modules, third-party integrations, and APIs.
6. Performance and Load Testing: The staging environment is a suitable place to perform performance and load testing on the software. This helps evaluate the system's behavior under normal and peak usage scenarios, ensuring it can handle the expected load in the production environment.
7. User Acceptance Testing (UAT): Staging environments are often used for UAT, where stakeholders, including end-users or product owners, validate the software's functionality, usability, and adherence to requirements. This feedback can help catch any issues or discrepancies before deploying to production.
8. Environment Isolation: The staging environment should be isolated from the production environment to prevent any accidental impact on live systems. This can be achieved through separate infrastructure, networks, and security measures.
9. Rollback and Recovery: In case of issues or unexpected behavior in the staging environment, it should be possible to roll back the software release or recover the environment to a known good state. This ensures that the production environment remains unaffected by any issues identified during staging.
10. Monitoring and Validation: Monitoring tools should be in place to track the performance, health, and behavior of the staging environment. This allows for the identification of any anomalies or issues during testing and validation.

By having a dedicated staging environment in the DevOps pipeline, organizations can gain confidence in the software releases and minimize the risk of introducing bugs or issues in the production environment. It provides a controlled and representative environment for final testing, validation, and stakeholder approval before deploying to production.

production environment

ChatGPT

The production environment in DevOps refers to the live, operational environment where the software or application is deployed and made available to end-users or customers. It is the final destination for the software after it has gone through development, testing, and staging.

Here are some key aspects and considerations for the production environment in DevOps:

1. Stability and Performance: The production environment should be stable and highly reliable to ensure the smooth operation of the software. It should be optimized for performance to handle the expected workload and user traffic.
2. Scalability and Availability: The production environment should be designed to scale horizontally or vertically as needed to accommodate increasing demands and to ensure high availability. This includes load balancing, auto-scaling, and fault-tolerant configurations.
3. Security: Security measures are crucial in the production environment to protect sensitive data and prevent unauthorized access. This includes network security, encryption, access controls, and regular security audits.
4. Continuous Deployment: DevOps aims to automate the deployment process to the production environment. Continuous Deployment (CD) pipelines, triggered by successful testing and validation in lower environments, can automatically deploy the software to production while ensuring minimal downtime and disruption.
5. Monitoring and Alerting: Monitoring tools and practices should be in place to track the health, performance, and usage of the production environment. This includes monitoring key metrics, logs, and error reporting to identify

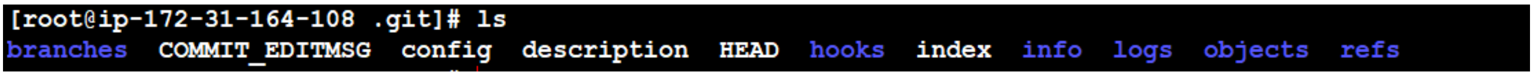
any issues or anomalies. Alerts should be set up to notify the appropriate teams in case of critical events or failures.

6. Configuration Management: The configuration of the production environment should be managed carefully to ensure consistency and avoid configuration drift. Infrastructure as Code (IaC) tools can be used to define and manage the environment's configuration, making it reproducible and version-controlled.
7. Backup and Disaster Recovery: Regular backups of production data and configurations are essential to minimize the risk of data loss or system failures. Disaster recovery plans should be in place to ensure timely recovery in case of major incidents or disasters.
8. Incident Management: Processes and tools for incident management should be established to handle and respond to incidents or outages in the production environment. This includes incident reporting, communication channels, root cause analysis, and incident resolution workflows.
9. Change Management: Changes to the production environment should be carefully planned, documented, and tested before deployment. Change management practices, such as change approval processes and change rollback mechanisms, help minimize the impact of changes and ensure a stable production environment.
10. Continuous Improvement: DevOps promotes a culture of continuous improvement, even in the production environment. Feedback from users, monitoring data, and incident analysis should be used to identify areas of improvement, implement optimizations, and enhance the overall reliability and performance of the software.

The production environment is the ultimate destination for the software in the DevOps lifecycle. It is where the software serves its intended purpose and delivers value to end-users or customers. Therefore, ensuring the stability, performance, security, and reliability of the production environment is critical for the success of the software and the organization.

GIT

Version = 2.39



```
[root@ip-172-31-164-108 .git]# ls
branches  COMMIT_EDITMSG  config  description  HEAD  hooks  index  info  logs  objects  refs
```

-------Branching strategy:

Currently we are using three branches   develop branch and staging branch and main branch or relase branch. For developer branch all the developers are connecting and they are committing the code to the branch. As soon as developer commit the code into the development branch, Jenkins is automatically triggered because we configured webhook trigger. Then it will run unit test cases. Once the unit test passes , it will build. Using maven tool. By Using pom.xml file. then it will create artifact war file for linux application or exe or msi for windows application .

Once developer says it is working fine, then create merge request to merge staging branch and team members reviews it by doing different variety of functionality testing …if it ok with the code and then create merge request to review product team and archetects and merge the staging branch to master or relase branch. We configured master branch with the pre prod environment and production environment .. these are my branching stragies   .

Some of the other branches used in other projects are like

- Feature branch= this branch used by developer to develop the new feature in the existing application, junior developers write the code in the feature branch to review these codes by the senior developers this feature branch is merged with the development branch.
- Development branch= used to deploy to the dev environment.
- Release branch= release branch used for deploy the application code to the test environment.

- Master branch= to deploy the code to the production environment.
- Hotfix branch= if any errors in the production branch those will be solved in the hotfix branch

Back to first page

-------stages in Git:

- **Working area**= once if you create or update it is in the working area
- **Staging area**= to move from working area to staging area you have to pass the command #git add /.
- **Local repository**= to move to the local area you have to pass the command #git commit -m “message / comment”

--------Commands

Install the git in your Linux machine #yum install git it configure in - `/usr/bin/git  it uses the c -language`

Git is distributed version control system where everyone pull and push the code remote repository and also it tracks the codes ..

And .git-contains head,config,description,hooks,objects and refrences.

- **git init** = it will initialize and creates an empty git repository
- **git clone=** downloaging complete file from remote repository to local repository of particular branch.
- **git add filename** = it will add the particular file to the staging area
- **git status** = it will give the status of the file which needs to be committed to the local repo.
- **git commit -m “message / comment”** = files are committed to the local git repo
- **git pull –** it downloads the lattest changes in the remote repository.
- **git branch -M main**=final all changes and test the code in develop branch and merge with master or main branch
- **git remote add origin “link copied from git repo”** = it ll add remote repo to local repo
- **git remote -v** = it ll check which origin is configured.

- **git pull origin master** = it ll pull the all the files from remote repo from latest changes
- **git push -u remotename branchname** = push the files to the one local remote repo to another local repo
- **git log=** will give the logs
- **git branch branchname** = creates the new branch
- **git branch** = shows the available branches and which branch you are currently in
- **git checkout branchname** = to enter the other branch

- **git cherry-pick -x <commit-hash>=**Cherry picking in Git means to choose a commit from one branch and apply it onto another if suppose 50 commits in develop branch out of which only 5 commits need to push in relase branch.
- **git reset=**suppose I moved any 2 files from working area to staging area, but I don’t want to commit both the files , I want one file back to the working area that time we use git reset .

  in the git reset hard , whatever the commit we have made that reverted also it deletes the commit ID as well

- **git revert=** in case I have wrongly commited the code and I want to revert that commit . for that get the commit ID (#git log) and use the command #git revert commit id
- **git rebase –** if suppose developer commits more commits in locally he has to push single commit to develop branch (# git rebase -iHEAD~1
- **git stash-** it stores the uncommiting changes suppose if developer stops the unfinished work without commiting and later he has to work with that file soon that time he can use git stash applyand git stash pop and git stash drop .
- **git squash-** merge all comits into new commit when merging branch or pushing repo.
- **git push origin -f branchname =** delete the commits in the remote repo
- **git clone -b local https://gitrepo =** it downloads the all the files in the local repository.

- **merge conflict-** when developers commits same file with different content in the lines of code so there merge conflicts exits on that time go to repository click the pull request at the bottom click resolve conflicts and concern developer make final changes and push the repository.
- **Git checkout-** b- create the new branch and shifting one branch to other branch and –d branch= delete merged branch
- **git diff-** it shows the difference between commits.
- **Git diff comitid1 commitid2-** it show changes between two commits.
- **git reflog-** tool to keep track of changes made to branch tips or If you want to fetch the log references of a particular branch
- **git log-** it shows the commit history with repective ids – **git log --oneline**
- `git branch --merged` helps to get the list of the branches that have been merged into the current branch.
- Note: `git branch --no-merged`
- **`git config -global user.name “Your Name”`:** This command will add a username.
  `git config -global user.email “Your E-mail Address”`: This command will add an email id.

1. Explore the gitops?
   GitOps is designed to make the process of deploying and managing infrastructure and applications more efficient and reliable. By using Git as the source of truth, teams can collaborate more effectively and avoid manual errors that can lead to downtime or other issues. Additionally, GitOps provides a clear audit trail of all changes, making it easier to troubleshoot issues or roll back changes if necessary.

1. FluxCD: A Kubernetes-native GitOps tool that automates the deployment and configuration of Kubernetes applications.
2. ArgoCD: Another Kubernetes-native GitOps tool that provides continuous delivery for Kubernetes applications.
3. Jenkins X: A GitOps tool that automates the continuous delivery of cloud-native applications.
4. GitLab: A GitOps tool that provides a complete DevOps platform, including source code management, continuous integration, and continuous delivery.

1.After building a latest image like testing , building, and creating a customized docker image and pushing into docker hub through jenkins integration.

2.maintained separate repo in git contain latest image of Kubernetes manifest files and installing and configuring argocd in Kubernetes cluster.

3. Add the repository as a source in ArgoCD: In the ArgoCD UI, provide the details of the Git repository you set up in the previous step. ArgoCD will then verify the repository and add it as a source.

4.Create an Application and provide the details of your Kubernetes application, including the Git repository, branch, and path where your Kubernetes manifests are located.

5.Configure the synchronization options for your application, such as how often ArgoCD should sync your Kubernetes resources with your Git repository.

6.Deploy your application: ArgoCD will now automatically deploy your Kubernetes application based on the manifests in your Git repository. You can monitor the progress of the deployment in the ArgoCD UI.

finally whenever, whenever we add a code in git repository it automatically build latest image and update image in manifest files in kubernetes and argocd syncs and deployed in kubernetes cluster after that end users can access new feature

Main difference between github and gitlab?

GitHub is built on top of Git, an open-source distributed version control system. It allows developers to easily share code with others and collaborate on projects, whether they are working on the same team or located in different parts of the world.

> GitLab is a web-based Git repository management tool that provides version control, continuous integration and deployment (CI/CD), and collaboration features. Like GitHub, GitLab is built on top of Git, an open-source distributed version control system.
>
> GitLab provides both a free and open-source Community Edition (CE), as well as a paid Enterprise Edition (EE) with additional features and support.

GitHub and GitLab are both web-based platforms that offer Git repository management, version control, and collaboration tools.

1. Hosting: GitHub is a cloud-based platform that provides hosting for Git repositories, while GitLab can be self-hosted on-premises or in the cloud. GitLab offers both a cloud-based version (GitLab.com) and a self-hosted version (GitLab CE/EE).
2. Pricing: GitHub offers a freemium pricing model, where users can create unlimited public repositories for free, but need to pay for private repositories or additional features. GitLab, on the other hand, offers both a free and open-source Community Edition (CE), as well as a paid Enterprise Edition (EE) with additional features and support.
3. Features: GitLab has a broader range of features than GitHub, including built-in continuous integration and delivery (CI/CD), project management tools, issue tracking, and code review tools. GitLab also provides more granular permissions and access control options than GitHub.
4. Integration: GitHub has a larger ecosystem of third-party integrations, including popular tools like Slack, Jira, and Trello. GitLab also has a range of integrations, but the selection is not as extensive as GitHub.
5. Community: GitHub has a larger community of users and contributors, which can make it easier to find resources and support. GitLab has a growing community, but it is not as large as GitHub's.

In summary, GitHub and GitLab are both popular Git repository management platforms that offer similar features, but with some differences in hosting, pricing, features, integration, and community. Choosing between the two platforms will depend on the specific needs and preferences of the organization or individual user.

MAVEN

Maven version = 3.8.6

Back to first page

Build tools automate the build process. Default path -/opt/maven

------Maven phase

- Maven is a build tool for java based application .
- It has 3 lifecycle- 1. Default (validate, all above below points) 2. Clean (preclean, clean,postclean) 3. Site(preside,site,postsite,sitedeploy)
- Validate=validate the code is correct and all necessary information available.
- compile-=compile the source code (convert human readable to machine readable)
- test-= Unit Test (these are the tests written by developer to test their own code.

- package-= package the code in its distributable format like .war .jar
- verify-= ensure the quality criteria met like intrgration testing developer write a code to test it
- install- install the package into the local repository
- deploy= done in the build environment , copies the final package to the remote repository for sharing with other developers and projects.
  #mvn install = creates target directory and .war file inside of the target directory.
  #mvn clean= will remove the target directory.

Pom.xml file?(project object model)

It is basically Build file for Java applications, whenever we building a war file, maven first check for the Pom.xml in the source code. It contains **information about the project and configuration details used by Maven to build the project**. Like **plugins, modelVersion, groupId, artifactId, version and it contains the base directory**

.Ear- java enterprise based application. **.war-** java web based application. **.jar-**class files files, audio files, image files

`-mvn install -DskipTests=false= it skips the tests in running project`.

Types of maven repository:

1. local- .m2 directory
2. central- shared repository
3. remote- developers manage repositories.

Maven archetecture

In the Apache platform there's a central repository that stores and distributes all available artifacts, both the dependencies and the plugins for Maven. You can configure your own remote repositories, but in a nutshell, all that sits in the cloud.

The local repository typically resides in the *~/.m2* directory. This directory also stores the Maven configuration in the form of the *settings.xml* file. And this can use it to configure the remote repositories, credentials to access them,

All Maven configuration comes from the pom.xml files, which declare the model for the project. They contain the configuration of the build, declaration of the dependencies and which plugins to use.

Plugins functionality is divided into **goals**. A goal is a build step, if you like. The goals are registered to be executed during the phases of the build. The goals perform the actual work, during the phases that come from the lifecycle of the project.

Logs: we can check logs from build job console output wheather it failed or pass to debug any errors or we can check mvn clean install redirects into build.log .we can check that file in target directory.

Maven logs and debug

Debug errors:

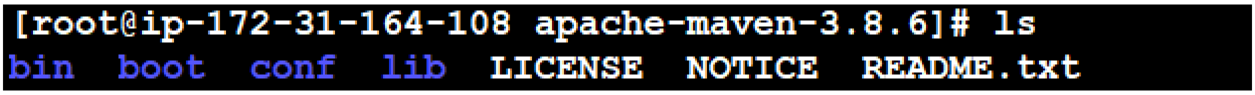
1. Check the Maven output: it will usually display an error message in the console. Start by carefully reading the error message to understand the problem.
2. Check the Maven logs: If the error message is not clear or does not provide enough information, you can check the Maven logs for more details. Maven logs are typically located in the target/logs directory of your project.
3. Verify your POM configuration: Check your POM file to ensure that all dependencies and plugins are correctly configured. Make sure that the versions of the dependencies and plugins are compatible with each other.
4. Check your project structure: Ensure that your project has the correct directory structure and that all the necessary files are in the right place.
5. Running Maven with the debug flag (-X) will provide more verbose output, which can help identify the source of the error.

Try cleaning the project: Sometimes, a project can become corrupted, which can result in errors. Try running the clean command to remove any generated files and rebuild the project

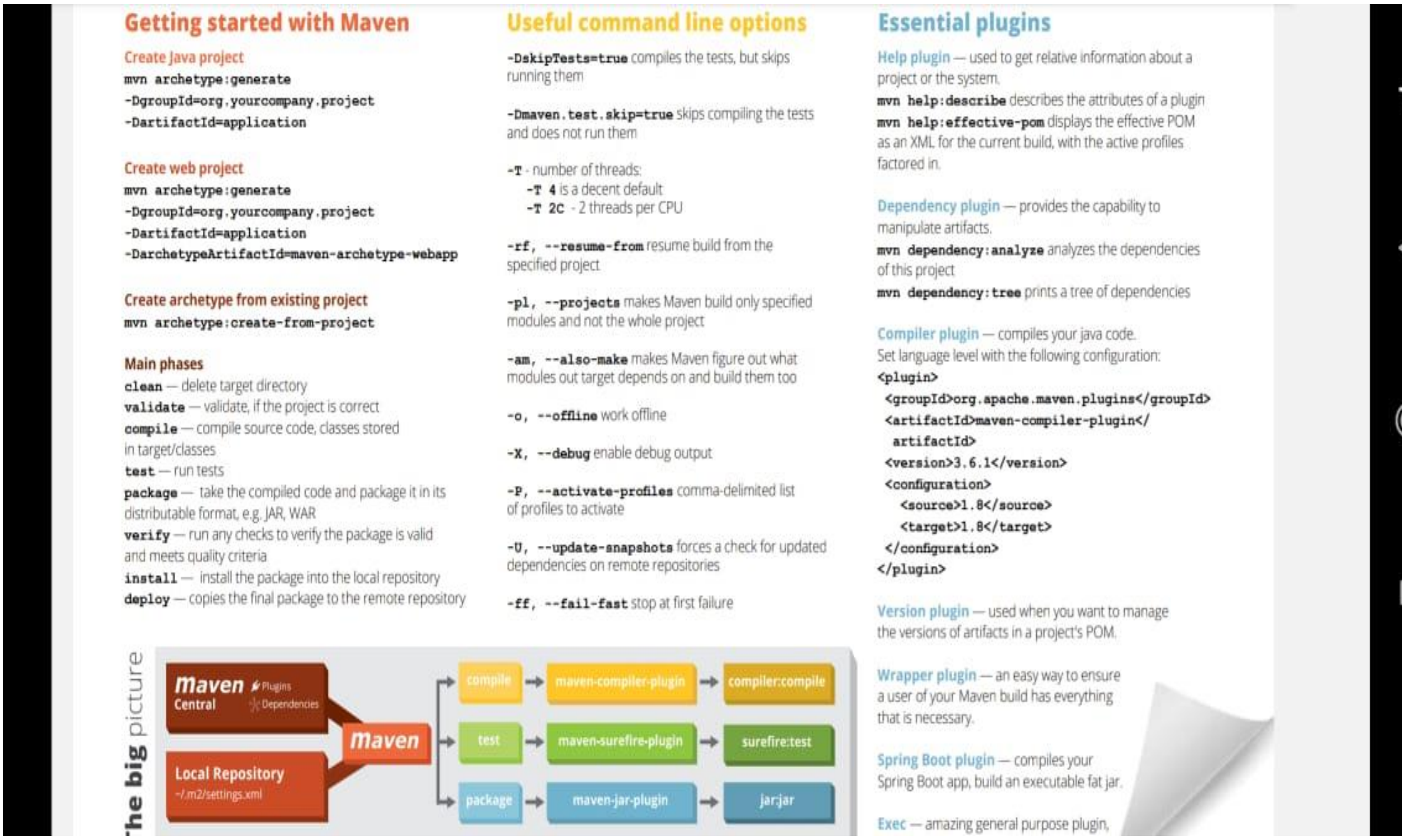
42. What is the difference between the maven package and the maven install?

package: converts the compiled code into a distributable format, such as a JAR.

install: adds the package to the local repository, allowing it to be used as a dependency in other projects.



```
[root@ip-172-31-164-108 apache-maven-3.8.6]# ls
bin  boot  conf  lib  LICENSE  NOTICE  README.txt
```



Getting started with Maven

Create Java project

```
mvn archetype:generate
-DgroupId=org.yourcompany.project
-DartifactId=application
```

Create web project

```
mvn archetype:generate
-DgroupId=org.yourcompany.project
-DartifactId=application
-DarchetypeArtifactId=maven-archetype-webapp
```

Create archetype from existing project

```
mvn archetype:create-from-project
```

Main phases

`clean` — delete target directory
`validate` — validate, if the project is correct
`compile` — compile source code, classes stored in target/classes
`test` — run tests
`package` — take the compiled code and package it in its distributable format, e.g. JAR, WAR
`verify` — run any checks to verify the package is valid and meets quality criteria
`install` — install the package into the local repository
`deploy` — copies the final package to the remote repository

Useful command line options

`-DskipTests=true` compiles the tests, but skips running them

`-Dmaven.test.skip=true` skips compiling the tests and does not run them

`-T` - number of threads:
  `-T 4` is a decent default
  `-T 2C` - 2 threads per CPU

`-rf, --resume-from` resume build from the specified project

`-pl, --projects` makes Maven build only specified modules and not the whole project

`-am, --also-make` makes Maven figure out what modules out target depends on and build them too

`-o, --offline` work offline

`-X, --debug` enable debug output

`-P, --activate-profiles` comma-delimited list of profiles to activate

`-U, --update-snapshots` forces a check for updated dependencies on remote repositories

`-ff, --fail-fast` stop at first failure

Essential plugins

Help plugin — used to get relative information about a project or the system.
`mvn help:describe` describes the attributes of a plugin
`mvn help:effective-pom` displays the effective POM as an XML for the current build, with the active profiles factored in.

Dependency plugin — provides the capability to manipulate artifacts.
`mvn dependency:analyze` analyzes the dependencies of this project
`mvn dependency:tree` prints a tree of dependencies

Compiler plugin — compiles your java code.
Set language level with the following configuration:

```
<plugin>
 <groupId>org.apache.maven.plugins</groupId>
 <artifactId>maven-compiler-plugin</
  artifactId>
 <version>3.6.1</version>
 <configuration>
   <source>1.8</source>
   <target>1.8</target>
 </configuration>
</plugin>
```

Version plugin — used when you want to manage the versions of artifacts in a project's POM.

Wrapper plugin — an easy way to ensure a user of your Maven build has everything that is necessary.

Spring Boot plugin — compiles your Spring Boot app, build an executable fat jar.

Exec — amazing general purpose plugin,

various build tools

ChatGPT

There are several build tools available for software development, each with its own strengths and weaknesses. Here are a few examples:

1. Maven: Maven is a popular build automation tool for Java projects. It uses an pom XML configuration file to define project dependencies, build targets, and plugins.Maven also supports automatic dependency resolution and can generate reports on code quality and test coverage.
2. Gradle: Gradle is a build automation tool that is designed to be flexible and scalable. It uses a Groovy-based DSL to define build configurations and supports incremental builds, parallel builds, and remote builds. Gradle is used by many Android developers and is gaining popularity in the Java community.

Commands: gradle build,run,test,clean. The `settings.gradle` file is used to configure the project's settings, such as the project name and version, while the `gradle.properties` file is used to define system properties and other configuration options.

3. Ant: Ant is a build automation tool that is widely used in the Java community. It uses an XML configuration file to define build targets and dependencies, and supports a wide range of plugins for tasks such as compilation, testing, and deployment.
4. Make: Make is a popular build automation tool for Unix-based systems. It uses a Makefile to define build rules and dependencies and can be used to build any type of software project. Make is often used in conjunction with other build tools, such as autoconf and automake, to manage complex build processes.
5. MSBuild: MSBuild is a build automation tool that is used by Microsoft's .NET Framework. It uses an XML configuration file to define build targets and dependencies, and can be used to build .NET applications, libraries, and web services.
6. Grunt: Grunt is a JavaScript-based build automation tool that is commonly used in front-end web development. It uses a configuration file to define build tasks and dependencies, and supports a wide range of plugins for tasks such as CSS preprocessing, JavaScript minification, and image optimization.
7. Npm: build tool of node js application and react js application

Sonarqube

Integrating SonarQube with Jenkins allows you to perform automated code analysis and quality checks as part of your continuous integration and delivery (CI/CD) pipeline. SonarQube is a popular open-source platform for static code analysis, and Jenkins is a widely used automation server. Here's a step-by-step guide on how to integrate SonarQube with Jenkins:

1. Install and configure SonarQube: Start by downloading and installing SonarQube on a server or your local machine. Follow the installation instructions provided by SonarQube. Once installed, start the SonarQube server and access the SonarQube web interface.
2. Configure SonarQube analysis properties: In SonarQube, create a new project or select an existing one. Configure the analysis properties for your project, such as language-specific settings, quality profiles, and rules. This configuration determines the code quality standards against which your code will be analyzed.
3. Install the SonarQube Scanner plugin in Jenkins: In your Jenkins instance, navigate to "Manage Jenkins" and select "Manage Plugins." Install the "SonarQube Scanner" plugin if it's not already installed. This plugin allows Jenkins to run the code analysis using SonarQube.
4. Configure SonarQube server in Jenkins: In the Jenkins dashboard, go to "Manage Jenkins" and select "Configure System." Scroll down to the "SonarQube servers" section and click on "Add SonarQube." Provide the necessary details, including the SonarQube server URL and access token. The access token can be generated from the SonarQube web interface under "My Account > Security > Tokens."
5. Configure SonarQube analysis in Jenkins pipeline: Assuming you have a Jenkins pipeline in place, you need to add a step to execute the SonarQube analysis. Within your Jenkinsfile or pipeline script, add a stage for SonarQube analysis. Below is an example:

.6.Trigger the Jenkins pipeline: Once you have integrated SonarQube analysis into your pipeline, trigger the pipeline execution. Jenkins will execute the build steps and then run the SonarQube analysis as part of the defined stage. The analysis results will be sent to the SonarQube server, and you can view them in the SonarQube web interface.

6. View SonarQube analysis results: Access the SonarQube web interface and navigate to your project. You will find various reports and metrics related to code quality, including issues, code smells, code duplication, test coverage, and more. These results help you identify areas for improvement and enforce coding best practices.

By integrating SonarQube with Jenkins, you can automatically enforce code quality checks in your CI/CD pipeline, enabling early detection and resolution of potential issues, improving code maintainability, and ensuring higher-quality software releases.

JENKINS

Version = 2.36

Installed in **cd /var/lib/Jenkins** Log files should be stored at **/var/log/jenkins/jenkins**.

-------Jenkins Pipeline Job

From Scratch i have take the Repository details first and Repository url & i configure in the build servers. And i transfer SSH keys for athentication purpose. Or if I using https method and i will use credentials binding for any of the manage Credentials any of that servers to store Credentials on the jenkins server where ever we are trying to do Ci-CD process. So that i Can use username & password binding for decrypting the username & password So then I can clone it and clone github by using https method using username & Api key which is generated. So This is the first step and I can use webhooks to create multi branch pipeline So that developer or we need not Create any jobs. whenever we create a branch for testing purpose. suppose if one guy is to Created feature branch. If I use multi branch pipeline & webhook . It will automatically do branch indexing. And Create a new job for them so that manual intervention will not be there. if some one pushed that Code to Repository to his branch, The job is automatically trigered, manual intervention is Reduced so i tried to put these kind of automation as much aspossible and even I need to trouble shoot some errors. even the pipeline, I will Create a such a way that basic error handling like I put a try and Catch block error and Suppose if iam pushing to artifactory, iam getting a 404 error for that Perticular path that means its a basic error right for that time the artifacting may be down, the developers always Comes backs us to is saying that this job is not working so to Reduce that kind of effort always i will put try and catchblock & Catch that error i will see what is error Code based on that error Code i take some steps in pipeline itself. Suppose it is 404 error is i know its just a intermiedate error. after Some time when my server is back this job is Going through. So i put it to sleep for some time & retrigger the job automatically without any manual interventions so again my job will be reduced like manual intervention will be Reduced for that and I create this entire Pipeline stadgewise and some of the builds which are independent of each other in a parallel way so that i can reduce the time of build using parallel builds and also i Containersed a builds so that every-time I need not to go and Configure the Complicated Server and configration of what these things. it will take much of time so I create docker image for that & i will run a container and I run these build inside the container so that it will be easy to add any more servers for the build execution. And I call to the product team to know the build flow of the project and then write a Jenkins file to create multibranch pipeline cicd for that build and every repository I will create a Jenkinsfile and every feature branch triggers the job. When product team a commits a code the job builds stage by stage and first stage is code checkout from gitlab url

And The next stage is to build the code using a maven repository, it downloads various plugings and dependencies from maven local repository and central maven repository by using pom.xml. This stage compiles the code and creates a distributable package like a JAR or WAR file for java based web applications and archeieving artifacts in the target directly in the workspace of target directly.

2. And the stage is to run automated tests on the code to verify its correct of vernabalities ,bugs,code smells of unit test and integration test static analysis and code coverage for generation of file.xml. so this file so it not human readable formate right so the generated results and reports like junit and jacoco report publishing to sonarqube server dashboard with the help of sonar scanner tool.
3. And if the code does not pass in quality gates or threshold creatria is not met. the build gets unstable and successfull it shows indication in yellow colour for feature branch and staging brand but in relase branch threshold creatria is not met .the build gets failed
4. And then sending notification to the relevant stakeholders like developers, testers, and managers. This can be done using a Email Extension Plugin and build timestamp plugin for future reference. This can be useful for auditing and debugging purposes in the reports.
5. And next stage is to publish the artifact to JFrog private Artifactory by jfrog cli. This makes it easy to share the package with other environments. And I set environmental variable to set latest build id number and time framer and I pass to delete previous build id for upload the latest build artifactory.
6. And the next stage is to build a Docker customaized image from the dockerfile containes latest artifactory . and uploading the image into the docker private registry. This makes it easy to deploy the code in a containerized environment like Kubernetes. And even we are running some projects docker containerized builds for deployment.

7. Final stadge Deploy on Kubernetes Cluster: This stage is to deploy the Docker image on a Kubernetes cluster using a servicefiles and Deployment files to deploy of appilication for our end users.
8. These are the steps used complete automation of continous integration and the continous delivery.

Back to first page

JENKINS_HOME directory is used to store all the settings, configurations, and logs.

Pluggins used:
Git
Maven (compile and build)
SonarQube(code analysis)
Nexus (artifact repo)
Buildtimestamp (to version the artifact)
Slack notification
Blueocean
Docker
Junit
Email,aws,ansible ,jira

- **Triggers In Jenkins**
  Webhook in github=whenever the commit is made in git repository it ll trigger to build the job
  Periodically=schedule the cron job so it ll trigger only when the scheduled time comes.
  Poll SCM = schedule the trigger but it will run only if any changes in the repo

Upstream and downstream Jobs for example, build,test,deploy these are the 3 jobs – here for test job, build is a upstream job and deploy is downstream Job.

Manage Jenkins section Back to first page

- How to Configure email notification?

- Navigate to the Jenkins dashboard and click on "Manage Jenkins" on the left-hand side of the screen.
- Click on "Configure System" and scroll down to the "E-mail Notification" section.
- Enter the SMTP server details for your email service provider, including the host name and port number. You can also specify a username and password if necessary.
- Enter the default "System Admin e-mail address" that will be used as the sender for all emails.
- Scroll down to the "Extended E-mail Notification" section and check the "Enable" checkbox to enable email notifications.
- Enter the email address(es) of the recipients in the "Project Recipient List" field. You can separate multiple email addresses with a comma.
- Specify the events for which you want to receive email notifications, such as build failures or successes.

- How to create Jenkins Job with parameters?

- Create a new job in Jenkins or open an existing one.
- Click on "Configure" to edit the job's configuration.
- Scroll down to the "General" section and check the "This project is parameterized" checkbox.
- Click the "Add Parameter" button and select the type of parameter you want to add. Jenkins supports a variety of parameter types, including strings, integers, booleans, and choice parameters.
- Enter a name and default value for the parameter, and any other necessary configuration options.

- How to trigger the Jobs using GitHub webhook, Periodically, poll SCM?
  Webhook=whenever the commit is made in git repository it ll trigger to build the job
  Periodically=schedule the cron job so it ll trigger only when the scheduled time comes.
  Poll SCM = schedule the trigger but it will run only if any changes in the repo

- How to create Upstream and downstream Job?
  1.create a multiple Jobs. (for examplebuild,test,deploy – here for test job, build is a upstream job and deploy is downstream Job)
  2.create a relation between the Jobs in the **Post build action** (and also add parameters if needed
  3.you can view this Jobs in a Pipeline view. For that build pipeline view needs to be installed

-------Jenkins scripted and declrative pipeline

Scripted pipeline is a, programmatic approach to defining pipelines in Jenkins. The pipeline in using Groovy dsl language, which allow the pipeline, including stages, steps, and conditions.

Declarative pipeline, used in yaml formate it includes predefined sections for stages, steps, and post-actions on the other hand it is designed to be more readable and easier to use than Groovy scripts. It is used for building, testing, and deploying code.

the choice between scripted and declarative pipeline will depend on the user's needs and . Scripted pipeline may be a better choice for advanced pipeline use cases that require maximum flexibility and control, while declarative pipeline may be a better choice for simple pipeline use cases that require a more structured .

-------Jenkins Master slave architecture

Jenkins is a open-source automation server that allows to building, testing, and deploying software. The master-slave architecture in Jenkins is a distributed system. where one Jenkins master controls one or more slave node to perform builds and other tasks.and the master node is responsible for scheduling and delegating tasks to the slave nodes, and the slave nodes are responsible for executing those tasks. This allows for better resource utilization, as the work can be distributed among multiple machines, and also provides more flexibility in terms of scaling and workload management.

To set up a Jenkins master-slave architecture, :

Install Jenkins on the master node:

Configure the Jenkins master:

Install the Jenkins slave agent on the remote machine: .

Configure the Jenkins slave: .

Add the Jenkins slave node to the Jenkins master: like defining the slave node's label,

Configure your build job to use the Jenkins slave: .

Logs:

1. Jenkins log file: This is the primary log file for Jenkins, and it contains detailed information about what Jenkins is doing. The location of this file depends on how you installed Jenkins and what operating system you are using. Typically, the log file is located in the Jenkins home directory.
2. Build logs: Jenkins also stores logs for each build it executes. These logs are associated with the specific job and build number, and they contain information about what happened during that particular build. You can view build logs by going to the build's page in the Jenkins UI and clicking on the "Console Output" link.

-------Jenkins debugging and troubleshooting

- Enable remote debugging option on Jenkins by setting the "JENKINS_JAVA_OPTIONS" environment variable to "-Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=8000". This allows you to connect a remote debugger to Jenkins and debug issues.
- The Jenkins log file contains detailed information about the behavior of Jenkins. Logs can help identify the issue and the cause of the problem. You can access the logs by clicking on the "Manage Jenkins" link in the Jenkins UI, and then selecting the "System Log" option.
- Verify that the installed plugins are compatible with your version of Jenkins. Check the plugin documentation for known issue.
- If the issue is related to a specific plugin, try disabling that plugin to see if the issue goes away. This can help you determine if the plugin is the source of the problem.
- Check the Jenkins configuration is set up correctly. Check the Jenkins configuration files and settings for any issues or misconfigurations.

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                script {
                    try {
                        sh 'make'
                    } catch (Exception e) {
                        echo "Build failed with error: ${e.getMessage()}"
                    }
                }
            }
        }
    }
}
```

-------Pipeline as a code

Back to first page

```
Pipeline{
                agent any
            /*
                     tools {
                    maven "maven3"
                }
                environment {
                    registry = "imranvisualpath/vproappdock"
                    registryCredential = 'dockerhub'
                }
                stages{
                    stage('BUILD'){
                        steps {
                            sh 'mvn clean install -DskipTests'
                        }
                        post {
                            success {
                                echo 'Now Archiving...'
                                archiveArtifacts artifacts: '**/target/*.war'
                            }
                        }
                    }
                    stage('UNIT TEST'){
                        steps {
                            sh 'mvn test'
                        }
                    }
                    stage('INTEGRATION TEST'){
                        steps {
                            sh 'mvn verify -DskipUnitTests'
                        }
```

```
        }
        stage ('CODE ANALYSIS WITH CHECKSTYLE'){
            steps {
                sh 'mvn checkstyle:checkstyle'
            }
            post {
                success {
                    echo 'Generated Analysis Result'
                }
            }
        }
        stage('Building image') {
            steps{
              script {
                dockerImage = docker.build registry + ":$BUILD_NUMBER"
              }
            }
        }

        stage('Deploy Image') {
          steps{
            script {
              docker.withRegistry( '', registryCredential ) {
                dockerImage.push("$BUILD_NUMBER")
                dockerImage.push('latest')
              }
            }
          }
        }
        stage('Remove Unused docker image') {
          steps{
            sh "docker rmi $registry:$BUILD_NUMBER"
          }
        }
        stage('CODE ANALYSIS with SONARQUBE') {
            environment {
                scannerHome = tool 'mysonarscanner4'
            }
            steps {
                withSonarQubeEnv('sonar-pro') {
                    sh '''${scannerHome}/bin/sonar-scanner -Dsonar.projectKey=vprofile \
                   -Dsonar.projectName=vprofile-repo \
                   -Dsonar.projectVersion=1.0 \
                   -Dsonar.sources=src/ \
                   -Dsonar.java.binaries=target/test-classes/com/visualpathit/account/controllerTest/ \
                   -Dsonar.junit.reportsPath=target/surefire-reports/ \
                   -Dsonar.jacoco.reportsPath=target/jacoco.exec \
                   -Dsonar.java.checkstyle.reportPaths=target/checkstyle-result.xml'''
                }
                timeout(time: 10, unit: 'MINUTES') {
                    waitForQualityGate abortPipeline: true
                }
            }
        }
        stage('Kubernetes Deploy') {
           agent { label 'KOPS' }
            steps {
                    sh "helm upgrade --install --force vproifle-stack helm/vprofilecharts --set
appimage=${registry}:${BUILD_NUMBER} --namespace prod"
            }
        }
    }
}
```

Basic pipeline script

```
pipeline {
  agent any
  stages {
    stage('Checkout') {
      steps {
        git(
          url: 'https://gitlab.com/my-project.git',
          branch: 'master'
        )
      }
    }
    stage('Build') {
      steps {
        sh 'mvn clean package'
      }
    }
    stage('Deploy') {
      steps {
        sh 'scp target/my-project.war user@server:/path/to/deploy'
      }
    }
  }
}
```

-------jenkins parameterized jobs

Jenkins parameterized jobs allow users to pass parameters or variables to the job at runtime, making the job configuration more flexible and reusable.

1. String Parameters: This type of parameter allows users to enter a string value, like version number or URL website.
2. Choice Parameters: Choice parameters are useful for specifying to build type, target environment, or test suite.
3. Boolean Parameters: This type of parameter allows users to select a true or false value.
4. File Parameters: This type of parameter allows users to upload a file. File parameters are useful for providing configuration files to the job at runtime, such as a script or test configuration.
5. Matrix Parameters: This type of parameter allows users to run the same job with different combinations of values. like running a test suite with different database versions
6. Run Parameters: This type of parameter allows users to trigger another job and pass in parameters to it. Run parameters are useful for chaining jobs together, so that the output of one job becomes the input to another.

DOCKER

DOCKER version = 20.10

/var/lib/docker

- Docker is a containerization tool , like in case without docker it is hard to run the same application in the different environments without any conflicts. So Docker will solve this issue.
- You can develop, build, and package that and deploy to the any of the environment.
- In micro services project , for each service here we are creating docker file. Using the docker file we are building the docker image. this Docker image contains war file along with the dependencies.
- Then we are deploying .
- **Docker deamon**
- The Docker daemon is a background process that manages and controls the lifecycle of Docker containers. It is also responsible for managing Docker images, networks, volumes, and other Docker resources.

- The Docker daemon runs on the host machine and listens for commands from the Docker client. When a Docker command is issued by the client, such as running a container or building an image, the Docker daemon receives the command and executes it.

- The Docker daemon can be configured through various settings, such as specifying which network and storage drivers to use, setting up remote access to the daemon, and controlling logging and debugging output.

- The Docker daemon is a critical component of the Docker platform, as it provides the core functionality for managing and deploying containerized applications.

-------Docker components:

Docker is a platform that enables developers to build, package, and deploy applications as containers. Containers are lightweight, portable, and self-contained environments that can run on any system that supports Docker.

1. Docker Engine: It is responsible for running and managing containers, and provides an API and command-line interface for interacting with Docker.
2. Docker Images: Docker images are the templates for containers. They contain everything needed to run an application, including the application code, runtime, system tools, libraries, and other dependencies. Docker images can be created from scratch, or can be based on existing images.
3. Docker Containers: Docker containers are instances of Docker images that are running in a separate and isolated environment. Containers are self-contained, and have their own file system, network, and environment variables. They can be started, stopped, and removed independently of other containers.
4. Docker Registry: its like central repository for storing and distributing Docker images. It allows developers to share and download Docker images from a central location.
5. Docker Compose: is arunning multi-container Docker applications. It allows developers to define a group of containers that are linked together, and specify how they should be configured and started.
6. Docker Swarm: Docker Swarm is a container orchestration tool that allows developers to manage and scale clusters of Docker nodes. It enables developers to deploy and manage a large number of containers across multiple hosts, and provides features such as load balancing, service discovery, and rolling updates.

Now we are following the method containerization deployment, just we create the Dockerfile using the instructions, then we will build the image using the dockerfile , with the command **#docker build -t accountname/imagename:Version /dockerfilepath** and push the docker file to the dockerhub registry. We can also integrate the docker with Jenkins as well.

-------Docker file instructions:

FROM= it sets the base image to run container
CMD & ENTRYPOINT=entrypoint is ping cmd is wwwgoogle.com
While writing docker file for CMD , you give both command and argument like it work after container created like any service like systemctl command in that container.it will show the out put when we login into that container
But for ENTRYPOINT you give command only, argument ll be given later by the user.ENTRYPOINT has the higher priority than CMD.it can run any important script bt never we cant not change one line command like –c passing argument
ADD= copy from source to destination.also It will unzip the files as well(download from internet get_url)
COPY= similar to ADD**. Copy file from host to container host**
ENV= to set environment variables
EXPOSE= during a container runtime ,the container listens a specific network port for connectivity
LABEL= means tags , which you write information. And adds metadata to an image
USER= sets the username to use when running the image
VOLUME= to attach the docker volume to host for backup when container stops
WORKDIR= Sets the working directory to run the docker filre commands
ONBUILD= it adds the instructions to execute later
RUN=to excecute shell commands
MAINTAINER= sets the author information who generated that image
ARG= defines variables that users can pass at build time like docker build –arg variable

Back to first page

-------Docker Commands

#docker images = list all the images in our local machine.
#docker rmiimagename:tag= remove the image

By default, only root can run docker commands, if other users want to run docker command you need add them in docker group. #usermod -aG docker ec2-user

#docker run image = It will run the image & start the container.
#docker ps= list the running container
#docker ps -a = list all the container.

#docker pull nginx = it will pull the image from the docker registry(docker hub)
#docker run –name webserver -p 7090:80 -d nginx= this will run the container with nginx image
(if give -P it will automatically pickup host and container port)

#docker exec -it containername   /bin/bash = here you are inside the container.

To delete the image in case it is running as a container.
*stop the container **- #docker stop containerID**
*delete the container **- #docker rm containerID**
*then delete the image- **#docker rmi image**

- Tags in the docker is used for versioning
- In docker you can run same images of different versions or tags.

#docker logs containername= list the logs of the container. If something goes wrong while starting the container if you check the logs you will get to know
#docker inspect imagename= it will show you the meta data of the image in Json format
#docker inspect containername = shows the complete details about the container.

-------Docker volumes

A Docker volume is a way to store and manage persistent data used by Docker containers. A volume can be thought of as a directory or a file system that is outside the container's file system and can be shared by multiple containers.

When you create a volume in Docker, it is created as a separate entity in the Docker host's file system, and the data stored in the volume persists even after the container using it is deleted or recreated. This makes volumes a convenient way to manage persistent data such as logs, databases, or configuration files.

Docker volumes can be created using the `docker volume create` command, and can be mounted in a container using the `docker run` command with the `--mount` or `-v` option. Volumes can also be specified in a Docker Compose file, allowing you to manage volumes as part of your container orchestration.

In case if you delete the container , the data in it also get deletes, for that you have to create the volumes or bind mount , so the data of the container will be stored in the local machine.
#docker volume create= to create
#docker volume ls = lists all the volumes
#docker volume rm volumename= removes the volume
#docker volume inspect volumename =gives detailed information about the volume.
#docker volume prune = will delete all the unused volumes(volumes which are not attached to any of the container)

```
#docker run --name dbcontainer -e MYSQL_ROOT_PASSWORD=password  -p 3030:3306  -v /dbvolume:/var/lib/mysql -d
mysql.
```

-------Docker network

A Docker network is a virtual network that allows Docker containers to communicate with each other, as well as with other network resources, such as other containers, host machines, and external services.

Docker provides a variety of network drivers that can be used to create and manage Docker networks. These drivers include bridge, host, overlay, macvlan, and none, among others.

By default, when a Docker container is created, it is attached to the default bridge network, which provides connectivity to other containers on the same network. However, it is often useful to create custom networks to isolate containers and control their communication. Custom networks can also provide better security, as they allow administrators to define network policies and control access to the network resources.

Docker networks can be created and managed using the Docker CLI or through Docker Compose, which provides a more convenient way to manage multiple containers and networks.

-------Docker namespace

In Docker, a namespace is a way of isolating system resources from each other so that they can be used by different processes without interfering with each other. Docker uses several types of namespaces to provide container isolation, each of which provides a different level of isolation for different types of system resources.

1. PID Namespace: This namespace isolates process IDs, so that each process in the container has its own unique process ID.
2. Network Namespace: This namespace isolates network interfaces, so that each container has its own virtual network interface and IP address.
3. Mount Namespace: This namespace isolates file systems, so that each container has its own view of the file system, and changes made in one container do not affect other containers.
4. UTS Namespace: This namespace isolates hostnames, so that each container can have its own hostname and domain name.
5. IPC Namespace: This namespace isolates interprocess communication resources, so that each container has its own shared memory segments, semaphores, and message queues.
6. User Namespace: This namespace isolates user and group IDs, so that each container can have its own root user and group ID, without affecting the root user and group ID on the host system.

-------Docker   c groups

Cgroups (short for control groups) is a Linux kernel feature that allows you to limit and prioritize system resources such as CPU, memory, disk I/O, and network bandwidth for a group of processes. Docker makes use of cgroups to provide containerization and resource isolation.

When you start a Docker container, the Docker daemon creates a cgroup hierarchy for that container. The cgroup hierarchy is a collection of cgroups that are used to limit the resources available to the processes inside the container.

For example, you can use cgroups to limit the amount of CPU and memory that a container can use. This can prevent a single container from monopolizing the resources of the host system and causing performance problems for other containers or processes running on the same system.

-------Docker stateless and statefull appli

stateless applications can be easily deployed as Docker containers, whereas stateful applications require some additional configuration to ensure that the data is preserved across container instances. Docker provides features such as volumes and mounts to help manage persistent data in stateful applications.

Stateless applications can be deployed as Docker containers by creating a new container for each request, and then discarding the container once the request is completed

Stateful applications can be deployed as Docker containers, but they require some additional configuration to ensure that the data is preserved across container instances

-------Docker backup configuations

Terraform does not have a built-in backup mechanism for storing previous versions of your infrastructure code. However, you can still create backups of your Terraform configuration files manually using standard file backup procedures.

Another way to create backups is to copy your Terraform configuration files to a backup directory on a regular basis. You can use tools like rsync or tar to automate this process.

-------Bind mount

Bind mount Is mostly used To inject the data from host machine to the container.

```
#docker run --name dbcontainer -e MYSQL_ROOT_PASSWORD -p 3030:3306  -v /dbvolume:/var/lib/mysql -d mysql.
```

Docker file

Build a docker file

```
#vi Dockerfile
```

While Building the image using any base images like, tomcat,sql you have to set the environment variables, & information about what variables need to set will be available in docker hub documentation.

Build the image

```
#docker build -t  imagename  /dockerfilepath
```

Back to first page

-------Publish the image

To push the image to the docker hub the image name should have account name as well

```
#docker build -t  shashankthumbri/imagename:Version  /dockerfilepath
```

Login to the docker hub = **#docker login**

Push the image to the docker hub = **#docker push shashankthumbri/imagename:Version**

------Docker file

```
FROM openjdk:8 AS BUILD_IMAGE
RUN apt update && apt install maven -y
RUN git clone -b vp-docker https://github.com/imranvisualpath/vprofile-repo.git
RUN cd directory-repo &&mvn install
```

```
FROM tomcat:8-jre11

RUN rm -rf /usr/local/tomcat/webapps/*

COPY --from=BUILD_IMAGE vprofile-repo/target/vprofile-v2.war /usr/local/tomcat/webapps/ROOT.war

EXPOSE 8080
CMD ["catalina.sh", "run"]
```

```
cd git clone -b docker https://github.com/devopshydclub/vprofile-project.git
cd vprofile-project/Docker-files/app/
ls
cd multistage/
ls
cat Dockerfile
docker build -t appimg:v1 .
docker images
```

Questions:

- **Docker Multistage Build?**

  With multi-stage builds, you use multiple `FROM` statements in your Dockerfile. Each `FROM` instruction can use a different base, and each of them begins a new stage of the build. You can selectively copy artifacts from one stage to another, like we can use previous layers output to build the new layer leaving everything you don't want in the final image. Basically we will be using multiple FROM commands and multiple COPY commands. It will allow us to create lightwight image which will utilize the less resources.

- **different states of docker containers?**

  Created.
  Running
  Restarting
  Exited
  Paused
  Dead

- **Docker repository vs docker registry?**

  A registry stores a collection of repositories. You could say a registry has many repositories and a repository has many different versions of the same image which are individually versioned with tags. Docker hub Is a registry which holds the repositories

-------Docker compose file

Docker compose file.yaml

```
. version: '3'
services:
  vprodb:
    image: prashanthba/vprofiledb:v1
    ports:
      - "3386:3386"
    volumes:
      - vprodbdata:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=vprodbpass

  vprocache01:
```

```
  image: memcached
  ports:
   - "11211:11211"

 vpromq01:
  image: rabbitmq
  ports:
   - "15672:15672"
  environment:
   - RABBITMQ_DEFAULT_USER=guest
   - RABBITMQ_DEFAULT_PASS=guest

 vproapp:
  image: prashanthba/vprofileapp:v1
  ports:
   - "8080:8080"
  volumes:
   - vproappdata:/usr/local/tomcat/webapps

 vproweb:
  image: prashanthba/vprofileweb:v1
  ports:
   - "80:80"

volumes:
 vprodbdata: {}
 vproappdata: {}
```

-------Docker logs and debugging

View container logs:

To view the logs of a running container, you can use the **`docker logs`** command followed by the name or ID of the container.

View specific logs:

You can use the **`docker logs`** command with options to view specific logs, such as the last N lines or logs since a certain timestamp.
For example: docker logs –-tail 100 container id container

This will display the last 100 lines of logs for the **`my-container`** container

Follow logs:

You can use the **`--follow`** option with the **`docker logs`** command to follow the logs in real time. This is useful for monitoring the output of a running container.

Debugging

Check container logs:
debugging a container is to check its logs using the docker logs command. This will show you the output of the container's stdout and stderr streams, which can help you identify any errors
docker logs my-container

Access container shell:

i can access a shell inside a running container using the docker exec command. This can be useful for inspecting the container's filesystem, running commands, and debugging issues.

```
docker exec -it my-container /bin/bash
```

This will open a shell inside the my-container container.

Check container environment:

i can use the docker inspect command to view the environment variables and configuration settings of a container. This can help you identify any configuration issues or misconfigurations that are causing problems.

```
docker inspect my-container
```

Use debug tools:

i can install and use various debug tools inside a container, such as strace, tcpdump, and netcat, to diagnose issues.

```
docker run –rm -it alpine sh
```

This will run a temporary container with the Alpine Linux image and install the strace tool, which can be used to trace system calls made by a command.

Use debugging containers:

There are several pre-built Docker containers that are designed specifically for debugging, such as busybox, alpine, and ubuntu. These containers include a wide range of debugging tools and utilities that can help you diagnose issues in other containers.

```
docker run –rm -it –net=container:my-container busybox sh
```

TERRAFORM

TERRAFORM version = 1.3

- Terraform is a tool for infrastructure as a code , so we can create& manage our infrastructure with code.
- And also we can define the **state** of our cloud services but its not possible with ansible.
- But One drawback of terraform is like when we edit the existing script , it will delete the existing infra and recreate. SoChances of loosing data more.
- Terraform file ends with Terraform.tf extension
- Where the terraform file exists , there 2 more files also exists , **State file &Backupfile** . Statefile Consist complete details of the infra.

-------Commands

#terraform init = initialize the terraform in the Directory where your terraform code file, terraform.tf exists.
#terraform validate = validates the terraform scrips for syntax errors.
#terraform fmt = restructure the syntax of our code.
#terraform plan = it will just show us the excecution like what it will excecute.

#terraform apply = excecute the terraform file
#terraform destroy = Destroy the infra created by that file.

-------null resourse

a null resource is a resource that has no real infrastructure associated with it, and is used to perform some specific actions outside the normal resource lifecycle. A null resource is typically used to run local-exec provisioners or to define dependencies between resources.

```
resource "null_resource" "my_resource" {

  provisioner "local-exec" {

    command = "echo 'Hello, world!'"
```

-------EC2 InstanceLaunch(code)

```
provider "aws" {step:1     we mention the cloud provider
  region = "us-east-2"
access_key = ""
secret_key = ""

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

resource “aws_key_pair” “key” {                              step2:   we create Key pair
key_name = ”mykey”
public_key = file" “
 }
-----------------------------------------------------------------------------------------------------------------------------

resource "aws_instance" "name-WEB" {step3:     we create instance
               ami  = "ami-03657b56516ab7912"
               instance_type        = "t2.micro"
               availability_zone      = "us-east-2a"
               key_name              = "aws_key_pair.key.mykey"
               vpc_security_group_ids = ["sg-ID"]
                 tags = {
               Name     = "Instance name"
                 Project = "Project name"
 }
}
```

Back to first page

-------Security group (inbound and outbound)

```
resource "aws_security_group" "my_stack_sg" {
vpc_id      = aws_vpc.my.id
  name        = "my-stack-sg"
  description = "Sec Grp for my ssh"
  egress {inbound rule
from_port   = 0
to_port     = 0
    protocol    = "-1"
cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {outbound rule
from_port   = 22
to_port     = 22
    protocol    = "tcp"
cidr_blocks = [var.MYIP]
  }
  tags = {
    Name = "allow-ssh"
  }
}
```

-------Volumes(create and attach)

```
resource "aws_ebs_volume" "vol_1" {(volume creation)
availability_zone = var.ZONE1
 size           = 3
 tags = {
   Name = "extr-vol-1"
 }
}

resource "aws_volume_attachment" "atch_vol" {(volume attaching)
device_name = "/dev/xvdh"
volume_id   = aws_ebs_volume.vol_1.id
instance_id = aws_instance.my-web.id
}
```

```
provisioner  “remote-exec” {
    inline = [
           “commands to run”                          step 4:   execute commands remotely
    ]
}

Connection  {
           Type = “SSH”
           User = “root”
           Password = xxx
  }
```

```
output "PublicIP" {                                 to print any output for ex: Public IP
 value = aws_instance.instance_name.public_ipresource type.resource name.details what u want to print
}

output "PrivateIP" {
 value = aws_instance.resource-name.private_ip
}
```

Back to first page

-------vpc

```
provider "aws" {step:1     we mention the cloud provider
 region = "us-east-2"
access_key = ""
secret_key = ""

resource “aws_vpc” “prod-vpc” {
cidr_block = “10.0.0.0/16”
enable_dns_support = “true” #gives you an internal domain name
enable_dns_hostnames = “true” #gives you an internal host name
instance_tenancy = “default”   (Tenancy defines how EC2 instances are distributed across physical hardware and affects pricing. )

tags {
Name = “prod-vpc”
}
}
```

-------Variables (code to launch instance)

wehave to create 3 files
Vars.tf = here we ll define variables
Providers.tf = cloud provider is defined here
infra.tf = here we write our code.

Providers.tf =

```
provider "aws" {
  region = var.REGION
}
```

Vars.tf =

```
variable REGION {
  default = "us-east-2"
}
------------------------------------------------------------------------------------------------------------------------------------
variable ZONE1 {
  default = "us-east-2a"
}
------------------------------------------------------------------------------------------------------------------------------------
variable AMIS {
  type = map
  default = {
    us-east-2 = "ami-xxxxx"
    us-east-1 = "ami-xxxxx"
  }
}
-------------------------------------------------------------------------------------------------------------------------------------
variable “PRIV_KEY_PATH” {
      default = “key-name”
}
-----------------------------------------------------------------------------------------------------------------------------------
variable “USER” {
      default = “username”
}
```

Terraform.tf=

```
resource "aws_instance" "my-inst" {
ami                = var.AMIS[var.REGION]
instance_type          = "t2.micro"
availability_zone       = var.ZONE1
key_name              = "new-my"
vpc_security_group_ids = ["sg-0780815f55104be8a"]
  tags = {
    Name    = "My-Instance"
    Project = "My"
  }
}
```

[Back to first page](#)

-------Provisioners

Provision is like, to do the task once our operating system is up, in general it is to provision some packages, files or upload artifacts etc..

File= Provision to share files remotely
Remote-exec = to execute the commands remotely
Local-exec = to execute the task locally, where yourterraform is there.

```
Provisioner “file” {
            Source = “/source”
            Destination = “/dest”
}

provisioner  “remote-exec” {
    inline = [
            “commands to run”
    ]
}
```

```
Connection  {
            Type = “SSH”
            User = “root”
            Password = xxx
   }
}
```

for remote-exec you have to mention the connection.

-------output (print details after instance creation)

If you want to print any details of the instace after creating , for example – to print public IP or Private IP after creating the instance.

```
output "PublicIP" {
  value = aws_instance.instance_name.public_ipresource type.resource name.details what u want to print
}

output "PrivateIP" {
  value = aws_instance.resource-name.private_ip
}
```

Back to first page

-------backend-Statefile in remote location

In terraform if somebody changes the state of the infrastructure it ll create a problem , so to sync the state of the infrastructure we have tokeep the terraform.tfstate file in a remote location like S3 Bucket.

```
terraform {
  backend "s3" {
bucket = "bucketname"
 key    = "terraform/backend"
 region = "us-east-2"
}
}
```

The statefile is important because it allows Terraform to keep track of the changes made to the infrastructure over time. This information is used to determine which resources need to be created, updated, or destroyed when changes are made to the configuration.

By default, the statefile is stored locally on the machine where Terraform is running. However, this can be configured to use remote storage, such as an S3 bucket, for improved collaboration and state management.

It's important to ensure that the statefile is properly managed, as it contains sensitive information about the infrastructure environment. If the statefile is lost or becomes corrupted, it can lead to inconsistencies in the infrastructure state and potentially cause issues with the infrastructure

In Terraform, a workspace is a way to create multiple instances of the same set of infrastructure resources within a single Terraform configuration. Workspaces allow you to create separate environments for your infrastructure, such as development, staging, and production, without duplicating your entire Terraform configuratio

-------Questions

- **Instance is already running, How to provision it under Terraform?**
  For this we use the #Terraform Import, what it will do is, it will map the existing infrastructure to the terraform code which is written and it will update the state to improve the configuration modify the terraform.tfstate file.

- **Create multiple instances at a time in terraform?**
  We can create with the help of **count** or **for_each.** if the configuration of the instances are same we can use **count .**
  If the configuration of instances are different we can use **for-each** loop.
  For that we have to create different variable files, in the variable files we ll be mentioning configuration details(like instance type of instance zone), other things are written in the resource block, by this way using for-each we can provision multiple instances.

ANSIBLE

/etc/ansible.

- **Ansible.cfg** **hosts** **host_vars** **group_vars** **roles** **templates**

- We are using ansible tool mainly for configuring like upgradation process. And also, for deployment purpose we are using.

- . Ansible is installed on the controlling server, and the controlling machine manage the nodes via SSH.

- Suppose in node 1 tomcat server is there, from repository we are taking war file and deploy that in webapps folder in the tomcat server, using ansible playbook.
- And since Jenkins and ansible are in the same environment , as we written in the Jenkins file, as soon as war file generated in the work space it will copy to the tomcat webapps, by this way linking happens between Jenkins and ansible

- - Ansible is a Control machine not a server

  -Ansible is agentless

-Connection to the servers through ssh(for linux), winrm(for windows),api(for cloud)

-Ansible playbooks written in yaml format

Back to first page

-------Inventory file:

In ansible you have to create a inventory file (default inventory file. **/etc/ansible/hosts)** , where information of target machine(ip,username,key) are listed in that file.

Here you can add group and add servers to that group. You can also create group of groups.

-Default Inventory file /etc/ansible/hosts

```
Web01 ansible_host=172.38.15.254        (hosts)
Web02 ansible_host=172.38.15.120
Db01 ansible_host=172.38.15.110
Db02 ansible_host=172.38.15.100

[webgroup1]                             (groups)
Web01
Web02

[Dbgroup1]
Db01
Db02

[maingroup:children]               (parent group- adding all the group)
Webgroup1
Dbgroup1

[maingroup:vars]       (adding all variable to the main group, it ll applicable for all hosts)
ansible_user=centos
ansible_ssh_private_key_file=privatekeyname.pem
```

-------passwordless connection

Ansible requires passwordless connection, for that generate keys(#ssh-keygen), then copy the keys to the client machine(command is #ssh-copy-id root@ipaddress).
And also, in client machine in /etc/ssh/sshd_config file, make passwordless connection **yes**

-------Variables:

/etc/ansible.

Ansible.cfg hosts host_vars group_vars roles

- **To set the variables in separate files then you must create the host_vars&group_vars directories .**
  -To create variables for separate servers, then create a file with server name under host_vars directory and
  -To create variables for separate Groups, then create a file with group name under group_vars directory

In variables highest priority goes to command line variable, then variable in the playbook, then the host file variable , then group file variable , and then global variable.

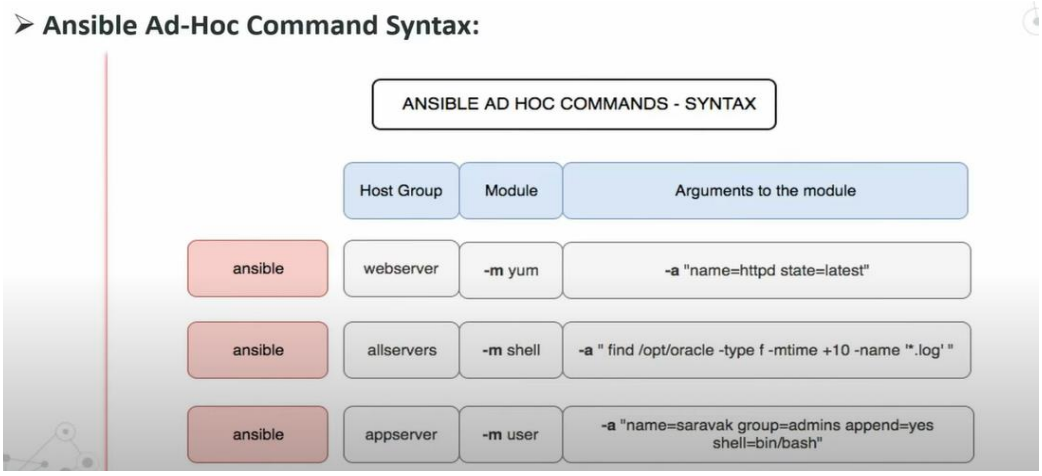
-------logs

- To get the ansible logs you must edit the ansible.cfg file , there you have to append the line **log_path=/tmp/ansible** under defaults line.

Back to first page

-------Ad-Hoc Command:

-In Ansible, ad-hoc commands are used to run simple tasks on remote hosts without creating a playbook. Ad-hoc commands are useful for tasks that are performed infrequently or tasks that don't require the complexity of a playbook



➢ **Ansible Ad-Hoc Command Syntax:**

ANSIBLE AD HOC COMMANDS - SYNTAX

	Host Group	Module	Arguments to the module
ansible	webserver	**-m** yum	**-a** "name=httpd state=latest"
ansible	allservers	**-m** shell	**-a** " find /opt/oracle -type f -mtime +10 -name '*.log' "
ansible	appserver	**-m** user	**-a** "name=saravak group=admins append=yes shell=bin/bash"

-------Modules

file = used to change the properties of the file, like permission ownership, also you can create directories links. (ansible -I inventoryfile -m file -a "path==/tmp state==directory mode==0775)
(to create **link** use state: link in file module**)**
Yum= Ansible -iinventoryfile -m yum -a "name=httpd state=latest" servername –become
(to uninstall change the state to absent)
Ping= Ansible -iinventoryfile -m ping servername --become
Copy= Ansible -iinventoryfile -m copy -a "src=source dest=destination" servername --become

Debug= it is to print the message (like echo command)
Register= store the output of the tasks into the variable that we define.
When= to define if statements we used when module (when you wanted to install a package in both centOS and ubuntu that time you can't use yum module for both the OS, that time will use when module
when: ansible_distribution == "centOS")
loop= to execute multiple tasks you have to use loop module(for example when installing a httpd, same time you want to install some other packages , then you can use loop module. Or to add multiple users)
setup=setup module helps us in gathering data about the target nodes like hostname, ip, os etc
cron= cron module is used to manage crontab entries and define environment variables.

Service= module is used to control service on target nodes, we can start/stop/restart/reload a service through this module. For windows-based target machine you can use win_service module.
Fetch=
Shell=to execute any commands in shell

https://faun.pub/ansible-30-most-important-modules-for-devops-professional-part-1-fdd536b0790d

-------Playbook

Back to first page

-To execute the tasks, we have to write a playbook, where we ll mention the hosts, and modules, then execute the play book.first we need to crate hostfile and playbook.yml
To **check the syntax error** in the playbook file = ansible-playbook playbook.yml --syntax-check

ansible-playbook -i hosts(configure host ips and ansible username password conection port) playbook.yml(collections of plays) –e 'pkg download =git supose' we can pass variables in command itself --skip-tags prashanth(we can skip that perticural module by passing tags in module) or (we can use - tags in command to run particular module – tags thatname whatever u can given name in the module)--ask-vault-pass or – -vault-passwordfile (we can store password or secrets ids in a file and ansible-vault encrypt that file it can in command line) if suppose if any errors in in particular module that module is not important on that time we can pass ignore_errors: yes provide in that module it skips that module) and if we want see logs during running a playbook what is happening the process we can run –vvv to see logs in verbose mode

Playbook contents

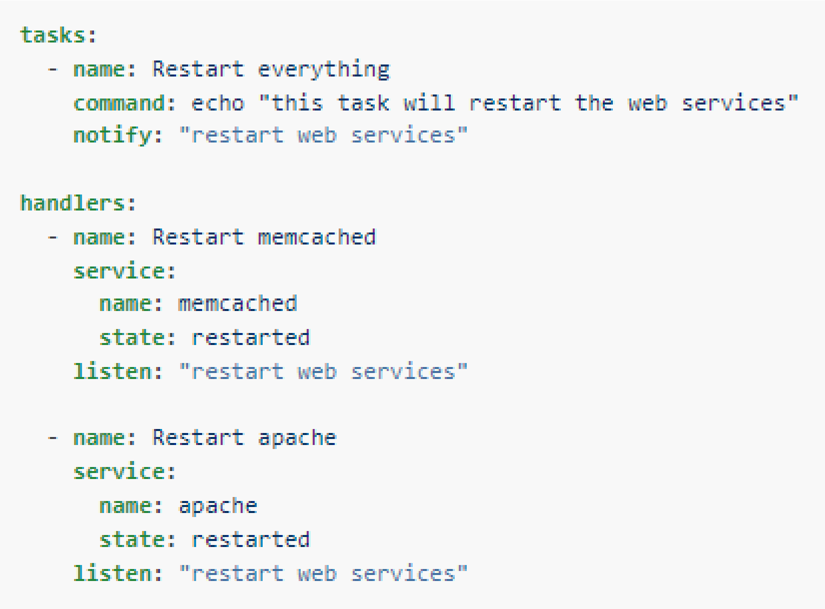
```
---
- name: maven-project
  hosts: linux
  become: yes
  vars_files:
    - gitcredentials.yml
  vars:
    pkgdownload1: 'git'
    pkgdownload2: 'java'
  tasks:
    - name: install git
      yum :
        name: "{{pkgdownload1}}"
        state: present
      ignore_errors: no

    - name: install maven
      get_url:
        url: https://dlcdn.apache.org/maven/maven-3/3.8.7/binaries/apache-maven-3.8.7-bin.tar.gz
        dest: /opt

    - name: extract maven from tar file
      unarchive:
        src: /opt/apache-maven-3.8.7-bin.tar.gz
        dest: /opt
        remote_src: yes

    - name: Defining MVN_HOME
      lineinfile:
        dest: /etc/profile
        state: present
        line: ‘{{ item }}’
      with_items:
        – ‘export MVN_HOME=apache-maven-3.6.3’
        – ‘export PATH=$MVN_HOME/bin:$PATH’

    - name: update sourece
      shell: source ~/.bash_profile
      args:
        executable: /bin/bash
      ignore_errors: no

    - name: create a directory
      file:
        path: /home/devops/maventest
        state: directory
        mode: '0755'

    - name: check whether folder is exits or not if exits delete it
      stat:
        path: /home/devops/maventest/maven-project
      register: check

    - debug:
        var: check

    - name: project exits delete it
      file:
        path: /home/devops/maventest/maven-project
        state: absent
      when: check.stat.exists

    - name: clone src from gitlab
      shell: git clone https://"{{username}}":"{{apikey}}"@gitlab.com/prashanth52/maven-project.git
      args:
        chdir: /home/devops/maventest

    - name: Adding the path in the Profile files
      shell: export M2_HOME=/opt/apache-maven-3.8.7; export PATH=${M2_HOME}/bin:${PATH}; mvn clean install
      args:
        chdir: /home/devops/maventest/maven-project
```

-------Handlers

if the changed value in the operation is true , then handlers will notified to execute the task

Whenever you want to execute something if there is any change , you can call for handlers. Handlers will execute only at the end of playbook.

And also
For example, there are 2 servers you want to restart by calling handlers.



```
tasks:
  - name: Restart everything
    command: echo "this task will restart the web services"
    notify: "restart web services"

handlers:
  - name: Restart memcached
    service:
      name: memcached
      state: restarted
    listen: "restart web services"

  - name: Restart apache
    service:
      name: apache
      state: restarted
    listen: "restart web services"
```

-------calling one playbook to another playbook

- import_playbook: test.yaml
  tags: test

-------templates

Templates are similar to any other files but difference is you can have dynamic data.

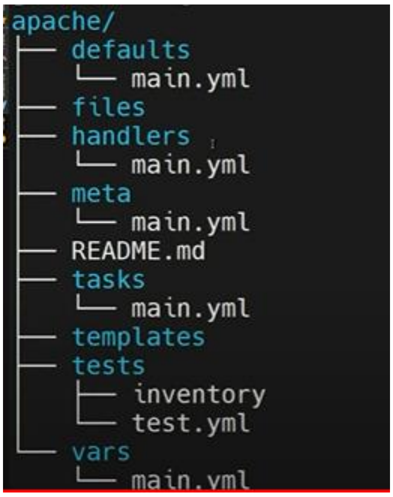
In templates you can have variables, when I am deploying with template module , template module will see if there is any dynamic content and replace with actual content & then push it.

Templates files ends with j2 extension (jinja2)

-------roles

By using roles it segregates all the content in the playbook. It is easy to access and make changes.
And roles can be re-usable.
For example, in case of tomcat server. Same can be used in different different projects

In case we are going to install and configure apache by using ansible roles , then
Run # ansible-galaxy init /etc/ansible/roles/apache = here directory structure ll be created



```
apache/
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── README.md
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml
```

.

In the **task** directory create the files with names install.yml(install related tasks),

config.yml (config related tasks),
service.yml(service-related tasks).
In the main.yml import tasks and mention these files.
In the **files** directory for example create index.html file
In **handler's** directory mention restart tasks in the main.yml
In **meta** directory you can give author and description.

In the ansible galaxy community repository, there are many roles that we can customize and use.
To install roles, **#Ansible-galaxy install rolename**

Create setup.yml file in the /etc/ansible directory and then you mention the hosts and roles for the hosts

Ansible-galaxy init
ansible-galaxy any package.yml and create like this different packages and place in file like import_roles.yml and run ansible-playbook -i hosts import_roles.yml --vault-password-file pass.yml

```
- name: import roles
  hosts: linux
  become: yes
  gather_facts: true
  vars_files:
    - gitcredentials.yml
  vars:
    pkgdownload1: 'git'
    pkgdownload2: 'java'

  tasks:
    - import_role:
        name: download

    - import_role:
        name: git_clone
```

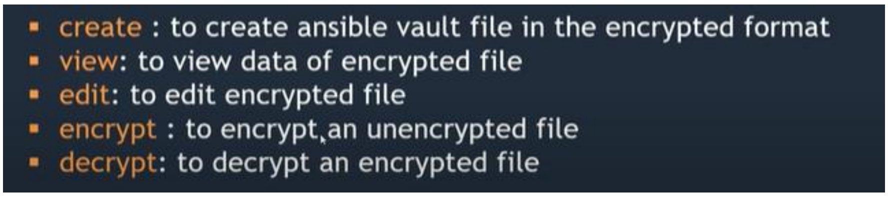
-------vaults

[Back to first page]

Vault is a feature that allows us to keep sensitive data like password or keys or any files in the encrypted files.

By this way we can protect our important data. For example , in a scenario while writing a playbook , if you want to hide a password in the play book, for that you have a create one file and mention the password in that file and encrypt that file. And call a variable file in the playbook which will redirect to the password file.

There are commands in ansible vault, like
Ansible-vault create credentials.yml



- create : to create ansible vault file in the encrypted format
- view: to view data of encrypted file
- edit: to edit encrypted file
- encrypt : to encrypt an unencrypted file
- decrypt: to decrypt an encrypted file

-top modules- https://faun.pub/ansible-30-most-important-modules-for-devops-professional-part-1-fdd536b0790d

-------idempotency

In Ansible, "idempotency" refers to the ability of a playbook or task to be executed multiple times without causing unintended changes to the system. In other words, if the same playbook or task is run multiple times, it should always produce the same result, without making any unnecessary or unintended changes.

Idempotency is a key feature of Ansible and helps ensure that the system is in a desired state, regardless of how many times the playbook or task is run. It also helps prevent unintended side effects or system disruptions that can occur when changes are made repeatedly.

-------Patching playbook

Patching of linux kernel

```
# vi patching.yml
```

```
---
- name: Patch Linux system
  hosts: Linux_Servers
  become: true
  tasks:
- name: Copy the Kernel Patch Repo File
  copy:
src: /etc/yum.repos.d/yum.repo
dest: /etc/yum.repos.d/
- name: Apply patches
  yum:
     name: kernel
     state: latest
```

-------deployment playbook

```
---
    - hosts: tomcatServer
     vars:
     - warName: ROOT.war
     - warRemotePath: /home/tomcat

     tasks:
     - name: Download WAR to server
       command: wget http://git-internal/release.war -O {{ warRemotePath }}/{{ warName }}
     - name: get current date
    set_fact: bkpdate="{{ lookup('pipe', 'date +%Y%m%d-%H%M') }}"

     - name: create directory with a date in name
       file: path="/home/tomcat/{{ bkpdate }}"
             state=directory
             mode=0755

     - name: backup war
       shell: "cp /usr/local/tomcat/webapps/{{ warName }} /home/tomcat/{{ bkpdate }}/"
     - name: Unzip WAR file
```

[Back to first page]

```
    unarchive: src={{ warRemotePath }}/{{ warName }} dest=/usr/local/tomcat/webapps/ROOT/ copy=no
mode=0755 owner=tomcat7 group=tomcat7
    notify:
        - restart tomcat7
  - name: Delete remote war file
    file: path={{ warRemotePath }}/{{ warName }} state=absent
  - name: wait for tomcat to start
wait_for: port=8080 timeout=60
  handlers:
    - name: Restart tomcat7
      service: name=tomcat7 state=restarted
```

-------ansible logs and debugging

. Logging to a file:

You can specify a log file by using the `log_path` option in the Ansible configuration file or by passing the `--log-file` option on the command line. For example, to set a log file for all Ansible commands, i can add the following line to the Ansible configuration file:

2. Logging to syslog:

You can also configure Ansible to log to a syslog server by setting the `syslog_facility` and `syslog_host` options in the Ansible configuration fi

3. Verbose logging:

You can also increase the verbosity of Ansible output by using the `-v` or `-vv` options on the command line. This will print more detailed output to the console or log file.

Debugging erros:

3. Verbose logging:

You can also increase the verbosity of Ansible output by using the `-v` or `-vv` options on the command line. This will print more detailed output to the console or log file.

2. Check syntax:

You can use the `ansible-playbook --syntax-check` command to check the syntax of your playbook without executing it. This will detect any syntax errors in your YAML file and give you an error message

3. Dry run:

You can use the `ansible-playbook --check`

4. Use debug module:

You can use the `debug` module to print the values of variables and expressions during playbook execution

5. Use the Ansible debugger:

Ansible provides a built-in debugger that allows you to interactively debug your playbook. The debugger lets you step through your playbook one task at a time, inspect variables, and execute ad-hoc commands on the target host. To use the debugger, add the `-D` or `--debugger` option to the `ansible-playbook`

-------ansible tower

Ansible Tower is an enterprise-level IT automation platform that provides a web-based user interface, REST API, and other features to help organizations manage and scale their Ansible automation infrastructure. It acts as a centralized hub for running Ansible playbooks, managing inventories, scheduling jobs, and monitoring activities across multiple systems and environments.

Here are some key features and concepts related to Ansible Tower:

1. Role-Based Access Control (RBAC): Ansible Tower allows administrators to define user roles and permissions, providing fine-grained access control over who can perform specific actions within the system.
2. Job Templates: Job templates define the Ansible playbook, inventory, and other parameters required to execute automation tasks. They provide a way to create reusable configurations for different scenarios.
3. Inventory Management: Ansible Tower offers inventory management capabilities, allowing you to define and organize your infrastructure resources (hosts) into groups and assign variables to them.
4. Scheduling and Workflow Automation: You can schedule Ansible jobs to run at specific times or on a recurring basis. Complex workflows involving multiple jobs and dependencies can also be defined in Tower.
5. Notification and Logging: Tower provides features for sending notifications and alerts based on job status or events. It also logs all job activities, making it easier to track changes and troubleshoot issues.
6. REST API: Ansible Tower exposes a RESTful API that enables programmatic interaction with the system. This API can be used to automate tasks, integrate with other systems, or build custom applications.

Ansible Tower offers a more scalable and controlled approach to managing Ansible automation at an enterprise level. It provides enhanced security, auditing, and collaboration features compared to using Ansible alone from the command line.

1. What is Ansible Tower, and what are its key features?
2. How does Ansible Tower differ from Ansible?
3. What is the purpose of Role-Based Access Control (RBAC) in Ansible Tower?
4. How do you define job templates in Ansible Tower? What components do they include?
5. Explain how inventories are managed in Ansible Tower.
6. What is the significance of scheduling in Ansible Tower? How can you schedule jobs?
7. How does Ansible Tower handle notification and logging of job activities?
8. What is the REST API in Ansible Tower used for? Provide some examples of its usage.
9. How would you troubleshoot issues in Ansible Tower? What are some common challenges you might face?
10. Describe a scenario where Ansible Tower would be beneficial for an organization.
11. How would you integrate Ansible Tower with other tools or systems?
12. What are some best practices for managing and organizing Ansible Tower projects and resources?
13. Can you explain how Ansible Tower handles scaling and high availability?
14. What security measures are available in Ansible Tower to protect sensitive information like credentials?
15. How do you upgrade Ansible Tower to a new version?
16.

1. Ansible Tower is an enterprise-level IT automation platform that provides a web-based user interface, REST API, and other features to help organizations manage and scale their Ansible automation infrastructure. Its key features include role-based access control (RBAC), job templates, inventory management, scheduling, notification and logging, and a REST API.
2. Ansible Tower differs from Ansible in that it provides a centralized management interface, RBAC, scheduling, and logging capabilities, which Ansible alone does not have. Ansible Tower is designed for larger deployments where multiple users, teams, and environments need to collaborate and manage automation tasks more efficiently.
3. Role-Based Access Control (RBAC) in Ansible Tower allows administrators to define user roles and permissions. It provides fine-grained access control, ensuring that only authorized users can perform specific actions within the system. RBAC enhances security and helps organizations enforce access control policies.
4. Job templates in Ansible Tower define the Ansible playbook, inventory, and other parameters required to execute automation tasks. They allow for the creation of reusable configurations for different scenarios. Job templates also specify variables, credentials, and other settings needed to run the automation tasks consistently.
5. Inventories in Ansible Tower are used to manage infrastructure resources (hosts). They allow for organizing hosts into groups and assigning variables to them. Inventories can be dynamic, pulling information from external sources, or static, defined within Ansible Tower. Inventories are essential for targeting specific hosts or groups of hosts for automation tasks.
6. Scheduling in Ansible Tower allows for the automated execution of Ansible jobs at specific times or on a recurring basis. Jobs can be scheduled to run once or repeatedly, providing flexibility for routine tasks or regular updates. Scheduling helps organizations automate repetitive tasks and maintain a predictable automation workflow.
7. Ansible Tower provides notification and logging features. Notifications can be configured to alert users or teams based on job status or specific events. Logging captures all job activities, providing a comprehensive audit trail and facilitating troubleshooting. These features enhance visibility and enable better tracking and analysis of automation activities.
8. The REST API in Ansible Tower allows for programmatic interaction with the system. It enables automation, integration with other tools or systems, and the development of custom applications. Using the REST API, you can create, update, and delete objects such as job templates, inventories, and credentials. You can also retrieve job status, logs, and other relevant information.
9. Troubleshooting issues in Ansible Tower may involve checking job logs, reviewing configuration settings, verifying network connectivity, and examining error messages. Common challenges include credential management, connectivity issues with target hosts, misconfigured job templates or playbooks, and permission-related problems. Familiarity with Ansible Tower's troubleshooting tools, such as logging and debugging features, is crucial for resolving issues effectively.
10. Ansible Tower is beneficial for organizations that require centralized management and control of their Ansible automation infrastructure. It improves collaboration between teams, enforces security policies through RBAC, simplifies scheduling and job execution, and provides detailed logging and auditing capabilities. Ansible Tower also enhances scalability by enabling efficient management of large-scale automation deployments.
11. Ansible Tower can be integrated with various tools and systems through its REST API. Examples include integrating with ticketing systems for incident management, integrating with configuration management databases (CMDBs) for asset tracking, or integrating with external monitoring systems to trigger automation based on specific events. Such integrations enable automation workflows that span multiple systems and processes.
12. Best practices for managing and organizing Ansible Tower projects and resources include using version control for playbooks and job templates, adopting naming conventions for consistency, leveraging inventory groups and variables effectively, regularly updating Ansible Tower to benefit from new features and security patches, and following RBAC best practices to ensure secure access control.
13. Ansible Tower supports scaling and high availability through the use of load balancers and multiple Tower instances. Load balancing distributes incoming requests across multiple Tower nodes, improving performance and fault tolerance. Multiple Tower instances can be set up in an active-passive or active-active configuration, ensuring availability and redundancy.
14. Ansible Tower provides security measures to protect sensitive information like credentials. It offers credential types for securely storing secrets such as usernames, passwords, and SSH keys. These credentials can be encrypted and stored in the Ansible Tower database or integrated with external credential management systems. Ansible Tower also supports RBAC to enforce access controls and restrict sensitive information to authorized users.
15. Upgrading Ansible Tower to a new version involves following the upgrade documentation provided by Red Hat, the maintainers of Ansible Tower. Typically, the upgrade process involves backing up the existing configuration and database, installing the new version, and migrating the configuration and data to the new installation. It's essential to carefully review the upgrade documentation, test the upgrade in a non-production environment, and have a rollback plan in case of any issues during the upgrade process

KUBERNETES

------kubernetes setup

There are several deployment methods for Kubernetes, including manual installation, cloud-based solutions (such as Google Kubernetes Engine, Amazon Elastic Kubernetes Service, or Microsoft Azure Kubernetes Service), and third-party tools (such as kops or kubeadm). We are deploying through kubeadm.

Kubeadm is a popular tool for bootstrapping a Kubernetes cluster. Here's a step-by-step explanation of the process:

1. Prerequisites:
    - Set up a set of machines (physical or virtual) that will form the cluster. These machines should meet the minimum requirements for running Kubernetes.
    - Install Docker or a container runtime on each machine.
2. Initialize the master node:
    - On the machine you choose as the master node, run the `kubeadm init` command. This initializes the Kubernetes control plane and generates a token for joining worker nodes.
    - After the initialization process completes, the command will output instructions for configuring `kubectl`, the command-line tool used to interact with the cluster.
3. Set up the cluster network:
    - Choose a container network interface (CNI) plugin for your cluster. Popular choices include Calico, Flannel, and Weave. Install the CNI plugin on the master node using the provided instructions.
    - Apply the CNI configuration to the cluster using `kubectl apply`.
4. Join worker nodes:
    - On each worker node, run the `kubeadm join` command with the token generated during the master node initialization. This command establishes the worker node's connection with the master node.
    - After joining, the worker node will be registered with the master and become part of the cluster.
5. Verify the cluster:
    - On the master node, run `kubectl get nodes` to see the list of nodes in the cluster. You should see both the master node and the joined worker nodes.
6. Deploy a containerized application:
    - Use `kubectl` to create deployments, services, or other Kubernetes resources to deploy your application onto the cluster.
    - Monitor the deployment status and check the logs of your application using `kubectl` commands.

This is a simplified overview of the Kubernetes setup using kubeadm. Keep in mind that there are additional considerations for production deployments, such as securing the cluster, configuring storage options, and setting up high availability. The official Kubernetes documentation provides detailed guides on using kubeadm for various deployment scenarios, and I recommend referring to it for more in-depth information.

-------Components of the master node:

the kubernetes cluster contains master node and worker nodes

The Kubernetes master node is the control plane of a Kubernetes cluster and manages the cluster's state and configuration. It consists

1. API server: The API server is the main management and control point for the Kubernetes cluster. It exposes the Kubernetes API, which is used by other components to communicate with the control plane.
2. etcd: etcd is a key-value store that is used by Kubernetes to store cluster state, configuration, and other data. The API server communicates with etcd to retrieve and update cluster information.
3. Controller Manager: The Controller Manager is a daemon that runs the controllers in Kubernetes. The controllers are responsible for maintaining the desired state of the system, such as ensuring that the correct number of replicas of a pod is running.
4. Scheduler: The Scheduler is responsible for assigning pods to nodes in the cluster based on resource availability and It monitors the resource usage of each node

-------Components of the worker node node:

worker node is responsible for running application workloads and executing containers. The worker node is a node that is joined to the cluster and runs the necessary components to be managed by the Kubernetes master node. It consist of

1. Kubelet: The Kubelet is the primary agent that runs on each worker node and is responsible for managing the state of the containers running on that node. It receives instructions from the Kubernetes master node to start, stop, or update containers.
2. Container pods: The container pods is responsible for executing containers and managing their lifecycle. Common container runtimes used in Kubernetes clusters include Docker, containerd,
3. kube-proxy: The kube-proxy is responsible for managing network connectivity for the containers running on the node. It maintains network rules and routes to ensure that the containers can communicate with each other and with external services.

-------components of defanations

Here are some commonly used objects in Kubernetes:

1. Pod:
    - The basic building block in Kubernetes, representing a single instance of a running process.
    - A pod can consist of one or more containers that share the same network and storage resources.
    - Pods are typically ephemeral and are created, scheduled, and managed by controllers like Deployments or StatefulSets.
2. Deployment:
    - Manages a set of replicated pods, ensuring a desired number of identical pods are running at all times.
    - Provides functionality for rolling updates and rollbacks, allowing seamless application updates without downtime.
3. StatefulSet:
    - Manages a set of pods, similar to Deployments, but with the added capability of providing stable network identities and persistent storage for each pod.
    - Suitable for stateful applications like databases that require unique identities and stable storage.
4. Service:
    - Defines a stable network endpoint for accessing a set of pods, providing load balancing and service discovery.
    - Services can be exposed internally within the cluster or externally to the outside world.
5. Volume:
    - Provides a way to store and persist data within pods.
    - Volumes can be used to mount host paths, network storage, or cloud storage providers to pods.
6. Namespace:
    - Provides a way to logically divide and isolate resources within a cluster.
    - Namespaces allow multiple teams or projects to share the same cluster while maintaining separation and resource quotas.
7. ConfigMap and Secret:

  - ConfigMaps store configuration data that can be consumed by pods as environment variables or mounted as files.
  - Secrets are similar to ConfigMaps but are specifically designed for sensitive data, like passwords or API keys, and are base64 encoded and encrypted at rest.
8. Ingress:
  - Manages external access to services within the cluster, typically acting as a reverse proxy and providing routing, SSL termination, and load balancing.
9. DaemonSet:
  - Ensures that a specific pod runs on each node in the cluster.
  - Useful for deploying system daemons or agents that need to run on every node, such as log collectors or monitoring agents.
10. Job:
- Manages the execution of a single task or batch job in the cluster.
- Jobs create one or more pods to perform the task, and they ensure the task completes successfully before terminating the pods.
11. CronJob:
- Similar to a Job, but it allows you to schedule the execution of tasks at specified intervals using a cron-like syntax.
- CronJobs are useful for running recurring or periodic tasks, such as backups or data synchronization.
12. ReplicaSet:
- An older, lower-level object for managing and ensuring the availability of a specified number of pod replicas.
- Deployments and StatefulSets are higher-level abstractions built on top of ReplicaSets and are generally preferred for managing applications.
13. PersistentVolume and PersistentVolumeClaim:
- PersistentVolumes (PVs) are storage resources provisioned by administrators and made available to users in the cluster.
- PersistentVolumeClaims (PVCs) are requests made by users for storage resources from a PersistentVolume.
- PVCs bind to PVs, allowing pods to use persistent storage, such as network-attached storage or cloud-based storage.
14. Secret:
- Stores sensitive data, such as passwords, tokens, or certificates, securely within the cluster.
- Secrets can be used by pods or other Kubernetes resources as environment variables or mounted as files.
15. ConfigMap:
- Stores configuration data, such as application settings or environment variables, as key-value pairs.
- ConfigMaps can be used by pods or other Kubernetes resources as environment variables or mounted as files.
16. Namespace:
- Provides a logical separation and isolation of resources within a cluster.
- Namespaces are used to organize and control access to resources, allowing multiple teams or projects to use the same cluster while maintaining separation.

These are additional Kubernetes objects commonly used to define and manage various aspects of applications, storage, scheduling, and resource management within a cluste

------- name space

- **Create namespace:** **my-service.test.svc.cluster.local**

```
kubectl get nskubectl get all --all-namespaces
```

```
kubectl get svc -n kube-system
kubectl create ns prashanth
kubectl run nginx1 --image=nginx -n prashanth

vim pod1.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx12
  namespace: prashanth
```

```
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80

kubectl apply -f pod1.yaml
kubectl get pod -n prashanth
```

---What is Ingress?

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

An Ingress does not expose arbitrary ports or protocols. Exposing services other than HTTP and HTTPS to the internet typically uses a service of type Service.Type=NodePort or Service.Type=LoadBalancer.

```
# Create Ingress

vim ingress.yaml

kind: Ingress

metadata:

  name: -ingress

  annotations:

    nginx.ingress.kubernetes.io/use-regex: "true"

spec:

  ingressClassName: nginx

  rules:

  - host: hostname.groophy.in

    http:

      paths:

      - path: /login

        pathType: Prefix

        backend:

          service:

            name: my-app

            port:

              number: 8080

kubectl apply -f vproingress.yaml

# Update Path in ingress from /login to /

kubectl get ingress
```

-------mount volumes

Volumes: There are several types of Kubernetes volumes that can be used to store data:

- EmptyDir: This is a temporary volume that is created when a Pod is started and deleted when the Pod is terminated. It is used to store data that needs to be shared between containers in the same Pod.

2. HostPath: This type of volume mounts a directory from the host node's filesystem into the container. It is used to store data that needs to persist across container restarts and is only accessible to containers running on the same node.
3. ConfigMap: This type of volume provides a way to inject configuration data into a container. The data is stored in a Kubernetes ConfigMap object, which can be updated independently of the container.
4. Secret: This type of volume is similar to a ConfigMap volume but is used to store sensitive information such as passwords, access keys, and other secrets.
5. PersistentVolumeClaim (PVC): This type of volume provides a way to use a network-attached storage volume to store data. A PVC is a request for a storage volume that can be fulfilled by a PersistentVolume (PV) object. PVs can be created statically or dynamically using a storage class.

```
vim mysqlpod.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: dbpod
spec:
  containers:
  - image: mysql:5.7
    name: mysql
    volumeMounts:
    - mountPath: /var/lib/mysql
      name: dbvol
  volumes:
  - name: dbvol
    hostPath:
      # directory location on host
      path: /data
      # this field is optional
      type: DirectoryOrCreate
```

```
kubectl apply -f mysqlpod.yaml
kubectl get pod
kubectl describe pod dbpod
```

-------secrets and tains and tolerations

Secret data:

we can use the `Secret` object to store sensitive information such as passwords, access keys, and other secrets. `Secrets` are stored as `base64` encoded strings, which can be decoded by anyone with access to the Kubernetes cluster.

```
echo -n 'your_password' | base64
```

taints and tolerations:

Let's say you have a Kubernetes cluster with a group of nodes that have GPUs(graphic possessing unit). You can add a taint to these nodes that specifies that they require pods to have a certain label that indicates they need GPU resources. Pods that don't have this label won't be scheduled on these nodes, which can help prevent resource contention and ensure that the nodes are used efficiently.

To allow certain pods to be scheduled on these nodes, you can add a toleration to the pod specification that indicates that it can tolerate the GPU taint. This will allow the pod to be scheduled on the nodes with GPUs, even though it doesn't have the required label.

Taints and tolerations can be applied at the node level or the pod level, depending on your needs. They can be useful for ensuring that workloads are scheduled on the right nodes, based on the resources they require and the limitations of the nodes in the cluster.

-------kubernetes logs and debugging

In Kubernetes, you can access the logs of a container running in a Pod using the `kubectl logs` command. The `kubectl logs` command allows you to view the logs of a container in real-time

```
kubectl logs -f <pod_name> -c <container_name>

kubectl logs <pod_name>

kubectl logs <pod_name> --since=1h --until=30m
```

debugging:

1. Check the logs: Check the logs of your Kubernetes resources, including pods, deployments, and services. You can use the kubectl logs command to view the logs of a specific pod.
2. kubectl describe: Use the kubectl describe command to get more information about a specific resource, such as a pod or a deployment. This command provides a detailed overview of the resource and its current state.
3. Monitor resource usage: Use Kubernetes tools like Prometheus or Grafana to monitor resource usage, including CPU, memory, and network usage. This can help you identify performance issues and potential bottlenecks.
4. Check the network: Network issues can often be the source of problems in a Kubernetes cluster. Check the network configuration of your pods and services to ensure they are properly configured and communicating with each other.
5. Use debugging tools: Kubernetes provides various debugging tools, such as kubectl exec and kubectl port-forward, which allow you to access a pod's shell and port-forward to a service, respectively.
6. Check the Kubernetes API server: The Kubernetes API server is the central control plane component of a Kubernetes cluster. If you're experiencing issues, check the API server logs to see if there are any error messages or warnings.
7. Test in a development environment: It's always a good practice to test any changes or new deployments in a development environment before pushing them to production. This can help you identify and fix issues before they affect your users.

-------helm and helmcharts

Helm is a package manager for Kubernetes that simplifies the process of deploying and managing applications on a Kubernetes cluster. It allows you to define, install, and upgrade Kubernetes applications using a set of templates called "charts".

Helm charts are collections of files that describe a related set of Kubernetes resources, such as deployments, services, and config maps. These templates can be versioned, shared, and reused across different environments and clusters.

Here are some key features and benefits of using Helm and Helm charts:

1. Reusability: Helm charts can be used to define reusable components that can be used across different Kubernetes clusters and environments. This can help to streamline the deployment and management of applications, and avoid having to write the same YAML files multiple times.
2. Modularity: Helm charts can be composed of multiple smaller charts, which can be used to build more complex applications. This makes it easy to build and manage large, multi-component applications with Helm.
3. Versioning: Helm charts can be versioned, allowing you to track changes to your applications over time. This can help with managing application updates and rolling back changes if needed.
4. Community Support: Helm has a large and active community of contributors and users, which provides access to a wide range of third-party Helm charts for popular applications and services. This can save time and effort when deploying and managing complex applications.

-------kubernetes security

ChatGPT

Kubernetes security is crucial for ensuring the protection of your containerized applications and the underlying infrastructure. Here are some key considerations and best practices for securing your Kubernetes environment:

1. Secure cluster configuration:
    - Use secure network policies to control traffic between pods and namespaces.
    - Employ RBAC (Role-Based Access Control) to restrict access to Kubernetes resources based on user roles and permissions.
    - Enable and configure audit logging to monitor and track activities within the cluster.
    - Regularly update and patch both the Kubernetes components and the underlying operating system to address security vulnerabilities.
2. Secure container images and deployments:
    - Use trusted container images from reputable sources and regularly update them with security patches.
    - Scan container images for vulnerabilities using tools like Clair, Anchore, or Trivy.
    - Implement secure image pull policies to ensure only approved images are used.
    - Utilize resource limits and isolation mechanisms (such as PodSecurityPolicy or PodSecurityContext) to prevent resource abuse and container escapes.
3. Network security:
    - Isolate the Kubernetes control plane from public networks and limit access to authorized entities only.
    - Encrypt communication within the cluster using Transport Layer Security (TLS) for API server and etcd communication.
    - Utilize network policies and firewalls to control ingress and egress traffic to and between pods.
    - Consider implementing a network security solution such as a Service Mesh (e.g., Istio) to enforce fine-grained access control and secure communication between services.
4. Secure authentication and authorization:
    - Utilize strong authentication mechanisms, such as client certificates, tokens, or integrations with external identity providers (e.g., LDAP, OIDC).
    - Implement RBAC to enforce least privilege access and restrict user permissions based on roles.
    - Regularly review and audit user access privileges to ensure they align with the principle of least privilege.
5. Secrets management:
    - Avoid hardcoding sensitive information within the application code or configuration files.
    - Utilize Kubernetes Secrets to securely store and manage sensitive data, such as passwords, API keys, or TLS certificates.
    - Encrypt Secrets at rest and in transit, and limit access to Secrets based on RBAC policies.
6. Monitoring and logging:
    - Implement comprehensive monitoring and logging solutions to detect and respond to security incidents.
    - Collect and analyze logs from Kubernetes components, containers, and applications.
    - Utilize security-focused monitoring tools, such as Kubernetes-specific threat detection systems or log analysis platforms, to identify and respond to security events.
7. Ongoing security assessments:
    - Perform regular security assessments, vulnerability scans, and penetration testing of your Kubernetes infrastructure and applications.
    - Conduct security reviews of third-party dependencies, including container images and Kubernetes add-ons.
    - Stay updated with security best practices and monitor security advisories from the Kubernetes community and relevant vendors.

By following these best practices and continuously evaluating and improving the security posture of your Kubernetes environment, you can mitigate risks and ensure the secure operation of your containerized applications

-------Deployment and statefullsets

The main difference between a Deployment and a StatefulSet in Kubernetes lies in how they manage the lifecycle and identity of the pods they create:

1. Deployment:
    - Deployments are primarily used for managing stateless applications or microservices.
    - Deployments provide a way to declaratively define and manage replica sets and pods.
    - They create and manage pods that are identical and interchangeable, meaning they don't have unique identities.
    - Pods created by a deployment have temporary storage, and any data stored within them is not persisted beyond the pod's lifecycle.
    - Deployments support rolling updates and rollbacks, allowing you to update the application without downtime by gradually replacing pods.
    - They are suitable for horizontally scalable applications, where adding or removing pods can handle increased or decreased demand.
2. StatefulSet:
    - StatefulSets are designed for managing stateful applications that require stable network identities and persistent storage.
    - StatefulSets provide ordered and unique identities to each pod, allowing them to have stable network hostnames and persistent storage volumes.
    - Pods created by a StatefulSet are typically not interchangeable, as they maintain a consistent identity throughout their lifecycle.
    - Each pod in a StatefulSet has its own unique storage, such as a PersistentVolumeClaim, enabling data persistence.
    - StatefulSets maintain a sequential order when creating or scaling pods, allowing for ordered deployment or scaling operations.
    - They are suitable for applications that require stable network identities, such as databases, key-value stores, or stateful applications that rely on shared storage.

In summary, Deployments are ideal for stateless applications that can scale horizontally, while StatefulSets are used for stateful applications that require stable network identities and persistent storage. Deployments focus on managing identical and interchangeable pods, while StatefulSets handle unique and ordered pods with stable identities and storage.

Vpa- vertical pod autoscaler

VPA (Vertical Pod Autoscaler) is a Kubernetes feature that automatically adjusts the CPU and memory requests and limits of a container to match its actual resource usage. This can help to optimize the use of resources in a cluster and improve the performance and scalability of applications

-------expose application outside

To access an application deployed in a Kubernetes cluster from outside the cluster, you can use a combination of a Service and an Ingress resource. Here's an overview of the steps involved:

1. Deploy your application using a Deployment:
    - Create a Deployment manifest that describes your application, including the container image, ports, and any necessary configuration.
    - Apply the Deployment manifest to the cluster using the `kubectl apply` command.
2. Create a Service:
    - Create a Service manifest to define a stable endpoint for accessing your application.
    - The Service can be of type `ClusterIP` (default), `NodePort`, or `LoadBalancer`, depending on your requirements.
    - Specify the target port and optionally map it to a different port if needed.

    - Apply the Service manifest to the cluster using `kubectl apply`.
3. Configure an Ingress:
    - Create an Ingress manifest that defines the rules for routing external traffic to the Service.
    - Ingress rules typically include the host, path, and backend Service to which traffic should be forwarded.
    - If you are using a managed Kubernetes service, ensure that the necessary Ingress controller is installed and configured.
4. Expose the Ingress:
    - Depending on your cluster setup, expose the Ingress to the external world.
    - This can involve creating a DNS record that points to the Ingress controller's external IP address or configuring a load balancer.
5. Access the application:
    - Once the Ingress and necessary DNS or load balancer configurations are in place, you can access your application using the defined host and path.
    - Access the application using a web browser or any HTTP client by visiting the specified URL.

Note that the exact steps may vary depending on your Kubernetes distribution, networking setup, and ingress controller implementation. Additionally, it's essential to ensure that your firewall rules, security groups, and network policies allow incoming traffic to reach the cluster.

Consult the documentation or specific guides for your Kubernetes distribution and ingress controller for detailed instructions on configuring external access to your applications.

-------cluster ip service

The ClusterIP service type in Kubernetes is used for providing access to a set of pods within the cluster. It creates a stable internal IP address (cluster IP) that other components within the cluster can use to communicate with the service. Here are some key uses and benefits of the ClusterIP service:

1. Service Discovery:
    - ClusterIP services allow other pods or services within the cluster to discover and communicate with the set of pods associated with the service.
    - Other components can access the service using the service name and cluster IP address.
2. Load Balancing:
    - When multiple pods are associated with a ClusterIP service, the service automatically load balances incoming requests across the pods.
    - This ensures that the workload is distributed evenly, improving performance and scalability.
3. Internal Communication:
    - ClusterIP services are typically used for internal communication between different components of an application running within the cluster.
    - Applications can communicate with each other by sending requests to the ClusterIP service's IP address and port.
4. Service-to-Service Communication:
    - By using ClusterIP services, applications and services within the cluster can communicate with each other using a reliable and stable internal IP address.
    - This allows for decoupled and scalable architecture, where services can be easily scaled or replaced without affecting the consuming applications.
5. Security:
    - ClusterIP services provide an additional layer of security by abstracting the individual pod IP addresses.

- Instead of directly exposing pod IP addresses, which may change due to scaling or pod rescheduling, clients can securely communicate with the ClusterIP service.

6. Efficiency:
    - ClusterIP services optimize network traffic within the cluster by keeping internal communication local to the cluster's network fabric.
    - Communication between pods using ClusterIP services doesn't need to traverse the cluster boundary, reducing network latency and improving overall efficiency.

-------copy config file to pod

For example, let's say you have a config file named `my-config.yaml` located in your current working directory, and you want to copy it into a pod named `my-pod` in the namespace `my-namespace`, placing it at the path `/path/to/config.yaml` inside the container. You would use the following command:

```
kubectl cp my-config.yaml my-namespace/my-pod:/path/to/config.yaml
```

-------High avaialabality of pod level

Do you know high availability in pod level?

ChatGPT

In Kubernetes, high availability at the pod level can be achieved by ensuring that multiple replicas of a pod are running concurrently across multiple nodes in the cluster. This helps to maintain application availability even if individual pods or nodes experience failures. Here are some approaches to achieving high availability at the pod level:

1. Replication Controllers (Deprecated):
    - Replication Controllers (RCs) were an earlier Kubernetes object used for ensuring a specified number of pod replicas are running at all times.
    - RCs have been deprecated in favor of Deployments, which provide more advanced features and rolling updates.
2. Deployments:
    - Deployments are the recommended approach for managing pod replicas and achieving high availability in Kubernetes.
    - Deployments allow you to define the desired number of replicas for a pod and manage the lifecycle of those replicas.
    - They provide features like rolling updates, scaling, and automatic reconciliation to maintain the desired state.
3. ReplicaSets:
    - ReplicaSets are lower-level objects that can be used for managing pod replicas, although Deployments are now preferred for most use cases.
    - ReplicaSets allow you to specify the desired number of replicas and ensure that the specified number is running at all times.
4. StatefulSets:
    - StatefulSets are used for managing stateful applications where each pod has a unique identity and requires stable network identifiers and persistent storage.
    - StatefulSets provide high availability by ensuring that pods are rescheduled on the same node after failure, maintaining their identity and associated data.
5. DaemonSets (for specific use cases):

- While DaemonSets are not primarily used for achieving high availability, they ensure that a specific pod runs on every node, which can provide redundancy and fault tolerance.
- DaemonSets are useful for deploying system daemons or agents that need to be present on every node in the cluster.

By using these different objects and approaches in Kubernetes, you can configure and manage the desired number of pod replicas, handle pod failures, and maintain high availability for your applications. The choice of the appropriate approach depends on the specific requirements of your application and its tolerance for failure.

------ deploy pod in particular node

Is it possible to decide to deploy pod in particular node?

ChatGPT

Yes, it is possible to influence the scheduling of pods and specify which node they should be deployed on in Kubernetes. Here are a few methods to achieve this:

1. Node Affinity:
    - Node affinity allows you to define rules or preferences for pod scheduling based on node attributes, such as labels or node names.
    - You can use `nodeSelector` or more advanced techniques like `nodeAffinity` and `podAffinity` to ensure pods are scheduled on specific nodes based on their characteristics or relationships with other pods.
2. Node Selector:
    - Node selector is a simple way to schedule pods on specific nodes based on labels assigned to the nodes.
    - By specifying a node selector in the pod's specification, you can target nodes with matching labels for deployment.
3. Node Name:
    - You can explicitly specify the name of the node where a pod should be deployed by setting the `nodeName` field in the pod's specification.
    - However, using `nodeName` directly can limit the flexibility and dynamic scheduling capabilities of Kubernetes.
4. Taints and Tolerations:
    - Taints are used to mark nodes with certain restrictions or preferences, preventing pods from being scheduled on those nodes unless they have corresponding tolerations.
    - By adding tolerations to pods, you can allow them to be scheduled on specific tainted nodes.

-------port numbers

Kubernetes API server- 6443
Etcd – 2379 to 3380
Kubelet API - 10250
Kube-scheduler- 10259
Kube control manager 10257
Node port services 30000-32767

-------Questions

Back to first page

- Use case of init container?

  Init container is short lived container, it starts and does some command execution and then it will be dead. When it is dead main container will start .

- Suppose 2 namespaces are there, dev and test . service in the dev namespace want to access the service in the test, How we can do that?
- **my-service.test.svc.cluster.local**

- how do you trouble shoot pod in Docker ?
- how to add worker node to Kubernetes cluster?

- how to configure VPA?

- What is taints and tolerations?

- How do you setup the Kubernetes?
  Kops, Kubeadm, Minikube.
- Replica set?
- Namespaces?

1.Ssl termination on Kubernetes cluster not inside the pods

2.Create CPU load inside the pod without CPU pinning and with cpu pinning.

3.PostgreSQL database as a stateful set of pod in Kubernetes

4. Setup HA K8s cluster with External etcd and stacked etcd and backup Of etcd

5.

python scripting.

We are using python for automation tasks.creating a file with extension of .py with various strings ,variables, variable length arguments ,conditions printing and commenting purpose ,break and continue statements,builtin method as per requirements. Iam using python scripting in Jenkins file and in ansible playbook, for Jenkins file I need to make it simple as much as possible so if Jenkins file contains more logic syntax iam just execute through adhoc commands through python script. this scripts can be used to build code, run tests, and deploy applications to various environments.even in ansible we use some scripts to automate tasks such as package installation, system configuration.

```
stage('Execute Python Script') {

   steps {

      sh 'python myscript.py' or we use adhoc command like sh 'python -c "print(\' just give me chance
like!\')"'

   }

} or

stage('Execute Python Script') {
  steps {
    python {
      script {
        println "Running Python script”
      } }}}
```

IN ansible we use like Executing the Python script using the "shell" module or script module: like - name: Execute Python Script ; shell or script: python my_script.py or we can use the custom module and pass in the values for "num1" and "num2". The "register" keyword stores the output of the module in the "result" variable, which can then be printed using the "debug" module.

```
- name: Call Custom Module
  custom_module:
    num1: 3
    num2: 4
  register: result
- debug:
    var: result.stdout
```

Python scripting plays a significant role in the field of DevOps, where it is commonly used for automation, infrastructure management, and deployment processes. Here are some key areas where Python scripting is widely utilized in the DevOps domain:

1. **Infrastructure Provisioning and Management:** Python is frequently employed to automate the creation and management of infrastructure resources. Tools like Ansible, Terraform, and AWS SDKs (Boto3) offer Python bindings, enabling the provisioning and configuration of cloud infrastructure, virtual machines, containers, networking, and more.
2. **Configuration Management:** Python allows DevOps teams to manage and configure systems efficiently. Tools like Puppet and Chef use Python for creating configuration files, managing package installations, and maintaining system states across multiple servers.
3. **Continuous Integration and Continuous Deployment (CI/CD):** Python scripting is widely used in CI/CD pipelines to automate the build, test, and deployment processes. Popular CI/CD platforms such as Jenkins and GitLab CI/CD provide native support for running Python scripts to perform tasks like building applications, running tests, and deploying artifacts.
4. **Monitoring and Alerting:** Python offers various libraries and frameworks for monitoring and alerting systems. Tools like Prometheus, Nagios, and Zabbix often use Python for writing custom plugins, integrating with APIs, and processing monitoring data.
5. **Log Analysis and Visualization:** Python's data processing capabilities make it an excellent choice for log analysis and visualization. Libraries such as Pandas, Matplotlib, and Seaborn enable DevOps professionals to extract insights from logs, generate reports, and create visualizations for effective troubleshooting and system analysis.
6. **Containerization and Orchestration:** Python is widely used in containerization technologies like Docker and container orchestration platforms like Kubernetes. Python-based libraries like Docker SDK and Kubernetes Client provide APIs to manage containers, deploy applications, and interact with the container ecosystem.
7. **Deployment and Release Management:** Python scripts are often utilized to automate application deployments and release management processes. From versioning to deploying artifacts to various

> environments, Python can be used to build deployment pipelines and streamline the release process.

-------yaml.file

1.I have one POD and inside 2 containers are running one is Nginx and another one is wordpress So, how can access these 2 containers from the Browser with IP address?

docker run -d --name nginx -p 80:80 nginx

docker run -d --name wordpress -p 8080:80 wordpress

To access the two containers from the browser with an IP address, you will need to expose the containers to the host network and map the container ports to the host ports. Here are the steps:

- Get the IP address of your host machine. You can use the command `ifconfig` or `ipconfig` depending on your operating system.
- Check the port number on which Nginx and WordPress are listening. For example, if Nginx is listening on port 80 and WordPress is listening on port 8080, you will need to map the container ports 80 and 8080 to the host ports.
- Use the following command to start the containers with the correct port mappings:

cssCopy code

```
          --name         80 80                                     8080 80
```

This will start the Nginx container and map port 80 of the container to port 80 of the host. Similarly, it will start the WordPress container and map port 80 of the container to port 8080 of the host.

- Open a web browser and enter the IP address of your host machine followed by the port number for the WordPress container, like this: `http://<host-ip-address>:8080/`.
- To access the Nginx container, you can use the IP address and port number of the host machine, like this: `http://<host-ip-address>/`.

You should now be able to access the WordPress and Nginx containers from your web browser using the IP address of your host machine.

2.What is node affinity and pod affinity?

Node affinity and Pod affinity are two features in Kubernetes that allow you to specify rules for how pods are scheduled on nodes.

Node affinity allows you to control which nodes your pods are scheduled on based on labels assigned to the nodes. You can use node affinity to ensure that pods are scheduled on nodes that meet specific requirements such as having certain hardware capabilities or being in a specific geographical region. For example, you can use node affinity to schedule pods on nodes with GPUs if your workload requires GPU acceleration.

Pod affinity, on the other hand, allows you to control which nodes your pods are scheduled on based on labels assigned to other pods running on the same node. With pod affinity, you can ensure that pods that are related to each other are co-located on the same node. This can improve performance by reducing network latency between pods and increase reliability by ensuring that dependent pods are always available together.

Both Node affinity and Pod affinity are defined using Kubernetes selectors and are expressed as rules that either "match" or "do not match" node or pod labels. These rules are used by the Kubernetes scheduler to determine which nodes to schedule pods on.

Overall, Node affinity and Pod affinity are powerful features in Kubernetes that can help you optimize your workload placement and improve performance and reliability.

3. If I have multiple containers running inside a pod, and I want to wait for a specific container to start before starting another one .explain how do you do that

In Kubernetes, you can use a readiness probe to specify when a container is ready to serve requests. If you want to wait for a specific container to start before starting another one in the same pod, you can configure the readiness probe for that container, and then configure the other containers to depend on the readiness of that container using a startup probe.

A readiness probe is a type of health check that determines whether a container is ready to serve requests. A readiness probe can be configured to check for specific conditions, such as the availability of a network endpoint or the presence of a specific file. If the conditions are met, the container is considered ready. If the conditions are not met, the container is not ready, and Kubernetes will not send traffic to it.

A startup probe is a type of health check that determines whether a container has started successfully. A startup probe can be used to delay the startup of dependent containers until a specific container has started successfully. If the startup probe for a container fails, Kubernetes will terminate the container and restart it.

4.How should you connect an app pod with a database pod?

To connect an app pod with a database pod in Kubernetes, there are a few different options depending on the specifics of your setup. Here are some general approaches:

- Use environment variables: You can pass the database connection details (such as the database hostname, port, username, and password) to your app container as environment variables. You can create a Kubernetes ConfigMap or Secret object to store the connection details and then inject them into your app container as environment variables. This approach is simple and easy to manage, but it may not be secure enough for sensitive data.
- Use a Kubernetes Service: You can create a Kubernetes Service object to expose your database pod as a network endpoint within your Kubernetes cluster. You can then configure your app container to connect to the database using the Service's DNS name and port. This approach provides a more robust and secure way of connecting to your database, as the Service can be configured to use encryption and authentication.
- Use a sidecar container: You can run a sidecar container in the same pod as your app container that handles the database connection. The sidecar container can be responsible for setting up the database connection and then exposing it to the app container via a shared volume or network interface. This approach can be useful if you need to use a specialized tool or library for connecting to your database.
- Use StatefulSets: If you are running a stateful application that requires a stable network identity, such as a database, you can use a Kubernetes StatefulSet object to manage your database pods. StatefulSets provide a way to assign stable network identities (such as DNS names

5.If you have a pod that is using a ConfigMap which you updated, and you want the container to be updated with those changes, what should you do?

When you update a ConfigMap that is being used by a pod in Kubernetes, the changes will not automatically be reflected in the container. In order to update the container with the updated ConfigMap, you have a few options:

- Delete and recreate the pod: One option is to delete the existing pod and recreate it. This will ensure that the new ConfigMap is used by the pod. However, this approach can cause downtime for your application.
- Use a rolling update: Another option is to use a rolling update. This involves updating the pod one at a time, ensuring that there is always a running instance of the application. You can use the **`kubectl apply`** command with the **`--force`** flag to force a rolling update of the pod.
- Use a Deployment: If you are using a Deployment to manage your pod, you can update the ConfigMap in the Deployment spec and then perform a rolling update using **`kubectl apply`**.

Regardless of which option you choose, it's important to ensure that your application is able to handle any changes to the ConfigMap. You may need to restart or reload your application to ensure that it is using the updated configuration.

Deployment.yaml

```
apiVersion:
apps/v1
       kind: Deployment
       metadata:
        name: vproapp
        labels:
         app: vproapp
       spec:
        replicas: 1
        selector:
         matchLabels:
          app: vproapp
        template:
         metadata:
          labels:
           app: vproapp
         spec:
          containers:
          - name: vproapp
           image: imranvisualpath/freshtomapp:V7
           ports:
           - name: vproapp-port
            containerPort: 8080
          initContainers:
          - name: init-mydb
           image: busybox
           command: ['sh', '-c', 'until nslookup vprodb; do echo waiting for mydb; sleep 2; done;']
          - name: init-memcache
           image: busybox
           command: ['sh', '-c', 'until nslookup vprocache01; do echo waiting for mydb; sleep 2; done;']
```

Secret.yaml

```
apiVersion:
v1
              kind: Secret
              metadata:
                name: app-secret
              type: Opaque
              data:
                db-pass: dnByb2RicGFzcw==
                rmq-pass: Z3Vlc3Q=
```

Service.yaml

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

Jfrog

JFrog is a software company that provides DevOps solutions for managing and distributing software artifacts. JFrog offers a suite of tools that help developers to manage their software development lifecycle. The main products of JFrog are:

1. JFrog Artifactory: Artifactory is a universal artifact repository manager that supports multiple software package formats such as Maven, npm, Docker, RubyGems, and more. It provides a central repository for storing and managing all your binary artifacts, which can be easily searched and retrieved.
2. JFrog Xray: Xray is a universal binary analysis and vulnerability detection tool that integrates with Artifactory. It automatically scans your artifacts and identifies vulnerabilities, license violations, and quality issues across your entire software stack.
3. JFrog Pipelines: Pipelines is an end-to-end DevOps automation platform that integrates with Artifactory and Xray. It allows you to define and execute complex build, test, and deployment pipelines in a simple and intuitive way.
4. JFrog Mission Control: Mission Control is a centralized platform for managing and monitoring JFrog Artifactory and JFrog Xray instances. It provides a single pane of glass for tracking the health, usage, and performance of your DevOps toolchain.

```
pipeline {
  agent any
  stages {
    stage('Build') {
      steps {
        // Build your application
        sh './maven build'
      }
    }
    stage('Archive') {
      steps {
        // Archive your built artifacts
        archiveArtifacts artifacts: 'build/libs/*.jar', fingerprint: true
      }
    }
    stage('Publish') {
      steps {
        // Push the artifacts to JFrog Artifactory
        rtUpload (
          serverId: 'jfrog', // The ID of the JFrog Artifactory server in Jenkins
          spec: '''{
            "files": [
              {
                "pattern": "build/libs/*.jar",
                "target": "my-repo-local/"
              }
```

Docker file to pull artifact

```
FROM openjdk:8-jdk-alpine
RUN apk --no-cache add curl
ARG ARTIFACTORY_URL=https://my.artifactory.server
ARG ARTIFACTORY_REPO=my-repo
ARG ARTIFACT_PATH=com/example/my-app/1.0.0/my-app-1.0.0.jar
RUN curl -fL $ARTIFACTORY_URL/artifactory/$ARTIFACTORY_REPO/$ARTIFACT_PATH -o my-app.jar
ENTRYPOINT ["java","-jar","/my-app.jar"]
```

nagios

Nagios can be a useful tool in DevOps, as it provides a comprehensive monitoring and alerting system for IT infrastructure.:

teams can monitor their IT infrastructure in real-time and be alerted immediately when issues arise. This can help to minimize downtime and improve the overall reliability and performance of the system. Additionally, Nagios can be integrated with other DevOps tools, such as Jenkins or Ansible, to automate and streamline the monitoring and response process.

. Nagios Core: This is the main component of Nagios and is responsible for scheduling checks, processing results, and triggering alerts. Nagios Core provides a flexible architecture that allows administrators to extend its functionality through plugins.

2. Nagios Plugins: Nagios plugins are scripts or programs that are used to perform specific checks on various aspects of the IT infrastructure, such as the status of a service or the availability of a network device. Nagios plugins can be written in any language, as long as they conform to a specific interface.
3. Nagios Web Interface: The Nagios web interface is a user-friendly web-based interface that allows administrators to monitor the status of the IT environment, view alerts, and acknowledge or resolve problems.
4. Nagios Event Broker: The Nagios Event Broker is an API that allows third-party developers to integrate with Nagios and extend its functionality. This component can be used to build custom applications that integrate with Nagios and provide additional monitoring capabilities.
5. Nagios Notifications: Nagios provides a variety of notification methods, such as email, SMS, and chat platforms, to notify administrators when issues arise. These notifications can be customized based on the severity of the issue and the preferences of the recipient.

Zira ticketing tool

Zira is a ticketing tool that can be useful in DevOps for managing software development and deployment processes. Zira is designed to simplify communication and collaboration between teams, track issues and progress, and improve overall productivity.

Some of the key features of Zira that make it useful in a DevOps context include:

1. Agile project management: Zira provides a flexible and customizable project management system that can be adapted to different development methodologies, such as Scrum or Kanban. This allows teams to plan and track their work, assign tasks, and monitor progress in real-time.
2. Integration with DevOps tools: Zira can be integrated with a wide range of DevOps tools, such as GitHub, Jira, Slack, and Trello. This enables teams to streamline their workflows and automate their processes, reducing the risk of errors and improving overall efficiency.
3. Customizable workflows: Zira provides a drag-and-drop interface that allows users to create custom workflows and automate their processes. This can help teams to standardize their processes, reduce manual work, and ensure that all team members follow the same steps.
4. Collaboration and communication: Zira provides a centralized platform for teams to collaborate and communicate, with features such as comments, tags, and mentions. This can help to improve communication, reduce misunderstandings, and ensure that everyone is on the same page.
5. Reporting and analytics: Zira provides a variety of reporting and analytics tools that allow teams to track progress, identify bottlenecks, and measure performance. This can help teams to continuously improve their processes and deliver high-quality software more quickly.

Overall, Zira can be a valuable tool for DevOps teams, providing a centralized platform for collaboration, task management, and process automation. By using Zira, teams can work more efficiently, reduce errors, and deliver higher quality software in less time

water and agile methodology

The Waterfall methodology follows a sequential approach to software development, where each phase of the SDLC must be completed before moving to the next. It is a more traditional and rigid approach to software development, where there is less room for changes or modifications once a phase is completed. This makes it less compatible with DevOps, which emphasizes continuous feedback, collaboration, and delivery.

On the other hand, Agile methodology is a more flexible approach to software development, emphasizing iterative and incremental development. Agile emphasizes frequent feedback and collaboration, which aligns with the principles of DevOps. This methodology is more compatible with the DevOps approach, as it allows for continuous integration, testing, and delivery.

In DevOps, the emphasis is on building a culture of collaboration, communication, and automation across the entire software development lifecycle, from planning to deployment. Both methodologies can be used in DevOps, but Agile is more commonly used due to its iterative approach, which allows for continuous feedback, testing, and delivery. This makes it easier to integrate DevOps practices, such as continuous integration and continuous delivery, which are essential to successful DevOps implementation.

json and yaml

JSON (JavaScript Object Notation) and YAML (YAML Ain't Markup Language) are both commonly used data serialization formats, but they have some differences in syntax and structure.

Here are a few differences between JSON and YAML:

1. **Syntax**: JSON uses a syntax that is based on a subset of the JavaScript programming language, while YAML has its own syntax that is designed to be more human-readable and user-friendly. **YAML** uses whitespace indentation to indicate hierarchy, while JSON uses curly braces and square brackets.
2. **Data types**: Both JSON and YAML support a range of data types, including strings, numbers, boolean values, arrays, and objects. However, **YAML** also supports additional data types such as dates, times, and binary data.

Comments: YAML supports comments, which can be used to provide additional context and documentation for the data, while JSON does not.

Compatibility: JSON is widely used in web applications and APIs, and is supported by most programming languages. **YAML** is also widely used, particularly in configuration files and scripts, but is not as widely supported as JSON.

Error handling: JSON has a strict syntax, so if there are any errors in the data, the parsing will fail. YAML, on the other hand, is more forgiving, and will attempt to parse the data even if there are errors. This can make it easier to work with in some cases, but can also make it more difficult to catch errors.

In summary, JSON and YAML are both widely used data serialization formats, but they have some differences in syntax, data types, and error handling. JSON is commonly used in web applications and APIs, while **YAML** is often used in configuration files and scripts.

JSON (JavaScript Object Notation) is a lightweight data interchange format that is commonly used for transmitting and storing data. It is based on a subset of the JavaScript programming language, but it can be used with any programming language that supports JSON parsing.

JSON is designed to be easy for humans to read and write, as well as for machines to parse and generate. It consists of a set of key-value pairs, where the key is a string and the value can be a string, number, boolean, array, or another JSON object.

YAML (short for "YAML Ain't Markup Language") is a human-readable data serialization format. It is often used for configuration files, but it can also be used for data exchange between programming languages. YAML is not tied to any particular programming language or technology, and is supported by a wide range of programming languages.

YAML uses indentation and whitespace to indicate structure and hierarchy, and has a simple syntax that is easy to read and write. It is designed to be easy for humans to read and write, and can also be parsed and generated by machines.

victoriametrics

ChatGPT

VictoriaMetrics is a high-performance, cost-effective, and scalable open-source time series database and monitoring solution. It is designed to handle large volumes of time series data efficiently and provides powerful querying capabilities. Here are some key features and information about VictoriaMetrics:

1. Time series database: VictoriaMetrics is primarily focused on storing and analyzing time series data. It is optimized for high ingestion rates and efficient storage, making it suitable for handling metrics, logs, and other time series data generated by applications, systems, or IoT devices.
2. High performance: VictoriaMetrics is known for its high-performance capabilities. It can handle millions of data points per second and provides fast query responses even on large data sets. It achieves this by utilizing efficient storage formats and indexing techniques.
3. Cost-effective storage: The storage efficiency of VictoriaMetrics is one of its strengths. It uses a compressed and efficient data format that reduces storage requirements, resulting in cost savings when dealing with large amounts of time series data. Additionally, it supports data retention policies to automatically manage the storage of historical data.
4. Query language and API: VictoriaMetrics provides its own query language called PromQL, which is compatible with Prometheus. PromQL allows you to perform advanced queries, aggregations, and filtering on time series data. In addition to PromQL, VictoriaMetrics supports various APIs, including Prometheus API, Graphite API, and InfluxDB Line Protocol, making it compatible with popular monitoring and observability tools.
5. Scalability and clustering: VictoriaMetrics is designed to scale horizontally, allowing you to handle increasing workloads by adding more nodes to the cluster. It supports sharding and replication for data distribution and high availability. Clustering enables seamless scaling of storage and query capacity as your data volume grows.
6. Integration with other tools: VictoriaMetrics can be integrated with popular monitoring and observability tools, such as Grafana, Prometheus, and Alertmanager. This allows you to leverage VictoriaMetrics as a storage backend for these tools and benefit from its efficient storage and query capabilities.
7. Open-source and commercial versions: VictoriaMetrics is available as an open-source project with permissive licensing, allowing you to use, modify, and distribute it freely. In addition, there is also a commercial version called VictoriaMetrics Cluster, which provides additional features, such as distributed querying, high availability, and advanced security options.

VictoriaMetrics is a powerful and efficient solution for storing, querying, and analyzing time series data. Whether you need to monitor system metrics, application performance, or IoT sensor data, VictoriaMetrics can handle the scale and provide fast and cost-effective storage and querying capabilities

argocd

ChatGPT

Argo CD is an open-source continuous delivery (CD) tool that focuses on managing and automating deployments of applications and infrastructure to Kubernetes clusters. It provides a declarative approach to application deployment, allowing you to define your desired state in version-controlled configuration files and letting Argo CD handle the deployment process. Here are some key features and information about Argo CD:

1. GitOps methodology: Argo CD follows the GitOps approach to application deployment. It relies on a Git repository as the single source of truth for defining the desired state of your applications. The repository contains declarative YAML files that specify the application configuration, including deployment manifests, service definitions, and any other Kubernetes resources required.
2. Continuous delivery and deployment automation: Argo CD automates the process of deploying applications to Kubernetes clusters based on changes in the Git repository. It continuously monitors the repository for updates and reconciles the desired state with the current state of the target cluster. If there are any differences, Argo CD will automatically initiate the necessary actions to bring the cluster into the desired state.
3. Declarative application definitions: With Argo CD, you define your application configuration in declarative YAML files. These files describe the desired state of your application, including the container images, environment variables, resource limits, and other deployment-specific parameters. By using declarative definitions, you can easily manage and version control your application configurations.
4. Application lifecycle management: Argo CD provides functionalities for managing the full lifecycle of applications. It handles application deployment, scaling, rollback, and canary releases. It also supports automated health checks and hooks for pre and post-deployment actions. Argo CD ensures that your applications are consistently and reliably deployed across multiple environments.
5. Web-based user interface (UI) and CLI: Argo CD offers a user-friendly web-based UI that provides visibility into your deployments, application status, and synchronization details. It allows you to view the differences between desired and current states and perform manual synchronization if needed. Additionally, Argo CD provides a command-line interface (CLI) for automation and integration into CI/CD pipelines.
6. Integration with other tools: Argo CD integrates well with other popular tools and frameworks in the Kubernetes ecosystem. It can be used alongside continuous integration (CI) tools like Jenkins or GitLab CI/CD for end-to-end automation. It also integrates with Kubernetes package managers like Helm and Kustomize, allowing you to manage and deploy complex applications with ease.
7. Security and RBAC: Argo CD provides robust security features, including authentication, authorization, and role-based access control (RBAC). It supports various authentication methods like token-based authentication, OAuth, and OpenID Connect. RBAC allows you to define fine-grained access control policies, enabling secure and controlled access to different resources within Argo CD.

Argo CD simplifies the deployment and management of applications in Kubernetes clusters by leveraging the GitOps methodology and providing automation capabilities. It promotes infrastructure as code practices, ensures consistency across environments, and improves collaboration among teams. Whether you are deploying simple or complex applications, Argo CD can be a valuable tool in your CI/CD workflow.

Prometheus and grafana

ChatGPT

Prometheus: Prometheus is an open-source monitoring and alerting system designed for monitoring highly dynamic and distributed environments, particularly suited for containerized applications running on Kubernetes. It collects metrics from various targets such as application instances, servers, or services and stores them in a time-series database. Key features of Prometheus include:

1. Data collection: Prometheus scrapes metrics from instrumented targets using a pull model. It supports multiple integrations and exporters to collect metrics from different systems, including applications, databases, and infrastructure components.
2. Data storage: Metrics collected by Prometheus are stored in a time-series database. The data is organized into samples with a timestamp, metric name, and associated labels, allowing for efficient querying and analysis.

3. Querying and alerting: Prometheus provides a powerful query language called PromQL, which allows you to query and aggregate metrics for monitoring and analysis. It also supports alerting rules based on metric thresholds or complex conditions, enabling proactive notifications when specific conditions are met.
4. Service discovery and dynamic configurations: Prometheus has built-in support for service discovery mechanisms such as Kubernetes, Consul, and DNS. It automatically discovers new targets and adjusts its configurations accordingly, simplifying the monitoring setup for dynamic environments.

Grafana: Grafana is an open-source data visualization and dashboarding tool that works seamlessly with various data sources, including Prometheus. It allows you to create interactive and customizable dashboards to visualize and analyze your metrics and logs. Key features of Grafana include:

1. Visualization: Grafana offers a wide range of visualization options, including graphs, charts, tables, and heatmaps. It provides various visualization settings, such as color schemes, thresholds, and annotations, to help you present your data effectively.
2. Dashboard creation: Grafana allows you to create and organize dashboards containing multiple panels, each representing different metrics or data sources. You can configure panels with custom queries, time ranges, and visualization options to create meaningful and informative dashboards.
3. Alerting and notifications: Grafana supports alerting based on metric thresholds or complex conditions. You can set up alert rules and define notification channels like email, Slack, or PagerDuty to receive alerts when specific conditions are met.
4. Templating and annotations: Grafana provides templating features that allow you to create dynamic dashboards by defining variables that can be used in queries, panels, and annotations. Annotations help add contextual information to your dashboards, such as events, incidents, or deployments.
5. Community and ecosystem: Grafana has a large and active community that contributes to its ecosystem. It offers a wide range of plugins and integrations, allowing you to connect to various data sources, including Prometheus, Elasticsearch, InfluxDB, and more. These plugins extend the functionality and data source options of Grafana.

Prometheus and Grafana work well together in a monitoring and observability stack. Prometheus collects metrics from various sources, stores them in its time-series database, and provides querying and alerting capabilities. Grafana, on the other hand, connects to Prometheus (among other data sources) to visualize and create dashboards, enabling you to gain insights and monitor the health and performance of your systems and applicati

EKS

ChatGPT

Amazon Elastic Kubernetes Service (Amazon EKS) is a managed container orchestration service provided by Amazon Web Services (AWS). It simplifies the deployment, management, and scaling of containerized applications using Kubernetes. Here are some key features and information about Amazon EKS:

1. Kubernetes as a managed service: Amazon EKS eliminates the need for manual setup and management of Kubernetes clusters. It provides a fully managed Kubernetes control plane, which is responsible for managing the underlying infrastructure and control plane components, such as the API server, scheduler, and etcd.
2. Scalability and high availability: Amazon EKS is designed to be highly scalable and available. It automatically manages the scaling of the control plane based on the size and requirements of your cluster. It also provides multi-Availability Zone (AZ) deployment options, ensuring high availability and fault tolerance.
3. Integration with AWS services: Amazon EKS integrates seamlessly with other AWS services, providing a unified experience for deploying and managing containerized applications. For example, you can leverage services like Amazon Elastic Container Registry (ECR) for container image storage and Amazon Elastic File System (EFS) for shared storage volumes in your EKS clusters.
4. Security and compliance: Amazon EKS incorporates various security features to protect your clusters and workloads. It integrates with AWS Identity and Access Management (IAM) for fine-grained access control. EKS also supports encryption of data in transit and at rest, and integrates with AWS Key Management Service (KMS) for managing encryption keys.
5. Container networking: EKS leverages Amazon VPC (Virtual Private Cloud) networking to provide networking capabilities for your Kubernetes clusters. It allows you to configure networking policies, implement security groups, and establish communication between pods and services in a secure and isolated manner.

6. Ecosystem and tooling: Amazon EKS benefits from the vast Kubernetes ecosystem, allowing you to leverage popular tools, frameworks, and plugins. This includes tools for logging, monitoring, and observability, such as Prometheus and Grafana. Additionally, EKS supports Kubernetes-native integrations and add-ons, simplifying the deployment of applications and management of resources.
7. Cost optimization: Amazon EKS offers cost optimization features to help you manage your resources efficiently. It provides features like cluster autoscaling, which automatically adjusts the number of worker nodes based on workload demands. EKS also offers integration with AWS Fargate, allowing you to run containers without managing the underlying infrastructure, which can help optimize costs for specific use cases.

Amazon EKS enables you to leverage the power of Kubernetes for deploying and managing containerized applications with ease. It abstracts away the complexities of managing Kubernetes clusters, allowing you to focus on application development and innovation. Whether you are running microservices, batch processing, or data-intensive workloads, Amazon EKS provides a scalable and robust platform to run your containers on AWS.

Agile, scrum, kanban

ChatGPT

Agile, Scrum, and Kanban are popular methodologies used in software development and project management. While all three approaches share common principles and values, they have distinct characteristics and practices. Here's an overview of each:

Agile: Agile is a flexible and iterative approach to software development that emphasizes collaboration, adaptability, and continuous improvement. It values customer satisfaction and encourages cross-functional teams to work together to deliver high-quality software. Key principles of Agile include responding to change, delivering working software frequently, and promoting face-to-face communication. Agile methodologies, such as Scrum and Kanban, are derived from the Agile philosophy.

Scrum: Scrum is an Agile framework that focuses on delivering value through iterative development and regular feedback. It divides work into time-boxed iterations called "sprints." Scrum teams consist of a Product Owner, Scrum Master, and the development team. Key elements of Scrum include:

1. Product Backlog: A prioritized list of requirements or user stories that define the work to be done.
2. Sprint Planning: The team selects a set of items from the Product Backlog to work on during the upcoming sprint.
3. Daily Scrum: A brief daily meeting where team members synchronize their work, discuss progress, and identify any impediments.
4. Sprint Review: A meeting at the end of each sprint where the team demonstrates the completed work to stakeholders and gathers feedback.
5. Sprint Retrospective: A reflection session at the end of each sprint to review what went well, what could be improved, and make adjustments for the next sprint.

Scrum provides a framework for managing complex projects, fostering collaboration, and promoting transparency.

Kanban: Kanban is a visual management method that helps teams optimize workflow and increase efficiency. It emphasizes continuous delivery and the reduction of work in progress (WIP). Kanban uses a Kanban board with columns representing different stages of work (e.g., to-do, in progress, done). Work items, represented as cards, move across the board as they progress through the workflow. Key aspects of Kanban include:

1. WIP Limit: Controlling the number of work items in progress to avoid overloading the team.
2. Pull System: New work items are pulled into the workflow as capacity becomes available, based on the team's WIP limit.
3. Continuous Improvement: Kanban encourages teams to continuously analyze and optimize their workflow to identify bottlenecks and improve efficiency.

Kanban provides visibility into work status, helps manage flow, and enables teams to respond to changing priorities effectively.

Both Scrum and Kanban can be used within an Agile context, depending on the project's requirements and team dynamics. They promote collaboration, flexibility, and iterative development, while Scrum introduces defined roles, ceremonies, and time-boxed iterations. Kanban, on the other hand, emphasizes visual management and optimizing flow. The choice between Scrum and Kanban depends on factors such as project complexity, team structure, and customer engagement.

Bash scripting

Bash scripting is a powerful tool in the DevOps world, as it can automate many tasks, from deployment and configuration to testing and monitoring. Here are some of the ways that bash scripting can be used in DevOps:

1. Automation: Bash scripts can be used to automate tasks such as deploying code, provisioning infrastructure, configuring servers, and running tests.
2. Continuous Integration and Deployment (CI/CD): Bash scripts can be used to automate the build, test, and deployment of software, making it easier to manage the entire CI/CD pipeline.
3. Configuration management: Bash scripts can be used to manage configurations for servers and applications, making it easier to maintain consistent environments across multiple machines.
4. Monitoring: Bash scripts can be used to monitor system resources, log files, and other metrics, allowing DevOps teams to proactively identify and resolve issues before they become problems.
5. Reporting: Bash scripts can be used to generate reports and visualizations of system metrics, making it easier to track performance and identify trends.

Overall, bash scripting is a versatile tool that can help DevOps teams automate many of the routine tasks that are part of the software development lifecycle. By using bash scripts, DevOps teams can save time, reduce errors, and improve the overall quality of their softwar

```
#!/bin/bash

# Variable Declaration

#PACKAGE="httpd wget unzip"

#SVC="httpd"

URL='https://www.tooplate.com/zip-templates/2098_health.zip'

ART_NAME='2098_health'

TEMPDIR="/tmp/webfiles"

yum --help &> /dev/null

if [ $? -eq 0 ]

then

  # Set Variables for CentOS

  PACKAGE="httpd wget unzip"

  SVC="httpd"

 echo "Running Setup on CentOS"

  # Installing Dependencies

  echo "########################################"

  echo "Installing packages."

  echo "########################################"

  sudo yum install $PACKAGE -y > /dev/null

  echo
```

```
# Start & Enable Service
echo "########################################"
echo "Start & Enable HTTPD Service"
echo "########################################"
sudo systemctl start $SVC
sudo systemctl enable $SVC
echo

# Creating Temp Directory
echo "########################################"
echo "Starting Artifact Deployment"
echo "########################################"
mkdir -p $TEMPDIR
cd $TEMPDIR
echo

wget $URL > /dev/null
unzip $ART_NAME.zip > /dev/null
sudo cp -r $ART_NAME/* /var/www/html/
echo

# Bounce Service
echo "########################################"
echo "Restarting HTTPD service"
echo "########################################"
systemctl restart $SVC
echo

# Clean Up
echo "########################################"
echo "Removing Temporary Files"
echo "########################################"
rm -rf $TEMPDIR
```

```
  echo

  sudo systemctl status $SVC
  ls /var/www/html/

else
   # Set Variables for Ubuntu
  PACKAGE="apache2 wget unzip"
  SVC="apache2"

  echo "Running Setup on CentOS"
  # Installing Dependencies
  echo "########################################"
  echo "Installing packages."
  echo "########################################"
  sudo apt update
  sudo apt install $PACKAGE -y > /dev/null
  echo

  # Start & Enable Service
  echo "########################################"
  echo "Start & Enable HTTPD Service"
  echo "########################################"
  sudo systemctl start $SVC
  sudo systemctl enable $SVC
  echo

  # Creating Temp Directory
  echo "########################################"
  echo "Starting Artifact Deployment"
  echo "########################################"
  mkdir -p $TEMPDIR
  cd $TEMPDIR
  echo
  wget $URL > /dev/null
  unzip $ART_NAME.zip > /dev/null
  sudo cp -r $ART_NAME/* /var/www/html/
  echo

  # Bounce Service
  echo "########################################"
  echo "Restarting HTTPD service"
  echo "########################################"
  systemctl restart $SVC
  echo

  # Clean Up
  echo "########################################"
  echo "Removing Temporary Files"
  echo "########################################"
  rm -rf $TEMPDIR
  echo

  sudo systemctl status $SVC
  ls /var/www/html/
fi
```

DA-1048	Explore the Mirroring feature in Istio
DA-1034	Gather information about the architechture of jaeger with istio - components, database, working.
DA-1030	Tracing with Sample application with 3 key areas
DA-1029	Understand concepts - Tracing vs Logging
DA-1016	Explore and R&D of Istio Service mesh

DA-1010	Setup EFK on k8s Baremetal
DA-1003	Documentation on EFK
DA-984	Installation of EFK on k8s - Lakhan
DA-978	Explore Nginx LB as sctp
DA-973	Setup LoxiLB as SCTP LB
DA-965	expose pods with service type Load Balancer and use loxilb load balancer
DA-964	setup a pod with replica set 2 or more
DA-963	Setup a k8s cluster with Metal LB as external load balancer
DA-961	write yaml files for nginxpod/sample pod with Class QOS
DA-953	CPU cores Pinning in Kubernetes
DA-948	Route HTTP2.0 to HTTP1.1 incoming traffic
DA-915	documentation on EFK
DA-914	Setup ingress and check sticky sessions on k8s
DA-903	ElasticSearch logs issue - Lakhan
DA-896	resolve issue fluent efk Lakhan
DA-881	Setup FluentD to monitor logs of docker containers (on single VM)
DA-873	Helm install issue - Garvit
DA-869	Review VM's inventory
DA-860	Deploy 8 UE veriphye 23.1.3.0 with new CPU core assignment

DA-852	Run ue8 containers
DA-836	Test VeriPHYE 23.1.3.0 on 2 docker containers on 65 machine
DA-833	Run 2 docker container on 67
DA-832	Run 2 docker container on 66
DA-829	Documentation for RF enabled release
DA-795	Jenkins job
DA-766	Test UL and DL on 8 UE running as Docker container.
DA-739	Git Trainings session Prep
DA-641	Create Docker image for RANP Veriphye 23.1.1.3
DA-636	Test Docker images with tag 23.1.1.3
DA-628	KT on Vphy Project UE+UEV and RANV containerization activity from Samridhi
DA-626	Set Up Cost Monitoring
DA-603	Docker entry point sh script for UEMP and UE
DA-581	Azure Resource Handling
DA-506	Setup k8s with only unblocking required ports in ubuntu
DA-504	DISK, CPU, RAM, Huge pages are available or not
DA-503	Interfaces are bind or not
DA-502	DPDK installed
DA-501	OS check

DA-500	Bash scrips for pre check
DA-473	Setup Azure account
DA-439	Veriphye Track-Falguni
DA-438	Learning on DevOps Required Skill
DA-410	Explore the EFK set-up
DA-401	write a script to diameter protocol enable in a pod
DA-398	Jan Updates 23