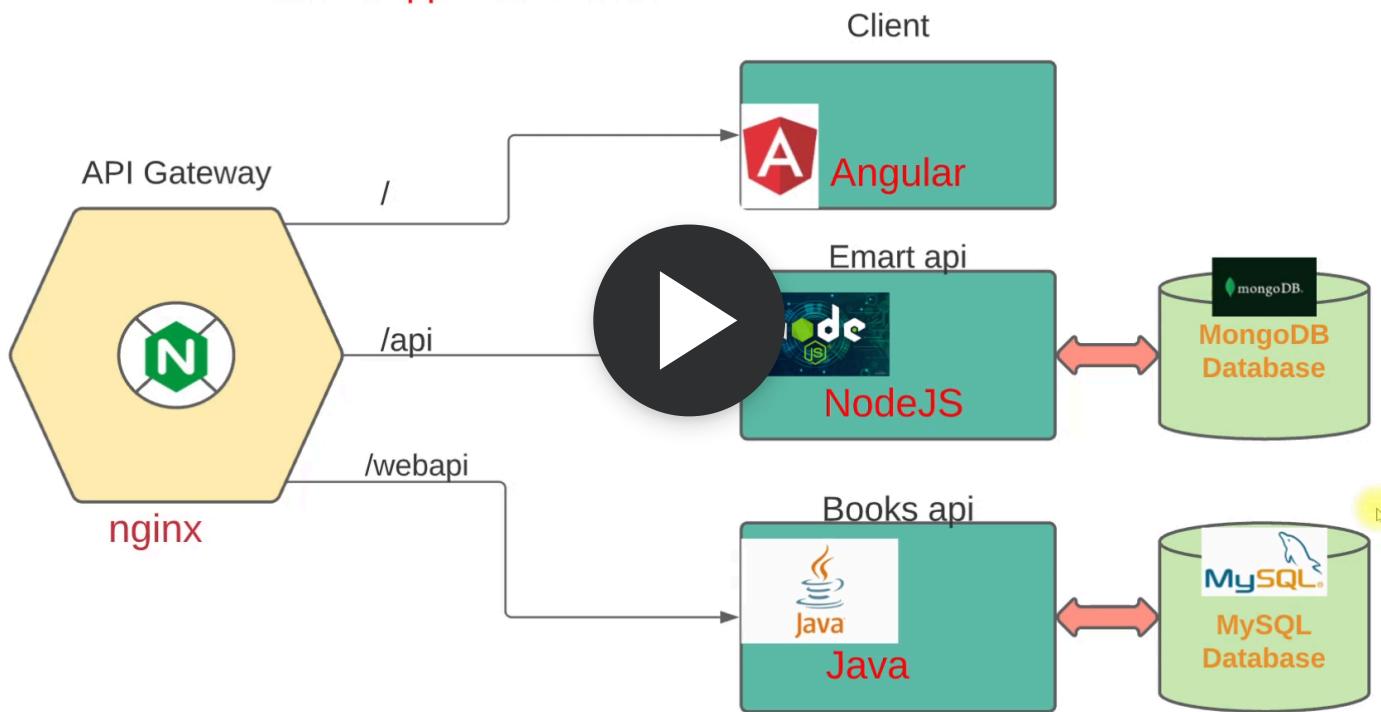




## EMart App Architecture



Udemy



Course content

Overview

Q&amp;A

Notes

Announcements

Reviews

Learning tools

Create a new note at 0:14



All lectures

Sort by oldest

0:36

**1. Introduction** 1. About this course

infrastructure , cloud computing and coding are the main components of the devops

1:33

**1. Introduction** 1. About this course

Ansible is used for configuration management

1:46

**1. Introduction** 1. About this course

Docker is used for containerization

1:52

**1. Introduction** 1. About this course



Container Orchestration by using kubernetes

1:57

**1. Introduction** 1. About this course



Cloud automation using terraform

3:00

**1. Introduction** 3. What is DevOps?



Software Development Process

1) Requirement Analysis

- \* Product Features
- \* Usage
- \* Users
- \* User requirement
- \* Market State

3:15

**1. Introduction** 3. What is DevOps?



2) Planning

- \* What do we want
- \* Defines the cost and resource and risk taken to build the application

3:23

**1. Introduction** 3. What is DevOps?



3) Design Architects : These are the road map for the developers

- \* Architects design the software application on how it should look like and the feature modeling by taking inputs from the previous phase and give the design document to the developers, so developers develop based on that design.

3:38

## 1. Introduction 3. What is DevOps?



4) Development by Developers

- \* Write software code based on the design model

3:56

## 1. Introduction 3. What is DevOps?



5) Software Testing: Software will be pushed to production only if all the issues are resolved that are found by the testers in the development.

4:12

## 1. Introduction 3. What is DevOps?



6) Deployment: Operation team works in deploying the product for the user. Software is pushed to the production environment. So the end user can use the product.

It is the duty of system admin and operation team to run the application without downtime.

4:38

## 1. Introduction 3. What is DevOps?



7) Maintenance: System health , performance , uptime.

- \* Balance the regular changes that are happening in production and uptime.

4:53

## 1. Introduction 3. What is DevOps?



Total overview of the SDLC

- 1) Requirement Analysis
- 2) Planning
- 3) Design
- 4) Development
- 5) Testing
- 6) Deployment and maintenance

5:02

## 1. Introduction 3. What is DevOps?



Architects design the software , developers develop the software , testers test the software and operations team deploy and maintain the software , this is part of SDLC.

5:40

## 1. Introduction 3. What is DevOps?



Models in SDLC

- \* Waterfall
- \* Spiral
- \* BIG BANG
- \* Agile

Model selection is based on the cost , risk and time taken to reach the destination

6:23

## 1. Introduction 3. What is DevOps?



Waterfall model of SDLC

- \* Waterfall model is a one way model of development
- \* No working software will be produced , untill last of the cycle , and at last we get the final product
- \* Customer may not know all his requirements initially , so this leads to the issue for the speed of the application as the cycle takes very long time to generate a final product.
- \* It actually takes months to generate a product for the user

7:03

## 1. Introduction 3. What is DevOps?



Agile SDLC : Breaks the whole tasks to design the products into so many sub parts and work on them in multiple sprint cycles of 2 to 4 weeks, where the product is built for the end user so fast , and add feature by feature to the application according to the requirements of the customer. After each sprint the product will be verified with the client and take feedback from the end user. And new features if required.

7:20

## 1. Introduction 3. What is DevOps?



Advantage of Agile

7:48

## 1. Introduction 3. What is DevOps?



After a developer has developed a software , he will request to publish the software into the servers to operations team , and prior to that , the QA (Quality Analysis) team will test the software.

8:23

## 1. Introduction 3. What is DevOps?



When the code was pushed from the Developer side to the Operations side , The ops team can't work the application on the server , In general Ops team in this SDLC they have more work to do , because they need to change the code and deploy the application on the servers and many more code changes will be coming so they need to do deployment for those.

8:53

## 1. Introduction 3. What is DevOps?



Operation team gets regular deployment requests with no clear instruction , with those they need to change the code and do deployment regularly which fails as usual and they have pre occupied work like maintaining the system uptime and ITIL Process driven. So operation teams feel the whole burden on them and there will be misalignment between developers and operation teams and in result customers will not get a good quality product. And the manager could not give the updated product presentation to the customer.

9:38

## 1. Introduction 3. What is DevOps?



Difference between Development and operations team

Development is agile where they have regular and quick changes.

Operations team is ITIL which ensures the stability of the application.

Due to miscommunication between development and operations team , operations team complain about the not proper instructions and developers complain about the slow in deployment by operation team.

Dev is agile and Operations team is still waterfall , so devops was introduced.

If a person having both the operations concept like stability and dev concepts like agile is an devops engineer.

Automation of every task during the development lead to overhead on anyone , and the process will become faster. For example automation of the code build, code testing , software testing , infra changes , deployments leads to an efficient progress towards agile product development.

10:05

## 1. Introduction 3. What is DevOps?



Dev vs Ops Arguments example which result in bad customer feedback, the reason of this is dev is agile and ops is waterfall

12:00

## 1. Introduction 3. What is DevOps?



Automation will be helpful to decrease the manual intervention in the tasks , So the devops role will be simplified.

12:36

## 1. Introduction 3. What is DevOps?



Basically the faster team consists of following pattern , and now dev and ops are together where we have automation tools for every team and integration of automation tools of every team is called integration. Basically the devops team mean he must have idea on the following team structure and idea on the automation tools which helps to build the application and an integration tool which build the application.

A basic team for a project must consists of

- 1) Developers
- 2) Testers
- 3) Sys Admin
- 4) DB Admin
- 5) Build and Release team

Every stage will develop their own automation tools and integrate every tool and develop a integration model for the development.

14:19

## 1. Introduction 3. What is DevOps?



Devops lifecycle

14:30

## 1. Introduction 3. What is DevOps?



DevOps automation is the addition of technology that performs tasks with reduced human assistance to processes that **facilitate feedback loops between operations and development teams so that iterative updates can be deployed faster to applications in production**. This automation leads to the faster interaction between the stages and the integrated model generates the end application faster to the customer.

0:08

## 1. Introduction 5. What is Continuous Integration?



Continuous integration is an automated process in devops , which generates software and its features quickly and efficiently. In general developers , write so many lines of code for an application and it is a practice to store the code in a version control system like GitHub. Everyday developers pull and push the code from and to the repository to be the code in up to date. And this code commit occurs continuously and this code is moved to build server where the code is being build , tested and evaluated and generate the artifact for the application or called as a software this software is stored in a repository in form of the archives which are generated from build process. Based on the programming language , the artifacts packaged in a specific format for example war and jar for the java , DLL and MSI and EXE for the windows , or ZIP and TAR

2:15

## 1. Introduction 5. What is Continuous Integration?



After deploying the artifacts on the server the software testers test the artifacts and if there is no error found then we push the artifact to the production. And the main problem is if we develop a code for the application for three weeks and send it to the deployment and there the software tester tests the application and found errors , it doesn't move to the production , so now the developers need to bugfix the code and they have to manage the new development. So actually the problem here is the code is merged into repository but not getting integrated on the server. So the solution for this would be , we need to build and test the code after every commit , so we do not get the errors accumulated finally. But it is not that easy to do that , because there will be multiple commits in a day , so it is difficult to perform build and test for every commit manually.

4:27

## 1. Introduction 5. What is Continuous Integration?



So we need to automate this process of building and testing the code , so if there is an error , we can revert back to the developer then and there to change the code of the application and send the response to the developer as a notification. So he changes the code and send to the deployment where we have build and test of the code as automated , so as the human intervention is less , it is less costly and time effective and there would not be any errors further , regarding the code in that particular commit.

5:17

## 1. Introduction 5. What is Continuous Integration?



Continuous Integration Process (All the following stages done by the automation)

- 1) Code (Developer writes the code)
- 2) Fetch (Fetch the latest code before deployment request)
- 3) Build (Build the edited code by the developer)
- 4) Test (Test the edited code and compatibility of the new code with the application)
- 5) Notify (Notify to the developer , if there is no error)
- 6) Feedback (Send feedback to developer , if the build and test succeeded and proceeded for production)

Goal of CI is to detect the errors at the early stages of the development

6:21

## 1. Introduction 5. What is Continuous Integration?



Some of the IDE's integrated with Version control system

6:32

## 1. Introduction 5. What is Continuous Integration?



Version Control system examples

- 1) GIT
- 2) SVN
- 3) TFS
- 4) PERFORCE

6:45

## 1. Introduction 5. What is Continuous Integration?



Build Tools ( Changes according to the language we use )

- 1) MAVEN , ANT , GRADLE
- 2) MSBUILD , VISUAL BUILD
- 3) IBM URBAN CODE
- 4) MAKE
- 5) GRUNT

6:56

## 1. Introduction 5. What is Continuous Integration?



Software Repository Tools (promoting collaborative use by offering remote access to code modules and software packages)

- 1) SONATYPE NEXUS
- 2) JFROG ARTIFACTORY
- 3) ARCHIVA
- 4) CLOUDSMITH PACKAGE
- 5) GRUNT

7:06

## 1. Introduction 5. What is Continuous Integration?



### CONTINUOUS INTEGRATION TOOLS

- 1) JENKINS
- 2) CIRCLECI
- 3) TEAMCITY
- 4) BAMBOO CI
- 5) CRUISE CONTROL

7:23

## 1. Introduction 5. What is Continuous Integration?



### STAGES OF CONTINUOUS INTEGRATION IN DEVOPS CYCLE ( SEE THE IMAGE )

- 1) CODE
- 2) CODE BUILD & TEST
- 3) CODE ANALYSIS
- 4) ARTIFACT REPO

0:05

## 1. Introduction 6. What is Continuous Delivery?



- 1) Continuous delivery is an automated process which deliver code changes to the server (quickly and efficiently)

Continuous Integration is an automated process of code build and test if everything is good the artifact is store into a software repository.

### Continuous delivery Process

- \* We are in the stage where development team generated an artifact which is successfully build and tested and send for the deployment on the server to the operations team , so at that moment the operations team may face issue of failed deployment for the artifact which built and tested successfully.
- \* Deployment is not only delivering the artifact to the server , we need to perform server provisioning , install the dependencies , load configuration changes , network and artifact deployment on to the server and more . And after manual deployment operations team send the artifact to the production team where we will be having the software testing. Which involves high human intervention , so to reduce the human intervention and increase the speed of the deployment we are having this continuous delivery .

Every step in the deployment should be automated

Ansible , puppet , chef ( System automation )

Terraform , CFormation ( Cloud Infrastructure automation )

Jenkins , Octopus Deploy ( CI/CD automation )

HELM Charts

Code Deploy

And similarly Software testing should also be automated

For example , Functional , load , DB , security , performance or any other test should be automated.

Ops team will write automation code for deployment and Testers team will write automation code for testing and sync with the developers source code.

Developers make Integrated CI (Build , Test)

Ops Team make automated deployment

QA team make automate testing

Ops team , QA team , Developers make Continuous Delivery finally.

Continuous Integration + Ops Team + QA team => Continuous Delivery

What is Devops?

Philosophy of combining dev and ops team together at culture , practice and tools level

What is Continuous Integration?

Developers regularly merge their code to the central repository , after which automated build and tests runs

What is Continuous Delivery?

Code changes are automatically prepared for release to production

0:57

## 2. Prerequisites Info & Setup 8. Tools Prerequisite Information



Source code or repository refers to this GitHub repository.

Tools that are used:

- \* Oracle VM Virtual Box
- \* Git Bash
- \* Vagrant
- \* Chocolatey for Windows / Brew for Linux

Other tools:

- \* JDK8
- \* Maven
- \* INTELLIJ
- \* Sublime text editor
- \* AWS CLIS

SIGN UP

- \* GITHUB
- \* GODADDY DOMAIN PURCHASE

\* DOCKERHUB

\* SONARCLUB

IN AWS

- \* We try to get a free tier account
- \* IAM with MFA
- \* Billing Alarm
- \* Certificate Setup

0:45

## 3. VM Setup 17. What is Virtualization



Now a days , one computer can run , multiple os's also denoted as computers at a time , but once upon a time it is not the case , before virtualization to run an app/ services we need servers , so for that we keep servers in datacenters. So in general before virtualization , one service will be done by the one server, services couldn't share the servers , so sometimes servers are overprovisioned as we don't know how much to allocate in max. So, as they are allocated much resources , the server resources are under utilized , for that we need huge capital expenditure and operation team employment expenditure.

3:04

### 3. VM Setup 17. What is Virtualization



When virtualization came , we can allow one computer to run multiple OS at a time, we can partition the physical resource and use as a virtual resource. In a physical resource , using virtualization we can create a virtual machine of any OS. Virtual machines are isolated from each other and each VM have it's own OS. Server Virtualization is the most common virtualization.

4:45

### 3. VM Setup 17. What is Virtualization



In general we will have our own hardware and on top of it we have a software called Hypervisor which is common and responsible for VM creation , where on top of hypervisor , we have independent VM's which have it's own OS and on top of it , utilizing that OS , we are able to run the applications.

5:40

### 3. VM Setup 17. What is Virtualization



Snapshot is a way of taking backup of the virtual machine , VM is a set of files .

Hypervisor is a tool or software which is used to create VM's , Hypervisor enables virtualization. Hypervisor are of two types

\* Type 1: Bare metal Hypervisor , it runs as a base OS , it is only used in production , Eg: VMware esxi , Xen Hypervisor.

\* Type 2: This Hypervisor runs as the software and this is used for only learn and testing , because this doesn't perform operations as core. Eg: Oracle virtualbox , Vmware server

7:11

### 3. VM Setup 17. What is Virtualization



Check the image

8:13

### 3. VM Setup 17. What is Virtualization



Hyper V of microsoft is of Type 1 Hypervisor but not Type 2 Hypervisor

0:49

### 3. VM Setup 18. Introduction



Ways to install Centos and Ubuntu VM

- \* Manually
- \* Automatically with Vagrant

2:41

### 3. VM Setup 18. Introduction



Tools Required:

- 1) Chocolatey for windows
- 2) Brew for Macos
- 3) Oracle VM Virtualbox
- 4) Git Bash / Terminal for Mac
- 5) Putty/Terminal for Mac
- 6) ISO => Centos 7 & Ubuntu Server 18
- 7) Vagrant (Automate the virtual machine setup)

3:47

### 3. VM Setup 19. VM-Manually(Windows & MacOS Intel chip)



[Important Video] VM manually installing in Windows and Mac

5:34

### 3. VM Setup 19. VM-Manually(Windows & MacOS Intel chip)



When we are allocating hard disk memory to the virtual machine we better allocate the memory dynamically , because if the size is fixed then the memory may or may not be used , so dynamic allocation saves memory.

11:19

### 3. VM Setup 19. VM-Manually(Windows & MacOS Intel chip)



virtual machine provides 4 network adapters , these can be either Wifi adapter or ethernet adapter or anything , our system connects to Wifi using Wifi adapter , so similarly as we want our VM to be connected to wifi we need such type of network adapters. nat is a private network adapter , so we are creating another adapter of bridged network which helps to give ip of adapter so we can connect to that.

1:56

### 3. VM Setup 20. VM-Automatically(Windows & MacOS Intel chip)



We need automation tools for VM , because to decrease installation overhead , and enable auto configuration. Vagrant is an automation tool where VM being setup through Images which are called as vagrant boxes. These images or boxes are available in vagrant cloud, we can manage virtual machines with a file called Vagrant file. We are having vagrant commands to manage VM's , We are having a provision in vagrant to execute VM commands & scripts. In Vagrant file we are having VM's configuration.

3:08

### 3. VM Setup 20. VM-Automatically(Windows & MacOS Intel chip)



Vagrant Architecture , we declare configuration of virtual machine in vagrant file , in which cloud image is being declared , and if we can't find that image locally , vagrant communicate with the vagrant cloud for the image and then after getting image, it contacts hypervisor , it may be oracle virtual box / VMWare etc and create multiple virtual machines as per required. We are having multiple vagrant commands such as vagrant up / vagrant halt / vagrant reload / vagrant destroy / vagrant ssh etc.

4:11

### 3. VM Setup 20. VM-Automatically(Windows & MacOS Intel chip)



To set up vagrant , we need following tools

- 1) Virtual Technology enabled in BIOS
- 2) Vagrant tool
- 3) Hypervisor like Oracle Virtual box
- 4) Command line tool ( Git Bash , Cygwin , CMD Prompt )

5:28

### 3. VM Setup 20. VM-Automatically(Windows & MacOS Intel chip)



Vagrant is a command line tool , for which we won't have any CLI. In Git bash we can execute linux commands in windows machine. Using vagrant init command we are going to give vagrant box name or the vagrant image name or the VM image name to download the vagrant file. In vagrant we called vagrant images as vagrant boxes , using which we

download the necessary VM's. After executing vagrant init we get Vagrant file as the output in the current directory. And next "vagrant up" command helps us to execute the vagrant file and create a VM for us.

10:38

### 3. VM Setup 20. VM-Automatically(Windows & MacOS Intel chip)



If we got any error like "VBox hardening" this is due to mainly antivirus, so disable it. And don't connect to any VPN because it doesn't work under a proxy network. After completing vagrant up , if we didn't get any error , we must use the command vagrant ssh , it makes us to enter the VM and the Git Bash will work inside the Virtual machine. And to off the VM's we need to use "vagrant halt". If we want to delete our VM. We must use vagrant destroy , so the VM will get destroy or removed. To reboot any particular VM , we need to use vagrant reload. We can use "history" command to get track of commands that we have used in GIT Bash.

1:41

### 4. Linux 23. Introduction to Linux



We are going to know

- 1) Introduction to Linux
- 2) Basic CLI commands
- 3) Understanding files in Linux
- 4) Filters & Redirection
- 5) Users & Group
- 6) Sudo
- 7) Software management
- 8) Services & Processes
- 9) Good to know commands
- 10) Server management

2:07

### 4. Linux 23. Introduction to Linux



Open Source software is a software with source code that anyone can inspect , modify and enhance. All open source software's are not free. Anyone can contribute to the open source project. Linux is an open source software management tool. Linux is a Unix like Kernel which is open source. Linux Kernel + GNU utilities is a complete open source , Unix like operating system. Linus Torvalds created Linux Kernel similar to Unix. In Linux everything is considered as files. So many such small single purpose programs together

help to solve complex operations. In dev ops we use captive user interface. Here configuration data stored in a text file.

8:35

## 4. Linux 23. Introduction to Linux



Architecture of Linux: Hardware is the core and out of the hardware we have the kernel software and around that we have shell where we can execute the Linux commands and around of shell we will have many applications

11:04

## 4. Linux 23. Introduction to Linux



Linux distributions : Most secured is Red Hat Enterprise Linux , and next we are having Ubuntu Server which is open source and it face negative in-terms of security and we are having centos where , RHEL and Centos both are same. And we have SUSE Enterprise Linux. These are some popular Linux distributions.

In IT industry , we mainly have two types of Linux distributions

1)RPM based(.rpm): RHEL , Centos , Oracle Linux , Amazon Linux

2)Debian based(.deb): Ubuntu Server , Kali Linux

They differ between type of packaging they have.

For server related work RPM based is better and for devops or for automation purpose we use Debian based. For debian we have "dpkg" and for rpm we have "rpm".

17:26

## 4. Linux 23. Introduction to Linux



Admin related directory : /root , /home/username

User executable commands : /bin , /usr/bin , /usr/local/bin

System executable commands ( system admin commands only can be executed by the root user ) : /sbin , /usr/sbin , /usr/local/sbin

Other mountpoints ( external storage device , or external files located in this loc ) : /media , /mnt

Configuration data present in : /etc

Temporary files located in : /tmp

Kernal and Boot loader: /boot

Server Data ( Web Server , Website files located in , mysql data) : /var , /srv

System Information : /proc , /sys

1:28

**4. Linux** 24. Commands and File systems

pwd - present working directory

ls - list current directory

cat /etc/os-release - Tell current VM name

sudo -i => Switching to root user using vagrant (vagrant have access to the root)

If we are able to see the # then it denotes that we are in the root user

If we are seeing ~ then it denotes that we are in the home directory

clear - command used to clear the terminal

cd / - we go to the top directory of linux or unix operating system

In boot folder of the top directory we have the boot loader and the kernal folder.

opt - optional directory

/bin - '/' denotes it starts from the absolute path

user executable programs are stored inside the bin

system executable commands are stored inside the sbin

etc is the configuration directory

'uptime' command is used to get from how much time we are running the particular VM instance.

'free -m' command shows the size of the RAM size

0:36

**4. Linux** 25. More Commands ( mkdir, cp, mv, touch etc)

From normal linux terminal to the vagrant user , we need to use the command 'vagrant up' , and from vagrant user to the root user , we need to use the sudo -i.

0:49

**4. Linux** 25. More Commands ( mkdir, cp, mv, touch etc)

File based commands

1) mkdir - make the new directory

2) touch - create a new file

touch devops{1..10}.txt => Creates multiple files in the same directory , naming as the following pattern.

3) cp filename folder/filename - Copy a file from the source to the destination

- 4) cp -r folder\_name dest\_folder/ - copy a directory from source to the destination folder
- 5) ls -l <path\_to\_list> - list the directory at the particular given location
- 6) cp --help - opens the documentation for the command that we asked
- 7) mv filename folder/ - move a file from the source to the destination
- 8) mv filename1.txt filename2.txt - if both files are need to be stored in same location then this works as the rename. Even if it is of different location , it copies the file to the different location by renaming it.
- 9) mv \*.txt dest\_folder/ - move all text files of current directory to the destination folder of mentioned location
- 10) rm filename1.txt - removes the file
- 11) rm -r folder\_name - removes the directory

11:39

#### 4. Linux 25. More Commands ( mkdir, cp, mv, touch etc)



- 1) rm -r \* - removes everything from current directory
- 2) rm -rf \* - removes everything from current directory forcefully
- 3) history - provide the details of all the commands that are executed in this session

0:44

#### 4. Linux 26. Vim editor



vim editor is an editor which is enhanced version of vi , to install vim "sudo yum install vim -y".

vim firstfile.txt => creates a new file , if the file not exists , if it exists it simply open the file.

There are three modes in vim file , command mode , insert mode , extended command mode ( done using (: colon.

'i' is used to insert in vi file , after using 'i' we can insert whatever we want in the file and then we need to press "ESC" to go into extended command mode, and then :wq , it mean ':' used to enter into extended command mode , 'w' is the write , 'q' is quit , :wq totally mean , enter into extended command mode and write save and quit the file , so that we update the necessary changes into the file. which we can see using vi or cat command.

'o' is used to go to the next line , in insert mode.

ESC ':q' is used to quit the file without saving it.

ESC ':q!' is used to quit the file without saving it forcefully.

6:29

## 4. Linux 26. Vim editor



Whenever we open the file using vim , we go into command mode , and later if we want to edit anything we go into the insert mode , and then if we want to save anything or quit from the file , we go to the extended command mode and use :wq or :q command to come out of the vi editor.

ESC ':se nu' => used to show line numbers in vi editor

If I want to go to the last line of the file using vim editor , use G

If I want to go to the first line of the file using vim editor , use gg

If I want to copy a particular line of a file using vim editor , use yy

If I want to paste something into vim editor , use p (Paste Below) P (Paste Above)

If I want to copy 4 lines from a line then use , 4yy

If I want to undo a line then use 'u'

If I want to delete/cut 5 next lines then use '5dd' , p is used to paste

If I want to search some word , then use '/<word-we-need>' in command mode. 'n' is used to go next matched word.

Linux is case sensitive

0:24

## 4. Linux 27. File Types



Everything is a file in Linux

In mac-OS we will be having f drive as a partition for the vagrant or the Virtual machine.

To know the type of the file , we must use "file <file-name>"

If the first letter of the permissions is d , then it is a directory , c is character file , b is a block file , l is some type of shortcut for some file it is not some original file , or else it will be some type of file.

mkdir -p <directory-path> => Here when we declare the path , if any of the middle folders are not present then -p automatically creates those files and create the final folder that is required by the user. -p is if present update or else create.

In -s <source\_path> <destination\_path> => which creates a softlink of the source\_path in the destination path , so that it will be easy to access. And if the original file in the source\_path is removed then the link will become a dead link.

We can remove the link using rm <file\_name> , unlink <file\_name>

9:27

## 4. Linux 27. File Types



Sort the file system by the timestamp , 'ls -lt' (latest file at the first) or 'ls -lrt' (latest file at the last)

We can change the configuration file (/etc/hostname/) using the vim , so we can change the hostname (for example we can change it from localhost to centos7.devops.in)

2:06

## 4. Linux 28. Filters



grep <word-to-search> <file-path> => This is used to find a particular word in the file

grep -i <word-to-search> <file-path> => This is used to find a particular word in the file , case insensitive

'<' is the input re-direction

grep -i <word-to-search> \* => This is used to find a particular word in all files of current directory.

grep -iR <word-to-search> \* => This is used to find a particular word in all files recursively from current directory.

grep -v <word-to-exclude> <path> => Show all the content of the file except the line which contains the word that we mentioned.

less <file\_name> => Show the content of the file , it is different and better than the cat command. We are having 'more' similarly , It shows the percentage of the file have been traversed , it is not such useful.

head -<num\_of\_lines> <file\_name> => To print first n lines of a file , similarly 'tail' command works in reverse.

tail -f <file\_name> => shows the dynamic content which is happening in the file

15:10

## 4. Linux 28. Filters



cut -d<delimiter> -f<field\_number> <path\_of\_file> => Here we separate values of the line using delimiter's and print values of the column.

awk => is a filtering technique where we can separate the values using regular expressions or many ways

awk -F'<delimiter>' '{print \$<field\_number>}' <path\_to\_file>

:%s/coronavirus/covid19/g => replace coronavirus to covid19 using vim globally

sed -i 's/coronavirus/covid19/g' <file\_name> => replace coronavirus to covid19 using vim globally , -i is used to save the changes , if we not use -i we just display the change , but not saving it.

0:20

## 4. Linux 29. Redirections



If we want to redirect the output to any file we need to redirect the output using '>' to a text file.

example: uptime > filename

example: ls > filename

> symbol redirects the content to the file or create the file or overwrite the file

>> appends the content into the file

free -m => denotes memory usage

df -h => hardisk partition utilization

echo "Text to print"

To empty the file using echo command => echo "" > filename/filelocation

Redirect the date to filename => date > filename/filelocation

Do the process of the command but should not show any output on the console => yum install vim -y > /dev/null

/dev/null is like a black hole , so everything that is redirect to /dev/null will be noting or being removed

To make file empty : cat /dev/null > /tmp/sysinfo.txt

To catch an error of a command we must use standard error number '2' : freee -m 2>> filename

Standard error => 2

Standard output => 1

Standard output or error => &

0:36

## 4. Linux 29. Redirections



Calculate number of lines : wc -l <filepath/name> ( or ) wc -l <filename/path>

0:45

## 4. Linux 29. Redirections



Calculate number of lines : wc -l <filepath/name> ( or ) wc -l <filename/path> , Using pipes , calculate number of files in current directory : ls | wc -l , search any particular file

or file with the pattern: ls | grep host ,

1:00

## 4. Linux 29. Redirections



search any particular word from last 20 lines of the file : tail -20 /var/log/messages | grep -i vagrant

To check the memory allocation: free -m | grep Mem , To check first 10 files : ls -l | head

To find the filenames that we required by the name: find /etc -name host\*

similarly we can use 'locate' command , which will be installed using 'yum install mlocate -y' , after installing mlocate , we must run 'updatedb'

locate command is not real time search , but find command is real time

1:16

## 4. Linux 30. Users and Groups



each file have it's user and the group id , one user couldn't access another user's files , so here we mostly have three types of users they are root user ( id = 0 , home dir (/root) , regular user ( created by us , user id = (1000,60000) , home dir (/home/username) , service user ( not usually login user , user id ( 1, 999 ) , homedir(/var/ftp)

3:18

## 4. Linux 30. Users and Groups



This is the structure of the password file

root:x:0:0:root:/root:/bin/bash

username:shadowfile:user\_id:group\_id:comment:home directory of root user:  
login shell

/etc/group - have the group information

This is the structure of the group file

vagrant:x:1000:vagrant

group\_name:shadow\_file:group\_id:username

8:12

## 4. Linux 30. Users and Groups



Adding a user

useradd new\_user ( which adds the new user )

tail -4 /etc/passwd ( which checks that the four users are added or not in password file)

tail -4 / etc / group ( which checks that the four users are added or not in group file) after creating a user if we want to create ID and the group do the following things

id new\_user

And for creating the group for new\_user , we must do

groupadd group\_name

Now we want to add the user into the group then , we must do ,

usermod -aG group\_name new\_user

"G" denotes secondary group

"g" denotes primary group

8:18

## 4. Linux 30. Users and Groups



To check if the user is added to the group or not , we must use the following command

id new\_user

Then we would get , the details from which we can know which group is mapped to which user

We can edit the group details using vim editor at particular location

vim / etc / group

To set password for the user , use the following command

passwd new\_user

We can reset the password of the user only by the root user, If we are root user then , we no need to have any user password because it is a root user , we use the following command to go into other user.

su - ansible

"last" command is used to track , which users have been accessed the server recently.

"who" command tells us , now which user is login

"lsof -u new\_user" lists all the open files of the user.

"yum install lsof -y" used to install lsof

12:59

## 4. Linux 30. Users and Groups



delete user with its home directory

userdel -r new\_user

delete group with its home directory

groupdel new\_group

We can reset the password of the current user using 'passwd' command

- \* Every file have it's own permissions to access and user and group and executable for it, which will be found using

- \* `ls -l <file_path>`

- \* We can have `user_name` and `group_name` details in "`ls -l`" , the first letter of the permissions is the file type and next first rwx denotes the permissions for the user and second rwx denotes the permissions for the group and third rwx denotes the permissions for others.

- \* The first name of the `ls -l` is the user name and then the group name

- \* In filetype if it is of 'd' it is directory , and if it is 'l' then it is link file type

Check this part of the video

we use "`ls -ld <folder_name>`" to check the permissions of the directory.

To change the ownership of the directory , we need to use , "`chown user_name:group_name <folder_name>`" To change permissions recursively inside the directory , "`chown -R user_name:group_name <folder_name>`"

To remove execute & read permissions for the others

```
chmod o-x <folder_name>
```

```
chmod o-r <folder_name>
```

execute permissions allows us to enter the directory , read permissions allows us to list the directory

To add execute permissions for the group

```
chmod g+x <folder_name>
```

We also have two methods of allocation of permissions , they are symbolic method and numeric method , In numeric method the first digit is related to the owner's permissions , second digit is related to group permissions , third digit is related to the other's permissions. Permissions are calculated by adding : 4 for read , 2 for write , 1 for execute

0:02

#### 4. Linux 32. Sudo



sudo gives power to a normal user to execute commands which is owned by root user , if any user have the sudoers privilege , then that user can become a root user anytime by executing it's commands.

We first logging in to our user using "cd / f / vagrant-vms / centos7" and then launch the ssh for centos7 using vagrant using "vagrant ssh", later we can see what user we are in , using whoami .

Later we can install using yum into the user , but we would get the error , so we must do the same with the root privilage access , so use sudo command for installing the package with root access.

"sudo yum install git -y" , we must have root access if we want to install any package into the server. Even , we couldn't add any other user using "useradd <user\_name>" , because we didn't have root access , we could do the same using "sudo useradd <user\_name>". As sudo gives us the root access. We can switch to the root user from another user using "sudo -i".

2:14

#### 4. Linux 32. Sudo



And then we can shift to another user using 'su - ansible' from the root user , we can set password for the user ansible using , 'passwd ansible' , from root user to the ansible which is the sub user , we don't need of any password , but when we try to create any other user from the ansible then it will ask us the password. After entering , we may get an error like "ansible is not in the sudoers file, This incident will be reported" , so when we try to switch to the root user , we may not able to switch to the root user because , the sudo command doesn't working well , so we must give access to ansible to execute sudo commands , so shift to the root user , by exiting the current user "exit".

4:14

#### 4. Linux 32. Sudo



Now we need to add the ansible which is the new user having password into the sudoers file , using "visudo" command from the root user , and there will be some chances where this sudoers file doesn't give any write access even for the root , so we must change that , this "visudo" opens the file "ls -l / etc / sudoers" in write mode. And there in around line 100 we are having a line like "root ALL=(ALL) ALL" , similarly add "ansible ALL=(ALL) ALL" , to give access of root user and sudo to the ansible user. So after this , we can use sudo

command in any user , which is been declared in sudoers file. If we don't want to ask any password for any user then just use this following syntax "ansible ALL=(ALL) NOPASSWD: ALL , so whenever we execute sudo command from our other users, it doesn't ask us any password to execute sudo command.

7:00

#### 4. Linux 32. Sudo



If we are having any syntax error in sudoers file , after executing visudo , we get a prompt in the terminal that there is a syntax error. And it asks a question as "what now?" , we have to give input as 'e' to edit that file , and correct the mistake and save it. If there is any human errors happen in sudoers file then there is a possibility that the sudo command functionality crashes. And if you don't know the access for the root , you will be stuck as you are not able to correct the error. We edit sudoers using "visudo" through the root user. And when this error occurs and the functionality of sudo doesn't work , you couldn't move from your user to the root user through "sudo -i" , so you will be stuck.

So instead of editing the main sudoers file , we can access the sudoer using sudoers.d file which is under / etc / sudoers.d/ move to that directory and add your own user or your own group , name the file with the name of the user , and add the similar syntax that we have used it prev

8:38

#### 4. Linux 32. Sudo



ansible ALL=(ALL) NOPASSWD: ALL => This is the syntax that you need to use in the file to create an access for sudo , Only user in /etc/sudoers file or /etc/sudoers.d dir can use sudo -i command to switch to root user .

0:25

#### 4. Linux 33. Package Management



In linux we have different pacakge management tools to do suppose we need to download a software on the windows machine , so we download into the file into the local system and install it , but when coming to the linux servers , we must download it through curl or yum , now let us see about curl , we can go to the internet and find the link of the rpm package that we need to find and download it using curl. And in curl command we must use -o as the output redirection which tells us to save the downloaded file with the particular name given.  
curl <link-from-where-we-must-download> -o <name-of-the-downloaded-file-to-be-saved>

rpm -ivh <local-rpm-file>

rpm -i => used to install rpm

-v => verbose to print output of the rpm

-h => print the output in human readable format

After using this command , we get to install that pacakge , which we want to install.

5:30

## 4. Linux 33. Package Management



using rpm you can install the package , but the main issue is , while installing a package using rpm , there will be some many dependencies for that rpm file , which also need to be installed , so this is the issue that we may face. you can get documentation for rpm using rpm --help. To find the rpm is installed or not , we need to use the command , "rpm -qa | grep tree" , If we want to delete an rpm then use "rpm -e <rpm-name>"

So , the better option for the package management is yum.

7:16

## 4. Linux 33. Package Management



yum automates the process of package management , To search the package named httpd , 'yum search httpd' , to install the packages including the dependencies 'yum install httpd' , and to remove the package which is been installed by the yum use , "yum remove httpd"

To check the repositories which are available

"ls / etc / yum.repos.d/"

If we are having the '\', it denotes that we can write a single command in multiple lines. If we want to download any software which is not in yum , first we need to download the software repository and save it inside the yum.repos.d and then we can execute yum command to install that software as we are having that software in yum repository.

We must do "yum update" before using yum because , this may resolve some security issues , and keep packages up to date. Watch the documentation for full clarity. We use RPM in redhat linux and dpkg in Debian OS for downloading the single package.

14:28

## 4. Linux 33. Package Management



yum repository located in /etc/yum.repos.d/

apt repository located in /etc/apt/sources.list & /etc/apt/sources.list.d

Services that will be running in Linux system?

- 1) httpd : it is a web service , it is a httpd package with httpd service
- a) yum install httpd -y => This installs a httpd package , we also need a httpd service.
- b) systemctl status httpd => This shows the status of the service.
- c) systemctl start httpd => This starts the service called httpd , this command gives the details of the process and process id , which are running.
- d) systemctl restart httpd => If we want to restart the service , we must use this command , this will be helpful when we want to restart the service after the configuration change.
- e) systemctl reload httpd => If we want to reload the configuration of the service , we can use this command.
- f) If we want to re-initialize the VM , we must use "sudo reboot". , to login into the vagrant VM , use the following command to enter the VM , "vagrant ssh"

- e) Whenever we want to enable the service at the time of the reboot automatically , use "systemctl enable httpd" and to start the service "systemctl start httpd"
- f) In general , we are having a service called sshd , if this service is not running in the VM, we couldn't run the ssh commands on the terminal , to check the status of this service use , "systemctl status sshd". This sshd is auto restarted at the reboot.

"top" command shows the dynamic process , based on the consumption of CPU and RAM , and it shows the uptime , details of the user in USER and the process that it is running in COMMAND. "top" is a dynamic command and "ps aux" is the details of the processes at particular second.

"ps -ef" , it shows the parent process id and processes process id of each process. So , using it we can find which child process been created by the which parent process.

Suppose using that , if we want to delete any process , we can use "kill <process\_id>" , so that , the particular process will be removed. This kill process is not by force , it is a

general process , but sometimes , we must forcefully kill them , if they have root privileges .

8:10

## 4. Linux 35. Processes



So if we want to kill them forcefully we must use "kill -9 <process\_id>" , normal removing using "kill" will remove the parent process with all it's child processes , but forceful killing through "kill -9" , will remove the parent process and not the child processes, the child processes will become orphan processes. These are adopted by systemd process.

Now if i want to filter all the orphan process , it is difficult to remove it individually , so we must filter it out , here I am using awk. assume httpd is a process.

```
ps -ef | grep httpd | grep -v "grep" | awk "{print $2}" => This will print all process id's
```

Now we need to pipe the results back to kill command

```
ps -ef | grep httpd | grep -v "grep" | awk "{print $2}" | xargs kill -9
```

zombie process is a process which completes it's execution , but it have it's entry in process table

0:32

## 4. Linux 36. Archiving



Archiving:

```
tar -czvf <file_name_of_archive> <path_of_directory_to_archive>
```

c => creating archive

z => compressing

v => verbose

f => file

We can see type of the file , using "file <file\_name\_that\_need\_to\_be\_checked>"

Unarchiving:

```
tar -xzvf <file_name_that_need_to_be_untar>
```

If we want to unarchive and store the untared version some where then use the following command

```
tar -xzvf <file_name_that_need_to_be_untar> -C  
<location_where_it_need_to_be_stored>
```

4:02

## 4. Linux 36. Archiving



We can do the same operation using zip and unzip command , which needs to be installed using yum repository

"yum install zip unzip -y"

For archiving:

zip -r <zip-file-name-user-defined> <folder-that-to-be-zipped>

The above command results in archiving a directory

For unarchiving:

unzip <file-that-need-to-be-unarchive>

0:26

## 4. Linux 37. Ubuntu commands



special ubuntu commands , which doesn't run on centos

See this video "Ubuntu commands"

useradd command of centos , creates a new user with home directory , but when we execute the same with ubuntu , it creates the user , but it doesn't create any home directory , when we switch to that directory , we move to the root directory. And even while deleting the user with the same syntax of centos , makes issue in ubuntu.

In ubuntu , we create a new user using "adduser <user-name>"

And whenever , we use "visudo" command in ubuntu , it opens sudoers configuration file using nano , but we need it in vi , so to make vi editor default , we must do the following.

"export EDITOR=vi" on bash line

So, now if we use "visudo" command in ubuntu , it opens sudoers file using vi editor.

5:28

## 4. Linux 37. Ubuntu commands



For centos and ubuntu , the main difference is package manager , in ubuntu , suppose if we want to install tree library , we must get the link of debian tree repository , make zip file downloaded by the wget and extract and download that debian package using "dpkg -i <deb\_package\_name>"

To search all debian packages in our system , we need to use "dpkg -l"

If we want to remove any debian package then use "dpkg -r <package\_name>"

"/etc/list/sources.list" have the repository information

"apt update" , refreshes list of all the repositories. ( just list but not inside of it )

"apt upgrade" , updates all the repositories

ubuntu "apt" have many more packages in it , so it is famous than yum.

To search any particular package inside apt "apt search <package\_name>"

To install any particular package "apt install <package\_name>"

"apt remove <package\_name>" to remove any package , it doesn't remove files and configuration of it , If you want to remove including those use "apt purge <package\_name>"

0:52

## 21. Docker 230. Introduction



In general , we try to isolate the services , because if we are having multiple services in the same system , those service compete for the resources , and there would be of information leak and the security threat among them, so we try to keep the service separate from one and another, in order to achieve that , we must have multiple systems , where the service have the OS of the system as the boundary , where the resources are available without competition from one and other. So instead having multiple individual systems we will arrange multiple VM's to run the services.

3:07

## 21. Docker 230. Introduction



To host our apps , we need infrastructure. Now we need VM's / Cloud computing to setup the infrastructure instead having multiple physical machines , we isolate our service in OS of VM. Due to this isolation , we end up and setting up multiple VM's / Instances. And these instances are overprovisioned , because we don't know how much memory a service needs , so we give more than required , and due to this we end up high capital expenditure to have multiple VM's with overprovisioned memory or operational expenditure to manage those VM instances.

4:16

## 21. Docker 230. Introduction



Here the main purpose is to isolate the services , and for that we are creating a VM for a Service , so we can isolate the service , but here we need to observe that , every VM has it's own OS , and OS needs a person to check on , and if it is a licensed OS , we have to take license for it , so it is cost increaser , and when ever we start a service , it takes time to boot the VM, even before starting the service. These VM's are portable but are larger in size , we port the VM , so that we can have a well setup environment to run the service , and which we can share between the teams , so that we can run the service in ready made fashion , without downloading the dependencies and software's . VM needs Resources for it's OS , it can be either memory , network etc.

5:07

## 21. Docker 230. Introduction



The core idea here is , if there is any change in the application , instead of adding the change in the development side and send those changes to the further stages for checking , we can simply add the change in VM and send that Image to the production or QA , so that they run that image and check it's functionality directly , without having external burden to download the dependencies and software's. As this VM is high in size we couldn't port the application so easy , so there is a thought to isolate the service without OS and share it ( Isolation without OS ). Remember OS have a larger share in memory consumption.

"Multiple services running in same OS but isolated" => each image will have it's own dependencies and software's to be installed , and there would be an external instruction to allocate resources for each and every image. So , there will be no dependency between images and there will be no resource conflict as it is already allocated.

6:34

## 21. Docker 230. Introduction



Here we will be having multiple services , which run their own process , so to stop them from interfering in other services , we must containerize the process , it is similar to process running in a directory. We will have a directory , where we will be having our binaries , packages and dependencies from which our process get's run, and this process will not interact with other service due to the process called containerization. Here we may have multiple containers but they individually doesn't have the OS to work on , all the containers share the common OS , but each and every process is isolated.

Here a process is isolated in a directory , and a directory have it's own terms like namespaces , cgroup . All the necessary binaries and libraries are present in this directory , so whenever a process runs will use these to run the service. And this directory have it's own IP address to connect.

8:24

## 21. Docker 230. Introduction



Containers share the machine's OS system kernel and therefore it doesn't require a additional OS per application , Here a container is a standard unit of software that packages up , code and the dependencies that is useful to run the application. Container is a standard unit of software , we can use it for packaging the software. package software into standardized units for development , shipment and deployment. Containers doesn't take more uptime than Virtual machine. containerized applications doesn't have OS of it's own , it shares the system OS , where virtual machine have it's own guest OS. This tells the difficulty in portable level of these images. Containers are light weight.

12:05

## 21. Docker 230. Introduction



- \* Containers offer Isolation not Virtualization, It offers Isolation of the resources , so there will be no overlap or resource conflict between them.
- \* Containers are OS virtualization , actually container uses the host machine OS as it's own OS , but here kernel tricks the process that it tells that the directory itself is an OS.
- \* VM's are Hardware virtualization , where we will be having separate Virtual RAM , Virtual memory , Virtual disk etc.. So we can install the OS on top of it. VM needs OS , where Containers don't need OS , containers uses Host OS for compute resources.

12:25

## 21. Docker 230. Introduction



Both Docker and container are different from each other , docker manages our containers. This is called container runtime environment. We can run container without docker but we need to build the directory with necessary dependencies. And build namespace and c-group and then we can run the container manually. By the way , which is very hectic to manage manually , so docker makes the work easier for us , so it is called as the one which manages containers. Docker runs the containers using docker engine. Docker initially called as DotCloud.inc , where we provide our artifacts for DotCloud and it runs the application , it is basically a PAAS business. It uses amazon EC2 to run user application. To be specific it uses Linux Containers. And to manage those containers they have developed many tools like docker engine , docker compose etc. Which is a failure business . And those tools that they have made , are moved into a open source project named as docker. Which is very successful.

14:32

## 21. Docker 230. Introduction



Docker mean Dock Worker

As this is a big hit , company name changed to DotCloud to Docker

Docker is the name of the company

Docker Engine is used to manage the containers , lot of automation is done.

Docker Project is still an open source project.

Docker is a service that runs in the OS. Which is used through REST API or Client docker CLI.

Docker images are the best business attraction. As we no need to create containers from scratch , we are having so many pre built images , where you can customize or use the images . Containerization became a big hit due to these docker pre built images , because we are having containers from a long back , even though it didn't came to lime light because as execution through containers been a hectic task until docker arrived., but the concept of image is not available until this arrived. We can use pre built images and develop the new image and run in the container.

16:51

**21. Docker** 230. Introduction

We can connect to data volumes to save and preserve the data. We can create our own network in the container or to the network. These docker images are OS independent , if we are having docker engine on our system , we can run any docker image.

Features of Docker containers that run on docker engine:

- \* It follows a standard to write the code , so that the image can be portable anywhere.
- \* These are light in weight because we don't need to have OS inside the container as we are sharing the system OS.
- \* The applications are safer inside the containers as Docker provides the strongest default isolation capabilities in the industry. It offers isolation without OS.

18:42

**21. Docker** 230. Introduction

We can run windows containers on windows OS and linux containers on linux OS and if we are using docker desktop on windows , we can run linux containers on it because , Docker desktop will create a linux VM in windows machine and top of that we can run linux containers which doesn't violate the basic rule which is mentioned in first line of this paragraph.

If process needs to run windows kernal we need a windows machine , and if process needs to run on linux kernal , we need linux machine.

0:14

**21. Docker** 231. Docker Setup

If you are going to create a terminal on amazon cloud , go to the instances tab under services and launch an EC2 instance of ubuntu 18 , instance is like launching virtual machine. If you are working on the free tier then go with the t2.micro (default option was already set). Add a tag pair of key value pair , security group and proceed furthur , and for security group we must give the source as my ip , so it uses our IP address for the network. add key pair , just before creating the instance , once check the video , and after creating the instance launch the instance , where we can get the ip address of the instance. And we use that IP address to login to the ubuntu instance that we installed , in the local git bash , the tag that we created while we install the ubuntu instance helps us to login in it. The ubuntu instance will be activated on the IP address when we launch the instance.

1:04

**21. Docker** 231. Docker Setup

We need the key pair to access the instance , a key pair consists of a public key that AWS stores and a private key file that we store , together they allow us to connect to our instance securely. For windows Amazon instance's the private key file is required to obtain the password used to log into your instance , For Linux Amazon instance's , the private key file allows us to securely SSH into the instance. So , we are creating a key pair just before the step to launch instance , this selected key pair will be added to set of keys authorized for this instance. Remember , we must download the key pair file to access the instance through this key. We couldn't come back and download the file again.

0:26

## 21. Docker 231. Docker Setup



Steps to launch an instance:

1) Choose an Amazon Machine image (AMI)

2) Choose an instance type

\* We are having a wide range of instances present in amazon , instances are like a virtual servers that run the applications. Each instance varies in amount of CPU , memory storage , networking capacity

3) Configure Instance Details

\* Configuring the instance to suit our requirements , we can launch multiple instance for same AMI. We can request spot instance to decrease the pricing. And we are able to assign the access management role to the instance. We can declare number of instances we need here.

4) Add Storage:

\* Your instance will be launched with the given storage device settings , We can attach addition EBS volumes and instance volumes to our instance. We can edit the settings of the root volume , which is declared by default, and we can add EBS volumes after launching an instance but not instance volumes. We can declare the size of the root volume here

0:32

## 21. Docker 231. Docker Setup



4) Add Storage:

\* Instance volume is like the temporary storage , because the data stored in instance store volumes is not persistent when instance stops , termination , hardware failure . For the data which we want to retain longer , or if we want to encrypt data , we use Amazon EBS ( Elastic Block Store ) volumes instead.

5) Add tags:

\* Here a tag consists of a key value pair , the tag that we write here will be applied to volumes , instances or both. This tag is typically used for segregation or identification of this instance from the other. So here we provide the tag for the instance.

#### 6) Configure security group:

\* A security group acts as a virtual firewall for our EC2 instances to control incoming and outgoing traffic. Inbound rules control the incoming traffic to our instance and outbound rules control the outgoing traffic from your instance. When we launch an instance we can specify one or more security groups.

0:44

## 21. Docker 231. Docker Setup



#### 6) Configure Security Group:

Suppose in our case,we are giving a run where the type of the request comes from ssh,And of TCP Protocol, and make sure you give the valid IP addresses that are really required,for instance now as we want to connect to that instance from our system,so give the IP as our systemip address,so we can send the request to the instance to access this instance.And rest requests coming from other IP addresses will not work because the security group firewall blocks it.

#### 7) Review instance Launch:

As discussed earlier,we must give a key pair which is required to login to this instance through the ssh.Don't forget to download the key pair that we will be creating, and this is used to login to this instance.

#### 8) Launch the instance

COPY THE PUBLIC IPV4 ADDRESS AFTER LAUNCHING

Execute this in GIT bash

use "ssh -i <path-to-.pem-file> ubuntu@<ip-address>"

\* The .pem file is key pair file and the ip-address is public ipv4 address of ubuntu instance that we launched

5:30

## 21. Docker 231. Docker Setup



How to install docker on ubuntu: <https://docs.docker.com/engine/install/ubuntu/>

After installation to check is the docker installed or not , use the following command to check does the docker service is running or not.

"systemctl status docker" where we can see status of docker daemon

"docker images" lists all the docker images in local server.

By default only root user can connect to docker daemon via CLI , so if we want other users should be able to access docker daemon , then we must edit a file called group.

Use the following command from root server

"sudo vim / etc / group" and go to the line where docker group is present and add the user from where you want to execute docker in this line where docker group is present in this file.

5:33

## 21. Docker 231. Docker Setup



Or use "sudo usermod -aG docker ubuntu"

Here docker is the group name and ubuntu is the user name from where we want to run the docker commands and we execute the above command from root user. And if you want to know what groups are accessible for current user , use "id ubuntu" => "id <username>"

To validate the docker is working or not , then use this command.

docker run hello-world => It tells to create a container out of this image , application runs inside the container.

docker ps => shows list of active containers

docker ps -a => shows list of all containers (active / stopped)

Containers may be short lived or takes long time to run the application.

0:16

## 21. Docker 232. Docker commands & concepts



In hub.docker.com , we have all the images , which may be official or non official , we can even add our own images in the docker hub. This docker hub is the registry for docker images. Docker image is a stopped container like VM image.

And image consists of multiple layers , where each RUN command generates a layer , and this image have it's own directory structure. And all the layers will be of read only mode , and we bundle all these layers of the app into an image , and container runs from these images. It is not same as vagrant box images and VM's because a VM is created using the vagrant box but VM doesn't run based on this image. But containers are created based on the image and its execution is also dependent on this image. We can't remove the image if the container is running which depends on this image. These images are similar to repositories in a registry.

These Docker images become containers when they run on Docker Engine. Docker hub is not only one registry , we call this as registry because we can store docker images in docker hub, docker hub is default registry , some of cloud based registries are docker hub , Google container registry (GCR) , Amazon ECR etc , in these cloud based registry we can have private registries for us. And we can have some Inhouse or Local Registries such as Nexus 3 , Jfrog Artifactory , Docker Trusted registry.

"Containers runs from the images"

Here we can run many containers for one image , for example , for ubuntu image we can run multiple containers on it , the container is a thin read / write layer and the image is strictly a read only layer. Containers access the data and file structure inside the image , instead having it's own data and file structure , so containers save a lot's of space.

"docker run" command creates a container for the image. But we need to pass the image name in order to do that process.

image tag is used for versioning purpose. And we can know the size of image using "docker images" command , we will have different tags inside the documentation of docker , so we can use those tags to download the image with that tag.

For every image , we will have some "how to use this image" documentation.

If we want to run the container in the background , we must use the detach mode '-d <image-name>' , so the container runs in the background , instead waiting in the shell. And if we want to specify a port number mapping from the local system to server in the image , we must use '-p <local-host-num-to-access-remote-service>:<remote-service-num>' , see the video 212 for full clarification. Here we are mapping the host port to the container port.

For example, Command look like:

"docker run --name myweb -p 7090:80 -d nginx"

We can check the status of the container using "docker ps"

Now to access this nginx service of container , we must get IP address of the EC2 instance of ubuntu we are using and go to the security group of the EC2 instance and we need to add an inbound rule , where we make allow all traffic from outside which

comes from my IP address , the another rule that we are previously having in this security group is SSH rule. In total we are having two rules.

10:40

## 21. Docker 232. Docker commands & concepts



So , by doing that we can access the nginx of container from the host machine using <ip-address>:<the-host-address-mapped-with-container-address> => 15.207.106.158:7090 , this will work until this container is running in the background. Here we are launching an nginx image where mapping 7090 host address to the 80 container port address where nginx is actually running , so we use this 7090 address to access the nginx inside the container from the host address.

We can use "docker stop <container-id>" to stop the container.

"docker ps" To display all the containers which are running , "docker ps -a" to display all the containers which are running or stopped.

"docker start <container-id>" to start the container.

12:39

## 21. Docker 232. Docker commands & concepts



### **Container is a process running from the directory**

Whenever we launch a container , we will create two processes , one is master process of root user and other is worker process of other user. We can check that using 'ps -ef'

To prove the above highlighted point , we can login to the root user in the terminal , go to the "/ var / lib / docker / containers / " , we can see there will be folders , which will be named after container id and the contents of it , will be actually the contents of the directory of the image. It proves , Container is a process running from the directory.

13:01

## 21. Docker 232. Docker commands & concepts



Containers doesn't have any data other than image have. So when we try to find the size of the container (folder inside the / var / lib / docker / containers) using "du -sh <container-id/folder-name>" , we get the size of the container in less than 1MB , it shows that , container doesn't have any data.

13:31

## 21. Docker 232. Docker commands & concepts



And to prove the point , **container has the files to run a process a process from the directory and the image have the actual data** , go to "/ var / lib / docker / image / overlay2 / " and where we can see the directory structure which have the data of the

images. It doesn't present in the container and container uses these data inside the image to run its process.

And even some containers can store some data , it is called dynamic container , where we connect volumes to it , to make the container as dynamic and mounting the data into the container in runtime and that data is not present inside the image or a container prior to this , after mounting the data , we add the data into the container.

14:49

## 21. Docker 232. Docker commands & concepts



we can execute the commands inside the docker container using "docker exec <container-name/container-id> <command needed-to-run-in-container>" , we can attach to the process executing by the container using the following.

"docker exec -it <container-name/container-id> /bin/bash" => so we would attach to the process and check the file directory inside of it during the process / container running.

We can remove docker image using "docker rmi <image-name/image-id>"

To check list of all images present in the server use "docker images"

We can't delete the docker image if the container is using it , and if we try to remove the running container , it shows we couldn't remove the running container. So , we must stop the container first and later remove it .

We can remove docker container using "docker rm <container-name/container-id>"

0:29

## 21. Docker 233. Docker Logs



"docker logs <container-name/container-id>" this command gives container logs as an output.

If we want to see metadata of the image , we must use "docker inspect <image-name>" and the result comes in JSON format. Whenever we run an image , we trigger the ENTRYPOINT of the image and later execute the ENTRYPOINT with the arguments provided by CMD.

This "docker logs" will be used whenever we run the image in detached mode , in general if we run it in normal mode , we get the output of the container immediately after executing the image. But when we run in detached mode , the output will not present in shell script , at that moment if we want to see that container logs we use this docker logs command. Docker logs provides the output of the commands that are executed inside the container. We basically see the output of the process here through this logs command.

6:21

## 21. Docker 233. Docker Logs



We can export a variable through docker run command as environmental variable ( declare it using '-e' )

Example: docker run -d -P -e <VARIABLE\_NAME>=<variable\_value> <image-name>

The process output is present in docker logs , but application logs doesn't present in docker logs

"docker inspect <image-name>" shows us the command that the image executes after launching, as CMD , it contains so many details about the image.

0:02

## 21. Docker 234. Docker volumes



Containers are famous for their volatile nature , because containers are disposable , and any change that we want to make in container , we make it through the image. Because any data stored in container is entirely depends on the image, we won't add the files through logging into the container.

The data of the container doesn't persist when that container no longer exists , and it can be difficult to get the data out of the container if another process needs it , as the data is present inside the container and it is being used in it. Here container's writable layer is tightly coupled to the host machine where container is running. You can't easily move the data somewhere else.

So , as the data in container's is going to destroy there are two options for docker to store files in the host machine , as the backup of the data.

\* Volumes

\* Managed by Docker (/ var / lib / docker / volumes /) on Linux

\* Bind mounts

\* Stored anywhere on the host system.

4:58

## 21. Docker 234. Docker volumes



Volumes can be declared using '-v hostlocation:remotedirlocation'

We can define the name of the container using 'docker run --name <name-of-the-container>'

We can define the environmental variables using '-e ENV\_VAR\_NAME=env\_value'

We can define the port mapping using '-p host\_port\_num:remote\_port\_num'

Bind mount of the volume '-v host\_dir\_path:remote\_dir\_path'

To make the docker run in the background add '-d' in docker run command, so the shell script will be not occupied by the docker.

If you want to go into the container , we need to use 'docker exec -it <container-name> /bin/bash'

10:05

**21. Docker** 234. Docker volumes

In general , these bind mounts are used to pass the source directory to the remote directory , in such a way that source can add as many files as it can during the container execution. But in actual for preserving the data the best option is docker volumes.

Basic command of docker volume => 'docker volume' => It shows the possible modules that docker volume can have.

docker volume have 'create' 'inspect' 'ls' 'prune' 'rm'

To create a volume of our own => docker volume create mydbdata

To list all the volumes => docker volume ls

Here in docker volumes , we are creating a new volume of our own instead using the host directory as the volume mount point. Example:

'docker volume create mydbdata'

'-v mydbdata:/var/lib/mysql'

This new volume directory 'mydbdata' is present inside '/ var / lib / docker / volumes ' , where inside we will be having a mydbdata volume in form of directory and it consists of \_data.

We can use docker inspect command even for container

12:53

**21. Docker** 234. Docker volumes

docker logs <container\_name> => retrieve logs of the container

Suppose let's think that , we are having mysql container running along. Where if we inspect the container while it runs , we will get the connection ip address , and at that moment , we can connect to the particular mysql shell using the following command.

"mysql -h 172.17.0.2 -u root -psecretpass"

To remove the image we use "docker rmi <image\_name>"

To remove the container we use "docker rm <container\_name>"

0:03

**21. Docker** 235. Building images

How to build docker images?

In general , we will take images from the docker hub and use those images to execute the operations , but if we want to have some changes in that images, now we need to build that image and modify it according to our needs using Dockerfile. To build docker

images we need to write Dockerfile , we provide instructions of how to build the image , what packages to be installed and what volumes to be exported. And later using docker build command we use Dockerfile to create the new Docker image. Dockerfile provides the set of instructions.

2:20

## 21. Docker 235. Building images



FROM => Provide the base image, may present in docker hub or any other repository

LABELS => Adds metadata to the image.

RUN => execute commands in a new layer and commit the results.

ADD / COPY => Adds files and folders into the image , COPY just dumps the content needed to be copied at the destination , but add will have multiple options besides copying the content. We can provide the link from which we can download the file or we can provide the achieve at the source , and according to the options we provide , we decrypt the achieve and store in the destination.

CMD => Run binaries or commands on docker run

3:06

## 21. Docker 235. Building images



ENTRYPOINT => Allows you to configure a container that will run as executable. It is the first command that will be run , if we execute the image. If we want to use ENTRYPOINT and CMD both at a time, we need to keep executable in ENTRYPOINT and commands in CMD.

VOLUME => Creates a mount point and marks it as holding externally mounted volumes.

EXPOSE => Suppose let us think , if we want to map the host port address to the container port address, and then the container port address must be recorded or declared inside the dockerfile , so it will work as intended when we try to mount the host to the remote through docker run -p , In short , it means Container listens on the specified network ports at runtime.

3:50

## 21. Docker 235. Building images



ENV => Sets the environmental variable

USER => Sets the username (or UID)

WORKDIR => Sets the working directory

ARG => Defines a variable that users can pass at build time

ONBUILD => Adds to the image a trigger instruction to be executed at a later time.

7:42

## 21. Docker 235. Building images



It is preferable to have less layers as possible , so use the privilege of '&' to combine multiple RUN commands and decrease the layers , each line creates a new layer , so use '&' command to decrease the layers.

9:59

## 21. Docker 235. Building images



Here in line 1 of Dockerfile , we are importing the base image as ubuntu , so we can perform ubuntu operations in it. And for the image that we are building using the Dockerfile , we are having the base image as ubuntu:latest

Line2&3 : We are declaring some labels inside the image , so it is used to declare some meta data inside the image.

Line 4&5: We are updating the debian apt , so apt will be up to date. And installing git and apache2 for furthur operations.

Line 6: We are running a command of apache binary to run in the foreground.

Line 7: As apache runs in port 80 , we need to access it from outside the container , we are exposing the port 80 of container to the host server.

Line 8: Define Working directory, so when ever we enter the image , we will work from this location.

Line 9: We are defining the log directory of apache as a volume because we must not lose the log data , after container exits.

Line 10: ADD command untar's the zip file and copy it at it's destination.

11:38

## 21. Docker 235. Building images



Line 4: If you want to make the docker build process non interactive , it means it must not ask any options , for example 'asking us to select our timezones' , we must use this line 'ENV DEBIAN\_FRONTEND=noninteractive'

And we used this command to run the docker image

```
'docker run -d --name nanowebsite -p 9080:80 nanoimg:V2'
```

It makes a connection between 9080 of the host server to the 80 of the remote server and run the nanoimg:V2 in detached mode and the container name would be nanowebsite as declared

14:36

## 21. Docker 235. Building images



As we know we are running this ubuntu terminal as EC2 instance of amazon cloud services. So get public ipv4 address from instances tab where we are already running 'Docker-Engine' as an instance through which we are running the ubuntu terminal locally.

So giving , <IPV4 address>:<host-port-number> ( 65.0.55.51:9080 ) in the local browser gives the connection to the apache application from the local browser.

15:16

## 21. Docker 235. Building images



This is the process to push our own image into the docker hub

Step1: Signup with docker hub

Step2: Go to Create repository

Step3: To push the image into the repository , we have a specific pattern to define the name of image, <DockerHubUName>/<ImageName>:<VersionName>

Step4: So , first we need to login into the docker using 'docker login' , then it prompts the username and password, after providing it , it prompts as Login Succeeded.

Step5: Use 'docker push <DockerHubUName>/<ImageName>:<VersionName>' to push the docker image into the repository. While we are pushing , dockerhub interprets only executable layers (for example: apt install) as the layers to be downloaded and the rest such as FROM command , will be directly mounted from the dockerhub repository that we are creating now as they will exist on the dockerhub.

Step6: The image will be pushed to the dockerhub , so we can delete it from the local server , because we can get it simply using docker pull command.

0:37

## 21. Docker 236. Entrypoint and CMD



CMD is the place where the binary or the command which starts the container process.

Example:

```
FROM ubuntu:latest
```

```
CMD ["echo","hello"]
```

so if you have this program in your basic image , you will have the output where your image will run the command "echo hello" , an image may or may not have the entrypoint.

Example with ENTRYPOINT:

```
FROM ubuntu:latest
```

```
ENTRYPOINT ["echo"]
```

So , if you run this Dockerfile 'docker run <image\_name>' , we get a empty line as an output. But if we pass some sentence for the image, it passes the sentence to the echo statement of the image. So you will get the sentence as the output , similar to 'echo <sentence>'!

6:01

## 21. Docker 236. Entrypoint and CMD



Example with ENTRYPPOINT and ARGUMENT:

FROM ubuntu:latest

ENTRYPOINT ["echo"]

CMD ["Hello"]

In general , we pass the binaries or executables through entrypoint and commands is being passed through CMD , and in some situations we need to initialize the image with some executable. So the image will run the files with that particular executable. And even we can use this CMD to declare some default arguments , so if we not pass the arguments , we use this to execute the entrypoint.

docker run <image\_name> => Hello

docker run <image\_name> User defined arg => User defined arg

ENTRYPOINT have the higher priority than CMD

0:37

## 21. Docker 237. Docker Compose



Docker compose is a tool which runs multi containers together. It is a tool for defining and running multi container Docker applications. With compose , you use a YAML file to configure your application services. Then with a single command you create and start all the services from your configuration.

In docker compose , we will have the information about all the containers , so we can start all containers at a time , with all the requirements to run the image. We use 'docker compose up' , to trigger docker compose.

Compose works in all environments , production , staging , development , testing as well as CI workflows.

Process:

Step 1: Define your app's environment with Dockerfile, so it can be used anywhere.

Step 2: Define the services that make your app in docker-compose.yaml , so they can run together in isolated environment.

Step 3: We can run 'docker compose up' and the docker compose command starts and runs our entire app, we can also run 'docker-compose up' to achieve the same.

2:30

## 21. Docker 237. Docker Compose



In yaml file , 'services' mean 'containers' , Here we can see we are having version declaration of the docker compose yaml file first , later we are having services section which defined all the containers. Next we are naming and declaring two containers 'web' and 'redis' , in the first container 'web' , we are declaring the content to build the image, so in web container declaration , we can see build is giving as the current context and ports are being mapped , volume mounting is declared , and this web container is dependent on redis image which is defined on links and on volumes , we declared those one using the bind mounts and other using volumes , and the new volume is created inside the volumes tag of this yaml file.

5:12

**21. Docker** 237. Docker Compose



We need to install docker compose externally in docker using sudo command which you can find in the official documentation. In this video we are having the application which interpret the docker compose by running the flask application inside the container using docker compose , we are having the same in the documentation. alpine is one of the lightest image. docker compose first creates a network and start building the container. So after running the application as said in the video we can run the flask application in our local system using <ipv4addressofEC2>:<hostportnumber>. We have been using EC2 instance ubuntu , so use the ip address of that EC2 instance. And as we mapped the hostportnumber to the containerportnumber. we access that flask application using <ipv4addressofEC2>:<hostportnumber>

10:53

**21. Docker** 237. Docker Compose



And we need to edit the inbound rules to access the application through EC2 instance , so go to the EC2>Security Groups>Instance that we want to go>Edit Inbound Rules> , add a rule with all traffic 'Type' which 'Source' from my ip address

'docker compose down' is used to bring down all the containers which are declared inside the docker compose yaml file at once.

### **Terminologies used in yaml file**

services => initialization of declaration of the containers.

build => give the build context .

ports => define the port mapping from the host to the remote.

volumes => can declare bind mounting or volume mounting.

environment => declare environmental variables.

image => declare the image name.

docker compose up -d => run docker compose in the background in detached mode.

docker-compose ps => shows the running containers which are declared through compose yaml file

14:24

## 21. Docker 237. Docker Compose



Containerize or Dockerise the project => Dockerfile(mandatory) & docker-compose.yaml  
(Options are present to replace)

0:42

## 21. Docker 238. Multi Stage Dockerfile



Here we can clone a repository from git hub or bitbucket through git clone command. And even while cloning the branch , we can have a particular branch to cloned inside the repository. For example , if we are having a docker folder inside the vprofile-project.git . We can retrive only the docker folder instead of whole repository. Command we use is:

'git clone -b docker https:<url>.git' => Clone only the docker branch of the git repository.

Here in 4:18 , we can see the Dockerfile code , where we are downloading the repository through git clone. And install the software and before this process , we must install the dependencies which is used to installing the software. And now with this kind of image , if we build , the image is going to have dependencies , software and the source code. For which the size of the image will be huge. So for that , we must have multi stage Dockerfile , until now we are having a single stage Dockerfile , which have all its contents saved in a single stage.

7:54

## 21. Docker 238. Multi Stage Dockerfile



Previously , we are having the dependencies , softwares , source code inside a single stage Dockerfile , but now to decrease the size of the image , we perform all the dependencies downloads and software installation in build stage 1 and the build stage 1 have it's own base image where it is ended with 'AS <build\_name>'

So, now when we are using the second stage , we lose everything in the first stage except the folders that we created. And in the second stage , we copy the output file from the first build to the destination of the image, Here , we are reducing the size of the base image , with additional dependencies to install a software. And just copying the folder/file output or software from the first build. You can see that in 7:54

19:23

## 22. Containerization 250. Containerizing Microservice Project



Containerizing is basically writing a dockerfile , docker-compose file , config file (if

1:02

**22. Containerization** 251. Build & Run Microservice App

When we are working with multiple containers , which are dependent on each other , our local machine will not work as intended , so here we are using an amazon instance of t3-medium instance. You can create an instance , on the instances tab. For us t2-micro instance type will not be useful , t3-medium instance type is the one which is required , and t3-medium instance type is medium chargeable service , whereas t2.micro instance type is free service.

Create a key pair in the instances , and download it , so we can connect to remote server from the local machine. Allow ssh is from 'myip'. And we need to give permission for http access to access it from internet , as here we are going to access nginx from the browser. Modify the configure storage to 20GB. And there we will have a column like 'User data' , where we can have our own dockerfile code or any code which we need to launch while the instance is launched. And after launching the instance , copy the IP address .

3:44

**22. Containerization** 251. Build & Run Microservice App

'ssh -i Downloads/dockerKey.pem ubuntu@18.191.226.63' Similar to this we can login to the instance that we created , the pem file is the key pair file that we downloaded earlier. Due to the Dockerfile content , that we entered in the user data earlier while creating instance , we got an instance which have docker and docker-compose already installed. After the server is created , we need to download the source code and then , we must go to the location where we are having the docker compose and use 'docker-compose build' , which is used to build all the images which are declared inside the docker-compose.yaml . 'docker-compose build' triggers the 'docker-compose.yaml'

7:17

**22. Containerization** 251. Build & Run Microservice App

When ever we do multi stage build , we will have two images for one dockerfile , for the first stage of the dockerfile , we would have named the image as <none>:<none> and for the second (the last stage) we would have the name that we declared during the build process.

After complete build process , if we want to run the build , we must do 'docker-compose up'

9:52

**22. Containerization** 251. Build & Run Microservice App

Whenever we try to 'docker-compose up' , we will have all our containers to be run on the background , we can check it using , 'docker ps' . We can check logs of individual container using 'docker logs <container-id>'. To stop and remove all the containers that are started using docker-compose is 'docker-compose down' . Suppose , if we already built the image once , and later if we have any change in the code. And we want to run the containers using docker compose , use 'docker-compose build' , then only changes will get built instead of building the whole images. And this build process will be automated using CI-CD pipeline. We need to stop the instance , if we are not using it , or else we will be charged for the usage.

0:04

## 23. Kubernetes 252. Introduction



Kubernetes is the popular container orchestration tool, for example , until now we are running all the containers in one docker engine , now suppose think a scenario where the docker engine fails and all the containers are down , then user will not have their services to be run. This will disappoints the user. So why not maintain multiple docker engines in the production , so even one docker engine down the other's will have the failed docker engine load. This is the clustering of the docker engine which we need to do when we are running in the production.

1:25

## 23. Kubernetes 252. Introduction



The multiple docker engines inside the cluster is called docker nodes and a node which monitors their functionality is the master node. Master node gives the information on what containers do the docker nodes need to run. Here master node monitors it's child nodes or worker nodes or docker nodes. If any of those docker nodes are down , the master node will transfer the load of the failed nodes to the healthy nodes. And this process is called container orchestration. Here master node is the one which orchestrates , and the worker nodes perform their actions accordingly.

We are having some container orchestration tools , such as

- 1) Docker Swarm
- 2) Kubernetes
- 3) Mesosphere marathon
- 4) AWS ECS & EKS
- 5) Azure Container
- 6) Google Container Engine
- 7) CoreOS Fleet
- 8) OpenShift

When ever we want to make a change in container , we must replace the container because containers are disposable. In google we are having 2 billion container instances running for all its services in 2014.

Kubernetes is created by the google to manage their containers it is named as AKA Borg.

Kubernetes partnered with the Linux foundation to form the cloud native computing foundation (CNCF), kubernetes release many tools like minikube , kubeadm , kops etc. And google even announced Istio , it is basically a load balancer. And later github runs on the kubernetes and later oracle joined the cloud native computing foundation (CNCF). This gives a very big boost for the kubernetes. Kuberenets manages the cluster of docker engine , it could not replace the docker engine.

1) Kubernetes provides the service discovery and load balancing service , where whenever we have a new container created for the process , here we call the container as a pod , Kubernetes auto balances the load of it. And even if that pod fails it will create another replica of it , or transfer it's load to the other pods which run it's process.

2) Kubernetes provides the storage orchestration , where it supports lots of storage options like SAN , NetApp , AWS EBS Volume ( Elastic Block Storage ) , CephFS storage. These helps to run the stateful containers, due to this statefulness the rollback is very easy to have.

3) Automated rollouts and rollbacks.

4) Automated bin packing , it means whenever we are creating a container , the kubernetes place our container on the right node where it gets the right resource based on the requirement.

5) It provides the self healing process , where if any of our instance or a node goes down , the auto scaling group will launch a replacement , where it does the same job. We are going to set that auto scaling group. To number of nodes to be as working , and if any one is down a replica of that node comes up.

6) We can manage the secret and configurations in form of the variables and volumes and also secrets which are encoded values.

In Kubernetes Architecture , we are having two types of nodes , one is master node and other is worker nodes , where we run our docker engine in the worker node and master node manages all the worker nodes. Whenever we want to communicate with the worker nodes , we must communicate through master node , where we even don't login to the master node , we uses a client to communicate with the master node and which in turn helps us to communicate with the worker node. The client will communicate with the master node with the input of yaml file , and this master node is also called as control plane. We have four services in the master node they are API server , control manager , scheduler , etcd. These are four primary services of the master node. In worker node we will be having three main services running they are kubelet , proxy , Docker Engine.

9:06

**23. Kubernetes** 252. Introduction



Master node services:

Kube API Server:

\* It handles all the incoming and outgoing communication through Kubernetes cluster. So , when we want to send instructions to Kubernetes , kube API server is going to receive that , and then it is going to pass the information to other services like schedule and etcd and worker nodes. For the kubernetes cluster the API server is the frontend. Admins connect to API server using Kubectl CLI. We must install kubectl on the machine and use it , to connect to the API Server. If we are going to manage kubernetes , we must learn kubectl. We can have some webdashboard to integrate with API server.

10:36

**23. Kubernetes** 252. Introduction



Master node services:

ETCD Server:

It is a key value store , where we store information related to the kubernetes cluster.

Etcd is an integral part of the Kubernetes control plane. Etcd stores Kubernetes cluster configuration and state data such as the number of pods, their state, namespace, etc. It also stores Kubernetes API objects and service discovery details. Kubernetes API server stores and retrieve the data from it. The etcd should be backed up regularly. Because if it fails all the important information will loose. ETCD has the data or the status of everything in the cluster.

11:53

**23. Kubernetes** 252. Introduction



Master Node Services:

Kube Scheduler:

Kube Scheduler schedules the right container on the right node. Watches newly created pods that have no node assigned , and selects a node for them to run on.

The scheduling defines on many factors

- \* Individual and collective resource requirements.
- \* Hardware , software and policy constraints.
- \* Affinity and anti affinity specifications , affinity in the sense , we are going to define the likelihood of the pod to place on some particular nodes , or we tell don't place on these nodes. Based on those user defined rules , they will be placed.
- \* Data locality
- \* Inter-workload interference and deadlines.

12:50

## 23. Kubernetes 252. Introduction



Master node services:

Controller manager:

- \* We are having so many controllers , for example : Node Controller , Replication Controller , Endpoints Controller , Service Account & Token Controllers , all these controllers has it's own separate process.
- \* But to reduce the complexity , we compiled all them into a single binary and run in a single process , but logically they are of separate process.
- \* Node Controller: It is responsible for noticing and responding when nodes go down.
- \* Replication controller: Responsible for maintaining the correct number of pods for every replication controller object in the system.
- \* Endpoints Controller: It populates the endpoints object that joins the Services and Pods.
- \* Service Account & Token Controllers: It manages the authentication and authorization. And create default accounts and API access tokens for new namespace.

13:01

## 23. Kubernetes 252. Introduction



Worker node components:

Kubelet: It runs on every worker node, an agent that runs on each node in the cluster. All the requests which are sent by the Kubernetes API will be received by these kubelet. It make sure that containers are running in pod. So when scheduler decides that this worker node is going to run the container , and then this assigns the responsibility to the kubelet. And now kubelet is going to fetch the image and run container from it. It does the main job actually.

Kube Proxy: It is a network proxy that runs on each node in our cluster. We can define our own network configuration and rules such that allow this and deny that (for example : Defining what IP addresses needed or to be blocked)

14:11

## 23. Kubernetes 252. Introduction



Worker Node Components:

Container Runtime: Kubernetes supports several container runtime , you can have Docker , containerd , cri-o , rktlet , Kubernetes container runtime interface.

Suppose if we take docker swarm as the container runtime , we must only have docker engine as the engine. In kubernetes , we can have other runtime environments

14:34

## 23. Kubernetes 252. Introduction



We can even have add ons along with these components in worker node , for example , DNS , WebUI , Container Resource Monitoring , Cluster Level Logging. These are taken from third party vendors who have some specialization in that area , for example better logging tool or better monitoring tool or web user interface or even DNS service.

14:52

## 23. Kubernetes 252. Introduction



Kubernetes Architecture is been discussed at this particular second

- 1) If we want to expose our container to global , it is done in kube proxy.
- 2) We have our containers inside the docker engine of worker node and this docker engine internally have a pod , where which multiple containers run at a time. Basically , the container in Kubernetes runs inside the pod. Worker node > docker > Pod > containers.

16:00

## 23. Kubernetes 252. Introduction



Pod provides all the resource to a container. The container will be running inside the pod using the resources that pod provide. Similar to a process which is running inside the virtual machine. Where virtual machine provides all the resources that are required by the process. And in case of pod there is no virtualization , it is basically an isolation. Here Pod gives the abstraction. We give information to the Pod what to do and Pod is going to do it , here there will be a container which is running inside it , so if we are running a tomcat process in the pod. Tomcat will be the container which is running on port 8080 and the pod will give the IP address , so we can access it by giving the POD , IP and port number of the container. In one pod we can have one or more containers. And we can

contain volumes inside the pod with the container. So that container has the access of that volume which is present with it , inside the same pod.

19:15

## 23. Kubernetes 252. Introduction



As said , we can have multiple containers in the same pod , it means we can have atleast one main container inside the pod and the rest will be the helper containers. And some containers will be like init containers , which is short lived. Finally it means , we could not run two main containers inside the same pod , but we can have some helper containers running with the main containers. For example: we could not run the mysql and tomcat process inside individual containers and placed in the same pod. As they are individual main containers of their own.

19:24

## 23. Kubernetes 252. Introduction



And pods will be distributed among multiple worker nodes. And those pods of different worker nodes can interact with each other. In general , we will have a subnet , like a local area network , a private network running inside the node. It acts like a bridge. This bridge acts like a switch , so all the pods running inside this node will able to communicate with each other with help of the bridge. But when we are connecting to a pod of another worker node then bridge zero will forward the request to the wg0, it acts like a router , it routes the signal to correct worker node. It is going to route it to the right node by based on the IP address. After router finds correct worker node which contains the pod that it needs to find. The router forwards the request to the destination switch which directs to the pod.

21:20

## 23. Kubernetes 252. Introduction



### Kubernetes Cluster Tools:

Minikube: It is used for testing and learning purpose , it sets only one node of Kubernetes cluster mostly on the computer. So it's going to launch a virtual machine using Virtual box and on that one VM , the master node and worker node components will be running only for testing and learning purpose , but not for the production purpose. We couldn't create multiple nodes.

Kubeadm: Popular tool to set up Kubernetes cluster for the production , in minikube we can have only one Kubernetes cluster , but coming to kubeadm we can have multiple Kubernetes cluster. We can have as many as worker nodes and as well as master nodes . We can run commands in master node and in worker node and connect together. Can be created on any platform VM's , EC2 and physical machines.

Kops: Multi node Kubernetes cluster on AWS. It is the most stable way of running kubernetes cluster for production. It works for AWS initially but now it supports Google cloud , Digital Ocean , open stack.

0:15

## 23. Kubernetes 253. Minikube for K8s Setup



How to create kubernetes cluster through minikube:

- \* Open Powershell as an admin
- \* Set up Chocolaty
- \* Install Minikube with Chocolaty ( choco install minikube kubernetes-cli -y ) , this installs the minikube and kubectl which is used to connect to the kubernetes cluster.
- \* And later open powershell as administrator, and run (minikube start) , it is going to launch one single VM on your Oracle VM virtual box, then using that VM , the kubernetes cluster will be up and running.

How to create kubernetes cluster through kops:

- \* Buy a domain for kubernetes DNS records. (Ex: groophy.in from GoDaddy)
- \* Create a Linux VM and install or setup kops , kubectl , ssh keys , awscli.
- \* Login to AWS account and setup the s3 bucket , IAM User for AWSCLI , Route53 Hosted Zone.

Clone VProfile-Project source code. And checkout to 'kubernetes-setup' branch , where we can see many folders , in which we can see the minikube , in which we are having the steps for installing the minikube.

4:18

## 23. Kubernetes 253. Minikube for K8s Setup



Chocolatey is the pre requisite to install minikube and kubectl which is a cli which helps to connect to the kubernetes cluster.

To check minikube is installed or not , use "minikube.exe --help"

"minikube start" is used to create a local kubernetes cluster.

This minikube start will create a VM in local and launch kubernetes cluster inside it.

To check the functionality of the kubernetes cluster , we can use "kubectl get nodes" command which gives the details of the nodes which are running on this VM. This kubectl command refers the config file which is present in ".kube/config" , which have the details of the cluster and user details with client key and certificate. And with cluster key and certificate details and cluster have it's own server IP address. check 6:23 of section 229

6:30

**23. Kubernetes** 253. Minikube for K8s Setup

When ever we execute "kubectl.exe get nodes" this refers to the kube config file , to access our cluster. And some of the basic commands are present here (<https://kubernetes.io/docs/tutorials/hello-minikube/>) , whenever we launch a deployment , a pod will be generated and deploy , where this deploy will run this pod.

`kubectl.exe get pod // To check available pods`

`kubectl.exe get deploy // To check available deployments.`

Here suppose if we want to expose the deployment and access the deployment through the service we use following commands

"`kubectl expose deployment hello-minikube --type=NodePort --port=8080`" , deployment exposed

To check the URL of our deployment of our service , run "`minikube service hello-minikube --url`" command and you should get a URL , so we can access our application. This URL is used to test that your minikube cluster is up and running. Here untill now , we have created the pod , deployment and create a service which is used to access the cluster.

8:58

**23. Kubernetes** 253. Minikube for K8s Setup

To get all the services that are in the cluster , we use '`kubectl.exe get svc`' , and if we want to delete any service , we use the '`kubectl.exe delete svc service-name`' to get what deployments are created , '`kubectl.exe get deploy`' and to delete the deployment and the pod which is associated with it is '`kubectl.exe delete deploy hello-minikube`' and to stop the minikube cluster we use , '`minikube.exe stop`' and to delete the minikube VM , we do '`minikube.exe delete`' .

0:47

**23. Kubernetes** 254. Kops for K8s Setup

For setup with kops , we need a domain for kubernetes DNS records , so for example groophy.in is a domain which is taken from GoDaddy , to setup with Kops we need to create a linux VM and do setup with the help of kubectl , ssh keys , awscli which is only used for setting up the kops and later it will not be used.

1:39

**23. Kubernetes** 255. Objects and Documentation

Kubernetes Objects:

- \* Pod (This is the smallest object that we create in kubernetes , container lives inside the pod , where if we want to make any change inside the container, we have to do it through

the pod).

- \* Service (To have a static endpoint to a pod , like a load balancer , we have EC2 instance)

- \* Replica set is to create a cluster of pods or replica of same pod.

- \* Deployment ( It is similar to the replica set , we can deploy new image tags by using deployment , and deployment is the most used object as far as DevOps.

- \* Config Map ( Used to store our variables and configuration

- \* Secret (We can store the variables and some information we can store in some not in clear text )

- \* Volumes (for example: EBS volumes for EC2 instances , we can have different kinds of volumes attached to our pod)

3:52

## 23. Kubernetes 255. Objects and Documentation



We can create multiple objects , by declaring them on the yaml file , which is taken as an input to the 'kubectl apply <url-for-yaml>'

Then all the objects that are created inside yaml file , will creates those objects.

'Kubectl get deployments' to check whether deployment is created or not.

0:15

## 23. Kubernetes 256. Kube Config



How kubectl is able to connect to the kubernetes cluster , and how does it guess the master node. And how does it get authenticated and get all that information from the cluster or even create information in the cluster, it gets through the kube config file.

1:19

## 23. Kubernetes 256. Kube Config



When we create the kubernetes cluster , we will get a file called kube config file , this file contains these information.

- 1) Cluster Information

- 2) Users that it is going to use to access the cluster. We can have many clusters , it means we can have multiple users.

- 3) Namespaces

- 4) Authentication mechanisms , for example when we want to do SSH , we need the IP address , username , password or the login key.

Finally kubectl needs cluster information , user information , namespaces , authentication mechanisms.

1:58

## 23. Kubernetes 256. Kube Config



This is the sample of Kube Config file , where it is "Kind: Config". We will have some cluster information , user information and context information where we will be having authentication details such as login details for this cluster and the user details for this cluster.

2:34

## 23. Kubernetes 256. Kube Config



We can find the config file at this particular location "home/ubuntu/.kube/config" , here we will have the details about the cluster and where we will have the server certificate details and the API server connected to it. And this API server is connected to the master node , which is created by the EC2 instance. We can see the IP address of it in EC2 instances , server details. The API server lives in the master node. When we created the kubernetes cluster by using kops , In "root 53>hosted zones" , we will have our API server which is also the domain server. Where we can see a record where kops created a api similar to 'api.kubevpro.groophy.in' which have the IP address , which connected to the master node.

4:13

## 23. Kubernetes 256. Kube Config



To verify this master node IP address , we can check it through EC2 instances tab where an instance is created , which is named as master node 'master.<name>' , where we can see the IP of it is same as the IP which we see under Root53>hosted zones. And this API server lives in control plane of the master node. Kubectl is connected to the API server of master node. So basically the IP address present in clusters of the config file is the one connected to the master node . It is master nodes IP address.

5:46

## 23. Kubernetes 256. Kube Config



We will be having multiple contexts in the kube config file , for each context there will be some associated cluster and user to it , we can have multiple contexts , but we must declare the current-context , so whenever kubectl executes it starts from this context.

8:09

## 23. Kubernetes 256. Kube Config



The kube config file is definitely needed for Jenkins or Ansible execution.

0:12

## 23. Kubernetes 257. Namespace



Namespaces is basically used for grouping or isolating the resources. Kubernetes can have multiple namespaces.

1:09

## 23. Kubernetes 257. Namespace



The namespaces are used to set various kinds of securities , quotas and few other things to your resources. We will be having some namespaces which are in default like default namespace , kube-system namespace , kube-public namespace. When we create a cluster , these namespaces create automatically. And even we can have different namespaces suppose one for production and for development and we can isolate the environments. We can create a different namespaces for different projects. Namespaces provides a mechanism for isolating groups of resources within a single cluster. Names of resources need to be unique within a namespace , but not across namespaces. Namespace based scoping is applicable only for namespaced objects (ex: Deployments , Services etc) and not for cluster wide objects (ex: Storage class , Nodes , PersistantVolumes etc)

1:57

## 23. Kubernetes 257. Namespace



Namespaces need to be created for use in environments with many users spread across multiple teams , or projects. It is not advisable to create a namespace , if we are having less users. Namespaces are a way to divide cluster resources between multiple users. We can delete a namespace , with a single command , so we must be careful.

To know what namespaces are present on system: 'kubectl get ns'

To show all the objects on default namespace: 'kubectl get all'

To show all the objects from all the namespaces : 'kubectl get all --all-namespaces'

4:12

## 23. Kubernetes 257. Namespace



In kube-system namespace , we have mostly all control plan resources . for example : ETCD manager , KOPS-controller , Kube-Proxy , Kube-Scheduler.

How to get service object from other namespace:

kubectl get svc -n kube-system

To create a namespace , use the following: kubectl create ns <namespace-name>

If we want to create a pod with particular image and pod name in particular namespace:

kubectl run <pod-name> --image=<image-name> -n <namespace-name>

We couldn't create a same object with the same name.

If we want to create an object using yaml file , we must do,

kubectl apply -f <yaml-file>

To delete everything of namespace and including the namespace: kubectl delete ns <namespace-name>

0:12

## 23. Kubernetes 258. Pods



Pods run the applications isolated , Container runs inside the pod and pod is the most basic unit in Kubernetes. It is the most smallest unit , it just represents a process running in your cluster , the most common use case is running one container per pod. If we are going to go with multi container then the other containers will be of helper containers inside the same pod. A pod is just a wrapper , it is a boundary around your containers. So , kubernetes manage the pods rather than managing the containers directly. So , this pod provides us with the abstraction. Whenever we are going to give commands for execution we give it for the pod , not for the container directly. Whenever , we are trying run a multi container pod , all the containers inside the same pod share the same resource that is provided by the pod.

1:45

## 23. Kubernetes 258. Pods



There must be only one main container and the rest will be of helper containers or init container. Whenever we are having a application with tomcat , mysql , rabbitMQ , we must have multiple pods with each of these containers , but not a pod with multiple containers. Horizontal scaling mean adding the additional nodes , vertical scaling refers to adding more power to the current machines , or nodes.

2:21

## 23. Kubernetes 258. Pods



The definition file , will have all the infrastructure defined as the code in yaml file. Most of the definition files will have 'apiVersion' , 'kind' , 'metadata' , 'spec' in their yaml files.

API-VERSION:

Kind - Version

Pod - v1

Service - v1

Deployment - apps/v1

Ingress - networking..../v1beta1

metadata: It has the information about the pod , like name of the pod or you can give labels such as key value pairs like you have tags on AWS.

Technical aspects of the yaml file:

apiVersion , kind is type of string

Metadata , spec , containers is a type of dictionary of lists.

Sample POD file is at 4:25

4:48

## 23. Kubernetes 258. Pods



Kubectl commands:

kubectl create -f <yaml-file> => Used to create the object

kubectl get pod => used to get the pod details (object) (We can have , for how much time this object is actually running)

kubectl describe pod <pod-name> => To get some important details regarding the object , we use describe , similar to what we are having in docker using docker inspect.

We can get the details of the pod in form of Yaml file , using

kubectl get pod <pod-name> -o yaml

We can pass on that yaml file to the new yaml file using redirection operator

kubectl get pod <pod-name> -o yaml > webpod-definition.yaml

We can edit a pod using 'kubectl edit pod <pod-name>' only some part of the pod is editable.

6:13

## 23. Kubernetes 258. Pods



Check 6:13 command to create a KOPS cluster or update a KOPS cluster or validate a KOPS cluster. Using the validate command , we can get how many master nodes and worker nodes are present inside our kubernetes cluster. The config file contains the hostname or URL of the cluster , and what user used to login to the cluster.

Kubectl get nodes => Get information of the nodes

Kubectl describe node <node-name> => Get more information related to the node in human readable format.

Kubectl describe node <node-name> -o yaml => Get more information related to the node in yaml format.

10:56

## 23. Kubernetes 258. Pods



Creation of the pod examples started from 10:55 , we can create name of the pod in meta data and name the app label even. We can create multiple containers inside the

spec section of the yaml file , we can name our containers and declare it's image name , we can declared the port number that the container should be connected to . And the port mapping will be done inside the service object. "INDENTATION" matters while writing the yaml file.

We can create a object using the yaml file using 'kubectl create -f <yaml-filename>'

We can know the status of the pod using 'kubectl describe pod' , we can even see the events that are happening inside the pods.

1/1 Ready mean => out of /1 containers 1/ container is running.

Kubectl delete pod => delete the pod

kubectl describe pod <pod-name> => We will have the IP address on which the pod is executing. And the image name which is executing inside the container.

1:44

## 23. Kubernetes 259. Different levels of Logging



To know what pods we are having , we use 'kubectl get pod' , and in the status of the pod , we can see the status of the pod. If there is in running state then there is no issue , and if it is in 'ImagePullBackOff' then it means the image pull process is not successful and the name that we given is wrong image name.

To see better details of the pod we use 'kubectl get pod -o wide' and even if we not able to find the error in image name , we check it using 'kubectl get pod <image-name> -o yaml' => The details about the pod is represented using the yaml format. Where you can see the details about the image in clear , by what name we passed to the container. And we can find the error.

We can get the details about the pod in clear using 'kubectl describe pod <pod\_name>' where you can see the events section , where you can see the process of the execution and even we can see if any error occurred here.

3:27

## 23. Kubernetes 259. Different levels of Logging



This yaml file is the root cause of the mistake , where we defined the name of the image is wrong for nginx container.

And in some cases like 'CrashLoopBackOff' we could not find the error , based on these three commands , so we have to find another way to find the error. Basically the crashloopbackoff occurs when the container is failing to restart repeatedly.

Command which we need to use : kubectl logs <pod-name>

In generally pod executes the container , and container internally executes the process , where the logs of this process will be given by this kubectl logs command.

When you execute this command you may get an error , related to the command that you try to execute or executed early. So based on that error you can do 'history | grep <error-main-reason>'

8:13

## 23. Kubernetes 259. Different levels of Logging



We could not edit the pod , what we do is we edit the deployment , replica set , daemon set and which in return replace the pod instead of editing the pod. So , we can do delete pod manually and re-create the pod without error.

0:00

## 23. Kubernetes 260. Service



Service is used to connect with or to your POD, If we want to expose our pod , then we need to host application running in the pod as the network service. Suppose if we want to host nginx or tomcat or any frontend or the backend , we need a service with our pod. And if we want to expose our pod to communicate with each , we need a service in front of our pod.

What is the need of the service to connect the pod's , why not we can have a simple pod mapping?

As we know container's are disposable and when we want to make any change in the pod having the container , we can't edit the pod , we must replace the pod , and as ever pod have it's own static IP , it will be changed whenever we change the pod , so if we create a pod mapping , whenever we replace the pod , we must change the pod mapping with the new IP address , so this process will be very hectic , when coming to large projects.

1:34

## 23. Kubernetes 260. Service



So, here there is a necessary of dynamic IP address , here comes the necessary of the service. Service provides the dynamic IP and where it provides a permanent link between other pods through their services , so whatever we change inside the pod doesn't affect the external IP address of a service. So , comes the connection between the pod using the service. It is similar to what elastic load balancer does to EC2 instances. We can do what ever changes for EC2 instances even we can add or delete the instances , and their IP address may be changed , but the load balancer will be dynamic because it doesn't be affected by the internal EC2 instances actions.

2:52

## 23. Kubernetes 260. Service



In kubernetes , we have three types of services , one is node port which is similar to port mapping , we have seen in docker a host port we pick up and map it to the container port, it works in the same way , it is mostly used for non production purpose , not for exposing the front end to the production. It is just used to expose your pod to the outside network.

A NodePort publicly exposes a service on a fixed port number. It lets you access the service from outside your cluster. You'll need to use the cluster's IP address and the **NodePort** number

2:04

## 23. Kubernetes 260. Service



A ClusterIP provides network connectivity *within* your cluster. It can't normally be accessed from outside. You use these services for internal networking between your workloads. If we don't want to expose it to the outside network , but internally like Tomcat connecting to MySQL. So for MySQL , we need a static endpoint , so you can create a service of type cluster IP. Cluster IP is only for internal communication , we don't have any port mapped to your node internal communication.

**Important:**

Container will be inside the Pod and the Pod will be inside of the Node and Node will be inside of the Cluster. A cluster have many nodes inside, Cluster IP is used for internal communication , and if we want a node to be published outside , we will use cluster ip with the node port , to access that node from outside the cluster.

2:55

## 23. Kubernetes 260. Service



**Load Balancer:**

This is used to expose to the outside network for production use cases. Suppose , we are running a tomcat pod and I want users from the internet to access that. So , I will create a service of type load balancer , which in-turn create a elastic load balancer and map my Pod to it.

So , for every pod or a cluster of pod that are running , we need a service in front , either if it providing a network service , like frontend to the user , or we will have a service for backend , service for the database (MySQL), This is how the communication happens between pods and between the users to the pods.

7:02

## 23. Kubernetes 260. Service



Let's see the example for NodePort Service type, where we are having a node ip and a node port to the node , and we are having a service in which we will be having a frontend port (port) and a backend port (targetPort) inside it , this service is mapped to the pods through the label selector (selector) for which pods we are having the same label. And the service will be having it's own IP address and the Pod have it's own IP address , and we are having a container inside the pod where the container have a port number which should be the same as backend port of the service to establish a connection between the service and the pod.

7:18

## 23. Kubernetes 260. Service



So , to go in the flow , to interact to the application such as tomcat or nginx , use the node ip:node port address to communicate with the service that are attached to the related pods of nginx , where this service have it's own IP and frontend port through which it can contact to the node IP and forward to connect the service to the nginx pods we are having backend port address and label selector which redirects us to the pods which is having nginx container. Here In service node both the frontend port address and backend port address need not to be same. The container that we required is present inside the pod. Where that container have it's own port exposed.

7:28

## 23. Kubernetes 260. Service



So, the response from container is exposed through it's port address to the service backend port address , these both address need to be the same and labels also should be matched for them. IP address may vary. And this service based on it's frontend port address , we are redirecting response to the nodeIP:nodePort. The user takes this response using the nodeIP:nodePort from UI.

Nodes of Node port type should have type as node port mentioned in spec, some of the ports need to be mentioned , for example : targetPort , nodePort , port , protocol under ports section of spec. And we need to declare a selector under spec which is used to map the service to the pods.

7:16

## 23. Kubernetes 260. Service



kubectl create -f <service-file> => Used to create a service

kubectl.exe get svc => To get information about all the services in the kube cluster.

kubectl.exe describe svc <service-name> => Used to get detailed description of the service that we created , to get output inform of yaml file use 'kubectl.exe describe svc

```
<service-name> -o yaml'
```

9:58

## 23. Kubernetes 260. Service



Here this is the example of a Service object file , we will declare the name of the service using metadata. And we declare the port numbers for the service using ports section. Under this 'port' is the front end port (8090) and 'nodePort' (30001) is the external port number which starts from the 30000 , and 'targetPort' (8080) is the number which points to the port that container runs. We can declare a port name instead of the port number. Protocol is TCP. Using the selector tag of spec section we declare the label to connect to the containers. We declare the type as the NodePort under spec section. Ports , Selector and Type will be at the same level.

13:48

## 23. Kubernetes 260. Service



While we are using 'kubectl get svc' , we will get ports inform of '8090:30001/TCP' , which denotes , 8090 is an internal frontend port and 30001 is external frontend port.

We can get more details of the service using 'kubectl describe svc <service-name>' the endpoint of the service will be mapped with the Pod IP and target Port number , this became possible due to the matching of our selector label.

When master nodes and worker nodes are created , we will have these to be created as the instance inside the instances of the AWS , so to access the service provided by the pod , we must edit the security group of the worker node to all traffic from my IP , because We are going to access the worker node through different IP's. So access that worker node with it's IP and external port number , '<IP-address>:<external-port-number>' , Then the application should run actually.

Whenever the usage of service is completed , we must delete it to save the instance cost.

16:21

## 23. Kubernetes 260. Service



Similar to the node port , we are having a load balancer , to expose the container outside the cluster. Even for load balancer , we are going to have a service object and of LoadBalancer type under spec. And under ports section , we are going to define the port , this is the external port through which we can access this service. And we are defining the target port which redirects the request to the pod port. And we declare the label in selector to map it to the pod. Here we are not creating a nodeport , but it is automatically get's created through the load balancer.

When we create a service for load balancer , it get's auto created at load balancer section of amazon AWS. You can check it's configuration even in AWS.

18:04

## 23. Kubernetes 260. Service



Node port and Load balancer are doing similar tasks , similar to node port , load balancers also redirects its request to the service. It is going to route the request to any of the worker node on the port number on the node port. As we didn't specify the node port , it is going to pick up a random node port as it has a particular range to select. After selecting it is going to assign the particular node port to the worker node.

We can access the load balancer through its DNS name. So we can access the pod or a container through Node Port or a Load Balancer. These two types of services ( NodePort , LoadBalancer ) will connect to the service and communicate with the internal pods. And expose the process to outside.

18:32

## 23. Kubernetes 260. Service



Load Balancer sends its request to the service , and service have its rule defined earlier . And this service routes the request to the pods and get the response and send it back to the load balancer , so the user can access it using the external port address. Services will be across the nodes , it can be in master node , it can be in worker nodes, service is just a rule defined for performing the process or it is used to redirecting the request. Service have the dynamic IP for the pods under it.

Cluster IP is without any external port and no node port will be present.

If two processes or two containers inside the cluster want to refer each other , then there will be usage of the cluster IP.

19:40

## 23. Kubernetes 260. Service



To create a service of type 'ClusterIP' , we will declare the frontend port and the backend port for the cluster IP , and no need to declare any other ports , and if any pod wants to interact with this tomcat server , it will interact to the port 8080 which is declared by this service which is created just now with type of ClusterIP. If a pod wants to use this tomcat service , we will declare the containerPort as 8080 , while we declare the structure of the pod.

We can have front end service of type node-port or load-balancer , and we can have the backend service of type cluster IP

20:29

## 23. Kubernetes 260. Service



If we want to see all the running objects in current namespace , use 'kubectl get all'

2:08

## 23. Kubernetes 261. Replica Set



ReplicaSet maintains a Replica of your pod, so when we are having a pod running on a node and users are accessing it , and for some reason the pod goes down , and the users won't able to access the service of the application. Then someone needs to get login , delete the pod , recreate it and fix the problem.

But if the pod is running with ReplicaSet , And any of the pod was crashed then this replica set creates a new pod for you. This ReplicaSet is used for basic health checks , if any of our pod goes down , we don't need to create it manually. And if we are declaring more pods than we are required then this replica scheduler will distribute the pods across multiple worker nodes. If a pod crashes , the replica set creates another pod replica , and if the node itself crashed then pod will be created on the healthy worker nodes.

ReplicaSet is used to guarantee the availability of specified number of identical pods. So , there will not be any unplanned downtime for the users.

3:23

## 23. Kubernetes 261. Replica Set



Basically replica-set yaml file looks like this , normally for older version of the kubernetes objects we have v1 as the apiVersion , and for the newer kubernetes objects like ReplicaSet , Ingress and Deployments we have apps/v1 as the apiVersion , kind of the Object is ReplicaSet. In spec section of the yaml file , we need to declare number of 'replicas' , we must define the matchLabels for the pods to be matched. And we declare the details of the pod , image in template tag. Whatever label that we give in matchlabels will be used to create new replica pods , for example here in replica service if we give the label as frontend and declare replicas as 3 then we can create 3 pods of name labeling as frontend , if any pod of them are failed then the pod recreates automatically. This replica set ensures to up 3 pods and run every time to prevent downtime.

7:42

## 23. Kubernetes 261. Replica Set



After creating a replica-set , and creating a replica-set object , use 'kubectl get rs' , to see that replica-set is created or not. So , with the replica-set , we get automatically get three pod's created and you can see that using 'kubectl get pod' , you can delete those pods using 'kubectl delete pod <pod-name>' , even though replica-set , immediately creates those pods. To stay up with number of replicas , we can upscale and downscale the replica set by editing the yaml file and recreating the replica-set. We have to recreate the replica-set using kubectl apply command.

We can even scale down or up using the 'kubectl scale --replicas=1 <replica-name>'

Or we can do the same using

kubectl edit rs <replica-name> => Then we can edit our replicaset and after modifying and exiting our replicaset yaml file the replicaset gets modified.

Last two commands for scaling replica set is not preferable in production , we must only do it using the modification of yaml file directly and applying it.

10:22

## 23. Kubernetes 261. Replica Set



We will be having a kubernetes cheat sheet similar to this , where you will have all your commands in one page , so you can find the required command easily.

We need to delete the replicaset using 'kubectl delete rs <replica-set-name>'

0:20

## 23. Kubernetes 262. Deployment



Most of the devops work is in the deployments. Making regular changes to the code and upgrade your pod images and we can do rollback if we have done anything wrong. We declare deployment object in a definition file for pods and replica set. Deployment uses Replica set , where this make changes to your pod or it maintains the replica of our pod.

We create definition of deployment which in turn creates the Replica Set and Replica Set will maintain your pod. Whenever we apply a change for the deployment , We need to define the desired state in Deployment , and the Deployment controller changes from actual state to the desired state at a controlled rate.

For example , when we want to make changes on the pod , it will happen only by deleting the existing pod and creating a new pod. So this deletion and recreation will happen in controlled state , one by one. Deployment creates the replica set to manage the number of pods.

2:52

## 23. Kubernetes 262. Deployment



So for example , currently if we have some nodes which have the image version as v1 which we mentioned in our pod definition , container information. Now we can mention in the deployment that I want to upgrade it to a newer tag , for example: v2. It is going to update on all the pods one by one. And if anything goes wrong , we can do the rollback , It works similar to the auto scaling operation of AWS.

3:44

## 23. Kubernetes 262. Deployment



File difference between Deployment and ReplicationController

5:39

## 23. Kubernetes 262. Deployment



We are creating a deployment using this file , so it creates a replica set and as replica set needs to maintain 3 pods , 3 pods are being created.

6:04

## 23. Kubernetes 262. Deployment



Here we after we creating a deployment , we are verifying the image name that the pod have . So next , when we are going to change the pod configuration through the deployment , such as image version , the pod will get deleted and replaced by the updated one. From this part of the video , it is shown practically.

8:44

## 23. Kubernetes 262. Deployment



Whenever we change the deployment configuration and update the deployment object , then the old container will be replaced by new container and hence three new pods get created , these pods are created using the replica set , and this replica set will be recreated rather than updating , the older replica set will be decreasing the pods one by one and increase pods in the new replica set , we can see that change in "kubectl get rs"

Rollout: when we want to roll out the replicaset and go back to previous status of the deployment , we must use these following commands for it.

The following command is used to check the recent deployment change commands that we have used. We can see the rollout number , so we can use it to rollback

"kubectl rollout history deployment/<deployment-name>"

Command which is used to rollback:

"kubectl rollout undo deployment/<deployment-name>"

After doing rollback the old replica set will be accumulated with the pods again and the new replica set will become again.

10:58

## 23. Kubernetes 262. Deployment



We can rollout to any particular revision number using

```
kubectl rollout undo deployment/<deployment-name> --to-revision=<revision_number>
```

For scaling purpose we can do it using

```
"kubectl scale deployment/<deployment-name> --replicas=10"
```

0:10

## 23. Kubernetes 263. Command and Arguments



How to pass commands and arguments into our POD. POD doesn't execute the command but container does. In general whenever we run the image , it generally triggers the CMD command of the Dockerfile , here in CMD , we generally declare the executable with parameters , only one CMD will be considered from the Dockerfile , if we declare multiple CMD's in same Dockerfile , then the last CMD will be executed. CMD in general provides the default values for an executing container. We can even declare the default executable or else we can omit the value of exec , but if we omit that value , we must declare the entry point. ENTRYPOINT and CMD perform same operation , but ENTRYPOINT have more priority than that of CMD. So if we wanted to use it , we must declare executable in ENTRYPOINT and default parameter values in CMD. We can use ENTRYPOINT and CMD individually or both at a time. This is the way which is used in container level.

3:45

## 23. Kubernetes 263. Command and Arguments



But when coming to our case of Kubernetes , we need container inside the pod and if we want to overwrite the image parameters using pod definition file , use the following process.

Container runs the command , Pod have the Container. In the level of kubernetes , we are going to declare the executable and parameters through the pod definition file.

In the pod definition file , we will give the command in the 'command' and the arguments under 'args' of spec<containers , here define the name of the container with the image name and the command and arguments for the image.

0:15

## 23. Kubernetes 264. Volumes



In volumes we have a sub topic which is called persistent volume. As a devops developer , we must know how to map a volume to your pod. When we are using the docker image , we are simply mounting the local volume to the container volume , so whatever changes that we done in container on the particular folder , the data will be persisted on the local , but here when we are working on kubernetes. We couldn't directly mount a volume to the container , because here we have other players in between the container and the resource , for example POD. So we have to map a volume to our pod.

1:23

## 23. Kubernetes 264. Volumes



There are so many options for volumes , once refer the documentation. local volumes is the easy way of having volumes. portworxvolume is a most popular enterprise volume. We generally use the "host pods" , where we mount the directory of the worker node to the directory of the pod, so the data which is created inside the pod will be persisted outside the pod. Best storage solution is the persistent volume , where volume will be stored separately.

3:59

## 23. Kubernetes 264. Volumes



This is hostPath configuration file example , where we are declaring the pod and we are declaring the volume mount to the volume named 'test-volume' in volumeMounts<containers<spec. The volume directory is declared under mountPath of volumeMounts and the name of the volume that we must refered to is declared under 'name' of volumeMounts. And the details of the volume is declared under volumes<spec , name of the volume under 'name' and location of the source directory for volume is under path<hostPath<volumes<spec and declare type of the volume as Directory , the name that we declared under volumes section must be same as the name under the volumeMounts section. And declare source volume type as the DirectoryOrCreate , to create the source directory if it not exists. So now what ever happens in the destination volume will be recorded in the source volume.

8:13

## 23. Kubernetes 264. Volumes



And even the pod which is having the destination volume get's deleted or complete it's execution , the source volume which is inside the worker node will have the data of it , irrespective of the status of the pod. In general , we use azure storage or some aws storage for source volume.

0:15

## 23. Kubernetes 265. Config Map



As we know pods are disposable , so whenever we want to have any change in the pod , we must have dispose the previous pod and create a new pod with the new changes , so we will have the persistence or we can send all the configuration and variables in the pod.

1:39

## 23. Kubernetes 265. Config Map



In general , when ever we want to pass any environmental variables to the container , we must define it under the spec>containers>env , but there are cases , where the environmental variables are flexible or the values of the configuration changes frequently , in that case , we couldn't delete and re execute the pod repeatedly. So, there is a option in kubernetes called config map. It is used to set and inject variables or files in the pod.

2:15

## 23. Kubernetes 265. Config Map



Creating a config map imperative manner through command line:

```
kubectl create configmap db-config --from-literal=MYSQL_DATABASE=accounts --from-literal=MYSQL_ROOT_PASSWORD=somecomplexpass configmap/db-config created
```

db-config => config file name

--from-literal => we are passing keys for the config map using this

kubectl get cm => used to get all the config maps available.

To get a detailed description of the config map , we need to use.

```
kubectl get cm <config-map-name> -o yaml
```

(or)

```
kubectl describe cm <config-map-name>
```

3:28

## 23. Kubernetes 265. Config Map



Creating a config map in declarative format:

We must declare the object kind as the ConfigMap and the name of the config in metadata , and the keys of the ConfigMap in the data.

5:22

## 23. Kubernetes 265. Config Map



We can map the config file to the pod using this following syntax , we will have a tag under spec>containers>envFrom>configMapRef , where we declare our config map and map the config map to the pod, and we can do this process specific to the key mapping , using spec>containers>env>valueFrom>configMapKeyRef>define key here with the config file name. This configMap is used to store non-confidential data in key-value pairs , pods can consume configMaps as environment variables , command line arguments , or as configuration files in a volume. This config map allows you to decouple environment specific configuration from your container images, so the application is easily portable. The config map doesn't provide secrecy or encryption , if the data that we want to store is confidential , we must use secret rather than a config map.

7:00

## 23. Kubernetes 265. Config Map



Here we can define , the config file in this following way , where the object type is the ConfigMap. Key's and it's values are defined under the 'data' tag . Where we define the key and it's value. Which is used to map to the pod through the config file , either we map the whole config file or we map only a key of the config file.

9:26

## 23. Kubernetes 265. Config Map



Here , the code which we can see is Pod object. Remember , we have already made config map previously and using that config map here in this pod object under spec<containers<env , where we declare a variable to store and retrieving the value from the config map by it's key. And when coming to the files , we declare some mounting which is declared under volumeMounts. volumeMounts declare volume in the container. And under volumes section , we set the volumes at the Pod Level , then mount them into containers inside that pod. Under spec<volumes we declare the volume and associating with the config map and creating the files , which have environmental variables , these files are part of the original config map object. Under spec<volumes<configMap<items , we declare the key for referencing the key inside the config map using key attribute and the path where we need to store under path attribute.

13:34

## 23. Kubernetes 265. Config Map



This is the final output we got after mounting the volume inside the container , variables are stored inside the container , we can echo those commands and files will be stored at particular mount location. These are the following commands which are used to verify the functionality.

1:02

## 23. Kubernetes 266. Secret



In general , if we want to send the details such as variables and files to the pod we use the config map, but there will be some cases where the information in variables or the files may be sensitive , so for security purpose , we use secrets object. Secrets share encoded/encrypted variables to pod , secrets store and manage sensitive information , such as passwords.

1:59

## 23. Kubernetes 266. Secret



This is the way to create secrets in an imperative way.

```
kubectl create secret generic db-secret --from-literal=MYSQL_ROOT_PASSWORD=somecomplexpassword
```

The following syntax will create a secret and have a variable MYSQL\_ROOT\_PASSWORD which have some encoded ROOT\_PASSWORD , but not the actual value , remember these values are just encoded , but not encrypted.

And for passing files in secret we do,

```
kubectl create secret generic db-user-pass --from-file=./username.txt --from-file=./password.txt
```

3:27

## 23. Kubernetes 266. Secret



In general , this secret files and variables are not encrypted they are just base64 encoded , so it is not that secure , so we can simply get that base64 value and decode it using base64. But how do you think it would be safe. These base64 encoded values are only visible to the kubernetes control plane , so only if you are able to access the control plane it will be insecure , or else it is secure in general cases.

```
echo -n 'secretpass' | base64 => encodes the value
```

```
echo -n 'encodedvalue' | base64 --decode => decodes the value
```

4:01

## 23. Kubernetes 266. Secret



Whenever we are declaring the secret in declarative format , we must encode the value into base64 in prior and use that in secret yaml file under 'data' tag. Secret Object is of kind Secret. suppose we are declaring db-secret in form of base64 under the data tag of

Secret object , and use that under db-pod of type Pod object , in pod we store it under spec>containers>envFrom>secretRef>name = db-secret.

4:31

## 23. Kubernetes 266. Secret



This is the difference between declaration all the environmental variables under one db-secret and export the whole secret or just export secrets of particular key under the secret object. Once follow the secrets documentation of kubernetes.

5:34

## 23. Kubernetes 266. Secret



This command is used to store docker credentials into docker hub or any other repository , for example when we are pulling some image from the private repository , it requires the docker-email , docker-username , docker-password and the server name. So , when we are working with kubernetes and we want to pull the images from private repositories , we couldn't give the credentials every time , so to prevent that , we are passing the details in secret object of docker-registry type.

Command:

```
kubectl create secret docker-registry <secret-name> \
--docker-email=tiger@acme.example \
--docker-username=tiger \
--docker-password=pass1234 \
--docker-server=my-registry.example:5000
```

6:35

## 23. Kubernetes 266. Secret



If we want to get the secret in form of json then use the following command.

```
kubectl get secret secret-tiger-docker -o jsonpath='{.data.*}' | base64 -d
```

Here base64 -d is used to decode the base64 encoded version.

7:42

## 23. Kubernetes 266. Secret



The whole process of this docker-registry secret is , we need to login to the private registry and pull over the image into the local server. And the docker-registry secret that we have seen just now creates the secret with all the necessary credentials to login to the docker hub private repository , and we provide that secret name to the pod using

spec>imagePullSecrets , where we provide the name of the secret which is having the credentials to login to the private repository and pulling the image. Here the image that we are mentioning is a image under private repository. The pod uses the docker-registry secrets under imagePullSecrets and login to the private repo and pull the image.

8:53

## 23. Kubernetes 266. Secret



Sample Secret file

9:31

## 23. Kubernetes 266. Secret



Simple pod file which is using the variables in encoded format which is declared in secret file.

10:45

## 23. Kubernetes 266. Secret



If we want to go to any running object , use this command

'kubectl exec --stdin --tty secret-env-pod -- /bin/bash'

1:12

## 23. Kubernetes 267. Ingress



API object that manages external access to the services in a cluster , typically HTTP. Ingress provide the load balancing , SSL Termination and name based virtual hosting.

2:21

## 23. Kubernetes 267. Ingress



In general , Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. It mean , if any user want to access the application , the user interacts with the cluster , and the cluster sends the request to the Ingress , which is also used as load balancer , Ingress manages the external access to the services in a cluster and it even provide the SSL termination and name based virtual hosting. The ingress , after checking the request validity it routes to the particular service based on the routing rules and this service will send the request to the necessary pod.

Service is built upon the pod and Ingress is built on the pod this Ingress manages the requests from/to the service , service is following the cluster IP and the client sends the request to the cluster which in turn sends the request to the ingress, and process goes on.

3:09

## 23. Kubernetes 267. Ingress



We need ingress controller to use the ingress , for example nginx is one of the ingress controller. nginx ingress controller for kubernetes works with the nginx webserver (as a proxy)

4:47

## 23. Kubernetes 267. Ingress



We must download the ingress controller and use it's services to control the routing of the requests. And if not the case , we must use nginx pod as a API gateway and use it for routing to multiple application or the nodes internally , as shown in the image here.

4:58

## 23. Kubernetes 267. Ingress



Process of downloading Ingress Controller of nginx

6:04

## 23. Kubernetes 267. Ingress



Basically Nginx is a pod , for example in AWS we use a Network load balancer (NLB) to expose the NGINX Ingress controller behind a service of Type=LoadBalancer

Command which is used to get the status of namespace and to check namespace is present or not:

```
kubectl get ns
```

Command to check what are all objects present in particular namespace:

```
kubectl get all -n ingress-nginx
```

nginx controller pod routes the request from the user to the service.

The command that we used to download nginx with AWS triggers the AWS to have a loadbalancer of nginx , which we can found in Load Balancer of Load Balancing option. This load balancer creates worker nodes for load balancing.

10:15

## 23. Kubernetes 267. Ingress



Here you are having three yaml files , where we create one deployment , one service , one ingress , ingress is used to connect and load balance the external users and redirect to the service, and from service it routes to the necessary pod through the deployment and access the application. Please see these three yaml files , we will get an idea on how to connect ingress to deployment through service.

10:52

## 23. Kubernetes 267. Ingress



Here we can see that , we are having the load balancer which is created by the Ingress. And where we have to retrieve the DNS name from Load Balancer section of Load Balancing and by copying this dns name , we must create a CNAME in godaddy , it is the place where we take the domain name. This CNAME is present under My Domains / Domain settings / DNS management , where we have our DNS records , and now create a new DNS record where TYPE = CNAME , Name is of random and the value must be of the dns name which we copied in AWS.

12:37

## 23. Kubernetes 267. Ingress



When we create a Ingress , we must keep the service in mind , the service must be in working state. And find the Endpoints of service using 'kubectl describe svc <service-name>' and bind it to the ingress.

13:09

## 23. Kubernetes 267. Ingress



This is the sample ingress file.

The process that is generally used is ,

Controller

Deployment

Service ( Should be working and running )

Create DNS Cname Record for the Load Balancer

Create Ingress

We can check the ingress using 'kubectl get ingress'

16:41

## 23. Kubernetes 267. Ingress



Please refer Ingress Documentation

There are so many ways of routing , for example port based routing , host based routing , path based routing.

If we delete a namespace then all the objects under it will get automatically deleted.

kubectl delete ns <namespace-name>

(or) Once go to the documentation , where we can see a way to delete the ingress using manifest. (kubectl delete -f <manifest-link>.yaml)

We create ingress for the service , so we must create Ingress rules for routing various API's to various services.

2:47

## 23. Kubernetes 268. Kubectl CLI & Cheatsheet



This is the example of dry run , and we can use this for getting the template of particular object.

kubectl run <pod-name> --image=<image-name> --dry-run=client -o yaml > ngpod.yaml

This creates the template for the pod which is named as <pod-name> and having the image as <image-name> , this generally perform a dry run and generate new yaml file named ngpod.yaml

10:58

## 23. Kubernetes 268. Kubectl CLI & Cheatsheet



We are having 3 new commands , where it is related to the workload balancing

kubectl cordon my-node => make the node as un schedulable , no new pod can run on this node to decrease the workload and for maintenance.

kubectl drain my-node => remove all the workload on the node and transfer it to the healthy worker nodes.

kubectl uncordon my-node => And after maintenance done , we uncordon the node , so the new nodes can run the pods on the node.

These works are been done by the kubernetes administrator.

2:48

## 23. Kubernetes 269. Extras



We have the option of taints and toleration , where we create a taint for the node and we will have toleration that is declared for each pod , so if the pod have this toleration which is suitable for the node then the pod runs on the particular node.

4:10

## 23. Kubernetes 269. Extras



We are having two options here , resources.requests and resources.limits , where resources.requests define the minimum amount of resource that is reserved for a container , so the container of the pod or the pod will be created only if the memory can be allocated to that pod , and resources.limits mean the maximum amount of the resource to be used by a container. That means , container can never consume more than the memory amount or CPU amount indicated.

4:39

## 23. Kubernetes 269. Extras



A Job creates one or more Pods and will continue to retry execution of the pods untill a specified number of them successfully terminate . As pods successfully complete , the job tracks the successful completions . When a specified number of successful completions is reached , the task is complete. Deleting a job will clean up the pods it created. Suspending a Job will delete it's active Pods untill the Job is resumed again.

Simple case is to create one job object in order to run one pod to completion. The job object will start a new pod if the first pod fails or is deleted. we can also use a job to run multiple pods in parallel.

5:31

## 23. Kubernetes 269. Extras



If we want to run a job for specific or defined time , we need to use a CronJob.

( min , hour , date , month , day) This is the pattern , CronJob follows.

Job just runs , executes and returns the information.

CronJob runs at specific time , executes and returns the information.

8:17

## 23. Kubernetes 269. Extras



DaemonSet ensures that all Nodes run a copy of a Pod. As nodes are added to the cluster , pods are added to them. As nodes are removed from the cluster , those pods are garbage collected. Deleting a DaemonSet will clean up the Pods if created.

Some typical uses of DaemonSet are:

- \* running a cluster storage daemon on every node.
- \* running a logs collection daemon on every node.
- \* running a node monitoring daemon on every node.

In simple case , one DaemonSet , covering all nodes , would be used for each type of daemon. But for a more complex setup might use multiple DaemonSets for a single type of daemon , but with different flags and/or different memory and cpu requests for different hardware types.

10:39

## 23. Kubernetes 269. Extras



Once go through the documentation of the DaemonSet , and where we can see a example where we are declaring the daemon set with tolerations and having a key 'node-role.kubernetes.io/master' and operator exists and effect as no schedule , it mean we are creating a rule for the master node , where we are giving a key for the master node and if it is exists and the effect of it is NoSchedule then we run the following pods on the master node , we can run the pods on the master node , due to we have declared the master node under the tolerations and where the pod will be deployed only when the condition gets true.

The key that we are giving is the key which is used to get the master node address.

In general pods couldn't run on the master nodes , but as we are having toleration declared here for the master node , we can run our pod on the master node , only if the specific conditions are met ( key , operator , effect )

10:45

## 23. Kubernetes 269. Extras



And we declare the resources that are required and the limit that a pod can use inside the resources section of limit and requests.

10:45

## 23. Kubernetes 269. Extras



And we declare the resources that are required and the limit that a pod can use inside the resources section of limit and requests.

12:00

## 23. Kubernetes 269. Extras



kubectl get ds -A => To get the list of daemon sets from all the namespaces.

We can see a daemon set is running on the master node , this is due to declaration of the toleration.

Now if we want to check the pods are created or not on the node , then use the following command.

kubectl get pod -n <namespace-where-daemonset-located>

for example: kubectl get pod -n kube-system

Here after creating pods through daemon set , we will be created 3 pods , where if any of those pods are deleted , the new pods will be created to replica those pods.

In this area, both Daemon set and Replica set work alike.

0:49

## 23. Kubernetes 270. Lens



Lens is an Integrated Development Environment (IDE) that **enables users to connect and manage multiple Kubernetes clusters from Mac, Windows, and Linux platforms**. The feature-rich and intuitive graphic interface allows users to deploy and manage their clusters directly from the console.

5:39

## 23. Kubernetes 270. Lens



When we want to transfer the docker image from one server to another server without using the repository , use docker config files. And when we want to transfer the cluster from one server to the other without using repository , we use kube config files. We can directly add the kubeconfig and open the cluster with help of Lens and use it for visualizing the usage of cluster and we need to change and download metrics in Lens , to keep it working.

Lens works like Kubernetes IDE

0:08

## 8. Networking 62. ISO



As a devops , you need to automate these things , so first we need to know , how these are performed manually.

- 1) Managing cloud computing environment.
- 2) Connecting multiple systems together.

1:41

## 8. Networking 62. ISO



In this video , we are going to talk about

- 1) 'Components' responsible for networking
- 2) 'OSI Model' and layers of OSI Model
- 3) 'Classification' of network based on the geography.

- 4) Networking devices
- 5) Home Network
- 6) IP addresses and the range of the IP addresses
- 7) Protocols
- 8) How DNS & DHCP service work
- 9) Some 'networking commands'

2:51

## 8. Networking 62. ISO



What is computer network?

1. Now a days we have billions and billions of devices connected together on the internet. Devices include both the smartphone and the Internet enabled devices or network enabled devices.
2. Computer network is basically a communication between two or more network interfaces.
3. Every device we use on the network will have an IP address assigned to it's network interface
4. For example , our laptop have ethernet adapter or wireless adapter. Our smartphone have a wireless adapter or a network interface , we can call it. And Network enabled devices , which are used in IOT , using which we can create a communication through which all these devices can exchange data.
5. Basically a computer network is communication between two or more network interfaces.

3:59

## 8. Networking 62. ISO



Components that are used to create a computer network.

- 1) We need two or more computer devices , smartphones or any IOT device.
- 2) We need cables to link these computer devices or the wireless network. So this cable or the wireless network will be connected to the network interface or card on each device. The device knows what to do with the data that it is receiving over the network.
- 3) Switches are used to connect multiple network interfaces together
- 4) Routers are used to connect multiple network together.
- 5) Software or to keep it simple it is called operating system which is running on the device , that can analyze the data that it is received on the network and present it to the user.

These are all the components which is used to create a computer network or a network.

5:08

## 8. Networking 62. ISO



There should be a common platform where every network device can communicate with each other. Similarly , all the network , computers , devices follow set of standards. So the hardware vendor or the app developer or the website developer knows what standard to be followed , so it can be communicated across the world. OSI model has given that standard.

5:49

## 8. Networking 62. ISO



OSI model been the standard for communication between all the network devices.

1. As many people want to communicate with each other , so for the world wide data communication , there should be some standard and systems which are compatible across all the devices , apps , operating systems.
2. We must have a standard communication method and devices also.
3. For example we are having WIFI routers from cisco or juniper or link or any other network device , they exhibit similar kind of communication. Because even the devices also follows some standards which everyone has to follow.
4. People around the world uses computer network to communicate with each other.
5. International Organization of Standardization (ISO) has developed this Open system interconnection (OSI) standard.

6:46

## 8. Networking 62. ISO



OSI Model have 7 layer architecture developed in 1984

8:08

## 8. Networking 62. ISO



Basic elements of a layered model are

1. services: Service is a set of actions that a layer offers to another higher layer.
2. protocol: Protocol is a set of rules that a layer uses to exchange information.
3. interface: Interface is communication between the layers.

10:48

## 8. Networking 62. ISO



For example , check the picture , we are having the sender and the receiver , for instance let's think we are posting a picture on Instagram , it means we are the sender and the

11:35

## 8. Networking 62. ISO



Detailed description of OSI model:

Physical layer:

1. Lowest layer in our OSI model and it is responsible for the actual physical connection between computer A and computer B. Here information is in bits , ones and zeros. And after the computer B physical layer receives the signal, it is going to send this information to the data link layer.
2. Here the transmission of data could be media , signal and binary.

Data Link Layer:

1. Main function of data link layer is to make sure transfer is error free from one node's physical layer to another node's physical layer.
2. At this layer the data is in frames, from a physical layer that is in bits 1 & 0 at datalink layer.
3. Here the physical addressing such as MAC address works.
4. Every device have it's own network interface card , which is used for connectivity for other devices in the computer network.
5. Network interface card will have a physical address which we know which we call it as MAC address, then we present the data to the network layer.

12:02

## 8. Networking 62. ISO



Network Layer:

1. Network layer works for the transmission of data from one node to other which are located in different networks and it works on the IP address.
2. When the data is presented to the network layer , it is assembled in packets and the senders and receivers IP address are placed in the header by the network layer in the header of the packet.
3. So the data entered into the node as bits in physical layer , moved into data link layer and converted as frames and moved to network layers and converted as packets. And it is transferred into Transport layer.

12:33

## 8. Networking 62. ISO



Transport layer:

\* It takes service from the network layer , it provides the service to the application layer.

\* Here the internet connection and the reliability is checked , it is responsible for the end to end delivery of the complete message. It also provides the acknowledgement, and if there is any failure or drop of data , it will retransmit the data.

Session layer:

1. In TCP/IP mode , the application , presentation and session are presented as one layer. These layers are responsible in the security , maintenance of the connection , encryption , decryption and then presenting the data to your apps or to your software like browsers or mobile apps.
2. So , if we send a email , then the application layer is going to produce the data , which would be then transmitted , transferred over the network , and if it is receiving then it is going to assemble all the data together and present to the application.

13:45

## 8. Networking 62. ISO



Details of the devices/apps and protocols:

In layer 1 (Physical layer):

1. used for network access connection
2. ethernet and token ring are used as the protocols
3. hub is the device that has been used in layer 1

In layer 2 (Data link layer):

1. used for network access
2. mac (arp) , rarp address has been used as protocols
3. bridge , layer 2 switch are been used as devices in layer 2

In layer 3 (Network layer):

1. used for internet access
2. IP , ICMP , IGMP protocols need to be used as protocols.
3. router , firewall , layer 3 switch has been used as devices in layer 3
4. Decides the route that needs to be followed to go to the destination.

In layer 4 (Transport layer):

1. used for establishing host to host communication.
2. TCP / UDP protocols are being used.
3. Gateway has been used as devices in Transport layer.
4. In transport layer , we will store the details of the port number of sending user and the receiving user.

14:34

## 8. Networking 62. ISO



In layer 5 (session , presentation , application layers):

1. In general in these three layers are termed as one layer in other than OSI model.
2. This is used for application
3. We use dns , dhcp , ntp , snmp , https , ftp , ssh , telnet , http , pop3 and more are used as protocols in layer 5.
4. Webserver , mail server , browser , mail client are been used as devices in layer 5

15:30

## 8. Networking 62. ISO



In general at senders side , data is been sent from the transport layer to the network layer and network layer determines the path that needs to be followed by the bits.

Network layer have two types of protocols for it

1. Routing Protocol
  - a. It is used to find the shortest path , to transfer the packets from source to destination
  - b. RIP and OSPX
2. Routed Protocol
  - a. Using IP address , it is helpful to transfer from source to destination and in the process router helps to send the data.

15:39

## 8. Networking 62. ISO



Data Link layer , generally ensures the error less delivery , responsible for next hop delivery.

We are having two internal separations in data link layer they are

- 1) Logical Link layer ( This ensures the error free transmission )
- 2) Media Access Control (Ensures that , when the transmission path is free to send the data)

Previously the packet that is in the network layer has been added with the destination and source IP address and now when reaching to the data link layer , we will add both the source and destination mac address.

15:41

## 8. Networking 62. ISO



The physical layer is the one which is also called the transmission media , where it can be ethernet cable , fiber optics , wireless media. We will have the actual hardware present in the physical layer.

15:47

## 8. Networking 62. ISO



Device used in physical layers are Cables

Device used in data link layers are Switches

Device used in Network layer are Router

Transport layer is responsible to application to application delivery

In Session layer , the authentication and login of the application happens.

1:48

## 8. Networking 63. Understanding Networks & IP



Classification of network based on the geography

1. It is generally based on the distance between the network interfaces that are communicating with each other , for example our laptop has a network interface and the google server has a network interface to communicate where this distance between them defines the classification.
2. Local Area Network (LAN): The network devices are very close to each other , probably in a room. We may have a few computers that are connected together through some devices or cables.
3. Wide area network (WAN): It is similar to internet here , the network interface can be far , far away from each other. For example , if we are accessing European data centers through the smartphone that I am having over the Internet. And Internet is the biggest example of Wide area network.

1:53

## 8. Networking 63. Understanding Networks & IP



Classification of network based on the geography:

1. Metropolitan area network (MAN): Metropolitan cities , municipalities which has computers and they are all connected in a network or metro trains computer network system. It mean all the computers under one metropolitan connected into one network and for a metropolitan area network.
2. Campus area network (CAN): It is like office campus or a college campus where we have computers connected together in few acres of land , and the network is restricted with in the office premises. It is also called as Intranet.
3. Personal area network (PAN): It may be our Bluetooth hotspot , our own personal network which has a very small range of network to connect.

If we are setting the local area network , we will need a switch , if we want to connect multiple computers together and printers and some servers together in a small room or floor , in this case we need a switch like we have node 1 (computer) which wants to send some data to node2. So to route the data here from node 1 to node 2 , it uses the switch to find the path , as switch is an intelligent devices which knows the path for node 2 , it sends the traffic over there and node 2 receives the data. It is one of the device which everyone would have seen it , if we are using internet. For example WIFI router , internally WIFI router will have a switch inside that. And for a bigger network , we may have many network interfaces , we can see a lot of cables coming out of them and they are used to create the local area network. Switches are used to create a local area network.

As we know switches connect multiple computers together , router connects multiple networks together.

To make it simple , switch connect multiple computers of same network together and if we want to establish a network connection between two or more networks , for example consider switch connects all computers of same building and if we want to connect two such buildings , we need a router.

A router receives and sends data on computer networks. Routers are sometimes confused with network hubs , modems or network switches. However routers can combine Multiple networks together.

Both switch and router are essential for proper network connection between all the devices.

Basically , our devices which are under same WIFI , can communicate with each other and these devices can communicate with the devices which are not under this WIFI network. Our devices which are under WIFI router , can communicate through switch and these switches are connected by the router to build the global network , which creates Internet.

In general , we connect our devices and our peer devices under a single access point in our home called WIFI and this access points are connected by the help of router , and

the router get the data that we required through modem which in turns connected to internet , so what ever file or data is sending or receiving must be part of the loop. We can transfer data through one device to any other device.

Every device have it's own IP address , it includes home technological devices , switches , router , modem.

6:40

## 8. Networking 63. Understanding Networks & IP



When the data is flowing in Internet , we will be having firewall for data protection and security. And there could be multiple switches and routers in the path , to ensure the continuous network flow if there is any down in any of the network devices.

7:21

## 8. Networking 63. Understanding Networks & IP



For example , in corporate datacenter we will be having many subnets which have it's own IP address . Subnet is a group of network devices which may be subset of devices under the same company and inter-connected through switch and this set up of switch and set of devices which are connected to it is a subnet , we will have multiple subnets in a network and each subnet have it's own IP address and each subnet have it's router which routes to it's superior router and the path is network security protected , and the main router ensures the connection between all the devices under this corporate network.

8:47

## 8. Networking 63. Understanding Networks & IP



In an IP address , we will be having four octets and where each octet is of 8 bits , the first octet is of the left side and the 4th octet is of the extreme right side, four octets is of total 32bits , this syntax is basically for the IPv4 address.

All the ipv4 addresses starts from 0.0.0.0 to 255.255.255.255 and it is equivalent in binaries , which represent by 0's and 1's , these 255 and 0 and the numbers in between are just a representation of a number in binary , for example , we use 8 bits as 255 because  $(1111111)_b = (255)_d$ .

The entire IP addresses , are divided into two types

- 1) Public IP address
- 2) Private IP address

In general Public IP addresses are used by the Internet service providers and the private IP addresses are generally used by the local network design.

Public IP address identifies you to the wider internet so that all the information you are searching for can find you. A private IP address is used within a private network to connect securely to other devices within same network.

Private IP ranges:

Class A: 10.0.0.0 - 10.255.255.255

Class B: 172.16.0.0 - 172.31.255.255

Class C: 192.168.0.0 - 192.168.255.255

Class D: Research IP's

Class E: Multicasting IP's

Class D & E are not used more.

Rest of IP address are of Public IP address.

In a computer network , we would have many devices , which have it's own individual IP address and those computers can be connected to each other through a switch , which have it's own IP address. Switch is used to connect same devices under a same network , but router is used to connect the devices of different networks with the help of the switch's.

In networking and communications area , a protocol is the formal specification that defines the procedures that defines the procedures that must be followed when transmitting or receiving data. Protocols define the format , timing , sequence and error checking used on the network.

Most of the layer 5 , 6 , 7 of TCP/IP layers are dependent on these layers , where some uses TCP protocol and some uses UDP protocol.

TCP :

It is reliable protocol , connection oriented. Which means the information which sends through the network is secured and ensures the data is securely transferred without any errors. For ensuring the error freeness , it performs three ways handshake check. This provides the error detection and retransmission provision. Most applications use TCP for reliability and for guaranteed transmission.

Example for TCP: FTP , HTTP , HTTPS

3:08

## 8. Networking 64. Protocols, ports etc



UDP:

It is unreliable protocol, connectionless . Which means once the data which sends out from the sender doesn't give any feedback to the sender about the status of the data , is it received by the receiver or is there any error in the data. It is much faster than the TCP. There is no , waiting for acknowledgement mechanism. Once the data send by the sender , it doesn't matter for the sender about it is received by the receiver or not. There is no sequencing mechanism for the data units that sender sends , this protocol is suitable for applications where speed matters more than reliability.

Examples of UDP: DNS , DHCP , TFTP , ARP , RARP

5:30

## 8. Networking 64. Protocols, ports etc



Here these are the Protocols and respective default IP address port numbers under UDP and TCP.

6:28

## 8. Networking 64. Protocols, ports etc



Here we can see the mapping of OSI 7 layer model to the TCP/IP 4 layer model.

In TCP/IP model , we divide the layers into 4 parts , they are

- 1) Application layer ( telnet , FTP , DHCP , TFTP , HTTP , SMTP , DNS , SNMP )
- 2) Transport layer ( TCP , UDP )
- 3) Internet layer ( ICMP , ARP , RARP , IP )
- 4) Network Interface

7:23

## 8. Networking 64. Protocols, ports etc



For example , if we want to communicate the tomcat server to the MySQL server , then we must know the IP address that the device is in and the port number through which we can access that application , in general port numbers remain in default. For example let's

think both the source (tomcat server) and the destination ( MySQL server ) are under same IP address (192.168.1.2) but on different ports. Where tomcat server is under the port 8080 and the mysql is under 3306 , so a tomcat server from 192.168.1.2:8080 access the mysql server of destination under 192.168.1.2:3306

8:26

## 8. Networking 64. Protocols, ports etc



Whenever we are creating an application , we must know under which server our service is currently running and under which port. Because , we must give some permissions to our services at the firewall as we need to bypass this security to run the application. It is used to create some extra privilege's under the firewall for running the application.

1:40

## 8. Networking 65. Networking Commands



Networking commands:

1) ifconfig:

- It shows us the all the active network interfaces and their names , IP addresses.
- There will be some loopback addresses which loop back to the same computer , for example we are having here as a <UP,LOOPBACK,RUNNING> where we use 127.0.0.1 address and not tries to connect to any other computer , but connect to it self , as we can see there is not broadcast is declared. This network interface is used when the computer is referring to itself. We can name it as the local network interface.
- And here we are having two other network interfaces , where it is of <UP,BROADCAST,RUNNING,MULTICAST> where we are having a source network address and the destination IP address in the broadcast. So this computer is connected to two other networks. One is 10.0.2 and other is 192.168.40 this is the IP address.

2) ip addr show

- This works same as the 'ifconfig' , it shows all the active network interfaces of this computer.

3:14

## 8. Networking 65. Networking Commands



Network commands:

ping <ip-address>

Previous commands are to show the ip details of the current computer , and if we want to check the status of any other IP address and if we want to see the IP address is connected or not , we use the following command , suppose assume that we want to

connect to the IP address 192.168.40.12 and if we want to see it is connected or not , we need to use the following command

```
ping <ip-address> => ping 192.168.40.12
```

If we use the ping command then it will be sending the ICMP packets to the IP address , that we provide , until we halt it through Ctrl+C. Here using this command we are sending the packets to an IP address and when this IP address receives the packets , it is going to reply back as when we click ctrl-c we can see a status n packets transmitted and n received , m percent packet loss. So this denotes the network connectivity from our machine to this IP address.

This is what ping command does.

4:12

## 8. Networking 65. Networking Commands



Network Commands:

Now if I want to connect to that machine with the name and not with the IP address , then we will need the DNS server or we can make a simple entry for the IP address in /etc/hosts file , this will be usually present in every os. We can enter the details of the IP address and the name that we want to use to connect to that IP address. in /etc/hosts file.

In general ping command shows whether there is a connectivity or not , it can show if there is any packet loss. But if we want to trace the complete path that computer will take to reach the target machine then we must use the Trace route command.

5:44

## 8. Networking 65. Networking Commands



Network commands:

To find the path that ping is using to connect to the destination server or the dns domain name or any website , we must use the tracert command , where we couldn't run this through the virtual machine or the virtual server , we must do it in the local system. For example:

```
tracert www.google.in
```

```
tracert <website-link-or-dest-ip-addr>
```

We can trace the destination website and the hops in between.

Hops mean those are the stages or particularly called as routers that our local system passes to reach the destination server.

7:00

## 8. Networking 65. Networking Commands



Network commands:

netstat -antp

With this command it is going to show all the TCP open ports in the current machine , to see the process id and service name that they are running , we must be of root user. If we use this command it shows some of the ports of TCP is open for anyone to access and there will be details about the process id and service name which is running. If we see :::<any-port-num> , it means this process is accessible for anyone and it is not backholed by any IP address in particular.

7:45

## 8. Networking 65. Networking Commands



For example , if we want to see any open process related to any software or a service use the

ps -ef | grep apache2

ps -ef | grep <service-name>

then using this command we would get details of the process with it's process id , and if we want to see the open ports related to this process id then use the following command

netstat -antp | grep 3336

netstat -antp | grep <process-id>

Then we could see the open port number that we could access the service.

And we can get all the information in a single command

ss -tunlp => which is used to get all the information , either it can be a process id or a service name or a open port details.

8:22

## 8. Networking 65. Networking Commands



Here we are installing the 'nmap' , it is illegal in many countries , so use it legitimatly. usually nmap is used to discover hosts and services on a computer network by sending packets and analyzing the responses , nmap provides a number of features for probing computer networks , including host discovery and service and operating system detection.

nmap allows us to scan our network and discover not only everything connected to it, but also a wide variety of information about what's connected , and what services each host is operating and so on. So this brings the security and privacy issue.

In general hackers use the nmap to gain access to uncontrolled ports on a system. And hacker would need to do to successfully get into a targeted system would be to run nmap on that system , look for vulnerabilities , and figure out how to exploit them.

8:28

## 8. Networking 65. Networking Commands



nmap is a short form of 'Network Mapper'. It is used to scan IP addresses and ports in a network and to detect installed applications. nmap allows network admins to find which devices are running on their network , discover open ports and services and detect vulnerabilities.

9:11

## 8. Networking 65. Networking Commands



Suppose here we can see two commands , one is used to scan the localhost and other is used scan another user.

1) nmap localhost

This command basically shows all the commands that are open in details with port number , state and the service that is running at the port , and it also shows the number of closed ports are present on the target machine, here in our case , we have only one target machine that is our local machine.

2) nmap <ip-address>

And if we try to access another ip address use the same syntax , we get similar details as the above.

9:50

## 8. Networking 65. Networking Commands



1) dig <web-base-url>

We have a command dig (domain information grooper) which is used to get the IP address of a target machine , through the DNS system. This is used to know the DNS is working or not.

2) nslookup <web-base-url>

Here we get the IP details of the host machine and target web based url and we even get IPV6 versioning of IP

10:43

## 8. Networking 65. Networking Commands



**route** command in Linux is used when you want to work with the IP/kernel routing table. It is mainly used to set up static routes to specific hosts or networks via an interface. It is used for showing or update the IP/kernel routing table.

route => To display IP/Kernel routing table (words like default , \_gateway) exists

route -n => To display routing table in full numeric form (There won't be any word like default , \_gateway or any word like that ) All the IP addresses will be shown as the binary.

sudo route add default gw 169.254.0.0 => To assign a default gateway, we use this command to assign a gateway address on which all the packets that do not belong to the network are forwarded.

route -Cn => To list kernel's routing cache information, to route the packets faster , kernel maintains this routing cache information.

10:45

## 8. Networking 65. Networking Commands



sudo route add -host 192.168.1.51 reject => This command is used to reject the routing to a particular host or network. Now if we ping the above IP address , we will get the "Network is unreachable"

ip route => This give the details of the kernel / IP routing table and in this case , we have used IP command.

route del default => To delete the default gateway

ip -4 route / ip -6 route =>To get output related to IPv4 / IPv6

10:47

## 8. Networking 65. Networking Commands



"arp" is used to add or remove a address into kernel arp table. It maps the IP address with the mac address.

12:15

## 8. Networking 65. Networking Commands



"mtr" command mean my traceroute ,where we can see a real time packet exchanges between the server and the domain and we can see the real time packet loss , and if we want to check any internet connectivity issue ,we can check through this, so we can

know either the issue occurred at local or at the destination server or any hops in between.

13:21

## 8. Networking 65. Networking Commands



We can connect to some IP address at a particular port address , if it is open.

Command used:

telnet <ip-address>

This shows the details of the IP address is connected or not , and if connected , it tries to connect to the port through it ,and if it is also successful then it is connected to the IP address else it is not.

2:16

## 11. AWS Part -1 94. What is Cloud Computing



In general scenario , imagine a datacenter with many computers or servers , these all together provide compute resource to an organization or its branches. Suppose if all these computers are virtualized , hypervisors are installed on them, so virtualization team can create virtual machines and allot it to the right individual. So , if employees in organization need a virtual machine or computer to do some computation , they will contact to the virtualization team to provide the computing resource. And using the hypervisor they will access to the virtual memory and host applications.

These virtualization team will work with Ops team will work around the clock to create , manage and maintain these virtual machines or virtual storage as well. So , if the company is big then the administration required to maintain the virtualized platform. This is called virtualization , it is in trend until the cloud computing came in.

2:18

## 11. AWS Part -1 94. What is Cloud Computing



Now a days , we can see a similar virtualized platforms but instead of contacting virtualization team to provide the resources , now we are using some self-help portals , where we are having a website or even command line to access the virtualized platform. So , if we want a virtual machine , we need to log in to the portal , and create a virtual machine yourself. We can create any virtual storage or any virtual service through this portal , and this flexibility is given by the cloud computing.

2:21

## 11. AWS Part -1 94. What is Cloud Computing



We can access our virtualized resource over the network , so now from anywhere we can connect to our cloud portal through APIs, create , maintain and manage virtual resource for ourselves , this is the flexibility in cloud computing. If this is for the organization in particular it is private cloud computing. And if this is done for the public such that anybody can sign up with the cloud provider with their subscription , it is called as public cloud computing. AWS , Azure , google cloud are some names in the public cloud computing environment.

2:46

## 11. AWS Part -1 94. What is Cloud Computing



What is cloud computing?

Cloud computing is the on-demand delivery of IT resources over the Internet with pay as you go pricing. Instead of buying , owning and maintaining physical data centers and servers , you can access technology services , such as computing power , storage and databases, on an as-needed basis from a cloud provider like Amazon Web Services (AWS).

90% of organizations are using cloud computing.

Benefits of cloud computing:

1) Agility (It is easy to procure the resources faster now , ever than before , we no need to face the overhead of building the infrastructure by ourselves , and even if we don't want those resources , we can release them to the cloud providers , so we can save the maintenance cost and the establishment cost , we can be clear with our ideas)

4:03

## 11. AWS Part -1 94. What is Cloud Computing



Types of cloud computing:

- 1) Infrastructure as a Service (IaaS)
- 2) Platform as a Service (PaaS)
- 3) Software as a Service (SaaS)

We will be using different services of AWS cloud that may fall under Infrastructure as a Service.

If we have virtual machines , we need to manage the operating system of that virtual machine , for example EC2 service of AWS , it is an example of infrastructure as a service.

Here in above , we are having virtual machines , we have given an infrastructure (memory , ram , processor, etcetera ) and we need to manage only the operating system in it , so this is the example of Infrastructure as a service (IAAS).

We will be having platform as a service , where we don't need to worry about the virtualized platform even , we will have a platform provided for the service for example for oracle database , we will have AWS RDS Service , so everything will be ready , and we can use the oracle database through the AWS RDS Service directly without any manual management from our side like IAAS. Here we don't need to create a virtual machine , we can simply use the AWS RDS Service. And the AWS does everything to us for provisioning our Oracle database.

And coming to software as a service , it is much easier , we need to just subscribe and start using them. We will have all these three types of services in AWS.

If we want to use AWS free tier account , we must have set up a AWS free tier account , set an alarm for billing and set up an IAM user. AWS is the biggest cloud computing provider or public cloud computing provider.

AWS is spread across the world , across the globe , it has its presence in so many countries , which is called regions, and it is growing everyday , within the region we will have multiple zones , think of the zones as multiple data centers , one zone have multiple data centers. And in one region , we may have minimum two zones and maximum. Some cases we may have six zones in a region where a zone have multiple data centers , which are clustered. As of 2020 there are 175 fully featured services.

In total as of 2020 , we are having 77 availability zones with 24 geographical regions , as we know 1 geographic region can have multiple zones. Each zone is a multiple cluster data centers.

<https://infrastructure.aws/>

This link has details of global regions , datacenters and zones.

One availability zone will have many and many data centers. Availability zones have the regions , zones and local zones , these are the type of datacenters from AWS. Which will place computer storage database and some other services closer to a large population , like in a big city where a lot of customers will have the point of presence for their data.

And this is going to be the edge location for the customers , so it is going to cache store the data.

AWS content delivery network where it is going to cache the data which will be used by the users.

6:02

## 11. AWS Part -1 95. Introduction



If we have multiple zones , we can have high availability for our infrastructure. So when we are planning to have web servers , we can place 2 web servers in one zone and other 2 web servers in another zone , so we can ensure the network in such a way that , both the users around the particular zones have high availability, Even if a zone goes down or become slower , then other part of the infra will be up and running.

If we want to become global we can have multiple regions that can support the global audience and even we can set the disaster recovery , so even one datacenter of a zone or a region is down, the users will be redirected to other regions and the data also can be backup to other datacenters and redirect to that datacenter , such way we can do the disaster recovery between two regions.

6:11

## 11. AWS Part -1 95. Introduction



Regions:

AWS has the concept of region , which is a physical location around the world where we cluster data centers , we call each group of logical data centers an availability zone.

Each AWS Region consists of minimum 3 , isolated and physically separate Availability Zones within a geographical area. Some of the other cloud providers define the Region as the single data center , but AWS provides the multiple AZ design of every AWS Region and offers the advantages for its customers , in terms of uptime and scalability.

Each Availability Zone has independent power , cooling and physical security and is connected via redundant , ultra low latency networks.

AWS customers focused on high availability , can design their applications to run in multiple AZ's to achieve even greater fault-tolerance. AWS Infrastructure regions meet the highest levels of security , compliance and data protection.

AWS opens multiple regions to ensure low latency data transfer and country data security.

6:13

## 11. AWS Part -1 95. Introduction



Availability Zones:

An Availability Zone (AZ) is one or more discrete data centers with redundant power, networking, and connectivity in an AWS Region. AZs give customers the ability to operate production applications and databases that are more highly available, fault tolerant, and scalable than would be possible from a single data center. All AZs in an AWS Region are interconnected with high-bandwidth, low-latency networking, over fully redundant, dedicated metro fiber providing high-throughput, low-latency networking between AZs. All traffic between AZs is encrypted. The network performance is sufficient to accomplish synchronous replication between AZs. AZs make partitioning applications for high availability easy.

6:15

## 11. AWS Part -1 95. Introduction



Availability Zones:

If an application is partitioned across AZs, companies are better isolated and protected from issues such as power outages, lightning strikes, tornadoes, earthquakes, and more. AZs are physically separated by a meaningful distance, many kilometers, from any other AZ, although all are within 100 km (60 miles) of each other.

6:19

## 11. AWS Part -1 95. Introduction



AWS Services:

AWS offers a broad set of global cloud-based products including compute, storage, database, analytics, networking, machine learning and AI, mobile, developer tools, IoT, security, enterprise applications, and much more.

6:20

## 11. AWS Part -1 95. Introduction



**The following core services are included in all Region launches:** Amazon API Gateway, Amazon Aurora, Amazon CloudWatch, Amazon DynamoDB, Amazon EC2 Auto Scaling, Amazon ElastiCache, Amazon Elastic Block Store (EBS), Amazon Elastic Compute Cloud (EC2), Amazon Elastic Container Registry (ECR), Amazon Elastic Container Service (ECS), Amazon Elastic MapReduce (EMR), Amazon OpenSearch Service, Amazon EventBridge, Amazon Kinesis Data Streams, Amazon Redshift, Amazon Relational Database Service (RDS), Amazon Route 53, Amazon Simple Notification Service (SNS), Amazon Simple Queue Service (SQS), Amazon Simple Storage Service (S3), Simple Workflow Service (SWF), Amazon Virtual Private Cloud (VPC), AWS Application Auto Scaling, AWS Certificate Manager, AWS CloudFormation, AWS CloudTrail, AWS CodeDeploy, AWS Config, AWS Database Migration Service

6:23

## 11. AWS Part -1 95. Introduction



## **The following core services are included in all Region Launches:**

AWS Direct Connect, AWS Identity & Access Management (IAM), AWS Key Management Service, AWS Lambda, AWS Marketplace, AWS Health Dashboard, AWS Step Functions, AWS Support, AWS Systems Manager, AWS Trusted Advisor, AWS VPN, and Elastic Load Balancing (ELB).

## **Following services usually launch within 12 months of new Region launch:**

Amazon Athena, Amazon CloudFront, Amazon Elastic File System (EFS), Amazon Elastic Kubernetes Services (EKS), Amazon GuardDuty, Amazon Kinesis Firehose, Amazon MQ, Amazon SageMaker, AWS Backup, AWS Batch, AWS Certificate Manager Private Certificate Authority, AWS Chatbot, AWS CodeBuild, AWS Console Mobile App, AWS Directory Service, AWS Fargate, AWS Glue, AWS LakeFormation, AWS License Manager, AWS Organizations, AWS Resource Access Manager (RAM), AWS Secrets Manager, AWS Security Hub, AWS Service Catalog, AWS Storage Gateway, AWS Transit Gateway, AWS WAF, and AWS X-Ray.

7:02

**11. AWS Part -1** 95. Introduction



Local Zones:

AWS Local Zones place compute, storage, database, and other select AWS services closer to end-users. With AWS Local Zones, you can easily run highly-demanding applications that require single-digit millisecond latencies to your end-users such as media & entertainment content creation, real-time gaming, reservoir simulations, electronic design automation, and machine learning.

Each AWS Local Zone location is an extension of an AWS Region where you can run your latency sensitive applications using AWS services such as Amazon Elastic Compute Cloud, Amazon Virtual Private Cloud, Amazon Elastic Block Store, Amazon File Storage, and Amazon Elastic Load Balancing in geographic proximity to end-users. AWS Local Zones provide a high-bandwidth, secure connection between local workloads and those running in the AWS Region, allowing you to seamlessly connect to the full range of in-region services through the same APIs and tool sets.

7:07

**11. AWS Part -1** 95. Introduction



AWS Wavelength enables developers to build applications that deliver single-digit millisecond latencies to mobile devices and end-users. AWS developers can deploy their applications to Wavelength Zones, AWS infrastructure deployments that embed AWS compute and storage services within the telecommunications providers' datacenters at the edge of the 5G networks, and seamlessly access the breadth of AWS services in the region. This enables developers to deliver applications that require single-digit millisecond latencies such as game and live video streaming, machine learning inference at the edge, and augmented and virtual reality (AR/VR).

AWS Wavelength brings AWS services to the edge of the 5G network, minimizing the latency to connect to an application from a mobile device. Application traffic can reach application servers running in Wavelength Zones without leaving the mobile provider's network. This reduces the extra network hops to the Internet that can result in latencies of more than 100 milliseconds, preventing customers from taking full advantage of the bandwidth and latency advancements of 5G.

### AWS Outposts

AWS Outposts bring native AWS services, infrastructure, and operating models to virtually any data center, co-location space, or on-premises facility. You can use the same AWS APIs, tools, and infrastructure across on-premises and the AWS cloud to deliver a truly consistent hybrid experience. AWS Outposts is designed for connected environments and can be used to support workloads that need to remain on-premises due to low latency or local data processing needs.

When we are running a project , sometimes we will have some compliances that we must follow , for example data residency. So those should be fitting in with the AWS compliance and for that reason , we have regions at different different countries. And for our business , we can use the global infrastructure , so we no need to build the data centers around the world.

So if we are having an application , and if we want to have the high availability for that application , we can have multiple availability zone and we can distribute our infrastructure among multiple availability zones in a region to get high availability.

If we are launching a virtual machine which we call it as instance , we need to select the availability zone or we can let AWS decides the availability zone for us.

When we are designing application , the infrastructure layout , we can select the multiple zones in a region and distribute our infra and the traffic will also that is coming towards your infra distributed.

### AWS Management Console

\* We will have all our services of Amazon declared in AWS Management Console.

\* We no need to learn every service of AWS , if we are a system admin or a devops , we want a virtual machine , so that we can use a EC2 service. And we will have services related to Quantum technology , Machine learning , Analytics , Media Services , Storage , Mobile , AR & VR , Blockchain , Robotics , IOT etc.

Mostly in the course , it will be focused on the sysops and devops services and some developer services like EC2 service , Bean-stack Service which are related to the devops. Storage services we will see S3 , EFS , Glacier. Database services like RDS and also elastic cache. Networking services like VPC , Cloud Front , Route 53 and Some developer tools like Code Commit Code , Artifact Code , Build Code , Deploy Code Pipeline. Monitoring services like Cloud Watch and also some compliance services.

10:25

## 11. AWS Part -1 95. Introduction



In AWS Management Console , In services tab of AWS , we will be having a zones location , where whatever data that we store in the server , it redirects to this particular zone and store in those datacenters of that location that we selected. It depends on the audience requirements , on where we need to store the data and in which datacenter.

11:12

## 11. AWS Part -1 95. Introduction



And in the free tier , we will have only some locations to work on using the AWS Services , but if we want to access the zones of other servers , we need to pay the amount for the service. Most of the free services are only present under US region.

1:25

## 11. AWS Part -1 96. Ec2 Introduction



EC2 is one of the most popular service of AWS , it is a short form of Elastic Compute Cloud. It is basically about virtual machines and related services. EC2 provides us the web services for managing and provisioning virtual machines inside Amazon Cloud or Amazon's data centers.

We can easily scale up or scale down our resource in EC2. For example if we have 8GB of RAM , we can scale it up to 16GB or even we scale down to 4GB. Similarly we can do for CPU network and storage.

We are only going to pay only for what we are using and the pricing differs from the resources that we use.

EC2 service is very famous for getting integrated with other services like S3 , EFS , RDS , DynamoDB and Lambda.

1:26

## 11. AWS Part -1 96. Ec2 Introduction



If we are using EC2 instance we are paying on hourly basis or seconds basis also sometime. We can even reserve the capacity for 1 to 3 years for which we can avail discounts. Some will be like spot instance , it is basically getting unused ec2 instance on the spot from other resource holders , for which we can get more discount. But after we pay the money and the resource of the original holder expires our instance will be gone. We can also have a dedicated physical server for us , but it increase cost for us , as that is dedicated for us , even we use or if we didn't.

2:23

## 11. AWS Part -1 96. Ec2 Introduction



If we don't want to go with the complete physical server , instead of the virtual machine , we can go for it , but it is very expensive.

3:38

## 11. AWS Part -1 96. Ec2 Introduction



Let's talk about the components in the EC2 instance or EC2 service.

### **AmazonMachineImage (AMI):**

We need AMI , Amazon machine image , this is ready made virtual machine like vagrant , we have seen vagrant boxes , here we have AMI , here we will have a huge list to choose from. This is something like docker images , where we use these images as base image for the container , in here Amazon Machine Image is used as base image for creating a EC2 instance.

### **InstanceType:**

Instance type will be based on the size of our instance , the compute resource , how much amount of CPU , RAM , Network Speed or Storage Speed. We will have a wide variety to choose from.

### **Elastic Block Storage (EBS):**

This is the storage , So on AMI we primarily have storage like 8BG for Linux machine and 30GB for Windows Machine. And then we will be having different sizes of AMI which use the EBS Service , we can even add our own storage. EBS is really the virtual hard disks on which you can store our operating system and our data.

4:34

## 11. AWS Part -1 96. Ec2 Introduction



Tags:

Tag is a simple label consisting of a customer - defined key and an optional value that can make it easier to manage , search for and filter resources. We will have a Tag to every resource in AWS. We can have some simple key value pairs and define tags like name of

an ec2 instance , the project , the customer who owns this or the environment. It is good for filtering and billing.

#### Security Group:

Security Group will acts a virtual firewall that controls the traffic for one or more instances. We will have security group on every EC2 instance. This is basically firewall which is used for inbound and outbound traffic.

4:39

## 11. AWS Part -1 96. Ec2 Introduction



To login to EC2 instance will need a key pair for linux machine , these are basically ssh keys , ssh log in keys , we will have both public key and the private key , as we have in bash scripting. So we need to create a key while creating an instance and download the key without fail for logging in to the instance. The private and public key will be present in EC2 instance , so we can use those keys to access the instance later. We can use the key to do an SSH for Windows machine. we use the key to generate the password.

Creating EC2 instance is easy.

4:43

## 11. AWS Part -1 96. Ec2 Introduction



This is the process of creating Amazon EC2 instance , First we need to choose what AMI we wanted , and we need to select the instance type and later we need to set some configuration for the instance , like network , it's permission and some scripts to execute.

After configuring the instance , we need to decide on the storage , this is where EBS comes in , we can add extra storage if we need or we can go with the default AMI Storage. Then we select the tags that we want ,give key value pairs and do as much as tagging that we want , later create security group , which acts as a firewall , finally we review the instance and create the instance.

Amazon EC2 instance creation:

Choose an AMI -> Choose an Instance Type -> Configuring the Instance -> Adding Storage -> Adding tags -> Configure Security Group -> Review -> Create Instance

3:38

## 10. Bash Scripting 73. Introduction



Most of the errors that occur in shell scripting is due to typographical mistakes , it may a spell mistake or a space that we have added or removed somewhere , so we may get error. If there is any error once go through to the script letter by letter to get the error out.

Why scripting is introduced?

We have

4:58

## 10. Bash Scripting 75. First Script



In shell script,  
'uptime' is used to check the system uptime  
'free -m' is used to check the memory utilization  
'df -h' is used to check the disk utilization

0:01

## 10. Bash Scripting 78. Variables



If we don't want any output from any shell script we do it by redirection of the output as "command > /dev/null"

1:06

## 10. Bash Scripting 78. Variables



Variable declaration:

```
SKILL="DevOps"  
echo $SKILL  
PACKAGE="httpd wget unzip"  
yum install $PACKAGE -y
```

2:34

## 10. Bash Scripting 78. Variables



Renaming the script:

```
mv firstscript.sh 1_firstscript.sh  
mv websetup.sh 2_websetup.sh
```

Copy the script:

```
cp firstscript.sh 1_firstscript.sh
```

We have to keep the variables inside the shell script and use it in the shell commands that we write in the script.

1:31

## 10. Bash Scripting 79. Command line arguments



It is the way of accessing the command line variables in the bash script

2:17

## 10. Bash Scripting 79. Command line arguments



If we retrieve a variable which is not declared earlier , then we get an empty string as the value for it.

2:39

## 10. Bash Scripting 79. Command line arguments



When we are passing values through command line then , the 0th value of the argument is the filename , that we are calling and the rest of all the arguments that we pass except the file name.

`./4_args.sh first_arg second_arg third_arg`

0 th value => `./4_args.sh`

1 st value => `first_arg`

2nd value => `second_arg`

3rd value => `third_arg`

0:12

## 10. Bash Scripting 80. System Variables



`$0` => Name of the Bash script.

`$1 - $9` => The first 9 arguments to the Bash script.

`$#` => How many arguments were passed to the Bash Script.

`#@` => All the arguments supplied to the bash script.

`$?` => The exit status of the most recently run process.

`$$` => The process ID of the current script.

`$USER` => Username of the user running the script.

`$HOSTNAME` => The hostname of the machine the script that is running on.

`$SECONDS` => The number of seconds since the script was started.

`$RANDOM` => Returns a different random number each time it is referred to.

`$LINENO` => Returns the current line number in the Bash Script.

0:56

## 10. Bash Scripting 81. Quotes



A value can be assigned to a variable through a single quote or a double quotes , there will be no difference in the output while we are echoing the variable. But when we are using a variable in a sentence and storing the whole sentence in some other variable then double quotes will understand the variable inside the sentence but not single quotes.

1:40

## 10. Bash Scripting 81. Quotes



```
[root@scriptbox scripts]# SKILL="DevOps"  
[root@scriptbox scripts]# echo $SKILL  
DevOps  
[root@scriptbox scripts]# SKILL='DevOps'  
[root@scriptbox scripts]# echo $SKILL  
DevOps  
[root@scriptbox scripts]# echo "I have got $SKILL skill."  
I have got DevOps skill.  
[root@scriptbox scripts]# echo 'I have got $SKILL skill.'  
I have got $SKILL skill.  
[root@scriptbox scripts]#
```

1:54

## 10. Bash Scripting 81. Quotes



Double quotes persists the special meaning of the \$var , but single quotes will not persist the special meaning.

2:31

## 10. Bash Scripting 81. Quotes



If we want to print '\$' on the sentence then , we have to add backward slash before the special character (\) .

```
[root@scriptbox scripts]# VIRUS=" covid19"  
[root@scriptbox scripts]# echo "Due to $VIRUS virus company have lost $9 million."  
Due to covid19 virus company have lost million.  
[root@scriptbox scripts]# echo 'Due to $VIRUS virus company have lost $9 million.  
Due to $VIRUS virus company have lost $9 million.  
[root@scriptbox scripts]# echo "Due to $VIRUS virus company have lost \$9 million."
```

Due to covid19 virus company have lost \$9 million.

[root@scriptbox scripts]#

1:03

## 10. Bash Scripting 82. Command Substitution



In general , when we want to store any command's output into a variable then we must store the specific command under backticks (`) rather than the single and double quotations and store the command on the variable , so we can use the output of that command in further executions.

```
[root@scriptbox scripts]# uptime  
13:34:47 up 1:31, 1 user, load average: 0.00, 0.01, 0.05  
[root@scriptbox scripts]# UP="uptime"  
[root@scriptbox scripts]# echo $UP  
uptime  
[root@scriptbox scripts]# UP= uptime  
[root@scriptbox scripts]# echo $UP  
13:35:19 up 1:32, 1 user, load average: 0.00, 0.01, 0.05
```

1:38

## 10. Bash Scripting 82. Command Substitution



Instead of the backticks (`) we can even use the ( \$( ) ) for storing value in a variable.

```
[root@scriptbox scripts]# CURRENT_USER=$(who)  
[root@scriptbox scripts]# echo $CURRENT_USER  
vagrant pts/0 2021-09-04 12:041 (10.0.2.2)
```

3:57

## 10. Bash Scripting 82. Command Substitution



Example which shows the usefulness of free , grep and awk command.

```
[root@scriptbox scripts] # free -m  
total used free shared buff/cache available  
Mem: 990 113 591 6 286 734
```

```
Swap: 1023      0      1023
```

```
[root@scriptbox scripts] # free -m | grep Mem
```

Mem:	990	113	590	6	286	734
------	-----	-----	-----	---	-----	-----

```
[root@scriptbox scripts] # free -m | grep Mem | awk '{print $4}'
```

590

```
[root@scriptbox scripts] # FREE_RAM= ` free -m | grep Mem | awk '{print $4}'`
```

0:13

## 10. Bash Scripting 83. Exporting Variables



Scope of the variables is local on the script , And if we want to pass those variables which are connected to the current script , we use the export command for exporting those variables.

Suppose if we are using a variable in parent shell and want to use the same variable in child shells under the parent shell , either the child shell can be some shell script file or python file , whatever which want to access the environmental variable. we couldn't use it without exporting the variable , we need to export the variable to use it in the child shells.

```
export VARNAME="VARVALUE"
```

```
echo $VARNAME
```

3:38

## 10. Bash Scripting 83. Exporting Variables



sudo -i => It is used to login to the root user.

5:29

## 10. Bash Scripting 83. Exporting Variables



In general , if we want our variable to be globally present on the user that we are working on , we need to edit the .bashrc file , this is a script which get's executed everytime when the terminal starts. So , if we want any variable which needs to be persisted on the server , we need to edit the .bashrc file , so when the server restarts , this bashrc will execute and the variable will be declared automatically , without repeated explicit declaration.

There are two files that are one of the important for the server , they are .bashrc and .bash\_profile. And if we want to have a variable accross root user and the child users or any user , we need to edit this file (/etc/profile).

At first /etc/profile will be sourced and next .bashrc will be sourced for any user.

So in default the terminal uses the variables which are declared in /etc/profile as a default across all the users. And if any variable is declared in .bashrc , the /etc/profile will be overwritten by the .bashrc

7:25

## 10. Bash Scripting 83. Exporting Variables



If we want to have a variable permanent for all the users , we need to edit the /etc/profile and if we want to have a variable permanent for only current user , we need to edit ~/.bashrc which presents in home directory.

2:03

## 10. Bash Scripting 84. User Input



```
#!/bin/bash
echo "Enter your skills:"
read SKILL
echo "Your $SKILL skill is in high Demand in the IT Industry."
read -p 'Username: ' USR
read -sp 'Password: ' pass
echo
echo "Login Successfull: Welcome USER $USR,"
```

This is the way to take a user input from the user , and we can also take the user input , by displaying the prompt using 'read -p' and for passwords , we can use 'read -sp' which is used for not displaying the characters what we are giving as an input.

3:10

## 10. Bash Scripting 84. User Input



In devops , we must avoid the user interaction with the images or scripts , because user interaction generates the error. The main purpose of the devops is to run the application in the background , this doesn't changes the code of the application.

4:32

## 10. Bash Scripting 85. Decision Making part1



### **Example for Conditional Statements:**

```
#!/bin/bash
```

```
read -p "Enter a number: " NUM
echo
if [ $NUM -gt 100 ]
then
    echo "We have entered in IF block."
    sleep 3
    echo "Your Number is greater than 100"
    echo
    date
else
    echo "You have entered number less than 100./7
fi
echo "Script execution completed successfully."
```

1:35

## 10. Bash Scripting 86. Decision Making part2



```
[root@scriptbox scripts]# ip addr show | grep -v LOOPBACK | grep -ic mtu
2
grep -v <Text> => This gives all the lines except the line which have the following <Text>
grep -ic <Word> => Here it provides occurrence (c) of a word (<Word>) case sensitively (i)
```

3:29

## 10. Bash Scripting 86. Decision Making part2



### If-else-if Conditional Statements

```
#!/bin/bash
value=$(ip addr show | grep -v LOOPBACK | grep -ic mtu)
if [ $value -eq 1 ]
then
    echo "1 Active Network Interface found. "
elif [ $value -gt 1 ]
then
    echo "Found Multiple active Interface. "
else
    echo "No Active interface found."
fi
```

2:42

**10. Bash Scripting** 88. Loops

```
#!/bin/bash
for VARI in java .net python ruby php
do
echo
“Looping..... .
sleep 1
echo "#####
echo “Value of VARI is $VARI. ”
echo "#####
date
done
```

4:21

**10. Bash Scripting** 88. Loops

```
#!/bin/bash
MYUSERS="alpha beta gamma"
for usr in $MYUSERS
do
echo “Adding user $usr. ”
useradd $usr
id $usr
echo "#####
done
```

4:25

**10. Bash Scripting** 88. Loops

```
#!/bin/bash
for (( c=1 ; c<=5; c++ ))
do
echo “Welcome $c times”
done
```

## For loop example

```
#!/bin/bash
for (( ; ; ))
do
    echo "Infinite Loop"
done
```

## Infinite For loop example

2:57

10. Bash Scripting 89. While Loops



```
#!/bin/bash
counter=0
while [ $counter -lt 5 ]
do
    echo "Looping...."
    echo "Value of counter is $counter. "
    counter=$(( $counter + 1 ))
    sleep 1
done
echo "Out of the loop"
```

## While loop

```
#!/bin/bash
counter=2
while true
do
    echo "Looping...."
    echo "Value of counter is $counter. "
    counter=$(( $counter * 2 ))
    sleep 1
done
echo "Out of the loop"
```

## Infinite While loop

0:06

10. Bash Scripting 91. SSH Key Exchange



In general , if we want to execute commands in different server , then we must login to the server through ssh , and we need to type the password for it. So , to bypass this step and make this step feasible , we must do ssh key exchange. Normally , when we try to login to the other server using ssh , we do it through password way , but we can also do it through the ssh key exchange , without necessity of the password. To make this process workout , we must do the following steps.

command 1> ssh-keygen

> Then leave the directory as the default for the location of id\_rsa , and leave the passphrase unchanged , if not needed.

> Then we will get two id\_rsa keys

1) id\_rsa

2) id\_rsa.pub

Where id\_rsa.pub is the lock for the server and the id\_rsa is the key for that server , so we must copy the id\_rsa from the place that we want to use the server that we want to use remotely without using password.

1:33

## 10. Bash Scripting 91. SSH Key Exchange



Here , as said we will have two id\_rsa produced one is id\_rsa.pub and other is id\_rsa , so we are going to keep id\_rsa.pub on the server that we want to access remotely and id\_rsa on the local server that we want to use to access the remote server , so the local server can login to remote server using ssh without any password needed to be given.

> ssh-copy-id remote-username@remote-servername

2:26

## 10. Bash Scripting 91. SSH Key Exchange



For every server , if we want to use the ssh , through ssh-keygen , we would get two id\_rsa keys. one is of id\_rsa.pub and other is id\_rsa , we use this id\_rsa.pub , store on the remote servers that we want to login password less. And as id\_rsa.pub is the lock for the server , id\_rsa is the key for that id\_rsa.pub , so to unlock any server through ssh and id\_rsa , we need both of the files , one as the lock and other as the key.

This is the command which we login from the local server.

We execute: ssh remote-username@remote-servername

Actually we need to execute: ssh -i .ssh/id\_rsa remote-username@remote-servername

3:24

## 10. Bash Scripting 91. SSH Key Exchange



When the id\_rsa key matches with the id\_rsa.pub then the lock for the server unlocks and login to the server passwordless.

7:29

## 10. Bash Scripting 92. Finale Part1



If you want to push output to the null , we must redirect the output to the /dev/null

```
yum -y install python > /dev/null
```

0:15

## 10. Bash Scripting 93. Finale Part2



command > /dev/null

The > /dev/null redirection operator directs output to the null device, which is a special file that discards all data written to it. It is commonly used to suppress output from a command

command &> /dev/null

The '&> /dev/null' redirection operator is similar, but it redirects both standard output and standard error to the null device. This can be useful if you want to suppress both types of output from a command.

This will run the command and discard both standard output and standard error.

Both '> /dev/null' and '&> /dev/null' can be useful for running a command in the background or for testing a command without being bothered by its output.

It's worth noting that the '&>' operator is not available in all shells. In shells that don't support it, you can use '> /dev/null 2>&1' to achieve the same effect. This redirects standard error (2) to the same location as standard output (1), which is then redirected to the null device using the > operator.

0:40

## 10. Bash Scripting 93. Finale Part2



If we want to move files into Linux machine we use scp , scp is pre installed if ssh installed. So the syntax would be

Syntax we are using:

```
scp <filename> remote-username@remote-server:/path-to-directory
```

Syntax we should be using:

```
scp -i ~/.ssh/id_rsa <filename> remote-username@remote-server:/path-to-directory
```

If there is a key exchange , there won't be any password prompt , but if there is an exchange in the key , then we don't have any authentication issue , we can directly login

to the server.

5:12

## 10. Bash Scripting 93. Finale Part2

```
1 | #!/bin/bash
2 | USR=' devops
3 | for host in "cat remhosts"
4 | do
5 | echo
6 | echo "######
7 | echo "Connecting to $host"
8 | echo "Pushing Script to $host"
9 | scp multios_websetup.sh $USR@$host:/tmp/
10 | echo "Executing Script on $host"
11 | ssh $USR@$host sudo /tmp/multios_websetup. sh
12 | ssh $USR@$host sudo rm -rf /tmp/multios_websetup. sh
13 | echo "######
14 | echo
15 | done
```

Here we can see , two main commands one is 'scp' and another is 'ssh' , scp is used for copying the file into the remote server and ssh is used to login to that server.

0:28

## 14. GIT 132. Introduction



\* Git is a version control system , and it is also called as a developer tool. And as a devops guy we need to learn it because , we need to work with the developers , so we need to understand their language.

\* Whenever the devops guy make a code change , we need to fetch the code , build that , test that and deploy that into servers. So , for that purpose , we need to understand the version control system like git.

\* We also need to write scripts and code and configuration files , so we can use GIT as version control system for our automation code.

4:24

## 14. GIT 132. Introduction



In general , for every devops engineer , they would write multiple lines of code at any day , so there would be a situation , where you must change multiple files of the code , where sometimes , you will lost track of it. So , for those type of the cases , we need to have a solution , in our cases , we must have two types of solutions. One is , we must have the local backup of files , and the second is we must update our working versions into the git repository , so , if we ruin the code , we can rollback to the previous working version of the application. The main usage of the git is just to manage and maintain the code.

7:20

## 14. GIT 132. Introduction



If we are having multiple files in the same directory , then it will be clumsy to look , so better , we can achieve those all files , and it will be easy if they have a specific pattern in the filename.

```
tar czvf <filenames>
```

But when are working with those files , we can't find difference between the lastest version and currrent version. So , to find and make it easy , we use version control system , so that we can spot the difference between the latest version and the current version using github,

9:02

## 14. GIT 132. Introduction



Normally , version control system is all about managing multiple versions of Documents , Programs , Websites etc. , And it is used tracks history of collection of files , which is used to track every modification to the code in a special kind of database.

10:05

## 14. GIT 132. Introduction



What is the use of Version Control System (VCS):

- \* For Individual Help:
  - \* It is used to going back to earlier versions.
- \* For working with team:
  - \* Greatly simplifies concurrent work and helps with efficient merging changes.
  - \* Management of changes to files
    - \* Keep track of what changes occurred , and allows people to work together.

10:39

## 14. GIT 132. Introduction



- \* Localized version control system keeps local copies of the files.
- \* In centralized source control , there is a server and a client, the server is the master repository which contains all the versions of the code.

10:52

**14. GIT** 132. Introduction

In Localized VCS , we are having a Localized Version Control system , where we can checkout to any file , and for which we can have multiple versions of that file , so which is useful to rollback to any previous working version or update any current working version. Here the version details and everything will be on our system , so we couldn't find the code on any server. It is just as a local backup tool.

11:27

**14. GIT** 132. Introduction

**Centralized VCS:**

Our code will be present at the central repository , which is the server , and the teammates who are working on that central repository will have their own working copy on their system , so they can make the necessary changes on the local working copy , and after basic verification of the code they written manually , they can push the code into the central repository. And it is suggested to pull the code before we push the code into the repository , so we will get all the necessary or happened changes which occurred during our code development. We basically commit the changes into the repository , so the repository will be up to date.

12:07

**14. GIT** 132. Introduction

**Centralized VCS:**

- \* Here a central repository holds the official copy of the code , the server maintains the sole version history of the repo. Here we can make a checkout of that branch into local , we can make changes of it but our changes are versioned , our changes will be on our local system , but when our changes are done, we can get back in to the server and our check in increments the repository version. In short , the changes on the checked out branch doesn't change the original branch code , but when we check in the code on to the branch of the server , our changes are sync on the server. Here we are having two copies , local copy and centralized copy.

12:34

**14. GIT** 132. Introduction

Both localized and the centralized version control system have their own negative affects.

- \* In localized version control systems it is individual computer and in centralized version control system it is the server machine.

- \* In a centralized version control system it is the server machine , both systems makes it also harder to work in parallel on different features. E.g.: Git , mercurial etc.
- \* In centralized VCS , if the central repository have any server issue , then there will be halt in development , and it is similar to localized repository , if the local server have the issue.

11:45

## 14. GIT 132. Introduction



In a centralized version control system, there is a single central repository where all versions of the files are stored, and developers check out files from this central repository and then check them back in after making changes. This central repository is the single source of truth for the project, and all changes must be made through this central repository.

In a localized version control system, each developer has their own copy of the repository on their local machine, and changes are made and committed locally. These changes can then be shared with other developers by pushing and pulling changes between repositories. Localized version control systems are often called distributed version control systems because the repository is distributed among the developers rather than being centralized in a single location.

11:51

## 14. GIT 132. Introduction



One key difference between centralized and localized version control systems is how they handle conflicts. In a centralized system, conflicts can only occur when multiple people try to check in changes to the same file at the same time. The version control system will alert the developers to the conflict and they will need to resolve it before the changes can be checked in. In a localized system, conflicts can occur anytime two developers have made changes to the same file and their changes overlap. The version control system will alert the developers to the conflict, but they can choose to resolve it immediately or later, depending on their workflow.

12:45

## 14. GIT 132. Introduction



Basically , in a centralized Version Control system , we will be having a central repository , where every other user in central VCS will be having a copy of the file that he want to edit and check in the changes into the repository which is present in the central server.

And when coming to localized version control system , we will be having a central repository and we will be having local copy of that repository into our system , so we can edit the changes into our system and check it's working status and if everything is Ok , we can push the changes into the central repo and if other users want to know those

changes then they must pull the changes from the repo. Where coming to the centralized version control system , it will be only one repository where everyone is editing the main repo and in Localized version control system , we will be having multiple repositories , where one is the main repository and others are the copy of the main repositories which is present in local.

12:47

## 14. GIT 132. Introduction



So , we couldn't work on multiple features of the application in the repositories in these central VCS and local VCS.

When coming to distributed VCS , we are having the main or master repository , and for which we will be having some extended repositories , which are called as sub branches or child branches , in general , we will be having a master repository and the code from the release branches will be pushed to the master repo , and develop branches push the code to the release branches and where as besides these develop branches , we will be also having feature branches , where we push these code into the develop branch.

We will be having more flexibility here , in developing and integrating the code with the project and parallel modification of the working code.

In distributed version control system , we will be having multiple repositories and similar to local VCS , we will have local copies of those repositories.

13:37

## 14. GIT 132. Introduction



In Distributed VCS , we will be having multiple copies of the repositories at every client , so , if we are having any server issue at any central level of the repository , doesn't affect the development process. As we can work on the code on local repository and we can commit to that code , and if there is no issues with the central repository , we can sync the local repository with the central repository.

14:37

## 14. GIT 132. Introduction



In a Git , mercurial etc.. , we don't checkout from a central repository , we clone the central repository into local and pull the changes from it. Our local repository is a complete copy of everything on the remote server. Here we can modify the local repository , such that we can check in / check out from the local repository , commit changes into local repository , local repository keeps the versioning history , and if we think our local repository is working fine , we can push those changes onto our server , so the changes would be global for that branch.

15:13

## 14. GIT 132. Introduction



Git is a version control system that allows developer to track changes to their code and collaborate with other developers on software projects. It is distributed version control system, which means that each developer has a local copy of the repository on their machine , and changes are made and committed locally before being shared with other developers.

### **Git is distributed version control system , but not Github**

Github is web based platform that provides hosting for Git Repositories , It is designed to help developers manage and share their code and collaborate on software projects.

In addition to providing Git repository hosting , Github also offers a range of tools and features that make it easier for developers to collaborate such as issue tracking , pull requests , and project management tools.

Git => Version control system

Github => Hosting service for Git Repositories

Github is used to manage and track changes to their code

3:32

**14. GIT** 133. Versioning



Git keeps track of files , not the directories , so , if we have any empty folder in our git repository ,the git doesn't recognize the directory which is empty into the repository.

4:04

**14. GIT** 133. Versioning



Some git commands in git bash

- 1) check if git is installed or not => git
- 2) check the git version => git --version
- 3) git init => initialize the directory into a repository , this command results to add a .git folder into the folder , so it records the status and updates and history of the files.
- 4) git status => git status is helpful to know the status of the files inside the repository , it shows all the file that are need to be committed which are called as untracked files and staged files which are the files which are ready to commit but not committed or it shows 'behind n versions' which tells us we need to push the changes into the repository.

4:29

**14. GIT** 133. Versioning



If currently our files are untracked , it means it is not staged yet , so we must stage them first , so we can push them to the repository. So , to stage the files under a directory , we use

```
git add directory_name/
```

or If we want to push any specific file , we use the

```
git add <file_name>
```

After using git add , it change the status to the staging stage. It means it is ready to be committed. Now we need to commit the files using the

```
git commit -m "message for the commit"
```

Prior to that , we must see the global user.name and global user.email for the git bash , which is used to update the repository while we commit , and helps in tracking the changes , if we are in the team.

```
git config --global user.email "jeevansaikanaparthi@gmail.com"
```

```
git config --global user.name "jsai2001"
```

And after setting this , we can commit our code , as usual , the above declaring of user.email and user.name is a onetime process. And later after commit , we can even check

```
git status
```

 => shows status of commit.

8:07

## 14. GIT 133. Versioning



Until now we done the operations of the local repository , Here when coming to remote repository , we can create one using Github , bitbucket , codecommit etc.

Previously , we have created our own local repository , but now , we can have a repository in remote either in Github or bitbucket and clone that into our local repository using

Clone Github repository to Local

```
git clone URL
```

Clone Local to Remote

\* cd to local repository

\* 

```
git remote add origin <ssh-remote-repository-name>.git
```

\* 

```
git push
```

 (which pushes the local changes to the remote repo)

\* 

```
git pull
```

 (pulls the latest changes from the repo)

12:30

## 14. GIT 133. Versioning



To connect the locally created repository , with the remote repository , we need to add the remote repository into the local repository config and authorize the changes to occur on remote by the local repository , so for that , we use the following commands.

Remember prior to run this command , we need to have some local repository , where we have git initialized and now go into the git local repository and use the following line.

```
git remote add origin <https-link-of-git-repository>
```

This made the local repository add this git repository link in it's config file , so making further changes and commit , results to reflect it on the remote repository in parallel.

13:13

## 14. GIT 133. Versioning



```
git status
```

This command helps us to show all files which are modified and not committed , even though it is staged for commit. This helps to know what files are being modified , from the latest commit (hopefully last working code).

13:21

## 14. GIT 133. Versioning



So to move this file from un-staged to staged , we use the following line.

```
git add <file-name>
```

And to move from staged changes stage to committing the changes (locally) , we do

```
git commit -m "commit-message"
```

This helps us to push the changes from local repo to the remote repo ,

```
git push origin <branch-name-to-push-into>
```

14:51

## 14. GIT 133. Versioning



So , if we want to see the git logs , so , we can see the latest changes and their commit id's , and using those commit id's we can see what changes has been done on those commits.

```
git log
```

Using "git log" , we get a long commit id's , but if we want commit id's of shorter length , we need to use

```
git log --oneline
```

Then we will get latest commit id's , and if we want to see changes done , by those commit id's use

```
git show <commit-id>
```

17:46

## 14. GIT 133. Versioning



Suppose , if a team is working on the same repository and we need to get their latest commits and updates , so we need to pull those changes from the remote repository into our local repository , so we need to do the following.

`git pull`

It pull's all the latest changes from the remote repository to the local.

7:24

## 14. GIT 134. Branches & More



We can create one or more sub repositories for a repository , this type of method is used in , having a master branch and many and more sub feature branches , so after making changes to the sub feature branches , we can push the changes into the master branch or the main branch , so this type of work can help us to work in teams productively and without work clash in agile manner.

For example , create a sub repository under main branch

1) Checkout to the main branch first

2) `git branch -c <sub-repo-branch-name>`

3) We can check the status of the branch creation using `git branch -a` , which helps us to see all the branches that are present on the local repository.

4) To checkout to that particular created branch use `git checkout <sub-repo-branch-name>`

5) To remove any file from the repository , it is better to use `git rm <file-names>` , which helps us to remove the file from repository and stage that change in our local repo , so only push the change into remote/local repo is remaining.

7:30

## 14. GIT 134. Branches & More



To know the status of the current local repository branch , we are in , we need to use the `git status`

Similarly to git rm , we are having `git mv <src-filename> <dst-filename>` , it moves the file and stage the change.

`git commit -m 'commit-message'` It is used to commit the changes with the message mentioned.

`git log --oneline` shows the addresses of the recent commits , so we can check the details of those commits using `git show <commit-id>`

As usual , `git add <filename>` , stage a change into the repository.

If we want to push the changes into a specific repository , when we are under main branch , we must do `git push origin <sub-branch-name>`

Similar to the `git checkout <branch-name>` , we have `git switch <branch-name>` command , which is used to checkout to a new branch.

If we want to merge the branch-1 with the branch-2 , then we must

Go into the branch-1 using `git checkout branch-1`

7:35

## 14. GIT 134. Branches & More



And we need to use `git merge branch-2` which is source branch and we merge the branch into branch-1

0:50

## 14. GIT 135. Rollback



Here for file rollback there are two cases , one is file rollback before staging and

For example , we have a file which was modified and it's name is jupyter1.rb , and if we want to rollback to the previous state which was committed , we need to do `git checkout jupyter1.rb` , that is noting but the filename , we are checking out to that filename , it means it rollbacks to the previous committed stage.

`git diff` from the repository that we are working shows all the file changes that we have done in that particular repository , without committing.

To check the difference of the file that was staged , we need to use `git diff --cached`

And if we want to restore or un-stage a file from a staged stage , we need to use

`git restore --staged <file-that-was-staged>`

4:00

## 14. GIT 135. Rollback



If we want to check the difference between two commits we get the current commit id and previous commit id , using `git log --oneline` , and use `git diff <prev-commit-id> <curr-commit-id>`

We can even take back our commit using `git revert HEAD` , or we can give the commit id that we want to revert `git revert <commit-id>` , it generally asks revert message for the revert commit. Even , if we revert the commit , our history of the repository doesn't change. It have all the deleted content displayed , even as deleted , but it will be displayed in prev commits.

So , if we even don't want that to be stored, `git reset --hard <commit-id>`

0:46

## 14. GIT 136. Git Ssh Login



when we try to login to the repository through the https , we need to give our username and the password with the URL , so , it is preferable to not expose them, while we are using those keys in our project. So , instead , we are having a method of ssh login's where we no need of any username and password to login through any repo through the git clone command.

In general , we can see the config of the git under

```
cat .git/config
```

How to activate ssh keys

Normally ssh keys are present under /.ssh/ , so if we planned to remove them , we need to use,

```
rm -rf .ssh/*
```

To generate ssh keys , we need to use

```
ssh-keygen.exe
```

Then we need to fill some details according to what is on the terminal , at this particular location

```
ls .ssh/ => id_rsa , id_rsa.pub
```

We have to copy the public key and paste it on the GitHub.

Settings>SSH and GPG keys>New SSH key> And paste the public key and give some title for it.

After adding this ssh key into repo , we can clone DIR from that repository without a password.

3:11

## 14. GIT 136. Git Ssh Login



Now we can even clone the private repository with git clone without a password.

```
git clone <ssh-link-for-repo>
```

You may ask fingerprint access , give yes , and then the repository will download without a password.

0:07

## 16. Continuous Integration with Jenkins 140. Introduction



What is Continuous Integration?

When developers are working on a product development , they will code , build code and test it and then push it to the centralized version control system. This is the continuous process. Similarly there will be so many developers will be developing parallelly and doing test on the local for their code and push it to the centralized control system. This process will be happened many times of the day by all the different developers. The codes or the code gets merged into the centralized repository on a regular basis. After

some time , when the code is been tested into the production , it will be generating a lot of bugs and issues and conflicts.

0:12

## 16. Continuous Integration with Jenkins 140. Introduction



This occurs even if we test our code while we are pushing the code into the main branch , because , our code is being merged but not integrated , we don't see how our code will affect other's or our own application , we need to run all the applications when any change in the source code , then this process is called continuous integration .

Basically Continuous integration , it integrates codes of multiple commits from different developers , and check whether the code and the application executes in proper way either in module level or application level , and if there is any error , the developer will be mentioned that , his part of the code in the module is incompatible with the application , doing this we can know if there are any integrating issues between the code.

So , if there is no such integration , developer needs to focus on the issues that were generated , instead of developing new code. Then Integration is a painful process as errors can also be related to other modules.

0:14

## 16. Continuous Integration with Jenkins 140. Introduction



This process of continuously checking the code workability for every commit is called continuous integration , so this must be automated , if it needs to be done for every commit. This process will be called a continuous integration process and Jenkins is the most famous tool for continuous integration tool. It is started as continuous integration tool , but now it is much more than that.

Whenever there is a code change by the developer , Jenkins is going to fetch the code , build and evaluate it and notify to the developer. This is a continuous integration process done by Jenkins. Jenkins is an open source. Now it is more than a tool as CI server . Now Jenkins has a plug-in for version control systems to integrate Jenkins with almost every version control system.

It is also having a build plugin , where we can build a Java , dotnet , NodeJS and any kind of source code by their build tools.

0:19

## 16. Continuous Integration with Jenkins 140. Introduction



It is having a cloud plugin , testing plug-in and many more plugins in Jenkins. So , now we can use Jenkins as a continuous integration server or as a continuous delivery tool or tool just to run your scripts or tool to run your cloud automation or tool to integrate with your other devops tools or developer tools or testers tools.

We can install Jenkins pretty easier , we just need Java JRE or JDK.

0:00

## 16. Continuous Integration with Jenkins 141. Installation



In this tutorial , we are going to launch an

2:30

## 16. Continuous Integration with Jenkins 141. Installation



```
#!/bin/bash
sudo apt update
sudo apt install openjdk-11-jdk -y
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo tee \
/usr/share/keyrings/jenkins-keyring.asc > /dev/null
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
/etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update
sudo apt-get install jenkins -y
```

This is used to install Jenkins through Linux terminal into the system.

2:33

## 16. Continuous Integration with Jenkins 141. Installation



We can set Jenkins on Docker , Kubernetes , Linux

For Fedora OS , we need really minimum 1 GB RAM and some disk space , so it is easy to install. And it also takes minimal space comparatively with other OS's.

Jenkins installation needs some dependencies that need to be installed in prior. We need JDK of version 11 and maven which helps us to run our builds.

3:06

## 16. Continuous Integration with Jenkins 141. Installation



As we need to have Jenkins installed on the cloud server , we need to launch an instance. Name the instance as per convenience , for example 'Jenkins Server' . And

select AMI as Ubuntu , with the AMI as you wished , here , we choose Ubuntu Server 20.04 LTS which comes under free tier eligible.

The Instance type is chosen to t2.small , t2.micro is sufficient but it could be slow , these instance types denotes the R/W operation speeds that given to us and the Storage allocated to us by the AWS as per we requested.

And we need to create a key pair which is used to login to the ubuntu terminal of the cloud from the local through ssh. Name the key pair , RSA key pair type is used to login to the server password less with help of private and public key pair. And the format of private key file should be .pem file so , we can login through the OpenSSH and .ppk is used for logging in through PuTTY.

And create a key value pair , download and save it to the local to connect to the terminal.

3:58

## 16. Continuous Integration with Jenkins 141. Installation



Now we need to create a Security Group , so that , we can control what can in and out in our terminal and the instance , in security group column , select create security group and there will be an Inbound security group rules where add some rules.

One of the rule is ssh rule

Type: ssh

Protocol: TCP

Port range: 22

Source type: My IP

Which helps to login to the instance with ssh through My IP password less manner

Second Rule:

Type: Custom TCP

Protocol: TCP

Port range: 8080

Source type: My IP

Which helps to connect to the instance through the browser , and the Public IP addresses changes every time when you power down the instance , it is preferable to shutdown the instance everyday after the work.

We can even configure the storage of our instance under "Configure storage".

And in Advanced details> User data , we can have our scripts that our terminal need to instantiate the time when it is creating , we can install all the software's , that are required here , download using yum or the apt-get.

5:16

## 16. Continuous Integration with Jenkins 141. Installation



After launching the instance , we will have a page which have details , security , storage , networking and more details , where we can have our public IPv4 address , through which we can connect through the browser and interact with Jenkins GUI. So try to open the page (Public IPv4 address of the instance : Port number) , to interact with Jenkins.

5:35

## 16. Continuous Integration with Jenkins 141. Installation



We get a page , similar to this to Unlock Jenkins on the browser , there will be path mentioned for the password.

**/var/lib/Jenkins/secrets/initialAdminPassword** => Login to the instance using ssh through the terminal , and get the password at this location.

**ssh -i Downloads/jenkins-key.pem ubuntu@3. 89. 49.9** => This line is used to login to the ubuntu terminal through ssh in the local terminal. Here 3.89.49.9 is the MyIP of the instance.

If there is any software that we couldn't see as installed that we mentioned in shell script , we can check the code which we entered in user data through the following command.

**curl http://169. 254. 169. 254/latest/user-data** , we can see , if we can find any error in it.

We can check the status of the Jenkins using "systemctl status jenkins". And then if we see , the status of the Jenkins is not running or Inactive , then we need to check the user-data for the error in the bash script that we written.

7:06

## 16. Continuous Integration with Jenkins 141. Installation



All the files of Jenkins are present under /var/lib/jenkins/ and the password is present under ( /var/lib/jenkins/secrets/initialAdminPassword ). We need to copy past it in the browser , so we can access the Jenkins GUI. The power of Jenkins lies in the plugins that it have.

Some of the plugins are:

- 1) Organization and Administration
- 2) Build Features
- 3) Build Tools
- 4) Build Analysis and Reporting
- 5) Pipelines and Continuous delivery
- 6) Source code management
- 7) Distributed Builds

- 8) User management and Security
- 9) Notifications and Publishing
- 10) Languages

9:49

## 16. Continuous Integration with Jenkins 141. Installation



Save the Jenkins URL , so we can access Jenkins as the admin and access the UI. The IP for the instance is dynamic , it means the IP changes , whenever we shutdown the instance and restart it , but it doesn't affect Jenkins URL , the URL will be the old and first IP address where we created the Jenkins.

After configuration of Jenkins is done , it says , "Welcome to Jenkins"

0:24

## 16. Continuous Integration with Jenkins 142. Freestyle Vs Pipeline As A Code



We have two types of Jobs in Jenkins, Freestyle and Pipeline as a code. Jobs basically means workload , which Jenkins runs for us.

2:27

## 16. Continuous Integration with Jenkins 142. Freestyle Vs Pipeline As A Code



In Bash script , we write a script to accomplish or automate a task , in Jenkins , we create jobs. As said there are two jobs , freestyle and pipeline as a code.

Basically freestyle jobs are all graphical jobs , so we need to click on create a job and fill all the information , it is basically a form filling , we run the job and see its output. It is fine , for small projects , but the problem is we will have many types and many jobs and those jobs needs to be connected together to create a pipeline , we need to fetch the code , build the code on different jobs and then run and test another job and deploy on another job and then we connect all of them together manually one by one. But this also can be achieved , the problem occurs when we want to replicate an already created pipeline , we need to do everything from scratch with the help of GUI manually , so this is the issue when you want a same pipeline in another project.

Freestylejobs are just for the learn and understanding purpose

2:28

## 16. Continuous Integration with Jenkins 142. Freestyle Vs Pipeline As A Code



Free style jobs are not recommended in real time now , we need everything as a code in devops, even the pipeline should be created through a code and Jenkins allows that , this is called pipeline as a code. The pipelines are written in Groovy language. This is

similar to creating a job , in job we mention the entire script and that script is our pipeline. So this code can be version control because we can store and retrieve it as a text instead GUI. So now the whole pipeline can be represented as a code.

0:09

## 16. Continuous Integration with Jenkins 143. Installing tools in Jenkins



Jenkins Dashboard looks like this page.

1:53

## 16. Continuous Integration with Jenkins 143. Installing tools in Jenkins



For Jenkins execution , we need both JDK and the Maven configured on the Jenkins. So , on the Jenkins Dashboard , we can go to the Global Tool Configuration and where we can see the JDK , where we need to name the JDK as required and we need to give the path of the JDK , under JAVA\_HOME , this must be the path inside the server , it is not the one on our local machine , so we have to login to the Jenkins instance , and install the JDK as follows. And fill the path of the JDK inside the JAVA\_HOME.

Jenkins Dashboard > Manage Jenkins > Global Tool Configuration > Change the settings as required.

2:20

## 16. Continuous Integration with Jenkins 143. Installing tools in Jenkins



The command we used to login to the Jenkins server through SSH is

```
ssh -i Downloads/jenkins.pem ubuntu@3.22.194.47
```

Then , we will login to the Jenkins server and then use the following commands to install jdk.

```
1 | sudo apt update  
2 | sudo apt install openjdk-8-jdk -y
```

And the jdk will be downloaded under `/usr/lib/jvm by default` , it will be named as `java-1.8.0-openjdk-amd64` => `/usr/lib/jvm/java-1.8.0-openjdk-amd64` , paste this path in JAVA\_HOME. similarly , we must declare the name and version of maven while we configure the Jenkins.

These tools that we declared here will be installed when we trigger the build.

0:05

## 16. Continuous Integration with Jenkins 144. First Job



On Jenkins Dashboard , we can see the "Create a Job" and in next stage we can name our build and we can see the ways to create a job here , for instance take free style

mode. Then we will see the page where we can edit the configuration of the Job. Then we will get to open the Configuration page. In which we have ( General , Source code management , Build Triggers , Build Environment , Build Steps , Post-build actions)

In Source Code management we need to select the git and then as we need to pull the code from the repository then select git and get the http URL of the repository to the Repository URL of Source Code management , if it is a public repo , we no need to give any credentials and if we have the private repo , we must declare one . And even we need to declare the branch , that we need to run from the repository. Here in Build Steps , we can declare the Build step either Windows terminal , or linux terminal or any other , by selecting them , we can write the script to exec

3:34

## 16. Continuous Integration with Jenkins 144. First Job



This build step , works as the entrypoint for the Jenkins Job. This build script executes first before the Job.

4:27

## 16. Continuous Integration with Jenkins 144. First Job



Here basically , in build steps , we will be having multiple options in build steps , so in general , there will be some plugins to install in the project , so we can install those plugins and use them directly instead of executing them as general commands , for example , we can install maven plugin , to run maven commands , instead executing them as normal commands , this gives us lot's of customization in execution.

4:45

## 16. Continuous Integration with Jenkins 144. First Job



Here in general , we have the Job which generally pulls the source code and run the build steps , this is normal execution process of any Job in Jenkins.

5:22

## 16. Continuous Integration with Jenkins 144. First Job



Here , we can see the Jenkins Dashboard , where we need to go under Build section , here , we assume that , you have already a Job and you are in that Job Build section , where you can see the following options. ( Status , Changes , Workspace , Build Now , Configure , Delete Project ) , here we can build our project and configure the settings of our project and , when we do the build now , we get the Job running under. When we click that , we get the console output , the data is stored in the workspace which is present under Job Build section. We can build any Job multiple times , the directory

structure and the data of the dependent repository is stored under the workspace. And files that we download through the Build steps also be stored here.

8:50

## 16. Continuous Integration with Jenkins 144. First Job



And suppose , if we want to declutter the workspace , we can archive the necessary data , for example we can declare to archive some files , which have some certain pattern , so then after the build, all the files that have the certain pattern will be archived. And then we can clear the clutter from the workspace.

1:06

## 16. Continuous Integration with Jenkins 146. Plugins, Versioning & more



Workspace is the place , we will get , all files downloaded due to the Job Build , and so for each and every build the files remain constant unless there is an updated version but the binaries changes , this Jenkins workspace acts as our local workspace , it is a place we have our file workspace and the binaries.

Workspace is present under Dashboard > Build > Workspace

As said , for each and every build the binary gets updated and if we want to get the last executed binary than the latest due to any reason , we could not get it in general way , so we need to maintain the versioning for the binary.

Under the Dashboard > Build , we are having the following options:

- \* Back to Dashboard
- \* Status
- \* Changes
- \* Workspace
- \* Build now
- \* Configure
- \* Delete Project
- \* Rename

1:25

## 16. Continuous Integration with Jenkins 146. Plugins, Versioning & more



Now we are trying to create a new Job , named as Versioning-Builds , which is of type freestyle project and we are copying the configuration of the Job from the previous existing Job called 'Build' Job. Then all the configuration of Versioning-Builds will be copied from the 'Build' Job.

Columns of Configuration:

General , Source Code Management , Build Triggers , Build Environment , Build , Post-build Actions.

Now in the Build Stage , we can have another stage of build where we can execute the shell commands there , you can find the option in 'Add build Step' under the Build section. We add the following:

```
1 | mkdir -p versions  
2 | cp target/vprofile-v2.war versions/vprofile-V$BUILD_ID.war
```

So , for each and every build there will be a new artifact generated and the artifact will be stored under the versions with specific build number.

4:33

## 16. Continuous Integration with Jenkins

146. Plugins, Versioning & more



Jenkins variable list:

<https://www.perforce.com/manuals/jenkins/Content/P4Jenkins/variable-expansion.html>

4:54

## 16. Continuous Integration with Jenkins

146. Plugins, Versioning & more



So, now when you run multiple builds , you will get to create multiple artifactory and stored in versions folder tagged with the build number as seen earlier.

5:47

## 16. Continuous Integration with Jenkins

146. Plugins, Versioning & more



Here , we can also run a build with parameters predeclared , we can declare parameters under general , and we can see any option similar to 'This project is parameterized' , to add parameters and where you can see multiple ways to declare name of one parameter. And after declaring , we can use this particular variable in our builds or further steps. Sometimes we can define version numbers here , which we need for the building based on the version branch. And after defining the name of the parameter , as we needed value for it to the build , on the Job Dashboard , we can see an option 'Build with parameters' , here we can give the values for the parameters and run the build.

8:22

## 16. Continuous Integration with Jenkins

146. Plugins, Versioning & more



Try to avoid the 'Build with Parameters' because , it slows the CI/CD process because it is going to slow down the process as it needs to wait until it gets value for the parameters.

8:30

**16. Continuous Integration with Jenkins** 146. Plugins, Versioning & more

So, there is a better way to do versioning , that is through plugins , so go to the Dashboard > Manage Jenkins > Manage Plugins

Manage Plugins is the main power house of Jenkins , the main power of Jenkins is it's plugins. We can install and uninstall the plugins through this manage plugins.

We are having a 'Zentimestamp' plugin to download , so we can use it's derived variable 'BUILD\_TIMESTAMP' to be used while we copying the artifactory into separate directory.

**This is the other way of versioning the artifactory**

```
cp target/vprofile-v2.war versions/vprofile-V$BUILD_TIMESTAMP-$BUILD_ID.war
```

So, we can use similarly , with such predefined variables , those variables may be came by default or through downloaded plugins.

16:56

**16. Continuous Integration with Jenkins** 146. Plugins, Versioning & more

But even after all these , we are storing our artifacts on our workspace , which is Jenkins server , for us the Jenkins is not the storage space , it is CI/CD server , so we are going to shift these files somewhere automatically during the build. We will see that in later comments.

3:59

**16. Continuous Integration with Jenkins** 147. Flow of Continuous Integration Pipeline

This is the pipeline , and in coming videos there will be a process for executing through all the stages.

0:10

**16. Continuous Integration with Jenkins** 147. Flow of Continuous Integration Pipeline

This is the pipeline , and in coming videos there will be a process for executing through all the stages. We will use git , maven sonarqube, nexus sonatype tools used here.

0:15

**16. Continuous Integration with Jenkins** 147. Flow of Continuous Integration Pipeline

**The flow:**

- \* Developer's job is to write the code and developer will write the code and make the changes to the code and test it locally and if they are good with the changes , they will

push it to a centralized repository like GitHub. Developers will have the git tool which will integrate with GitHub repository and the code will be committed to GitHub repository.

\* Jenkins will detect the change in the repository and fetch the new code by using git tool. Jenkins will have the git tool and git plugin. which will help accomplish this task to fetch the code.

\* After pulling the changes to the Jenkins workspace , the code will build. we will be using MAVEN to build the code because we have java code and our code can be built with maven tool , the source code can vary , so does the build tools , once the build is done , it will generate artifacts.

0:20

## 16. Continuous Integration with Jenkins 147. Flow of Continuous Integration Pipeline



### The flow:

\* Next we will conduct unit test again by using the maven, maven will have some unit testing framework that developer will use. unit testing will be part of our source code , so a devops no need to do much here , we just need to execute some steps that will run this test and generate reports mostly in xml format

\* Once the reports are ready , we will conduct another kind of test called 'Code Analysis'.

\* The unit test checks whether the unit of the code works or not.

\* Code analysis checks if the code has any vulnerability , are we following the best practices or not or is there any bug in the code , there will be many parameters on which code analysis will judge our code. We will use 'sonar cube scanner' to scan the code.

0:25

## 16. Continuous Integration with Jenkins 147. Flow of Continuous Integration Pipeline



### The flow:

\* There are many code analysis tools available in the market , here we use the sonar cube scanner and scan the code and this will generate reports in xml format , these reports will be uploaded to SonarQube server.

\* In SonarQube , we will have a proper graph , charts and we can see what are the bugs , vulnerabilities and many other things in our code , we can also set a quality gate and we can say , if the code doesn't follow these practices then fail and in turn the pipeline will stop.

\* And if it passes the code analysis gate , we will have reports in the SonarQube and some verified artifacts.

\* We Built the Code => Test the code => Analyze the code > Get Verified artifact => Now we can deploy the artifact on the servers

\* These verified artifacts will be versioned before deployment and store in the Nexus Sona type repository.

0:30

## 16. Continuous Integration with Jenkins 147. Flow of Continuous Integration Pipeline



- \* All these steps until now happened in Jenkins ( CI tool ) , we can have any other CI tool such as GitLab circle , Bamboo or any CI tool but the process would be the same.
- \* And whatever CI tool we are using , we must need to integrate with other tools like GitHub , SonarQube , Nexus or any other tool.

0:32

## 16. Continuous Integration with Jenkins 148. Steps for Continuous Integration Pipeline



To create the Continuous Integration pipeline , we setup the Jenkins , Nexus and SonarQube through the user setup code by EC2 instance. And later we will set the security group and give permissions to interact between them. And add all the necessary plugins ( Git plugin , Nexus plugin , SonarQube plugin , maven plugin etc ) and then we integrate nexus with Jenkins and sonarqube with Jenkins , integration of sonarqube with Jenkins need some script , while Nexus integration would be easy. Once the entire setup is done , we enter the pipeline script and set the notification , it means if any thing goes wrong in pipeline we get the notification.

0:19

## 16. Continuous Integration with Jenkins 149. Jenkins, Nexus & Sonarqube Setup



We set up the Nexus and Sonarqube server setup through the user scripts that will be setup through the EC2 instances.

[github . com/devopshydcclub/vprofile-project/tree/ci-jenkins/userdata](https://github.com/devopshydcclub/vprofile-project/tree/ci-jenkins/userdata) this is the link where we have the scripts to setup the jenkins , nexus and sonarqube. It will be done in EC2 instance's ubuntu18 OS for sonar , Jenkins and centOS for nexus 7

3:42

## 16. Continuous Integration with Jenkins 149. Jenkins, Nexus & Sonarqube Setup



Check the Jenkins , Nexus , Sonarqube setup on EC2 instance in this video.

Some of the security group rules mentioned as follows:

ssh => My IP => Port 22

Custom TCP => Anywhere => Port 8080

Custom TCP => Anywhere => Port 80

SonarQube checks at port 80 at Quality Gate checks so , to keep it simple and less confusing , we are assigning Source Type to Anywhere instead of My IP.

And under user data , we can paste the Jenkins installation script , and we can launch instance.

For Nexus , you need a Centos7 AMI ,  
Instance type for Jenkins will be t2.small and Nexus will be t2.medium

Security Group for Nexus as follows:

ssh => My IP => Port 22

Custom TCP => Anywhere => Port 8081

Jenkins access the nexus through port 8081

Add the user data script under the advanced section and launch the instance.

Now we will create EC2 instance for SonarQube

OS: Ubuntu18

InstanceType: t2.medium

Security Group will be as follows:

ssh => My IP => Port 22

Custom TCP => Anywhere => Port 80

Custom TCP => Anywhere => Port 9000

Both Port 80 and 9000 access the same sonarQube service.

In general SonarQube runs on the port 9000 , but Nexus transfers the requests to SonarQube to port 80. We can directly use port 9000 , then there is another service called nginx which will be called during the sonar setup , for forwarding the requests which are received at port 80 to 9000 , so anyways is fine.

Plugins needed to be installed:

Nexus: To integrate Nexus

SonarQube: To integrate SonarQube

Git

Pipeline Maven Integration Plugin

BuildTimestamp (For Getting Some predefined variables in Jenkins)

Pipeline Utility Steps

How to install a plugin:

Jenkins Dashboard > Manage Jenkins > Manage Plugins > Available > Search & Download what plugins you need

Pipeline as a Code:

- \* We can Automate pipeline setup with Jenkinsfile
- \* Jenkinsfile defines stages in CI/CD Pipeline
- \* Jenkinsfile is a text file with Pipeline DSL Syntax
- \* The code is similar to groovy language.
- \* We have two ways to declare the pipeline , one is scripted and another one is Declarative.

We are going to use declarative process to setup the pipeline

**Pipeline concepts:**

**Pipeline:** It is the main block and everything inside pipeline will be executed by Jenkins.

**Node or agent:** There are settings where you can define where pipeline can get executed on which node or on which agent.

**Stages:** These are the place where actual execution happened , and we have many steps , it could be the commands like maven install or git pull or upload artifact to Nexus or any such step that we want to execute the pipeline.

The code looks similar to this structure , pipeline is the main block and under it we define any agents and multiple stages and in each and every stage we will have multiple steps to execute.

Build , Test , Deploy are some of the following stages.

Pipeline is the main block of the code.

```
1 | pipeline{  
2 |     agent{  
3 |         }  
4 |     tools{  
5 |         }  
6 |     environment{  
7 |         }  
8 |     stages{  
9 |         }  
10| }
```

In pipeline we are having things like agent , where we can define where the job is going to run , and define tools from your global tool configuration. If we have some tools included , then we can mention tools like maven or jdk.

Environment will be the environment variables and stages will be the steps that we are going to execute in our Job.

2:27

## 16. Continuous Integration with Jenkins

151. Pipeline As A Code Introduction



```
1 | pipeline {  
2 |     agent {  
3 |         label "master"  
4 |     }  
5 |     tools {  
6 |         maven "Maven"  
7 |     }  
8 | }
```

In agent , we will have the node to run details , we can mention the pipeline to run on either master or slave or any other , and in tools we define the tools that we declared in global tool configuration.

```
1 | pipeline {  
2 |     environment {  
3 |         NEXUS_VERSION= "nexus3"  
4 |         NEXUS_PROTOCOL = "http"  
5 |         NEXUS_URL = "you-ip-addr-here:8081"  
6 |         NEXUS_REPOSITORY = "maven-nexus-repo"  
7 |         NEXUS_CREDENTIAL_ID = "nexus-user-credentials"  
8 |         ARTVERSION = "${env.BUILD_ID}"  
9 |     }  
10| }
```

Other things in pipeline are environmental variables , so setting a variable of value , so it can be used in our steps and in our stages. For example , you can see we are using a nexus variable declared as env variable , which is going to use in our pipeline.

3:08

## 16. Continuous Integration with Jenkins

151. Pipeline As A Code Introduction



```
1 | pipeline {
2 |     stages {
3 |         stage("Clone code from VCS") {
4 |             }
5 |         stage("Maven Build") {
6 |             }
7 |         stage("Publish to Nexus Repository Manager") {
8 |             }
9 |         }
10|     }
```

Stages are the place where main execution goes , we will have stages , in which we can have multiple stages , like one stage that clones the source code and another stage that is going to run Maven build and another stage which can publish the artifact to Nexus and similar to that , we can have stages to deploy artifact to do code analysis. We can have n number of stages. And in these stages , we can have a stage that would like to send an email notification or if we want to achieve the artifact.

3:52

## 16. Continuous Integration with Jenkins 151. Pipeline As A Code Introduction



```
1 | pipeline {
2 |     stage('BuildAndTest') {
3 |         steps {
4 |             sh 'mvn install'
5 |         }
6 |         post {
7 |             success {
8 |                 echo 'Now Archiving...'
9 |                 archiveArtifacts artifacts: '**/target/*.war'
10|             }
11|         }
12|     }
13| }
```

We can declare a stage similar to this , we can have multiple stages and inside of it we can have multiple steps declared inside the steps , this is the place where we put actual command and also we can put some post installation steps like if we want to send an email notifications or if we want to archive any files. Here we can even declare the name of the stages, then inside we can write commands like maven install.

4:03

## 16. Continuous Integration with Jenkins 151. Pipeline As A Code Introduction



In Devops everything is an automation , and even the pipeline is automated with the Jenkinsfile.

While you are declaring anything in the tools section of the Jenkins pipeline , we need to ensure , whatever we mention here should be same as the tools mentioned in Global tool configuration.

8:32

## 16. Continuous Integration with Jenkins 151. Pipeline As A Code Introduction



```
1 | pipeline {
2 |     agent any
3 |     tools {
4 |         maven "MAVEN3"
5 |         jdk "Oracle JDK8"
6 |     }
7 |     stages {
8 |         stage('Fetch code') {
9 |             steps {
10 |                 git branch: 'vp-rem', url:
11 | 'https://github.com/devopshydclub/vprofile-repo.git'
12 |             }
13 |         }
14 |         stage('Build') {
15 |             steps{
16 |                 sh 'mvn install -DskipTests'
17 |             }
18 |             post {
19 |                 success {
20 |                     echo 'Now Archiving it...'
21 |                     archiveArtifacts artifacts: '**/target/*.war'
22 |                 }
23 |             }
24 |         }
25 |         stage('UNIT TEST') {
26 |             steps{
27 |                 sh 'mvn test'
28 |             }
29 |         }
30 |         stage('checkstyle Analysis'){
31 |             steps {
32 |                 sh 'mvn checkstyle:checkstyle'
33 |             }
34 |         }
35 |     }
36 | }
```

10:55

## 16. Continuous Integration with Jenkins 151. Pipeline As A Code Introduction



The following is the sample pipeline code , where we are having following sections

1) agent

## 2) tools

\* We must declare the tools that we are using in our pipeline , and these tools must be declared in Global tool configuration , ensure that these both matches , because as said , it must be declared in pipeline to use it in our application. And even if it is declared first those softwares should be installed and configured in the project.

## 3) stages

\* This is the place where we declare the actual steps to happen on our pipeline , for example , fetching , building the code and performing the unit tests are some of the stages that could be declared on the pipeline.

\* And even in stages , we can see a tag called 'post' it means , the pipeline executes this block after that stage successfully executed , for instance you can see a 'success' sub block under the 'post' block , if it got succeeded , it executes success block or else the other.

12:43

## 16. Continuous Integration with Jenkins

151. Pipeline As A Code Introduction



And here , we can see that , we are creating a pipeline , using the code , we can do that through

Jenkins Homepage > New Item > Pipeline

Go to the pipeline stage of the page , where we have two options

1) Pipeline script

2) Pipeline script through SCM

In Pipeline script , we will have part of the code , where we can upload the pipeline script that we have developed and save the pipeline

And in pipeline script through SCM , we will have option to add repository URL and credentials to upload and the branch to execute and in this branch , we must have the Jenkinsfile on the branch home page to execute and create a pipeline through it's code (In default).

Once check this part of the video.

In this video they didn't customize the path of Jenkinsfile to refer by the pipeline.

The above process which is been discussed is the declarative way of creating Jenkins Pipeline.

13:39

## 16. Continuous Integration with Jenkins

151. Pipeline As A Code Introduction



And after creating the pipeline using declarative way , we are having some options as mentioned here.

1) Back to Dashboard

- 2) Status
- 3) Changes
- 4) Build Now
- 5) Configure
- 6) Delete pipeline
- 7) Full stage view
- 8) Rename
- 9) Pipeline Syntax

We are following will have build history to track the latest or recent or earlier builds.

After creating pipeline , if we want to run the pipeline , we need to enter that pipeline first , for instance here the pipeline is named as 'sample-paac'. And next we need to click the option of Build Now. Then the particular build will be started and execute and following stages which are declared inside the Jenkinsfile will visible here

13:52

## 16. Continuous Integration with Jenkins 151. Pipeline As A Code Introduction



And for every build id that we have got , there will be some dedicated workspace for pipeline created which will be presented under the

Select the build id from build history > Workspaces > traverse the path it provides > We get the workspace of that pipeline.

14:55

## 16. Continuous Integration with Jenkins 151. Pipeline As A Code Introduction



KT Link for Jenkins pipeline : <https://www.jenkins.io/doc/book/pipeline/>

0:25

## 16. Continuous Integration with Jenkins 152. Code Analysis



The main moto of the processes is build a pipeline which does

- 1) Fetch code
- 2) Build code
- 3) Unit Test Code
- 4) Code analysis
- 5) Upload Artifact

We done the stage 1,2,3 through the pipeline earlier.

Now we need to work on code analysis. This is a test , which does on the code to detect vulnerability and functional errors. This is not about software testing this is the code testing.

Code analysis checks our code against best practices in that programming languages. It also able to get vulnerability out of the code , for example a simple memory leak can be a vulnerability. These are vulnerable for hackers to hack our software. And it also prevents us to 'detect functional errors before deployment' , these functional errors , doesn't create an issue with the software , but it can provide a weak spot in the software , so these issues are spotted in code analysis. To increase the code quality , we do code analysis.

2:24

## 16. Continuous Integration with Jenkins 152. Code Analysis



There are many tools to perform the Code analysis , they are

- 1) Checkstyle
- 2) Cobertura
- 3) mstest
- 4) owasp
- 5) SonarQube Scanner

For now we use Checkstyle and sonarqube

Before writing pipeline as a code , and integrate Code analysis in pipeline , we need to integrate sonarqube with Jenkins. And install sonarqube scanner.

Process:

- \* Install sonarqube scanner tools
- \* Integrate it with Jenkins server
- \* write our code analysis stage of the code in Jenkinsfile

3:59

## 16. Continuous Integration with Jenkins 152. Code Analysis



Here , if we remember , we have created an instance for sonarqube in AWS through AMI through it's instance on a server and that server was publicly hosted on some ip address , and also we given some port connection from Jenkins to the sonarqube , so now on Jenkins Dashboard page , go to Global Tool Configuration , we can add another tool , we can see 'Add Sonarqube' or something similar to that , and there , we must add it through suggested options that are directed to us. And ensure that whatever software package name that you give in Global Tool Configuration must be remembered because , we are going to initialize it through the Jenkinsfile under tools section of pipeline.

In Global tool configuration , you would definitely see that there is 'Add Sonarqube' , if not , it means that we need to install Sonarqube plugin in Jenkins.

6:36

## 16. Continuous Integration with Jenkins 152. Code Analysis



And here in the configuration , we can declare the name of the sonarqube instance , and we also need to declare the web url to access sonarqube , so we are giving the url of SonarQube's private address , which we created through AWS instance for sonarqube and ensure to declare port number , or leave it if it is default. And later add the server authentication token here , for this , we need to go to the sonarqube > profile > settings > Tokens > Generate Token

Ensure to save the token , else it will be lost on refresh.

This process until now ensures the sonarqube is integrated with the Jenkins.

2:54

## 16. Continuous Integration with Jenkins 153. Code Analysis Demonstration



We are going to install a sonar plugin to use the sonar scanner on the code and do the scan and upload it to the sonarqube server.

2:32

## 16. Continuous Integration with Jenkins 153. Code Analysis Demonstration



How to scan the code through sonar scanner and push the reports to sonarqube through the pipeline?

- \* We install a sonar plugin for that
- \* Once read this webpage (<https://www.jenkins.io/doc/pipeline/steps/sonar/>) for knowing how to use the sonarqube and integrate it to the pipeline.
- \* We run sonar scanning tool by using maven

4:44

## 16. Continuous Integration with Jenkins 153. Code Analysis Demonstration



This stages are example of the pipeline script that we are using , but we are not using the same script

```
1 | stage("build & SonarQube analysis") {  
2 |     agent any  
3 |     steps {  
4 |         withSonarQubeEnv('My SonarQube Server') {  
5 |             sh 'mvn clean package sonar:sonar  
6 |         }  
7 |     }  
8 |     stage("Quality Gate") {  
9 |         steps {  
10 |             timeout(time: 1, unit: 'HOURS') {  
11 |                 waitForQualityGate abortPipeline: true  
12 |             }  
13 |         }  
14 |     }
```

Here we are adding another stage , that is sonarqube analysis , and where we are installing the package and starting the sonarqube through the maven, and later we are adding another stage called quality gate , which have some metrics measure that need to be done before pushing it into the sonarqube.

9:58

## 16. Continuous Integration with Jenkins 153. Code Analysis Demonstration



```
1 | stage('Sonar Analysis') {
2 |     environment {
3 |         scannerHome = tool 'sonar4.7'
4 |     }
5 |     steps {
6 |         withSonarQubeEnv('sonar') {
7 |             sh '''${scannerHome}/bin/sonar-scanner -Dsonar.projectKey=vprofile \
8 |                 -Dsonar.projectName=vprofile \
9 |                 -Dsonar.projectVersion=1.0 \
10 |                 -Dsonar.sources=src/ \
11 |                 -Dsonar.java.binaries=target/test-
12 |                     classes/com/visualpathit/account/controllerTest/ \
13 |                     -Dsonar.junit.reportsPath=target/surefire-reports/ \
14 |                     -Dsonar.jacoco.reportsPath=target/jacoco.exec \
15 |                     -Dsonar.java.checkstyle.reportPaths=target/checkstyle-
16 |                         result.xml'''
17 |         }
18 |     }
19 | }
```

This is also part of the pipeline script , where we integrate the sonar scanner to scan our code and upload the reports generated to the sonarqube. Here , we are declaring the environmental , because the version of the software or upgraded software that we use should not make an issue in future.

12:02

## 16. Continuous Integration with Jenkins 153. Code Analysis Demonstration



And if we run the pipeline code , we will now have five stages including this sonarqube stage , where we can see that sonarqube quality gate was passed , but we didn't set any quality gate , the sonarqube have it's own quality gate in default , we can see that under sonarqube > Quality Gates

9:07

## 16. Continuous Integration with Jenkins 153. Code Analysis Demonstration



In the stage of sonar analysis , we declare the environment variable for sonar by it's version and now we are going to open sonarqube by it's IP address using withSonarQubeEnv function. We are using a variable canned scannerHome which we declared earlier. This variable gives us the location of the sonarQube executable (It's home path) or the tool home path. Now in that tool we need to go to the subLocation

/bin/sonar-scanner and we need to pass some properties which will tell sonar scanner where is our source code and what it needs to do , what are the reports that it needs to upload to the SonarQube server and all the information that we need to pass.

The sonarQube have some properties

- Dsonar.projectKey => It is going to be the key for the project
- Dsonar.projectName => It is going to be the name of the project
- Dsonar.sources => We are basically telling sonarqube to scan that directory

8:29

## 16. Continuous Integration with Jenkins 153. Code Analysis Demonstration



-Dsonar.java.binaries => When we run the build , and maven install , we get this target directory inside that will have these Java binaries will store at this location.

-Dsonar.junit.reportsPath => The unit test that we are conducting will be stored at this particular location.

These variables are predefined in the sonarqube documentation.

-Dsonar.java.checkstyle.reportPaths => we will store our checkstyle reports (target/checkstyle-result.xml) in this target path.

This pipeline code must be entered in a new Job ( Pipeline as a code ) , when we run the Job and it got success , we can open it , and visit the workspace of the Job which executed the pipeline.

9:40

## 16. Continuous Integration with Jenkins 153. Code Analysis Demonstration



Here we are basically doing is SonarQube scans the code , uploads all the reports on SonarQube server. This can see through accessing the sonarqube through its IP address or if the sonarqube tool is added in Jenkins , we can access that through Jenkins it'self.

2:26

## 16. Continuous Integration with Jenkins 154. Quality Gates



We need to create a new Quality Gate on the sonarqube server and we must assign that particular quality gate as the main quality gate for the project. Jenkins is not going to search , which quality gate that it must use from the sonarqube , but the sonarqube provides that detail to Jenkins through webhooks.

2:37

## 16. Continuous Integration with Jenkins 154. Quality Gates



Here we are creating a new webhook , naming it as jenkins-ci-webhook and we need to provide the URL , for that , we need to declare the pattern like.

<Jenkins-url>/sonarqube-webhook

Jenkins-url is the public IP that we are using to access the Jenkins server and which is created as an instance in AWS.

We will be having port number in the url , here this port number should be accessed to sonarqube , we must create that security group while creating the instance or we must give permissions to access it by anyone , or else we can't integrate jenkins with the sonarqube.

5:11

## 16. Continuous Integration with Jenkins 154. Quality Gates



For now , we have linked the sonarqube scanner and Jenkins , so artifacts will be generated during the build stage of the code and the code will be scanned for quality checks at SonarQube scanner and if the code pass in the quality checks performed in SonarQube scanner , we need to pass those artifacts to nexus repository.

0:25

## 16. Continuous Integration with Jenkins 155. Software Repositories Intro (Nexus)



Now the main moto is to upload the artifact to Nexus OSS Sonatype repository , as the name says it is a place to store and retrieve softwares. Here we will see how to store the artifact to the repository. But also these repositories are used to download the dependencies , like when we run mvn install , it downloads dependencies from the internet. Here we can also point the maven to download dependency from our own repository like nexus repository that we are creating now. Else we can download it from maven repository directly through online.

Here there are different kinds of repositories like maven to store maven dependencies , apt - package manager for debian based systems , yum - packages for red hat based , nugget for dotnet , npm for JavaScript , Dockerhub to store and receive docker images.

For now we are using Nexus sonatype repository , because we are using java and that repository is using Java. So the repository that we are using depends on the language we used in source code.

2:02

## 16. Continuous Integration with Jenkins 155. Software Repositories Intro (Nexus)



### Nexus Software Repository Manager

- Runs on java
- Used to store artifacts
- Used as a Package manager for dependencies

- Opensource & Enterprise Versions
- Supports Variety of repo like maven,apt,docker, Ruby gems etc

2:35

## 16. Continuous Integration with Jenkins 155. Software Repositories Intro (Nexus)



### Jenkins Integration with Nexus

Developers develop the code and commit to the github repository , and from Github repo , the jenkins fetch the code and build that code which was fetched and upload the binaries or artifactory into Nexus repository after the application pass all the Quality Gates. We will have multiple versions of artifacts pushed into the repo. And the operations team can download the server and upload the needful artifact onto the server.

We also need to have nexus instance to be run on the AWS , to use that repository manager.

4:03

## 16. Continuous Integration with Jenkins 155. Software Repositories Intro (Nexus)



On the sonatype nexus repository manager , we will be having repositories column where we can create a repository to store our artifacts in it , so we need to create a repository in here , we are naming it as vprofile-repo , and also we need to get the credentials for the repository to use that software repository in our pipeline.

To use nexus repository in our Jenkins pipeline , we need to declare credentials here.

Process:

Dashboard > Credentials > System > Global Credentials

Here we can add our credentials and tokens , so we can access any server through Jenkins pipeline , without any restrictions.

In general , in here , we added two credentials here in Global Credentials , they are MySonarToken and nexuslogin

5:36

## 16. Continuous Integration with Jenkins 156. Nexus PAAC Demo



```
1 | stage("UploadArtifact"){
2 |     steps{
3 |         nexusArtifactUploader(
4 |             nexusVersion: 'nexus3',
5 |             protocol: 'http',
6 |             nexusUrl:'172.31.18.28:8081',
7 |             groupId:'QA',
8 |             version: "${env.BUILD_ID}-${env.BUILD_TIMESTAMP}",
```

```
9     repository: 'vprofile-repo',
10    credentialsId:'nexuslogin',
11    artifacts: [
12      [artifactId:'vproapp',
13       classifier: '',
14       file: 'target/vprofile-v2.war',
15       type: 'war']
16    ]
17  )
18 }
19 }
```

This is also one of the stage , in the pipeline , this stage sends the artifact to the nexus sonar repository.

6:54

## 16. Continuous Integration with Jenkins 156. Nexus PAAC Demo



We need to use private IP's for all three instances that we are using , or else , everytime we restart the instance our public IP's get changed , so we must configure the Jenkins and Sonarqube and nexus such that it takes private IP rather than public one. If we take public IP , we need to change the IP addresses everytime we relaunch the instance.

In Jenkins => Dashboard > Configure

And even in sonarqube , we need to update the webhook which connects sonarqube to the Jenkins , change the public URL to the private URL in webhook of Sonar webhook.

Public IP changes whenever we relaunch the instance.

And Private IP is the static one.

9:25

## 16. Continuous Integration with Jenkins 156. Nexus PAAC Demo



The artifacts which are generated and saved in the sonatype nexus repository will be tested by the operations team for further testing or it can be directly deployed to the server , but this stage is not part of the Jenkins pipeline.

0:14

## 16. Continuous Integration with Jenkins 157. Notification, Slack



Until now , we automated all the stages using Jenkins , but there are two stages that we didn't automate , one is if we want to build the project we must come here and build it manually using build now or build with parameters option and if we want to know the result of the build we need to come back to the Jenkins and see the build details , these are the two that we want to change.

- 1) Autobuilding the code
- 2) Emailing the result of the Job to the user.

1:31

## 16. Continuous Integration with Jenkins 157. Notification, Slack



Jenkins supports vast plugins , and we use one those plugins to send the notification for the user after the result of the build. We use slack notification plugin of Jenkins for sending the email notifications. Slack is the main collaboration tool that most of the people uses now.

2:48

## 16. Continuous Integration with Jenkins 157. Notification, Slack



We are doing the setup for the slack here, creating a profile , signing in and provide the email addresses of teammates to add into the slack. We can add our channel for our group and we can add members in it. And to Slack getting authenticated by the Jenkins , we need a token from the slack , so we can login passwordless. For that , search for "Add Apps for Slack" > And search "JenkinsCI" , the "Add to Slack" And then There will be option to Post to channel , and we need to give the channel that Jenkins need to push notifications to , and by selecting it , click "Add Jenkins CI Integration" , And we will get token after that , save it or the future and "Save settings". We need to remember , the workspace name and the channel name of the slack to integrate with the Jenkins.

5:53

## 16. Continuous Integration with Jenkins 157. Notification, Slack



Download the "slack notification" plugin from plugin manager to integrate slack to the Jenkins.

7:04

## 16. Continuous Integration with Jenkins 157. Notification, Slack



Later , we can integrate it with Jenkins , by Getting to Dashboard>Configuration and there we can see an option called Slack , this is the place , where we can integrate , Here provide our slack workspace name as said and add our credentials with the token that we copied earlier and the new channel that we created and 'test connection' , and if we get success then we can SAVE the connection.

8:11

## 16. Continuous Integration with Jenkins 157. Notification, Slack



To get the notification , we need to integrate slack with Jenkins and we did that , now we need to connect the Jenkinsfile with slack to send the build details to the Slack, so we need to add a post installation step after the stage build.

Before starting pipeline , we must add a function to provide the colors for slack notification , slack take the result as good or danger , so we map our Jenkins SUCCESS & FAILURE to good & danger.

```
1 | def COLOR_MAP = [
2 |     'SUCCESS': 'good',
3 |     'FAILURE': 'danger',
4 | ]
```

And at the end of the pipeline after all the stages , we must add post installation step.

```
1 | post {
2 |     always {
3 |         echo 'Slack Notifications.'
4 |         slackSend channel: '#jenkinscicd',
5 |             color: COLOR_MAP[currentBuild.currentResult],
6 |             message: "*${currentBuild.currentResult}:* Job ${env.JOB_NAME} build
$env.BUILD_NUMBER \n More info at: $env.BUILD_URL"
7 |     }
8 | }
```

Here in this post installation step , we will be using the slack plugin that we downloaded in Jenkins plugins manager and sending notification into the slack channel "#jenkinscicd" with the result color good ( green ) or danger ( red ) and the message as current build result with the job name (pipeline name) and build num with build url

Here , we are going to do the pipeline integrating with the docker , if we want to tell the process in the sentences :

Developer writes his code and publishes into the github after testing in the local with his testcases , and later the Jenkins checks for the updates in the repository , if it finds one , it fetches the code from the repository and perform the unit tests using maven and do the check styling also using maven and later we perform code analysis with sonarqube scanner and if it passes all the quality gates we push the war files and quality gate reports into the sonarqube and after code analysis we docker build the code and get the artifacts as images and publish into the any of the Amazon ECR , Google Cloud Registry , Azure Cloud Registry.

This is the new way of using the variables inclusive to the string.

```
print(f"The name of virus is {name} and it causes {disease}")
```

0:53

**17. Python** 180. Built-in Functions or Methods

<https://docs.python.org/3/library/functions.html>

1:26

**17. Python** 180. Built-in Functions or Methods

[https://www.w3schools.com/python/python\\_ref\\_functions.asp](https://www.w3schools.com/python/python_ref_functions.asp)

-----

[https://www.w3schools.com/python/python\\_ref\\_string.asp](https://www.w3schools.com/python/python_ref_string.asp)

-----

[https://www.w3schools.com/python/python\\_ref\\_list.asp](https://www.w3schools.com/python/python_ref_list.asp)

-----

[https://www.w3schools.com/python/python\\_ref\\_dictionary.asp](https://www.w3schools.com/python/python_ref_dictionary.asp)

-----

[https://www.w3schools.com/python/python\\_ref\\_tuple.asp](https://www.w3schools.com/python/python_ref_tuple.asp)

-----

[https://www.w3schools.com/python/python\\_ref\\_set.asp](https://www.w3schools.com/python/python_ref_set.asp)

-----

[https://www.w3schools.com/python/python\\_ref\\_file.asp](https://www.w3schools.com/python/python_ref_file.asp)

3:59

**17. Python** 180. Built-in Functions or Methods

Strings are immutable , so even if a strings modified by the built in function , it doesn't change the original variable value until we modify and save manually into original variable , this is not called modification of value , this is entirely replacing the value of variable.

5:12

**17. Python** 180. Built-in Functions or Methods

dir(variable) => provides all the classes , methods and built in functions of that particular variable type or an instance type that we mentioned in the brackets of dir.

6:07

**17. Python** 182. Functions part-2

Example for variable arguments and keyword arguments

\* args stores the multiple arguments in a tuple and multiple keyword arguments in dictionary

```

1 | def time_activity(*args, **kwargs):
2 |     #Input: Multiple values for minutes, key=value pair activity
3 |     #Output: Return sum of minutes + random minute spent on a random activity
4 |     print(args)
5 |     print(kwargs)
6 | time_activity(10,20,10, sport="Boxing", fun="Driving", work="DevOps")

```

0:02

**17. Python** 183. Modules

We have a concept called modules , where we can create a function in a file and name it as any py file , for example let's think admin.py then we can access this admin.py and its functions inside it by any other file , using

`from admin import *` => import all functions from admin.py or it is also called as admin module

`from admin import headmaster` => import only headmaster function from the admin module

`import admin` => import admin module with all its internal functions

6:10

**17. Python** 184. OS Tasks

<https://visualpath.in/devopstutorials/devops>

In this link , we are having the total notes for this devops tutorial

8:40

**17. Python** 184. OS Tasks

```

1 |#!/usr/bin/python3
2 |import os
3 |path = '/tmp/testfile.txt'
4 |if os.path.isdir(path):
5 |    print("It is a directory")
6 |elif os.path.isfile(path):
7 |    print("It is a file.")
8 |else:
9 |    print("file or dir doesn't exists")

```

13:39

**17. Python** 184. OS Tasks

```

1  #!/usr/bin/python3
2  import os
3  userlist = ["alpha", "beta", "gamma"]
4  print("Adding Users to system")
5  for user in userlist:
6      exitcode=os.system("id {}".format(user))
7      if exitcode!=0:
8          print("User {} doesn't exist. Adding it.".format(user))
9          os.system("useradd {}".format(user))
10     else:
11         print("User already exist, skipping it.")
12     exitcode = os.system("grep science /etc/group")
13     if exitcode!=0:
14         print("Group science doesn't exists , creating one")
15         os.system("groupadd science")
16     else:
17         os.system("Group already exists , skipping it")
18     for user in userlist:
19         print("Adding user {} in science group")
20         os.system("usermod -G science {}".format(user))
21     print("Adding directory")
22     if os.path.isdir("/opt/science_dir"):
23         print("Directory already exist, skipping it")
24     else:
25         os.mkdir("/opt/science_dir")
26     print("Assigning permission and ownership to the directory")
27     os.system("chown :science /opt/science_dir")
28     os.system("chmod 770 /opt/science_dir")

```

1:38

**17. Python** 185. Python Fabric

File from where we can download the pip

```
wget https://bootstrap.pypa.io/get-pip.py
```

And run this particular get-pip.py to get pip onto our system

3:43

**17. Python** 185. Python Fabric

After getting pip onto system , we can use a python module to manipulate or use jenkins to create jobs and run the jobs and do all the operations of jenkins through the python code. To install that we need to use

```
pip install python-jenkins -y
```

4:46

## 17. Python 185. Python Fabric



<https://python-jenkins.readthedocs.io/en/latest/examples.html>

Jenkins API through python module

3:17

## 17. Python 185. Python Fabric



We like to use python-jenkins to automate our Jenkins servers. Here are some of the things you can use it for:

- Create new jobs
- Copy existing jobs
- Delete jobs
- Update jobs
- Get a job's build information
- Get Jenkins master version information
- Get Jenkins plugin information
- Start a build on a job
- Create nodes
- Enable/Disable nodes
- Get information on nodes
- Create/delete/reconfig views
- Put server in shutdown mode (quiet down)
- List running builds
- Delete builds
- Wipeout job workspace
- Create/delete/update folders [1]
- Set the next build number [2]
- Install plugins
- and many more..

5:00

## 17. Python 185. Python Fabric



We are having boto3 for AWS

Boto is the Amazon Web Services (AWS) SDK for Python. It enables Python developers to create, configure, and manage AWS services, such as EC2 and S3. Boto provides an easy to use, object-oriented API, as well as low-level access to AWS services,

<https://boto3.amazonaws.com/v1/documentation/api/latest/index.html>

If we are having permissions to use configure management tool or automation tool like ansible or terraform , we can use that , but if our needs are not satisfying by them , we can use python libraries.

6:39

## 17. Python 185. Python Fabric



Fabric is a useful Python module that works with SSH and the operating system **to automate a variety of application development and administration activities**. This command-line program is basic and straightforward to use. Fabric is very simple and powerful and can help to automate repetitive command-line tasks. This approach can save time by automating your entire workflow.

```
pip install fabric<2.0
```

10:31

## 17. Python 185. Python Fabric



```
1 | from fabric.api import *
2 | def greeting(msg):
3 |     print("Good %s"%msg)
4 | def system_info():
5 |     print("Disk Space")
6 |     local("df -h")
7 |     print("RAM size")
8 |     local("free -m")
9 |     print("System uptime.")
10|    local("uptime")
11|def remote_exec():
12|    print("Get System info")
13|    run("hostname")
14|    run("uptime")
15|    run("df -h")
16|    run("free -m")
17|    sudo("yum install mariadb-server")
18|    sudo("systemctl start mariadb")
19|    sudo("systemctl enable mariadb")
20|def web_setup(WEBURL,DIRNAME):
21|    local("apt install zip unzip -y")
22|    sudo("yum install httpd wget unzip -y")
23|    sudo("systemctl start httpd")
24|    sudo("systemctl enable httpd")
25|    local(("wget -o website.zip %s")%WEBURL)
26|    local("unzip -o website.zip")
27|    with lcd(DIRNAME):
28|        local("zip -r tooplate.zip *")
29|        put("tooplate.zip","/var/www/html/",use_sudo=True)
30|    with cd("/var/www/html/"):
31|        sudo("unzip tooplate.zip")
```

```
32 | sudo("systemctl restart httpd")
33 | print("Website setup is done")
```

6:46

## 17. Python 185. Python Fabric



After installing fabric , we need to make a directory called fabric

```
1 | mkdir fabric
2 | cd fabric/
3 | vim fabfile.py
```

Then after filing the code that mentioned below , write this command.

```
fab -l
```

This initializes the fabric module in this directory and get available commands to execute.

And to execute the function written in the file, using fabric use this command.

```
fab greeting:Evening
```

Then the output will be like "Good Evening". Fabric module executes this function. using fab command.

And if we pass the `fab systeminfo` , as parameter , then we get Diskspace , RAM size , System uptime as the output , which is declared in the systeminfo function. Remember this python file is declared in the same folder as the fab initialized.

11:34

## 17. Python 185. Python Fabric



And to execute the remote\_exec function through fabric , we need to give the credentials for the remote server to run the commands on , and we need to do it through the command line.

```
fab remote_exec
```

We need to have user and password for some user on the server , so we need to create one using,

```
1 | useradd devops
2 | passwd devops
```

create the following through root user to pass the security restrictions.

Later we need to provide all the permissions for this user 'devops' else we will have the security issue. Proceed with that using

```
visudo
```

Add the following line in visudo: `devops ALL=(ALL) NOPASSWD:ALL`

Then you will not get asked about the password when you are trying to sudo the command from the devops user.

And enable the password authentication (PasswordAuthentication yes) , for this user using this command

`vi /etc/ssh/sshd_config` then when we are trying to login to this user through ssh , we will be prompted with the password.

After editing these , we need to restart the server , to apply changes.

16:07

## 17. Python 185. Python Fabric



And if we want to login to the server through ssh , then we need to use Password based login:

`ssh devops@192.168.10.3` => `ssh username@ip-address` , then we need to enter the password to enter to the server for logging in.

16:17

## 17. Python 185. Python Fabric



But to login passwordless , we need to generate a ssh key through ssh-keygen from the server that we are in using `ssh-keygen` , and get the ssh key

And now , we need to copy the ssh key to the server that we want to login passwordless using `ssh-copy-id devops@192.168.10.3` , for one last time , we need to enter the password and from next on , we can login passwordless , without entering the password.

18:59

## 17. Python 185. Python Fabric



Now as we established passwordless login , we can login to the user and execute the fabric function remote\_exec that we created and tried to execute remotely.

`fab -H <ip-address/server-name> -u <user-name> <function-to-exec>`

19:43

## 17. Python 185. Python Fabric



The sudo commands that we have given under remote\_exec() function , use the root privileges and execute the commands in root level.

22:23

## 17. Python 185. Python Fabric



When we execute web\_setup function , we are having two types of fabric commands here , one is local and other is sudo , local runs those commands locally from the place

where the host server is executing and the sudo function works on the destination server with root privileges.

10:34

17. Python 185. Python Fabric



Here , we can see that using fabric api module , we can execute the linux commands in the python it is similar to the os.system , this is mainly used to automating the repeatable command line tasks.

22:48

17. Python 185. Python Fabric



Here the lcd function searches the directory on the local , and here in that with clause , we are ziping the tooplate recursively and putting it onto the /var/www/html of remote user , with sudo privileges.

And in further , we can search current directory , using the cd function and in here , we are unzipping the zip file at the destination directory or remote server , we can assume this is happened on remote server because , cd checks the directory structure of remote so the contents of it also could be remote. Here basically sudo executes the command on the remote server. and finally restarting the httpd on the remote server after successful installation of resources on the remote by the local.

29:53

17. Python 185. Python Fabric



### How to create a virtualenv:

```
pip install virtualenv
```

```
virtualenv <virtual-env-name>
```

We will get a folder naming <virtual-env-name> , where we will have all the python and every stuff which is independent from the system python , it is it's own environment.

Now , if we want to activate it , we need to type

```
source ./<virtual-env-name>/bin/activate
```

To deactivate the virtual environment , we need to do

```
source ./<virtual-env-name>/deactivate
```

3:27

16. Continuous Integration with Jenkins 159. Docker PAAC Prereqs info



Here , if we want to connect the Jenkins to the ECR , we need the following:

\* IAM user with ECR permissions ( Elastic container Registry ) in AWS

\* Store AWS credentials in Jenkins

\* Create ECR repository on AWS

After having these , we need to set some environmental variables.

\* registryCredential => This should be created and taken from AWS , after creating an IAM user with ECR permissions in AWS

\* appRegistry => This is the docker image location , that use to access from the AWS

\* vprofileRegistry => This registry is the home of all images including the image that we mentioned in appRegistry.

4:55

## 16. Continuous Integration with Jenkins 159. Docker PAAC Prereqs info



And if we need to run docker on the Jenkins , we need to do some setup's first

- 1) Install docker pipeline plugin
- 2) Install docker engine on Jenkins
- 3) Install ECR plugin

8:07

## 16. Continuous Integration with Jenkins 159. Docker PAAC Prereqs info



Docker CI in Jenkins:

- 1) Install docker engine in Jenkins
  - \* Add jenkins user to docker group & reboot
- 2) Install AWS CLI
  - \* This is generally used while we are performing Continuous Delivery
- 3) Create IAM user
  - \* So we can get the access and security for the aws.
- 4) Create ECR Repository
  - \* This is the place we store the docker images.
- 5) Plugins
  - \* ECR , Docker pipeline , AWS SDK for credentials
- 6) Store AWS credentials in Jenkins
- 7) Run the pipeline

9:18

## 16. Continuous Integration with Jenkins 159. Docker PAAC Prereqs info



```
1 | pipeline {  
2 |     agent any
```

```
3     tools {
4         maven "MAVEN3"
5         jdk "OracleJDK8"
6     }
7
8     environment {
9         registryCredential = 'ecr:us-east-2:awscreds'
10        appRegistry = "951401132355.dkr.ecr.us-east-
11            .amazonaws.com/vprofileappimg"
12        vprofileRegistry = "https://951401132355.dkr.ecr.us-east-2.amazonaws.com"
13    }
14}
```

9:22

## 16. Continuous Integration with Jenkins 159. Docker PAAC Prereqs info



Write this code under the pipeline block

```
1     stages {
2         stage('Fetch code'){
3             steps {
4                 git branch: 'docker', url: 'https://github.com/devopshydclub/vprofile-
5                     project.git'
6             }
7         }
8         stage('Test'){
9             steps {
10                sh 'mvn test'
11            }
12        }
13        stage ('CODE ANALYSIS WITH CHECKSTYLE'){
14            steps {
15                sh 'mvn checkstyle:checkstyle'
16            }
17            post {
18                success {
19                    echo 'Generated Analysis Result'
20                }
21            }
22        }
23    }
```

9:24

## 16. Continuous Integration with Jenkins 159. Docker PAAC Prereqs info



```
1     stage('build && SonarQube analysis') {
2         environment {
3             scannerHome = tool 'sonar4.7'
4         }
5         steps {
6             withSonarQubeEnv('sonar') {
```

```

7   sh '''${scannerHome}/bin/sonar-scanner - 
8     Dsonar.projectKey=vprofile \
9       -Dsonar.projectName=vprofile-repo \
10      -Dsonar.projectVersion=1.0 \
11      -Dsonar.sources=src/ \
12      -Dsonar.java.binaries=target/test-
13      classes/com/visualpathit/account/controllerTest/ \
14      -Dsonar.junit.reportsPath=target/surefire-reports/ \
15      -Dsonar.jacoco.reportsPath=target/jacoco.exec \
16      -Dsonar.java.checkstyle.reportPaths=target/checkstyle-
17      result.xml''' 
18   }
19 }
20 }
```

9:25

## 16. Continuous Integration with Jenkins 159. Docker PAAC Prereqs info



```

1  stage("Quality Gate") {
2    steps {
3      timeout(time: 1, unit: 'HOURS') {
4        // Parameter indicates whether to set pipeline to UNSTABLE
5        if Quality Gate fails
6          // true = set pipeline to UNSTABLE, false = don't
7          waitForQualityGate abortPipeline: true
8        }
9      }
10 }
11
12 stage('Build App Image') {
13   steps {
14     script {
15       dockerImage = docker.build( appRegistry + ":$BUILD_NUMBER",
16         "./Docker-files/app/multistage/")
17     }
18   }
19 }
20
21 stage('Upload App Image') {
22   steps{
23     script {
24       docker.withRegistry( vprofileRegistry, registryCredential ) {
25         dockerImage.push("$BUILD_NUMBER")
26         dockerImage.push('latest')
27       }
28     }
29   }
30 }
```

0:27

## 16. Continuous Integration with Jenkins 160. Docker PAAC Demo



First , we need to login to the Jenkins installed server , and we need to download the docker in it. So , for installing , we need to refer the official documentation , of docker.

Refer this site for that ,

```
https://docs.docker.com/engine/install/ubuntu/
```

We need to install docker as a root user.

2:02

## 16. Continuous Integration with Jenkins 160. Docker PAAC Demo



Here , after installation of docker , we must make that jenkins user to have docker group mapped for it , so jenkins can access and use the docker commands.

2:18

## 16. Continuous Integration with Jenkins 160. Docker PAAC Demo



Now to check that jenkins user have docker group in it or not , we must use

```
id jenkins
```

Then we would see that docker group is present with jenkins user or not

So , if it is not present , we must add the docker group to the jenkins user.

```
usermod -a -G docker jenkins
```

This means , we are adding the docker group to the jenkins user. so , we can execute the docker commands in jenkins terminal with jenkins user.

2:24

## 16. Continuous Integration with Jenkins 160. Docker PAAC Demo



Now , we also needed to install awscli , so that we can push our build images into the aws container registry to store all the docker images , that we built.

```
apt install awscli -y
```

Basically , `reboot` command reboots the terminal , or else we have to restart jenkins as docker is a new group which is assigned to the jenkins user.

3:06

## 16. Continuous Integration with Jenkins 160. Docker PAAC Demo



Now we are going to create an IAM user and the ECR repository for uploading the docker images into the AWS Container Registry.

Now we came to AWS to create an IAM User , we need to search IAM user under the Services section of AWS , then in access management , we need to go to users section and create a new user / Add user.

After clicking the Add user , we need to give the username and need to check the checkbox of Access key - programmatic access , before going to next page , because here we are creating a user called 'Jenkins' to which we can login through access key by Jenkins.

And in there , we can see set permissions page , where we have three options to edit on, we can choose one of those.

- 1) Add user to group
- 2) Copy permissions from existing user
- 3) Attach existing policies directly

And now select the option 3 , and search or AmazonEC2ContainerRegistryFullAccess and check the option. And we are going to deliver the AWS images to AmazonECS\_FullAccess , so we can continuously deliver the images to that service.

And then create the user with those permissions. And after creating the user , we will get access key and secret key to access the user. So in the last page where it shows the secret key and the access key , we can download the the details as csv format , so download the details.

After creating an IAM user , we need to create an ECR repository. Go to the services and search for 'Elastic Container Registry' and click 'Create Repository' , there we must give the repository name , and we must remember that. And create a repository , and here we can copy the URI of the ECR repository , so we can use that in Jenkins.

Now we need to login to the Jenkins server and download the plugins that are needed to connect the Jenkins with the AWS and Docker.

We need to go to Plugin manager which is under the dashboard , download the plugins which are present in there.

- 1) Docker pipeline (Build and use Docker containers from pipelines)
- 2) Amazon ECR (This plugin generates Docker authentication token from Amazon credentials to access Amazon ECR)
- 3) Amazon web services SDK (This plugin provides all AWS SDK for Java modules not packaged as standalone plugins)
- 4) CloudBees Docker Build and Publish (This plugin enables building Dockerfile based projects , as well as publishing of the built images/repos to the docker registry)

And install them without restart

6:34

## 16. Continuous Integration with Jenkins 160. Docker PAAC Demo



Now to add AWS ECR credentials here , we must go the  
Dashboard > Credentials > System > Global Credentials (unrestricted) > Add Credentials

6:59

## 16. Continuous Integration with Jenkins 160. Docker PAAC Demo



And in here , we must see the kind of the credentials is 'AWS credentials' , and if this is not there , it means that the plugins are not installed correctly , so we need to re install them , and then we can give the access key id and secret access key id in here that we got earlier under creating a new IAM user

7:47

## 16. Continuous Integration with Jenkins 160. Docker PAAC Demo



Now as the setup is done , we just edit the code in the pipeline and run the job , as we are already having AWS credentials , we pass out into AWS passwordless.

8:04

## 16. Continuous Integration with Jenkins 160. Docker PAAC Demo



update the ECR details as required in the pipeline script.

In the pipeline , we are declaring the pipeline environmental variables , one of them is

\* registryCredential = 'ecr:<region>:<credential-id-that-we-given-under-manage-credential>'

\* appRegistry = 'REGISTRYUrl/registryname'

\* registry url is the one , we have in ECR registry and the name is the one that we given for the registry.

\* appRegistry is the image name inclusive of registry name

- \* vprofileRegistry is the one ECR registry.
- \* appRegistry includes the ECR registry detail with the image name and the tag
- \* And vprofileRegistry is the one we have to declare the path of registry through the https

10:31

## 16. Continuous Integration with Jenkins 160. Docker PAAC Demo



After editing the pipeline script , we can build the pipeline.

0:14

## 16. Continuous Integration with Jenkins 161. Docker CICD Intro



We are going to extend our continuous Integration pipeline to continuous delivery in here.

0:44

## 16. Continuous Integration with Jenkins 161. Docker CICD Intro



At the final stage of continuous integration , we are going to publish our docker image to amazon ECR , now we need to host this docker image or the docker container to a docker solution like ECS , ECS is docker container hosting platform. we host our dockerized application on ECS

1:34

## 16. Continuous Integration with Jenkins 161. Docker CICD Intro



Amazon ECS is going to fetch the docker image that we published into Amazon ECR , it fetches the latest image and host it on ECS.

3:21

## 16. Continuous Integration with Jenkins 161. Docker CICD Intro



We are going to talk about the docker hosting platforms, we have containerised or dockerized our application , we created docker images for that , and where we need to host it , one of the method is the local method or normal method , it is having a docker engine and just we use docker run by mentioning the image name and it runs your application on the docker engine.

This type of local storing of images and running in local is for testing and local development environment. but in reality , we do not want to run containers directly on docker engine , because we need to manage all the settings that we need to do on

docker engine and the platform where it is running , for example EC2 instance or virtual machines or physical server.

Even if we do all these , we couldn't have all production related features like high availability , self-healing and there are many other things. For production , we can get all these features in Kubernetes platform.

3:26

## 16. Continuous Integration with Jenkins 161. Docker CICD Intro



For deploying/storing the images , in production , we are having so many standalone kubernetes platforms , we can use the EKS (Elastic Kubernetes Service) which is from AWS , AKS (Azure kubernetes service) , GKE (Google Kubernetes engine) , OpenShift which is from Redhat.

And for deploying our container , we are currently using the Amazon ECS (Elastic container service).

1:45

## 16. Continuous Integration with Jenkins 162. Docker CICD Code



In Elastic Container Service (ECS), a cluster is a logical grouping of one or more EC2 instances or Fargate tasks that run containerized applications.

When you create an ECS cluster, you can specify the number of EC2 instances or Fargate tasks to include in the cluster, as well as other configuration details such as the instance type, AMI (Amazon Machine Image), and networking settings.

Once you have created an ECS cluster, you can deploy containerized applications to it by defining a task definition that describes the containers, resources, and networking requirements for your application. You can then create a service to run your task definition on the cluster and specify the desired number of tasks to run at any given time.

1:47

## 16. Continuous Integration with Jenkins 162. Docker CICD Code



ECS clusters are designed to be highly scalable and fault-tolerant. You can add or remove EC2 instances or Fargate tasks from a cluster at any time, and ECS will automatically manage the placement of tasks on the available resources. Clusters can also be integrated with other AWS services, such as Elastic Load Balancing and AWS Identity and Access Management (IAM), to provide additional functionality and security.

1:49

## 16. Continuous Integration with Jenkins 162. Docker CICD Code



In Elastic Container Service (ECS), a service is a configuration that allows you to run and maintain a specified number of instances of a task definition simultaneously in a cluster.

A task definition in ECS defines how a containerized application should be launched and what resources it requires. A service builds on top of this concept by specifying the desired number of tasks to run and how they should be deployed and managed.

When you create an ECS service, you specify the task definition to use, the number of tasks to run, and other configuration details such as the launch type (Fargate or EC2), networking settings, and load balancing. Once the service is created, ECS will automatically start and manage the specified number of tasks to ensure that the desired number of instances of the application are always running.

1:50

## 16. Continuous Integration with Jenkins 162. Docker CICD Code



ECS services can be updated and scaled dynamically to meet changing demands. For example, you can update a service to use a new version of a task definition, and ECS will automatically launch new tasks with the updated definition while stopping the old ones. You can also scale a service up or down by changing the desired number of tasks, and ECS will automatically adjust the number of running tasks accordingly.

Services in ECS can be integrated with other AWS services such as Elastic Load Balancing and Auto Scaling to provide additional functionality and scalability.

1:52

## 16. Continuous Integration with Jenkins 162. Docker CICD Code



```
1 | environment {
2 |     registryCredential = 'ecr:us-east-2:awscreds'
3 |     appRegistry = "951401132355.dkr.ecr.us-east-
4 | .amazonaws.com/vprofileappimg"
5 |     vprofileRegistry = "https://951401132355.dkr.ecr.us-east-2.amazonaws.com"
6 |     cluster = "vprofile"
7 |     service = "vprofileappsvc"
8 | }
```

We need to mention the cluster name and the service name that ECS gonna use , and this cluster and service needs to be created in the Elastic container service.

1:53

## 16. Continuous Integration with Jenkins 162. Docker CICD Code



```
1 | stage('Deploy to ecs') {
2 |     steps {
3 |         withAWS(credentials: 'awscreds', region: 'us-east-2') {
4 |             sh 'aws ecs update-service --cluster ${cluster} --service ${service} --
5 | force-new-deployment'
6 |         }
7 |     }
8 | }
```

```
6 |     }
7 | }
```

And we also need to declare this stage to deploy the image to the Elastic container service , where we are giving the AWS credentials and the region details in here , and if those details are correct then , we are updating the service with the latest image , which will be triggered through the '--force-new-deployment' , we are providing the cluster and service details in here , by shell command.

--force-new-deployment => creates new task in the service with latest image , it fetch the image , run the container and remove the older image and update with the new image from Amazon ECR

2:06

## 16. Continuous Integration with Jenkins 163. AWS ECS Setup



How to create ECS cluster and the service inside that? And we are also going to install Jenkins plugin required to use this cluster.

We can find the ECS (Elastic container service) in the AWS services and if we have a prompt to switch to new ECS experience , we need to switch to that. And then Get started with the setup of the ECS.

In general , Amazon ECS is a highly scalable and fast container management service that makes it easy to run , stop and manage container on a cluster.

0:45

## 16. Continuous Integration with Jenkins 163. AWS ECS Setup



When we are creating the new cluster , we need to give the cluster name and VPC and subnet details which are part of networking details , we are going to leave those details by default.

By default tasks and services run in the default subnets for our default VPC. For using non default VPC , we need to specify the VPC and the subnets.

Subnets are where our tasks run, we need to at least use three subnets in production.

1:38

## 16. Continuous Integration with Jenkins 163. AWS ECS Setup



And we have the option to choose the infrastructure for our ECS. Our ECS can run our EC2 instances , so it can launch an auto scaling group on EC2 instances , it will managed by the ECS. This auto scaling group is server based , it is based on the EC2 server , but AWS provides , us much better solution or it is also called as serverless solution for our containers. We no need to worry about the EC2 instance or we don't even need to worry about where our containers are running.

1:43

## 16. Continuous Integration with Jenkins 163. AWS ECS Setup



This serverless solution is named as AWS Fargate , it is a pay as you go , service , we can use it if we have tiny , batch or burst workloads or for zero maintenance overhead. We also have two other options to choose as infrastructure , one of them is Amazon EC2 instances , this is the manual configurations , used for large workloads with consistent resource demands. And another option is External instances using ECS anywhere , for this we need to manually add the configurations and EC2 instance details so , we can use the ECS anywhere we need. As we know EC2 instance exists in various countries and several cities in the same countries.

1:51

## 16. Continuous Integration with Jenkins 163. AWS ECS Setup



We will be having another option while creating the ECS , that is monitoring , we are enabling that option. so we can see the logs of our container when it is coming up. This monitoring option is off by default. We will have separate cost for this option to use.

If we use this cloud watch , we automatically collects metrics for many resources , such as CPU , memory , disk , and network. Container insights also provides diagnostic information such as container restart failures , that we use to isolate issues and resolve them quickly. We can also set the cloudwatch alarms on metrics that container insights collects.

And next we create the ECS (Elastic Container Service) cluster, it takes time to create one.

2:16

## 16. Continuous Integration with Jenkins 163. AWS ECS Setup



After creating the cluster , we can see that on the screen , we need to create a task and run. In the task we mention the image that needs to be fetched and run. Our image is in Elastic Container Registry.

2:23

## 16. Continuous Integration with Jenkins 163. AWS ECS Setup



So , now we need to create a task definition , give name for it , container name , image uri of image which is present in Elastic container Registry. And while doing that , set the container port as 8080 and Protocol as TCP.

3:31

## 16. Continuous Integration with Jenkins 163. AWS ECS Setup



And in the next page of creation of the task definition we need to configure the environment , storage , monitoring , and tags , where we need to declare the App environment , it means we need to declare the infrastructure for the ECS task that we are creating , we are assigning it to AWS fargate which is serverless and zero maintenance and cost effective. We need to mention the Operating system , CPU's and Memory to reserve our task.

3:52

## 16. Continuous Integration with Jenkins 163. AWS ECS Setup



Leaving everything as default , we are creating the task definition for the cluster.

4:21

## 16. Continuous Integration with Jenkins 163. AWS ECS Setup



After creating the task definition , we come to the clusters home page on Amazon Elastic Container Service. And then we can see multiple tabs on the page , (Services , Tasks , Infrastructure , Metrics , Tags) , we can go to the tasks tab and directly run a task through task definition that we have created , but we are going to create a service to run the task.

4:28

## 16. Continuous Integration with Jenkins 163. AWS ECS Setup



To create the task , we need to go under deploy of service , where we can have service option and we need to select our task definition from our drop down of 'Family' , need to declare the service name , and specify number of tasks to launch in Desired tasks. And we can also , declare the minimum & maximum number of tasks to run at any instance.

5:13

## 16. Continuous Integration with Jenkins 163. AWS ECS Setup



We also need to create a load balancer for our service that runs the tasks , this load balancer is used to distribute incoming traffic across the tasks running in your service. We choose the load balancer as 'Application Load Balancer'. Name the load balancer and declare the port number of load balancer as 80 as it listens at that port. And protocol as HTTP.

6:24

## 16. Continuous Integration with Jenkins 163. AWS ECS Setup



And we also needed to give the security group for the Amazon ECS. And deploy our service.

9:07

## 16. Continuous Integration with Jenkins 163. AWS ECS Setup



And suppose , if we are hosting any application on any port then we need to edit our inbound rules for the EC2 instance, for example if we want to host an application at 8080 then we need to Add inbound rules to accept or provide info for the port 8080 for any IP address (IPV4,IPV6) or for any specific IP address of (IPV4,IPV6).

9:51

## 16. Continuous Integration with Jenkins 163. AWS ECS Setup



Here , you can see , we are having two port numbers one is indirectly mentioned in target group and another is listener. Listener specify the port and protocol that the load balancer will listen for connection requests on.

We create a target group that the load balancer will user to route request to the tasks in your service.

7:59

## 16. Continuous Integration with Jenkins 163. AWS ECS Setup



The target for the load balancer is the tomcat service which runs the web application , and this is the one which we launch application from inside the container , it is hosted on 8080 ,so when we access 8080 we get load balancer services from port 80 , we need to give the necessary security group permissions for 8080 to access the website.

9:54

## 16. Continuous Integration with Jenkins 163. AWS ECS Setup



load balancer target port which have the original application is on port 8080 and the load balancer listens at port 80

10:11

## 16. Continuous Integration with Jenkins 163. AWS ECS Setup



We can access the application using load balancer output service address , or using task ip address directly using port 8080.

0:20

## 16. Continuous Integration with Jenkins 164. Docker CICD Demonstration



After creating cluster , tasks , service and load balancer , we need to connect it to the Jenkins pipeline , so we can automate the task.

Now to use the following in our Jenkinsfile , we need to download the Jenkins plugin which helps us to integrate them into our pipeline , so we need to go to plugin manager , and search for 'Pipeline: AWS Steps' plugin , install the plugin.

2:00

## 16. Continuous Integration with Jenkins 164. Docker CICD Demonstration



Create a new pipeline with the following code. The stages in here are :

- Fetch code
- Test
- CODE ANALYSIS WITH CHECKSTYLE
- build && SonarQube analysis
- Quality Gate
- Build App Image
- Upload App Image
- Deploy to ecs

4:16

## 16. Continuous Integration with Jenkins 164. Docker CICD Demonstration



And after running the pipeline , we execute a command 'aws ecs update-service --cluster \${cluster} --service \${service} --force-new-deployment' , which updates the service with a new task to be created and executed so , the old containers in it will be replaced by the new ones.

0:44

## 16. Continuous Integration with Jenkins 165. Cleanup



If we want to delete an ECS , we couldn't do it directly , first we have to make number of tasks in a service to 0 and delete the service and then delete the cluster.

0:24

## 16. Continuous Integration with Jenkins 166. Build Triggers Intro



Triggers are basically , which automate the Jenkins pipeline execution process, until now , we are manually building the pipeline using 'Build Now' or 'Build Now with Parameters'.

But now we are automating the pipeline execution or the build process using the Job triggers.

2:20

## 16. Continuous Integration with Jenkins 166. Build Triggers Intro



Popular triggers:

- Git Webhook
  - We will send an adjacent payload whenever there is a commit in the repository , so GitHub repository will trigger our Jenkins Job.
- Poll SCM
  - It is similar to Git Webhook , but here we define an interval , so after that interval , we will fetch if there are any new commits in the repository , and if there are any the Job get's triggered.
- Scheduled jobs
  - This is the simplest once , we mention date and time like an alarm clock in cronjob format and Jenkins will make sure our job runs at that particular time or in some intervals that specified.
- Remote triggers
  - This is the most useful trigger for DevOps Engineer , by using this we can trigger Jenkins jobs from anywhere , from a script , from an Ansible playbook , from anywhere that matter , And we will have the tokens and secrets for these to set if needed , we can also get an API call which we can use to trigger the Jenkins Job.
- Build after other projects are built

2:27

## 16. Continuous Integration with Jenkins 166. Build Triggers Intro



Popular Triggers:

- Build after other projects are built
  - Auto Execution of current Job depends on the previous Jobs.

3:17

## 16. Continuous Integration with Jenkins 166. Build Triggers Intro



Auto Build triggers:

1. Create git repository on GitHub
2. SSH AUTH
3. Create a Jenkins file in git repo & commit
4. Create Jenkins Job to access Jenkins file from git repo

3:34

**16. Continuous Integration with Jenkins** 166. Build Triggers Intro

Login to GitHub account and create a new Git Repository with any name as convenient. Here , we naming it as 'jenkinstriggers' and we can choose that repository to be private or public. And after creating a repository , we are going to use that repository with the SSH access of the repo.

4:43

**16. Continuous Integration with Jenkins** 166. Build Triggers Intro

Prior to access that repository , we need to have the ssh key generated on our terminal , so we can give permissions to access the remote repository passwordless.

To generate the ssh key , use `ssh-keygen.exe` and generate the keys , we can find the key's are generated or not under , `ls ~/.ssh/` as `id_rsa.pub & id_rsa`

We need to get the public key of ssh from `id_rsa.pub`. And we need to go to the github account settings , under which we can see the SSH and GPG keys. And add a new SSH key here. we need to add our public ssh key in here. This makes us to access the repository passwordless.

After doing that , we can clone the git repository directly by using the `git clone`

7:29

**16. Continuous Integration with Jenkins** 166. Build Triggers Intro

```

1 | pipeline {
2 |     agent any
3 |     stages {
4 |         stage('Build') {
5 |             steps{
6 |                 sh 'echo "Build completed."'
7 |             }
8 |         }
9 |     }
10 |

```

We are adding this Jenkinsfile to the repository , for the demonstration of auto triggering of the build.

Commit the code using `git commit -m ""`

Push the code into repository after committing `git push origin <master>` , here master is the branch name.

9:05

## 16. Continuous Integration with Jenkins 166. Build Triggers Intro



Now we need to go to Jenkins and make the Job to fetch this particular Jenkinsfile from the Repository and Build the Job.

10:03

## 16. Continuous Integration with Jenkins 166. Build Triggers Intro



Sometimes , for the first time , when we are Entering into any repository , we will be asked about ssh key fingerprint (y/n) , so this is due to we have strict host key verification enabled , to resolve this issue in upfront , we need to go to 'Manage Jenkins' and then 'Configure Global Security'.

And in there , we will have an option called 'Git hostkey verification configuration'. The default value of it would be the 'Known Hosts file' change it to 'Accept First Connection' and apply and save the settings.

10:57

## 16. Continuous Integration with Jenkins 166. Build Triggers Intro



And now after these settings , we need to go to the Jenkins Dashboard , and create a Job , select pipeline from the options , and in following configuration under pipeline , we will have two options to have as an input , 1 is Pipeline script & other is Pipeline script from SCM.

We need to select 'Pipeline script from SCM' , then we will see some options

11:04

## 16. Continuous Integration with Jenkins 166. Build Triggers Intro



We need to select the SCM , and enter the repository details , for example , Repository URL ( we use SSH URL as Repository URL ) , add the Credentials and in the Options select the credentials that we provide is in the form of 'SSH username with private key'. Provide the ID , Description and Github username , Github private key (we can find this at `ls ~/ssh/` at `id_rsa` ). Remember this is not public key , this is the private key.

We need to add private key in Jenkins configurations ( This is done for giving Jenkinsfile input from the SCM )

We need to add public key in Github ( This is done for passwordless login)

And we also need to provide the branch and the Script path of Jenkinsfile that we want to execute. After giving those , we can save and build now the Jenkins Job.

Now we are going to add a webhook , so go into repository settings and search for webhooks option , for which we need the IP of Jenkins , so copy the URL of how we are accessing Jenkins through a browser untill 8080 , it looks like

`http://18.221.221.216:8080/` paste this in github repository settings 'Payload URL' as  
`http://18.221.221.216:8080/github-webhook/` and content type as 'application/json'

Here in these settings , we are going to select option for the question 'Which events would you like to trigger this webhook' We can select the event as convenient and add the webhook. For default we are choosing it to be the 'push' event as the triggering event.

After creating a webhook , we can see our webhook has been delivered. If it isn't , first case we need to verify the Jenkins URL and the Payload URL that we mentioned in Git webhook configuration are same , and then we need to see the security group for the Jenkins Instance , where we have to see that Jenkins must be accessible from any URL , if not this could be the error. Because Github couldn't reach the Jenkins Server.

And if we want to make that trigger to auto work when ever we perform the push code event into repository , we need to configure the job , and where we need to select the following build trigger , 'GitHub hook trigger for GITScm polling'.

As we already mentioned the Jenkins Job to take the code from the particular repository and of particular branch , now we also mentioned it to use particular webhook to trigger the Job. So immediately after any push in the code into the repository , the code will get autobuild.

Now we are using the Poll SCM , for Github hooks the github triggers the Job whenever there is a commit , but now in Poll SCM , the Jenkins searches for new commits and triggers the Job whenever there is a new commit.

4:41

## 16. Continuous Integration with Jenkins 167. Build Triggers Demo



And if we are using the 'poll SCM' as the trigger , we need to set a schedule to run that , we need to give that schedule in cronjob format '`'MINUTE HOUR DOM MONTH DOW'`' , so if we want to run every seconds , we need to give `* * * * *` , and we need to save that , and then we can see 'Git polling Log' under the Build section of the Dashboard. Where we can see the details of the poll SCM , it is nothing but the logs of poll SCM , it shows when was the last poll done. And this pollSCM executes in Jenkins , it checks every second that we have a new commit or not in the repository. And logs it under Git polling log and if there is a commit , the build will be auto trigger and pipeline code executes.

5:51

## 16. Continuous Integration with Jenkins 167. Build Triggers Demo



Git Webhook , sends a Json payload to the Jenkins immediately after there is some commit happens in the Github repository from the side of Github. And here alternatively , Jenkins it's self find and scans the commits that are happening inside the Github using Poll SCM.

Git Webhook => Github sending the request run Jenkins Job when commit happens  
Poll SCM => Jenkins scannes the Github for any new commit , and if it found it trigger the Jenkins Job. We need to give the schedule to scan the repository for new commits , the schedule is represented using cron job syntax.

7:01

## 16. Continuous Integration with Jenkins 167. Build Triggers Demo



Scheduled Jobs : Building the Jobs periodically , irrespective of commit's code changes . We can find the option in Jenkins Job Configuration (Dashboard > Build > Configure) , and where under 'Build triggers' we can see an option called 'Build periodically' , under which we can declare the schedule , where we can declare the schedule in terms of cron job. If we want to mention , monday to friday of the week in cronjob , we can represent the Day of the Week (DOW) as '1-5'

7:34

## 16. Continuous Integration with Jenkins 167. Build Triggers Demo



**Remote Triggers :** We can trigger the Jenkins Job from another script or another Jenkins Server or from our laptop unless we have network access for our Jenkins server.

16:06

## 16. Continuous Integration with Jenkins 167. Build Triggers Demo



### Build Triggers Remotely

Generate JOB URL

1. Job Configure => Build Triggers
2. Check mark on “Trigger builds remotely”
3. Give a token name
4. Generate URL & save in a file

Generate Token for User

1. Click your username drop down button (Top right corner of the page)
2. configure => API Token => Generate
3. Copy token name and save username:tokenname in a file

Generate CRUMB

1. wget command is required for this, so download wget binary for git bash
2. Extract content in c:/program files/Git/mingw64/bin
3. Run below command in Git Bash, (replace username,password,Jenkins URL)  
wget -q --auth-no-challenge --user username --password password --output-document - 'http://JENNKINS\_IP:8080/crumbIssuer/api/xml?xpath=concat(//crumbRequestField,":",//crumb)' 4. Save the token in a file

16:12

## 16. Continuous Integration with Jenkins 167. Build Triggers Demo



### Build Job from URL

By now we should have below details

1. JENKINS Job URL with token

E:g http://52.15.216.180:8080/job/vprofile-Code-Analysis/build?token=testtoken

2. API Token USERNAME:API\_TOKEN

E:g admin:116ce8f1ae914b477d0c74a68ffcc9777c

3. Crumb

E:g Jenkins-Crumb:8cb80f4f56d6d35c2121a1cf35b7b501

Fill all the above details in below URL and Execute

```
curl -I -X POST http://username:APIToken @Jenkins_IP:8080/job/JOB_NAME/build?  
token=TOKENNAME -H "Jenkins-Crumb:CRUMB"
```

```
e:g curl -I -X POST  
http://admin:110305ffb46e298491ae082236301bde8e@52.15.216.180:8080/job/  
vprofile-Code-Analysis/build?token=testtoken -H "Jenkins-  
Crumb:8cb80f4f56d6d35c2121a1cf35b7b501"
```

16:54

## 16. Continuous Integration with Jenkins 167. Build Triggers Demo



Build Job after another Job was done:

Suppose think Two Jobs A and B , where B job depends on the execution of A , So , now we need to go to B Job and configure it , we will go under the build triggers , under which we can see the 'Build after other projects are built' , where we can give the Job name of A , so whenever the Job A executes and Done , Job B auto executes. These are some common build triggers that are available in the market.

1:46

## 6. Variables, JSON & YAML 49. Introduction



Data Exchange in DevOps tool sets uses mostly JSON & YAML, As a DevOps Engineer it's essential to understand , read & write this data format for data exchange between different applications and languages.

2:39

## 6. Variables, JSON & YAML 50. Variables & Python DS



We need to use double quote while using echo statement to print some variable inclusive to the statement

```
name = "Jeevan"  
"Hello $name" => Hello Jeevan  
'Hello $name' => Hello $name
```

0:38

## 9. Introducing Containers 67. What is Docker



Docker is an open platform for developing, shipping, and running applications. Docker enables

you to separate your applications from your infrastructure so you can deliver software quickly.

1:34

## 9. Introducing Containers 67. What is Docker



Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allows you to run many containers simultaneously on a given host. We no need multiple computers to run multiple containers , the containers may used to run the same process or different processes.

2:27

## 9. Introducing Containers 67. What is Docker



We will be having such images , which are an applications of itself , which can be found in dockerhub , these are called docker images technically , which can be run as containers to run that application , image is just an abstract of the application , container is the real process.

3:00

## 9. Introducing Containers 67. What is Docker



These are the following sample docker commands , which can be run on client to run on the docker host. Basically the client and the docker host , remains on the same server , but sometimes , those can be on different servers , we run the docker commands on the client and that triggers those commands on the docker host , which execute them using docker daemon , if we are using a docker run command , and if we not having the required image to run the application , it will fetch that image from docker registry and if found , it runs the docker container.

2:35

## 9. Introducing Containers 67. What is Docker



Docker is a container runtime environment for developing , shipping and running applications.

Docker hub is one of the official docker registry.

0:52

## 9. Introducing Containers 68. Hands on Docker Containers



Provisioning commands , which helps us to install docker engine on Linux environment

```
1 | sudo apt-get update
2 | sudo apt-get install \
3 | ca-certificates \
4 | curl \
5 | gnupg \
6 | lsb-release -y
7 | curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
   /usr/share/keyrings/docker-archive-keyring.gpg
```

```
8 | echo \
9 | "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-
archive-keyring.gpg] https://download.docker.com/linux/ubuntu \
10 | $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list >
/dev/null
11 | sudo apt-get update
12 | sudo apt-get install docker-ce docker-ce-cli containerd.io -y
```

1:38

## 9. Introducing Containers 68. Hands on Docker Containers



Here , is the documentation for installing docker daemon on Linux

<https://docs.docker.com/engine/install/ubuntu/>

3:41

## 9. Introducing Containers 68. Hands on Docker Containers



docker images => Find all the images on the local machine

docker ps => Find all the processes which are running on the server we are on

docker ps -a => Find all the processes which are in any state on the server we are on

docker rm <container-name> => Remove <container-name> container

docker rmi <image-id> => Remove <image-id> image

5:07

## 9. Introducing Containers 68. Hands on Docker Containers



When an application is created and running inside a container , we wouldn't know , what is happening inside the container from the host server , we can't interact with the container from the host server , so , if we want to interact with the container ports , we need to map those container ports with our host ports , so , we can access the container service through these host ports.

`docker run --name web01 -d -p 9080:80 nginx`

We example , we are running nginx image as web01 container , so , we are applying port mapping for the host's 9080 to the container's port 80 , so , whatever request that we pass to the port 9080 on the host will also trigger the request to the port 80 in the container.

If we click `docker ps` , the port will be looking something like this

`0.0.0.0:9080->80/tcp, ::: 9080-> 80/tcp`

5:50

## 9. Introducing Containers 68. Hands on Docker Containers



If we wanted to know the IP-Address of the container , we need to use the following command to find that ,

```
docker inspect <container-name>
```

Under "IPAddress" we will find the IP address of the container.

Inside the container , our IPAddress is "curl http://172.17.0.2:80"

Where as , if we wanted to access the page of nginx from our host machine , we need to check our host IP address , that is here "192.168.0.127" , we can check that using `ip addr show`

So, we can access that nginx page which is present at `172.17.0.2:80` in container from host machine using `192.168.0.127:9080` , where 9080 is the host port number which is mapped to port 80 of container using port mapping.

5:09

## 9. Introducing Containers 69. Vprofile Project on Containers



Vagrant is used to launch multiple VM's dynamically and where as in this video , we are having example for docker-compose , which have the concept of ports , volumes and launching a MYSQL service , memcached and rabbitmq , tomcat , nginx containers through docker-compose.

7:37

## 9. Introducing Containers 69. Vprofile Project on Containers



If we want to launch all the services that are present under the docker-compose.yml , we need to use the command `docker compose up -d`

By being in a place , where docker-compose.yml file present.

0:10

## 9. Introducing Containers 70. Microservices



### Difference between monolithic and microservices applications:

Consider an example where an Java application have User Interface , Chat feature , Posts feature and Notification reminder which all are coded using Java , this Java application is maintained under a single tomcat server , and if we want to make a change in any of this feature , we must stop the tomcat server and change the code , and there will be chances of disturbing the app code of a service while upgrading a service. This type of architecture is **Monolithic**.

Where as coming to microservices , we will be having multiple applications for multiple services , and those services will interact with one other using the API Gateway , where making a change in a service doesn't impact code or performance of another service.

These services interact one to other through API Gateway and this architecture is called **Microservices**.

4:23

## 9. Introducing Containers 70. Microservices



The main backdrop of monolithic structure is change in one service of the application can impact other service and all these services run on a single Linux server , so until who team are ready with the changes , we can't proceed forward and test.

But it will be efficient , if we have a dedicated Linux server for each and every service of the application , so that , we can maintain our own choice of language , customization , and achieve modularity. And there will be less chance of work lock , because , each service have it's own environment in this microservice architecture.

Microservice architecture , is the best solution rather than the monolithic architecture , but the issue comes with the multi-server maintenance cost. So instead of maintaining multiple Linux servers , but to achieve the modularity , we use the concept of containers.

5:14

## 9. Introducing Containers 70. Microservices



To save the infrastructure cost , we use the concept of docker instead of having dedicated Linux servers for each service. We will be having docker on the local hardware which is having it's own OS , either windows or Linux , on top of it docker container environment will be created , where for each an every application , we will create an image and when we run that image , we will be creating a dedicated runtime container which have it's own environment for that application.

5:55

## 9. Introducing Containers 70. Microservices



Every feature , will be running in different containers.

7:33

## 9. Introducing Containers 70. Microservices



Microservices are an architectural and organizational approach to software development where software is composed of small independent services that communicate over well-defined APIs. These services are owned by small, self-contained teams.

Microservices architectures make applications easier to scale and faster to develop, enabling innovation and accelerating time-to-market for new features.

```

1  version: '3.8'
2  services:
3    vprodb:
4      image: vprocontainers/vprofiledb
5      ports:
6        - "3306:3306"
7      volumes:
8        - vprodbdata:/var/lib/mysql
9      environment:
10        - MYSQL_ROOT_PASSWORD=vprodbpass
11
12    vprocache01:
13      image: memcached
14      ports:
15        - "11211:11211"
16
17    vpromq01:
18      image: rabbitmq
19      ports:
20        - "15672:15672"
21      environment:
22        - RABBITMQ_DEFAULT_USER=guest
23        - RABBITMQ_DEFAULT_PASS=guest
24
25    vproapp:
26      image: vprocontainers/vprofileapp
27      ports:
28        - "8080:8080"
29      volumes:
30        - vproappdata:/usr/local/tomcat/webapps
31
32    vproweb:
33      image: vprocontainers/vprofileweb
34      ports:
35        - "80:80"
36      volumes:
37        vprodbdata: {}
38        vproappdata: {}

```

This is the sample docker-compose file , where we will be having multiple services triggered by docker-compose with the help of images under that we are having mentions for image , ports , environment variables declaration and volumes.

\* We no need to have overhead resources ahead for the worst case of scaling up. We can use these cloud resources as per how much it is needed and scale up and down , these cloud providers own their hardware and provide to us as pay as you go subscription , this prevents us to own our own hardware cost.

3) Cost savings

\* If we are able to scale up and scale down the resources as per requirement , we will definitely save the cost of the cloud.

4) Deploy globally in minutes

4:27

## 11. AWS Part -1 94. What is Cloud Computing



### **Infrastructure as a Service (IaaS)**

IaaS contains the basic building blocks for cloud IT. It typically provides access to networking features, computers (virtual or on dedicated hardware), and data storage space. IaaS gives you the highest level of flexibility and management control over your IT resources. It is most similar to the existing IT resources with which many IT departments and developers are familiar.

### **Platform as a Service (PaaS)**

PaaS removes the need for you to manage underlying infrastructure (usually hardware and operating systems), and allows you to focus on the deployment and management of your applications. This helps you be more efficient as you don't need to worry about resource procurement, capacity planning, software maintenance, patching, or any of the other undifferentiated heavy lifting involved in running your application.

4:29

## 11. AWS Part -1 94. What is Cloud Computing



### **Software as a Service (SaaS)**

SaaS provides you with a complete product that is run and managed by the service provider. In most cases, people referring to SaaS are referring to end-user applications (such as web-based email). With a SaaS offering, you don't have to think about how the service is maintained or how the underlying infrastructure is managed. You only need to think about how you will use that particular software.

4:51

## 11. AWS Part -1 94. What is Cloud Computing



**Infrastructure as a Service:** We can configure our infrastructure , we can change the network ports or security configurations etc , we will have control our infrastructure configuration.

**Platform as a Service:** We no need to manage the infrastructure , the infra will auto scale up & down , we can concentrate mostly on our application.

**Software as a Service:** We will be given an end-to-end application , which we requested from the cloud provider , we can use the product , but not configure or manage the actual product , everything will be managed by the cloud provider , we only use it's services.

4:03

## 11. AWS Part -1 96. Ec2 Introduction



### Tags:

We will be having Tags for all the resources in AWS , which will be in-format of the key-value pairs , where key can be as name of the EC2 instance , region of it's presence and corresponding values are it's selected or chosen values. Tags make it easier to manage , search for , and filter resources.

### Security Group:

Security Group acts as a virtual firewall that controls the traffic for one or more instances. This will control the inflow and the outflow in cloud resources , which enhances the security.

To login to the EC2 instance , we need key pair , so that we can login to the EC2 instance remotely.

*Amazon EC2 uses .ssh level login validation, it uses public-key cryptography to encrypt and decrypt login information.*

0:56

## 11. AWS Part -1 97. Ec2 Quick Start



EC2 instance is typically a virtual machine , in free tier , it is preferable to select North Virginia of US as this is the AWS region which offers free services in AWS free tier.

To create an EC2 instance , select 'Services' menu of the Dashboard , under which , try to find the EC2 (Virtual Servers in the Cloud) option to start creating the EC2 instance.

Under 'All Services' of 'Services' on Dashboard , we can find EC2 instance option , else we can find EC2 option on the global search which is present on the dashboard.

If we click EC2 option then we will be directed to EC2 dashboard , where we will have the information related to "number of Amazon EC2 Resources that we are using in US East (N. Virginia) Region" , and following resources are the example ,

=> Instances , Dedicated Hosts , Elastic IPs , Instances , Key Pairs , Load Balancers , Placement Groups , Security Groups , Snapshots , Volumes

Under here , we will also be shown the availability zones under the region that we selected.

1:39

## 11. AWS Part -1 97. Ec2 Quick Start



We will be having at least 2 zones under a region.

We can launch instance from the EC2 Dashboard. Or we can create an instance by going to the instance option which is present under EC2. When we click launch instance , we will be redirected to a page , where we give,

Name of the Instance

Tag of the Instance

Application and OS Images (AMI Image)

=> *An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance.*

**Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud.**

3:04

## 11. AWS Part -1 97. Ec2 Quick Start



Name of the Tag is the Key and Actual name of that will be the value

**Key : Value => Name : web01**

**Key : Value => Project : alpha**

Name and Project are the tags , these are used to filter the EC2 instances.

Amazon Linux , Ubuntu , Windows , Red Hat , SUSE Linux are some of the AMI's

5:13

## 11. AWS Part -1 97. Ec2 Quick Start



We will be having two types of AMI's ,

1) AWS Marketplace AMIs => ( AWS & Trusted third-party AMI's , mostly secured and verified)

2) Community AMIs => ( Mostly these are open source , any one can publish their own image in here , these images are unverified and can be security sensitive. )

6:14

## 11. AWS Part -1 97. Ec2 Quick Start



We are selecting centos 9 stream image for our EC2 instance testing , where we are selecting the instance type as t2.micro which is free tier eligible , by default t2.small will be selected , the instance type depends on , the amount of CPU and amount of memory and type of the storage that is allocated to the EC2 instance.

6:50

## 11. AWS Part -1 97. Ec2 Quick Start



For compute intensive workloads , like machine learning tasks , we have to use the 'c series' in instance type , these can do compute intensive works.

7:29

## 11. AWS Part -1 97. Ec2 Quick Start



Here , we are having multiple types of instance types , with details of it

<https://aws.amazon.com/ec2/instance-types/>

8:13

## 11. AWS Part -1 97. Ec2 Quick Start



We need to create a key-pair for an EC2 instance , this key-pair will help us to login to the EC2 instance remotely , this acts as a key to login to the instance , which is a level of security. This is similar to generating a ssh key , and using that ssh public key to login to the respective server. After creating the EC2 instance we need to store the private key in secure and accessible location on our computer. We need that to connect our EC2 instance later.

We will be having two types of Key-Pair types

- 1) RSA (RSA encrypted private and public key pair)
- 2) ED25519 (ED25519 encrypted private and public key pair (Not supported for windows instances))

We will be having two types of private key file formats

- 1) .pem (For use with OpenSSH)
- 2) .ppk (For use with PuTTY)

8:29

## 11. AWS Part -1 97. Ec2 Quick Start



Down in the configuration of EC2 instance , we will be having **Network Settings** , where we will have details like **Network name** , **Subnet** (If we kept it as 'No Preference' then , default subnet will be selected in any availability zone , depends on which region our instance is launching) , **Auto assign public IP** (We will be allocated with an IP address for this instance , if this option is enabled , this is enabled by default) , We will be having options to modify our **Security groups** , where we can select , what requests we can allow through our EC2 instance and to what IP's we can allow the EC2 instance to generate response to.

By default , EC2 instance Allow SSH traffic from Anywhere , we are modifying that to Allow only from My IP , to not to allow un-necessary and random requests to the instance.

9:43

## 11. AWS Part -1 97. Ec2 Quick Start



We can configure the storage of our EC2 instance , where we can choose , how much storage we need for our instance as Hard disk.

Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage

10:06

## 11. AWS Part -1 97. Ec2 Quick Start



Under **Advanced details** , we will be having a text box as **User data** , under which you can mention all your shell commands , which you want to execute immediately after launching the EC2 instance , these commands that we give can be some software's that we wanted to install on top of the VM that we create , it can also be tools that we required.

This are some commands that we are giving as user data , that we wanted to execute immediately , when we Launch an instance

```
1 | #!/bin/bash
2 | sudo yum install httpd -y
3 | sudo systemctl start httpd
4 | sudo systemctl enable httpd
5 | mkdir /tmp/test1
```

12:01

## 11. AWS Part -1 97. Ec2 Quick Start



We can check the available EC2 instances on the Instances tab under Instances section , where we will be having details such as **Name of the instance** , **Instance ID** ,

Instance state , Instance type , Status Check , Alarm Status , Availability Zone , Public IPv4 address , Private IPv4 Address and so on....

We will be passing **two stages of health checks** on EC2 instances , one on hardware and another on the launching script on the instance.

**Public IPv4 address** is used to connect to the instance remotely from anywhere.

**Private IPv4 address** is used to connect to the instance under the subnet.

12:49

## 11. AWS Part -1 97. Ec2 Quick Start



### How to connect to EC2 instance?

1. Open an SSH client.
2. Locate your private key file. The key used to launch this instance is web-dev-key.pem
3. Run this command, if necessary, to ensure your key is not publicly viewable.

```
chmod 400 web-dev-key.pem
```

4. Connect to your instance using its Public DNS:

```
ec2-3-87-210-157.compute-1.amazonaws.com
```

Public DNS can be found on the instance home page

```
ssh -i "web-dev-key.pem" ec2-user@ec2-3-87-210-157.compute-1.amazonaws.com
```

15:22

## 11. AWS Part -1 97. Ec2 Quick Start



After logging in to the EC2 instance through ssh from our local system , we can check the status of the httpd , which we installed in EC2 instance through user-data while creating EC2 instance.

```
imran@LAPTOP-2J0OK66A MINGW64 ~/Downloads
```

```
$ ssh -i "web-dev-key.pem" ec2-user@ec2-3-87-210-157.compute-1.amazonaws.com
```

```
Last login: Sat Jun 24 00:38:08 2023 from 106.212.235.125
```

```
[ec2-user@ip-172-31-94-50 ~]$ sudo -i
```

15:24

## 11. AWS Part -1 97. Ec2 Quick Start



```
1 | [root@ip-172-31-94-50 ~]# systemctl status httpd
2 |   1 httpd.service - The Apache HTTP Server
3 |     Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; preset:
4 |       disabled)
5 |       Active: active (running) since Sat 2023-06-24 00:38:50 UTC; 1min 19s ago
6 |         Docs: man:httpd.service (8)
```

```
6 | Main PID: 13174 (httpd)
7 | Status: "Total requests: 0; Idle/Busy workers 100/0; Requests/sec: 0; Bytes
   | served/sec: 0 B/sec"
8 | Tasks: 213 (limit: 4488)
9 | Memory: 23. OM
10 | CPU: 100ms
11 | CGroup: /system.slice/httpd.service
12 | -13174 /usr/sbin/httpd -DFOREGROUND
13 | -13175 /usr/sbin/httpd -DFOREGROUND
14 | -13176 /usr/sbin/httpd -DFOREGROUND
15 | -13177 /usr/sbin/httpd -DFOREGROUND
16 | 13178 /usr/sbin/httpd -DFOREGROUND
17 |
18 | Jun 24 00:38:50 ip-172-31-94-50. ec2. internal systemd[1]: Starting The Apache
   | HTTP Server ...
19 | Jun 24 00:38:50 ip-172-31-94-50. ec2. internal systemd[1]: Started The Apache
   | HTTP Server.
20 | Jun 24 00:38:50 ip-172-31-94-50. ec2. internal httpd[13174]: Server configured,
   | listening on: port 80
```

15:51

## 11. AWS Part -1 97. Ec2 Quick Start



The command '`ss -tunlp | grep 80`' is used to display information about network sockets that use TCP protocol and listen on port 80. The '`ss`' command stands for '`socket statistics`' and it can show various details about network connections. The '`-t`' option filters the output by TCP sockets, the '`-u`' option filters by UDP sockets, the '`-n`' option shows numerical addresses instead of resolving them, the '`-l`' option shows listening sockets, and the '`-p`' option shows process information. The '`grep 80`' part of the command searches for the string '80' in the output, which corresponds to the port number used by HTTP protocol. This command can be useful for troubleshooting network issues or monitoring web server activity.

This command helps us to find , are we having any running ports? , so that we can connect to the server , if it's in running state.

16:30

## 11. AWS Part -1 97. Ec2 Quick Start



So , as we see , a port running on 80 , we can access that using the browser by , `https://<public-ip-address>:<port-number>` , but we can't access that , because of firewall. We need to modify the security group , such that we can access the website on port 80 , for now , we are only having access for port 22 , which is allowed from the time of creating the instance. Port 22 allows the ssh connections through the EC2 instance.

17:33

## 11. AWS Part -1 97. Ec2 Quick Start



## Managing Inbound Rules:

To modify the security group , go to the instance page , select the instance , and under that , select the 'security group' tab , modify the inbound rules , under which add a new rule of type 'Custom TCP' on Port '80' , and the Source of accessing that could be 'My IP' (If only I want to access that page) , Anywhere IPv4 and IPv6 (If we want to launch the site globally , all the people can listen this particular port from their IP addresses).

Finally Save Rules , after completing the edit , your changes will be reflected on AWS.

18:07

11. AWS Part -1 97. Ec2 Quick Start



We will be having an option to **set the state of the instance** , they are

- 1 | Stop Instance
- 2 | Start Instance
- 3 | Reboot Instance
- 4 | Hibernate Instance
- 5 | Terminate Instance ( Remove the Instance , Clear Volume allocated to it)

0:48

11. AWS Part -1 98. More In Ec2 Part1



## EC2 instance creation:

- 1) Requirement Gathering
- 2) Key pairs
- 3) Security Group
- 4) Instance Launch

2:14

11. AWS Part -1 98. More In Ec2 Part1



**Requirement Gathering** (Following are the one , which are minimum to gather information about to create a EC2 instance)

- 1) **OS** (Operating system on which we want to host our application)
  - a. Centos
- 2) **Size** => Ram, CPU, Network etc. ( How much computational ability we need to host our app )
  - a. Min
- 3) **Storage size** ( Storage and Hard disk we need to run our application )
  - a. 10 gigs

- 4) **Project** ( Basic details on what software we need and what is the end result that we want to generate)
- 5) **Services/Apps Running** ( We need to create proper inbound / outbound rules , So , request and response within the application will be uninterrupted )
  - a. SSH, Http, MySQL etc.
- 6) **Environment** (Dev, QA, Staging, Prod) ( Based on Environment , we need to configure permissions for the users , how is gonna use these instances )
- 7) **Login User/ Owner** ( Used to restrict the access for users , based on his role in the company)

5:19

## 11. AWS Part -1 98. More In Ec2 Part1



When we are trying to delete a security group which is attached to other resources , then we will see a warning of there are dependent resources , so we need to delete them first , and then delete the security group.

6:25

## 11. AWS Part -1 98. More In Ec2 Part1



**Security Group** is kind of firewall , which control the request and response of the EC2 instance. If we want to allow all the connections without any condition , just keep Inbound rules open for all the IPv4 and IPv6 IP addresses.

**Outbound rules** is mainly used to maintain the connection between the EC2 instance and the internet.

We can apply one security group to multiple instances.

7:19

## 11. AWS Part -1 98. More In Ec2 Part1



Security group acts as a virtual firewall that controls the traffic for one or more instances.

You can add rules to each security group that allow traffic to or from its associated instances

Security groups are "stateful".

8:18

## 11. AWS Part -1 98. More In Ec2 Part1



Firewall acts as a security guard , where it will have a register called as security group, in which he will have set of rules to allow someone from outside to the instance as Inbound Rules (Rules for getting In). He will have set of rules to allow someone to go outside call

outbound Rules (Rules for letting someone out from the instance). These rules will be declared under security group.

8:57

## 11. AWS Part -1 98. More In Ec2 Part1



If we want no one to connect to the instance then , we need to create an Inbound Rules of Type 'SSH' , Source 'My IP' , this results connections for the ssh , can be established through only MyIP not by any other IP.

11:47

## 11. AWS Part -1 98. More In Ec2 Part1



We can create multiple key-pairs for different , environments , and we can use these key-pairs while create and launching instances , so all the instances that we launch will be having unique environments with unique key-pair. key-pairs are noting but private-keys that are used to login to the instance. Public key will be fed into the instance , while we create , and the private key , we downloaded earlier , we can use those private keys and login to the instance.

12:26

## 11. AWS Part -1 98. More In Ec2 Part1



We can launch multiple instances at a time , while doing the configuration and launch instance. The tag which we give while creating an instance , helps us in filtering the instances globally.

If we wanted to connect to an instance , then click the instance and choose connect option , then we can connect to that instance through SSH Client.

13:54

## 11. AWS Part -1 98. More In Ec2 Part1



After creating an new instance , we can host a website using following command , this is tooplate website , which is this course specific.

```
1 | sudo apt update
2 | sudo apt install apache2 wget unzip -y
3 | wget https://www.tooplate.com/zip-templates/2128_tween_agency.zip
4 | unzip 2128_tween_agency.zip
5 | cp -r 2128_tween_agency/# /var/www/html/
6 | systemctl restart apache2
```

14:26

## 11. AWS Part -1 98. More In Ec2 Part1



We can check the code under,

```
root@ip-172-31-16-60: # ls /var/www/html/
```

We can check the status of the Apache server , using ,

```
1 | root@ip-172-31-16-60: # systemctl status apache2
2 |      apache2. service - The Apache HTTP Server
3 |      Loaded: loaded (/lib/systemd/system/apache2. service; enabled; vendor preset:
4 |      enabled)
5 |      Active: active (running) since Fri 2022-04-29 21:31:12 UTC; 22s ago
6 |      Apr 29 21:31:12 ip-172-31-16-60 systemd[1]: apache2. service: Succeeded.
7 |      Apr 29 21:31:12 ip-172-31-16-60 systemd[1]: Stopped The Apache HTTP Server.
8 |      Apr 29 21:31:12 ip-172-31-16-60 systemd[1]: Starting The Apache HTTP Server ...
9 |      Apr 29 21:31:12 ip-172-31-16-60 systemd[1]: Started The Apache HTTP Server.
```

15:21

## 11. AWS Part -1 98. More In Ec2 Part1



If we wanted to access the port 80 from our IP , we can't access directly using IP Address using port number using internet browser , we also needed to modify the Inbound rules and Add a rule of Type Custom TCP of Port 80 can accessible for My IP. So that we can access the website on port 80 with our IP address.

1:14

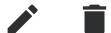
## 11. AWS Part -1 99. More in Ec2 Part2



For an Instance , Public IP address changes for every power down and power up , this is because , when we stop the instance that particular IP address is released and someone can use that IP address , where as Private IPv4 address remains same until we terminate that instance completely.

1:19

## 11. AWS Part -1 99. More in Ec2 Part2



If we don't want our Public IPv4 Address to be changed from each power down and up , we can request one IP address , which will be allocated to us , by using Elastic IP address , we can find this option under

```
Network & Security > Elastic IPs > Create
```

And after Creating , we can allocate this elastic IP address to an instance of our choice.

We can create a [Network Interface which is under Network & Security](#) , and assign that to instance of our choice.

We can create a [Volumes which is under Elastic Block Store](#) , and assign that to instance of our choice.

We can see all the resources that we created on EC2 Dashboard , Instances , Dedicated Hosts , Elastic IPs , Instances , Key Pairs , Load Balancers , Placement Groups , Security Groups , Snapshots , Volumes.

7:15

## 11. AWS Part -1 99. More in Ec2 Part2



We can also perform quick actions for instances , from the instance tab , quick action include

- 1 | Connect to the instance
- 2 | Manage Instance State
- 3 | Instance Settings
- 4 | Networking
- 5 | Security
- 6 | Image and Templates
- 7 | Monitor and Troubleshoot

We can also clone an instance with similar settings , we can clone the instance , but not the data in it.

9:14

## 11. AWS Part -1 99. More in Ec2 Part2



We must release the resources to the amazon pool , if we are not using them , because AWS is pay as you go basis , so even if we are having the resources , without using them , we will be costed for those. It's better to release them.

0:50

## 11. AWS Part -1 100. AWS CLI



We can work with AWS , through GUI and as well as CLI , we have been working with AWS through GUI , we can also work with CLI using AWS commands.

We have to install AWS CLI in windows using `choco install awscli` in windows PowerShell by running it as an administrator.

2:10

## 11. AWS Part -1 100. AWS CLI



If we want to use AWS CLI , we need an IAM User , so to create one , search for IAM on AWS , this IAM manage access to AWS resources. Then we will redirect to IAM Dashboard , IAM means Identity and Access Management. In here , we will create a User by using an Option called 'Add User'. Give the user a name , we need to set permissions for the user.

We typically , have three types of permission options ,

**Add user to group** ( Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function)

**Copy Permissions** ( Copy all group memberships, attached managed policies, and inline policies from an existing user )

**Attach Policies directly** ( Attach a managed policy directly to a user. As a best practice, we recommend attaching policies to a group instead. Then, add the user to the appropriate group )

2:33

## 11. AWS Part -1 100. AWS CLI



Attach Policies directly give privileges to user instead giving it to a group , the permission policies are directly assigned to the user rather than to the group then the user.

For our user , we are giving Administrator Access , so that , our user will have access to manage the AWS resources , but If we want our user to have only access to one resource , then we can give access for only that resource.

If we want to give access to a user rather than a group we need to use this Attach Policies and assign a permission policy for that.

3:21

## 11. AWS Part -1 100. AWS CLI



After creating a user , click the username , then we will be redirected to user page , where we have a column like security credentials , under which we will be having **Access Keys** , we need this **access keys to login through CLI** , so create one access key. After creating an access key , we will be able to get an access key and **secret access key** , **this behaves like username and password for the IAM user** . Do not share these access key and secret access key to anyone , because these will have admin privileges , so hackers can hijack our account or they can misuse them , we can download the access key and secret access key as a csv file.

5:43

## 11. AWS Part -1 100. AWS CLI



We need to configure these access keys on AWS CLI , use **aws configure** command to give the details to access the aws account through the CLI.

```
1 | $ aws configure
2 | AWS Access Key ID [None]: AKIAZRTSKWK7HCX4GPSV
3 | AWS Secret Access Key [None]: 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
4 | Default region name [None]: us-east-1
5 | Default output format [None]: json
```

The above values are demo values , we need to fill with ours

6:13

## 11. AWS Part -1 100. AWS CLI



AWS CLI will store the details that we have given in it's configuration files , in the following way

```
1 imran@LAPTOP-2J00K66A MINGW64
2 $ ls ~/. aws/
3 config credentials
4
5 imran@LAPTOP-2J00K66A MINGW64
6 $ cat ~/. aws/config
7 [default]
8 region = us-east-1
9 output = json
10
11 imran@LAPTOP-2J00K66A MINGW64
12 $ cat /. aws/credentials
13 [default]
14 aws_access_key_id = AKIAZRTSKWK7HCX4GPSV
15 aws_secret_access_key = mbMcImE9myp+c0AXp1GrUOXXII1aEMaWHCKvI6W9o
```

6:49

## 11. AWS Part -1 100. AWS CLI



We need to delete the access keys instead of deleting user , if we think our credentials are compromised.

To get basic details like what is the user id and account id of the AWS account we having now , use `aws sts get-caller-identity` command

To know , what all instances that we are having in our AWS account , use `aws ec2 describe-instances` command.

13:28

## 11. AWS Part -1 100. AWS CLI



<https://awscli.amazonaws.com/v2/documentation/api/latest/index.html>

Refer this website , to find commands that we can use in AWS CLI , to manage the resources in AWS or you can use ChatGPT for generating commands , but you need some basic idea to use them. And you know , we can also use GUI to manage AWS.

0:11

## 11. AWS Part -1 101. EBS



**Elastic Block Storage** have two types , one is Amazon Elastic Block Store (EBS) that is also called as virtual hard disk which we use for our EC2 instance. And another is the snapshot , which is the backup for the Elastic Block Store.

- \* Block based storage
- \* Runs ec2 OS, store data from db, file data, etc
- \* Placed in specific AZ. Automatically replicated within the AZ to protect from failure.
- \* Snapshot is backup of a volume

2:59

## 11. AWS Part -1 101. EBS



### EBS Volume Types

**General Purpose (SSD)** (Most General Work uses this storage type)

**Provisioned IOPS** (If we work on Large Databases , where high I/O per second is the highest priority , we use this)

**Throughput Optimized HD** ( If we want process huge data every second , we use this , For Big Data Applications and Data Warehouses , we use this )

**Cold HDD** ( Preferable for Storage of Files , typically File Servers )

**Magnetic** ( Cheapest of all , mostly used for backup and archiving purpose )

3:00

## 11. AWS Part -1 101. EBS



**General Purpose SSD** volumes (gp2 and gp3) have a baseline throughput of 128 MiB/s per TiB, which can burst up to 250 MiB/s or 1,000 MiB/s depending on the volume size and IOPS1. **Provisioned IOPS SSD** volumes (io1 and io2) have a maximum throughput of 1,000 MiB/s per volume, regardless of the volume size1. **Throughput Optimized HDD** volumes (st1) have a maximum throughput of 500 MiB/s per volume, and are suitable for large, sequential workloads1. **Cold HDD** volumes (sc1) have a maximum throughput of 250 MiB/s per volume, and are suitable for infrequent, sequential access1.

You can use **Amazon CloudWatch** metrics to monitor the throughput of your EBS volumes and calculate the average throughput over a period of time. You can also use the EBS performance test tool to benchmark the throughput of your volumes under different workloads.

4:47

## 11. AWS Part -1 101. EBS



For testing the EBS , we need a centos installed EC2 instance ,

We will run the following script in that VM or we will give that in the user data ,

```
1 | yum install httpd wget unzip -y
```

```
2 | systemctl start httpd
3 | systemctl enable httpd
4 | cd /tmp
5 | wget https://www.tooplate.com/zip-templates/2128_tween_agency.zip
6 | unzip -o 2128_tween_agency.zip
7 | cp -r 2128_tween_agency/ "/var/www/html/"
8 | systemctl restart httpd
```

While we create a EC2 instance , the default storage we use is gp2 storage , which is general purpose storage tier 2 , which is SSD type.

7:08

## 11. AWS Part -1 101. EBS



After running those commands , we will be able to see some folders under `/var/www/html/` , which contains the code of the webpage , and we will be also having a folder `/var/www/html/images/` where we have images which are required to represent the website.

Both our instance and volume will be in the same Availability zone , we can't use volume which is in other availability zone to our instance which is different to the zone of the volume.

8:49

## 11. AWS Part -1 101. EBS



We will have max 30GB of storage in free tier of EBS in AWS.

14:02

## 11. AWS Part -1 101. EBS



For the volumes that we created , we can create a partition as following ,

**List the disk information:** `fdisk -l`

```
1 | Disk /dev/xvda: 8589 MB, 8589934592 bytes, 16777216 sectors
2 | Units = sectors of 1 * 512 = 512 bytes
3 | Sector size (logical/physical): 512 bytes / 512 bytes
4 | I/O size (minimum/optimal) : 512 bytes / 512 bytes
5 | Disk label type: dos
6 | Disk identifier: 0x0000b723c
```

As we see , this is the root directory for the volume that we have ,

We can double confirm that using `df -h`

Now if we want to make partition on that , we need to use `fdisk /dev/xvda` , fill the necessary details further , the volume will partition finally.

15:08

## 11. AWS Part -1 101. EBS



## How to format volume?

Formatting a volume in Linux means creating a file system on a disk or partition that can store and organize data. The mkfs command is a tool that allows you to format a volume with different file system types, such as ext4 and xfs. Ext4 is the default and most widely used file system in Linux, while xfs is a high-performance file system that supports large files and parallel I/O. Both file systems have their own advantages and disadvantages, depending on the use case and workload.

15:10

## 11. AWS Part -1 101. EBS



To format a volume with ext4 or xfs, you need to specify the file system type and the target device. For example, to format the /dev/sdb1 partition with ext4, you can use:

```
sudo mkfs -t ext4 /dev/sdb1
```

or

```
sudo mkfs.ext4 /dev/sdb1
```

To format the same partition with xfs, you can use:

```
sudo mkfs -t xfs /dev/sdb1
```

or

```
sudo mkfs.xfs /dev/sdb1
```

Note that formatting a volume will erase all the data on it, so make sure you have a backup before proceeding. You can also check for bad blocks on the device before formatting, but this may take a long time. To do so, you can add the -c option to the mkfs command.

16:42

## 11. AWS Part -1 101. EBS



If we want to **mount the volume** on a directory , and make that directory as a root folder for that partition.

```
mount /dev/xvdf1 /var/www/html/images/
```

/dev/xvdf1 volume is mounted to /var/www/html/images

If we want to unmount the volume , we need to use the

```
umount /var/www/html/images
```

This mount is temporary , when we restart the instance this mount point will be gone , so if we want to make this mount point permanent , we need to do the following ,

Open `vi /etc/fstab` , then enter following line to the last of the file ,

/dev/xvdf1	/var/www/html/images	ext4	defaults	0	0
------------	----------------------	------	----------	---	---

/dev/xvdf1 => Volume

/var/www/html/images => Mount point

ext4 => Volume format

After editing fstab file , we can refresh the mount points which are present in fstab file using `mount -a`

2:15

## 11. AWS Part -1 102. EBS Snapshots



If we are deleting any folder , and we get any error while deleting that , it is due to that particular folder , may be used by some other process , we can check that using

`lsof /var/www/html/images`

Then we would get , process id's which are using them , we can verify those are necessary or not , and we can kill the process and delete the folder that we wanted to delete.

We need to delete the volumes which are not in use , else we will end up paying for those.

We can create new volumes and attach those to our EC2 instances. And after attaching , we can check that using `fdisk -l` command.

7:10

## 11. AWS Part -1 102. EBS Snapshots



We can use these partitions for allocating the storage for dedicated applications , for example , if we wanted to install MySQL , we know that it's files will be present on /var/lib/mysql , so now , we will create a partition in volume , where that partition is mounted to /var/lib/mysql , now , what is the benefit of this is , we will have dedicated storage for the application. We need to do this process even before installing the MySQL , so , when it is ready , it can use it's dedicated partition for it's operation.

13:37

## 11. AWS Part -1 102. EBS Snapshots



The Concept of **SNAPSHOT** is to having a backup for the volume , if we wanted to create a SNAPSHOT , go to volumes section under Elastic Block Store , where select the volume for which we want to create Snapshot , and under actions click Create Snapshot , and create one , which will be visible under snapshots section under Elastic Block Store. This snapshot will have all the files of the volume until the point of the creation of this snapshot.

Later , for demo purpose , delete the /var/lib/mysql contents , so to get the contents back , we gonna use snapshot for that.

10:22

## 11. AWS Part -1 102. EBS Snapshots



## Retrieve data through Snapshot:

First stop the mariadb service , to modify it's root folders `systemctl stop mariadb` , later unmount the root folder for the sql `umount /var/lib/mysql` and now , we want to detach volume from the instance , so under volumes section , select the volume that we wanted to detach and detach that volume using Actions tab , and then we can use the snapshot to create volume , now go to snapshot and then using create volume which is under actions tab , we can create a volume which we will attach it later to the instance. We can also modify the size , availability zone of the volume using this snapshot. So snapshots are used for changing the availability zones , size and even for backup the volume. After creating volume through snapshot , we can attach volume to an instance.

14:15

11. AWS Part -1 102. EBS Snapshots



We can copy the snapshot , from one region to other region , sometimes , we want a file structure to be present across regions , and we can also create AMI's using Snapshots. We can even make the snapshot public and we can share it through their account number , this action is present under Actions tab of Snapshot , as 'Modify Permissions'

0:40

11. AWS Part -1 103. ELB Introduction



A load balancer in AWS terms is a service that automatically distributes incoming application traffic across multiple targets, such as EC2 instances, containers, and IP addresses, in one or more Availability Zones. It helps to improve the scalability, availability, and performance of your application. There are different types of load balancers in AWS, such as Application Load Balancer, Network Load Balancer, Gateway Load Balancer, and Classic Load Balancer, each with its own features and use cases.

0:41

11. AWS Part -1 103. ELB Introduction



Elastic Load Balancer (ELB) is used to distribute incoming application traffic across multiple targets, such as EC2 instances, containers, and IP addresses, in one or more Availability Zones. ELB helps to improve the scalability, availability, and performance of your application by automatically adjusting the load balancer capacity according to the changes in incoming traffic2. ELB also supports SSL/TLS termination, integrated certificate management, health checks, and real-time monitoring2. You can choose from different types of ELB, such as Application Load Balancer, Network Load Balancer, Gateway Load Balancer, and Classic Load Balancer, depending on your application needs.

In load balancer , we will be having two ports , one is frontend port and other is the backend port.

**Frontend port:** Listens from the user requests , on this port. These are also called as Listeners , as these are the listener ports.

**Backend Ports:** Services running on the OS listening on this port.

Basically , on Frontend we will get the user requests , where as on the backend , the software application will communicate. Based on the user requests , load balancer decides number of backend resources we need. And the application , will generate response to the user and pass them through these backend ports.

**Elastic Load Balancing** distributes incoming application or network traffic across multiple targets, such as Amazon EC2 instances, containers, and IP addresses, in multiple Availability Zones. Main operation is to split the load based on the amount of requests received by the user among the AWS resources.

***Elastic Load Balancing supports three types of load balancers:***

- \* Application Load Balancer ( Mostly used to manage web traffic )
- \* Network Load Balancer ( High Performance , Costly )
- \* Classic Load Balancer ( General Load Balancer , Mostly used , Cheapest among others )

### Classic Load Balancer

Basically accepts the incoming traffic and distribute to backend servers.

The Classic Load Balancer that routes traffic based on either application or network level information.

Works at the network layer in OSI

Classic Load Balancer is ideal for simple load balancing of traffic across multiple EC2 instances

### Application Load Balancer:

Works at application layer in OSI model.

It basically transfer the user requests to the cluster of resources which is required , for example , if we search for www.google.com , it redirects to one cluster of EC2 instance , and if we search for www.google.com/images or www.udemy.com it redirects to other cluster of EC2 instance. This is basically network path redirection between the AWS resources.

Application Load Balancer that routes traffic based on advanced application level information that includes the content of the request.

4:03

## 11. AWS Part -1 103. ELB Introduction



### Network Load Balancer:

Comes at fourth layer of OSI Model.

If we need a load balancer that can handle millions of requests from the user , then we need to choose network load balancer.

It can handle millions of requests per second.

If we need a static IP then we need to use this , both classic load balancer and application load balancer will have dynamic IP's for their endpoints , where as Network load balancer has a static IP , so , if we need an application which need a static IP allocated load balancer , we need to use this , and this load balancer , is far better in performance than the other two.

5:23

## 11. AWS Part -1 103. ELB Introduction



### Gateway load balancers:

Helps to deploy , scale and manage virtual appliances like firewall. We can also use these gateway load balancers for intrusion detection and prevention systems, and deep packet inspection systems. Every load balancers including this distribute the traffic among the across multiple targets in one or more Availability Zones.

2:02

## 11. AWS Part -1 104. ELB Hands On



We use the following script to launch , EC2 instance on , to test the ELB Storage

```
1  #!/bin/bash
2
3  # Variable Declaration
4  #PACKAGE="httpd wget unzip"
5  #SVC="httpd"
6  URL='https://www.tooplate.com/zip-templates/2098_health.zip'
7  ART_NAME='2098_health'
8  TEMPDIR="/tmp/webfiles"
```

```
9  
10 yum --help &> /dev/null
```

2:03

## 11. AWS Part -1 104. ELB Hands On



```
1 if [ $? -eq 0 ]  
2 then  
3     # Set Variables for CentOS  
4     PACKAGE="httpd wget unzip"  
5     SVC="httpd"  
6  
7     echo "Running Setup on CentOS"  
8     # Installing Dependencies  
9     echo "Installing packages."  
10    sudo yum install $PACKAGE -y > /dev/null  
11    echo  
12  
13    # Start & Enable Service  
14    echo "Start & Enable HTTPD Service"  
15    sudo systemctl start $SVC  
16    sudo systemctl enable $SVC  
17    echo  
18  
19    # Creating Temp Directory  
20    echo "Starting Artifact Deployment"  
21    mkdir -p $TEMPDIR  
22    cd $TEMPDIR  
23    echo  
24  
25    wget $URL > /dev/null  
26    unzip $ART_NAME.zip > /dev/null  
27    sudo cp -r $ART_NAME/* /var/www/html/  
28    echo  
29  
30    # Bounce Service  
31    echo "Restarting HTTPD service"  
32    systemctl restart $SVC  
33    echo  
34  
35    # Clean Up  
36    echo "Removing Temporary Files"  
37    rm -rf $TEMPDIR  
38    echo  
39  
40    sudo systemctl status $SVC  
41    ls /var/www/html/
```

2:05

## 11. AWS Part -1 104. ELB Hands On



```

1  else
2      # Set Variables for Ubuntu
3  PACKAGE="apache2 wget unzip"
4  SVC="apache2"
5
6  echo "Running Setup on CentOS"
7  # Installing Dependencies
8  echo "Installing packages."
9  sudo apt update
10 sudo apt install $PACKAGE -y > /dev/null
11 echo
12
13 # Start & Enable Service
14 echo "Start & Enable HTTPD Service"
15 sudo systemctl start $SVC
16 sudo systemctl enable $SVC
17 echo
18
19 # Creating Temp Directory
20 echo "Starting Artifact Deployment"
21 mkdir -p $TEMPDIR
22 cd $TEMPDIR
23 echo
24
25 wget $URL > /dev/null
26 unzip $ART_NAME.zip > /dev/null
27 sudo cp -r $ART_NAME/* /var/www/html/
28 echo
29
30 # Bounce Service
31 echo "Restarting HTTPD service"
32 systemctl restart $SVC
33 echo
34
35 # Clean Up
36 echo "Removing Temporary Files"
37 rm -rf $TEMPDIR
38 echo
39
40 sudo systemctl status $SVC
41 ls /var/www/html/
42
43 fi

```

2:11

**11. AWS Part -1 104. ELB Hands On**

Which will create a website , you can visit that website , by using <IP-address-of-  
EC2>:8080

If the website isn't working properly , login to the EC2 instance , check Apache and httpd services are running , if not install them and execute the script manually.

3:53

## 11. AWS Part -1 104. ELB Hands On



Now , if we wanted to make multiple instances of EC2 instance in which we hosted the application , we need to create an AMI based on the EC2 instance , we can do that on the EC2 instances page , *create image* based on instance option. And we can check the image which is created at AMI's section.

4:26

## 11. AWS Part -1 104. ELB Hands On



We can **COPY AMI** from one region to another region , **Edit Permissions for AMI** we can make the AMI public or keep it private , and even we can share that AMI personally to their amazon account id or to the shared organization.

5:12

## 11. AWS Part -1 104. ELB Hands On



We are also having an option to Build the Image automatically based on the instance using **EC2 Image Builder** , here , we will create a pipeline for automating the process of AMI creation.

6:51

## 11. AWS Part -1 104. ELB Hands On



We can also create a template for launching the instance . Every time , when we plan to create an instance , we need to fill all the fields , so instead of that , we can fill some of those fields and save that as a template. These are called as **Launch Templates**. When we want to launch a new instance , we can use **Launch Instance from Template** , and edit the remaining columns and launch a new instance , this increase reusability of templates and make our work faster.

7:55

## 11. AWS Part -1 104. ELB Hands On



And assume that we launch multiple instances with same template , if we want to interact with those two instances , we shouldn't do that manually , we will be connecting them with common endpoint that is the **Load Balancer** which redirects the requests among those instances.

We will be having concept called target group , where we do the health checks for all our instances , only send the request to only the resources which are healthy.

It will have option to select which load balancer , we want to choose (Classic , Network , Application ) , Target Group Name , Protocol (HTTPS/HTTP) , Port (Which port of IP address that we want to check , to know if the website is healthy or not (localhost:8080) , here port is 8080 , in our current case it is 80. We need to select Health Check Protocol (Probably HTTPS/HTTP) , and we need to choose the URL path , which is next to the IP Address and port number .

`<IP_Address>:<Port-Number>/<URL-Path>`

In Health Check Path , only <URL-Path> should be mentioned , at this particular URL , we do the health check , is the instance is working or not.

We also need to give the

**Healthy Threshold** (The number of consecutive health checks successes required before considering an unhealthy target healthy.)

**Unhealthy Threshold** (The number of consecutive health check failures required before considering a target unhealthy.)

**TimeOut** (The amount of time, in seconds, during which no response means a failed health check.)

**Interval** (The approximate amount of time between health checks of an individual target)

**Success Codes** (The HTTP codes to use when checking for a successful response from a target. You can specify multiple values (for example, "200,202") or a range of values (for example, "200-299").)

If the response from the instance is 200 then , we can assume that , that particular instance is healthy.

Create target , based on the inputs , that are provided.

### Load Balancers:

Now under Load Balancer section , select 'Create Load Balancer' , first we need to select , which type of load balancer , we wanted to create , Application Load Balancer or Network Load Balancer or Gateway Load Balancer , now we want to play with the http

traffic , check the health status of the website using target group , so select Application Load Balancer.

## We need to give basic details to the Load Balancers like

### Load Balancer name

### Scheme of Load Balancer

- 1 | => Internet Facing
- 2 | => Load Balancer can be reached through Internet
- 3 | => An internet-facing load balancer routes requests **from** clients over the internet to targets. Requires a **public** subnet. Here target **is** part of Load balancer , which check the health status of the resources.
- 4 | => Internal
- 5 | => An **internal** load balancer routes requests **from** clients to targets **using private** IP addresses.

12:40

11. AWS Part -1 104. ELB Hands On



**Load balancer** is a device that distributes incoming traffic among a group of servers or instances. There are **two types of load balancers: internet-facing and internal**.

An **internet-facing load balancer** has a *public IP address and can receive requests from clients over the internet*. It can route the requests to servers or instances that are either public or private. An internet-facing load balancer is useful for applications that need to serve external users, such as web servers or e-commerce sites.

An **internal load balancer** has a *private IP address and can only receive requests from clients within the same VPC network or from clients that are connected to the VPC network by using VPC Network Peering, Cloud VPN, or Cloud Interconnect*. It can route the requests to servers or instances that are also private. An internal load balancer is useful for applications that need to serve internal users, such as database servers or microservices.

12:41

11. AWS Part -1 104. ELB Hands On



The **main difference between internet-facing and internal load balancers** is the *scope of their accessibility*. Internet-facing load balancers can be accessed by anyone on the internet, while internal load balancers can only be accessed by authorized clients within or connected to the VPC network.

A **VPC network** is a virtual network that is *logically isolated from other networks in the cloud*. It allows you to launch and connect cloud resources, such as virtual machines, load balancers, VPNs, and more, in a secure and flexible way. Different cloud providers, such as Google Cloud, AWS, and DigitalOcean, offer different features and services for VPC networks.

The full form of VPC network is **Virtual Private Cloud** network. It is a type of network that is isolated from other networks in the cloud and can be customized according to the needs of the user.

12:51

## 11. AWS Part -1 104. ELB Hands On



What type of IP Address that we want to use for the *load balancer identification* IPv4 or IPv6.

We also need to select multiple availability zones for the load balancers , we need to do that under *Mappings* .

In Mappings we need to Select at least one Availability Zone and one subnet for each zone. We recommend selecting at least two Availability Zones.

The load balancer will route traffic only to targets in the selected Availability Zones. Zones that are not supported by the load balancer or VPC cannot be selected. Subnets can be added, but not removed, once a load balancer is created.

We need to select multiple Availability zones because to make our load balancer highly available.

13:52

## 11. AWS Part -1 104. ELB Hands On



We also need to have a security group for the load balancer , basic details like security group name , Inbound rules , outbound rules (mostly no need to change).

In Inbound rules , if we want everyone to access our port 80 with our IP address , such that this is a real time application , for which getting a huge load is possible , so we can balance that load with the load balancer. We need to set anyone could access port 80 from IPv4 and IPv6.

14:32

## 11. AWS Part -1 104. ELB Hands On



Under Listeners and Routing section of Load Balancer , we must redirect the requests that are receiving on port 80 using http protocol , should be redirected to the target group on port 80 for health checks. We will give the target group details under Listeners and routing section of Load balancers. And create the load balancer , at last.

17:32

## 11. AWS Part -1 104. ELB Hands On



If we are not able to access the website which is hosted with the domain name of the load balancer , but present on the individual instances which target groups monitor , then we must go to the target group and check are those individual instances are healthy or not? , if not , it is due to the security group rules of those individual instances.

So , add an inbound rule under those individual instances as , for port 80 allow all the requests which are coming from the security group of the load balancer. We can select security group of the load balancer under security rules of those individual instances.

So , now , whatever requests which are coming from the load balancers to the individual instances which are present under the target group , will be allowed by those instances. So , now , we can access the website using the domain name of the load balancer.

19:22

## 11. AWS Part -1 104. ELB Hands On



### How to do maintenance in load balancers?

Suppose , if we are doing any maintenance activity on the individual instances , first we need to come to the target group and **Deregister** those instance first by selecting them , then the load balancer first drain the requests from that instance and release that , and after the maintenance done , we need to **Re-register** that instance that we added earlier and de-registered for maintenance.

Only the healthy instances will receive requests and distribute the load among the target group , we can check status of any instance , if we feel the server is down , by coming to the Target group option under the Load Balancer section.

0:11

## 11. AWS Part -1 105. Cloudwatch Introduction



AWS CloudWatch is a monitoring and management service that helps you observe and optimize the performance and health of your AWS resources and applications. You can use CloudWatch to collect and track metrics, create alarms, automate actions, and visualize data in dashboards. Some of the services that CloudWatch provides are:

- **CloudWatch Metrics:** You can use CloudWatch Metrics to monitor the utilization and status of your AWS resources, such as CPU, memory, disk, and network. You can also create custom metrics for your own applications and services.
- **CloudWatch Alarms:** You can use CloudWatch Alarms to trigger notifications or actions when a metric crosses a specified threshold. For example, you can send an email, SMS, or Amazon SNS message when your CPU utilization is too high or too low.

1:08

## 11. AWS Part -1 105. Cloudwatch Introduction



- **CloudWatch Events:** You can use CloudWatch Events to deliver a stream of real-time events from AWS services, such as EC2, Lambda, S3, and DynamoDB. You can also create custom events for your own applications and services. You can use CloudWatch Events to trigger actions, such as invoking a Lambda function, running an EC2 instance, or sending a message to a queue.
- **CloudWatch Logs:** You can use CloudWatch Logs to store and access log files from your AWS resources, such as EC2 instances, Lambda functions, and

CloudTrail. You can also create custom logs for your own applications and services. You can use CloudWatch Logs to monitor, filter, search, and analyze your log data.

- **CloudWatch Dashboards:** You can use CloudWatch Dashboards to create and share graphical displays of your CloudWatch data. You can customize your dashboards with widgets, such as charts, tables, text, and images. You can also embed your dashboards in other web pages or applications.

3:24

## 11. AWS Part -1 105. Cloudwatch Introduction



### AWS Cloud Watch

Monitor performance of AWS environment - standard infrastructure metrics. Some of the metrics are assigned by default , if we go to any resource , under that , check the Monitoring tab , then we will see some analysis in there , those are rendering from Cloud Watch. We can also create custom metric and create a cloud watch for that.

3:25

## 11. AWS Part -1 105. Cloudwatch Introduction



Following are possible ways we use Amazon Cloud Watch ,

**Metrics:** AWS cloud watch allows. you to record metrics for services such as EBS, EC2, ELB, Route53 Health checks, RDS, Amazon S3, cloudfront etc. etc. ...

**Events:** AWS Events delivers a near real-time stream of system events that describe changes in Amazon Web Services (AWS) resources. Basically we can create some events , so that when that event is triggered , we can perform some action on the resources , for example , if the resource is 75% full , we can extend the EBS Storage or get an alert to do that.

**Logs:** You can use Amazon CloudWatch Logs to monitor, store, and access your log files from Amazon Elastic Compute Cloud (Amazon EC2) instances, AWS CloudTrail, Route 53, and other sources. By default , we won't have Logs generated and displayed , We must retrieve the logs , by establishing an agent in VM and use that to get logs to AWS Cloud Watch

3:56

## 11. AWS Part -1 105. Cloudwatch Introduction



Following is **one of use case to use Amazon CloudWatch**

We use Cloud Watch metrics of instances or resources to *set Realtime alarm* , so that we can watch over the resource extreme conditions , this helps to avoid occupying total resources or prevent over costing etc.

For this , we use *Simple Notification Service (Amazon SNS)* , which is a web service that coordinates and manages the delivery or sending of messages to subscribing endpoints or clients.

1:17

## 11. AWS Part -1 106. Cloudwatch Hands On



We will get details in Cloud Watch monitoring like

```
1 | CPU Utilization  
2 | Network in (Bytes)  
3 | Network out (Bytes)  
4 | Network packets in (count)  
5 | Network packets out (count)  
6 | Disk reads (bytes)  
7 | Disk read operations (operations)  
8 | Disk writes (bytes)  
9 | Disk write operations (operations)  
10 | CPU Credit Usage (count)  
11 | CPU Credit Balance (count)
```

If we want some utilization metrics like RAM Utilization , Storage Utilization , we must create custom metrics for those and add them to the cloud watch , those are present by default.

The Cloud Watch will update its metrics and it's graphs for every 5 mins , and if we want for every second or more detailed analysis , we must use "Manage Detailed Monitoring" Option , enable that option. Now we will get detailed monitoring for every 1 minute , and the graphs will also update for every minute.

***Basic monitoring is free , where as Detailed monitoring is a paid service , which is cheaper in cost.***

3:30

## 11. AWS Part -1 106. Cloudwatch Hands On



### Installing Stress Library in EC2 instance

```
1 | sudo amazon-linux-extras install epel -y  
2 | sudo yum install stress -y
```

The stress library is a tool that can generate CPU, memory, disk, and network load on a system. It is used in AWS to test the resilience and performance of applications and services under different stress conditions. For example, you can use the stress library to simulate high CPU utilization, memory pressure, disk I/O, or network congestion, and observe how your application responds to these scenarios. You can also use the stress library to benchmark your system and measure its capacity and limits.

One way to **use the stress library in AWS is through the AWS Fault Injection Service (FIS)**, which is a fully managed service for running fault injection experiments. FIS provides pre-configured SSM documents that use the stress library to run CPU stress, memory stress, disk stress, or network stress on EC2 instances. You can use FIS to create and run experiments that inject these faults and monitor the impact on your application's performance, observability, and resilience. You can also use FIS to define stop conditions and rollbacks for your experiments, to ensure that they are safe and controlled.

Using stress , we are faking the EC2 instance to stress , which helps us to visually see the action of cloud watch on the AWS.

We use the following command to stress out for 300 seconds with help of 4 fake processes

```
nohup stress -c 4 -t 300 &
```

Now , if we run the command top on the EC2 instance , we will see our 4 fake processes are running on the background.

`stress -c 4 -t 300` is the main command , it is surrounded by `nohup &` only to make the command to execute in the background silently.

We can fake the stress process , and generate a fake scenario , where the EC2 instance is sleeping for a time and working for a time , using the following command.

```
sleep 60 && stress -c 4 -t 60 && sleep 60 && stress -c 4 -t 30 && sleep 30 &&
stress -c 4 -t 200 && sleep 30 && stress -c 4 -t 500
```

We will get a graph under CPU utilization on the Cloud Watch.

Create an alarm on the Cloud Watch home page of AWS , select an instance , and choose the metric for which threshold we want the alarm to trigger. We will get an option to give our email's or set of email's to send alarm when the threshold triggers.

We can also take action , while we are triggering the alarm it self , we are having an option under **EC2 actions** in there , when alarm got triggered , either we can stop the

instance or terminate or reboot the instance. We also having an option of **Auto Scaling Option**.

0:19

## 11. AWS Part -1 107. EFS



If we want a shared storage on AWS , we can use Amazon Elastic File System. If we have cluster of servers , and if we want to store that data at one centralized place , we can go with the EFS.

**Cost of EFS:** Scalable, elastic, cloud-native NFS file system for \$0.08/GB-Month

**Amazon Elastic File System (Amazon EFS)** provides a simple, scalable, fully managed elastic NFS file system for use with AWS Cloud services and on-premises resources. It is built to scale on demand to petabytes without disrupting applications, growing and shrinking automatically as you add and remove files, eliminating the need to provision and manage capacity to accommodate growth.

Amazon EFS offers two storage classes: the **Standard storage class, and the Infrequent Access storage class (EFS IA)**. EFS IA provides price/performance that's cost-optimized for files not accessed every day.

*We can assign EBS for one instance at a time , where as EFS can be assigned to multiple instance at a time , so it supports Shared Storage*

1:53

## 11. AWS Part -1 107. EFS



**AWS EFS** is a cloud file storage service that can be **used for various purposes**, such as:

- **Web serving and content management:** You can store and serve web content, such as images, videos, documents, and scripts, from an EFS file system that can scale on demand and provide high availability and durability.
- **Enterprise application usage:** You can migrate or build enterprise applications on AWS using EFS as a shared file system that supports standard file system interfaces and semantics, and integrates with other AWS services.
- **Media and entertainment:** You can store and process large volumes of media files, such as audio, video, and graphics, on EFS and leverage its performance and scalability for workflows such as transcoding, rendering, and streaming.
- **Shared and home directories:** You can use EFS to create and manage user home directories or shared folders that can be accessed by multiple instances or users across different regions and availability zones.

1:54

## 11. AWS Part -1 107. EFS



- **Database backups:** You can use EFS to store backups of your relational or non-relational databases, and restore them quickly and easily when needed.

- **Developer and application tools:** You can use EFS to store and share code, configuration files, logs, and other assets for your development and testing environments, and use tools such as Git, Jenkins, or Docker with EFS.

If we go in detail , if multiple users are uploading their data into the website which was hosted by multiple instances , and all those instances can be attached to a single EFS storage , where user data will be uploaded into EFS Storage. And all those instances will be sync with the storage , so user can have Realtime updates based on other user uploaded data.

6:03

## 11. AWS Part -1 107. EFS



Ok , now for creating EFS file system , we need to create a security group for that , where , the inbound rules will be like Type is NFS and the Source should be of custom IP where it should accept all request which are coming from the security group of the Instance that we hosted a website in it.

And next , we need to create a EFS , where we will have customization , where we will have some features free under free tier , among them Under Performance Mode => General Purpose is free & Max I/O is paid and Under Throughput Mode => Bursting is free & Provisioned is paid.

And next under mount targets of EFS customization , give the security group of the EFS , that we created just before for all the availability zones. So that the request which is coming through the instance can reach the EFS by passing by security group of EC2 instance and the security group of EFS that we created.

We need to create an access point , if we want to access the EFS. Which can be created under EFS window.

7:18

## 11. AWS Part -1 107. EFS



After creating access point , if we want to access EFS using this access point using amazon-efs-utils using Amazon Linux. For that , we need to install that using the following command.

```
sudo yum install -y amazon-efs-utils
```

If we are using any other centos machine , we can install efs-utils using ,

```
sudo yum -y install git
```

```
git clone https://github.com/aws/efs-utils
```

```
sudo yum -y install make
```

```
sudo yum -y install rpm-build
```

```
sudo make rpm
```

```
sudo yum -y install ./build/amazon-efs-utils*rpm
```

If we are using any other debian machine , we can install efs-utils using ,

```
sudo apt-get -y install binutils  
./build-deb.sh  
sudo apt-get -y install ./build/amazon-efs-utils*deb
```

10:41

## 11. AWS Part -1 107. EFS



If we want to create a mount point from the EC2 instance to the EFS , we need to place the below string in the fstab file , so that , whenever we reboot the machine , the mount point will be auto created , this is the best way to do , if we want to access through access point.

Syntax:

```
file-system-id efs-mount-point efs _netdev, tls, accesspoint=access-point-id 0  
0
```

file-system-id , we will get on EFS details

access-point-id , we will get on Access-point details , we can get the file-system-id which is attached to the access-point under Access-point itself.

That will be looking like this ,

```
fs-47a7ccb2 /var/www/html/img efs _netdev, tls, accesspoint=fsap-  
03f6334520365d2d7 0 0
```

In the process , if we get any timeout error , check the security groups , if we are getting any filesystem error , check the file system id and access point id.

14:06

## 11. AWS Part -1 107. EFS



Now , we are creating the AMI based on the EFS storage enabled EC2 instance , so , we can use this , whenever we want , instead of recreating all the structure all over again , all the security group and instance details , will be stored on this AMI , so we no need to give rules again.

1:05

## 11. AWS Part -1 108. Autoscaling Group Introduction



**Auto Scaling** is a service that automatically monitors and adjusts compute resources to maintain performance for applications hosted in the AWS. Auto Scaling closely works with Alarm monitors of Cloud Watch metrics.

If the CPU of an instance is around threshold , we will be creating another instance to support that instance under the same load balancer , and if there are more number of instances , but less usage of those resources , this auto scaling reduce the resources , so that we can upscale and downscale dynamically , So , by upscaling we can ensure

24/7 user satisfaction , and by down scaling whenever it is required , we can save the cost.

A launch configuration/Template is an instance configuration template that an Auto Scaling group uses to launch EC2 instances.

Scaling policy is used to increase and decrease the number of running instances in the group dynamically to meet changing conditions.

3:10

## 11. AWS Part -1 108. Autoscaling Group Introduction



**Under Auto Scaling , we will have three terminologies to mention ,**

**Minimum Size:** Minimum number of Instances we nee to run at the least , even if no process is running on the instances.

**Desired Capacity:** Average number of Instances , that we prefer to run , to give good user experience to the websites for the user.

**Maximum Size:** Maximum number of Instances that you can launch , if huge amount of users , surf our website at once , Auto Scaler can't launch more instances than maximum size. This is necessary to setup , for cost control.

In general , maximum size of instances will be reaching at the time of huge load , and sometimes , the server get's lost , as the number of instances that are launched are not enough to the user base that the site is currently having. This time , we see , 'Site Not Reachable' , sometimes , even if one instance is also not allocated , we would face this issue.

Based on the scaling policy , auto scaling group creates or removes instances.

2:31

## 11. AWS Part -1 109. Autoscaling Group Hands On



For **Autoscaling group demo** , we need to create target group which allows the requests from the Load balancer , so add security group of load balancer into the inbound rules of the target group. And also create a load balancer.

And we also need to have an AMI , which is built on the instance that we have previously , and to launch that AMI , we are creating a customized Launch template , which is already having rules like Inbound Rules , Outbound rules, AMI to be used etc.

2:33

## 11. AWS Part -1 109. Autoscaling Group Hands On



Now create an **Auto Scaling Group** , we need to give basic details like *Auto Scaling Group name* , *Launch template* which we use to launch further instances dynamically

while doing upscale or downscale. We can also use launch configuration instead of Launch templates , but Launch template is the preferable one. We need to choose , *availability zones that we want to host our auto scaling groups on*. It is preferable to attach at least two availability zones. If we want to attach this auto scaling group to an load balancer , we can give which load balancer that we wanted it to attach to , or create it immediately. And if we wanted to attach to an existing load balancer , we need to select the target groups that we earlier created and attached to the load balancer.

5:59

## 11. AWS Part -1 109. Autoscaling Group Hands On



And under *Health Check* , we will have two options for the health check , one is checking the *EC2 level health check* , it is basically *software and hardware level health check* , there will be situations where software and hardware of the instance are fine , but application may be not in running state , which will be checked by the *ELB (Elastic Load Balancer)* , which gives the user request to the instance , and if it get success code as a response , then it will be healthy , else no. So , such instances will be identified by the load balancer with the help of the target group , and auto scaling group create another instance by removing the stopped instance. We can choose , how frequent that , this Auto Scaling group need to do this health check , by default it is 300 seconds.

We also declare the **Group Size** , under Auto Scaling Group configuration , where Desired Capacity , Minimum Capacity , Maximum Capacity should be declared.

**Minimum Capacity <= Desired Capacity <= Maximum Capacity**

7:42

## 11. AWS Part -1 109. Autoscaling Group Hands On



Based on the scaling policies , we declare in here as part of configuration of Auto Scaling group , the number of instances will be auto scale between Minimum and Maximum Capacity number of instances.

We can also send notification to ourself , either via SMS or through mail for events of auto scaling like Launch a new instance , Terminate , Fail to launch , Fail to terminate.

We also can give 'tag' for identification of this auto scaling resource , which helps us for filtering out from pool of resources.

11:13

## 11. AWS Part -1 109. Autoscaling Group Hands On



Under instance management section of our Auto Scaling group , we can detach , standby , Inservice , set scale-in protection , remove scale in protection for instances individually.

When we are doing some trouble shooting , sometimes there is possibility of launching n number of instances by mistake , so we can set scale in protection of auto scaling group , to avoid such instances.

Its not preferable to make changes to the instances of auto scaling group , because , these changes are temporary , if we do login to the instance and made a change , if the auto scaling group creates a new instance then , it won't have the change that you made , so if we want to make any change to the instance , change the template , so that , whenever , we launch any instance further , it will have your modified changes.

14:56

## 11. AWS Part -1 109. Autoscaling Group Hands On



So , when we want our changes to be applied , we should update those changes in the Launch template , and we need to refresh the auto scaling group to make our changes to be applied on the instances , so we will have an option to "**Refresh instance**" , in there , we must keep minimum healthy percentage to 90% , it means if we have 10 instances , we need to apply those changes on one instance , while 9 other are in running state. It will do the same for all the instance , it will update one instance at a time.

So , by the help of auto scaling group , even if we delete one of our instance manually , auto scaling group will create an instance based on desired , maximum , minimum capacity that we given as group size under auto scaling group.

We should not store any data in EC2 instances , because , we made those instances as dynamic , so the data could be lost due to a refresh or reset, we need to mount the instance with the EBS or EFS, so that, we won't lose the data, while the reset happens.

18:20

## 11. AWS Part -1 109. Autoscaling Group Hands On



Deleting the auto scaling group is the only way to delete our instances which are under it , else the instances will be up again according to the scaling policy. And we also need to remove Load balancer , which is attaching to this auto scaling group's target group , if we want to do a project cleanup.

0:08

## 11. AWS Part -1 113. RDS



**Relational Database Service** solves the DB Administration Problem , It take care of ,  
*Installation* of DB in the first place  
*Patching* for the DB server  
*Monitoring* the DB

*Performance Tuning*

*Backup of the Database*

*Scaling the Database , depending on the user base*

*Security & Privacy for user data*

*Hardware upgrades will be taken care smoothly by RDS*

*Storage Management tools will be available for better Storage management*

2:37

11. AWS Part -1 113. RDS



### **Relational Database Service:**

Amazon Relational Database Service is a **distributed relational database service**.

**High Availability Multi-AZ Deployments:** We can host RDS on two servers of two availability zones , considering one server as the primary and the second as secondary , whenever the primary got corrupted , the secondary become the new primary , the primary and the secondary will be in sync.

### **Effortless Scaling.**

**Read Replicas for performance:** We will use read replicas , for reading the data for the applications , due to this , we will see a spike in the performance , because , we are using some read-only copies through some code for the application , instead from the database directly , this decreases the load on the database.

2:38

11. AWS Part -1 113. RDS



Read replicas can also be used for disaster recovery, business reporting, or data warehousing scenarios. To create a read replica, you need to enable the built-in replication features of the database engine. Amazon RDS supports read replicas for MySQL, MariaDB, PostgreSQL, Oracle, and SQL Server as well as Amazon Aurora , The replication process is asynchronous, which means that there might be some lag between the data on the primary instance and the read replica.

You can create one or more read replicas for a given primary instance, and you can also create read replicas in different AWS regions for cross-region replication. You can monitor the replication status and performance of your read replicas using Amazon CloudWatch metrics. You can also promote a read replica to a standalone instance if needed.

3:11

11. AWS Part -1 113. RDS



### **Basic use case of RDS:**

We will have RDS which is private to the VPC , and where as we will have an EC2 instance , to host an Web Application , and that web application , we can access through the internet with the help of internet gateway which is present in the VPC , which helps us to connect. In here RDS is private to the VPC , EC2 instance let's its inbound rules to be public , so we can access the RDS through the website , but not manipulate that directly.

Search RDS through a website , next , select "**Choose a database creation method**" as one of below ,

**Standard create :** You set all of the configuration options, including ones for availability, security, backups, and maintenance.

**Easy create:** Use recommended best-practice Configurations. Some configuration options can be changed after the database is created.

3:29

## 11. AWS Part -1 113. RDS



*AWS suggests , if we wanted to use MySQL or PostgreSQL Database , then it is preferable to use AWS Aurora. Aurora supports up to 64TB of auto-scaling storage capacity, 6-way replication across three availability zones, and 15 low-latency read replicas. Amazon Aurora is 5 times faster than MySQL and 3 times faster than PostgreSQL. It is a serverless architecture.*

Later , in specifying the configuration for the database , we need to provide the type of the Engine , we want , the possible ones are Amazon Aurora , MySQL , MariaDB , PostgreSQL , Oracle , Microsoft SQL Server. We are choosing MySQL , we also need to select its version down below.

5:26

## 11. AWS Part -1 113. RDS



We also needed to select the template for the Database , it should be either Production , Dev/Test , Free tier.

Difference among them is ,

**Production:** Use defaults for high availability and fast, consistent performance.

**Dev/Test:** This instance is intended for development use outside of a production environment.

**Free tier:** Use RDS Free Tier to develop new applications, test existing applications, or gain hands-on experience with Amazon RDS.

We are selecting Dev/Test.

We need to choose **DB instance identifier** , that name should be unique in our region of our AWS account.

We need to provide the username and the password for the Database , we can set password to be auto generated.

We need to provide **DB instance size** , where , we will have three DB instance classes , one is Standard Classes ( includes m classes )

Memory Optimized ( include r and x classes )

Burstable Classes ( include t classes )

We use Burstable Classes , to use Storage which is coming under free tier.

6:32

## 11. AWS Part -1 113. RDS



We also need to select **type of the storage that we want in RDS** , there are three types ,

**General Purpose (SSD)**

**Provisioned IOPS (SSD)**

**Magnetic**

We need to select amount of storage , that we want to be allocated. We can also choose Storage autoscaling. This Provides dynamic scaling support for your database's storage based on your application's needs. We can establish the upper limit of the storage , that can be scaled up , if we set a threshold value , we can't auto scale more than that , this is typically used to cut the cost.

We can choose **Multi AZ deployment by creating a Standby instance** in here.

7:01

## 11. AWS Part -1 113. RDS



**Multi-AZ deployment** in AWS RDS is a feature that **provides high availability, durability, and performance for your relational databases**. It allows you to *have one primary database instance and one or two standby replicas in different Availability Zones (AZs) within the same region*. The *data is synchronously replicated from the primary to the standby instances using the native replication capabilities of the database engine*. **In case of a failure of the primary instance, RDS automatically fails over to one of the standby instances with minimal downtime and no data loss**. You can also *use the standby instances to serve read-only queries and increase the read throughput of your application*

7:06

## 11. AWS Part -1 113. RDS



To convert a single-AZ RDS instance to a multi-AZ deployment, you can use the RDS console, the AWS CLI, or the RDS API. You need to choose the option to apply the changes immediately or during the next maintenance window. The conversion process involves creating a snapshot of the primary instance, restoring it to a new instance in a

different AZ, and setting up the replication between the primary and the standby instances. During the conversion, the primary instance is unavailable for a brief period of time.

Multi-AZ deployment is supported for the following database engines: MySQL, PostgreSQL, MariaDB, Oracle, and SQL Server. The pricing for multi-AZ deployment is higher than single-AZ deployment, as you pay for the additional standby instances and the data transfer between them. However, you can benefit from the enhanced availability, durability, and performance that multi-AZ deployment offers for your mission-critical workloads.

7:11

## 11. AWS Part -1 113. RDS



**RDS replicas** will be used to increase read throughput for our application.

Main RDS , will only be used for data modification process.

We can set the subnet access to public or private in here , if we keep it to public , we can access the RDS over the internet , that can be also a use case , where as if we keep , subnet to private , we can access RDS only over our VPC. We need to create a VPC security group , so that can specify the inbound and outbound rules for the VPC.

We can set RDS authentication , through username and the password , where as , we can also authenticate it using username and IAM user.

We can choose **Backup Retention period** , where number of days that RDS should retain automatic backups for this instance. We can also set the backup window , such that , we can give rules like , how frequent , we must initiate the backup. Backup Retention period is maximum 35 days , after 35 days , we will be loosing the data. The backup files , will be looking like snapshots.

9:01

## 11. AWS Part -1 113. RDS



We can set up backup replication , where we can store the backup on to another region , but this increases our storage cost. Can enable encryption. We can enable monitoring , we can monitor the RDS , as low as 1 second. We have to set up the monitoring role , and we can also export logs to Amazon CloudWatch Logs , some of the logs , that we can export are ,

**Audit Log , Error Log , General Log , Slow query Log**

Under maintenance section , we can check "Enable auto minor version upgrade" , where , we can check this option , such that , if there are any minor upgrades in SQL , we will auto upgrade , it won't upgrade , only if that upgrade is a major upgrade. We can enable , delete protection , such that , no one can delete this RDS instance , accidentally.

Finally , it will give the estimated monthly costs , for using the RDS with our customizations.

It will charge for the ,

DB Instance , Storage , Multi AZ standby instance

Able to watch the credentials only once on screen , can reset though.

16:09

## 11. AWS Part -1 113. RDS



We can connect to the RDS , through command line , using ,

`mysql -h <RDS-endpoint> -u <user-name> -p <password>`

If we are not able to login to the DB , we need to do check , are we able to reach the server or not , through telnet

`telnet <RDS-endpoint> <port-num>`

If we are not able to connect , to the RDS endpoints , then first we need to check the security group of RDS , we need to edit the Inbound Rules for the Security Group of RDS , where it must allow all the request coming from the security group of Instance on which application is running on , where as , we can also add a rule , such that , all the requests that are coming from the instance private IP address , it must allow , if we add those conditions to the RDS , then we will be able to access the RDS server (SQL Server) through MySQL command through the instance.

18:57

## 11. AWS Part -1 113. RDS



We can create a read replica from the home page of RDS using the RDS that we created earlier , and if we have chosen to create snapshot , while creating RDS , it will be also created , if we deleted our RDS , we can create the same RDS , using that snapshot.

There are so many automation done in RDS , so , we won't be requiring a Database administrator , a normal software engineer , can maintain that , with the minimum knowledge in AWS and SQL.

0:07

## 19. AWS Part-2 207. VPC Introduction



Initially , we have data centers at the time , we don't have any fully developed EC2 instance. In that data center components , we would have different networking components like Switches , Routers , Firewall. In those network , we will be having lots of subnets , subnet can be allocated to a process , that could be one subnet can be connected to the frontend service of the project , where as another subnet can be connected to the backend service of the project.

2:21

## 19. AWS Part-2 207. VPC Introduction



We use subnets to decide , what traffic to come to the subnet , and what not . Some subnet , will have access to all the traffic , and some subnets use Network ACL to filter the traffic , which is coming through the Router. All these models , will be created and managed , by the networking team , of what rules , should we assign to the network components , and what architecture , that we wanted to build up.

Initially , we have the option , to create EC2 instance in the cloud and use that , but we don't have the option to create such architecture , and create custom network configuration , So , AWS , have introduced the topic of VPC , which is virtual private cloud.

2:19

## 19. AWS Part-2 207. VPC Introduction



A VPC, or Virtual Private Cloud, is a service that lets you launch AWS resources in a logically isolated virtual network that you define. The main benefit of using a VPC is that it gives you full control over your virtual networking environment, including resource placement, connectivity, and security. You can also customize your VPC by choosing your own IP address range, creating subnets, and configuring route tables. A VPC also offers enhanced security by keeping traffic within the AWS network, reducing exposure to the public internet. Additionally, a VPC can help reduce data transfer costs and improve performance by providing low-latency access to AWS services.

VPC is like a private network that you can create on AWS, where you can run your own resources, such as servers, databases, or applications. You can decide how your network looks, who can access it, and what services you can use. A VPC helps you keep your data secure, save money, and improve performance.

2:29

## 19. AWS Part-2 207. VPC Introduction



We can distribute our subnets to multiple availability zones under our VPC Network for high availability.

2:40

## 19. AWS Part-2 207. VPC Introduction



**VPC is called Virtual Private Cloud** , we can create logical data center within an AWS Region. This is pretty similar to Local Area network , where we will have whole control on the infrastructure , of what IP scheme to use , securities and firewalls. Once the VPC is created , we can create Instances , RDS and majority of services in VPC. Basically we will

have a default VPC for every region , where as if we create our own VPC , we will have more control on the VPC.

VPC is an on-demand configurable pool of shared computing resources which are allocated to us within a public cloud environment. Now the allocated resources are private to us. We will have control over environment , we can select IP address range , subnets and configure route tables and gateways.

4:30

## 19. AWS Part-2 207. VPC Introduction



An IPv4 Address is typically a 32 bit address , which is divided by four octets , where each octet is of 8 bits , we will have four such octets in an IPv4 address. IPv4 address , will look something like **192.168.100.1** , the following decimal number which is representing as IPv4 address , is derived from binary number.

5:39

## 19. AWS Part-2 207. VPC Introduction



IPv4 Range => **0.0.0.0** - **255.255.255.255** , in binary representation it looks like

**00000000.00000000.00000000.00000000 => 0.0.0.0**

**11111111.11111111.11111111.11111111 => 255.255.255.255**

This IPv4 Address is divided into two parts ,

Public IP address => Used for websites of Internet **54.86.23.90**

Private IP address => Used for local network design , these are the private IP , which can only be used under the particular network. **192.168.1.10**

### Private IP ranges

Class A => 10.0.0.0 - 10.255.255.255

Class B => 172.16.0.0 - 172.31.255.255

Class C => 192.168.0.0 - 192.168.255.255

8:49

## 19. AWS Part-2 207. VPC Introduction



Certainly! Let's delve into the purposes of **Class D** and **Class E** IP addresses:

#### 1. Class D (Multicast Addresses):

- **Purpose:** Class D addresses are reserved for **multicast networking**.
- **Characteristics:**
  - The higher-order bits of the first octet of Class D IP addresses are always set to **1110**.

- These addresses are used for **one-to-many communication**, where data is sent from one sender to multiple receivers.
- Multicast addresses allow efficient distribution of data, such as audio or video streams, to a group of interested hosts.
- Examples of applications using Class D addresses include **streaming services, video conferencing, and online gaming**.
- **Note:** Class D addresses cannot be used for regular unicast traffic.

8:51

## 19. AWS Part-2 207. VPC Introduction



### 1. Class E (Reserved Addresses):

- **Purpose:** Class E addresses are **reserved** for future or experimental purposes.
- **Characteristics:**
  - These addresses are not commonly used in practice.
  - The higher-order bits of the first octet of Class E IP addresses are set to **1111**.
  - Class E addresses are not assigned for general network communication.
  - They are primarily reserved for **research, development, and specialized experiments** by those responsible for Internet networking and IP address management.
- **Note:** Class E addresses are not used on the public Internet and are not part of regular IP routing3.

In summary, while Class D addresses facilitate multicast communication, Class E addresses remain reserved for future innovations and research.

14:27

## 19. AWS Part-2 207. VPC Introduction



### Purpose and significance of subnet masks:

#### 1. Definition:

- A **subnet mask** is a 32-bit address that divides an **IP address** into two parts:
  - **Network bits:** Identify the network to which the device belongs.
  - **Host bits:** Identify the specific device (host) within that network.
- It encapsulates a range of IP addresses that a **subnet** can use, where a subnet refers to a smaller network within a larger network.

14:28

## 19. AWS Part-2 207. VPC Introduction



## Purpose and significance of subnet masks:

### 1. Functionality:

- **Network Segmentation:** Subnet masks allow the division of a larger IP network into smaller, more manageable subnetworks. Each subnet can have its own set of IP addresses.
- **Efficient Address Allocation:** By creating subnets, organizations can allocate IP addresses more efficiently. For example, different departments or floors in a company can have separate subnets.
- **Broadcast Domain Control:** Subnetting limits the broadcast domain, reducing unnecessary broadcast traffic. Broadcasts are confined to devices within the same subnet.
- **Security and Isolation:** Subnets enhance security by isolating different parts of a network. Devices in one subnet cannot directly communicate with devices in another unless routing is configured.
- **Routing Decisions:** Routers use subnet masks to determine whether an IP address belongs to the local network or needs to be forwarded to another network.

14:29

19. AWS Part-2 207. VPC Introduction



## Purpose and significance of subnet masks:

### 1. CIDR Notation:

- Subnet masks are often expressed using **CIDR (Classless Inter-Domain Routing)** notation. For example:
  - `/24` corresponds to a subnet mask of `255.255.255.0`.
  - `/16` corresponds to `255.255.0.0`.
- CIDR notation simplifies subnet calculations and allows efficient allocation of IP addresses.

### 2. Example:

- Consider an IP address `192.168.1.10` with a subnet mask `255.255.255.0` (equivalent to `/24`).
  - The first 24 bits represent the network portion (`192.168.1`).
  - The last 8 bits identify the host (`10`).
  - Devices within this subnet share the same network prefix.

In summary, subnet masks play a crucial role in organizing and managing IP networks, ensuring efficient address utilization and enhancing security. They are fundamental to modern networking.

13:33

19. AWS Part-2 207. VPC Introduction



If we are in IP address of **192.168.0.174** with a subnet mask of **255.255.255.0** , then

- 1 | 192.168.0.0 [Network IP]
- 2 | 192.168.0.1 => 192.168.0.254 [Usable IPs]
- 3 | 192.168.0.255 [Broadcast IP]

Broadcast IP is used to communicate with all other IP's in that subnet.

We will be having Total IP's of 256 and among them 254 IP's are usable.

15:22

## 19. AWS Part-2 207. VPC Introduction



If we are in IP address of **172.16.12.36** with a subnet mask of **255.255.0.0** , then

- 1 | 172.16.0.0 [Network IP]
- 2 | 172.16.0.1 => 172.16.255.254 [Usable IPs]
- 3 | 172.16.255.255 [Broadcast IP]

Broadcast IP is used to communicate with all other IP's in that subnet.

We will be having Total IP's of 65536 and among them 65534 IP's are usable.

21:14

## 19. AWS Part-2 207. VPC Introduction



If we are in IP address of **10.23.12.56** with a subnet mask of **255.0.0.0** , then

- 1 | 10.0.0.0 [Network IP]
- 2 | 10.0.0.1 => 10.255.255.254 [Usable IPs]
- 3 | 10.255.255.255 [Broadcast IP]

Broadcast IP is used to communicate with all other IP's in that subnet.

We will be having Total IP's of  $256 \times 256 \times 256 = 16777216$  and among them 16777214 IP's are usable.

22:24

## 19. AWS Part-2 207. VPC Introduction



### CIDR Notation (Classless Internet Domain Rotation)

**CIDR (Classless Inter-Domain Routing)** is a method for allocating IP addresses that allows for more flexible address assignment and routing. Here's how it works:

#### 1. Background:

- In the past, IP addresses were divided into five classes (A, B, C, D, and E), each with fixed network sizes and host capacities.
- However, this system was inefficient, leading to wasted addresses and impractical network sizes.

#### 2. CIDR's Solution:

- CIDR introduced variable-length subnet masking (VLSM), allowing for more precise control over subnet sizes.

- Instead of fixed-length prefixes, CIDR allocates address space on any bit boundary.
- This flexibility helps slow the exhaustion of IPv4 addresses.

22:45

## 19. AWS Part-2 207. VPC Introduction



### CIDR Notation (Classless Internet Domain Rotation)

#### 1. CIDR Notation:

- CIDR notation represents an IP address followed by a slash ('/') and a decimal number.
- The decimal number indicates the count of consecutive leading 1-bits in the network mask.
- Examples:
  - IPv4: **192.0.2.0/24**
  - IPv6: **2001:db8::/32**

#### 2. Advantages:

- Fine-grained control over subnet sizes.
- Reduced global routing table entries through aggregation.
- Efficient address allocation.

24:46

## 19. AWS Part-2 207. VPC Introduction



### CIDR Notation

1 | Subnet Mask: 255.0.0.0  
2 | Decimal Representation: 11111111.00000000.00000000.00000000  
3 | CIDR: /8

1 | Subnet Mask: 255.255.0.0  
2 | Decimal Representation: 11111111.11111111.00000000.00000000  
3 | CIDR: /16

1 | Subnet Mask: 255.255.255.0  
2 | Decimal Representation: 11111111.11111111.11111111.00000000  
3 | CIDR: /24

CIDR represents number of leading decimals which represents network and the later once for representation of the host.

For example,

1 | 172.20.0.0/16 => IP Address  
2 | 172.20 => Network Address  
3 | 0.0 => Host Address

**Wild Card** is at flip side of Subnet mask , if subnet mask is 255.255.0.0 then wild card will be 0.0.255.255

VPC is the main network, where we have smaller subnets, we will have **two kinds of subnets** in the VPC, those are **public subnets and private subnets**.

Public subnets are the one which can be reached through internet. The instances which are launched through public subnets are able to access the internet through the internet gateway, which is part of VPC, whereas instances which are part of private subnet can't access the internet because private subnet won't have the access for the internet as it doesn't have a public IP, whereas instances under public subnet will be able to access the internet as they are having the public IP address.

If the instances under private subnet wanted to access internet for installing some service, we can do that through **VPC NAT Gateway**, which is present in public subnet, which helps private subnet to access internet.

Each subnet will be assigned to an individual availability zone.

We can check our private IP in windows using **ipconfig** command on command prompt. Whereas we can check our public IP by searching "**What is my IP**" on google. Where our internet service provider given that IP for our router. If 10 people are using a same Wi-Fi, so they would also use a same router, all their user requests will go through the router which has the same IP address for all the users under it.

**Request originating from the private subnet, will pass through VPC NAT Gateway to internet, and carry the response from the internet in the same path backwards. But Request originating from internet cannot reach the private subnet anyway. It's like private subnets are arrogant, where it only asks requests and get its responses, but don't answer any requests.**

**Network Address Translation (NAT) gateway** to enable instances in a private subnet to connect to the internet or other AWS services.

An **Internet gateway** is a horizontally scaled, redundant, and highly available VPC component that allows communication between instances in your VPC and the internet.

We will be having route tables for each subnet, where the network ACL will fetch the route table to where the request needs to go, depends on the route table, router

redirects the request either to Internet Gateway if it is public subnet and NAT Gateway if it is private subnet, NAT Gateway requests Internet Gateway for our requests.

5:41

## 19. AWS Part-2 208. VPC Design & Components



To ensure high availability of VPC in a region, we will launch at least one Public and Private Subnets in each of availability zones. Zones are clustered data centers. If a subnet in any of the availability zone goes down or working slow, then other subnets will be supporting the load.

Every subnet will have a route table, requests from *private subnet will be redirected to VPC NAT Gateway* and requests from *public subnet will be redirected to Internet Gateway*.

Instances of public subnet can connect with the private subnet instances, where if we wanted to connect subnets across availability zones, we can connect two public subnets through internet, and we can connect a public subnet present in AVZ1 with private subnet present in AVZ2 with the help of bastion host which is present in the public subnet which is under AVZ2 , using which you can connect to private subnet , which is present under same availability zone.

VPC NAT Gateway is charged by AWS. VPC,subnets,route tables are not charged.

2:45

## 19. AWS Part-2 209. VPC Setup Details



If we wanted to ensure high availability, we need to have at least 1 VPC NAT gateway for each availability zone. We will be having two route tables assigned one for public subnet and one for private subnet. These route tables redirect the request from private subnet to NAT Gateway and from public subnet to Internet Gateway.

For each and every subnet, we will have an NACL, which is similar to security group of instances, we can define rules for subnets in there.

### Architecture needed to build an VPC:

- 1 | Region: us-west-1
- 2 | VPC Range 172.20.0.0/16
- 3 | 4 subnets: 2 pub **sub**, 2 priv **sub**
- 4 | 2 zones: us-west-1a, us-west-1b
- 5 | 172.20.1.0/24 => pub-sub1 [us-west-1a]
- 6 | 172.20.2.0/24
- 7 | 172.20.3.0/24
- 8 | 172.20.4.0/24
- 9 | 1 Internet GW
- 10 | 2 NAT Gateway
- 11 | 2 EIP (Elastic IP)

12 | 2 Route Tables: 1 Pub Sub RT, 1 Pub Sub RT

13 | 1 Bastion host in Pub subnet

To save the cost, we are using only 1 NAT Gateway and 1 EIP

2:14

## 19. AWS Part-2 210. Default VPC



Every region will have a default VPC, under that, we will be having two default public subnets.

### How to check a subnet is a private subnet or a public one?

We can select the subnet that we wanted to check the status of , and check the route table of that , if an entry in the destination address of the route table is in the range of the subnet IPv4 range , then it's local communication in the subnet , and if we also find , any other IP other than IPv4 range , for example , 0.0.0.0/0 whereas the target for it is something like 'igw-09a1b2f759a2b29c7' , it denotes the target as Internet Gateway. It means any other request coming from subnet from other than the default IPv4 range, it means, all those requests are forwarded to Internet Gateway for the response.

If the subnets are routing the traffic through route tables to the Internet gateway, then those are **public subnets**.

If the subnets are routing the traffic through route tables to the NAT gateway, then those are **private subnets**.

6:43

## 19. AWS Part-2 210. Default VPC



**We shouldn't delete the default VPC and the route tables.** If we delete them by mistake, we must contact the amazon service for re-creating them, we can't create default VPC, by our own, but we can create our own VPC and route tables and attach it to our default VPC.

1:51

## 19. AWS Part-2 211. Create VPC



If we wanted to create only the VPC, we need to select "VPC only" option under our VPC creation window and give the name of the VPC, CIDR range which will be in the form of **x.x.x.x/x** and create VPC.

3:47

## 19. AWS Part-2 211. Create VPC



If we wanted to create the whole architecture which includes public and private subnets, NAT and Internet Gateway, then we need to select "VPC and more" option under our VPC

creation window, later we need to provide tag for the VPC, and the IPv4 address CIDR Range. And if we needed a dedicated hardware for our VPC then we need choose the option for tenancy as 'Dedicated'. Else leave that for default. We also needed to choose number of availability zones needed and we also can customize the availability zone locations. We also need to choose, number of public subnets and private subnets required for us, if our availability zones are in even number then our required subnets also be in even numbers, for example, if no of AZ is 2 then public subnet count would be 0 , 2 and private subnet count would be 0, 2, 4.

3:48

## 19. AWS Part-2 211. Create VPC



We can also customize the range of public and private subnet CIDR range, in here for example.

We assigned,

- 1 | 172.20.0.0/24 => VPC Range
- 2 | 172.20.1.0/24 => Public Subnet 1 CIDR range
- 3 | 172.20.2.0/24 => Public Subnet 2 CIDR range
- 4 | 172.20.3.0/24 => Private Subnet 1 CIDR range
- 5 | 172.20.4.0/24 => Private Subnet 2 CIDR range

We need not to give the similar CIDR range for the subnets, but this looks look-able.

We will also have an option to choose, number of NAT Gateways to create, this service of AWS is chargeable, we will have an option to choose like

- 1 | **None** => No NAT Gateways
- 2 | In 1 AZ => 1 NAT Gateway will be present in one AZ's public subnet, which will be used for all private Subnets
- 3 | 1 per AZ => 1 NAT Gateway will be present for each AZ, this ensures high availability, but more costly.

If we have stored all our data in S3, and we don't want to access that DB, through NAT Gateway, which is resource costly, we must use VPC endpoints and add S3 Gateway, we will get this option while customizing the VPC.

4:42

## 19. AWS Part-2 211. Create VPC



Endpoints can help reduce NAT gateway charges and improve security by accessing S3 directly from the VPC. By default, full access policy is used. You can customize this policy at any time.

After customizing, we will be able to preview our created architecture in here. We can see, how many public and private subnets we are having and under which AZ they are under, and through which route tables they are reaching the NAT Gateway's and Internet Gateway. And if we have chosen to have VPC endpoints, we can use the S3 Bucket privately through private subnets instead of redirecting through NAT Gateway and

routing through Internet, we can create a direct link between the S3 and Subnets through end points.

0:45

## 19. AWS Part-2 212. Subnets



We can create subnets individually by our own under the subnet menu, we need to choose the subnet name and the IPv4 CIDR block address and under which availability zone the subnet is under, we should assign all these subnets by tagging them to the main VPC CIDR address. And all these subnets are having private IP addresses, so they are not discoverable for the internet.

0:57

## 19. AWS Part-2 213. Internet Gateway



### **Creating an internet gateway:**

An internet gateway is a virtual router that connects a VPC to the internet. We need to specify the name of the Internet gateway and create one.

For our internet gateway, if we see the state of that, we see it is detached, it means, we didn't attach our internet gateway for any VPC. So, we need to attach the IG to VPC that we created earlier. Then we will be changing the state of internet gateway from detached to attached.

0:16

## 19. AWS Part-2 214. Route Tables



We have to create pathway from Internet gateways to subnets through Route tables. So, create a route table by choosing VPC for our route table and create one route table, later we need to create rules in the route tables. For that, under route tables, under Edit subnet associations, we need to select the subnets that we want to associate and create an association. Basically, we create an association for the public subnets.

Later for that association under the created route tables, we need to edit one route table, where we need to add a new rule, such that 0.0.0.0/0 destination IP address requests should be targeted to our internet gateway that we created, so that our subnets will be public.

3:42

## 19. AWS Part-2 214. Route Tables



By the process we done, subnets will be having access to the internet through internet gateway, but we cannot access subnets, basically, we can access the subnets only when we are having some IP address for that, so, we won't have IPv4 address for the subnet by default, we need to enable the option to assign IP address to a subnet manually.

We need to "**Edit subnet settings**" under that, we need to "**Enable auto-assign public IPv4 address**", so that, each subnet will have its own unique IP address to recognize on the internet, so we can access the subnets through our terminal.

We have assigned our **public subnets to the internet gateway's**, we will assign our **private subnets to NAT Gateway**, in similar fashion.

1:16

## 19. AWS Part-2 215. NAT Gateway



For **NAT Gateway**, we need to allocate a static IP address, as we are managing the architecture by ourselves, so as we don't want dynamic IP address, we are creating a static IP address using Elastic IP address. Now while creating the NAT Gateway, we need to assign the **Elastic IP** that we have created earlier. And we also need to **assign Subnet for NAT Gateway**, we always need to select public subnet, so that, private subnet requests come to this NAT Gateway, and it can redirect the request to internet and get the response.

Now for that, we also need to create a route table, where we establish a connection between private subnets and the NAT Gateway. We do the same process as we done for the public subnet, but we need to target all the requests which we want to send to IP address other than the subnet's CIDR to NAT Gateway.

If we wanted to have hostnames with IP addresses for the subnets, we need to choose an option "**Enable DNS hostnames**" for the VPC by editing the VPC settings.

1:27

## 19. AWS Part-2 216. Bastion Host



**bastion host** is the single way *entry to access the resources of private subnet*, bastion host acts as a security group to private subnet which will be present in public subnet.

As said , bastion host acts as a security group for the private subnet , we need to create a security group and change it from default VPC to our created VPC. And underneath , we must give the inbound rules for the security group , we want to enable ssh connections for all the requests that are coming from My IP , so that , whenever we are trying to use the resources under private subnet , as the security group has a rule to allow all the requests which are coming from My IP , I will be able to access the resources.

4:55

## 19. AWS Part-2 216. Bastion Host



If we created this rule from our corporate network , then we will be able to access the resources of private subnet only from our corporate network , not from any other network.

We also needed to create a key-pair , which helps us to launch instance and we can also give this as the login key-pair while creating the EC2 instance.

AWS Marketplace AMI's are the one , which are well tested in performance.

7:19

## 19. AWS Part-2 216. Bastion Host



While Creating a EC2 instance , in there , we need to change the Default VPC to the VPC that we have created earlier , and we need to add our created key pairs & security group in here and add the required AMI and launch EC2 instance. After launching , we will be able to see our public IP address and the DNS. And if we are able to login to the IP address from our IP , then it shows that our public subnet is reachable to our IP address.

2:09

## 19. AWS Part-2 217. Website in VPC



Earlier we have created bastion host with a key created for that , and under the private subnet , we also creating another instance , in which where , we host our website , and we also will be creating another bastion host / load balancer under public subnet , so all the requests will be to & from this load balancer from the internet and the website.

So , to create an access between the bastion host and the EC2 instance which is present under private subnet , the bastion host , must have the access for public subnet. So , we copy the EC2 instance key-pair which is present under private subnet to the load balancer which is present under public subnet using the following command.

```
scp -i Downloads/vpro-bastion-key.pem Downloads/web-key.pem  
ubuntu@52.53.150.72:/home/ubuntu/web-key.pem
```

We are copying , web-key.pem from local to bastion instance using bastion-key.pem , We use this web key to login to web instance from the bastion instance.

3:21

## 19. AWS Part-2 217. Website in VPC



While creating an EC2 instance for hosting a website , ensure , we are using a private subnet , so we can make this demonstration meaningful. And while we giving security groups, we need to create a custom security group , where it should accept all the requests which are coming from bastion host , so assign the source as security group of

bastion key. And we also need to accept requests from the load balancer , we will add this rule , after creating the load balancer.

5:03

## 19. AWS Part-2 217. Website in VPC



Now , if we wanted to login to EC2 instance which is present under private subnet , we can do that through bastion host , as the instance which is present under the private subnet , doesn't have the public IP address , so we can't connect to that directly by username and IP address with key , as we are only allowing the requests into EC2 instance only if it is coming from the security group of bastion host , which is present under public subnet.

So , if we wanted to login to the EC2 instance under private subnet , we need to use the following command from the bastion host , by giving the key-pair of the destination host to connect.

```
ssh -i web-key.pem ec2-user@172.20.4.129
```

For the copied web-key.pem , we need to change the permissions for the key-pair file

```
chmod 400 <key-file>
```

Else we will get unprotected file error

5:37

## 19. AWS Part-2 217. Website in VPC



To install the website on the EC2 instance which is present under the private subnet , we need to login to this through the bastion host , which is discussed previously , and execute following commands.

```
yum install httpd wget unzip -y
```

httpd => download the packages

wget => download the website , from tooplate.com

unzip => unzip the downloaded template

Now we need to go to the tooplate website , and select the template that we wanted to download , and then open the network tab , after click the download button , then there will be one network which is coming up , which represents the zip file , and we will have the request link to download the zip file , it looks similar to this.

```
https://www.tooplate.com/zip-templates/2136\_kool\_form\_pack.zip
```

After downloading the website on the EC2 instance , we need to copy the contents for the website to /var/www/html

```
cp -r 2136_kool_form_pack/* /var/www/html/
```

7:08

## 19. AWS Part-2 217. Website in VPC



Later we need to restart the httpd , so that we can setup the website in the private subnet , and to reach this website to public , we gonna install load balancer on public subnet , and through this , we will run our operations to and fro from internet to the website.

```
1 | [root@ip-172-20-4-129 ]# systemctl restart httpd
2 | [root@ip-172-20-4-129 ]# systemctl enable httpd
3 | Created symlink from /etc/systemd/system/multi-user.target.wants/httpd.service to /usr/lib/systemd/system/httpd.service.
4 | [root@ip-172-20-4-129 ~]#
```

For creating a load balancer , we need to create a target group first , and while creating there , we should give the VPC for the target group. All the resources (private/public subnet) should be present under same VPC.

We also need to create security group for the load balancer , so create one. Even under there , we must select VProfile-VPC as our VPC. And as our website should be publicly accessible , we must be able to accept requests from anywhere of IPv4 and IPv6 addresses.

9:21

## 19. AWS Part-2 217. Website in VPC



Now , we need to create a load balancer. We will create an Application Load Balancer , where we need to select the correct VPC under network mapping section. Under the mappings , we must select our publicly available subnets , in both Availability zones , ensure , we not select private subnet in here , because , we want the load balancers to access through the internet , but that can't happen in private subnets. We need to select the load balancer security group , that we created. We will also redirect the http requests to the target group that we created. This target group is used by the load balancer , hence it is like load balancer listens on port 80 with the help of the target group.

10:15

## 19. AWS Part-2 217. Website in VPC



Earlier , we have created a rule for inbound of EC2 instance only through security group of bastion host through ssh , but we also need to create a similar rule for security group of load balancer , which listens on the port 80.

Now after all these configurations , we will be able to access the website through the DNS name of load balancer. If we are not able to , please check , did we give the inbound

rule access for security group of load balancer in the EC2 instance of private group , which is actually having the website.

DNS name , will be looking something like this ,

vpro-web-elb-401942659.us-west-1.elb.amazonaws.com

After successful completion of tests , we need to clear all the resources which we used to host the website.

2:15

## 19. AWS Part-2 218. Peering



We won't be having a single VPC architecture every time , sometimes , we will be having multiple VPC , for multiple reasons , one VPC for Website hosting and another for Database and something similar , at these situations , if we want to establish communication between these VPC's resources , we will do the VPC Peering.

While we create another VPC for the demonstration , we should ensure , we give another IPv4 CIDR , such that , we must not have any clash with another VPC CIDR. If we give the same CIDR , sometimes , two different resources from two different VPC , may have the same IP , and which is an error to be. For the VPC , that we created , we will be automatically gets created an routing table , re-name that table conveniently.

So , assume , we have created two VPC's under two regions , and the process of connecting those VPC's together is called VPC Peering. We need to declare it under "Peering Connections" option.

We can even peer two VPC's either under same account or the diff.

3:28

## 19. AWS Part-2 218. Peering



We can peer with VPC , which is under same/different account & same/different region , we need to provide the Acceptor VPC , and the Requester VPC is the local VPC of our account , and Acceptor can be the one of same/different account of same/different region.

We need to provide the Acceptor VPC & Requester VPC id's in Peering Connections. We will get a request on the other account / other region , for acceptance. And if we accept that , we would establish the peering connection.

6:40

## 19. AWS Part-2 218. Peering



We also need to establish connection between VPC's , by establishing a route to peering connections through Route tables under both VPC's. Under route-table of VPC1 of region 1 , we must give the IPv4 CIDR of VPC2 of region 2 and the target as the peering connection , it means whatever request that is forwarding to the address of VPC2 CIDR , should redirect to region 2 through peering connection , this will be same to VPC2 of region 2.

If we have nVPC's , we need to establish  $n!/2$  peering connections and  $n!$  routes between VPC's , we can also create less than that , but , if we wanted to make the every VPC to connect to every other VPC , this will be the case.

1:09

## 19. AWS Part-2 219. Ec2 Logs



For a website that we will be hosting thorough EC2 instance , we will be having an access\_log for each and every website , that will be present under the httpd folder at following location `/var/log/httpd/` with the name of `access_log` and we will be also having another log , which tracks the errors in `error_log` . While we run our website service , we can check this access\_log , so we can see all the actions of website through this log.

So , in the similar fashion , if we are having a website that is hosted for public , then we can think , how much log , we would generate , that will occupy our local disk , so instead of using local disk , we can archive these logs and store it somewhere , like S3 Bucket , and delete the log on the local disk.

3:43

## 19. AWS Part-2 219. Ec2 Logs



Now , we can archive , logs in the local directory , into one folder as

```
1 [root@ip-172-31-24-219 httpd]# ls
2 access_log error_log
3 [root@ip-172-31-24-219 httpd]# tar czvf wave-web01-httpdlogs19122020.tar.gz *
4 access_log
5 error_log
6 [root@ip-172-31-24-219 httpd]# ls
7 access_log error_log wave-web01-httpdlogs19122020.tar.gz
8 [root@ip-172-31-24-219 httpd]#
```

Now , store that archive , into a separate temporary folder , so we can map that folder to the S3 bucket.

```
1 [root@ip-172-31-24-219 httpd]# mkdir /tmp/logs-wave
2 [root@ip-172-31-24-219 httpd]# mv wave-web01-httpdlogs19122020.tar.gz /tmp/logs-
wave/
3 [root@ip-172-31-24-219 httpd]# ls
4 access_log error_log
5 [root@ip-172-31-24-219 httpd]# cat /dev/null > access_log
6 [root@ip-172-31-24-219 httpd]# cat /dev/null > error_log
```

5:05

## 19. AWS Part-2 219. Ec2 Logs



Now we gonna , install awscli on terminal , so that we can use S3 bucket on the terminal.

```
yum install awscli
```

If we want to manipulate the aws through the awscli , we need to configure terminal by providing the AWS Access Key ID

And to get that particular Access Key ID , we need to create an IAM User , we need to provide programmable access for it , it means it enables an access key ID and secret access key for AWS API , CLI , SDK and other development tools. We need to provide these details on the terminal , when terminal asks these details , when we are configuring terminal through `aws configure`

And for that IAM User , we can assign that user to a group , or copy permissions from an existing user or Attach existing policy for it , in here , we are attaching only S3 access for IAM User , as we need r/w access only for S3 , this rule is a pre-created rule , which we can take preset from the UI.

At the end of IAM User creation , we would be getting , Access Key ID and Secret access Key.

7:08

## 19. AWS Part-2 219. Ec2 Logs



We need to provide all these details in-order to configure aws in the terminal.

```
1 | [root@ip-172-31-24-219 httpd]# aws configure
2 | AWS Access Key ID [None]: AKIAIXXFJTQEWTB25Z
3 | AWS Secret Access Key [None]: YSwnBDzP1DHGuSpzYK5W6XZE3ziMhT440s58CGWF
4 | Default region name [None]: us-east-2
5 | Default output format [None]: json
```

Later , we can use aws commands to manipulate aws resources.

We use , following syntax , for copying data from source terminal to destination s3 storage ,

```
aws s3 cp <local-tar-file> s3://<destination-path>
```

If we wanted to keep a sync between source folder on the terminal to the destination folder of s3 , we need to use following command.

```
aws s3 sync <source-folder> s3://<destination-s3-folder-path>
```

```
aws s3 sync /tmp/logs-wave/ s3://wave-web-logs-321/
```

Whenever , we use the following sync command , we will copy only the files , which are present one side , but not on the other. If we add this command in cronjob , the source and s3 destination folder , will be always in sync.

10:57

## 19. AWS Part-2 219. Ec2 Logs



Consider there will be a call , where , we generated logs in the production server , we wanted to solve that issue with the help of developer , but we don't want to give him the production server credentials , and in such cases , if we wanted to give him realtime access for the logs , there is a service in AWS called CloudWatch logs , which helps us to monitor , analyze and store logs. It enables us to centralize logs from all our systems , applications and AWS Services in a single highly scalable , service.

12:53

## 19. AWS Part-2 219. Ec2 Logs



For an IAM User , we will be creating a role , such that , we should be having full S3 access and full Cloud Watch access , so we assign those roles to the IAM User.

As we have created a dedicated user for these operations , we can delete the earlier configuration that we made , by following `rm -rf .aws/credentials`

Now , our aws commands , won't work on the terminal , so , we need to assign our created role to the EC2 instance , this role have the access for both the S3 and cloud watch , hence , we would have access for `aws s3` commands

To install aws cloudwatch agent , use the following ,

```
yum install awslogs -y
```

17:24

## 19. AWS Part-2 219. Ec2 Logs



To streamline the logs to the cloud watch using awslogs , edit the following file  
`/etc/awslogs/awslogs.conf`

We need to enter the following , to streamline "messages" & "access\_log" to cloud\_watch

```
1 | [ /var/log/messages ]
2 | datetime_format = %b %d %H: %M : %S
3 | file = /var/log/messages
4 | buffer_duration = 5000
5 | log_stream_name = web01-sys-logs
6 | initial_position = start_of_file
7 | log_group_name = wave-web
8 |
9 | [ /var/log/httpd/access_log ]
```

```
10 |     datetime_format = %b %d %H : %M: %S
11 |     file = /var/log/httpd/access_log
12 |     buffer_duration = 5000
13 |     log_stream_name = web01-httpd-access
14 |     initial_position = start_of_file
15 |     log_group_name = wave-web
[ /var/log/messages ] => Configuration name
file => File that we want to stream to cloud watch
log_stream_name => Identification name for the log stream
log_group_name => Logical group under which our streams will be under.
```

After this, we need to restart and enable the awslogs daemon service.

```
1 | [root@ip-172-31-20-227 "]# systemctl restart awslogs
2 | [root@ip-172-31-20-227 ~]# systemctl enable awslogs
```

18:02

## 19. AWS Part-2 219. Ec2 Logs



These commands already started to stream the awslogs files to cloud watch. We need to go to Cloud Watch Service on AWS , There check the log groups section of it , under where , we can see the log group that we named 'wave-web' , under log streams of it , we see 'web01-sys-logs' that we created , in here , we can see the logs that are created under 'access-log'

We also need to do the similar change as [/etc/awslogs/awslogs.conf](#) in [/var/awslogs/etc/awslogs.conf](#) and restart awslogs service , using following command [service awslogs restart](#)

This is the main file to make a change and give access to the logs and streamline through cloud watch [/var/awslogs/etc/awslogs.conf](#)

Can export all these streamlined logs to Amazon S3 , can create elastic search & lambda subscription filter. We can also create some lambda functions.

We will be having an option create metric filter , where we can filter out the contents of the log , based on the text pattern / ERROR / INFO / WARNING / Status Code and create

22:16

## 19. AWS Part-2 219. Ec2 Logs



Based on the created metric filter , we can create an alarm by scanning logs in real time.

For every load balancer , we can enable access logs and store them in some s3 bucket in frequent interval , and we can scan those access logs to create a metric filter as discussed earlier.

But , if we wanted to send the access logs of a load balancer to the s3 bucket , we need to edit the S3 policy of S3 Storage account.

<https://docs.aws.amazon.com/elasticloadbalancing/latest/classic/enable-access-logs.html>

We need to provide the necessities in AWS S3 Bucket policy , in such a way to access the logs of classic load balancer. After setting up from the side of AWS S3 Bucket policy , we need to enable access of ELB's access logs , and push that to S3 storage account.

Now , we must access the logs of ELB and pushed to S3 storage account actively.

We need to provide details of <S3://S3Bucket/policy> details from the Load Balancer side.

0:27

## 25. Terraform Tutorial 287. Introduction and Exercise 1



Terraform:

Infrastructure as a Code

We can automate the infrastructure

We can manage the infrastructure through code

Our terraform , can define Infrastructure state , we can define all the resources state to be , so terraform monitor those resources to be in that desired state.

Ansible , puppet , chef are some of the peer automation tools.

2:03

## 25. Terraform Tutorial 287. Introduction and Exercise 1



Certainly! Let's explore the differences between **Terraform**, **Ansible**, **Puppet**, and **Chef**:

### 1. Terraform:

- **Purpose:** Terraform is primarily a **cloud infrastructure provisioning and deprovisioning tool**. It allows you to define your infrastructure as code (IaC) using HashiCorp Configuration Language (HCL).
- **Approach:** Terraform follows a **declarative approach**, where you specify the desired state of your infrastructure, and it handles the provisioning and management.
- **Strengths:** It excels at creating and managing cloud resources (like VMs, networks, storage) across various cloud providers.
- **Use Case:** Ideal for setting up and maintaining cloud infrastructure.
- **Community:** Active open-source community and well-supported downstream commercial products.

### 1. Ansible:

- **Purpose:** Ansible is an **IT automation platform** that focuses on configuration management, application deployment, and orchestration.
- **Approach:** Ansible uses **YAML-based playbooks** to configure systems. It is **agentless**, meaning it doesn't require an agent on managed nodes.
- **Strengths:** Known for its simplicity, ease of use, and strong support for cloud-based infrastructure.
- **Use Case:** Great for automating repetitive tasks, system updates, and network configuration.
- **Community:** Widely adopted with a large community.

### 1. Puppet:

- **Purpose:** Puppet is designed to manage **large and complex infrastructures**.
- **Approach:** It uses a **model-driven approach** with imperative task execution and a declarative language to define configurations.
- **Strengths:** Enforces consistency across many systems, offers strong reporting and monitoring capabilities.
- **Use Case:** Suitable for maintaining consistent configurations across a large number of servers.
- **Community:** Well-established community support.

### 1. Chef:

- **Purpose:** Chef specializes in **configuration management**.
- **Approach:** It follows a **convergent model**, where it ensures that the desired state is achieved by applying changes incrementally.
- **Strengths:** Offers flexibility and scalability, especially for complex infrastructures.
- **Use Case:** Useful for managing configurations, software installations, and system updates.
- **Community:** Strong community support.

In summary:

- Terraform focuses on **provisioning** cloud resources.

- **Ansible** emphasizes **orchestration** and is agentless.
- **Puppet** enforces consistency across large infrastructures.
- **Chef** excels in configuration management.

1:59

## 25. Terraform Tutorial 287. Introduction and Exercise 1



Using terraform , we can alter the infrastructure , where as using ansible and other configuration tools , we can only alter the software's under the infrastructure , but not modify the infrastructure itself.

We can use ansible with terraform , instead of ansible , we can use simple bash scripting to manage the software's under the infrastructure.

There is No Programming to do in Terraform , we can code through HCL , which is similar to JSON syntax. We have good documentation on how to work with Terraform.

In DevOps , we need to automate everything , we have to know , how to automate systems & cloud with ansible. How to do CI-CD through Jenkins & AWS , but to automate infrastructure itself , Terraform is the best tool to perform this , today in the market.

4:02

## 25. Terraform Tutorial 287. Introduction and Exercise 1



To Install Terraform , we need to download Terraform binary from its website on all OS. Store binary in exported PATH , If on Linux , the PATH will be `/usr/local/bin`

4:22

## 25. Terraform Tutorial 287. Introduction and Exercise 1



How to launch an EC2 instance , using terraform scripts

- \* We need an AWS account
- \* Create an IAM User with access keys
- \* Create a Terraform file to launch instance
- \* Run Terraform apply command

Then Terraform will launch our EC2 instance

4:56

## 25. Terraform Tutorial 287. Introduction and Exercise 1



We gonna perform some tasks , to test the terraform workability ,

Exercise:

- \* Write instance.tf file

- \* Launch instance through terraform file
- \* Make some changes to instance.tf file
- \* Reapply the changes

We can work with terraform through terraform CLI on windows , mac , Linux many other OS

We can also install terraform CLI through `choco` in Windows Powershell

```
choco install terraform
```

If we wanted to access AWS through terraform CLI , we need access key and secret key.

If we have those details , provide them to terraform CLI , through `aws configure` command.

```
1 | imran@LAPTOP-2J00K66A MINGW64
2 | $ aws configure
3 | AWS Access Key ID [ ****]: AKIAIXFJTQESN4RONS4
4 | AWS Secret Access Key [ ****]: ko0tdMBDm5cFA8uDDF+D7xL64GVn+q0VCHOBYQ15
5 | Default region name [us-east-1]: us-east-2
6 | Default output format [json]: json
```

9:48

## 25. Terraform Tutorial 287. Introduction and Exercise 1



```
1 | provider "aws" {
2 |     region = "us-east-2"
3 |     access_key = ""
4 |     secret_key = ""
5 | }
```

We need to provide the cloud service provider that we want to use , and the access\_key and secret\_key for accessing AWS , but we configured our terminal with the credentials , and giving the access\_key and secret\_key on the terraform code , which projects the credentials on a public repository is a risky one. So , we shouldn't add those in the code.

```
1 | provider "aws" {
2 |     region = "us-east-2"
3 | }
```

12:38

## 25. Terraform Tutorial 287. Introduction and Exercise 1



If we wanted to mention the resource that we wanted to use , we need to use the following syntax

```
1 | resource "aws_instance" "intro" {
2 | }
```

Above , we will create an aws instance of name "intro" , of type "aws\_instance". And for creating an instance , we need to provide an AMI ID. This is noting but an image-id. We

can also pass the details to create such as , instance\_type , availability\_zone , security group , we can also pass the key\_name to login to the instance by creating it first , or we can also create that key on the run , similarly security group.

```
1 provider "aws" {
2     region = "us-east-2"
3     access_key = ""
4     secret_key = ""
5 }
6
7 resource "aws_instance" "intro" {
8     ami = "ami-03657b56516ab7912"
9     instance_type = "t2.micro"
10    availability_zone = "us-east-2a"
11    key_name = "dove-key"
12    vpc_security_group_ids = ["sg-0780815f55104be8a"]
13    tags = {
14        Name = "Dove-Instance"
15    }
16 }
```

16:43

## 25. Terraform Tutorial 287. Introduction and Exercise 1



The first command to run our terraform script is `terraform init`

It's going to run our terraform script and check with our resource provider and download all the related plugins to work with it. After this , a hidden directory named as

`.terraform` will be created. Underneath we will have a plugins folder

`.terraform/plugins/`

Later , if we wanted to validate our terraform script syntactically , we can use `terraform validate` command. It will return if we are having any syntactical errors on our terraform configuration file.

If we wanted to rewrite our terraform configuration file and format that as look-able , we can use `terraform fmt`

We can apply `terraform plan` , and get the clear detail on what will happen , if we execute `terraform apply` , we will be able to see , all the other parameters that we can pass to our resource , so we can utilize them by editing the terraform file , else , we can know them for knowledgeable purpose , and edit them once the instance is created.

19:40

## 25. Terraform Tutorial 287. Introduction and Exercise 1



`terraform apply` apply all the changes or modifications that we given through terraform configuration files.

If we are able to see `terraform.tfstate` , it means , it consists the state details related to the terraform instance , which state is it , is it on or off , and many such details. We will also have a backup file for this state file as `terraform.tfstate.backup`

We can also destroy our created instance through terraform using [terraform destroy](#) command.

0:48

## 25. Terraform Tutorial 288. Exercise 2 - Variables



We can use [variables in terraform](#) , to move the sensitive information into files , which we will publish to public repositories , so if we create variables on the local system and pass those to these configuration files , as we don't push these variable files to public repo , our credentials will be safe.

And we can also use these variables , for values that changes frequently , and also used most in the configuration , for example those can be , AMI id , resource tags , keypairs etc. We can re-use our variable code , and use the same variable multiple places where it's required , instead of changing the value at every place , if we wanted to make a change , we can change that value commonly , if we have a variable file. We can re-use the same terraform configuration file for all the regions , as we can change the region dynamically through variables.

2:22

## 25. Terraform Tutorial 288. Exercise 2 - Variables



We will be having different patterns to declare terraform variables , for each and every 'provider' , 'vars' , 'terraform variables' , 'instance'

### providers.tf

```
1 | provider "aws" {  
2 |     region = var.REGION  
3 | }
```

### vars.tf

```
1 | variable AWS_ACCESS_KEY {}  
2 | variable AWS_SECRET_KEY {}  
3 | variable REGION {  
4 |     default = "us-west-1"  
5 | }  
6 | variable AMIS {  
7 |     type = "map"  
8 |     default {  
9 |         us-west-1 = "ami-06397100adf427136"  
10 |        us-west-2 = "ami-ao42f4d8"  
11 |     }  
12 | }
```

### terraform.tfvars

```
1 | AWS_ACCESS_KEY = ""  
2 | AWS_SECRET_KEY = ""
```

### instance.tf

```
1 | resource "aws_instance" "intro" {  
2 |     ami = var.AMIS[var.REGION]  
3 |     instance_type = "t2.micro"
```

5:03

## 25. Terraform Tutorial 288. Exercise 2 - Variables



Example 2 for Variables ,

### vars.tf

```

1  variable REGION {
2      default = "us-east-2"
3  }
4  variable ZONE1 {
5      default = "us-east-2a"
6  }
7  variable AMIS {
8      type = map
9      default = {
10         us-east-2 = "ami-03657b56516ab7912"
11         us-east-1 = "ami-0947d2ba12ee1ff75"
12     }
13 }
```

### providers.tf

```

1 provider "aws" {
2     region = var.REGION
3 }
```

### instance.tf

```

1 resource "aws_instance" "dove-inst" {
2     ami = var.AMIS[var.REGION]
3     instance_type = "t2.micro"
4     availability_zone = var.ZONE1
5     key_name = "new-dove"
6     vpc_security_group_ids = ["sg-0780815f55104be8a"]
7     tags = {
8         Name = "Dove-Instance"
9         Project = "Dove"
10    }
11 }
```

We can't change the public IP for any instance , unless delete it and re-create a new instance.

1:09

## 25. Terraform Tutorial 289. Exercise 3 - Provisioners



We can perform provisioning using terraform , we use provisioning to ,

- \* Build Custom Images with tools like packer
- \* Use standard Image and use provisioner to setup software's and files.
- => We can upload the files manually to the standard image and make customized image.

=> We can install repository softwares through yum package manager , using remote\_exec command , like "`remote_exec yum install sqllite`"

=> We can connect terraform with Ansible , Puppet and Chef. Puppet and Chef need dedicated client to use them , where as ansible don't require any client , as it doesn't have client master relationship. We can connect to ansible directly and write playbooks using remote\_exec.

1:31

## 25. Terraform Tutorial 289. Exercise 3 - Provisioners



If we wanted to provision , we need to provide provision connection details to the terraform , so it can connect to other resources such as Linux machine or windows machine through `remote_exec` or any other way. We have to provide the credentials to provisioner.

If we wanted to establish connection with linux machine using provisioner , we need to do the following , source follows local file directory , where as destination follows the ec2 instance path that we are targeting to. And if we are trying to establish , an ssh connection , we also will provide the username and the password for the instance , and we can also provide the private key pair to the provisioner , so we can login passwordless

```
1 |   provisioner "file" {
2 |     source = "files/test.conf"
3 |     destination = "/etc/test.conf"
4 |     connection {
5 |       type = "ssh"
6 |       user = "root"
7 |       password = var.root_password
8 |     }
9 | }
```

2:26

## 25. Terraform Tutorial 289. Exercise 3 - Provisioners



We can connect same in windows by,

```
1 |   provisioner "file" {
2 |     source = "conf/myapp.conf"
3 |     destination = "C:/App/myapp.conf"
4 |
5 |     connection {
6 |       type = "winrm"
7 |       user = "Administrator"
8 |       password = var.admin_password
9 |     }
10 | }
```

### More Provisioners

\* The `file provisioner` is used to copy files or directories

- \* `remote-exec` invokes a command/script on remote resource.
- \* `local-exec` provisioner invokes a local executable after a resource is created
- \* The `puppet provisioner` installs, configures and runs the Puppet agent on a remote resource.  
=> Supports both ssh and winrm type connections.
- \* The `chef provisioner` installs, configures and runs the Chef Client on a remote resource.  
=> Supports both ssh and winrm type connections.
- \* `Ansible` : run terraform, Output IP address, run playbook with local-exec

4:05

## 25. Terraform Tutorial 289. Exercise 3 - Provisioners



We can define , private key and public key and user names as variables , so we can re-use that later.

```

1 |   variable "PRIV_KEY_PATH" {
2 |     default = "infi-inst_key"
3 |   }
4 |   variable "PUB_KEY_PATH" {
5 |     default = "infi-inst_key.pub"
6 |   }
7 |   variable "USER" {
8 |     default = "ubuntu"
9 |   }

```

6:20

## 25. Terraform Tutorial 289. Exercise 3 - Provisioners



Basically , we know , public key is lock and the private key is the key , so if upload public key to the cloud resource and furthur upload to an instance this public key acts as a lock , as we created that public key on our local machine , we already having private key with us , which acts as a key. We can login to that instance from our local machine using that private key passwordless. To upload the public key to aws , we use "aws\_key\_pair" resource type , and upload our local public key to the cloud.

```

1 |   resource "aws_key_pair" "dove-key" {
2 |     key_name = "dovekey"
3 |     public_key = file("dovekey.pub")
4 |   }

```

And we can use that aws\_key\_pair upload to the aws\_instance , so we can login to that using our private key.

```

1 |   resource "aws_instance" "intro" {
2 |     ami = var.AMIS[var.REGION]
3 |     instance_type = "t2.micro"
4 |     availability_zone = var.ZONE1
5 |     key_name = aws_key_pair.dove-key.key_name

```

```
6   vpc_security_group_ids = ["sg-833e24fd"]
7 }
```

7:51

## 25. Terraform Tutorial 289. Exercise 3 - Provisioners



If we wanted to push a file from the local machine to host machine / destination machine , we need to use the file provisioner.

```
1 provisioner "file" {
2   source = "web.sh"
3   destination = "/tmp/web.sh"
4   connection {
5     user = var.USER
6     private_key = file(var.PRIV_KEY_PATH)
7     host = self.public_ip
8   }
9 }
```

And related variables , will be mentioned under `vars.tf`

```
1 variable "PRIV_KEY_PATH" {
2   default = "infi-inst_key"
3 }
4 variable "PUB_KEY_PATH" {
5   default = "infi-inst_key.pub"
6 }
7 variable "USER" {
8   default = "ubuntu"
9 }
```

Under the provisioner , it is important to note , we need to give the host machine IP address , so that , our local machine will redirect to that IP and try to login to that using the given username and private\_key file.

If you wanted to execute that script , you need to use `remote_exec` provisioner.

```
1 provisioner "remote-exec" {
2   inline = [
3     "chmod u+x /tmp/web.sh",
4     "sudo /tmp/web.sh"
5   ]
6 }
```

9:41

## 25. Terraform Tutorial 289. Exercise 3 - Provisioners



### End - to - End workflow:

- Generate key pair
- Write script
- Write providers.tf
- Write vars.tf
- Write instances.tf

- key pair resource
- aws\_instance resource
  - provisioners
    - file
    - remote-exec
- Apply changes.

11:04

## 25. Terraform Tutorial 289. Exercise 3 - Provisioners



```
vars.tf
1 variable REGION {
2     default = "us-east-2"
3 }
4 variable ZONE1 {
5     default = "us-east-2a"
6 }
7 variable AMIS {
8     type = map
9     default = {
10        us-east-2 = "ami-03657b56516ab7912"
11        us-east-1 = "ami-0947d2ba12ee1ff75"
12    }
13 }
```

  
**web.sh**

```
1 #!/bin/bash
2 yum install wget unzip httpd -y
3 systemctl start httpd
4 systemctl enable httpd
5 wget https://www.tooplate.com/zip-templates/2117_infinite_loop.zip
6 unzip -o 2117_infinite_loop.zip
7 cp -r 2117_infinite_loop/* /var/www/html/
8 systemctl restart httpd
```

  
**providers.tf**

```
1 provider "aws" {
2     region = var.REGION
3 }
```

16:41

## 25. Terraform Tutorial 289. Exercise 3 - Provisioners



```
instance.tf
1 resource "aws_key_pair" "dove-key" {
2     key_name = "dovekey"
3     public_key = file("dovekey.pub")
4 }
5 resource "aws_instance" "dove-inst" {
6     ami = var.AMIS[var.REGION]
```

```

7   instance_type = "t2.micro"
8   availability_zone = var.ZONE1
9   key_name = aws_key_pair.dove-key.key_name
10  vpc_security_group_ids = ["sg-0780815f55104be8a"]
11  tags = {
12    Name = "Dove-Instance"
13    Project = "Dove"
14  }
15
16  provisioner "file" {
17    source = "web.sh"
18    destination = "/tmp/web.sh"
19  }
20
21  provisioner "remote-exec" {
22    inline = [
23      "chmod u+x /tmp/web.sh",
24      "sudo /tmp/web.sh"
25    ]
26  }
27
28  connection {
29    user = var.USER
30    private_key = file("dovekey")
31    host = self.public_ip
32  }
33}

```

Later to apply these changes , apply below standard commands , later apply

```

1  terraform init
2  terraform validate
3  terraform fmt
4  terraform plan
5  terraform apply

```

If we wanted to delete all the resources that we created,

**terraform destroy**

22:02

25. Terraform Tutorial 289. Exercise 3 - Provisioners



```

1  resource "aws_key_pair" "dove-key" {
2    key_name = "dovekey"
3    public_key = file("dovekey.pub")
4  }

```

In here dove-key is the resource name that we are going to create , we use resource-name to refer among terraform resources , where as the actual key name is mentioned under 'key\_name' attribute , if we go and check in aws , the key\_pair will be created on name "dovekey" which is present under attribute section of dove-key resource.

0:43

25. Terraform Tutorial 290. Exercise 4 - Output



If we wanted to output instance details , after it is created , we can use these output blocks under instance file. In here , we are ordering , terraform to print **PublicIP** and **PrivateIP** after the instance is created , which is triggered by **terraform apply** . Can use **local-exec** to save the resources info to this output file.

```
1 |   output "PublicIP" {
2 |     value = aws_instance.dove-inst.public_ip
3 |   }
4 |   output "PrivateIP" {
5 |     value = aws_instance.dove-inst.private_ip
6 |   }
```

These are mentioned under **instance.tf** file

We will get output , something similar to this ,

```
1 | Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
2 |
3 | Outputs:
4 |
5 | PrivateIP = 172. 31. 1. 145
6 | PublicIP = 18. 189. 145. 9
```

2:51

## 25. Terraform Tutorial 291. Exercise 5 - Backend



For the person , who uses terraform , he will have his own **terraform.tfstate** file , so consider cases , you should have one state file for *1 production server , 1 development server , 1 QA server* , and that state file , should be sync with all the people who are working under that category.

In such cases , we want **S3 Bucket** , which we use as external storage , which is common storage for all the employees , who are working in that environment. We need to give that S3 bucket and folder that we want to upload to , details to terraform through **backend.tf** file as following.

### backend.tf

```
1 |   terraform {
2 |     backend "s3" {
3 |       bucket = "terra-state-dove"
4 |       key = "terraform/backend"
5 |       region = "us-east-2"
6 |     }
7 |   }
```

s3 => type of backend resource

bucket => s3 bucket name

key => file path

To apply these , use series of commands to apply ,

```
1 |   terraform init  
2 |   terraform validate  
3 |   terraform fmt  
4 |   terraform plan  
5 |   terraform apply
```

Now , resources will be created

0:30

## 25. Terraform Tutorial 292. Exercise 6 - Multi Resource



Some of the featured providers of Terraform are ,

AWS , Azure , Google Cloud Platform , Kubernetes , Oracle Cloud Infrastructure , Alibaba Cloud

Check in here , to get the code snippets , on how to use terraform on different cloud providers.

<https://registry.terraform.io/>

For dedicated AWS:

<https://registry.terraform.io/providers/hashicorp/aws/latest/docs>

We will be having details on how to create all cloud resources , using terraform in this documentation , we can use this as reference to write code.

4:20

## 25. Terraform Tutorial 292. Exercise 6 - Multi Resource



To create a VPC , we create following resources through terraform code ,

### vars.tf

```
1 | variable REGION {  
2 |     default = "us-east-2"  
3 | }  
4 |  
5 | variable ZONE1 {  
6 |     default = "us-east-2a"  
7 | }  
8 |  
9 | variable ZONE2 {  
10|     default = "us-east-2b"  
11| }  
12 |  
13| variable ZONE3 {  
14|     default = "us-east-2c"  
15| }  
16 |  
17| variable AMIS {  
18|     type = map  
19|     default = {  
20|         us-east-2 = "ami-03657b56516ab7912"  
21|         us-east-1 = "ami-0947d2ba12ee1ff75"
```

```

22     }
23 }
24
25 variable USER {
26     default = "ec2-user"
27 }
28
29 variable USER {
30     default = "ec2-user"
31 }
32
33 variable PUB_KEY {
34     default = "dovekey.pub"
35 }
36
37 variable PRIV_KEY {
38     default = "dovekey"
39 }

```

### providers.tf

```

1 provider "aws" {
2     region = var.REGION
3 }

```

7:35

25. Terraform Tutorial 292. Exercise 6 - Multi Resource



We gonna use previously created files , and create another file `vpc.tf` , which creates vpc's and subnets under it.

```

1 resource "aws_vpc" "dove" {
2     cidr_block = "10.0.0.0/16"
3     instance_tenancy = "default"
4     enable_dns_support = "true"
5     enable_dns_hostnames = "true"
6     tags = {
7         Name = "dove-vpc"
8     }
9 }
10
11 resource "aws_subnet" "dove-pub-1" {
12     vpc_id = aws_vpc.dove.id
13     cidr_block = "10.0.1.0/24"
14     map_public_ip_on_launch = "true"
15     availability_zone = var.ZONE1
16     tags = {
17         Name = "dove-pub-1"
18     }
19 }
20
21 resource "aws_subnet" "dove-pub-2" {
22     vpc_id = aws_vpc.dove.id
23     cidr_block = "10.0.2.0/24"
24     map_public_ip_on_launch = "true"

```

```

25   availability_zone = var.ZONE2
26   tags = {
27     Name = "dove-pub-2"
28   }
29 }
30
31 resource "aws_subnet" "dove-pub-3" {
32   vpc_id = aws_vpc.dove.id
33   cidr_block= "10.0.3.0/24"
34   map_public_ip_on_launch = "true"
35   availability_zone = var.ZONE3
36   tags = {
37     Name = "dove-pub-3"
38   }
39 }

```

8:12

## 25. Terraform Tutorial 292. Exercise 6 - Multi Resource



If we are creating multiple subnets under same VPC , we must ensure to not having any two subnets a same cidr\_block address. Earlier , we have created public subnets , now we should create private subnets.

```

1  resource "aws_subnet" "dove-priv-1" {
2    vpc_id = aws_vpc.dove.id
3    cidr_block = "10.0.4.0/24"
4    map_public_ip_on_launch = "true"
5    availability_zone = var.ZONE1
6    tags = {
7      Name = "dove-priv-1"
8    }
9  }
10
11 resource "aws_subnet" "dove-priv-2" {
12   vpc_id = aws_vpc.dove.id
13   cidr_block = "10.0.5.0/24"
14   map_public_ip_on_launch = "true"
15   availability_zone = var.ZONE2
16   tags = {
17     Name = "dove-priv-2"
18   }
19 }
20
21 resource "aws_subnet" "dove-priv-3" {
22   vpc_id = aws_vpc.dove.id
23   cidr_block = "10.0.6.0/24"
24   map_public_ip_on_launch = "true"
25   availability_zone = var.ZONE3
26   tags = {
27     Name = "dove-priv-3"
28   }
29 }
30

```

```
31   resource "aws_internet_gateway" "dove-IGW" {
32     vpc_id = aws_vpc.dove.id
33     tags = {
34       Name = "dove-IGW"
35     }
36   }
```

9:51

## 25. Terraform Tutorial 292. Exercise 6 - Multi Resource



Now , we created all the resources , we need to connect those resources , if we wanted to connect public subnets and internet gateway , you need to create a route table. We are mentioning below , to route all the traffic which is going to any IP address through route\_table.

```
1   resource "aws_route_table" "dove-pub-RT" {
2     vpc_id = aws_vpc.dove.id
3     route {
4       cidr_block = "0.0.0.0/0"
5       gateway_id = aws_internet_gateway.dove-IGW.id
6     }
7     tags = {
8       Name = "dove-pub-RT"
9     }
10  }
```

We have created a route table , now we need to make entries in that , we make entries using **route\_table\_association** , as we have three subnets , we need to make 3 route\_table\_associations.

10:43

## 25. Terraform Tutorial 292. Exercise 6 - Multi Resource



### route table associations

```
1   resource "aws_route_table_association" "dove-pub-1-a" {
2     subnet_id = aws_subnet.dove-pub-1.id
3     route_table_id = aws_route_table.dove-pub-RT.id
4   }
5   resource "aws_route_table_association" "dove-pub-2-a" {
6     subnet_id = aws_subnet.dove-pub-2.id
7     route_table_id = aws_route_table.dove-pub-RT.id
8   }
9   resource "aws_route_table_association" "dove-pub-3-a" {
10    subnet_id = aws_subnet.dove-pub-3.id
11    route_table_id = aws_route_table.dove-pub-RT.id
12  }
```

In here , we have three public subnets , if there is any request from these public subnets , route\_table\_associations redirect the request to the route\_table , and route\_table redirects request to internet gateway.

We need to have a security group , for the VPC , so create one , for which , we are having all open outgress rules , where as ingress rule only from MYIP , it mean , VPC accepts only requests from My IP address.

### Security Group ( secgrp.tf )

```

1 | resource "aws_security_group" "dove_stack_sg" {
2 |   vpc_id = aws_vpc.dove.id
3 |   name = "dove-stack-sg"
4 |   description = "Sec Grp for dove ssh"
5 |   egress {
6 |     from_port = 0
7 |     to_port = 0
8 |     protocol = "-1"
9 |     cidr_blocks = ["0.0.0.0/0"]
10 |
11 }
12
13   ingress {
14     from_port = 22
15     to_port = 22
16     protocol = "tcp"
17     cidr_blocks = [var.MYIP]
18   }
19
20   tags = {
21     Name = "allow-ssh"
22   }
23 }
```

And variable as ,

```

1 | variable MYIP {
2 |   default = "183.83.39.203/32"
3 | }
```

We also wanted to store the terraform.tfstate file on the S3 bucket , so doing the following ,

```

1 | terraform {
2 |   backend "s3" {
3 |     bucket = "terra-state-dove"
4 |     key = "terraform/backend_exercise6"
5 |     region = "us-east-2"
6 |   }
7 | }
```

For provisioning , we need the shell script that we have used earlier ,

### web.sh

```

1 | #!/bin/bash
2 | yum install wget unzip httpd -y
```

```
3 systemctl start httpd
4 systemctl enable httpd
5 wget https://www.tooplate.com/zip-templates/2117_infiniteLoop.zip
6 unzip -o 2117_infiniteLoop.zip
7 cp -r 2117_infiniteLoop/* /var/www/html/
8 systemctl restart httpd
```

19:56

## 25. Terraform Tutorial 292. Exercise 6 - Multi Resource



### instance.tf

```
1 resource "aws_key_pair" "dove-key" {
2     key_name = "dovekey"
3     public_key = file(var.PUB_KEY)
4 }
5 resource "aws_instance" "dove-web" {
6     ami = var.AMIS[var.REGION]
7     instance_type "t2.micro"
8     subnet_id = aws_subnet.dove-pub-1.id
9     key_name = aws_key_pair.dove-key.key_name
10    vpc_security_group_ids = [aws_security_group.dove_stack_sg.id]
11    tags = {
12        Name = "my-dove"
13    }
14 }
15 resource "aws_ebs_volume" "vol_4_dove" {
16     availability_zone = var.ZONE1
17     size = 3
18     tags = {
19         Name = "extr-vol-4-dove"
20     }
21 }
22 resource "aws_volume_attachment" "atch_vol_dove" {
23     device_name = "/dev/xvdh"
24     volume_id = aws_ebs_volume.vol_4_dove.id
25     instance_id = aws_instance.dove-web.id
26 }
27 output "PublicIP" {
28     value = aws_instance.dove-web.public_ip
29 }
```

20:50

## 25. Terraform Tutorial 292. Exercise 6 - Multi Resource



In here , in ebs\_volume , we can see the size , and we are mentioning '3' in there , 3 mean 3GB , we are also creating **volume\_attachment** resource , which binds the volume with the instance , we are declaring the file type under device\_name. We can declare output variables in here , for example PublicIP is an output , after **terraform apply** , after creating all the resources , we will print all the mentioned output variables.

### Setting up Amazon EKS Cluster using Terraform

We can set-up kubernetes using kubemd or cops , or we can use Amazon EKS Service by amazon. After creating Amazon EKS , we will be getting kube.config file.

If we create our kubernetes cluster using kubemd or cops , we must have to admin by ourself , which will be quite a difficult task. Where as if we have create our Kubernetes using Amazon EKS , it will be well managed with the help of services that amazon provides.

We need to provide the cluster details and the node groups details to AWS , so that Amazon EKS service will create the kubernetes cluster , we can also upscale or downscale the resources whenever its required.

### Amazon EKS => Amazon Elastic Kubernetes Service

#### Key Features:

- \* Reduce costs with efficient compute resource provisioning and automatic Kubernetes application scaling.
- \* Ensure a more secure Kubernetes environment with security patches automatically applied to your cluster's control plane.
- \* Leverage built-in integrations with AWS services such as EC2, VPC, IAM, EBS and more

Amazon EKS , is an expensive service

All the kubectl commands , works same in here with Amazon EKS. First of all , we need to deploy EKS Cluster in Kubernetes , later we need to deploy worker nodes by adding to EKS Cluster. For the created EKS Cluster , we can use our kubectl commands to manipulate. Now , we can deploy our Kubernetes applications on our EKS cluster.

EKS also needs VPC , Now , we gonna make a Kubernetes cluster using terraform. We gonna create VPC using terraform , simply using terraform modules , terraform modules are written by someone , to do some process , that we can use directly , using that module , it's like a pre-defined function , which is defined by the module provider.

You can see those pre-defined modules in here ,

If you open your required module , you can see the usage instructions.

4:53

## 25. Terraform Tutorial 293. AWS Elastic Kubernetes Service



Code of the module will be looking something like this ,

```
1  module "vpc" {
2      source = "terraform-aws-modules/vpc/aws"
3
4      name = "my-vpc"
5      cidr = "10.0.0.0/16"
6
7      azs = ["eu-west-1a", "eu-west-1b", "eu-west-1c"]
8      private_subnets = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]
9      public_subnets = ["10.0.101.0/24", "10.0.102.0/24", "10.0.103.0/24"]
10
11     enableTnat_gateway = true
12     enable_vpn_gateway = true
13
14     tags = {
15         Terraform = "true"
16         Environment = "dev"
17     }
18 }
```

We need to pass all the required values , to the module , then module will create the resource.

We can have many inputs to the modules , so we can pass those to the module and create the resource through the module.

9:00

## 25. Terraform Tutorial 293. AWS Elastic Kubernetes Service



### terraform.tf

```
1  terraform {
2      required_providers {
3          aws = {
4              source = "hashicorp/aws"
5              version = "~> 4.46.0"
6          }
7
8          random = {
9              source = "hashicorp/random"
10             version = "~> 3.4.3"
11         }
12
13         tls = {
14             source = "hashicorp/tls"
15             version = "~> 4.0.4"
16         }
17     }
18 }
```

```

16    }
17
18    cloudinit = {
19      source = "hashicorp/cloudinit"
20      version = "~> 2.2.0"
21    }
22
23    kubernetes = {
24      source = "hashicorp/kubernetes"
25      version = "~> 2.16.1"
26    }
27  }
28
29  backend "s3" {
30    bucket          = "terra-eks12"
31    key             = "state/terraform.tfstate"
32    region          = "us-east-1"
33  }
34
35  required_version = "~> 1.3"
36 }

```

We mention all the dependencies under this file , under required providers , where as , we also need to create an S3 Bucket , to store our terraform state file. So , mentioning the details in here.

Under the s3 bucket that is created , we are creating another folder called state , under which we will be storing a state file.

10:30

**25. Terraform Tutorial** 293. AWS Elastic Kubernetes Service



### variables.tf

```

1  variable "region" {
2    description = "AWS region"
3    type        = string
4    default     = "us-east-1"
5  }
6
7  variable "clusterName" {
8    description = "Name of the EKS cluster"
9    type        = string
10   default    = "vpro-eks"
11 }

```

11:34

**25. Terraform Tutorial** 293. AWS Elastic Kubernetes Service



### main.tf

```

1 provider "kubernetes" {

```

```

2   host = module.eks.cluster_endpoint
3   cluster_ca_certificate =
4     base64decode(module.eks.cluster_certificate_authority_data)
5   }
6
7   provider "aws" {
8     region = var.region
9   }
10
11
12   locals {
13     cluster_name = var.clusterName
14   }

```

In this file , we will give the details for the provider , under kubernetes , we need to give details like host & cluster\_ca\_certificate , but those details are present only once , if we have created eks cluster thorugh eks module. And region for aws provider we are talking the value of variable thorugh var.tf file.

We also wanted to know all the availability zones under a region , as we have declared our aws region in here , we wanted to know all the available availability zones , so using the following line.

```
1 | data "aws_availability_zones" "available" {}
```

In here , we will get list of available availability zones under mentioned region "var.region".

12:58

## 25. Terraform Tutorial 293. AWS Elastic Kubernetes Service



We will be using the list , which we used to store list of availability zones under here, as `data.aws_availability_zones.available.names`

### vpc.tf

```

1  module "vpc" {
2    source = "terraform-aws-modules/vpc/aws"
3    version = "3.14.2"
4
5    name = "vprofile-eks"
6
7    cidr = "172.20.0.0/16"
8    azs = slice(data.aws_availability_zones.available.names, 0, 3)
9
10   private_subnets = ["172.20.1.0/24", "172.20.2.0/24", "172.20.3.0/24"]
11   public_subnets = ["172.20.4.0/24", "172.20.5.0/24", "172.20.6.0/24"]
12
13   enable_nat_gateway = true
14   single_nat_gateway = true
15   enable_dns_hostnames = true
16
17   public_subnet_tags = {
18     "kubernetes.io/cluster/${local.cluster_name}" = "shared"

```

```
19     "kubernetes.io/role/elb" = 1
20   }
21
22   private_subnet_tags = {
23     "kubernetes.io/cluster/${local.cluster_name}" = "shared"
24     "kubernetes.io/role/internal-elb" = 1
25   }
26 }
```

13:26

## 25. Terraform Tutorial 293. AWS Elastic Kubernetes Service



We need to pass private and public subnet details to the VPC , while we are creating , we can create it prior and assign those IP's to the VPC in the code. Or we can also create private and public subnets through terraform code and use them in here. Here , we are doing the first way.

Here , we are having two attributes , `enable_nat_gateway` and `single_nat_gateway` , if we set `enable_nat_gateway` to true , we will be having 1 `nat_gateway` for every `private_subnet` , as we want only one `nat_gateway` for all the 3 private subnet's , we are enabling the `single_nat_gateway` to true.

We no need to bihard these , we can find all these details on the documentation.

17:03

## 25. Terraform Tutorial 293. AWS Elastic Kubernetes Service



We are creating eks cluster using the following code ,

### eks-cluster.tf

```
1 module "eks" {
2   source = "terraform-aws-modules/eks/aws"
3   version = "19.0.4"
4
5   cluster_name = local.cluster_name
6   cluster_version = "1.27"
7
8   vpc_id = module.vpc.vpc_id
9   subnet_ids = module.vpc.private_subnets
10  cluster_endpoint_public_access = true
11
12  eks_managed_node_group_defaults = {
13    ami_type = "AL2_x86_64"
14
15  }
16
17  eks_managed_node_groups = {
18    one = {
```

```

19     name = "node-group-1"
20
21     instance_types = ["t3.small"]
22
23     min_size = 1
24     max_size = 3
25     desired_size = 2
26   }
27
28   two = {
29     name = "node-group-2"
30
31     instance_types = ["t3.small"]
32
33     min_size = 1
34     max_size = 2
35     desired_size = 1
36   }
37 }
38 }
```

We are creating these modules , and we can access these module's attributes using ,

`module.<module_name>.<attribute>`

For creating , eks , we need to pass all the details of vpc , subnets , and worker nodes.

17:53

## 25. Terraform Tutorial 293. AWS Elastic Kubernetes Service



We need to pass these details , if we wanted to **configure AWS on our Linux user**,

```

1 imran@LAPTOP-2J00K66A MINGW64
2 $ aws configure
3 AWS Access Key ID [ ****]: CX6S]:
4 AWS Secret Access Key [ ****]: 4pcC]:
5 Default region name [us-east-1]:
6 Default output format [json]:
```

22:54

## 25. Terraform Tutorial 293. AWS Elastic Kubernetes Service



If we wanted to generate a kube config file , we need to use the following command ,

```

1 imran@LAPTOP-2J00K66A MINGW64 /f/terraform-eks/vprofile-project (terraform-eks)
2 $ aws eks update-kubeconfig --region us-east-1 --name vprof-eks
3 Added new context arn:aws:eks:us-east-1:656296030910:cluster/vprof-eks to
C:\Users\imran\.kube\config
```

By using this kube config file , kubectl can performs actions , kube config stores the state of Kubernetes cluster.

We can install kubectl through windows PowerShell using choco ,

```
choco install kubernetes-cli -y
```

0:09

## 18. Ansible 188. Introduction



**Ansible** is the most popular devops tool today.

First of all , in the history of automation , first **Bash Scripting/Batch Scripting** was evolved to automate the OS related tasks , later they have adapted to use **Python / PERL / Ruby** languages after cloud providers got entered. Bash Scripting is for Linux and Batch Scripting is for Windows , later stages , **PowerShell** got introduced , it's basically for Windows , where it also automate the VMware's. And for configuration management , they have introduced the Puppet language , this is the first time a language got introduced for configuration management. Puppet is not considered as automation tool initially , it is considered as configuration management tool. Configuration state across the infrastructure is managed through Puppet server.

3:24

## 18. Ansible 188. Introduction



There will be cases , where we will be having set of servers , where a single configuration needs to be executed , in those cases , if there is no configuration management tool , one of the server will be outdated , results sometime to affect the infrastructure. So , to avoid this , we have configuration management tool like Puppet.

Similar to Puppet , we have Salt Stack , where we can use that to execute commands on remote machines. We typically run Puppet Agents through Salt Stack , Puppet manages infrastructure configuration on a single server , where each server will have puppet agent. Where as through Salt Stack we can execute commands on remote machines , we can execute these Puppet agents through Salt Stacks remotely and manipulate infrastructure configuration.

**Puppet** is built on Ruby , we pass DSL commands to use Puppet.

**Salt Stack** on Python.

**Chef** is a complex infrastructure management tool , where we can also execute our code in there , it have GUI.

4:59

## 18. Ansible 188. Introduction



Later another configuration management tool , is created based on Python , that is **Ansible**. This is the simplest configuration management tool , among all those previously discussed. Mostly designed for Linux machine automation , later supported Windows machine automation. Cloud automation , Networking automation , Database automation and there are so many integrations came with Ansible.

And later there comes another automation tool called **Terraform** , which is mainly focused on Cloud automation , which is simplest than Ansible.

6:28

18. Ansible 188. Introduction



### Use Cases of Ansible:

**Automation:** We can perform any automation , either it can be Linux or windows. We can set up , web-services , database-services , configuration , start the services and restart the services , these kinds of automation , we can perform through Ansible.

**Change Management:** We can manage our production machines. Any changes to production servers , we can do that through Ansible.

**Provisioning:** If we wanted to set up anything from the scratch , we can use Ansible to launch cloud resources , as said Ansible can perform cloud automation. We can set-up services on them , frontend , backend , database , webservices , complete provisioning , we can do through Ansible.

**Orchestration:** We can combine any two large automation , and orchestrate them to execute them in a order , we can integrate Ansible with Jenkins or any other continuous integration tools or cloud services we can integrate with Ansible.

8:20

18. Ansible 188. Introduction



Ansible is very simple , it doesn't install agents on destination servers , where as chef , puppet have their agents on destination servers , so whenever , we want to manipulate infrastructure , puppet use those agents to install the softwares on those servers , where as ansible doesn't have any agents on the destination servers , ansible target those machines or access those services by SSH , winrm & API's. There are no inbuilt databases for ansible , we write all the configurations through YAML ,INI & texts and output is written in json format. Setting up of Ansible is also easy , its just a python library or we can install Ansible with the help of any package manager. We typically write playbooks and that gonna generate python code using that , we execute that python code and return output. So , there is no separate or residual softwares installed to run ansible.

9:36

18. Ansible 188. Introduction



**Ansible playbooks** are written in yaml format , no additional programming is required to write the script , yaml is structured method of declaration. Ansible has lots of modules with that , it also have modules which are API based. Suppose , if we wanted to create any EC2 instance through ansible , we can use the API's which helps for cloud resource integration through ansible. We can also run shell commands through ansible.

### How ansible can connect to its targets:

Target can be Linux server , windows server , cloud account , database , network equipment.

For Linux it uses SSH

For Windows it uses WINRM

For Cloud it uses API's

Ansible can connect to switches , routers , and databases , for connecting to the database , Ansible uses python packages.

Puppet , Chef and Salt Stack are server oriented , they need a server to operate on , they need a master node to operate , where as ansible is a python module , which helps us to connect to different services with help of some utilities.

12:09

## 18. Ansible 188. Introduction



There is no master and slave concept in Ansible , there is no agent on the destination machine. Ansible just execute the scripts of its playbook on the destination machine and returns the output. For other services other than the cloud , it does the remote execution , where as the cloud services , all the requests it process in the local , and send the request to the cloud to provide the resources.

13:01

## 18. Ansible 188. Introduction



### **Ansible Architecture:**

Ansible Playbooks takes its main inputs from Inventory file and the modules. In Inventory file , we will be having information about the target machine , like IP address , username and passwords. Ansible have lots of inbuilt modules , that are supposed to do particular task , may be it's installing a package or restarting the service or taking snapshot of EBS volume , similarly , we are having hundreds of modules.

We use these host and destination details provided by Inventory and modules , to create a Ansible Configuration using yaml , and later create python packages and python modules using these. These packages and modules will be executed on destination servers , either it can be cloud or non cloud resource.

### **Power of Ansible:**

Ansible can replace almost every automation tool in DevOps or Ops work.

0:11

## 18. Ansible 189. Setup Ansible & Infra



In this session , we are going to set up Ansible Infrastructure , consists of one EC2 instance with ansible , with other three EC2 instances with CentOS operating system , among those two are installed with web-server , one with DB Server. The EC2 instance

with ansible is called control machine , in generic terms. We gonna set-up on web and DB server through Ansible.

We are creating an instance for Ansible based on ubuntu machine , where , we need to provide the details of the security group , we need to login to this instance through my local machine , so assigning security group such a way that allow all requests coming from My IP address to the port 22. Don't forget to download the key-pair file , which helps us to login to our running instance.

Later , we also need to create three more instances for the web-server , which runs on the CentOS , don't forget to download the key-pair. We can launch 3 instances at time with same configuration.

3:29

## 18. Ansible 189. Setup Ansible & Infra



### Security Group rules for 3 instances:

Need to allow all requests coming from MyIP to port 22

Need to allow all requests coming from control machine to the port 22

Here , we are giving access to port 22 for these , because these target machines want's to target these 3 instances through ssh.

We are renaming those three instances as vprofile-web01 & vprofile-web02 & vprofile-db01 , now we can install programs on these instances through control machine instance.

We can see all the ssh machines that we have login so far through `cat`

`~/.ssh/known_hosts` , we can login to the control machine using ssh through

`ssh -i Downloads/control.pem ubuntu@35.88.145.184`

8:23

## 18. Ansible 189. Setup Ansible & Infra



Ansible is basically a python package , in here , we will install ansible in Ubuntu through the following commands.

```
1 | $ sudo apt update
2 | $ sudo apt install software-properties-common
3 | $ sudo add-apt-repository -- yes -- update ppa:ansible/ansible
4 | $ sudo apt install ansible
```

In third line , we are adding ansible repository , so that we can install ansible with help of that repository.

Every Ansible module would require a different python module , so it's not the source python version that needs to be cautious , it's the destination server , where our ansible code run it's code. The destination server's software's should match with the code , so it could execute flawlessly.

0:52

## 18. Ansible 190. Inventory & Ping Module



Control machine communicate with target machines through SSH.

Inventory files provide target machine details to Ansible.

Some of such details are IP Address , username , password , login key , port number etc.

We can write inventory files in two format's , one is ini file format and other is yaml format.

Default location for inventory files in ansible is `/etc/ansible/hosts` , If we didn't mention any location for the inventory , ansible fetches , inventory file at this location `/etc/ansible/hosts` , Its preferable to store inventory files in our repository , rather than the default location.

You can specify a different inventory file at the command line using the `-i <path>` option or in

configuration using `inventory` .

3:07

## 18. Ansible 190. Inventory & Ping Module



**basic INI** `/etc/ansible/hosts` might look like this:

```
1 | mail.example.com
2 |
3 | [webservers]
4 | foo.example.com
5 | bar.example.com
6 |
7 | [dbservers]
8 | one.example.com
9 | two.example.com
10| three.example.com
```

The headings in brackets are group names, which are used in classifying hosts and deciding what hosts you are controlling at what times and for what purpose. Group names should follow the same guidelines as Creating valid variable names.

**basic inventory file in YAML format:**

```
1 | all:
2 |   hosts:
3 |     mail.example.com:
4 |   children:
```

```
5    webservers:
6        hosts:
7            foo.example.com:
8            bar.example.com:
9        dbservers:
10       hosts:
11         one.example.com:
12         two.example.com:
13         three.example.com:
```

3:29

## 18. Ansible 190. Inventory & Ping Module



### **Ansible Variables:**

#### **ansible\_connection**

Connection type to the host. This can be the name of any of ansible's connection plugins. SSH protocol types are smart , ssh Or paramiko . The default is smart. Non-SSH based types are described in the next section.

### **General for all connections:**

#### **ansible\_host**

The name of the host to connect to, if different from the alias you wish to give to it.

#### **ansible\_port**

The connection port number, if not the default (22 for ssh)

#### **ansible\_user**

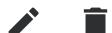
The user name to use when connecting to the host

#### **ansible\_password**

The password to use to authenticate to the host (never store this variable in plain text; always use a vault).

4:01

## 18. Ansible 190. Inventory & Ping Module



Specific to the SSH connection:

#### **ansible\_ssh\_private\_key\_file**

Private key file used by SSH. Useful if using multiple keys and you do not want to use SSH agent.

Please check this link for rest of the variables:

9:40

## 18. Ansible 190. Inventory & Ping Module



If we wanted to connect control machine to target machine using ansible , we need an inventory file for that ,

### inventory

```
1 | ubuntu@ip-172-31-24-198 :~ /vprofile/exercise1$ cat inventory
2 | all:
3 |     hosts:
4 |         web01:
5 |             ansible_host: 172. 31. 31. 178
6 |             ansible_user: ec2-user
7 |             ansible_ssh_private_key_file: clientkey.pem
```

We are mentioning the host details in here , by providing the hostname and giving details of the host such as hostname , username , private\_key\_file. And if we are executing ansible commands , we need to pass the path of the inventory file , so it can fetch the host details using the inventory file.

clientkey.pem is the key-pair file for the target server to login by the control machine.

9:43

## 18. Ansible 190. Inventory & Ping Module



We can connect to a host through ansible , by using following command ,

```
ansible web01 -m ping -i inventory
```

web01 is the hostname , which is mentioned in the inventory file

-m denotes module that needs to be executed , in here , we are executing ping module

-i denotes path of inventory file

While we connect to a target machine from a new machine for the first time , it needs host key verification. You can see some text like

```
1 | ubuntu@ip-172-31-24-198: /vprofile/exercise1$ ansible web01 -m ping -i inventory
2 | The authenticity of host '172. 31. 31. 178 (172. 31. 31. 178)' can't be
   established.
3 | ED25519 key fingerprint is SHA256: iyiDDGANeGc/9SQZRP+UFdpnrE11wXG++iCf/1AawSY.
4 | This key is not known by any other names
5 | Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

So , to by pass this , we need to disable the host key verification through ansible configuration file.

Basically ansible configuration file , will be present under `/etc/ansible/` as `ansible.cfg`

Basically , we will have a dummy ansible configuration file , to generate a full end configuration file for ansible , we need to use the following command in `/etc/ansible/`

```
ansible-config init --disabled -t all > ansible.cfg
```

Under which , search for `host_key_checking` and set the value to `False` , which disables host key verification.

We also need to change the permissions for key-pair file , from 644 to 400 , we need to make that file to private , else we would get error , while we are connecting to the target server with this unprotected key file.

For every module of ansible there will be response for the success status , for `ping` module the success response status is `pong` .

### inventory file

```
1 |   all:
2 |     hosts:
3 |       web01:
4 |         ansible_host: 172.31.31.178
5 |         ansible_user: ec2-user
6 |         ansible_ssh_private_key_file: clientkey.pem
7 |       web02:
8 |         ansible_host: 172.31.22.225
9 |         ansible_user: ec2-user
10 |        ansible_ssh_private_key_file: clientkey.pem
11 |       db01:
12 |         ansible_host: 172.31.19.215
13 |         ansible_user: ec2-user
14 |         ansible_ssh_private_key_file: clientkey.pem
15 |       children:
16 |         webservers:
17 |           hosts:
18 |             web01:
19 |             web02:
20 |           dbservers:
21 |             hosts:
22 |               db01:
23 |             dc_oregon:
24 |               children:
25 |                 webservers:
26 |                 dbservers:
```

We are adding another two hosts , where username and key-pair file both are same , now we will try to connect to these servers through ansible.

```
ansible web02 -m ping -i inventory
```

```
ansible db01 -m ping -i inventory
```

5:50

## 18. Ansible 191. Inventory Part 2



In an inventory file , we create groups , which will be helpful for us to launch multiple hosts at a time , above you can see , we are creating two groups **webservers & dbservers** , web01 & web02 are part of webservers group and db01 is part of dbservers group , for these groups , we are creating another parent group , which helps us to launch these two groups at a time. The parent group name we declared is **dc\_oregon** .

```
ansible webservers -m ping -i inventory
```

```
ansible dbservers -m ping -i inventory
```

```
ansible dc_oregon -m ping -i inventory
```

If we wanted to execute all groups of inventory ,

```
ansible '*' -m ping -i inventory
```

If we wanted to execute any group which is starting with web , we can use ,

```
ansible 'web*' -m ping -i inventory
```

9:27

## 18. Ansible 191. Inventory Part 2



We can declare variables at group level , to avoid repetition of variable declaration at each and every host level. **Priority of host variable > Priority of group level variable** , If we are going to declare a variable at group level , we shouldn't declare variable at host level , else the variable of host level will be prioritized than the group level.

```
1  all:
2    hosts:
3      web01:
4        ansible_host: 172.31.31.178
5      web02:
6        ansible_host: 172.31.22.225
7      db01:
8        ansible_host: 172.31.19.215
9      children:
10     webservers:
11       hosts:
12         web01:
13         web02:
14     dbservers:
15       hosts:
16         db01:
17     dc_oregon:
18       children:
19         webservers:
20         dbservers:
```

```
21     vars:  
22         ansible_user: ec2-user  
23         ansible_ssh_private_key_file: clientkey.pem
```

Executing the inventory file in whole scope of the file => `ansible all -m ping -i inventory`

2:36

## 18. Ansible 192. YAML & JSON



### Sample JSON Format:

```
1  {  
2      "DevOps":  
3          [  
4              "AWS",  
5              "Jenkins",  
6              "Python",  
7              "Ansible"  
8          ],  
9          "Development":  
10         [  
11             "Java",  
12             "NodeJS",  
13             ".net"  
14         ],  
15         "ansible_facts":  
16         {  
17             "python": "/usr/bin/python"  
18         }  
19     }
```

Above is the Sample JSON Format , where elements of the JSON are

```
1  DevOps  
2  Development  
3  ansible_facts
```

And values for those respective keys are,

```
1  DevOps => List  
2  Development => List  
3  ansible_facts => Dictionary
```

JSON Format , will be almost looking like a Python Dictionary , but JSON is a vertical format dictionary.

Most of the tools are using JSON instead of XML format , in Ansible , we will be also using YAML format to declare our requirement.

5:38

## 18. Ansible 192. YAML & JSON



### Equivalent YAML Format:

```
1  DevOps:  
2      - AWS  
3      - Jenkins
```

```

4   - Python
5   - Ansible
6
7   Development:
8   - Java
9   - NodeJS
10  - .net
11
12  ansible_facts:
13    python: /usr/bin/python
14    version: 2.7

```

Here Devops and its values are representation of Key with List of values , and ansible\_facts and it's values are representation of Key with dictionary of values.

We write ansible playbooks in YAML format , and the response from the ansible will be in json format.

This is the list of dictionaries in YAML Format ,

```

1   # Employee records
2   - martin:
3     name: Martin D'vloper
4     job: Developer
5     skills:
6       - python
7       - perl
8       - pascal Q
9   - tabitha:
10    name: Tabitha Bitumen
11    job: Developer
12    skills:
13      - lisp
14      - fortran
15      - erlang

```

martin and tabitha are dictionaries , and one of it's values "skills" is a list.

Please refer this further , for detailed analysis ,

[https://docs.ansible.com/ansible/latest/reference\\_appendices/YAMLSyntax.html](https://docs.ansible.com/ansible/latest/reference_appendices/YAMLSyntax.html)

0:36

18. Ansible 193. Ad Hoc Commands



## AdHoc Commands:

Ping module that we executed is an adhoc command.

[https://docs.ansible.com/ansible/latest/command\\_guide/intro\\_adhoc.html](https://docs.ansible.com/ansible/latest/command_guide/intro_adhoc.html)

For every configuration management tool , we need scripts to give commands. For ansible we have yaml format files , the ansible scripts are called as ansible playbooks. But if we wanted to execute some commands quickly , may be for verification purpose ,

or for some debugging purpose , we can use ansible AdHoc commands to execute immediately. AdHoc commands are simpler version of ansible playbooks. But if we wanted to use some commands , which should be same across all the instance , it's better to version control them and execute them through ansible playbooks.

4:25

## 18. Ansible 193. Ad Hoc Commands



**Use an AdHoc Command to install httpd package using yum package manager in web01 server:**

```
ansible web01 -m ansible.builtin.yum -a "name=httpd state=present" -i inventory
```

In here , web01 is the server name , -m is the module of ansible which we wanted to execute , we are passing arguments to yum package manager using `-a` , we are passing name of the package that we wanted to install under name argument , and as we are passing the server name as web01 , the details for that server will be present under the `inventory` file.

Mostly , if we are to install any package , we require sudo privilages. We use the following command to escalate the ansible to use the sudo command while installing the packages using yum package manager.

```
ansible web01 -m ansible.builtin.yum -a "name=httpd state=present" -i inventory --become
```

7:19

## 18. Ansible 193. Ad Hoc Commands



The main benefit of ansible when compared to automated scripts , which can manipulate the cloud resource is , automated scripts reapply the configuration irrespective of the cloud resource state , where as configuration management tool like ansible compare the states of the cloud resource and they apply only if the cloud resource state change. If it found the state changes , then ansible will apply the resource changes to the cloud provider.

Configuration management tools stores the state , these are idempotent in nature , what it means is if the target is in different state , it will apply the changes , but if those are in a same state then no changes applied.

8:59

## 18. Ansible 193. Ad Hoc Commands



**Adhoc Command 2:**

```
ansible webservers -m ansible.builtin.service -a "name=httpd state=started enabled=yes" -i inventory --become
```

Above , we are starting a httpd service and enabling it through ansible adhoc command on all the servers under webservers group with root privileges.

The above is equivalent to ,

```
systemctl start httpd
```

```
systemctl enable httpd
```

### Adhoc Command 3:

Copy a file from local server to group of server using ansible

```
ansible webservers -m ansible.builtin.copy -a "src=index.html  
dest=/var/www/html/index.html" -i inventory -- become
```

We can access this website by modifying the security group of the target servers , by giving permission to port 80 from MyIP , as we have launched httpd service , we can access this server through it's IP address access through the browser , we will see our index.html hosted in here.

1:15

18. Ansible 194. Playbook & Modules



Ansible Playbooks are written in YAML format , Playbooks determine a series of tasks to be executed on the target servers , whether it's packages installation , or executing some file on the remote etc. It will be something similar like this ,

```
1  - hosts: websrvgrp  
2    tasks:  
3      - name: Install Apache  
4        yum:  
5          name: httpd  
6          state: latest  
7      - name: Deploy Config  
8        copy:  
9          src: file/httpd.conf  
10         dest: /etc/httpd.conf  
11  - hosts: dbsrvgrp  
12    tasks:  
13      - name: Install Postgresql  
14        yum:  
15          name: postgresql  
16          state: latest
```

1:20

18. Ansible 194. Playbook & Modules



Ansible Playbooks , detailed explanation ,

```
1  - hosts: websrvgrp  
2    tasks:  
3      - yum:  
4          name: httpd  
5          state: present
```

Here hosts define list of servers where we want our playbook to be executed , tasks is one of the plays that is part of the playbook. Where as yum is the module that we wanted to execute and name and state are the arguments for the yum module.

1:57

## 18. Ansible 194. Playbook & Modules



If we wanted to give multiple tasks under a same play , we need to mention name for that under tasks section.

```
1  - hosts: websrvgrp
2
3  tasks:
4    - name: Install Apache
5      yum:
6        name: httpd
7        state: latest
8    - name: Deploy Config
9      copy:
10        src: file/httpd.conf
11        dest: /etc/httpd.conf
12
13   - hosts: dbsrvgrp
14     tasks:
15       - name: Install Postgresql
16         yum:
17           name: postgresql
18           state: latest
```

Here , we are having two plays , one is websrvgrp & dbsrvgrp , which will be executed on the respective list of hosts. We have three tasks in total , Install Apache & Deploy Config & Install Postgresql , with respective modules doing actions under them. First play having two tasks , second play having one task to do.

8:22

## 18. Ansible 194. Playbook & Modules



Example of Playbook ,

```
1  - name: Webserver setup
2    hosts: webservers
3    become: yes
4    tasks:
5      - name: Install httpd
6        ansible.builtin.yum:
7          name: httpd
8          state: present
9      - name: Start service
10        ansible.builtin.service:
11          name: httpd
12          state: started
13          enabled: yes
14      - name: DBserver setup
15        hosts: dbservers
16        become: yes
```

```
17 tasks:  
18   - name: Install mariadb-server  
19     ansible.builtin.yum:  
20       name: mariadb-server  
21       state: present  
22   - name: Start mariadb service  
23     ansible.builtin.service:  
24       name: mariadb  
25       state: started  
26       enabled: yes
```

8:50

## 18. Ansible 194. Playbook & Modules



We can manipulate servers through adhoc commands instead of playbooks , but for complex tasks playbooks are better.

```
ansible webservers -m yum -a "name=httpd state=absent" -i inventory -- become
```

How to execute ansible playbook,

```
ansible-playbook -i inventory web-db.yaml
```

Ansible will be having inbuilt modules for fetching up server information , through some inbuilt modules like setup.

15:09

## 18. Ansible 194. Playbook & Modules



We can print the debug output of the ansible playbook execution through verbose in four levels , which we can use for debugging.

```
ansible-playbook -i inventory web-db.yaml -v
```

```
ansible-playbook -i inventory web-db.yaml -vv
```

```
ansible-playbook -i inventory web-db.yaml -vvv
```

```
ansible-playbook -i inventory web-db.yaml -vvvv
```

We can also do a basic syntax check using the following command ,

```
ansible-playbook -i inventory web-db.yaml --syntax-check
```

We can do a dry run , before actually executing the playbook , it basically check the syntactical errors or any logical errors like , is that server exists or not.

```
ansible-playbook -i inventory web-db.yaml -C
```

Using Ansible's dry run feature enables users to **execute a playbook without making changes to the servers**. It uses the built-in check mode to proof a playbook for errors before execution. This option is very useful when executing complex playbooks that contain commands which make major changes to servers.

**Best practice of execution ,**

- 1) Run Syntax check
- 2) Run Dry Run
- 3) Run Actual ansible playbook

**Documentation:**

[https://docs.ansible.com/ansible/latest/playbook\\_guide/playbooks\\_intro.html](https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_intro.html)

**Ansible Modules Documentation:**

We will be having all the modules that are supported by ansible across all cloud service providers.

[https://docs.ansible.com/ansible/latest/collections/index\\_module.html](https://docs.ansible.com/ansible/latest/collections/index_module.html)

In here , we will be having a list of all modules that ansible supports ,

[https://docs.ansible.com/ansible/2.9/modules/modules\\_by\\_category.html](https://docs.ansible.com/ansible/2.9/modules/modules_by_category.html)

among them Cloud Module and Files Module are the famous.

If we wanted to change the hostname of the current machine which should be attached to IP address , we need to change the hostname while being on the destination machine.

`sudo vim /etc/hostname`

We are giving "control" as the hostname

`sudo hostname control`

Now , when ever , we login to the same server , we will see the hostname change from the actual IP address to the hostname that we changed , that's control.

Ansible , we basically convert the yaml format source file into python scripts and dump them to the destination server , where python should be installed , all the dependencies of python scripts that were dumped , should be installed , else ansible playbooks execution would be failed.

We can download the packages that we wanted to be on destination machines , by searching the package first in yum or pip or apt or any other package manager , if we are able to find there , its easy to install those packages and their dependencies.

We are installing that python MySQL package through yum package manager.

```

1   - name: DBserver setup
2     hosts: dbservers
3     become: yes
4     tasks:
5       - name: Install mariadb-server
6         ansible.builtin.yum:
7           name: mariadb-server
8           state: present
9       - name: Install pymysql
10      ansible.builtin.yum:
11        name: python3-PyMySQL
12        state: present
13      - name: Start mariadb service
14        ansible.builtin.service:
15          name: mariadb
16          state: started
17          enabled: yes
18      - name: Create a new database with name 'accounts'
19        mysql_db:
20          name: accounts
21          state: present
22          login_unix_socket: /var/lib/mysql/mysql.sock
23      - name: Create database user with name 'vprofile'
24        community.mysql.mysql_user:
25          name: vprofile
26          password: 'admin943'
27          priv: '*.*: ALL'
28          state: present
29          login_unix_socket: /var/lib/mysql/mysql.sock

```

In the world of DevOps, **socket files** play a crucial role in enabling efficient communication between processes.

## 1. What Are Socket Files?

- A **socket file** (also known as a Unix domain socket) is a special type of file used for **inter-process communication (IPC)** within the same machine.
- Unlike network sockets that operate over the network, socket files facilitate communication between processes running on the local system.
- They appear as regular files on disk but serve as endpoints for communication channels.

### Common Use Cases in DevOps:

- **Web Servers and Application Servers:**
  - Web servers (e.g., Nginx, Apache) use socket files to communicate with application servers (e.g., uWSGI, Gunicorn).
  - The web server forwards requests to the application server via socket files, improving performance and security.
- **Database Connections:**
  - Database servers (e.g., MySQL, PostgreSQL) often use socket files for local connections.
  - Applications communicate with the database server using these sockets, avoiding network overhead.
- **Container Orchestration:**
  - Container runtimes (e.g., Docker, Podman) create socket files for communication between containers.
  - Containers within the same pod can share data via these sockets.
- **System Services:**
  - System services (e.g., systemd, cron) use socket files for inter-process communication.
  - For example, systemd activates services when a socket receives a connection.

Whenever , we wanted to change the ansible default behavior , for example we wanted Ansible to connect to ssh by port 2020 instead of 22 which is default port to connect to ssh by ansible , we need to edit the **Ansible Configuration Settings**.

If we want to edit the configuration at the global level of ansible , we need to edit this file `/etc/ansible/ansible.cfg` , this file is a default configuration file , so this has the last priority.

File that we mentioned in `ANSIBLE_CONFIG` have the highest priority , ansible refers to this file for configuration details. Details in `ANSIBLE_CONFIG` override the default configuration file `/etc/ansible/ansible.cfg` , Note : **ANSIBLE\_CONFIG** is an environmental variable.

Second priority , is for the file in the project directory , if we mentioned any ansible.cfg file in the project directory , that will have the second priority. We use this method the most , because , we can commit this file to the repository , so , who ever is pulling the repo , can use this file directly for applying ansible related configuration.

Third Priority , is for the hidden file , which is in home directory `~/.ansible.cfg`

Last priority , is for the `/etc/ansible/ansible.cfg`

4:49

## 18. Ansible 196. Ansible Configuration



# and ; are comments in ansible configuration file.

By below environmental variables , you can set the ansible configuration.

[https://docs.ansible.com/ansible/latest/reference\\_appendices/config.html#ansible-configuration-settings-locations](https://docs.ansible.com/ansible/latest/reference_appendices/config.html#ansible-configuration-settings-locations)

9:20

## 18. Ansible 196. Ansible Configuration



We are writing a small ansible configuration file under our project directory.

```
1 [defaults]
2 host_key_checking=False
3 inventory = . /inventory
4 forks = 5
5 log_path = /var/log/ansible.log
6
7 [privilege_escalation]
8 become=True
9 become_method=sudo
10 become_ask_pass=False
```

Now , we are executing our db.yaml file , without passing any inventory details , because those details are present in ansible configuration file.

`ansible-playbook db.yaml`

If we are getting any such error ,

[WARNING]: log file at /var/log/ansible.log is not writeable and we cannot create it, aborting

It means that our current user have no permissions to create a file "ansible.log" and root user directory , so , let's create one by our own.

```
1 sudo touch /var/log/ansible.log
2 sudo chown ubuntu:ubuntu /var/log/ansible.log
3 ansible-playbook db.yaml
```

We must always have ansible configuration file at the same place of ansible playbooks in the repository , that's the standard.

2:25

## 18. Ansible 197. Variables & Debug



We can define variables in the playbook , immediately after the hosts section.

```
1 | - hosts: websrvgrp
2 | vars:
3 |   http_port: 80
4 |   sqluser: admin
```

In here , we have created two variables , http\_port & sqluser. These variables are the custom variables , that we define. Ansible , having it's own variables , which are inbuilt by default. The majority of the Ansible default variables are generated by the setup module , which will be executed , immediately the moment , we run ansible playbook.

Some of those are ,

**ansible\_os\_family** => OS name like RedHat, Debian etc

**ansible\_processor\_cores** => Number of CPU cores

**ansible\_kernel** => Kernel Version.

**ansible\_devices** => Connected Device Information

**ansible\_default\_ipv4** => IP, MAC address, gateway etc.

**ansible\_architecture** => 64 bit or 32 bit

4:45

## 18. Ansible 197. Variables & Debug



We can store the output of a playbook task into a variable with the help of **Register module**. Basically , Playbooks after Module execution returns output which is in JSON Format , we can store that output in a variable , either to print the output or for making a decision for another set of logics. The variables that we are speaking about are the runtime variables.

Defining variables inside the playbook is not a good practice , but using that outside the variable is good one.

To make the playbooks more generic , we can use the variables in the playbooks. So , we can use a single playbook for multiple environments and for various use cases , sometimes there will be cases , to change the use case , we need to change many occurrences of variable in the script.

10:03

## 18. Ansible 197. Variables & Debug



```
1 | - name: DBserver setup
2 | hosts: dbservers
```

```

3   become: yes
4
5   vars:
6     dbname: electric
7     dbuser: current
8     dbpass: tesla
9
10  tasks:
11    - debug:
12      msg: " The dbname is { {dbname} }"
13    - debug :
14      var: dbuser
15
16    - name: Install mariadb-server
17      ansible.builtin.yum:
18        name: mariadb-server
19        state: present
20
21    - name: Install pymysql
22      ansible.builtin.yum:
23        name: python3-PyMySQL
24        state: present
25
26    - name: Start mariadb service
27      ansible.builtin.service:
28        name: mariadb
29        state: started
30        enabled: yes
31
32    - name: Create a new database with name 'accounts'
33      mysql_db:
34        name: "{{dbname}}"
35        state: present
36        login_unix_socket: /var/lib/mysql/mysql.sock
37
38    - name: Create database user with name ' vprofile'
39      community.mysql.mysql_user:
40        name: "{{dbuser}}"
41        password: "{{dbpass}}"
42        priv: '*.*: ALL'
43        state: present
44        login_unix_socket: /var/lib/mysql/mysql.sock

```

12:26

**18. Ansible** 197. Variables & Debug

```

1   - name: Create database user with name ' vprofile'
2     community.mysql.mysql_user:
3       name: "{{dbuser}}"
4       password: "{{dbpass}}"
5       priv: '*.*: ALL'
6       state: present
7       login_unix_socket: /var/lib/mysql/mysql.sock
8     register: dbout
9
10    - name: print dbout variable
11      debug:
12        var: dbout

```

As said , if we want to store output of a module and use that in another module , we can take help of register module , above , we have created a register variable called dbout under first module , and in next module , we are debugging the output of first module.

1:02

## 18. Ansible 198. Group & Host Variables



Now we will check the approach of declaring the variables in the inventory level outside the playbooks , for example , if we wanted to create variables for set of groups , first we need to create a directory called group\_vars under the project directory , under that create a file called "all" , in which we need to define the variables.

```
1 | dbname: sky  
2 | dbuser: pilot  
3 | dbpass: aircraft
```

Now , if we run the playbooks , under our project directory , it uses these variables unless , we define similar variables in the playbook. Variables declared inside the playbook have higher priority than the one which are declared outside.

12:52

## 18. Ansible 198. Group & Host Variables



### Ansible Variables:

[https://docs.ansible.com/ansible/latest/playbook\\_guide/playbooks\\_variables.html](https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_variables.html)

1:05

## 18. Ansible 198. Group & Host Variables



If we wanted to create group variables for "all" scope , details of that scope is defined in the inventory file. So , variables for this scope should be defined under `group_vars/all` file under project directory. These are the variables , that we declared under scope "all" ,

```
1 | dbname: sky  
2 | dbuser: pilot  
3 | dbpass: aircraft
```

If we want our ansible to use these variables , ensure that , we don't declare these variables inside the playbook. Variables of playbook have more priority than the variables defined on global level.

Here , `all` means all hosts that are defined in inventory , these variables are applied for all those hosts which are defined in the inventory. In here , web01 , web02 , db01 are all those hosts.

6:12

## 18. Ansible 198. Group & Host Variables



Here , we are going to create two variables , and create an ansible builtin user with help of that, later registering the output of that task to a register variable : usrout , and using that variable , we are displaying the output using debug task.

```
1 | - name: Understanding vars
2 |   hosts: all
3 |   become: yes
4 |   vars:
5 |     USRNM: playuser
6 |     COMM: variable from playbook
7 |   tasks:
8 |     - name: create user
9 |       ansible.builtin.user:
10 |         name : "{{USRNM}}"
11 |         comment : "{{COMM}}"
12 |         register: usrout
13 |     - debug :
14 |       var: usrout.name
15 |     - debug :
16 |       var: usrout.comment
```

6:19

## 18. Ansible 198. Group & Host Variables



We will get output something like this ,

```
1 | ok: [web01] => {
2 |   "usrout.name": "playuser"
3 | }
4 | ok: [web02] => {
5 |   "usrout.name": "playuser"
6 | }
7 | ok: [db01] => {
8 |   "usrout.name": "playuser"
9 | }
1 | ok: [web01] => {
2 |   "usrout.comment": "variable from playbook"
3 | }
4 | ok: [web02] => {
5 |   "usrout.comment": "variable from playbook"
6 | }
7 | ok: [db01] => {
8 |   "usrout.comment": "variable from playbook"
9 | }
```

We can also define variables for groups , in here , webservers , dbservers , dc\_oregon are the groups. For example , if we wanted to define , variable for webservers , we have to declare under the file **group\_vars/webservers**

8:56

## 18. Ansible 198. Group & Host Variables



These are the variables defined ,

```
1 |   ubuntu@control: /vprofile/exercise9$ cat group_vars/all
2 |   USRNM: commonuser
3 |   COMM: variable from groupvars_all file
4 |   ubuntu@control: /vprofile/exercise9$ cat group_vars/webservers
5 |   USRNM: webgroup
6 |   COMM: variable from group_vars/webservers file
```

9:14

## 18. Ansible 198. Group & Host Variables



We can define variable under a yaml file `vars_precedence.yaml`

```
1 |   - name: Understanding vars
2 |     hosts: all
3 |     become: yes
4 |     vars:
5 |       USRNM: playuser
6 |       COMM: variable from playbook
7 |     tasks:
8 |       - name: create user
9 |         ansible.builtin.user:
10 |           name : "{{USRNM}}"
11 |           comment : "{{COMM}}"
12 |           register: usrout
13 |       - debug :
14 |         var: usrout.name
15 |       - debug :
16 |         var: usrout.comment
```

We can make that available on the respective hosts by running

```
ansible-playbook vars_precedence.yaml
```

9:41

## 18. Ansible 198. Group & Host Variables



We can define host variables under `host_vars` directory , and the file name will be according to the inventory file. `web01` , `web02` , `db01` are the hosts for us according to inventory file.

```
1 |   ubuntu@control: /vprofile/exercise9$ cat host_vars/web02
2 |   USRNM: web02user
3 |   COMM: variables from host_vars/web02 file
```

```
inventory & playbook variables > host variables > group variables > all host
variables
```

14:23

## 18. Ansible 198. Group & Host Variables



We can also define variables in an external file , and use that in the playbooks as ,

```
1 | - hosts: all
2 |   remote_user: root
3 |   vars:
4 |     favcolor: blue
5 |   vars_files:
6 |     - /vars/external_vars.yml
7 |   tasks:
8 |     - name: This is just a placeholder
9 |       ansible.builtin.command: /bin/echo foo
```

0:38

18. Ansible 199. Fact Variables



**Fact variables** will be generated only if setup module get's executed. These variables are runtime variables. Some of them are ,

**ansible\_os\_family** => (OS name like RedHat , Debian etc)

**ansible\_processor\_cores** => (Number of CPU Cores)

**ansible\_kernel** => (Kernel Version)

**ansible\_devices** => (Connected Device Information)

**ansible\_default\_ipv4** => (IP, MAC address, gateway etc.)

**ansible\_architecture** => (64 bit or 32 bit)

When we run Ansible Playbook , the first stage that executes is "Gathering Facts" , this stage runs the setup module , which helps to gather these facts. So , we can use these variables , according to our usage.

1:46

18. Ansible 199. Fact Variables



If we wanted to skip this **Gathering Facts** Stage , we must use the following command in ansible playbook **gather\_facts: False**

```
1 | - name: Understanding vars
2 |   hosts: all
3 |   become: yes
4 |   gather_facts: False
5 |   vars:
6 |     USRNM: playuser
7 |     COMM: variable from playbook
8 |   tasks:
9 |     - name: create user
10 |       ansible.builtin.user:
11 |         name : "{{USRNM}}"
12 |         comment : "{{COMM}}"
13 |         register: usrout
```

```
14     - debug :  
15         var: usrout.name  
16     - debug :  
17         var: usrout.comment
```

Now , when we run the playbook , gathering facts stage will be skipped.

2:46

## 18. Ansible 199. Fact Variables



If we wanted to get all the configuration and setup details of a host , then we have to use the following command.

```
ansible -m setup web01
```

We can use those setup details in the playbooks , we can even debug them by ,

```
1     - name: Print facts  
2     hosts: all  
3     tasks:  
4         - name: Print OS name  
5             debug:  
6                 var: ansible_distribution
```

If we are debugging a variable , which is not define , we will get such output ,

```
1     ok: [web01] => {  
2         "ansible_distribution": "VARIABLE IS NOT DEFINED!"  
3     }
```

We can also **declare the user for the host** in the inventory file , like this ,

```
1     all:  
2         hosts:  
3             web01:  
4                 ansible_host: 172. 31. 31. 178  
5             web02:  
6                 ansible_host: 172. 31. 22. 225  
7             web03:  
8                 ansible_host: 172. 31. 23. 53  
9                 ansible_user: ubuntu  
10            db01:  
11                 ansible_host: 172. 31. 19. 215
```

We can check are we able to connect all these hosts using the following adhoc ping command,

```
ansible -m ping all
```

8:40

## 18. Ansible 199. Fact Variables



Following is the example of usage of Fact Variables ,

```
1     - name: Print facts  
2     hosts: all  
3     #gather_facts: False  
4     tasks:  
5         - name: Print OS name
```

```

6   debug :
7     var: ansible_distribution
8   - name: Print selinux mode
9     debug :
10    var: ansible_selinux.mode
11   - name: Print RAM memory
12     debug:
13    var: ansible_memory_mb.real.free
14   - name: Print Processor name
15     debug :
16    var: ansible_processor[2]

```

In here , ansible\_distribution is normal fact variable , ansible\_selinux is a dictionary , ansible\_memory\_mb is dictionary or dictionary , ansible processor is a list.

0:06

## 18. Ansible 200. Decision Making



We will be **provisioning server**, while we are doing that , we will go through many important topics like the below mentioned.

- 1) Setting up NTP Service on multi OS (Centos & Ubuntu)
- 2) We will be setting up users & groups
- 3) We will be creating configuration files on the server.
- 4) In the process of these , we will do the decision making in the playbook.
- 5) Loops
- 6) Templates for configurations.
- 7) Handlers
- 8) Ansible Roles

We mainly know , how to provision a server.

5:01

## 18. Ansible 200. Decision Making



Now , we are writing a playbook where tasks of that should be executed conditionally ,

```

1   - name: Provisioning servers
2     hosts: all
3     become: yes
4     tasks:
5       - name: Install ntp agent on centos
6         yum:
7           name: chrony
8           state: present
9           when: ansible_distribution == "CentOS"
10      - name: Install ntp agent on ubuntu
11        apt:

```

```

12     name: ntp
13     state: present
14     update_cache: yes
15     when: ansible_distribution == "Ubuntu"
16   - name: Start service on centos
17     service:
18       name: chronyd
19       state: started
20       enabled: yes
21     when: ansible_distribution == "CentOS"
22   - name: Start service on ubuntu
23     service:
24       name: ntp
25       state: started
26       enabled: yes
27     when: ansible_distribution == "Ubuntu"

```

In here , when conditions are the decision making clauses , that particular task block will only be executed , if respective when condition satisfied.

Dry Run: `ansible-playbook provisioning.yaml -C`

7:07

## 18. Ansible 200. Decision Making



`update_cache: yes` perform `apt update` and then install the required package

10:36

## 18. Ansible 200. Decision Making



Detailed Documentation on Decision making in Ansible

Playbooks: [https://docs.ansible.com/ansible/latest/playbook\\_guide/playbooks\\_conditionals.html](https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_conditionals.html)

1:52

## 18. Ansible 201. Loops



If we wanted to execute a task multiple times , for example there will be cases where we install many modules on the same server using same package manager. There will be code duplication , if we do that through Ansible. So , to prevent code duplication , we use loops and use placeholders at module name , as following ,

```

1   - name: Provisioning servers
2     hosts: all
3     become: yes
4     tasks:
5       - name: Install ntp agent on centos
6         yum:
7           name: "{{item}}"
8           state: present

```

```

9   when: ansible_distribution == "CentOS"
10  loop:
11    - chrony
12    - wget
13    - git
14    - zip
15    - unzip

```

`{{item}}` is the placeholder for loop , so use `{{item}}` , whenever , we want to access the loop iterator.

So , by following code , this particular yum task runs 5 times through all the modules that are present in the loop.

3:00

## 18. Ansible 201. Loops

```

1  - name: Provisioning servers
2    hosts: all
3    become: yes
4    tasks:
5      - name: Install ntp agent on centos
6        yum:
7          name: "{{item}}"
8          state: present
9          when: ansible_distribution == "CentOS"
10         loop:
11           - chrony
12           - wget
13           - git
14           - zip
15           - unzip
16      - name: Install ntp agent on ubuntu
17        apt:
18          name: "{{item}}"
19          state: present
20          update_cache: yes
21          when: ansible_distribution == "Ubuntu"
22         loop:
23           - ntp
24           - wget
25           - git
26           - zip
27           - unzip

```

In below , we have two tasks , which are having loops respectively.

5:16

## 18. Ansible 201. Loops

We can also loop through list of dictionaries ,

```

1  - name: Add several users
2    ansible.builtin.user:

```

```
3   name: "{{ item.name }}"
4   state: present
5   groups: "{{ item.groups }}"
6   loop:
7     - { name: 'testuser1', groups: 'wheel' }
8     - { name: 'testuser2', groups: 'root' }
```

You can have the whole documentation related to loops under here

[https://docs.ansible.com/ansible/latest/playbook\\_guide/playbooks\\_loops.html](https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_loops.html)

1:53

## 18. Ansible 202. File, copy & template modules



If we want to deal with the files using ansible , we need to use file module of ansible.

Below is the clear documentation ,

[https://docs.ansible.com/ansible/2.8/modules/list\\_of\\_files\\_modules.html](https://docs.ansible.com/ansible/2.8/modules/list_of_files_modules.html)

We will be having many functions to deal with the files using this ansible file module. Below , we are using copy method of file module , copy module copies the files to the remote location

3:36

## 18. Ansible 202. File, copy & template modules



Below , we are copying our content into destination `/etc/motd` file of remote server , where this file ensures to display the text of that , while anyone login to this server. This file is also called as Banner file.

```
1   - name: Banner file
2     copy:
3       content: '# This server is managed by ansible. No manual changes please. '
4       dest: /etc/motd
```

We can also use a source file , instead of content , if we have minimal text to export , it's preferable to use content rather than the source file.

9:45

## 18. Ansible 202. File, copy & template modules



We have a template module , which works similar to copy module , which copies the files from source to the destination , in here , we are copying the configuration files from our local machine to the desired file under specified directory.

```
1   - name: Deploy ntp agent conf on centos
2     template:
3       src: templates/ntpconf_centos
4       dest: /etc/chrony.conf
5       backup: yes
6       when: ansible_distribution == "CentOS"
7
8   - name: Deploy ntp agent conf on ubuntu
```

```

9   template:
10  src: templates/ntpconf_ubuntu
11  dest: /etc/ntp.conf
12  backup: yes
13  when: ansible_distribution == "Ubuntu"

```

These two should be present under tasks section. After making some changes on configuration files , we must restart the services.

```

1  - name: reStart service on centos
2    service:
3      name: chronyd
4      state: restarted
5      enabled: yes
6    when: ansible_distribution == "CentOS"
7
8  - name: reStart service on ubuntu
9    service:
10   name: ntp
11   state: restarted
12   enabled: yes
13  when: ansible_distribution == "Ubuntu"

```

12:53

**18. Ansible** 202. File, copy & template modules



In Ansible, both the **copy** and **template** modules are used for transferring files to remote systems, but they serve slightly different purposes:

### Copy Module:

- The **copy** module does exactly what you'd expect: it copies a file from the local host to a remote server.
  - It's straightforward to use and requires little explanation.
  - Example playbook snippet using the **copy** module:

```

1  ---
2  - hosts: nyc1-webserver-1.example.com
3  gather_facts: no
4  tasks:
5    - name: Copy MOTD into place
6      copy:
7        dest: /etc/motd
8        src: etc/motd
9        owner: root
10       group: root
11       mode: 0644

```

- The **copy** module is ideal for files that are consistent across all systems, such as login banners, messages of the day (motd), or base application configuration files.

12:54

**18. Ansible** 202. File, copy & template modules



## Template Module:

- The **template** module also copies a file to a remote server, but it allows you to use Jinja2 templating to render the file dynamically.
- With the **template** module, you can customize files based on variables (e.g., Ansible facts) specific to each server.
- Example use case: configuring an application like **keepalived** with a dynamically generated configuration file.
- The playbook snippet below installs and configures **keepalived** using the **template** module:

```
1 | ---
2 | - hosts: nyc1-webserver-1.example.com
3 |   gather_facts: no
4 | 
5 |   tasks:
6 |     - name: Install and configure keepalived
7 |       template:
8 |         src: templates/etc/keepalived/keepalived.conf.j2
9 |         dest: /etc/keepalived/keepalived.conf
10 |        owner: root
11 |        group: root
12 |        mode: 0644
```

- The **template** module is useful when you need to customize files based on server-specific data.

12:55

18. Ansible 202. File, copy & template modules



In summary, use the **copy** module for consistent files and the **template** module when you need dynamic content with Jinja2 templating

12:56

18. Ansible 202. File, copy & template modules



The **template** module in Ansible reads files dynamically using Jinja2 templating. Let's break it down:

### 1. File Format:

- The **template** module works with plain text files (usually configuration files) that contain placeholders or variables.
- These placeholders are written in a specific format, which Jinja2 recognizes and replaces with actual values during runtime.

12:57

18. Ansible 202. File, copy & template modules



### 2) Jinja2 Templating Syntax:

- Jinja2 uses double curly braces (`{}{}`) to denote placeholders.
  - You can also use control structures (such as loops and conditionals) using `{% %}` tags.

- Here's a simple example of a Jinja2 template snippet:

```

1  # Example template file: nginx.conf.j2
2  server {
3      listen {{ nginx_port }};
4      server_name {{ server_name }};
5      location / {
6          root {{ web_root }};
7          index index.html;
8      }
9  }
```

- In this example:

- `nginx_port`, `server_name`, and `web_root` are placeholders.
- During execution, Ansible will replace these placeholders with actual values based on the variables you provide.

12:58

## 18. Ansible 202. File, copy & template modules



### 3) Providing Variables:

- In your Ansible playbook, you define variables (either globally or per host/group) that correspond to the placeholders in the template.
- Example playbook snippet:

```

1  ---
2  - hosts: web_servers
3  vars:
4      nginx_port: 80
5      server_name: example.com
6      web_root: /var/www/html
7  tasks:
8      - name: Configure Nginx
9          template:
10         src: templates/nginx.conf.j2
11         dest: /etc/nginx/nginx.conf
```

- When Ansible runs this playbook, it will replace the placeholders in `nginx.conf.j2` with the provided variable values.

12:59

## 18. Ansible 202. File, copy & template modules



### 4) Dynamic Configuration:

- The power of the **template** module lies in its ability to create dynamic configuration files.

- For instance, you can generate different Nginx configurations for different servers by varying the values of `nginx_port`, `server_name`, and `web_root`.
- This flexibility allows you to tailor configurations to specific hosts or environments.

Remember that the **template** module is not limited to configuration files; you can use it for any text-based file that requires dynamic content. Whether it's Nginx, Apache, database configurations, or custom scripts, Jinja2 templating makes it easy to create personalized files for each system.

14:22

## 18. Ansible 202. File, copy & template modules



`groupvars/all` file, where we define all the required variables for all groups , that we will use in ansible playbooks.

```

1 | USRNM: commonuser
2 | COMM: variable from groupvars_all file
3 | ntp0: 0. north-america. pool. ntp. org
4 | ntpl: 1. north-america. pool. ntp. org
5 | ntp2: 2. north-america. pool. ntp. org
6 | ntp3: 3 north-america. pool. ntp. org

```

We defined such content under **templates/ntpconf\_centos** , this is configuration file, we will do the same in **templates/ntpconf\_ubuntu** ,

```

1 | # Use public servers from the pool. ntp. org project.
2 | # Please consider joining the pool (https://www.pool.ntp.org/join.html).
3 | pool "{{ntp0}}" iburst
4 | pool "{{ntp1}}" iburst
5 | pool "{{ntp2}}" iburst
6 | pool "{{ntp3}}" iburst

```

Now for such kinds of configuration files , if we use the copy module , the configuration file will be dumped with the placeholders as it is , If we use template module , all these placeholders will be replaced and dumped into the server.

6:23

## 18. Ansible 203. Handlers



### Handlers Documentation:

[https://docs.ansible.com/ansible/latest/playbook\\_guide/playbooks\\_handlers.html](https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_handlers.html)

We must place handlers at the same level of "tasks" in yml file.

```

1 | handlers:
2 |   - name: reStart service on centos
3 |     service:
4 |       name: chronyd
5 |       state: restarted

```

```

6   enabled: yes
7   when: ansible_distribution == "CentOS"
8
9     - name: reStart service on ubuntu
10    service:
11      name: ntp
12      state: restarted
13      enabled: yes
14      when: ansible_distribution == "Ubuntu"

```

2:27

## 18. Ansible 203. Handlers



We must link these handlers based on name to actual tasks using notify attribute. So , whenever , the state of the task is changed , this task notifies the handler , else , the handler won't get trigger.

Under "tasks" section declare this ,

```

1   - name: Deploy ntp agent conf on centos
2     template:
3       src: templates/ntpconf_centos
4       dest: /etc/chrony.conf
5       backup: yes
6       when: ansible_distribution == "CentOS"
7       notify:
8         - reStart service on centos
9   - name: Deploy ntp agent conf on ubuntu
10    template:
11      src: templates/ntpconf_ubuntu
12      dest: /etc/ntp.conf
13      backup: yes
14      when: ansible_distribution == "Ubuntu"
15      notify:
16        - reStart service on ubuntu

```

**We can also trigger two handlers under same task.**

0:14

## 18. Ansible 204. Roles



### Ansible Roles:

Ansible Roles is to simplify playbook content. Until now , we have seen Global Declaration like 'become' in Playbook & Variables , Tasks , Files , Templates , Handlers in Playbooks.

3:33

## 18. Ansible 204. Roles



**Ansible roles** are a way to organize and structure your Ansible playbooks and tasks. They provide a modular and reusable approach to managing configuration, making it easier to maintain and scale your infrastructure automation.

3:34

## 18. Ansible 204. Roles



Here are the key points about Ansible roles:

1. **Modularity:** Roles allow you to break down your playbook into smaller, self-contained units. Each role represents a specific functionality or component of your system (e.g., installing a web server, configuring a database, setting up monitoring).
2. **File Structure:** A typical Ansible role has a predefined directory structure. This structure includes directories for tasks, variables, templates, handlers, and other artifacts. This consistency makes it easier to organize and find relevant files.
3. **Reusability:** Roles can be reused across different playbooks. You can create a role once and then include it in multiple playbooks, reducing duplication and ensuring consistency.
4. **Dependencies:** Roles can depend on other roles. For example, a web server role might depend on a common utilities role. Ansible automatically resolves these dependencies when you include roles in your playbook.

3:35

## 18. Ansible 204. Roles



5. **Variables:** Roles can define their own variables. These variables can be set at the playbook level or overridden when including the role. This flexibility allows you to customize role behavior without modifying the role itself.

6. **Handlers:** Roles can define handlers, which are tasks that only run when notified. Handlers are useful for actions like restarting services after configuration changes.

3:37

## 18. Ansible 204. Roles



Structure would be something like this ,

```
1 |   roles
2 |   ==>common-server
3 |   ===>defaults
4 |   =====>main.yml
5 |   ===>files
6 |   ===>handlers
7 |   =====>main.yml
8 |   ===>meta
```

```
9 | =====>main.yml
10| =====>README.md
11| =====>tasks
12| =====>main.yml
13| =====>templates
14| =====>tests
15| =====>inventory
16| =====>test.yml
17| =====>vars
18| =====>main.yml
```

A complete playbook , breakdown into group of files , so this gives modularity and reusability and readability. The folder names are static , declaration of variables should be done under vars folder , tasks should be done under tasks folder.

4:42

## 18. Ansible 204. Roles



We gonna add some more tasks into the playbook ,

```
1 | - name: Dump file
2 |   copy:
3 |     src: files/myfile.txt
4 |     dest: /tmp/myfile.txt
5 |
6 | - name: Create a folder
7 |   file:
8 |     path: "{{mydir}}"
9 |     state: directory
```

We gonna define variable for mydir in the playbook level.

```
1 | vars:
2 |   mydir: /opt/dir22
```

8:33

## 18. Ansible 204. Roles



If we wanted to see the directory structure in tree format , you must need to install tree module first. Do that through ,

```
sudo apt install tree -y
```

Later , you can select a directory to list it's files and folders in tree format , consider exercise14 is a directory , use `tree exercise14` to list subdirectories and files in tree format.

Creating an ansible role from the start is an hectic task , but converting an ansible playbook into ansible role is an easy task.

9:36

## 18. Ansible 204. Roles



Now create a directory roles under your project directory , where to create a role , use `ansible-galaxy init post-install` command to create a role called 'post-install' , later you can see the basic structure of ansible role is created under post-install directory which we discussed below.

10:14

## 18. Ansible 204. Roles



Some of the role files that we created ,

**roles/post-install/vars/main.yml** (earlier we kept this under group\_vars/all file excluding mydir variable , which we defined directly in the playbook , so add that now in main.yml) , remove group\_vars & host\_vars as they are not needed furthur.

```
1 | # vars file for post-install
2 | USRNM: commonuser
3 | COMM: variable from groupvars_all file
4 | ntp0: 0.north-america.pool.ntp.org
5 | ntpl: 1.north-america.pool.ntp.org
6 | ntp2: 2.north-america.pool.ntp.org
7 | ntp3: 3.north-america.pool.ntp.org
8 | mydir: /opt/dir22
```

Copy all the files from files folder under the project directory to the files directory under role , which is `roles/post-install/files`

```
cp files/* roles/post-install/files/
```

Do the same for templates ,

```
cp templates/* roles/post-install/templates/
```

12:35

## 18. Ansible 204. Roles



Accumulate Handlers main.yml file ,

**roles/post-install/handlers/main.yml**

```
1 | # handlers file for post-install
2 | - name: reStart service on centos
3 |   service:
4 |     name: chronyd
5 |     state: restarted
6 |     enabled: yes
7 |     when: ansible_distribution == "CentOS"
8 |
9 | - name: reStart service on ubuntu
10 |   service:
11 |     name: ntp
12 |     state: restarted
13 |     enabled: yes
14 |     when: ansible_distribution == "Ubuntu"
```

Similarly accumulate , **roles/post-install/tasks/main.yml** with all the tasks.

Now the ansible-playbook will be as simple as ,

```
1 | - name: Provisioning servers
2 |   hosts: all
3 |   become: yes
4 |   roles:
5 |     - post-install
```

All the handlers and tasks , variables under post-install role will be executed , we can also define multiple roles in a single ansible-playbook.

Now , in the respective main.yml files of specified role , we no need to declare the relative path for the file/template or any other , because , the role is smart enough to find the file , so just give the file name , rest role will take care.

18:48

## 18. Ansible 204. Roles



We can declare the variables under `defaults/main.yml` rather than `vars/main.yml` , its the general and preferable practice.

**defaults/main.yml have lesser priority than the vars/main.yml**

**Ansible Roles Documentation:**

[https://docs.ansible.com/ansible/2.9/user\\_guide/playbooks\\_reuse\\_roles.html](https://docs.ansible.com/ansible/2.9/user_guide/playbooks_reuse_roles.html)

We can override the variables for specific role by mentioning them in the playbook like ,

```
1 | - name: Provisioning servers
2 |   hosts: all
3 |   become: yes
4 |   roles:
5 |     - role: post-install
6 |       vars:
7 |         ntp0: 0.in.pool.ntp.org
8 |         ntp1: 1.in.pool.ntp.org
9 |         ntp2: 2.in.pool.ntp.org
10 |        ntp3: 3.in.pool.ntp.org
```

25:30

## 18. Ansible 204. Roles



Instead of creating roles manually , we can use pre-created roles , which are published in ansible-galaxy community. You can check those in here ,

<https://galaxy.ansible.com/ui/standalone/roles/>

For example , geerlingguy.java is the role , we use that in ansible playbook as ,

```
1 | - name: Provisioning servers
```

```
2 hosts: all
3   become: yes
4   roles:
5     - geerlingguy.java
6     - role: post-install
7   vars:
8     ntp0: 0.in.pool.ntp.org
9     ntp1: 1.in.pool.ntp.org
10    ntp2: 2.in.pool.ntp.org
11    ntp3: 3.in.pool.ntp.org
```

geerlingguy is the pre-defined role , and post-install is user-defined role , first geerlingguy will execute later post-install , in the order of declaration of roles in ansible-playbook.

Using pre-defined roles is less preferable if we need more customization.

We can import variables of other yml files to the main.yml using `include_vars` attribute.

0:25

## 18. Ansible 205. Ansible for AWS



### How to use Ansible to manage our AWS Services:

If we wanted ansible to connect to AWS , we basically needed Access key and Secret Key of IAM User , we export these variables in ansible , so that ansible can connect AWS resources.

```
1 export AWS_ACCESS_KEY_ID='AK123'
2 export AWS_SECRET_ACCESS_KEY='abc123'
```

As seen in here , we have to create an IAM User , and under there under security credentials , we need to create one access key and one secret key. Download access keys as csv file format. Now export these access key and secret access key variables with IAM User generated values.

If we export these variables through export command , these will be available for only one session , if we re-login , these variables will be destroyed. To prevent this , we can add these variables to bashrc script , so whenever , we login to a session , these variables get auto assigned.

```
vim .bashrc
```

7:12

## 18. Ansible 205. Ansible for AWS



To utilize amazon AWS in ansible , we need a python library called 'boto3' , we need to install that on the server , where we have ansible.

```
sudo apt install python3-pip y
```

```
pip3.10 install boto3
```

Main idea of us is to create a key-pair using ansible yml files in amazon AWS ,

```
1  - hosts: localhost
2    gather_facts: False
3    tasks:
4      - name: Create key pair
5        ec2_key:
6          name: sample
7          region: us-east-1
8          register: keyout
9      - name: save key
10        copy:
11          content: "{{keyout.key.private_key}}"
12          dest: ./sample.pem
13        when: keyout.changed
```

14:09

**18. Ansible** 205. Ansible for AWS



In here , after providing the access key and secret key to ansible for connecting to the AWS , we here , are creating a key-pair , and later saving the private key which is created from one task into a variable , carry forwarding that to another task through register variables , and create a file for private key only if the key pair created for the first time , later , even if we re-executed , this stage will be skipped unless the key-pair that we earlier created gets deleted.

To launch aws instance through ansible , we need this package , install it through ansible-galaxy,

```
ansible-galaxy collection install amazon.aws
```

, which installs all the amazon aws modules in the control machine.

18:28

**18. Ansible** 205. Ansible for AWS



```
1  - name: start an instance
2    amazon.aws.ec2_instance:
3      name: "public-compute-instance"
4      key_name: "sample"
5      #vpc_subnet_id: subnet-5calable
6      instance_type: t2.micro
7      security_group: default
8      #network:
9        # assign_public_ip: true
10     image_id: ami-02d8bad0a1da4b6fd
11     exact_count: 1
12     region: us-west-2
13     tags:
14       Environment: Testing
```

In here , we are launching an instance through ansible yml scripts , we are mentioning the name of the instance , key-pair for it , instance type , security\_group , image\_id on which the instance should be based on , and the region it must be launched to , and tags.

We must surely mention , the exact\_count of the instance , so that , even we re-execute the yml scripts multiple times , only one instance will be created , if this is not mentioned , there is a chance that we may create a pile of instances.

0:30

## 22. Containerization 239. Introduction



This Lecture basically tells us , how and when we need to containerize the application.

Suppose , consider we are having an multi tier application stack or many services which are running on VM's. This VM can be either on the host machine or on the cloud. In today in agile environment , we must do continuous changes with continuous deployments. If we are performing these actions , through manual deployment , if we are maintaining the resources physically , we must raise Capex , even if we are using the cloud , to maintain the workflow , we must spend more on Opex , as we must setup everything manually on all the destination servers manually.

But there will be a change of human errors in deployment , which may take huge amount of time , if the process is not done in a workflow.

And this doesn't support microservice architecture , because , every service must consist a host os for that , so having os for every service requires more space. And resource wastage is possible.

3:22

## 22. Containerization 239. Introduction



Usual Method can't even synchronize all the environments (dev , prod , QA) , which results in failure in deployment , but it is possible while using in docker , as we can store all the requirements and environmental variables in docker image , and we can export that to all the machines , where we will have kept them in sync , when we are using the same docker image.

Solution for this is to use Containers , where these can consume less resource as these not needed to contain the whole operating system as it contains in microservice architecture.

In Containers , deployment are done through Docker Images. So , where-ever , we send that docker image , we will be having same configuration across servers. As we are having same image across all environments , we also be having same containers across.

5:59

## 22. Containerization 239. Introduction



**Docker** is a container runtime environment to build our images. We can use docker-compose to host our docker images on multiple servers.

All the customization that we need to make for the base image , we can make it through **Dockerfile**.

We write **docker-compose.yml** to run multiple containers at a time , and these containers can be run based on the base images or customized images which we built.

If our customized image is helpful for others , we can publish that docker image on to DockerHub.

9:36

## 22. Containerization 239. Introduction



Basically with the docker the workflow will be like , we pull the code from the github , In that , we will be having Dockerfile , we will be building that image based on this Dockerfile

```
docker build -t imranvisualpath/vproapp
```

In the Dockerfile , while we are building our customized image , we will be pulling the base images from dockerfile. We will be running containers based on the customized image or the images which are present on the Dockerhub on multiple servers through docker-compose , if we are fine with the output , we will push these Docker Images to Dockerhub through

```
docker push imranvisualpath/vproapp
```

11:16

## 22. Containerization 240. Overview of Base Image



In here , it is been discussed about setting up all the services. For some services , we can use direct docker images on docker hub , where as some images need customization , for example , we want our application artifact in our tomcat container to run our application in it.

1:20

## 22. Containerization 241. Dockerhub Setup



### Benefits of DockerHub Organization:

We use docker organization plans to collaborate among developers and develop docker images , basically companies will definitely take this plan. It provides docker desktop , unlimited private repositories , supports 5000 image pulls a day , enhanced container isolation , more security. We can also name the images by the organization name rather

than account name , which usually happens in free tier. We can also give our repository links (GitHub / Bitbucket) to the docker hub organization , so whenever , there is a change or a commit on the repository , the organization triggers a build for a docker image to update it. Can also set the repository privacy , to be public or private. For free accounts , there won't be an option to be private. Have option to send build notification to sent , in any of the following events (Only Failures or Every Build).

2:30

## 22. Containerization 242. Setup Docker Engine



In a vagrant file , while we are running the vagrant boxes locally to launch a VM locally , we can configure the **Vagrantfile** after pulling the **Vagrant box** with **Vagrant init**. Later to reach that through internet , we can either connect to it , via private IP address , that we configure under **Vagrantfile** or reach the VM via public network.

We have to edit these configurations in Vagantfile ,

```
1 # Create a private network, which allows host-only access to the machine
2 # using a specific IP.
3 config.vm.network "private_network", ip: "192.168.56.38"
4
5 # Create a public network, which generally matched to bridged network.
6 # Bridged networks make the machine appear as another physical device on
7 # your network.
8 config.vm.network "public_network"
```

2:59

## 22. Containerization 242. Setup Docker Engine



Based on the Available RAM on the system , we can allocate to the Virtual Box , by editing Vagrantfile.

```
1 # Provider-specific configuration so you can fine-tune various
2 # backing providers for Vagrant. These expose provider-specific options.
3 # Example for VirtualBox:
4
5 config.vm.provider "virtualbox" do | vb |
6   # Display the VirtualBox GUI when booting the machine
7   # vb.gui = true
8   # Customize the amount of memory on the VM:
9   vb.memory = "2048"
10 end
```

Allocating 2GB RAM is better to be faster VM , if it's 1GB it will be slower , but working.

7:10

## 22. Containerization 242. Setup Docker Engine



Initialize vagrant box by: **vagrant init bento/ubuntu-22.04**

Edit Vagrantfile by: **vim Vagrantfile**

Run the vagrant box and launch the VM by : `vagrant up`

Install docker engine on ubuntu: <https://docs.docker.com/engine/install/ubuntu/>

Login to the vagrant VM instance by `vagrant ssh` command.

If we wanted to execute docker commands on vagrant machine , then we must add our vagrant's user to the docker group

```
usermod -aG docker vagrant
```

ensure you are root user , while you execute this command , later logout and re-login , now we can execute docker commands.

Test command of docker: `docker images`

1:03

## 22. Containerization 243. Dockerhub & Dockerfile References



Forking in GitHub is copying the repository of a account with all the details of the branches into another independent account. We can fork only current branch of the repository or we can copy the whole repository.

6:05

## 22. Containerization 243. Dockerhub & Dockerfile References



If we wanted to store our newly created images into Dockerhub , we need to create repositories , where namespace would be our username and repository name would be vprofileweb , where as when we are pulling an image or pushing it , we will reference that image as ,

```
docker pull namespace/repository_name:tagname
```

In here mostly , namespace would be organization name , if it exists , where as repository\_name is mandatory , which is the image name and default tag name for the image is latest.

8:31

## 22. Containerization 243. Dockerhub & Dockerfile References



**Dockerfile references:** <https://docs.docker.com/reference/dockerfile/>

RUN command basically execute the instructions to build and image for application container.

FROM command imports the base image , which is the first step that we do for customizing an image.

Dockerfile is a series of instructions to build an image.

COPY copies the file from the local machine to container directory.

CMD executes the binaries that process in the container.

LABEL is a way of creating a key value pairs , that we can further use.

The EXPOSE instruction in a Dockerfile informs Docker that the container listens on specific ports at runtime. It acts as a documentation mechanism for users of the image, letting them know which ports are relevant for the application.

However, it's important to note that EXPOSE doesn't actually publish the ports to the host system or make them accessible externally. To make ports accessible, you'll need to use the -p flag when running the container with docker run.

8:57

## 22. Containerization 243. Dockerhub & Dockerfile References



ENV is used to set environmental variables for an image that can be used in container.

ADD is to push any file into container.

Both ADD and COPY commands in a Dockerfile are used to copy files or directories from the host machine into a Docker image. However, they have some key differences:

\* Functionality:

\* ADD offers more functionalities than COPY. In addition to copying files and directories, ADD can also:

- \* Fetch files from remote URLs.

- \* Automatically extract compressed files (tar, gzip, bzip2) during the copy process.

\* Use Cases:

- \* Use ADD when you need to:

- \* Copy files from URLs.

- \* Extract compressed files during the build process.

- \* Use COPY for simpler tasks like copying local files or directories without any additional functionalities.

In summary, while ADD is more versatile, COPY is preferred for its simplicity and transparency.

10:00

## 22. Containerization 243. Dockerhub & Dockerfile References



ENTRYPOINT have higher priority than CMD , for an entry point , can provide executable and commands that need to get executed.

Whatever , we mention in CMD will be the arguments for the ENTRYPPOINT , if we are providing any argument to the image , the commands in the CMD will get overridden.

1:33

## 22. Containerization 244. App Image Dockerfile



It is suggested to make the docker images as small as possible , for that case , we are introducing multi stage build , in one stage , we gonna build the artifacts , which occupies more space , where as in the next stage , we just copy those artifacts , so all the build cache will be eliminated , so the resultant image would be less.

1:58

## 22. Containerization 244. App Image Dockerfile



### Better Organization and Standardization:

- Multi-stage builds provide a standardized method for running build actions.
- You can organize your Docker commands and files more effectively, making your Dockerfile easier to manage and understand.

1:55

## 22. Containerization 244. App Image Dockerfile



Multi-stage builds are incredibly useful for optimizing Dockerfiles while maintaining readability and ease of maintenance. Let me explain why they are beneficial:

#### 1. Parallel Build Steps:

- Multi-stage builds allow you to run build steps in parallel, making your build pipeline faster and more efficient1.
- By breaking down the build process into separate stages, you can take advantage of parallel execution, reducing overall build time.

#### 2. Smaller Final Images:

- With multi-stage builds, you create a final image with a smaller footprint. It contains only what's needed to run your program1.
- The resulting image includes only the essential artifacts, leaving behind any intermediate build tools or dependencies.
- This smaller image size improves deployment speed and reduces storage requirements.

10:22

## 22. Containerization 244. App Image Dockerfile



```
1 | FROM openjdk:11 AS BUILD_IMAGE
2 | RUN apt update && apt install maven -y
3 | RUN git clone https://github.com/devopshydclub/vprofile-project.git
```

```

4 RUN cd vprofile-project && git checkout docker && mvn install
5
6 FROM tomcat:9-jre11
7 RUN rm -rf /usr/local/tomcat/webapps/*
8 COPY --from=BUILD_IMAGE vprofile-project/target/vprofile-v2.war
   /usr/local/tomcat/webapps/ROOT.rar
9
10 EXPOSE 8080
11 CMD ["catalina.sh", "run"]

```

In here , we are using multi stage Dockerfile , where in first stage , we are using openjdk as the base image , and building the vprofile-project using maven and generating an artifact using maven. And we gonna keep this artifact in the location of tomcat to host the website. In here , tomcat is the second image that we are using in the file , all the cache of the stage 1 will be cleared up , after the build. In the second stage , we are clearing up the default artifact that will be present under /usr/local/tomcat/webapps/ , and copying our artifact that we built in the first stage.

3:36

## 22. Containerization 246. Web Image Dockerfile



### web\ Dockerfile

```

1 FROM nginx
2 LABEL "Project"="Vprofile"
3 LABEL "Author"="Imran"
4
5 RUN rm -rf /etc/nginx/conf.d/default.conf
6 COPY nginxvaproapp.conf /etc/nginx/conf.d/vaproapp.conf

```

### app\ Dockerfile

```

1 FROM tomcat:8-jre11
2 LABEL "Project"="Vprofile"
3 LABEL "Author"="Imran"
4 RUN rm -rf /usr/local/tomcat/webapps/*
5 COPY target/vprofile-v2.war /usr/local/tomcat/webapps/ROOT.war
6
7 EXPOSE 8080
8 CMD ["catalina.sh", "run"]
9 WORKDIR /usr/local/tomcat/
10 VOLUME /usr/local/tomcat/webapps

```

### db\ Dockerfile

```

1 FROM mysql:5.7.25
2 LABEL "Project"="Vprofile"
3 LABEL "Author"="Imran"
4 ENV MYSQL_ROOT_PASSWORD=vprodbpass
5 ENV MYSQL_DATABASE=accounts
6
7 ADD db_backup.sql docker-entrypoint-initdb.d/db_backup.sql

```

1:15

## 22. Containerization 247. Docker Compose



We can build and run multiple containers at a time using docker compose. Using Vagrant Files , we can create and run multiple VM's , similarly , we can build images and run containers using them using docker-compose.

`docker-compose up -d` helpful to run the docker-compose in detached state , it means docker-compose runs in background , without locking the terminal , until completing the process.

7:10

## 22. Containerization 247. Docker Compose



```
1 | version: '3.8'
2 | services:
3 |   vprodbs:
4 |     build:
5 |       context: ./Docker-files/db
6 |       image: vprocontainers/vprofiledb
7 |       container_name: vprodbs
8 |     ports:
9 |       - "3306:3306"
10 |     volumes:
11 |       - vprodbsdata:/var/lib/mysql
12 |     environment:
13 |       - MYSQL_ROOT_PASSWORD=vprodbspass
14 |   vprocache01:
15 |     image: memcached
16 |     ports:
17 |       - "11211:11211"
18 |   vromq01:
19 |     image: rabbitmq
20 |     ports:
21 |       - "15672:15672"
22 |     environment:
23 |       - RABBITMQ_DEFAULT_USER=guest
24 |       - RABBITMQ_DEFAULT_PASS=guest
25 |   vroapp:
26 |     build:
27 |       context: ./Docker-files/app
28 |       image: vprocontainers/vprofileapp
29 |       container_name: vroapp
30 |     ports:
31 |       - "8080:8080"
32 |     volumes:
33 |       - vroappdata:/usr/local/tomcat/webapps
```

8:38

## 22. Containerization 247. Docker Compose



In here , we can see , that vprodb and redis are the names of the docker containers which are created based on the image "vprocontainers/vprofiledb" , and the image is built based on the context mentioned. And we are declaring the port mapping under here with volume mapping , and declaring some required environmental variable. Similarly , we do for other containers like redis , rabbitmq , memcache, elastic search etc..

All these details will be present under `application.properties` under  
`src/main/resources`

11:04

## 22. Containerization 247. Docker Compose

```
1  vproweb:
2    build:
3      context: ./Docker-files/web
4      image: vprocontainers/vprofileweb
5      container_name: vproweb
6      ports:
7        - "80:80"
8    volumes:
9      vprodadata: {}
10     vproappdata: {}
```

We declare volumes at last of the docker-compose file , but this is the first block to execute in docker-compose file , first docker-engine gonna create these volumes and later map the volumes furthur.

2:19

## 22. Containerization 248. Build and Run

`docker compose build` refers the docker-compose.yml and build all the images by referring the Dockerfile context

If we use the `docker compose up -d` , we build all the images by referring to docker-compose.yml and later start the containers based on those images. We start the networks , volumes and all the objects which are present under docker-compose.yml

We see the containers running by `docker ps` command or through `docker compose ps` command.

`docker compose ps` gives only the composed containers , whereas `docker ps` , gives the details of all the containers.

So , if everything is Ok , then all the containers will start , and later , if we enter our VM's IP address in the browser then the website should pick up.

If we wanted to push image to the dockerhub , we do `docker login` through terminal , enter username and password , after successful login , `docker push` the image that we wanted to push to the repository.

`docker compose down` stops and removes containers.

8:10

## 22. Containerization 248. Build and Run



`docker system prune -a` Clear up the images and containers , if they are in stopped or exited state. Writing docker-compose.yml with multiple Dockerfiles for images is the preferred way of containerizing the application. We must get the setup document for containerizing the application , version and package details , how we are building and running the application from the developers.

Ensure the build image and the app image to be separated , where all the build tools will be present in the first part , and only the artifact from the first part is transferred to the second part on another base image , this saves us image space , as unnecessary build tools will not be present , these will be removed from the image , as soon as the first part of the image ends.