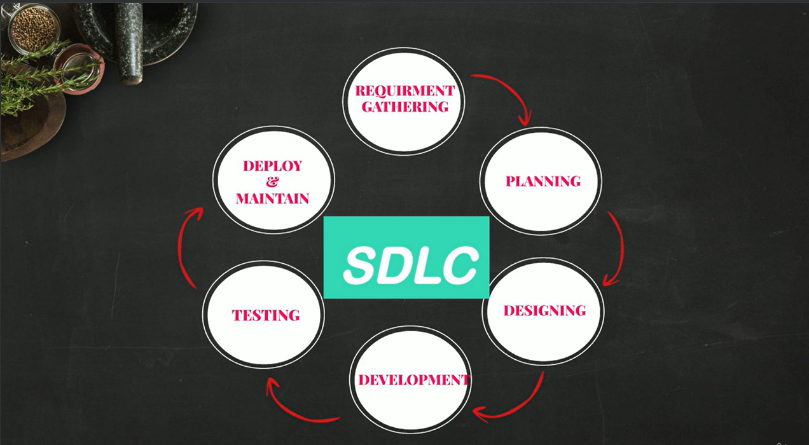
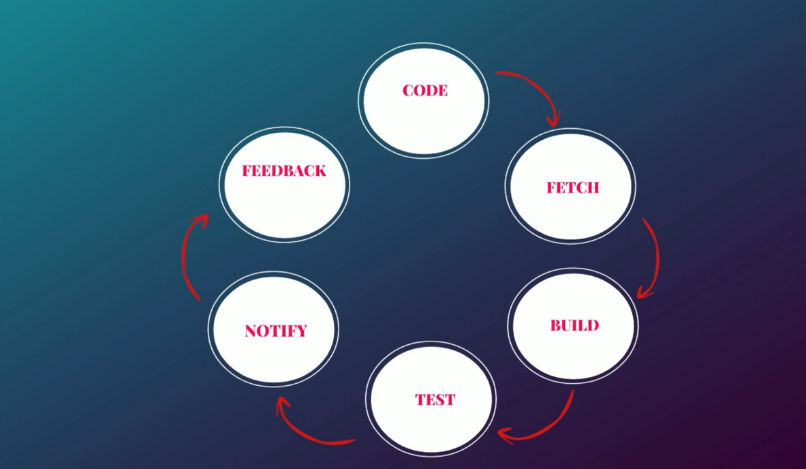
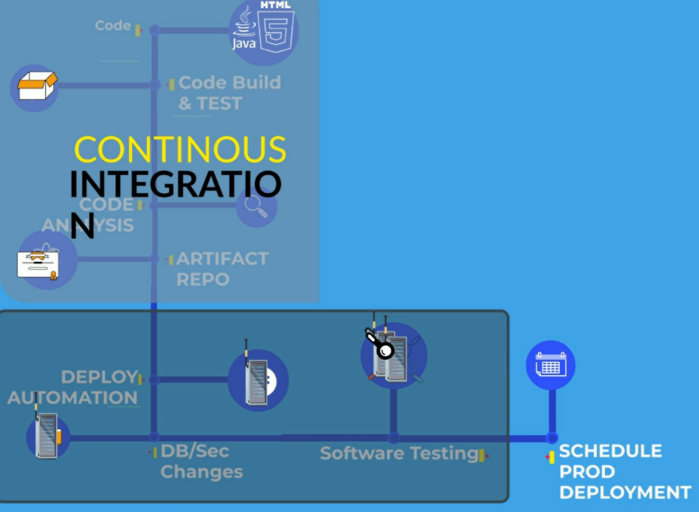
**Devops Zero to Hero**

* Data gathering > Plannning < Design architecture < Developers write the code < Software testing < Deployment for use < Maintenance.
* This entire process is called SDLC( Software development Life cycle).
* Models of SDLC are **Waterfall, Agile, spiral and big bang.**
* **Waterfall model** takes more time as the after completing each phase only the next phase begins.
* **Agile** takes less time compared to waterfall as the stages are divided and planned to complete early. It has regular code changes and testing. Many iterations causes the failures during the deployment.
* Dev + Ops (Development and operations)
* Dev is agile. It has regular & quick changes. Operations is ITIL driven like waterfall model, it is stable.
* Developers toss the completed code and the ops team deploys the code. Due to frequent changes in code and unclear instructions the ops team faces severe issues during the deployment.
* Automation also plays an important role in Devops for code writing, testing, deployment.
* Everything is automated in devops.

**Continuous Integration:**

* The developers create a code in IDE like Eclipse, Visual studio, atom and send it to GITHUB( a version control system) to store it and for retrieve it.

Continuous Integration

* The tester takes the code and tests it. If he founds any bugs then the code is sent back to the developer. This takes lots of time.
* The testing is done by Jenkins. It is continuous integration tool.
* But using the automation tools the code is tested by this and the feedback about the errors and bugs is sent to the developer for immediate changes.
* This entire automated process is called Continuous integration. It helps in detecting the defects at very early stage.
* Build tools > Software repositories > CI tools like Jenkins.
* The deployment of the code to the servers can also be automated.
* By this way the testing the deployment of the code is done easily and takes less time.
* Automating all the steps and linking them one by one gives a Continuous delivery Automation.

**Prerequisite Tools:**

* Git hub repositories

**Day 1**

**What is devops?**

* A culture or practice that improves the organization ability to deliver the applications. (a process of improving quick delivery).
* Devops is also called CI/CD (Continuous Integration and Continuous Delivery). Devops is improve the delivery using automation with good quality along with monitoring.
* **Improving delivery, automation, quality, monitoring and testing.**

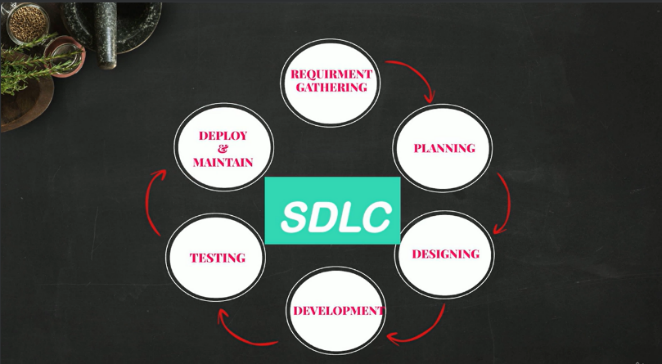
Or

* Devops is process of improving the application delivery by ensuring there is proper automation, with good quality, there is a continuous monitoring and continuous testing.
* By this the time for which a project takes 2 weeks gets reduced to 1 week.

**Why is devops?**

* Previously we have different people for each work
* **System administrators** creates a server, **Server administrators** creates the app server to deploy the application into server and **BRE** deploys application into the server. All these people work together to do all the things which takes more time and requires communication between these people.
* Here the devops will automate all the processes thus reducing the time to deliver the applications to the client.

**Software Development Life Cycle: SDLC**

* A set of standards or process that is followed by software industry to **design, develop and test to deliver the high quality product**.
* The end goal is to deliver the high quality product.
* Example.com we are developing an e-commerce application.

It includes steps like **Planning, Designing, Developing, Testing and**

**Deploying.**

The application was completed with men and women section.

Now they want to include kids section they will follow the same steps.

**Planning:** based on the requirements and feedback from the customers the planning is done.

**Defining:**  write the requirements in document which will be software requirements document.

**Designing: HLD** (High level design) and LLD (low level design), high level includes the scalability.

**Development:** writes the code based on the design and moves it to source code repository (GIThub).

**Testing:** application code from repository is taken and deployed in a server. The QA team tests the software application.

**Deployment:** promote to the production and to the customer.

Here the devops comes to picture and automates the **development, testing and deployment** of the application to increase the efficiency.

**Devops writes the automation code to automate the process.**

**Day 2**

**Virtual Machines(VM)**

**Serve**r: A hardware computer system that provides services like deployment, testing etc.

**Hypervisor**: A software that can install virtual machines on your physical servers or computers.

* We install hypervisor on servers to create virtual machines by which the server efficiency can be enhanced.
* We can create multiple servers using single server. It is a logical partitioning of the servers.
* Popular Hypervisors are VMware and Xen

**Hypervisor Server Server**

The hypervisor splits the physical server into virtual machines because of which we got five servers. The resources are split among these.

As we know the AWS provides the data centers for the pay on use. The Amazon built many data centers across the globe. The data centers have hundreds and thousands of physical servers installed in them.

If I need a server with 10GB RAM and 12 processors then the AWS informs the hypervisor installed in the physical server to logically partitioning the server or called as an Instance.

This instance is the virtual server we have for our work provided with IP address, ram and processors.

**AWS EC2 API:**

* In your organization if you require 100 virtual machines daily the creating the virtual machine manually is difficult.
* For this the AWS has automated the virtualization to increase the efficiency. This automation feature is called AWS API.
* The developer at AWS writes the API for automation.
* The Devops engineer will write a script which will call the AWS API EC2 to create 10 VMs or 1 VM based on the requirement. By this the Devops engineer automates the process which will help in avoiding the errors and saves the time.

EC2

**Script request Valid**

**Instances Authorization**

**Authentication**

* The devops engineer will write the script and that script will request the AWS API.
* The request should be valid with authorization and must be authenticated.
* Then only the instance will be created and sent to the client as output.
* Valid: following all the standards this ec2 instance requires, authenticated: user should have access to resources, Authorized: should have permission to use.

**SCRIPT:**

* AWS CLI, AWS API (boto 3), AWS CDK(cloud development kit), AWS CFT (cloud formation template) and Terraform
* Terraform is used for automating multiple cloud providers. It is open source.
* For AWS mostly AWS scripts are used. Azure has its own Scripts.
* The scripts communicate with the AWS API to create an instance which will be automated.

**Important Point:**

Difference between Access Keys and Public key & private key.

**Access Key & Secret Access key:** It is sued to login into the AWS console or AWS account through CLI.

**Public key & Private Key**: These are used to login into the Instance or Virtual Machine.

**Login to EC2 instance using pem file:(public and private key)**

* First install AWS CLI in your system and open CMD and run command aws –version.
* To login to AWS instance you need to save the .pem file in the desktop location.

**Commands:**

ssh -i (path of the pem file) ec2-user@(public ip)

#(give the permissions to user only using chmod)

ssh -I /user/keys/sai-kp.pem [ec2-user@31.51.168.246](mailto:ec2-user@31.51.168.246)

chmod 600 /user/keys/sai-kp.pem (changes permission)

thus you will be logged into the instance

**Login to AWS console using Access key:(access key and secret access key)**

**You can create the access key at the security and credentials of the AWS account. Download the csv file.**

* Check for aws version install aws cli lastest version

**Commands:**

aws configure

access key:

secret access key:

region:

format: json

aws ec2 describe

By this way you can login to aws console through cli and get details of all services provided by the AWS.

To write the script we have AWS CLI documentation where the complete details are provided.

**Day-6**

**LINUX:**

* Linux is free operating system or open-source software. It is very secure it does not require any anti-virus.
* Distributions of Linux are RHEL, Ubuntu, CentOS, Fedora and Debian. Linux is fast in production.
* It is best for the production.

User Processes

Compilers

System

Software

Operating System

System Libraries

Kernel

* Kernel is the heart of the operating system. The kernel is responsible for the communication b/w your hardware and software. Device, Memory, Process management and handling system calls. These are the responsibility of kernel.
* Libraries are responsible for performing the task.

**SHELL SCRIPTING:**

* It is a way through which we can communicate with the OS. This is by using CLI as there is no GUI available in production server.

**Popular Shell Command:**

* ls- list the content
* pwd- present working directory
* cd (name) – change directory followed by directory name
* clear- make space
* ll- lists all info of the files and directories
* ls -ltr – lists info of the files with timestamp

LS command shows the timestamp, file type, file permissions, size, ownership etc.

**Create a file:**

* Touch filename- creates an empty file.
* tat > file name – creates a file with name.
* vi filename – create with file name and can edit the contents ( :wq! To save file).
* cat filename – to view the contents of the file
* rm filename– remove a file

**Create a Directory:**

* mkdir directoryname – create a directory
* rm -r directoryname – remove a directory

**Memory check:**

* free -g – memory of the server.
* nproc – no. of CPU
* df -h – disk size and usage
* top – complete info of the memory, cpu and disks.

**AWS Cognizant login details:**

**ACCOUNT ALIAS:** academy-lab

**USERNAME:** 2298870

**PASSWORD:** 0yG!PW3FsUFq9+

**Day-7**

**Shell Scripting:**

* First thing to write a shell script is to create a file with extension .sh. (test.sh)
* A shell script always starts with #!/bin/bash “#!” is called she bang.
* We can use bash,sh,dash,ksh are the executables depending upon the OS.
* These have only small syntax differences and mostly used is bash(born again shell).
* Mostly use dash or bash. Previously the sh is redirecting to bash so they used the bash.

But in recent years the dash is mostly used by default.

**Shell Commands:**

**1. echo:** To write a print statement.

**Ex**: echo “My name is kumar”

**2. cat:** to view the content of the files. Ex: cat filename

**3. sh filename:** This command will execute the file or executable shell script file.

**4. ./filename:** This command will execute the file or executable shell script file.

**5. chmod:** Helps to change the permissions of the file to users, groups and others by two ways.

**Ex:** chmod u+x filename (gives users execute permission) or chmod 700 filename (gives users r,w,x permissions)

R=4, W=2 & X=1

**6. nproc :**  To liosts the processors in the vm

**7. free:** to view the memory available

**8. top :** to view the processes running and memory available and cpu usage.

**9. df -h :** to view the disk space

**10. ctrl + c :** close the executing shell script.

**11. ctrl + D :** does something or exit

**12. ps -ef:** to view all the processes that are running in the vm.(child , zombie, processes ID , etc)

**13. ps -ef | grep “ ” :** this gives the process that starts with name provided in the grep.

**14. | :** This helps in sending the output of first command to second command.

**15. awk:**  It can filter out the output like a specific column or a row.

**16. grep “text” filename :** gives the entire row in which the text is present.

**17 set -e:** exit the script when there is an error. It does not find error in pipe command.

**18. set -o pipefail:** Any error in pipe can be shown and stops the code execution.

**19. wget (website url)** : downloads the logfiles.

**20. kill -9 (process name or id):** this kills the process

21. **find / -name filename:** search the files with that name

**Script 1 : vim script.sh**

Create a directory with two files in it and changing the permissions.

**vim script1.sh** (a file is created and opened for writing the script)

1. #!/bin/bash
2. #create a directory (comments)
3. mkdir saikumar
4. cd saikumar
5. #create files
6. touch file1 file2
7. chmod 700 file1
8. chmod u+x file2
9. #end of the script

./script1.sh (to run the script)

Output: As the script runs it creates a directory and move into the directory. Now it creates the two files and changes their permissions.

Mostly the Linux is used in devops because it has security , easy for automation and light weight.

* **Script 2 : vim nodestat.sh**

Create a script that gives the status of the node health. (memory, process, disk space)

1. #!/bin/bash
2. ####################################
3. #Author: Sai Kumar
4. #Date: 08-04-2024
5. #This script outputs the node health
6. #version: v1
7. ###################################### (this is the meta data)
8. echo “This is the disk space”
9. df -h
10. echo “This is the memory space”
11. free -g
12. echo “This is the processor details”
13. nproc
14. ## end of script

This helps in easy understanding of the output. But another way is there by using debug mode.

1. #!/bin/bash
2. ####################################
3. #Author: Sai Kumar
4. #Date: 08-04-2024
5. #This script outputs the node health
6. #version: v1
7. ###################################### (this is the meta data)
8. set -x #debugg mode
9. set -e #this exit when there is an error
10. set -o pipefail
11. df -h
12. free -g
13. nproc
14. ## end of script

Output: Before every command output the command will be shown. We can use the both practices.

As we used set -e helps in finding the errors after execution. If any error occurs then the execution will stop. But it cannot find the error in pipe(|). So we should use the set -o pipefail.

* **Script 3 : vim process\_id.sh**

1. ps -ef | grep “amazon”

root 2565 1 0 00:08 ? 00:00:00 /snap/amazon-ssm-agent/7983/amazon-ssm-agent

ubuntu 2877 2744 0 06:05 pts/0 00:00:00 grep --color=auto amazon

Here the process related to amazon will appear

1. ps -ef | grep “amazon” | awk -F” ” ‘ {print $2} ’

2565

2874

Here as we used the awk command the specified column where the process id is there will be printed.(second column)

**Log Files:**

Mostly log files are used to check the errors in the system or code. But they are huge, so they get stored in s3 or google drive or azure block store or github. So we use one command called curl.

* curl (website address) | grep “text”

The website address is the place where the logs are stored. The output gives the logs that have the ”text” in it.

* curl -X GET api.foo.com : to request information from that website.

* wget (website url) : this downloads the log files from that location and save in your system.

Here if we use curl then we can get the information in logs from that website.

But if we use the wget then we can download the information in logs from the website

* **sudo su -** : this will take us to the root user. “Sudo” means we run commands on behalf of root user. "su” means switch user.
* **sudo find / -name filename** : this finds all the files that has the same filename.

**Trap:**

This command is special because it traps the signals from the keyboard to the linux machine.

For example if you have executed a process to view the contents of large file then you will want to stop by using the ctrl + c. So if you use the trap in your script then the signal from keyboard will not reach the machine.

trap “echo do not use ctrl+c “ SIGINT

this is a command where the input signal from key board gets trapped. Each signal has a signal ID. SIGINT this is for signal interrupt.

**Loops in Shell script:**

**Ifelse:**

1. #!/bin/bash
2. a=1
3. b=10
4. if [$a > $b]
5. then
6. echo “a is greater than b”
7. Else
8. echo “b is greater than a”
9. Fi

**For:**

1. #!/bin/bash
2. for i in {1.100}; do echo $1; done

prints from 1 to 100

**Shell Script for printing numbers divisible by 3,5 and not divisible by 15.**

Range : 1 to 100

1. #!/bin/bash
2. ##########
3. #
4. #
5. ##########
6. For i in {1..100};do
7. If ([‘expr $i % 3’ == 0] && [‘expr $i % 5’ == 0]) && [‘expr $i % 15’ != 0];
8. then
9. echo “$i”
10. Fi;
11. Done

Prints the output that number which are divisible by 3 and 5 and not divisible by 15.

**Print the number of S in mississipi**:

1. #!/bin/bash
2. ##########
3. x= mississipi
4. grep -o “s” <<<”$x” | wc -l
5. ###end of script

Here the grep -o is used to only read one letter and that is taken as input from $x and wc -l is for the word count(letters).

1. #!/bin/bash
2. echo “enter the name” #for the std input from user
3. read x
4. grep -o “s” <<<”$x” | wc -l
5. ###end of script

**User Input:**

**read** name is the command used to read the inputs from the user.

1. echo enter the names
2. **read** name
3. **echo $name**

Ouput: user input is printed as output.

**AWS**

In older days companies faced issues with buying servers, issue of traffic and monitoring & maintaining.

* Servers are costly, troubleshooting problems, servers are idle most of the time.

Why cloud?

A lot of data is being generated and that has to be stored somewhere. To store it we need some digital space using the servers. Cloud provides us that space.

A data center is a space where you can store your data. This cloud is a collection of data centers. The data centers are connected together through a network so that the user can access them and use them efficiently.

When you take a cloud space you need not have to worry about the servers, functionality and errors.

We can focus on the business.

**Cloud computing:**

The cloud computing is storing data on remote servers, processing data from servers and accessing the data via internet.

Through cloud computing we can access the data from anywhere in the world and can manage the applications for that all we need is a good internet connection.

**Service models**:

1. SaaS:

Cloud provider leases an application or a software for the user. Ex: Gmail. We just have to use it that’s all.

1. PaaS:

A platform is provided for the users to create our own applications. Ex. Google app engine. Where we can create and let others use these applications.

1. IaaS

An infrastructure is provided where we can use any operating system, technology and applications we want to build. Required resources are provided.

**Deployment Models:**

1. Public cloud

The service provider make the applications, storage etc available to the general public over the internet. Ex. Google drive.

1. Private cloud

Private companies uses these models for few number of people. I has security feature to restrict unauthorized access.

1. Hybrid cloud

This is a mix of both private and public cloud.

Cloud providers are GCP, AWS, Azure

**AWS:**

It is a secure cloud service platform offering compute power, data storage, content delivery and etc. It is six times more compute power than other cloud services.

Advantage of AWS:

* Flexibility
* Cost effective
* Scalability( easy to increase and decrease the usage)
* Security

**AWS architecture:**

AWS infrastructure is divided into Regions and Availability Zones.

Regions: Different locations across the world. Multiple data centers are there in regions.

Availability zones: The data centers in the regions are called availability zones.

Each region(country) has multiple availability zones(areas).

**Domains of AWS:**

* Compute

1. EC2(Elastic cloud compute)
2. Elastic bean stack has pre-defined libraries.

* Migration
* Security and compliance
* Storage (executable files)
* Networking
* Messaging
* Database (data)
* Management tools

**Instance:**

An instance is a virtual server for running applications on EC2. A tiny part of a computer that has its own OS, Hard drive, network connection, etc.

**EC2:**

A web service from amazon that provides resizable compute services.

**AWS Project Using Shell Scripting:**

To maintain cost effective for which we have to track the resource usage.

**Monitoring resource usage:**

* EC2, S3, lambda and IAM
* Report to manager at 6pm
* Using Shell script create a file and it should have all resource information.
* If we are not available at that time we can integrate the shell script with cron job.
* Cron job will automatically do the job for it. It will automatically execute the shell script.
* Cron job is available in Linux. Now use AWS CLI and integrate with cron job.

**Day-7**

**Script that gives the usage report of the AWS resources:**

So here the usage report has to be sent to manager everyday at 6pm. For this we can write a shell script and run the script by creating a Cron Job. The Cron Job runs in Linux machine and executes the shell script at desired time.

Commands:

* aws configure

access key copy paste

secret key copy paste

default region enter

default output Jason enter

* vim aws\_resource\_tracker.sh
* insert mode

#!/bin/bash

# Author: self

# date: 11th jan

#version:v1

#this script will report the aws resource usage

* set -x (debug mode)

#AWS s3

#AWS EC2  
#AWS LAMBDA

#AWS IAM users

# list s3 buckets

* echo “Print list of s3 buckets”
* aws s3 ls > resourceTracker

# list EC2 instances

* echo “Print list of ec2 instances”
* aws ec2 describe-instances >> resourceTracker

# list lambda functions

* echo “Print list of lambda functions”
* aws lambda list-functions >> resourceTracker

# list IAM users

* echo “Print list of IAM users”
* aws iam list-users >> resourceTracker
* :wq! (save and quit)

(out of vim)

* chmod 777 aws\_resource\_tracker.sh

executing command - ./(filename)

* ./aws\_resource\_tracker.sh | more

Output will be generated

(This command will give you the instance id only, we can replace this inside the file)

* aws ec2 describe-instances | jq ‘.Resevations[]. Instances[].InstanceId’

this command where jq is the parser of json which filters the instance ID.

Now we have to integrate this with cron tab

Here we have used the resourceTracker file so that the output of the job will be stored in that particular file.

**Day-8**

**PROJECT 2:**

* To know the people who can access the git hub repository. We write a shell script and integrate it with git hub. We can use API or CLI to integrate with github. We use API with github.
* In general we communicate with github through user interface like graphical interface. But we can communicate with the git hub with program also.
* We can use modules like curl, requests or postman through http.

We can Communicate to the applications in two ways:

1. API (Application Programmig Interface) (uses a either python or shell script)
2. CLI (Command Line Interface)
3. User Interface

**Shell Script lists the users of the GitHub:**

1. #!/bin/bash
2. # GitHub API URL
3. API\_URL="https://api.github.com"
4. # GitHub username and personal access token
5. USERNAME=$username
6. TOKEN=$token
7. # User and Repository information
8. REPO\_OWNER=$1
9. REPO\_NAME=$2
10. # Function to make a GET request to the GitHub API
11. function github\_api\_get {
12. local endpoint="$1"
13. local url="${API\_URL}/${endpoint}"
14. # Send a GET request to the GitHub API with authentication
15. curl -s -u "${USERNAME}:${TOKEN}" "$url"
16. }
17. # Function to list users with read access to the repository
18. function list\_users\_with\_read\_access {
19. local endpoint="repos/${REPO\_OWNER}/${REPO\_NAME}/collaborators"
20. # Fetch the list of collaborators on the repository
21. collaborators="$(github\_api\_get "$endpoint" | jq -r '.[] | select(.permissions.pull == true) | .login')"
22. # Display the list of collaborators with read access
23. if [[ -z "$collaborators" ]]; then
24. echo "No users with read access found for ${REPO\_OWNER}/${REPO\_NAME}."
25. else
26. echo "Users with read access to ${REPO\_OWNER}/${REPO\_NAME}:"
27. echo "$collaborators"
28. fi
29. }
30. # Main script
31. echo "Listing users with read access to ${REPO\_OWNER}/${REPO\_NAME}..."
32. list\_users\_with\_read\_access

Output:

Gives the usernames who have access to the github repository.

Basically this script is stored in the git hub repository. So we need to clone the github repo into the local disk and run the script.

Commands:

1. git clone url
2. export username=”ch-saikumar-720”
3. export token=” ” (this token is generated in the github)

We need to install jq pacer for the json request for this use the command “sudo apt install jq -y”.

**Important:**

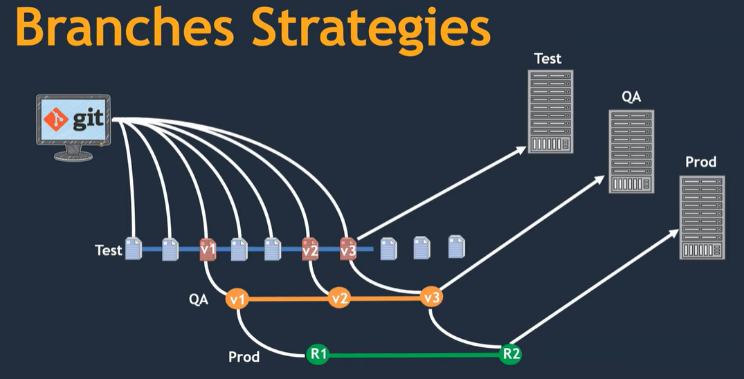
This script has command line arguments. So we need to give the organization name and repository name.

./listusers\_script.sh devops-by-script list\_users

Here the develops-by-script is the organization name and list\_users is the repository name.

**Day-9,10**

**Git & GitHub and Branching strategy are in another pdf**

**The commits done in the branch are isolated from the other branches.**

**Additional:**

* git add && git commit -m “” && git push
* git remote add “url of git hub account”
* git remote -v (to check the remote repos)
* git branch (to view the branches in the repo)
* git log (to view the commits of the repo)
* git log branchname (to get the logs from that branch)
* git checkout -b branchname (creates a branch from that point)
* git checkout branchname (used to switch to that branch)
* git checkout branchname && git log (gives logs of that branch and opens)

**After all changes the branches must be combined to sent for the release.**

Here the commits we done will be added to the main branch or other branches using the commit id. For this we use

* cherry-pick
* git cherry-pick commitid

the commit will be added to the branch in which you are using this command.

You can cross check using git log. It is for few commits.

* Merge
* git merge branchname (the commits will be added to the branch in which you are using this command.) changes will updated at top irrespective of commit time.
* git log (to cross check)
* Rebase
* git rebase branchname (the commits will be added to the branch in which you are using this command.) (it will an organized one)
* git log (to cross check)

**Day-13**

**AWS services for Devops:**

* EC2 Instance
* VPC (private cloud)
* EBS (volume)
* S3 (Storage)
* IAM
* Cloud Watch (Monitoring service in AWS)
* Cloud Trail (saves the logs of api for auditing)
* Lambda (a function to mail send notifications)
* Cloud Build Services

AWS also provides the CI/CD services ex: code build, code pipeline, code commit, code deploy

* AWS Configuration service
* Billing and Costing
* AWS KMS (encryption keys store, certificates)
* AWS EKS
* Fargate , ECS
* ELK (elastic search)

**Serverless:**

A lambda is a serverless compute i.e without a server. We can directly run the program in lambda because the AWS will do the background work. Once the code is executed then the lambda is terminated. Here the AWS will do the work behind like creating server, os, networking, etc.

**Day-14**

**Configuration Management:**

Managing the configuration of the server or infrastructure.

**Need for configuration management: (Problem)**

Previously we had on premises servers or data centers. These servers are managed by the system admins. The activities performed by the system admins are Upgrades, security patches and installation of softwares.

All servers are of different OS in it. So, to do the upgrades we need to login to each server and do them manually which is highly impossible if we have 100 to 500 servers.

Even after the cloud is introduced it became difficult because the no. of servers increased to 10000 and also many microservices are orginated.

If we try to do them using the shell script or power shell or python is also difficult.

For this need we need an application which will do the scripting or power shell script based on the OS and OS distributions. Commands vary with the OS distribution.

Configuration management will solve this problems.

**Tools:** Puppet, Ansible, Salt and Chef are the popular configuration management tools.

|  |  |
| --- | --- |
| Puppet | Ansible |
| Pull mode  Master-slave architecture. | Push model  Agentless model. Simply we put the names of servers or IP in an inventory file. Or password less authentication |
| To do anything we need to configure the instances as slaves and one master. Like jenkins. | During the peak loads this is the best approach. This has Dynamic inventory where ansible will auto detect the servers and store its details. |
| A little bit difficult | Support with windows and linux is easy |
| Uses puppet language | Uses yaml language which is simple. |

**Ansible is agentless model where we doesn’t need any agents in the target servers like jenkins,puppet, etc.**

**Disadvantages of Ansible:**

* With windows it was a difficult.
* Debugging is also difficult. Difficulty in understanding the logs.
* Performance is little bit low.

We create playbooks in Ansible.

Ansible supports linux using protocol SSH. It also supports windows using protocol win RM.

Ansible supports all cloud providers. It doesn’t care about the cloud providers. It require only the shh or win rm allowed and public IP address of the server.

**Day-15**

So in ansible we don’t have master-slave concept for this we need to do is a password less authentication for the main server to the target servers.

**Password Less Authentication:**

1. Create two virtual machines as Ansible and target server of Ubuntu OS.
2. Login to the Ansible server through mobaxterm.

sudo apt update

sudo apt install ansible

ansible –version

ssh (ip of target server) (to login to server)

#requests for the password. But we need passwordless authentication.

ssh-keygen (generates a public and private key)

1. Go to the location of .ssh/id\_rsa and open the id\_rsa.pub and copy it.
2. Now login to the target server

ssh-keygen

ls ~/.ssh/

vim authorized\_keys

paste the public key of the ansible server in here and save.

1. Now open the Ansible server

ssh (ip of target server)

Now you can login to the target server without password.

To logout from second server use command “logout”. This take you back to the previous logged in server.

In this way you can do to all the servers you want to configure. Copy paste the public key of ansible server in the target servers.

**Ansible Playbooks or Files:**

In shell scripting we write shell script files and in python we write python files. Similarly in ansible the files are called Playbooks.

* Ansible adhoc commands are simply used for performing one or two tasks on target servers.

Ex: ansible -i inventory all -m “shell” -a “touch devops\_class”

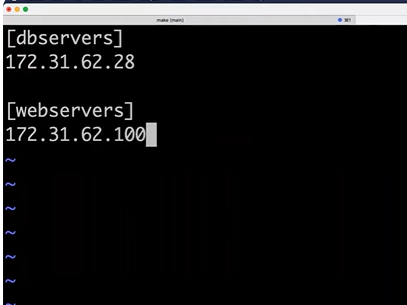
ansible -i inventory all -m “shell” -a “nproc”

ansible -i inventory all -m “shell” -a “df -h”

The file name devops\_class is created in all the servers that are listed in the inventory file of the main ansible server.

* Ansible playbook is used for performing multiple tasks on target server.

**Adding the servers into inventory file:**

* Login to the ansible server
* After installing ansible an inventory file is created where the IP addresses of the target servers are stored.
* We can group them into one.
* If we want to configure or execute on a particular servers of a group then the grouping is helpful.

ansible -i inventory groupname -m “shell” -a “df -h”

Here m represents module and a represents arguments to be passed.

**Inventory file**:

We create the inventory file in the directory where the play book is saved. Use the command “vim inventory” to create a file and add the private IP of the target server in the file.

If you run the adhoc commands or playbook then it will work.

**First Playbook for installing Ngnix and start the services:**

* Open the server where the ansible is installed.
* Create a file naming first-playbook.yml (the extension is for the yaml files)

1. vim first-playbook.yml

--- (indicate a yaml file)

* name: Install and start Ngnix (name of play book)

hosts: all (all servers)

become: root or true

tasks:

- name: Install Ngnix

apt:

name: ngnix

state: present (this command is equal to shell: apt install ngnix)

- name: Start Ngnix

service:

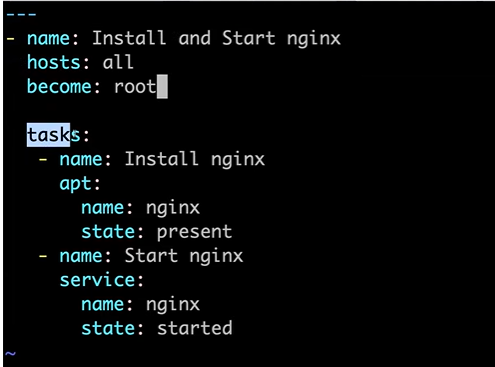
name: ngnix

state: started ( this is equal to shell: systemctl start ngnix)

* name : second playbook (this will be the second playbook)

#ansible-playbook -i inventory first-playbook.yml (To execute the playbook)

Check in the target server verify the installation of Ngnix and its status.

1. Here the apt, shell and service are the modules

**Ansible Galaxy Roles:**

It is the efficient way of writing ansible playbooks that will improve the efficiency in writing complex playbooks.

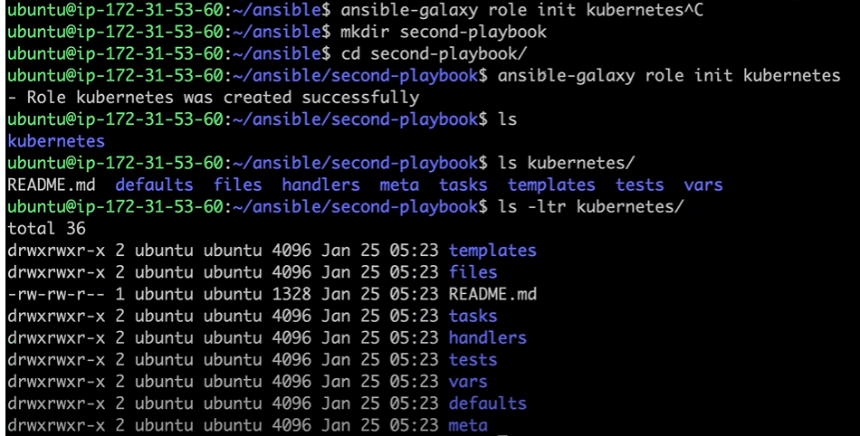
In some events we need to write complex playbooks. For some we need to write 50 to 60 tasks and it will increase the complexity of the playbook.

To start using the roles we need the commands

1. mkdir second-playbook
2. cd second-playbook/
3. ansible-galaxy role init Kubernetes

(a folder named Kubernetes will be created)

1. ls -ltr Kubernetes

gives some files

Here we write a simple yaml file that has the roles and inside the roles we have different folders.

**Task folder:** It will contain an yaml file that has tasks in it.

**Meta:** It is used to write the meta data information

**Defaults:** It stores variables

**Vars:** it is also used for storing variables

**Handlers:** For writing the exceptions so that the playbook can run smoothly.

**Readme.md**: It contains all the details of the playbook and logs

**Files:** To store certificates and transferring of the content.

**Day-16**

**Infrastructure as Code:**

Creating the infrastructure using the code (IAC) and API’s.

**Problem:**

Suppose we need 100 instances from AWS. For this we simply use the AWS CFT or CDK to write a script which will automate the creation of the instances as per the requirements such as CPU, compute power, storage and Inbound and outbound traffic.

For Azure if we need to do the same, then it has Azure Resource Manager where we do the similar thing. If we have on-premises servers the we need open stack to create virtual machines and it uses heat templates to create the infrastructure.

1. Here we cannot use the same code and the service provider is different. We need to write different codes as per the syntax related to that particular cloud service (AWS, Azure & GCP).
2. Also now-a-days most of the companies are using the hybrid cloud where the storage service is taken from AWS and Development services are taken from Azure.

The people who handles both of these infrastructure should have an idea of both the services and code syntaxes.

**Terraform:**

It is the tool that provides the API as code. The above mentioned two problems can be solved by the Terraform, where we code in the terraform and mention the cloud provider and it will automatically convert into API calls for that particular cloud provider.

**API: Application Interface**

We can communicate with applications programmatically. In UI (user Interface) we type the details of search and gets the result back from the google.

But if we write a script and runs it, then it will communicate with google programmatically and gets the results to you. This is called API.

Terraform has different modules related to services provided by the cloud providers.

The script will be written by the using the terraform and run the script. Then the terraform will convert the script into API calls and communicate with respective cloud provider for the resources.

Here it uses API as Code. It helps in

* Manage any Infrastructure
* Track your infrastructure
* Automate changes
* Standardize configuration
* Collaborate

Basic terraform commands:

* Terraform init (initialize)
* Terraform plan (dry run)
* Terraform apply
* Terraform destroy

Along with the terraform file for the IAC another two files called input.tf and output.tf files are created.

This helps in keeping the original code as it is and we can modify the inputs given like name, sg group, subnet, vpc, kp, etc.

The output file is like a details ofbthe created infrastructure like OS, size, location, etc.

To write a script in terraform we can use the hashicorp terraform website for the selected cloud providers.

**Install terraform in Ubuntu:**

wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg

echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb\_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list

sudo apt update && sudo apt install terraform

Install AWS CLI:

sudo snap install aws-cli –classic

aws --version

**Steps to create resources in AWS using terraform:**

1. We need the CLI and install the terraform using the below commands.

Sudo apt-get update && sudo apt-get install -y gnupg

Software-properties-common

Terraform --version

1. Now create a directory and move into it and create a file named main.tf
2. In the directory run the command “terraform init” to initialize the terraform. This is similar to git init in git hub.
3. Now write the script in the main.tf file and save it. We can use the hashicorp terraform aws documentation to write the script.
4. After that we need to login to the cloud provider account. In this we need to login to the AWS account through the CLI using the commands.

aws configure

access key:

secret access key:

We can generate the access keys in the AWS account at security and credentials.

1. Use command “terraform plan” this is used for the dry run of the code which give the sample output of the script.
2. Run the command “terraform apply” to create the resources in the logged in aws account.
3. The resource id will be generated at the bottom.

So here we created the resource using the terraform script but the user who requested for the resource should have the details of the id, network, key pairs, size, os type, etc.

For this we need a script that generates the output details of this resource. So we write a script in the output.tf file.

We can write the variables.tf file which can store the inputs to the main.tf file.

This is the best practice while writing the terraform script.

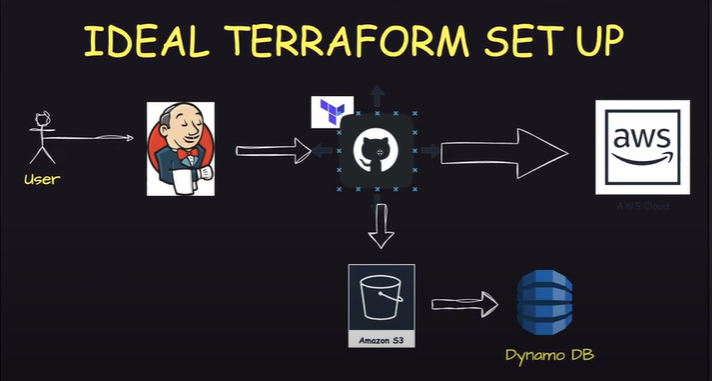
main.tf inputs.tf/variables.tf output.tf

**terraform.tfstate:**

This the file that tracks every event of the infrastructure. Just like .git file of the git hub the terraform keeps tracking the changes and events in it. Everything created by terraform will be tracked.

If we initialize the terraform in the local then the state file will be stored in the local. As it contains sensitive information about the infrastructure it cannot be pushed to the git repos.

We have to store these in a remote location, not in local machine. Do not manipulate the state file.

**Ideal Terraform Setup:**

* Here the admin requests for the resources by executing the Jenkins.
* The Jenkins will clone the terraform scripts from the git hub and build the code.
* This execution will create a resource in the AWS.
* The state file gets stored in the remote backend location. For which we use the S3 bucket to store the terraform.tfstate file.
* There are chances where the requests can come from multiple users at a single time which may confuse the state file. For this we use the Dynamo DB to execute single user request. This is a locking solution which is integrated with the terraform setup.
* The user gets the output of the resource details.

**Remote backend:**

We create the backend first and the details of the backend are added in the main.tf file.

In this way we can complete the above ideal terraform setup.

The S3 bucket and Dynamo DB table are created and the details of them are added in the main terraform script for resource creation.

**Modules:**

It is a way of writing reusable components. The code reusability is done through the modules which makes our tasks easy.

**Challenges faced with Terraform:**

* State file is single source of truth. (changes of corruption)
* Manual changes to cloud providers cannot be identified and auto-corrected. (cannot track changes done through console)
* Not a GitOps friendly tool. Don’t play well with Flux or Argo CD.
* Can become very complex and difficult to manage.
* Trying to position as a configuration management tool as well. (It is also doing the configuration management also)

**CI/CD by Jenkins**

Continuous Integration and Continuous Delivery

Jenkins will orchestrate the tools for the continuous integration and continuous delivery.

It will integrate all the tools and organize them. Manual intervention is reduced.

The pipeline is built using the Jenkins. Along with this we can move the applications to different environments like dev, prod, stage, etc.

It has a Master-Slave concept where the Jenkins is setup in one VM and nodes are set in different VMs which are the slaves.

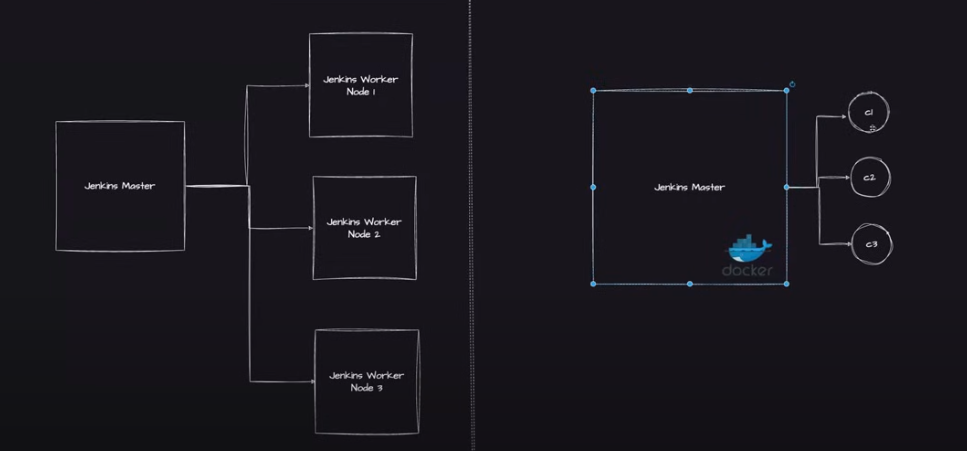
**Disadvantages of Jenkins:**

A scenario where we need to scale up or scale down the resources based on the demand like requirement of 1000 VMs. Here we need minimum 10 master nodes and rest will be worker nodes.

Based on the requests the worker nodes perform the operations. The worker nodes which doesn’t have any requests will remain idle. This is wastage of compute.

Now if we scale up all we may not have used all which is bad way.

For this we are using the Kubernetes and Docker.

**Jenkins Project using Docker agent:**

For different applications we use different worker nodes. It is because the if we have used the Master node for all applications the load increases.

But even when there is no load or zero requests we have to keep the worker nodes in running which is not an effective method.

As a solution for this we need can use the Docker agent to deploy the containers depending on the requests.

To upgrade version of the worker node you need to change the configurations in the master and it will be compactable with few applications only as the version changes some depending applications may not run properly.

A docker container is very light in weight. We can easily upgrade the docker image, can deploy and destroy it very easily. It also increases the efficiency.

Jenkins username: admin

Password: 720757

**Steps for project:**

* Login to the VM and follow the below commands to install the Jenkins in the server.

sudo apt update

sudo apt install openjdk-11-jre

curl -fsSL https://pkg.jenkins.io/debian/jenkins.io-2023.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 binary/ | sudo tee \

/etc/apt/sources.list.d/jenkins.list > /dev/null

sudo apt-get update

sudo apt-get install Jenkins

* Login to VM and install the Jenkins. Open port 8080 in the VM.
* Now open the Jenkins with <http://(IP> of VM):8080
* Use the “sudo cat (path from jenkins)” to view the password of Jenkins.
* Create the user and password to login.
* Now Install the docker

sudo apt update

sudo apt install docker.io

sudo su -

usermod -aG docker jenkins

usermod -aG docker ubuntu

systemctl restart docker

* After the docker installation the above commands will ensure that the Jenkins has access to the docker. To cross check

su – jenkins

Docker run hello-word

(If failed then logout from Jenkins user and run the below command)

usermod -aG docker Jenkins

After this follow the above and check.

* Now it is important to restart the Jenkins after the above commands usage. Now at the http path of Jenkins add /restart.

http://<ec2-instance-public-ip>:8080/restart

* Now install the docker pipeline plug-in in the Jenkins.
* Restart the Jenkins.

Now follow the Git hub repository of the abhishek verramallu and create pipeline.

**GitHub Actions**

This is also a CI/CD integration similar to the Jenkins. This is platform dependent which is GitHub.

You have to use the GitHub if your organizations depends only on the GitHub for the source code.

See Later as we have an idea about the Jenkins tool.

**Project Management in Devops**

What we do on the first week into devops?

* First thing we need to understand is Agile. It is most important than the tools.
* For faster delivery we use Agile methodology.

**Waterfall Model:**

We plan all the things and make a high level or low-level design. The developers start writing the code and testing is done. Finally, the application is deployed into the production.

**Agile Model:**

Here the entire project is divided into multiple microservices or chunks. Instead of developing all the code at once which takes at least 3 months we can complete each micro service and testing will be done.

**JIRA:** It is a tool to perform agile. Project management is done through JIRA. It is also used to track the project works.

**Confluence:** Knowledge sharing platform. The documentation related to any microservice or application created by your organization. Just like AWS documentation, Ansible, terraform documentation. The person developing the project will be documented for the reference.

**Share point**: It is also similar to Confluence.

**Service Now:** most of the service-based companies use this.

If there is any traffic, memory leak, issues etc. can reduce the application availability and efficiency. Like when application is not working properly. So there will be a monitoring system which checks everything and throughs an alert if it finds any abnormality or error. These must be addressed immediately and resolved.

The monitoring tools will be directly integrated with Servicenow API and create the incidents and sends to the support team for resolution.

**Incident Management:** the alerts caused by the errors will be created into an incident and sent to the support team. Depending on the impact of the error or alert the priority to resolve the incident will be increased.

**Change Management:** There will be some incidents that requires a major-changes in the code level or configuration level. So, during this time the applications availability is zero. The changes will be planned and each team is assigned with certain tasks. One team will set the blackout and remove the blackout, another team will do the code changes or configuration changes. During this the server availability to the client will be given by another deployment models.

**Read the docs:** Similar to the confluence but it is an open source platform. It can be easily integrated with git hub.

**Git & Github:** We can track the project using the github for the open-source tools. Similar to JIRA but open source.

**JIRA**