##

LINUX COMMANDS

==========

LINUX DEVOPS COMMANDS:

sudo :super user do this command is used to call the root user

sudo su root: switch to different users

pwd:present working directory

cd/: change directory

clear: To wipe the screen.

history: To display the recent executed commands

Date: To display current date & Time

ifconfig: to display ip address

cd ..:will take u one step Back.

ll : long list of details

ls: List of services

mkdir : To make Directory

touch<filename> : To create a new filename

cat<filename> : to read new filename

vi<filename> : To add content into new filename

cp :to copy and paste files

mv :to move And cut files

locate: To locate file in linux

ps :this is used to see currently running process and their PIDS

find: this is used to find the current directory

grep:this is used to search the text in given file

df: this is used to report on disk space

head -n1 filename this is used to find first line of sentense

tail -n1 filename :this is used to find last line of sentense

useradd:this this is used to add no.of users

userremove:this this is used to remove no.of users

hostname: this is used to display host

host -i: this is used to display IP address.

kill command :this is used to terminate the resposive programme.

job command :this is used to display all current jobs with their status

uniq:this command is used to remove duplicate lines

uniq sort :this is used to sort uniq names sort<filename> | uniq

##
MAVEN COMMANDS:
##
mvn clean:This command cleans the maven project by deleting the target directory. The command output relevant m essages are shown below.

mvn -help :This command prints the maven usage and all the available options for us to use

mvn compiler:This command compiles the java source classes of the maven project

mvn compiler:testCompile:This command compiles the test classes of the maven project.

mvn package:This command builds the maven project and packages them into a JAR, WAR, etc.

mvn install:This command builds the maven project and installs the project files (JAR, WAR, pom.xml, etc) to the lo cal repository.

mvn deploy:This command is used to deploy the artifact to the remote repository. The remote repository should be c onfigured properly in the project pom.xml file distributionManagement tag. The server entries in the maven settings. xml file is used to provide authentication details.

mvn validate:This command validates the maven project that everything is correct and all the necessary information is available.

mvn dependency:tree ==This command generates the dependency tree of the maven project.

mvn dependency:analyze == This command analyzes the maven project to identify the unused declared and used un declared dependencies. It’s useful in reducing the build size by identifying the unused dependencies and then remov e it from the pom.xml file.

mvn archetype:generate == Maven archetypes is a maven project templating toolkit. We can use this command to ge nerate a skeleton maven project of different types, such as JAR, web application, maven site, etc. Recommended Rea ding: Creating a Java Project using Maven Archetypes

mvn site:site == This command generates a site for the project. You will notice a “site” directory in the target after e xecuting this command. There will be multiple HTML files inside the site directory that provides information related to the project.

mvn test == This command is used to run the test cases of the project using the maven-surefire-plugin.

mvn compile == It's used to compile the source Java classes of the project.

mvn verify == This command build the project, runs all the test cases and run any checks on the results of the integra tion tests to ensure quality criteria are met.

mvn -f maven-example-jar/pom.xml package : This command is used to build a project from a different location. We are providing the pom.xml file location to build the project. It's useful when you have to run a maven build from a s cript.

mvn -o package : This command is used to run the maven build in the offline mode. It's useful when we have all the required JARs download in the local repository and we don't want Maven to look for any JARs in the remote reposit ory.

mvn -q package : Runs the maven build in the quiet mode, only the test cases results and errors are displayed.

mvn -X package : Prints the maven version and runs the build in the debug mode. It's opposite of the quiet mode and you will see a lot of debug messages in the console

mvn -v : Used to display the maven version information.

mvn -V package: This command prints the maven version and then continue with the build. It's equivalent to the co mmands mvn -v;mvn package.

mvn -DskipTests package: Therehe skipTests system property is used to skip the unit test cases from the build cycle. We can also use -Dmaven.test.skip=true to skip the test cases execution.

mvn -T 4 package : This command tells maven to run parallel builds using the specified thread count. It's useful in multiple module projects where modules can be built in parallel. It can reduce the build time of the project.
##

GIT COMMANDS:
##

GIT CONFIGURATION COMMANDS:

git config –global user.name: Set the username to be used for all actions

git config –global user.email:Set the email to be used for all the actions

git config –global alias. : Create a shortcut for the Git command.

git config –system core.editor : Set the text editor for all the command actions.

git config –global –edit : Open global configuration file in the text editor for manual editing.

git config –global color.ui auto : Enable helpful colourization of command line outputs.
##

Set Up a Git Repository:

git init : Initialize an empty Git repo in the current project.

git clone (Repo URL): Clone the repository from GitHub to the project folder.

git clone (Repo URL) (Folder): Clone the repository into a specific folder.

git remote add origin:
https://github.com/username/(repo_name).git Create a remote repo pointing on your existing GitHub repository.

git remote : Shows the name of remote repositories.

git remote -v: Shows the name and the URL of the remote repositories.

git remote rm (remote repo name): Removes the remote repository.

git remote set-url origin (git URL): Changes the URL of the repository.

git fetch: Get the latest changes from the origin but not merge.

git pull: Get the latest changes from the origin and merge them.

LOCAL FILES CHANGES:

git add (file name): Add the current changes to the file to staging.

git add . :Add the whole directory changes to staging (no delete files).

git add -A :Add all new, modified and deleted files to staging.

git rm (file_name): Removes the file and untracks (stop tracking) it.

git rm –cached (file_name): Untracks the current file.

git mv (file_name) (new_file_name) :Changes the filename and prepare it for Commit.

git checkout <deleted file name>: Recovers the deleted file and prepares it for Commit

git status :Shows the status of the modified files.

git ls-files –other –ignored –exclude-standard: Shows the list of all ignored files.

git diff :Shows unstaged changes in the index and the working directory.

git diff –staged: Shows file differences between staging and the last file version.

git diff (file_name): Shows changes in a single file compared to the last Commit.

##

DOCKER COMMANDS :

##

docker search : docker search MySQL

docker pull : docker pull --platform linux/x86_64 mysql

docker images: docker images

docker run: docker run --env MYSQL_ROOT_PASSWORD=my-secret-pw --detach mysql

docker ps: docker ps

docker ps --all

docker stop: docker stop f8c52bedeecc

docker restart:docker restart f8c52bedeecc(containerID)

docker rename: docker rename compassionate_fermi test_db

docker exec:docker exec -it test_db bash
mysql -uroot -pmy-secret-pw
SHOW DATABASES;

docker logs: docker logs test_db

docker rm: docker rm test_db

docker rmi: docker rmi eb0e825dc3cf

```
                ##############################################################################
                                    JENKINS ONE SCRIPT:
    ##############################################################################
    pipeline {
     agent {label 'master'}
    stages{
    stage("Demo"){
       steps{

          echo 'Hello World'
       }
    }
    stage("Source"){
       parallel{
          stage('CalibrationResults'){
             steps{
                echo 'Checking out CalibrationResults'
                checkout([$class: 'GitSCM', branches: [[name: '*/CI-CD-Demo']], doGenerateSubmoduleConfiguration
s: false, extensions: [[$class: 'CloneOption', depth: 0, noTags: true, reference: '', shallow: false, timeout: 60], [$class:
'SubmoduleOption', disableSubmodules: false, parentCredentials: false, recursiveSubmodules: true, reference: '', tim
eout: 60, trackingSubmodules: true], [$class: 'RelativeTargetDirectory', relativeTargetDir: 'server-core'],[$class: 'Che
ckoutOption', timeout: 60]], submoduleCfg: [], userRemoteConfigs: [[url: 'https://github.com/AtlasBID/CalibrationR
esults.git']]])
             }
          }
          stage('Combination'){

             steps{
                echo 'Checking out server spoke'
                checkout([$class: 'GitSCM', branches: [[name: '*/master']], doGenerateSubmoduleConfigurations: false,
 extensions: [[$class: 'CloneOption', depth: 0, noTags: true, reference: '', shallow: false, timeout: 60], [$class: 'Submo
duleOption', disableSubmodules: false, parentCredentials: false, recursiveSubmodules: true, reference: '', timeout: 60
, trackingSubmodules: true], [$class: 'RelativeTargetDirectory', relativeTargetDir: 'server-spoke'], [$class: 'Checkout
Option', timeout: 60]], submoduleCfg: [], userRemoteConfigs: [[url: 'https://github.com/AtlasBID/CombinationBuil
der.git']]])
```

```
          }
        }
      }

   }

}
}
```

```
################################################################################################################################
```

KUBERNATES COMMANDS

```
   ################################################################################################################################
```

KUBECTL CONTEXT AND CONFIGURATION

```
kubectl config view # Show Merged kubeconfig settings.

# use multiple kubeconfig files at the same time and view merged config
KUBECONFIG=~/.kube/config:~/.kube/kubconfig2

kubectl config view

# get the password for the e2e user
kubectl config view -o jsonpath='{.users[?(@.name == "e2e")].user.password}'

kubectl config view -o jsonpath='{.users[].name}'    # display the first user
kubectl config view -o jsonpath='{.users[*].name}'   # get a list of users
kubectl config get-contexts                          # display list of contexts
kubectl config current-context                       # display the current-context
kubectl config use-context my-cluster-name           # set the default context to my-cluster-name

kubectl config set-cluster my-cluster-name           # set a cluster entry in the kubeconfig

# configure the URL to a proxy server to use for requests made by this client in the kubeconfig
kubectl config set-cluster my-cluster-name --proxy-url=my-proxy-url

# add a new user to your kubeconf that supports basic auth
kubectl config set-credentials kubeuser/foo.kubernetes.com --username=kubeuser --password=kubepassword

# permanently save the namespace for all subsequent kubectl commands in that context.
kubectl config set-context --current --namespace=ggckad-s2

# set a context utilizing a specific username and namespace.
kubectl config set-context gce --user=cluster-admin --namespace=foo \
  && kubectl config use-context gce

kubectl config unset users.foo                       # delete user foo

# short alias to set/show context/namespace (only works for bash and bash-compatible shells, current context to be s
et before using kn to set namespace)
alias kx='f() { [ "$1" ] && kubectl config use-context $1 || kubectl config current-context ; } ; f'
alias kn='f() { [ "$1" ] && kubectl config set-context --current --namespace $1 || kubectl config view --minify | grep
namespace | cut -d" " -f6 ; } ; f'
```

CREATING OBJECTS

```
kubectl apply -f ./my-manifest.yaml            # create resource(s)
kubectl apply -f ./my1.yaml -f ./my2.yaml      # create from multiple files
kubectl apply -f ./dir                         # create resource(s) in all manifest files in dir
kubectl apply -f https://git.io/vPieo          # create resource(s) from url
kubectl create deployment nginx --image=nginx  # start a single instance of nginx

# create a Job which prints "Hello World"
kubectl create job hello --image=busybox:1.28 -- echo "Hello World"

# create a CronJob that prints "Hello World" every minute
kubectl create cronjob hello --image=busybox:1.28   --schedule="*/1 * * * *" -- echo "Hello World"

kubectl explain pods                           # get the documentation for pod manifests

# Create multiple YAML objects from stdin
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Pod
metadata:
  name: busybox-sleep
spec:
  containers:
  - name: busybox
    image: busybox:1.28
    args:
    - sleep
    - "1000000"
---
apiVersion: v1
kind: Pod
metadata:
  name: busybox-sleep-less
spec:
  containers:
  - name: busybox
    image: busybox:1.28
    args:
    - sleep
    - "1000"
EOF

# Create a secret with several keys
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  password: $(echo -n "s33msi4" | base64 -w0)
  username: $(echo -n "jane" | base64 -w0)
```

Viewing, finding resources

```
# Get commands with basic output
kubectl get services                          # List all services in the namespace
kubectl get pods --all-namespaces             # List all pods in all namespaces
kubectl get pods -o wide                      # List all pods in the current namespace, with more details
kubectl get deployment my-dep                 # List a particular deployment
kubectl get pods                              # List all pods in the namespace
kubectl get pod my-pod -o yaml                # Get a pod's YAML

# Describe commands with verbose output
kubectl describe nodes my-node
kubectl describe pods my-pod

# List Services Sorted by Name
kubectl get services --sort-by=.metadata.name

# List pods Sorted by Restart Count
kubectl get pods --sort-by='.status.containerStatuses[0].restartCount'

# List PersistentVolumes sorted by capacity
kubectl get pv --sort-by=.spec.capacity.storage

# Get the version label of all pods with label app=cassandra
kubectl get pods --selector=app=cassandra -o \
  jsonpath='{.items[*].metadata.labels.version}'

# Retrieve the value of a key with dots, e.g. 'ca.crt'
kubectl get configmap myconfig \
  -o jsonpath='{.data.ca\.crt}'

# Retrieve a base64 encoded value with dashes instead of underscores.
kubectl get secret my-secret --template='{{index .data "key-name-with-dashes"}}'

# Get all worker nodes (use a selector to exclude results that have a label
# named 'node-role.kubernetes.io/control-plane')
kubectl get node --selector='!node-role.kubernetes.io/control-plane'

# Get all running pods in the namespace
kubectl get pods --field-selector=status.phase=Running

# Get ExternalIPs of all nodes
kubectl get nodes -o jsonpath='{.items[*].status.addresses[?(@.type=="ExternalIP")].address}'

# List Names of Pods that belong to Particular RC
# "jq" command useful for transformations that are too complex for jsonpath, it can be found at https://stedolan.githu
b.io/jq/
sel=${$(kubectl get rc my-rc --output=json | jq -j '.spec.selector | to_entries | .[] | "\(.key)=\(.value),"')%?}
echo $(kubectl get pods --selector=$sel --output=jsonpath={.items..metadata.name})

# Show labels for all pods (or any other Kubernetes object that supports labelling)
kubectl get pods --show-labels
```

```
# Check which nodes are ready
JSONPATH='{range .items[*]}{@.metadata.name}:{range @.status.conditions[*]}{@.type}={@.status};{end}{end}' \
 && kubectl get nodes -o jsonpath="$JSONPATH" | grep "Ready=True"

# Output decoded secrets without external tools
kubectl get secret my-secret -o go-template='{{range $k,$v := .data}}{{"### "}}{{$k}}{{"\n"}}{{$v|base64decode}}{{"\n\n"}}{{end}}'

# List all Secrets currently in use by a pod
kubectl get pods -o json | jq '.items[].spec.containers[].env[]?.valueFrom.secretKeyRef.name' | grep -v null | sort | uniq

# List all containerIDs of initContainer of all pods
# Helpful when cleaning up stopped containers, while avoiding removal of initContainers.
kubectl get pods --all-namespaces -o jsonpath='{range .items[*].status.initContainerStatuses[*]}{.containerID}{"\n"}{end}' | cut -d/ -f3

# List Events sorted by timestamp
kubectl get events --sort-by=.metadata.creationTimestamp

# Compares the current state of the cluster against the state that the cluster would be in if the manifest was applied.
kubectl diff -f ./my-manifest.yaml

# Produce a period-delimited tree of all keys returned for nodes
# Helpful when locating a key within a complex nested JSON structure
kubectl get nodes -o json | jq -c 'paths|join(".")'

# Produce a period-delimited tree of all keys returned for pods, etc
kubectl get pods -o json | jq -c 'paths|join(".")'

# Produce ENV for all pods, assuming you have a default container for the pods, default namespace and the `env` command is supported.
# Helpful when running any supported command across all pods, not just `env`
for pod in $(kubectl get po --output=jsonpath={.items..metadata.name}); do echo $pod && kubectl exec -it $pod -- env; done

# Get a deployment's status subresource
kubectl get deployment nginx-deployment --subresource=status
```

Updating resources

```
kubectl set image deployment/frontend www=image:v2               # Rolling update "www" containers of "frontend" deployment, updating the image
kubectl rollout history deployment/frontend                      # Check the history of deployments including the revision
kubectl rollout undo deployment/frontend                         # Rollback to the previous deployment
kubectl rollout undo deployment/frontend --to-revision=2         # Rollback to a specific revision
kubectl rollout status -w deployment/frontend                    # Watch rolling update status of "frontend" deployment until completion
kubectl rollout restart deployment/frontend                      # Rolling restart of the "frontend" deployment


cat pod.json | kubectl replace -f -                              # Replace a pod based on the JSON passed into stdin
```

```
# Force replace, delete and then re-create the resource. Will cause a service outage.
kubectl replace --force -f ./pod.json

# Create a service for a replicated nginx, which serves on port 80 and connects to the containers on port 8000
kubectl expose rc nginx --port=80 --target-port=8000

# Update a single-container pod's image version (tag) to v4
kubectl get pod mypod -o yaml | sed 's/\(image: myimage\):.*$/\1:v4/' | kubectl replace -f -

kubectl label pods my-pod new-label=awesome                      # Add a Label
kubectl annotate pods my-pod icon-url=http://goo.gl/XXBTWq       # Add an annotation
kubectl autoscale deployment foo --min=2 --max=10                # Auto scale a deployment "foo"
```

PATCHING RESOURCES

```
# Partially update a node
kubectl patch node k8s-node-1 -p '{"spec":{"unschedulable":true}}'

# Update a container's image; spec.containers[*].name is required because it's a merge key
kubectl patch pod valid-pod -p '{"spec":{"containers":[{"name":"kubernetes-serve-hostname","image":"new image"
}]}}'

# Update a container's image using a json patch with positional arrays
kubectl patch pod valid-pod --type='json' -p='[{"op": "replace", "path": "/spec/containers/0/image", "value":"new im
age"}]'

# Disable a deployment livenessProbe using a json patch with positional arrays
kubectl patch deployment valid-deployment  --type json   -p='[{"op": "remove", "path": "/spec/template/spec/contain
ers/0/livenessProbe"}]'

# Add a new element to a positional array
kubectl patch sa default --type='json' -p='[{"op": "add", "path": "/secrets/1", "value": {"name": "whatever" } }]'

# Update a deployment's replica count by patching its scale subresource
kubectl patch deployment nginx-deployment --subresource='scale' --type='merge' -p '{"spec":{"replicas":2}}'
```

EDITING RESOURCES

```
kubectl edit svc/docker-registry                      # Edit the service named docker-registry
KUBE_EDITOR="nano" kubectl edit svc/docker-registry   # Use an alternative editor
```

SCALING RESOURCES

```
kubectl scale --replicas=3 rs/foo                                 # Scale a replicaset named 'foo' to 3
kubectl scale --replicas=3 -f foo.yaml                            # Scale a resource specified in "foo.yaml" to 3
kubectl scale --current-replicas=2 --replicas=3 deployment/mysql  # If the deployment named mysql's current size is
2, scale mysql to 3
kubectl scale --replicas=5 rc/foo rc/bar rc/baz                   # Scale multiple replication controllers
```

DELETING RESOURCES

kubectl delete -f ./pod.json # Delete a pod using the type and name specified in pod.json
kubectl delete pod unwanted --now # Delete a pod with no grace period
kubectl delete pod,service baz foo # Delete pods and services with same names "baz" and "foo"
kubectl delete pods,services -l name=myLabel # Delete pods and services with label name=myLabel
kubectl -n my-ns delete pod,svc --all # Delete all pods and services in namespace my-ns,
Delete all pods matching the awk pattern1 or pattern2
kubectl get pods -n mynamespace --no-headers=true | awk '/pattern1|pattern2/{print $1}' | xargs kubectl delete -n m
ynamespace pod

INTERACTING WITH RUNNING PODS

kubectl logs my-pod # dump pod logs (stdout)
kubectl logs -l name=myLabel # dump pod logs, with label name=myLabel (stdout)
kubectl logs my-pod --previous # dump pod logs (stdout) for a previous instantiation of a container
kubectl logs my-pod -c my-container # dump pod container logs (stdout, multi-container case)
kubectl logs -l name=myLabel -c my-container # dump pod logs, with label name=myLabel (stdout)
kubectl logs my-pod -c my-container --previous # dump pod container logs (stdout, multi-container case) for a pr
evious instantiation of a container
kubectl logs -f my-pod # stream pod logs (stdout)
kubectl logs -f my-pod -c my-container # stream pod container logs (stdout, multi-container case)
kubectl logs -f -l name=myLabel --all-containers # stream all pods logs with label name=myLabel (stdout)
kubectl run -i --tty busybox --image=busybox:1.28 -- sh # Run pod as interactive shell
kubectl run nginx --image=nginx -n mynamespace # Start a single instance of nginx pod in the namespace of myn
amespace
kubectl run nginx --image=nginx # Run pod nginx and write its spec into a file called pod.yaml
--dry-run=client -o yaml > pod.yaml

kubectl attach my-pod -i # Attach to Running Container
kubectl port-forward my-pod 5000:6000 # Listen on port 5000 on the local machine and forward to port 600
0 on my-pod
kubectl exec my-pod -- ls / # Run command in existing pod (1 container case)
kubectl exec --stdin --tty my-pod -- /bin/sh # Interactive shell access to a running pod (1 container case)
kubectl exec my-pod -c my-container -- ls / # Run command in existing pod (multi-container case)
kubectl top pod POD_NAME --containers # Show metrics for a given pod and its containers
kubectl top pod POD_NAME --sort-by=cpu # Show metrics for a given pod and sort it by 'cpu' or 'memory'

COPY FILES AND DIRECTORIES TO AND FROM CONTAINERS

kubectl cp /tmp/foo_dir my-pod:/tmp/bar_dir # Copy /tmp/foo_dir local directory to /tmp/bar_dir in a remote
pod in the current namespace
kubectl cp /tmp/foo my-pod:/tmp/bar -c my-container # Copy /tmp/foo local file to /tmp/bar in a remote pod in a s
pecific container
kubectl cp /tmp/foo my-namespace/my-pod:/tmp/bar # Copy /tmp/foo local file to /tmp/bar in a remote pod in na
mespace my-namespace
kubectl cp my-namespace/my-pod:/tmp/foo /tmp/bar # Copy /tmp/foo from a remote pod to /tmp/bar locally

INTERACTING WITH DEPLOYMENTS AND SERVICES

kubectl logs deploy/my-deployment # dump Pod logs for a Deployment (single-container case)
kubectl logs deploy/my-deployment -c my-container # dump Pod logs for a Deployment (multi-container case)

kubectl port-forward svc/my-service 5000 # listen on local port 5000 and forward to port 5000 on Service backend
kubectl port-forward svc/my-service 5000:my-service-port # listen on local port 5000 and forward to Service target port with name <my-service-port>

kubectl port-forward deploy/my-deployment 5000:6000 # listen on local port 5000 and forward to port 6000 on a Pod created by <my-deployment>
kubectl exec deploy/my-deployment -- ls # run command in first Pod and first container in Deployment (single- or multi-container cases)

INTERACTING WITH NODES AND CLUSTER

kubectl cordon my-node # Mark my-node as unschedulable
kubectl drain my-node # Drain my-node in preparation for maintenance
kubectl uncordon my-node # Mark my-node as schedulable
kubectl top node my-node # Show metrics for a given node
kubectl cluster-info # Display addresses of the master and services
kubectl cluster-info dump # Dump current cluster state to stdout
kubectl cluster-info dump --output-directory=/path/to/cluster-state # Dump current cluster state to /path/to/cluster-state

View existing taints on which exist on current nodes.
kubectl get nodes -o=custom-columns=NodeName:.metadata.name,TaintKey:.spec.taints[*].key,TaintValue:.spec.taints[*].value,TaintEffect:.spec.taints[*].effect

If a taint with that key and effect already exists, its value is replaced as specified.
kubectl taint nodes foo dedicated=special-user:NoSchedule

##
ANSIBLE
##
ANSIBLE PLAYBOOK
YAML EXAMPLE:

```
name: install and configure DB
  hosts: testServer
  become: yes

  vars:
    oracle_db_port_value : 1521

  tasks:
  -name: Install the Oracle DB
    yum: <code to install the DB>

  -name: Ensure the installed service is enabled and running
  service:
```

```
    name: <your service name>
```

Above is a basic syntax of a playbook. Save it in a file as test.yml. A YAML syntax needs to follow the correct indentation.

ANSIBLE SHELL

```
- name: Shell Examples
  hosts: testservers
  tasks:
  - name: Clear Cache and Restart tomcat
    become: yes
    delay: 10
    async: 10
    poll: 50
    shell: |
      echo -e "\n Change directory to the Tomcat"
      cd tomcat8/
      echo -e "\n Present working directory is" `pwd`

      echo -e "\n Stopping the tomcat instance"
      bin/shutdown.sh
      echo -e "\n Clearning the tmp and work directory of tomcat"
      rm -rfv tmp/*
      rm -rfv work/*
      echo -e "\nTruncate the log file"
      > logs/catalina.out
      echo -e "\nDirectory listing"
      ls -lrtd logs/catalina.out
      echo -e "\nStarting the instance"
      bin/startup.sh
    args:
      chdir: "/apps/tomcat/"
    register: fileout
    tags: fileout
  - debug: msg="{{ fileout.stdout_lines }}"
```

ANSIBLE YAML:

YAML is used to describe configuration that has been increasing in the past few years with the help of Ansible and SaltStack.

YAML is more comfortable for humans to read and write in comparison to other standard data formats such as XML or JSON. There are libraries available in most programming languages for working with YAML.

```
# A student record
- Martin:
name: Martin
roll no: 10
class: 12th
div: A
likes:
- Physics
- Chemistry
- Math
- Edward:
 name: Edward
```

roll no: 11
class: 12th
div: A
likes:
- Biology
- English

ANSIBLE INSTALLATION IN LINUX

When you have compared and weighed your options and decided to go for Ansible. Then installed it on your system. Let's go step by step of the installation in different Linux distributions, such as:

Prerequisites
PyYAML: a YAML parser and emitter for the python programming language.
Httplib2: a comprehensive HTTP client library.
parmiko: native python SSHv2 protocol library.
Distro: RHEL/ CentOS/ Debian/ Ubuntu Linux.
Jinja2: a modern and designer friendly templating language for python.
sshpass: a non-interactive ssh password authentication.
Install Ansible on RedHat/Centos systems
Step 1: Install the EPEL repo

[root@ansible-server ~]# sudo yum install epel-release
Step 2: Install the Ansible package.

[root@ansible-server ~]# sudo yum install -y ansible
Install Ansible on Debian/Ubuntu systems
Step 1: First perform an update to the packages

$ sudo apt update
Step 2: Then install the software properties common package.

$ sudo apt install software-properties-common
Step 3: And install the Ansible personal package archive.

$ sudo apt-add-repository ppa:ansible/ansible
Step 4: Install the Ansible.

$ sudo apt update
$ sudo apt install ansible

##
##

TERRAFORM

##
##

HashiCorp Terraform is an open source infrastructure as code (IaC) software tool
that allows DevOps engineers to programmatically provision the physical resources an application requires to run.
Infrastructure as code is an IT practice that manages an application's underlying IT infrastructure through programmi
ng.

EXAMPLE OF TERRAFORM SCRIPTING

```
 terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.0"
    }
  }
}

# Configure the AWS Provider
provider "aws" {
  region = "us-east-1"
}

# Create a VPC
resource "aws_vpc" "example" {
  cidr_block = "10.0.0.0/16"
}
```

EXAMPLE FOR S3 BUCKET IN TERRAFORM:

```
       provider "aws" {
  region = "eu-west-1"
}

provider "aws" {
  alias  = "central"
  region = "eu-central-1"
}

resource "aws_iam_role" "replication" {
  name = "tf-iam-role-replication-12345"

  assume_role_policy = <<POLICY
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Principal": {
        "Service": "s3.amazonaws.com"
      },
      "Effect": "Allow",
      "Sid": ""
    }
  ]
}
POLICY
}

resource "aws_iam_policy" "replication" {
  name = "tf-iam-role-policy-replication-12345"
```

```
  policy = <<POLICY
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:GetReplicationConfiguration",
        "s3:ListBucket"
      ],
      "Effect": "Allow",
      "Resource": [
        "${aws_s3_bucket.source.arn}"
      ]
    },
    {
      "Action": [
        "s3:GetObjectVersionForReplication",
        "s3:GetObjectVersionAcl",
         "s3:GetObjectVersionTagging"
      ],
      "Effect": "Allow",
      "Resource": [
        "${aws_s3_bucket.source.arn}/*"
      ]
    },
    {
      "Action": [
        "s3:ReplicateObject",
        "s3:ReplicateDelete",
        "s3:ReplicateTags"
      ],
      "Effect": "Allow",
      "Resource": "${aws_s3_bucket.destination.arn}/*"
    }
  ]
}
POLICY
}

resource "aws_iam_role_policy_attachment" "replication" {
  role       = aws_iam_role.replication.name
  policy_arn = aws_iam_policy.replication.arn
}

resource "aws_s3_bucket" "destination" {
  bucket = "tf-test-bucket-destination-12345"

  versioning {
    enabled = true
  }
}

resource "aws_s3_bucket" "source" {
  provider = aws.central
  bucket   = "tf-test-bucket-source-12345"
```

```
  acl      = "private"

  versioning {
    enabled = true
  }

  replication_configuration {
    role = aws_iam_role.replication.arn

    rules {
      id     = "foobar"
      status = "Enabled"

      filter {
        tags = {}
      }
      destination {
        bucket        = aws_s3_bucket.destination.arn
        storage_class = "STANDARD"

        replication_time {
          status  = "Enabled"
          minutes = 15
        }

        metrics {
          status  = "Enabled"
          minutes = 15
        }
      }
    }
  }
}

################################################################################################
#########################################################################
```