DON BOSCO INSTITUTE OF TECHNOLOGY

Department of Information and Technology

Course: ITL503

INDEX

EXPERIMENT A : SELF STUDY

1. **What is devops?**

   DevOps is a software development approach that combines development (Dev) and operations (Ops) teams to improve collaboration, communication, and efficiency throughout the software development lifecycle. It aims to automate processes, eliminate silos, and foster a culture of continuous integration, delivery, and deployment. DevOps practices involve using tools, such as version control, continuous integration, and containerization, to streamline development, testing, and deployment processes. By adopting DevOps, organizations can achieve faster time to market, improved software quality, and increased customer satisfaction.

0. List down the stages of development & operations.

The stages of development and operations in the software development lifecycle can vary depending on the specific methodology or framework being followed. However, here are the commonly recognized stages:

. Planning: This phase involves defining project objectives, requirements gathering, and creating a roadmap for development and operations.

. Development: In this stage, developers write and test code, design software architecture, and build the application or system according to the requirements.

. Continuous Integration: This phase involves integrating code changes frequently and automatically to ensure that the software remains stable and functional.

. Testing: Quality assurance engineers perform various types of testing, including unit testing, integration testing, system testing, and acceptance testing, to identify and fix bugs and ensure software reliability.

. Deployment: This stage involves deploying the software to production servers or cloud environments, making it available for users.

. Operations: Once the software is deployed, operations teams monitor and manage the system, ensuring its availability, performance, and scalability. They also handle user support and perform maintenance tasks.

. Continuous Delivery/Deployment: Organizations that adopt continuous delivery or continuous deployment practices automate the process of releasing new features or updates to the software, allowing for faster and more frequent releases.

. Monitoring and Feedback: This stage involves monitoring the software's performance, collecting user feedback, and making improvements based on the feedback and monitoring data.

0. For every stage of devops. Mention at least two tools and a line detail about each tool.

Here are two commonly used tools for each stage of the DevOps process:

1. Planning:

Jira: Jira is a widely used project management tool that helps teams plan, track, and manage software development projects. It provides features such as issue tracking, task management, and agile project management.

Trello: Trello is a visual collaboration tool that allows teams to organize and prioritize tasks using boards, lists, and cards. It provides a simple and intuitive interface for managing projects and tasks.

2. Development:

- Git: Git is a distributed version control system that allows developers to track changes in their codebase, collaborate with others, and manage different versions of their software. It provides features like branching, merging, and conflict resolution.

- Bitbucket: Bitbucket is a web-based version control repository hosting service that supports both Git and Mercurial. It provides features like code collaboration, pull requests, and continuous integration.

3. Continuous Integration:

- Jenkins: Jenkins is an open-source automation server that facilitates continuous integration and delivery of software projects. It allows developers to automate the build, test, and deployment process, ensuring that code changes are integrated smoothly.

- CircleCI: CircleCI is a cloud-based continuous integration and delivery platform that automates software builds, tests, and deployment. It provides a scalable and customizable environment for running CI/CD pipelines.

4. Testing:

- Selenium: Selenium is a popular open-source framework for automating web browsers. It allows developers to write tests in various programming languages to verify the functionality of web applications across different browsers and platforms.

- JUnit: JUnit is a unit testing framework for Java. It provides a simple and flexible way to write and run tests for Java applications, ensuring that each unit of code behaves as expected.

5. Deployment:

- Docker: Docker is an open-source platform that allows developers to automate the deployment of applications inside lightweight, portable containers. It provides an efficient and consistent environment for running applications across different systems.

- Kubernetes: Kubernetes is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications. It provides features like load balancing, service discovery, and self-healing capabilities.

6. Operations:

- Nagios: Nagios is a powerful monitoring and alerting tool that helps IT teams monitor the health and performance of their infrastructure, servers, and network devices. It provides real-time monitoring, notifications, and reporting capabilities.

- Zabbix: Zabbix is an open-source monitoring software that allows organizations to monitor and track the performance and availability of their IT infrastructure. It offers features such as network monitoring, server monitoring, and application monitoring.

7. Continuous Delivery/Deployment:

- AWS CodePipeline: AWS CodePipeline is a fully managed continuous delivery service that helps teams automate their software release process. It allows developers to build, test, and deploy applications using various AWS services.

- GitLab CI/CD: GitLab CI/CD is a built-in continuous integration and continuous deployment tool provided by GitLab. It enables developers to automate the entire software development lifecycle, from code commit to production deployment.

8. Monitoring and Feedback:

- New Relic: New Relic is a monitoring and observability platform that provides real-time insights into the performance and health of applications, infrastructure, and customer experiences. It helps teams identify and resolve issues before they impact end-users.

- Grafana: Grafana is an open-source data visualization and monitoring tool that allows teams to create interactive dashboards and graphs to analyze and monitor metrics from various data sources. It provides a flexible and customizable interface for visualizing data.

4. What do you mean by version control or versioning?

Version control, also known as versioning or source control, is a system or process used in software development to manage changes to source code, documents, or any other type of files over time. It provides a way to track and organize modifications, additions, and deletions made to files, allowing multiple people to collaborate on a project while keeping track of the history of changes.

The primary goals of version control are:

1. Collaboration: Version control enables multiple developers to work on the same codebase simultaneously without interfering with each other's work. Changes can be integrated and merged together.

2. History and Tracking: Version control systems maintain a detailed history of all changes made to files, including who made the changes, when they were made, and what specific modifications were performed. This historical information is invaluable for understanding the evolution of a project and for troubleshooting issues.

3. Reproducibility: Version control allows you to recreate the state of a project at any point in its history. This is crucial for reproducing specific versions of software or documents, which can be important for debugging or maintaining compatibility.

4. Backup and Recovery: Version control serves as a form of backup by storing previous versions of files. This can be useful for recovering lost or mistakenly deleted work.

5. Branching and Merging: Version control systems provide the ability to create separate branches of the codebase, which allows developers to work on different features or fixes independently. Branches can later be merged back together, combining the changes made in each branch.

Version control systems can be categorized into two main types: centralized and distributed.

• Centralized Version Control: In a centralized system, there is a single, central repository that stores all versions of files. Developers check out files from this repository, make changes, and then check them back in. Examples of centralized version control systems include Subversion (SVN) and Perforce.

• Distributed Version Control: In a distributed system, each developer has their own local copy of the entire repository, including the complete history. Developers can work independently, commit changes to their local copy, and then share those changes with others by pushing them to a shared remote repository. Examples of distributed version control systems include Git and Mercurial.

5. What is a microservice & lightweight , monolithic server.

Microservices and monolithic architectures are two different approaches to designing and structuring software applications. The terms "lightweight" and "monolithic server" can also be used in the context of these architectural styles. Let's break down each concept:

1. Microservices: Microservices architecture is an approach to building applications as a collection of small, loosely coupled services that can be developed, deployed, and scaled independently. Each microservice is responsible for a specific, well-defined functionality of the application. These services communicate with each other through well-defined APIs (often over HTTP/REST or message queues).

Advantages of microservices include:

• Scalability: Each microservice can be scaled independently based on demand.

• Flexibility: Different services can be developed using different technologies or programming languages.

• Ease of Maintenance: Changes or updates to one microservice have minimal impact on others.

• Resilience: Failures in one microservice do not necessarily affect the entire application.

• Team Autonomy: Different teams can work on different microservices concurrently.

Challenges of microservices include:

• Complexity: Managing a distributed system can be complex, especially in terms of communication and data consistency.

• Deployment and Monitoring: Managing multiple services and their deployments can be challenging.

• Testing: End-to-end testing can be more complicated due to the distributed nature of the architecture.

2. Monolithic Architecture: In a monolithic architecture, an application is built as a single, self-contained unit. All the components and functionality of the application are tightly integrated into a single codebase. This approach was common in traditional software development.

Advantages of monolithic architecture include:

• Simplicity: Easier development and testing due to the centralized nature of the application.

• Deployment: Deploying a monolithic application is relatively straightforward.

Challenges of monolithic architecture include:

• Scalability: Scaling the application as a whole can be more challenging, as all components are interconnected.

• Flexibility: Making changes to one part of the application can impact the entire system.

• Maintenance: As the application grows, it can become harder to manage and maintain.

3. Lightweight Monolithic Server: The term "lightweight monolithic server" is not a standard architectural concept but can refer to a monolithic application that is designed to be efficient and lightweight in terms of its resource usage and performance. This might involve optimizing code, minimizing dependencies, and focusing on efficient data processing

EXPERIMENT 1 & 2

Git installation and versioning

Aim: To install git (local repository) and synchronize with github (remote repository) and perform version controlling.

Steps for installation and version control:
git config
Usage: git config –global user.name "[name]"
Usage: git config –global user.email "[email address]"
This command sets the author name and email address respectively to be used with your commits.

git init
Usage: git init [repository name]

This command is used to start a new repository.

git clone
Usage: git clone [url]
This command is used to obtain a repository from an existing URL.

git add
Usage: git add [file]
This command adds a file to the staging area.

Usage: git add *
This command adds one or more to the staging area.

git commit
Usage: git commit -m "[Type in the commit message]"
This command records or snapshots the file permanently in the version history.

Usage: git commit -a
This command commits any files you've added with the git add command and also commits any files you've changed since then.

git diff
Usage: git diff
This command shows the file differences which are not yet staged.

Usage: git diff –staged
This command shows the differences between the files in the staging area and the latest version is present.

Usage: git diff [first branch] [second branch]
This command shows the differences between the two branches mentioned.

git reset
Usage: git reset [file]

This command unstages the file, but it preserves the file contents.

Usage: git reset [commit]
This command undoes all the commits after the specified commit and preserves the changes locally.

Usage: git reset –hard [commit] This command discards all history and goes back to the specified commit.

git status
Usage: git status
This command lists all the files that have to be committed.

git rm
Usage: git rm [file]
This command deletes the file from your working directory and stages the deletion.

git log
Usage: git log
This command is used to list the version history for the current branch.

Usage: git log –follow[file]
This command lists version history for a file, including the renaming of files also.

git show
Usage: git show [commit]
This command shows the metadata and content changes of the specified commit.

git tag
Usage: git tag [commitID]
This command is used to give tags to the specific commit.

git branch
Usage: git branch
This command lists all the local branches in the current repository.

Usage: git branch [branch name]

This command creates a new branch.

Usage: git branch -d [branch name]
This command deletes the feature branch.

git checkout
Usage: git checkout [branch name]
This command is used to switch from one branch to another.

Usage: git checkout -b [branch name]
This command creates a new branch and also switches to it.

git merge
Usage: git merge [branch name]
This command merges the specified branch's history into the current branch.

git remote
Usage: git remote add [variable name] [Remote Server Link]
This command is used to connect your local repository to the remote server.

git push
Usage: git push [variable name] master
This command sends the committed changes of master branch to your remote repository.

Usage: git push [variable name] [branch]
This command sends the branch commits to your remote repository.

Usage: git push –all [variable name]
This command pushes all branches to your remote repository.

Usage: git push [variable name] :[branch name]
This command deletes a branch on your remote repository.

git pull
Usage: git pull [Repository Link]
This command fetches and merges changes on the remote server to your working directory.

git stash
Usage: git stash save

This command temporarily stores all the modified tracked files.

Usage: git stash pop
This command restores the most recently stashed files.

Usage: git stash list
This command lists all stashed changesets.

Usage: git stash drop
This command discards the most recently stash009564ed changeset.

Get Token
1. Log into GitHub.
2. Click on your name / Avatar in the upper right corner and select Settings.
3. On the left, click Developer settings.
4. Select Personal access tokens and click Generate new token.
5. Give the token a description/name and select the scope of the token. ...
6. Click Generate token.
7. This configures the computer to remember the complex token by enable caching of the credentials.
git config --global credential.helper cache
8. If needed, you can later clear the token from the local computer by running
git config --global --unset credential.helper
Once GIT is configured,
git config --global user.name ""
git config --global user.email ""
git config -l
we can begin using it to access GitHub. In this example I perform a git clone command to copy a repository to the local computer. When prompted for the username and password, enter your GitHub username and the previously generated token as the password.
Git clone url

Conclusion: Successfully performed and installed git commands and version control.

References: **https://www.digitalocean.com/community/tutorials/how-to-install-git-on-ubuntu-22-04**

EXPERIMENT 3

Installation of Jenkins and to perform CI/CD

Aim: Installation of Jenkins and to perform Continuous Integration/Continuous Development

Steps for installation:

Step 1 — Installing Jenkins

The version of Jenkins included with the default Ubuntu packages is often behind the latest available version from the project itself. To ensure you have the latest fixes and features, use the project-maintained packages to install Jenkins.

First, add the repository key to your system:

wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key |sudo gpg --dearmor -o /usr/share/keyrings/jenkins.gpg

The gpg --dearmor command is used to convert the key into a format that apt recognizes.

Next, let's append the Debian package repository address to the server's sources.list:

sudo sh -c 'echo deb [signed-by=/usr/share/keyrings/jenkins.gpg] http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'

The [signed-by=/usr/share/keyrings/jenkins.gpg] portion of the line ensures that apt will verify files in the repository using the GPG key that you just downloaded.

After both commands have been entered, run apt update so that apt will use the new repository.

sudo apt update

Finally, install Jenkins and its dependencies:

sudo apt install jenkins

Now that Jenkins and its dependencies are in place, we'll start the Jenkins server.

Step 2 — Starting Jenkins

now that Jenkins is installed, start it by using systemctl:

sudo systemctl start jenkins.service

Since systemctl doesn't display status output, we'll use the status command to verify that Jenkins started successfully:

sudo systemctl status jenkins

If everything went well, the beginning of the status output shows that the service is active and configured to start at boot:

Output
● jenkins.service - Jenkins Continuous Integration Server
Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor preset: enabled)
Active: active (running) since Mon 2022-04-18 16:07:28 UTC; 2min 3s ago
Main PID: 88180 (java)
Tasks: 42 (limit: 4665)
Memory: 1.1G
CPU: 46.997s
CGroup: /system.slice/jenkins.service
└─88180 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080

Now that Jenkins is up and running, adjust your firewall rules so that you can reach it from a web browser to complete the initial setup.

Step 3 — Opening the Firewall

To set up a UFW firewall, visit Initial Server Setup with Ubuntu 22.04, Step 4- Setting up a Basic Firewall. By default, Jenkins runs on port 8080. Open that port using ufw:

sudo ufw allow 8080

Note: If the firewall is inactive, the following commands will allow OpenSSH and enable the firewall:

sudo ufw allow OpenSSH

sudo ufw enable

Check ufw's status to confirm the new rules:

sudo ufw status

You'll notice that traffic is allowed to port 8080 from anywhere:

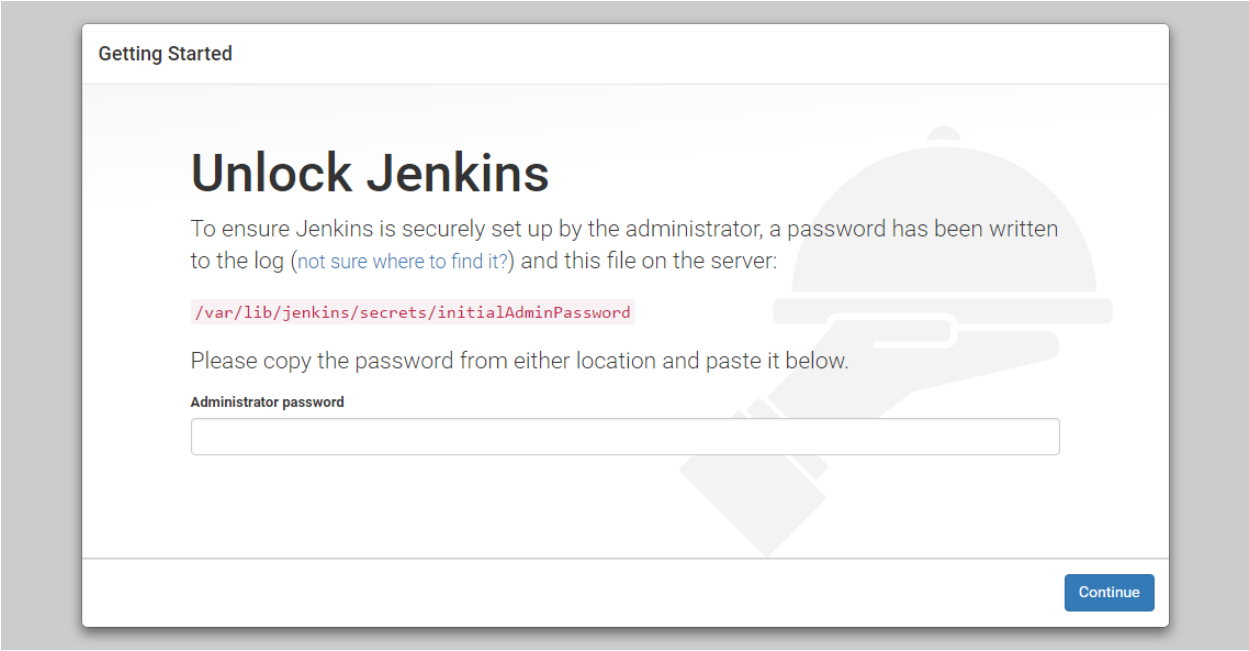
```
Output
Status: active

To                         Action      From
--                         ------      ----
OpenSSH                    ALLOW       Anywhere
8080                       ALLOW       Anywhere
OpenSSH (v6)               ALLOW       Anywhere (v6)
8080 (v6)                  ALLOW       Anywhere (v6)
```

With Jenkins installed and a firewall configured, you have completed the installation stage and can continue with configuring Jenkins.

Step 4 — Setting Up Jenkins

To set up your installation, visit Jenkins on its default port, 8080, using your server domain name or IP address: http://your_server_ip_or_domain:8080

You should receive the **Unlock Jenkins** screen, which displays the location of the initial password:

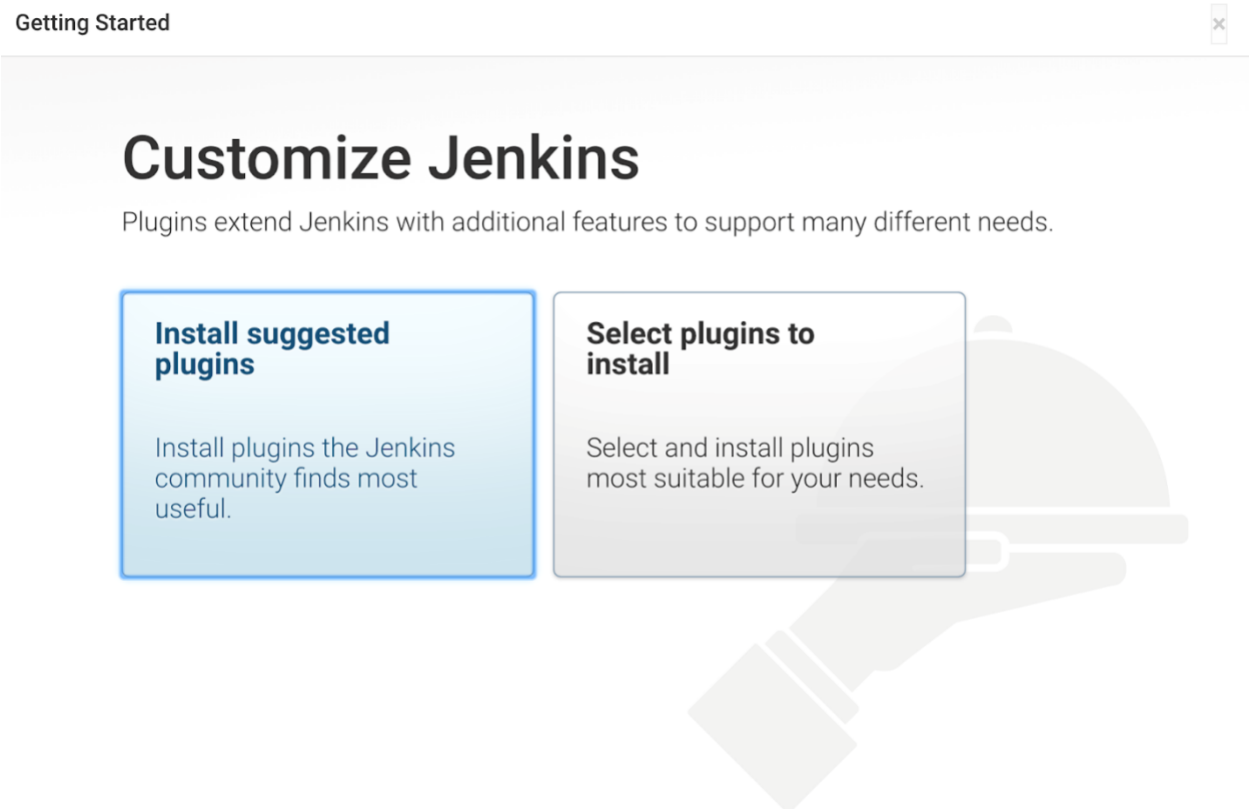


In the terminal window, use the cat command to display the password:

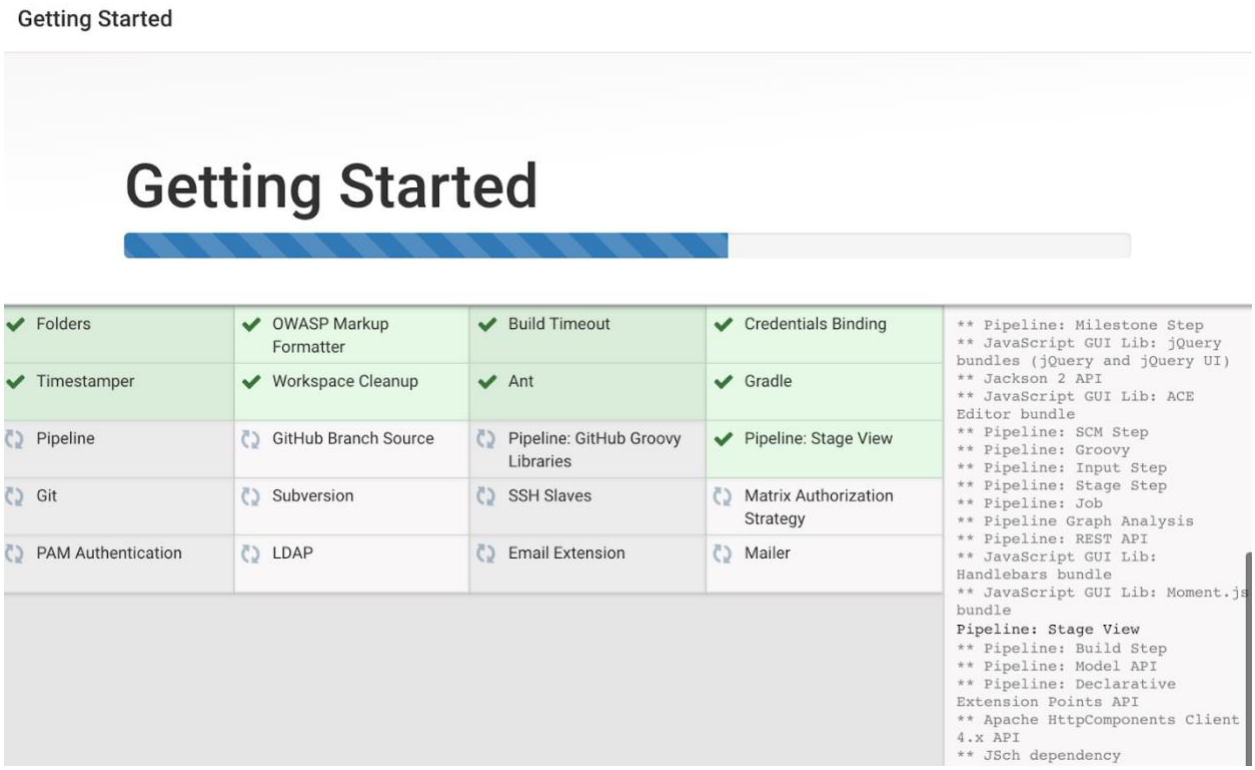
```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

Copy the 32-character alphanumeric password from the terminal and paste it into the **Administrator password** field, then click **Continue**.

The next screen presents the option of installing suggested plugins or selecting specific plugins:

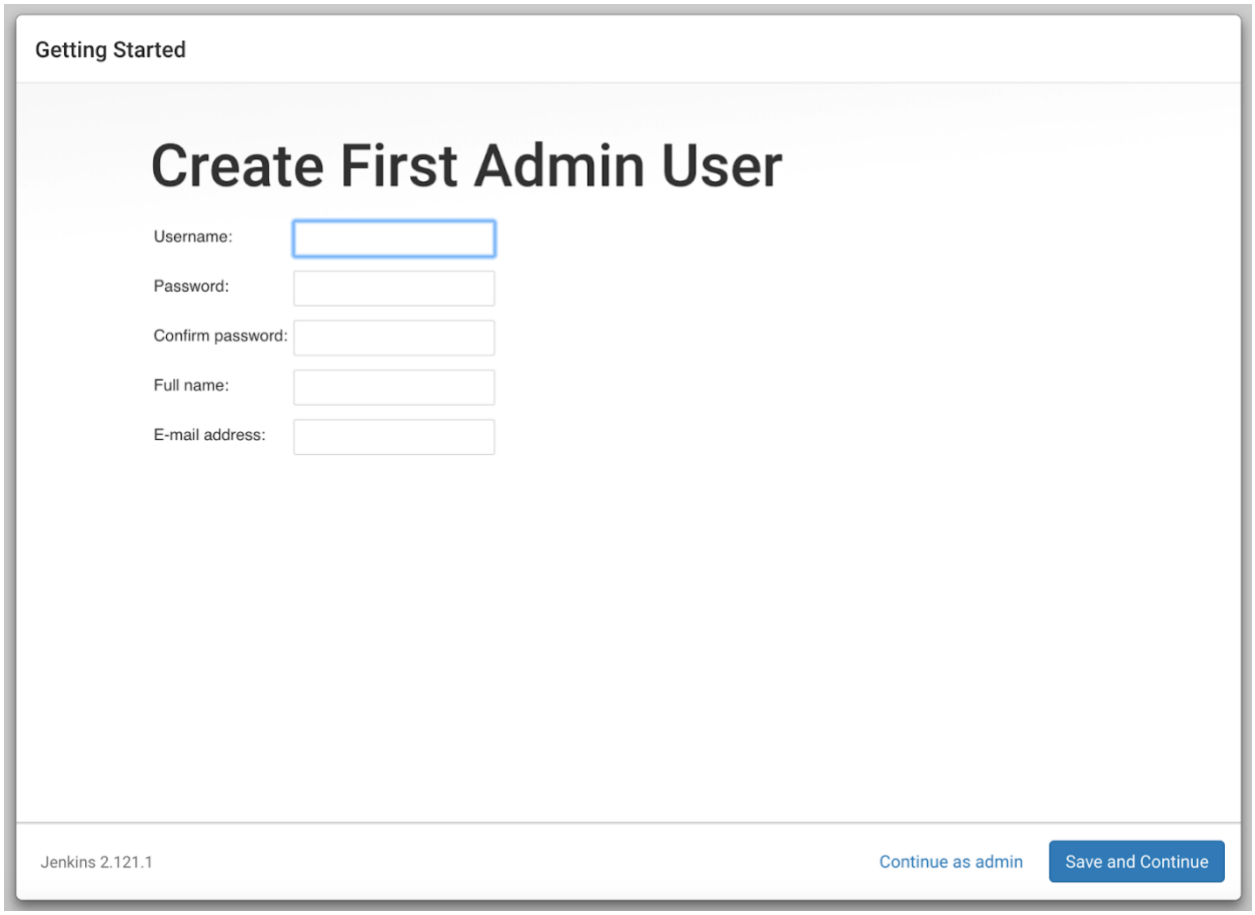


We'll click the **Install suggested plugins** option, which will immediately begin the installation process.

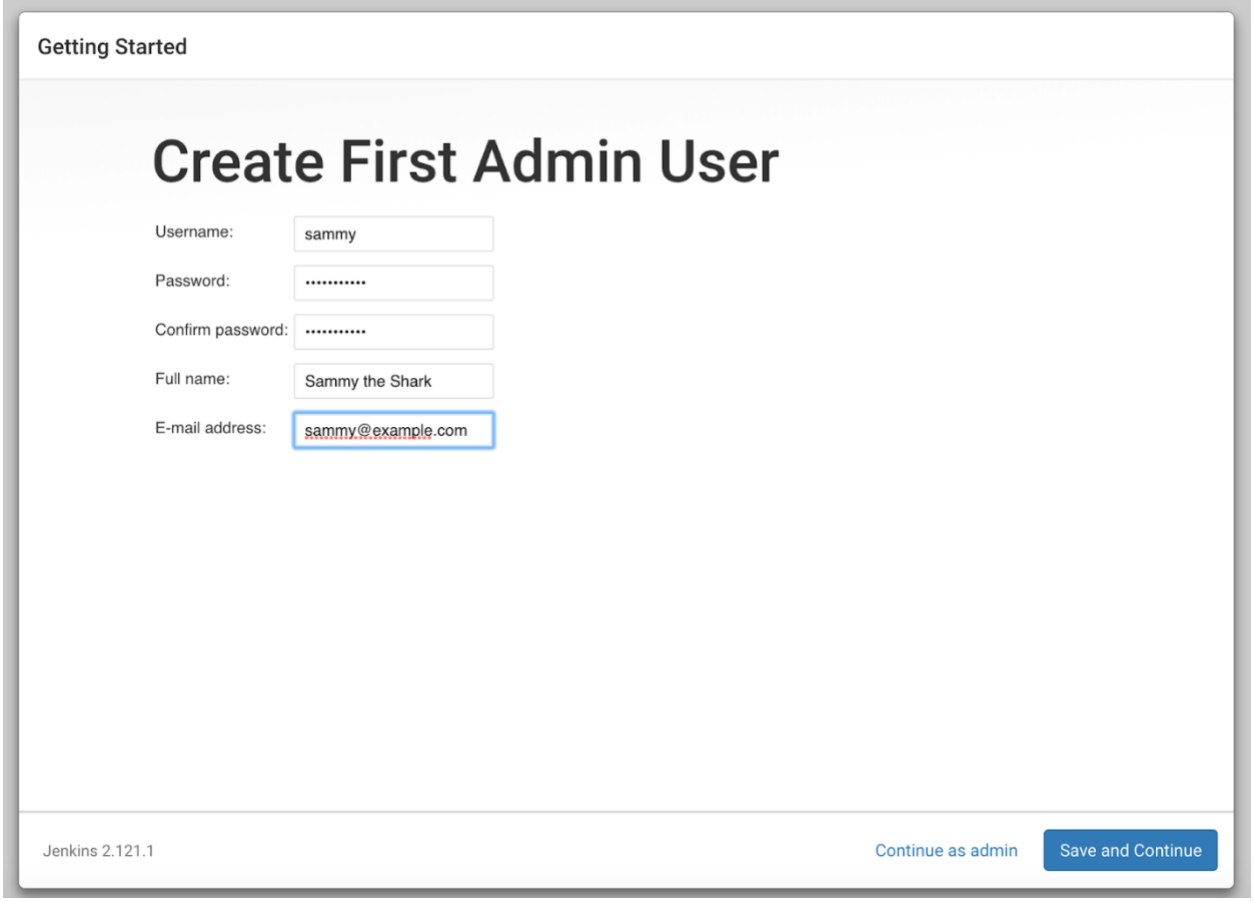


When the installation is complete, you'll be prompted to set up the first administrative user. It's possible to skip this step and continue as admin using the initial password from above, but we'll take a moment to create the user.

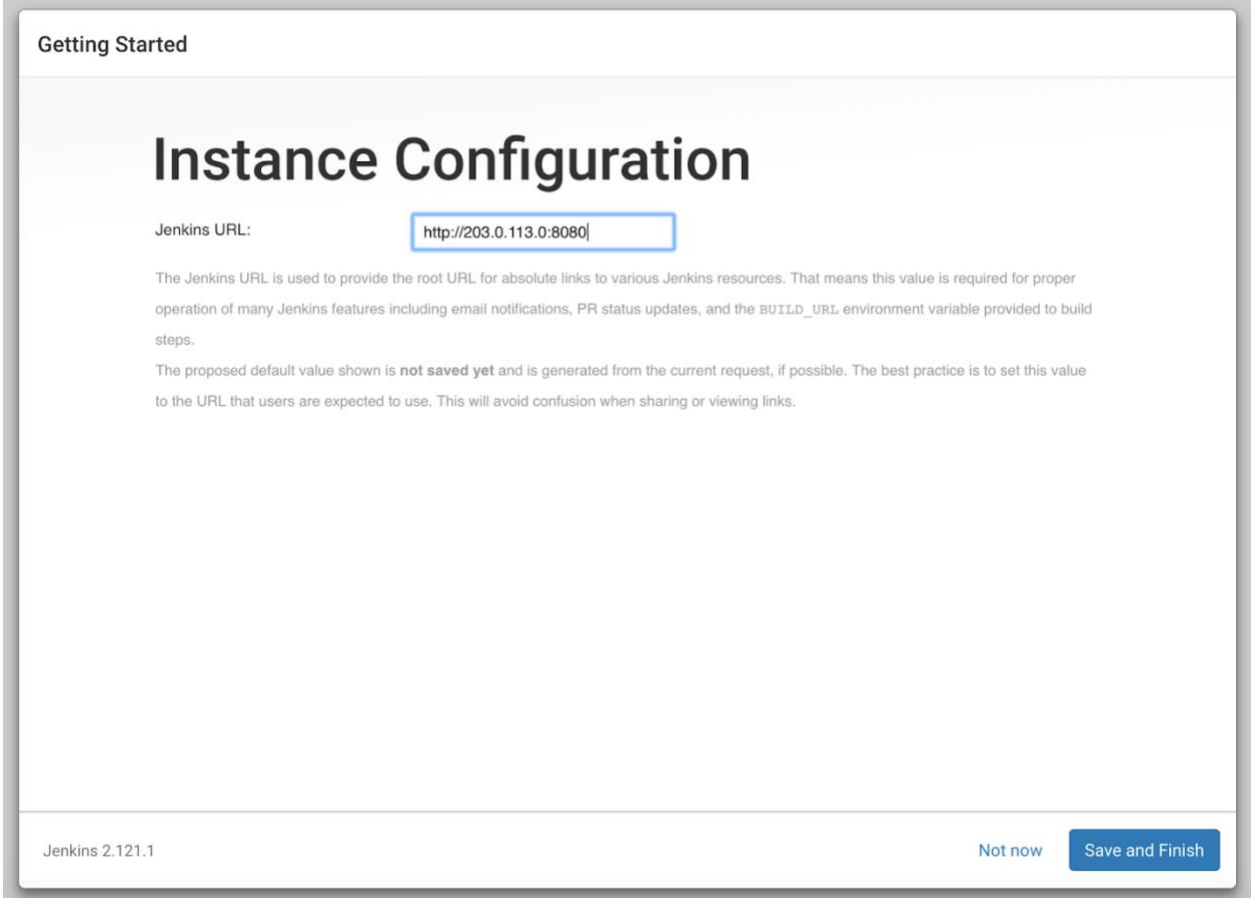
Note: The default Jenkins server is NOT encrypted, so the data submitted with this form is not protected. Refer to How to Configure Jenkins with SSL Using an Nginx Reverse Proxy on Ubuntu 22.04 to protect user credentials and information about builds that are transmitted via the web interface.



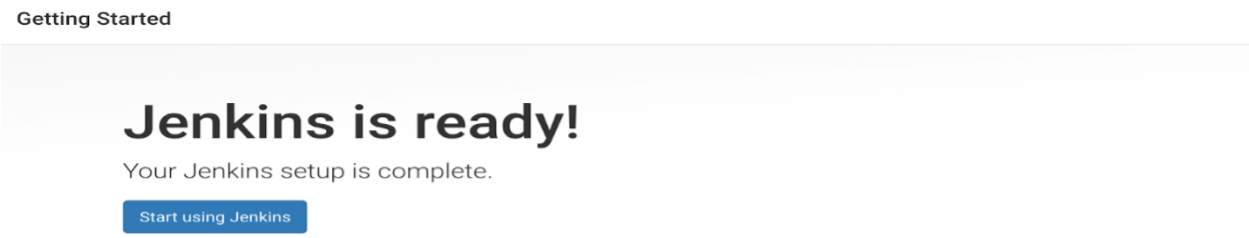
Enter the name and password for your user:



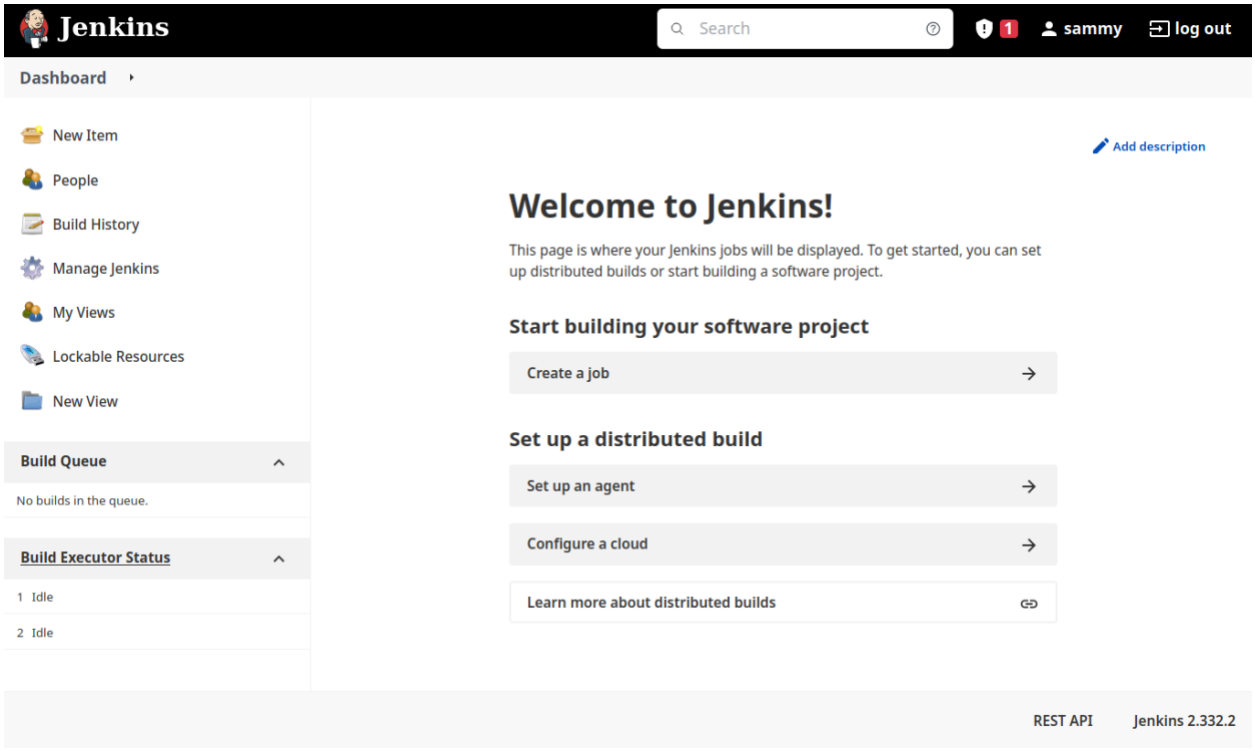
You'll receive an **Instance Configuration** page that will ask you to confirm the preferred URL for your Jenkins instance. Confirm either the domain name for your server or your server's IP address:



After confirming the appropriate information, click **Save and Finish**. You'll receive a confirmation page confirming that **"Jenkins is Ready!"**:



Click **Start using Jenkins** to visit the main Jenkins dashboard:



At this point, you have completed a successful installation of Jenkins.

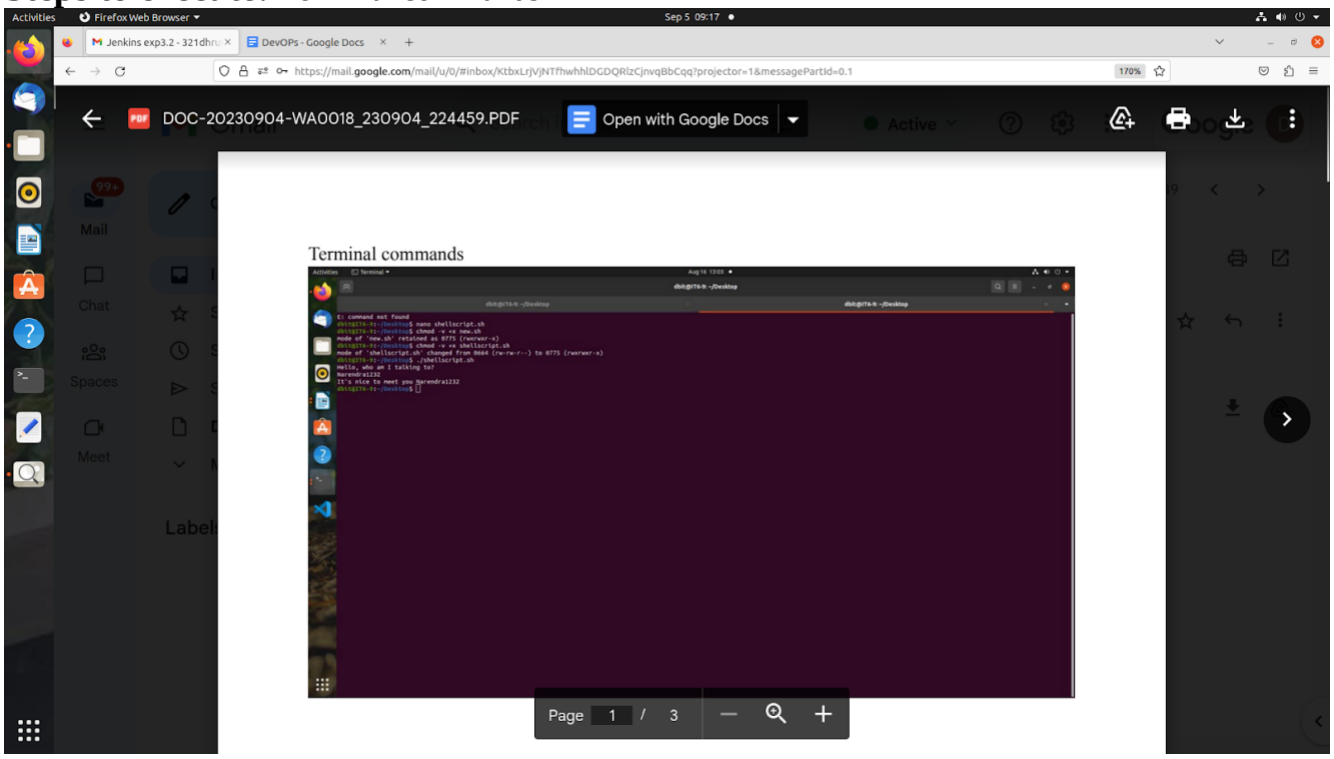
Conclusion: Successfully installed jenkins.

References: https://www.digitalocean.com/community/tutorials/how-to-install-jenkins-on-ubuntu-22-04

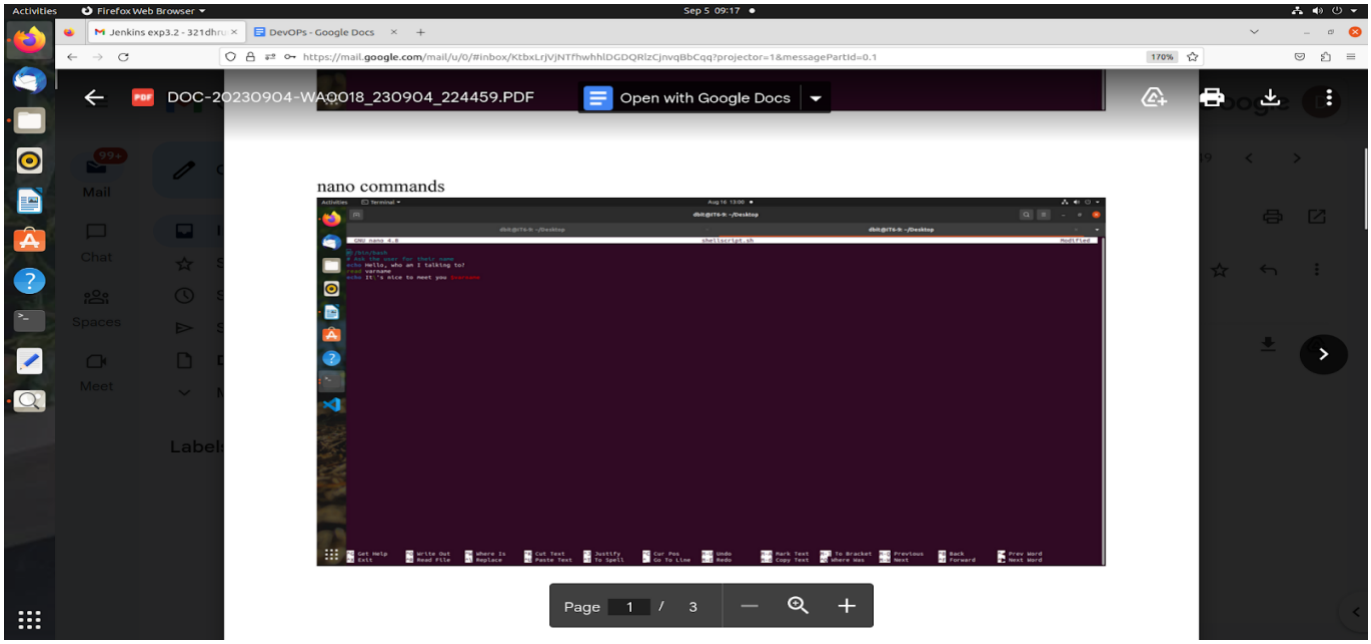
Experiment 3:
Jenkin 2

Aim: To create shell script, pipeline and parameterized program in Jenkin

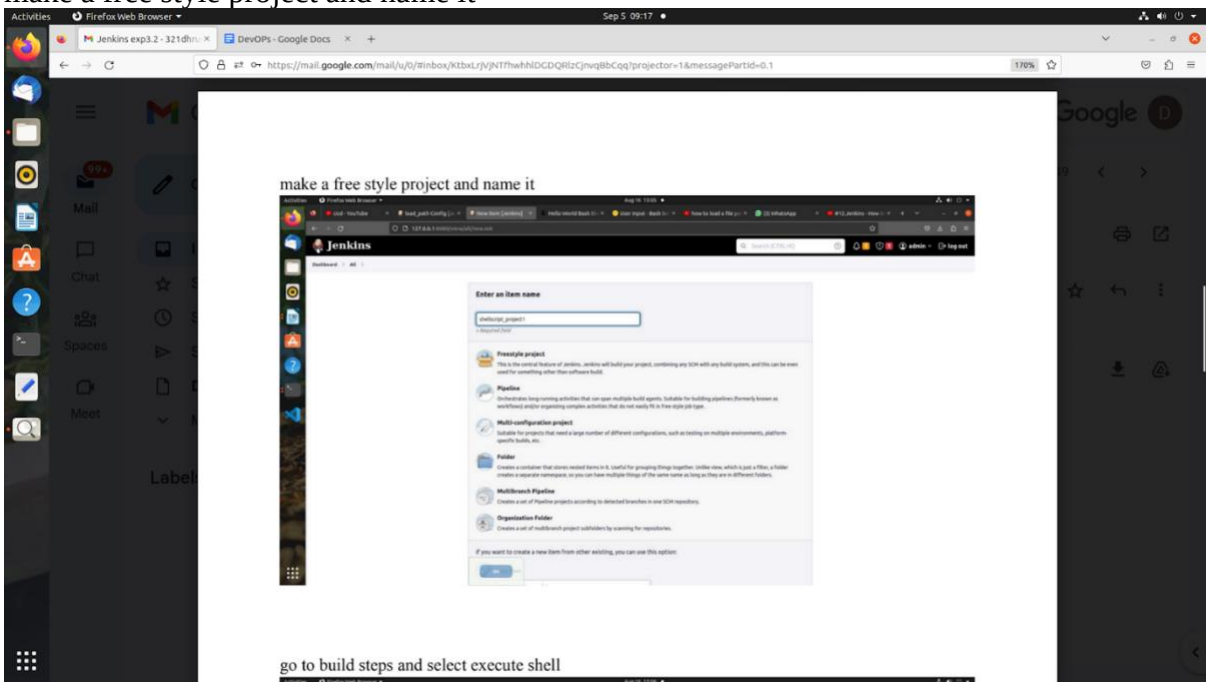
Steps to execute: Terminal commands



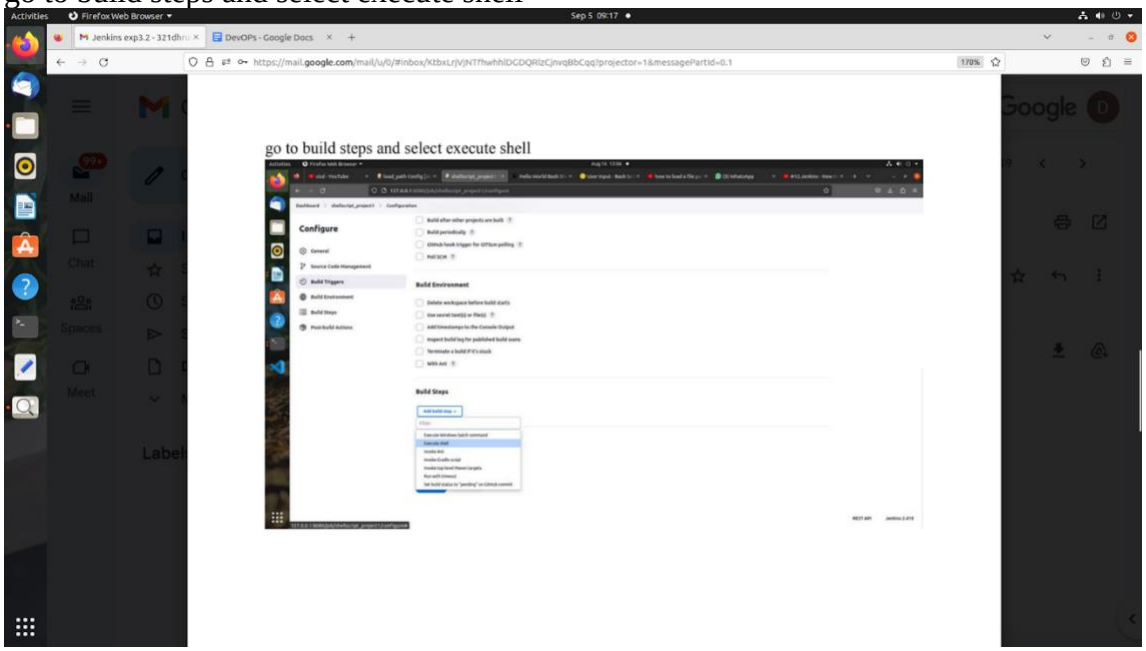
nano commands



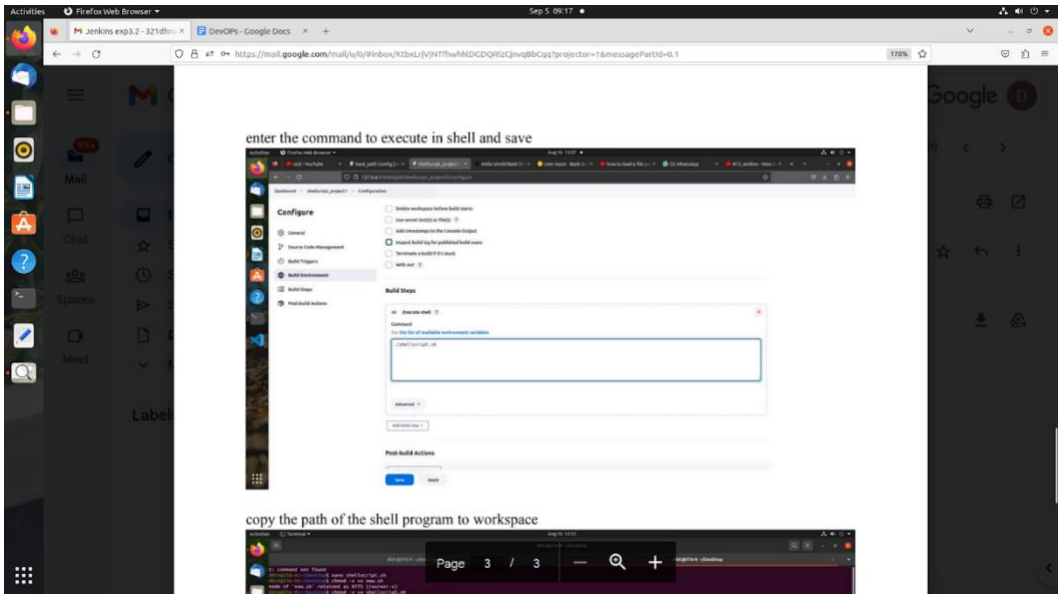
make a free style project and name it



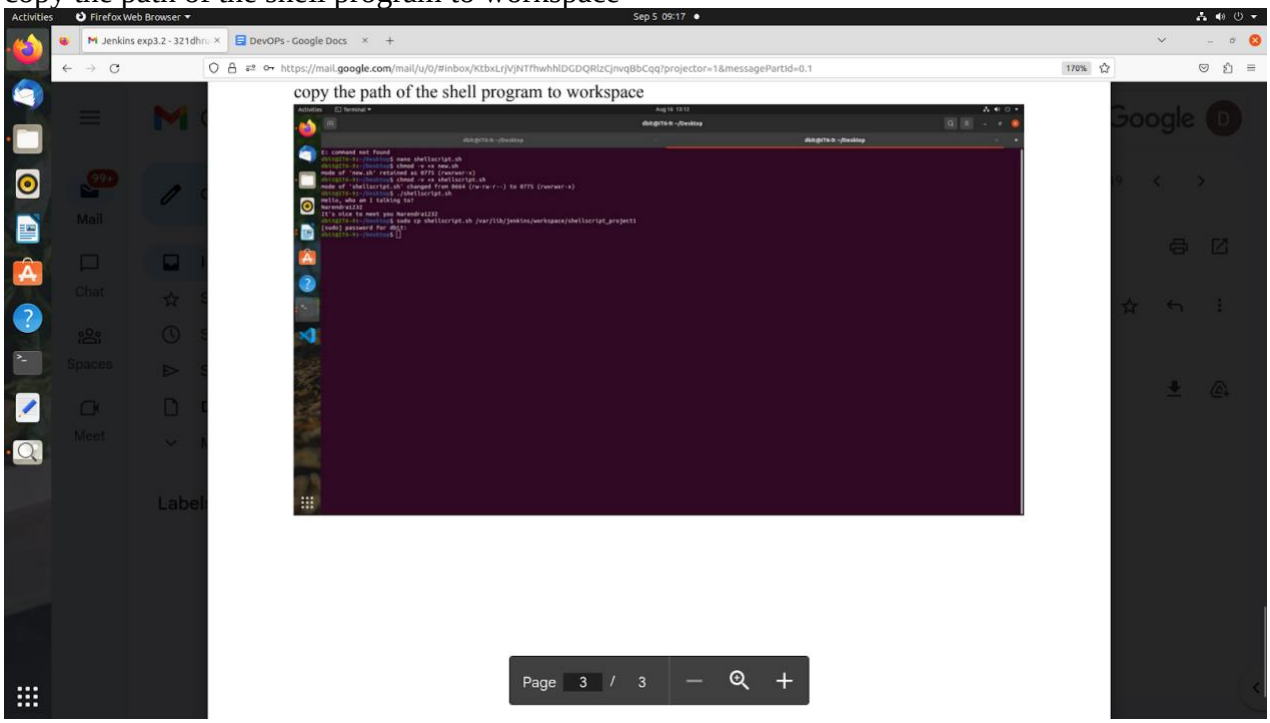
go to build steps and select execute shell



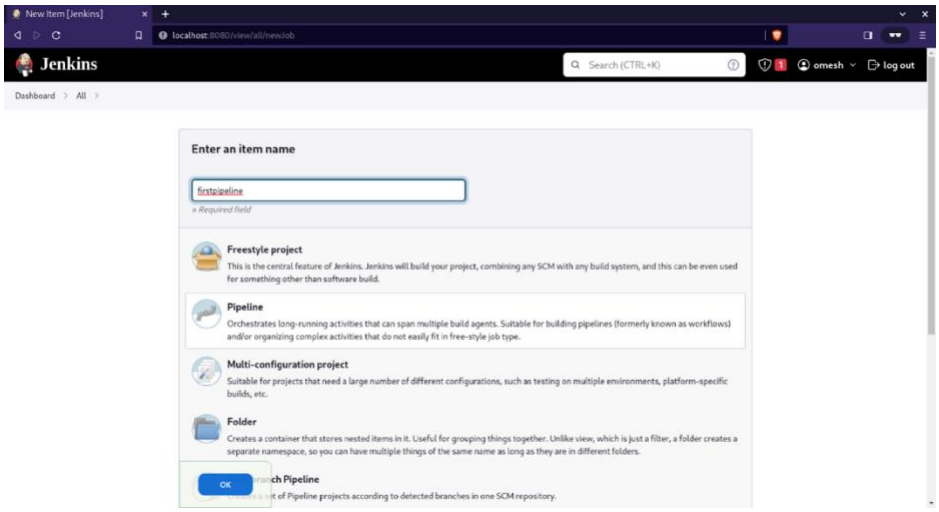
enter the command to execute in shell and save



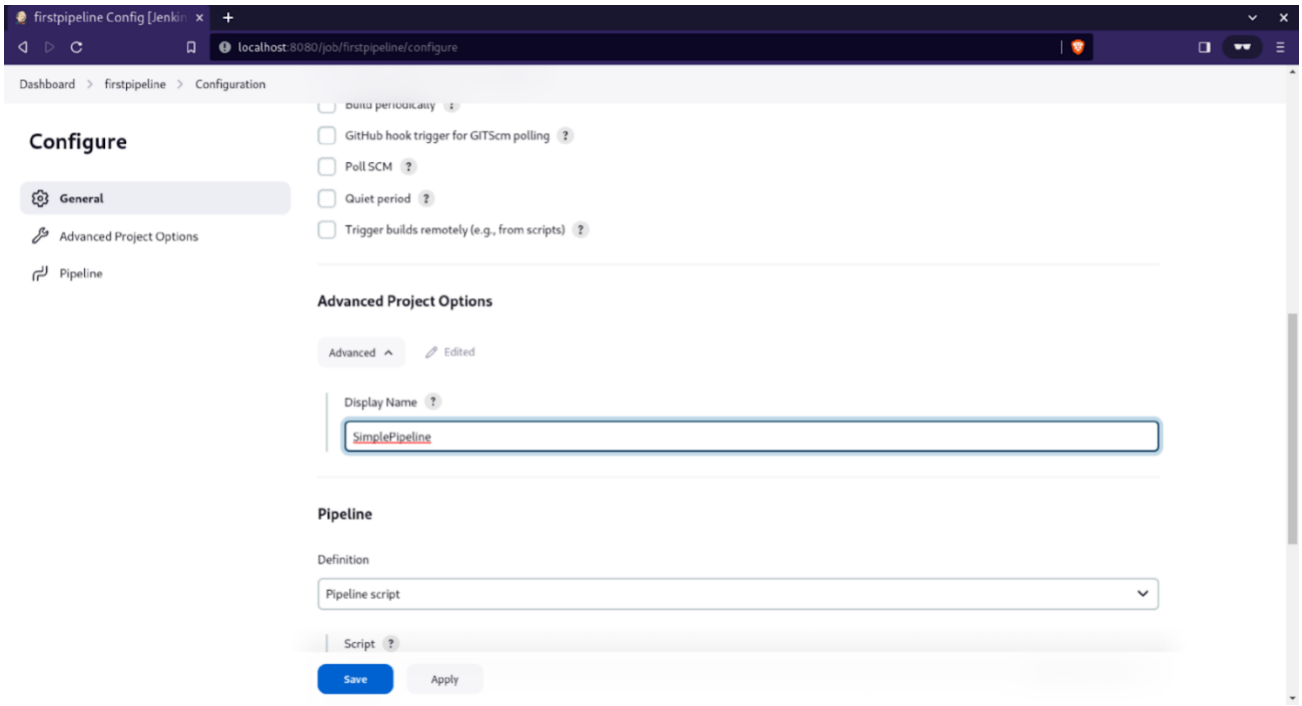
copy the path of the shell program to workspace



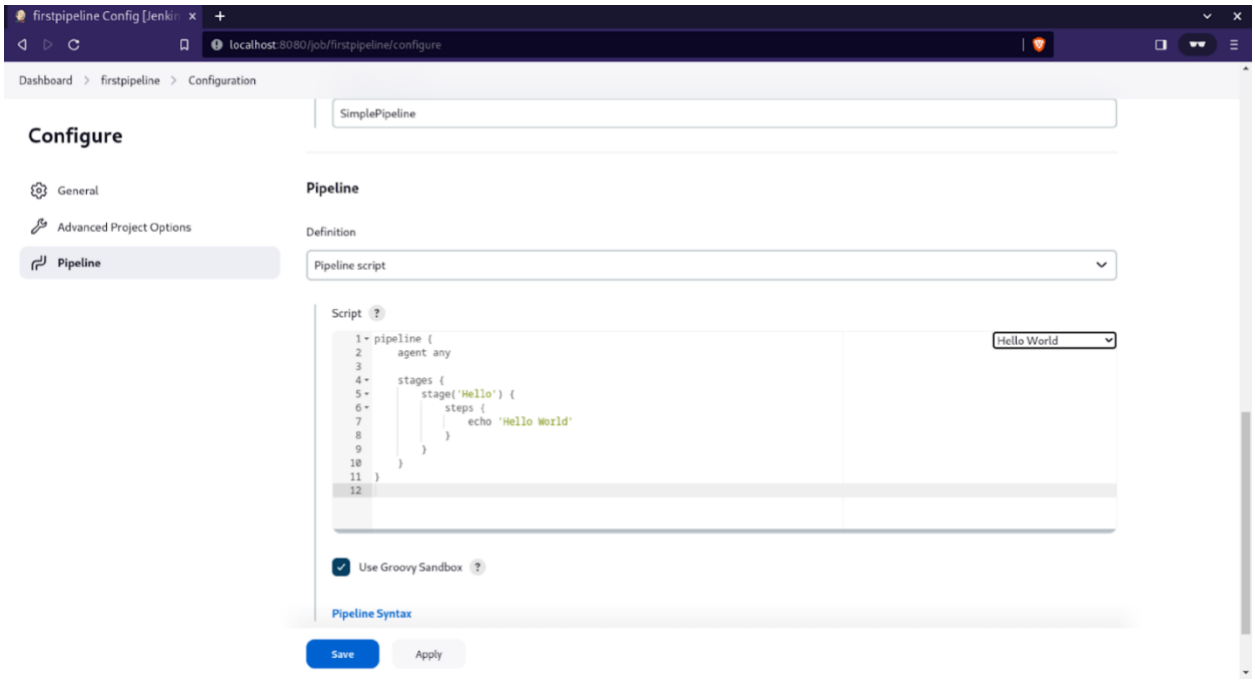
Steps for Jenkins CI/CD Pipeline:
Click on new item and create new pipeline



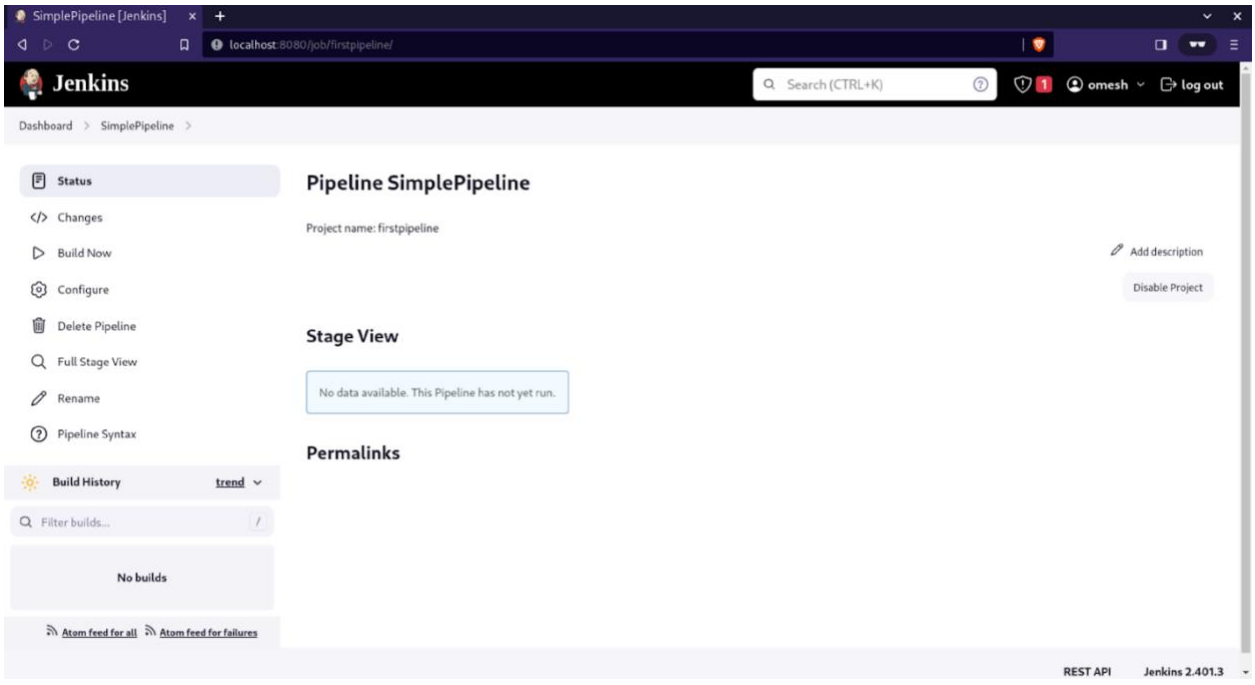
click ok ; and go to the advanced project options as shown in the below image
click on advanced dropdown and set the project display name



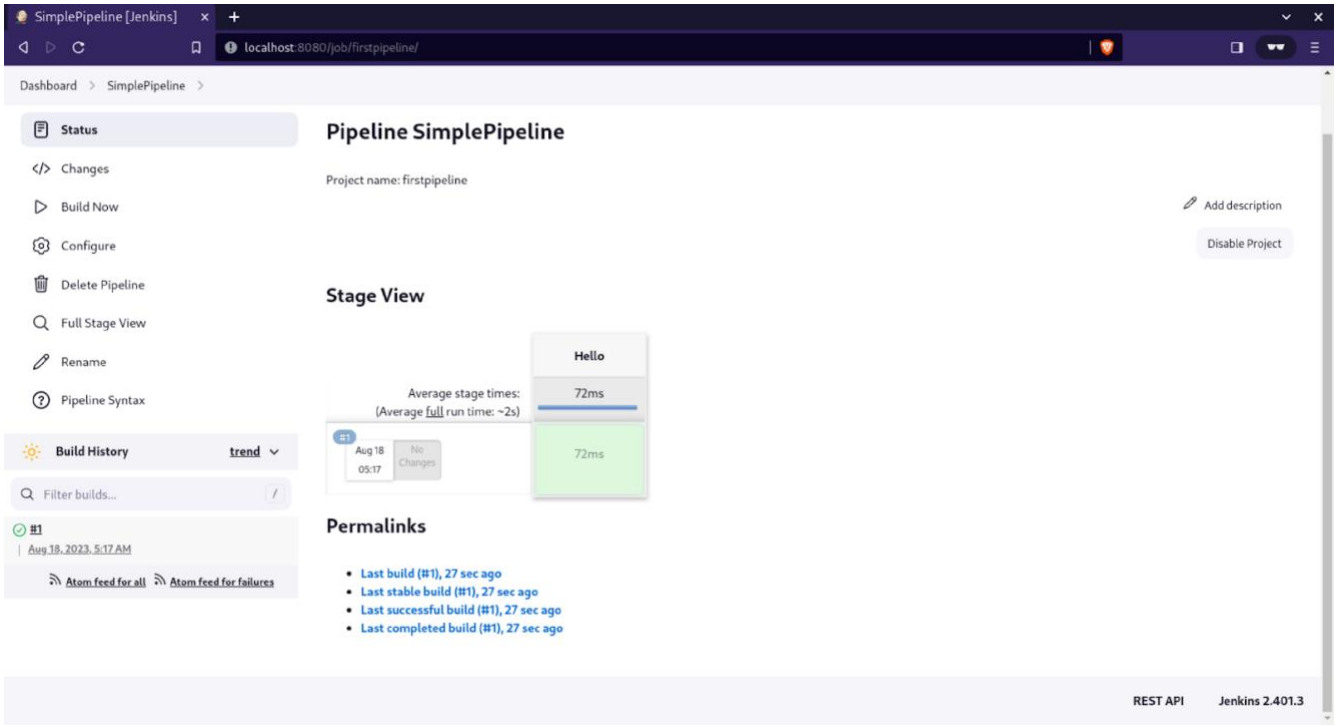
now go to the pipeline section ; choose pipeline script from the drop down ; now ,there will be a small dropdown showing at the right side of the script editor ; select Hello world from that drop down and save



now you will be in the dashboard of your pipeline

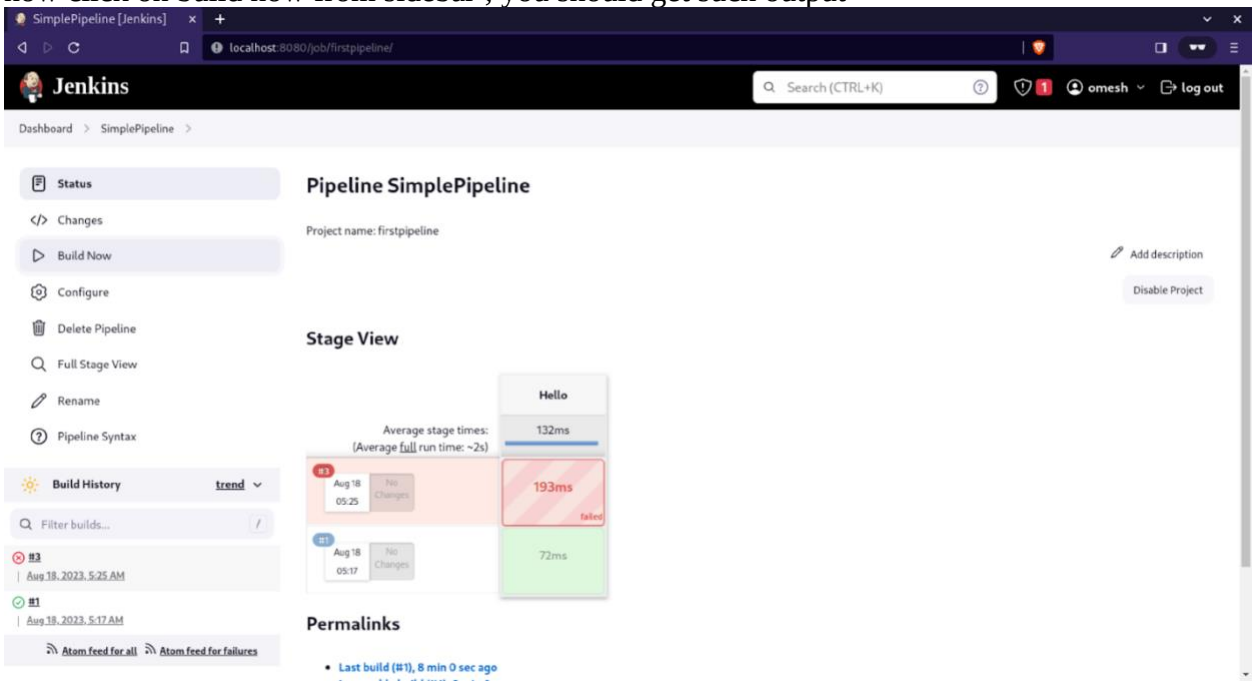


click build now from the sidebar available to the left ; wait for 1-2 minutes and your dashboard will be updated like this

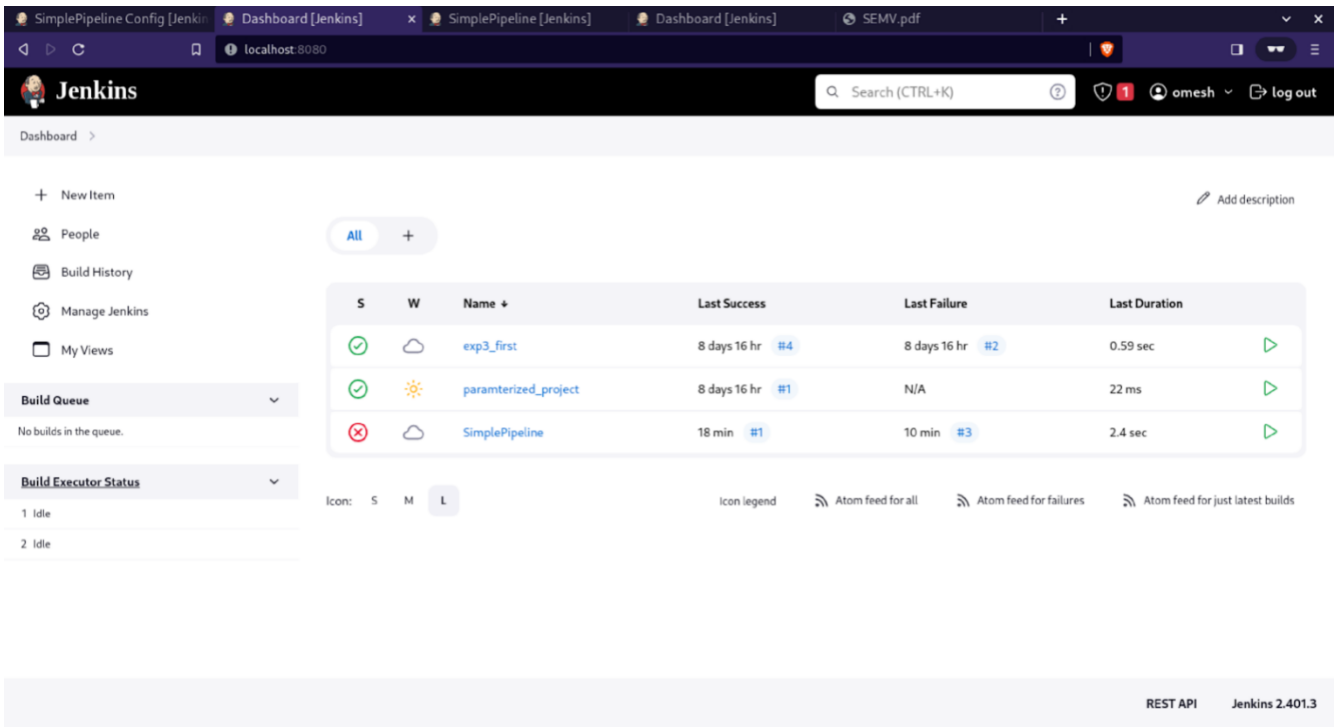


now our stage is deployed ; but that lets try some error in the script ; so first lets create some error in the script that we wrote in the script editor ; click on configure on the left sidebar
now go the pipeline section ; purposely make error in the script so that we can come to now about failure in the stage deployment ; sample of **(script with error is given below); click on save**

now click on build now from sidebar ; you should get such output



So till now we have known stage deployment ; lets learn how to build a job using jenkins pipeline click on configure from the sidebar ; and correct the error that we have made.
Now lets automate building of our freestyle project that we did in previous experiment my jenkins dashboard looks like

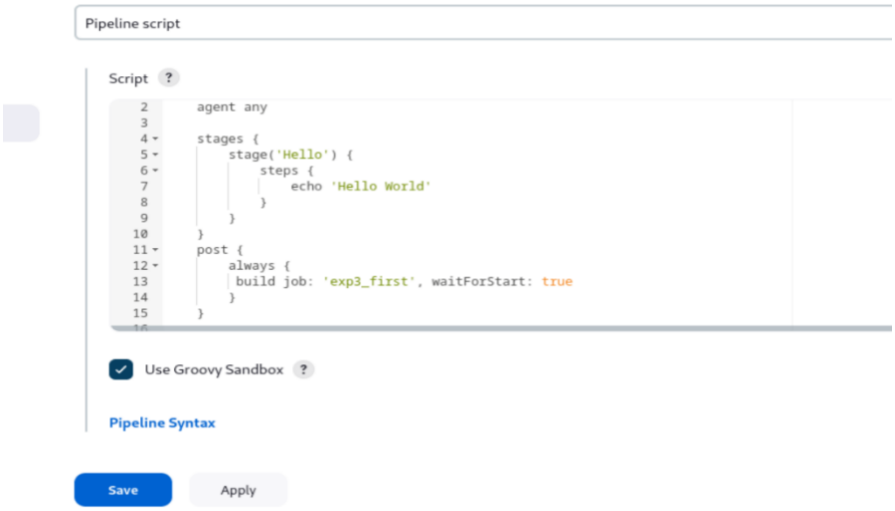


now lets automate building of ‘exp3_first’ project

Note :- project name may differ ; copy your project name my project name was ‘exp3_first’ ; use your project name in the pipeline script

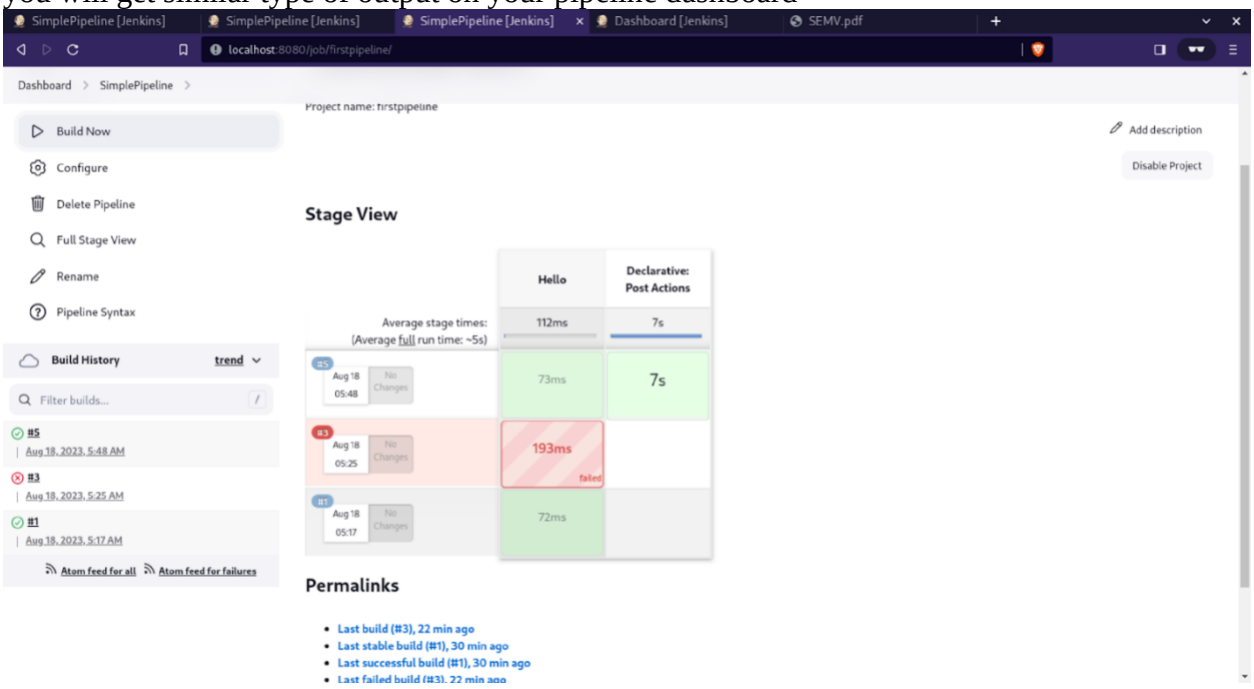
now click on your pipeline which is failed click on configure

go to the pipeline section ; edit script as below ; click on save



now click on build now from the sidebar

you will get similar type of output on your pipeline dashboard



Steps for Jenkins Parameterized Program:

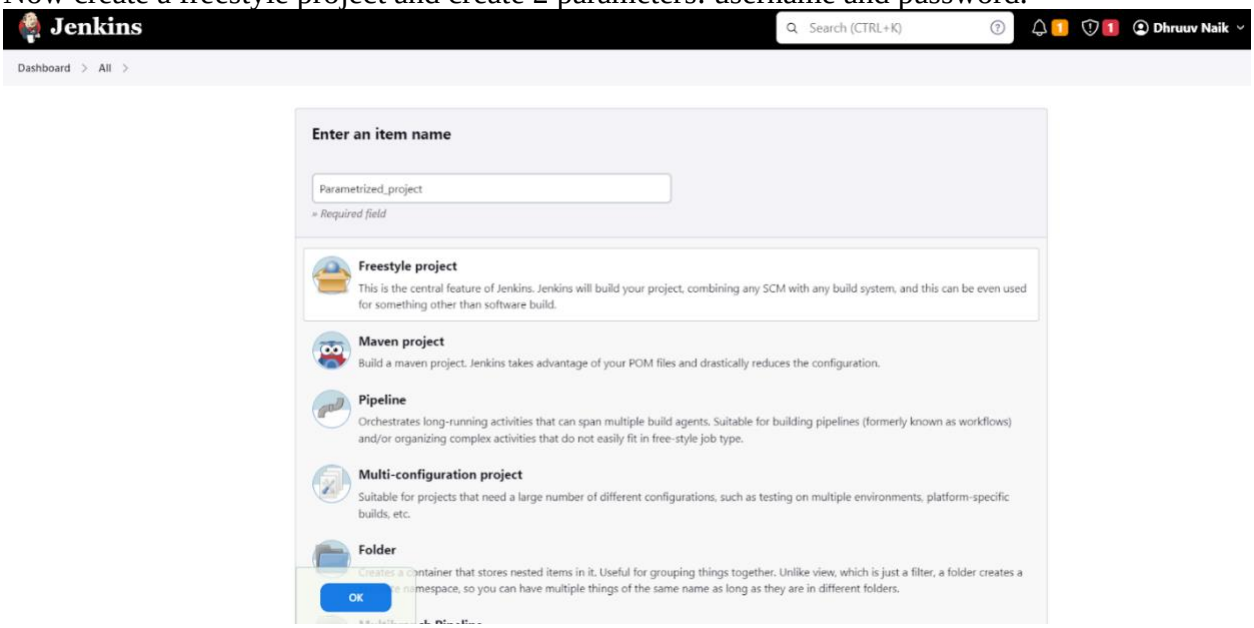
Terminal commands: Create a shell script and give it execution permission.

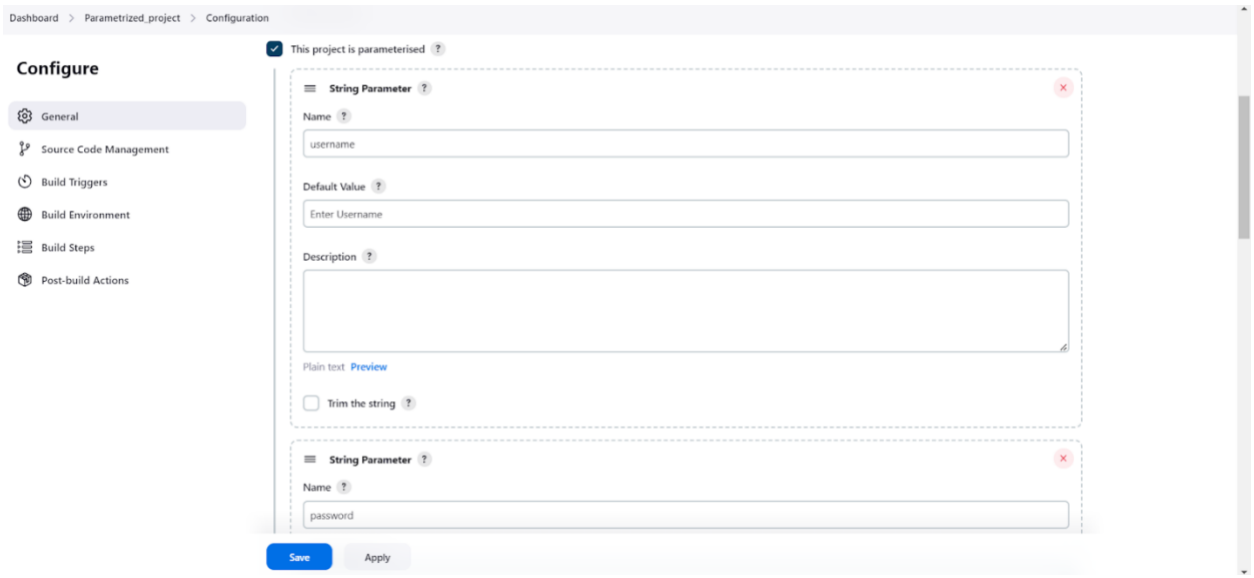
```
dhruu@D-Junior2:~/devops$ touch login.sh
dhruu@D-Junior2:~/devops$ cat login.sh
#!/bin/bash
```

```
username=$1
password=$2

if [[ "$username" == "admin" && "$password" == "Dbit2023" ]]; then
    echo "Hello $username"
else
    echo "Invalid username or password"
fi
```

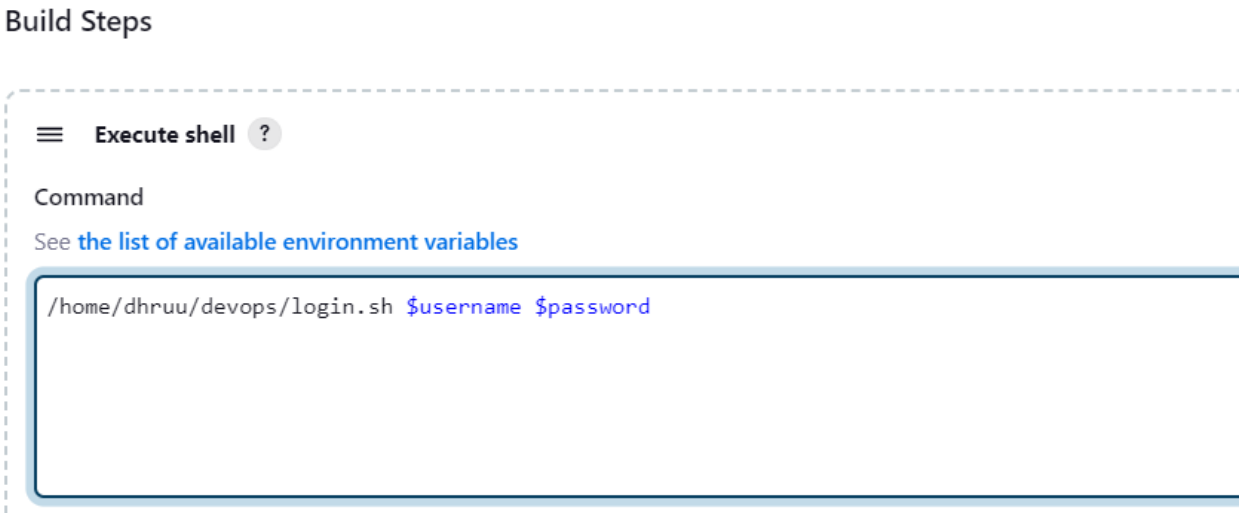
```
dhruu@D-Junior2:~/devops$ chmod +x login.sh
dhruu@D-Junior2:~/devops$ ./login.sh admin Dbit2023
Hello admin
```

Now create a freestyle project and create 2 parameters: username and password.





Under build step select Execute shell and copy the path to your shell script and don't forget to add parameters.



This is what the output would look like.

Conclusion: Successfully implemented shell script, pipeline and parameterized Jenkins program.

References:

https://www.youtube.com/watch?v=Rs-bTXQ8mjA
https://www.youtube.com/watch?v=AqITZLJ5eZ4
https://www.youtube.com/watch?v=I712r_Jtar4

Experiment 3:
Jenkin 3

Aim: To implement and execute Maven project

Procedure:
Step1: Check if Maven is available in the project style, if no install the plugin using
manage plugins -> plugin -> search for maven -> choose the button -> choose the item -> below click on download and install and restart button -> installation will start -> wait for a while -> once completed

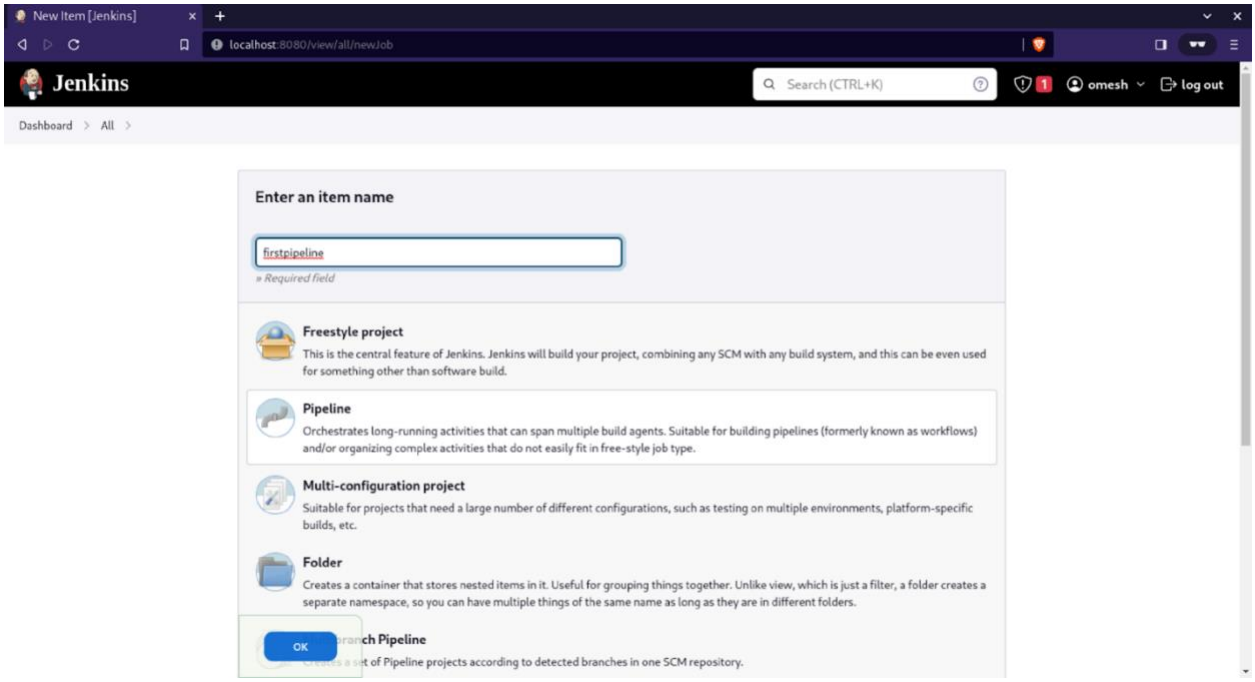
Conclusion:

References: https://www.youtube.com/watch?v=o2gAw929--g

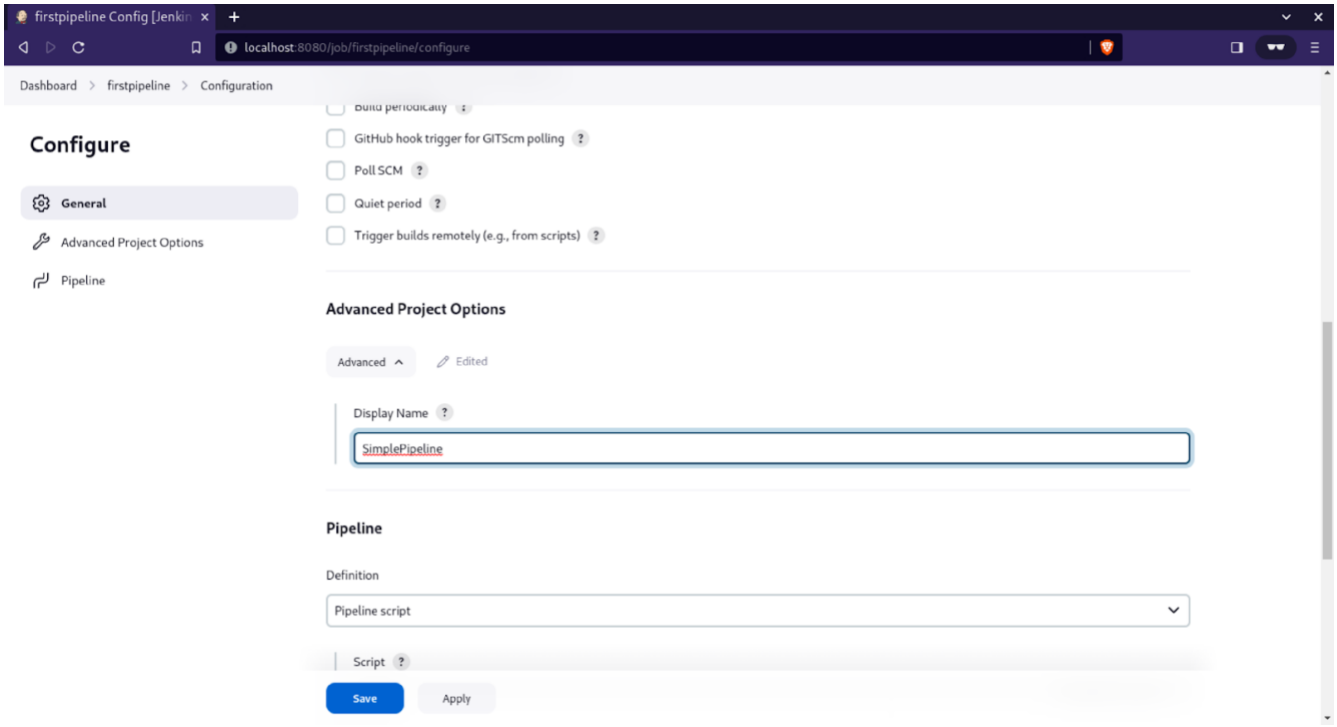
Assignment 1

Aim: CI/CD
Implementation of login Id & password using Jenkins pipeline
Mini project ISO file creation using Docker
Hosting of mini project ISO file on DockerHub

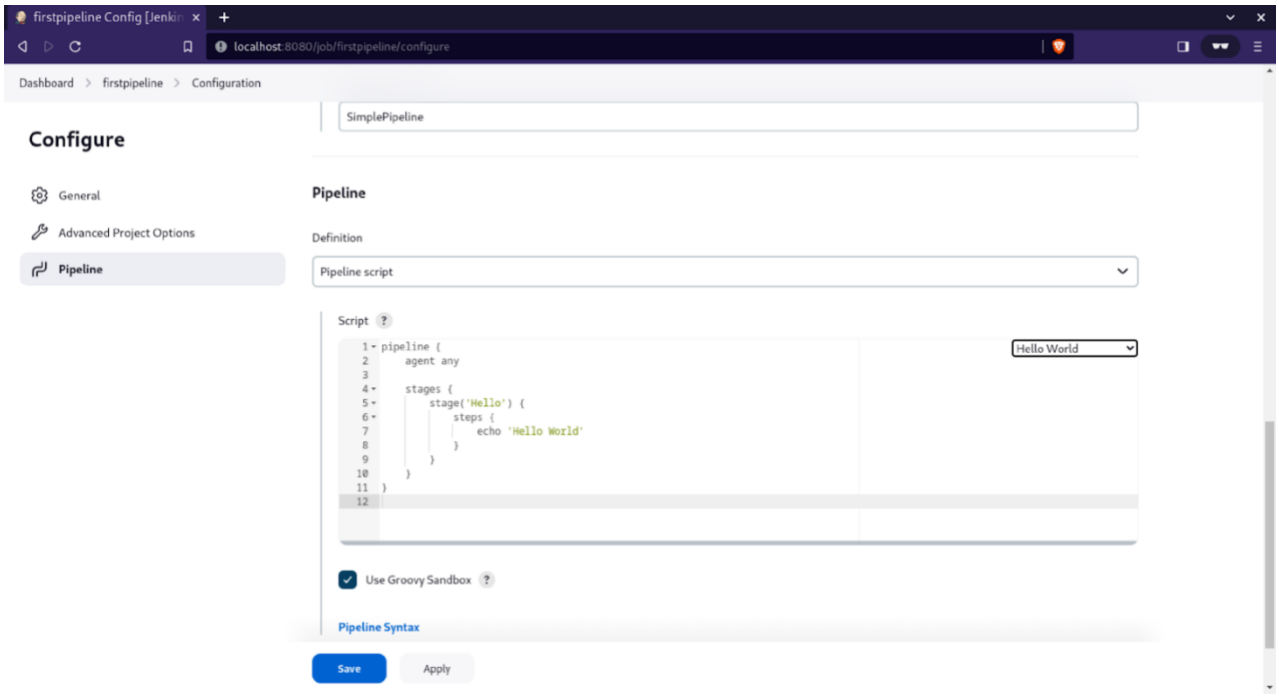
Steps for Jenkins CI/CD Pipeline:
Click on new item and create new pipeline



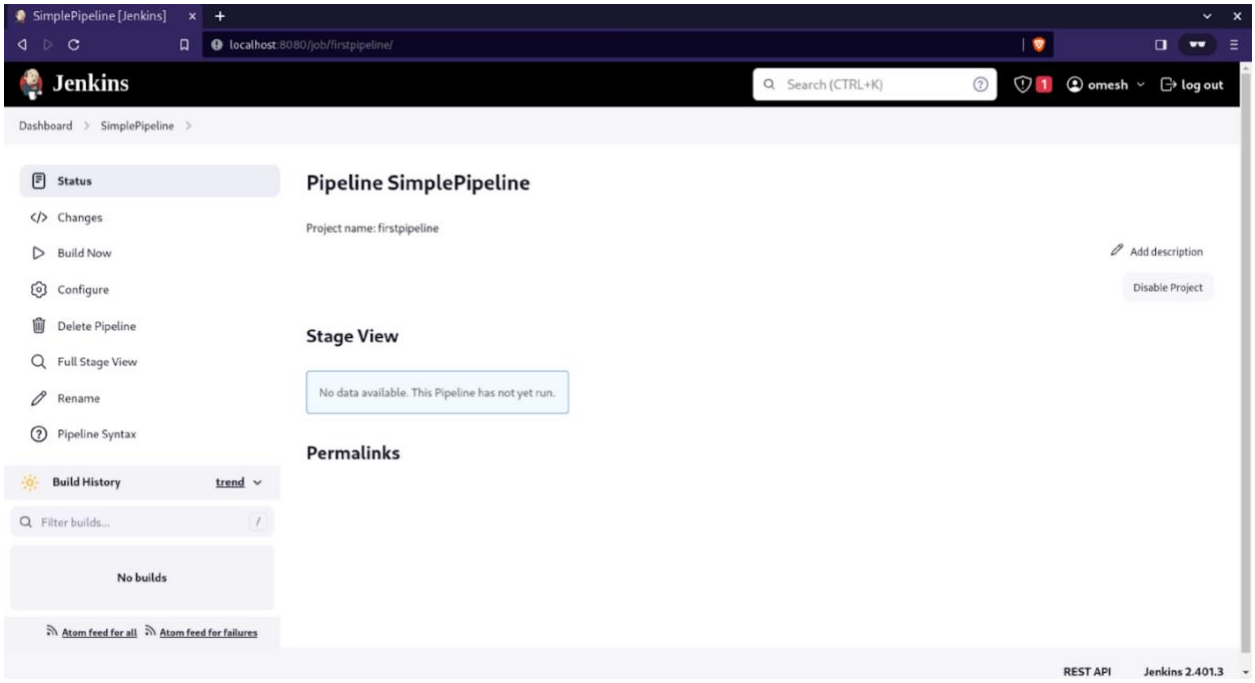
click ok ; and go to the advanced project options as shown in the below image
click on advanced dropdown and set the project display name



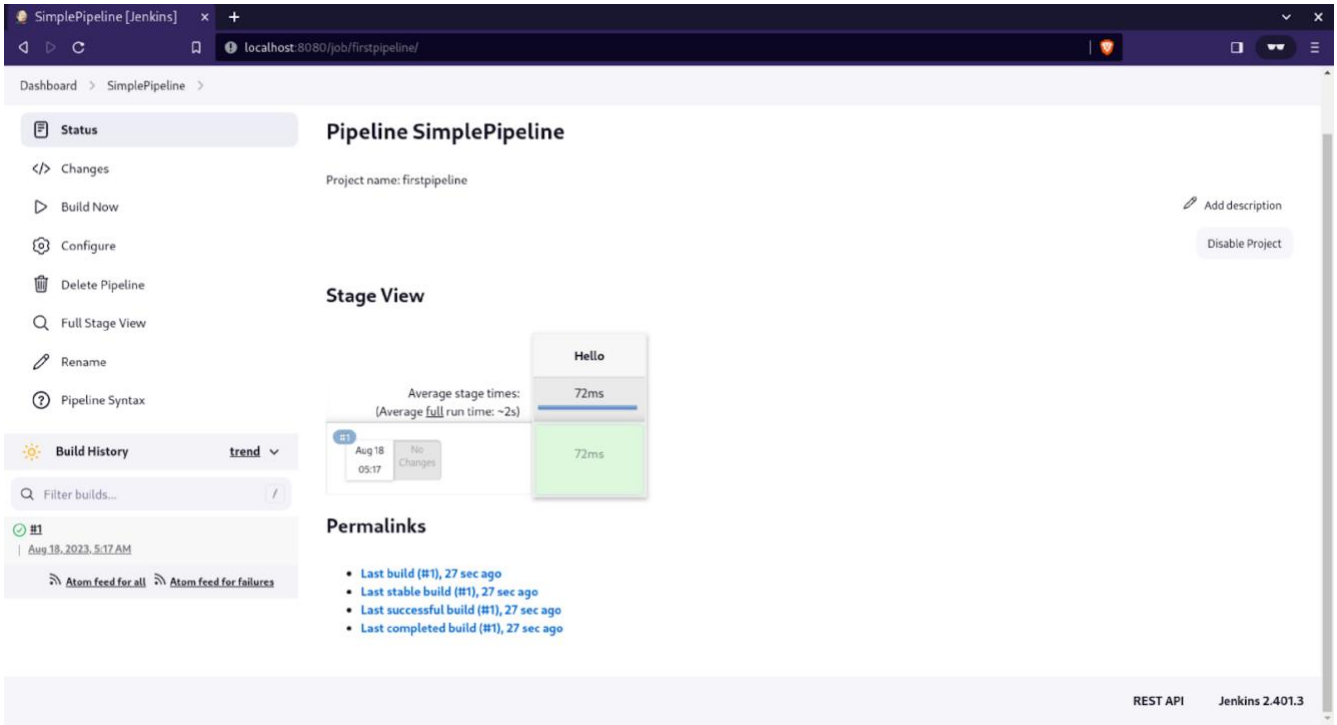
now go to the pipeline section ; choose pipeline script from the drop down ; now ,there will be a small dropdown showing at the right side of the script editor ; select Hello world from that drop down and save



now you will be in the dashboard of your pipeline



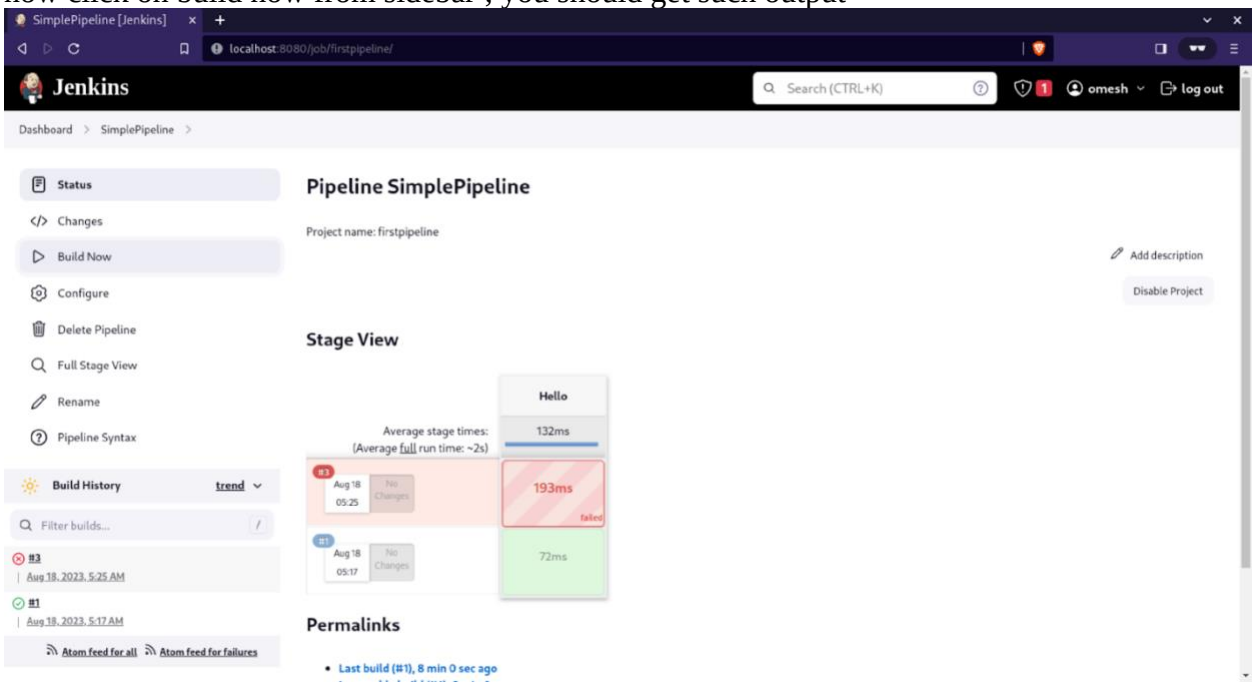
click build now from the sidebar available to the left ; wait for 1-2 minutes and your dashboard will be updated like this



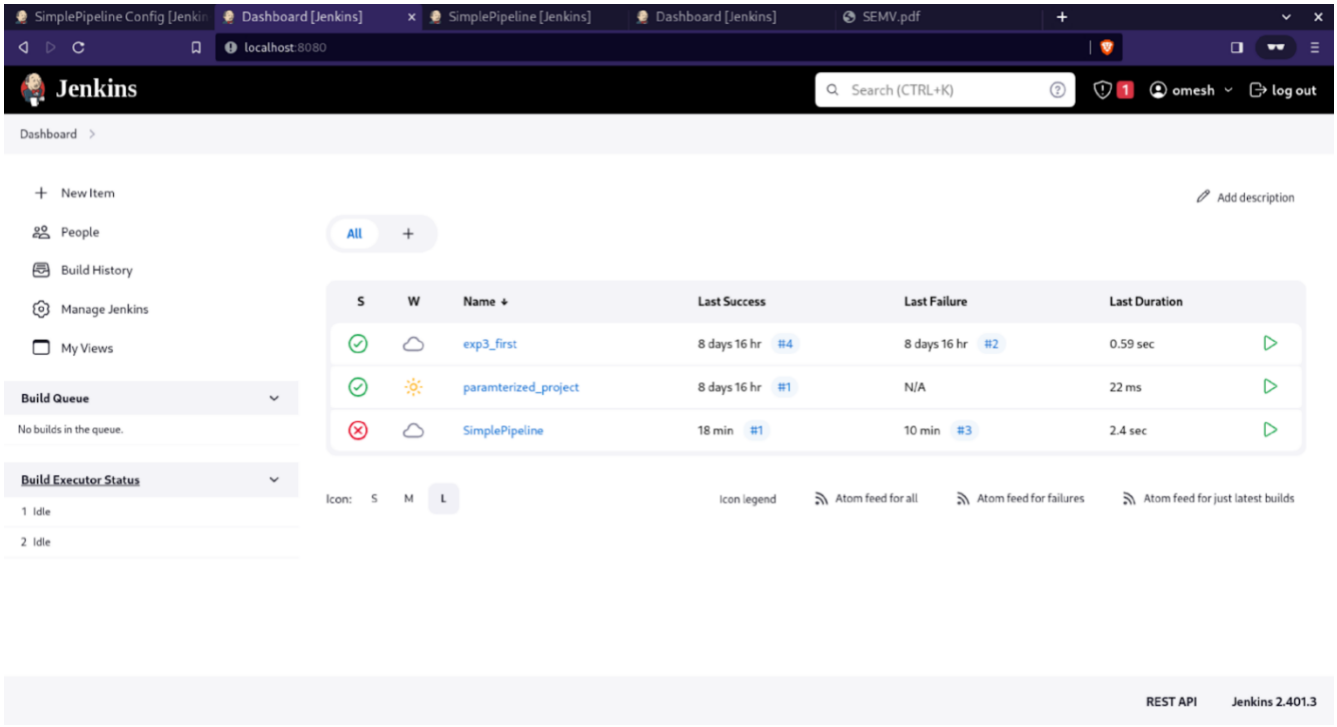
now our stage is deployed ; but that lets try some error in the script ; so first lets create some error in the script that we wrote in the script editor ; click on configure on the left sidebar

now go the pipeline section ; purposely make error in the script so that we can come to now about failure in the stage deployment ; sample of **(script with error is given below); click on save**

now click on build now from sidebar ; you should get such output



So till now we have known stage deployment ; lets learn how to build a job using jenkins pipeline click on configure from the sidebar ; and correct the error that we have made.
Now lets automate building of our freestyle project that we did in previous experiment my jenkins dashboard looks like

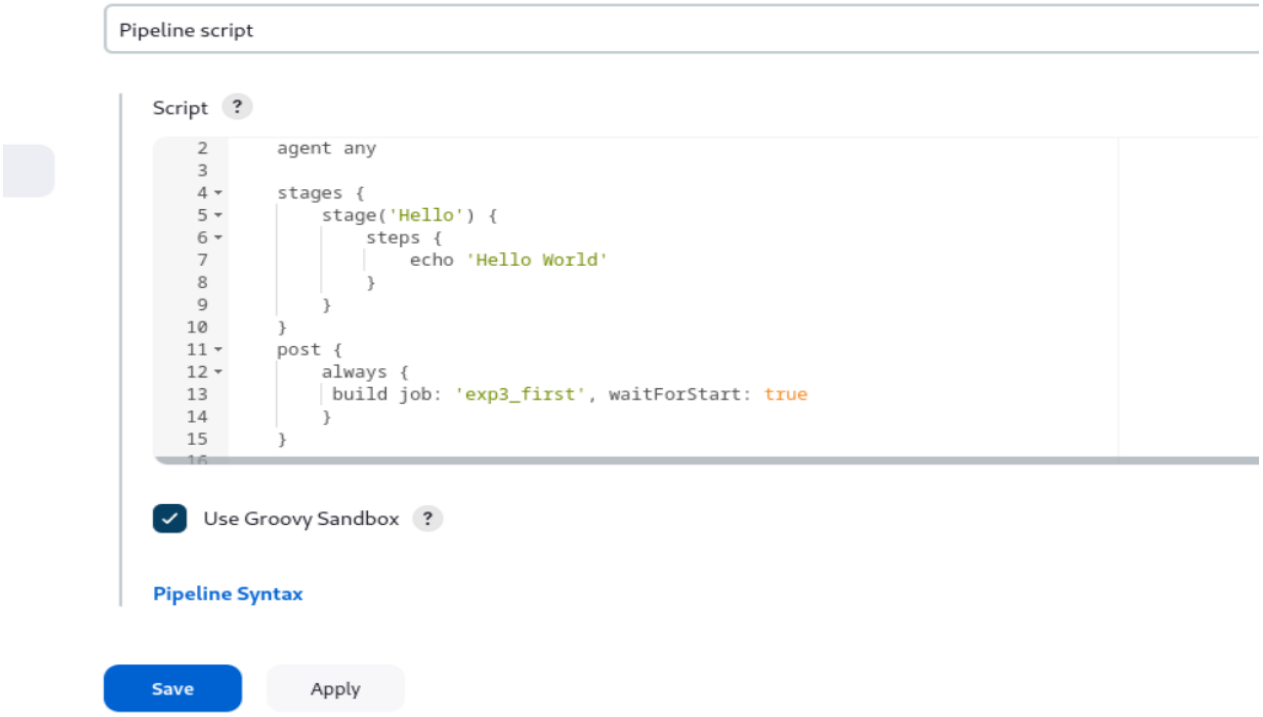


now lets automate building of ‘exp3_first’ project

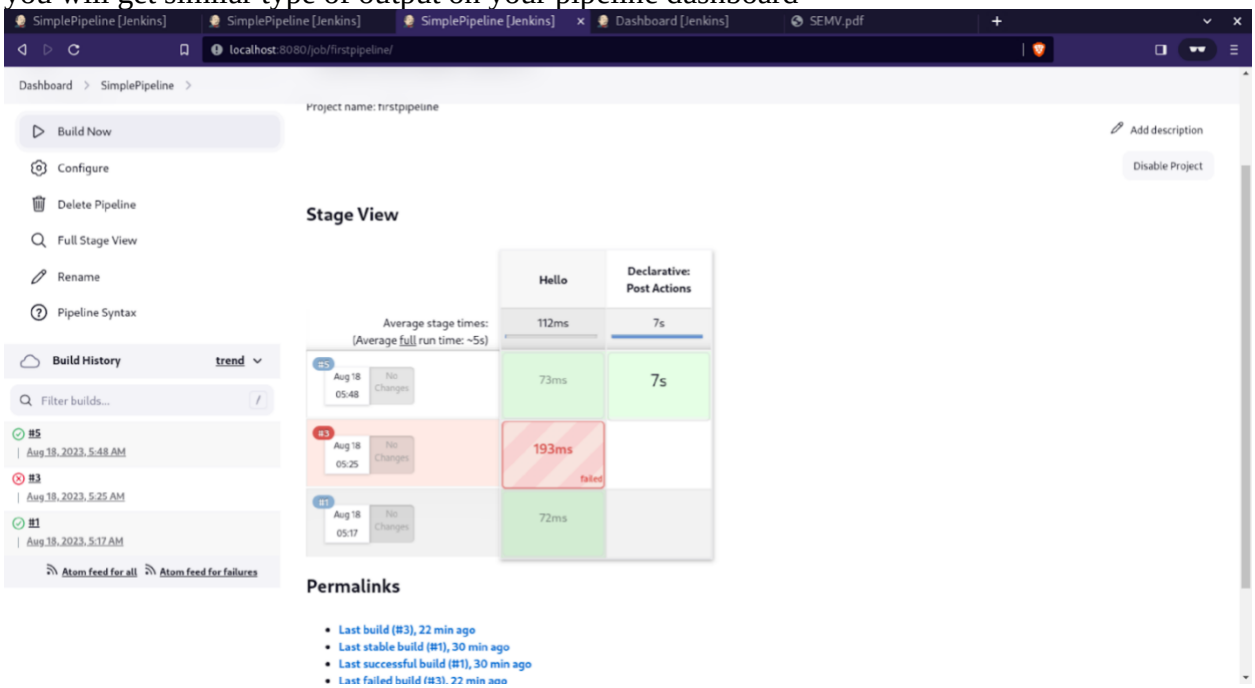
Note :- project name may differ ; copy your project name my project name was ‘exp3_first’ ; use your project name in the pipeline script

now click on your pipeline which is failed click on configure

go to the pipeline section ; edit script as below ; click on save



now click on build now from the sidebar
you will get similar type of output on your pipeline dashboard



Steps for Mini project ISO file creation using Docker & hosting of mini project ISO file on DockerHub

Step 1: Create a directory and add dockerfile in it

```
PS D:\docker_assignment1> ls


    Directory: D:\docker_assignment1


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        15-10-2023     14:34              0 dockerfile
```

Step 2: Add the website files in the directory and following code in dockerfile
FROM httpd
COPY . /usr/local/apache2/htdocs

```
PS D:\docker_assignment1> cat .\dockerfile
FROM httpd
COPY . /usr/local/apache2/htdocs
PS D:\docker_assignment1> ls


    Directory: D:\docker_assignment1


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        15-10-2023     14:36             44 dockerfile
-a----        28-09-2023     17:35           3441 index.html
-a----        12-08-2023     22:40           1268 style.css
```

Step 3: Build the dockerfile into an ISO
docker build -t website .

```
PS D:\docker_assignment1> docker build -t website .
[+] Building 11.5s (8/8) FINISHED                                                                                  docker:default
 => [internal] load build definition from dockerfile                                                                         0.1s
 => => transferring dockerfile: 81B                                                                                          0.0s
 => [internal] load .dockerignore                                                                                            0.1s
 => => transferring context: 2B                                                                                              0.0s
 => [internal] load metadata for docker.io/library/httpd:latest                                                              3.9s
 => [auth] library/httpd:pull token for registry-1.docker.io                                                                 0.0s
 => [internal] load build context                                                                                            0.1s
 => => transferring context: 4.87kB                                                                                          0.0s
 => [1/2] FROM docker.io/library/httpd@sha256:5201524443f9026753e25540a44495b7f6e6ca706c71208bb3a5f2daac205c31               7.1s
 => => resolve docker.io/library/httpd@sha256:5201524443f9026753e25540a44495b7f6e6ca706c71208bb3a5f2daac205c31               0.0s
 => => sha256:c20157372e943d84bb5a0624e80395697de1f41ecd54b3bcead2b03bb6b13fe8 176B / 176B                                   0.5s
 => => sha256:073cbcfef6634b5131786873a7a92a3b3bda43672e5830126dfa94352649358d 4.20MB / 4.20MB                               0.7s
 => => sha256:5201524443f9026753e25540a44495b7f6e6ca706c71208bb3a5f2daac205c31 1.86kB / 1.86kB                               0.0s
 => => sha256:28074b6f3722be675b200c1c87c288066e90972e96ee51d87f83d763dc4e09a1 1.37kB / 1.37kB                               0.0s
 => => sha256:ca77aadc3cbc20bfe57686d7d715acb10be6904ce303d9632b73c6fe4c5f6b85 9.38kB / 9.38kB                               0.0s
 => => sha256:a378f10b321842c3042cdeff4f6997f34f4cb21f2eff27704b7f6193ab7b5fea 29.15MB / 29.15MB                             3.0s
 => => sha256:c36006acf55e9d0c608ef6466d254a69faa43f7be8fe065d1b2a340d9002054b 31.37MB / 31.37MB                             3.0s
 => => sha256:e94c45cb708a06cd1826d92621df930cba22fbd733fb20867b72c1871b605ca3 297B / 297B                                   1.1s
 => => extracting sha256:a378f10b321842c3042cdeff4f6997f34f4cb21f2eff27704b7f6193ab7b5fea                                    2.3s
 => => extracting sha256:c20157372e943d84bb5a0624e80395697de1f41ecd54b3bcead2b03bb6b13fe8                                    0.0s
 => => extracting sha256:073cbcfef6634b5131786873a7a92a3b3bda43672e5830126dfa94352649358d                                    0.2s
 => => extracting sha256:c36006acf55e9d0c608ef6466d254a69faa43f7be8fe065d1b2a340d9002054b                                    1.1s
 => => extracting sha256:e94c45cb708a06cd1826d92621df930cba22fbd733fb20867b72c1871b605ca3                                    0.0s
 => [2/2] COPY . /usr/local/apache2/htdocs                                                                                   0.3s
 => exporting to image                                                                                                       0.1s
 => => exporting layers                                                                                                      0.0s
 => => writing image sha256:402ec1cc43c125373d3a2815cb6f9e3fc149963eac3cb3278b31deeafafc7f01                                 0.0s
 => => naming to docker.io/library/website                                                                                   0.0s

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview
```

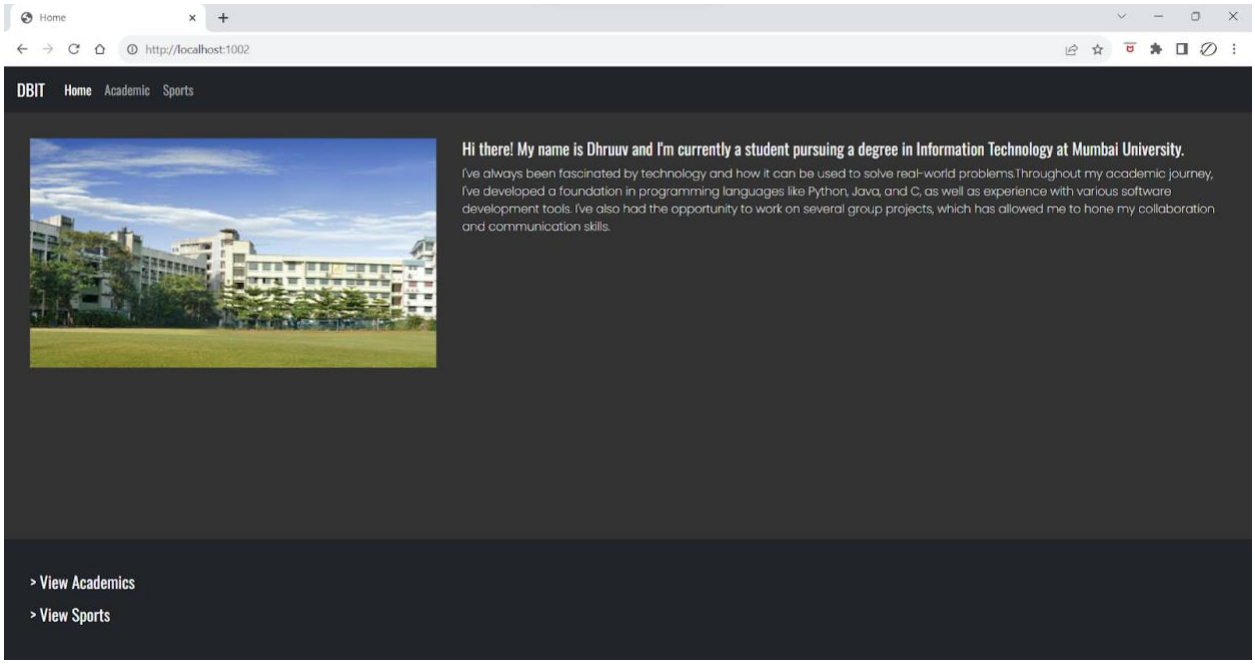
docker images

```
PS D:\docker_assignment1> docker images
REPOSITORY           TAG       IMAGE ID       CREATED              SIZE
website              latest    402ec1cc43c1   About a minute ago   168MB
dhruuvnaik5/devops   latest    7707c0fe8f27   5 days ago           77.8MB
ubuntu               latest    3565a89d9e81   2 weeks ago          77.8MB
PS D:\docker_assignment1> docker run -d -p 1002:80 dhruuvwebsite
Unable to find image 'dhruuvwebsite:latest' locally
docker: Error response from daemon: pull access denied for dhruuvwebsite, repository does not exist or may require 'docker login': denied: requested a
to the resource is denied.
See 'docker run --help'.
PS D:\docker_assignment1> docker run -d -p 1002:80 website
8971da0f8d8b87fcbb597661bf01d5f66e2d055459529b1aae7604fab2d01238
PS D:\docker_assignment1> docker ps -a
CONTAINER ID   IMAGE               COMMAND              CREATED          STATUS                     PORTS
   NAMES
8971da0f8d8b   website             "httpd-foreground"   37 seconds ago   Up 36 seconds              0.0.0.0:1002->80/tcp   ecstatic_lalande
eaf942a20831   dhruuvnaik5/devops   "bash"               5 days ago       Exited (137) 5 days ago
   intelligent_nash
29492348494a   ubuntu              "/bin/bash"          5 days ago       Exited (137) 5 days ago
```

Step 4: When you run this command, it will start a Docker container from the website image, and you can access the website within the container by visiting http://localhost:1002 in your web browser, assuming your Docker daemon is running on your local machine.

docker run -d -p 1002:80 website
docker ps -a


```
PS D:\docker_assignment1> docker images
REPOSITORY           TAG       IMAGE ID       CREATED              SIZE
website              latest    402ec1cc43c1   About a minute ago   168MB
dhruuvnaik5/devops   latest    7707c0fe8f27   5 days ago           77.8MB
ubuntu               latest    3565a89d9e81   2 weeks ago          77.8MB
PS D:\docker_assignment1> docker run -d -p 1002:80 dhruuvwebsite
Unable to find image 'dhruuvwebsite:latest' locally
docker: Error response from daemon: pull access denied for dhruuvwebsite, repository does not exist or may require 'docker login': denied: re
to the resource is denied.
See 'docker run --help'.
PS D:\docker_assignment1> docker run -d -p 1002:80 website
8971da0f8d8b87fcbb597661bf01d5f66e2d055459529b1aae7604fab2d01238
PS D:\docker_assignment1> docker ps -a
CONTAINER ID   IMAGE               COMMAND              CREATED          STATUS                     PORTS
   NAMES
8971da0f8d8b   website             "httpd-foreground"   37 seconds ago   Up 36 seconds              0.0.0.0:1002->80/tcp   ecstatic_lalande
eaf942a20831   dhruuvnaik5/devops   "bash"               5 days ago       Exited (137) 5 days ago
   intelligent_nash
29492348494a   ubuntu              "/bin/bash"          5 days ago       Exited (137) 5 days ago
```



Step 5: This command tags the "website" image with a new name "dhruuvnaik5/website." If there is a tag associated with the "website" image, it will also be inherited by the new image.

docker ps -a
docker image tag [image name] [username]/[repository name]
docker images

```
PS D:\docker_assignment1> docker ps -a
CONTAINER ID   IMAGE                  COMMAND              CREATED          STATUS                      PORTS                    NAMES
8971da0f8d8b   website                "httpd-foreground"   37 seconds ago   Up 36 seconds               0.0.0.0:1002->80/tcp     ecstatic_lalande
eaf942a20831   dhruuvnaik5/devops     "bash"               5 days ago       Exited (137) 5 days ago                              intelligent_nash
29492348494a   ubuntu                 "/bin/bash"          5 days ago       Exited (137) 5 days ago                              reverent_maxwell
PS D:\docker_assignment1> docker image tag website dhruuvnaik5/website
PS D:\docker_assignment1> docker images
REPOSITORY             TAG       IMAGE ID       CREATED          SIZE
dhruuvnaik5/website    latest    402ec1cc43c1   3 minutes ago    168MB
website                latest    402ec1cc43c1   3 minutes ago    168MB
dhruuvnaik5/devops     latest    7707c0fe8f27   5 days ago       77.8MB
ubuntu                 latest    3565a89d9e81   2 weeks ago      77.8MB
```

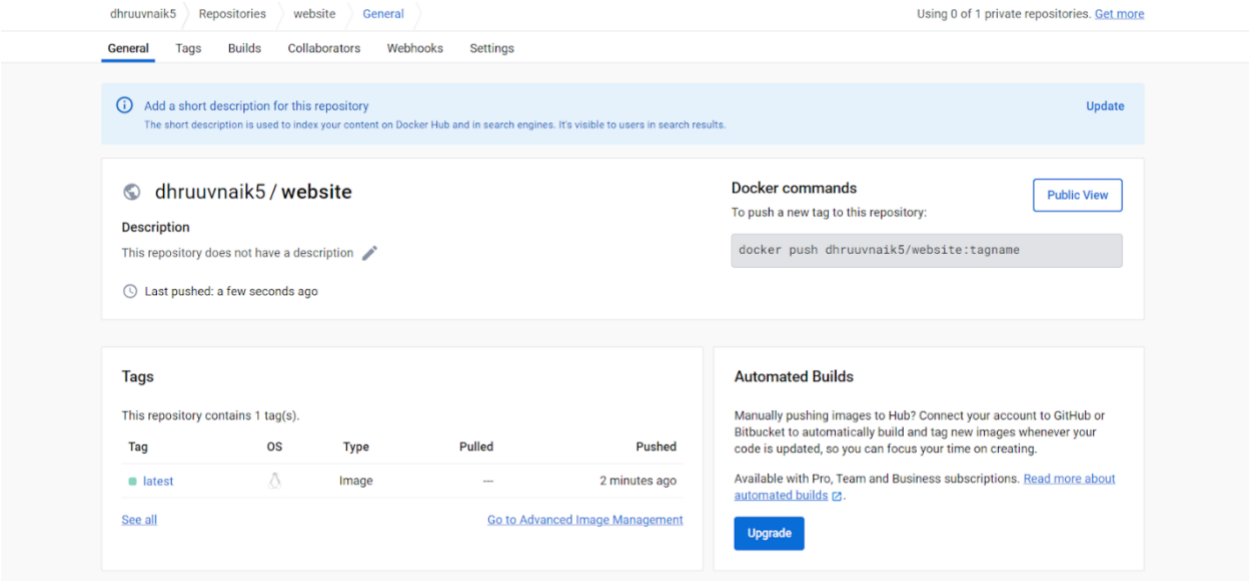
Step 6: Push the image onto DockerHub
docker push [username]/[repository name]

```
PS D:\docker_assignment1> docker push dhruuvnaik5/website
Using default tag: latest
The push refers to repository [docker.io/dhruuvnaik5/website]
341eb2ac1031: Pushed
34a45e1e8c59: Mounted from library/httpd
16a05025a277: Mounted from library/httpd
1343ea427053: Mounted from library/httpd
8db3e477577e: Mounted from library/httpd
cb4596cc1454: Mounted from library/httpd
latest: digest: sha256:47fd10858064adc667a70eb362c5029fafb3babb95908c1d2aa31db1021184a0 size: 1574
```

Step 7: Now you may stop the running containers
docker stop [containerId]
docker ps -a

```
PS D:\docker_assignment1> docker stop 8971da0f8d8b
8971da0f8d8b
PS D:\docker_assignment1> docker ps -a
CONTAINER ID   IMAGE                COMMAND                  CREATED          STATUS                      PORTS     NAMES
8971da0f8d8b   website              "httpd-foreground"       2 minutes ago    Exited (0) 4 seconds ago              ecstatic_lalande
eaf942a20831   dhruuvnaik5/devops   "bash"                   5 days ago       Exited (137) 5 days ago               intelligent_nash
29492348494a   ubuntu               "bin/bash"               5 days ago       Exited (137) 5 days ago               reverent_maxwell
PS D:\docker_assignment1>
```

Conclusion: Successfully implemented CI/CD pipeline in Jenkins and uploaded Mini Project ISO file on DockerHub



References: [Convert Web application to Docker Image by using Dockerfile | How to create Docker image]

Experiment 5:
Docker installation and basic commands

Prerequisite: Knowledge about Docker, DockerHub, Container and Virtual Machine

Aim: To install docker and practice basic commands

Steps for Installing Docker on Ubuntu:

1. Open the terminal on Ubuntu.

2. Remove any Docker files that are running in the system, using the following command:

```
$ sudo apt-get remove docker docker-engine docker.io
```

After entering the above command, you will need to enter the password of the root and press enter.

3. Check if the system is up-to-date using the following command:

```
$ sudo apt-get update
```

4. Install Docker using the following command:

```
$ sudo apt install docker.io
```

You'll then get a prompt asking you to choose between y/n - choose y

5. Install all the dependency packages using the following command:

```
$ sudo snap install docker
```

6. Before testing Docker, check the version installed using the following command:

```
$ docker --version
```

7. Pull an image from the Docker hub using the following command:

```
$ sudo docker run hello-world
```

Here, hello-world is the docker image present on the Docker hub.

8. Check if the docker image has been pulled and is present in your system using the following command:

```
$ sudo docker images
```

9. To display all the containers pulled, use the following command:

```
$ sudo docker ps -a
```

10. To check for containers in a running state, use the following command:

```
$ sudo docker ps
```

You've just successfully installed Docker on Ubuntu!

Basic Commands:

1. docker –version

This command is used to get the current version of the docker

docker - -version [OPTIONS]

By default, this will render all version information in an easy-to-read layout.

2. docker pull

Pull an image or a repository from a registry

docker pull [OPTIONS] NAME[: TAG|@DIGEST]

To download an image or set of images (i.e. A Repository) , Once can use docker pull command

Example:

$ docker pull dockerimage

3. docker run

This command is used to create a container from an image

docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

The docker run command creates a writeable container layer over the specified image and then starts it using the specified command.

The docker run command can be used with many variations, One can refer to the following documentation docker run.

4. docker ps

This command is used to list all the containers

```
docker ps [OPTIONS]
```

The above command can be used with other options like - all or –a

docker ps -all: Lists all containers

Example:

$ docker ps

$ docker ps -a

5. docker exec

This command is used to run a command in a running container

```
docker exec [OPTIONS] CONTAINER COMMAND [ARG...]
```

Docker exec command runs a new command in a running container.

Refer to the following article for more detail regarding the usage of the docker exec command docker exec.

6. docker stop

This command is used to stop one or more running containers.

```
docker stop [OPTIONS] CONTAINER [CONTAINER...]
```

The main process inside the container will receive SIGTERM, and after a grace period, SIGKILL. The first signal can be changed with the STOPSIGNAL instruction in the container's Dockerfile, or the --stop-signal option to docker run.

Example:

$ docker stop my_container

7. docker restart

This command is used to restart one or more containers.

```
docker restart [OPTIONS] CONTAINER [CONTAINER...]
```

Example:

```
$ docker restart my_container
```

8. docker kill

This command is used to kill one or more containers.

```
docker kill [OPTIONS] CONTAINER [CONTAINER...]
```

Example:

```
$ docker kill my_container
```

9. docker commit

This command is used to create a new image from the container image.

```
docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]
```

Docker commit command allows users to take an existing running container and save its current state as an image

There are certain steps to be followed before running the command

- First , Pull the image from docker hub
- Deploy the container using the image id from first step
- Modify the container (Any changes ,if needed)
- Commit the changes

Example:

```
$ docker commit c3f279d17e0a dev/testimage:version3.
```

10. docker push

This command is used to push an image or repository to a registry.

```
docker push [OPTIONS] NAME[: TAG]
```

Use docker image push to share your images to the Docker Hub registry or to a self-hosted one.

Example:

$ docker image push registry-host:5000/myadmin/rhel-httpd:lates

Conclusion: Successfully installed Docker and pulled images

References:

https://www.simplilearn.com/tutorials/docker-tutorial/how-to-install-docker-on-ubuntu
https://www.knowledgehut.com/blog/devops/basic-docker-commands
https://hub.docker.com/
https://www.oreilly.com/member/login/

.

Assignment 2

Chef

What is Chef?

Chef is a powerful configuration management tool that automates the process of configuring and managing servers. It allows developers and system administrators to define the state of their infrastructure as code, enabling the automation of repetitive tasks, ensuring consistency, and simplifying the management of complex systems.

Features of Chef

1. Recipes and Cookbooks: Chef uses recipes and cookbooks to define how resources should be configured. Recipes describe a particular configuration or policy, while cookbooks are collections of recipes and other resources needed to configure a particular piece of infrastructure.
2. Resource Abstraction: Chef abstracts the configuration of resources, allowing users to describe the desired state of their systems without needing to specify how to achieve that state. This allows for greater flexibility and portability across different operating systems and environments.
3. Chef Server: The Chef server acts as a central hub for configuration data. It stores information about the nodes in the infrastructure, the cookbooks, and the policies that should be applied to each node.
4. Chef Client: The Chef client runs on each node in the infrastructure and is responsible for pulling configuration information from the Chef server and applying it to the local system. The Chef client ensures that the desired state of the system matches the state defined in the recipes and cookbooks.
5. Ohai: Ohai is a tool that is bundled with Chef, which collects system configuration data to provide a comprehensive view of each node's current state. This data is then used by the Chef client to determine how best to apply the desired configurations.

Installation And Setup

To install the Chef Server, follow these steps.

1. Download the Chef Server core using wget.

wget https://packages.chef.io/files/stable/chef-server/15.1.7/ubuntu/20.04/chef-server-core_15.1.7-1_amd64.deb

2. Install the server core.

```
sudo dpkg -i chef-server-core_*.deb
```

3. For better security and to preserve server space, remove the downloaded .deb file.

```
rm chef-server-core_*.deb
```

4. Start the Chef server. Answer yes when prompted to accept the product licenses.

Note
The installation process takes several minutes to complete. Upon a successful installation, the message Chef Infra Server Reconfigured! is displayed.

```
sudo chef-server-ctl reconfigure
```

How to Configure a Chef User and Organization

To use Chef, configure an organization and at least one user on the Chef Server. This enables server access for workstations and nodes. To create these accounts, follow these steps.

1. Create a .chef directory to store the keys. This should be a subdirectory located inside the home directory.

```
mkdir .chef
```

2. Use the chef-server-ctl command to create a user account for the Chef administrator. Additional user accounts can be created later. Replace the USER_NAME, FIRST_NAME, LAST_NAME, EMAIL, and PASSWORD fields with the relevant information. For the --filename argument, replace USER_NAME.pem with the user name used earlier in the command.

```
sudo chef-server-ctl user-create USER_NAME FIRST_NAME LAST_NAME EMAIL 'PASSWORD' --filename ~/.chef/USER_NAME.pem
```

3. Review the user list and confirm the account now exists.

```
sudo chef-server-ctl user-list
```

4. Create a new organization, also using the chef-server-ctl command.
Replace ORG_NAME and ORG_FULL_NAME with the actual name of the organization. The ORG_NAME field must be all lower case. The value for USER_NAME must be the same name used in the user-create command. For the --filename argument, in ORG_NAME.pem, replace ORG_NAME with the organization name used elsewhere in the command.

```
sudo chef-server-ctl org-create ORG_NAME "ORG_FULL_NAME" --association_user USER_NAME --filename ~/.chef/ORG_NAME.pem
```

5. List the organizations to confirm the new organization is successfully created.

```
sudo chef-server-ctl org-list
```

How to Install and Configure a Chef Workstation

A Chef Workstation is for users to create and test recipes. Any Linode with at least 2GB of memory can be used for this task. Unlike the Chef Server, a workstation can also be used for other tasks. However, in a larger organization hosting many users, it is often efficient to centralize workstation activities on one server hosting multiple accounts.

How to Install a Chef Workstation

The steps for installing a Chef Workstation are similar to those for installing the Server. Download the correct file using wget, then install it. To install a Chef Workstation, follow these steps.

1. Download the source files for the Chef Workstation. For different releases of the Workstation, or downloads for earlier releases, see the Chef Workstation Downloads page. For more information on the installation process, see the Chef Workstation Installation Documentation.

```
wget https://packages.chef.io/files/stable/chef-workstation/22.10.1013/ubuntu/20.04/chef-workstation_22.10.1013-1_amd64.deb
```

2. Install the Chef Workstation.

```
sudo dpkg -i chef-workstation_*.deb
```

3. Remove the source file.

```
rm chef-workstation_*.deb
```

4. Confirm the correct release of the Chef Workstation is installed.

```
chef -v
```

How to Configure a Chef Workstation

A few more items must be configured before the Workstation is operational. Tasks include generating a repository, editing the hosts file, and creating a subdirectory. To fully configure the workstation, follow these steps.

1. Generate the chef-repo repository. This directory stores the Chef cookbooks and recipes. Enter yes when asked whether to accept the product licenses.

```
chef generate repo chef-repo
```

2. Edit the /etc/hosts file. This file contains mappings between host names and their IP addresses. Add an entry for the Chef Server, containing the name of the server, which is also the domain name, and its IP address. In this example, this is indicated in the line 192.0.1.0 example.com. There must also be an entry for the local server. This is the 192.0.2.0 chefworkstation line in the example. This entry must contain the local IP address and the hostname of the server hosting the Chef Workstation. The file should resemble the following example.

File: /etc/hosts

```
1 127.0.0.1 localhost
2 192.0.1.0 example.com
3 192.0.2.0 chefworkstation
```

3. Create a .chef subdirectory. This is where the knife file is stored, along with files for encryption and security.
4. mkdir ~/chef-repo/.chef

```
cd chef-repo
```

How to Add RSA Private Keys

RSA private keys enable better security between the Chef Server and associated workstations through the use of encryption. Earlier, RSA private keys were created on the Chef Server. Copying these keys to a workstation allows it to communicate with the server. To enable encryption using RSA private keys, follow these steps.

Note
SSH password authentication must be enabled on the Chef Server to complete the key exchange. If SSH password authentication has been disabled for better security, enable it again before proceeding. After the keys have been retrieved and added to the workstation, SSH password authentication can be disabled again. See the Linode guide to How to Secure Your Server for more information.

1. On the workstation, generate an RSA key pair. This key can be used to initially access the Chef server to copy over the private encryption files.

```
ssh-keygen -b 4096
```

2. Hit the **Enter** key to accept the default file names id_rsa and id_rsa.pub. Ubuntu stores these files in the /home/username/.ssh directory.
3. Enter a password when prompted, then enter it again. An identifier and public key are saved to the directory.

4. Copy the new public key from the workstation to the Chef Server. In the following command, use the account name for the Chef Server along with its IP address.

```
ssh-copy-id username@192.0.1.0
```

5. Use the scp command to copy the .pem files from the Chef Server to the workstation. In the following example, replace username with the user account for the Chef Server and 192.0.1.0 with the actual Chef Server IP address.

```
scp username@192.0.1.0:~/.chef/*.pem ~/chef-repo/.chef/
```

6. List the contents of the .chef subdirectory to ensure the .pem files were successfully copied.

```
ls ~/chef-repo/.chef
```

How to Configure Git on a Chef Workstation

A *version control system* helps the Chef Workstation track any changes to the cookbooks and restore earlier versions if necessary. This example uses Git, which is compatible with the Chef system. The following steps explain how to configure Git, initialize a Git repository, add new files, and commit them.

1. Configure Git using the git config command. Replace username and user@email.com with your own values.
2. git config --global user.name username

```
git config --global user.email user@email.com
```

3. Add the .chef directory to the .gitignore file. This ensures system and auto-generated files are not shown in the output of git status and other Git commands.

```
echo ".chef" > ~/chef-repo/.gitignore
```

4. Ensure the chef-repo directory is the current working directory. Add and commit the existing files using git add and git commit.
5. cd ~/chef-repo
6. git add .

```
git commit -m "initial commit"
```

7. Run the git status command to ensure all files have been committed.

```
git status
```

How to Generate a Chef Cookbook

To generate a new Chef cookbook, use the chef generate command.
chef generate cookbook my cookbook

How to Configure the Knife Utility

The Chef Knife utility helps a Chef workstation communicate with the server. It provides a method of managing cookbooks, nodes, and the Chef environment. Chef uses the config.rb file in the .chef subdirectory to store the Knife configuration. To configure Knife, follow these steps.

1. Create a config.rb file in the ~/chef-repo/.chef directory. This example uses vi, but any text editor can be used.
2. cd ~/chef-repo/.chef

```
vi config.rb
```

3. Use the following config.rb file as an example of how to configure Knife. Copy this sample configuration to the file.

File: ~/chef-repo/.chef/config.rb

```
current_dir = File.dirname(__FILE__)
log_level                :info
log_location             STDOUT
node_name                'node_name'
client_key               "USER.pem"
validation_client_name   'ORG_NAME-validator'
validation_key           "ORG_NAME-validator.pem"
chef_server_url          'https://example.com/organizations/ORG_NAME'
cache_type               'BasicFile'
cache_options( :path => "#{ENV['HOME']}/.chef/checksums" )
cookbook_path            ["#{current_dir}/../cookbooks"]
```

4. Make the following changes:

- node_name must be the name of the user account created when configuring the Chef Server.
- For client_key, replace USER with the user name associated with the .pem file, followed by .pem.
- validation_client_name requires the same ORG_NAME used when creating the organization followed by -validator.
- the validation_key field must contain the name used for ORG_NAME when the organization was created, followed by -validator.pem.
- For chef_server_url, change example.com to the name of the domain. Follow the domain name with /organizations/ and the ORG_NAME used when creating the organization.
- Leave the remaining fields unchanged.

5. Move back to the chef-repo directory and fetch the necessary SSL certificates from the server using the knife fetch command.

Note
The SSL certificates were generated when the Chef server was installed. The certificates are self-signed. This means a certificate authority has not verified them. Before fetching the certificates, log in to the Chef server and ensure the hostname and fully qualified domain name (FQDN) are the same. These values can be confirmed using the commands hostname and hostname -f.

```
cd ..
knife ssl fetch
```

6. To confirm the config.rb file is correct, run the knife client list command. The relevant validator name should be displayed.

```
knife client list
```

How to Bootstrap a Node

At this point, both the Chef Server and Chef Workstation are configured. They can now be used to bootstrap the node. The bootstrap process installs the chef client on the node and performs validation. The node can then retrieve any necessary updates from the Chef Server. To bootstrap the node, follow these steps.

1. Log in to the target node, which is the node to be bootstrapped, and edit the /etc/hosts file. Add entries for the node, the Chef server domain name, and the workstation. The file should resemble the following example, using the actual names of the Chef Server, workstation, and the target node, along with their IP addresses.

File: /etc/hosts

```
127.0.0.1 localhost
192.0.100.0 targetnode
192.0.2.0 chefworkstation
192.0.1.0 example.com
```

2. Return to the Linode hosting the Chef Workstation and change the working directory to ~/chef-repo/.chef. cd ~/chef-repo/.chef
3. Bootstrap the node using the knife bootstrap command. Specify the IP address of the target node for node_ip_address. This is the address of the node to bootstrap. In the following example, use the actual user name and password for the account in place of username and password. Enter the name of the node in place of nodename. Answer Y when asked “Are you sure you want to continue connecting”.

Note
The option to bootstrap using key-pair authentication no longer appears to be supported.
knife bootstrap node_ip_address -U username -P password --sudo --use-sudo-password --node-name nodename

4. Confirm the node has been successfully bootstrapped. List the client nodes using the knife client list command. All bootstrapped nodes should be listed.

knife client list

5. Add the bootstrapped node to the workstation /etc/hosts file as follows. Replace 192.0.100.0 targetnode with the IP address and name of the bootstrapped node.

File: /etc/hosts

```
127.0.0.1 localhost
192.0.1.0 example.com
192.0.2.0 chefworkstation
192.0.100.0 targetnode
```

How to Download and Apply a Cookbook (Optional)

A cookbook is the most efficient way of keeping target nodes up to date. In addition, a cookbook can delete the validation.pem file that was created on the node when it was bootstrapped. It is important to delete this file for security reasons.

It is not mandatory to download or create cookbooks to use Chef. But this section provides a brief example of how to download a cookbook and apply it to a node.

1. On the Chef workstation, change to the ~/chef-repo/.chef directory.

```
cd ~/chef-repo/.chef
```

2. Download the cron-delvalidate cookbook from the Chef Supermarket. For more information on the supermarket command see the Chef supermarket documentation.

```
knife supermarket download cron-delvalidate
```

3. If the cookbook is downloaded as a .tar.gz file, use the tar command to extract it. Move the extracted directory to the cookbooks directory.
4. tar -xf cron-delvalidate-0.1.3.tar.gz

```
cp -r  cron-delvalidate ~/chef-repo/cookbooks/
```

5. Review the cookbook's default.rb file to see the recipe. This recipe is written in Ruby and demonstrates how a typical recipe is structured. It contains a cron job named clientrun. This job instantiates a new cron job to run the chef-client command on an hourly basis. It also removes the extraneous validation.pem file.

File: ~/chef-repo/cookbooks/cron-delvalidate/recipes/default.rb

```
1   #
2   # Cookbook Name:: cron-delvalidate
3   # Recipe:: Chef-Client Cron & Delete Validation.pem
4   #
5   #
6   cron "clientrun" do
7     minute '0'
8     hour '*/1'
9     command "/usr/bin/chef-client"
10    action :create
11  end
12
13  file "/etc/chef/validation.pem" do
14    action :delete
15  end
```

6. Add the recipe to the run list for the node. In the following command, replace nodename with the name of the node.

```
knife node run_list add nodename 'recipe[cron-delvalidate::default]'
```

7. Upload the cookbook and its recipes to the Chef Server.

```
knife cookbook upload cron-delvalidate
```

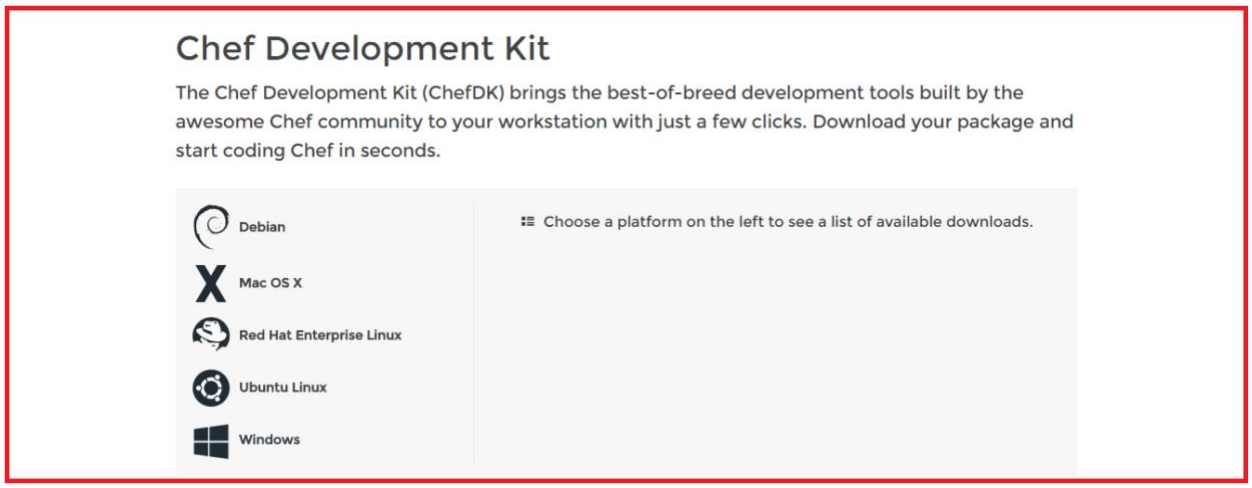
8. Run the chef-client command on the node using the knife ssh utility. This command causes the node to pull the recipes in its run list from the server. It also determines whether there are any updates. The Chef Server transmits the recipes to the target node. When the recipe runs, it deletes the file and installs a cron job to keep the node up to date in the future. In the following command, replace nodename with the actual name of the target node. Replace username with the name of a user account with sudo access. Enter the password for the account when prompted to do so.

```
knife ssh 'name:nodename' 'sudo chef-client' -x username
```

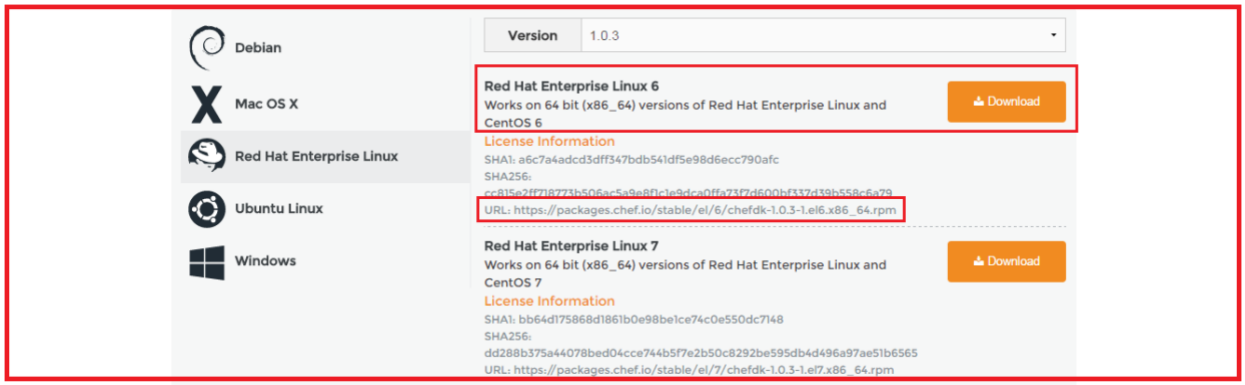
Demo:

Step 1: Install Chef DK (Development Kit) in your Chef Workstation.

Chef DK is a package that contains all the development tools that you will need when coding Chef. Here is the link to download ***Chef DK***.



Here, choose the operating system that you are using. I am using CentOS 6.8 .So, I will click on **Red Hat Enterprise Linux**.



Copy the link according to the version of CentOS that you are using. I am using CentOS 6, as you can see that I have highlighted in the above screenshot.

Go to your Workstation terminal and download the Chef DK by using wget command and paste the link.

Execute this:

1wget https://packages.chef.io/stable/el/6/chefdk-1.0.3-1.el6.x86_64.rpm

```
[root@Workstation ~]# wget https://packages.chef.io/stable/el/6/chefdk-1.0.3-1.el6.x86_64.rpm
--2016-11-25 07:40:38--  https://packages.chef.io/stable/el/6/chefdk-1.0.3-1.el6.x86_64.rpm
Resolving packages.chef.io... 151.101.8.65
Connecting to packages.chef.io|151.101.8.65|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://packages.chef.io/files/stable/chefdk/1.0.3/el/6/chefdk-1.0.3-1.el6.x86_64.rpm [following]
--2016-11-25 07:40:44--  https://packages.chef.io/files/stable/chefdk/1.0.3/el/6/chefdk-1.0.3-1.el6.x86_64.rpm
Reusing existing connection to packages.chef.io:443.
HTTP request sent, awaiting response... 200 OK
Length: 105276444 (100M) [application/x-rpm]
Saving to: “chefdk-1.0.3-1.el6.x86_64.rpm”

100%[======================================================================>] 105,276,444  300K/s   in 3m 19s

2016-11-25 07:44:15 (517 KB/s) - “chefdk-1.0.3-1.el6.x86_64.rpm” saved [105276444/105276444]
```

The package is now downloaded. It is time to install this package using rpm.

Execute this:

1rpm -ivh chefdk-1.0.3-1.el6.x86_64.rpm

```
[root@Workstation ~]# rpm -ivh chefdk-1.0.3-1.el6.x86_64.rpm
warning: chefdk-1.0.3-1.el6.x86_64.rpm: Header V4 DSA/SHA1 Signature, key ID 83ef826a: NOKEY
Preparing...                ########################################### [100%]
   1:chefdk                 ########################################### [100%]
Thank you for installing Chef Development Kit!
[root@Workstation ~]#
```

Chef DK is now installed in my Workstation.

Step 2: Create a Recipe in the Workstation

Let us start by creating a Recipe in the Workstation and test it locally to ensure it is working. Create a folder named chef-repo. We can create our Recipes inside this folder.

Execute this:

1mkdir chef-repo

2cd chef-repo

```
[root@Workstation ~]# mkdir chef-repo
[root@Workstation ~]# cd chef-repo/
[root@Workstation chef-repo]#
```

In this chef-repo directory I will create a Recipe named edureka.rb. .rb is the extension used for ruby. I will use vim editor, you can use any other editor that you want like gedit, emac, vi etc.

Execute this:

1vim edureka.rb

Here add the following:

1file '/etc/motd'

2content 'Welcome to Chef'

3end

```
file '/etc/motd' do
content 'Welcome to Chef'
end
```

This Recipe **edureka.rb** creates a file named /etc/motd with content “Welcome to Chef”.

Now I will use this Recipe to check if it is working.

Execute this:

1chef-apply edureka.rb

```
[root@Workstation chef-repo]# chef-apply edureka.rb
Recipe: (chef-apply cookbook)::(chef-apply recipe)
  * file[/etc/motd] action create
    - update content in file /etc/motd from e3b0c4 to 40a30c
    --- /etc/motd       2010-01-12 13:28:22.000000000 +0000
    +++ /etc/.chef-motd20161125-5713-ba857q     2016-11-25 09:00:05.262933936 +0000
    @@ -1 +1,2 @@
    +Welcome to Chef
    - restore selinux security context
[root@Workstation chef-repo]#
```

So there is a file created in the chef-repo that has content **Welcome to Chef.**

Step 3: Modifying Recipe file to install httpd package

I will modify the Recipe to install httpd package on my Workstation and copy an index.html file to the default document root to confirm the installation. The default action for a package resource is installation, hence I don’t need to specify that action separately.

Execute this:

1vim edureka.rb

Over here add the following:

1package 'httpd'

2service 'httpd' do

3action [:enable, :start]

4end

5file '/var/www/html/index.html' do

6content 'Welcome to Apache in Chef'

7end

```
package 'httpd'
service 'httpd' do
action [:enable, :start]
end

file '/var/www/html/index.html' do
content 'Welcome to Apache in Chef'
end
```

Now I will apply these configurations by executing the below command:

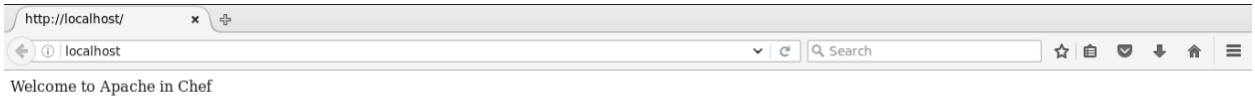
Execute this:

1chef-apply edureka.rb

```
[root@Workstation chef-repo]# chef-apply edureka.rb
Recipe: (chef-apply cookbook)::(chef-apply recipe)
  * yum_package[httpd] action install (up to date)
  * service[httpd] action enable
    - enable service service[httpd]
  * service[httpd] action start
    - start service service[httpd]
  * file[/var/www/html/index.html] action create
    - create new file /var/www/html/index.html
    - update content in file /var/www/html/index.html from none to 152204
    --- /var/www/html/index.html        2016-11-25 10:02:52.399858608 +0000
    +++ /var/www/html/.chef-index20161125-6176-1w10rp.html      2016-11-25 10:02:52.399858608 +0000
    @@ -1 +1,2 @@
    +Welcome to Apache in Chef
    - restore selinux security context
[root@Workstation chef-repo]#
```

The command execution clearly describes each instance in the Recipe. It installs the Apache package, enables and starts the httpd service on the Workstation. And it creates an index.html file in the default document root with the content “Welcome to Apache in Chef”.

Now confirm the installation of Apache2 by opening your web-browser. Type your public IP address or the name of your host. In my case, it is localhost.



Step 4: Now we will create our first Cookbook.

Create a directory called cookbooks, and execute the below command to generate the Cookbook.

Execute this:

1mkdir cookbooks

2cd cookbooks

3chef generate cookbook httpd_deploy

httpd_deploy is a name given to the Cookbook. You can give any name that you want.

```
[root@Workstation chef-repo]# mkdir cookbooks
[root@Workstation chef-repo]# cd cookbooks/
[root@Workstation cookbooks]# chef generate cookbook httpd_deploy
Generating cookbook httpd_deploy
- Ensuring correct cookbook file content
- Ensuring delivery configuration
- Ensuring correct delivery build cookbook content

Your cookbook is ready. Type `cd httpd_deploy` to enter it.

There are several commands you can run to get started locally developing and testing your cookbook.
Type `delivery local --help` to see a full list.

Why not start by writing a test? Tests for the default recipe are stored at:

test/recipes/default_test.rb

If you'd prefer to dive right in, the default recipe can be found at:

recipes/default.rb

[root@Workstation cookbooks]#
```

Let us move to this new directory httpd_deploy.

Execute this:

1cd httpd_deploy

Now let us see the file structure of the created Cookbook.

Execute this:

1tree

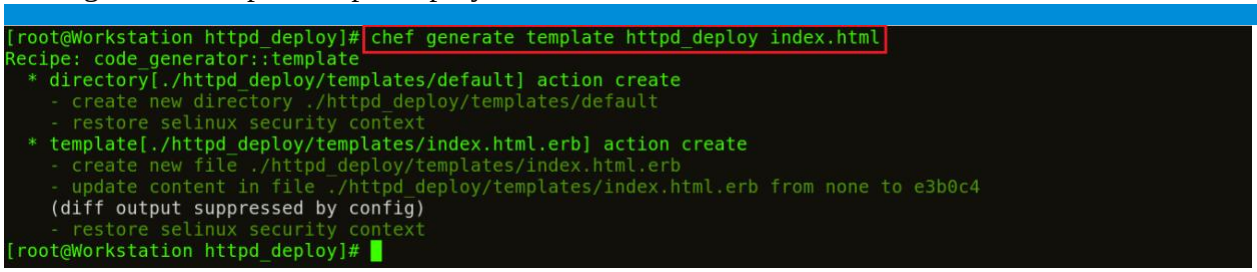
```
[root@Workstation httpd_deploy]# tree
.
├── Berksfile
├── chefignore
├── metadata.rb
├── README.md
├── recipes
│   └── default.rb
├── spec
│   ├── spec_helper.rb
│   └── unit
│       └── recipes
│           └── default_spec.rb
└── test
    └── recipes
        └── default_test.rb

6 directories, 8 files
[root@Workstation httpd_deploy]#
```

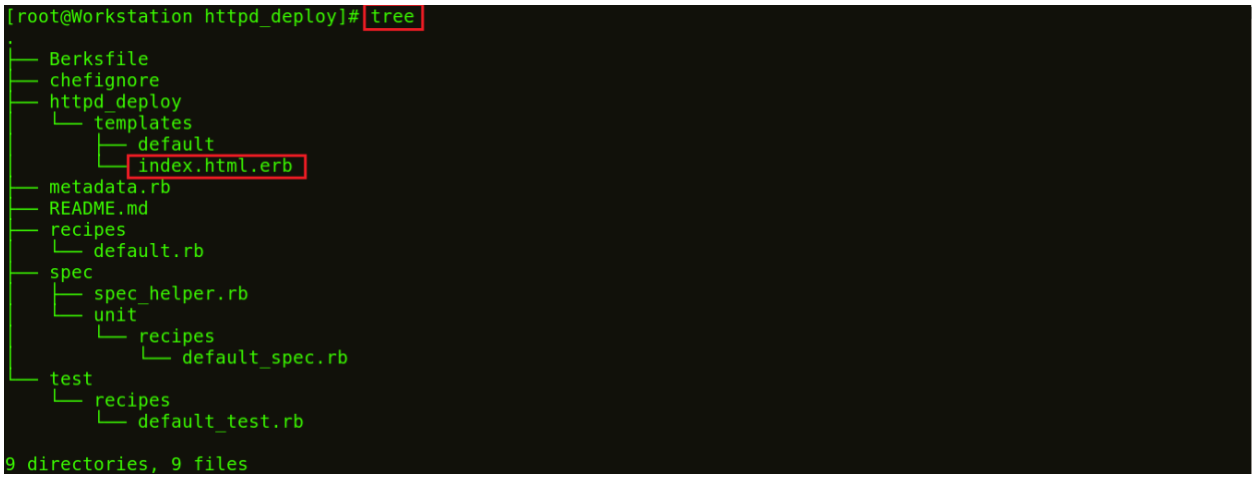
Step 5: Create a Template file.

Earlier, I created a file with some contents, but that can't fit with my Recipes and Cookbook structures. So let us see how we can create a Template for index.html page.

Execute this:

```
1chef generate template httpd_deploy index.html
```





Now if you see my Cookbook file structure, there is a folder created with the name templates with index.html.erb file. I will edit this index.html.erb template file and add my Recipe to it. Refer the example below:

Go to the default directory

Execute this:

```
1cd /root/chef-repo/cookbook/httpd_deploy/templates/default
```

Over here, edit the index.html.erb template by using any editor that you are comfortable with. I will use vim editor.

Execute this:

```
1vim index.html.erb
```

Now add the following:

```
1Welcome to Chef Apache Deployment
```

Step 6: Create a Recipe with this template.

Go to the Recipes directory.

Execute this:

```
1cd /root/chef-repo/cookbooks/httpd_deploy/recipes
```

Now edit the default.rb file by using any editor that you want. I will use vim editor.

Execute this:

```
1vim default.rb
```

Over here add the following:

```
1package 'httpd'
2service 'httpd' do
3action [:enable, :start]
4end
5template '/var/www/html/index.html' do
6source 'index.html.erb'
7end
```

```
#
# Cookbook Name:: httpd_deploy
# Recipe:: default
#
# Copyright (c) 2016 The Authors, All Rights Reserved.
package 'httpd'
service 'httpd' do
action [:enable, :start]
end

template '/var/www/html/index.html' do
source 'index.html.erb'
end
~
~
```

Now I will go back to my chef-repo folder and run/test my recipe on my Workstation.

Execute this:

```
1cd /root/chef-repo
```

2chef-client --local-mode --runlist 'recipe[httpd_deploy]'

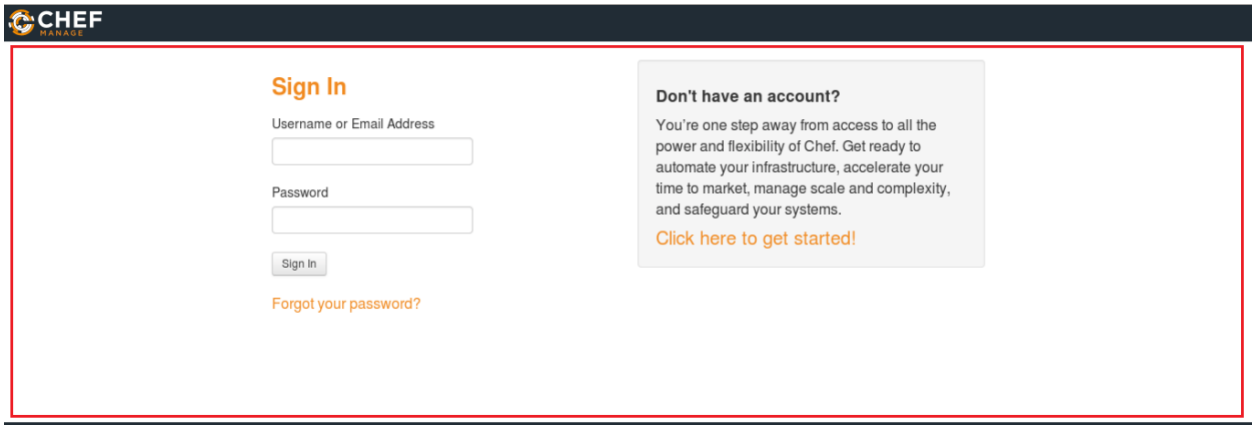
```
[root@Workstation chef-repo]# chef-client --local-mode --runlist 'recipe[httpd_deploy]'
[2016-11-28T06:05:31+00:00] WARN: No config file found or specified on command line, using command line options.
Starting Chef Client, version 12.16.42
resolving cookbooks for run list: ["httpd_deploy"]
Synchronizing Cookbooks:
  - httpd_deploy (0.1.0)
Installing Cookbook Gems:
Compiling Cookbooks...
Converging 3 resources
Recipe: httpd_deploy::default
  * yum_package[httpd] action install (up to date)
  * service[httpd] action enable (up to date)
  * service[httpd] action start (up to date)
  * template[/var/www/html/index.html] action create
    - update content in file /var/www/html/index.html from 152204 to 748cbd
    --- /var/www/html/index.html        2016-11-25 10:02:52.399858608 +0000
    +++ /var/www/html/.chef-index20161128-22551-eiujfo.html     2016-11-28 06:05:51.806388896 +0000
    @@ -1,2 +1,2 @@
    -Welcome to Apache in Chef
    +Welcome to Chef Apache Deployment
```

According to my Recipe, Apache is installed on my Workstation, service is being started and enabled on boot. Also a template file has been created on my default document root.

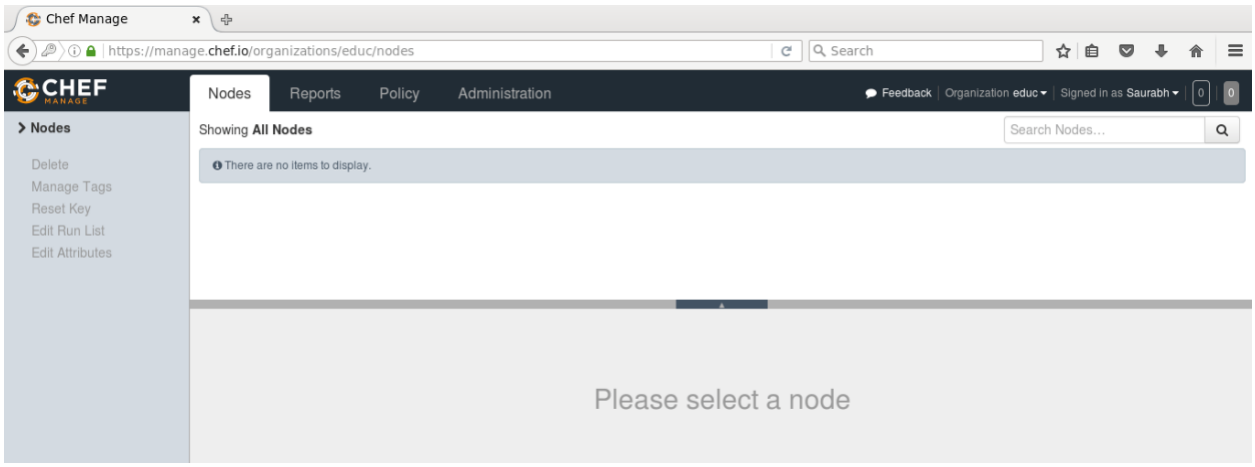
Now that I have tested my Workstation. It's time to setup the Chef Server.

Step 7: Setup Chef Server

I will use the hosted version of Chef Server on the cloud but you can use a physical machine as well. This Chef-Server is present at ***manage.chef.io***



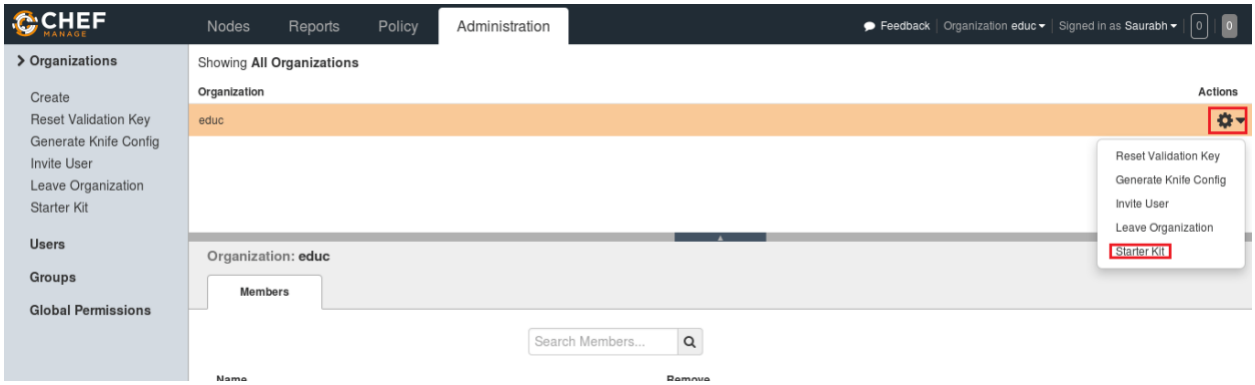
Over here create an account if you don't have one. Once you have created an account, sign-in with your login credentials.



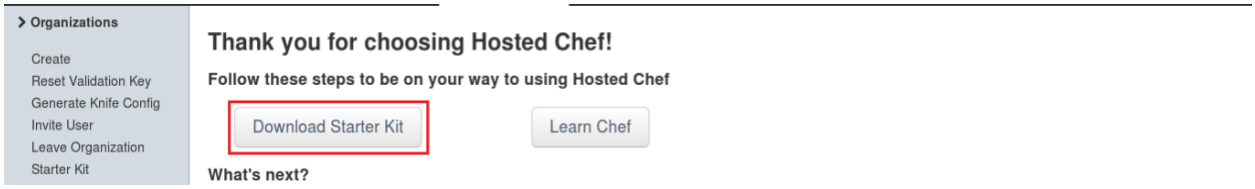
This is how Chef Server looks like.

If you are signing in for the first time, the very first thing that you will be doing is creating an organization. Organization is basically a group of Machines that you will be managing with the Chef Server.

First, I will go to the administration tab. Over there, I have already created an organization called edu. So I need to download the starter kit in my Workstation. This starter kit will help you to push files from the Workstation to the Chef Server. Click on the settings icon on the right hand side and click on Starter Kit.



When you click over there you will get an option to download the Starter Kit. Just click on it to download the Starter Kit zip file.



Move this file to your root directory. Now unzip this zip file by using unzip command in your terminal. You will notice that it includes a directory called chef-repo.

Execute this:

1unzip chef-starter.zip

```
[root@Workstation ~]# unzip chef-starter.zip
Archive:  chef-starter.zip
  inflating: chef-repo/cookbooks/chefignore
   creating: chef-repo/cookbooks/starter/
   creating: chef-repo/cookbooks/starter/files/
   creating: chef-repo/cookbooks/starter/files/default/
  inflating: chef-repo/cookbooks/starter/files/default/sample.txt
  inflating: chef-repo/cookbooks/starter/metadata.rb
   creating: chef-repo/cookbooks/starter/attributes/
  inflating: chef-repo/cookbooks/starter/attributes/default.rb
   creating: chef-repo/cookbooks/starter/templates/
   creating: chef-repo/cookbooks/starter/templates/default/
  inflating: chef-repo/cookbooks/starter/templates/default/sample.erb
   creating: chef-repo/cookbooks/starter/recipes/
  inflating: chef-repo/cookbooks/starter/recipes/default.rb
  inflating: chef-repo/README.md
  inflating: chef-repo/.gitignore
   creating: chef-repo/.chef/
   creating: chef-repo/roles/
  inflating: chef-repo/.chef/knife.rb
  inflating: chef-repo/roles/starter.rb
  inflating: chef-repo/.chef/saurabh010.pem
```

Now move this starter kit to the cookbook directory in chef-repo directory.

Execute this:

1mv starter /root/chef-repo/cookbook

Chef Cookbooks are available in the Cookbook Super Market, we can go to the Chef SuperMarket. Download the required Cookbooks from ***supermarket.chef.io***. I'm downloading one of the Cookbook to install Apache from there.

Execute this:

1cd chef-repo

2knife cookbook site download learn_chef_httpd

There is Tar ball downloaded for the Apache Cookbook. Now, we need to extract the contents from this downloaded Tar file. For that, I will use tar command.

1tar -xvf learn_chef_httpd-0.2.0.tar.gz

```
[root@Workstation chef-repo]# tar -xvf learn_chef_httpd-0.2.0.tar.gz
learn_chef_httpd/
learn_chef_httpd/.kitchen.yml
learn_chef_httpd/Berksfile
learn_chef_httpd/chefignore
learn_chef_httpd/metadata.json
learn_chef_httpd/metadata.rb
learn_chef_httpd/README.md
learn_chef_httpd/recipes/
learn_chef_httpd/templates/
learn_chef_httpd/templates/default/
learn_chef_httpd/templates/default/index.html.erb
learn_chef_httpd/recipes/default.rb
[root@Workstation chef-repo]#
```

All the required files are automatically created under this Cookbook. There is no need to make any modifications. Let's check the Recipe description inside my recipes folder.

Execute this:

1cd /root/chef-repo/learn_chef_httpd/recipes

2cat default.rb

```
[root@Workstation recipes]# cat default.rb
#
# Cookbook Name:: learn_chef_httpd
# Recipe:: default
#
# Copyright (C) 2014
#
#
#
package 'httpd'

service 'httpd' do
  action [:enable, :start]
end

template '/var/www/html/index.html' do
  source 'index.html.erb'
end

service 'iptables' do
  action :stop
end
[root@Workstation recipes]#
```

Now, I will just upload this cookbook to my Chef Server as it looks perfect to me.

Step 8: Upload Cookbook to the Chef Server.

In order to upload the Apache Cookbook that I have downloaded, first move this learn_chef_httpd file to the Cookbooks folder in the chef-repo. Then change your directory to cookbooks.

Execute this:

1mv /root/chef-repo/learn_chef_httpd /root/chef-repo/cookbooks

Now move to this cookbooks directory.

Execute this:

1cd cookbooks

Now in this directory, execute the below command to upload the Apache Cookbook:

Execute this:

1knife cookbook upload learn_chef_httpd

```
[root@Workstation cookbooks]# knife cookbook upload learn_chef_httpd
Uploading learn_chef_httpd [0.2.0]
Uploaded 1 cookbook.
[root@Workstation cookbooks]#
```

Verify the Cookbook from the Chef Server Management console. In the policy section, you will find the Cookbook that you have uploaded. Refer the screenshot below:

Now our final step is to add Chef Node. I have setup a Workstation, a Chef Server and now I need to add my Clients to the Chef Server for automation.

Step 9: Adding Chef Node to the Chef Server.

For demonstration purpose I will use one CentOS machine as Chef Node. There can be hundreds of Nodes connected to one Chef Server. The terminal color of my Node machine is different from the Workstation so that you will be able to differentiate between both.

I just need the IP address of my Node for that I will execute the below command in my Node machine.

Execute this:

1ifconfig

```
[root@ChefNode ~]# ifconfig
eth1      Link encap:Ethernet  HWaddr 08:00:27:BA:23:1B
          inet addr:10.0.2.16  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::a00:27ff:feba:231b/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:62 errors:0 dropped:0 overruns:0 frame:0
          TX packets:66 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:7822 (7.6 KiB)  TX bytes:5885 (5.7 KiB)

eth2      Link encap:Ethernet  HWaddr 08:00:27:85:36:02
          inet addr:192.168.56.102  Bcast:192.168.56.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe85:3602/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1909 errors:0 dropped:0 overruns:0 frame:0
          TX packets:34 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:254899 (248.9 KiB)  TX bytes:8260 (8.0 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
```

I will add my Chef Node to the Server by executing Knife Bootstrap command in which I will specify the IP address of The Chef Node and its name. Execute the command shown below:

Execute this:

1knife bootstrap 192.168.56.102 --ssh-user root --ssh-password edureka --node-name chefNode

```
[root@Workstation cookbooks]# knife bootstrap 192.168.56.102 --ssh-user root --ssh-password edureka --node-name
chefNode
Node chefNode exists, overwrite it? (Y/N) y
Client chefNode exists, overwrite it? (Y/N) y
Creating new client for chefNode
Creating new node for chefNode
Connecting to 192.168.56.102
192.168.56.102 -----> Installing Chef Omnibus (-v 12)
192.168.56.102 downloading https://omnitruck-direct.chef.io/chef/install.sh
192.168.56.102   to file /tmp/install.sh.5743/install.sh
192.168.56.102 trying wget...
192.168.56.102 el 6 x86_64
192.168.56.102 Getting information for chef stable 12 for el...
192.168.56.102 downloading https://omnitruck-direct.chef.io/stable/chef/metadata?v=12&p=el&pv=6&m=x86_64
192.168.56.102   to file /tmp/install.sh.5754/metadata.txt
192.168.56.102 trying wget...
192.168.56.102 sha1      abe23132483c1ff015148ad72becf092c0fac428
192.168.56.102 sha256    bca097d9107bd4b20df88045b676a1d032b68a8b66bcf751dabc2e58715ae0ae
192.168.56.102 url       https://packages.chef.io/files/stable/chef/12.16.42/el/6/chef-12.16.42-1.el6.x86_64.rpm
192.168.56.102 version   12.16.42
192.168.56.102 downloaded metadata file looks valid...
192.168.56.102 downloading https://packages.chef.io/files/stable/chef/12.16.42/el/6/chef-12.16.42-1.el6.x86_64.r
```

```
192.168.56.102   to file /tmp/install.sh.5754/chef-12.16.42-1.el6.x86_64.rpm
192.168.56.102 trying wget...
192.168.56.102 Comparing checksum with sha256sum...
192.168.56.102 Installing chef 12
192.168.56.102 installing with rpm...
192.168.56.102 warning: /tmp/install.sh.5754/chef-12.16.42-1.el6.x86_64.rpm: Header V4 DSA/SHA1 Signature, key I
D 83ef826a: NOKEY
192.168.56.102 Preparing...                ########################################### [100%]
192.168.56.102    1:chef                   ########################################### [100%]
192.168.56.102 Thank you for installing Chef!
192.168.56.102 Starting the first Chef Client run...
192.168.56.102 Starting Chef Client, version 12.16.42
192.168.56.102 resolving cookbooks for run list: []
192.168.56.102 Synchronizing Cookbooks:
192.168.56.102 Installing Cookbook Gems:
192.168.56.102 Compiling Cookbooks...
192.168.56.102 [2016-11-28T11:13:27+00:00] WARN: Node chefNode has an empty run list.
192.168.56.102 Converging 0 resources
192.168.56.102
192.168.56.102 Running handlers:
192.168.56.102 Running handlers complete
192.168.56.102 Chef Client finished, 0/0 resources updated in 49 seconds
```

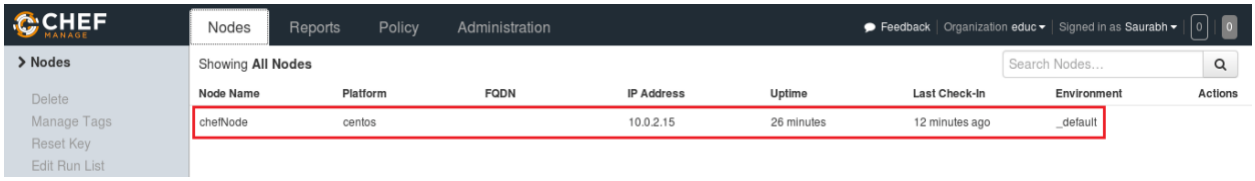
This command will also initialize the installation of the Chef-Client in the Chef Node. You can verify it from the CLI on the Workstation using the knife command,as shown below:

Execute this:

1Knife node list

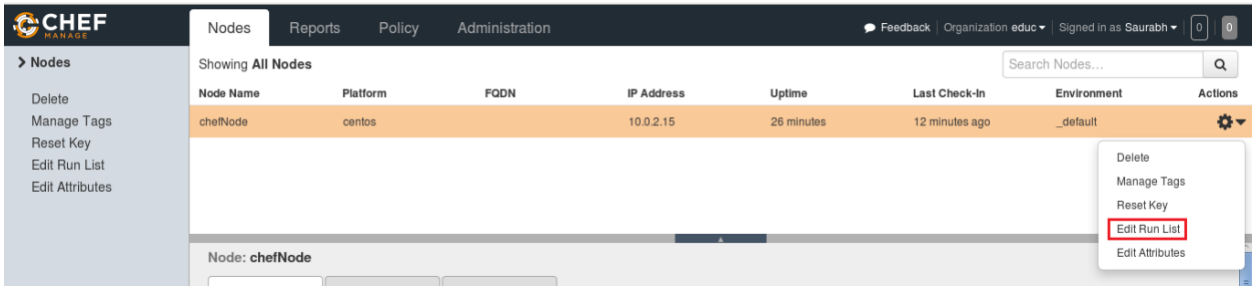
```
[root@Workstation cookbooks]# knife node list
chefNode
[root@Workstation cookbooks]#
```

You can also verify from the Chef Server. Go to the nodes tab in your Server Management Console, here you will notice that the node that you have added is present. Refer the screenshot below.

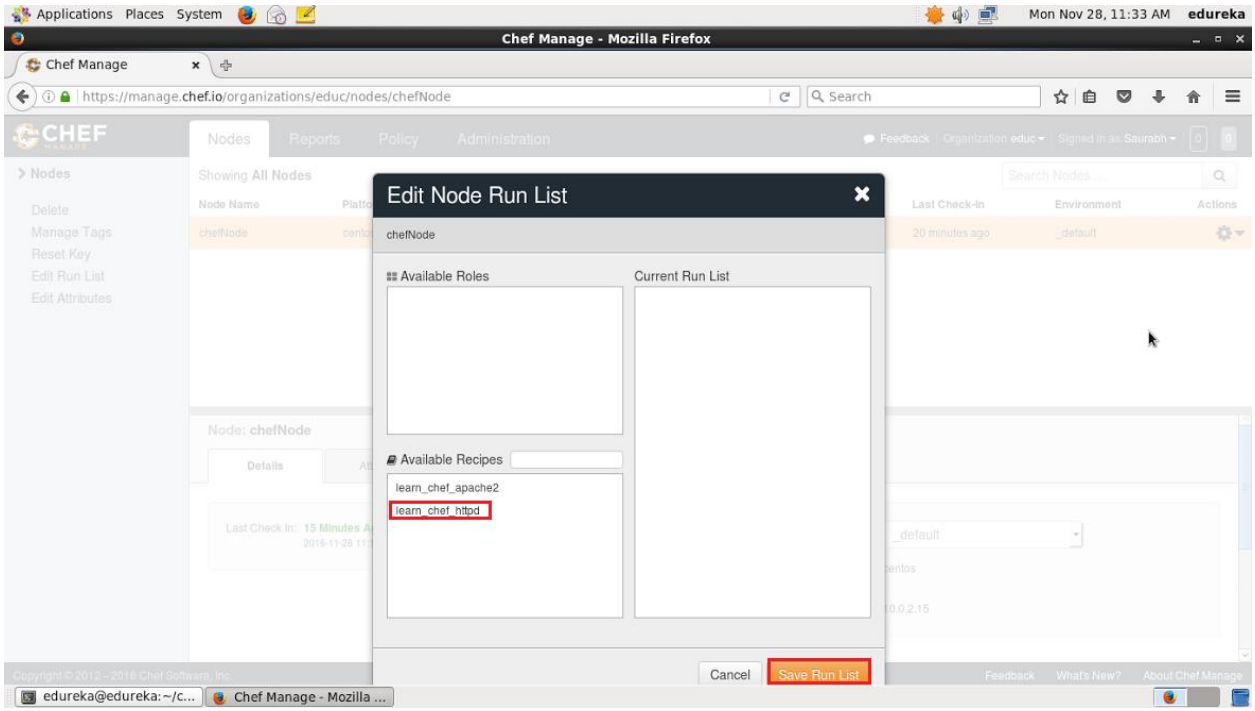


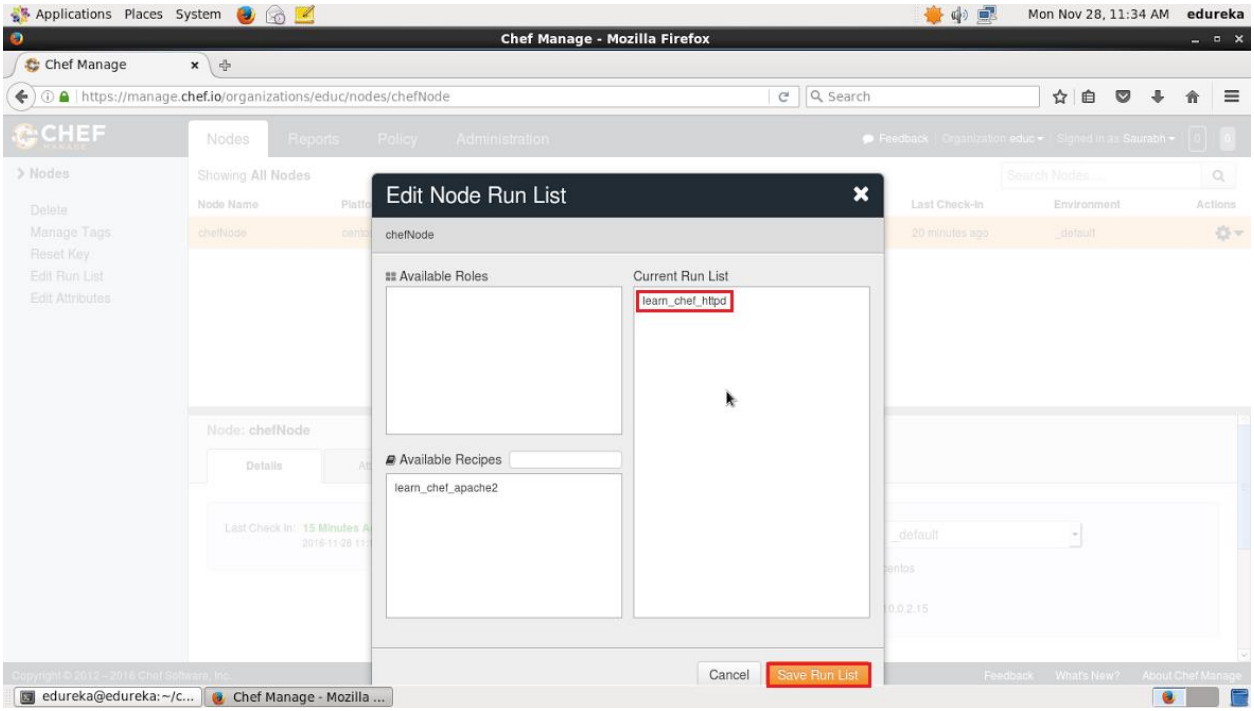
Step 10: Manage Node Run List

Let's see how we can add a Cookbook to the Node and manage its Run list from the Chef Server. As you can see in the screenshot below, click the Actions tab and select the Edit Run list option to manage the Run list.



In the Available Recipes, you can see our learn_chef_httpd Recipe, you can drag that from the available packages to the current Run List and save the Run list.





Now login to your Node and just run chef-client to execute the Run List.

Execute this:

1chef-client

```
[root@ChefNode ~]# chef-client
Starting Chef Client, version 12.16.42
resolving cookbooks for run list: ["learn_chef_httpd"]
Synchronizing Cookbooks:
  - learn_chef_httpd (0.2.0)
Installing Cookbook Gems:
Compiling Cookbooks...
Converging 4 resources
Recipe: learn_chef_httpd::default
  * yum_package[httpd] action install (up to date)
  * service[httpd] action enable
    - enable service service[httpd]
  * service[httpd] action start
    - start service service[httpd]
  * template[/var/www/html/index.html] action create
    - create new file /var/www/html/index.html
    - update content in file /var/www/html/index.html from none to ef4ffd
    --- /var/www/html/index.html        2016-11-28 11:47:17.414304893 +0000
    +++ /var/www/html/.chef-index20161128-6284-hzok74.html      2016-11-28 11:47:17.414304893 +0000
    @@ -1 +1,6 @@
    +<html>
```

Conclusion:

In conclusion, Chef is a powerful configuration management tool that automates the provisioning, configuration, and management of infrastructure. It enables the definition of infrastructure as code through recipes and cookbooks, simplifying the management of complex systems. With its centralized Chef server, Chef ensures consistency and reliability across different nodes, while its resource abstraction and data-gathering tool, Ohai, contribute to maintaining desired configurations and providing comprehensive insights. Chef's role in the DevOps toolchain is pivotal, facilitating efficient and reliable management of IT infrastructure at scale.

References:

- **https://www.edureka.co/blog/chef-tutorial/**
- **https://www.linode.com/docs/guides/how-to-install-chef-on-ubuntu-20-04/**