**Jenkins:**

A shell script is a text file containing a series of commands that are executed by a Unix/Linux shell (such as Bash). It is used to automate repetitive tasks, perform batch processing, and manage system operations.

EC2—ubuntu—inbound—custom TCP with port 8080 anywhere.

Create the script file:

vi Jenkins\_installation.sh

Paste the script content and save the file.

Make the script executable:

chmod +x Jenkins\_installation.sh

Run the script:

./Jenkins\_installation.sh

---------------------------------------------- Notes --------------------------------------------------------

The chmod command in Unix/Linux is used to change the permissions of a file or directory. The x permission is the execute permission. Here's a detailed explanation of its use and implications:

### Understanding File Permissions.

Each file or directory in Unix/Linux has three types of permissions:

* **Read (r):** Allows the reading of the file or listing of the directory's contents.
* **Write (w):** Allows writing to the file or making changes to the directory's contents (like creating or deleting files).
* **Execute (x):** Allows executing the file if it is a script or program, or accessing the directory and its contents.

**Making a File Executable:** chmod +x filename

**Removing Execute Permission:** chmod -x filename

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

systemctl status jenkins

copy public ip and search in google with port.

<http://100.26.197.117:8080/>

EC2—INBOUND RULES—add – custom tcp with port number 8080 from anywhere.

## **Jenkins Installation**

### **Master node configuration**

------------------------------------------install jenkins in linux----------------------------------------------

sudo apt update -y

sudo apt install openjdk-17-jre -y

java --version

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

/usr/share/keyrings/jenkins-keyring.asc > /dev/null

echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \

https://pkg.jenkins.io/debian-stable binary/ | sudo tee \

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

sudo apt-get update -y

sudo apt-get install jenkins -y

sudo systemctl daemon-reload

sudo systemctl enable jenkins

sudo systemctl start jenkins

sudo systemctl status jenkins

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

## **Initial admin password Location:**

/var/lib/jenkins/secrets/initialAdminPassword

Create job—free style proj—buid steps-- Execute shell—echo “hello”—save—build now--#1—console output.

Git clone:

URL for git REPO: <https://github.com/Bhavesh-55/proj_devops.git>

Job—name—save—config—source code management—git—url—branch=main—save—buildnow—console output.

UBUNTU—cd /var/lib/jenkins/workspace/<job name>

**BUILD TRIGGER**

Configure—Build trigger.

It is a mechanism that initiates the execution of a build for a project.

1. **Build Periodically**:
   * Schedule a job to run at regular intervals using a cron-like syntax.
   * Example syntax: H/5 \* \* \* \* (runs every 5 minutes).
   * <https://crontab.guru/>
2. **Poll SCM**:
   * Check the source code repository for changes at regular intervals and trigger a build if changes are found.
   * Example syntax: H/5 \* \* \* \* (polls(check) the SCM every 5 minutes).
3. **GitHub Hook Trigger for GITScm Polling**:
   * Automatically trigger a build when changes are pushed to a GitHub repository.
   * Requires configuring the GitHub repository to send webhooks to the Jenkins server.
   * **In GitHub:**
   * Go to your repository's settings.
   * Under Webhooks, click Add webhook.
   * Set the Payload URL to http://your-jenkins-server/github-webhook/.
   * Choose application/json as the content type.
   * Leave the Secret field empty unless you have configured a secret token in Jenkins.
   * Select Just the push event--Click Add webhook.

**Build Environment Options:**

1. **Delete workspace before build starts**:
   * **Purpose**: Ensures a clean workspace for each build by deleting any files from previous builds.
2. **Use secret text(s) or file(s)**:
   * **Purpose**: Allows the injection of secret texts or files into the build environment, useful for handling sensitive information like API keys or credentials.
3. **Add timestamps to the Console Output**:
   * **Purpose**: Adds timestamps to each line of the console output, which can help in debugging and performance analysis.
4. **Inspect build log for published build scans**:
   * **Purpose**: Useful for integration with tools like Gradle that provide build scans, which offer detailed insights into the build process.
     + .
5. **Terminate a build if it's stuck**:
   * **Purpose**: Automatically terminates a build if it gets stuck, preventing resource wastage and build queue blockage.

**Example Configuration:**

1. **Delete workspace before build starts**:
   * Check the box for a clean start.
2. **Use secret text(s) or file(s)**:
   * Check the box and click **Add** to configure secrets, e.g., API keys or SSH credentials.
3. **Add timestamps to the Console Output**:
   * Check the box for detailed logging.
4. **Inspect build log for published build scans**:
   * Check the box if you are using tools that provide build scans.
5. **Terminate a build if it's stuck**:
   * Check the box and set a reasonable timeout, e.g., 30 minutes.

**Jenkins Architecture in Simple Terms:**

**1. Jenkins Master**

* **Role**: The main control unit.
* **Responsibilities**:
  + Manages configurations and schedules jobs.
  + Provides the user interface for managing Jenkins and viewing job results.
  + Distributes tasks to agents (slaves).

**2. Jenkins Slaves (Agents)**

* **Role**: The workers.
* **Responsibilities**:
  + Execute the tasks (jobs) assigned by the master.
  + Can be any machine (physical, virtual, or container) that connects to the master.

**Example Workflow:**

1. **Developer commits code** to a repository (e.g., Git).
2. **Jenkins detects the change** and triggers a job.
3. The **master assigns the job** to an available agent.
4. The **agent performs the job** (e.g., builds the code, runs tests).
5. The **agent reports the results** back to the master.
6. The **master shows the results** in the Jenkins dashboard.

In essence, the master is the brains of Jenkins, coordinating everything, while the agents do the heavy lifting by performing the tasks.

My notes:

Launch 2 EC2 instances. --> Jenkins\_master(install Jenkins and connect with IP:8080) and Jenkins\_slave(No need to install Jenkins. Need to install jave).

**Jenkins\_slave** EC2(ubuntu) with pem key, Allow SSH traffic—sudo su—

apt update – y

apt install openjdk-17-jre -y

mkdir Jenkins

manage jenkins—Nodes—create new node--Number of executors(job): 5 --Remote root directory—check pwd and copy the path--/home/ubuntu/jenkins—Labels--Launch method—SSH—HOST—private ip of slave--172.31.47.86—Credentials—add—select ssh—user name=ubuntu---private key(pem)—add—select credential which has been created--Verification Strategy—non verify—save.

NOT ABLE TO CONNECT. NEED TO B CHECKED.

sudo mkdir -p /home/ubuntu/jenkins

sudo chown ubuntu:ubuntu /home/ubuntu/jenkins

sudo chmod 755 /home/ubuntu/jenkins

ls -ld /home/ubuntu/jenkins

--------------------------------------------------------------Note----------------------------------------------------

**Understanding chmod 755.**

1. **First digit (7)**: Permissions for the **owner** (user who owns the file or directory).
   * 7 means the owner has read (r), write (w), and execute (x) permissions.
   * rwx = 4 (read) + 2 (write) + 1 (execute) = 7
2. **Second digit (5)**: Permissions for the **group** (users who are in the same group as the file or directory).
   * 5 means the group has read (r) and execute (x) permissions, but not write permission.
   * r-x = 4 (read) + 0 (write) + 1 (execute) = 5
3. **Third digit (5)**: Permissions for **others** (all other users).
   * 5 means others have read (r) and execute (x) permissions, but not write permission.
   * r-x = 4 (read) + 0 (write) + 1 (execute) = 5

**Permissions Breakdown:**

* r (read) = 4
* w (write) = 2
* x (execute) = 1

goto job—giturl--select slave server lable—select “restrict where job proj can be run” .

Build: (source code🡪executable format)

In Jenkins, the build phase is where your code is compiled, tested, and packaged.

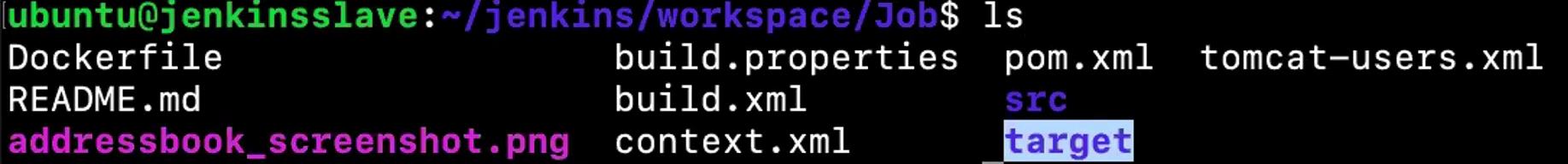
1. **GitHub Hook Trigger for GITScm Polling**:
   * Automatically trigger a build when changes are pushed to a GitHub repository.
   * Requires configuring the GitHub repository to send webhooks to the Jenkins server.
   * **In GitHub:**
   * Go to your repository's settings.
   * Under Webhooks, click Add webhook.
   * Set the Payload URL to <http://your-jenkins-server/github-webhook/>.
   * <http://54.211.136.75:8080/github-webhook/>
   * Select Just the push event--Click Add webhook.

Jenkins—manage jenkins—plugin—available plugin—maven integration—install.

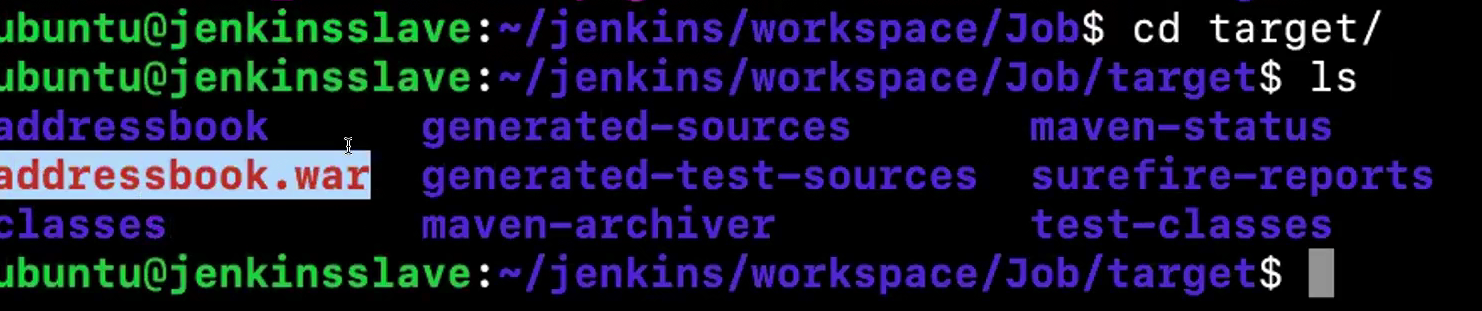
**Manage jenkins**—tools—**maven install**—**install auto**—save.

EC2 for slave—create node for slave.

Create Job—git—URL—branch name—build steps—invoke top maven target—goal=clean install—build now.



TARGET folder will be created.



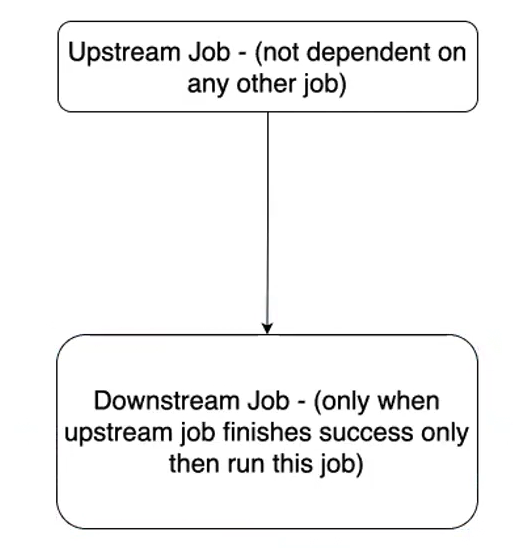
With the help of maven .war file will be created.

**Upstream Job**

* **Definition**: An upstream job is one that triggers another job after it completes.
* **Example**: If Job A completes and then triggers Job B, Job A is the upstream job of Job B.

**Downstream Job**

* **Definition**: A downstream job is one that gets triggered by another job.
* **Example**: If Job A completes and then triggers Job B, Job B is the downstream job of Job A.



**Simple Example**

Let's say you have two jobs:

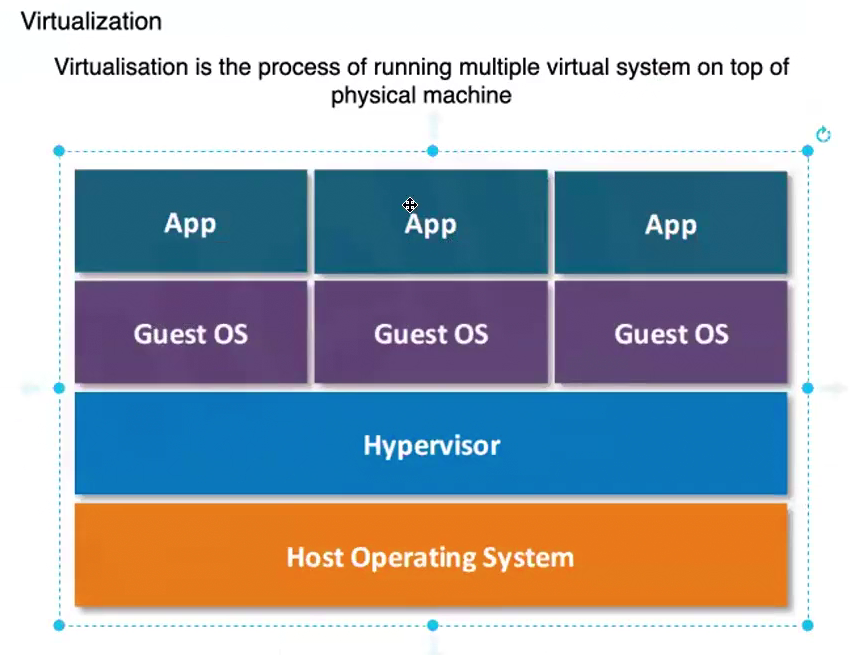
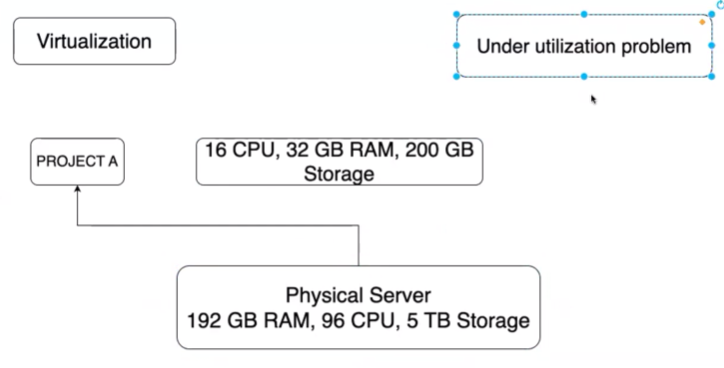
1. **Build Project** (Job A): This job compiles the code.
2. **Run Tests** (Job B): This job runs tests on the compiled code.

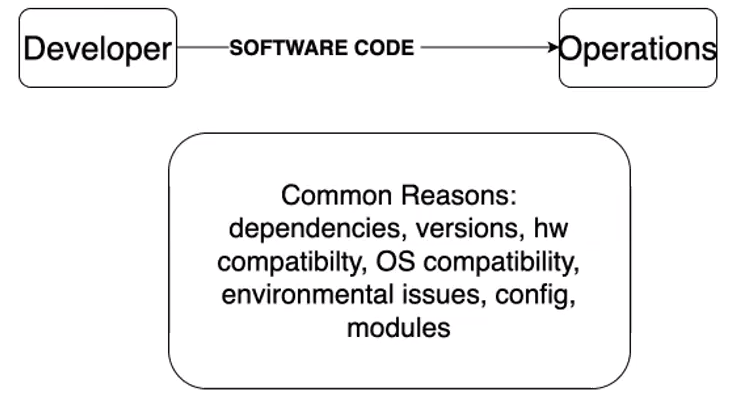
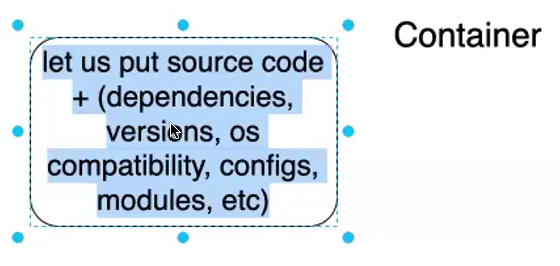
**How to Set Up Upstream and Downstream Jobs:**

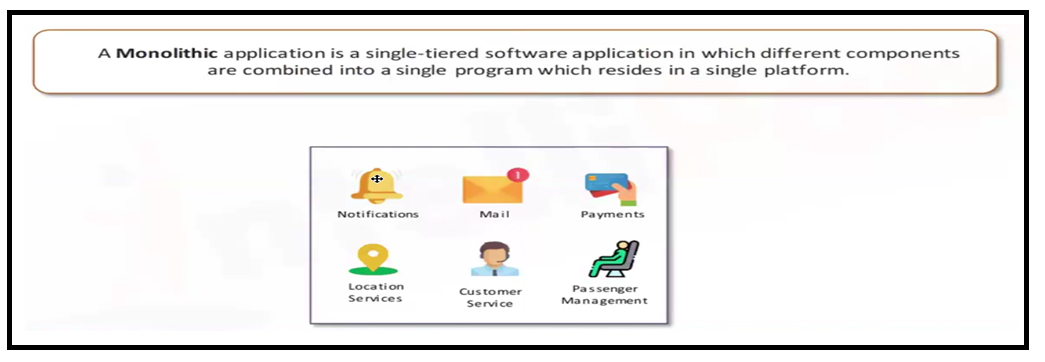
**In a Freestyle Project:**

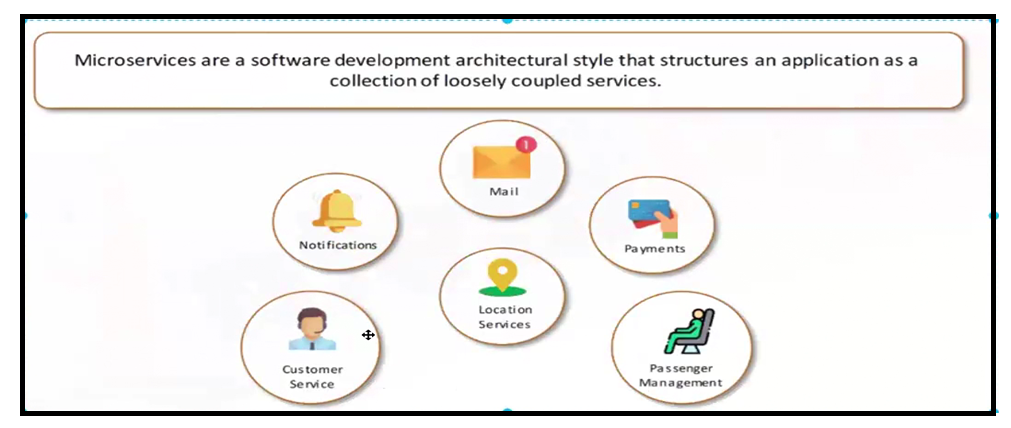
1. **Configure Job A (Upstream Job)**:
   * Go to the job configuration.
   * Add a build step or **post-build action** to trigger Job B.
2. **Configure Job B (Downstream Job)**:
   * No special configuration is needed in Job B to be a downstream job. It's triggered by Job A.

# **Virtualization** is a technology that allows you to create multiple systems from a single physical hardware system.









EC2—UBUNTU—RUN .sh file.

#!/bin/bash

# Update existing list of packages

sudo apt update -y

# Install prerequisite packages which let apt use packages over HTTPS

sudo apt install -y apt-transport-https ca-certificates curl software-properties-common

# Add the GPG key for the official Docker repository to your system

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

# Add the Docker repository to APT sources

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"

# Update the package database with the Docker packages from the newly added repo

sudo apt update -y

# Install Docker

sudo apt install -y docker-ce

# Start Docker service

sudo systemctl start docker

# Enable Docker service to start on boot

sudo systemctl enable docker

# Add current user to the Docker group to run Docker commands without sudo (optional)

sudo usermod -aG docker ${USER}

# Print Docker version to verify installation

docker --version

# Print success message

echo "Docker has been installed successfully!"

**chmod +x filename.sh**

**./filename.sh**

**Docker Image**: A blueprint (recipe) containing all the necessary parts to run an application.

**Docker Container**: A running instance (dish) created from the image (recipe), capable of running the application.

**docker run hello-world**

1. **docker run**: This command creates a new container and runs it.
2. **hello-world**: This is the name of the Docker image you want to run. If the image is not already available locally, Docker will pull it from the Docker Hub registry.

docker pull nginx (it will only pull image but won’t create container).

docker run nginx

docker run -d nginx: (docker run -d --name myserver nginx )

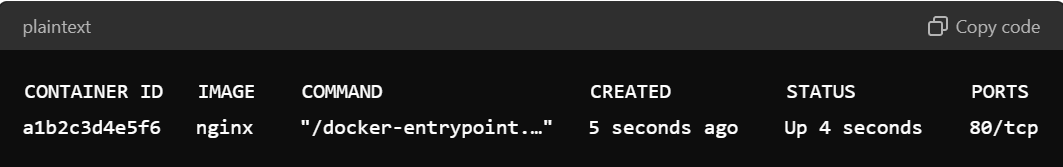
1. **docker run**: This command creates and starts a new Docker container.
2. **-d**: This flag runs the container in detached mode, meaning it runs in the background.
3. **nginx**: This is the name of the Docker image you want to run. If the Nginx image is not already available locally, Docker will pull it from the Docker Hub registry.

When you run this command, you should receive a container ID as output, indicating that the Nginx container is running in the background.

To verify that the container is running, you can use the following command:

docker ps: (ps stands for **process status**)

This will list all the running containers.



docker rename <container id> <name>

docker ps -a (both running and (exited))

**to access the shell of a running Nginx container using the docker exec command**

docker exec -it mynginx bash

Combined as -it, these options allow for an interactive terminal session inside the Docker container.

docker stop <id>

docker ps -a

docker ps -qa 🡪 only id

docker rm <id> -------🡪delete

docker rm <id> -f 🡪stop and delete(f=force)

docker rm <id> <id> -----🡪 2 container will be deleted

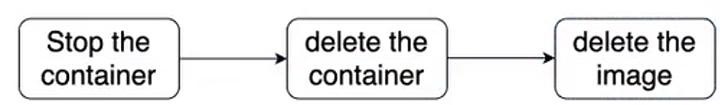
docker container prune 🡪 delete all the container which are exit state(stopped)

docker stop $(docker ps -qa)

docker rm $(docker ps -qa)

docker images vs docker ps

docker rmi <id> 🡪remove image(delete)



docker run -d -p 8080:80 nginx

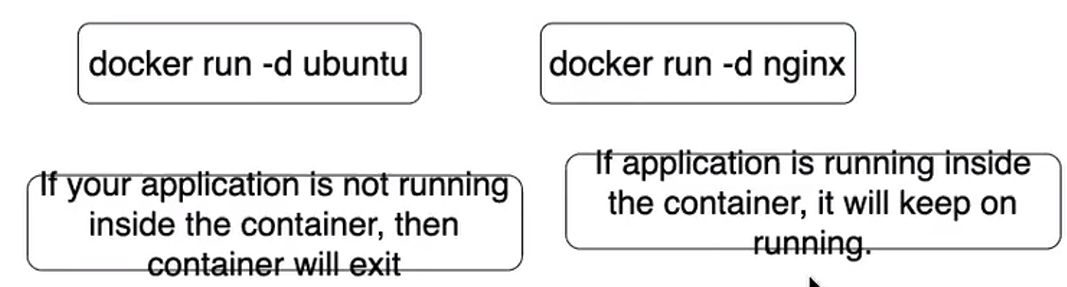
In this command:

* **8080**: The port on the host machine.
* **80**: The port inside the Docker container where the Nginx server is listening.
* This maps port 8080 on the host to port 80 in the container, allowing you to access the Nginx server using <http://localhost:8080>
* Ec2 Server—inbound rule custom TCP🡪port 8080 from anywhere.

docker run ubuntu

docker ps

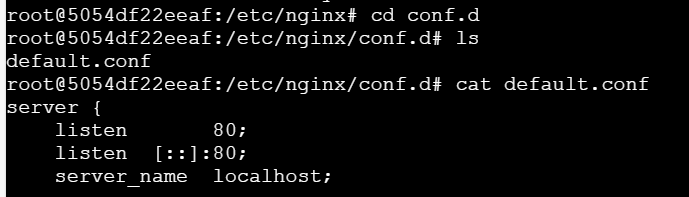
docker ps -a



You can check. Goto inside the continer(ngnix):

docker exec -it mynginx bash

cd etc/nginx/conf.d



exit

docker run -itd ubuntu 🡪 user is going to interact with terminal. (trick)

you can create shortcut like:

docker exec -it mynginx bash == dex mynginx bash

vi ~/.bashrc 🡪 open the .bashrc file.

**Add the alias**: Scroll to the bottom of the .bashrc file and add the following alias:

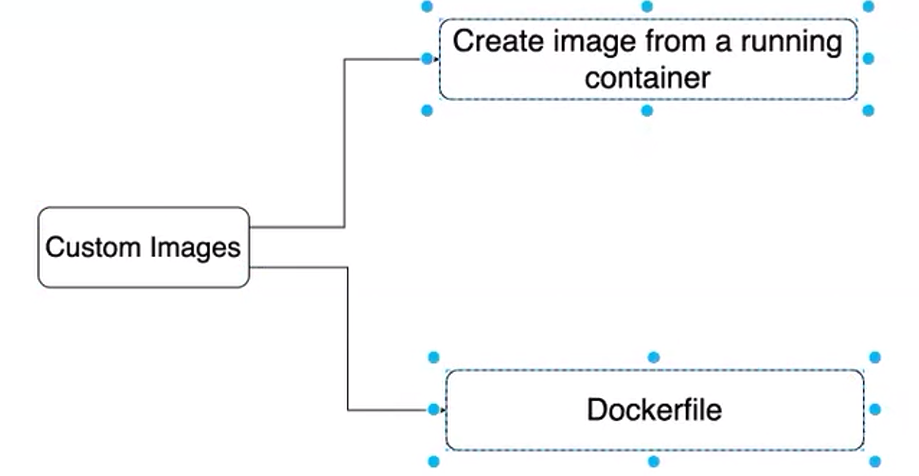
**alias dex ="docker exec -it"**

**Reload the .bashrc file: Run the following command to apply the changes:**

**source ~/.bashrc**

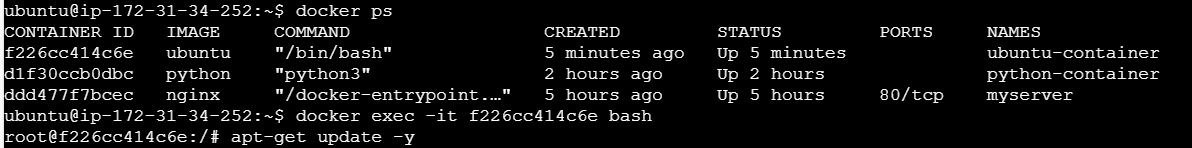
**Use the shortcut**: Now, whenever you want to run a command inside a container, use dex followed by the container name and the command. For example, to start a bash session in the nginx container:

dex mynginx bash



**Create an Image from the Container:**

docker ps



apt-get update -y

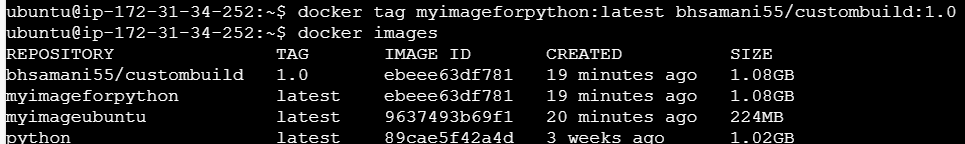
apt-get install apache2

exit

docker commit <container\_id\_or\_name> <your\_image\_name>

**docker images**

docker tag myimageforpython:latest bhsamani55/custombuild:1.0 🡪(rename image name as per dockerhub name)



**Log in to Docker Hub**

First, log in to Docker Hub using the **docker login** command. You will be prompted to enter your Docker Hub username and password.

**Tag the Image**

If you haven't already tagged the image with the appropriate name and tag, do so now. Since you've already tagged your image as bhsamani55/custombuild:1.0, you can skip this step

**Push the Image to Docker Hub**

Use the docker push command to push the image to Docker Hub:

docker push <name>:<tag>

docker push bhsamani55/custombuild:1.0

if u delete the image:

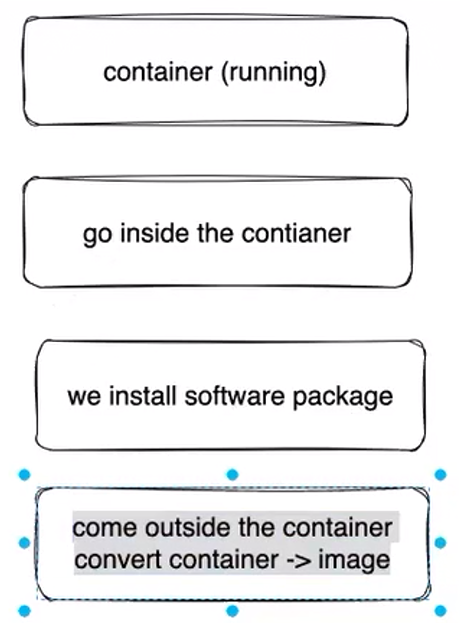
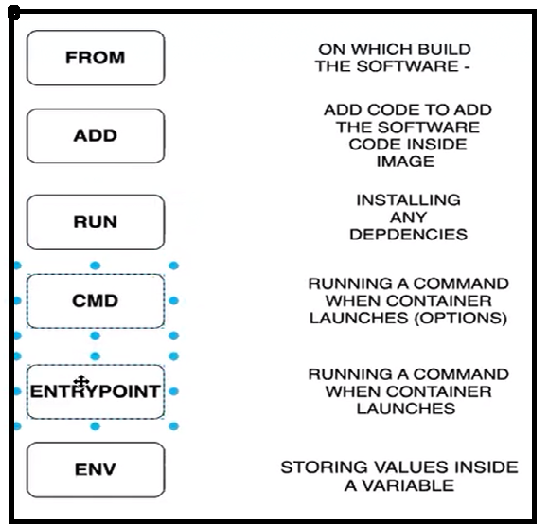
docker rmi -f bhsamani55/custombuild:1.0

docker rmi ebeee63df781

**To download the image from dockerhub**

docker pull bhsamani55/custombuild:1.0

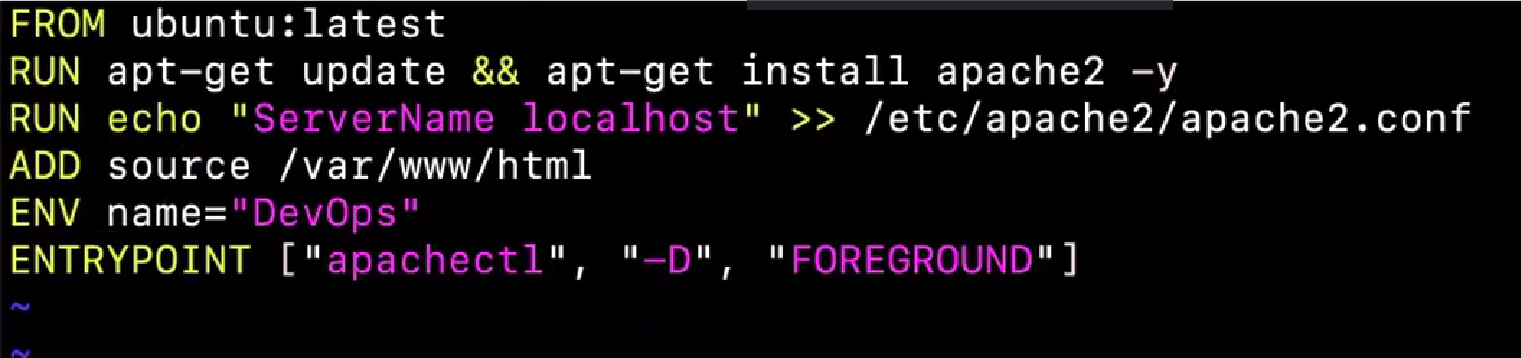
**Dockerfile** 🡪instructions🡪create Image.



Use **CMD** when you need a default command that can be overridden.

Use **ENTRYPOINT** when you need a specific command to always run, ensuring the container behaves like a dedicated executable.

vi Dockerfile



docker build <name> .

DOCKER VOLUME: (data persistance)

What happen if container gets deleted.

**TO create docker volume:**

**docker volume** **create** bhavesh

It will be created to /var/lib/docker/volumes/bhavesh

**Syntax:**

bash

Copy code

docker run -d --mount source=<volume\_name>,destination=<container\_path> <image\_ID >

* **-d**: Runs the container in **detached mode** (in the background).
* **--mount**: This is used to mount a volume or bind a directory from the host to the container. The mount syntax is:

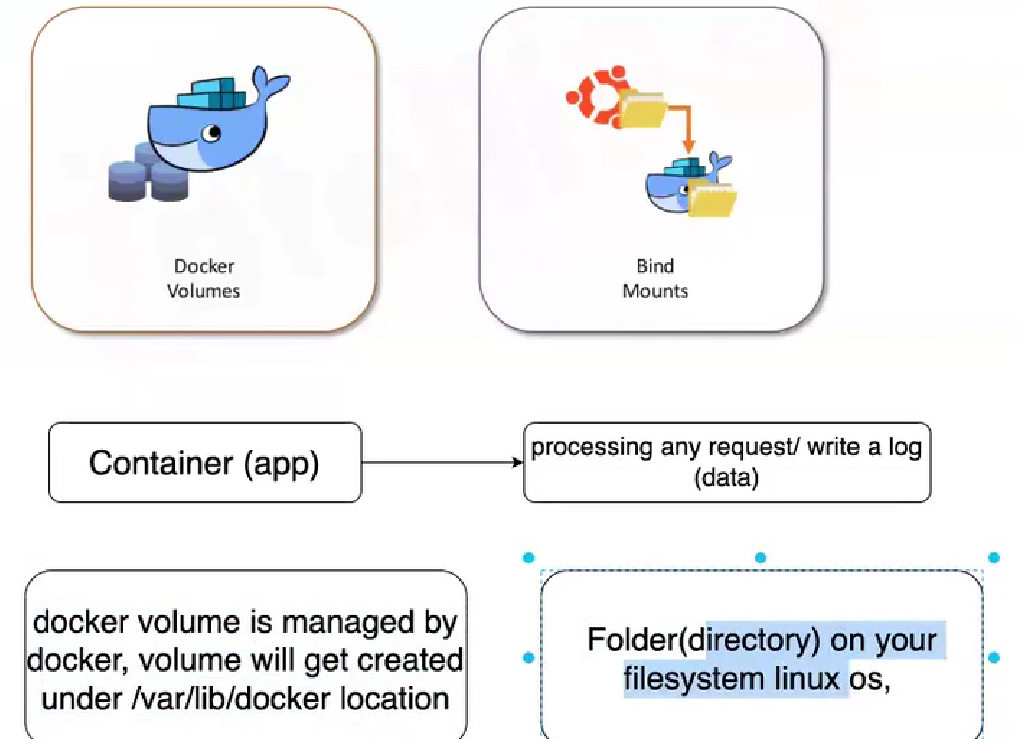
docker run -itd --mount source=bhavesh,destination=/app dbe68f247902

cd app

ls

echo “app log has been created” > bhs.log

now you can check inside Bhavesh folder.



**Dockerfile**:

* Used to define the steps to create a Docker image.
* Focuses on building the environment for a single container.

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

docker run -d nginx🡪only 1 container will be created

docker compose 🡪multi container can be created.

**Docker Compose**:

* Written in YAML syntax.
* Describes the services, networks, and volumes for running multiple containers.

**Command:**

vi docker-compose.yml



Sample1 & sample2 are container names.

To execute the file:

docker compose up -d ----to bring up all containers **with file name docker-compose**.

docker compose down ---to bring down all the containers which are created by docker compose.

Repo link:  
https://github.com/suhailasad/JenkinsMavenCode.git

GitHub + Jenkins + Maven + Docker + Kubernetes

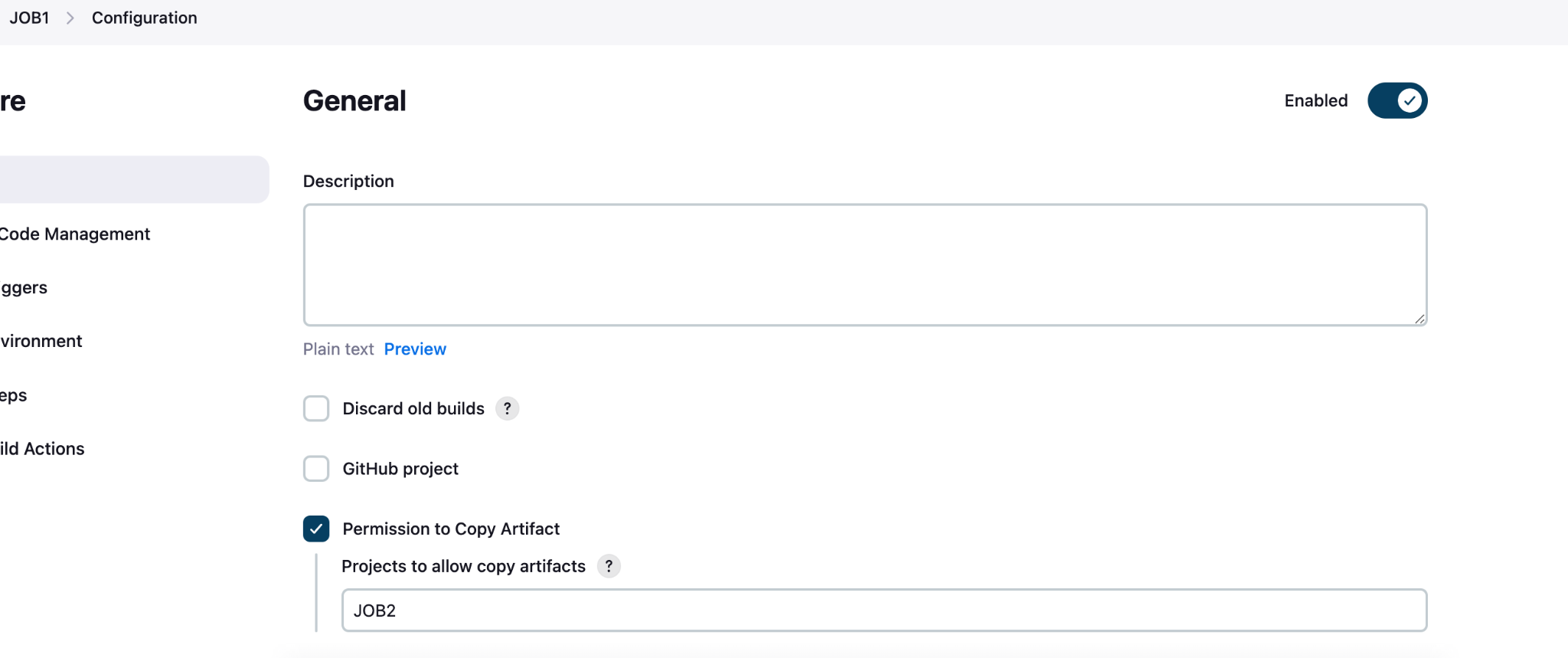
Configure the job and commands as per below screenshots;  
   
List of plugins to be installed: maven, copy artifact

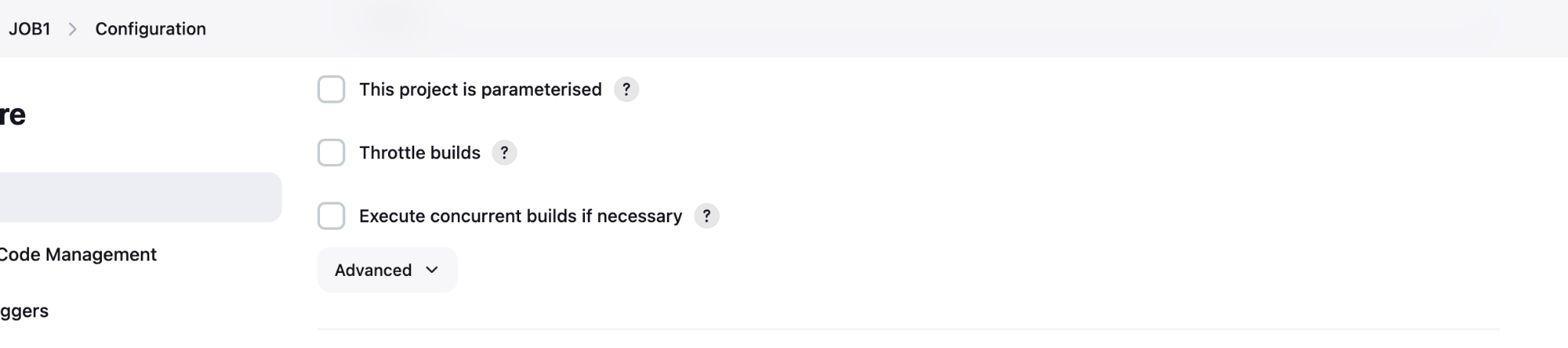
Please see the recording to see how to install the maven plugin and configure it.

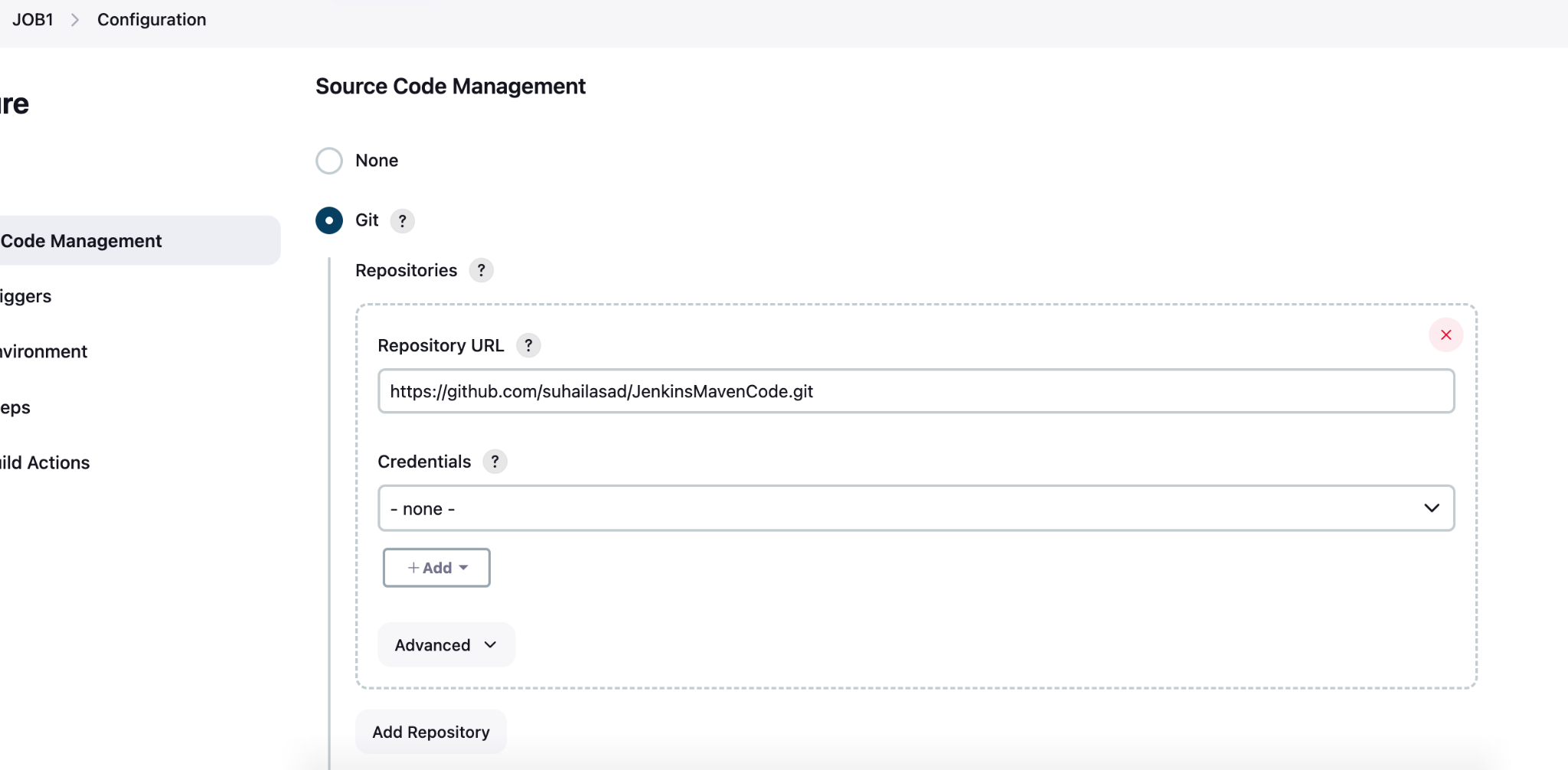
In Jenkins instance, install docker and kubectl utility

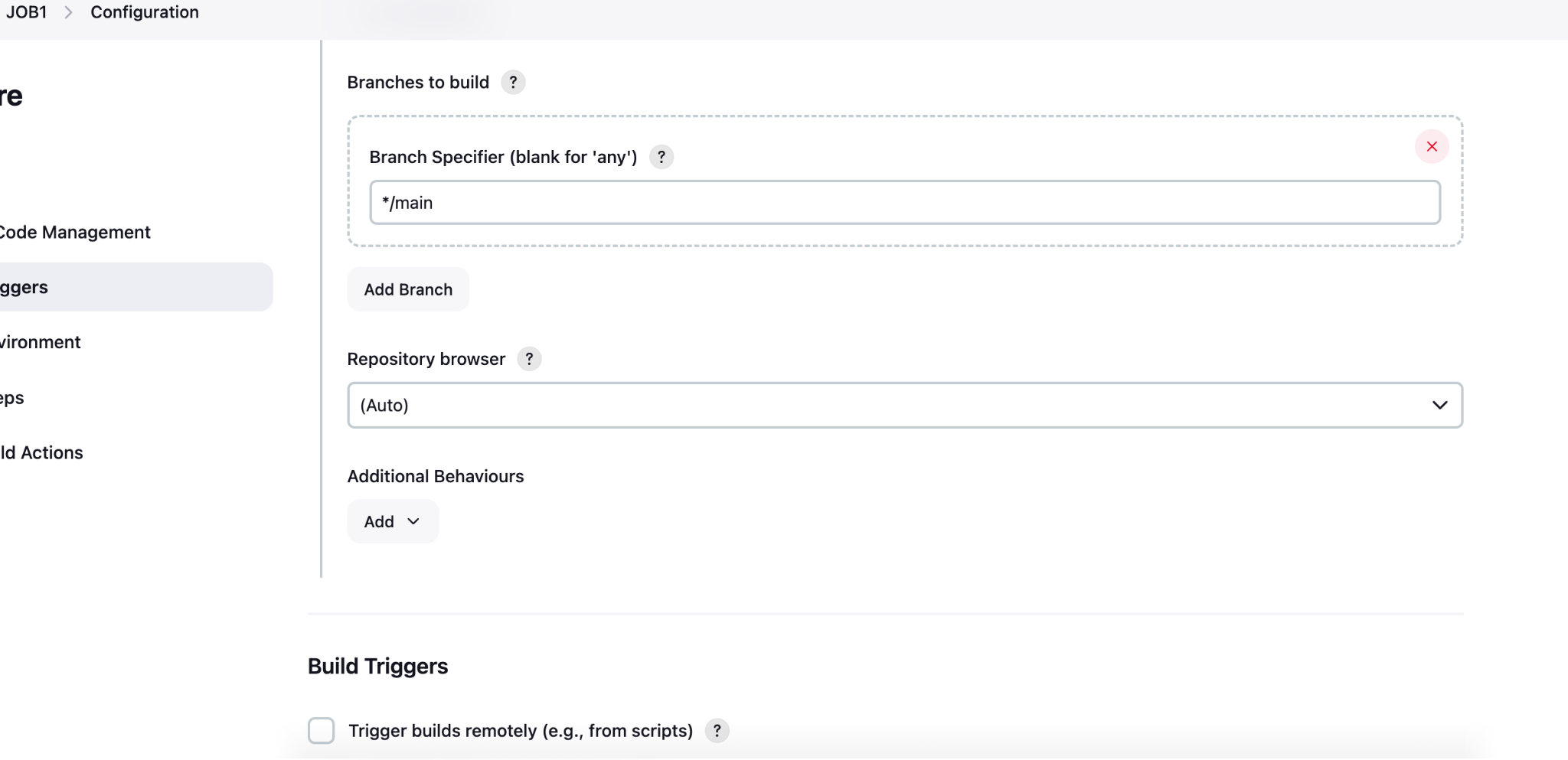
Sudo snap install kubectl –classic

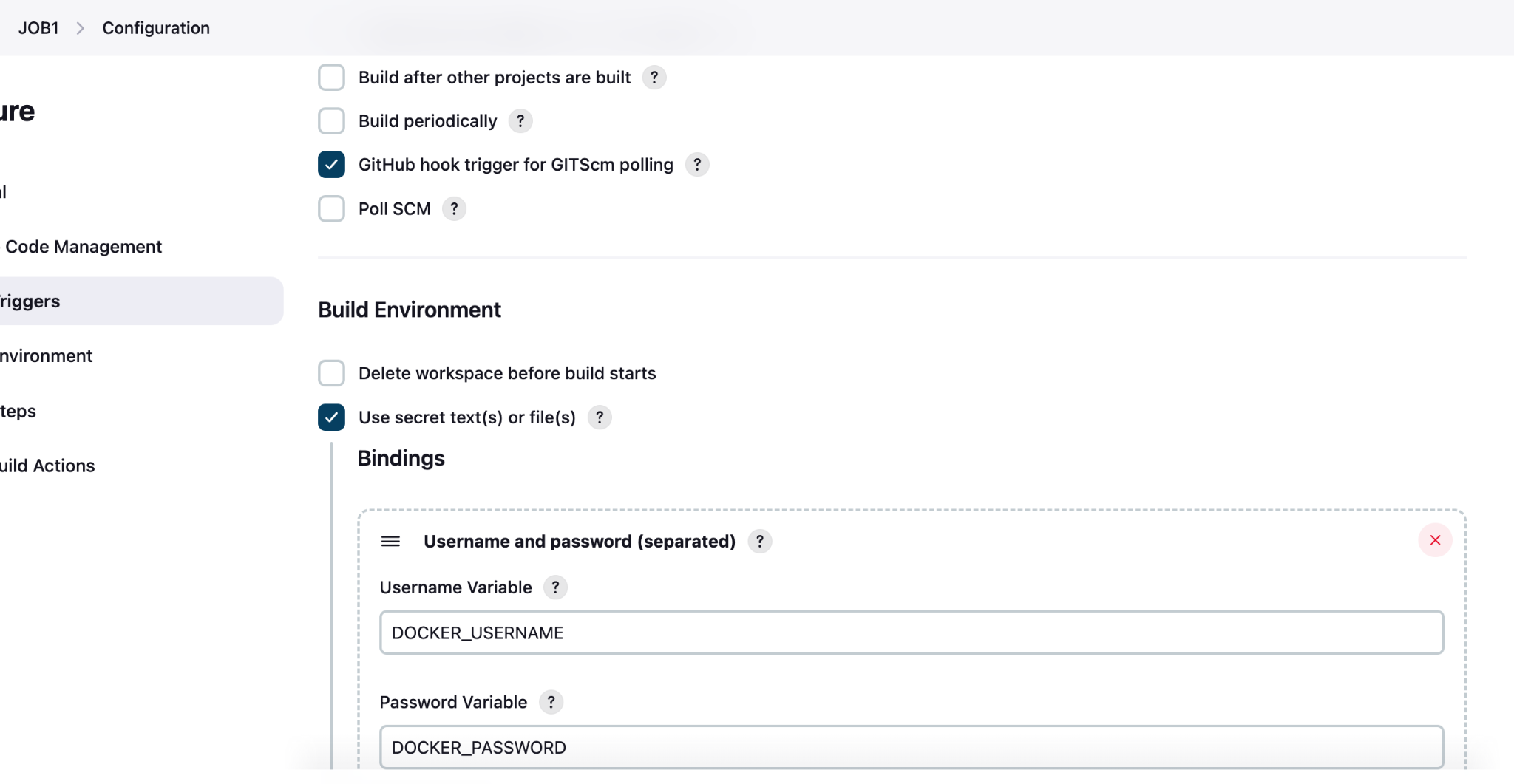
**JOB1**: config steps and commands

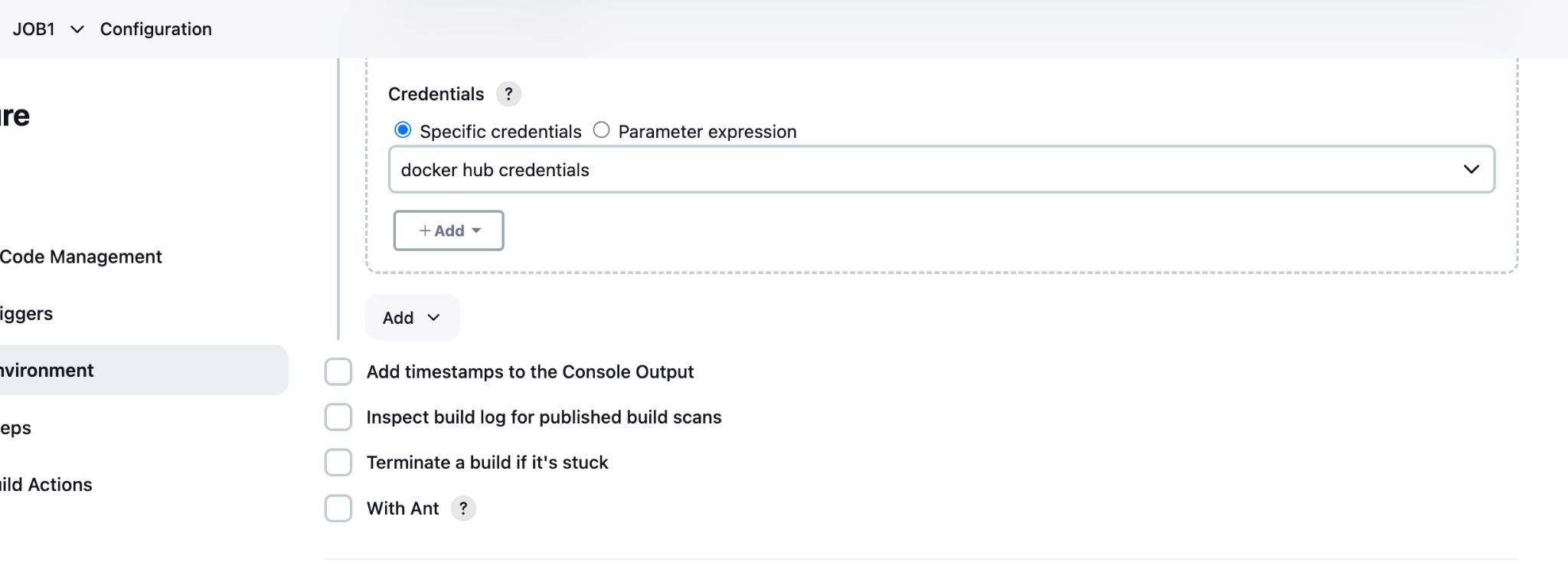


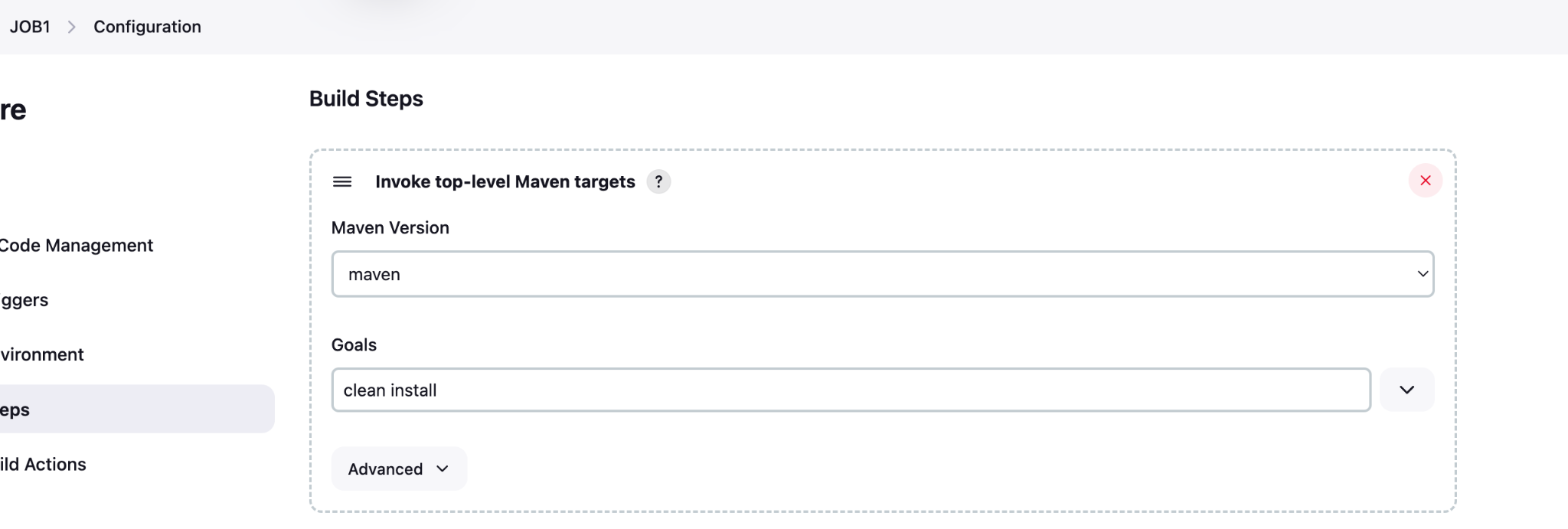


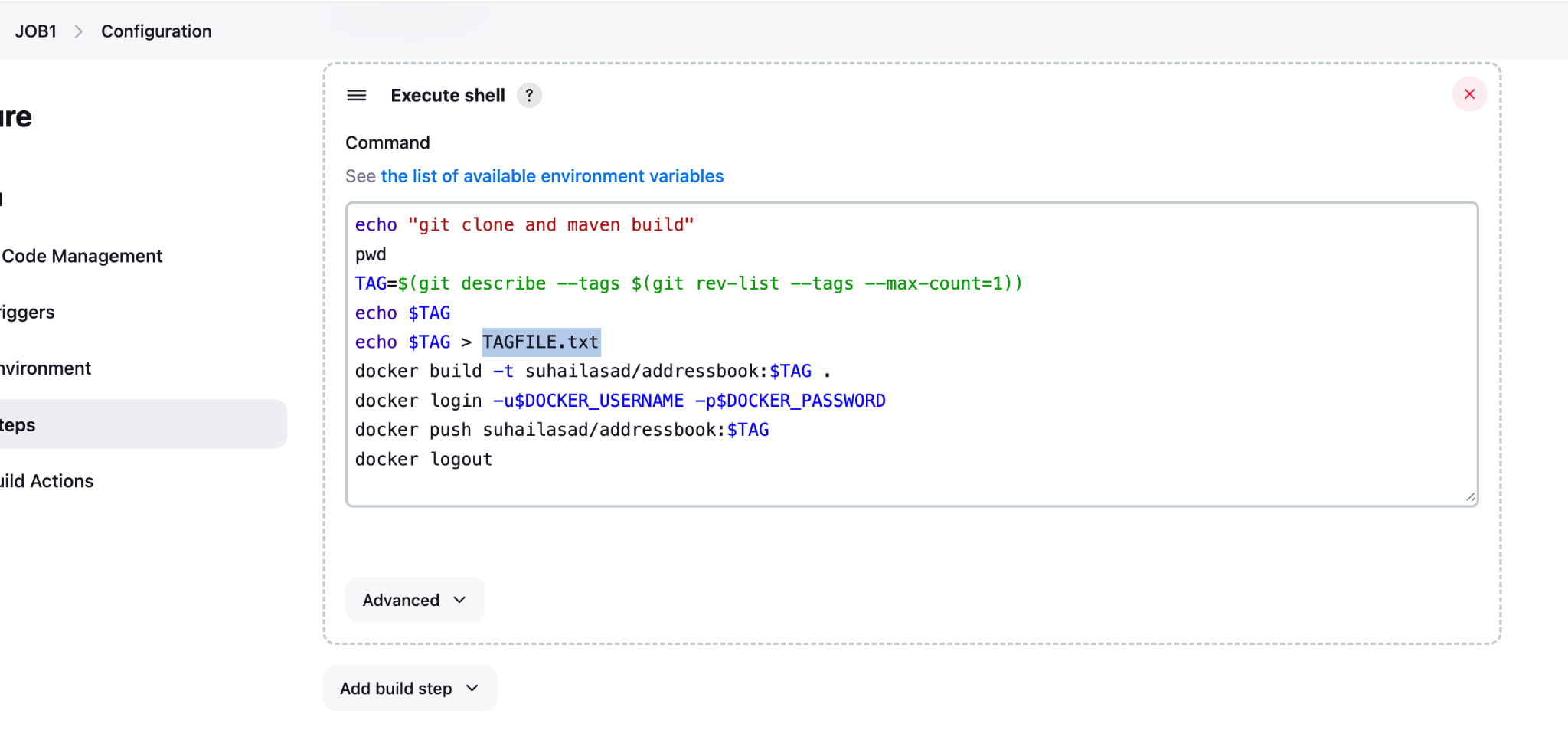


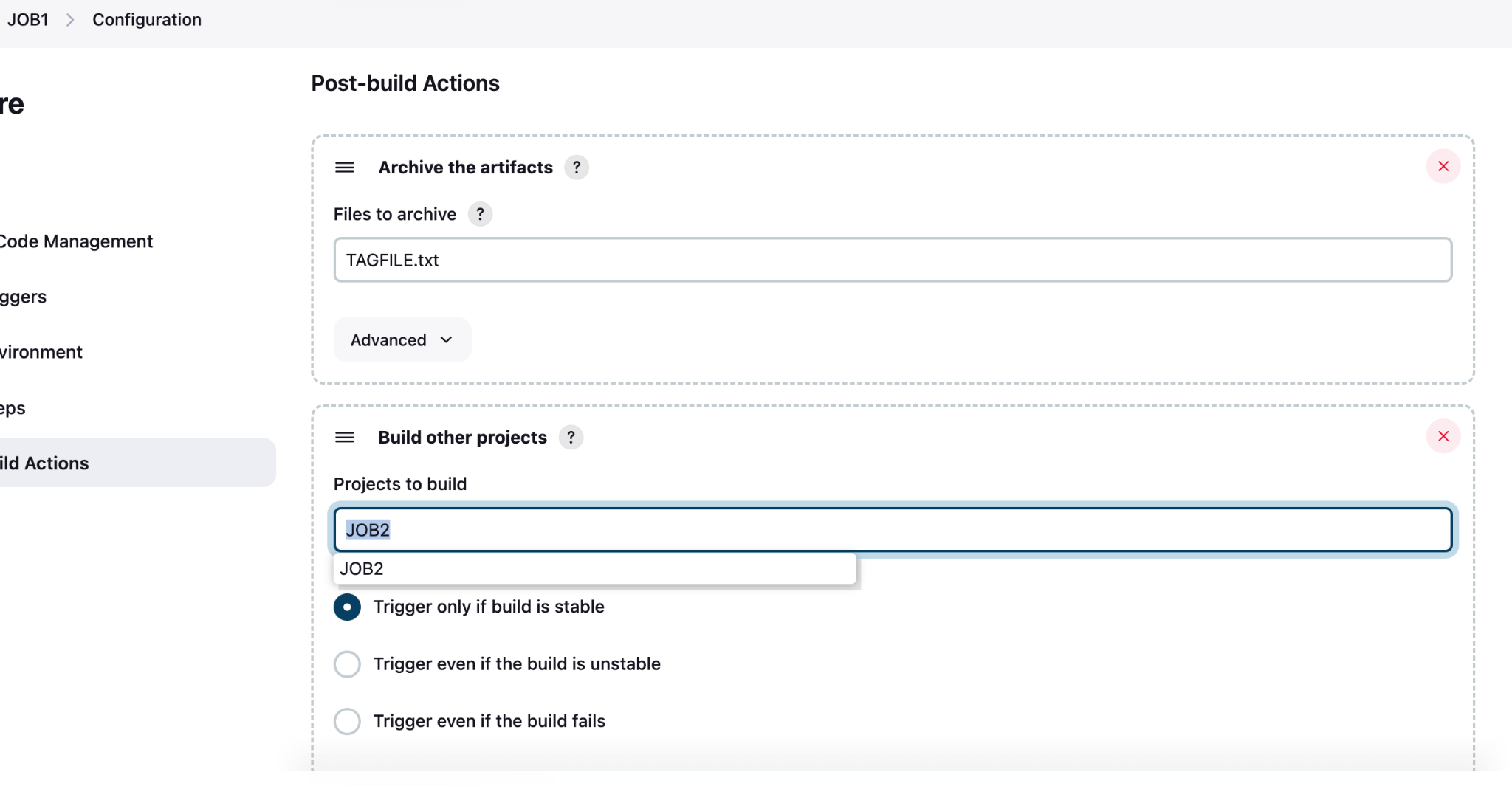












**JOB2**: config steps and commands

