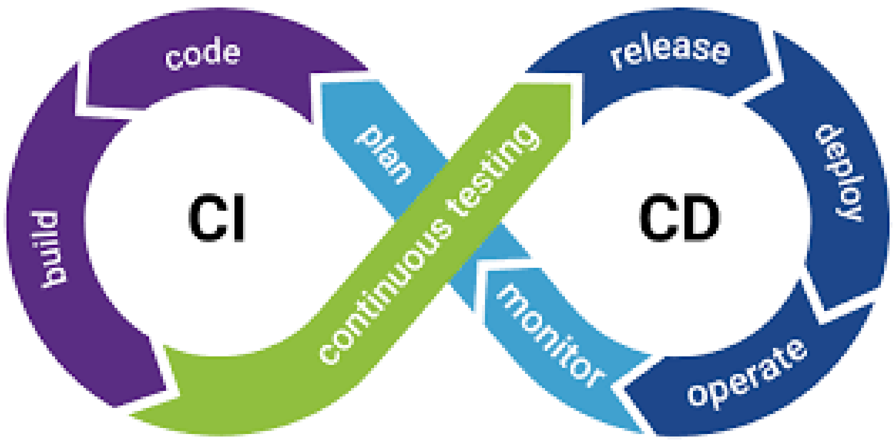
Day 23 Task: Jenkins Freestyle Project for DevOps Engineers.



What is CI/CD?

Continuous Integration (CI) and Continuous Deployment (CD) are software development methodologies that aim to automate and streamline the software development process.

CI refers to the practice of regularly integrating code changes into a shared repository, such as Git, by individual developers. This integration helps to catch conflicts early in the development process and ensure that code changes do not break the application when combined with other changes.

Once code is integrated, the CI process kicks in, automatically building and testing the application to make sure it meets the necessary quality standards. This involves running a suite of automated tests, such as unit tests, integration tests, and regression tests, to verify the application's functionality.

CD refers to the process of automatically deploying code changes to production once the CI process is complete and tests are successful. This allows for quick and reliable delivery of new features and bug fixes to end-users, without the need for manual intervention.

CI/CD pipelines are typically automated, using tools like Jenkins, Travis CI, CircleCI, and GitLab CI/CD, to manage the process of code integration, building, testing, and deployment. This helps to ensure that the development process is efficient and consistent, and that software releases are of high quality.

- **What Is a Build Job?**

A build job is a step in the Continuous Integration (CI) process that is responsible for compiling and assembling the source code into a deployable software application. It takes the source code that has been committed to a version control system, such as Git, and builds it into an executable package.

The build job performs a series of tasks, such as:

- Checking out the source code from the repository
- Compiling the code
- Running static code analysis tools to catch potential problems

- Running automated tests to validate the code
- Packaging the code into an executable format, such as a JAR, WAR, or EXE file
- Storing the build artifacts for later use in the deployment process.

The outcome of a build job is a deployable artifact, which can then be used as input for further stages in the CI/CD pipeline, such as testing and deployment.

What is Freestyle Projects ?? 😲

Freestyle projects in Jenkins are a type of build job that provides a flexible and configurable way to automate the software build process. They allow developers to define the steps involved in the build process, including checking out the source code, compiling, testing, and packaging the application.

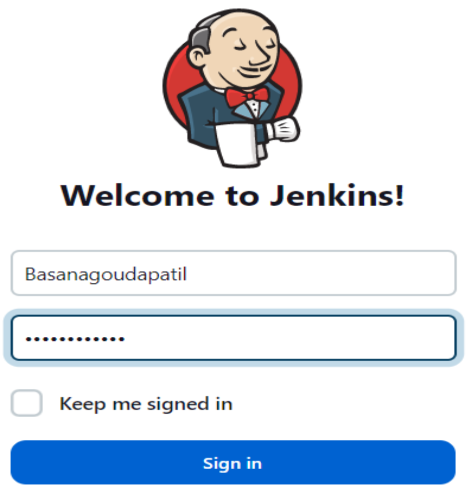
Freestyle projects provide a high degree of customization, making them a suitable choice for a wide range of software development projects. They are easy to set up and can be configured for different use cases, such as building and deploying applications, running tests, and performing other build-related tasks.

Here are a few tasks that you could complete when working with a freestyle project in Jenkins:

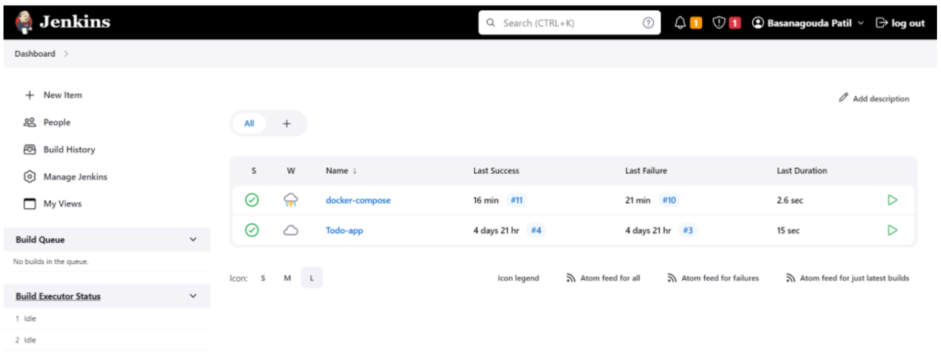
Task-01

1. Create a new Jenkins freestyle project for your app.

- Log in to Jenkins and navigate to the main dashboard.

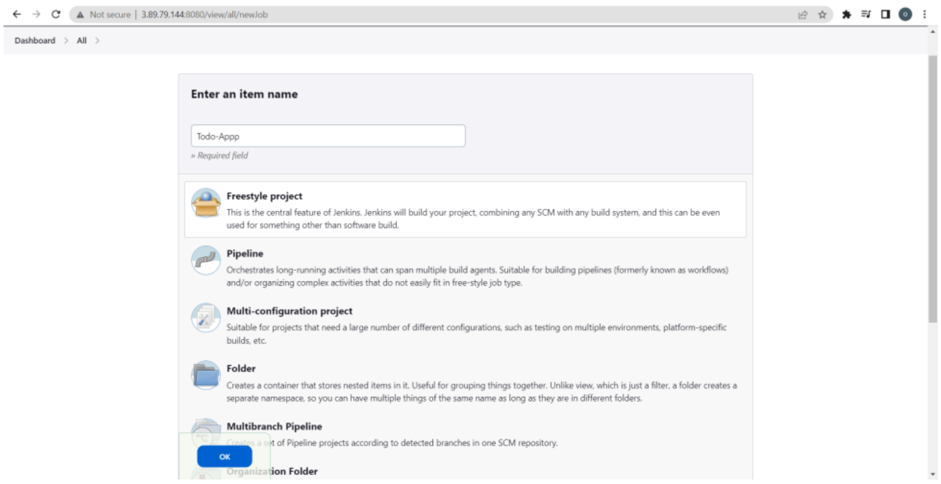


- Click on the "New Item" button to create a new project.



S	W	Name ↓	Last Success	Last Failure	Last Duration
✓		docker-compose	16 min #11	21 min #10	2.6 sec
✓		Todo-app	4 days 21 hr #4	4 days 21 hr #3	15 sec

- Select "Freestyle project" and give the project a name.
- Click on the "OK" button to create the project.



Enter an item name

Todo-Appp

» Required field

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

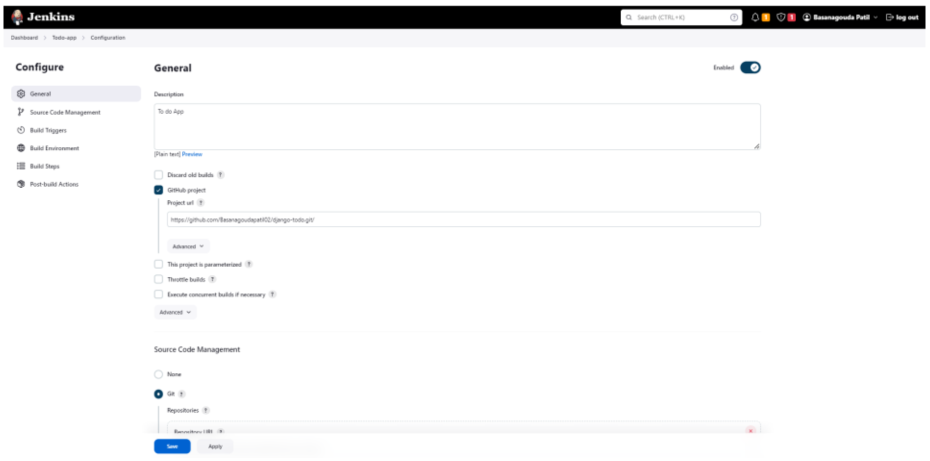
Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

Multibranch Pipeline
Creates a set of Pipeline projects according to detected branches in one SCM repository.

- In the project configuration page, you can specify the details of the project, such as the source code management system, build triggers, and build actions. In GitHub project write your GitHub project repository URL.



General

Description: To do App

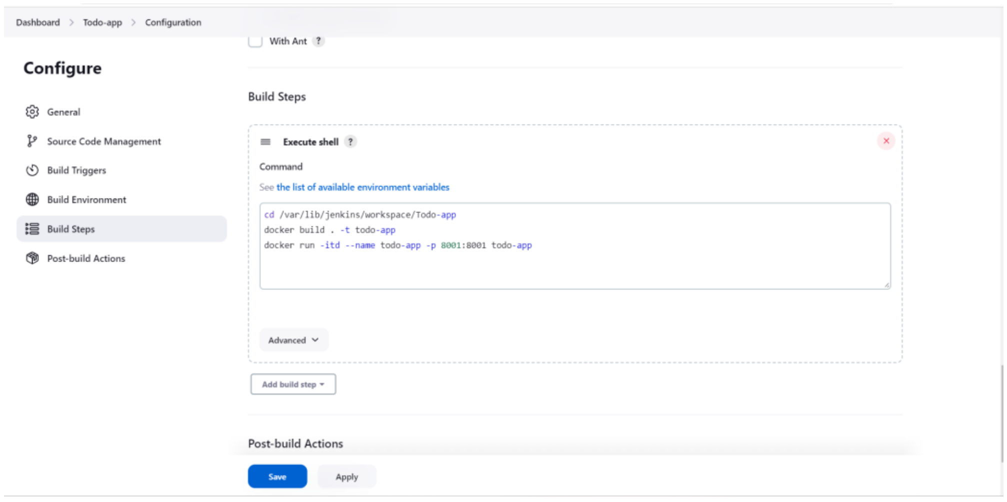
[Plain text] Preview

- [ ] Discard old builds
- [x] GitHub project
  - Project url: https://github.com/Basanagoudapatil02/django-todo.git/
- [ ] This project is parameterized
- [ ] Throttle builds
- [ ] Execute concurrent builds if necessary

Source Code Management

- ○ None
- ● Git
  - Repositories
  - Repository URL

- In the “Build” section, click on the “Add build step” button and select “Execute shell” option.



```
cd /var/lib/jenkins/workspace/Todo-app
docker build . -t todo-app
docker run -itd --name todo-app -p 8001:8001 todo-app
```

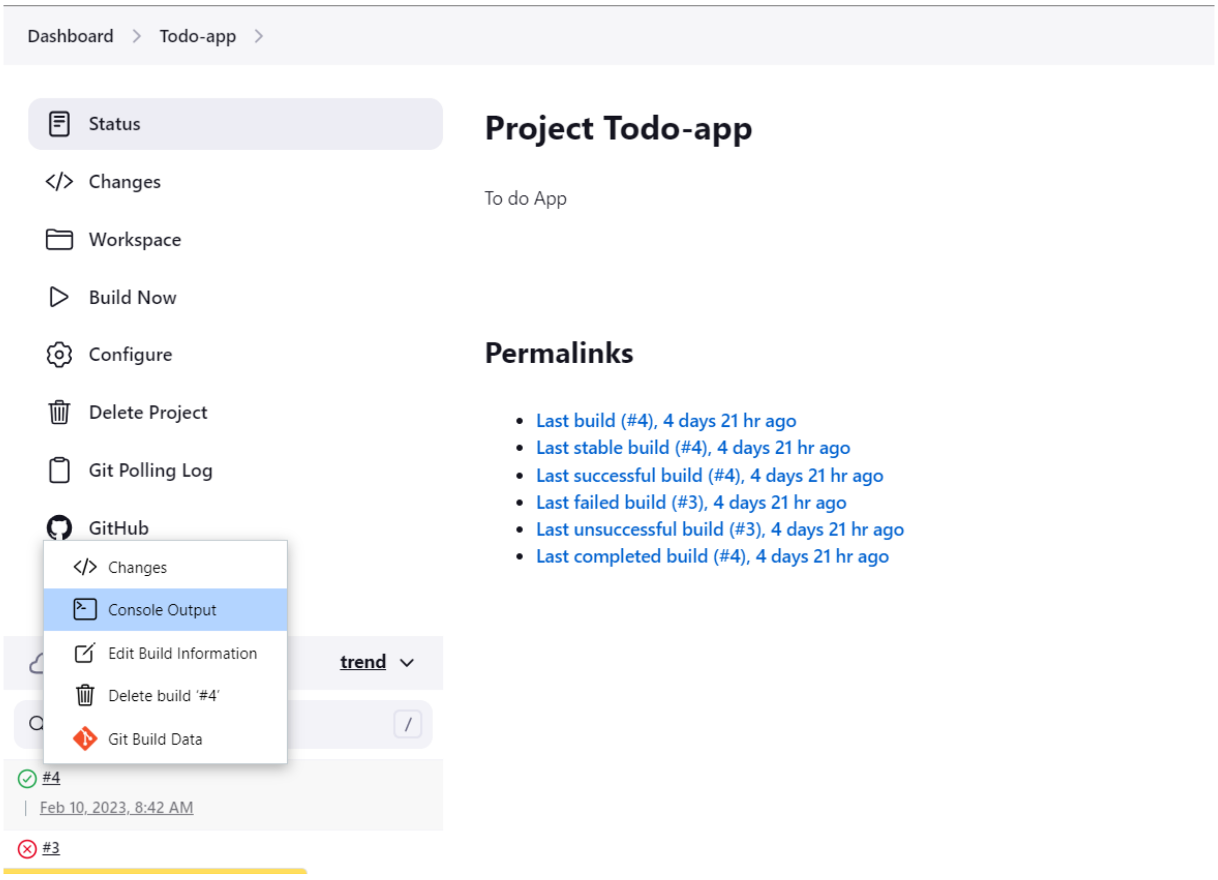
- In the “Command” field, enter the following command

cd /var/lib/jenkins/workspace/Todo-app

docker build . -t todo-app

docker run -itd --name todo-app -p 8001:8001 todo-app

- Click on the "Save" button to save the project configuration.
- Build the project: You can manually build the project by clicking on the "Build Now" link in the project's main page. This will start the build process and execute the steps specified in the project configuration.
- To view the console output of a Jenkins build, follow these steps:
- Go to the Jenkins dashboard and select the project that you want to view the console output for.
- Click on the build number of the build that you want to view the console output for.
- In the build details page, you will see a button named “Console Output”. Click on the button to view the console output.
- The console output will show you the output of the build steps, including the output of any shell commands that were run as part of the build.

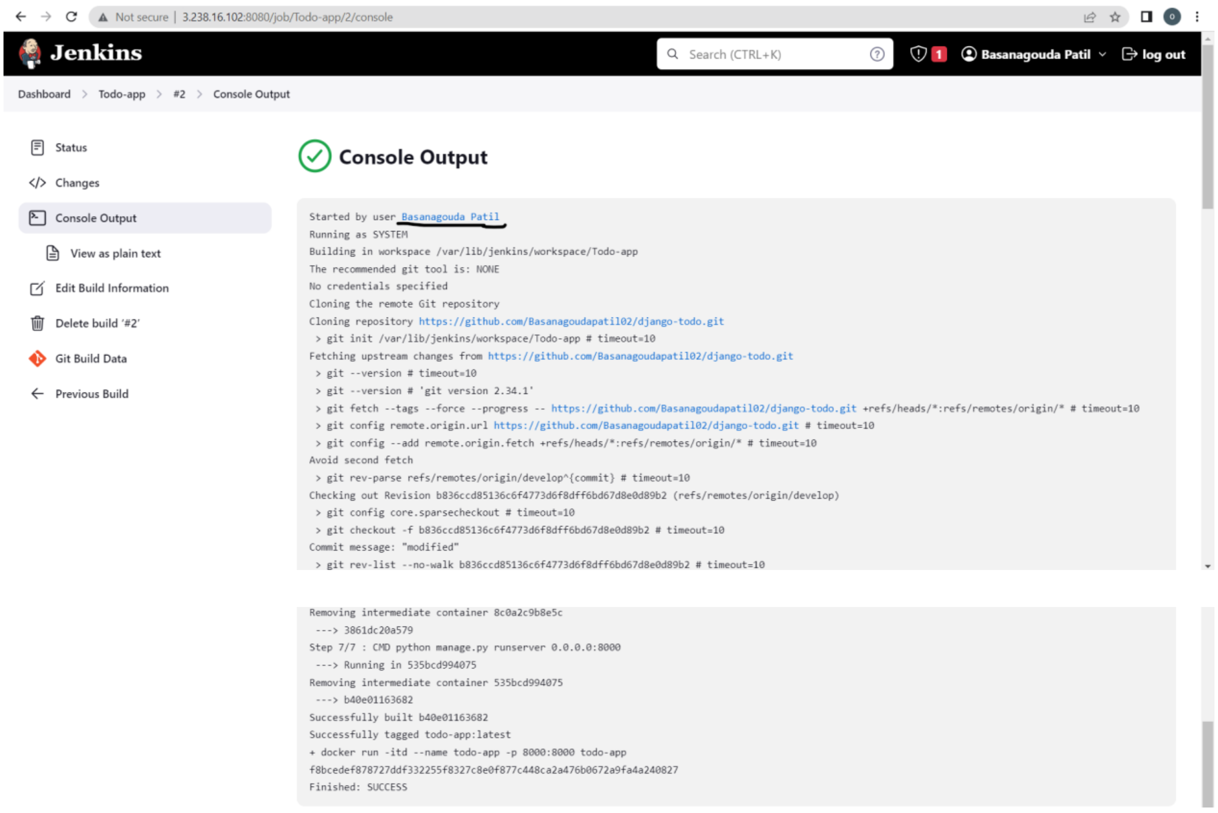


Project Todo-app

To do App

Permalinks

- Last build (#4), 4 days 21 hr ago
- Last stable build (#4), 4 days 21 hr ago
- Last successful build (#4), 4 days 21 hr ago
- Last failed build (#3), 4 days 21 hr ago
- Last unsuccessful build (#3), 4 days 21 hr ago
- Last completed build (#4), 4 days 21 hr ago

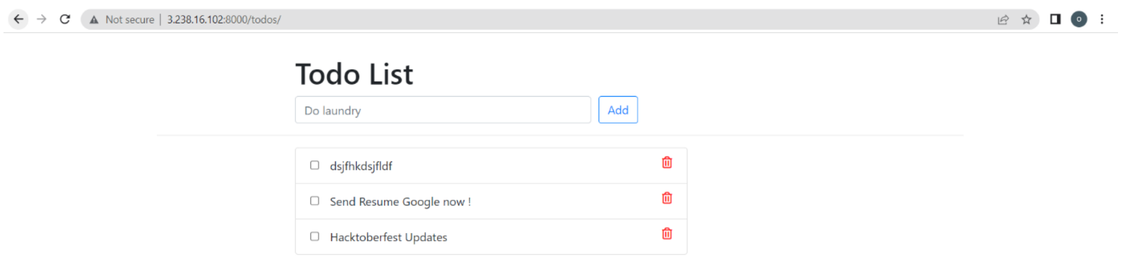


Console Output

```
Started by user Basanagouda Patil
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/Todo-app
The recommended git tool is: NONE
No credentials specified
Cloning the remote Git repository
Cloning repository https://github.com/Basanagoudapatil02/django-todo.git
 > git init /var/lib/jenkins/workspace/Todo-app # timeout=10
Fetching upstream changes from https://github.com/Basanagoudapatil02/django-todo.git
 > git --version # timeout=10
 > git --version # 'git version 2.34.1'
 > git fetch --tags --force --progress -- https://github.com/Basanagoudapatil02/django-todo.git +refs/heads/*:refs/remotes/origin/* # timeout=10
 > git config remote.origin.url https://github.com/Basanagoudapatil02/django-todo.git # timeout=10
 > git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
Avoid second fetch
 > git rev-parse refs/remotes/origin/develop^{commit} # timeout=10
Checking out Revision b836ccd85136c6f4773d6f8dff6bd67d8e0d89b2 (refs/remotes/origin/develop)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f b836ccd85136c6f4773d6f8dff6bd67d8e0d89b2 # timeout=10
Commit message: "modified"
 > git rev-list --no-walk b836ccd85136c6f4773d6f8dff6bd67d8e0d89b2 # timeout=10
```

```
Removing intermediate container 8c0a2c9b8e5c
 ---> 3861dc20a579
Step 7/7 : CMD python manage.py runserver 0.0.0.0:8000
 ---> Running in 535bcd994075
Removing intermediate container 535bcd994075
 ---> b40e01163682
Successfully built b40e01163682
Successfully tagged todo-app:latest
+ docker run -itd --name todo-app -p 8000:8000 todo-app
f8bcedef878727ddf332255f8327c8e0f877c448ca2a476b0672a9fa4a240827
Finished: SUCCESS
```

- Set the inbound rule on AWS and access the app at “public-ip:8000”


Not secure | 3.238.16.102:8000/todos/

Todo List

Do laundry

Add

- dsjfhkdsjfldf
- Send Resume Google now !
- Hacktoberfest Updates

Task-02

- Create Jenkins project to run "docker-compose up -d" command to start the multiple containers defined in the compose file (Hint- use day-19 Application & Database docker-compose file)

  Things to conform before going to Jenkins

  Install docker-compose on server using command

  sudo apt-get install docker-compose

  docker-compose file should be present in project folder

- Log in to your Jenkins instance and click on the “New Item” button.

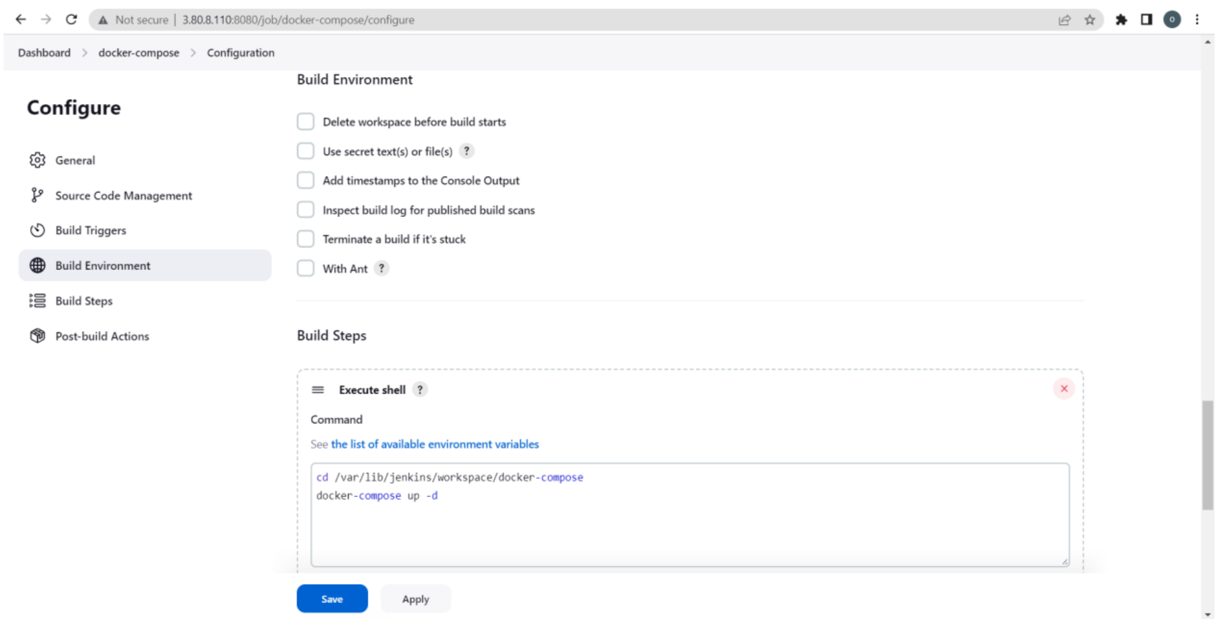
- Give your project a name “docker-compose” , select “Freestyle project” as the type, and click on the “OK” button.

- Add your GitHub project URL.

- In the “Build” section, click on the “Add build step” button and select “Execute shell” option.

- In the “Command” field, enter the following command

  ```
  cd /var/lib/jenkins/workspace/docker-compose
  docker-compose up -d
  ```



Build Environment

- [ ] Delete workspace before build starts
- [ ] Use secret text(s) or file(s)
- [ ] Add timestamps to the Console Output
- [ ] Inspect build log for published build scans
- [ ] Terminate a build if it's stuck
- [ ] With Ant

Build Steps

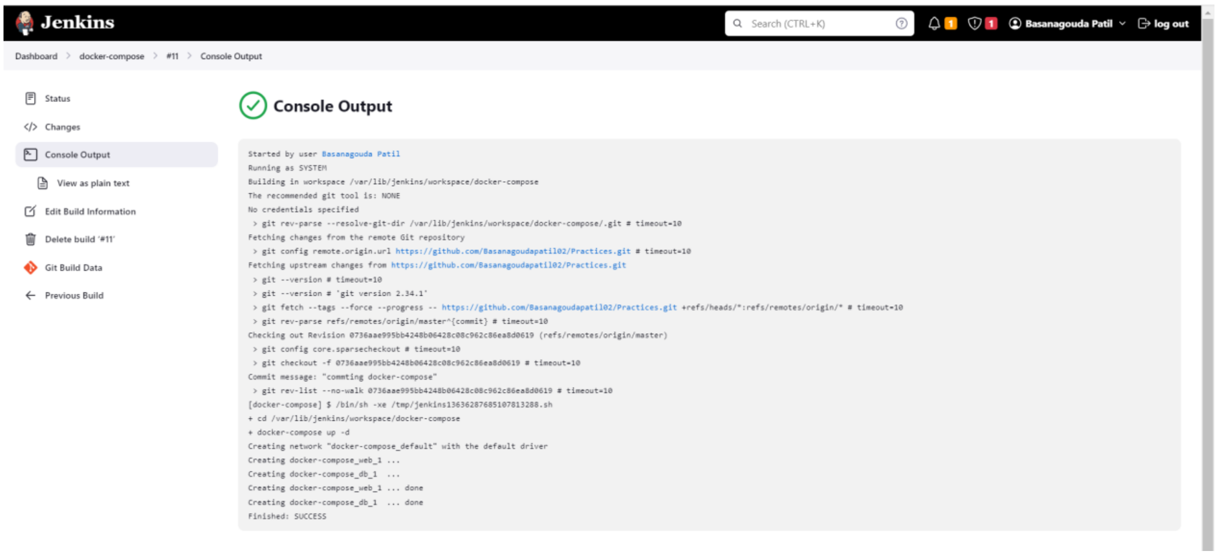
Execute shell

Command

See the list of available environment variables

```
cd /var/lib/jenkins/workspace/docker-compose
docker-compose up -d
```

- After the successful build , your project should be able to :



Console Output

```
Started by user Basanagouda Patil
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/docker-compose
The recommended git tool is: NONE
No credentials specified
 > git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/docker-compose/.git # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url https://github.com/Basanagoudapatil02/Practices.git # timeout=10
Fetching upstream changes from https://github.com/Basanagoudapatil02/Practices.git
 > git --version # timeout=10
 > git --version # 'git version 2.34.1'
 > git fetch --tags --force --progress -- https://github.com/Basanagoudapatil02/Practices.git +refs/heads/*:refs/remotes/origin/* # timeout=10
 > git rev-parse refs/remotes/origin/master^{commit} # timeout=10
Checking out Revision 0736aae995bb4248b06428c08c962c86ea8d0619 (refs/remotes/origin/master)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f 0736aae995bb4248b06428c08c962c86ea8d0619 # timeout=10
Commit message: "commting docker-compose"
 > git rev-list --no-walk 0736aae995bb4248b06428c08c962c86ea8d0619 # timeout=10
[docker-compose] $ /bin/sh -xe /tmp/jenkins13636287685107813288.sh
+ cd /var/lib/jenkins/workspace/docker-compose
+ docker-compose up -d
Creating network "docker-compose_default" with the default driver
Creating docker-compose_web_1 ...
Creating docker-compose_db_1  ...
Creating docker-compose_web_1 ... done
Creating docker-compose_db_1  ... done
Finished: SUCCESS
```

Happy Learning:)