Experiment No. 7

Title:

Applying CI/CD Principles to Web Development Using Jenkins, Git, using Docker Containers.

Objective:

The objective of this experiment is to set up a CI/CD pipeline for a web application using Jenkins, Git, Docker containers, and GitHub webhooks. The pipeline will automatically build, test, and deploy the web application whenever changes are pushed to the Git repository, without the need for a pipeline script.

Introduction:

Continuous Integration and Continuous Deployment (CI/CD) principles are integral to modern web development practices, allowing for the automation of code integration, testing, and deployment. This experiment demonstrates how to implement CI/CD for web development using Jenkins, Git, Docker containers, and GitHub webhooks without a pipeline script. Instead, we'll utilize Jenkins' "GitHub hook trigger for GITScm polling" feature.

In the fast-paced world of modern web development, the ability to deliver high-quality software efficiently and reliably is paramount. Continuous Integration and Continuous Deployment (CI/CD) are integral principles and practices that have revolutionized the way software is developed, tested, and deployed. These practices bring automation, consistency, and speed to the software development lifecycle, enabling development teams to deliver code changes to production with confidence.

Continuous Integration (CI):

CI is the practice of frequently and automatically integrating code changes from multiple contributors into a shared repository. The core idea is that developers regularly merge their code into a central repository, triggering automated builds and tests.

Key aspects of CI include:

● **Automation:** CI tools, like Jenkins, Travis CI, or CircleCI, automate the building and testing of code whenever changes are pushed to the repository.

● **Frequent Integration:** Developers commit and integrate their code changes multiple times a day, reducing integration conflicts and catching bugs early.

● **Testing:** Automated tests, including unit tests and integration tests, are run to ensure that new code changes do not introduce regressions.

● **Quick Feedback:** CI provides rapid feedback to developers about the quality and correctness of their code changes.

Continuous Deployment (CD):

CD is the natural extension of CI. It is the practice of automatically and continuously deploying code changes to production or staging environments after successful integration and testing.

Key aspects of CD include:

● **Automation:** CD pipelines automate the deployment process, reducing the risk of human error and ensuring consistent deployments.

● **Deployment to Staging:** Code changes are deployed first to a staging environment where further testing and validation occur.

● **Deployment to Production:** After passing all tests in the staging environment, code changes are automatically deployed to the production environment, often with zero downtime.

● **Rollbacks:** In case of issues, CD pipelines provide the ability to rollback to a previous version quickly.

Benefits of CI/CD in Web Development:

● **Rapid Development:** CI/CD accelerates development cycles by automating time-consuming tasks, allowing developers to focus on coding.

● **Quality Assurance:** Automated testing ensures code quality, reducing the number of bugs and regressions.

● **Consistency:** CI/CD ensures that code is built, tested, and deployed consistently, regardless of the development environment.

● **Continuous Feedback:** Developers receive immediate feedback on the impact of their changes, improving collaboration and productivity.

● **Reduced Risk:** Automated deployments reduce the likelihood of deployment errors and downtime, enhancing reliability.

● **Scalability:** CI/CD can scale to accommodate projects of all sizes, from small startups to large enterprises.

Prerequisites:

● A computer with Docker installed.

● Jenkins installed and configured (https://www.jenkins.io/download/).

● A web application code repository hosted on GitHub.

Experiment Steps:

Step 1: Set Up the Web Application and Git Repository.

● Create a simple web application or use an existing one. Ensure it can be hosted in a Docker container.

● Initialize a Git repository for your web application and push it to GitHub.

Step 2: Install and Configure Jenkins

● Install Jenkins on your computer or server following the instructions for your operating system (https://www.jenkins.io/download/).

● Open Jenkins in your web browser (usually at http://localhost:8080) and complete the initial setup, including setting up an admin user and installing necessary plugins.

● Configure Jenkins to work with Git by setting up Git credentials in the Jenkins Credential Manager.

- Make sure that the user `jenkins` exists.

  ```
  cat /etc/passwd | grep jenkins
  ```

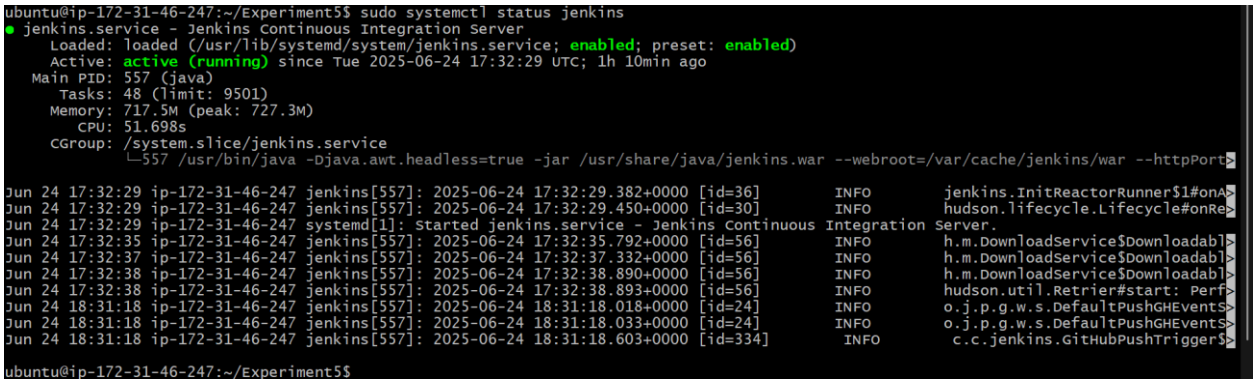
```
ubuntu@ip-172-31-46-247:~/Experiment5$ cat /etc/passwd | grep jenkins
jenkins:x:111:114:Jenkins,,,:/var/lib/jenkins:/bin/bash
ubuntu@ip-172-31-46-247:~/Experiment5$
```

- Add `jenkins` user to Docker group so that we can run docker commands without sudo.

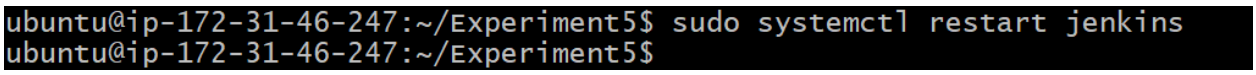
```
ubuntu@ip-172-31-46-247:~/Experiment5$ sudo usermod -aG docker jenkins
ubuntu@ip-172-31-46-247:~/Experiment5$
```

- Check the status of Jenkins service and restart it to make the changes to be affected.

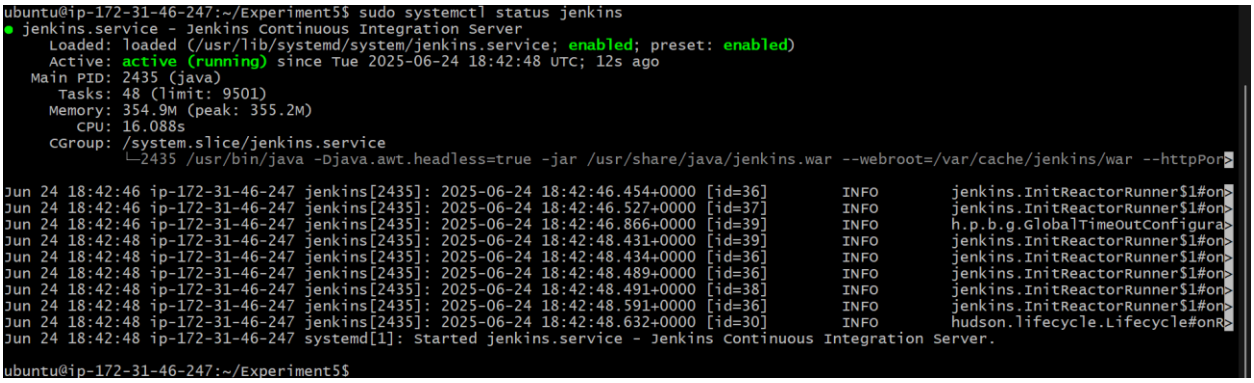
  ```
  sudo systemctl status jenkins
  ```




```
sudo systemctl restart jenkins
```



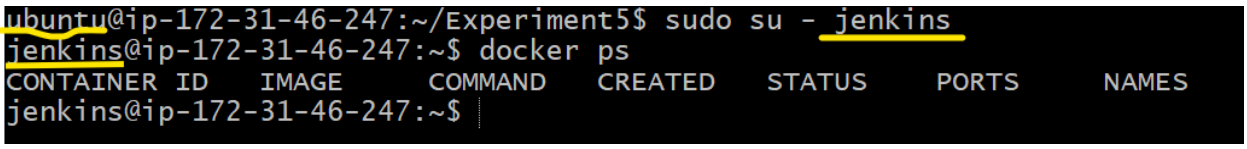

```
sudo systemctl status jenkins
```



- Verify that the user Jenkins can run docker commands without sudo now.

```
sudo su - jenkins
```

```
docker ps
```



Become ubuntu user after verification.

```
exit
```

```
jenkins@ip-172-31-46-247:~$ exit
logout
ubuntu@ip-172-31-46-247:~/Experiment5$
```

NOTE: *Because of restarting the Jenkins service, you may need to re-login in the UI.*

- Create a Dockerfile

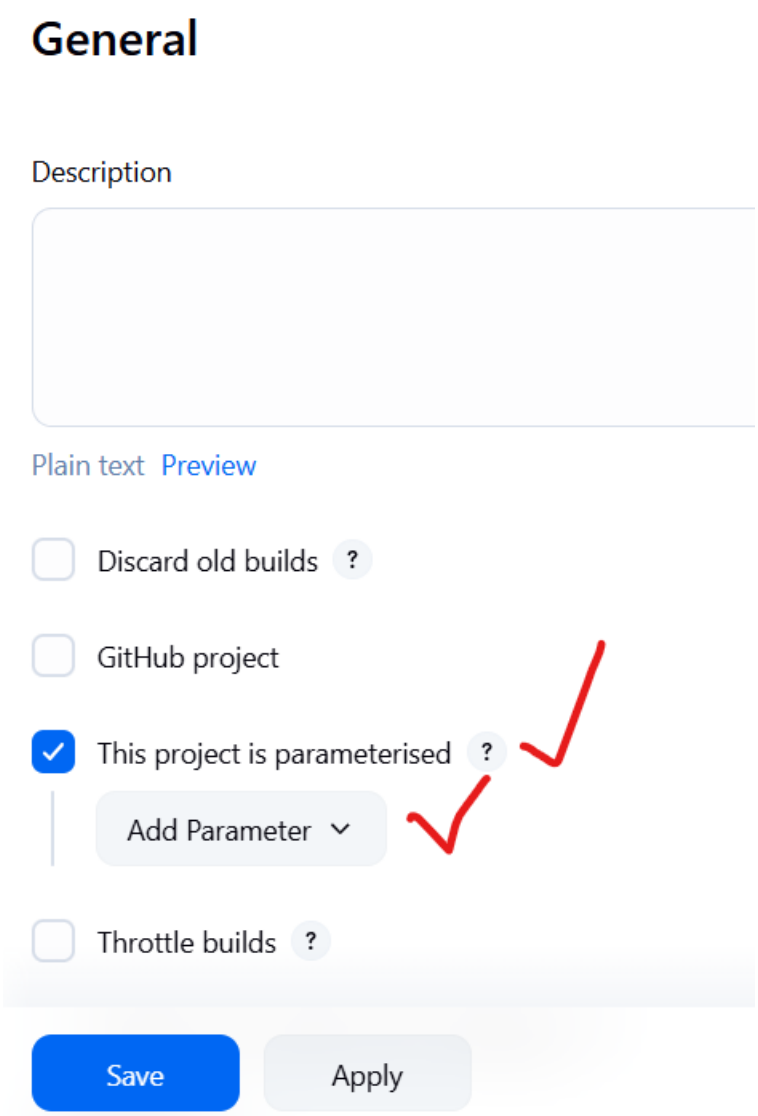
```
FROM nginx:alpine
COPY . /usr/share/nginx/html
```

```
FROM nginx:alpine
COPY . /usr/share/nginx/html
~
```

Step 3: Create a Jenkins Job

● Create a new Jenkins job using the "Freestyle project" type.

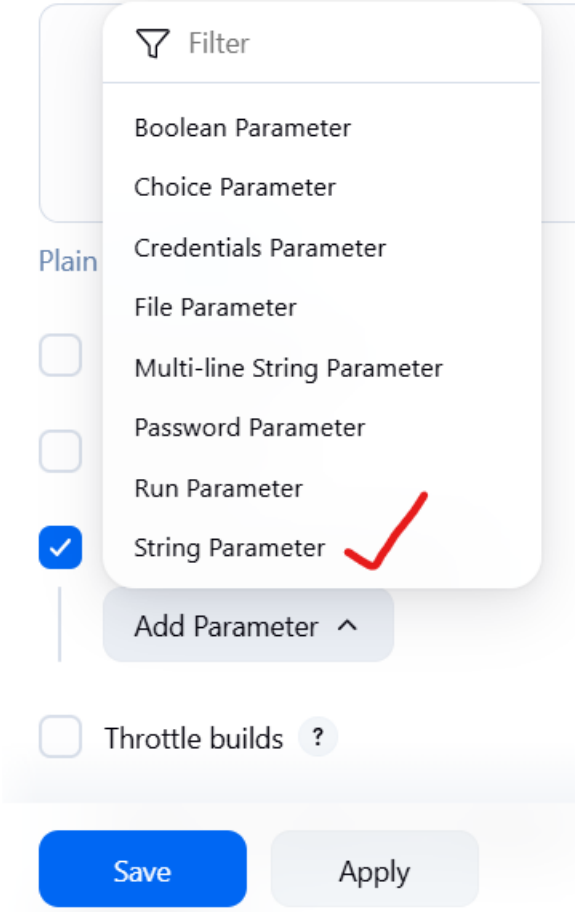
● In the job configuration, specify a name for your job and choose "This project is parameterized."

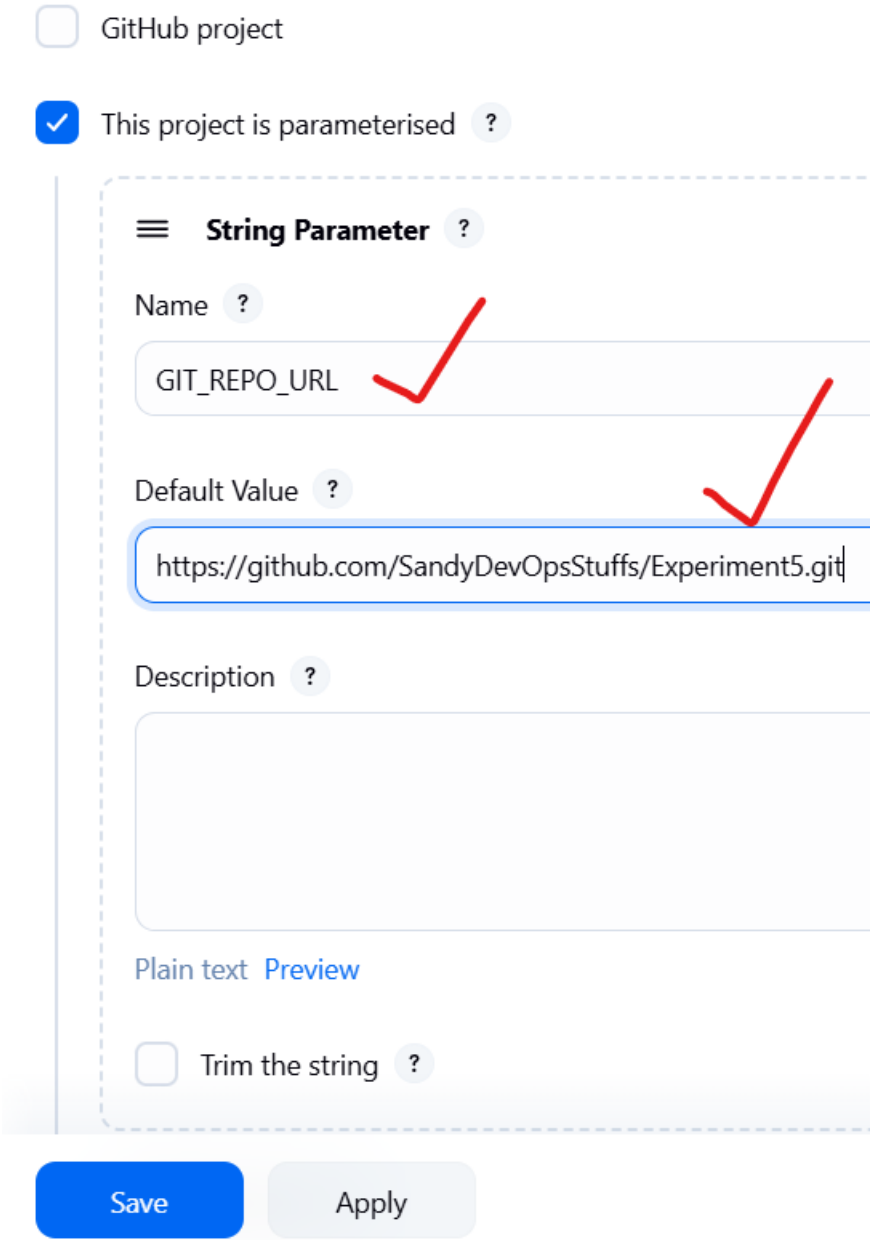


● Add a "String Parameter" named `GIT_REPO_URL` and set its default value to your Git repository URL.

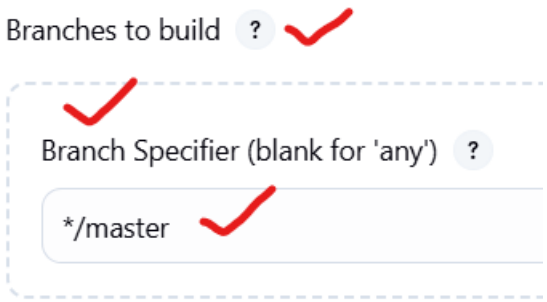
General

Description

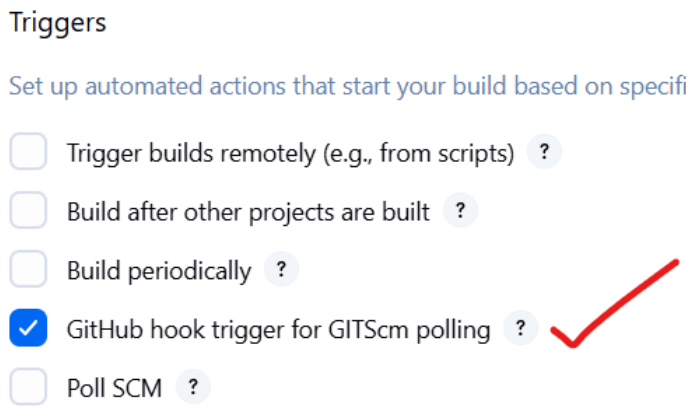




- Scroll down and set `Branches to build` -> `Branch Specifier to the working Git branch` (ex: `*/master`).



● Scroll down to "Triggers" section and select the "`GitHub hook trigger for GITScm polling`" option. This enables Jenkins to listen to GitHub webhook triggers.



Step 4: Configure Build Steps

● Scroll down and go to the "Build Steps" section.

● Add build steps to execute Docker commands for building and deploying the containerized web application. Use the following commands:

Remove the existing container if it exists

```
docker rm --force container1
```

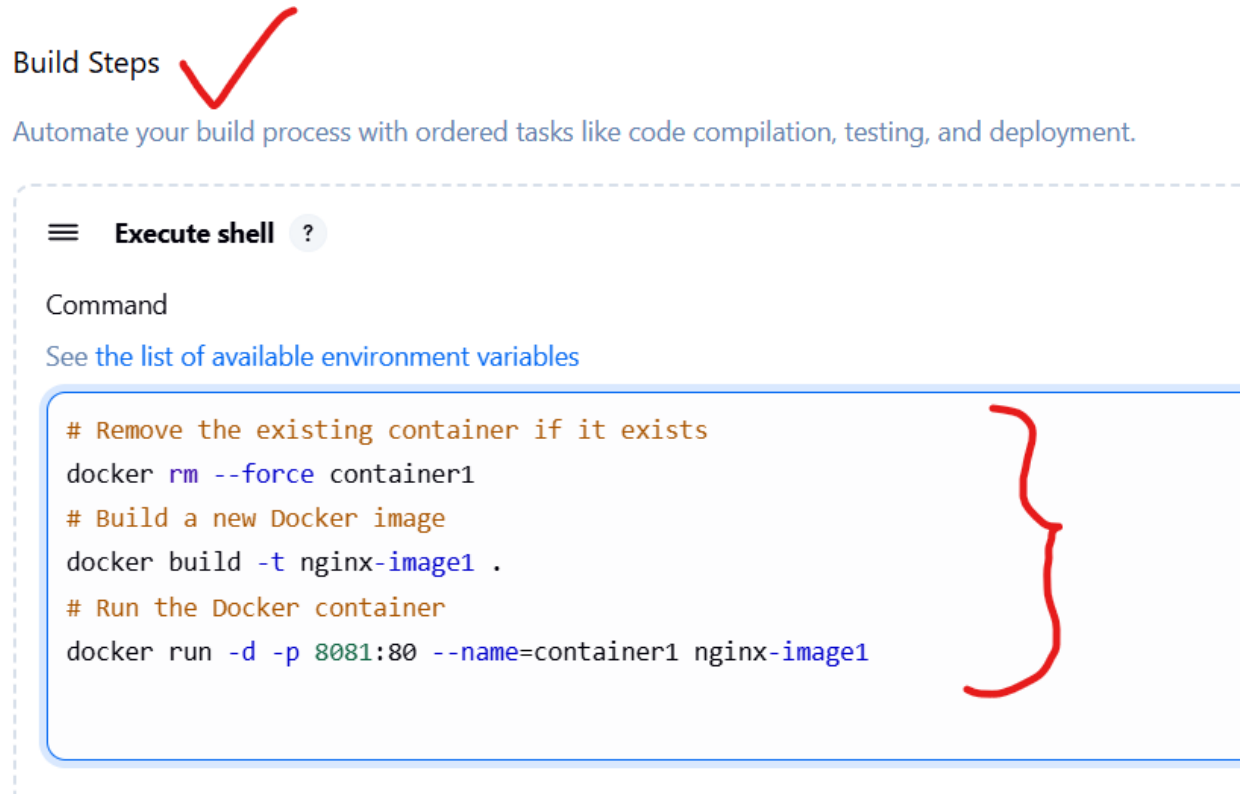
Build a new Docker image

```
docker build -t nginx-image1 .
```

Run the Docker container

```
docker run -d -p 8081:80 --name=container1 nginx-image1
```

● These commands remove the existing container (if any), build a Docker image named "`nginx-image1`," and run a Docker container named "`container1`" on port 8081.



Save and Apply the configuration.

Step 5: Set Up a GitHub Webhook

● In your GitHub repository, navigate to "Settings" and then "Webhooks."

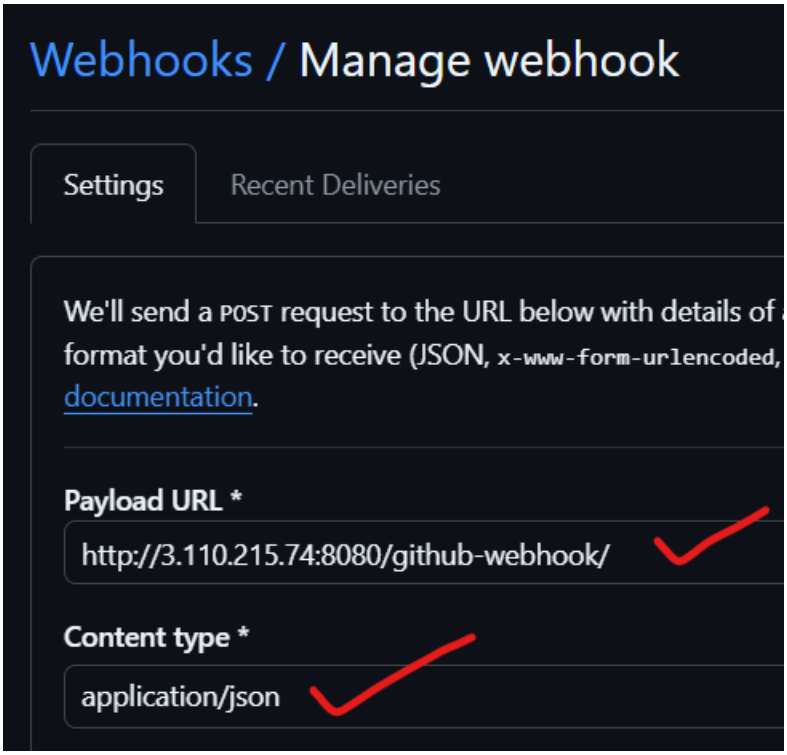
● Create a new webhook and configure it to send a payload to the Jenkins webhook URL.

It is usually http://jenkins-server/github-webhook/

i.e.

http://<Public_IP_of_EC2_Instance>:8080/github-webhook/

Set the content type to "`application/json`"



Click on `Update webhook` at the bottom.

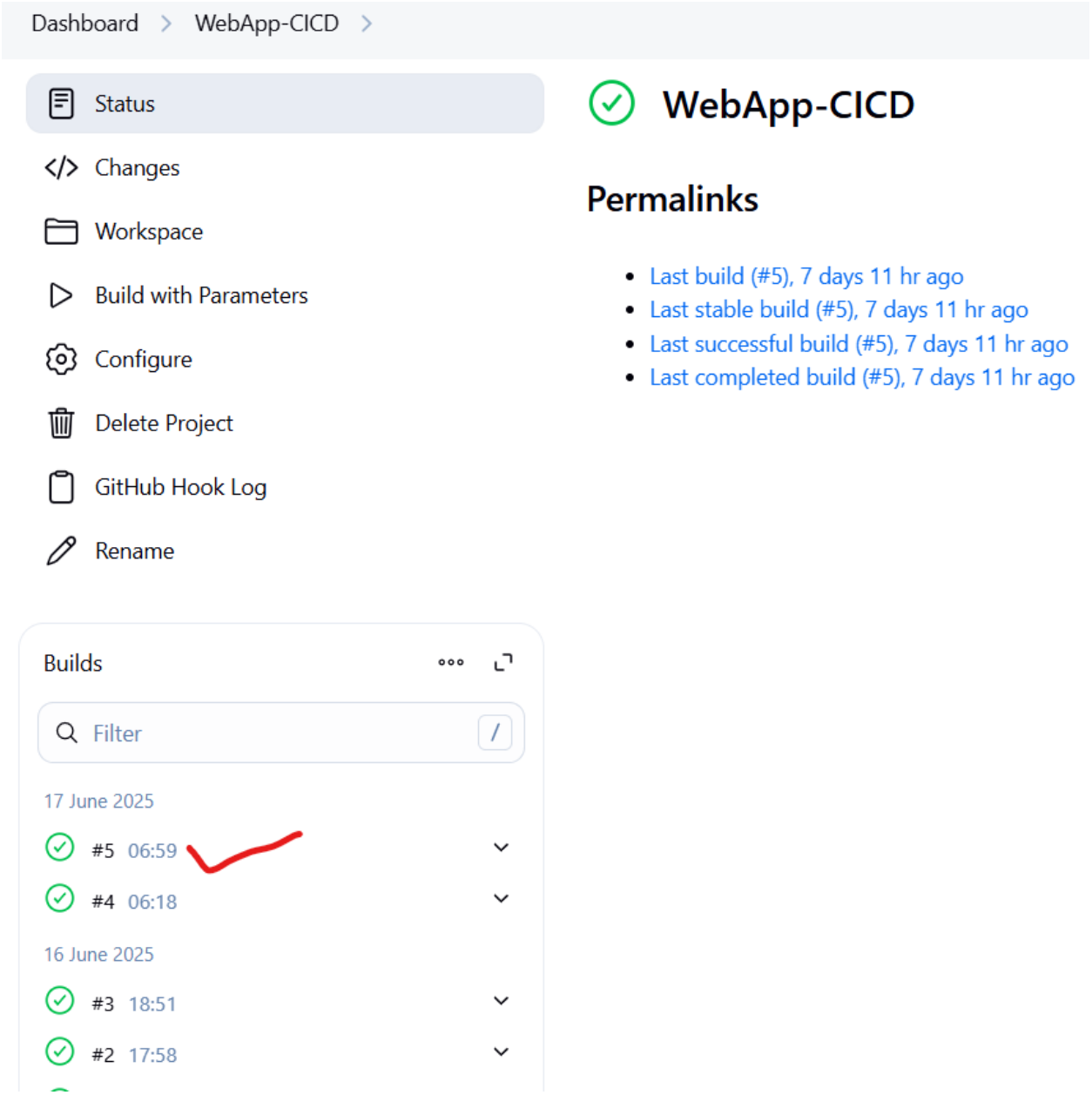
Step 6: Trigger the CI/CD Pipeline

- Now before triggering the pipeline, let us see the current images, containers and job builds in Jenkins.

```
ubuntu@ip-172-31-46-247:~/Experiment5$ docker --version
Docker version 28.2.2, build e6534b4
ubuntu@ip-172-31-46-247:~/Experiment5$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
<none>        <none>    94a4d52c2251   47 hours ago    547MB
hello-world   latest    74cc54e27dc4   5 months ago    10.1kB
ubuntu@ip-172-31-46-247:~/Experiment5$
ubuntu@ip-172-31-46-247:~/Experiment5$
ubuntu@ip-172-31-46-247:~/Experiment5$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
ubuntu@ip-172-31-46-247:~/Experiment5$
```

Right now, there is no image named `nginx-image1` and there is no container named `container1`. But still for the safer side our pipeline will delete the `container1` if it exists.

In Jenkins right now the build is 5th build.



● Now make some changes to the file `index.html` and push the changes as well as the Dockerfile to your GitHub repository. The webhook will trigger the Jenkins job automatically, executing the build and deployment steps defined in the job configuration.

```
ls -la
```

```
cat index.html
```

```
ubuntu@ip-172-31-46-247:~/Experiment5$ ls -la
total 16
drwxrwxr-x 3 ubuntu ubuntu 4096 Jun 17 06:54 .
drwxr-x--- 9 ubuntu ubuntu 4096 Jun 23 04:07 ..
drwxrwxr-x 8 ubuntu ubuntu 4096 Jun 17 06:58 .git
-rw-rw-r-- 1 ubuntu ubuntu   40 Jun 17 06:54 index.html
ubuntu@ip-172-31-46-247:~/Experiment5$
ubuntu@ip-172-31-46-247:~/Experiment5$
ubuntu@ip-172-31-46-247:~/Experiment5$ cat index.html
<h1>Version 2 - Updated via CI/CD!</h1>
ubuntu@ip-172-31-46-247:~/Experiment5$
```

`vi index.html`

Now make some changes, save and exit.

```
ubuntu@ip-172-31-46-247: ~/Experiment5
<h1>Version 3 - Implemented CI/CD to deploy the application in Docker container!</h1>
~
```

`cat index.html`

```
ubuntu@ip-172-31-46-247:~/Experiment5$ cat index.html
<h1>Version 3 - Implemented CI/CD to deploy the application in Docker container!</h1>
ubuntu@ip-172-31-46-247:~/Experiment5$
```

`git status`

```
ubuntu@ip-172-31-46-247:~/Experiment5$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   index.html

no changes added to commit (use "git add" and/or "git commit -a")
ubuntu@ip-172-31-46-247:~/Experiment5$
```

`git add index.html Dockerfile`

`git status`

```
ubuntu@ip-172-31-46-247:~/Experiment5$ git add index.html Dockerfile
ubuntu@ip-172-31-46-247:~/Experiment5$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   Dockerfile
        modified:   index.html

ubuntu@ip-172-31-46-247:~/Experiment5$
```

`git commit -m "Updated Build Steps in Jenkins with docker commands and committed both index.html and Dockerfile"`

```
ubuntu@ip-172-31-46-247:~/Experiment5$ git commit -m "Updated Build Steps in Jenkins with docker commands and committed both index.html and Dockerfile"
[master 137fc27] Updated Build Steps in Jenkins with docker commands and committed both index.html and Dockerfile
 Committer: Ubuntu <ubuntu@ip-172-31-46-247.ap-south-1.compute.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 2 files changed, 4 insertions(+), 1 deletion(-)
 create mode 100644 Dockerfile
ubuntu@ip-172-31-46-247:~/Experiment5$
```

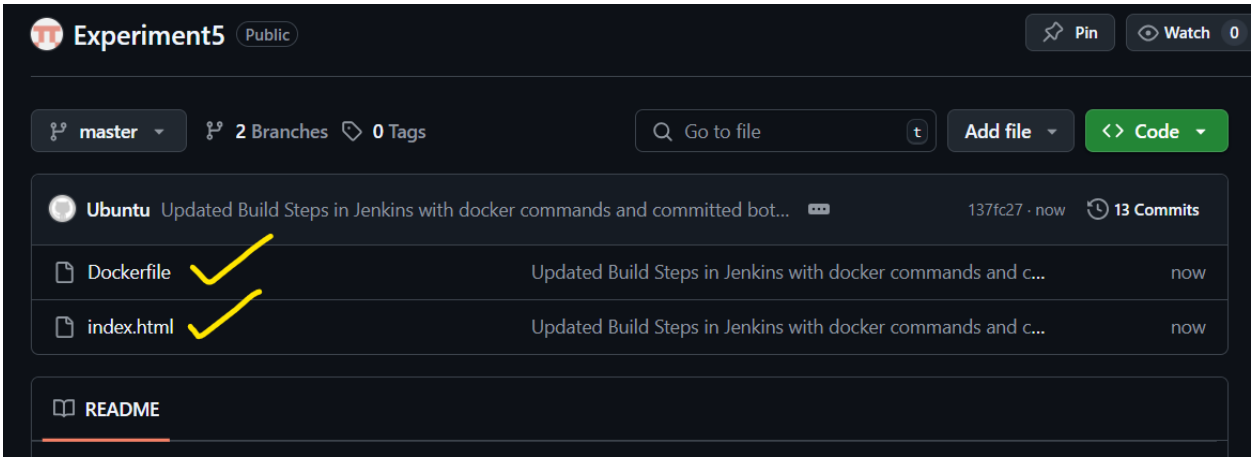
`git status`

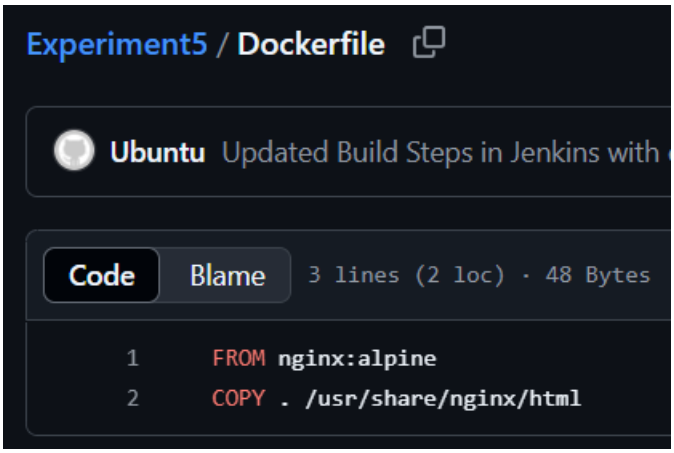
```
ubuntu@ip-172-31-46-247:~/Experiment5$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
ubuntu@ip-172-31-46-247:~/Experiment5$
```

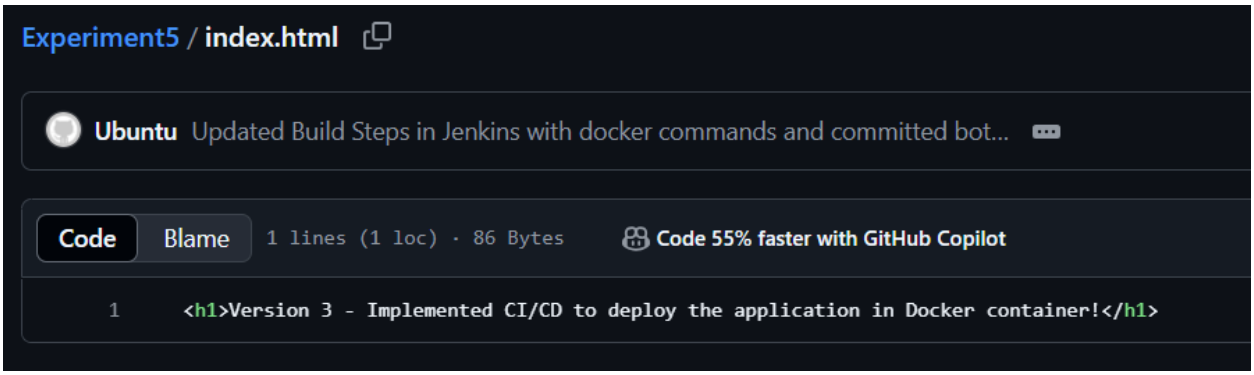
`git push origin master`

```
ubuntu@ip-172-31-46-247:~/Experiment5$ git push origin master
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 2 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 492 bytes | 492.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:SandyDevOpsStuffs/Experiment5.git
   e856ad8..137fc27  master -> master
ubuntu@ip-172-31-46-247:~/Experiment5$
```

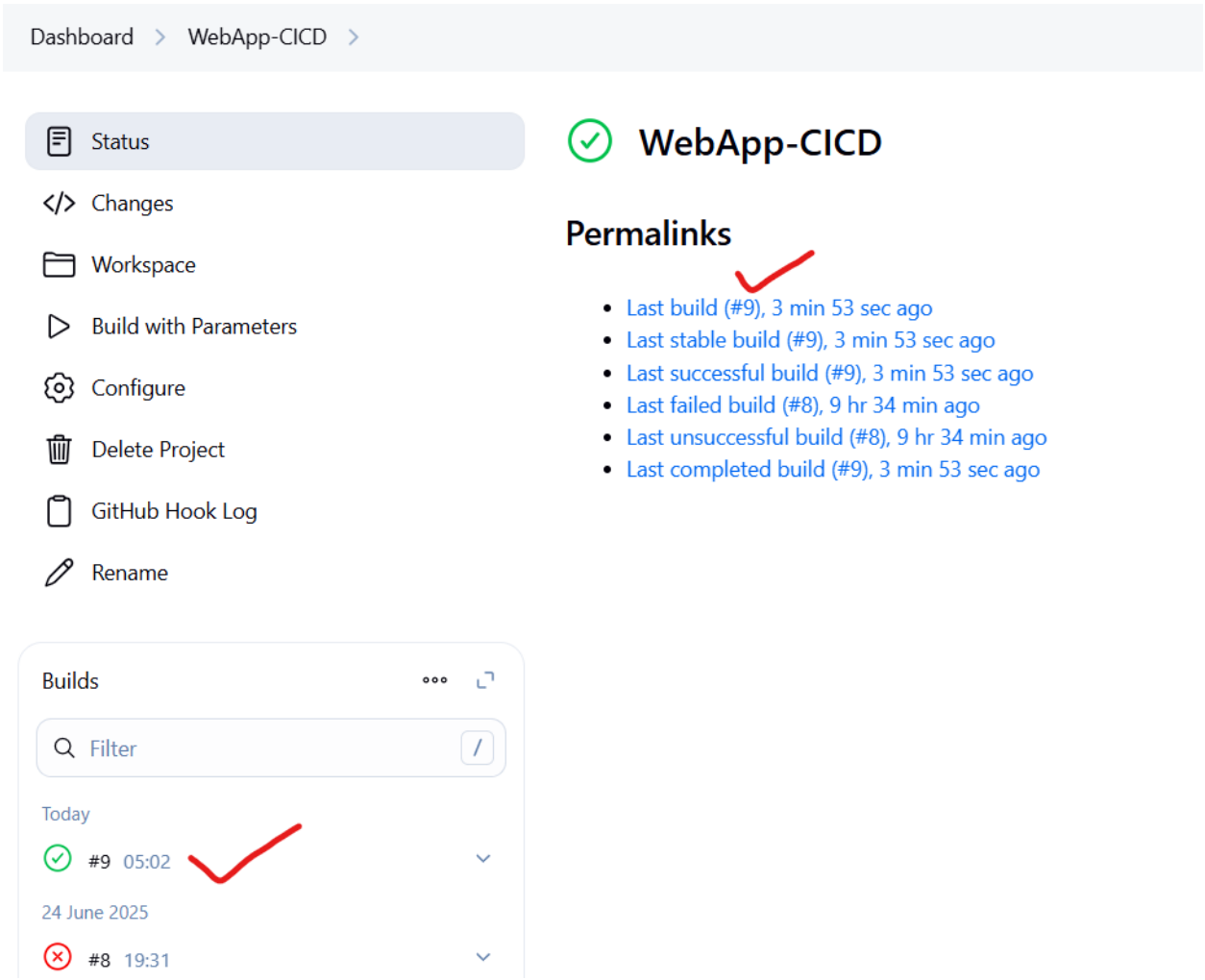
- Check the code changes in GitHub.







- Monitor the Jenkins job's progress in the Jenkins web interface.



Console Output

Download Copy View as plain

```
Started by GitHub push by SandyDevOpsStuffs
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/WebApp-CICD
The recommended git tool is: NONE
No credentials specified
 > git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/WebApp-CICD/.git # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url https://github.com/SandyDevOpsStuffs/Experiment5.git # timeout=10
Fetching upstream changes from https://github.com/SandyDevOpsStuffs/Experiment5.git
 > git --version # timeout=10
 > git --version # 'git version 2.43.0'
 > git fetch --tags --force --progress -- https://github.com/SandyDevOpsStuffs/Experiment5.git +refs/heads/*:refs/remotes/origin/* #
timeout=10
 > git rev-parse refs/remotes/origin/master^{commit} # timeout=10
Checking out Revision 030f1f953083105026b4e718d067c191ee26891b (refs/remotes/origin/master)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f 030f1f953083105026b4e718d067c191ee26891b # timeout=10
Commit message: "Updated Build Steps in Jenkins with docker commands and committed both index.html and Dockerfile"
 > git rev-list --no-walk e856ad89fc25d39f299e73d9d349304577442325 # timeout=10
[WebApp-CICD] $ /bin/sh -xe /tmp/jenkins12386123177297800118.sh
+ docker rm --force container1
```

```
Error response from daemon: No such container: container1
+ docker build -t nginx-image1 .
#0 building with "default" instance using docker driver

#1 [internal] load build definition from Dockerfile
#1 transferring dockerfile: 85B done
#1 DONE 0.0s

#2 [internal] load metadata for docker.io/library/nginx:alpine
#2 DONE 2.5s

#3 [internal] load .dockerignore
#3 transferring context: 2B done
#3 DONE 0.0s

#4 [1/2] FROM docker.io/library/nginx:alpine@sha256:b2e814d28359e77bd0aa5fed1939620075e4ffa0eb20423cc557b375bd5c14ad
#4 resolve docker.io/library/nginx:alpine@sha256:b2e814d28359e77bd0aa5fed1939620075e4ffa0eb20423cc557b375bd5c14ad 0.0s done
#4 ...

#5 [internal] load build context
#5 transferring context: 44.67kB 0.0s done
#5 DONE 0.1s

#4 [1/2] FROM docker.io/library/nginx:alpine@sha256:b2e814d28359e77bd0aa5fed1939620075e4ffa0eb20423cc557b375bd5c14ad
#4 sha256:b2e814d28359e77bd0aa5fed1939620075e4ffa0eb20423cc557b375bd5c14ad 10.33kB / 10.33kB done
#4 sha256:6544c26a789f03b1a36e45ce8c77ea71d5d3e8d4e07c49ddceccfe0de47aa3e0 2.50kB / 2.50kB done
#4 sha256:77656422f7009bf47cd4992cb1a118af09c25bb157336478bdbf399af09e8e41 10.78kB / 10.78kB done
#4 sha256:fe07684b16b82247c3539ed86a65ff37a76138ec25d380bd80c869a1a4c73236 0B / 3.80MB 0.2s
#4 DONE 2.8s

#6 [2/2] COPY . /usr/share/nginx/html
#6 DONE 0.2s

#7 exporting to image
#7 exporting layers 0.1s done
#7 writing image sha256:4c5df2b76147210486b948799b01886725223ee8d715c46421a710d21c6d5a83 done
#7 naming to docker.io/library/nginx-image1 0.0s done
#7 DONE 0.1s
+ docker run -d -p 8081:80 --name=container1 nginx-image1
87434ad44d137c2af89d398fb18caba9a8de38a54e5f76b9cad0cc4a40b2b9fc
Finished: SUCCESS
```

Step 7: Verify the Deployment

`docker images`

```
ubuntu@ip-172-31-46-247:~/Experiment5$ docker images
REPOSITORY     TAG       IMAGE ID       CREATED          SIZE
nginx-image1   latest    4c5df2b76147   6 minutes ago    52.5MB
<none>         <none>    94a4d52c2251   2 days ago       547MB
hello-world    latest    74cc54e27dc4   5 months ago     10.1kB
ubuntu@ip-172-31-46-247:~/Experiment5$
```

`docker ps`

```
ubuntu@ip-172-31-46-247:~/Experiment5$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS                                       NAMES
87434ad44d13   nginx-image1   "/docker-entrypoint.…"   8 minutes ago   Up 8 minutes   0.0.0.0:8081->80/tcp, [::]:8081->80/tcp   container1
ubuntu@ip-172-31-46-247:~/Experiment5$
```

Access your web application by opening a web browser and navigating to http://localhost:8081 (or the appropriate URL if hosted elsewhere like http://<Public_IP_of_EC2Instance>:8081).

Version 3 - Implemented CI/CD to deploy the application in Docker container!

Hu hoo...! Here we go!! We deployed the application successfully onto the Docker container and it is running! , Awesome! .

NOTE:

- *CI part*
- *CD part*
- *Hard coding of repo URL and alternatives for it.*
- *Freestyle (Less structured, no groovy) vs Pipeline (groovy, more structured, Declarative vs Scripted)*

Declarative syntax:

```
pipeline {
  agent any
  stages {
    stage('Build') {
      steps {
        echo 'Building...'
      }
    }
  }
}
```

Scripted syntax:

```
node {
  stage('Build') {
    echo 'Building...'
  }
}
```

Conclusion:

This experiment demonstrates how to apply CI/CD principles to web development using Jenkins, Git, Docker containers, and GitHub webhooks. By configuring Jenkins to listen for GitHub webhook triggers and executing Docker commands in response to code changes, you can automate the build and deployment of your web application, ensuring a more efficient and reliable development workflow.

Exercise / Questions :

1. Explain the core principles of Continuous Integration (CI) and Continuous Deployment (CD) in the context of web development. How do these practices enhance the software development lifecycle?

2. Discuss the key differences between Continuous Integration and Continuous Deployment. When might you choose to implement one over the other in a web development project?

3. Describe the role of automation in CI/CD. How do CI/CD pipelines automate code integration, testing, and deployment processes?

4. Explain the concept of a CI/CD pipeline in web development. What are the typical stages or steps in a CI/CD pipeline, and why are they important?

5. Discuss the benefits of CI/CD for web development teams. How does CI/CD impact the speed, quality, and reliability of software delivery?

6. What role do version control systems like Git play in CI/CD workflows for web development? How does version control contribute to collaboration and automation?

7. Examine the challenges and potential risks associated with implementing CI/CD in web development. How can these challenges be mitigated?

8. Provide examples of popular CI/CD tools and platforms used in web development. How do these tools facilitate the implementation of CI/CD principles?

9. Explain the concept of "Infrastructure as Code" (IaC) and its relevance to CI/CD. How can IaC be used to automate infrastructure provisioning in web development projects?

10. Discuss the cultural and organisational changes that may be necessary when adopting CI/CD practices in a web development team. How does CI/CD align with DevOps principles and culture?