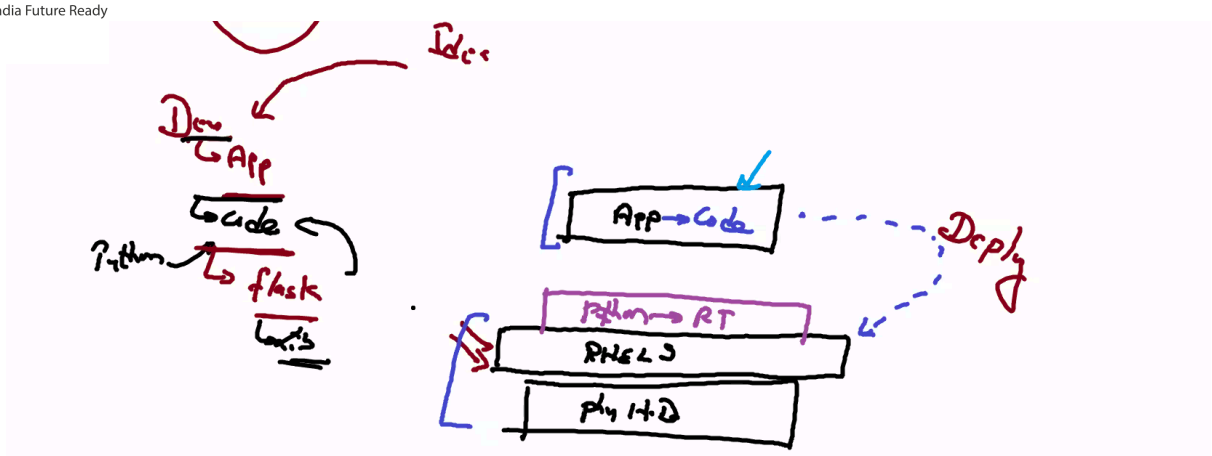
7 Days DevOps Deloitte Session 3

Summary 04-01-2025

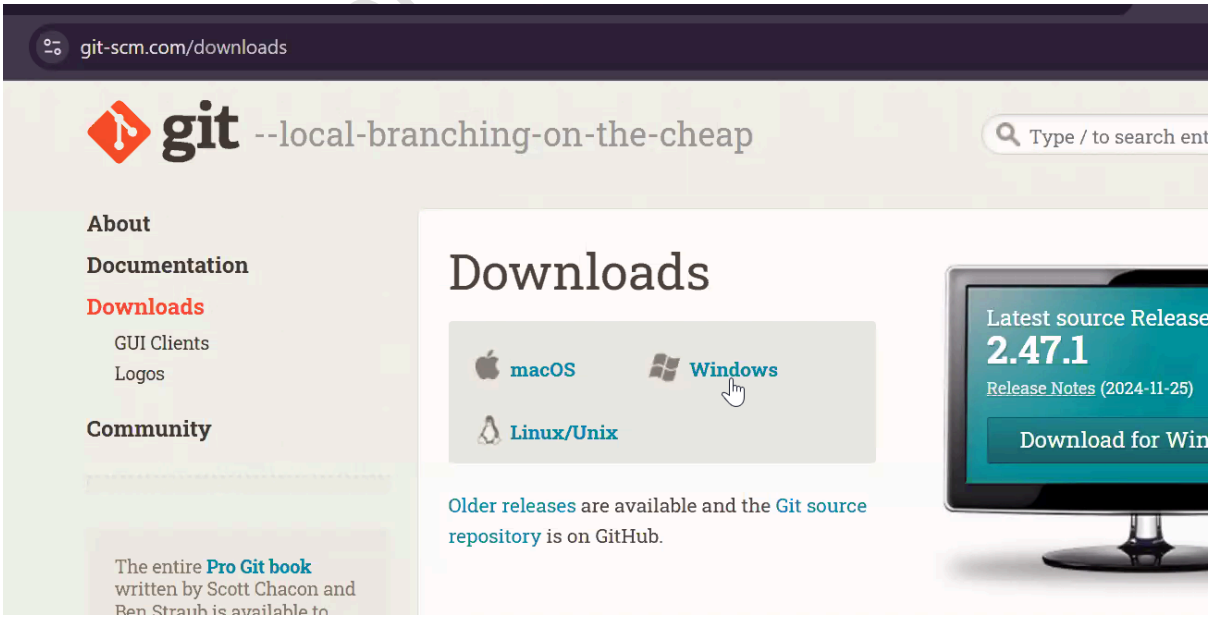
- If you don't have the required Docker images or container images, it's impossible to launch the operating system. For example, if the `cal` command is not included in the image, the `cal` command will not work in your container.
- Therefore, you need to think carefully about which programs you require. Based on your use case, you should include those programs in your image. When using images from Docker Hub, we don't use them directly; instead, we create our own custom images. After creating the image, you can use it to launch your own container. Creating an image is also known as "building" an image.
- Before creating an image, it's essential to have a roadmap outlining what you want in your image. For instance, if you need Red Hat version 9, you should remember that we don't use the operating system directly; we use the programs installed on it. So, in Red Hat, if you need a specific application, you have to copy that application into your operating system. When you copy and execute the application (i.e., run it), it's technically referred to as "deployment."
- Additionally, if your application is written in Python, you'll need a runtime environment for that programming language. Most applications also require several libraries; for example, in Python, you might use Flask, Django, and others. These libraries must also be included in your image.



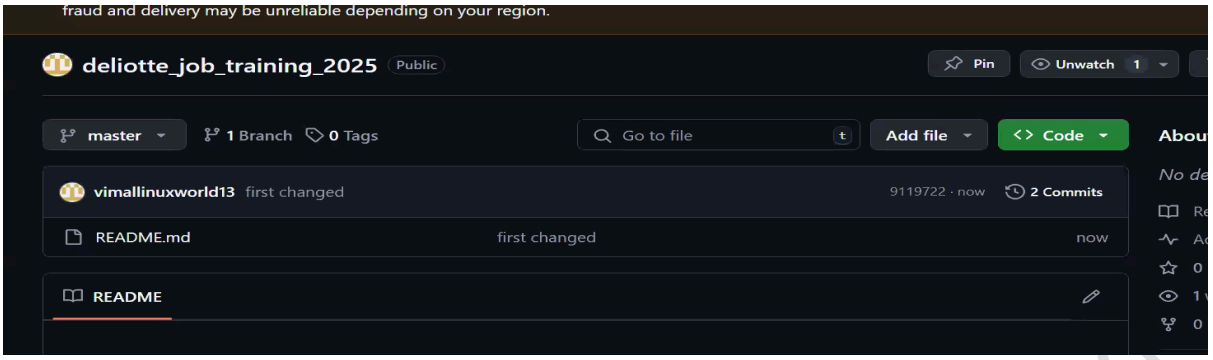
- Until we have this setup, the application will not run. Therefore, we need this environment. If the developer used Python version 3.9, then we need version 3.9 in our container.

- To create a custom image, we use a Dockerfile where we write the code to define the image. When you run the Dockerfile (i.e., compile this file), your Docker image is created. You can then use this image on other platforms that support container technology, such as ECS, EKS, AKS, GKE, Kubernetes, or any similar platform.

- Typically, developers share their code using a Git-based SCM system, like GitHub. You can use Git Bash software to get Git commands on your Windows system.



- You can upload the code on github using the git.



- Now we are going to create a function in Python. Whenever anyone calls this function, it will print Welcome to LW.

```
def lw():
    return "Welcome to LW...."
~
```

- But we are not gone call the function directly its going call with Webapp so we can create a route using the Flask library.

```
MINGW64:/c/Users/Vimal Daga/Documents/deliotte_training_2025
from flask import Flask

app = Flask(__name__)


@app.route("/info")
def lw():
    return "Welcome to LW...."

app.run(host='0.0.0.0')
```

- After writing the code we mostly upload into github we can use git command to add this code in github.

```
Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/deliotte_training_2025 (master)
$ git add app.py
warning: LF will be replaced by CRLF in app.py.
The file will have its original line endings in your working directory

Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/deliotte_training_2025 (master)
$ git commit app.py -m "secchange"
warning: LF will be replaced by CRLF in app.py.
The file will have its original line endings in your working directory
[master 469607a] secchange
 1 file changed, 17 insertions(+)
 create mode 100644 app.py

Vimal Daga@DESKTOP-3E1AGGT MINGW64 ~/Documents/deliotte_training_2025 (master)
$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 387 bytes | 129.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/vimallinuxworld13/deliotte_job_training_2025.git
   9119722..469607a  master -> master
```

- Our code is uploaded on github.

- Now, anyone who wants to use this code can pull it. The person who uploads the code can use the `git push` command, and for downloading the code, we can use the `git pull` or `git clone` command.
- If you want to return to your base OS (Docker host) without shutting down the container, you can use the shortcut `Ctrl + P + Q`.
- To download the code, you can use the `git clone` command.

```
[root@ip-172-31-37-40 ~]#
[root@ip-172-31-37-40 ~]# git clone https://github.com/vimallinuxworld13/deliotte_job_training_2025.git
```

- And our code is downloaded into our host computer.

```
[root@ip-172-31-37-40 ~]# ls
deliotte_job_training_2025  jibbran
[root@ip-172-31-37-40 ~]# cd deliotte_job_training_2025/
[root@ip-172-31-37-40 deliotte_job_training_2025]# ls
README.md  app.py
[root@ip-172-31-37-40 deliotte_job_training_2025]#
```

- We are going to use this code in a container. If you want to copy a file from the base OS to the container, you can use the `docker cp` command.

```
[root@ip-172-31-37-40 deliotte_job_training_2025]# docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED          STATUS          PORTS     NAMES
62d5784940f7   redhat/ubi8                    "/bin/bash"              18 minutes ago   Up 18 minutes             agitated_ar
yabhata
be436543ac21   vimal13/apache-webserver-php   "/usr/sbin/httpd -DF…"   24 hours ago     Up 24 hours     80/tcp    competent_k
hayyam
86eb1ad03984   ubuntu:14.04                   "/bin/bash"              25 hours ago     Up 25 hours               blissful_cr
ay
[root@ip-172-31-37-40 deliotte_job_training_2025]# docker cp  app.py  62d5784940f7:/
Successfully copied 2.05kB to 62d5784940f7:/
[root@ip-172-31-37-40 deliotte_job_training_2025]#
```

- If you want to run any command in running container then we can use exec command.
- We want to take the terminal so we can use bash command to get the terminal.

```
[root@ip-172-31-37-40 deliotte_job_training_2025]# docker exec  -it 62d5784940f7   bash
[root@62d5784940f7 /]#
[root@62d5784940f7 /]#
[root@62d5784940f7 /]#
```

- Now we can use python command to start our python application.

```
[root@62d5784940f7 /]# ls
app.py  bin  boot  dev  etc  home  lib  lib64  lost+found  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
[root@62d5784940f7 /]# python3  app.py
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.4:5000
Press CTRL+C to quit
```

- In networking, if a computer has a private IP, no one can connect to it directly from the outside. If you want others to connect to it, you need to set up a public IP.
- Cloud providers also offer public IPs. If you check the IP address of a container, it typically has a private IP address, meaning it cannot be accessed from outside.

- To interact with the container, you can connect to it from the base OS. If you ping the container, you will see that you can access the web app.

```
[root@ip-172-31-37-40 ~]#
[root@ip-172-31-37-40 ~]# curl http://172.17.0.4:5000/info
Welcome to LW....[root@ip-172-31-37-40 ~]#
```

- Currently, we are doing this manually, but we can use Ansible to automate configuration management. Competitors of Ansible include Ceph and Puppet.
- Building the image is a manual process, but now we are going to use a Dockerfile to automate the process of creating the image.
- In a Dockerfile, we use the `FROM` keyword to specify the base OS and the `RUN` keyword to execute commands.

```
FROM  redhat/ubi8

RUN yum install python39
```

- If you want to copy anything, the `COPY` command is available. To specify which port your app is currently using, you can use the `EXPOSE` keyword.
- Remember, in a Dockerfile, you should always use non-interactive commands. If you include interactive commands, the process will fail.

```
FROM  redhat/ubi8

RUN yum install python39 -y

EXPOSE 5000

RUN pip3 install flask
```

- When you are building the image its know as build time so run command and other copy keyword run at build time but if you want to run any command at run time then we have entrypoint keyword.

```
FROM  redhat/ubi8

RUN yum install python39 -y

EXPOSE 5000

RUN pip3 install flask

COPY app.py  /app.py

ENTRYPOINT [ "python3", "/app.py" ]
```

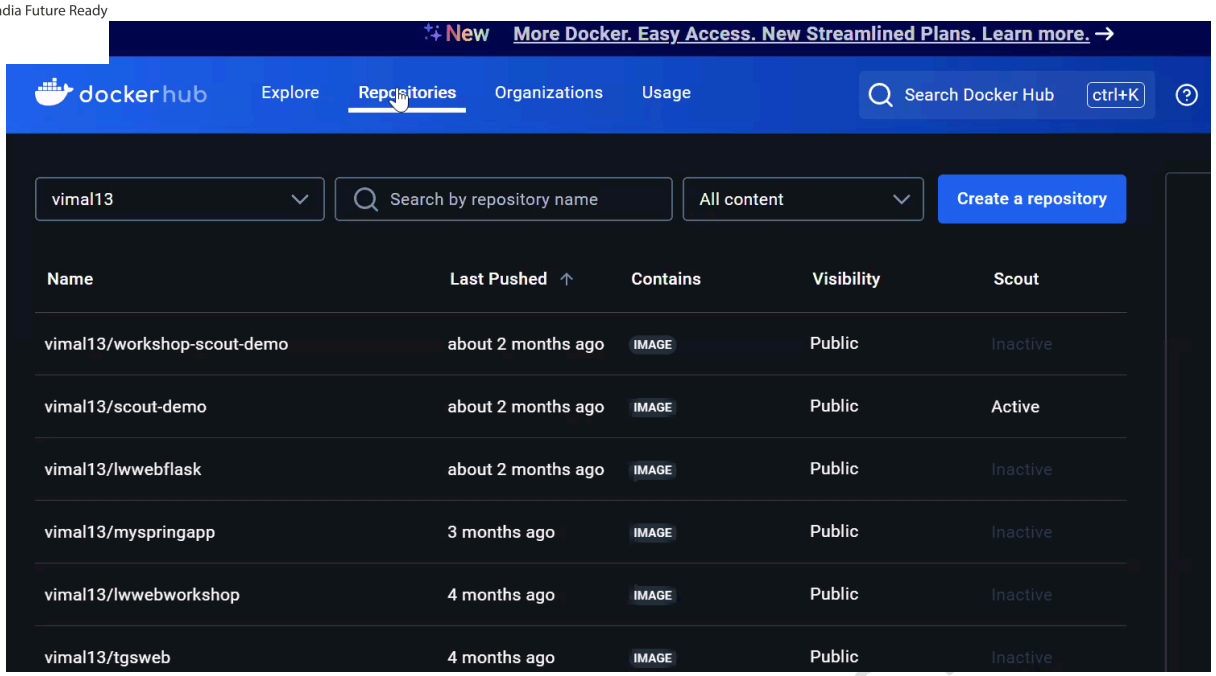
- In the `ENTRYPOINT` instruction, commands are provided in list format.
- You can use the `docker build` command to build the image, and the `-t` option is used to assign a tag to the image.

```
[root@ip-172-31-37-40 deliotte_job_training_2025]# docker build -t  myvimalweb:v1  .
[+] Building 4.4s (5/8)                                                                    docker:default
 => [internal] load build definition from Dockerfile                                                0.1s
 => => transferring dockerfile: 248B                                                                0.0s
 => [internal] load metadata for docker.io/redhat/ubi8:latest                                       0.0s
 => [internal] load .dockerignore                                                                   0.0s
 => => transferring context: 2B                                                                     0.0s
 => [1/4] FROM docker.io/redhat/ubi8:latest                                                         0.0s
 => [internal] load build context                                                                   0.0s
 => => transferring context: 237B                                                                   0.0s
 => [2/4] RUN yum install python39 -y                                                               4.3s
 => => # Running transaction test
 => => # Transaction test succeeded.
 => => # Running transaction
 => => #   Preparing        :                                                          1/1
 => => #   Installing       : python39-setuptools-wheel-50.3.2-6.module+el8.10.0+2   1/6
 => => #   Installing       : python39-pip-wheel-20.2.4-9.module+el8.10.0+21329+8d   2/6
```

- Once the image is created, you can use it anywhere you want. These images are known as OCI images.
- Now, you can launch the container with the image and access the application using the `curl` command.

```
[root@ip-172-31-37-40 deliotte_job_training_2025]# curl http://172.17.0.5:5000/info
Welcome to LW....[root@ip-172-31-37-40 deliotte_job_training_2025]#
```

- This is one of the big reasons why we need to create our custom image. Now, if you want to use this image in Kubernetes, we have to upload it to Docker Hub. There are thousands of accounts on Docker Hub, so there is a rule that every image name must start with your account name.



- So when using docker build command always give the image name starting with your username.

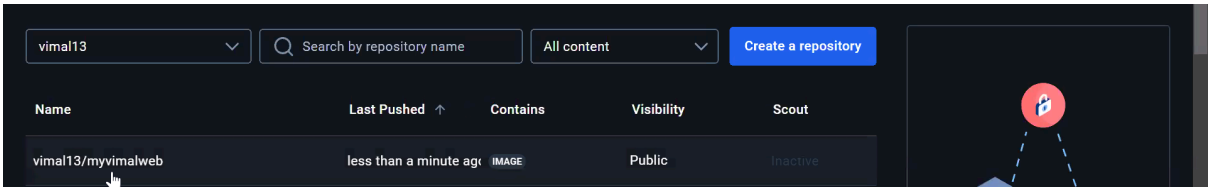
```
[root@ip-172-31-37-40 deliotte_job_training_2025]# docker build -t  vimal13/myvimalweb:v1  .
```

- Now if you want to upload in docker hub then we have docker push command available.

```
[root@ip-172-31-37-40 deliotte_job_training_2025]# docker push  vimal13/myvimalweb:v1
The push refers to repository [docker.io/vimal13/myvimalweb]
218ae178102b: Preparing
d844e1189a8e: Preparing
fbeea1d95fd4: Preparing
452b32d0ddb3: Preparing
denied: requested access to the resource is denied
[root@ip-172-31-37-40 deliotte_job_training_2025]#
```

- But first you have to authenticate to your docker hub so we have docker login command after login you can push to docker hub.

```
[root@ip-172-31-37-40 deliotte_job_training_2025]# docker push  vimal13/myvimalweb:v1
The push refers to repository [docker.io/vimal13/myvimalweb]
218ae178102b: Pushing [==================================================>]  2.048kB
d844e1189a8e: Pushing [>                                                  ]  68.1kB/4.915MB
fbeea1d95fd4: Preparing
452b32d0ddb3: Preparing
```



- We have some ideas, and to bring these ideas to the real world, we create an app with the help of developers. They write the code and build the app. For a single idea, we don't work alone; we have a team. To manage this collaboration, we use an SCM (Source Code Management) system to store our code. In AWS, we have **AWS CodeCommit** to store our code.

- Now, the steps for handling the code depend on the programming language. For example, if the code is written in Java, we can't run it directly. First, we need to compile it into an object file, which means we have to **build** the code. For this, we have systems that take the code from the SCM and build it. Building means taking the source code, compiling it, and creating a package, similar to how a Dockerfile works.

- In companies, there are multiple teams, such as testing, development, and operations, and they often lack coordination. This lack of coordination is called **silos**. One of the goals of the **DevOps methodology** is to eliminate silos, ensuring that teams work together seamlessly. DevOps encourages developers to collaborate with operations teams, and this is where the term **DevOps** comes into play.

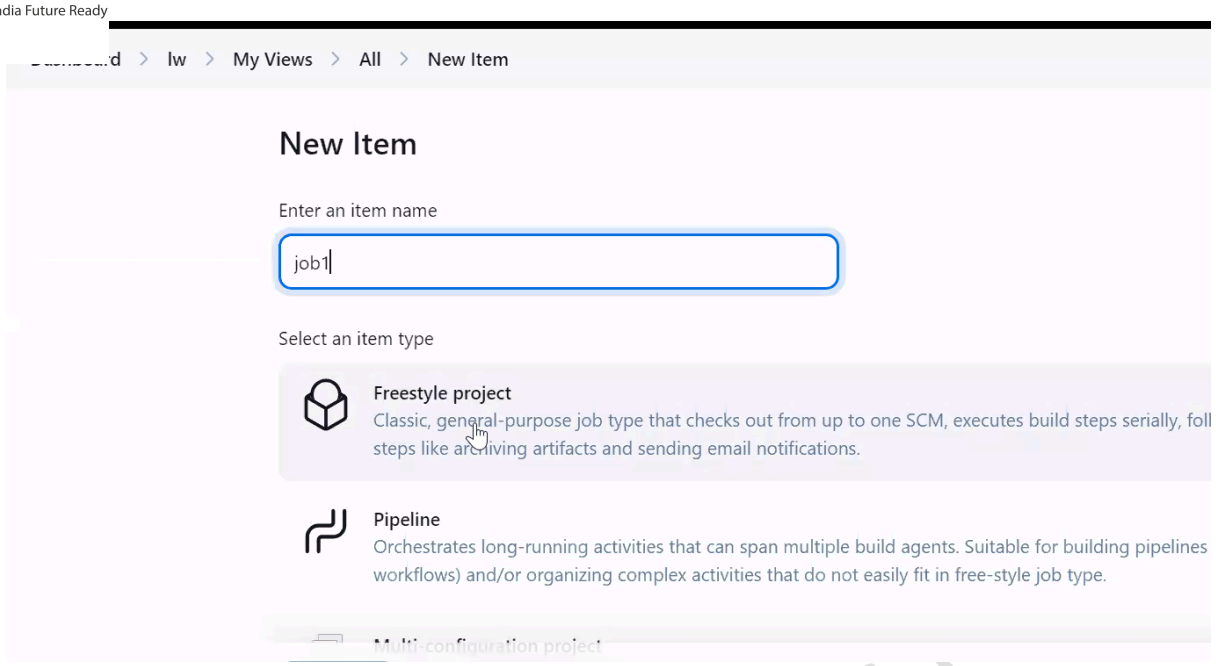
- For example, when creating a Dockerfile, we also have an `app.py` file and the Dockerfile itself. When we push this to the SCM, the build team runs it and builds the image. The system that builds the code is known as a **build server**. Once the image is created, it needs to be stored. If it's a Docker image, we can use Docker Hub or a private image repository. If it's a package, it's referred to as an **artifact**, and tools like **Artifactory** or **Nexus Sonatype** can be used to store it.

- To build code in a specific language, we use language-specific tools. For example, for Java, we use **Maven**. However, once an image is created, there's no guarantee that it works as intended. Another team then runs the container with the image and tests the application. When testing the load or performing other types of testing, this is referred to as **chaos engineering**.

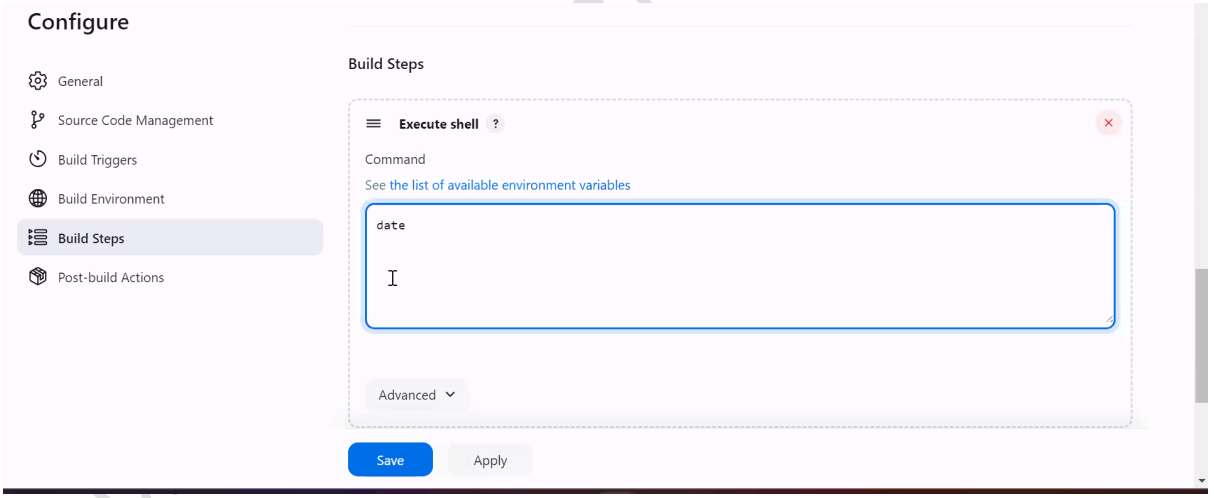
- In the development environment, test cases often fail multiple times. If a test fails, a report is sent to the developers, who review it, update the code, and push the changes again. The build team downloads the updated code, builds it again, and the testing team repeats their process. This

continuous cycle of development, building, and testing is known as **continuous integration (CI)**.

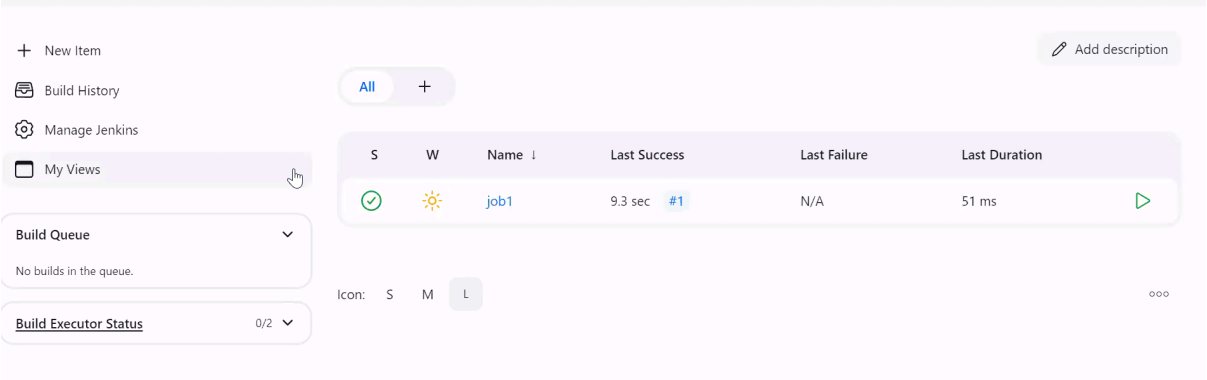
- While this process is often manual, it can be automated using CI tools like **Jenkins**, **CircleCI**, **TeamCity**, **GitLab**, or **GitHub Actions**. AWS also provides services for DevOps. Automating this process creates a **pipeline**, where each step is a **stage**, such as the test stage or build stage.
- If the testing team approves the code or image, it is deployed to the production server, where clients can connect. You can use either Docker or Kubernetes for this, though Kubernetes is more commonly used.
- Between the testing and production teams, there is often a QA team. This team tests the quality, security, and stability of the code and ensures there are no bugs before approving it for production. Once testing is complete and the code is deployed to production, this is called a **release**. The first release is known as **v1**, and subsequent releases are numbered incrementally. This is a continuous process for the lifetime of the application or company.
- This is known as **continuous deployment (CD)**. In CD, there are two terms: **continuous delivery** and **continuous deployment**. In **continuous delivery**, the approval stage is manual, while in **continuous deployment**, the approval is automated.
- In Jenkins, each process in the pipeline is called a **job**, and all these processes collectively represent CI/CD.



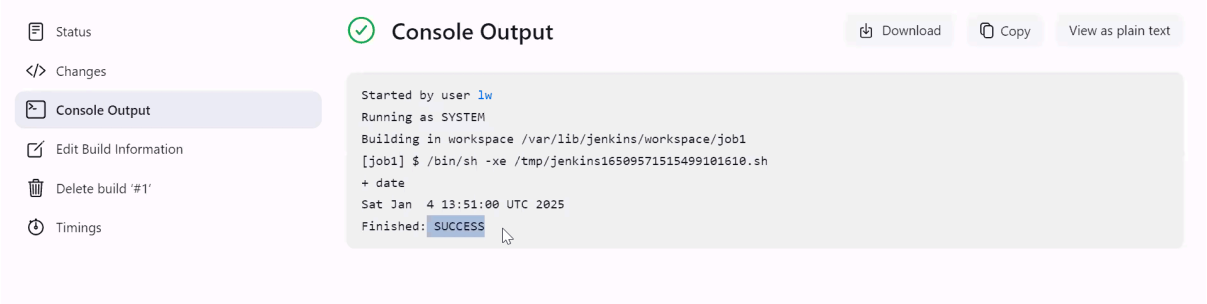
- For example, if you want to run any command, you can add it to the build steps. It will execute that command on the system where Jenkins is running.



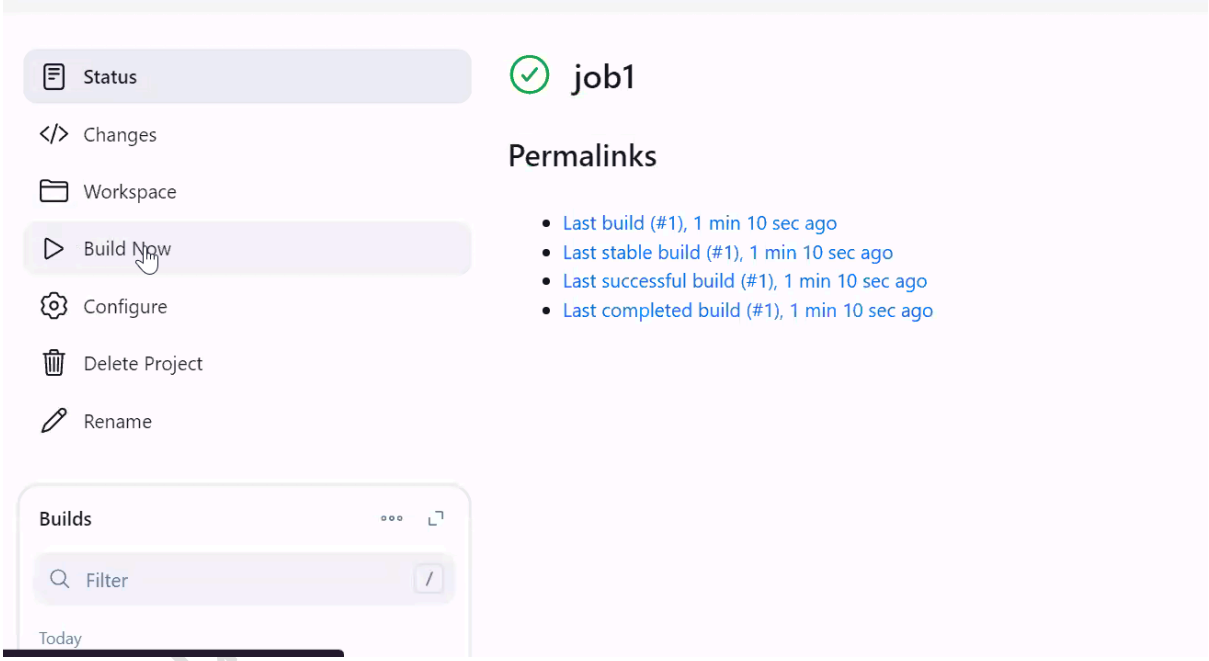
- You can run the job from here after run it will give the output.



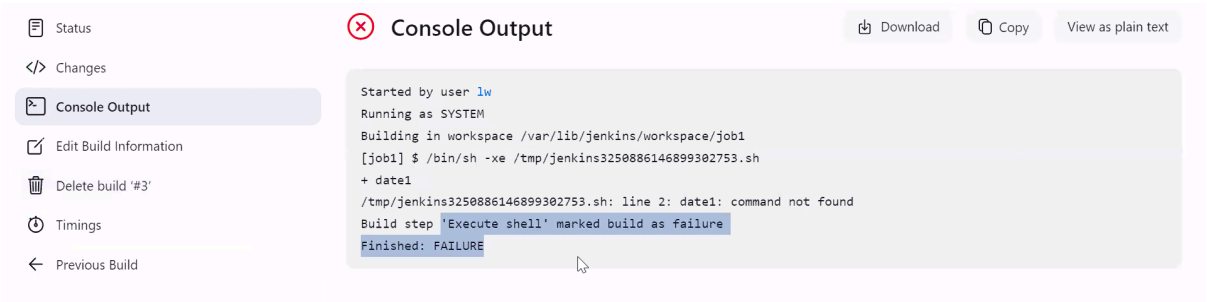
- To check the output of job go to ths job and every job has there id and to check the output of the data command click on id in console output. it will show the output.



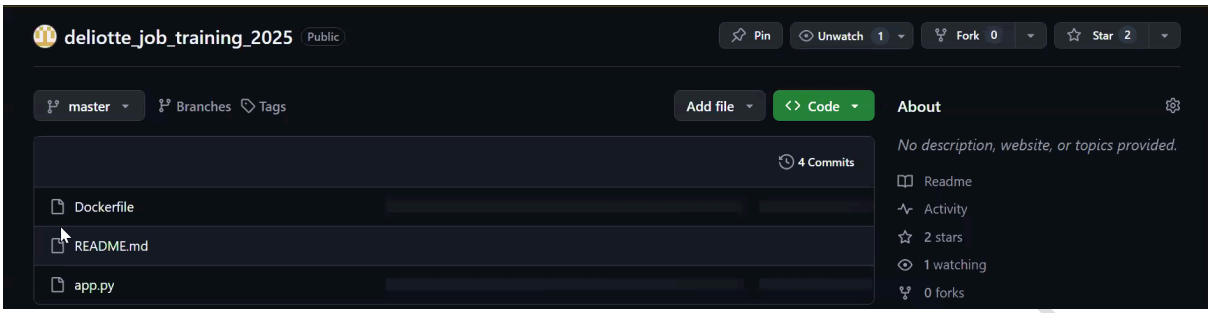
- If you want to run the job again the you can click on build now.



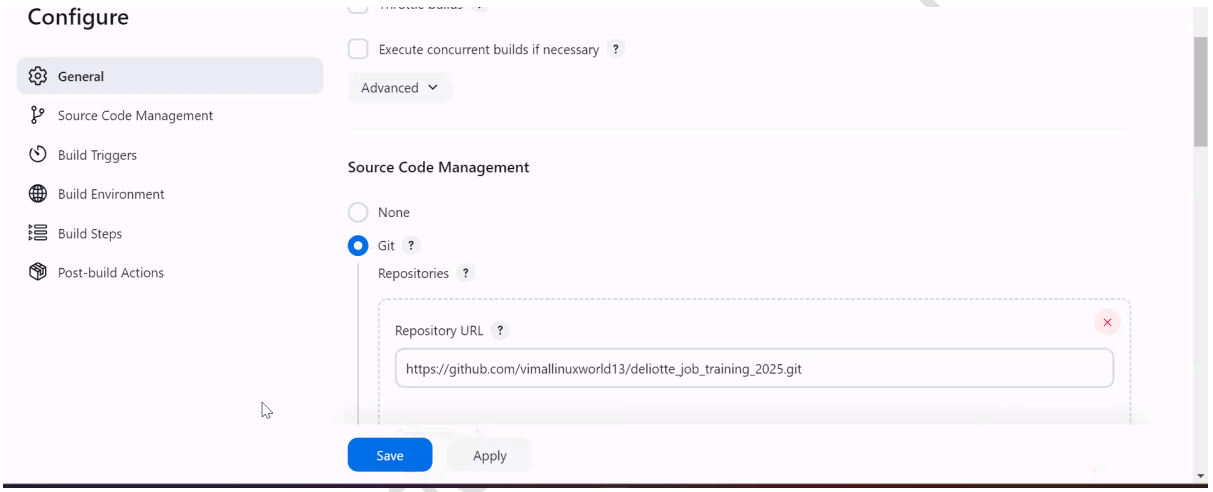
- If you want to configure any thing in job you can configure and if you add the any random command so it will fail.



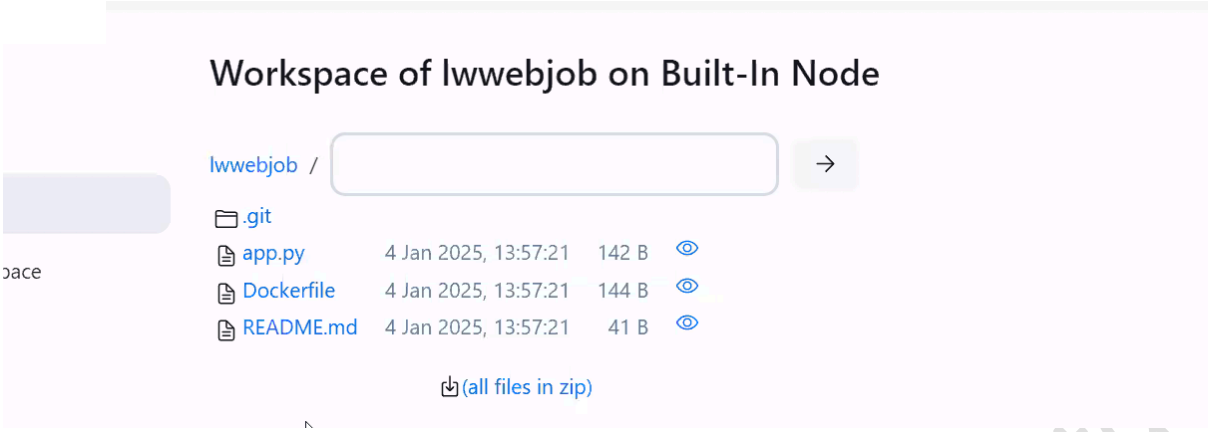
- But our job is what we want that our job go to the github and download the code.



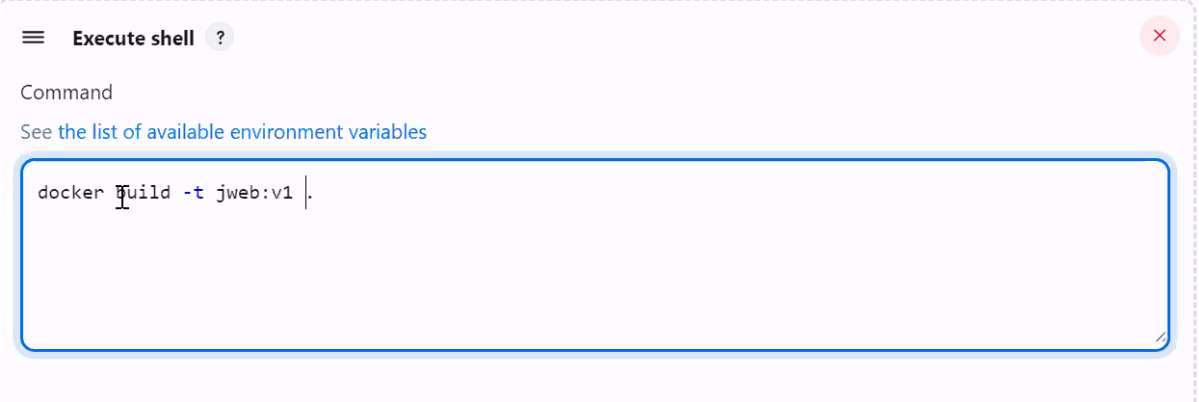
- What you need to do is copy the URL of this code and create a freestyle project in your project. Then, in the SCM settings, enter your Git URL.



- Jenkins download the the data but jenkins store there data in the workspace.
- If you run this job it will download the code in your workspace.



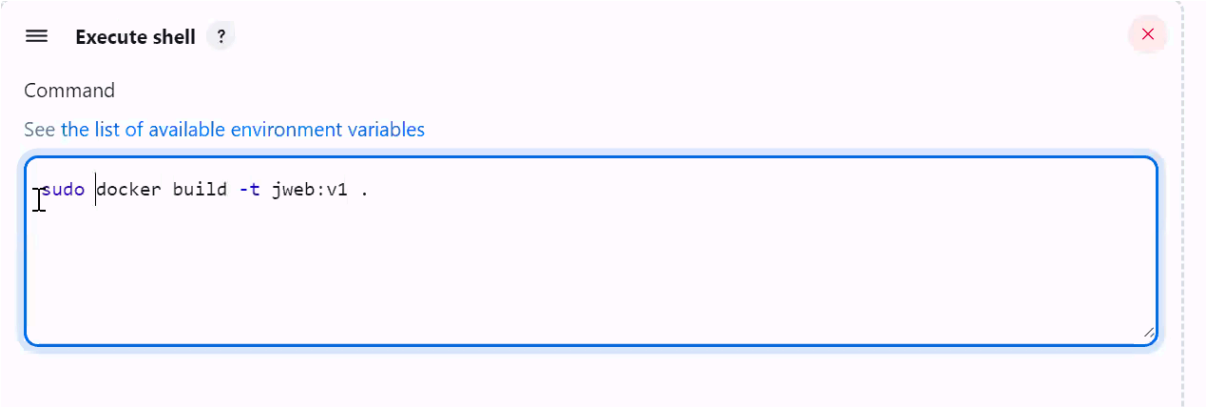
- The first step is done now we want to build it.
- Open the same job and go to configure add the steps to execute the command.



- So when we run job it will fail because of the permission issue of the because to run the docker we need to login with the root account but jenkins work with the jenkins user power.
- So for that we need to add the sudo file entry for jenkins user.

```
##
## Allow root to run any commands anywhere
root    ALL=(ALL)       ALL
jenkins ALL=(ALL) NOPASSWD:ALL
## Allows members of the 'sys' group to run networking, software,
## service management apps and more.
# %sys ALL = NETWORKING, SOFTWARE, SERVICES, STORAGE, DELEGATING, PROCESSES, LOCATE, DRIVERS

## Allows people in group wheel to run all commands
```

- And in the jenkins job we have to run the docker build command with sudo.



- If you check on local system we don't have any image with that we enter in job.

```
[root@ip-172-31-37-40 ~]#
[root@ip-172-31-37-40 ~]# docker images
REPOSITORY                    TAG       IMAGE ID       CREATED        SIZE
vimal13/myvimalweb            v1        7ff4f06ef38e   3 hours ago    282MB
myvimalweb                    v1        7ff4f06ef38e   3 hours ago    282MB
redhat/ubi8                   latest    56c43cbf8701   3 weeks ago    205MB
ubuntu                        14.04     13b66b487594   3 years ago    197MB
vimal13/apache-webserver-php  latest    05774ad1cd23   7 years ago    350MB
[root@ip-172-31-37-40 ~]#
```

- But if you run the job then you can see it will successfully run and build the image.

```
#5 DONE 0.0s

#6 [3/4] RUN pip3 install flask
#6 CACHED

#7 [2/4] RUN yum install python39 -y
#7 CACHED

#8 [4/4] COPY app.py  /app.py
#8 CACHED

#9 exporting to image
#9 exporting layers done
#9 writing image sha256:7ff4f06ef38ef855197c10b55b70b2d5af232871895d9eae26576e991b0db15a done
#9 naming to docker.io/library/jweb:v1 done
#9 DONE 0.0s
Finished: SUCCESS
```

- And now if you want to push the image then you can add docker push command.

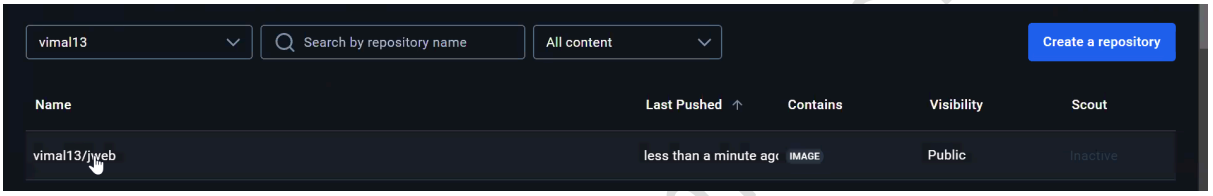
Execute shell ?

Command

See the list of available environment variables

```
sudo docker build -t vimal13/jweb:v1 .

sudo docker push vimal13/jweb:v1
```

- If you run this job again iit will download the code build the image and also push the image to dockerhub.



- Now for testing we can add the docker run command to run the container from this image.

```
sudo docker push vimal13/jweb:v1

sudo docker run -dit --name jos1 vimal13/jweb:v1
```

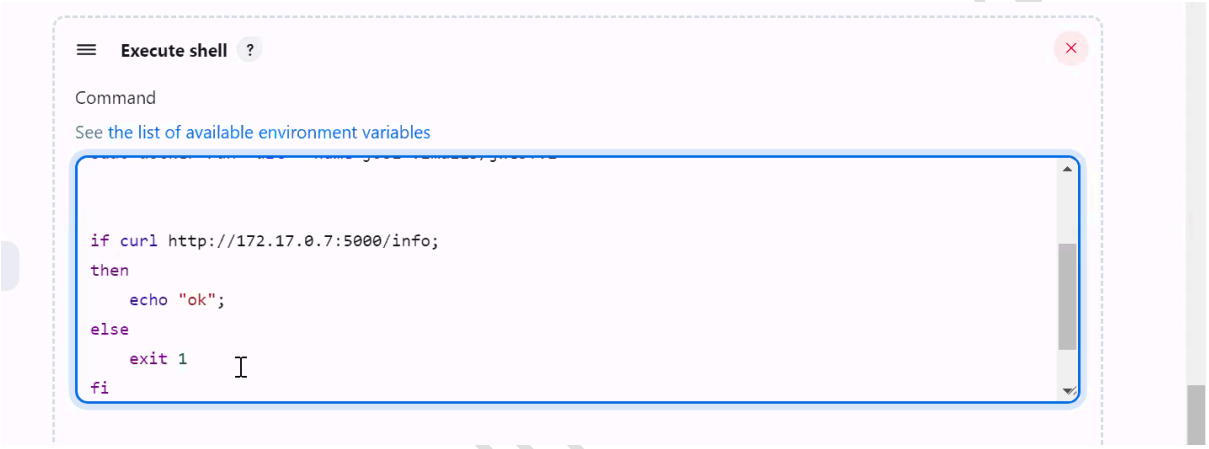
- And if you run the job then your container get started and what normally we do we use curl command for testing.

```
[root@ip-172-31-37-40 ~]#
[root@ip-172-31-37-40 ~]# curl http://172.17.0.6:5000/info
Welcome to LW....[root@ip-172-31-37-40 ~]#
```

- We can also make this automated.
- We can also implement the if and else condition here if it accessible then say okay if not print not.

```
[root@ip-172-31-37-40 ~]#
[root@ip-172-31-37-40 ~]# if curl http://172.17.0.6:5000/info
> then
> echo "ok"
> else
> echo "not"
> fi
Welcome to LW....ok
[root@ip-172-31-37-40 ~]#
```

- To make this automated we can use this in jenkins.



- We can use json syntax to retrieve the ip address from docker inspect command.

```
[root@ip-172-31-37-40 ~]# docker inspect  -f '{{json .NetworkSettings.IPAddress }}'  jos1
"172.17.0.6"
[root@ip-172-31-37-40 ~]#
```

- And to remove double quote we can use cut command to remove it.

```
[root@ip-172-31-37-40 ~]# docker inspect  -f '{{json .NetworkSettings.IPAddress }}'  jos1
"172.17.0.6"
[root@ip-172-31-37-40 ~]# docker inspect  -f '{{json .NetworkSettings.IPAddress }}'  jos1  | cut -d"\"" -f1

[root@ip-172-31-37-40 ~]# docker inspect  -f '{{json .NetworkSettings.IPAddress }}'  jos1  | cut -d"\"" -f2
172.17.0.6
[root@ip-172-31-37-40 ~]#
```

- And we can store this command in variable and use in the variable in curl.

```
[root@ip-172-31-37-40 ~]# docker inspect  -f '{{json .NetworkSettings.IPAddress }}'  jos1
"172.17.0.6"
[root@ip-172-31-37-40 ~]# docker inspect  -f '{{json .NetworkSettings.IPAddress }}'  jos1  | cut -d"\"" -f1

[root@ip-172-31-37-40 ~]# docker inspect  -f '{{json .NetworkSettings.IPAddress }}'  jos1  | cut -d"\"" -f2
172.17.0.6
[root@ip-172-31-37-40 ~]#
[root@ip-172-31-37-40 ~]# ip=`docker inspect  -f '{{json .NetworkSettings.IPAddress }}'  jos1  | cut -d"\"" -f2`
[root@ip-172-31-37-40 ~]# echo $ip
172.17.0.6
[root@ip-172-31-37-40 ~]# curl http://$ip:5000/info
Welcome to LW....[root@ip-172-31-37-40 ~]#
[root@ip-172-31-37-40 ~]#
[root@ip-172-31-37-40 ~]#
```

- We can use this concept in jenkins.

Execute shell ?

Command

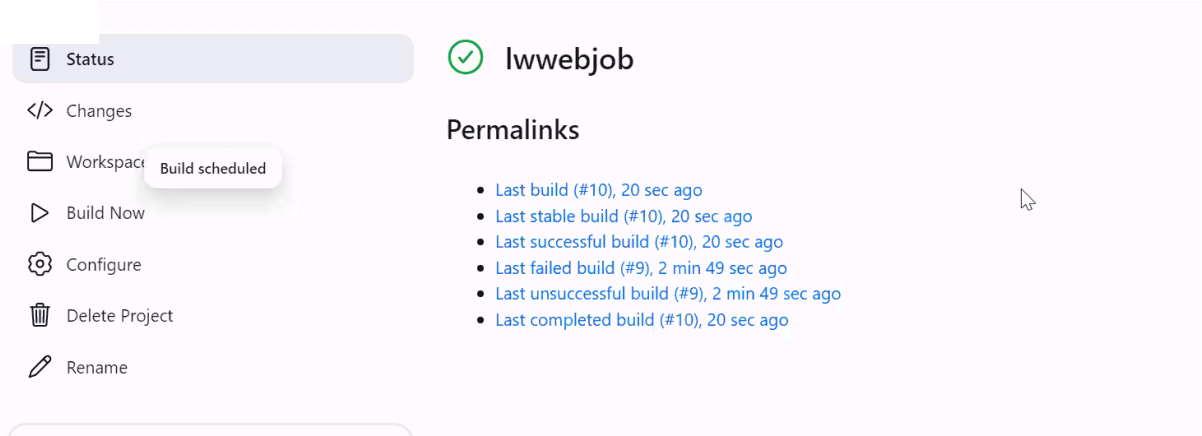
See the list of available environment variables

```
sudo docker build -t vimal13/jweb:v1 .

sudo docker push vimal13/jweb:v1

sudo docker rm -f jos1

sudo docker run -dit --name jos1 vimal13/jweb:v1

ip=`docker inspect  -f '{{json .NetworkSettings.IPAddress }}'  jos1  | cut -d"\"" -f2`

if curl http://$ip:5000/info;
then
    exit 0
else
    exit 1
```

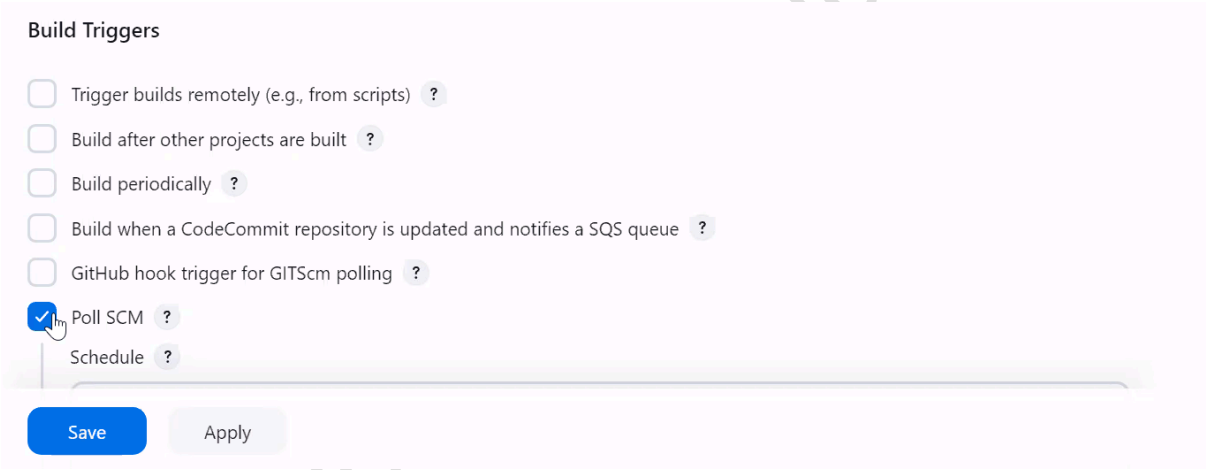
Save Apply

```
if curl http://$ip:5000/info;
then
    echo "link app working..."
    exit 0
else
    echo "link failed"
    exit 1
```

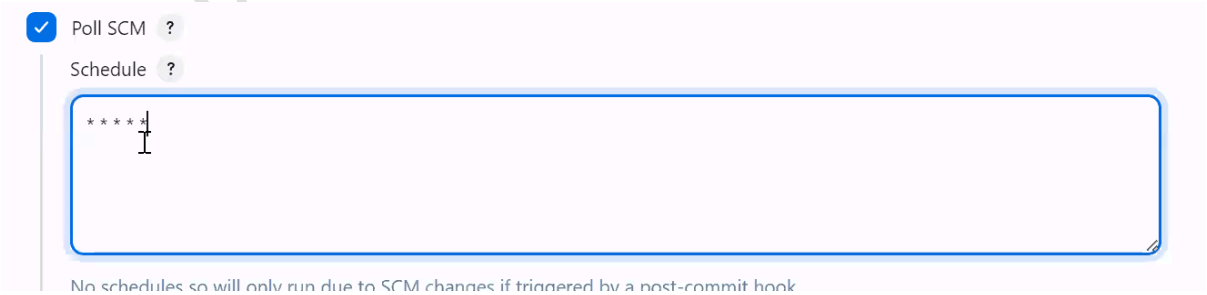
- Now every process is automated if developer make changes in application so you just need to run the jenkins job and everything is automated.



- But still we have one manual process we have to run the job manually of you go to build trigger setting you find the poll SCM option.



- Here you have to set in how much time jenkins check the github
- Here the support the crontab syntax.



- Now it will go to SCM every minute if any change market it will run this job automatically.