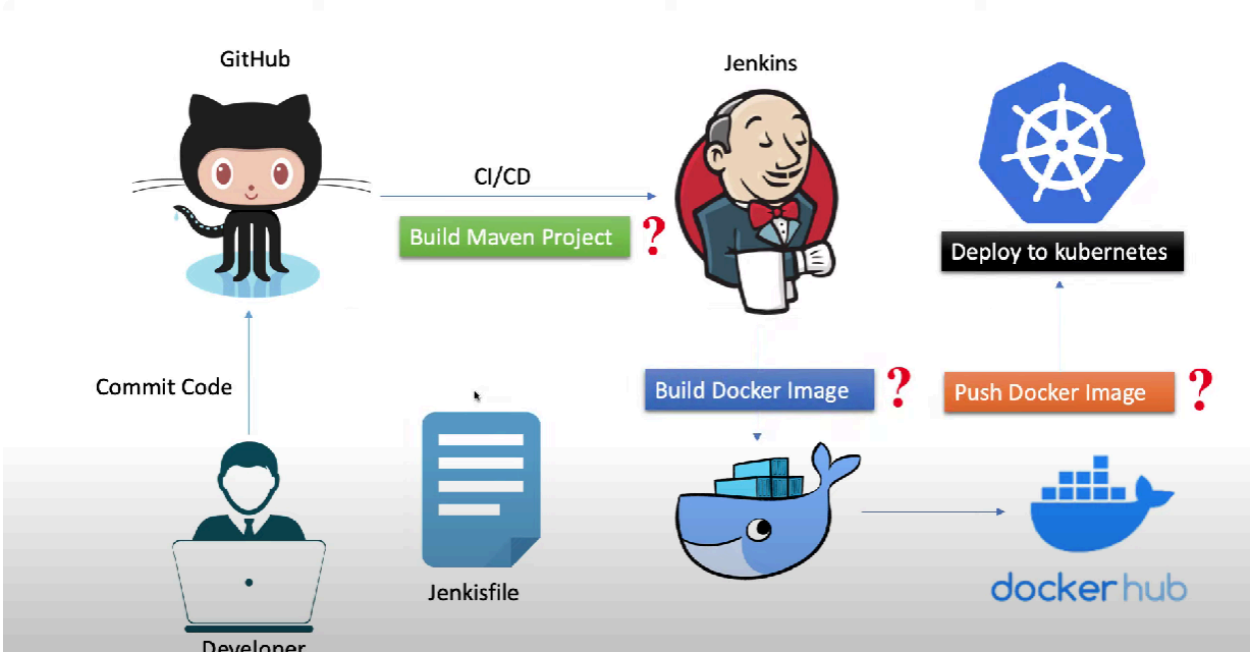
CI CD Pipeline Using Jenkins | Deploy Docker Image to Kubernetes using Jenkins



1. Git & GitHub/GitLab/Bitbucket (Version Control)

- **What is Git?**

Git is a distributed **version control system (VCS)** that helps track code changes, collaborate with teams, and manage different versions of a project.

- **Why do we use Git?**
  - Maintains a history of code changes.
  - Enables collaboration among developers.
  - Supports branching and merging for feature development.
- **Applications**
  - Software development teams use Git to manage source code.
  - Open-source projects store and track contributions on GitHub.

🔹 **GitHub/GitLab/Bitbucket**

These are **cloud-based repositories** where developers store and manage Git repositories.

- **GitHub** – Most popular for open-source and enterprise projects.
- **GitLab** – CI/CD pipelines built-in for automation.
- **Bitbucket** – Preferred for private repositories.

2. Jenkins / GitHub Actions / GitLab CI/CD (Continuous Integration - CI)

Jenkins is a tool that is used for automation. It is mainly an open-source server that allows all the developers to build, test and deploy software. It is written in Java and runs on java only. By using Jenkins we can make a continuous integration of projects(jobs) or end-to-endpoint automation.

Jenkins can deploy applications to various environments, depending on your setup and deployment strategy. Here are some common deployment targets:

On-Premise Servers

- **Tomcat, JBoss, or WebLogic** – Deploy Java applications (WAR/JAR files).
- **Virtual Machines** – Deploy to servers running Linux, Windows, etc.

Cloud Platforms

- **AWS (Amazon Web Services)** – Deploy to EC2, Lambda, or EKS (Kubernetes).
- **Azure** – Deploy to Azure App Services, VMs, or Kubernetes (AKS).
- **Google Cloud (GCP)** – Deploy to Compute Engine, Cloud Run, or Kubernetes (GKE).

Containerization & Orchestration

- **Docker** – Build and push images to Docker Hub or a private registry.
- **Kubernetes** – Deploy to Kubernetes clusters for scalable applications.

Why do we use Jenkins?

- Automates repetitive tasks (build, test, deploy).
- Detects errors quickly (running tests after every commit).
- Speeds up software delivery.

🔹 **Applications**

- Running unit tests automatically after pushing code.
- Triggering builds when a pull request is merged.
- Deploying applications to staging or production environments.

Deployment Strategy: Understanding Environments and Servers

When deploying an application, different **environments** (DEV, QA, STG, PROD) are used to **test, validate, and release** software. The choice between **on-premise** and **cloud** deployments depends on the infrastructure, security, and scalability needs.

What is a Jenkins Pipeline?

A pipeline is a set of stages that automate software delivery. Each stage takes input, processes it, and passes it to the next stage. If any step fails, the pipeline stops.

CI/CD Pipeline Stages in Jenkins

Test Code – Runs unit, integration, and functional tests.
Build Application – Compiles the code into an executable (e.g., .jar, .war).
Push to Repository – Uploads the build artifact to a repository (e.g., Nexus, JFrog, Docker Hub).
Deploy to Server – Deploys the application to DEV, QA, STG, or PROD environments.

CI/CD: Continuous Integration, Delivery, and Deployment

Continuous Integration (CI)

Continuous integration means whenever new code is committed to remote repositories like GitHub, GitLab, . CI will continuously build, test, and merge into a shared repository and it helps to detect and fix issues early.

🔹 **Key Features**

- Developers commit code multiple times a day.
- Automated unit, integration, and regression tests run on each commit.
- Identifies and fixes bugs early in development.
- Reduces integration conflicts between team members.

🔹 **Example Workflow**

- Developer pushes code to **GitHub/GitLab**.
- Jenkins/GitHub Actions triggers an **automated build**.
- Runs **unit tests & integration tests** (e.g., JUnit, Selenium).
- Fails? Developer fixes it before merging.
- Passes? Code is **merged** into the main branch.

🔹 **Tools Used**: Jenkins, GitHub Actions, GitLab CI/CD, CircleCI, TravisCI.

Continuous Delivery (CD)

Continuous Delivery ensures that **every successful CI build** is automatically **tested and ready for deployment** in staging or production. However, deployment still requires **manual approval**.

🔹 **Key Features**

✔ Code is always in a **deployable** state.
✔ Automated testing (functional, performance, security) is included.
✔ Deployments are **predictable and scheduled**.
✔ Reduces risk by catching issues before production.

🔹 **Example Workflow**

- After passing CI, Jenkins triggers **automated deployment** to **staging (STG)**.
- Runs additional tests (performance, security, acceptance).
- If stable, the **QA team** manually approves the release.
- Jenkins deploys to **production (PROD)** at a scheduled time.

🔹 **Tools Used**: Jenkins, AWS CodeDeploy, Azure DevOps, Spinnaker.

Continuous Deployment (CD)

Continuous Deployment is an **advanced level** of automation where **every change that passes all tests is deployed automatically** to production **without manual intervention**.

🔹 **Key Features**

✔ No need for **manual approvals**.
✔ Faster feedback loops—code goes live **within minutes**.
✔ No “release days” or waiting for scheduled deployments.
✔ Ensures rapid innovation and bug fixes.

🔹 **Example Workflow**

- Developer merges code → CI/CD pipeline runs tests.
- If tests pass, Jenkins **automatically deploys** the changes to **production**.
- Users instantly see the new feature or fix.
- If an issue is found, a **rollback** happens automatically.

🔹 **Tools Used**: Kubernetes, ArgoCD, Spinnaker, AWS CodePipeline.

1️⃣Continuous Integration (CI) →2️⃣Continuous Delivery (CD) →3️⃣Continuous Deployment (CD)

Explanation of the Order:

1️⃣**Continuous Integration (CI)** → Developers frequently commit code to a repository, triggering **automated builds and tests**. This ensures early bug detection.

2️⃣**Continuous Delivery (CD)** → Code that passes CI is **automatically deployed to staging or testing environments**. A manual approval step is required before production.

3️⃣**Continuous Deployment (CD)** → Every successful change that passes all tests is **automatically deployed to production**, eliminating the need for manual intervention.

Visual Representation of CI/CD Flow

[CI] → Build & Test Code

↓

[CD] → Deploy to Staging (Manual Approval for Production)

↓

[CD] → Auto Deploy to Production (No Manual Approval)

CI is always the first step. Then, based on the business needs, teams can implement either **Continuous Delivery (CD)** or **Continuous Deployment (CD)**.

3. Docker (Containerization)

- **What is Docker?**

Docker is a containerization platform that packages applications and their dependencies into lightweight, portable containers. As its name implies, Docker is a containerization platform that packages applications and their dependencies together in containers to ensure that the applications run seamlessly across multiple environments.Docker containers are entire file systems that contain all the components for running a piece of software: code, system tools, runtime, system libraries, etc. Any software that can be installed on a server can be packaged in a Docker container. It ensures that the software always runs the same in any environment.

- **Why do we use Docker?**
  - Ensures the app runs consistently across different environments.
  - Eliminates “works on my machine” issues.
  - Lightweight compared to traditional VMs.
- **Applications**
  - Running a microservice architecture.
  - Deploying applications in Kubernetes.
  - Building DevOps pipelines with containerized services.

Example Dockerfile for a Spring Boot App:

```
FROM openjdk:17
COPY target/myapp.jar app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

EX:

You have a Spring Boot REST API that needs to run on different environments without compatibility issues. Instead of installing Java, dependencies, and setting up the server manually, you use Docker to package everything into a container.

When working with **software deployment**, we often deal with three main components:
✔ **Environment** – Defines where the application runs (DEV, QA, STG, PROD).
✔ **Server** – The physical or virtual machine that hosts the application.
✔ **Docker** – A tool that packages applications and their dependencies to run anywhere.

Traditional Deployment (Without Docker)

In a traditional setup, each **environment** (DEV, QA, STG, PROD) runs on a **server** where you manually install:

- Java
- Apache Tomcat
- Database (MySQL, PostgreSQL)
- Other dependencies

👉 **Challenges:**

- Each environment may have different software versions, leading to compatibility issues.
- If you move to a new server, you must install everything again.
- More maintenance and risk of "it works on my machine" issues.

Docker-Based Deployment (With Docker)

With **Docker**, instead of manually installing software on different servers, you create a **Docker container** that includes:

- Java, MySQL, Tomcat, and your Spring Boot app.
- The same image runs in **any environment** (DEV, QA, STG, PROD) without additional setup.

You create a **Docker image** with all dependencies.

Deploy the **same image** to different environments without reinstallation.

The server only needs **Docker installed**—nothing else.

No compatibility issues since everything runs inside a **container**.

4.Kubernetes (Container Orchestration)

🔹 **What is Kubernetes (K8s)?**

Kubernetes is a container orchestration tool that manages the deployment, scaling, and operation of containerized applications. Kubernetes, also known as K8s, is an open-source container orchestration platform that performs different tasks like deployment, scaling, management, and monitoring containerized applications.

🔹 **Why do we use Kubernetes?**

- Automates container deployment and scaling.
- Ensures high availability and load balancing.
- Manages rolling updates and rollbacks.

🔹 **Applications**

- Deploying microservices in cloud environments.
- Scaling applications automatically based on demand.

What are the components of Kubernetes?

1️⃣**Control Plane Components** (Manage the cluster)
2️⃣**Worker Node Components** (Run applications)

Simple Example - Coffee Shop ☕

🔸 Node (Worker Node) = Coffee Machine
🔸 Pod = A Coffee Cup
🔸 Container = Coffee inside the cup

💡 Each Node (Coffee Machine) can serve multiple Pods (Coffee Cups), and each Pod has one or more Containers (Types of Coffee).

🔹 **Summary**

- ✅ Node = A machine that runs applications (Worker Node in Kubernetes)
- ✅ Pod = A unit inside a Node that contains one or more containers
- ✅ Containers = Actual applications running inside a Pod

What is a Pod?

A Pod is one or more containers (applications) that work together inside a Worker Node.
Pods share the same network and storage.
Example: A Pod might contain:

- **Container 1** → Spring Boot API
- **Container 2** → MySQL Database

Role of the Control Plane

The **Control Plane manages Pods** and **assigns them to Worker Nodes** for execution.
two main categories:

It decides **which Worker Node should run which Pod**.

Control Plane Components (Brain of Kubernetes)

These components **control** and **manage** the cluster. A **Kubernetes Cluster** is a **group of nodes** that work together to run **containerized applications**. It consists of a **Control Plane (Master Node)** and **Worker Nodes** that manage and execute applications.

Simple Example to Understand Kubernetes Control Plane & Worker Nodes

1️⃣ Control Plane Components (Manage the Cluster) = Restaurant Manager 👨‍🍳

The **Control Plane** is like the **Restaurant Manager** who organizes everything:

- Takes orders
- Assigns work to chefs
- Ensures everything runs smoothly

Control Plane Component	Role	Restaurant Analogy
API Server	Handles requests from users (developers)	The **receptionist** who takes food orders 📋
Scheduler	Assigns tasks (Pods) to Worker Nodes	The **manager** who decides which chef prepares each dish 🍽️
Controller Manager	Ensures the system is always running correctly	The **supervisor** who makes sure all orders are being cooked 🔍
etcd	Stores cluster information	The **kitchen logbook** that keeps track of all orders 📖

👉 **Example:**
When a customer places an order, the **receptionist (API Server)** receives it. The **manager (Scheduler)** assigns the order to the right chef (Worker Node). The **supervisor (Controller Manager)** ensures the food is being prepared correctly.

Worker Node Components (Run Applications)

Worker nodes are responsible for **running** containerized applications.

2️⃣ Worker Node Components (Run Applications) = Kitchen Chefs 👨‍🍳👩‍🍳

The **Worker Nodes** are like **chefs** in the kitchen who prepare and serve the food.

Worker Node Component	Role	Restaurant Analogy
Kubelet	Ensures the assigned tasks (Pods) are running properly	The **chef** who follows the order and cooks the food 🍳
Kube Proxy	Manages networking between services	The **waiter** who delivers food to the right table 🍽️
Container Runtime (Docker, etc.)	Runs applications inside containers	The **stove/oven** where food is cooked 🔥

👉 **Example:**

The **chef (Worker Node)** gets an order from the **manager (Control Plane)** and starts cooking. The **waiter (Kube Proxy)** ensures the food reaches the right table (users). The **stove (Container Runtime)** is where the food is prepared.

Docker vs. Kubernetes – What's the Difference?

Docker and Kubernetes are related but different technologies.

- Docker is a containerization tool that packages applications and their dependencies.
- Kubernetes is a container orchestration platform that manages multiple Docker containers across different servers.

FLOW

Application to GitHub and GitHub to Jenkins and Jenkins to DockerHub

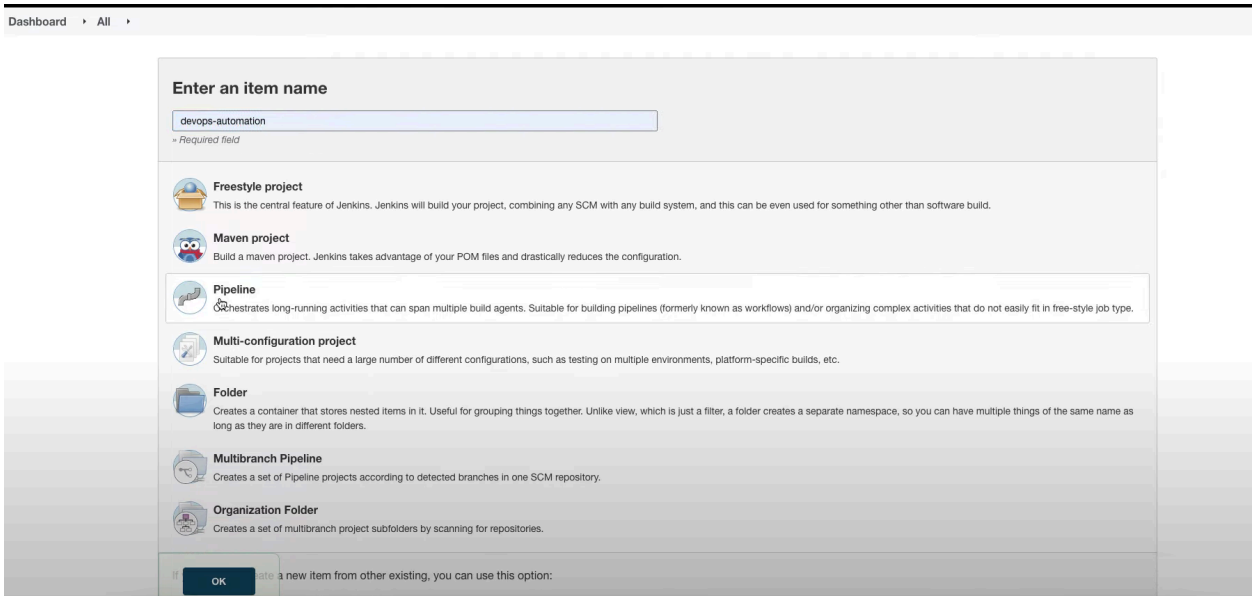
Step1

Push changes to github

Step2:

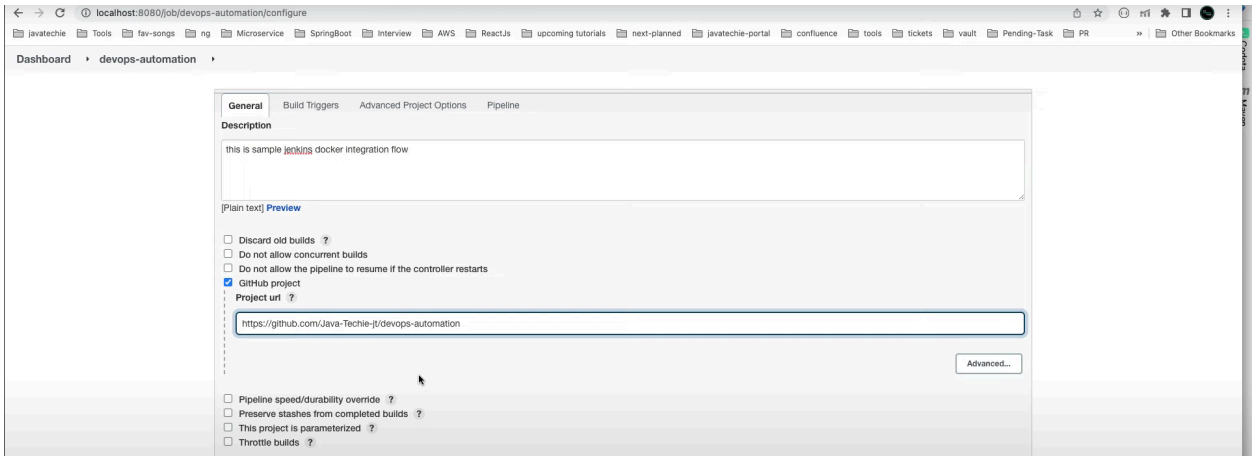
Start the jenkins in your machine and login into the console.

Click on a new **item** to create a new folder jenkins . Enter the project name that you want and select the pipeline and enter **OK**.

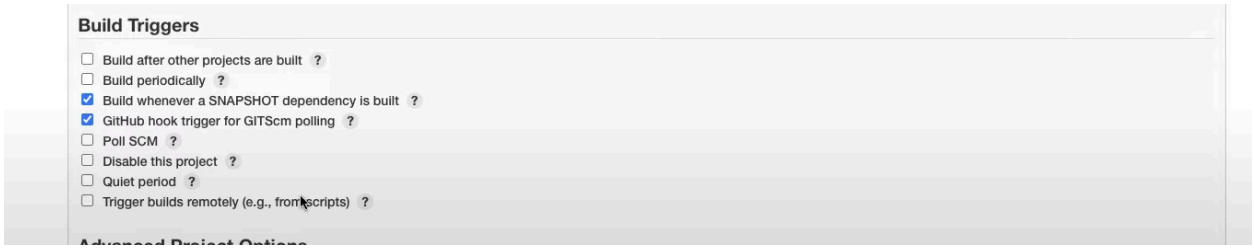


Add some description about what it is and select the source code where it is integrated with jenkins.

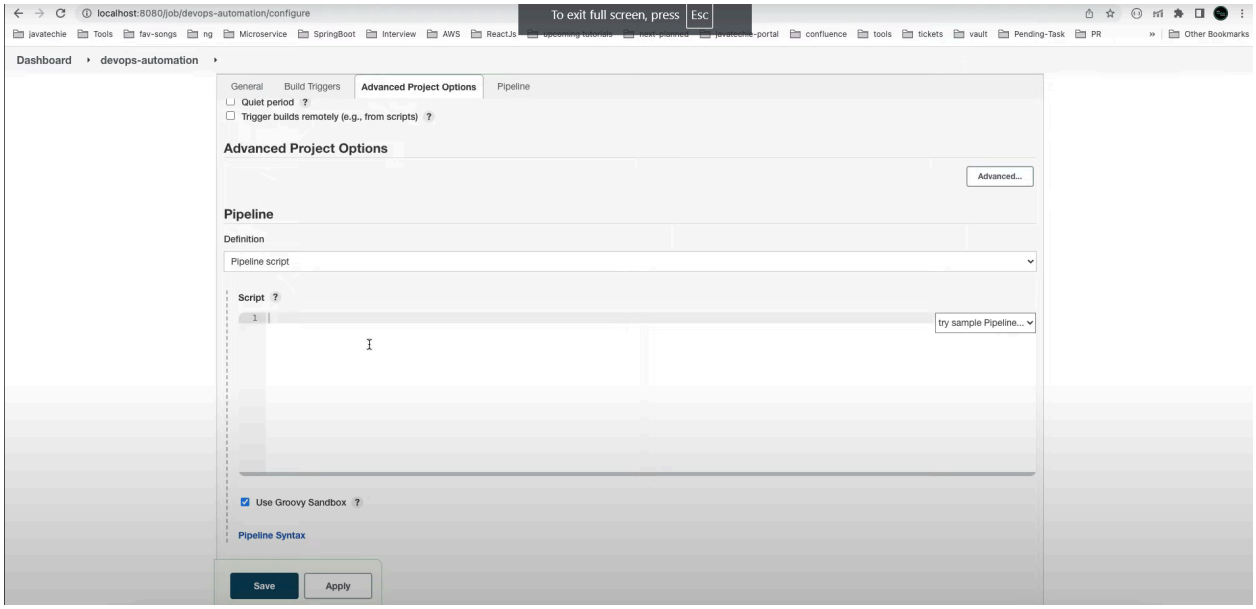
Ex: Github and provide the repository URL

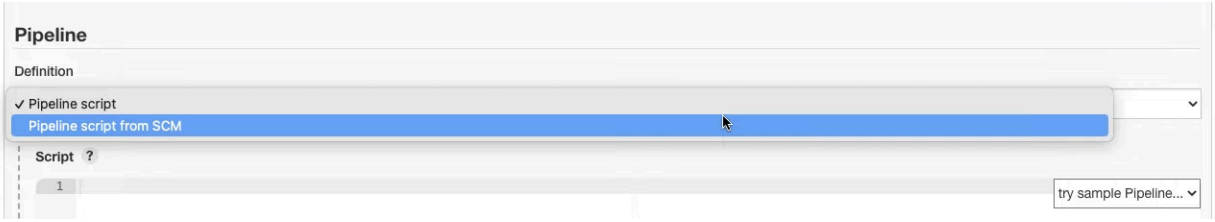


Select the required the options in Build Trigger

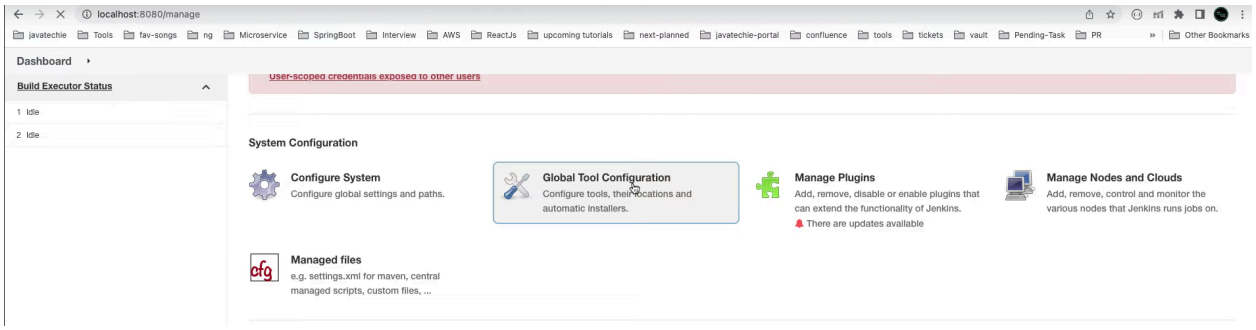


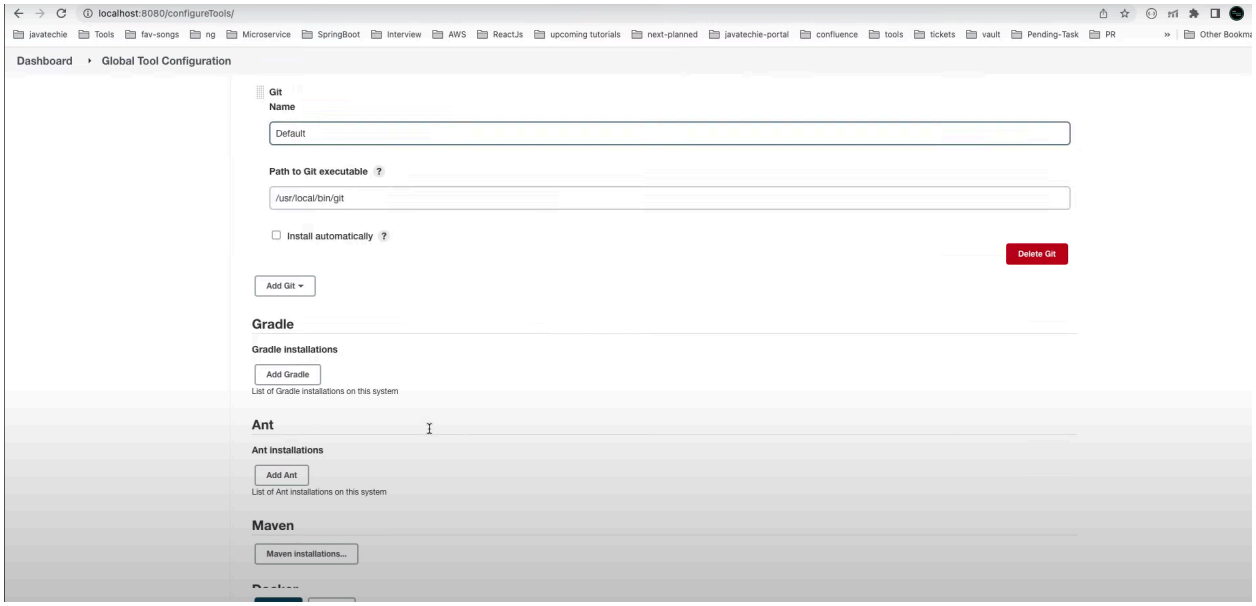
Provide pipeline script.Here will provide scripts for what and all jenkins needs to perform operation. This is called **jenkinsfile .** Either we can write code/script here or we can select **jenkinsfile directly.**





Basically we all provide tools details like git , maven ,jdk ,Docker installation paths in jenkins Global Tool Configuration.





Basic Syntax script of jenkinsfile

pipeline {}

- The main block that defines the Jenkins pipeline.

agent

- Specifies where the pipeline runs.
- Examples:
  agent any // Runs on any available agent

tools

- Specifies tools like Maven, JDK, or Gradle.
- Example:
  tools { maven 'Maven_3_6_3' }

stages

- Defines multiple steps in a pipeline.

- Each stage contains **steps** to perform actions like building, testing, and deploying.

steps

- Contains the actual commands to execute.

Example:

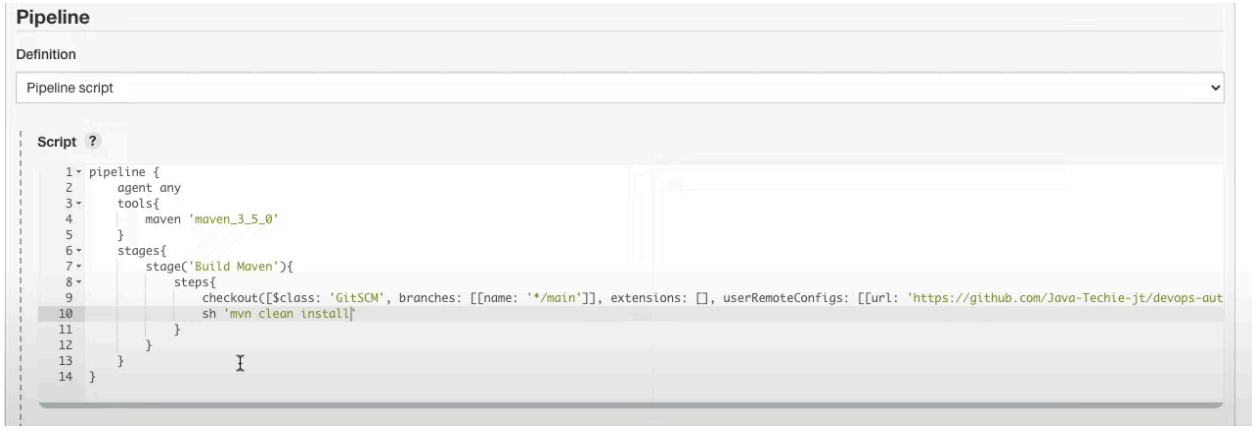
```
steps {

   sh 'mvn clean install'

}
```

post

- Defines actions to perform **after** the pipeline execution.
- Common conditions:
    - always → Runs regardless of success or failure.
    - success → Runs only if the pipeline succeeds.
    - failure → Runs only if the pipeline fails.

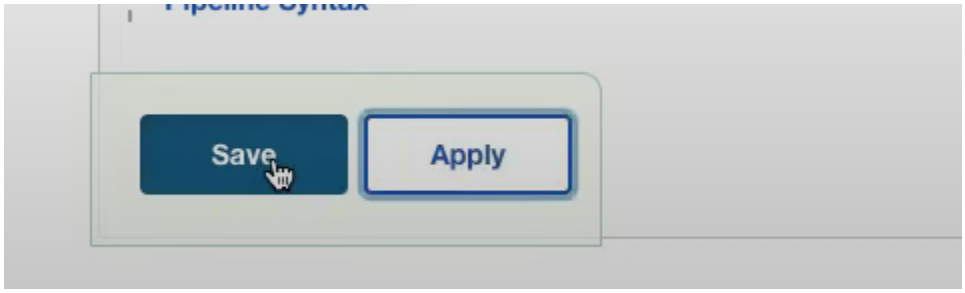
EX:

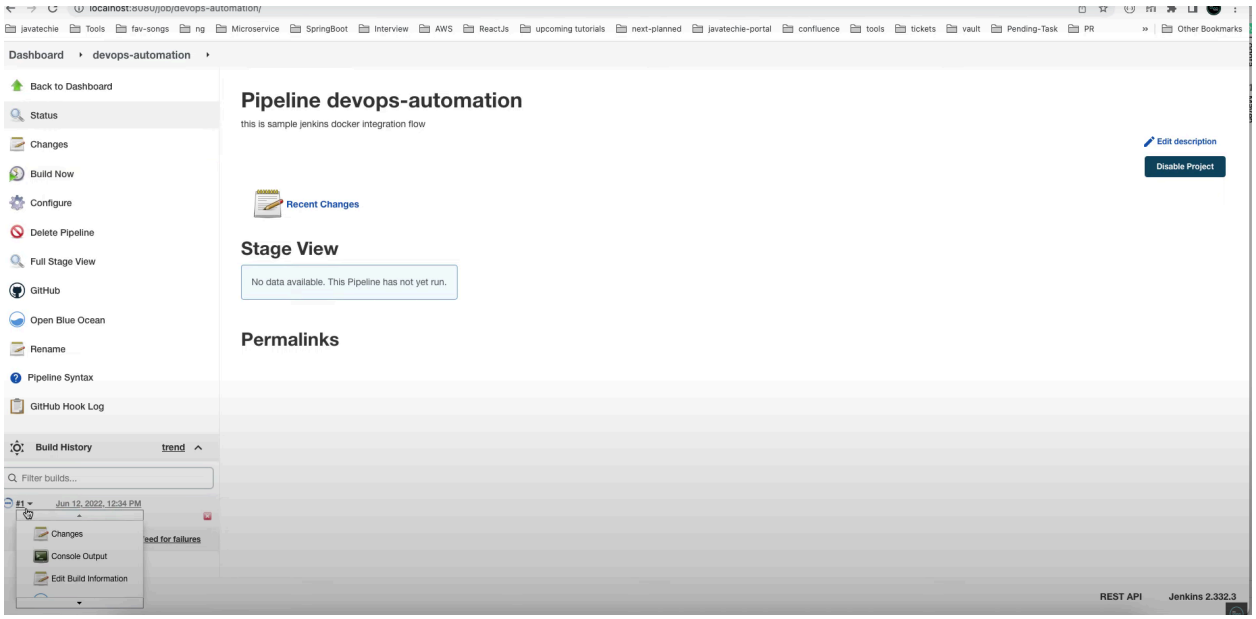
Just check out the code and do maven clean install



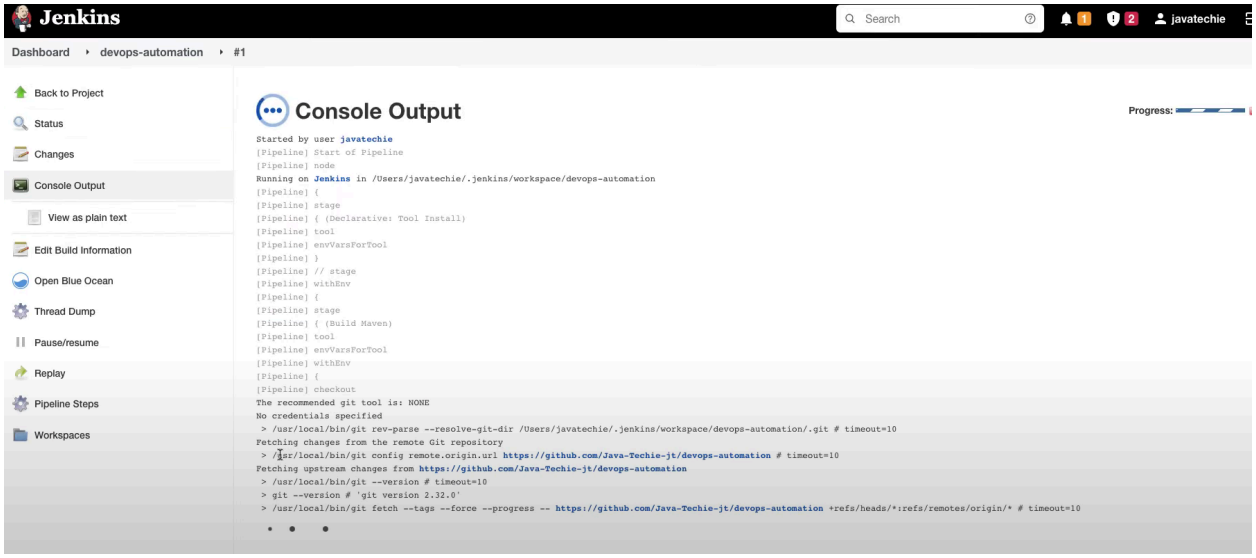
We can use Pipeline Syntax to get syntax by selecting respective opertaions

After that click on Save and click Build option in Dashboard

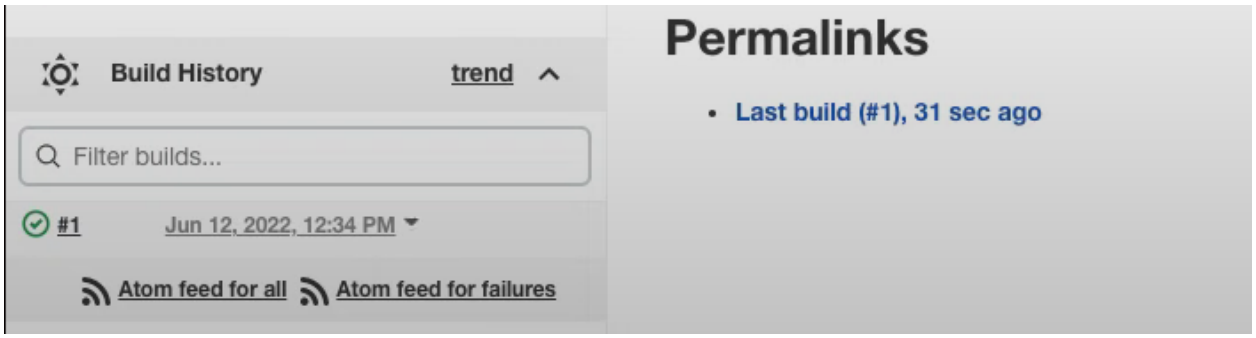




It will start execution and we can see by clicking on console output

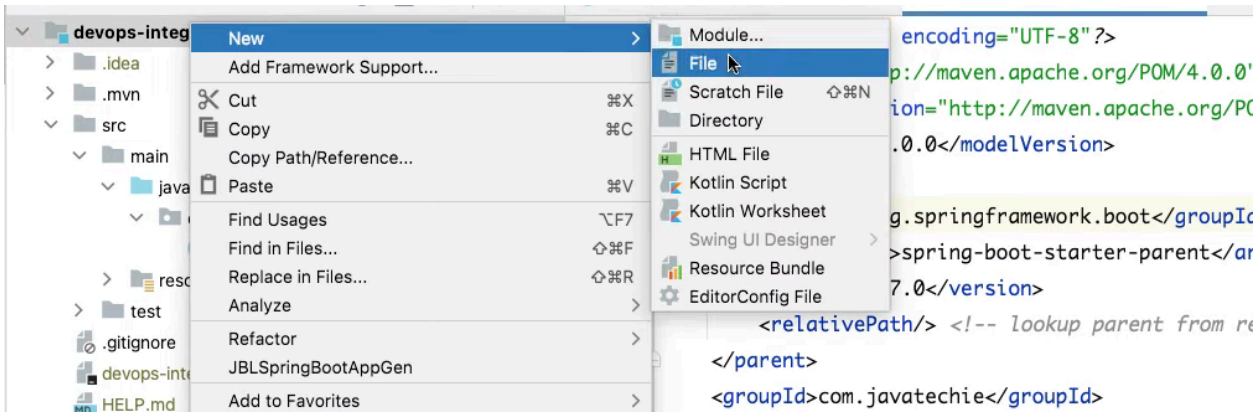


We can see the build got succeed.



To building the Docker Image , Our application should have docker file

Go to Spring boot project / any project and create **dockerfile** by right click on the project



In that specifies jar name of application and other details

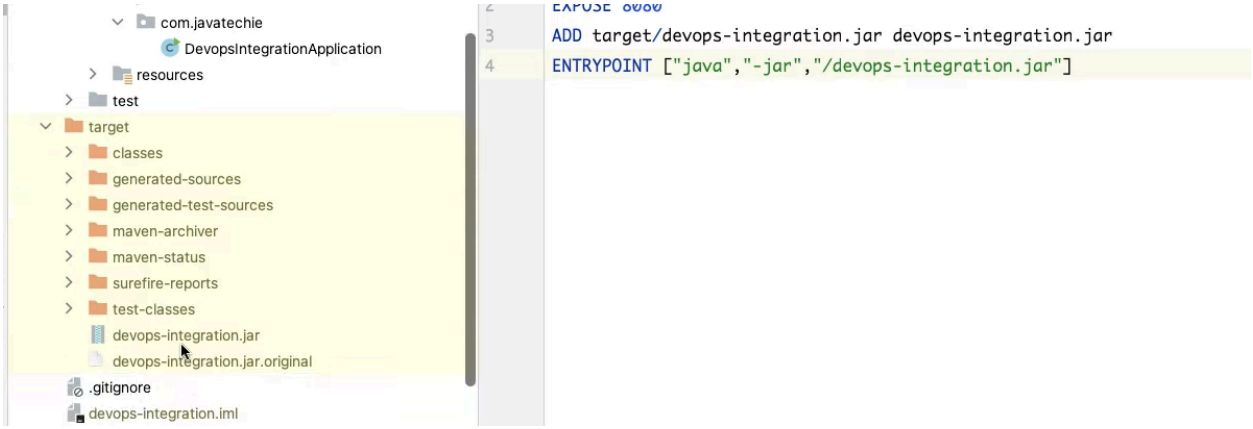
Add name you want to generate of jar file

```
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
        <finalName>devops-integration</finalName>
    </build>
```

DevopsIntegrationApplication.java | pom.xml (devops-integration) | Dockerfile

```
1  FROM openjdk:8
2  EXPOSE 8080
3  ADD target/devops-integration.jar devops-integration.jar
4  ENTRYPOINT ["java","-jar","/devops-integration.jar"]
```

We can do maven clean install from where pom.xml file is available and then it will generate .**jar file** and cross check weather it generates with same filename what you specifies in pom.xml or not



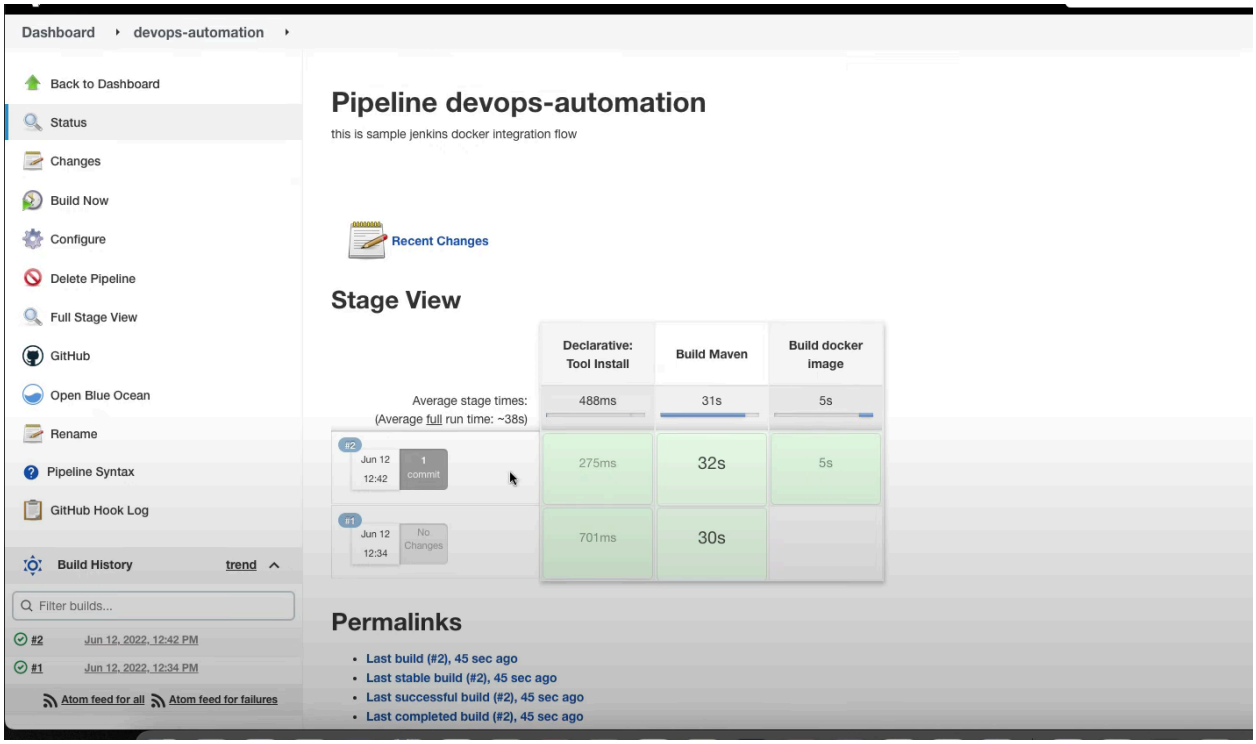
Now we can push the latest changes to GitHub.

Now again will provide script in jenkins to generate this docker image using dockerfile in application

```
stage('Build docker image'){
    steps{
        script{
            sh 'docker build -t javatechie/devops-integration .'
        }
    }
}
```

And again After that click on Save and click Build option in Dashboard

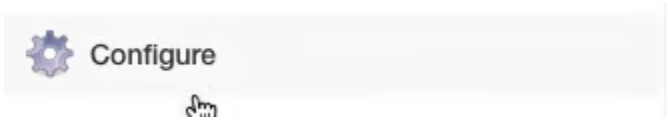
After building success , We can see here it has added another pipeline “Build docker image”.



Now the docker image has been generated.

Now we can write script in jenkins to push **docker image** to **dockerHub**

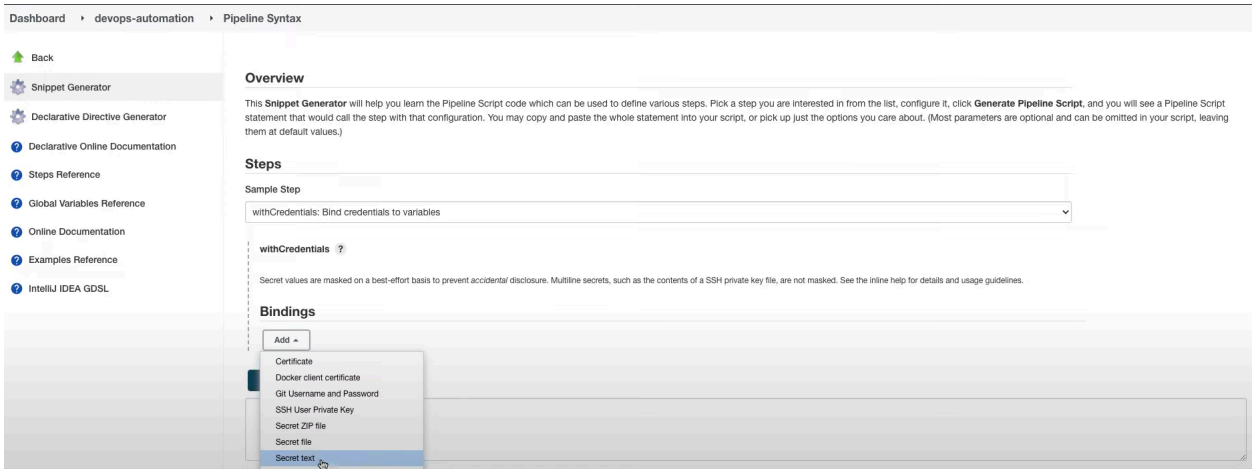
Click on project




```
stage('Build docker image'){
    steps{
        script{
            sh 'docker build -t javatechie/devops-integration .'
        }
    }
}
```

```
    stage('Push image to Hub'){
        steps{
            script{
                withCredentials([string(credentialsId: 'dockerhub-pwd', variable: 'dockerhubpwd')]) {
                sh 'docker login -u javatechie -p ${dockerhubpwd}'

                sh 'docker push javatechie/devops-integration'
                }
            }
        }
    }
}
```

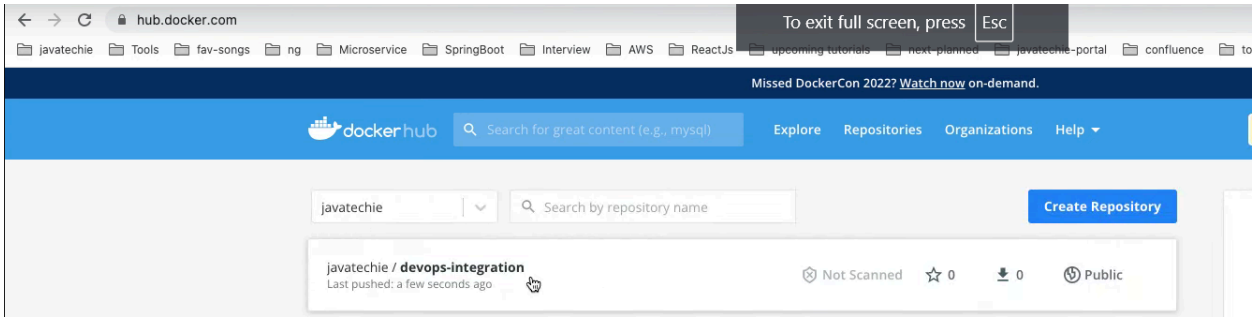
We can use pipeline syntax to validate docker credentials here and provide syntax like above.



Again we save and build it will generate docker images and it will push to dockerHub.

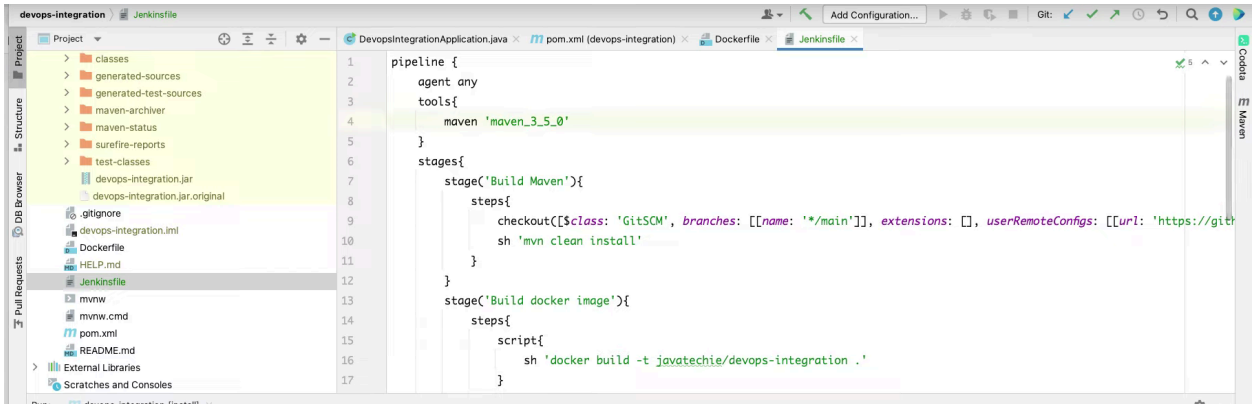
We can see here it has added another pipeline “push image to Hub”.

We can login and see in Docker portal.

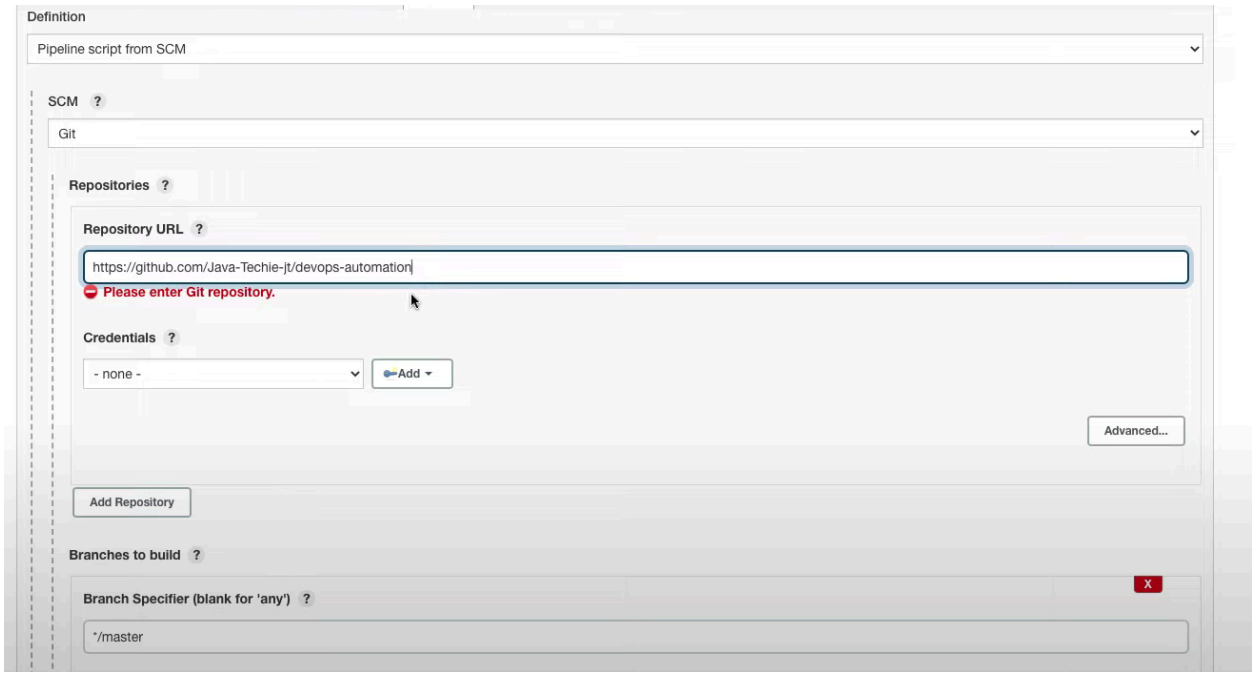


If I don't want to write the script in the jenkins portal and make it global.

We can write that script in code itself by creating a folder in the project (create **file** by right clicking on the project). Just copy the script paste in folder.

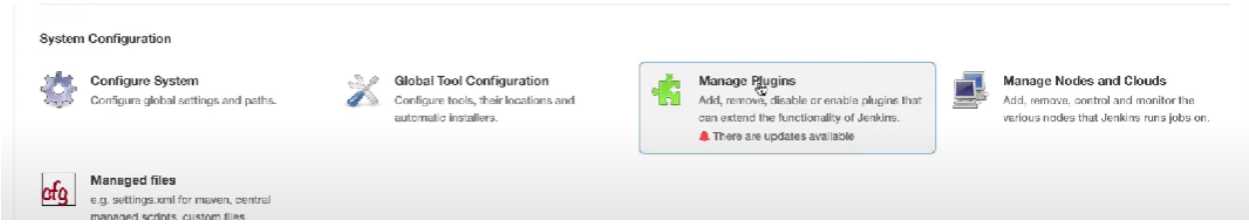


So do commit changes again and use



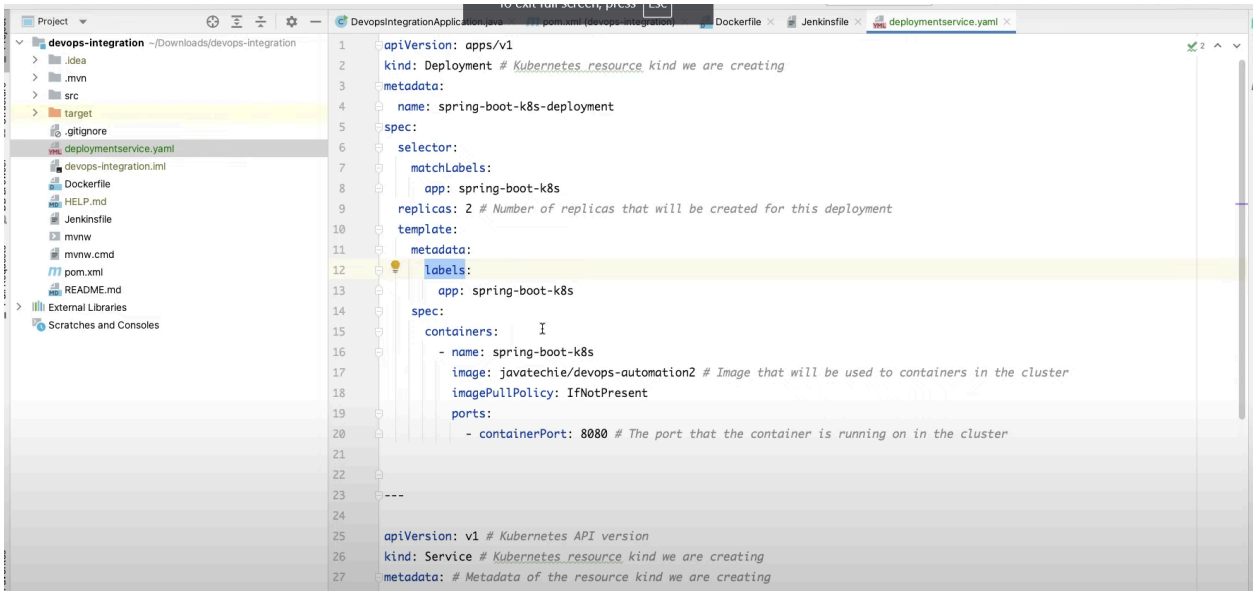
And we will get the same results.

How to deploy docker image into kubernetes cluster using Jenkins pipeline



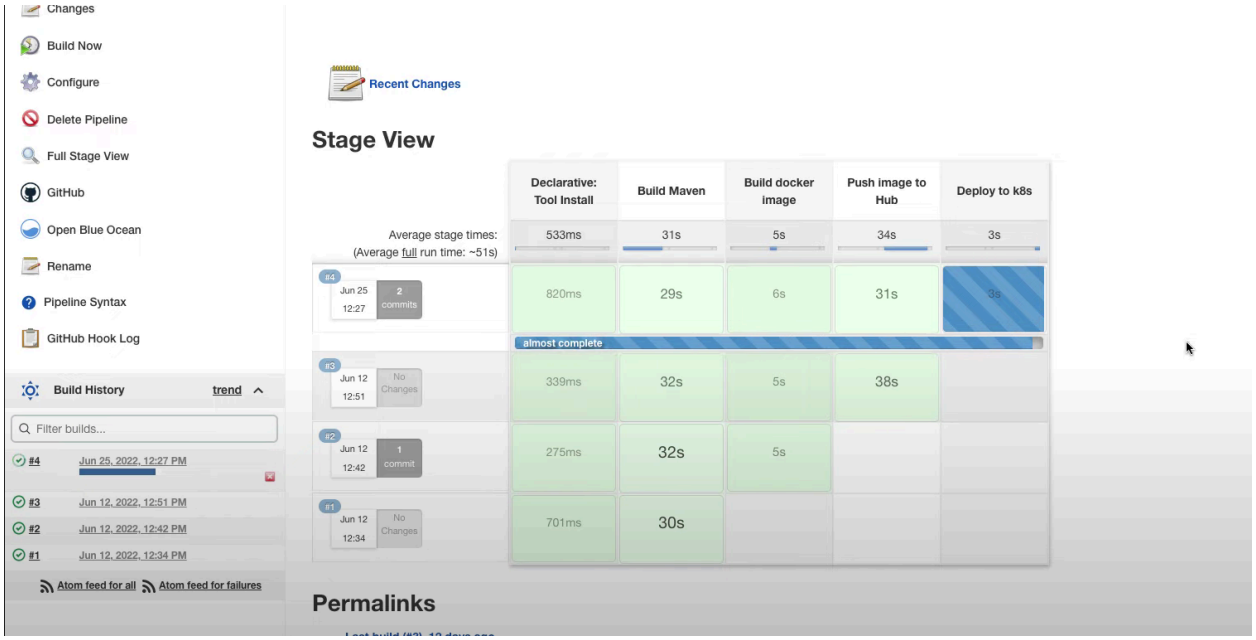
Got available and installed the kubernetes plugin.

Create a file in application , add **deployment**(docker image name , container names and other details) and **service** info.



Add the respective script using pipeline syntax and run .

We can see here it has added another pipeline “deploy to k8s”.

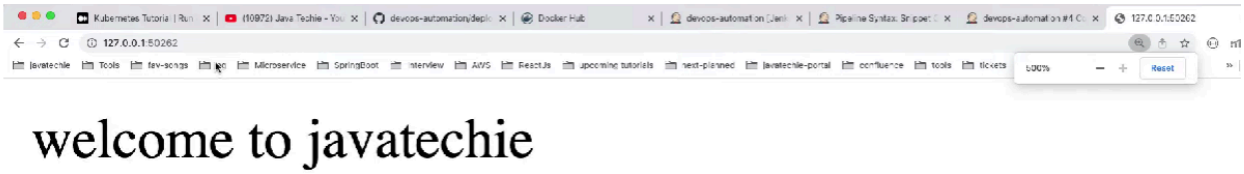


We can open kubernetes and verify

```
javatechie@Basantas-iMac .kube % open config
javatechie@Basantas-iMac .kube % kubctl get nodes
zsh: command not found: kubctl
javatechie@Basantas-iMac .kube % kubectl get pods
No resources found in default namespace.
javatechie@Basantas-iMac .kube % kubctl get nodes
zsh: command not found: kubctl
javatechie@Basantas-iMac .kube % kubectl get nodes
NAME       STATUS   ROLES                  AGE   VERSION
minikube   Ready    control-plane,master   15m   v1.22.3
javatechie@Basantas-iMac .kube % kubectl get deployments
NAME                     READY   UP-TO-DATE   AVAILABLE   AGE
spring-boot-k8s-deployment   0/2     2            0           41s
javatechie@Basantas-iMac .kube % kubectl get pods
NAME                                      READY   STATUS              RESTARTS   AGE
spring-boot-k8s-deployment-54f7c748f5-chf5r   0/1     ContainerCreating   0          50s
spring-boot-k8s-deployment-54f7c748f5-qj4cx   0/1     ContainerCreating   0          50s
javatechie@Basantas-iMac .kube % kubectl get pods
NAME                                      READY   STATUS              RESTARTS   AGE
spring-boot-k8s-deployment-54f7c748f5-chf5r   0/1     ContainerCreating   0          60s
spring-boot-k8s-deployment-54f7c748f5-qj4cx   0/1     ContainerCreating   0          60s
javatechie@Basantas-iMac .kube % kubectl get pods
NAME                                      READY   STATUS    RESTARTS   AGE
spring-boot-k8s-deployment-54f7c748f5-chf5r   1/1     Running   0          81s
spring-boot-k8s-deployment-54f7c748f5-qj4cx   1/1     Running   0          81s
javatechie@Basantas-iMac .kube % cd ~
javatechie@Basantas-iMac ~ % kubectl get logs spring-boot-k8s-deployment-54f7c748f5-chf5r
error: the server doesn't have a resource type "logs"
javatechie@Basantas-iMac ~ % kubectl logs spring-boot-k8s-deployment-54f7c748f5-chf5r

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.7.0)

2022-06-25 07:00:19.426  INFO 1 --- [           main] c.j.DevopsIntegrationApplication         : Starting DevopsIntegrationApplication v0.0.1-SNAPSHOT using Java 1.8.0_322 on spring-boot-k8s-deployment-54f7c748f5
-chf5r with PID 1 (/devops-integration.jar started by root in /)
2022-06-25 07:00:19.433  INFO 1 --- [           main] c.j.DevopsIntegrationApplication         : No active profile set, falling back to 1 default profile: "default"
2022-06-25 07:00:21.566  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2022-06-25 07:00:21.585  INFO 1 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2022-06-25 07:00:21.585  INFO 1 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.63]
2022-06-25 07:00:21.734  INFO 1 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2022-06-25 07:00:21.734  INFO 1 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 2155 ms
2022-06-25 07:00:23.106  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2022-06-25 07:00:23.186  INFO 1 --- [           main] c.j.DevopsIntegrationApplication         : Started DevopsIntegrationApplication in 4.551 seconds (JVM running for 5.464)
javatechie@Basantas-iMac ~ % kubectl get svc
NAME              TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
kubernetes        ClusterIP   10.96.0.1        <none>        443/TCP          20m
springboot-k8ssvc   NodePort    10.100.229.111   <none>        8080:31144/TCP   3m14s
javatechie@Basantas-iMac ~ %
```

```
javatechie@Basantas-iMac ~ % minikube service springboot-k8ssvc --url
🏃  Starting tunnel for service springboot-k8ssvc.
|-----------|-------------------|-------------|------------------------|
| NAMESPACE |       NAME        | TARGET PORT |          URL           |
|-----------|-------------------|-------------|------------------------|
| default   | springboot-k8ssvc |             | http://127.0.0.1:50262 |
|-----------|-------------------|-------------|------------------------|
http://127.0.0.1:50262
❗  Because you are using a Docker driver on darwin, the terminal needs to be open to run it.
```

From kubernetes run application URL and see the api page



welcome to javatechie

Step-by-Step Flow:

1️⃣ Git (Local Development)

- Developers write code and push it to **GitHub**.

2️⃣ GitHub (Remote Code Storage)

- Stores the source code repository.
- Jenkins pulls the latest code from GitHub.

3️⃣ Jenkins (CI/CD Automation)

- Builds the Spring Boot app using **Maven**.
- Runs **unit tests**.
- Creates a **Docker image**.
- Pushes the **Docker image to Docker Hub**.

4️⃣ Docker (Containerization)

- Ensures the app runs the same everywhere.
- Jenkins pushes the image to Docker Hub.

5️⃣ Kubernetes (Container Orchestration - Deployment Stage 1)

- Runs multiple containers efficiently.
- Jenkins can deploy the Docker image to a **local Kubernetes cluster (Minikube)** or **AWS Kubernetes (EKS)**.

AWS (or any cloud provider) is used for **hosting the Kubernetes cluster** where the app runs.

- **Kubernetes needs an infrastructure** to run on. It can be: ✅ **Local (Minikube on Windows/Linux/Mac)**
  ✅ **Cloud-based (AWS, GCP, Azure, DigitalOcean, etc.)**

So AWS fits **after Kubernetes**, when you want to deploy the app to a **cloud environment** for scalability, availability, and security.

How AWS is Used in This Process

✔ Instead of Minikube (Local Kubernetes), we use AWS EKS (Elastic Kubernetes Service)
✔ Instead of a local server, we use AWS EC2 instances to run workloads
✔ Instead of local databases, we use AWS RDS (Relational Database Service) for better performance
✔ Instead of local logs, we use AWS CloudWatch for monitoring and logging
✔ Instead of Jenkins running locally, we use AWS CodePipeline for CI/CD

Setup AWS Infrastructure

We need the following **AWS services**:

- **EKS (Elastic Kubernetes Service)** → Runs the Kubernetes cluster
- **ECR (Elastic Container Registry)** → Stores Docker images
- **EC2 (Elastic Compute Cloud)** → Runs Jenkins
- **IAM (Identity & Access Management)** → Manages permissions

GitHub → Jenkins → Docker → Kubernetes (EKS) → AWS