Run jenkins on docker

```
# Create docker network
docker network create jenkins

# Run docker daemon exposed with port 2376
docker run --name jenkins-docker --rm --detach ^
  --privileged --network jenkins --network-alias docker ^
  --env DOCKER_TLS_CERTDIR=/certs ^
  --volume jenkins-docker-certs:/certs/client ^
  --volume jenkins-data:/var/jenkins_home ^
  --publish 2376:2376 ^
  docker:dind

# Run the image
docker run --name jenkins-blueocean --restart=on-failure --detach ^
  --network jenkins --env DOCKER_HOST=tcp://docker:2376 ^
  --env DOCKER_CERT_PATH=/certs/client --env DOCKER_TLS_VERIFY=1 ^
  --volume jenkins-data:/var/jenkins_home ^
  --volume jenkins-docker-certs:/certs/client:ro ^
  --publish 8080:8080 --publish 50000:50000 vijaynvb/jenkins:1.0
```

Step up admin details

- Access the jenkins application in localhost:8080
- Login with default credentials - To get the password , go to the shown directory or simply run docker logs containerId & get the password from logs.
- Install suggested plugins
- Create new user 'administrator' with password '********'
- Now start jenkins

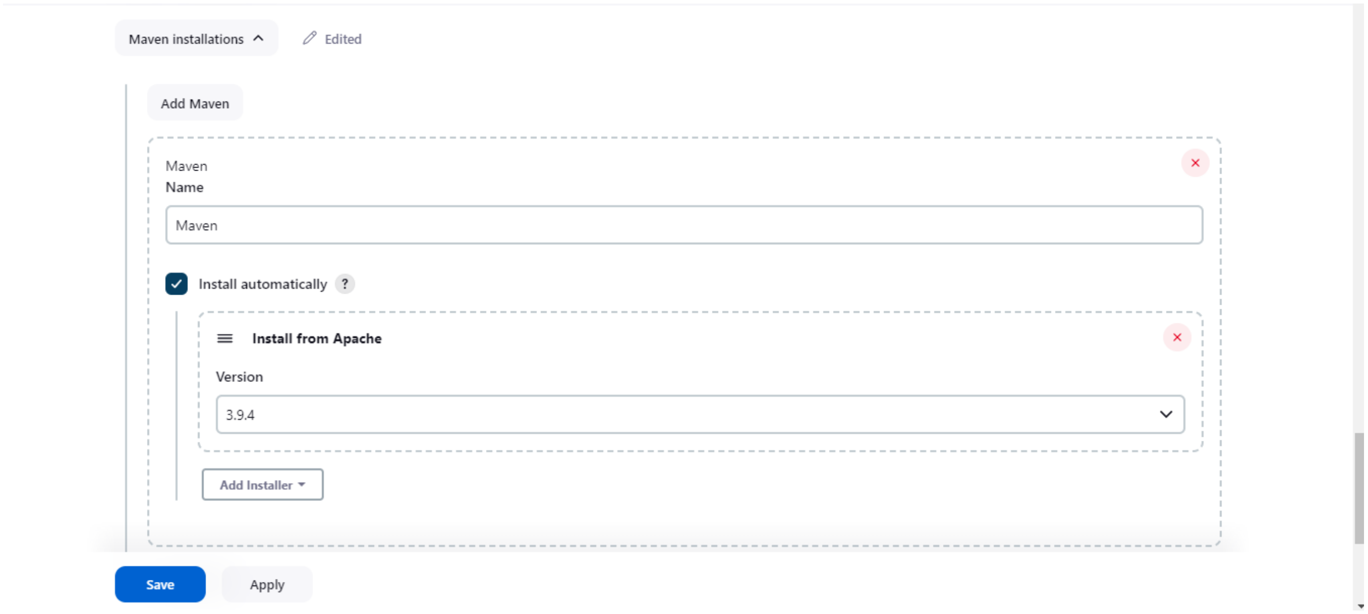
Prerequisites: Maven & Docker configuration

NOTE: To run the examples, modify the `Dockerfile` and `Jenkinsfile` content accordingly (Provided in the example section)

Maven Configuration

- Go to manage jenkins -> plugins
- Under Available plugins -> Choose `Maven Integration` and `Pipeline Maven Integration` and install
- Once it is installed, configure it in `Manage Jenkins -> tools`
- Scroll to the `Maven Installation` section -> `Maven Installation`

- Give name as `Maven` and check `install automatically`
- Apply and Save



Maven
Name: Maven

Install automatically: ✓

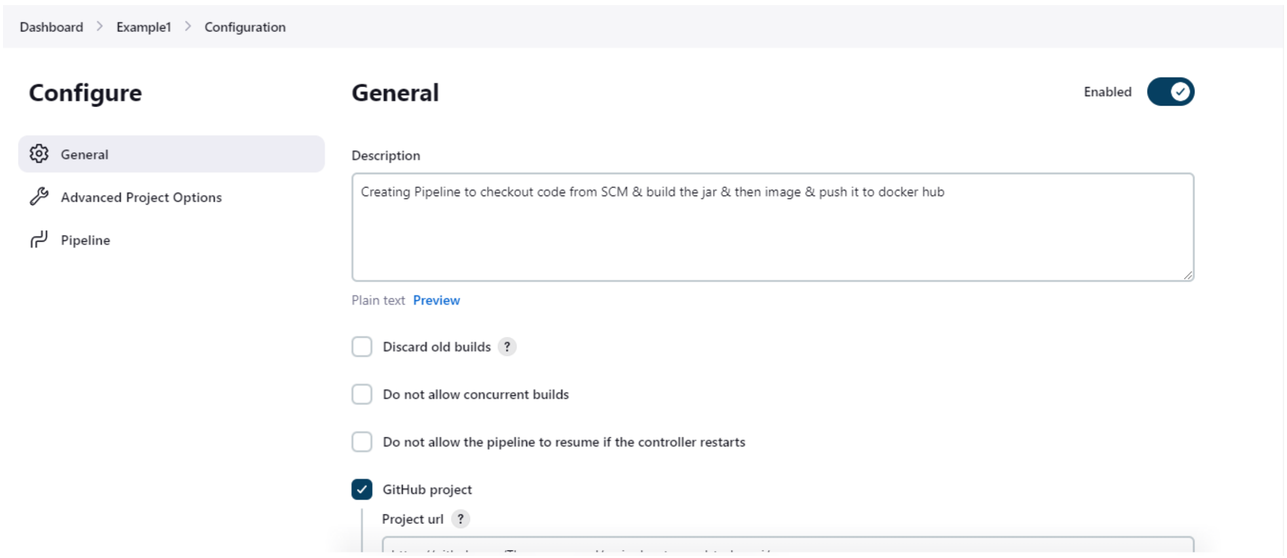
Install from Apache
Version: 3.9.4

Docker configuration

- Since we are running `dnd` image. We dont have to install any docker additional plugins
- We use docker deamon running on host machine itself for `docker`
- We only need to configure the docker hub repository credentials
- Go to `Manage Jenkins -> Credentials`
- Click on `global`
- Now click on `Add Credentials`
- Select kind as `Username and password`
- Scope keep it default to `global`
- Enter dockerhub username `************`
- Enter password - ********
- Give description as `Docker credentials`
- click on create

Example 1: Creating Pipeline to checkout code from SCM & build the jar & then image & push it to docker hub

1. Click on new item
2. Give name as `Example1`
3. Select `Pipeline`
4. Click `ok`
5. Now give decription as `Creating Pipeline to checkout code from SCM & build the jar & then image & push it to docker hub`



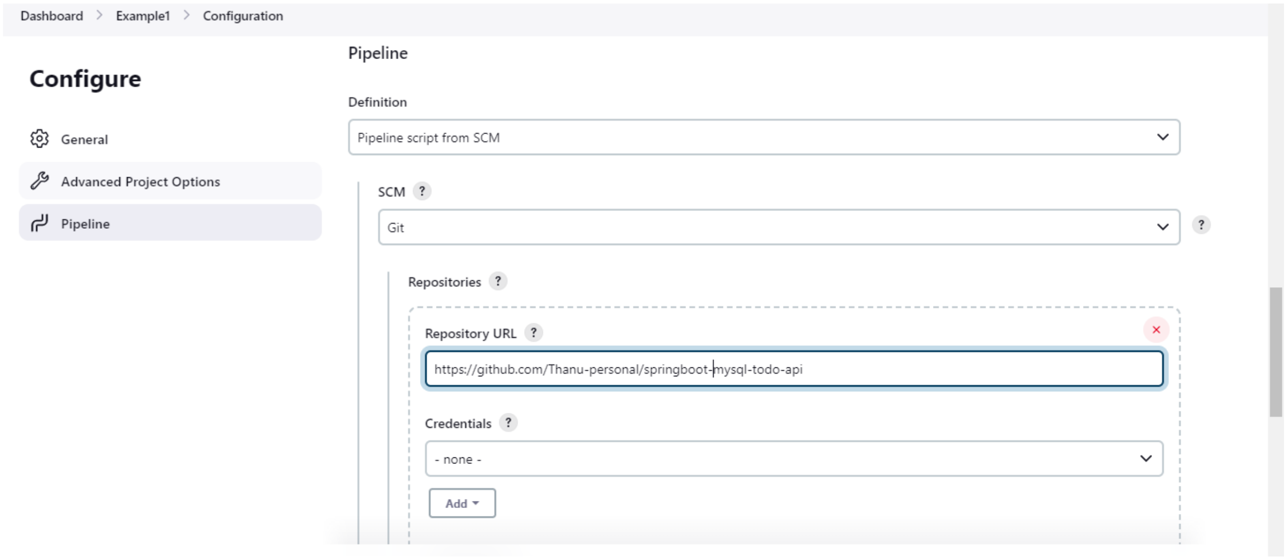
General

Description: Creating Pipeline to checkout code from SCM & build the jar & then image & push it to docker hub

- [ ] Discard old builds
- [ ] Do not allow concurrent builds
- [ ] Do not allow the pipeline to resume if the controller restarts
- [x] GitHub project

Project url

6. Scroll down to pipeline
7. Select `Pipeline from SCM`



Pipeline

Definition: Pipeline script from SCM

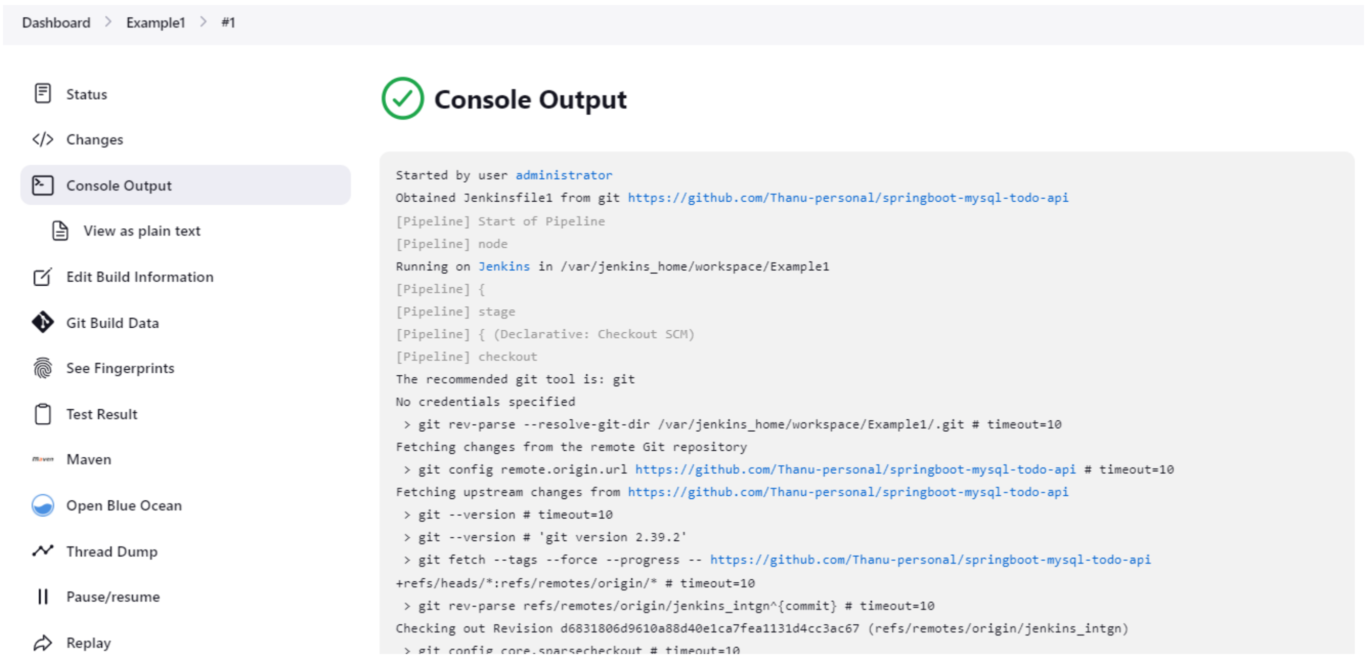
SCM: Git

Repositories

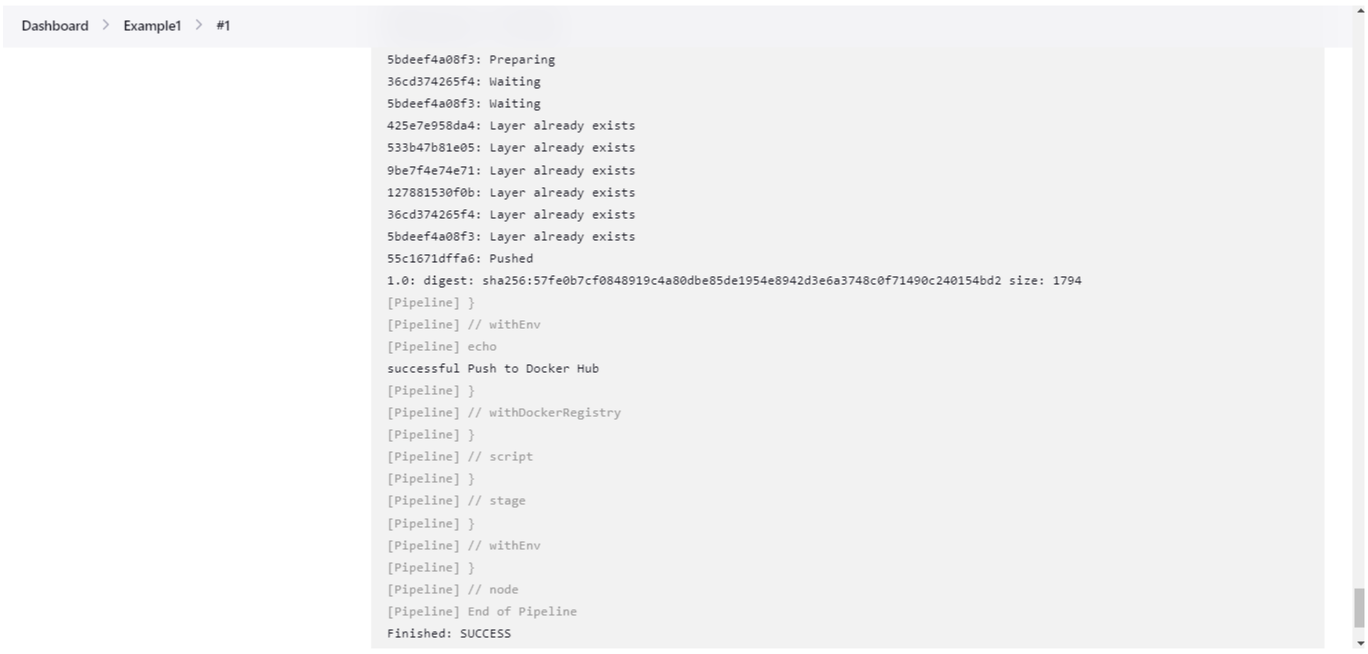
Repository URL: https://github.com/Thanu-personal/springboot-mysql-todo-api

Credentials: - none -

8. Select SCM as `Git`
9. Give your project repository url `https://github.com/vijaynvb/todoapi.git`
10. Select Branch as */master. Create a jenkinsfile in the project root folder and add the below content (Given below as, JenkinsFile Content)
11. Scroll down to script path & type the jenkinsfile you want to execute.
12. Apply and save
13. Now select your pipeline `Example1` and click on `Build now`
14. You can check the status of your build in `console output`



```
Started by user administrator
Obtained Jenkinsfile1 from git https://github.com/Thanu-personal/springboot-mysql-todo-api
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/Example1
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Checkout SCM)
[Pipeline] checkout
The recommended git tool is: git
No credentials specified
 > git rev-parse --resolve-git-dir /var/jenkins_home/workspace/Example1/.git # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url https://github.com/Thanu-personal/springboot-mysql-todo-api # timeout=10
Fetching upstream changes from https://github.com/Thanu-personal/springboot-mysql-todo-api
 > git --version # timeout=10
 > git --version # 'git version 2.39.2'
 > git fetch --tags --force --progress -- https://github.com/Thanu-personal/springboot-mysql-todo-api
+refs/heads/*:refs/remotes/origin/* # timeout=10
 > git rev-parse refs/remotes/origin/jenkins_intgn^{commit} # timeout=10
Checking out Revision d6831806d9610a88d40e1ca7fea1131d4cc3ac67 (refs/remotes/origin/jenkins_intgn)
 > git config core.sparsecheckout # timeout=10
```



```
5bdeef4a08f3: Preparing
36cd374265f4: Waiting
5bdeef4a08f3: Waiting
425e7e958da4: Layer already exists
533b47b81e05: Layer already exists
9be7f4e74e71: Layer already exists
127881530f0b: Layer already exists
36cd374265f4: Layer already exists
5bdeef4a08f3: Layer already exists
55c1671dffa6: Pushed
1.0: digest: sha256:57fe0b7cf0848919c4a80dbe85de1954e8942d3e6a3748c0f71490c240154bd2 size: 1794
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] echo
successful Push to Docker Hub
[Pipeline] }
[Pipeline] // withDockerRegistry
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

NOTE: You can use `pipeline script to generate script for you`

Dockerfile content:

```
FROM openjdk:11.0.15-jre
ADD target/*.jar app.jar
ENTRYPOINT ["java","-jar","app.jar"]
```

Jenkinsfile Content:

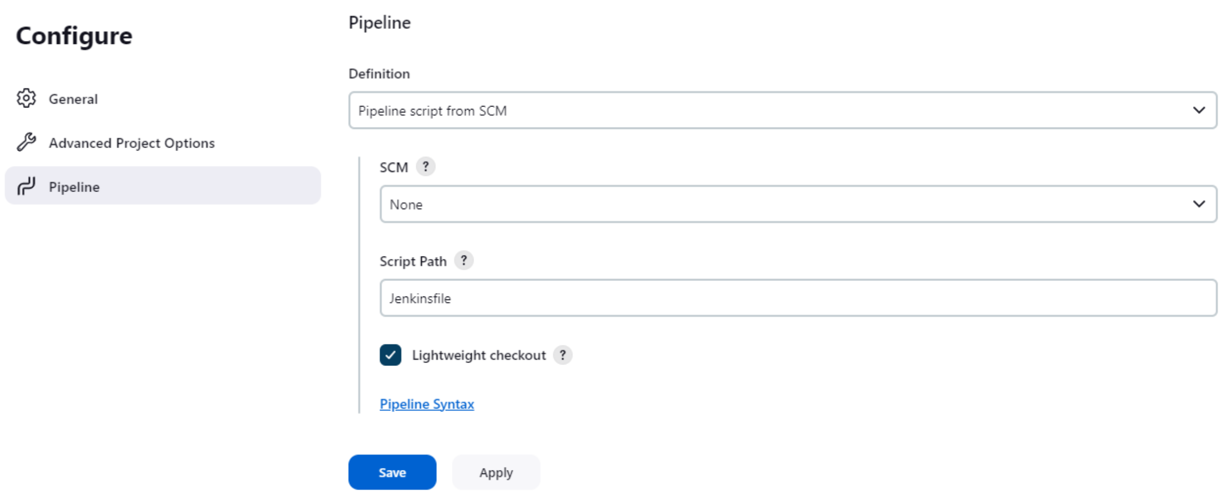
```
#File: Jenkinsfile1
pipeline {
    agent any
    stages {
        stage('Checkout') {
            steps {
                checkout scmGit(branches: [[name: '*/master']], extensions: [],
userRemoteConfigs: [[url: 'https://github.com/vijaynvb/todoapi.git']])
                echo 'successful checkout'
            }
        }

        stage('Build and Package') {
            steps {
                withMaven(maven: 'Maven') {
                    sh 'mvn clean install'
                  echo 'successful Build and Package'
                }
            }
        }
        stage('Build Docker Image') {
            steps {
                script {
                    def imageTag = "<username/imagename:tag>"
                    docker.build(imageTag, '.')
                    echo 'successful Build Docker Image'
                }
            }
        }

        stage('Push to Docker Hub') {
            steps {
                script {
                    withDockerRegistry(credentialsId: '2536b1eb-a541-44c2-b1be-
848c5d6451c9', url: 'https://index.docker.io/v1/') {
                        def imageTag = "<username/imagename:tag>"
                        docker.image(imageTag).push()
                        echo 'successful Push to Docker Hub'
                    }
                }
            }
        }

    }
}
```

Generate checkout script using piepline syntax

To generate,

- `Example1` -> `Configure` -> scroll to the `Pipeline` section

- Click on `pipeline syntax`



Pipeline

Definition: Pipeline script from SCM

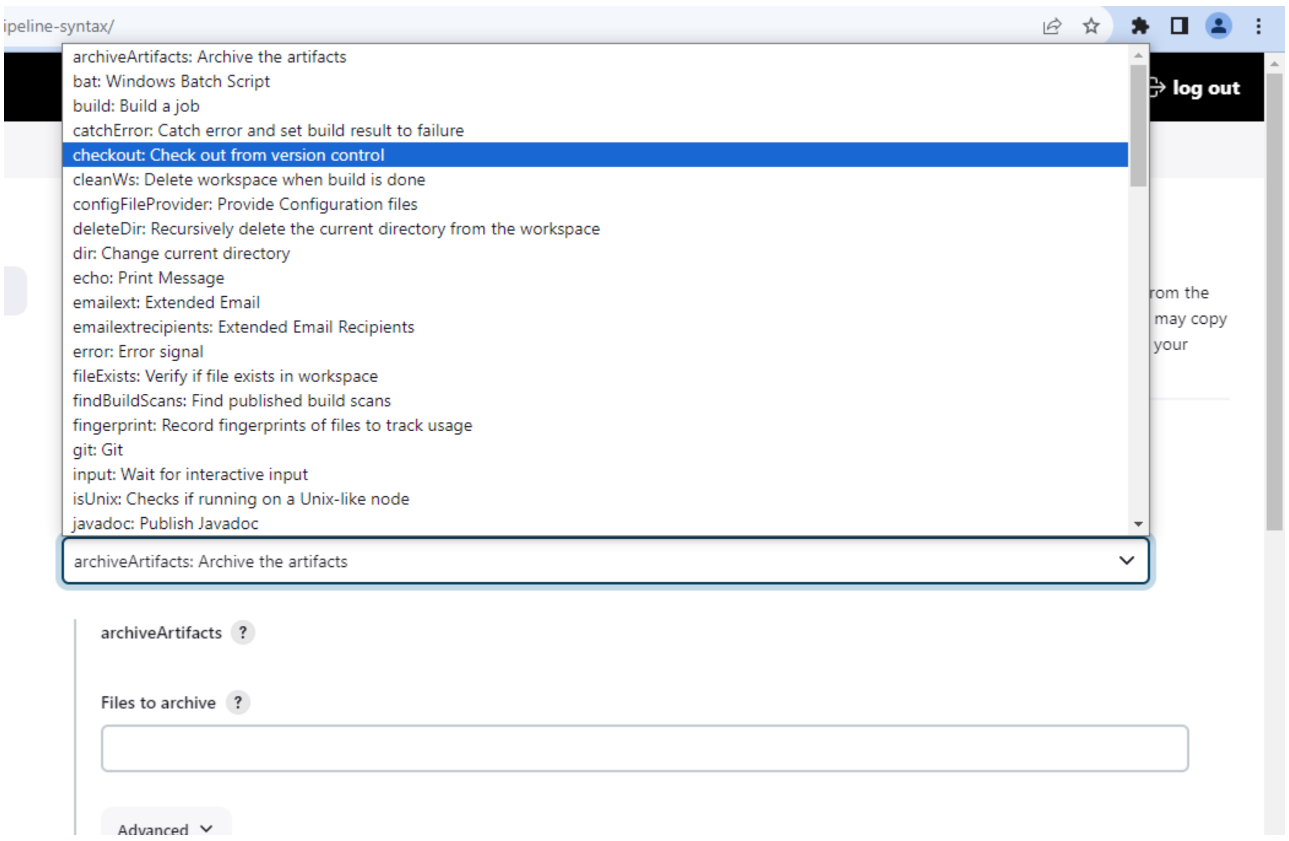
SCM: None

Script Path: Jenkinsfile

Lightweight checkout ✓

Pipeline Syntax

- Select `checkout: checkout from version control`



archiveArtifacts: Archive the artifacts
bat: Windows Batch Script
build: Build a job
catchError: Catch error and set build result to failure
checkout: Check out from version control
cleanWs: Delete workspace when build is done
configFileProvider: Provide Configuration files
deleteDir: Recursively delete the current directory from the workspace
dir: Change current directory
echo: Print Message
emailext: Extended Email
emailextrecipients: Extended Email Recipients
error: Error signal
fileExists: Verify if file exists in workspace
findBuildScans: Find published build scans
fingerprint: Record fingerprints of files to track usage
git: Git
input: Wait for interactive input
isUnix: Checks if running on a Unix-like node
javadoc: Publish Javadoc

archiveArtifacts: Archive the artifacts

archiveArtifacts

Files to archive

- Select `SCM` as `GIT`
- Enter your git repo url - `https://github.com/vijaynvb/todoapi.git`
- Enter credentials (if private) - `none`
- Select the branch `*/master`
- Click on Generate pipeline syntax
- Replace it in checkout scm line in your `Jenkinsfile`

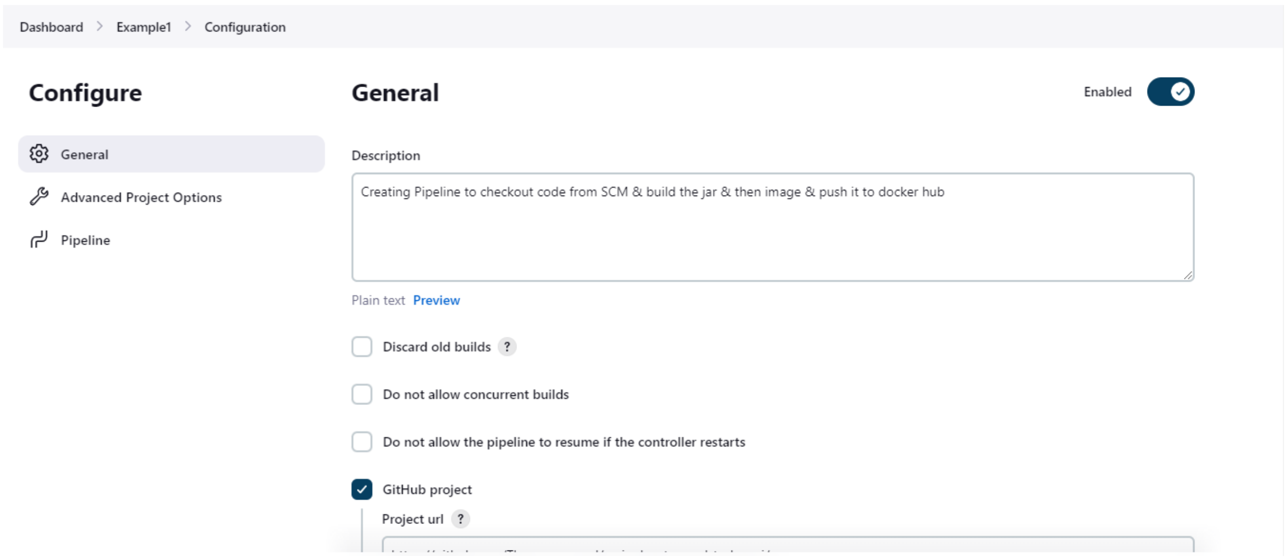
Generate withDockerRegistry script using pipeline syntax

To generate,

- `Example1` -> `Configure` -> scroll to the `Pipeline` section
- Click on `pipeline syntax`
- Select `withDockerRegistry: setup docker registry end point`
- Enter Docker registry url - `https://index.docker.io/v1/`
- Registry credentials `Docker credentials` (which we added earlier)
- Click on Generate pipeline syntax
- Replace it in `withDockerRegistry` line in your `Jenkinsfile`

Example 2: Creating Pipeline to checkout code from SCM & build image from docker file & push it to docker hub

1. Click on new item
2. Give name as `Example2`
3. Select `Pipeline`
4. Click `ok`
5. Now give decription as `Creating Pipeline to checkout code from SCM & build image from docker file & push it to docker hub`



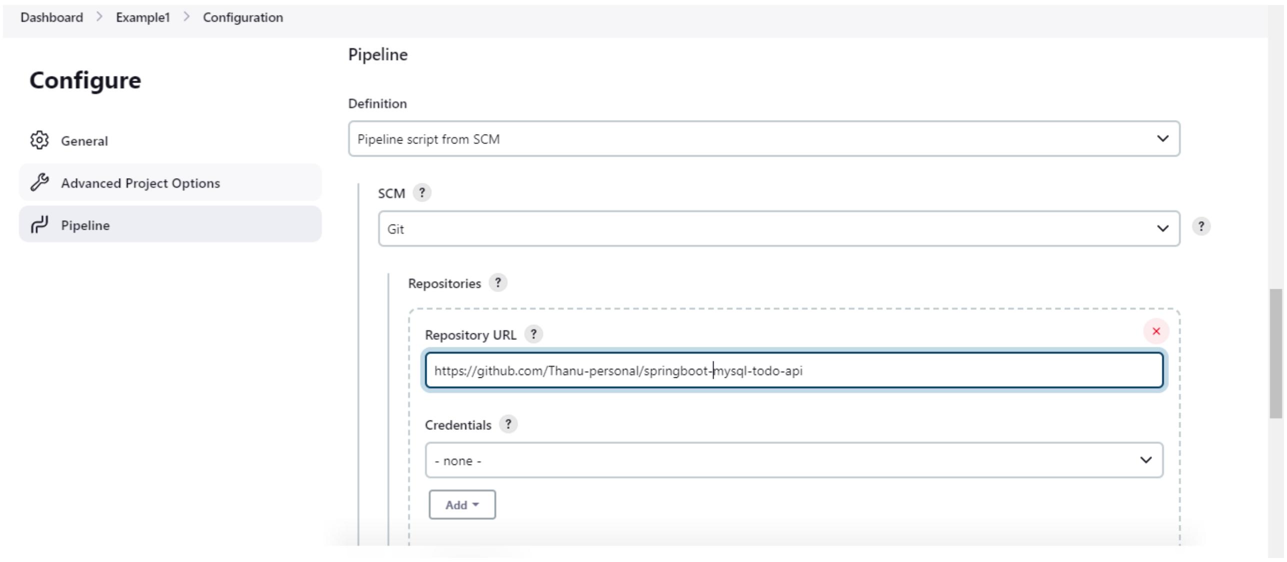
Description: Creating Pipeline to checkout code from SCM & build the jar & then image & push it to docker hub

- [ ] Discard old builds
- [ ] Do not allow concurrent builds
- [ ] Do not allow the pipeline to resume if the controller restarts
- [x] GitHub project

Project url

6. Scroll down to pipeline

7. Select `Pipeline from SCM`


Dashboard > Example1 > Configuration

Configure

Pipeline

Definition: Pipeline script from SCM

SCM: Git

Repositories

Repository URL: https://github.com/Thanu-personal/springboot-mysql-todo-api

Credentials: - none -

8. Select SCM as `Git`
9. Give your project repository url `https://github.com/vijaynvb/todoapi.git`
10. Select Branch as `*/jenkins_intgn`
11. Scroll down to script path & type the jenkinsfile you want to execute.
12. Apply and save
13. Now select your pipeline `Example2` and click on `Build now`
14. You can check the status of your build in `console output`

Dockerfile Content:

```
# Use an official Maven image as the base image for building
FROM maven:3.8-openjdk-11 AS build

# Set the working directory inside the container
WORKDIR /app

# Copy the Maven project definition (pom.xml) into the container
COPY pom.xml .

# Download the project dependencies
RUN mvn dependency:go-offline

# Copy the application source code into the container
COPY src ./src

# Build the Spring Boot application JAR
RUN mvn package

# Use an official OpenJDK runtime image as the base image
FROM openjdk:11-jre-slim

# Set the working directory inside the container
WORKDIR /app

# Copy the Spring Boot JAR from the build stage into the container
```

```
COPY --from=build /app/target/*.jar app.jar

# Expose the port that the Spring Boot app will listen on
EXPOSE 8081

# Specify the command to run the Spring Boot app when the container starts
CMD ["java", "-jar", "app.jar"]
```

Jenkinsfile Content:

```
pipeline {
    agent any

    stages {
        stage('Checkout') {
            steps {
                 checkout scmGit(branches: [[name: '*/master']], extensions: [],
userRemoteConfigs: [[url: 'https://github.com/vijaynvb/todoapi.git']])
                echo 'successful checkout'
            }
        }

        stage('Build jar and image using Docker file ') {
            steps {
                script {
                     def imageTag = "<username/imagename:tag>"
                    docker.build(imageTag, '.')
                    echo 'successful Build Docker Image'
                }
            }
        }

        stage('Push to Docker Hub') {
            steps {
                script {
                    withDockerRegistry(credentialsId: 'ba01f7fb-97f4-4039-bd96-
e8ce7964251e', url: 'https://index.docker.io/v1/') {
                        def imageTag = "<username/imagename:tag>"
                        docker.image(imageTag).push()
                        echo 'successful Push to Docker Hub'
                    }
                }
            }
        }

    }

}
```

Example 3: Creating Pipeline to pull the image from the docker hub and deploy it in kubernetes cluster

- Before we start, make sure jenkins is configured to work with kubernetes

1. Click on new item
2. Give name as `Example3`
3. Select `Pipeline`
4. Click `ok`
5. Now give decription as `Creating Pipeline to pull the image from the docker hub and deploy it in kubernetes cluster`
6. Scroll down to pipeline
7. Select `Pipeline from SCM`
8. Select SCM as `Git`
9. Give your project repository url `https://github.com/vijaynvb/todoapi.git`
10. Select Branch as `*/master` [Make sure you have kubernetes-config.yaml file]
11. Scroll down to script path & type the jenkinsfile you want to execute. [NOTE: Click on pipeline script to generate pipeline script to use kubectl]
12. Replace the command in `pipeline script`
13. Apply and save
14. Now select your pipeline `Example3` and click on `Build now`
15. You can check the status of your build in `console output`

Jenkinsfile content

```
pipeline {
    agent any


   stages {
        stage('Checkout') {
            steps {
                // Replace your generated pipeline script here
                 checkout scmGit(branches: [[name: '*/master']], extensions: [],
userRemoteConfigs: [[url: 'https://github.com/vijaynvb/todoapi.git']])
                echo 'successful checkout'
            }
        }

        stage('Build jar and image using Docker file ') {
            steps {
                script {
                    def imageTag = "<username/imagename:tag>"
                    docker.build(imageTag, '.')
                    echo 'successful Build Docker Image'
                }
            }
        }

         stage('Push to Docker Hub') {
            steps {
                script {
                 // Replace your generated pipeline script here
                    // This step should not normally be used in your script.
Consult the inline help for details.
                    withDockerRegistry(credentialsId: '98340929-73a9-44aa-abb6-
1c3faf40de97', url: 'https://index.docker.io/v1/') {
                    def imageTag = "<username/imagename:tag>"
                    docker.image(imageTag).push()
                    echo 'successful Push to Docker Hub'
                    }
                }
            }
        }


        stage('Deploy to Kubernetes') {
            steps {
                script {
                // Replace your generated pipeline script here
                   kubeconfig(credentialsId: 'e8af095b-b076-40b5-97d6-
cf534e7e9d58', serverUrl: 'https://kubernetes.docker.internal:6443') {
                    def kubeConfig = readFile 'kubernetes-config.yaml'
                    sh "kubectl apply -f kubernetes-config.yaml"
                    }
               }
            }
        }
    }

  }
```

Dockerfile content

```
#Use this for Example1 - jenkins
#FROM openjdk:11.0.15-jre
#ADD target/*.jar app.jar
#ENTRYPOINT ["java","-jar","app.jar"]

#Use this for Example2 - jenkins
# Use an official Maven image as the base image for building
FROM maven:3.8-openjdk-11 AS build

# Set the working directory inside the container
WORKDIR /app

# Copy the Maven project definition (pom.xml) into the container
```

```
COPY pom.xml .

# Download the project dependencies
RUN mvn dependency:go-offline

# Copy the application source code into the container
COPY src ./src

# Build the Spring Boot application JAR
RUN mvn package

# Use an official OpenJDK runtime image as the base image
FROM openjdk:11-jre-slim

# Set the working directory inside the container
WORKDIR /app

# Copy the Spring Boot JAR from the build stage into the container
COPY --from=build /app/target/*.jar app.jar

# Expose the port that the Spring Boot app will listen on
EXPOSE 8081

# Specify the command to run the Spring Boot app when the container starts
CMD ["java", "-jar", "app.jar"]
```

kubernetes-config.yaml content

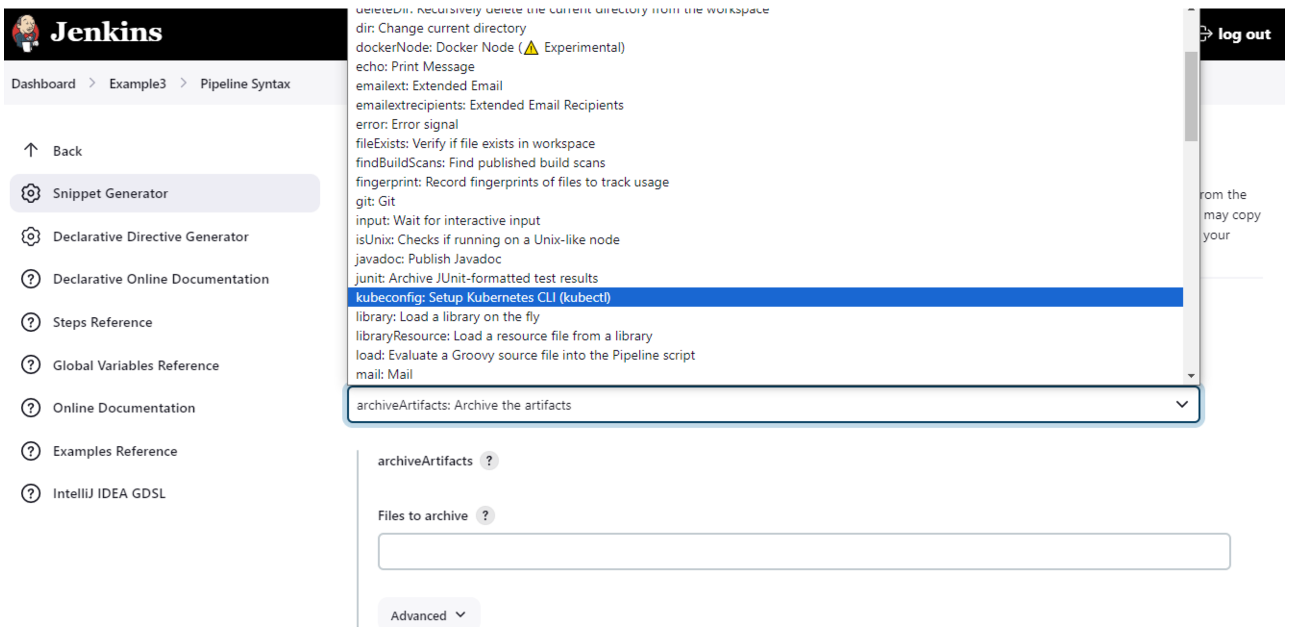
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: todo-app-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: todo-app
  template:
    metadata:
      labels:
        app: todo-app
    spec:
      containers:
        - name: todo-app-container
          image: todo/image/path
          ports:
            - containerPort: 8081

---
apiVersion: v1
```

```
kind: Service
metadata:
  name: todo-app-service
spec:
  selector:
    app: todo-app
  ports:
    - protocol: TCP
      port: 8081
      targetPort: 8081
  type: NodePort
```

Generate checkout script using piepline syntax

To generate,

- `Example3` -> `Configure` -> scroll to the `Pipeline` section
- Click on `pipeline syntax`
- Select `checkout: checkout from version control`


Jenkins
log out
Dashboard > Example3 > Pipeline Syntax
Back
Snippet Generator
Declarative Directive Generator
Declarative Online Documentation
Steps Reference
Global Variables Reference
Online Documentation
Examples Reference
IntelliJ IDEA GDSL
deleteDir: Recursively delete the current directory from the workspace
dir: Change current directory
dockerNode: Docker Node (⚠ Experimental)
echo: Print Message
emailext: Extended Email
emailextrecipients: Extended Email Recipients
error: Error signal
fileExists: Verify if file exists in workspace
findBuildScans: Find published build scans
fingerprint: Record fingerprints of files to track usage
git: Git
input: Wait for interactive input
isUnix: Checks if running on a Unix-like node
javadoc: Publish Javadoc
junit: Archive JUnit-formatted test results
kubeconfig: Setup Kubernetes CLI (kubectl)
library: Load a library on the fly
libraryResource: Load a resource file from a library
load: Evaluate a Groovy source file into the Pipeline script
mail: Mail
archiveArtifacts: Archive the artifacts
archiveArtifacts
Files to archive
Advanced

- Enter the kubernetes end point (To get the kubernetes end point , type `kubectl cluster-info` in command prompt. In my case, `https://kubernetes.docker.internal:6443`)
- Select `Credentials` as `config(Kubernetes_config)`
- Click on Generate pipeline syntax
- Replace it in checkout scm line in your `Jenkinsfile`

Kubernetes Configuration

- Install `Kubernetes` plugin

- Add `kubeconfig` details in credentials
    - Go to `Manage Jenkins`-> `Credentials`
    - Click on system `global` -> Add credentials
    - Select kind as `Secret`
    - Click on `choose file` and locate the file in your host machine (In my case, it is located in C:\Users\Admin.kube\config)
    - Give `description` as `Kubernetes_config` and save

Example 4 : Creating Pipeline to checkout charts and deploy it in kubernetes cluster using HELM

1. Click on new item
2. Give name as `Example4`
3. Select `Pipeline`
4. Click `ok`
5. Now give decription as `Creating Pipeline to checkout charts and deploy it in kubernetes cluster using HELM`
6. Scroll down to pipeline
7. Select `Pipeline from SCM`
8. Select SCM as `Git`
9. Give your project repository url `https://github.com/vijaynvb/todoapi.git`
10. Select Branch as `*/master` [Make sure you have kubernetes-config.yaml file - if not please add (content given below)]
11. Scroll down to script path & type the jenkinsfile you want to execute. [NOTE: Click on pipeline script to generate pipeline script to use kubectl]
12. Replace the command in `pipeline script`
13. Apply and save
14. Now select your pipeline `Example4` and click on `Build now`
15. You can check the status of your build in `console output`

NOTE: The chart im going to deploy is the `Application_helm` in the repo `https://github.com/vijaynvb/helm_project`. It is an umbrella chart which has 3 microservices charts

Jenkinsfile content

```
pipeline {
    agent any
    environment {
        HELM_VERSION = "3.5.3" // Change to the desired Helm version
        CHART_NAME = "Application_helm"
        RELEASE_NAME = "helmdeployment"

    }
   stages {

        stage('Checkout') {
            steps {
                // Replace your generated pipeline script here
                checkout scmGit(branches: [[name: '*/main']], extensions: [],
userRemoteConfigs: [[url: 'https://github.com/vijaynvb/helm_project']])
                echo 'successful checkout'
            }
        }

        stage ('Helm Deploy') {
          steps {
            script {
                // Replace your generated pipeline script here
                kubeconfig(credentialsId: 'e8af095b-b076-40b5-97d6-cf534e7e9d58',
serverUrl: 'https://kubernetes.docker.internal:6443') {
                sh "helm install ${RELEASE_NAME} ./${CHART_NAME}"
                }
            }
        }
    }
    }

  }
```

kubernetes-config.yaml content

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: todo-app-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: todo-app
  template:
    metadata:
      labels:
        app: todo-app
    spec:
      containers:
        - name: todo-app-container
          image: todo/image/path
          ports:
            - containerPort: 8081

---
apiVersion: v1
kind: Service
metadata:
  name: todo-app-service
```

```
spec:
  selector:
    app: todo-app
  ports:
    - protocol: TCP
      port: 8081
      targetPort: 8081
  type: NodePort
```