# JENKINS

## To run with Docker:

```
docker run -p 8080:8080 -p 50000:50000 --restart=on-failure -v
jenkins_home:/var/jenkins_home jenkins/jenkins:lts-jdk17
```

↳ save admin pass
↳ go to localhost:8080
↳ install suggested plugins
↳ continue as admin

UDEMY Jenkins: Beginner To Pro

# CI


CI (continuous integration)

CODE → BUILD → TEST

commit

Dev

if test fails dev commits a new change

- Not efficient to build on local machine
  - Jenkins turns the steps into pipeline

When dev commits, Jenkins scans for new commits and starts a Build

---

# CD

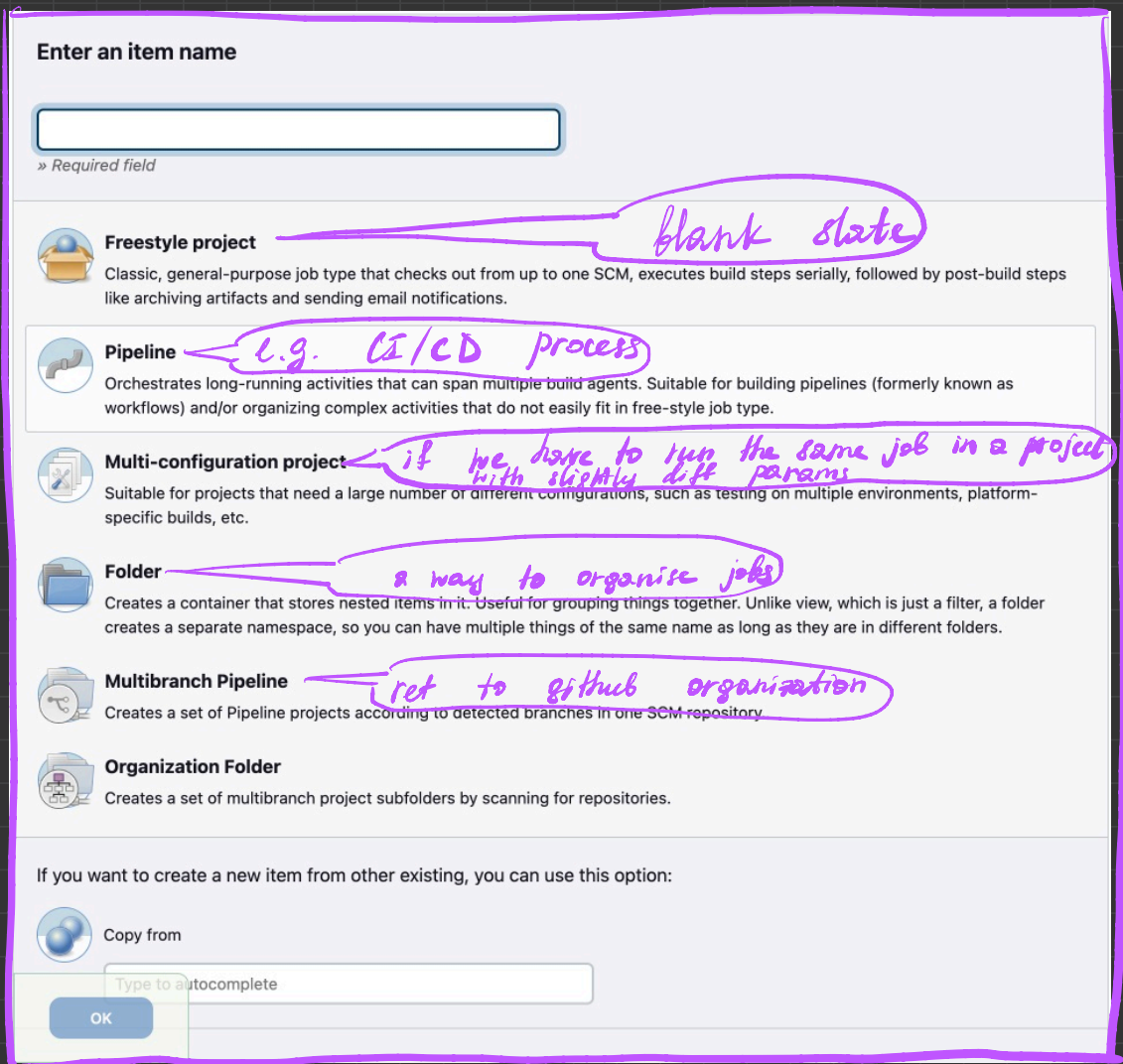
CI

Where devops start getting involved

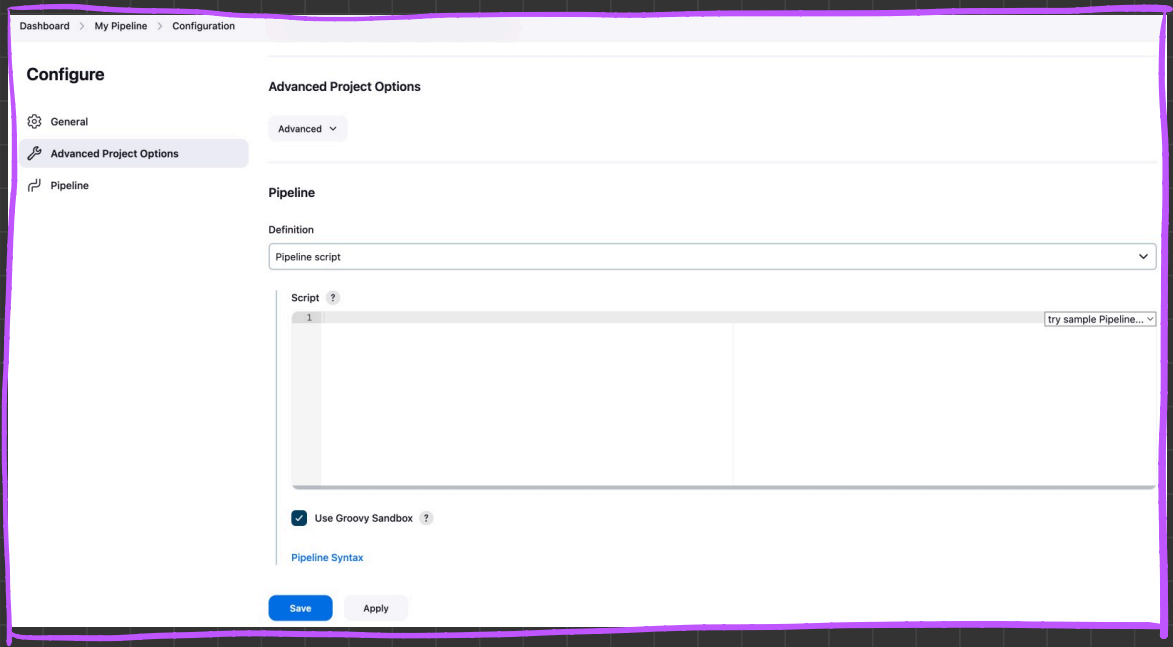
CD

Build Test → Build Docker | Push | k8s | Deploy

# CREATE NEW JOB



# CREATE PIPELINE EDITOR




It's better to use VS code bc the pipeline editor is bad in Jenkins

# CREATE PIPELINE

Jenkinsfile

```
// any jenkins pipeline starts with pipeline statement
pipeline {
  // agent specifies where the pipeline will be executed
  agent any
  // stages block contains all the stages in the pipeline
  stages {
    // jenkins stage to clean up the workspace, because when jenkins runs the job it creates a folder
    // better to put in the beginning otherwise if the job fails, the workspace won't be cleaned up
    stage("Clean Up"){
      steps {
        deleteDir() // deletes the workspace
      }
    }
    // stage to clone the repository
    stage("Clone Repo"){
      steps {
        // run bash shell
        sh "git clone https://github.com/jenkins-docs/simple-java-maven-app.git"
      }
    }
    // stage to build
    stage("Build"){
      steps {
        // each steps reverts back to the original workspace
        // change directory to the cloned repository
        // we don't use cd because it's persistent and every step will be inside that directory
        dir("simple-java-maven-app"){
          // the following builds java file and cleans libraries/binaries that may have been created before
          sh "mvn clean package"
        }
      }
    }
    stage("Test"){
      steps {
        dir("simple-java-maven-app"){
          // run tests
          sh "mvn test"
        }
      }
    }
  }
}
```

→ then copy/paste to jenkins

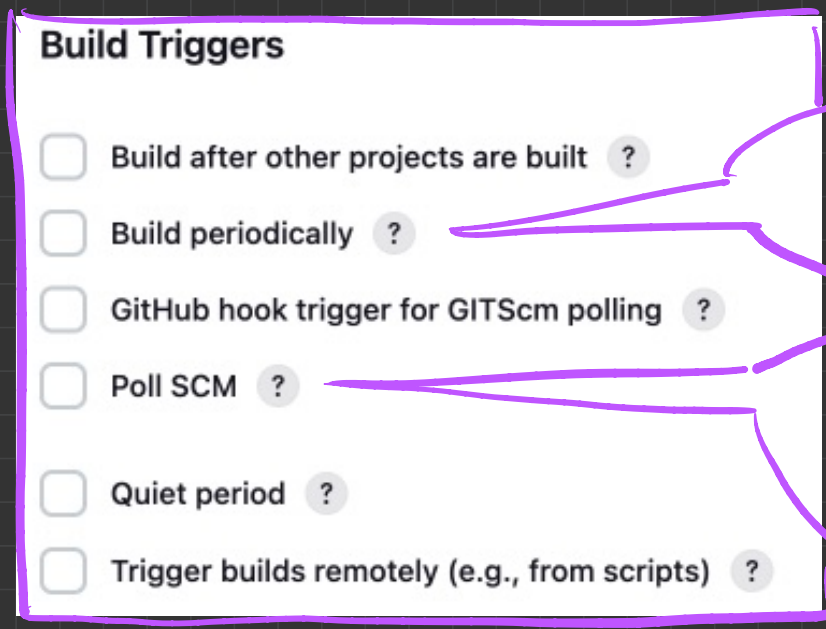
# INSTALL MAVEN

1. docker container ls
   ls id of jenkins container
2. docker exec -it -u root id /bin/bash
3. apt-get update
4. apt-get install maven

# CREATE PIPELINE

## BUILD TRIGGERS

→ they say when to run this job

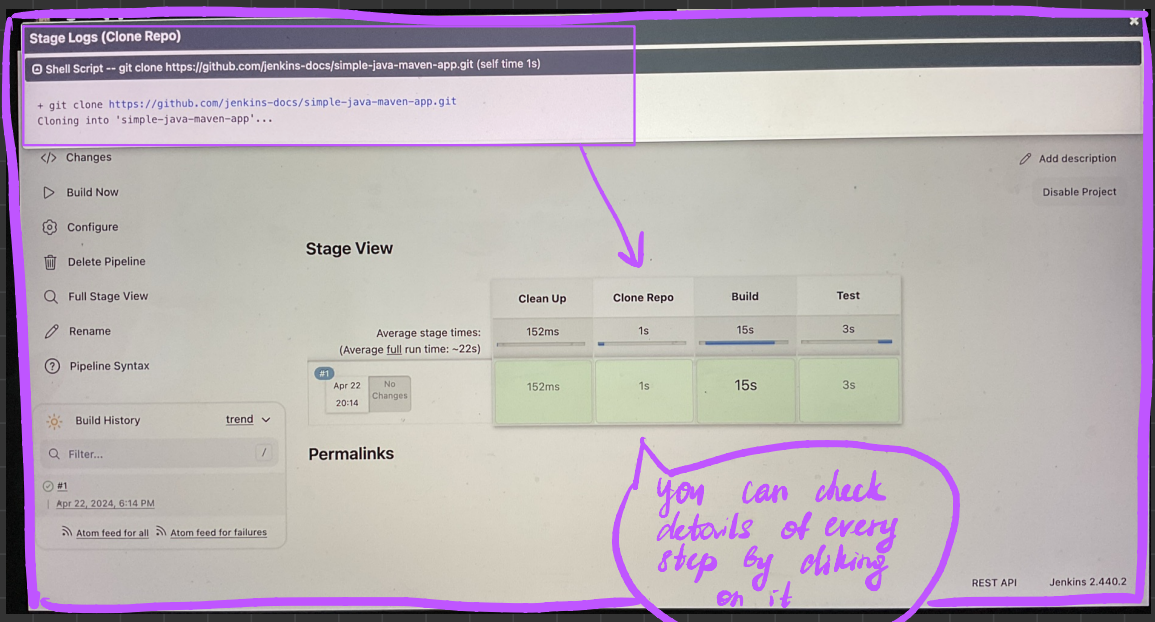


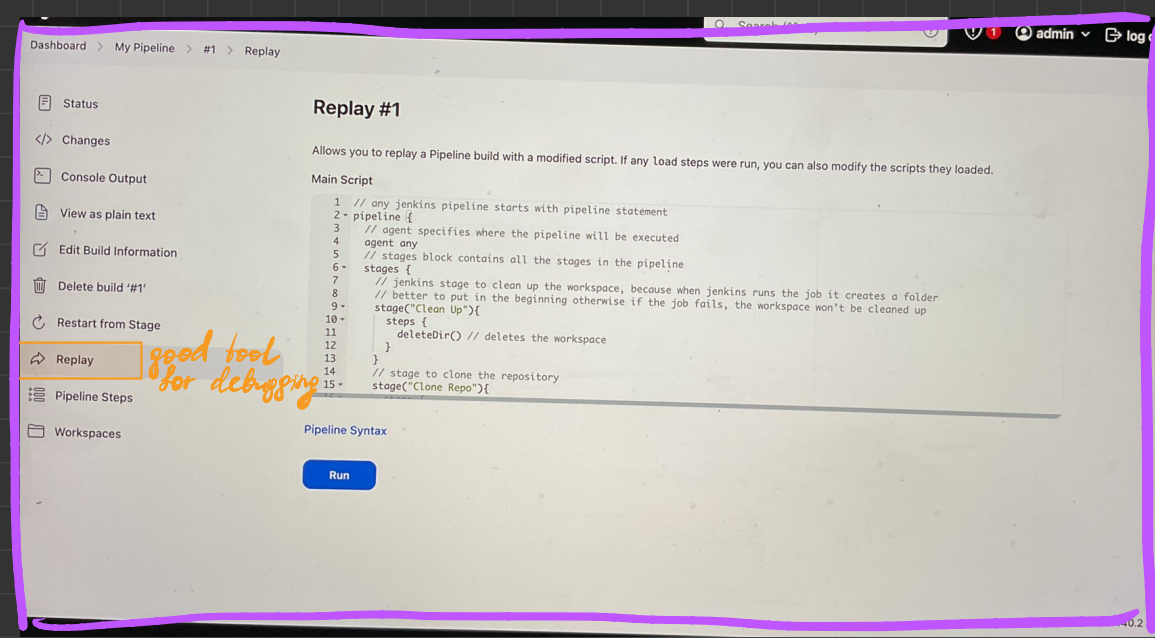

e.g. cleanup job (morning/evening)

e.g. runs the job when the change is made to the source code

# CREATE PIPELINE

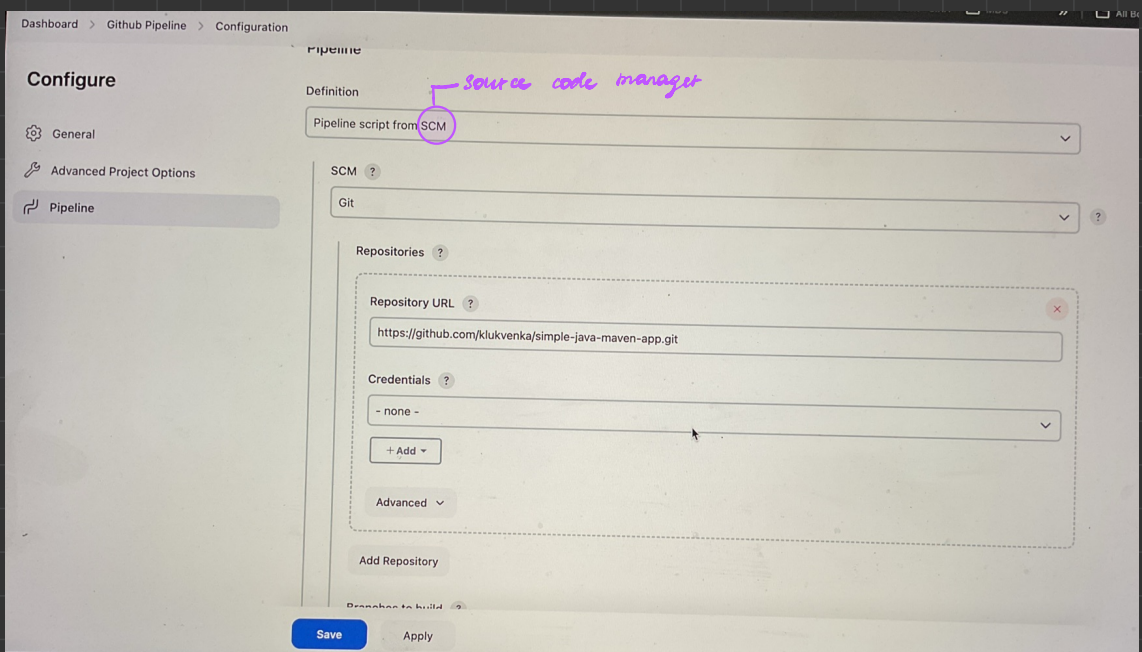
## | BUILD / REPLAY



# PIPELINE.GITHUB

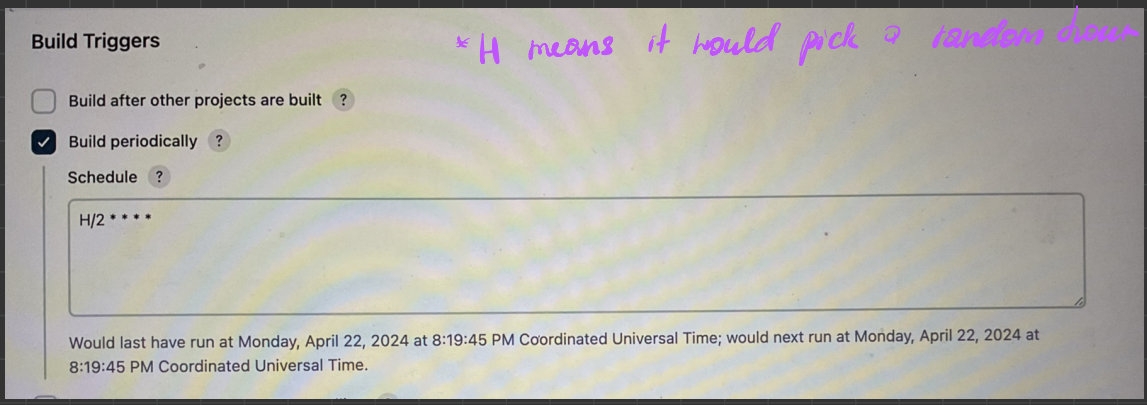
* You should already have a github repo with jenkinsfile in it




1. Now jenkins automatically will clone the repo
2. Then will load jenkinsfile that creates the pipeline

# PIPELINE.GITHUB

## auto builds



# MULTIBRANCH PIPELINE

Allows to implement different Jenkinsfiles for different branches of same project

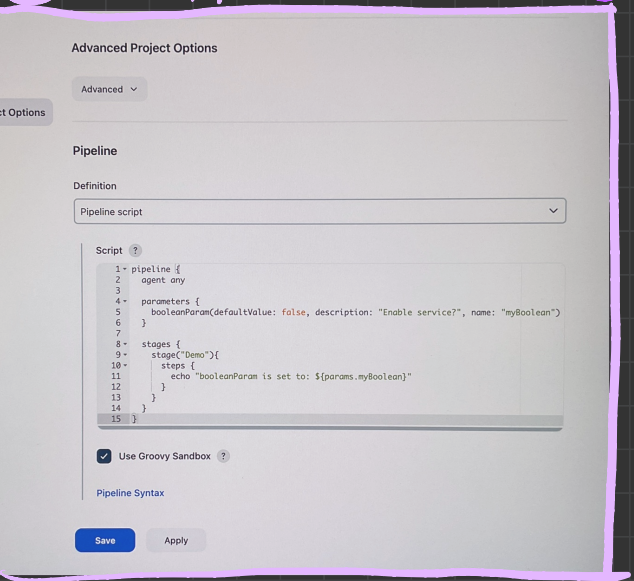
# PARAMETERIZED PIPELINES

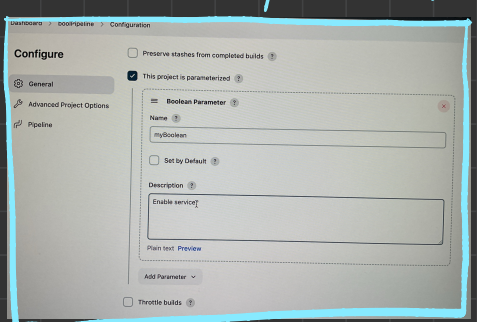
Allows to get **input** before the job runs

- checkbox
- user input
- dropdowns
- ...

## BOOLEAN

1. Create pipeline → 2. Build w/o params for the 1st time → 3. Then you see params in config (Jenkins picked up the info)




↳ 4. Click "Build with params"

# PARAMETERIZED PIPELINES

## STRING

params > input > Jenkinsfile

```groovy
pipeline {
  agent any

  parameters {
    // there is also a text parameter that supports multi-line text
    string(defaultValue: "TEST", description: "Which environment to deploy in?", name: "deployEnv")
  }

  stages {
    stage("Demo"){
      steps {
        echo "string is set to: ${params.deployEnv}"
      }
    }
  }
}
```

## DROPDOWN

params > choice > Jenkinsfile

```groovy
pipeline {
  agent any

  parameters {
    choice(choices: ["TEST", "DEV", "QA", "PRE_PROD", "PROD"], description: "Which environment to deploy in?", name: "deployEnv")
  }

  stages {
    stage("Demo"){
      steps {
        echo "Choice is set to: ${params.deployEnv}"
      }
    }
  }
}
```

# VARIABLES

this is how to use regular variables:

variables > Jenkinsfile

```
pipeline {
  agent any

  environment {
    // define variables here
    def myString = "Hello World"
    def myNumber = 10
    def myBool = true
  }

  stages {
    stage("Demo"){
      steps {
        echo "myString: ${myString}"
        echo "myNumber: ${myNumber}"
        echo "myBool: ${myBool}"
      }
    }
  }
}
```

# ENV VARS

https://www.jenkins.io/doc/book/pipeline/jenkinsfile/#using-environment-variables

list of Jenkins env vars available

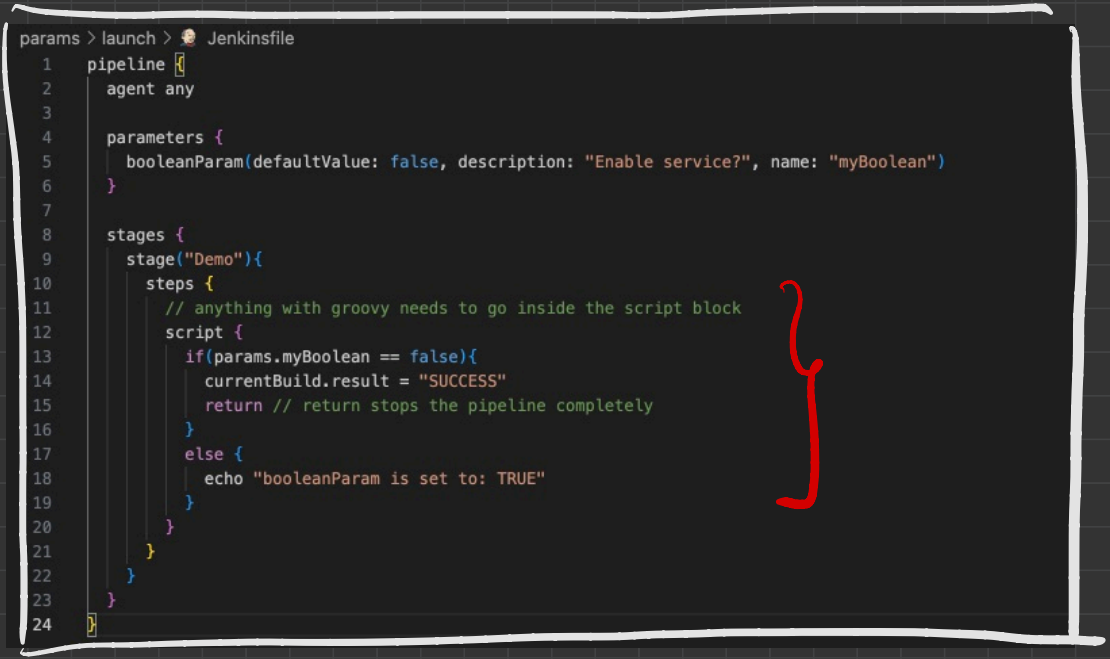
Script ?

```
pipeline {
    agent any
    stages {
        stage('Example') {
            steps {
                // e.g. you can use build number as a tag when working with docker
                echo "Build number: ${env.BUILD_NUMBER}"
            }
        }
    }
}
```

# GROOVY

Docs → https://groovy-lang.org/syntax.html

- Jenkins uses Groovy behind the scenes
- Really helpful for complex pipelines



params > launch > Jenkinsfile

```
pipeline {
  agent any

  parameters {
    booleanParam(defaultValue: false, description: "Enable service?", name: "myBoolean")
  }

  stages {
    stage("Demo"){
      steps {
        // anything with groovy needs to go inside the script block
        script {
          if(params.myBoolean == false){
            currentBuild.result = "SUCCESS"
            return // return stops the pipeline completely
          }
          else {
            echo "booleanParam is set to: TRUE"
          }
        }
      }
    }
  }
}
```

# FUNCTIONS

func > Jenkinsfile

```groovy
pipeline {
  agent any

  env {
    def myNumber = 123
  }

  stages{
    stage("Demo"){
      steps {
        myFunc("Hello from the demo stage!")
      }
    }
  }
}

def myFunc(String myText) {
  echo "myText is set to: ${myText}"
}
```

can't be used outside of the pipeline SCOPE

e.g. can't be used here

Vars defined inside a stage can be used only in also that stage

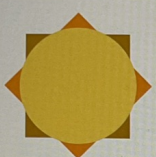
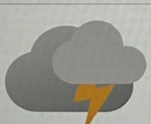
# MULTILINE BASH

```groovy
pipeline {
  agent any

  stages{
    stage("Demo"){
      steps {
        sh """
          echo "Hello from the shell"
        """
      }
    }
  }
}
```

# BUILD HEALTH

A useful indication on builds status.
E.g. if we can notice it goes from Sunny to Rainy the problem there

| Icon | Health |
| --- | --- |
|  | **Sunny**, more than 80% of Runs passing |
|  | **Partially Sunny**, 61% to 80% of Runs passing |
|  | **Cloudy**, 41% to 60% of Runs passing |
|  | **Raining**, 21% to 40% of Runs passing |
|  | **Storm**, less than 21% of Runs passing |

# CREDENTIALS

https://www.jenkins.io/doc/book/pipeline/jenkinsfile/#handling-credentials

There are many guides depending on what platform we are using

Advice for AWS:

When Jenkins's running on EC2 instances on AWS, instead of using credentials, assign the role to the node

# JOB FROM A JOB

```
41      stage("Build Remote"){
42        steps {
43          build 'boolPipeline'
44        }
45      }
46    }
47  }
```

name of the job

## To set a parameter value:

```
28    stage("Build Remote"){
29      steps {
30        build job: 'boolPipeline', parameters: [[$class: 'BooleanParameterValue', name: 'myBoolean', value: true]]
31      }
32    }
33  }
34 }
```