Devops Labmanual(R20)

DBMS (Jawaharlal Nehru Technological University, Kakinada)

KRISHNA CHAITANYA INSTITUTE OF TECHNOLOGY & SCIENCES
Devarajugattu (Post) , Peddaraveedu (Mandal), Prakasam Dist. - 523 320.
(Approved by A.I.C.T.E., New Delhi, & Affiliated to JNTUK, Kakinada)
NAAC ACCREDITED INSTITUTION

CERTIFICATE

This is to **c**ertify that this is a bonafide work done by Mr/Miss ……..… Bearing Roll No …………………………………of…………......……B.Tech in……………Branch for the …………………………………. Laboratory Course ……………..…………………. During the Academic Year 2022-2023.

No. of Experiments Recorded: ……………………

Marks Awarded: …………………………………………..

Signature of the Staff In-charge **Signature of HOD**

Date: ……………….. Date: ……………

Signatures:

1. Internal Examiner ____________________________________
2. External Examiner ____________________________________

INDEX

EXP.NO	DATE	Name Of The Experiment	PAGE NO	SIGNATURE

c

CONTINEOUS INTERATION &CONTINEOUS DELIVERY USING DevOps

(SKILL ORIENTED COURSE)

Experiment: 01

Software Engineering and Agile Software Development

In earlier days Iterative Waterfall model was very popular to complete a project. But nowadays developers face various problems while using it to develop software. The main difficulties included handling change requests from customers during project development and the high cost and time required to incorporate these changes. To overcome these drawbacks of Waterfall model, in the mid-1990s the Agile Software Development model was proposed.

The Agile model was primarily designed to help a project to adapt to change requests quickly. So, the main aim of the Agile model is to facilitate quick project completion. To accomplish this task agility is required. Agility is achieved by fitting the process to the project, removing activities that may not be essential for a specific project. Also, anything that is waste of time and effort is avoided.

Actually, Agile model refers to a group of development processes. These processes share some basic characteristics but do have certain subtle differences among themselves. A few Agile SDLC models are given below:

- Crystal
- Atern
- Feature-driven development
- Scrum
- Extreme programming (XP)
- Lean development
- Unified process

Agile model is the combination of iterative and incremental process models. The steps involve in agile SDLC models are:

- Requirement gathering
- Requirement Analysis
- Design
- Coding
- Unit testing
- Acceptance testing

Principles of Agile model:

- To establish close contact with the customer during development and to gain a clear understanding of various requirements, each Agile project usually includes a customer representative on the team. At the end of each iteration

stakeholders and the customer representative review, the progress made and re-evaluate the requirements.
- Agile model relies on working software deployment rather than comprehensive documentation.
- Frequent delivery of incremental versions of the software to the customer representative in intervals of few weeks.
- Requirement change requests from the customer are encouraged and efficiently incorporated.
- It emphasizes on having efficient team members and enhancing communications among them is given more importance. It is realized that enhanced communication among the development team members can be achieved through face-to-face communication rather than through the exchange of formal documents.
- It is recommended that the development team size should be kept small (5 to 9 people) to help the team members meaningfully engage in face-to-face communication and have collaborative work environment.
- Agile development process usually deploys Pair Programming. In Pair programming, two programmers work together at one work-station. One does code while the other reviews the code as it is typed in. The two programmers switch their roles every hour or so.

Advantages:

- Working through Pair programming produce well written compact programs which have fewer errors as compared to programmers working alone.
- It reduces total development time of the whole project.
- Customer representatives get the idea of updated software products after each iteration. So, it is easy for him to change any requirement if needed.

Disadvantages:

- Due to lack of formal documents, it creates confusion and important decisions taken during different phases can be misinterpreted at any time by different team members.
- Due to the absence of proper documentation, when the project completes and the developers are assigned to another project, maintenance of the developed project can become a problem.

QUIZ-1

1. Which of the following statements are true with respect to Kanban?

Ans: - * Kanban is based on the philosophy "stop finishing, start working"

2. Which of the following statements(s) are correct about continuous integration (CI)?

ANS: - * code needed to be frequently checked in

*** CI helps to identify integration defects in the early stages of project**

3. identify the activities (build) that can be automated to create a continuous integration pipeline.

ANS: - static code analysis

4. the most efficient and effective method of conveying information to and within a development team is____________________________.

ANS: - face to face conversation

5. which of the following XP practice ensure 100% code coverage, review and ensure that no extra line of code is written.

ANS: - test driven development

6. Tom dev team member, has mentioned in a daily scrum meeting that he is unable to proceed with his work due to unavailability of a software library. He also started that the library is available in another project team within the company. What would be the most appropriate corrective action in this scenario?

ANS: - scrum master should get the issue resolve by speaking to another project team and make the software library available.

7. which code practice of Kanban helps understand the activities being done and the various stages that lead to completion?

ANS: - visualize the workflow

8. In scrum, I am responsible for the return on investment6, goals and the vision of the project. I am responsible for the product backlog and the release date. Who am i?

ANS: - product owner

9. The IT team management at pura vida company has decided to adopt Devops and has drawn a roadmap for the journey. This information is communicated to all the team members (developers, testers, architects, operations team, (includes infrastructure, system administration and

deployment).

ANS: - * managers express that the buy –in should be there from business (customers) and the top management in pure Vida

*** IT management team members feel that additional roles will be required (other than scrum master, product owner and Dev team) to execute this**

10. which of the following benefits does Agile NOT offer in comparison to waterfall approach?

ANS: - * final product is visible at the end of the project only in Agile method of software development

*** There is a lot of focus on documentation in Agile method of software development**

EXERCISE: - 02

Development & Testing with Agile: Extreme Programming

What is extreme programming in SDLC?

Extreme Programming (XP) is **an agile software development framework that aims to produce higher quality software, and higher quality of life for the development team**. XP is the most specific of the agile frameworks regarding appropriate engineering practices for software development.

Values

The five values of XP are communication, simplicity, feedback, courage, and respect and are described in more detail below.

Communication

Software development is inherently a team sport that relies on communication to transfer knowledge from one team member to everyone else on the team. XP stresses the importance of the appropriate kind of communication – face to face discussion with the aid of a white board or other drawing mechanism.

Simplicity

Simplicity means "what is the simplest thing that will work?" The purpose of this is to avoid waste and do only absolutely necessary things such as keep the design of the system as simple as possible so that it is easier to maintain, support, and revise. Simplicity also means address only the requirements that you know about; don't try to predict the future.

Feedback

Through constant feedback about their previous efforts, teams can identify areas for improvement and revise their practices. Feedback also supports simple design. Your team builds something, gathers feedback on your design and implementation, and then adjust your product going forward.

Courage

Kent Beck defined courage as "effective action in the face of fear" (Extreme Programming Explained P. 20). This definition shows a preference for action based on other principles so that the results aren't harmful to the team. You need courage to raise organizational issues that reduce your team's effectiveness. You need courage to stop doing something that doesn't work and try something else. You need courage to accept and act on feedback, even when it's difficult to accept.

Respect

The members of your team need to respect each other in order to communicate with each other, provide and accept feedback that honours your relationship, and to work together to identify simple designs and solutions.

The XP Practices have changed a bit since they were initially introduced. The original twelve practices are listed below.

- The Planning Game
- Small Releases
- Metaphor
- Simple Design
- Testing
- Refactoring
- Pair Programming
- Collective Ownership
- Continuous Integration
- 40-hour week
- On-site Customer
- Coding Standard

QUIZ -1

1. A team is executing a project for a customer using agile software development approach. for every activity, they take a sign – off from the customer as per the clause present in the contract. the requirements are also being developed as per what was agreed upon in the contract during start of the project with minimal interactions with the customer. as per the manifesto for agile software development, which value is being violated here?

ANS: - * Customer collaboration over contract negotiation

2. Which among these represents the primary measure of progress?

ANS: - * registration feature which was released to customer's demo

3. As per agile manifesto for software development, which is values more than comprehensive documentation?

ANS: - * working software

Self-assignment: -

1. What are some advantages of extreme programming?

ANS: - * Focuses on key activities * Values communication and work transparency

2. What are some good use cases for extreme programming?

ANS: - * when releases can be iterative

*** When resilience during change is needed**

3. What are some of the phases of iteration planning?

ANS: - * steering

*** Exploration**

4. what are some benefits of pair programming?

ANS: - *** It reduces coordination efforts**

*** It reduces risk**

5. which pair programming strategy involves one developer creating a test and the other developer creating code to satisfy the test

ANS: - *ping-pong pairing

6. what are some benefits of test-driven development?

ANS: -* Early bug notification

7. what is the second phase of the test-driven development cycle?

ANS: - * confirm test fails

8.which statement best describes the importance of the customer role in xp?

ANS: - * only the customer knows what needs to be done and why

9.which statement best describes the difference between source control and version control?

ANS: -*source control specification manage code, while version control includes other types of files like binaries

10. which operations is used to merge code from one branch to another?

ANS: - * pull

11. which statements best describes the different between continuous integration (CI) and continuous deployment (CD)?

ANS: - * CI manages compilation, testing and packaging while CD manages the distribution

12. which operations are used to implement continuous integration in GitHub?

ANS: - * Actions

13. which category of coding standards can be left in via lotion if there are agreed up on to do so?

ANS: - * Recommendation

14.what are some potential negative to collective code owner ship?

ANS: - * Decreased motivations

***Reliance on team expertise**

15. what are some benefits of code refactoring?

ANS: - * **Increases extensibility**

*** Increased maintainability**

16. what are some effective refactoring strategies?

ANS: - * **Reduce method length**

*** Reduce duplication**

17. what is the maximum amount of time recommended between small releases in agile software development?

ANS: - * Two weeks

18. what are some benefits of system metaphors?

ANS: - * Allow everyone to share the same vocabulary * give everyone a shared understanding of the vision

19. what should be the first step when implementing a 40-hour work week?

ANS: - * Experiment

EXERCISE: -03

DevOps adoption in project

Progressive Collaboration

DevOps promises to bridge the gap between the two where both employ bottom-up and top-down feedback from each other. With DevOps, when development seeks operational help or when operations require immediate development, both remain ready for each other at any given time. In such a scenario, the software development culture brings in to focus combined development instead of individual goals; The development environment becomes more progressive as all the team members work in cohesion towards a common goal.

Processing Acceleration

With conjoined operational and developmental paradigms, the communication lag betweenhood is reduced to null. Organizations continuously strive for a better edge over their competing rivals, and if such acceleration is not achieved, the organization will have to succumb to competing forces— innovation will be slower, and the product market will decay.

Shorter Recovery Time

DevOps deployment functions on a more focused and exclusive approach which makes issues more accessible to spot; this helps error rectification faster and easier to implement. The resolution to problems is inherently quicker, as troubleshooting happens to take place at the current development level only, within a single team. Thus, the overall time for recovery and rectification is drastically reduced.

Lower Failure Rate

The abridged departments yield shorter development cycles which result in rapid production. The entire process becomes modular wherein issues related to configuration, application code, and infrastructure become more apparent and pre-accessible A decrease in error count also positively affects the success rates of development. Therefore, very few fixes will be required to attain a fully functional code for the desired output.

Higher Job Satisfaction

DevOps fosters equality by bringing different officials at the same level of interaction. DevOps serves as a handy tool for achieving that feat; it enables the workforce to work in consistency where chances for failure are minimal, and production is rapid. As a result, the processing becomes efficient and workspace more promising

The DevOps adoption requires focus on the People, Process and Technology.

QUIZ- 1 Definition of DevOps

1. The IT term management at "pura vida" company has decided to adopt DevOps and has drawn a roadmap for the journey. This information is communication to all the term members (developers, testers, architects, operations team (include infrastructure, system administration and deployment). Here is the first reaction of the team

ANS: - * managers express that the buy-in should be there from the business (customers) and the top management in: pura vida".

*** IT management team member feel that additional roles will be required (other than scrum master, product owner and Dev team) to execute this**

2. One of the project team of a portfolio for a customer account mention the following which of the statement is TURE?

ANS: - *the tools and automation may cost my project more, but we get good quality and speed as a benefits

QUIZ -2 :- Aligning Capabilities

1.choose the aspects considered by Infosys for Devops adoption

a) People

b) Process

c)Technology

ANS: - * a, b and c

2. choose the business drivers for adoption of DevOps (multiple response question)

ANS: - * Early time to market

*** Quick deployment with good quality**

3. Match the scenarios with the feature/capability that can be applied.

Capability:

a. Feature toggle
b. Microservices
c. Big room planning
d. Service virtualization
e. Infrastructure as code

Scenarios

1.A bank is introducing the online fixed deposit scheme. if this feature has to be deployed in production, the account service module which provides the customer account details online would need to be used and also updated. The updating would disrupt the account service module. This cannot be afforded by the bank. However, the new feature needs to be tested

2. An online audio and video steaming company receive a million calla every day from different types of devices for different services. They need an architectural style which consists of lightweight components

3. A support team receives a ticket from the customer that a specific server is not reachable. The support staff try out quick fixes, but is not working and the server crashes. It needs to now be reconfigured. The support staff face this situation very often and are wasting a lot of time doing reconfiguration manually all the time

4. Team A has completed working on a feature. However, they are waiting for same related features from B and C so that deployment can be done together. Customer is keen on having feature A urgently

5. A software services company brings all its stakeholders right from developers to support teams together for effective execution of projects

ANS: - * a4, b2, c5, d1, e3

4. Match the stakeholder and what capabilities they need to build while embarking on the Dev Ops Implementation journey.

   a) Business
   b) Dev Team
   c) Testing Team
   d) Infra team
   e) Ops team
   f) Organization
   1. Continuous integration
   2. Automated environment management
   3. Progressive test automation
   4. Policies to support merging of Dev and Ops teams
   5. Big room planning

ANS: - * a5, b1, c3, d2, e4

QIS-3 :- CI-CD

1.The customer insists a Dev team to use Jenkins and construct an automated continuous integration pipeline. The team accepts this request and constructs a Cl pipeline orchestrated by Jenkins. They schedule daily integration. After a month of implementation, the customer finds that the bugs that are released to production are increasing. When they inspect the pipeline stages, they find the following stages

Version control-> build automation -> baseline in artifact repository

What is the team missing here?

ANS: - * The automated pipeline should have in-built quality with static code analysis included with a good number of quality rules and gating conditions for quality

*** Unit tests should be automated and included so that they can be repeatedly invoked**

2.A development team which is implementing C using an orchestration tool are doing the following activities Choose the ones which may not be good practices.

ANS: - * If the QA tests fail, the developers make the changes in the server where the QA tests run, compile and run the tests again

*** The team auto-trigger the pipeline whenever a team member completes the work and push code to the central version control repository**

*** If the Cl pipeline is broken, the teams continue with the features they planned during that day instead of fixing the pipeline as it might take a long time to do it**

3.Choose the statement(s) that are TRUE with respect to choosing tool stack for automating the CICD pipeline.

ANS: - * Based on appetite of customer for automation of tasks

*** Based on budget availability**

*** Based on domain and project requirements**

*** After consultation with a tool expert/coach**

4.Choose the statement(s) that are TRUE with respect to choosing tool stack for automating the CICD pipeline

Ans: - * Open-source tools need to be OSS compliant

*** Tooling and infrastructure budget for automation should be upfront communicated** to **customers**

FINAL ASSIGNMENT: -

Continuous Integration and Delivery - DevOps Assessment

1. Daily buy is an e-commerce website focusing on selling domestic items and delivering them to the doorsteps of their customers. They were struggling with very slow site updates and the site was down multiple times in a week due to maintenance and feature releases. This was affecting their sales tremendously and they were slowly losing to their competitors. Which of the following practice(s) can help this team? [multiple response]

ANS: - * Continuous monitoring

2. A team comprising of ten members was working on a retail project in Waterfall mode and the first version of the software has been released. In view of the market dynamics and competition, the client wants the team to switch to Agile methodology for its next release due in 8 months. What changes should the team bring in their working to adopt Agile way of software development?
   A) Work towards building a cross-functional team
   B) Plan to adopt technical practices/XP practices to sustain the pace
   C) Drop all documentation from its processes
   D) Identify means for collaboration and communication between teams

ANS: - * A, B and D

3. Match the scenario with the feature/capability that can be applied to help/alleviate the problem. The development team has completed working on a page offering festival discounts to customers. However, they are waiting for some related features from third party vendors so that deployment can be done together. Customer is keen on having the discount feature urgently as there is bound to be tight competition for sales among contemporaries.

ANS: - * Feature toggle

4. Match the scenario with the feature/capability that can be applied to help/alleviate the problem. An infrastructure team that maintains a data centre with many servers receives a ticket frequently from an internal team to reconfigure it. The frequency is as much as thrice in a week. Presently, the staff repeats the configuration manually. The steps are always the same.

ANS: - Infrastructure as code

5. Match the scenario with the feature/capability that can be applied to help/alleviate the problem. An online services company is ever expanding its services and new services get added often. Some of them get terminated if the service becomes obsolete. The company receives a million calls every day from different types of customers for different services. The developers need an architectural style which consists of lightweight components which can be easily plugged in and out.

ANS: - * Microservices

6. From the options given, choose the stage(s) which are likely to be automated as part of continuous integration [multiple response]

ANS: - 1. Continuous integration invocation

2.Code coverage

3. Static analysis of code

4.Source code version control

7. What are the benefits of Continuous Integration? [Multiple response question].

ANS: - * Provides ability to completely rebuild and test applications

*** Automated Build integrates various tools**

8. ____metric helps track production release agility over a period of time.

ANS: -* Total number of releases per week

9. Benefits of infrastructure automation include:

ANS: - 1. Consistent, production-like environments across deployment pipeline

2.Ease of creating and managing changes to environments or related configuration

3.Maintaining the quality of code

4. Small sized binaries

10. Choose the option which represents the need for DevOps adoption

1. To increase frequency of deployment
2. To ensure quality software is deployed
3. To ensure service levels are maintained for operations, maintenance and production Support
4. To combine Dev and Ops teams together

ANS: - Options a, b, c only

11. A Dev team working on an e-commerce project has adopted continuous integration, delivery and deployment. However, they are not using an artifact repository in the pipeline. What could possibly go wrong?

ANS: - Inconsistency of versions in build, test and release stages

12. Which among the following is an orchestration tool?

ANS: -Jenkins

13. A static code analysis tool helps ensure &blank.

ANS: - Code quality

14. Which of the following statements represent a salient feature of a pure DevOps team structure? [Multiple response question].

ANS: - Teams keep separate backlogs but take each other's user stories in their backlogs

15. Which of the following is a definition of code coverage?

ANS: - Measure used to describe the degree to which the source code of a program has been tested.

16. Which of the options represents the aspects to be taken care while choice of tools is being made?
    A. Repeatability
    B. Reliability
    C. End-end automation
    D. Auto build quality in the pipeline
    E. Customer's appetite for tooling

ANS: - All the given options

17. Your project has a 2-week sprint cadence and by the end of the sprint the feature (or developed code) should be deployed into QA environment. You notice that the version going for release is not the same as the one being tested. What best practices would you recommend to avoid this?

ANS: - * Ensure using an artifact repository and any change made to the code or tests will go through the entire CICD pipeline

18. Which of the following can be used for releasing feature and enabling and disabling them without changing the code?

ANS: - * Feature toggle

19. Which of these represent the advantages of adopting DevOps practices using an end-to-end automated pipeline?
    A. Faster & Frequent Deployments
    B. Improved quality, environment stability and application availability
    C. Quick time to market

ANS: - All the given options

20. Choose the statement which represents the benefit of adoption of DevOps practices in projects following Agile approach to software development.

ANS: - * Helps in quick development and delivery to customers with good quality

21. Duplicate lines of codes metric measures &blank___.

ANS: - * Quality of code

22. Tracking coding rule violations, code complexity and duplications in code help &blank__.

ANS: -* Developers write good quality code

23. Your production support team wants to pro-actively know when an issue is likely to occur. They can then take corrective action. Which among the capabilities mentioned would you implement?

ANS: - * Logging and continuous monitoring

24. Artifact repository ensures &blank__________.

ANS: - * The right version of the build is used for QA and the completely tested version goes for release.

25. When should a security testing be done?

ANS: - * Before code is compiled and before deploying to QA environment

Exercise 4:

Module name: Implementation of CICD with Java and open-source stack
Configure the web application and Version control using Git using Git commands and version control operations.

What is GIT?

Git is a free open-source distributed version control system you can use to track changes in your files. You can work on all types of projects in Git, from small to large.

With Git, you can add changes to your code and then commit them (or save them) when you're ready. This means you can also go back to changes you made before.

Git works hand in hand with GitHub – so what is GitHub?

What is GitHub?

GitHub is a web interface where you store your Git repositories and track and manage your changes effectively. It gives access to the code to various developers working on the same project. You can make your own changes to a project at the same time as other developers are making theirs.

If you accidentally mess up some code in your project while making changes, you can easily go back to the previous stage where the mess has not occurred yet.

Why use GitHub

There are so many reasons you should learn and use GitHub. Let's look at a few of them now.

Effective Project Management

GitHub is a place where your Git repositories are stored. GitHub makes it easy for developers working on the same project but in different locations to be on the same page.

With GitHub, you can easily track and manage the changes you have made and check on the progress you've made in your project.

Easy Collaboration and Cooperation

With GitHub, developers from all over the world can work together on a project without having any problems.

Teams are able to stay on the same page while working on a project together and can easily organize and manage the project effectively.

Open Source

GitHub is a free and open-source system. This means that developers can easily access different types of code/projects which they can use in learning and developing their skills.

Versatility

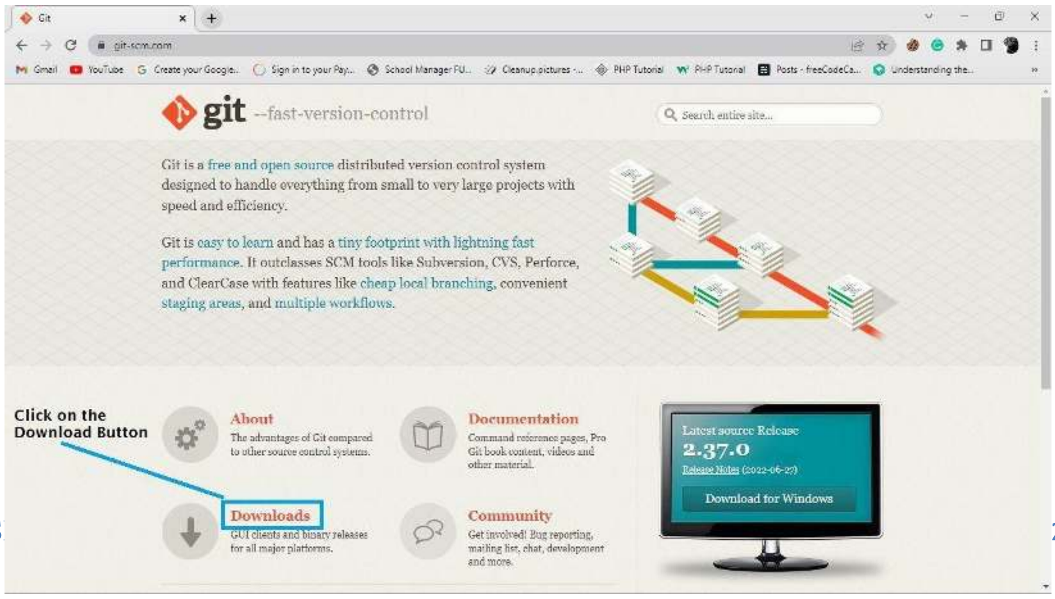
This attribute of GitHub is very important. GitHub is not a web interface for only developers. It can be used by designers, writers, and anyone who wants to keep track of the history of their projects.

How to Setup Git

To start using Git, you'll need to download it to your computer if you haven't already. You can do this by going to their official website.

When Git opens, scroll down a bit and you should see a download button. Go ahead and click on it.Download button on the Git website

Choose your operating system whether it's Windows, MacOS, Linux/Unix. In my case, I will be choosing the Windows option because I am using a Windows computer:



Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

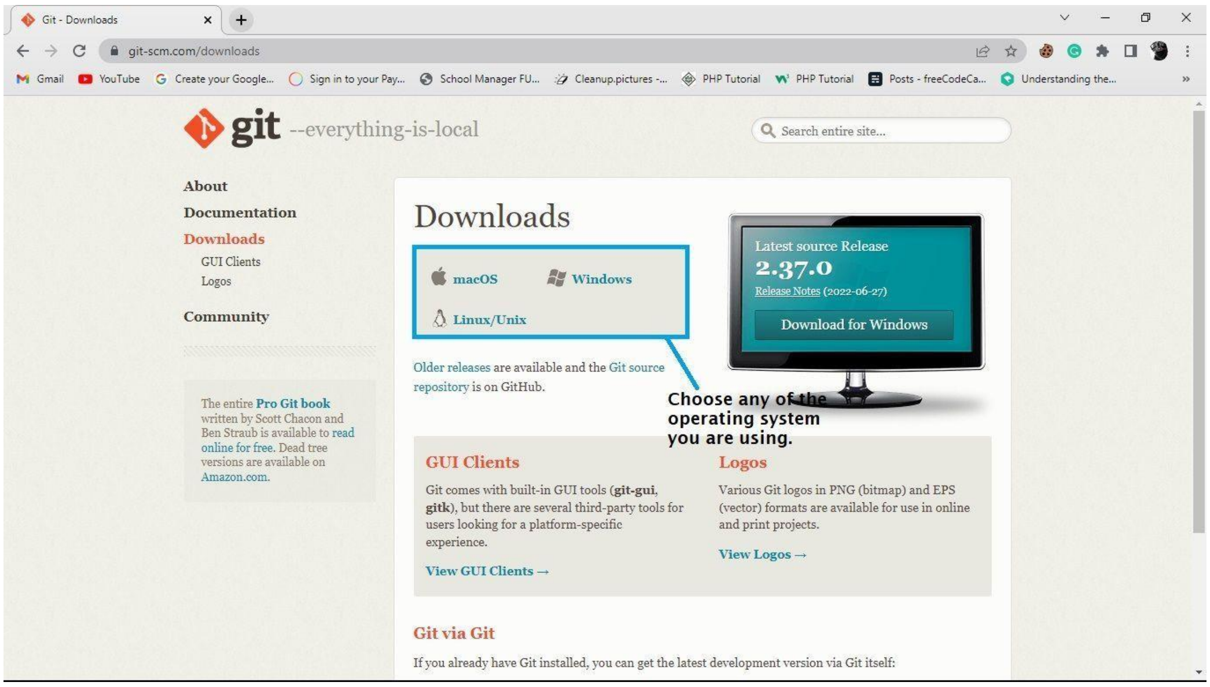
Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows.

About: The advantages of Git compared to other source control systems.

Documentation: Command reference pages, Pro Git book content, videos and other material.

Downloads: GUI clients and binary releases for all major platforms.

Community: Get involved! Bug reporting, mailing list, chat, development and more.



Downloads

Older releases are available and the Git source repository is on GitHub.

The entire Pro Git book written by Scott Chacon and Ben Straub is available to read online for free. Dead tree versions are available on Amazon.com.

GUI Clients

Git comes with built-in GUI tools (git-gui, gitk), but there are several third-party tools for users looking for a platform-specific experience.

View GUI Clients →

Logos

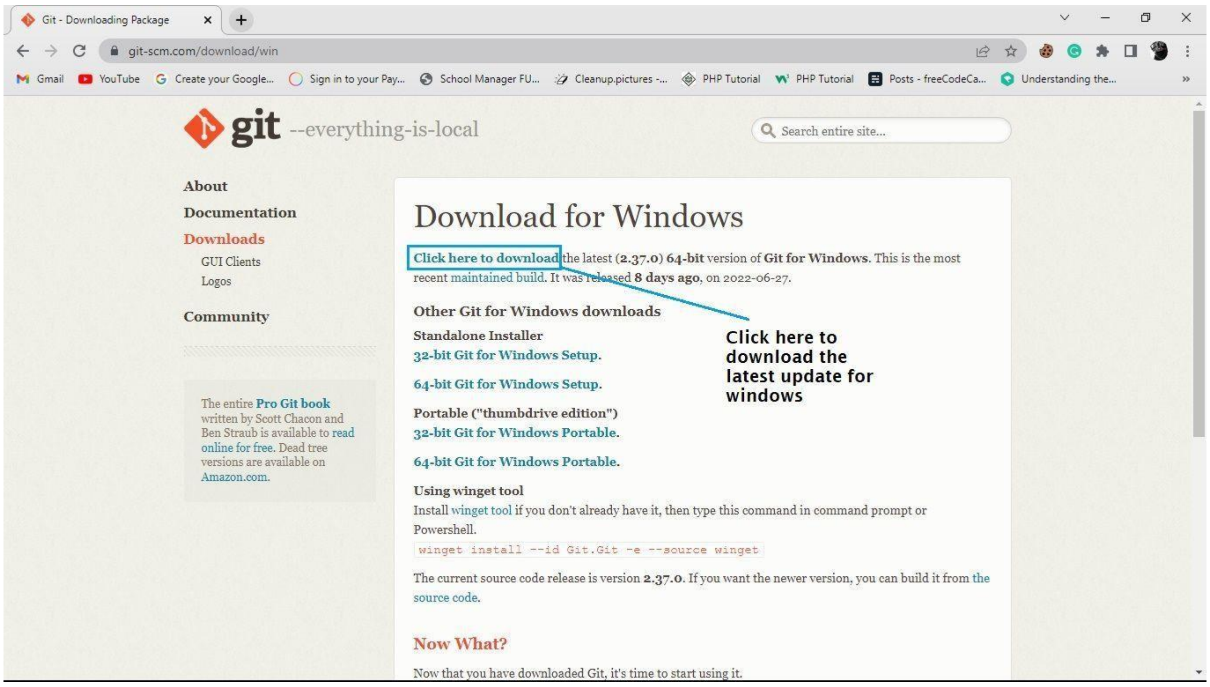
Various Git logos in PNG (bitmap) and EPS (vector) formats are available for use in online and print projects.

View Logos →

Git via Git

If you already have Git installed, you can get the latest development version via Git itself:

Choose your operating see you system
Click on the first link at the very top of the page to download the latest version of Git.



Download for Windows

Click here to download the latest (2.37.0) 64-bit version of Git for Windows. This is the most recent maintained build. It was released 8 days ago, on 2022-06-27.

The entire Pro Git book written by Scott Chacon and Ben Straub is available to read online for free. Dead tree versions are available on Amazon.com.

Other Git for Windows downloads

Standalone Installer
32-bit Git for Windows Setup.

64-bit Git for Windows Setup.

Portable ("thumbdrive edition")
32-bit Git for Windows Portable.

64-bit Git for Windows Portable.

Using winget tool
Install winget tool if you don't already have it, then type this command in command prompt or Powershell.

```
winget install --id Git.Git -e --source winget
```

The current source code release is version 2.37.0. If you want the newer version, you can build it from the source code.

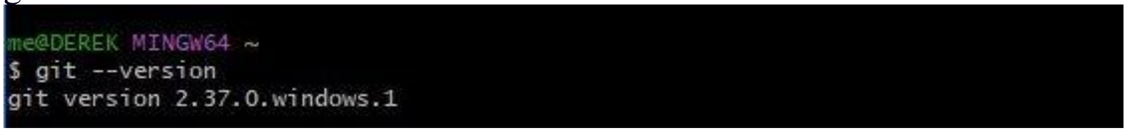
Now What?

Now that you have downloaded Git, it's time to start using it.

Download the latest version of Git by clicking the first link
When the download is complete, then go ahead and install Git to your computer. You'll need to go to the location where the file has been downloaded and install it.

After the installation, you'll want to make sure that Git is successfully installed on your system. Open your command prompt or Git bash (whichever one you choose to use) and run the command:

git --version



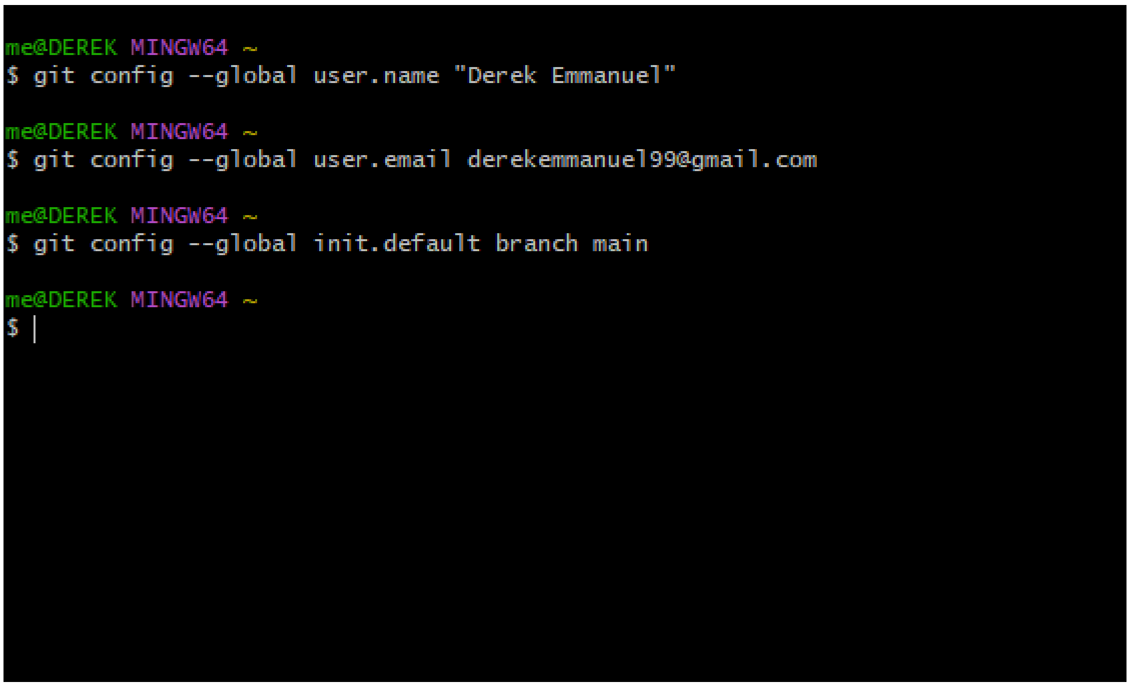
```
me@DEREK MINGW64 ~
$ git --version
git version 2.37.0.windows.1
```

If Git was successfully installed on your computer, it should display the current version of Git below the command you just ran. If the current version is being displayed, congratulations!

How to Configure Git

Now that we have installed Git on our computer, we have to configure it. We do this so that any time we are working in a team on a project, we can easily identify the commits we have made in the repository.

To configure Git, we need to specify the name, email address, and branch by using the git config --global command. For example:



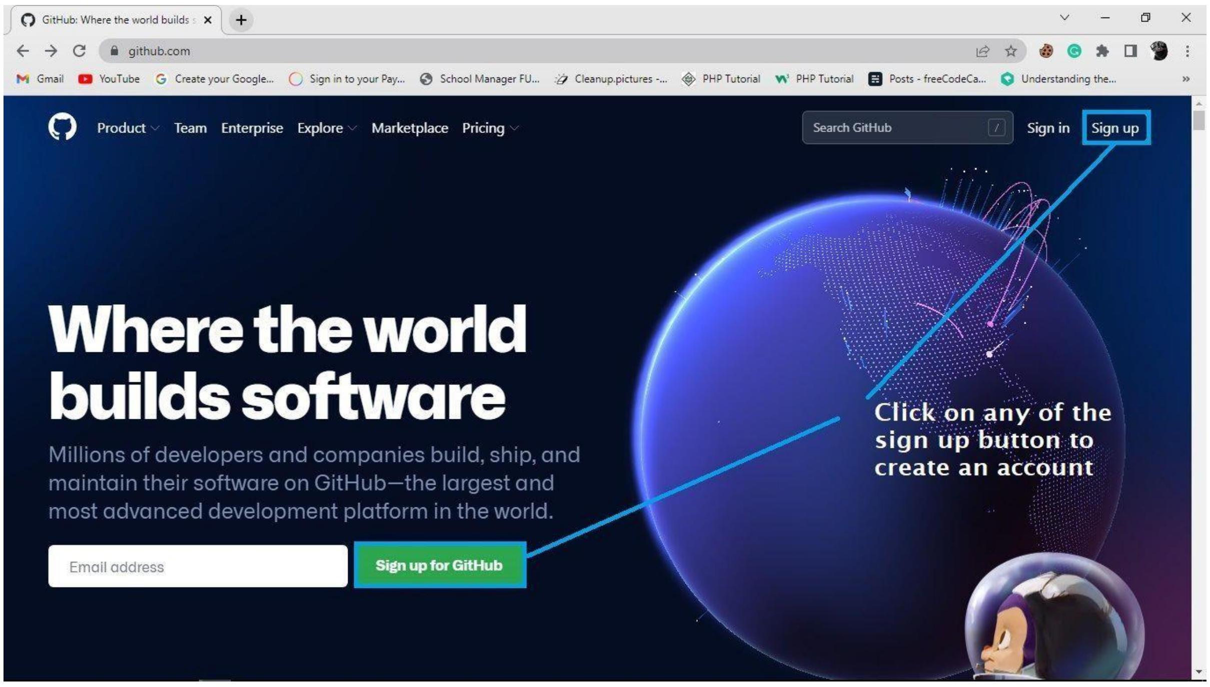
```
me@DEREK MINGW64 ~
$ git config --global user.name "Derek Emmanuel"

me@DEREK MINGW64 ~
$ git config --global user.email derekemmanuel99@gmail.com

me@DEREK MINGW64 ~
$ git config --global init.default branch main

me@DEREK MINGW64 ~
$ |
```

From the image above, we used git config --global user.name to configure the username. In my case I used my name “Derek Emmanuel”. The same applies for the git config --global user.email.

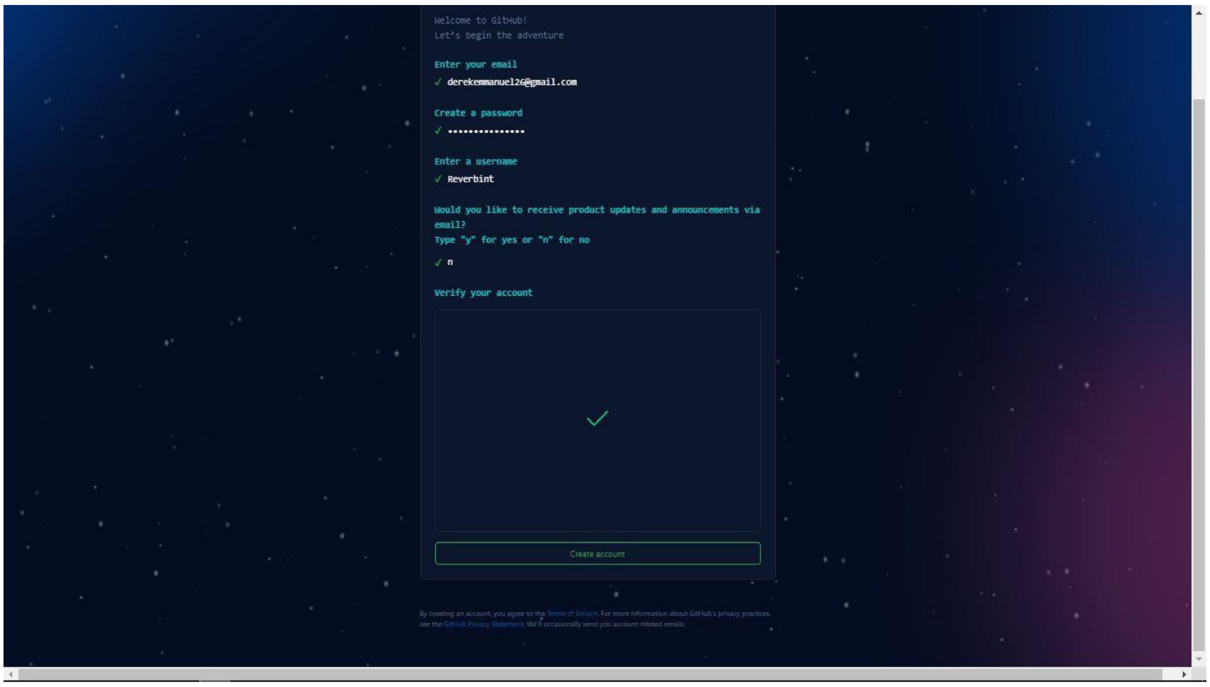
Git comes with a default branch of master, so I changed it to be called the main branch by using the git config --global init.default branch main command.
Now you're ready to start using Git.

How to Setup a GitHub Account

To set up a GitHub account, visit their official website. Click on the sign up button in the upper right corner:


Product
Team
Enterprise
Explore
Marketplace
Pricing
Search GitHub
Sign in
Sign up
Where the world builds software
Millions of developers and companies build, ship, and maintain their software on GitHub—the largest and most advanced development platform in the world.
Email address
Sign up for GitHub
Click on any of the sign up button to create an account

When the sign up form opens up, enter your email, create a password, enter your username, and then verify your account before clicking on the create account button.



```
Welcome to GitHub!
Let's begin the adventure

Enter your email
✓ derekemmanuel26@gmail.com

Create a password
✓ •••••••••••••••

Enter a username
✓ Reverbint

Would you like to receive product updates and announcements via
email?
Type "y" for yes or "n" for no
✓ n

Verify your account
```

Create your GitHub account

Commonly Used Git Commands

There are some basic Git commands that every developer should know how to use:

- git config
- git init
- git add
- git commit
- git clone
- git push
- git rm
- git branch

Let's go through each of these briefly so you know how to use them.

How to Use the git config Command

You use this command to set the username, email, and branch of a user so as to identify who made a commit when working on a project. This command is used when you have downloaded git into your computer and you want to customize it for your use.

For example:

```
git config --global user.name “ [username]”
git config --global user. Email [email address]
```

How to Use the git init Command

You use the git init command to start Git in your project. This git command is used when you are working on a project and would like to initialize git to the project in order to keep track of the changes made in the project.

For example: it init

When you run this command, you should see a folder named. git being created automatically in the current folder you are working on.

How to Use the git add Command

This command adds your file to the staging area. The staging area is the area where files we make changes to are added and where they wait for the next commit.

To add a file to the staging area, you use the git add command. It adds all the files in the folder to the staging area.

git add (file name) adds the name of the particular file you want to commit in the staging area.

Use this command when you have made changes to a file and want to commit them to your project.

How to Use the git commit **Command**
This commits any file you added with the git add command as well as every file in the staging area.

For example:

git commit –m “first commit”
This command saves a file permanently to the Git repository. You use it whenever a file has been added to the staging area using the git add command.

How to Use the git clone **Command**

You use the git clone command to copy an existing repository in another location to the current location where you want it to be.

For example:

git clone (repository name)
You use this command when you want to duplicate a Git repository from GitHub into your local storage.

How to Use the git push **Command**
You use this command to upload/push files from the local repository/storage to another storage, like a remote storage such as GitHub.

For example:

git push (remote storage name)
You only use this command when you're satisfied with the changes and commits you've made on a project and finally want to upload/push it to the Git repository in GitHub.

How to Use the git rm **Command**
You use this Git command to remove a file from a working repository. For example:

git rm (filename)

You use this command only when you wish to get rid of an unwanted changes/file from the Git repository.

How to Use the git branch Command

You use this command to check the current branch you are working on, either main or master.

For example:
git branch
This command helps you know the current branch you are working on.

Conclusion

In this tutorial you learned what version control systems are all about. You also learned how to install and setup Git on your computer and setup a GitHub account. Lastly, we went through some commonly used Git commands.

Exercise 5:

Module Name: Implementation of CICD with Java and open-source stack
Configure a static code analyser which will perform static analysis of the web application code and identify the coding practices that are not appropriate. Configure the profiles and dashboard of the static code analysis tool.

What is static analysis?

Static analysis is a method of analysing code for defects, bugs, or security issues prior to pushing to production. Often referred to as "linters," static analysis tools remove the unnecessary fluff from your code and perform some automated checks to improve code quality. Static analysis tools can check for:

- Inconsistencies in code style conventions and standards. It can be as simple as enforcing consistent indentation and variable names or as complex as enforcing compliance with the MISRA or CERT Secure Coding Standards
- Resource leaks such as a failure to release allocated memory, which can eventually lead to program crashes or failure to close files
- Incorrect usage of Application Programming Interfaces (APIs)
- Common security vulnerabilities such as those identified by the Open Web Application Security Project (OWASP) or Common Weakness Enumeration (CWE)

What kinds of static analysis tools are available?

The static analysis tools available can be categorized by the capabilities they support, including:

Programming languages: Tools may support single or multiple languages. If your codebase spans multiple languages, a single tool like Coverity which supports 14 languages including JavaScript, .NET, Java, and Python may be the most thorough option for discovering bugs across languages.

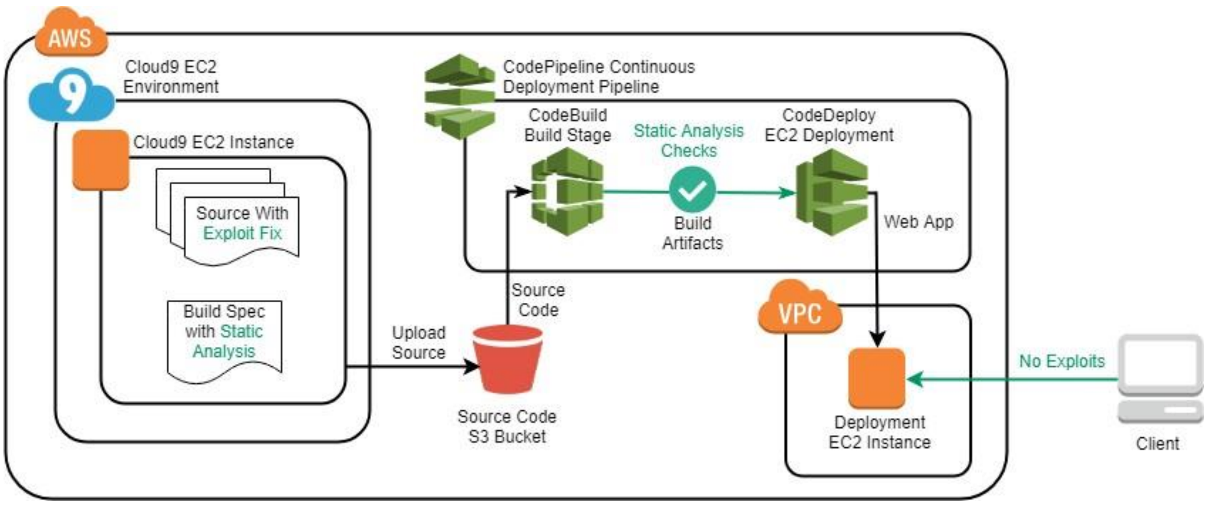
Real-time tools: Instantaneous analysis tools are ideal for checking code in development environments as it's being written. Here, the trade-off is speed over more thorough, time-consuming checks. Many of these are open source, which allows for easier adoption and customization.

Deep analysis tools: On the other end of the spectrum, deep analysis tools can take much longer and are likely to identify issues that a real-time tool would miss. Enterprise-grade tools in this area often have hefty licensing fees and they may bring more issues to light than you have the bandwidth to address. Many of these tools may be configured to report only the most important issues,

Compilers: Although not a dedicated static analysis tool, compilers may also be used to improve the quality of your code. You can use configuration flags to adjust the number of checks they perform.

Integrating Static Analysis within CI/CD

- Among the many benefits of using static analysis tools, the one that is most beneficial to organizations is the ability to discover bugs before they are released into the wild (and when they are less costly to fix). Within the DevOps practice of CI/CD, static analysis tools provide additional benefits.
- Tools that take a long time to run tend to be ignored during development. Even if static analysis isn't always a long process, it's still not the best use of a developer's time.
- Integrating analysis tools within CI/CD ensures that they are used consistently and automatically while offering an extra level of analysis to make sure that nothing is able to sneak through.
- There are different options for how to integrate static analysis tools in your environment. One approach is to run it early in the pipeline along with other automated tests.
- At this point, you'll be able to fix any issues before the peer code review and it speeds up the overall process. In turn, developers spend less time reviewing and have more time to develop new code.
- If you have large code bases, running a deep analysis on every commit may take too much time. Instead, you can use a less thorough analysis configuration on development branches and perform more expensive scans on a schedule or when integrating into upstream branches.
- The goal is to discover bugs as early as practically possible and it's up to you to choose the system that works best for your team.
- Tools like Klocwork have fully embraced CI/CD workflows and can incrementally analyse the code changes on each commit.
- Higher-end static analysis tools can also track bugs over time. This can help you select which issues to work on in the current release cycle as source code is continuously being integrated.
- Issues reported in longstanding legacy code that haven't caused problems are probably not worth the time investment to resolve them in the immediate term. Instead, use precious developer time to focus on more recent issues.
- Another practical constraint is the budget available for static analysis. Rather than obtaining a license for each developer, run the analysis tools on a set number of build machines (or a single machine if possible).


AWS
Cloud9 EC2 Environment
Cloud9 EC2 Instance
Source With Exploit Fix
Build Spec with Static Analysis
Upload Source
Source Code S3 Bucket
Source Code
CodePipeline Continuous Deployment Pipeline
CodeBuild Build Stage
Static Analysis Checks
Build Artifacts
CodeDeploy EC2 Deployment
Web App
VPC
Deployment EC2 Instance
No Exploits
Client

Exercise 6:

Module Name: Implementation of CICD with Java and open-source stack
Write a build script to build the application using a build automation tool like Maven. Create a folder structure that will run the build script and invoke the various software development build stages. This script should invoke the static analysis tool and unit test cases and deploy the application to a web application server like Tomcat.

Build a Java app with Maven

Table of Contents how to use Jenkins to orchestrate building a simple Java application with Maven.

If you are a Java developer who uses Maven and who is new to CI/CD concepts, or you might be familiar with these concepts but don't know how to implement building your application using Jenkins, then this tutorial is

Prerequisites

For this tutorial, you will require:

- A macOS, Linux or Windows machine with:
  - 256 MB of RAM, although more than 2 GB is recommended.
  - 10 GB of drive space for Jenkins and your Docker images and containers.
- The following software installed:
  - Docker - Read more about installing Docker in the Installing Docker section of the Installing Jenkins page.

    Note: If you use Linux, this tutorial assumes that you are not running Docker commands as the root user, but instead with a single user account that also has access to the other tools used throughout this tutorial.

  - Git and optionally GitHub Desktop.

Run Jenkins in Docker

In this tutorial, you'll be running Jenkins as a Docker container from the jenkins/jenkins Docker image.

To run Jenkins in Docker, follow the relevant instructions below for either macOS and Linux or Windows.

You can read more about Docker container and image concepts in the Docker section of the Installing Jenkins page.

On macOS and Linux

1. Open up a terminal window.
2. Create a bridge network in Docker using the following docker network create command:

```
docker network create jenkins
```

3. In order to execute Docker commands inside Jenkins nodes, download and run the docker:dind Docker image using the following docker run command:

```
4.  docker run \
5.    --name jenkins-docker \
6.    --rm \
7.    --detach \
8.    --privileged \
9.    --network jenkins \
10.   --network-alias docker \
11.   --env DOCKER_TLS_CERTDIR=/certs \
12.   --volume jenkins-docker-certs:/certs/client \
13.   --volume jenkins-data:/var/jenkins_home \
14. --publish 2376:2376 \
15.   --publish 3000:3000 --publish 5000:5000 \
16.   docker:dind \
      --storage-driver overlay2
```

	(*Optional*) Specifies the Docker container name to use for running the image. By default, Docker will generate a unique name for the container.
	(*Optional*) Automatically removes the Docker container (the instance of the Docker image) when it is shut down.
	(*Optional*) Runs the Docker container in the background. This instance can be stopped later by running docker stop jenkins-docker.
	Running Docker in Docker currently requires privileged access to function properly. This requirement may be relaxed with newer Linux kernel versions.
	This corresponds with the network created in the earlier step.
	Makes the Docker in Docker container available as the hostname docker within the jenkins network.
	Enables the use of TLS in the Docker server. Due to the use of a privileged container, this is recommended, though it requires the use of the shared

	volume described below. This environment variable controls the root directory where Docker TLS certificates are managed.
	Maps the /certs/client directory inside the container to a Docker volume named jenkins-docker-certs as created above.
	Maps the /var/jenkins_home directory inside the container to the Docker volume named jenkins-data. This will allow for other Docker containers controlled by this Docker container's Docker daemon to mount data from Jenkins.
	(*Optional*) Exposes the Docker daemon port on the host machine. This is useful for executing docker commands on the host machine to control this inner Docker daemon.
	Exposes ports 3000 and 5000 from the docker in docker container, used by some of the tutorials.
	The docker:dind image itself. This image can be downloaded before running by using the command: docker image pull docker:dind.
	The storage driver for the Docker volume. See "Docker storage drivers" for supported options.

Note: If copying and pasting the command snippet above does not work, try copying and pasting this annotation-free version here:

```
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 3000:3000 --publish 5000:5000 --publish 2376:2376 \
  docker:dind --storage-driver overlay2
```

17. Customise official Jenkins Docker image, by executing below two steps:
    a. Create Dockerfile with the following content:
    b. FROM jenkins/jenkins:2.361.4-jdk11
    c. USER root
    d. RUN apt-get update && apt-get install -y lsb-release
    e. RUN curl -fsSLo /usr/share/keyrings/docker-archive-keyring.asc \
    f.   https://download.docker.com/linux/debian/gpg

```
g. RUN echo "deb [arch=$(dpkg --print-architecture) \
h.   signed-by=/usr/share/keyrings/docker-archive-keyring.asc] \
i.   https://download.docker.com/linux/debian \
j.   $(lsb_release -cs) stable" > /etc/apt/sources.list.d/docker.list
k. RUN apt-get update && apt-get install -y docker-ce-cli
l. USER jenkins
   RUN jenkins-plugin-cli --plugins "blueocean:1.25.8 docker-
   workflow:521.v1a_a_dd2073b_2e"
```

m. Build a new docker image from this Dockerfile and assign the image a meaningful name, e.g. "myjenkins-blueocean:2.361.4-1":

```
docker build -t myjenkins-blueocean:2.361.4-1 .
```

Keep in mind that the process described above will automatically download the official Jenkins Docker image if this hasn't been done before.

18. Run your own myjenkins-blueocean:2.361.4-1 image as a container in Docker using the following docker run command:

```
19. docker run \
20.   --name jenkins-blueocean \
21.   --detach \
22.   --network jenkins \
23.   --env DOCKER_HOST=tcp://docker:2376 \
24.   --env DOCKER_CERT_PATH=/certs/client \
25.   --env DOCKER_TLS_VERIFY=1 \
26. --publish 8080:8080 \
27. --publish 50000:50000 \
28.   --volume jenkins-data:/var/jenkins_home \
29.   --volume jenkins-docker-certs:/certs/client:ro \
30.   --volume "$HOME":/home \
31.   --restart=on-failure \
32.--env JAVA_OPTS="-
    Dhudson.plugins.git.GitSCM.ALLOW_LOCAL_CHECKOUT=true" \
    myjenkins-blueocean:2.361.4-1
```

	(*Optional*) Specifies the Docker container name for this instance of the Docker image.
	(*Optional*) Runs the current container in the background (i.e. "detached" mode) and outputs the container ID. If you do not specify this option, then the running Docker log for this container is output in the terminal window.

	Connects this container to the jenkins network defined in the earlier step. This makes the Docker daemon from the previous step available to this Jenkins container through the hostname docker.
	Specifies the environment variables used by docker, docker-compose, and other Docker tools to connect to the Docker daemon from the previous step.
	Maps (i.e. "publishes") port 8080 of the current container to port 8080 on the host machine. The first number represents the port on the host while the last represents the container's port. Therefore, if you specified -p 49000:8080 for this option, you would be accessing Jenkins on your host machine through port 49000.
	(*Optional*) Maps port 50000 of the current container to port 50000 on the host machine. This is only necessary if you have set up one or more inbound Jenkins agents on other machines, which in turn interact with your jenkins-blueocean container (the Jenkins "controller"). Inbound Jenkins agents communicate with the Jenkins controller through TCP port 50000 by default. You can change this port number on your Jenkins controller through the Configure Global Security page. If you were to change the **TCP port for inbound Jenkins agents** of your Jenkins controller to 51000 (for example), then you would need to re-run Jenkins (via this docker run … command) and specify this "publish" option with something like --publish 52000:51000, where the last value matches this changed value on the Jenkins controller and the first value is the port number on the machine hosting the Jenkins controller. Inbound Jenkins agents communicate with the Jenkins controller on that port (52000 in this example). Note that WebSocket agents do not need this configuration.
	Maps the /var/jenkins_home directory in the container to the Docker volume with the name jenkins-data. Instead of mapping the /var/jenkins_home directory to a Docker volume, you could also map this directory to one on your machine's local file system. For example, specifying the option --volume $HOME/jenkins:/var/jenkins_home would map the container's /var/jenkins_home directory to the jenkins subdirectory within the $HOME directory on your local machine, which would typically be /Users/<your-username>/jenkins or /home/<your-username>/jenkins.

	Note that if you change the source volume or directory for this, the volume from the docker:dind container above needs to be updated to match this.
	Maps the /certs/client directory to the previously created jenkins-docker-certs volume. This makes the client TLS certificates needed to connect to the Docker daemon available in the path specified by the DOCKER_CERT_PATH environment variable.
	Maps the $HOME directory on the host (i.e. your local) machine (usually the /Users/<your-username> directory) to the /home directory in the container. Used to access local changes to the tutorial repository.
	Configure the Docker container restart policy to restart on failure as described in the blog post.
	Allow local checkout for the tutorial. See SECURITY-2478 for the reasons why this argument should not be used on a production installation.
	The name of the Docker image, which you built in the previous step.

Note: If copying and pasting the command snippet above does not work, try copying and pasting this annotation-free version here:

```
docker run --name jenkins-blueocean --detach \
  --network jenkins --env DOCKER_HOST=tcp://docker:2376 \
  --env DOCKER_CERT_PATH=/certs/client --env
DOCKER_TLS_VERIFY=1 \
  --publish 8080:8080 --publish 50000:50000 \
  --volume jenkins-data:/var/jenkins_home \
  --volume jenkins-docker-certs:/certs/client:ro \
  --volume "$HOME":/home \
  --restart=on-failure \
  --env JAVA_OPTS="-
Dhudson.plugins.git.GitSCM.ALLOW_LOCAL_CHECKOUT=true" \
  myjenkins-blueocean:2.361.4-1
```

33. Proceed to the Post-installation setup wizard.

On Windows

The Jenkins project provides a Linux container image, not a Windows container image. Be sure that your Docker for Windows installation is configured to run Linux Containers rather than Windows Containers. See the Docker documentation for instructions to switch to Linux containers. Once configured to run Linux Containers, the steps are:

1. Open up a command prompt window and similar to the macOS and Linux instructions above do the following:
2. Create a bridge network in Docker
   docker network create jenkins
3. Run a docker:dind Docker image
4. docker run --name jenkins-docker --detach ^
5.   --privileged --network jenkins --network-alias docker ^
6.   --env DOCKER_TLS_CERTDIR=/certs ^
7.   --volume jenkins-docker-certs:/certs/client ^
8.   --volume jenkins-data:/var/jenkins_home ^
9.   --publish 3000:3000 --publish 5000:5000 --publish 2376:2376 ^
     docker:dind
10. Customise official Jenkins Docker image, by executing below two steps:
    a. Create Dockerfile with the following content:
    b. FROM jenkins/jenkins:2.361.4-jdk11
    c. USER root
    d. RUN apt-get update && apt-get install -y lsb-release
    e. RUN curl -fsSLo /usr/share/keyrings/docker-archive-keyring.asc \
    f.   https://download.docker.com/linux/debian/gpg
    g. RUN echo "deb [arch=$(dpkg --print-architecture) \
    h.   signed-by=/usr/share/keyrings/docker-archive-keyring.asc] \
    i.   https://download.docker.com/linux/debian \
    j. $(lsb_release -cs) stable" > /etc/apt/sources.list.d/docker.list
    k. RUN apt-get update && apt-get install -y docker-ce-cli
    l. USER jenkins
       RUN jenkins-plugin-cli --plugins "blueocean:1.25.8 docker-workflow:521.v1a_a_dd2073b_2e"
    m. Build a new docker image from this Dockerfile and assign the image a meaningful name, e.g. "myjenkins-blueocean:2.361.4-1":
       docker build -t myjenkins-blueocean:2.361.4-1 .
       Keep in mind that the process described above will automatically download the official Jenkins Docker image if this hasn't been done before.

11. Run your own myjenkins-blueocean:2.361.4-1 image as a container in Docker using the following docker run command:
12. docker run --name jenkins-blueocean --detach ^
13.   --network jenkins --env DOCKER_HOST=tcp://docker:2376 ^
14. --env DOCKER_CERT_PATH=/certs/client --env DOCKER_TLS_VERIFY=1 ^

```
15.  --volume jenkins-data:/var/jenkins_home ^
16.  --volume jenkins-docker-certs:/certs/client:ro ^
17.  --volume "%HOMEDRIVE%%HOMEPATH%":/home ^
18.  --restart=on-failure ^
19.--env JAVA_OPTS="-
    Dhudson.plugins.git.GitSCM.ALLOW_LOCAL_CHECKOUT=true" ^
     --publish 8080:8080 --publish 50000:50000 myjenkins-blueocean:2.361.4-1
```
20.Proceed to the Setup wizard.

Accessing the Docker container

If you have some experience with Docker and you wish or need to access your Docker container through a terminal/command prompt using the docker exec command, you can add an option like --name jenkins-tutorial to the docker exec command. That will access the Jenkins Docker container named "jenkins-tutorial".

This means you could access your docker container (through a separate terminal/command prompt window) with a docker exec command like:

docker exec -it jenkins-blueocean bash

Accessing the Docker logs

There is a possibility you may need to access the Jenkins console log, for instance, when Unlocking Jenkins as part of the Post-installation setup wizard.

The Jenkins console log is easily accessible through the terminal/command prompt window from which you executed the docker run … command. In case if needed you can also access the Jenkins console log through the Docker logs of your container using the following command:

docker logs <docker-container-name>

Your <docker-container-name> can be obtained using the docker ps command.

Accessing the Jenkins home directory

There is a possibility you may need to access the Jenkins home directory, for instance, to check the details of a Jenkins build in the workspace subdirectory.

If you mapped the Jenkins home directory (/var/jenkins_home) to one on your machine's local file system (i.e. in the docker run … command above), then you can access the contents of this directory through your machine's usual terminal/command prompt.

Otherwise, if you specified the --volume jenkins-data:/var/jenkins_home option in the docker run … command, you can access the contents of the Jenkins home directory through your container's terminal/command prompt using the docker container exec command:

```
docker container exec -it <docker-container-name> bash
```

As mentioned above, your <docker-container-name> can be obtained using the docker container ls command. If you specified the --name jenkins-blueocean option in the docker container run …command above (see also Accessing the Jenkins/Blue Ocean Docker container), you can simply use the docker container exec command:

```
docker container exec -it jenkins-blueocean bash
```

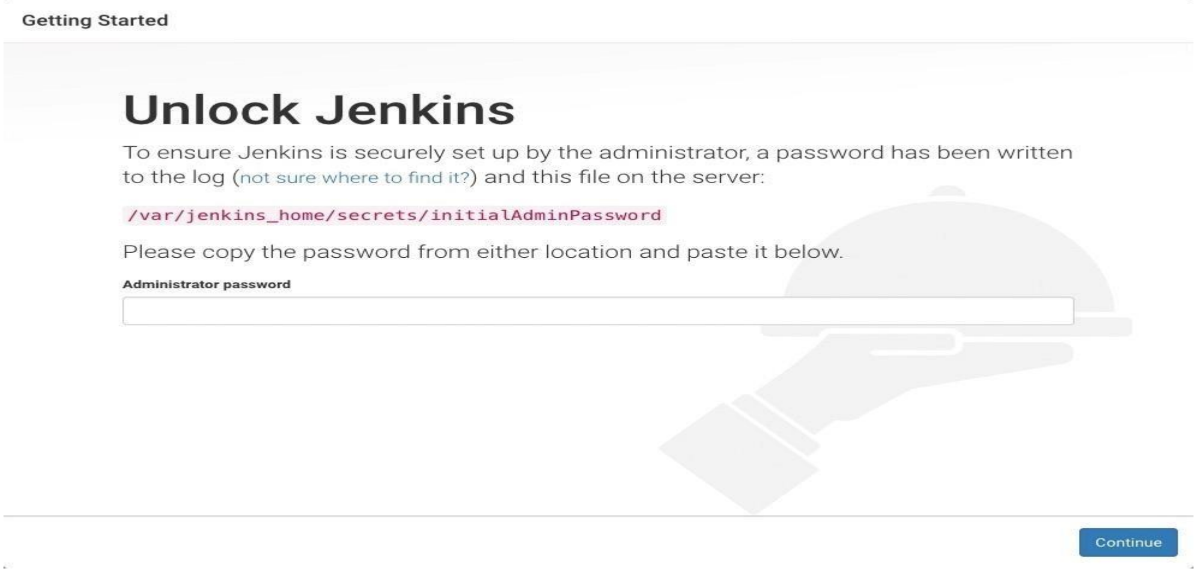
Setup wizard

Before you can access Jenkins, there are a few quick "one-off" steps you'll need to perform.

Unlocking Jenkins

When you first access a new Jenkins instance, you are asked to unlock it using an automatically-generated password.

1. After the 2 sets of asterisks appear in the terminal/command prompt window, browse to http://localhost:8080 and wait until the **Unlock Jenkins** page appears.



Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:

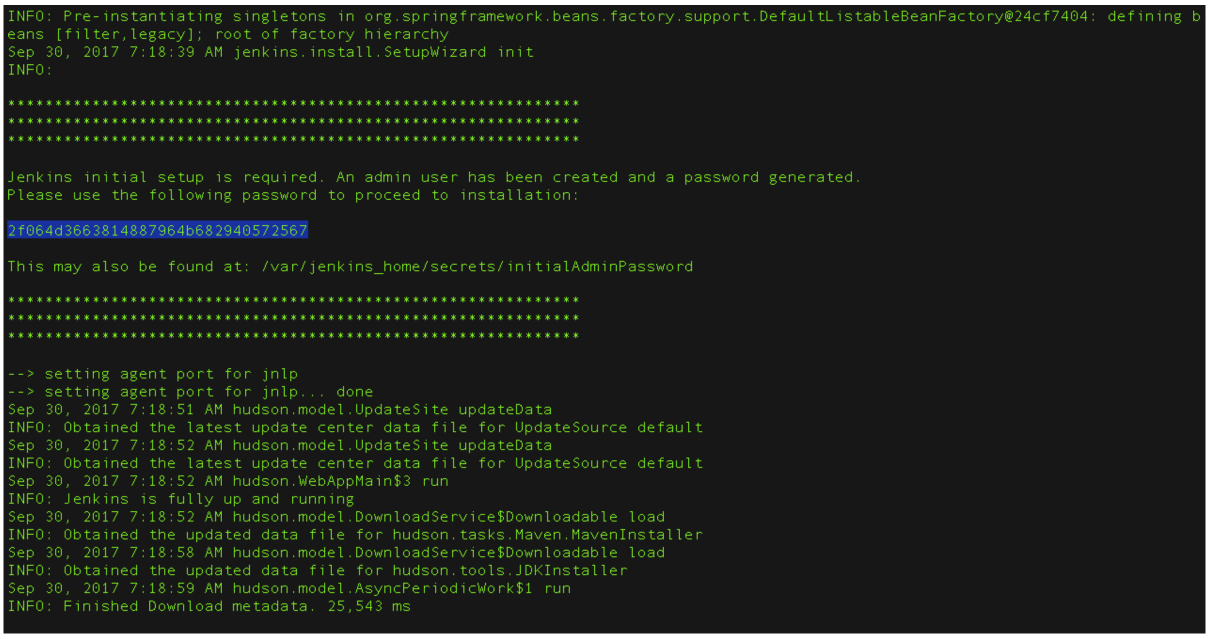
/var/jenkins_home/secrets/initialAdminPassword

Please copy the password from either location and paste it below.

Administrator password

2. Display the Jenkins console log with the command:
   ```
   docker logs jenkins-blueocean
   ```

3. From your terminal/command prompt window again, copy the automatically-generated alphanumeric password (between the 2 sets of asterisks).



```
INFO: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory@24cf7404: defining b
eans [filter,legacy]; root of factory hierarchy
Sep 30, 2017 7:18:39 AM jenkins.install.SetupWizard init
INFO:

*************************************************************
*************************************************************
*************************************************************

Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:

2f064d3663814887964b682940572567

This may also be found at: /var/jenkins_home/secrets/initialAdminPassword

*************************************************************
*************************************************************
*************************************************************

--> setting agent port for jnlp
--> setting agent port for jnlp... done
Sep 30, 2017 7:18:51 AM hudson.model.UpdateSite updateData
INFO: Obtained the latest update center data file for UpdateSource default
Sep 30, 2017 7:18:52 AM hudson.model.UpdateSite updateData
INFO: Obtained the latest update center data file for UpdateSource default
Sep 30, 2017 7:18:52 AM hudson.WebAppMain$3 run
INFO: Jenkins is fully up and running
Sep 30, 2017 7:18:52 AM hudson.model.DownloadService$Downloadable load
INFO: Obtained the updated data file for hudson.tasks.Maven.MavenInstaller
Sep 30, 2017 7:18:58 AM hudson.model.DownloadService$Downloadable load
INFO: Obtained the updated data file for hudson.tools.JDKInstaller
Sep 30, 2017 7:18:59 AM hudson.model.AsyncPeriodicWork$1 run
INFO: Finished Download metadata. 25,543 ms
```

4. On the **Unlock Jenkins** page, paste this password into the **Administrator password** field and click **Continue**.

Customizing Jenkins with plugins

After unlocking Jenkins, the **Customize Jenkins** page appears.

On this page, click **Install suggested plugins**.

The setup wizard shows the progression of Jenkins being configured and the suggested plugins being installed. This process may take a few minutes.

Creating the first administrator user

Finally, Jenkins asks you to create your first administrator user.

1. When the **Create First Admin User** page appears, specify your details in the respective fields and click **Save and Finish**.

2. When the **Jenkins is ready** page appears, click **Start using Jenkins**.

Notes:

- This page may indicate **Jenkins is almost ready!** instead and if so, click **Restart**.
- If the page doesn't automatically refresh after a minute, use your web browser to refresh the page manually.

3. If required, log in to Jenkins with the credentials of the user you just created and you're ready to start using Jenkins!

Stopping and restarting Jenkins

Throughout the remainder of this tutorial, you can stop your Docker container by running:

```
docker stop jenkins-blueocean jenkins-docker
```

To restart your Docker container:

1. Run the same docker run … commands you ran for macOS, Linux or Windows above.
2. Browse to http://localhost:8080.
3. Wait until the log in page appears and log in.

Fork and clone the sample repository

Obtain the simple "Hello world!" Java application from GitHub, by forking the sample repository of the application's source code into your own GitHub account and then cloning this fork locally.

1. Ensure you are signed in to your GitHub account. If you don't yet have a GitHub account, sign up for a free one on the GitHub website.

2. Fork the simple-java-maven-app on GitHub into your local GitHub account. If you need help with this process, refer to the Fork A Repo documentation on the GitHub website for more information.

3. Clone your forked simple-java-maven-app repository (on GitHub) locally to your machine. To begin this process, do either of the following (where <your-username> is the name of your user account on your operating system):

   - If you have the GitHub Desktop app installed on your machine:
     a. In GitHub, click the green **Clone or download** button on your forked repository, then **Open in Desktop**.

b. In GitHub Desktop, before clicking **Clone** on the **Clone a Repository** dialog box, ensure **Local Path** for:

  - macOS is /Users/<your-username>/Documents/GitHub/simple-java-maven-app
  - Linux is /home/<your-username>/GitHub/simple-java-maven-app
  - Windows is C:\Users\<your-username>\Documents\GitHub\simple-java-maven-app

- Otherwise:

  a. Open up a terminal/command line prompt and cd to the appropriate directory on:
    - macOS - /Users/<your-username>/Documents/GitHub/
    - Linux - /home/<your-username>/GitHub/
    - Windows - C:\Users\<your-username>\Documents\GitHub\ (although use a Git bash command line window as opposed to the usual Microsoft command prompt)

  b. Run the following command to continue/complete cloning your forked repo:
  git clone https://github.com/YOUR-GITHUB-ACCOUNT-NAME/simple-java-maven-app
  where YOUR-GITHUB-ACCOUNT-NAME is the name of your GitHub account.

Create your Pipeline project in Jenkins

1. Go back to Jenkins, log in again if necessary and click **create new jobs** under **Welcome to Jenkins!**

   Note: If you don't see this, click **New Item** at the top left.

2. In the **Enter an item name** field, specify the name for your new Pipeline project (e.g. simple-java-maven-app).

3. Scroll down and click **Pipeline**, then click **OK** at the end of the page.

4. (*Optional*) On the next page, specify a brief description for your Pipeline in the **Description** field (e.g. An entry-level Pipeline demonstrating how to use Jenkins to build a simple Java application with Maven.)

5. Click the **Pipeline** tab at the top of the page to scroll down to the **Pipeline** section.

6. From the **Definition** field, choose the **Pipeline script from SCM** option. This option instructs Jenkins to obtain your Pipeline from Source Control Management (SCM), which will be your locally cloned Git repository.

7. From the **SCM** field, choose **Git**.

8. In the **Repository URL** field, specify the directory path of your locally cloned repository above, which is from your user account/home directory on your host machine, mapped to the /home directory of the Jenkins container - i.e.
   - For macOS - /home/Documents/GitHub/simple-java-maven-app
   - For Linux - /home/GitHub/simple-java-maven-app
   - For Windows - /home/Documents/GitHub/simple-java-maven-app

9. Click **Save** to save your new Pipeline project. You're now ready to begin creating your Jenkins file, which you'll be checking into your locally cloned Git repository.

Create your initial Pipeline as a Jenkins file

You're now ready to create your Pipeline that will automate building your Java application with Maven in Jenkins. Your Pipeline will be created as a Jenkins file, which will be committed to your locally cloned Git repository (simple-java-maven-app).

This is the foundation of "Pipeline-as-Code", which treats the continuous delivery pipeline as a part of the application to be versioned and reviewed like any other code. Read more about Pipeline and what a Jenkins file is in the Pipeline and Using a Jenkins file sections of the User Handbook.

First, create an initial Pipeline to download a Maven Docker image and run it as a Docker container (which will build your simple Java application). Also add a "Build" stage to the Pipeline that begins orchestrating this whole process.

1. Using your favourite text editor or IDE, create and save new text file with the name Jenkins file at the root of your local simple-java-maven-app Git repository.

2. Copy the following Declarative Pipeline code and paste it into your empty Jenkins file:

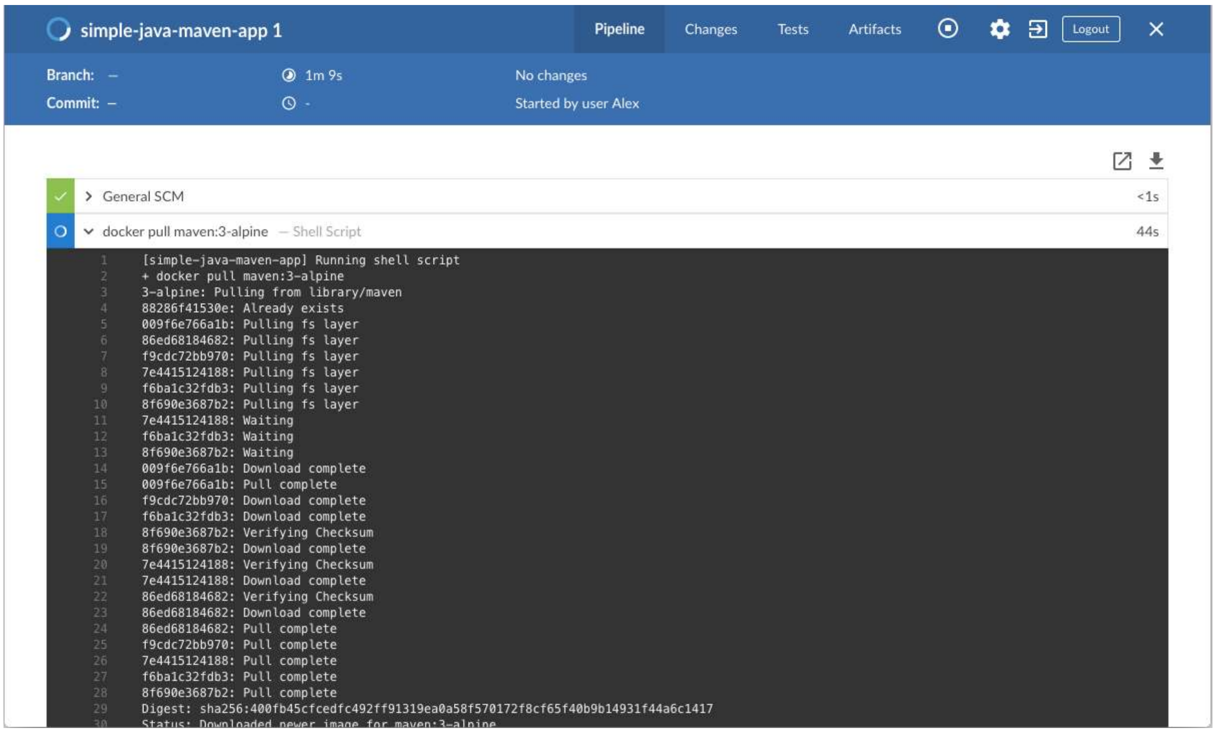
```
pipeline {
    agent {
        docker {
            image 'maven:3.8.1-adoptopenjdk-11'
        args '-v /root/.m2:/root/.m2'
        }
    }
    stages {
        stage('Build') {
            steps {
                sh 'mvn -B -DskipTests clean package'
            }
        }
    }
}
```

	This image parameter (of the agent section's docker parameter) downloads the maven:3.8.1-adoptopenjdk-11 Docker image (if it's not already available on your machine) and runs this image as a separate container. This means that: ○ You'll have separate Jenkins and Maven containers running locally in Docker. ○ The Maven container becomes the agent that Jenkins uses to run your Pipeline project. However, this container is short-lived - its lifespan is only that of the duration of your Pipeline's execution.
	This args parameter creates a reciprocal mapping between the /root/.m2 (i.e. Maven repository) directories in the short-lived Maven Docker container and that of your Docker host's filesystem.
	Defines a stage (directive) called Build that appears on the Jenkins UI.
	This sh step (of the steps section) runs the Maven command to cleanly build your Java application (without running any tests).

17. Save your edited Jenkins file and commit it to your local simple-java-maven-app Git repository. E.g., Within the simple-java-maven-app directory, run the commands:
    git add.
    then
    git commit -m "Add initial Jenkins file"

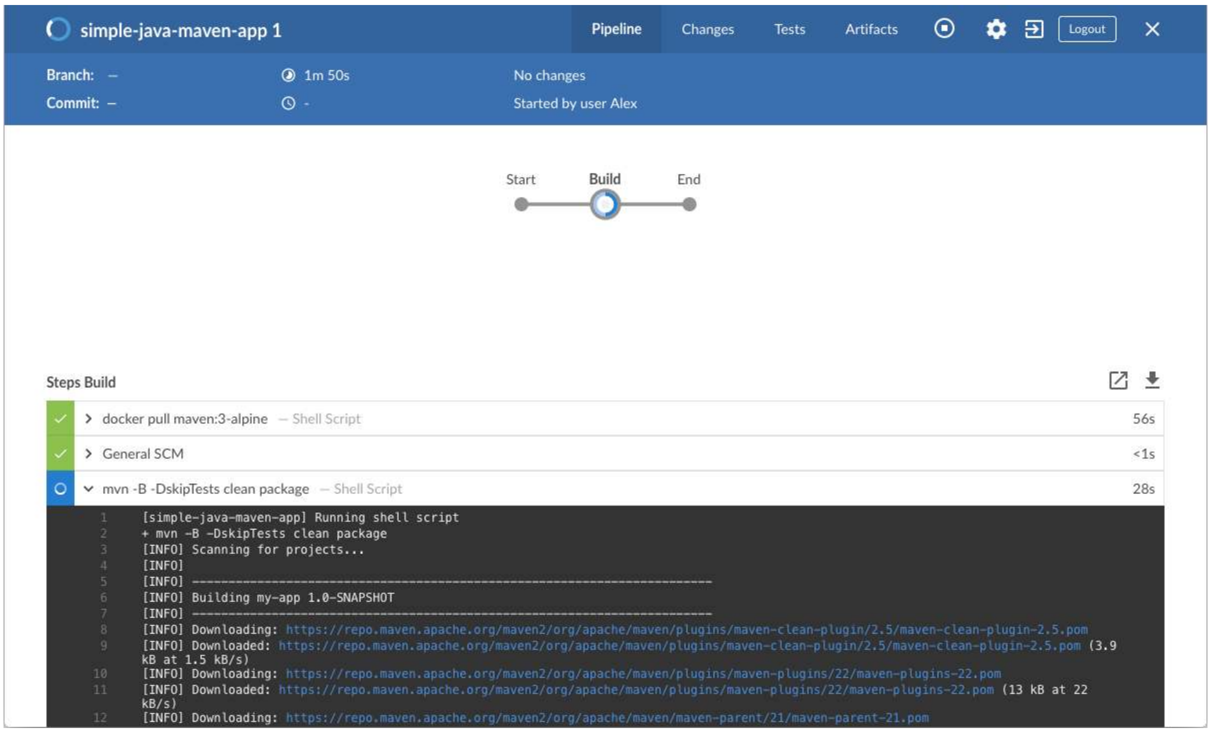
18. Go back to Jenkins again, log in again if necessary and click **Open Blue Ocean** on the left to access Jenkins's Blue Ocean interface.

19. In the **This job has not been run** message box, click **Run**, then quickly click the **OPEN** link which appears briefly at the lower-right to see Jenkins running your Pipeline project. If you weren't able to click the **OPEN** link, click the row on the main Blue Ocean interface to access this feature.
    Note: You may need to wait several minutes for this first run to complete. After making a clone of your local simple-java-maven-app Git repository itself, Jenkins:
    - Initially queues the project to be run on the agent.
    - Downloads the Maven Docker image and runs it in a container on Docker.



```
[simple-java-maven-app] Running shell script
+ docker pull maven:3-alpine
3-alpine: Pulling from library/maven
88286f41530e: Already exists
009f6e766a1b: Pulling fs layer
86ed68184682: Pulling fs layer
f9cdc72bb970: Pulling fs layer
7e4415124188: Pulling fs layer
f6ba1c32fdb3: Pulling fs layer
8f690e3687b2: Pulling fs layer
7e4415124188: Waiting
f6ba1c32fdb3: Waiting
8f690e3687b2: Waiting
009f6e766a1b: Download complete
009f6e766a1b: Pull complete
f9cdc72bb970: Download complete
f6ba1c32fdb3: Download complete
8f690e3687b2: Verifying Checksum
8f690e3687b2: Download complete
7e4415124188: Verifying Checksum
7e4415124188: Download complete
86ed68184682: Verifying Checksum
86ed68184682: Download complete
86ed68184682: Pull complete
f9cdc72bb970: Pull complete
7e4415124188: Pull complete
f6ba1c32fdb3: Pull complete
8f690e3687b2: Pull complete
Digest: sha256:400fb45cfcedfc492ff91319ea0a58f570172f8cf65f40b9b14931f44a6c1417
Status: Downloaded newer image for maven:3-alpine
```

Runs the Build stage (defined in the Jenkins file) on the Maven container. During this time, Maven downloads many artifacts necessary to build your Java application, which will ultimately be stored in Jenkins's local Maven repository (in the Docker host's filesystem).

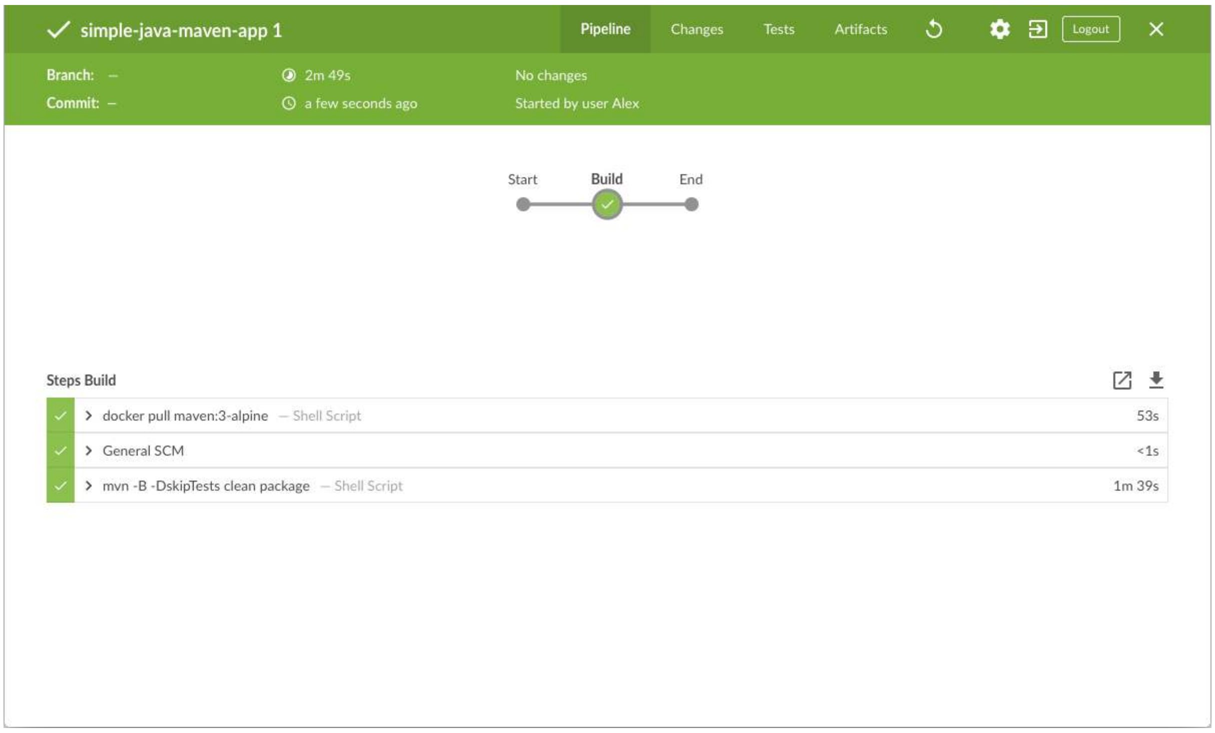


Branch: –
Commit: –
1m 50s
-
No changes
Started by user Alex

docker pull maven:3-alpine — Shell Script 56s
General SCM <1s
mvn -B -DskipTests clean package — Shell Script 28s

```
[simple-java-maven-app] Running shell script
+ mvn -B -DskipTests clean package
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building my-app 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-clean-plugin/2.5/maven-clean-plugin-2.5.pom
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-clean-plugin/2.5/maven-clean-plugin-2.5.pom (3.9
kB at 1.5 kB/s)
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-plugins/22/maven-plugins-22.pom
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-plugins/22/maven-plugins-22.pom (13 kB at 22
kB/s)
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/maven-parent/21/maven-parent-21.pom
```

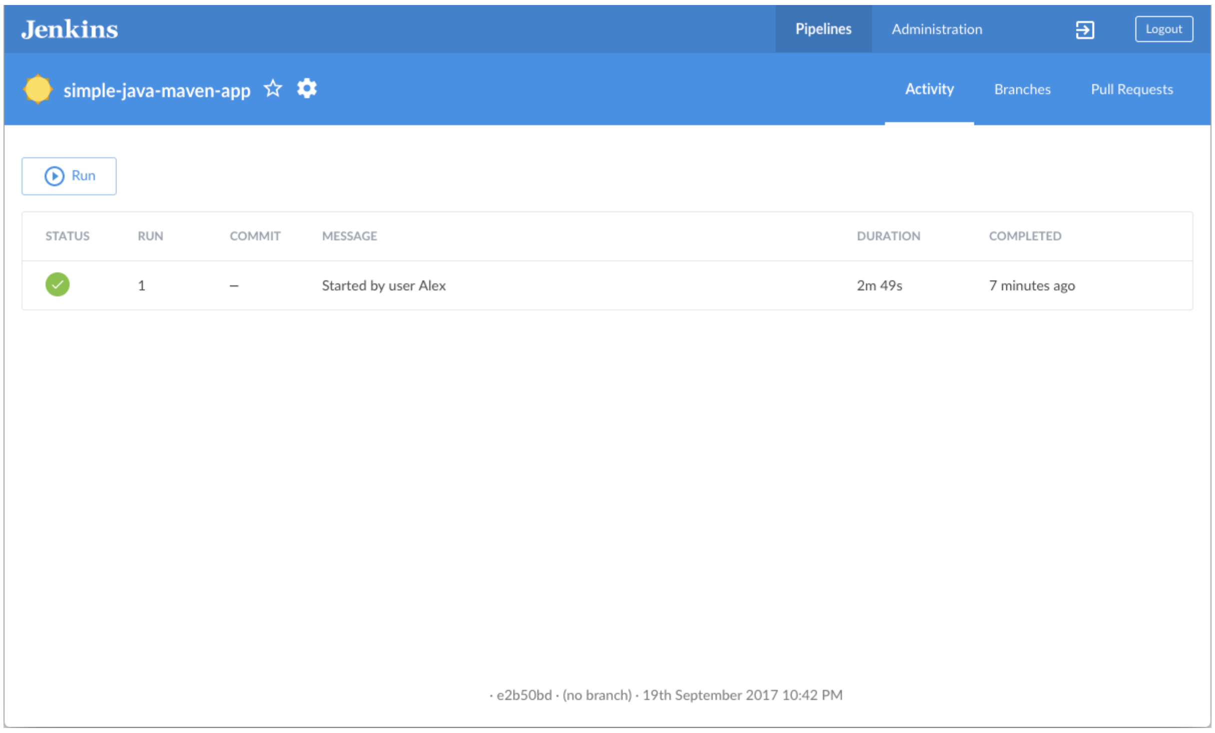
20. The Blue Ocean interface turns green if Jenkins built your Java application successfully.



Branch: –
Commit: –
2m 49s
a few seconds ago
No changes
Started by user Alex

docker pull maven:3-alpine — Shell Script 53s
General SCM <1s
mvn -B -DskipTests clean package — Shell Script 1m 39s

21. Click the **X** at the top-right to return to the main Blue Ocean interface.



STATUS	RUN	COMMIT	MESSAGE	DURATION	COMPLETED
✓	1	–	Started by user Alex	2m 49s	7 minutes ago

· e2b50bd · (no branch) · 19th September 2017 10:42 PM

Add a test stage to your Pipeline

1. Go back to your text editor/IDE and ensure your Jenkins file is open.
2. Copy and paste the following Declarative Pipeline syntax immediately under the Build stage of your Jenkins file:

```
        stage('Test') {
            steps {
                sh 'mvn test'
            }
            post {
                always {
                    junit 'target/surefire-reports/*.xml'
                }
            }
        }
```

so that you end up with:

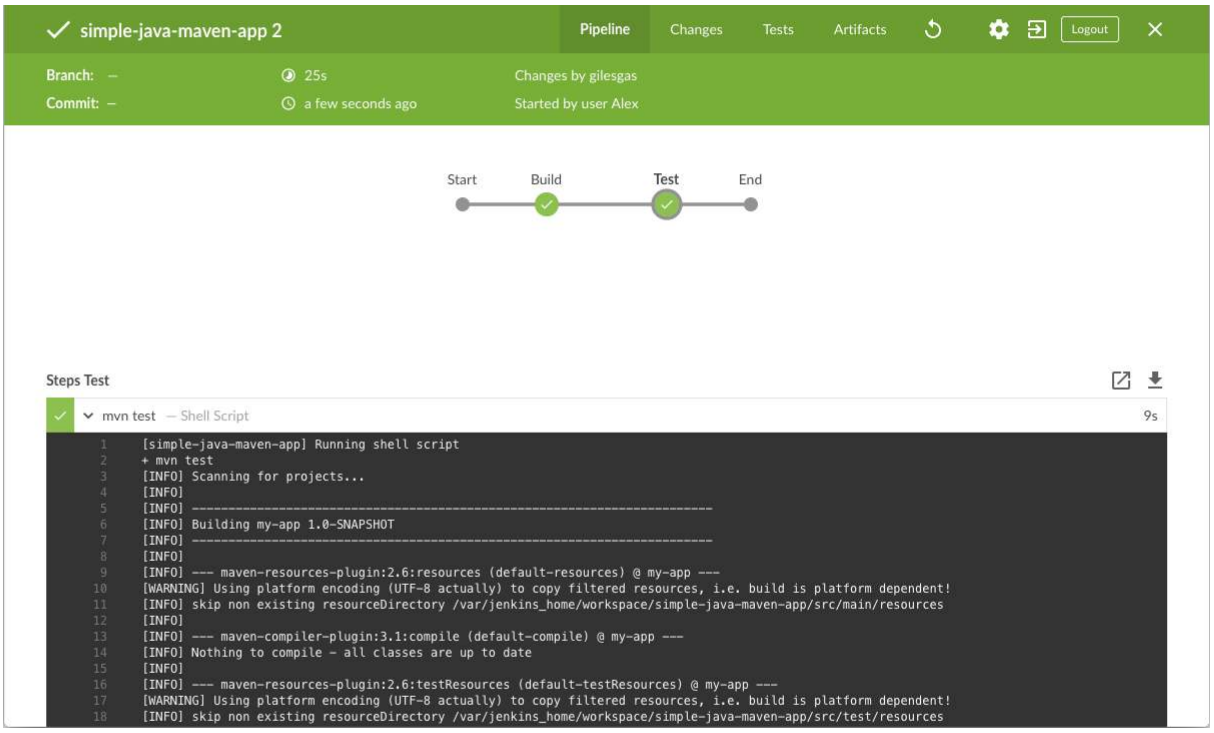
```
pipeline {
    agent {
        docker {
            image 'maven:3.8.1-adoptopenjdk-11'
            args '-v /root/.m2:/root/.m2'
        }
    }
    stages {
        stage('Build') {
            steps {
                sh 'mvn -B -DskipTests clean package'
            }
        }
        stage('Test') {
            steps {
                sh 'mvn test'
            }
            post {
                always {
                    junit 'target/surefire-reports/*.xml'
                }
            }
        }
    }
}
```

	Defines a stage (directive) called Test that appears on the Jenkins UI.
	This sh step (of the steps section) executes the Maven command to run the unit test on your simple Java application. This command also generates a JUnit XML report, which is saved to the target/sure-fire-reports directory (within the /var/Jenkins home/workspace/simple-java-maven-app directory in the Jenkins container).
	This junit step (provided by the JUnit Plugin) archives the JUnit XML report (generated by the mvn test command above) and exposes the results through the Jenkins interface. In Blue Ocean, the results are accessible through the **Tests** page of a Pipeline run. The post section's always condition that contains this junit step ensures that the step is *always* executed *at the completion* of the Test stage, regardless of the stage's outcome.

12. Save your edited Jenkins file and commit it to your local simple-java-maven-app Git repository. E.g., Within the simple-java-maven-app directory, run the commands:
    git stage.
    then
    git commit -m "Add 'Test' stage"

13. Go back to Jenkins again, log in again if necessary and ensure you've accessed Jenkins's Blue Ocean interface.

14. Click **Run** at the top left, then quickly click the **OPEN** link which appears briefly at the lower-right to see Jenkins running your amended Pipeline project. If you weren't able to click the **OPEN** link, click the *top* row on the Blue Ocean interface to access this feature.



```
[simple-java-maven-app] Running shell script
+ mvn test
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building my-app 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ my-app ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory /var/jenkins_home/workspace/simple-java-maven-app/src/main/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ my-app ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ my-app ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory /var/jenkins_home/workspace/simple-java-maven-app/src/test/resources
```

15. Click the **X** at the top-right to return to the main Blue Ocean interface.

Add a final deliver stage to your Pipeline

1. Go back to your text editor/IDE and ensure your Jenkins file is open.
2. Copy and paste the following Declarative Pipeline syntax immediately under the Test stage of your Jenkins file:

```
3.      stage('Deliver') {
4.          steps {
5.              sh './jenkins/scripts/deliver.sh'
6.          }
        }
```

and add a skipStagesAfterUnstable option so that you end up with:

```
pipeline {
    agent {
        docker {
            image 'maven:3.8.1-adoptopenjdk-11'
            args '-v /root/.m2:/root/.m2'
        }
    }
    options {
        skipStagesAfterUnstable()
    }
    stages {
        stage('Build') {
            steps {
                sh 'mvn -B -DskipTests clean package'
            }
        }
        stage('Test') {
            steps {
                sh 'mvn test'
            }
            post {
                always {
                    junit 'target/surefire-reports/*.xml'
                }
            }
        }
        stage('Deliver') {
            steps {
                sh './jenkins/scripts/deliver.sh'
            }
        }
    }
}
```

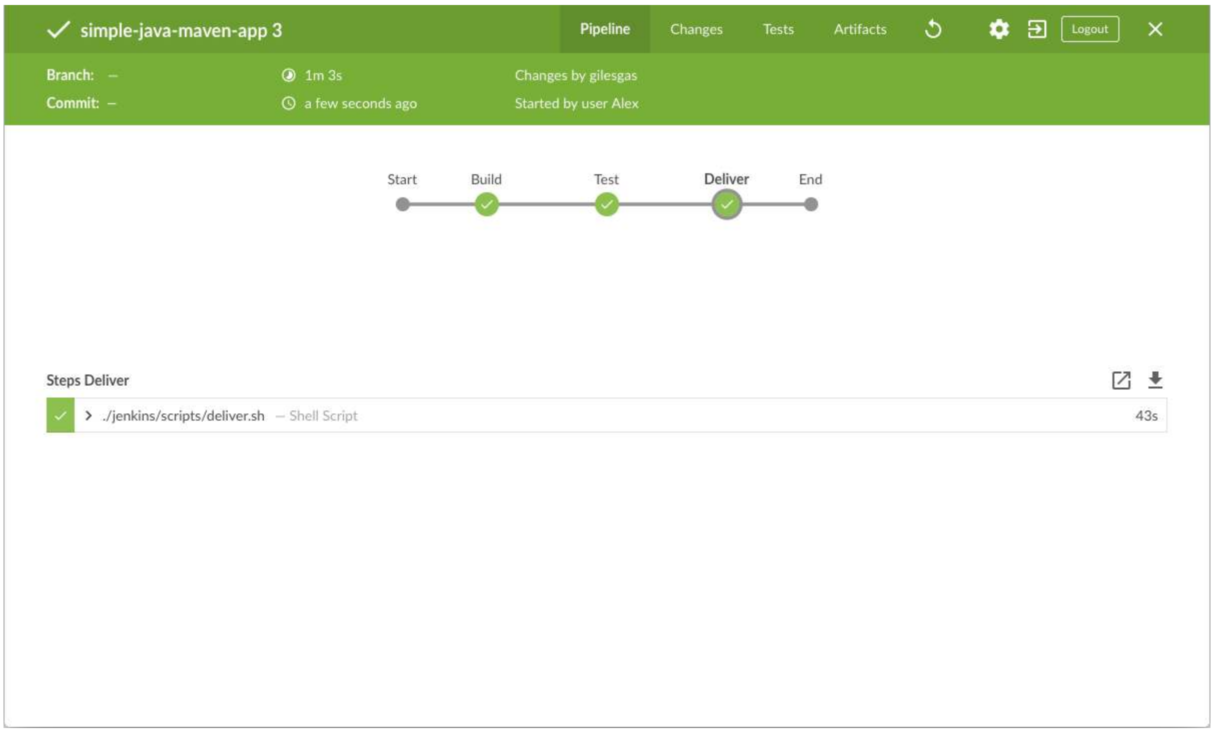
	Defines a new stage called Deliver that appears on the Jenkins UI.
	This sh step (of the steps section) runs the shell script deliver.sh located in the Jenkins/scripts directory from the root of the simple-java-maven-app repository. Explanations about what this script does are covered in the deliver.sh file itself. As a general principle, it's a good idea to keep your Pipeline code (i.e. the Jenkins file) as tidy as possible and place more complex build steps (particularly for stages consisting of 2 or more steps) into separate shell script files like the deliver.sh file. This ultimately makes maintaining your Pipeline code easier, especially if your Pipeline gains more complexity.

7. Save your edited Jenkins file and commit it to your local simple-java-maven-app Git repository. E.g., Within the simple-java-maven-app directory, run the commands:
   git stage.
   then
   git commit -m "Add 'Deliver' stage"

8. Go back to Jenkins again, log in again if necessary and ensure you've accessed Jenkins's Blue Ocean interface.

9. Click **Run** at the top left, then quickly click the **OPEN** link which appears briefly at the lower-right to see Jenkins running your amended Pipeline project. If you weren't able to click the **OPEN** link, click the *top* row on the Blue Ocean interface to access this feature.

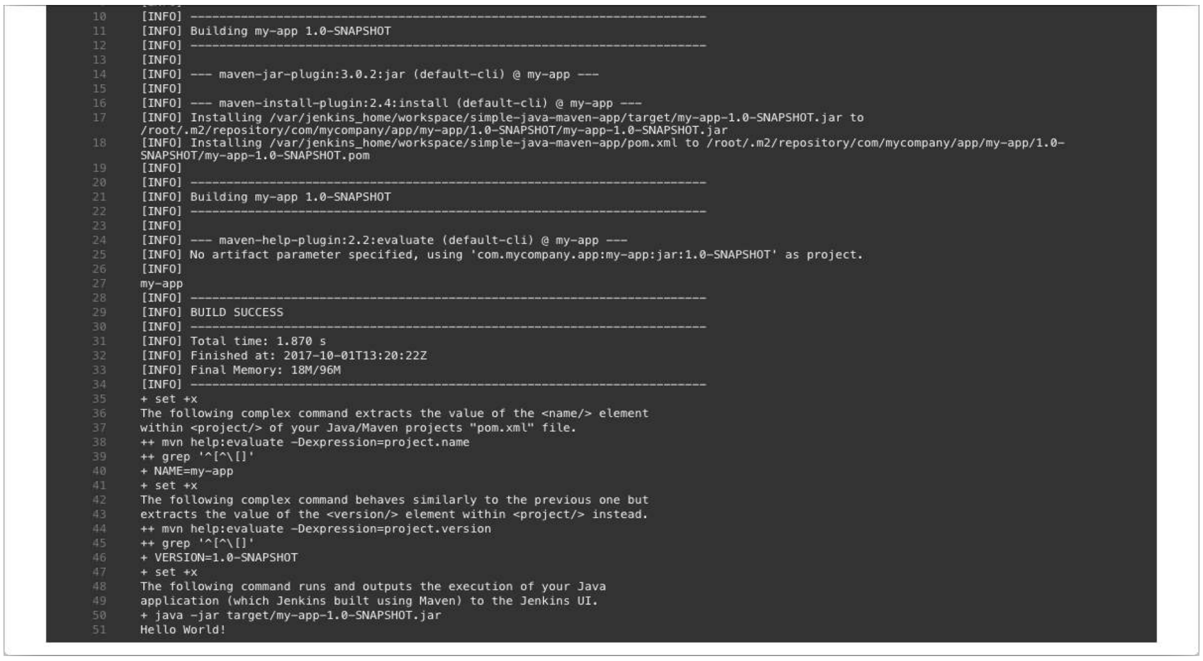
   If your amended Pipeline ran successfully, here's what the Blue Ocean interface should look like. Notice the additional "Deliver" stage. Click on the previous "Test" and "Build" stage circles to access the outputs from those stages.



Branch: –
Commit: –
1m 3s
a few seconds ago
Changes by gilesgas
Started by user Alex

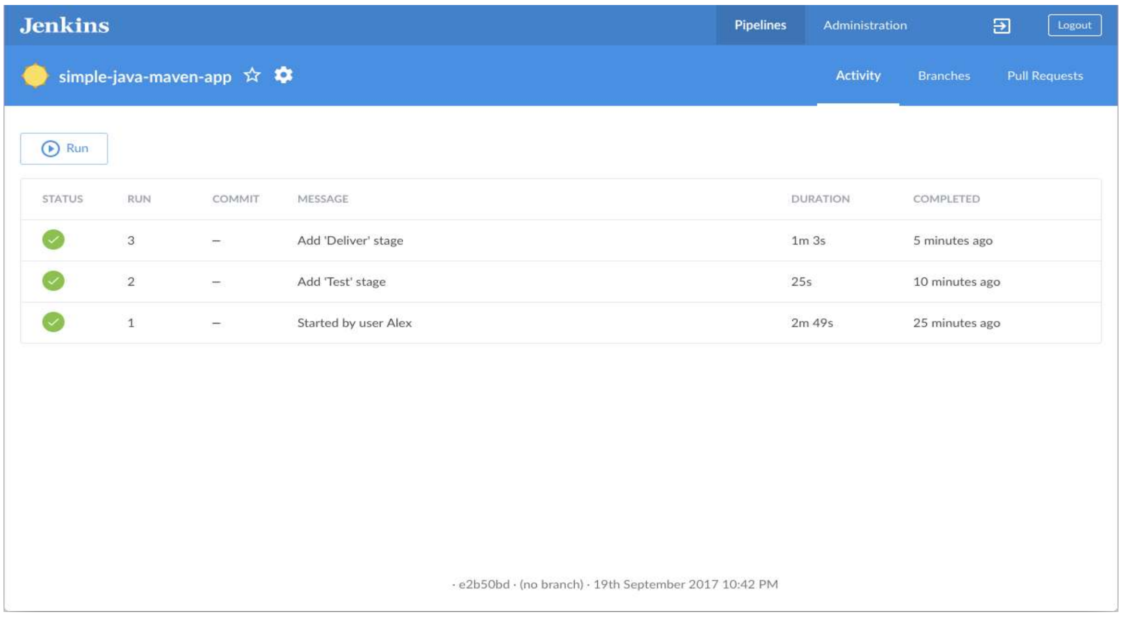
./jenkins/scripts/deliver.sh – Shell Script 43s

Here's what the output of the "Deliver" stage should look like, showing you the execution results of your Java application at the end.



```
[INFO] ------------------------------------------------------------------------
[INFO] Building my-app 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-jar-plugin:3.0.2:jar (default-cli) @ my-app ---
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-cli) @ my-app ---
[INFO] Installing /var/jenkins_home/workspace/simple-java-maven-app/target/my-app-1.0-SNAPSHOT.jar to
/root/.m2/repository/com/mycompany/app/my-app/1.0-SNAPSHOT/my-app-1.0-SNAPSHOT.jar
[INFO] Installing /var/jenkins_home/workspace/simple-java-maven-app/pom.xml to /root/.m2/repository/com/mycompany/app/my-app/1.0-
SNAPSHOT/my-app-1.0-SNAPSHOT.pom
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building my-app 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-help-plugin:2.2:evaluate (default-cli) @ my-app ---
[INFO] No artifact parameter specified, using 'com.mycompany.app:my-app:jar:1.0-SNAPSHOT' as project.
[INFO]
my-app
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.870 s
[INFO] Finished at: 2017-10-01T13:20:22Z
[INFO] Final Memory: 18M/96M
[INFO] ------------------------------------------------------------------------
+ set +x
The following complex command extracts the value of the <name/> element
within <project/> of your Java/Maven projects "pom.xml" file.
++ mvn help:evaluate -Dexpression=project.name
++ grep '^[^\[]'
+ NAME=my-app
+ set +x
The following complex command behaves similarly to the previous one but
extracts the value of the <version/> element within <project/> instead.
++ mvn help:evaluate -Dexpression=project.version
++ grep '^[^\[]'
+ VERSION=1.0-SNAPSHOT
+ set +x
The following command runs and outputs the execution of your Java
application (which Jenkins built using Maven) to the Jenkins UI.
+ java -jar target/my-app-1.0-SNAPSHOT.jar
Hello World!
```

10. Click the **X** at the top-right to return to the main Blue Ocean interface, which lists your previous Pipeline runs in reverse chronological order.



STATUS	RUN	COMMIT	MESSAGE	DURATION	COMPLETED
✓	3	–	Add 'Deliver' stage	1m 3s	5 minutes ago
✓	2	–	Add 'Test' stage	25s	10 minutes ago
✓	1	–	Started by user Alex	2m 49s	25 minutes ago

· e2b50bd · (no branch) · 19th September 2017 10:42 PM

CONCLUSION:

The "Build", "Test" and "Deliver" stages you created above are the basis for building more complex Java applications with Maven in Jenkins, as well as Java and Maven applications that integrate with other technology stacks.

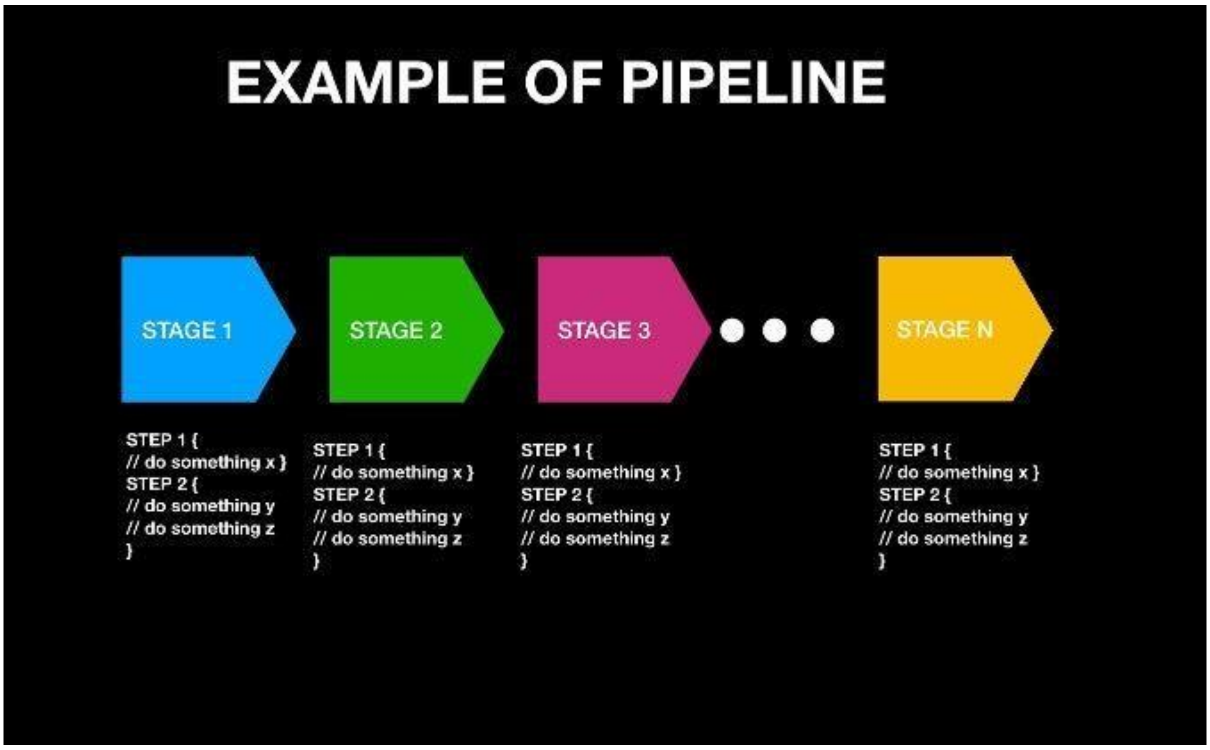
Because Jenkins is extremely extensible, it can be modified and configured to handle practically any aspect of build orchestration and automation.

Exercise 7:

Module Name: Implementation of CICD with Java and open-source stack
Configure the Jenkins tool with the required paths, path variables, users and pipeline views.

What is a pipeline?

A **pipeline** is a sequence of events or jobs that can be executed. The easiest way to understand a pipeline is to visualize a sequence of stages, like this:


EXAMPLE OF PIPELINE
STAGE 1
STAGE 2
STAGE 3
STAGE N
STEP 1 { // do something x } STEP 2 { // do something y // do something z }

Here, you should see two familiar concepts: *Stage* and *Step*.

- **Stage:** A block that contains a series of steps. A stage block can be named anything; it is used to visualize the pipeline process.
- **Step:** A task that says what to do. Steps are defined inside a stage block.

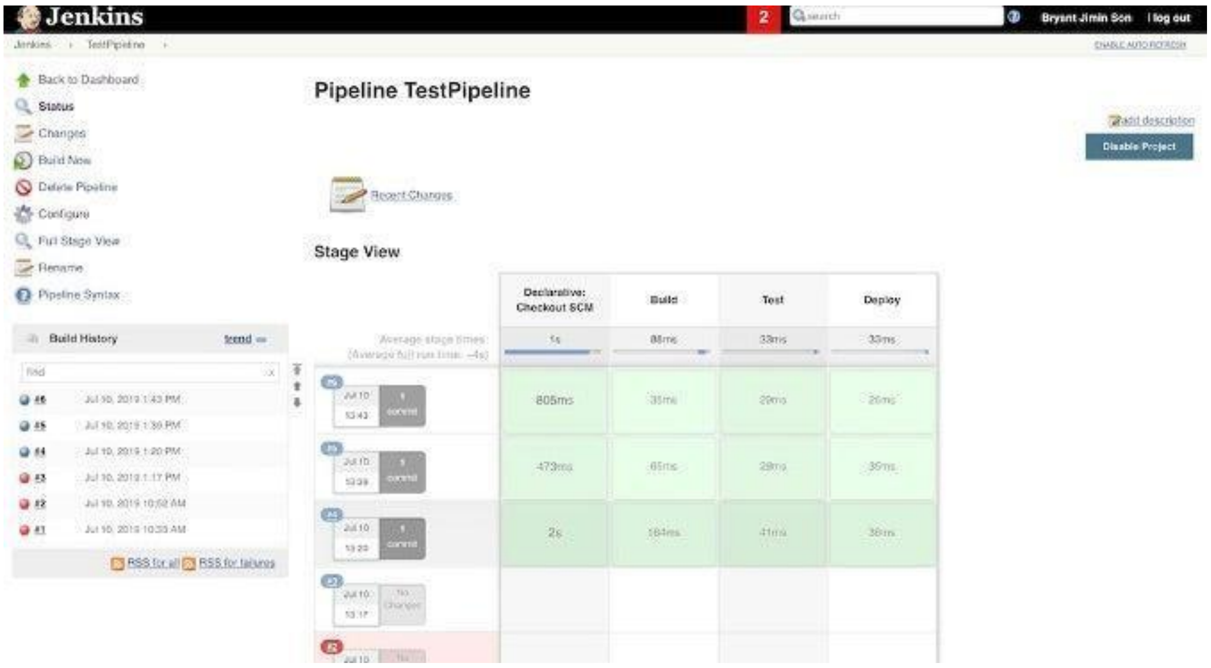
In the example diagram above, Stage 1 can be named "Build," "Gather Information," or whatever, and a similar idea is applied for the other stage blocks. "Step" simply says what to execute, and this can be a simple print command (e.g., **echo "Hello, World"**), a program-execution command (e.g., **java HelloWorld**), a shell-execution command (e.g., **chmod 755 Hello**), or any other command—as long as it is recognized as an executable command through the Jenkins environment.

The Jenkins pipeline is provided as a *codified script* typically called a **Jenkins file**, although the file name can be different. Here is an example of a simple Jenkins pipeline file.

```
// Example of Jenkins pipeline script

pipeline {
  stages {
    stage("Build") {
      steps {
        // Just print a Hello, Pipeline to the console
        echo "Hello, Pipeline!"
        // Compile a Java file. This requires JDKconfiguration from Jenkins
        javac HelloWorld.java
        // Execute the compiled Java binary called HelloWorld. This requires JDK
configuration from Jenkins
        java HelloWorld
        // Executes the Apache Maven commands, clean then package. This requires
Apache Maven configuration from Jenkins
        mvn clean package ./HelloPackage
        // List the files in current directory path by executing a default shell command
        sh "ls -ltr"
      }
    }
   // And next stages if you want to define further...
  } // End of stages
} // End of pipeline
```

It's easy to see the structure of a Jenkins pipeline from this sample script. Note that some commands, like **java**, **javac**, and **mvn**, are not available by default, and they need to be installed and configured through Jenkins. Therefore:

A **Jenkins pipeline** is the way to execute a Jenkins job sequentially in a defined way by codifying it and structuring it inside multiple blocks that can include multiple steps containing tasks.



How to build a Jenkins pipeline

Before starting this, you'll need:

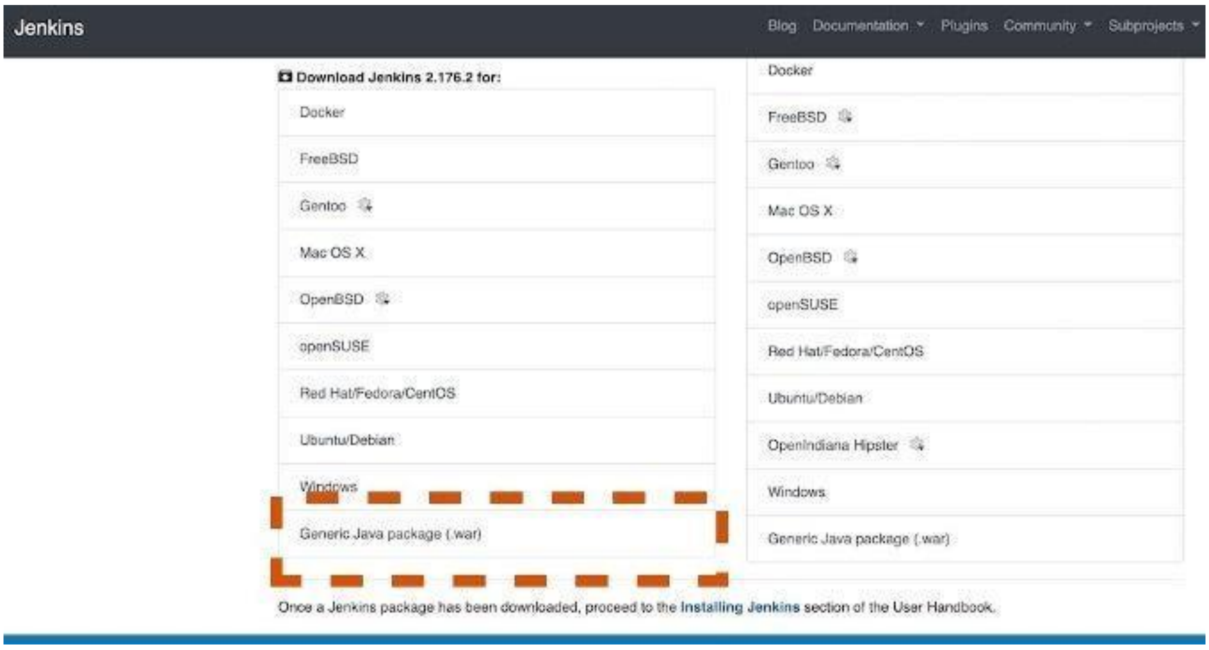
- **Java Development Kit:** If you don't already have it, install a JDK and add it to the environment path so a Java command (like **java jar**) can be executed through a terminal. This is necessary to leverage the Java Web Archive (WAR) version of Jenkins that is used in this tutorial (although you can use any other distribution).

- **Basic computer operations:** You should know how to type some code, execute basic Linux commands through the shell, and open a browser.

Let's get started.

Step 1: Download Jenkins

Navigate to the Jenkins download page. Scroll down to **Generic Java package (.war)** and click on it to download the file; save it someplace where you can locate it easily. (If you choose another Jenkins distribution, the rest of tutorial steps should be pretty much the same, except for Step 2.) The reason to use the WAR file is that it is a one-time executable file that is easily executable and removable.



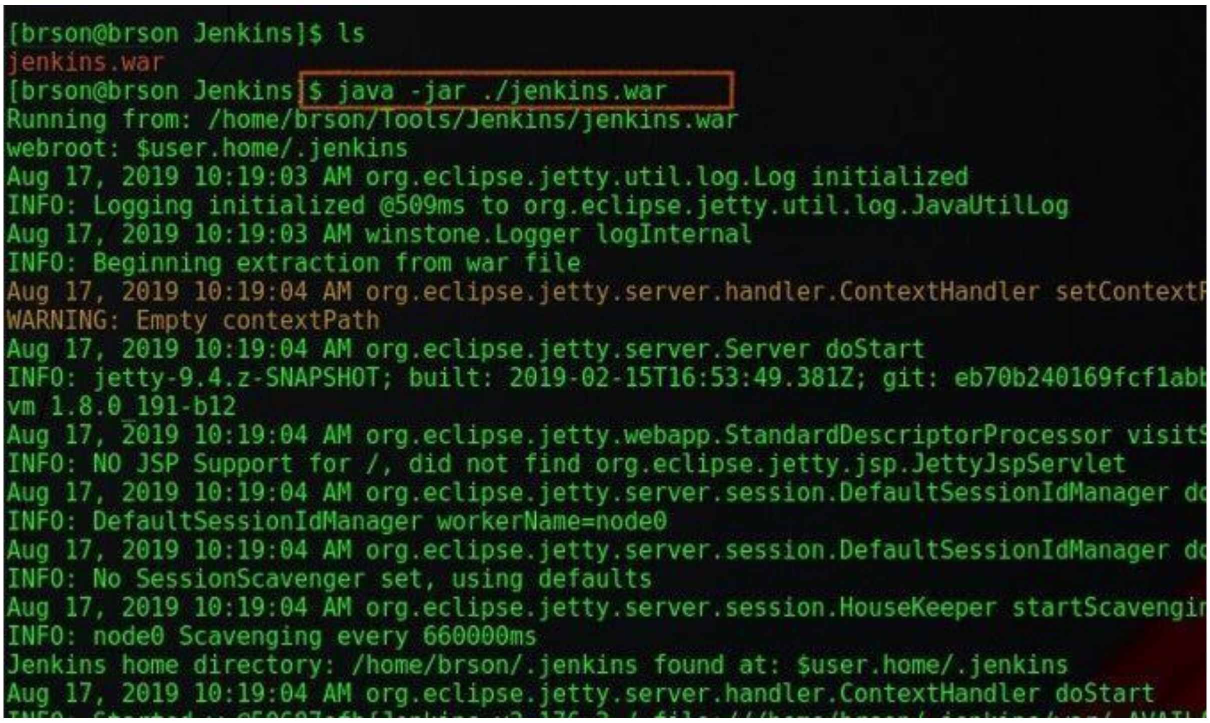
Once a Jenkins package has been downloaded, proceed to the Installing Jenkins section of the User Handbook.

Step 2: Execute Jenkins as a Java binary

Open a terminal window and enter the directory where you downloaded Jenkins with **cd <your path>**. (Before you proceed, make sure JDK is installed and added to the environment path.) Execute the following command, which will run the WAR file as an executable binary:

```
java -jar ./Jenkins. War
```

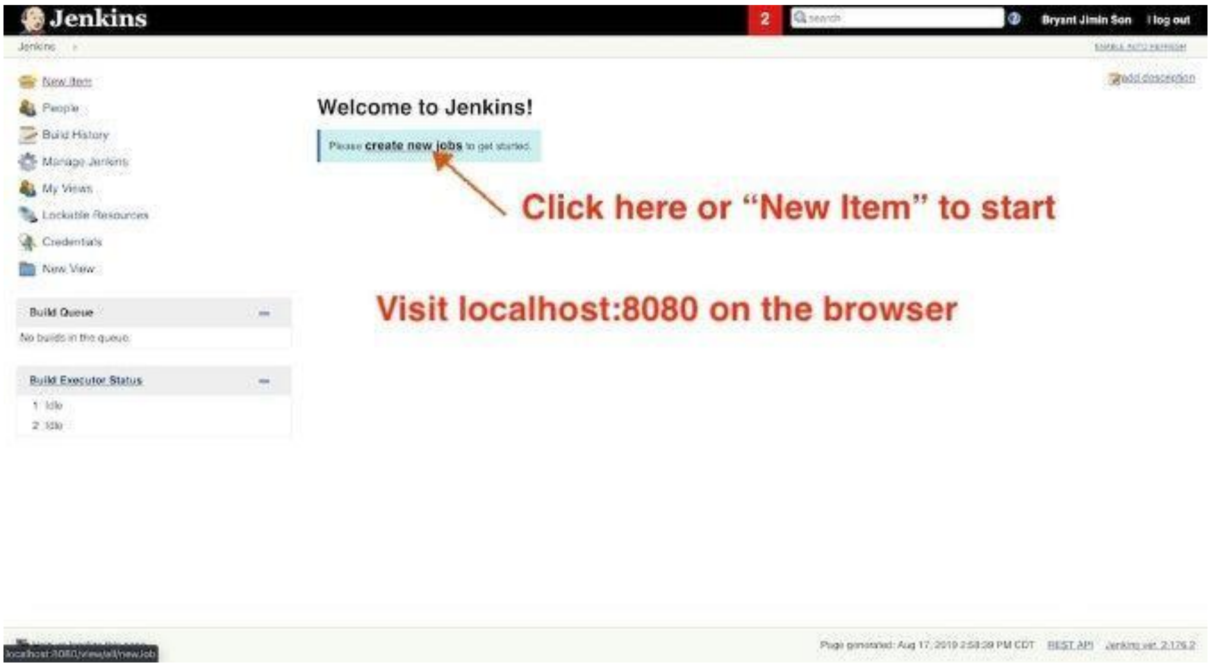
If everything goes smoothly, Jenkins should be up and running at the default port 8080.



```
[brson@brson Jenkins]$ ls
jenkins.war
[brson@brson Jenkins]$ java -jar ./jenkins.war
Running from: /home/brson/Tools/Jenkins/jenkins.war
webroot: $user.home/.jenkins
Aug 17, 2019 10:19:03 AM org.eclipse.jetty.util.log.Log initialized
INFO: Logging initialized @509ms to org.eclipse.jetty.util.log.JavaUtilLog
Aug 17, 2019 10:19:03 AM winstone.Logger logInternal
INFO: Beginning extraction from war file
Aug 17, 2019 10:19:04 AM org.eclipse.jetty.server.handler.ContextHandler setContextP
WARNING: Empty contextPath
Aug 17, 2019 10:19:04 AM org.eclipse.jetty.server.Server doStart
INFO: jetty-9.4.z-SNAPSHOT; built: 2019-02-15T16:53:49.381Z; git: eb70b240169fcf1abb
vm 1.8.0_191-b12
Aug 17, 2019 10:19:04 AM org.eclipse.jetty.webapp.StandardDescriptorProcessor visitS
INFO: NO JSP Support for /, did not find org.eclipse.jetty.jsp.JettyJspServlet
Aug 17, 2019 10:19:04 AM org.eclipse.jetty.server.session.DefaultSessionIdManager do
INFO: DefaultSessionIdManager workerName=node0
Aug 17, 2019 10:19:04 AM org.eclipse.jetty.server.session.DefaultSessionIdManager do
INFO: No SessionScavenger set, using defaults
Aug 17, 2019 10:19:04 AM org.eclipse.jetty.server.session.HouseKeeper startScavengin
INFO: node0 Scavenging every 660000ms
Jenkins home directory: /home/brson/.jenkins found at: $user.home/.jenkins
Aug 17, 2019 10:19:04 AM org.eclipse.jetty.server.handler.ContextHandler doStart
```

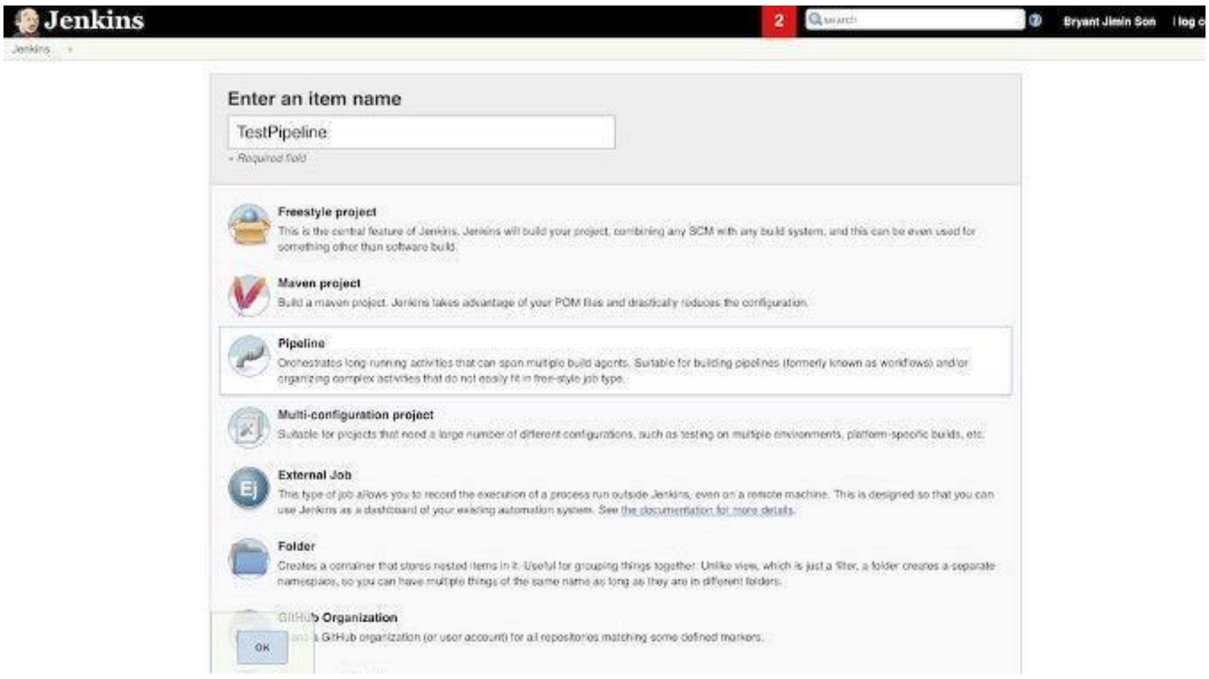
Step 3: Create a new Jenkins job

Open a web browser and navigate to **localhost:8080**. Unless you have a previous Jenkins installation, it should go straight to the Jenkins dashboard. Click **Create New Jobs**. You can also click **New Item** on the left.



Step 4: Create a pipeline job

In this step, you can select and define what type of Jenkins job you want to create. Select **Pipeline** and give it a name (e.g., Test Pipeline). Click **OK** to create a pipeline job.



Enter an item name

TestPipeline

» Required field

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

External Job
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system. See the documentation for more details.

Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

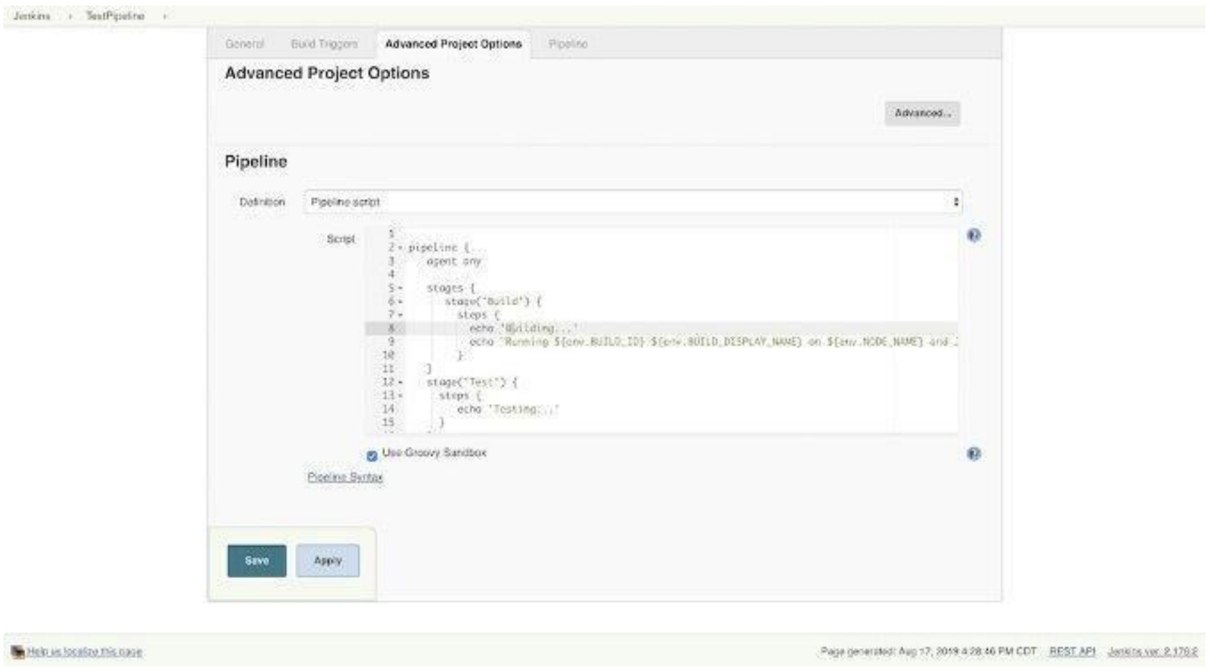
GitHub Organization
[illegible] a GitHub organization (or user account) for all repositories matching some defined markers.

You will see a Jenkins job configuration page. Scroll down to find **Pipeline section**. There are two ways to execute a Jenkins pipeline. One way is by *directly writing a pipeline script* on Jenkins, and the other way is by retrieving the *Jenkins file from SCM* (source control management). We will go through both ways in the next two steps.

Step 5: Configure and execute a pipeline job through a direct script

To execute the pipeline with a direct script, begin by copying the contents of the sample Jenkinsfile from GitHub. Choose **Pipeline script** as the **Destination** and paste the **Jenkins file** contents in **Script**. Spend a little time studying how the Jenkins file is structured. Notice that there are three Stages: Build, Test, and Deploy, which are arbitrary and can be anything. Inside each Stage, there are Steps; in this example, they just print some random messages.

Click **Save** to keep the changes, and it should automatically take you back to the Job Overview.

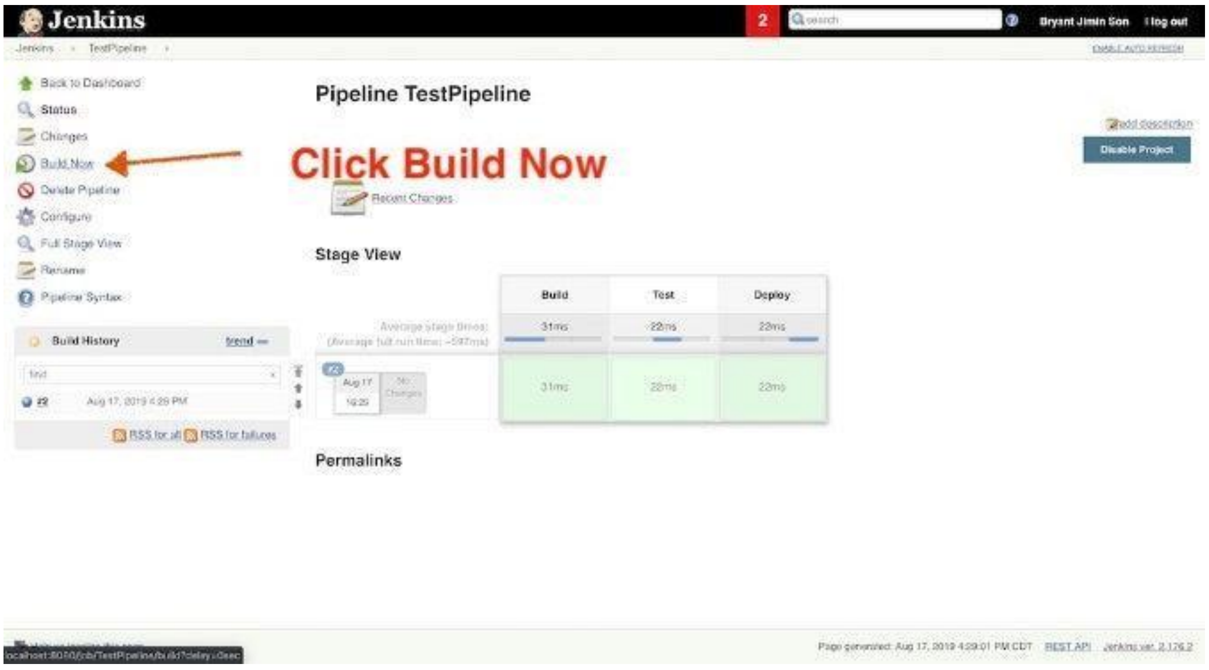


```
pipeline {
  agent any

  stages {
     stage('Build') {
       steps {
         echo 'Building...'
         echo 'Running ${env.BUILD_ID} ${env.BUILD_DISPLAY_NAME} on ${env.NODE_NAME} and 
       }
  }
  stage('Test') {
    steps {
       echo 'Testing...'
    }
```

Use Groovy Sandbox [✓]

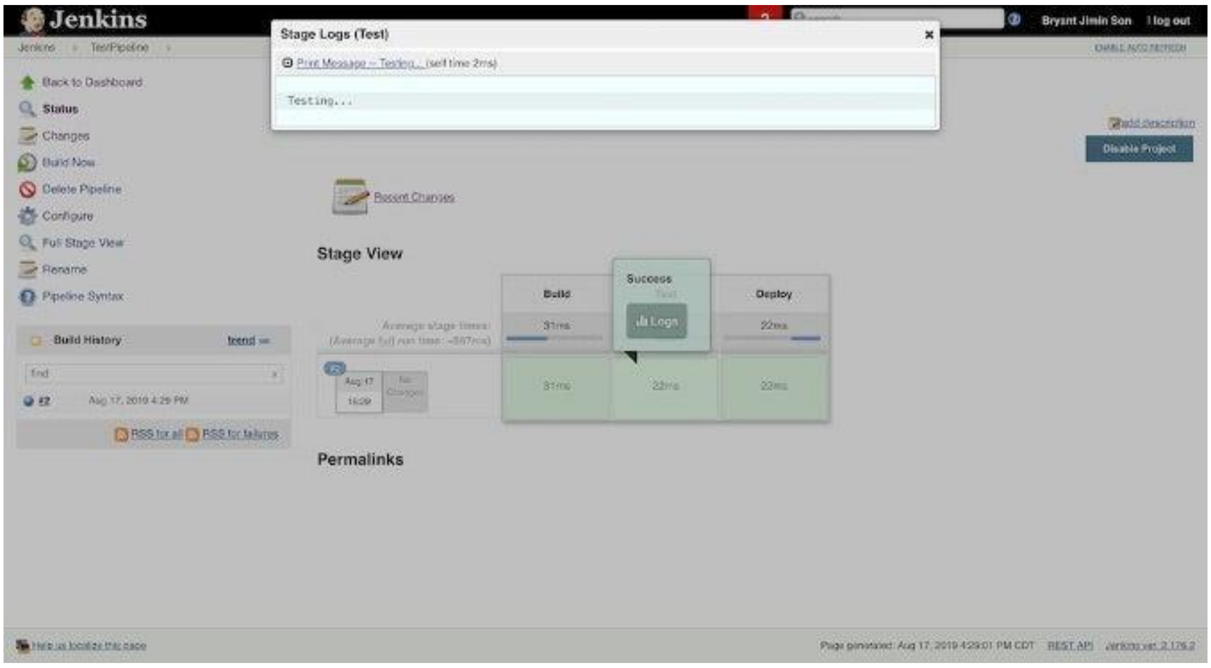
Pipeline Syntax

To start the process to build the pipeline, click **Build Now**. If everything works, you will see your first pipeline (like the one below).



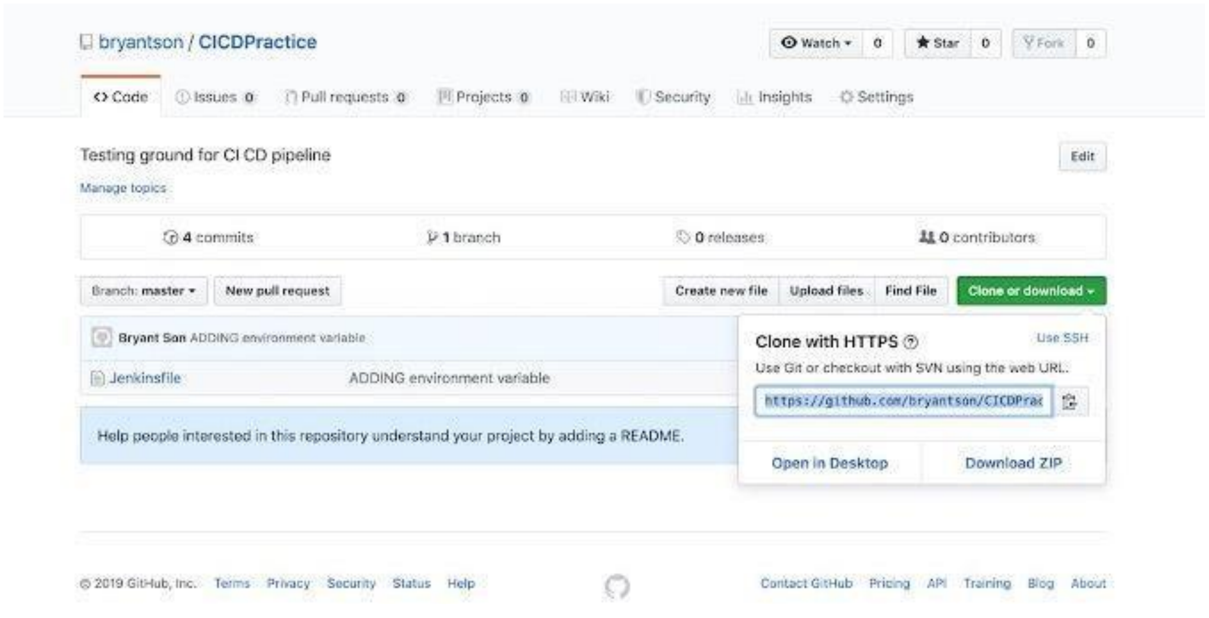
	Build	Test	Deploy
Average stage times: (Average full run time: ~597ms)	31ms	22ms	22ms
#2 Aug 17 16:29 No Changes	31ms	22ms	22ms

To see the output from the pipeline script build, click any of the Stages and click **Log**. You will see a message like this.

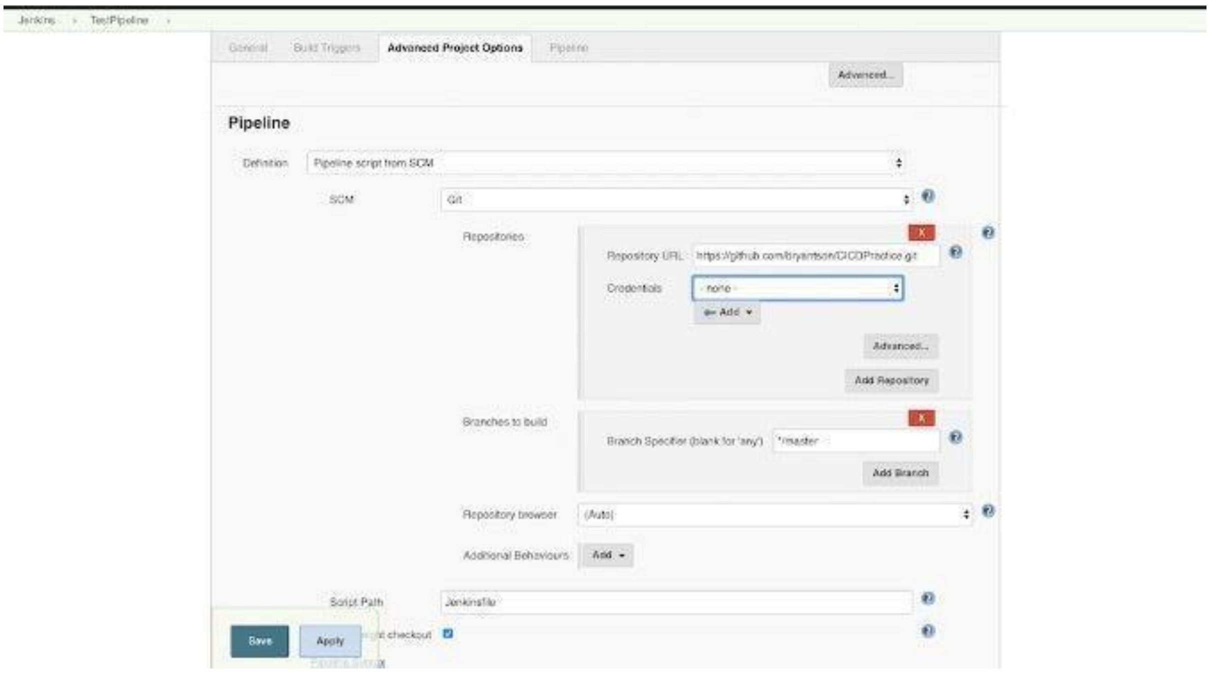


Step 6: Configure and execute a pipeline job with SCM

Now, switch gears: In this step, you will Deploy the same Jenkins job by copying the **Jenkins file** from a source-controlled GitHub. In the same GitHub repository, pick up the repository URL by clicking **Clone or download** and copying its URL.


bryantson / CICDPractice
Testing ground for CI CD pipeline
Jenkinsfile
ADDING environment variable
Clone or download
Clone with HTTPS
Use Git or checkout with SVN using the web URL.
https://github.com/bryantson/CICDPrac
Open in Desktop
Download ZIP

Click **Configure** to modify the existing job. Scroll to the **Advanced Project Options** setting, but this time, select the **Pipeline script from SCM** option in the **Destination** dropdown. Paste the GitHub repo's URL in the **Repository URL**, and type **Jenkins file** in the **Script Path**. Save by clicking the **Save** button.


Jenkins › TestPipeline ›

General | Build Triggers | Advanced Project Options | Pipeline

Pipeline

Definition: Pipeline script from SCM

SCM: Git

Repositories
Repository URL: https://github.com/bryantson/CICDPractice.git
Credentials: - none -

Branches to build
Branch Specifier (blank for 'any'): */master

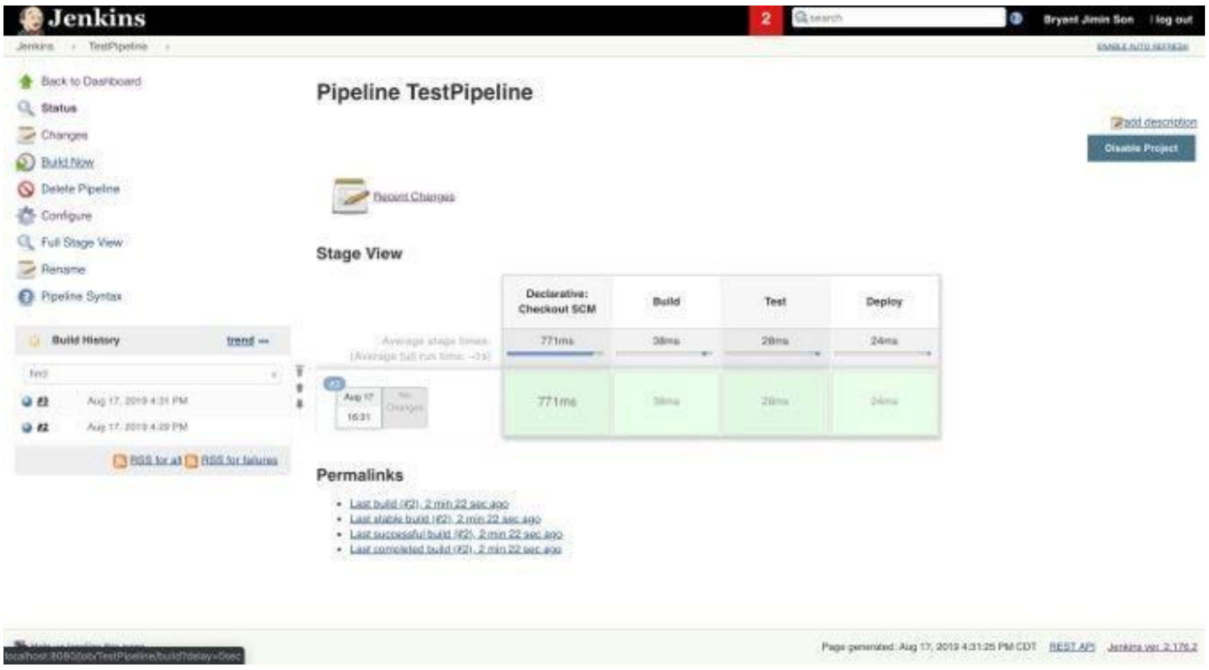
Repository browser: (Auto)

Additional Behaviours: Add

Script Path: Jenkinsfile

Lightweight checkout: ✓

To build the pipeline, once you are back to the Task Overview page, click **Build Now** to execute the job again. The result will be the same as before, except you have one additional stage called **Declaration: Checkout SCM**.


Jenkins › TestPipeline ›

Pipeline TestPipeline

Stage View

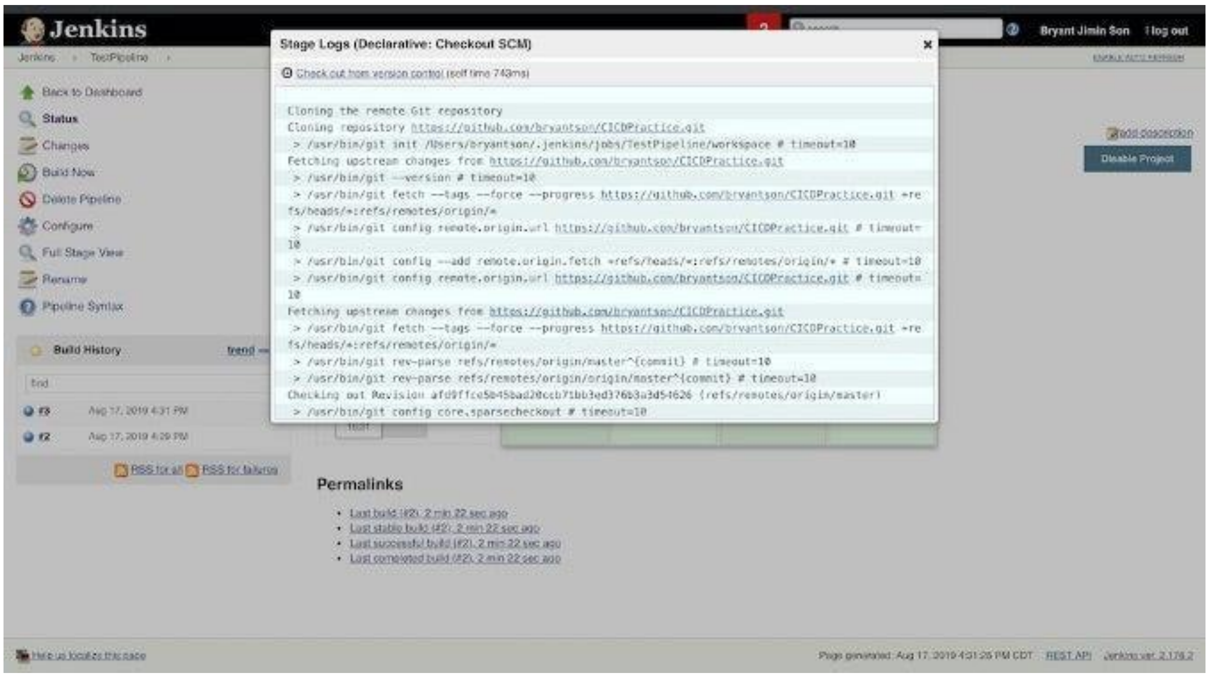
	Declarative: Checkout SCM	Build	Test	Deploy
Average stage times: (Average full run time: ~1s)	771ms	38ms	28ms	24ms
#2 Aug 17 16:31 No Changes	771ms	38ms	28ms	24ms

Build History
#2 Aug 17, 2019 4:31 PM
#2 Aug 17, 2019 4:29 PM

Permalinks
- Last build (#2), 2 min 22 sec ago
- Last stable build (#2), 2 min 22 sec ago
- Last successful build (#2), 2 min 22 sec ago
- Last completed build (#2), 2 min 22 sec ago

Page generated: Aug 17, 2019 4:31:25 PM CDT REST API Jenkins ver. 2.176.2

To see the pipeline's output from the SCM build, click the Stage and view the **Log** to check how the source control cloning process went.

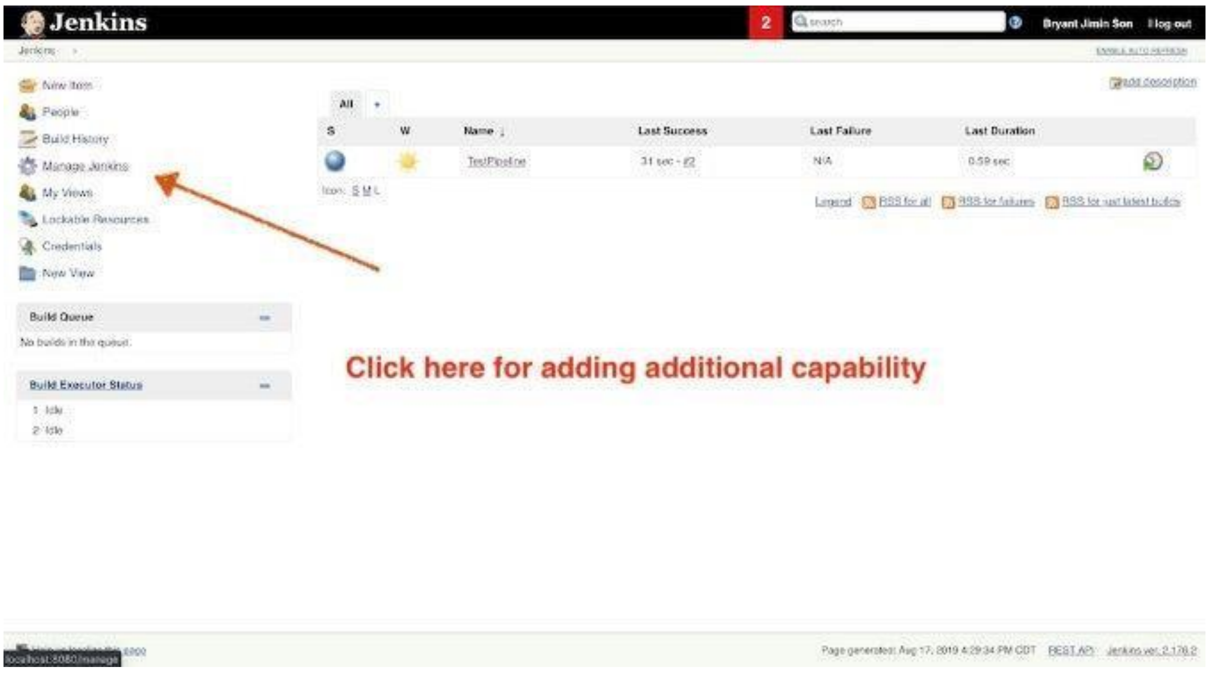

Stage Logs (Declarative: Checkout SCM)

Check out from version control (self time 743ms)

```
Cloning the remote Git repository
Cloning repository https://github.com/bryantson/CICDPractice.git
 > /usr/bin/git init /Users/bryantson/.jenkins/jobs/TestPipeline/workspace # timeout=10
Fetching upstream changes from https://github.com/bryantson/CICDPractice.git
 > /usr/bin/git --version # timeout=10
 > /usr/bin/git fetch --tags --force --progress https://github.com/bryantson/CICDPractice.git +re
fs/heads/*:refs/remotes/origin/*
 > /usr/bin/git config remote.origin.url https://github.com/bryantson/CICDPractice.git # timeout=
10
 > /usr/bin/git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
 > /usr/bin/git config remote.origin.url https://github.com/bryantson/CICDPractice.git # timeout=
10
Fetching upstream changes from https://github.com/bryantson/CICDPractice.git
 > /usr/bin/git fetch --tags --force --progress https://github.com/bryantson/CICDPractice.git +re
fs/heads/*:refs/remotes/origin/*
 > /usr/bin/git rev-parse refs/remotes/origin/master^{commit} # timeout=10
 > /usr/bin/git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision afd9ffce5b45bad20ccb71bb3ed376b3a3d54626 (refs/remotes/origin/master)
 > /usr/bin/git config core.sparsecheckout # timeout=10
```

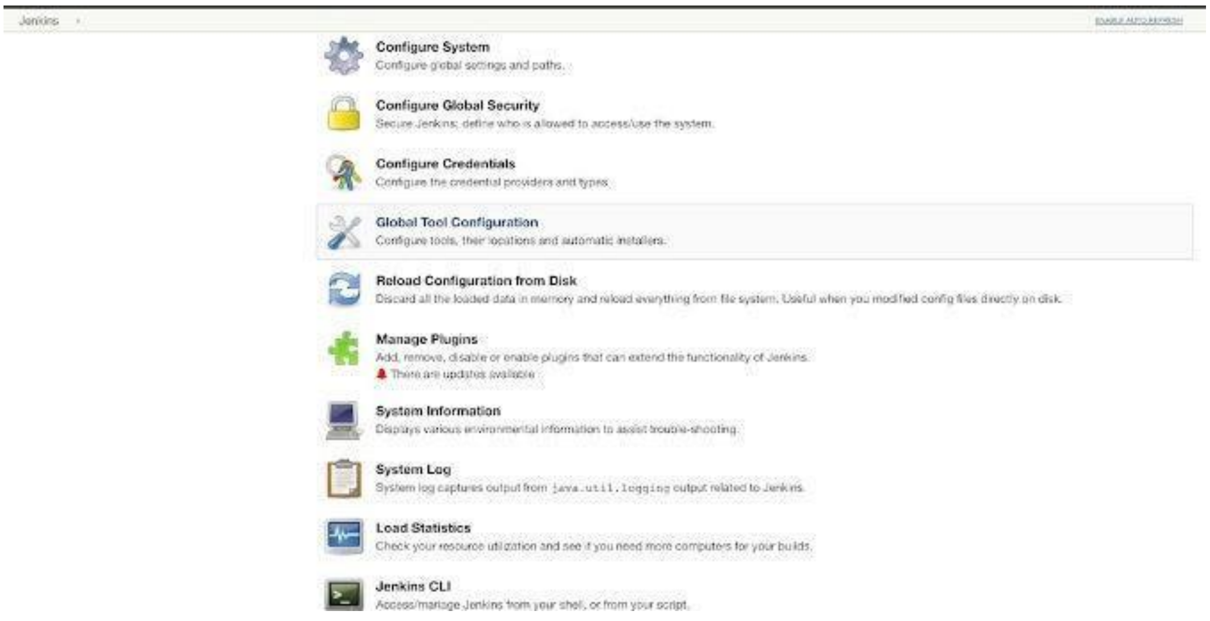
Manage Jenkins

From the main Jenkins dashboard, click **Manage Jenkins**.



Global tool configuration

There are many options available, including managing plugins, viewing the system log, etc. Click **Global Tool Configuration**.


Jenkins

ENABLE AUTO REFRESH

Configure System
Configure global settings and paths.

Configure Global Security
Secure Jenkins; define who is allowed to access/use the system.

Configure Credentials
Configure the credential providers and types

Global Tool Configuration
Configure tools, their locations and automatic installers.

Reload Configuration from Disk
Discard all the loaded data in memory and reload everything from file system. Useful when you modified config files directly on disk.

Manage Plugins
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.
There are updates available

System Information
Displays various environmental information to assist trouble-shooting.

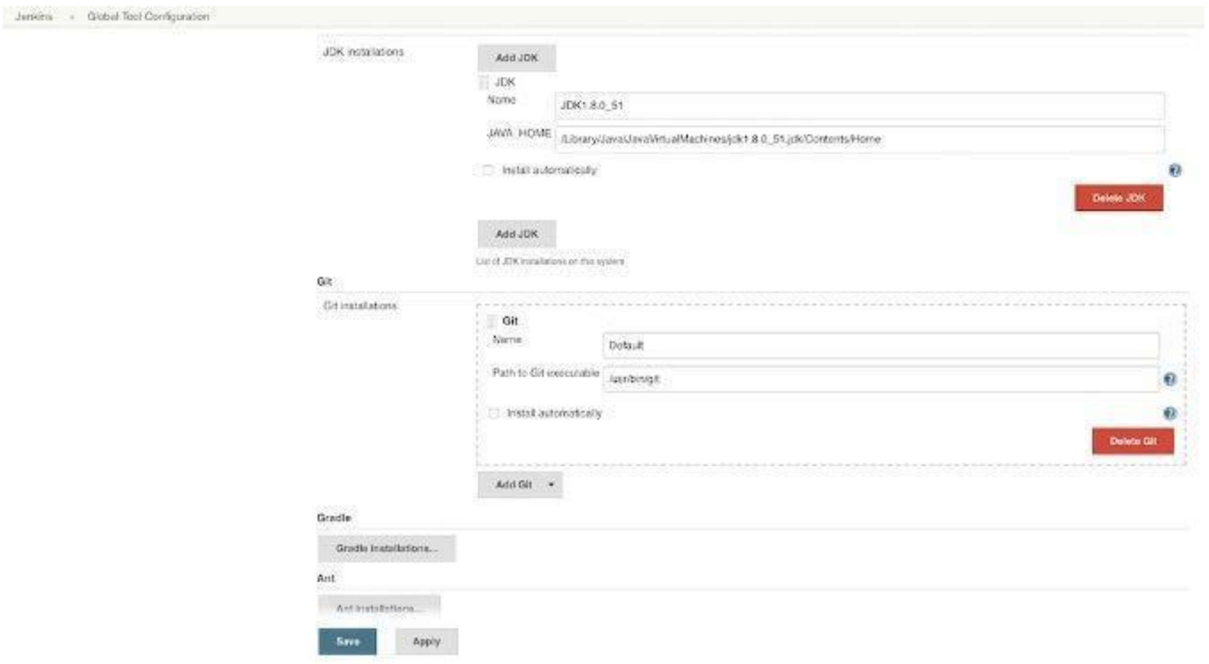
System Log
System log captures output from `java.util.logging` output related to Jenkins.

Load Statistics
Check your resource utilization and see if you need more computers for your builds.

Jenkins CLI
Access/manage Jenkins from your shell, or from your script.

Add additional capabilities

Here, you can add the JDK path, Git, Gradle, and so much more. After you configure a tool, it is just a matter of adding the command into your Jenkinsfile or executing it through your Jenkins script.


Jenkins » Global Tool Configuration

JDK installations

Add JDK

JDK

Name: JDK1.8.0_51

JAVA_HOME: /Library/Java/JavaVirtualMachines/jdk1.8.0_51.jdk/Contents/Home

☐ Install automatically

Delete JDK

Add JDK

List of JDK installations on this system

Git

Git installations

Git

Name: Default

Path to Git executable: /usr/bin/git

☐ Install automatically

Delete Git

Add Git

Gradle

Gradle installations...

Ant

Ant installations...

Save Apply

Exercise 10:

Module name: Implementation of CICD with Java and open-source stack
In the configured Jenkins pipeline created in Exercise 8 and 9, implement quality gates for static analysis of code.

Enforce quality gates using SonarQube and Jenkins

Code analysis in the agile product development cycle is one of the important and necessary items to avoid possible failures and defects arising out of the continuous changes in the source codes. There are few good reasons to include this in our development lifecycle.

- It can help to find vulnerabilities in the distant corners of your application, which are not even used, then also static analysis has a higher probability of finding those vulnerabilities.
- You can define your project specific rules, and they will be ensured to follow without any manual intervention.
- It can help to find the bug early in the development cycle, which means less cost to fix them.

Quality Gates can be defined as a set of threshold measures set on your project like Code Coverage, Technical Debt Measure, Number of Blocker/Critical issues, Security Rating/ Unit Test Pass Rate and more.

Enforce Quality Gates

Failing your build jobs when the code doesn't meet criteria set in Quality Gates should be the way to go. We were using Jenkins as our CI tool and therefore we wanted to setup Jenkins job to fail if the code doesn't meet quality gates.

In this, we are going to setup following

1. Quality gate metrics setup in SonarQube.
2. Configure Jenkins job to fail the build when not meeting Quality Gates.

Jenkins job setup

Prerequisites

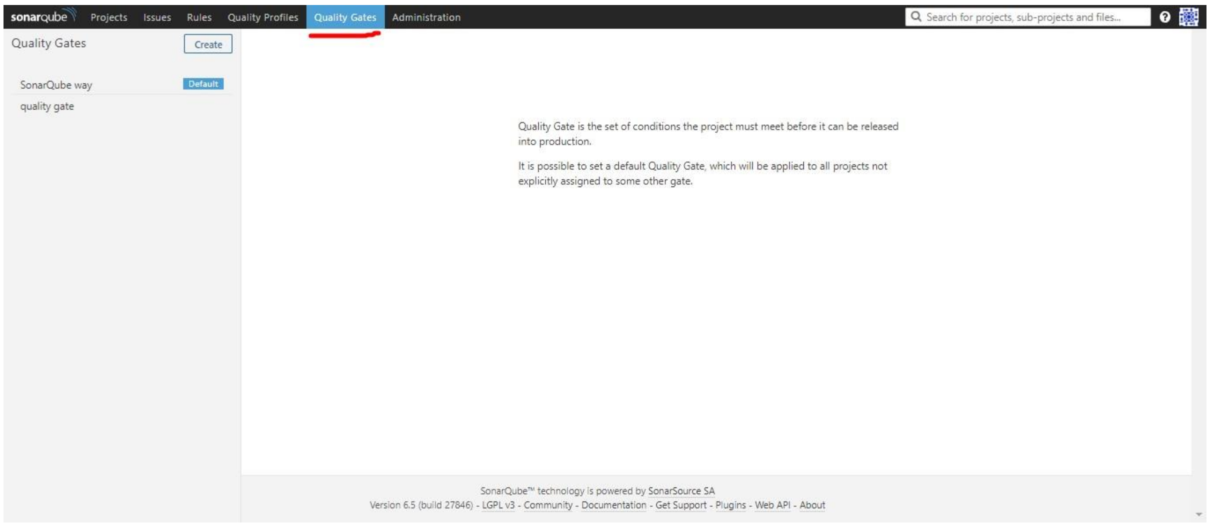
- Install Jenkins plugin "sonar-quality-gates-plugin" if not already present.
- Email-ext. plugin for Jenkins to be able to send emails.
- Jenkins project configured i.e., successfully passing the build already.

Here is the snapshot of the job that currently passing build before Quality Gates setup.

```
[DEBUG] 07:05:25.194 Execution stop
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO]
[INFO] miqp ............................................... SUCCESS [ 12.475 s]
[INFO] Miqp Ui ............................................ SUCCESS [ 32.142 s]
[INFO] miqp-server ........................................ SUCCESS [  3.910 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 49.951 s
[INFO] Finished at: 2018-05-30T07:05:25+00:00
[INFO] Final Memory: 62M/612M
[INFO] ------------------------------------------------------------------------
```

Let's setup Quality gate metrics in the SonarQube server. We are going to create quality gate only for the metrics "Code coverage" for demo purpose. But there are more metrics available that you should be selecting while creating quality gates.

Login to SonarQube as admin → go to Quality Gates



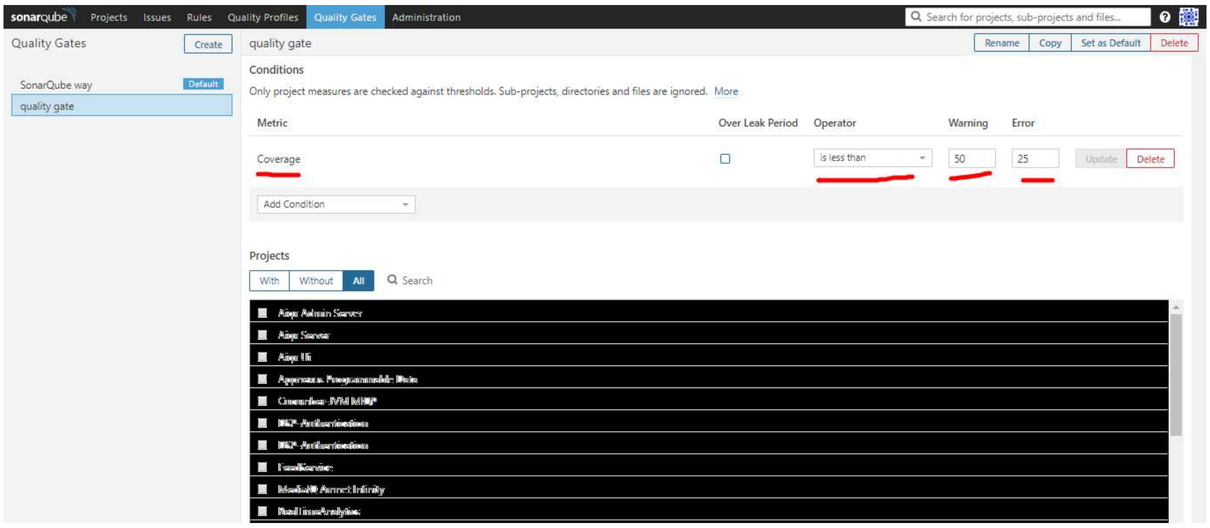
Quality Gate is the set of conditions the project must meet before it can be released into production.

It is possible to set a default Quality Gate, which will be applied to all projects not explicitly assigned to some other gate.

SonarQube™ technology is powered by SonarSource SA
Version 6.5 (build 27846) - LGPL v3 - Community - Documentation - Get Support - Plugins - Web API - About

Click on create -> Add Condition -> Choose metrics

(In this example, we selected **Code Coverage**) -> select operator along with warning and error threshold.



quality gate

Conditions

Only project measures are checked against thresholds. Sub-projects, directories and files are ignored. More

Metric	Over Leak Period	Operator	Warning	Error	
Coverage	☐	is less than	50	25	Update Delete

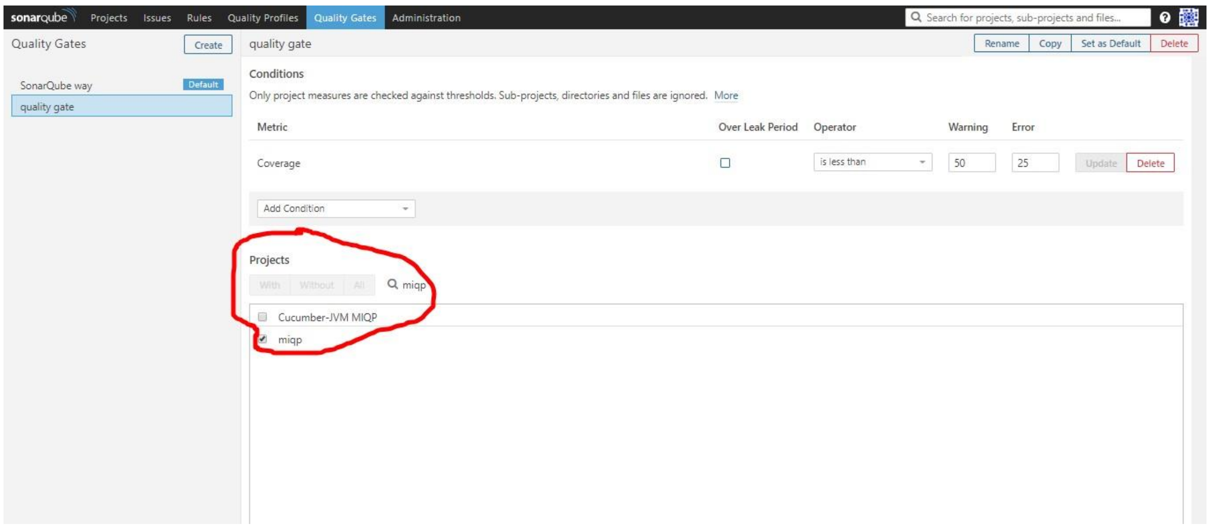
Add Condition

Projects

With | Without | All | Search

- ☐ Aiops Admin Server
- ☐ Aiops Server
- ☐ Aiops Ui
- ☐ Apparatus Programmable Data
- ☐ Cucumber-JVM MIQP
- ☐ DSP-Authentication
- ☐ DSP-Authentication
- ☐ FeedService
- ☐ MediaIQ Annet Infinity
- ☐ RealTimeAnalytics

Select the project from the available list to which you want to associate this quality gate. We have selected sample map project for which we have set up Jenkins job.



quality gate

Conditions

Only project measures are checked against thresholds. Sub-projects, directories and files are ignored. More

Metric	Over Leak Period	Operator	Warning	Error	
Coverage	☐	is less than	50	25	Update Delete

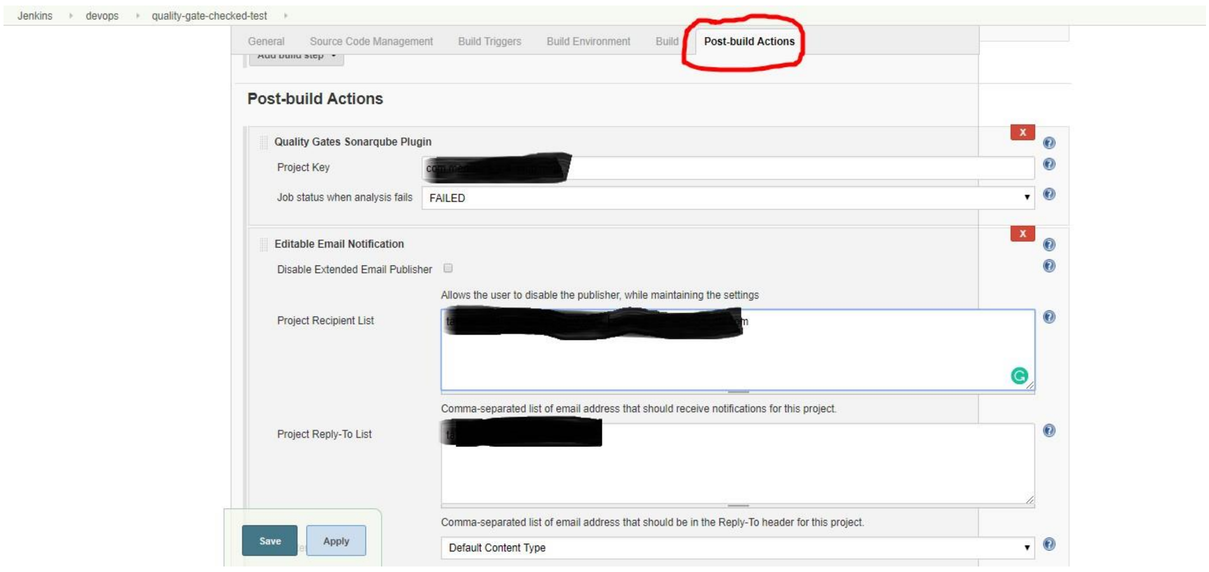
Add Condition

Projects

With | Without | All | miqp

- ☐ Cucumber-JVM MIQP
- ☑ miqp

Now go to the Jenkins job and configure the quality gate validation. Click on the job and go to **Post-build Actions** and provide the project details you have associated with Quality Gate created in the earlier steps.



Post-build Actions

Quality Gates Sonarqube Plugin

Project Key [redacted]

Job status when analysis fails: FAILED

Editable Email Notification

Disable Extended Email Publisher ☐

Allows the user to disable the publisher, while maintaining the settings

Project Recipient List [redacted]

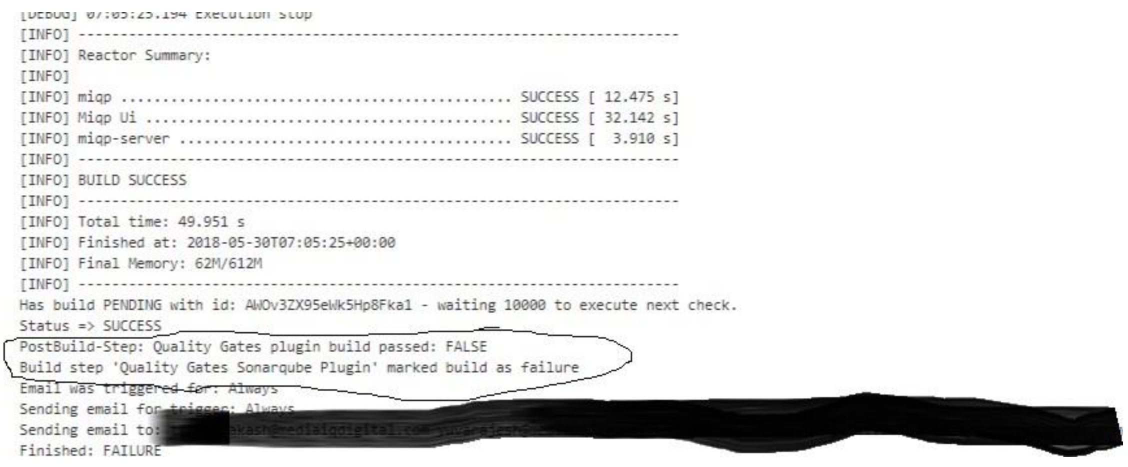
Comma-separated list of email address that should receive notifications for this project.

Project Reply-To List [redacted]

Comma-separated list of email address that should be in the Reply-To header for this project.

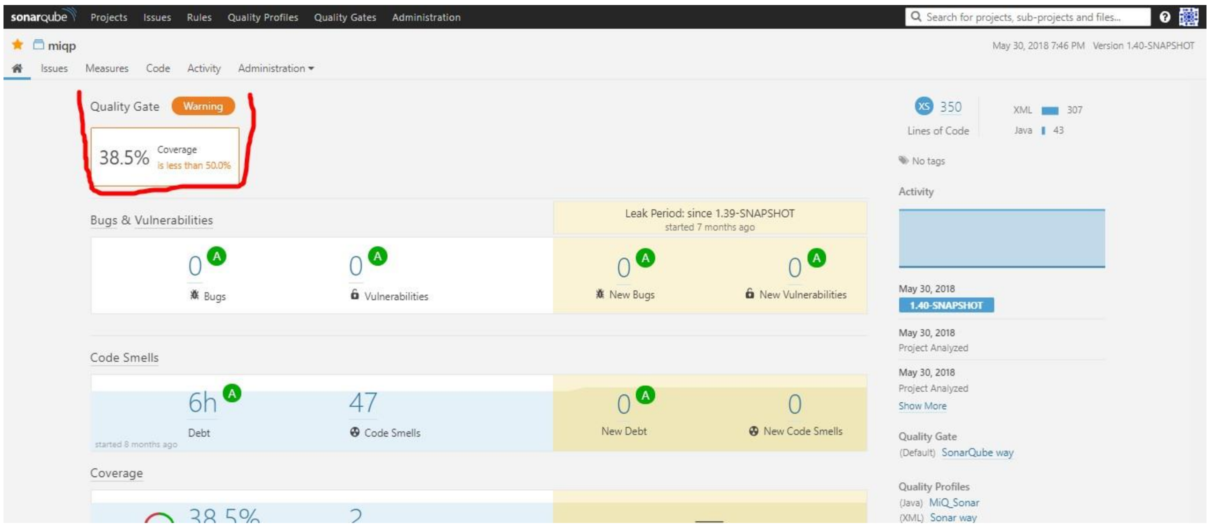
Default Content Type

Run the Jenkins job again and verify the build status post quality check enabled.



```
[DEBUG] 07:05:25.194 Execution stop
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO]
[INFO] miqp ............................................... SUCCESS [ 12.475 s]
[INFO] Miqp Ui ............................................ SUCCESS [ 32.142 s]
[INFO] miqp-server ........................................ SUCCESS [  3.910 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 49.951 s
[INFO] Finished at: 2018-05-30T07:05:25+00:00
[INFO] Final Memory: 62M/612M
[INFO] ------------------------------------------------------------------------
Has build PENDING with id: AWOv3ZX95eWk5Hp8Fka1 - waiting 10000 to execute next check.
Status => SUCCESS
PostBuild-Step: Quality Gates plugin build passed: FALSE
Build step 'Quality Gates Sonarqube Plugin' marked build as failure
Email was triggered for: Always
Sending email for trigger: Always
Sending email to: [redacted]
Finished: FAILURE
```

As we could see that code passed the build, however, it doesn't pass quality gate check. Therefore, build fails in the end. We can verify the same with the project status in SonarQube server.



Conclusion

In this, we have shown how can we enforce quality gates on code base using SonarQube and Jenkins.

Exercise 11:

Module name: Implementation of CICD with Java and open-source stack
In the configured Jenkins pipeline created in Exercise 8 and 9, implement quality gates for static unit testing.
Configuring Jenkins to Run a Build Automatically on Code Push

In the last article we configured a build on Jenkins to run unit tests and lint checks. We now focus on triggering a build in Jenkins when you push code to a GitHub repository.

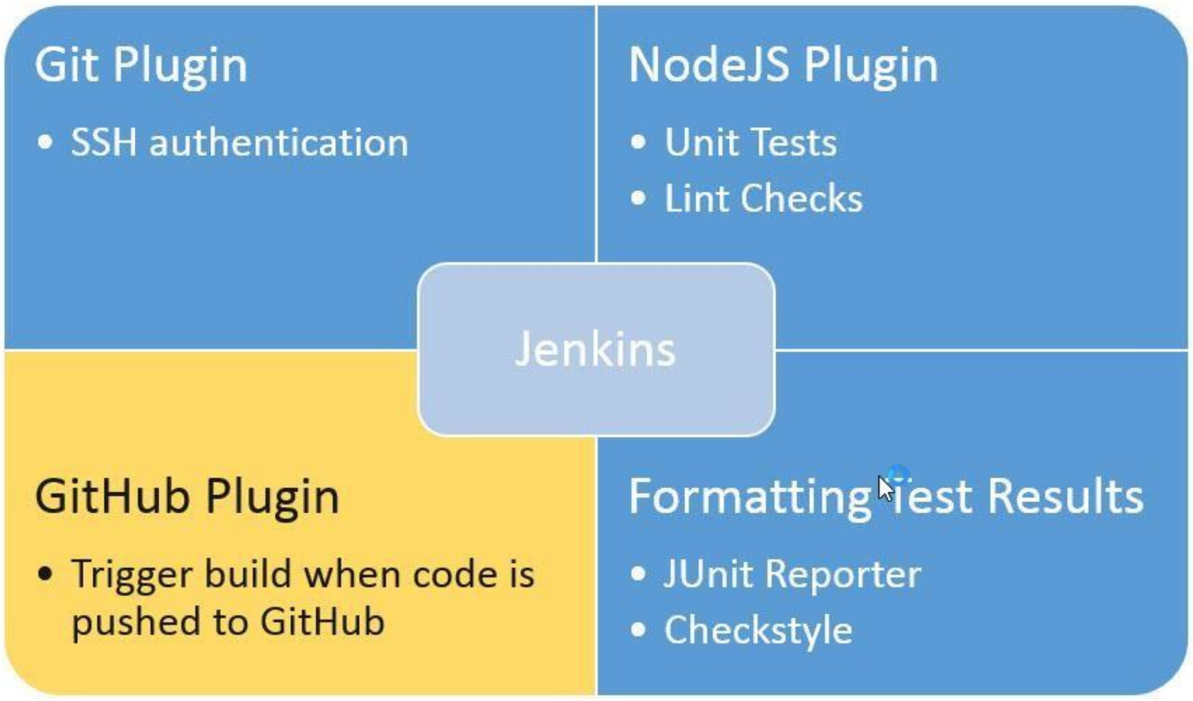


Figure 1: Continuous Integration setup with Jenkins

Running builds as soon as code is pushed to the central repository is important, as it enables you to detect problems quickly and locate them more easily.

To know how to configure a Jenkins build for a project, please refer to Setting Up Jenkins with GitHub.

Before you can start with the process given below, you will need to install the ***GitHub plugin*** *on your Jenkins server.*

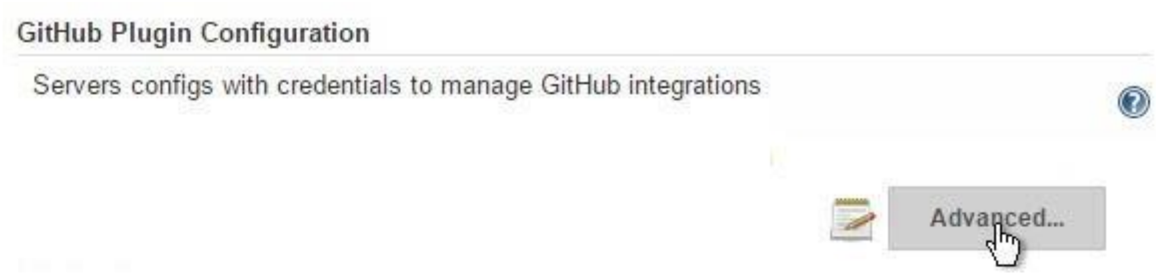
1. Configuring Jenkins

Although we have configured Jenkins to communicate with our repository on GitHub, we still have to manually start the build from Jenkins. To automatically run builds, Jenkins listens for POST requests at a Hook URL. We need to give this URL to the repository on GitHub. Then, whenever code is pushed to that repository, GitHub will send a POST request to the Hook URL and Jenkins will run the build.

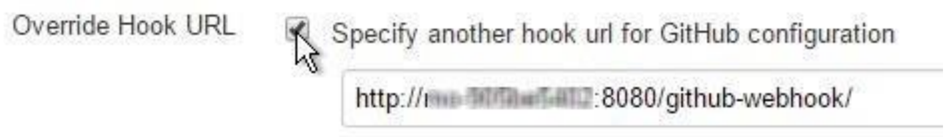
To get the Hook URL of Jenkins, Open the Jenkins Dashboard.

Go to: Manage Jenkins > Configure System

Under GitHub Plugin Configuration, Click on 'Advanced…'


GitHub Plugin Configuration
Servers configs with credentials to manage GitHub integrations
Advanced...

Check 'Specify another hook url for GitHub configuration'


Override Hook URL

Specify another hook url for GitHub configuration

http://[redacted]:8080/github-webhook/

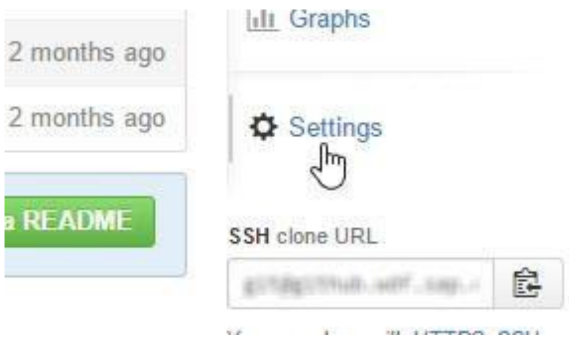
A textbox will appear with a hook URL. This is the Hook URL at which Jenkins will listen for POST requests. Copy this URL and go to the next step.

2. Configuring GitHub Repository

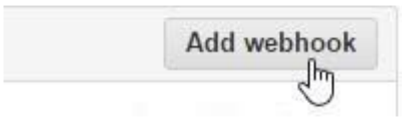
We now have to provide the Hook URL we got from Jenkins in the previous step.

Open your repository on GitHub.

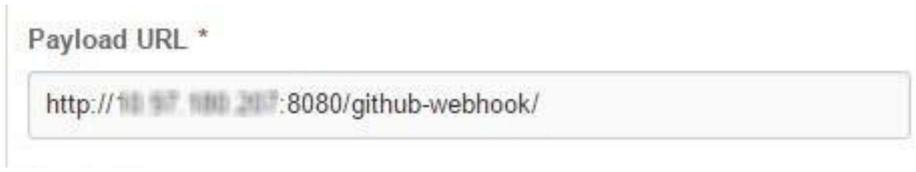
Click 'Settings' on the navigation bar on the right-hand side of the screen.



Click ‘Webhooks & services’ on the navigation bar on the left-hand side of the screen.

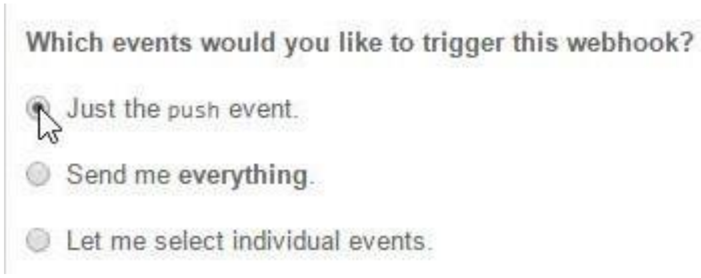


Paste the URL you copied in the previous step as the ‘Payload URL’.


Payload URL *

http://[redacted]:8080/github-webhook/

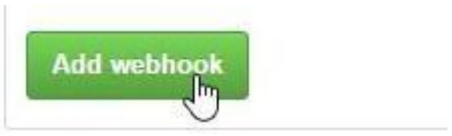
You can select the events for which you want the Jenkins build to be triggered. We will select ‘Just the push event’ because we want to run the build when we push our code to the repository.


Which events would you like to trigger this webhook?

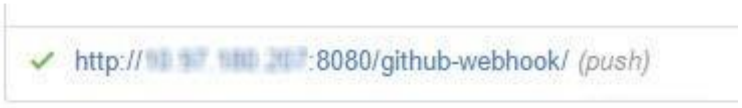
- ● Just the push event.
- ○ Send me everything.
- ○ Let me select individual events.

Alternatively, you can click on ‘Let me select individual events’ to get a list of all the events that you can select to trigger your Jenkins build.

Click ‘Add webhook’ to add the webhook.



You should now see the webhook you just added in the list of Webhooks for that repository like this.

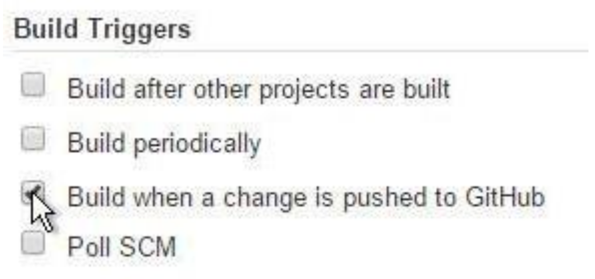

✓ http://[redacted]:8080/github-webhook/ *(push)*

<u>3. Configuring Jenkins Project</u>

We now have Jenkins configured to run builds automatically when code is pushed to central repositories. However, Jenkins doesn't run all builds for all projects. To specify which project builds, need to run, we have to modify the project configuration.

In Jenkins, go to the project configuration of the project for which you want to run an automated build.

In the 'Build Triggers' section, select 'Build when a change is pushed to GitHub'.



Build Triggers

- [ ] Build after other projects are built
- [ ] Build periodically
- [x] Build when a change is pushed to GitHub
- [ ] Poll SCM

Save your project.

Jenkins will now run the build when you push your code to the GitHub repository.

Conclusion

you will now be able to:

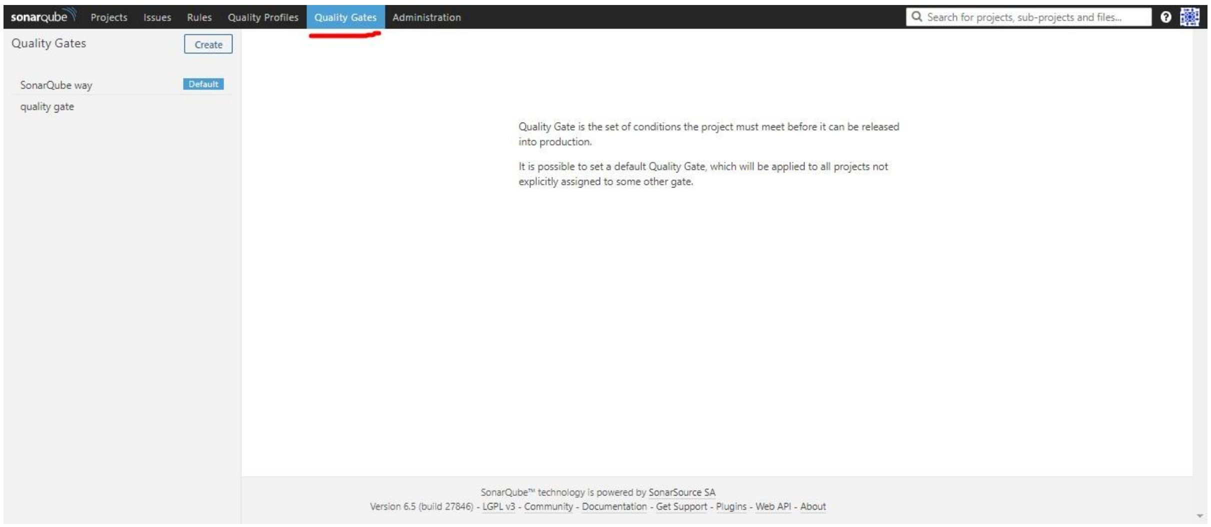
- Configure a continuous integration server (running Jenkins) to communicate with a repository on GitHub (Setting Up Jenkins with GitHub).
- Configure the continuous integration server to run unit tests and lint checks as a part of the build (Configuring Jenkins for Unit tests and Lint checks).
- Configure the continuous integration server to automatically run your build when code is pushed to your repository on GitHub (this document).

Exercise 12:

Module name: Course end assessment
In the configured Jenkins pipeline created in Exercise 8 and 9, implement quality gates for code coverage.

Jenkins job setup

Prerequisites

- Install Jenkins plugin “sonar-quality-gates-plugin” if not already present.
- Email-ext plugin for Jenkins to be able to send emails.
- Jenkins project configured i.e., successfully passing the build already.
- Let’s setup Quality gate metrics in the SonarQube server. We are going to create quality gate only for the metrics “Code coverage” for demo purpose. But there are more metrics available that you should be selecting while creating quality gates.
    - *Login to SonarQube as admin → go to Quality Gates*
- 

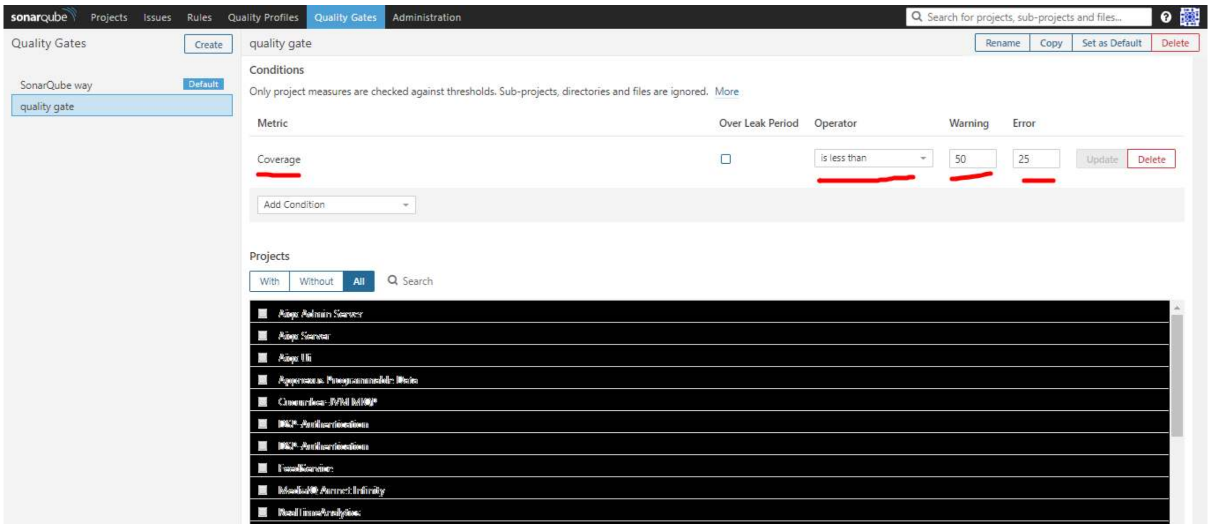
Quality Gate is the set of conditions the project must meet before it can be released into production.

It is possible to set a default Quality Gate, which will be applied to all projects not explicitly assigned to some other gate.

SonarQube™ technology is powered by SonarSource SA
Version 6.5 (build 27846) - LGPL v3 - Community - Documentation - Get Support - Plugins - Web API - About

  Click on create -> Add Condition -> Choose metrics

- (In this example, we selected **Code Coverage**) -> select operator along with warning and error threshold.



quality gate

Conditions

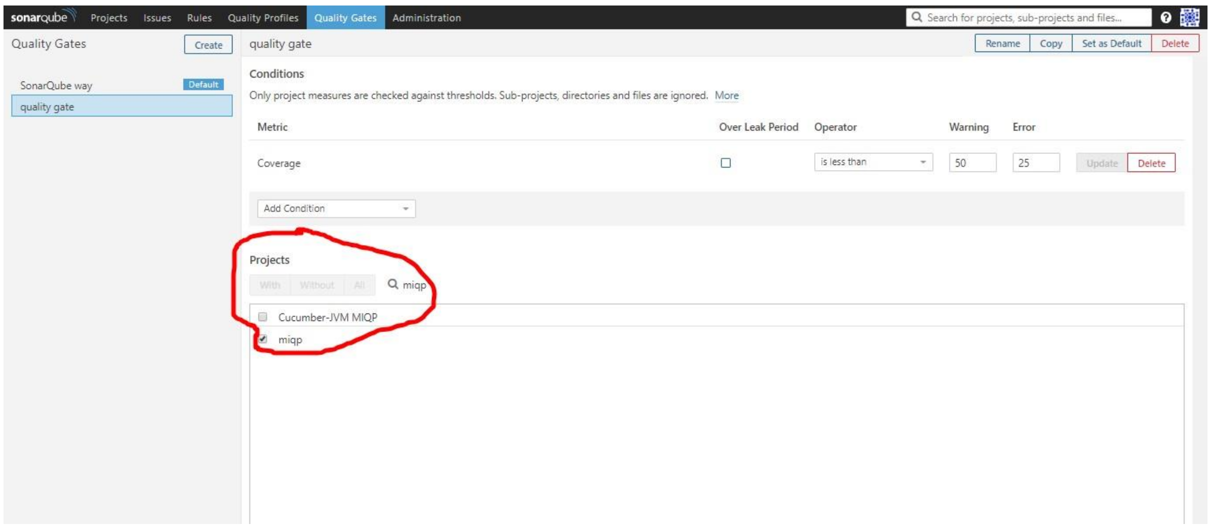
Only project measures are checked against thresholds. Sub-projects, directories and files are ignored. More

Metric	Over Leak Period	Operator	Warning	Error
Coverage	☐	is less than	50	25

Projects

-
-

Select the project from the available list to which you want to associate this quality gate. We have selected sample miqp project for which we have set up Jenkins job.



quality gate

Conditions

Only project measures are checked against thresholds. Sub-projects, directories and files are ignored. More

Metric	Over Leak Period	Operator	Warning	Error
Coverage	☐	is less than	50	25

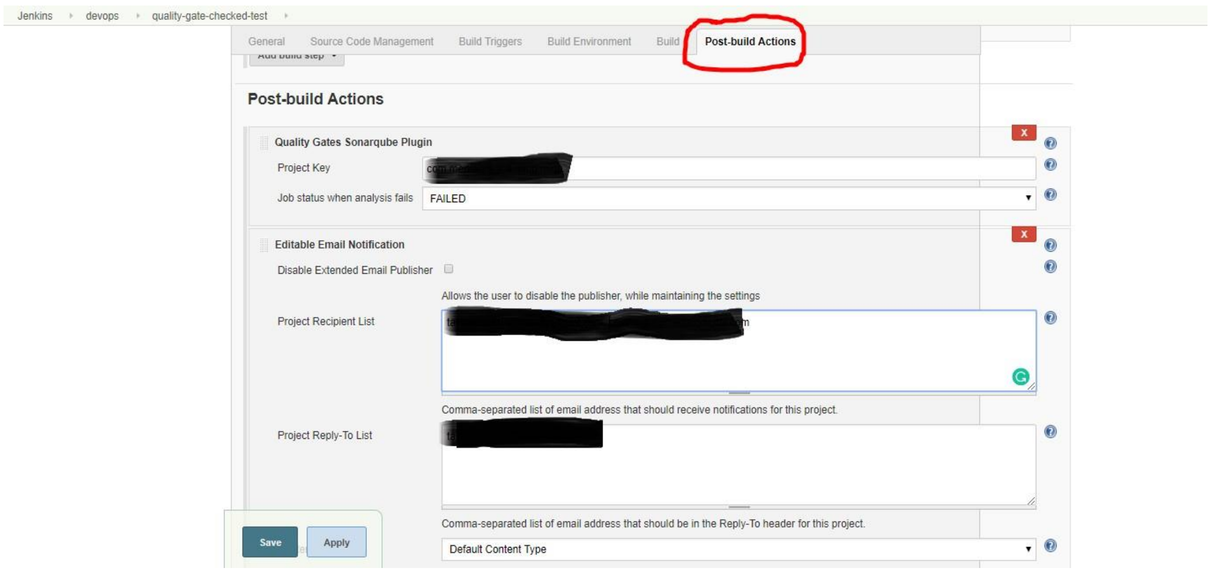
Projects

miqp

- ☐ Cucumber-JVM MIQP
- ☑ miqp

-
-

Now go to the Jenkins job and configure the quality gate validation. Click on the job and go to **Post-build Actions** and provide the project details you have associated with Quality Gate created in the earlier steps.



Jenkins › devops › quality-gate-checked-test ›

General | Source Code Management | Build Triggers | Build Environment | Build | Post-build Actions

Post-build Actions

Quality Gates Sonarqube Plugin

Project Key [redacted]

Job status when analysis fails: FAILED

Editable Email Notification

Disable Extended Email Publisher ☐

Allows the user to disable the publisher, while maintaining the settings

Project Recipient List [redacted]

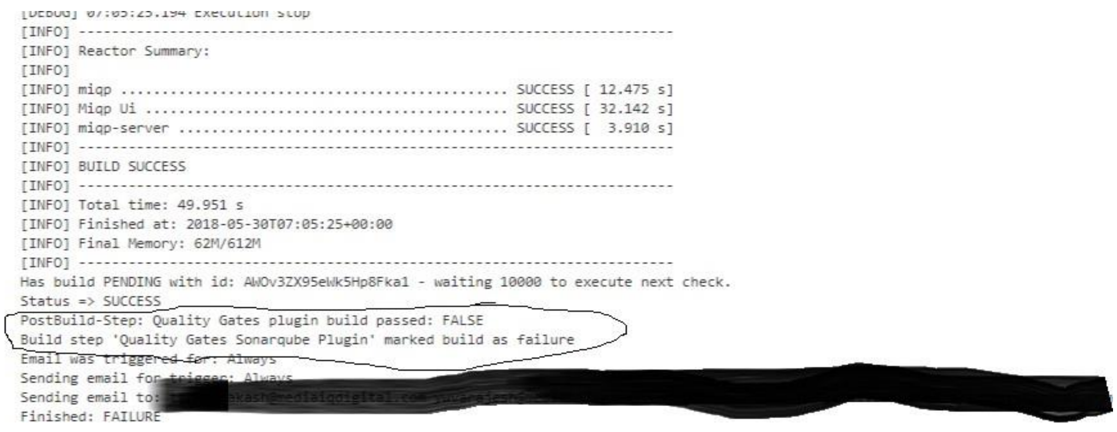
Comma-separated list of email address that should receive notifications for this project.

Project Reply-To List [redacted]

Comma-separated list of email address that should be in the Reply-To header for this project.

Default Content Type

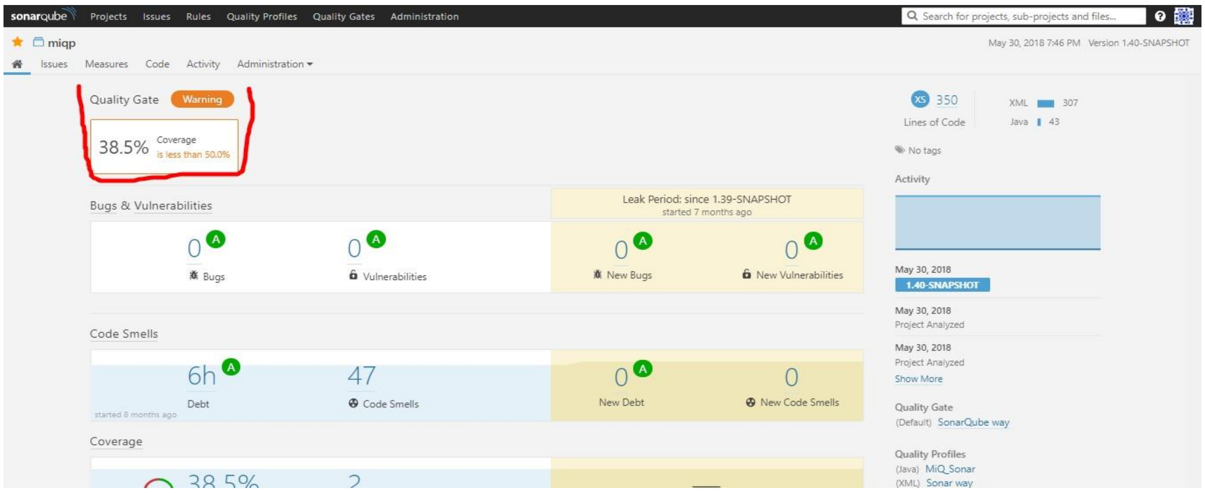
-
- Run the Jenkins job again and verify the build status post quality check enabled.



```
[DEBUG] 07:05:25.194 Execution stop
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO]
[INFO] miqp ............................................... SUCCESS [ 12.475 s]
[INFO] Miqp Ui ............................................ SUCCESS [ 32.142 s]
[INFO] miqp-server ........................................ SUCCESS [  3.910 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 49.951 s
[INFO] Finished at: 2018-05-30T07:05:25+00:00
[INFO] Final Memory: 62M/612M
[INFO] ------------------------------------------------------------------------
Has build PENDING with id: AWOv3ZX95eWk5Hp8Fka1 - waiting 10000 to execute next check.
Status => SUCCESS
PostBuild-Step: Quality Gates plugin build passed: FALSE
Build step 'Quality Gates Sonarqube Plugin' marked build as failure
Email was triggered for: Always
Sending email for trigger: Always
Sending email to: [redacted]
Finished: FAILURE
```

-

As we could see that code passed the build, however, it doesn't pass quality gate check. Therefore, build fails in the end. We can verify the same with the project status in SonarQube server.



sonarqube Projects Issues Rules Quality Profiles Quality Gates Administration

Search for projects, sub-projects and files...

miqp

May 30, 2018 7:46 PM Version 1.40-SNAPSHOT

Issues Measures Code Activity Administration

Quality Gate Warning

38.5% Coverage is less than 50.0%

Bugs & Vulnerabilities

Leak Period: since 1.39-SNAPSHOT started 7 months ago

0 A Bugs | 0 A Vulnerabilities | 0 A New Bugs | 0 A New Vulnerabilities

Code Smells

6h A Debt started 8 months ago | 47 Code Smells | 0 A New Debt | 0 New Code Smells

Coverage

38.5% | 2

XS 350 Lines of Code

XML 307

Java 43

No tags

Activity

May 30, 2018 1.40-SNAPSHOT

May 30, 2018 Project Analyzed

May 30, 2018 Project Analyzed

Show More

Quality Gate (Default) SonarQube way

Quality Profiles (Java) MiQ_Sonar (XML) Sonar way

- **Conclusion**

In this, we have shown how can we implement quality gates for code coverage using SonarQube and Jenkins.