Devops lab manual

DevOps (Jawaharlal Nehru Technological University, Kakinada)

INDEX

Exercise 1: Reference course name Software engineering and Agile software development Get an understanding of the stages in software development lifecycle, the process models, values and principles of agility and the need for agile software development. This will enable you to work in projects following an agile approach to software development. Solve the questions given in the reference course name to gauge your understanding of the topic
Exercise 2 Reference course name Development & Testing with Agile Extreme Programming Get a working knowledge of using extreme automation through XP programming practices of test first development, refactoring and automating test case writing. Solve the questions in the "Take test" module given in the reference course name to gauge your understanding of the topic
Exercise 3: Module name DevOps adoption in projects It is important to comprehend the need to automate the software development lifecycle stages through DevOps. Gain an understanding of the capabilities required to implement DevOps, continuous integration and continuous delivery practices. Solve the questions given in Quiz1, Quiz2, Quiz 3
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.
Exercise 5 Module Name Implementation of CICD with Java and open source stack Configure a static code analyzer 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.

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.

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.

Exercise 8

Module name Implementation of CICD with Java and open source stack

Configure the Jenkins pipeline to call the build script jobs and configure to run it whenever there is a change made to an application in the version control system. Make a change to the background color of the landing page of the web application and check if the configured pipeline runs.

Exercise 9

Module name Implementation of CICD with Java and open source stack Create a pipeline view of the Jenkins pipeline used in Exercise 8. Configure it with user defined messages.

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.

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.

Exercise 12

Module name Course end assessment

In the configured Jenkins pipeline created in Exercise 8 and 9, implement quality gates for code coverage.

Exercise 1:

Reference course name :Software engineering and Agile software development
Get an understanding of the stages in software development lifecycle, the process models, values and principles of agility and the need for agile software development. This will enable we to work in projects following an agile approach to software development.
Solve the questions given in the reference course name to gauge wer understanding of the topic

Agile software development is one of the most effective and reliable processes to turn a vision for business needs into software solutions. The term "Agile" describes software development approaches that employ continual planning, improvement, learning, evolutionary development, team collaboration, and early delivery. The method encourages flexible responses to change.

In Agile software development, four main values are emphasized:

- Individual and team interactions over processes and tools;
- Working software over comprehensive documentation;
- Customer collaboration over contract negotiation;
- Responding to change over following a plan.

How Does Agile Work in Software Development?

Many people think that Agile is restricted only to software development. This is not true. Agile is a mindset that can be used in any project that requires working together to get better results. It can be applied in any sector, from healthcare or construction to the manufacturing field.

The key reason why the Agile methodology has become so associated with software development is that software development teams adopted this approach without doubt as it's such a natural fit for working on digital solutions.

Enterprises that deal with software development companies use Agile in their projects to be focused on incremental delivery, continual planning, collaboration, and learning.

The method can provide amazing results by including multiple feedback instances throughout the development cycle. wer customers are able to see wer solution as it grows and provide valuable feedback instead of observing the product for the first time near the end of development.
Agile software development is focused on creating MVPs (minimum viable products) that pass through iterations, each of which provides incremental value.

What Are the Roots of the Agile Software Development Concept?

Everyone who works according to Agile knows the story of 17 developers who met in Utah in 2001 to discuss these lightweight development methods. They published the Manifesto for Agile Software Development. The Agile Manifesto outlines values for developing software in an iterative and flexible manner.

These values are focused on collaboration, self-empowerment, responsiveness, and building software solutions rather than final products. The Manifesto includes 12 principles.

Agile SDLC

Agile software development lifecycle (SDLC) requires collaborative decision-making and development over several short cycles or sprints.

Agile SDLC is based on a cyclical development approach to software in iterations instead of all in one shot. Teams usually work in multiple cycles that last for 2-4 weeks. Developers are focused on the essential features at any given moment instead of going "according to plan."

Agile software development is represented with various methodologies, including:

- Agile Modeling
- Kanban
- Scrum
- XP – Extreme Programming
- Scrumban

- Adaptive Software Development
- DAD – Disciplined Agile Delivery
- DSDM – Dynamic Systems Development
- Feature Driven Development
- Lean Software Development
  Every methodology has its own goal but the main goal for all is to adapt to change and deliver working software — as quickly as possible.

Waterfall SDLC versus Agile SDLC: defining the difference between two approaches

Let's dive into both models to see the difference between them.

Waterfall

The traditional Waterfall life cycle contains 5 stages:

- Requirement analysis
- Design
- Implementation
- Verification
- Maintenance
  All these stages may last any amount of time. Before reaching the implementation stage, it may take weeks or months. The Waterfall SDLC isn't designed with speed in mind. It looks quite a rationale for complex software projects, where we strive to avoid a brand-destroying fiasco.

However, it is almost impossible to predict all the necessary elements before jumping in. Therefore, wer initial requirement analysis often does not include everything we require. Instead, iterations that are characterized by ongoing testing, quality assurance, and communication usually lead to better results.

Agile

The key characteristic of the Agile model is that the development process breaks larger projects down into smaller sprints. Something valuable is produced at the end of each iteration.

Any product solution produced during a sprint should have a chance to be introduced to the world to receive users' or stakeholders' feedback. This process should be repeated with every sprint.

Agile is strictly sequenced in comparison with the traditional model. we can not start design until research is complete as well as development can not commence until all designs are approved.

In Agile, business people, developers, and designers work together simultaneously. The Agile development cycle looks like this:

- Project planning
- Creating a product roadmap
- Release planning
- Sprint planning
- Daily meetings
- Sprint Review

What Does the Agile Software Development Process Flow Include?

. Concept. wer project should be envisioned and thoroughly prioritized.

. Inception. we should determine team members, provide funds, and discuss initial environments and requirements.

. Iteration (construction). Developers deliver working software based on feedback and iteration requirements.

. Release. This stage includes QA testing, training (internal and external), documentation development, and the final release of the iteration.

. Production with the ongoing support of the software.

. Retirement includes completing activities and customer notification.

What Does the Agile Iteration Workflow Include?

The Agile SDLC is dominated by the iterative process when every iteration results in the working software and supporting elements (for example, documentation, available for use by customers).

During the software development lifecycle in Agile multiple iterations take place. Each of them follows its own workflow. It is important during every iteration that the business stakeholders and clients provide feedback to guarantee that features meet their needs.

Here's how a typical iteration process flow may look like:

- Requirements. we have to determine them for the iteration based on the product backlog, sprint backlog, feedback of customers and stakeholders.
- Development and design start based on the defined requirements.
- Testing includes QA testing, internal and external training, and documentation development.
- Delivery is time to deliver the working iteration into production.
- Feedback. At this stage, we should accept customer and stakeholder feedback and work it into the requirements of the next iteration.
  During the project, the product backlog can be fulfilled with the additional features, and the rest of the process is repeating the steps over and over until all of the product backlog elements have been fulfilled. As a result, we get the process flow rather than a linear process.

Now, when we know so much theory, it's time to learn the specific steps to implement the Agile software development methodology.

Consistent Steps on How to Implement Agile SDLC in 2021

By choosing the Agile SDLC model, we follow a distinct process to foster a smooth design and development transition for wer company.

1. Take a decision about the transition as a whole

we should have all the required skills and levers to transmit wer team to the Agile SDLC. Consider not just wer developers and managers, but also involve key customers and other stakeholders. Remember that one of the core values of the Agile methodology is constant collaboration with wer stakeholders.

2. Dive into the values and principles to shift wer mindset

Do not hesitate to refer to the Agile Manifesto routinely. Use it during the team meetings to discuss company values and projects. Experienced Agile teams meet their project objectives much more often than immature teams.

3. Select a proper Agile framework

Choosing the right Agile model depends on wer company's development teams' size, goals, needs, and processes. For example:

- Kanban will suit we in case wer team is already used to the boards. Scrumban is also becoming a popular option.
- Scrum will be great if we want to leverage the key benefits like adaptability and speed at a small scale.
  To manage the life cycles of multiple products at scale, we may use a scaling Agile framework, for example – SAFe.

4. Optimize a product backlog

we should work with the PO and stakeholders to build a backlog of potential features and required user stories for wer product. As new ideas arise and priorities change, make sure to communicate updates in real-time with everyone involved.

5. Plan sprints

Review all the features, stories, and bugs in wer backlog during a set meeting with the PO and all team members. Discuss what should be prioritized in the upcoming project sprint.

6. Run daily meetings

The Daily Stand Up is a basic event for Agile teams whether they use Scrum or other frameworks. This meeting is about focusing on what we did yesterday, what we'll do today, and what roadblocks we face.

7. Test the iteration with the stakeholders

If wer stakeholders accept the changes, the sprint was a success. It means we move on to new features and user stories. If not — we will have to tackle their issues in the next sprint.

8. Evaluate we immediate goals and backlog

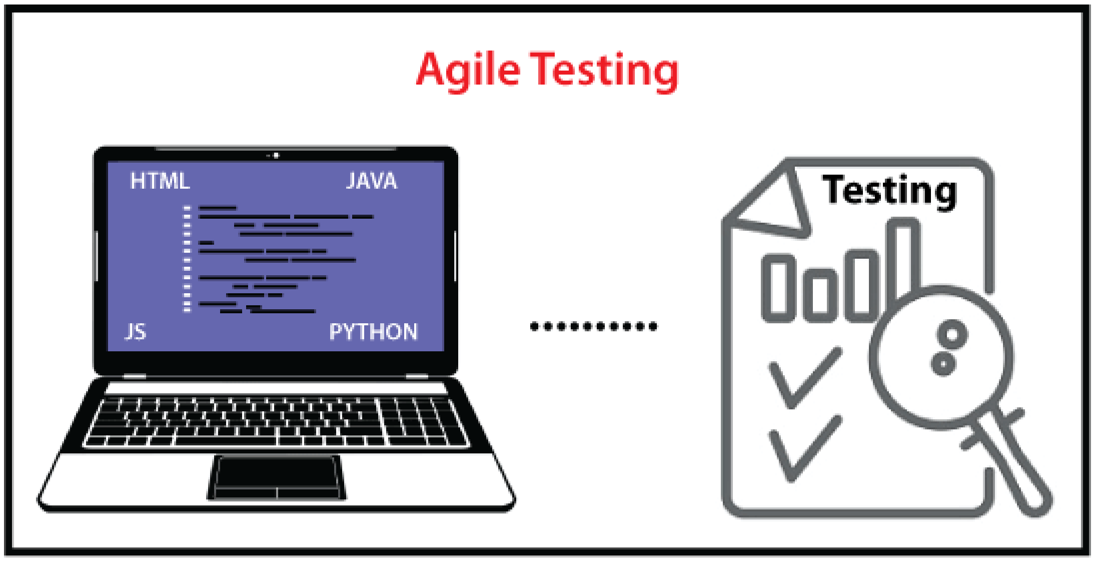
After we complete a sprint, a kind of inventory of sorts should be run. Define the areas for improvement, and consider how the results of the sprint affect wer future backlog priorities. The Agile SDLC is made up of several sprints. So just because this is the last step doesn't mean we're done.

Exercise 2:

Reference course name: Development & Testing with Agile: Extreme Programming
Get a working knowledge of using extreme automation through XP programming practices of test first development, refactoring and automating test case writing.
Solve the questions in the "Take test" module given in the reference course name to gauge wer understanding of the topic

In agile testing, the word "Agile" primarily signifies something that can be performed quickly and immediately, but also in the area of software development.

The core-functional agile team implements it in order to test the software product and its several modules. The implementation of agile testing makes sure to deliver a high quality product as bugs or defects get deleted in the initial stage of the project itself.



Unlike the Waterfall model, Agile Testing can create at the beginning of the project with endless incorporation between development and testing. It is not a sequential but the continuous process.

The agile testing process is a smart way of testing complicated software, which accepts more effective results as compared to the traditional testing procedures.

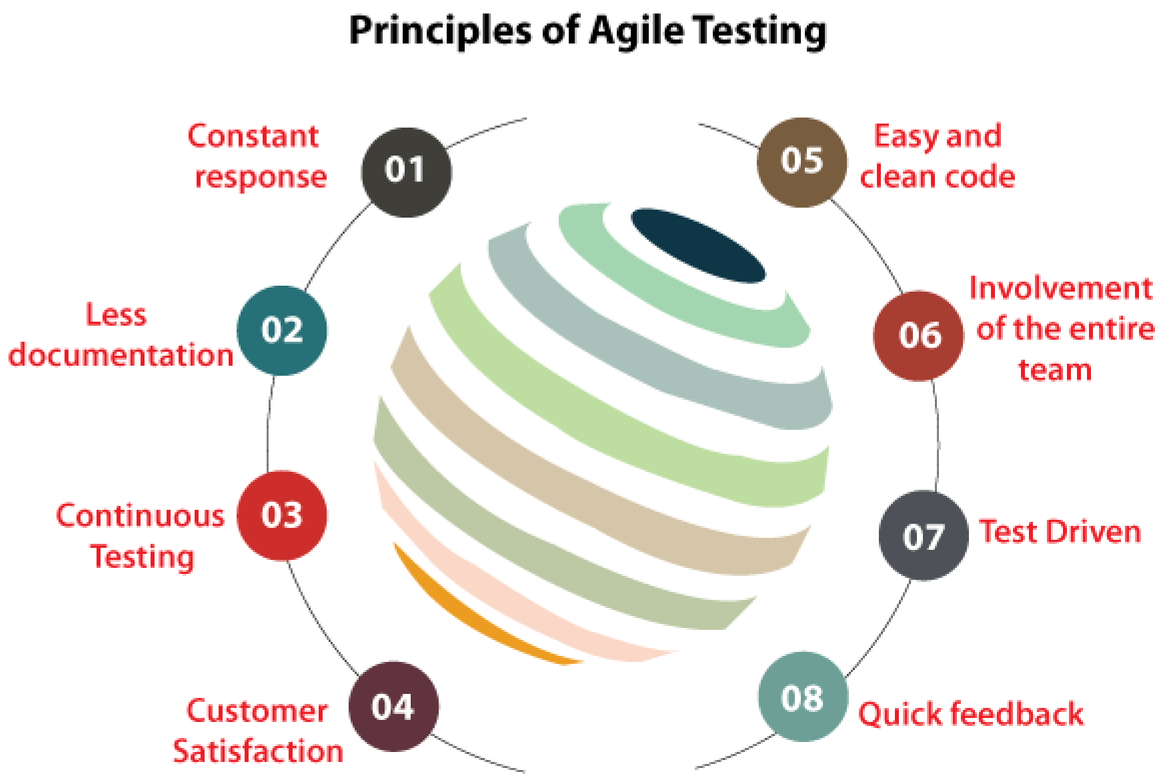
In the modern days of software testing, agile testing has gained a lot of acceptance and significance. The execution of agile testing will help us identify the initial error and elimination, giving better results in less development time and costs.

Principles of Agile Testing

Agile Testing includes various different principles that help us to increase the productivity of our software.

1. Constant response
2. Less documentation
3. Continuous Testing
4. Customer Satisfaction
5. Easy and clean code
6. Involvement of the entire team
7. Test-Driven
8. Quick feedback

For our better understanding, let's see them one by one in detail:



1. Constant Response

The implementation of Agile testing delivers a response or feedback on an ongoing basis. Therefore, our product can meet the business needs.

In other words, we can say that the Product and business requirements are understood throughout the constant response.

2. Less Documentation

The execution of agile testing requires less documentation as the Agile teams or all the test engineers use a reusable specification or a checklist. And the team emphases the test rather than the secondary information.

3. Continuous Testing

The agile test engineers execute the testing endlessly as this is the only technique to make sure that the constant improvement of the product.

4. Customer Satisfaction

In any project delivery, customer satisfaction is important as the customers are exposed to their product throughout the development process.

As the development phase progresses, the customer can easily modify and update requirements. And the tests can also be changed as per the updated requirements.

5. Easy and clean code

When the bugs or defects occurred by the agile team or the testing team are fixed in a similar iteration, which leads us to get the easy and clean code.

6. Involvement of the entire team

As we know that, the testing team is the only team who is responsible for a testing process in the Software Development Life Cycle. But on the other hand, in agile testing, the business analysts (BA) and the developers can also test the application or the software.

7. Test-Driven

While doing the agile testing, we need to execute the testing process during the implementation that helps us to decrease the development time. However, the testing is implemented after implementation or when the software is developed in the traditional process.

8. Quick response

In each iteration of agile testing, the business team is involved. Therefore, we can get continuous feedback that helps us to reduces the time of feedback response on development work.

How is Agile Methodology used in Testing?

Agile Testing is a fast and informal testing process. In simple terms, we can say that it is specified as an advanced and dynamic type of Testing that is performed regularly throughout every iteration of the SDLC (Software Development Life Cycle) by the agile test engineers.

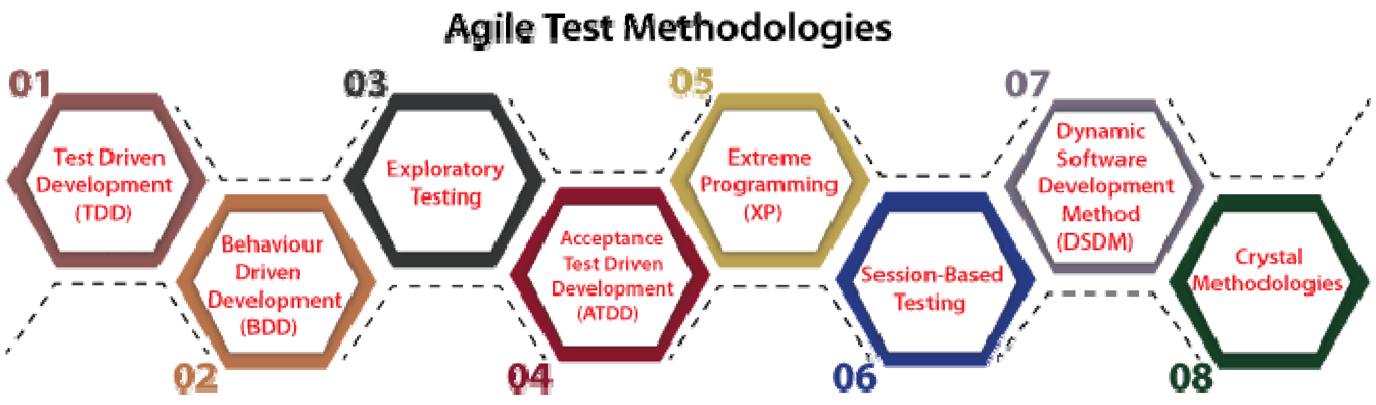
If we deliver the software quickly with the best of the attributes, and the customer satisfaction is the primary concern at some stage in the agile testing process.

Agile Testing Methodologies

When we are executing the agile testing, the team takes help from several agile methodologies, which support them in accomplishing the precise results.

Some of the effective agile testing methodologies are as follows:

- Test-Driven Development (TDD)
- Behavior Driven Development (BDD)
- Exploratory Testing
- Acceptance Test-Driven Development (ATDD)
- Extreme Programming (XP)
- Session-Based Testing
- Dynamic Software Development Method (DSDM)
- Crystal Methodologies


Agile Test Methodologies
01
Test Driven Development (TDD)
02
Behaviour Driven Development (BDD)
03
Exploratory Testing
04
Acceptance Test Driven Development (ATDD)
05
Extreme Programming (XP)
06
Session-Based Testing
07
Dynamic Software Development Method (DSDM)
08
Crystal Methodologies

Test-Driven Development (TDD)

The test-driven development method begins with the test itself. As the name proposes, the TDD varies upon the repetition of the development cycle.

We already knew that the first step in of development cycle is to create a unit test case. And in the next step, we will be designing the code that fits the test case in order to execute the test cases.

Hence, the whole code is designed until the unit test passes. Generally, the test-driven development is executed by using the automated testing tools and implement on units and components of the code.

Behavior-Driven Development (BDD)

The following method in agile testing is behavior-driven development. The BDD enhances the communication between the project stakeholders to facilitate the members adequately and understood all the components before the development process begins.

It is constructed on the same rules as TDD and ATDD. Therefore, the code is developed as per the test case designed in this testing methodology too.

The primary purpose of this development is to emphasize the identification of business needs and outputs. And the development should be consistent to a business output.

In behavior-driven development, we need to follow the below steps:

1. Describe the behavior.
2. Generating the test case.
3. Writing code as per the test case is specified.
4. Continuing the process until the code passes the test case.

Exploratory Testing

In Software testing, exploratory testing is one particular type where the test engineers have the fundamental freedom to explore the code and create the most effective software.

In simple words, we can say that if we don't have the requirement, then we do one round of exploratory testing.

Exploratory testing is a very significant part of the agile test as it helps discover the unknown risks from the software that a simple testing approach could not have noticed.

To explore each aspect of the software functionality, the test engineer creates various test cases, executes different tests, and records the process to learn it and understand its particular flow.

While performing the exploratory testing, we need to follow the below steps:

- Exploring the application in all possible ways
- Understanding the flow of the application
- Preparing a test document
- Testing the application

For more information related to the exploratory testing, refers to the following link: https://www.javatpoint.com/exploratory-testing.

Acceptance Test-Driven Development (ATDD)

Another methodology of agile testing is Acceptance Test-Driven Development (ATDD). The ATDD approach emphasizes the customer's requirements by involving the team members with different viewpoints.

The team's members of development, testing, and the customers come together in order to develop the acceptance test from the customer's perspective.

In Acceptance Test Driven Development, the code is acquired along with the developed acceptance test case.

It is a very customer-centered methodology; the primary objective of using the ATDD methodology is to develop a program based on the user's view.

Extreme Programming (XP)

The next significant agile methodology is Extreme Programming which is denoted as XP. When there is a continuous modification in the user requirements, we will go for the extreme programming methodology.

Just like other agile testing methodologies, Extreme Programming is also a customer-centric methodology.

The XP will help us deliver a quality product, which meets the customer's requirements that are made clear throughout the process of development and testing.

Session-Based Testing

In the row of various agile testing methodologies, the next methodology is Session-based testing. It is mainly is created on the values of exploratory testing.

Though session-based testing contains some structure and on the other hand, exploratory testing is performed unexpectedly without any planning. It is used to help us identify the hidden bugs and defects in the particular software.

The session-based testing structure is prepared by executing the tests in continuous sessions where test engineers must report the tests, which took place throughout the process.

Dynamic Software Development Method (DSDM)

Another effective method of agile testing is Dynamic Software Development Method. It is a Rapid Application Development (RAD) approach that offers a delivery framework to agile projects.

In other words, we can say that the Dynamic Systems Development technique (DSDM) is a correlate degree agile code development approach, which gives a framework for developing and maintaining systems.

The Dynamic Software Development Method can be used by users, development, and testing teams.

Crystal Methodologies

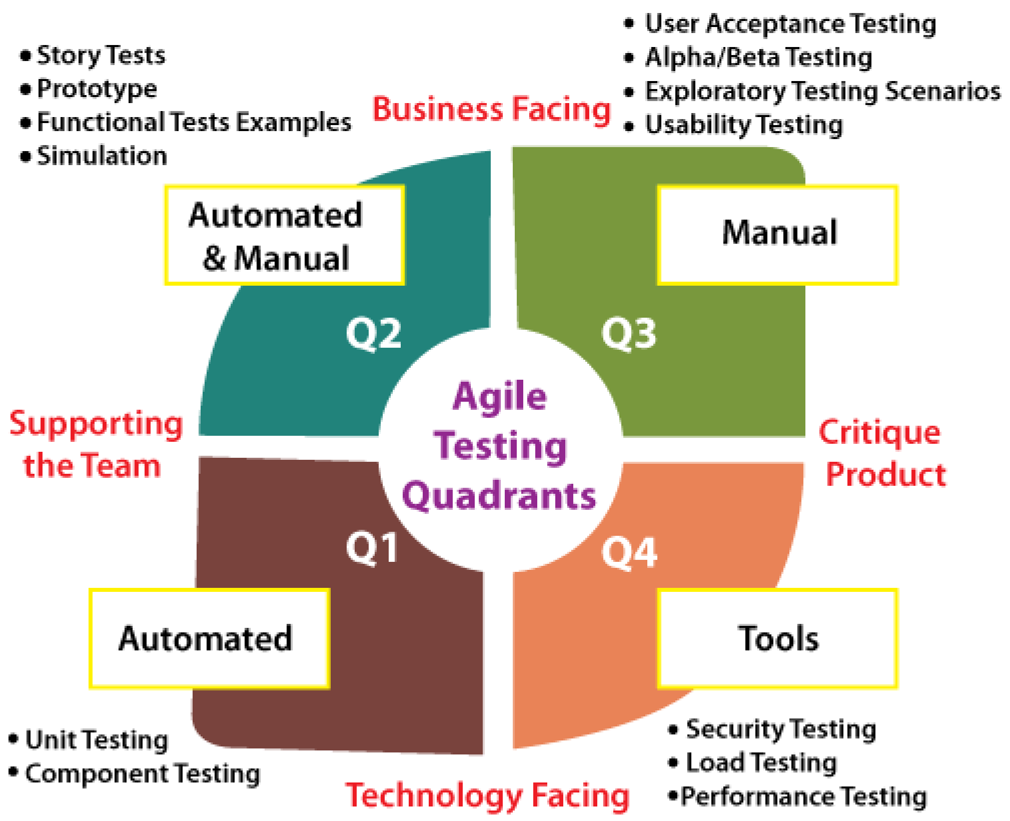
The subsequent agile testing is crystal methodologies. This methodology mainly emphasizes recording, cyclic delivery, and wrap-up, which is to make sure during the various analyses.

Agile Testing Quadrants

It has different quadrants to easily understand agile testing, which divides the whole testing process into four parts.

In addition to the four quadrants, the left two specify the test engineer that code to write, and the right two quadrants help them understand the code improved with the support of response to the left quadrants.

These agile testing quadrants may be understood as a traditional process or strategies to perform the end-to-end agile testing of a software application in four different stages, as we can see in the following image:



Let us discuss them one by one to understand the process of agile testing:

1. Quadrant 1 (Automated)
2. Quadrant 2 (Automated and manual)
3. Quadrant 3 (Manual)
4. Quadrant 4 (Tools)

Quadrant 1 (Automated)

In the first quadrant of Agile testing, we will see mainly emphasis on the quality of the code. We can say internal code quality, which contains the test cases and test components that is executed by the test engineers.

And these test cases are technology-driven and used for automation testing in order to enhance the code and support the testing team to perform their tasks.

All through the first quadrant of agile testing, we can execute the following testing:

- Unit Testing
- Component Testing

Quadrant 2 (Automated and manual)

In the second quadrant of Agile testing, we will see mainly emphasis on the customer requirements given to the team before and throughout the testing procedures, which expands the business results of the newly created software.

The test case involved in this second quadrant is business-driven, usually manual and automated functional tests, prototypes, and examples of test scenarios performed by the testing team.

In quadrant 2, we can execute the following tests:

- Testing scenarios which may occur and workflow
- Implementing the pair testing
- Testing the user story and experiences like prototypes.

Quadrant 3 (Manual)

The third quadrant of agile testing primarily emphasizes the response for the previous two phases (Quadrant 1 and Quadrant 2).

The execution of agile testing involves many iterations. And in this quadrant, these reviews and responses of the particular iterations are sustained that helps to strengthen the code.

To test the user experience and determine business results allows the testing team to learn as the test develops.

The team, business owners, and even customers realistically use the product. In the third quadrant, the test cases have been designed to implement automation testing, which helps us develop certainty in the particular product.

In quadrant 3, below types of testing can be executed:

- Usability testing
- Collaborative testing
- Exploratory testing

- User acceptance testing
- Pair testing with customers

Quadrant 4 (Tools)

The last and fourth Quadrant of agile testing primarily emphasizes the product's non-functional requirements, including compatibility, performance, security, and constancy.

In other words, we can say that the fourth Quadrant ensures that the code fulfils all the non-functional requirements.

Like other Quadrants, various types of testing are performed in quadrant 4 to deliver the non-functional qualities and the expected value.

- Non-functional testing such as Stress testing, performance testing, and load testing, etc.
- Scalability testing
- Security Testing
- Data Migration Testing
- Infrastructure testing

Agile Test Plan

As compared to the waterfall model, the agile test plan is created and updated for every release. Furthermore, the agile test plan contains those types of testing executed in a specific iteration, such as test environments, test data requirements, test results, and infrastructure.

The agile test plans emphasize the following:

- Testing Scope: The testing scope specifies the sprint goals, test scope, and test coverage in which the test will be implemented.
- Performance and Load Testing: Here, it specifies the different testing methods and procedures.
- Types of testing or levels as per the feature's complexity: It defines those types of testing or levels of testing which are going to be used. And also specifies the data and configurations for the test and the environment in which the test will be executed.

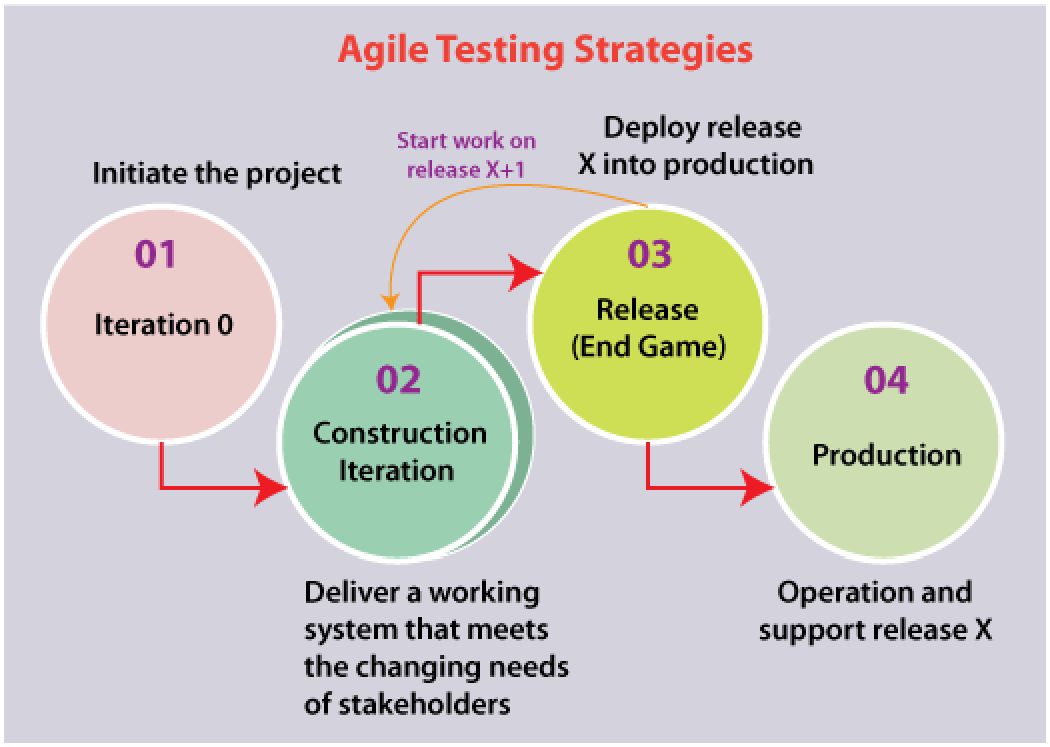
- Mitigation or Risks Plan: It defines the backup plan prepared to overcome the risks or issues. And it also identifies the challenges which might face at the time of testing of the application in the current release.
- Deliverables and Milestones: It sets the deliverables and milestones of the tests as per the customer's perspective.
- Infrastructure Consideration: It governs the infrastructure which is required to perform the tests.
- Resourcing: It lists out the test tasks and the occurrence of tests, which defines how many times the tests will be executed.
- Establish the New functionalities which are being tested.

Agile Testing Strategies

Agile testing has four different approaches, which help us to enhance our product quality.

1. Iteration 0
2. Construction iteration
3. Release End Game or Transition Phase
4. Production

Let us discuss them one by one in detail:


Agile Testing Strategies
Initiate the project
Start work on release X+1
Deploy release X into production
01
Iteration 0
02
Construction Iteration
03
Release (End Game)
04
Production
Deliver a working system that meets the changing needs of stakeholders
Operation and support release X

1. Iteration 0

The first strategy or approach of agile testing is iteration 0. In this, we execute the preliminary setup tasks such as finding people for testing, establishing testing tools, preparing resources or usability testing lab, etc.

In Iteration 0, the below steps are accomplished:

- Verifying a business case for the project and boundary situations, and the project scope.
- Summarise the important requirements and use cases that will determine the strategic trade-offs.
- Plan the initial project and cost valuation
- Detecting the risk.
- Outline one or more candidate designs

2. Construction Iteration

The next strategy of agile testing is Construction Iteration. During this approach, the majority of the testing is performed.

The construction iteration is performed as a set of iterations in order to create an increment of the solution.

In simple words, we can say that the agile team follows the listed requirement within each iteration where they can acquire the most significant business needs or requirements left behind from the work item stack and then execute them.

The construction iteration process divided into the following two types of testing:

- Confirmatory Testing
- Investigative Testing

1. Confirmatory Testing

To ensure that the product meets all the stakeholders' requirements, we will execute the confirmatory testing.

Confirmatory testing can be further separated into another two types of testing, which are as follows:

- Agile Acceptance Testing
- Developer Testing

Agile Acceptance Testing: It is a combination of functional testing and acceptance testing. The agile acceptance testing can be executed by the development team and stakeholders together.

Developer Testing: It is a combination of unit testing and integration testing. And it validates both the application code as well as the database schema.

Note: We can automate both (agile acceptance testing and developer testing) to ensures that continuous regression testing has occurred.

2. Investigative Testing

In order to test deep and identify all the issues, which are overlooked in confirmatory testing, we will execute the investigative testing.

3. Release End Game or Transition Phase

The third approach of agile testing is release. The objective of this specific approach is to implement our system effectively in production.

The test engineer will be working on its defect stories in the end game. In the release end game or transition stage, we have the following activities:

- Support Individuals
- Training of end-users
- Operational People

Similarly, it involves some additional activities as well:

- Back-up and Restoration
- Marketing of the product release
- User documentation
- Completion of system

The last agile methodology testing stage encompasses whole system testing and acceptance testing. To complete our final testing phase without having any difficulties, we should have to test the product more thoroughly in construction iterations.

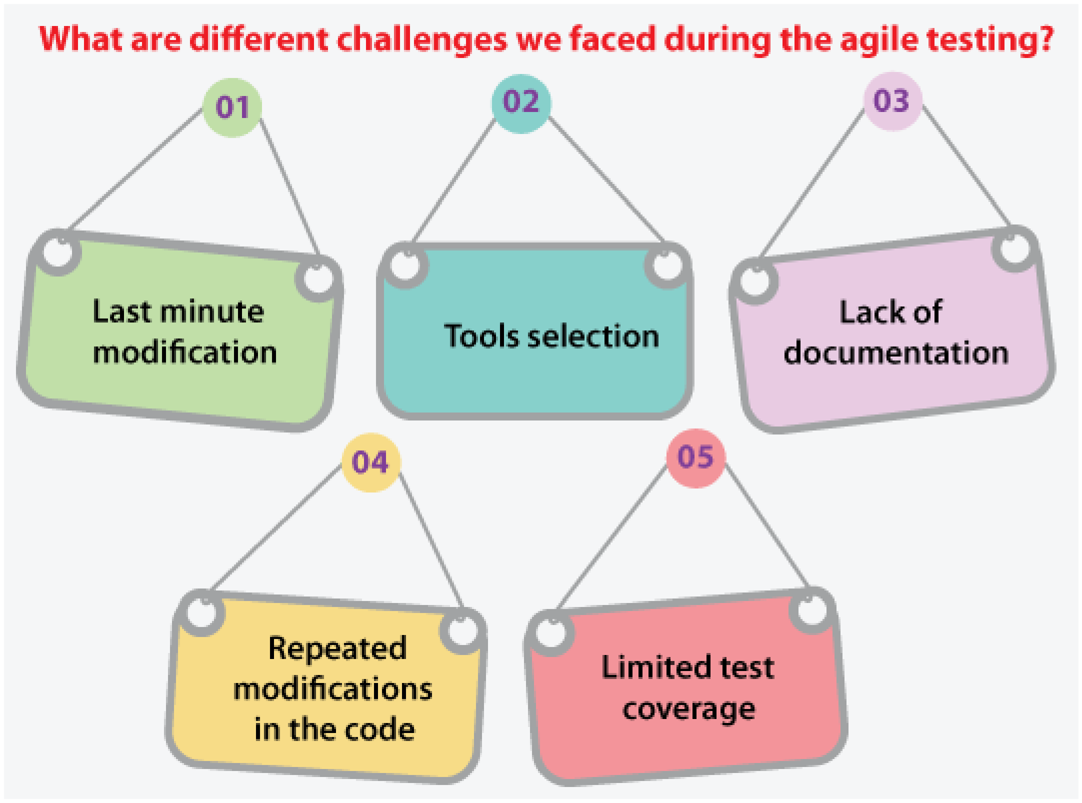
4. Production

The product will move on to the production stage as soon as the release stage is completed.

What are the different challenges we faced during the agile testing?

Generally, while performing agile testing, a testing team may face some challenges. Let see those challenges all together for our better understanding:

- Last-minute modification
- Tools selection
- Lack of documentation
- Repeated modifications in the code
- Limited test coverage


What are different challenges we faced during the agile testing?
01
Last minute modification
02
Tools selection
03
Lack of documentation
04
Repeated modifications in the code
05
Limited test coverage

- Last-minute Modification

The most faced challenges during the agile testing are last-minute modifications by the client, which gives significantly less time to the testing team to design the test plan, which may affect the product quality. And sometimes, the test engineer is often required to play a semi-developer role.

- Tools Selection

The selection of tools during agile testing is essential because if we select the wrong tool, it will waste our time as well as money.

As we already knew, the Test execution cycles are highly reduced, and for the regression testing, we will have minimal timing.

- Lack of Documentation

Another frequently faced challenge while executing agile testing is the lack of documentation. The probabilities of error are more agile as documentation is given less importance and ultimately puts more burden on the testing team.

- Repeated Modifications in the code

In an agile method, requirement modification and updation are fundamental, making it the major challenge for the Quality assurance team.

- Limited Test Coverage

In agile testing, new features are initiated quickly, decreasing the available time for the testing teams to find whether the latest features are as per the requirement and address the business suits.

After seeing all the frequent challenges, the question arises how do we overcome them? Therefore, in the below topic, we are going to discuss that:

How do we overcome Agile testing challenges?

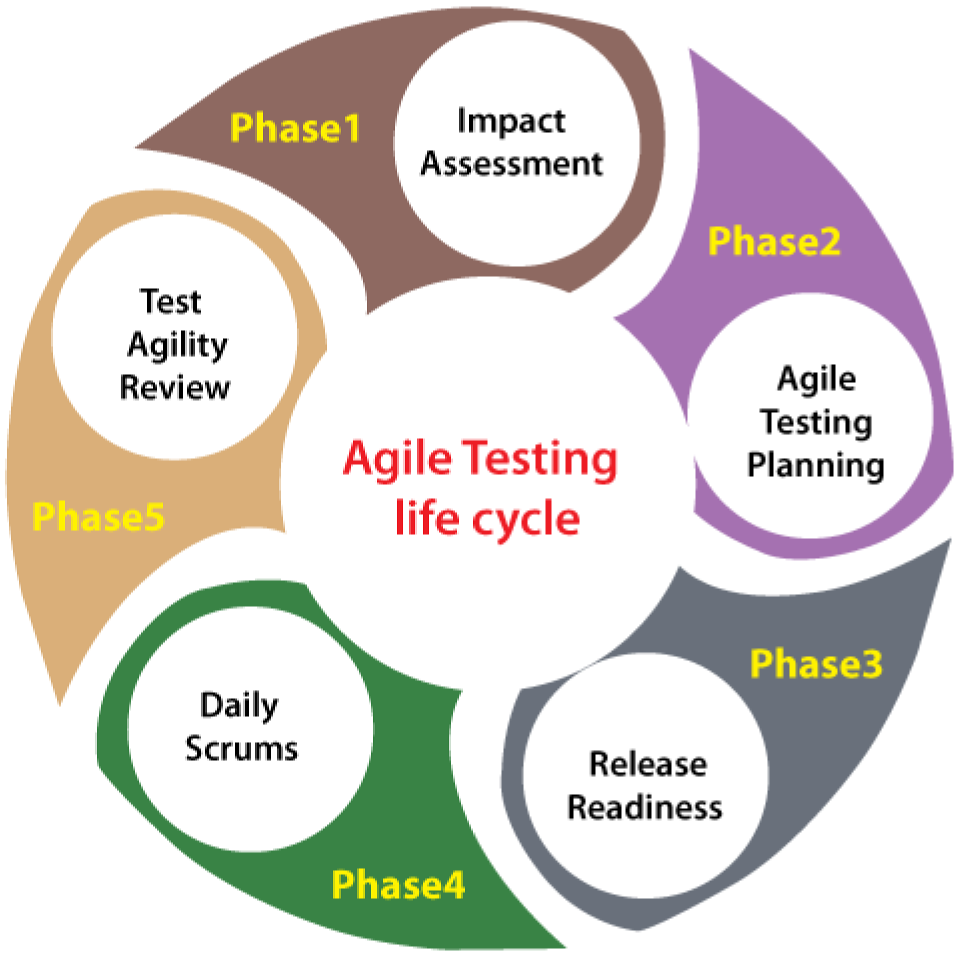
As we understood from the definition of Agile Testing, it comprises less or no documentation, which creates problems for the testing team to predict the expected results and becomes the obstacle in the testing process.

And it also makes it challenging to choose the direction and path of the testing to be performed. Hence, to overcome the agile testing challenges, we can implement the following best options:

- We can execute the Exploratory Testing to conquer the agile testing challenges.
- We can perform the automated unit tests to speed up the agile testing process.
- Test-Driven Development could be a good option in order to overcome the agile testing challenges.
- Furthermore, we can overcome these issues or challenges with the help of the agile testing specification and make sure to perform improved and qualitative Testing in a guided way.
- We can implement automated Regression Testing.

Agile Testing life cycle

Just like other types of testing has their life cycle process, Agile Testing life cycle completed into five different phases, as we can see in the following image:



Let understand all the phases in detail:

Phase1: Impact Assessment

The first phase of the Agile testing life cycle is Impact assessment. Here, we collect the inputs and responses from users and stakeholders to execute the impact assessment phase. This phase is also known as the feedback phase, which supports the test engineers to set the purpose for the next life cycle.

Phase2: Agile Testing Planning

The second phase of the Agile testing life cycle is agile testing planning. In this phase, the developers, test engineers, stakeholders, customers, and end-users team up to plan the testing process schedules, regular meetings, and deliverables.

Phase3: Release Readiness

The next phase of the Agile testing life cycle is release readiness, where test engineers have to review the features which have been created entirely and test if they are ready to go live or not and which ones need to go back to the previous development phase.

Phase4: Daily Scrums

Daily scrums are the next phase of the Agile testing life cycle, which involves the daily morning meetings to check on testing and determine the objectives for the day.

And, in order to help the test engineers to understand the status of testing, the goals and targets of the day are set daily.

Phase5: Test Agility Review

The last and final phase of the Agile life cycle is the test agility review. The test agility phase encompasses the weekly meetings with the stakeholders to evaluate and assess the progress against goals.

In other words, we can say that the agility reviews are implemented regularly in the development process to analyze the progress of the development.

Advantages of Agile Testing

For the past few years, Agile software testing has been a significant part of the IT field. Here, we are discussing some essential benefits of Agile testing:

- Agile testing gives way to get regular feedback and reviews directly from the end-user, which helps us enhance the software product's quality and attribute.
- The implementation of agile testing will save lots of time and money, making the estimation of cost more transparent.
- Through daily meetings, we can determine better issues.
- Agile testing reduces documentation, or we can say it requires less documentation to execute the agile testing.
- The most significant advantage of implementing Agile software testing is reducing errors and enhancing software productivity.
- As we understood from the above discussion, the workload is divided into small parts in agile software development and restricts the developer from going off the track. And in the results, we will get more minor inconsistencies and higher efficiency.

Disadvantage of Agile Testing

Agile testing is a creative method to test the software application, but still, we have some disadvantages of using Agile Testing:

- The most significant disadvantage of agile testing is that if two or more members leave a job, it will lead to project failure.
- Determination is needed while performing any testing to test the application or the software and becomes inconsistent for the testing team.
- It makes it difficult for us to predict expected results because there are no or fewer documentation results into explicit conditions and requirements.
- Sometimes, it leads us to introduce new bugs in the system because the bug fixes, modifications, and releases repeatedly happen in agile testing.

Overview

In this tutorial, we have seen the in-depth knowledge of agile testing like Principle of Agile Testing, Strategies, Quadrants, agile testing life cycle, different challenges we faced during the agile testing, and Advantages/ Disadvantages of Agile Testing.

After understanding all the topic mentioned earlier, we can say that Agile testing is one of the best approaches for the present-day software as this software are highly complex and demand comprehensive testing.

It allows the test engineer to be flexible and able to encompassing any requirement modification.

Agile testing is becoming very famous among significant software development organizations as it a very customer-centric one that ensures a high-quality product.

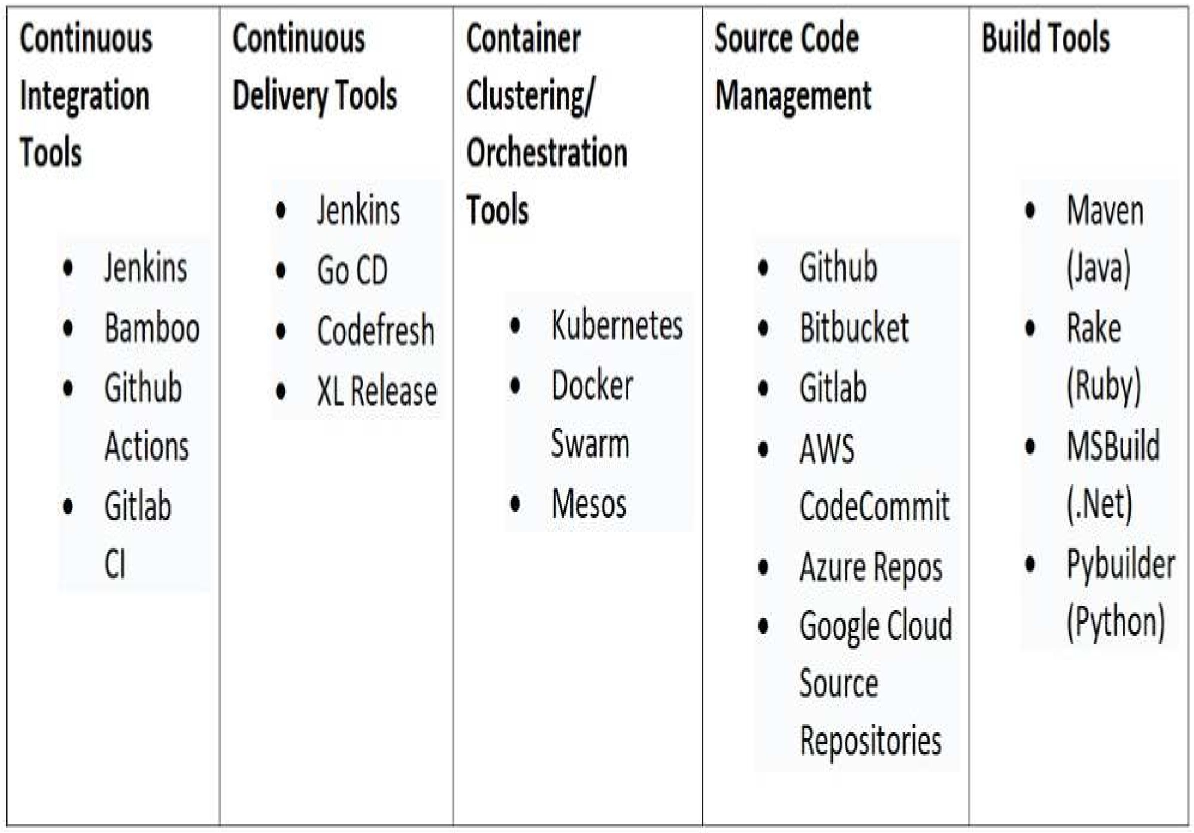
The execution of agile testing requires the involvement of customers, a high level of communication among the developers, test engineers as they all are working together in order to test the particular software.

And in a result, we can deliver a better quality of software that fulfils the customer and user expectations.

In software testing, the agile methodology encompasses testing as soon as possible in the Software Development Life Cycle.

At last, we can say that the communication among the teams (development team, testing team, customers) plays an essential role in making the agile testing successful.

Example: Some popular tools used for enabling enterprise DevOps setup



Continuous Integration Tools	Continuous Delivery Tools	Container Clustering/ Orchestration Tools	Source Code Management	Build Tools
• Jenkins • Bamboo • Github Actions • Gitlab CI	• Jenkins • Go CD • Codefresh • XL Release	• Kubernetes • Docker Swarm • Mesos	• Github • Bitbucket • Gitlab • AWS CodeCommit • Azure Repos • Google Cloud Source Repositories	• Maven (Java) • Rake (Ruby) • MSBuild (.Net) • Pybuilder (Python)

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.

Git is a version control system which lets we track changes we make to wer files over time. With Git, we can revert to various states of wer files (like a time traveling machine). we can also make a copy of our file, make changes to that copy, and then merge these changes to the original copy.

With Git, we can create an identical copy of that file and play around with the navigation bar. Then, when we are satisfied with wer changes, we can merge the copy to the original file.

we are not limited to using Git just for source code files – we can also use it to keep track of text files or even images. This means that Git is not just for developers – anyone can find it helpful.

How to install Git

In order to use Git, we have to install it on wer computer. To do this, we can download the latest version on the official website. we can download for wer operating system from the options given.

we can also install Git using the command line, but since the commands vary with each operating system, we'll focus on the more general approach.

How to configure Git

I will assume that at this point we have installed Git. To verify this, we can run this command on the command line: git --version. This shows we the current version installed on we PC.

The next thing we'll need to do is to set wer username and email address. Git will use this information to identify who made specific changes to files.

To set wer username, type and execute these commands: git config --global user.name "YOUR_USERNAME" and git config --global user.email "YOUR_EMAIL". Just make sure to replace "YOUR_USERNAME" and "YOUR_EMAIL" with the values we choose.

How to Create and Initialize a Project in Git

We are finally done with installing and setting up Git. It is now time to create our project.

I have created a folder on my desktop called Git and GitHub tutorial. Using the command line, navigate to wer new project's location. For me, I would run the following commands:

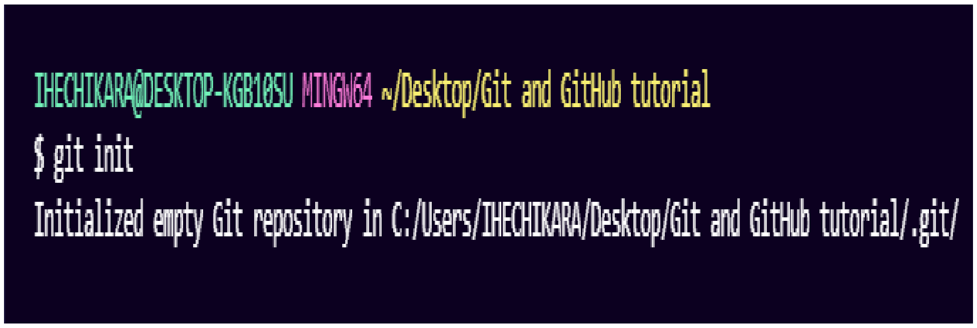
cd desktop

cd Git and GitHub tutorial

If we are new to the command line and are still learning how to use it to navigate around wer PC, then I would suggest using Microsoft's Visual Studio Code. It is a code editor which has an inbuilt terminal for executing commands.

After installing VS Code, open wer project in the editor and open a new terminal for wer project. This automatically points the terminal/command line to wer project's path.

Now to initialize wer project, simply run git init. This will tell Git to get ready to start watching wer files for every change that occurs. It looks like this:



```
IHECHIKARA@DESKTOP-KGB10SU MINGW64 ~/Desktop/Git and GitHub tutorial
$ git init
Initialized empty Git repository in C:/Users/IHECHIKARA/Desktop/Git and GitHub tutorial/.git/
```

git init

The first line has information about my PC and the path to where the folder exists. The second line is the command git init, and the third line is the response sent back telling me that my repository (repo) has been initialized. It is considered empty because we have not told Git what files to track.

A repository is just another way to define a project being watched/tracked by Git.

Git project files
I have created only one file called todo.txt. This is what the file looks like:

Before we proceed with learning other Git commands, let's talk about GitHub.

GitHub
GitHub is an online hosting service for Git repositories. Imagine working on a project at home and while we are away, maybe at a friend's place, we suddenly remember the solution to a code error that has kept we restless for days.

we cannot make these changes because wer PC is not with we. But if we have wer project hosted on GitHub, we can access and download that project with a command on whatever computer we have access to. Then we can make wer changes and push the latest version back to GitHub.

In summary, GitHub lets we store wer repo on their platform. Another awesome feature that comes with GitHub is the ability to collaborate with other developers from any location.

Now that we have created and initialized our project locally, let's push it to GitHub.

If we are a beginner, we will come across some new terms like push, commit, add, and so on – but do not be overwhelmed by them. With some practice we will be able to remember these terms and what they do.

How to push a repository to GitHub

I will divide this section into steps to help we understand the process more clearly.

Step 1 – Create a GitHub account

To be able to use GitHub, we will have to create an account first. we can do that on their website.

Step 2 – Create a repository

we can click on the + symbol on the top right corner of the page then choose "New repository". Give wer repo a name then scroll down and click on "Create repository".

Step 3 – Add and commit file(s)

Before we "add" and "commit" our files, we need to understand the stages of a file being tracked by Git.

Committed state

A file is in the committed state when all the changes made to the file have been saved in the local repo. Files in the committed stage are files ready to be pushed to the remote repo (on GitHub).

Modified state

A file in the modified state has some changes made to it but it's not yet saved. This means that the state of the file has been altered from its previous state in the committed state.

Staged state

A file in the staged state means it is ready to be committed. In this state, all necessary changes have been made so the next step is to move the file to the commit state.

we can understand this better by imagining Git as a camera. The camera will only take a snapshot when the file reaches the commit state. After this state, the camera starts comparing changes being made to the same file with the last snapshot (this is the modified state). And when the required changes have been made, the file is staged and moved to the commit state for a new snapshot.

This might be a lot of information to take in at the moment, but do not be discouraged – it gets easier with practice.

How to add files in Git

When we first initialized our project, the file was not being tracked by Git. To do that, we use this command git add . The period or dot that comes after add means all the files that exist in the repository. If we want to add a specific file, maybe one named about.txt, we use git add about.txt.

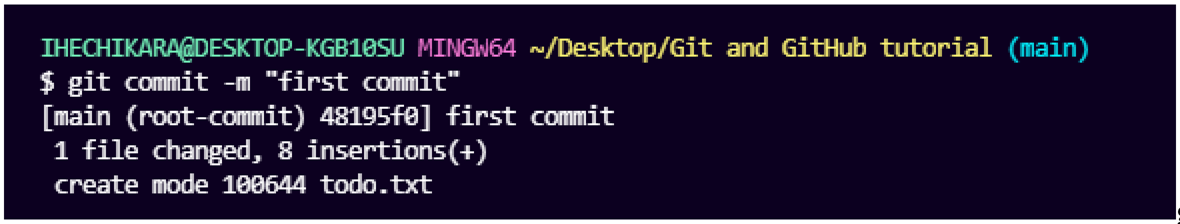
Now our file is in the staged state. we will not get a response after this command, but to know what state wer file is in, we can run the git status command.

How to commit files in Git

The next state for a file after the staged state is the committed state. To commit our file, we use the git commit -m "first commit" command.

The first part of the command git commit tells Git that all the files staged are ready to be committed so it is time to take a snapshot. The second part -m "first commit" is the commit message. -m is shorthand for message while the text inside the parenthesis is the commit message.

After executing this command, we should get a response similar to this:



```
IHECHIKARA@DESKTOP-KGB10SU MINGW64 ~/Desktop/Git and GitHub tutorial (main)
$ git commit -m "first commit"
[main (root-commit) 48195f0] first commit
 1 file changed, 8 insertions(+)
 create mode 100644 todo.txt
```

git commit

Now our file is in the committed state.

Step 4 – Push the repository to GitHub

After we create the repo, we should be redirected to a page that tells we how to create a repo locally or push an existing one.

In our case, the project already exists locally so we will use commands in the "…or push an existing repository from the command line" section. These are the commands:

git remote add origin https://github.com/ihechikara/git-and-github-tutorial.git

git branch -M main

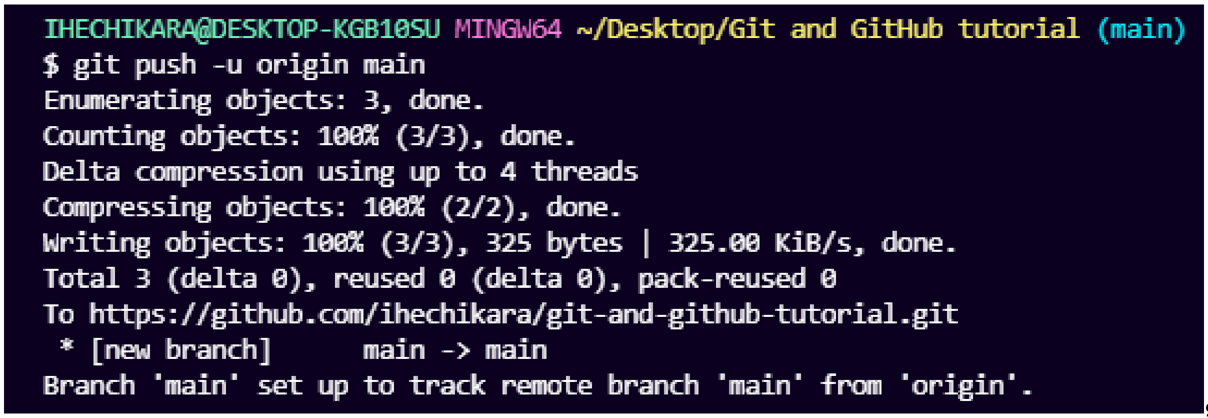
git push -u origin main

The first command git remote add origin https://github.com/ihechikara/git-and-github-tutorial.git creates a connection between wer local repo and the remote repo on Github.

The URL for wer remote project should be entirely different from the one above. So to follow along, make sure we are following the steps and working with wer own remote repo. we won't usually get a response after executing this command but make sure we have an internet connection.

The second command git branch -M main changes wer main branch's name to "main". The default branch might be created as "master", but "main" is the standard name for this repo now. There is usually no response here.

The last command git push -u origin main pushes wer repo from wer local device to GitHub. we should get a response similar to this:



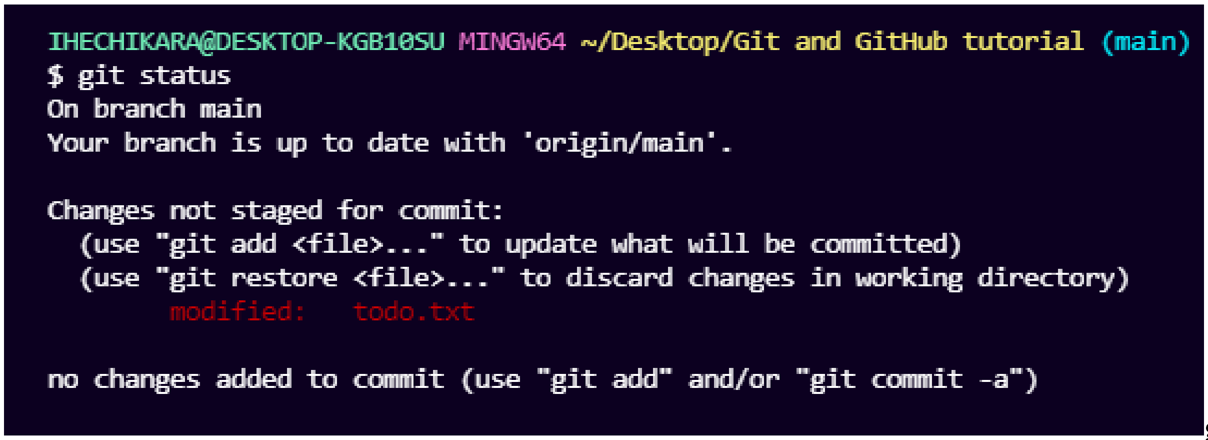
```
IHECHIKARA@DESKTOP-KGB10SU MINGW64 ~/Desktop/Git and GitHub tutorial (main)
$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 325 bytes | 325.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/ihechikara/git-and-github-tutorial.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

git push

To help we deepen wer understanding of file stages, I will make changes to the file and then push the new version to GitHub.

Recall that our file is now in the committed state. Let's make changes to the file and take note of the states.

After adding the new task, run the git status command. This is what we should see:



```
IHECHIKARA@DESKTOP-KGB10SU MINGW64 ~/Desktop/Git and GitHub tutorial (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   todo.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

git status

After making changes to the file, it moved to the modified state – but it's not yet staged for commit, so we can't push it to GitHub yet. Git has not taken a final snapshot of this current state as it's only comparing the changes we have made now with the last snapshot.

Now we are going to add (stage) this file and then commit and push it. This is the same as in the last section.

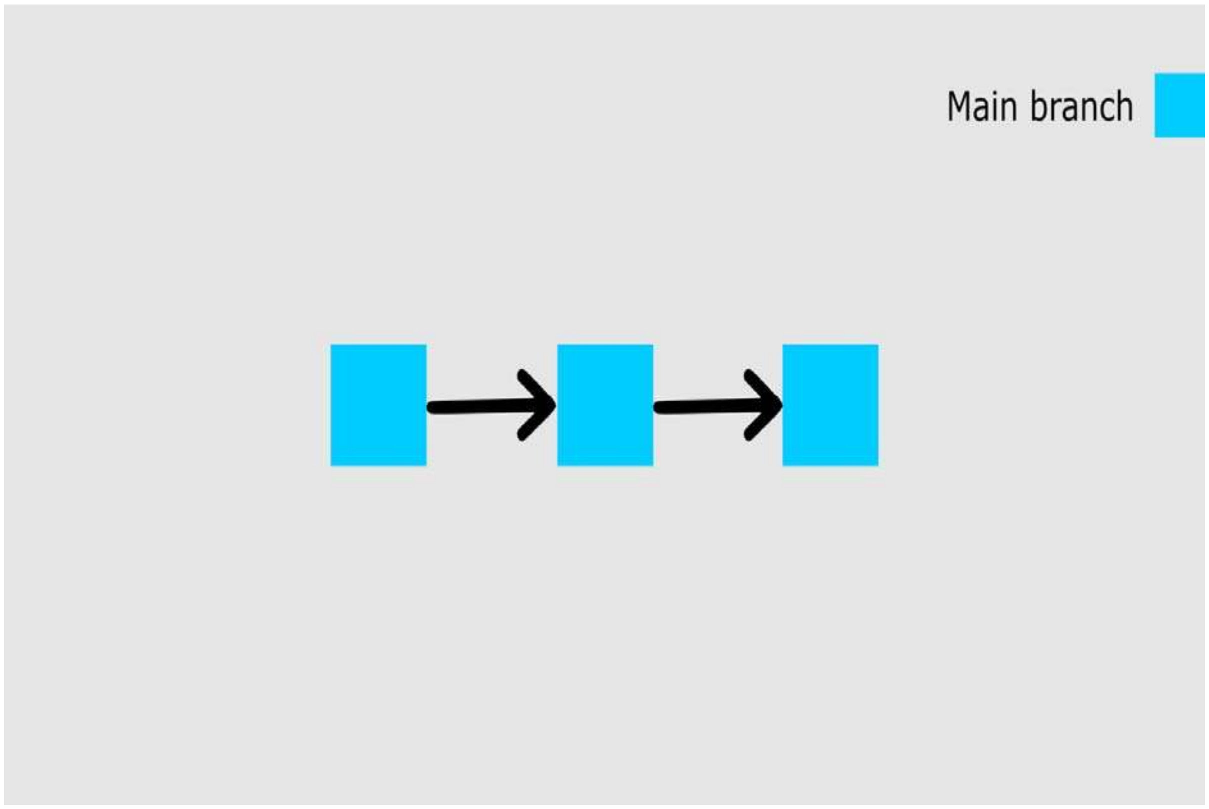
We first add the file by using git add . which adds all the files in the folder (one file in our case). Then we commit the file by running git commit -m "added new task" followed by git push -u origin main.

Those are the three steps to pushing wer modified files to GitHub. we add, commit, and then push. I hope we now understand file stages and the commands associated with them.

How to Use Branches in Git

With branches, we can create a copy of a file we would like to work on without messing up the original copy. we can either merge these changes to the original copy or just let the branch remain independent.

Before we go into using branches, I want to show we a visual representation of our repo which looks like this:



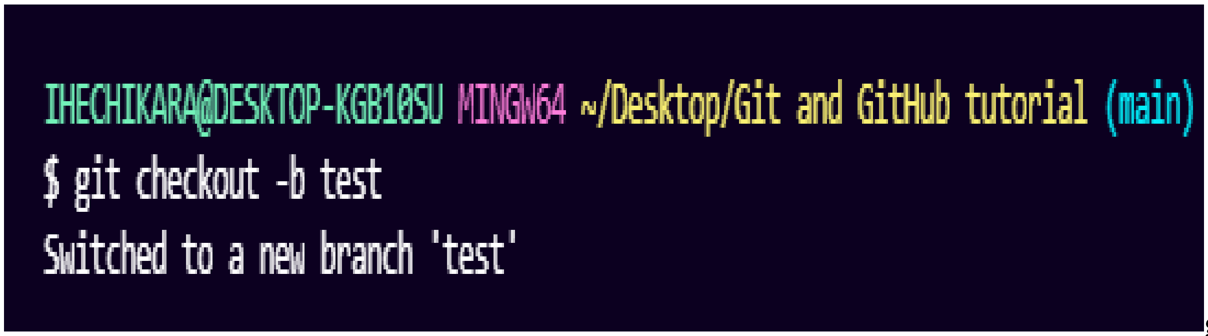
main banch

The image above shows our main branch with the last two commits (the first commit and the added new task commit).

At this point, I want to add more tasks to the list but I am not yet sure whether I want them on my main list. So I will create a new branch called test to see what my list would look like with more tasks included.

To create a new branch, run this command: git checkout -b test. I will break it down.

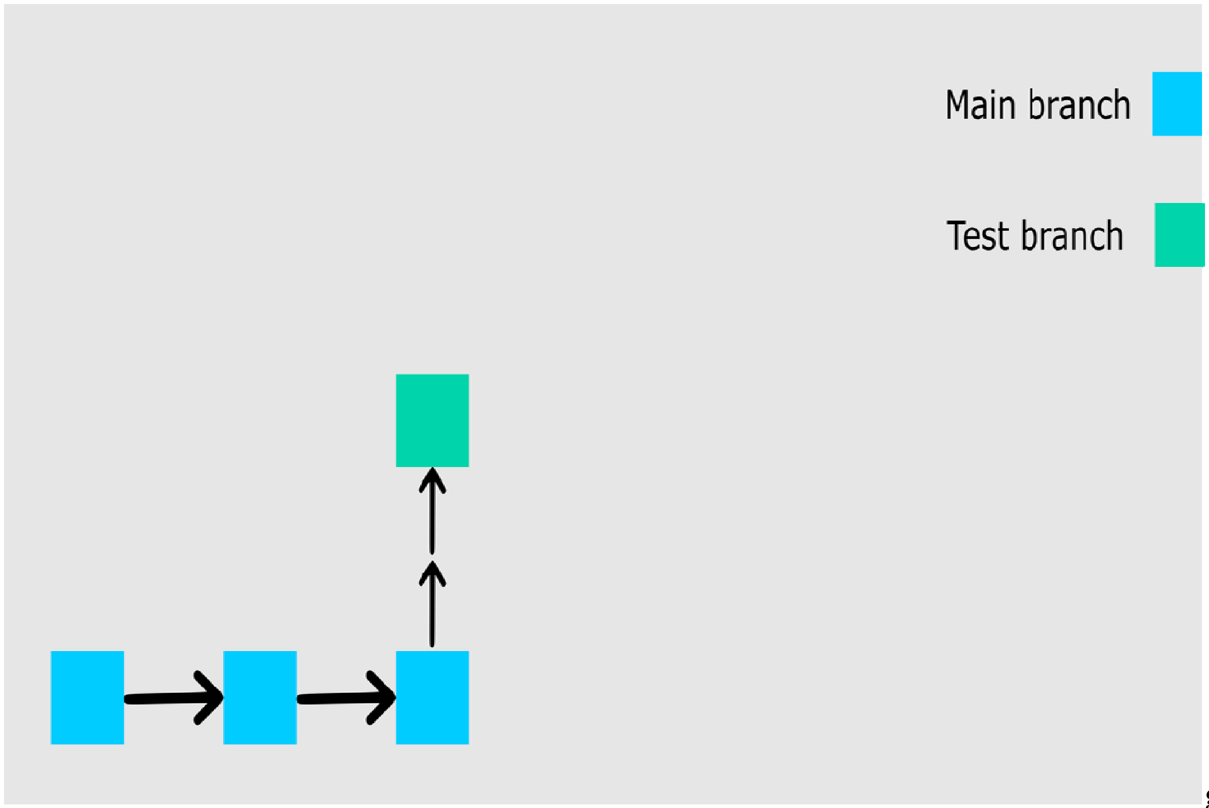
checkout tells Git it is supposed to switch to a new branch. -b tells Git to create a new branch. test is the name of the branch to be created and switched to. Here is the response we should get:



```
IHECHIKARA@DESKTOP-KGB10SU MINGW64 ~/Desktop/Git and GitHub tutorial (main)
$ git checkout -b test
Switched to a new branch 'test'
```

git checkout -b

Now that we have a new branch created, this is what our repo will look like:



git branch

We created the new branch from the state of our last commit. Let's now add more tasks to this new branch.

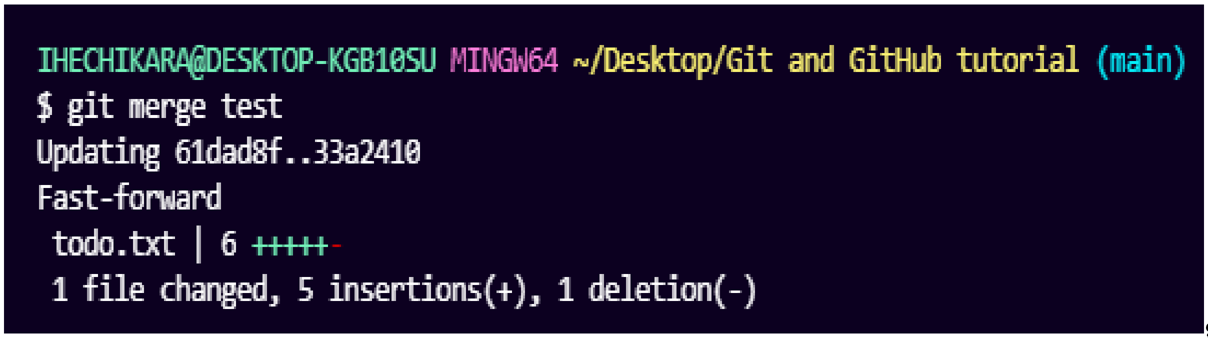
I have added four new tasks. To merge the new state to the main branch, we have to first stage and commit this branch. I will not go into details about how to do this as we did it twice in the last section.

we should try doing it werself so we understand how it works. As a hint, add the file and then commit with a message (refer to the previous section for details showing we how to do that).

After committing wer test branch, switch back to the main branch by running this command: git checkout main.

Did we notice that we did not add -b ? This is because we are not creating a new branch but rather switching to an existing one. we can check all the branches that exist in wer repo by running the git branch command.

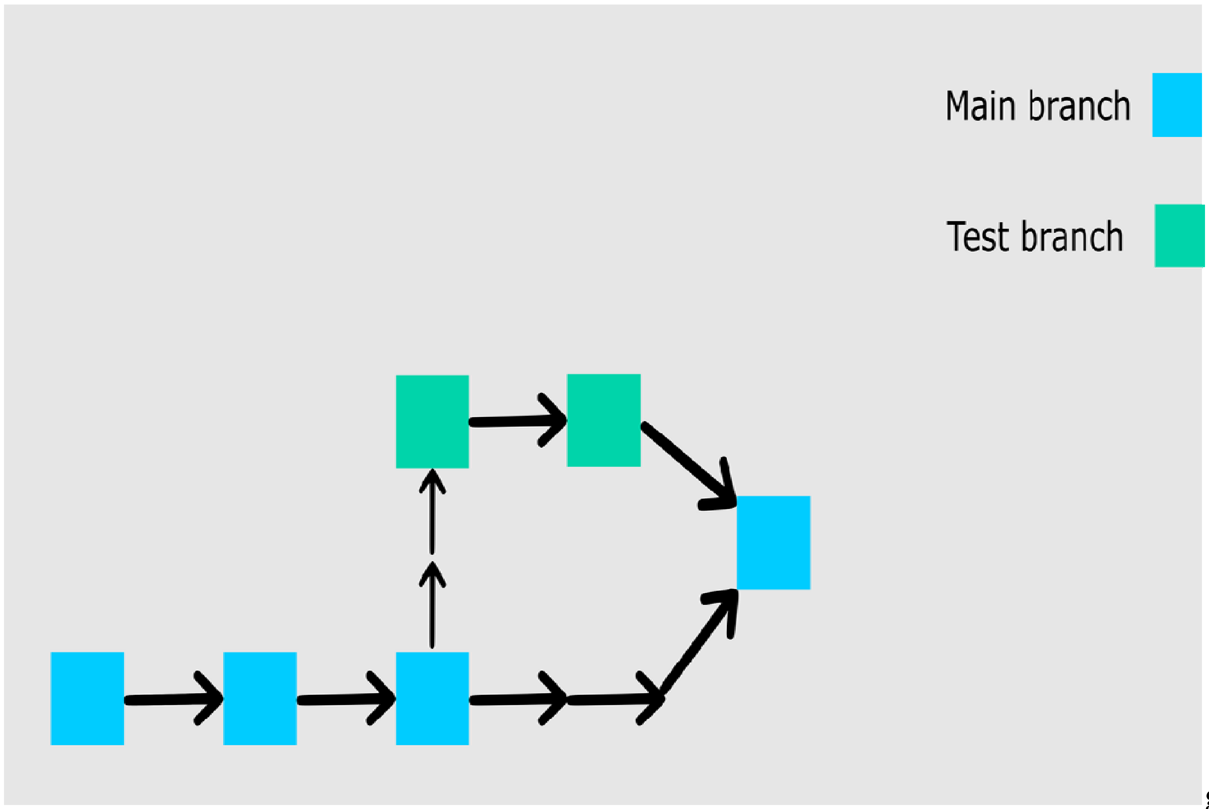
Now we can merge the changes we made in the test branch into the main branch by running git merge test. At this point, we will see all the changes made in the test branch reflected on the main branch. we should also receive a response similar to this:



```
IHECHIKARA@DESKTOP-KGB10SU MINGW64 ~/Desktop/Git and GitHub tutorial (main)
$ git merge test
Updating 61dad8f..33a2410
Fast-forward
 todo.txt | 6 +++++-
 1 file changed, 5 insertions(+), 1 deletion(-)
```

git merge

Here is a visual representation of our repo:



git merge

If we go on to push wer repo to GitHub, we'll see that the test branch will not be pushed. It will only remain in wer local repo. If we would like to push wer test branch, switch to the branch using git checkout test and then run git push -u origin test.

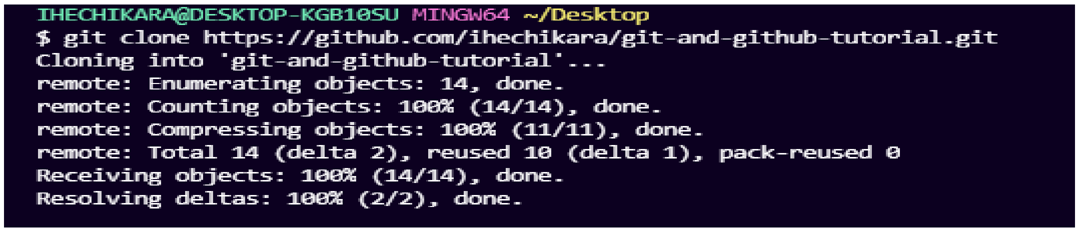
How to Pull a Repository in Git

To pull in Git means to clone a remote repository's current state into wer computer/repository. This comes in handy when we want to work on wer repo from a different computer or when we are contributing to an open source project online.

To test this, don't worry about switching to a new computer. Just run cd .. to leave the current directory and go back one step. In my own case, I have navigated back to my desktop.

Go to GitHub, and on wer repository's main page we should see a green button that says "Code". When we click on the button, we should see some options in a dropdown menu. Go on and copy the HTTPS URL.

After that, run git clone weR_HTTPS_URL. This command pulls the remote repository into wer local computer in a folder called git-and-git-tutorial. That is:



```
IHECHIKARA@DESKTOP-KGB10SU MINGW64 ~/Desktop
$ git clone https://github.com/ihechikara/git-and-github-tutorial.git
Cloning into 'git-and-github-tutorial'...
remote: Enumerating objects: 14, done.
remote: Counting objects: 100% (14/14), done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 14 (delta 2), reused 10 (delta 1), pack-reused 0
Receiving objects: 100% (14/14), done.
Resolving deltas: 100% (2/2), done.
```

git clone

5)Configure a static code analyzer 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.

static code analysis:It helps us to ensure the overall code quality, fix bugs in the early stage of development, and ensure that each developer is using the same coding standards when writing the code.

There are three basic tools that we are going to use for our static code analysis: CheckStyle, Findbugs, PMD.

CheckStyle

CheckStyle is a tool that helps programmers write code that aligns with already agreed upon coding standards. It automatically checks if the code adheres to the coding standards used on a project.

FindBugs

Fine people of the University of Maryland built this fantastic tool for us. What it basically does for us is, of course, find bugs in our code. FindBugs analyses our code and generates a report, giving us a list of all the bugs that could cause a program to misbehave. One of the good examples of the bugs that could be detected are infinite loops, unused variables, security, threading issues, and many more.

PMD

PMD is another useful tool in our static code analyzers toolbox. Beside reporting many coding issues (e.g. possible memory leaks), PMD can check if our code was commented properly if our variables are named properly and our method contains more than the specified number of lines. PMD is highly configurable and the latest releases play quite well with Lombok annotations. Previously, we needed to define custom excluded rules in order for PMD to play nice with Lombok.

Copy/Paste Detector (CPD) is an integrated part of PMD and is used to detect duplicated code in a source code.

Setting Up Static Code Analysis

We are going to use Gradle as our build tool. Gradle already has a set of plugins that we are going to use for our static code analysis.

Static analysis setup files will reside in ./gradle/static-code-analysis folder. Each of our static analysis tools will have its own dedicated folder where we will hold additional configuration files for each of the tools. Finally, we'll use staticCodeAnalysis.gradle to aggregate all our static code analysis settings. It will contain all the configuration settings needed to run static code analysis for our project.

```
buildscript {

  repositories {

    mavenCentral()

  }

  dependencies {

    classpath 'de.aaschmid:gradle-cpd-plugin:1.1'

  }

}
apply plugin: 'checkstyle'

apply plugin: 'findbugs'

apply plugin: 'pmd'

apply plugin: de.aaschmid.gradle.plugins.cpd.CpdPlugin
```

Ok, let's jump into setting the static code analysis.

Once the plugins are included in the build script, we can start configuring each of the plugins. First, we are going to configure the CheckStyle plugin.

Setting CheckStyle

For CheckStyle, we are going to set the ignoreFailures flag, toolVersion, and configFile, which will point to the location of a configuration file. As a base for our configuration file, we are going to use Google Code Style settings. The configurations are basically the same — the only difference is that we are going to use four space instead of two space indentation. And, that's it. Nothing more needs to be done for CheckStyle. Let's set up the FindBugs next:

```
checkstyle {
    toolVersion = '8.12'
    ignoreFailures = false
    configFile = file("${rootGradleDir}/static-code-analysis/checkstyle/checkstyle.xml")
}
```

We are explicitly setting the plugin not to ignore failures i.e. ignoreFailures flag is set to false. What that basically means is that our project build will fail if we run into any issue during our static code analysis check. If we think about it, this has a lot of sense. Our CI/CD pipeline should fail in case we run upon any issue in our code base. Being it compile failure, unit test failure, code analysis, as long as we have an issue, we shouldn't be able to continue with our pipeline.

Setting FindBugs

In most cases, we should specify only toolVersion and ignoreFailures. There are other options we could set here, such as specifying which bug detectors are going to be run or to include/exclude lists of files that are going to be checked in our code base. For our example, we will leave the default values here: all default bug detectors will be run, and we are not going to exclude any file from FindBugs detection.

```
findbugs {
    toolVersion = '3.0.1'
    ignoreFailures = false
}
```

Setting PMD

For PMD, besides toolVersion and ignoreFailures, we are going to set the rule sets for our code analysis. We have can set the rule sets in two ways. We can specify them directly inside the PMD plugin configuration using ruleSets array, or we could extract the rule sets to separate the XML file and reference the file using the ruleSetFiles configuration parameter. We are going to choose the latter option since it is more descriptive and allows us to provide exclusions to default rule sets categories. For the codestyle category, we are excluding DefaultPackage and OnlyOneReturn rules. we can check out ruleset.xml for full setup.

```
pmd {
    toolVersion = '6.7.0'
    ignoreFailures = false
    ruleSetFiles = files("${rootGradleDir}/static-code-analysis/pmd/ruleset.xml")
    ruleSets = []
    rulePriority = 3
}
```

Setting CPD

For Copy/Paste bug detection, we need to configure the CPD plugin. First, let's set the minimumTokenCount to 100. This means that the plugin will detect a duplicate code bug if it finds around 5– 10 lines of the same code in separate places. If only four lines of code are matched, the bug will not be detected. One useful option — especially if we are using frameworks — is to set the ignoreAnnotations to true. It will allow us to ignore "false positives" and ignore cases where classes or methods have the same 5– 6 lines of annotations. Finally, we'll enable and generate XML by setting xml.enabled to true.

```
cpd {
    language = 'java'
    toolVersion = '6.0.0'
    minimumTokenCount = 100// approximately 5-10 lines
}

cpdCheck {
    reports {
        text.enabled = false
        xml.enabled = true
    }
    ignoreAnnotations = true
    source = sourceSets.main.allJava
}
```

view raw

For remaining static analysis report plugins, we will enable generation of the HTML report instead of XML one.

```
tasks.withType(Checkstyle) {
    reports {
        xml.enabled false
        html.enabled true
    }
}
tasks.withType(FindBugs) {
    reports {
        xml.enabled false
        html.enabled true
    }
}
tasks.withType(Pmd) {
    reports {
        xml.enabled false
        html.enabled true
    }
}
```

Great! We are done with the static analysis code configuration. Now, we just need to include `staticCodeAnalysis.gradle` into our Gradle build script:

```
apply from: "${rootGradleDir}/staticCodeAnalysis.gradle"
```

Running Static Code Analysis

Static code analysis plugins will run with the same Java version used to run Gradle.

Each plugin will add its own dependencies to the Java plugin `check` task (e.g. pmdMain, cpdMain). Whenever we run `./gradlew clean build`, the internally `check` task will be triggered and static analysis steps will be run for our project. If any of the code analysis steps fail, our build will fail as well. Static code analysis reports will be generated under `./build/reports`.

If in some situations we need to "loose" the specified static code rules, we can always suppress static analysis errors by using `@SuppressWarnings` annotation. In order to suppress the warning for having too many methods in a class, we could put `@SuppressWargning("PMD.TooManyMethods")` on the given class.

We advise keeping static analysis "on" for the test classes as well. We should always treat tests as an integrated part of our project. Test code should conform to the same styles/rules we use throughout our project.

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.

What is Maven

Maven is a powerful project management tool that is based on POM (project object model). It is used for projects build, dependency and documentation. It simplifies the build process like ANT. But it is too much advanced than ANT.
In short terms we can tell maven is a tool that can be used for building and managing any Java-based project. maven make the day-to-day work of Java developers easier and generally help with the comprehension of any Java-based project.

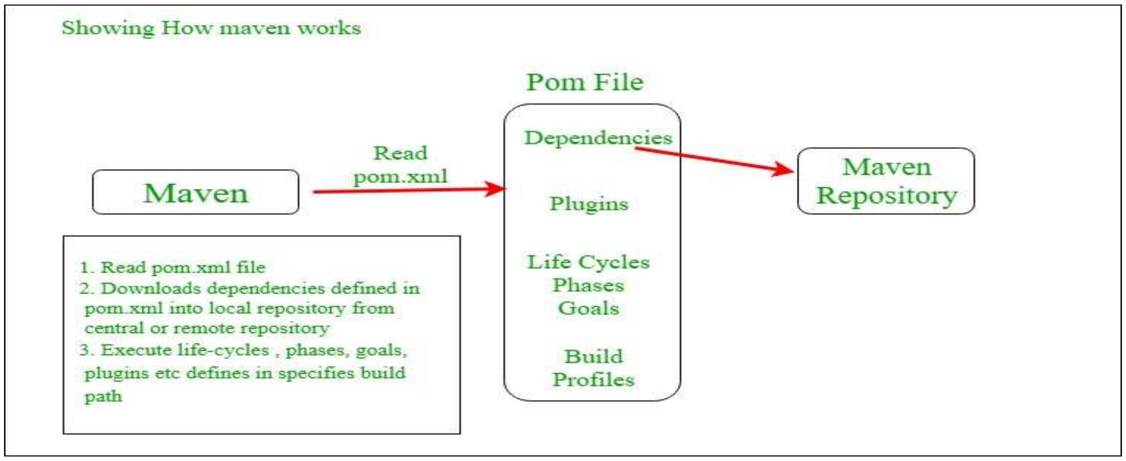
What maven does

Maven does a lot of helpful task like

1. We can easily build a project using maven.
2. We can add jars and other dependencies of the project easily using the help of maven.
3. Maven provides project information (log document, dependency list, unit test reports etc.)
4. Maven is very helpful for a project while updating central repository of JARs and other dependencies.

5. With the help of Maven we can build any number of projects into output types like the JAR, WAR etc without doing any scripting.
6. Using maven we can easily integrate our project with source control system (such as Subversion or Git).

How maven works?


Showing How maven works
Pom File
Maven
Read pom.xml
Dependencies
Plugins
Life Cycles
Phases
Goals
Build Profiles
Maven Repository
1. Read pom.xml file
2. Downloads dependencies defined in pom.xml into local repository from central or remote repository
3. Execute life-cycles , phases, goals, plugins etc defines in specifies build path

Core Concepts of Maven:

1. **POM Files:** Project Object Model(POM) Files are XML file that contains information related to the project and configuration information such as dependencies, source directory, plugin, goals etc. used by Maven to build the project. When you should execute a maven command you give maven a POM file to execute the commands. Maven reads pom.xml file to accomplish its configuration and operations.
2. **Dependencies and Repositories:** Dependencies are external Java libraries required for Project and repositories are directories of packaged JAR files. The local repository is just a directory on your machine hard drive. If the dependencies are not found in the local Maven repository, Maven downloads them from a central Maven repository and puts them in your local repository.
3. **Build Life Cycles, Phases and Goals:** A build life cycle consists of a sequence of build phases, and each build phase consists of a sequence of goals. Maven command is the name of a build lifecycle, phase or goal. If a lifecycle is requested executed by giving maven command, all build phases in that life cycle are executed also. If a build phase is requested executed, all build phases before it in the defined sequence are executed too.
4. **Build Profiles:** Build profiles a set of configuration values which allows you to build your project using different configurations. For example, you may need to build your project for your local computer, for development and test. To enable different builds you can add different build profiles to your POM files using its profiles elements and are triggered in the variety of ways.

5. **Build Plugins:** Build plugins are used to perform specific goal. you can add a plugin to the POM file. Maven has some standard plugins you can use, and you can also implement your own in Java.

Installation process of Maven

The installation of Maven includes following Steps:

1. Verify that your system has java installed or not. if not then install java (Link for Java Installation)
2. Check java Environmental variable is set or not. if not then set java environmental variable.(link to install java and setting environmental variable)
3. Download maven (Link)
4. Unpack your maven zip at any place in your system.
5. Add the bin directory of the created directory apache-maven-3.5.3(it depends upon your installation version) to the PATH environment variable and system variable.
6. open cmd and run **mvn -v** command. If it print following lines of code then installation completed.

```
Apache Maven 3.5.3 (3383c37e1f9e9b3bc3df5050c29c8aff9f295297; 2018-02-
25T01:19:05+05:30)
Maven home: C:\apache-maven-3.5.3\bin\..
Java version: 1.8.0_151, vendor: Oracle Corporation
Java home: C:\Program Files\Java\jdk1.8.0_151\jre
Default locale: en_US, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

Maven pom.xml file

POM means Project Object Model is key to operate Maven. Maven reads pom.xml file to accomplish its configuration and operations. It is an XML file that contains information related to the project and configuration information such as **dependencies**, **source directory**, **plugin**, **goals etc**. used by Maven to build the project.
The sample of pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"

  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
     <modelVersion>4.0.0</modelVersion>
     <groupId> com.project.loggerapi </groupId>
     <artifactId>LoggerApi</artifactId>
     <version>0.0.1-SNAPSHOT</version>
    <!-- Add typical dependencies for a web application -->
    <dependencies>
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-api</artifactId>
            <version>2.11.0</version>
         </dependency>
    </dependencies>
  </project>
```

Elements used for Creating pom.xml file

1. **project-** It is the root element of the pom.xml file.
2. **modelVersion-** modelversion means what version of the POM model you are using. Use version 4.0.0 for maven 2 and maven 3.
3. **groupId-** groupId means the id for the project group. It is unique and Most often you will use a group ID which is similar to the root Java package name of the project like we used the groupId com.project.loggerapi.
4. **artifactId-** artifactId used to give name of the project you are building.in our example name of our project is LoggerApi.

5. **version-** version element contains the version number of the project. If your project has been released in different versions then it is useful to give version of your project.

Other Elements of Pom.xml file

1. **dependencies-** dependencies element is used to defines a list of dependency of project.
2. **dependency-** dependency defines a dependency and used inside dependencies tag. Each dependency is described by its groupId, artifactId and version.
3. **name-** this element is used to give name to our maven project.
4. **scope-** this element used to define scope for this maven project that can be compile, runtime, test, provided system etc.
5. **packaging-** packaging element is used to packaging our project to output types like JAR, WAR etc.

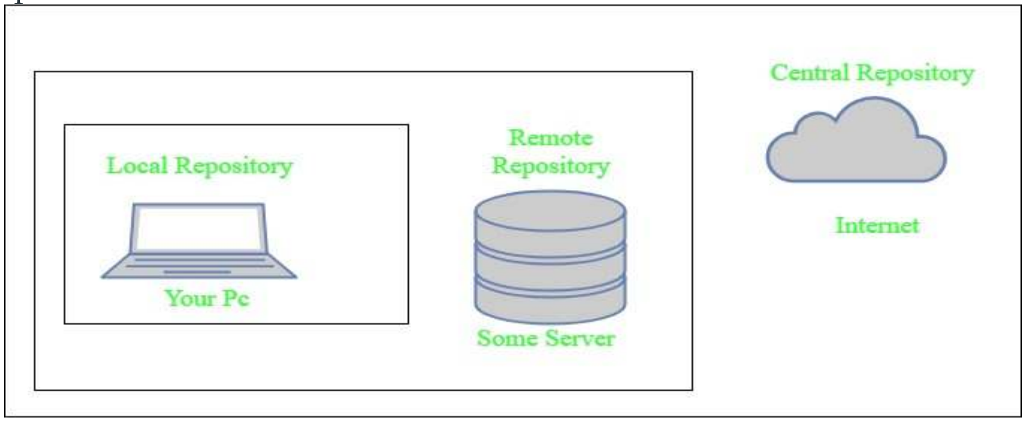
Maven Repository

Maven repositories are directories of packaged JAR files with some metadata. The metadata are POM files related to the projects each packaged JAR file belongs to, including what external dependencies each packaged JAR has. This metadata enables Maven to download dependencies of your dependencies recursively until all dependencies are download and put into your local machine.

Maven has three types of repository :

1. **Local repository**
2. **Central repository**
3. **Remote repository**

Maven searches for dependencies in this repositories. First maven searches in Local repository then Central repository then Remote repository if Remote repository specified in the POM.



1. **Local repository-** A local repository is a directory on the machine of developer. This repository contains all the dependencies Maven downloads. Maven only needs to download the dependencies once, even if multiple projects depends on them

(e.g. ODBC).
By default, maven local repository is user_home/m2 directory.
example – **C:\Users\asingh\.m2**

2. **Central repository-** The central Maven repository is created Maven community. Maven looks in this central repository for any dependencies needed but not found in your local repository. Maven then downloads these dependencies into your local repository.
3. **Remote repository-** remote repository is a repository on a web server from which Maven can download dependencies.it often used for hosting projects internal to organization. Maven then downloads these dependencies into your local repository.

Practical Application Of Maven

When working on a java project and that project contains a lot of dependencies, builds, requirement, then handling all those things manually is very difficult and tiresome. Thus using some tool which can do these works is very helpful.
Maven is such a *build management tool* which can do all the things like adding dependencies, managing the classpath to project, generating war and jar file automatically and many other things.

Pros and Cons of using Maven

Pros:

1. Maven can add all the dependencies required for the project automatically by reading pom file.
2. One can easily build their project to jar, war etc. as per their requirements using Maven.
3. Maven makes easy to start project in different environments and one doesn't needs to handle the dependencies injection, builds, processing, etc.
4. Adding a new dependency is very easy. One has to just write the dependency code in pom file.

Cons:

5. Maven needs the maven installation in the system for working and maven plugin for the ide.
6. If the maven code for an existing dependency is not available, then one cannot add that dependency using maven.

When should someone use Maven

One can use the Maven Build Tool in the following condition:

7. When there are a lot of dependencies for the project. Then it is easy to handle those dependencies using maven.

8. When dependency version update frequently. Then one has to only update version ID in pom file to update dependencies.
9. Continuous builds, integration, and testing can be easily handled by using maven.
10. When one needs an easy way to Generating documentation from the source code, Compiling source code, Packaging compiled code into JAR files or ZIP files.

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.

Jenkins has the most optimal product community and set of really useful plugins that suits most of your software projects: you can build software, deploy software, websites, portals to various places (e.g including AWS, DigitalOcean, bare metal servers) or to run unit tests. It can be integrated with communication tools of your choice, like Slack, HipChat or email.

If you haven't had a chance to try Jenkins earlier, feel free to use the tutorial below to get started.

Manual installation

In order to install Jenkins, we will need:

- Unix system. I would recommend a Debian-based machine, like Ubuntu server LTS
- Java runtime environment installed. I usually use Java 8
- Get base Jenkins setup
- Install necessary plugins
- Put everything behind your web server.

Install Java

The easiest way to install Java is through the apt-get package manager:

```
sudo apt-get install python-software-properties

sudo add-apt-repository ppa:webupd8team/java

sudo apt-get update
```

Once you added ppa above, you can install java with the following command:

```
sudo apt-get install oracle-java8-installer
```

Get base Jenkins Setup

You will need to execute a series of the following commands, namely: add the Jenkins signing key, register Jenkins apt sources, update package lists, and install Jenkins package.

```
wget -q -O - http://pkg.jenkins-ci.org/debian/jenkins-ci.org.key | sudo apt-key add -

sudo echo deb http://pkg.jenkins-ci.org/debian binary/ >
/etc/apt/sources.list.d/jenkins.list

sudo apt-get update

sudo apt-get install jenkins
```

By default, it will install the base Jenkins setup, which is insecure. You will need to go to the host where your Jenkins is installed, for example: http://jenkins-host:8080/.

Navigate to Manage Jenkins (on the left) and choose the "Configure Global Security" item on the page loaded.

Now look below on the Matrix-based security (select it, if it is not selected previously), and make sure Anonymous only has the "Read" right under the View group.

Click **Save** at the bottom of the page. After the page reloads, you'll see a login form, but simply ignore that and
go to the home page (like, for example, http://jenkins-host:8080/). You'll see this signup form, and the first signed up account will be the administrator.

The Power of Plugins

Jenkins would not be so powerful without plugins. Usually, I install these plugins by default:

- Bitbucket:
  The BitBucket plugin is designed to offer integration between BitBucket and Jenkins. BitBucket offers a Jenkins hook, but this one just triggers a build for a specific job on commit, nothing more. The BitBucket plugin, like the GitHub plugin, uses the POST hook payload to check which job has to get triggered based on the changed repository/branch.
  Plugin URL: https://wiki.jenkins-ci.org/display/JENKINS/BitBucket+Plugin
- bitbucket-pullrequest-builder:
  This plugin builds pull requests from Bitbucket.org. This is a must-have plugin if you perform QA deploy for each submitted pull request.
  Plugin URL: https://wiki.jenkins-ci.org/display/JENKINS/Bitbucket+pullrequest+builder+plugin
- build-pipeline-plugin:
  This plugin provides a Build Pipeline View of upstream and downstream connected jobs that typically form a build pipeline. In addition, it offers the ability to define manual triggers for jobs that require intervention prior to execution, e.g. an approval process outside of Jenkins. Provides nice visualization of the paths & flows.
  Plugin URL: https://wiki.jenkins-ci.org/display/JENKINS/Build+Pipeline+Plugin
- copyartifact:
  Adds a build step to copy artifacts from another project. The plugin lets you

specify which build to copy artifacts from (e.g. the last successful/stable build, by build number, or by a build parameter). You can also control the copying process by filtering the files being copied, specifying a destination directory within the target project, etc.
Plugin URL: https://wiki.jenkins-ci.org/display/JENKINS/Copy+Artifact+Plugin

- credentials:
  Adds a build step to copy artifacts from another project. The plugin lets you specify which build to copy artifacts from (e.g. the last successful/stable build, by build number, or by a build parameter). You can also control the copying process by filtering the files being copied, specifying a destination directory within the target project, etc.
  Plugin URL: https://wiki.jenkins-ci.org/display/JENKINS/Credentials+Plugin
- delivery-pipeline-plugin:
  Visualisation of Delivery/Build Pipelines, renders pipelines based on upstream/downstream jobs. When using Jenkins as a build server it is now possible with the Delivery Pipeline Plugin to visualise one or more Delivery Pipelines in the same view even in full screen.
  Plugin URL: https://wiki.jenkins-ci.org/display/JENKINS/Delivery+Pipeline+Plugin
- environment-script:
  Environment Script Plugin allows you to have a script run after SCM checkout, before the build. If the script fails (exit code isn't zero), the build is marked as failed.
  Any output on standard out is parsed as environment variables that are applied to the build. It supports "override syntax" to append paths to PATH-like variables.
  Plugin URL: https://wiki.jenkins-ci.org/display/JENKINS/Environment+Script+Plugin
- git:
  Supports popular git version control system

- ghprb:
  This plugin builds pull requests in GitHub. It's another must-have plugin if your software development life cycle includes deploying pull requests to PR environment to test.
  Plugin URL: https://wiki.jenkins-ci.org/display/JENKINS/GitHub+pull+request+builder+plugin
- greenballs: The funniest plugin - changes Jenkins to use green balls instead of blue for successful builds.
  Plugin URL: https://wiki.jenkins-ci.org/display/JENKINS/Green+Balls
- hipchat:
  This plugin allows your team to setup build notifications to be sent to HipChat rooms.To enable notifications, add "HipChat Notifications" as a post-build step.
  Plugin URL: https://wiki.jenkins-ci.org/display/JENKINS/HipChat+Plugin
- junit:
  Allows JUnit-format test results to be published. Note: number of tools, including Karma, PhpUNIT & other tools allow to publish test results in a JUnit format. Thus, this is a must-have plugin for unit test flows.
  Plugin URL: https://wiki.jenkins-ci.org/display/JENKINS/JUnit+Plugin
- matrix-auth:
  Offers matrix-based security authorization strategies (global and per-project). This is quite handy if you have shared build server across several teams.
  Plugin URL: https://wiki.jenkins-ci.org/display/JENKINS/Matrix+Authorization+Strategy+Plugin
- parameterized-trigger:
  This plugin lets you trigger new builds when your build has completed, with various ways of specifying parameters for the new build.
  You can add multiple configurations: each has a list of projects to trigger, a condition for when to trigger them (based on the result of the current build), and a parameters section.
  Plugin URL: https://wiki.jenkins-ci.org/display/JENKINS/Parameterized+Trigger+Plugin

- rebuild:
  This plays nice with the parameterized-trigger plugin. The plug-in allows the user to rebuild a parametrized build without entering the parameters again.
  Plugin URL: https://wiki.jenkins-ci.org/display/JENKINS/Rebuild+Plugin
- ssh: You can use the SSH Plugin to run shell commands on a remote machine via ssh.
  Plugin URL: https://wiki.jenkins-ci.org/display/JENKINS/SSH+plugin
- s3: Allows uploading artifacts to S3 with multiple options.
  Plugin URL: https://wiki.jenkins-ci.org/display/JENKINS/S3+Plugin
- throttle-concurrents:
  This plugin allows for throttling the number of concurrent builds of a project running per node or globally.
  Unfortunately, this is also a must-have plugin for Node (0.10-0.12) projects with NPM - two concurrent npm installs often fail.
  Plugin URL: https://wiki.jenkins-ci.org/display/JENKINS/Throttle+Concurrent+Builds+Plugin

Plugins are installed using Plugin manager on a Manage Jenkins Section.

Putting Jenkins Behind a Web Server

Usually I hide Jenkins behind nginx. A typical configuration looks like the one below:

```
server {

 listen 443 ssl;

 server_name jenkins.vagrant.dev;

 ssl_certificate /etc/nginx/jenkins_selfsigned.crt;

 ssl_certificate_key /etc/nginx/jenkins_selfsigned.key

 location / {
  proxy_pass http://127.0.0.1:8080;

  proxy_set_header Host $host;

  proxy_set_header X-Real-IP $remote_addr;

  proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

  proxy_redirect off;

  proxy_connect_timeout 150;

  proxy_send_timeout 100;

  proxy_read_timeout 100;

}}
```

Automated installation

Do I install Jenkins manually each time? Of course not, I do it often for my customers. With ansible, and sa-box-jenkins role new Jenkins installation can be deployed while you drink your coffee.

Let's prepare a basic bootstrap project that can be used by you in the future.
It includes following files:

- *bootstrap.sh* - installs ansible alongside with dependencies.
- *init.sh* - initializes 3rd party dependencies
- *.projmodules* - fully compatible with `.gitmodules` git syntax, specifies list of the dependencies
  that will be used by the playbook.
  In particular, it includes ansible- by default developer_recipes (repository with set of handy deployment recipes)

and ansible role called *sa-box-bootstrap* responsible for box securing steps (assuming you plan to put Jenkins on remote hosts).

```
[submodule "public/ansible_developer_recipes"]

  path = public/ansible_developer_recipes

  url = git@github.com:Voronenko/ansible-developer_recipes.git

[submodule "roles/sa-box-bootstrap"]

        path = roles/sa-box-bootstrap

        url = git@github.com:softasap/sa-box-bootstrap.git

[submodule "roles/sa-box-jenkins"]

        path = roles/sa-box-jenkins

        url = git@github.com:softasap/sa-box-jenkins.git
```

- *hosts* - list here the initial box credentials that were provided to you for the server. Note: jenkins-bootstrap assumes you have the fresh box with root access only. If your box already secured, adjust credentials appropriately

```
[jenkins-bootstrap]

jenkins_bootstrap ansible_ssh_host=192.168.0.17 ansible_ssh_user=yourrootuser
ansible_ssh_pass=yourpassword

[jenkins]

jenkins ansible_ssh_host=192.168.0.17 ansible_ssh_user=jenkins
```

- *jenkins_vars.yml* - set here specific environment overrides, like your preferred deploy user name and keys.

- *jenkins_bootstrap.yml* - First step - box securing. Creates jenkins user, and secures the box using sa-box-bootstrap role.
  See more details about the sa-box-bootstrap role
  In order, to override params for *sa-box-bootstrap* - pass the parameters like in the example below:

```
- hosts: all

 vars_files:

  - ./jenkins_vars.yml

 roles:

   - {

       role: "sa-box-bootstrap",

       root_dir: "{{playbook_dir}}/public/ansible_developer_recipes",

       deploy_user: "{{jenkins_user}}",

       deploy_user_keys: "{{jenkins_authorized_keys}}"

     }
```

- *jenkins.yml* provisioning script that configures jenkins with set of plugins and users.
- *jenkins_vars.yml* configuration options for jenkins deployment.
- *setup_jenkins.sh* shell script that invokes deployment in two steps: initial box bootstraping & jenkins setup

```
#!/bin/sh

ansible-playbook jenkins_bootstrap.yml --limit jenkins_bootstrap
```

```
ansible-playbook jenkins.yml --limit jenkins
```

Configuration Options for Automated Installation

You need to override:

- jenkins_authorized_keys (this is list of the keys, that allow you to login to Jenkins box under Jenkins)
- jenkins_domain - your agency domain
- jenkins_host - name of the Jenkins host (Site will be bound to jenkins_host.jenkins_domain)
- java_version - your Java choice (6,7,8 supported)

```
jenkins_user: jenkins

jenkins_authorized_keys:

  - "{{playbook_dir}}/components/files/ssh/vyacheslav.pub"

jenkins_domain: "vagrant.dev"

jenkins_host: "jenkins"

java_version: 8
```

-jenkins_users list of users with passwords to create. Admin and deploy are required users.
Admin is used to manage instance, deploy is used to access the artifacts via deployment scripts.
If you won't override passwords, the default one will be used (per role), which is not the best, for public deployments.

```
jenkins_users:

  - {
```

```
  name: "Admin",

  password: "AAAdmin",

  email: "no-reply@localhost"

  }

 - {

  name: "deploy",

  password: "DeDeDeDeploy",

  email: "no-reply@localhost"

  }
```

- jenkins_plugins Your choice of plugins to install. By default:

```
jenkins_plugins:

 - bitbucket # https://wiki.jenkins-ci.org/display/JENKINS/BitBucket+Plugin

 - bitbucket-pullrequest-builder

 - build-pipeline-plugin

 - copyartifact # https://wiki.jenkins-ci.org/display/JENKINS/Copy+Artifact+Plugin

 - credentials # https://wiki.jenkins-ci.org/display/JENKINS/Credentials+Plugin

 - delivery-pipeline-plugin # https://wiki.jenkins-
ci.org/display/JENKINS/Delivery+Pipeline+Plugin

 - environment-script # https://wiki.jenkins-
ci.org/display/JENKINS/Environment+Script+Plugin
```

```
  - git

  - ghprb # https://wiki.jenkins-
ci.org/display/JENKINS/GitHub+pull+request+builder+plugin

  - greenballs # https://wiki.jenkins-ci.org/display/JENKINS/Green+Balls

  - hipchat # https://wiki.jenkins-ci.org/display/JENKINS/HipChat+Plugin

  - junit # https://wiki.jenkins-ci.org/display/JENKINS/JUnit+Plugin

  - matrix-auth # https://wiki.jenkins-
ci.org/display/JENKINS/Matrix+Authorization+Strategy+Plugin

  - matrix-project #https://wiki.jenkins-ci.org/display/JENKINS/Matrix+Project+Plugin

  - parameterized-trigger #https://wiki.jenkins-
ci.org/display/JENKINS/Parameterized+Trigger+Plugin

  - rebuild # https://wiki.jenkins-ci.org/display/JENKINS/Rebuild+Plugin

  - ssh

  - s3 # https://wiki.jenkins-ci.org/display/JENKINS/S3+Plugin

  - throttle-concurrents #https://wiki.jenkins-
ci.org/display/JENKINS/Throttle+Concurrent+Builds+Plugin
```

The Code in Action

You can download this template code from this repository. In order to use it - fork it, adjust parameters to your needs, and use.

Running is as simple as `./setup_jenkins.sh`

Note: don't write highlighted data i.e red colour

Exercise 8:

Module name: Implementation of CICD with Java and open source stack

Configure the Jenkins pipeline to call the build script jobs and configure to run it whenever there is a change made to an application in the version control system. Make a change to the background color of the landing page of the web application and check if the configured pipeline runs.

There are six steps to building a pipeline with Jenkins. But, before you begin those six steps, make sure you have the following in your system.

- Java Development Kit
- Knowledge to execute some basic Linux commands

The steps to build CI/CD pipeline with Jenkins are:

1. Download Jenkins

- Download Jenkins from the Jenkins downloads page 'https://www.jenkins.io/download/'.
- Download the file 'Generic Java package (.war)'.

2. Execute Jenkins as a Java binary

- Open the terminal window and enter cd <your path>.
- Use the command java –jar ./Jenkins. war to run the WAR file.

3. Create a Jenkins Job

- Open the web browser and open localhost:8080.
- The Jenkins dashboard opens creates new jobs there.

4. Create a Pipeline Job

- Select and define what Jenkins job that is to be created.
- Select Pipeline, give it a name and click OK.

- Scroll down and find the pipeline section.
- Either directly write a pipeline script or retrieve the Jenkins file from SCM (Source Code Management).

5. Configure and Execute a Pipeline Job With a Direct Script

- Choose Pipeline script as the Destination and paste the Jenkins file content in the Script from the GitHub.
- Click on Save to keep the changes.
- Now click on the Build Now to process the build.
- To check the output, click on any stage and click Log; a message will appear on the screen.

6. Configure and Execute a Pipeline With SCM

- Copy the GitHub repository URL by clicking on Clone or download.
- Now, click on Configure to modify the existing job.
- Scroll to the Advanced Project Options setting and select Pipeline script from the SCM option.
- Paste the GitHub repository URL here.
- Type Jenkinsfile in the Script, and then click on the Save button.
- Next, click on Build Now to execute the job again.
- There will be an additional stage, in this case, i.e., Declaration: Checkout SCM.
- Click on any stage and click on Log.

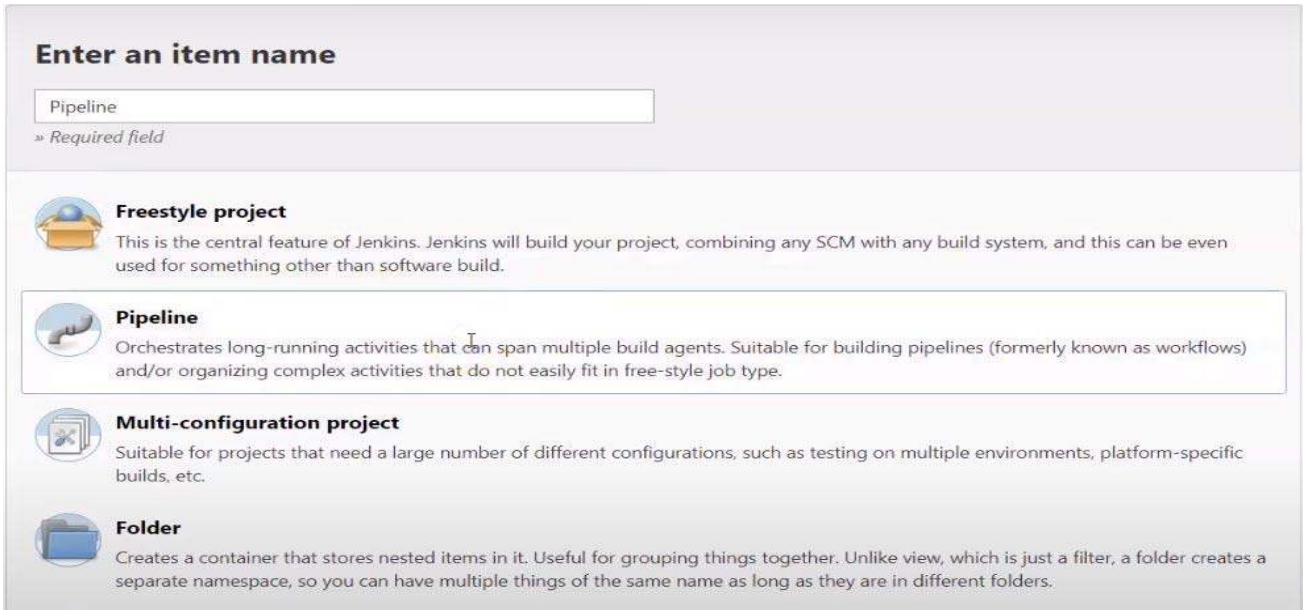
After you have grasped all the essential steps to build a CI/CD pipeline using Jenkins, a hands-on demonstration will serve as the icing on the cake.

Demo - To Build a CI/CD Pipeline With Jenkins

Go to your Jenkins Portal:

- Click on 'Create a job'.
- In the item name dialog box, you may enter the 'pipeline'.
- Select the pipeline job type in the list below.

- Click on OK.



Enter an item name

Pipeline

» 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.

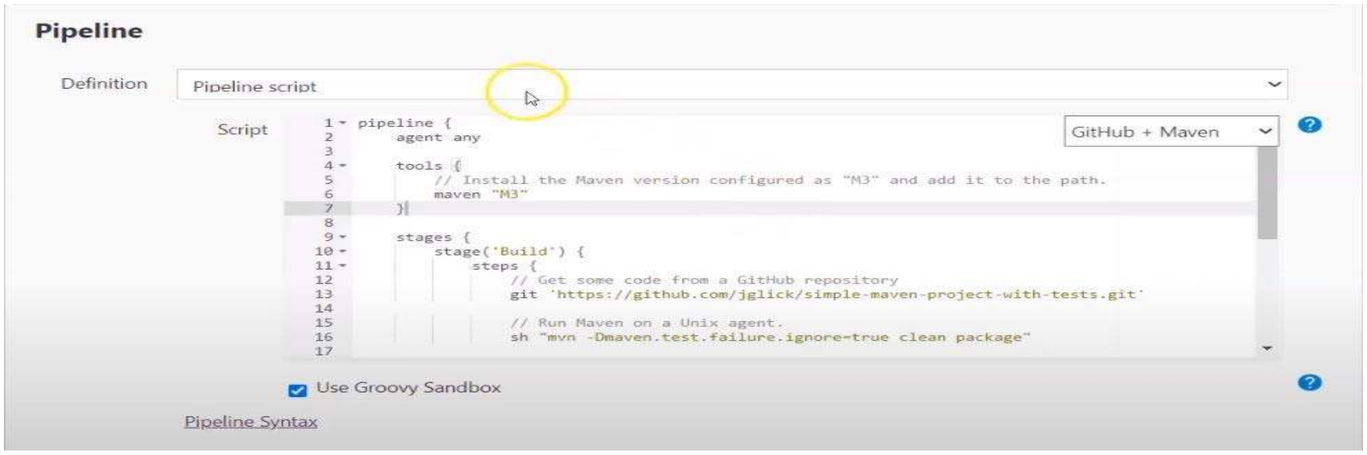
Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

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

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.

A configuration related to the pipeline opens on the screen.

- Scroll down on that page.
- There in the dialog box, choose GitHub+Maven.



Pipeline

Definition: Pipeline script

Script: GitHub + Maven

```
pipeline {
    agent any

    tools {
        // Install the Maven version configured as "M3" and add it to the path.
        maven "M3"
    }

    stages {
        stage('Build') {
            steps {
                // Get some code from a GitHub repository
                git 'https://github.com/jglick/simple-maven-project-with-tests.git'

                // Run Maven on a Unix agent.
                sh "mvn -Dmaven.test.failure.ignore=true clean package"

```

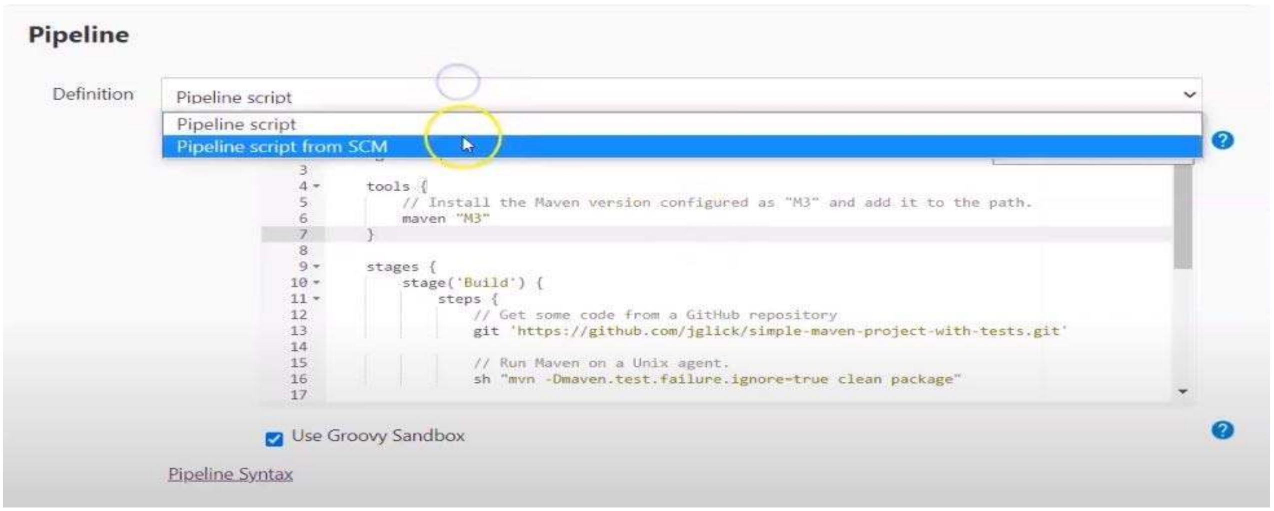
Use Groovy Sandbox: ✓

Pipeline Syntax

Some steps will appear on the screen. The next step is to integrate the Jenkins file into the Version Control system.

So, to do that, you must:

- Select 'Pipeline script from SCM'.



Pipeline

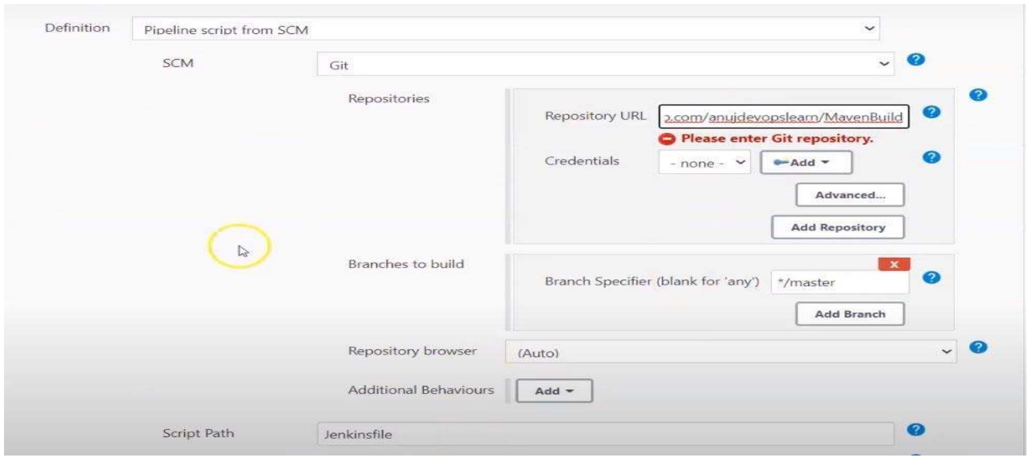
Definition: Pipeline script

```

    tools {
        // Install the Maven version configured as "M3" and add it to the path.
        maven "M3"
    }

    stages {
        stage('Build') {
            steps {
                // Get some code from a GitHub repository
                git 'https://github.com/jglick/simple-maven-project-with-tests.git'

                // Run Maven on a Unix agent.
                sh "mvn -Dmaven.test.failure.ignore=true clean package"

```

☑ Use Groovy Sandbox

Pipeline Syntax

- Then in the SCM dialog box, select Git.
- 'Jenkins file' is the name of the Script.
- Add the Git repository URL.
- You can add the credentials if any.



Definition: Pipeline script from SCM

SCM: Git

Repositories

Repository URL: ›.com/anujdevopslearn/MavenBuild

Please enter Git repository.

Credentials: - none -

Branches to build

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

Repository browser: (Auto)

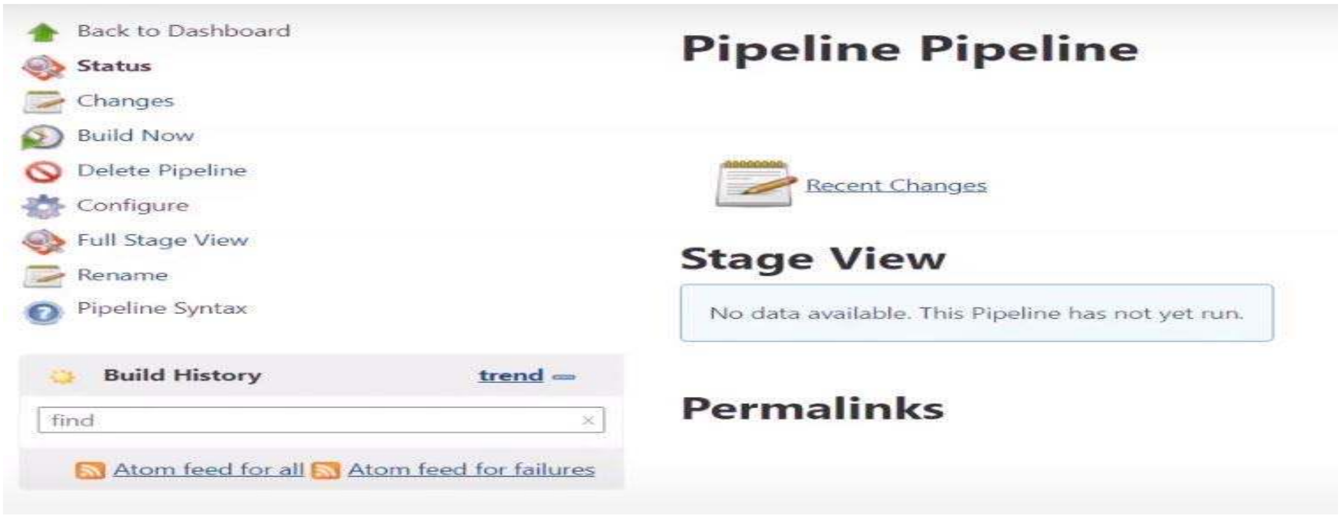
Additional Behaviours

Script Path: Jenkinsfile

The credentials can be added with the help of the 'Add' option.

- Then save the configuration

A page now appears on the screen that gives you various options like 'Build Now', 'Delete Pipeline', 'Configure', etc.



Pipeline Pipeline

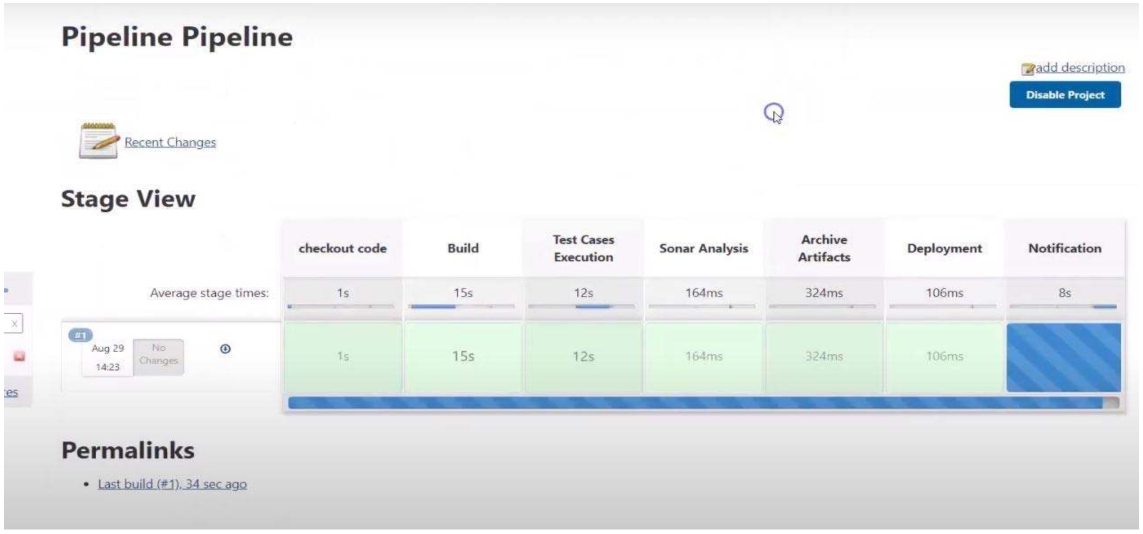
Stage View

No data available. This Pipeline has not yet run.

Permalinks

- Click on the Build Now option.

The pipeline will start downloading. The checkout will be visible on the screen and you can see the build being complete on the screen.



Pipeline Pipeline

Stage View

	checkout code	Build	Test Cases Execution	Sonar Analysis	Archive Artifacts	Deployment	Notification
Average stage times:	1s	15s	12s	164ms	324ms	106ms	8s
#1 Aug 29 14:23 No Changes	1s	15s	12s	164ms	324ms	106ms	

Permalinks

- Last build (#1), 34 sec ago

You can go to the console output option to check the log that is taking place.

You will soon be able to see that all the segments of the pipeline are completed. The artifact will be present to download. The war file can be downloaded using that link.

The entire process helps us understand how the whole Pipeline is configured. Using similar types of steps, different kinds of automation pipelines can be configured.

Using CI/ CD Pipelines and IndexedDB for a jQuery UI Web Application

Using a simple PostgreSQL database, we'll be storing some client-side data in IndexedDB storage. This technique can be a convenient and robust solution for storing some data in the browser.

Using IndexedDB is a much better alternative to the Document Object Model (DOM) for storing and querying client-side data because of how easy it is to manage and query data. Let's get started with our project.

Creating a New Project

First, create a new directory for our project in a new directory and run this command:

git init

You should see something like the following:

$ mkdir jquery-ui-pipeline-demo $ cd jquery-ui-pipeline-demo $ bin/pipeline create

The "Pipeline create" command lets you specify the template where your code will be built and tested. This method is like how a knitter uses yarn but in our case, we're using the web toolbelt instead of the yarn.

The only difference is that you won't need to use your own build environment to run tests. The web toolbelt will do this for you automatically.

That's all you need to do to create a new project.

Building a jQuery UI Web App Using a Pipeline

To use the pipeline to build your sample app, we will open a new terminal window and type:

$ bin/pipeline build

This command will take a while because it's building and testing the app on our local development machine.

After the build process is complete, you can see that the completed project is stored in our index.html file. We can use these files in other apps or even deploy the app to a server to test it out.

Exercise 9:

Building CI CD Pipeline Using Jenkins and dockers

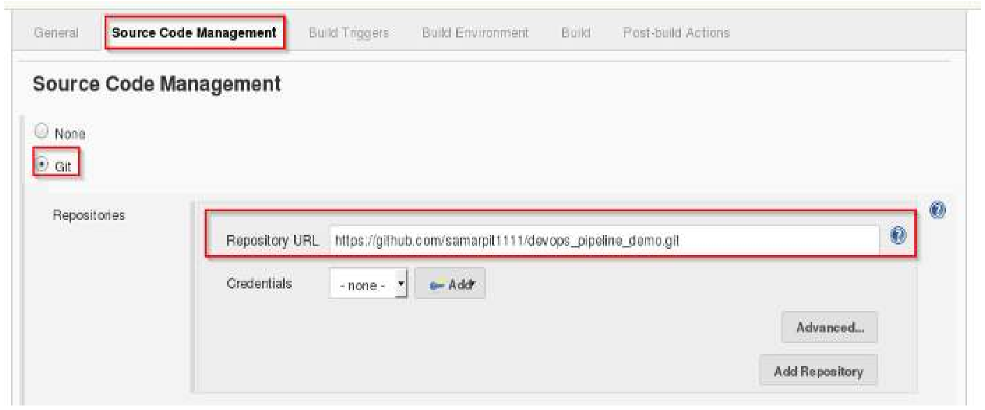
Step 1: In your Terminal or CLI, Start and enable Jenkins and docker

```
systemctl start jenkins
systemctl enable jenkins
systemctl start docker
```

Step 2: In your Jenkins console click on New Item from where you will create your first job.

Step 3: After you click on New Item , You need to choose an option freestyle project with name and save

Step 4: In the configuration section select SCM and you will add the git repo link and save it.


Source Code Management

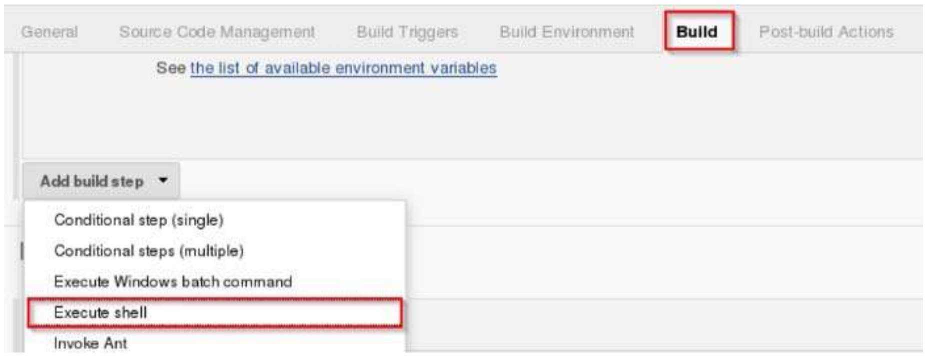
None [✓ Git]

Repositories

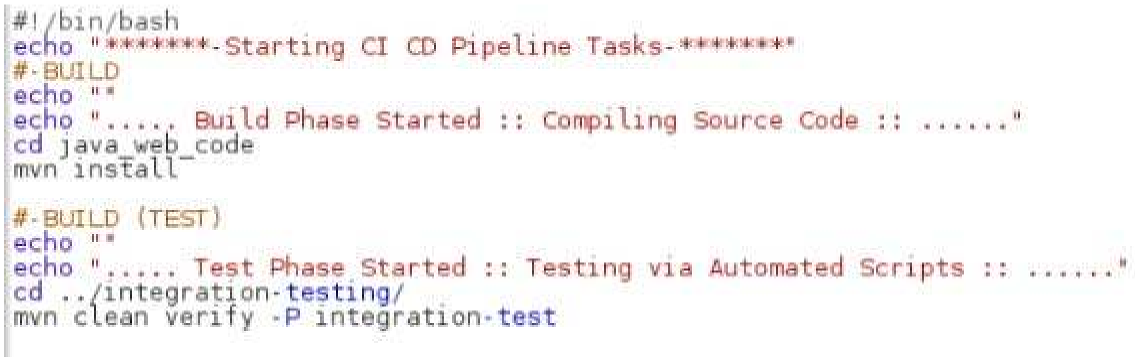
Repository URL: https://github.com/samarpit1111/devops_pipeline_demo.git

Credentials: - none -

Step 5: Then you will select Build option and choose to Execute shell


General
Source Code Management
Build Triggers
Build Environment
Build
Post-build Actions
See the list of available environment variables
Add build step
Conditional step (single)
Conditional steps (multiple)
Execute Windows batch command
Execute shell
Invoke Ant

Step 6: Provide the shell commands. Here it'll build the archive file to induce a war file. After that, it'll get the code that already forces then it uses wiz to put in the package. So, it merely installs the dependencies and compiles the applying.

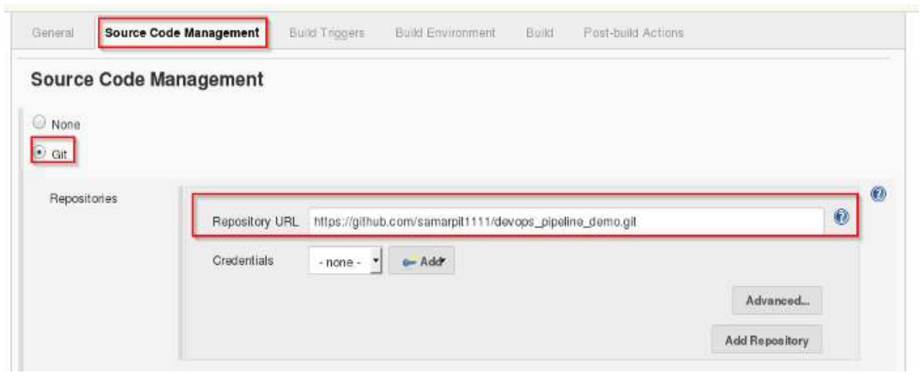


```
#!/bin/bash
echo "*******-Starting CI CD Pipeline Tasks-*******"
#-BUILD
echo ""
echo "..... Build Phase Started :: Compiling Source Code :: ......"
cd java_web_code
mvn install

#-BUILD (TEST)
echo ""
echo "..... Test Phase Started :: Testing via Automated Scripts :: ......"
cd ../integration-testing/
mvn clean verify -P integration-test
```

Step 7: Similarly you will create a new job as before.

Step 8: Click on the .freestyle project and save it with the proper name.

Step 9: Again repeat step 4, In the configuration section select SCM and you will add the git repo link and save it.

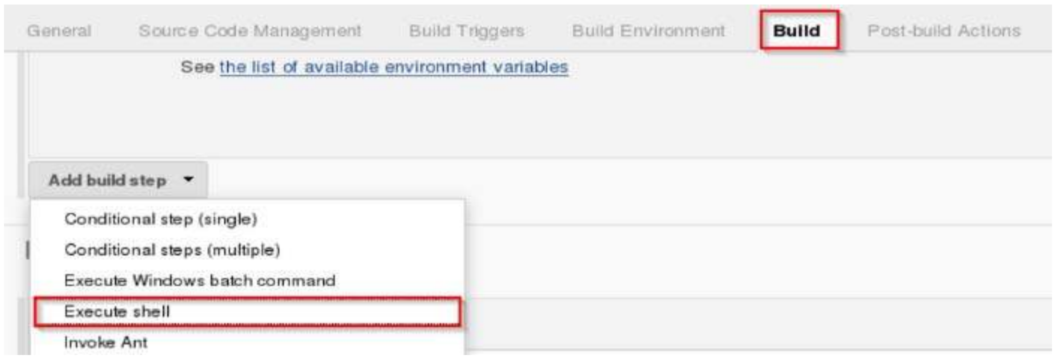


Source Code Management

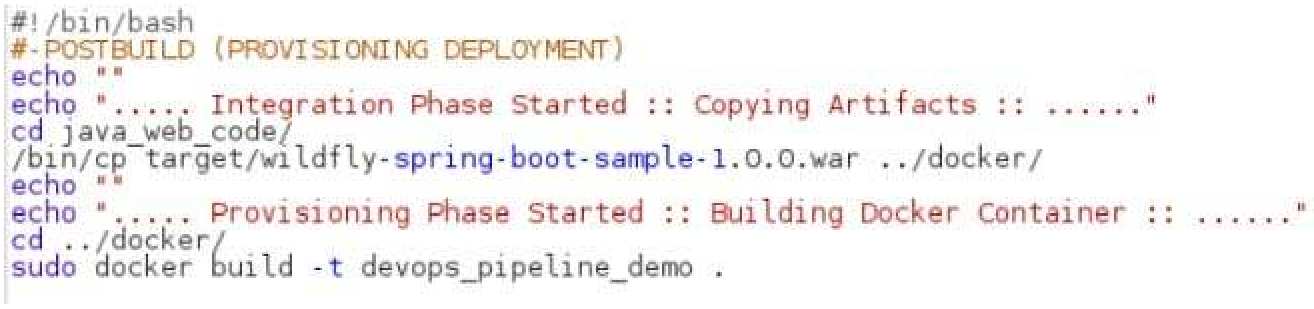
None
Git

Repositories
Repository URL: https://github.com/samarpit1111/devops_pipeline_demo.git
Credentials: - none -

Step 10: Repeat step 5, You will select Build option and choose to Execute shell


General
Source Code Management
Build Triggers
Build Environment
Build
Post-build Actions
See the list of available environment variables
Add build step
Conditional step (single)
Conditional steps (multiple)
Execute Windows batch command
Execute shell
Invoke Ant

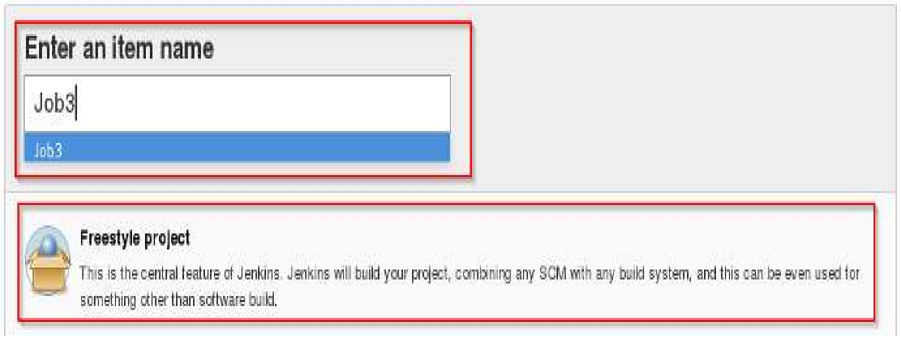
Step 11: You will now write the shell module commands as for int phase and build the container.



```
#!/bin/bash
#-POSTBUILD (PROVISIONING DEPLOYMENT)
echo ""
echo "..... Integration Phase Started :: Copying Artifacts :: ......"
cd java_web_code/
/bin/cp target/wildfly-spring-boot-sample-1.0.0.war ../docker/
echo ""
echo "..... Provisioning Phase Started :: Building Docker Container :: ......"
cd ../docker/
sudo docker build -t devops_pipeline_demo .
```

Step 12: Again you will create a new job as before in previous steps.

Step 13: Select freestyle project and provide the item name (here I have given Job3) and click on OK.

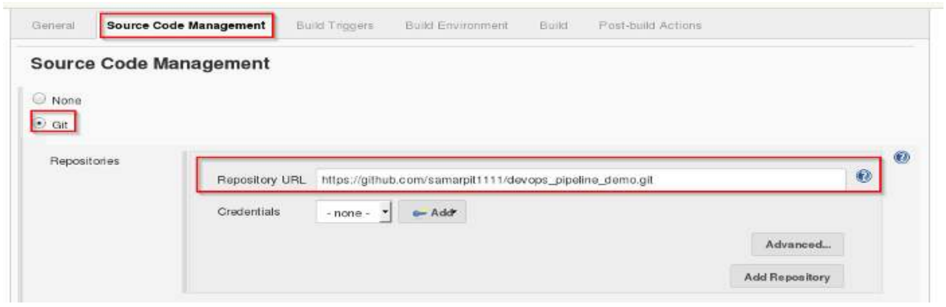

Enter an item name

Job3

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.

Step 14: Again repeat step 4, In the configuration section select SCM and you will add the git repo link and save it.



Source Code Management

None

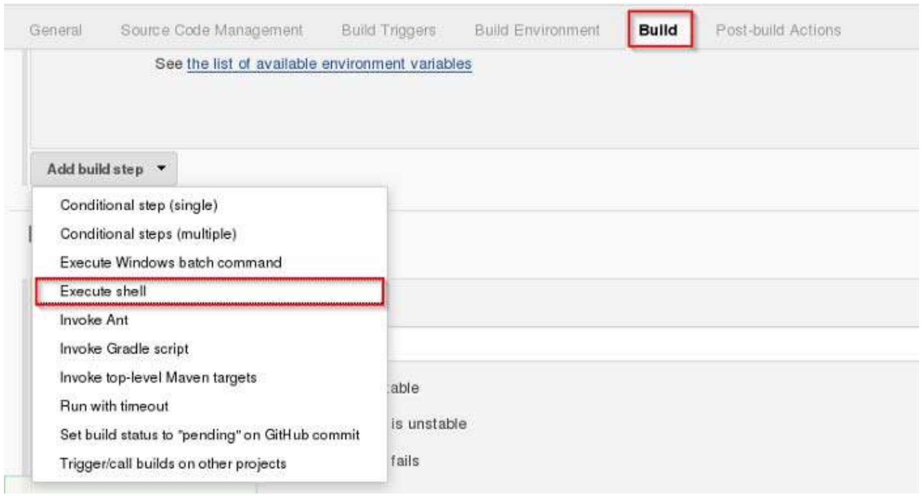
[✓ Git]

Repositories

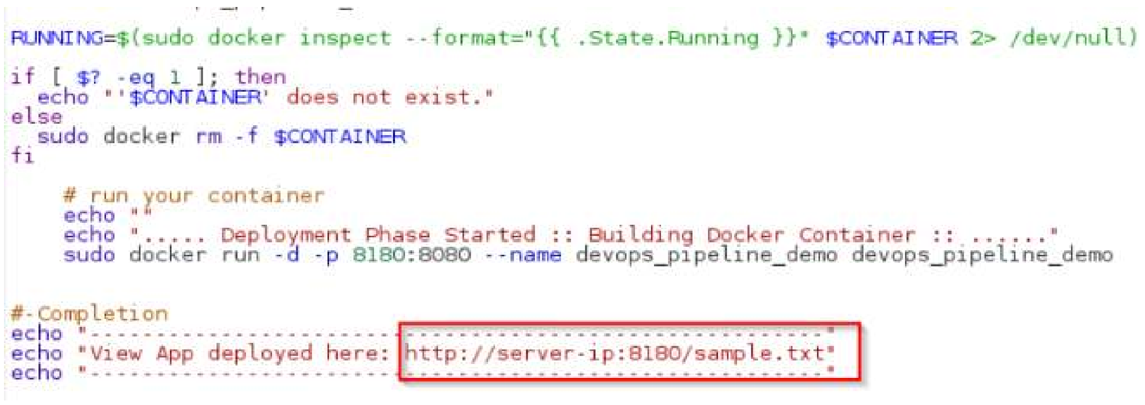
Repository URL https://github.com/samarpit1111/devops_pipeline_demo.git

Credentials - none -

Step 15: Repeat step 10, You will select Build option and choose to Execute shell.

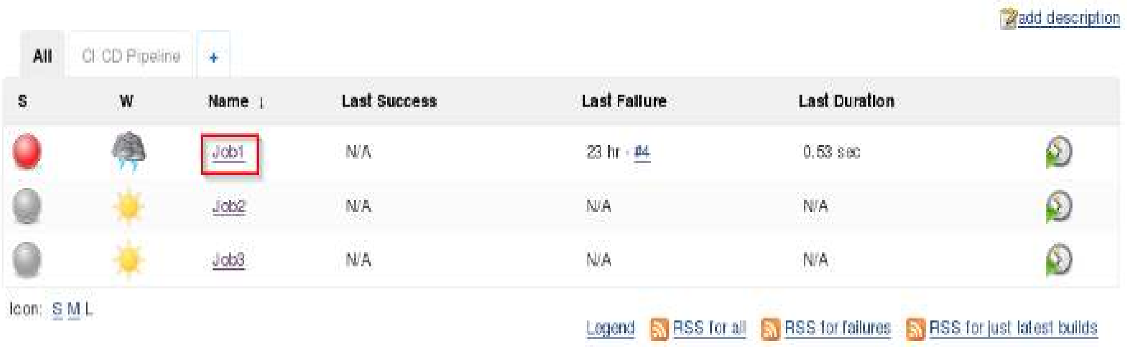

General
Source Code Management
Build Triggers
Build Environment
Build
Post-build Actions
See the list of available environment variables
Add build step
Conditional step (single)
Conditional steps (multiple)
Execute Windows batch command
Execute shell
Invoke Ant
Invoke Gradle script
Invoke top-level Maven targets
Run with timeout
Set build status to "pending" on GitHub commit
Trigger/call builds on other projects
able
is unstable
fails

Step 16: Write the shell commands, Now it will verify the container files and the deployment will be doe on port 8180, save it



```
RUNNING=$(sudo docker inspect --format="{{ .State.Running }}" $CONTAINER 2> /dev/null)
if [ $? -eq 1 ]; then
  echo "'$CONTAINER' does not exist."
else
  sudo docker rm -f $CONTAINER
fi

    # run your container
    echo ""
    echo "..... Deployment Phase Started :: Building Docker Container :: ......"
    sudo docker run -d -p 8180:8080 --name devops_pipeline_demo devops_pipeline_demo

#-Completion
echo "--------------------------------------------------------------"
echo "View App deployed here: http://server-ip:8180/sample.txt"
echo "--------------------------------------------------------------"
```

Step 17: Now, you will choose job 1 and click to configure.



add description

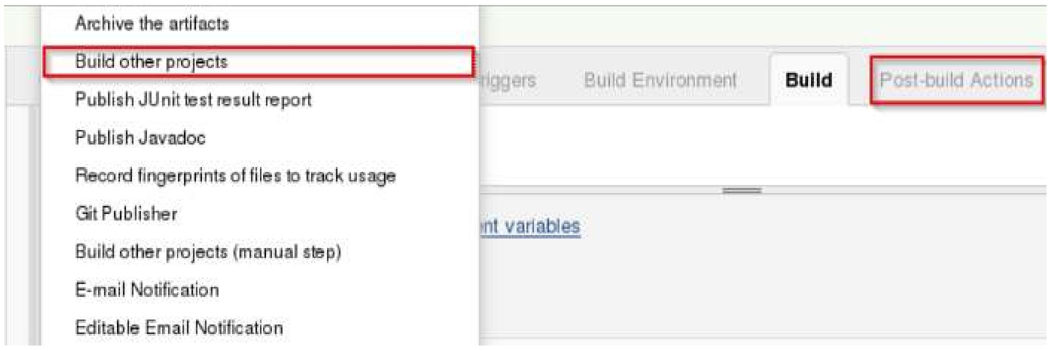
All | CI CD Pipeline | +

S	W	Name ↓	Last Success	Last Failure	Last Duration	
●		Job1	N/A	23 hr - #4	0.53 sec	
●		Job2	N/A	N/A	N/A	
●		Job3	N/A	N/A	N/A	

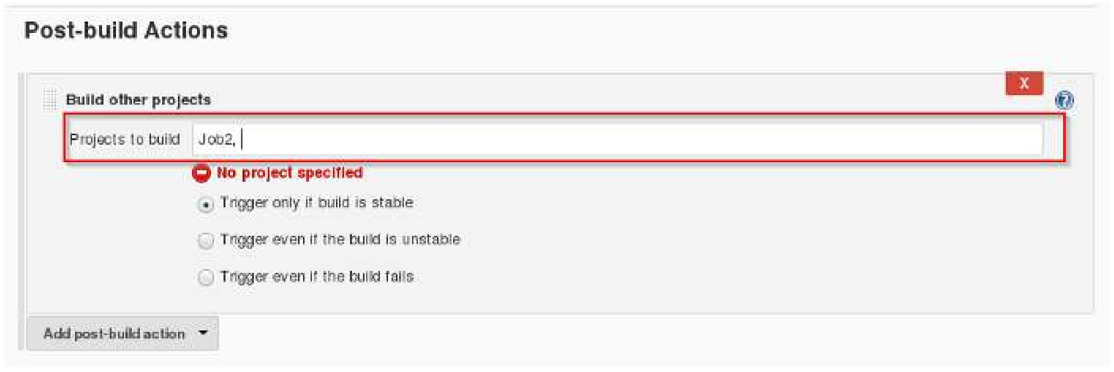
Icon: S M L

Legend RSS for all RSS for failures RSS for just latest builds

Step 18:From the build actions, You will choose post-build and click on build other projects



Step 19: You will need to provide the name of the project to build after the job 1 and then click save


Post-build Actions

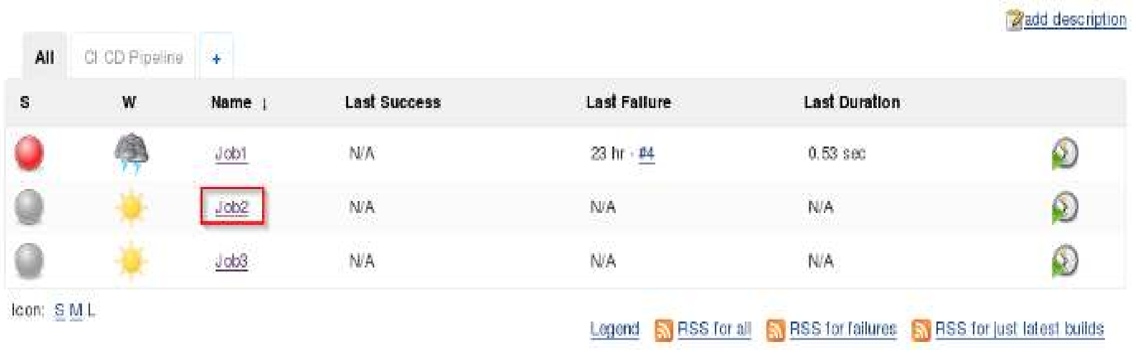
Build other projects

Projects to build: Job2,

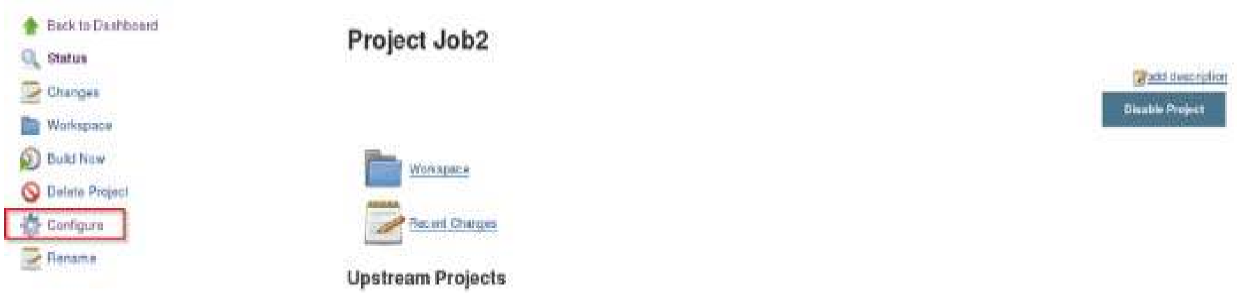
No project specified

- [✓ Trigger only if build is stable]
- Trigger even if the build is unstable
- Trigger even if the build fails

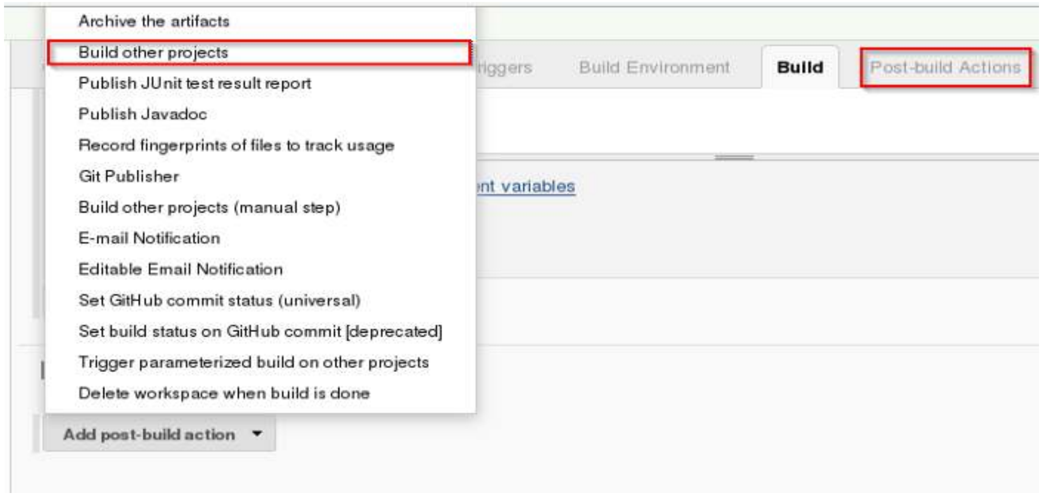
Step 20: Now, you will choose job 2 and click to configure.



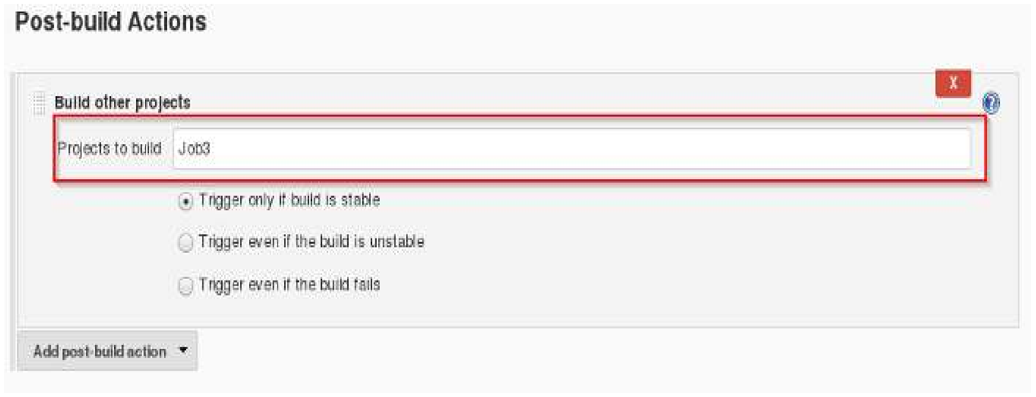
S	W	Name ↓	Last Success	Last Failure	Last Duration
●		Job1	N/A	23 hr - #4	0.53 sec
●		Job2	N/A	N/A	N/A
●		Job3	N/A	N/A	N/A



Step 21: From the build actions, You will choose post-build and click on build other projects


Archive the artifacts
Build other projects
Publish JUnit test result report
Publish Javadoc
Record fingerprints of files to track usage
Git Publisher
Build other projects (manual step)
E-mail Notification
Editable Email Notification
Set GitHub commit status (universal)
Set build status on GitHub commit [deprecated]
Trigger parameterized build on other projects
Delete workspace when build is done
Add post-build action
riggers
Build Environment
Build
Post-build Actions
nt variables

Step 22: You will need to provide the name of the project to build after the job 2 and then click save

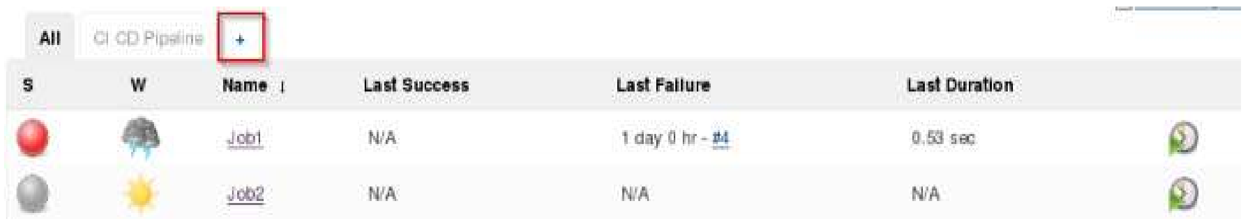

Post-build Actions

Build other projects

Projects to build: Job3

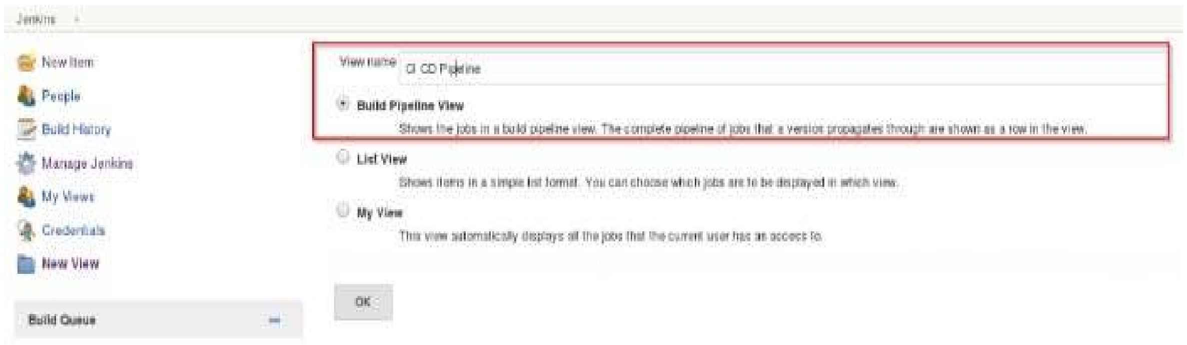
(•) Trigger only if build is stable
( ) Trigger even if the build is unstable
( ) Trigger even if the build fails

Step 23: let's create a pipeline, by adding to + sign



S	W	Name ↓	Last Success	Last Failure	Last Duration
🔴		Job1	N/A	1 day 0 hr - #4	0.53 sec
⚪		Job2	N/A	N/A	N/A

Step 24: Now, You will choose and select a build Pipeline view and add the name.



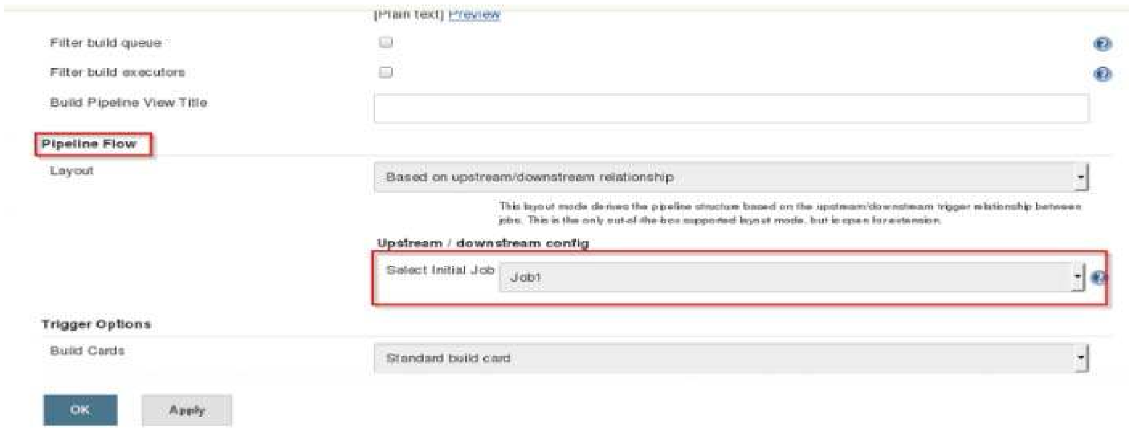
View name: CI CD Pipeline

(●) **Build Pipeline View**
Shows the jobs in a build pipeline view. The complete pipeline of jobs that a version propagates through are shown as a row in the view.

( ) **List View**
Shows items in a simple list format. You can choose which jobs are to be displayed in which view.

( ) **My View**
This view automatically displays all the jobs that the current user has an access to.

Step 25: Choose the Job 1 and save OK



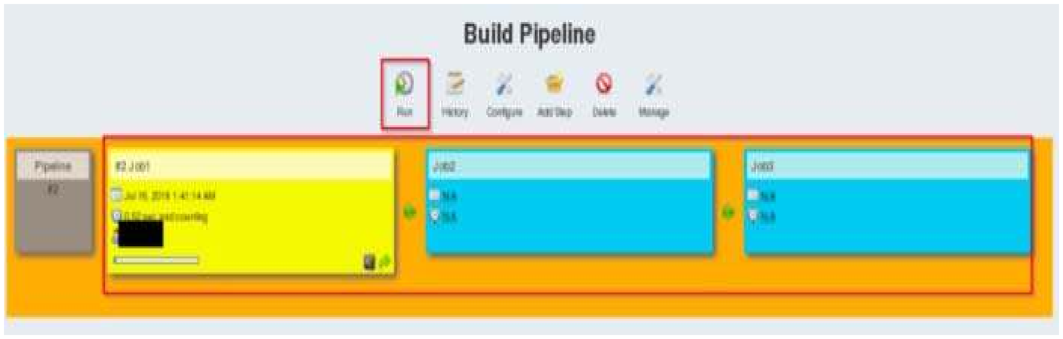
Filter build queue ☐
Filter build executors ☐
Build Pipeline View Title

Pipeline Flow
Layout: Based on upstream/downstream relationship
This layout mode derives the pipeline structure based on the upstream/downstream trigger relationship between jobs. This is the only out-of-the-box supported layout mode, but is open for extension.

Upstream / downstream config
Select Initial Job: Job1

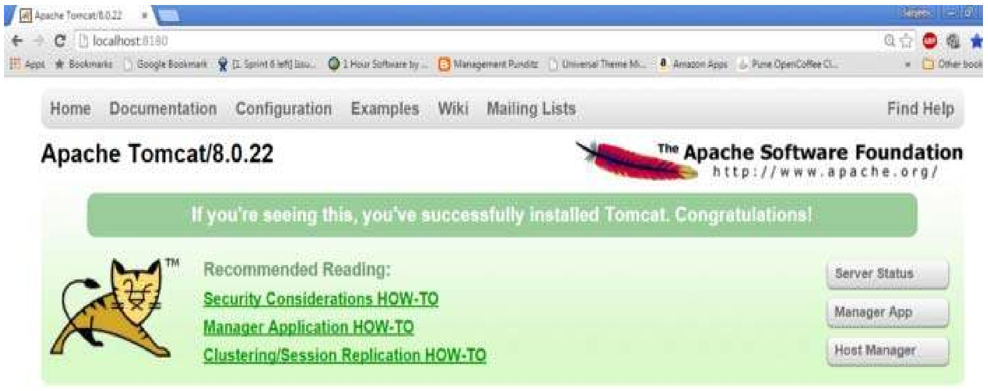
Trigger Options
Build Cards: Standard build card

Step 26: let's RUN it and start the CICD process now



[redacted]

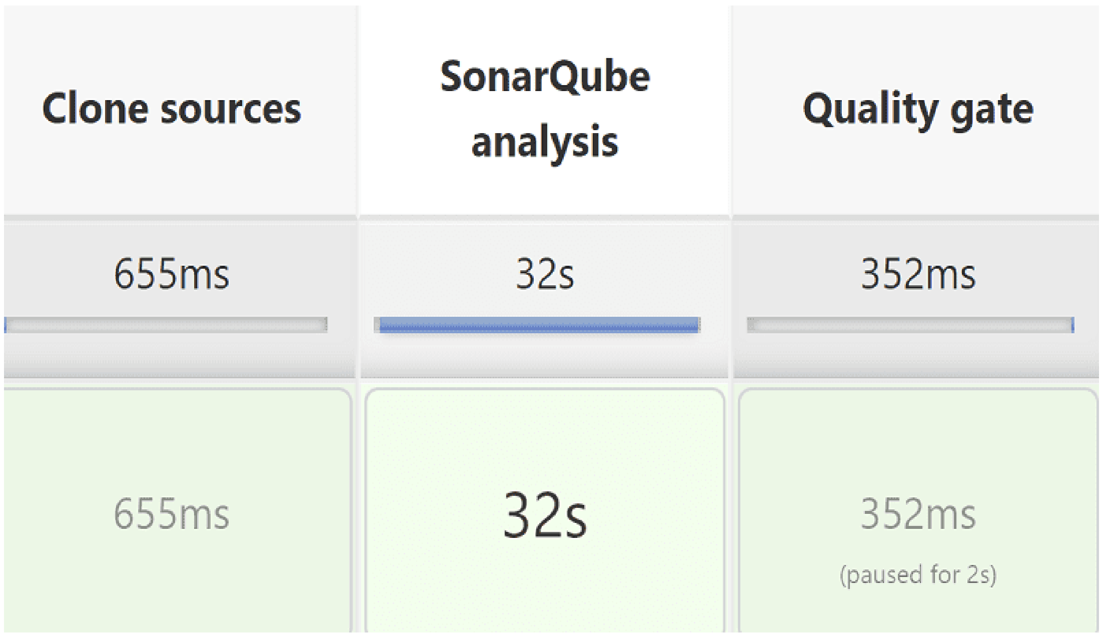
Step 27:After you build the job, To verify open the link in your browser localhost:8180/sample.text, This is the port where your app is running


localhost:8180
Home
Documentation
Configuration
Examples
Wiki
Mailing Lists
Find Help
Apache Tomcat/8.0.22
The Apache Software Foundation
http://www.apache.org/
If you're seeing this, you've successfully installed Tomcat. Congratulations!
Recommended Reading:
Security Considerations HOW-TO
Manager Application HOW-TO
Clustering/Session Replication HOW-TO
Server Status
Manager App
Host Manager

Exercise 10:

SonarQube is an excellent tool for measuring code quality, using static analysis to find code smells, bugs, vulnerabilities, and poor test coverage. Rather than manually analysing the reports, why not automate the process by integrating SonarQube with your Jenkins continuous integration pipeline? This way, you can configure a quality gate based on your own requirements, ensuring bad code always fails the build.

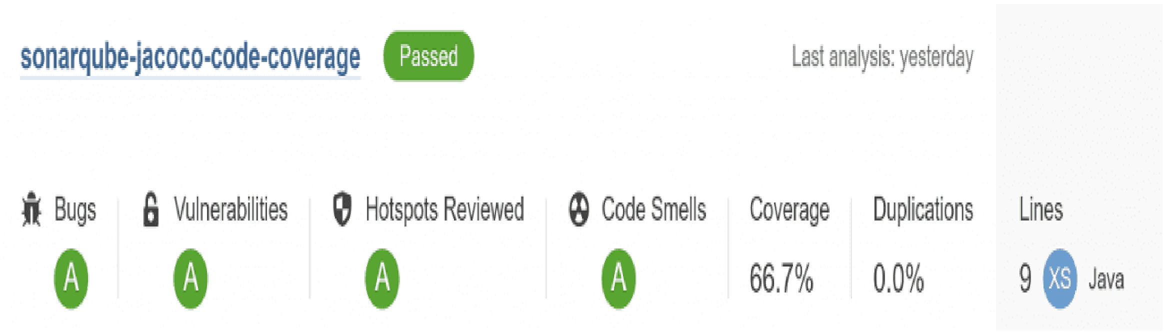
You'll learn exactly how to do that in this article, through a full worked example where we add SonarQube analysis and SonarQube quality gate stages to a Jenkins pipeline.



SonarQube refresher

SonarQube works by running a local process to scan your project, called the SonarQube scanner. This sends reports to a central server, known as the SonarQube server.

The SonarQube server also has a UI where you can browse these reports. They look like this:

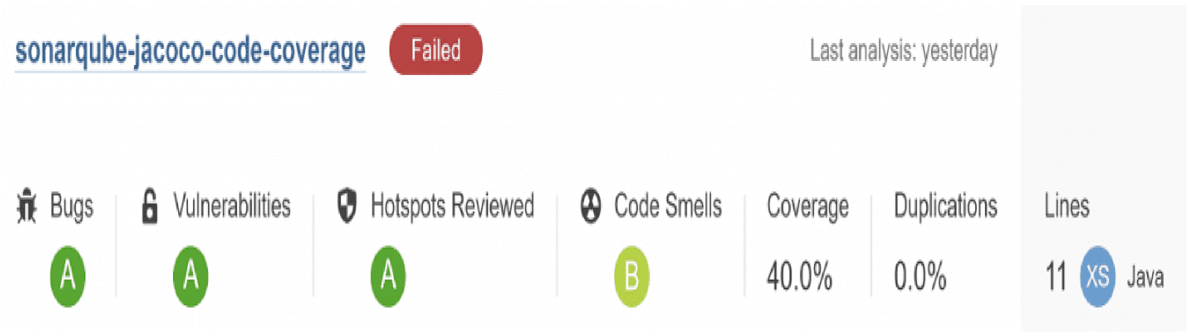


Bugs	Vulnerabilities	Hotspots Reviewed	Code Smells	Coverage	Duplications	Lines
A	A	A	A	66.7%	0.0%	9 XS Java

Quality gates

In SonarQube a quality gate is a set of conditions that must be met in order for a project to be marked as *passed*. In the above example the project met all the conditions.

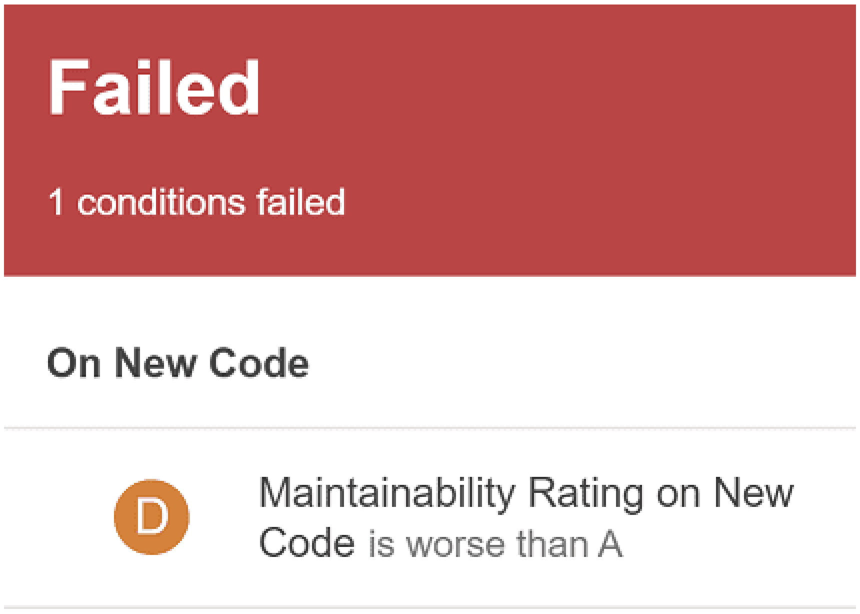
Here's an example where things didn't go so well.



Bugs	Vulnerabilities	Hotspots Reviewed	Code Smells	Coverage	Duplications	Lines
A	A	A	B	40.0%	0.0%	11 XS Java

Clicking on the project name gives full details of the failure.

Here you can see here that a condition failed because the maintainability rating was a *D* rather than *A*.


Failed

1 conditions failed

On New Code

D Maintainability Rating on New Code is worse than A

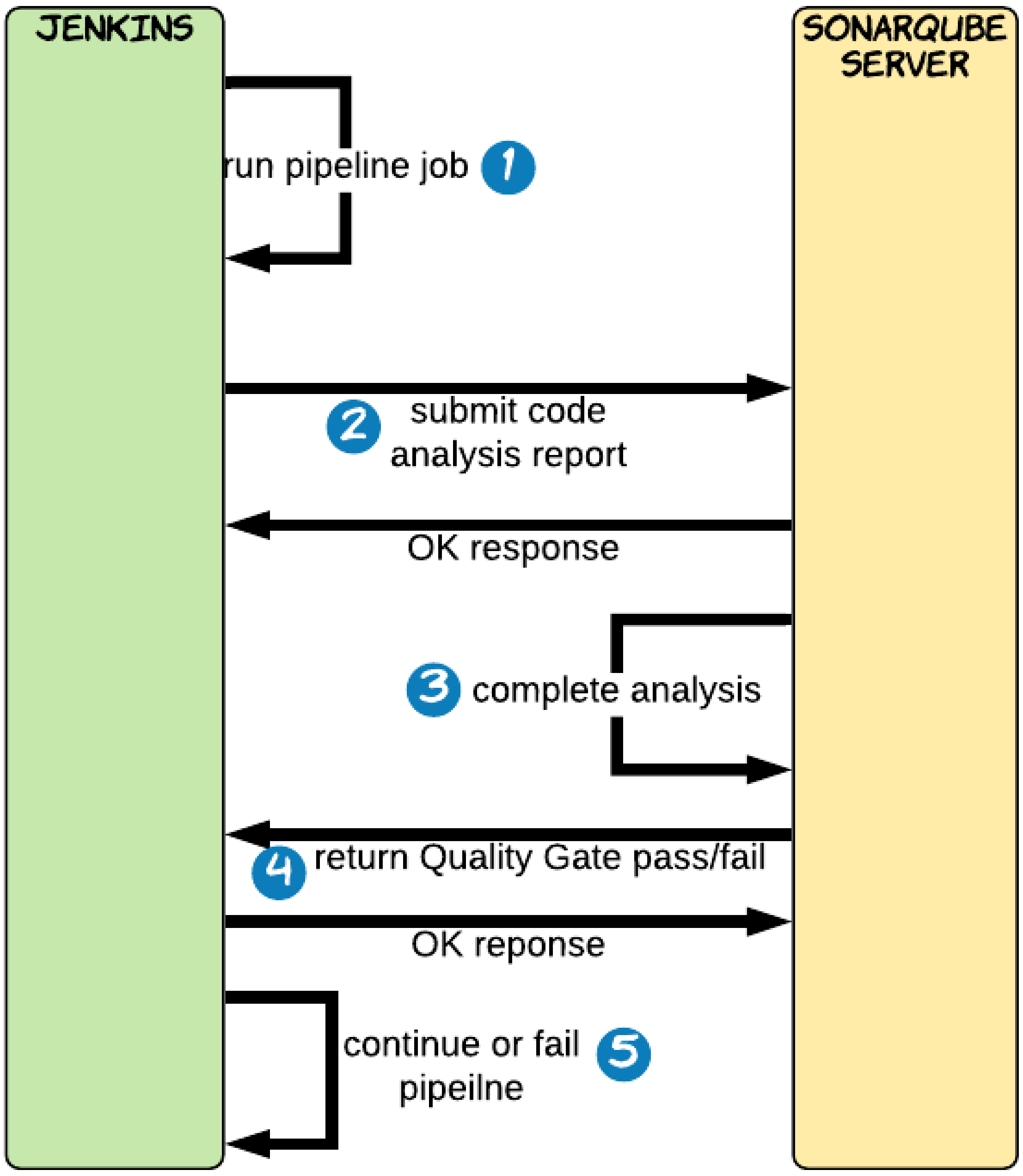
SonarQube and Jenkins

Running a SonarQube scan from a build on your local workstation is fine, but a robust solution needs to include SonarQube as part of the continuous integration process. If you add SonarQube analysis into a Jenkins pipeline, you can ensure that if the quality gate fails then the pipeline won't continue to further stages such as publish or release.

After all, nobody wants to release crappy code into production. 👌

To do this, we can use the SonarQube Scanner plugin for Jenkins. It includes two features that we're going to make use of today:

1. SonarQube server configuration – the plugin lets you set your SonarQube server location and credentials. This information is then used in a SonarQube analysis pipeline stage to send code analysis reports to that SonarQube server.
2. SonarQube Quality Gate webhook – when a code analysis report is submitted to SonarQube, unfortunately it doesn't respond synchronously with the result of whether the report passed the quality gate or not. To do this, a webhook call must be configured in SonarQube to call back into Jenkins to allow our pipeline to continue (or fail). The SonarQube Scanner Jenkins plugin makes this webhook available.


JENKINS
SONARQUBE SERVER
run pipeline job 1
2 submit code analysis report
OK response
3 complete analysis
4 return Quality Gate pass/fail
OK reponse
continue or fail pipeilne 5

Here's a full breakdown of the interaction between Jenkins and SonarQube:

1. a Jenkins pipeline is started
2. the SonarQube scanner is run against a code project, and the analysis report is sent to SonarQube server

3. SonarQube finishes analysis and checking the project meets the configured Quality Gate

4. SonarQube sends a pass or failure result back to the Jenkins webhook exposed by the plugin

5. the Jenkins pipeline will continue if the analysis result is a pass or optionally otherwise fail

Full worked example

Let's get our hands dirty with a worked example. We'll run through all the steps in the UI manually as this is the best way to understand the setup.

In this example we'll:

1. get Jenkins and SonarQube up and running
2. install the SonarQube Scanner Jenkins plugin and configure it to point to our SonarQube instance
3. configure SonarQube to call the Jenkins webhook when project analysis is finished
4. create two Jenkins pipelines
    - one that runs against a codebase with zero issues (I wish all my code was like this 😉)
    - one that runs against a codebase with bad code issues
5. run the pipelines and see it all working

You'll need to make sure you have Docker installed before carrying on.

> Fast track: to get up and running quickly check out this GitHub repository. Everything is setup through configuration-as-code, except the steps under Configure SonarQube below.

Running Jenkins and SonarQube

What better way to start these two services than with Docker Compose? Create the following file *docker-compose.yml*:

```
version: "3"
```

```
services:

sonarqube:

image: sonarqube:lts

ports:

- 9000:9000

networks:

- mynetwork

environment:

- SONAR_FORCEAUTHENTICATION=false

jenkins:

image: jenkins/jenkins:2.319.1-jdk11

ports:

- 8080:8080

networks:

- mynetwork

networks:

mynetwork:
```

- we're configuring two containers in Docker Compose: Jenkins and SonarQube
- the Docker images used come from the official repositories in Docker Hub
- we're adding both containers to the same network so they can talk to each other
- for demo purposes SonarQube authentication is disabled so Jenkins won't need to pass a token

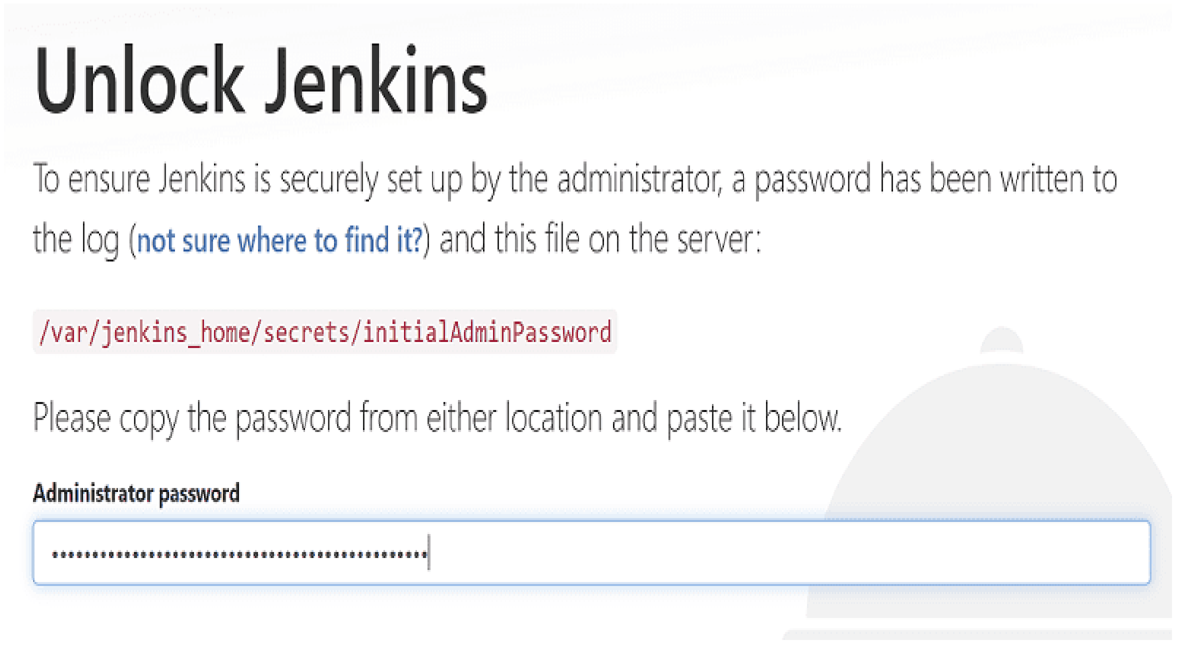
Running `docker-compose up` in the directory containing the file will start Jenkins on http://localhost:8080 and SonarQube on http://localhost:9000. Awesomeness!

Configuring the SonarQube Scanner Jenkins plugin

Grab the Jenkins administrator password from the Jenkins logs in the console output of the Docker Compose command you just ran.

```
jenkins_1    | Please use the following password to proceed to installation:
jenkins_1    |
jenkins_1    | 7efed7f025ee430c8938beaa975f5dde
```

Head over to your Jenkins instance and paste in the password.



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:

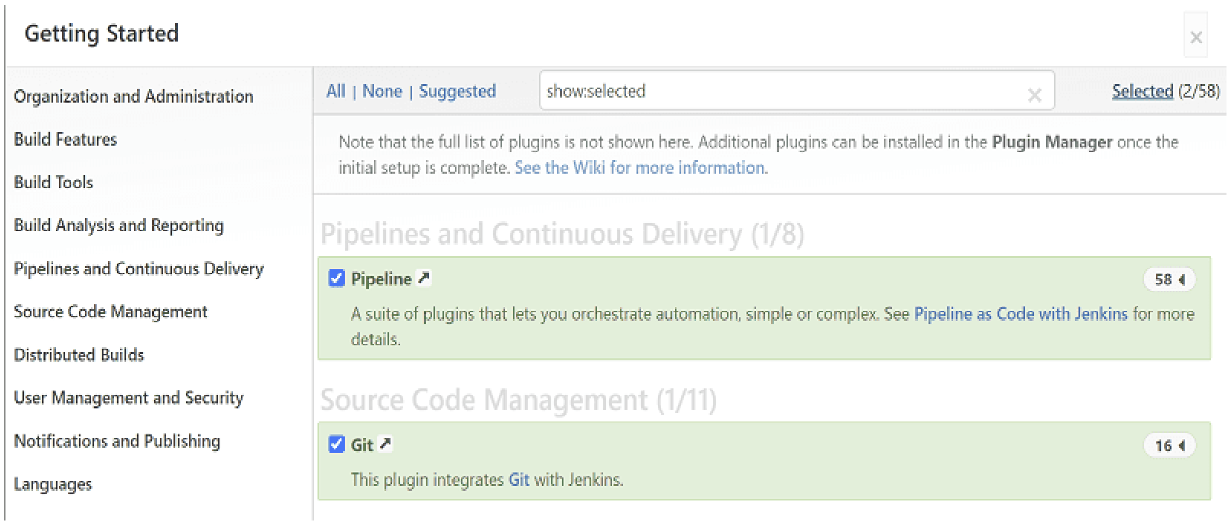
`/var/jenkins_home/secrets/initialAdminPassword`

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

Administrator password

··

On the next page choose Select plugins to install and install only the *pipeline* and *git* plugins. The *SonarQube Scanner* plugin we'll have to install afterwards since this Getting Started page doesn't give us the full choice of plugins.


Getting Started

×
Organization and Administration
Build Features
Build Tools
Build Analysis and Reporting
Pipelines and Continuous Delivery
Source Code Management
Distributed Builds
User Management and Security
Notifications and Publishing
Languages
All | None | Suggested
show:selected
Selected (2/58)

Note that the full list of plugins is not shown here. Additional plugins can be installed in the **Plugin Manager** once the initial setup is complete. See the Wiki for more information.

Pipelines and Continuous Delivery (1/8)

☑ Pipeline ↗ 58

A suite of plugins that lets you orchestrate automation, simple or complex. See Pipeline as Code with Jenkins for more details.

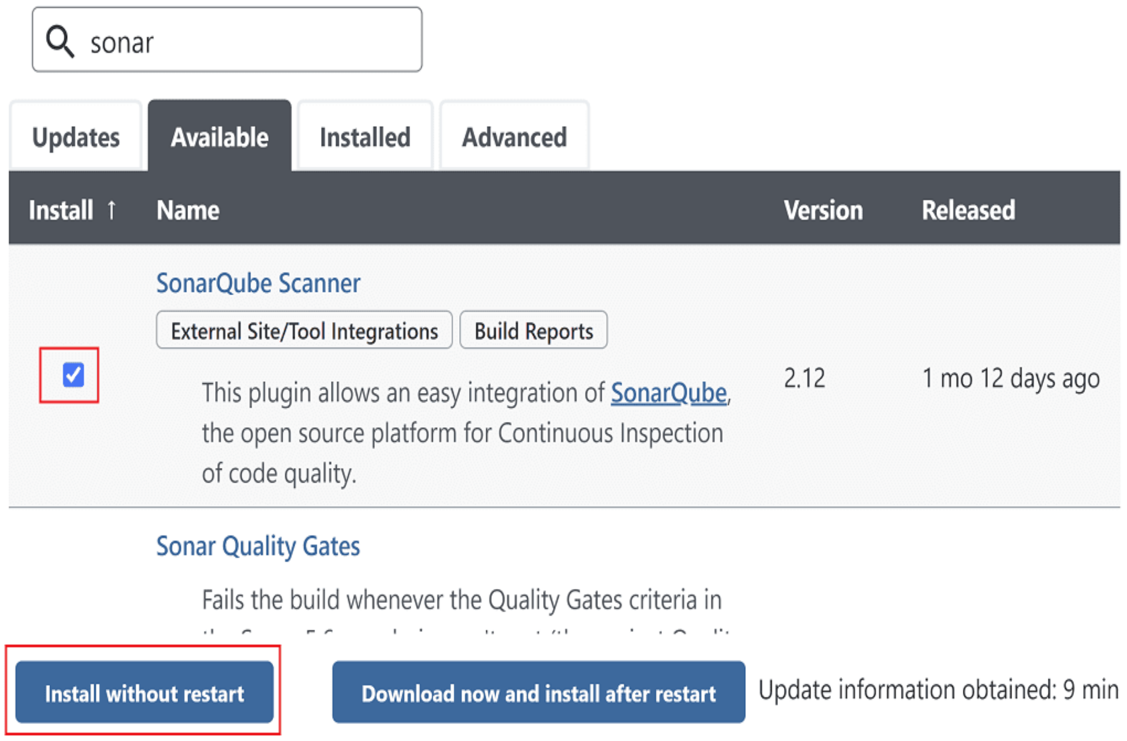
Source Code Management (1/11)

☑ Git ↗ 16

This plugin integrates Git with Jenkins.

In the final steps you'll have to create a user and confirm the Jenkins URL of http://localhost:8080.

Once complete head over to Manage Jenkins > Manage Plugins > Available and search for *sonar*. Select the SonarQube Scanner plugin and click Install without restart.



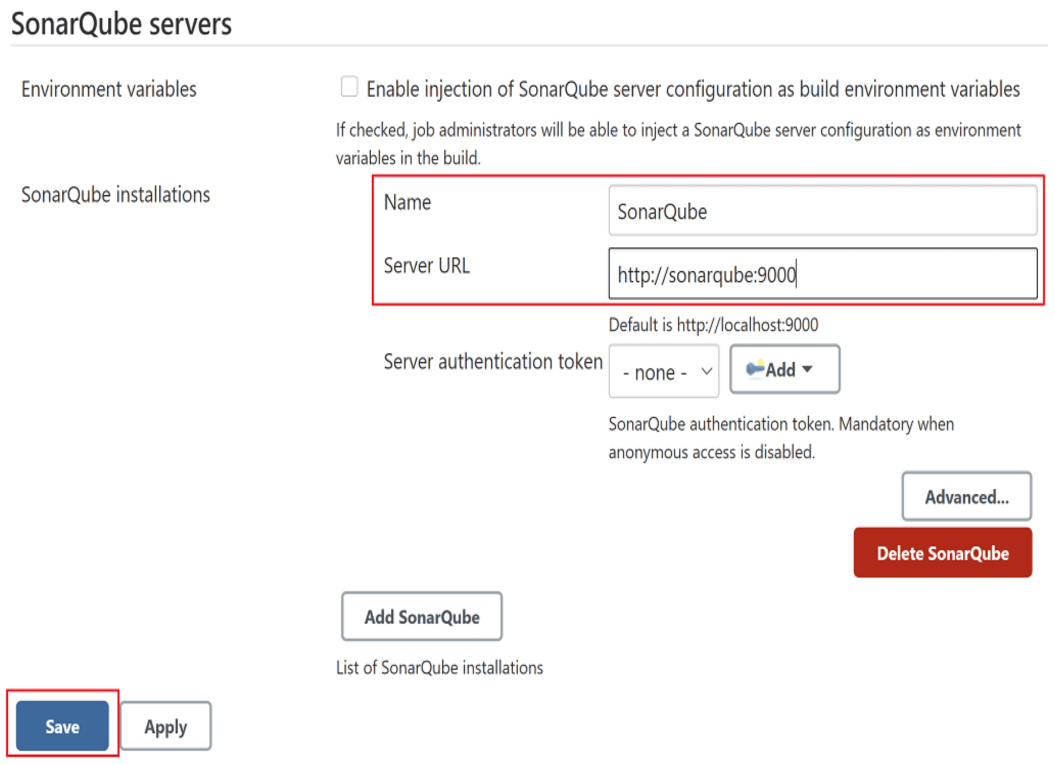
Install	Name	Version	Released
✓	SonarQube Scanner — This plugin allows an easy integration of SonarQube, the open source platform for Continuous Inspection of code quality.	2.12	1 mo 12 days ago
	Sonar Quality Gates — Fails the build whenever the Quality Gates criteria in		

Update information obtained: 9 min

Once the plugin is installed, let's configure it!

Go to Manage Jenkins > Configure System and scroll down to the SonarQube servers section. This is where we'll add details of our SonarQube server so Jenkins can pass its details to our project's build when we run it.

Click the Add SonarQube button. Now add a Name for the server, such as *SonarQube*. The Server URL will be *http://sonarqube:9000*. Remember to click Save.


SonarQube servers

Environment variables: ☐ Enable injection of SonarQube server configuration as build environment variables

If checked, job administrators will be able to inject a SonarQube server configuration as environment variables in the build.

SonarQube installations

Name: SonarQube

Server URL: http://sonarqube:9000

Default is http://localhost:9000

Server authentication token: - none -

SonarQube authentication token. Mandatory when anonymous access is disabled.

List of SonarQube installations

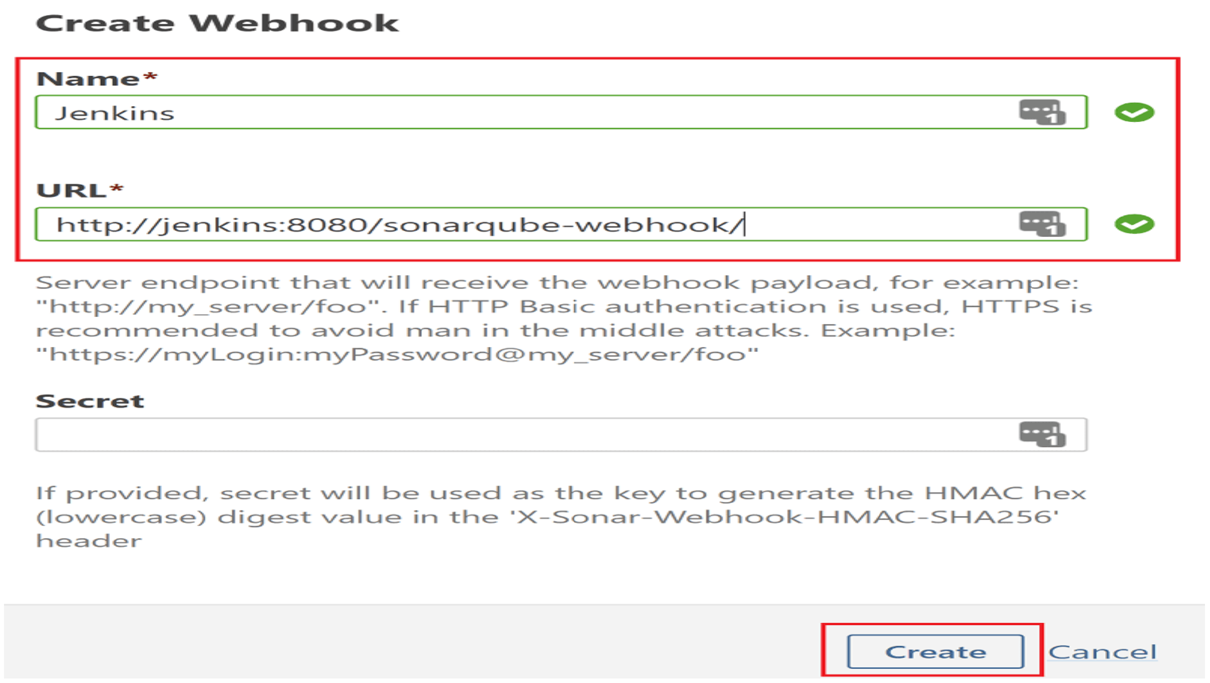
Networking in Docker Compose: the SonarQube URL is http://sonarqube:9000 because by default Docker Compose allows any service to call any other service in the same network. You do this by using the service name as the hostname in the request URL, as defined in *docker-compose.yml*. This is why we use a host of sonarqube.

Configuring SonarQube

Let's jump over to SonarQube. Click Log in at the top-right of the page, and log in with the default credentials of admin/admin. You'll then have to set a new password.

Now go to Administration > Configuration > Webhooks. This is where we can add webhooks that get called when project analysis is completed. In our case we need to configure SonarQube to call Jenkins to let it know the results of the analysis.

Click Create, and in the popup that appears give the webhook a name of *Jenkins*, set the URL to *http://jenkins:8080/sonarqube-webhook* and click Create.



Create Webhook

Name*	Jenkins
URL*	http://jenkins:8080/sonarqube-webhook/

Server endpoint that will receive the webhook payload, for example: "http://my_server/foo". If HTTP Basic authentication is used, HTTPS is recommended to avoid man in the middle attacks. Example: "https://myLogin:myPassword@my_server/foo"

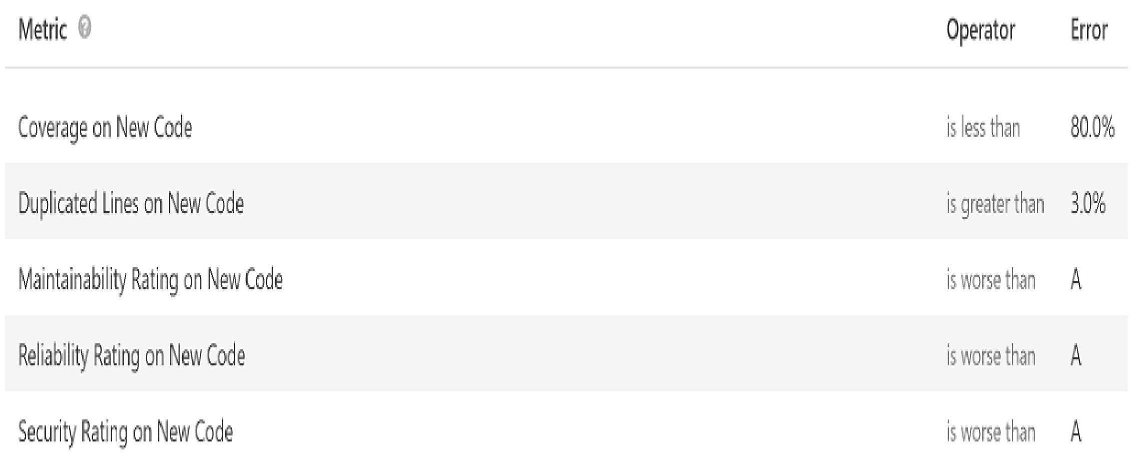
Secret

If provided, secret will be used as the key to generate the HMAC hex (lowercase) digest value in the 'X-Sonar-Webhook-HMAC-SHA256' header

In this case, the URL has the path sonarqube-webhook which is exposed by the SonarQube Scanner plugin we installed earlier.

Adding a quality gate

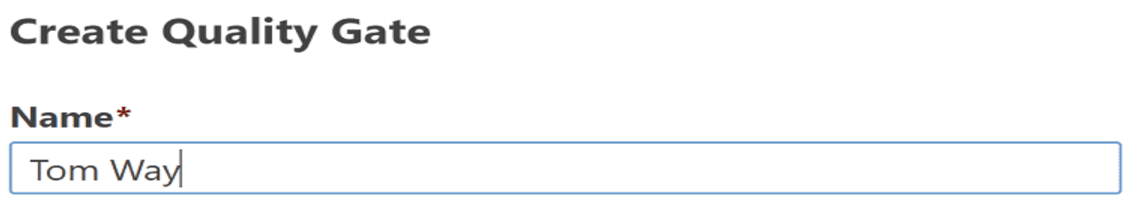
SonarQube comes with its own *Sonar way* quality gate enabled by default. If you click on Quality Gates you can see the details of this.



Metric	Operator	Error
Coverage on New Code	is less than	80.0%
Duplicated Lines on New Code	is greater than	3.0%
Maintainability Rating on New Code	is worse than	A
Reliability Rating on New Code	is worse than	A
Security Rating on New Code	is worse than	A

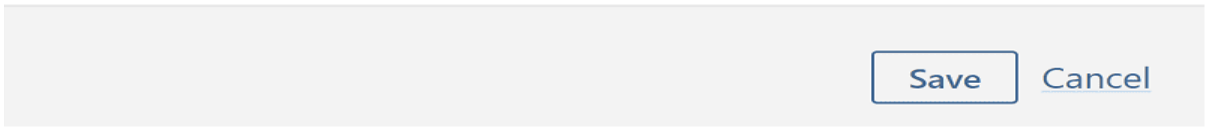
It's all about making sure that new code is of a high quality. In this example we want to check the quality of existing code, so we need to create a new quality gate.

Click Create, then give the quality gate a name. I've called mine *Tom Way* 😉

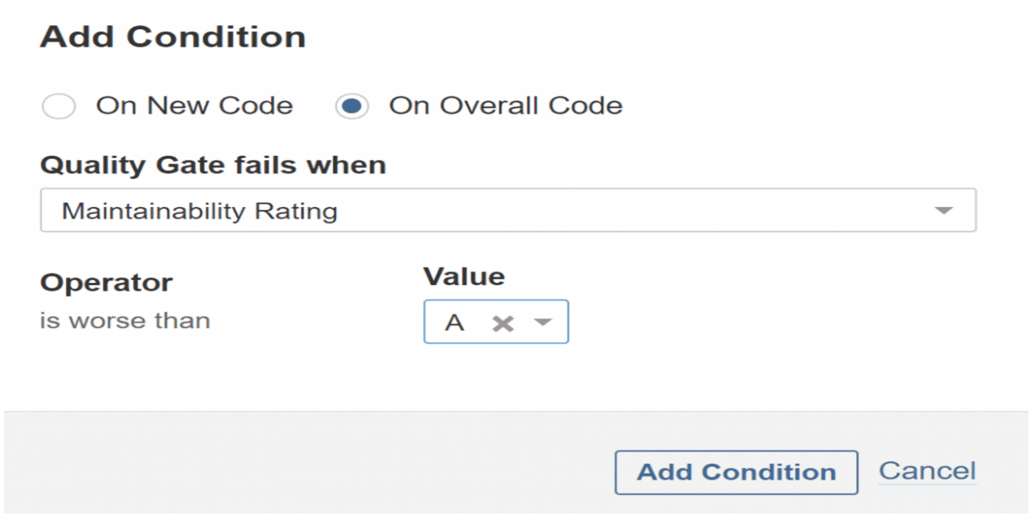

Create Quality Gate

Name*

Tom Way



Click Save then on the next screen click Add Condition. Select On Overall Code. Search for the metric Maintainability Rating and choose worse than A. This means that if existing code is not maintainable then the quality gate will fail. Click Add Condition to save the condition.


Add Condition

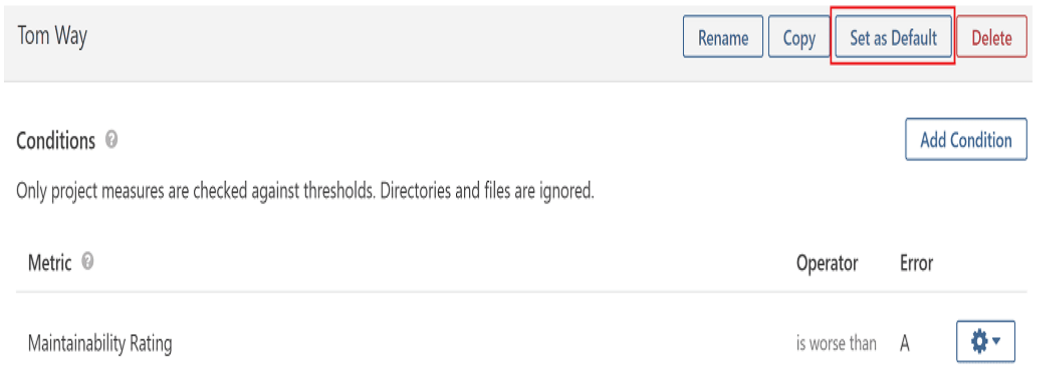
On New Code [✓ On Overall Code]

Quality Gate fails when: Maintainability Rating

Operator: is worse than

Value: A

Finally click Set as Default at the top of the page to make sure that this quality gate will apply to any new code analysis.


Tom Way

Conditions

Only project measures are checked against thresholds. Directories and files are ignored.

Metric	Operator	Error
Maintainability Rating	is worse than	A

Creating Jenkins pipelines

Last thing to do is setup two Jenkins pipelines:

1. A pipeline which runs against a code project over at the sonarqube-jacoco-code-coverage GitHub repository. The code here is decent enough that the pipeline should pass.
2. A pipeline which runs against the same project, but uses the bad-code branch. The code here is so bad that the pipeline should fail.

Good code pipeline

Back in Jenkins click New Item and give it a name of *sonarqube-good-code*, select the Pipeline job type, then click OK.

Scroll down to the Pipeline section of the configuration page and enter the following declarative pipeline script in the Script textbox:

```
pipeline {

agent any

stages {

stage('Clone sources'){

steps {

git url: 'https://github.com/tkgregory/sonarqube-jacoco-code-coverage.git'

}

}

stage('SonarQube analysis'){

steps {

withSonarQubeEnv('SonarQube'){

sh "./gradlew sonarqube"

}
}

}

stage("Quality gate"){

steps {

waitForQualityGate abortPipeline: true

}

}

}

}
```

The script has three stages:

1. in the Clone sources stage code is cloned from the GitHub repository mentioned earlier
2. in the SonarQube analysis stage we use the withSonarQubeEnv('Sonarqube') method exposed by the plugin to wrap the Gradle build of the code repository. This provides all the configuration required for the build to know where to find SonarQube. Note that the project build itself must have a way of running SonarQube analysis, which in this case is done by running ./gradlew sonarqube. For more information about running SonarQube analysis in a Gradle build see this article
3. in the Quality gate stage we use the waitForQualityGate method exposed by the plugin to wait until the SonarQube server has called the Jenkins webhook. The abortPipeline flag means if the SonarQube analysis result is a failure, we abort the pipeline.

Click Save to save the pipeline.

SonarQube magic: all the withSonarQubeEnv method does is export some environment variables that the project's build understands. By adding a pipeline step which runs the command printenv wrapped in withSonarQubeEnv, you'll be able to see environment variables such as SONAR_HOST_URL being set. These get picked up by the Gradle build of the code project to tell it which SonarQube server to connect to.

Bad code pipeline

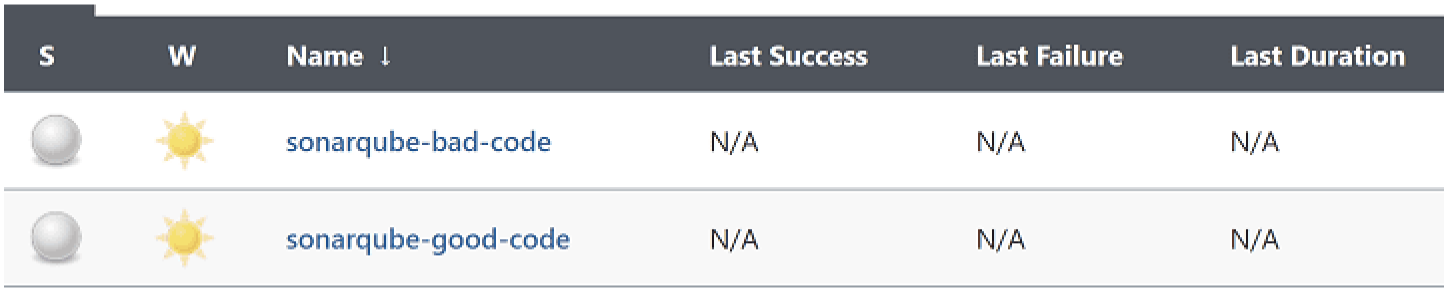
Create another pipeline in the same way, but name it *sonarqube-bad-code*. The pipeline script is almost exactly the same, except this time we need to check out the *bad-code* branch of the same repository.

```
pipeline {

agent any

stages {

stage('Clone sources'){

steps {

git branch: 'bad-code', url: 'https://github.com/tkgregory/sonarqube-jacoco-code-coverage.git'

}

}

stage('SonarQube analysis'){

steps {

withSonarQubeEnv('SonarQube'){

sh "./gradlew sonarqube"

}

}

}

stage("Quality gate"){

steps {

waitForQualityGate abortPipeline: true

}
}

}

}
```

- in the Clone sources stage we're now also specifying the branch attribute to point to the *bad-code* branch

Again, click Save.

You should now have two Jenkins jobs waiting to be run.



S	W	Name ↓	Last Success	Last Failure	Last Duration
●	☀	sonarqube-bad-code	N/A	N/A	N/A
●	☀	sonarqube-good-code	N/A	N/A	N/A

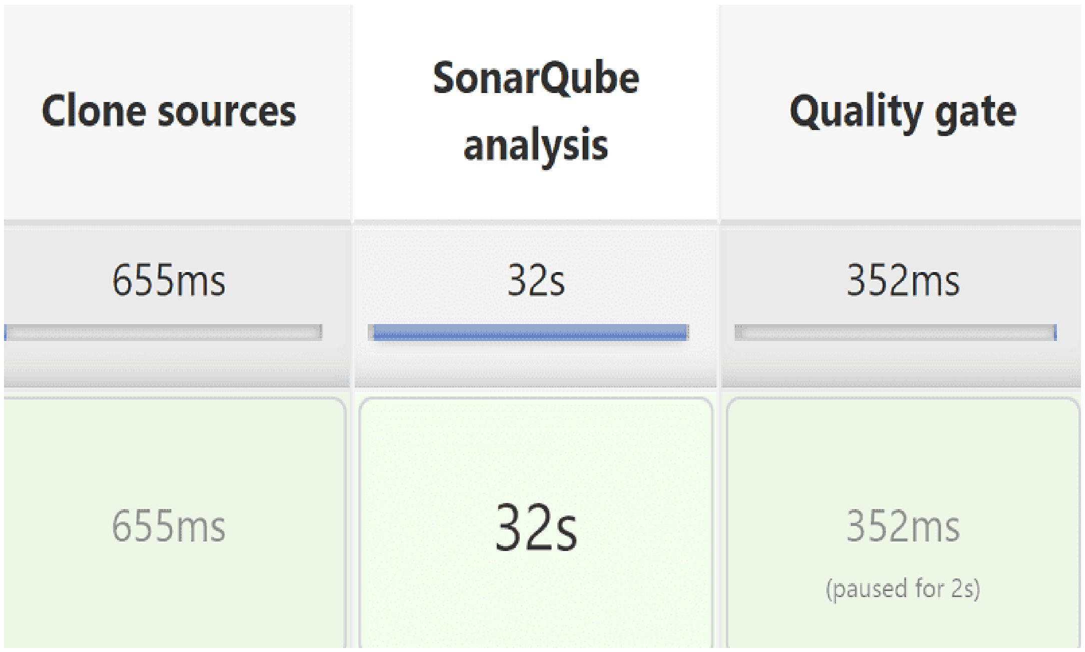
Any guesses as to what we're going to do next?

SonarQube analysis and quality gate stages in action

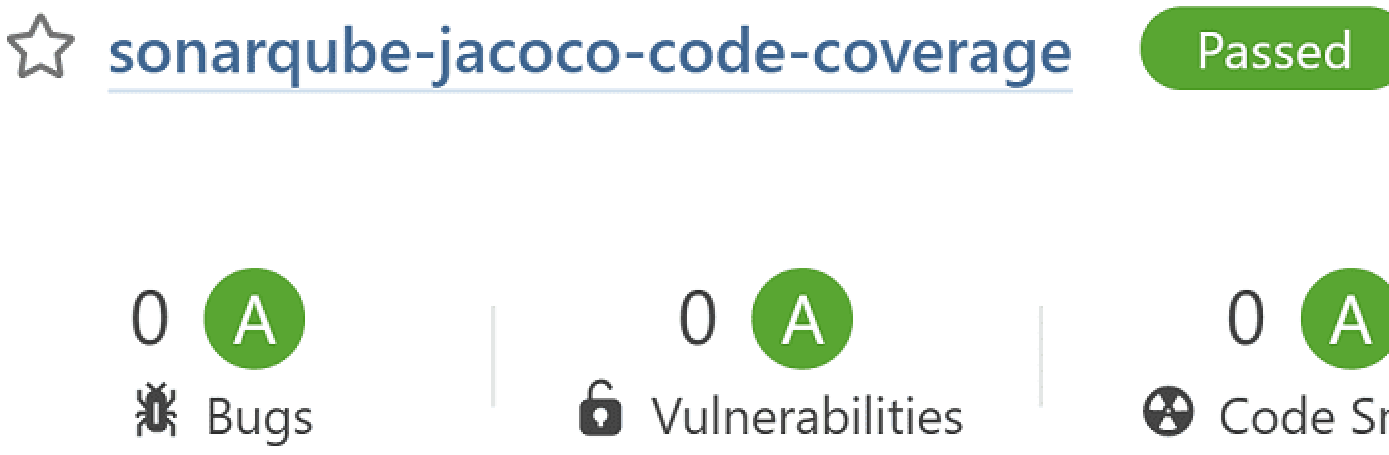
Yes, that's right, now it's time to run our pipelines! ✅

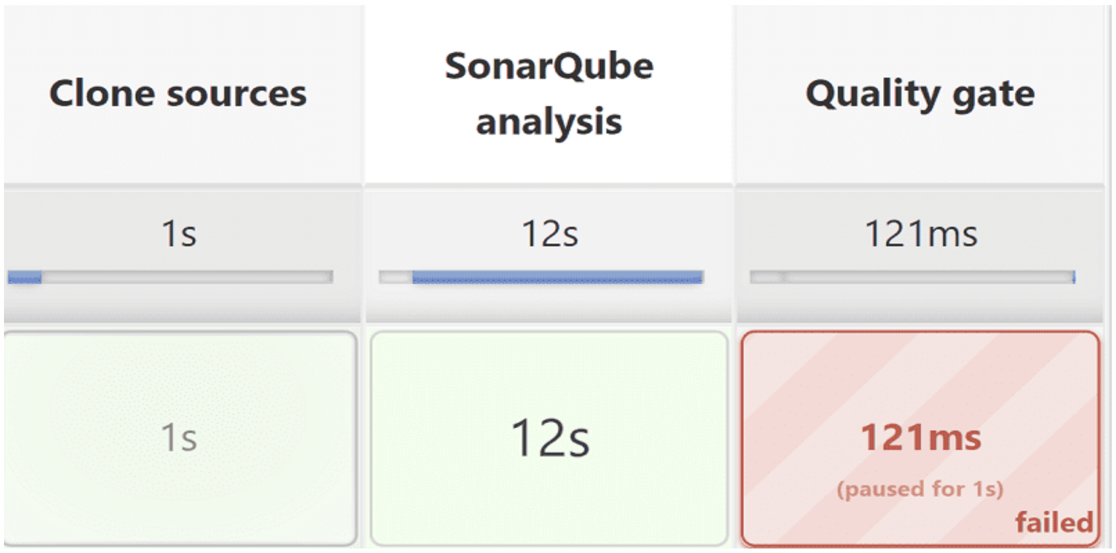
Let's run the sonarqube-good-code pipeline first.

You should get a build with all three stages passing.


Clone sources	SonarQube analysis	Quality gate
655ms	32s	352ms
655ms	32s	352ms (paused for 2s)

If we head over to SonarQube we can see that indeed our project has passed the quality gate.

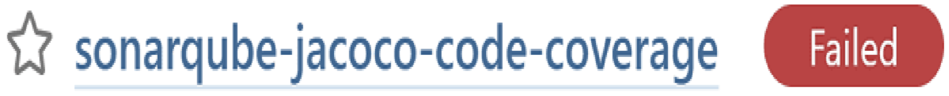


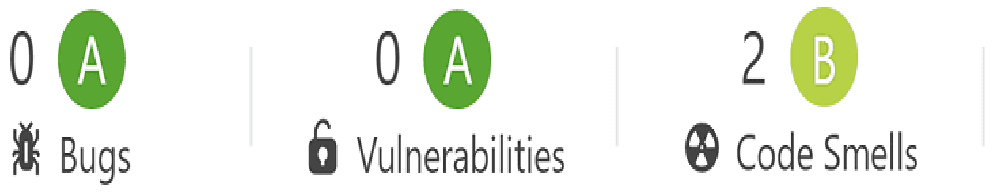


Now let's run the sonarqube-bad-code pipeline. Remember this is running against some really bad code!

You'll be able to see that the Quality gate stage of the pipeline has failed. Exactly what we wanted, blocking any future progress of this pipeline.

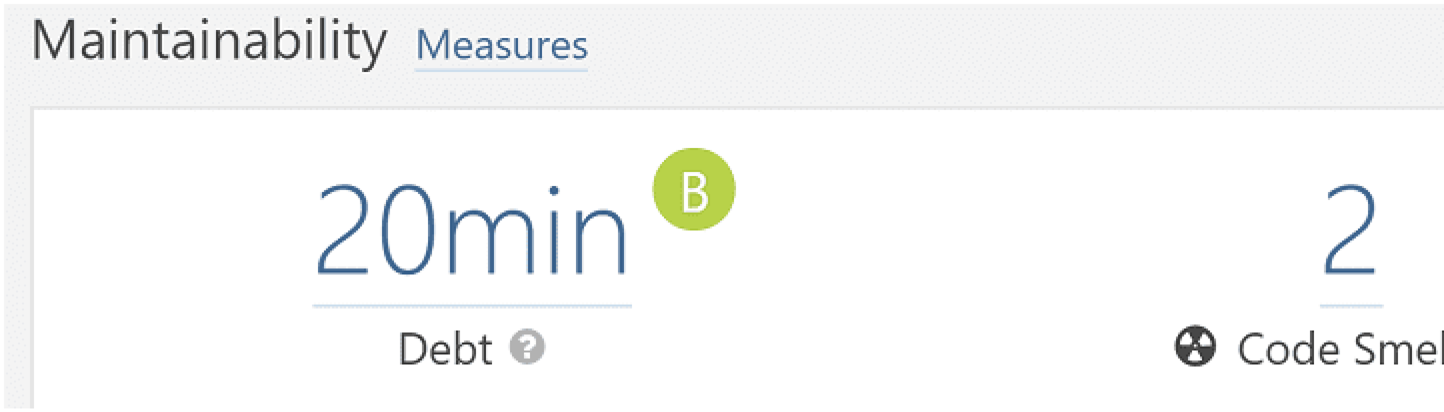
In the build's Console Output you'll see the message ERROR: Pipeline aborted due to quality gate failure: ERROR which shows that the pipeline failed for the right reason.





Over in SonarQube you'll see that this time it's reporting a Quality Gate failure.

Looks like we got some code smells on our hands!



Click on the project name for more details.

We can see that the maintainability rating has dropped to B because of the two code smells. This doesn't meet our quality gate, which requires a minimum A rating.

Final thoughts

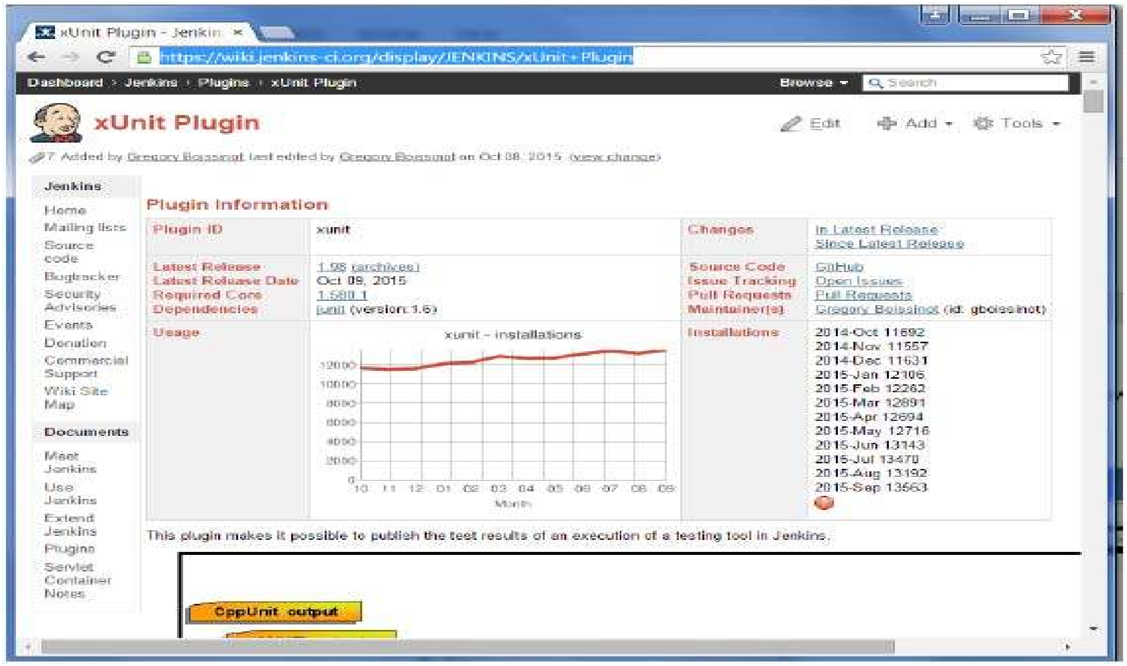
You've seen that integrating SonarQube quality gates into Jenkins is straightforward using the SonarQube Scanner Jenkins plugin. To apply this to a production setup, I suggest also to:

- remove the SONAR_FORCEAUTHENTICATION environment variable from SonarQube & configure the webhook in Jenkins to require an authentication token (see the SonarQube Scanner plugin configuration)
- consider running SonarQube analysis on feature branches, so developers get early feedback on whether their code changes are good before merging into master. However, multi-branch analysis does require a paid subscription to SonarQube.

For full details about setting up SonarQube analysis in a Gradle code project, see How To Measure Code Coverage Using SonarQube and Jacoco. If you're using Maven, check out this documentation from SonarQube.

Exercise 11:

Jenkins provides an out of box functionality for Junit, and provides a host of plugins for unit testing for other technologies, an example being MSTest for .Net Unit tests. If you go to the link https://wiki.jenkins-ci.org/display/JENKINS/xUnit+Plugin it will give the list of Unit Testing plugins available.



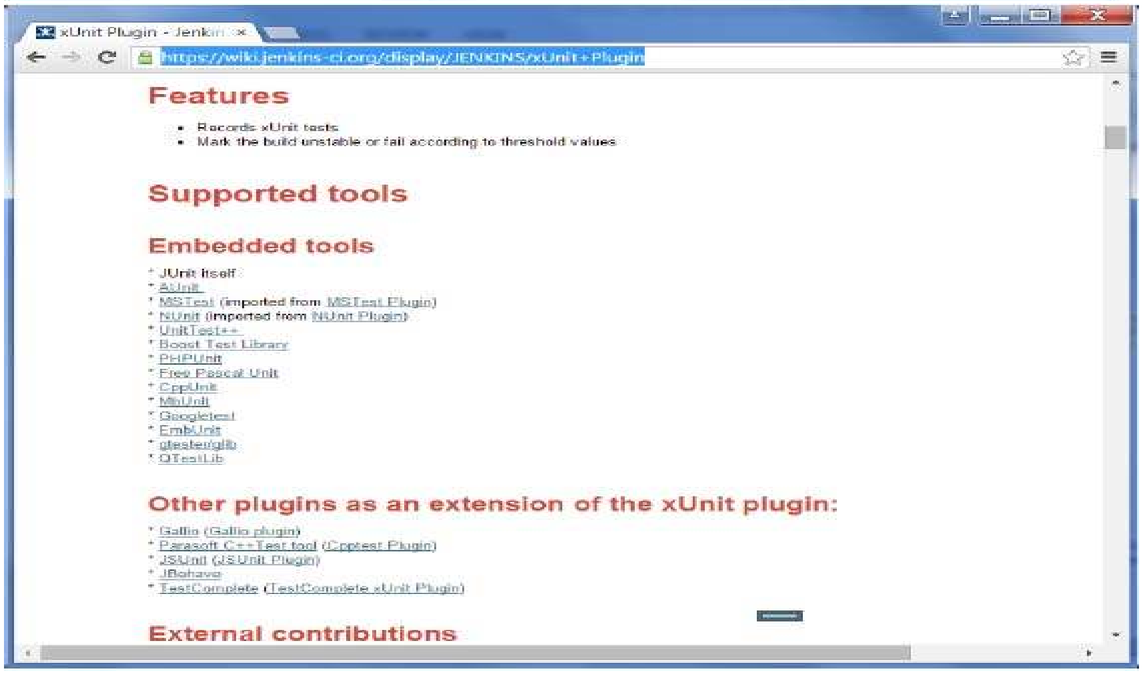
xUnit Plugin

7 Added by Gregory Boissinot, last edited by Gregory Boissinot on Oct 08, 2015 (view change)

Plugin Information

Plugin ID	xunit	Changes	In Latest Release Since Latest Release
Latest Release Latest Release Date Required Core Dependencies	1.98 (archives) Oct 09, 2015 1.580.1 junit (version: 1.6)	Source Code Issue Tracking Pull Requests Maintainer(s)	GitHub Open Issues Pull Requests Gregory Boissinot (id: gboissinot)
Usage	xunit - installations	Installations	2014-Oct 11692 2014-Nov 11557 2014-Dec 11631 2015-Jan 12106 2015-Feb 12262 2015-Mar 12891 2015-Apr 12694 2015-May 12716 2015-Jun 13143 2015-Jul 13470 2015-Aug 13192 2015-Sep 13563

This plugin makes it possible to publish the test results of an execution of a testing tool in Jenkins.



Features

- Records xUnit tests
- Mark the build unstable or fail according to threshold values

Supported tools

Embedded tools

* JUnit itself
* AUnit
* MSTest (imported from MSTest Plugin)
* NUnit (imported from NUnit Plugin)
* UnitTest++
* Boost Test Library
* PHPUnit
* Free Pascal Unit
* CppUnit
* MbUnit
* Googletest
* EmbUnit
* gtester/glib
* QTestLib

Other plugins as an extension of the xUnit plugin:

* Gallio (Gallio plugin)
* Parasoft C++Test tool (Cpptest Plugin)
* JSUnit (JSUnit Plugin)
* JBehave
* TestComplete (TestComplete xUnit Plugin)

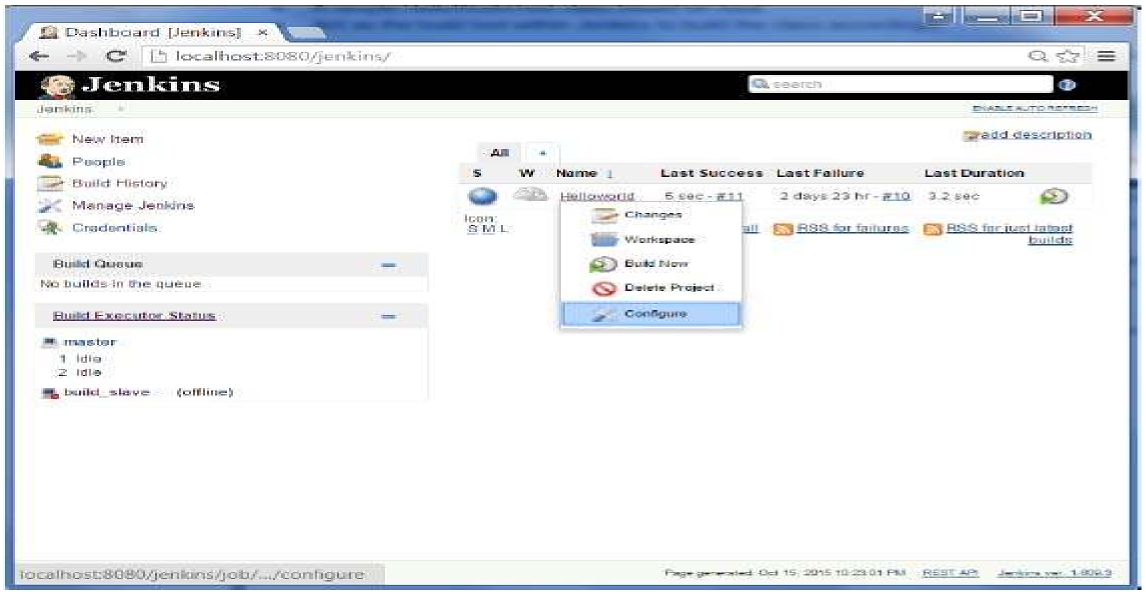
External contributions

Example of a Junit Test in Jenkins

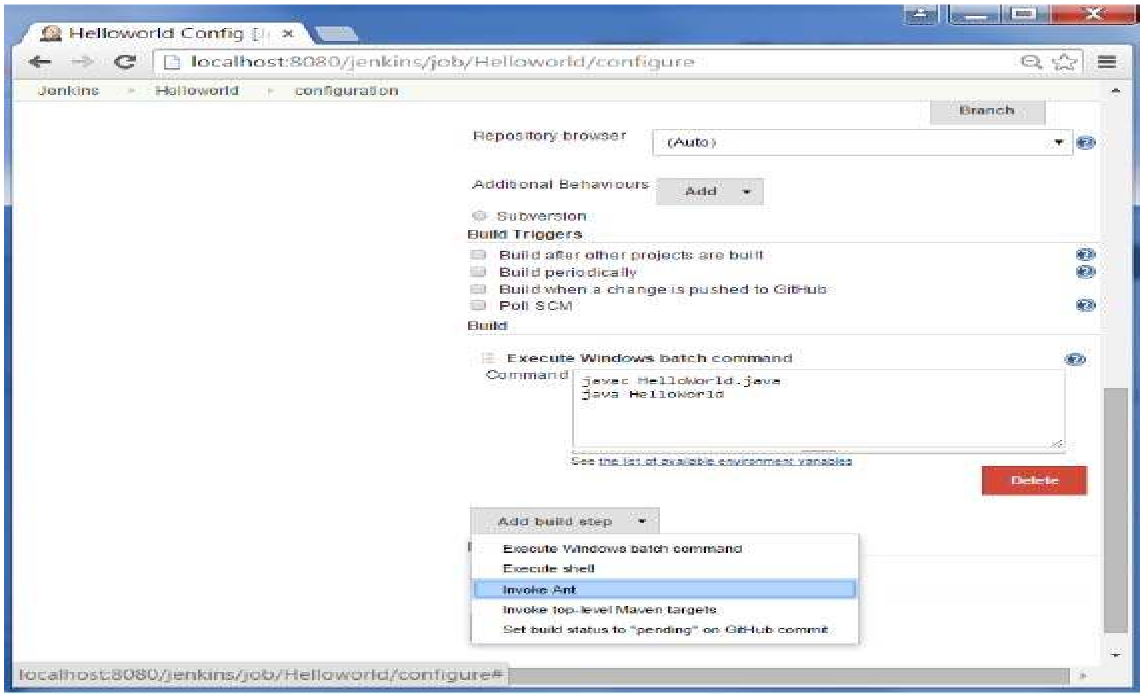
The following example will consider

- A simple HelloWorldTest class based on Junit.
- Ant as the build tool within Jenkins to build the class accordingly.

Step 1 − Go to the Jenkins dashboard and Click on the existing HelloWorld project and choose the Configure option

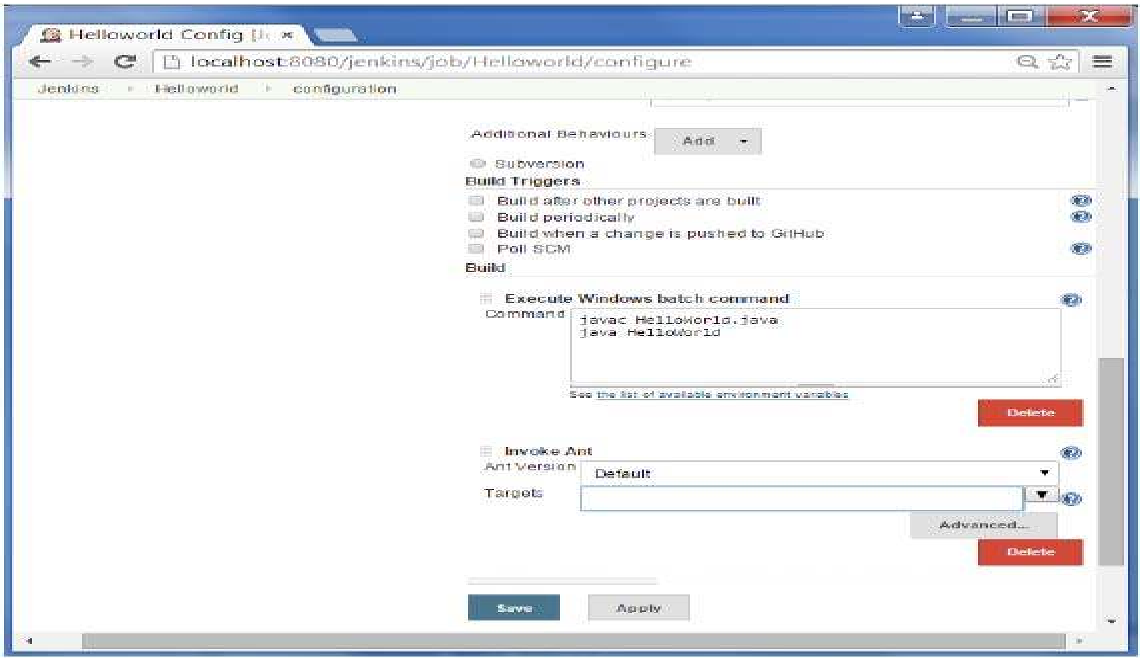


Step 2 − Browse to the section to Add a Build step and choose the option to Invoke Ant.


Helloworld Config
localhost:8080/jenkins/job/Helloworld/configure
Jenkins » Helloworld » configuration
Branch
Repository browser (Auto)
Additional Behaviours Add
Subversion
Build Triggers
Build after other projects are built
Build periodically
Build when a change is pushed to GitHub
Poll SCM
Build
Execute Windows batch command
Command
See the list of available environment variables
Delete
Add build step
Execute Windows batch command
Execute shell
Invoke Ant
Invoke top-level Maven targets
Set build status to "pending" on GitHub commit
localhost:8080/jenkins/job/Helloworld/configure#

```
javac HelloWorld.java
java HelloWorld
```

Step 3 − Click on the Advanced button.



Helloworld Config
localhost:8080/jenkins/job/Helloworld/configure
Jenkins › Helloworld › configuration

Additional Behaviours: Add
Subversion

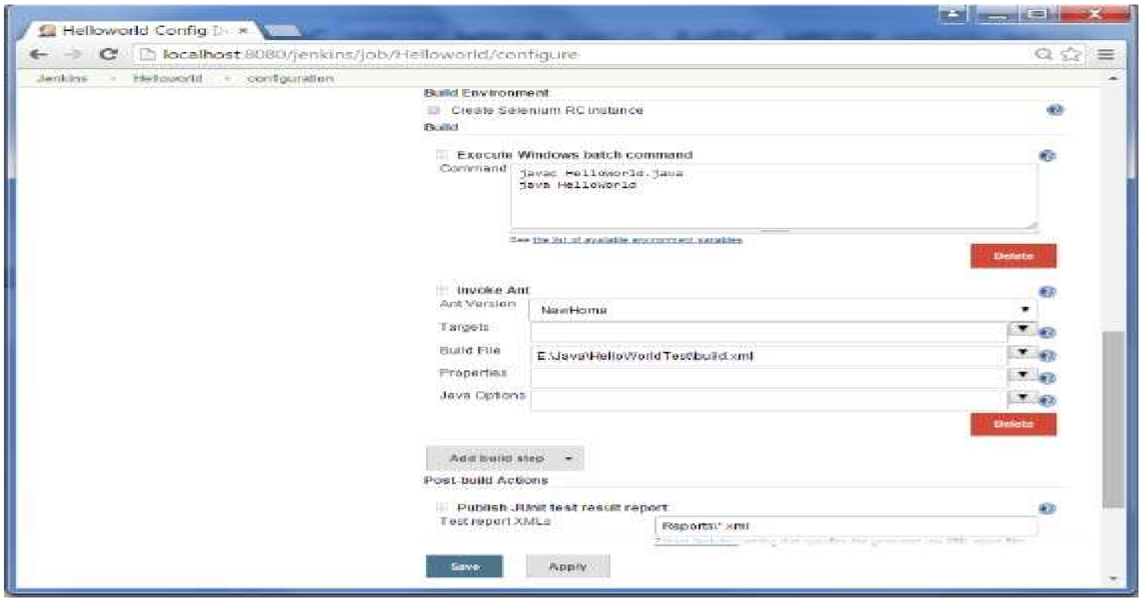
Build Triggers
- Build after other projects are built
- Build periodically
- Build when a change is pushed to GitHub
- Poll SCM

Build

Execute Windows batch command
Command:
```
javac HelloWorld.java
java HelloWorld
```
See the list of available environment variables

Invoke Ant
Ant Version: Default
Targets:

Step 4 – In the build file section, enter the location of the build.xml file.



Helloworld Config
localhost:8080/jenkins/job/Helloworld/configure
Jenkins › Helloworld › configuration

Build Environment
- Create Selenium RC instance

Build

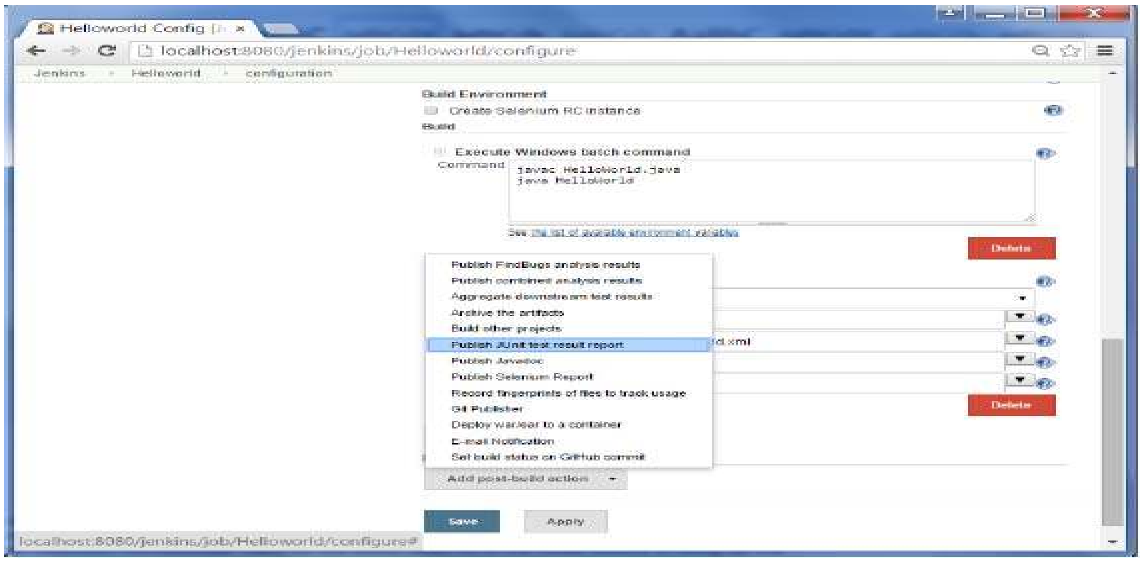
Execute Windows batch command
Command:
```
javac Helloworld.java
java Helloworld
```
See the list of available environment variables

Invoke Ant
Ant Version: NewHome
Targets:
Build File: E:\Java\HelloWorldTest\build.xml
Properties:
Java Options:

Post-build Actions

Publish JUnit test result report
Test report XMLs: Reports*.xml

Step 5 – Next click the option to Add post-build option and choose the option of "Publish Junit test result report"


Helloworld Config
localhost:8080/jenkins/job/Helloworld/configure
Jenkins
Helloworld
configuration
Delete
Publish FindBugs analysis results
Publish combined analysis results
Aggregate downstream test results
Archive the artifacts
Build other projects
Publish JUnit test result report
Publish Javadoc
Publish Selenium Report
Record fingerprints of files to track usage
Git Publisher
Deploy war/ear to a container
E-mail Notification
Set build status on GitHub commit
Delete
Add post-build action
Save
Apply
localhost:8080/jenkins/job/Helloworld/configure#

Build Environment

Create Selenium RC instance

Build

Execute Windows batch command

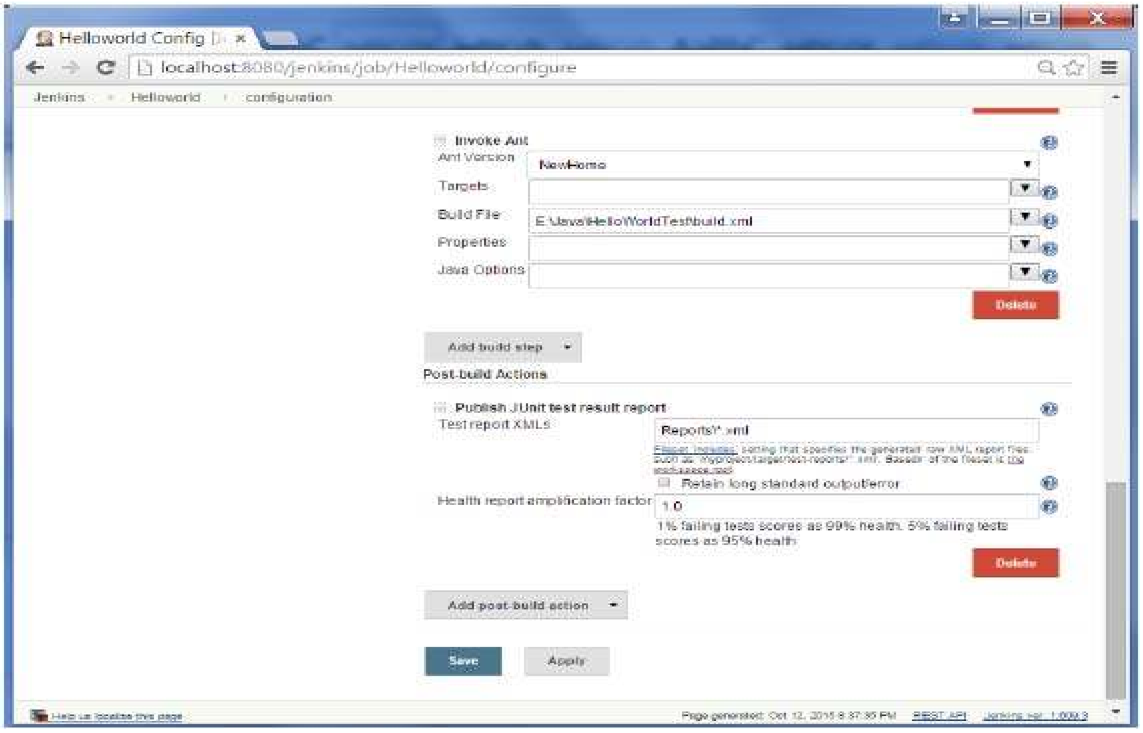
Command

```
javac HelloWorld.java
java HelloWorld
```

See the list of available environment variables

Step 6 − In the Test reports XML's, enter the location as shown below. Ensure that Reports is a folder which is created in the HelloWorld project workspace. The "*.xml" basically tells Jenkins to pick up the result xml files which are produced by the running of the Junit test cases. These xml files which then be converted into reports which can be viewed later.

Once done, click the Save option at the end.



Invoke Ant

Ant Version: NewHome

Targets

Build File: E:\Java\HelloWorldTest\build.xml

Properties

Java Options

Post-build Actions

Publish JUnit test result report

Test report XMLs: Reports*.xml

Fileset 'includes' setting that specifies the generated raw XML report files, such as 'myproject/target/test-reports/*.xml'. Basedir of the fileset is the workspace root.

Retain long standard output/error

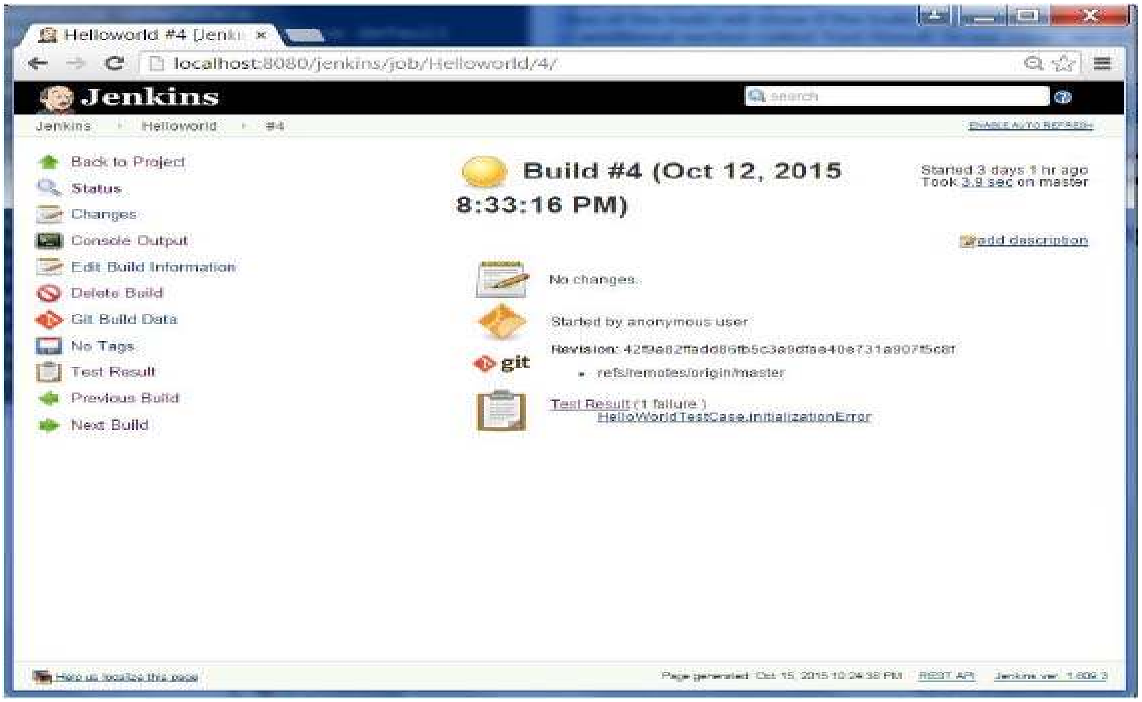
Health report amplification factor: 1.0

1% failing tests scores as 99% health. 5% failing tests scores as 95% health

Page generated: Oct 12, 2015 8:37:35 PM

Step 7 − Once saved, you can click on the Build Now option.

Once the build is completed, a status of the build will show if the build was successful or not. In the Build output information, you will now notice an additional section called Test Result. In our case, we entered a negative Test case so that the result would fail just as an example.



Build #4 (Oct 12, 2015 8:33:16 PM)

Started 3 days 1 hr ago
Took 3.9 sec on master

No changes.

Started by anonymous user

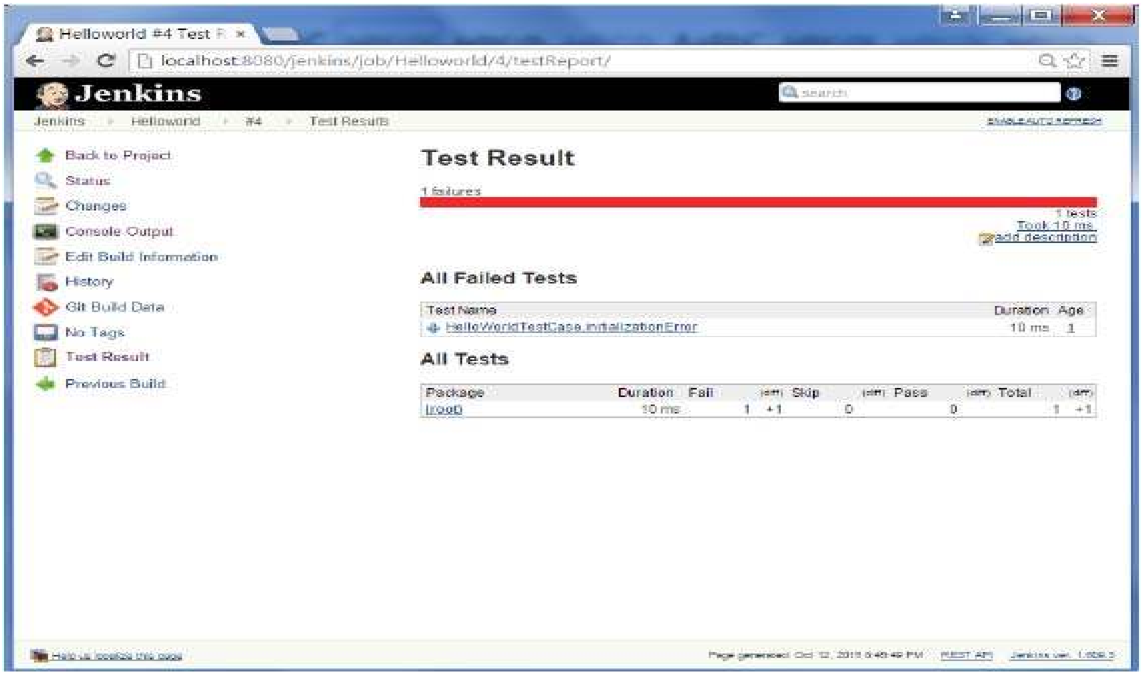
Revision: 42f9a82ffadd86fb5c3a9dfae40e731a907f5c8f

- refs/remotes/origin/master

Test Result (1 failure)
HelloWorldTestCase.initializationError

Page generated: Oct 15, 2015 10:24:38 PM
Jenkins ver. 1.609.3

One can go to the Console output to see further information. But what's more interesting is that if you click on Test Result, you will now see a drill down of the Test results.



Test Result

1 failures

1 tests
Took 10 ms.

All Failed Tests

Test Name	Duration	Age
HelloWorldTestCase.initializationError	10 ms	1

All Tests

Package	Duration	Fail	(diff)	Skip	(diff)	Pass	(diff)	Total	(diff)
(root)	10 ms	1	+1	0		0		1	+1

Page generated: Oct 12, 2015 8:48:49 PM
Jenkins ver. 1.609.3