



Hands-on Pipeline as YAML with Jenkins

A Beginner's Guide to Implement CI/CD Pipelines for Mobile,
Hybrid, and Web Applications Using Jenkins



MITESH SONI



Hands-on Pipeline as YAML with Jenkins

*A Beginner's Guide to Implement CI/CD Pipelines for
Mobile, Hybrid, and Web Applications Using Jenkins*

Mitesh Soni



www.bpbonline.com

FIRST EDITION 2021

Copyright © BPB Publications, India

ISBN: 978-93-90684-63-2

All Rights Reserved. No part of this publication may be reproduced, distributed or transmitted in any form or by any means or stored in a database or retrieval system, without the prior written permission of the publisher with the exception to the program listings which may be entered, stored and executed in a computer system, but they can not be reproduced by the means of publication, photocopy, recording, or by any electronic and mechanical means.

LIMITS OF LIABILITY AND DISCLAIMER OF WARRANTY

The information contained in this book is true to correct and the best of author's and publisher's knowledge. The author has made every effort to ensure the accuracy of these publications, but publisher cannot be held responsible for any loss or damage arising from any information in this book.

All trademarks referred to in the book are acknowledged as properties of their respective owners but BPB Publications cannot guarantee the accuracy of this information.

Distributors:

BPB PUBLICATIONS

20, Ansari Road, Darya Ganj

New Delhi-110002

Ph: 23254990/23254991

MICRO MEDIA

Shop No. 5, Mahendra Chambers,

150 DN Rd. Next to Capital Cinema,

V.T. (C.S.T.) Station, MUMBAI-400 001

Ph: 22078296/22078297

DECCAN AGENCIES

4-3-329, Bank Street,

Hyderabad-500195

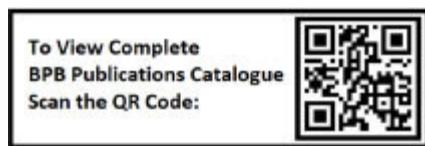
Ph: 24756967/24756400

BPB BOOK CENTRE

376 Old Lajpat Rai Market,

Delhi-110006

Ph: 23861747



Published by Manish Jain for BPB Publications, 20 Ansari Road, Darya Ganj, New Delhi-110002 and Printed by him at Repro India Ltd, Mumbai

www.bpbonline.com

Dedicated to

*Arpan, Kanak, Dada, Dadi, Shreyu, Mummy-Papa, Jigi-Nitesh,
Priyanka, Ruby and Mummy-Papa, Mayurand Vinay Kher*

About the Author

Mitesh is a DevOps engineer. He is in love with the DevOps culture and concept. Continuous improvement is his motto in life with existing imperfection. Mitesh has worked on multiple DevOps practices implementation initiatives. His primary focus is on the improvement of the existing culture of an organization or a project using Continuous Integration and Continuous Delivery. He believes that attitude and dedication are some of the biggest virtues that can improve professional as well as personal life! He has good experience in DevOps consulting, and he enjoys talking about DevOps and CULTURE transformation using existing practices and improving them with open source or commercial tools.

Mitesh always believes that DevOps is a cultural transformation, and it is facilitated by People, Processes, and Tools. DevOps transformation is a tools agnostic approach. He loves to give training and share knowledge with the community. He has a keen knowledge of programming, and he is aware of different languages/frameworks/platforms such as Java, Android, iOS, NodeJS, Angular. His main objective is to get enough information related to the project in a way that helps create an end-to-end automation pipeline.

In his leisure time, he likes to walk in Garden, to click photographs, and to do cycling. He prefers to spend time in peaceful places. His favorite tools/services for DevOps Practices implementation are Azure DevOps and Jenkins in commercial and open sources categories, respectively.

Mitesh has authored following books with BPB:

Hands-on Pipeline as Code with Jenkins

Hands-on Azure DevOps

Agile, DevOps, and Cloud Computing with Microsoft Azure

About the Reviewers

Aytunc Beken has more than 10 years of experience in CI/CD with using various technologies, mostly based on OpenSource Solutions and acquired Jenkins Engineer, Docker Associate and DevOps certificates. He has more than 8 years of experience on Jenkins and developed three Jenkins Plugins during this period.

Aytunc, graduated from Yıldız Technic University, Turkey. He worked for IBM, Turkcell and Authenteq and currently is working for Tesla as Senior DevOps Engineer.

James is currently the Platform Services Engineering Lead at Enable Midstream Partners and a Certified DevOps leader. In addition, He serves as an Ambassador for the DevOps Institute. James has spent his career looking for ways to eliminate bottlenecks and increase flow using value stream mapping, automation platforms and infrastructure as code. His background includes assisting and lead digital transformations at financial institutions, healthcare providers, education and oil and gas providers.

Acknowledgement

Ruby, every life has an amazing story. Thank you for being part of one of the BEST stories of my life. You are by far the most amazing, beautiful, loving, kind, and “ANGRY” woman in the world. I included that last one, so you knew that I was honest!

Special thanks to Dr. Ankit Chauhan, Dr. Kunal Aterkar, Dr. Mithun Natrajan, Devilal, Omprakash, Amit, Sanjay, Dr. Pushpa, Dr. Palak, Dr. Rajendra, Bhavna and Aarti, and other Medical Staff at Apollo Hospital International Limited (Gandhinagar - Ahmedabad Rd, GIDC Bhat).

I would like to thank Masi-Malav, my family members, Daksh-Parul Didi-Amit Jiju, Apoorva-Saurabh, Mayank Bhai and Bhabhi, Navrang, Dharmesh, Akkusss, Nalini and her Family, Anupama-Mihir and Priyanka-Hemant, Rohini, Yohan, Radhika, her Parents and Mukund, Ramya-Srivats, Radhika's all cousins, Piyushi, Prajakta – Keep Singing, Priyanka S, Gauri, Mitul, Bapu, Vimal, Ashish, Vijay, Rinka, Parinda, Arpita and her Family, Kim and Yaashi, Jai Jamba, Nitesh, Munal, Jyotiben, Niralee-Khushboo, Rohan C, Mayur, Chintan, Vijay, Nikul, Paresh, Raju, Yogendra, Jayesh and his family, Ramesh and his Family, Munni Bhabhi and her Family, Jyoti N, Bharti, Chitra

Madam, Kittu and Family, Aarohi, Poonam Aunty, Uncle, Laukik and Bhabhi, Oracle Team, Deepika, Aniket, Prasanna, Mahendra, Arvind, Dinesh, Viral, Chaitali, My Village, School and College Friends, and Teachers for being there always.

Special Thanks to Gowri-Arya, Sourabh Mishra, Bhavna, Amit R, Sid, Sudeep, Rita-Yashvi, Ajay, Sneha, Ankita, Palash for being there always.

Manpreet, Tarannum, Niralee, Vaishal, and all ICARE members with whom I shared some amazing memories ...Thank you so much for inspiring me to do good work.

I would like to thank BPB Team for giving me this opportunity to write my book for them.

Preface

Expectations change and hence the application changes over time based on the market, technical evolution, and other factors such as demands! Failures are detected too late and issues come at a time that is no point of return! To deal with the known issues of the traditional development approach, Agile development methods came into the picture. Integrative and Incremental development of features brings customer feedback into the development. Customers know what they are going to get after each iteration. It is not an approach where things are considered serially. A lot of communication and effective collaboration (arguably) takes place between stakeholders to understand requirements or explain expectations. All stakeholders are continuously involved. To meet the speed of the incremental and iterative model, automation is necessary to speed up things concerning application lifecycle management activities.

Here comes the challenge, manual processes bring delay in such an incremental and iterative approach. With agile, all issues and inefficiencies are magnified like never before. The question is how to automate? Can we start using some tools directly that automates application lifecycle management

activities? Is there something more to Automation than just tools? The answer is Yes!

DevOps, Continuous Practices, or DevOps Practices! DevOps is a cultural movement for transformation! It is a movement that gives human touch in the team with caring and sharing knowledge, enabling and empowering people, setting up effective and efficient processes for faster time to market with speed and quality. It helps to bring change and adopt a change smoothly in way of thinking and way of working. We will understand how issues and inefficiencies are magnified and how DevOps practices implementation help in Digital transformation or cultural transformation.

DevOps Practices implementation is popular in all customer discussions. At times, it is considered as a value add. The combination of people, processes, and tools brings the Culture change initiative to reality. DevOps pipeline or CI/CD pipeline is a popular word used. What does it mean? The pipeline includes different operations in different environments. Continuous Integration (CI) and Continuous Delivery (CD) are some of the most popular DevOps practices. Continuous Integration involves development, code analysis, unit testing, code coverage calculation, and build activities that are automated using various tools. Continuous Delivery is all about deploying your package into different environments, so

end-users can access it. There are different ways to create a pipeline/orchestration that involves Continuous Integration, Continuous Delivery, Continuous Testing, Continuous Deployment, Continuous Monitoring, and other DevOps Practices. Each tool provides different ways to create a pipeline.

In this book, we will discuss Jenkins. Jenkins is an open-source automation tool that offers an easy way to set up a CI/CD pipeline for almost any combination of languages, tools, and source code repositories. The Jenkins project was started in 2004 (originally Hudson). Initially, Jenkins was more popular for Continuous Integration, but after the release of Jenkins 2.0, Jenkins is utilized to automate and orchestrate the entire Application Management Lifecycle. Jenkins is the first project to graduate in the Continuous Delivery Foundation. The Jenkins community offers more than 1,500 plugins that empower Jenkins users to create orchestration by integrating almost all popular tools in different categories.

A pipeline is a most talked about the concept that refers to the groups of stages or events or jobs that are connected in a specific sequence with sequential or parallel execution flow based on the use case. There are different ways to create a Pipeline in Jenkins. In this book, we will create Pipeline using YAML, we will also discuss Multibranch pipeline that facilitates

to create different Jenkinsfiles for different branches in the branches. Jenkins will automatically discover branches where Jenkinsfile is available. Jenkinsfile contains YAML or Scripted or Declarative Pipeline. It is managed in Version Control and hence easy to maintain. The main objective of this book is to provide an easy path for beginners who want to create YAML Pipeline for programming languages such as Flutter, Android, AngularJS, and Ionic Cordova. Each chapter on a specific programming language focuses on Pipeline that includes Static Code Analysis using SonarQube or Lint tools, Unit test execution, calculating code coverage, publishing unit tests and code coverage in uniform reports, verifying the quality of code coverage with Quality Gate, creating build/package, and distributing package to a specific environment based on the type of programming languages. Over the seven chapters in this book, you will learn the following:

[Chapter 1](#) introduces all the areas that encompass the field of DevOps and DevOps Practices implementation, Overview of Jenkins, different types of Pipelines such as Build pipeline with plugin, Scripted pipeline, Declarative pipeline, Blue Ocean, and YAML pipeline and its evolvement over the years.

[Chapter 2](#) discusses the different components of the YAML pipeline in Jenkins. It focuses on Controller-Agent Architecture and pipeline structure.

[Chapter 3](#) introduces how to implement Continuous Integration, and Continuous Delivery for Hybrid Mobile Apps developed in Flutter using YAML Pipeline as a Code in Jenkins. It covers multi-stage pipeline for Hybrid Mobile applications for tasks such as flutter doctor, Code analysis, Unit test Execution and Code coverage, Build Quality Check, distributing App package to App center.

[Chapter 4](#) introduces how to implement Continuous Integration, and Continuous Delivery for Hybrid Mobile Apps developed in Ionic Cordova using YAML Pipeline as a Code in Jenkins. It covers multi-stage pipeline for Hybrid Mobile applications for tasks such as Static code analysis, Unit test Execution and Code coverage, Build Quality Check, distributing App package to the App center.

[Chapter 5](#) introduces how to implement Continuous Integration and Continuous Delivery for Android Application using YAML Pipeline as a Code in Jenkins. This chapter provides step by step detail to create Multi-Stage Pipeline for Android App, how to import a Repository, how to perform Lint Analysis for Android application, execute Unit Tests, calculate Code Coverage, verify Build Quality, create APK file, and configure Continuous Delivery by deploying Package/APK to App Center.

[Chapter 6](#) covers how to implement Continuous Integration and Continuous Delivery for Angular Application using YAML Pipeline as a Code in Jenkins. It covers step by step instructions for a multi-stage pipeline that includes Junit and Cobertura configuration in karma.conf.js, Lint, Unit tests, and Code Coverage configuration in Package.json, Unit Tests and Code Coverage calculation, End to End Test Execution, and NPM Audit, verify Build Quality, and deploy Angular application to docker container.

[Chapter 7](#) covers best practices to implement DevOps Practices using Jenkins and also best practices on how to utilize Jenkins effectively.

Downloading the coloured images:

Please follow the link to download the
Coloured Images of the book:

<https://rebrand.ly/c5e8b5>

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

Did you know that BPB offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.bpbonline.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at business@bpbonline.com for more details.

At you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on BPB books and eBooks.

BPB IS SEARCHING FOR AUTHORS LIKE YOU

If you're interested in becoming an author for BPB, please visit www.bpbonline.com and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

The code bundle for the book is also hosted on GitHub at In case there's an update to the code, it will be updated on the existing GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at Check them out!

PIRACY

If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please

contact us at business@bpbonline.com with a link to the material.

IF YOU ARE INTERESTED IN BECOMING AN AUTHOR

If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit

REVIEWS

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at BPB can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about BPB, please visit

Table of Contents

1. Introducing Pipelines

Structure

Objectives

What is DevOps?

DevOps history

Benefits of DevOps

DevOps practices

Continuous code inspection

Continuous integration (CI)

Cloud computing and containers

Continuous delivery

Continuous testing

Continuous deployment

Introducing Jenkins

History of Jenkins

Overview of Jenkins

Prerequisites

How to run Jenkins?

Pipelines

Build pipeline

Scripted pipeline

Declarative pipeline

Blue Ocean

Pipeline as a YAML

Conclusion

Points to remember

Multiple choice questions

Answer

Questions

Key terms

2. Basic Components of YAML Pipeline

Structure

Objectives

Controller-agent architecture

Agent pools

YAML pipeline structure

Conclusion

Multiple choice questions

Answer

Questions

3. Building CI/CD Pipeline with YAML for Flutter Application

Structure

Objectives

Multi-stage CI/CD pipeline for Flutter app

Continuous integration for Flutter app

Continuous delivery for Flutter app

YAML pipeline script for Flutter app

Conclusion

Multiple choice questions

Answer

Questions

4. Building CI/CD Pipeline with YAML for Ionic Cordova Application

Structure

Objectives

Multi-stage CI/CD pipeline for Ionic Cordova app

Continuous integration for Ionic – Android App

Continuous delivery for Ionic Cordova App

YAML pipeline script for Ionic Cordova app

Conclusion

Multiple choice questions

Answer

Questions

5. Building CI/CD Pipeline with YAML for Android App

Structure

Objectives

Introduction

Multi-stage CI/CD pipeline for Android app

Continuous integration for Android App

Understand how to perform Lint analysis for Android application

Execute unit tests and calculate code coverage

Continuous delivery for Android app

[YAML pipeline script for Android app](#)

[Conclusion](#)

[Multiple choice questions](#)

[Answer](#)

[Questions](#)

[6. Building CI/CD Pipeline with YAML for Angular Application](#)

[Structure](#)

[Objectives](#)

[Introduction](#)

[Multi-stage CI/CD pipeline for Angular app](#)

[Continuous integration for Angular App](#)

[Junit and Cobertura configuration in karma.conf.js](#)

[Lint, unit tests, and code coverage configuration in Package.json](#)

[Continuous delivery for Angular app](#)

[YAML pipeline script for Angular app](#)

[Conclusion](#)

[Multiple choice questions](#)

[Answer](#)

[Questions](#)

[7. Pipeline Best Practices](#)

[Structure](#)

[Objectives](#)

[Best practices](#)

[Easy installation with fault tolerance](#)

[Install Jenkins using Docker](#)

[Dockerfile agent in declarative pipeline](#)

[Install Jenkins on Azure Kubernetes Services \(AKS\)](#)

[Install Jenkins on Amazon Elastic Kubernetes Service \(Amazon EKS\)](#)

[Always secure Jenkins](#)

[Project-based security](#)

[Pipeline – best practices](#)

[Backup and restore](#)

[Monitoring](#)

[Conclusion](#)

[Multiple choice questions](#)

[Answers](#)

[Questions](#)

[**Index**](#)

CHAPTER 1

Introducing Pipelines

DevOps, Continuous Practices, or DevOps Practices! DevOps is a cultural movement for transformation! It is a movement that gives human touch in the team with caring and sharing knowledge, enabling and empowering people, setting up effective, and efficient processes for faster time to market with speed and quality.

In this book, we will focus on Jenkins and different types of pipelines we can create in Jenkins. Jenkins 2.0 provides new features such as Pipeline as code – technical aspect, a new setup experience, and other UI improvements – enhancements with Jenkins interface. The entire user experience had a drastic change. Easy navigation to a different section in the job configuration is an eye-catching difference.

In this chapter, we will focus on an overview of Jenkins and also pipeline evolution from Build Pipeline, Scripted Pipeline, Declarative Pipeline, Blue Ocean, and Pipeline as a YAML. In this book, we will create CICD pipeline using YAML.

Structure

In this chapter, we will discuss the following topics:

What is DevOps?

Benefits of DevOps

DevOps practices

Pipelines

Build pipeline

Scripted pipeline

Declarative pipeline

Blue Ocean

Pipeline as a YAML

Objectives

This chapter introduces all the areas that encompass the field of DevOps and DevOps practices implementation, an overview of Jenkins, Azure DevOps, different types of pipelines, and its evolution over the years. After studying this unit, you should be able to:

Understand the concept of DevOps

Discuss the types of Continuous Practices such as Continuous Integration and Continuous Delivery

Understand the importance of Mindset, People, Processes, and Tools

Discuss the different DevOps practices

Understand prerequisites before installing Jenkins

How to install Jenkins

How to change Port and JENKINS_HOME

Basic installation and configuration

Best practices of Jenkins configuration

What is DevOps?

DevOps is a disruptive shift in how to manage the mindsets of people, ways of working, and the **Application Lifecycle Management**. It is a process of transforming culture rather than implementing tools or a tool-specific approach. DevOps is all about culture transformation using a combination of people, processes, and tools. DevOps is known to be associated with practices such as continuous code inspections, **continuous integration continuous delivery**, continuous testing, continuous monitoring, continuous feedback, continuous improvement, and continuous innovation.

Now, DevOps practices also accommodate infrastructure provisioning in the cloud, infrastructure as a code, configuration management, and pipeline as a Code too.

DevOps history

This flow is based on my exploration and experience. That is how I realized how we started our DevOps culture transformation:

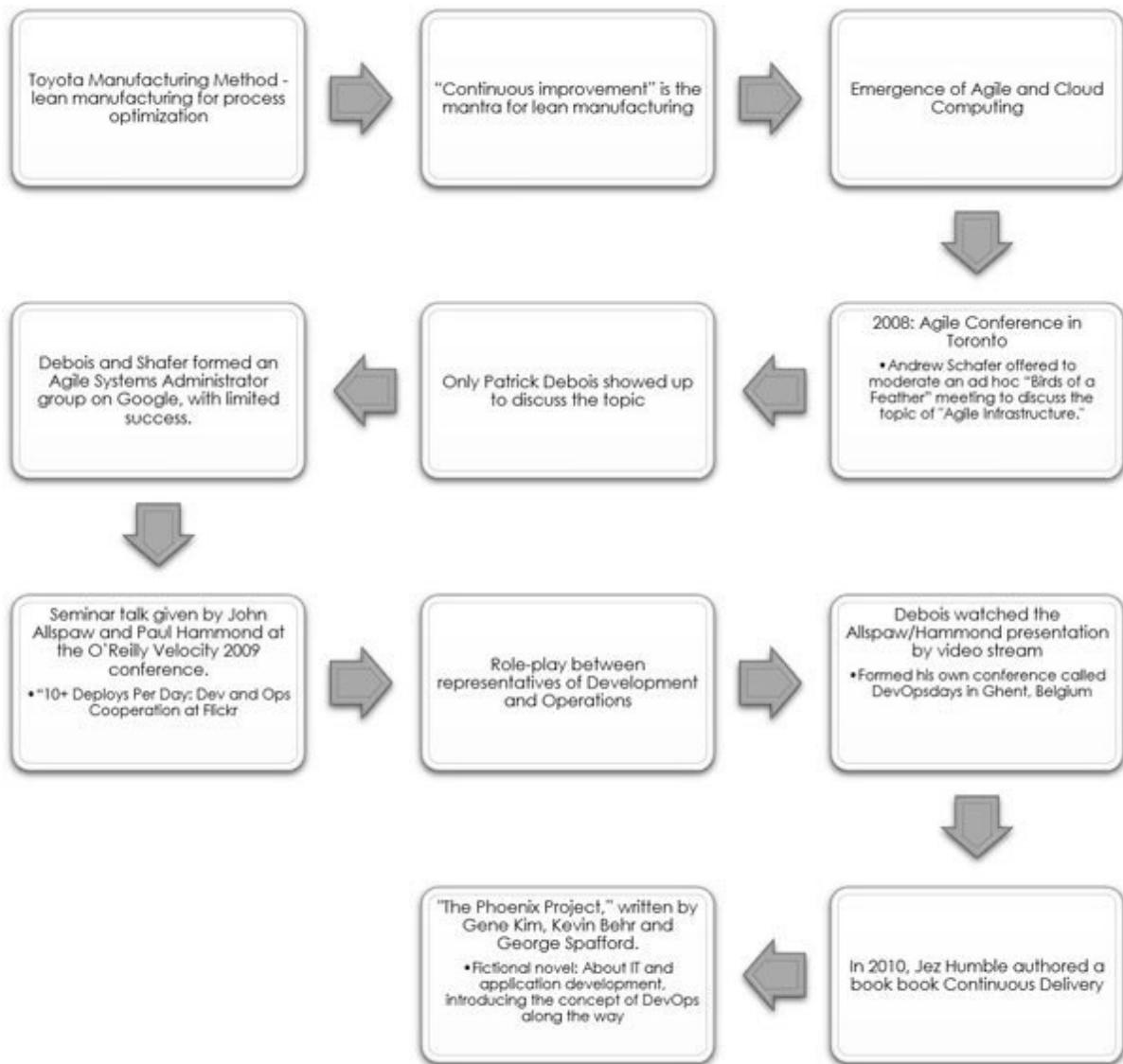


Figure 1.1: DevOps history

Now, let us define DevOps. It is a part of the quality-related discussion in organizations. In simple language, DevOps is all about effective collaboration and communication between different stakeholders such as development, operations, quality team, and security team for better quality, and faster time to

market. There are no set of guidelines or methodologies available for DevOps practices implementation:

Let us first understand the responsibilities of Dev and Ops teams before defining DevOps:

DevOps:
DevOps: DevOps: DevOps: DevOps: DevOps: DevOps: DevOps:
DevOps: DevOps: DevOps: DevOps: DevOps: DevOps: DevOps:
DevOps: DevOps: DevOps: DevOps: DevOps: DevOps: DevOps:
DevOps: DevOps: DevOps: DevOps: DevOps: DevOps: DevOps:
DevOps: DevOps: DevOps: DevOps:

Table 1.1: Dev and Ops responsibilities

Based on existing culture, implementation roadmaps may differ from organization to organization as **NO** two organizations can have the same culture. DevOps is a culture that consists of continuous practices that help to achieve faster time to market for an application with the highest quality by Continuous Improvement and Continuous Innovation by involving People, Processes, and Tools.

Organizational culture can be affected by cognitive biases as it is all about changing culture, and hence resistance is inevitable. In this chapter, we will have a look at some cognitive biases and their solution at some intervals with no specific context.

DevOps practices implementation needs visibility. It requires vision and transparency and acceptance of the AS-IS scenario. Hence, it is important to assess the ground situation in the organization and derives a roadmap for effective adoption of the roadmap of this cultural transformation journey:

DevOps transformation depends on three major aspects:

People

Processes

Tools

DevOps is all about People, Processes, and Tools. It is important to note that people still have a high priority. People have a different notion about what DevOps is. Some people are not even aware of it. People are tightly coupled with the existing culture, and hence transformation is difficult

Each organization has different cultures and different sets of practices, making it challenging to change things until people change their attitude and mindset.

DevOps is a culture and succeeds in an environment that is Tools Agnostic. Following are some of the evaluation criteria when we want to select tools to implement DevOps practices: ease of use, learning curve, distributed architecture, extensibility using plugins, features based on technology evolution, compatibility with the existing culture, and future roadmap.

All tools provide similar kinds of features and functionality. It is all up to the organization and existing culture of an organization to make sure how the tool fits in the requirements. Nowadays, all tools provide integration with popular tools available in the market with the use of plugins or extensions.

Benefits of DevOps

Let us understand business and technical benefits realized by multiple organizations after culture transformation achieved using DevOps practices:

practices: practices:

practices: practices: practices: practices: practices: practices:
practices: practices: practices: practices: practices: practices:
practices: practices: practices: practices: practices: practices:
practices: practices: practices: practices: practices: practices:
practices: practices:

Table 1.2: Business and technical benefits

DevOps practices

DevOps practices help to improve activities that are blocking the flow of products from development to deployment. It is important to identify bottlenecks in the entire value stream and fix them using culture transformation activities and automation. We need to identify and assess the existing situation; we need to find bottlenecks and we should try to fix them.

Continuous practices play an important role in cultural transformation. [Figure 1.2](#) indicates some if not all, continuous practices:

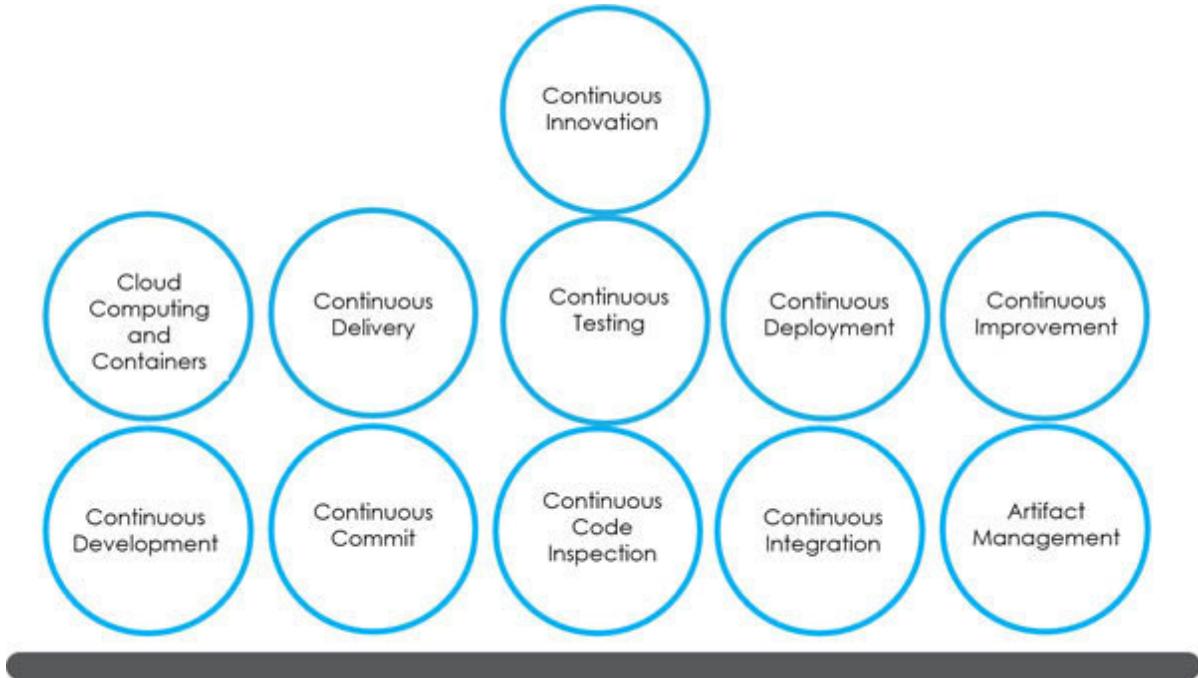


Figure 1.2: Continuous Practices

What are the activities and operations involved in different DevOps practices at a high level? Let us understand the above continuous practices that we are going to cover in this book using the Jenkins pipeline.

Continuous practice implementation is the best in phase-wise as prioritized by VSM or Assessment workshop. It helps to understand the benefits of automation and make the entire process stable based on continuous learning!

Let us start our small journey to understand continuous practices in brief. In the next section, we will discuss continuous code inspection.

Continuous code inspection

Continuous code inspection allows performing static code analysis (SCA) and highlights bugs, vulnerabilities, and other code issues related to standards. Quality profiles and quality gates can be integrated with a CI server. In short, **SCA** helps to analyze the code without running it.

Code analysis is the first step of verification. Good quality of code reduces failures at later stages. It is easier to manage and maintain code. Once code is reviewed and verified, the next logical step is to perform compilation, unit test execution, and creating a package.

Author's Code quality is often neglected until it is driven or monitored by customers or it is automated using tools and quality gates are configured. At times, development teams think that code issues highlighted by the code analysis tools are extra work, and it should be done away with. Hence, it is useful to give training to the development team on the importance of the code quality and make sure that code issues are fixed in a phase-wise manner. If the development team is forced to fix issues in one go then there will be resistance due to work commitments.

In the next section, we will discuss continuous integration.

Continuous integration (CI)

CI is the DevOps practice to integrate new feature implementation or a bug fix in the existing distributed version control systems such as Git that triggers code quality checks, unit test execution, and build. CI is easy to implement, and hence for DevOps starters, it works as a solid base on which the rest of DevOps practices can stand.

CI helps in communication and collaboration between different stakeholders. It helps to detect issues faster and if notifications are configured, it is helpful in better communication. Branch policies or pull requests are also one of the better ways to collaborate effectively.

The main objective of this book is to keep the uniform format of all reports such as Unit test results and code coverage reports. Published results should be in the same format irrespective of programming languages. We have successfully demonstrated these capabilities using plugins available in Jenkins. Same types of reports help to analyze all different types of the application easily and hence we focused on uniform reports irrespective of code coverage tools such as Jacoco, Cobertura, or XCCov.

Once the CI stage is completed successfully, the next logical step is to deliver or deploy it into the specific environment such as dev, test, stage, or production. Environments can contain resources such as physical, virtual, cloud, containers, or hybrid based on the need.

Author's As per my experience, CI is relatively easy to implement among other DevOps practices and face less resistance. There are times when teams are not aware of build tools and why it is required. Again, training helps the development team to understand the importance of automated build, automated unit tests, code coverage, and artifact management. It is important to make it flexible and easy so the development team can use it with proper authentication and authorization.

In the next section, we will cover details about cloud computing and containers.

Cloud computing and containers

Cloud computing is a disruptive innovation in recent times. As per the **National Institute of Standards and Technology** definition, there are following aspects of cloud computing:

Cloud deployment There are four cloud deployment models:

Public cloud: We can access public cloud services over the Internet. Anyone who has a credit card or enterprise account or Free tier can access public cloud services.

Private cloud: Private cloud is the cloud deployment model that serves the need for a single organization that builds it and owns it. A private cloud is built behind the organization's firewall, and complete infrastructure is in control of an organization.

Hybrid cloud: Hybrid cloud is a cloud deployment model that has a mixture of other cloud deployment models such as public cloud and private cloud. The main objective behind this design is to keep business-critical applications and data on-premises (behind firewall) in a more secure environment or

perceived to be a more secure environment such as private cloud and rest of the components on a public cloud.

Community cloud: Community cloud is a deployment model where resources are shared among the organizations who share common interests or security requirements or compliance requirements or performance requirements.

Essential characteristics of cloud There are five essential characteristics:

On-demand self-service

Broad network access

Resource pooling

Rapid elasticity

Measured service

Now, let us talk about containers in brief. A container can package application code, libraries, and configurations. The container engine is installed on the host OS. Hence, all

containers share a single host, and that can be a security threat. However, all containers are run as isolated processes and managed by a container engine. Many containers can run on a single host operating system. Docker is an open-source tool to create, deploy, and manage containers on a different host operating system using resource isolation features, such as cgroups and Linux kernels.

In this book, we have used different types of environments, and step-by-step description is available on how to create an environment using cloud or container resources. We have used Platform as a Service offering from Microsoft Azure cloud for web application deployment while in some cases we have used docker containers and Kubernetes as well. Detailed discussion on cloud, containers, and Kubernetes is out of the scope of this book.

Most of the automation servers such as Jenkins or Azure DevOps provide support to integrate cloud environment and containers. Based on no. of environments and teams, it is difficult to track and maintain artifacts. It is also important to make security and governance a priority while managing these artifacts. Artifact repository helps to manage different versions of artifacts effectively. It also provides integration with different automation tools such as Jenkins, Atlassian Bamboo, Azure

DevOps, and so on. Automated deployment and rollback are important practices in the DevOps culture.

Author's As per my experience, cloud resources and containers are still not utilized heavily in organizations. Especially, service organizations. However, COVID 19 pandemic has changed that cloud computing and containers are at forefront of the discussions and it makes things much easier with elastic resources and pay per use pricing model.

In the next section, we will cover details about CD.

Continuous delivery.

The objective of CD and continuous deployment is to deploy an application in Dev, Test, UAT, Production environment in an automated manner.

In this book, we have covered CD extensively in most of the chapters. We have covered web application deployment to Azure App Services, Docker containers, and Azure Kubernetes Services, while in the case of mobile and hybrid applications, we have distributed app packages to the app center using plugins available in Jenkins or using command-line tools that are supported by Cloud Service CLIs.

Once artifacts are deployed in a specific environment, the next logical step is to verify and validate for the expected outcome. Testing can be performed in different environments based on the existing processes and policies of an organization.

Author's As per my experience, CD with an automated script or some popular tools such as IBM Urbancode deploy or app center are very popular among teams. It helps specific people in the team whose extra responsibility is to oversee

deployment activities. It usually takes 15–30 min and considers multiple deployments in a day to multiple systems. Automated operations in such cases are useful and time-saving, and I have seen some smart developers doing it to save themselves from manual work.

In the next section, we will cover details about continuous testing.

Continuous testing

Continuous testing helps to verify the functional aspects of an application in an automated manner and to keep the application production-ready after all verifications. In an agile world, speed and quality matter, while the need for verification and validation has exponentially increased. In this scenario, manual testing no longer satisfies those essential requirements. Continuous testing is the answer to speed and quality for testing. Manual testing is still used in some cases where automation is not feasible.

Continuous testing is the process of executing automated tests as a part of the software delivery pipeline to obtain feedback on the business risks associated with a software release candidate as rapidly as possible. Automated testing helps to accelerate time to market and quality of the product. Once the application is verified and validated, the final step is to deploy an application in the production environment.

Author's As per my experience, continuous testing is one of the difficult changes in the transformation exercise. Teams are not writing automated unit test cases willingly and again it is considered as an extra work. There is a lack of skilled

resources for functional test automation. However, stakeholders are realizing the value of automated testing, and it is now being discussed earlier than before.

In the next section, we will discuss continuous deployment.

Continuous deployment

What is the difference between CD and continuous deployment? Let us understand in Table

CD	Continuous Deployment
Automated	Automated
Build	Build
Test	Test
Deploy	Deploy

Table 1.3: Continuous delivery vs. continuous deployment

Once the application is deployed in the production environment, it is important to monitor the environment as well as application performance. Continuous feedback from the monitoring systems helps to improve the application and its runtime environment continuously.

There are many automation tools available that help us to create automation pipelines or CICD pipelines. Jenkins is one of the most popular automation servers, and it has evolved a lot in recent years.

Author's As per my experience, continuous deployment is usually not in control. It is mostly controlled by customers and hence automated deployment is hardly a scenario. However, the automated script for CD can be replicated in production deployment if it is allowed and feasible.

In the next section, we will discuss Jenkins and different ways to create pipelines in Jenkins.

Introducing Jenkins

Let us start by getting some information about Jenkins. The first question is: What is Jenkins?

History of Jenkins

Let us have a quick look at the history of Jenkins and how it evolved over the years including its current footprint:

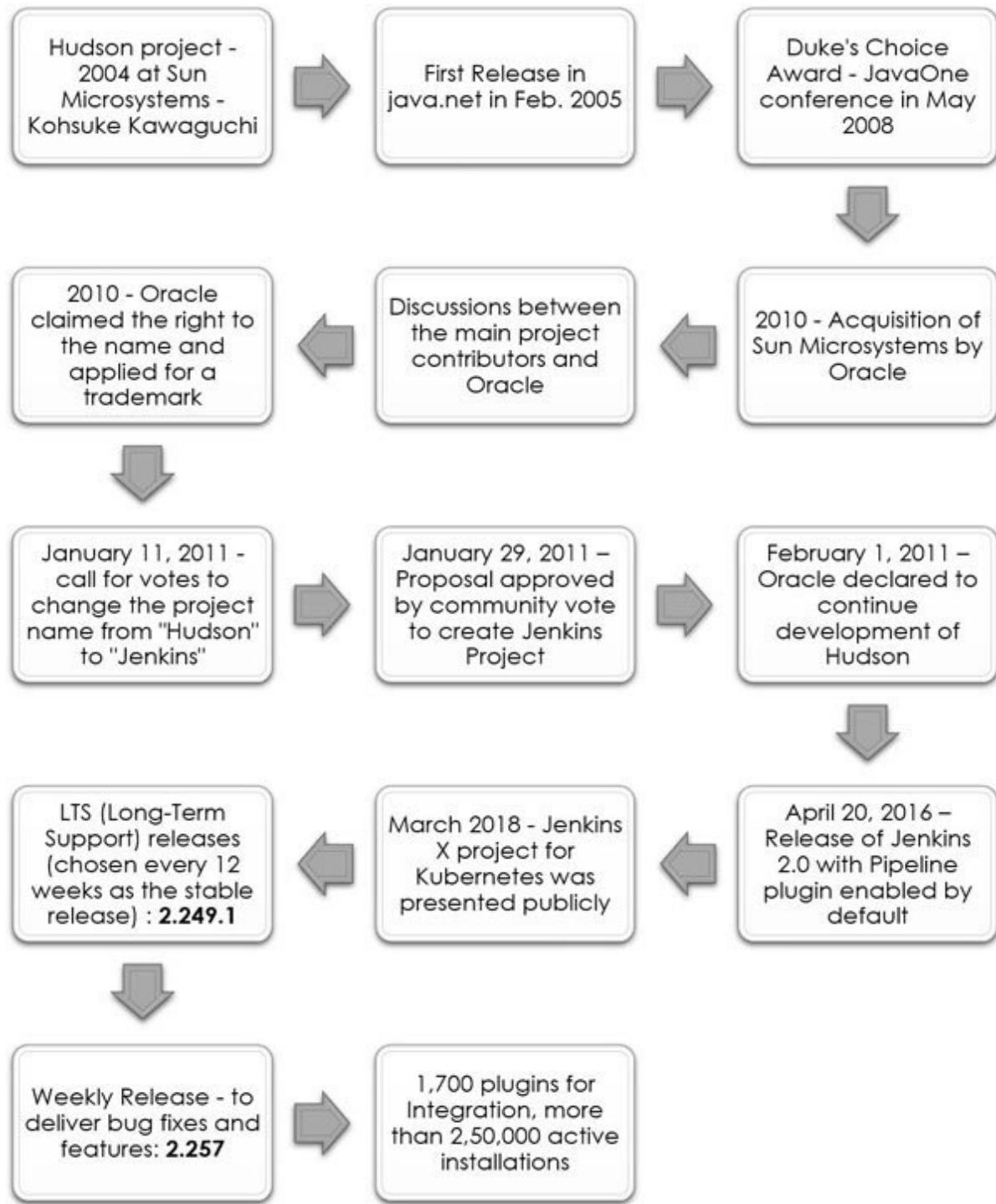


Figure 1.3: Jenkins history

Jenkins provides a simple way to create a pipeline that includes CI and CD.

Let us understand some more details about Jenkins.

Overview of Jenkins

Jenkins is an open-source tool that provides integration with existing tools used in application lifecycle management to automate the entire process based on feasibility.

Jenkins provides the following features:

Open-source tool with huge community support.

Easy installation - platform-agnostic (Generic Java package (.war), Docker, FreeBSD, Gentoo, Arch Linux, macOS, Red Hat/Fedora/CentOS, Ubuntu/Debian, OpenBSD, openSUSE, OpenIndiana Hipster, and Windows).

Easy way to configure continuous integration, continuous testing, continuous delivery, and continuous deployment.

It supports arguably all languages as it provides plugin-based architecture. In case plugins are not available, CLI or command execution helps to setup continuous practices – extensible.

Plugins are written in Java and as Jenkins has this extensibility, it is easier to integrate new tools based on market trends.

Organization writes their plugins to integrate internal tools with open source Jenkins.

Default plugin installation at the time of the initial setup.

Simple and easy configuration of tools, proxy, environment variables, etc.

Distributed architecture - Controller/Master Agent architecture to scale and distribute a load of processing along with ease of use, fast execution, and parallel execution.

A huge leap from being popular as a continuous integration server to become an Automation server after the release of Jenkins 2.0.

Rich community and support.

Huge knowledge base and documentation available for beginners.

Jenkins installation and configuration process are different after Jenkins 2.0. Now, there are suggested plugins that you can install in one go. It takes a huge burden away from the Jenkins Engineer's shoulder. You do not need to remember and document which plugins are required and which dependencies are required. Another important option available is to skip this new configuration and install Jenkins in the usual manner as it was done earlier.

The major change is an introduction and big push for Pipeline as a Code rather than creating pipelines or orchestration using traditional pipelines using plugins such as Build pipeline.

Important thing was to focus on Pipeline as a Code and commit the pipeline into a repository so versions can be maintained as it is maintained for code. Visualization is an important part of an end-to-end automation pipeline and Jenkins 2.x focuses on that part significantly.

A multi-branch pipeline provides a way to have different pipeline scripts for different branches. Jenkins automatically detects branches with Jenkinsfile and starts its execution. It is one of the fastest ways to recover concerning Jenkins failure as well. Reason? Because pipeline is available in the repository. Jenkins will fetch branches available in the repo that has

Jenkinsfile and your pipeline is ready immediately considering the environment is available for execution.

Considering UI enhancements and pipeline-related changes, it looks like effective use of design thinking in the new Jenkins 2.0. In the next section, we will discuss the prerequisites to install Jenkins.

Prerequisites

The following are the prerequisites to install Jenkins on a specific system as per our experience:

Jenkins official documentation recommends 256 MB of RAM, although more than 512MB is recommended however we suggest 4–8 GB RAM to manage CI/CD pipeline across multiple projects. One of the common reasons is the way system configuration is available in the organization as well as with a personal laptop or system.

Jenkins official documentation recommends 10 GB of drive space however we suggest 50–80 GB of free space on a disk drive.

Jenkins official documentation recommends Java 8 or 11 (either a JRE or Java Development Kit (JDK) is fine) however we suggest JDK installation only.

How to run Jenkins?

Following are the commands to run Jenkins:

Start Jenkins using command line by using Generic Java package (.war):

```
java -jar jenkins.war
```

Browse to <http://localhost:8080>

Change the port to run Jenkins with the following command:

```
java -jar jenkins.war --httpPort=9999
```

Following is the output of the above command to start Jenkins:

```
Picked up _JAVA_OPTIONS: -Xmx512M
Running from: F:\1.DevOps\2020\jenkins.war
webroot: EnvVars.masterEnvVars.get("JENKINS_HOME")
.
.
```

.

.

2020-04-25 15:42:30.741+0000

[id=33] INFO jenkins.install.SetupWizard#init:

Jenkins initial setup is required. An admin user has been created and a password generated.

Please use the following password to proceed to installation:

a8ce35dccd3a4ec6911647816c602733

This may also be found at:

F:\1.DevOps\2020\Jenkins_Home\secrets\initialAdminPassword

2020-04-25 15:42:51.700+0000

[id=47] INFO h.m.DownloadService\$Downloadable#load:

Obtained the updated data file for
hudson.tasks.Maven.MavenInstaller

2020-04-25 15:42:51.700+0000

[id=47] INFO hudson.util.Retrier#start: Performed the action check updates server successfully at the attempt #1

2020-04-25 15:42:51.712+0000

[id=47] INFO hudson.model.AsyncPeriodicWork#lambda\$doRun\$0: Finished Download metadata. 23,838 ms

2020-04-25 15:43:13.069+0000

[id=21] INFO hudson.WebAppMain\$3#run: Jenkins is fully up and running

Generic War file usage is the easiest way to install and configure Jenkins across Operating systems. You have complete control over its operation and awareness with the uniform approach of Jenkins installation hence it becomes easy and fast.

It is important to note that the installation of Jenkins as service helps to manage Jenkins better in a way that at the startup of the system, Jenkins starts automatically.

Browse to <http://localhost:9999>. It will redirect to Unlock Jenkins page where you must provide Administrator Password available in the console or at Jenkins_Home\secrets\initialAdminPassword:

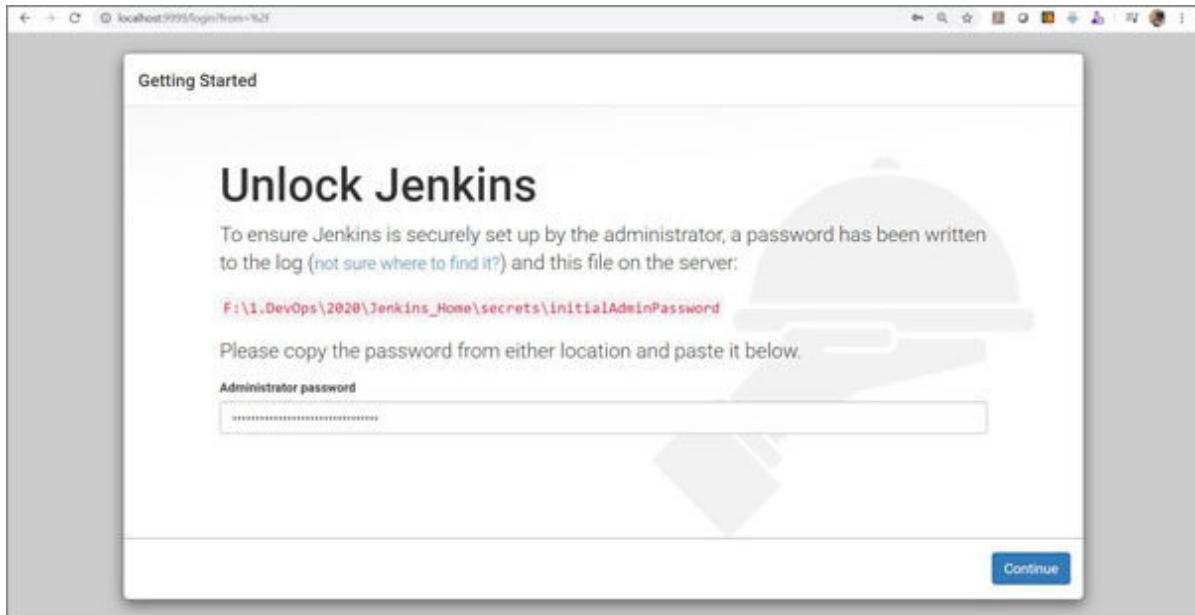


Figure 1.4: *Unlock Jenkins*

Click on **Install Suggested**

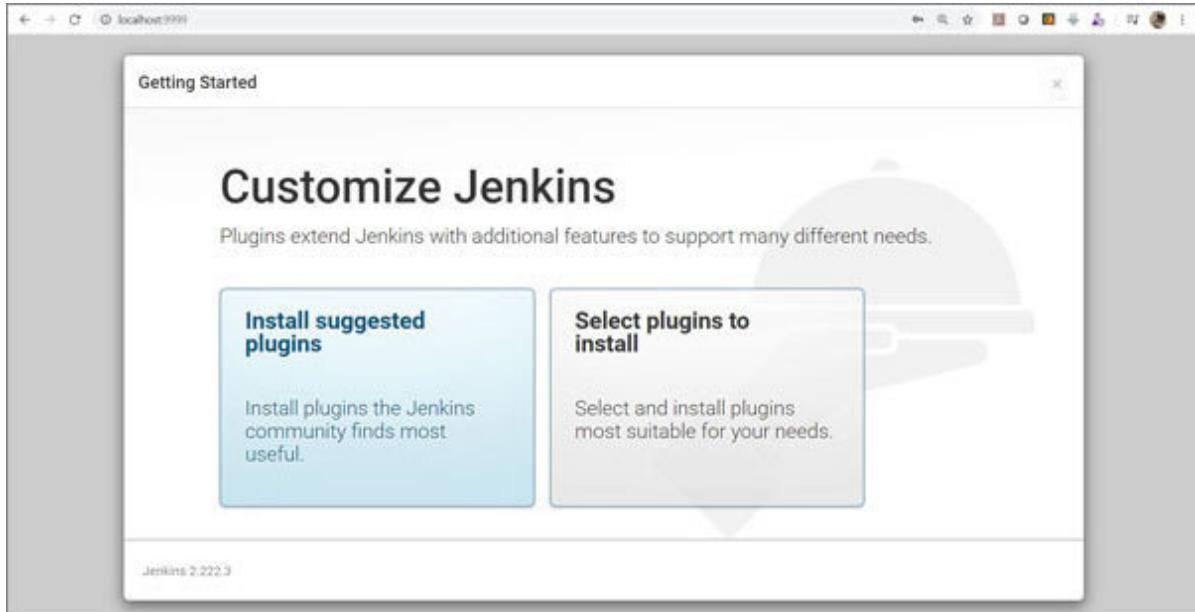


Figure 1.5: *Customize Jenkins*

If Jenkins server is behind a proxy then you need to provide proxy details and go ahead with the installation. In some cases, skip plugin installation, go configure proxy details and verify the connection and then install plugins.

Have patience! Wait until all plugins are installed:

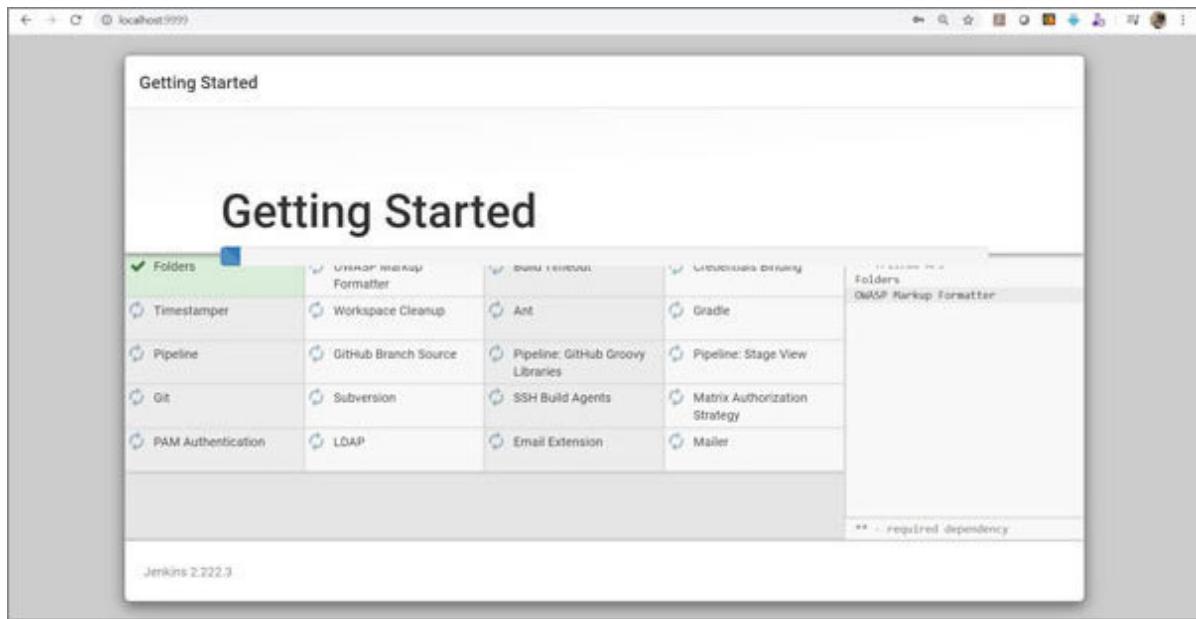


Figure 1.6: Plugins installation

If plugins installation fails then click on

Getting Started

Installation Failures

Some plugins failed to install properly, you may retry installing them or continue

✓ Folders	✓ OWASP Markup Formatter	✓ Build Timeout	✓ Credentials Binding
✓ Timestamper	✓ Workspace Cleanup	✓ Ant	✓ Gradle
✓ Pipeline	✓ GitHub Branch Source	✓ Pipeline: GitHub Groovy Libraries	✓ Pipeline: Stage View
✓ Git	✓ Subversion	✗ SSH Build Agents	✗ Matrix Authorization Strategy
✗ PAM Authentication	✗ LDAP	✗ Email Extension	✓ Mailer

Jenkins 2.222.3

Continue

Retry

Figure 1.7: Plugins installation failures

Once all plugins are installed, create your first admin user and click on **Save and**

Getting Started

Create First Admin User

Username:

Password:

Confirm password:

Full name:

E-mail address:

Jenkins 2.222.3 Continue as admin Save and Continue

This screenshot shows the 'Create First Admin User' page from the Jenkins 'Getting Started' section. It contains five input fields: 'Username', 'Password', 'Confirm password', 'Full name', and 'E-mail address'. Below the form are two buttons: 'Continue as admin' and 'Save and Continue'. The status bar at the bottom indicates 'Jenkins 2.222.3'.

Figure 1.8: Admin user

Provide Jenkins URL or keep the default based on need:

Getting Started

Instance Configuration

Jenkins URL:

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the \$BUILD_URL environment variable provided to build steps.
The proposed default value shown is not saved yet and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

Jenkins 2.222.3 Not now Save and Finish

This screenshot shows the 'Instance Configuration' page from the Jenkins 'Getting Started' section. It features a single input field for 'Jenkins URL' with the value 'http://localhost:9999/'. Below the field is a detailed explanatory text block. At the bottom are 'Not now' and 'Save and Finish' buttons.

Figure 1.9: Jenkins instance configuration

Use the IP address or Domain name in place of localhost if required to access Jenkins in the network. If it is not changed then none can access it outside your system.

Once Jenkins setup is complete, click on **Start using**

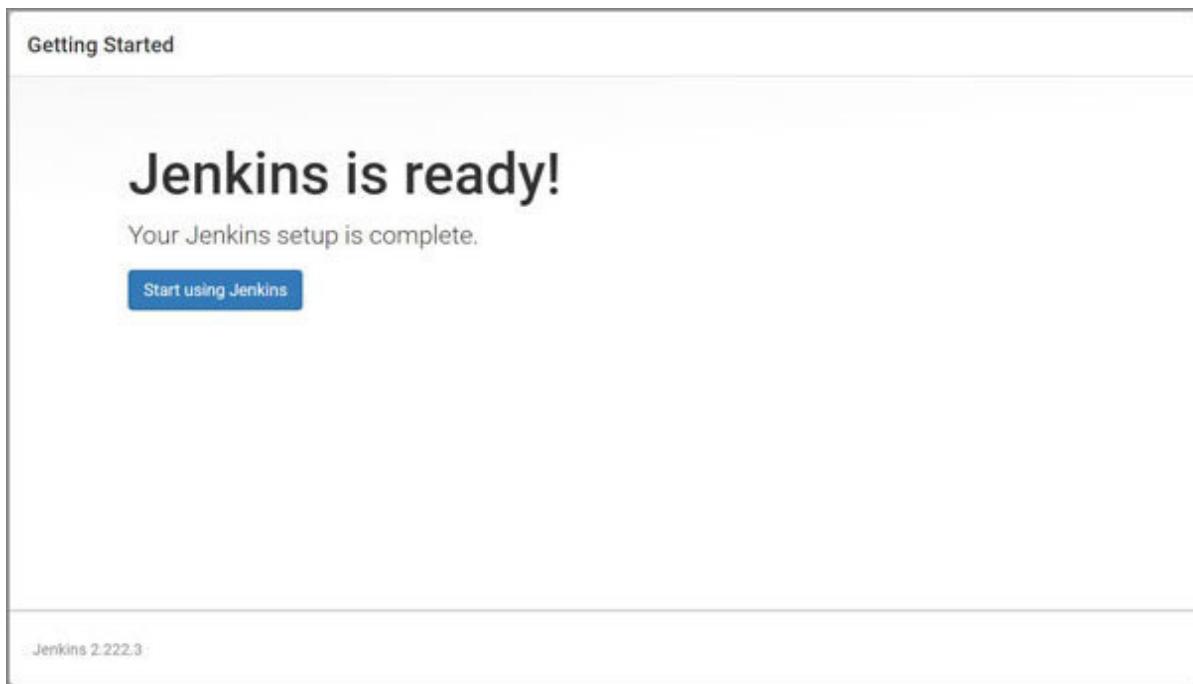


Figure 1.10: Jenkins set-up is complete

Verify Jenkins Dashboard. Click on **Manage**

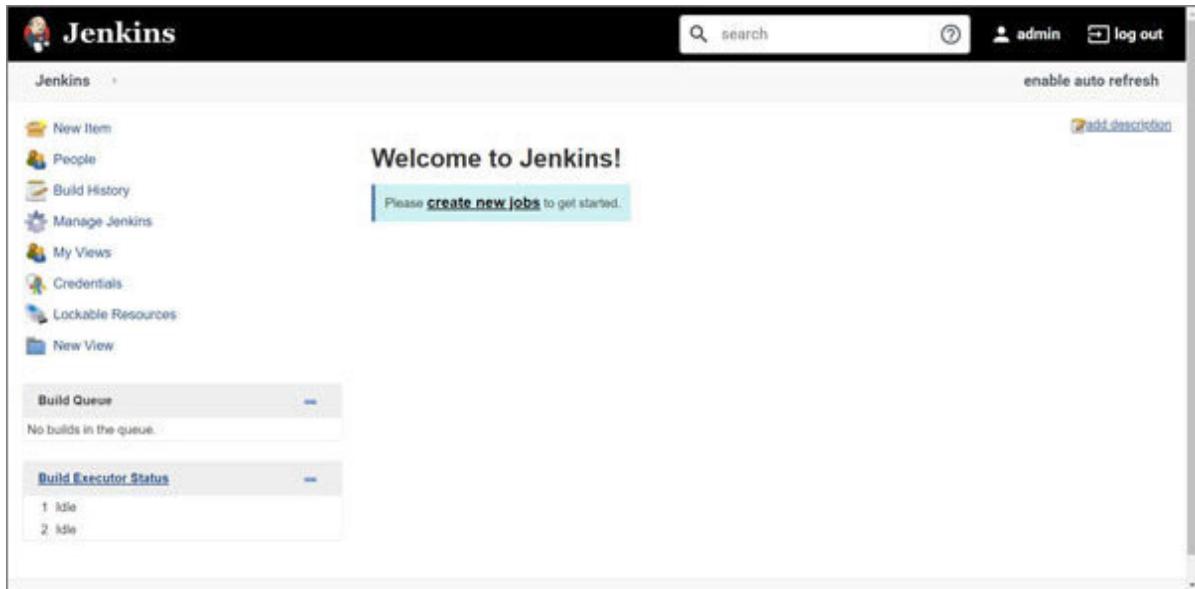


Figure 1.11: Jenkins dashboard

Click on **Configure System** to monitor Jenkins configuration.

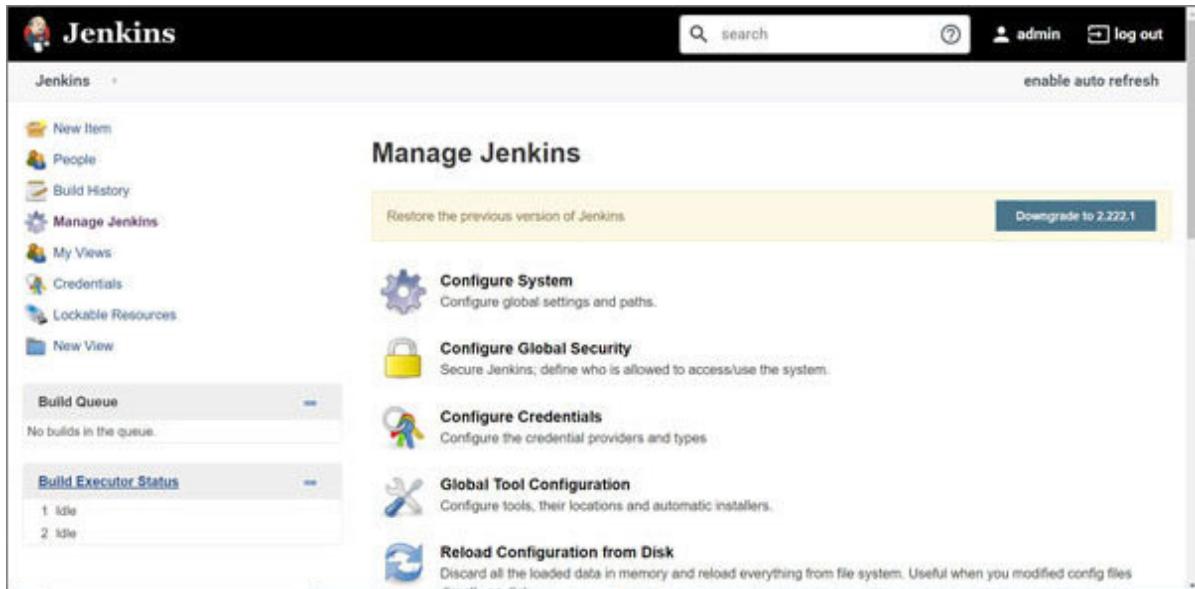


Figure 1.12: Manage Jenkins

Verify the JENKINS_HOME directory and Jenkins' URL. If you want to change Jenkins URL then modify it in this section:

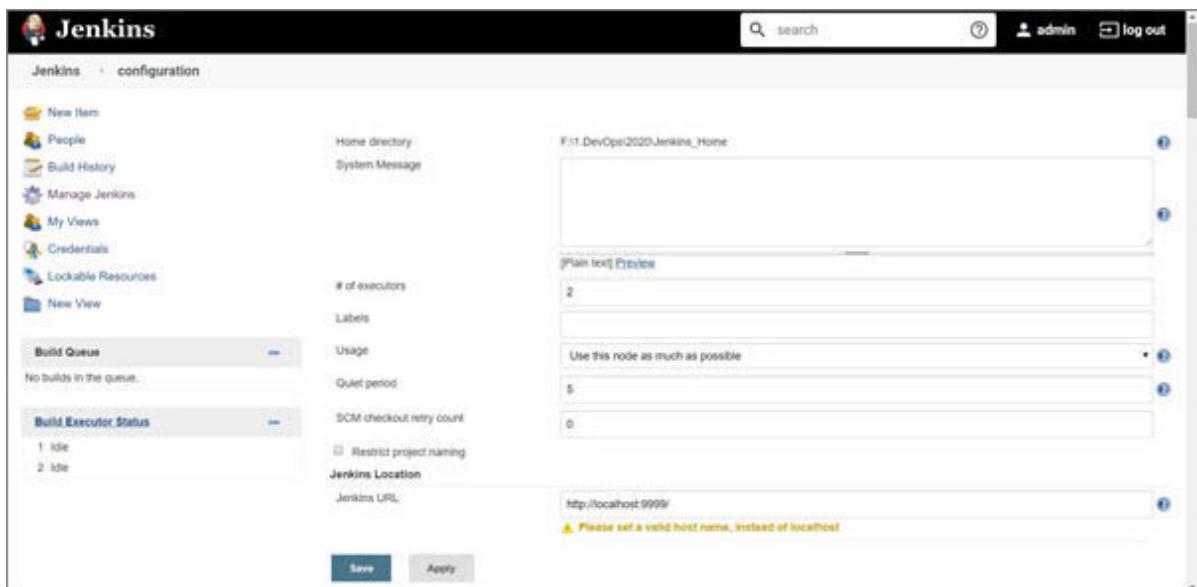


Figure 1.13: Configure Jenkins

Always configure Backup and restore practices and processes for JENKINS_HOME or keep manual backup based on the feasibility to avoid any disaster in future.

Go to **Manage Jenkins | Global Tool**

Here, you can configure all tools that Jenkins will use to automate application lifecycle management activities.

You can provide a path, if tools are already installed or configured or you can install automatically from this section only:

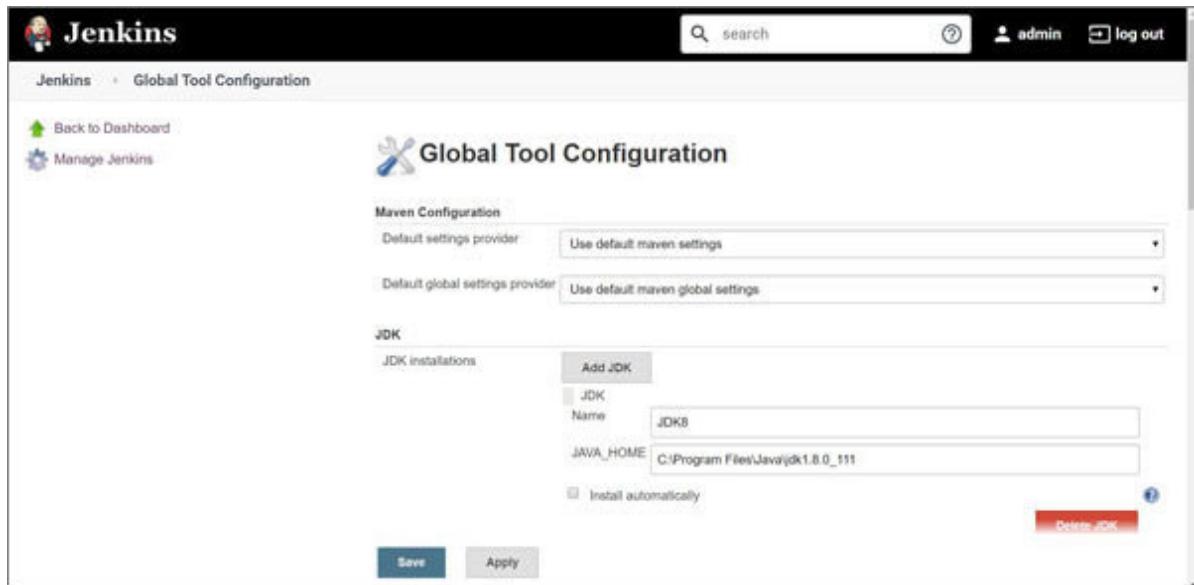


Figure 1.14: Global Tool Configuration

Keep the generic name in such as jdk8 or jdk11 and configure all tools here with the path. It is easier to use in pipeline without any issues.

Go to **Manage Jenkins** | This is the section where we can manage Controller/Master-Agent nodes.

As of now, only the Controller/Master node is available that is the system on which Jenkins is installed.

Click on the Controller/Master node name:

The screenshot shows the Jenkins interface for managing nodes. At the top, there's a navigation bar with links for 'Back to Dashboard', 'Manage Jenkins', 'New Node', 'Configure Clouds', and 'Node Monitoring'. On the right side of the header, there are 'admin' and 'log out' buttons, along with a 'search' field and a 'enable auto refresh' checkbox. Below the header, the main content area is titled 'Nodes'. It displays a table with one row for the 'master' node. The columns in the table are: S, Name, Architecture, Clock Difference, Free Disk Space, Free Swap Space, Free Temp Space, and Response Time. The 'master' node details are: Name is 'master', Architecture is 'Windows 10 (amd64)', Clock Difference is 'In sync', Free Disk Space is '132.85 GB', Free Swap Space is '12.63 GB', Free Temp Space is '54.64 GB', and Response Time is '0ms'. Below the table, there's a 'Data obtained' timestamp and a 'Refresh status' button. On the left, there are two collapsed sections: 'Build Queue' (which says 'No builds in the queue.') and 'Build Executor Status' (which shows '1 Idle' and '2 Idle'). At the bottom of the page, there's a footer with the text 'Page generated: Apr 26, 2020 9:43:31 PM IST REST API Jenkins ver. 2.222.3'.

S	Name	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time
	master	Windows 10 (amd64)	In sync	132.85 GB	12.63 GB	54.64 GB	0ms
		Data obtained	30 min	30 min	30 min	30 min	30 min

Figure 1.15: Nodes in Jenkins

Click on **Configure** and observe the usage of # of executors based on the help section:

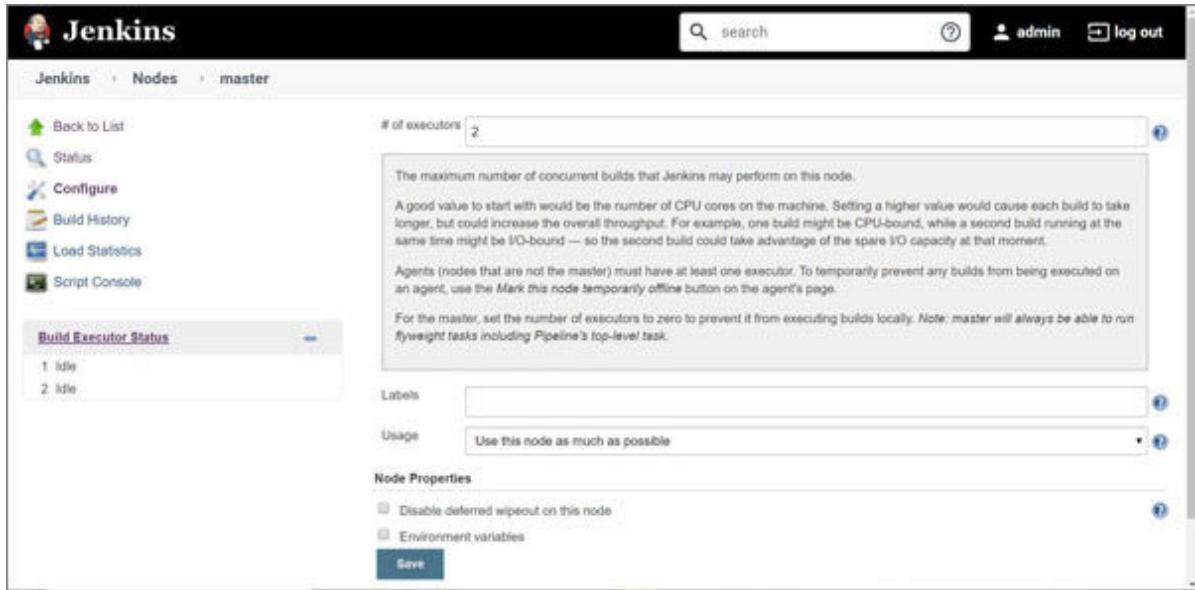


Figure 1.16: Node Configuration

What is the default JENKINS_HOME directory in Windows?

Answer: C:\Users\\.jenkins

How to change the default JENKINS_HOME directory in Windows?

Go to **System Properties** and click on

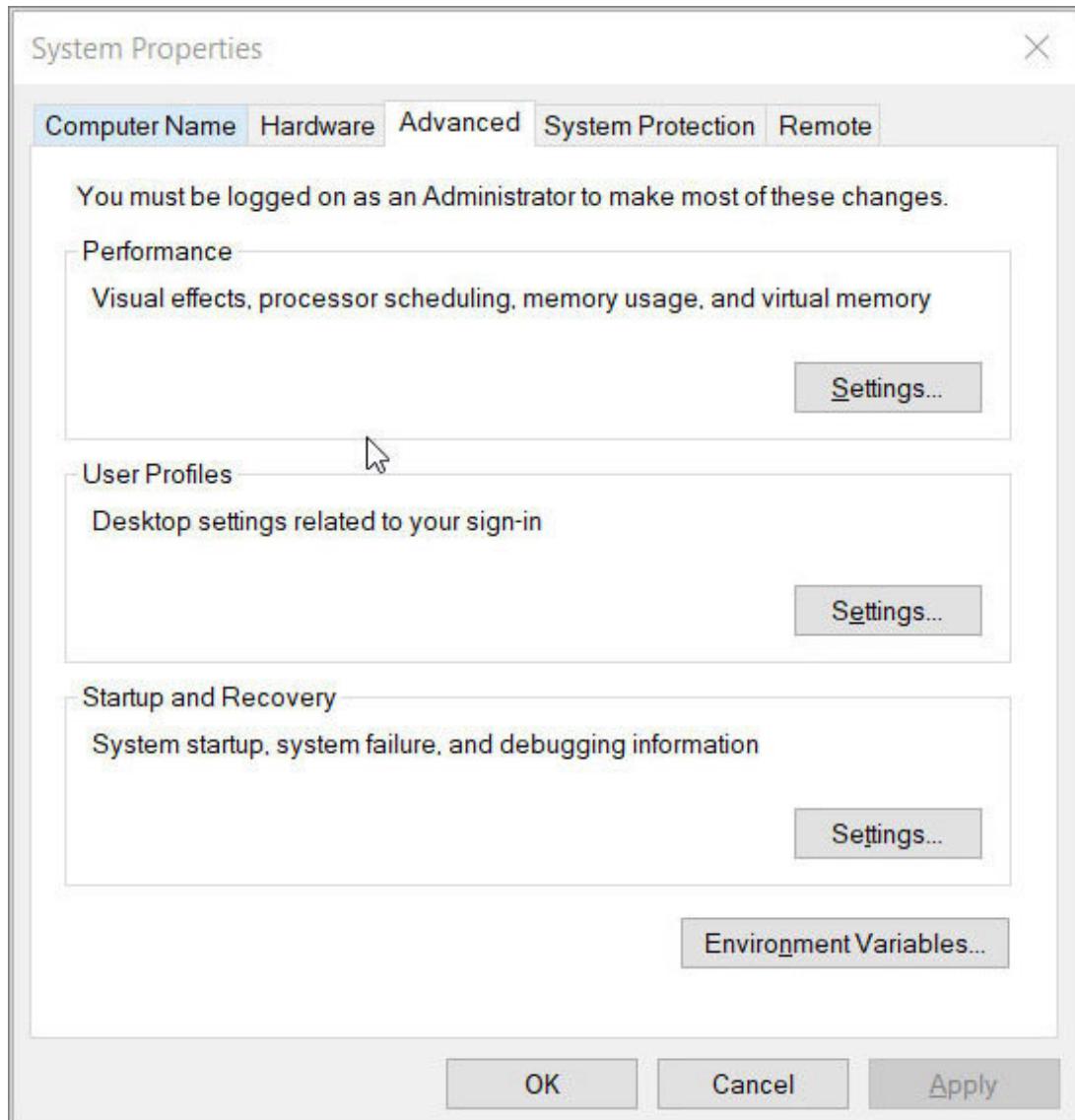


Figure 1.17: System Properties

Click on **Environment**

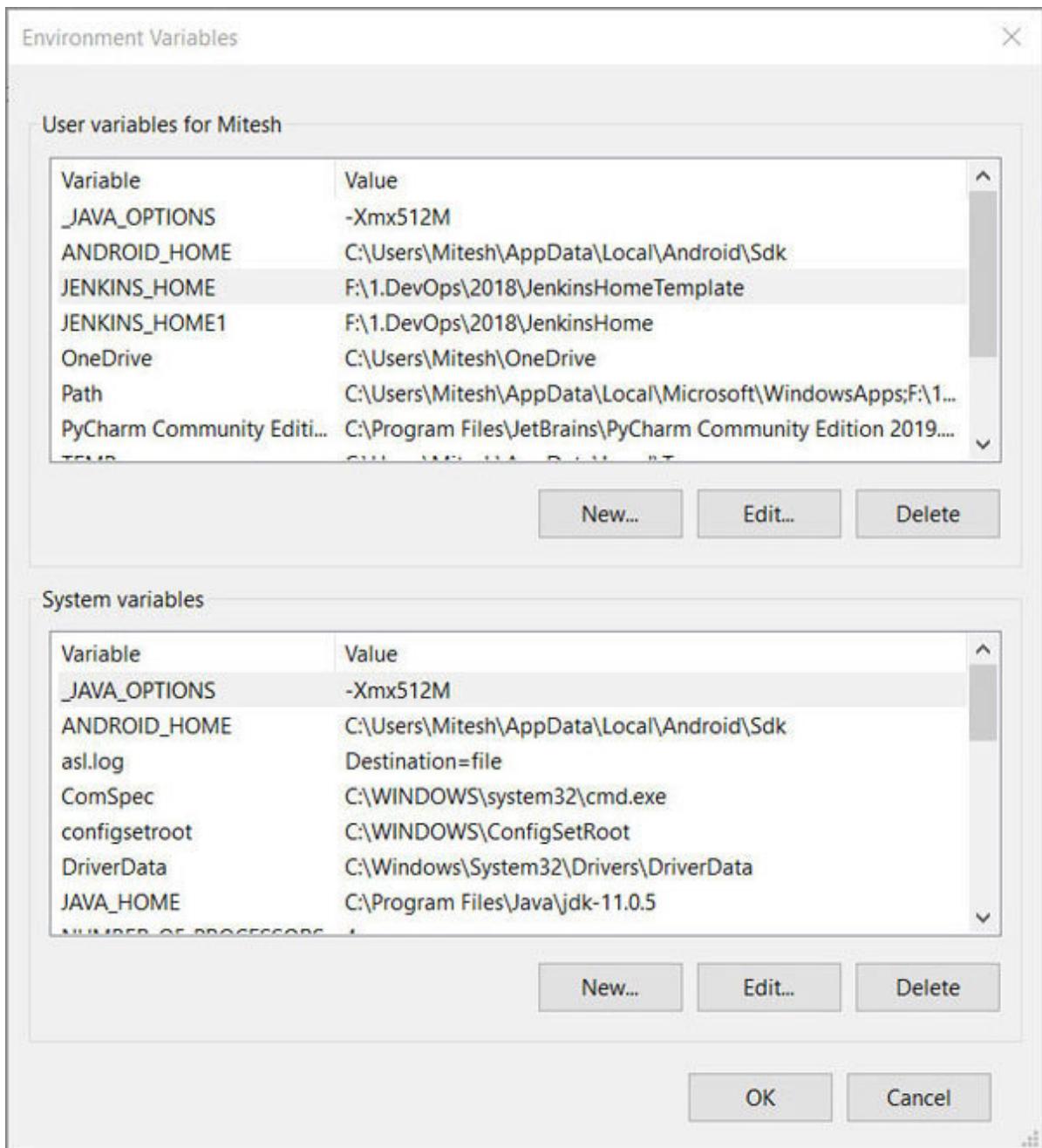


Figure 1.18: Environment Variables

Once you start Jenkins, you can find the JENKINS_HOME in the console:

```
F:\1.DevOps\2020>java -jar jenkins.war
Picked up _JAVA_OPTIONS: -Xmx512M
Running from: F:\1.DevOps\2020\jenkins.war
webroot: EnvVars.masterEnvVars.get("JENKINS_HOME")
```

Thus we have now covered the basics of Jenkins in this section. Next section will be about pipelines.

Pipelines

Jenkins 2.0 and later versions have Built-in support for delivery pipelines. Jenkins 2.x has improved usability, and it is fully backwards compatible. In today's competitive market, all organizations compete to release better quality products with faster time to market.

Even before Jenkins 2.0, Plugins helped to utilize CD as well but Jenkins 2.0 brings backs the focus on end-to-end automation. Pipelines help to organize and manage the orchestration of Application Lifecycle Management easily and robustly.

Pipelines use **domain-specific language** to visualize and orchestrate Application Life Cycle Management activities. It essentially means **Pipeline as a Code**. The highlight of this feature is, users can commit PaaC (essentially a script) in the repository and keep many versions of it.

Sounds familiar? Yes, we all do it while managing code in code repositories, Is not it?

In short, we can use **DSL** to create an automation pipeline that will orchestrate SCA, build, test, and deploy operations. It can also include load testing, security testing, and many other operations based on the existing culture of an organization and requirements of the project.

The reason why pipelines are getting more popular is the usage of Jenkins over the years. It is all about patterns. Usage of Jenkins and build pipeline make user realize the complexities of Jenkins management over time. It is easier to use the Build pipeline plugin in Jenkins to create pipelines using upstream and downstream jobs. Most beginners follow this approach but over time it becomes difficult to manage.

Let us understand the traditional pipeline using the build pipeline plugin.

Build pipeline

Following are the reasons PaaS is preferred over the traditional approach of using upstream and downstream jobs:

Each task has a corresponding unique build job or Project hence there are too many jobs to create and manage.

A business unit or organization manages Multiple projects.

Build pipeline configuration is in Jenkins itself and in case of failure it is a rework.

It is difficult to track changes in the pipeline created using Build pipeline.

Go to **Manage Jenkins | Manage Plugins | Available** tab and select **Build Pipeline** plugin for installation.

Let us create three simple freestyle jobs:

SonarQubeAnalysis

Build

Deploy2Tomcat

Let us try to configure orchestration:

Go to SonarQubeAnalysis job in Jenkins and click on Go to the **Post-build Actions** section and click on **Add post-build** action and select **Build other projects**.

We want to execute a Build job once this job is successfully completed.

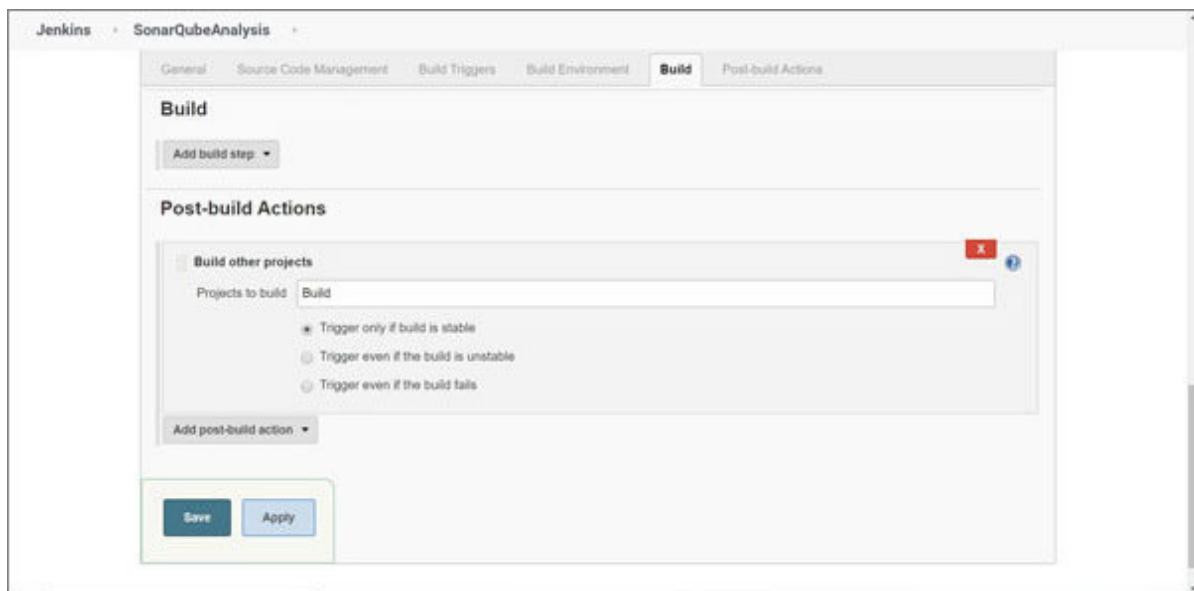


Figure 1.19: Build other projects

Verify the **Downstream Project** in Job's dashboard.

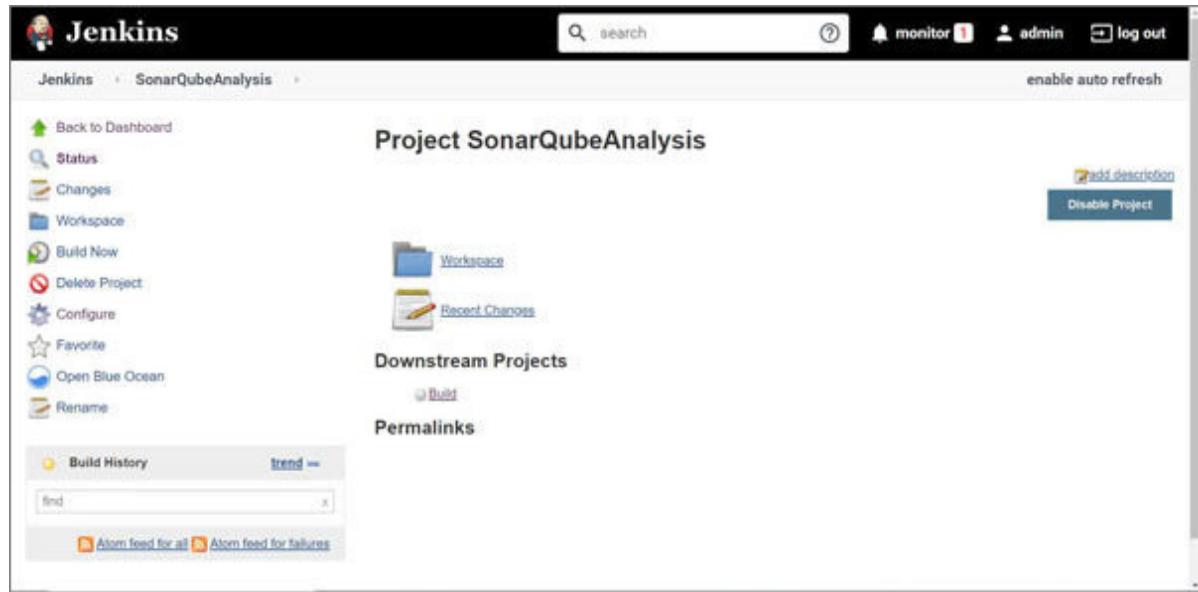


Figure 1.20: Downstream Projects

Verify the **Upstream Project** in Job's dashboard:

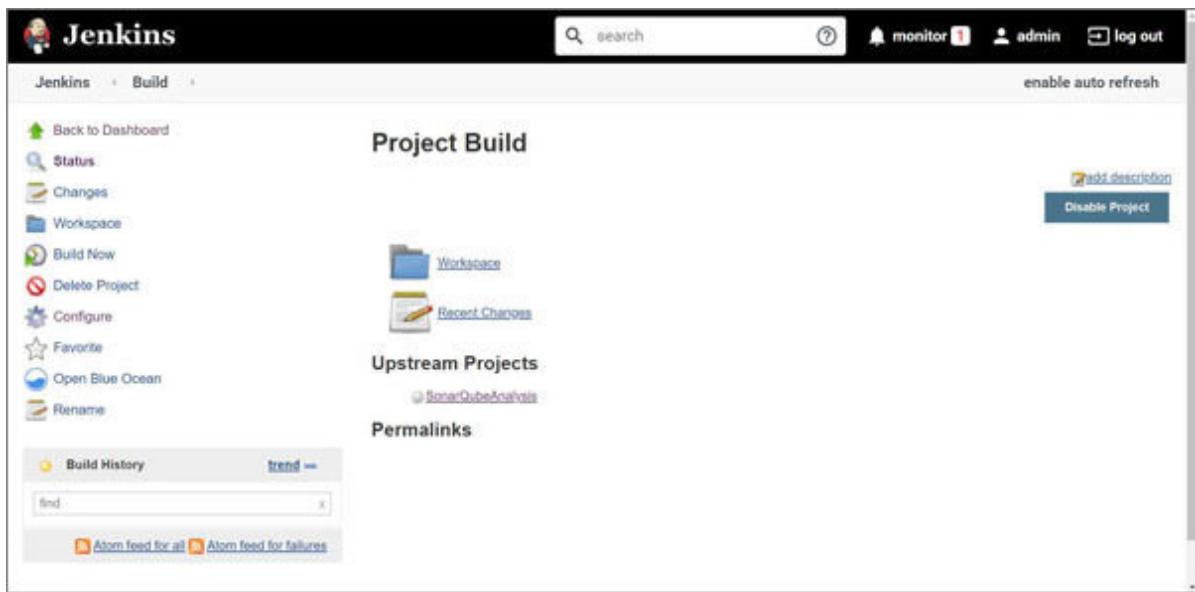


Figure 1.21: Upstream Projects

Go to **Build** job in Jenkins and click on **Configure** and go to **Post-build Actions** section. Click on **Add post-build action** and select **Build** other projects.

We want to execute the Deploy2Tomcat job once the Build job is completed:

The screenshot shows the Jenkins Project Build page. On the left, there's a sidebar with links like Back to Dashboard, Status, Changes, Workspace, Build Now, Delete Project, Configure, Favorite, Open Blue Ocean, and Rename. Below that is a search bar and a 'Build History' section with a 'trend' dropdown and a 'find' input field. At the bottom of the sidebar are two Atom feed links: 'Atom feed for all' and 'Atom feed for failures'. The main content area is titled 'Project Build' and contains sections for 'Workspace' (with a link to 'Recent Changes'), 'Upstream Projects' (listing 'SonarQubeAnalysis'), 'Downstream Projects' (listing 'Deploy2Tomcat'), and 'Permalinks'. In the top right corner, there are buttons for 'add description' and 'Disable Project'.

Figure 1.22: Upstream and Downstream Projects

Create **Build Pipeline View** from Jenkins Dashboard:

The screenshot shows the Jenkins 'New View' dialog. The left sidebar lists options like New Item, People, Build History, Manage Jenkins, My Views, Open Blue Ocean, Lockable Resources, Credentials, and New View. The main area has a 'View name' input field containing 'Build_Pipeline'. Below it, there are three radio button options: 'Build Pipeline View' (selected), 'List View', and 'My View'. The 'Build Pipeline View' option is described as showing jobs in a pipeline view. The 'List View' option is described as showing items in a simple list format. The 'My View' option is described as automatically displaying all accessible jobs. At the bottom right is an 'OK' button.

Figure 1.23: Build Pipeline View

Select Initial Job as SonarQubeAnalysis as we want to start our pipeline from this job:

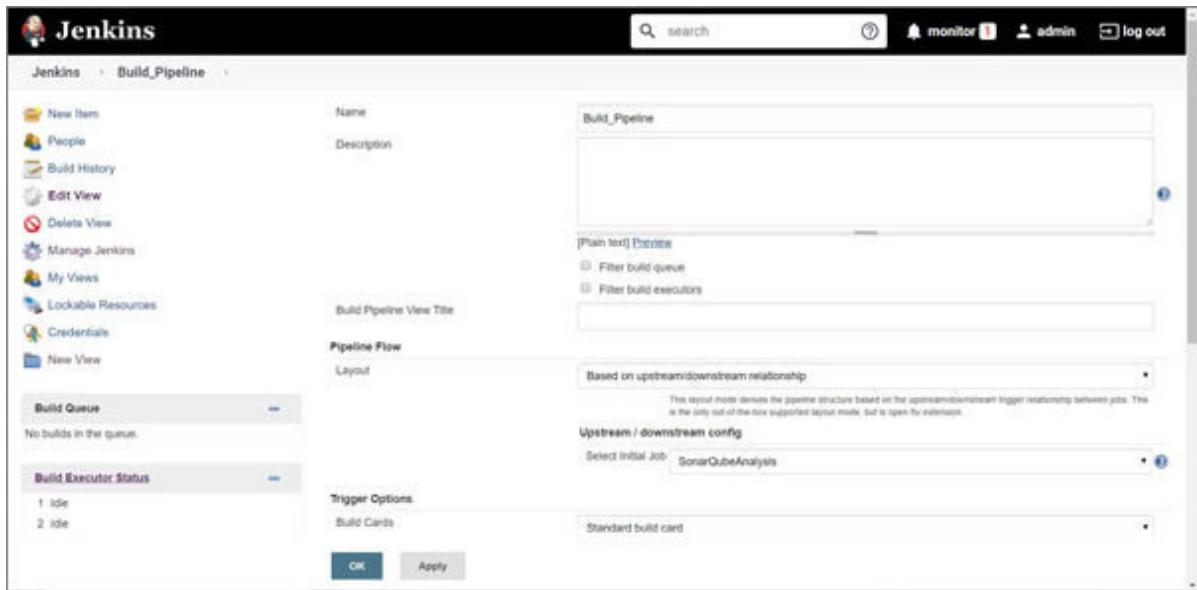


Figure 1.24: Build Pipeline Configuration

Execute the Pipeline:

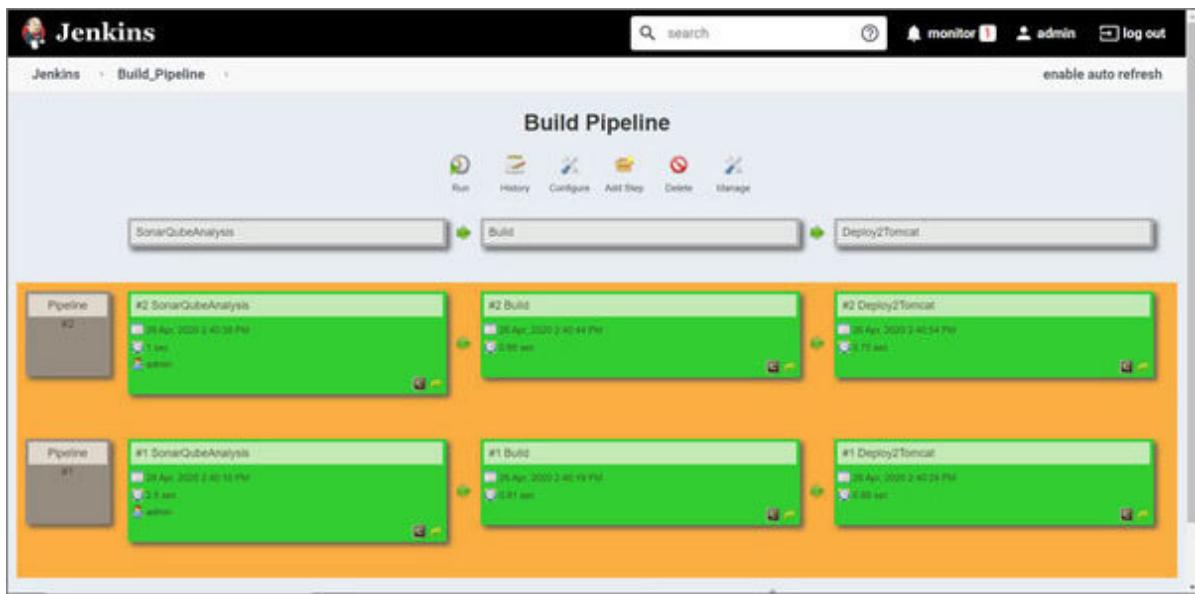


Figure 1.25: Build Pipeline View

Now let us understand how Pipeline as code helps you to fix the above issues as it supports:

DSL helps to create pipelines through the DSL or Jenkinsfile.

Pipeline as a Code can utilize all programming concepts.

Common approach and standards for all teams.

Version control of the pipelines as it is also a script and managed in version control systems such as SVN or Git.

The pipeline helps to define and implement CI, continuous testing, and CD pipeline using DSL in Jenkins. Jenkinsfile contains the script to automate CI, continuous testing, and CD or it is available in Jenkins pipeline job. The visible benefit of creating a Jenkinsfile to contain Pipeline as a Code is its version control in the repository. Hence, Infrastructure as a Code has a sister in the name of Pipeline as a Code.

As there are two ways to create a pipeline:

Jenkins dashboard

Jenkins file

It is a best practice to use Jenkinsfile. Let us understand some basic pipeline concepts:

Pipeline It models CICD pipeline or it contains stages or phases of Application Lifecycle Management.

Agent or It is a system (virtual or physical), utilized in the Controller/Master Agent Architecture of Jenkins. Jenkins execution takes place on the Controller/Master or Agent Node.

It is a collection of tasks. For example, compilation of source files, unit test execution, and publishing Junit test reports. We can represent it as a block. To understand it more easily, consider it as a Source Code Analysis or CI or continuous testing or CD.

It is a task. multiple tasks make a stage. execution of the script, execution of maven command, execution of SonarQube analysis command, and execution of Gradle command, and so on.

There are two types of pipelines in Jenkins as of today. It means that Jenkinsfile can contain two different types of style/syntax and yet it can achieve the same thing. Yes, DevOps Practices implementation.

Scripted pipeline

Scripted pipelines follow the Imperative programming model. Scripted pipelines are written in Groovy script in Jenkins. All Groovy blocks/constructs help to manage flow as well as error reporting. Scripted pipeline does not have a restart stage feature, Post-build actions, error checking and reporting (try-catch-finally blocks), and notifications constructs. It requires Groovy programming skills hence it has a steep learning curve. It is not easy to understand, manage, and maintain.

Let us see how a scripted pipeline looks like:

Available since long.

How to do approach?

Complex in comparison to Declarative pipeline

The groovy syntax is the last limitation hence complex but we can easily utilize it in complex pipeline structures.

Imperative programming paradigm of computer programming in which the program or script has a collection of steps or tasks that defines how this script should achieve the outcome.

A node block is a root of Scripted pipeline syntax.

Following is the structure of the scripted pipeline:

```
node {  
    /* Stages and Steps */
```

```
}
```

```
node {  
    stage('SCA') {
```

```
        // steps
```

```
}
```

```
    stage('CI') {
```

```
// steps  
}  
  
{  
  
// steps  
}  
  
}
```

Let us see the Scripted pipeline template available in the pipeline:

Create a Pipeline job in Jenkins:

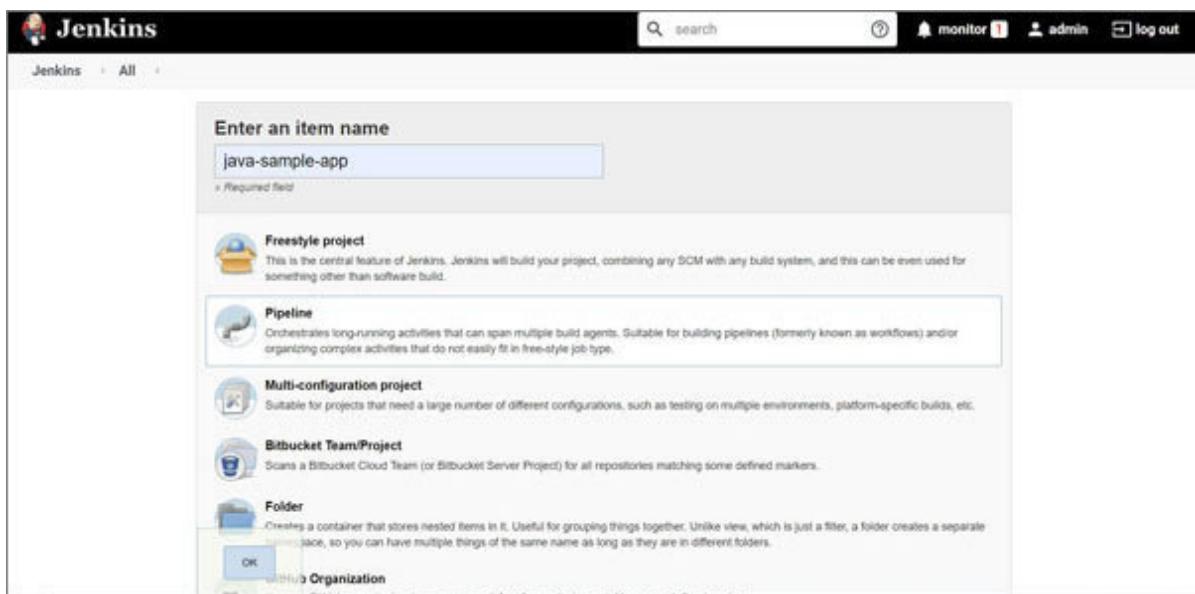


Figure 1.26: Pipeline Job

Go to the **Pipeline** section in the Job.

Select the **Scripted Pipeline** template.

Click on

The screenshot shows the Jenkins Pipeline configuration page for a project named "java-sample-app". The "Pipeline" tab is selected. The "Scripted Pipeline" dropdown is set to "Script". The pipeline script is defined as follows:

```
1+ node {
2+     def maven
3+
4+     stage('Preparation') { // for display purposes
5+         // Get some code from a GitHub repository
6+         git 'https://github.com/tglick/single-maven-project-with-tests.git'
7+         // Get the Maven tool.
8+         // ** NOTE: This 'M0' Maven tool must be configured
9+         //          in the global configuration.
10+        maven = tool 'M0'
11+
12+        stage('Build') {
13+            // Run the Maven build
14+            withMaven(['MAVEN_HOME/bin/mvn']) {
15+                if (isUnix()) {
16+                    sh "MAVEN_HOME/bin/mvn -Dmaven.test.failure.ignore clean package"
17+                } else {
18+                    bat("MAVEN_HOME/bin/mvn -Dmaven.test.failure.ignore clean package")
19+                }
20+            }
21+        }
22+    }
23+}
```

Below the script, there are two buttons: "Save" and "Apply". At the bottom right of the page, there is a footer note: "Page generated Apr 26, 2020 6:43:56 AM IST REST API Jenkins ver. 2.222.3".

Figure 1.27: Scripted pipeline

Click on the **Pipeline**

Use **Snippet Generator** to create script commands to use in the pipeline:

The screenshot shows the Jenkins Pipeline Syntax Snippet Generator page. The left sidebar contains links for Back, Snippet Generator, Declarative Directive Generator, Declarative Online Documentation, Steps Reference, Global Variables Reference, Online Documentation, Examples Reference, and IntelliJ IDEA Groovy. The main content area has a title "Overview" with a description of the Snippet Generator's purpose. A "Steps" section shows a "Sample Step" of "archiveArtifacts: Archive the artifacts". Below it, a "Files to archive" field contains "**/*.war". There is an "Advanced..." button. At the bottom is a "Generate Pipeline Script" button, which generates the code "archiveArtifacts '**/*.war'".

Figure 1.28: Pipeline Syntax - Snippet Generator

Similarly, the **Declarative Directive Generator** section is used to create a script in declarative syntax:

The screenshot shows the Jenkins Pipeline Syntax Declarative Directive Generator page. The left sidebar contains links for Snippet Generator, Declarative Directive Generator, Declarative Online Documentation, Steps Reference, Global Variables Reference, Online Documentation, Examples Reference, and IntelliJ IDEA Groovy. The main content area has a title "Directives" with a description of the Directive Generator. A "Sample Directive" dropdown is set to "stages: Stages". Below it, a "Stage" section shows a "Name" field with "build" and a "This stage will contain..." field with "Steps". There is an "Add" button and a "Delete" button. At the bottom is a "Generate Declarative Directive" button, which generates the code "stages [stage('build') { steps [] }]".

Figure 1.29: Declarative Directive Generator

Let us understand the Declarative pipeline.

Declarative pipeline

Declarative pipeline follows a declarative programming model. Declarative pipelines are written in DSL in Jenkins that is easy to understand and clear. It doesn't have a Restart Stage feature, Post-build actions, error checking and reporting (try-catch-finally blocks), and notifications constructs. Declarative pipeline has script {} block to execute Scripted pipeline inside a Declarative pipeline:

```
pipeline {  
  
    agent any  
  
    stages {  
  
        stage('Rollback') {  
  
            steps {  
  
                bat 'echo "RollBack..."'  
            }  
        }  
    }  
}
```

```
script {

    Successful_Build = readFile
'C:\\\\Users\\\\Mitesh\\\\2019\\\\JENKINS_HOME\\\\workspace\\\\Rollback.
txt'

    echo Successful_Build

    rtDownload (
        serverId: "artifactory",

        spec:

        """{

            "files": [

                {

                    "pattern": "example-repo-
local/example/${Successful_Build}/example.4.2.5-SNAPSHOT.war",

                    "target": "Rollback/"
```

```
        }
```

```
    ]
```

```
    }""""
```

```
    )
```

```
    }
```

```
}
```

```
}
```

```
post {
```

```
    success {
```

```
        script {
```

```
            Successful_Build = "${BUILD_TAG}" // this is
```

```
Groovy
```

```
echo Successful_Build // printing via Groovy  
works
```

```
        writeFile(file:  
"C:\\\\Users\\\\Mitesh\\\\2019\\\\JENKINS_HOME\\\\workspace\\\\Rollback  
.txt", text: "${Successful_Build}")
```

```
}
```

```
}
```

```
}
```

```
}
```

Let us see how a declarative pipeline looks like:

Recent feature

What to do

Easy to understand and developed for being user friendly.

Fixed structure hence Simple; Good for beginners.

Declarative programming paradigm of computer programming in which the program or script has a collection of steps or tasks that defines what this script should achieve as an outcome.

A pipeline block is a root of Declarative pipeline syntax.

Once the pipeline templates are ready for a specific type of project and specific to the programming language, it is easy to replicate the pipelines and work directly with scripts.

Following is a sample Declarative pipeline:

```
pipeline {
```

```
    /* Stages and Steps */
```

```
}
```

```
pipeline {
```

```
    agent any
```

```
stages {  
  
    stage('SCA') {  
  
        steps {  
  
            //  
  
        }  
  
    }  
  
    stage('CI') {  
  
        steps {  
  
            //  
  
        }  
  
    }  
  
    stage('CD') {
```

```
steps {
```

```
//
```

```
}
```

```
}
```

```
}
```

Let us see the declarative syntax template:

In the Pipeline job that we created earlier, select **Hello World**

Click on

The screenshot shows the Jenkins Pipeline configuration page for a project named "java-sample-app". The "Pipeline" tab is selected. In the "Definition" section, there is a "Script" tab containing the following Groovy code:

```
1+ pipeline {
2+   agent any
3+
4+   stages {
5+     stage('Hello') {
6+       steps {
7+         echo 'Hello world'
8+       }
9+     }
10+   }
11+ }
```

A dropdown menu is open next to the "Hello World" stage name, listing options: "Hello World", "try sample Pipeline", "Hello World", "GitHub + Maven", and "Scripted Pipeline". Below the script, there is a "Pipeline Syntax" link and two buttons: "Save" and "Apply".

Figure 1.30: Sample Declarative Pipeline

Click on **Build now** and verify the stage view.

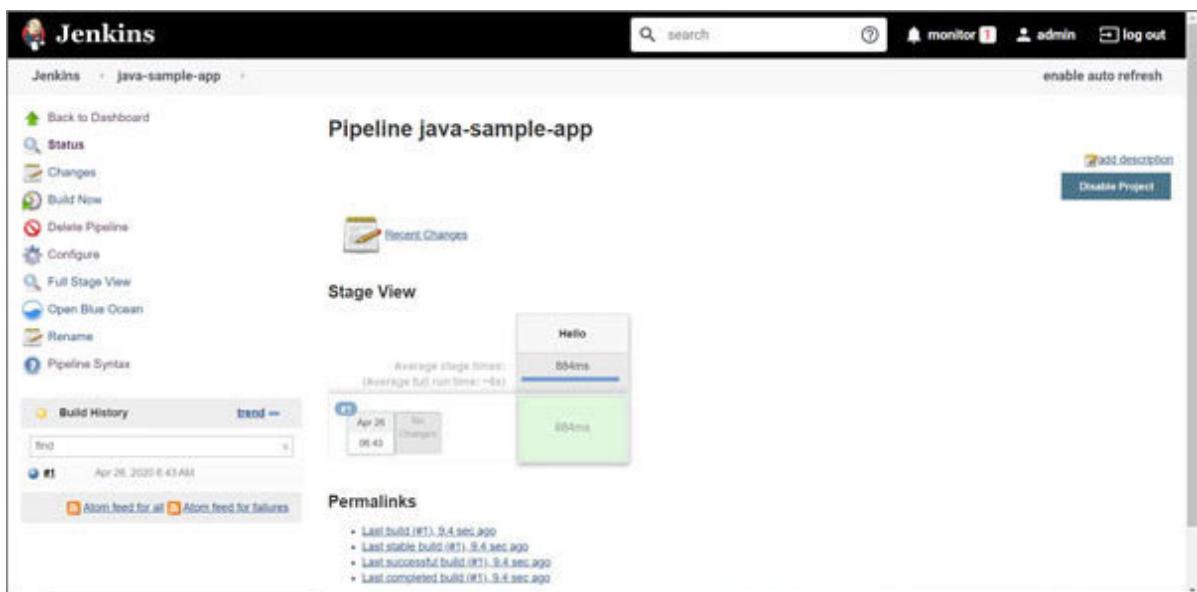


Figure 1.31: Stage View

In the next section, we will discuss the Blue Ocean.

Blue Ocean

Blue Ocean provides an easy way to create a Declarative pipeline using a new user experience available in the Blue Ocean dashboard. It is like creating a script based on selecting components or steps or tasks. Most of the automation tools and services provide similar functionality to attract beginners to adapt to new Jenkins. It is a mature and transparent representation of end-to-end automation pipeline. We can access the log stage-wise and access script directly into Jenkins Dashboard. It provides view including multiple branches if branches have Jenkinsfile. We can create Jenkinsfile using Blue Ocean if it is not already available and if Jenkinsfile is already available with valid syntax then Jenkins detects the Jenkinsfile and starts execution if a valid runtime environment is available. Blue Ocean provides a pipeline editor where we can directly enter script or add stages and steps using UI elements. Stages are represented properly including Parallel stages execution or if the stage is skipped during execution based on condition. Stage-wise logs are available to provide quick insights in case of failures and you do not need to scan the entire log to find an issue.

Let us understand how to start Blue Ocean pipeline configuration:

Go to **Manage Jenkins | Manage Plugins | Available** and select **Blue Ocean** and **Install** without restart:

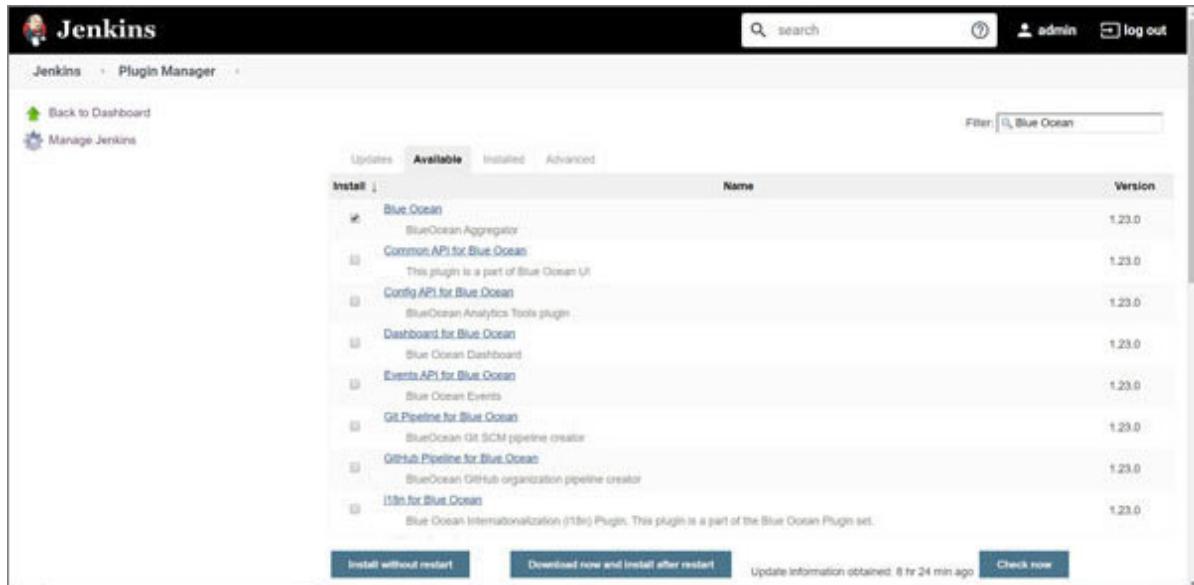


Figure 1.32: Available Plugins

Verify successful installation of Blue Ocean plugin and its dependencies:

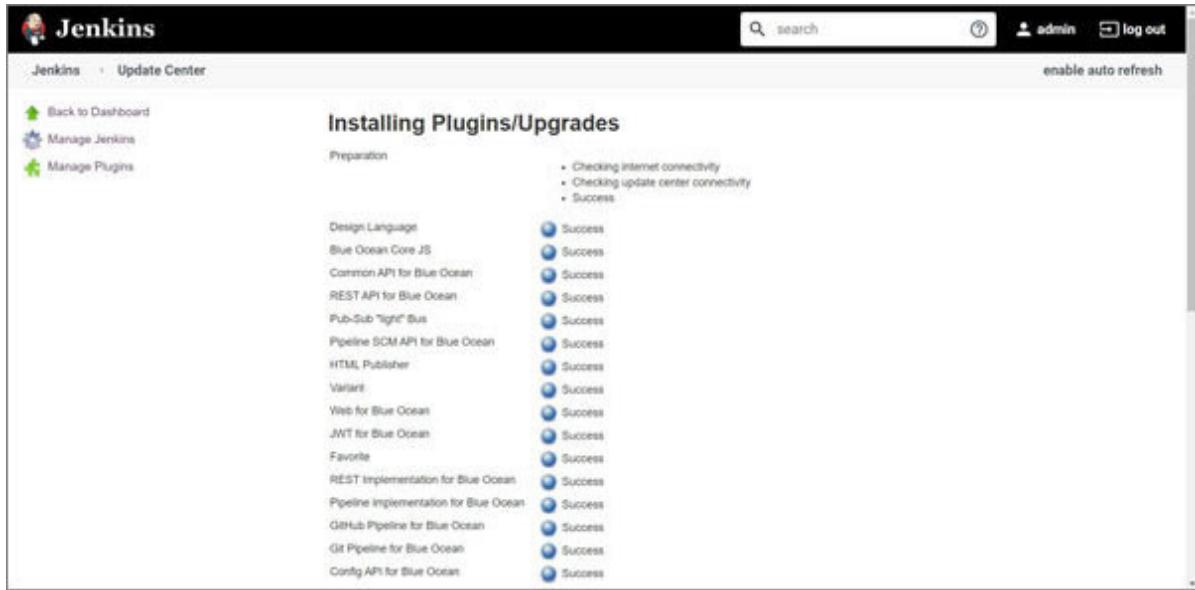


Figure 1.33: Jenkins Update Center

After Blue Ocean plugin installation verifies the left sidebar where the new option has emerged.

Click on **Open Blue**

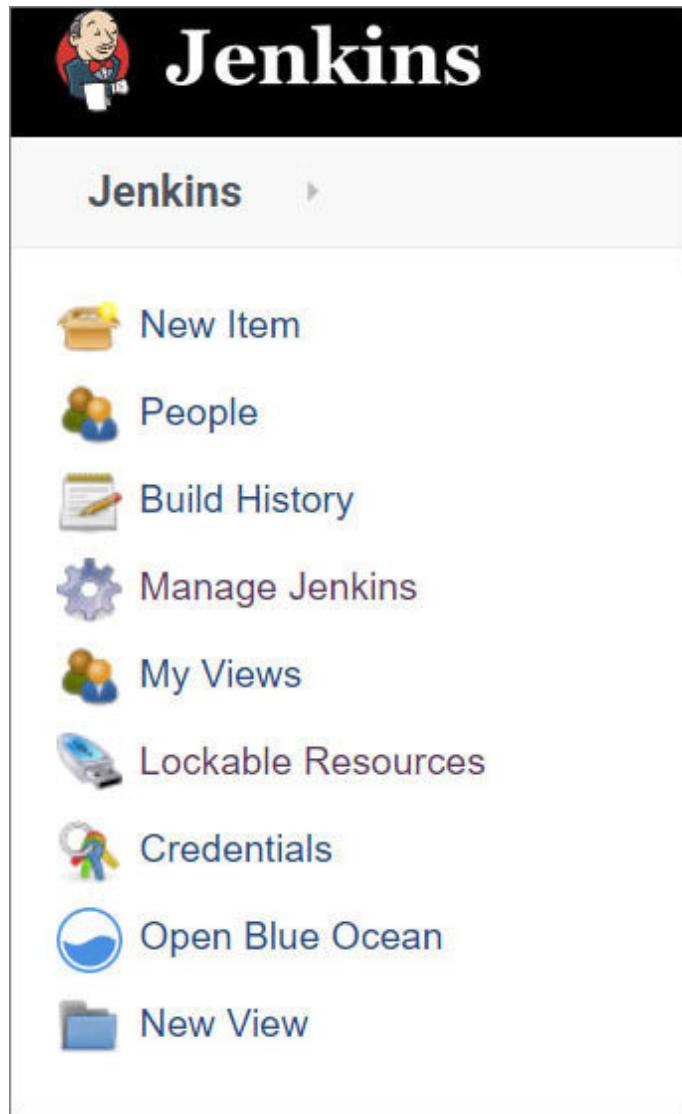


Figure 1.34: Blue Ocean in Jenkins Dashboard

Verify the Blue Ocean Dashboard. It provides a new user interface.

Click on **Create New**

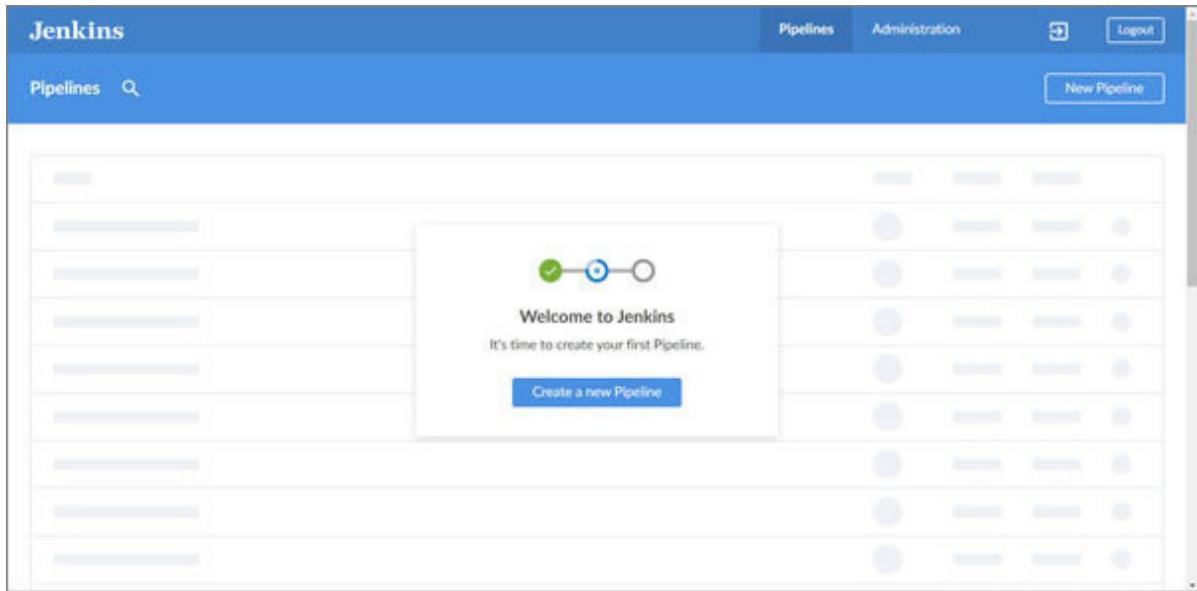


Figure 1.35: Blue Ocean Dashboard

Click on **GitHub** as our code is available in GitHub:

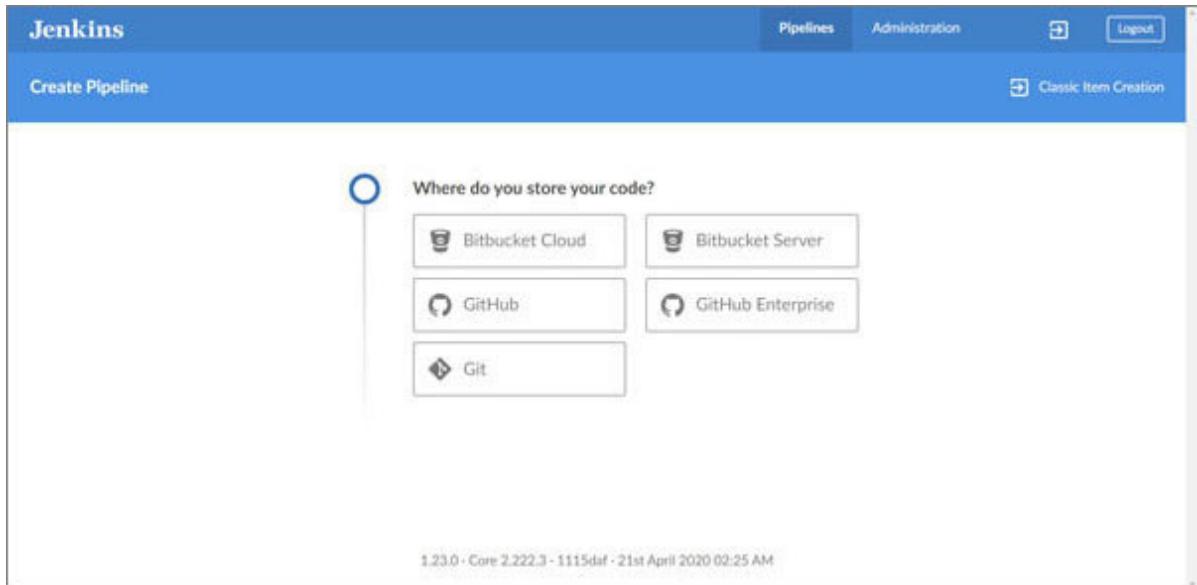


Figure 1.36: Source Code

Create an Access

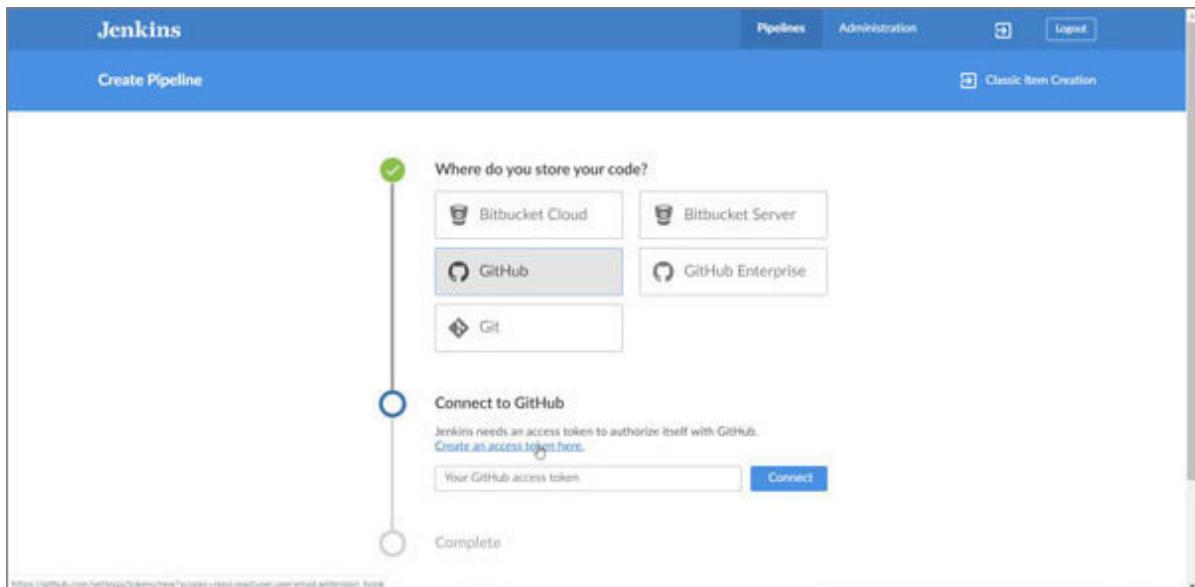


Figure 1.37: Connect to GitHub

Provide a name for an access token and click on

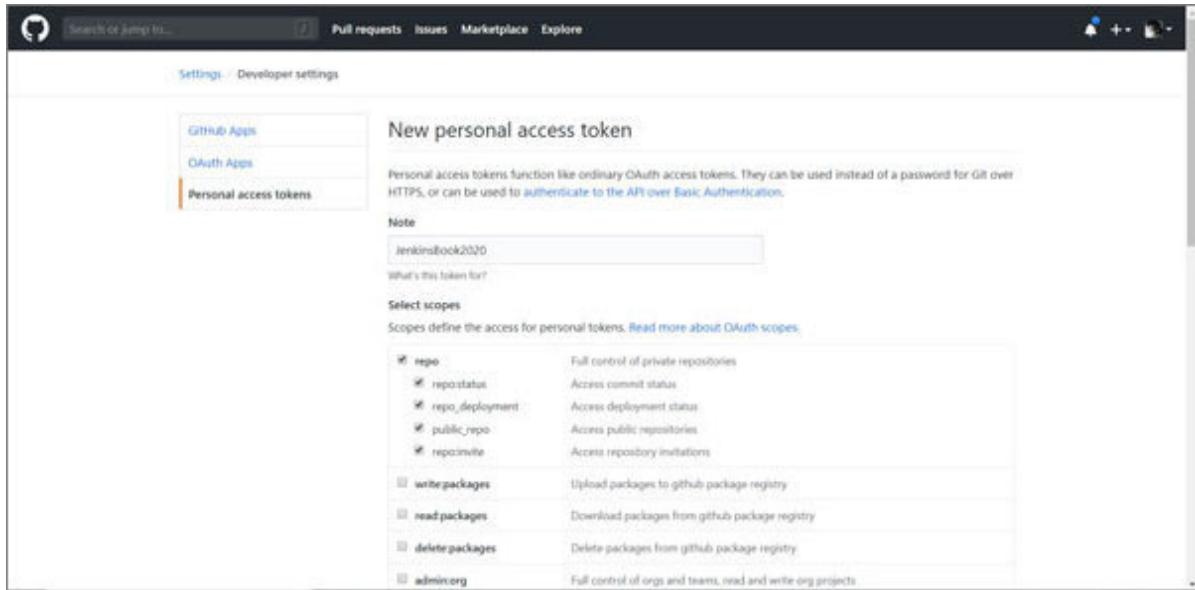


Figure 1.38: Personal Access Token - GitHub

Copy Personal Access Token from

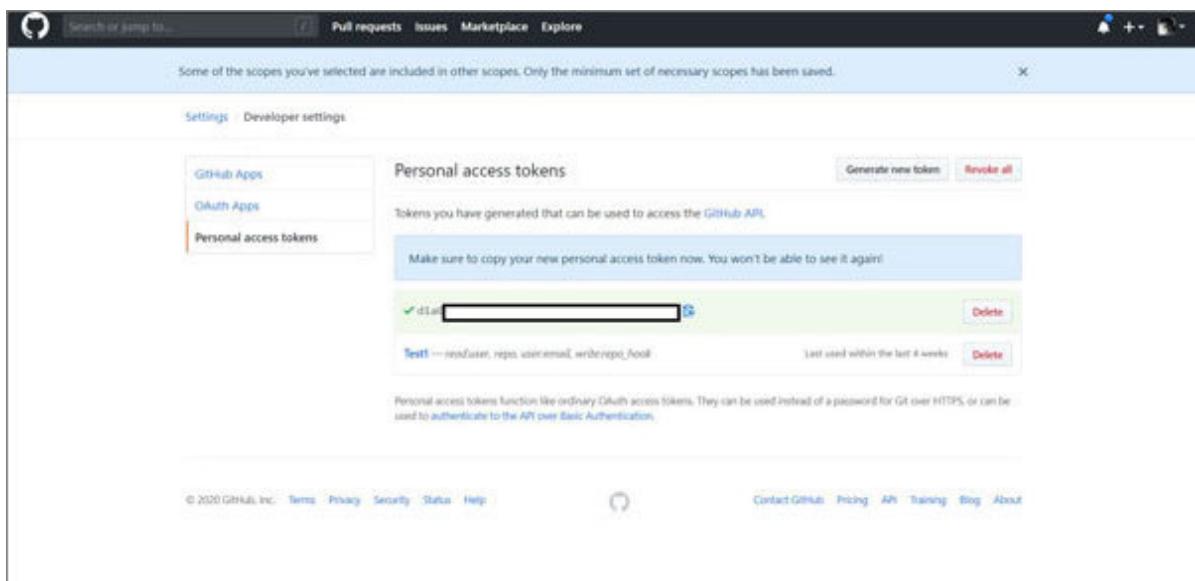


Figure 1.39: PAT in GitHub

Select the **Organization** where all code is available. Select

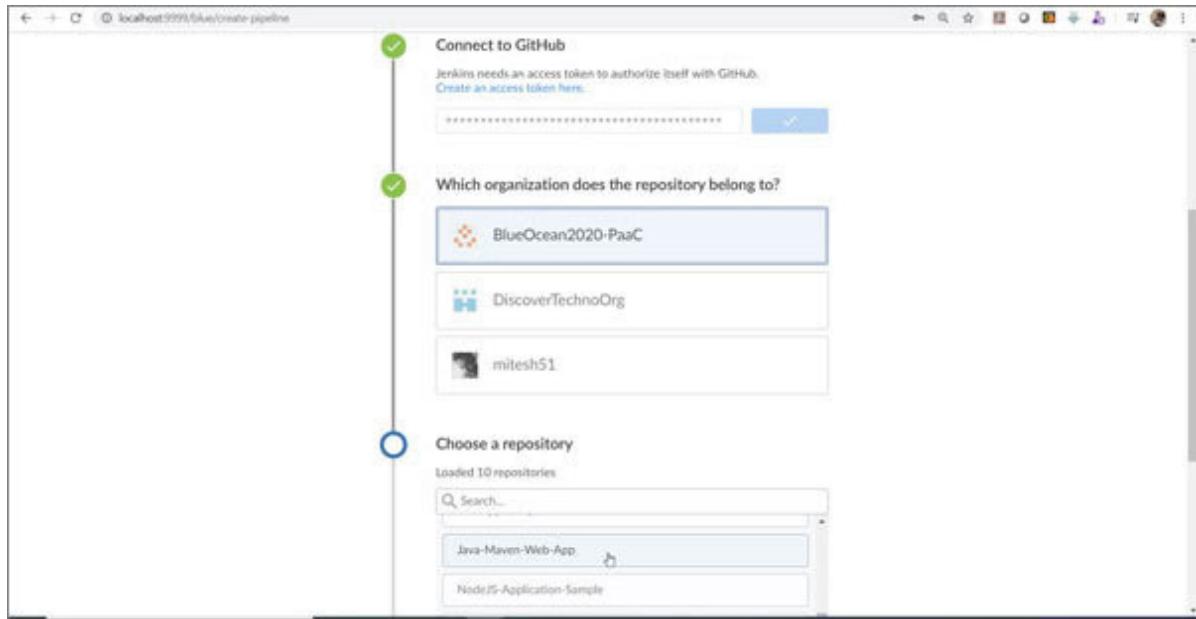


Figure 1.40: GitHub Organization in BlueOcean

If all goes well, verify the branch available in the Blue Ocean Dashboard:

The screenshot shows the Jenkins Blue Ocean - Activity interface. At the top, there's a navigation bar with links for Pipelines, Administration, and Logout. Below the navigation is a header for the 'Java-Maven-Web-App' pipeline, showing a sun icon, the pipeline name, a star icon, and a gear icon. To the right of the header are links for Activity, Branches, and Pull Requests. The main content area has tabs for STATUS, RUN, COMMIT, and BRANCH, with BRANCH selected. It displays a single build entry: Run 1, Commit 32ebef9, Branch master, Message 'Branch indexing', Duration 10s, and Completed status. At the bottom of the page, there's a footer with the text '1.23.0 - Core 2.222.3 · 1115def · 21st April 2020 03:25 AM'.

Figure 1.41: Blue Ocean - Activity

Go to **Classic Jenkins** Dashboard. Click on the

The screenshot shows the Jenkins Classic Dashboard. On the left, there's a sidebar with links for New Item, People, Build History, Manage Jenkins, My Views, Lockable Resources, Credentials, Open Blue Ocean, and New View. The main content area features a search bar and a 'enable auto refresh' button. Below that is a table for the 'Java-Maven-Web-App' pipeline. The table columns are S, W, Name, Last Success, Last Failure, Last Duration, and Fav. The row shows an icon for S, an icon for W, the name 'Java-Maven-Web-App', Last Success at '30 sec - log', Last Failure at 'N/A', Last Duration at '4.1 sec', and Fav buttons. A legend at the bottom indicates three types of Atom feeds. At the bottom of the dashboard, there are sections for Build Queue (No builds in the queue), Build Executor Status (1 idle, 2 idle), and a footer with the text 'Page generated Apr 26, 2020 6:00:53 AM IST RESTART Jenkins ver. 2.222.3'.

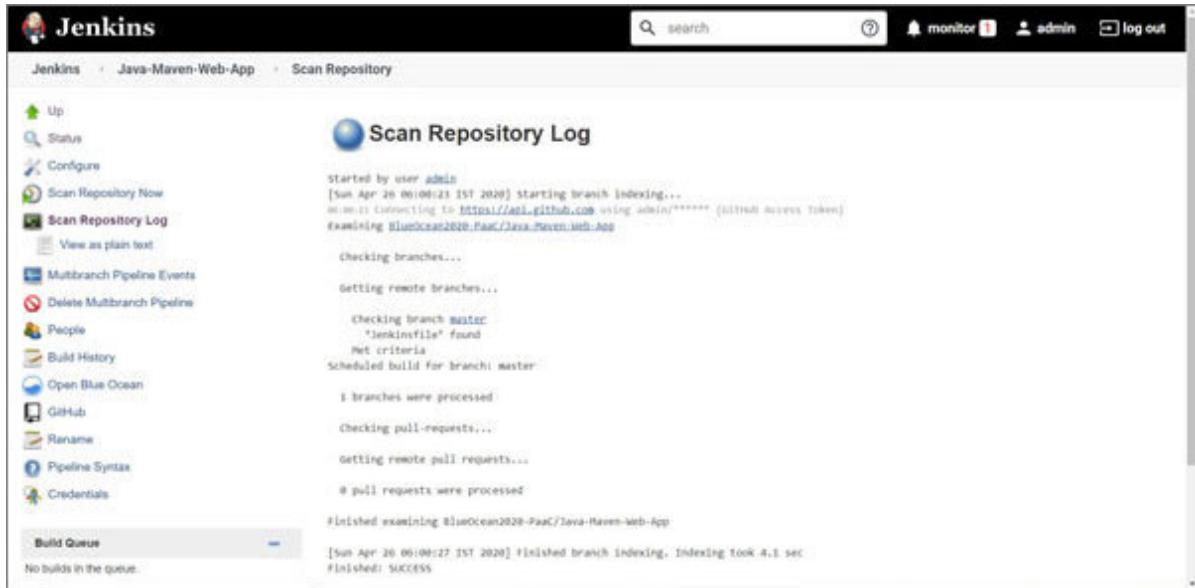
Figure 1.42: Jenkins Dashboard for Pipeline

Verify the

The screenshot shows the Jenkins Java-Maven-Web-App dashboard. On the left, there's a sidebar with various Jenkins management links like Up, Status, Configure, Scan Repository Now, Scan Repository Log, Multibranch Pipeline Events, Delete Multibranch Pipeline, People, Build History, Open Blue Ocean, GitHub, Rename, Pipeline Syntax, and Credentials. Below these are sections for Build Queue (No builds in the queue) and Build Executor Status. The main content area is titled "Java-Maven-Web-App" and describes it as a "Sample Java Web App Using Maven as build tool". It features a table titled "Branches (1)" with one entry: "master". The master branch has an icon of a red circle with a white dot, a "W" status indicator, and the name "master". It shows "N/A" for Last Success, "48 sec - 85" for Last Failure, and "26 sec" for Last Duration. A "Fav" button is also present. A legend at the bottom right indicates three types of feeds: Atom feed for all, Atom feed for failures, and Atom feed for just latest builds.

Figure 1.43: Branches

Click on **Scan Repository Log** here verify in which branch Jenkinsfile is used:



The screenshot shows the Jenkins interface for a project named "Java-Maven-Web-App". The left sidebar contains various Jenkins management links like Up, Status, Configure, Scan Repository Now, Scan Repository Log, Multibranch Pipeline Events, Delete Multibranch Pipeline, People, Build History, Open Blue Ocean, GitHub, Rename, Pipeline Syntax, and Credentials. A "Build Queue" section indicates "No builds in the queue". The main content area is titled "Scan Repository Log" and displays the log output for a scan initiated by user "admin" on April 26, 2020. The log details the indexing process, connecting to GitHub, examining branches, checking remote branches, and processing pull requests. It concludes with a success message.

```
Started by user admin
[Sun Apr 26 06:00:23 IST 2020] Starting branch indexing...
jenkins@ip-10-0-0-111:~$ curl -H "Authorization: token ****" https://api.github.com/repos/BlueOcean2020/PaaS/Java-Maven-Web-App
{
  "id": 1234567890,
  "name": "Java-Maven-Web-App",
  "full_name": "BlueOcean2020/PaaS/Java-Maven-Web-App",
  "owner": {
    "login": "jenkins",
    "id": 1234567890
  },
  "private": false,
  "html_url": "https://github.com/BlueOcean2020/PaaS/Java-Maven-Web-App",
  "description": "A Java Maven Web Application for Jenkins CI/CD Pipeline"
}
Examing BlueOcean2020/PaaS/Java-Maven-Web-App
Checking branches...
Getting remote branches...
Checking branch master
"Jenkinsfile" found
Met criteria
Scheduled build for branch: master
1 branches were processed
Checking pull-requests...
Getting remote pull requests...
0 pull requests were processed
Finished examining BlueOcean2020/PaaS/Java-Maven-Web-App
[Sun Apr 26 06:00:27 IST 2020] Finished branch indexing. Indexing took 4.1 sec
Finished: SUCCESS
```

Figure 1.44: Scan Repository Log

If you want to keep only specific branches then use **Filter with**

The screenshot shows the Jenkins 'Branch Sources' configuration page for a project named 'Java-Maven-Web-App'. The 'Repository HTTPS URL' is set to `https://github.com/BlueOcean2020-PaaS/Java-Maven-Web-App`. Under 'Behaviors', there are three sections: 'Discover branches' (Strategy: All branches), 'Discover pull requests from forks' (Strategy: Merging the pull request with the current target branch revision, Trust: From users with Admin or Write permission), and 'Discover pull requests from origin' (Strategy: Merging the pull request with the current target branch revision). A 'Filter by name (with wildcards)' field contains the value 'master'. At the bottom are 'Save' and 'Apply' buttons.

Figure 1.45: Filter by Name

If `Jenkinsfile` is not in the root directory then provide the path so the pipeline can be executed:

The screenshot shows the Jenkins 'Build Configuration' page. The 'Mode' dropdown is set to 'by Jenkinsfile'. The 'Script Path' input field contains the value 'Jenkinsfile'. Other tabs visible include General, Branch Sources, Scan Repository Triggers, Orphaned Item Strategy, Health metrics, and Properties.

Figure 1.46: Jenkinsfile location

A Jenkinsfile contains pipeline code in the scripted pipeline or declarative pipeline. By default, the pipeline created using BlueOcean us stored in Jenkinsfile. If Jenkinsfile is already available in the repository then Blue Ocean detects it and starts executing the pipeline while creating pipeline in the Blue Ocean dashboard.

Pipeline as a YAML

The Pipeline as YAML concept help to create a pipeline in Jenkins using YAML script and Pipeline as a YAML converts the script in form of Declarative pipeline:

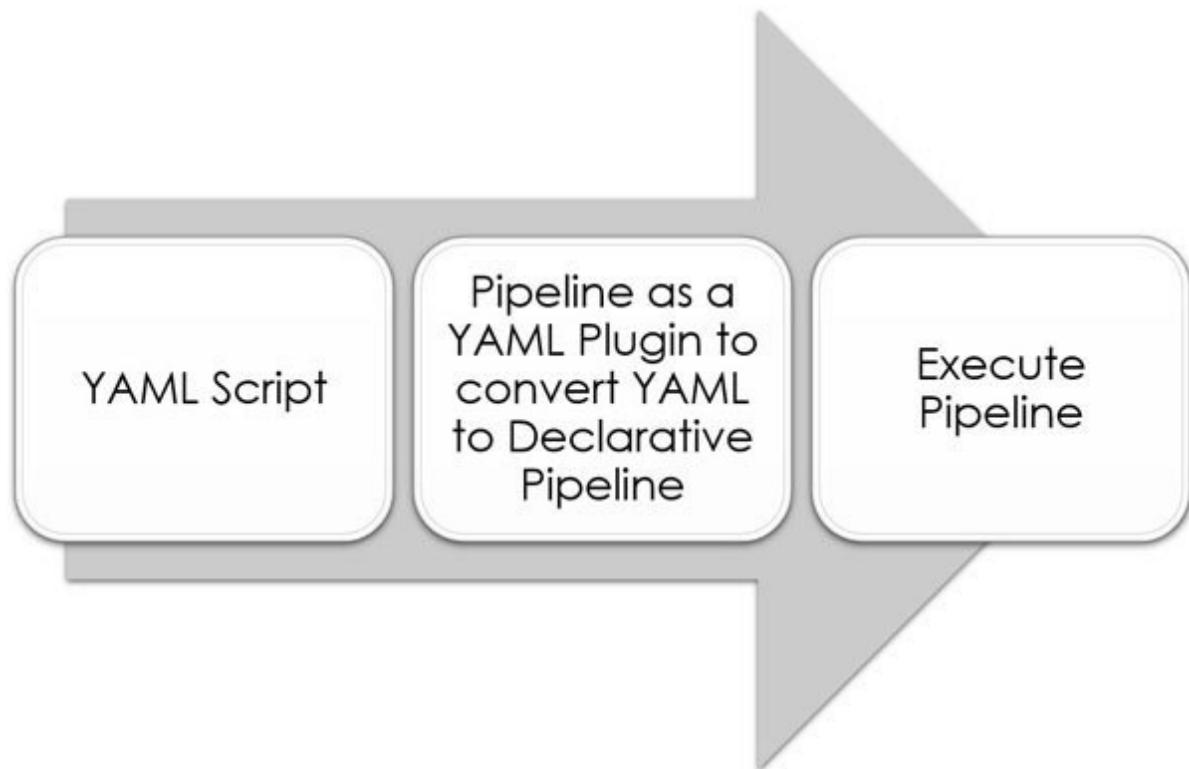


Figure 1.47: Pipeline as a YAML

A plugin is more used or in place as a converter. Once the conversion is completed successfully, the pipeline injects the

newly converted/created declarative pipeline into pipeline runtime.

Following is a sample YAML pipeline:

pipeline:

Define Agent name for this pipeline

agent:

label: 'master'

Define tool for Gradle. Define this tool in Jenkins configuration

tools:

gradle: default

Defines stages

stages:

```
# Define Code Analysis stage
```

- stage: SCA

```
# Define steps
```

steps:

- gradle lint

```
# Define CI Stage
```

- stage: ContinuousIntegration

stages:

- stage: Test

steps:

```
# Use 'sh' step for running gradle commands
```

- sh 'gradle test'

- stage: Build

steps:

script:

- sh "gradle build"

Define post actions

post:

always:

- cleanWs()

In the next chapter, we will discuss different components of pipeline as a YAML in detail.

Conclusion

DevOps is a culture and it essentially supports Culture Transformation to increase quality and make time to market faster. DevOps practices implementation is not a short-term plan. It takes 2–5 years for culture transformation and implements DevOps practices to improve quality and to achieve faster time to market. DevOps practices implementation must be a phase-wise implementation as per the defined maturity model. It is important to perform Proof of Concepts and pilots before the full-fledged implementation of DevOps practices in projects. DevOps is more about culture and mindset with automation, while cloud and containers are more about technology and service with automation. The agile methodology needs DevOps and cloud to get effective results.

In this chapter, we have covered the introduction of DevOps, Continuous Practices, Jenkins, the history of Jenkins and how it evolved over the years, features of Jenkins, prerequisites to install Jenkins, how to run and configure Jenkins, Pipelines, Build pipeline, Scripted pipeline, Declarative pipeline, Blue Ocean, and YAML pipeline.

In the next chapter, we will cover the basics of YAML pipelines in detail.

Points to remember

DevOps is a culture.

Culture Transformation needs a triangle of people, processes, and tools.

Continuous Integration and Continuous Delivery are not DevOps. There are other Continuous Practices available as well.

Culture Transformation is a long-term approach – no quick fixes are available.

DevOps is a tools agnostic approach.

Agile and DevOps are complementary and not mutually exclusive.

DevOps practices implementation and culture transformation are a journey and not only a single project initiative.

Phase-wise implementation should be preferred for DevOps practices implementation.

The maturity model should be dynamic, and it should evolve over time.

Multiple choice questions

DevOps consists of:

Development

Operations

Operations Logic

a and b

DevSecOps consists of:

Development

Operations

Security

a, b, and c

Agile and DevOps are not mutually exclusive.

True

False

Agile and DevOps do not complement each other.

True

False

DevOps culture consists of:

People

Processes

Tools

All of these

None of these

DevOps maturity model should evolve with time.

True

False

Which are the following benefits that can be achieved using DevOps practices implementation?

High Quality

Faster time to Market

Productivity Gains

Continuous Improvement

Continuous Innovation

All of these

None of these

Which of the following ways can be used to create a pipeline?

Build pipeline Plugin

Scripted pipeline

Declarative pipeline

Blue Ocean

All of the above

Following is the scripted pipeline: State whether it is true or false.

```
pipeline {  
    /* Stages and Steps */  
}
```

False

True

Following is the declarative pipeline: State whether it is true or false.

```
node {  
/* Stages and Steps */  
}
```

False

True

Answer

d

d

a

b

d

a

f

e

a

a

Questions

What is DevOps?

How Containers are better than Virtual Machines?

What is the difference between Continuous Delivery and Continuous Deployment?

Why Value Stream is important?

What are the continuous practices that help culture transformation?

Why DevOps practices implementation has to be phase-wise?

Explain different ways to create pipeline.

Explain the difference between the scripted and declarative pipelines.

Explain the significance of Blue Ocean.

Key terms

Build Traditional way to create a pipeline using upstream and downstream jobs.

Scripted Boon for developers to create dynamic pipelines using programming constructs.

Declarative Boon for beginners to write pipeline script using DSL.

Blue Easy way to create a declarative pipeline using a completely new user experience in Jenkins.

File that contains Scripted or Declarative pipeline by default.

Multi-branch Pipeline that executes the script for all branches where Jenkinsfile is available and valid.

CHAPTER 2

Basic Components of YAML Pipeline

Jenkins is an open-source automation server that is written in Java. Jenkins has evolved a lot based on the evolution of technology and from a user experience perspective. Jenkins has improved not only its technical features but user experience or UI too. After Jenkins 2.0 release in 2016, the Game has changed. Jenkins has become a popular choice for the DevOps community as an Automation server and not only a Continuous Integration server.

In this book, our main objective is to understand the basic components of YAML based pipeline.

Structure

In this chapter, we will discuss the following topics:

Introduction

Controller-agent architecture

Controller

Agent

Agent pools

Executors

YAML pipeline structure

Jenkins

Azure DevOps

Objectives

This chapter will introduce the basic components of YAML pipeline. After studying this unit, you should be able to:

Create agents and configure agents to execute Jenkins jobs.

Write pipeline as a YAML script for DevOps practices implementation.

Controller-agent architecture

Initially, a single Jenkins server can serve to build jobs but over time Application components increases, applications change, new projects come, and scenario changes within a year. Jenkins is almost out of resources if all the Projects and all the components are managed on a single server. It is bound to go down in such a situation.

Freestyle project is very important in Jenkins as it can be used in orchestration with command execution such as batch or shell commands.

Controller-agent architecture or distributed architecture comes to the rescue in such a situation. Agents can be physical systems, Virtual machines, Docker Containers, Cloud-based virtual machines such as Amazon EC2, Azure virtual machines, and so on:

on: on:

on: on: on: on: on: on: on: on: on: on: on: on:

on: on: on: on: on: on: on: on: on:

on: on: on: on: on: on:
on: on: on: on:

on:
on: on: on: on: on: on: on: on: on:
on:
on:
on: on: on: on: on:
on:

Table 2.1: Jenkins controller vs. Jenkins agent

Let us check if agent can access the controller and vice versa with ping command and IP address:

Go to Virtualbox VM and ping the IP address of the system where Jenkins is installed:



Figure 2.1: Virtual machine as Jenkins agent

Pipeline template helps to orchestrate tasks that can be executed on multiple build agents.

Go to **Manage Jenkins|Manage Nodes** and click on a **New**

The screenshot shows the Jenkins 'Nodes' page. At the top, there is a navigation bar with links for 'Back to Dashboard', 'Manage Jenkins', 'New Node', 'Configure Clouds', and 'Node Monitoring'. On the right side of the header, there is a search bar, a help icon, and a 'monitors' button. Below the header, there is a table titled 'Nodes' with columns: Name, Architecture, Clock Difference, Free Disk Space, Free Swap Space, Free Temp Space, and Response Time. A single row is present in the table, representing the 'master' node. The 'master' node is listed as 'Windows 10 (x64)', 'In sync', with 131.71 GB free disk space, 10.58 GB free swap space, 69.73 GB free temp space, and a response time of 2 min 2 sec. There is a 'Refresh status' button at the bottom right of the table. The footer of the page includes a 'Page generated' timestamp and a 'RESTART Jenkins ver. 2.222.3' link.

Figure 2.2: Nodes in Jenkins

Provide node name and click

The screenshot shows the Jenkins 'New Node' configuration dialog. The 'Node name' field is set to 'Maven'. The 'Permanent Agent' type is selected. A descriptive text explains that this adds a plain, permanent agent to Jenkins, suitable for physical computers or virtual machines managed outside Jenkins. An 'OK' button is visible at the bottom left of the dialog. The background shows the Jenkins 'Nodes' page with one node listed.

Figure 2.3: New node

Configure `/home/mitesh/Desktop/JenkinsAgent` as remote root directory:



Figure 2.4: Node Configuration

The agent directive suggests where the pipeline or a specific stage must run. Following are the configuration details for an agent:

```
agent: agent: agent: agent: agent: agent: agent: agent:
```

```
agent: agent: agent: agent: agent: agent: agent: agent:
```


agent: agent: agent: agent: agent: agent: agent: agent: agent: agent: agent: agent:
--

agent: agent:

agent: agent: agent: agent: agent: agent: agent: agent: agent: agent: agent: agent: agent: agent:
--

Table 2.2: Jenkins Agent details

An agent is not connected. Let us try to connect the Maven agent. Change the Jenkins URL if it is using the Use IP address so Jenkins can be accessed outside the system where it is installed:

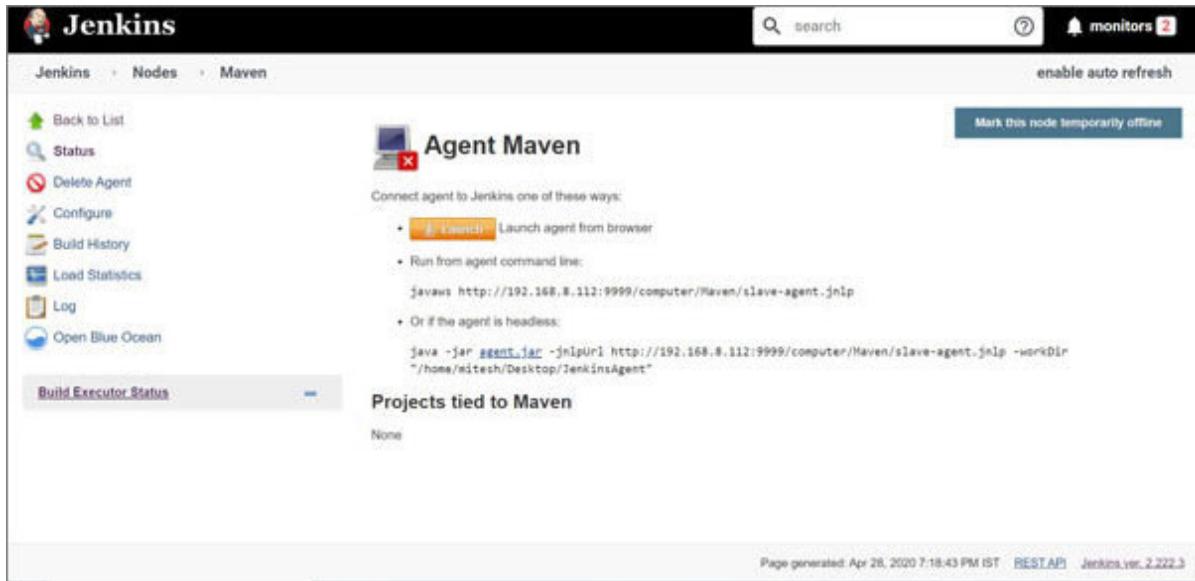


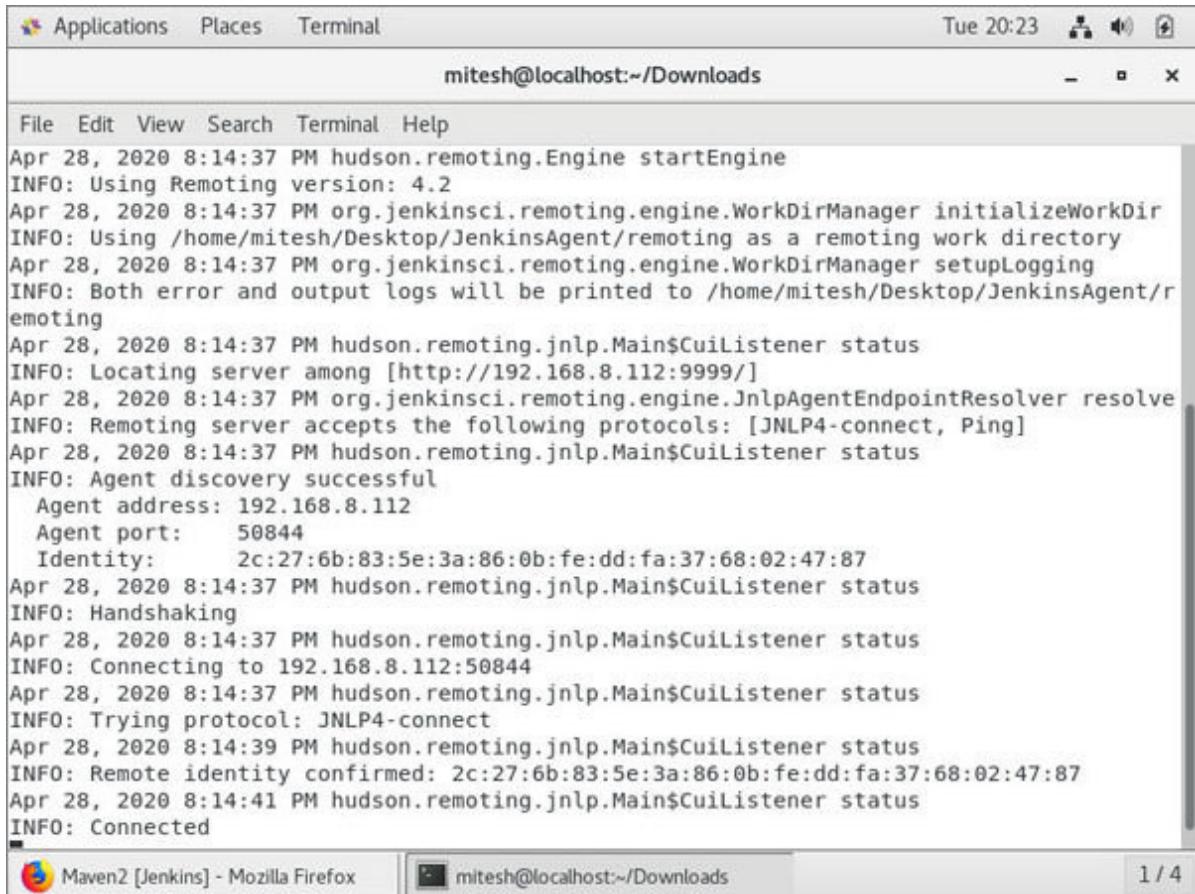
Figure 2.5: Disconnected agent

Go to Virtualbox VM. Open Jenkins URL and open Nodes.
Click on agent.jar and download it in the local system.
Similarly, execute **jnlpUrl** in the browser and it will ask you to save

External Job template helps to record the execution of a process executed outside Jenkins.

Keep both files in the same folder. For example,

Execute the following command:



A screenshot of a terminal window titled "mitesh@localhost:~/Downloads". The window shows a log of Jenkins agent connection attempts. The log includes timestamped messages from the "hudson.remoting.Engine" and "org.jenkinsci.remoting.engine.WorkDirManager" classes, indicating the start of the engine, initialization of the work directory, setup of logging, and discovery of the agent's address and port. It also shows the agent attempting to connect to the Jenkins master at 192.168.8.112:50844 using the JNLP4-connect protocol.

```
File Edit View Search Terminal Help
Apr 28, 2020 8:14:37 PM hudson.remoting.Engine startEngine
INFO: Using Remoting version: 4.2
Apr 28, 2020 8:14:37 PM org.jenkinsci.remoting.engine.WorkDirManager initializeWorkDir
INFO: Using /home/mitesh/Desktop/JenkinsAgent/remoting as a remoting work directory
Apr 28, 2020 8:14:37 PM org.jenkinsci.remoting.engine.WorkDirManager setupLogging
INFO: Both error and output logs will be printed to /home/mitesh/Desktop/JenkinsAgent/r
emoting
Apr 28, 2020 8:14:37 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Locating server among [http://192.168.8.112:9999/]
Apr 28, 2020 8:14:37 PM org.jenkinsci.remoting.engine.JnlpAgentEndpointResolver resolve
INFO: Remoting server accepts the following protocols: [JNLP4-connect, Ping]
Apr 28, 2020 8:14:37 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Agent discovery successful
  Agent address: 192.168.8.112
  Agent port: 50844
  Identity: 2c:27:6b:83:5e:3a:86:0b:fe:dd:fa:37:68:02:47:87
Apr 28, 2020 8:14:37 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Handshaking
Apr 28, 2020 8:14:37 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Connecting to 192.168.8.112:50844
Apr 28, 2020 8:14:37 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Trying protocol: JNLP4-connect
Apr 28, 2020 8:14:39 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Remote identity confirmed: 2c:27:6b:83:5e:3a:86:0b:fe:dd:fa:37:68:02:47:87
Apr 28, 2020 8:14:41 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Connected
```

Figure 2.6: Connect Agent with the controller using the command line

Verify that agent is connected in Jenkins Dashboard:

The screenshot shows the Jenkins interface for managing agents. On the left, there's a sidebar with links like 'Back to List', 'Status', 'Delete Agent', 'Configure', 'Build History', 'Load Statistics', 'Script Console', 'Log', 'System Information', 'Disconnect', and 'Open Blue Ocean'. The main content area is titled 'Agent Maven' and displays the message 'Agent is connected.' Below this, there's a section for 'Labels' with 'JavaWebApp-Build' listed. A table titled 'Projects tied to Maven' shows one project: 'Build' with a status of 'Last Success' at '46 min - 85' and 'Last Failure' at '56 min - 83'. The table includes columns for Status (S), Warning (W), Name, Last Success, Last Failure, and Last Duration. A legend at the bottom right indicates colors for Atom feed for all, Atom feed for failures, and Atom feed for just latest builds. At the bottom of the page, it says 'Page generated: Apr 28, 2020 8:24:37 PM IST' and 'Jenkins ver. 2.222.3'.

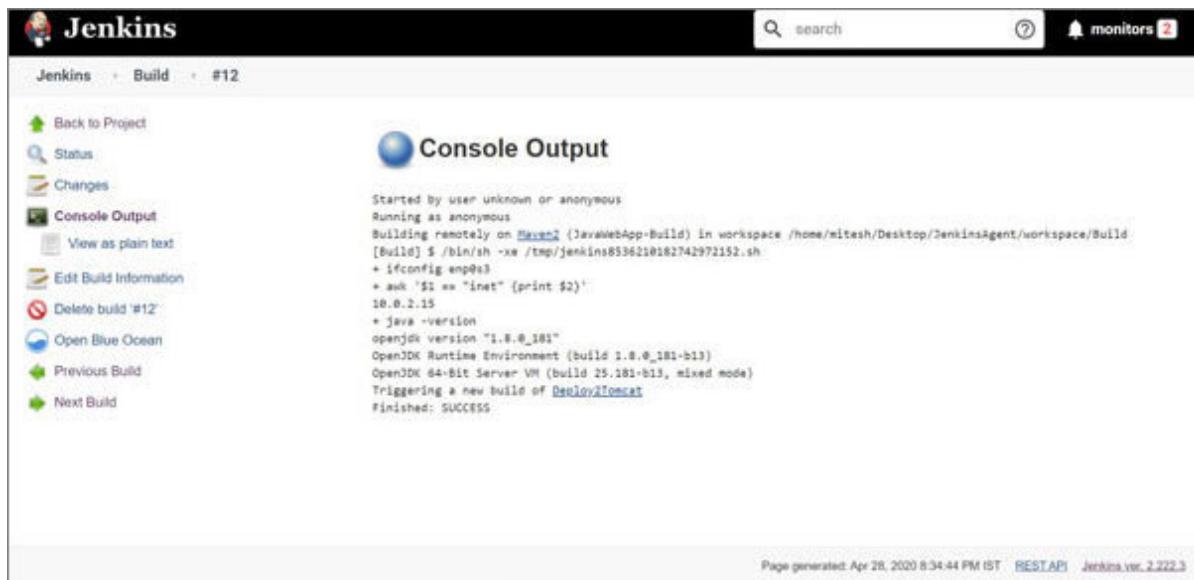
Figure 2.7: Connected agent

The multi-configuration project template helps to orchestrate projects that need a large number of different configurations:

The screenshot shows the 'Multi_Config_Project' configuration matrix screen. The top navigation bar includes 'General', 'Advanced Project Options', 'Source Code Management', 'Build Triggers', 'Configuration Matrix' (which is currently selected), and 'Build Environment'. Below the navigation, there's a 'Build' tab and a 'Post-build Actions' section. The 'Configuration Matrix' section contains an 'Add axis' button and three checkboxes: 'Combination Filter', 'Run each configuration sequentially', and 'Execute touchstone builds first'. The 'Build Environment' section contains several checkboxes: 'Delete workspace before build starts', 'Use secret text(s) or file(s)', 'Provide Configuration files', 'Abort the build if it's stuck', 'Add timestamps to the Console Output', and 'Ant/Ivy-Artifactory Integration'. At the bottom, there's a 'Generate Release Notes' checkbox, a 'Save' button, and an 'Apply' button.

Figure 2.8: Multi-configuration project

Execute the job and verify the console output. Note IP address and Java version:



The screenshot shows the Jenkins interface for a multi-configuration project. On the left, there's a sidebar with links like Back to Project, Status, Changes, and Console Output (which is currently selected). The main area is titled "Console Output" and displays the following log output:

```
Started by user unknown or anonymous
Running as anonymous
Building remotely on Maven2 (JavaWebApp-Build) in workspace /home/mitesh/Desktop/JenkinsAgent/workspace/Build
[Build] $ /bin/sh -xe /tmp/jenkins8536210182742972152.sh
+ ifconfig enp0s3
+ awk '$1 == "inet" {print $2}'
10.0.2.15
+ java -version
openjdk version "1.8.0_181"
OpenJDK Runtime Environment (build 1.8.0_181-b13)
OpenJDK 64-Bit Server VM (build 25.181-b13, mixed mode)
Triggering a new build of Deploy2Tomcat
Finished: SUCCESS
```

At the bottom right of the console output, it says "Page generated: Apr 28, 2020 8:34:44 PM IST REST API Jenkins ver. 2.222.3".

Figure 2.9: Build execution on a specific agent

This is how an agent can be connected and jobs can be delegated to agents for execution so the load on the controller is not much and you don't need to set up the entire environment on the Jenkins controller.

NOTE: The happy team performs better. Appreciate team members and celebrate success to motivate teams.

Folder is like a container that can be useful to store nested items in it. It is more like grouping things.

Let us discuss agent pools.

Agent pools

Labels in agent configuration helps to balance load or make agents available for execution. It is kind of a pool of agents available for service:

Provide the same label while configuring the Jenkins agent. Let us assume we want to have two VMS for Java build execution. Make them Jenkins agents as we did in the earlier section and provide the same label to each of them:



Figure 2.10: Labels in agent configuration

Multibranch pipeline template creates a set of pipelines based on detected branches having Jenkinsfile in one repository.

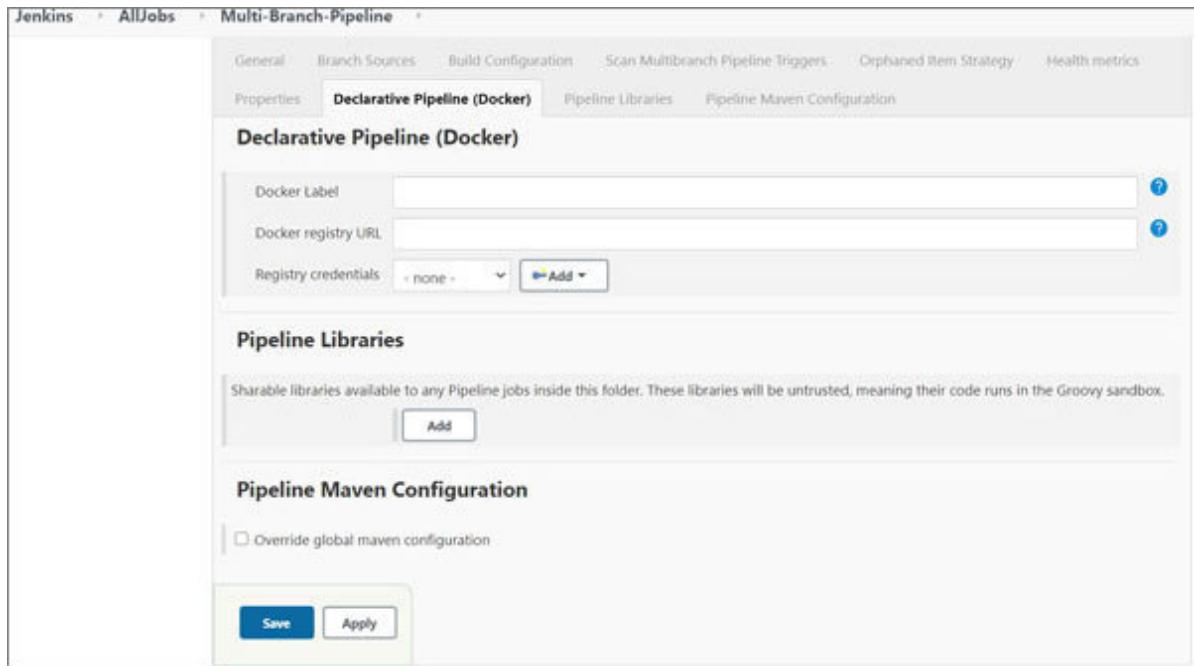


Figure 2.11: Multibranch pipeline

Click on the label link available on the agent page and we can see how many agents have the same label attached to it:

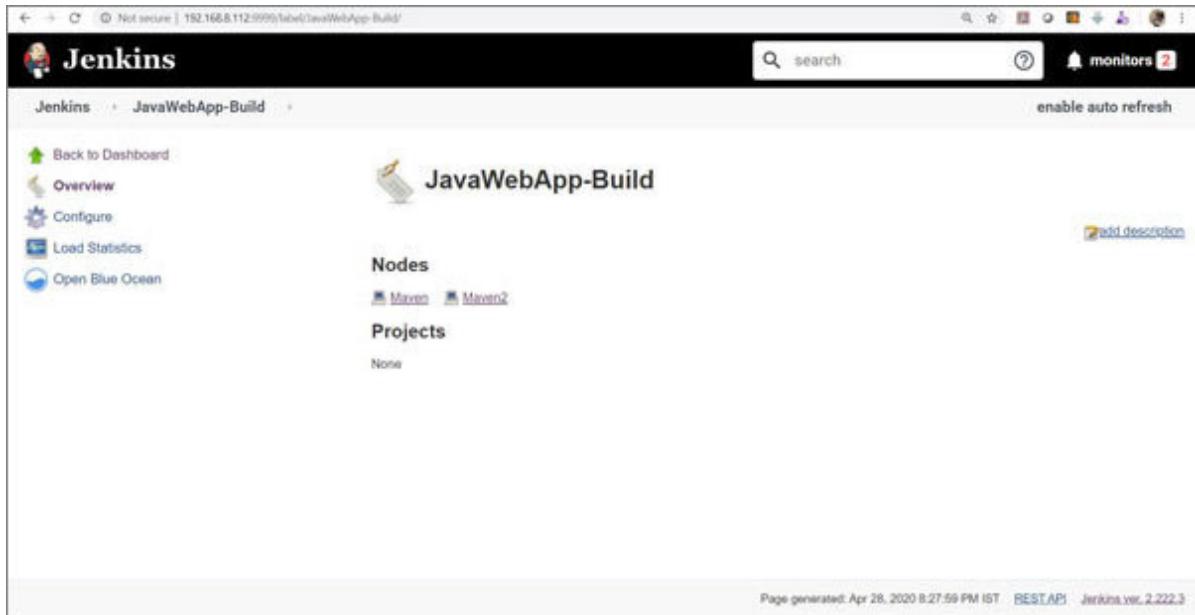


Figure 2.12: Nodes with the same Label

While configuring any Build Job or pipeline, provide Label as an agent and not the name of it:

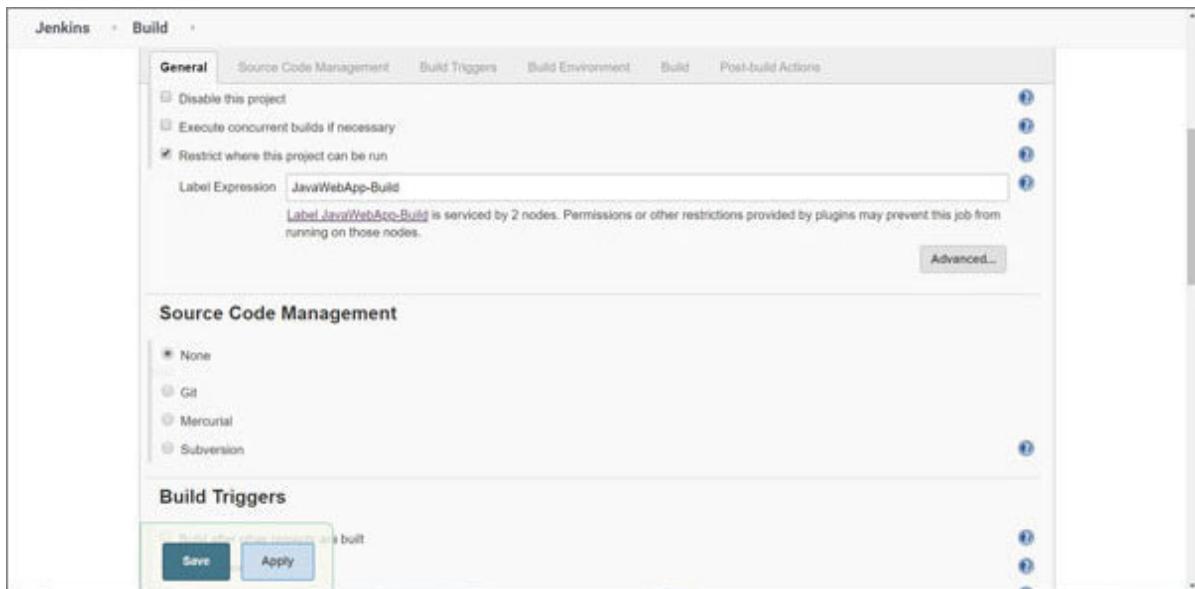


Figure 2.13: Label Name as Label Expression

Execute the build and verify the output. Note the IP address and agent in this case which executed the build:



The screenshot shows the Jenkins interface for a build labeled '#15'. The left sidebar includes links for Back to Project, Status, Changes, Console Output (which is currently selected), View as plain text, Edit Build Information, Delete build #15, Open Blue Ocean, Previous Build, and Next Build. The main content area is titled 'Console Output' and displays the following log output:

```
Started by user unknown or anonymous
Running as anonymous
Building remotely on labeled (JavaWebApp-Build) in workspace /home/mitesh/Desktop/JenkinsAgent/workspace/Build
[Build] $ /bin/sh -xe /tmp/jenkins195586346437576746.sh
+ ifconfig enp0s3
+ awk '$1 == "inet" {print $2}'
10.0.2.15
+ java -version
openjdk version "1.8.0_181"
OpenJDK Runtime Environment (build 1.8.0_181-b13)
OpenJDK 64-Bit Server VM (build 25.181-b13, mixed mode)
Triggering a new build of DeployToTomcat
Finished: SUCCESS
```

At the bottom right, it says 'Page generated: Apr 28, 2020 8:32:45 PM IST REST API Jenkins ver. 2.222.3'.

Figure 2.14: Build execution on Labeled node

To verify whether both the agents are available for execution or not, make the agent offline. Hence only one agent will be active to serve the build execution request:

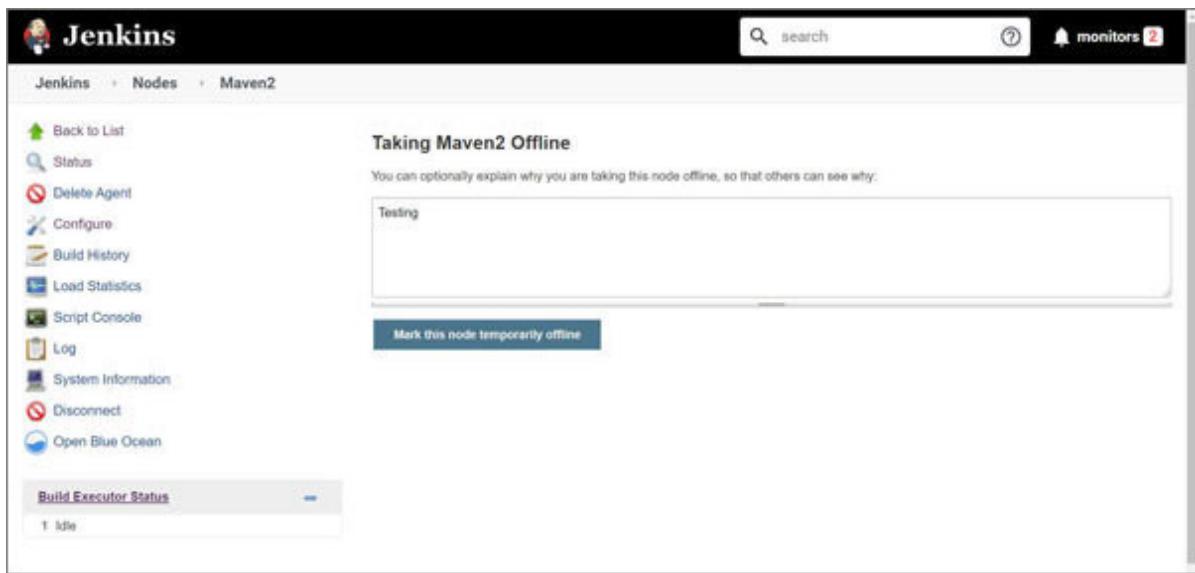


Figure 2.15: Make Agent offline

Maven2 agent on which earlier job was executed is disconnected and not available for use:

The screenshot shows the Jenkins interface for managing a node named 'Agent Maven2'. The top navigation bar includes links for 'Jenkins', 'Nodes', and 'Maven2'. On the left, a sidebar lists various management options: 'Back to List', 'Status', 'Delete Agent', 'Configure', 'Build History', 'Load Statistics', 'Script Console', 'Log', 'System Information', 'Disconnect', and 'Open Blue Ocean'. Below this is a 'Build Executor Status' button. The main content area displays the node's status: 'Agent Maven2' with a computer icon, last updated 'Apr 28, 2020 8:32:11 PM', and a warning message 'Disconnected by anonymous : Testing'. It also indicates that the agent is connected. Under 'Labels', the node is assigned to 'JavaWebApp-Build'. A section titled 'Projects tied to Maven2' shows 'None'.

Figure 2.16: Disconnected Agent

Execute the build again. Now verify the console output and you will find that the IP address is different as a different agent that was having the same label executed the build request as it was available to serve the build execution request:

The screenshot shows the Jenkins interface for a build named 'JavaWebApp-Build' with build number '#16'. The left sidebar contains links for 'Back to Project', 'Status', 'Changes', 'Console Output' (which is currently selected), 'View as plain text', 'Edit Build Information', 'Delete build #16', 'Open Blue Ocean', and 'Previous Build'. The main content area is titled 'Console Output' and displays the following log output:

```
Started by user unknown or anonymous
Running as anonymous
Building remotely on mitesh (JavaWebApp-Build) in workspace /home/mitesh/Desktop/JenkinsAgent/workspace/Build
[Build] $ /bin/sh -xe /tmp/jenkins5485269910885105086.sh
+ ifconfig enp0s3
+ awk '$2 == "inet" {print $2}'
10.0.2.4
+ java -version
openjdk version "1.8.0_181"
OpenJDK Runtime Environment (build 1.8.0_181-b13)
OpenJDK 64-Bit Server VM (build 25.181-b13, mixed mode)
Triggering a new build of Deploy2Tomcat
Finished: SUCCESS
```

At the bottom right of the page, it says 'Page generated: Apr 28, 2020 8:33:07 PM IST REST API Jenkins ver. 2.222.3'.

Figure 2.17: Build execution on another node

In the next section, we will discuss YAML pipeline structure.

[YAML pipeline structure](#)

We can create YAML pipeline after the installation of the plugin available at:

Go to **Manage Jenkins | Manage Plugins | Find Pipeline As YAML (Incubated)** and install it. Now, we are ready to work with YAML pipelines.

Let us understand YAML pipeline structure:

Go to Jenkins dashboard and click on **New Create a pipeline** by selecting pipeline job.

Select **Pipeline As YAML** from the **Definition** list box in the pipeline section.

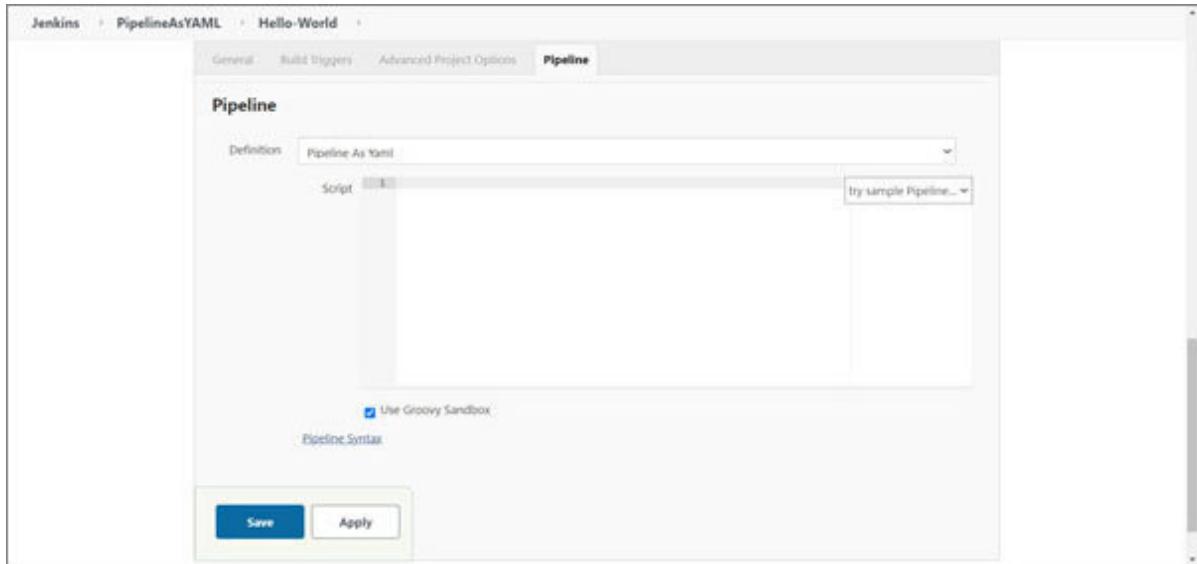
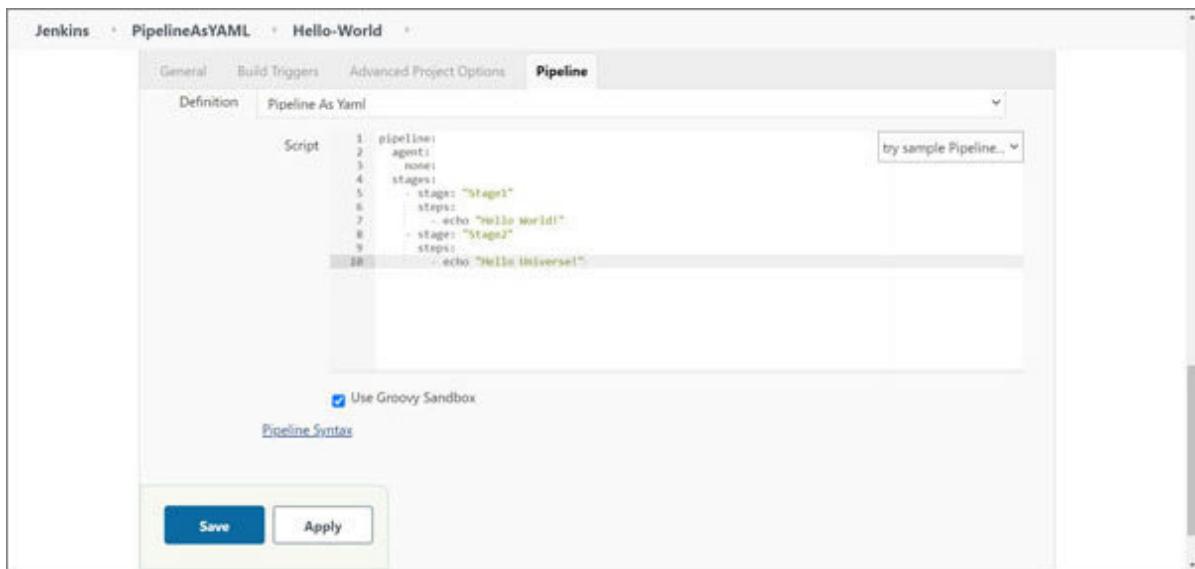


Figure 2.18: Pipeline as a YAML

Stages help to organize different tasks into logical blocks. A stage has the name and set of steps. A stage can display the message, perform static code analysis or perform unit testing can calculate code coverage.

Let us write a sample pipeline in YAML:

Let us write a simple YAML pipeline with two stages that display messages on stage execution.



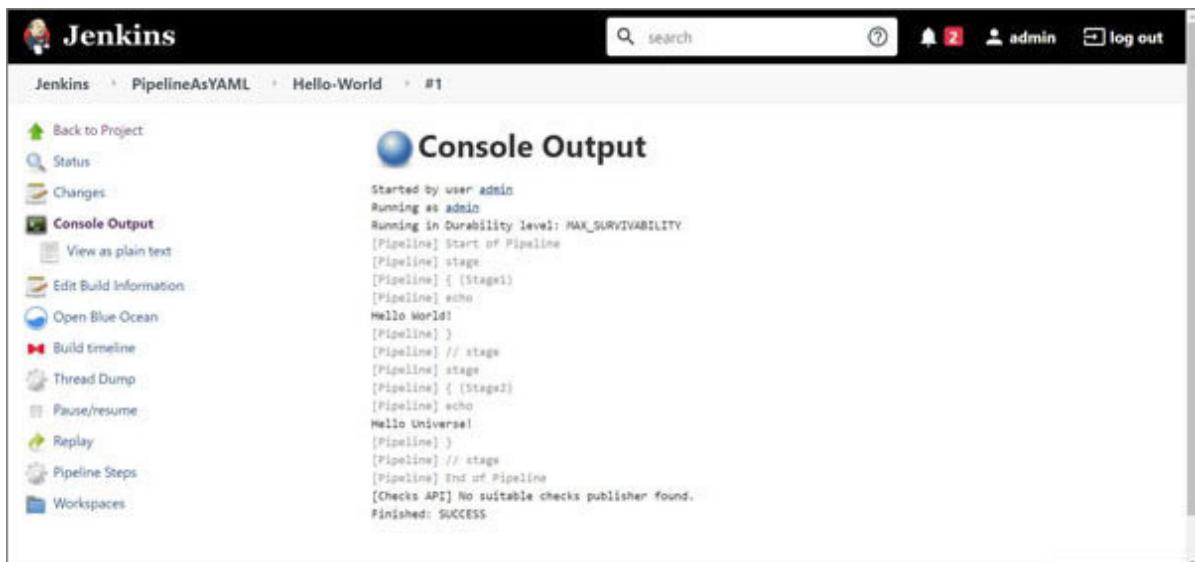
The screenshot shows the Jenkins PipelineAsYAML configuration interface. At the top, there are tabs for General, Build Triggers, Advanced Project Options, and Pipeline. The Pipeline tab is selected. Below it, there are two tabs: Definition and Pipeline As Yaml. The Pipeline As Yaml tab is selected, displaying a code editor with the following YAML script:

```
1 pipeline:
2   agent:
3     none:
4   stages:
5     - stage: "Stage1"
6       steps:
7         - echo "Hello world!"
8     - stage: "Stage2"
9       steps:
10        - echo "Hello Universe!"
```

Below the code editor, there is a checkbox labeled "Use Groovy Sandbox". Underneath the code editor, there are "Save" and "Apply" buttons.

Figure 2.19: YAML pipeline with stages

Click on **Save** and then click on **Build**. Verify the Console Output and the messages based on the stage execution.



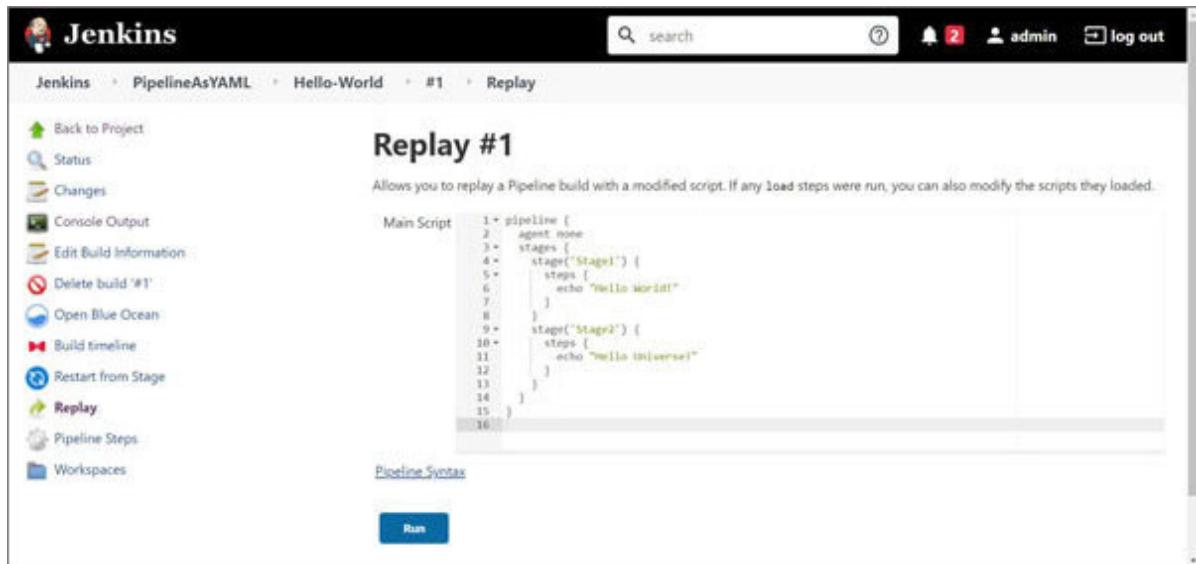
The screenshot shows the Jenkins Console Output page for a build named "Hello-World #1". The left sidebar contains links such as Back to Project, Status, Changes, Console Output (which is selected), View as plain text, Edit Build Information, Open Blue Ocean, Build timeline, Thread Dump, Pause/resume, Replay, Pipeline Steps, and Workspaces. The main content area is titled "Console Output" and displays the following log output:

```
Started by user admin
Running as admin
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] Start of Pipeline
[Pipeline] stage
[Pipeline] {
  [Pipeline] stage
  [Pipeline] {
    [Pipeline] echo
    Hello World!
  }
}
[Pipeline] // stage
[Pipeline] stage
[Pipeline] {
  [Pipeline] stage
  [Pipeline] {
    [Pipeline] echo
    Hello Universe!
  }
}
[Pipeline] // stages
[Pipeline] End of Pipeline
[Checks API] No suitable checks publisher found.
Finished: SUCCESS
```

Figure 2.20: Console Output of YAML pipeline execution

Click on the

Verify that the script available is not YAML script but pipeline in Declarative syntax.



The screenshot shows the Jenkins PipelineAsYAML interface. The top navigation bar includes the Jenkins logo, search bar, notifications (2), user account (admin), and log out link. The main page title is "PipelineAsYAML > Hello-World > #1 > Replay". On the left, there's a sidebar with links: Back to Project, Status, Changes, Console Output, Edit Build Information, Delete build '#1', Open Blue Ocean, Build timeline, Restart from Stage, Replay (which is highlighted in blue), Pipeline Steps, and Workspaces. The central area is titled "Replay #1" and contains the following text: "Allows you to replay a Pipeline build with a modified script. If any load steps were run, you can also modify the scripts they loaded." Below this is the "Main Script" code block:

```
1+ pipeline {
2+   agent none
3+   stages {
4+     stage('Stage1') {
5+       steps [
6+         echo "Hello World"
7+       ]
8+     }
9+     stage('Stage2') {
10+    steps [
11+      echo "Hello Universe"
12+    ]
13+
14+
15+
16+ }
```

At the bottom of the script area, there's a "Pipeline Syntax" link and a large blue "Run" button.

Figure 2.21: Replay pipeline

It is simple to verify YAML pipeline script.

Go to the newly created pipeline and click on pipeline Syntax and then click on **Pipeline As YAML**

This helper aims to provide an easy way to convert **Pipeline As YAML** to **Pipeline Declarative Script Format** and also validating Pipeline Declarative Script. Enter your YAML script and click on **Convert To Pipeline Declarative**

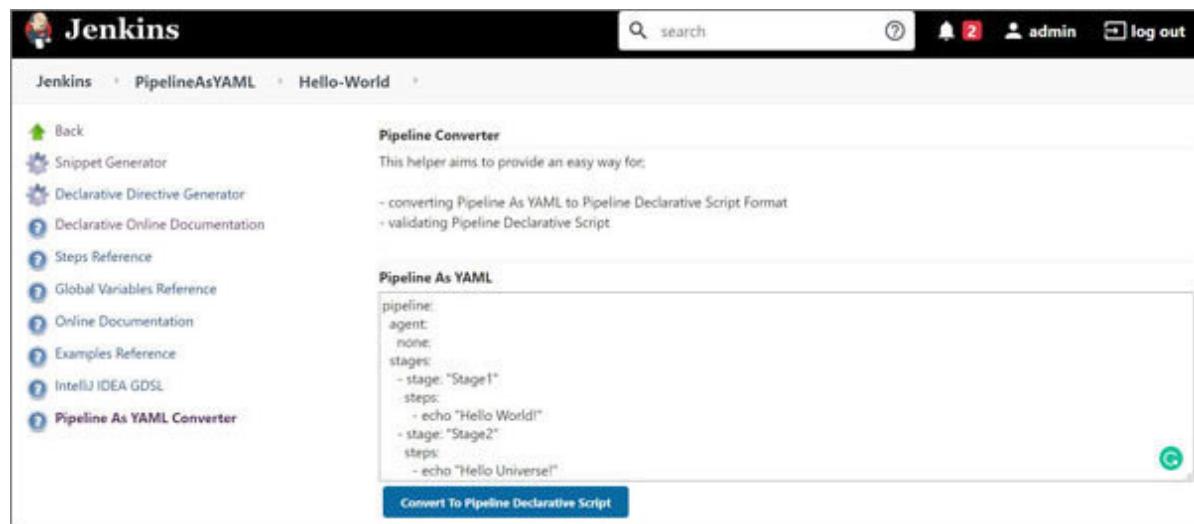


Figure 2.22: Convert to pipeline Declarative script

If the script is valid then it will be converted in the Declarative pipeline syntax.

The screenshot shows the Jenkins PipelineAsYAML interface. On the left, there's a sidebar with links: Global Variables Reference, Online Documentation, Examples Reference, IntelliJ IDEA GDSL, and Pipeline As YAML Converter. The main area has two tabs: 'Pipeline As YAML' and 'Pipeline Declarative Script'. Under 'Pipeline As YAML', the following code is displayed:

```
pipeline:
  agent:
    none:
  stages:
    - stage: "Stage1"
      steps:
        - echo "Hello World!"
    - stage: "Stage2"
      steps:
        - echo "Hello Universe!"
```

Below this is a blue button labeled 'Convert To Pipeline Declarative Script'. Under 'Pipeline Declarative Script', the following Groovy code is shown:

```
pipeline {
  agent none
  stages {
    stage('Stage1') {
      steps {
        echo "Hello World!"
      }
    }
    stage('Stage2') {
      steps {
        echo "Hello Universe!"
      }
    }
  }
}
```

At the bottom is a blue button labeled 'Validate Pipeline'.

Figure 2.23: Valid pipeline declarative script

If YAML script is not valid then the exception will be thrown.

The screenshot shows the Jenkins PipelineAsYAML interface. The sidebar and tabs are identical to Figure 2.23. However, the 'Pipeline As YAML' tab contains an invalid YAML script:

```
pipeline:
  agent:
    none:
  stages:
    - stage: "Stage1"
      steps:
        - echo "Hello Universe!"
        - stage: "Stage1"
          steps:
            - echo "Hello World!"
```

This results in an error message in the 'Pipeline Declarative Script' tab: 'Exception happened while converting. Please check the logs'.

Figure 2.24: Convert to pipeline declarative script – exception

Click on **Validate Pipeline** to get more details on the exception and fix the issues.

The screenshot shows the Jenkins Pipeline As YAML converter interface. On the left, there's a sidebar with links: Steps Reference, Global Variables Reference, Online Documentation, Examples Reference, IntelliJ IDEA Groovy, and Pipeline As YAML Converter (which is selected). The main area has two tabs: 'Pipeline As YAML' and 'Pipeline Declarative Script'. Under 'Pipeline As YAML', there's a code editor with the following YAML:

```
pipeline:
  agent:
    none:
  stages:
    - stage: "Stage1"
      steps:
        - echo "Hello Universe!"
    - stage: "Stage1"
      steps:
        - echo "Hello World!"
```

Below the code editor is a 'Convert To Pipeline Declarative Script' button. Under 'Pipeline Declarative Script', it says 'Exception happened while converting. Please check the logs.' At the bottom, there's a 'Validate Pipeline' button. A tooltip for this button provides error details:

while parsing a block mapping
in 'string', line 5, column 7:
 - stage: "Stage1"
 -
expected <block end>, but found '-'
in 'string', line 8, column 7:
 - stage: "Stage1"
 -

Figure 2.25: Convert to pipeline Declarative script – Validate pipeline

For example, only specific blocks are allowed in the stage block:

The screenshot shows the Jenkins PipelineAsYAML plugin interface. On the left, there's a sidebar with links: Global Variables Reference, Online Documentation, Examples Reference, IntelliJ IDEA GDML, and Pipeline As YAML Converter. The main area has two tabs: "Pipeline As YAML" and "Pipeline Declarative Script".

Pipeline As YAML:

```
agent:  
  none:  
stages:  
  - stage: "Stage1"  
    steps:  
      - echo "Hello Universe!"  
stages:  
  - stage: "Stage1.1"  
    steps:  
      - echo "Hello World!"
```

Pipeline Declarative Script:

```
pipeline {  
  agent none  
  stages {  
    stage('Stage1') {  
      stages {  
        stage('Stage1.1') {  
          steps {  
            echo "Hello World!"  
          }  
        }  
      }  
    }  
  }  
}
```

Below the tabs are two buttons: "Convert To Pipeline Declarative Script" and "Validate Pipeline". A status message at the bottom says "startup failed: WorkflowScript: 4: Only one of 'matrix', 'parallel', 'stages', or 'steps' allowed for stage 'Stage1' @ line 4, column 5. stage('Stage1')".

Figure 2.26: Convert to pipeline Declarative script – Validate pipeline

Fix the issues and validate the pipeline again:

Bingo! It is a valid YAML pipeline now as pipeline conversion is successful:

The screenshot shows the Jenkins PipelineAsYAML converter interface. On the left, there's a sidebar with links: Steps Reference, Global Variables Reference, Online Documentation, Examples Reference, IntelliJ IDEA GDScript, and Pipeline As YAML Converter. The main area has two tabs: "Pipeline As YAML" and "Pipeline Declarative Script".

Pipeline As YAML:

```
pipeline:
  agent:
    none:
  stages:
    - stage: "Stage1"
      stages:
        - stage: "Stage1.1"
          steps:
            - echo "Hello World in the Universe!"
```

Pipeline Declarative Script:

```
pipeline {
  agent none
  stages {
    stage('Stage1') {
      stages {
        stage('Stage1.1') {
          steps {
            echo "Hello World in the Universe!"
          }
        }
      }
    }
}
```

Below the tabs are "Convert To Pipeline Declarative Script" and "Validate Pipeline" buttons. A status message "Valid" is displayed at the bottom.

Figure 2.27: Convert to pipeline Declarative script – Valid pipeline

In the Pipeline job, go to Pipeline Steps to find out how steps have been executed and how much time specific stage and steps took to execute.

The screenshot shows the Jenkins Pipeline Steps page for the "Hello-World" pipeline. The left sidebar includes links: Back to Project, Status, Changes, Console Output, Edit Build Information, Delete build #5, Pipeline Configuration History, Open Blue Ocean, Build timeline, Restart from Stage, Replay, Pipeline Steps (which is selected), Workspaces, and Previous Build.

The main content area displays a table of pipeline steps:

Step	Arguments	Status
Start of Pipeline - (2.3 sec in block)		Success
Stage : Start - (0.00 sec in block)	Stage1	Success
Stage1 - (0.75 sec in block)	Stage1	Success
Stage : Start - (0.42 sec in block)	Stage1.1	Success
Stage1.1 - (0.26 sec in block)	Stage1.1	Success
Echo Message - (79 ms in self)	Hello World in the Universe!	Success

At the bottom right, it says "Page generated: Oct 18, 2020 12:47:18 AM IST Jenkins 2.235.4".

Figure 2.28: Pipeline Steps

In Declarative pipeline version 1.2, we can define stages to run in parallel while in Declarative pipeline version 1.3, we can specify stages nested within other stages as part of the Declarative syntax. The same can be utilized in YAML pipeline:

pipeline:

agent:

none:

stages:

- stage: "Stage1"

stages:

- stage: "Stage1.1"

steps:

- echo "Hello World in the Universe!"

Parallel block helps us to execute steps/stages in parallel:

pipeline:

agent:

none:

stages:

- stage: "Stage1"

stages:

- stage: "Stage1.1"

parallel:

- stage: "Stage1.1.1"

steps:

- echo "Hello India!"

- stage: "Stage1.1.2"

steps:

- echo "Hello USA!"

- stage: "Stage1.2"

agent:

label: 'master'

steps:

- echo "Hello World in the Universe!"

Tools definitions can be used under pipeline. It helps you to define tools that can be utilized for pipeline execution:

pipeline:

agent:

```
label: 'master'
```

```
tools:
```

```
gradle: "gradle-5.4.1-all"
```

```
jdk: "JDK8"
```

Tools definitions can also be used under stage:

```
pipeline:
```

```
agent:
```

```
any:
```

```
stages:
```

```
- stage: 'Build Docker Image'
```

```
agent:
```

```
label: 'docker'
```

tools:

```
git: 'CentOSGIT'
```

Triggers and environments help to configure trigger to execute pipeline and declare environment variables, respectively:

pipeline:

agent:

any:

stages:

- stage: 'Static Code Analysis'

agent:

label: 'master'

triggers:

- pollSCM('o o 6 * * ?')

environment:

ANDROID_HOME: 'C:\\Program Files
(x86)\\Android\\android-sdk'

JAVA_HOME: 'C:\\Program Files\\Java\\jdk1.8.0_111'

Following are some important tips while taking Backup:

Do not upgrade any incompatible Plugin. It might break the setup.

In production support, upgrade plugins one by one and verify before proceeding ahead.

Take backup of JENKINS_HOME manually as well at a certain interval.

Server snapshot can be helpful if there is no point of return.

Simulate failure and practice Backup and Restore process multiple times.

In YAML pipeline, we can define Agents for pipeline execution in different ways such as at pipeline level or stage level:

pipeline:

agent:

any:

stages:

- stage: 'Static Code Analysis'

agent:

label: 'master'

- stage: 'Build Docker Image'

agent:

label: 'docker'

How to specify the agent in the Declarative pipeline?

Run pipeline on any agent:

pipeline:

agent:

any:

Run pipeline on the agent that is matching the label:

pipeline:

agent:

node:

label: 'master'

Run Agent inside a docker container:

pipeline:

agent:

docker:

```
image: 'nginx:1-alpine'
```

Build a **Dockerfile** and run in a container using the same image:

pipeline:

agent:

dockerfile:

```
filename: 'Dockerfile'
```

Do not run on an agent:

pipeline:

agent:

none:

Parameters help you provide values at the time of pipeline execution rather than hard-coding values in the YAML pipeline:

pipeline:

agent:

label: 'master'

stages:

- stage: "Stage1"

stages:

- stage: "Stage1.1"

parallel:

- stage: "Stage1.1.1"

steps:

- bat "echo Hello %country1% "

- stage: "Stage1.1.2"

steps:

- bat "echo Hello %country2% "

- stage: "Stage1.2"

agent:

label: 'master'

steps:

- echo "Hello World in the Universe!"

parameters:

- "string(name: 'country1', defaultValue: 'country-1', description: 'Description')"

```
- "string(name: 'country2', defaultValue: 'country-2',
description: 'Description')"
```

The post section provides one or more additional steps that need to be executed for cleanup or notifications or other activities after the pipeline's or stage's execution:

execution: execution: execution: execution: execution: execution: execution: execution: execution: execution: execution: execution: execution: execution: execution: execution: execution: execution: execution: execution:
--

execution: execution: execution: execution: execution: execution: execution: execution: execution: execution:

execution: execution: execution: execution: execution: execution: execution: execution: execution: execution: execution: execution: execution: execution:

execution: execution:

```
execution: execution: execution: execution: execution: execution:  
execution: execution: execution: execution: execution: execution:  
execution: execution: execution: execution: execution: execution:  
execution: execution: execution: execution: execution:
```

Table 2.3: post section

Syntax of post section in YAML pipeline:

pipeline:

agent:

dockerfile:

filename: 'Dockerfile'

stages:

- stage: "SCA"

steps:

- echo "Static Code Analysis"
- stage: "Build"

steps:

- echo "Build Execution"

post:

always:

- echo "Include One or more steps within each block."

unstable:

- echo " Include One or more steps within each block."

notBuilt:

- echo " Include One or more steps within each block."

cleanup:

- echo " Include One or more steps within each block."

regression:

- echo " Include One or more steps within each block."

aborted:

- echo " Include One or more steps within each block."

success:

- echo " Include One or more steps within each block."

failure:

- echo " Include One or more steps within each block."

unsuccessful:

- echo " Include One or more steps within each block."

fixed:

- echo " Include One or more steps within each block."

changed:

- echo " Include One or more steps within each block."

Archives artifacts with each build using step: **archiveArtifacts**
artifacts: '/*.war', onlyIfSuccessful:** How to archive artifacts when the pipeline fails?

pipeline:

agent:

none:

stages:

- stage: "SCA"

steps:

- echo "Static Code Analysis"

- stage: "Unit Tests"

steps:

- echo "Unit Tests Execution"

- stage: "Build"

steps:

- echo "Build Execution"

post:

always:

- "archiveArtifacts '**/*.apk'"

The options directive can be used to specify properties for settings that should apply across the pipeline. Following are some of the options we can add to the pipeline:

The skipDefaultCheckout option disables the standard, automatic checkout SCM before the first stage. If specified and SCM checkout is desired, it will need to be explicitly included.

Enable project-based security.

This determines when, if ever, build records for this project should be discarded. Build records include the console output, archived artifacts, and any other metadata related to a particular build.

If specified, any stage after the build has become UNSTABLE will be skipped, as if the build had failed.

Executes the code inside the block with a determined time out limit.

Add timestamps to the Console Output

Following is a Declarative script block for options we discussed above:

pipeline:

agent:

dockerfile:

```
filename: 'Dockerfile'
```

```
stages:
```

- stage: "SCA"

```
steps:
```

- echo "Static Code Analysis"

- stage: "Unit Tests"

```
steps:
```

- echo "Unit Tests Execution"

- stage: "Build"

```
steps:
```

- echo "Build Execution"

```
post:
```

always:

- "archiveArtifacts '**/*.apk'"

options:

- skipDefaultCheckout true
- timeout(10)
- timestamps
- skipStagesAfterUnstable()
- "buildDiscarderlogRotator(artifactDaysToKeepStr: "", artifactNumToKeepStr: "", daysToKeepStr: "", numToKeepStr: '10')"
- "authorizationMatrix(['hudson.model.Item.Build:mitesh', 'hudson.model.Item.Cancel:mitesh', 'hudson.model.Item.Configure:mitesh', 'hudson.model.Item.Delete:mitesh', 'hudson.model.Item.Discover:mitesh', 'hudson.model.Item.Move:mitesh',

```
'hudson.model.Item.Read:mitesh',
'hudson.model.Item.Workspace:mitesh',
'hudson.model.Run.Delete:mitesh',
'hudson.model.Run.Replay:mitesh',
'hudson.model.Run.Update:mitesh',
'hudson.scm.SCM.Tag:mitesh'])"
```

Double Quotes for entire step is required if converter validation fails for specific steps such as:

- "cobertura(coberturaReportFile: '/coverage/cobertura-coverage.xml', sourceEncoding: 'ASCII')"
- "archiveArtifacts
'platforms/android/app/build/outputs/apk/debug/app-debug.apk'"
- "appCenter(apiToken:
'75794011ded8da608c433437107efd53d45e9028', ownerName:
'xxx.soni-xxxxxxxx.com', appName: 'SampleIonicCordovaApp',
pathToApp: 'platforms/android/app/build/outputs/apk/debug/app-
debug.apk', distributionGroups: 'Dev-Distribution', releaseNotes:
'Bug Fixed - Ticket 2020.05.30')"
- "publishCoverage(adapters: [coberturaAdapter('coverage.xml')],
sourceFileResolver: sourceFiles('NEVER_STORE'))"
- " catchError(buildResult: 'SUCCESS', stageResult: 'FAILURE') {

```
        bat 'flutter analyze'  
  
    }"
```

Configure Security in Jenkins. Go to Manage Jenkins|Security.

Configure Global Security.

Configure Authentication using Azure Active Directory or Jenkins' user database or LDAP.

Configure Authorization Strategy using Azure Active Directory Matrix-based security or Matrix-based security or Project-based Matrix Authorization Strategy.

Enable Agent - Master Access Control.

Credentials: Configure Credentials.

Users: Manage Users.

Conclusion

In this chapter, we have covered how to create Agents, create agent pools and utilize Agents to execute build jobs. We have also covered how to create sample YAML script using parallel stage and nested stages, using Pipeline as a YAML converter, tools definitions, triggers and environments, agents, parameters, post section, archives artifacts, and options.

In the next chapter, we will cover how to build CI/CD pipeline with YAML script for flutter application.

Multiple choice questions

Authorization strategy include:

Matrix-based security

Project-based Matrix Authorization Strategy

All of the above

Following are allowed in post section of YAML pipeline:

Always

Fixed

Failure

All of the above

Answer

c

d

Questions

Explain the various options available for YAML pipeline.

Explain the various post actions available for YAML pipeline.

CHAPTER 3

Building CI/CD Pipeline with YAML for Flutter Application

"Believe in yourself! Have faith in your abilities! Without a humble but reasonable confidence in your own powers you cannot be successful or happy."

—*Norman Vincent Peale*

An multinational software organization makes applications for the edutainment and media industries. The marketing team decides to promote mobile apps for end-users and principle architects suggest hybrid apps rather than Native apps. Project teams decide to use Flutter for edutainment and media industries related apps. Flutter is a UI toolkit for creating beautiful, natively compiled applications for mobile, web, and desktop from a single codebase for faster time to market. Management wants to automate manual processes in application lifecycle management and to increase productivity with frequent deployments implementing continuous integration and continuous delivery. In this chapter, we will cover the CI/CD implementation of Flutter applications with Jenkins. We will use Pipeline as a YAML to create CI/CD pipeline. We will distribute an application to the App Center to a specific Group. We will

also provide some valuable " *notes* " related to DevOps, culture, challenges, market trends, and so on for better understanding.

Structure

In this chapter, we will discuss the following topics:

DevOps tools, expected deliverables, or features

Multi-stage YAML pipeline for Flutter app

Continuous integration for Flutter – Android app

Continuous delivery for Flutter – Android app

Objectives

This chapter will introduce how to implement continuous integration and continuous delivery for Flutter apps using Pipeline as a YAML. After studying this unit, you should be able to:

Understand how to perform static code analysis for the Flutter application

Execute unit tests

Calculate code coverage

Verify build quality

DEVOPS TOOLS, EXPECTED DELIVERABLES, OR FEATURES

The organization has selected one application as a pilot while there are other five early adopter Flutter applications waiting in the queue. We have a responsibility to create CI/CD pipeline

using Pipeline as a YAML in Jenkins. Following is the list of tools, and deliverables that will be integrated into the pipeline:

pipeline:

pipeline:

pipeline: pipeline:

pipeline: pipeline: pipeline: pipeline:

pipeline:

pipeline: pipeline:

pipeline:

pipeline: pipeline:

pipeline: pipeline: pipeline: pipeline: pipeline:

pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline:

pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline:

pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline:

pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline:

pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline:

pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline:

pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline:

pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline:

Jenkins plugins:

plugins:

plugins: plugins: plugins: plugins: plugins: plugins:
plugins: plugins: plugins: plugins: plugins: plugins:

plugins: plugins: plugins: plugins: plugins: plugins: plugins:
plugins: plugins: plugins: plugins: plugins:

plugins: plugins: plugins: plugins: plugins: plugins: plugins:
plugins: plugins: plugins:

plugins: plugins: plugins: plugins: plugins: plugins: plugins:

plugins: plugins: plugins: plugins: plugins: plugins: plugins: plugins: plugins: plugins: plugins: plugins:
--

| plugins: plugins: plugins: plugins: plugins: plugins: |
| plugins: plugins: plugins: plugins: plugins: plugins: plugins: plugins: |
| plugins: plugins: plugins: plugins: plugins: plugins: plugins: |

plugins: plugins: plugins: plugins: plugins: plugins: plugins: plugins: plugins: plugins: plugins: plugins: plugins:
--

| plugins: plugins: plugins: plugins: plugins: plugins: plugins: |

Table 3.1: Tools and deliverables

In the next section, we will create Pipeline as a YAML for sample application in a step-by-step manner.

Multi-stage CI/CD pipeline for Flutter app

In this chapter, we will cover CI/CD for sample Flutter application:

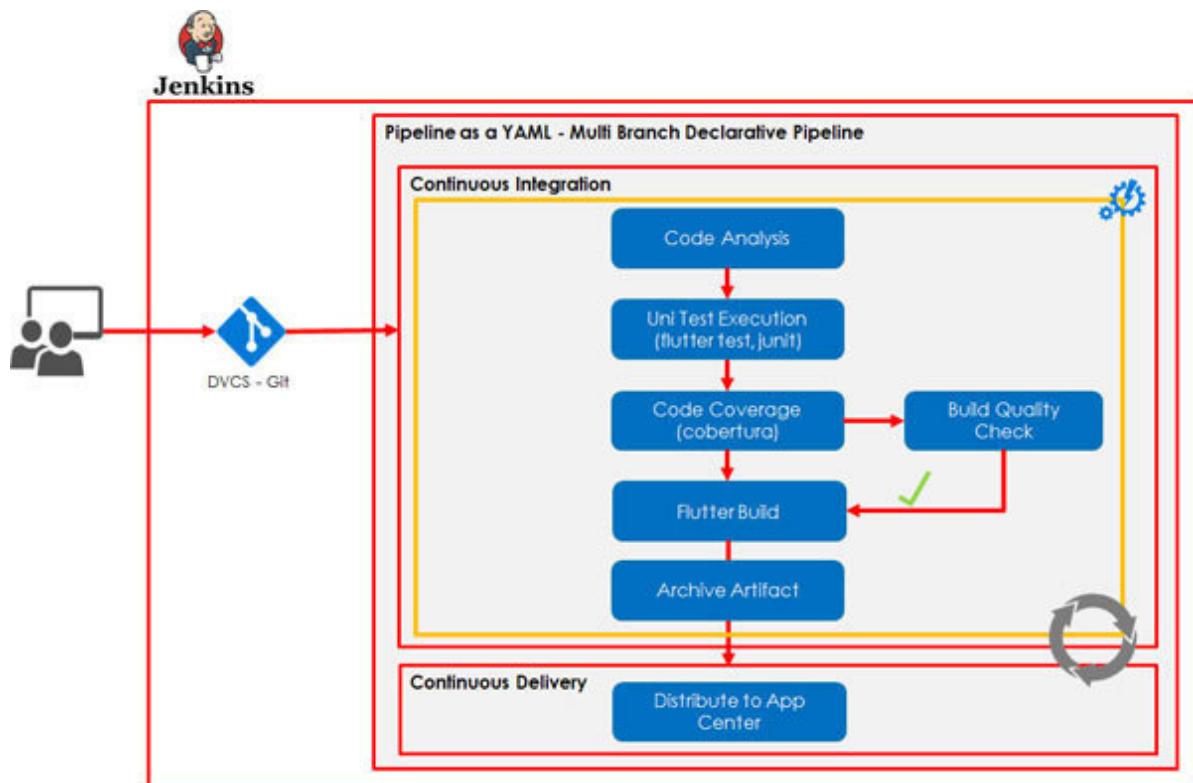


Figure 3.1: Big picture for CI/CD of Flutter App

Before going further, let us discuss the installation of Flutter on the Windows operating system:

Download the installation bundle to get the latest stable release of the Flutter SDK from

Extract the ZIP file.

Put the directory into a directory that doesn't require high privileges.

Click on the search bar, enter env, or edit and select **Edit Environment Variables** for your account.

Configure `\.pub-cache\bin` and `\bin\cache\dart-sdk\bin` as well:

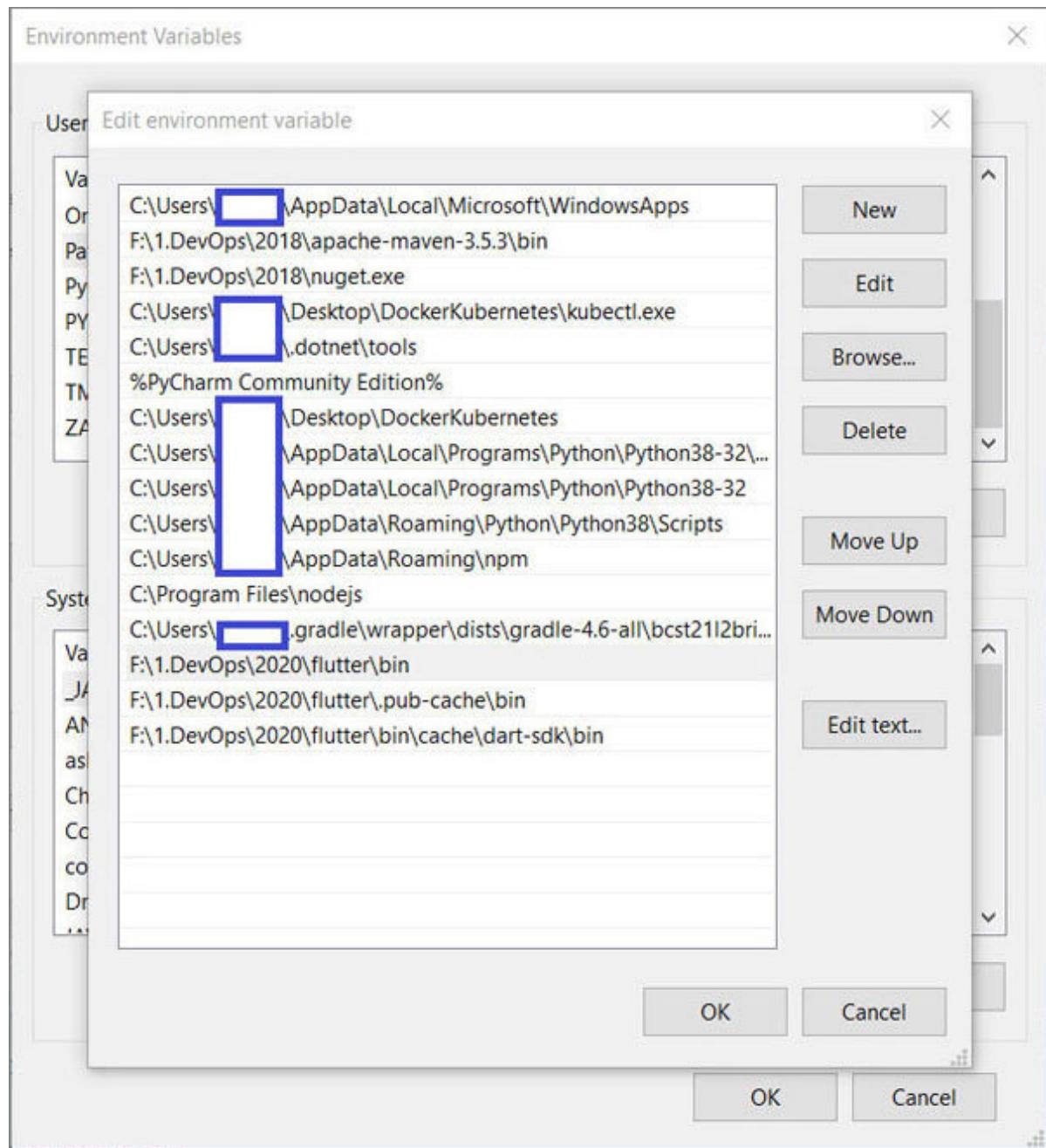


Figure 3.2: Environment variables

For more details on Flutter installation on Windows, visit

Let us a create pipeline for a sample Flutter application using YAML.

We will configure static code analysis using flutter analyze, unit test execution, and code coverage calculation and report for sample Flutter application.

Continuous integration for Flutter app

Let us configure continuous integration. Create a Jenkins pipeline job from the Jenkins Dashboard and click on configure to set Flutter application's repository:

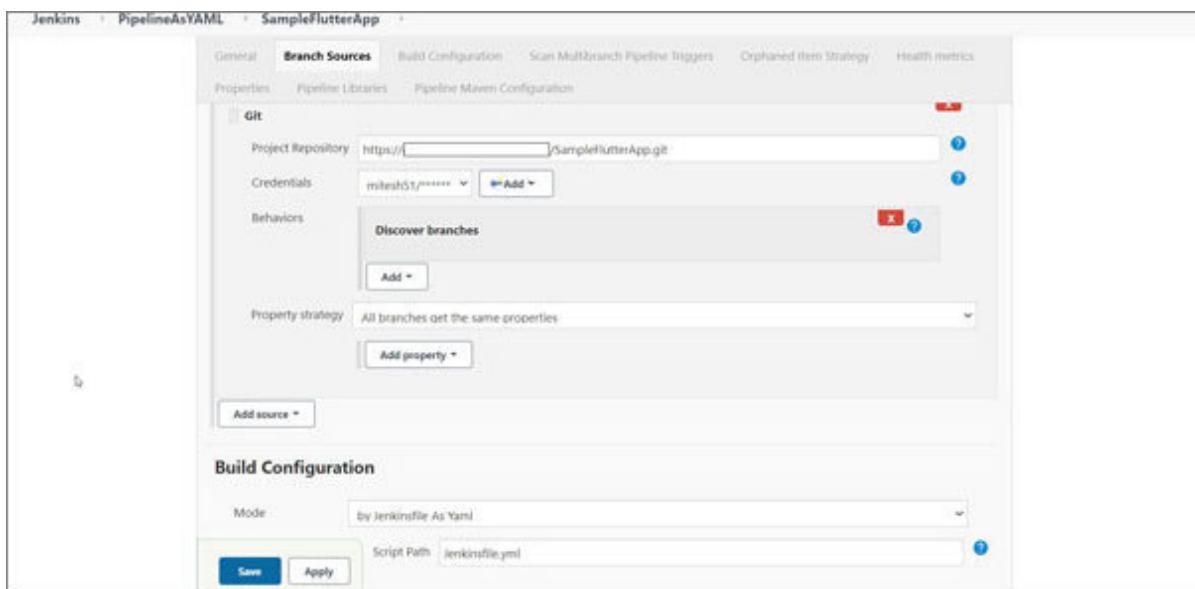
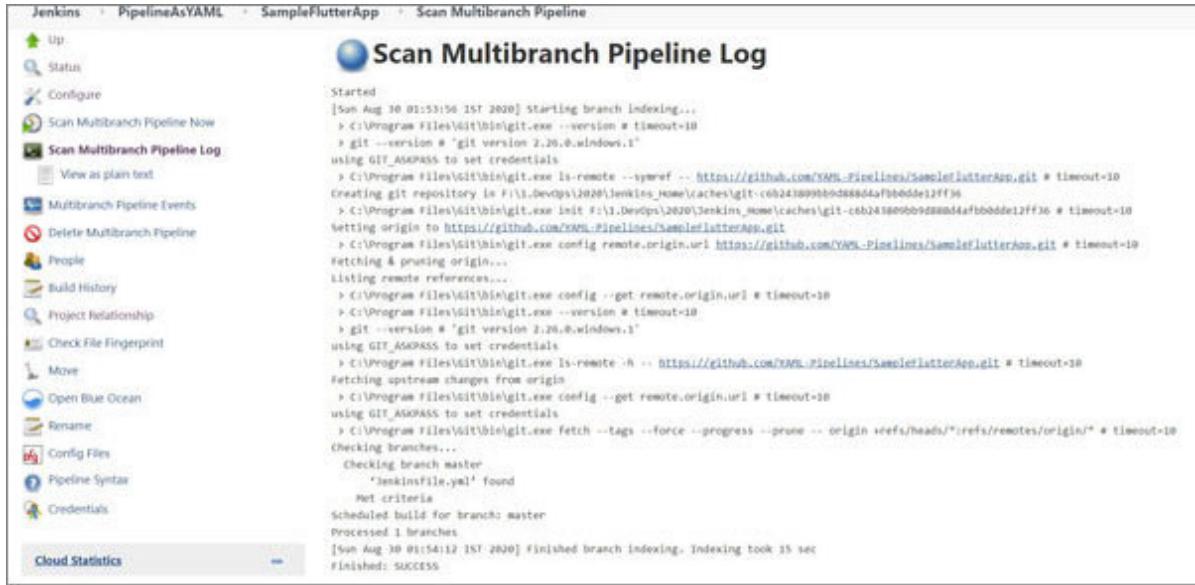


Figure 3.3: Branch sources

Freestyle project is very important in Jenkins as it can be used in orchestration with command execution such as batch or shell commands.

Verify multi-branch pipeline log. There is one branch that has Jenkinsfile:



The screenshot shows the Jenkins interface for a 'Scan Multibranch Pipeline' job. On the left, there's a sidebar with various Jenkins management links like 'Up', 'Status', 'Configure', etc. The main content area is titled 'Scan Multibranch Pipeline Log'. It displays a command-line log of the indexing process. The log starts with '[Sun Aug 30 01:53:56 IST 2020] starting branch indexing...', followed by several git commands for cloning and setting up remotes. It then moves on to fetching upstream changes and checking branches. Finally, it concludes with '[Sun Aug 30 01:54:12 IST 2020] finished branch indexing. Indexing took 15 sec' and 'Finished: SUCCESS'.

```
Started
[Sun Aug 30 01:53:56 IST 2020] starting branch indexing...
> C:\Program Files\Git\bin\git.exe --version # timeout=10
> git --version # "git version 2.26.0.windows.1"
using GIT_ASKPASS to set credentials
> C:\Program Files\Git\bin\git.exe ls-remote --verbose -- https://github.com/YAML-Pipelines/SampleFlutterApp.git # timeout=10
Creating git repository in F:\Jenkins\jenkins_home\caches\git\cd243099bb0d888d4af00ddde12ff36
> C:\Program Files\Git\bin\git.exe init #> F:\Jenkins\jenkins_home\caches\git\cd243099bb0d888d4af00ddde12ff36 # timeout=10
Setting origin to https://github.com/YAML-Pipelines/SampleFlutterApp.git
> C:\Program Files\Git\bin\git.exe config remote.origin.url https://github.com/YAML-Pipelines/SampleFlutterApp.git # timeout=10
Fetching & pruning origin...
Listing remote references...
> C:\Program Files\Git\bin\git.exe config --get remote.origin.url # timeout=10
> C:\Program Files\Git\bin\git.exe --version # timeout=10
> git --version # "git version 2.26.0.windows.1"
using GIT_ASKPASS to set credentials
> C:\Program Files\Git\bin\git.exe ls-remote :< https://github.com/YAML-Pipelines/SampleFlutterApp.git # timeout=10
Fetching upstream changes from origin
> C:\Program Files\Git\bin\git.exe config --get remote.origin.url # timeout=10
using GIT_ASKPASS to set credentials
> C:\Program Files\Git\bin\git.exe fetch --tags --force --progress --prune -- origin refs/heads/*:refs/remotes/origin/* # timeout=10
Checking branches...
  Checking branch master
    'Jenkinsfile.yml' found
      Met criteria
Scheduled build for branch: master
Processed 1 branches
[Sun Aug 30 01:54:12 IST 2020] finished branch indexing. Indexing took 15 sec
Finished: SUCCESS
```

Figure 3.4: Multi-branch pipeline log

The pipeline template helps to orchestrate tasks that can be executed on multiple build agents.

Click on the status link to get list of branches available where Jenkinsfile is available or branches that are filtered based on wildcards:

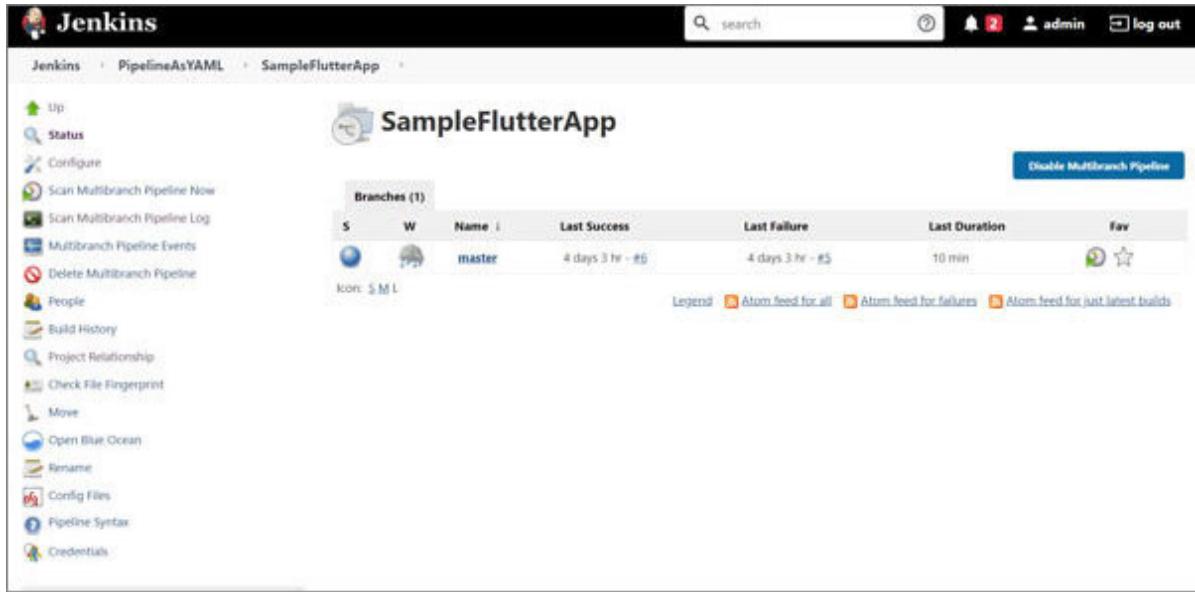


Figure 3.5: Branches in Jenkins Dashboard

Command flutter doctor verifies the environment and displays your Flutter installation status:

pipeline:

agent:

label: 'master'

stages:

- stage: 'Flutter Doctor'

steps:

- bat 'flutter doctor'

External Job template helps to record the execution of a process executed outside Jenkins.

Verify the logs that are stage-wise and steps added in each stage. It is easier to navigate through logs in Blue Ocean rather than using the traditional Jenkins console:



Figure 3.6: Flutter doctor

Command flutter analysis verifies code quality and standard. It fails the pipeline if it finds issues in the code. We are using the following block to continue the pipeline even after failure:

```
- stage: 'SCA'
```

```
steps:
```

```
- "catchError(buildResult: 'SUCCESS', stageResult:  
'FAILURE') {
```

```
    bat 'flutter analyze'
```

```
}
```

Verify SCA stage execution in Blue Ocean dashboard:



Figure 3.7: Flutter doctor – code analysis

Execute flutter pub get to add a package dependency to an app. To get details such as a number of tests executed, and which tests are passed or failed execute flutter test machine:

- stage: 'Test & Coverage'

steps:

- bat 'flutter pub get && flutter pub global activate junitreport && flutter test --machine | tojunit --output test.xml'

```
- junit 'test.xml'

- bat 'flutter test --coverage && python
C:\\Python38\\Lib\\site-packages\\lcov_cobertura.py
coverage\\lcov.info'

- "publishCoverage(adapters:
[coberturaAdapter('coverage.xml')], sourceFileResolver:
sourceFiles('NEVER_STORE'))"
```

JUnit Report is a Dart package that allows converting the JSON output of the machine argument to produce a JUnit test report. It helps to publish a report in Jenkins. To use JUnit Report (Dart package), add Dart DK and other packages available by adding them in the path.

lcov to Cobertura XML converter converts code coverage report files from lcov to Cobertura's XML report. It helps to publish the report in Jenkins.

Install lcov_cobertura in Windows:

```
C:\\Users\\Tempuser>pip install lcov_cobertura
Collecting lcov_cobertura
  Downloading lcov_cobertura-1.6.tar.gz (6.3 kB)
```

Using legacy setup.py install for lcov-Cobertura, since package 'wheel' is not installed.

Installing collected packages: lcov-Cobertura

Running setup.py install for lcov-cobertura ... done

Successfully installed lcov-cobertura-1.6

Execute the pipeline and verify the Tests section for unit test report:

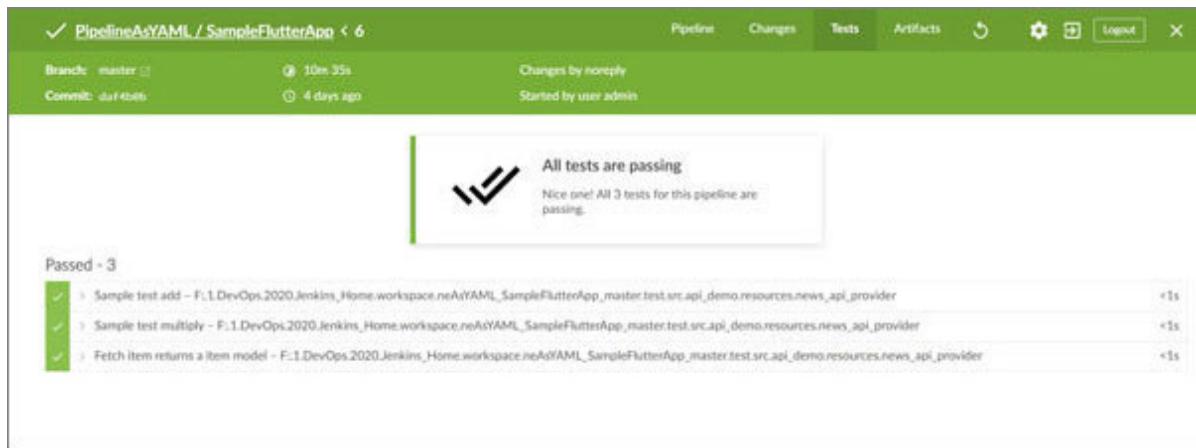


Figure 3.8: Unit tests results

Multi-configuration project template helps to orchestrate projects that need a large number of different configurations.

The Test Results Analyzer plugin shows the build result history of test-class, test-class, and test-package in a tabular tree format. The plugin can be used enabling the **Publish junit**

results or Publish TestNG results (in case of TestNG) feature of Jenkins:

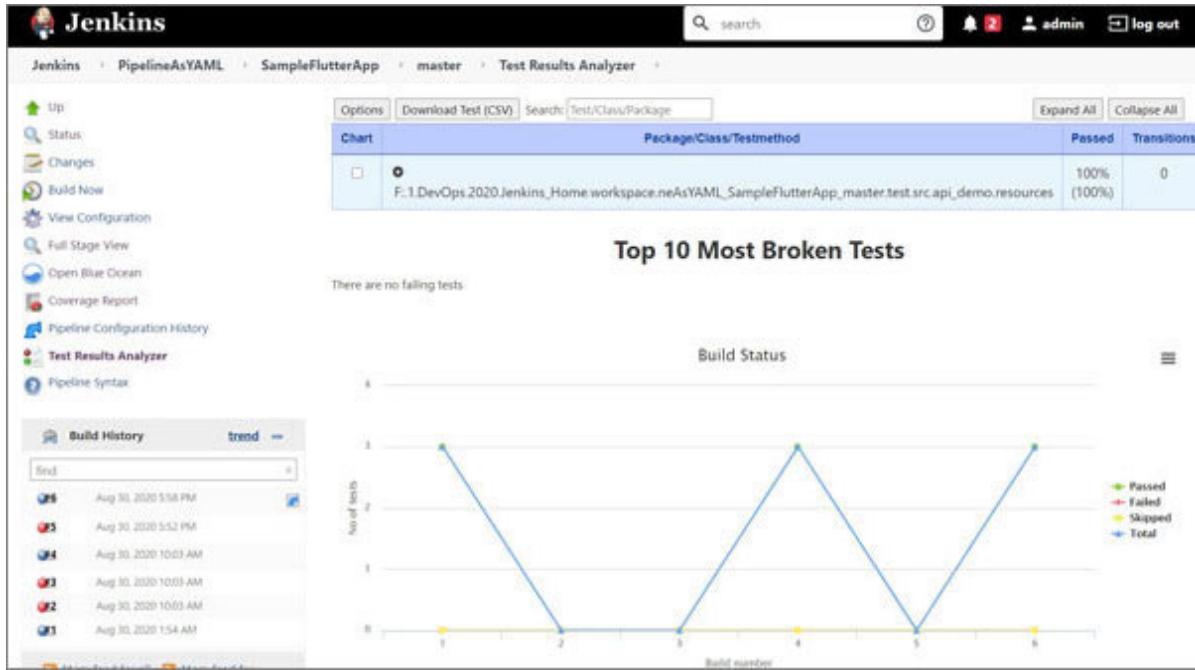


Figure 3.9: Test Results Analyzer

The Test Results Analyzer plugin allows users to filter the results based on passed, failed, and skipped status. We can click on the **Test Results Analyzer** link on the left side of the pipeline job. It also supports the generation of graphs such as line charts, pie charts, and bar charts:



Figure 3.10: Test Results Analyzer – Chart

Bitbucket Team/Project scans a Bitbucket Cloud Team (or Bitbucket Server Project) for all repositories matching some defined markers.

Code Coverage API serves as an API to integrate and publish multiple coverage report types such as JaCoCo, Cobertura (Cobertura Plugin), and OpenCover. It also supports pipeline configuration, you can generate pipeline code in Jenkins Snippet Generator.

Verify the Coverage report in the Jenkins dashboard for the pipeline job:

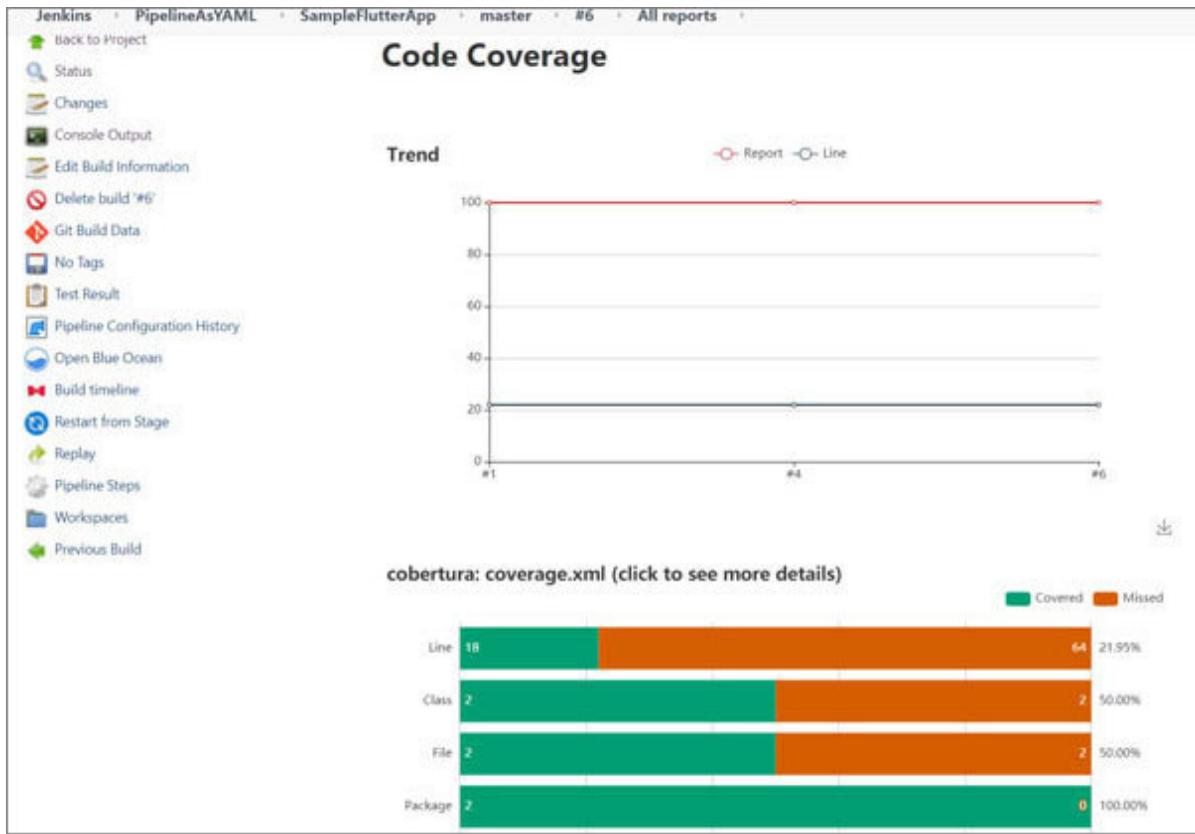


Figure 3.11: Code Coverage Report

Following is the script to create build for flutter application and archive artifacts:

- stage: 'Build'

steps:

- bat 'flutter build apk --debug'

- archiveArtifacts

Verify the Build stage where the APK file is created:

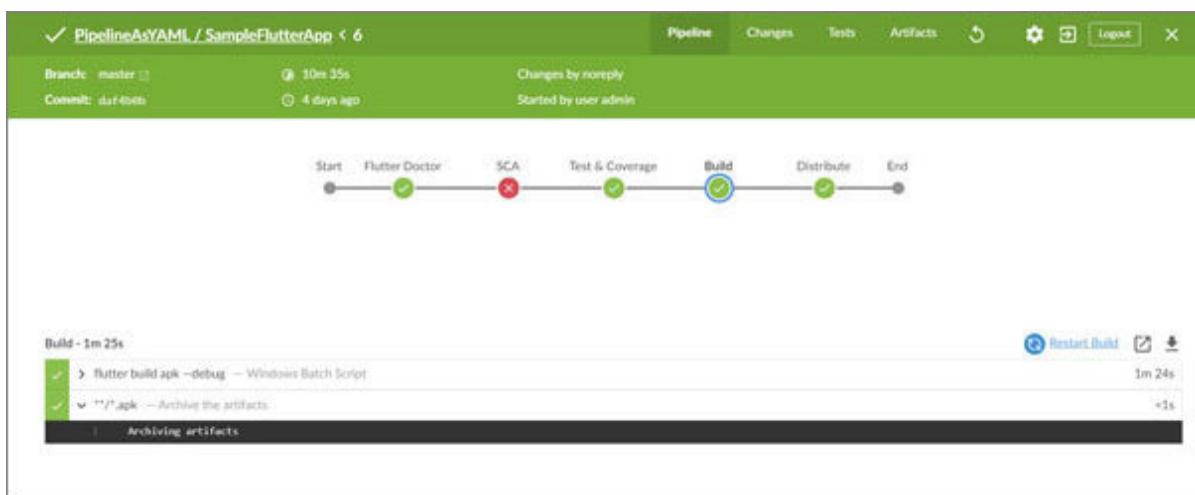


Figure 3.12: Build Logs

Verify the APK file in the Artifacts section:

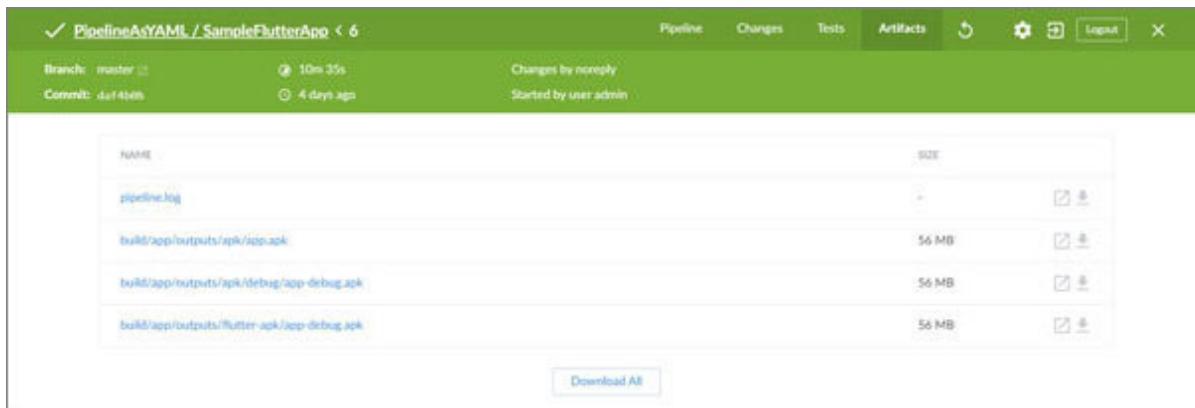


Figure 3.13: Build Artifacts

A folder is like a container that can be useful to store nested items in it. It is more like grouping things.

Go to the traditional Jenkins dashboard and verify the status of pipeline execution:

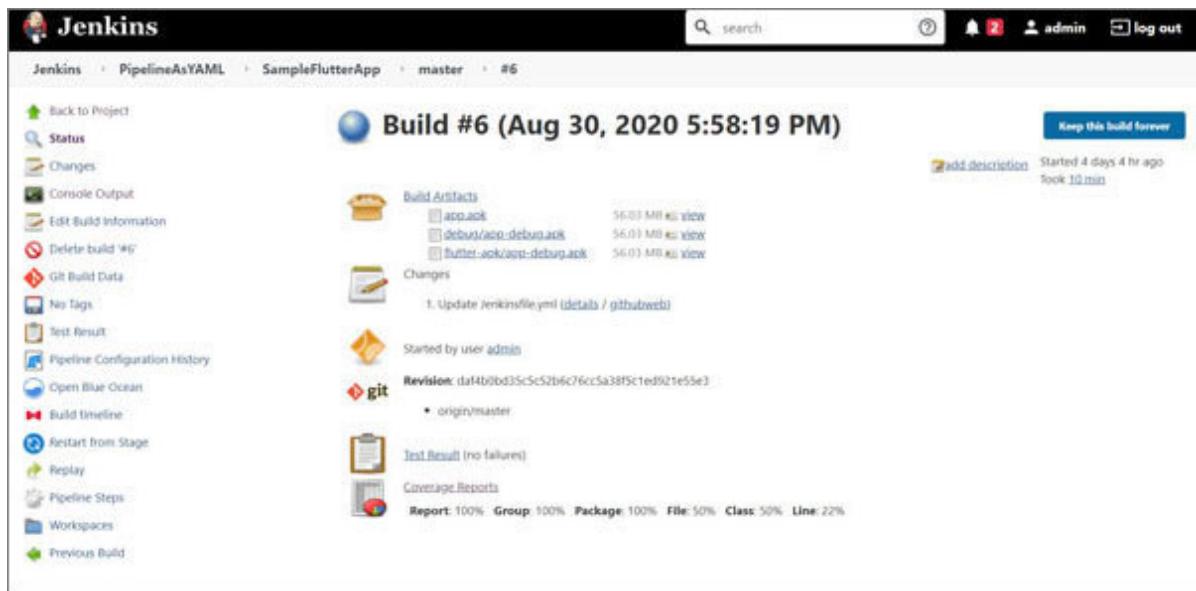


Figure 3.14: Pipeline Execution Status

In the next section, we will configure Continuous Delivery for the Sample Flutter application.

Continuous delivery for Flutter app

App Center is used to distribute mobile applications to the QA team. It is also used to build, test, and distributed Android, iOS, and Windows applications. Go to Log in with the appropriate method. Provide Username for login.

Before we start the delivery step, we need the below details from the App Center.

App center login token (Can be first created from the terminal and reused from Jenkins shell script by using command app center login as sudo user. Note-Copy the token generated for future use)

App Name in App Center

Username and Password

Tester Group name to whom you need to share the app.

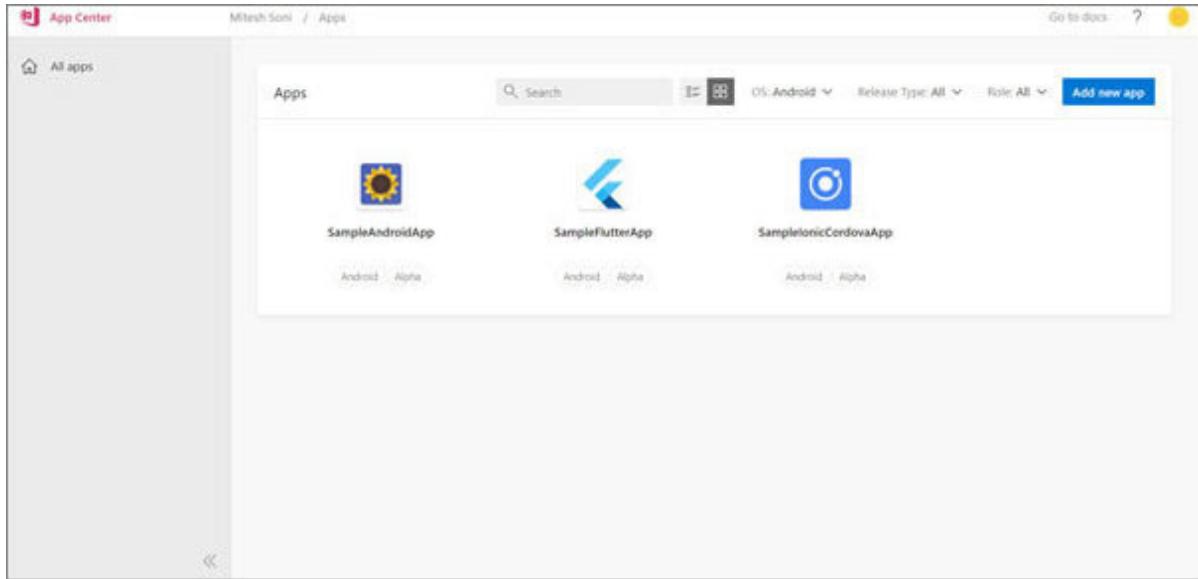


Figure 3.15: App center Dashboard

Multi-branch pipeline template creates a set of pipelines based on detected branches having Jenkinsfile in one repository.

Let us create app in App Center and configure stings.

Let us add a new App in App Center by clicking on **Add**

Select **App** and

Click on **Add new**

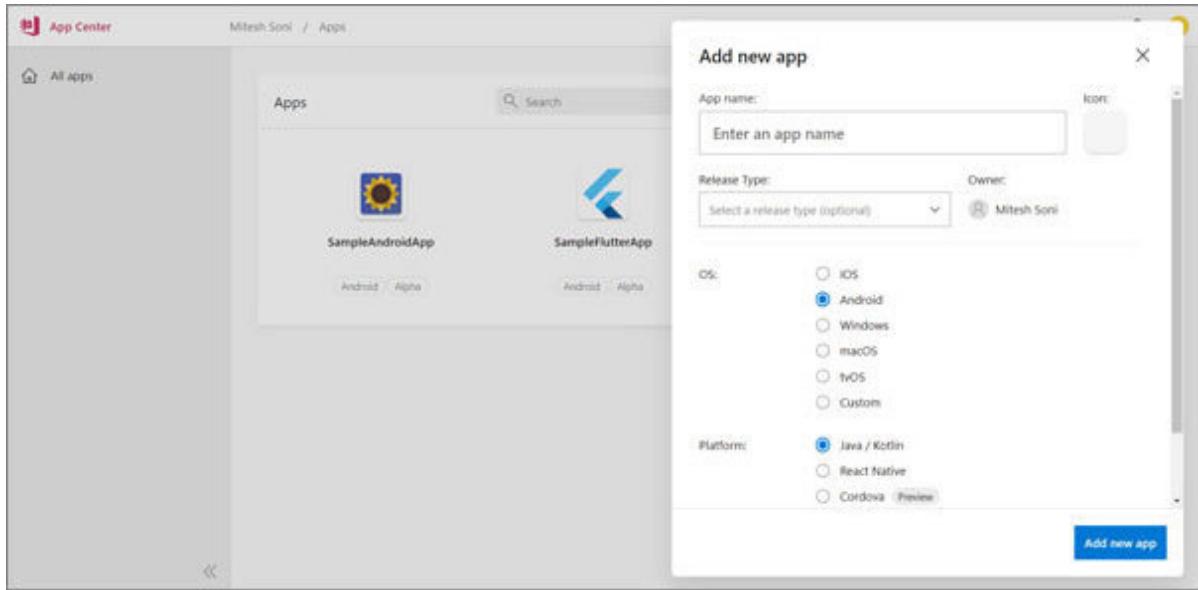


Figure 3.16: App Center – add application

To integrate App center in Jenkins, we will need API token. Let us create it first.

Let us go to **Account Settings** of App Center.

Scroll down in the **Account Settings** page.

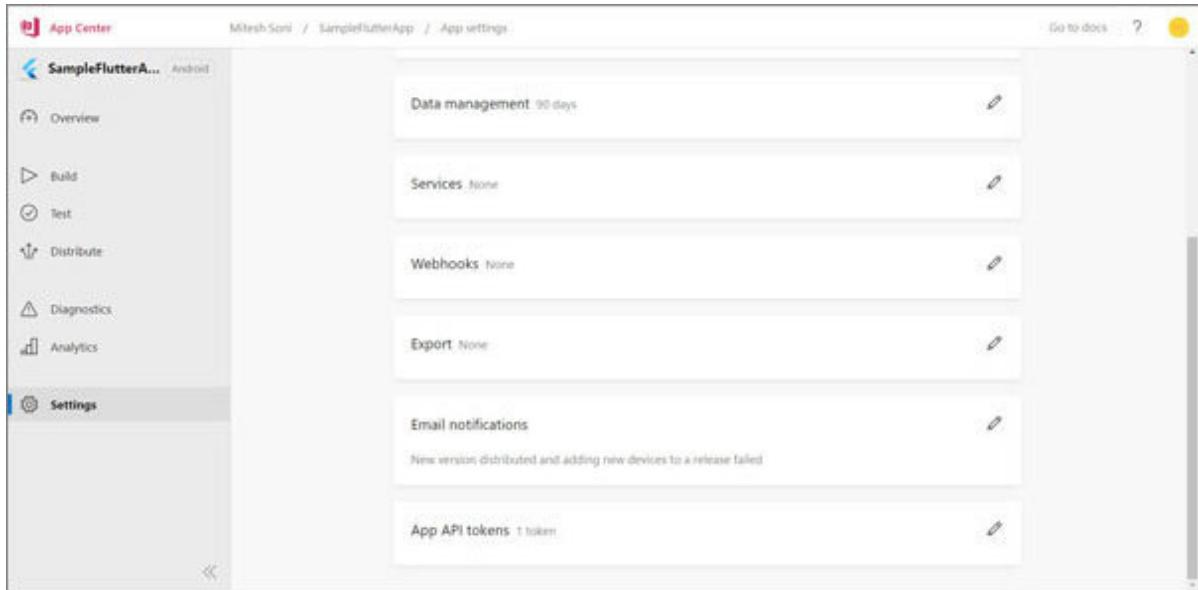


Figure 3.17: App Center – Settings

Click on the **New API**

Provide **Description** and

Click on **Add new API**

Copy the **API**

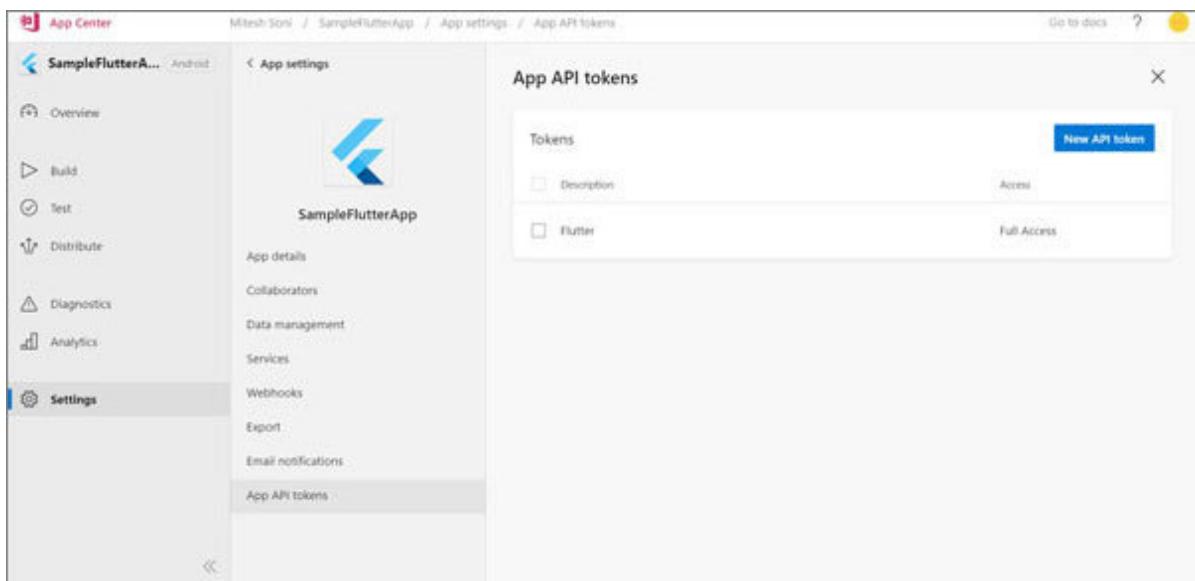


Figure 3.18: App Center – API Token

Following is the YAML script to distribute APK file to App center:

- stage: 'Distribute'

steps:

```
- "appCenter(apiToken: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
ownerName: 'xxxxxx-xxxxxx.com', appName: 'SampleFlutterApp',
pathToApp: 'build/app/outputs/apk/debug/app-debug.apk',
distributionGroups: 'Dev-Distribution', releaseNotes: 'Security Bug
Fixed - Ticket 2020.o8.20')"
```

Verify the Distribute stage logs:

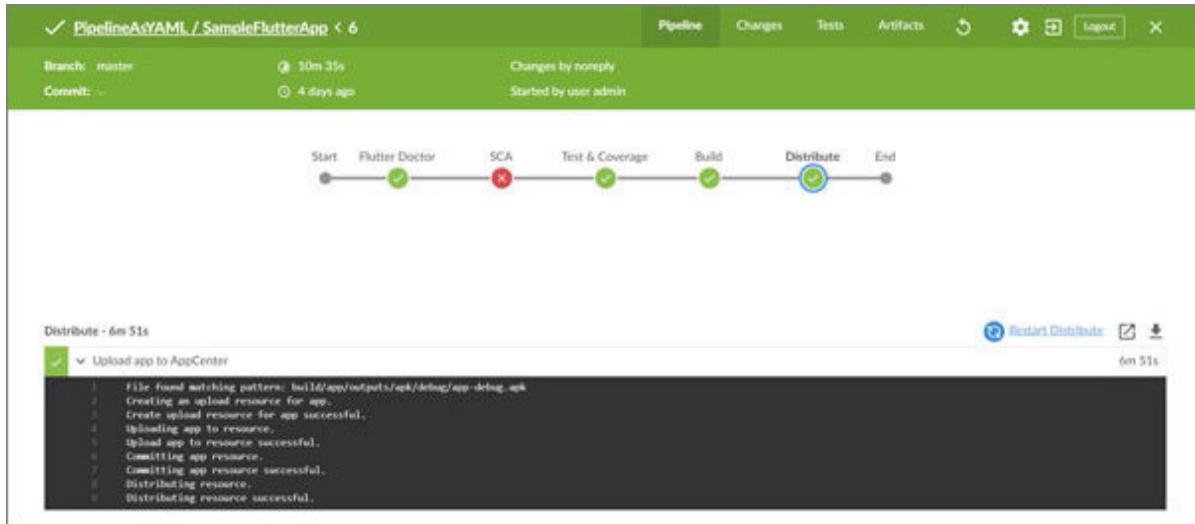


Figure 3.19: App Center – Distribution

Verify the artifacts in the Jenkins dashboard after pipeline is executed:

A screenshot of the Jenkins Pipeline master dashboard. The left sidebar includes links for Up, Status, Changes, Build Now, View Configuration, Full Stage View, Open Blue Ocean, Coverage Report, Pipeline Configuration History, Test Results Analyzer, and Pipeline Syntax. The main area shows the full project name: PipelineAsYAML/SampleFlutterApp/master. It displays the last successful artifacts: app.apk, debug/app-debug.apk, and flutter.apk/app-debug.apk, each 56.03 MB. A Test Result Trend chart shows a single blue bar for Passed. A Code Coverage chart shows the following data:

Level	Covered	Mixed	Total	Percentage
Line	18	64	82	21.95%
Class	2	2	4	50.00%
File	2	2	4	50.00%
Package	2	0	2	100.00%

A Build History table lists three recent builds: #85 (Aug 30, 2020 5:52 PM), #84 (Aug 30, 2020 10:01 AM), and #83 (Aug 30, 2020 10:01 AM).

Figure 3.20: Flutter pipeline status

Click on a **Full Stage View** in the left side bar to get the pipeline in a full view:

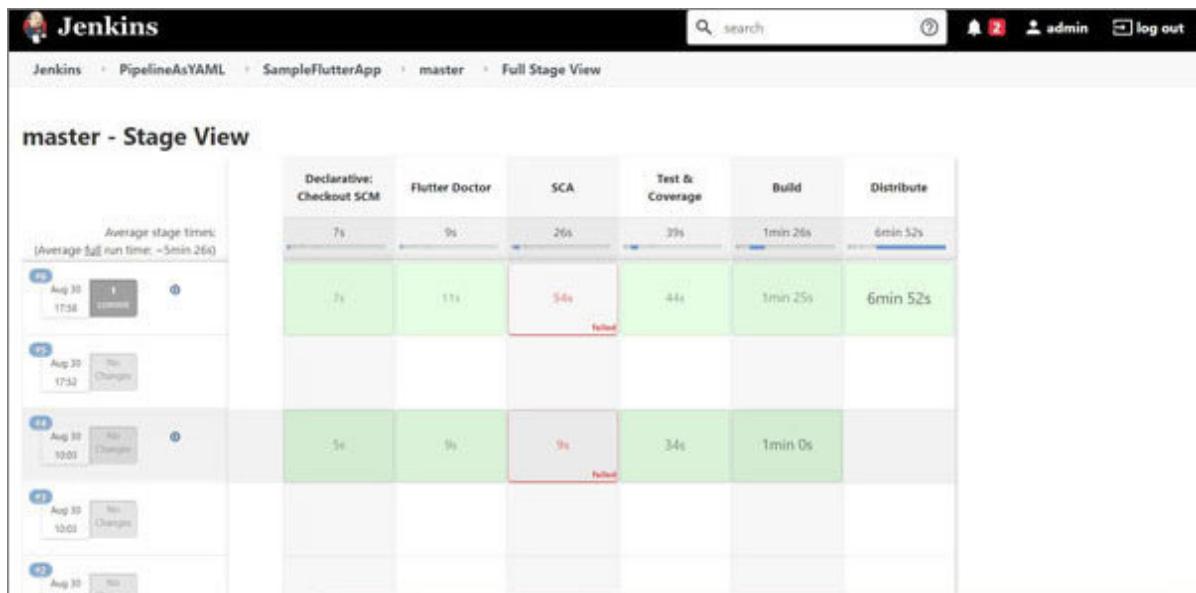


Figure 3.21: Full stage view

Go to **App Center** and verify the **Releases** section. The package will be available for download:

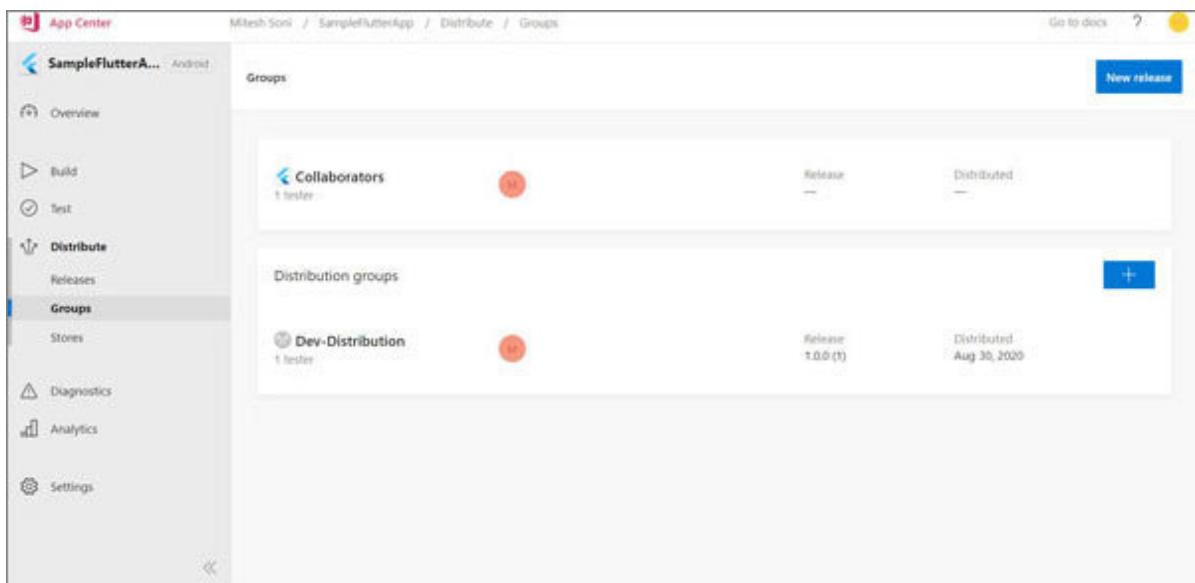


Figure 3.22: App Center – Distribution Group

Click on the **Groups** and select the **Dev-Distribution Testing** group that we created. App is available in that section too:

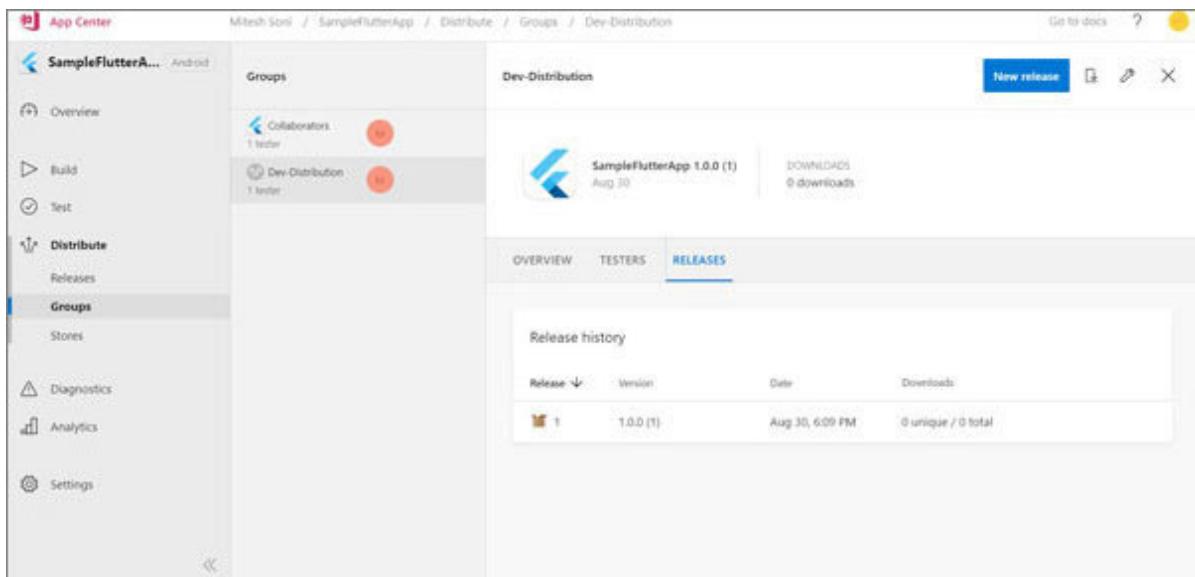


Figure 3.23: App Center – release history

In the next section, we will verify the entire YAML script covered in this chapter.

YAML pipeline script for Flutter app

Following is the complete YAML script to create CI/CD pipeline for sample Flutter App:

pipeline:

agent:

label: 'master'

stages:

- stage: 'Flutter Doctor'

steps:

- bat 'flutter doctor'

- stage: 'SCA'

steps:

- " catchError(buildResult: 'SUCCESS', stageResult: 'FAILURE') {

- bat 'flutter analyze'

- }"

- stage: 'Test & Coverage'

steps:

- bat 'flutter pub get && flutter pub global activate junitreport && flutter test --machine | tojunit --output test.xml'

- junit 'test.xml'

- bat 'flutter test --coverage && python C:\\Python38\\Lib\\site-packages\\lcov_cobertura.py coverage\\lcov.info'

- "publishCoverage(adapters: [coberturaAdapter('coverage.xml')], sourceFileResolver: sourceFiles('NEVER_STORE'))"

- stage: 'Build'

steps:

- bat 'flutter build apk --debug'

- archiveArtifacts '**/*.apk'

- stage: 'Distribute'

steps:

- "appCenter(apiToken: 'aa4185245d89f03465b707ffdef6d48299e99282', ownerName: 'xxxxxxxxxxoutlook.com', appName: 'SampleFlutterApp', pathToApp: 'build/app/outputs/apk/debug/app-debug.apk', distributionGroups: 'Dev-Distribution', releaseNotes: 'Security Bug Fixed - Ticket 2020.08.20')"

Done!

Conclusion

In this chapter, we have created the CI/CD pipeline for sample applications written in Flutter. It covers continuous integration that includes code analysis, unit test execution, code coverage, and build creation for sample flutter application. We also verified if all dependencies are installed properly or not before building and distributing the sample Flutter application. We have also covered app distribution using the App center.

In the next chapter, we will cover DevOps practices implementation for Ionic Cordova applications. We will configure CI/CD for Ionic Cordova applications.

Multiple choice questions

Code Coverage tab provides the following details:

Covered Lines

Uncovered Lines

Line Coverage

All of these

State True or False: App center can be used to create build pipeline.

True

False

Answer

d

a

Questions

What is the difference between the App Center and App Services?

How to analyze code written in Flutter?

What are the main features of Flutter and why it is more developer friendly?

Which plugins are required plugins for Junit (unit testing) and Cobertura (code coverage) output?

CHAPTER 4

Building CI/CD Pipeline with YAML for Ionic Cordova Application

"Believe in yourself! Have faith in your abilities! Without a humble but reasonable confidence in your own powers you cannot be successful or happy."

—*Norman Vincent Peale*

An multinational software organization makes applications for the News industries. The marketing team decides to promote mobile apps for end-users and technical architects suggest hybrid apps rather than native apps. Project teams decide to use Ionic Cordova for edutainment and media industries-related apps. Ionic provides robust UI components for building a hybrid mobile application. Following are some important features or advantages of the Ionic framework such as open source, library of mobile-optimized UI components, gestures, reusable, components, write once, run anywhere, and it has MIT license. Management wants to automate manual processes in application lifecycle management and to increase productivity with frequent deployments implementing continuous integration and continuous delivery. In this chapter, we will cover the CI/CD implementation of Ionic Cordova

applications with Jenkins. We will use Pipeline as a YAML to create CI/CD pipeline. We will distribute an application to the App Center to a specific group. We will also provide some valuable " *notes* " related to DevOps, culture, challenges, market trends, and so on for better understanding.

Structure

In this chapter, we will discuss the following topics:

DevOps tools, expected deliverables, or features

Multi stage-YAML pipeline for Ionic Cordova app

Continuous integration for Ionic Cordova – Android app

Continuous delivery for Ionic Cordova – Android app

Objectives

This chapter will introduce how to implement continuous integration and continuous delivery for Ionic Cordova apps using Pipeline as a YAML. After studying this unit, you should be able to:

Understand how to perform static code analysis for the Ionic Cordova application

Execute unit tests

Calculate code coverage

Verify build quality

DEVOPS TOOLS, EXPECTED DELIVERABLES, OR FEATURES

The organization has selected one application as a pilot while there are other three early adopter Ionic Cordova applications waiting in the queue. We have a responsibility to create CI/CD pipeline using pipeline as a YAML in Jenkins. Following is the

list of tools, and deliverables that will be integrated into the pipeline:

pipeline:

pipeline:

pipeline:

pipeline: pipeline: pipeline:

pipeline:

pipeline:

pipeline:

pipeline: pipeline:

pipeline: pipeline: pipeline: pipeline: pipeline:

pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline:

pipeline: pipeline:

pipeline:

pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline:
pipeline:

pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline:
pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline:

pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline:

pipeline: pipeline: pipeline: pipeline: pipeline: pipeline:

pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline:
pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline:

pipeline:
pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline:
pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline:
pipeline: pipeline: pipeline: pipeline: pipeline: pipeline:
pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline:
pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline:

pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline:
pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline:

Table 4.1: Tools and deliverables

In the next section, we will create Pipeline as a YAML for sample application in a step-by-step manner.

Multi-stage CI/CD pipeline for Ionic Cordova app

In this chapter, we will cover CI/CD for sample Ionic Cordova application. Following is the Big Picture for CI/CD implementation of the Ionic Cordova App:

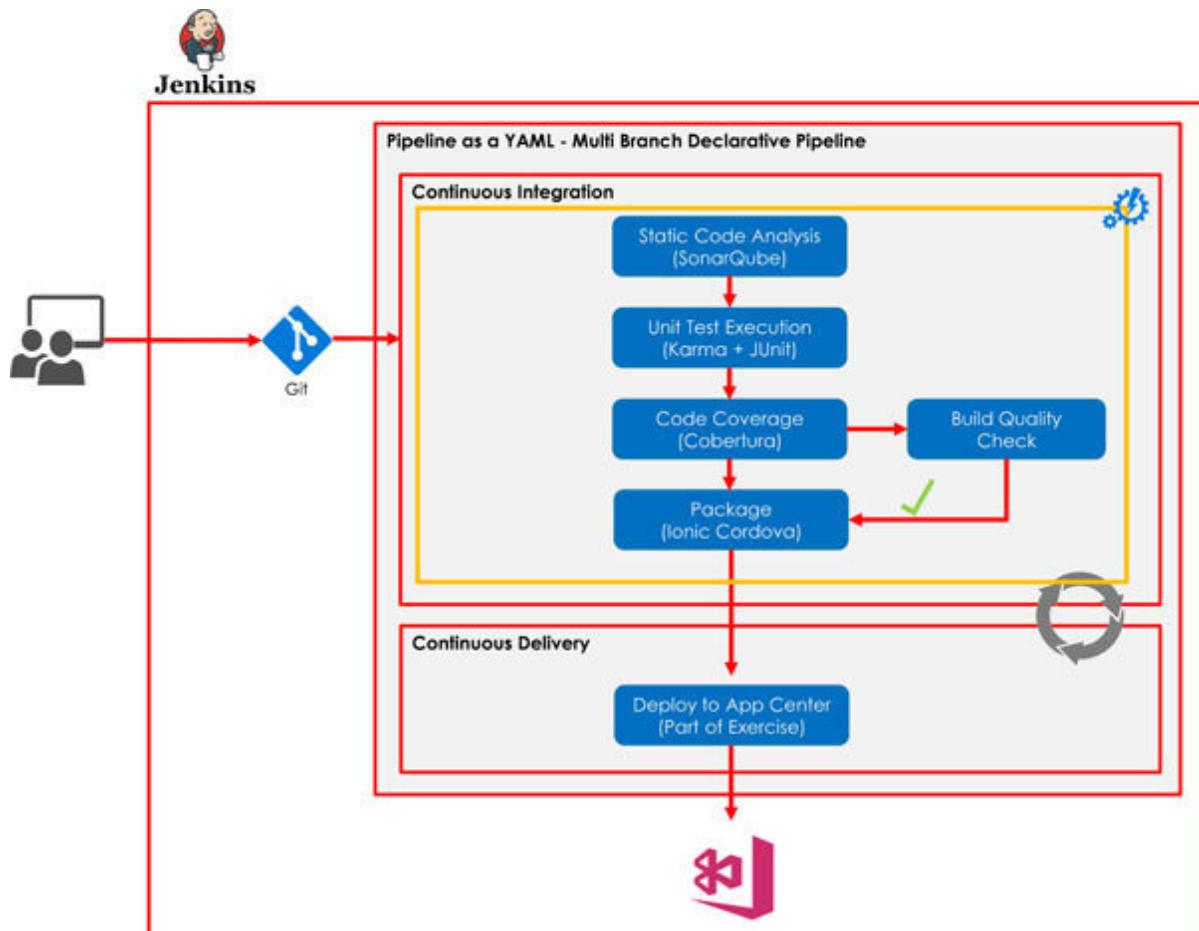


Figure 4.1: Big picture for CI/CD of Ionic Cordova App

Let us create pipeline for a sample Ionic Cordova application using YAML.

We will configure static code analysis/Lint analysis using SonarQube/Lint tools, unit test execution, and code coverage calculation and report for sample Ionic Cordova application.

[*Continuous integration for Ionic – Android App*](#)

Let us configure plugins and other blocks related to Junit and coverage:

Let us do a few configurations in file karma.conf.js.

Add required plugins for Junit (unit testing) and Cobertura (code coverage) output shown as follows:

```
plugins: [  
  require('karma-jasmine'),  
  
  require('karma-chrome-launcher'),  
  
  require('karma-mocha-reporter'),  
  
  require('karma-coverage-istanbul-reporter'),  
  
  require('karma-junit-reporter'),
```

```
require('@angular-devkit/build-angular/plugins/karma')

],
```

Add Cobertura in reports key section to karma.conf.js shown as follows:

```
coverageIstanbulReporter: {

  dir: require('path').join(__dirname, './coverage'),
  reports: ['html', 'lcovonly','cobertura'],
  fixWebpackSourcePaths: true

},
reporters: ['progress', 'kjhtml', 'junit'],
```

Add JunitReporter configuration details in karma.conf.js if it is not available in the file shown as follows:

```
junitReporter: {
```

```
    outputDir: '',  
  
    outputFile: undefined,  
  
    suite: '',  
  
    useBrowserName: true,  
  
    nameFormatter: undefined,  
  
    classNameFormatter: undefined,  
  
    properties: {}  
  
},
```

Configure Headless Browser so test cases can be executed shown as follows:

```
autoWatch: true,  
  
singleRun: true,  
  
browsers: ['ChromeHeadlessNoSandbox'],
```

```
customLaunchers: {  
  
  ChromeHeadlessNoSandbox: {  
  
    base: 'ChromeHeadless',  
  
    flags: ['--no-sandbox']  
  
  },  
},
```

Configure Package.json with scripts shown as follows:

```
"scripts": {  
  
  "ng": "ng",  
  
  "start": "ng serve",  
  
  "build": "ng build",  
  
  "test": "ng test --code-coverage",  
  
  "lint": "ng lint",
```

```
  "e2e": "ng e2e",

  "postinstall": "webdriver-manager update --standalone false -
-gecko false"

  },

"devDependencies": {

  .

  .

  .

  "karma-junit-reporter": "2.0.1",

  .

  .

  .

  }

},
```

npm install installs the dependencies in the local
node_modules folder. If we use -g or --global then it installs

the current working directory as a global package. The important thing to note is npm install execution will install all the modules available as dependencies in package.json. To skip installation of modules available in devDependencies section, we need to provide production flag.

For more details visit: <https://docs.npmjs.com/cli/install>

In the continuous integration stage, configure tasks related to Static Code Analysis using SonarQube, NPM module installation, update, test execution, and publish unit tests and code coverage results. Following are commands to perform few operations:

// SonarQube Analysis command

```
sonar-scanner.bat -  
Dsonar.login=cba67104bb4f4ef081b55e7ef43168a40b49d6b9 -  
Dsonar.projectVersion=1.0 -Dsonar.projectKey=ionic-sample -  
Dsonar.sources=src
```

// -Dsonar.login will have sonarqube token as its value. Generate token in Sonarqube.

```
// -Dsonar.sources will have a name of the directory in which your  
source code resides so that can be analyzed using SonarQube
```

```
// Node Modules installation
```

```
npm install && ng update && npm install karma-junit-reporter  
--save-dev && npm audit fix && npm i @angular-devkit/build-  
angular@0.803.25
```

```
// Test execution
```

```
npm run test
```

```
// Ionic cordova commands
```

```
npm i @angular-devkit/build-angular@0.803.25 && ionic  
cordova platform add android. & ionic cordova build android -  
prod'
```

Blue Ocean dashboard will look like below for the Continuous Integration Stage:

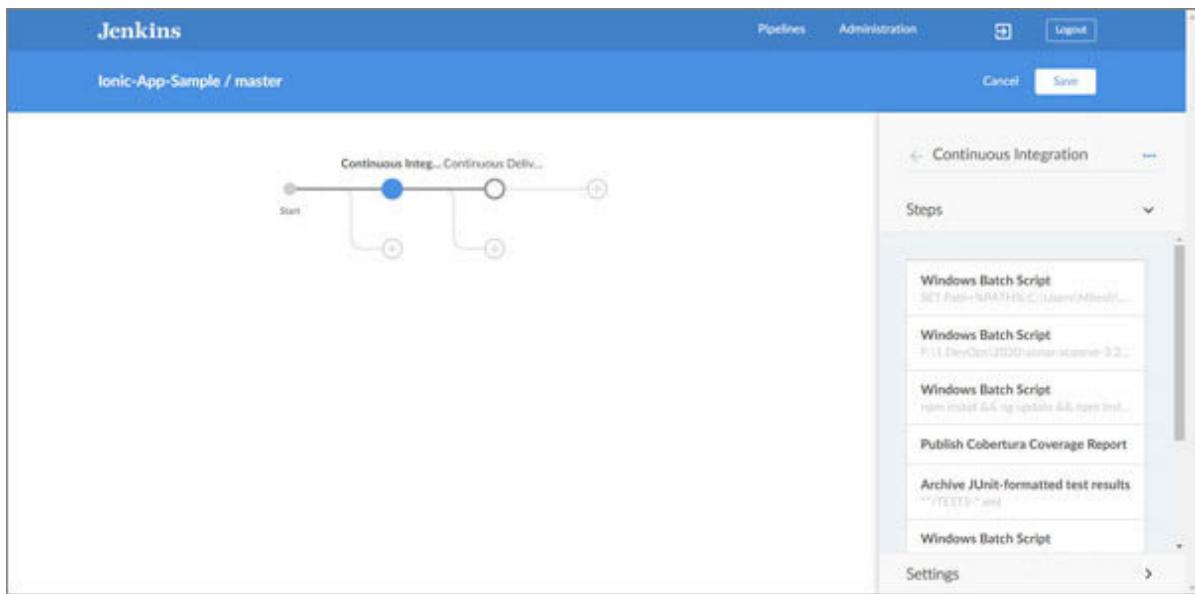


Figure 4.2: Continuous Integration Stage

While executing the above commands in the Jenkins pipeline, we encountered few issues and hence we have listed them down with solution that worked to resolve them here:

Issue Vulnerabilities

found 6 low severity vulnerabilities

run `npm audit fix` to fix them, or `npm audit` for details

at `getTargetPackageSpecFromNpmInstallOutput`
`(C:\Users\Tempuser\AppData\Roaming\npm\node_modules\cordova\node_modules\cordova-fetch\index.js:104:11)`

```
at processTicksAndRejections
(internal/process/task_queues.js:97:5) {
code: 0,
context: undefined
}
```

Solution: execute npm audit fix

Issue Job name "..getProjectMetadata" does not exist.

An unhandled exception occurred: Job name
"..getProjectMetadata" does not exist.

See "C:\Users\Tempuser\AppData\Local\Temp\ng-QlctEV\angular-errors.log" for further details.

```
npm ERR! code ELIFECYCLE
npm ERR! syscall spawn
npm ERR! file C:\WINDOWS\system32\cmd.exe
npm ERR! errno ENOENT
npm ERR! ionic-conference-app@0.0.0 test: `ng test --code-coverage`
npm ERR! spawn ENOENT
npm ERR!
```

```
npm ERR! Failed at the ionic-conference-app@0.0.0 test script.  
npm ERR! This is probably not a problem with npm. There is  
likely additional logging output above.
```

```
npm ERR! A complete log of this run can be found in:
```

```
npm ERR!     C:\Users\Tempuser\AppData\Roaming\npm-  
cache\_logs\2020-06-17T07_13_00_542Z-debug.log
```

```
script returned exit code 1
```

Solution: It looks like a problem with @angular-devkit/build-angular. Command npm audit reported vulnerabilities in the version of @angular-devkit/build-angular After executing npm audit fix We found the error: An unhandled exception occurred: Job name "..getProjectMetadata" does not exist.

Execute npm i @angular-devkit/build-angular Or downgrade it by specifying a previous version, such as 0.803.25 with the command: npm i @angular-devkit/build-angular@0.803.25.

Issue stderr: warning: failed to remove

```
stderr: warning: failed to remove  
platforms/android/CordovaLib/build/intermediates/javac/debug/co  
mpileDebugJavaWithJavac/classes/org/apache/cordova/NativeToJs  
MessageQueue$OnlineEventsBridgeMode  
$OnlineEventsBridgeModeDelegate.class: Filename too long
```

```
at org.jenkinsci.plugins.gitclient.CliGitAPIImpl.launchCommandIn
(CliGitAPIImpl.java:2430)
at org.jenkinsci.plugins.gitclient.CliGitAPIImpl.launchCommandIn
(CliGitAPIImpl.java:2360)
at org.jenkinsci.plugins.gitclient.CliGitAPIImpl.launchCommandIn
(CliGitAPIImpl.java:2356)
at org.jenkinsci.plugins.gitclient.CliGitAPIImpl.launchCommand
(CliGitAPIImpl.java:1916)
at org.jenkinsci.plugins.gitclient.CliGitAPIImpl.launchCommand
(CliGitAPIImpl.java:1928)
at org.jenkinsci.plugins.gitclient.CliGitAPIImpl.clean(CliGit
APIImpl.java:1010)
at
hudson.plugins.git.extensions.impl.CleanBeforeCheckout.decorateFe
tchCommand(CleanBeforeCheckout.java:44)

at hudson.plugins.git.extensions.GitSCMExtension.decorateFetch
Command(GitSCMExtension.java:288)
at hudson.plugins.git.GitSCM.fetchFrom(GitSCM.java:905)
... 11 more
Error fetching remote repo 'origin'
```

Solution: Remove the platform directory from the workspace and execute the pipeline again.

Issue 4: stderr: warning: failed to remove platforms/: Directory not empty

Solution: Probably you are using the folder or you have opened it in explorer and hence when the pipeline is getting executed, it is not able to remove all directories at the time of checkout.

Issue Could not find an installed version of Gradle

Could not find an installed version of Gradle either in Android Studio,

or on your system to install the Gradle wrapper. Please include Gradle

in your path, or install Android Studio
script returned exit code 1

Solution: Set Gradle Path (SET Path=%PATH%;C:\\gradle-4.6-all\\gradle-4.6\\bin)

Issue Could not find plugin "proposal-numeric-separator" in angular app

Solution: Add "@babel/compat-data": "7.8.0" in devDependencies section in package.json.

ng update command execution updates your application and its dependencies. For more details visit

Execute `npm install karma-junit-reporter --save-dev` to keep karma-junit-reporter as a devDependency in package.json. It results in npm automatically add it package.json.

Use continuous integration and continuous delivery for all the environments, including production.

Let us configure continuous integration. Create a Jenkins pipeline job from the Jenkins dashboard and click on configure to set Android application's repository:

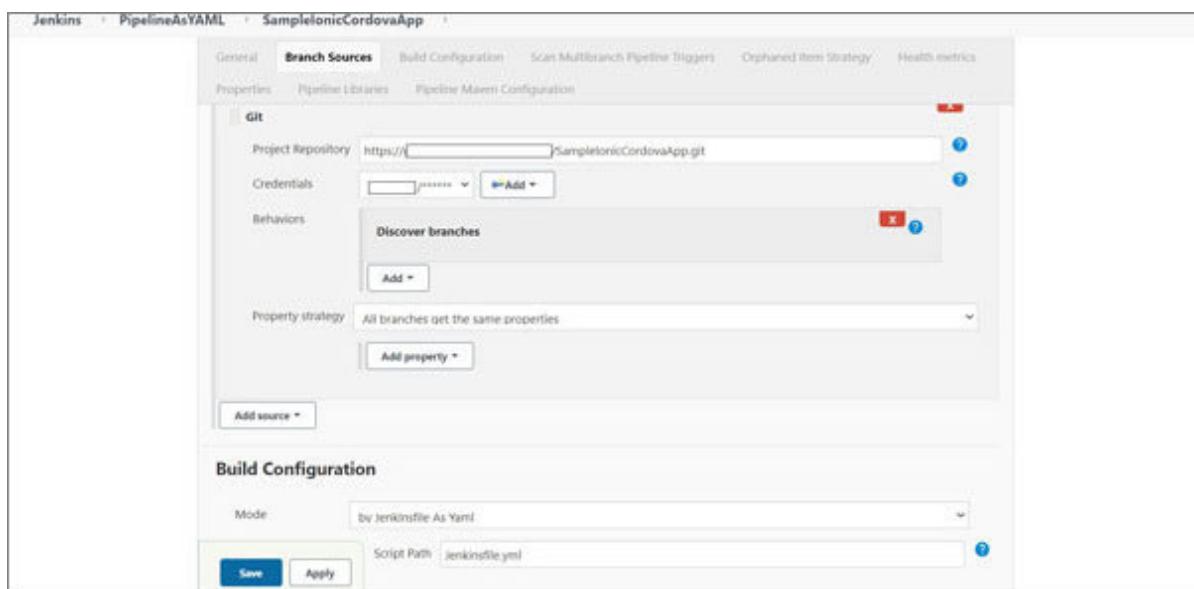


Figure 4.3: Branch Sources

Identification of key metrics and tracking of them is required to compare AS IS and later improvements.

Click on the status link to get list of branches available where Jenkinsfile is available or branches that are filtered based on wildcards:

S	W	Name	Last Success	Last Failure	Last Duration	Fav
		master	4 days 17 hr - #2	4 days 21 hr - #5	16 mins	

Figure 4.4: Master branch of Sample Ionic Cordova App

Stash and unstash are designed to share files between stages and nodes. Archive Artifacts helps to store the output files such as WAR file or APK file or IPA file after we build the project. Users can download artifacts later. We can access

**archived artifacts from the Jenkins Dashboard/Project
Dashboard. Artifacts are available until the build log is kept.**

Following is the script to perform static code analysis using SonarQube. We will configure environment variables, define agent for execution and configure SonarQube step in the Declarative pipeline. Download and Start SonarQube and create Sonar Token that we can utilize in the pipeline step:

pipeline:

environment:

```
ANDROID_HOME: 'C:\\Program Files  
(x86)\\Android\\android-sdk'
```

```
JAVA_HOME: 'C:\\Program Files\\Java\\jdk1.8.0_111'
```

```
GRADLE_HOME:  
'C:\\Users\\.gradle\\wrapper\\dists\\gradle-4.6-  
all\\bcst21l2brirad8k2ben1letg\\gradle-4.6'
```

agent:

```
label: 'master'
```

stages:

- stage: 'Static Code Analysis'

steps:

```
- bat 'F:\\1.DevOps\\2020\\sonar-scanner-3.2.0.1227-windows\\bin\\sonar-scanner.bat' -  
Dsonar.host.url=http://localhost:9000/ -  
Dsonar.login=d39153de87276ec525e77b1601b94e7ddffd2f23 -  
Dsonar.projectVersion=1.0 -Dsonar.projectKey=sample-ionic-cordova-app -Dsonar.sources=src'
```

In case of Jenkins crash to state, when configurations cannot be recovered then Jenkinsfile can help us get up and running in no time as all steps to execute pipeline are written in the Jenkinsfile itself.

Verify the logs for the code analysis stage:



Figure 4.5: Static Code Analysis Stage logs

Goals: End-to-end automation, standardized toolset and processes, communication and collaboration tools, effective usage of cloud and container resources, common goals and roadmap based on maturity model, common key performance indicators, continuous improvement, and continuous innovation.

The console log will provide URL for the SonarQube dashboard or login with SonarQube and get details about Android application from Projects:

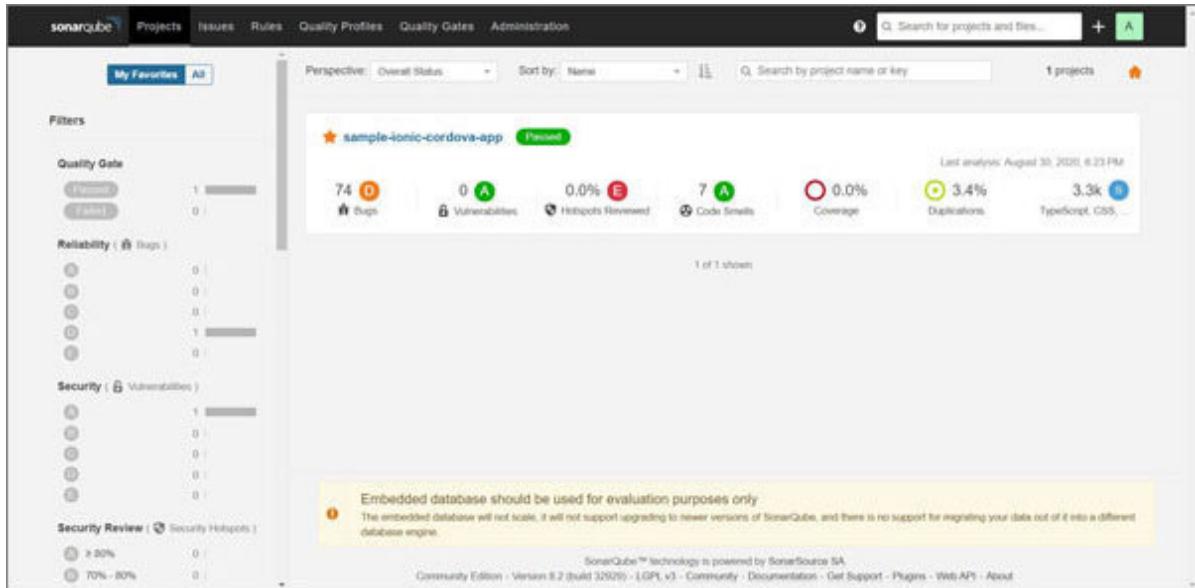


Figure 4.6: SonarQube Dashboard

Click on the project name to get more details about code analysis of an application in SonarQube along with the Quality Gate result:

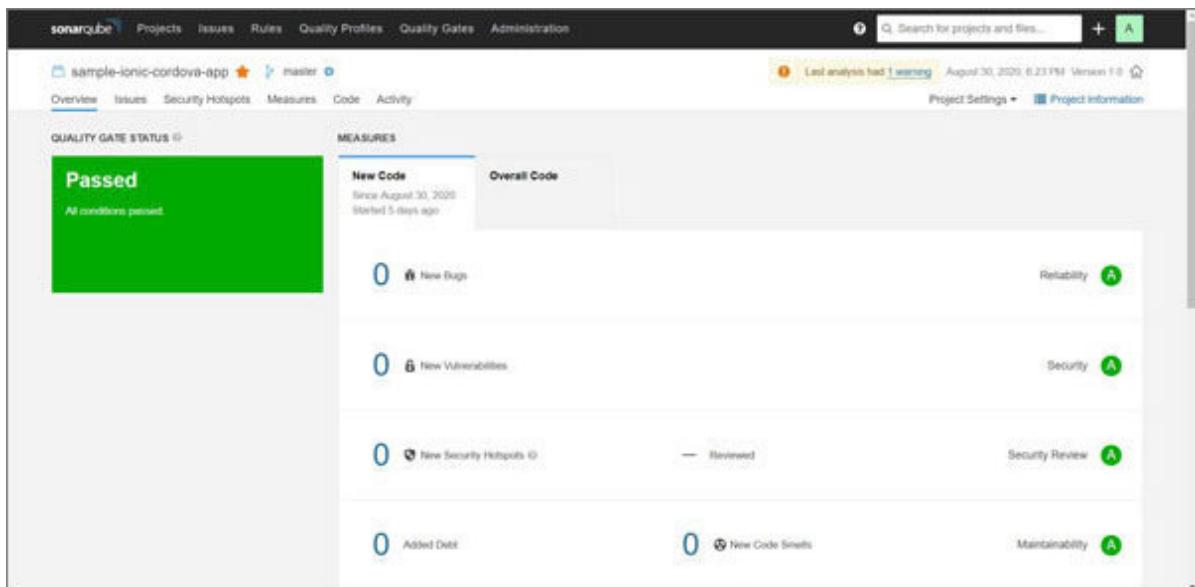


Figure 4.7: Quality Gate Status and other details

Set email or slack notification for each failure or important event so that the pipeline is maintained in a healthy state.

Click on the Overall Code tab to get more insights on Code analysis result:

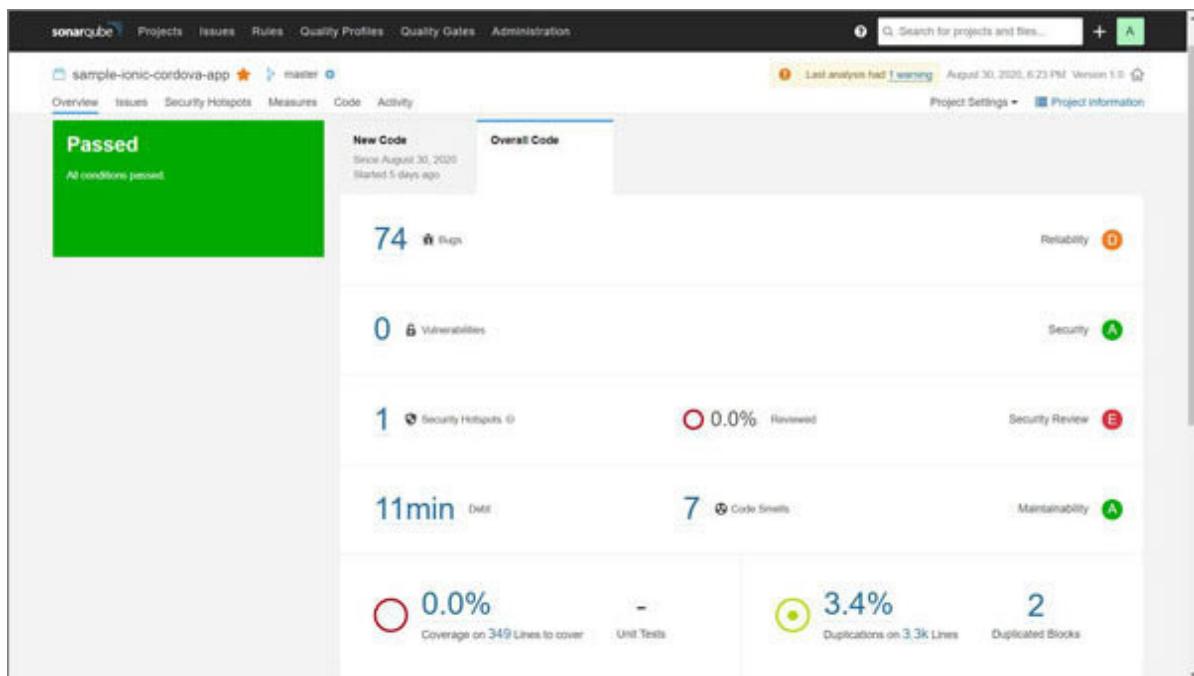


Figure 4.8: Overall code details in SonarQube

Click on the **Issues** tab in SonarQube to get details on bugs, vulnerabilities, and code smells:

The screenshot shows the SonarQube web interface for the 'sample-ionic-cordova-app' project. The top navigation bar includes 'Projects', 'Issues', 'Rules', 'Quality Profiles', 'Quality Gates', and 'Administration'. A search bar at the top right contains the placeholder 'Search for projects and files...'. Below the navigation is a header for the 'sample-ionic-cordova-app' project, master branch, and analysis date 'August 30, 2020, 6:23 PM Version 1.0'. A green button on the right has the letter 'A' on it.

The main area is titled 'Issues' and displays a list of 74 issues. The left sidebar contains filters for 'Type' (selected 'Bug'), 'Severity' (selected 'Minor'), and other options like 'Resolution', 'Status', 'Security Category', etc. The list of issues is grouped under 'scnappapp component locs' and shows 11 items, each with details like '5 days ago', '1.1.', 'T-', and 'No tags'.

Issue Description	Severity	Impact	Effort	Comments	Last Analysis
Unexpected unknown type selector "ion-menu". Why is this an issue?	Bug	Critical	Open	Not assigned	5 days ago
Unexpected unknown type selector "ion-content". Why is this an issue?	Bug	Critical	Open	Not assigned	5 days ago
Unexpected unknown type selector "ion-menu". Why is this an issue?	Bug	Critical	Open	Not assigned	5 days ago
Unexpected unknown type selector "ion-item". Why is this an issue?	Bug	Critical	Open	Not assigned	5 days ago
Unexpected unknown type selector "ion-item". Why is this an issue?	Bug	Critical	Open	Not assigned	5 days ago
Unexpected unknown type selector "ion-list". Why is this an issue?	Bug	Critical	Open	Not assigned	5 days ago
Unexpected unknown type selector "ion-menu". Why is this an issue?	Bug	Critical	Open	Not assigned	5 days ago
Unexpected unknown type selector "ion-list-header". Why is this an issue?	Bug	Critical	Open	Not assigned	5 days ago

Figure 4.9: Issues in SonarQube

Once Static code analysis is done, let us see YAML script block for continuous integration stage that will perform npm install, install junit-related dependencies, execute unit tests, publish unit test results, publish code coverage, build distribution content or artifacts, and archive artifacts:

- stage: 'Continuous Integration'

steps:

- bat 'SET

```
Path=%PATH%;C:\\\\Users\\\\Mitesh\\\\.gradle\\\\wrapper\\\\dists\\\\gradle
```

```
-4.6-all\\bcst21l2brirad8k2ben1letg\\gradle-4.6\\bin'
```

```
- bat 'npm install && ng update && npm install && npm audit fix && npm i @angular-devkit/build-angular@0.803.25 && npm run test'
```

```
- "cobertura(coberturaReportFile: '*/coverage/cobertura-coverage.xml', sourceEncoding: 'ASCII')"
```

```
- junit '**/TESTS-*.xml'
```

```
- bat 'npm i @angular-devkit/build-angular@0.803.25 && ionic cordova platform add android. & ionic cordova build android --prod'
```

```
- "archiveArtifacts '**/*.apk'"
```

Execute the pipeline and verify the logs in Blue Ocean dashboard:

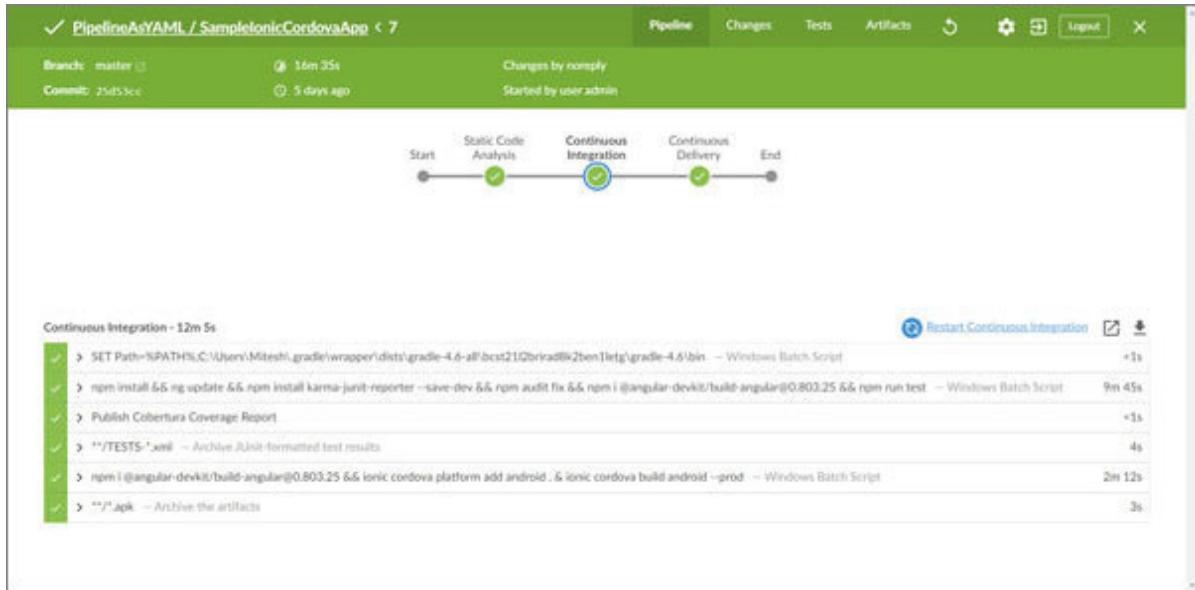


Figure 4.10: CI Stage logs

How to quickly create a pipeline?

Learn Groovy.

Create a pipeline script in Jenkinsfile.

Use the pipeline Syntax section available in Jenkins pipeline or Jenkins job or use Blue Ocean to create pipeline syntax easily.

The efficient answer is to use the pipeline Syntax section available in Jenkins pipeline or Jenkins job or use Blue Ocean to create pipeline syntax easily.

Click on the Tests link to get results of all Unit tests:

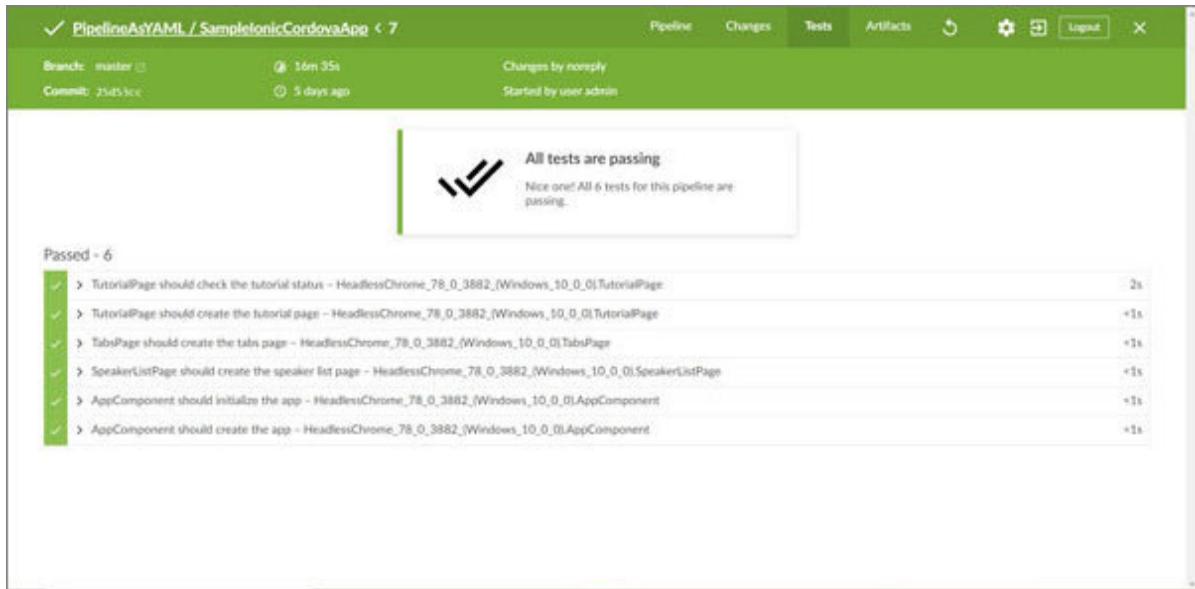


Figure 4.11: Unit Tests report

Click on **Status** link on the left sidebar of pipeline job:

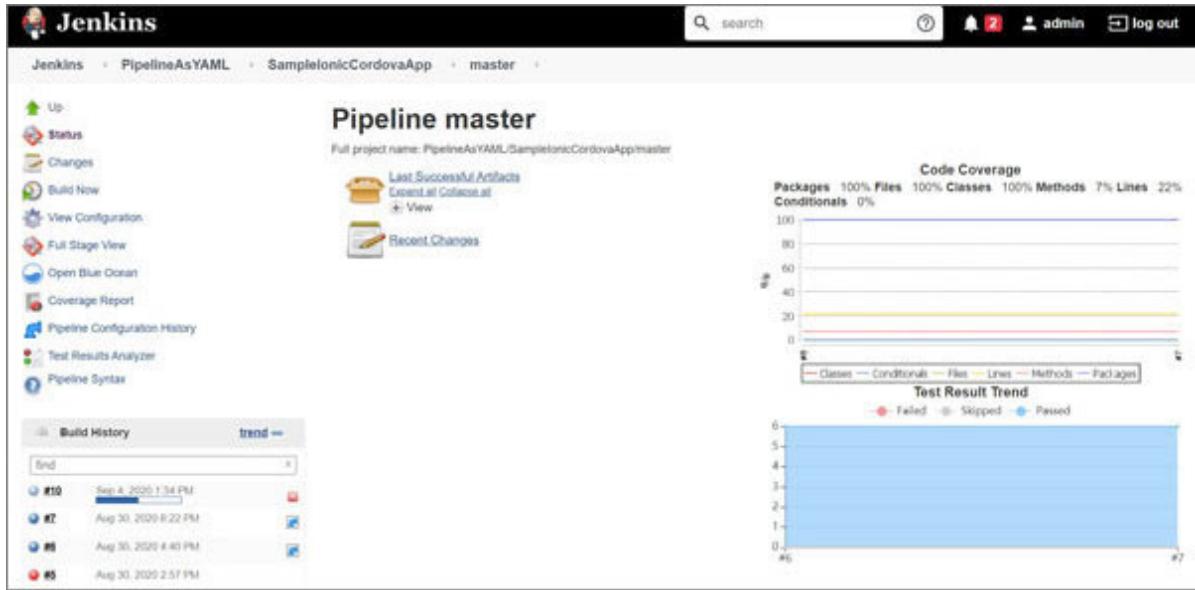


Figure 4.12: Pipeline execution status

The Test Results Analyzer plugin provides detailed information regarding the build result history of test-class, test-class, and test-package in a tabular form. The plugin can be used by using the **junit** or **TestNG** (in case of TestNG) step of Jenkins in pipelines or other Jenkins templates.

Metrics: % decrease in Bugs, Vulnerabilities, and Code Smells, % increase in Code Coverage, % increase in test automation, % decrease in efforts of deployment, % decrease in the overall release process:

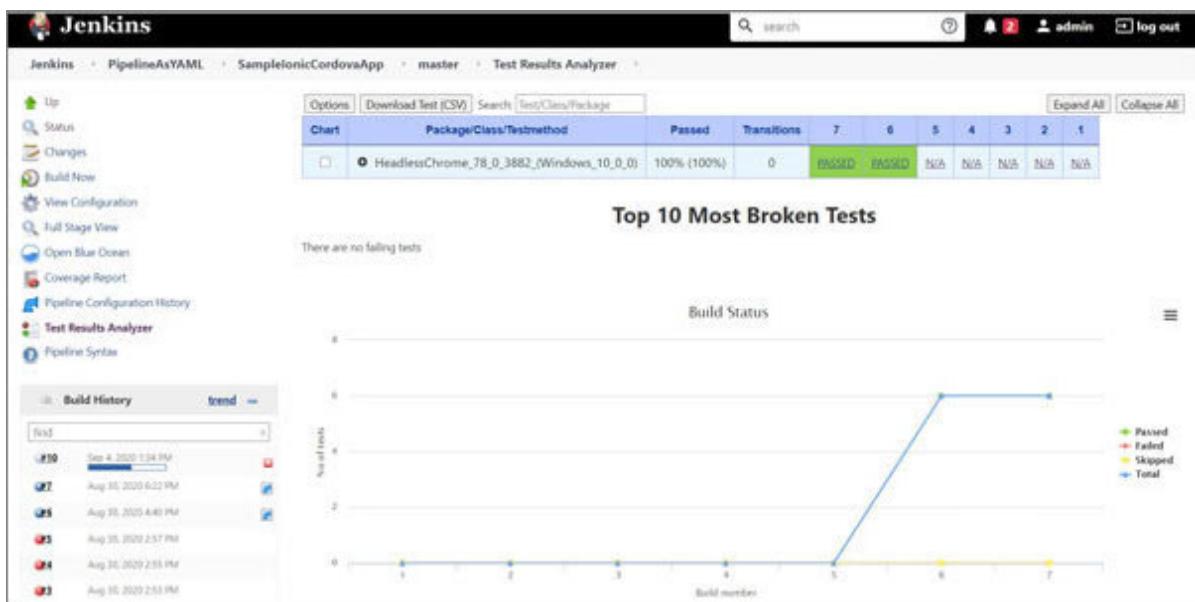


Figure 4.13: Test Results Analyzer

The Test Results Analyzer plugin allows users to filter the test results based on its execution status such as passed, failed, and skipped. Click on the **Test Results Analyzer** link on the left side of pipeline job to get more details. It also supports the generation of Graphs such as line charts, pie charts, and bar charts:



Figure 4.14: Test Results Analyzer – Pie Chart

Cobertura plugin allows you to publish code coverage report from Cobertura. Jenkins will generate the trend report of coverage. Verify the Coverage report in Jenkins dashboard for the pipeline job:

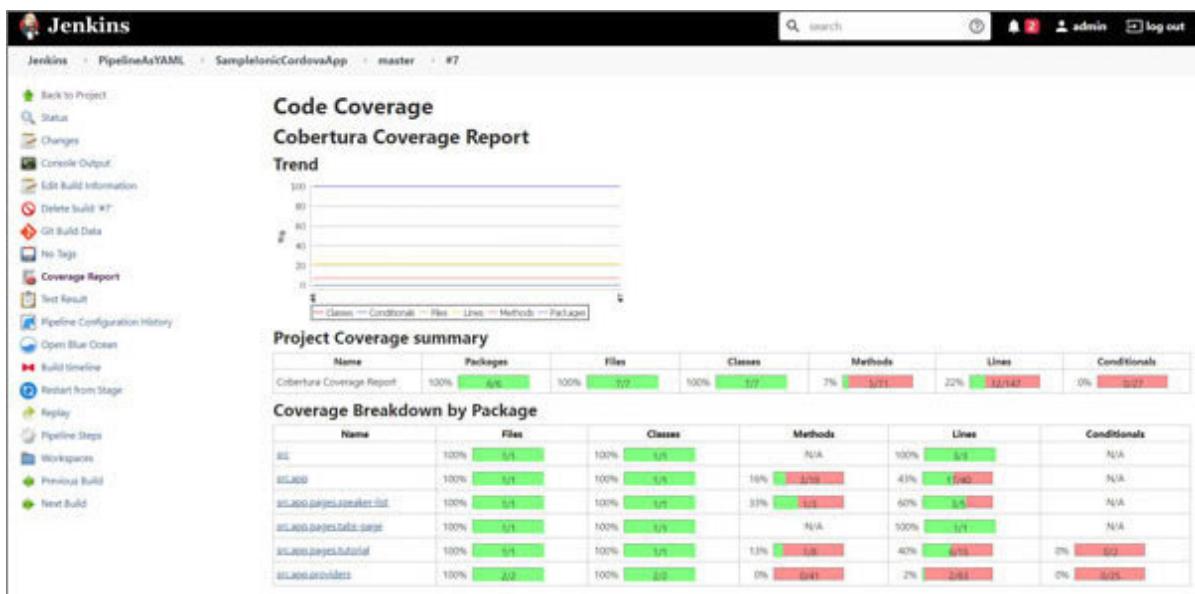


Figure 4.15: Cobertura Coverage Report

Click on the **Status** link available in Jenkins dashboard to get details about Test results and Coverage reports for a specific build:

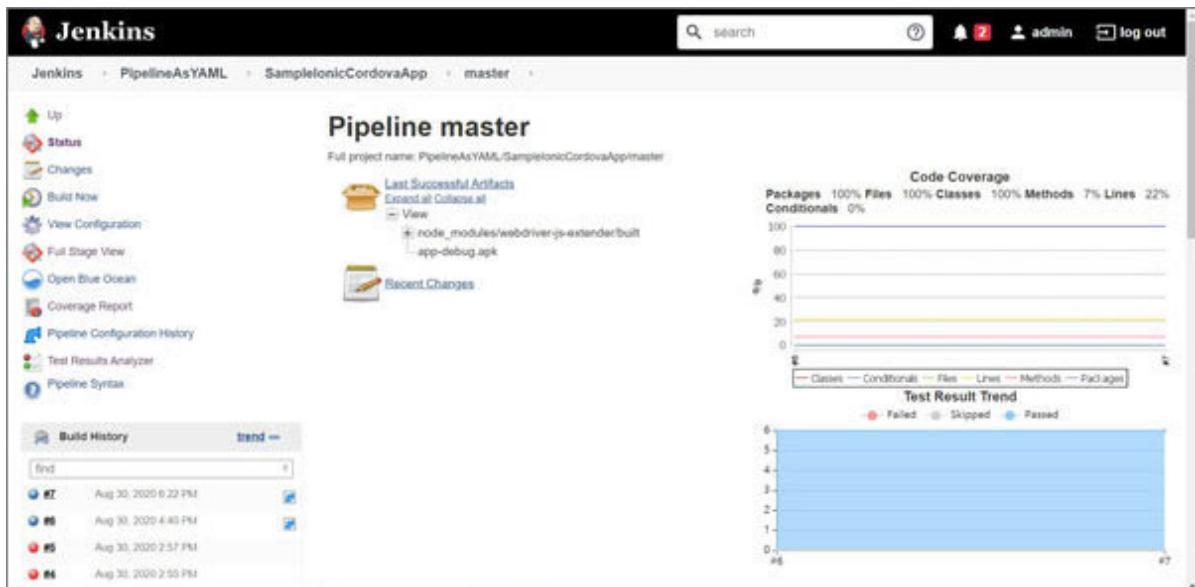


Figure 4.16: Archived Artifact

In the next section, we will verify YAML script for continuous delivery to distribute the application to the app center.

[*Continuous delivery for Ionic Cordova App*](#)

App Center is used to distribute mobile applications to the QA team. It is also used to build, test, and distributed Android, iOS, and Windows applications.

Go to

Log in with the appropriate method. Provide Username for login.

Let's add a new App in App Center by clicking on **Add**

Select **App** and

Click on **Add new**

As of now, there is no distribution group is available in the Android App we have created. Distribution groups help to organize testers and manage who can have access to the application.

Click on **Add**

Provide Group name and click on **Create**

A New Group is available. We will configure this group while we try to upload Android Package from Jenkins.

To integrate the App center in Jenkins, we will need an API token. Let us create it first.

Let us go to **Account Settings** of App Center.

Scroll down to the **Account Settings** page.

Click on the **New API**

Provide **Description** and

Click on the **Add new API**

Copy the **API**

Go to Jenkins and open the pipeline Syntax section of the sample pipeline project.

Select **Upload** to App Center step in the Sample step.

Provide the values such as API Token that we created earlier, Owner name from App center, App Name, Path to the APK file, and Distribution group that we created:

- stage: 'Continuous Delivery'

steps:

- "appCenter(apiToken: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
ownerName: 'xxxxxx-xxxxxx.com', appName:
'SampleIonicCordovaApp', pathToApp:
'platforms/android/app/build/outputs/apk/debug/app-debug.apk',
distributionGroups: 'Dev-Distribution', releaseNotes: 'Bug Fixed -
Ticket 2020.05.30')"

Verify the continuous delivery stage logs:

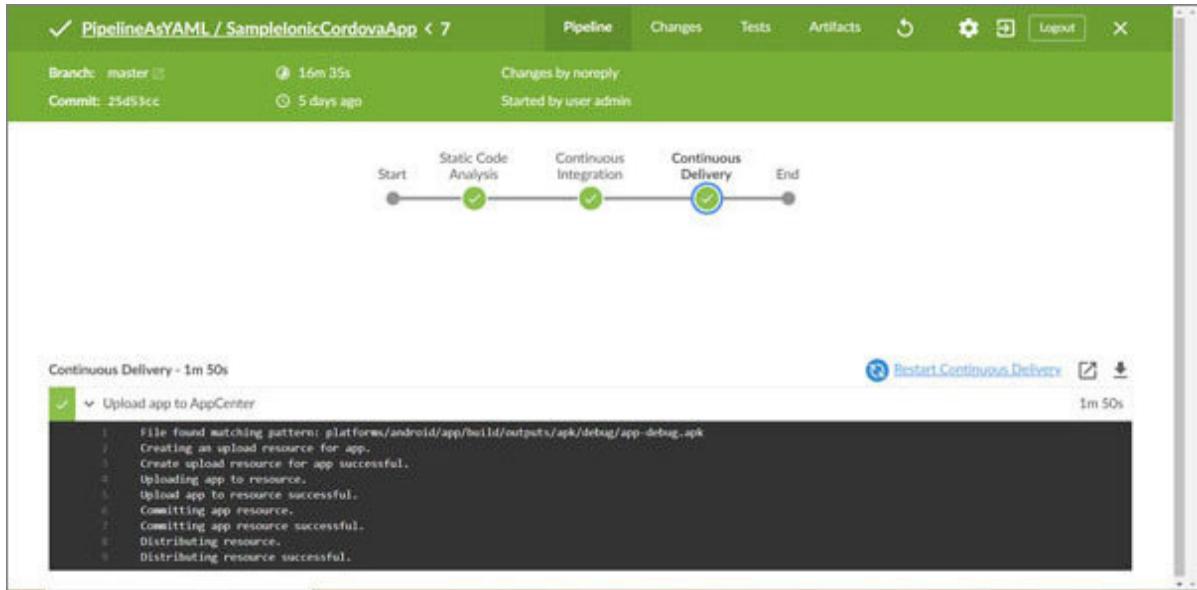


Figure 4.17: CD Stage logs

Click on a Full Stage View in the left side bar to get the pipeline in a full view:

A screenshot of the Jenkins master - Stage View page. The top navigation bar includes the Jenkins logo, search, notifications (2), user "admin", and "log out". The current path is "Jenkins > PipelineAsYAML > SampleIonicCordovaApp > master > Full Stage View". The main content is titled "master - Stage View". It features a table with columns for "Declarative: Checkout SCM", "Static Code Analysis", "Continuous Integration", "Continuous Delivery", and "Declarative: Post Actions". The first row shows average stage times: 56s, 1min 51s, 26min 44s, 37s, and 754ms. Below this are four rows of data corresponding to specific build runs on August 30, 2016. The first three rows have green backgrounds and show times of 7s, 12min 4s, and 1min 50s respectively. The fourth row has a pink background and shows times of 54s, 31min 59s, and 737ms. The last two columns for "Continuous Delivery" and "Declarative: Post Actions" are red and show "36min 8s" and "465ms" with the word "failed" underneath. The first column for "Declarative: Checkout SCM" also has a pink background and shows "1min 47s". The second column for "Static Code Analysis" shows "2min 2s". The last two columns for "Continuous Integration" and "Declarative: Post Actions" have green backgrounds.

Figure 4.18: Jenkins – Full Stage view

Go to App Center and verify the **Releases** section. The package will be available for download:

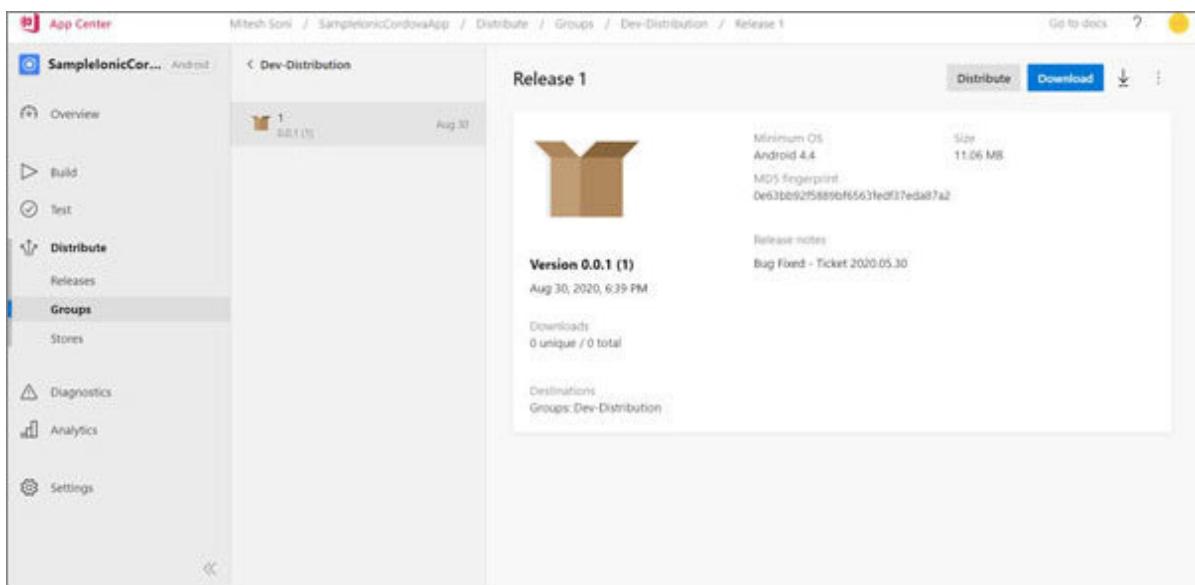


Figure 4.19: App Center Distribution

Multibranch pipelines help to create pipelines based on the `Jenkinsfile` available in the branch. How it is helpful? Consider the scenario when you want to have a different kind of configuration in different environments. In such cases, each branch specific to dev, test, staging, and prod branch can have different `Jenkinsfile` and each file can have a different configuration in `Jenkinsfile`. While creating a pipeline in Blue

Ocean, it will detect multiple branches with Jenkinsfile and considering all the agents are available, the pipeline will be executed:

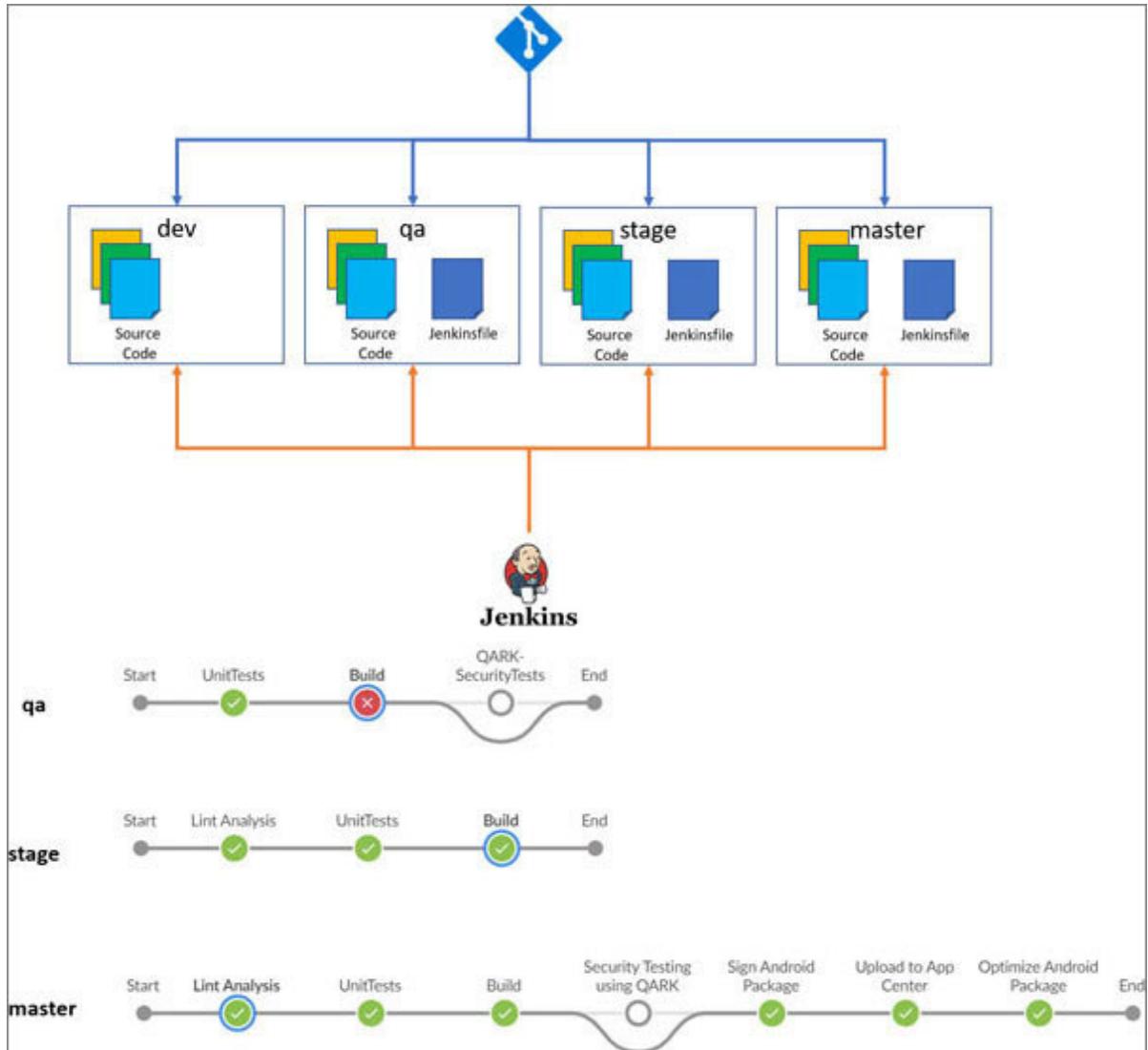


Figure 4.20: Multi-Branch pipeline

It is useful when Jenkins is crashed and you want to receiver all your pipelines. If automation setup and agents are available

then all pipelines can be restored as all pipelines reside in a branch (version control system) in the form of Pipeline as a Code.

Multibranch pipelines inject BRANCH_NAME and CHANGE_ID information about the branch through the env global variable.

Example of Scan Multibranch Pipeline Log:

```
Checking branches...
Checking branch sprint_cleanup
'Jenkinsfile' not found
Does not meet criteria
Checking branch jenkins
'Jenkinsfile' found
Met criteria
Changes detected: jenkins
(899e3705ec381e81d4886af637f8a10d56a4bceb →
33935c216079b9b9083dc10c8d24a24b019b41fa)
Scheduled build for branch: jenkins
Checking branch master
'Jenkinsfile' found
Met criteria
Checking branch param_rename
'Jenkinsfile' not found
Does not meet criteria
```

Checking branch update_click

'Jenkinsfile' not found

Does not meet criteria

Checking branch blog-55555

'Jenkinsfile' not found

Does not meet criteria

Checking branch add_header

'Jenkinsfile' not found

Does not meet criteria

Checking branch selection

'Jenkinsfile' not found

Does not meet criteria

Checking branch rv-selection

'Jenkinsfile' not found

Does not meet criteria

Checking branch transition

'Jenkinsfile' not found

Does not meet criteria

Processed 10 branches

[Mon May 18 14:02:01 IST 2020] Finished branch indexing.

Indexing took 6.1 sec

Finished: SUCCESS

In nutshell, Multibranch pipeline project, Jenkins automatically discovers, manages, and executes pipelines for all the branches which meet the criteria (Jenkinsfile in source control)

considering the automation environment including agents are available.

In the next section, we will verify the entire YAML script covered in this chapter.

YAML pipeline script for Ionic Cordova app

Following is the complete YAML script to create CI/CD pipeline for sample Ionic Cordova App:

pipeline:

agent:

label: 'master'

stages:

- stage: 'Static Code Analysis'

steps:

- bat 'F:\\1.DevOps\\2020\\sonar-scanner-3.2.0.1227-windows\\bin\\sonar-scanner.bat -Dsonar.host.url=http://localhost:9000/ -Dsonar.login=d39153de87276ec525e77b1601b94e7ddfd2f23 -

```
Dsonar.projectVersion=1.0 -Dsonar.projectKey=sample-ionic-cordova-app -Dsonar.sources=src'
```

- stage: 'Continuous Integration'

steps:

- bat 'SET

```
Path=%PATH%;C:\\Users\\Mitesh\\.gradle\\wrapper\\dists\\gradle-4.6-all\\bcst21l2brirad8k2ben1letg\\gradle-4.6\\bin'
```

- bat 'npm install && ng update && npm install karma-junit-reporter --save-dev && npm audit fix && npm i @angular-devkit/build-angular@0.803.25 && npm run test'

- "cobertura(coberturaReportFile: '*/coverage/cobertura-coverage.xml', sourceEncoding: 'ASCII')"

- junit '**/TESTS-*.xml'

- bat 'npm i @angular-devkit/build-angular@0.803.25 && ionic cordova platform add android . & ionic cordova build android --prod'

- "archiveArtifacts '**/*.apk'"

- stage: 'Continuous Delivery'

steps:

- "appCenter(apiToken: '75794011ded8da608c433437107efd53d45e9028', ownerId: 'mitesh.soni-outlook.com', appName: 'SampleIonicCordovaApp', pathToApp: 'platforms/android/app/build/outputs/apk/debug/app-debug.apk', distributionGroups: 'Dev-Distribution', releaseNotes: 'Bug Fixed - Ticket 2020.05.30')"

environment:

ANDROID_HOME: 'C:\\Program Files (x86)\\Android\\android-sdk'

JAVA_HOME: 'C:\\Program Files\\Java\\jdk1.8.0_111'

GRADLE_HOME:
'C:\\Users\\Mitesh\\.gradle\\wrapper\\dists\\gradle-4.6-all\\bcst21l2brirad8k2ben1letg\\gradle-4.6'

Done!

Conclusion

In this chapter, we have created the CI/CD pipeline for sample applications written in Ionic Cordova. It covers Continuous Integration that includes code analysis using SonarQube to highlight bugs, vulnerabilities, and code smells; unit test execution, code coverage, and build creation. We have also covered App distribution using App center to specific distribution group created in App center for better communication and collaboration.

In the next chapter, we will cover DevOps practices implementation for Android applications. We will configure CI/CD for Android applications.

Multiple choice questions

Code Coverage tab provides the following details:

Covered Lines

Uncovered Lines

Line Coverage

All of these

State True or be configured in options section in the pipeline

True

False

State True or be configured in the environment section in the pipeline.

True

False

Answer

d

b

a

Questions

What is the difference between the App Center and App Services?

How to configure Headless Browser so test cases can be executed?

Which JunitReporter configuration details needs to be included in karma.conf.js?

What coverageistanbulReporter configuration is required in karma.conf.js?

Which Plugins are required plugins for Junit (unit testing) and Cobertura (code coverage) output?

How to fix vulnerabilities in node modules?

How to solve the following error: "..getProjectMetadata" does not exist?

What can be the cause of the error that indicates a failure to remove the platform directory?

How to solve the issue: Could not find an installed version of Gradle

What is the solution of the issue: could not find plugin "proposal-numeric-separator" in an angular app?

CHAPTER 5

Building CI/CD Pipeline with YAML for Android App

"Believe in yourself! Have faith in your abilities! Without a humble but reasonable confidence in your own powers you cannot be successful or happy."

- —*Norman Vincent Peale*

A healthcare services organization wants to bring focus on building high-quality applications with faster time to market and improved security. Chief Technology Officer (CTO) wants to evolve platform and technology stack to manage user-requests efficiently. In the time of the pandemic, they want to roll out more features for mobile apps without compromising security, compliance, and quality. CTO wants to accelerate application delivery with high quality using a phase-wise implementation of DevOps Practices. The need of the hour is to implement continuous integration (CI) and continuous delivery. In this chapter, we will cover the CI/CD implementation of an Android application with Jenkins. We will use Pipeline as a YAML to create CI/CD pipeline. We will distribute an application to the App Center to a specific Group. We will also provide some

valuable " *Notes* " related to DevOps, culture, challenges, market trends, and so on for better understanding.

Structure

In this chapter, we will discuss the following topics:

Introduction

Multi-stage YAML pipeline for Android app

Continuous integration

Understand how to perform Lint analysis for Android application

Execute Unit Tests and calculate code coverage continuous delivery for Android app

YAML pipeline script for Android app

Objectives

This chapter will introduce how to implement CI and continuous delivery pipeline for Android App using YAML. After studying this unit, you should be able to:

Understand how to perform code analysis for Android application.

Execute unit tests and publish test results in Jenkins dashboard.

Calculate code coverage.

Verify build quality for line coverage.

Introduction

The organization has selected one application as a pilot while other 15 early adopter Android applications are waiting in the queue. We have a responsibility to create CI/CD pipeline using Pipeline as a YAML in Jenkins. Following is the list of tools, and deliverables that will be integrated into the pipeline:

pipeline:

pipeline:

pipeline:

pipeline: pipeline: pipeline:

pipeline:

pipeline:

pipeline:

pipeline: pipeline:

pipeline: pipeline: pipeline: pipeline: pipeline:

pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline:

pipeline: pipeline:

pipeline:

Table 5.1: Tools and deliverables

In the next section, we will create Pipeline as a YAML for sample application in a step-by-step manner.

Multi-stage CI/CD pipeline for Android app

In this chapter, we will cover CI/CD for sample Android applications. [Figure 5.1](#) is the big picture for CI/CD of Android app:

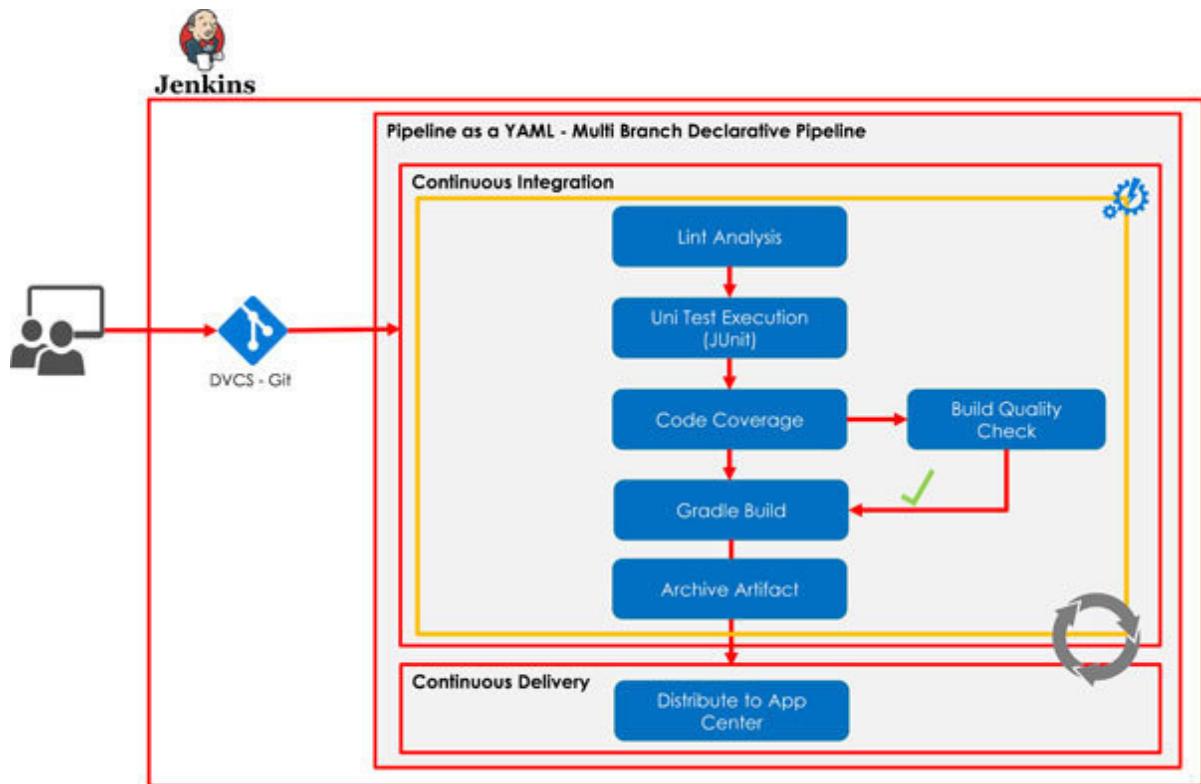


Figure 5.1: Big picture for CI/CD of Android App

Let us create the pipeline for a sample Android application using YAML. We will configure Lint analysis using the Lint tool, unit test execution, and code coverage calculation and report for sample Ionic Cordova application.

Continuous integration for Android App

CI is a prevalent DevOps practice that requires a development team to commit the Android App code into a shared distributed version control system several times a day based on a bug fix or feature implementation. Each commit is verified by an automated code analysis, test execution, build, and code coverage, allowing teams to detect, and locate issues early in the development cycle. Following are tasks that can be executed in the CI implementation.

Maintain repository in Distributed Version Control

Automate unit tests

Automate the build

Every commit should trigger a build in Azure DevOps

Keep the build fast – use parallel jobs using multiple agents

Calculate code coverage

Maintain standard reporting for unit tests and code coverage

Publish artifacts such as Lint Analysis, Unit Test results, APK file(s), Code Coverage results

Configure Build Quality checks for Android App

[*Understand how to perform Lint analysis for Android application*](#)

In the CI phase, we will start with the Lint Analysis. Lint analysis highlights structural problems in the code. It helps to increase the quality, reliability, maintainability, and efficiency of the code.

**Use lint.xml for verifying source files with specific lint rules.
Keep this file in the root folder of the Project. More details are available at**

Android Lint tool is a static code analysis tool that verifies source files for potential issues such as bugs or vulnerabilities for following the attributes available in [*Figure*](#)

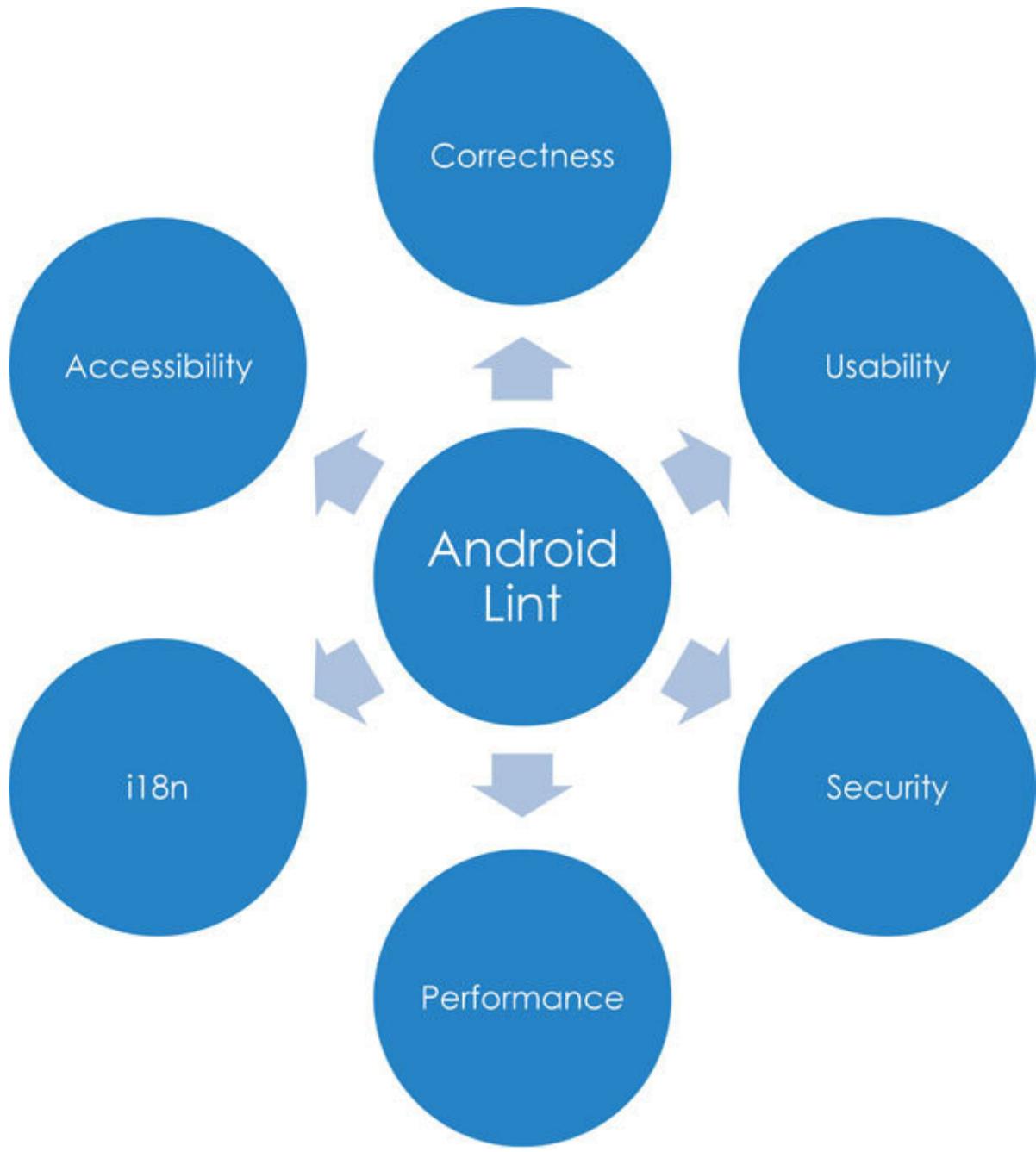


Figure 5.2: Lint output category

Lint tool can be utilized by command line or in Android Studio. We will use the command-line option in the Jenkins

pipeline for Lint analysis.

Use CI for Android applications when the team comprises of more than two developers to build, execute unit test cases, perform lint analysis, and to calculate code coverage for better quality.

If our Android project has multiple build variants, then we can execute the lint task for a specific build variant. To find out the tasks available for the execution based on variants execute gradle tasks or gradlew tasks based on the operating systems:

> Task :tasks

Tasks runnable from root project

Android tasks

androidDependencies - Displays the Android dependencies of the project.

signingReport - Displays the signing info for the base and test modules

sourceSets - Prints out all the source sets defined in this project.

Build tasks

assemble - Assembles the outputs of this project.
assembleAndroidTest - Assembles all the Test applications.
build - Assembles and tests this project.
buildDependents - Assembles and tests this project and all projects that depend on it.
buildNeeded - Assembles and tests this project and all projects it depends on.
bundle - Assemble bundles for all the variants.
clean - Deletes the build directory.
cleanBuildCache - Deletes the build cache directory.
compileDebugAndroidTestSources
compileDebugSources
compileDebugUnitTestSources

compileReleaseSources
compileReleaseUnitTestSources
Build Setup tasks

init - Initializes a new Gradle build.
wrapper - Generates Gradle wrapper files.
Cleanup tasks

lintFix - Runs lint on all variants and applies any safe suggestions to the source code.

Help tasks

buildEnvironment - Displays all buildscript dependencies declared in root project 'Android-App-Sample_jenkins'.

components - Displays the components produced by root project 'Android-App-Sample_jenkins'. [incubating]
dependencies - Displays all dependencies declared in root project 'Android-App-Sample_jenkins'.
dependencyInsight - Displays the insight into a specific dependency in root project 'Android-App-Sample_jenkins'.
dependentComponents - Displays the dependent components of components in root project 'Android-App-Sample_jenkins'.
[incubating]
help - Displays a help message.
model - Displays the configuration model of root project 'Android-App-Sample_jenkins'. [incubating]
projects - Displays the sub-projects of root project 'Android-App-Sample_jenkins'.

properties - Displays the properties of root project 'Android-App-Sample_jenkins'.
tasks - Displays the tasks runnable from root project 'Android-App-Sample_jenkins' (some of the displayed tasks may belong to subprojects).

Install tasks

installDebug - Installs the Debug build.
installDebugAndroidTest - Installs the android (on device) tests for the Debug build.
uninstallAll - Uninstall all applications.
uninstallDebug - Uninstalls the Debug build.

`uninstallDebugAndroidTest` - Uninstalls the android (on device) tests for the Debug build.

`uninstallRelease` - Uninstalls the Release build.

Reporting tasks

`combinedTestReportDebug` - Generate Jacoco coverage reports after running debug tests.

`jacocoTestReportDebug` - Generate Jacoco coverage reports after running debug tests.

`jacocoTestReportRelease` - Generate Jacoco coverage reports after running release tests.

Verification tasks

`check` - Runs all checks.

`connectedAndroidTest` - Installs and runs instrumentation tests for all flavors on connected devices.

`connectedCheck` - Runs all device checks on currently connected devices.

`connectedDebugAndroidTest` - Installs and runs the tests for debug on connected devices.

`createDebugCoverageReport` - Creates test coverage reports for the debug variant.

`deviceAndroidTest` - Installs and runs instrumentation tests using all Device Providers.

`deviceCheck` - Runs all device checks using Device Providers and Test Servers.

`lint` - Runs lint on all variants.

lintDebug - Runs lint on the Debug build.

lintRelease - Runs lint on the Release build.

lintVitalRelease - Runs lint on just the fatal issues in the release build.

.

.

test - Run unit tests for all variants.

testDebugUnitTest - Run unit tests for the debug build.

testReleaseUnitTest - Run unit tests for the release build.

Rules

Pattern: clean: Cleans the output files of a task.

Pattern: build: Assembles the artifacts of a configuration.

Pattern: upload: Assembles and uploads the artifacts belonging to a configuration.

The pipeline configuration provides repository URL in the Branch sources section of pipeline job:

The screenshot shows the Jenkins PipelineAsYAML configuration page for the 'SampleAndroidApp' project. The 'Branch Sources' tab is selected. Under the 'Git' section, the 'Project Repository' is set to 'https://...', and there is a dropdown for 'Credentials'. The 'Behaviors' section contains a 'Discover branches' button and an 'Add' button. The 'Property strategy' is set to 'All branches get the same properties'. Below this, there is a 'Build Configuration' section with 'Mode' set to 'By Jenkinsfile As Yaml' and 'Script Path' set to 'Jenkinsfile.yml'. At the bottom are 'Save' and 'Apply' buttons.

Figure 5.3: Branch sources

Click on the status link to get the list of branches available where Jenkinsfile is available or branches that are filtered based on wildcards.

The screenshot shows the main Jenkins project page for 'SampleAndroidApp'. On the left is a sidebar with various Jenkins-related links like Up, Status, Configure, Scan Multibranch Pipeline Now, Scan Multibranch Pipeline Log, Multibranch Pipeline Events, Delete Multibranch Pipeline, People, Build History, Project Relationship, Check File Fingerprint, Move, Open Blue Ocean, Rename, Config Files, Pipeline Syntax, and Credentials. The main content area displays the project name 'SampleAndroidApp' and a table titled 'Branches (1)'. The table has columns: S, W, Name, Last Success, Last Failure, Last Duration, and Fav. It shows one entry: 'master' with a blue icon, last success at 4 days 1 hr - #21, last failure at 4 days 1 hr - #20, and a duration of 8 min 57 sec. A legend at the bottom right indicates three types of feeds: 'Atom feed for all', 'Atom feed for failures', and 'Atom feed for just latest builds'. There is also a 'Disable Multibranch Pipeline' button.

Figure 5.4: Jenkins pipeline – branches

Following is the YAML script for lint analysis. Here, we will perform lint analysis using gradle lint command:

pipeline:

tools:

```
gradle: "gradle-5.4.1-all"
```

```
jdk: "JDK8"
```

environment:

```
ANDROID_HOME:  
'C:\\\\Users\\\\Mitesh\\\\AppData\\\\Local\\\\Android\\\\Sdk'
```

```
JAVA_HOME: 'C:\\\\Program Files\\\\Java\\\\jdk1.8.0_111'
```

agent:

```
label: 'master'
```

stages:

- stage: 'Lint Analysis'

steps:

- bat 'gradle --version'
- tool 'JDK8'
- tool 'gradle-5.4.1-all'
- bat 'gradlew.bat clean lint'
- androidLint()

Once the Android lint task is executed, the next task is to configure task to publish lint results in the dashboard. If this option is not available in Blue Ocean then install Android Lint plugin. Plugin details are available at

Verify the logs that are stage-wise and steps added in each stage. It is easier to navigate through logs in Blue Ocean rather than using the traditional Jenkins console:



Figure 5.5: Lint analysis stage

Integrate quality gates in the pipelines/orchestration and pipeline should be promoted to a specific environment based on the quality gate clearance only. For example, code should be promoted to QA only if static code analysis quality gate is cleared or code coverage is above the threshold.

It is recommended to have quality gates in the end-to-end process. Quality has to be part of the culture and routine for continuous improvement and continuous innovation.

Thought-provoking exercise: how do you differentiate quality gate and quality assurance as both are in place to improve quality.

Go to the traditional Jenkins dashboard and verify the Lint trend chart available after the pipeline execution:

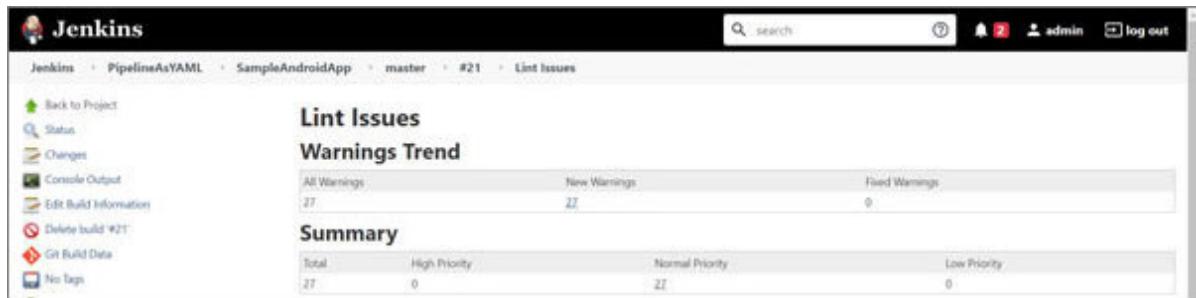


Figure 5.6: Lint Issues

Click on the Lint Issues link available in the left sidebar to get details on the issues with priority.

We can configure lint options in the build.gradle file as well. Let us consider a scenario where we want to abort the execution process if there is any lint errors exist:

```
android {
```

```
...
```

```
lintOptions {  
    abortOnError false  
  
    // if true, only report errors.  
  
    ignoreWarnings true  
  
}  
  
}
```

In the next section, we will configure unit test execution and calculating code coverage.

Execute unit tests and calculate code coverage

We will use the Gradle plugin that generates JaCoCo reports from an Android Gradle Project:

<https://github.com/vanniktech/gradle-android-junit-jacoco-plugin>

Make the following changes in the root

```
buildscript {  
  
    // Define versions in a single place  
  
    ext {  
  
        .  
        .  
        .  
  
        jacoco_version = '0.8.2'  
    }  
}
```

```
}
```

```
repositories {
```

```
    google()
```

```
    jcenter()
```

```
}
```

```
dependencies {
```

```
.
```

```
.
```

```
.
```

```
    classpath "com.vanniktech:gradle-android-junit-jacoco-
    plugin:0.14.0"
```

```
}
```

```
}
```

Add the following section in

```
apply plugin: "com.vanniktech.android.junit.jacoco"

junitJacoco {

    jacocoVersion = "$jacoco_version" // type String

    ignoreProjects = ["markdown"] // type String array

    excludes = [
        '**/*Test.*',
        '**/*Activity.*',
        '**/*Fragment.*',
        '**/AutoValue_*.',
        '**/*JavascriptBridge.class',
        '**/R.class',
```

```
'**/R$*.class',
'**/Manifest.*',
'android/**/*.*',
'**/BuildConfig.*',
'**/*$ViewBinder.*',
'**/*$ViewInjector.*'

]

includeNoLocationClasses = true // type boolean

includeInstrumentationCoverageInMergedReport = false //
type boolean

}

android {

buildTypes {
```

```
    debug {  
        testCoverageEnabled true  
    }  
}  
}
```

Following is the YAML script for Unit test execution stage:

- stage: 'Unit Tests'

steps:

- bat 'gradlew.bat jacocoTestReportDebug'
- junit
- "publishCoverage(adapters:
[jacocoAdapter('app/build/reports/jacoco/debug/jacoco.xml')],
sourceFileResolver: sourceFiles('NEVER_STORE'))"

Create a new stage for unit tests and configure steps for Gradle and JDK along with the batch script to run gradle task as we configured it in previous section. Execute gradle task Execute gradle tasks command and find the available tasks for Jacoco and code coverage.

Once code coverage is executed along with unit tests archive test reports using step **Archive Junit-formatted test**

Verify all the steps configured for unit tests and code coverage. Execute the pipeline:



Figure 5.7: Unit tests logs

Once the pipeline is executed successfully, we can see logs as per steps configured in the pipeline in specific section and hence navigating through logs is easy compared to the traditional way of execution:

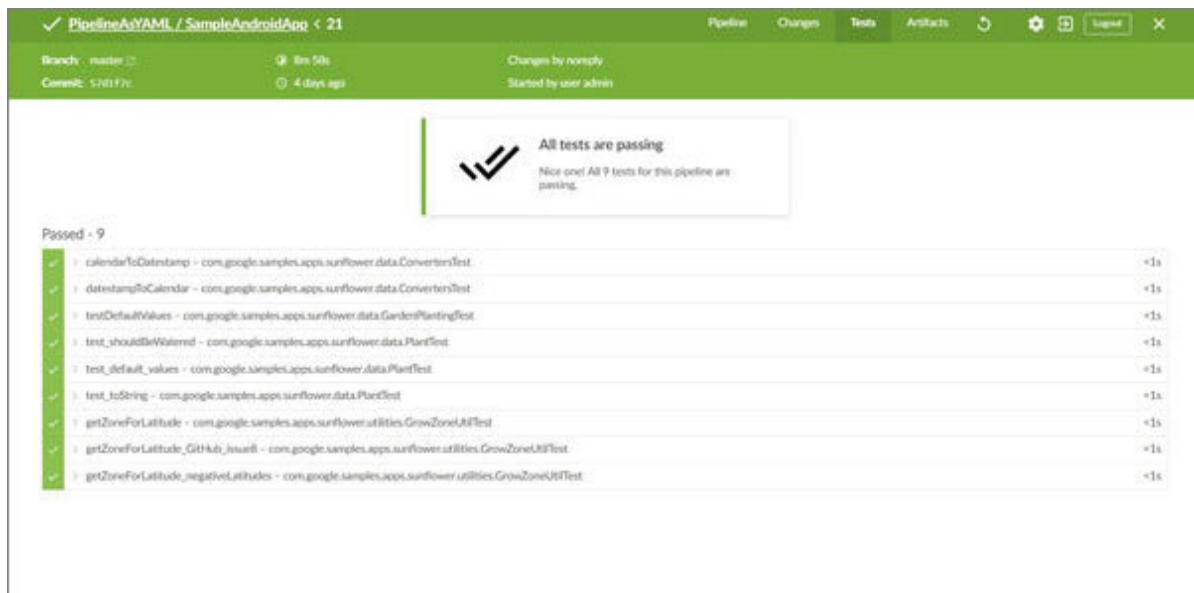


Figure 5.8: Unit tests result

The test results analyzer plugin shows the build result history of test-class, test-class, and test-package in a tabular tree format. The plugin can be used to enable the **junit** or **TestNG** (in case of TestNG) feature of Jenkins.

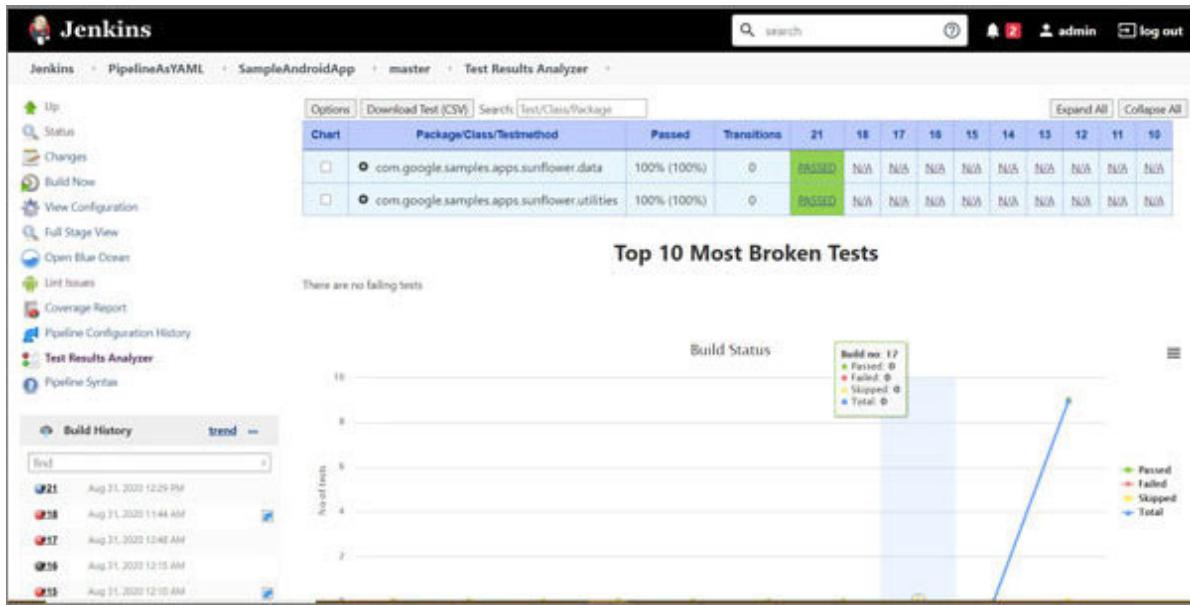


Figure 5.9: Test results analyzer

DevOps practices implementation helps organizations to accelerate time to market with quality. It empowers teams to automate application lifecycle activities.

The test results analyzer plugin allows users to filter the results based on passed, failed, and skipped status. We can click on the **Test Results Analyzer** link on the left side of the pipeline job. It also supports generation of graphs such as line charts, pie charts, and bar charts:

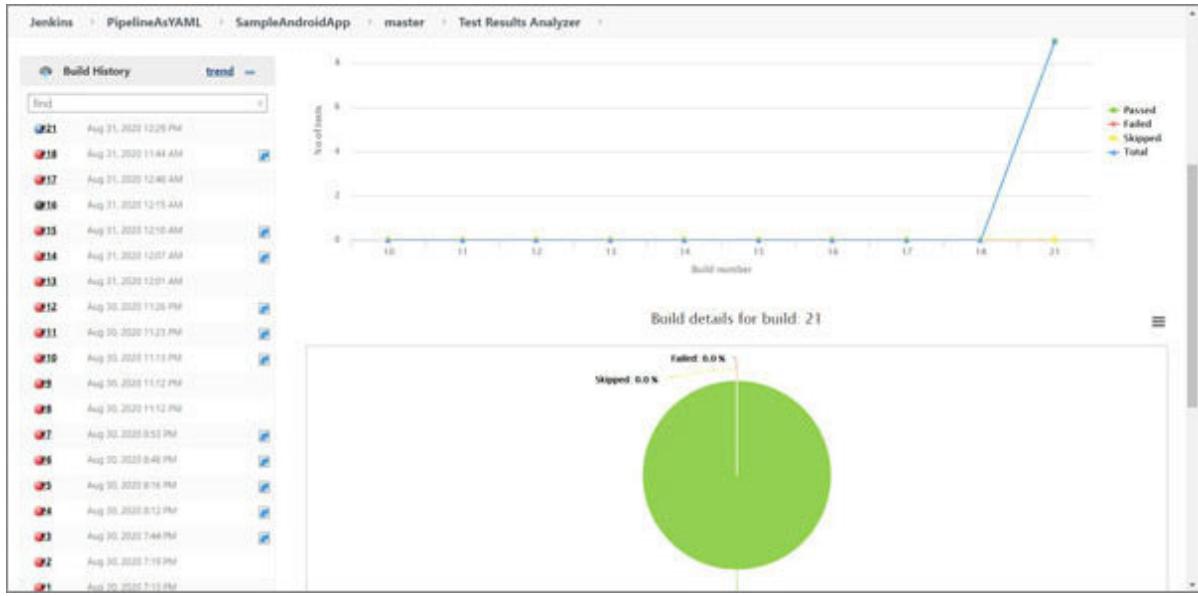


Figure 5.10: Test results analyzer – pie chart

Code coverage API serves as API to integrate and publish multiple coverage report types such as JaCoCo, Cobertura (Cobertura Plugin), and OpenCover. It also supports the pipeline configuration, you can generate pipeline code in Jenkins Snippet Generator.

Verify the coverage report in Jenkins dashboard for the pipeline job:

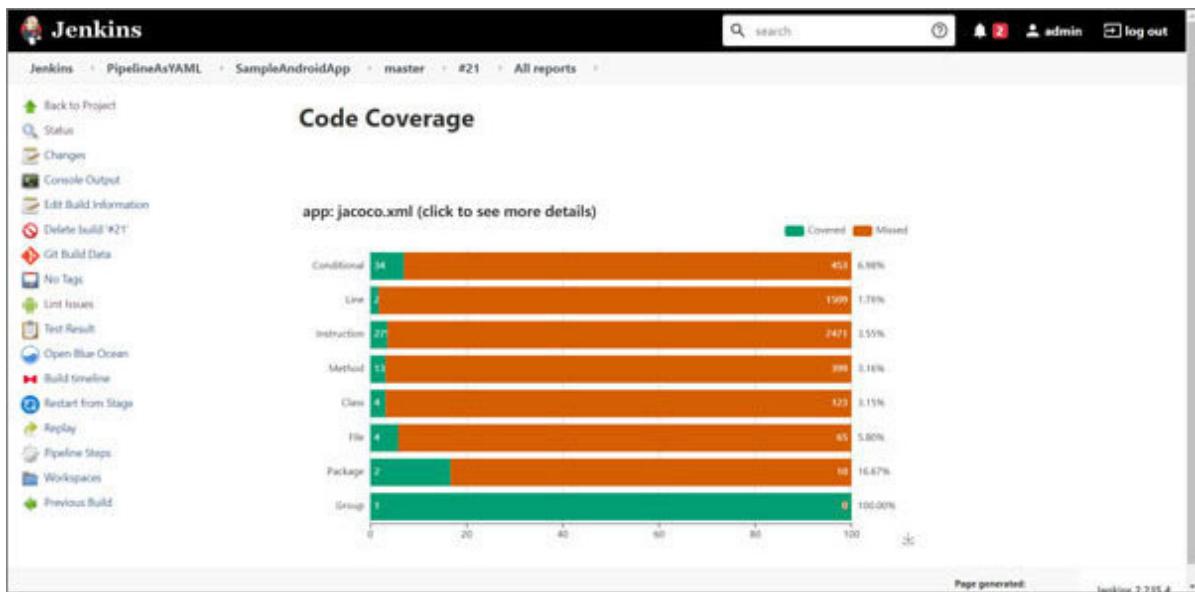


Figure 5.11: Code coverage

Following is the YAML script for Build stage of sample Android application:

- stage: 'Build'

- steps:

- bat 'gradlew.bat assemble'

- "archiveArtifacts"

There are multiple gradle tasks available to build the project:

Build tasks

assemble - Assembles the outputs of this project.

assembleAndroidTest - Assembles all the Test applications.

build - Assembles and tests this project.

buildDependents - Assembles and tests this project and all projects that depend on it.

buildNeeded - Assembles and tests this project and all projects it depends on.

bundle - Assemble bundles for all the variants.

clean - Deletes the build directory.

cleanBuildCache - Deletes the build cache directory.

compileDebugAndroidTestSources

compileDebugSources

compileDebugUnitTestSources

compileReleaseSources

compileReleaseUnitTestSources

We have already executed Lint analysis and test execution hence we will run assemble task. If we want to exclude any gradle task execution then we can use the command-line argument which excludes any task.

For example: gradle build -x test

Create a new stage for and configure steps for Gradle and JDK along with batch script to run gradle task as we configured it in the previous section. Execute the pipeline. Verify the build stage logs:

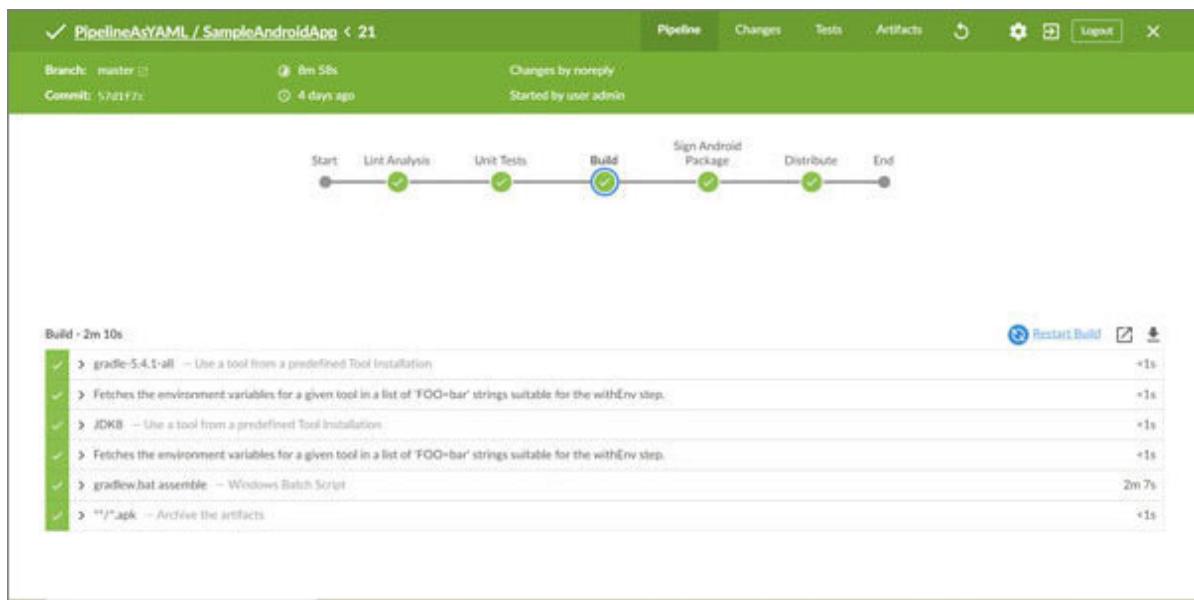


Figure 5.12: Build stage logs

Now we can see that Android Packages files) are available. app-debug.apk file can be used to test and debug apk file quickly:

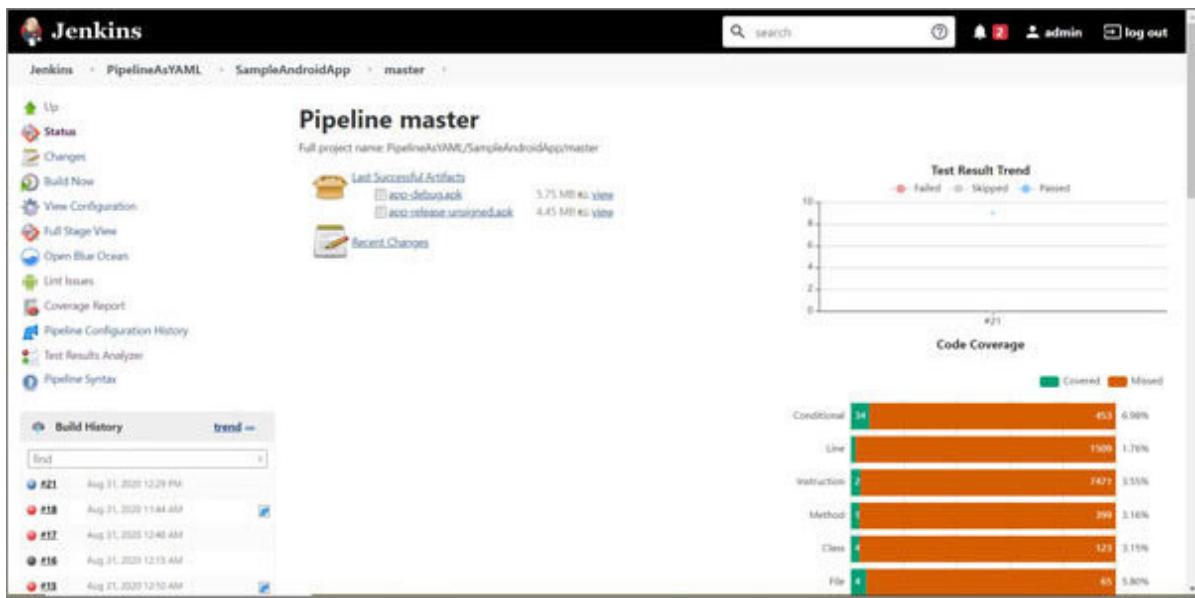


Figure 5.13: Android pipeline status

Click on the Artifact section to verify the artifacts available to download:

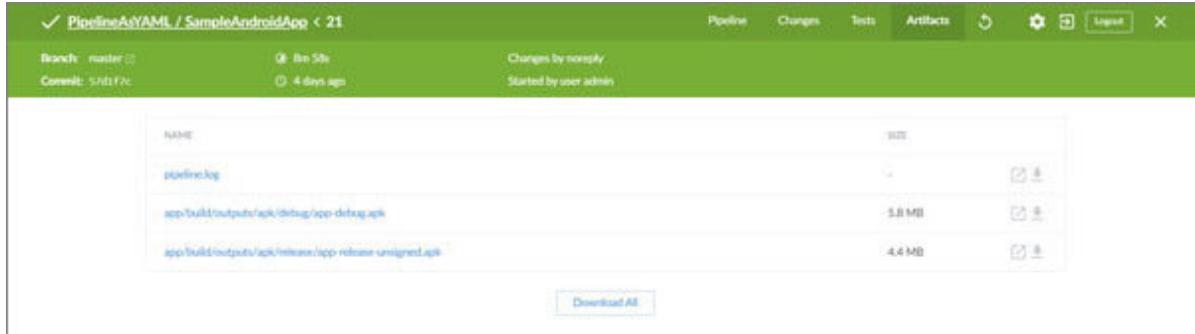


Figure 5.14: Artifacts

Build a release APK after signing the unsigned app with a private key:

The screenshot shows the Jenkins interface for a pipeline named 'PipelineAsYAML' under the project 'SampleAndroidApp'. The build number is master #21, and the build time is Aug 31, 2020 12:29:28 PM. The left sidebar lists various pipeline steps: Back to Project, Status, Changes, Console Output, Edit Build Information, Delete build #21, Git Build Data, No Tags, Lint Issues, Test Result, Open Blue Ocean, Build timeline, Restart from Stage, Replay, Pipeline Steps, Workspaces, and Previous Build. The main content area displays the following details:

- Build Artifacts:** Includes 'apk-debug.apk' (3.75 MB) and 'apk-release-unsigned.apk' (4.45 MB).
- Changes:** Shows a single commit: '1. Update build.gradle (details / git blame)'.
- Started by user:** admin
- Revision:** 57d1f7cd895d4e854c2dd9254718e456c54a87, origin/master.
- Lint:** 27 issues (27 new issues).
- Test Result:** (no failures).
- Coverage Reports:** Report 100% (Group: 100%, Package: 17%, File: 6%, Class: 3%, Method: 3%, Instruction: 4%, Line: 2%, Conditional: 7%).

A 'Keep this build forever' button is located in the top right corner, along with a link to add a description and a note indicating the build started 4 days 1 hr ago and took 8 min 57 sec.

Figure 5.15: Pipeline status

Pipeline steps in the left sidebar help you to be aware about the time each step and stage takes and its status:

The screenshot shows the Jenkins interface for a pipeline named 'PipelineAsYAML' under the project 'SampleAndroidApp'. The build number is master #22, and the stage is Pipeline Steps. The left sidebar lists various pipeline steps: Back to Project, Status, Changes, Console Output, Edit Build Information, Delete build #22, Git Build Data, No Tags, Lint Issues, Test Result, Pipeline Configuration History, Open Blue Ocean, Build timeline, Restart from Stage, Replay, Pipeline Steps, and Workspaces. The main content area displays a table of pipeline steps with columns: Step, Arguments, and Status.

Step	Arguments	Status
Start of Pipeline - (10 min in block)		Success
Allocate node : Start - (10 min in block)	master	Success
Allocate node : Body : Start - (10 min in block)		Success
Stage : Start - (17 sec in block)	Declarative: Checkout SCM	Success
Declarative: Checkout SCM - (16 sec in block)		Success
Check out from version control - (16 sec in self)		Success
Set environment variables : Start - (10 min in block)	GIT_BRANCH, GIT_COMMIT, GIT_PREVIOUS_COMMIT, GIT_PREVIOUS_SUCCESSFUL_COMMIT, GIT_URL	Success
Set environment variables : Body : Start - (10 min in block)		Success
Set environment variables : Start - (10 min in block)	JAVA_HOME, ANDROID_HOME	Success
Set environment variables : Body : Start - (10 min in block)		Success
Stage : Start - (1.2 sec in block)	Declarative: Tool Install	Success
Declarative: Tool Install - (0.98 sec in block)		Success

Figure 5.16: Pipeline steps

Jenkins timeline is a Jenkins plugin that allows users to gain details about the execution of pipeline builds. Clicking **Build** pipeline link in the left sidebar will open up the build timeline in a new tab. The timeline can be opened during a build for an incremental break-down of the job or after a build is finished for an overview of old jobs:

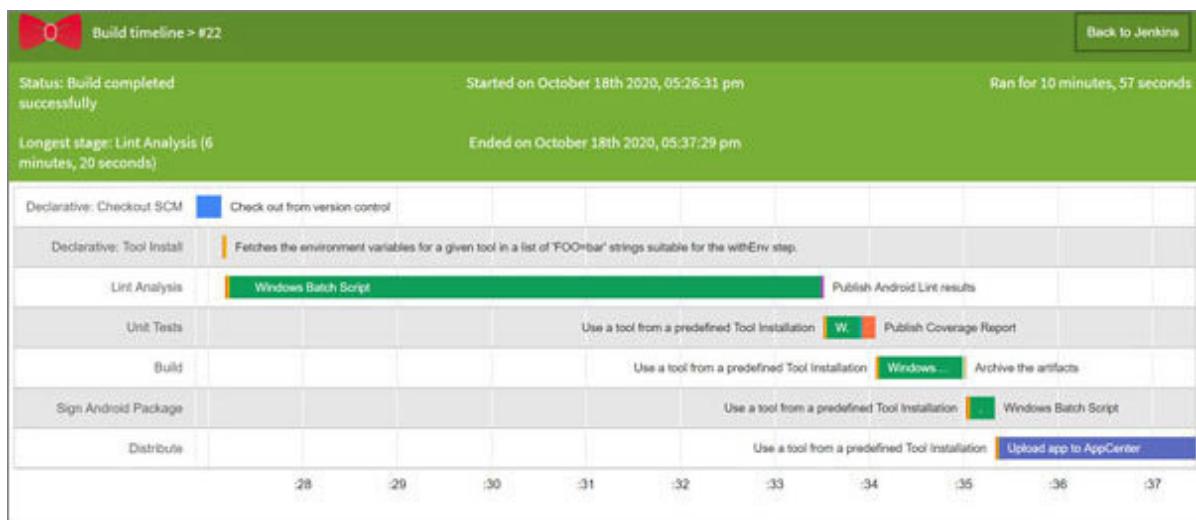


Figure 5.17: Build timeline

DevOps practices implementation tends to be more successful while it is implemented in a phase-wise manner and not with a Big bang approach. The organizations evaluate benefits for

pilot projects and then consider them for critical applications based on the level of confidence.

In the next section, we will configure YAML script for distributing Android package to App center.

Continuous delivery for Android app

App Center is used to distribute mobile applications to the QA team. It is also used to build, test, and distributed Android, iOS, and Windows applications. Go to Log in with the appropriate method. Provide Username for login.

Following is the YAML script for distributing the Android package to App center:

- stage: 'Sign Android Package'

steps:

- bat '"C:/Program Files
(x86)/Java/jdk1.8.0_192/bin/jarsigner.exe" -verbose -keystore
"C:/jenkinsbook.keystore" -storepass jenkinsbook -signedjar
"app\\build\\outputs\\apk\\release\\app-release-signed.apk"
"app/build/outputs/apk/release/app-release-unsigned.apk"
jenkinsbook'

- bat '"C:/Program Files
(x86)/Java/jdk1.8.0_192/bin/jarsigner.exe" -verify

```
"app/build/outputs/apk/release/app-release-signed.apk""
```

- stage: 'Distribute'

steps:

```
- "appCenter(apiToken:  
'a2197ddab495db8adae6d4f23e5918f77028d476', ownerName:  
'xxxxxx.xxxxxx-xxxxxxxx.com', appName: 'SampleAndroidApp',  
pathToApp: 'app/build/outputs/apk/release/app-release-  
signed.apk', distributionGroups: 'QA-Distribution', releaseNotes:  
'Bug Fixed - Ticket 2020.07.20')"
```

DevOps is the default!

DevOps is mainstream!

DevOps is a new normal!

DevOps is conventional!

DevOps is change!

DevOps is a transformation!

DevOps is just a beginning for most organizations!

Let us add a new app in App Center by clicking on **Add**

Select **App** and

Click on **Add new**

To integrate App center in Jenkins, we will need API token. Let us create it first.

Let us go to **Account Settings** of App Center.

Scroll down to the **Account Settings** page.

Click on **New API**

Provide description and access.

Click on **Add new API**

Copy the **API**

Execute the pipeline and verify the Sign Android Package stage logs:



Figure 5.18: Sign android package

Happy teams are productive teams. DevOps practices increase the productivity of project teams by trying to automate manual activities and providing visibility and transparency.

Execute the pipeline and verify the distribute stage logs:



Figure 5.19: Distribute package to App Center

Mindset and culture change are more important factors in organizations rather than tools and technology. Highly collaborative teams with effective communication help to change the culture and implement DevOps practices effectively.

Go to App Center and verify the **Releases** section. The package will be available for download.

Click on the **Groups** and select the **QA-Distribution Testing** group that we created. App is available in that section too.

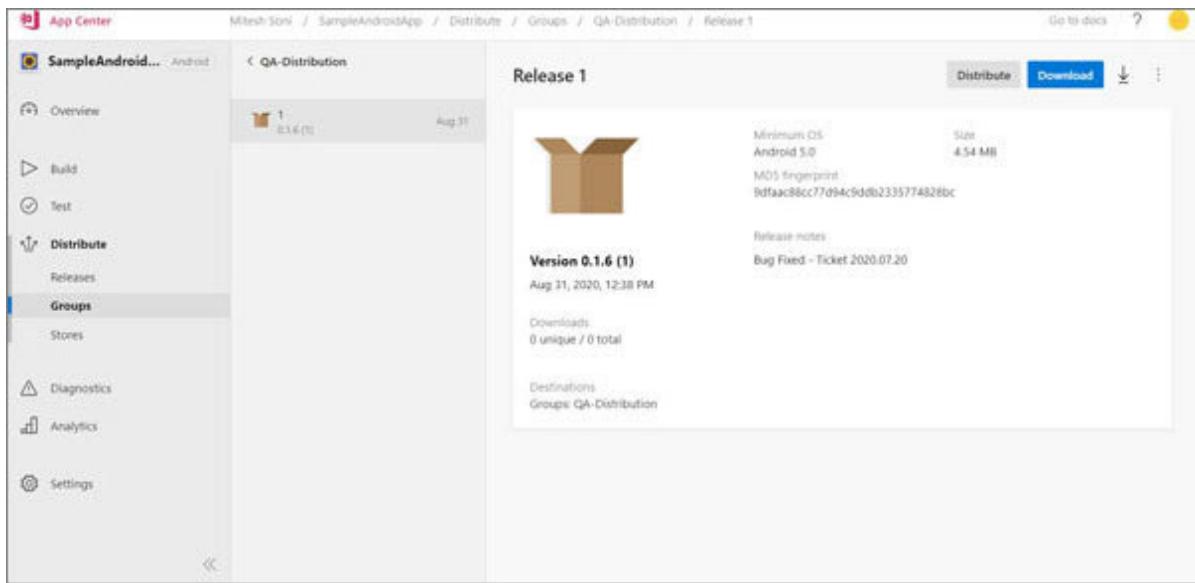


Figure 5.20: App Center dashboard

DevOps practices implementations help to have a continuous focus to increase organizational/resource productivity using people, processes, and tools.

In the next section, we will verify the entire YAML script covered in this chapter.

[YAML pipeline script for Android app](#)

Following is the complete YAML script to create CI/CD pipeline for sample Android application:

pipeline:

agent:

label: 'master'

stages:

- stage: 'Lint Analysis'

steps:

- bat 'gradle --version'

- tool 'JDK8'

- tool 'gradle-5.4.1-all'
 - bat 'gradlew.bat clean lint'
 - androidLint()
- stage: 'Unit Tests'

steps:

- bat 'gradlew.bat jacocoTestReportDebug'
 - junit '**/testDebugUnitTest/*.xml'
 - "publishCoverage(adapters:
[jacocoAdapter('app/build/reports/jacoco/debug/jacoco.xml')],
sourceFileResolver: sourceFiles('NEVER_STORE'))"
- stage: 'Build'

steps:

- bat 'gradlew.bat assemble'
- "archiveArtifacts '**/*.apk'"

- stage: 'Sign Android Package'

steps:

- bat '"C:/Program Files
(x86)/Java/jdk1.8.0_192/bin/jarsigner.exe" -verbose -keystore
"C:/jenkinsbook.keystore" -storepass jenkinsbook -signedjar
"app\\build\\outputs\\apk\\release\\app-release-signed.apk"
"app/build/outputs/apk/release/app-release-unsigned.apk"
jenkinsbook'

- bat '"C:/Program Files
(x86)/Java/jdk1.8.0_192/bin/jarsigner.exe" -verify
"app/build/outputs/apk/release/app-release-signed.apk"'

- stage: 'Distribute'

steps:

- "appCenter(apiToken:
'a2197ddab495db8adae6d4f23e5918f77028d476', ownerName:
'xxxxx.xxxx-xxxxx.com', appName: 'SampleAndroidApp',
pathToApp: 'app/build/outputs/apk/release/app-release-

```
signed.apk', distributionGroups: 'QA-Distribution', releaseNotes:  
'Bug Fixed - Ticket 2020.07.20')"
```

tools:

```
gradle: "gradle-5.4.1-all"
```

```
jdk: "JDK8"
```

environment:

```
ANDROID_HOME: 'C:\\Program Files  
(x86)\\Android\\android-sdk'
```

```
JAVA_HOME: 'C:\\Program Files\\Java\\jdk1.8.0_111'
```

Faster time to market, quality of outcome, and productivity gains are essential and drivers for DevOps movement.

Done!

Conclusion

CI and continuous delivery implementation are more or less similar in mobile and web applications. However, tools and distribution may change. It is essential to have some knowledge of Android, how it is uploaded to the play store, how different environment variables are utilized, which secure files or keystore files are utilized, and so on.

In the next chapter, we will use YAML syntax to create pipeline for Angular application.

Multiple choice questions

Tests results are required to calculate code coverage.

True

False

To get code coverage, the following configuration is not required.

```
debug {  
    testCoverageEnabled true  
}
```

True

False

Use Gradle tasks command to find out available tasks for execution.

True

False

Code Coverage for Android app written in Java and Kotlin will have a different configuration.

True

False

Answer

a

b

a

a

Questions

What is the purpose of unit tests?

Android app can be developed in how many languages?

What is code coverage, and why is it important?

Which build tool is used for Android applications generally?

CHAPTER 6

Building CI/CD Pipeline with YAML for Angular Application

"Believe in yourself! Have faith in your abilities! Without a humble but reasonable confidence in your own powers you cannot be successful or happy."

—*Norman Vincent Peale*

The financial service giant needs effective and faster application lifecycle management to cope up with the competition in the market. Higher management wants to migrate from older technologies to newer technologies. Angular is the preferred choice. Angular 8 is a client-side TypeScript-based framework that is used to create dynamic web applications and mobile applications. Angular has the following features or advantages: such as new compiler – ivy rendering engine, deep linking routing, Restful APIs, lazy loading, builders APIs, and service workers and web worker building. The need of an hour is to implement continuous integration and continuous delivery. The leadership team wants to adopt phase-wise implementation of DevOps practices to remove frictions between development and operations teams.

We can deploy an application to Docker containers or in the Microsoft Azure Cloud environment. In this book, we will cover the CI/CD implementation of an Angular application with Jenkins using YAML pipeline.

Structure

In this chapter, we will discuss the following topics:

Multi-stage YAML pipeline for Angular App

Continuous integration

Junit and Cobertura configuration in karma.conf.js

Lint, unit tests, and code coverage configuration in
Package.json

Configure unit tests and code coverage in multi-stage pipeline

End to end test execution

NPM audit

Continuous delivery

Deploy Angular app to Azure App Services

Objectives

This chapter will introduce how to implement continuous integration and continuous delivery pipeline for Angular application using YAML. After studying this unit, you should be able to:

Understand how to perform Lint analysis for Angular application

Execute unit tests

Calculate code coverage

Verify build quality

Introduction

The organization has selected one application as a pilot while other five early adopter Angular applications are waiting in the queue. We have a responsibility to create CI/CD pipeline using Pipeline as a YAML in Jenkins. Following is the list of tools, and deliverables that will be integrated into the pipeline:

pipeline:

pipeline:

pipeline:

pipeline: pipeline: pipeline:

pipeline:

pipeline:

pipeline:

pipeline:

pipeline: pipeline: pipeline: pipeline: pipeline:

pipeline: pipeline: pipeline: pipeline: pipeline: pipeline: pipeline:
pipeline:

Table 6.1: Tools and deliverables

In the next section, we will create Pipeline as a YAML for sample application in a step-by-step manner.

Multi-stage CI/CD pipeline for Angular app

In this chapter, we will cover CI/CD for sample Angular application. We will try to achieve the following in this chapter:

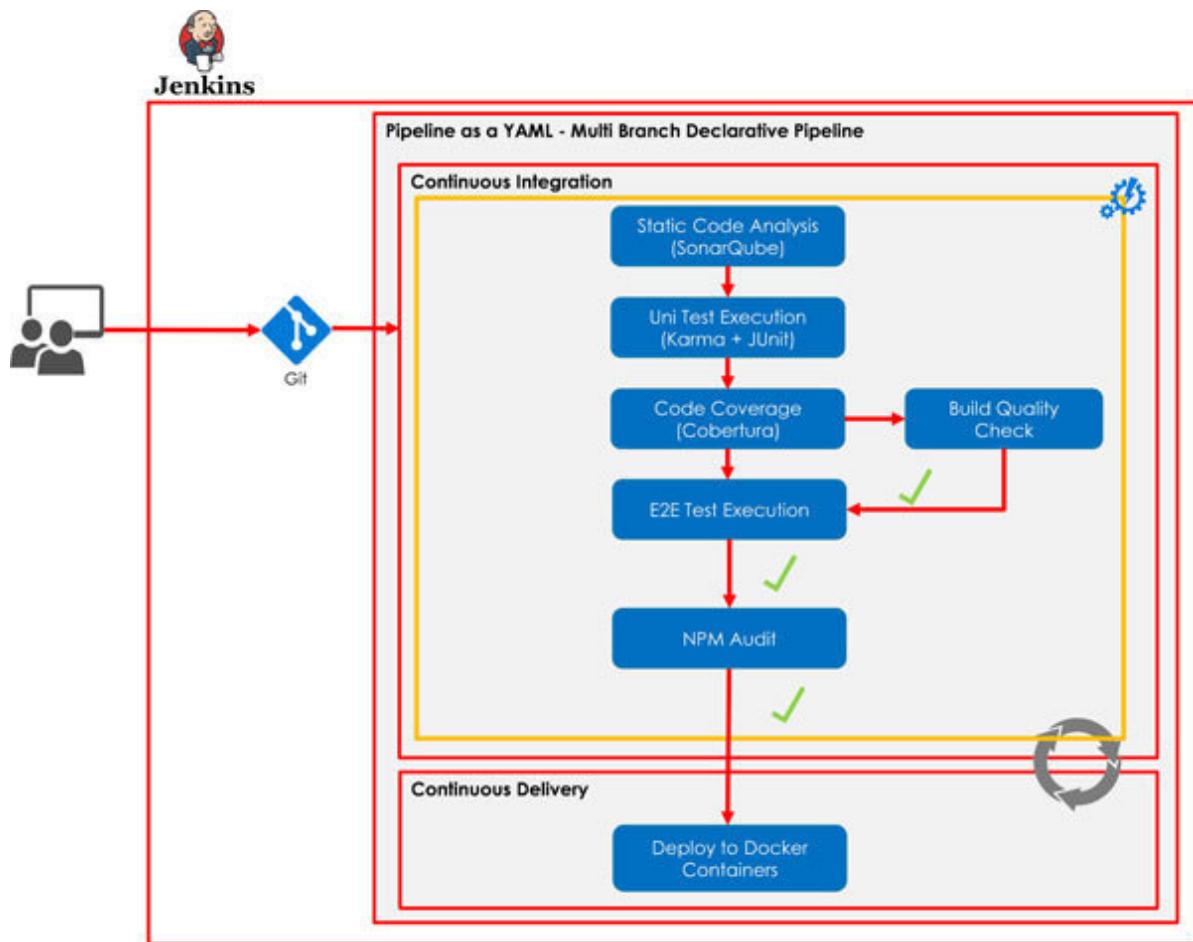


Figure 6.1: Big Picture for CI/CD of Angular App

Let us create a pipeline for a sample Angular application using YAML. We will configure static code analysis/Lint analysis using SonarQube/Lint tools, unit test execution, and code coverage calculation and report for sample Angular application.

Let us create a pipeline for a sample Angular application using Blue Ocean.

Install NodeJs from



Figure 6.2: Download NodeJS

Verify the version of Node and NPM in the command prompt.

Microsoft Windows [Version 10.0.18362.836]

(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Mitesh>npm

Usage: npm

where is one of:

access, adduser, audit, bin, bugs, c, cache, ci, cit,

clean-install, clean-install-test, completion, config,

create, ddp, dedupe, deprecate, dist-tag, docs, doctor,

edit, explore, fund, get, help, help-search, hook, I, init,

install, install-ci-test, install-test, it, link, list, ln,

login, logout, ls, org, outdated, owner, pack, ping, prefix,

profile, prune, publish, rb, rebuild, repo, restart, root,

run, run-script, s, se, search, set, shrinkwrap, star,
stars, start, stop, t, team, test, token, tst, un,
uninstall, unpublish, unstar, up, update, v, version, view,

whoami

npm -h quick help on

npm -l display full usage info

npm help search for help on

npm help npm involved overview

Specify configs in the ini-formatted file:

C:\Users\Mitesh\.npmrc

or on the command line via: npm --key value

Config info can be viewed via: npm help config

npm@6.14.4 C:\Program Files\nodejs\node_modules\npm

```
C:\Users\Mitesh>npm -version
```

6.14.4

```
C:\Users\Mitesh>node --version
```

v12.18.0

In the next section, we will configure continuous integration for the Angular app.

Continuous integration for Angular App

In this section, we will create a pipeline for sample Angular applications by using sample code available in the Git repository.

Continuous improvement and continuous innovation are the main priorities of all organizations to remain competitive in a dynamic market. DevOps practices are an enabler of digital transformation as they enable high quality and faster time to market by complementing Agile principles and cloud resources.

Junit and Cobertura configuration in karma.conf.js

In this section, we want to configure Lint analysis, unit test execution, code coverage, build quality checks, execute end-to-end tests, and execute audit command.

Import a sample Angular application code from GitHub with unit tests.

Let us do a few configurations in file

Add required plugins for Junit (unit testing) and Cobertura (code coverage) output.

```
plugins: [
```

```
    require('karma-jasmine'),  
  
    require('karma-chrome-launcher'),  
  
    require('karma-mocha-reporter'),
```

```
require('karma-coverage-istanbul-reporter'),  
  
require('karma-junit-reporter'),  
  
require('@angular-devkit/build-angular/plugins/karma')  
  
],
```

Add Cobertura in reports key section to

```
coverageIstanbulReporter: {  
  
  dir: require('path').join(__dirname, './coverage'),  
  
  reports: ['html', 'lcovonly','cobertura'],  
  
  fixWebpackSourcePaths: true  
  
},  
  
reporters: ['progress', 'mocha', 'junit'],
```

Add JunitReporter configuration details in karma.conf.js if it is not available in the file:

```
junitReporter: {  
  
    outputDir: "",  
  
    outputFile: undefined,  
  
    suite: "",  
  
    useBrowserName: true,  
  
    nameFormatter: undefined,  
  
    classNameFormatter: undefined,  
  
    properties: {}  
  
},
```

Configure Headless Browser so test cases can be executed:

```
autoWatch: true,
```

```
singleRun: false,  
  
browsers: ['ChromeHeadlessNoSandbox'],  
  
customLaunchers: {  
  
  ChromeHeadlessNoSandbox: {  
  
    base: 'ChromeHeadless',  
  
    flags: ['--no-sandbox']  
  
  }  
  
},
```

In the next section, we will configure scripts in

Lint, unit tests, and code coverage configuration in Package.json

Let us configure scripts in

Configure package.json with scripts:

```
"lint": "tslint --project tslint.json",  
  
"test": "ng test angular-sample-app --code-coverage --no-watch",  
  
"e2e": "ng e2e",
```

To execute lint analysis, following command needs to be executed: npm install && npm install --save-dev

Linting "angular-sample-app"...

All files pass linting.

Linting "ngx-example-library"...

All files pass linting.

Once we have configured the required settings, we need to configure them in

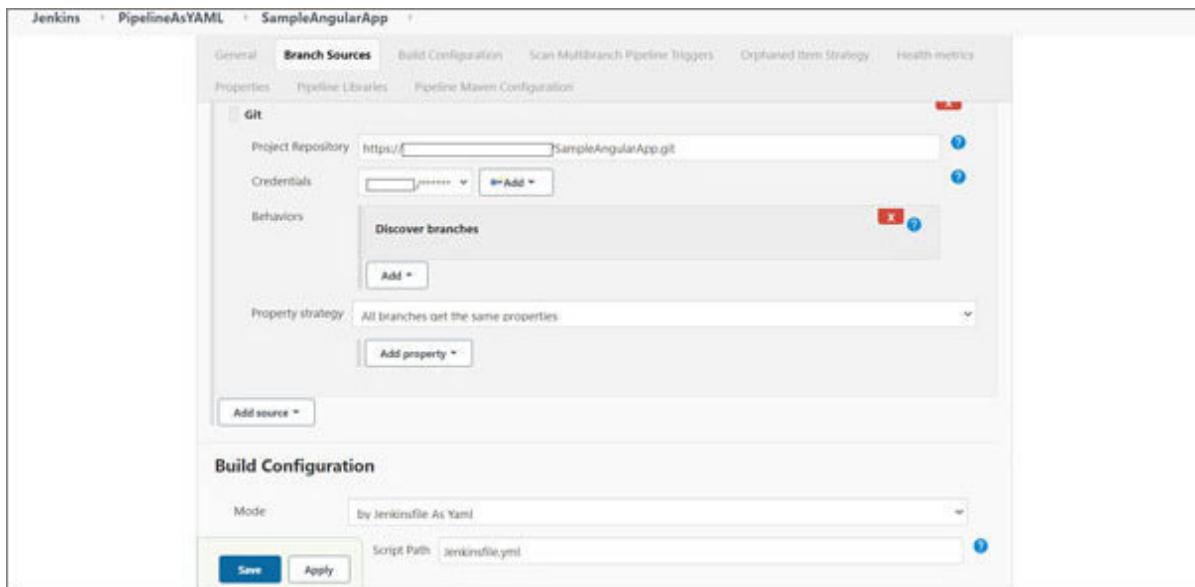


Figure 6.3: Project repository

Our master branch contains Jenkinfile and hence it is detected by Jenkins and the master branch is available in the Jenkins dashboard:

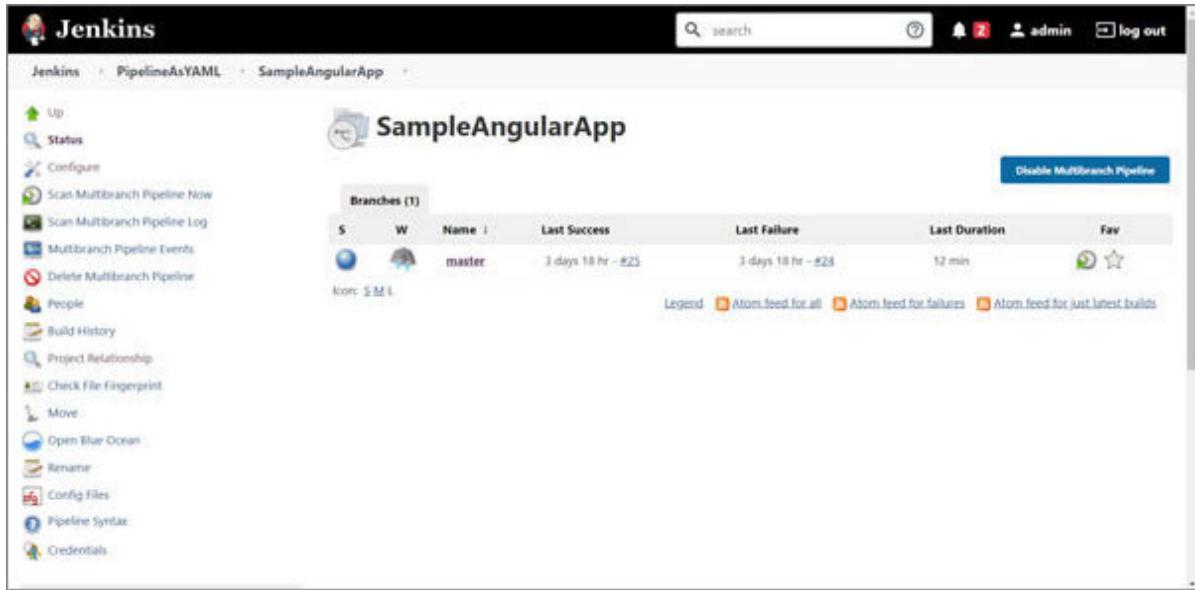


Figure 6.4: Branches in Jenkins Dashboard

Following is the YAML script that helps to analyze Angular code using SonarQube.

pipeline:

agent:

any:

stages:

- stage: 'Static Code Analysis'

```
agent:
```

```
label: 'master'
```

```
steps:
```

```
- bat 'F:\\1.DevOps\\2020\\sonar-scanner-3.2.0.1227-windows\\bin\\sonar-scanner.bat' -  
Dsonar.host.url=http://localhost:9000/ -  
Dsonar.login=d39153de87276ec525e77b1601b94e7ddffd2f23 -  
Dsonar.projectVersion=1.0 -Dsonar.projectKey=sample-angular-app  
-Dsonar.sources=src'
```

Configure SonarQube token in the script directly for proof of concept. It is ideal to configure SonarQube as a credential secret and utilize it in the YAML pipeline.

Agile and DevOps are not the same. Agile and DevOps complement each other. The agile approach is a driver for DevOps practices implementation.

Execute the Pipeline and verify the logs for the stage Static Code Analysis in console log or Blue ocean dashboard:

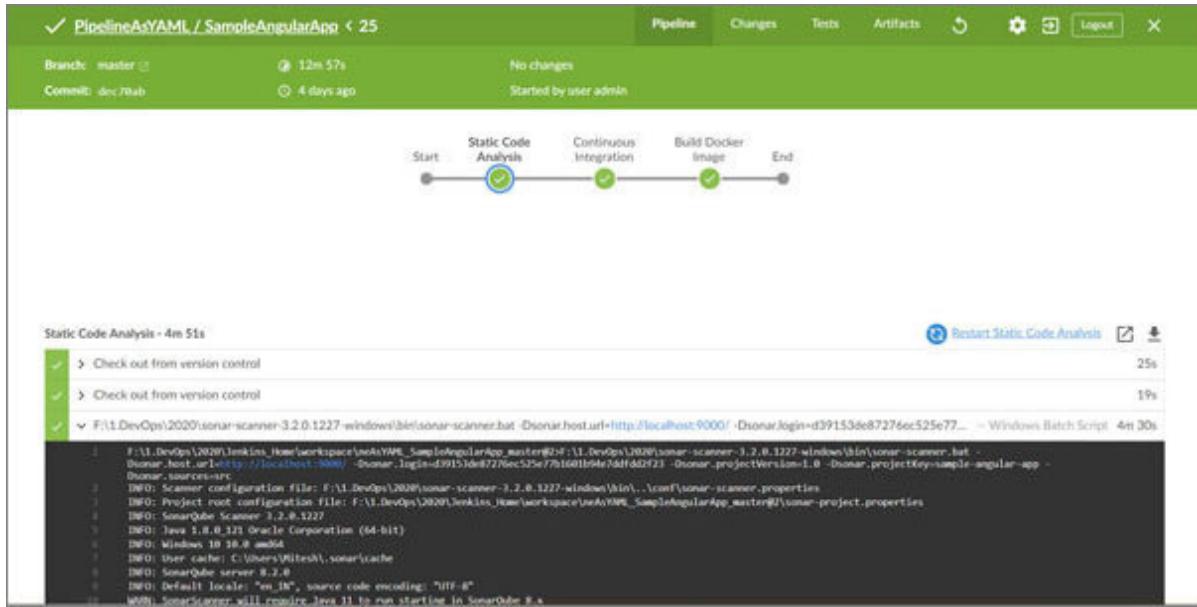


Figure 6.5: Logs for the stage Static Code Analysis

The console log will provide URL for the SonarQube dashboard or login with SonarQube and get details about Angular application from Projects:

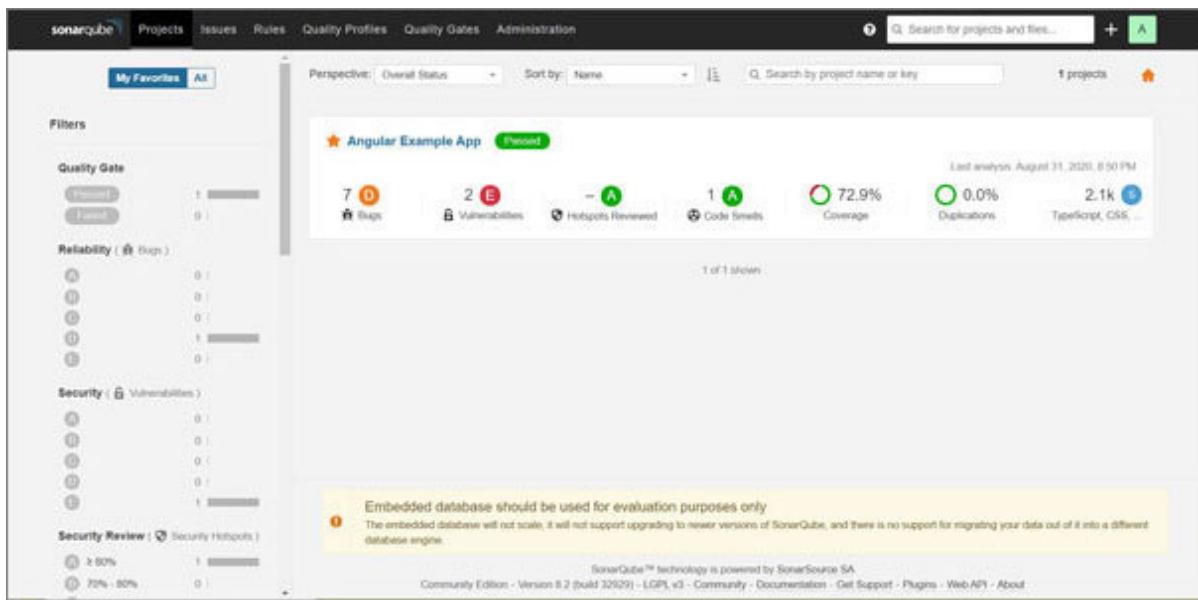


Figure 6.6: Angular Code Analysis in SonarQube

Automation doesn't imply DevOps. Automation is one enabler. People, processes, and tools are critical for cultural transformation

Click on the project name to get more details about code analysis of an application in SonarQube along with Quality Gate result:

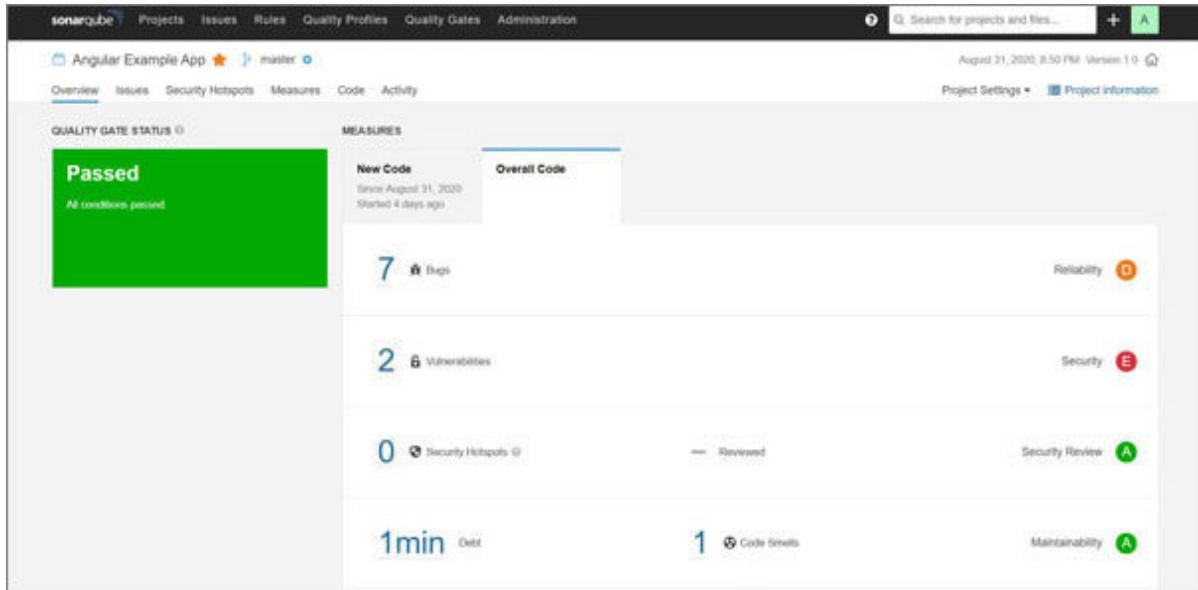


Figure 6.7: Detailed Angular Code Analysis in SonarQube

Click on the code link in the SonarQube dashboard to get detail on the code directory, line of code, bugs, vulnerabilities, Code smells, Security Hotspots, Coverage, and duplications related information:

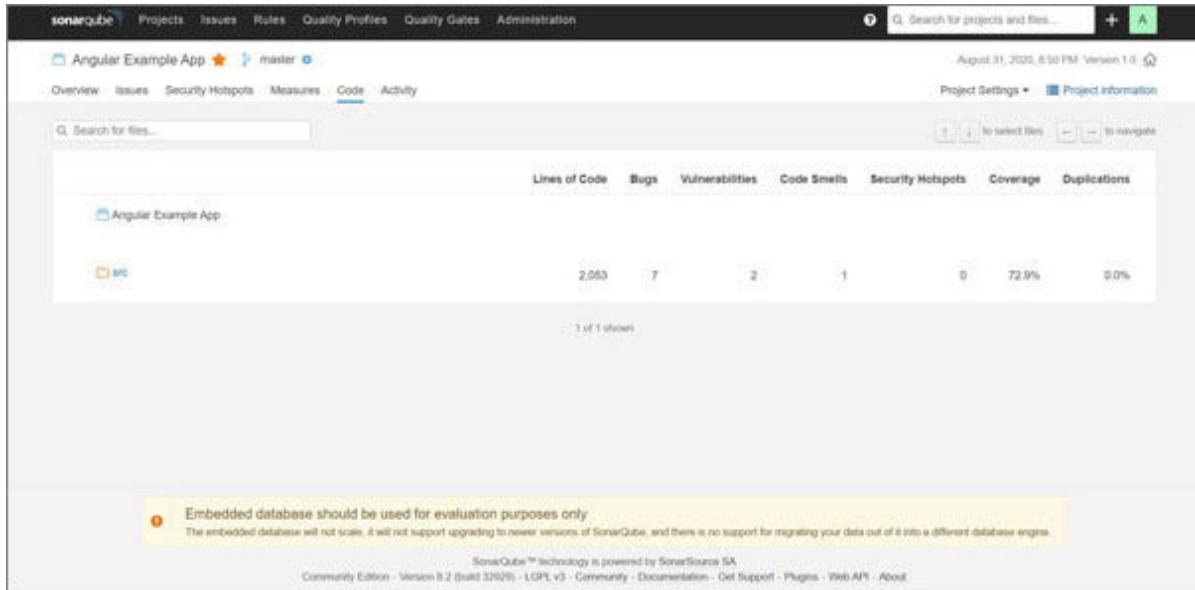


Figure 6.8: SonarQube – code

Once static code analysis is done, let us see YAML script block for the continuous integration stage that will perform npm install, install junit related dependencies, execute unit tests, publish unit test results, publish code coverage, build distribution content or artifacts, compress the folder and stash the zip file so another stage can unstash and utilize the file. Following is a YAML script to perform continuous integration:

- stage: 'Continuous Integration'

agent:

label: 'master'

steps:

- bat 'npm install'
- # - bat 'npm run lint > lint.txt'
- bat 'npm install karma-junit-reporter --save-dev && npm run test'
- junit 'TESTS-*.xml'
- "publishCoverage(adapters:[coberturaAdapter('coverage/cobertura-coverage.xml')], sourceFileResolver: sourceFiles('NEVER_STORE'))"
- bat 'npm run build:prod:en'
- "zip(dir: 'dist/browser', zipFile: 'browser.zip')"
- "stash/includes: 'browser.zip', name: 'dist'"

Execute the pipeline and verify the logs in Blue Ocean dashboard:



Figure 6.9: CI stage in Blue Ocean

DevOps is useful for all kinds of organizations and applicable to most of applications. However, a detailed assessment is required before DevOps practices implementation.

Click on the **Tests** link to get results of all unit tests:

✓ PipelineAsYAML / SampleAngularApp < 25

Branch: master | 12m 57s | No changes
Commit: alex/Haha | 4 days ago | Started by user admin

All tests are passing
Nice one! All 43 tests for this pipeline are passing.

Passed - 43

- ✓ > HeroRemoveComponent should create the component - HeadlessChrome_84_0_4147_(Windows_10_0_0)HeroRemoveComponent
- ✓ > AppComponent should create the app - HeadlessChrome_84_0_4147_(Windows_10_0_0)AppComponent
- ✓ > AppComponent should check browser features - HeadlessChrome_84_0_4147_(Windows_10_0_0)AppComponent
- ✓ > HeroesListPageComponent should create new hero error - HeadlessChrome_84_0_4147_(Windows_10_0_0)HeroesListPageComponent
- ✓ > HeroesListPageComponent should like a hero - HeadlessChrome_84_0_4147_(Windows_10_0_0)HeroesListPageComponent
- ✓ > HeroesListPageComponent should delete a hero - HeadlessChrome_84_0_4147_(Windows_10_0_0)HeroesListPageComponent
- ✓ > HeroesListPageComponent should create component and load heroes - HeadlessChrome_84_0_4147_(Windows_10_0_0)HeroesListPageComponent
- ✓ > HeroesListPageComponent should create new hero success - HeadlessChrome_84_0_4147_(Windows_10_0_0)HeroesListPageComponent
- ✓ > HeroDetailPage should create hero detail component - HeadlessChrome_84_0_4147_(Windows_10_0_0)HeroDetailPage
- ✓ > HeroCardComponent should create - HeadlessChrome_84_0_4147_(Windows_10_0_0)HeroCardComponent
- ✓ > HeroCardComponent should like a hero - HeadlessChrome_84_0_4147_(Windows_10_0_0)HeroCardComponent

Figure 6.10: Unit test results in Blue Ocean

In the pipeline, click on the Test Results Analyzer to get more details on the unit tests in Jenkins dashboard:

Jenkins | PipelineAsYAML | SampleAngularApp | master | Test Results Analyzer

Chart	Package/Class/Testmethod	Passed	Transitions	25	24	23	22	21	20	19	18	17	16
Pass	HeadlessChrome_84_0_4147_(Windows_10_0_0)	100% (100%)	0	PASSED	PASSED	PASSED	N/A						

Top 10 Most Broken Tests

There are no failing tests.

Build Status

Build History

Trend

Builds: Aug 31, 2020 8:48 PM, Aug 31, 2020 8:24 PM, Aug 31, 2020 7:47 PM, Aug 31, 2020 7:46 PM, Aug 31, 2020 7:44 PM, Aug 31, 2020 7:41 PM

Legend: Passed (green), Failed (red), Skipped (yellow), Total (blue)

Figure 6.11: Test Results Analyzer

Scroll down to get more details on all kinds of results in **Build** details for all Pie chart:

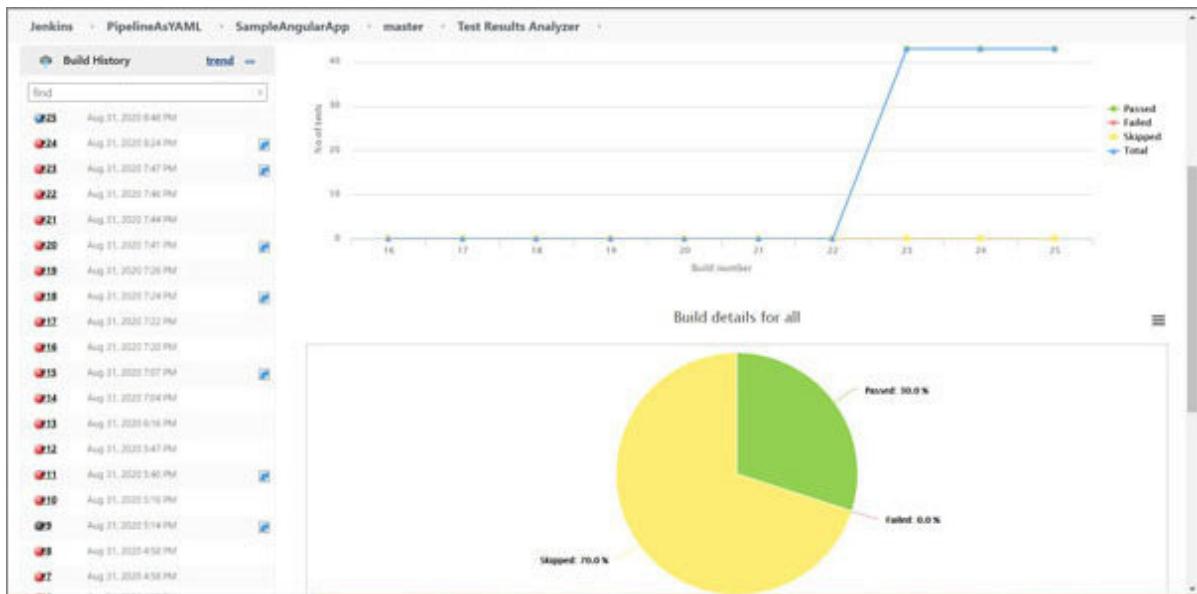


Figure 6.12: Test Results Analyzer – Build details for all

Click on the **Coverage Report** link on the left sidebar to get more details on Unit test code coverage:

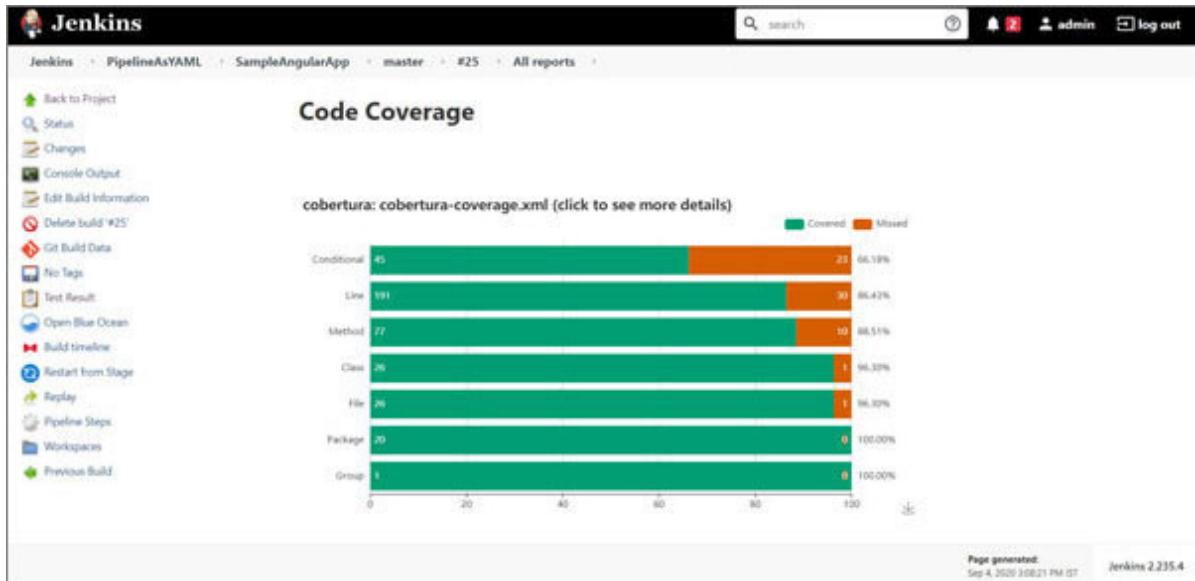


Figure 6.13: Unit test code coverage

Different toolsets that can be used in Jenkins:

SCM: Git, SVN, CVS, BitBucket

CI: Azure DevOps, Jenkins, TeamCity, Atlassian Bamboo, TFS

Infrastructure Automation: Public Cloud, Private Cloud, Docker, Kubernetes

Configuration Management: Chef, Puppet, Salt, Ansible

Testing: Appium, Selenium, Apache JMeter, Load Runner

Monitoring: New Relic, Nagios, Zabbix

App Server: Tomcat, Apache, IIS, Nginx, Weblogic, Websphere, JBoss

Artifact Repository: Artifactory, Nexus

Build Tools: Maven, Ant, Gradle, MS Build

Click on the **Status** link available in the Jenkins dashboard to get details about Test results and Coverage reports for a specific build:

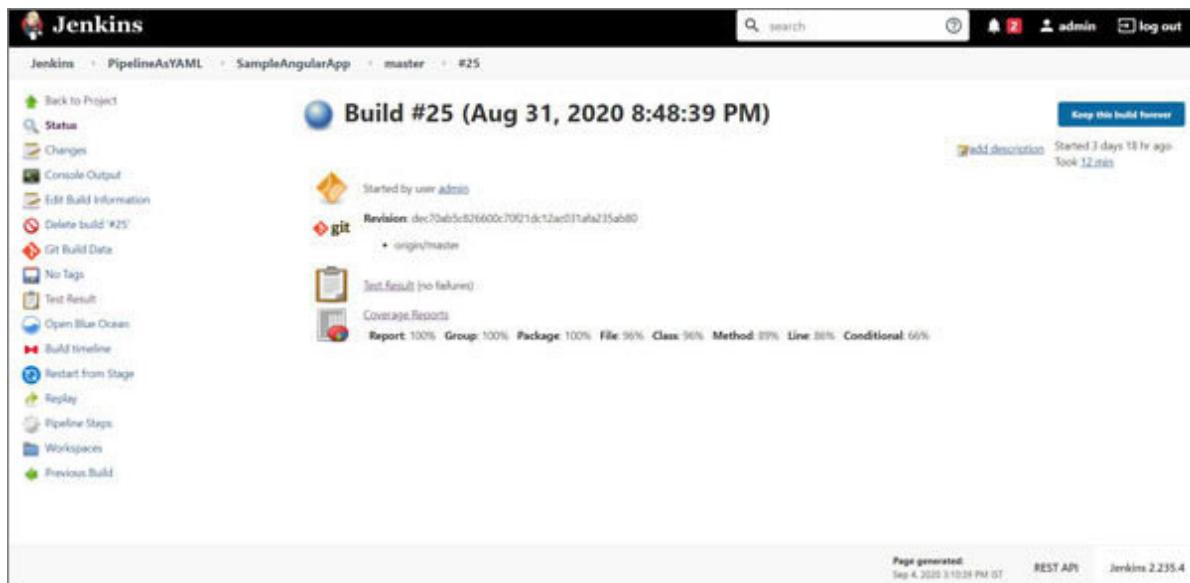


Figure 6.14: Status of the Pipeline execution

Hence, we have the YAML script for code analysis and continuous integration of sample Angular application. In the next section, we will verify YAML script for continuous delivery that is creating a docker image and deploying instance from the image to access sample angular application.

Continuous delivery for Angular app

Following is the stage to build docker image. We have configured a specific agent where docker is already installed. We unstash the distribution folder and provide permissions to the file.

Next task helps to unzip the file and then build the image using docker build command:

```
- stage: 'Build Docker Image'
```

```
agent:
```

```
label: 'docker'
```

```
tools:
```

```
git: 'CentOSGIT'
```

```
steps:
```

- sh 'docker version'
- unstash 'dist'
- sh 'chmod 755 browser.zip'
- "unzip(dir: 'dist/browser', zipFile: 'browser.zip')"
- sh 'docker build . -t yaml/angular-sample'

Cron expression consists of 5 fields separated by Tab or whitespace:

MINUTE HOUR DOM MONTH DOW

DOW DOW DOW DOW DOW

DOW DOW DOW DOW DOW DOW

DOW DOW DOW DOW DOW DOW

DOW DOW DOW

DOW DOW DOW DOW DOW DOW DOW DOW DOW DOW

DOW DOW

Examples:

* * * * *

Execute after every minute

H/30 * * * *

Every 30 Minutes

H(0-29)/10 * * * *

Every ten minutes in the first half of every hour

@yearly, @annually, @monthly, @weekly, @daily, @midnight, and @hourly can be used.

```
triggers {  
  cron '@midnight'  
}
```

Execute the YAML script and verify console log in the Blue Ocean dashboard:



```
Build Docker Image - 15s  
Restart Build Docker Image  
6s  
> Check out from version control  
<1s  
> CentOSGIT — Use a tool from a predefined Tool Installation  
<1s  
> Fetches the environment variables for a given tool in a list of 'FOO=bar' strings suitable for the withEnv step.  
1s  
> docker version — Shell Script  
<1s  
> dist — Restore files previously stashed  
<1s  
> chmod 755 browser.zip — Shell Script  
<1s  
> Extract Zip file  
<1s  
docke build . -t yaml/angular-sample — Shell Script  
1 + docker build . -t yaml/angular-sample  
2 Sending build context to Docker daemon 9.63MB  
3 Step 1/3 : FROM nginx:alpine  
4 --> 6f715d30fc00  
5 Step 2/3 : COPY dist/browser/ /usr/share/nginx/html  
6 --> f2aeef5f529  
7 Step 3/3 : EXPOSE 80  
8 --> Running in 8491190f74b2  
9 Removing intermediate container 8491190f74b2  
10 --> e08930dd71f7  
11 Successfully built e08930dd71f7  
12 Successfully tagged yaml/angular-sample:latest
```

Figure 6.15: Building docker image

Verify the traditional dashboard in Jenkins:

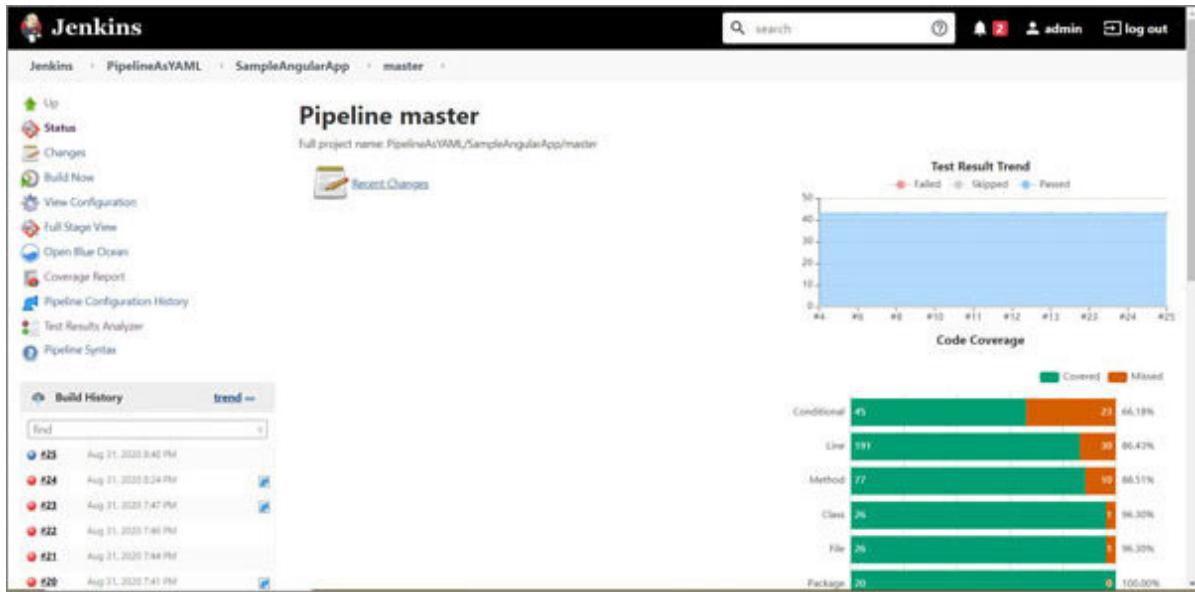


Figure 6.16: Traditional dashboard in Jenkins

Environment variables in Jenkins are available in the pipeline directly, not as steps. Visit http://JENKINS_URL/env-vars.html/

Click on the Full stage view link in the left sidebar to get details on stage execution:

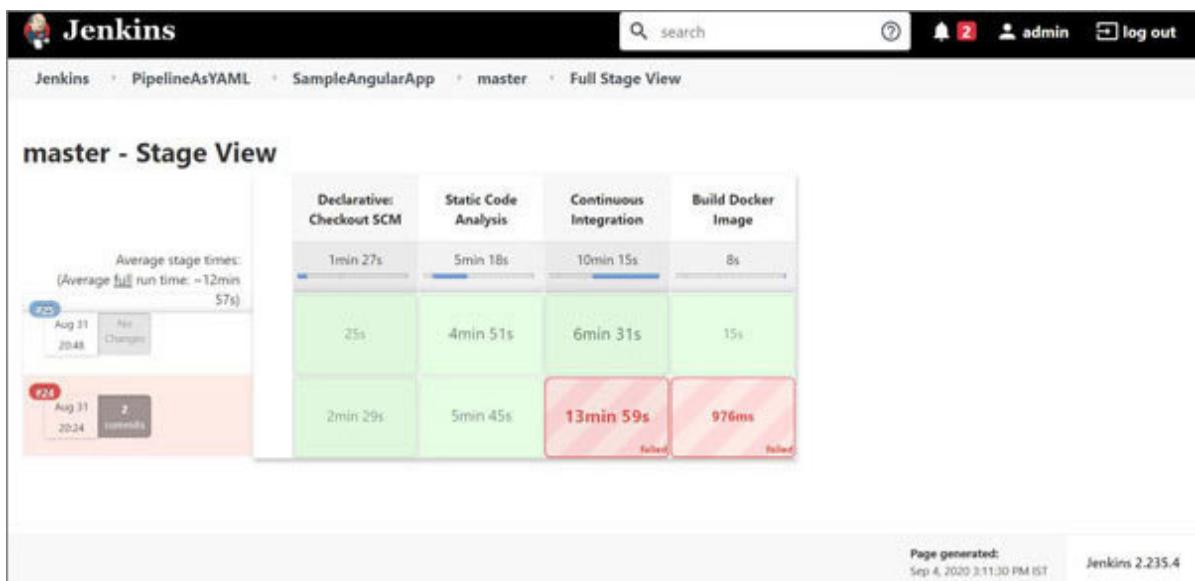


Figure 6.17: Full stage view

To create a docker instance, execute the step - sh 'docker run - p 8080:80 --detach yaml/angular-sample' from the agent where the docker is already installed.

YAML pipeline script for Angular app

Following is the complete YAML script to create CI/CD pipeline for sample Angular application:

pipeline:

agent:

any:

stages:

- stage: 'Static Code Analysis'

agent:

label: 'master'

steps:

```
- bat 'F:\\1.DevOps\\2020\\sonar-scanner-3.2.0.1227-windows\\bin\\sonar-scanner.bat' -  
Dsonar.host.url=http://localhost:9000/ -  
Dsonar.login=d39153de87276ec525e77b1601b94e7ddfdd2f23 -  
Dsonar.projectVersion=1.0 -Dsonar.projectKey=sample-angular-app  
-Dsonar.sources=src'
```

```
- stage: 'Continuous Integration'
```

```
agent:
```

```
label: 'master'
```

```
steps:
```

```
- bat 'npm install'
```

```
#      - bat 'npm run lint > lint.txt'
```

```
      - bat 'npm install karma-junit-reporter --save-dev &&  
npm run test'
```

```
- junit 'TESTS-*.xml'
```

```
- "publishCoverage(adapters:  
[coberturaAdapter('coverage/cobertura-coverage.xml')],  
sourceFileResolver: sourceFiles('NEVER_STORE'))"
```

```
- bat 'npm run build:prod:en'
```

```
- "zip(dir: 'dist/browser', zipFile: 'browser.zip')"
```

```
- "stash/includes: 'browser.zip', name: 'dist')"
```

```
- stage: 'Build Docker Image'
```

agent:

label: 'docker'

tools:

git: 'CentOSGIT'

steps:

```
- sh 'docker version'
```

- unstash 'dist'
- sh 'chmod 755 browser.zip'
- "unzip(dir: 'dist/browser', zipFile: 'browser.zip')"
- sh 'docker build . -t yaml/angular-sample'

Done!

Manage Jenkins section has a new look and feel in Jenkins 2.235.1. It is more user friendly with Categories.

Conclusion

In this chapter, we have configured continuous integration and continuous delivery for an Angular app. We have covered Lint Analysis, unit test execution, code coverage, Build Quality Check, and deployment to Docker container. It helps to create end to end pipeline and maintain quality in a big development team as everything is visible. Code Quality and Coverage helps to find issues at early stage and hence it is less costly in terms of rework.

In the next chapter, we will cover best practices for creating pipelines.

Multiple choice questions

To fix the vulnerabilities identified by npm audit command, use NPM audit fix based on the console log.

True

False

Environment variables in Jenkins are available in the Pipeline directly.

True

False

Answer

a

a

Questions

What is the objective of the Angular?

Explain the significance of scripts in package.json.

Explain the significance of karma-conf.js for unit tests and code coverage calculations.

CHAPTER 7

Pipeline Best Practices

"If you always do what you've always done, you'll always be where you've always been."

— *T.D. Jakes*

Jenkins has evolved a lot based on the evolution of technology and from a user experience perspective.

Following are some of the areas where Jenkins had evolved over the years with time:

Jenkins installation in Docker Container

Jenkins in Azure Kubernetes Services (AKS)

Different types of Jenkins Agents (Docker containers and Kubernetes pods)

Security (authentication and authorization)

Backup (full backup, incremental backup), tools configuration

Environment variable configuration

Jenkins has become a popular choice for the DevOps community as an automation server and not only a continuous integration server.

Over the years, a lot of best practices have been documented, and we are discussing some best practices in this chapter based on our experience.

Structure

In this chapter, we will discuss the following topics:

Best practices

Jenkins installation

Security

Configuration

Pipeline best practices

Objectives

After studying this unit, you should be able to configure and utilize Jenkins in an effective manner. Once best practices are in place, it is easier to maintain and manage Jenkins and focus on CI/CD pipeline rather than managing automation infrastructure.

Best practices

In this chapter, we will discuss Jenkins Best practices that make the life of a Jenkins Engineer easier than usual.

This section describes Jenkins best practices to get a feel of what Jenkins can contribute throughout the Application Lifecycle Management Activities. We will discuss the best practices related to installation, distributed architecture, security, monitoring, backup and restore, and pipelines.

Easy installation with fault tolerance

Let us see how to install Jenkins using Docker and Kubernetes to get it quickly up and running.

Install Jenkins using Docker

How to install Jenkins using Docker containers and use Blue Ocean?

Use the following Docker image available at DockerHub:

<https://hub.docker.com/r/jenkinsci/blueocean/>

Docker pull command: docker pull jenkinsci/blueocean

Execute the following command to create a container:

```
sudo docker run -p 8080:8080 -p 50001:50001 -v  
/home//Desktop/jenkins_home:/home//jenkins_home  
jenkinsci/blueocean
```

The admin password is available in the log, open a browser on <http://localhost:8080>. Complete initial setup wizard and visit <http://localhost:8080/blue>

Visit <https://www.jenkins.io/doc/book/installing/> for more details.

Dockerfile agent in declarative pipeline

Dockerfile agents can help to easily test and build the project and save resources. Following is a content of Dockerfile that exists in our code:

```
// download the code from the repository
```

```
FROM alpine/git as clone
```

```
WORKDIR /app
```

```
RUN git clone
```

```
// Use maven image to copy the directory of application code  
from above
```

```
FROM maven:3.6.3-jdk-8 as build
```

```
WORKDIR /app
```

```
COPY --from=clone /app/sample-java-app/app'
```

Jenkinsfile contains details about the agent and in this case, it is dockerfile:

```
pipeline {  
  
    agent {  
  
        dockerfile true  
  
    }  
  
    stages {  
  
        stage('Build') {  
  
            steps {  
  
                // Execute Batch script if OS flavor is Windows  
  
                sh 'mvn clean package'  
  
                // Publish JUnit Report  
            }  
        }  
    }  
}
```

```
junit '**/target/surefire-reports/TEST-*.xml'  
  
archiveArtifacts(artifacts: 'target/**/*.war',  
onlyIfSuccessful: true, fingerprint: true)
```

```
}
```

```
}
```

```
}
```

```
}
```

Let us observe important logs from Jenkins Console:

Obtained Jenkinsfile from
dfe1102804f5b906ba5b4b49cb966f10e960c1ab

Running in Durability level: MAX_SURVIVABILITY

[Pipeline] Start of Pipeline

[Pipeline] node

Running on Jenkins in /var/jenkins_home/workspace/java-sample-web_docker

[Pipeline] {

[Pipeline] stage

[Pipeline] {(Declarative: Checkout SCM)}

[Pipeline] checkout

using credential o6cddaf1-532c-4d90-a204-0dccfd54aba3

Cloning the remote Git repository

Cloning with configured refspecs honoured and without tags

Cloning repository xxxxxxxxxxxxxxxxxxxxxxxxx.git

> git init *timeout=10*

Fetching upstream changes from xxxxxxxxxxxxxxxxxxxxxxxxx.git

```
> # timeout=10
```

using GIT_ASKPASS to set credentials

```
> git --force --progress -- xxxxxxxxxxxxxxxxxxxxxxxx.git  
+refs/heads/*:refs/remotes/origin/* # timeout=10
```

```
> git config remote.origin.url timeout=10
```

```
> git remote.origin.fetch +refs/heads/*:refs/remotes/origin/* #  
timeout=10
```

```
> git config remote.origin.url timeout=10
```

Fetching without tags

Fetching upstream changes from xxxxxxxxxxxxxxxxxxxxxxxx.git

using GIT_ASKPASS to set credentials

```
> git --force --progress -- xxxxxxxxxxxxxxxxxxxxxxxx.git  
+refs/heads/*:refs/remotes/origin/* # timeout=10
```

Checking out Revision
dfe1102804f5b906ba5b4b49cb966f10e960c1ab (docker)

> git config *timeout=10*

> git checkout -f *timeout=10*

Commit message: "Update Jenkinsfile"

First time build. Skipping changelog.

> # *timeout=10*

[Pipeline]}

[Pipeline] // stage

[Pipeline] withEnv

[Pipeline] {

[Pipeline] stage

[Pipeline] { (Declarative: Agent Setup)

[Pipeline] isUnix

[Pipeline] readFile

[Pipeline] sh

```
+ docker build -t 5dd3ddc4a1806600b5fc1cd2387eoeedff92b008 -  
f Dockerfile .
```

Sending build context to Docker daemon 13.76MB

Step 1/6 : FROM alpine/git as clone

---> *f54f496311fb*

Step 2/6 : WORKDIR /app

---> *Running in c8ba2ced5cco*

Removing intermediate container c8ba2ced5cco

---> *921ao675126a*

Step 3/6 : RUN git clonexxxxxxxxxxxxxxxxxxxxxx.git

---> *Running in e5oco8d6a710*

[91mCloning into 'sample-java-app'...

[omRemoving intermediate container e5oco8d6a710

---> *7cb7beoc3d5f*

Step 4/6 : FROM maven:3.6.3-jdk-8 as build

---> *97495355e4f9*

Step 5/6 : WORKDIR /app

---> *Using cache*

---> *18c34e2doe6d*

Step 6/6 : /app/sample-java-app/app

---> *41eddegae237*

Successfully built 41edde9ae237

Successfully tagged

5dd3ddc4a1806600b5fc1cd2387eoeedff92b008: latest

[Pipeline]}

[Pipeline] // stage

[Pipeline] isUnix

[Pipeline] sh

+ docker inspect -f .

5dd3ddc4a1806600b5fc1cd2387eoeedff92b008

.

[Pipeline] withDockerContainer

Jenkins seems to be running inside container

96770dd957b2e5ed4afbooa48869ec33764965f2996obcf214269be26

584e389

[Pipeline] {

[Pipeline] stage

[Pipeline] {(Build)}

[Pipeline] sh

+ mvn clean package

[INFO] Scanning for projects...

Downloading from central:

<https://repo.maven.apache.org/maven2/io/spring/platform/platform-bom/2.0.3.RELEASE/platform-bom-2.0.3.RELEASE.pom>

Progress (1): 2.7/40 kB

Progress (1): 5.5/40 kB

Progress (1): 74/79 kB

Progress (1): 78/79 kB

Progress (1): 79 kB

Downloaded from central:

<https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-junit4/2.13/surefire-junit4-2.13.jar> (79 kB at 122 kB/s)

T E S T S

Running org.springframework.samples.sample-java-app.model.ValidatorTests

INFO Version - HVoooooooooooo: Hibernate Validator 5.2.4.Final

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed:
5.103 sec

INFO EhCacheManagerFactoryBean - Shutting down EhCache
CacheManager

INFO LocalContainerEntityManagerFactoryBean - Closing JPA EntityManagerFactory for persistence unit 'sample-java-app'

Results :

Tests run: 59, Failures: 0, Errors: 0, Skipped: 0

[INFO]

maven-war-plugin:2.3:war (default-war) @ sample-java-app ---

webapp

[INFO] Assembling webapp [sample-java-app] in
[/var/jenkins_home/workspace/java-sample-
web_docker/target/sample-java-app-4.2.5-SNAPSHOT]

[INFO] Processing war project

webapp resources

[INFO] Webapp assembled in [3014 ms]

[INFO] Building war:

[INFO] BUILD SUCCESS

[INFO] Total time: 08:48 min

[INFO] Finished at: 2020-07-07T06:48:30Z

[Pipeline] junit

Recording test results

[Pipeline] archiveArtifacts

Archiving artifacts

Recording fingerprints

[Pipeline]}

[Pipeline] // stage

[Pipeline]}

```
$ docker stop --time=1  
82184fd356aoaa4o3d6c1ad3cb8f8b651797624a  
354230f7ee87b743e38fde2a
```

```
$ docker rm 30f7ee87b743e38fde2a
```

[Pipeline] // withDockerContainer

[Pipeline]}

[Pipeline] // withEnv

[Pipeline]}

[Pipeline] // node

[Pipeline] End of Pipeline

Finished: SUCCESS

In the next section, we will install Jenkins on AKS.

[Install Jenkins on Azure Kubernetes Services \(AKS\)](#)

We need to install Azure CLI to perform this task. To get details on how to install Azure CLI go to Once Azure CLI is installed, follow the following steps:

We can execute the Azure CLI with the az command. Use the az login command to sign in. A successful login will open a browser window with a message in the agent.

Create resource group in Azure Cloud using az group create -l westus -n

We can create Azure Container Registry using az acr create -n sampleImages -g aks-jenkins --sku basic command.

Verify the creation of the container registry using az acr list command.

Before we push an image to the container registry, we must tag it with the fully qualified name of the ACR login server. The login server name is in the format .azurecr.io (all lowercase), for example, sampleimages.azurecr.io using docker

tag jenkinsci/blueocean sampleimages.azurecr.io/blueocean:v1 command.

Verify with docker images command:

REPOSITORY	TAG	IMAGE
ID	CREATED	SIZE
jenkinsci/blueocean	latest	c4239db11ad3 34 hours ago 562MB
sampleimages.azurecr.io/blueocean	v1	c4239db11ad3 34 hours ago 562MB

Push the image to ACR using docker push sampleimages.azurecr.io/blueocean command.

Verify ACR using az acr repository list --name sampleimages --output table command:

Result

Blueocean

Docker image is available in ACR. The next task is to perform AKS operation from the agent. Install AKS CLI using az aks install-cli command.

Create Kubernetes cluster using az aks create command: az aks create -g aks-jenkins -n JenkinsCluster --generate-ssh-keys --attach-acr sampleimages

Use --generate-ssh-keys to generate SSH key files '/home//.ssh/id_rsa' and '/home//.ssh/id_rsa.pub' under ~/.ssh to allow SSH access to the VM.

Use az aks get-credentials to get access credentials for a managed Kubernetes cluster (AKS). Once the context is merged in the current context, we can operate kubectl commands directly from the terminal and get details on the AKS cluster:

```
az aks get-credentials --resource-group aks-jenkins --name JenkinsCluster
```

Following is the YAML that we have used to deploy Docker image stored in ACR to AKS. We have used deployment and service kind for this deployment:

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
```

```
name: blueocean-deployment
```

```
labels:
```

```
  app: myapp
```

```
  type: front-end
```

```
spec:
```

```
template:
```

```
  metadata:
```

```
    name: blueocean-pod
```

```
  labels:
```

```
    app: blueocean
```

```
spec:
```

containers:

- name: blueocean

image: sampleimages.azurecr.io/blueocean:v1

ports:

- containerPort: 8080

volumeMounts:

- name: jenkins-home

mountPath: /var/jenkins_home

volumes:

- name: jenkins-home

emptyDir: {}

replicas: 1

selector:

matchLabels:

app: blueocean

apiVersion: v1

kind: Service

metadata:

name: blueocean-service

labels:

app: blueocean

type: front-end

spec:

selector:

app: blueocean

ports:

- protocol: TCP

port: 8080

targetPort: 8080

type: LoadBalancer

Execute kubectl apply -f

kubectl apply command is used to manage applications with the use of files. These files define Kubernetes resources. It helps to create and update resources in a Kubernetes cluster by running kubectl apply. This is one of the most used ways of managing Kubernetes applications in a production environment.

We have already merged the context so let us execute `kubectl get services` to get details on **Services** available in the cluster. In the above YAML, one deployment and one service are available:

```
[root@localhost Desktop]# kubectl get nodes -o wide
```

NAME	AGE	VERSION	INTERNAL-IP	EXTERNAL-IP	OS-IMAGE	KERNEL-VERSION	CONTAINER-RUNTIME	STATUS	ROLES
aks-nodepool1-21399502-vmss000000	17m	v1.15.11	10.240.0.4		4.15.0-1089-azure	docker://3.0.10+azure		Ready	agent
aks-nodepool1-21399502-vmss000001	v1.15.11	10.240.0.5			4.15.0-1089-azure	docker://3.0.10+azure		Ready	agent
aks-nodepool1-21399502-vmss000002	17m	v1.15.11	10.240.0.6		4.15.0-1089-azure	docker://3.0.10+azure		Ready	agent

To get details on pods, execute `kubectl get pods` command:

```
[root@localhost Desktop]# kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
blueocean-deployment-5bf949dc79-v9trt	1/1	Running	0	4m49s

We need an admin password and to get that execute kubectl logs blueocean-deployment-5bf949dc79-v9trt.

Configure Jenkins using the external IP address of service. To get external IP use command: kubectl get services. Use port number given in deployment YAML.

Observe volumeMounts block in YAML and explore other options for that block. Some useful resources for further exploration are:

Persistent <https://kubernetes.io/docs/concepts/storage/persistent-volumes/>

Jenkins on Kubernetes <https://cloud.google.com/solutions/jenkins-on-kubernetes-engine>

How to configure a manually provisioned Azure Managed Disk to use as a Kubernetes persistent volume?

<https://stackoverflow.com/questions/48774021/how-to-configure-a-manually-provisioned-azure-managed-disk-to-use-as-a-kubernetes-persistent-volume>

Exploring Jenkins on Kubernetes with Azure

Done!

[Install Jenkins on Amazon Elastic Kubernetes Service \(Amazon EKS\)](#)

We need to install AWS CLI to perform this task. To get details on how to install AWS CLI go to Once AWS CLI is installed, follow the below steps:

```
C:\Users\>aws --version
```

```
aws-cli/2.0.43 Python/3.7.7 Windows/10 exe/AMD64
```

Use AWS configure command to set up AWS CLI installation. AWS CLI prompts for four pieces of information as shown below:

```
C:\Users\>aws configure
```

```
AWS Access Key ID [*****ABCD]:
```

```
AWS Secret Access Key [*****wxYz]:
```

```
Default region name [eu-east-2]:
```

Default output format [None]:

The AWS CLI stores this information in a profile named default in the credentials file. By default, the information in this profile is considered when AWS CLI command is executed and it has no explicit mention of a profile to use.

Let us consider Amazon EKS is installed and configured. Visit <https://www.eksworkshop.com/> to explore multiple ways to configure VPC, ALB, and EC2 Kubernetes workers, and Amazon Elastic Kubernetes Service.

Once the EKS cluster is ready, we can deploy Kubernetes Pods, Service, and Deployment for Jenkins installation using the below script.

Before Jenkins deployment on EKS, let us understand some concepts of Kubernetes. Complete discussion on Kubernetes is out of the scope of this book.

book.

book. book. book. book. book. book. book. book.
book. book. book. book. book. book. book. book.
book. book. book. book. book. book. book. book.
book. book. book. book. book. book. book. book.

book. book.

book. book. book. book. book. book. book. book. book. book. book. book. book. book. book. book. book. book. book.

book. book. book. book. book. book. book. book. book. book. book. book. book. book. book. book. book. book.
--

book. book. book. book. book. book. book. book. book. book. book. book. book. book. book. book. book. book.

Table 7.1: Concepts of Kubernetes

To get more details on Kubernetes, its concepts and architecture, visit <https://kubernetes.io/docs/setup/>

Let us create Amazon Elastic File System and note that file system ID and File System Access Point:

Go to AWS management console and go to Amazon Elastic File System in AWS management console.

Click on **Create file**

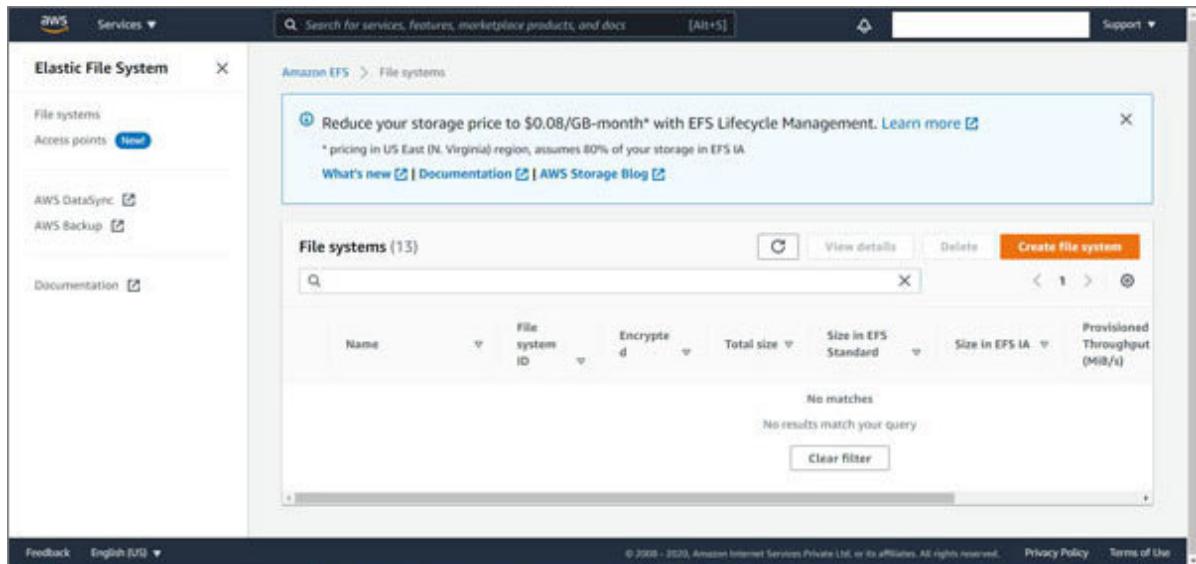


Figure 7.1: Elastic file system

Provide **Name** and click on

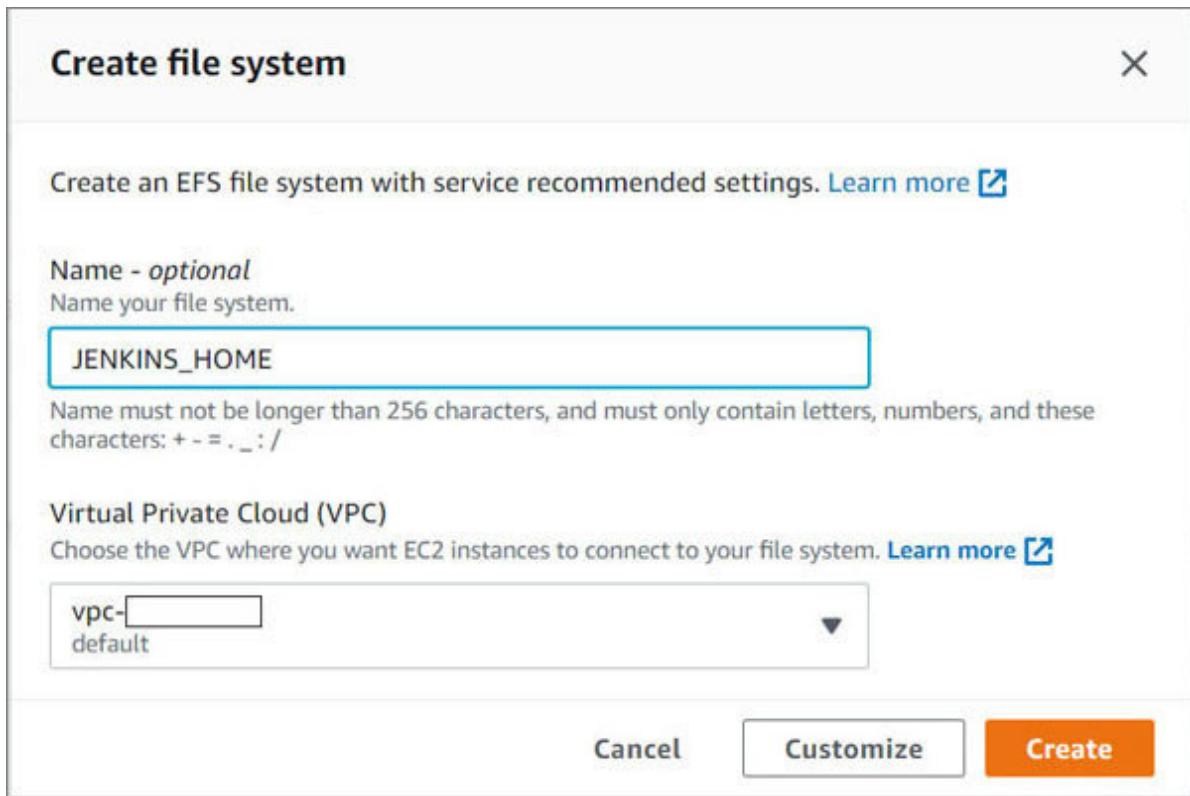


Figure 7.2: Create file system

Note file system ID available in the format: fs-xxxxxooo:

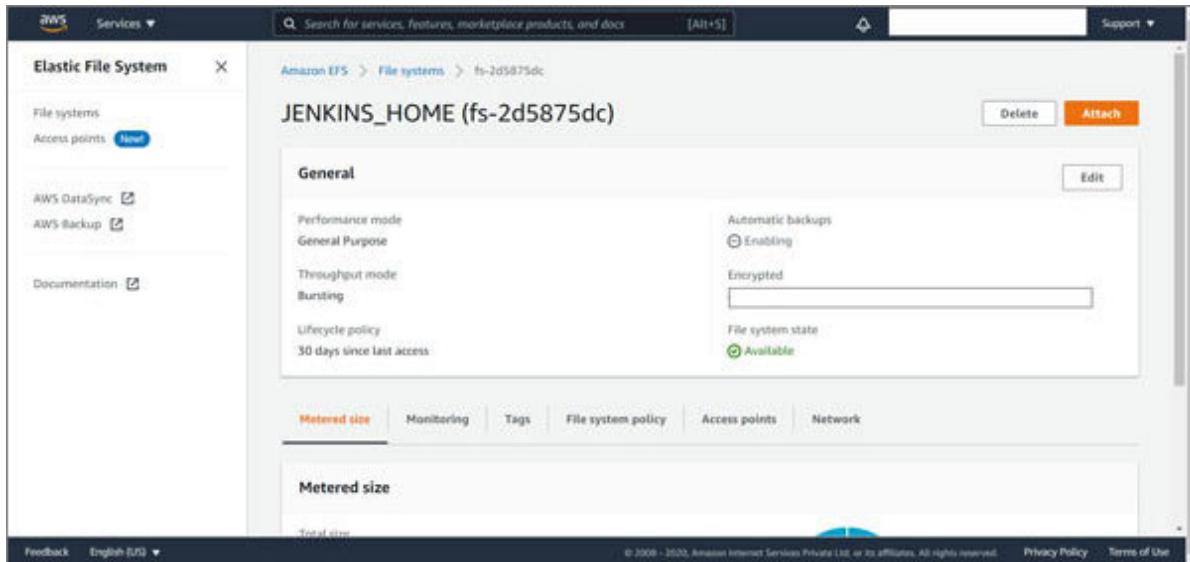


Figure 7.3: Elastic file system

Note file system access point available in the format: fsap-oxooooxxoxoxooooox:

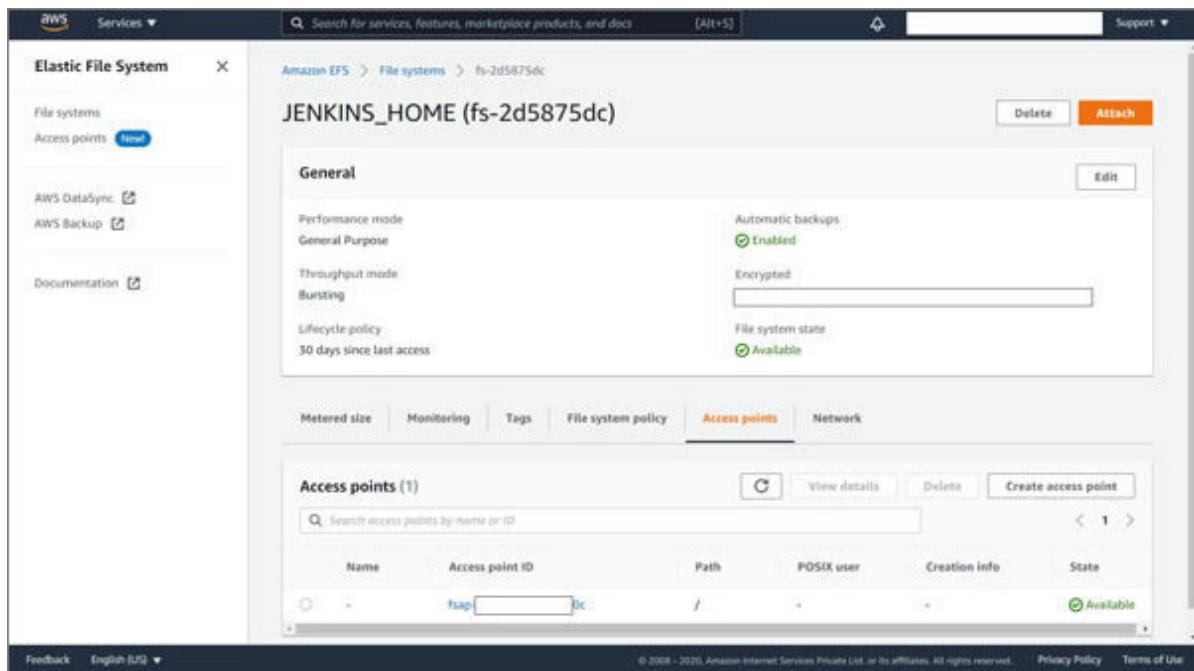


Figure 7.4: EFS access point

In the PV file, use file system ID and file system access point.

Following is the YAML for Persistent Volume:

```
apiVersion: v1
```

kind: PersistentVolume

metadata:

name: jenkins-efs-pv

spec:

capacity:

storage: 5Gi

volumeMode: Filesystem

accessModes:

- ReadWriteMany

persistentVolumeReclaimPolicy: Retain

storageClassName: jenkins-efs-sc

csi:

```
driver: efs.csi.aws.com
```

```
volumeHandle: fs-xxxxxooo::fsap-oc991ff9c6c46074f
```

Following is the YAML for PVC:

```
apiVersion: v1
```

```
kind: PersistentVolumeClaim
```

```
metadata:
```

```
  name: jenkins-efs-claim
```

```
spec:
```

```
  accessModes:
```

```
    - ReadWriteMany
```

```
  storageClassName: jenkins-efs-sc
```

```
resources:
```

```
  requests:
```

```
    storage: 10Gi
```

Following is the YAML for Jenkins on EKS Deployment:

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
```

```
  name: blueocean-deployment
```

```
labels:
```

```
  app: myapp
```

```
  type: front-end
```

```
spec:
```

```
template:
```

metadata:

name: blueocean-pod

labels:

app: blueocean

spec:

containers:

- name: blueocean

image: jenkinsci/blueocean

ports:

- name: http-port

containerPort: 8080

- name: jnlp-port

- containerPort: 50000

- volumeMounts:

- name: jenkins-home

- mountPath: /var/jenkins_home

- volumes:

- name: jenkins-home

- persistentVolumeClaim:

- claimName: jenkins-efs-claim

- replicas: 1

- selector:

- matchLabels:

- app: blueocean

apiVersion: v1

kind: Service

metadata:

name: blueocean-service

labels:

app: blueocean

type: front-end

spec:

selector:

app: blueocean

ports:

- protocol: TCP

- name: http

- port: 8080

- targetPort: 8080

- protocol: TCP

- name: agent

- port: 50000

- targetPort: 50000

type: LoadBalancer

We only need to deploy the above files in the EKS cluster using kubectl apply command.

Once deployment is successful, all JENKINS_HOME data from the EKS/Kubernetes pod will be stored in Amazon Elastic File

System.

Even in the situation of pod failure, new pod will be created and Jenkins will be up and running as it was in the previous state as Jenkins pod will access data from the Elastic File System available in AWS.

In the next section, we will see how to secure Jenkins.

Always secure Jenkins

This best practice is around authentication and authorization of users and enforcing access control. Security is an integral part of DevOps practices. In Jenkins, there are multiple ways available to configure Jenkins securely. We can configure authentication and authorization using multiple tools such as Azure AD, OpenLDAP, etc. We can configure authorization at a global level as well as the project level as well.

Project-based security

Jenkins authentication and authorization are good but what if we want to provide security at Project or Job level? How to provide access to certain users in specific jobs or pipeline only?

Go to **Specific job** and check **Enable project-based security** checkbox:

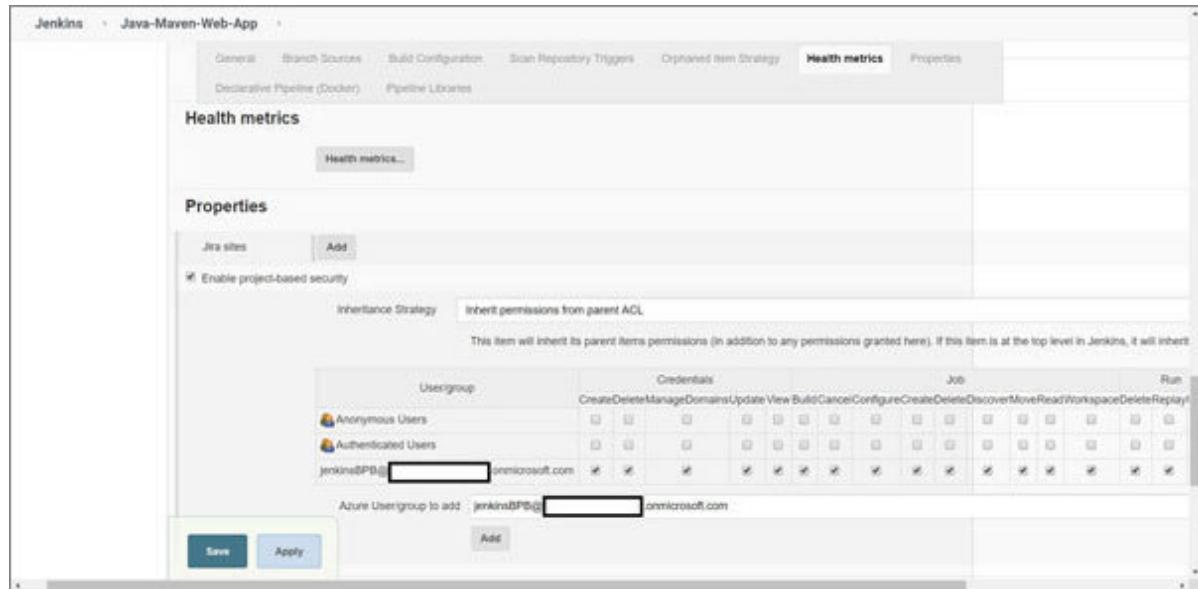
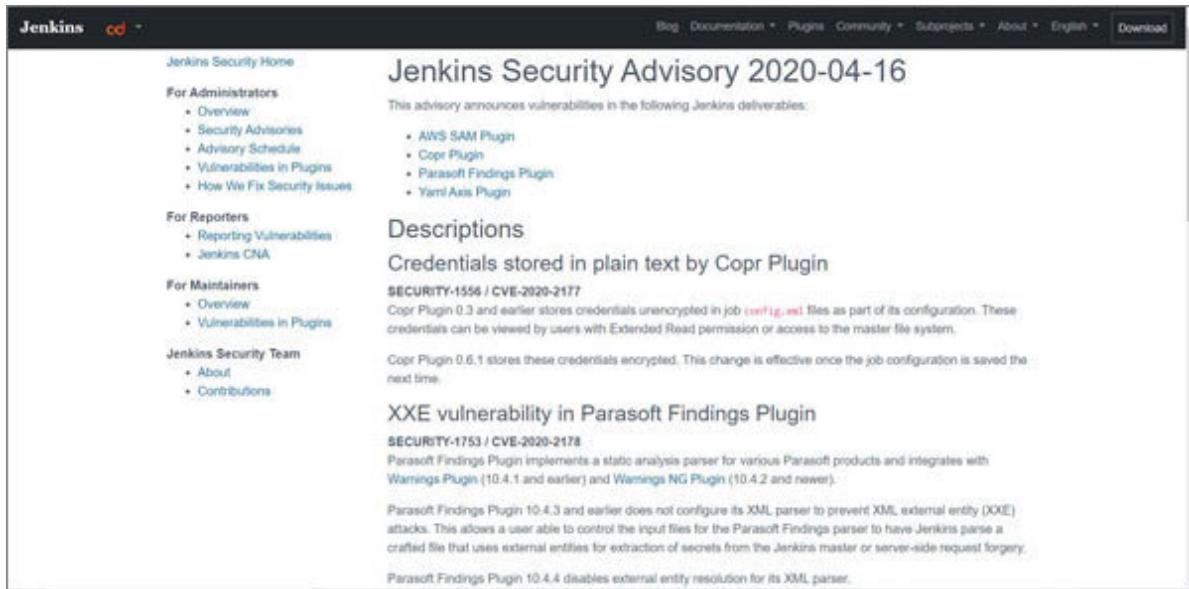


Figure 7.5: Project-based security

DevOps helps gain competitive advantage with faster times to market that results in happier customers and happier employees.

Jenkins also publishes Security advisories. Visit



The screenshot shows the Jenkins Security Advisory page for April 2020. The left sidebar contains links for Administrators, Reporters, Maintainers, and the Jenkins Security Team. The main content area is titled "Jenkins Security Advisory 2020-04-16" and discusses vulnerabilities in Jenkins deliverables, specifically mentioning the AWS SAM Plugin, Copr Plugin, Parasoft Findings Plugin, and Yaml Axis Plugin. It also details a vulnerability in the Copr Plugin where credentials were stored in plain text, and another in the Parasoft Findings Plugin related to XXE attacks. The page includes sections for Descriptions and Credentials stored in plain text by the Copr Plugin.

Figure 7.6: Security advisories

In the next section, we will discuss about Pipeline Best Practices.

Pipeline – best practices

Following are some of the best practices while creating a Pipeline or orchestration:

Use Blue Ocean to create Jenkinsfile and a multibranch pipeline for its efficient usage.

If the Domain-Specific Language construct is not available then use a script in the DSL to make automation work.

Combine commands in a single step or use a script file for multiple command execution.

Avoid using a Scripted pipeline as it has its learning curve and not easy to understand. The scripted pipeline is difficult to manage and maintain over the time and people who manage it find it difficult to hand over to new people as this may continue to be difficult to learn quickly.

Use Blue Ocean, Snippet generator to generate Jenkinsfile for the declarative pipeline quickly:

The screenshot shows the Jenkins Pipeline Syntax Snippet Generator interface. On the left, there's a sidebar with links like Snippet Generator, Declarative Directive Generator, Declarative Online Documentation, Steps Reference, Global Variables Reference, Online Documentation, Examples Reference, IntelliJ IDEA GDSL, and Pipeline As YAML Converter. The main area has a title 'Overview' with a brief description of the Snippet Generator. Below it is a 'Steps' section with a dropdown menu set to 'Sample Step: input: Wait for interactive input'. Underneath are several configuration fields: 'Message' (Deploy to QA Environment!), 'Custom ID' (empty), 'OK Button Caption' (Deploy to QA!), 'Allowed Submitter' (mitesh,chitrangada), and 'Parameter to store the approving submitter' (empty). Each field has a question mark icon to its right.

Figure 7.7: Pipeline syntax

After configuration click on **Generate Pipeline**

The screenshot shows the Jenkins Pipeline Syntax Snippet Generator interface after generating the pipeline script. It features a large central text area containing the generated script:

```
input submitter: 'mitesh,chitrangada', message: 'Deploy to QA Environment!', ok: 'Deploy to QA!'
```

 Above this text area is a blue button labeled 'Generate Pipeline Script'. Below the generated script, there's a section titled 'Global Variables' with a note: 'There are many features of the Pipeline that are not steps. These are often exposed via global variables, which are not supported by the snippet generator. See the [Global Variables Reference](#) for details.' At the bottom of the page, there are footer links for 'Page generated:' (Dec 22, 2020 12:19:59 AM IST) and 'Jenkins 2.235.4'.

Figure 7.8: Generate Pipeline script

It helps to create pipeline quickly in initial stages.

Create and configure Jenkinsfile for different branches based on requirement and make it as a practice to utilize pipelines in day-to-day operations to cultivate the mindset of people.

Role-based access to pipeline management and maintenance.

Configure branches for pipeline execution on need bases. Use Filter by name (with wildcards) in Branch sources to keep specific branches in the Jenkins pipeline execution:

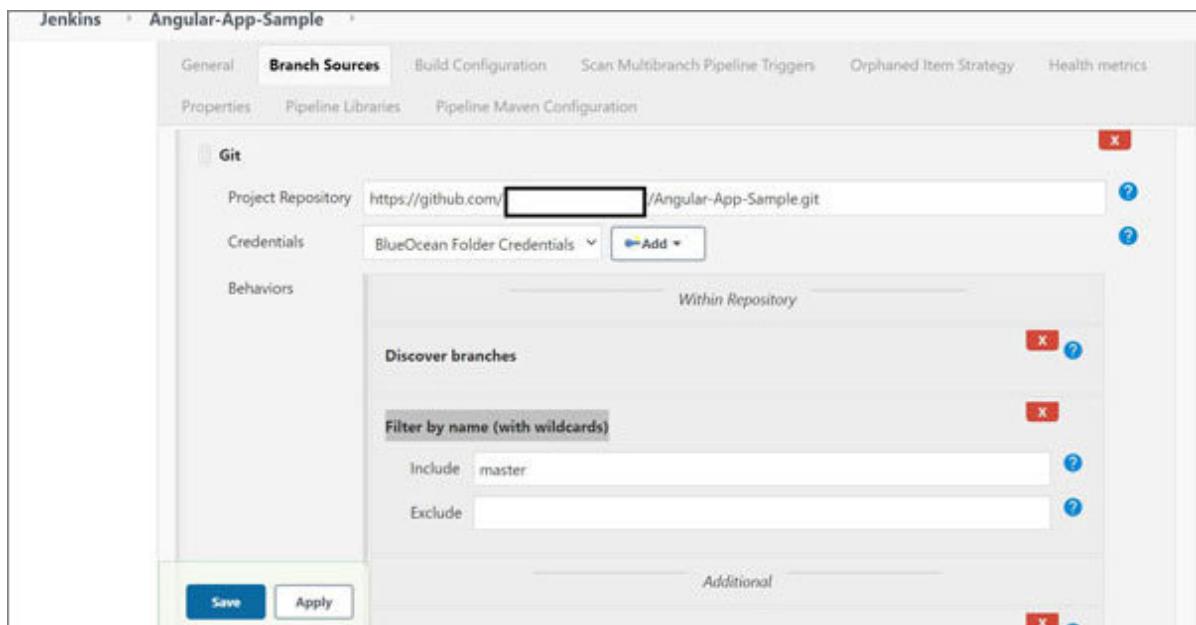


Figure 7.9: Filter by name (with wildcards)

It filters specific branches only and hence it limits resource usage.

Use controller agent architecture in the pipeline and use multiple agents for execution so parallel execution is possible.

Use Docker containers or Kubernetes pods as an agent to execute Jenkins pipeline:

```
pipeline {
```

```
    agent {
```

```
        kubernetes {
```

idleMinutes how long the pod will live after no jobs have run on it

```
        yamlFile 'build-agent-pod.yaml'
```

// path to the pod definition relative to the root of our project

```
        defaultContainer 'node'
```

```
// define a default container if more than a few stages use it, will  
default to jnlp container
```

```
}
```

```
}
```

```
stages {
```

```
.
```

```
.
```

```
.
```

```
}
```

```
}
```

Configure tools location and environment in agents to make things more generic:



Figure 7.10: Node configuration

Use publish coverage report plugin to publish unit test results. It supports Cobertura, Jacoco, and Istanbul report formats.

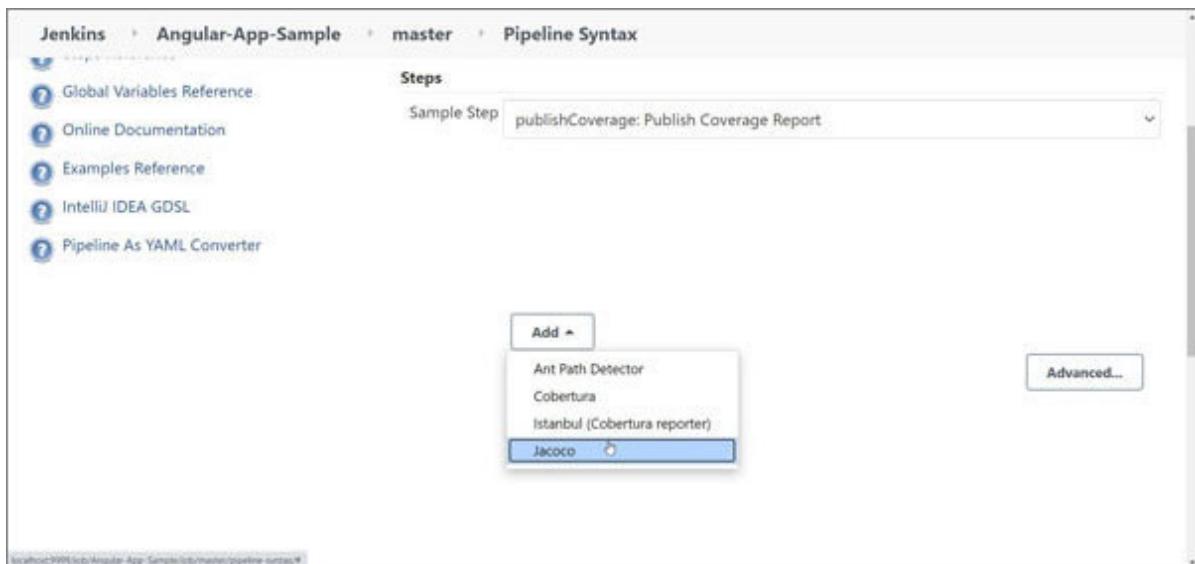


Figure 7.11: Publish coverage report

The report provides detailed information on coverage area:



Figure 7.12: Coverage report

Use Quality Gate for Code coverage for Unit tests execution.
Use thresholds based on business requirements and make sure to keep them in effect at the beginning:

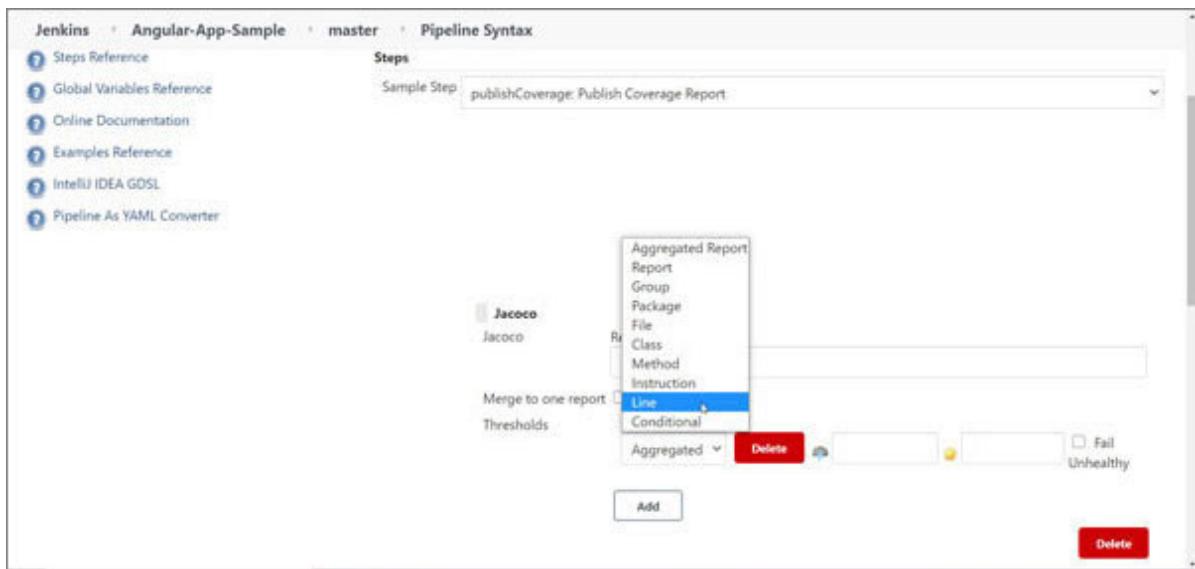


Figure 7.13: Coverage threshold

Use Test Results Analyzer plugin to get details to report on Unit test execution:

The screenshot shows the Jenkins Test Results Analyzer plugin interface. The left sidebar includes links for Up, Status, Changes, Build Now, View Configuration, Full Stage View, Open Blue Ocean, Coverage Report, Pipeline Configuration History, Test Results Analyzer (which is selected and highlighted in blue), and Pipeline Syntax. The main content area has tabs for Options, Download Test (JCM), Search, and Test Class/Package. Under 'Test Class/Package', there's a chart showing test results for HeadlessChrome across three browser versions. Below the chart is a section titled 'Top 10 Most Broken Tests' with a message stating 'There are no failing tests.' To the right is a 'Build Status' section showing a timeline from build 100 to 118. The timeline has colored dots representing test outcomes: green for Passed, red for Failed, yellow for Skipped, and blue for Total. A legend at the bottom right of the timeline defines these colors. A tooltip for the timeline indicates 'Build # 110: Passed: 0, Failed: 0, Skipped: 0, Total: 0'.

Figure 7.14: Test Results Analyzer plugin

Use folders to manage pipelines in a better way. We can configure Project based security at folder level in folder configuration as well:

The screenshot shows the Jenkins interface for a folder named "PipelineAsYAML". The left sidebar contains standard Jenkins navigation links like Up, Status, Configure, New Item, Delete Folder, People, Build History, Project Relationship, Check File Fingerprint, Move, Open Blue Ocean, Rename, Config Files, Credentials, and New View. Below these are sections for Build Queue (No builds in the queue) and Build Executor Status. The main content area is titled "PipelineAsYAML" and displays a table of five pipeline items. The table columns are S, W, Name, Last Success, Last Failure, Last Duration, and Fav. The items listed are:

S	W	Name	Last Success	Last Failure	Last Duration	Fav	
		Hello-World	2 mo 4 days - #20	2 mo 4 days - #22	6.5 sec		
		SampleAndroidApp	3 mo 23 days - #39	N/A	32 sec		
		SampleAngularApp	3 mo 22 days - #39	N/A	21 sec		
		SampleFlutterApp	3 mo 24 days - #39	N/A	15 sec		
		SampleIonicCordovaApp	3 mo 23 days - #39	N/A	24 sec		

Below the table is a legend: Atom feed for all, Atom feed for failures, and Atom feed for just latest builds.

Figure 7.15: Folder

It is important to backup JENKINS_HOME regularly if it is not stored in Cloud File Systems such as Amazon Elastic File Systems.

Use Jenkins environment variables in Declarative pipeline script for creating effective pipelines:

It is important to provide training to all team members on how CI/CD implementation will help them to save time and how Pipeline as a Code is more effective to automate Application Life Cycle Management phases

In organization, automated deployment to most of the environment is not feasible due to a lot of processes that need to be followed for audit purposes. Hence, it is important to introduce Input parameter in the pipeline that allows manual intervention before pipeline stage execution goes ahead

In the next section, we will discuss the *Backup and restore* plugin available in Jenkins.

[Backup and restore](#)

Backup and restore is an important activity in Jenkins management and maintenance. Consider a scenario where you need to migrate Jenkins setup from one system to another system due to a lack of resources in the existing system. One way is to copy the entire Jenkins Home directory and keep backup for multiple days and perform it manually or using a script.

Another way is to use the ThinBackup plugin. Let's see how the ThinBackup plugin works:

Install **Plugins** from the **Manage Plugins** section available in **Manage**

The screenshot shows the Jenkins Plugin Manager interface. The top navigation bar includes links for 'Jenkins', 'Plugin Manager', 'Back to Dashboard', 'Manage Jenkins', and 'Update Center'. On the right side of the header are icons for 'monitor', 'JenkinsBPPUser', and 'log out'. Below the header, there are tabs for 'Updates', 'Available' (which is selected), 'Installed', and 'Advanced'. A search bar with the placeholder 'search' and a filter input field containing 'backup' are located at the top right. The main content area displays a table of available plugins:

Install	Name	Version
	Backup	1.6.1
	Backup and Interrupt Job plugin	1.0
	Google Cloud Backup	0.6
	Periodic Backup	1.5
	ThinBackup	1.9

Below the table are two buttons: 'Install without restart' and 'Download now and install after restart'. A status message 'Update information obtained: 12 hr ago' and a 'Check now' button are also present. At the bottom right of the page, a footer note reads 'Page generated Apr 26, 2020 12:27:23 AM IST RESTART Jenkins ver. 2.222.3'.

Figure 7.16: Thin backup plugin

Go to **Manage Jenkins** | Click on

This screenshot shows the Jenkins 'Manage Jenkins' interface, specifically the 'ThinBackup' configuration page. The left sidebar lists various Jenkins management options like 'New Item', 'People', 'Build History', etc. The main content area has a title 'ThinBackup' with a folder icon. It contains three main buttons with icons: 'Backup Now' (blue arrow pointing up), 'Restore' (blue arrow pointing down), and 'Settings' (gear icon). Below these buttons, there are three expandable sections: 'Cloud Statistics', 'Build Queue (1)', and 'Build Executor Status'. The 'Build Queue (1)' section shows one build item with a progress bar and a red error icon. The 'Build Executor Status' section shows one idle executor.

Figure 7.17: Thin backup

Configure backup directory, backup schedule for full and incremental backup as per need:

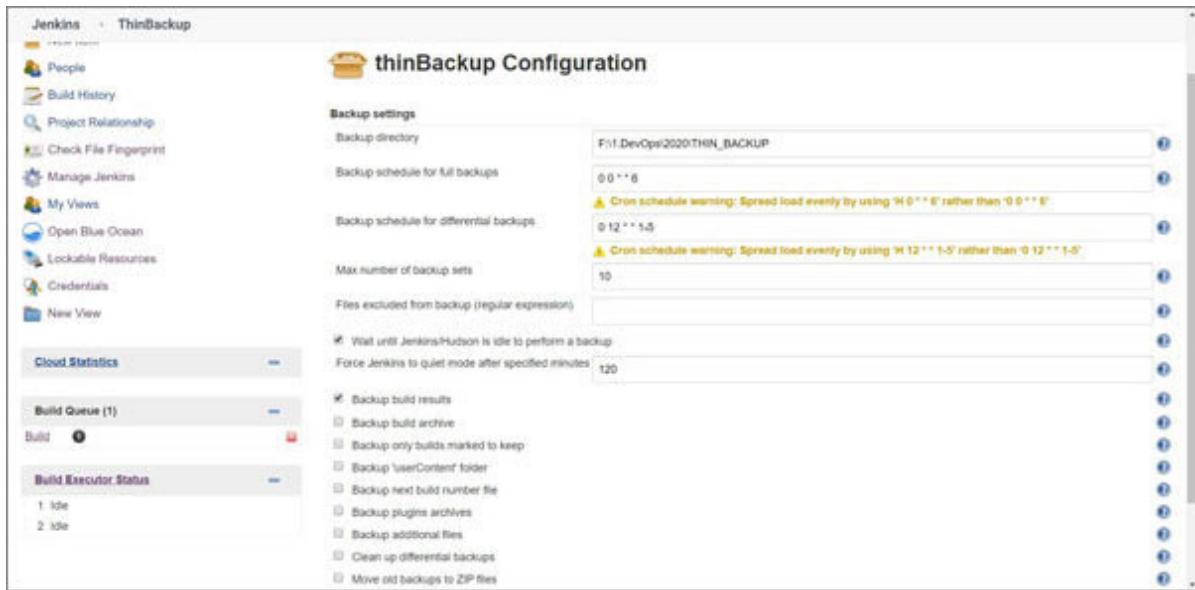


Figure 7.18: Thin backup configuration

Click on **Backup**

Go to restore and verify multiple list items based on a number of times you have taken backup:

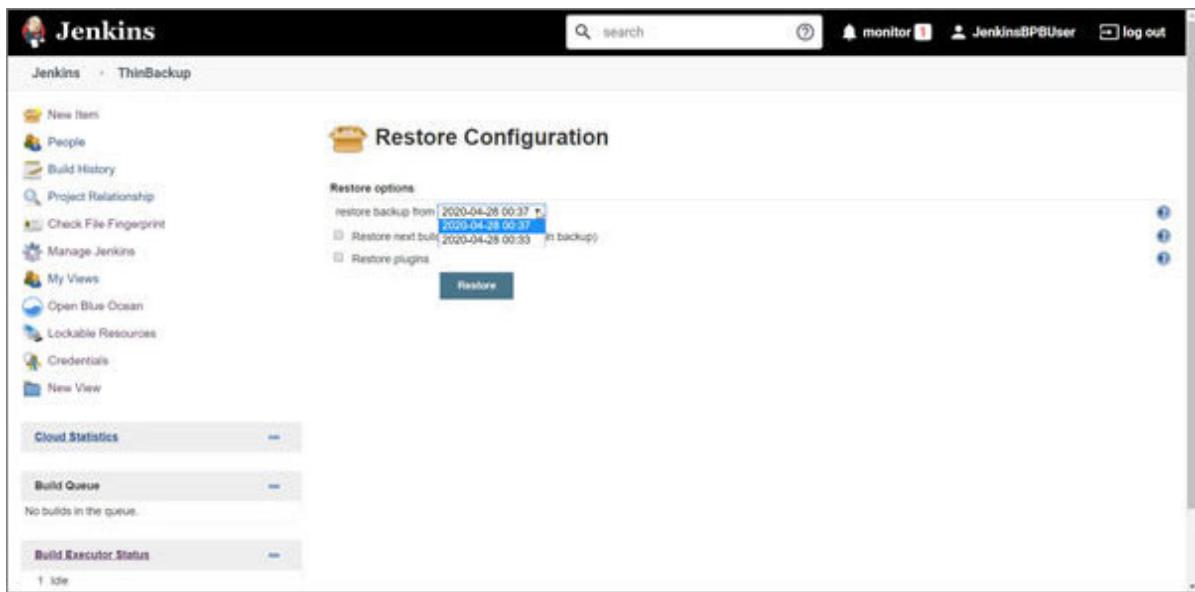


Figure 7.19: Restore configuration

To verify whether backup and restore works as expected, change the JENKINS_HOME directory in **Environment variables** and restart the Jenkins.

This means a fresh setup. Skip all plugin installations and go to Jenkins Dashboard.

Go to **Manage Jenkins | Manage Plugins | Available** and install **ThinBackup** plugin only:



Figure 7.20: New Jenkins setup - plugins

After successful installation, go to **Manage Jenkins** | Click on

Configure **Backup Directory** that was configured earlier.

Go to **Restore** and verify if the same Backup list is available.

Click on

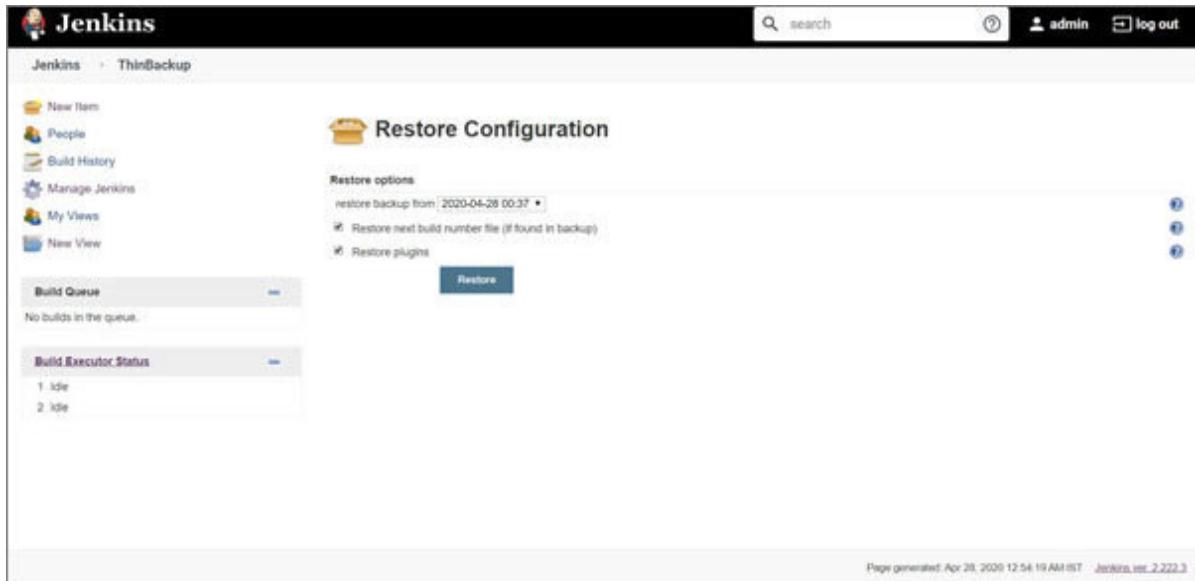


Figure 7.21: Restore configuration

Following are some important tips while taking Backup:

Don't upgrade any incompatible Plugin. It might break the setup

In production support, upgrade plugins one by one and verify before proceeding ahead

Take backup as well at certain interval

Server snapshot can be helpful if there is no point of return

Simulate failure and practice backup and restore process multiple times

Restoration activity will take time. Make sure that you are connected to the internet so plugins are also installed based on backup.

Verify the following console log in Jenkins:

2020-04-27 19:42:59.700+0000 [id=108]
INFO h.model.UpdateCenter\$DownloadJob#run: Starting the installation of branch-api on behalf of admin

2020-04-27 19:42:59.700+0000 [id=108]
INFO h.m.UpdateCenter\$InstallationJob#_run: Skipping duplicate install of: Branch API@2.5.6

2020-04-27 19:42:59.700+0000 [id=108]
INFO h.model.UpdateCenter\$DownloadJob#run: Installation
successful: branch-api

2020-04-27 19:42:59.700+0000 [id=108]
INFO h.model.UpdateCenter\$DownloadJob#run: Starting the
installation of workflow-api on behalf of admin

2020-04-27 19:42:59.700+0000 [id=108]
INFO h.m.UpdateCenter\$InstallationJob#_run: Skipping
duplicate install of: Pipeline: API@2.40

2020-04-27 19:42:59.708+0000 [id=108]
INFO h.model.UpdateCenter\$DownloadJob#run: Installation
successful: workflow-api

2020-04-27 19:42:59.718+0000 [id=108]
INFO h.model.UpdateCenter\$DownloadJob#run: Starting the
installation of pubsub-light on behalf of admin

2020-04-27 19:42:59.719+0000 [id=108]
INFO h.m.UpdateCenter\$InstallationJob#_run: Skipping
duplicate install of: Pub-Sub "light" Bus@1.13

2020-04-27 19:42:59.720+0000 [id=108]
INFO h.model.UpdateCenter\$DownloadJob#run: Installation
successful: pubsub-light

2020-04-27 19:42:59.838+0000
[id=18] INFO o.j.h.p.t.ThinBackupMgmtLink#doRestore:
Restore finished.

Once Restore activity is finished, verify that all plugins and jobs are available as it was earlier in the original JENKINS_HOME directory:

The screenshot shows the Jenkins dashboard with the following details:

- Left Sidebar:** Includes links for New Item, People, Build History, Manage Jenkins, Open Blue Ocean, Lockable Resources, Credentials, and New View.
- Top Bar:** Includes search, monitor, and enable auto refresh buttons.
- Job List:** A table showing five jobs: Build, Destroy2Tomcat, Java-Maven-Web-App, Java-sample-app, and SonarQubeAnalysis. The table includes columns for Status (S), Warning (W), Name, Last Success, Last Failure, and Last Duration.
- Cloud Statistics:** Shows 0 MUL icon.
- Build Queue:** Shows "No builds in the queue."
- Build Executor Status:** Shows 1 idle and 2 idle executors.
- Bottom Right:** Legend for Atom feed links and a page footer indicating the page was generated on April 28, 2020 at 12:33 AM IST.

Figure 7.22: Jenkins dashboard

In the next section, we will discuss Monitoring.

Monitoring

On Jenkins Dashboard, there are three notifications. Let us verify and fix them:

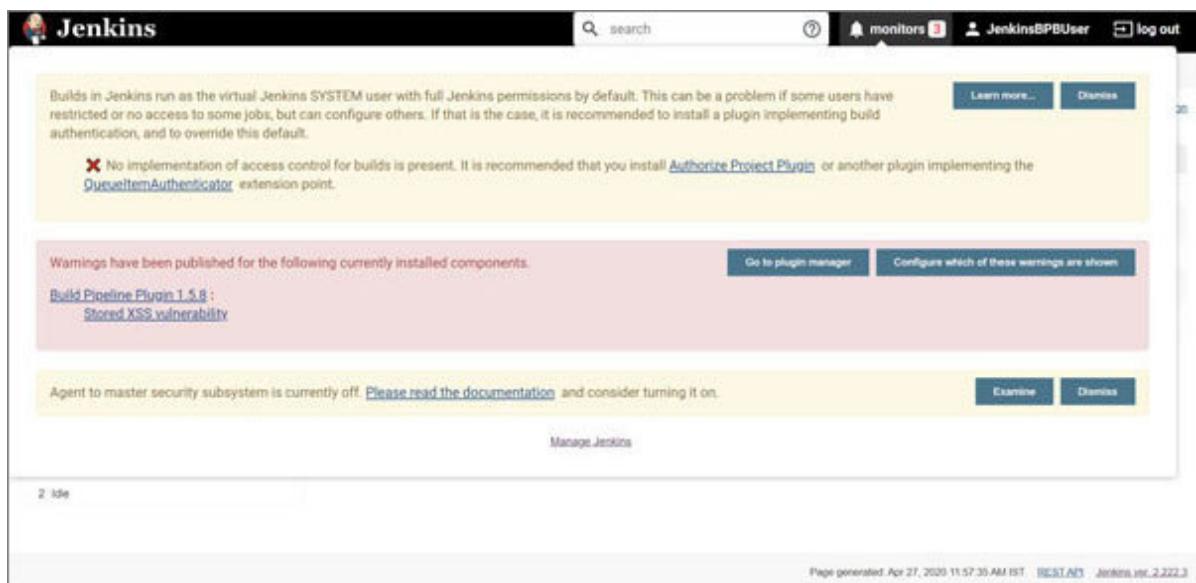


Figure 7.23: Jenkins monitoring

Go to **Manage Jenkins** and verify the messages available at the top. We have enabled Project-based security in the **Security** section in this chapter. Access control is still not configured:

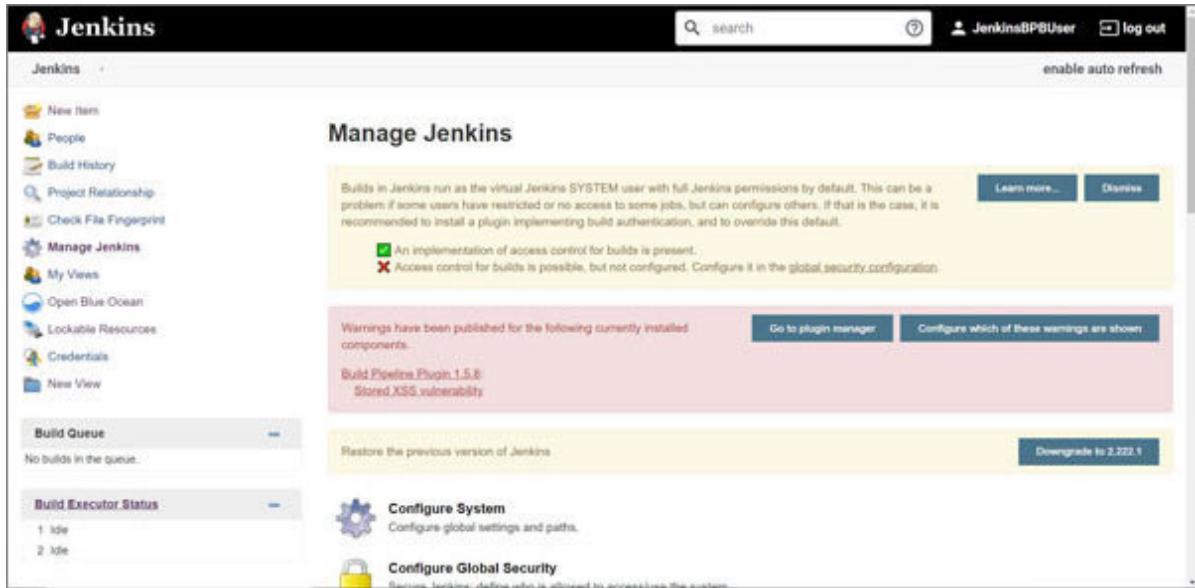


Figure 7.24: Manage Jenkins

To configure **Agent Controller** go to **Manage Jenkins | Global Security**

Configure **Random as TCP Port** for an inbound agent.

Enable **Agent Controller Access**

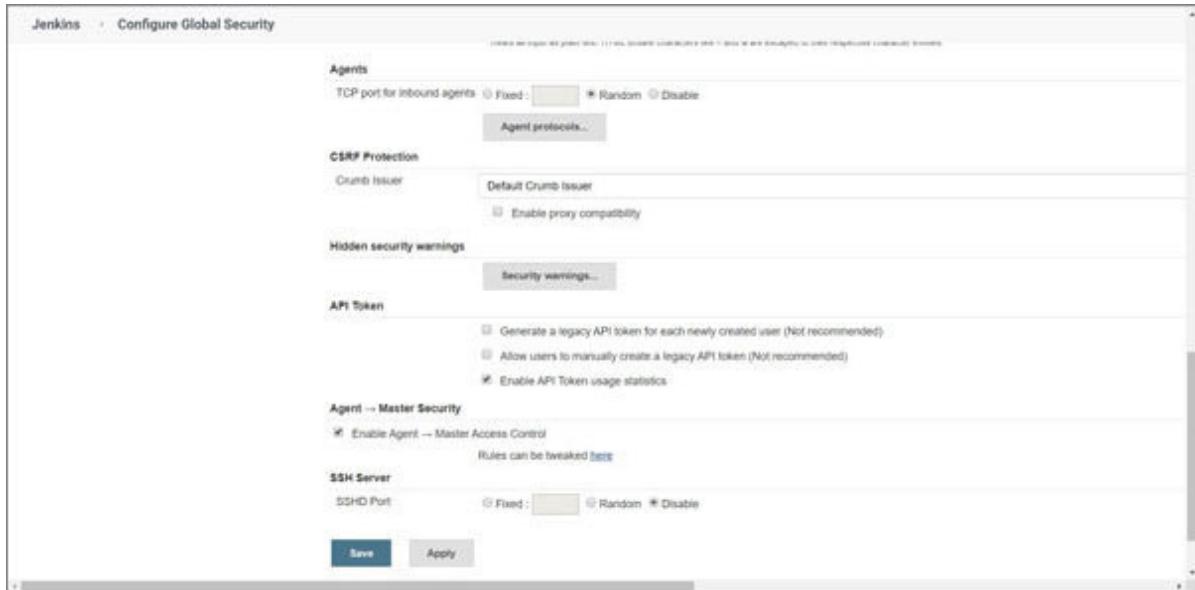


Figure 7.25: Agent controller security

Go to **Manage Jenkins | Manage Plugins | Available** and install **Authorize Project** plugin:

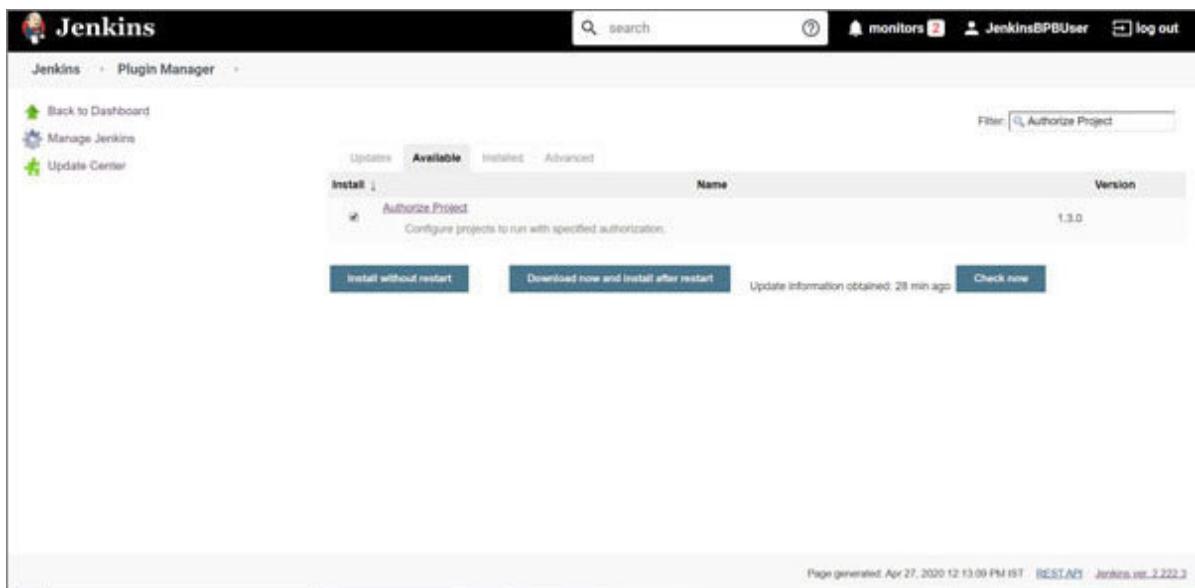


Figure 7.26: Authorize project plugins

Go to **Manage Jenkins | Global Security**

A new section called **Access Control for Builds** is available.

Configure the **Strategy** and click on

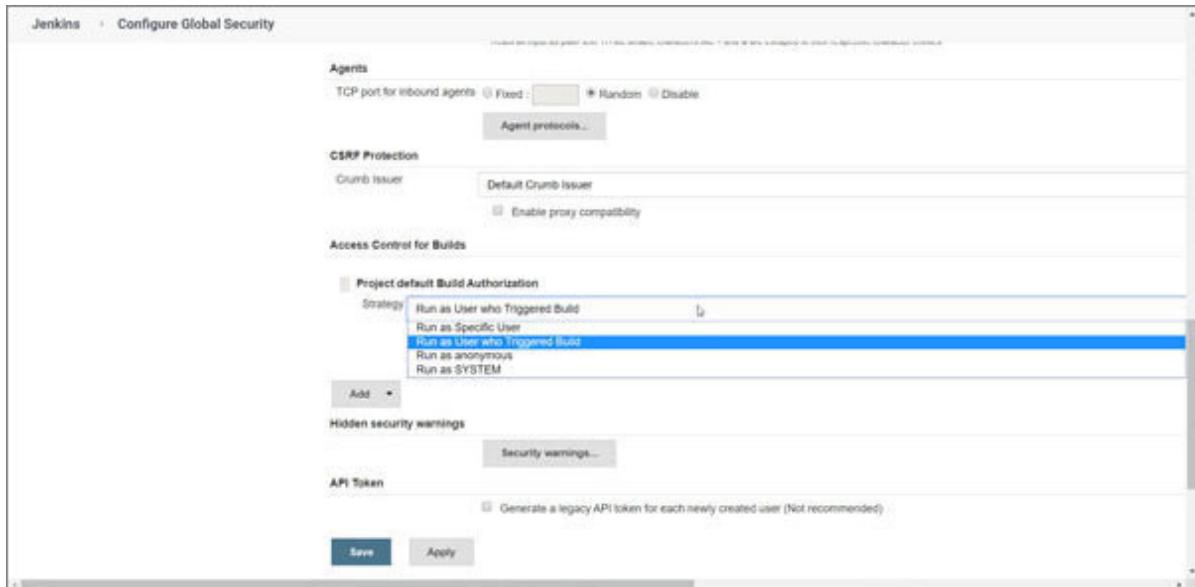


Figure 7.27: Access controls for builds

Done!

Conclusion

Jenkins Best practices have evolved and they will continue to evolve in the future based on technical aspects as well as user experience is concerned. Controller agent architecture helps to improve overall build execution performance and to manage and maintain automation infrastructure with ease. We have covered authentication and authorization concerning security in this chapter. Backup and restore is a critical important the moment your Jenkins controller crash and there is no backup available. Best practices help to increase the usage of Jenkins or any automation server to transform the existing culture of an organization and this is a continuous process.

Multiple choice questions

For Load Balancing, Labels can be used in Jenkins while configuring Controller Agent Architecture.

True

False

To Disable the login in Jenkins change true to false in

True

False

Answers

a

a

Questions

What Azure Active Directory?

What is authentication?

What is authorization?

What is distributed architecture?

[Index](#)

A

agent pools
agent offline [62](#)
build execution, on another node [63](#)
build execution, on labeled node [62](#)
disconnected agent [63](#)
label name as label expression [61](#)
labels in agent configuration [60](#)
nodes with same label [61](#)
Amazon EKS
reference link [193](#)
Amazon Elastic File System
creating
Android Lint plugin
URL [141](#)
Android Lint tool [136](#)
Angular [8](#) [159](#)
App Center
about [94](#)
account settings [96](#)
API Token [96](#)
app, creating [95](#)

Distribution [97](#)
full stage view [98](#)
release history [99](#)
Application Lifecycle Management Cycle [2](#)
AWS CLI
reference link, for installation [192](#)

B

backup and restore, Jenkins
about [207](#)
restore configuration
ThinBackup plugin, using [208](#)
basic pipeline concepts
Agent or Node [30](#)
pipeline (declarative) [30](#)
stage [30](#)
step [30](#)
best practices, Jenkins
about [182](#)
installation, with fault tolerance [182](#)
security [199](#)
best practices, Pipeline [201](#)
Blue Ocean
about [37](#)
configuring [38](#)

dashboard [39](#)
GitHub, connecting to [40](#)
GitHub Organization [42](#)
Jenkins Dashboard [43](#)
Jenkinsfile location [45](#)
personal access token, creating [41](#)
pipeline, creating [40](#)
Scan Repository Log [44](#)
build pipeline
Build job, creating [27](#)
configuration [29](#)

Deploy2Tomcat job, creating [28](#)
SonarQubeAnalysis job, creating [26](#)
using [26](#)
Build Pipeline View [30](#)

c

CI/CD pipeline with YAML, for Android app
building [131](#)
continuous delivery, configuring [151](#)
continuous integration, configuring [135](#)
deliverables [133](#)
Jenkins plugin [134](#)
multi-stage CI/CD pipeline, creating [135](#)
tools [132](#)

unit tests, executing
YAML pipeline script [156](#)
CI/CD Pipeline with YAML, for Angular Application
building [159](#)
continuous delivery, configuring [174](#)
continuous integration [164](#)
deliverables [161](#)
multi-stage CI/CD pipeline, creating
tools [160](#)
YAML pipeline script [178](#)
CI/CD pipeline with YAML, for Flutter app
continuous delivery, configuring [95](#)
continuous integration, configuring [86](#)
Coverage report, verifying [92](#)
deliverables [83](#)
DevOps tools [82](#)

multi-stage CI/CD pipeline, creating [84](#)
YAML pipeline script [100](#)
CI/CD Pipeline with YAML, for Ionic Cordova Application
building [103](#)
continuous delivery, configuring [122](#)
continuous integration, configuring
deliverables [105](#)
DevOps tools [104](#)
Jenkins plugins [106](#)
multi-stage CI/CD pipeline, creating [107](#)
YAML pipeline script, creating [128](#)

cloud computing
about [7](#)
cloud deployment models [8](#)
essential characteristics [8](#)
cloud deployment models
community cloud [8](#)
hybrid cloud [8](#)
private cloud [8](#)
public cloud [8](#)
Code Coverage API [92](#)
command flutter analysis [88](#)
continuous delivery (CD)
about [9](#)
versus, continuous deployment [10](#)
continuous delivery, for Android app
about [151](#)
app, adding in App Center [152](#)
App Center dashboard [154](#).

distribute stage logs, verifying [153](#)
Sign Android Package stage logs, verifying [153](#)
YAML script, for distributing Android package to App Center
[151](#)
continuous delivery, for Angular app
configuring [175](#)
Full stage view [177](#)
traditional dashboard, verifying in Jenkins [176](#)
continuous delivery, for Flutter app

configuring [94](#)
continuous delivery, for Ionic Cordova app
App Center Distribution [124](#)
App Center, integrating in Jenkins [123](#)
App Center, using [122](#)
configuring [122](#)
Full Stage View [124](#)
multi-branch pipeline [125](#)
continuous deployment [11](#)
continuous integration (CI) [7](#)
continuous integration, for Android App
Android pipeline status [149](#)
artifacts, verifying [149](#)
build timeline [151](#)
code coverage, calculating [147](#)
implementing [136](#)
Lint analysis, performing for Android App
pipeline status [150](#)
pipeline steps [150](#)
YAML script for Build stage [148](#)

continuous integration, for Angular App
about [164](#)
Angular Code analysis, in SonarQube [170](#)
branches, in Jenkins dashboard [167](#)
Cobertura configuration, in karma.conf.js [165](#)
Headless Browser, configuring [166](#)
Jenkinsfile.yml, configuring [167](#)

Junit configuration, in karma.conf.js [165](#)
lint configuration, in package.json [166](#)
logs, verifying in Blue Ocean dashboard [171](#)
pipeline execution status [174](#)
scripts, configuring in package.json [166](#)
SonarQube, configuring [168](#)
static code analysis [169](#)
unit test code coverage [173](#)
unit test results [172](#)
continuous integration, for Flutter app
APK file, verifying in Artifacts section [93](#)
branches, in Jenkins dashboard [87](#)
build stage, verifying [93](#)
command flutter analysis [88](#)
configuring [86](#)
multi-branch pipeline log, verifying [87](#)
pipeline execution status, verifying [94](#)
SCA stage execution, verifying in Blue Ocean dashboard [89](#)
Test Results Analyzer charts [91](#)
unit test report, verifying [90](#)
continuous integration, for Ionic Cordova app
branch sources [114](#).

Cobertura Coverage report [122](#)
configuring [107](#)
Issues tab in SonarQube [118](#)
logs for code analysis, verifying [116](#)
logs, verifying in Blue Ocean dashboard [119](#)

pipeline execution status [120](#)
Quality Gate result [117](#)
SonarQube dashboard [116](#)
static code analysis, performing [115](#)
Test Results Analyzer [120](#)
Test Results Analyzer charts [121](#)
Unit Tests report [119](#).
continuous practices
about [6](#)
implementation [6](#)
continuous testing [10](#)
controller-agent architecture [52](#)

D

Declarative pipeline
about [35](#)
stage view [37](#)
syntax template [36](#)
DevOps
about [2](#)
business benefits [5](#)
defining [3](#)
history [3](#)
People [4](#).
Processes [4](#)

responsibilities [4](#)

technical benefits [5](#)

Tools [4](#)

DevOps practices

about [5](#)

cloud computing

continuous code inspection [7](#)

continuous delivery [9](#)

continuous deployment [11](#)

continuous integration (CI) [7](#)

continuous practices [6](#)

continuous testing [10](#)

implementation [4](#)

domain-specific language (DSL) [25](#)

E

external Job template [57](#)

F

Flutter

environment variables [85](#)

installing, on Windows

Freestyle project [52](#)

G

Generic War file usage [15](#)

Global Security

configuring [78](#)

I

issues, Jenkins pipeline commands

resolving

J

Jenkins

about [11](#)

admin user, creating [18](#)

backup and restore [207](#)

best practices [182](#)

commands, for running [14](#)

configuring [21](#)

customizing [16](#)

dashboard, verifying [19](#)

environment variables [24](#)

features [13](#)

global tool configuration [21](#)

history [12](#)
instance configuration [18](#)
managing [20](#)
monitoring
node configuration [22](#)
overview [12](#)
plugins installation failures [17](#)
prerequisites [14](#)
Security, configuring [78](#)
suggested plugins, installing [16](#)
system properties [23](#)
Jenkins agent
connected agent [58](#)
controller, accessing [53](#)
disconnected agent [57](#)
node configuration [56](#)

nodes [54](#).
Jenkins controller
versus, Jenkins agent [53](#)
Jenkinsfile
using [30](#)
Jenkins installation, with fault tolerance
Dockerfile agent, in declarative pipeline
Docker used [183](#)
on Amazon Elastic Kubernetes Service (Amazon EKS) [193](#)
on Azure Kubernetes Services (AKS)
Jenkins plugins [84](#).

JUnit Report

about [89](#)

using [90](#)

K

karma.conf.js file configurations

about [107](#)

Corbertura, adding [108](#)

Headless Browser, configuring [108](#)

Package.json, configuring [109](#)

unitReporter configuration, adding [108](#)

Kubernetes

concepts [194](#)

reference link [194](#)

L

labels in agent configuration [59](#)

Lint analysis

about [136](#)

Lint Issues [143](#)

performing, for Android App

M

monitoring, Jenkins [212](#)
multibranch pipeline template [60](#)
multi-configuration project template [58](#)

N

National Institute of Standards and Technology (NIST) [7](#)

P

Pipeline
best practices
Pipeline as a Code (PaaS) [25](#)
Pipeline as YAML [46](#)
pipelines
about [25](#)
Blue Ocean [37](#)
build pipeline [26](#)
Declarative pipeline [34](#)
Pipeline as YAML [45](#)
Scripted pipeline [30](#)

S

Scripted pipeline

about [30](#)

Declarative Directive Generator, using [33](#)

Pipeline Syntax [33](#)

Snippet Generator, using [33](#)

structure [31](#)

template [32](#)

Security advisories

reference link [200](#)

Security, Jenkins

about [199](#)

project-based security [200](#)

static code analysis (SCA) [6](#)

T

Test Results Analyzer plugin [91](#)

ThinBackup plugin

working [207](#)

Y

YAML pipeline

Agents, defining [73](#)

artifacts, archiving [76](#)

components [52](#)

creating [64](#)

Declarative script block [77](#)
execution steps [69](#)
parameters [73](#)
post section [74](#)
replaying [66](#)
sample pipeline [64](#)
stage execution [65](#)
stages, defining [70](#)
syntax of post section [75](#)
tools definitions [71](#)
triggers and environments [71](#)
validating [69](#)

valid pipeline declarative script [67](#)
writing [65](#)