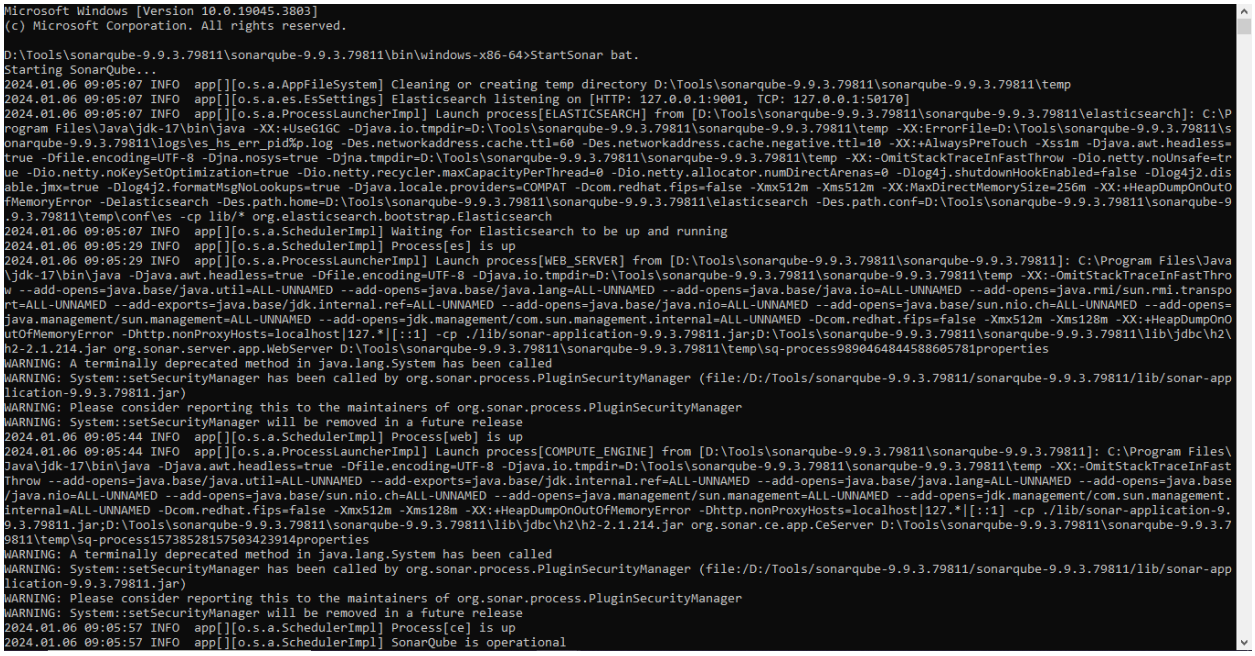
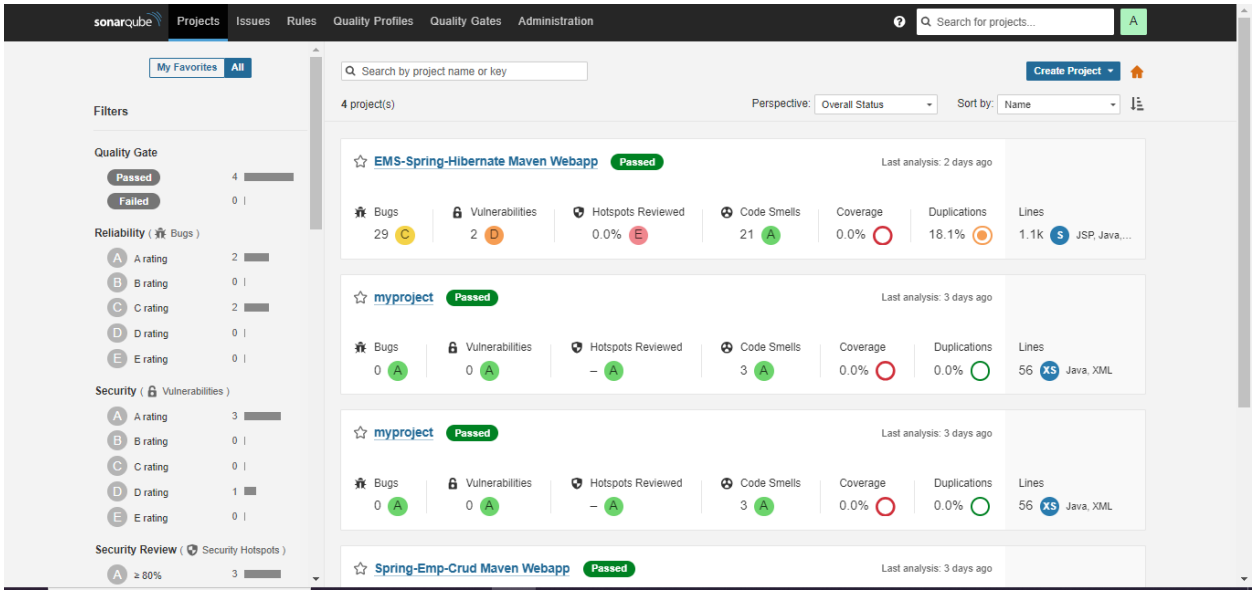
Devops:

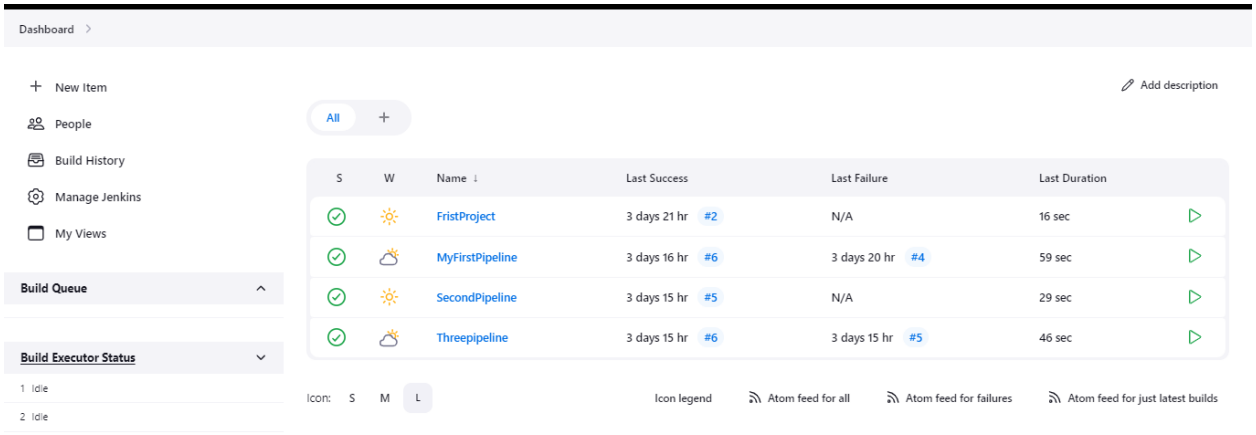
DevOps is a set of practices, tools, and a cultural philosophy that automate and integrate the processes between software development and IT teams. It emphasizes team empowerment, cross-team communication and collaboration, and technology automation.

SonarQube:





Jenkins:



Docker:

Jenkins:
Jenkins Types:

Jenkins : Type of Jenkins Jobs

Jenkins jobs are the building blocks of the automation process within Jenkins. Each job represents a specific task or process that Jenkins executes as part of the CI/CD pipeline

Types of Jenkins Jobs:

- ➔ Freestyle Project
- ➔ Pipeline Project
- ➔ Multibranch Pipeline

Let's discuss them in detail..

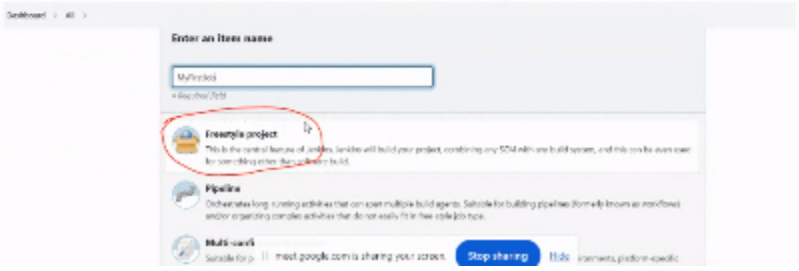
Jenkins : Type of Jenkins Jobs

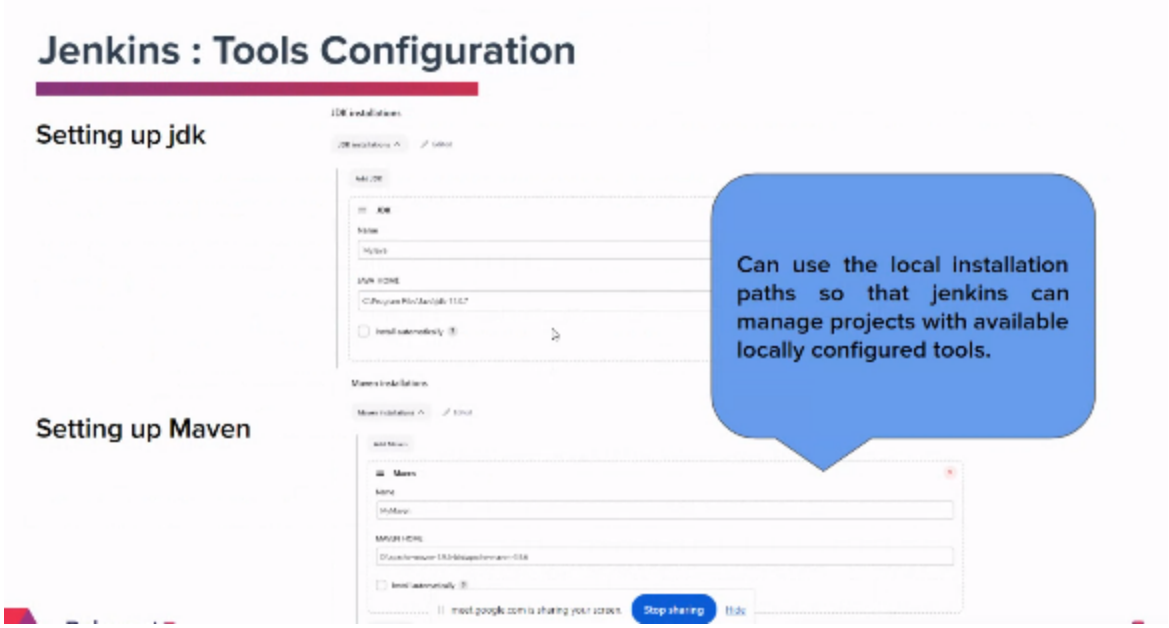
Freestyle Project:

- The most flexible job type in Jenkins.
- Allows users to configure each build step manually.
- Suitable for simple or custom build and deployment processes.

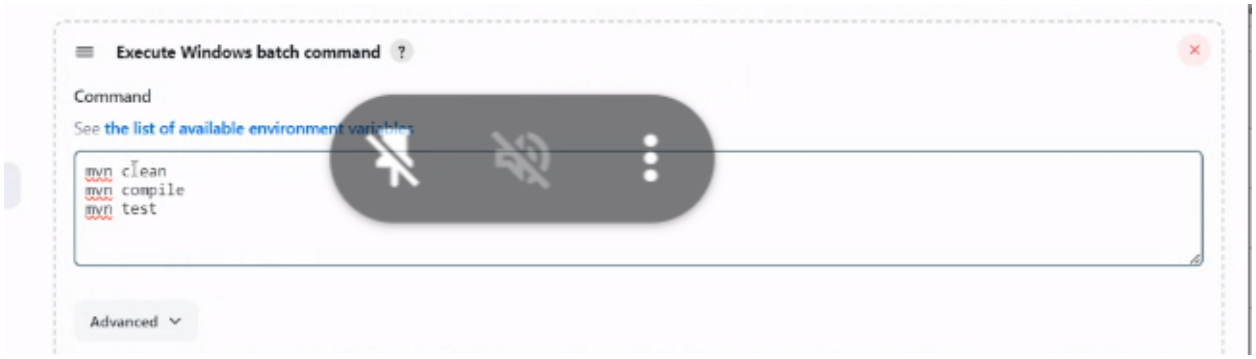
To create a Freestyle Job:

Dashboard -> + New Item -> FreeStyle Project ->Name it and configure

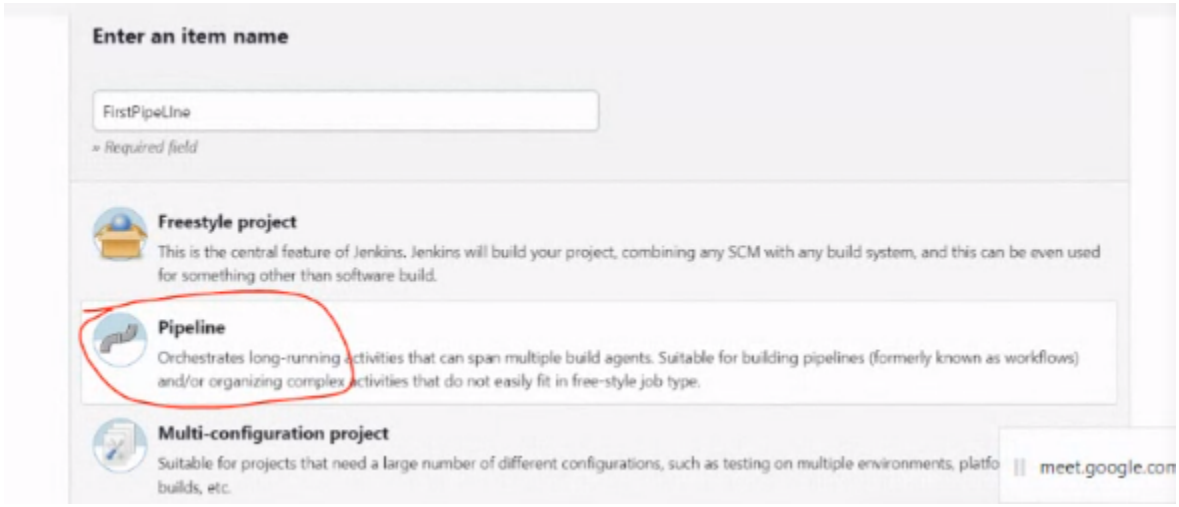




Command in see list of available environment variables:



PipeLine: Scripting language



General

Enabled

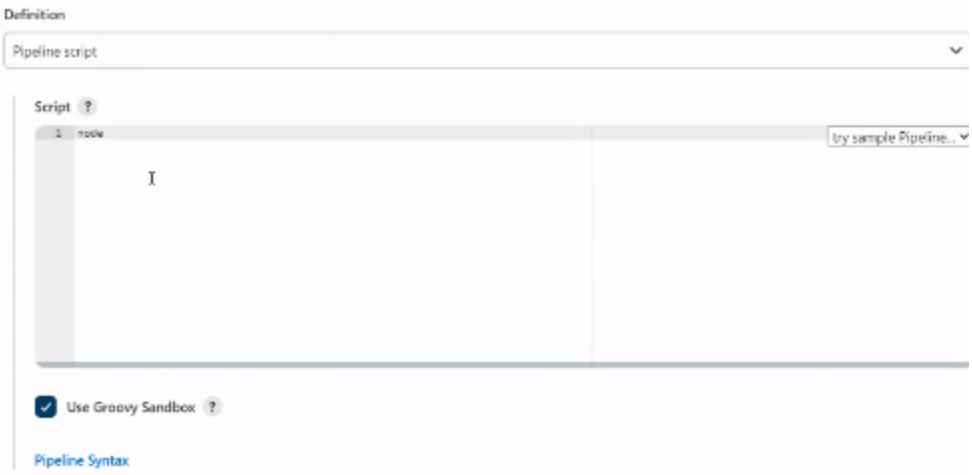
Description

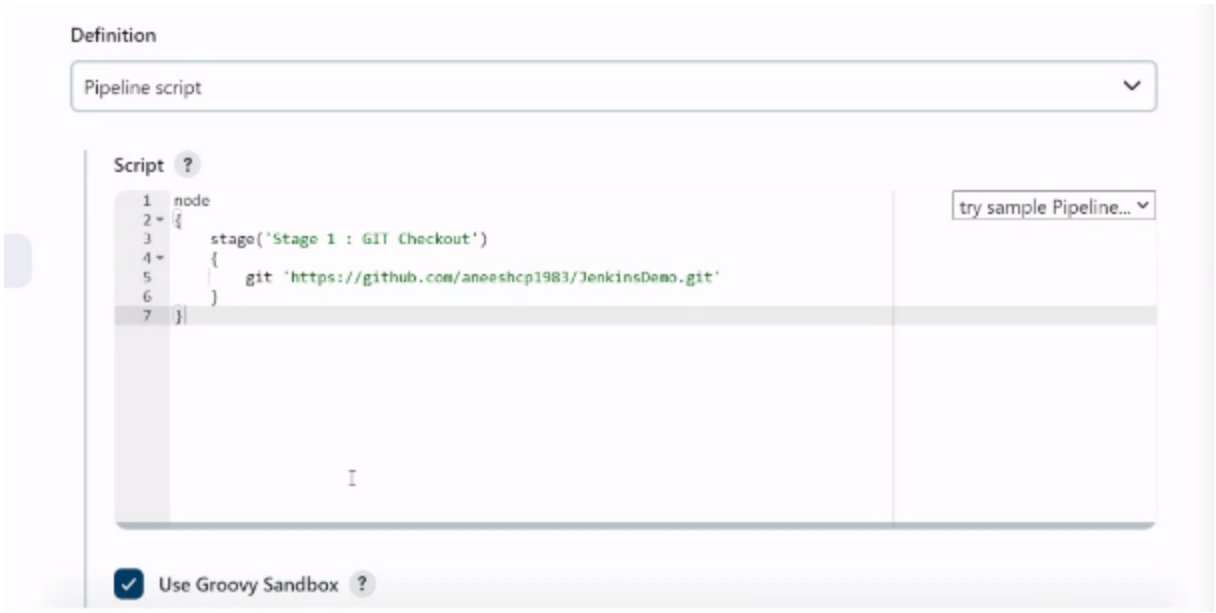
My Frist Pipeline project

Plain text Preview

- [ ] Discard old builds ?
- [ ] Do not allow concurrent builds
- [ ] Do not allow the pipeline to resume if the controller restarts
- [ ] GitHub project
- [ ] Pipeline speed/durability override ?
- [ ] Preserve stashes from completed builds ?
- [ ] This project is parameterized ?
- [ ] Throttle builds ?

Build Triggers

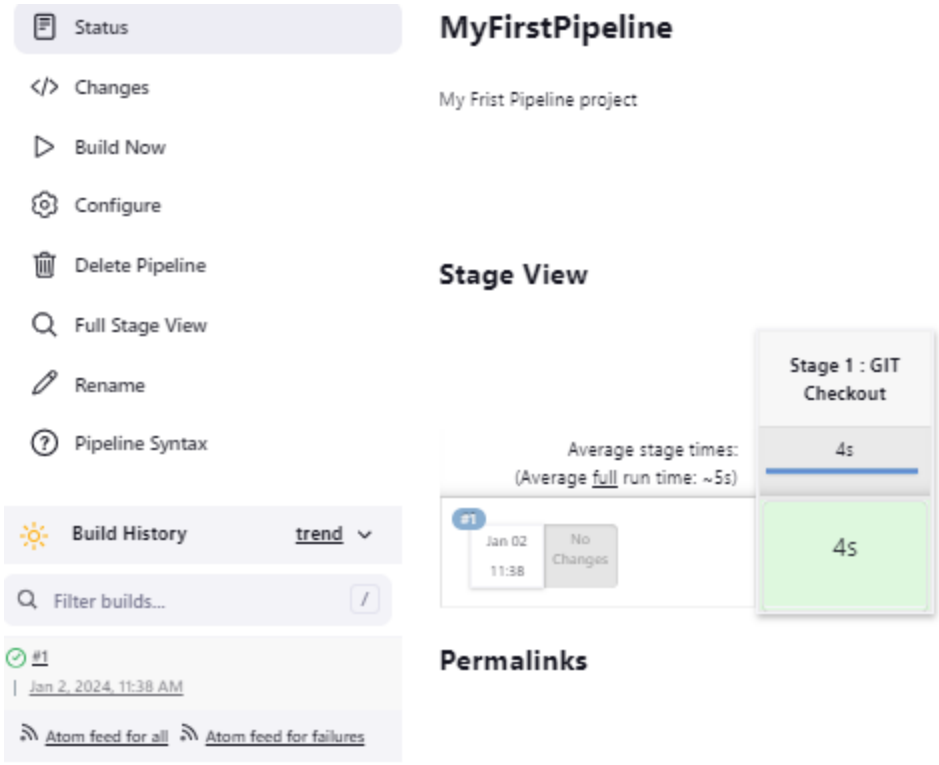




Apply -Save:

—----------

1 Stage: Build Now: Successfully:



2 Stage:

Script ?

```
node
{
    stage('Stage 1 : GIT Checkout')
    {
        git 'https://github.com/Muthu0706/JenkinsDemo.git'
    }
     stage('Stage 2 : Build The Project')
    {
      bat 'mvn install'
    }
}
```

3 Stage:

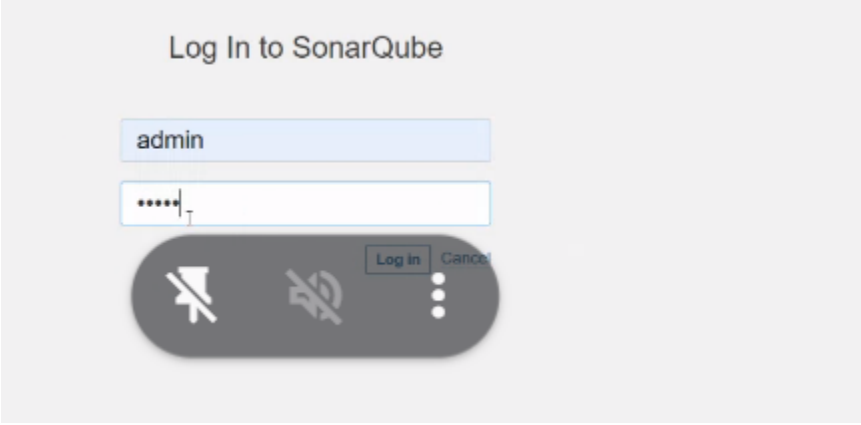
FROM tomcat:9.0
ADD **/*.war /usr/local/tomcat/webapps/
EXPOSE 8088
CMD ["catalina.sh", "run"]

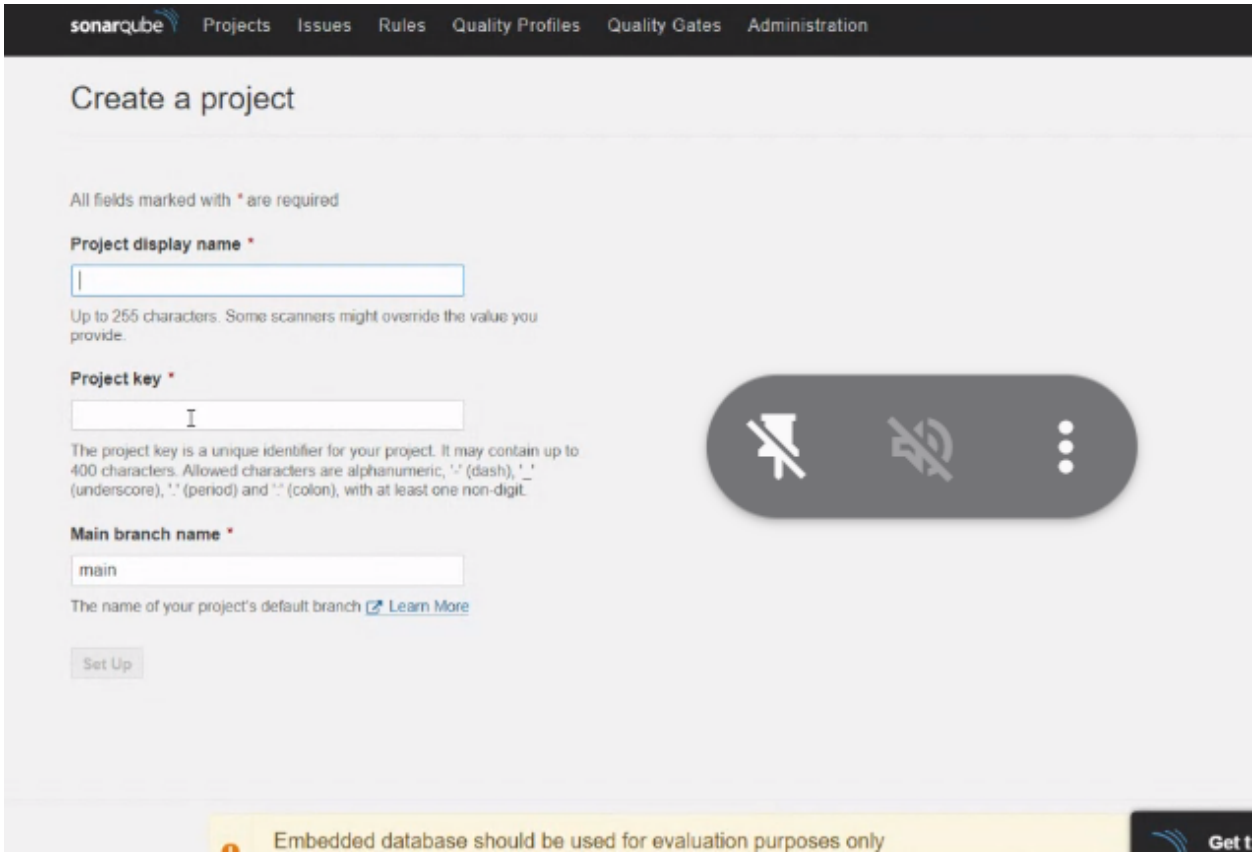
Script ?

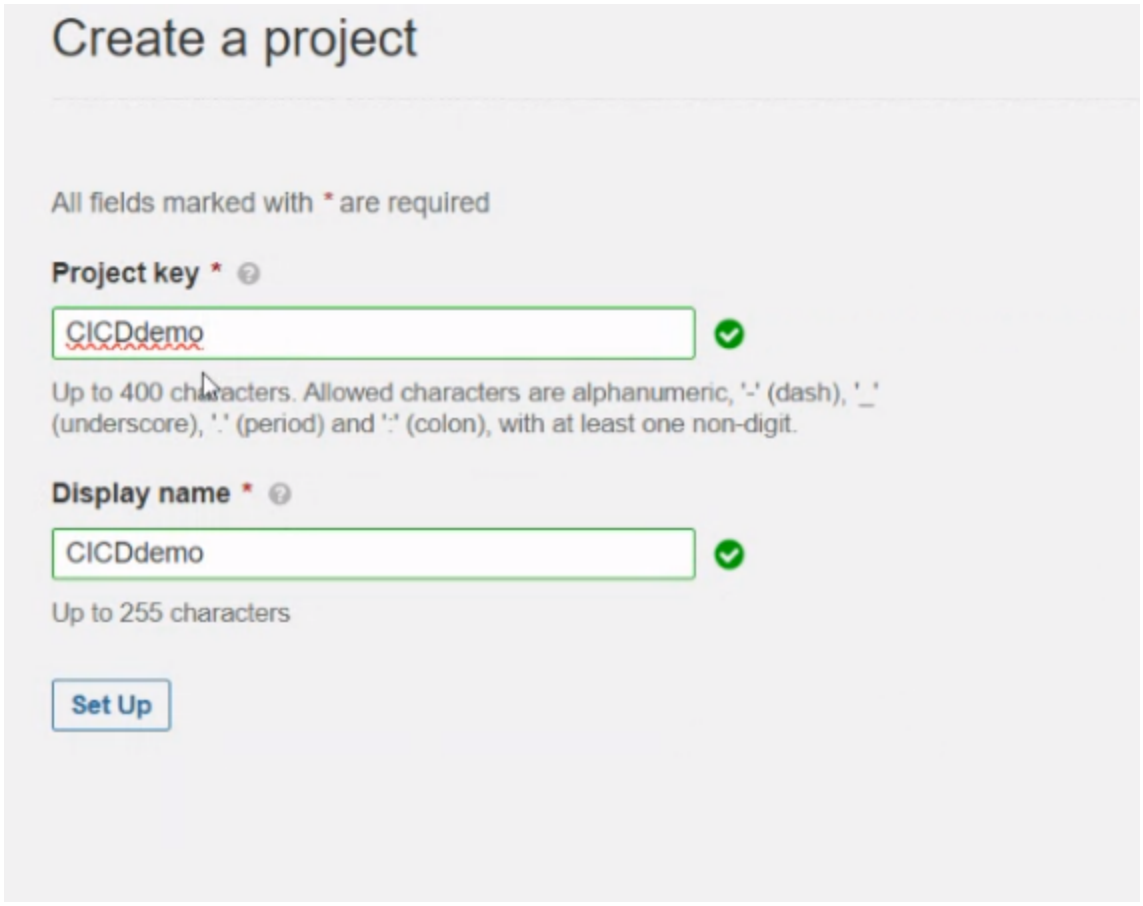
```
stage('Stage 1 : GIT Checkout')
{
    git 'https://github.com/Muthu0706/JenkinsDemo.git'
}
 stage('Stage 2 : Build The Project')
{
  bat 'mvn install'
}
 stage('Stage 3 : Run the Test Cases')
{
  bat 'mvn test'
}
 stage('Stage 4 : Deploy project in Docker')
{
  bat 'docker image build --tag cicddemo .'
}
```

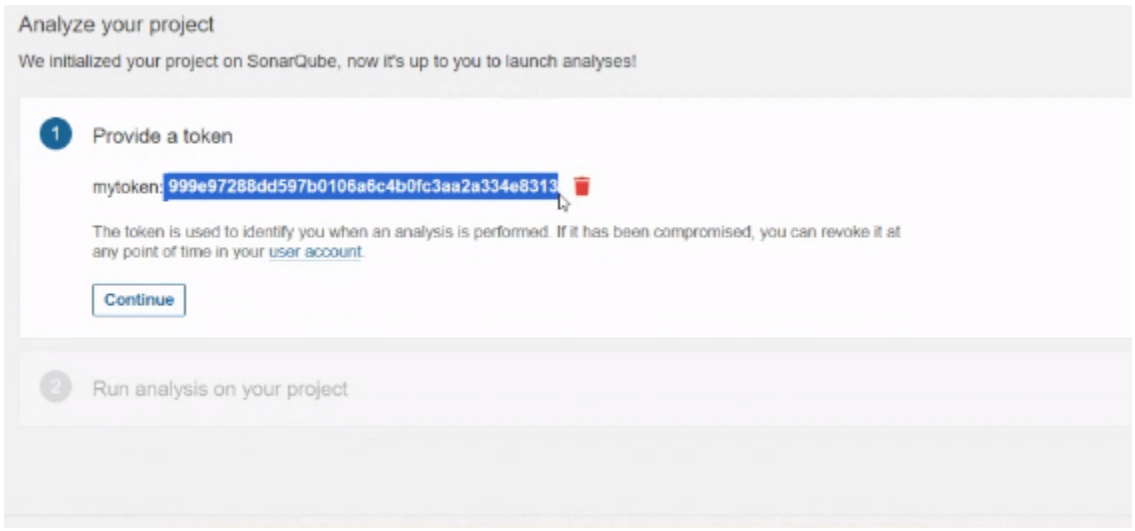
Sonar: localhost:9000
This error:

```
Microsoft Windows [Version 10.0.19045.3803]
(c) Microsoft Corporation. All rights reserved.

D:\Tools\sonarqube-9.9.3.79811\sonarqube-9.9.3.79811\bin\windows-x86-64>StartSonar.bat
Starting SonarQube...
2024.01.02 12:05:53 INFO  app[][o.s.a.AppFileSystem] Cleaning or creating temp directory D:\Tools\sonarqube-9.9.3.79811\sonarqube-9.9.3.79811\temp
2024.01.02 12:05:53 INFO  app[][o.s.a.es.EsSettings] Elasticsearch listening on [HTTP: 127.0.0.1:9001, TCP: 127.0.0.1:59889]
2024.01.02 12:05:53 INFO  app[][o.s.a.ProcessLauncherImpl] Launch process[ELASTICSEARCH] from [D:\Tools\sonarqube-9.9.3.79811\sonarqube-9.9.3.79811\elasticsearch]: C:\Program Files\Java\jdk-17
\bin\java -XX:+UseG1GC -Djava.io.tmpdir=D:\Tools\sonarqube-9.9.3.79811\sonarqube-9.9.3.79811\temp -XX:ErrorFile=D:\Tools\sonarqube-9.9.3.79811\sonarqube-9.9.3.79811\logs\es_hs_err_pid%p.log -D
es.networkaddress.cache.ttl=60 -Des.networkaddress.cache.negative.ttl=10 -XX:+AlwaysPreTouch -Xss1m -Djava.awt.headless=true -Dfile.encoding=UTF-8 -Djna.nosys=true -Djna.tmpdir=D:\Tools\sonarq
ube-9.9.3.79811\sonarqube-9.9.3.79811\temp -XX:-OmitStackTraceInFastThrow -Dio.netty.noUnsafe=true -Dio.netty.noKeySetOptimization=true -Dio.netty.recycler.maxCapacityPerThread=0 -Dio.netty.al
locator.numDirectArenas=0 -Dlog4j.shutdownHookEnabled=false -Dlog4j2.disable.jmx=true -Dlog4j2.formatMsgNoLookups=true -Djava.locale.providers=COMPAT -Dcom.redhat.fips=false -Xmx512m -Xms512m
-XX:MaxDirectMemorySize=256m -XX:+HeapDumpOnOutOfMemoryError -Delasticsearch -Des.path.home=D:\Tools\sonarqube-9.9.3.79811\sonarqube-9.9.3.79811\elasticsearch -Des.path.conf=D:\Tools\sonarqube
-9.9.3.79811\sonarqube-9.9.3.79811\temp\conf\es -cp lib/* org.elasticsearch.bootstrap.Elasticsearch
2024.01.02 12:05:53 INFO  app[][o.s.a.SchedulerImpl] Waiting for Elasticsearch to be up and running
2024.01.02 12:06:19 INFO  app[][o.s.a.SchedulerImpl] Process[es] is up
2024.01.02 12:06:21 INFO  app[][o.s.a.ProcessLauncherImpl] Launch process[WEB_SERVER] from [D:\Tools\sonarqube-9.9.3.79811\sonarqube-9.9.3.79811]: C:\Program Files\Java\jdk-17\bin\java -Djava.
awt.headless=true -Dfile.encoding=UTF-8 -Djava.io.tmpdir=D:\Tools\sonarqube-9.9.3.79811\sonarqube-9.9.3.79811\temp -XX:-OmitStackTraceInFastThrow --add-opens=java.base/java.util=ALL-UNNAMED --
add-opens=java.base/java.lang=ALL-UNNAMED --add-opens=java.base/java.io=ALL-UNNAMED --add-opens=java.rmi/sun.rmi.transport=ALL-UNNAMED --add-exports=java.base/jdk.internal.ref=ALL-UNNAMED --ad
d-opens=java.base/java.nio=ALL-UNNAMED --add-opens=java.base/sun.nio.ch=ALL-UNNAMED --add-opens=java.management/sun.management=ALL-UNNAMED --add-opens=jdk.management/com.sun.management.interna
l=ALL-UNNAMED -Dcom.redhat.fips=false -Xmx512m -Xms128m -XX:+HeapDumpOnOutOfMemoryError -Dhttp.nonProxyHosts=localhost|127.*|[::1] -cp ./lib/sonar-application-9.9.3.79811.jar;D:\Tools\sonarqub
e-9.9.3.79811\sonarqube-9.9.3.79811\lib\jdbc\h2\h2-2.1.214.jar org.sonar.server.app.WebServer D:\Tools\sonarqube-9.9.3.79811\sonarqube-9.9.3.79811\temp\sq-process15839125436701066062properties

WARNING: A terminally deprecated method in java.lang.System has been called
WARNING: System::setSecurityManager has been called by org.sonar.process.PluginSecurityManager (file:/D:/Tools/sonarqube-9.9.3.79811/sonarqube-9.9.3.79811/lib/sonar-application-9.9.3.79811.jar
)
WARNING: Please consider reporting this to the maintainers of org.sonar.process.PluginSecurityManager
WARNING: System::setSecurityManager will be removed in a future release
2024.01.02 12:06:22 INFO  app[][o.s.a.SchedulerImpl] Process[Web Server] is stopped
2024.01.02 12:06:22 INFO  app[][o.s.a.SchedulerImpl] Process[ElasticSearch] is stopped
2024.01.02 12:06:22 INFO  app[][o.s.a.SchedulerImpl] SonarQube is stopped

D:\Tools\sonarqube-9.9.3.79811\sonarqube-9.9.3.79811\bin\windows-x86-64>SonarService.bat
```

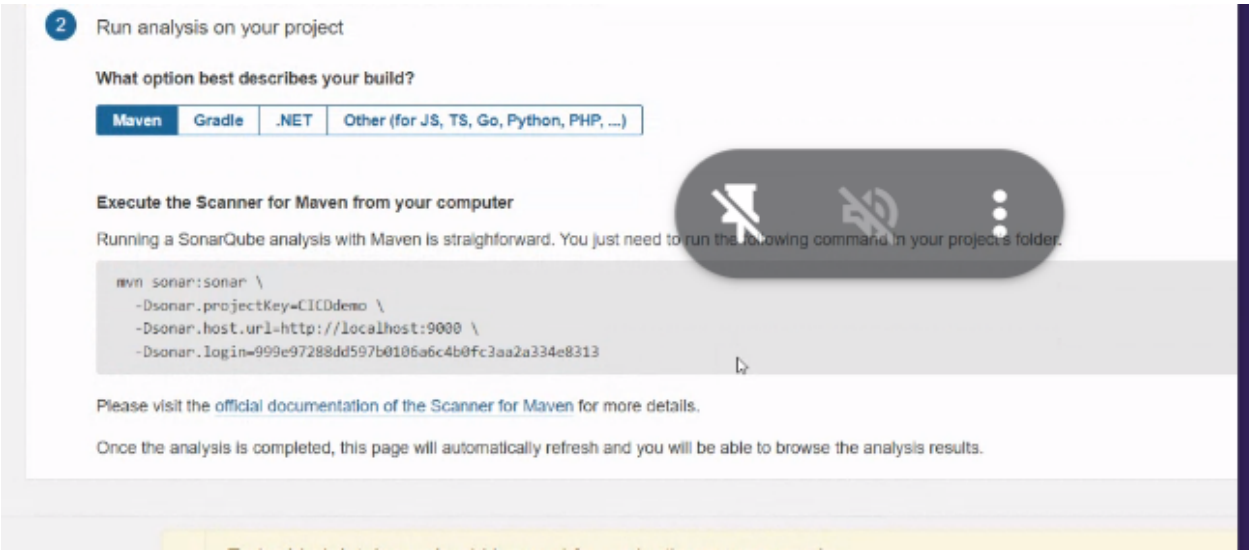


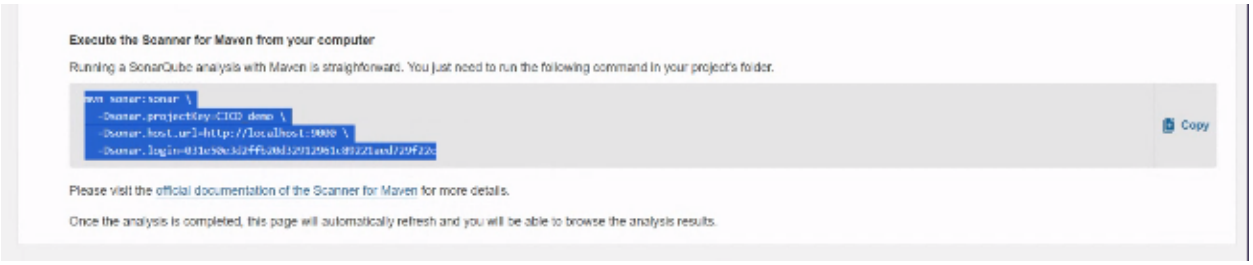

sonarqube
Projects
Issues
Rules
Quality Profiles
Quality Gates
Administration
Create a project
All fields marked with * are required
Project display name *
Up to 255 characters. Some scanners might override the value you provide.
Project key *
The project key is a unique identifier for your project. It may contain up to 400 characters. Allowed characters are alphanumeric, '-' (dash), '_' (underscore), '.' (period) and ':' (colon), with at least one non-digit.
Main branch name *
main
The name of your project's default branch
Learn More
Set Up
Embedded database should be used for evaluation purposes only





Continue click:



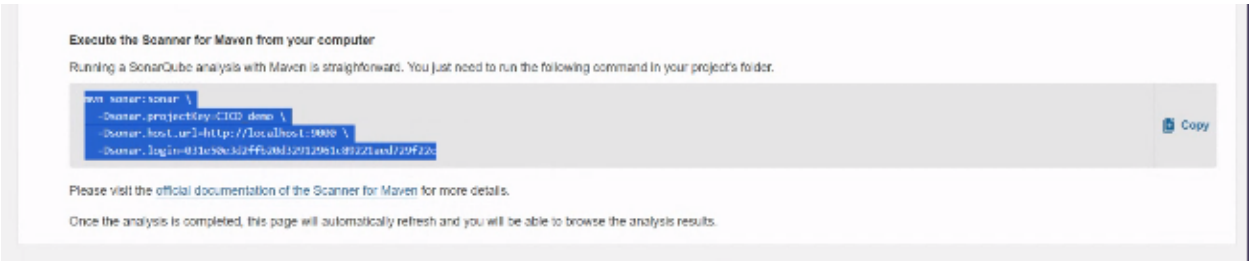


Copy this .

Next Open Jenkins:

Stage 4:

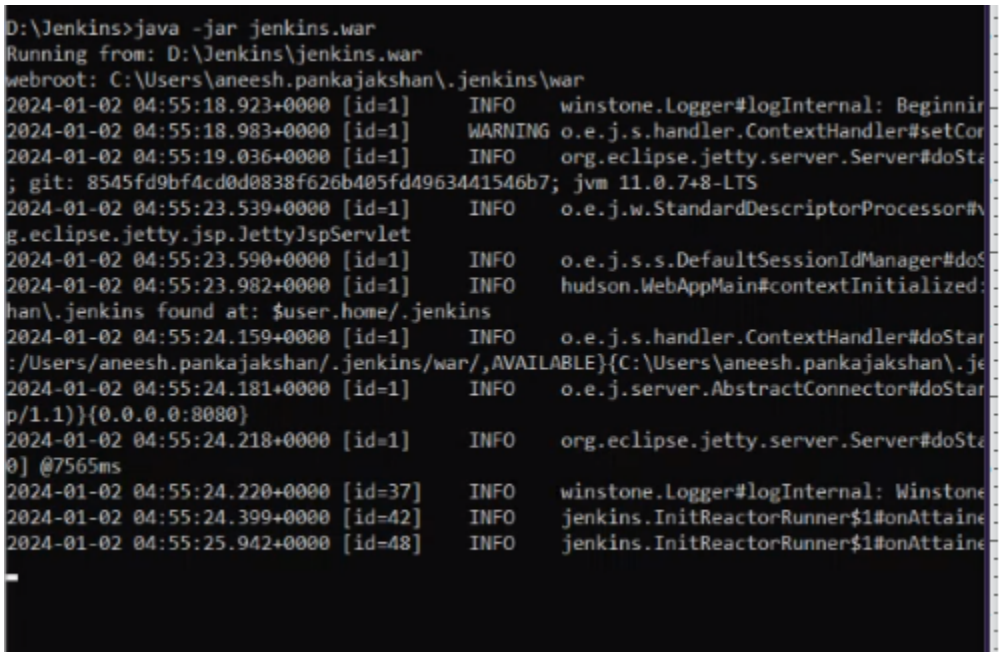
stage('stage 4: Run Quality Test using Sonar')
{
bat 'mvn sonar:sonar



}
Next Apply click Save:

Build Now: install

Docker:



Jenkins is a Java-based open-source automation platform with plugins designed for continuous integration. It is used to continually create and test software projects, making it easier for developers and DevOps engineers to integrate changes to the project and for consumers to get a new one.

MAVEN: 12-27-2023
Jar-Java ARchive.
WAR file (Web Application Resource or Web application ARchive).

UTF8- stands for Unicode Transformation Format.

LocalRepository

How do I use mvn command?

2. Maven Commands

1. mvn compile: This command is used to compile the project's source code.
2. mvn clean: Here, the project is cleaned to remove all previous-build files generated.
3. mvn test: With this command, one can run project testing steps.
4. mvn test-compile: ...
5. mvn install: ...
6. mvn package: ...
7. mvn deploy:

Maven : Build Lifecycle

Maven follows a predefined sequence of phases and goals to manage the software build process. The default build life cycle phases are :

- validate: Validates the project's structure and dependencies.
- compile: Compiles the source code.
- test: Executes tests using a testing framework.
- package: Packages the compiled code into a distributable format (e.g., JAR, WAR).
- verify: Runs checks on the code to ensure quality criteria.
- install: Installs the packaged artifact into the local repository for use in other projects.
- deploy: Copies the final package to the remote repository for sharing with other developers or projects.

Goals: Specific tasks executed during each phase (e.g., compile, test, install).

Maven : Plugins

Maven plugins are extensions that enhance Maven's functionalities and enable additional capabilities beyond its core features. These plugins are essential in executing specific tasks during the build lifecycle. They offer a way to customize and extend Maven's behavior to suit the project's requirements. Types of plugins are

Core Plugins:

- Bundled with Maven by default, covering essential functionalities like compiling, testing, packaging, and deploying.

Standard Plugins:

- Offer additional capabilities and are commonly used across various projects (e.g., maven-compiler-plugin, maven-surefire-plugin for testing).

Custom or Third-Party Plugins:

- Developed by the community or specific to certain projects to address unique requirements.
- These plugins cater to specific needs beyond what the standard plugins offer.

Maven : Dependency Management

Maven's dependency management is a pivotal feature that simplifies the handling of external libraries or dependencies within a project. It streamlines the process of declaring, downloading, and managing dependencies, ensuring consistent and reliable builds by managing versions, resolving conflicts, and fetching required libraries.

Which includes:

- POM Definitions: Dependencies specified within the POM.
- Transitive Dependencies: Automatically included dependencies required by declared dependencies.
- Conflict Resolution: Resolving conflicts among different versions of the same library.

1.mvn –version

```
D:\MyMavenProject\myproject>mvn --version
Apache Maven 3.9.6 (bc0240f3c744dd6b6ec2920b3cd08dcc295161ae)
Maven home: C:\apache-maven-3.9.6
Java version: 17.0.9, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-17
Default locale: en_US, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

2. mvn clean

```
D:\MyMavenProject\myproject>mvn clean
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------< com.rts:myproject >-------------------------
[INFO] Building myproject 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- clean:3.2.0:clean (default-clean) @ myproject ---
[INFO] Deleting D:\MyMavenProject\myproject\target
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  0.671 s
[INFO] Finished at: 2023-12-27T12:02:00+05:30
[INFO] ------------------------------------------------------------------------
```

3.mvn compile

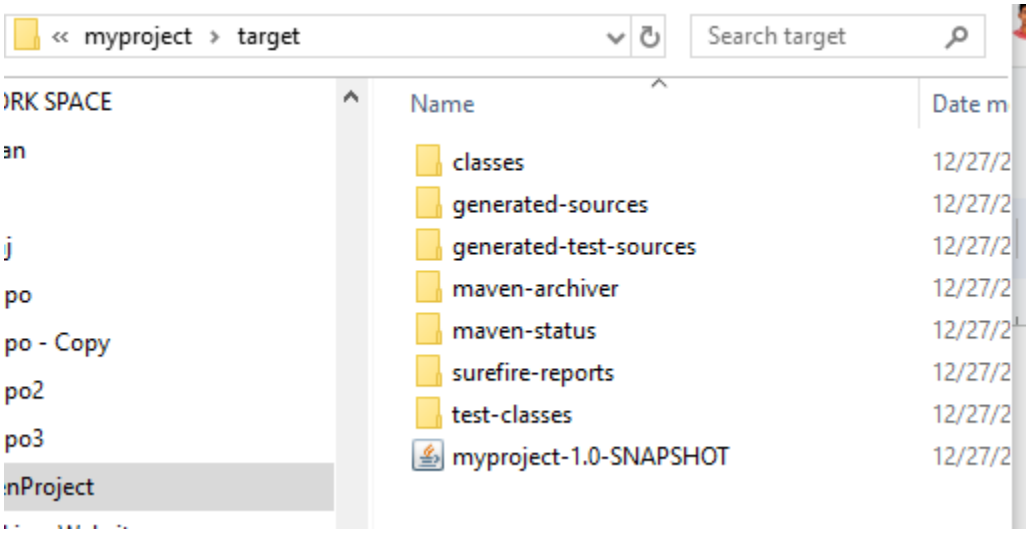
```
D:\MyMavenProject\myproject>mvn compile
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------< com.rts:myproject >-------------------------
[INFO] Building myproject 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ myproject ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is pl
atform dependent!
[INFO] skip non existing resourceDirectory D:\MyMavenProject\myproject\src\main\resources
[INFO]
[INFO] --- compiler:3.11.0:compile (default-compile) @ myproject ---
[INFO] Changes detected - recompiling the module! :input tree
[WARNING] File encoding has not been set, using platform encoding Cp1252, i.e. build is platform
 dependent!
[INFO] Compiling 2 source files with javac [debug target 1.8] to target\classes
[WARNING] bootstrap class path not set in conjunction with -source 8
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.790 s
[INFO] Finished at: 2023-12-27T12:07:28+05:30
```

4.mvn test

```
INFO]
INFO] -------------------------------------------------------
INFO]  T E S T S
INFO] -------------------------------------------------------
INFO] Running com.rts.AppTest
INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.014 s -- in com.rts.
est
INFO]
INFO] Results:
INFO]
INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
INFO]
INFO] ------------------------------------------------------------------------
INFO] BUILD SUCCESS
INFO] ------------------------------------------------------------------------
INFO] Total time:  12.999 s
INFO] Finished at: 2023-12-27T12:12:19+05:30
INFO] ------------------------------------------------------------------------
```

5.mvn package

```
:\MyMavenProject\myproject>mvn package
INFO] Scanning for projects...
INFO]
INFO] -------------------------< com.rts:myproject >--------------------------
INFO] Building myproject 1.0-SNAPSHOT
INFO]   from pom.xml
INFO] --------------------------------[ jar ]---------------------------------
INFO]
INFO] --- resources:3.3.1:resources (default-resources) @ myproject ---
WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is pl
tform dependent!
INFO] skip non existing resourceDirectory D:\MyMavenProject\myproject\src\main\resources
INFO]
INFO] --- compiler:3.11.0:compile (default-compile) @ myproject ---
INFO] Nothing to compile - all classes are up to date
INFO]
INFO] --- resources:3.3.1:testResources (default-testResources) @ myproject ---
WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is pl
```



6.mvn install

Before:

```
D:\MyMavenProject\myproject>mvn clean
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------< com.rts:myproject >-------------------------
[INFO] Building myproject 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- clean:3.2.0:clean (default-clean) @ myproject ---
[INFO] Deleting D:\MyMavenProject\myproject\target
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  0.397 s
[INFO] Finished at: 2023-12-27T12:19:34+05:30
[INFO] ------------------------------------------------------------------------
```

After:

```
D:\MyMavenProject\myproject>mvn install
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------< com.rts:myproject >-------------------------
[INFO] Building myproject 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ myproject ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is pl
atform dependent!
[INFO] skip non existing resourceDirectory D:\MyMavenProject\myproject\src\main\resources
[INFO]
[INFO] --- compiler:3.11.0:compile (default-compile) @ myproject ---
[INFO] Changes detected - recompiling the module! :source
[WARNING] File encoding has not been set, using platform encoding Cp1252, i.e. build is platform
 dependent!
[INFO] Compiling 2 source files with javac [debug target 1.8] to target\classes
[WARNING] bootstrap class path not set in conjunction with -source 8
[INFO]
[INFO] --- resources:3.3.1:testResources (default-testResources) @ myproject ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is pl
atform dependent!
```

mvn dependency :tree

```
D:\MyMavenProject\myproject>mvn dependency:tree
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------< com.rts:myproject >-------------
[INFO] Building myproject 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------
[INFO]
[INFO] --- dependency:3.6.1:tree (default-cli) @ myproject ---
```

Maven : Commands

Maven offers a variety of commands to manage and build projects. Here are so commonly used Maven commands:

Project Building and Lifecycle Execution:

mvn clean:

- Cleans the project by removing the target directory and all build artifacts.

mvn compile:

- Compiles the source code of the project.

mvn test:

- Runs unit tests using a testing framework (like JUnit).

Maven : Commands

mvn package:

- Packages the compiled code into a distributable format (JAR, WAR, etc.).

mvn install:

- Installs the packaged artifact into the local repository, making it available for other local projects as a dependency.

mvn deploy:

- Deploys the built artifact to a remote repository for sharing it with other developers or projects.

Maven : Best Practices

Maven offers several best practices that help streamline the development process, ensure consistency, and enhance project maintainability. Here are some key Maven best practices:

1. **Follow Standard Project Structure:**
   Adhere to Maven standard directory structure (src/main/java, src/test/java, etc.) for consistency and ease of navigation.

2. **Use Descriptive Artifact IDs and Group IDs:**
   Artifact IDs and Group IDs should be meaningful, reflecting the project's purpose and ownership.

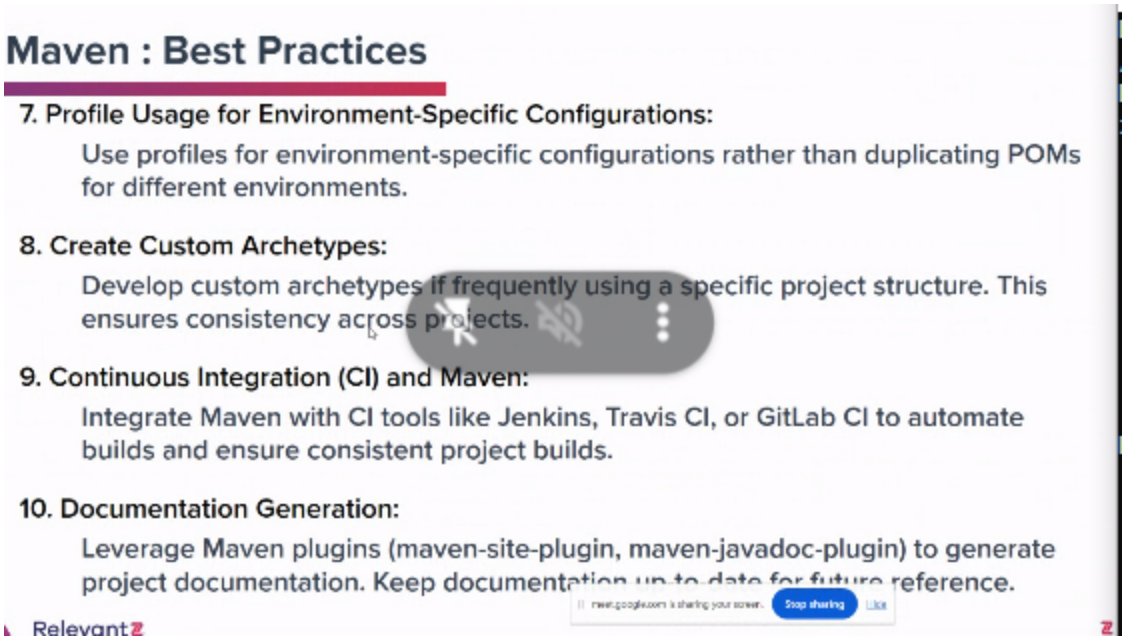
3. **Maintain Clean and Concise POMs:**
   Avoid unnecessary complexity in the POM (Project Object Model) file. Keep it concise and readable.

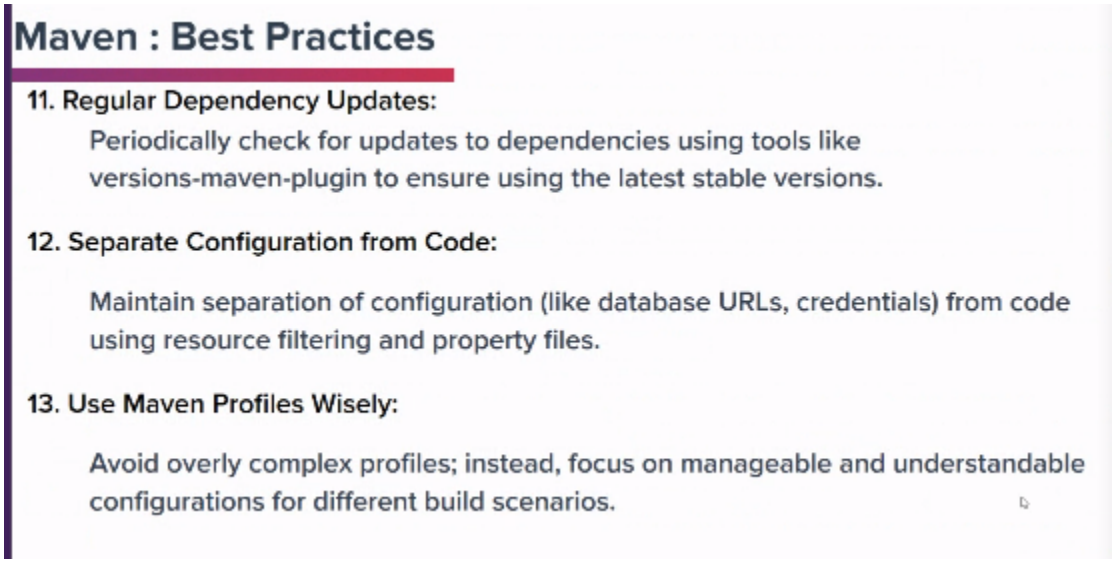
Maven : Best Practices

4. **Leverage Dependency Management:**
   Use Maven's dependency management to handle project dependencies. Clearly define dependencies and versions to ensure consistency.

5. **Version Control .gitignore and pom.xml:**
   Ensure that generated files, build artifacts, and IDE-specific files are included in .gitignore.Keep the pom.xml under version control and commit changes when adding dependencies or modifying configurations.

6. **Avoid Direct Use of Repository URLs:**
   Minimize the direct use of repository URLs in the POM. Rely on repository declarations in settings.xml or repository management tools.





What is Maven:

Maven is a popular open-source build tool developed by the Apache Group to build, publish, and deploy several projects at once for better project management.

How does DevOps work?

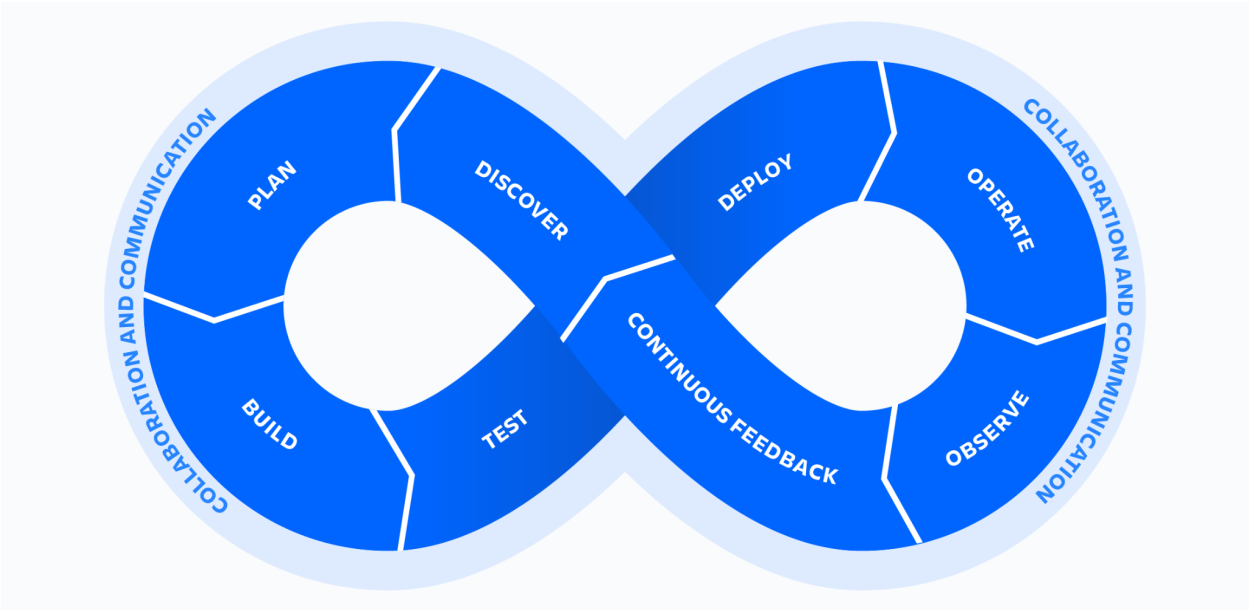
****

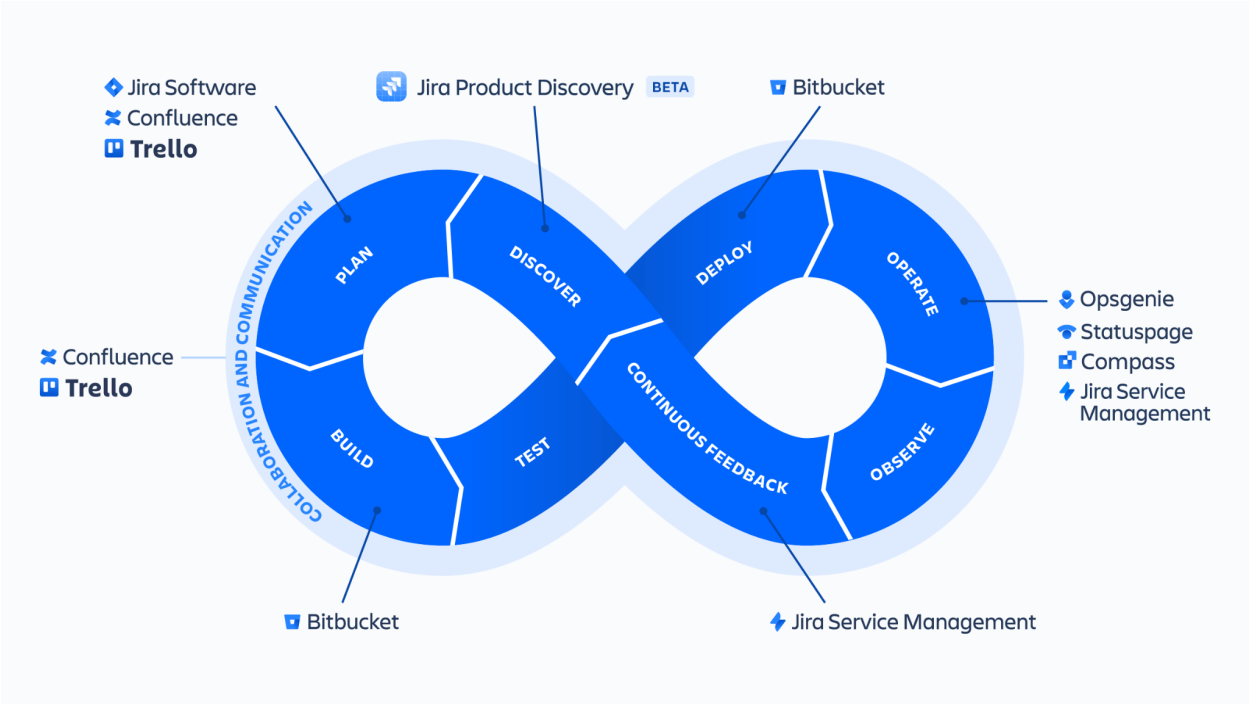
A DevOps team includes developers and IT operations working collaboratively throughout the product life cycle, in order to increase the speed and quality of software deployment. It's a new way of working, a cultural shift, that has significant implications for teams and the organizations they work for.

Under a DevOps model, development and operations teams are no longer "siloed." Sometimes, these two teams merge into a single team where the engineers work across the entire application lifecycle — from development and test to deployment and operations — and have a range of multidisciplinary skills.

DevOps teams use tools to automate and accelerate processes, which helps to increase reliability. A DevOps toolchain helps teams tackle important DevOps fundamentals including continuous integration, continuous delivery, automation, and collaboration.

DevOps values are sometimes applied to teams other than development. When security teams adopt a DevOps approach, security is an active and integrated part of the development process. This is called DevSecOps.





Bitbucket :

If you boil it down to the most basic difference between GitHub and Bitbucket, it is that **GitHub is focused around public code and Bitbucket is for private**. GitHub has a huge open-source community and Bitbucket tends to have mostly enterprise and business users.

What are the benefits of DevOps?

In Atlassian's 2020 DevOps Trends survey, 99 percent of respondents said that DevOps had a positive impact on their organization. The benefits of DevOps include faster and easier releases, team efficiency, increased security, higher quality products, and consequently happier teams and customers.

Speed

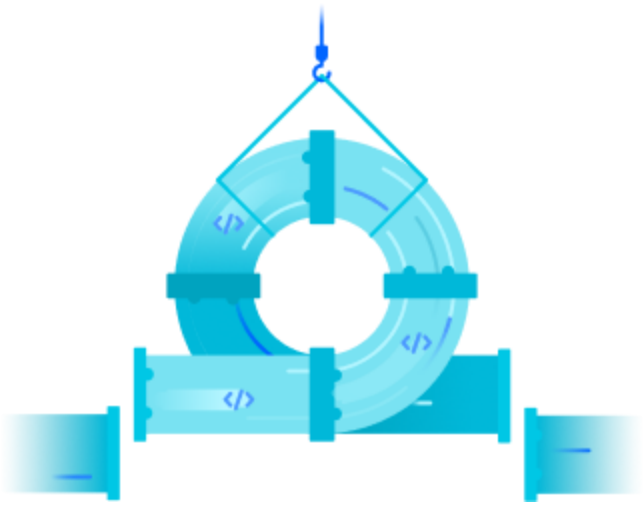
Teams that practice DevOps release deliverables more frequently, with higher quality and stability. In fact, the DORA 2019 State of DevOps report found that elite teams deploy 208 times more frequently and 106 times faster than low-performing teams. Continuous delivery allows teams to build, test, and deliver software with automated tools.

Improved collaboration

The foundation of DevOps is a culture of collaboration between developers and operations teams, who share responsibilities and combine work. This makes teams more efficient and saves time related to work handoffs and creating code that is designed for the environment where it runs.

Rapid deployment

By increasing the frequency and velocity of releases, DevOps teams improve products rapidly. A competitive advantage can be gained by quickly releasing new features and repairing bugs.



Quality and reliability

Practices like continuous integration and continuous delivery ensure changes are functional and safe, which improves the quality of a software product. Monitoring helps teams keep informed of performance in real-time.

Security

By integrating security into a continuous integration, continuous delivery, and continuous deployment pipeline, DevSecOps is an active, integrated part of the development process. Security is built into the product by integrating active security audits and security testing into agile development and DevOps workflows.

What are the challenges of adopting DevOps?

Habits are hard to break. Teams entrenched in siloed ways of working can struggle with, or even be resistant to, overhauling team structures to embrace DevOps practices. Some teams may mistakenly believe new tools are sufficient to adopt DevOps. Yet, DevOps is a combination of people, tools, and culture. Everyone on a DevOps team must understand the entire value stream — from ideation, to development, to the end user experience. It requires breaking down silos in order to collaborate throughout the product life cycle.

> Devops isn't any single person's job. It's everyone's job.
>
> **Robert Krohn**
>
> HEAD OF ENGINEERING, DEVOPS AT ATLASSIAN

Moving from a legacy infrastructure to using Infrastructure as Code (IaC) and microservices can offer faster development and innovation, but the increased operational workload can be challenging. It's best to build out a strong foundation of automation, configuration management, and continuous delivery practices to help ease the load.

An over-reliance on tools can detract teams from the necessary foundations of DevOps: the team and organization structure. Once a structure is established, the processes and team should come next and the tools should follow.

How to adopt DevOps

Adopting DevOps first requires a commitment to evaluating and possibly changing or removing any teams, tools, or processes your organization currently uses. It means building the necessary infrastructure to give teams the autonomy to build, deploy, and manage their products without having to rely too heavily on external teams.

DevOps culture

A DevOps culture is where teams embrace new ways of working that involve greater collaboration and communication. It's an alignment of people, processes, and tools toward a more unified customer focus. Multidisciplinary teams take accountability for the entire lifecycle of a product.

Continuous learning

Organizations that do DevOps well are places where experimentation and some amount of risk-taking are encouraged. Where thinking outside the box is the norm, and failure is understood to be a natural part of learning and improving.

Agile

Agile methodologies are immensely popular in the software industry since they empower teams to be inherently flexible, well-organized, and capable of responding to change. DevOps is a cultural shift that fosters collaboration between those who build and maintain software. When used together, agile and DevOps result in high efficiency and reliability.

DevOps practices

Continuous integration

Continuous integration is the practice of automating the integration of code changes into a software project. It allows developers to frequently merge code changes into a central repository where builds and tests are executed. This helps

DevOps teams address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates.

Learn about continuous integration

Continuous delivery

Continuous delivery expands upon continuous integration by automatically deploying code changes to a testing/production environment. It follows a continuous delivery pipeline, where automated builds, tests, and deployments are orchestrated as one release workflow.

Learn about continuous delivery

Situational awareness

It is vital for every member of the organization to have access to the data they need to do their job as effectively and quickly as possible. Team members need to be alerted of failures in the deployment pipeline — whether systemic or due to failed tests — and receive timely updates on the health and performance of applications running in production. Metrics, logs, traces, monitoring, and alerts are all essential sources of feedback teams need to inform their work.

Learn about DevOps metrics

Automation

Automation is one of the most important DevOps practices because it enables teams to move much more quickly through the process of developing and deploying high-quality software. With automation the simple act of pushing code changes to a source code repository can trigger a build, test, and deployment process that significantly reduces the time these steps take.

Learn about DevOps automation best practices

Infrastructure as Code

Whether your organization has an on-premise data center or is completely in the cloud, having the ability to quickly and consistently provision, configure, and manage infrastructure is key to successful DevOps adoption. Infrastructure as Code (IaC) goes beyond simply scripting infrastructure configuration to treating your infrastructure definitions as actual code: using source control, code reviews, tests, etc.

Learn about Infrastructure as Code

Microservices

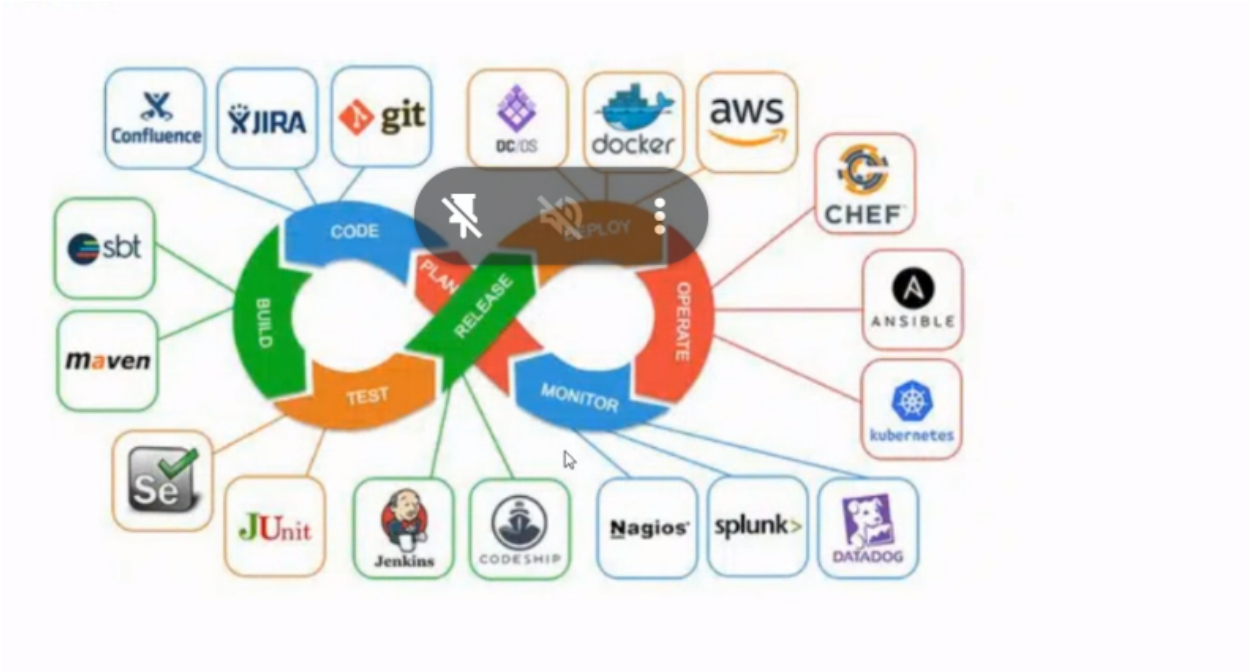
Microservices is an architectural technique where an application is built as a collection of smaller services that can be deployed and operated independently from each other. Each service has its own processes and communicates with other services through an interface. This separation of concerns and decoupled independent function allows for DevOps practices like continuous delivery and continuous integration.

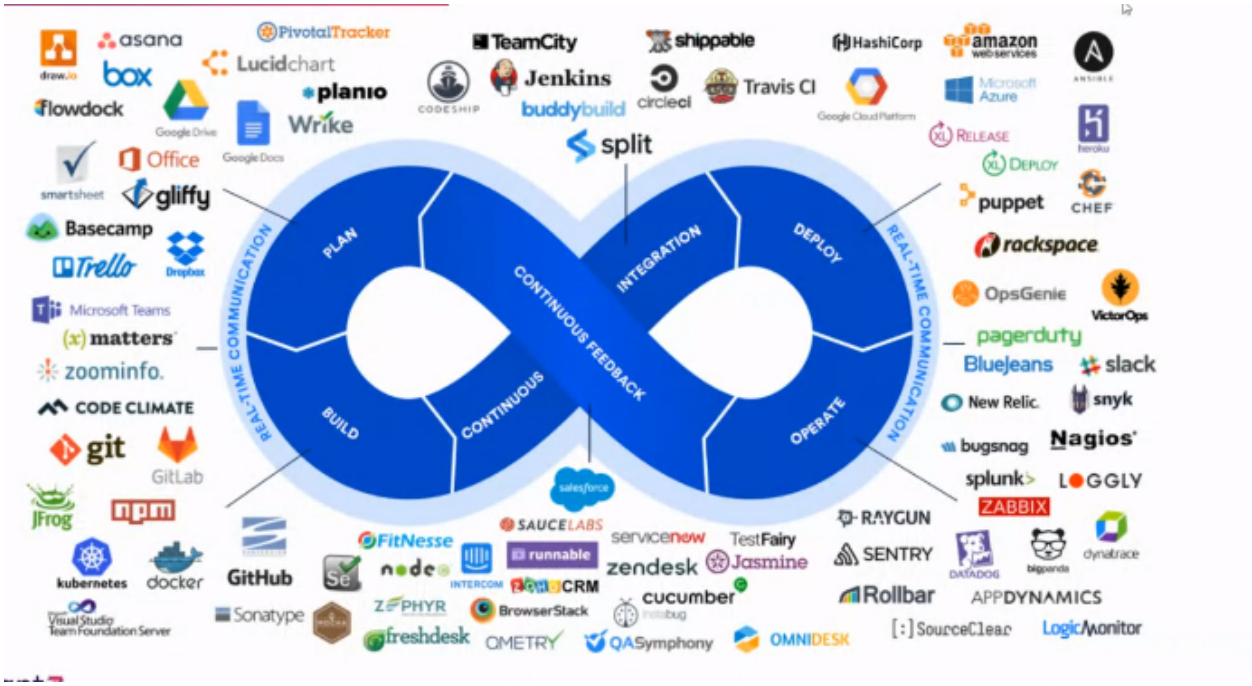
Learn about microservices

Monitoring

DevOps teams monitor the entire development lifecycle — from planning, development, integration and testing, deployment, and operations. This allows teams to respond to any degradation in the customer experience, quickly and automatically. More importantly, it allows teams to "shift left" to earlier stages in development and minimize broken production changes.

What is DevOps?
Why DevOps?
It allows the Development team and the Operation team to directly.Tis thing further avoids delays,fix bugs and deliver quality .
DevOps allows continuous connection
Devops vs Existing

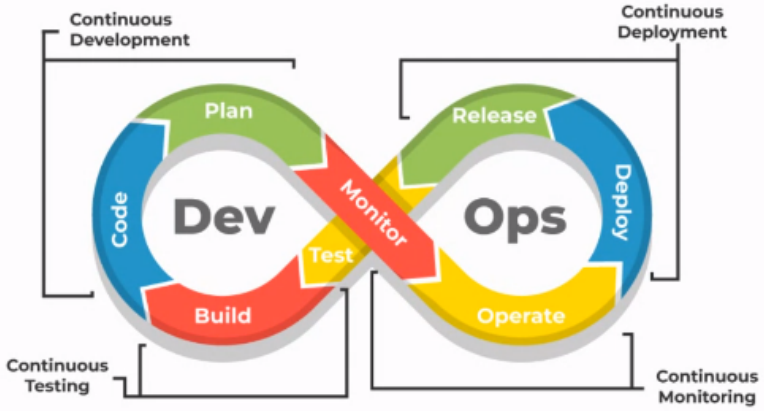




DevOps Architecture

The phases

- Plan
- Code
- Build
- Test
- Release
- Deploy
- Operate
- Monitor



Let's discuss in detail..

Maven:

1.mvn --version -It is checked Build and p

In order to support various aspects of the Devops Lifecycle,various tools are called Devops Toolchain.

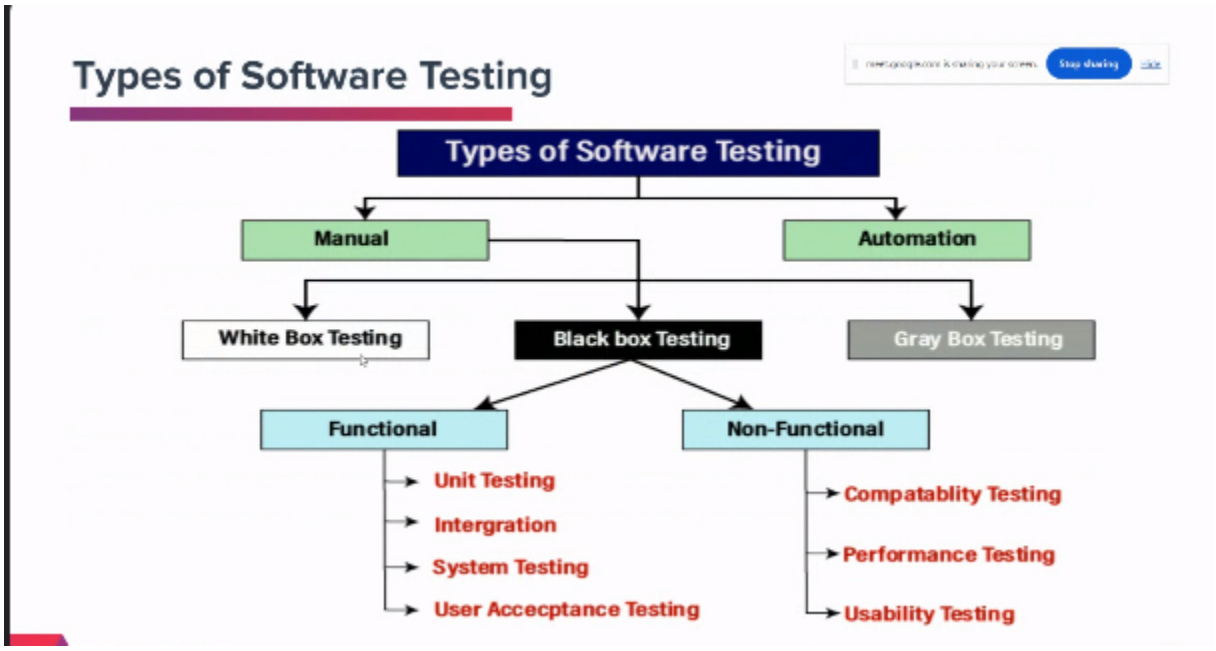
Git is **a DevOps tool used for source code management**. It is a free and open-source version control system used to handle small to very large projects efficiently. Git is used to track changes in the source code, enabling multiple developers to work together on non-linear development.

Software Development Operation(ops)

- **Individuals and interactions over processes and tools.**
- **Working software over comprehensive documentation.**
- **Customer collaboration over contract negotiation.**
- **Responding to change over following a project plan.**

The key difference between Agile and Scrum is that while **Agile is a project management philosophy that utilizes a core set of values or principles, Scrum is a specific Agile methodology that is used to facilitate a project**

What is Software Testing:



Objective of Unit Testing

1. To isolate a section of code.
2. To verify the correctness of the code.
3. To test every function and procedure.
4. To fix bugs early in the development cycle and to save costs.
5. To help the developers understand the code base and enable them to make changes quickly.
6. To help with code reuse

Types of Unit Testing

There are 2 types of Unit Testing:

- Manual Testing , and
- Automated Testing.

Unit Testing Techniques:

There are 3 types of Unit Testing Techniques. They are

1. Black Box Testing: This testing technique is used in covering the unit tests for input, user interface, and output parts.
2. White Box Testing: This technique is used in testing the functional behavior of the system by giving the input and checking the functionality output including the internal design structure and code of the modules.
3. Gray Box Testing: This technique is used in executing the relevant test cases, test methods, and test functions, and analyzing the code performance for the modules.

Unit Testing Tools:

Here are some commonly used Unit Testing tools:

1. Jtest
2. Junit
3. NUnit
4. Jest
5. karma

Java is J-UNIT:

JUnit Jupiter

JUnit Jupiter is a perfect blend of the JUnit 5 programming model and extension model for writing tests and extensions.

The Jupiter sub-project provides a TestEngine for running Jupiter-based tests on the platform.

It also defines the TestEngine API for developing new testing frameworks that run on the platform.

Use Case of JUnit

JUnit is used to do unit testing in Java. A JUnit test case is the set of codes which ensure that our program code works as expected or not.

In Java, two types of unit testing are possible, manual testing and automated testing.

Manual testing is a special type of testing in which test cases are executed without using any equipment.

Unlike manual testing, automated testing is not possible without the support of any equipment.

The org.junit package provides several classes and packages, which help us to ensure whether our code provides the expected output or not.

Advantages of JUnit

- The JUnit framework is open source
- It provides text-based command lines as well as AWT-based and Swing-based graphical test mechanisms
- It has some annotations to utilize test functions
- It has a test runner to test running applications
- It allows you to write code
- It can test automatically and provide feedback

Assert -is compare and

JUnit Assert Methods

Null object

If you want to check the initial value of an object/variable, you have the following methods:

1. assertNull(object)
2. assertNotNull(object)

Here object is Java object e.g. assertNull(actual);

JUnit Annotations

JUNIT 4 ANNOTATION	JUNIT 5 ANNOTATION
@Before	@BeforeEach
@After	@AfterEach
@BeforeClass	@BeforeAll
@AfterClass	@AfterAll
@Test	@Test

JUnit Tutorial | Testing Framework for Java

JUnit tutorial provides basic and advanced concepts of unit testing in java with examples. Our junit tutorial is designed for beginners and professionals.

It is an *open-source testing framework* for java programmers. The java programmer can create test cases and test his/her own code.

It is one of the unit testing framework. Current version is junit 4.

To perform unit testing, we need to create test cases. The unit test case is a code which ensures that the program logic works as expected.

The org.junit package contains many interfaces and classes for junit testing such as Assert, Test, Before, After etc.

Types of unit testing

There are two ways to perform unit testing: 1) manual testing 2) automated testing.

1) Manual Testing

If you execute the test cases manually without any tool support, it is known as manual testing. It is time consuming and less reliable.

2) Automated Testing

If you execute the test cases by tool support, it is known as automated testing. It is fast and more reliable.

Annotations for Junit testing

The Junit 4.x framework is annotation based, so let's see the annotations that can be used while writing the test cases.

@Test annotation specifies that method is the test method.

@Test(timeout=1000) annotation specifies that method will be failed if it takes longer than 1000 milliseconds (1 second).

@BeforeClass annotation specifies that method will be invoked only once, before starting all the tests.

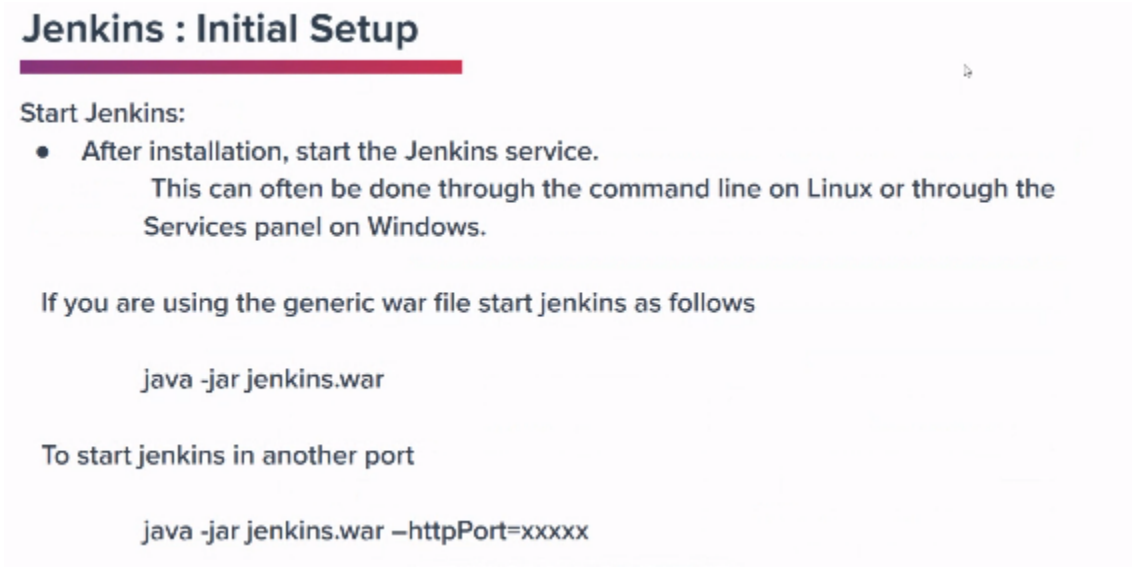
@Before annotation specifies that method will be invoked before each test.

@After annotation specifies that method will be invoked after each test.

@AfterClass annotation specifies that method will be invoked only once, after finishing all the tests.

Jenkins:

The leading open source automation server, Jenkins provides hundreds of plugins to support building, deploying and automating any project.

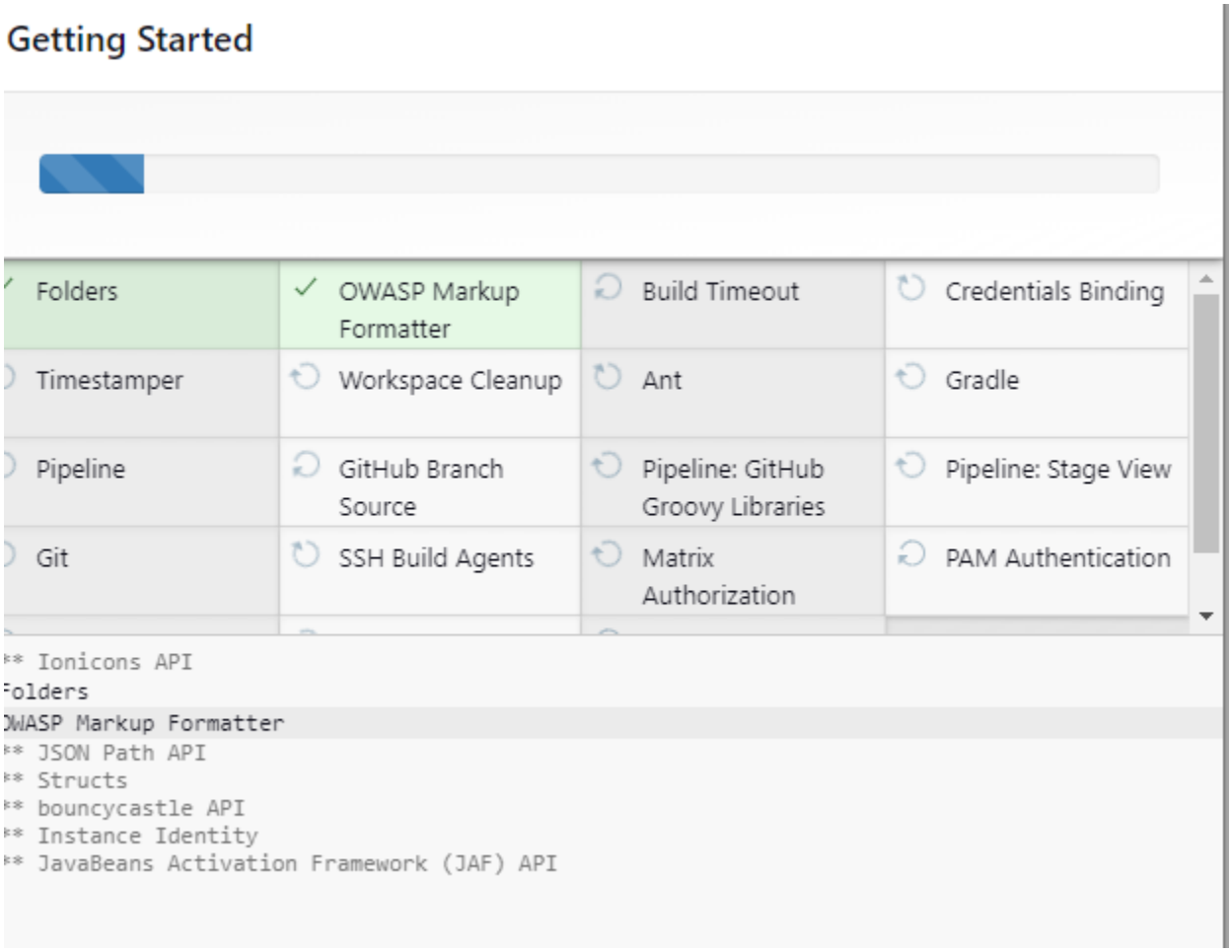


HttpPort -8087

1.java -jar jenkins.war

2.java -jar jenkins.war --httpPort=8087

The HTTP port is the port at which your service is listening for incoming HTTP requests.



Code Quality: Sona

What is Continuous Integration ?

Continuous Integration is a software development practice where developers regularly merge their code changes into a central repository. Each merge triggers an automated build and test process, ensuring that new code integrates smoothly with the existing codebase.

Key Elements of CI:

- ➔ Frequent Code Integration: Developers merge their code changes multiple times a day or whenever a task is completed, rather than waiting for the end of a development cycle.
- ➔ Automated Build and Test: Upon each code commit, an automated process builds the application and runs a suite of tests, including unit tests, integration tests, and sometimes even deployment tests.

1.Frequency Code Integration
2.Automated Build and Test
3.immediate Feedback
4.Shared Code Repository

What is Continuous Integration ?

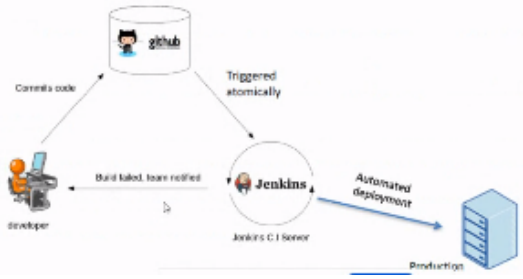
- ➔ Immediate Feedback: CI provides rapid feedback to developers. If any issues or bugs are detected during the automated testing, developers are alerted immediately, allowing them to fix problems quickly.

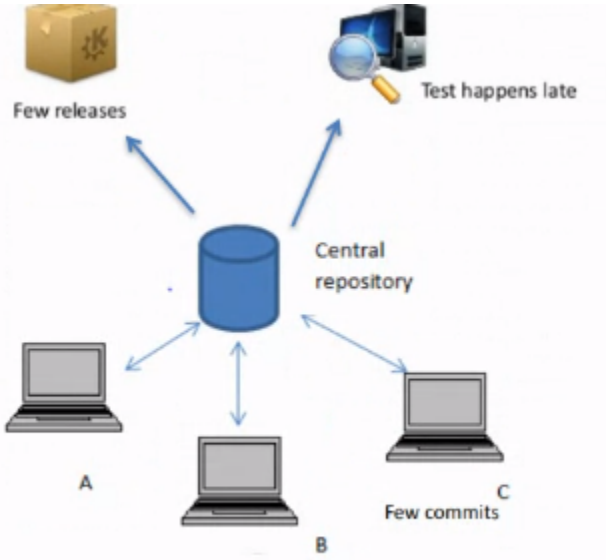
- ➔ Shared Code Repository: All code changes are integrated into a shared version control repository (e.g., Git), ensuring that everyone is working with the most up-to-date code.

What is Jenkins ?

Jenkins is an open-source automation server widely used in software development for Continuous Integration (CI) and Continuous Delivery (CD) processes.

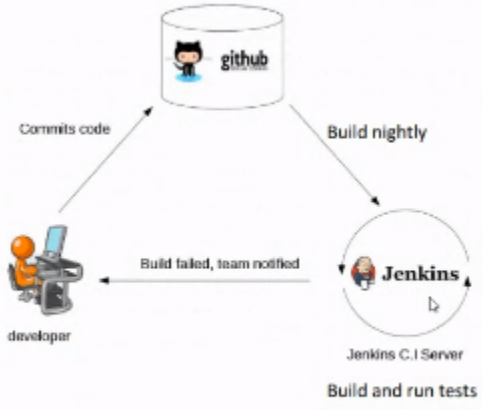
Its primary role is to automate various stages of the software development lifecycle, making development, testing, and deployment more efficient and reliable.





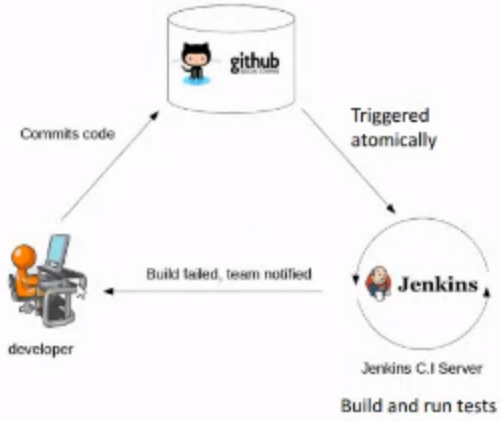
Stage 1:

- No build servers.
- Developers commit on a regular basis.
- Changes are integrated and tested manually.
- Fewer releases.



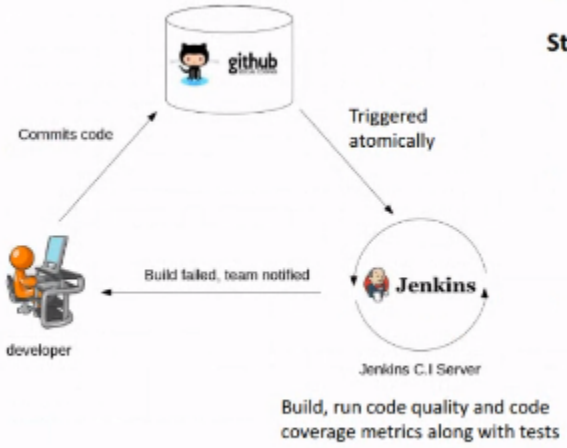
Stage 2:

- Automated builds are scheduled on a regular basis.
- Build script compiles the application and runs a set of automated tests.
- Developers now commit their changes regularly.
- Build servers would alert the team members in case of build failure.



Stage 3:

- A build is triggered whenever new code is committed to the central repository.
- Broken builds are usually treated as a high priority issue and are fixed quickly.

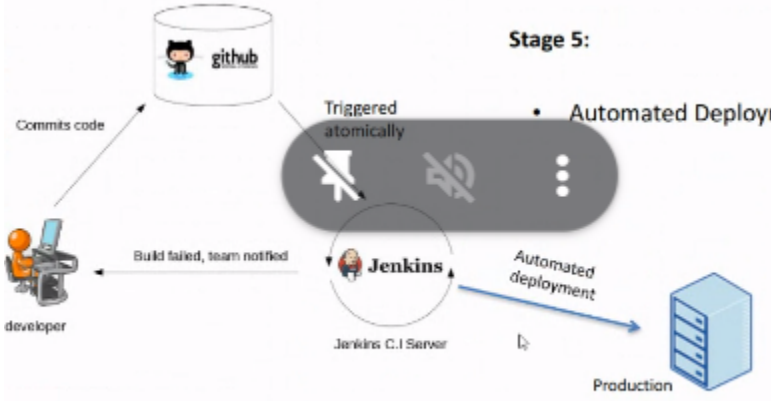


Stage 4:

- Automated code quality and code coverage metrics are now run along with unit tests to continuously evaluate the code quality.

Is the code coverage increasing?

Do we have fewer and fewer build failures?



Stage 5:

- Automated Deployment

Why Jenkins :

4. Extensibility with Plugins:
   - Offers a vast plugin ecosystem, enabling integration with a wide array of tools, technologies, and platforms, making it adaptable to diverse project requirements.
5. Scalability and Distributed Builds:
   - Supports distributed builds across multiple machines, enhancing scalability for larger projects or teams.
6. Visibility and Collaboration:
   - Provides visibility into the development pipeline, fostering collaboration among developers, testers, and other stakeholders. This transparency aids in identifying issues early and facilitates better communication.
7. Enhanced Code Quality:
   - Automated testing in Jenkins helps maintain code quality by executing tests with each code change, reducing the likelihood of bugs reaching production.

Jenkins :

9. Community Support and Active Development:
   - Being an open-source tool, Jenkins benefits from a vibrant community that contributes plugins, updates, and support, ensuring continuous improvement and innovation.

10. Cost-Effective Solution:
   - Jenkins being open-source eliminates licensing costs, making it an attractive choice for organizations looking for a cost-effective automation solution.

11. Adaptability to Various Workflows:
   - Jenkins can accommodate different development workflows and methodologies, making it versatile for teams practicing Agile, DevOps, or other development approaches.

Date:1-3-2024

Code Quality:

1.Code readability
2.Maintainable
3.Debugging
4.Reusability

Course Learning Objectives:

What is Clean Code:

Clean code is the practice of writing straightforward,readable,teachable,and easy -to-understand code.
Even bad code can function.

Why should we care about clean code?

Maintainable Codebase: Any software that we develop has a producttive life and during this period will require chandges and general maintenance.Clean code help develop

Easier Troubleshooting: Software during its lifetime will see many developers create,update, and maintain it,with developers joining at different points in time.This requires a quicker onboarding to keep productivity high,and clean code helps achieve this goal.

Characteristics of clean Code:

1.Focused.
2.Simple
3.Teastable

Clean Coding in java:

Java Code Guidelines:

1.Use Standard project Structure.
2.Stick to java Naming Conversions.
3.Source File Structure
4.White Spaces
5.Indentation
6.Method parameters
7.Hard Coding
8.Code Comments
9.Readability over Reusability.

1.Use Standard project Structure.

2.Sick to java Naming Conventions

Java naming Conventions are a set of rules that dictate how java developers should name identifiers.

Class and interface name should be nouns and have first letter capitalized
Method names should be verbs.
Package names should be lowercase.
Constant names should be short and meaningful.
Variables names should be short and meaningful
Constant names should be capitalized.

3.3.Source File Structure

3. Source File Structure

Let's see how should a typical ordering of elements in a source file look:

- Package statement
- Import statements
  - All static imports
  - All non-static imports
- Exactly one top-level class
  - Class variables
  - Instance variables
  - Constructors
  - Methods

Apart from the above, methods can be grouped based on their functionality or scope.

MainClass.java × Math.class

```
1  package com.valan.pack;
2
3  import java.util.Scanner; //Non-Static Import
4
5  import static java.lang.Math.PI; //Static Import
6  import static java.lang.Integer.parseInt;
7
8  public class MainClass {
9      static int a = 10;
10
11     public static void main(String[] args) {
12         System.out.println(PI);
13
14         int a = parseInt("123");
15
16         System.out.println(a);
17     }
18 }
```

```java
package com.valan.pack;

import java.util.Scanner; //Non-Static Import

import static java.lang.Math.PI; //Static Import
import static java.lang.Integer.parseInt;
import static java.lang.System.out;

public class MainClass {
    static int a = 10;

    public static void main(String[] args) {
        out.println(PI);

        int a = parseInt("123");

        System.out.println(a);
    }
}
```

SOLID is an acronym for five key design principles:

- Single Responsibility Principle.
- Open-closed principle.
- Liskov substitution principle.
- Interface segregation principle.
- Dependency inversion principle.

White Spaces:

4. White Spaces

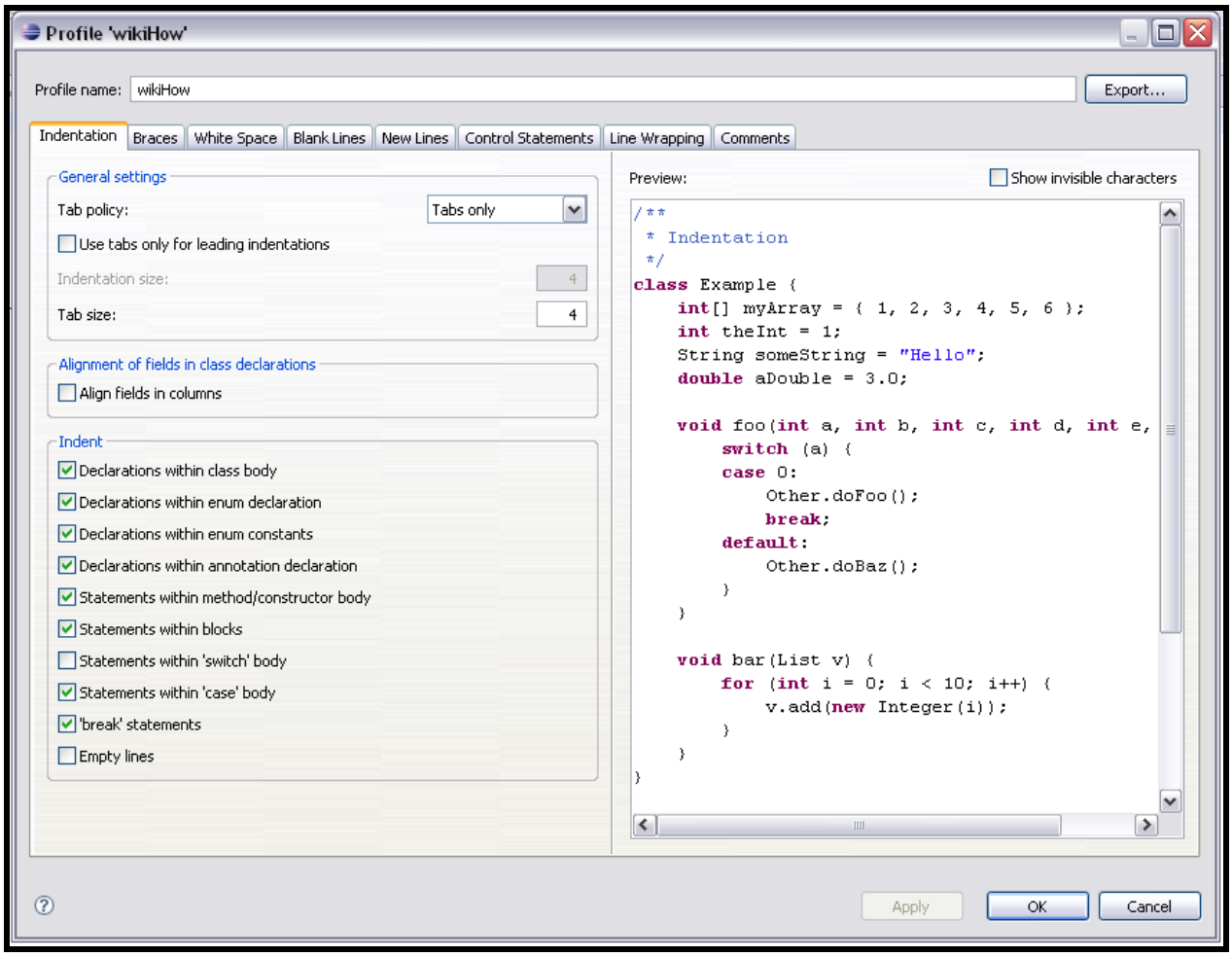
We all know that it is easier to read and understand short paragraphs compared to a large block of text. It is not very different when it comes to reading code as well. Well-placed and consistent white spaces and blank lines can enhance the readability of the code.

- Two blank lines before starting static blocks, fields, constructors and inner classes
- One blank line after a method signature that is multiline
- A single space separating reserved keywords like if, for, catch from an open parentheses
- A single space separating reserved keywords like else, catch from a closing parentheses

Ctrl + shift + f = for alignment(Eclipse IDE)

5.Indentation.

1.A well-indented code is much easier to read and undersatnd.




```
public class MainClass {
    static int a = 10;
    int b = 20;
    String message = "welcome to code quality. It is always better to write clean code. "
            + "So it is easy to test and maintain the code";

    public static void main(String[] args) {
        out.println(PI);

        int a = parseInt("123");

        System.out.println(a + b);

        if (true) {

        }
    }
}
```

6.Method parameters:

6. Method Parameters

Parameters are essential for methods to work as per specification. But, a long list of parameters can make it difficult for someone to read and understand the code. So, where should we draw the line? Let's understand the best practices which may help us:

- Try to restrict the number of parameters a method accepts, three parameters can be one good choice
- Consider refactoring the method if it needs more than recommended parameters, typically a long parameter list also indicate that the method may be doing multiple things
- We may consider bundling parameters into custom-types but must be careful not to dump unrelated parameters into a single type

6. Method Parameters

Let's see an example of this:

```
public boolean setCustomerAddress(String firstName, String lastName, String streetAddress,
  String city, String zipCode, String state, String country, String phoneNumber) {
}

// This can be refactored as below to increase readability

public boolean setCustomerAddress(Address address) {
}
```

7.

7. Hard Coding

Hardcoding values in code can often lead to multiple side effects. For instance, it can lead to duplication, which makes change more difficult. It can often lead to undesirable behavior if the values need to be dynamic.

In most of the cases, hardcoded values can be refactored in one of the following ways:

- Consider replacing with constants or enums defined within Java
- Or else, replace with constants defined at the class level or in a separate class file
- If possible, replace with values which can be picked from configuration or environment

```java
package com.valan.pack;

enum Days {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
}

public class MainClass {

    public static void main(String[] args) {
        switch(Days.TUESDAY) {
        case SUNDAY:
            System.out.println("Today is Sunday");
            break;
        case MONDAY:
            System.out.println("Today is Monday");
            break;
        case TUESDAY:
            System.out.println("Today is Tuesday");
            break;
        default:
            System.out.println("Invalid Day");
            break;

        }
```

```
<terminated> MainClass (3) [Java Applicatio
Today is Tuesday
```

8.Code Command:

Three types of command:

**Single -line Command **

Multi line command /* */

Documation command / *@version ***/**

8. Code Comments

Code comments can be beneficial while reading code to understand the non-trivial aspects. At the same time, care should be taken to not include obvious things in the comments.

This can bloat comments making it difficult to read the relevant parts.

Java allows two types of comments: Implementation comments and documentation comments. They have different purposes and different formats, as well.

```java
package com.valan.pack;

enum Days {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
}

/*@
 * version : jdk 1.8
 * author : valan
 */
public class MainClass {
    /*
     * Multi-Line Comments
     */
    public static void main(String[] args) {

        //Single-Line Comment
        switch(Days.TUESDAY) {
        case SUNDAY:
            System.out.println("Today is Sunday");
            break;
        case MONDAY:
            System.out.println("Today is Monday");
            break;
```

9.Readability over Reusability:

9. Readability over Reusability

Reusability is one of the most advocated concepts in software development. It lessens development time and reduces the effort required to maintain software when developers understand the reusable components very well.

While the concept of reusability sounds great and has many benefits, it also has many potential pitfalls, especially when working on an unfamiliar codebase.

Tools for Help

There are several tools available in the Java ecosystem, which take at least some of these responsibilities away from code reviewers. Let's see what some of these tools are:

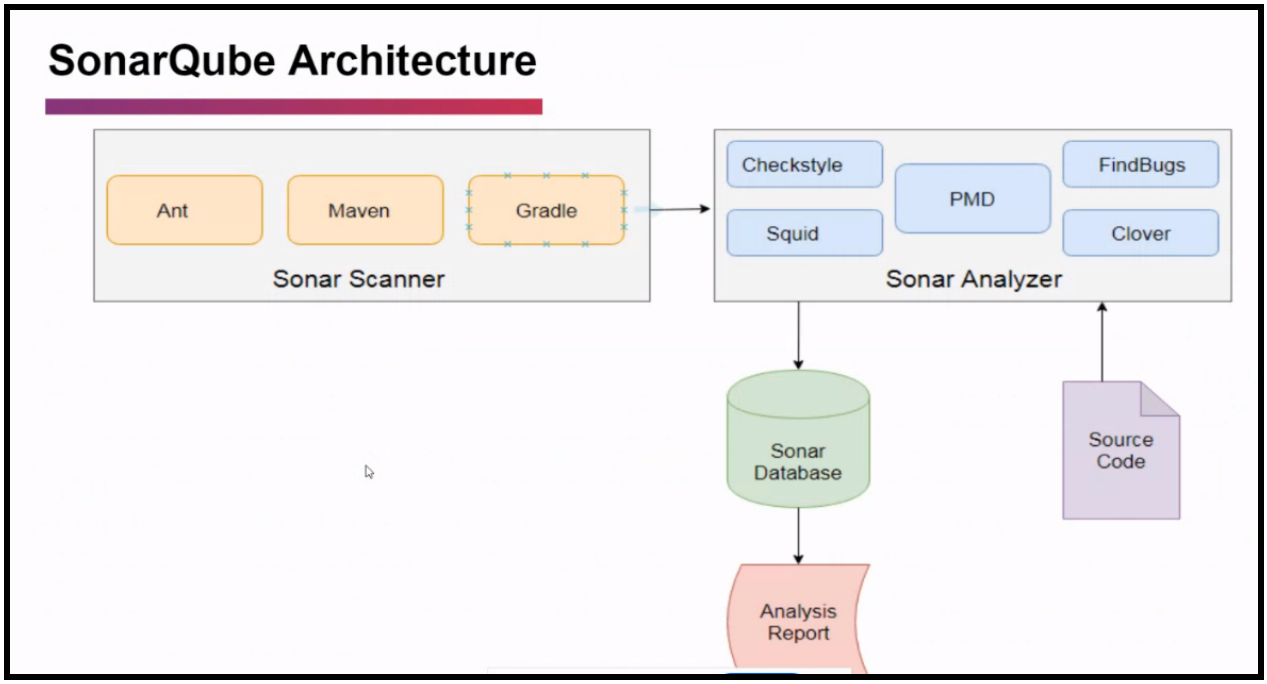
- **Code Formatters:** Most of the popular Java code editors, including Eclipse and IntelliJ, allows for automatic code formatting. We can use the default formatting rules, customize them, or replace them with custom formatting rules. This takes care of a lot of structural code conventions.
- **Static Analysis Tools:** There are several static code analysis tools for Java, including SonarQube, SonarLint, Checkstyle, PMD and SpotBugs. They have a rich set of rules which can be used as-is or customized for a specific project. They are great in detecting a lot of code smells like violations of naming conventions and resource leakage.

What is SonarQube?

SonarQube is a Code Quality Assurance tool that collects and analyzes source code, and provides reports for the code quality of your project.

It combines static and dynamic analysis tools and enables quality to be measured continually over time.

It is used to find tricky issues and It can work with 25 languages like Java, .NET, COBOL, PHP, Python, C++, JS, Ruby, Kotlin, and Scala.



4-1-2024
SOLID:
1.Single Responsibility Principle
2. Open-Closed Principle

The application or module entities the methods, functions , variable,
Its module should be open for extension but extension but closed for modification.

3.Liskov Substitution Principle.
4.Interface Segregation Principle.

Criteria	Unit Testing	Functional Testing
Purpose	Verify individual units of code	Verify software application meets requirements
Scope	Individual units of code	Entire software application
Errors	Code branches, Edge cases	Software / Application
Level of Detail	Low-level, code-based testing	High-level, user-based testing
Complexity	Easy to write and conduct	Complext than unit testing
Test Cases	Automated and isolated from other code	Can be automated or manual, covers end-to-end functionality
Testing Objectives	Test code correctness and quality	Test software functionality and user experience
Testers	Developers	QA Engineers and Testers
Testing Technique	White Box Testing	Black Box Testing
Execution Frequency	Run frequently during development	Run at the end of development cycle before release
Speed	Faster than functional testing	Slow and more complex
Benefits	Early bug detection, improved code quality, easier maintenance	Ensuring software meets requirements, identifying defects early, improving software quality
Examples	Testing individual methods or functions	User interface testing, performance testing, data validation testing, security testing

RONWELL

5.Depen

05-1-2024

Software Testing:

What is Software Testing

1. **Unit testing is fast and helps writing clean code, but doesn't provide confidence that the system will work as expected. Basically, it tells us where the problem is in the code. Functional testing is slow and complex but it ensures that the system will work according to the requirements.**

What is Software Testing

- Software Testing is a method to check whether the actual software product matches expected requirements and to ensure that software product is Defect free.
- It involves execution of software/system components using manual or automated tools to evaluate one or more properties of interest.
- The purpose of software testing is to identify errors, gaps or missing requirements in contrast to actual requirements.

What is the need of Testing?

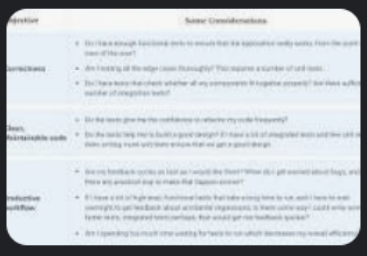
Testing is important because software bugs could be expensive or even dangerous. Some software bugs can potentially cause monetary and human loss also

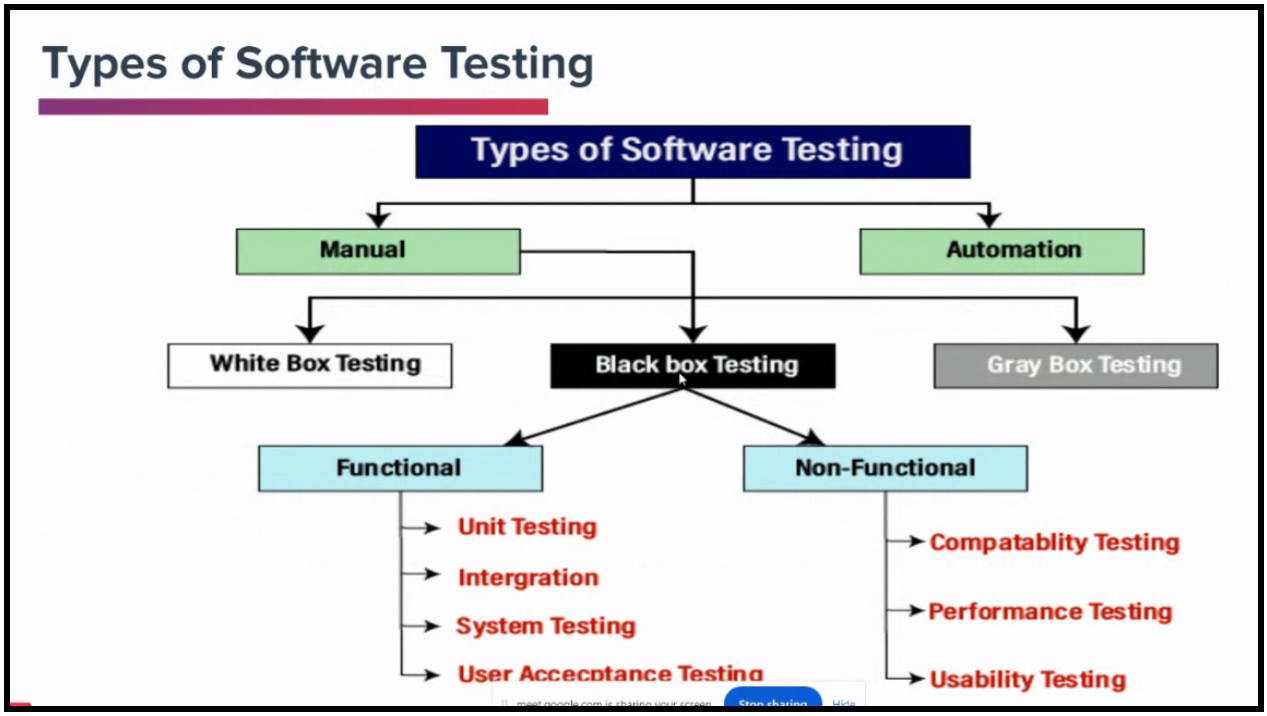
Unit Testing is a part of Functional Testing. Functional testing is slow and tests the overall functionality of the application. Unit test is fast and specific to individual components. Testing the overall working of a Bluetooth speaker is an example of functional testing. 1 Nov 2022

What is the difference between unit testing and functional testing?

Unit testing is fast and helps writing clean code, but doesn't provide confidence that the system will work as expected. Basically, it tells us where is the problem in the code. Functional testing is slow and complex but it ensures that the system will work according to the requirements.

18 Sept 2023





White testing:

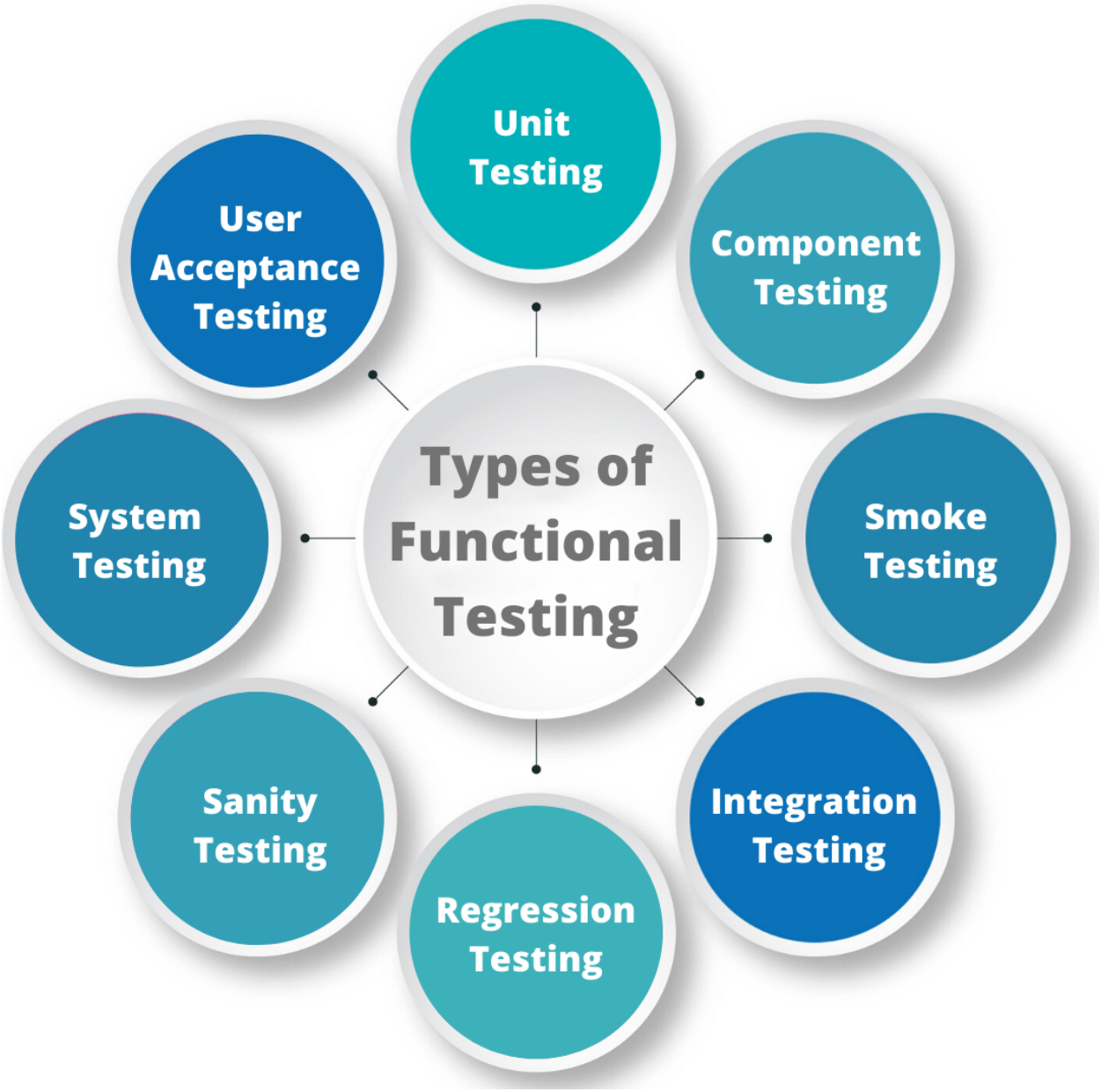
White Box Testing types include Unit Testing, Static and Dynamic Analysis, Statement, Branch, Path Coverage, Security Testing, Loop and Conditional Testing, Mutation and Integration Testing, Penetration Testing, and Memory Perspective Testing.

Gray Box Testing

White box testing

- Tester has complete knowledge of the internal workings of the system being tested.
- It's also known as structural, clear-box, or transparent testing.
- Tester has access to the source code, design documents, and system architecture.
- Tester has an extremely high level of technical and coding expertise.
- Ensures that all code paths are tested and helps identify potential performance issues.

Criteria	Functional Testing	Non-Functional Testing
Purpose	Evaluates the functionality of the software application and ensures it meets specified requirements.	Evaluates the non-functional aspects of the software application, such as performance, usability, security, and reliability.
Testing Techniques	Black box testing, White box testing, Unit testing, Integration testing, System testing, Acceptance testing.	Load testing, Stress testing, Usability testing, Security testing.
Automation	Often performed manually, but can be automated using frameworks like Selenium or Appium.	Typically automated using tools like Apache JMeter, LoadRunner, or SoapUI.
Metrics	Number of features tested, Pass/Fail rate.	Transaction success rate, Response time, Resource utilization rate.


Types of Functional Testing
Unit Testing
Component Testing
Smoke Testing
Integration Testing
Regression Testing
Sanity Testing
System Testing
User Acceptance Testing

What is Manual Testing?

- Manual testing is a software testing process in which test cases are executed manually without using any automated tool.
- All test cases executed by the tester manually according to the end user's perspective.
- Manual testing is mandatory for every newly developed software before automated testing
- Manual Software Testing requires more effort but is necessary to check automation feasibility. Manual Testing concepts does not require knowledge of any testing tool.

Sample Test Case Template

Test Scenario ID	Login-1	**Test Case ID**	Login-1A
Test Case Description	Login – Positive test case	**Test Priority**	High
Pre-Requisite	A valid user account	**Post-Requisite**	NA

Test Execution Steps:

S.No	Action	Inputs	Expected Output	Actual Output	Test Browser	Test Result	Test Comments
1	Launch application	https://www.facebook.com/	Facebook home	Facebook home	IE-11	Pass	[Priya 10/17/2017 11:44 AM]: Launch successful
2	Enter correct Email & Password and hit login button	Email id : test@xyz.com Password: ******	Login success	Login success	IE-11	Pass	[Priya 10/17/2017 11:45 AM]: Login successful

Purpose of Manual Testing ?

- The key concept of manual testing is to ensure that the application is error free and it is working in conformance to the specified functional requirements.
- Test Suites or cases, are designed during the testing phase and should have 100% test coverage.
- It also makes sure that reported defects are fixed by developers and re-testing has been performed by testers on the fixed defects.
- Basically, this testing checks the quality of the system and delivers bug-free product to the customer.

How to perform Manual Testing

Follow these steps to do manual testing:

Step1: First, review all the documents related to the software, for selecting the testing areas.

Step2: Analyse all the required documents to get all the requirements mentioned by the end-user.

Step3: Build test cases as per the requirement document.

Step4: Execute all the test cases manually by using white-box testing and black-box testing.

Step5: If bugs are detected, report them to the development team.

Step6: When the developer fixes the bug, retest it.

Software Testing Life Cycle

STLC(Software Testing Life Cycle) is a process that identifies which test activities to perform and when to execute those test activities.
Following are the different phases of STLC:

Why do we develop test cases ?

We build the test cases due to the following reasons:

- For maintaining consistency in the test case execution.
- To ensure better test coverage.
- For avoiding training to every new software tester on the product.
- It relies on the process instead of a person.