Caltech | Center for Technology & Management Education

Post Graduate Program in DevOps

simplilearn

Caltech | Center for Technology & Management Education

CI/CD Pipeline with Jenkins

DevOps Practices and Methodologies

Learning Objectives

By the end of this lesson, you will be able to:

- Classify Continuous Integration, Continuous Delivery, and Continuous Deployment
- Enumerate the similarities and differences between Continuous Integration, Continuous Delivery, and Continuous Deployment
- Detail the functions of source code management and branching
- Discuss the importance of code scanning tools
- Outline the architecture of Jenkins Distributed Builds

Continuous Integration

Introduction to Continuous Integration

Integration is the process of incorporating changes such as refactored code, new features, or bug fixes into the existing codebase.

Problem

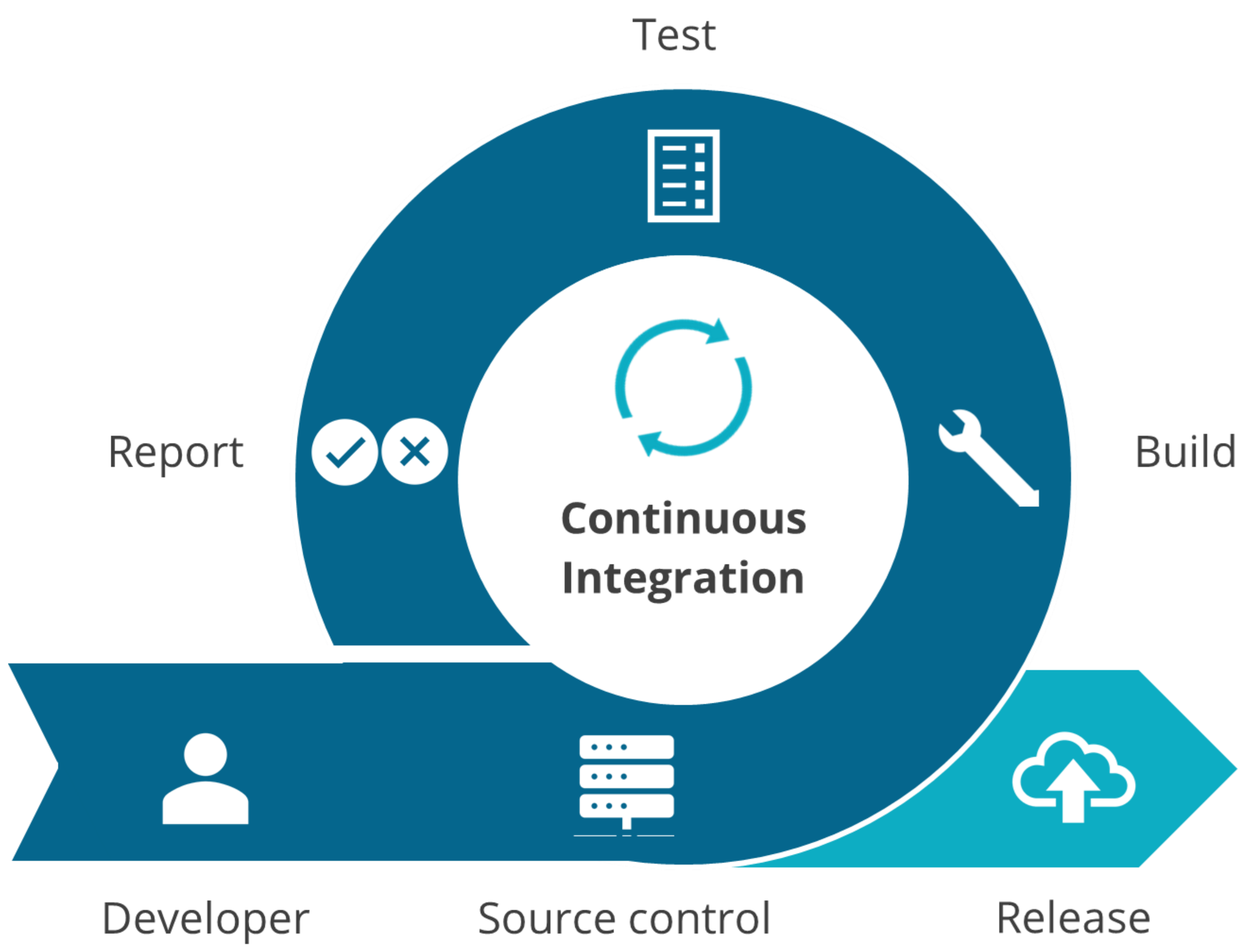
- Integrating multiple or large changes in one go may result in last-minute conflicts and bugs.

Solution

- Continuous Integration is one among the many strategies that have emerged to mitigate these problems.

Continuous Integration

Visual representation of CI is as shown below:



Continuous Integration

The CI process includes creating workflows, ensuring daily commits, and running tests.

Workflow ensures every commit made to the codebase is tested and built.

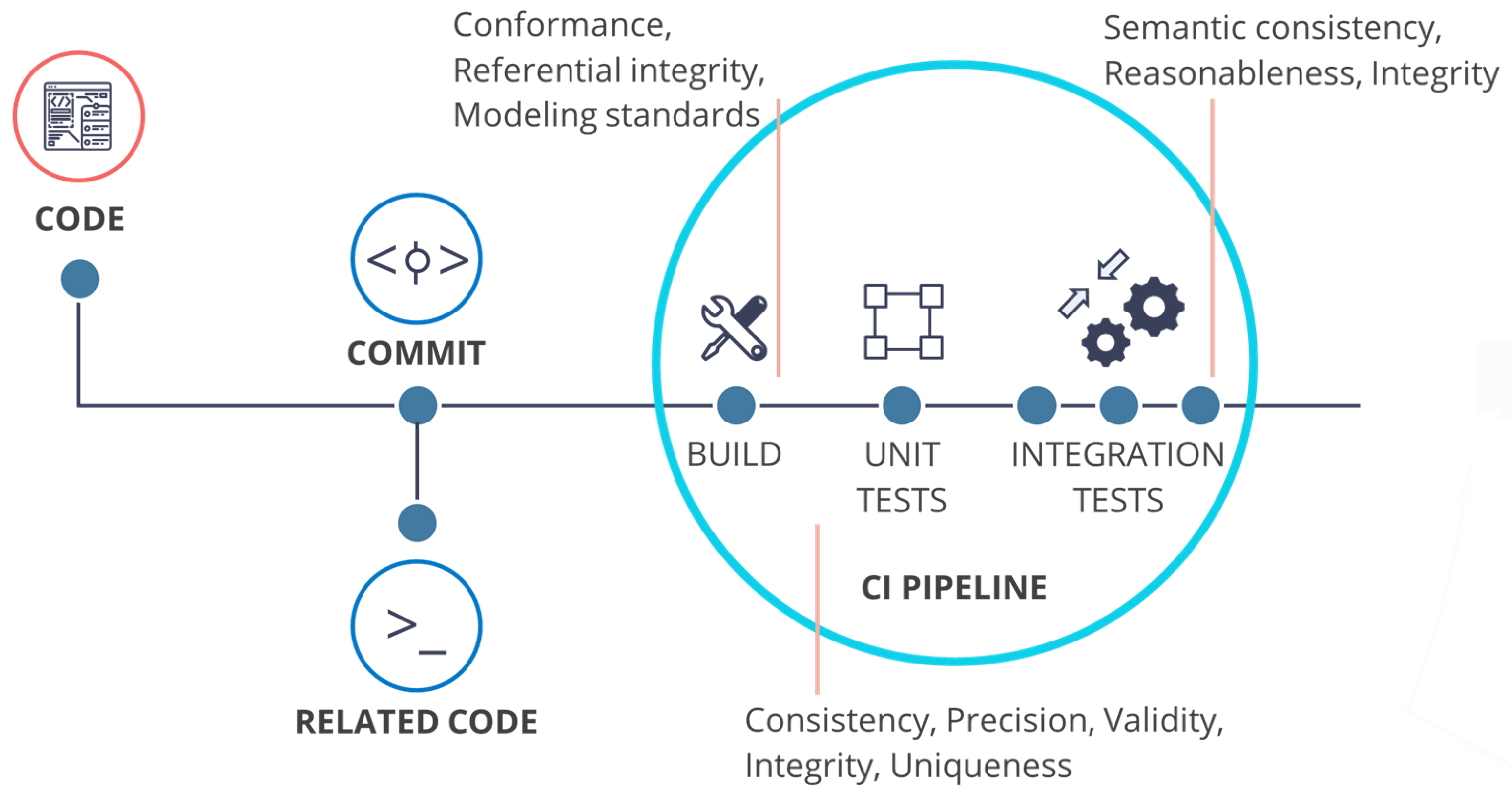
The daily commit expectation ensures that each developer on the project commits code at least once per day.

A successful build triggers a run of the test suite.

The release process is smoother and less painful with CI.

Continuous Integration

Detailed representation of Continuous Integration with Build & Tests Automation:



Continuous Integration Principles

To build automation, continuous integration relies on four principles.

Maintenance of Code Repository

A code repository acts as a revision control system for the project's source code. All artifacts required to build the project should be placed in the repository.

Continuous Integration Principles

Automation of Build

A single command should have the capability to build the system. Many build tools, such as make, have existed for many years. Other more recent tools are frequently used in continuous integration environments.

Continuous Integration Principles

Creation of build self-testing builds

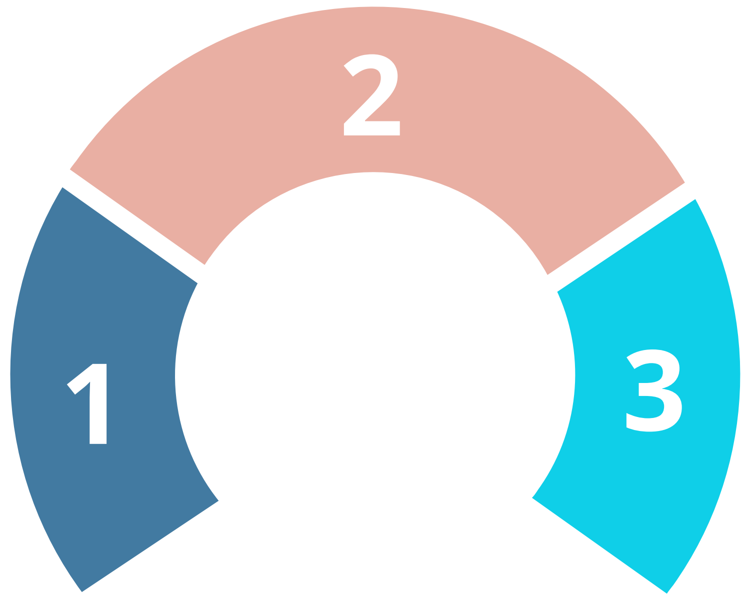
Once code is built, all tests should run to confirm that it behaves as the developers expect.

Continuous Integration Principles

Daily commits to baseline

Daily commitment reduces the number of conflicting changes.
Checking in a week's worth of work runs the risk of conflicts that may be difficult to resolve.

Benefits of Continuous Integration



Prevention of chaos at release dates

Early detection of integration bugs

Small loss in changes during codebase reversion

Build Automation Principles

CI principles are used in the implementation of build automation to help:

Build every commit

Build commits to the current working version to verify proper integration. Use **Automated CI.** (May be done manually too)

Build fast

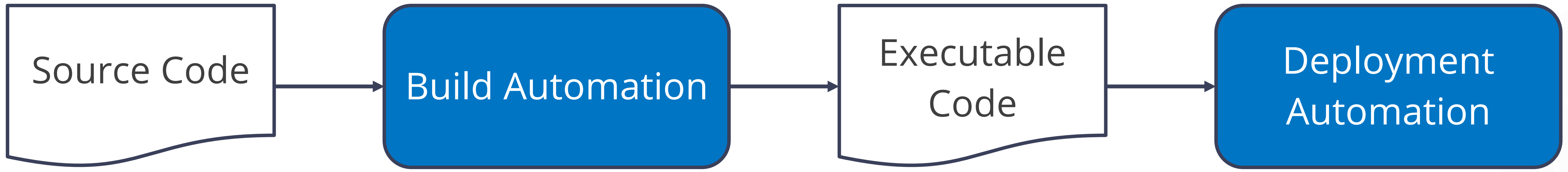
The build needs to complete rapidly. This helps to quickly identify integration issues.

Build transparently

Find out if the build breaks easily. If so, identify who made the change and the nature of the change.

Build Automation

Build automation is the process of compiling the source code into executables.



It is the very first thing implemented during the CI-CD pipeline setup.

Build Automation

The benefits of build automation are:

Fast Feedback
Bugs are fixed at the early stages.

Fast Overall Release Cycle
Manual Builds have slower release cycles.

Standardized Builds
Human errors and bugs are eliminated.

Build Tools

These tools can be used to perform build automation.

<APACHE ANT>

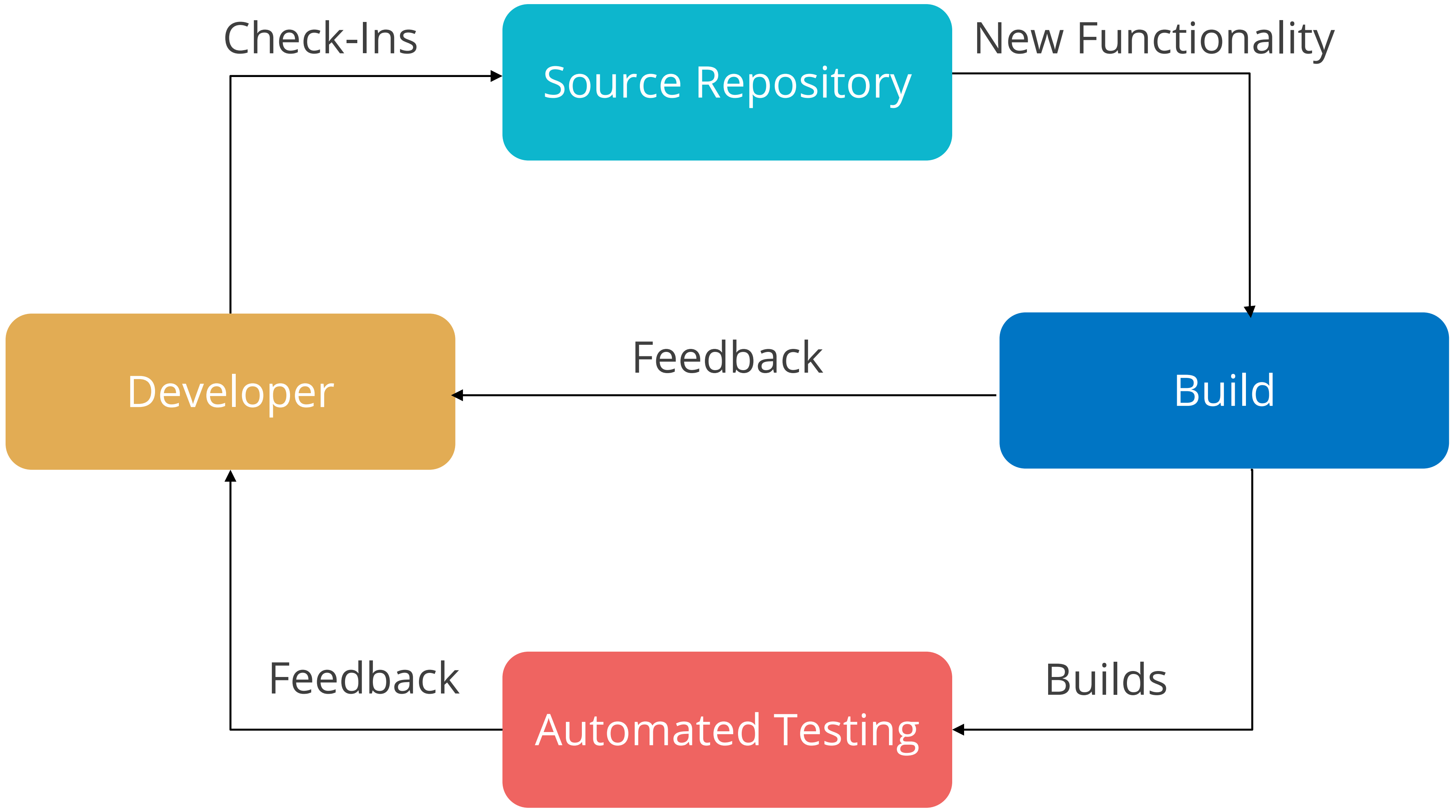
Gradle

MSBuild

maven

They can be easily integrated with IDE also.

Test Executions



Test Executions

Can the software application fulfill all the business requirements?
Software testing helps us determine the answer.

- QA personnel have done manual testing traditionally.
- Automated tests allow for multiple daily tests without human dependencies on QA professionals.
- Manual testing post-deployment is time-consuming and expensive; automated testing can be done easily and economically post-deployment.

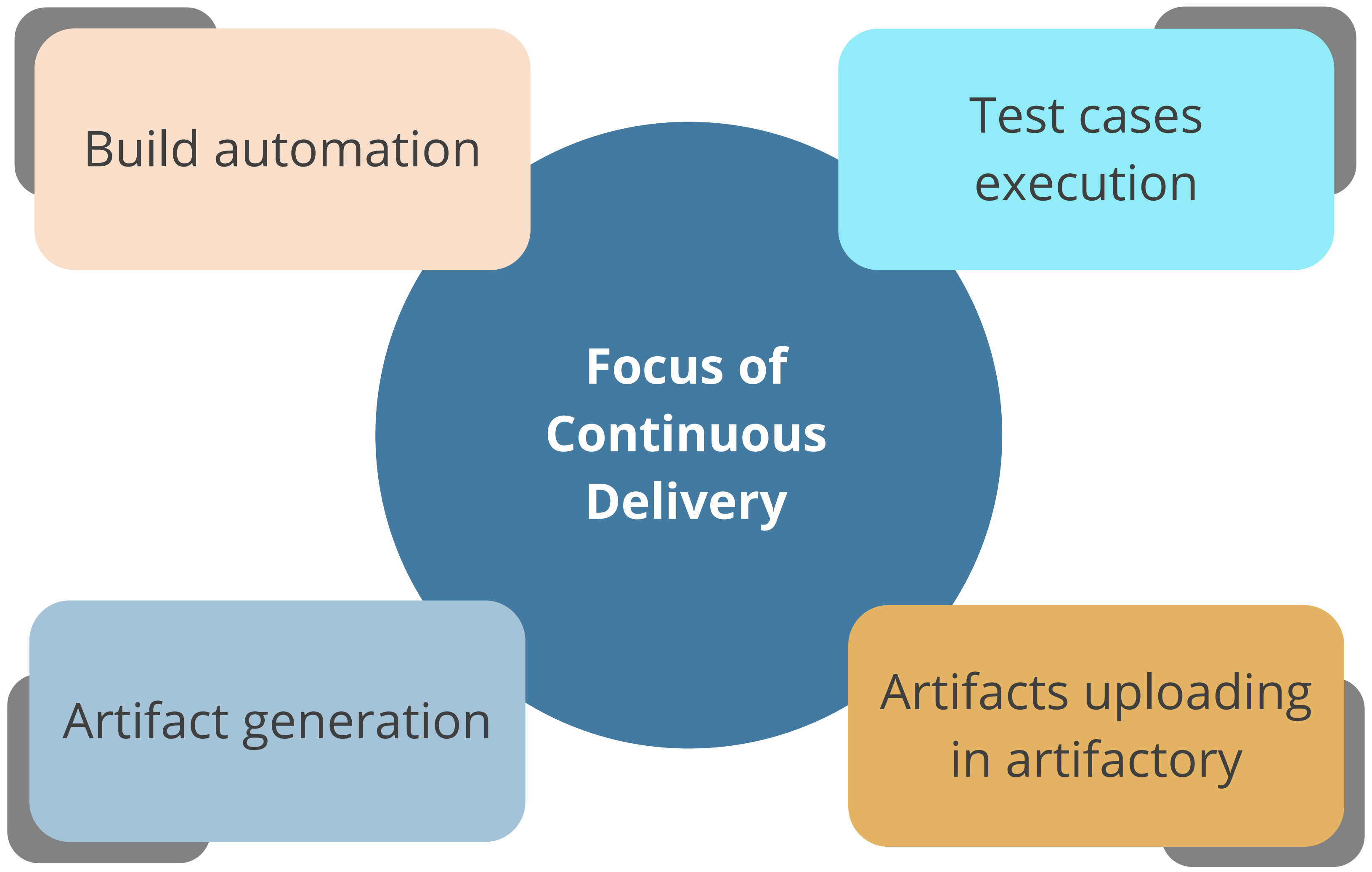
Benefits of Test Executions

- Increased Efficiency
- More Test coverage
- Low Operation cost
- No human errors
- Reusable Test Scripts

Continuous Delivery

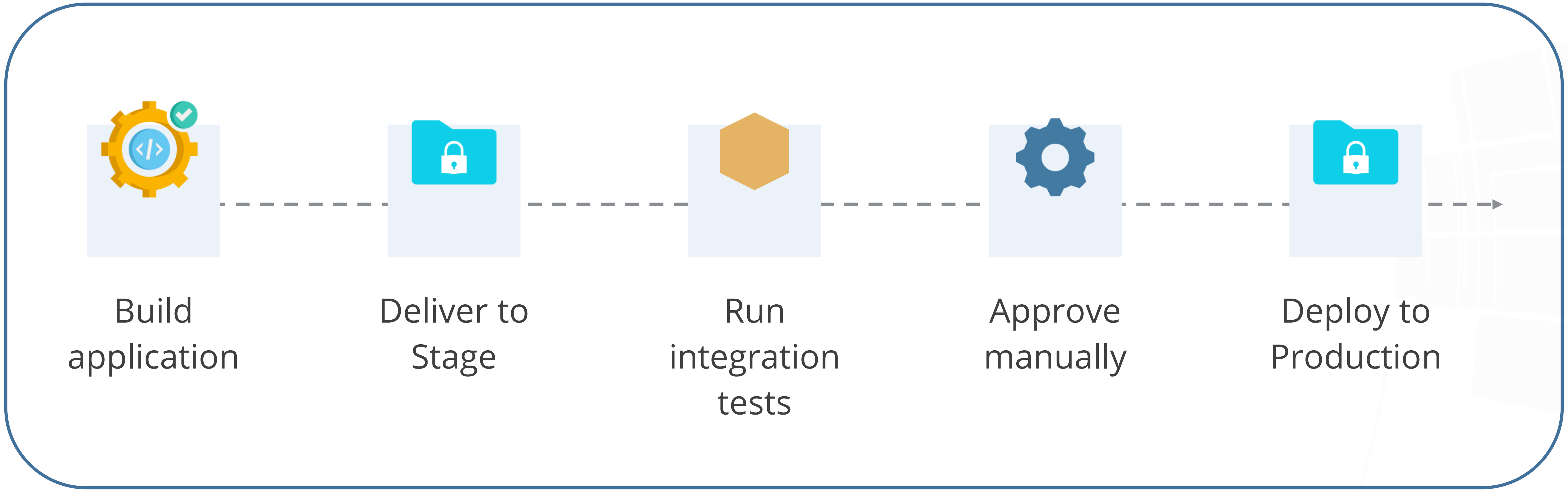
Continuous Delivery

Continuous Delivery (CD) is the process of delivering software in shorter release cycles.


Build automation
Test cases execution
Focus of Continuous Delivery
Artifact generation
Artifacts uploading in artifactory

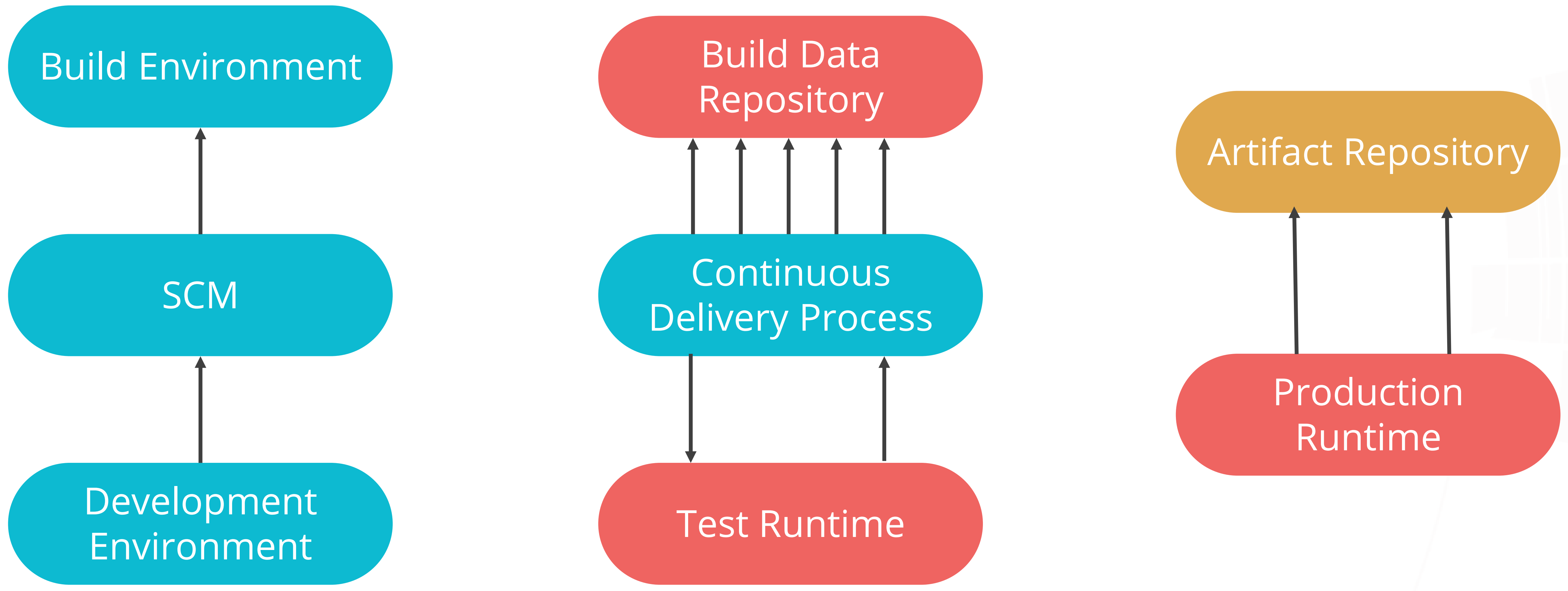
Continuous Delivery

Continuous delivery provides a fully automated pipeline that continuously delivers the product/service to the business and make releases in shorter cycles.



Build Artifacts

Build artifacts are artifacts produced by build automation, such as WAR, EAR, DLL and EXE. They act as executables that are used to access the application.

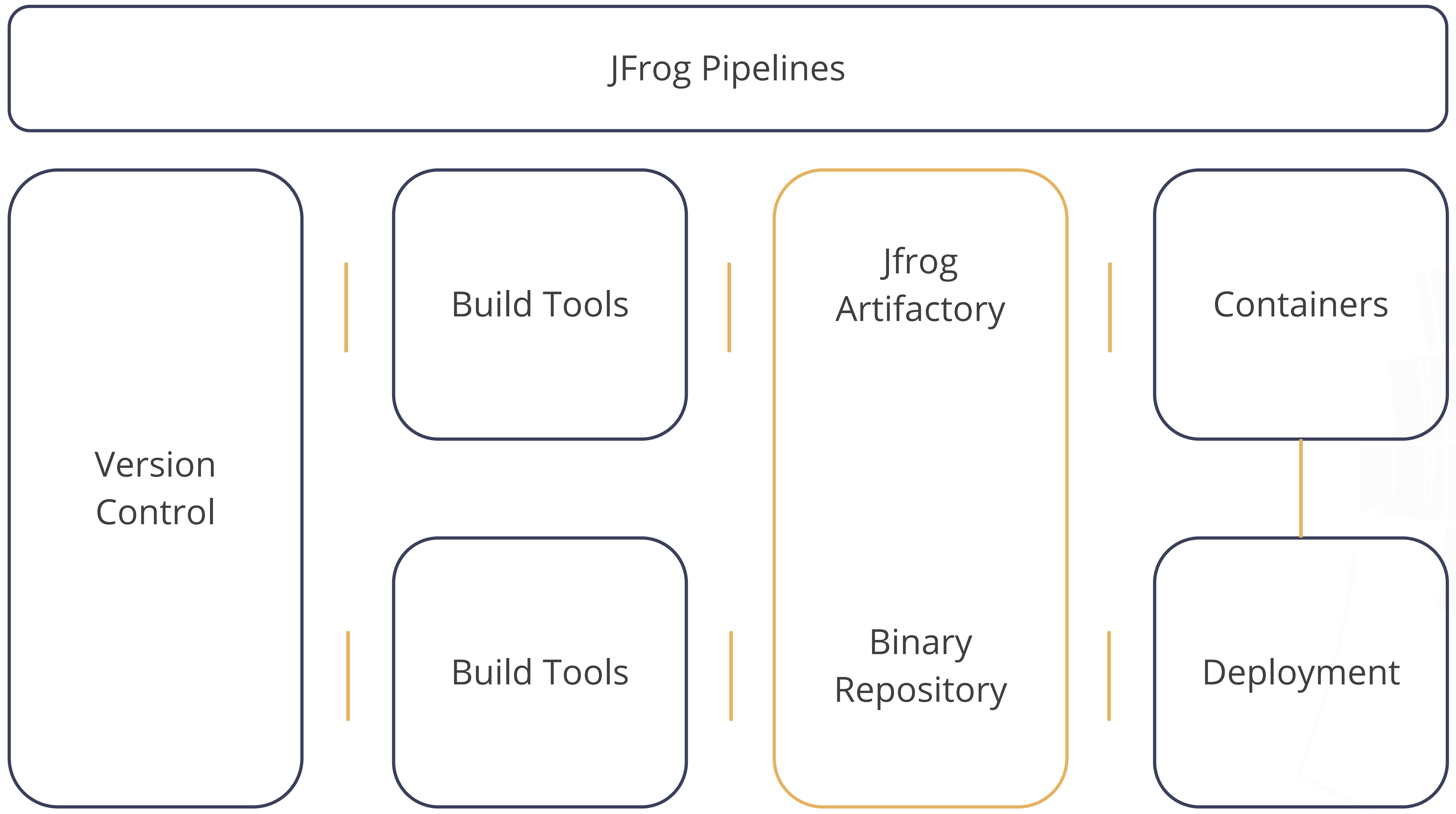


JFrog Artifactory

A universal DevOps solution that provides end-to-end management of artifacts and binaries.

- Automates **artifacts delivery**
- Can be used by developers to store maven artifacts, npm node modules, Docker images, Helm Charts and generic repositories
- Comes with **complete CLI and REST APIs** interfaces that can be easily integrated with other DevOps tools
- Available in 2 variants: a self-hosted solution and a SaaS-based cloud solution, where binaries and executables can be hosted.

JFrog Artifactory



Continuous Deployment

Continuous Deployment

Automatic release of source code into live environment

It helps developers to achieve an automated deployment process.

Faster development process with no pauses for releases

Deployments pipelines are triggered automatically for every change.

Continuous Deployment

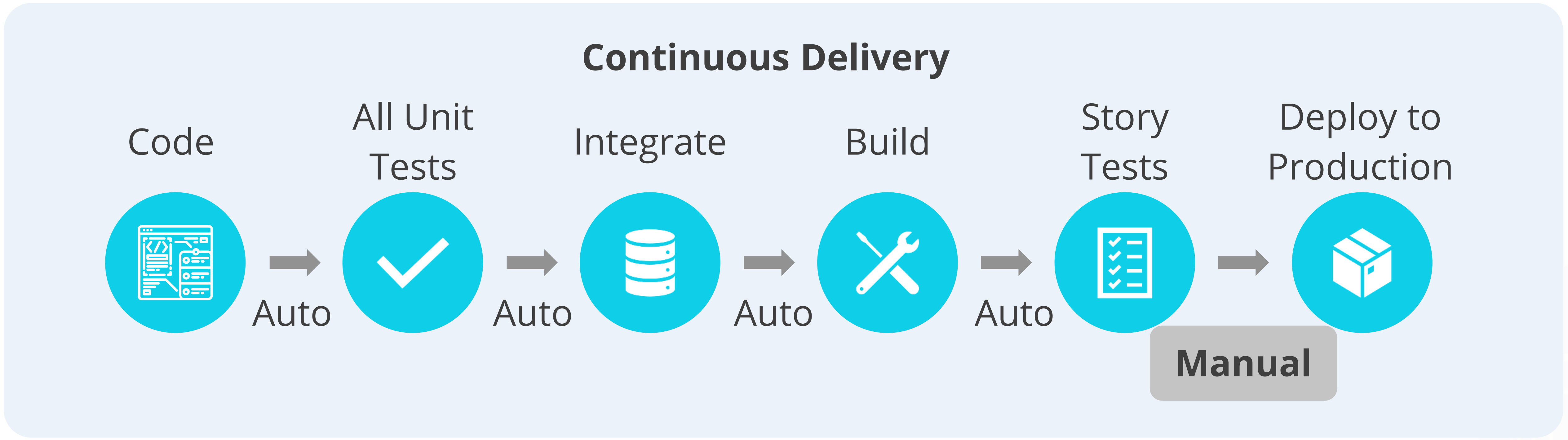
Trigger of automated build from version control system

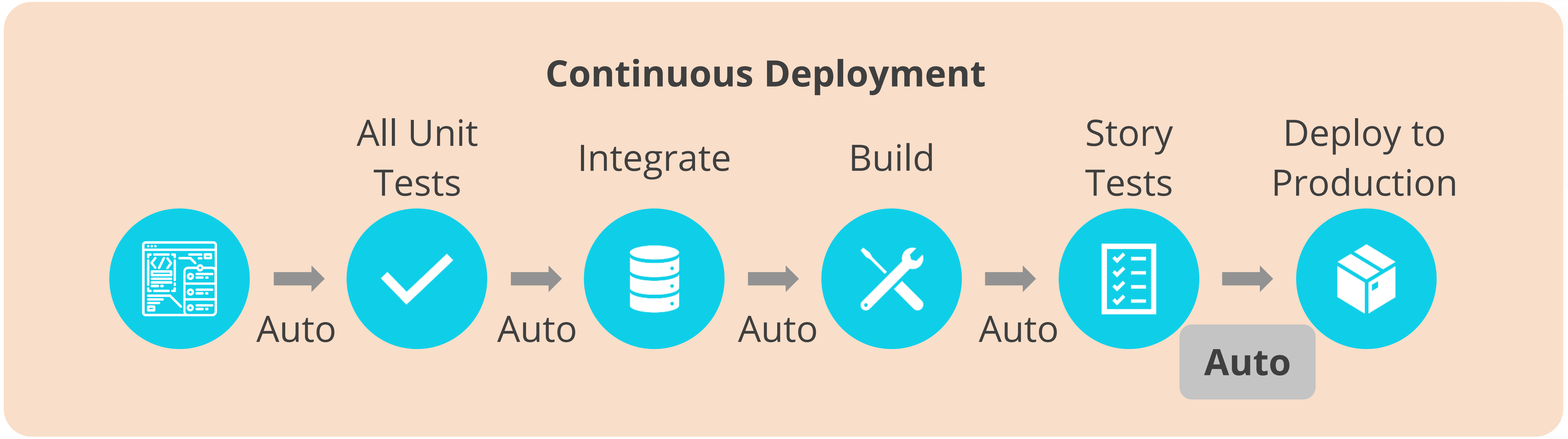
Compilation of the source code will be done in the automated build.

Generation of final binaries

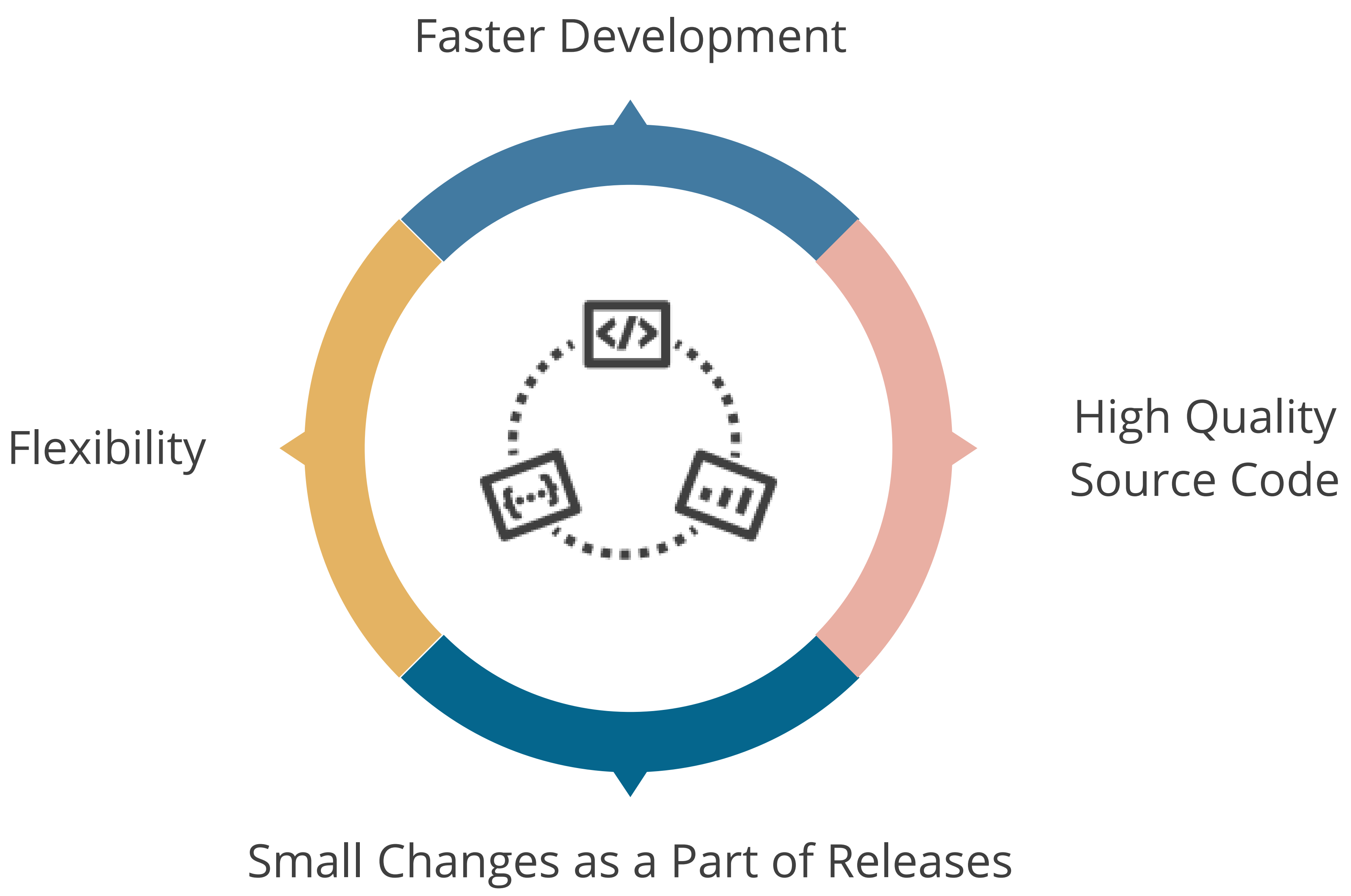
The source code can be deployed on the desired infrastructure.

Continuous Delivery vs Continuous Deployment

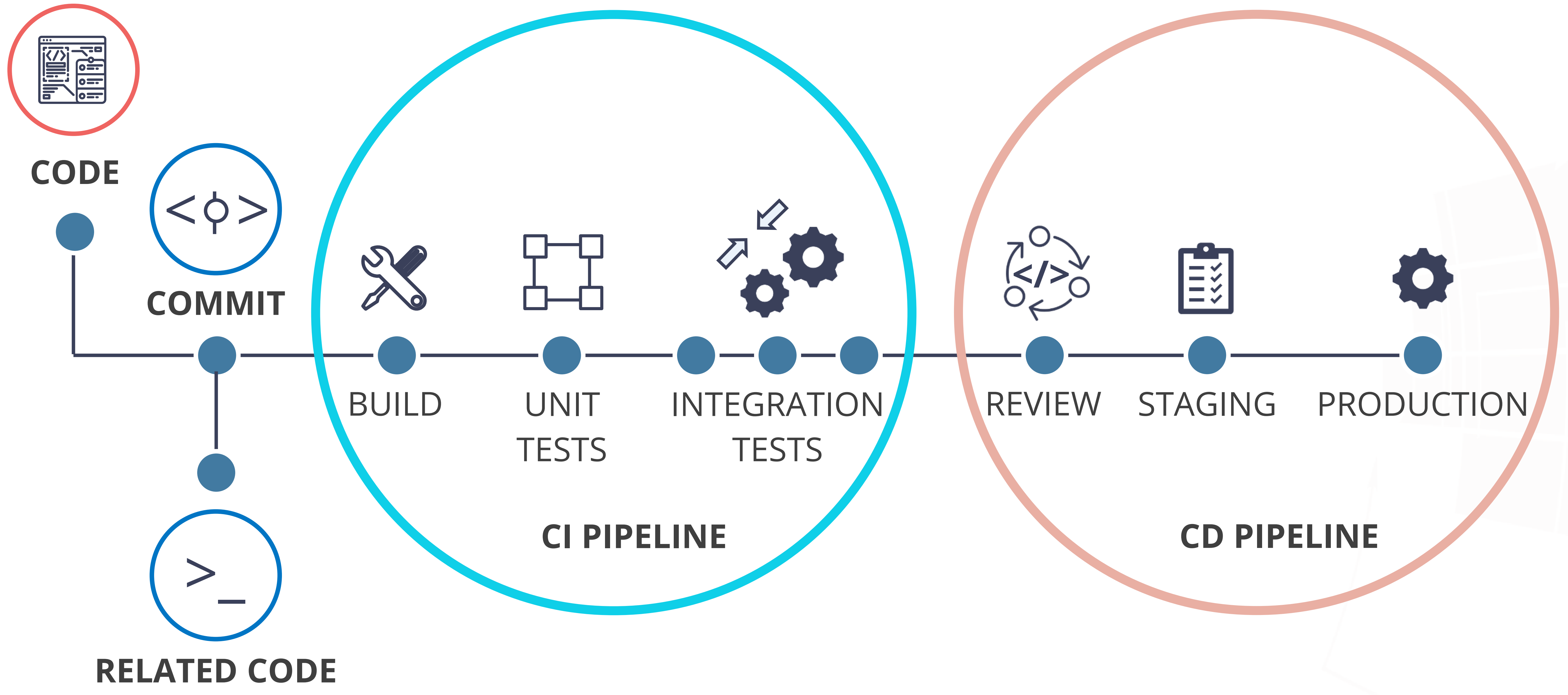




Benefits of Continuous Deployment



Fully Automated CI/CD Pipeline



Fully Automated CI/CD Pipeline

1 A git repository is created, or an existing one is used.

2 The Dev team commits the code to the Dev-Branch.

3 Jenkins fetches the code from Github and maps it with the job enabled for the particular task.

4 It is ensured that CI and CD are done for the task.

5 Jenkins pulls the code and enters it to the commit phase of the task.

Fully Automated CI/CD Pipeline

6 The code is compiled. This is known as the build phase of the task.

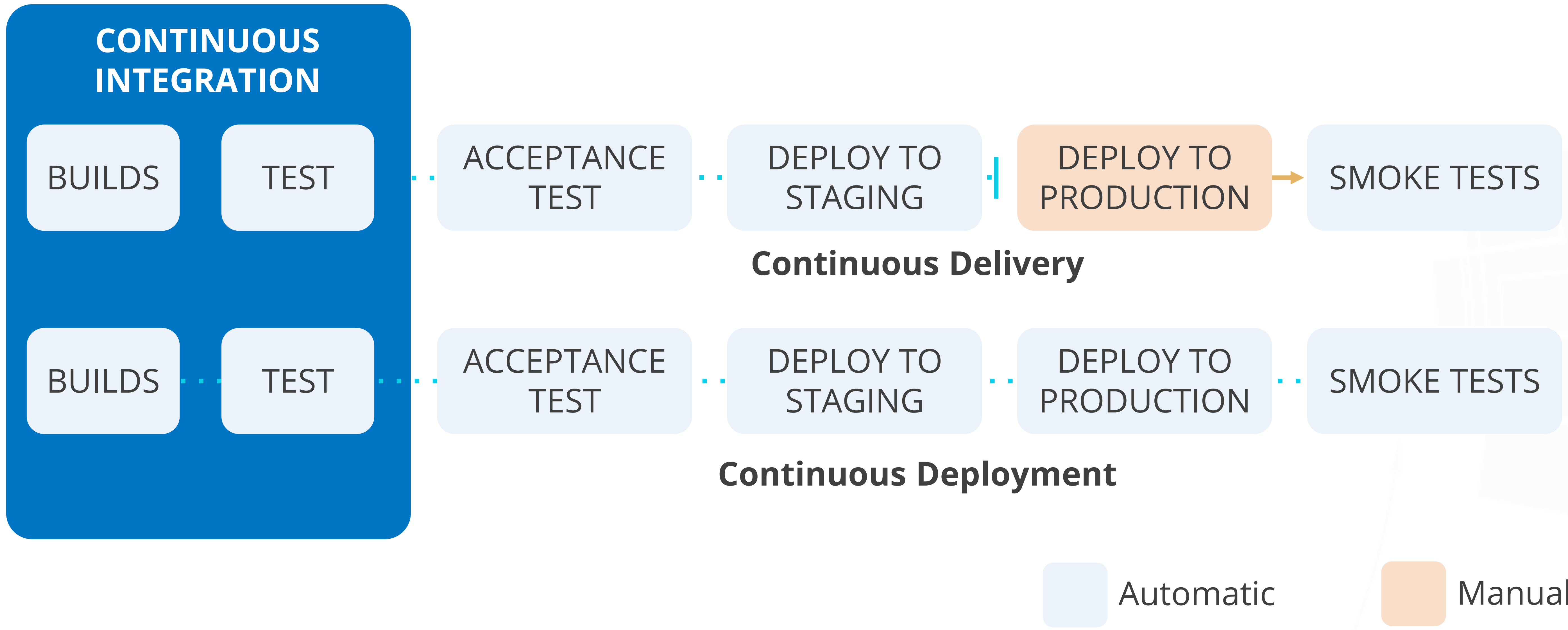
7 After the DevOps team merges the code to the Master branch, Jenkins deploys the code. This initiates the job for the application.

8 The code enters the deployment phase cycle.

9 The code is deployed to the server through the docker container.

10 The code is then deployed on the production server once it passes unit testing.

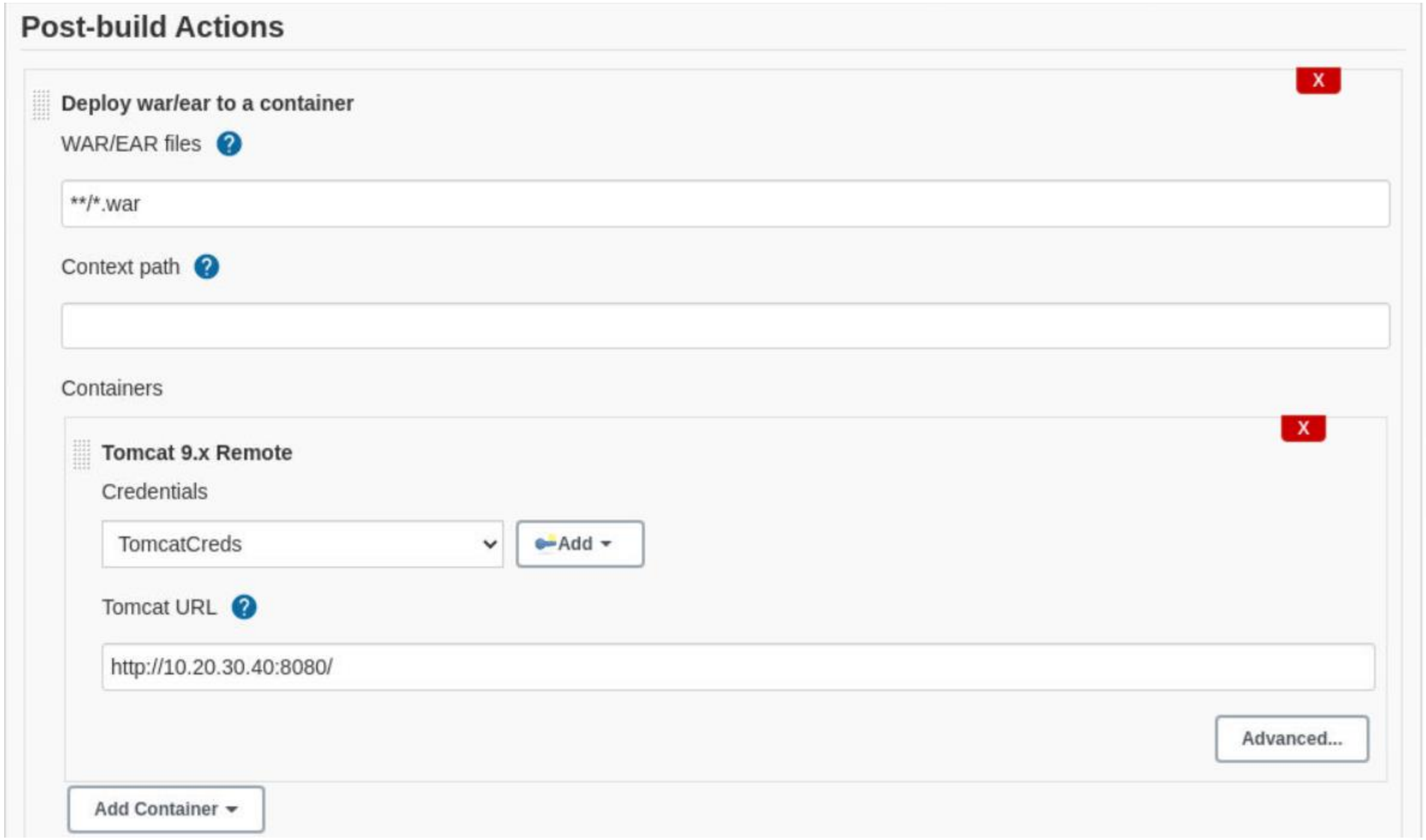
Continuous Integration vs Continuous Delivery vs Continuous Deployment



Continuous Integration vs Continuous Delivery vs Continuous Deployment

Continuous Integration	Continuous Delivery	Continuous Deployment
Focuses on build automation and test cases execution	Focuses on delivering software artifacts to clients	Focuses on performing automated deployment to production environment
Gets executed the moment a developer performs a new commit to the source code repository	Helps to continuously deliver the software, but deployment is manual	Helps developers to deploy the source code directly to the production environment
Helps developers to understand if the source code is having bugs or issues by running unit test cases	Helps developers to rectify software updates	Helps developers to deploy the source code frequently to production environment
With CI, unit test cases execution and static code scan is performed.	With Continuously Delivery, source code binaries are packaged and uploaded to the artifactory.	With Continuous Deployment, source code deployment id performed automatically once the development is completed.

Deployment Plugin for Jenkins



Post-build Actions

Deploy war/ear to a container

WAR/EAR files: `**/*.war`

Context path:

Containers

Tomcat 9.x Remote

Credentials: TomcatCreds

Tomcat URL: `http://10.20.30.40:8080/`

Image Source: Jenkins.io

Deployment Plugin for Jenkins

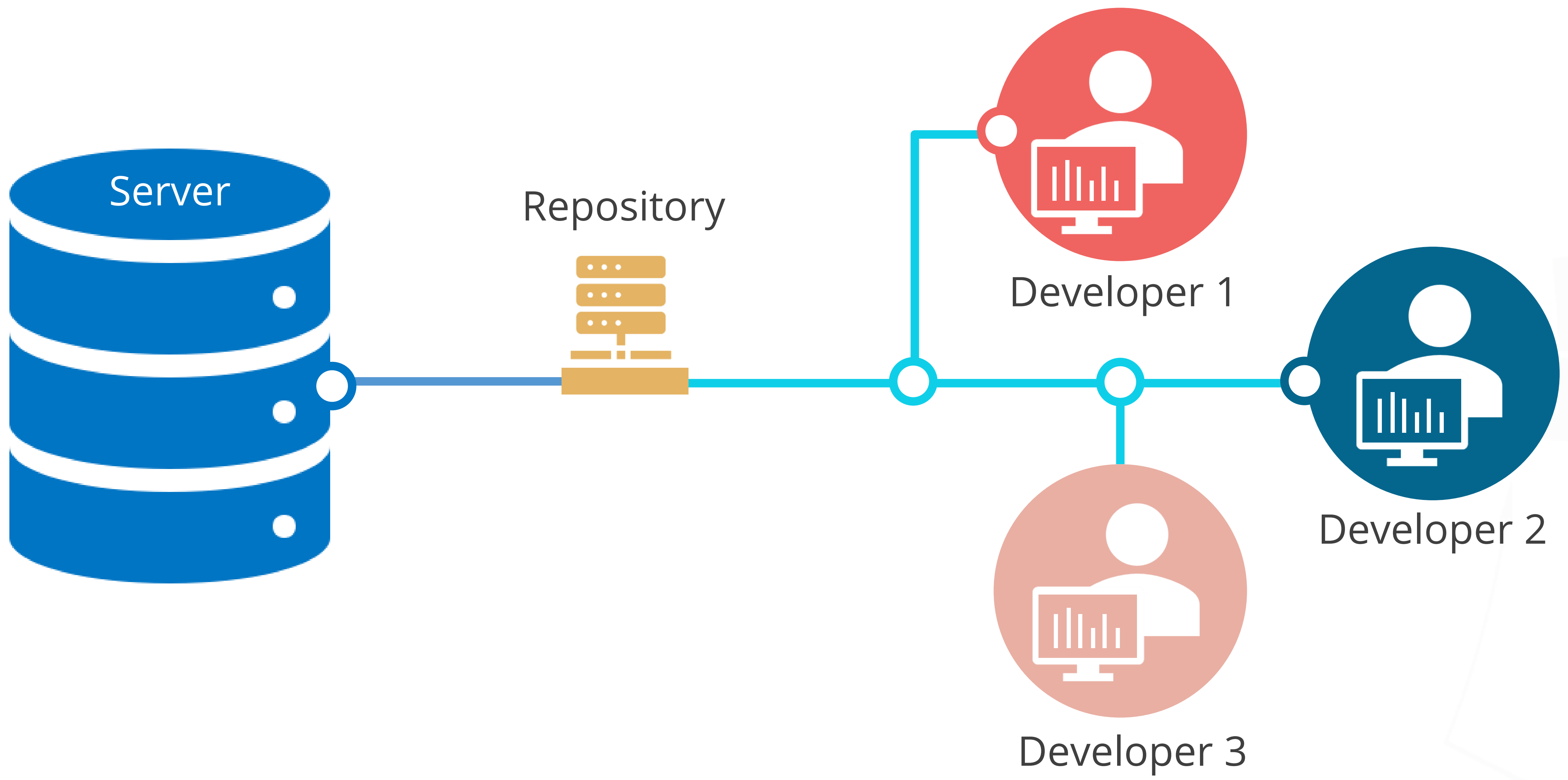
The deployment plugin supports the deployment of artifacts to **Tomcat, Jboss and Glassfish**.

In case of a necessity to deploy to **WebSphere and WebLogic**, there are separate deployment plugins for them.

Source Control Management

Source Code Management Integration

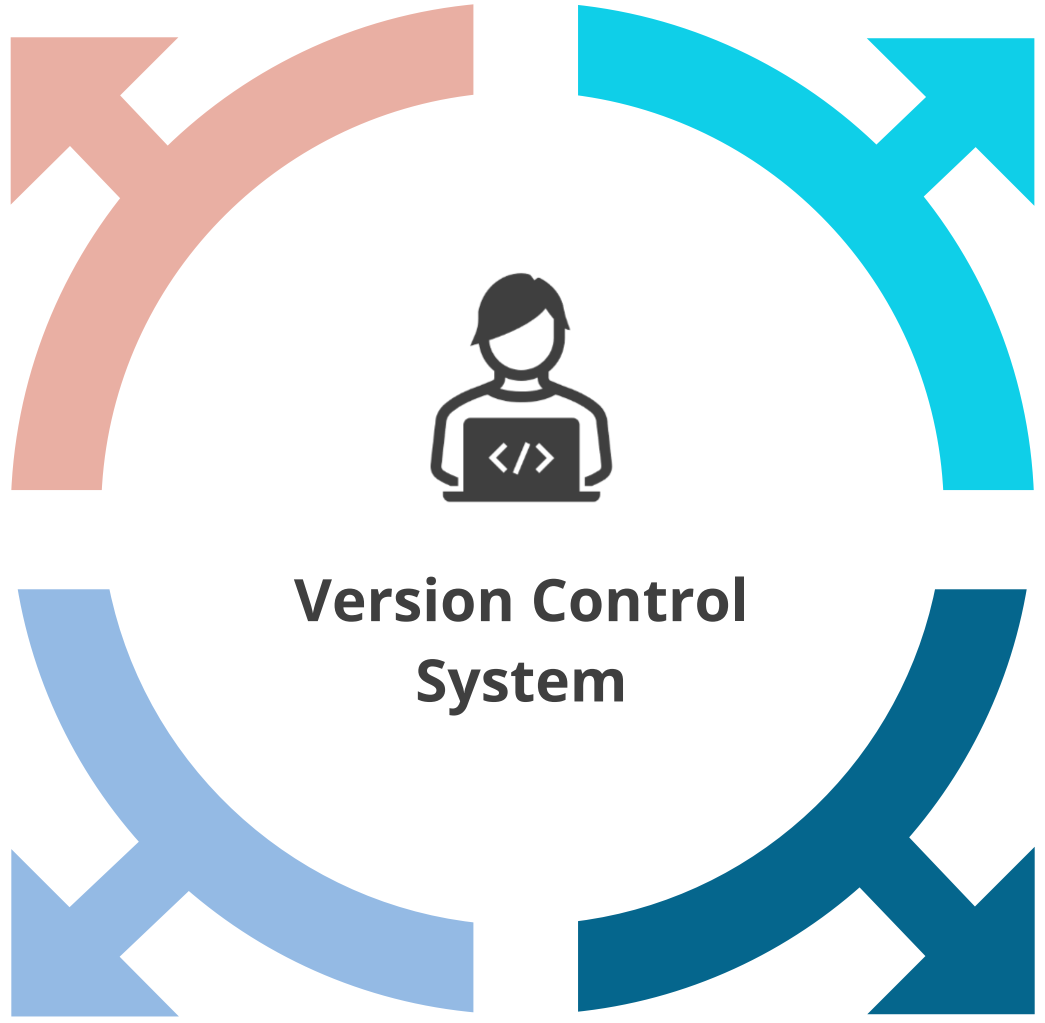
Source code management helps developers work together on code and separate their tasks through branches.



Source Code Management Integration

Facilitates the separation of tasks through branches

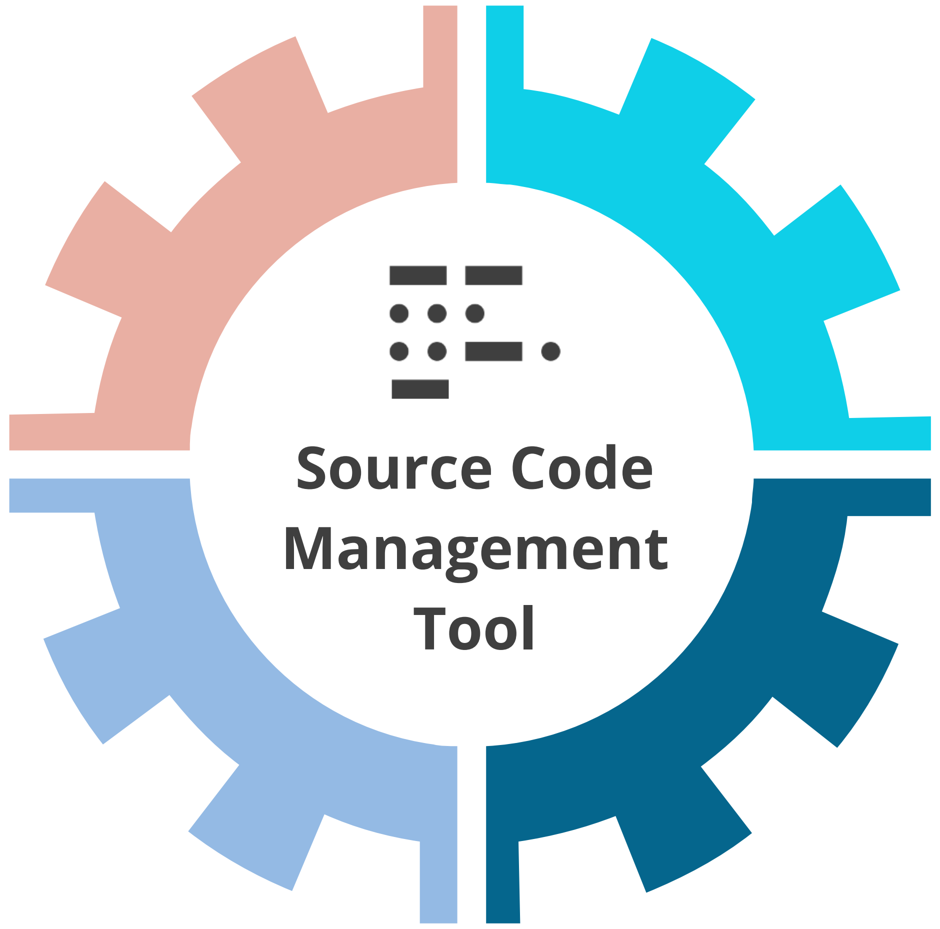
Allows users to keep track of the changes in software development projects



Helps developers to work together on code

Enables collaboration on projects

Source Code Management Integration



Modification of the source code in the repository

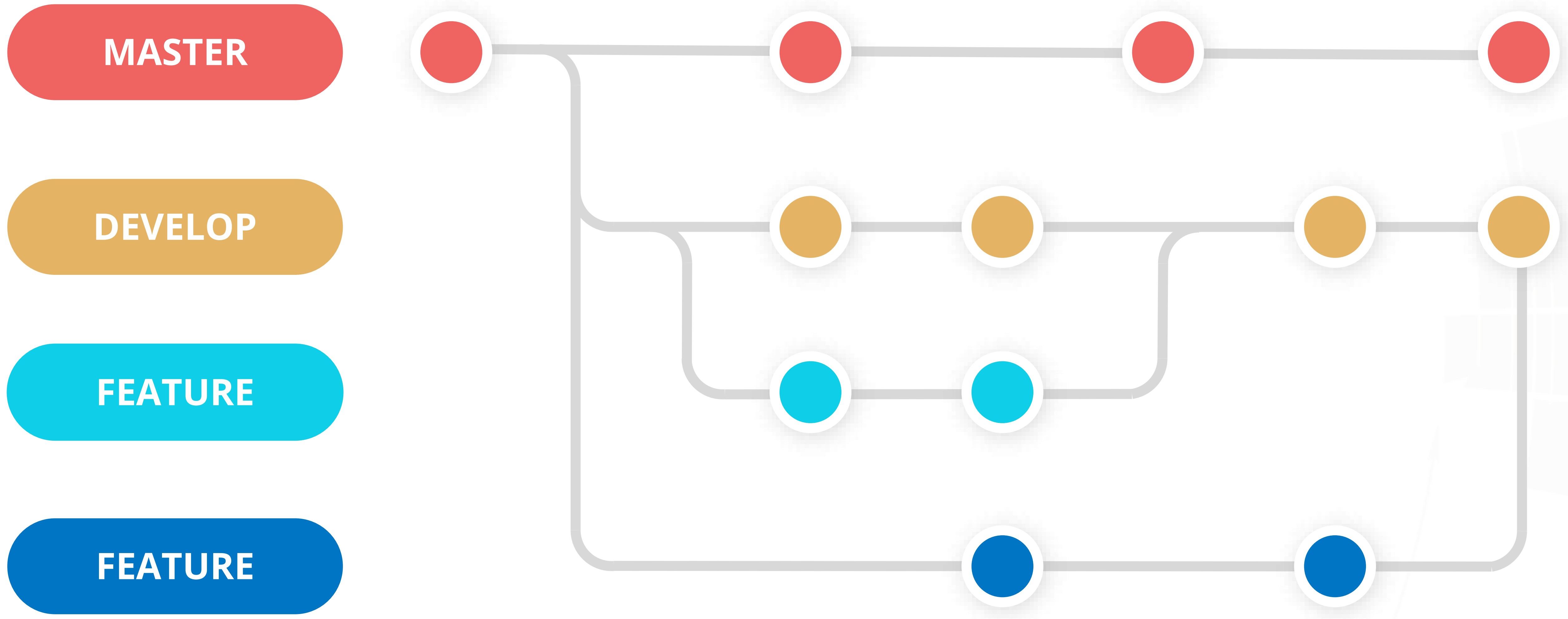
Availability of activities to manage source code in repositories

Ability to revert

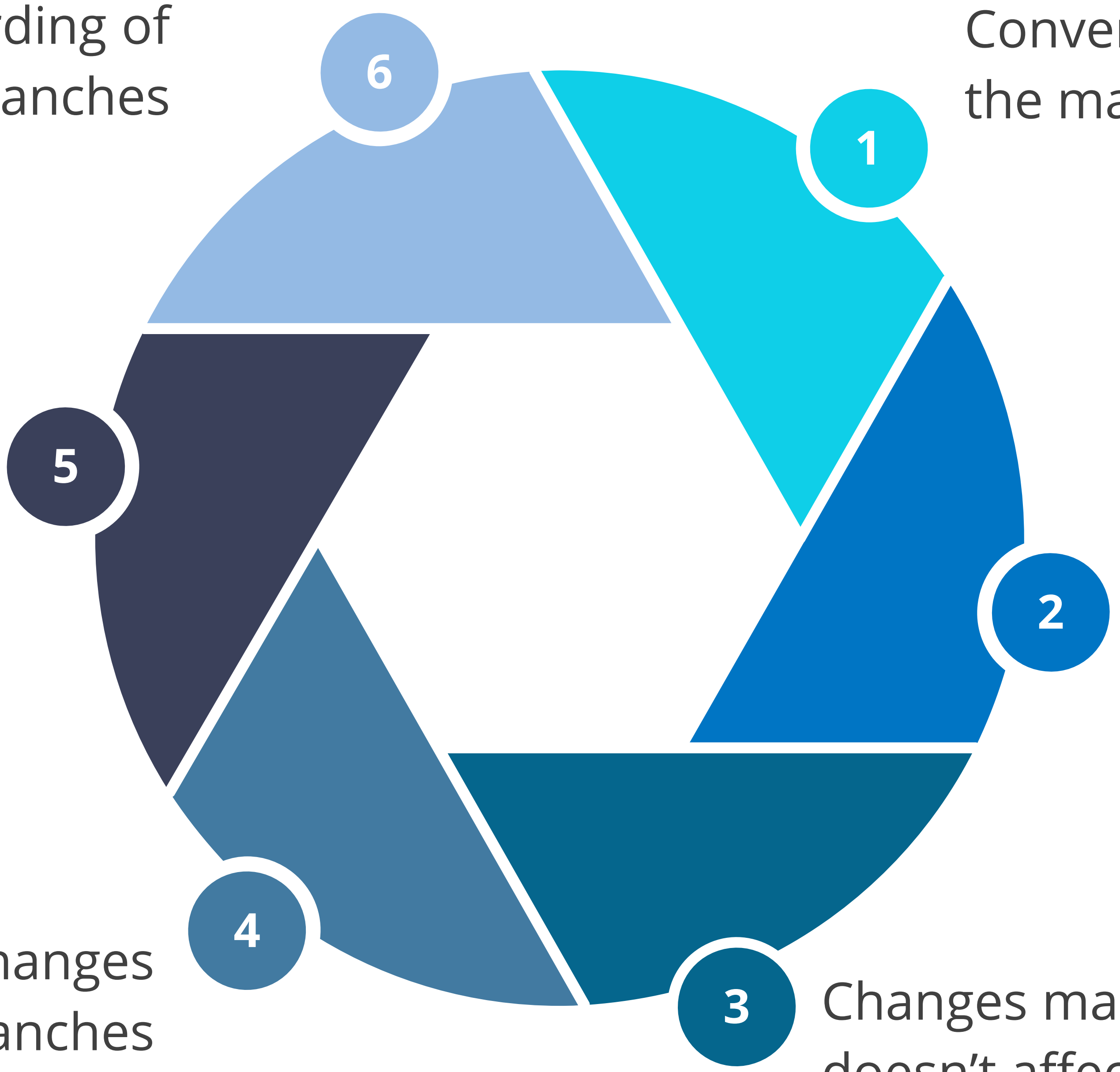
Availability of file history

Branching

Visual representation of Gitflow Workflow is as show below:



Importance of Branching



Convenient divergence from the mainstream branch

Easy collaboration

Changes made in one branch doesn't affect the other

Easy merging of changes within branches

Parallel work on multiple features

Simple discarding of feature branches

Code Scanning Tools

Source Code Vulnerabilities

Code vulnerability is termed as the flaws in an application source code that are at potential risk.

- Design flaw
- Implementation flaw
- Security flaw

Top 10 Source Code Vulnerabilities

1. Injection flaws
2. Broken authentication
3. Cross Site Scripting (XSS)
4. Sensitive data exposure
5. Cross Site Request Forgery (CSRF)
6. Memory Leak
7. Insecure Deserialization
8. Security Misconfiguration
9. XML External Entities
10. Insufficient Logging and Monitoring

Source Code Scan Integration with Pipeline

Static analysis identifies defects, vulnerabilities, and compliance issues by scanning the source code without deploying it.

Integration of code scan tools within the Continuous Integration pipeline
Early-stage identification of issues and bugs

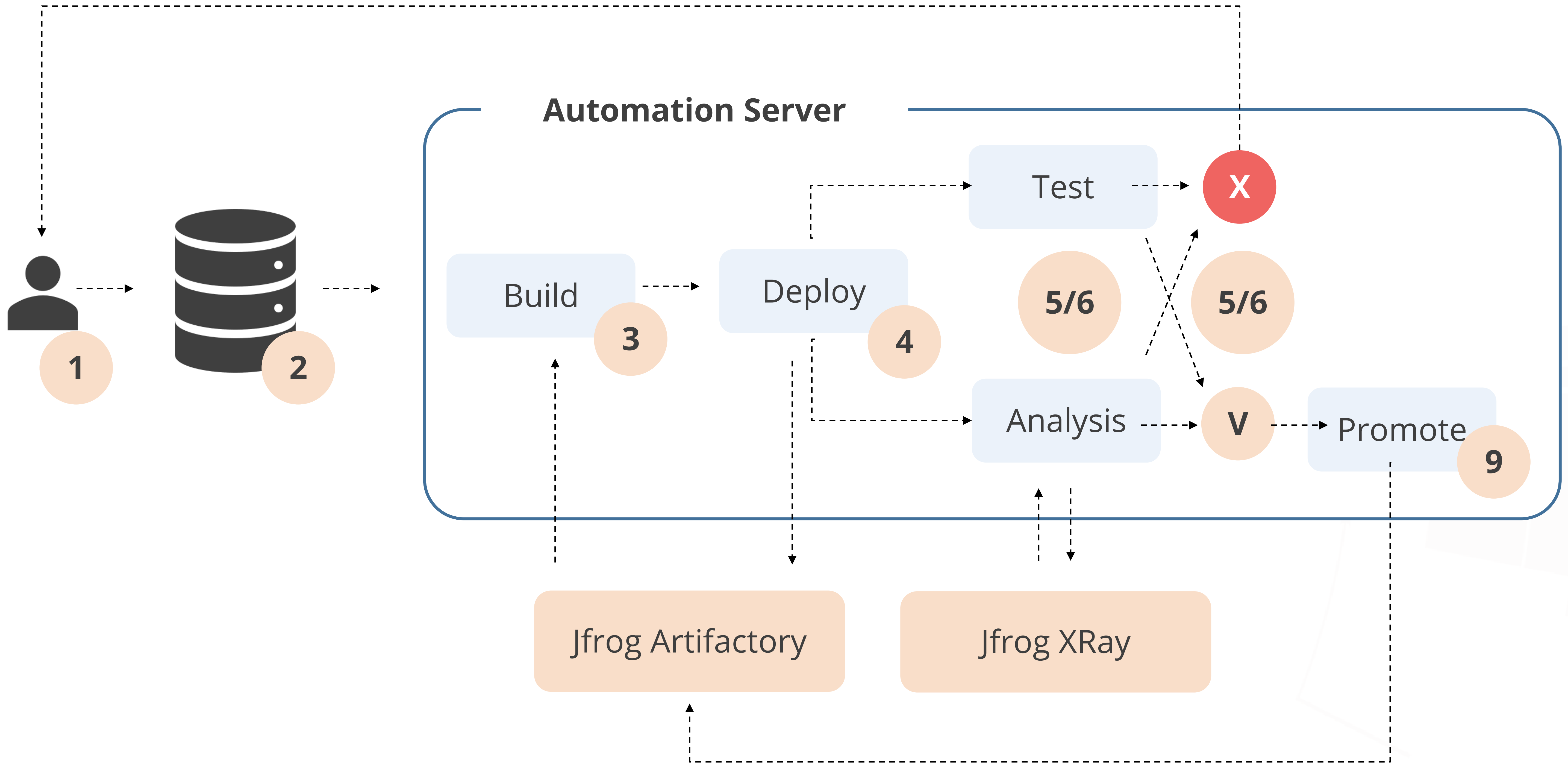
Scheduling of code scans
Elimination of the time needed for running code scans for each commit

All-round scanning
Identification of common bugs and non-standard codes instances

Benefits of Code Scan Tools

- Identification and resolution of code bugs and issues
- Easy detection of security vulnerabilities and prevention of serious security flaws
- Integration of code scan tools within the pipeline to ensure good code quality
- Further inspection of memory leaks, wrong function call, and other source code-related issues

Source Code Scan Integration with Pipeline



Source Code Scan Tools

- SonarQube
- Veracode
- Black Duck
- Coverity
- Fortify Static Code Analyzer
- Smart Bear Collaborator
- PMD

Distributed Builds

Jenkins Distributed Builds

Jenkins supports a distributed build architecture, which helps to distribute builds across multiple nodes instead of a single master node.

Master node

- Manages and distributes build requests on the slave node
- Doesn't have to execute any build

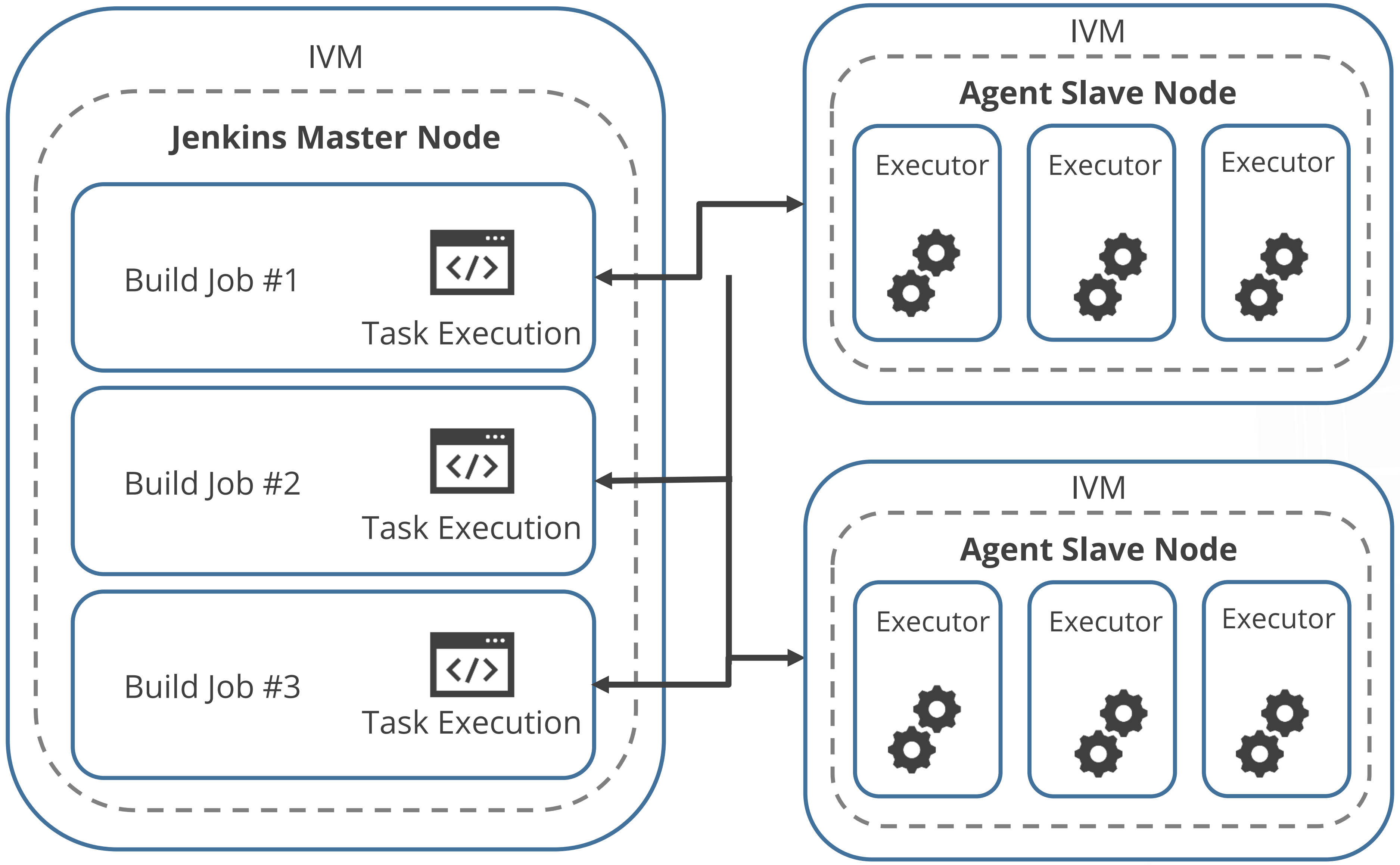
Slave nodes

- Randomly pick jobs to be built
- If any node is down, another slave machine will pick the build request.

Jenkins Distributed Builds

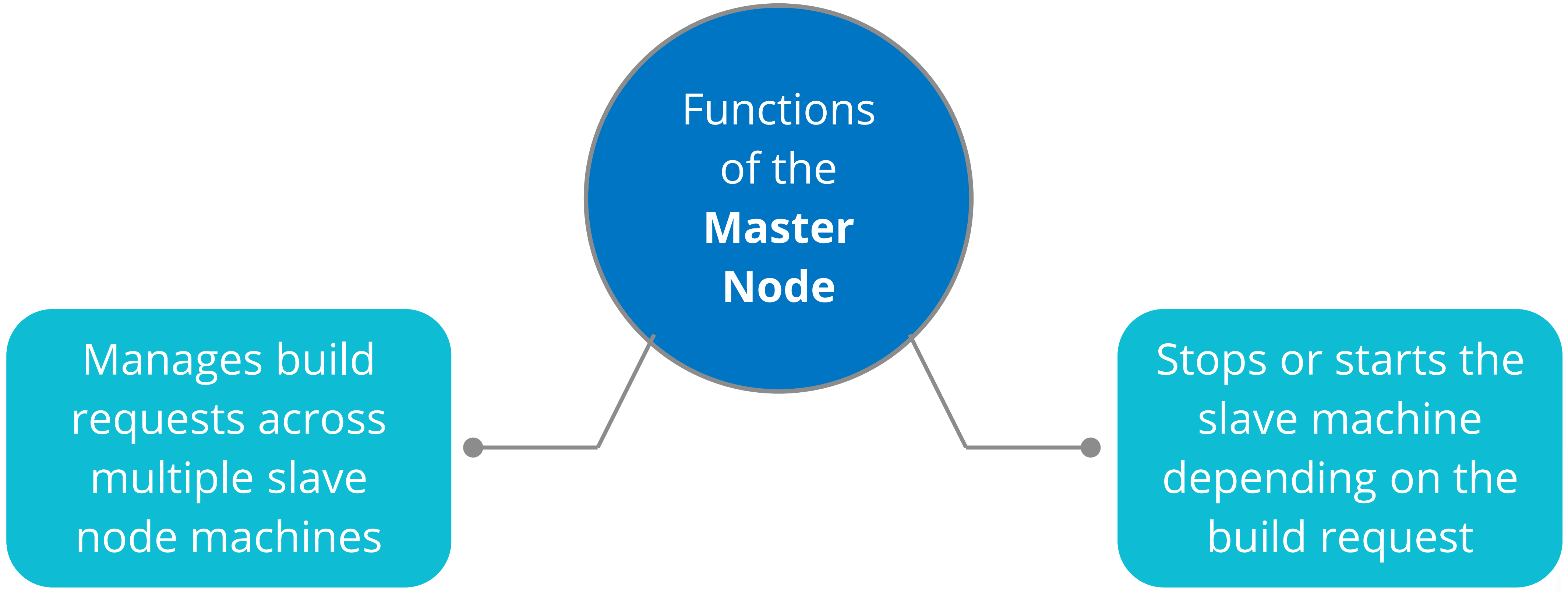
Distributed build environments use **Jenkins' master-slave architecture**.

The workload is distributed among multiple agent **nodes or slaves**.



Source: DigitalVarys

Jenkins Master-Slave Architecture



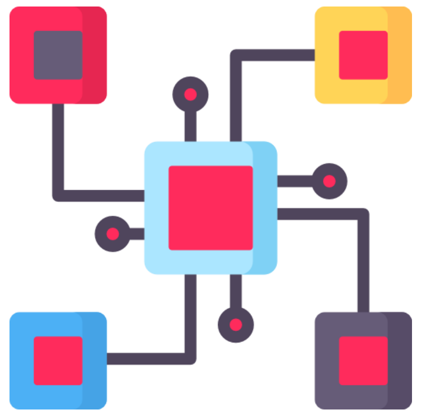
Functions of the **Master Node**

Manages build requests across multiple slave node machines

Stops or starts the slave machine depending on the build request

Jenkins Master-Slave Architecture

While adding new nodes, the master communicates with the slave using:



- SSH agent Launch
- JNLP Agent Launch
- Slave jar file Based Launch

Key Takeaways

- Continuous Integration, Continuous Delivery, and Continuous Deployment are essential solutions for the problems faced during software development.
- Continuous Integration, Continuous Delivery, and Continuous Deployment have different functions that are interrelated to each other.
- Source control management systems allow developers to collaborate and perform tasks easily.
- Static code analysis is an essential component of the end-to-end pipeline.
- The distributed build architecture of Jenkins distributes builds across multiple nodes rather than a single master node.

Thank You