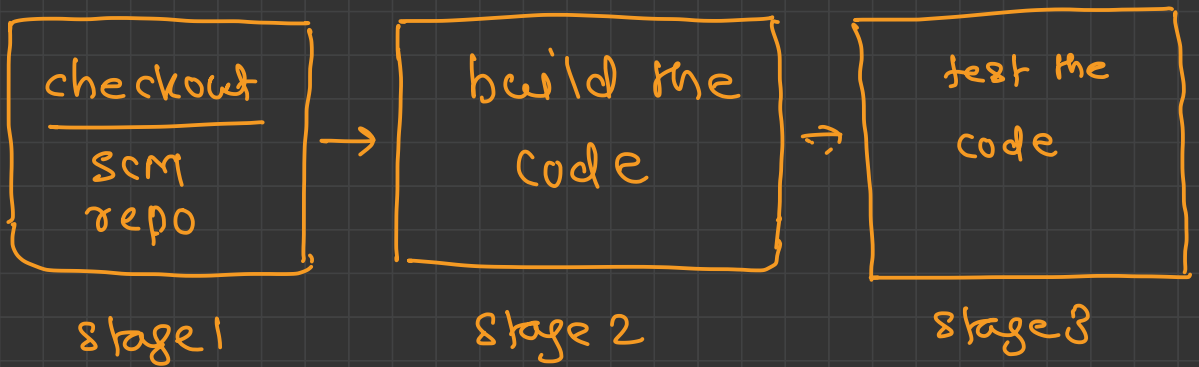
CI/CD

Continuous Integration

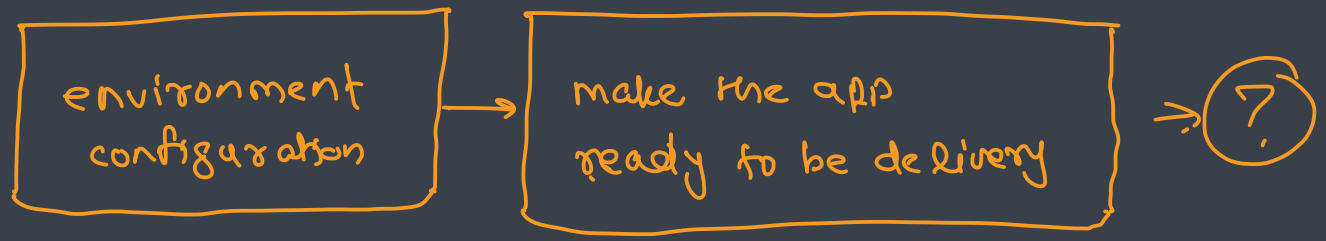
- Continuous integration is the practice of integrating all your code changes into the main branch of a shared source code repository early and often, automatically testing each change when you commit or merge them, and automatically kicking off a build
- With continuous integration, errors and security issues can be identified and fixed more easily, and much earlier in the software development lifecycle
- By merging changes frequently and triggering automatic testing and validation processes, you minimize the possibility of code conflict, even with multiple developers working on the same application
- A secondary advantage is that you don't have to wait long for answers and can, if necessary, fix bugs and security issues while the topic is still fresh in your mind
- Common code validation processes start with a static code analysis that verifies the quality of the code
- Once the code passes the static tests, automated CI routines package and compile the code for further automated testing
- CI processes should have a version control system that tracks changes, so you know the version of the code used

C.I.


checkout
scm
repo
build the
code
test the
code
stage1
stage 2
stage3

Continuous Delivery

- Continuous delivery is a software development practice that works in conjunction with continuous integration to automate the infrastructure provisioning and application release process
- Once code has been tested and built as part of the CI process, continuous delivery takes over during the final stages to ensure it can be deployed's packaged with everything it needs to deploy to any environment at any time
- Continuous delivery can cover everything from provisioning the infrastructure to deploying the application to the testing or production environment
- With continuous delivery, the software is built so that it can be deployed to production at any time
- Then you can trigger the deployments manually or move to continuous deployment where deployments are automated as well



Continuous Deployment

- Continuous deployment is a strategy in software development where code changes to an application are released automatically into the production environment
- This automation is driven by a series of predefined tests
- Once new updates pass those tests, the system pushes the updates directly to the software's users
- Continuous deployment offers several benefits for enterprises looking to scale their applications and IT portfolio
- First, it speeds time to market by eliminating the lag between coding and customer value—typically days, weeks, or even months
- In order to achieve this, regression tests must be automated, thereby eliminating expensive manual regression testing
- The systems that organizations put in place to manage large bundles of production change—including release planning and approval meetings—can also be eliminated for most changes

CI/CD Pipeline [collection of stages]

- A CI/CD pipeline automates the process of software delivery
- It builds code, runs tests, and helps you to safely deploy a new version of the software
- CI/CD pipeline reduces manual errors, provides feedback to developers, and allows fast product iterations
- CI/CD pipeline introduces automation and continuous monitoring throughout the lifecycle of a software product
- It involves from the integration and testing phase to delivery and deployment. These connected practices are referred as CI/CD pipeline

Pipeline Stages

- **Build** → yarn build
  - This phase is part of the continuous integration process and involves the creation and compiling of code
  - Teams build off of source code collaboratively and integrate new code while quickly determining any issues or conflicts
- **Test** → test automation
  - At this stage, teams test the code
  - Automated tests happen in both continuous delivery and deployment
  - These tests could include integration tests, unit tests, and regression tests
- **Deliver**
  - Here, an approved codebase is sent to a production environment
  - This stage is automated in continuous deployment and is only automated in continuous delivery after developer approval
- **Deploy**
  - Lastly, the changes are deployed and the final product moves into production
  - In continuous delivery, products or code are sent to repositories and then moved into production or deployment by human approval
  - In continuous deployment, this step is automated

Best Practices

- Write up the current development process therefore, you can know the procedures that require to change and one that can be easily automated
- Start off with a small proof of project before going ahead and complete whole development process at once
- Set up a pipeline with more than one stage in which fast fundamental tests run first
- Start each workflow from the same, clean, and isolated environment
- Run open source tools that cover everything from code style to security scanning
- Setup a better code hub to continuously check the quality of your code by running the standard set of tests against every branch
- Peer code review each pull request to solve a problem in a collaborative manner
- You have to define success metrics before you start the transition to CD automation. This will help you to consistently analyze your software, developing progress help refining where needed

Advantages

- Builds and testing can be easily performed automatically
- It can improve the consistency and quality of code
- Improves flexibility and has the ability to ship new functionalities
- CI/CD pipeline can streamline communication
- It can automate the process of software delivery
- Helps you to achieve faster customer feedback
- CI/CD pipeline helps you to increase your product visibility
- It enables you to remove manual errors
- Reduces costs and labor
- CI/CD pipelines can make the software development lifecycle faster
- It has automated pipeline deployment
- A CD pipeline gives a rapid feedback loop starting from developer to client
- Improves communications between organization employees
- It enables developers to know which changes in the build can turn to the brokerage and to avoid them in the future
- The automated tests, along with few manual test runs, help to fix any issues that may arise

Why is matters ?

- CI/CD pipeline can improve reliability
- It makes IT team more attractive to developers
- CI/CD pipeline helps IT leaders, to pull code from version control and execute software build
- Helps to move code to target computing environment
- Enables project leaders to easily manage environment variables and configure for the target environment
- Project managers can publish push application components to services like web services, database services, API services
- Providing log data and alerts on the delivery state
- It enables programmers to verify code changes before they move forward, reducing the chances of defects ending up in production

Tools

- Jenkins
  - Jenkins is an open-source Continuous Integration server that helps to achieve the Continuous Integration process (and not only) in an automated fashion.
  - Jenkins is free and is entirely written in Java

- Bamboo
  - It is a continuous integration build server that performs – automatic build, test, and releases in a single place
  - It works seamlessly with JIRA software and Bitbucket
  - It is developed by Atlassian

- CircleCi
  - It is a flexible CI tool that runs in any environment like a cross-platform mobile app, Python API server, or Docker cluster
  - This tool reduces bugs and improves the quality of the application

- Travis CI
  - Travis CI tool can easily integrate with the common cloud repositories like GitHub and Bitbucket
  - It offers many automated CI options which cut out the need for a dedicated server as the Travis CI server is hosted in the cloud
  - This allows you to test in different environments, on various machines, running on different Operating Systems

Jenkins

What is Jenkins ?

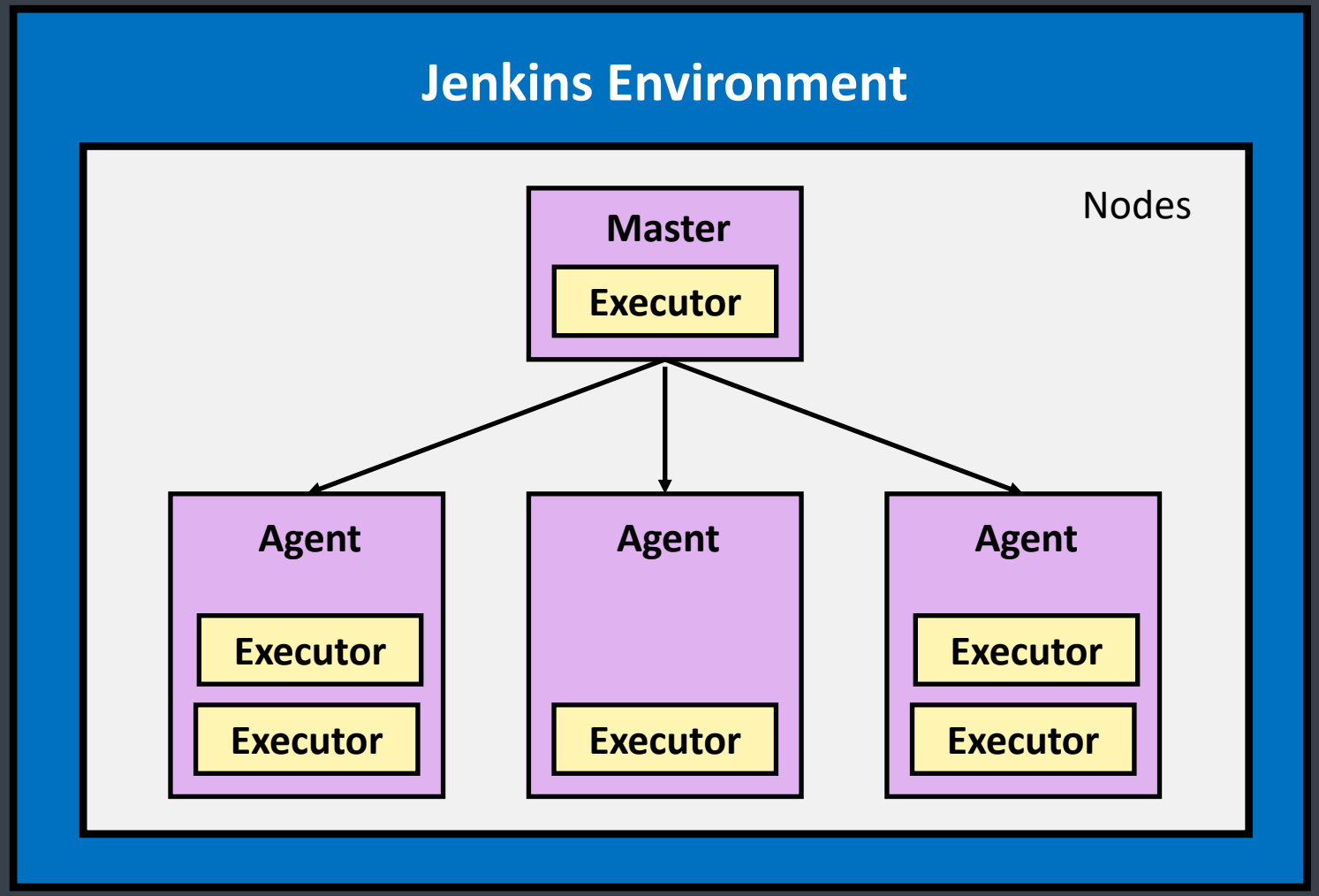
Hudson → Java → Sun

- Jenkins is a powerful application that allows continuous integration and continuous delivery of projects
- It is a free and open source application that can handle any kind of build or continuous integration
- The Jenkins project was started in 2004 (originally called Hudson) by Kohsuke Kawaguchi, while he worked for Sun Microsystems
- He was a developer at Sun and got tired of incurring the wrath of his team every time his code broke the build
- He created Jenkins as a way to perform continuous integration – that is, to test his code before he did an actual commit to the repository, to be sure all was well
- The Jenkins project was originally named Hudson, and was renamed in 2011 after a dispute with Oracle, which had forked the project and claimed rights to the project name
- The Oracle fork, Hudson, continued to be developed for a time before being donated to the Eclipse Foundation.
- Oracle's Hudson is no longer maintained and was announced as obsolete in February 2017

Features

- Easy installation on different operating systems
- Supports pipelines as code that uses **domain-specific language (DSL)** to model application delivery pipelines as code
- Easily extensible with the use of third-party plugins
- Easy to configure the setup environment in the user interface (dashboard)
- Master slave architecture supports distributed builds to reduce the load on CI servers
- Build scheduling based on cron expressions
- Shell and Windows command execution that makes any command-line tool integration in the pipeline very easy
- Notification support related to build status
  - → email / slack / sms ...

Jenkins Environment



Terminologies

- **Node**
  - Node is the generic term that is used in Jenkins to mean any system that can run Jenkins jobs
  - This covers both masters and agents, and is sometimes used in place of those terms
  - Furthermore, a node might be a container, such as one for Docker
- **Master**
  - A Jenkins *master* is the primary controlling system for a Jenkins instance
  - It has complete access to all Jenkins configuration and options and the full list of jobs
  - It is the default location for executing jobs if another system is not specified
  - Master node must be present in Jenkins installation
- **Agent**
  - Is also known as Jenkins slave
  - This refers to any non-master system
  - The idea is that these systems are managed by the master system and allocated as needed, or as specified, to handle processing the individual jobs

Terminologies

- **Executor**
  - It is a slot in which to run a job on a node/agent
  - A node can have zero or more executors
  - The number of executors defines how many concurrent jobs can be run on that node
  - When the master funnels jobs to a particular node, there must be an available executor slot in order for the job to be processed immediately. Otherwise, it will wait until an executor becomes available.

What is Jenkins pipeline ?

- Jenkins is, fundamentally, an automation engine which supports a number of automation patterns
- Pipeline adds a powerful set of automation tools onto Jenkins, supporting use cases that span from simple continuous integration to comprehensive CD pipelines
- It is a suite of plugins which supports implementing and integrating continuous delivery pipelines
- A continuous delivery (CD) pipeline is an automated expression of your process for getting software from version control right through to your users and customers
- Every change to your software (committed in source control) goes through a complex process on its way to being released
- This process involves building the software in a reliable and repeatable manner, as well as progressing the built software (called a "build") through multiple stages of testing and deployment