Jenkins:

* Jenkins is an open source and java based Continous Integration automation tool.
* It manages and controls several stages of the software delivery process including build, documentation, automated testing, packaging and static code analysis.
* Jenkins is a highly popular devops tool used by thousands of development teams.
* Jenkins automation is generally triggered by code changes in repositories like Github and integrates with the build tools like Maven.
* Jenkins supports container technologies like Docker, Maven, Github, Kubernetes
* Jenkins port number - 8080
* Jenkins overcomes the drawbacks of Waterfall model and agile Methodology.
* Jenkins is a heart of Devops
* Helps to integrate all the tools and deploy the application on the server.
* Supports 1800+ plugins and these plugins help to accelerate the activities like building, deploying and automating the project.
* Jenkins uses CI/CD pipelines.
* Pipelines rapidly detect the defects in a codebase, build software, automates the testing of builds, and prepare for the deployment of the codebase to the containers or virtual machines.
* Jenkins is a fork project of HUDSON, trademarked by Oracle.
* Jenkins development is now managed as an open source project under the governance of CD foundation.
* Jenkins is a server based automation tool.
* Operates within the servlet containers.
* Maven, Git, Dockers, Kubernetes, Ansible and Terraform will use the same instance.
* For deployment we use another instance.
* Jenkins makes the tasks easy for developers by automating, they can concentrate on the betterment of the product by integrating the changes to the project.
* Jenkins automates the build in continous manner and let’s the developers know the errors at an early stage.
* Instead of Versions it maintains Configurations.
* Execution in both Sequential and Parallel jobs are done.

**Why use Jenkins?**

* **Automation:** 
  + It helps in automating repetitive tasks like building, testing and deploying the applications.
  + Automates SDLC and reduces manual effort.
* **CI:**
  + Developers can integrate their code frequently into a shared repository.
  + Detects the changes in version control and triggers automatic builds and tests.
* **CD:**
  + Enables automatic deployments of applications to different environments.
  + Ensures faster and more reliable release cycle.
* **Plugin Ecosystem:**
  + Jenkins supports 1800+ plugins.
  + Whether you need to integrate with version control systems like Git, build tools like Maven, or deploy to cloud platforms, there's likely a plugin for that.
* **Scalability and Flexibility:**
  + Can distribute builds across multiple machines.
  + Works with both On-Premise and Cloud environments.
* **Open Source:**
  + It is an Open source, making it a cost-effective CI/CD solution.
* **Easy Integration with devops tools:**
  + Can easily integrate with tools like Docker, Kubernetes for monitoring.

**Advantages:**

* **Highly Extensible:**
  + Supports thousands of plugins that lets you connect with the other tools like Git, Maven.
  + These plugins help to add new features and help to customize the jenkins project.
  + Can automate SDLC by using scripts and pipelines.
* **Robust and Reliable at Any Scale:**
  + Works well with both small and large projects.
  + Can handle hundreds of builds per day without breaking.
  + If one part fails, jenkins will notify immediately.
* **Mature and Battle-Tested:**
  + Jenkins has been around for a while and has been widely used in the software industry.
  + Many big companies trust Jenkins because it is stable, secure, and constantly improving.
* **Supports Hybrid and Multi-Cloud Environments:**
  + Can work with on-premise, public cloud and hybrid environments.
  + Can develop and deploy the applications anywhere without having the difference of cloud environments.
* **Large Knowledge Base, Documentation, and Community Support:**
  + There are tons of tutorials, guides, and forums to help you solve any Jenkins issue.
  + A large community of developers regularly updates and improves.
  + If you get stuck, you can find solutions easily.
* **Built with Java for Enterprise Use:**
  + Built with Java, widely used in many Organizations.
  + This makes it easy to integrate with older enterprise systems that also run on Java.

**Disadvantages:**

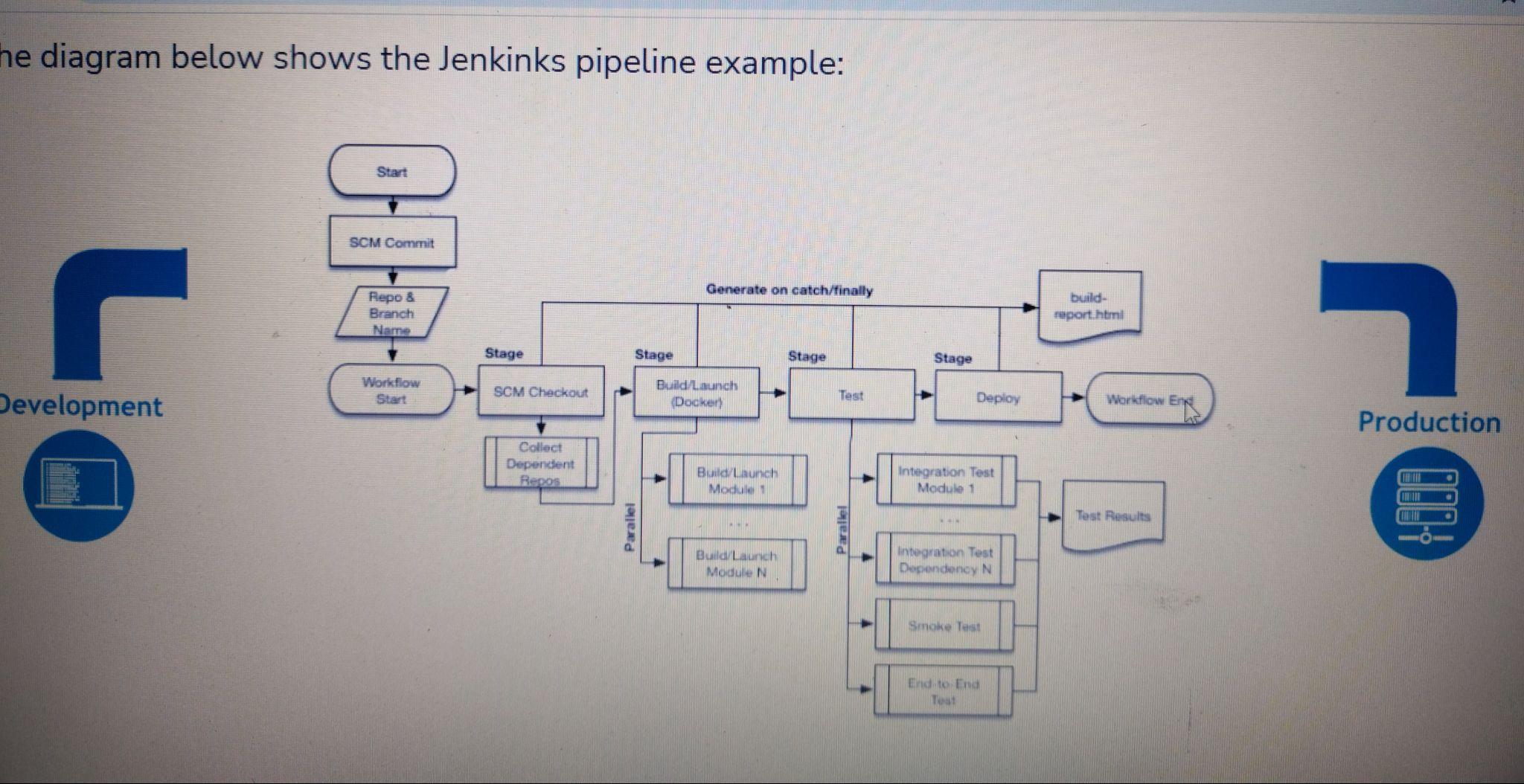
* **Complex Setup:** Initial setup and configuration can be challenging especially for beginners. It may take some time to understand.
* **Jenkins Sprawl:** Different teams may set up their own Jenkins servers, leading to too many separate instances.Managing all these servers becomes confusing and difficult.
* **Uses old Java Technology:** Based on old java frameworks like Maven. Jenkins doesn’t fully support newer technologies like GradleVM.
* **Hard to Set Up for Large Projects:** Creating CI/CD pipelines in Jenkins requires coding in Groovy (a scripting language).Writing and debugging complex pipelines can be tricky.
* **Groovy Scripting is Hard to Learn:**Jenkins pipelines are written in Groovy, a less common programming language.Developers may struggle to learn and maintain Jenkins scripts.

**Jenkins Architecture Core Concepts:**

* **Jenkins Master:** Supports distributed builds. One node functions as an Organiser and that node is known as master/controller. This node manages the jenkins agents. It manages agents and their connections, loads plugins, and coordinates project flow.
* **Jenkins Agent:** Connects to the Master to run builds. To run it, you’ll need to install Java on a physical machine, virtual machine, cloud compute instance, Docker image, or Kubernetes cluster.
* **Jenkins node:** A Jenkins node is an umbrella term for Agents and Controllers, regardless of their actual roles. A node is a machine on which you can build projects and pipelines. Jenkins automatically monitors the health of all connected nodes
* **Jenkins Project:** Jenkins project/ a task is an automated process created by a jenkins user. The plain jenkins distribution offers a variety of build tasks that can supports continous integration workflows and available through a large ecosystems.
* **Jenkins plugins:** Plugins are community developed modules you can install on a jenkins server. This adds features that jenkins doesn't have by default. You can install/upgrade those plugins by using jenkins dashboard.
* **Jenkins Pipeline:** Jenkins is a user created pipeline model. This includes a variety of plugins that helps you to define step-by-step actions in your software pipeline This includes: Automated builds, Multi-Step Testing, Deployment Procedures and Security Scanning.
* **Jenkins Triggers:**
  + Triggers are the actions or build that initiate the build or deploy actions
  + SCM triggers: Jenkins polls changes for SCM repository. When the changes are found then jenkins starts for executing the pipeline jobs and processes the latest source code.
  + **Parameterized Triggers:** this triggers the build to be triggered based on user-defined inputs.
  + **Manual Triggers:**you may explicitly trigger the pipeline by logging into the jenkins environment.
  + **Time-based Triggers:** pipelines can be run on a specific schedule. A cron job is set to start the pipeline execution after every set interval.
  + **Web Hooks:** when the pipeline runs depending on the same event occurring in an external platform, we can use webhooks to trigger the pipeline execution in jenkins.

**Jenkins Architecture:**

* Developers modify the source code, committing changes to the repository, and Jenkins creates a new build in order to handle the new Git commit.
* Jenkins can work in “push” or “pull” mode. The Jenkins CI server is either triggered by an event such as a code commit, or it can regularly check the repository for changes.
* The build server builds the code and generates an artifact. If the build fails, the developer receives an alert.
* Jenkins deploys the built application/executable to the test server, which can execute continuous, automated tests. Developers receive alerts if their changes impact functionality.
* Jenkins optionally deploys the changes to the production server if the code has no issues.



**Jenkins Important Configuration Files:**

* Jenkins home directory: /var/lib/jenkins
* Installed Plugins: /var/lib/jenkins/plugins
* Created Job List: /var/lib/jenkins/workspace
* Nodes info: /var/lib/jenkins/nodes
* Jenkins log info: /var/lib/jenkins/log
* List of jobs: /var/lib/jenkins/jobs
* Created users list: /var/lib/jenkins/users

**Jenkins Installation Steps:**

* We need to take an EC2 instance.
* We need to install Java.
* We need to install Git.
* We need to install Maven.
* We need to install Jenkins.
* Start the service of Jenkins.
* Copy the public IP of jenkins and paste it in the URL with :8080 extension.
* Jenkins Dashboard.

**Types of Environments:**

* **Development:** A development environment is a setup where software developers create, test, and debug their applications.
* **Quality Assurance:** A QA (Quality Assurance) environment is a setup where software testers ensure that an application or system works as intended before it's released to production.
* **UAT:A UAT** (User Acceptance Testing) environment is a setup where end users or clients test the software to ensure it meets their requirements and works as expected before it goes live.
* **Production:**A production environment is where the final version of your software or application is deployed for end users to use. It's the live environment where real users interact with your application.

**Need of Environment:**

* Ensure to deliver error free application
* Productivity improvement
* Ensure Customer Satisfaction
* To deliver the software in time

**CI/CD:**

* It is an approach to software Development that combines the practices of CI/CD to make the development faster,safer and more efficient.

**Need of CI/CD:**

* **Faster Developments:** allows developers to integrate changes so quickly, leading to a faster development lifecycle.This helps to deploy new features or bug fixes to the clients quickly.
* **Automated Testing:** automates the testing process whenever the code changes were made. This reduces the manual effort required from developers and allows them to focus on more important tasks.
* **Reduced Manual Effort:** ensures that deployment process is consistent and repeatable.Reduces the risk of errors in the deployment and makes the software more reliable.
* **Consistent Deployments:**CI/CD ensures that the deployment process is consistent and repeatable. This reduces the risk of errors during deployment and makes the process more reliable.
* **Early bug Detection:** by continous integration and testing code changes frequently, errors can be detected and fixed early.
* **Improved Collaboration:** encourages better collaboration between Development,QA and Operational teams.It encourages a culture of shared responsibility and continuous improvement.
* **Scalability:** pipelines can be scaled easily to handle larger projects and more frequent changes.this makes it suitable for both large and smaller teams.
* **Continous Feedback:** provides continous feedback to the users about the quality and the performance of the software.This helps them make the decisions and improve their work further.
* **User Satisfaction:**By delivering new features and improvements more frequently, CI/CD helps keep users happy and engaged. It also allows for faster responses to user feedback and changing market demands.

**Key Features of CI:**

* **Automated Builds:** CI tools automatically compile and build the code whenever the changes are made and ensure that the application is always in a working state.
* **Automated Testing:** Integrates automated tests that run every time the new code is added.This helps to catch the bugs early and ensure that it doesn't break the existing functionality.
* **Frequent Commits:** Developers are encourages to integrate their code commits frequently.This allows complexity of merging code and helps to identify the issues in the early stages.
* **Version Control Integration:** Provides immediate feedback to the developers whenever the code changes are made. This helps them to identify the code changes quickly and helps them to fix bugs early.
* **Immediate feedback:** ensures that tests run in an consistent environment
* **Build Artifacts:** generated build artifacts that can be tested for further testing and deployment. These artifacts are stored and visioned.

**Key Features of CD:**

* **Auto Deployment:** automates the process of deploying the code changes to the production. Reduces the human intervention and fastens the releases.
* **Continous Deployment Pipeline:** CD pipelines are the steps that take the code from development to the production. It includes the development, testing and deploying the application.
* **Frequent Releases:** enables frequent and smaller releases, allows new features and updates and fixes the bug more quickly.
* **Automated Testing:** relies on automated tests to ensure code quality and stability.
* **Rollback Mechanism:** : In case of deployment issues, CD provides mechanisms to quickly roll back to a previous stable version, minimizing downtime and impact on users.
* **Monitoring and logging:** Continuous monitoring and logging are integral to CD. They help detect issues early and provide visibility into the deployment process.

**Types of Jobs:**

**FreeStyle job:**

* A type of job that offers a simple way to configure and run tasks.
* It's the most basic type of Jenkins project and allows you to define custom build steps and post-build actions.
* **Build Configuration:** U can set up a freestyle job with basic settings, such as Project name, Description, Where to get the code from.
* **Build Trigger:** U can configure how and when the job should be triggered.
* **Build Steps:** These are the core actions that your job performs.You can run multiple build steps, running a shell script, invoking an ANt or Maven or executing the command.
* **Post Build Actions:** after the actions were triggered, you can trigger the post-build actions such as archiving build artifacts, sending email notifications, or triggering other jobs.
* **Plugins:**Freestyle Jobs can make use of various Jenkins plugins to extend their functionality.

**Pipeline Jobs:**

* Allows you to define the entire build process as code, using simple, domain specific language called Groovy.
* This is useful for more complex and multi-stage workflows.
* **Pipeline as Code:** The entire build, test, and deploy process is defined in a script file (usually named Jenkinsfile). This makes the process version-controlled and more maintainable.
* **Stages and Steps:** Pipelines are divided into stages and steps. Each stage represents a major part of the process (e.g., Build, Test, Deploy), and within each stage, you can define individual steps to perform actions.
* **Declarative and Scripted Pipelines:** Jenkins supports two types of pipelines:
  + Declarative Pipelines: Easier to read and write, using a predefined syntax.
  + Scripted Pipelines: More flexible and powerful, allowing complex Groovy scripting.
* **Parallel Execution:** Pipelines support parallel execution, allowing multiple tasks to run at the same time, which can speed up the build process.
* **Integration with Tools:** Pipelines can integrate with various tools and services, such as Git, Maven, Gradle, Docker and more.
* **Environment Variables:** Pipelines can use environment variables to configure the build process dynamically.
* **Error Handling and Notifications:** Pipelines can include error handling to manage failures and send notifications (e.g., email, Slack) when builds fail or succeed.
* **Reusable Code:** You can create shared libraries and functions to reuse code across multiple pipeline jobs, making your configuration more modular and maintainable.

**Parameterized Jobs:**

* Jenkins allows users to define the parameters that can be passed to the job at the time of runtime.
* This makes the job more flexible and reusable.
* **Parameters Definition:** When setting up the job you can define various types of jobs such as string parameters, choice parameters, boolean parameters and more. These parameters will be triggered by the user when they trigger the job.
* **User Input:** When triggering a parameterized job, you can use specific values for the user defined parameters.This allows to run the same job with different configurations or inputs.
* **Accessing Parameters:** you can access these parameters inside the job using environment variables.This allows you to use the provided values in your build steps, scripts, and configurations.
* **Parameter Types:**
  + **String Parameter:** A simple text input
  + **Boolean Parameter:** A checkbox
  + **Choice Parameter:** A dropdown list with predefined options.
  + **File Parameter:** Allows the users to upload a file.
  + **Password Parameter:** A secure text input that hides the entered value.