**DEVOPS**

**UNIT-I**

**Introduction to Devops:** Introduction to SDLC, Agile Model. Introduction to De- vops, DevOps Features, DevOps Architecture, DevOps Lifecycle, Understanding Work- flow and principles, Introduction to DevOps tools, Build Automation, Delivery Automa- tion, Understanding Code Quality, Automation of CI/CD.

**Introduction to Devops:**

1. **Introduction to SDLC**

Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality softwares. The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.

* SDLC is the acronym of Software Development Life Cycle.
* It is also called as Software Development Process.
* SDLC is a framework defining tasks performed at each step in the software development process.
* ISO/IEC 12207 is an international standard for software life-cycle processes. It aims to be the standard that defines all the tasks required for developing and maintaining software.

What is SDLC?

SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.

The following figure is a graphical representation of the various stages of a typical SDLC.



A typical Software Development Life Cycle consists of the following stages −

Stage 1: Planning and Requirement Analysis

Requirement analysis is the most important and fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational and technical areas.

Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage. The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks.

Stage 2: Defining Requirements

Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done through an **SRS (Software Requirement Specification)** document which consists of all the product requirements to be designed and developed during the project life cycle.

Stage 3: Designing the Product Architecture

SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification.

This DDS is reviewed by all the important stakeholders and based on various parameters as risk assessment, product robustness, design modularity, budget and time constraints, the best design approach is selected for the product.

A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third party modules (if any). The internal design of all the modules of the proposed architecture should be clearly defined with the minutest of the details in DDS.

Stage 4: Building or Developing the Product

In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.

Developers must follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers, etc. are used to generate the code. Different high level programming languages such as C, C++, Pascal, Java and PHP are used for coding. The programming language is chosen with respect to the type of software being developed.

Stage 5: Testing the Product

This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However, this stage refers to the testing only stage of the product where product defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS.

Stage 6: Deployment in the Market and Maintenance

Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometimes product deployment happens in stages as per the business strategy of that organization. The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing).

Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment. After the product is released in the market, its maintenance is done for the existing customer base.

SDLC Models

There are various software development life cycle models defined and designed which are followed during the software development process. These models are also referred as Software Development Process Models". Each process model follows a Series of steps unique to its type to ensure success in the process of software development.

Following are the most important and popular SDLC models followed in the industry −

* Waterfall Model
* Iterative Model
* Spiral Model
* V-Model
* Big Bang Model

Other related methodologies are Agile Model, RAD Model, Rapid Application Development and Prototyping Models.

1. **Agile Model.**

Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product. Agile Methods break the product into small incremental builds. These builds are provided in iterations. Each iteration typically lasts from about one to three weeks. Every iteration involves cross functional teams working simultaneously on various areas like −

* Planning
* Requirements Analysis
* Design
* Coding
* Unit Testing and
* Acceptance Testing.

At the end of the iteration, a working product is displayed to the customer and important stakeholders.

What is Agile?

Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements. In Agile, the tasks are divided to time boxes (small time frames) to deliver specific features for a release.

Iterative approach is taken and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer.

Here is a graphical illustration of the Agile Model −



The Agile thought process had started early in the software development and started becoming popular with time due to its flexibility and adaptability.

The most popular Agile methods include Rational Unified Process (1994), Scrum (1995), Crystal Clear, Extreme Programming (1996), Adaptive Software Development, Feature Driven Development, and Dynamic Systems Development Method (DSDM) (1995). These are now collectively referred to as **Agile Methodologies**, after the Agile Manifesto was published in 2001.

Following are the Agile Manifesto principles −

* **Individuals and interactions** − In Agile development, self-organization and motivation are important, as are interactions like co-location and pair programming.
* **Working software** − Demo working software is considered the best means of communication with the customers to understand their requirements, instead of just depending on documentation.
* **Customer collaboration** − As the requirements cannot be gathered completely in the beginning of the project due to various factors, continuous customer interaction is very important to get proper product requirements.
* **Responding to change** − Agile Development is focused on quick responses to change and continuous development.

Agile Vs Traditional SDLC Models

Agile is based on the **adaptive software development methods**, whereas the traditional SDLC models like the waterfall model is based on a predictive approach. Predictive teams in the traditional SDLC models usually work with detailed planning and have a complete forecast of the exact tasks and features to be delivered in the next few months or during the product life cycle.

Predictive methods entirely depend on the **requirement analysis and planning** done in the beginning of cycle. Any changes to be incorporated go through a strict change control management and prioritization.

Agile uses an **adaptive approach** where there is no detailed planning and there is clarity on future tasks only in respect of what features need to be developed. There is feature driven development and the team adapts to the changing product requirements dynamically. The product is tested very frequently, through the release iterations, minimizing the risk of any major failures in future.

**Customer Interaction** is the backbone of this Agile methodology, and open communication with minimum documentation are the typical features of Agile development environment. The agile teams work in close collaboration with each other and are most often located in the same geographical location.

Agile Model - Pros and Cons

Agile methods are being widely accepted in the software world recently. However, this method may not always be suitable for all products. Here are some pros and cons of the Agile model.

The advantages of the Agile Model are as follows −

* Is a very realistic approach to software development.
* Promotes teamwork and cross training.
* Functionality can be developed rapidly and demonstrated.
* Resource requirements are minimum.
* Suitable for fixed or changing requirements
* Delivers early partial working solutions.
* Good model for environments that change steadily.
* Minimal rules, documentation easily employed.
* Enables concurrent development and delivery within an overall planned context.
* Little or no planning required.
* Easy to manage.
* Gives flexibility to developers.

The disadvantages of the Agile Model are as follows −

* Not suitable for handling complex dependencies.
* More risk of sustainability, maintainability and extensibility.
* An overall plan, an agile leader and agile PM practice is a must without which it will not work.
* Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines.
* Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.
* There is a very high individual dependency, since there is minimum documentation generated.
* Transfer of technology to new team members may be quite challenging due to lack of documentation.

1. **Introduction to Devops**

The DevOps is the combination of two words, one is **Development** and other is **Operations**. It is a culture to promote the development and operation process collectively.

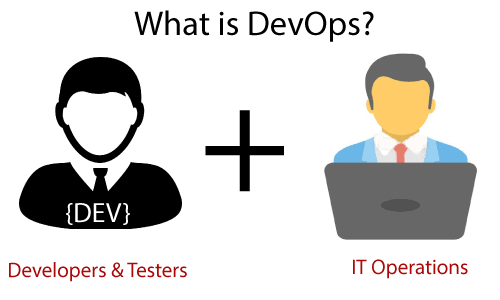
The DevOps tutorial will help you to learn DevOps basics and provide depth knowledge of various DevOps tools such as **Git, Ansible, Docker, Puppet, Jenkins, Chef, Nagios**, and **Kubernetes**.

What is DevOps?

The DevOps is a combination of two words, one is software Development, and second is Operations. This allows a single team to handle the entire application lifecycle, from development to **testing, deployment**, and **operations**. DevOps helps you to reduce the disconnection between software developers, quality assurance (QA) engineers, and system administrators.

DevOps is the combination of cultural philosophies, practices, and tools that increases an organization’s ability to deliver applications and services at high velocity: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes. This speed enables organizations to better serve their customers and compete more effectively in the market.

DevOps promotes collaboration between Development and Operations team to deploy code to production faster in an automated & repeatable way.



DevOps helps to increase organization speed to deliver applications and services. It also allows organizations to serve their customers better and compete more strongly in the market.

DevOps can also be defined as a sequence of development and IT operations with better communication and collaboration.

DevOps has become one of the most valuable business disciplines for enterprises or organizations. With the help of DevOps, **quality**, and **speed** of the application delivery has improved to a great extent.

DevOps is nothing but a practice or methodology of making "**Developers**" and "**Operations**" folks work together. DevOps represents a change in the IT culture with a complete focus on rapid IT service delivery through the adoption of agile practices in the context of a system-oriented approach.

DevOps is all about the integration of the operations and development process. Organizations that have adopted DevOps noticed a 22% improvement in software quality and a 17% improvement in application deployment frequency and achieve a 22% hike in customer satisfaction. 19% of revenue hikes as a result of the successful DevOps implementation.

Why DevOps?

Before going further, we need to understand why we need the DevOps over the other methods.

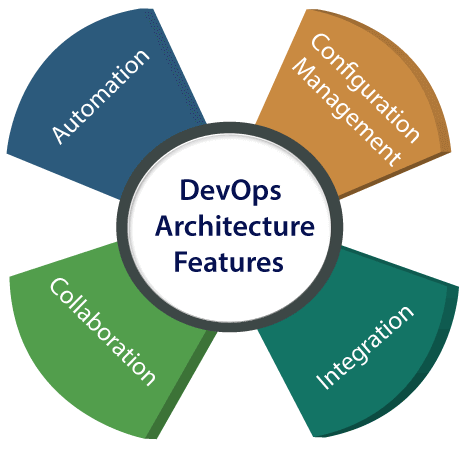
* The operation and development team worked in complete isolation.
* After the design-build, the testing and deployment are performed respectively. That's why they consumed more time than actual build cycles.
* Without the use of DevOps, the team members are spending a large amount of time on designing, testing, and deploying instead of building the project.
* Manual code deployment leads to human errors in production.
* Coding and operation teams have their separate timelines and are not in synch, causing further delays.

DevOps History

* In 2009, the first conference named **DevOpsdays** was held in Ghent Belgium. Belgian consultant and Patrick Debois founded the conference.
* In 2012, the state of DevOps report was launched and conceived by Alanna Brown at Puppet.
* In 2014, the annual State of DevOps report was published by Nicole Forsgren, Jez Humble, Gene Kim, and others. They found DevOps adoption was accelerating in 2014 also.
* In 2015, Nicole Forsgren, Gene Kim, and Jez Humble founded DORA (DevOps Research and Assignment).
* In 2017, Nicole Forsgren, Gene Kim, and Jez Humble published "Accelerate: Building and Scaling High Performing Technology Organizations".

**4. DevOps Features**

Here are some key features of DevOps architecture, such as:



1) Automation

Automation can reduce time consumption, especially during the testing and deployment phase. The productivity increases, and releases are made quicker by automation. This will lead in catching bugs quickly so that it can be fixed easily. For contiguous delivery, each code is defined through automated tests, cloud-based services, and builds. This promotes production using automated deploys.

2) Collaboration

The Development and Operations team collaborates as a DevOps team, which improves the cultural model as the teams become more productive with their productivity, which strengthens accountability and ownership. The teams share their responsibilities and work closely in sync, which in turn makes the deployment to production faster.

3) Integration

Applications need to be integrated with other components in the environment. The integration phase is where the existing code is combined with new functionality and then tested. Continuous integration and testing enable continuous development. The frequency in the releases and micro-services leads to significant operational challenges. To overcome such problems, continuous integration and delivery are implemented to deliver in a **quicker, safer**, and **reliable manner**.

4) Configuration management

It ensures the application to interact with only those resources that are concerned with the environment in which it runs. The configuration files are not created where the external configuration to the application is separated from the source code. The configuration file can be written during deployment, or they can be loaded at the run time, depending on the environment in which it is running.

DevOps Advantages and Disadvantages

Here are some advantages and disadvantages that DevOps can have for business, such as:

Advantages

* DevOps is an excellent approach for quick development and deployment of applications.
* It responds faster to the market changes to improve business growth.
* DevOps escalate business profit by decreasing software delivery time and transportation costs.
* DevOps clears the descriptive process, which gives clarity on product development and delivery.
* It improves customer experience and satisfaction.
* DevOps simplifies collaboration and places all tools in the cloud for customers to access.
* DevOps means collective responsibility, which leads to better team engagement and productivity.

Disadvantages

* DevOps professional or expert's developers are less available.
* Developing with DevOps is so expensive.
* Adopting new DevOps technology into the industries is hard to manage in short time.
* Lack of DevOps knowledge can be a problem in the continuous integration of automation projects.

**4. DevOps Architecture**

To make successful deliveries of applications, development and operations play an essential role. Deployment means analyzing requirements, design, development, and testing of the software components or frameworks.

In DevOps architecture, both the development and operations collaborate to fix the gap between deployment and operation terms; so that delivery can be faster.

This architecture is mainly used for the applications that are hosted on the cloud platform and also for large distributed applications. These days companies are mainly focusing on Agile Development for the early rollout of deliveries. In the DevOps architecture, agile development is used so that integration and delivery can be contiguous. Development and operations team working in silos is time-consuming to design, test, and deploy. And if the teams are not in sync, then it adds to the delay in the delivery. DevOps brings the development and operations team together and increases productivity.

The various components used in DevOps architecture are depicted below:

1) Build

In absence of DevOps, the cost of the consumption of the resources was evaluated based on the individual usage hardware allocation. With DevOps, the sharing of resources and usage of the cloud is enabled. The build is dependent upon the user's need, hence the usage of resources or capacity can be controlled.

2) Code

The code is properly arranged in files, folders, etc., is maintainable, and can be reused.

3) Test

After testing the application is ready for production. Manual testing consumes more time in testing and moving the code to the output. By automating the testing, the time taken for testing is decreased and the time taken to deploy the application to production is also reduced. With automation, the test case execution is faster and removes the time taken in performing manual steps.

4) Plan

In DevOps, planning for development is done using Agile methodology. Since the operations and development team are in sync, the plan to work is organized and accordingly, it increases productivity.

5) Monitor

Continuous monitoring of the application is required and used to identify any risk of failure. It also helps in tracking the health status of the application. There are many third-party tools used for monitoring such as Splunk.

6) Deploy

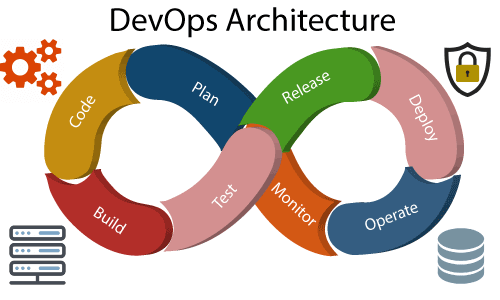
A scheduler for automated deployment is supported by many systems in the current scenario. With the help of a cloud management platform, users are enabled to capture accurate insights and view the optimization scenario, analytics on trends by the deployment of dashboards.

7) Operate

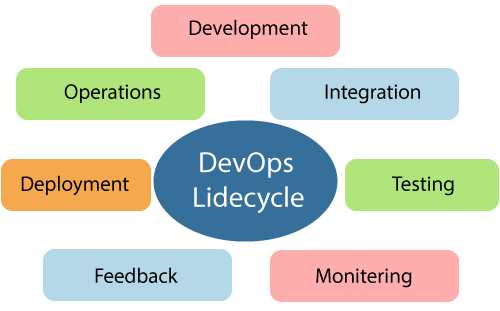
Before DevOps, the traditional approach of developing and testing separately was followed. With DevOps, both the development and operations teams operate collaboratively by actively participating throughout the service lifecycle. The operation team interacts with developers, and together they come up with a monitoring plan which serves the IT and business requirements.

8) Release

Deployment to an environment is done by automation. But deployment to the production environment is done manually. There are many processes involved in release management that are commonly used to do the deployment in the production environment manually so that there is less impact on the customers



**5. DevOps Lifecycle**

DevOps defines an agile relationship between operations and Development. It is a process that is practiced by the development team and operational engineers together from beginning to the final stage of the product.

Learning DevOps is not complete without understanding the DevOps lifecycle phases. The DevOps lifecycle includes seven phases as given below:

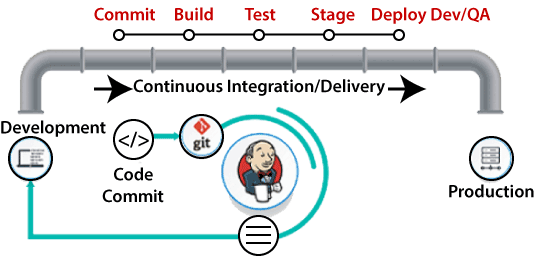
1) Continuous Development

This phase involves the planning and coding of the software. The vision of the project is decided during the planning phase. And the developers begin developing the code for the application. There are no DevOps tools that are required for planning, but there are several tools for maintaining the code.

2) Continuous Integration

This stage is the heart of the entire DevOps lifecycle. It is a software development practice in which the developers require to commit changes to the source code more frequently. This may be on a daily or weekly basis. Then every commit is built, and this allows early detection of problems if they are present. Building code is not only involved compilation, but it also includes **unit testing, integration testing, code review**, and **packaging**.

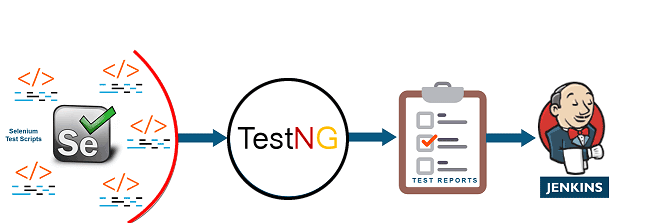
The code supporting new functionality is continuously integrated with the existing code. Therefore, there is continuous development of software. The updated code needs to be integrated continuously and smoothly with the systems to reflect changes to the end-users.



Jenkins is a popular tool used in this phase. Whenever there is a change in the Git repository, then Jenkins fetches the updated code and prepares a build of that code, which is an executable file in the form of war or jar. Then this build is forwarded to the test server or the production server.

3) Continuous Testing

This phase, where the developed software is continuously testing for bugs. For constant testing, automation testing tools such as **TestNG, JUnit, Selenium**, etc are used. These tools allow QAs to test multiple code-bases thoroughly in parallel to ensure that there is no flaw in the functionality. In this phase, **Docker** Containers can be used for simulating the test environment.



**Selenium** does the automation testing, and TestNG generates the reports. This entire testing phase can automate with the help of a Continuous Integration tool called **Jenkins**.

Automation testing saves a lot of time and effort for executing the tests instead of doing this manually. Apart from that, report generation is a big plus. The task of evaluating the test cases that failed in a test suite gets simpler. Also, we can schedule the execution of the test cases at predefined times. After testing, the code is continuously integrated with the existing code.

4) Continuous Monitoring

Monitoring is a phase that involves all the operational factors of the entire DevOps process, where important information about the use of the software is recorded and carefully processed to find out trends and identify problem areas. Usually, the monitoring is integrated within the operational capabilities of the software application.

It may occur in the form of documentation files or maybe produce large-scale data about the application parameters when it is in a continuous use position. The system errors such as server not reachable, low memory, etc are resolved in this phase. It maintains the security and availability of the service.

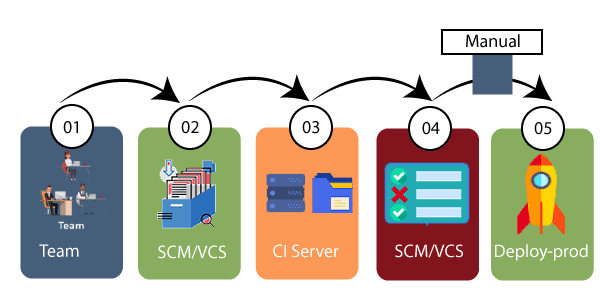
5) Continuous Feedback

The application development is consistently improved by analyzing the results from the operations of the software. This is carried out by placing the critical phase of constant feedback between the operations and the development of the next version of the current software application.

The continuity is the essential factor in the DevOps as it removes the unnecessary steps which are required to take a software application from development, using it to find out its issues and then producing a better version. It kills the efficiency that may be possible with the app and reduce the number of interested customers.

6) Continuous Deployment

In this phase, the code is deployed to the production servers. Also, it is essential to ensure that the code is correctly used on all the servers.



The new code is deployed continuously, and configuration management tools play an essential role in executing tasks frequently and quickly. Here are some popular tools which are used in this phase, such as **Chef, Puppet, Ansible**, and **SaltStack**.

Containerization tools are also playing an essential role in the deployment phase. **Vagrant** and **Docker** are popular tools that are used for this purpose. These tools help to produce consistency across development, staging, testing, and production environment. They also help in scaling up and scaling down instances softly.

Containerization tools help to maintain consistency across the environments where the application is tested, developed, and deployed. There is no chance of errors or failure in the production environment as they package and replicate the same dependencies and packages used in the testing, development, and staging environment. It makes the application easy to run on different computers.

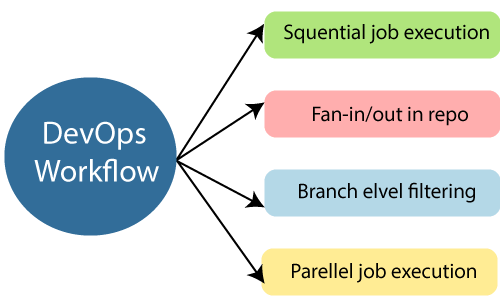
7) Continuous Operations

All DevOps operations are based on the continuity with complete automation of the release process and allow the organization to accelerate the overall time to market continuingly.

It is clear from the discussion that continuity is the critical factor in the DevOps in removing steps that often distract the development, take it longer to detect issues and produce a better version of the product after several months. With DevOps, we can make any software product more efficient and increase the overall count of interested customers in your product.

**6. Understanding Workflow and principles**

DevOps workflow provides a visual overview of the sequence in which input is provided. Also, it tells about which one action is performed, and output is generated for an operations process.



DevOps workflow allows the ability to separate and arrange the jobs which are top requested by the users. Also, it gives the ability to mirror their ideal process in the configuration jobs.

DevOps Principles

The main principles of DevOps are Continuous delivery, automation, and fast reaction to the feedback.

1. **End to End Responsibility:** DevOps team need to provide performance support until they become the end of life. It enhances the responsibility and the quality of the products engineered.
2. **Continuous Improvement:** DevOps culture focuses on continuous improvement to minimize waste. It continuously speeds up the growth of products or services offered.
3. **Automate Everything:** Automation is an essential principle of the DevOps process. This is for software development and also for the entire infrastructure landscape.
4. **Custom Centric Action:** DevOps team must take customer-centric for that they should continuously invest in products and services.
5. **Monitor and test everything:** The DevOps team needs to have robust monitoring and testing procedures.
6. **Work as one team:** In the DevOps culture role of the designers, developers, and testers are already defined. All they needed to do is work as one team with complete collaboration.

These principles are achieved through several DevOps practices, which include frequent deployments, QA automation, continuous delivery, validating ideas as early as possible, and in-team collaboration.

**7. Introduction to DevOps tools**

DevOps tools such as **Git, Ansible, Docker, Jenkins,** and **Kubernetes**.

**Git**

**Git** is an **open-source distributed version control system**. It is designed to handle minor to major projects with high speed and efficiency. It is developed to co-ordinate the work among the developers. The version control allows us to track and work together with our team members at the same workspace.

Git is foundation of many services like **GitHub** and **GitLab**, but we can use Git without using any other Git services. Git can be used **privately** and **publicly**.

Git was created by **Linus Torvalds** in **2005** to develop Linux Kernel. It is also used as an important distributed version-control tool for **the DevOps**.

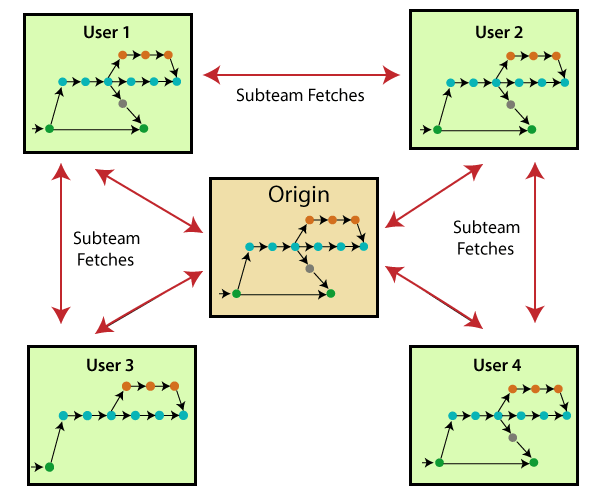
Git is easy to learn, and has fast performance. It is superior to other SCM tools like Subversion, CVS, Perforce, and ClearCase.

Features of Git

Some remarkable features of Git are as follows:



* **Open Source**  
  Git is an **open-source tool**. It is released under the **GPL** (General Public License) license.
* **Scalable**  
  Git is **scalable**, which means when the number of users increases, the Git can easily handle such situations.
* **Distributed**  
  One of Git's great features is that it is **distributed**. Distributed means that instead of switching the project to another machine, we can create a "clone" of the entire repository. Also, instead of just having one central repository that you send changes to, every user has their own repository that contains the entire commit history of the project. We do not need to connect to the remote repository; the change is just stored on our local repository. If necessary, we can push these changes to a remote repository.



* **Security**  
  Git is secure. It uses the **SHA1 (Secure Hash Function)** to name and identify objects within its repository. Files and commits are checked and retrieved by its checksum at the time of checkout. It stores its history in such a way that the ID of particular commits depends upon the complete development history leading up to that commit. Once it is published, one cannot make changes to its old version.
* **Speed**  
  Git is very **fast**, so it can complete all the tasks in a while. Most of the git operations are done on the local repository, so it provides a **huge speed**. Also, a centralized version control system continually communicates with a server somewhere.  
  Performance tests conducted by Mozilla showed that it was **extremely fast compared to other VCSs**. Fetching version history from a locally stored repository is much faster than fetching it from the remote server. The **core part of Git**is **written in C**, which **ignores** runtime overheads associated with other high-level languages.  
  Git was developed to work on the Linux kernel; therefore, it is **capable** enough to **handle large** **repositories** effectively. From the beginning, **speed** and **performance** have been Git's primary goals.
* **Supports non-linear development**  
  Git supports **seamless branching and merging**, which helps in visualizing and navigating a non-linear development. A branch in Git represents a single commit. We can construct the full branch structure with the help of its parental commit.
* **Branching and Merging**  
  **Branching and merging** are the **great feature**s of Git, which makes it different from the other SCM tools. Git allows the **creation of multiple branches** without affecting each other. We can perform tasks like **creation**, **deletion**, and **merging** on branches, and these tasks take a few seconds only. Below are some features that can be achieved by branching:
  + We can **create a separate branch** for a new module of the project, commit and delete it whenever we want.
  + We can have a **production branch**, which always has what goes into production and can be merged for testing in the test branch.
  + We can create a **demo branch** for the experiment and check if it is working. We can also remove it if needed.
  + The core benefit of branching is if we want to push something to a remote repository, we do not have to push all of our branches. We can select a few of our branches, or all of them together.
* **Data Assurance**  
  The Git data model ensures the **cryptographic integrity** of every unit of our project. It provides a **unique commit ID** to every commit through a **SHA algorithm**. We can **retrieve** and **update** the commit by commit ID. Most of the centralized version control systems do not provide such integrity by default.
* **Staging Area**  
  The **Staging area** is also a **unique functionality** of Git. It can be considered as a **preview of our next commit**, moreover, an **intermediate area** where commits can be formatted and reviewed before completion. When you make a commit, Git takes changes that are in the staging area and make them as a new commit. We are allowed to add and remove changes from the staging area. The staging area can be considered as a place where Git stores the changes.  
  Although, Git doesn't have a dedicated staging directory where it can store some objects representing file changes (blobs). Instead of this, it uses a file called index.



Another feature of Git that makes it apart from other SCM tools is that **it is possible to quickly stage some of our files and commit them without committing other modified files in our working directory.**

* **Maintain the clean history**  
  Git facilitates with Git Rebase; It is one of the most helpful features of Git. It fetches the latest commits from the master branch and puts our code on top of that. Thus, it maintains a clean history of the project.

Benefits of Git

A version control application allows us to **keep track** of all the changes that we make in the files of our project. Every time we make changes in files of an existing project, we can push those changes to a repository. Other developers are allowed to pull your changes from the repository and continue to work with the updates that you added to the project files.

Some **significant benefits** of using Git are as follows:



* **Saves Time**  
  Git is lightning fast technology. Each command takes only a few seconds to execute so we can save a lot of time as compared to login to a GitHub account and find out its features.
* **Offline Working**  
  One of the most important benefits of Git is that it supports **offline working**. If we are facing internet connectivity issues, it will not affect our work. In Git, we can do almost everything locally. Comparatively, other CVS like SVN is limited and prefer the connection with the central repository.
* **Undo Mistakes**  
  One additional benefit of Git is we can **Undo** mistakes. Sometimes the undo can be a savior option for us. Git provides the undo option for almost everything.
* **Track the Changes**  
  Git facilitates with some exciting features such as **Diff, Log,** and **Status**, which allows us to track changes so we can **check the status, compare** our files or branches.

Why Git?

We have discussed many **features** and **benefits** of Git that demonstrate the undoubtedly Git as the **leading version control system**. Now, we will discuss some other points about why should we choose Git.



* **Git Integrity**  
  Git is **developed to ensure** the **security** and **integrity** of content being version controlled. It uses checksum during transit or tampering with the file system to confirm that information is not lost. Internally it creates a checksum value from the contents of the file and then verifies it when transmitting or storing data.
* **Trendy Version Control System**  
  Git is the **most widely used version control system**. It has **maximum projects** among all the version control systems. Due to its **amazing workflow** and features, it is a preferred choice of developers.
* **Everything is Local**  
  Almost All operations of Git can be performed locally; this is a significant reason for the use of Git. We will not have to ensure internet connectivity.
* **Collaborate to Public Projects**  
  There are many public projects available on the GitHub. We can collaborate on those projects and show our creativity to the world. Many developers are collaborating on public projects. The collaboration allows us to stand with experienced developers and learn a lot from them; thus, it takes our programming skills to the next level.
* **Impress Recruiters**  
  We can impress recruiters by mentioning the Git and GitHub on our resume. Send your GitHub profile link to the HR of the organization you want to join. Show your skills and influence them through your work. It increases the chances of getting hired.

**2. Ansible**

**Ansible** is simple open source IT engine which automates application deployment, intra service orchestration, cloud provisioning and many other IT tools.

Ansible is easy to deploy because it does not use any agents or custom security infrastructure.

Ansible uses playbook to describe automation jobs, and playbook uses very simple language i.e. **YAML** (It’s a human-readable data serialization language & is commonly used for configuration files, but could be used in many applications where data is being stored)which is very easy for humans to understand, read and write. Hence the advantage is that even the IT infrastructure support guys can read and understand the playbook and debug if needed (YAML – It is in human readable form).

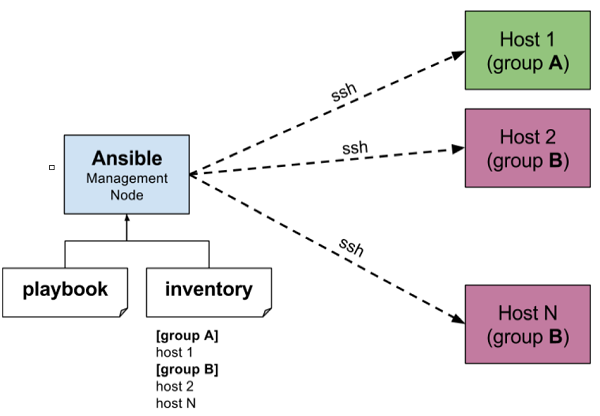
Ansible is designed for multi-tier deployment. Ansible does not manage one system at time, it models IT infrastructure by describing all of your systems are interrelated. Ansible is completely agentless which means Ansible works by connecting your nodes through ssh(by default). But if you want other method for connection like Kerberos, Ansible gives that option to you.

After connecting to your nodes, Ansible pushes small programs called as “Ansible Modules”. Ansible runs that modules on your nodes and removes them when finished. Ansible manages your inventory in simple text files (These are the hosts file). Ansible uses the hosts file where one can group the hosts and can control the actions on a specific group in the playbooks.

How Ansible Works?

The picture given below shows the working of Ansible.

**Ansible works** by connecting to your nodes and pushing out small programs, called "**Ansible** modules" to them. **Ansible** then executes these modules (over SSH by default), and removes them when finished. Your library of modules can reside on any machine, and there are no servers, daemons, or databases required.



The management node in the above picture is the controlling node (managing node) which controls the entire execution of the playbook. It’s the node from which you are running the installation. The inventory file provides the list of hosts where the Ansible modules needs to be run and the management node does a SSH connection and executes the small modules on the hosts machine and installs the product/software.

**Beauty** of Ansible is that it removes the modules once those are installed so effectively it connects to host machine , executes the instructions and if it’s successfully installed removes the code which was copied on the host machine which was executed.

**3. Docker**

**A developer builds an application and sends it to the tester. But, the environments of development and testing systems are different; thus, the code does not work. There are two solutions to this: Docker and Virtual Machines, but Docker is far better in terms of performance, scaling, and efficiency.**

### What is Docker?

Docker is an open-source project with a friendly-whale logo that facilitates the deployment of applications in software containers. It is a set of PaaS products that deliver containers (software packages) using OS-level virtualization. It embodies resource isolation features of the Linux kernel but offers a friendly API.

In simple words, Docker is a tool or platform design to simplify the process of creating, deploying, and packaging and shipping out applications along with its parts such as libraries and other dependencies. Its primary purpose is to automate the application deployment process and operating-system-level virtualization on Linux. It allows multiple containers to run on the same hardware and provides high productivity, along with maintaining isolated applications and facilitating seamless configuration.

#### Docker benefits include:

* High ROI and cost savings
* Productivity and standardization
* Maintenance and compatibility
* Rapid deployment
* Faster configurations
* Seamless portability
* Continuous testing and deployment
* Isolation, segregation, and security

#### Docker vs. Virtual Machines

Virtual Machine is an application environment that imitates dedicated hardware by providing an emulation of the computer system. Docker and Vmboth have their set of benefits and uses, but when it comes to running applications in multiple environments, both can be utilized. So which one wins? Let's get into a quick Docker vs. VM comparison.

OS Support: VM requires a lot of memory when installed in an OS, whereas Docker containers occupy less space.

Performance: Running several VMs can affect the performance, whereas, Docker containers are stored in a single Docker engine; thus, they provide better performance.

Boot-up time: VMs have a longer booting time compared to Docker.  
Efficiency: VMs have lower efficiency than Docker.

Scaling: VMs are difficult to scale up, whereas Docker is easy to scale up.

Space allocation: You cannot share data volumes with VMs, but you can share and reuse them among various Docker containers.

Portability: With VMs, you can face compatibility issues while porting across different platforms; Docker is easily portable.  
Clearly, Docker is a hands-down winner.

**4. Jenkins**

Jenkins Tutorial is designed for both beginners and professionals. Our Tutorial provides all the basic and advanced concepts of Jenkins, such as Jenkins installation, Jenkins Configuration, Jenkins Pipeline, etc.

Jenkins is an open source automation tool written in Java programming language that allows continuous integration.

Jenkins builds and tests our software projects, which continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build.

## What is Jenkins?

Jenkins is an open source automation tool written in Java programming language that allows continuous integration.

Jenkins **builds** and **tests** our software projects which continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build.

It also allows us to continuously **deliver** our software by integrating with a large number of testing and deployment technologies.

Jenkins offers a straightforward way to set up a continuous integration or continuous delivery environment for almost any combination of languages and source code repositories using pipelines, as well as automating other routine development tasks.

With the help of Jenkins, organizations can speed up the software development process through automation. Jenkins adds development life-cycle processes of all kinds, including build, document, test, package, stage, deploy static analysis and much more.

Jenkins achieves CI (Continuous Integration) with the help of plugins. Plugins is used to allow the integration of various DevOps stages. If you want to integrate a particular tool, you have to install the plugins for that tool. For example: Maven 2 Project, Git, HTML Publisher, Amazon EC2, etc.

**For example:** If any organization is developing a project, then **Jenkins** will continuously test your project builds and show you the errors in early stages of your development.

Possible steps executed by Jenkins are for example:

* Perform a software build using a build system like Gradle or Maven Apache
* Execute a shell script
* Archive a build result
* Running software tests

**Work Flow:**



## History of Jenkins

Kohsuke Kawaguchi, who is a Java developer, working at SUN Microsystems, was tired of building the code and fixing errors repetitively. In 2004, he created an automation server called **Hudson** that automates build and test task.

In 2011, Oracle who owned Sun Microsystems had a dispute with Hudson open source community, so they forked Hudson and renamed it as **Jenkins**.

Both Hudson and Jenkins continued to operate independently. But in short span of time, Jenkins acquired a lot of contributors and projects while Hudson remained with only 32 projects. Then with time, Jenkins became more popular, and Hudson is not maintained anymore.

## What is Continuous Integration?

Continuous Integration (CI) is a development practice in which the developers are needs to commit changes to the source code in a shared repository at regular intervals. Every commit made in the repository is then built. This allows the development teams to detect the problems early.

Continuous integration requires the developers to have regular builds. The general practice is that whenever a code commit occurs, a build should be triggered.

## Continuous Integration with Jenkins

Let's consider a scenario where the complete source code of the application was built and then deployed on test server for testing. It sounds like a perfect way to develop software, but this process has many problems.

* Developer teams have to wait till the complete software is developed for the test results.
* There is a high prospect that the test results might show multiple bugs. It was tough for developers to locate those bugs because they have to check the entire source code of the application.
* It slows the software delivery process.
* Continuous feedback pertaining to things like architectural or coding issues, build failures, test status and file release uploads was missing due to which the quality of software can go down.
* The whole process was manual which increases the threat of frequent failure.

It is obvious from the above stated problems that not only the software delivery process became slow but the quality of software also went down. This leads to customer dissatisfaction.

So to overcome such problem there was a need for a system to exist where developers can continuously trigger a build and test for every change made in the source code.

This is what Continuous Integration (CI) is all about. Jenkins is the most mature Continuous Integration tool available so let us see how Continuous Integration with Jenkins overcame the above shortcomings.

Let's see a generic flow diagram of Continuous Integration with Jenkins:



**Let's see how Jenkins works**. The above diagram is representing the following functions:

* First of all, a developer commits the code to the source code repository. Meanwhile, the Jenkins checks the repository at regular intervals for changes.
* Soon after a commit occurs, the Jenkins server finds the changes that have occurred in the source code repository. Jenkins will draw those changes and will start preparing a new build.
* If the build fails, then the concerned team will be notified.
* If built is successful, then Jenkins server deploys the built in the test server.
* After testing, Jenkins server generates a feedback and then notifies the developers about the build and test results.
* It will continue to verify the source code repository for changes made in the source code and the whole process keeps on repeating.

## Advantages and Disadvantages of using Jenkins

**Advantages of Jenkins**

* It is an open source tool.
* It is free of cost.
* It does not require additional installations or components. Means it is easy to install.
* Easily configurable.
* It supports 1000 or more plugins to ease your work. If a plugin does not exist, you can write the script for it and share with community.
* It is built in java and hence it is portable.
* It is platform independent. It is available for all platforms and different operating systems. Like OS X, Windows or Linux.
* Easy support, since it open source and widely used.
* Jenkins also supports cloud based architecture so that we can deploy Jenkins in cloud based platforms.

**Disadvantages of Jenkins**

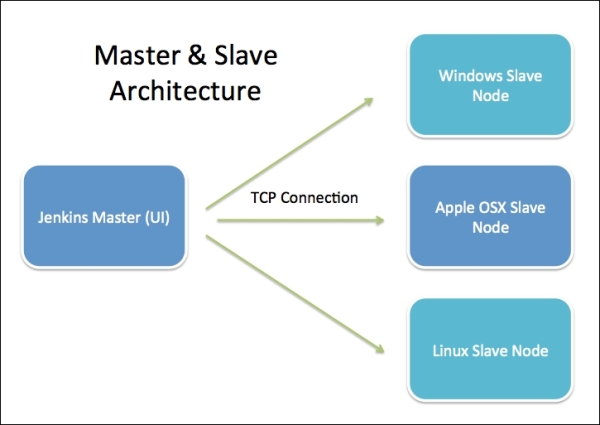
* Its interface is out dated and not user friendly compared to current user interface trends.
* Not easy to maintain it because it runs on a server and requires some skills as server administrator to monitor its activity.
* CI regularly breaks due to some small setting changes. CI will be paused and therefore requires some developer's team attention.

## Jenkins Architecture

Jenkins follows Master-Slave architecture to manage distributed builds. In this architecture, slave and master communicate through TCP/IP protocol.

Jenkins architecture has two components:

* Jenkins Master/Server
* Jenkins Slave/Node/Build Server



### Jenkins Master

The main server of Jenkins is the Jenkins Master. It is a web dashboard which is nothing but powered from a war file. By default it runs on 8080 port. With the help of Dashboard, we can configure the jobs/projects but the build takes place in Nodes/Slave. By default one node (slave) is configured and running in Jenkins server. We can add more nodes using IP address, user name and password using the ssh, jnlp or webstart methods.

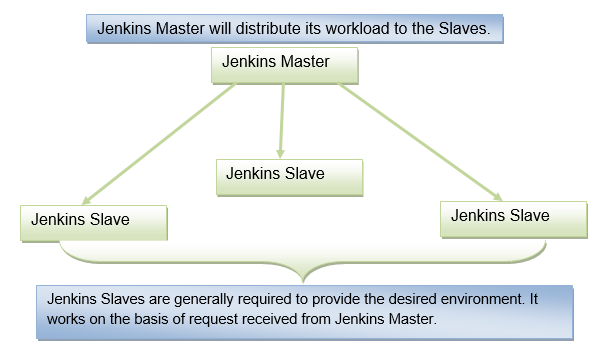
The server's job or master's job is to handle:

* Scheduling build jobs.
* Dispatching builds to the nodes/slaves for the actual execution.
* Monitor the nodes/slaves (possibly taking them online and offline as required).
* Recording and presenting the build results.
* A Master/Server instance of Jenkins can also execute build jobs directly.

### Jenkins Slave

Jenkins slave is used to execute the build jobs dispatched by the master. We can configure a project to always run on a particular slave machine, or particular type of slave machine, or simple let the Jenkins to pick the next available slave/node.

As we know Jenkins is developed using Java is platform independent thus Jenkins Master/Servers and Slave/nodes can be configured in any servers including Linux, Windows, and Mac.

 **5. Kubernetes**.

Kubernetes is a production-grade, open-source platform that orchestrates the placement (scheduling) and execution of application containers within and across computer clusters

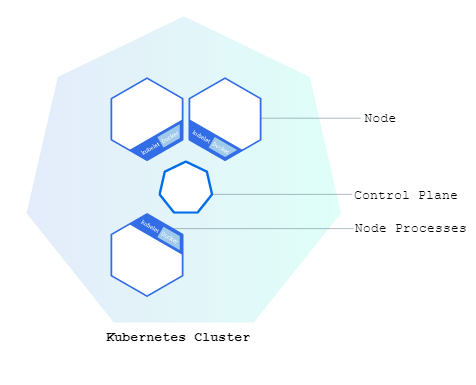
Kubernetes coordinates a highly available cluster of computers that are connected to work as a single unit. The abstractions in Kubernetes allow you to deploy containerized applications to a cluster without tying them specifically to individual machines. To make use of this new model of deployment, applications need to be packaged in a way that decouples them from individual hosts: they need to be containerized. Containerized applications are more flexible and available than in past deployment models, where applications were installed directly onto specific machines as packages deeply integrated into the host. Kubernetes automates the distribution and scheduling of application containers across a cluster in a more efficient way. Kubernetes is an open-source platform and is production-ready.

A Kubernetes cluster consists of two types of resources:

The Control Plane coordinates the cluster

Nodes are the workers that run applications

**Cluster Diagram**



The Control Plane is responsible for managing the cluster. The Control Plane coordinates all activities in your cluster, such as scheduling applications, maintaining applications' desired state, scaling applications, and rolling out new updates.

A node is a VM or a physical computer that serves as a worker machine in a Kubernetes cluster. Each node has a Kubelet, which is an agent for managing the node and communicating with the Kubernetes control plane. The node should also have tools for handling container operations, such as containerd or Docker. A Kubernetes cluster that handles production traffic should have a minimum of three nodes because if one node goes down, both an etcd member and a control plane instance are lost, and redundancy is compromised. You can mitigate this risk by adding more control plane nodes.

Control Planes manage the cluster and the nodes that are used to host the running applications.

When you deploy applications on Kubernetes, you tell the control plane to start the application containers. The control plane schedules the containers to run on the cluster's nodes. The nodes communicate with the control plane using the Kubernetes API, which the control plane exposes. End users can also use the Kubernetes API directly to interact with the cluster.

A Kubernetes cluster can be deployed on either physical or virtual machines. To get started with Kubernetes development, you can use Minikube. Minikube is a lightweight Kubernetes implementation that creates a VM on your local machine and deploys a simple cluster containing only one node. Minikube is available for Linux, macOS, and Windows systems. The Minikube CLI provides basic bootstrapping operations for working with your cluster, including start, stop, status, and delete. For this tutorial, however, you'll use a provided online terminal with Minikube pre-installed.

**7.Build Automation**

**The build** is a process of preparing source code for production so that users can happily interact with it. It may include compilation, file compression, updating database schema, creating an installer or executable file, etc. That said,

**Build automation** is an approach to handling builds within a CI/CD pipeline that has several steps. A developer commits source code to a repository, a CI server detects the change, runs the build process outside of the developer’s IDE (on a dedicated cloud or in-house machine), checks it with unit tests, and either returns for fixes or sends it further down the pipeline.

Automation helps standardize builds, make them faster, and avoid awkward mistakes typical for humans doing things manually.

## Build automation process

As mentioned before, build automation is a part of a larger CI/CD pipeline in DevOps.

**0. A developer commits code to a repository.**The commit term comes from the realm of version control systems (VCS), services that help manage updates to the software. These may be GitHub, GitLab, Bitbucket, etc. A code commit operation means that an engineer sends new code to a repository residing inside one of those VCSs.

**1. CI server detects changes inside the repository.**A CI server is a system that orchestrates a CI pipeline, allowing teams to commit code multiple times per day, and build automatically outside of their workstations on different target servers. So, a CI server regularly polls a repository to check for new commits and save changes in a separate database.

CI server software allows engineers to tweak build configuration, add build agents that execute build jobs on dedicated machines, and integrate with build tools, which are specific for each programming language. What do these terms mean? Let’s keep exploring the process.

**2.** **Build configuration recognizes changes and triggers a build agent.**A build configuration is a set of rules determening how the build is supposed to happen. For instance, it sets the timing when the build begins, e.g.

* Once a new code commit arrives
* On an established schedule, say, the team knows that by 5 pm every day they must commit their code, or
* After some external process is finished, like a security scan.

Here you can also define exactly which operations and which tools you need to complete the build, adjust steps, or configure your VCS settings.

Build configuration sees changes in a CI server database and, based on settings, initiates different build agents.

**3. Build agent activates build tools.**A *build agent*is a program deployed on a build server (normally, outside of the CI server) that takes orders from the CI server and starts a build. It’s important that a single commit can trigger several parallel agents on different target machines, say, for different operating systems. Depending on a specific CI server, concurrent build agents’ work may be slightly different.

The agent doesn’t compile code, compress files, generate installers, or do any kind of low-level dirty work. It just listens to what a server with its configuration has to say and orders *build tools* to do their job in the way a developer configured. There may be a sequence of operations on the code that amounts to a build. The agent would be triggering those operations in a set order.

**4. Build tools compile, compress, run tests, etc.**Finally, *build tools*do their low-level work. A build tool is a script, framework, or any kind of software created specifically for a given programming language to compile code, run tests on it, or do other operations needed for a build. You’d expect your CI server to support integrations with such tools.

For example, [Maven](https://maven.apache.org/) is popular for [Java](https://www.altexsoft.com/blog/engineering/pros-and-cons-of-java-programming/) compilation and unit testing, you’d use [MSBuild](https://docs.microsoft.com/en-us/visualstudio/msbuild/msbuild?view=vs-2022) with [C#](https://www.altexsoft.com/blog/c-sharp-pros-and-cons/) and C++, to run tasks on [JavaScript](https://www.altexsoft.com/blog/engineering/mean-mern-javascript-full-stack/) code you have [Grunt](https://gruntjs.com/), you get the idea.

At this stage, you’ll likely do *unit tests*, the lowest level quality assurance checks that verify that this particular unit or component works as intended. Some build tools already have unit testing support. Sometimes, an agent can initiate several build tools in a sequence to compile code with one tool and test it with another.

**5. Build artifacts are sent back to the server.**If the code passes all build stages including testing, you’d normally send it back to the CI server, at which point it can go to production, further testing, or other storage, depending on how your CI/CD pipeline works.

**8.Understanding Code Quality**

Code quality is a term that denotes good quality or poor quality code. The quality of code is still a subjective concept, as what is high quality for a web application developer may not be so for an automotive developer. That being said, some common factors determine the quality of the code.

 At the most basic level, good code:

* Does what it should
* Follows a consistent style
* Is easy to understand
* Has been well-documented
* Can be tested flawlessly

As you can gauge, producing quality code has several advantages for software development. Professional software developers consistently focus on writing quality code as it offers a higher ROI on their efforts in the long run.

Below, we have identified some reasons highlighting the importance of quality code in software development:

* Clean and consistent code is easier to understand and maintain. It can also be extended or re-used with minimal effort
* Quality code is highly readable. It is also well-documented, which makes it simpler to review and reduces the development effort
* Well-designed software achieves lower code complexity. It is more robust, less prone to bugs, and also easier to test
* High code quality reduces technical debt, which refers to the implied cost of additional work due to the usage of a limited solution instead of a better approach.

**What are the Coding Standards?**

Coding standards can be defined as a collection of coding guidelines, rules and best practices that help you write cleaner code.

Coding standards also promote sound programming practices and improve efficiency, while ensuring your software is:

* Safe and secure so that it cannot be hacked, and used without causing harm
* Reliable, easily maintainable and works as it should, even as the codebase grows
* Testable, even at the code level
* Portable or compatible with various implementation environments and produces consistent results

Whether you are a beginner or an experienced software developer, always follow the right coding standards and industry best practices to write clean, reliable, and secure code. This not only avoids future rework as your codebase grows but also accelerates the time to market and performance of the software.

By following coding standards, you also ensure compliance with industry standards like ISO and IEC automatically, apart from enhancing the security, consistency, and reliability of your code.

Some of the general coding standards are limiting the use of globals, using standard headers for different modules, and avoiding the use of identifiers for multiple purposes. These standards also guide software developers to document their code properly, use a simple coding style, use proper indentation and follow the standard naming conventions for variables, constants, and functions.

**Measuring Code Quality**

Five key factors are used to measure code quality are:

* **Reliability** – This measures whether the system will run over a specific period. Low defect count is the hallmark of a reliable codebase.
* **Maintainability** – This measures if the software can be easily maintained, which depends on the size, structure, and complexity of the codebase.
* **Testability** – Quality code is crucial for [developing quality software](https://talent500.co/blog/history-of-software-engineering/)that supports testing at the code level. Testability can be measured by the number of test cases required to find potential faults in the system.
* **Portability** – By testing the code on different platforms, you can determine platform independency or its performance in different environments. Quality code is portable and performs consistently in different environments.
* **Reusability** – Lesser interdependencies imply the code is re-usable, which is a prized characteristic to speed future development.

While these factors determine the quality of code, it is also important to quantify these characteristics or use a base metric to test quality. The most popular way to check is by using defect metrics that use the number and severity of defects to indicate the overall code quality.

**Tips on Improving Code Quality**

**1. Follow Coding Standards**

As we touched upon previously, following coding standards is imperative to writing quality code that is consistent with industry standards. To ensure this, we suggest that you start by listing the coding conventions and picking the ones that the whole team is going to follow throughout to make the process consistent and efficient.

**2. Write Legible Code**

Quality code is also legible and easily understandable code. To make sure your code is readable, go through it impartially to determine whether you can understand it one reading or not.

If not, you need to simplify it. Of course, if you follow the industry best practices while writing code, it is bound to be legible and easy to read for most software developers. You can simplify the code further by using well-named functions, classes, and variables, and grouping common code into separate files for better classification of code.

**3. Continuous Testing**

The fewer the bugs the higher is the quality of your code. This points to the importance of proper testing to ensure the software works as it is intended to, which is the benchmark of high-quality code.

We suggest having an appropriate test strategy in place to test the expected use cases. For example, test-driven development is an excellent practice to follow as it forces you to write down the tests before you proceed with writing the code. This gives you a clearer idea of what you need to feed to your code and what it will churn out – which also speeds up the development process.

It is also essential to use different tiers of testing, including the largest number of unit tests followed by a lesser number of integration and end-to-end tests.

**4. Code Review**

Code review is the test of fire that will tell you how easily or well your code is understood by others. Code review requires that at least two pairs of eyes, including the author of the code for reviewing. Various things can be checked during the review, such as whether the code breaks any convention rules, maintainability and scope of the code, and error handling.

Most teams conduct reviews within themselves, but if you don’t work with a team, you can arrange with a colleague or developer friend to review each other’s codes as needed.

**5. Make Use of a Linter While Developing**

Code linter reads your code and send you warnings if your code is not compliant with the standards of a coding language to prevent issues in the future. By using a linter for the language or the framework you are using, you can effortlessly enforce the accepted set of standards for the language in use, ironing out a lot of creases that may eat up your time in the later stages of development.

**6. Use Comments Judiciously**

Writing too many or too few comments will not give you any benefit. Rather, you should only add comments when they provide value. For example, it is good practice to write a short comment at the top of a file to introduce the scope of the code within. Similarly, a brief comment at the beginning of every class may be added to introduce its primary goals and scope. For function-level comments, we suggest being judicious and only add comments when the function is complicated or lengthy to save time.

In an increasingly digital world, software developers are in high demand, and it’s imperative to acquaint yourself with best coding practices to give yourself and your employer a competitive edge in the market.

**10.Automation of CI/CD.**

CI/CD is a method to frequently deliver apps to customers by introducing automation into the stages of app development. The main concepts attributed to CI/CD are continuous integration, continuous delivery, and continuous deployment. CI/CD is a solution to the problems integrating new code can cause for development and operations teams (AKA "integration hell").

Specifically, CI/CD introduces ongoing automation and continuous monitoring throughout the lifecycle of apps, from integration and testing phases to delivery and deployment. Taken together, these connected practices are often referred to as a "CI/CD pipeline" and are supported by development and operations teams working together in an agile way with either a DevOps or site reliability engineering (SRE) approach

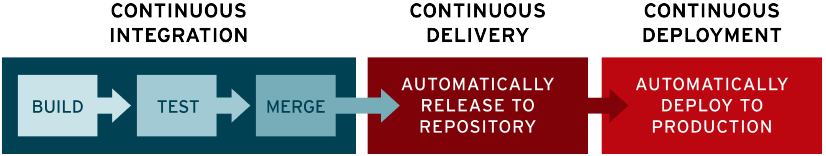
**What's the difference between CI and CD (and the other CD)?**

The acronym CI/CD has a few different meanings. The "CI" in CI/CD always refers to continuous integration, which is an automation process for developers. Successful CI means new code changes to an app are regularly built, tested, and merged to a shared repository. It’s a solution to the problem of having too many branches of an app in development at once that might conflict with each other.

The "CD" in CI/CD refers to continuous delivery and/or continuous deployment, which are related concepts that sometimes get used interchangeably. Both are about automating further stages of the pipeline, but they’re sometimes used separately to illustrate just how much automation is happening.

Continuous delivery usually means a developer’s changes to an application are automatically bug tested and uploaded to a repository (like GitHub or a container registry), where they can then be deployed to a live production environment by the operations team. It’s an answer to the problem of poor visibility and communication between dev and business teams. To that end, the purpose of continuous delivery is to ensure that it takes minimal effort to deploy new code.

Continuous deployment (the other possible "CD") can refer to automatically releasing a developer’s changes from the repository to production, where it is usable by customers. It addresses the problem of overloading operations teams with manual processes that slow down app delivery. It builds on the benefits of continuous delivery by automating the next stage in the pipeline.

[](https://www.redhat.com/rhdc/managed-files/ci-cd-flow-desktop.png?cicd=32h281b)

It’s possible for CI/CD to specify just the connected practices of continuous integration and continuous delivery, or it can also mean all 3 connected practices of continuous integration, continuous delivery, and continuous deployment. To make it more complicated, sometimes "continuous delivery" is used in a way that encompasses the processes of continuous deployment as well.

In the end, it’s probably not worth your time to get bogged down in these semantics—just remember that CI/CD is really a process, often visualized as a pipeline, that involves adding a high degree of ongoing automation and continuous monitoring to app development.

Case-by-case, what the terms refer to depends on how much automation has been built into the CI/CD pipeline. Many enterprises start by adding CI, and then work their way towards automating delivery and deployment down the road, for instance as part of cloud-native apps.