DEVOPS

Course	B.Tech.-VI-Sem.	L	T	P	C
Course Code	22CSPC63	3	-	-	3

Course Outcomes (COs) & CO-PO Mapping (3-Strong; 2-Medium; 1-Weak Correlation)

COs	Upon completion of course the students will be able to	PO2	PO3	PO6	PO12	PSO1
CO1	explain DevOps fundamentals and version control concepts	3	3	3	3	3
CO2	summarize DevOps architecture	3	3	3	3	3
CO3	articulate source code control in system building	3	2	3	3	3
CO4	develop containerization and orchestration	3	3	3	3	3
CO5	plan monitoring, operations and testing for DevOps	3	2	3	3	3

Syllabus

Unit	Title/Topics	Hours
I	**Introduction to DevOps and Version Control**	8
Fundamentals of DevOps: What is DevOps?, DevOps principles, DevOps lifecycle, DevOps delivery pipeline, DevOps ecosystem and tools. The Agile wheel of wheels. **Version Control with Git:** Introduction to version control, Basics of Git, Installing and configuring Git, Common Git commands, Branching and merging strategies, Working with remote repositories.		
II	**DevOps influence on Architecture**	7
Introducing software architecture, The monolithic scenario, Architecture rules of thumb, The separation of concerns, Handling database migrations, Micro-services, and the data tier, DevOps, architecture, and resilience.		
III	**DevOps Integration with Jenkins and Ansible**	6+6=12
Part-A: Continuous Integration with Jenkins: Introduction to Continuous Integration (CI), using Maven for build, Jenkins architecture and setup, Managing Jenkins plugins and nodes, Building and deploying applications using Jenkins, Creating and managing Jenkins pipelines, Pipeline as code with Jenkins.		
Part-B: Configuration Management with Ansible: Introduction to Ansible, Ansible architecture and installation, Inventory management, Ad-hoc commands and playbooks, Roles and modules in Ansible, Writing and managing Ansible playbooks, Integrating Ansible with other tools.		
IV	**Containerization with Docker**	8
Containerization with Docker: Introduction to Docker, Docker installation and setup, Working with Docker images and containers Dockerfile and image creation, Docker compose and Docker Swarm. **Orchestration with Kubernetes:** Introduction to Kubernetes, Kubernetes architecture and components, Setting up a Kubernetes cluster, Managing pods, deployments, and services.		
V	**Monitoring, Operations and Testing Tools**	7
Prometheus: Introduction, Why learn Prometheus, Infrastructure Monitoring, Alerting and Alert Receivers. Grafana: Introduction, Creating Grafana Dashboards, Grafana API and Auto Healing Testing: Selenium features, Testing backend integration points, Test-driven Development, REPL-driven Development.		
Textbooks		
1. Joakim Verona. Practical DevOps, Ingram short title, 2nd Edition, 2018, ISBN-10: 1788392574.		
References		
1. Len Bass, Ingo Weber, Liming Zhu. DevOps: A Software Architect's Perspective. Addison Wesley; ISBN-10. 2. Deepak Gaikwad, Viral Thakkar. DevOps Tools from Practitioner's Viewpoint. Wiley publications. ISBN: 9788126579952.		

Fundamentals of Devops:-

1. **What is Devops?**
2. **Devops Principles**
3. **Devops Lifecycle**
4. **Devops delivery pipeline**
5. **Devops Ecosystem and Tools**
6. **The Agile wheel of wheels.**

1. 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**. **In 2009, the first conference named DevOpsdays was held in Ghent Belgium. Belgian consultant and Patrick Debois founded the conference.**

DevOps helps you to reduce the disconnection between software developers, quality assurance (QA) engineers, and system administrators.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.

Fig.1.Devops

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".

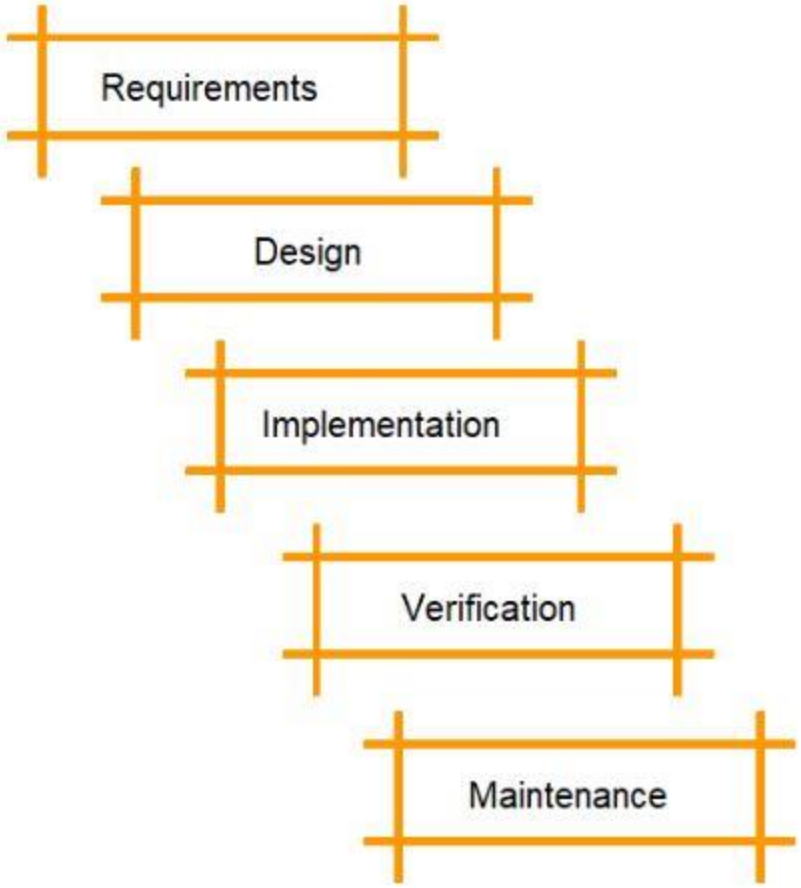
Why DevOps?

Before we get deep into what DevOps is and all the revolutions it brought with us, first understand why DevOps in the first place. and Before DevOps, there were two development models: Waterfall and Agile Method.

1. Waterfall Model

The waterfall model is the first model to be introduced in software development. It is a sequential process and very easy to understand. In this approach, software development is divided into several phases, and the output of one phase becomes the input for the next phase. This model is similar to a waterfall when the water flows off from the cliff; it cannot go back to its previous state.

The phases are; Requirements, Design, Implementation, Verification, and Maintenance.



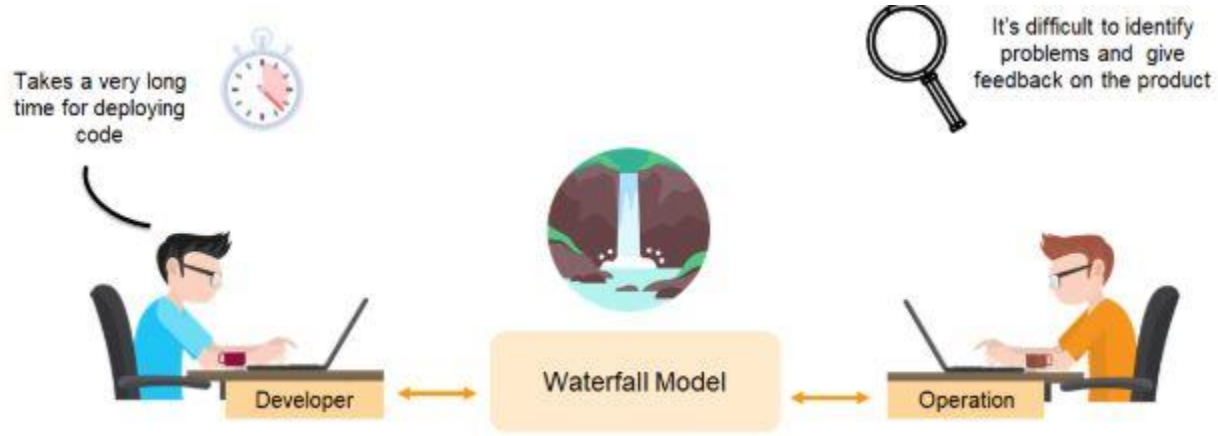
Drawbacks of the waterfall model:

- It’s difficult to make changes to the previous stage
- Not recommended for large-sized projects
- Developers and testers don’t work together (which can result in a lot of bugs at the end)
- Not recommended for projects that will likely have changing requirements

From the figure below, we can see the issues with the waterfall model:

- The developer took a very long time to deploy code

- On the operations side, the tester found it challenging to identify problems and give useful feedback

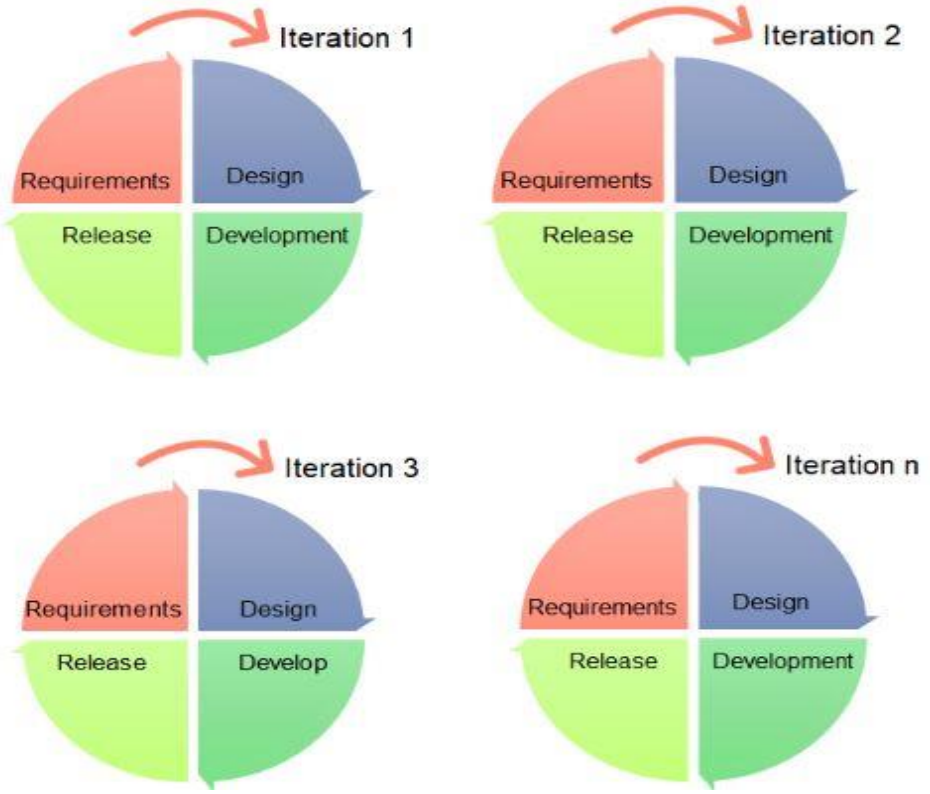


2. Agile Model

Agile is an approach in software development where each project splits into multiple iterations. As a result, at the end of each iteration, a software product is delivered. Each iteration lasts about one to three weeks. Every iteration involves functional teams working simultaneously on various areas, such as:

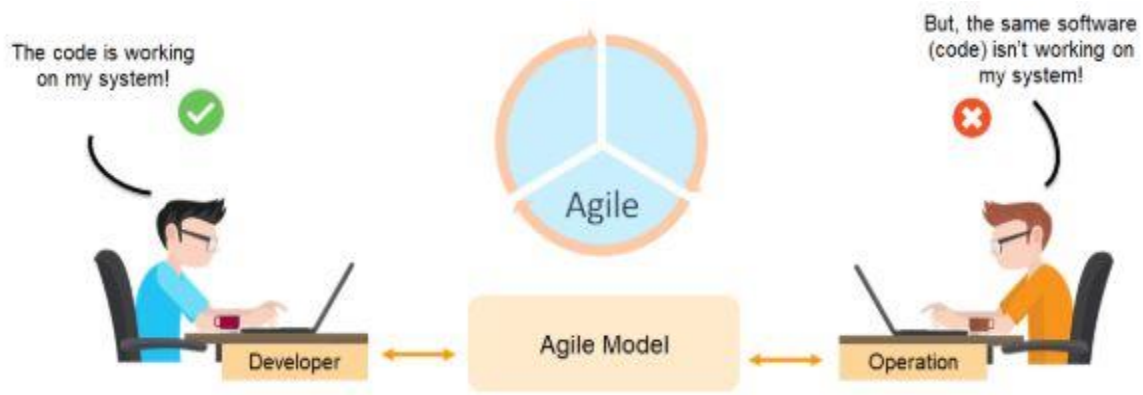
- Requirements
- Design
- Development
- Release

The figure below indicates that there can be a number of iterations needed to deliver a final product in agile method.



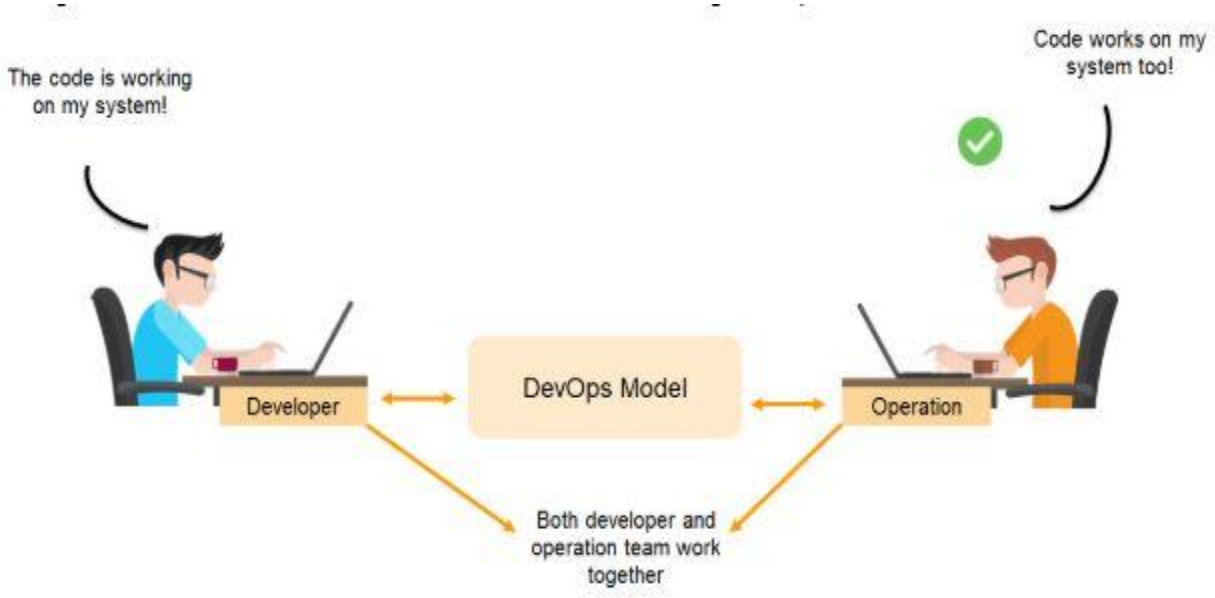
Using the agile method, the code that works for the developer may not work for the operations team.

So how can this issue be solved?



With DevOps, there is continuous integration between deployment of code and the testing of it. Near real-time monitoring and immediate feedback through a DevOps continuous monitoring tool enables both the developer and operations team work together.

The figure below shows how well the software is handled using DevOps.



Some more points for need devops:-

- 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.

2. Devops Principles

1. Collaboration and Communication

DevOps emphasizes collaboration and open communication between development, operations, and other involved teams. By removing silos and fostering a culture where teams work together toward shared goals, DevOps encourages regular interaction, transparency, and joint ownership of the entire software development lifecycle, improving efficiency and resolving issues faster.

2. Automation

Automation is a core principle in DevOps, focusing on reducing manual tasks to increase speed and consistency. Continuous integration (CI) and continuous delivery (CD) automate the process of building, testing, and deploying software. Infrastructure automation using Infrastructure-as-Code (IaC) ensures that infrastructure is provisioned and managed consistently, eliminating human error and reducing time spent on repetitive tasks.

3. Continuous Improvement

DevOps promotes continuous improvement by iterating on processes, tools, and products. Teams regularly review and refine their workflows, incorporating feedback from various stakeholders. This iterative approach leads to faster adaptation to changes, encourages experimentation, and allows organizations to learn from failures, constantly evolving to meet new challenges.

4. Consistency and Standardization

In DevOps, maintaining consistency and standardization across environments is crucial to reduce errors and improve efficiency. By using tools like containers and automated configuration management, teams ensure that code behaves the same way in development, testing, and production. This consistency helps eliminate deployment issues like "it works on my machine" and reduces the risk of environmental discrepancies.

5. Lean and Agile Practices

DevOps integrates lean and agile practices to streamline development and delivery. Lean thinking focuses on minimizing waste and optimizing the flow of work, while agile methodologies encourage delivering small, incremental updates. This combination helps teams quickly respond to customer feedback, adapt to changes, and continuously improve product quality.

6. Continuous Testing

Continuous testing is integral to DevOps, ensuring that code changes are automatically validated early and throughout the development process. By automating testing at multiple levels (unit, integration, system), DevOps encourages early detection of bugs, reduces the cost of fixing defects, and ensures that software is always in a deployable state.

7. Monitoring and Logging

Monitoring and logging are critical components of the DevOps process, providing real-time insights into application performance and system health. Continuous monitoring helps teams detect issues proactively,

while centralized logging allows for quick diagnosis of problems across different systems. This visibility helps improve performance, reliability, and user experience.

8. Security (DevSecOps)

DevSecOps integrates security into the DevOps pipeline, making it an integral part of the development process rather than an afterthought. Automation of security checks, vulnerability assessments, and regular code scanning ensures that software is secure at every stage. This approach shifts security "left" to identify and address vulnerabilities early, making security a shared responsibility among development, operations, and security teams.

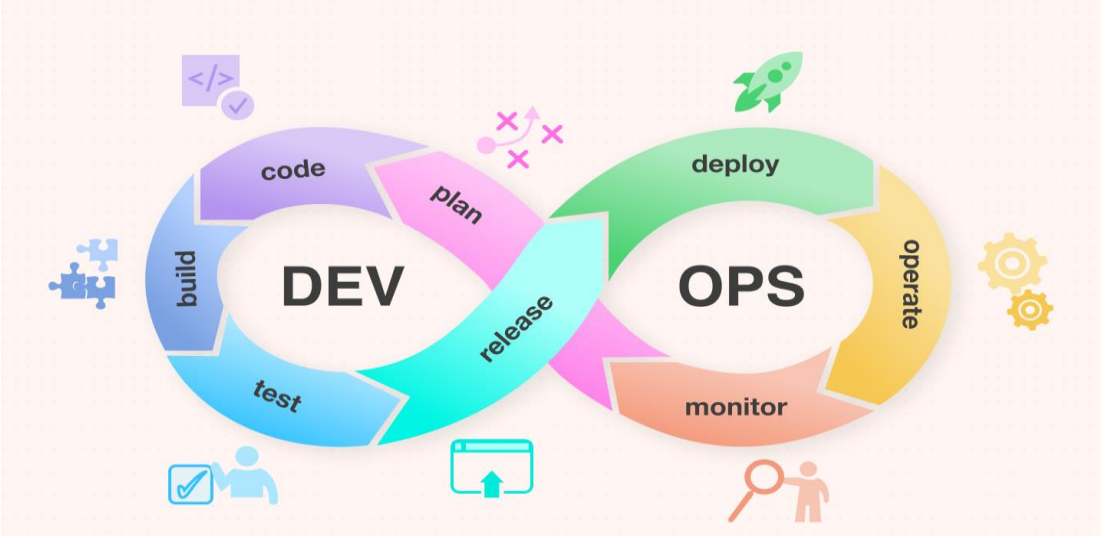
9. Culture of Empowerment

DevOps fosters a culture where teams are empowered to take ownership of both their code and the infrastructure that runs it. Teams are encouraged to collaborate, innovate, and make decisions autonomously, improving job satisfaction and accountability. This culture of empowerment leads to higher engagement, faster problem resolution, and better quality outcomes.

10. Infrastructure as Code (IaC)

Infrastructure as Code (IaC) is a key DevOps practice where infrastructure is managed and provisioned using code instead of manual processes. IaC enables teams to automate the setup of servers, networks, and other resources, ensuring consistency across environments. It allows for easy scaling, quick recovery from failures, and more efficient management of infrastructure, aligning with DevOps' goal of reducing manual intervention.

3. Devops Lifecycle



The **DevOps lifecycle** is a continuous process that involves several stages to automate and streamline the development, testing, deployment, and monitoring of software. The lifecycle focuses on collaboration, automation, and constant feedback to improve software delivery speed, quality, and reliability. Here are the main stages of the DevOps lifecycle:

1. Planning

In this stage, development teams and operations teams collaborate to plan the features, tasks, and overall roadmap for the software. Agile methodologies are often used to break down work into manageable sprints, and both teams ensure they align on requirements, timelines, and priorities. Continuous feedback from stakeholders helps to refine the planning process and adjust goals as needed.

2. Development

During the development stage, developers write the code, implement new features, and fix bugs. DevOps emphasizes the use of version control systems like Git to track code changes and ensure collaboration across teams. Developers use continuous integration (CI) practices to merge their code changes regularly, allowing for quicker detection of issues and improving the overall quality of the codebase.

3. Building

After code development, the building stage involves compiling the code into executable files or artifacts. Automation tools are used to streamline the build process, ensuring that the code can be compiled in a consistent manner every time. This is usually integrated with CI pipelines to run automated tests during the build process and catch any issues early, preventing faulty code from being deployed.

4. Testing

The testing phase is crucial for ensuring that the software works as expected and meets quality standards. DevOps encourages continuous testing by automating unit tests, integration tests, and acceptance tests. Automated tests are run as part of the CI pipeline to identify issues early in the development process. This reduces manual intervention and accelerates the testing process, ensuring that the code is of high quality before deployment.

5. Release

After testing, the software is ready to be released. Continuous Delivery (CD) tools automate the process of releasing code to various environments, such as staging or production. This ensures that code is always in a deployable state, and software can be released at any time. Release management tools help to track and monitor the deployment process, reducing the chances of errors during the release phase.

6. Deploy

Deployment involves moving the code into the production environment where end-users can access it. DevOps practices focus on automating deployment processes to ensure that code is deployed consistently and reliably. Infrastructure as Code (IaC) is often used to manage and provision production environments, ensuring that infrastructure is consistent and scalable. The deployment process is typically automated using CI/CD pipelines, enabling faster and more frequent releases.

7. Operate

Once the software is live, it enters the operations phase, where it is actively monitored for performance, uptime, and user experience. Teams use monitoring and logging tools to track application performance, system health, and detect any potential issues. This phase ensures that the software is running smoothly in production, and operational teams are ready to address any incidents, downtime, or scaling needs.

8. Monitor

Continuous monitoring plays a key role in the DevOps lifecycle. Monitoring tools help track application and system performance, logging errors, and gathering feedback from users. Insights from this data help teams identify potential problems, improve performance, and ensure high availability. Monitoring tools also provide feedback to developers and operations teams, enabling them to make data-driven decisions and continuously improve the application and infrastructure.

Feedback Loop

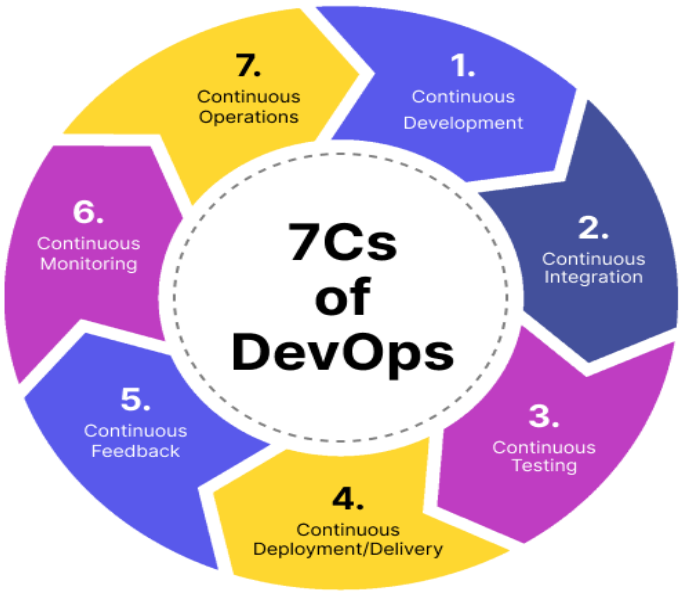
Throughout the entire DevOps lifecycle, continuous feedback is essential. Data from monitoring and logging, as well as feedback from users, operations, and quality assurance teams, helps guide the development of new features, bug fixes, and optimizations. This feedback is incorporated into the planning and development phases, creating a loop that drives constant improvement in the software and processes.

4. Devops delivery pipeline

The **DevOps Delivery Pipeline** is a set of automated processes and stages that facilitate the continuous delivery (CD) of software from development to production. It integrates various DevOps practices such as continuous integration (CI), continuous testing, and continuous deployment, enabling teams to deliver software in an efficient, consistent, and reliable manner. The pipeline automates the entire process, reducing manual effort, minimizing errors, and accelerating the software delivery lifecycle.

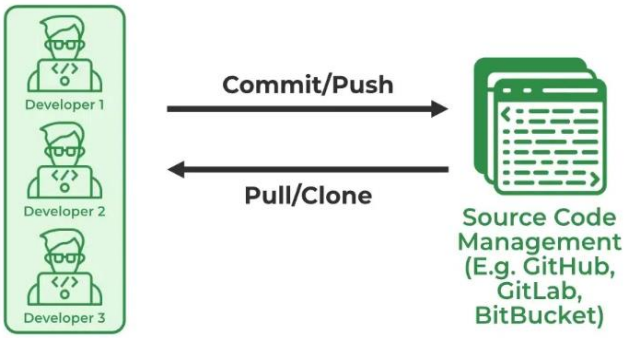
7 Cs of DevOps

1. Continuous Development
2. Continuous Integration
3. Continuous Testing
4. Continuous Deployment/Continuous Delivery
5. Continuous Monitoring
6. Continuous Feedback
7. Continuous Operations



1. Continuous Development

In Continuous Development code is written in small, continuous bits rather than all at once, Continuous Development is important in DevOps because this improves efficiency every time a piece of code is created, it is tested, built, and deployed into production. Continuous Development raises the standard of the code and streamlines the process of repairing flaws, vulnerabilities, and defects. It facilitates developers' ability to concentrate on creating high-quality code.



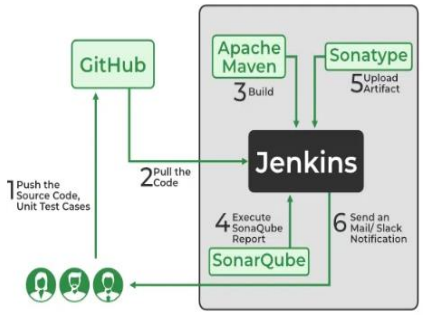
2. Continuous Integration

Continuous Integration can be explained mainly in 4 stages in DevOps. They are as follows:

1. Getting the SourceCode from SCM
2. Building the code
3. Code quality review
4. Storing the build artifacts

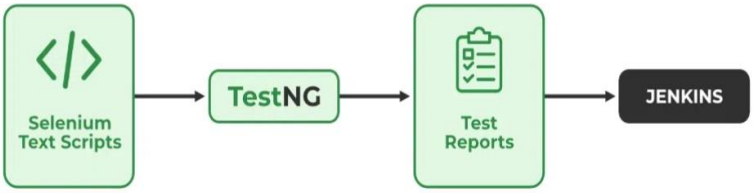
The stages mentioned above are the flow of Continuous Integration and we can use any of the tools that suit our requirement in each stage and of the most popular tools are **GitHub for source code management(SCM)** when the developer develops the code on his local machine he pushes it to the remote repository which is GitHub from here who is having the access can Pull, clone and can make required changes to the code. From there by using **Maven we can build** them into the required package (war, jar, ear) and can test the Junit cases.**SonarQube performs code quality reviews** where it will measure the quality of

source code and generates a report in the form of HTML or PDF format. **Nexus for storing the build artifacts** will help us to store the artifacts that are build by using Maven and this whole process is achieved by using a Continuous Integration tool Jenkins.



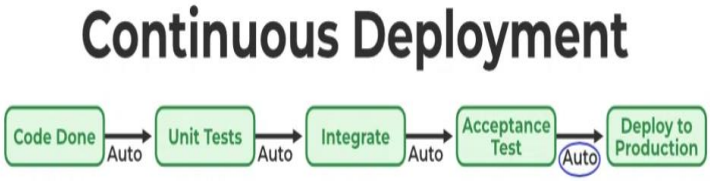
3. Continuous Testing

Any firm can deploy continuous testing with the use of the agile and DevOps methodologies. Depending on our needs, we can perform continuous testing using automation testing tools such as **Testsigma, Selenium, LambdaTest,** etc. With these tools, we can test our code and prevent problems and code smells, as well as test more quickly and intelligently. With the aid of a continuous integration platform like Jenkins, the entire process can be automated, which is another added benefit.



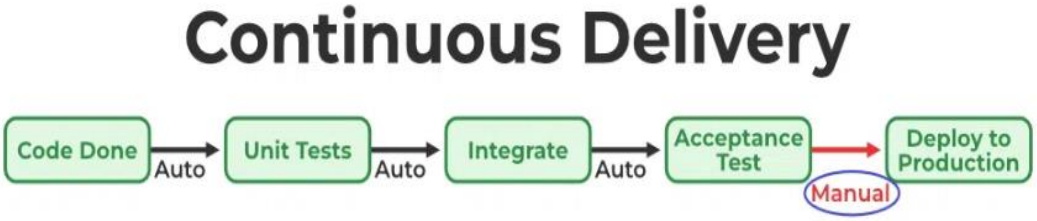
4.Continuous Deployment/ Continuous Delivery

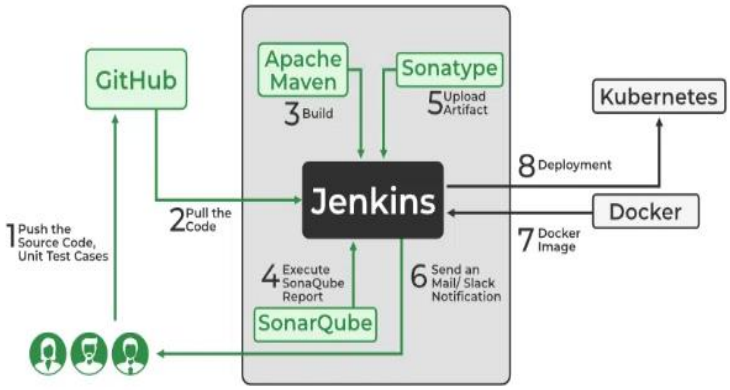
Continuous Deployment: Continuous Deployment is the process of automatically deploying an application into the production environment when it has completed testing and the build stages. Here, we'll automate everything from obtaining the application's source code to deploying it.



Continuous Delivery: Continuous Delivery is the process of deploying an application into production servers manually when it has completed testing and the build stages. Here, we'll automate the continuous integration

processes, however, manual involvement is still required for deploying it to the production environment.





5. Continuous Monitoring

DevOps lifecycle is incomplete if there was no Continuous Monitoring. Continuous Monitoring can be achieved with the help of Prometheus and Grafana we can continuously monitor and can get notified before anything goes wrong with the help of Prometheus we can gather many performance measures, including CPU and memory utilization, network traffic, application response times, error rates, and others. Grafana makes it possible to visually represent and keep track of data from time series, such as CPU and memory utilization.

6. Continuous Feedback

Once the application is released into the market the end users will use the application and they will give us feedback about the performance of the application and any glitches affecting the user experience after getting multiple feedback from the end users' the DevOps team will analyze the feedbacks given by end users and they will reach out to the developer team tries to rectify the mistakes they are performed in that piece of code by this we can reduce the errors or bugs that which we are currently developing and can produce much more effective results for the end users also we reduce any unnecessary steps to deploy the application. Continuous Feedback can increase the performance of the application and reduce bugs in the code making it smooth for end users to use the application.

7. Continuous Operations

We will sustain the higher application uptime by implementing continuous operation, which will assist us to cut down on the maintenance downtime that will negatively impact end users' experiences. More output, lower manufacturing costs, and better quality control are benefits of continuous operations.

5. Devops Ecosystem and Tools

The **DevOps Ecosystem** refers to the collection of tools, practices, and technologies that work together to support and automate the entire software development and delivery lifecycle. These tools are designed to help streamline development, continuous integration, testing, deployment, monitoring, and collaboration across development and operations teams. The goal is to enhance the speed, reliability, and quality of software delivery.

Below are key components of the **DevOps Ecosystem** and the tools commonly used in each stage:

1. Version Control

Version control systems help teams track changes in code, collaborate effectively, and maintain a history of modifications. They are critical for managing source code and ensuring collaboration across teams.

- **Popular Tools**:
  - **Git**: A distributed version control system used for tracking changes in source code.
  - **GitHub/GitLab/Bitbucket**: Hosting services for Git repositories, offering additional features like issue tracking, code reviews, and CI/CD integration.

2. Continuous Integration (CI) and Continuous Delivery (CD)

CI/CD tools automate the process of integrating code changes, running tests, and deploying software. CI focuses on automating the build and testing processes, while CD automates the deployment of code to production or staging environments.

- **Popular Tools**:
  - **Jenkins**: A widely-used automation server for building, testing, and deploying code.
  - **Travis CI**: A cloud-based CI tool for building and testing code, integrated with GitHub.
  - **CircleCI**: A cloud-native CI/CD tool that automates testing and deployment pipelines.
  - **GitLab CI/CD**: A built-in CI/CD solution within GitLab for automating testing and deployments.

3. Configuration Management and Infrastructure as Code (IaC)

These tools automate the configuration and provisioning of infrastructure. Infrastructure as Code (IaC) enables teams to define infrastructure using code, which makes it reproducible, scalable, and easier to manage.

- **Popular Tools**:
  - **Ansible**: A configuration management tool for automating server configuration and software deployment.
  - **Chef**: A configuration management tool that allows infrastructure to be defined as code.
  - **Puppet**: Used for automating the provisioning and management of servers.
  - **Terraform**: A popular IaC tool for provisioning and managing cloud infrastructure across multiple cloud providers.

4. Containerization and Orchestration

Containers package an application and its dependencies into a single unit, enabling portability across environments. Orchestration tools automate the deployment, scaling, and management of containerized applications.

- **Popular Tools**:
  - **Docker**: A platform for building, shipping, and running applications in containers.
  - **Kubernetes**: A container orchestration platform for automating the deployment, scaling, and management of containerized applications.
  - **Docker Compose**: A tool for defining and running multi-container Docker applications.
  - **OpenShift**: A Kubernetes-based container platform that provides enhanced security and scalability.

5. Monitoring and Logging

Monitoring and logging tools collect data from applications and infrastructure to provide insights into system performance, health, and user behavior. They enable proactive issue detection and help in troubleshooting.

- **Popular Tools**:
  - **Prometheus**: A monitoring and alerting toolkit designed for reliability and scalability.
  - **Grafana**: An open-source analytics platform that integrates with Prometheus to visualize and analyze monitoring data.
  - **ELK Stack (Elasticsearch, Logstash, Kibana)**: A powerful set of tools for searching, analyzing, and visualizing logs in real time.
  - **Splunk**: A platform for searching, analyzing, and visualizing machine-generated data (logs) in real time.
  - **Datadog**: A monitoring and analytics tool for cloud-scale applications, providing real-time observability.

6. Collaboration and Communication

DevOps requires strong communication and collaboration between development, operations, and other teams. Tools for collaboration ensure that everyone is on the same page and can quickly respond to issues.

- **Popular Tools**:
  - **Slack**: A messaging platform for teams that integrates with various DevOps tools and provides real-time communication.
  - **Microsoft Teams**: A collaboration platform that offers chat, video meetings, file sharing, and integrations with DevOps tools.
  - **Jira**: A project management tool used for issue tracking, agile planning, and team collaboration.
  - **Confluence**: A knowledge-sharing tool often used alongside Jira to document processes, guidelines, and best practices.

7. Security (DevSecOps)

DevSecOps integrates security practices into the DevOps pipeline. Security tools help automate vulnerability scanning, code analysis, and security testing to ensure that security is baked into the development process.

- **Popular Tools**:
  - **SonarQube**: A tool for static code analysis that helps detect bugs, vulnerabilities, and code smells in the codebase.
  - **OWASP ZAP**: A penetration testing tool that identifies security vulnerabilities in web applications.
  - **Aqua Security**: A security platform for securing containerized applications and cloud-native environments.
  - **Snyk**: A tool for finding and fixing vulnerabilities in open-source libraries, containers, and infrastructure.

8. Automated Testing

Automated testing is a key part of the DevOps process, allowing teams to ensure code quality and functionality early in the development cycle. It involves unit tests, integration tests, and acceptance tests.

- **Popular Tools**:
  - **Selenium**: A popular tool for automating web browsers to test web applications.
  - **JUnit**: A widely-used framework for unit testing in Java applications.
  - **TestNG**: A testing framework designed for test configuration and running tests in parallel.
  - **Cypress**: A tool for end-to-end testing of web applications, focusing on developer productivity and ease of use.

9. Artifact Repositories

Artifact repositories store build artifacts such as libraries, dependencies, and executables, which can be used during the build, test, and deployment stages.

- **Popular Tools**:
  - **Nexus Repository**: A repository manager for managing software artifacts and dependencies.
  - **Artifactory**: A universal artifact repository that supports Maven, Docker, and other technologies.
  - **AWS CodeArtifact**: A fully managed artifact repository service that integrates with AWS services.

Popular DevOps Tools:

Category	Tool Examples	Functionality
Planning & Tracking	Jira, Asana, Trello	Project management, task tracking, issue management
Development	Git, Visual Studio Code, Eclipse	Code development, version control, code editing
Testing	Selenium, JUnit, Pytest	Automated testing, test automation frameworks
Deployment	Terraform, Ansible, Jenkins	Infrastructure provisioning, application deployment, automation
Monitoring	Prometheus, Grafana, ELK Stack	Real-time monitoring, performance analysis, logging
CI/CD	Jenkins, CircleCI, GitHub Actions	Automated build, test, and deployment pipelines
Containerization	Docker	Packaging applications into containers
Container Orchestration	Kubernetes	Automating deployment, scaling, and management of containers
Infrastructure as Code (IaC)	Terraform, Ansible	Defining and managing infrastructure as code

6. The Agile wheel of wheels.

The **Agile Wheel of Wheels** is a visual metaphor that represents the various interconnected elements that drive Agile software development. It emphasizes that Agile is not a single linear process, but rather a continuous cycle of feedback, improvement, and collaboration. The wheel's "spokes" represent the principles, practices, and values that support Agile methods, and the "wheels" themselves represent different aspects of Agile that work together to form a cohesive, iterative development process.

The **Agile Wheel of Wheels** typically includes the following core components:

1. Agile Values

Agile is based on four key values:

- **Individuals and interactions over processes and tools**: Focus on people working together effectively rather than rigid processes or tools.
- **Working software over comprehensive documentation**: Prioritize delivering functional software that provides value.
- **Customer collaboration over contract negotiation**: Engage with customers throughout the development process to ensure their needs are met.
- **Responding to change over following a plan**: Be adaptable to changes in requirements and external conditions.

2. Agile Principles

Agile development is governed by 12 principles that guide teams toward delivering value more efficiently. Some of these principles include:

- Delivering working software frequently, with a preference for shorter timescales.
- Welcoming changing requirements, even late in development.
- Maintaining a sustainable pace of work.
- Fostering close, daily cooperation between business stakeholders and developers.

3. Collaboration

Collaboration is key in Agile. The "wheel" emphasizes teamwork between developers, business stakeholders, customers, and even between different teams. Agile relies on constant feedback and open communication channels to ensure that the software aligns with user needs and business goals.

4. Iteration and Increment

Agile processes are iterative and incremental. Development is broken down into small, manageable units (called **iterations** or **sprints**) that typically last 1-4 weeks. Each iteration results in a working increment of software. The short cycles allow for continuous improvement, feedback, and adaptation.

5. Continuous Improvement

Agile practices encourage continuous improvement of processes, workflows, and the product itself. Retrospectives, which are held at the end of each iteration, provide a forum for teams to reflect on their performance, discuss challenges, and plan improvements for the next cycle.

6. Customer-Centric Development

At the heart of Agile is the focus on delivering value to customers. The Agile wheel stresses continuous customer involvement and feedback, ensuring that the software being developed truly meets user needs and delivers meaningful outcomes.

7. Simplicity

Agile encourages **simplicity** in design and development. By focusing on the most essential features and avoiding overcomplication, Agile aims to deliver the maximum amount of value with the least amount of work. Simplicity also applies to decision-making, prioritizing tasks, and maintaining flexibility.

8. Self-Organizing Teams

Agile promotes **self-organizing teams** that have the autonomy to make decisions about how to best complete their work. This empowerment fosters creativity, accountability, and faster decision-making, which leads to better outcomes and higher morale.

9. Feedback Loops

Agile thrives on feedback loops, where progress is regularly assessed, and corrections are made in response to new insights. This includes both technical feedback (e.g., from automated tests) and business feedback (e.g., from stakeholders or customers). These loops ensure that the project stays aligned with its goals.

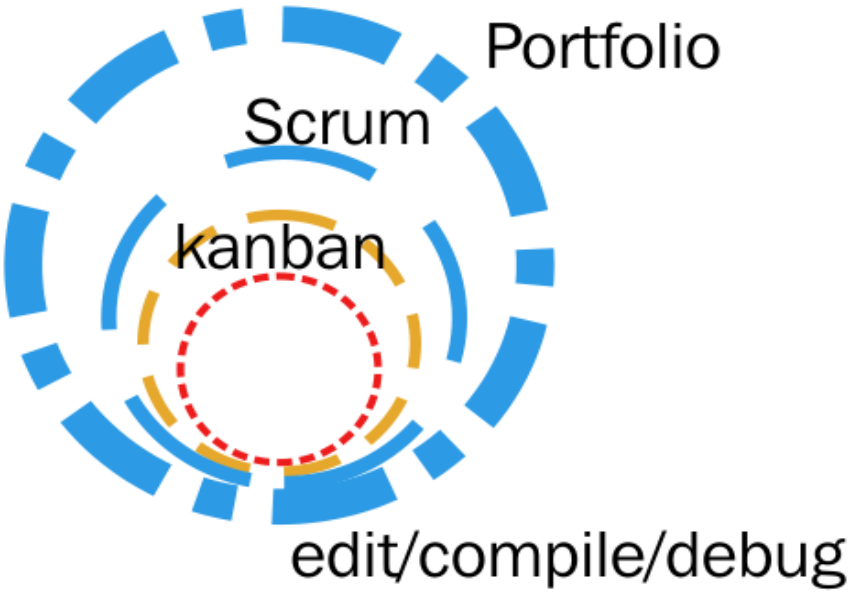
10. Sustainability

Agile methodologies emphasize sustainable development practices. Teams strive for a pace that can be maintained over the long term without burnout. The idea is to create a sustainable work environment where developers can keep producing quality work over extended periods.

The Agile Wheel of Wheels represents these interconnected elements in a circular, continuous manner, where each part supports and enhances the others. By iterating on all these aspects—values, principles, collaboration, and feedback—Agile teams are able to adapt, improve, and deliver value in an ever-changing environment. Each "wheel" turns independently, but they all drive toward the same goal: creating better software, faster.

The Agile wheel of wheels:-

There are several different cycles in Agile development, from the Portfolio level through to the Scrum and Kanban cycles and down to the **Continuous Integration** (**CI**) cycle. The emphasis on which cadence work happens in is a bit different depending on which Agile framework you are working with. Kanban emphasizes the 24-hour cycle, which is popular in operations teams. Scrum cycles can be two to four weeks and are often used by development teams using the Scrum Agile process. Longer cycles are also common and are called **Program Increments** (**PI**), which span several Scrum Sprint cycles, in the **Scaled Agile Framework** (**SAFe®**):



The Agile wheel of wheels

DevOps must be able to support all these cycles. This is quite natural given the central theme of DevOps: cooperation between disciplines in an Agile organization.

The most obvious and measurably concrete benefits of DevOps occur in the shorter cycles, which in turn makes the longer cycles more efficient.

Here are some examples of when DevOps can benefit Agile cycles:

- Deployment systems, maintained by DevOps engineers, make the deliveries at the end of Scrum cycles faster and more efficient. These can take place with a periodicity of two to four weeks.
- In organizations where deployments are done mostly by hand, the time to deploy can be several days. Organizations that have these inefficient deployment processes will benefit greatly from a DevOps mindset.
- The Kanban cycle is 24 hours, and it's therefore obvious that the deployment cycle needs to be much faster than that if we are to succeed with Kanban.
- A well-designed DevOps CD pipeline can deploy code from being committed to the code repository to production in a matter of minutes, depending on the size of the change.

Version Control with Git:-

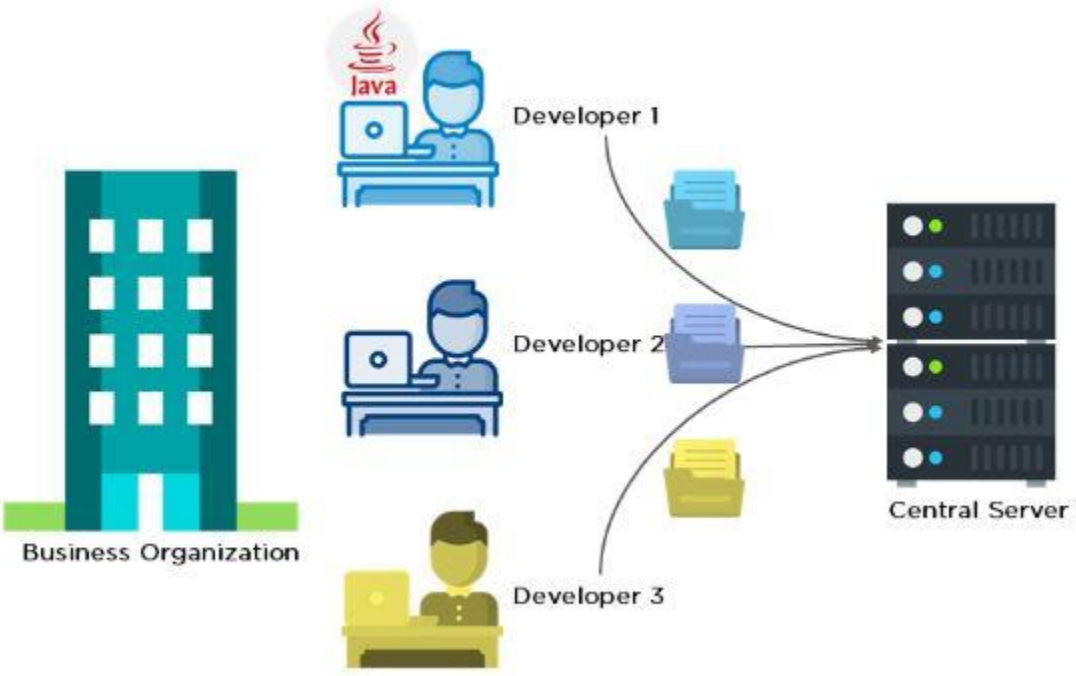
1. **Introduction to version control**
2. **Basics of Git**
3. **Installing and Configuring Git**
4. **Common Git Commands**
5. **Branching and Merging Strategies**
6. **Working with Remote Repositories**

1. Introduction to version control

Getting Started with Git

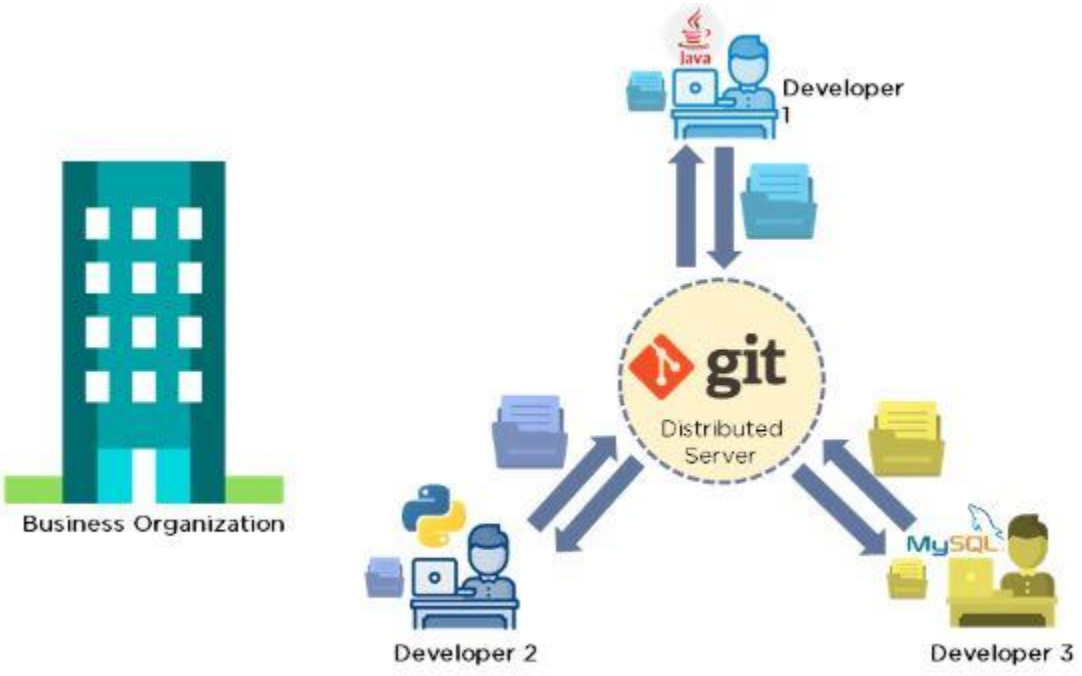
Before diving deep, let's explain a scenario before Git:

- Developers used to submit their codes to the central server without having copies of their own
- Any changes made to the source code were unknown to the other developers
- There was no communication between any of the developers



Now let's look at the scenario after Git:

- Every developer has an entire copy of the code on their local systems
- Any changes made to the source code can be tracked by others
- There is regular communication between the developers



1.1 Git Version Control System(VCS)

A version control system is a software that tracks changes to a file or set of files over time so that you can recall specific versions later. It also allows you to work together with other programmers.

The version control system is a collection of software tools that help a team to manage changes in a source code. It uses a special kind of database to keep track of every modification to the code.

Developers can compare earlier versions of the code with an older version to fix the mistakes.

- **Benefits of the Version Control System**

The Version Control System is very helpful and beneficial in software development; developing software without using version control is unsafe. It provides backups for uncertainty. Version control systems offer a speedy interface to developers. It also allows software teams to preserve efficiency and agility according to the team scales to include more developers.

Some key benefits of having a version control system are as follows.

- Complete change history of the file
- Simultaneously working
- Branching and merging
- Traceability

Types of Version Control System

a. Localized version Control System

b. Centralized version control systems

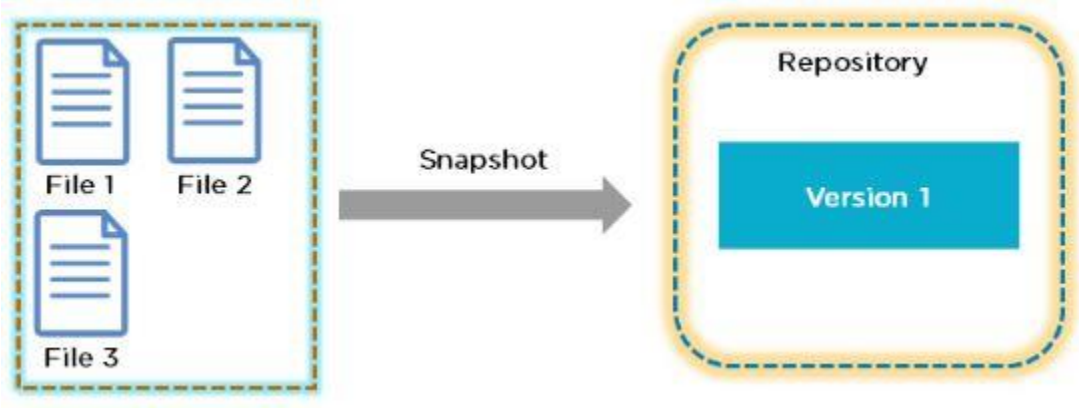
c. Distributed version control systems

a. Localized Version Control Systems

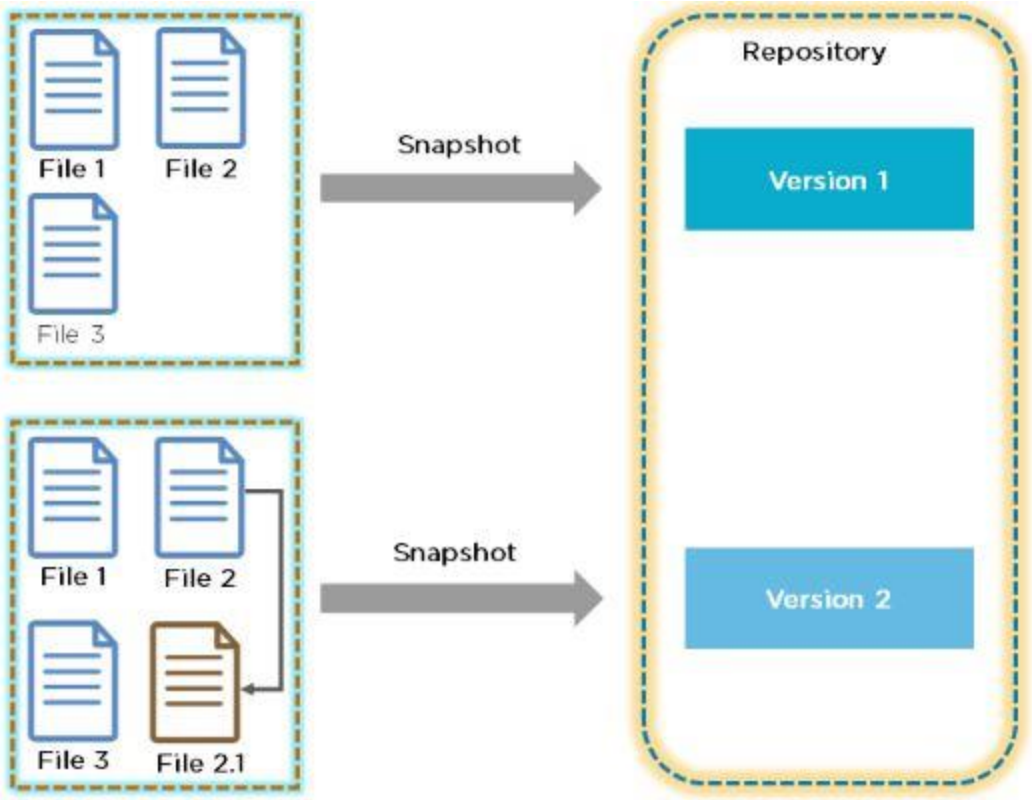
The localized version control method is a common approach because of its simplicity. But this approach leads to a higher chance of error. In this approach, you may forget which directory you're in and accidentally write to the wrong file or copy over files you don't want to.

To deal with this issue, programmers developed local VCSs that had a simple database. Such databases kept all the changes to files under revision control. A local version control system keeps local copies of the files.

The diagram below shows there are three files in the local system. A snapshot of these files are stored in the remote repository as Version 1.



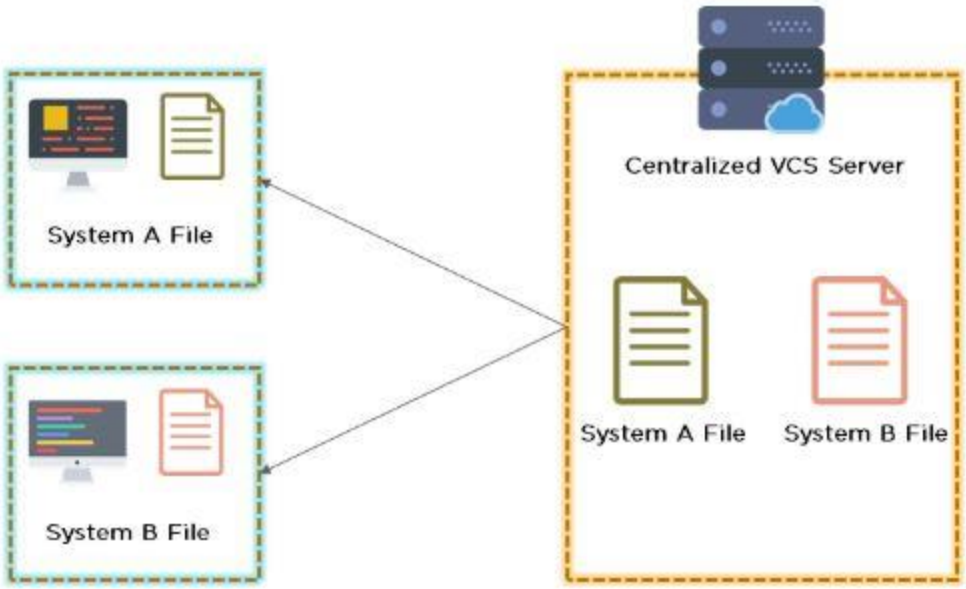
The following diagram includes a few changes:



There have been some changes to file 2 and is updated to file 2.1. This change is stored as Version 2 in the repository. VCS enables you to track the history of a file collection. Each version captures a snapshot of the files at a certain point in time, and the VCS allows you to switch between these versions.

The major drawback of Local VCS is that it has a single point of failure.

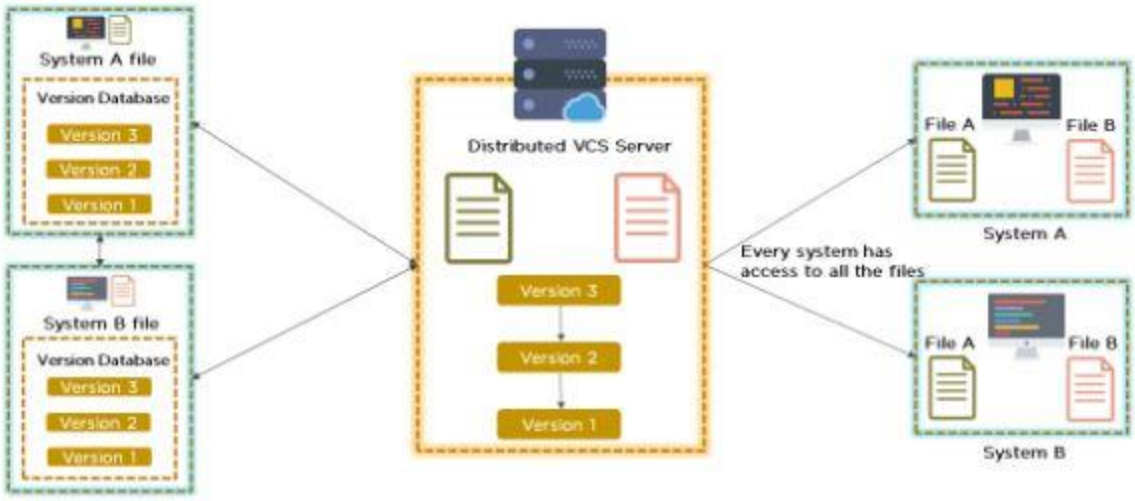
b. Centralized Version Control System



- Uses a central server to store all the files
- Every operation is performed directly on the repository
- All the versions of the file are stored on the Central VCS server
- In case the central server crashes, the entire data of the project will be lost. Hence, distributed VCS was introduced.

c. Distributed Version Control System

- Every programmer has a copy of all the versions of the code on their local systems
- Distributed VCS moves from the client-server approach of central VCS to a peer-to-peer approach
- They can update their local repositories with new data from the central server and changes are reflected in the principal repository
- Git is one such distributed VCS tool



- **Difference between Centralized Version Control System and Distributed Version Control System**

Centralized Version Control Systems are systems that use **client/server** architecture. In a centralized Version Control System, one or more client systems are directly connected to a central server. Contrarily the Distributed Version Control Systems are systems that use **peer-to-peer** architecture.

There are many benefits and drawbacks of using both the version control systems. Let's have a look at some significant differences between Centralized and Distributed version control system.

Centralized Version Control System	Distributed Version Control System
In CVCS, The repository is placed at one place and delivers information to many clients.	In DVCS, Every user has a local copy of the repository in place of the central repository on the server-side.
It is based on the client-server approach.	It is based on the client-server approach.
It is the most straightforward system based on the concept of the central repository.	It is flexible and has emerged with the concept that everyone has their repository.
In CVCS, the server provides the latest code to all the clients across the globe.	In DVCS, every user can check out the snapshot of the code, and they can fully mirror the central repository.
CVCS is easy to administrate and has additional control over users and access by its server from one place.	DVCS is fast comparing to CVCS as you don't have to interact with the central server for every command.
The popular tools of CVCS are **SVN** (Subversion) and **CVS**.	The popular tools of DVCS are **Git** and **Mercurial**.

CVCS is easy to understand for beginners.	DVCS has some complex process for beginners.
If the server fails, No system can access data from another system.	if any server fails and other systems were collaborating via it, that server can restore any of the client repositories

2. Basics of Git

What is 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(Source code management) tools like Subversion, CVS(Concurrent Versions System), Perforce, and ClearCase.

- ## Features of Git

Some remarkable features of Git are as follows:

1. **Open Source**

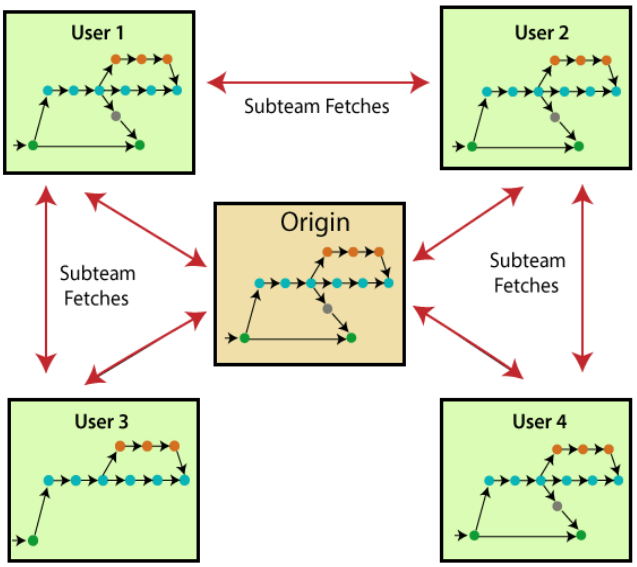
Git is an **open-source tool**. It is released under the **GPL** (General Public License) license.

2. **Scalablility**

   Git is **scalable**, which means when the number of users increases, the Git can easily handle such situations.

3. **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.



4. **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.

5. **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.

6. **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.

7. **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:

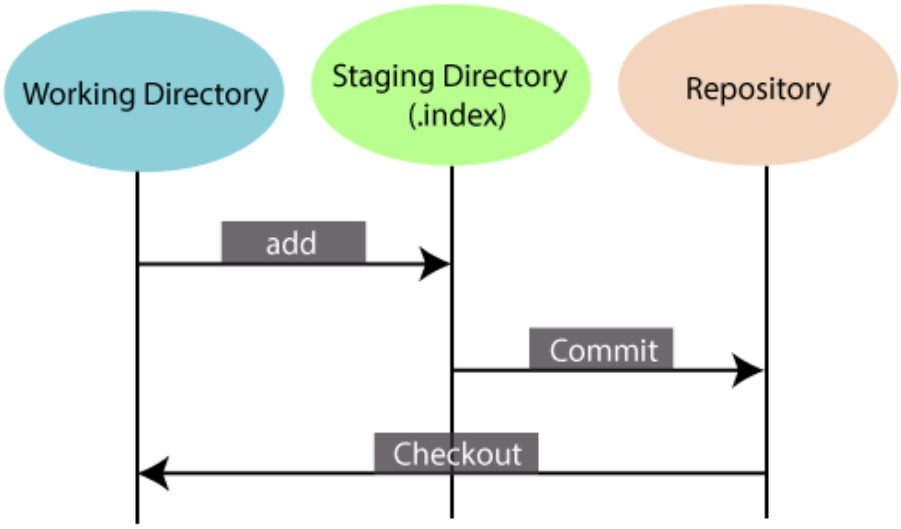
- We can **create a separate branch** for a new module of the project, commit and delete it whenever we want.
- We can have a **production branch**, which always has what goes into production and can be merged for testing in the test branch.
- We can create a **demo branch** for the experiment and check 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.

8. **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.

9. **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.**

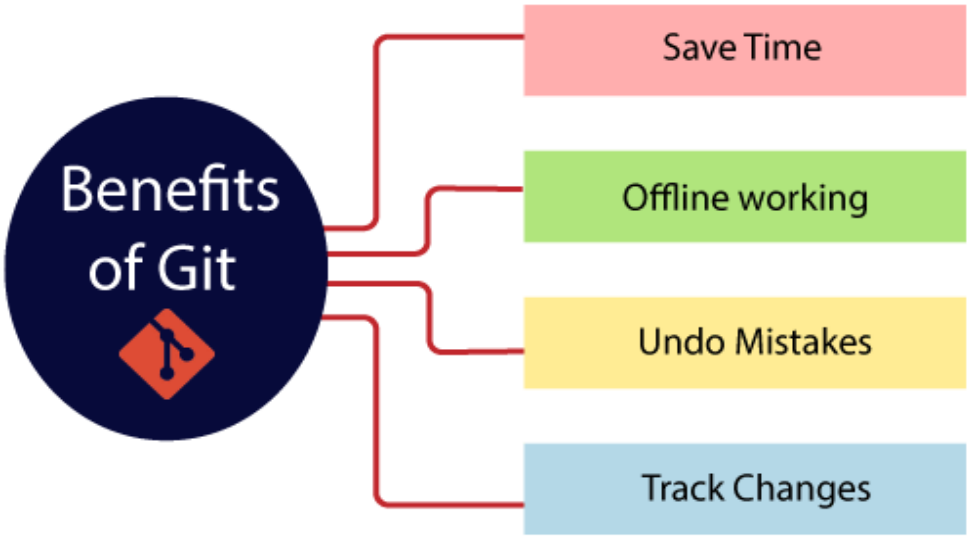
10. Maintain the clean history

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:



1. **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.

2. **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.

3. **Undo Mistakes**

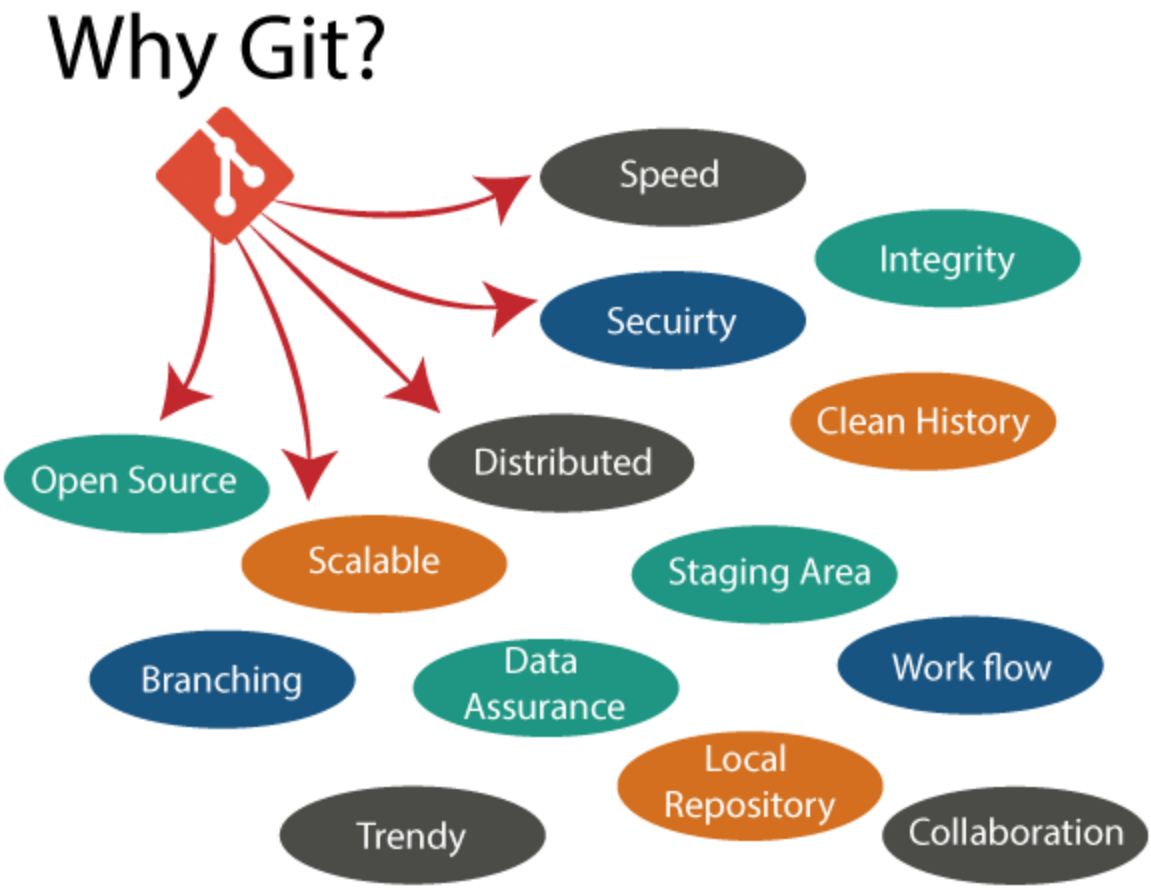
   One additional benefit of Git is we can **Undo** mistakes. Sometimes the undo can be a savior option for us. Git provides the undo option for almost everything.

4. **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.



1. **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.

2. **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.

3. **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.

4. **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.

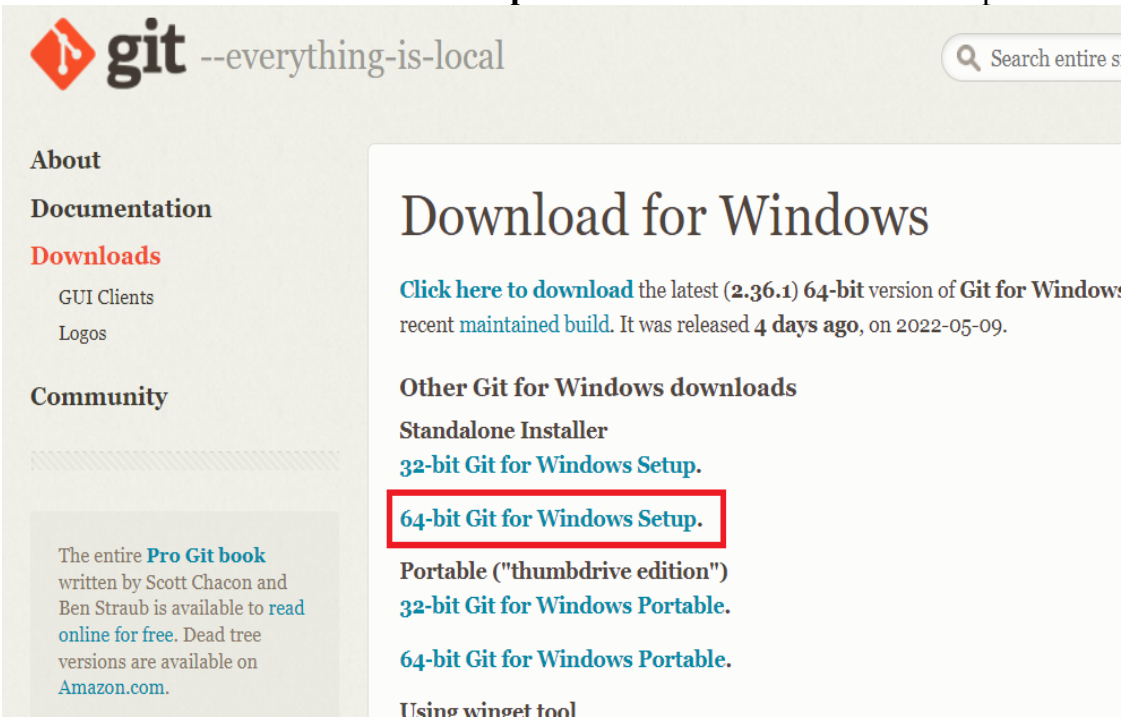
5. **Impress Recruiters**

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.

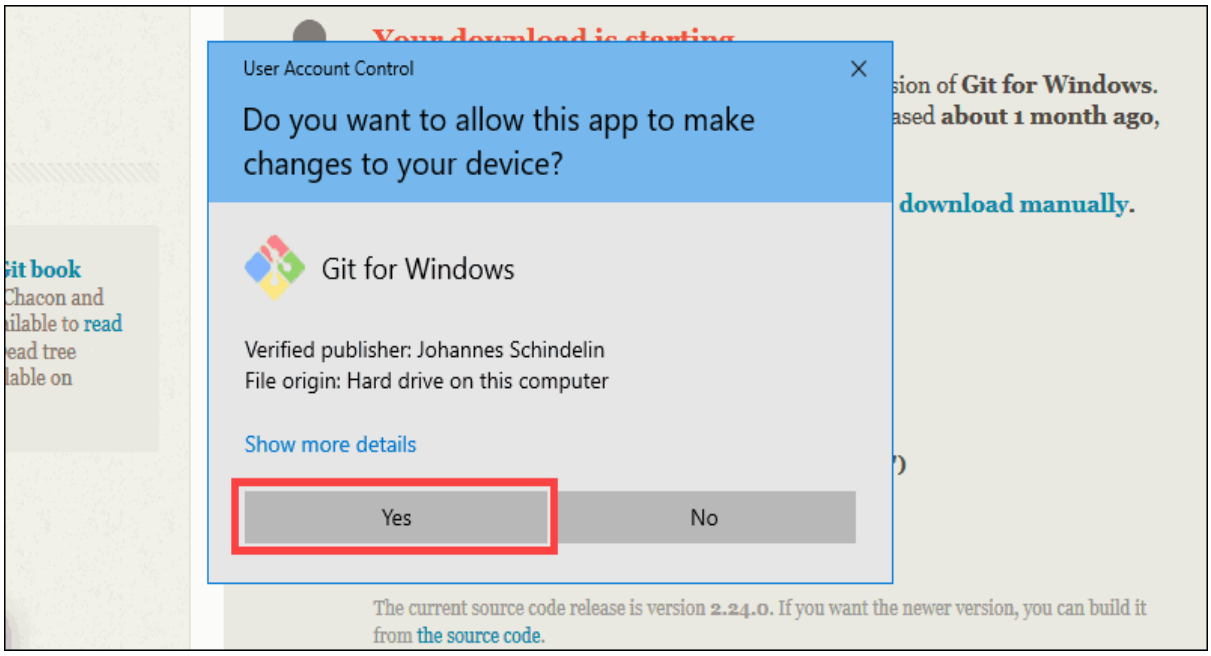
3. Installing and Configuring Git

Steps to download and install Git on Windows

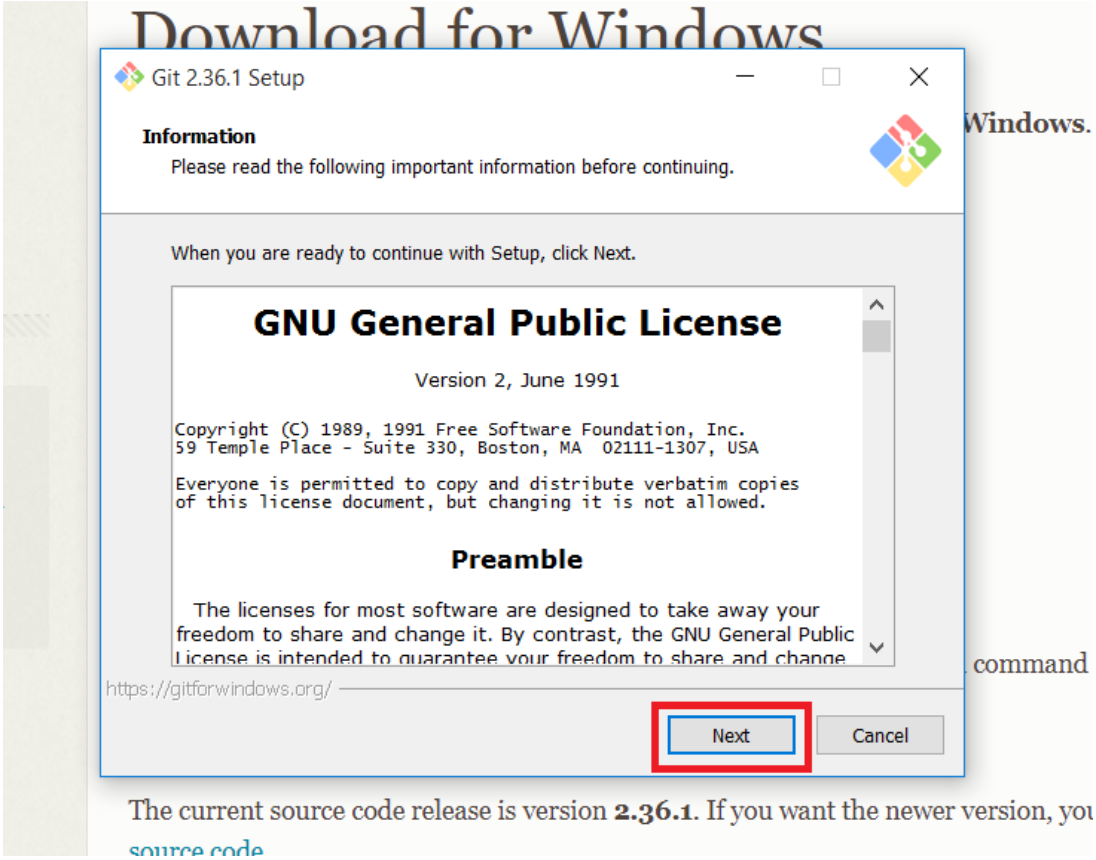
Downloading
Step 1: Go to the official website: https://git-scm.com
Step 2: Click on **64-bit Git for Windows Setup** and allow the download to complete.



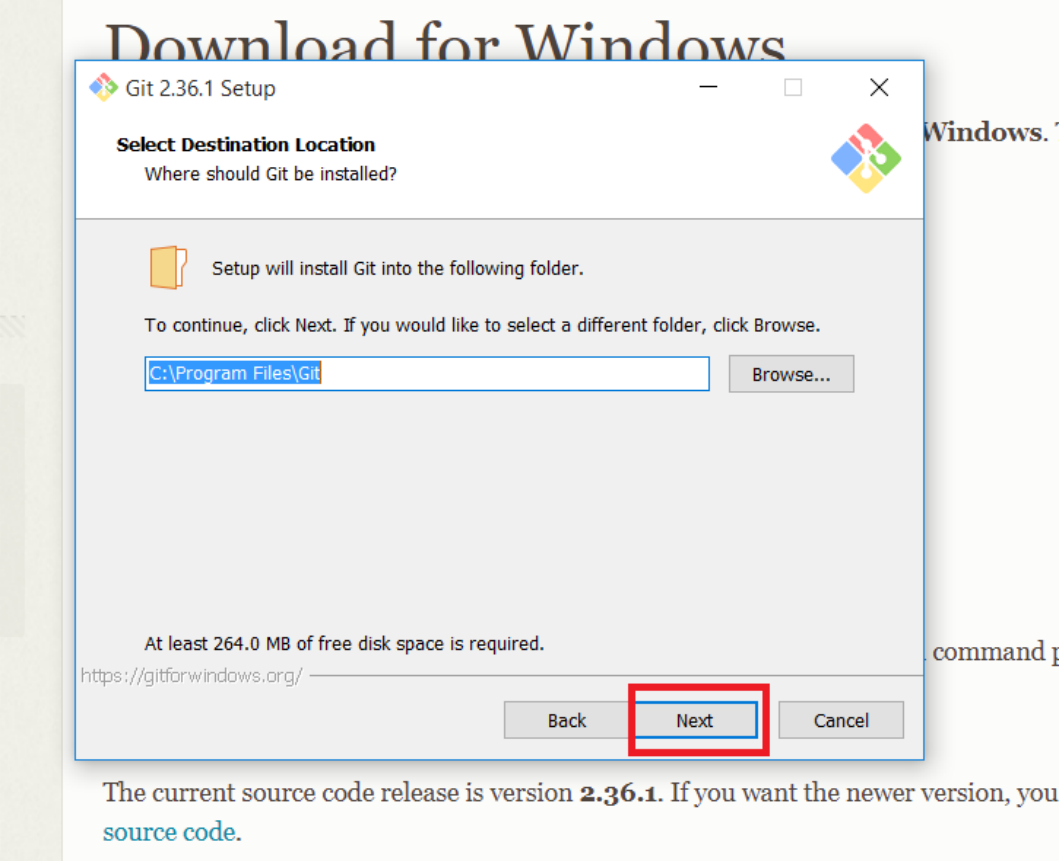
Extract and Launch Git Installer
Step 3: Go to your download location and double-click the file to launch the installer.
Step 4: Allow the app to modify your device by selecting Yes in the User Account Control window that appears.



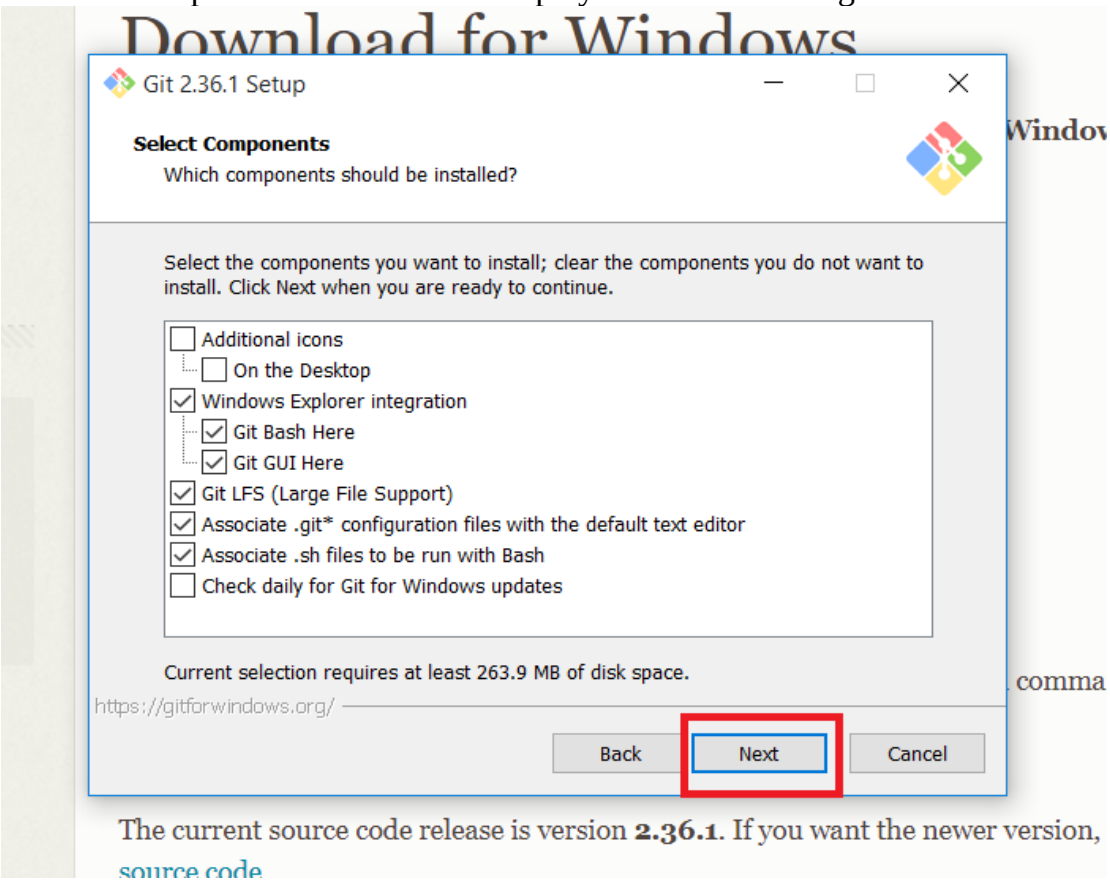
Step 5: Check the GNU General Public License and click **Next**.



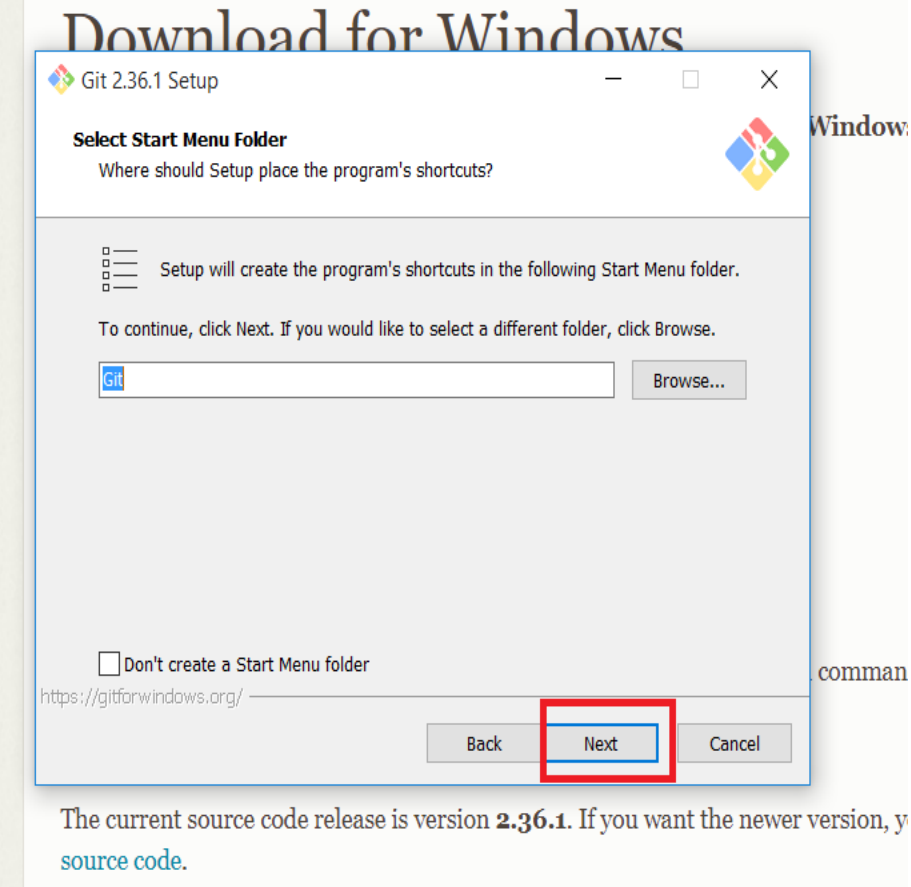
Step 6: Select the install location. If you don't have a reason to modify it, leave it to default and click **Next**.



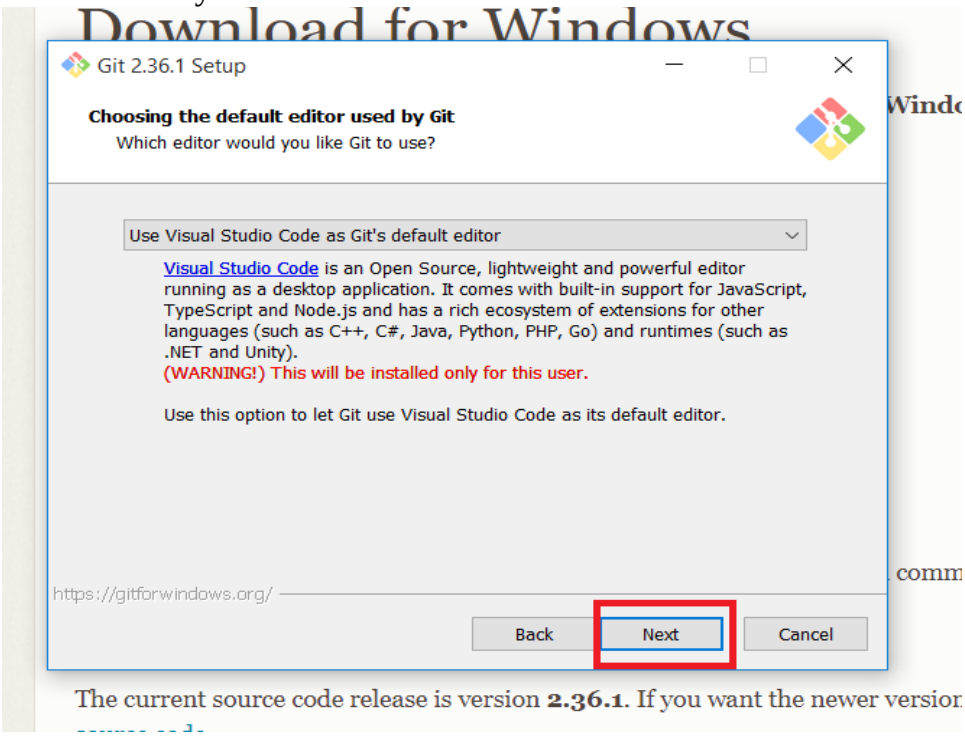
Step 7: A screen for component selection will display. Leave the settings as it is and click **Next**.



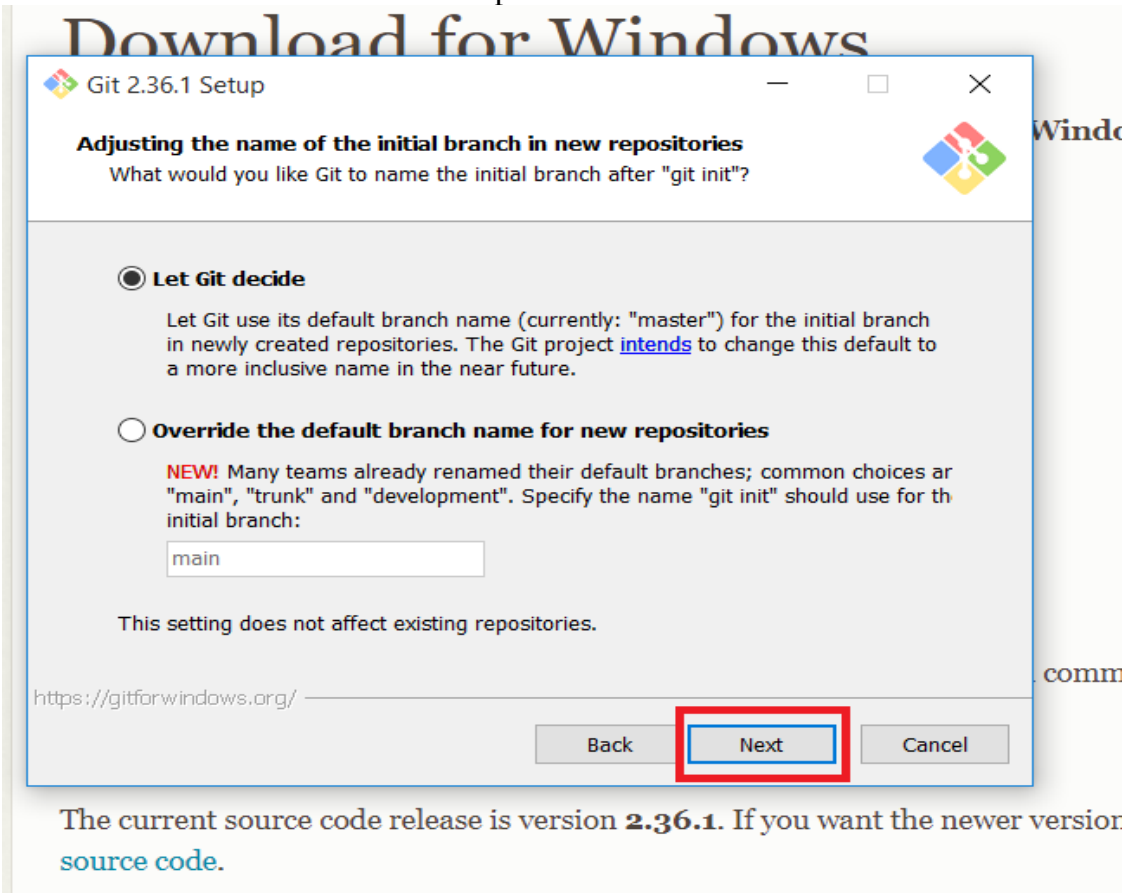
Step 8: The installer asks you to create a start menu folder. Simply click **Next.**



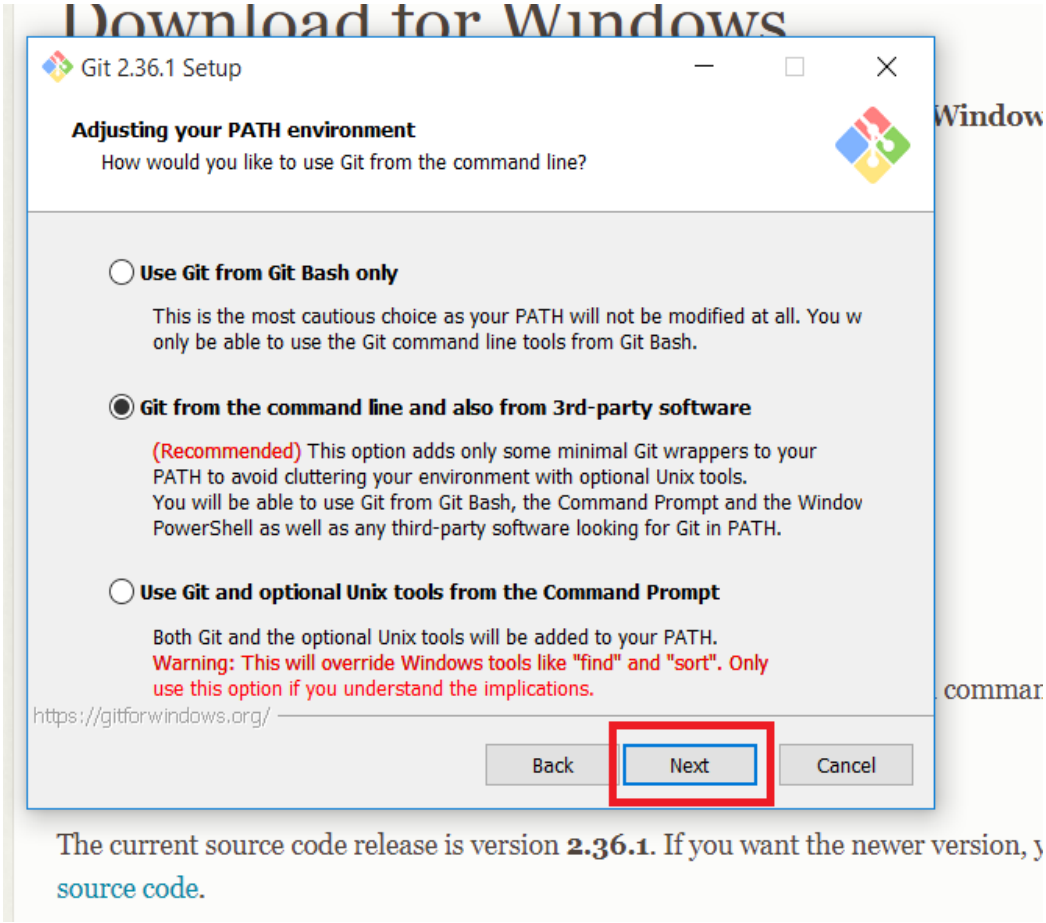
Step 9: Choose the text editor you want to use with Git and click **Next**.



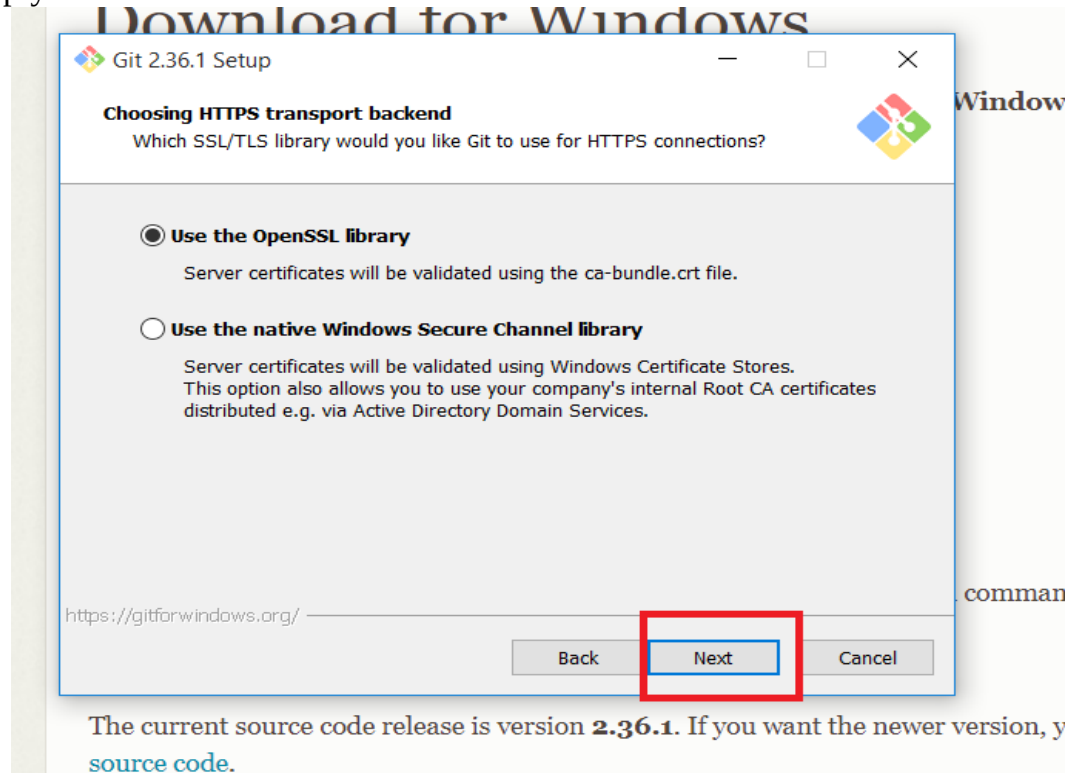
Step 10: The following step allows you to give your original branch a new name. ‘Master’ is the default. Leave the default choice selected and press the **Next** button**.**



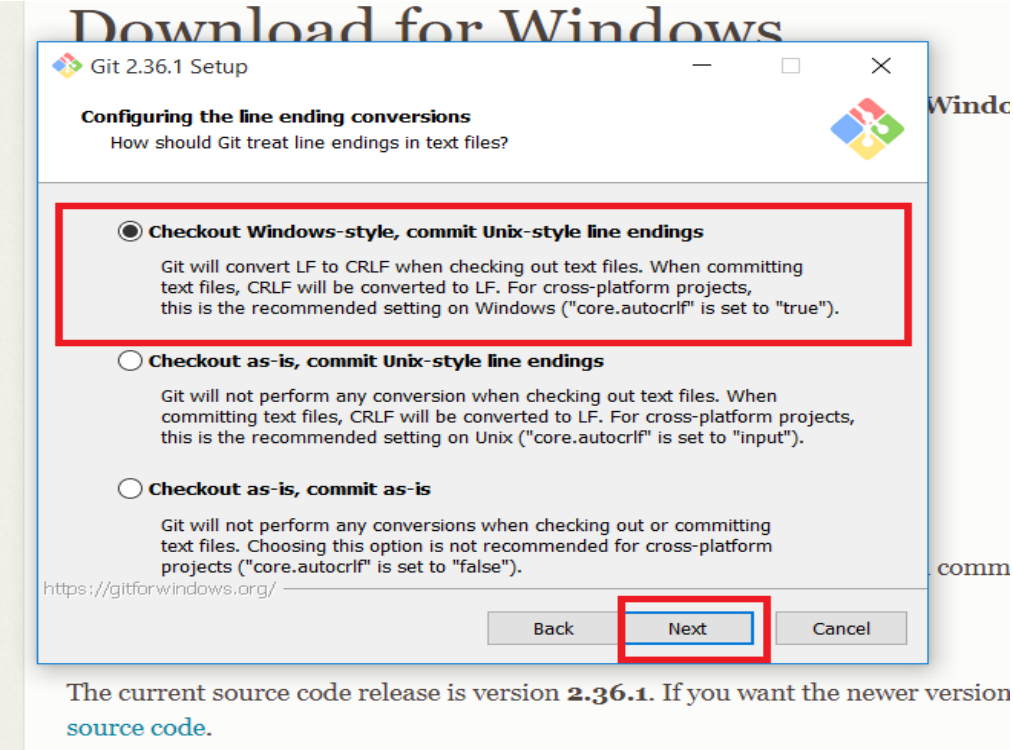
Step 11: You can adjust the PATH environment during this installation phase. When you run a command from the command line, the PATH is the default set of folders that are included. Continue by selecting the middle (recommended) option and clicking **Next**.



Step 12: The following option concerns server certificates. The default choice is used by the majority of users**.** Simply click **Next**.

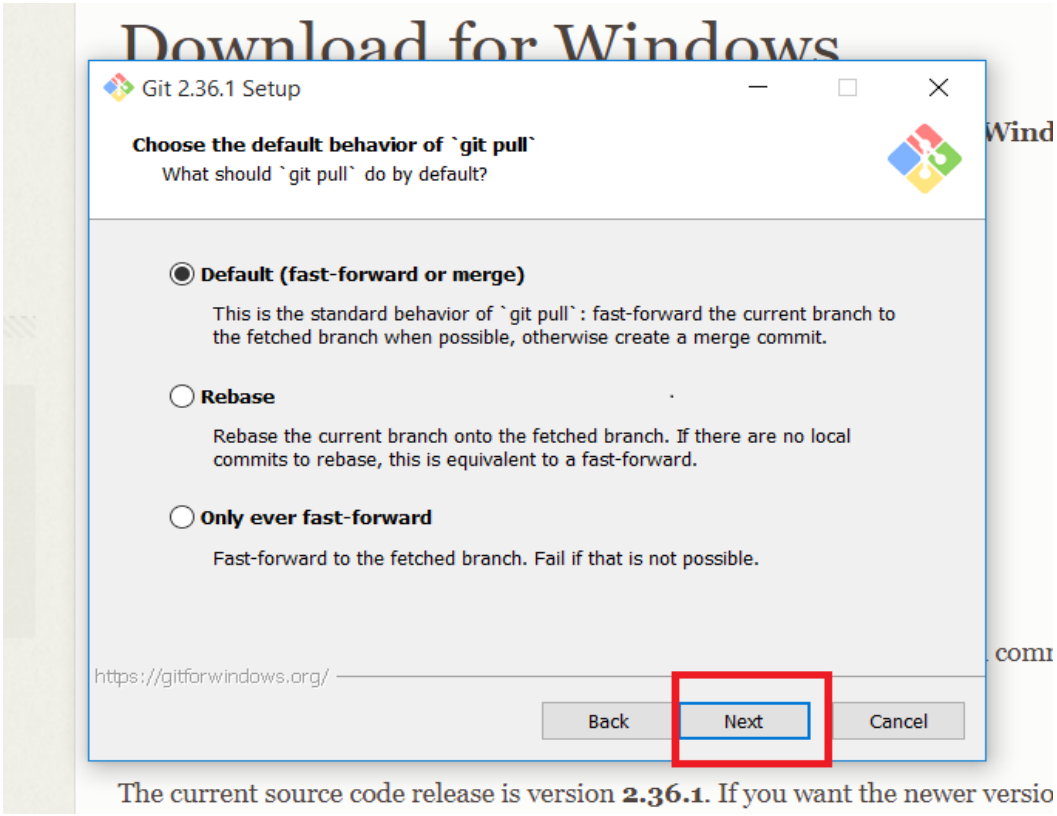


Step 13: This step deals with how data is structured, and altering this option may create issues. So, it is advised to leave the default selection.

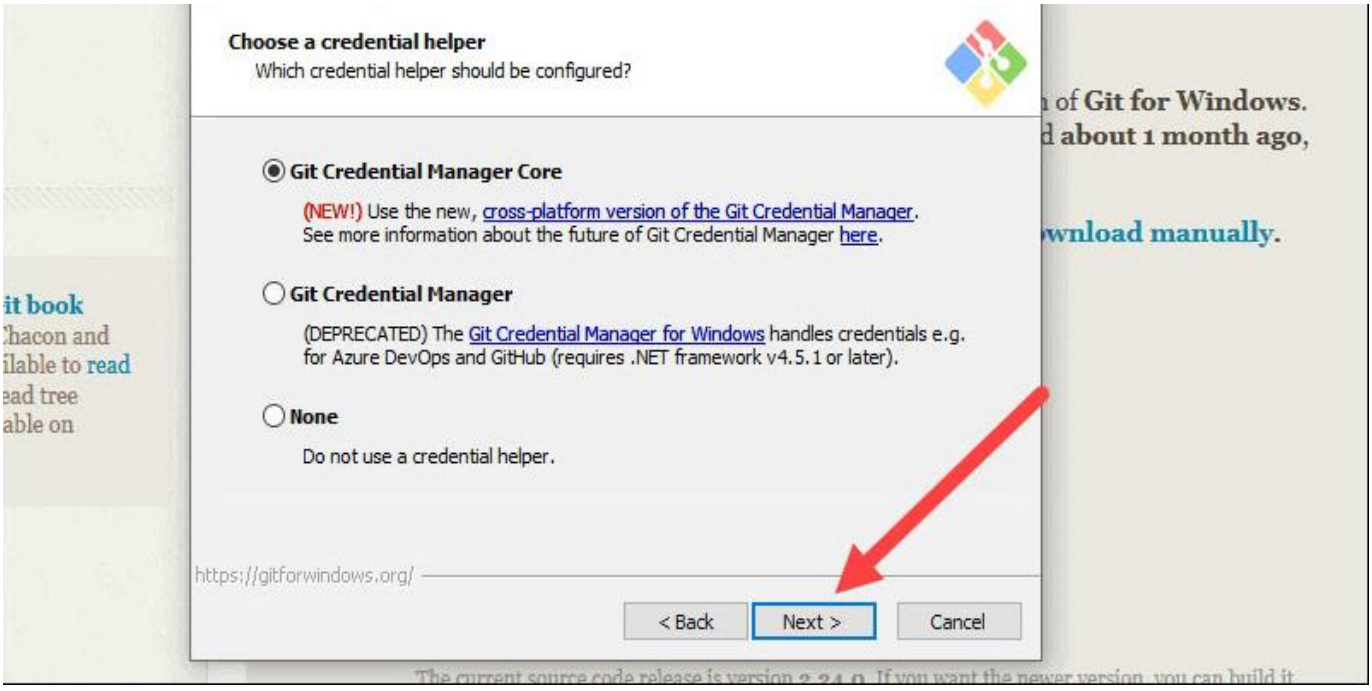


Step 14: Select the terminal emulator that you wish to use. Because of its features, the default MinTTY is suggested. Click **Next**.

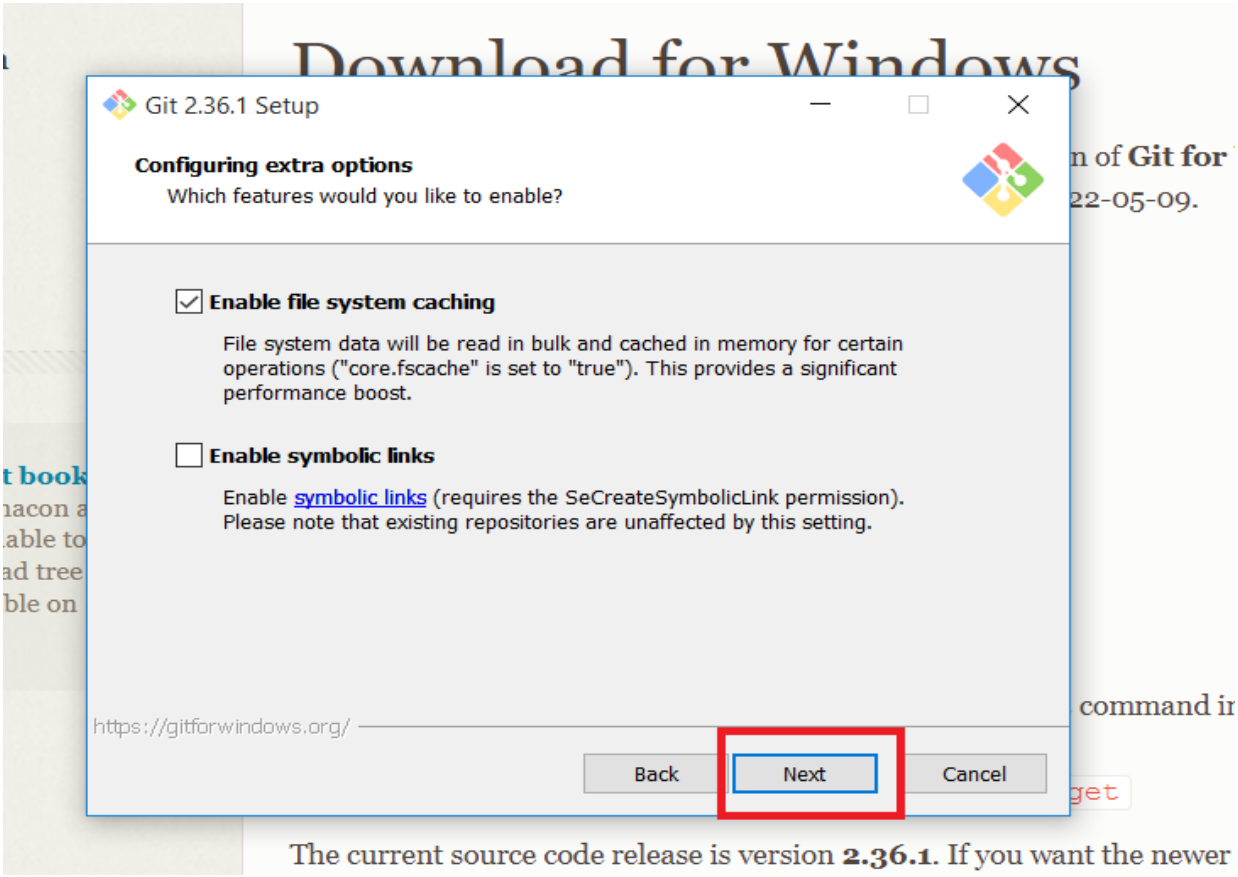
Step 15: The installer now prompts you to specify what the git pull command should perform. Leave the default selected option and click **Next**.



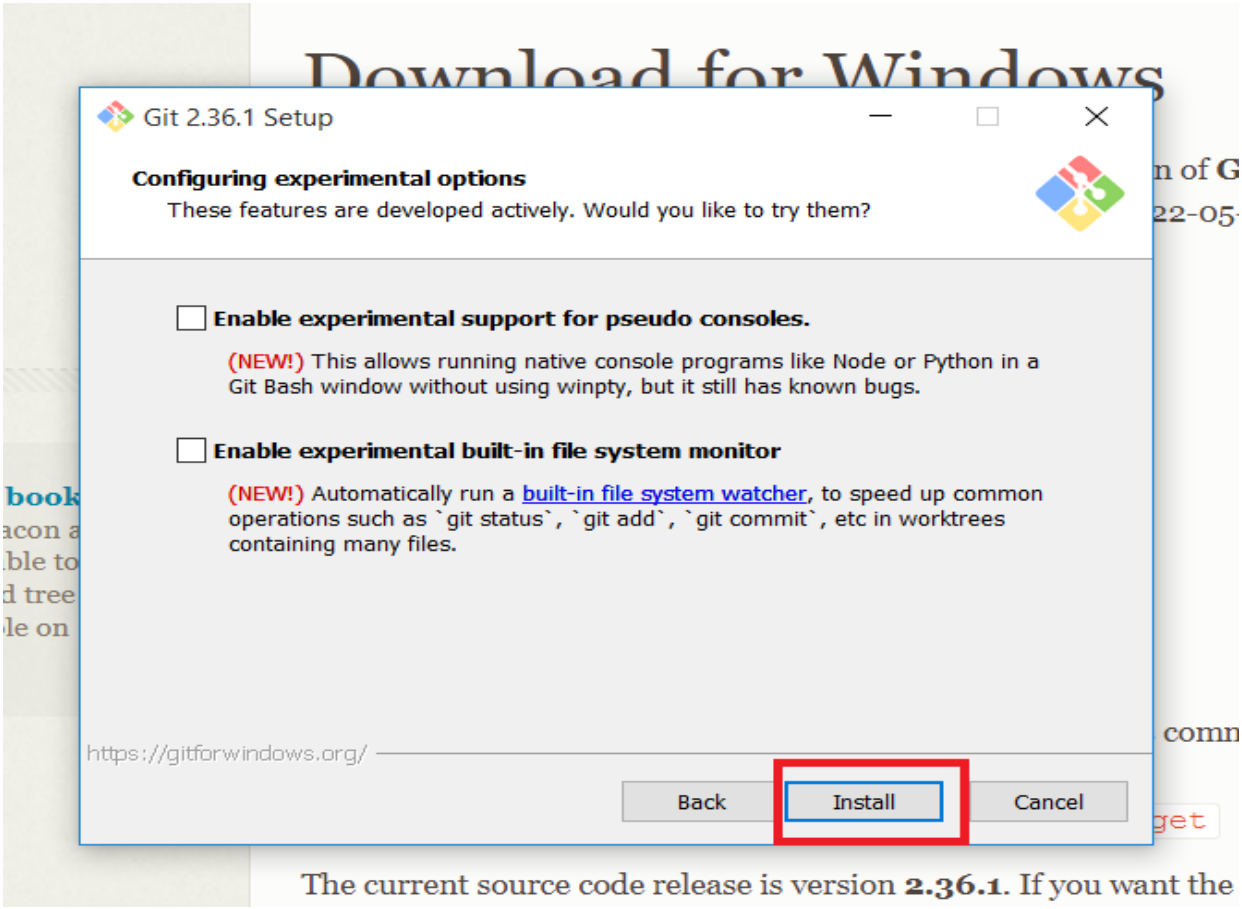
Step 16: The next step is to decide which credential helper to employ. Credential helpers are used by Git to retrieve or save credentials. Leave the default selection and click **Next**.



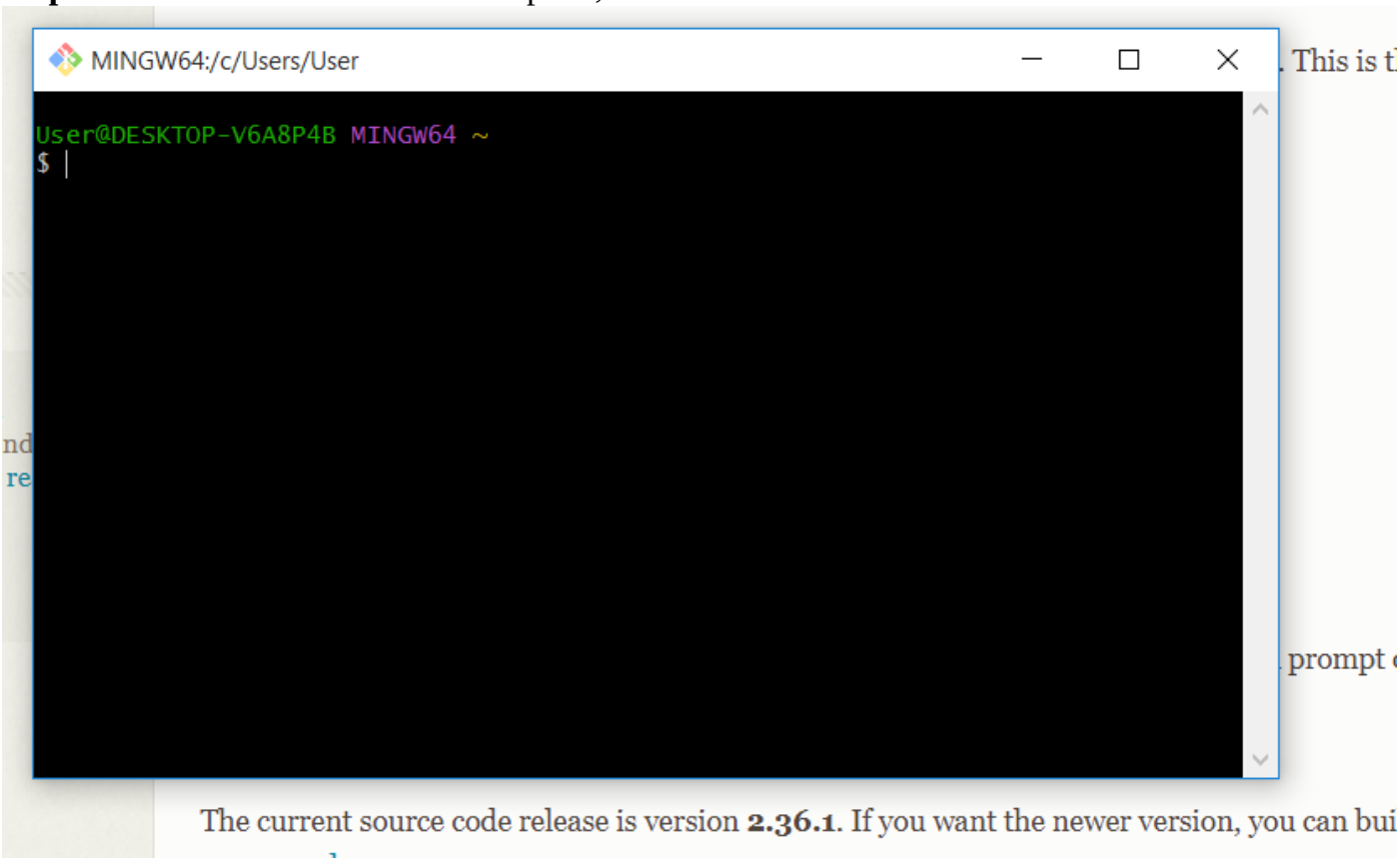
Step 17: Although the default choices are suggested, this step allows you to select which additional features to activate.



Step 18: Git offers to install some experimental features. Leave them unchecked and click **Install**.



Step 19: Once the installation is complete, launch the Git bash.



Command to **create git repository** in folder and store files and track changes.

- Check the version of Git.

```
SSPL-LP-DNS-YTO+Simplilearn@SSPL-LP-DNS-YTO1 MINGW64 ~
$ git --version
git version 2.18.0.windows.1
```

- Create a “week1” repository in the local system.
- Move to the week1 repository.

```
MINGW64:/f/Anusha/week1

Admin@DESKTOP-T3DAU1H MINGW64 ~ (master)
$ pwd
/c/Users/Admin

Admin@DESKTOP-T3DAU1H MINGW64 ~ (master)
$ cd "F:\Anusha\week1"
```

- Create a new git instance for a project.

1.git init

- The command git init is used to create an empty Git repository.
- After the git init command is used, a .git folder is created in the directory with some subdirectories. Once the repository is initialized, the process of creating other files begins.

```
Admin@DESKTOP-T3DAU1H MINGW64 /f/Anusha/week1
$ git init
Initialized empty Git repository in F:/Anusha/week1/.git/
```

- Set up global config variables with github account username and email- If you are working with other developers, you need to know who is checking the code in and out, and to make the changes.

2.git config

- The gitconfig command is used initially to configure the user.name and user.email. This specifies what email id and username will be used from a local repository.
- When gitconfig is used with --global flag, it writes the settings to all repositories on the computer.

git config –global user.name “user name”

git config –global user.email “email id”

```
Admin@DESKTOP-T3DAU1H MINGW64 /f/Anusha/week1 (master)
$ git config --global user.name "devisar"

Admin@DESKTOP-T3DAU1H MINGW64 /f/Anusha/week1 (master)
$ git config --global user.email "anupenugonda1998@cmritonline.ac.in"
```

MINGW64:/f/Anusha/week1

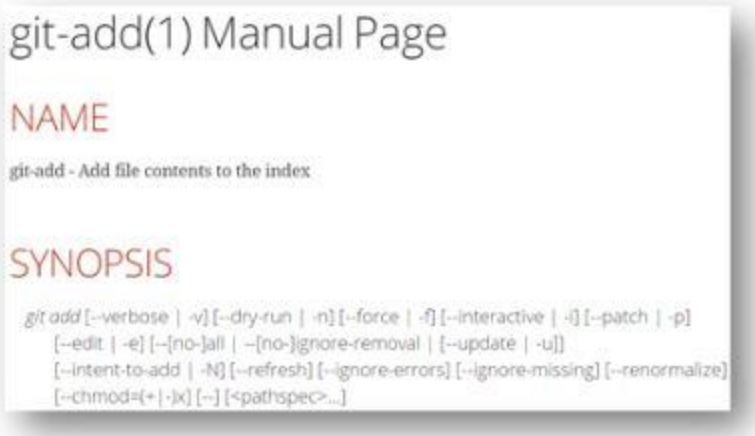
```
Admin@DESKTOP-T3DAU1H MINGW64 /f/Anusha/week1 (master)
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/etc/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
user.name=devisar
user.email=anupenugonda1998@cmritonline.ac.in
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true
```

3. git help

- If in case you need help, use the following commands:

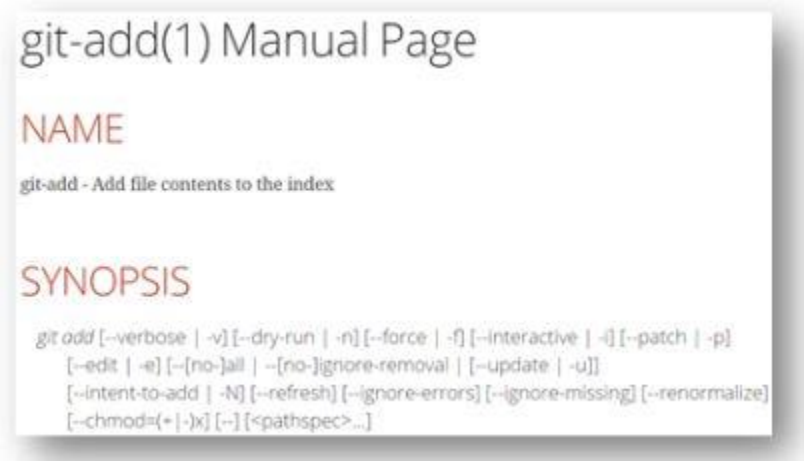
```
SSPL-LP-DNS-YTO+Simplilearn@SSPL-LP-DNS-YTO1 MINGW64 ~
$ git help config

SSPL-LP-DNS-YTO+Simplilearn@SSPL-LP-DNS-YTO1 MINGW64 ~
$ git config --help
```

This will lead you to the Git help page on the browser, which will display the following:

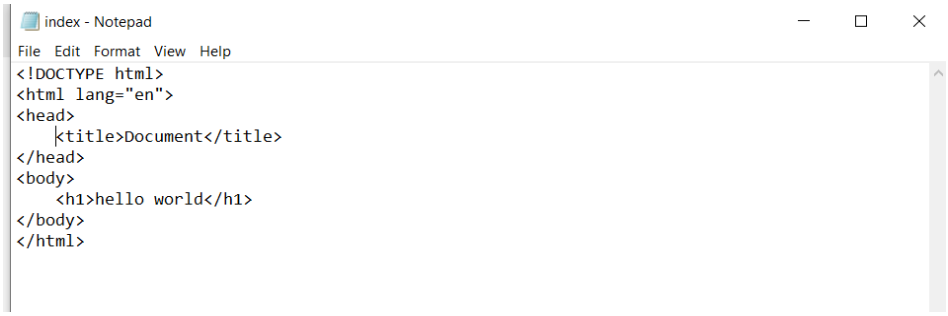



```
SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~
$ git help add
```

This will lead you to the Git help page on the browser, which will display the following:



- Create a file called index.html in the week1 folder; write something and save it.



4. Common Git Commands

GIT Repository

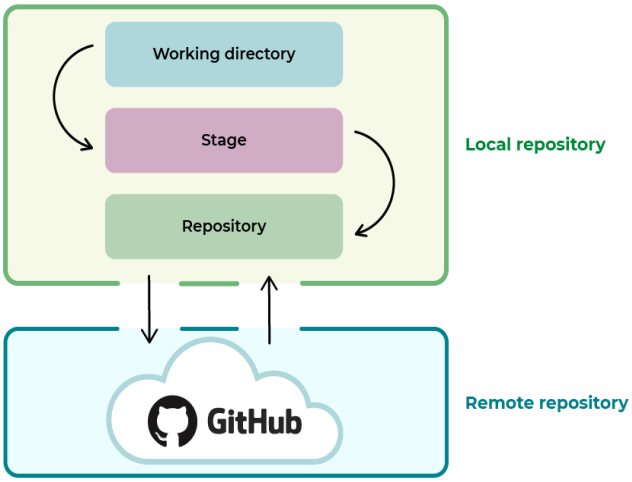
Repositories in GIT contain a collection of files of various different versions of a Project. These files are imported from the repository into the local server of the user for further updations and modifications in the content of the file. A VCS or the Version Control System is used to create these versions and store them in a specific place termed as a repository.

Repositories in Git are of two types:

Local Repository: Git allows the users to perform work on a project from all over the world because of its Distributive feature. This can be done by cloning the content from the Central repository stored in the GitHub on the user's local machine. This local copy is used to perform operations and test them on the local machine before adding them to the central repository.

Remote Repository: Git allows the users to sync their copy of the local repository to other repositories present over the internet. This can be done to avoid performing a similar operation by multiple developers. Each repository in Git can be addressed by a shortcut called remote.

Git provides tools to perform work on these repositories according to the needs of the user. This workflow of performing modifications to a Repository is referred to as the Working Tree.



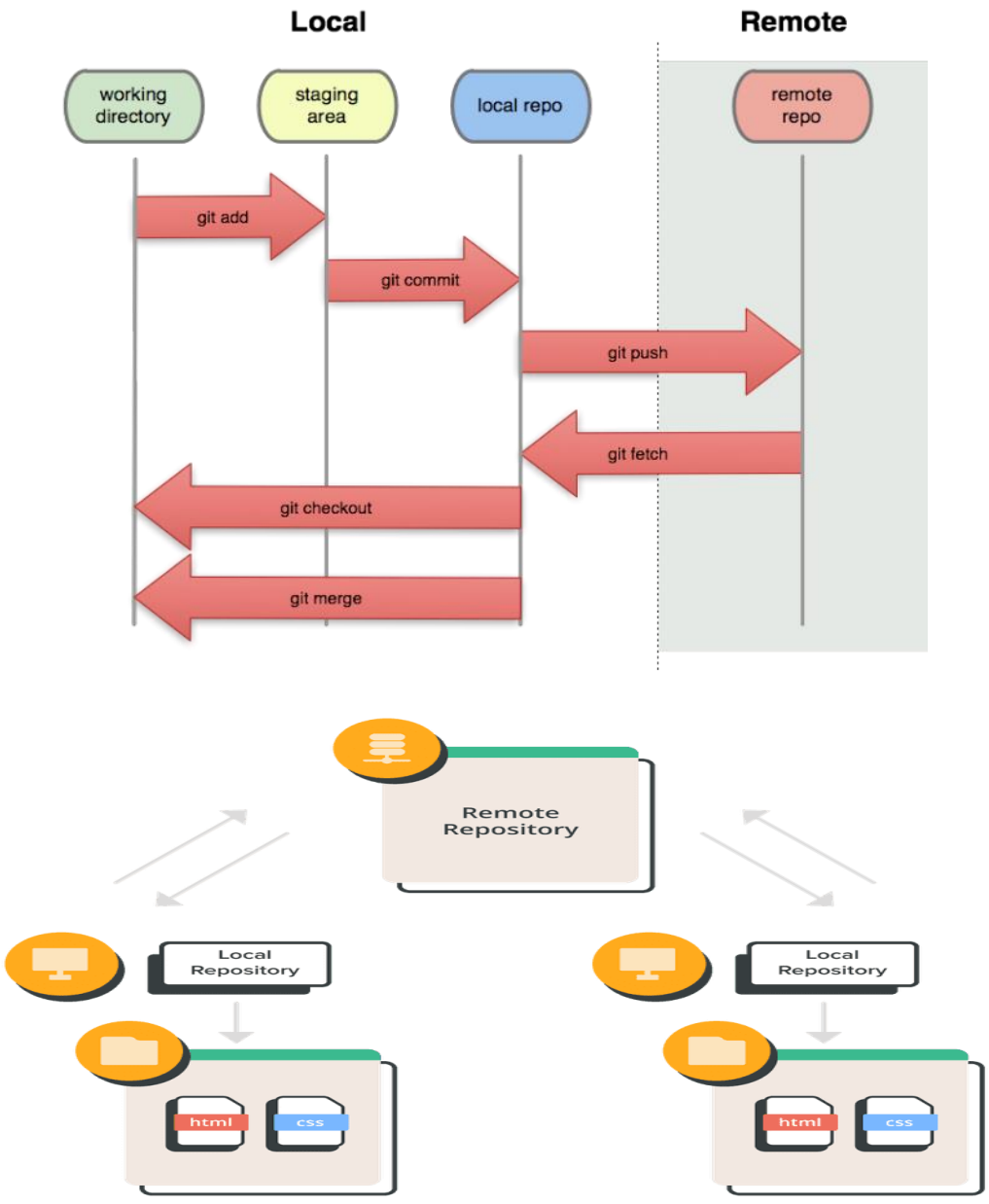


Fig.Working Flow of Local Repository and Remote Repository

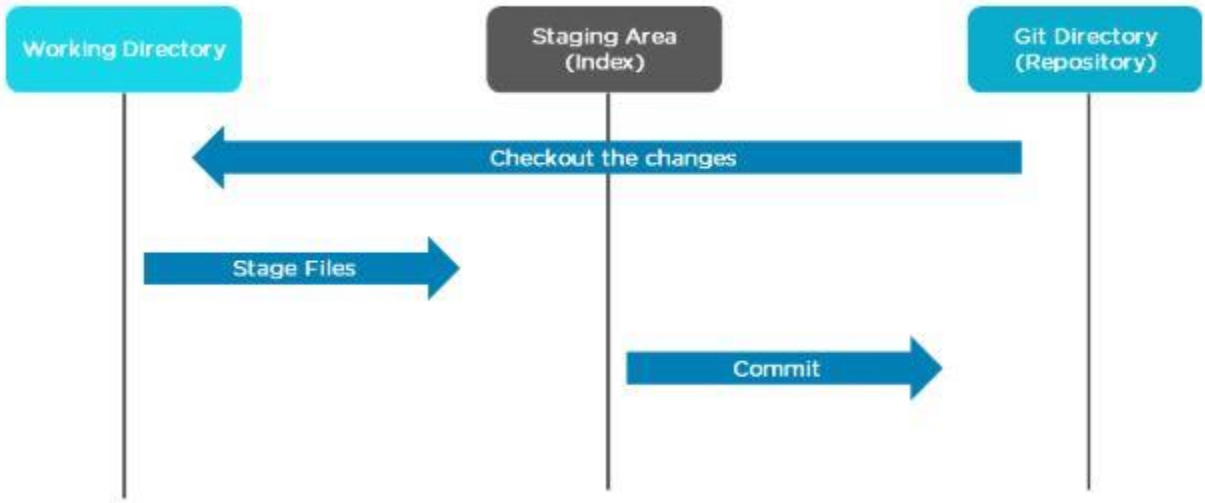
Working with A Local Repository

Git allows the users to perform work on a project from all over the world because of its Distributive feature. This can be done by cloning the content from the Central repository stored in the GitHub on the user's local machine. This local copy is used to perform operations and test them on the local machine before adding them to the central repository.

The Git workflow is divided into three states:

- Working directory - Modify files in your working directory

- Staging area (Index) - Stage the files and add snapshots of them to your staging area
- Git directory (Local Repository) - Perform a commit that stores the snapshots permanently to your Git directory. Checkout any existing version, make changes, stage them and commit.



Command to **create git repository** in folder and store files and track changes.

- Check the version of Git.

```
SSPL-LP-DNS-YTO+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~
$ git --version
git version 2.18.0.windows.1
```

- Create a “week1” repository in the local system.
- Move to the week1 repository.

```
MINGW64:/f/Anusha/week1

Admin@DESKTOP-T3DAU1H MINGW64 ~ (master)
$ pwd
/c/Users/Admin

Admin@DESKTOP-T3DAU1H MINGW64 ~ (master)
$ cd "F:\Anusha\week1"
```

- Create a new git instance for a project.

1.gitinit

- The command git init is used to create an empty Git repository.

- After the git init command is used, a .git folder is created in the directory with some subdirectories. Once the repository is initialized, the process of creating other files begins.

```
Admin@DESKTOP-T3DAU1H MINGW64 /f/Anusha/week1
$ git init
Initialized empty Git repository in F:/Anusha/week1/.git/
```

- Set up global config variables with github account username and email- If you are working with other developers, you need to know who is checking the code in and out, and to make the changes.

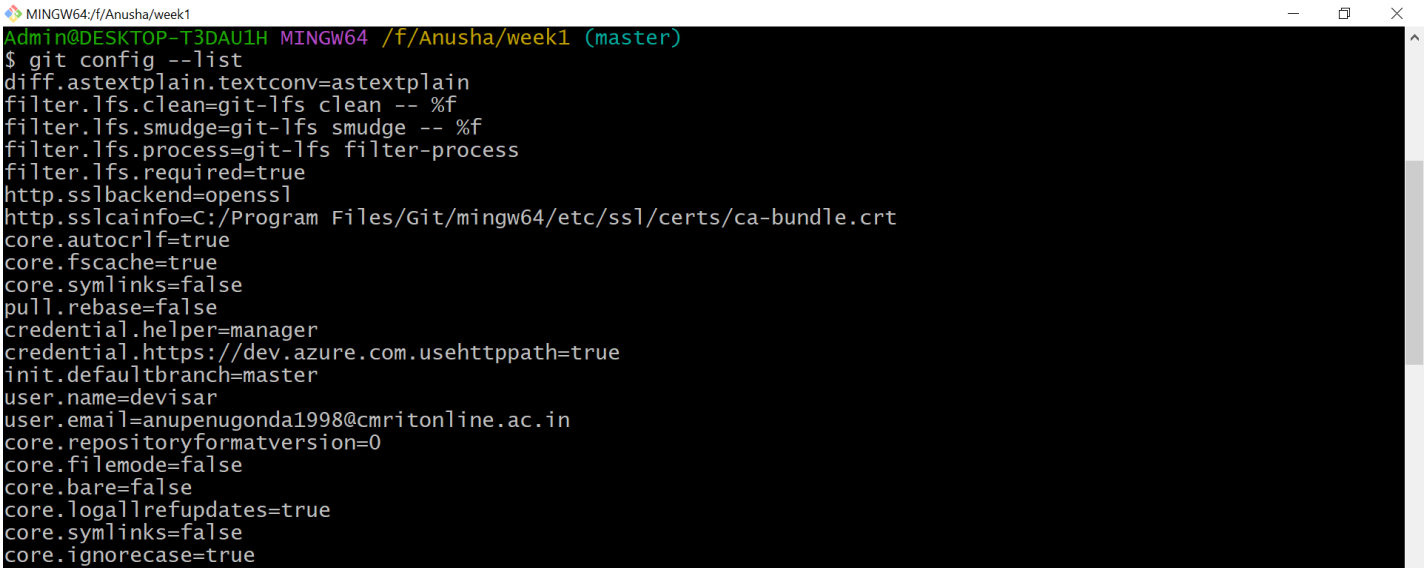
2.gitconfig

- The gitconfig command is used initially to configure the user.name and user.email. This specifies what email id and username will be used from a local repository.
- When gitconfig is used with --global flag, it writes the settings to all repositories on the computer.

gitconfig –global user.name “user name”

gitconfig –global user.email “email id”

```
Admin@DESKTOP-T3DAU1H MINGW64 /f/Anusha/week1 (master)
$ git config --global user.name "devisar"

Admin@DESKTOP-T3DAU1H MINGW64 /f/Anusha/week1 (master)
$ git config --global user.email "anupenugonda1998@cmritonline.ac.in"
```

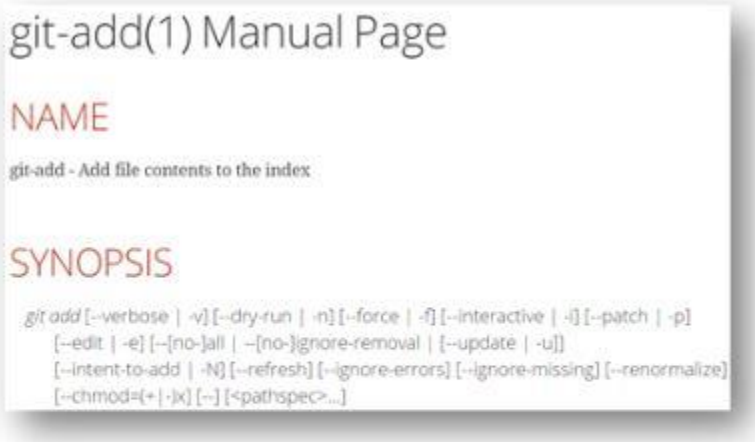


3. git help

- If in case you need help, use the following commands:

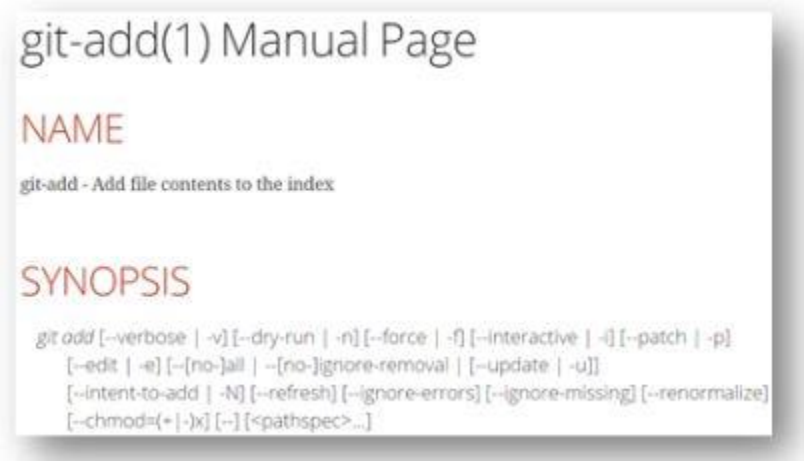
```
SSPL-LP-DNS-YTO+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~
$ git help config

SSPL-LP-DNS-YTO+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~
$ git config --help
```

This will lead you to the Git help page on the browser, which will display the following:

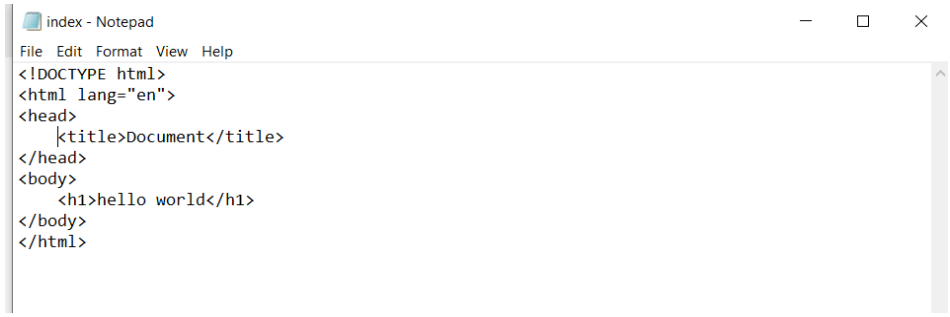



```
SSPL-LP-DNS-YTO+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~
$ git help add
```

This will lead you to the Git help page on the browser, which will display the following:



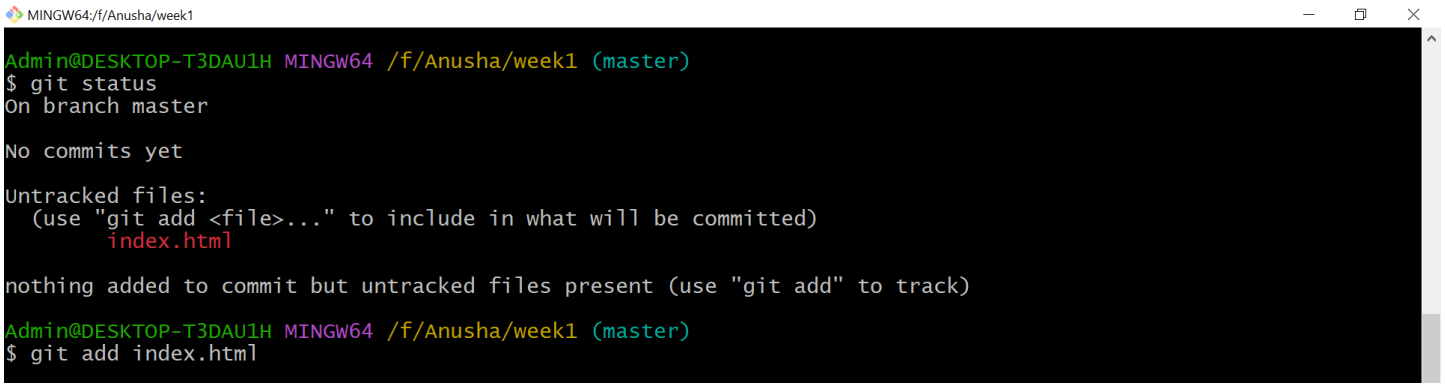
- Create a file called index.html in the week1 folder; write something and save it.



3.git add

- Add command is used after checking the status of the files, to add those files to the staging area.
- Before running the commit command, "git add" is used to add any new or modified files.

git add . (or) git add file name



4.git commit

- The commit command makes sure that the changes are saved to the local repository.
- The command "git commit –m <message>" allows you to describe everyone and help them understand what has happened.

git commit -m “commit message”

```
Admin@DESKTOP-T3DAU1H MINGW64 /f/Anusha/week1 (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   index.html


Admin@DESKTOP-T3DAU1H MINGW64 /f/Anusha/week1 (master)
$ git commit -m "index file commiting"
[master (root-commit) f7aecd2] index file commiting
 1 file changed, 19 insertions(+)
 create mode 100644 index.html
```

5.git status

- The git status command tells the current state of the repository.
- The command provides the current working branch. If the files are in the staging area, but not committed, it will be shown by the git status. Also, if there are no changes, it will show the message no changes to commit, working directory clean.

```
Admin@DESKTOP-T3DAU1H MINGW64 /f/Anusha/week1 (master)
$ git status
On branch master
nothing to commit, working tree clean
```

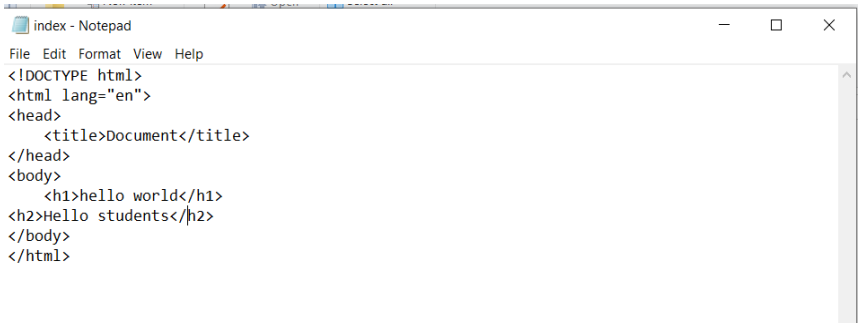
6.git log

- The git log command shows the order of the commit history for a repository.
- The command helps in understanding the state of the current branch by showing the commits that lead to this state.

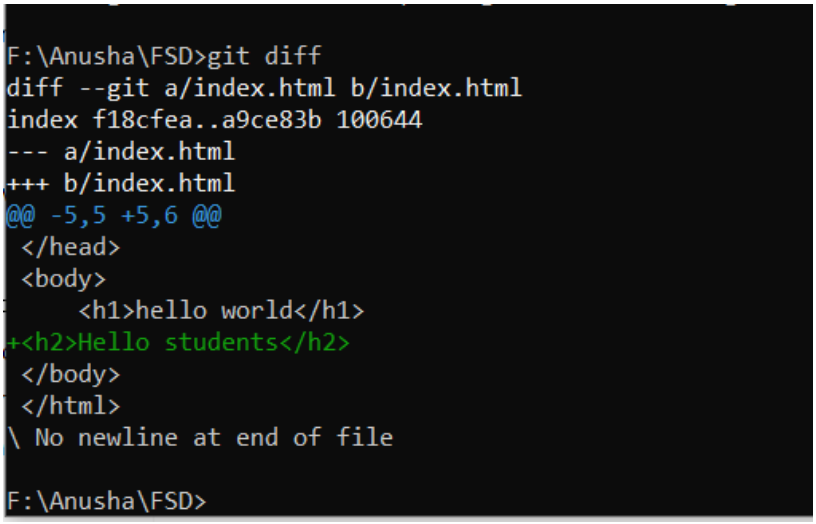
```
Admin@DESKTOP-T3DAU1H MINGW64 /f/Anusha/week1 (master)
$ git log
commit f7aecd23caba4bec5f2adda3ca163fe5e3f78581 (HEAD -> master)
Author: devisar <anupenugonda1998@cmritonline.ac.in>
Date:   Tue Mar 12 19:17:44 2024 +0530

    index file commiting
```

- **7.git diff**

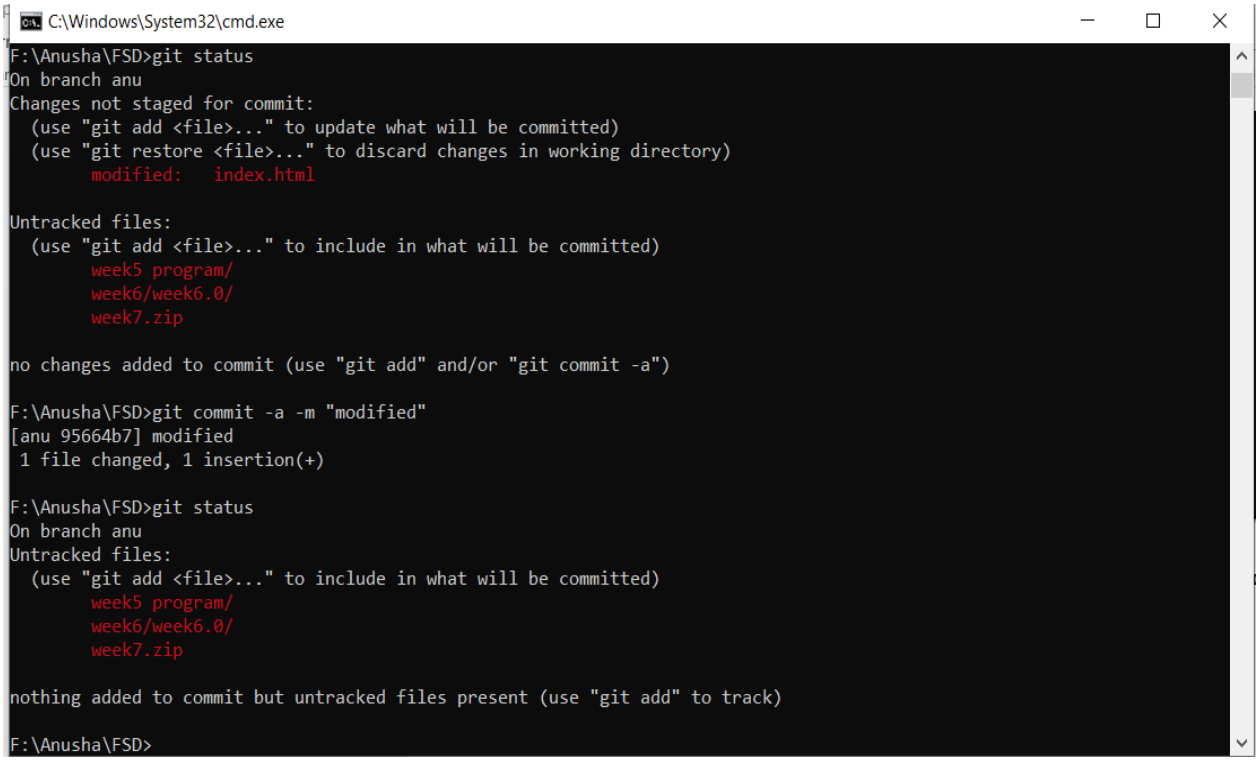
- Make any necessary changes to the file and save.



- Now that you've made changes to the file, you can compare the differences since your last commit.
- Use command **git diff**



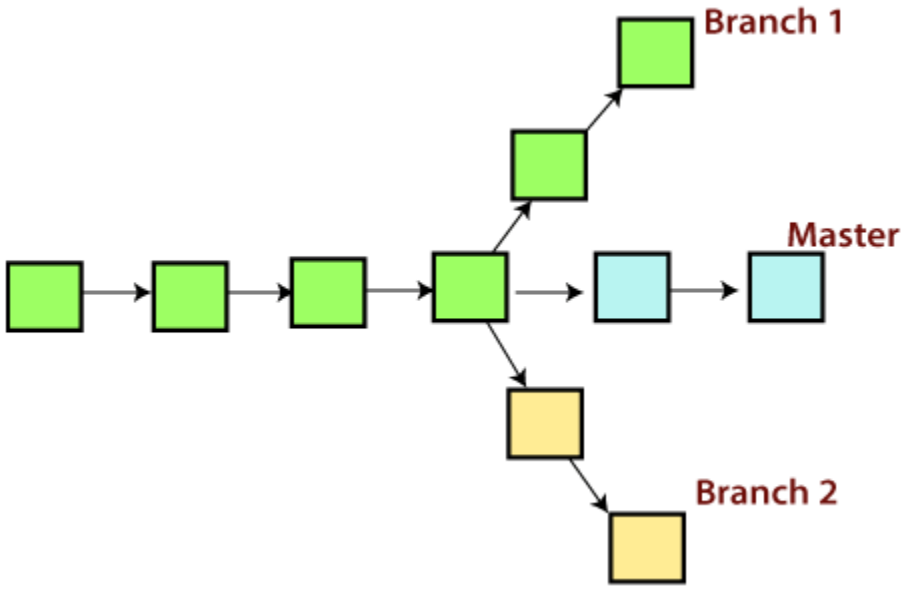
- Save modified data to git repository using command **git commit -a -m "modified"**



5. Branching and Merging Strategies

Git Branch

A branch is a version of the repository that diverges from the main working project. It is a feature available in most modern version control systems. A Git project can have more than one branch. These branches are a pointer to a snapshot of your changes. When you want to add a new feature or fix a bug, you spawn a new branch to summarize your changes. So, it is complex to merge the unstable code with the main code base and also facilitates you to clean up your future history before merging with the main branch.



Git Master Branch

The master branch is a default branch in Git. It is instantiated when first commit made on the project. When you make the first commit, you're given a master branch to the starting commit point. When you start making a commit, then master branch pointer automatically moves forward. A repository can have only one master branch.

Master branch is the branch in which all the changes eventually get merged back. It can be called as an official working version of your project.

Operations on Branches

We can perform various operations on Git branches. The **git branch command** allows you to **create**, **list**, **rename** and **delete** branches. Many operations on branches are applied by git checkout and git merge command. So, the git branch is tightly integrated with the **git checkout** and **git merge commands**.

The Operations that can be performed on a branch:

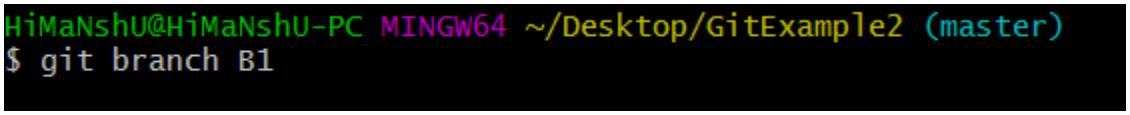
Create Branch

You can create a new branch with the help of the **git branch** command. This command will be used as:

Syntax:

1. $ git branch <**branch** name>

Output:



This command will create the **branch B1** locally in Git directory.

List Branch

You can List all of the available branches in your repository by using the following command.

Either we can use **git branch - list** or **git branch** command to list the available branches in the repository.

Syntax:

1. $ git branch --list

or

1. $ git branch

Output:

Here, both commands are listing the available branches in the repository. The symbol * is representing currently active branch.

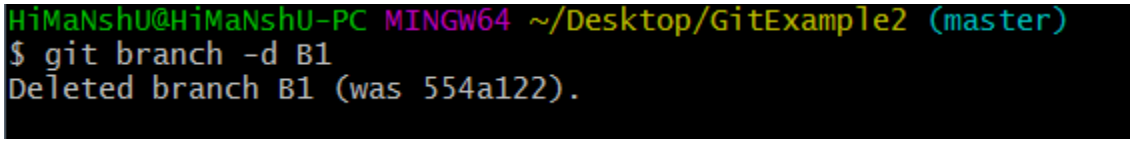
Delete Branch

You can delete the specified branch. It is a safe operation. In this command, Git prevents you from deleting the branch if it has unmerged changes. Below is the command to do this.

Syntax:

1. $ git branch -d<**branch** name>

Output:



This command will delete the existing branch B1 from the repository.

The **git branch d** command can be used in two formats. Another format of this command is **git branch D**. The '**git branch D**' command is used to delete the specified branch.

1. $ git branch -D <**branch** name>

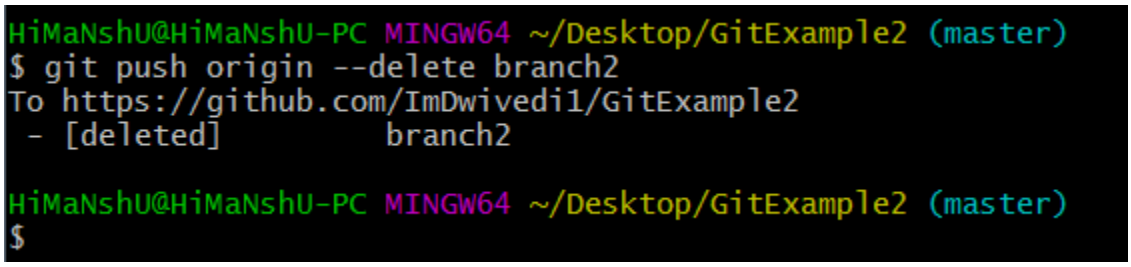
Delete a Remote Branch

You can delete a remote branch from Git desktop application. Below command is used to delete a remote branch:

Syntax:

1. $ git push origin -delete <**branch** name>

Output:



As you can see in the above output, the remote branch named **branch2** from my GitHub account is deleted.

Switch Branch

Git allows you to switch between the branches without making a commit. You can switch between two branches with the **git checkout** command. To switch between the branches, below command is used:

1. $ git checkout<**branch** name>

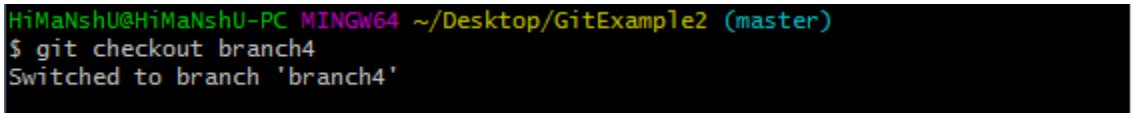
Switch from master Branch

You can switch from master to any other branch available on your repository without making any commit.

Syntax:

1. $ git checkout <**branch** name>

Output:



As you can see in the output, branches are switched from **master** to **branch4** without making any commit.

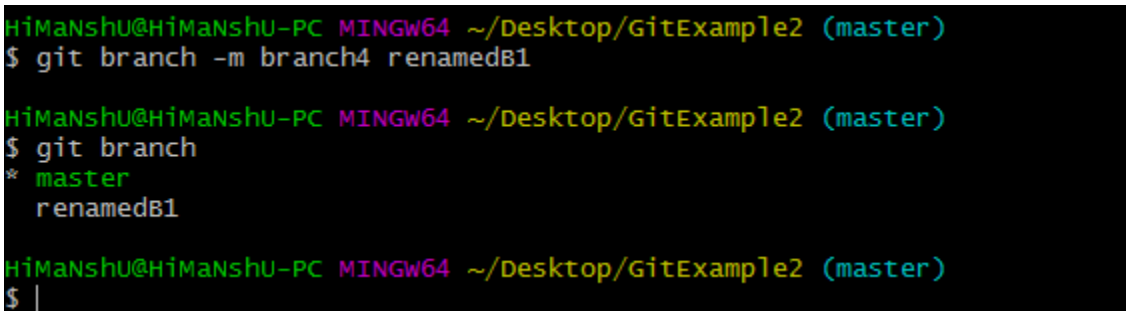
Rename Branch

We can rename the branch with the help of the **git branch** command. To rename a branch, use the below command:

Syntax:

1. $ git branch -m <**old** branch name><**new** branch name>

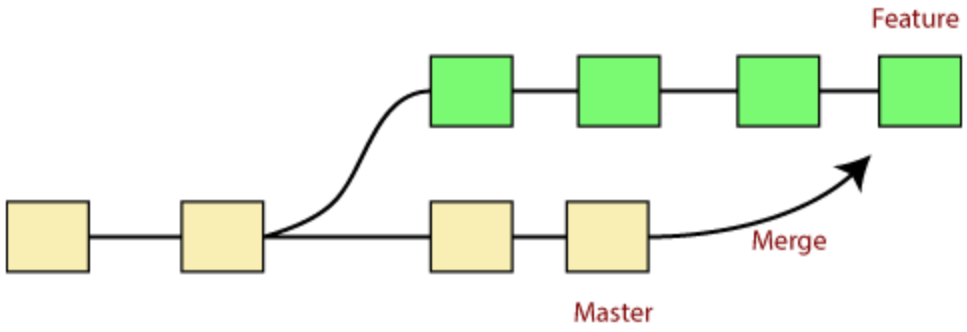
Output:



As you can see in the above output, **branch4** renamed as **renamedB1**.

Git Merge

In Git, the merging is a procedure to connect the forked history. It joins two or more development history together. The git merge command facilitates you to take the data created by git branch and integrate them into a single branch. Git merge will associate a series of commits into one unified history. Generally, git merge is used to combine two branches.



It is used to maintain distinct lines of development; at some stage, you want to merge the changes in one branch. It is essential to understand how merging works in Git.

In the above figure, there are two branches **master** and **feature**. We can see that we made some commits in both functionality and master branch, and merge them. It works as a pointer. It will find a common base commit between branches. Once Git finds a shared base commit, it will create a new "merge commit." It combines the changes of each queued merge commit sequence.

The "git merge" command

The git merge command is used to merge the branches.

The syntax for the git merge command is as:

1. $ git merge <**query**>

It can be used in various context. Some are as follows:

Scenario1: To merge the specified commit to currently active branch:

Use the below command to merge the specified commit to currently active branch.

1. $ git merge <**commit**>

The above command will merge the specified commit to the currently active branch. You can also merge the specified commit to a specified branch by passing in the branch name in <commit>. Let's see how to commit to a currently active branch.

See the below example. I have made some changes in my project's file **newfile1.txt** and committed it in my **test** branch.

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (test)
$ git add newfile1.txt

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (test)
$ git commit -m "edited newfile1.txt"
[test d2bb07d] edited newfile1.txt
 1 file changed, 1 insertion(+), 1 deletion(-)

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (test)
$ git log
commit d2bb07dc9352e194b13075dcfd28e4de802c070b (HEAD -> test)
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Wed Sep 25 11:27:44 2019 +0530

    edited newfile1.txt

commit 2852e020909dfe705707695fd6d715cd723f9540 (test2, master)
Author: ImDwivedi1 <himanshudubey481@gmail.com>
Date:   Wed Sep 25 10:29:07 2019 +0530

    newfile1 added
```

Copy the particular commit you want to merge on an active branch and perform the merge operation. See the below output:

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (test)
$ git checkout test2
Switched to branch 'test2'

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (test2)
$ git merge  d2bb07dc9352e194b13075dcfd28e4de802c070b
Updating 2852e02..d2bb07d
Fast-forward
 newfile1.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (test2)
$
```

In the above output, we have merged the previous commit in the active branch test2.

Merge Branch

Git allows you to merge the other branch with the currently active branch. You can merge two branches with the help of **git merge** command. Below command is used to merge the branches:

Syntax:

1. $ git merge <**branch** name>

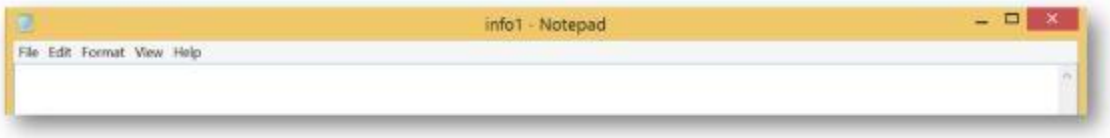
Output:

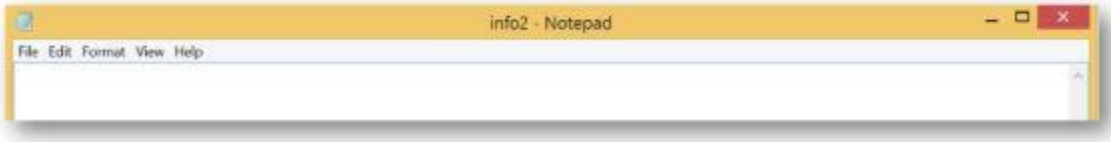
```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git merge renamedB1
Already up to date.

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$
```

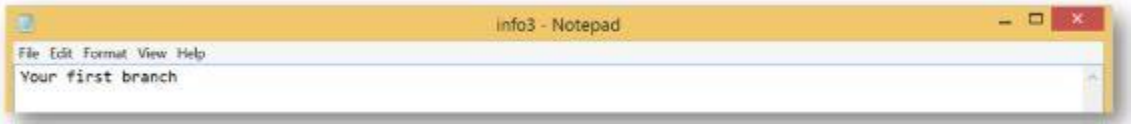
From the above output, you can see that **the master** branch **merged** with **renamedB1**. Since I have made no-commit before merging, so the output is showing as already up to date.

Example:-

- Create three more text files in the local repository - “info1.txt”, “info2.txt”, “info3.txt”.







- Create a branch “first_branch” and merge it to the main (master) branch.

```
SSPL-LP-DNS-YTO+Simplilearn@SSPL-LP-DNS-YTO1 MINGW64 ~/test (master)
$ git branch first_branch
```

The above command creates a branch.

```
SSPL-LP-DNS-YTO+Simplilearn@SSPL-LP-DNS-YTO1 MINGW64 ~/test (master)
$ git checkout first_branch
Switched to branch 'first_branch'
M       info.txt
```

The above command switches to the new branch from the master branch.

```
SSPL-LP-DNS-YTO+Simplilearn@SSPL-LP-DNS-YTO1 MINGW64 ~/test (first_branch)
$ git add info3.txt
```

The above command creates and adds “info3.txt” to the first_branch.

```
SSPL-LP-DNS-YTO+Simplilearn@SSPL-LP-DNS-YTO1 MINGW64 ~/test (first_branch)
$ git commit -m "make some changes to first_branch"
[first_branch b14d609] make some changes to first_branch
 Committer: Simplilearn <Simplilearn@SSPL-LP-DNS-YTO1.blrsimplilearn.com>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

    git config --global user.name "Your Name"
    git config --global user.email you@example.com

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 1 file changed, 1 insertion(+)
 create mode 100644 info3.txt
```

The above command makes a commit to the first_branch.

```
SSPL-LP-DNS-YTO+Simplilearn@SSPL-LP-DNS-YTO1 MINGW64 ~/test (first_branch)
$ ls
info.txt  info1.txt  info2.txt  info3.txt
```

The above command shows that the new branch has access to all the files.

```
SSPL-LP-DNS-YTO+Simplilearn@SSPL-LP-DNS-YTO1 MINGW64 ~/test (first_branch)
$ git checkout master
Switched to branch 'master'
M       info.txt
```

```
SSPL-LP-DNS-YTO+Simplilearn@SSPL-LP-DNS-YTO1 MINGW64 ~/test (master)
$ ls
info.txt  info1.txt  info2.txt
```

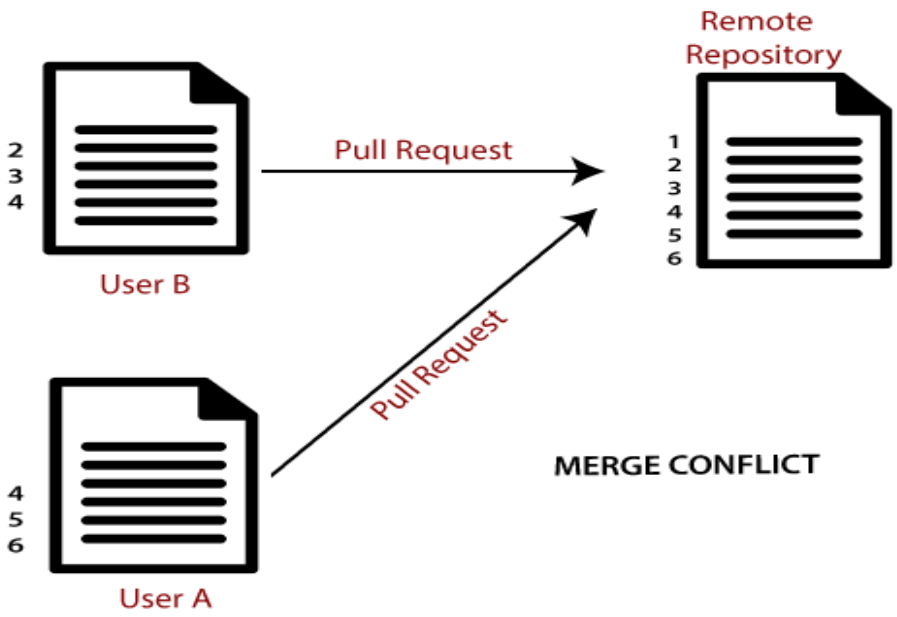
The above command shows that the master branch does not have an “info3.txt” file.

```
SSPL-LP-DNS-YTO+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~/test (master)
$ git merge first_branch
Updating 3ae78fd..b14d609
Fast-forward
 info3.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 info3.txt

SSPL-LP-DNS-YTO+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~/test (master)
$ ls
info.txt  info1.txt  info2.txt  info3.txt
```

The above command is used to merge “first_branch” with the master branch. Now, the master branch has “info3.txt” file.

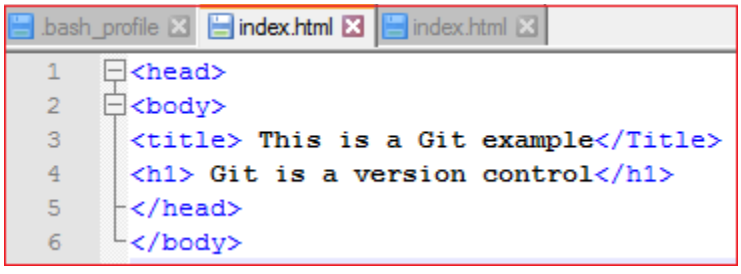
Git Merge Conflict

When two branches are trying to merge, and both are edited at the same time and in the same file, Git won't be able to identify which version is to take for changes. Such a situation is called merge conflict. If such a situation occurs, it stops just before the merge commit so that you can resolve the conflicts manually.



Let's understand it by an example.

Suppose my remote repository has cloned by two of my team member **user1** and **user2**. The user1 made changes as below in my projects index file.



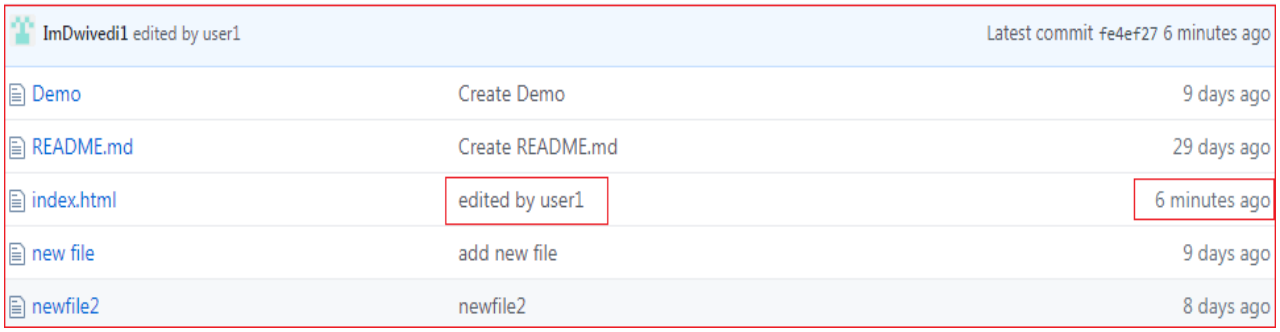
Update it in the local repository with the help of git add command.

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user1repo (master)
$ git add index.html
```

Now commit the changes and update it with the remote repository. See the below output:

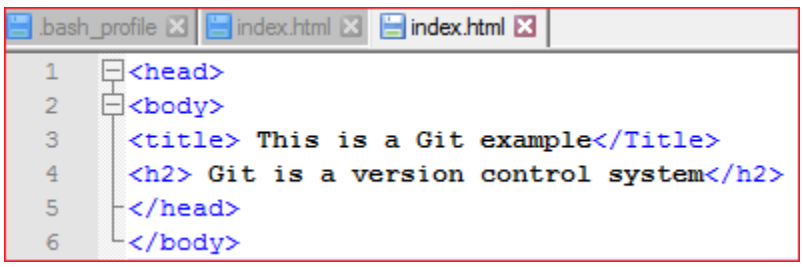
```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user1repo (master)
$ git commit -m "edited by user1"
[master fe4ef27] edited by user1
 1 file changed, 1 insertion(+)

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user1repo (master)
$ git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 2 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 345 bytes | 345.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/ImDwivedi1/Git-Example
   039c01b..fe4ef27  master -> master

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user1repo (master)
```

Now, my remote repository will look like this:



It will show the status of the file like edited by whom and when.

Now, at the same time, **user2** also update the index file as follows.



User2 has added and committed the changes in the local repository. But when he tries to push it to remote server, it will throw errors. See the below output:

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user2repo (master)
$ git add index.html

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user2repo (master)
$ git commit -m " edited by user2"
[master 3ee71e0]  edited by user2
 1 file changed, 1 insertion(+)

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user2repo (master)
$ git push origin master
To https://github.com/ImDwivedi1/Git-Example
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://github.com/ImDwivedi1/Git-Example'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user2repo (master)
$
```

In the above output, the server knows that the file is already updated and not merged with other branches. So, the push request was rejected by the remote server. It will throw an error message like **[rejected] failed to push some refs to <remote URL>**. It will suggest you to pull the repository first before the push. See the below command:

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user2repo (master)
$ git pull --rebase origin master
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/ImDwivedi1/Git-Example
 * branch            master     -> FETCH_HEAD
   039c01b..fe4ef27  master     -> origin/master
First, rewinding head to replay your work on top of it...
Applying:  edited by user2
Using index info to reconstruct a base tree...
M       index.html
Falling back to patching base and 3-way merge...
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
error: Failed to merge in the changes.
hint: Use 'git am --show-current-patch' to see the failed patch
Patch failed at 0001  edited by user2
Resolve all conflicts manually, mark them as resolved with
"git add/rm <conflicted_files>", then run "git rebase --continue".
You can instead skip this commit: run "git rebase --skip".
To abort and get back to the state before "git rebase", run "git rebase --abort"
.

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/user2repo (master|REBASE 1/1)
$
```

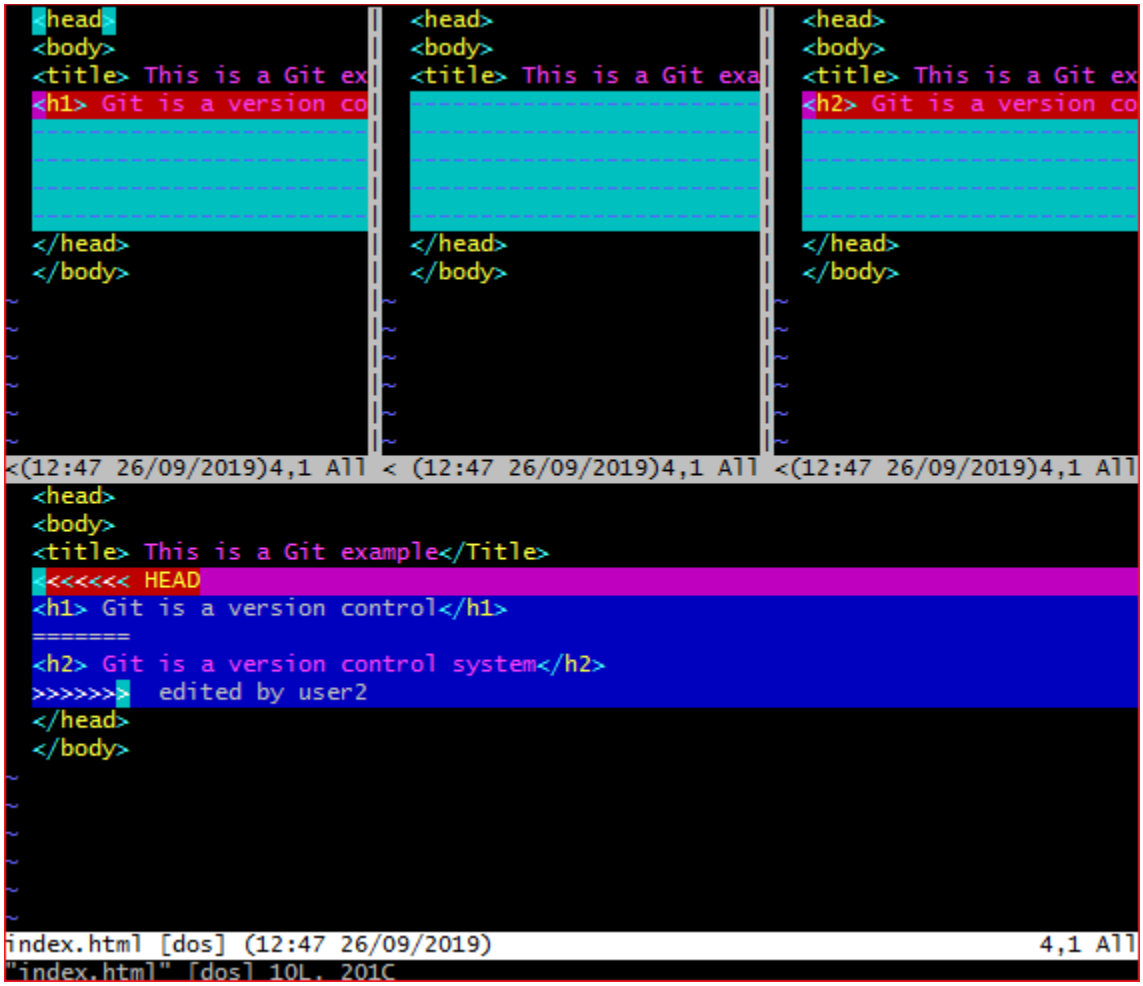
In the given output, git rebase command is used to pull the repository from the remote URL. Here, it will show the error message like **merge conflict in <filename>**.

Resolve Conflict:

To resolve the conflict, it is necessary to know whether the conflict occurs and why it occurs. Git merge tool command is used to resolve the conflict. The merge command is used as follows:

1. $ git mergetool

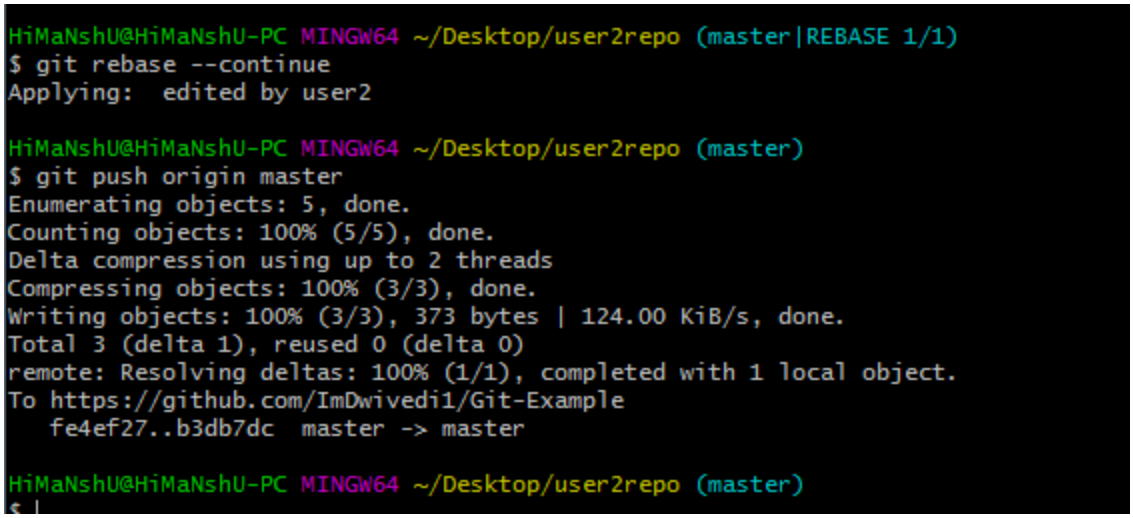
In my repository, it will result in:



The above output shows the status of the conflicted file. To resolve the conflict, enter in the insert mode by merely pressing **I key** and make changes as you want. Press the **Esc key**, to come out from insert mode. Type the: **w!** at the bottom of the editor to save and exit the changes. To accept the changes, use the rebase command. It will be used as follows:

1. $ git rebase --continue

Hence, the conflict has resolved. See the below output:



In the above output, the conflict has resolved, and the local repository is synchronized with a remote repository.

To see that which is the first edited text of the merge conflict in your file, search the file attached with conflict marker <<<<<<<. You can see the changes from the **HEAD** or base branch after the line <<<<<<< **HEAD** in your text editor. Next, you can see the divider like =======. It divides your changes from the changes in the other branch, **followed by** >>>>>>> **BRANCH-NAME**. In the above example, user1 wrote "<h1>Git is a version control</h1>" in the base or HEAD branch and user2 wrote "<h2>Git is a version control</h2>".

Decide whether you want to keep only your branch's changes or the other branch's changes, or create a new change. Delete the conflict markers **<<<<<<<, =======, >>>>>>>** and create final changes you want to merge.

6. Working with Remote Repositories

What is GitHub?

GitHub is a Git repository hosting service. GitHub also facilitates with many of its features, such as access control and collaboration. It provides a Web-based graphical interface.

GitHub is an American company. It hosts source code of your project in the form of different programming languages and keeps track of the various changes made by programmers.

It offers both **distributed version control and source code management (SCM)** functionality of Git. It also facilitates with some collaboration features such as bug tracking, feature requests, task management for every project.

What is a repository in GitHub?

A repository is the most basic element of GitHub. It's a place where you can store your code, your files, and each file's revision history. Repositories can have multiple collaborators and can be either public or private. To create a new repository, go to https://github.com/new.

- # Features of GitHub

GitHub is a place where programmers and designers work together. They collaborate, contribute, and fix bugs together. It hosts plenty of open source projects and codes of various programming languages.

Some of its significant features are as follows.

- Collaboration
- Integrated issue and bug tracking
- Graphical representation of branches
- Git repositories hosting
- Project management
- Team management
- Code hosting
- Track and assign tasks
- Conversations
- **Wikisc**:- Every repository on GitHub.com comes equipped with a section for hosting documentation, called a wiki. You can use your repository's wiki to share long-form content about your project, such as how to use it, how you designed it, or its core principles.

- # Benefits of GitHub

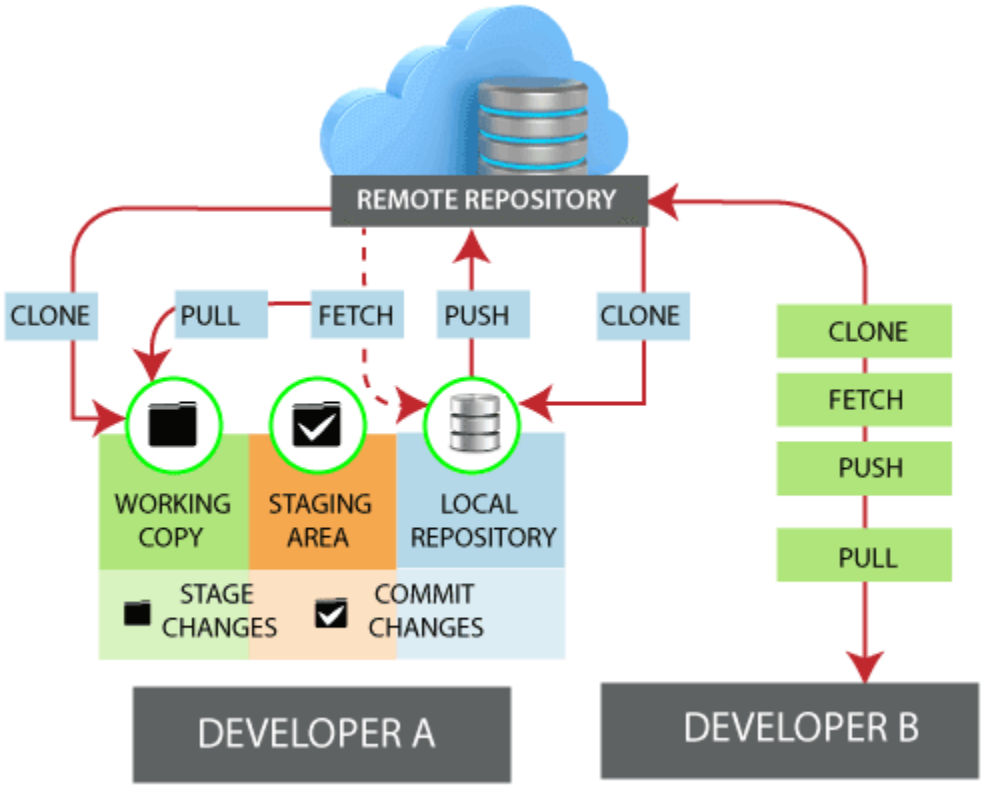
GitHub can be separated as the Git and the Hub. GitHub service includes access controls as well as collaboration features like task management, repository hosting, and team management.

The key benefits of GitHub are as follows.

- It is easy to contribute to open source projects via GitHub.
- It helps to create an excellent document.
- You can attract recruiter by showing off your work. If you have a profile on GitHub, you will have a higher chance of being recruited.
- It allows your work to get out there in front of the public.
- You can track changes in your code across versions.

Working with Remote Repository

The developers can perform many operations with the remote server. These operations can be a clone, fetch, push, pull, and more. Consider the below image:



a. First Create GitHub Account and Login

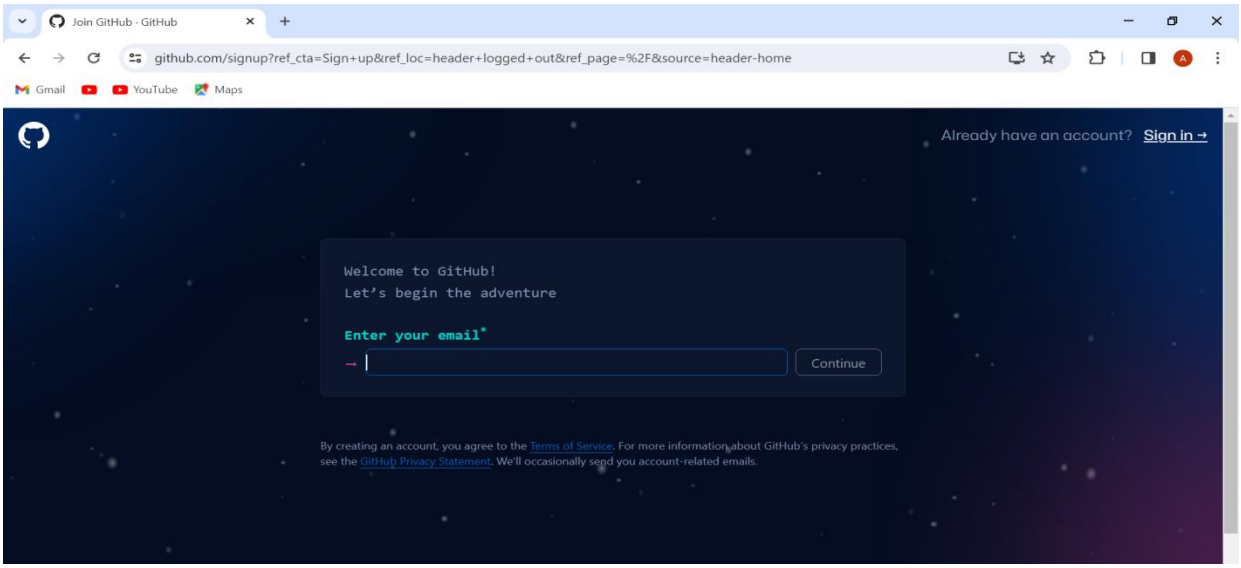


Fig.1.Github sign in and sign up page

b. Next create new Repository name week1.0

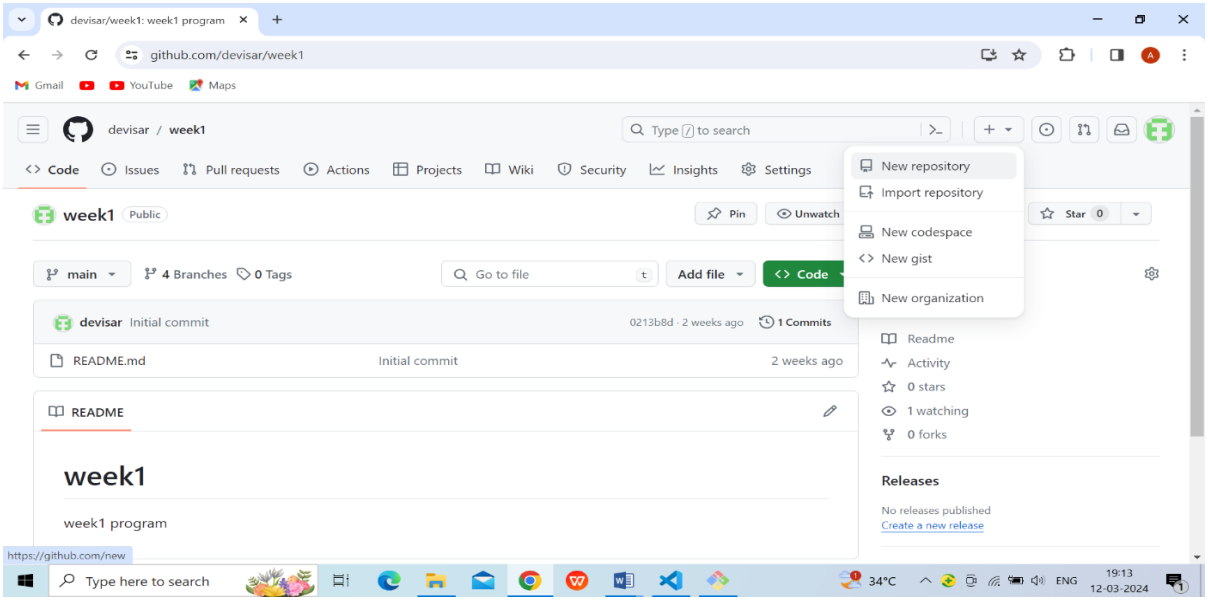


Fig.2.creating new repository page

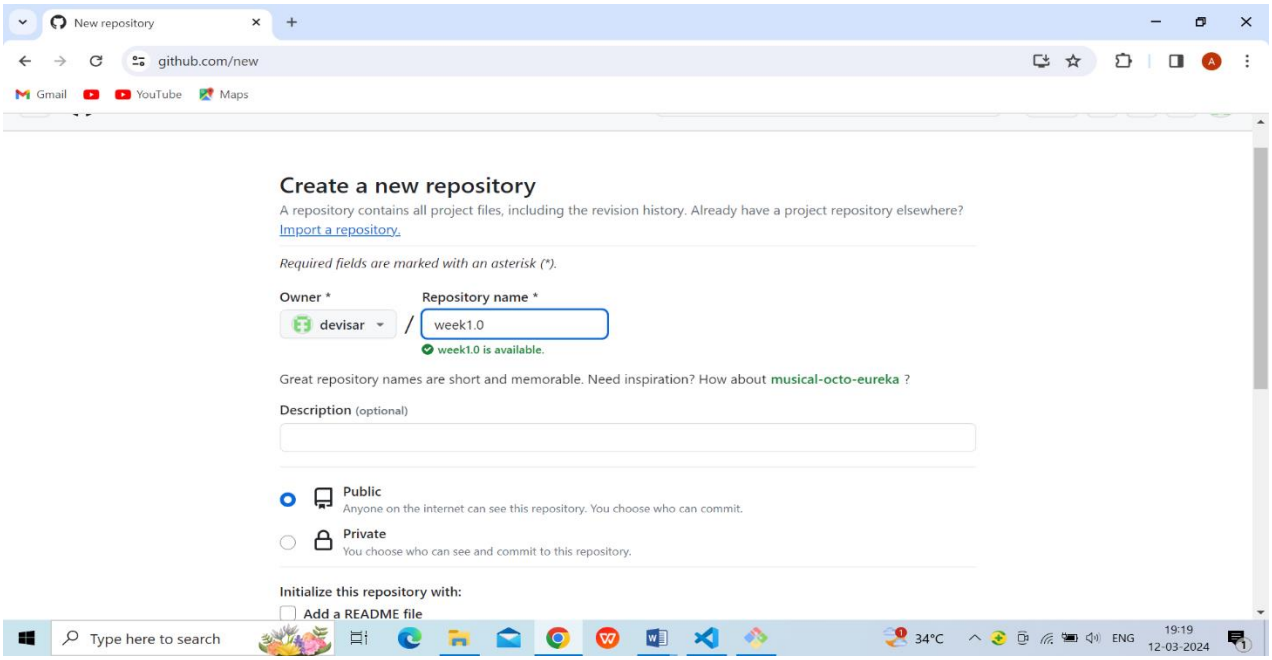


Fig.3.Enter repository name page

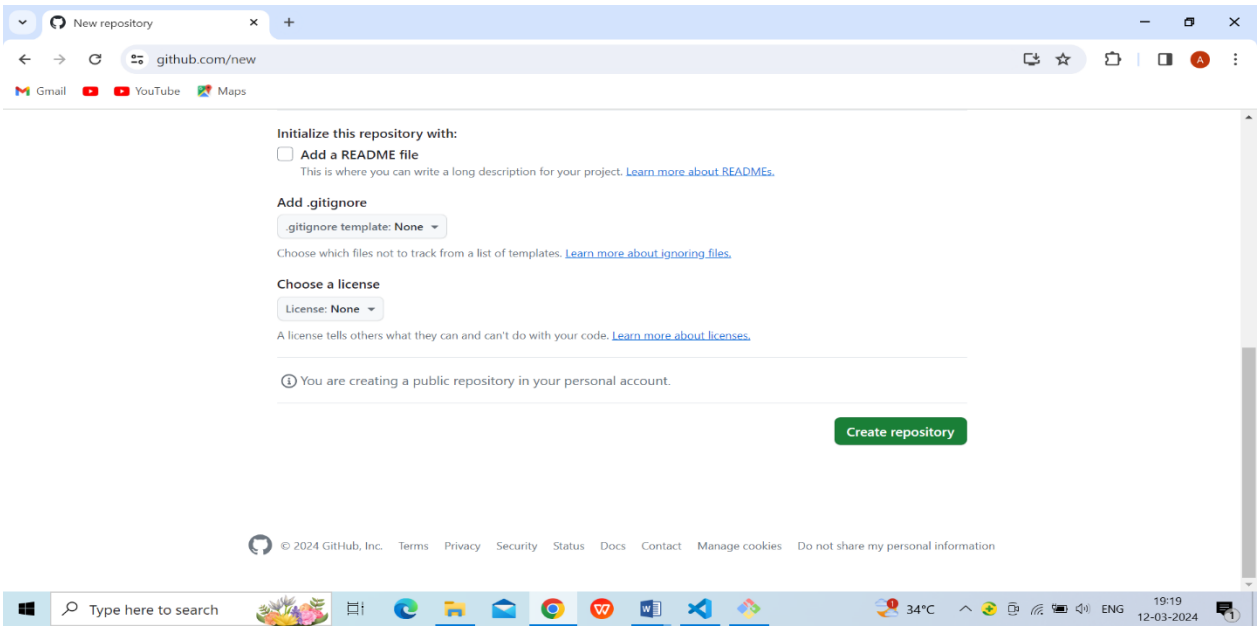


Fig.4.Click button to create repository page

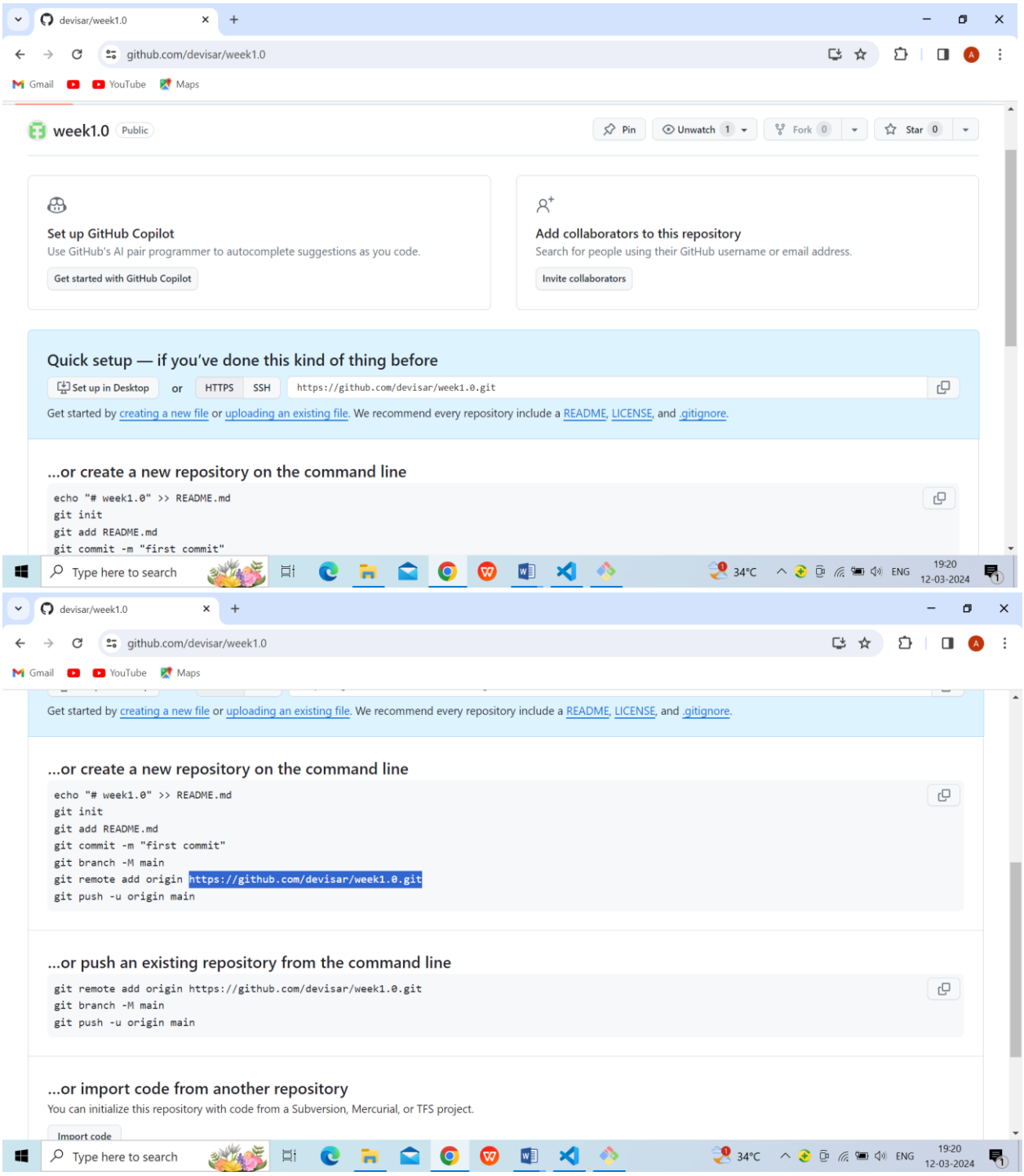


Fig.5.After opening page

- Connect the local repository to your remote repository.

1.git remote

- The git remote command is used to create, view, and delete connections to other repositories.

- The connections here are not like direct links into other repositories, but as bookmarks that serve as convenient names to be used as a reference.

Git remote add origin <address>

```
Admin@DESKTOP-T3DAU1H MINGW64 /f/Anusha/week1 (master)
$ git remote add origin "https://github.com/devisar/week1.0.git"

Admin@DESKTOP-T3DAU1H MINGW64 /f/Anusha/week1 (master)
$ git remote -v
origin  https://github.com/devisar/week1.0.git (fetch)
origin  https://github.com/devisar/week1.0.git (push)
```

- Push the file to the remote repository.

2.git push

- The command git push is used to transfer the commits or pushing the content from the local repository to the remote repository.
- The command is used after a local repository has been modified, and the modifications are to be shared with the remote team members.

git push –u origin master

```
Admin@DESKTOP-T3DAU1H MINGW64 /f/Anusha/week1 (master)
$ git push -u origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 688 bytes | 344.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/devisar/week1.0.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
```

Refresh your repository page on GitHub. You will get your local file on your remote GitHub repository

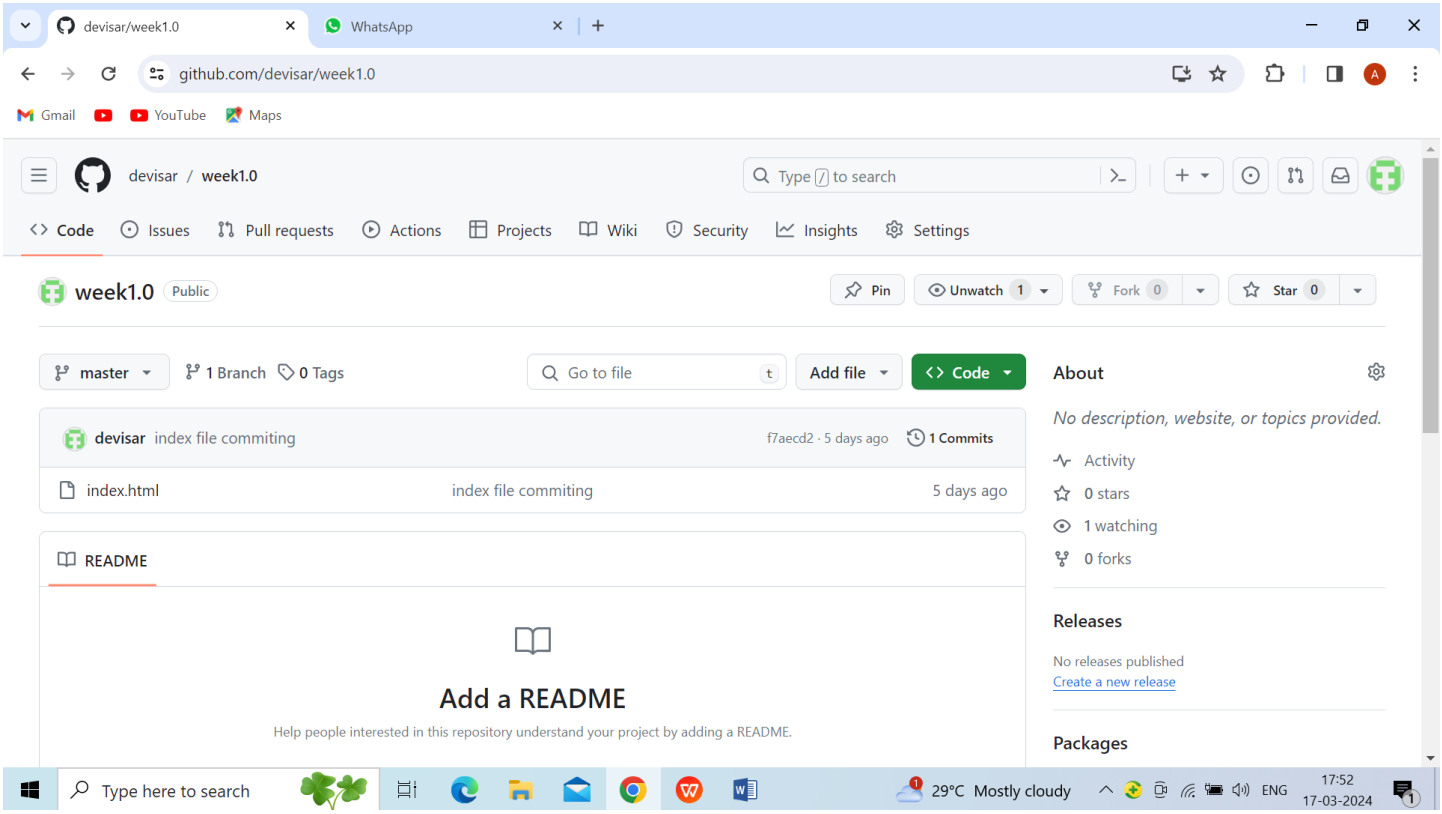


Fig.6.Now File is uploaded

3.remote commands

3.1 Check your Remote

To check the configuration of the remote server, run the **git remote** command. The git remote command allows accessing the connection between remote and local. If you want to see the original existence of your cloned repository, use the git remote command. It can be used as:

git remote

And

git remote -v

```
F:\Anusha\FSD>git remote
origin

F:\Anusha\FSD>git remote -v
origin  https://github.com/devisar/FSD-LAB.git (fetch)
origin  https://github.com/devisar/FSD-LAB.git (push)

F:\Anusha\FSD>
```

3.2 Remove Remote

You can remove a remote connection from a repository. To remove a connection, perform the git remote command with **remove** or **rm** option. It can be done as:

Syntax:

1. $**git remote rm <destination>**

   Or

1. $ git remote remove <destination>

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git remote rm origin

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git remote -v

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$
```

In the above output, I have removed remote server "origin" from my repository.

3.3 Git Remote Rename

Git allows renaming the remote server name so that you can use a short name in place of the remote server name. Below command is used to rename the remote server:

Syntax:

1. $ git remote rename <old name><**new** name>

Output:

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git remote rename origin hd

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git remote -v
hd      https://github.com/ImDwivedi1/GitExample2 (fetch)
hd      https://github.com/ImDwivedi1/GitExample2 (push)
```

In the above output, I have renamed my default server name origin to hd. Now, I can operate using this name in place of origin. Consider the below output:

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git pull hd master
From https://github.com/ImDwivedi1/GitExample2
 * branch            master     -> FETCH_HEAD
 * [new branch]      master     -> hd/master
Already up to date.

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git pull origin  master
fatal: 'origin' does not appear to be a git repository
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
```

In the above output, I have pulled the remote repository using the server name hd. But, when I am using the old server name, it is throwing an error with the message "'**origin' does not appear to be a git repository**." It means Git is not identifying the old name, so all the operations will be performed by a new name.

3.4 Git Change Remote (Changing a Remote's URL)

We can change the URL of a remote repository. The git remote set command is used to change the URL of the repository. It changes an existing remote repository URL.

Git Remote Set:

We can change the remote URL simply by using the git remote set command. Suppose we want to make a unique name for our project to specify it. Git allows us to do so. It is a simple process. To change the remote URL, use the below command:

1. $ **git remote set-url <remote name><newURL>**

The **remote set-url** command takes two types of arguments. The first one is <remote name >, it is your current server name for the repository. The second argument is <newURL>, it is your new URL name for the repository. The <new URL> should be in below format: **https://github.com/URLChanged**

Consider the below image:

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git remote set-url origin  https://github.com/URLChanged

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git remote -v
origin  https://github.com/URLChanged (fetch)
origin  https://github.com/URLChanged (push)

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$
```

In the above output, I have changed my existing repository URL as **https://github.com/URLChanged** from **https://github.com/ImDwivedi1/GitExample2**. It can be understood by my URL name that I have changed this. To check the latest URL, perform the below command:

4.Git Clone Command

The **git clone** is a command-line utility which is used to make a local copy of a remote repository. It accesses the repository through a remote URL.

Usually, the original repository is located on a remote server, often from a Git service like GitHub, Bitbucket, or GitLab. The remote repository URL is referred to the **origin**.

Syntax:

1. $ git clone **<repository** URL**>**

Git Clone Repository

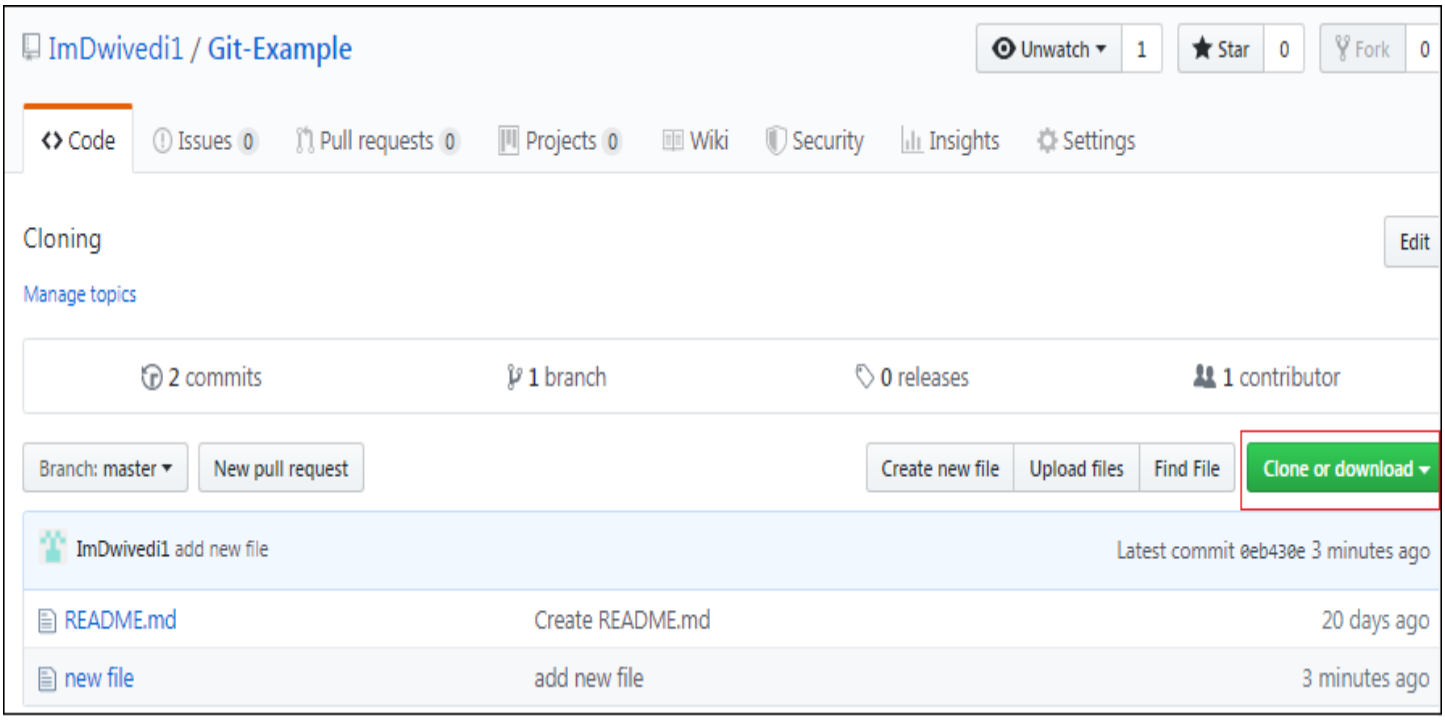
Suppose, you want to clone a repository from GitHub, or have an existing repository owned by any other user you would like to contribute. Steps to clone a repository are as follows:

Step 1:

Open GitHub and navigate to the main page of the repository.

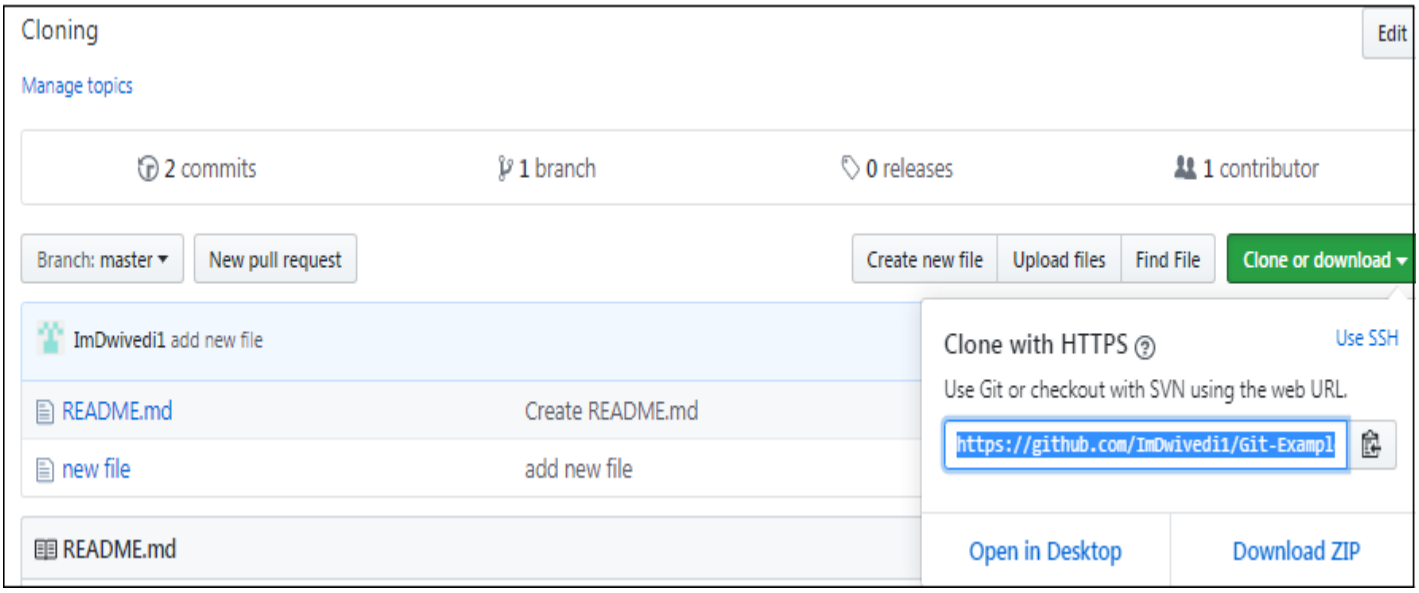
Step 2:

Under the repository name, click on **Clone or download**.



Step 3:

Select the **Clone with HTTPs section** and **copy the clone URL** for the repository. For the empty repository, you can copy the repository page URL from your browser and skip to next step.



Step 4:

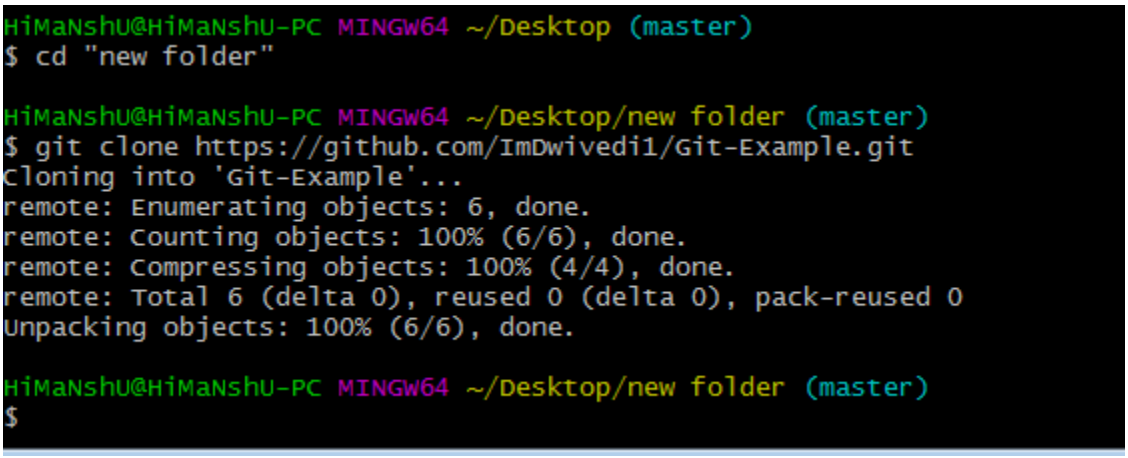
Open Git Bash and change the current working directory to your desired location where you want to create the local copy of the repository.

Step 5:

Use the git clone command with repository URL to make a copy of the remote repository. See the below command:

1. $ git clone https://github.com/ImDwivedi1/Git-Example.git

Now, Press Enter. Hence, your local cloned repository will be created. See the below output:



Git Clone Branch

Git allows making a copy of only a particular branch from a repository. You can make a directory for the individual branch by using the git clone command. To make a clone branch, you need to specify the branch name with -b command. Below is the syntax of the command to clone the specific git branch:

Syntax:

1. $ git clone -b **<Branch** name**><Repository** URL**>**

See the below command:

1. $ git clone -b master https://github.com/ImDwivedi1/Git-Example.git "new folder(2)"

In the given output, only the master branch is cloned from the principal repository Git-Example.

5. "git pull" command

The pull command is used to access the changes (commits)from a remote repository to the local repository. It updates the local branches with the remote-tracking branches. Remote tracking branches are branches that have been set up to push and pull from the remote repository. Generally, it is a collection of the fetch and merges command. First, it fetches the changes from remote and combined them with the local repository.

The syntax of the git pull command is given below:

Syntax:

1. $ git pull **<option>** [**<repository** URL**><refspec>**...]

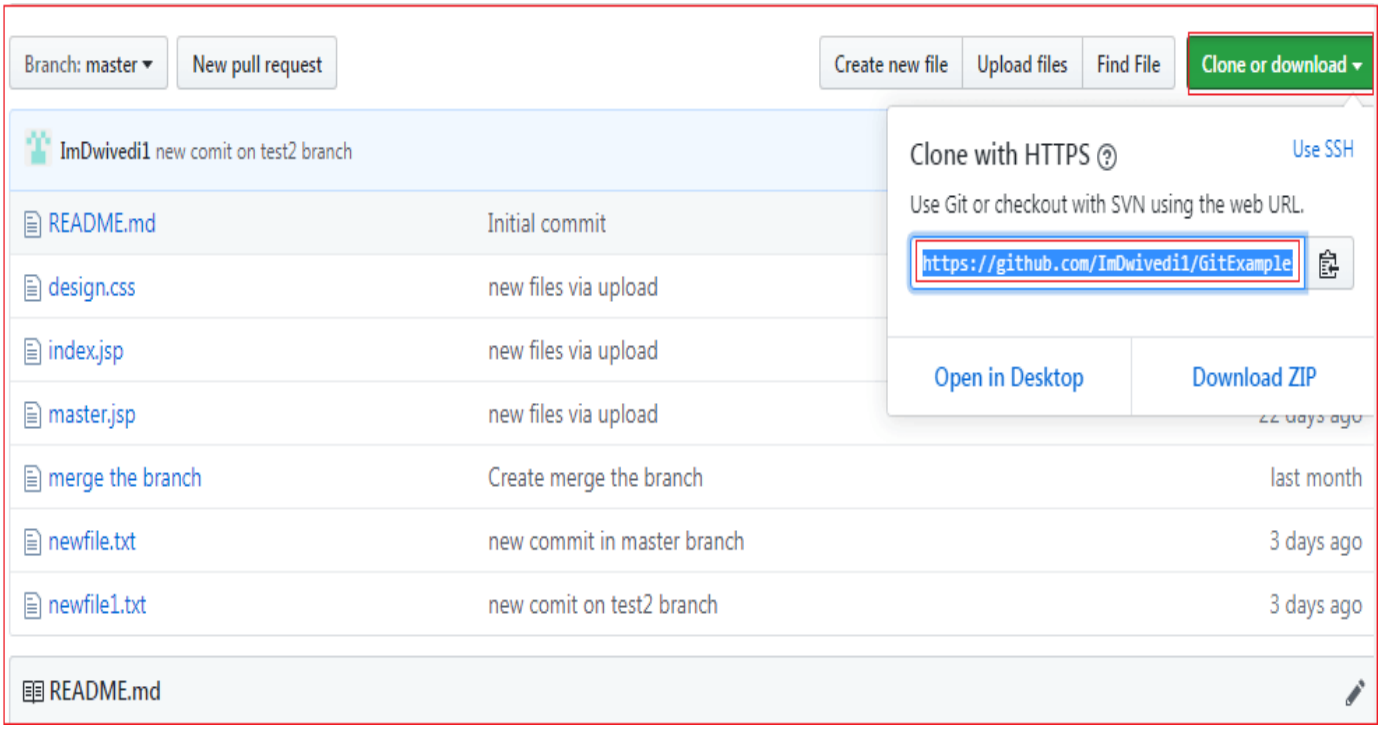
In which:

<option>: Options are the commands; these commands are used as an additional option in a particular command. Options can be **-q** (quiet), **-v** (verbose), **-e**(edit) and more.

<repository URL>: Repository URL is your remote repository's URL where you have stored your original repositories like GitHub or any other git service. This URL looks like:

1. https://github.com/ImDwivedi1/GitExample2.git

To access this URL, go to your account on GitHub and select the repository you want to clone. After that, click on the **clone** or **download** option from the repository menu. A new pop up window will open, select **clone with https option** from available options. See the below screenshot:



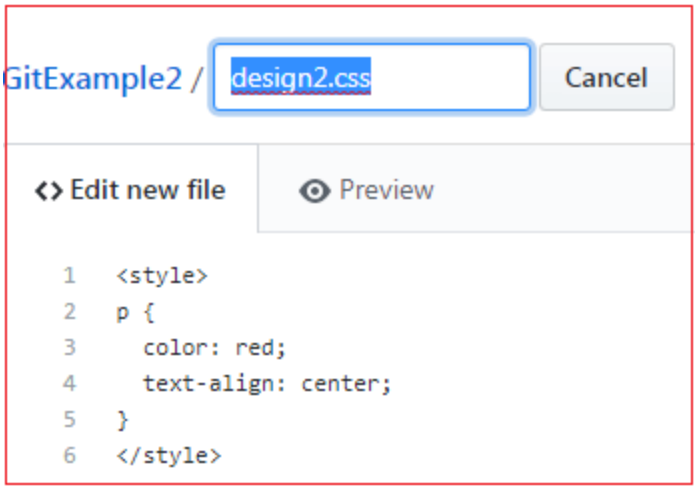
Copy the highlighted URL. This URL is used to Clone the repository.

<Refspec>: A ref is referred to commit, for example, head (branches), tags, and remote branches. You can check head, tags, and remote repository in **.git/ref** directory on your local repository. **Refspec** specifies and updates the refs.

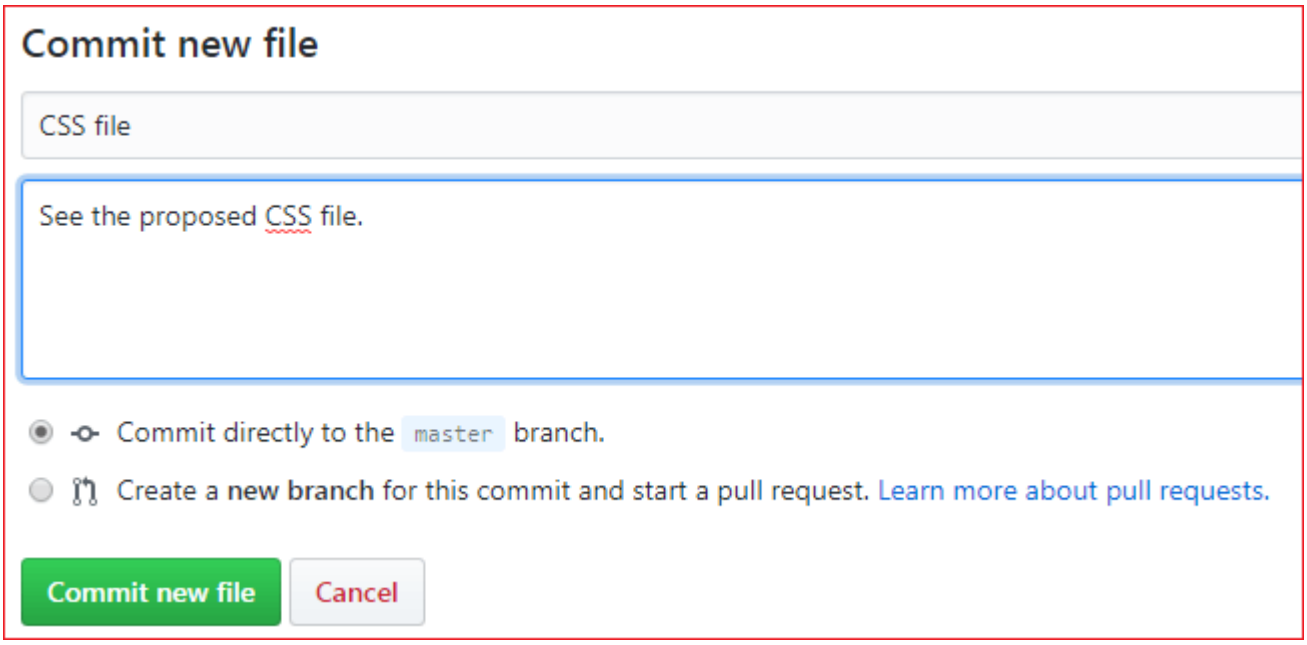
How to use pull:

It is essential to understand how it works and how to use it. Let's take an example to understand how it works and how to use it. Suppose I have added a new file say **design2.css** in my remote repository of project GitExample2.

To create the file first, go to create a file option given on repository sub-functions. After that, select the file name and edit the file as you want. Consider the below image.



Go to the bottom of the page, select a commit message and description of the file. Select whether you want to create a new branch or commit it directly in the master branch. Consider the below image:



Now, we have successfully committed the changes.

To pull these changes in your local repository, perform the git pull operation on your cloned repository. There are many specific options available for pull command. Let's have a look at some of its usage.

Default git pull:

We can pull a remote repository by just using the git pull command. It's a default option. Syntax of git pull is given below:

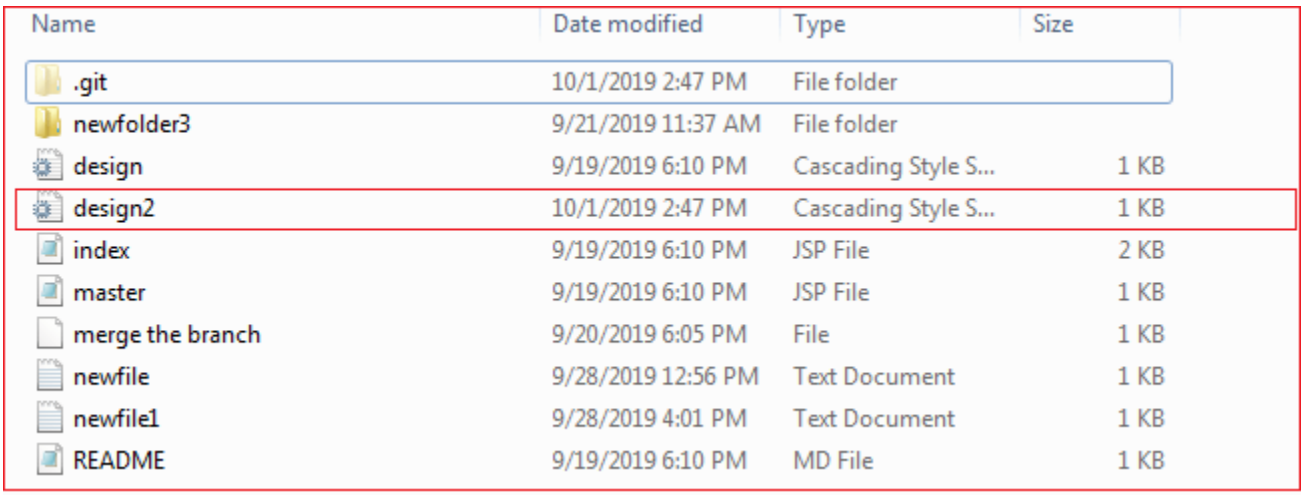
Syntax:

1. $ git pull

Output:

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$ git pull
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/ImDwivedi1/GitExample2
   f1ddc7c..0a1a475  master     -> origin/master
Updating f1ddc7c..0a1a475
Fast-forward
 design2.css | 6 ++++++
 1 file changed, 6 insertions(+)
 create mode 100644 design2.css

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/GitExample2 (master)
$
```

In the given output, the newly updated objects of the repository are fetched through the git pull command. It is the default version of the git pull command. It will update the newly created file **design2.css** file and related object in the local repository. See the below image.

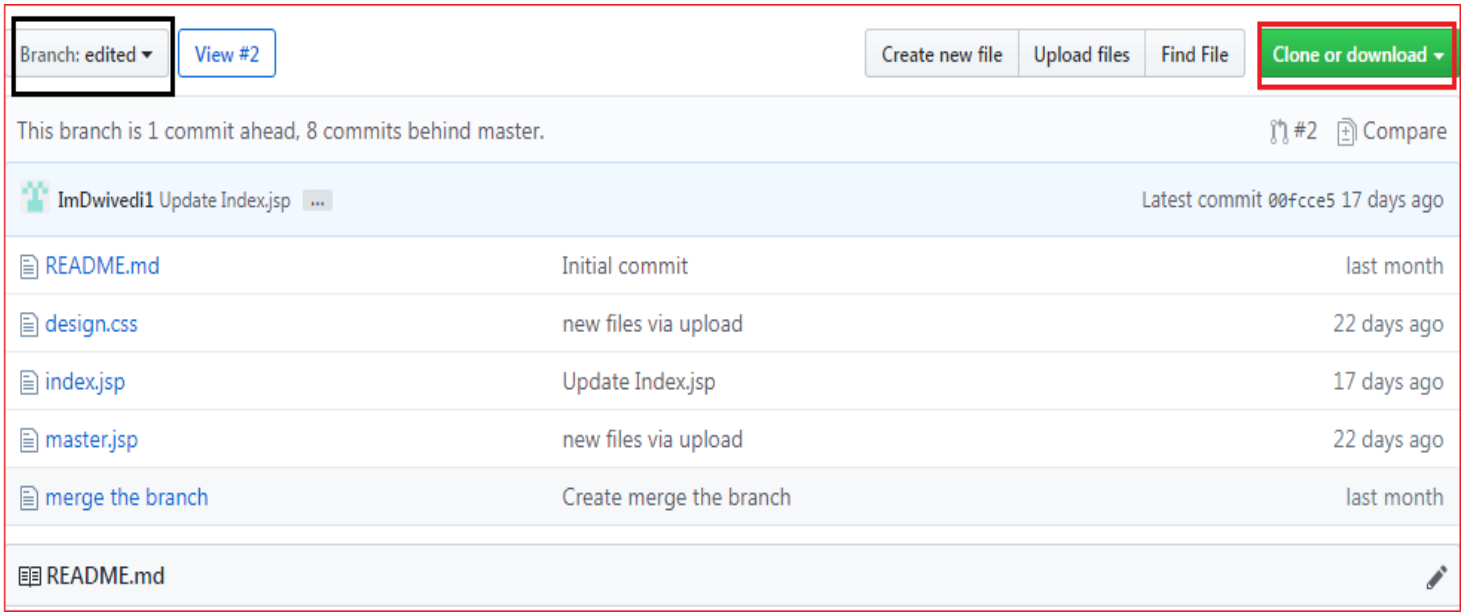


Name	Date modified	Type	Size
.git	10/1/2019 2:47 PM	File folder	
newfolder3	9/21/2019 11:37 AM	File folder	
design	9/19/2019 6:10 PM	Cascading Style S...	1 KB
design2	10/1/2019 2:47 PM	Cascading Style S...	1 KB
index	9/19/2019 6:10 PM	JSP File	2 KB
master	9/19/2019 6:10 PM	JSP File	1 KB
merge the branch	9/20/2019 6:05 PM	File	1 KB
newfile	9/28/2019 12:56 PM	Text Document	1 KB
newfile1	9/28/2019 4:01 PM	Text Document	1 KB
README	9/19/2019 6:10 PM	MD File	1 KB

As you can see in the above output, the design2.css file is added to the local repository. The git pull command is equivalent to **git fetch origin head** and **git merge head**. The head is referred to as the ref of the current branch.

Git Pull Remote Branch

Git allows fetching a particular branch. Fetching a remote branch is a similar process, as mentioned above, in **git pull command**. The only difference is we have to copy the URL of the particular branch we want to pull. To do so, we will select a specific branch. See the below image:



In the above screenshot, I have chosen my branch named **edited** to copy the URL of the edited branch. Now, I am going to pull the data from the edited branch. Below command is used to pull a remote branch:

Syntax:

1. $ git pull **<remote** branch URL**>**

Output:

```
HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/Demo (master)
$ git pull https://github.com/ImDwivedi1/GitExample2.git
remote: Enumerating objects: 38, done.
remote: Counting objects: 100% (38/38), done.
remote: Compressing objects: 100% (25/25), done.
remote: Total 38 (delta 13), reused 19 (delta 7), pack-reused 0
Unpacking objects: 100% (38/38), done.
From https://github.com/ImDwivedi1/GitExample2
 * branch            HEAD       -> FETCH_HEAD

HiMaNshU@HiMaNshU-PC MINGW64 ~/Desktop/Demo (master)
$
```

In the above output, the remote branch **edited** has copied.

6.git fetch command

The "**git fetch**" **command** is used to pull the updates from remote-tracking branches. Additionally, we can get the updates that have been pushed to our remote branches to our local machines. As we know, a branch is a variation of our repositories main code, so the remote-tracking branches are branches that have been set up to pull and push from remote repository.

How to fetch Git Repository

We can use fetch command with many arguments for a particular data fetch. See the below scenarios to understand the uses of fetch command.

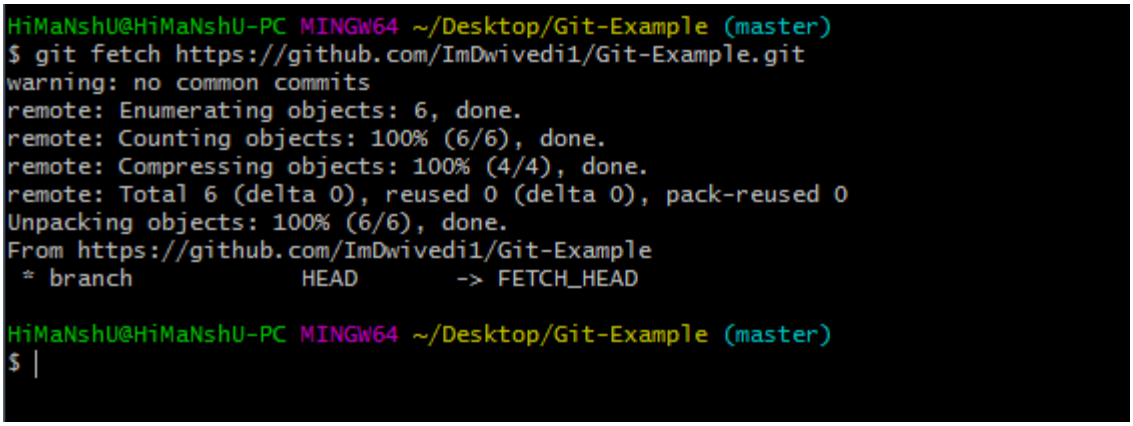
Scenario 1: To fetch the remote repository:

We can fetch the complete repository with the help of fetch command from a repository URL like a pull command does. See the below output:

Syntax:

1. $ git fetch< repository Url>

Output:



In the above output, the complete repository has fetched from a remote URL.

If you want to add working directory use command **git merge**

Differences between Git and GitHub:-

Git: Git is a distributed version control system for tracking changes in source code during software development. It is designed for coordinating work among programmers, but it can be used to track changes in any set of files. Its goals include speed, data integrity, and support for distributed, non-linear workflows.

GitHub: GitHub is a web-based Git repository hosting service, which offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features.

Below is a table of differences between Git and GitHub:

S.No.	Git	GitHub

S.No.	Git	GitHub
1.	Git is a software.	GitHub is a service.
2.	Git is a command-line tool	GitHub is a graphical user interface
3.	Git is installed locally on the system	GitHub is hosted on the web
4.	Git is maintained by linux.	GitHub is maintained by Microsoft.
5.	Git is focused on version control and code sharing.	GitHub is focused on centralized source code hosting.
6.	Git is a version control system to manage source code history.	GitHub is a hosting service for Git repositories.
7.	Git was first released in 2005.	GitHub was launched in 2008.
8.	Git has no user management feature.	GitHub has a built-in user management feature.
9.	Git is open-source licensed.	GitHub includes a free-tier and pay-for-use tier.
10.	Git has minimal external tool configuration.	GitHub has an active marketplace for tool integration.
11.	Git provides a Desktop interface named GitGui.	GitHub provides a Desktop interface named GitHub Desktop.
12.	Git competes with CVS, Azure DevOps Server, Subversion, Mercurial, etc.	GitHub competes with GitLab, Bit Bucket, AWS Code Commit, etc.

Key Git Commands

- **git init:** Initialize a new repository.
- **git clone <repo-url>:** Clone a remote repository to your local machine.
- **git status:** Check the status of the working directory.
- **git add <file>:** Add changes to the staging area.
- **git commit -m "message":** Commit the staged changes.
- **git push origin <branch>:** Push changes to a remote repository.
- **git pull origin <branch>:** Fetch and merge changes from a remote repository.
- **git branch:** View local branches.
- **git checkout <branch>:** Switch to a different branch.
- **git merge <branch>:** Merge changes from one branch into another.
- **git log:** View commit history.
- **git diff:** View the changes between commits or working files.
- **git branch -d <branch>:** Delete a local branch.

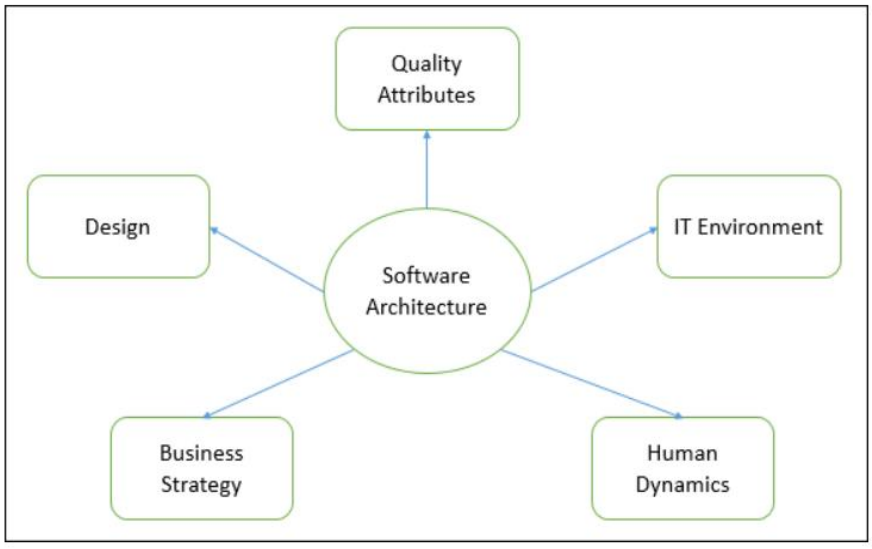
Unit-2

DevOps Influence on Architecture

1. **Introducing software architecture**
2. **The monolithic scenario**
3. **Architecture rules of thumb**
4. **The separation of concerns**
5. **Handling database migrations**
6. **Micro-services**
7. **And the data tier**
8. **DevOps Architecture and resilience**

1. Introducing software architecture

The architecture of a system describes its major components, their relationships (structures), and how they interact with each other. Software architecture and design includes several contributory factors such as Business strategy, quality attributes, human dynamics, design, and IT environment.



We can segregate Software Architecture and Design into two distinct phases: Software Architecture and Software Design. In **Architecture**, nonfunctional decisions are cast and separated by the functional requirements. In Design, functional requirements are accomplished.

Software Architecture

Software architecture is the high-level structure of a software system, focusing on how the system's components are designed and interact with each other. It serves as the blueprint that guides the system's development, ensuring that it meets both functional (what the system does) and non-functional (how the system performs) requirements.

Key Aspects of Software Architecture:

1. **Components**: The major building blocks of the system, such as modules, services, libraries, databases, etc. These components encapsulate specific functionality and often interact with one another to form the complete system.
2. **Patterns and Styles**: Architectural patterns are reusable solutions to common problems. Examples include:
    - **Layered Architecture**: Divides the system into layers such as presentation, business logic, and data access.
    - **Microservices Architecture**: Breaks the system into smaller, independent services that can be developed and deployed separately.
    - **Monolithic Architecture**: A single unified application where all components are interconnected.
3. **Interfaces and Communication**: The way different components of the system communicate. This could involve APIs, messaging queues, or direct calls between services.
4. **Data Management**: How the system stores, retrieves, and manipulates data. This involves databases, file systems, caching mechanisms, and the design of how data is shared or processed.
5. **Non-Functional Requirements**: These are attributes like **performance**, **scalability**, **security**, **availability**, and **maintainability** that the architecture must address. For example, a highly scalable architecture might use microservices to allow different parts of the system to scale independently.

6. **Deployment and Infrastructure**: How the software is deployed and managed in production environments. This includes considerations for cloud infrastructure, containers, orchestration (e.g., Kubernetes), and monitoring.

Why is Software Architecture Important?

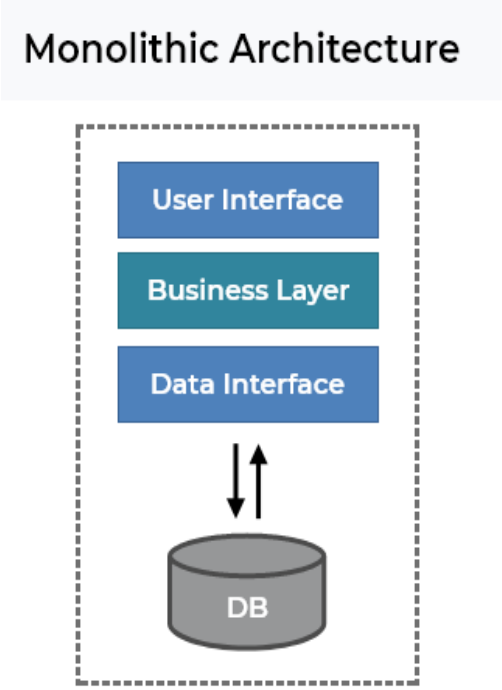
- **System Quality**: A well-designed architecture ensures that the software is reliable, maintainable, and scalable over time. It helps meet both the expected functionality and performance requirements.
- **Guiding Development**: The architecture provides a blueprint that guides the development team throughout the software lifecycle, ensuring consistency and proper organization.
- **Managing Complexity**: As systems grow in size and functionality, a good architecture helps manage complexity by breaking the system into manageable, loosely coupled components.
- **Risk Mitigation**: By addressing potential problems early in the design phase, such as scalability or security concerns, software architecture can help avoid costly issues later.

2. The monolithic scenario

Monolithic Service Architecture:-(Single block)

Monolithic Architecture in Devops refers to a traditional Software Design where all components of a application like front end, Back end and database all these are tightly integrated into a single codebase. Which means everything is kept together in a single folder.

(OR)



- In Monolithic Architecture, User Interface, Business Logic and data access layer works as a single unit, so its easier to manage all operations because all components of the software located in one place.
- In a Monolithic Architecture changes made in one layer can affect other layers Because they all are tightly integrated.

Example:-I created a college website in that college website, I want to add a new feature like adding new courses . So, whenever I change the presentation layers, I need to change the business logic layers and data access layer also.

Advantages:-

1. Simple to develop, test and deploy. Since Everything is in one place.
2. Communication between components is faster since everything in one place.
3. Developers have a clear understanding of entire application as all components are tightly integrated.
4. Deploying Monolithic applications is straight forward compared to micro services since there's only one codebase to manage.

Disadvantages:-

1. It's challenging to update existing ones as the entire application needs to be modified.
2. A fault in one part of the application can bring down the entire system since everything is tightly coupled.
3. As everything is tightly coupled it is difficult for teams to work independently. Leading to coordination challenges.

3. Architecture rules of thumb

Architecture Rules of Thumb in Devops:-

- "Rules of Thumb" are general principles or guidelines widely used in a specific area. They are not strict rules, more like practical suggestions to achieve positive results in System design and implementation.
- In Devops architecture, They represent general best practices that teams can follow to achieve effective and efficient system design.

- Modularity:-Design systems in a modular fashion(i.e., break software into smaller modules that work independently) enabling easier maintenance, updates.
- Scalability:- Ensure that Architecture can handle increased load without sacrificing performance.
- Automation:-Automate process wherever possible, including deployment, testing and monitoring, to increase efficiency and reduce human error.
- Resilience:-Resilience means building systems that can handle failures by including backup plans, automatic switches.
- Infrastructure as Code(Ioc):-Manage Infrastructure like settings to track changes, recreate setups, and setup automatically.
- Continuous Integration/Continuous Deployment(CI/CD):-Automate CI/CD process by creating pipelines
- Monitoring and Logging:-
  Creating monitoring and logging systems to check system performance, and detect issues early.

- Security:- Integrate Security practices into every stage of the development and deployment process, including encryption & access control.
- Containerization:- Utilize Containerization technologies like docker tool to package and manage applications efficiency.
- Feedback loop:- Establish a feedback loop between development, operations and customers to continuously improve process and systems based on real world usage and feedback.

4. The separation of concerns

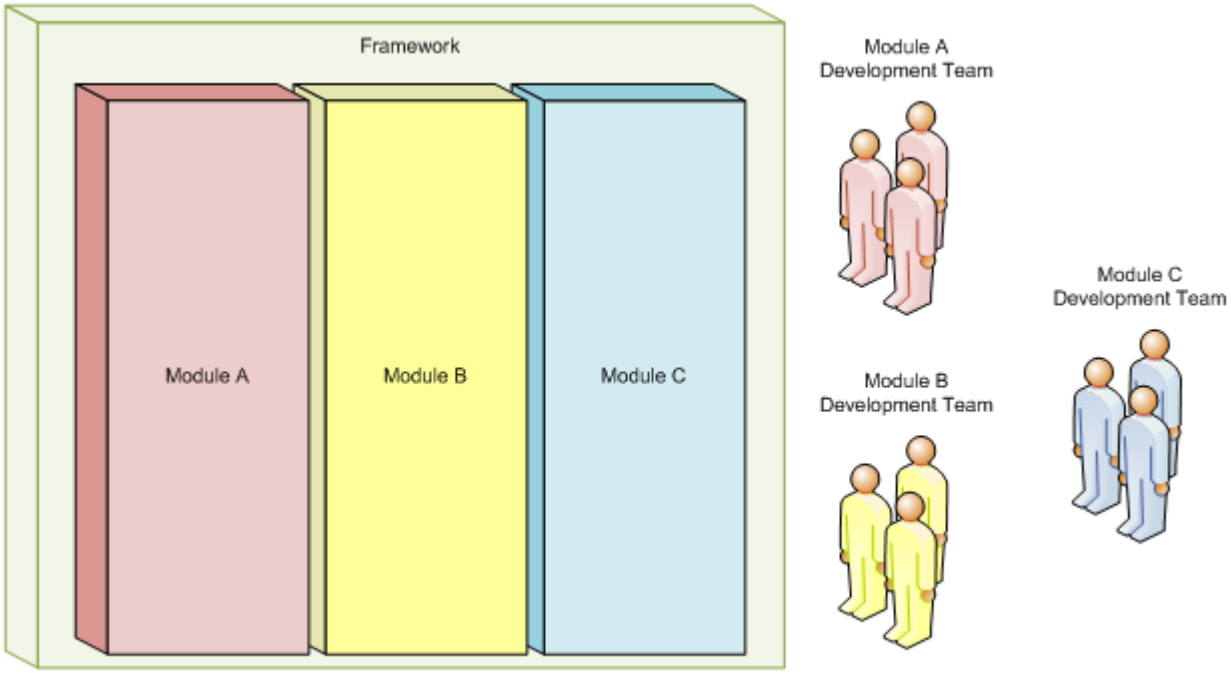
Separation of Concerns in Software Architecture:-

Separation of Concerns(SOC) means dividing entire program into various sections based on their functionalities, so that each section runs independently without effecting other services. Changes made in one section will not affect other sections.

→Separation of Concerns(Soc) will improve

- Maintainability:- Easier to update and fix parts of the code without affecting the whole system.
- Scalability:- Simple to expand the system to handle more load and add simple to add new features
- Readability:-Code is easier to Understand, making it simpler for developers to work with that code.
- Debugging:-Simple to identify and fix bugs

Example:- I am creating a college application. My college application contains four pages, so I will divide it into four modules. The first module contains code related to the student login page, the second module contains code related to the student results page, the third module contains code related to the student attendance page, and fourth module contains code related to faculty details page.Changes made in one module will not affect other modules because they all are separated.

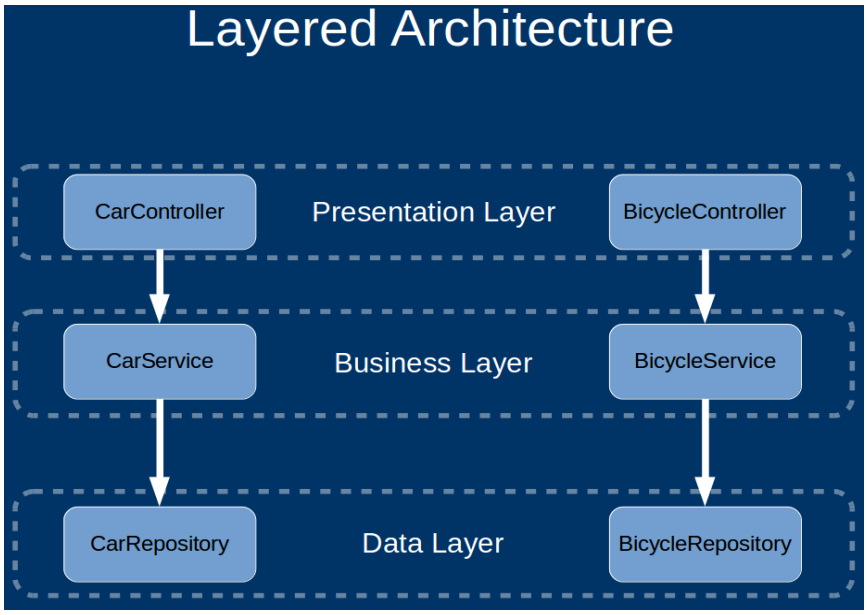


Real world Examples of SOC implementation:-

1. Model-View-Controller(MVC) Architecture:-
   MVC is a classic example of SOC because it divides on application into three components.
   Model:-Manages data and business logic
   View:- Handles the presentation layer
   Controller:-Interface between model and view process user inputs.
   Example:-Web Application
2. Micro Services Architecture:-
   In a Micro Service Architecture, an application is built as a collection of loosely coupled services. Where each service will perform certain task.
   Ex:-E-commerce Platform
3. Client-Server Architecture:-
   This Architecture Splits an application into two main components. They are client and service
   - Client:-Front end that interacts with users
   - Server:-Backend that process User requests and manages data

   Example:-Social Media Applications

4. Layered Architecture:-It divides an application into layers, each layer has specific role.



Example:-Online Banking System

5. Handling database migrations

Handling Database Migrations in Devops:-

→Database migration means transferring data from one database to another. This process can not only include just moving the data itself, but also transforming it to fit the new database structure(schema), updating databse structure, and ensuring data integrity and consistency throughout the migration.

→In Devops handling database migrations means managing the process of moving data from one databse to another, while ensuring it's done smoothly and without disturbances. This is important because changes to the databse structure or data need to be matched with application updates to avoid errors.

Techniques Used for Database Migration in Devops:-

1. Version Control(Git):-One of the essential techniques for database migration in devops is to use version control for both the application code and the databse schema(Structure). Tools like Liquibase and Flyway can

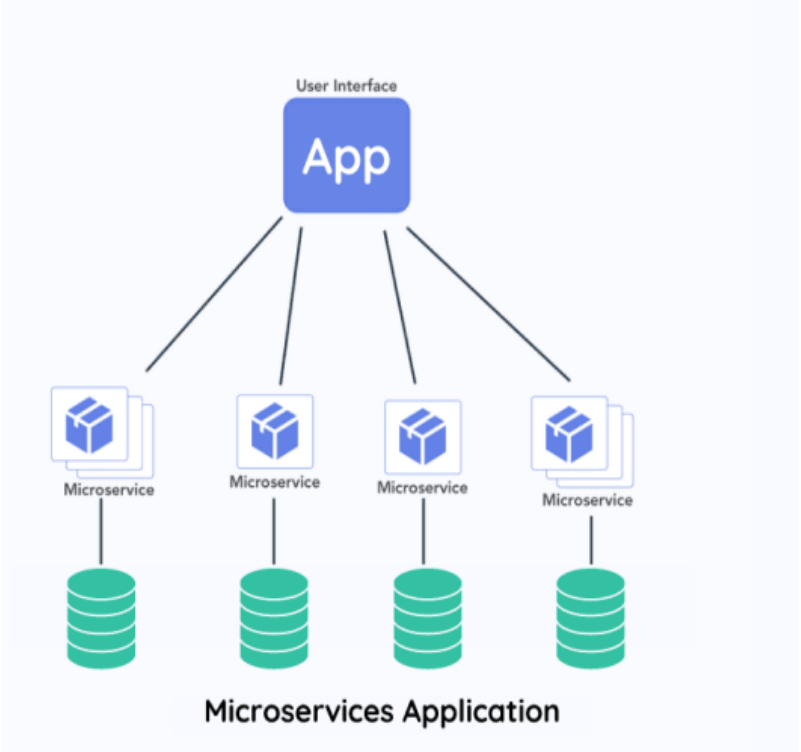
be integrated into CI/CD pipelines to automatically apply database schema changes when corresponding changes are made to the code.

2. Schema Migration Tools:-
   These tools automate making and running SQL Scripts that change your database structure(Schema). They prevent mistakes and reduce downtime. Examples are SQl server Data tools and MySql workbench(Just write SQL Script to Specific Changes instead of writing complete code)
3. Data Migration Tools:-
   Sometimes, you need to move data between databases Data migration tools make this easy and safe they can handle big data, different formats, and keep everything in synchronization.
   Example:- AWS Database Migration Services(DMS) and Azure Data Factory
4. Testing Strategies:-
   Lastly, you need to test the migrated database to make sure it works well. This includes various tests like checking performance and security. Testing ensures the databases meets all requirements and fixes any issues.
   Examples are SQL Test and JMeter.

6. Micro-services

Micro Services Architecture in Devops:-

If you consider Monolithic Architecture entire application is placed in one single folder. So changes made to one component will effect other components. But where as in micro services Architecture you are application is placed in bunch of folders each folder is responsible for a specific function. So changes made in one folder will not effect other folders.



Example:-

In a college website each function like student attendance, faculty Attendance, Student Results , college courses can be considered as separate services in a micro services architecture. Changes made to one service, such as updating the student attendance feature won't effect the other services like faculty attendance or student results.

Benefits:-

1. Scalability:-
   You can modify Specific part of our application independently without effecting other parts.
2. Flexibility:-
   Teams can choose the best tools and technologies for each micro service making it easier to adapt changing requirements.
3. Faster Deployment:-
   Since each micro service is independent you can deploy updates without effecting whole system.
4. Resilience:-

If one micro service fails, it doesn't bring down the entire application. The rest can still function normally.

Challenges:-

1. Complexity:-
   Managing multiple micro services can be more complex then monolithic architecture.
2. Communication:-
   Services need to communicate effectively, which requires good design and coordination.
3. Monitoring and Debugging:-
   With more micro services monitoring and debugging can be challenging.

What are the main components of Microservices Architecture?

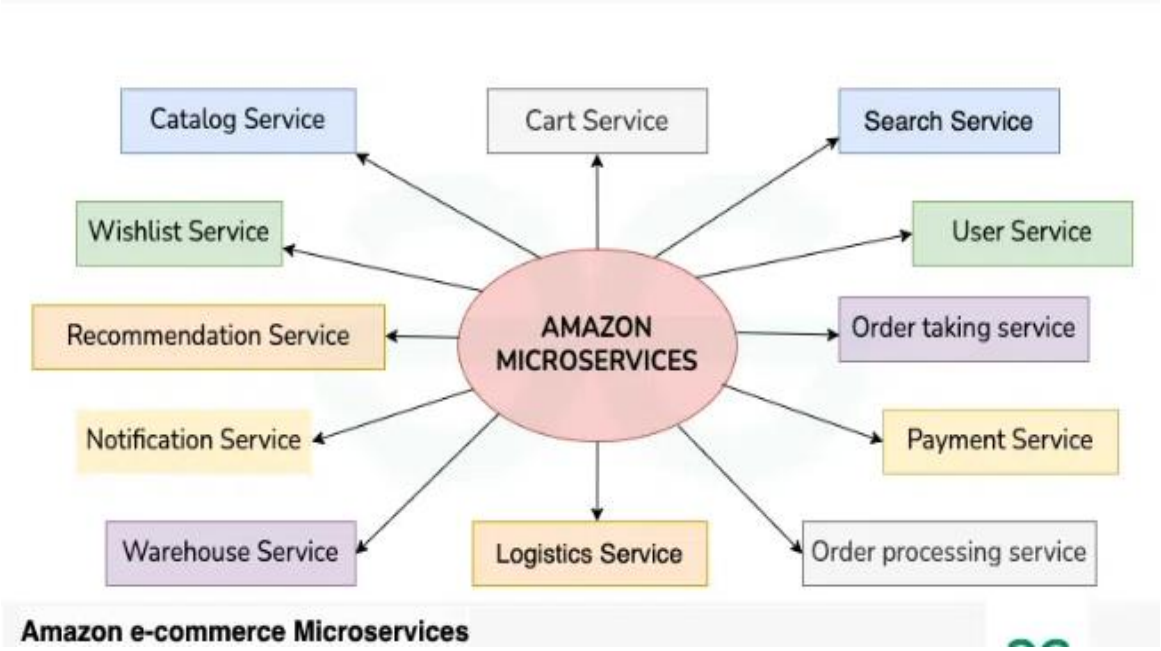
Main components of microservices architecture include:

- **Microservices:** Small, loosely coupled services that handle specific business functions, each focusing on a distinct capability.
- **API Gateway:** Acts as a central entry point for external clients also they manage requests, authentication and route the requests to the appropriate microservice.
- **Service Registry and Discovery:** Keeps track of the locations and addresses of all microservices, enabling them to locate and communicate with each other dynamically.
- **Load Balancer:** Distributes incoming traffic across multiple service instances and prevent any of the microservice from being overwhelmed.
- **Containerization:** Docker encapsulate microservices and their dependencies and orchestration tools like Kubernetes manage their deployment and scaling.
- **Event Bus/Message Broker:** Facilitates communication between microservices, allowing pub/sub asynchronous interaction of events between components/microservices.
- **Database per Microservice:** Each microservice usually has its own database, promoting data autonomy and allowing for independent management and scaling.
- **Caching:** Cache stores frequently accessed data close to the microservice which improved performance by reducing the repetitive queries.
- **Fault Tolerance and Resilience Components:** Components like circuit breakers and retry mechanisms ensure that the system can handle failures gracefully, maintaining overall functionality.

→Real-World Example of Microservices

Let’s understand the Miscroservices using the real-world example of Amazon E-Commerce Application:

Amazon’s online store is like a giant puzzle made of many small, specialized pieces called microservices. Each microservice does a specific job to make sure everything runs smoothly. Together, these microservices work behind the scenes to give you a great shopping experience.



Amazon e-commerce Microservices

Below are the microservices involved in Amazon E-commerce Application:

- **User Service**: Handles user accounts and preferences, making sure each person has a personalized experience.
- **Search Service**: Helps users find products quickly by organizing and indexing product information.
- **Catalog Service**: Manages the product listings, ensuring all details are accurate and easy to access.
- **Cart Service**: Lets users add, remove, or change items in their shopping cart before checking out.
- **Wishlist Service**: Allows users to save items for later, helping them keep track of products they want.
- **Order Taking Service**: Processes customer orders, checking availability and validating details.
- **Order Processing Service**: Oversees the entire fulfillment process, working with inventory and shipping to get orders delivered.
- **Payment Service**: Manages secure transactions and keeps track of payment details.

- **Logistics Service**: Coordinates everything related to delivery, including shipping costs and tracking.
- **Warehouse Service**: Keeps an eye on inventory levels and helps with restocking when needed.
- **Notification Service**: Sends updates to users about their orders and any special offers.
- **Recommendation Service**: Suggests products to users based on their browsing and purchase history

7. And the data tier

The Data Tier:-

The Three-Tier Client-Server Architecture is a layered approach to building distributed systems, with each tier serving distinct roles. Below is a detailed explanation of each component:

1. Presentation Tier (Client Tier)

The user interface and user interaction are under the control of the Presentation Tier. It functions as the application's front end, where users enter information and see the outcomes.

- **Components:**
    - **User Interface (UI):** Includes web browsers, mobile apps, or desktop applications that users interact with. It displays data and collects user inputs.
    - **User Interaction Logic:** Handles how user inputs are processed and communicated to the Application Tier. This can involve form validation, data formatting, and sending requests to the server.
- **Responsibilities:**
    - Display data from the Application Tier to the user.
    - Collect user inputs and forward them to the Application Tier.
    - Provide a user-friendly interface and manage user interactions.

2. Application Tier (Business Logic Tier)

The Application Tier is where the core business logic resides. It processes user requests, performs calculations, enforces business rules, and interacts with the Data Tier to retrieve or store data.

- **Components:**

    - **Business Logic:** Implements the rules and processes specific to the application, such as order processing, authentication, or data validation.
    - **Application Server:** Manages communication between the Presentation and Data Tiers and hosts the business logic. Web servers and application servers such as Apache Tomcat, Microsoft IIS, or JBoss are examples.
- **Responsibilities:**
    - Process and interpret data received from the Presentation Tier.
    - Execute business logic and apply rules.
    - Communicate with the Data Tier to retrieve or store information.
    - Send processed data back to the Presentation Tier for user display.

3. Data Tier (Database Tier)

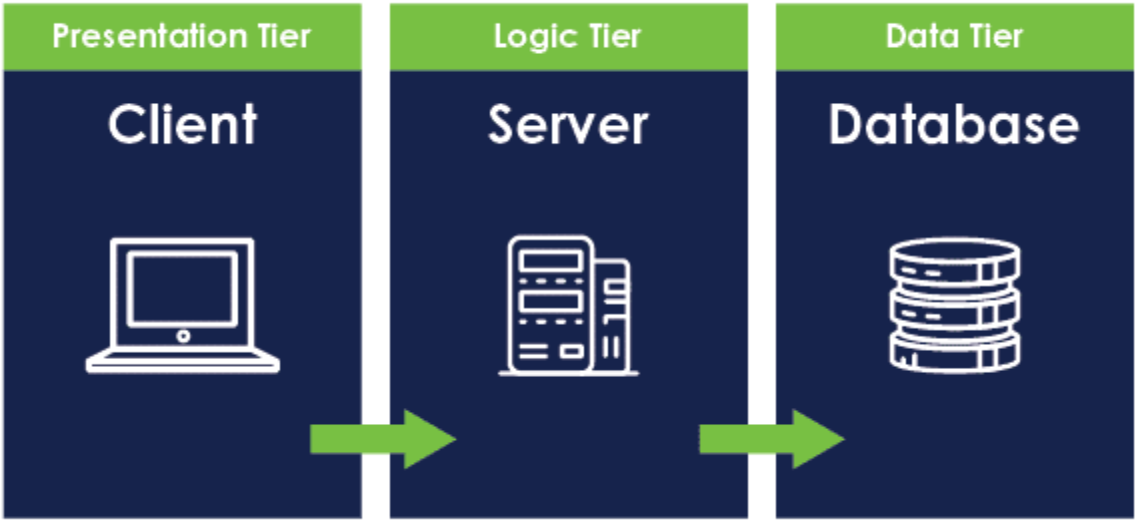
The Data Tier is responsible for data management and storage. It handles all database operations, including data retrieval, updates, and management.

- **Components:**
    - **Database Management System (DBMS):** Software like MySQL, Oracle, or Microsoft SQL Server that manages data storage and retrieval.
    - **Database:** The actual repository where data is stored, organized in tables or other structures.
- **Responsibilities:**
    - Store and manage data securely and efficiently.
    - Handle queries and transactions initiated by the Application Tier.
    - Ensure data integrity, consistency, and availability.
    - Provide backup and recovery mechanisms to protect data.

Interactions Among Tiers

- **Client Request:** The user interacts with the Presentation Tier, which sends a request to the Application Tier.
- **Business Processing:** The Application Tier processes the request using its business logic and may interact with the Data Tier to retrieve or update data.
- **Data Retrieval/Update:** The Data Tier handles data operations and sends the results back to the Application Tier.
- **Response to Client:** The Application Tier processes the results and sends the response back to the Presentation Tier for display to the user.

This tiered approach helps manage complexity by separating responsibilities, allowing for easier maintenance, scalability, and flexibility in distributed systems



The Power of Separation:
The beauty of the three-tier architecture lies in its **separation of concerns**. Each tier is independent, with its own set of responsibilities and technologies. This brings a wealth of benefits:

- **Flexibility and Scalability:** Imagine expanding the palace! Each tier can be scaled independently, allowing you to beef up the kitchen (application tier) to handle increased orders without affecting the ballroom (presentation tier).
- **Maintainability and Reusability:** Need to update the palace gardens (presentation tier)? No need to tear down the entire structure! Changes to one tier are isolated, making maintenance a breeze and code reusable across projects.
- **Teamwork and Expertise:** Different teams can focus on their areas of expertise, like frontend developers beautifying the gardens while backend engineers strengthen the foundations (application and data tiers).

Examples in Action:
To truly understand the three-tier architecture, let's take a trip to familiar territory:

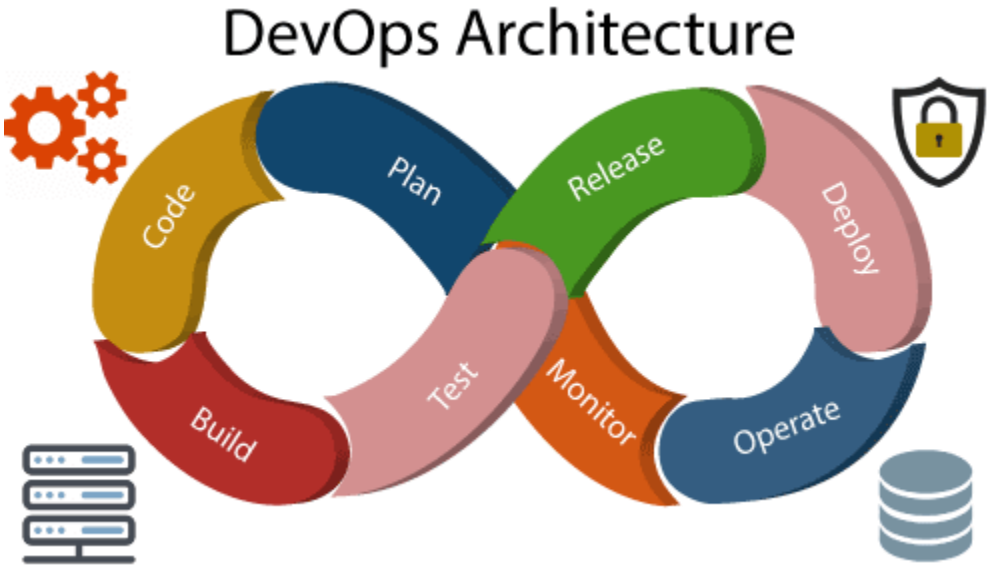
- **E-commerce website:** The product pages are the presentation tier, the shopping cart and checkout process are the application tier, and the product database is the data tier.

- **Online banking app:** The account overview and transaction history are the presentation tier, the payment processing and security checks are the application tier, and your financial data resides securely in the data tier.

A DevOps Champion:
As a DevOps champion, I see the three-tier architecture as a boon for continuous delivery and deployment. With independent tiers, updates and bug fixes can be rolled out to specific layers without disrupting the entire system. Imagine renovating the palace kitchen without disturbing the ongoing royal banquet in the grand hall!

8. DevOps Architecture

DevOps Architecture:-



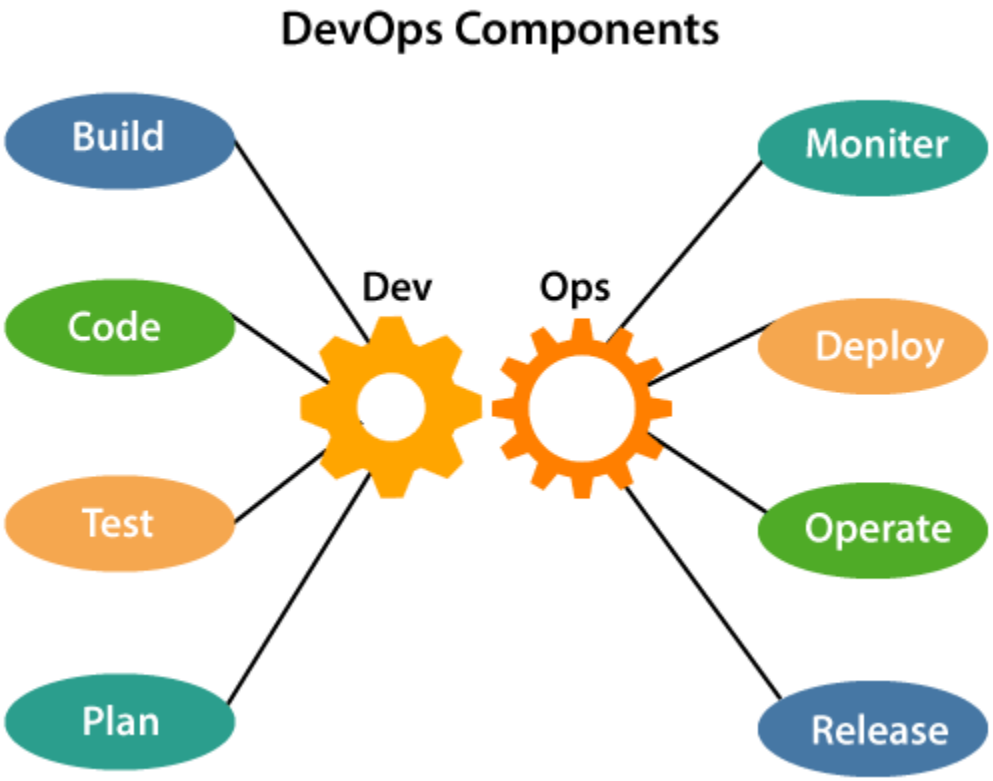
Development and operations both play essential roles in order to deliver applications. The deployment comprises analyzing the **requirements, designing, developing**, and **testing** of the software components or frameworks.

The operation consists of the administrative processes, services, and support for the software. When both the development and operations are combined with

collaborating, then the DevOps architecture is the solution to fix the gap between deployment and operation terms; therefore, delivery can be faster.

DevOps architecture is used for the applications hosted on the cloud platform and large distributed applications. Agile Development is used in the DevOps architecture so that integration and delivery can be contiguous. When the development and operations team works separately from each other, then it is time-consuming to **design, test**, and **deploy**. And if the terms are not in sync with each other, then it may cause a delay in the delivery. So DevOps enables the teams to change their shortcomings and increases productivity.

Below are the various components that are used in the DevOps architecture:



1) Build

Without DevOps, the cost of the consumption of the resources was evaluated based on the pre-defined individual usage with fixed hardware allocation. And with DevOps, the usage of cloud, sharing of resources comes into the picture, and the build is dependent upon the user's need, which is a mechanism to control the usage of resources or capacity.

2) Code

Many good practices such as Git enables the code to be used, which ensures writing the code for business, helps to track changes, getting notified about the reason behind the difference in the actual and the expected output, and if necessary reverting to the original code developed. The code can be appropriately arranged in **files, folders**, etc. And they can be reused.

3) Test

The application will be ready for production after testing. In the case of manual testing, it consumes more time in testing and moving the code to the output. The testing can be automated, which decreases the time for testing so that the time to deploy the code to production can be reduced as automating the running of the scripts will remove many manual steps.

4) Plan

DevOps use Agile methodology to plan the development. With the operations and development team in sync, it helps in organizing the work to plan accordingly to increase productivity.

5) Monitor

Continuous monitoring is used to identify any risk of failure. Also, it helps in tracking the system accurately so that the health of the application can be checked. The monitoring becomes more comfortable with services where the log data may get monitored through many third-party tools such as **Splunk**.

6) Deploy

Many systems can support the scheduler for automated deployment. The cloud management platform enables users to capture accurate insights and view the optimization scenario, analytics on trends by the deployment of dashboards.

7) Operate

DevOps changes the way traditional approach of developing and testing separately. The teams operate in a collaborative way where both the teams actively participate throughout the service lifecycle. The operation team interacts with developers, and they come up with a monitoring plan which serves the IT and business requirements.

8) Release

Deployment to an environment can be done by automation. But when the deployment is made to the production environment, it is done by manual triggering. Many processes involved in release management commonly used to do the deployment in the production environment manually to lessen the impact on the customers.

9. Resilience

Resilience in Devops:-

Resilience in Devops means the ability of Systems to recover quickly from problems and continue working smoothly. It ensures that services remain available even when there are issues like failures or high traffic.

How Organizations Achieve Resilience Using Devops:-

1. Automation:-
   Automate tasks like deployment and testing. Reduces human errors and speeds up recovery
2. Continuous Integration/Continuous Deployment(CI/CD):-
   Regularly integrate and deploy small changes. Makes it easier to find and fix issues quickly.
3. Monitoring and Logging:-
   Logging(It Means Recording detailed information about the operations of software)
   Constantly check systems for problems. Use logs to understand issues and respond faster.
4. Infrastructure as code(Ias):-
   Infrastructure(It means hardware, software, networks, servers, database and other components)
   Manage Infrastructure with code. Easily rebuild or update systems using scripts.
5. Micro Services Architecture:-
   Break applications into smaller, independent services. If one service fails, others continue to work.

6. Regular Backups and Disaster Recovery Plans:-
   Keep backups of data and systems. Have Plans to recover from major failures.
7. Scalability:-
   Design Systems to handle more load automatically.

Unit-3

Part-B

Configuration Management with Ansible:

1. **Introduction to Ansible,**
2. **Ansible architecture and installation,**
3. **Inventory management,**
4. **Ad-hoc commands and playbooks,**
5. **Roles and modules in Ansible,**
6. **Writing and managing Ansible playbooks,**
7. **Integrating Ansible with other tools**.

1.Introduction to Ansible

Ansible:

Ansible is a simple, powerful, and agentless IT automation tool that is used for IT configuration management, application deployment, and perform repetitive task. It simplifies complex tasks by automating them using simple YAML-based scripts called Playbooks.

Why Ansible is important

• Automates repetitive tasks (e.g., software installation, configuration updates).

• Manages multiple servers with a single script.

• Works across platforms (Linux, Windows, Cloud, On-Premise).

• Integrates with DevOps tools like Jenkins, Docker, Kubernetes.

• Enhances security by enforcing configuration consistency.

Overview :

Ansible is an IT automation engine that can automate various IT needs. And it has features like application deployment that means you can deploy your application easily as per your requirements, cloud provisioning, configuration management is also the main feature where you can configure and describe your automation job, and intra-service orchestration. In this, (Yet Another Markup Language)YAML is used for configuring that helps for describing automation jobs as per requirement. It is Designed for multi-tier deployments, Ansible models the IT

infrastructure by describing how various systems interrelate, instead of managing one system at a time.

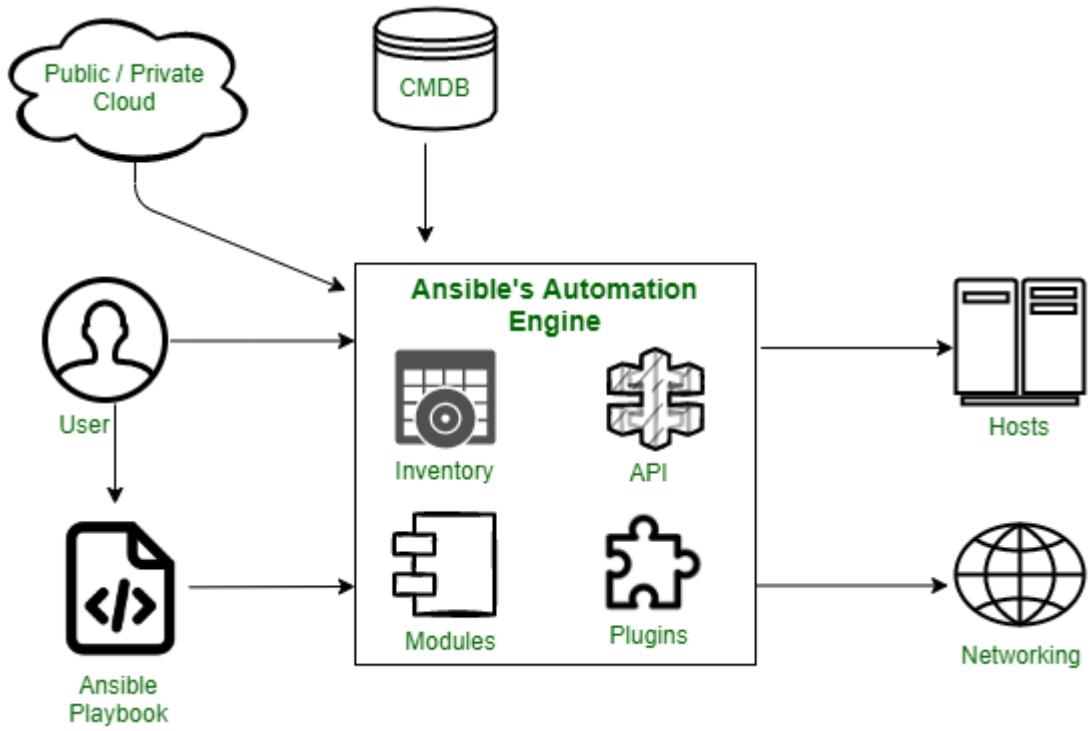
Features :

In this, It uses no extra functionality and cost like no agents and no extra custom security infrastructure, hence it is easy to deploy.

It uses a very simple language called YAML (Yet Another Markup Language) in the form of Ansible Playbooks and you can configure it as per your requirement, and it helps describe the automation jobs in a way that looks like basic English.

The Ansible Automation Engine has a direct interaction with the users who write playbooks and also interacts with cloud services and the Configuration Management Database (CMDB).

2. Ansible architecture and installation



a.Inventories -

Ansible inventories are lists of hosts with their IP addresses, servers, and databases which have to be managed via an SSH for UNIX, Linux, or Networking devices, and WinRM for Windows systems.

b.APIs -
Application Programming Interface or APIs are used as a mode of transport for public and private cloud services.

c.Modules -
Modules are executed directly on remote hosts through playbooks and can control resources like services, packages, files, or execute system commands. They act on system files, install packages and make API calls to the service network. There are over 450 Ansible that provide modules that automate various jobs in an environment. For example, Cloud Modules like Cloud Formation create or delete an AWS cloud formation stack.

d.Plugins -
Plugins are pieces of code that augment Ansible's core functionality and allow executing Ansible tasks as a job build step. Ansible ships with several handy plugins and one can also write it on their own. For example, Action plugins act as front-ends to modules and can execute tasks on the controller before calling the modules themselves.

e.Networking-
Ansible uses a simple, powerful, and agent-less automation framework to automate network tasks. It uses a separate data model and spans different network hardware.

f.Hosts -
Hosts refer to the nodes or systems (Linux, Windows, etc) which are automated by Ansible.

g.Playbooks -
Playbooks are simple files written in YAML format which describe the tasks to be executed by Ansible. Playbooks can declare configurations, orchestrate the steps of any manual ordered process and can also launch various tasks.

h.CMDB -
It stands for Configuration Management Database (CMDB). In this, it holds data to a collection of IT assets, and it is a repository or data warehouse where we will store this kind of data, and It also defines the relationships between such assets.

I.Cloud -
It is a network of remote servers hosted on the internet to store, manage and process data instead of storing it on a local server.

J.Control Node

The Control Node is the machine where Ansible is installed and executed. It is responsible for managing and orchestrating the execution of automation tasks.

K.Managed Nodes (Hosts)

These are the target machines (Linux, Windows, or network devices) where Ansible executes tasks. Managed nodes do not require Ansible to be installed, making it agentless.

Inventory

The inventory file lists all the managed nodes (hosts) that Ansible will control. Managed Nodes

(Hosts)

[Webserver]

125.11.58.85

Playbooks

Playbooks are YAML files that define automation tasks in a structured manner.

YAML Structure:

- name: Install Apache

hosts: webservers

tasks:

- name: Install Apache on Debian

apt:

name: apache2

state: present

systemd:

name: apache

state: started

enabled: yes

Installation

For Installing ansible in Linux (Ubuntu) System we have some steps.

 Sudo apt update

 sudo apt install -y software-properties-common

 Sudo apt install ansible

For Installing ansible in Windows System we have some steps.

 Open windows power shell as administrator mode

 Run command “wsl –install”

 Verify WSL installation using “wsl --list –online” command

 Install Ubuntu as the default WSL distribution using “wsl --install -d Ubuntu” After installation of ubuntu, open ubuntu from start menu and run the ubuntu and create and account

 Now run “sudo apt update command” for update the dependencies

 Install Ansible using “sudo apt install ansible” command

3.Inventory management

Inventory management in Ansible, refers to the process of defining, organizing, and managing the list of target hosts that Ansible will control. The inventory file contains details such as hostnames, IP addresses, groups, and connection parameters.

 A target host may be a Cloud instance, Network Device, or physical servers.

Mainly we have two types of inventory:

Static Inventory: Defined by manually using .ini or .yml file for configure and manage the host server.

A. [web_servers]

web1.example.com

web2.example.com

B. [db_servers]

db1.example.com ansible_host=192.168.1.10 ansible_user=root

Dynamic Inventory: Dynamic inventory allows Ansible to fetch host details from cloud providers, container platforms, or other external sources dynamically. This is useful for large, frequently changing infrastructures.

Ex: AWS, Azure, Cisco, MS-365, VPN Services, GoogleHome and Alexa, Cryptocurrency, Airtel, Jio Starlink etc.

To enable AWS dynamic inventory, create a configuration file (aws_ec2.yml)

plugin: aws_ec2

regions:

- us-east-1

keyed_groups:

- key: tags.Environment

prefix: env_

Show the list of dynamic inventory

ansible-inventory -i aws_ec2.yml –list

Managing Inventory in Ansible

1. First Check all inventory host list using given command

ansible-inventory -i inventory.yml –list

2. Check weather it is connected or not with host server with given command

ansible all -m ping -i inventory.yml

3. Run ad-hoc command on any one host for checking uptime downtime details using given

command

ansible web_servers -m command -a "uptime" -i inventory.yml

4.Ad-hoc commands and playbooks

Ad-Hoc Commands and playbook

Ad-hoc command in Ansible is a single line or linear command that is used for execute single task or one or more managed node without writing a long playbook. it is often used for Restart,

stop, ping and copy a file. Ad-hoc command is a light-weight and easy but they are not a reusable command.

Basics Syntax of Ad-hoc command

ansible <host-pattern> -m <module> -a "<module-arguments>"

Some Ad-Hoc Commands

1. Ping a single host or all host:

 ansible <host-name> -m ping

 ansible all -m ping

2. Restart single services or all services:

 ansible <host-name> -m service -a "name=httpd state=restarted“

 ansible all -m service -a "name=httpd state=restarted“

3. Copy a file to a remote server:

 ansible all -m copy -a "src=/home/user/file.txt dest=/tmp/file.txt“

4. Install a package using yum command:

 ansible all -m yum -a "name=httpd state=present"

Playbook:-

An Ansible Playbook is a YAML file that defines a series of automation tasks to be executed on

remote machines. Unlike ad-hoc commands (which are for one-off tasks), playbooks are declarative,

repeatable, and designed for complex configurations and orchestration.

YAML Syntax: Playbooks are written in YAML, which is human-readable and structured using

indentation.

Play: A play is a mapping between a group of hosts and tasks. A playbook can contain one or more

plays.

- **YAML File:** Playbooks are just text files ending with .yaml or .yml.

- **Hosts:** You tell the playbook *which* servers to run on (e.g., all servers, web servers, database servers).
- **Tasks:** These are the individual steps or actions Ansible will perform. Each task uses an Ansible **module** to do something specific.
- **Modules:** These are like specialized tools. For example:
    - ansible.builtin.ping: Checks if a server is reachable.
    - ansible.builtin.apt or ansible.builtin.yum: Installs software packages.
    - ansible.builtin.copy: Copies files.
    - ansible.builtin.service: Starts, stops, or restarts services.
- **become: true:** This means "run this task with elevated privileges," usually like sudo (acting as an administrator).
- **Order Matters:** Tasks in a playbook run from top to bottom, in the order you list them.

Small Example: Installing Nginx

Let's say you want to install the Nginx web server on a machine. Here's a super simple playbook:

YAML

```
---
- name: Install Nginx web server
  hosts: webserver_group # This refers to a group of servers in your inventory
  become: true           # We need root privileges to install software

  tasks:
    - name: Ensure Nginx package is installed
      ansible.builtin.apt:
        name: nginx
        state: present

    - name: Ensure Nginx service is running and enabled
      ansible.builtin.service:
        name: nginx
        state: started
        enabled: true
```

Simple YAML Structure and YAML Structure with Roles.

Simple YAML	YAML with roles
- name: Install the correct web server for RHEL import_tasks: redhat.yml when: ansible_facts['os_family']|lower == 'redhat' - name: Install the correct web server for Debian import_tasks: debian.yml when: ansible_facts['os_family']|lower == 'debian' - name: Install web server ansible.builtin.yum: name: "httpd" state: present - name: Install web server ansible.builtin.apt: name: "apache2" state: present	# roles/example/tasks/main.yml - name: Install the correct web server for RHEL import_tasks: redhat.yml when: ansible_facts['os_family']|lower == 'redhat' - name: Install the correct web server for Debian import_tasks: debian.yml when: ansible_facts['os_family']|lower == 'debian' # roles/example/tasks/redhat.yml - name: Install web server ansible.builtin.yum: name: "httpd" state: present # roles/example/tasks/debian.yml - name: Install web server ansible.builtin.apt: name: "apache2" state: present

5.Roles and modules in Ansible

Roles:

In Ansible, Roles is a structured way of organizing playbooks, making complex playbooks more manageable, reusable, and scalable using breakdown a long playbook into a different small files. It can make complex playbook Easier to manage and share.

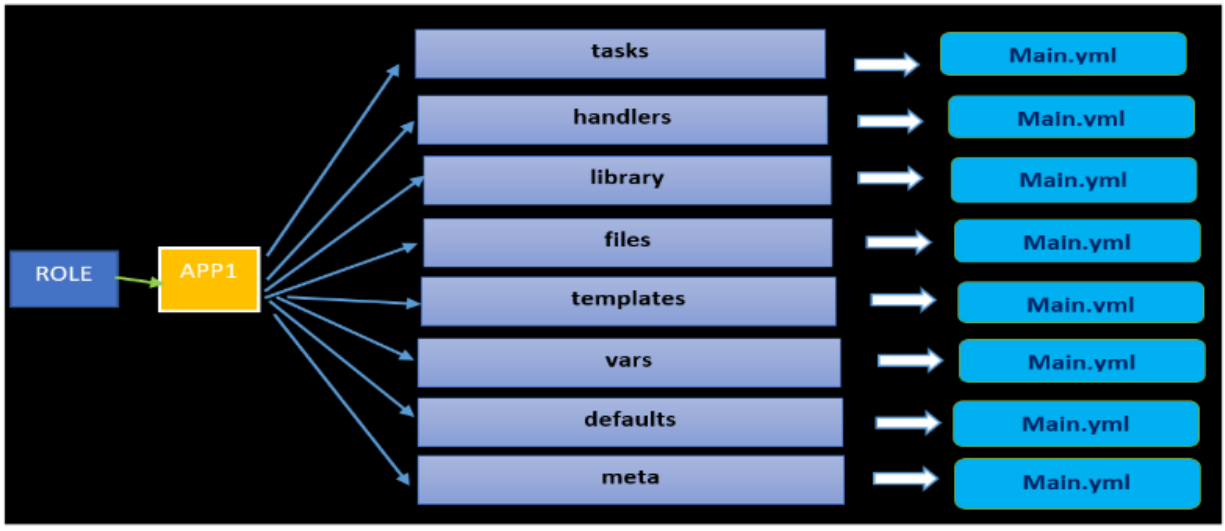


Fig: Architecture of Roles.

Purpose of Each Folder

 tasks/ – Main list of tasks to be executed.

 handlers/ – Tasks triggered by notify.

 files/ – Static files to copy to hosts.

 templates/ – contain templates for configuration files.

 vars/ – Variables with higher precedence.

 defaults/ – Default variables lowest precedence.

 meta/ – Contain Role metadata and dependencies.

Why roles are Important

 Reusability: you can create it once and reuse it in different projects and environments.

 Modularity: Roles allow breaking down playbooks into reusable components. Each role can manage a particular aspect of the system, such as installing a web server or configuring a database.

 Maintainability: By organizing tasks, variables, handlers, and other components into separate directories, roles make the codebase easier to navigate and maintain.

 Scalability: As infrastructure grows, roles help manage complexity by providing a clear structure, making it easier to scale configurations.

Use Cases of Ansible Role

Ansible Roles are used in a variety of contexts, including:

 Infrastructure as Code (IaC): Managing and provisioning infrastructure in a consistent and repeatable manner.

 Application Deployment: Deploying applications with all necessary dependencies and configurations.

 Configuration Management: Ensuring systems are configured correctly and consistently across environments.

 Continuous Integration/Continuous Deployment (CI/CD): Integrating with CI/CD pipelines to automate the deployment process.

When to Use Ansible Roles

Ansible Roles should be used when:

 Managing Large Codebases: For projects with large and complex playbooks, roles help in organizing and simplifying the code.

 Promoting Code Reusability: When there is a need to use the same configuration across multiple projects or environments.

 Improving Collaboration: In teams where multiple developers or operators are working on the same codebase, roles enhance collaboration by providing a clear structure.

 Automating Repetitive Tasks: For tasks that are repetitive and consistent across different environments, roles ensure standardization and efficiency.

How can we Install a Roles in Ansible:

In Ansible, we have a Ansible Packge Manager that is called “Ansible-galaxy” that is used for roles and connection. It connects to Ansible Galaxy a public repository (https://galaxy.ansible.com/) where developers and organizations share roles and collections. It can also work with private Galaxy servers or local directories.

Modules in Ansible:

In Ansible, Modules is a keyword or predefines small tasks or tool, that is used for performs specific task like.. Install a package or server, copy a file, create a user etc.

Common Modules Examples in Ansible:

 apt, yum, dnf: manage packages

 copy: copy files to remote machines

 file: set file properties

 service: start/stop services

 user: manage users

 command, shell: run shell commands

 git: clone repositories

Example:

- name: Install nginx

apt:

name: nginx

state: present

6.Writing and managing Ansible playbooks

Writing and managing Ansible playbooks is a crucial part of using Ansible effectively. (Write any YAML file) We know that, An Ansible playbook is a YAML file that defines one or more plays or tasks. Each play maps a group of hosts to tasks that define what you want Ansible to do. In Ansible, playbooks can be written and managed in two different ways

1. Simple Structure

2. Advance Structure with ansible roles

1. Simple Structure

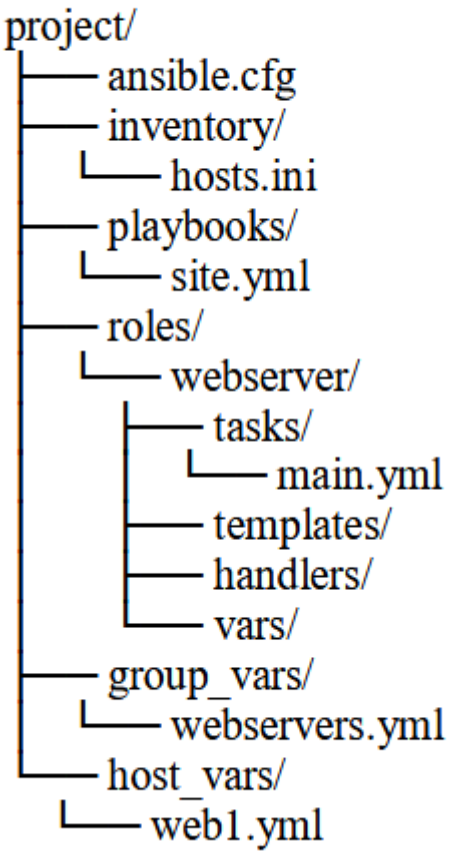
project/

├── inventory.ini

├── playbook.yml

└── ansible.cfg

2. Advance Structure with ansible roles.



Key Component Description

Component	Description
name	Human-readable name for the play or task
hosts	Target hosts from your inventory
vars	Define variables
tasks	List of actions to perform
roles	Include reusable role structures

7.Integrating Ansible with other tools

1. **Integrating with Git**

Purpose: Manage playbooks as code (IaC), mange version control and enable triggers via webhooks.

Integration:

- Store playbooks in Git.
- Trigger Ansible runs via GitHub Actions or GitLab CI/CD.
- Use ansible-pull for pull-based deployment models.

2. **Integrating with Jenkins**

Purpose: Automate infrastructure changes and deployments as part of CI/CD pipelines.

We have an option to write Ansible playbook inside Jenkins pipeline scripts or as a post -build step.

After that it can Trigger playbooks to deploy applications or configure environments.

Command:

ansible-playbook -i inventory site.yml

3. Integrating with Docker and Kubernates

Docker:

Use Ansible to build images, start/stop containers, and manage Docker Compose.

Ansible has built-in Docker modules (e.g., community.docker.docker_container).

Kubernetes:

Deploy and manage Kubernetes resources using Ansible modules or kubectl commands.

Combine with Helm charts via (community.kubernetes.helm) command

4. **Integrating with Cloud Services**

Purpose: Provision and manage cloud resources.

Integration:

 Ansible has cloud-specific modules and collections (e.g., amazon.aws, azure.azcollection).

 Use Ansible to automate EC2 instances, S3, VPC, load balancers, etc.

5. Integrating with Terraform

Purpose: Provision infrastructure (like servers, networks), then configure it with Ansible.

Integration:

 Terraform provisions infrastructure.

 Use null_resource and local-exec or remote-exec to call Ansible.

 Output dynamic inventory from Terraform to feed into Ansible.

6. integrating with monitoring tool like Grafana, Promethus.

Purpose: Automate alert responses or configure monitoring setups.We can Use Ansible with Monitering tool for automate the Alert System in our Application using ansible playbook and webhook.

Unit-3

Part-A

Continuous Integration with Jenkins:

1. **Introduction to Continuous Integration (CI),**
2. **Using Maven for build,**
3. **Jenkins architecture and setup,**
4. **Managing Jenkins plugins and nodes,**
5. **Building and deploying applications using Jenkins,**
6. **Creating and managing Jenkins pipelines,**
7. **Pipeline as code with Jenkins.**

1.Introduction to Continuous Integration (CI)

Definition:-

Continuous Integration (CI) is a software development practice where developers regularly merge their code changes into a central repository, and automated builds and tests are run to quickly detect and fix errors. This helps ensure that the software remains in a working state and reduces integration problems.

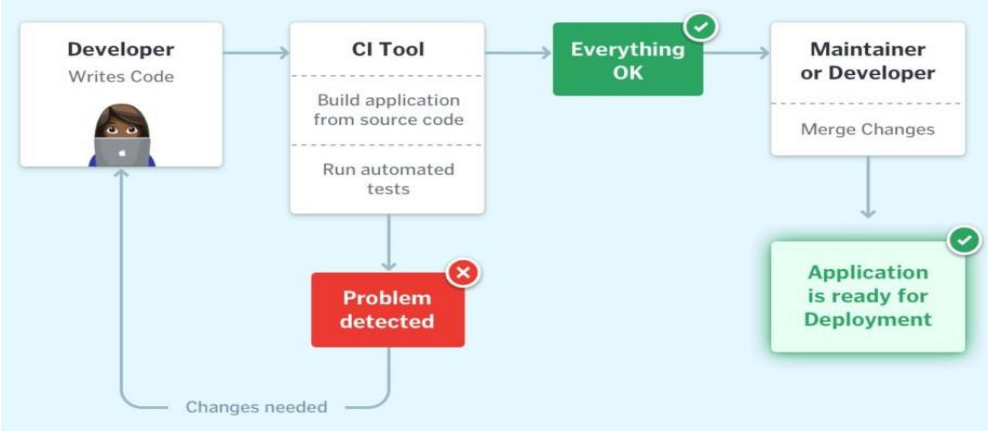
There could be scenarios when developers in a team work in isolation for an extended period and only merge their changes to the master branch once their work is completed. This not only makes the merging of code very difficult, prone to conflicts, and time-consuming but also results in bugs accumulating for a long time which are only identified in later stages of development. These factors make it harder to deliver updates to customers quickly.

With **Continuous Integration**, developers frequently commit to a shared common repository using a version control system such as Git. A continuous integration pipeline can automatically run builds, store the artifacts, run unit tests, and even conduct code reviews using tools like Sonar. We can configure the CI pipeline to be triggered every time there is a commit/merge in the codebase.

CI Workflow

Below is a pictorial representation of a CI pipeline- the workflow from developers checking in their code to its automated build, test, and final notification of the build status.

Once the developer commits their code to a version control system like Git, it triggers the CI pipeline which fetches the changes and runs automated build and unit tests. Based on the status of the step, the server then notifies the concerned developer whether the integration of the new code to the existing code base was a success or a failure.



This helps in finding and addressing the bugs much more quickly, makes the team more productive by freeing the developers from manual tasks, and helps teams deliver updates to their customers more frequently. It has been found that integrating the entire development cycle can reduce the developer's time involved by ~25 - 30%.

2.Using Maven for build

What is Maven?

- Maven is a **build automation tool** mainly for Java projects.
- It helps you **compile code, manage libraries (dependencies), run tests, and package your app** automatically.
- Uses a file called **pom.xml** to configure project details and build steps.

Why Use Maven?

- **Automates repetitive tasks** like compiling, testing, and packaging.
- **Manages dependencies** so you don’t have to download libraries manually.
- Ensures **consistent builds** on any machine.
- Integrates easily with Continuous Integration (CI) tools like Jenkins.

Common command used in builds:

mvn clean install

This cleans old files, compiles code, runs tests, and packages the project.

What is pom.xml?

- The **Project Object Model** file.
- It lists:
  - Project info (name, version)
  - Dependencies (libraries your project needs)
  - Build instructions (plugins, goals)

Install maven

In a Linux OS, We can install maven using "Sudo apt install maven" command and in a windows,for installing the maven we have to install maven environment application or we can access maven using IDE like. Eclipse, net-beans etc.

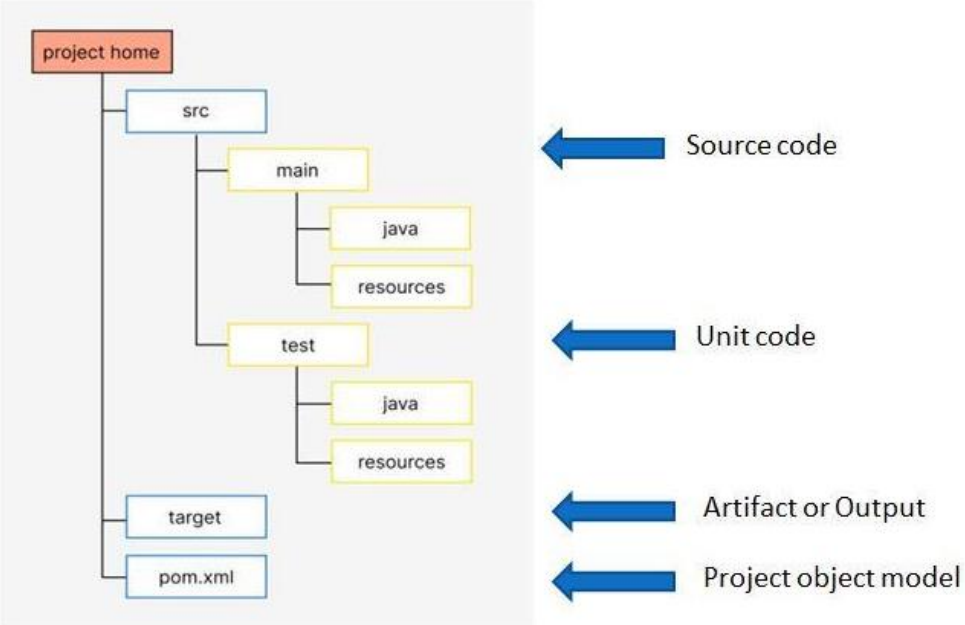
creating a Maven Project

For creating a Maven Project we have to provide two necessary things.

1. Archetype Group Id (e.g. CSD)

2. Artifact ID (e.g. Project Name)

Maven project Architecture



Maven Build Life Cycle

Maven Build life cycle consist some stages

1. Compile: For compile the Source code.

2. Test: Perform Test operation.

3. Package: Build a war file.

4. Install: Install the package in local system.

5. Deploy: Deploy the package to a remote repo

3.Jenkins architecture and setup

What is Jenkins?

- Jenkins is an **open-source automation server** used to automate software development tasks like building, testing, and deploying.
- It supports **Continuous Integration (CI)** and **Continuous Delivery (CD)**.
- Helps teams **integrate code frequently** and detect problems early.

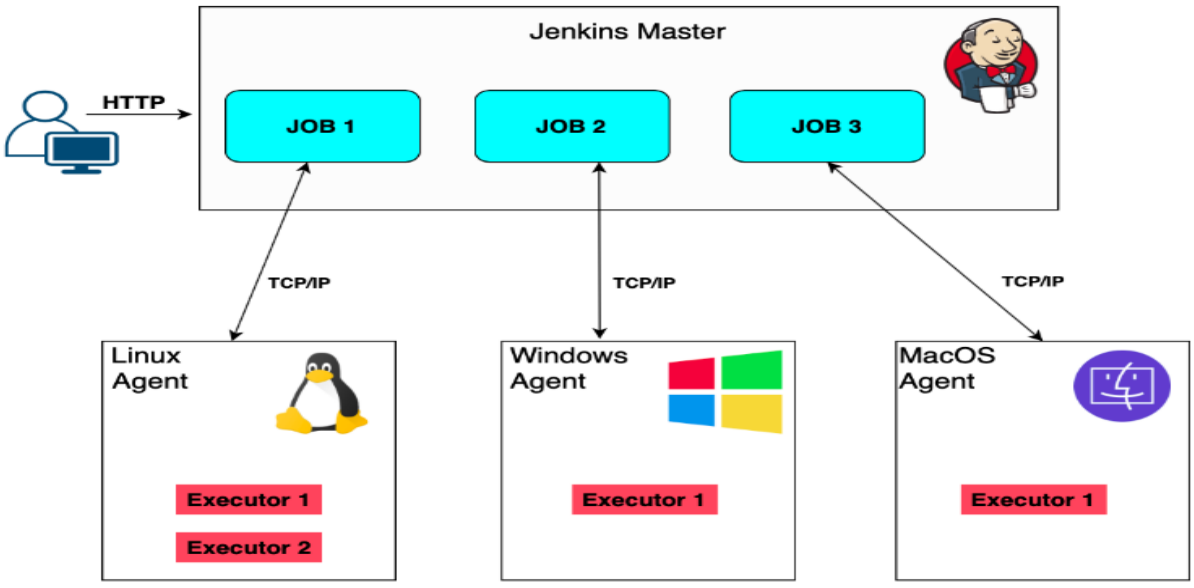


Fig.1.Jenkins Architecture

Jenkins follows a **master-agent (or master-slave)** architecture:

a) Jenkins Master

- The **central server** that manages the build environment.
- Responsible for:
    - Scheduling build jobs.
    - Dispatching builds to agents.
    - Monitoring agents.
    - Aggregating and displaying build results.
- Provides the **user interface (UI)** for configuring jobs and viewing results.

b) Jenkins Agents (or Slaves)

- Machines connected to the Jenkins master.
- Responsible for **executing build jobs** dispatched by the master.
- Can run on different platforms or environments.
- Help **distribute the workload**, speeding up the build process.

How Jenkins Works

1. Developers push code to a repository (e.g., Git).
2. Jenkins master detects the code change (via polling or webhooks).
3. Jenkins master schedules a build job.
4. Master dispatches the build to an available agent.
5. Agent performs the build steps (compile, test, package).
6. Agent sends build results back to the master.
7. Jenkins master displays the build status to users.

Jenkins Setup — Basic Steps

a) Install Jenkins

b) Step-1: first we must have to download the java jdk-11, 17 or 21 and install in our system.

c) Step-2: In a Linux OS, we can install Jenkins using “sudo apt install Jenkins” command. In a

d) windows we can directly download the environment and configure the Jenkins.

e) Step-3: In a Linux we have some command for start the Jenkins servers

f) “sudo Systemctl enable Jenkins”

g) “sudo systemctl start Jenkins”

h) “sudo systemctl status Jenkins”

(or)

- Can be installed on Windows, Linux, macOS, or run via Docker.
- Download from the official Jenkins website: https://jenkins.io

b) Initial Configuration

- Access Jenkins via web browser (default: http://localhost:8080).
- Unlock Jenkins by entering the initial admin password (found in installation logs or files).
- Install suggested plugins (supports many build tools, version control systems, etc.).
- Create the first admin user.

c) Configure Tools and Environment

- Configure JDK, Maven, Git, etc., in Jenkins global tools configuration.
- Set up credentials (like GitHub tokens, SSH keys).

d) Create Your First Job

- Click **New Item**.
- Choose job type (e.g., Freestyle project, Pipeline).
- Configure source code repository.
- Add build steps (e.g., run Maven commands).
- Save and run the job.

4. Managing Jenkins plugins and nodes

Managing Jenkins Plugins

- **Plugins** add new features to Jenkins (e.g., Git integration, build tools, notifications).
- Jenkins comes with many plugins, and you can add more as needed.

How to Manage Plugins:

1. Go to **Manage Jenkins → Manage Plugins**.
2. You will see tabs:
    - **Updates:** Plugins with available updates.
    - **Available:** Plugins you can install.
    - **Installed:** Plugins already installed.
    - **Advanced:** Upload plugins manually or change update settings.
3. To **install plugins**, go to the **Available** tab, select plugins, and click **Install**.
4. To **update plugins**, go to the **Updates** tab, select plugins, and install updates.
5. To **remove plugins**, go to the **Installed** tab, find the plugin, and click **Uninstall**.
6. Sometimes Jenkins requires a **restart** after installing or updating plugins.

Managing Jenkins Nodes (Agents)

- Jenkins master controls everything, but **nodes** (or agents) do the actual build work.
- Nodes can be on different machines or operating systems.
- Using nodes allows parallel builds and running builds in different environments.

How to Manage Nodes:

1. Go to **Manage Jenkins → Manage Nodes and Clouds**.
2. You will see a list of all nodes, including the master.
3. To **add a new node**:
    - Click **New Node**.
    - Enter a name and choose **Permanent Agent**.
    - Configure node details like remote directory, labels, and launch method (e.g., SSH).
    - Save the node.
4. Connect the node based on the launch method:
    - SSH: Jenkins connects automatically.
    - Java Web Start: Run a command on the agent machine to connect.
5. You can **disable** or **delete** nodes if needed.
6. Nodes help distribute build jobs and improve performance.

5.Building and deploying applications using Jenkins

1.What is Building?

- Compiling source code, running tests, and packaging the application.

How Jenkins Builds Applications:

1. **Pull Code:** Jenkins fetches the latest code from a repository (e.g., Git).
2. **Compile Code:** Runs build tools like Maven or Gradle (mvn clean install).
3. **Run Tests:** Executes automated tests to verify code quality.
4. **Package:** Creates the executable file or archive (like a JAR, WAR, or Docker image).
5. **Archive Artifacts:** Saves the build outputs for later use or deployment.

Setting up a Build Job:

- Create a new Jenkins job (Freestyle or Pipeline).
- Configure repository details.
- Add build steps (commands or scripts).Ex- mvn clean install
- Run the job manually or trigger automatically on code changes.

2.deploying applications using Jenkins

Deploying an application using Jenkins and Tomcat automates the process of moving your built web application (usually a WAR file) to a Tomcat server, where it can be accessed and run. After Jenkins builds the WAR file, it can automatically transfer the file to the Tomcat server and deploy it using Tomcat's manager web application or by copying it into Tomcat's deployment directory. This automation saves time, reduces manual errors, and ensures that the latest

application version is always deployed. Jenkins can use plugins like the **Deploy to Container Plugin** to make deployment easier and more integrated, or use custom scripts to copy files and restart Tomcat if needed.

Steps to Deploy Application on Tomcat Using Jenkins

1. **Prepare Tomcat Server:**
    - Install Apache Tomcat on your server or local machine.
    - Enable Tomcat Manager App (make sure user credentials with deploy permissions are set in tomcat-users.xml):

      xml
      CopyEdit
      <user username="jenkins" password="jenkins123" roles="manager-script"/>

    - Note the Tomcat server URL (e.g., http://localhost:8080/manager/text).
2. **Install Jenkins Plugins:**
    - Go to **Manage Jenkins → Manage Plugins**.
    - Install **Deploy to Container Plugin** (for easy Tomcat deployments).
3. **Create a Jenkins Job:**
    - Create a Freestyle project or Pipeline job.
    - Configure source code repository (e.g., Git).
    - Add build steps to compile and package the application (e.g., run Maven clean package to create WAR).
4. **Configure Deployment in Jenkins:**
    - In the post-build section, add **Deploy war/ear to a container**.
    - Enter:
        - WAR/EAR files: **/*.war (path to your built WAR file).
        - Context path (optional): e.g., /myapp.
        - Containers: Choose **Tomcat 7+ Remote**.
        - Credentials: Provide username/password for Tomcat Manager.
        - Tomcat URL: e.g., http://localhost:9090/manager/text.
5. **Save and Run the Job:**
    - Run the job manually or trigger via SCM changes.
    - Jenkins will build the WAR and deploy it automatically to Tomcat.
6. **Verify Deployment:**
    - Access the application in a browser at http://localhost:9090/myapp.
    - Check Jenkins console logs for deployment success messages.

6.Creating and managing Jenkins pipelines

Jenkins Pipeline

A Jenkins Pipeline is a concept that represents the fully automated Continuous Integration (CI) and Continuous Delivery (CD) process for a software using a Jenkinsfile. It consists of a series of steps that define the complete CI/CD workflow, from source code management to final deployment for end users.

Jenkin file

Jenkinfile is a text file that contain the definition and steps of jenkins Pipeline.

There are two types of pipelines in Jenkins

1. Declarative Pipeline: Simplified, structured syntax for easy pipeline creation.

2. Scripted Pipeline: More flexible, written in standard Groovy syntax.

1.Declarative Pipeline:

```
pipeline {

agent any // Runs on any available agent

stages {

stage('Build') {

steps {

echo 'Building the application...'

sh 'mvn clean package'

}

}

stage('Test') {

steps {

echo 'Running tests...'

sh 'mvn test'

}
```

```
}

}

}
```

2.Scripted Pipeline:

```
node {

stage('Checkout') {

echo 'Fetching source code from GitHub...'

git url: 'https://github.com/user/repository.git', branch: 'main'

}

stage('Build') {

echo 'Building the application...'

sh 'mvn clean package'

}

}
```

Setup Process

For setup a pipeline, first we have to add and install a Pipeline plugins from manage jenkins option.

After installation of pipeline plugins it will automatically add a jenkinfile in our git repository.

Steps for Creating a Pipeline

 Step-1: Click on new Item in Dashboard.

 Step-2: Give a Name and select Pipeline

 Step-3: Click OK and navigate to the Pipeline section.

 Step-4: Choose Pipeline Script (inline) or Pipeline from SCM (Jenkinsfile in Git).

 Step-5: Write down the pipeline script according to your need.

 Step-6: Save and run the pipeline.

Managing Jenkins Pipeline

We can Manage jenkins pipeline manually using Manually trigger a pipeline from the Jenkins UI

Dashboard,

And we have option to manage jenkins using automation tool like…

1. Upstream project builds.

2. Webhooks (GitHub/GitLab integration).

3. Poll SCM (H/5 * * * * for every 5 minutes).

7. Pipeline as code with Jenkins

Pipeline as Code is a practice where the entire Continuous Integration/Continuous Delivery (CI/CD) process is defined and managed through a Jenkinsfile. Instead of configuring pipelines manually via the Jenkins UI dashboard, they are written as code, stored in version control (e.g., Git), and executed by Jenkins automatically.

Why Pipeline as a code is important

- Version Control: Pipelines are stored jenkinfile in Git, using that one we can track the
- changes.
- Automation: Pipeline Reduce the Manuel configuration process
- Reproducibility: Ensures consistency across builds.
- Scalability: Easy to maintain and modify as projects grow.
- Collaboration: Developers and DevOps teams can work together on CI/CD pipelines.

Syntax:- of Pipeline as a code

```
pipeline {

agent any

stages {

stage(“Stage_name“) {

steps {

// Source info or command

}

}
```

}

}

Example of Pipeline as a code

For Testing

```
pipeline {

agent any

stages {

stage(„Test“) {

steps {

echo 'Running tests...'

sh 'mvn test'

}

}

}

}
```

For Source Code Management

```
pipeline {

agent any

stages {

stage('Checkout') {

steps {

git url: 'https://github.com/repo.git', branch: 'main'

}

}

}
```

}

For Build an artifact

```
pipeline {
agent any
stages {
stage(„Build') {
steps {
echo 'Building the application...'
sh 'mvn clean package'
}
}
}
}
```

Unit-4

Docker & Kubernates

Introduction to Docker

Docker is a open source tool or platform that allows developers to build, package, and deploy application in a lightweight, portable container. These docker containers bundle and application with all its dependencies, libraries and configuration files, ensuring it runs consistently across different environment.

Features of Docker

- **Containerization:** Divide application into small and portable container
- **Portability**: Containers run anywhere
- **Light-weight:** All containers are lightweight.
- **Scalability:** We can change the size of container.
- **Fast Deployment:** Faster to start and deploy than VMs.

Disadvantage of Docker

1. Complexity at Scale

- Managing containers at scale (hundreds or thousands) can get complicated quickly.
- Requires orchestration tools like Kubernetes, which have their own steep learning curves.

2. Security Concerns

- Containers share the host OS kernel, which can lead to security vulnerabilities if not properly isolated or configured.

3. Tooling Overhead

- Docker might require integration with multiple other tools (CI/CD, logging, monitoring, etc.), adding to the overall DevOps stack complexity.

4. Not Ideal for GUI app

Docker Command

1. Run Command: run a container from a image

```
$ docker run <image_name>
$ docker run --name <container_name>   <image_name>
```

2. Pull Command: To pull a image from docker regitory

```
$ docker pull <image_name>
```

3. Docker PS: Listout the all Docker images

```
$ docker ps
```

4. Docker Stop: To stop a running container

```
$ docker stop <container_ID>
```

5. Docker start: To start or Run a container

$ docker start <container_ID>

6. Docker rm: To delete a container

$ docker rm <container_ID>

7. Docker rmi: To delete a docker image

$ docker rmi <image ID/ image name>

Docker Installation and Setup

Steps of Docker installation in Windows Operating System and Mac Operation System.

Step-1: Open any browser and Visit Docker Official Website.

Step-2: Download windows installer of Windows OS or Drag Docker into your Applications folder for mac OS

Step-3: Run the installer for windows and follow the instruction prompt. But in Mac OS just launch the docker

Step-4: After installation run from Start menu or Whale ICON

Steps of Docker installation Linux

Open linux Terminal and Run the given command

- sudo apt update
- sudo apt install docker.io -y
- sudo systemctl start docker
- sudo systemctl enable docker

Working with Docker Image and Container

Docker Image: A Docker image is a lightweight, standalone, and executable package that includes everything needed to run a piece of software,

Including:

- The code or application itself
- Runtime (e.g., Node.js, Python)
- System tools and libraries
- Dependencies
- Default environment variables
- Any configurations needed

A Docker image is built in layers. Each instruction in a Dockerfile (like RUN, COPY, FROM) creates a new layer. Layers are cached and reused to make building efficient.

- FROM node:22.14.0 #show a node version for base image
- WORKDIR /app #it connect to a js directory file
- COPY . /app #Copy the js file for image

- RUN npm install #Install dependencies
- EXPOSE 8080 #Represent port Number
- CMD ["node" "index.js"] #command for run the App

Docker Container: A Docker container is a running instance of a Docker image. It is a light-weight, Isolated, portable instance.

- Image = The blueprint (like a recipe)
- Container = The dish made from the recipe (alive and running!)

To Run a Docker image into a Container

docker run -d --name my-container -p 3000:3000 my-node-app

- Docker run: Command for Run a image to a container
- -d: Run in detached mode
- --name: container name
- -p: Port number where you want to run

Dockerfile and image Creation

Dockerfile: A Dockerfile is a text file that contains a set of instructions used to build a Docker image. It uses a Domain Specific Language (DSL) to define the steps required to create an image, which can then be used to run containers. Dockerfiles are essential for automating the process of creating Docker images, ensuring consistency and reproducibility across different environments

Structure of Docker file

```
my-app/
├── Dockerfile
├── package.json
├── package-lock.json
└── index.js
```

Set of Instruction of Docker file

- FROM node:18 # Use the official Node.js base image
- WORKDIR /app # Set the working directory inside the container
- COPY package*.json ./ # Copy package.json and install dependencies
- RUN npm install
- COPY . . # Copy the rest of your app's code
- EXPOSE 3000 # Expose the port No.
- CMD ["node", "index.js"] # Command to run your app

➢ For image Creation in Docker from a docker file, we have a command only we have to navigate the file directory and run the given command for image creation

docker build -t my-node-app .

➢ And After that you can check your image is created or not using given command

docker images

Docker Compose and Docker Swarm

Docker Compose: Docker Compose is a tool that allows developers to define and run multi-container Docker applications using single YAML file. It is most commonly used in development, testing, and staging environments. Its primary purpose is to simplify the orchestration of containers that work together within a single application.

Docker Compose YAML STRucture

```
version: '3'
services:
  web:
    image: node:18
    volumes:
      - .:/app
    working_dir: /app
    ports:
      - "3000:3000"
    command: node index.js
  db:
    image: mongo
    ports:
      - "27017:27017"
```

How Docker Compose Works

Imagine an application that has a frontend, backend, and database. Using Docker Compose, you can define all three components and their relationships in one YAML file. E.g. services, networks, and volumes in "docker-compose.yml"

After that we can directly run everything using a single command

"docker-compose up"

Features of Docker Compose

- **Declarative Configuration:** Uses a YAML file (commonly docker-compose.yml) to configure application services. This file specifies how to build the container images, configure volumes, networks, and set environment variables.
- **Multi-Container Setup**: Define and manage all the containers needed for an application in one file.
- **Effortless Networking**: Provides an isolated network where containers can communicate with each other by default, using their service names as DNS.
- **Portability:** You can easily share the configuration file, allowing the application to run consistently across different environments.
- **Command-Line Interface:** Simple commands like docker-compose up to start services or docker-compose down to stop them.

Docker Swarm

Docker Swarm is a container orchestration tool, but unlike Docker Compose, it is designed for managing clusters of Docker nodes in production environments. It provides fault-tolerant and distributed deployments for services at scale.

Docker Swarm Command for init and Deploy.

- Initialize a Swarm: docker swarm init
- Deploy a Stack: docker stack deploy -c <compose-file> <stack-name>

Features of Docker Swarm

- **Cluster Management:** Converts multiple Docker nodes into a single logical cluster. Nodes can be designated as Manager Nodes (orchestrating the cluster) or Worker Nodes (executing tasks).
- **Service Scaling:** Easily scale services up or down by specifying the number of replicas.
- **Load Balancing:** Automatically balances traffic among replicas of a service.
- **High Availability:** Ensures services remain active, even if a node goes offline.
- **Secure by Default:** Uses TLS encryption for communication between nodes in the cluster.
- **Declarative Configuration:** Define desired service states (e.g., how many replicas) in YAML files or directly via the Swarm CLI.

Differences

Feature	Docker Compose	Docker Swarm
Purpose	Local development and testing	Production-grade container orchestration
Scale	Single-node setups	Multi-node clusters
Networking	Local network	Distributed overlay network
CLI Simplicity	Simple commands for basic needs	Comprehensive commands for orchestration
Fault Tolerance	Not inherently fault-tolerant	Built-in high availability

Orchestration with Kubernates

Introduction to Kubernates

Kubernetes is a Open Source Automated Orchestration tool that is used to automate the deployment, scaling, and management of containerized applications. Originally developed by Google and now maintained by the Cloud Native Computing Foundation (CNCF), Kubernetes enables you to run containers at scale across multiple machines, ensuring your applications are resilient, efficient, and highly available.

Features of Kubernetes

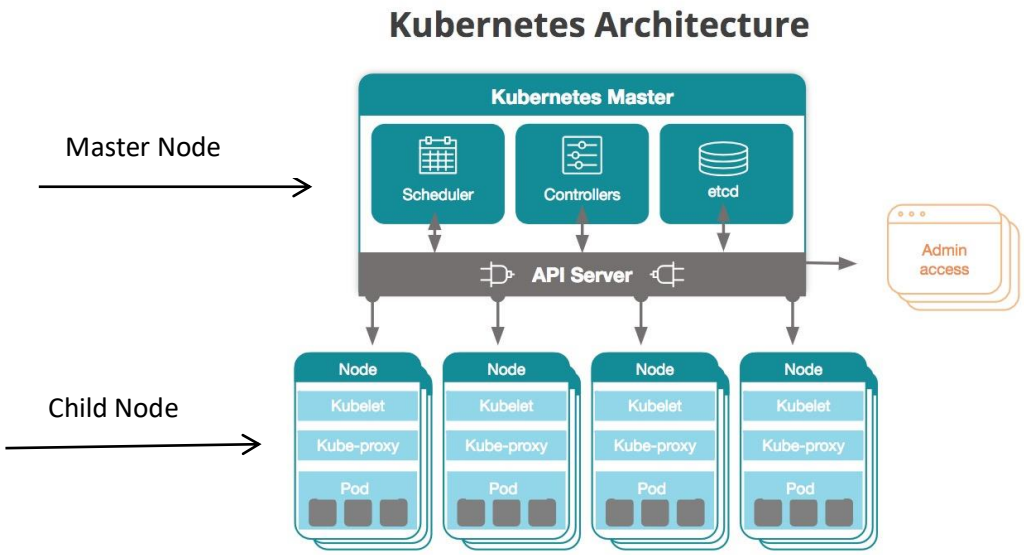
- **Container Orchestration:** Kubernetes schedules and manages containers across a cluster of machines. It ensures the right containers run on the right nodes and handles their lifecycle.

- **Service Discovery and Load Balancing:** It provides networking capabilities to automatically expose services. Kubernetes can load-balance traffic and route requests to healthy containers.
- **Scaling:** Automatically scales your application up or down based on demand. This can be done manually or via auto-scaling rules.
- **Self-Healing:** If a container fails, Kubernetes restarts it. If a node crashes, Kubernetes reschedules containers to other healthy nodes.
- **Declarative Configuration:** Kubernetes uses YAML or JSON configuration files to define the desired state of your system, such as how many replicas of an application should run.
- **Storage Orchestration:** Dynamically mount and manage storage for containers, including support for persistent storage.
- **Rollouts and Rollbacks**: Manage updates to your application with minimal downtime. Roll back to a previous version if something goes wrong.

Why Organization Use Kubernates

- Kubernetes is ideal for large-scale containerized applications that require high availability, fault tolerance, and scalability.
- It works seamlessly with cloud providers (like AWS, Azure, GCP) and can also run in on-premises environments.
- Many organizations rely on Kubernetes for modern microservices-based applications.

Kubernates Architecture and Components



Components

Master Node: The brain of the cluster, responsible for managing the desired state of the system.

Key components:

- **API Server:** Exposes the Kubernetes API.
- **Controller Manager:** Ensures the desired state matches the actual state.
- **Scheduler:** Allocates resources by assigning containers to nodes.
- **etcd:** A distributed key-value store used for storing configuration and state data.

Worker Nodes: Run the containerized applications and are managed by the master.

Key components:

- **Kubelet:** Ensures containers are running.
- **Kube:** proxy: Manages networking.
- **Pods:** The smallest deployable units, each containing one or more containers.

Setting up a Kubernates Cluster

Cluster: A Kubernetes Cluster is a collection of machines or node that work together to run and manage containerized applications. It consists of two main types of nodes: The Control or Master Node and Worker Nodes. Together, they provide a scalable and fault-tolerant environment for deploying, managing, and scaling containers.

- Setup a kubernates Cluster can vary on different types of Machines like. Local Machine, Cloud services and Virtual Machines.
- So In simple term, we can build a Kubernates cluster based on our need or Machines. And we have to setup an Environment for each type of machine. We have three common types of Setup
  - Cloud Based Setup.
  - Minikube based Setup. (For local system)
  - Using Kubeadm based Setup. (Bare Metal/Best for production)

1. Cloud Based: Cloud Based Cluster are fully managed or it can be created by Cloud web UI or CLI.
We have many types of Cloud like.

- AWS → AWS EKS (Elastic Kubernates Services)
- Azure → Azure AKS (Azure Kubernates Services)
- Google → GKE (Google Kubernates Engine)

Setup on GKE:

For Create a Cluster with Zone

- Step-1: gcloud container clusters create my-cluster --zone us-central1-a
  After Creating a Cluster we have to create a Pod
- Step-2: Kubectl run --image tomcat webserver
  After creating cluster and Pods we can check our cluster is working or not using given command
- Step-3:
  - kubectl get nodes
  - kubectl get pods -A
  - kubectl create deployment nginx --image=nginx
  - kubectl expose deployment nginx --port=80 --type=NodePort
  - kubectl get svc

Note: In a pods we have an option to define our steps or machine configuration using .YAML file like. Container type, Container name and Target port number

Example:

```
apiVersion: v1
kind: Pod
metadata:
  name: jenkins-pod
spec:
  containers:
  - name: myjenkins
    image: jenkins/jenkins
    ports:
    - containerPort: 8080
      hostPort: 8080
```

Setup on Minikube:

Minikube runs a single-node Kubernetes cluster on your local machine. For execute a Minikube on local system we need Docker Application, Kubectl CLI, Minikube application.

Steps:

- Open Minikube and start the server
- Open Kubectl CLI and Run Given Command
- kubectl create deployment hello-minikube --image=k8s.gcr.io/echoserver:1.4
- kubectl expose deployment hello-minikube --type=NodePort --port=8080
- Now Open any browser and access your cluster using provided port number.

Managing Pods, Deployment and Services

Managing Pods in Kubernetes

Pods are the smallest deployable units in Kubernetes. They encapsulate one or more containers, along with shared storage and networking resources**.**

Pod Management task in kubernates.

1. Creating a Pod:

In Kubernates, we can create a pod with or without task, if we want to create a pod with some task kubernates provide yml file to define one or more than one task in single file.

YML file Structure

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: nginx-container
    image: nginx:latest
```

Apply the configuration in yml file → kubectl apply -f my-pod.yaml

2. Viewing Running Pods

- kubectl get pods

3. Deleting a Pod

- kubectl delete pod <pod-name>

Managing Deployments in Kubernetes

Deployments ensure that an application is running the correct number of Pods and allow for automated updates, scaling, and rollbacks.

1. Creating a Deployment: For creating a deployment in kubernates we have a yml script to assigned a deployment task to kubernates server after that it provide us a automation in deployment.

YML File

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: nginx-container
        image: nginx:latest
        ports:
        - containerPort: 80
```

Apply the Deployment → kubectl apply -f my-deployment.yaml

2. Scaling Deployments : We can Scale up or down our deployement dynamically using given command

- kubectl scale deployment my-deployment --replicas=5

3. Rolling Updates: We can perform rolling update task of our deployment using given command

- kubectl set image deployment/my-deployment nginx-container=nginx:1.21

4. Deleting a Deployment: We can delete our deployment using given command

- kubectl delete deployment my-deployment

Managing Services in Kubernetes

In Kubernates, Services expose Pods internally within the cluster or externally to users. They provide network accessibility and load balancing for applications.

Types of Services

- ClusterIP (default): Exposes a Service only inside the cluster.
- NodePort: Exposes a Service on each Node's IP, accessible externally.
- LoadBalancer: Uses a cloud provider's load balancer to distribute traffic.
- ExternalName: Maps a Service to an external domain name.

1. For Creating a Service: For create a service in kubernates we have a yml structure.
Structure:

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: NodePort
```

2. Apply the Service

- kubectl apply -f my-service.yaml

3. For view a Services

- kubectl get services

4. Deleting a Service

- kubectl delete service my-service

Unit-5
Prometheus, Grafana and Selenium

Prometheus

Prometheus is an open-source, powerful system monitoring, Alerting, and performance management tool. Originally, it is developed by SoundCloud, But at this time it is managed by Cloud Native Computing Foundation (CNCF) like kubernates.

Features of Prometheus

Time-series Database

Prometheus stores all data as time-series data or metrics. which means it tracks how values change over time, all data is associated with a timestamp, metric name, and labels.

Multi-dimensional Data Model

Prometheus organizes data using a multi-dimensional system with key-value pairs called labels. This makes it highly flexible for querying.

Pull-Based Data Collection

Prometheus actively pulls metrics from configured endpoints using HTTP, rather than relying on agents to push data.

PromQL: (Prometheus query language)

Prometheus uses its own powerful and expressive query language called PromQL for querying collected metrics.

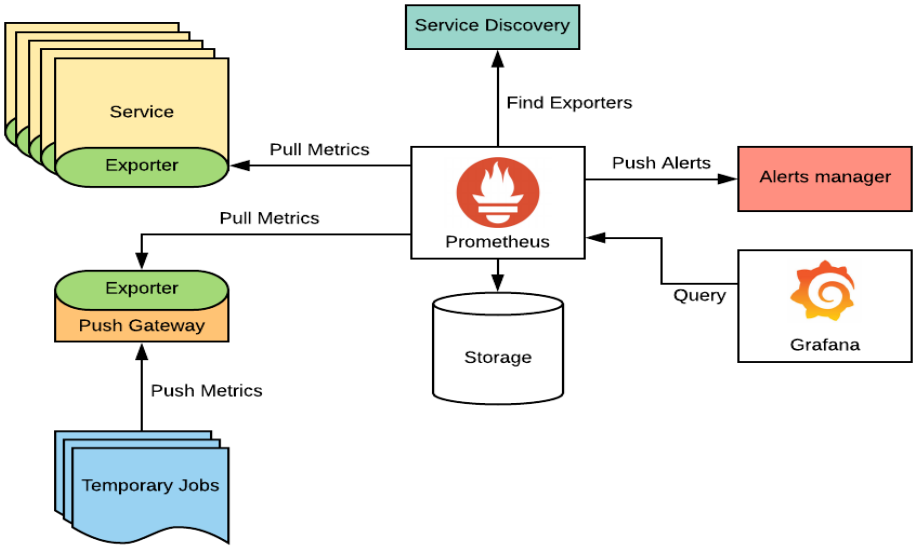
Standalone Architecture

Prometheus operates independently, without dependencies on external storage systems.

Alerting System

Prometheus integrates with alerting tools to notify users about critical conditions.

Prometheus Architecture



Why Learn Prometheus

1. Comprehensive Monitoring and Alerting

Prometheus specializes in collecting and analyzing time-series data, making it invaluable for:

- Monitoring the health and performance of applications, servers, and networks.
- Setting up alerting systems to notify you about potential issues before they impact users.

2. PromQL – A Powerful Query Language

Prometheus Query Language (PromQL) allows for:

- Writing highly flexible and detailed queries to extract metrics.
- Generating precise insights, which can be visualized using tools like Grafana.

3. Open-Source and Cost-Efficient

- Prometheus is completely free and open-source, which means it's accessible to everyone.
- Organizations of all sizes adopt Prometheus to save costs while achieving enterprise-grade monitoring capabilities.

4. Versatility and Custom Metrics

With Prometheus, you can monitor not just infrastructure metrics (like CPU, memory, and disk) but also custom application metrics. Developers can instrument their code to track key metrics like:

- API latency.
- Number of user requests.
- Error rates.

5. Career Growth Opportunities

- Expertise in Prometheus is in high demand across DevOps, SRE, and IT operations roles.
- By learning Prometheus, you not only develop practical monitoring skills but also position yourself as a critical asset in maintaining reliable systems.

6. Integration with Tools Like Grafana

Prometheus works effortlessly with visualization tools like Grafana to create dashboards. This combination makes it easier to:

- Spot trends.
- Share insights with stakeholders.
- Debug production issues visually.

Infrastructure monitoring in Prometheus

Infrastructure monitoring in Prometheus involves collecting, analyzing, and visualizing metrics from servers, virtual machines, containers, and other system resources to ensure optimal performance and identify potential issues in real time.

Why Infrastructure Monitoring Matters:

We know that, System Infrastructure is the backbone of any application. So Monitoring helps us to:

- Track system health (e.g., CPU, memory, disk usage).
- Predict and prevent failures.
- Optimize resource usage.
- Diagnose issues quickly and ensure high availability of services.

Prometheus provides a powerful solution for this by collecting time-series data from infrastructure components and making it available for analysis through querying and alerting.

How Prometheus Monitors Infrastructure

We know that prometheus is a System Monitoring and Alerting tool. So It collects data from system components like servers, containers, and network devices using exporters, which expose metrics in a format Prometheus can scrape via HTTP. Key exporters include Node Exporter for hardware metrics (CPU, memory, disk, network) and cAdvisor for container monitoring.

The collected metrics are stored in Prometheus' time-series database, organized with labels and timestamps for flexible querying through PromQL. After that, Metrics can be visualized using dashboards created in tools like Grafana, that can ensure infrastructure reliability, scalability, and performance and improe to solve real time problem

Alerting and Alert Receiver in Prometheus

Alerting

Alerting in Prometheus enables you to define conditions based on the metrics collected, and when these conditions are met, alerts are triggered. These alerts notify users or systems so that they can take timely actions to resolve potential issues before they escalate.

How Alerting Works in Prometheus:

Alerting Rules

- Alerts are created using alerting rules in Prometheus' configuration file (prometheus.yml) or separate rule files.
- Each rule specifies a condition, an evaluation interval, and labels for the alert.

Alerting YAML File Structure

```
groups:
  - name: example-alert
    rules:
      - alert: HighCPUUsage
        expr: rate(node_cpu_seconds_total{job="node-exporter"}[5m]) > 0.85
        for: 2m
        labels:
          severity: critical
        annotations:
          summary: "High CPU usage detected"
          description: "CPU usage is above 85% for more than 2 minutes."
```

Alert manager

The Alert manager is a companion tool to Prometheus that handles alerts and manages notification delivery to receivers.

Alert Receivers

Alert receivers are the endpoints that Alertmanager sends notifications to. Common types of receivers include:

Email:
Sends alerts as email notifications.

Messaging Services:
Integrates with platforms like Slack, Teams, or PagerDuty for real-time communication.

Webhook:
Sends alerts to custom HTTP endpoints for integration with other systems.

SMS or Phone Calls:
Integrates with third-party tools to deliver alerts via SMS or automated phone calls.

Grafana

Grafana

Grafana is an open-source data analytics and data visualization tool, that is designed for monitoring and managing system metrics data, It also used for creating interactive dashboards, exploring data, and observing trends across multiple sources like, Node, Web-based source and medical machines.

Features of Grafana

- Grafana provide Interactive Dashboard and it allows us to design and customized with chat, graph and other visualization for real-time monitoring of metrics.
- Grafana Provide different types of Data source integration. It means we can connect to a wide variety of data-source like. Prometheus, MySQL, SqlLite, InfluxDB, Elasticsearch etc..
- Grafana provides an alerting system. You can set rule-based alerts on dashboards, and when the specified conditions are met, it will send notifications via SMS, email, Slack, and other channels.
- Grafana provide intuitive query editor where we can write any SQL query like, PromQL, MySQL etc for retrieving and transforming data from sources.
- Grafana Provide Predefine templates and plugins, using this we can Enhance the Functionality of Grafana for data visualization.

Creating Grafana Dashboard

Creating a Grafana dashboard helps us to visualize and moniter the data from various data source like MySQL, SQL-Lite etc.

We have some steps for creating a Grafana Dashboard.

Step-1: First Install Grafna in our System or on Cloud using given Command

- sudo apt-get update
- sudo apt-get install -y grafana
- sudo systemctl start grafana-server
- sudo systemctl enable grafana-server

Step-2: Now Open your Grafana tool on given port using localhost:<port> and login using default user_id and password

- Username: Admin
- Password: Admin

Step-3: After that Add data source using given sub-steps (Note: You can add various data source in Grafana).

To add a data source:

- Navigate to Configuration → Data Sources.
- Click Add data source and select a database (e.g., Prometheus).
- Provide connection details (e.g., Prometheus URL: http://localhost:9090).
- Click Save & Test to ensure the data source is successfully added.

Step-4: After adding data source create a Dashboard using dashboard option

- Go to Dashboards → Create → New Dashboard.
- Click Add a new panel to start creating visualizations.

Step-5: After creating a dashboard and panel you have to configure the panel according to data-source and here you can add and configure data visualization type like. Graph, table, heat map etc. and you can also set some adjust setting like. Title, Legend, Time Range, Alert etc..

(Note: You can Add Multiple Panels)

Step-6: After configure the panel you can set a alert system and threshold **(Optional)**.

Step-7: Now Save a Dashboard and share with your team.

Grafana API and Auto Healing

Grafana API

Grafana provides a robust REST API that allows programmatic access to its functionalities, including creating dashboards, managing users, querying data sources, and automating tasks.

Features of Grafana API

1. Dashboard Management
Create, update, delete, and retrieve dashboards via API calls.

2. User & Authentication Management
Manage users, roles, and authentication tokens.

3. Data Source Integration
Add, update, and remove data sources programmatically.

4. Alerts & Notification Channels
Configure alerts and set up notification channels via API.

5. Organizations & Permissions
Organize Grafana users and configure permissions dynamically.

Note: Grafana API uses API tokens for authentication, ensuring secure and controlled access to its endpoints.

Some API Calls Command in Grafana

1. **Create a dashboard**
   curl -X POST http://localhost:3000/api/dashboards/db -H "Content-Type: application/json" -d @dashboard.json
2. **Get all dashboards**
   curl -X GET http://localhost:3000/api/search
3. **Retrieve metrics from Prometheus**
   curl -X GET http://localhost:9090/api/v1/query?query=node_cpu_seconds_total

Auto Healing in Grafana

Grafana Doesn't have any auto healing system or capabilities, it can be integrated with monitoring tools (e.g., Prometheus, Kubernetes) to trigger self-healing mechanisms, and these tools provide a auto-healing facilities to grafana.

Auto-healing Mechanism in Monitoring System

1. Detection

- Grafana monitors infrastructure for failures using real-time metrics.
- Prometheus alerts Grafana when a system metric exceeds defined thresholds.

2. Alerting & Incident Response

- Grafana alerts are sent to Alertmanager or other notification tools.
- Alerts trigger automated workflows to handle incidents.

3. Automated Remediation (Self-Healing)

- Kubernetes can automatically restart failing pods when an issue is detected.
- Infrastructure automation tools like Ansible or Terraform can respond to alerts by adjusting configurations.

Selenium

Selenium

Selenium is an open-source testing tools, that is used to automate web browsers. the primarily use of selenium is automated testing of web applications. However, it's also used for web scraping and browser automation tasks.

Features of Selenium

1. Selenium is used for Cross-Browser Compatibility: e.g. Chrome, Firefox, Edge, safari etc

2. Cross-Platform Support: e.g. Windows, Linux, MacOS.

3. Supports Many Programming Languages for Script: e.g. Java, JavaScript, Python, Ruby, C# etc..

4. Selenium can integrate with Multiple Framework: e.g. Junit, TestNG(java). PyTest(python). NUnit(C#)

5. Selenium can Supports Headless Browser Testing: e.g. CI/CD Pipeline, testing in environment without display.

6. Selenium Support Mobile Testing: e.g. Appium, Selendroid

7. Selenium can integrate with Cloud-Based Test Execution: e.g. Sauce Labs, BrowserStack, LambdaTest

8. Selenium is Open Source(Free) and Strong Community Driven

Testing Backend Integration points

Testing backend integration points is essential to ensure that different system components communicate and function correctly. It helps identify issues related to APIs, databases, third-party services, and internal system dependencies.

Backend Integration Testing

Backend integration testing involves verifying data flow, interactions, and communication between different services, APIs, and databases in a system. It ensures that backend components work together seamlessly without errors.

Backend Integration Testing Points

1. API Testing:

- Ensures that REST, GraphQL, or SOAP APIs function correctly.
- Verifies request/response formats, status codes, headers, and authentication mechanisms.

2. Database Testing:

- Validates data storage, retrieval, updates, transactions, and integrity.
- Checks for performance issues in queries and indexing.

3. Message Queues & Event Systems:

- Tests communication via Kafka, RabbitMQ, AWS SQS to ensure messages are correctly published and consumed.

4. Third-Party Services:

- Ensures integration with payment gateways, authentication providers, analytics tools, etc.

5. Authentication & Security:

- Verifies login, token generation, encryption, and authorization policies.

Types of Backend Integration Tests

1. Unit Tests

- Focus on individual components using mocking or stubbing.
- Example: Testing a function that retrieves data from a database.

2. Functional Tests

- Validate APIs and backend logic against expected outcomes.
- Example: Sending requests and checking JSON responses.

3. Load & Performance Testing

- Tests how backend systems handle high volumes of requests.
- Tools like JMeter, Locust, or k6 measure response times and scalability.

4. End-to-End Tests

- Validate interactions across multiple backend services.
- Example: Checking database updates after an API call.

5. Security Tests

- Ensure APIs and databases are protected from vulnerabilities (SQL injection, unauthorized access).
-

Test-Driven Development

Test-Driven Development (TDD) is a software development methodology where tests are written before the actual implementation of the code. The idea is to create tests that define the expected behaviour of a function or feature and then write the minimal amount of code required to pass those tests.

Test-Driven Development is based on a simple cycle called Red-Green-Refactor:

- **Red→** Write a failing test for a function that does not exist yet.
- **Green→** Implement the function minimally to make the test pass.
- **Refactor→** Optimize the code without changing its functionality.

This iterative approach ensures high code quality, bug prevention, and continuous validation.

Steps that are exist in Test-Driven Development

- Write a test. e.g. Like function for add two number
- Run the test (Failing Stage - RED). It will check add function is exist or not.
- Write minimal code to pass the test (GREEN)
- Run the test again
- Refactor the code
- Repeat for other functions

Programming Languages Tools for Test-Driven Development

- **Python** → pytest, unittest
- **JavaScript** → Jest, Mocha
- **Java** → JUnit
- **C#** → xUnit, NUnit
- **Go** → Testify

Benefits of Test-Driven Development

- **Early Bug Detection→** Writing tests first helps catch problems before they reach production.
- **Improved Code Quality→** Forces developers to write cleaner, more modular code.
- **Faster Debugging→** When a test fails, it's clear which part of the code is broken.
- **Encourages Better Design→** Encourages writing small, testable, and reusable functions.
- **Confidence in Changes→** Code modifications can be done safely without fear of breaking functionality.

REPL Driven Development

REPL-Driven Development (RDD) is an interactive programming approach where developers write, test, and refine code in a Read-Eval-Print Loop (REPL) environment. This method allows for immediate feedback, making it a highly efficient way to develop software.

REPL stands for

- **Read →** Accepts user input (code snippets).
- **Evaluate →** Executes the code in a live runtime.
- **Print →** Displays the result instantly.
- **Loop →** Repeats the process, allowing continuous interaction.

REPL environments exist for many languages, like.

- Python (python shell)
- JavaScript (node REPL)
- Ruby (irb)
- Clojure (clojure REPL)
- Scala (scala interactive shell)

REPL-Driven Development Works and Workflow

REPL-Driven Development focuses on incremental coding, where developers experiment with functions, debug issues, and refine logic in real time before committing changes to files.

Basic Workflow

- **Start the REPL→** Open an interactive shell for the programming language.
- **Write a small function→** Immediately execute to check the output.
- **Iterate on improvements→** Modify the function and test again.
- **Debug in real-time→** Identify mistakes and fix them instantly.
- **Save finalized code→** Store well-tested logic into actual source files.

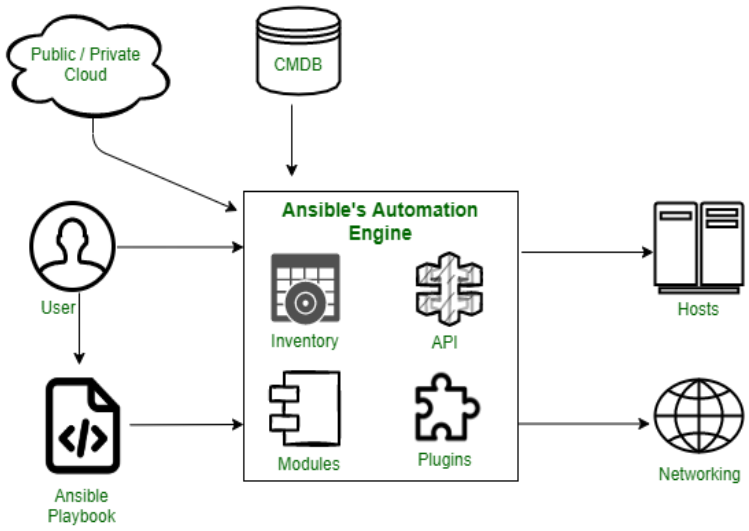
Benefits of REPL-Driven Development

- **Rapid Prototyping→** Quickly try ideas and refine them before writing full applications.
- **Immediate Feedback→** Instantly see errors or outputs, reducing debugging time.
- **Interactive Debugging→** Modify functions while the program is running.
- **Better Understanding of Language Features→** Experiment with syntax and libraries in real time.

Set-1

1.List the core components of Ansible architecture.

Ansible Architecture components:



- **Inventories** – Ansible inventories are lists of hosts with their IP addresses, servers, and databases which have to be managed via an SSH for UNIX, Linux, or Networking devices, and WinRM for Windows systems.

- **APIs –** Application Programming Interface or APIs are used as a mode of transport for public and private cloud services.

- **Modules** – Modules are executed directly on remote hosts through playbooks and can control resources like services, packages, files, or execute system commands. They act on system files, install packages and make API calls to the service network. There are over 450 Ansible that provide modules that automate various jobs in an environment. For example, Cloud Modules like Cloud Formation create or delete an AWS cloud formation stack.

- **Plugins –** Plugins are pieces of code that augment Ansible's core functionality and allow executing Ansible tasks as a job build step. Ansible ships with several handy plugins and one can also write it on their own. For example, Action plugins act as front-ends to modules and can execute tasks on the controller before calling the modules themselves.

- **Networking –** Ansible uses a simple, powerful, and agent-less automation framework to automate network tasks. It uses a separate data model and spans different network hardware.

- **Hosts** – Hosts refer to the nodes or systems (Linux, Windows, etc) which are automated by Ansible.

- **Playbooks –**
  Playbooks are simple files written in YAML format which describe the tasks to be executed by Ansible. Playbooks can declare configurations, orchestrate the steps of any manual ordered process and can also launch various tasks.

- **CMDB –**
  It stands for Configuration Management Database (CMDB). In this, it holds data to a collection of IT assets, and it is a repository or data warehouse where we will store this kind of data, and It also defines the relationships between such assets.

- **Cloud –**
  It is a network of remote servers hosted on the internet to store, manage and process data instead of storing it on a local server.

2. Explain the difference between a Docker image and a Docker container.

Difference between Docker Images and Containers

Docker Image	Docker Container
It is a blueprint of the Container.	It is an instance of the Image.
Image is a logical entity.	The container is a real-world entity.
Images are created only once.	Containers are created any number of times using an image.
Images are immutable. One cannot attach volumes and networks.	Containers change only if the old image is deleted and a new one is used to build the container. One can attach volumes, networks, etc.
Images do not require computing resources to work.	Containers require computing resources to run as they run with a Docker Virtual Machine.
To make a docker image, you have to write a script in a Dockerfile.	To make a container from an image, you have to run the “docker run <image>” command
Docker Images are used to package up applications and pre-configured server environments.	Containers use server information and a file system provided by an image in order to operate.
Images can be shared on Docker Hub.	It makes no sense in sharing a running entity, always docker images are shared.
There is no such thing as a running state of a Docker Image.	Containers use RAM when created and in a running state.
An image must not refer to any state to remove the image.	A container must be in a running state to remove it.

Docker Image	Docker Container
One cannot connect to the images as these images are like snapshots.	In this, one cannot connect them and execute the commands.
Sharing of Docker Images is possible.	Sharing of containers is not possible directly.
It has multiple read-only layers.	It has a single writable layer.
These image templates can exist in isolation.	These containers cannot exist without images.

Note:-Write Any 10 points in exam.

3. Describe how Kubernetes Services help in exposing pods to external or internal traffic.

Introduction to Kubernetes Services

In Kubernetes, **pods** are temporary and can be replaced or restarted at any time. Each pod has its own IP address, but that IP is not permanent. To provide a **stable way to access pods**, Kubernetes uses **Services**.

A **Service** in Kubernetes is an abstraction that defines a **logical set of pods** and a policy by which to **access them**—either internally (within the cluster) or externally (outside the cluster).

2.Types of Kubernetes Services

Kubernetes supports different types of services based on how and where you want the traffic to be routed:

- **ClusterIP (default)**: Exposes the service inside the cluster only.
- **NodePort**: Exposes the service on a port on each node, allowing external access.
- **LoadBalancer**: Exposes the service through an external cloud-based load balancer.
- **ExternalName**: Maps the service to an external DNS name (for accessing external services).

3. How Services Work

- Services **select pods using labels**.
- Kubernetes automatically creates a **DNS entry** for each service so that it can be accessed by name.
- A **service proxy (kube-proxy)** runs on each node to route traffic to the appropriate pod behind the service.

4. Example: Service YAML to Expose a Pod Internally

yaml
CopyEdit

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
```

```
spec:
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
  type: ClusterIP
```

This service routes traffic from port 80 of the service to port 8080 of matching pods labeled app: my-app.

4.Explain Grafana API and Auto healing in detail.

1. Grafana API

The **Grafana API** is a set of HTTP endpoints that allows users to interact programmatically with Grafana, enabling automation and integration of various Grafana resources like dashboards, data sources, and users. Key features include:

1. **Authentication**: It uses **API keys** or **basic authentication** to secure access.
2. **Dashboard Management**: You can create, update, and retrieve dashboards via API endpoints. This is useful for automating dashboard setup.
3. **Datasources & Alerts**: It allows management of data sources and alert configurations. For example, you can add a new data source or set up alert rules.

Example:

- To create a new dashboard, you can send a **POST** request to /api/dashboards/db with the dashboard's JSON configuration.

The Grafana API helps in automating monitoring setups, integrating Grafana with CI/CD pipelines, and managing resources programmatically.

2. Auto Healing in Kubernetes

Auto healing in Kubernetes refers to the system's ability to automatically detect and recover from failures, ensuring the application remains in the desired state without manual intervention. Key components include:

1. **ReplicaSets**: Ensures that the desired number of Pod replicas are running. If a Pod fails, it automatically creates a new one to maintain the number of replicas.
2. **Health Checks**: Kubernetes uses **liveness** and **readiness probes** to monitor Pod health. If a Pod fails a liveness probe, it is restarted automatically.

3. **Pod Disruption Budgets (PDB)**: Controls how many Pods can be disrupted during voluntary actions (like rolling updates), ensuring application availability.

Example: If a web server Pod crashes, the **ReplicaSet** will automatically launch a new Pod to replace it, ensuring high availability.

5.Write a Selenium test script snippet that checks if a login form returns an error on invalid credentials.

Below is a simple **Selenium WebDriver** test script snippet in **Python** that checks if a login form returns an error when invalid credentials are entered. This script assumes you have a basic login form with fields for username and password, and an error message is displayed when the credentials are invalid.

Prerequisites:

- Install Selenium: pip install selenium
- Ensure you have the appropriate **WebDriver** (e.g., ChromeDriver) for your browser.

Selenium Test Script for Invalid Login:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
import time

# Set up WebDriver (using Chrome in this example)
driver = webdriver.Chrome(executable_path='/path/to/chromedriver')  # Update with your ChromeDriver path

# Open the login page
driver.get('https://yourwebsite.com/login')  # Replace with your login URL

# Find the username and password fields
username_field = driver.find_element(By.ID, 'username')  # Update with the correct element ID
password_field = driver.find_element(By.ID, 'password')  # Update with the correct element ID

# Find the submit button (Assuming it's a button with id 'login_button')
login_button = driver.find_element(By.ID, 'login_button')  # Update with the correct element ID

# Input invalid credentials
username_field.send_keys('invalid_user')  # Invalid username
password_field.send_keys('wrong_password')  # Invalid password

# Submit the form
login_button.click()

# Wait for a moment to allow the error message to appear
time.sleep(2)
```

```
# Check if the error message is displayed (Assuming the error message has a specific class 'error')
error_message = driver.find_element(By.CLASS_NAME, 'error')  # Update with the correct class or selector

# Assert if the error message is present
assert error_message.is_displayed(), "Error message is not displayed for invalid login credentials."

# Print success message
print("Test passed: Error message displayed for invalid credentials.")

# Close the browser
driver.quit()
```

Explanation:

1. **WebDriver Setup**: The script uses Chrome WebDriver (webdriver.Chrome()), but you can change it to Firefox or another browser as needed.
2. **Login Form Interaction**:
    - It locates the username and password fields, then inputs invalid values into these fields.
    - The script submits the form by clicking the login button.
3. **Error Message Verification**:
    - After the form submission, the script waits for 2 seconds to allow time for the error message to appear (adjust time based on actual form response time).
    - It checks whether an error message (identified by its class name error) is displayed.
4. **Assertion**: The assert statement checks if the error message is visible. If it's not, the test will fail, and an assertion error will be raised.
5. **Cleanup**: After the test completes, driver.quit() ensures the browser is closed.

SET-2

1.Explain the purpose of an inventory file in Ansible and describe the types of inventories it supports

Purpose of an Inventory File in Ansible

In Ansible, an **inventory file** defines the list of hosts (machines or devices) that Ansible will manage and automate tasks on. It specifies which servers to target for executing playbooks, commands, or other tasks. The inventory file can also define host groups and associated variables, making it easier to organize hosts into logical units (e.g., web servers, database servers). This file allows Ansible to know where and how to run tasks across a network of machines.

Types of Inventories Supported in Ansible

1. **Static Inventory**:
   - A **static inventory** is a simple, manually maintained text file (usually in INI or YAML format) that lists hosts and groups of hosts. It is easy to set up for small infrastructures where the list of managed hosts doesn't change frequently.
   - **Example**: A typical static inventory could look like this:

     ```
     [web_servers]
     web1.example.com
     web2.example.com

     [db_servers]
     db1.example.com
     ```

2. **Dynamic Inventory**:
   - A **dynamic inventory** is generated by a script or plugin that queries an external system (like AWS, GCP, or Azure) to fetch the list of hosts in real-time. This is ideal for environments where hosts are frequently added or removed, like cloud-based or containerized systems.
   - **Example**: Using a plugin like aws_ec2 to dynamically fetch EC2 instances:

     ```
     ansible-inventory -i aws_ec2.yml --list
     ```

3. **Hybrid Inventory**:
   - A **hybrid inventory** combines both static and dynamic inventory sources. This is useful in scenarios where part of the infrastructure is static (e.g., on-premises servers) and part is dynamic (e.g., cloud instances).
   - **Example**: A hybrid setup could include both a static file and a dynamic script or plugin to manage different types of hosts.

2. Write and define the command for creating a container, start a container and stop a Container?

Here are the commands for creating, starting, and stopping a Docker container:

1. Creating a Container

To create a container from an image, use the docker run command. This command not only creates the container but also starts it. The general syntax is:

docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

- IMAGE: The Docker image you want to use to create the container.
- [OPTIONS]: Various options you can provide (e.g., port mapping, volume mounting, etc.).

Example:

To create a container from the nginx image:

docker run --name my_nginx -d -p 8080:80 nginx

- --name my_nginx: Assigns the container the name my_nginx.
- -d: Runs the container in detached mode (in the background).
- -p 8080:80: Maps port 8080 on the host to port 80 inside the container (useful for accessing the container's web service).
- nginx: The image name (Docker will pull this from Docker Hub if not present locally).

2. Starting a Container

To start a container that has already been created (but is stopped), use the docker start command.

bash
CopyEdit
docker start [OPTIONS] CONTAINER

- CONTAINER: The name or ID of the container you want to start.

Example:

To start the container named my_nginx:

docker start my_nginx

This will start the container that was previously created with the docker run command.

3. Stopping a Container

To stop a running container, use the docker stop command.

bash
CopyEdit

docker stop [OPTIONS] CONTAINER

- CONTAINER: The name or ID of the container you want to stop.

Example:

To stop the container named my_nginx:

docker stop my_nginx

This command will stop the my_nginx container if it's running.

Note:-you can also write docker container creating command in lab manual line.
3.Write a basic YAML manifest to deploy an Nginx pod in a Kubernetes cluster and expose it using a ClusterIP service.

3.Basic YAML Manifest for Nginx Pod Deployment and Service (5 Marks)

Below is a YAML manifest to deploy an **Nginx Pod** in a Kubernetes cluster and expose it using a **ClusterIP service**.

Yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80

---
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
```

```
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: ClusterIP
```

Explanation :

1. **Deployment Resource**:
    - **apiVersion: apps/v1**: Specifies the API version for the Deployment.
    - **kind: Deployment**: Declares the resource type as a Deployment.
    - **metadata**: Provides a name for the deployment (nginx-deployment).
    - **spec**:
        - **replicas: 1**: Ensures that only one Nginx Pod is running.
        - **selector**: Matches the labels for selecting the Nginx Pods.
        - **template**: Defines the Pod template used to create Nginx Pods.
            - **containers**: Specifies the container details.
                - **image: nginx:latest**: Uses the latest Nginx image from Docker Hub.
                - **ports**: Exposes port 80 on the container.
2. **Service Resource**:
    - **apiVersion: v1**: Specifies the API version for the Service.
    - **kind: Service**: Declares the resource type as a Service.
    - **metadata**: Names the service nginx-service.
    - **spec**:
        - **selector**: Selects Pods with the label app: nginx to associate with the service.
        - **ports**:
            - **port: 80**: Exposes the service on port 80.
            - **targetPort: 80**: Forwards the traffic to port 80 inside the Nginx container.
        - **type: ClusterIP**: Exposes the service within the cluster, not externally.

4. What is a Data Source in Grafana? Which types of data sources can Grafana connect to?

What is a Data Source in Grafana?

In **Grafana**, a **data source** refers to the connection configuration that allows Grafana to fetch and visualize data from external systems or services. It acts as a bridge between Grafana and the underlying data storage (such as databases, monitoring systems, or cloud platforms), enabling users to query, process, and display the data on Grafana dashboards. Each data source has its own configuration and query language, allowing Grafana to interact with different types of data.

Types of Data Sources Grafana Can Connect To

Grafana supports a wide variety of data sources to accommodate different use cases. Below are the main types of data sources Grafana can connect to:

1. **Time-Series Databases**:
    - **Prometheus**: Used for storing time-series metrics, widely used in monitoring and alerting.

    - **InfluxDB**: A time-series database for high-volume data, commonly used in IoT and monitoring.
    - **Graphite**: A tool for storing and visualizing time-series data.
    - **OpenTSDB**: A distributed time-series database for monitoring large-scale systems.
2. **SQL Databases**:
    - **MySQL**: A popular relational database management system (RDBMS).
    - **PostgreSQL**: A powerful open-source RDBMS.
    - **SQL Server**: Microsoft's RDBMS used for enterprise applications.
3. **Cloud Services and APIs**:
    - **AWS CloudWatch**: Monitors AWS resources and applications.
    - **Google Cloud Monitoring (Stackdriver)**: For monitoring applications in Google Cloud.
    - **Azure Monitor**: Microsoft Azure's cloud monitoring service.
4. **NoSQL Databases**:
    - **Elasticsearch**: A search and analytics engine, often used for log and event data.
    - **MongoDB**: A NoSQL database designed for scalability and flexibility.
5. **Other Data Sources**:
    - **Loki**: A log aggregation system often used with Prometheus.
    - **Zabbix**: A network monitoring tool.
    - **CSV Files**: Grafana can read and visualize data from CSV files.

5.What is Test-Driven Development (TDD), What are the main steps involved in the TDD process?

What is Test-Driven Development (TDD)?

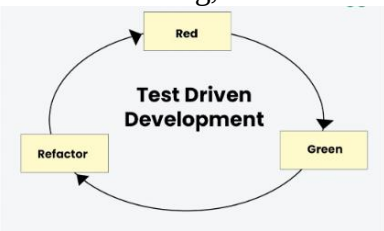
Test-Driven Development (TDD) is a software development approach where tests are written before writing the actual code. The primary goal of TDD is to ensure that code meets the specified requirements by continuously writing tests, implementing functionality, and refactoring code in small iterations. TDD encourages clean, maintainable, and error-free code by focusing on testing throughout the development process.

Main Steps Involved in the TDD Process

TDD follows a simple, structured cycle often referred to as the **Red-Green-Refactor** cycle, which includes the following main steps:

1. **Write a Test (Red)**:
    - First, you write a **test** for the functionality you plan to implement. This test typically focuses on a small, specific unit of the application.
    - Initially, the test will fail because the functionality has not been implemented yet.
    - The goal at this stage is to define the expected behavior of the code.
2. **Write the Code (Green)**:
    - Next, you write the **minimal code** necessary to make the test pass. This code should be as simple as possible to achieve the test's goal.
    - After writing the code, run the tests to ensure that the written code passes the test.
    - This step ensures that your implementation is focused on fulfilling the test's requirements.
3. **Refactor (Refactor)**:
    - Once the test passes, **refactor** the code to improve its structure and readability without changing its behavior.
    - Refactoring makes the code cleaner, more efficient, and easier to maintain.

- After refactoring, the test should still pass to ensure that no functionality was broken.



Set-3

1.Describe the difference between ad-hoc commands and playbooks in Ansible.

1.Definition and Purpose:

- **Ad-Hoc Commands**:
  - **Ad-Hoc commands** are used for quick, one-time tasks without needing to write a playbook. They are useful for executing simple tasks directly from the command line.
  - **Example (Installing Tomcat using Ad-Hoc Command)**: You can run an ad-hoc command to install Tomcat on a remote server:

    ansible all -m yum -a "name=tomcat state=present"

    This command installs Tomcat on all the servers defined in your Ansible inventory.

- **Playbooks**:
  - **Playbooks** are more structured and allow you to define a series of tasks to be executed on remote hosts. Playbooks are written in **YAML format** and are used for more complex tasks or workflows.
  - **Example (Installing and Configuring Tomcat using a Playbook)**:

```
---
- name: Install and configure Tomcat
  hosts: all
  become: yes
  tasks:
    - name: Install Tomcat package
      yum:
        name: tomcat
        state: present
    - name: Start Tomcat service
      service:
        name: tomcat
        state: started
    - name: Ensure Tomcat is enabled on boot
      service:
        name: tomcat
        enabled: yes
```

2. Execution and Flexibility:

- **Ad-Hoc Commands**:
  - Ad-Hoc commands are executed **immediately** from the command line, without the need for a playbook. They are suitable for simple, single-step tasks.
  - They are **not flexible** for more complex tasks or repetitive actions.
- **Playbooks**:
  - Playbooks allow for **multiple tasks** to be defined and executed in sequence. They support **loops**, **conditionals**, and **variables** for more complex and reusable workflows.
  - **Example**: You can write a playbook that installs Tomcat, configures it, and ensures it starts on boot — all in one place.

3. Use Cases:

- **Ad-Hoc Commands**:
  - Ideal for **quick, one-off tasks** such as installing a package, restarting a service, or checking the status of a service.
  - Example use case: Quickly install **Tomcat** on all remote hosts:

    ansible all -m yum -a "name=tomcat state=present"

- **Playbooks**:
  - Ideal for **automating complex, multi-step tasks** that need to be executed across multiple machines or environments. Playbooks are **reusable** and can be used for tasks like **Tomcat installation, configuration, and service management**.
  - Example use case: Install Tomcat, configure the system settings, and ensure that the Tomcat service is started and enabled on boot.

4. Reusability and Maintainability:

- **Ad-Hoc Commands**:
  - Ad-Hoc commands are **not reusable** in the traditional sense. Each time you need to perform a task, you must re-enter the command.
  - Example: If you want to install Tomcat on a new server, you would have to manually type out the ad-hoc command again.
- **Playbooks**:
  - Playbooks are **highly reusable** and can be saved for future use. You can use version control (e.g., Git) to manage and update your playbooks.
  - Example: The Tomcat installation and configuration playbook can be reused for multiple environments or servers by simply modifying the host inventory.

5. Complexity:

- **Ad-Hoc Commands**:

    - Simple and best for **single, straightforward tasks**. They don’t support complex logic such as loops, conditionals, or handling multiple steps in one execution.
- **Playbooks**:
    - Playbooks can handle **complex tasks** and sequences of steps. They are written in YAML, which makes them easy to read and manage. They can include logic, variables, and loops for more sophisticated workflows.
    - Example: A Tomcat playbook could not only install Tomcat but also configure Java, download application files, and start the Tomcat service with the required configuration.

2.Write a simple docker-compose.yml file to set up a web application with an Nginx container and a backend container.

version: '3.8'

services:

backend:

image: node:14

container_name: backend

working_dir: /app

volumes:

- ./backend:/app

command: npm start

ports:

- "5000:5000"

networks:

- webapp_network

nginx:

image: nginx:latest

container_name: nginx

volumes:

- ./nginx.conf:/etc/nginx/nginx.conf

- ./frontend:/usr/share/nginx/html

ports:

- "80:80"

depends_on:

- backend

networks:

- webapp_network

networks:

webapp_network:

(Below Explanation understanding purpose only no need to write)

Explanation

1. **Version**:
   - The version: '3.8' specifies the version of Docker Compose syntax to use.
2. **Backend Service**:
   - image: node:14: Uses the official Node.js image to run the backend service.
   - container_name: backend: Names the backend container backend.
   - working_dir: /app: Sets the working directory inside the container.
   - volumes: ./backend:/app: Mounts the local ./backend directory to /app in the container.
   - command: npm start: Runs npm start to start the Node.js application.
   - ports: "5000:5000": Maps port 5000 of the container to port 5000 on the host machine.
   - networks: webapp_network: Connects the backend container to a custom network.
3. **Nginx Service**:
   - image: nginx:latest: Uses the official Nginx image for the web server.
   - container_name: nginx: Names the Nginx container nginx.
   - volumes: ./nginx.conf:/etc/nginx/nginx.conf: Mounts a custom nginx.conf configuration file for Nginx.
   - volumes: ./frontend:/usr/share/nginx/html: Mounts the local ./frontend directory containing static files to the container.
   - ports: "80:80": Maps port 80 of the container to port 80 on the host machine.
   - depends_on: backend: Ensures Nginx starts after the backend service is up.
   - networks: webapp_network: Connects Nginx to the same network as the backend container.
4. **Networks**:
   - Defines a custom bridge network (webapp_network) to allow communication between the containers.

3.Compare the orchestration approaches of Docker Swarm and Kubernetes, and determine in which scenarios each would be more appropriate.

Comparison of Docker Swarm and Kubernetes (Table Format)

Feature	Docker Swarm	Kubernetes
Ease of Use	Simple to set up and use, integrated with Docker	Steeper learning curve, requires understanding of complex concepts
Scalability	Limited scalability, suitable for small-medium projects	Highly scalable, can manage large clusters and applications
High Availability	Basic fault tolerance, manual recovery	Advanced self-healing, auto-scaling, high availability
Networking	Simple overlay network, basic load balancing	Advanced networking, customizable, supports complex network policies and external load balancers
Ecosystem	Limited integrations and third-party tools	Rich ecosystem with integrations for monitoring, logging, CI/CD, etc.
Best Use Cases	Small to medium deployments, simpler applications	Large-scale applications, complex microservices, multi-cloud/hybrid-cloud environments

4.Explain how Prometheus collects metrics from infrastructure components.

1. Prometheus
Prometheus is an open-source monitoring and alerting toolkit designed for reliability and scalability. It is widely used for collecting, storing, and querying metrics from various infrastructure components like servers, databases, applications, and networking devices.

2. Data Collection Method - Pull Model
Prometheus primarily uses a **pull-based model** to collect metrics. In this model:

- **Prometheus server** periodically scrapes metrics data from targets (infrastructure components) via HTTP endpoints.
- The components (or exporters) expose their metrics on specific **HTTP endpoints**, usually in **Prometheus exposition format**.
- Prometheus scrapes these endpoints at regular intervals to collect time-series data.

Example: Prometheus might scrape metrics from a web server by making HTTP requests to http://<target>/metrics, where the target is exposed by the infrastructure component.

3. Exporters
Infrastructure components generally do not expose metrics in a format Prometheus understands directly. To bridge this gap, **exporters** are used:

- **Exporters** are small programs or services that convert metrics from various systems (like operating systems, databases, or web servers) into a format Prometheus can scrape.
- For example:
    - **Node Exporter**: Exposes hardware and OS metrics (CPU, memory, disk, etc.) for Linux/Unix systems.
    - **MySQL Exporter**: Collects and exposes metrics from MySQL databases.

These exporters expose metrics on predefined HTTP endpoints (e.g., http://localhost:9100/metrics for Node Exporter).

4. Metric Scraping
Prometheus uses the **scraping mechanism** to collect data:

- **Scrape Interval**: Prometheus is configured to scrape metrics at specific intervals, for example, every 15 seconds.
- It pulls data from a set of **targets**, which can be:
  - Individual infrastructure components (e.g., servers, databases).
  - **Service discovery** mechanisms (to automatically find and track services in dynamic environments like Kubernetes).

Prometheus stores this data as **time-series metrics**, which are timestamped and organized by **labels** (such as instance, job, and region) to track different dimensions of the data.

5. Service Discovery
Prometheus can dynamically discover services in an environment, particularly in dynamic environments like cloud infrastructure or containerized environments. **Service discovery** allows Prometheus to find new targets automatically:

- **Kubernetes**: Prometheus can use Kubernetes API to discover containerized applications.
- **Consul/Etcd**: Service discovery for infrastructure components.
- This dynamic discovery ensures that Prometheus always scrapes relevant components, even in environments where services frequently change.

5.What is Selenium and what are its main components? Explainthe features of Selenium.

1. Introduction to Selenium
Selenium is a popular **open-source** automation tool for **web application testing**. It allows testers and developers to automate the process of interacting with web browsers, mimicking real user actions to validate the functionality of a website or web application. Selenium supports multiple programming languages like **Java**, **Python**, **C#**, **Ruby**, and more, allowing users to write scripts in their preferred language.

2. Main Components of Selenium
Selenium consists of four main components, each serving different roles in the automation process:

- **Selenium WebDriver**:
  WebDriver is the core component of Selenium that allows for browser automation. It interacts directly with the web browser, sends commands to control browser actions like clicking buttons, filling forms, and verifying page content. WebDriver supports multiple browsers such as Chrome, Firefox, Safari, and Edge.
- **Selenium IDE (Integrated Development Environment)**:
  Selenium IDE is a **record and playback** tool, primarily used for beginners. It is a browser extension (available for Firefox and Chrome) that allows users to record their interactions with a web application and then generate corresponding test scripts. While it's easier to use, it is less flexible compared to WebDriver.
- **Selenium Grid**:
  Selenium Grid enables parallel test execution on multiple machines and browsers simultaneously. It helps in **distributing tests** across various environments, reducing the overall execution time for large test suites. Grid allows running tests on remote machines, making it ideal for cross-browser testing.
- **Selenium Remote Control (RC)** (Deprecated):
  Selenium RC was one of the earlier components used for browser automation but has been deprecated in favor of WebDriver due to its limitations. RC required a special server to be running, which created extra complexity and performance issues.

3. Features of Selenium
Selenium provides several powerful features for web automation:

- **Cross-Browser Compatibility**:
  Selenium supports all major browsers, including Google Chrome, Mozilla Firefox, Safari, and Microsoft Edge. It allows users to write tests once and run them across different browsers to ensure cross-browser compatibility.
- **Language Support**:
  Selenium supports multiple programming languages like **Java**, **Python**, **C#**, **Ruby**, **JavaScript**, and **Kotlin**, making it flexible for developers and testers with varying technical backgrounds.
- **Platform Independence**:
  Selenium is platform-independent, meaning it can run on multiple operating systems, including Windows, macOS, and Linux. It integrates with various tools and frameworks, such as **TestNG**, **JUnit**, and **Maven**, to facilitate a wide range of testing needs.
- **Automated Interaction with Web Elements**:
  Selenium allows users to interact with web elements such as buttons, links, text boxes, and dropdowns, just like a human user would. This includes actions like clicking, typing, selecting, and verifying elements.
- **Support for Dynamic Web Applications**:
  Selenium is capable of handling dynamic web pages that rely on JavaScript, such as Ajax-based applications. It can wait for elements to load asynchronously, ensuring that tests are executed correctly even in complex web applications.
- **Integration with Other Tools**:
  Selenium can be easily integrated with various testing frameworks like **JUnit**, **TestNG**, and **Cucumber**. It also works with continuous integration tools like **Jenkins** and can be used in combination with tools like **Appium** for mobile testing and **Allure** for reporting.
- **Parallel Test Execution**:
  Selenium Grid allows parallel test execution on multiple machines, browsers, and platforms, which speeds up the testing process, especially for large test suites.

Set-4

1.Describe the Roles and modules in Ansible? Give some examples.

Introduction to Ansible
Ansible is an **open-source automation tool** used for configuration management, application deployment, task automation, and orchestration. It uses **simple YAML files** called playbooks to define tasks that automate the configuration of systems, networks, and applications.

1. Roles in Ansible
A **role** in Ansible is a way of organizing playbooks and related files in a structured manner. It is a modular and reusable component that allows for the logical grouping of tasks, variables, files, templates, and handlers.

- **Purpose of Roles**: Roles are designed to facilitate code reusability, modularity, and maintainability. By defining roles, you can divide a large playbook into smaller, more manageable sections, each performing a specific task (like setting up a web server or a database).
- **Structure of a Role**: Each role has a specific directory structure that includes the following key directories:
  - **tasks/**: Contains the main set of tasks (YAML files).
  - **vars/**: Contains variables used by the role.
  - **files/**: Contains static files that need to be copied to managed nodes.
  - **templates/**: Contains Jinja2 templates.
  - **handlers/**: Contains tasks triggered by events, such as service restarts.
  - **defaults/**: Contains default variables for the role.
- **Example of Using a Role**:
  To use a role in a playbook, you can simply reference it as follows:

  ```
  - hosts: webservers
    roles:
      - apache
  ```

  Here, apache is the role defined to set up an Apache web server.

2. Modules in Ansible
A **module** in Ansible is a discrete unit of work that Ansible executes. Modules perform tasks like installing packages, copying files, managing services, and more. Ansible provides a wide range of built-in modules, and users can also create custom modules if needed.

- **Purpose of Modules**: Modules are the building blocks of Ansible tasks. They allow Ansible to interact with systems and perform actions like modifying files, executing commands, installing software, and managing system services.
- **Types of Modules**:
  - **Core Modules**: These are modules that come with Ansible by default, such as yum, apt, copy, and service.
  - **Extra Modules**: These modules are maintained separately and can be installed from external collections or repositories. Examples include modules for cloud providers like ec2 for AWS or gce for Google Cloud.
- **Examples of Common Modules**:
  - **apt**: Installs, removes, or upgrades packages on **Debian-based systems**.

```
- name: Install nginx
  apt:
    name: nginx
    state: present
```

- **service:** Manages system services (start, stop, restart, etc.).

```
- name: Ensure nginx is running
  service:
    name: nginx
    state: started
```

  - **copy**: Copies files to remote machines.

```
- name: Copy the configuration file
  copy:
    src: /local/path/config.yml
    dest: /remote/path/config.yml
```

  - **user**: Manages user accounts on remote systems.

```
- name: Create a new user
  user:
    name: johndoe
    state: present
```

2.What is Docker, and what are its primary use cases in software development?

Introduction to Docker
Docker is an **open-source platform** that allows developers to automate the deployment, scaling, and management of applications in lightweight, portable containers. It uses containerization technology to package software and its dependencies into a standardized unit, ensuring that the application works seamlessly across different environments, from a developer's machine to production servers.

- **Containers** are isolated environments that run applications and services independently, with all required libraries, configurations, and dependencies bundled together. This solves the common problem of "works on my machine" by ensuring consistency across various systems.

1. Primary Components of Docker

- **Docker Engine**: The core component responsible for creating, running, and managing containers.
- **Docker Images**: Read-only templates that contain the application and its dependencies. Docker containers are created from images.
- **Docker Containers**: Lightweight, standalone, and executable packages created from Docker images. They encapsulate an application and all its dependencies.
- **Docker Hub**: A cloud-based registry service for sharing and storing Docker images.

2. Primary Use Cases of Docker in Software Development

- **Environment Consistency**:
  Docker ensures that applications run consistently across different environments (e.g., development, testing, staging, production). This eliminates issues related to environment discrepancies and dependency mismatches. Developers can package their applications and run them on any system with Docker, ensuring **environment consistency**.

  Example: A developer builds and tests an application on their local machine, and the exact same environment can be used in staging and production without changes.

- **Simplified Dependency Management**:
  Docker containers bundle the application and all of its dependencies (libraries, frameworks, configurations) together, ensuring that developers do not have to worry about installing and managing dependencies manually on each machine. This **dependency isolation** reduces conflicts between different software versions.

  Example: A Python application with specific library versions can be packaged in a Docker container, ensuring it runs with the same libraries across all environments.

- **Microservices Architecture**:
  Docker is widely used in **microservices architecture**, where each service (e.g., database, front-end, back-end) is encapsulated within its own container. This approach allows for scalable, modular development and easier management of independent services.

  Example: A complex application can be broken down into multiple Docker containers: one for the user interface, one for the back-end logic, and one for the database, all running and scaling independently.

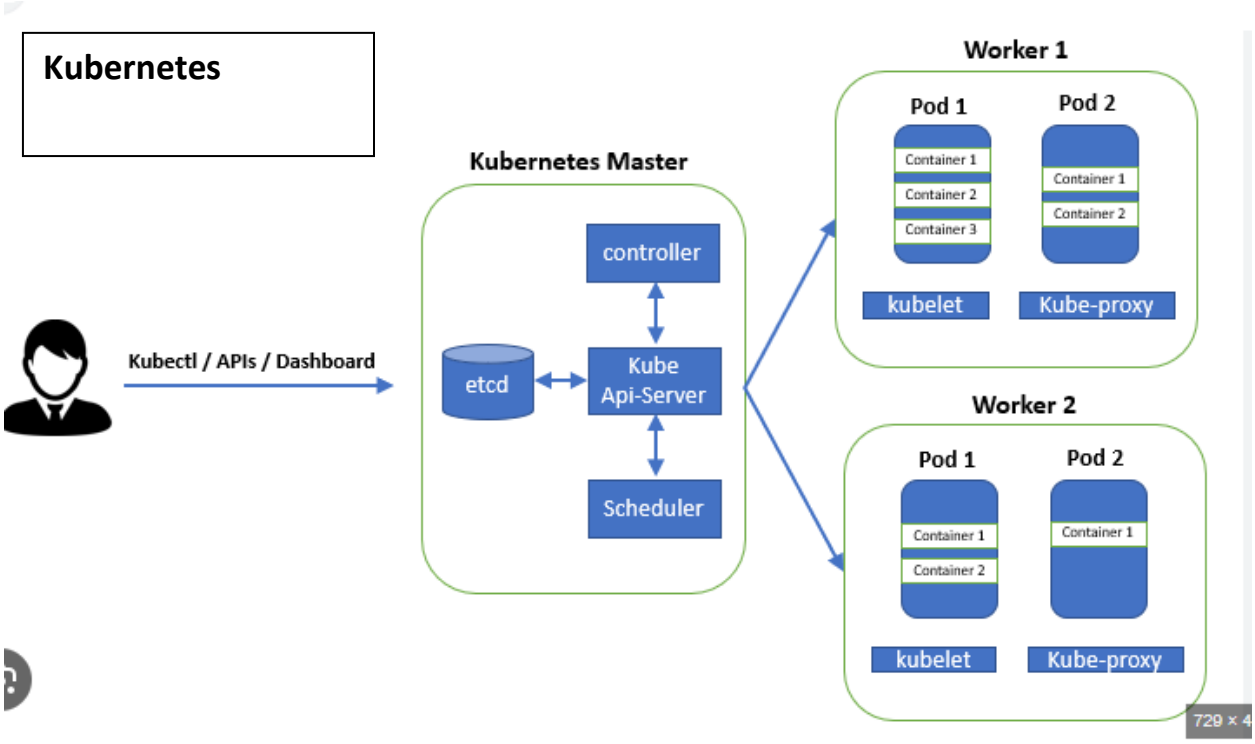
- **Continuous Integration and Continuous Deployment (CI/CD)**:
  Docker enables **automated testing, building, and deployment** in CI/CD pipelines. It helps in **faster delivery** of applications by ensuring consistency in every stage, from development to production. Docker containers can be spun up quickly for testing and deployment, making them ideal for **automation**.

  Example: In a CI/CD pipeline, Docker is used to run tests in isolated containers to ensure the application works correctly before it is deployed to production.

- **Resource Efficiency and Scalability**:
  Docker containers are **lightweight** compared to traditional virtual machines, making them resource-efficient. This allows developers to run more containers on the same hardware. Additionally, containers can be easily scaled up or down depending on demand, making Docker ideal for applications that need to scale efficiently.

  Example: In cloud environments, a Docker containerized application can automatically scale based on traffic, using orchestration tools like Kubernetes to manage the scaling process.

3.Explain the role of the Kubernetes Master Node and Worker Node.



Introduction to Kubernetes Architecture

Kubernetes is an open-source container orchestration platform used to automate the deployment, scaling, and management of containerized applications. The Kubernetes architecture consists of **two primary types of nodes**: the **Master Node** and the **Worker Nodes**. Each node has specific roles and responsibilities in managing the cluster and ensuring that the containerized applications are running effectively.

1. Role of the Kubernetes Master Node

The **Master Node** is the central control plane of a Kubernetes cluster. It is responsible for managing the overall cluster state, making global decisions about scheduling, monitoring, and controlling the system. It ensures the cluster's health and operation.

Key responsibilities of the Master Node include:

- **API Server**: The API Server (kube-apiserver) is the front-end of the Kubernetes control plane and exposes the REST API to interact with the cluster. It acts as the main entry point for users, administrators, and components to communicate with the Kubernetes cluster.
- **Controller Manager**: The Controller Manager (kube-controller-manager) runs controllers that regulate the state of the system. For example, the replication controller ensures that the correct number of pod replicas are running, and the deployment controller manages the deployment lifecycle of applications.
- **Scheduler**: The Scheduler (kube-scheduler) assigns newly created pods to the worker nodes based on resource availability, constraints, and policies. It determines which worker node should run the pod.
- **etcd**: The etcd database is a distributed key-value store that holds the cluster's state and configuration data. It is used for storing all the necessary data to maintain the cluster's configuration and state, such as pod details, deployment information, and more.
- **Cloud Controller Manager**: This component interacts with cloud provider APIs (if using a cloud environment) to manage services like load balancers, persistent volumes, and other cloud-specific resources.

2. Role of the Kubernetes Worker Node
The **Worker Node** is responsible for running the containerized applications (pods) and providing the necessary resources to support them. It communicates with the Master Node to receive instructions on what to do and reports back on the state of the pods running on it.

Key responsibilities of the Worker Node include:

- **Kubelet**: The kubelet is an agent that runs on each worker node. It ensures that the containers (pods) are running in the desired state, as specified by the Master Node. The kubelet communicates with the API server to get the desired state and manages the lifecycle of the pods and containers on its node.
- **Container Runtime**: The container runtime (e.g., Docker, containerd) is responsible for running and managing containers within the pods. It handles the starting, stopping, and execution of containers inside the pods on the worker node.
- **Kube Proxy**: The kube-proxy is a network proxy that maintains network rules on the worker node. It enables communication between services within the cluster by load balancing network traffic and managing routing to the correct pod.

4.Explain about PromQL with some example, and define time-series database.

What is PromQL?

PromQL (Prometheus Query Language) is a powerful query language used in **Prometheus** to retrieve and analyze time-series data stored in its database. It allows users to filter, aggregate, and calculate metrics for monitoring, alerting, and visualization purposes.

PromQL is designed specifically for working with **metrics data** collected over time (e.g., CPU usage, memory usage, HTTP requests, etc.). It supports functions like rate calculation, averages, and comparisons to help in real-time monitoring.

1. Examples of PromQL Queries

- **Basic Metric Query**

  http_requests_total

  This returns the total number of HTTP requests recorded.

- **Filtering by Labels**

  http_requests_total{method="GET", status="200"}

  This returns the number of successful GET requests.

- **Rate of Change Over Time**

  rate(cpu_usage_seconds_total[5m])

  This calculates the **per-second rate** of CPU usage over the last 5 minutes.

- **Average CPU Usage Across All Instances**

  avg(cpu_usage_seconds_total)

  This returns the **average CPU usage** across all instances.

- **Alerting Example**

  node_memory_Active_bytes / node_memory_MemTotal_bytes > 0.9

  This expression can be used to trigger an alert if memory usage goes above **90%**.

3. What is a Time-Series Database?

A **Time-Series Database (TSDB)** is a type of database optimized for storing and querying **time-stamped data**. Each data point in a time-series has:

- A **timestamp** (when it was recorded),
- A **metric name** (what is being measured),
- A **value** (the measurement),
- And optional **labels** (key-value pairs for identifying dimensions like host, region, etc.).

In Prometheus, the built-in time-series database stores all the collected metrics as time-series data, making it easy to query trends over time using PromQL.

5.What is REPL-driven development, How can REPL-driven development help in writing Selenium tests?

1. What is REPL-Driven Development?

REPL stands for **Read-Eval-Print Loop**. It is an interactive programming environment that:

- **Reads** the user's input (a line of code),
- **Evaluates** the input (executes the code),
- **Prints** the result,
- And **Loops** back for the next input.

REPL-driven development is a development approach where programmers use a REPL to write and test small chunks of code **interactively** and **iteratively**, rather than writing large blocks of code at once. This allows for **real-time feedback**, **faster debugging**, and **immediate testing of code logic**.

Languages like **Python**, **JavaScript**, and **Clojure** support REPL environments.

2. How REPL-Driven Development Helps in Writing Selenium Tests

Selenium is used for automating web browser interactions. REPL-driven development can make writing Selenium tests **easier**, **faster**, and **more reliable** in the following ways:

a) Instant Feedback for Browser Commands

Using REPL, testers can execute Selenium commands (like click(), find_element(), get()) one at a time and immediately see how the browser responds. This helps in:

- Verifying that elements are correctly located,
- Checking if interactions like clicking or typing work,
- Quickly experimenting with test steps before adding them to a full script.

Example:

```
>>> from selenium import webdriver
>>> driver = webdriver.Chrome()
>>> driver.get("https://example.com")
>>> driver.find_element("id", "login").click()
```

Each line is executed and tested immediately, making it easy to debug and refine.

b) Faster Test Development

REPL-driven testing avoids the need to run full test scripts repeatedly. Developers and testers can build tests step-by-step and only include steps that are confirmed to work, resulting in faster test creation.

c) Better Debugging

When a test step fails, you can try alternative commands right away in the REPL instead of modifying and re-running the whole script. This reduces the time spent debugging test scripts.

d) Learning and Exploration

New users of Selenium can use REPL to learn how Selenium works by trying commands interactively. It helps them understand how to locate elements and perform actions without writing full test cases.