Devops

1. Introduction (10–15 minutes)

- **Introduce Yourself:** Share your background, experience, and journey in DevOps.

Introduction:
Hello everyone, my name is Rajat zade, and I am thrilled to be your DevOps trainer for this course. I have five years of hands-on experience as a DevOps Engineer at confidential organisation ., Additionally, I have over seven years of experience as a trainer,
Throughout this journey, I've gained in-depth expertise in various DevOps technologies, including CI/CD pipelines, containerization, cloud platforms, and infrastructure automation. I am passionate about sharing knowledge and helping others excel in their DevOps careers.
Let's work together to make this training a valuable and enriching experience for all of us!

- **Explain the Course Outline:** Provide a high-level overview of the topics to be covered, like CI/CD, containerization, configuration management, Kubernetes, etc.

Introduction to DevOps
Source Code Management Tool (GIT)
Containerisation using Docker
Orchestration tool (Kubernetes)
Code Infrastructure with Terraform
Jenkins Essentials
Build Tool Maven
Check Quality of Code (Sonarqube)
Monitoring Tool (Datadog)

2. **Understand the Students (10–15 minutes)**

Student Introductions: Let each student briefly introduce yourselves, their background, and their goals.

DevOps Overview Explanation

1. What is DevOps: Definition, Principles, and Culture

- **Definition**:
  DevOps is a set of practices that combines software development (Dev) and IT operations (Ops) to shorten the system development life cycle and deliver high-quality software continuously.

- **Principles**:
  - **Collaboration**: Teams work together across development and operations.
  - **Automation**: Automating repetitive tasks to save time and reduce errors.
  - **Continuous Improvement**: Regularly refining processes to improve speed and quality.
- **Culture**:
  DevOps fosters a culture of ownership, accountability, and shared goals. Teams are empowered to work together rather than in silos, ensuring faster and more reliable delivery.

2. Benefits of DevOps

- **Faster Delivery**:
  DevOps automates and streamlines the development and deployment process, reducing time-to-market.
- **Collaboration**:
  Encourages cross-functional teams to work together, breaking down silos.
- **Improved Scalability**:
  Enables teams to build infrastructure and applications that can scale easily to meet demand.

3. DevOps Lifecycle

- **Plan**:
  Teams use tools like Jira or Trello to define requirements and plan tasks.
- **Develop**:
  Developers write and test code using IDEs (Integrated Development Environment) and version control systems like Git.
- **Build**:
  Tools like Jenkins compile the code and create executable artifacts. Using maven ,Gradle,TravisCI,CircleCI,Jenkins
- **Test**:
  Automated testing using tools like sonarqube,Selenium or JUnit ensures code quality.
- **Release**:
  Release management tools ensure code is ready for deployment.
- **Deploy**:
  Deployment is automated using tools like Kubernetes or AWS CodeDeploy.
- **Operate**:
  Tools like Ansible and Chef help manage the infrastructure in production.
- **Monitor**:

Monitoring tools like Prometheus and Grafana ensure the health and performance of systems.

What is the Software Industry?
The software industry encompasses businesses involved in the development, maintenance, and publication of software products. It includes companies offering services like custom software development, software as a service (SaaS), cloud computing, and technology consulting.

Types of Software

1. **System Software**:
   Includes operating systems (e.g., Windows, Linux), database management systems, and utility software that manage hardware and application software.
2. **Application Software**:
   Tools and programs used for end-user tasks like word processing, web browsing, and gaming.
3. **Middleware**:
   Connects different software applications, enabling communication and data management.

Key Roles in the Software Industry

1. **Software Developers**: Design, code, and build applications and systems.
2. **Quality Assurance Engineers**: Ensure software quality through testing.
3. **DevOps Engineers**: Streamline the development, deployment, and operations processes.
4. **Product Managers**: Define product requirements and manage delivery.
5. **UI/UX Designers**: Focus on user experience and interface design.

Current Trends in the Software Industry

1. **Cloud Computing**: Organizations are migrating to cloud platforms like AWS, Azure, and Google Cloud.
2. **Artificial Intelligence (AI)**: AI and machine learning are transforming industries, enhancing automation and decision-making.
3. **DevOps and CI/CD**: Rapid software delivery through continuous integration and deployment is now a norm.
4. **Cybersecurity**: With the rise in cyber threats, software security is more critical than ever.
5. **Remote Work Tools**: Demand for collaboration tools (e.g., Slack, Zoom) has surged post-pandemic.

Challenges in the Software Industry

1. **Talent Shortages**: High demand for skilled software professionals.
2. **Rapidly Evolving Technology**: Keeping up with trends requires continuous learning.
3. **Cybersecurity Risks**: Threats to software security are increasing.
4. **Global Competition**: Companies face intense competition from across the globe.

Types of IT Companies

1. Product-Based Companies

- **Definition**: These companies develop and sell their own software products or platforms.
- **Examples**: Microsoft, Google, Adobe, Oracle.
- **Characteristics**:
  - Focus on innovation and scalability.
  - Revenue primarily through licensing, subscriptions, or sales.
  - Products are used by multiple customers across industries.

2. Service-Based Companies

- **Definition**: These companies provide IT services such as consulting, software development, and system integration to other businesses.
- **Examples**: TCS, Infosys, Wipro, Accenture.
- **Characteristics**:
  - Offer custom solutions tailored to clients' needs.
  - Revenue through projects and contracts.
  - Operate across multiple industries.

3. Startup Companies

- **Definition**: Emerging businesses that focus on innovative solutions, often in niche markets or emerging technologies.
- **Examples**: Airbnb (early years), Zoom, and many local SaaS or AI startups.
- **Characteristics**:
  - Agile and fast-moving.
  - Aim for rapid growth and disruption in their domain.
  - Often rely on venture capital funding.

4. Cloud Service Providers

- **Definition**: Companies specializing in cloud-based platforms, storage, and computing services.
- **Examples**: Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP).
- **Characteristics**:
  - Offer scalable and flexible solutions.
  - Charge on a pay-as-you-go basis.
  - Focus on reliability, security, and global reach.

5. IT Consulting Firms

- **Definition**: Provide strategic guidance to organizations on how to use IT effectively.
- **Examples**: Deloitte, Capgemini, McKinsey (IT Division).
- **Characteristics**:
  - Focus on optimizing business processes.
  - Revenue through consultancy fees.
  - Often overlap with service-based companies.

6. Software Development Companies

- **Definition**: Specialize in developing custom software applications for clients.
- **Examples**: Atlassian, ThoughtWorks, EPAM.
- **Characteristics**:
  - Customizable software solutions for specific needs.
  - Cover web, mobile, and desktop platforms.
  - Often work with niche industries or technologies.

7. Hardware and Networking Companies

- **Definition**: Develop and maintain physical IT infrastructure and networking solutions.
- **Examples**: Cisco, HP, Dell, Intel.
- **Characteristics**:
  - Provide servers, networking devices, and hardware solutions.
  - Often paired with software or cloud services.
  - Focus on enterprise-grade reliability.

8. Cybersecurity Companies

- **Definition**: Focus on providing solutions to protect digital assets, networks, and data from cyber threats.
- **Examples**: Palo Alto Networks, McAfee, Symantec.
- **Characteristics**:
  - Offer tools like firewalls, antivirus, and encryption.
  - Provide threat detection and incident response services.
  - Focus heavily on compliance and regulatory standards.

9. E-commerce and Internet Companies

- **Definition**: Develop and maintain platforms for online shopping, marketplaces, and web services.
- **Examples**: Amazon, Flipkart, Alibaba, eBay.
- **Characteristics**:
  - Rely on large-scale IT infrastructure.
  - Integrate cloud computing, AI, and big data.
  - Revenue through product sales, subscriptions, and advertisements.

10. Gaming and Multimedia Companies

- **Definition**: Create games, entertainment software, and multimedia applications.
- **Examples**: EA Games, Unity, Netflix (IT division).

- **Characteristics**:
  - Blend creativity with technology.
  - Often focus on user engagement and experience.
  - Use cutting-edge graphics and real-time data processing.

11. Research and Development (R&D) Companies

- **Definition**: Focus on innovating new technologies, algorithms, and frameworks.
- **Examples**: IBM Research, Bell Labs, DeepMind.
- **Characteristics**:
  - Aim for breakthroughs in technology.
  - Heavily invested in AI, machine learning, and quantum computing.
  - Often partner with universities and academic institutions.

Software Development Life Cycle (SDLC): Waterfall vs Agile Model

1. What is SDLC?

SDLC is a structured framework used to design, develop, and test high-quality software. It consists of several phases that guide software teams in managing projects efficiently.

Waterfall Model

The Waterfall Model is a linear and sequential approach to software development. Each phase must be completed before moving to the next.

Key Phases of the Waterfall Model:

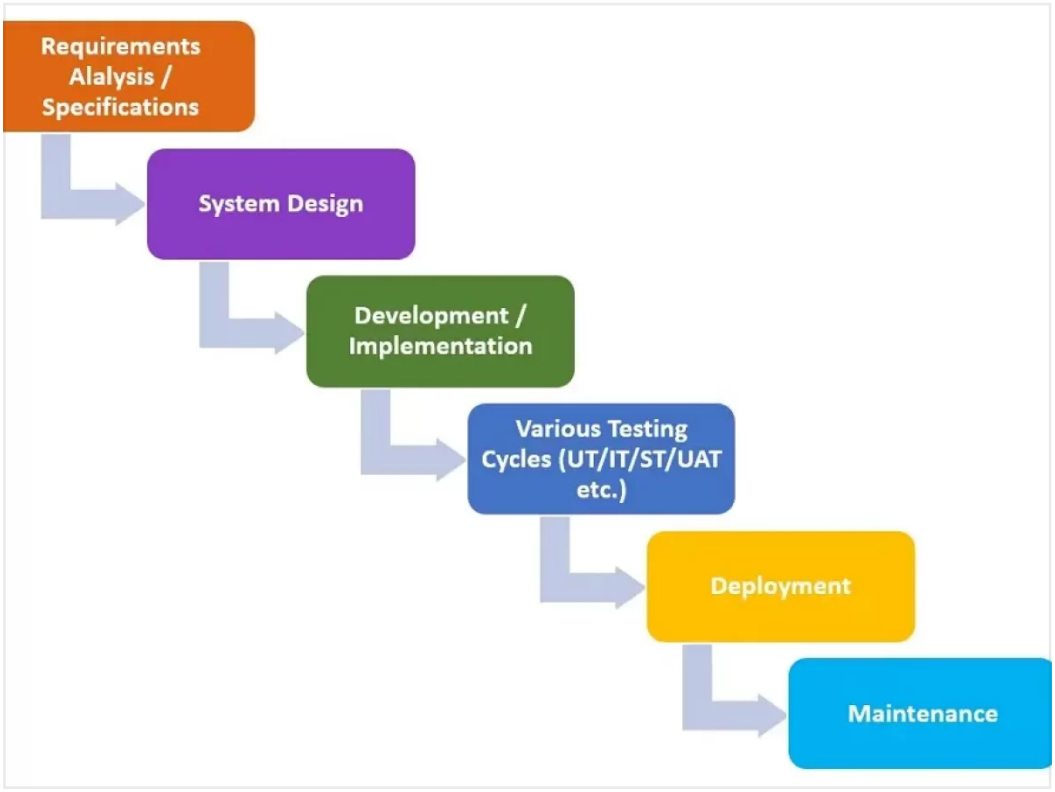
1. **Requirement Analysis**: Detailed documentation of customer needs.
2. **System Design**: Designing architecture and components.
3. **Implementation**: Coding based on design specifications.
4. **Testing**: Verifying that the system works as expected.
5. **Deployment**: Delivering the final product to the customer.
6. **Maintenance**: Providing updates and fixes.

Advantages of the Waterfall Model:

- Clear structure and easy to understand.
- Works well for projects with fixed requirements.
- Easy to manage due to defined stages.

Disadvantages of the Waterfall Model:

- Inflexible to changes once the process begins.
- Testing happens late in the cycle, leading to delayed bug detection.
- Not suitable for complex or long-term projects.



Agile Model

The Agile Model is an iterative and flexible approach focusing on collaboration and customer feedback. It delivers software in small, functional increments.

Key Principles of Agile:

1. **Iterative Development**: Work divided into short iterations or sprints.
2. **Customer Collaboration**: Continuous interaction with the customer.
3. **Flexibility**: Adapts to changes even late in the development cycle.
4. **Team Collaboration**: Cross-functional teams work closely together.

Key Practices in Agile:

- **Scrum**: Uses sprints and daily stand-ups.
- **Kanban**: Visualizes workflows using boards.
- **Extreme Programming (XP)**: Emphasizes frequent releases and feedback.

Advantages of the Agile Model:

- Highly flexible to changing requirements.
- Continuous delivery ensures early feedback.
- Promotes collaboration between developers and stakeholders.

Disadvantages of the Agile Model:

- Requires skilled and collaborative teams.
- Less predictability in terms of cost and time for large projects.
- Difficult to measure progress due to its dynamic nature.

———

Introduction to APIs (Application Programming Interfaces)

An **API (Application Programming Interface)** is a set of rules and protocols that allows one software application to interact with another. It defines the methods and data formats that applications use to communicate with each other. APIs are used to enable the functionality of different systems, services, or platforms to work together.

An API (Application Programming Interface) ek niyam aur protokol ka set hai jo ek software application ko doosre software application ke sath interact karne ki anumati deta hai. Yeh un methods aur data formats ko define karta hai jo applications ke beech communication ke liye use hote hain. APIs ka istemal alag-alag systems, services, ya platforms ki functionality ko ek sath kaam karne ke liye kiya jata hai.

Why APIs are important?

- **Communication:** APIs allow different software components to interact and communicate, even if they are built using different technologies or platforms.
- **Efficiency:** APIs help automate processes and provide pre-built functionality, reducing the need for developers to write code from scratch.
- **Integration:** APIs make it easier to integrate with third-party services and external applications.

Types of APIs

1. **Open APIs (Public APIs)**
    - Open to external developers and available for public use.
    - Example: APIs provided by social media platforms like Twitter, Facebook, or Google Maps.
    - **Use case:** A weather application uses an open API to fetch real-time weather data.
2. **Internal APIs (Private APIs)**
    - Used internally within an organization to streamline processes and allow different teams or systems to share data.
    - Example: A company's internal sales system may interact with its inventory management system through an internal API.
    - **Use case:** A company's backend service might use an internal API to access databases or other internal services.
3. **Partner APIs**
    - Shared between a business and its partners. They are not available to the public but can be accessed by specific users or applications.
    - Example: A bank's API to integrate its payment gateway with a merchant’s website.
    - **Use case:** A retailer uses a partner API to accept payments through a third-party payment gateway.

4. **Composite APIs**
    - Allow multiple endpoints to be accessed in a single call. This type is especially useful in microservices architecture, where multiple services are involved in a single task.
    - Example: A request to an API that retrieves data from both an inventory system and a customer relationship management (CRM) system.
    - **Use case:** A healthcare application using a composite API to fetch patient records, lab results, and medical history from multiple microservices.

Common Use Cases of APIs

1. **Web Services and Applications**
   APIs enable web and mobile applications to communicate with servers, databases, or other services. For example, a weather app pulls data from an external weather service via an API.
2. **Third-Party Integrations**
   APIs allow different software applications to integrate with third-party services, such as payment gateways (PayPal, Stripe), mapping services (Google Maps API), and social media (Twitter API).
3. **Microservices Communication**
   In microservices architectures, APIs serve as the communication layer between different services. Each microservice exposes its functionality via APIs to be consumed by other services.
4. **IoT (Internet of Things)**
   APIs enable communication between IoT devices and other systems. For instance, an IoT device may use an API to send data to a central server or application.
5. **Cloud Services**
   Cloud providers like AWS, Google Cloud, and Azure provide APIs for automating tasks like creating virtual machines, provisioning databases, managing storage, etc.
6. **Data Exchange and Automation**
   APIs facilitate data exchange between systems and automate repetitive tasks. For example, an API can be used to update customer data between a CRM and an email marketing platform.

API Methods (HTTP Methods)

APIs often use HTTP methods to perform different operations:

- **GET**: Retrieves data from a server (e.g., fetching user details).
- **POST**: Sends data to the server (e.g., creating a new user).
- **PUT**: Updates existing data on the server (e.g., updating user information).
- **DELETE**: Removes data from the server (e.g., deleting a user).
- **PATCH**: Partially updates data on the server (e.g., updating part of a

user profile).

API Authentication and Security

To ensure that only authorized users can access sensitive data and perform certain operations, APIs often require authentication and security mechanisms:

1. **API Keys**: A unique identifier passed with each API request to authenticate the caller.
2. **OAuth**: A more secure and complex protocol for authorizing applications to access data on behalf of a user.
3. **JWT (JSON Web Tokens)**: Used for securely transmitting information between client and server.

Benefits of Using APIs

- **Automation**: APIs enable automation of repetitive tasks and seamless data exchange.
- **Interoperability**: APIs help different systems or services to work together, even if they are built with different technologies.
- **Flexibility**: APIs provide a flexible way to integrate and extend the capabilities of applications.
- **Scalability**: APIs allow the integration of new services and tools without disturbing the entire system.

Conclusion:
APIs software ecosystem mein ek mahatvapurn bhumika nibhate hain, jo systems ke beech communication ko enable karte hain, custom development ki zarurat ko kam karte hain, aur external services ke sath integration ko asaan banate hain. APIs ke types, methods, aur use cases ko samajhna un developers ke liye atyant zaruri hai jo modern applications par kaam karte hain, khaaskar microservices, cloud environments, aur DevOps practices mein.

Frontend vs Backend: Understanding the Difference

In software development, **frontend** and **backend** refer to different parts of an application or website, each with its own responsibilities and technologies.

Frontend (Client-Side)

The **frontend** is everything the user interacts with on a website or application. It's what you see in your browser or on your mobile app. It’s the **"client-side"** of the application, and its main goal is to provide a seamless and engaging user experience.

Responsibilities of the Frontend:

1. **User Interface (UI)**: Everything the user sees and interacts with — buttons, text, images, menus, forms, and the layout of the page.
2. **User Experience (UX)**: The design, flow, and responsiveness of the application to ensure it's easy to use and navigate.
3. **Interaction with Backend**: The frontend communicates with the backend to retrieve and send data, typically via **APIs** (Application

Programming Interfaces).

Technologies Used in Frontend:

- **HTML** (HyperText Markup Language): Structures the content of the web page.
- **CSS** (Cascading Style Sheets): Styles the page, including colors, fonts, and layout.
- **JavaScript**: Adds interactivity, such as form validation, animations, or dynamic content updates.
- **Frontend Frameworks/Libraries**: Tools like **React**, **Vue.js**, **Angular**, and **Svelte** help build complex user interfaces efficiently.

Example of Frontend in Action:

When you visit a website like **Facebook**, everything you see on the page — your feed, the "Like" buttons, images, and text — is part of the **frontend**.

—

Backend (Server-Side)

The **backend** refers to the part of the application that runs on the **server-side**. It is responsible for managing the data, ensuring the logic of the application works, and handling requests from the frontend.

Responsibilities of the Backend:

1. **Data Management**: Storing, retrieving, and managing the data (like user information, posts, or transactions) in databases.
2. **Business Logic**: Processing the application's functionality, like authentication, calculations, or operations based on requests from the frontend.
3. **API Handling**: Responding to frontend requests via APIs (sending data, updating records, etc.).
4. **Security**: Ensuring that the application is secure by managing authentication (user login), authorization (permissions), and data encryption.

Technologies Used in Backend:

- **Server-Side Languages**: Languages like **Node.js** (JavaScript), **Python**, **Ruby**, **Java**, **PHP**, **C#**, and **Go** are commonly used for backend development.
- **Databases**: Relational databases like **MySQL**, **PostgreSQL**, and **SQL Server**, or NoSQL databases like **MongoDB**, **Cassandra**, and **Redis**.
- **Frameworks**: Tools like **Express.js**, **Django**, **Spring Boot**, **Ruby on Rails**, and **Flask** make it easier to build backend applications.
- **Web Servers**: Software like **Apache**, **Nginx**, or cloud-based services like **AWS** or **Azure** host the backend services.

Example of Backend in Action:

In the **Facebook** example, the backend handles tasks like storing your posts in the database, managing login/authentication, and sending the right content to your feed based on the server-side logic.

—

Stages of Application Development: Roles of Developer, Tester, Database, and DevOps Teams

Application development involves multiple stages where different teams collaborate to ensure the smooth creation, testing, deployment, and maintenance of the software. Here's an overview of the key stages in the development lifecycle and the roles of the **Developer**, **Tester**, **Database**, and **DevOps** teams at each stage.

1. Requirement Gathering and Analysis

- **Developer**:
  - Participate in discussions with business analysts and product managers to understand the application's requirements, features, and functionalities.
  - Provide input on technical feasibility based on the requirements.
- **Tester**:
  - Review requirements to understand the expected functionality.
  - Help in preparing test plans and identifying potential test cases based on requirements.
- **Database**:
  - Analyze the data requirements and suggest suitable database design strategies.
  - Help define data models and schemas for the application.
- **DevOps**:
  - Ensure infrastructure requirements are considered for deployment and scaling.
  - Plan for the environment setup, including servers, CI/CD pipelines, and cloud resources.

2. Design Phase

- **Developer**:
  - Design the overall application architecture, code structure, and user interface (UI).
  - Create design patterns for both frontend and backend systems.
- **Tester**:
  - Design testing strategies and write test cases, both functional and non-functional.
  - Identify potential risks in the application and address them in test planning.
- **Database**:
  - Design the database schema, tables, relationships, and stored procedures.
  - Ensure database normalization and optimization.
- **DevOps**:

    - Design the infrastructure layout and automation workflows.
    - Plan deployment processes for various environments (e.g., dev, test, production).
    - Prepare version control strategies and repository setups.

3. Development Phase

- **Developer**:
    - Write the code for both the frontend and backend of the application based on the design.
    - Implement API endpoints, business logic, and integrate third-party services.
- **Tester**:
    - Begin writing unit tests to validate individual code components.
    - Prepare test environments and tools for integration and system testing.
- **Database**:
    - Implement database schema changes, manage migrations, and populate initial data.
    - Ensure data integrity and write queries for backend integration.
- **DevOps**:
    - Set up continuous integration (CI) and continuous deployment (CD) pipelines for automatic builds and deployments.
    - Ensure the necessary infrastructure is provisioned for development and testing environments.

4. Testing Phase

- **Developer**:
    - Debug and fix issues identified during testing.
    - Collaborate with testers to ensure the application meets functional and non-functional requirements.
- **Tester**:
    - Execute different types of testing, including unit tests, integration tests, system tests, performance tests, and user acceptance tests (UAT).
    - Document test results and report any bugs or issues found.
- **Database**:
    - Validate the correctness of database interactions.
    - Run database tests, check for data consistency, and ensure optimization.
    - Assist in validating transactions, indexes, and stored procedures.
- **DevOps**:
    - Ensure automated tests are running as part of the CI pipeline.
    - Monitor build and test environments for any issues.
    - Assist in troubleshooting deployment issues in the test

environment.

5. Deployment Phase

- **Developer**:
  - Ensure that the code is production-ready by conducting final checks and reviews.
  - Assist in the deployment process, ensuring code is properly integrated and functional.
- **Tester**:
  - Perform sanity and smoke tests in the production environment to ensure critical functionalities are working.
  - Verify that no critical issues from testing remain in production.
- **Database**:
  - Manage database migrations and data updates in the production environment.
  - Ensure database performance and integrity during the deployment.
- **DevOps**:
  - Deploy the application to production using CI/CD pipelines.
  - Automate the deployment process to ensure consistency across environments (staging, production).
  - Monitor the deployment and handle any issues that arise.
  - Ensure scalability and high availability by managing cloud resources, load balancers, and databases.

6. Post-Deployment & Maintenance

- **Developer**:
  - Monitor the application for bugs or new features and make improvements or fixes.
  - Collaborate with testers and DevOps to ensure smooth post-deployment monitoring.
- **Tester**:
  - Conduct regression testing after bug fixes or new features are introduced.
  - Ensure the application continues to meet quality standards.
- **Database**:
  - Monitor database performance, handle queries, optimize data storage, and address any data-related issues post-deployment.
  - Assist in scaling the database as the application grows.
- **DevOps**:
  - Monitor the application and infrastructure to ensure stability and uptime.
  - Scale the infrastructure based on usage demands (auto-scaling).
  - Implement logging, monitoring, and alerting systems to track performance, uptime, and issues.

- Ensure the smooth operation of the CI/CD pipeline for ongoing deployments and updates.

7. Continuous Improvement and Feedback

- **Developer**:
  - Continually improve the codebase by refactoring and addressing technical debt.
  - Respond to feedback from users and stakeholders, making updates as needed.
- **Tester**:
  - Update test cases based on new features and changes.
  - Improve testing processes based on feedback and identified issues.
- **Database**:
  - Refine database performance and optimize queries and structures based on feedback.
  - Implement backup and disaster recovery plans as the application grows.
- **DevOps**:
  - Continuously improve infrastructure automation, scaling, and security practices.
  - Gather feedback on deployment processes and make improvements to reduce downtime and improve deployment speed.

————————

Day-2

Source Code Management Tool: Git

Git is a widely-used distributed version control system designed to track changes in source code during software development. It helps developers collaborate, maintain version history, and manage their codebase effectively.

Key Features of Git

1. **Version Control**:
   - Tracks changes to files and directories.
   - Enables reverting to previous versions when necessary.
2. **Distributed System**:
   - Every developer has a full copy of the repository, including the complete history.
   - No reliance on a central server for most operations.
3. **Branching and Merging**:
   - Allows developers to work on isolated branches for features or fixes.
   - Facilitates easy merging of changes into the main codebase.
4. **Collaboration**:
   - Enables multiple developers to work on the same project

simultaneously.
    - Provides tools to resolve conflicts when changes overlap.
5. **Staging Area**:
    - Changes can be reviewed before committing to the repository.
    - Encourages better commit hygiene.
6. **Lightweight**:
    - Efficient in terms of performance and storage.
    - Handles large projects with ease.

Types of SCM Tools

SCM tools can be categorized into two types:

1. **Centralized Version Control Systems (CVCS)**:
    - **Example**: Subversion (SVN), Perforce.
    - A single central server hosts the code repository.
    - Developers check out files and work on them.
2. **Distributed Version Control Systems (DVCS)**:
    - **Example**: Git, Mercurial.
    - Each developer has a full copy of the repository.
    - Changes can be pushed and pulled between repositories

Popular SCM Tools

1. **Git**:
    - Distributed system, highly scalable.
    - Most widely used in modern development workflows.
    - Example platform: GitHub, GitLab, Bitbucket.
2. **Subversion (SVN)**:
    - Centralized system, popular for older projects.
    - Good for managing large binary files.
3. **Mercurial**:
    - Distributed system similar to Git.
    - Known for its simplicity and performance.
4. **Perforce**:
    - Centralized system, often used in enterprise environments.
    - Handles large codebases and binary files effectively.
5. **Azure DevOps**:
    - Integrated with CI/CD pipelines and project management tools.
    - Supports both Git and TFVC (Team Foundation Version Control).

Real-Life Use Case

Imagine a team of developers working on a web application:

- **Repository Setup**: A Git repository is set up to manage the project's source code.
- **Branching**: Each developer creates their own branch for new features or bug fixes.
- **Pull Requests**: Developers submit their changes via pull requests,

which are reviewed before merging into the main branch.
- **CI/CD Integration**: Upon merging, the CI/CD pipeline builds, tests, and deploys the updated application automatically.

Common Git Commands

Command	Description
git init	Initializes a new Git repository in the current directory.
git clone <repo>	Clones a repository from a remote server to your local system.
git status	Displays the status of the working directory and staging area.
git add <file>	Adds changes in a file to the staging area.
git commit -m "<msg>"	Commits staged changes with a descriptive message.
git push	Pushes committed changes to a remote repository.
git pull	Fetches and merges changes from a remote repository into the current branch.
git branch	Lists all branches in the repository.
git checkout <branch>	Switches to a different branch.
git merge <branch>	Merges another branch into the current branch.
git log	Displays a history of commits in the repository.

Key States of Files in Git

1. **Untracked**:
    - A new file in the working directory that Git is not tracking yet.
    - Example: You create a new file but haven't added it to Git.
2. **Tracked**:
    - Files that Git is aware of and is monitoring for changes.
    - These files can be in one of the following states:
        - **Unmodified**: The file hasn't been changed since the last commit.
        - **Modified**: The file has been changed but not yet staged.

- **Staged**: Changes have been added to the staging area and are ready to be committed.

Git Lifecycle Stages

1. **Working Directory**:
   - The local directory where your project files reside.
   - You create, edit, and delete files here.
   - Any changes made here are not tracked until explicitly added to the staging area.
2. **Staging Area**:
   - A temporary area where changes are prepared before committing.
   - Allows developers to organize which changes they want to commit.
   - Files in this area are tracked and ready to be committed.
3. **Git Repository**:
   - The .git directory inside your project.
   - Stores all the committed changes and history.
   - Tracks the state of your project over time.

Git Lifecycle Commands

1. **Untracked → Staged**:
   - Use git add <file> to move untracked files to the staging area.
2. **Modified → Staged**:
   - Use git add <file> to stage modified files for commit.
3. **Staged → Committed**:
   - Use git commit -m "commit message" to save staged changes to the repository.
4. **Committed → Pushed**:
   - Use git push to send committed changes to a remote repository like GitHub.
5. **Committed → Modified**:
   - Edit a file that has already been committed to make it "modified."

Git Workflow

1. **Clone or Initialize**:
   - Clone an existing repository using git clone <repository-url>.
   - Or initialize a new repository using git init.
2. **Edit Files**:
   - Work on your project files in the working directory.
3. **Stage Changes**:
   - Use git add <file> or git add . to stage changes.
4. **Commit Changes**:
   - Use git commit -m "commit message" to save your changes to the repository.
5. **Push to Remote**:
   - Push committed changes to a remote repository using git push.
6. **Pull Updates**:

    - Pull updates from the remote repository using git pull to sync your local repository.
7. **Branching**:
    - Create a new branch for feature development using git branch <branch-name>.
    - Switch branches using git checkout <branch-name> or git switch <branch-name>.
8. **Merge Changes**:
    - Merge feature branches into the main branch using git merge.
9. **Resolve Conflicts**:
    - Handle merge conflicts if changes from different branches overlap.

States Visualization

Benefits of Git

1. **Efficient Collaboration**:
    - Multiple team members can work on the same project without overwriting each other's changes.
    - Distributed nature allows offline work.
2. **Reliable Version History**:
    - Tracks every change made, by whom, and when.
    - Enables reverting to stable versions.
3. **Branching and Isolation**:
    - Facilitates development of features in isolation, without disturbing the main codebase.
4. **Open-Source and Versatile**:
    - Free to use and compatible with most platforms and development environments.

1. Installing Git on Windows

Steps to Install Git on Windows:

1. **Download Git for Windows**:
    - Go to the official Git website: https://git-scm.com/download/win
    - The download should start automatically. If not, you can click the link to manually download the installer.
2. **Run the Git Installer**:
    - Once the installer is downloaded, double-click the .exe file to run the Git setup.
    - Follow the installation prompts. You can select the default options for most users.

3. **Configure Git**:
   - After installation, open **Git Bash** (installed as part of the Git installation) or a command prompt and set up your Git configuration:bash
     Copy code

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

4. **Verify the Installation**:
   - To verify that Git is successfully installed, open **Git Bash** or **Command Prompt** and type:
     bash
     Copy code

     git --version
   -

   - You should see an output like:
     git version 2.x.x.windows

Practical Example: Using Git Commands

1. Initialize a Git Repository
First, let's create a new project folder and initialize a Git repository in that folder.

```
mkdir git-demo
cd git-demo
git init
```

Effect: A .git directory is created, and the folder is now a Git repository.

2. **Create a File and Make Changes**

echo "This is a sample README file." > README.md

3. **Check Git Status**

Stage the File for Commit

git add README.md

Commit the Changes

git commit -m "Initial commit with README"

6. **View Commit Logs**

git log

7. **Make More Changes to the File**

echo "Adding more details to the README." >> README.md

8. **Check Git Status Again**

9. **Commit the New Changes**

git add README.md
git commit -m "Update README with more details"

10. **Check the Git Log Again**

git log

11. **Revert a Commit (Undo Changes)**

git revert abc5678

Effect: This will create a new commit that undoes the changes made in abc5678

12. **Check the Status After Reverting**

git status

13. Restore a File to Its Previous State
If you make changes to a file and want to discard those changes (without reverting the commit), you can use git restore.
Let's modify README.md again.

echo "This is an additional change that we will discard." >> README.md

Now, restore the file to the last committed version.

git restore README.md

Effect: This restores the file to its state as it was in the last commit, discarding your recent changes.

Summary of Git Commands Used:

1. **git init**: Initializes a new Git repository.
2. **git add**: Stages changes to be committed.
3. **git commit**: Commits staged changes to the repository.
4. **git log**: Views the commit history.
5. **git revert**: Reverts a commit by creating a new commit that undoes the changes.
6. **git restore**: Restores a file to its last committed state.
7. **git status**: Displays the current state of the repository, showing untracked, modified, or staged files.

——

1. Git Branch, Common Types of Branches, Checkout, Diff, Merge
1.1 Git Branch
Branches in Git are used to create independent development environments.

- **Common Branch Types**:
  - **Master/Main**: The default branch for production-ready code.
  - **Feature Branches**: Used for developing new features.
  - **Hotfix Branches**: For critical fixes in production.
  - **Development Branches**: Where all active development happens before merging into main.

git branch feature-login

git branch

1.2 Git Checkout

git checkout feature-login

Git Diff

git diff main feature-login

1.4 Git Merge

Switch to the target branch

git checkout main

Merge the feature branch

git merge feature-login