DEVOPS (D75PE1C)

Unit-1

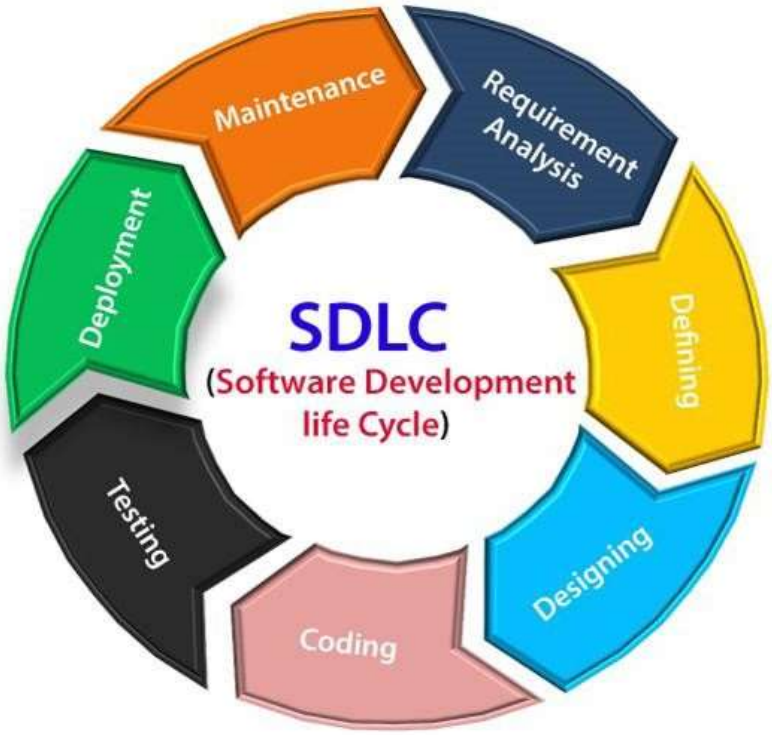
Introduction to DevOps: Introduction, Agile development model, DevOps and ITIL. DevOps process and Continuous Delivery, Release management, Scrum, Kanban, delivery pipeline, identifying bottlenecks.

Introduction

History

Software Development Life Cycle (SDLC)

A software life cycle model (also termed process model) is a pictorial and diagrammatic representation of the software life cycle. A life cycle model represents all the methods required to make a software product transit through its life cycle stages. It also captures the structure in which these methods are to be undertaken.



Stage1: Planning and requirement analysis

Requirement Analysis is the most important and necessary stage in SDLC.

The senior members of the team perform it with inputs from all the stakeholders and domain experts or SMEs in the industry.

Planning for the quality assurance requirements and identifications of the risks associated with the projects is also done at this stage. Business analyst and Project organizer set up a meeting with the client to gather all the data like

what the customer wants to build, who will be the end user, what is the objective of the product.

Before creating a product, a core understanding or knowledge of the product is very necessary.

For Example, A client wants to have an application which concerns money transactions. In this method, the requirement has to be precise like what kind of operations will be done, how it will be done, in which currency it will be done, etc.

Once the required function is done, an analysis is complete with auditing the feasibility of the growth of a product. In case of any ambiguity, a signal is set up for further discussion.

Once the requirement is understood, the SRS (Software Requirement Specification) document is created. The developers should thoroughly follow this document and also should be reviewed by the customer for future reference.

Stage2: Defining Requirements

Once the requirement analysis is done, the next stage is to certainly represent and document the software requirements and get them accepted from the project stakeholders.

This is accomplished through "SRS"- Software Requirement Specification document which contains all the product requirements to be constructed and developed during the project life cycle.

Stage3: Designing the Software

The next phase is about to bring down all the knowledge of requirements, analysis, and design of the software project. This phase is the product of the last two, like inputs from the customer and requirement gathering.

Stage4: Developing the project

In this phase of SDLC, the actual development begins, and the programming is built. The implementation of design begins concerning writing code. Developers have to follow the coding guidelines described by their management and programming tools like compilers, interpreters,debuggers, etc. are used to develop and implement the code.

Stage5: Testing

After the code is generated, it is tested against the requirements to make sure that the products are solving the needs addressed and gathered during the requirements stage. During this stage, unit testing, integration testing, system testing, acceptance testing are done.

Stage6: Deployment

Once the software is certified, and no bugs or errors are stated, then it is deployed.

Then based on the assessment, the software may be released as it is or with suggested enhancement in the object segment.

After the software is deployed, then its maintenance begins.

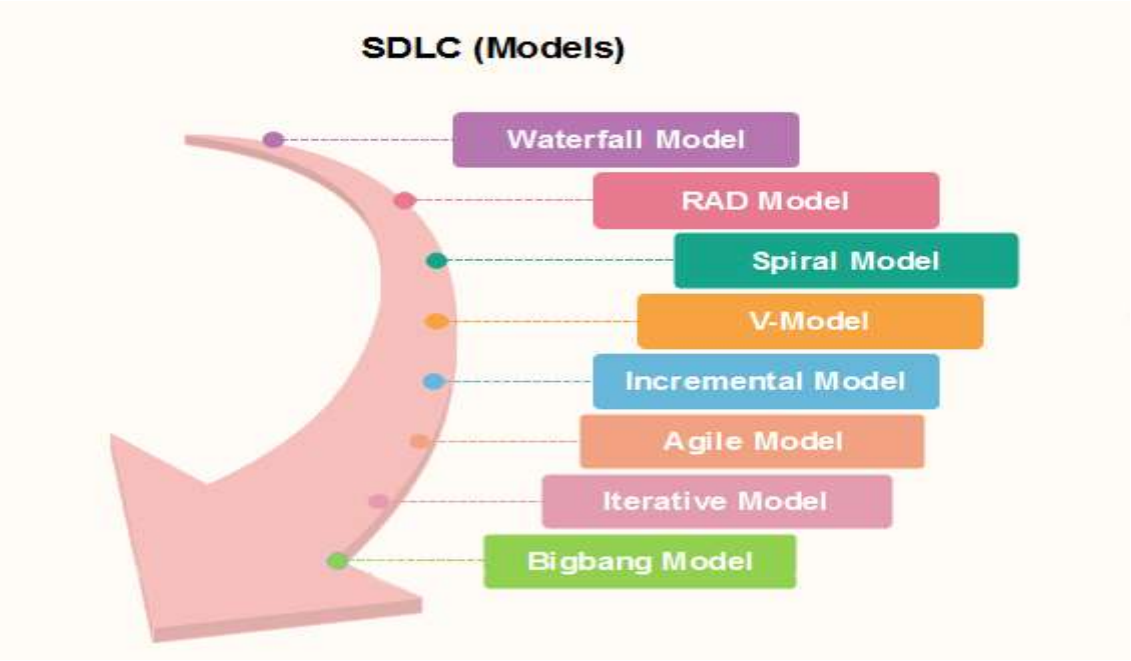
Stage7: Maintenance

Once when the client starts using the developed systems, then the real issues come up and

requirements to be solved from time to time.

This procedure where the care is taken for the developed product is known as maintenance.

SDLC MODELS



Software Development Life Cycle (SDLC) Models

The Software Development Life Cycle (SDLC) is a structured process for developing software applications. Various models have been developed to manage and streamline this process. Here are some of the most common SDLC models:

There are various Software development models or methodologies. They are as follows:

1. Waterfall model
2. V model
3. Incremental model
4. RAD model
5. Agile model
6. Iterative model
7. Spiral model
8. Prototype model

Waterfall Model:

Description: A linear and sequential approach where each phase must be completed before the next begins.

Phases: Requirements, Design, Implementation, Testing, Deployment, Maintenance.

Advantages: Simple and easy to understand; works well for small projects with well-defined requirements.

Disadvantages: Inflexible to changes; difficult to go back to previous phases.

V-Model (Validation and Verification Model):

Description: An extension of the Waterfall model that emphasizes verification and validation. Each development phase has a corresponding testing phase.

Phases: Requirements, System Design, Architecture Design, Module Design, Coding, Unit Testing, Integration Testing, System Testing, Acceptance Testing.

Advantages: Early detection of defects; clear project milestones.

Disadvantages: Still inflexible; not suitable for projects with unclear requirements.

Incremental Model:

Description: Development is carried out in small, manageable increments or modules. Each increment adds functional capabilities.

Advantages: Allows for partial implementation; easier to manage risk; accommodates changing requirements.

Disadvantages: Requires good planning and design; may lead to integration issues.

Iterative Model:

Description: Involves building a prototype and refining it through multiple iterations based on feedback.

Advantages: Flexibility to adapt to changes; focuses on customer feedback.

Disadvantages: Can lead to scope creep if not managed well; may require more resources.

Spiral Model:

Description: Combines iterative development with the systematic aspects of the Waterfall model. It emphasizes risk assessment and minimizes risks through iterative cycles (spirals).

Phases: Planning, Risk Analysis, Engineering, Evaluation.

Advantages: Focus on risk management; allows for incremental releases.

Disadvantages: Can be complex; requires extensive documentation and expertise.

Agile Model:

Description: An iterative and incremental approach that emphasizes collaboration, customer feedback, and flexibility. Agile frameworks (e.g., Scrum, Kanban) are commonly used.

Advantages: Highly adaptable; promotes continuous delivery and improvement; fosters strong team collaboration.

Disadvantages: Requires cultural change in organizations; less emphasis on documentation.

DevOps Model:

Description: Integrates development and operations teams to improve collaboration, efficiency, and deployment frequency.

Advantages: Continuous integration and delivery; faster time to market; improved collaboration.

Disadvantages: Requires cultural shifts and investment in tools; can be challenging to implement.

Waterfall model

Winston Royce introduced the Waterfall Model in 1970.This model has five phases:

Requirements analysis and specification, design, implementation, and unit testing, integration and system testing, and operation and maintenance. The steps always follow in this order and do not overlap. The developer must complete every phase before the next phase begins. This model is named "**Waterfall Model**", because its diagrammatic representation resembles a cascade of waterfalls.

1. Requirements analysis and specification phase: The aim of this phase is to understand the exact requirements of the customer and to document them properly. Both the customer and the software developer work together so as to document all the functions, performance, and interfacing requirement of the software. It describes the "what" of the system to be produced and not "how."In this phase, a large document called **Software Requirement Specification (SRS)** document is created which contained a detailed description of what the system will do in the common language.

2. Design Phase: This phase aims to transform the requirements gathered in the SRS into a suitable form which permits further coding in a programming language. It defines the overall software architecture together with high level and detailed design. All this work is documented as a Software Design Document (SDD).

3. Implementation and unit testing: During this phase, design is implemented. If the SDD is complete, the implementation or coding phase proceeds smoothly, because all the information needed by software developers is contained in the SDD. During testing, the code is thoroughly examined and modified. Small modules are tested in isolation initially. After that these modules are tested by writing some overhead code to check the interaction between these modules and the flow of intermediate output.

4. Integration and System Testing: This phase is highly crucial as the quality of the end product is determined by the effectiveness of the testing carried out. The better output will lead to satisfied customers, lower maintenance costs, and accurate results. Unit testing determines the efficiency of individual modules. However, in this phase, the modules are tested for their interactions with each other and with the system.

5. Operation and maintenance phase: Maintenance is the task performed by every user once the software has been delivered to the customer, installed, and operational.

Advantages of Waterfall model

o This model is simple to implement also the number of resources that are required for it is minimal.

o the requirements are simple and explicitly declared; they remain unchanged during the entire project development.

o the start and end points for each phase is fixed, which makes it easy to cover progress.

o the release date for the complete product, as well as its final cost, can be determined before development.

O It gives easy to control and clarity for the customer due to a strict reporting system.

Disadvantages of Waterfall model

o In this model, the risk factor is higher, so this model is not suitable for more significant and complex projects.

o This model cannot accept the changes in requirements during development.

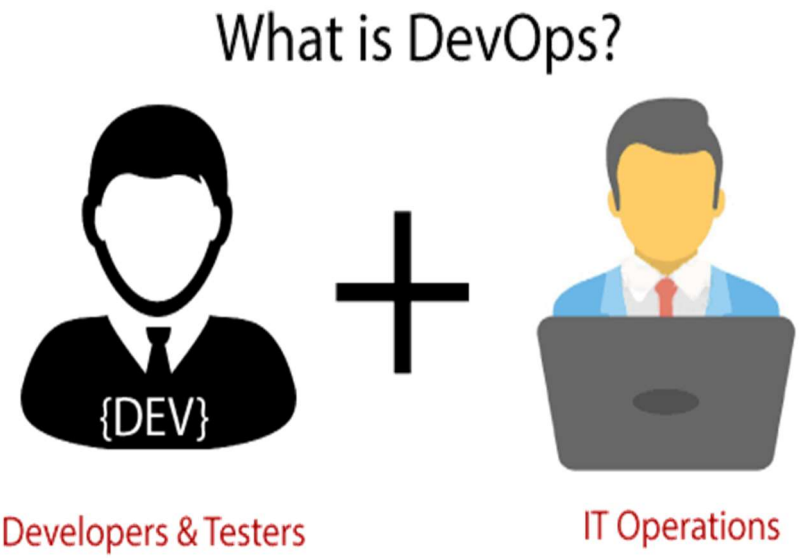
o It becomes tough to go back to the phase. For example, if the application has now shifted to the coding phase, and there is a change in requirement, It becomes tough to go back and change it.

o Since the testing done at a later stage, it does not allow identifying the challenges and risks in the earlier phase, so the risk reduction strategy is difficult to prepare.

Introduction

The DevOps is the combination of two words, one is **Development** and other is **Operations**. It is a culture to promote the development and operation process collectively. The DevOps tutorial will help you to learn DevOps basics and provide depth knowledge of various DevOps tools such as **Git, Ansible, Docker, Puppet, Jenkins, Chef, Nagios** and **Kubernetes**.

What is DevOps?



Why DevOps?

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

o the operation and development team worked in complete isolation.

o After the design-build, the testing and deployment are performed respectively. That's why they consumed more time than actual build cycles.

o 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.

o Manual code deployment leads to human errors in production.

o Coding and operation teams have their separate timelines and are not in synch, causing further delays.

DevOps History

2009 – DevOps Begins

- DevOpsDays conference held in Ghent, Belgium, started by Patrick Debois.
- Marked the origin of DevOps to bridge gaps between development and operations.

2012 – State of DevOps Report Starts

- Launched by **Alanna Brown** at **Puppet**
- In this they Started measuring how DevOps practices affect IT performance.

2014 – DevOps Adoption Accelerates/DevOps Grows Rapidlys

- **Nicole Forsgren**, **Gene Kim**, and **Jez Humble** joined the research
- Found that DevOps adoption was **accelerating** across organizations.
- They published that DevOps practices like automation, version control, and continuous integration significantly improved speed and stability.
- From this year (2014), many Tech- Focused Organizations like Amazon, Netflix, Etsy started adapting DevOps practices

2015 – DORA is Founded

- **DevOps Research and Assessment (DORA)** founded by Forsgren, Humble, and Kim.
- They Introduced **Four Key Metrics** for DevOps success:

    1. Deployment frequency - *How often new releases are delivered.*
    2. Lead time for changes - *How quickly code goes from development to production*
    3. Change failure rate - *How often deployments cause issues*
    4. Time to restore service - *How fast a system recovers from failure.*

2017 – "Accelerate" Book Published

- Book: **"Accelerate"** by Forsgren, Kim, and Humble - scientific proof that DevOps improves speed, quality, and business outcomes.
- The book combined **years of data and scientific research** to show that DevOps not only improves IT performance, but also helps businesses grow faster

2018- Many Corporate Company Started Adopting DevOps.

- At this stage, DevOps was seen as a **competitive advantage**. Companies realized that improving collaboration between development and operations teams could reduce failure rates and improve customer satisfaction.
- Some key drivers for this shift included:
    - Need for faster software releases
    - Customer demand for real-time services
    - Migration to cloud platforms (AWS, Azure, GCP)

- Automation tools maturing (Jenkins, Docker, Kubernetes, Ansible, Terraform)

2019- Industry Standard

- The Industry Standard was released or came into existence. It is now widely used in many sectors like Banking, Healthcare, e-commerce, manufacturing, and even in government Organizations.

- It was also Integrated into job roles, Project workflows and cloud-native-Development.

Agile development Model

An agile methodology is an iterative and incremental approach to software development. Each iteration of agile methodology takes a short time interval of 1 to 4 weeks. The agile development process is aligned to deliver the changing business requirement. It distributes the software with faster and fewer changes.

The single-phase software development takes 6 to 18 months. In single-phase development, all the requirement gathering and risks management factors are predicted initially.

The agile software development process frequently takes the feedback of workable product. The workable product is delivered within 1 to 4 weeks of iteration.

What is the Agile Methodology?

Agile methodologies are iterative and incremental, which means it's known for breaking a project into smaller parts and adjusting to changing requirements.

1. They prioritize flexibility, collaboration, and customer satisfaction.
2. Major companies like Facebook, Google, and Amazon use Agile because of its adaptability and customer-focused approach.

Agile Manifesto:

- The principles outlined in the Agile Manifesto enable businesses to strive towards excellence.
- The manifesto promotes trust, transparency, collaboration, and consumer involvement.
- While software strategies may seem basic, it is always critical to work with experts that
- understand the significance of delivering a quality product and maintaining customer satisfaction.

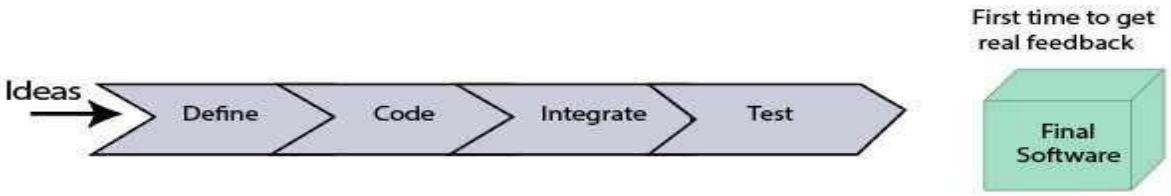
The Agile Manifesto has four key values:

1. Individuals and interactions over processes and tools

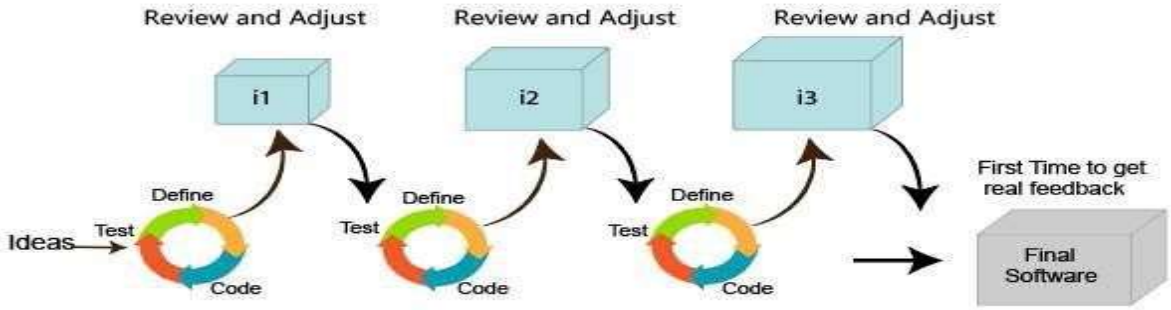
2. Working software over comprehensive documentation

3. Customer collaboration over contract negotiation

4. Responding to change over following a plan

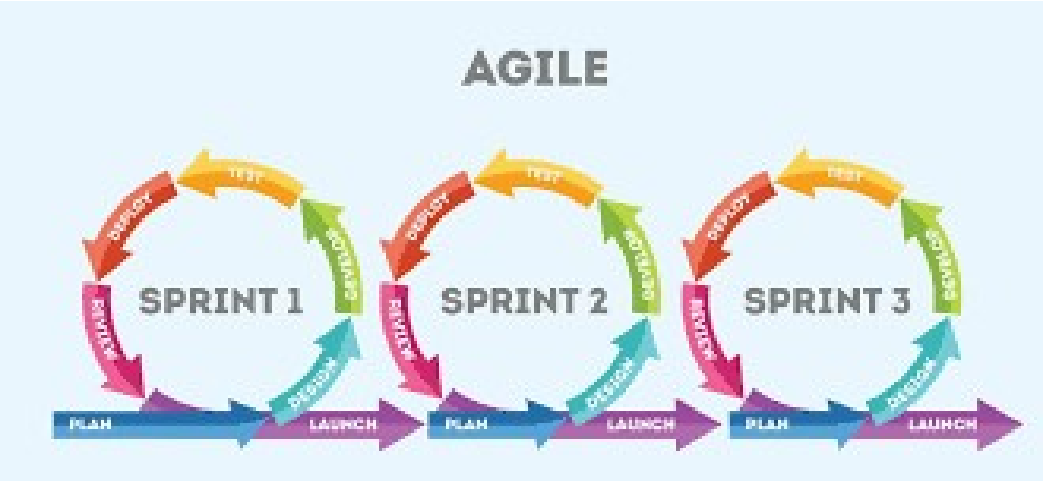


Traditional Method



Agile Method

Life cycle of Agile Methodology



1. Requirement Gathering

- In this stage, the project team identifies and documents the needs and expectations of various stakeholders, including clients, users, and subject matter experts.
- It involves defining the project's scope, objectives, and requirements.
- Establishing a budget and schedule.
- Creating a project plan and allocating resources.

2. Design

- Developing a high-level system architecture.
- Creating detailed specifications, which include data structures, algorithms, and interfaces.
- Planning for the software's user interface.

3. Development (Coding)

Writing the actual code for the software. Conducting unit testing to verify the functionality of individual components.

4. Testing

This phase involves several types of testing:

1. **Integration Testing:** Ensuring that different components work together.
2. **System Testing:** Testing the entire system as a whole.
3. **User Acceptance Testing:** Confirming that the software meets user requirements.
4. **Performance Testing:** Assessing the system's speed, scalability, and stability.

5. Deployment

1. Deploying the software to a production environment.

2. Put the software into the real world where people can use it.
3. Make sure it works smoothly in the real world.
4. Providing training and support for end-users.

6. Review (Maintenance)

1. Addressing and resolving any issues that may arise after deployment.
2. Releasing updates and patches to enhance the software and address problems.

History of Agile

In 1957, people started figuring out new ways to build computer programs. They wanted to make the process better over time, so they came up with iterative and incremental methods.

In the 1970s, people started using adaptive software development and evolutionary project management. This means they were adjusting and evolving how they built software.

In 1990s, there was a big change. Some people didn't like the strict and super-planned ways of doing things in software development. They called these old ways "waterfall." So, in response, lighter and more flexible methods showed up. These included:

- Rapid Application Development (RAD) in 1991.
- Unified Process (UP), Dynamic Systems Development Method (DSDM) in 1994.
- Scrum in 1995.
- Crystal Clear and Extreme Programming (XP) in 1996.
- Feature-Driven Development (FDD) in 1997.

Agile is the process that follows concept of agile manifestation held in in Feb 2001

Agile manifesto contains 12 principles of agile software development.

Understanding the 12 Agile principles

The 12 Agile principles guide teams on how to work more flexibly and respond quickly to changes, which is key in project management. Here's a quick overview of the 12 Agile principles:

1. Make customers happy through early and continuous delivery of useful software.
2. Embrace changing requirements, even in later stages.
3. Deliver work frequently, from a couple of weeks to a couple of months, with a preference for the shorter timescale.
4. Stakeholders and developers must work together daily throughout the project.
5. Build projects around motivated individuals, giving them the environment and support they need, and trusting them to get the job done.

6. Face-to-face conversations are the most effective method of communication.
7. The main measure of progress is working software.
8. The working pace should be constant yet sustainable.
9. Pay continuous attention to technical excellence and good design.
10. Keep things as simple as possible.
11. The best results come from self-organizing teams.
12. The team reflects on how to become more effective at regular intervals, adjusting behaviour accordingly.

Advantages

1) Customer satisfactions
2) Customers, developers, tester constantly interact with each other
3) Working software is delivered frequently
4) Face to face conversion is the best from of communication
5) Even late changes in requirement are welcomed
6) Continuous attention to technical excellence and good design.

Disadvantages

1) Lack of documentation
2) Lack of planning and project management software
3) Team must be knowledgeable
4) Project can easily take off track is costumer is not satisfied.

Methodologies that are used to implement Agile

1) Extreme programming (XP)
2) Feature driven deleopment(FDD)
3) Kanban
4) Scrum

==========

SCRUM

Agile Scrum methodology:

Agile is the process model whereas the Scrum is the kind of frame work through which we develop, test, release software.

Agile define principle i.e, how process must be and scrum is a frame work that follows agile principles.

Scrum is a framework within the Agile methodology that focuses on iterative development, collaboration, and flexibility. It is designed to improve the efficiency and effectiveness of project management, particularly in software development.

Scrum is a framework used by teams to manage work and solve problems collaboratively in short cycles. Scrum implements the principles of Agile as a concrete set of artifacts, practices,and roles

Scrum is the frame work though which we build software product by following agile principles.

Scrum Framework:

Core Components:

Scrum team: A typical team has between five and nine people, but projects can easily scale into the hundreds, or just be one-or two-person teams. Everyone on the project works together to complete the set of work they have collectively committed to complete within a sprint.

Teams develop a deep form of camaraderie and a feeling that "we're all in this together."

Product Owner: Defines and prioritizes the product backlog. Ensures that the team delivers value to the business. Represents the stakeholders and is responsible for maximizing the value of the product. They manage the Product Backlog, ensuring it is prioritized and clear

Scrum Master: Facilitates Scrum processes, helps remove impediments, and ensures adherence to Scrum practices. Acts as a facilitator and coach for the Scrum Team. They help remove impediments, ensure that Scrum practices are followed, and support the team in improving their processes.

Development Team: Self-organizing group responsible for delivering potentially releasable increments of the product. Ex – developer and tester. A cross-functional group responsible for delivering potentially shippable product increments. They are self-organizing and work collaboratively to meet their goals.

Artifacts:

- **Product backlog**: The product backlog is a prioritized features list containing every desired feature or change to the product. Note: The term "backlog" can get confusing because it's used for two different things. To clarify, the product backlog is a list of desired features for the product.

-The **sprint backlog** is a list of tasks to be completed in a sprint.

Events:

- **Sprint:** A time-boxed period, usually lasting 2-4 weeks, during which the Scrum Team works to create a usable product increment. Sprints are of consistent length throughout the project.

-Sprint Planning:

At the start of each sprint, a sprint planning meeting is held, during which the product owner presents the top items on the product backlog to the team.

The team selects the work they can complete during the coming sprint.

That work is then moved from the product backlog to a sprint backlog, which is the list of tasks needed to complete the product backlog items the team has committed to complete in the sprint.

- **Daily Scrum**: Each day during the sprint, a brief meeting called the daily scrum is conducted. This meeting helps set the context for each day's work and helps the team stay on track. All team members are required to attend the daily scrum.

- **Sprint review meeting**: At the end of each sprint, the team demonstrates the completed functionality at a sprint review meeting, during which, the team shows what they accomplished during the sprint.

- **Sprint retrospective**: Also at the end of each sprint, the team conducts a sprint retrospective, which is a meeting during which the team reflects on how well their process is working for them and what changes they may wish to make for it to work even better.

Key Principles:

- **Transparency**: All aspects of the process must be visible to those responsible for the outcome. This includes the Product Backlog, Sprint Backlog, and progress toward goals.

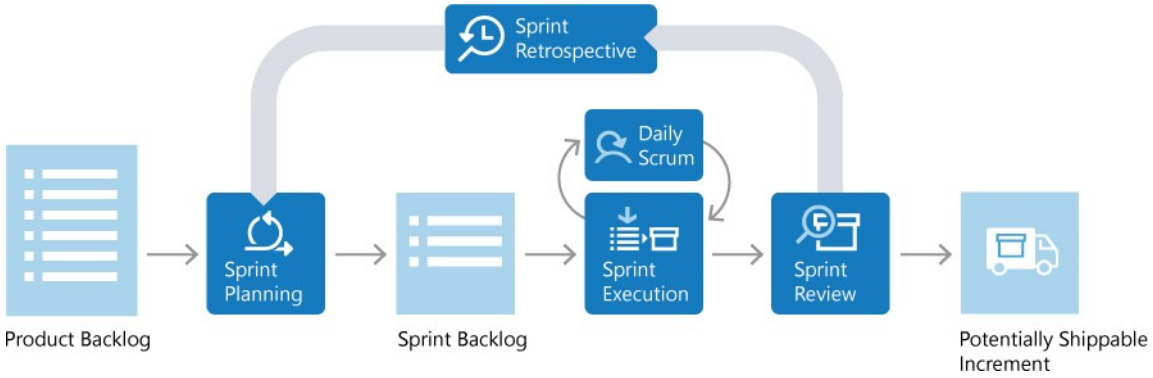
- **Inspection**: Regularly review artifacts and progress to identify any deviations from the plan and make adjustments as needed.

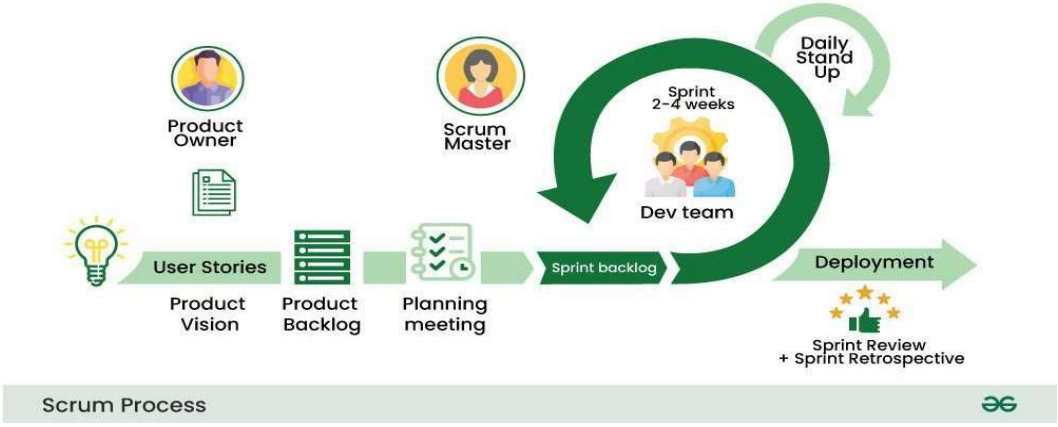
-Adaptation: Make adjustments to processes and practices based on feedback and inspections to improve efficiency and effectiveness.

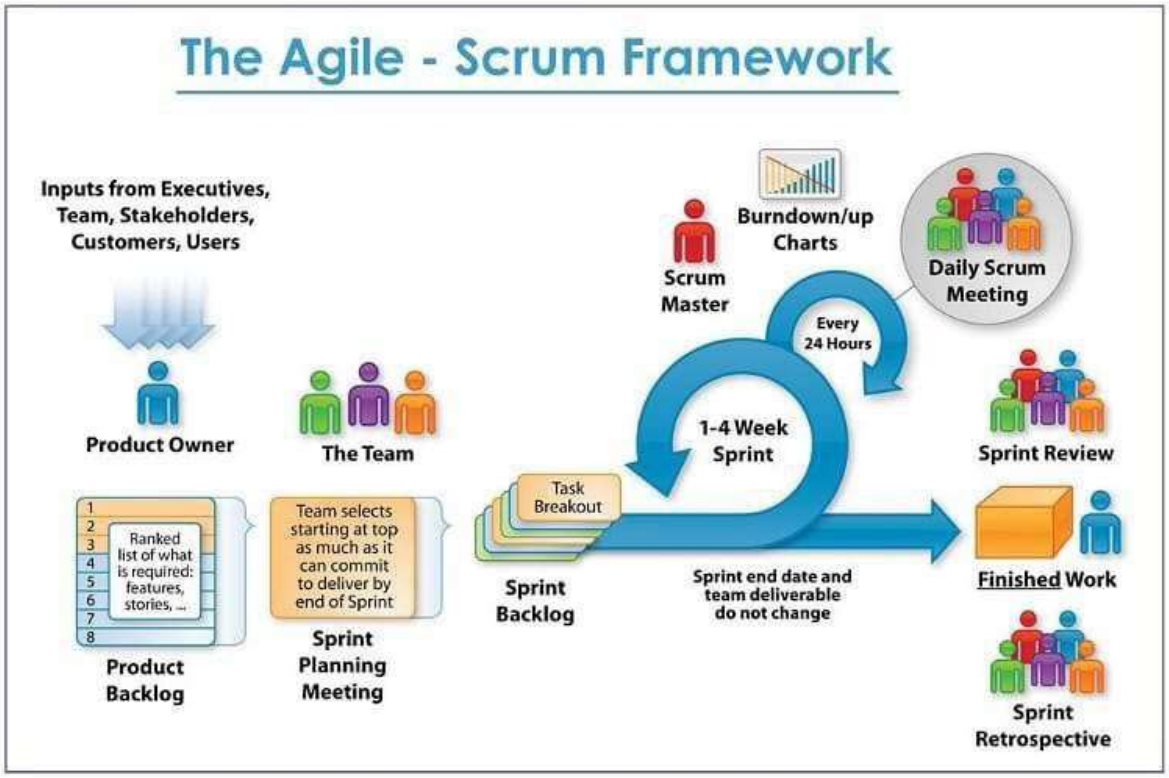
The Scrum lifecycle

The diagram below details the iterative Scrum lifecycle. The entire lifecycle is completed in fixed time

periods called sprints. A sprint is typically one-to-four weeks long







Scrum Framework workflow:

- The product owner creates a product backlog (essentially, a wish list of tasks that need to be prioritized in a project).
- The Scrum team conducts a sprint planning session where the tasks necessary to complete items on the wish list is broken down into small, more easily manageable chunks.
- The team creates a sprint backlog and plans its implementation.
- The team decides a time duration for every sprint (the most common intervals is probably two weeks).

- The team gets together every day for a brief Scrum meeting (often referred to as a Daily Standup) where each member of the team shares daily updates, helping the team and the project manager assess the progress of the project.
- The **certified Scrum Master** guides the team and keeps them focused and motivated.
- The stakeholders and the product owner conduct a review at the end of each sprint.

This is the cycle followed by a Scrum team in a product development project. The three roles mentioned above - the Product Owner, the Scrum Team, and the ScrumMaster together play a major role in exercising this framework.

ADVANTAGES OF SCRUM:

- Scrum can help teams complete project deliverables quickly and efficiently.
- Scrum ensures effective use of time and money.
- Large projects are divided into easily manageable sprints.
- Works well for fast-moving development projects.
- The team gets clear visibility through scrum meetings.
- Scrum, being agile, adopts feedback from customers and stakeholders.
- Short sprints enable changes based on feedback a lot more easily.
- The individual effort of each team member is visible during daily scrum meetings.

DISADVACTAGES OF SCRUM:

- Scrum often leads to scope creep, due to the lack of a definite end-date.
- The chances of project failure are high if individuals aren't very committed or cooperative.
- Adopting the Scrum framework in large teams is challenging.
- The framework can be successful only with experienced team members.
- Daily meetings sometimes frustrate team members.
- If any team member leaves in the middle of a project, it can have a huge negative impact on the project.

Quality is hard to implement until the team goes through an aggressive testing process

KANBAN:

Kanban is a popular framework which is used to implement agile and devops software development.

Kanban is a Japanese term that means signboard or billboard. An industrial engineer named Taiichi Ohno developed Kanban at Toyota Motor Corporation to improve manufacturing efficiency.

Although Kanban was created for manufacturing, software development shares many of the same goals, such as increasing flow and throughput. Software development teams can improve their efficiency and deliver value to users faster by using Kanban guiding principles and methods.

Kanban methodology is an agile method that aims at continuous improvement, flexibility in task management, and enhanced workflow.

Kanban is one of the simplest frameworks used as it allows project managers to efficiently manage and keep track of their projects. A distinguishing feature of the Kanban framework among different agile methodologies is its compatibility with the existing organizational setting.

Kanban Framework focuses on visualizing the entire project on boards in order to increase project transparency and collaboration between team members.

The work items are represented in a kanban board visually, allowing team members to see the state of every piece of work at any time.

KANBAN BOARD:

The kanban board is the agile project management tool that designed the necessary visualized work, limited work-in-progress, and maximizes flow (or efficiency).

It is a tool that visualizes the entire project to track the flow of their project.

It uses cards, columns, and provides continuous improvement to help technology and service teams who commit the right amount of work and get it done.

Through this graphical approach of Kanban boards, a new member or an external entity can understand what's happening right now, tasks completed and future tasks.

Kanban board indicates

- TO DO -- the current tasks that are being performed.
- DOING -- the tasks to do in the future.
- DONE -- the tasks that are completed

Core Principles:

1. Visualize the Workflow: Use a board or chart to visualize the workflow, making it easy to track progress and identify bottlenecks.

2. Limit Work in Progress (WIP): Set limits on the amount of work in progress at each stage to ensure focused effort and prevent overwhelm.

3. Focus on Flow: Maintain a smooth flow of work, rather than focusing on individual tasks or iterations.

4. Continuous Improvement: Regularly analyze and improve the workflow through retrospectives and lead time analysis.

5. Pull-Based Workflow: Work is pulled into each stage as capacity allows, rather than pushing work into the next stage.

6. Emphasis on Lead Time: Focus on reducing lead time (time from start to delivery) rather than just focusing on individual task duration.

Key Components:

1. Kanban Board: A visual representation of the workflow, showing columns for each stage (e.g., To-Do, In Progress, Done).

2. Columns: Each column represents a stage in the workflow, with clear definitions and criteria for moving work items between columns.

3. Work Items: Individual tasks or user stories being worked on, represented by cards or sticky notes on the board.

4. WIP Limits: Clearly defined limits on the number of work items in each column to prevent overload.

5. Lead Time: The time it takes for a work item to move through the entire workflow.

6. Cycle Time: The time it takes for a work item to move through a specific stage or set of stages.

Benefits:

1. Improved Workflow Efficiency: Reduced waste, increased productivity, and faster lead times.

2. Increased Transparency: Clear visualization of the workflow and work items.

3. Enhanced Collaboration: Team members have a shared understanding of the workflow and priorities.

4. Adaptability: Kanban is suitable for teams with varying work items and priorities.

Common Kanban Metrics:

1. Lead Time: Average time for a work item to move through the entire workflow.

2. Cycle Time: Average time for a work item to move through a specific stage or set of stages.

3. Throughput: Number of work items completed within a given timeframe.

4. WIP: Number of work items in progress at any given time.

Kanban Roles:

1. Service Delivery Manager: Oversees the workflow and ensures alignment with business objectives.

2. Team Members: Work on individual tasks and collaborate to maintain flow.

3. Replenishment Meeting: Team members meet to discuss priorities and replenish the backlog.

Challenges and Limitations:

1. Overemphasis on WIP Limits: Focusing too much on WIP limits can lead to neglect of other important aspects of the workflow.

2. Lack of Clear Priorities: Without clear priorities, work items may not be pulled into the workflow in the most efficient order.

3. Inadequate Metrics: Without proper metrics, it's difficult to measure the effectiveness of the Kanban system.

Difference between Scrum vs Kanban

Scrum	Kanban
Based on Agile values	Based on Lean values
Requires a change in current processes	Reuses your current processes and doesn't require a massive change in how your team works
Team is required to estimate work	Your team is not required to estimate the tasks
Velocity is a metric to determine when working software can be delivered to a customer or stakeholder	Lead time is a metric to determine when working software can be delivered to a customer or stakeholder
Prescribes 3 roles - **Scrum Master, Product Owner** and members of the **Delivery Team**	Does not have specific or prescribed roles
Can use a Sprint burndown report to show the health of a Sprint	May use a cumulative flow diagram to report progress
Priorities do not change during a Sprint	Priorities can change daily
Great for new product or feature development	Better suited for tasks that are repetitive such as tasks that most would consider, **'Business as Usual'**
Requires a small, dedicated team	Team size is irrelevant
Bottlenecks are not always obvious until your Sprint Review or Retrospective	Bottlenecks are revealed quickly - through visualization

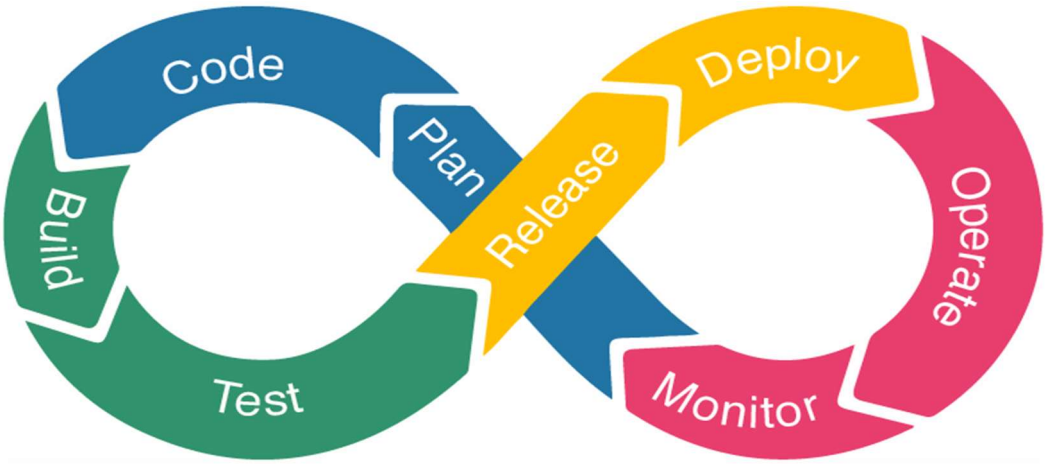
---

DevOps:

Devops is a combination of software development (Dev) and operation (Ops)

It is a software development process which aims to integrate the work of development team and operation team by facilitating a culture of collaboration and shared responsibility.

DevOps Lifecycle Phases:



1. Plan

- Tools: Jira, Trello, VersionOne, CA Agile Central, Confluence

- Description: Define project goals, requirements, and timelines. Identify stakeholders and their roles.

2. Code

- Tools: Git, Jenkins, Eclipse, Visual Studio Code, SonarQube

- Description: Write, review, and commit code. Conduct static code analysis and code reviews.

3. Build

- Tools: Jenkins, Travis CI, CircleCI, GitLab CI/CD, Maven

- Description: Compile, package, and create deployable artifacts. Run automated tests and create build reports.

4. Test

- Tools: Selenium, JUnit, TestNG, Cucumber, Appium

- Description: Conduct unit testing, integration testing, system testing, and acceptance testing.

5. Deploy

- Tools: Ansible, Puppet, Chef, Docker, Kubernetes

- Description: Deploy applications to production, configure environments, and manage infrastructure.

6. Operate

- Tools: Nagios, Prometheus, Grafana, New Relic, Splunk

- Description: Monitor applications, infrastructure, and performance. Identify areas for improvement.

7. Monitor

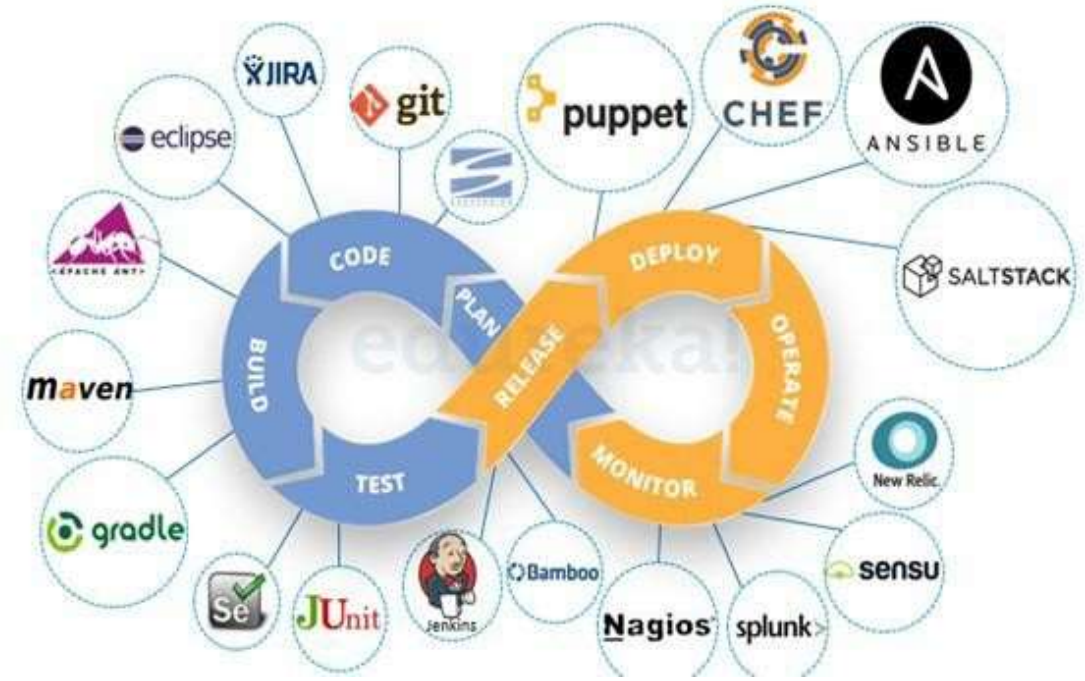
- Tools: ELK Stack, Splunk, New Relic, Datadog, Grafana

- Description: Monitor logs, metrics, and performance. Conduct root cause analysis and troubleshooting.

8. Feedback

- Tools: Slack, JIRA Service Management, Freshservice, Zendesk

- Description: Gather feedback from users, customers, and stakeholders. Use feedback to improve the application and process.



Tools Used by Dev Team are:

Jira: This Software provides planning and roadmap tools so teams can manage stakeholders, budgets, and feature requirements from day one.

- **Git**: It is a distributed version control system that tracks changes in any set of computer files, usually used for coordinating work among programmers collaboratively developing source code during software development.

- **Eclipse**: It is a free, Java-based development platform known for its plugins that allow developers to develop and test code written in other programming languages.

- **Maven**: This is open-source tool from Apache automates the build process and resolution of dependencies and is used primarily for Java projects.

- **Ant**: Ant (Another Neat Tool) is a Java library and a software tool used for automate software build processes such as compile, run, test and assemble Java application. It is designed and developed by Apache Software Foundation.

- **Gradle**: Gradle is a build automation tool known for its flexibility to build software. A build automation tool is used to automate the creation of applications. The building process includes compiling, linking, and packaging the code. The process becomes more consistent with the help of build automation tools.

- **Selenium**: This open-source tool for automating tests for web applications is used by Google, IBM, and other big-name companies. It's used only for web applications—not desktop or mobile ones.

- **JUnit**: It is a unit testing open-source framework for the Java programming language. Java Developers use this framework to write and execute automated tests.

Tools Used by Ops Team are:

- **Puppet**: It lets you manage and automate software inspection, delivery, and operation. This open-source tool has a solid track record and thousands of modules and is easily integrated with many other platforms.

- **Chef**: This is powerful open-source configuration management tool lets you turn infrastructure into a code to manage data, attributes, roles, environments, and more.

- **Docker**: It is a Linux-based open-source platform that focuses on containers, meaning you package up the software with its dependencies and ship everything together as a unit—no need to worry about managing dependencies separately.

- **Ansible**: This is open-source tool for automating software provisioning, configuration management, and application deployment is easy to use, Plus, it's agentless and uses a simple syntax written in the YAML language

- **Nagios**: Its Used to find and correct problems in networks and infrastructure, Nagios is one of the most popular free and open-source monitoring tools.

- **Kubernetes**: A relatively new container orchestration platform lets you manage hundreds of containers.

- **Jenkins**: It's a CICD tool, It is known for quickly finding issues in code. It's a free, opensource tool used for automating the delivery pipeline, and lets you test and report changes almost in real-time

---

ITIL:

ITIL is an abbreviation of **Information Technology Infrastructure Library.**

It is a library of volumes that describes the best practice for delivering IT services.

ITIL (Information Technology Infrastructure Library) is a set of principles for IT service management. It is a highly structured model built to boost productivity and offer statistics for IT teams.

It is a framework which helps the IT professionals for delivering the best services of IT.

This framework is a set of best practices to create and improve the process of ITSM (IT Service Management). It provides a framework within an organization, which helps in planning, measuring, and implementing the services of IT.

Information Technology Service Management- It is the concept that enables an organization to maximize the business values obtained from the use of IT.

The main motive of this framework is that the resources are used in such a way so that the customer get the better services and business get the profit.

ITIL defines and documents IT processes and procedures with a large focus on oversight and planning.

The guidelines cover best practices and tried-and-true processes for everything from incident management to problem management to change management.

The latest version, ITIL 4, which debuted in 2019, continues to provide the guidance needed for organizations to address new service management challenges, using different technology in the era of digital transformation, cloud, and DevOps.

ITIL Life Cycle: -

It is a set of steps that shows how IT Servies are Planned, Build, Delivered overtime. It is like roadmap for managing IT services.

The ITIL lifecycle consists of five stages:

1. Service Strategy: This stage defines IT services and ensures they align with business objectives. Key processes include:

- Service Portfolio Management
- Service Level Management
- Demand Management
- Capacity Management

2. Service Design: This stage designs IT services to meet business requirements. Key processes include:

- Service Catalogue Management
- Service Level Agreement (SLA) Management
- Capacity Management
- Availability Management
- IT Service Continuity Management

3. Service Transition: This stage transitions designed services into production. Key processes include:

- Change Management
- Service Asset and Configuration Management
- Release and Deployment Management
- Service Validation and Testing

4. Service Operation: This stage manages IT services in production. Key processes include:

- Event Management
- Incident Management
- Request Fulfillment
- Problem Management

5. Continual Service Improvement: This stage ensures IT services continually improve. Key processes include:

- Service Reporting
- Service Measurement
- Continual Service Improvement

ITIL Processes

ITIL defines 26 processes across the five lifecycle stages. These processes are:

1. Service Strategy (4 processes)
2. Service Design (7 processes)
3. Service Transition (5 processes)
4. Service Operation (13 processes)
5. Continual Service Improvement (5 processes)

ITIL Roles

ITIL defines several roles, including:

1. Service Manager
2. Process Owner
3. Process Manager

4. Service Desk Manager

5. Technical Manager

Benefits of ITIL

Implementing ITIL can bring numerous benefits, including:

1. Improved IT service quality

2. Increased efficiency

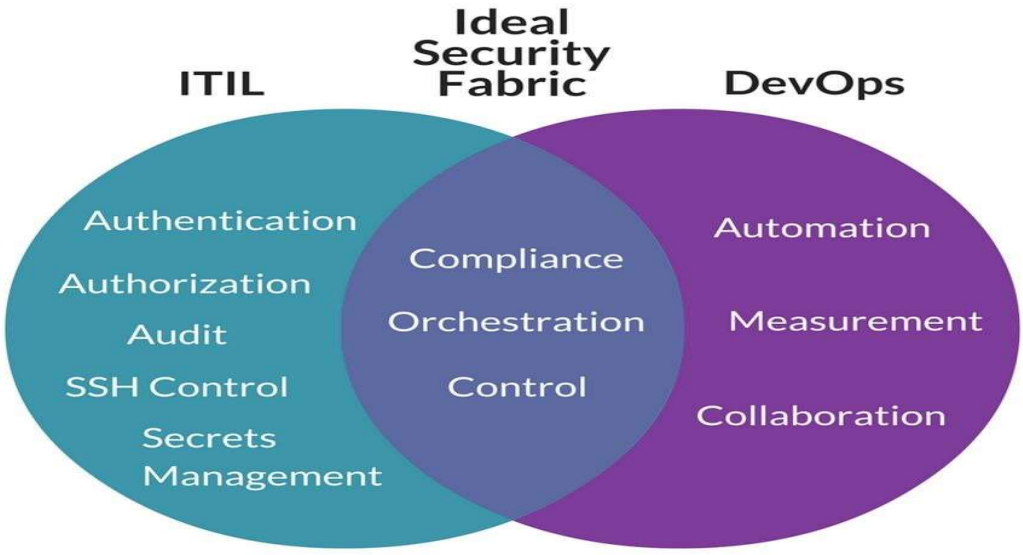
3. Better alignment with business objectives

4. Enhanced customer satisfaction

5. Reduced costs

ITIL and Devops:-

ITIL (Information Technology Infrastructure Library) and DevOps are two popular frameworks that can co-exist and complement each other to achieve efficient IT service management and delivery.



ITIL's Focus:

- Service Management: ITIL focuses on managing IT services to deliver value to customers.

- Standardized Processes: ITIL provides standardized processes and best practices for IT service management.

- Governance and Compliance: ITIL emphasizes governance and compliance to ensure IT services meet business requirements.

DevOps' Focus:

- Agile Delivery: DevOps focuses on agile delivery of IT services through collaboration and automation.

- Culture and Practices: DevOps emphasizes culture and practices that promote collaboration, experimentation, and continuous improvement.

- Automation and Integration: DevOps aims to automate and integrate IT processes to improve efficiency and speed.

Co-Existence Benefits:

- Improved Service Management: ITIL provides a framework for managing IT services, while DevOps improves the delivery and operation of those services.

- Enhanced Collaboration: DevOps promotes collaboration between development and operations teams, while ITIL ensures that IT services meet business requirements.

- Increased Efficiency: ITIL's standardized processes and DevOps' automation and integration improve efficiency and reduce waste.

- Better Governance and Compliance: ITIL's emphasis on governance and compliance ensures that DevOps practices meet business requirements.

Key Integration Points:

- Incident and Problem Management: ITIL's incident and problem management processes can be integrated with DevOps' monitoring and logging tools.

- Change and Release Management: ITIL's change and release management processes can be integrated with DevOps' continuous integration and delivery pipelines.

- Service Catalogue and Configuration Management: ITIL's service catalogue and configuration management processes can be integrated with DevOps' service discovery and infrastructure as code tools.

Best Practices for Co-Existence:

- Start with ITIL Foundation: Establish a solid foundation in ITIL processes and best practices.

- Integrate DevOps Practices: Integrate DevOps practices and tools into existing ITIL processes.

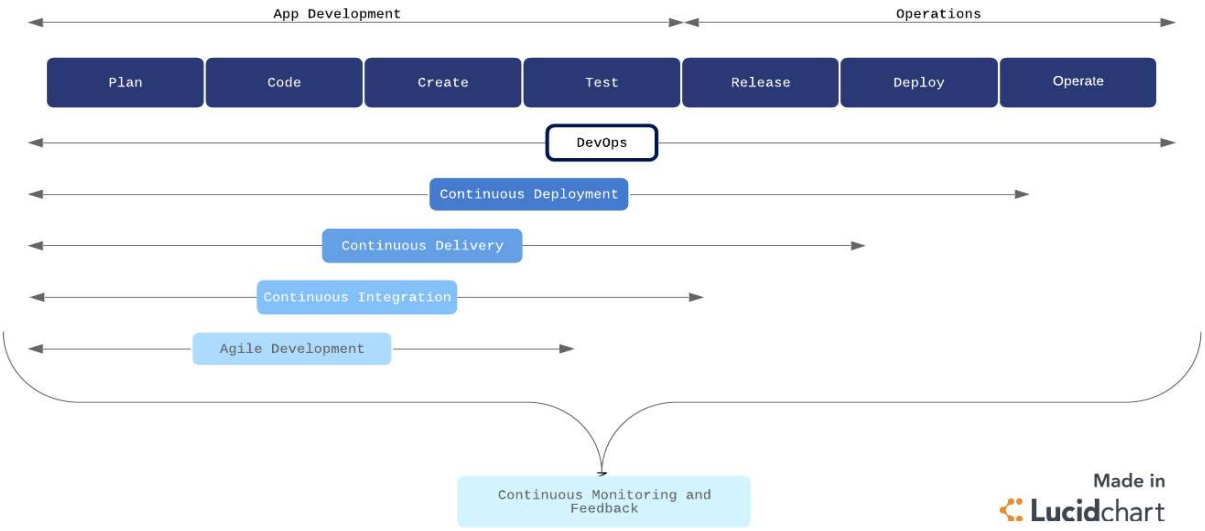
- Focus on Culture and Collaboration: Emphasize culture and collaboration to ensure successful co-existence.

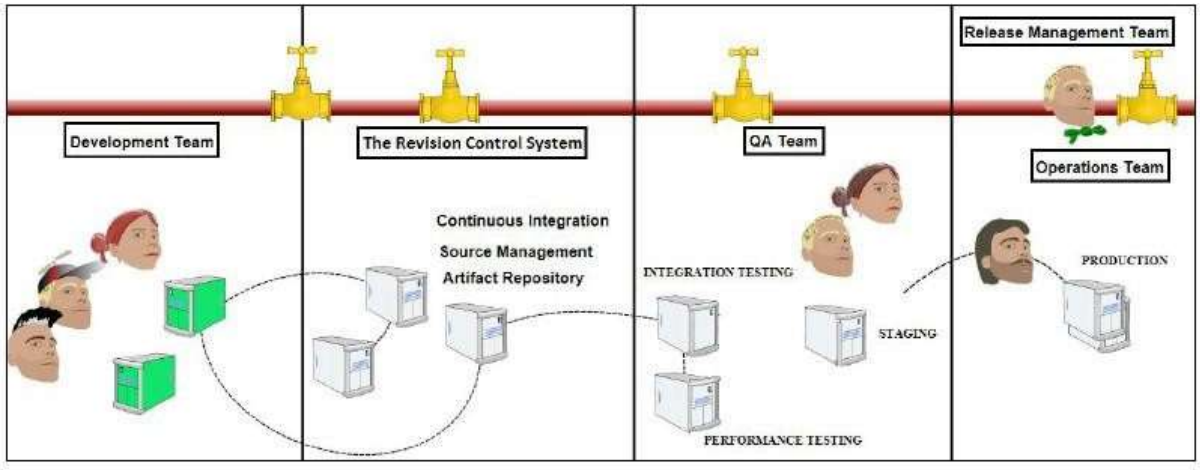
- Continuously Monitor and Improve: Continuously monitor and improve ITIL and DevOps processes to ensure alignment and efficiency.

DevOps Process: -

The DevOps process flow is all about agility and automation. Each phase in the DevOps lifecycle focuses on closing the loop between development and operations and driving production through continuous development, integration, testing, monitoring and feedback, delivery, and deployment

As mentioned earlier, the various phases such as Continuous Planning, Continuous Development, Continuous Integration, Continuous Testing, Continuous Deployment, Continuous monitoring and Continuous Feedback constitute DevOps Life cycle.





Continuous Planning: Continuous Planning process makes business capable of planning delivery of new feature continuously.

New business requirement, Change required in existing functionality, Prioritized defect

Could be planned to deliver continuously and becomes input to implementation team. Requirements management tools, matured communication models, and effective feedback loops make it possible to establish continuous planning.

Continuous Development: This is the phase that involves 'coding'of the software. The vision of the project is decided during the planning phase and the developers begin developing the code for the application. there are a number of tools for maintaining the code. The code can be written in any language, but it is maintained by using Version Control tools. The most popular tools used are Git,SVN, Mercurial, CVS, and JIRA. Also tools like Ant, Maven, Gradle can be used in this phase for building/ packaging the code into an executable file that can be forwarded to any of the next phases.

Continuous Integration: This stage is the heart of the entire DevOps life cycle. It is a software development practice in which the developers require to commit changes to the source code more frequently. This may be on a daily or a weekly basis. Every commit is then built and this allows early detection of problems if they are present. Building code not only involves compilation but it also includes code review, unit testing, integration testing, and packaging. The code supporting new functionality is continuously integrated with the existing code. Since there is continuous development of software, the updated code needs to be integrated continuously as well as smoothly with the systems to reflect changes to the end-users. Jenkins is a very popular tool used in this phase. Whenever there is a change in the Git repository, Jenkins fetches the updated code and it prepares a build of that code which is an executable file in the form of a war or a jar. This build is then forwarded to the test server or the production server.

Continuous Testing: This is the stage where the developed software is continuously tested for bugs. For Continuous testing, automation testing tools like Selenium, TestNG, JUnit, etc are used. These tools allow QAs to test multiple code bases thoroughly in parallel to ensure that there are no flaws in the functionality. In this phase, Docker Containers can be used for simulating the test environment. Selenium does the automation testing, and the reports are generated by TestNG. This entire testing phase can be automated with the help of a Continuous Integration tool called Jenkins.

Continuous Deployment: This is the stage where the code is deployed to the production servers. It is also important to ensure that the code is correctly deployed on all the servers. Configuration management tools play an important role in executing tasks quickly and frequently. Some popular tools that are used here are Puppet, Chef, Ansible, Puppet ect. Containerization tools help in maintaining consistency across the environments where the application is developed, tested and deployed.

Continuous monitoring: This is a very crucial stage of the DevOps life cycle where you continuously monitor the performance of your application. Here vital information about the use of the software is recorded. This information is processed to recognize the proper functionality of the application. The system errors such as low memory, server not reachable, etc are resolved in this phase. The popular tools used for this are Splunk, ELK Stack, Nagios, New Relic and Sensu.

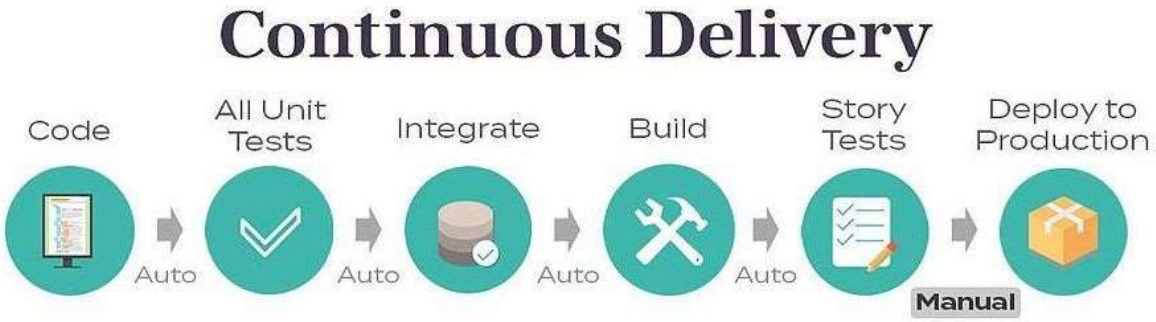
Continuous Feedback: In this phase the feedback on multiple application aspects from different stakeholders and tools is given back to a team who can consider it for next iteration of release.

Continuous Delivery: -

Continuous delivery (CD) is a software development methodology in which code modifications are automatically packaged and deployed to production. It aims to accelerate development, cut expenses, and lower risks without losing code quality.

CD is attained by designing a simple release procedure that is easily reproducible and restricts manual activities. In an ideal CD procedure, only application deployment into production requires human participation.

Once a development team has accomplished continuous integration, CD is the next step in software pipeline automation (CI). CI automates the merging and testing of code modifications, focusing particularly on unit testing. The application is deployed to a staging environment for additional testing once the code has passed evaluations. These assessments consist of integration testing, performance testing, user interface testing, and more.



Continuous delivery is an extension of continuous integration since it automatically deploys all code changes to a testing and/or production environment after the build stage.

This means that on top of automated testing, you have an automated release process and you can deploy your application any time by clicking a button.

In theory, with continuous delivery, you can decide to release daily, weekly, fortnightly, or whatever suits your business requirements. However, if you truly want to get the benefits of continuous delivery, you should deploy to production as early as possible to make sure that you release small batches that are easy to troubleshoot in case of a problem.

Eg:-

Tools:

1. Git (Version Control)
2. GitHub (Remote Repository)
3. Jenkins (Continuous Integration/Continuous Deployment)
4. Ansible (Configuration Management)

Workflow:

Step 1: Code Commit (Git)

- Developer commits code changes to local Git repository.
- Git push to GitHub remote repository.

Step 2: Continuous Integration (Jenkins)

- Jenkins polls GitHub repository for changes.
- Jenkins triggers build job:
  - Clones Git repository.
  - Runs automated tests (e.g., JUnit).
  - Builds and packages application (e.g., Maven).

Step 3: Continuous Delivery: (Ansible)

- Jenkins deploys application to staging environment using Ansible:
  - Ansible playbooks configure environment.
  - Ansible deploys application artifact.
- Jenkins runs automated tests (e.g., Selenium) in staging environment.

Step 4: Production Deployment (Jenkins + Ansible)

- Jenkins promotes application to production environment using Ansible:
  - Ansible playbooks configure environment.
  - Ansible deploys application artifact.
- Jenkins monitors application performance.

Release management in Devops: -

Release Management is the process of planning, coordinating, and controlling the release of software products to ensure that they meet the required quality, functionality, and timeline.

Release Management typically involves version control, Continuous Integration and CI/CD Pipeline, automated testing, Deployment automation and Monitoring. We use various tools and process to automate this process. This helps us release update faster and fewer mistake which is good for business and users.

Release management typically included the testing and deployment of software as well.

In simple terms release management in devops is all about making sure that when we update or release new software everything goes smoothly this is done using automation tools.

Key Activities:

1. Release Planning: Define release objectives, scope, timeline, and resources.

2. Build and Test: Create a release candidate, conduct thorough testing, and fix defects.

3. Deployment: Deploy the release to production, configure, and verify.

4. Monitoring and Feedback: Monitor release performance, gather feedback, and identify areas for improvement.

Release Management Process:

1. Release Initiation: Identify the need for a release, define objectives, and obtain approval.

2. Release Planning: Create a detailed plan, including scope, timeline, resources, and budget.

3. Build and Test: Develop, test, and stabilize the release candidate.

4. Deployment: Deploy the release to production, configure, and verify.

5. Post-Release Review: Conduct a retrospective, identify lessons learned, and document improvements.

Release Management Best Practices:

1. Automate: Automate build, test, and deployment processes where possible.

2. Continuous Integration: Integrate code changes frequently to reduce integration risks.

3. Continuous Testing: Test continuously to ensure quality and functionality.

4. Change Management: Manage changes effectively to minimize disruptions.

5. Communication: Communicate release plans, progress, and issues to stakeholders.

Release Management Tools:

1. Version Control Systems: Git, SVN, Mercurial.

2. Build Automation Tools: Jenkins, Travis CI, CircleCI.

3. Deployment Automation Tools: Ansible, Docker, Kubernetes.

4. Monitoring and Logging Tools: Nagios, Prometheus, ELK Stack.

Benefits of Effective Release Management:

1. Faster Time-to-Market: Reduce release cycles and deliver software faster.

2. Improved Quality: Ensure software meets required quality standards.

3. Reduced Risk: Minimize risks associated with software releases.

4. Increased Efficiency: Streamline release processes and reduce manual effort.

5. Better Collaboration: Improve communication and collaboration among teams.

Identifying bottlenecks

Bottleneck is a point where things slow down or get stuck.

Think of a bottle:

- The wide part of the bottle can hold a lot of liquid, but...

- The narrow neck of the bottle restricts the flow of liquid, making it harder for it to come out.

In simple words, a bottleneck is like the narrow neck of the bottle. It's a point where things can't move forward quickly or easily, causing delays and slowing everything down.

A bottleneck is a constraint or limitation in a process, system, or workflow that slows down or restricts the flow of work, resources, or information. It is a point of congestion or obstruction that hinders the efficiency and productivity of a process or system.

In the context of DevOps, bottlenecks can occur in various areas, such as:

1. Code development: Slow code reviews, inefficient coding practices, or lack of automation.

2. Testing: Inadequate testing, manual testing processes, or insufficient test coverage.

3. Deployment: Manual deployment processes, lack of automation, or slow deployment cycles.

4. Monitoring: Inadequate monitoring, lack of real-time monitoring, or insufficient alerting.

5. Communication: Poor communication between teams, stakeholders, or lack of collaboration tools.

Bottlenecks can cause:

1. Delays

2. Increased costs

3. Reduced quality

4. Decreased productivity

5. Frustration among team members

Delivery Pipeline: -

A delivery pipeline is a series of automated processes for delivering new software. It's an implementation of the continuous paradigm, where automated builds, tests, and deployments are orchestrated as one release workflow.

Delivery pipelines are continuous integration, continuous delivery, and continuous deployment (CI/CD)

Continuous integration

Developers practicing continuous integration merge their changes back to the main branch as often as possible. The developer's changes are validated by creating a build and running automated tests against the build. By doing so, you avoid integration challenges that can happen when waiting for release day to merge changes into the release branch.

Continuous integration puts a great emphasis on testing automation to check that the application is not broken whenever new commits are integrated into the main branch.

Continuous delivery

Continuous delivery is an extension of continuous integration since it automatically deploys all code changes to a testing and/or production environment after the build stage.

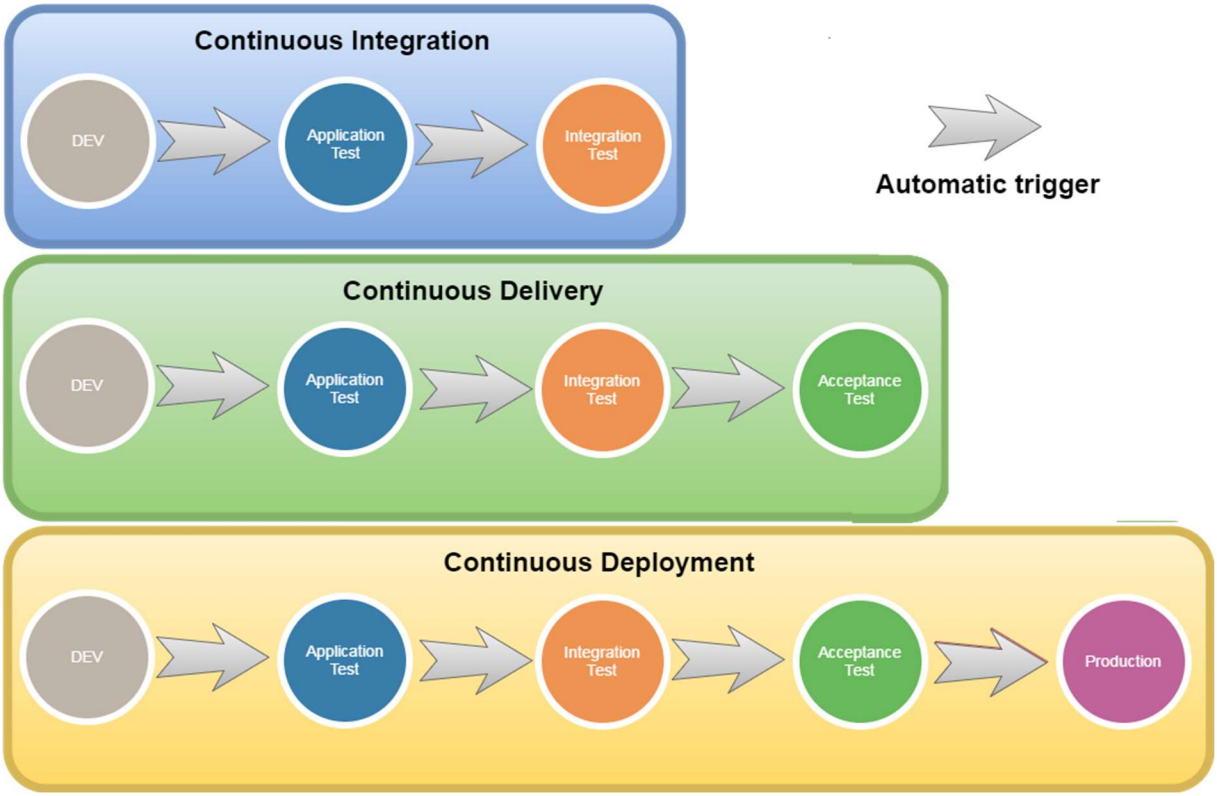
This means that on top of automated testing, you have an automated release process and you can deploy your application any time by clicking a button.

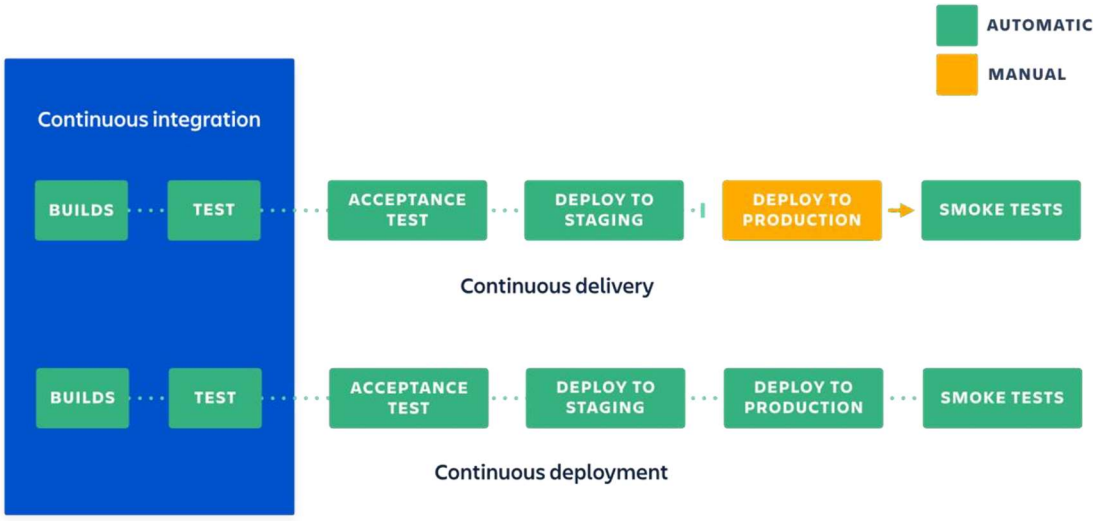
In theory, with continuous delivery, you can decide to release daily, weekly, fortnightly, or whatever suits your business requirements. However, if you truly want to get the benefits of continuous delivery, you should deploy to production as early as possible to make sure that you release small batches that are easy to troubleshoot in case of a problem.

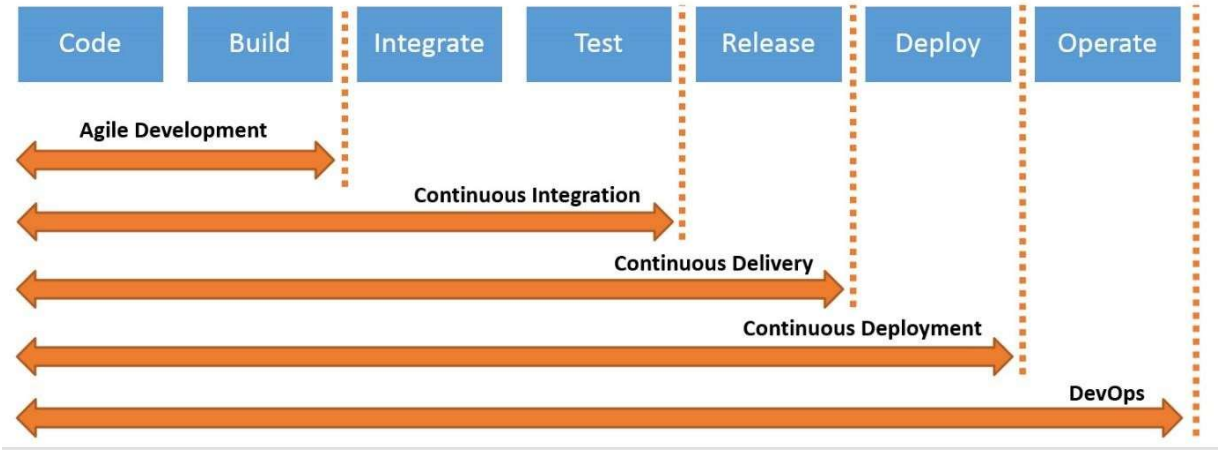
Continuous deployment

Continuous deployment goes one step further than continuous delivery. With this practice, every change that passes all stages of your production pipeline is released to your customers. There's no human intervention, and only a failed test will prevent a new change to be deployed to production.

Continuous deployment is an excellent way to accelerate the feedback loop with your customers and take pressure off the team as there isn't a "release day" anymore. Developers can focus on building software, and they see their work go live minutes after they've finished working on it.


Continuous Integration
DEV
Application Test
Integration Test
Automatic trigger
Continuous Delivery
DEV
Application Test
Integration Test
Acceptance Test
Continuous Deployment
DEV
Application Test
Integration Test
Acceptance Test
Production


AUTOMATIC
MANUAL
Continuous integration
BUILDS
TEST
ACCEPTANCE TEST
DEPLOY TO STAGING
DEPLOY TO PRODUCTION
SMOKE TESTS
Continuous delivery
BUILDS
TEST
ACCEPTANCE TEST
DEPLOY TO STAGING
DEPLOY TO PRODUCTION
SMOKE TESTS
Continuous deployment



Continuous Integration	Continuous Delivery	Continuous Deployment
Fully automated integration, build and test processes with quick feedback	CI + an automatic software release process with a manual trigger	CI + CD + fully automated deployment to production
Occurs immediately after the developer checks-in	Deliveries happen until the team feels the code is ready for shipping	All code automatically goes directly to the production stage
Uses unit tests	Uses unit and business logic tests	Can use any testing strategy
Requires a CI server to monitor the repository	Requires a strong CI foundation	Requires robust CI and a good testing culture
The team must write automated tests for each new feature or a bug fix	The test suite must cover enough of the code base	The quality of the test suite determines the quality of the release
Developers merge their changes as often as possible (at least once a day)	The team has the option of using feature flags	Feature flags are an essential part of the process

DEVOPS (D75PE1C)

UNIT-II

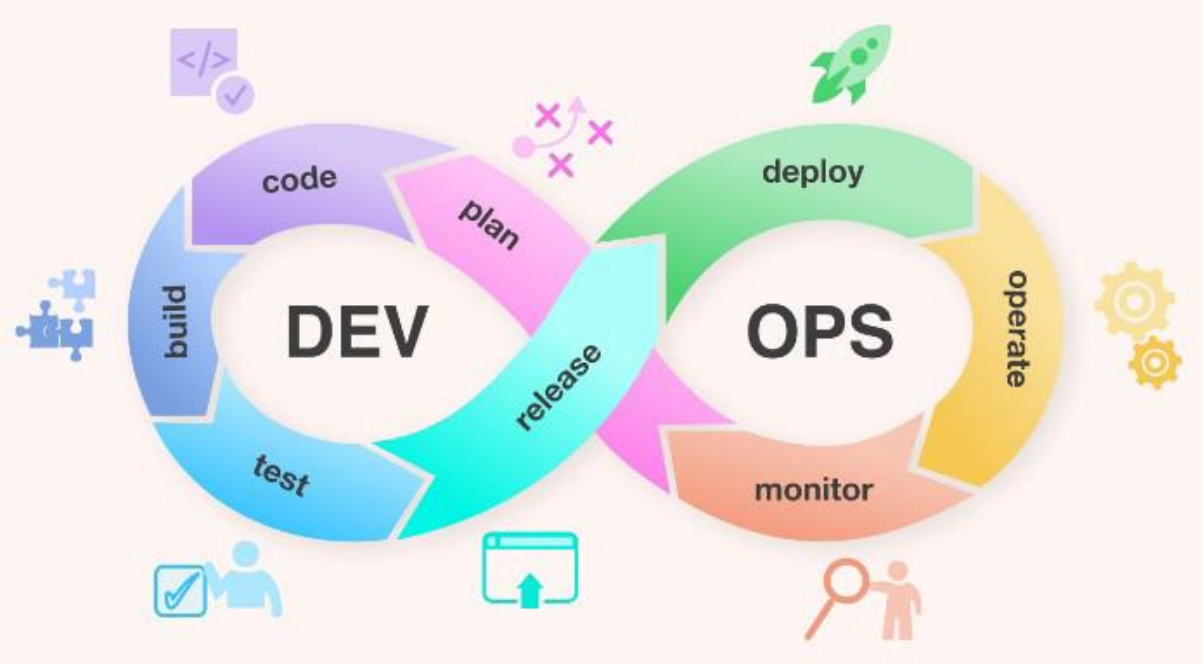
Software development models and DevOps: DevOps Lifecycle for Business Agility, DevOps, and Continuous Testing. DevOps influence on Architecture: Introducing software architecture, The monolithic scenario, Architecture rules of thumb, The separation of concerns, Handling database migrations, Micro services and the data tier, DevOps, architecture, and resilience.

DevOps Lifecycle for Business Agility

Devops define an agile relating between operation and development.

Agile- able to move quickly and easily.

DevOps lifecycle is a series of automated development processes or workflows within an iterative development lifecycle. It follows a continuous approach; hence its lifecycle is symbolized in the form of an infinity loop. This loop depicts the collaborative and iterative approach throughout the application lifecycle, consisting of tools and technology stacks for each stage. The left part deals with software development and testing. And in contrast, the right side of the infinity loop represents the deployment and operations cycle.



Let's briefly overview how the DevOps lifecycle works at every stage.

1. **Plan:** In this stage, teams identify the business requirement and collect end-user feedback. They create a project roadmap to maximize the business value and deliver the desired product during this stage.
2. **Code:** The **code development** takes place at this stage. The development teams use some tools and plugins like *Git* to streamline the development process, which help them avoid security flaws and lousy coding practices**.**

3. **Build:** In this stage, once developers finish their task, they commit the code to the shared code repository using build tools like Maven and Gradle.
4. **Test:** Once the build is ready, it is deployed to the test environment first to perform several types of testing like user acceptance test, security test, integration testing, performance testing, etc., using tools like JUnit, Selenium, etc., to ensure software quality.
5. **Release:** The build is ready to deploy on the production environment at this phase. Once the build passes all tests, the operations team schedules the releases or deploys multiple releases to production, depending on the organizational needs.
6. **Deploy:** In this stage, Infrastructure-as-Code helps build the production environment and then releases the build with the help of different tools.
7. **Operate:** The release is live now to use by customers. The operation team at this stage takes care of server configuring and provisioning using tools like Chef.
8. **Monitor:** In this stage, the DevOps pipeline is monitored based on data collected from customer behaviour, application performance, etc. Monitoring the entire environment helps teams find the bottlenecks impacting the development and operations teams' productivity.

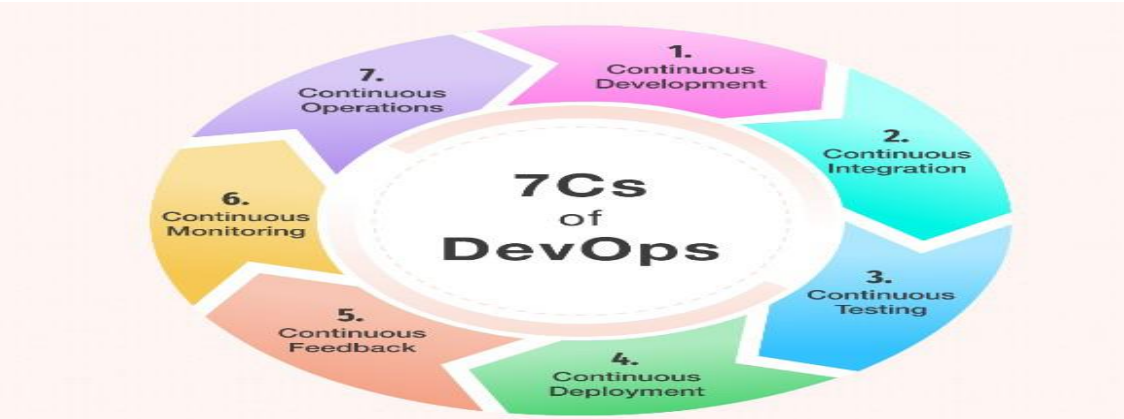
DevOps Importance and Benefits:

1. Speed
2. Time to Market
3. Reliability
4. Scale
5. Improved Collaboration
6. Contribution to Quality Assurance
7. Contribution to Services
8. Contribution to Information System Development

7 C'S OF DEVOPS LIFE CYCLE FOR BUSINESS AGILITY

Everything is continuous in DevOps - from planning to monitoring. So let's break down the entire lifecycle into seven phases where continuity is at its core. Any phase in the lifecycle can iterate throughout the project's multiple times until its finished.

The 7Cs of the DevOps lifecycle are:



1. Continuous development

This phase plays a pivotal role in delineating the vision for the entire software development cycle. It primarily focuses on project planning and coding. During this phase, project requirements are gathered and discussed with stakeholders. Moreover, the product backlog is also maintained based on customer feedback which is broken down into smaller releases and milestones for continuous software development. Once the team agrees upon the business needs, the development team starts coding for the desired requirements. It's a continuous process where developers are required to code whenever any changes occur in the project requirement or in case of any performance issues.

Tools Used: There are no specific tools for planning, but the development team requires some tools for code maintenance. GitLab, GIT, TFS, SVN, Mercurial, Jira, BitBucket, Confluence, and Subversion are a few tools used for version control. Many companies prefer agile practices for collaboration and use Scrum, Lean, and Kanban. Among all those tools, GIT and Jira are the most popular ones used for complex projects and the outstanding collaboration between teams while developing.

2. Continuous integration

Continuous integration is the most crucial phase in the entire DevOps lifecycle. In this phase, **updated code** or add-on functionalities and features are developed and integrated into existing code. Furthermore, bugs are detected and identi旳椀ed in the code during this phase at every step through **unit testing**, and then the source code is modi旳椀ed accordingly. This step makes integration a continuous approach where code is tested at **every commit**. Moreover, the tests needed are also planned in this phase.

Tools Used: Jenkin, Bamboo, GitLab CI, Buddy, TeamCity, Travis, and CircleCI are a few DevOps tools used to make the project work旳斃ow smooth and more productive. For example, Jenkin (open-source tool) is used widely to automate builds and tests. CircleCI and Buddy, on the other hand, are commercial tools.

3. Continuous testing

Quality analysts continuously test the software for bugs and issues during this stage using Docker containers. In case of a bug or an error, the code is sent back to the integration phase for modification. Automation testing also reduces the time and effort to deliver quality results. Teams use tools like *Selenium* at this stage. Moreover, continuous testing enhances the test evaluation report and minimizes the provisioning and maintenance cost of the test environments.

Tools Used: JUnit, Selenium, TestNG, and TestSigma are a few DevOps tools for continuous testing. Selenium is the most popular open-source automation testing tool that supports multiple platforms and browsers. TestSigma, on the other hand, is a unified AI-driven test

automation platform that eliminates the technical complexity of test automation through **artificial intelligence**.

4. Continuous deployment

This phase is the crucial and most active one in the DevOps lifecycle, where final code is deployed on production servers. The continuous deployment includes configuration management to make the deployment of code on servers accurate and smooth. Development teams release the code to servers and schedule the updates for servers, keeping the configurations consistent throughout the production process. Containerization tools also help in the deployment process by providing consistency across development, testing, production, and **staging environments**. This practice made the continuous delivery of new features in production possible.

Tools Used: Ansible, Puppet, and Chef are the configuration management tools that make the deployment process smooth and consistent throughout the production process. Docker and Vagrant are another DevOps tool used widely for handling the scalability of the continuous deployment process.

5. Continuous feedback

Continuous feedback came into existence to analyze and improve the application code. During this phase, customer behavior is evaluated regularly on each release to improve future releases and deployments. Businesses can either opt for a structural or unstructured approach to gather feedback. In the structural approach, feedback is collected through surveys and questionnaires. In contrast, the feedback is received through social media platforms in an unstructured approach. Overall, this phase is quintessential in making continuous delivery possible to introduce a **better version** of the application.

Tools Used: Pendo is a product analytics tool used to collect customer reviews and insights. Qentelli's TED is another tool used primarily for tracking the entire DevOps process to gather actionable insights for bugs and flaws.

6. Continuous monitoring

During this phase, the application's functionality and features are monitored continuously to detect **system errors** such as low memory, non-reachable server, etc. This process helps the IT team quickly **identify issues** related to app performance and the **root cause** behind it. If IT teams find any critical issue, the application goes through the entire DevOps cycle again to find the solution. However, the security issues can be detected and resolved automatically during this phase.

Tools Used: Nagios, Kibana, Splunk, PagerDuty, ELK Stack, New Relic, and Sensu are a few DevOps tools used to make the continuous monitoring process fast and straightforward.

7. Continuous operations

The last phase in the DevOps lifecycle is crucial for reducing the planned downtime, such as scheduled maintenance. Generally, developers are required to take the server offline to make the updates, which increases the downtime and might even cost a significant loss to the company. Eventually, continuous operation automates the process of launching the app and its

updates. It uses container management systems like Kubernetes and Docker to eliminate downtime. These container management tools help simplify the process of building, testing, and deploying the application on multiple environments. The key objective of this phase is to boost the application's uptime to ensure uninterrupted services. Through continuous operations, developers **save time** that can be used to accelerate the application's time-to-market.

Tools Used: Kubernetes and Docker Swarm are the container orchestration tools used for the high availability of the application and to make the deployment faster.

Business Agility: - Business agility means a company can easily adjust to market changes by being flexible and quick to adopt.

How Devops help to achieve business agility: -

Devops is about making teamwork between developer and operations teams smoother. Automating tasks and deploying software frequently. This helps companies reach faster to change in market and what customer wants.

CONTINUOUS TESTING IN DEVOPS: -

Continuous Testing refers to the execution of automated tests that are carried out at regular intervals every time code changes are made. These tests are conducted as a part of the software delivery pipeline to drive faster feedback on recent changes pushed to the code repository. Continuous Testing was introduced initially with the intention of reducing the time taken to provide feedback to developers.

In Devops continuous testing mean your testing your code non-stop from the moment you start writing your code until it is rady to place in a server. its about automation testing so they run automatically whenever there's a change, helping to catch bugs early and ensure the software works well.

Definition and example for each type of continuous testing tools

1.Unit Testing Tools-

Definition: Unit testing tools focus on testing individual units or components of the software, such as functions or methods.

Example: JUnit (for Java), NUnit (for .NET), PyUnit (for Python)

2. Integration Testing Tools-

Definition: Integration testing tools verify how different units or components interact with each other.

Example: Selenium (for web applications), Apache JMeter (for load and performance testing), TestComplete

3. Functional Testing Tools

Definition: Functional testing tools evaluate the software's functionality, ensuring it meets the specified requirements.

Example: Selenium, Appium (for mobile applications), TestComplete, Ranorex

4.Performance Testing Tools

Definition: Performance testing tools assess the software's performance under various conditions.

Example: Apache JMeter, Gatling, NeoLoad

5. API Testing Tools

Definition: API testing tools verify the functionality and performance of Application Programming Interfaces.

Example: Postman, SoapUI, RestAssured.

6.Security Testing Tools

Definition: Security testing tools identify vulnerabilities and weaknesses in the software.

Example: OWASP ZAP, Burp Suite, Veracode

7. Regression Testing: Ensuring Software Stability, Regression testing is a crucial software testing technique that verifies changes to an application or system do not break existing functionality. It involves re-executing functional and non-functional tests to ensure previously developed and tested software still works after a change.

The benefits of Continuous Testing in DevOps are:

 Early discovery of critical bugs

 Seamless collaboration among developers, QA and Operations team

 Helps to assess the quality of software developed at each stage

 Can be seamlessly incorporated into DevOps

 Helps drive faster test results which leads to improved code quality

 Repeated testing ensures minimal failure rate for new releases

 Faster time to market with a viable product and continuous feedback mechanism

Tools commonly used for continuous testing in Devops include: -

Junit and TestNG for unit testing

Selenium and cypress for functional testing

Jenkins and Travis CI for continuous testing, automation, Integration.

JMeter and Gatling for Performance testing

OWASP ZAP and Nessus for security testing

Challenges

Automation complexity – test across various tools and platforms can be complex and time consuming.

Tool integration – tool with other tool integration is complex in devops.

Test date management – Ensuring the availability and manageability of test data for various scenarios and environment can be difficult.

Benefits: -

-Early bug Detection

-Faster feedback loop

-improves release confidence

-cost and time saving

-collaboration between testing and operation teams.

DevOps Influence on Architecture: -

DevOps is a cultural and philosophical movement that aims to improve collaboration and communication between development and operations teams.

Architecture plays a crucial role in DevOps, as it provides the foundation for building scalable, maintainable, and efficient systems.

DevOps Principles and Architecture

- Continuous Integration: Automate testing and validation of code changes to ensure architectural integrity.

- Continuous Delivery: Automate deployment of code changes to production environments, ensuring architectural consistency.

- Continuous Monitoring: Monitor system performance and health, enabling architectural optimization.

- Infrastructure as Code: Manage infrastructure configuration through code, ensuring architectural consistency and reproducibility.

DevOps Influence on Architecture

- Microservices Architecture: DevOps enables the adoption of microservices architecture, allowing for greater flexibility and scalability.

- Cloud-Native Architecture: DevOps facilitates the adoption of cloud-native architecture, enabling greater agility and efficiency.

- Service-Oriented Architecture: DevOps promotes the adoption of service-oriented architecture, enabling greater modularity and reuse.

- Event-Driven Architecture: DevOps enables the adoption of event-driven architecture, allowing for greater scalability and flexibility.

Benefits of DevOps on Architecture

- Faster Time-to-Market: DevOps enables faster deployment of architectural changes, reducing time-to-market.

- Improved Quality: DevOps ensures continuous testing and validation, improving architectural quality.

- Increased Efficiency: DevOps automates repetitive tasks, increasing efficiency and reducing costs.

- Greater Flexibility: DevOps enables greater flexibility and adaptability in architecture, allowing for faster response to changing requirements.

Conclusion

- DevOps has a significant influence on architecture, enabling faster, more efficient, and more flexible systems.

- Architects must consider DevOps principles and practices when designing systems, ensuring alignment with business goals and objectives.

Software Architecture

- Software architecture refers to the fundamental structures and relationships between software components.

- It provides a blueprint for building and maintaining software systems.

Software Architecture Patterns

- MVC (Model-View-Controller): Separates concerns into three interconnected components.

- MVP (Model-View-Presenter): A variation of MVC that uses a presenter to manage data flow.

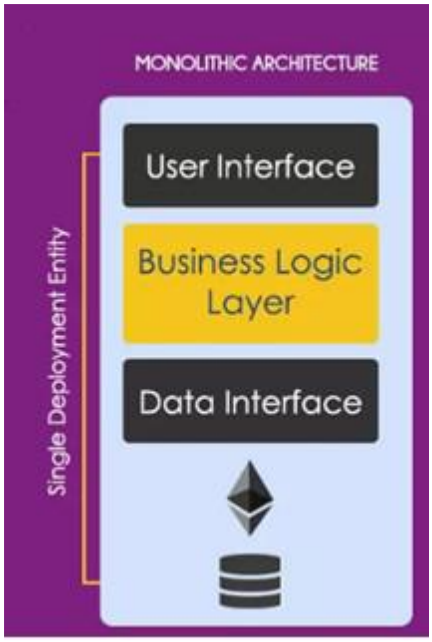
- Repository Pattern: Abstracts data access and storage.

- Service-Oriented Architecture (SOA): Structures software as a collection of services.

Monolithic Architecture: -

Monolithic architecture is a software design pattern where an application is built as a single, self-contained unit. All components (presentation, business logic, and data access) are tightly integrated and interconnected.

A monolithic architecture is a single-tiered, traditional, unified architecture for designing a software application. In context, monolithic refers to" composed all in one", "unable to change", and "too large". Monolithic applications are self-contained, complex, and tightly coupled. It is simple to develop, deploy, test, and scale horizontally.



Monolithic architecture can be used in projects which do not require a real-time response and they can withstand downtime. These kinds of projects are limited to a certain size; if the web application's size is expected to go beyond that then Microservice Architecture should be adopted. Most of the web applications that we normally use are based on monolithic architecture.

However, maintaining such a large application is difficult and its size can slow down the time to start up. Even a small update, requires redeploying the entire application, and scaling modules with conflicting resource requirements can also become challenging. Not only that it is unreliable and can have difficulty in advancing and adopting new tools and technologies

Web Applications usually consists of three parts

1. Front End client-side application is written in JavaScript and other languages.

2. Back End Server-side application which contains business logic written in java, PHP, ***Python*** or some other language.

3. Database of whole web application.

All of these parts are closely coupled and frequently communicate with each other. Hence the whole web application works as a monolith where every part is dependent on others.

Examples:

1. WordPress (blogging platform)

2. Joomla (content management system)

3. Magento (e-commerce platform)

4. Older banking systems

5. Simple web applications (e.g., to-do list apps)

Advantages of Monolithic Architecture

The code structure of monolithic architecture is small as compared to microservices architecture. Hence the Monolithic architecture-based web applications are easy to develop, easy to test, easy to deploy and easy to scale.

1. Simpler development: Easier to develop and understand, especially for small applications.

2. Faster deployment: Single deployment unit makes it quicker to roll out updates.

3. Easier testing: Less complex testing process due to fewer moving parts.

4. Better performance: Reduced overhead from inter-service communication.

5. Less overhead: No need for microservices infrastructure (e.g., service discovery).

6. Easier debugging: Single codebase makes it easier to identify issues.

Disadvantages of Monolithic Architecture

1.The complexity in Monolithic Architecture increases too much with bigger size which makes this approach limited to a certain size of projects.

2.The increase in the size of the web application increases startup time.

3. Bigger web applications become more complex and consequences in reduced code readability, difficulty in development and debugging.

4.Changes are one section of the code can cause an unanticipated impact on

the rest of the code.

5.Extensive testing and debugging are required for integrating new code.

6.Continuous Integration and continuous deployment become difficult.

7.In case a part of the web application shuts down then the rest of the web application will go down as well.

When to Use Monolithic Architecture:

1. Small applications

2. Simple web applications

3. Prototyping or proof-of-concept

4. Low-traffic websites

Architecture rules of thumb (AROT): -

Architecture Rule of Thumb (AROT) is a set of guidelines or general principles that provides a simplified approach to designing and evaluating software architectures.

AROT provides a simplified approach to architecture design, focusing on key principles and best practices.

These are not strict rules but suggestion or best practices.

These rules are based on best practices that teams can follow to achieve effective and efficient system, experience and industry standards and are intended to help architecture make informed decision when designing software system.

Basic definition: - Rule of thumb are general principal or guidelines widely used in a specific area. They are not strict rules more like practical suggestions to archive positive results in system design and implementation.

Characteristics of AROT

- Concise: Easy to remember and apply

- General: Applicable to a wide range of situations

- Flexible: Allow for adaptation to specific contexts

- Heuristic: Provide guidance rather than strict rules

The Rules or Principle

1. Modularity: Breaking down a system into interchangeable, independent modules.

2. Scalability: Ability of a system to adapt to increased load or demand.

3. Automation: Using technology to streamline repetitive processes.

4. Resilience: Ability of a system to recover from failure or disruption.

5. Infrastructure as Code (IaC): Managing infrastructure through configuration files.

6. Continuous Integration/Continuous Deployment (CI/CD): Automating testing, building, and deployment.

7. Monitoring and Logging: Tracking system performance and issues.

8. Security: Protecting systems from unauthorized access.

9. Containerization: Packaging apps with dependencies.

10. Feedback: Continuous learning and improvement.

The separation of concerns

Definition

- Concern: A set of requirements, features, or functionalities that are related to a specific aspect of the software system

- Separation of Concerns: The process of identifying, separating, and organizing concerns into distinct components or modules

Separation of Concerns means dividing entire program into various sections based on their functionality, so that each section runs independently without effecting other services.

- Separation of Concerns (SoC) is a fundamental principle in software design and architecture

- SoC aims to divide software into distinct, independent components, each responsible for a specific concern

- This separation enables easier maintenance, modification, and scalability of software systems

Benefits

1. Easier Maintenance: SoC makes it easier to modify or update individual components without affecting others

2. Improved Scalability: SoC enables adding new components or features without disrupting existing functionality

3. Reduced Complexity: SoC simplifies software design and architecture by breaking down complex systems into manageable components

4. Reusability: SoC promotes reusability of components across different parts of the system

5. Flexibility: SoC allows for easier integration of new technologies or frameworks

Real world example of SOC Implementation

1. Model View Controller (MVC) Architecture: - MVC is classic example of SOC because it divides on applications into 3 components
   Eg -web application
   Model -manages data and logic
   View – handle the presentation
   Controller – interference b/w mode and view.
2. Micro Services Application: - In a micro srvies architecture ana application is bulid as a collection of loosely coupled services where each service will perform certain task.
   Eg- e-commerce platform.
3. Client-Server Architecture – this architecture splits an application into two main component client and server that process user request.
   Eg – social media application.
4. Layered Architecture – It divides ana application into layers each layer has specific roles.
   Eg-online banking system.

Handling Database Migration

- Database migration is the process of transferring data from one database to another

- It involves changing the database schema, structure, or platform.

For example, a business might want to save resources by switching to a cloud-based database.

• Similarly, another organization could move because they find a particular database suitable for their unique business needs.

Fig: Database Migration

Types of Database Migration

1. Schema Migration: Changing the database schema (e.g., adding/removing tables, columns)

2. Data Migration: Transferring data from one database to another

3. Platform Migration: Moving from one database platform to another (e.g., MySQL to PostgreSQL)

Database Migration Challenges

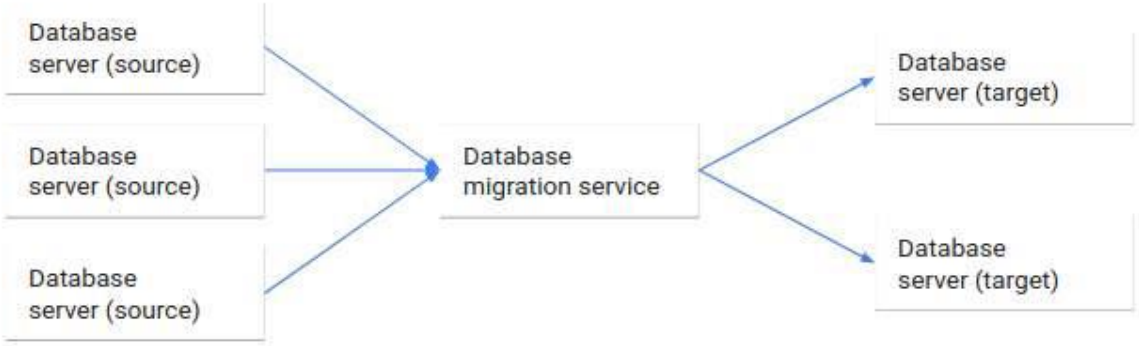
Data Base Migration is a complex process, some key challenges companies encounter while migrating their data include:

• **Data Loss**: The most common issue firms face data loss during the DB migration.

• **Data Security**: Data is a business' s most valuable asset. Therefore, its security is of utmost importance. Before the DB migration process occurs, data encryption should be a top priority.

• **Difficulty during planning**: Large companies usually have disparate databases in different departments of the companies. During the planning stage of database migration, locating these databases and planning how to convert all schemas and normalize data is a common challenge.

• **Migration strategy**: A common question asked is how to do DB migration. Companies miss out on some crucial aspects and use a database migration strategy that is not suitable for their company. Therefore, it is necessary to conduct ample research before DB migration occurs



Techniques for Database Migration

1. **Lift and Shift Migration**:
    - **Description**: This technique involves moving databases directly from one environment to another with minimal changes.
    - **Use Case**: Suitable for small databases or when moving to a cloud provider with similar database technologies.
2. **Replatforming**:
    - **Description**: Slightly modifying the database architecture or schema to take advantage of the new environment.
    - **Use Case**: Useful when migrating to a cloud-native database service or when needing to improve performance.
3. **Refactoring**:

- **Description**: Involves making significant changes to the database schema, queries, or structure to optimize for the new environment.
- **Use Case**: Ideal for large, complex applications that require performance tuning or architectural changes.

4. **Streaming Replication**:
   - **Description**: Continuous copying of data from one database to another, allowing for near-zero downtime.
   - **Use Case**: Suitable for high-availability environments where downtime needs to be minimized.
5. **Database Cloning**:
   - **Description**: Creating a duplicate of the database in the new environment, which can be modified as needed.
   - **Use Case**: Good for testing migrations without affecting the production database.

Tools for Database Migration

1. **Database Migration Tools**:
   - **AWS Database Migration Service (DMS)**: Enables migration to and from various database engines with minimal downtime.
   - **Azure Database Migration Service**: Facilitates migrations to Azure SQL Database and other services with built-in tools for assessment and migration.
   - **Google Cloud Database Migration Service**: Offers managed migration services for several database engines.
2. **Schema Migration Tools**:
   - **Liquibase**: A popular open-source tool for tracking, managing, and applying database schema changes.
   - **Flyway**: Another widely used open-source tool that allows version control for database migrations through SQL scripts.
   - **Alembic**: Used primarily with SQLAlchemy in Python applications for schema migrations.
3. **Data Migration Tools**:
   - **Talend**: An open-source data integration tool that supports data migration between various systems.
   - **Apache NiFi**: A robust data integration tool that supports the automation of data flows, including migrations.
4. **Version Control and CI/CD Integration**:

- **Git**: Storing migration scripts in a Git repository allows for version control and collaboration.
- **Jenkins**: Can automate database migrations as part of the CI/CD pipeline, triggering migrations based on changes in the application code.

5. **Monitoring and Logging Tools**:
   - **Prometheus & Grafana**: Used to monitor database performance during and after migration, helping to identify any issues.
   - **ELK Stack (Elasticsearch, Logstash, Kibana)**: Helps in logging and visualizing migration processes and outcomes

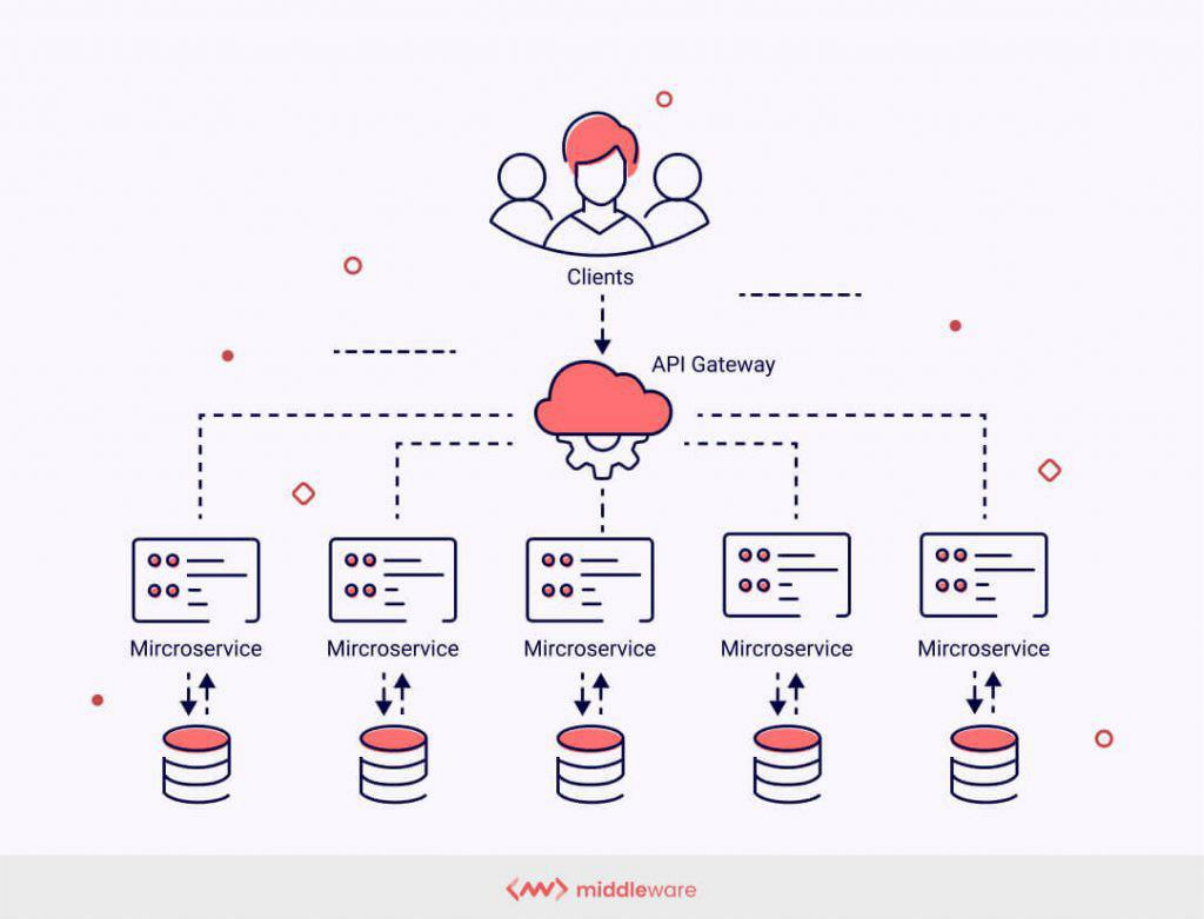
---

Micro services and the data tier

Micro services Architecture: -

Micro services are a design approach where an application is divided into small, independent services that communicate through API's

If you consider monolithic architecture entire application is placed in one ingle folder. So, changes made to one component will affect other components but where as in micro services architecture your application is places in bunch of folders each folder is responsible for a specific function so changes made in one folder will not affect other folder.



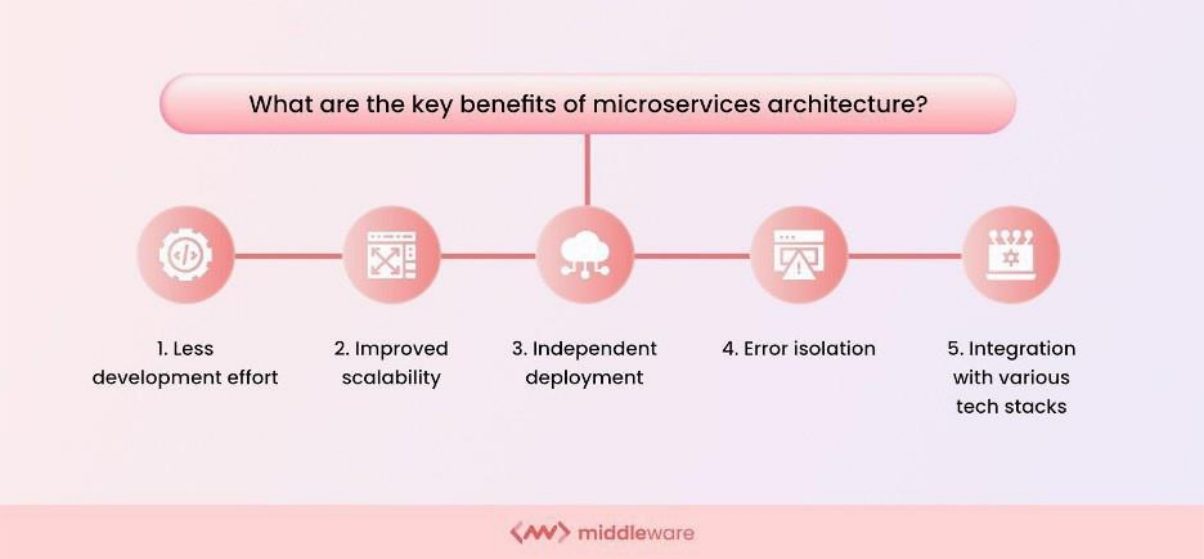
Micro services, often referred to as Micro services architecture, is an architectural approach that involves dividing large applications into smaller, functional units capable of functioning and communicating independently.

This approach arose in response to the limitations of monolithic architecture. Because monoliths are large containers holding all software components of an application, they are severely limited: inflexible, unreliable, and often develop slowly. With micro services, however, each unit is independently deployable but can communicate with each other when necessary. Developers can now achieve the scalability, simplicity, and flexibility needed to create highly sophisticated software

When to Use Microservices Architecture:

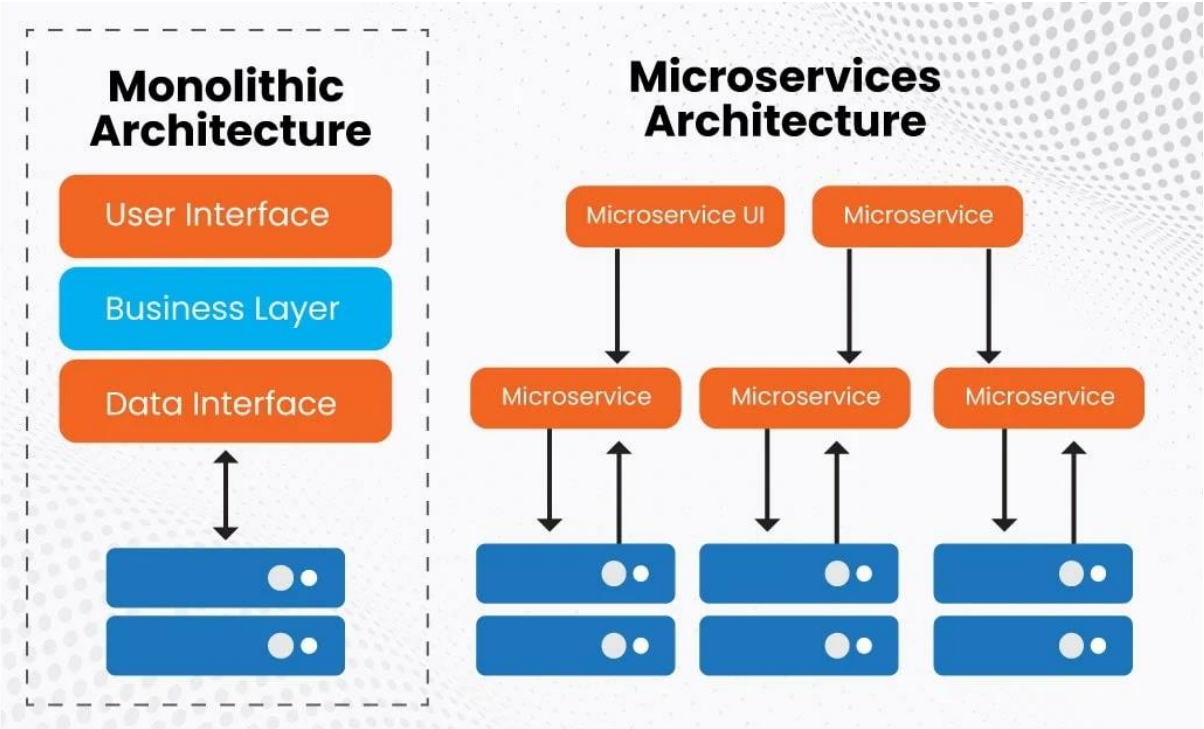
1. Large, complex applications.
2. High-traffic websites.
3. Scalability requirements.
4. Frequent updates or changes.
5. Distributed teams

The key benefits of microservices architecture



Challenges:

1. Complexity: More moving parts to manage.
2. Communication Overhead: Microservices need to communicate.
3. Data Consistency: Ensure data consistency across microservices.
4. Security: Secure communication between microservices.



	Monolithic	Microservices
Deployment	Simple and fast deployment of the entire system	Requires distinct resources, making orchestrating the deployment complicated
Scalability	It is hard to maintain and handle new changes; the whole system needs to be redeployed	Each element can be scaled independently without downtime
Agility	Not flexible and impossible to adopt new tech, languages, or frameworks	Integrate with new technologies to solve business purposes
Resiliency	One bug or issue can affect the whole system	A failure in one microservice does not affect other services
Testing	End-to-end testing	Independent components need to be tested individually
Security	Communication within a single unit makes data processing secure	Interprocess communication requires API getaways raising security issues
Development	Impossible to distribute the team's efforts due to the huge indivisible database	A team of developers can work independently on each component

Monolithic Architecture	Microservices Architecture
Consists of a singe codebase with multiple modules within according to the business functionalities.	Consists of individual services with each service being responsible for exactly one functionality.
Do not need expert domain knowledge for development.	Risky to implement without domain expertise and container knowledge.
Easier deployment.	Relatively complex deployment.
Updating the system is a tedious process which would need the entire system to be redeployed.	Only the service which is updated needs to be redeployed.
Reusing the modules from one software into other software systems is difficult.	Microservices can be easily used in development of other software.

Data tier

The data tier in DevOps refers to the layer of the application architecture that is responsible for storing, retrieving, and processing data. The data tier is typically composed of databases, data warehouses, and data processing systems that manage large amounts of structured and unstructured data.

In DevOps, the data tier is considered an important aspect of the overall application architecture and is typically managed as part of the DevOps process. This includes:

1. Data management and migration: Ensuring that data is properly managed and migrated as part of the software delivery pipeline.

2. Data backup and recovery: Implementing data backup and recovery strategies to ensure that data can be recovered in case of failures or disruptions.

3. Data security: Implementing data security measures to protect sensitive information and comply with regulations.

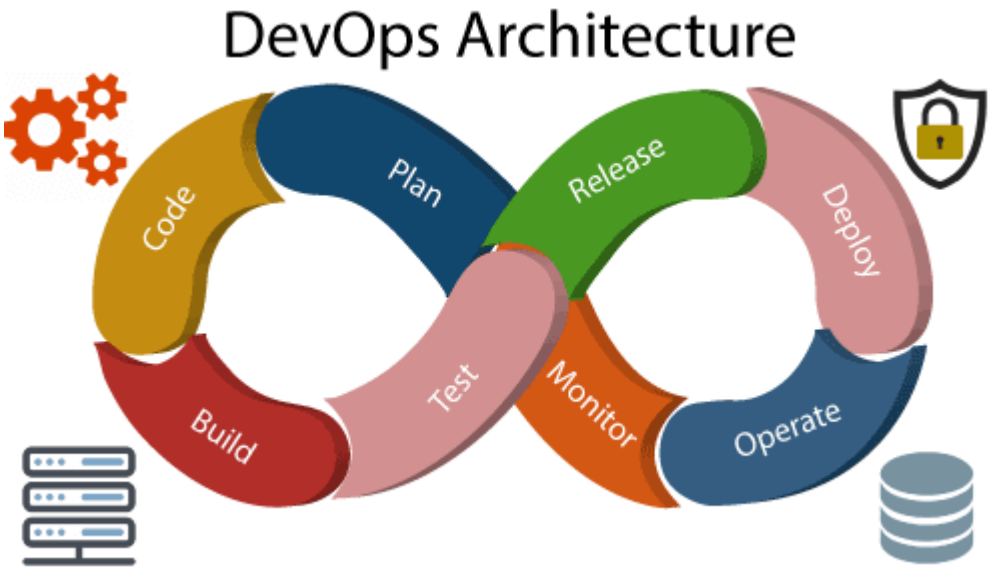
4. Data performance optimization: Optimizing data performance to ensure that applications and services perform well, even with large amounts of data.

5. Data integration: Integrating data from multiple sources to provide a unified view of data and support business decisions.

By integrating data management into the DevOps process, teams can ensure that data is properly managed and protected, and that data-driven applications and services perform well and deliver value to customers.

DevOps Architecture and Resilience: -

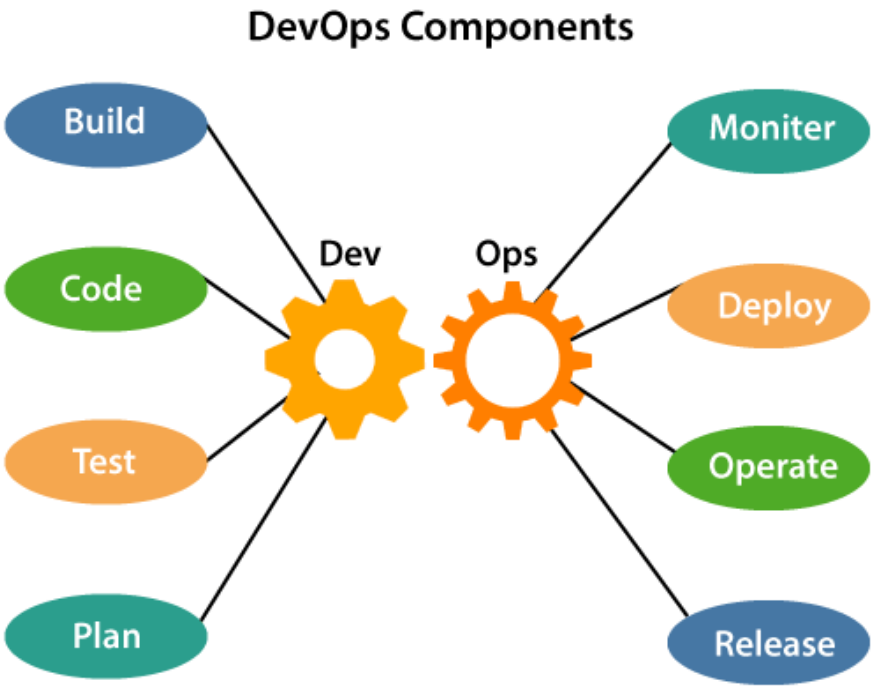
DevOps Architecture: -



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

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 work 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. **Planning**: The phase where project goals, requirements, and timelines are defined to guide the development process.
2. **Code**: The stage where developers write and maintain source code based on requirements and design specifications
3. **Test**: The process of validating code functionality and performance through automated and manual testing to ensure quality.
4. **Build**: The compilation and packaging of code into executable artifacts, preparing it for deployment.
5. **Release**: The process of preparing and making software available to users, often including final checks and documentation.
6. **Deploy**: The act of transferring the software to a production environment where it becomes accessible to end users.
7. **Operate**: Managing and maintaining the application in production to ensure it runs smoothly and meets user needs.
8. **Monitor**: Continuously tracking application performance and user interactions to identify issues and gather insights for improvement.
9. **Continuous Integration (CI)**: The practice of frequently integrating code changes into a shared repository to detect errors early.
10. **Continuous Development**: The process of continuously developing new features and improvements in short, iterative cycles.
11. **Continuous Delivery (CD)**: The practice of ensuring code changes are automatically prepared for production deployment, allowing for frequent releases.
12. **Continuous Deployment**: An extension of continuous delivery where every change that passes automated testing is automatically deployed to production.
13. **Continuous Monitoring**: The ongoing process of tracking application performance and system health in real-time to identify issues quickly.
14. **Continuous Feedback**: Gathering insights from users and system metrics throughout the development process to improve future iterations.
15. **Continuous Planning**: An iterative approach to planning that adapts based on feedback and changes in requirements throughout the development lifecycle.
16. **Continuous Testing**: The practice of executing automated tests at every stage of the development pipeline to ensure software quality and reliability.

Resilience: -

DevOps resilience refers to the ability of a DevOps system to withstand and recover from failures and disruptions. This means ensuring that the systems and processes used in DevOps are robust, scalable, and able to adapt to changing conditions. Some of the key components of DevOps resilience include:

1. Infrastructure automation: Automating infrastructure deployment, scaling, and management helps to ensure that systems are deployed consistently and are easier to manage in case of failures or disruptions.

2. Monitoring and logging: Monitoring systems, applications, and infrastructure in real-time and collecting logs can help detect and diagnose issues quickly, reducing downtime.

3. Disaster recovery: Having a well-designed disaster recovery plan and regularly testing it can help ensure that systems can quickly recover from disruptions.

4. Continuous testing: Continuously testing systems and applications can help identify and fix issues before they become critical.

5. High availability: Designing systems for high availability helps to ensure that systems remain up and running even in the event of failures or disruptions.

DEVOPS (D75PE1C)

UNIT-III

Introduction to project management: The need for source code control, the history of source code management, Roles and code, source code management system and migrations, shared authentication, Hosted Git servers, Different Git server implementations, Docker intermission, Gerrit, The pull request model, GitLab.

Introduction to project management:

Everything is code, and you need somewhere to store it. The organization's source code management system is that place. Developers and operations personnel share the same central storage for their different types of code. There are many ways of hosting the central code repository:

1.You can use a software as a service solution, such as GitHub, Bitbucket, or GitLab. This can be cost-effective and provide good availability.

2. You can use a cloud provider, such as AWS or Rackspace, to host your repositories.

3. Some types of organization simply can't let their code leave the building. For them, a private in-house setup is the best choice.

The need for source code control: -

Source code control is also known as version control is an essential part of devops.

It is a system that helps you to manage and store your files particularly the code you write.

Essential for collaboration software development, version control and bug tracking.

Source code control, also known as version control, is a crucial component of software development. It helps manage changes to codebases over time, ensuring collaboration, tracking, and backup of modifications.

Git and GitHub are primary tools used for code control.

Git: Git is a distributed version control system that allows developers to track changes and manage different versions of their source code locally in our system.

GitHub: GitHub is a cloud-based platform that hosts and manages Git repositories, providing a collaborative environment for developers to share, review, and contribute to software projects.

Terence McKenna, an American author, once said that everything is code. While one might not agree with McKenna about whether everything in the universe can be represented as code, for DevOps purposes, indeed nearly everything can be expressed in codified form, including the following:

- The applications that we build
- The infrastructure that hosts our applications

- The documentation that documents our products
- Even the hardware that runs our applications can be expressed in the form of software.

why we need source code control

-Version Tracking and Management - Tracks changes made to the codebase and Maintains a record of all versions

- Collaboration and Coordination - Multiple developers can work on the same codebase

- Change Tracking and Auditing - Records who made changes, when, and why and Provides transparency and accountability.

- Backup and Recovery - Prevents code loss due to hardware failure or disasters and Ensures business continuity.

- Code Reuse and Sharing - Facilitates reuse of code across projects and Reduces duplication of effort.

- Security and Access Control - Controls access to sensitive code and ensures only authorized changes.

- Release Management - Manages different versions of software and Tracks changes between releases

---

The history of source code management: -

The history of source code management (SCM) in DevOps dates back to the early days of software development. Early SCM systems were simple and focused on tracking changes to source code over time.

In the late 1990s and early 2000s, the open-source movement and the rise of the internet led to a proliferation of new SCM tools, including CVS (Concurrent Versions System), Subversion, and Git. These systems made it easier for developers to collaborate on projects, manage multiple versions of code, and automate the build, test, and deployment process.

As DevOps emerged as a software development methodology in the mid-2000s, SCM became an integral part of the DevOps toolchain. DevOps teams adopted Git as their SCM tool of choice, leveraging its distributed nature, branch and merge capabilities, and integration with CI/CD pipelines.

In order to understand the central need for source code control, it can be illuminating to have a brief look at the development history of source code management. This gives us an insight into the features that we ourselves might need. Some examples are as follows:

Storing historical versions of source in separate archives. This is the simplest form, and it still lives on to some degree, with many free software projects offering tar archives of older releases to download.

 Centralized source code management with check in and check out. In some systems, a developer can lock es for exclusive use. Every file is managed separately. Tools like this include RCS and SCCS.

 A centralized store where you merge before you commit. Examples include Concurrent Versions System (CVS) and Subversion. Subversion in particular is still in heavy use. Many organizations have centralized workflows, and Subversion implements such workflows well enough for them.

 A decentralized store. On each step of the evolutionary ladder, we are red more flexibility and concurrency as well as faster and more efficient workflows. We are also offered more advanced and powerful guns to shoot ourselves in the foot with, which we need to keep in mind! Currently, Git is the most popular tool in this class, but there are many other similar tools in use, such as Bazaar and Mercurial.

Version Control Systems: An Overview: -

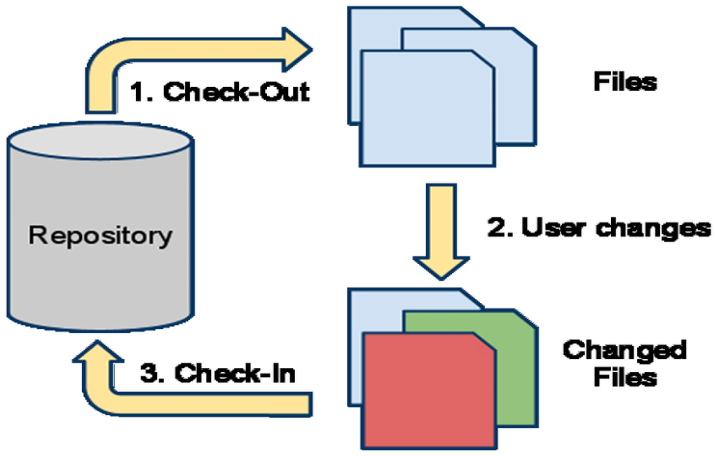
Version control systems (VCS) are essential tools in software development and collaborative projects, allowing teams to manage changes to code and documents effectively. They provide a structured way to track revisions, facilitate collaboration, and maintain the integrity of projects.

Below, we explore the primary types of version control systems, their characteristics, advantages, and disadvantages.

types of version control systems:

1. Local Version Control Systems (LVCS)

represent the simplest form of version control. In these systems, all version tracking is performed on the local machine. LVCS typically involves a basic database that records changes made to files over time. While they offer the advantage of speed and simplicity, as operations occur locally, they lack collaboration features, making it difficult for teams to share changes. Moreover, since all data is stored on the local machine, there is a significant risk of data loss if the machine fails

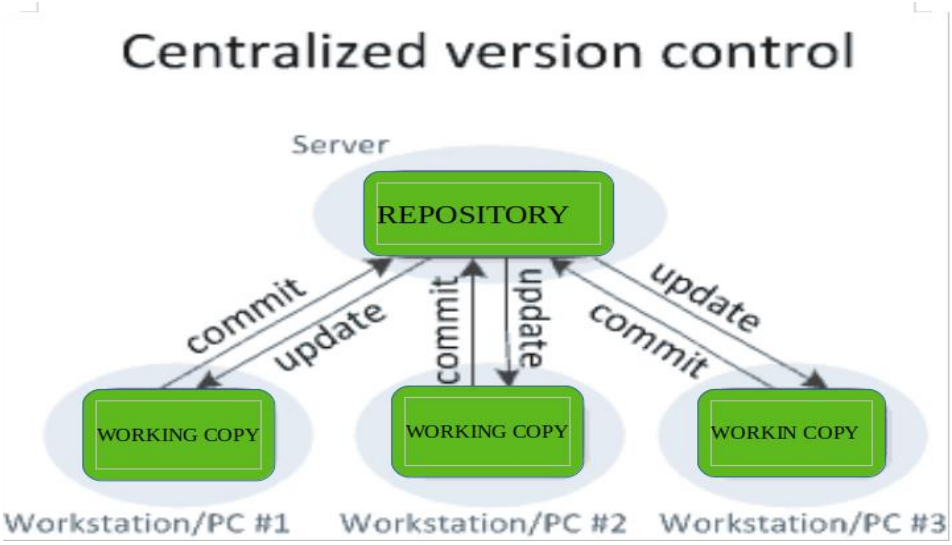


Local Version Control Systems (LVCS)

---

2. Centralized Version Control Systems (CVCS)

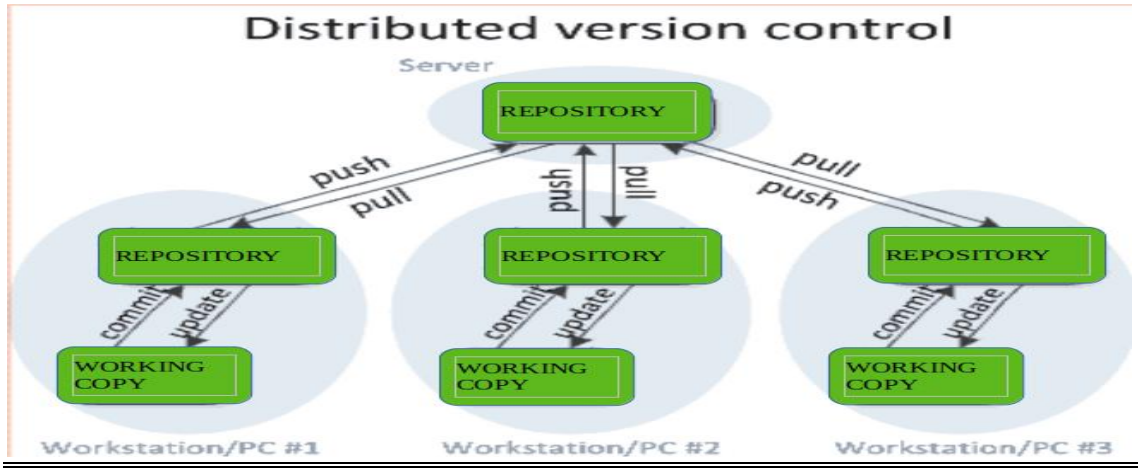
Centralized Version Control Systems (CVCS), on the other hand, maintain a single central server where all files and their histories are stored. Users check out files from this central repository, make changes, and then check them back in. This approach simplifies collaboration and allows for easier management of permissions and access control. However, it introduces a single point of failure; if the central server goes down, all users lose access to the repository. Furthermore, performance can be an issue for remote users, as they rely on server connectivity to retrieve files.



---

3. Distributed Version Control Systems (DVCS)

Distributed Version Control Systems (DVCS) take a different approach by providing every user with a complete copy of the repository, including its entire history. Systems like Git and Mercurial allow users to work offline, making changes locally before syncing with others. This method offers significant resilience, as there is no single point of failure; any user's repository can serve as a backup. DVCS also supports powerful branching and merging capabilities, enabling experimental work without disrupting the main codebase. However, the complexity of these systems can pose challenges for new users, particularly in mastering concepts like merging and conflict resolution.



Roles and code:

- In Devops source code management tools are very important because they help team members work together smoothly.

- DevOps is a collaborative approach to software development and operations

- Roles and code are essential components of DevOps

- Understanding roles and code helps ensure successful DevOps implementation

- It plays an important role in development and operations of software.

From a DevOps perspective, it's important to make the most of source code management (SCM) tools, as they serve as a central hub for various roles within a team. While it's straightforward for technical roles to use these tools, it can be more challenging for others, like project managers.

For developers, SCM is essential; it's a core part of their daily work. Operations teams also benefit from managing infrastructure descriptions using code, scripts, and other files. This includes details like network layouts and the specific software versions that need to be installed on servers.

Quality assurance (QA) teams can store their automated tests, like those created with Selenium or JUnit, directly in the source code repository, making them easily accessible.

However, there's a challenge with documenting manual processes. This issue is more about culture than technology. Many organizations use wikis (like the one that powers Wikipedia) for documentation, but there's still a lot of information stuck in Word documents on shared drives or in emails. This scattered approach makes it difficult for some roles to find and use the documentation, while others might have easier access. From a DevOps viewpoint, this is unfortunate, and steps should be taken to ensure everyone can easily access useful documentation.

One solution is to store all documentation in a wiki format within the central source code repository, depending on the capabilities of the wiki engine used.

Source code management system and migrations: -

Source code management system: -

A source code management (SCM) system is a software application that provides version control for source code. It tracks changes made to the code over time, enabling teams to revert to previous versions if necessary, and helps ensure that code can be collaborated on by multiple team members.

SCM systems typically provide features such as version tracking, branching and merging, change history, and rollback capabilities. Some popular SCM systems include Git, Subversion, Mercurial, and Microsoft Team Foundation Server.

Source code management (SCM) systems are often used to manage code migrations, which are the process of moving code from one environment to another. This is typically done as part of

a software development project, where code is moved from a development environment to a testing environment and finally to a production environment.

There are many sources code management (SCM) system and they are essentials for software development among them Git is the most popular tool.

Sum popular SCM System include – Git, Sub version, Perforce, Bazar, Mercurial.

Git has an interesting story: it was initiated by Linus Torvalds to move Linux kernel development from BitKeeper, which was a proprietary system used at the time. The license of BitKeeper changed, so it wasn't practical to use it for the kernel anymore.

Git therefore supports the fairly complicated workflow of Linux kernel development and is, at the base technical level, good enough for most organizations.

The primary benefit of Git versus older systems is that it is a distributed version control system (DVCS). There are many other distributed version control systems, but Git is the most pervasive one.

Distributed version control systems have several advantages, including, but not limited to, the following:

It is possible to use a DVCS efficiently even when not connected to a network. You can take your work with you when traveling on a train or an intercontinental flight.

Since you don't need to connect to a server for every operation, a DVCS can be faster than the alternatives in most scenarios.

You can work privately until you feel your work is ready enough to be shared.

It is possible to work with several remote logins simultaneously, which avoids a single point of failure.

Other distributed version control systems apart from Git include the following:

Bazaar: This is abbreviated as bzr. Bazaar is endorsed and supported by Canonical, which is the company behind Ubuntu. Launchpad, which is Canonical's code hosting service, supports Bazaar.

Mercurial: Notable open-source projects such as Firefox and OpenJDK use Mercurial. It originated around the same time as Git.

Git can be complex, but it makes up for this by being fast and efficient. It can be hard to understand, but that can be made easier by using frontends that support different tasks.

Migration:-

The Process of moving source code from one sources code management system(SCMs) to another.

SCM system example - Git, Sub version, Perforce, Bazar, Mercurial.

After worked with many source code management systems and experienced many transitions from one type of system to another.

Moving data from one source code management system to another can b complicated or simple depends on system

For many project history is not necessary, older system can still be accessed if needed.

Migration is necessary for adopting to changing development environment.

Reasons for Migration:

1. Changing development environment
2. Merging teams or projects
3. Upgrading to newer technology
4. Improving collaboration
5. Enhancing security

Migration Process:

1. Prepare migration plan
2. Export data from old SCMS
3. Import data into new SCMS
4. Verify data integrity
5. Update dependencies and configurations

Challenges in Migration:

1. Data loss or corruption
2. Version conflicts
3. Compatibility issues
4. User authentication and access
5. Training and support

Best Practices for Migration:

1. Test migration process
2. Validate data integrity
3. Communicate with team members
4. Document changes
5. Provide training and support

Tools for Migration:

1. Git-SVN

2. SVN-Git

3. Mercurial-Git

4. Perforce-Git

Benefits:

1. Improved collaboration

2. Enhanced security

3. Better change management

4. Increased productivity

5. Disaster recovery

- SCMS is essential for managing source code changes.

- Migration is necessary for adapting to changing development environments.

- Careful planning and execution ensure successful migration.

Shared authentication: -

Shared authentication in DevOps refers to the practice of using a common identity management system to control access to the various tools, resources, and systems used in software development and operations.

This helps to simplify the process of managing users and permissions and ensures that everyone has the necessary access to perform their jobs.

Examples of shared authentication systems include Active Directory, LDAP, and SAML-based identity providers.

It is a security mechanism that enables multiple application or servers to share the same authentication process, eliminating the need for multiple login credentials.

In most organizations, there is some form of a central server for handling authentication. An LDAP server is a fairly common choice. While it is assumed that your organization has already dealt with this core issue one way or the other and is already running an LDAP server of some sort, it is comparatively easy to set up an LDAP server for testing purposes.

One possibility is using 389 Server, named after the port commonly used for the LDAP server, together with the php LDAP admin web application for administration of the LDAP server.

Having this test LDAP server setup is useful for the purposes of this book, since we can then use the same LDAP server for all the different servers we are going to investigate.

Hosted Git servers: -

Hosted Git servers are online platforms that provide Git repository hosting services for software development teams. They are widely used in DevOps to centralize version control of source code, track changes, and collaborate on code development. Some popular hosted Git servers include GitHub, GitLab, and Bitbucket. These platforms offer features such as pull requests, code reviews, issue tracking, and continuous integration/continuous deployment (CI/CD) pipelines. By using a hosted Git server, DevOps teams can streamline their development processes and collaborate more efficiently on code projects.

Eg- GitHub, GitLab, Bitbucket, Gutbucket, Azure Devops , Aws code commit.

Git was first originally designed for text files, not for large binary files.

Like image, audio, video files, although you can store these files.

Git is a product of Linux

GitHub is the product of Microsoft

Git lab is a product of git lab company.

Many organizations can't use services hosted within another organization's walls at all.

These might be government organizations or organizations dealing with money, such as bank, insurance, and gaming organizations. The causes might be legal or, simply nervousness about letting critical code leave the organization's doors, so to speak. If you have no such qualms, it is quite reasonable to use a hosted service, such as GitHub or GitLab, that offers private accounts.

Using GitHub or GitLab is, at any rate, a convenient way to get to learn to use Git and explore its possibilities. Both vendors are easy to evaluate, given that they offer free accounts where you can get to know the services and what they offer. See if you really need all the services or if you can make do with something simpler.

Some of the features offered by both GitLab and GitHub over plain Git are as follows:

Web interfaces

A documentation facility with an inbuilt wiki

Issue trackers

Commit visualization

Different Git server implementations: -

Git is a system that allows you to work with multiple copies of your project. These copies can be on different servers like GitHub, GitLab, Bitbucket etc. to see which works better.

There are several different Git server implementations that organizations can use to host their Git repositories. Some of the most popular include:

GitHub: One of the largest Git repository hosting services, GitHub is widely used by developers for version control, collaboration, and code sharing.

GitLab: An open-source Git repository management platform that provides version control, issue tracking, code review, and more.

Bitbucket: A web-based Git repository hosting service that provides version control, issue tracking, and project management tools.

Gitea: An open-source Git server that is designed to be lightweight, fast, and easy to use.

Gogs: Another open-source Git server, Gogs is designed for small teams and organizations and provides a simple, user-friendly interface.

GitBucket: A Git server written in Scala that provides a wide range of features, including issue tracking, pull requests, and code reviews.

Organizations can choose the Git server implementation that best fits their needs, taking into account factors such as cost, scalability, and security requirement.

Docker intermission: -

Docker is an open-source project with a friendly Wale logo.

Docker is a tool or platform design to simplify the process of creating deploying and packaging and shipping out application. Docker helps us test different git server implementation like GitHub, GitLab, Git bucket, Bit bucket etc without much struggle

As docker packages your application into a container so your application will run same way on different system like laptop, desktop, virtual machine physical servers. Docker container run independently so you can test different setup without interfering with others

It's primary purpose is to automate the application development process

We need to be able to try out a couple of different Git server implementations to see which one suits our organization best.

Benefits: -

Production and standardization

Maintenance and compatibility

Rapid deployment

Faster configuration

Gerrit: -

Gerrit is a Git-based code review tool that can offer a solution in these situations. In brief, Gerrit allows you to set up rules to allow developers to review and approve changes made to a codebase by other developers. These might be senior developers reviewing changes made by inexperienced developers or the more common case, which is simply that more eyeballs on the code is good for quality in general.

Gerrit is a tool for reviewing and approving code changes before they are merged in the main codebase.

It is developed by google.

It is open-source software.

Gerrit is Java-based and uses a Java-based Git implementation under the hood.

Why Use Gerrit?

Following are certain reasons, why you should use Gerrit.

● You can easily find the error in the source code using Gerrit.

● You can work with Gerrit, if you have regular Git client; no need to install any Gerrit client.

● Gerrit can be used as an intermediate between developers and git repositories.

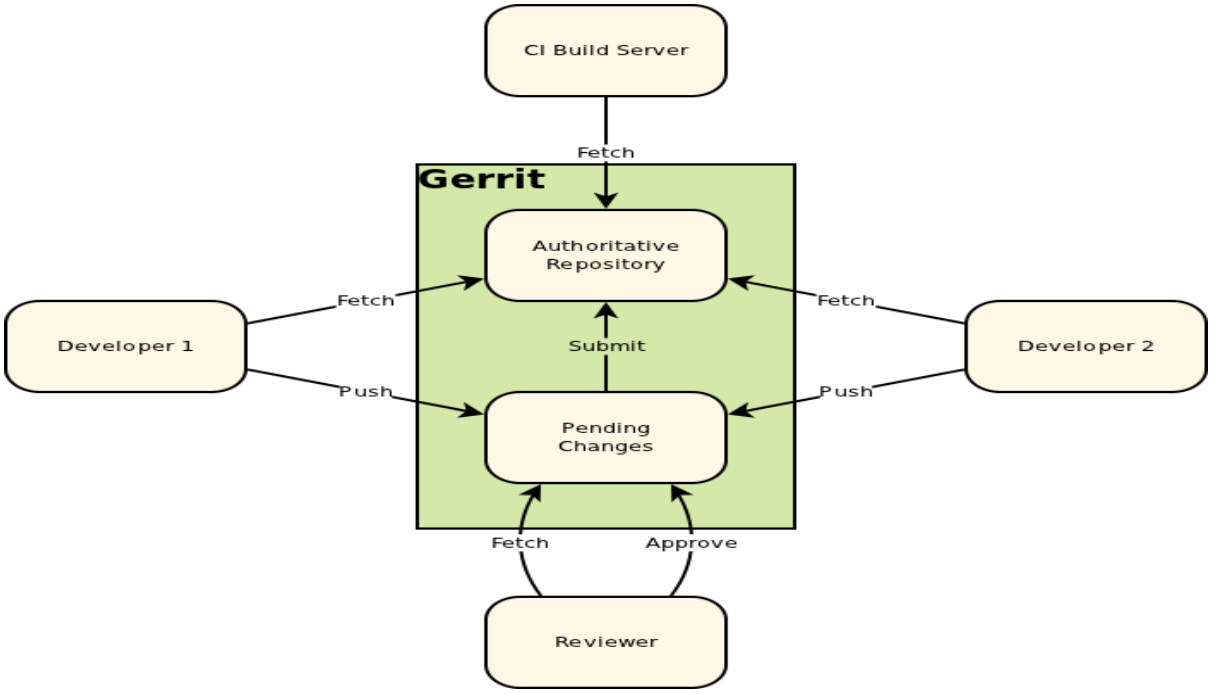
Features of Gerrit

● Gerrit is a free and an open-source Git version control system.

● The user interface of Gerrit is formed on *Google Web Toolkit*.

● It is a lightweight framework for reviewing every commit.

● Gerrit acts as a repository, which allows pushing the code and creates the review for your



When Gerrit is configured as the central source repository, all code changes are sent to Pending Changes for others to review and discuss. When enough reviewers have approved a code change, you can submit the change to the code base.

In addition to the store of Pending Changes, Gerrit captures notes and comments made about each change. This enables you to review changes at your convenience or when a conversation

about a change can't happen in person. In addition, notes and comments provide a history of each change (what was changed and why and who reviewed the change).

Like any repository hosting product, Gerrit provides a powerful access control model, which enables you to fine-tune access to your repository.

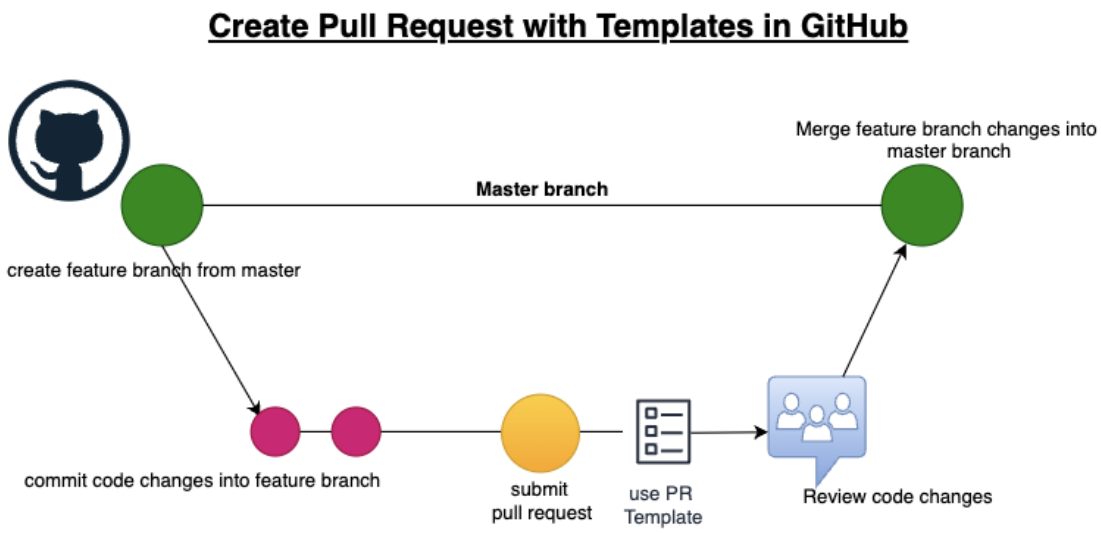
The pull request model: -

Pull request model is a way of managing code changes where developers make changes in their own copies of the Repository and hen request to merge those changes into the main repository.

There is another solution to the problem of creating workflows around code reviews: the pull request model, which has been made popular by GitHub.

In this model, pushing to repositories can be disallowed except for the repository owners. Other developers are allowed to fork the repository, though, and make changes in their fork. When they are done making changes, they can submit a pull request. The repository owners can then review the request and opt to pull the changes into the master repository.

This model has the advantage of being easy to understand, and many developers have experience in it from the many open-source projects on GitHub.



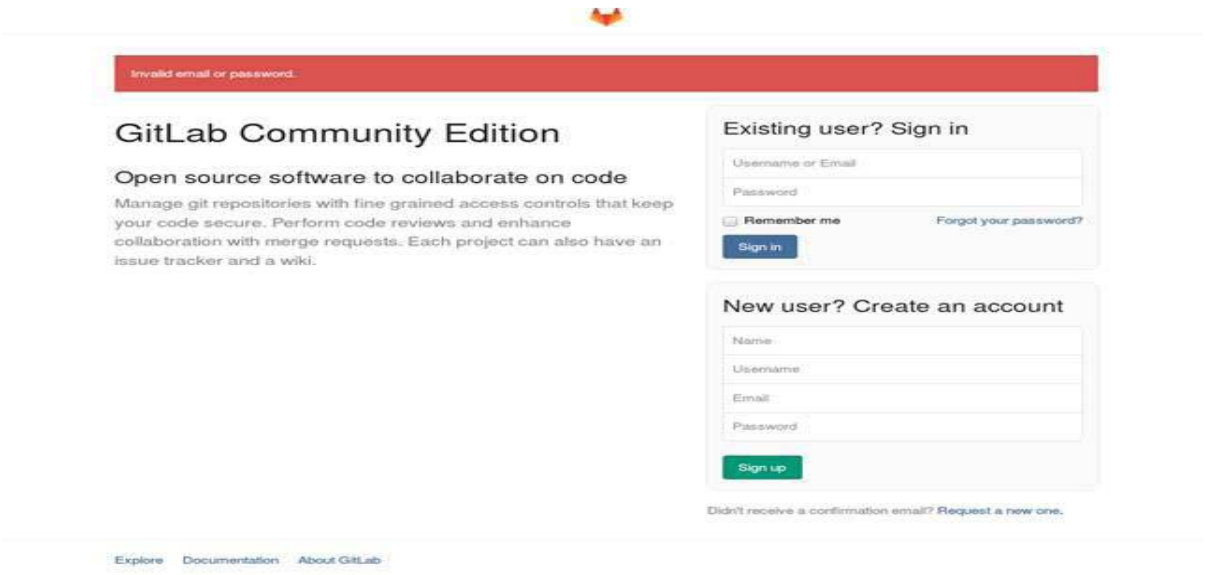
GitLab: -

GitLab is an open-source Git repository management platform that provides a wide range of features for software development teams. It is commonly used in DevOps for version control, issue tracking, code review, and continuous integration/continuous deployment (CI/CD) pipelines.

GitLab provides a centralized platform for teams to manage their Git repositories, track changes to source code, and collaborate on code development. It offers a range of tools to support code review and collaboration, including pull requests, code comments, and merge request approvals.

In addition, GitLab provides a CI/CD pipeline tool that allows teams to automate the process of building, testing, and deploying code. This helps to streamline the development process and reduce the risk of introducing bugs or other issues into the codebase.

Overall, GitLab is a comprehensive Git repository management platform that provides a wide range of tools and features for software development teams. By using GitLab, DevOps teams can improve the efficiency, transparency, and collaboration of their software development processes.



==========XXXXXXXXXXXXXX==========

DEVOPS (D75PE1C)

UNIT-IV

Integrating the system: Build systems, Jenkins build server, managing build dependencies, Jenkins plugins, and file system layout, The host server, Build slaves, Software on the host, Triggers, Job chaining and build pipelines, Build servers and infrastructure as code, Building by dependency order, Build phases, Alternative build servers, Collating quality measures.

Integrating the system:

Build systems:

A build system is a key component in DevOps, and it plays an important role in the software development and delivery process. It automates the process of compiling and packaging source code into a deployable artifact, allowing for efficient and consistent builds.

Build system is the software that can compile and execute and test code to verify code is working or not and then it will package code and create an executable file for deployment.

There are many build systems that have evolved over the history of software development. Sometimes, it might feel as if there are more build systems than there are programming languages.

Finally, of course, we have shell scripts of all kinds

For C and C++, there is Make in many different flavours

For Clojure, a language on the JVM, there is Leiningen and Boot apart from Maven

For Java, there is Maven, Gradle, and Ant

For JavaScript, there is Grunt

For Ruby, we have Rake

For Scala, there is sbt

Here is a brief list, just to get a feeling for how many there are:

Depending on the size of your organization and the type of product you are building, you might encounter any number of these tools. To deal with the complexity of the many build tools, there is also often the idea of standardizing

a particular tool. Normally, organizations standardize on a single ecosystem, such as Java and Maven or Ruby and Rake.

At any rate, we cannot assume that we will encounter only one build system within our organization's code base, nor can we assume only one programming language. If you have more than one build system to support, this basically means that you need to wrap one build system in another. Maven, for example, is good for declarative Java builds. Maven is also capable of starting other builds from within Maven builds.

Here are some of the key functions performed by a build system:

1. **Compilation:** The build system compiles source code into machine-executable formats like binaries or executable JAR files.
2. **Dependency Management:** The build system ensures that all necessary dependencies are available and integrated into the build artifact.
3. **Testing:** The build system runs automated tests to verify code functionality and catch issues early in development.
4. **Packaging:** The build system packages compiled code and dependencies into a deployable artifact, such as a Docker image or tar archive.
5. **Version Control:** The build system integrates with version control systems like Git to track code changes and manage releases.
6. **Continuous Integration:** The build system automates builds whenever changes are made, enabling fast feedback and continuous code integration.
7. **Deployment:** The build system automates the deployment of build artifacts to production environments through integration with deployment tools

---

Jenkins build server: -

-Build server is a machine that runs the build process automatically.

-Build server uses the build system (software) to compile code, run, test and create executable files one popular build server is Jenkins which is written in java.

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

- A Jenkins build server is typically set up on a dedicated machine or a virtual machine, and is used to manage the continuous integration and continuous delivery (CI/CD) pipeline for a software project. The build server is configured

with all the necessary tools, dependencies, and plugins to build, test, and deploy the project.

- Jenkins is a fork of the Hudson build server. Kohsuke Kawaguchi was Hudson's principal contributor, and in 2010, after Oracle acquired Hudson, he continued work on the Jenkins fork. Jenkins is clearly the more successful of the two strains today.

-Jenkins has special support for building Java code but is in no way limited to just building Java.

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

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

o Perform a software build using a build system like Gradle or Maven Apache

o Execute a shell script

o Archive a build result

o Running software tests

-Jenkins is especially useful for continuous integration and continuous delivery (CI/CD) pipeline it ensures that software is quality and efficient.

Managing build dependencies: -

-Managing build dependencies is an important aspect of continuous integration and continuous delivery (CI/CD) pipelines. In software development, dependencies refer to external libraries, tools, or resources that a project relies on to build, test, and deploy.

-Build dependencies in simple terms refer to other software component or files that your application needs in order to successfully build and run your application.

- Proper management of dependencies can ensure that builds are repeatable and that the build environment is consistent and up-to-date.

- Some build systems, such as the Maven tool, are nice in the way that the Maven POM file contains descriptions of which build dependencies are needed, and they

are fetched automatically by Maven if they aren't already present on the build server.

- Grunt works in a similar way for JavaScript builds.

- C and C++ builds present challenges in a different way. Many projects use GNU Autotools; among them is Autoconf, which adapts itself to the dependencies that are available on the host rather than describing which dependencies they need.

Jenkins plugins: -

-Jenkins plugins are packages of software that extend the functionality of the Jenkins automation server. Plugins allow you to integrate Jenkins with various tools, technologies, and workflows, and can be easily installed and configured through the Jenkins web interface.

-Jenkins uses plugin to add extra feature to its build server. There are many different plugins available and hey can be installed directly from the Jenkins web interface. Among other plugins we need the git plugin to pull our source code.

Some popular Jenkins plugins include:

Git Plugin: This plugin integrates Jenkins with Git version control system, allowing you to pull code changes, build and test them, and deploy the code to production.

Maven Plugin: This plugin integrates Jenkins with Apache Maven, a build automation tool commonly used in Java projects.

Amazon Web Services (AWS) Plugin: This plugin allows you to integrate Jenkins with Amazon Web Services (AWS), making it easier to run builds, tests, and deployments on AWS infrastructure.

Slack Plugin: This plugin integrates Jenkins with Slack, allowing you to receive notifications about build status, failures, and other important events in your Slack channels.

Blue Ocean Plugin: This plugin provides a new and modern user interface for Jenkins, making it easier to use and navigate.

Pipeline Plugin: This plugin provides a simple way to define and manage complex CI/CD pipelines in Jenkins

Jenkins plugins are easy to install and can be managed through the Jenkins web interface. There are hundreds of plugins available, covering a wide range of tools, technologies, and use cases, so you can easily find the plugins that best meet your needs.

By using plugins, you can greatly improve the efficiency and automation of your software development process, and make it easier to integrate Jenkins with the tools and workflows you use.

File system layout: -

In DevOps, the file system layout refers to the organization and structure of files and directories on the systems and servers used for software development and deployment. A well-designed file system layout is critical for efficient and reliable operations in a DevOps environment.

Here are some common elements of a file system layout in DevOps:

Code Repository: A central code repository, such as Git, is used to store and manage source code, configuration files, and other artifacts.

Build Artifacts: Build artifacts, such as compiled code, are stored in a designated directory for easy access and management.

Dependencies: Directories for storing dependencies, such as libraries and tools, are designated for easy management and version control.

Configuration Files: Configuration files, such as YAML or JSON files, are stored in a designated directory for easy access and management.

Log Files: Log files generated by applications, builds, and deployments are stored in a designated directory for easy access and management.

Backup and Recovery: Directories for storing backups and recovery data are designated for easy management and to ensure business continuity.

Environment-specific Directories: Directories are designated for each environment, such as development, test, and production, to ensure that the correct configuration files and artifacts are used for each environment.

By following a well-designed file system layout in a DevOps environment, you can improve the efficiency, reliability, and security of your software development and deployment processes.

The host server: -

-While a build server focuses on creating the s/w a host server focuses on running the software. Once the application is placed in the host server end user can access it.

-The build server is usually a pretty important machine for the organization. Building software is processor as well as memory and disk intensive. Builds

shouldn't take too long, so you will need a server with good specifications for the build server—with lots of disk space, processor cores, and RAM.
-The build server also has a kind of social aspect: it is here that the code of many different people and roles integrates properly for the first time. This aspect grows in importance if the servers are fast enough.
-Machines are cheaper than people, so don't let this particular machine be the area you save money on.

-In Jenkins, a host server refers to the physical or virtual machine that runs the Jenkins automation server.

-The host server is responsible for running the Jenkins process and providing resources, such as memory, storage, and CPU, for executing builds and other tasks.

-The host server can be either a standalone machine or part of a network or cloud-based infrastructure. When running Jenkins on a standalone machine, the host server is responsible for all aspects of the Jenkins installation, including setup, configuration, and maintenance.

Host Server in Jenkins: -

Install Jenkins: You can install Jenkins on a server by downloading the Jenkins WAR file, deploying it to a servlet container such as Apache Tomcat, and starting the server.

Configure Jenkins: Once Jenkins is up and running, you can access its web interface to configure and manage the build environment. You can install plugins, set up security, and configure build jobs.

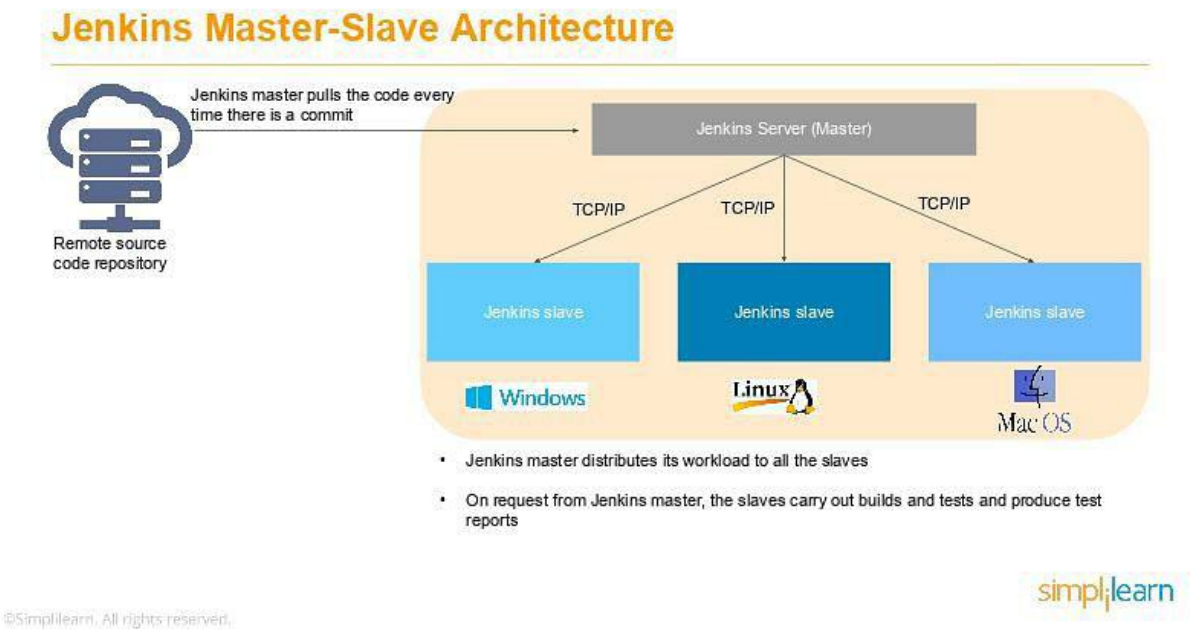
Create a Build Job: To build your project, you'll need to create a build job in Jenkins. This will define the steps involved in building your project, such as checking out the code from version control, compiling the code, running tests, and packaging the application.

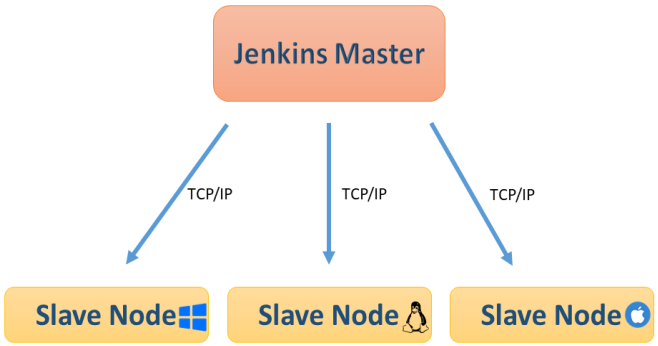
Schedule Builds: You can configure your build job to run automatically at a specific time or when certain conditions are met. You can also trigger builds manually from the web interface.

Monitor Builds: Jenkins provides a variety of tools for monitoring builds, such as build history, build console output, and build artifacts. You can use these tools to keep track of the status of your builds and to diagnose problems when they occur.

Build Slaves: -

To make build process faster in Jenkins, you can add extra server called build slaves.





A Jenkins Slave (or Agent) is a remote machine that Jenkins uses to distribute build jobs, enabling better scalability and efficiency. In Jenkins, slaves are separate systems that communicate with the master server to execute tasks (jobs) assigned by the master.

Jenkins operates in a **master-slave architecture**:

- **Master Node**: The central Jenkins instance that schedules builds, manages configurations, and provides the web interface.

- **Slave Nodes (Agents)**: Remote systems that execute build tasks delegated by the Jenkins master.

Steps to Set Up a Jenkins Slave Node

Step 1: Create a New Slave Node in Jenkins Master

1. **Login to Jenkins Master**: Use the Jenkins master web interface to log in.
2. **Navigate to Manage Jenkins**: From the Jenkins dashboard, go to **Manage Jenkins** > **Manage Nodes and Clouds**.
3. **Add New Node**: Click **New Node**, give it a name (e.g., Slave-Node-1), select the **Permanent Agent** option, and click **OK**.

Step 2: Configure Node Settings

- **Remote Root Directory**: This is the directory on the slave machine where Jenkins will store build workspaces, logs, etc.
- **Labels**: Labels help categorize slaves for job assignment. A job can be restricted to run only on slaves with specific labels.
- **Launch Method**: You need to specify how Jenkins should connect to the slave node.
    - **Launch Agent via Java Web Start**: (This is often used for manual setup and requires the slave to have a web browser).
    - **Launch Agent via SSH**: Common method, especially for Unix/Linux-based slaves.
    - **Launch via Windows Service**: For Windows-based slave nodes, this method sets up a Windows service to automatically start the Jenkins slave.

Step 3: Set Up the Slave Node

1. **Install Jenkins Agent on the Slave**:
    - For **Linux/Unix** systems, you can install the Jenkins agent by downloading the agent JAR file from the Jenkins master.

      wget http://<jenkins_master>:8080/jnlpJars/agent.jar

    - For **Windows** systems, you will usually run the agent through an installer or by launching a Windows service.
2. **Start the Agent**:

- For a **Linux/Unix** machine, you can start the Jenkins agent by running the following command:

Java -jar agent.jar -jnlpUrl http://<jenkins_master>:8080/computer/<slave_name>/slave-agent.jnlp -secret <secret_key>

- For **Windows**, you can either run the agent manually or configure it to run as a service.

Step 4: Verify Slave Node Connection

Once the slave node is configured and running, return to the **Manage Nodes and Clouds** page on the Jenkins master and verify that the slave is **online**. If the slave is not responding, you may need to troubleshoot network configurations, firewall settings, or ensure that Java is installed properly on the slave.

Software on the host: -

To run software on the host in Jenkins, you need to have the necessary dependencies and tools installed on the host machine.

The exact software you'll need will depend on the specific requirements of your project and build process.

Some common tools and software used in Jenkins include:

-Java -- Java in a machine (11,12,21)

-Git -- Git

-Build tools – Maven, Gradle or ANT

-Testing tools -JUint,TestNG, Selenium

-Database systems – MySQL, PostgreSQL, Oracle

-CI Plugin -Jenkins, GitHub Plugin, Jnekins Pipeline,Jenkins Slack.

Triggers: -

A Trigger is like a start button that begins an action automatically based on certain condition or events.

These are the most common Jenkins build triggers:

● Trigger builds remotely

● Build after other projects are built

● Build periodically

● GitHub hook trigger for GITScm polling

● Poll SCM

1. Trigger builds remotely :

If you want to trigger your project built from anywhere anytime then you should select **Trigger builds remotely** option from the build triggers.

You'll need to provide an authorization token in the form of a string so that only those who know it would be able to remotely trigger this project's builds. This provides the predefined URL to invoke this trigger remotely.

You need to predefined URL to trigger build remotely.

You should provide authentication token in the predefined URL.

2. Build after other projects are built

If your project depends on another project build then you should select **Build after other projects are built** option from the build triggers.

In this, you must specify the project(Job) names in the Projects to watch field section and select one of the following options:

1. Trigger only if the build is

Note: A build is stable if it was built successfully and no publisher reports it as unstable

2. Trigger even if the build is

Note: A build is unstable if it was built successfully and one or more publishers report it unstable

3. Trigger even if the build fails

After that, It starts watching the specified projects in the Projects to watch section. Whenever the build of the specified project completes (either is stable, unstable or failed according to your selected option) then this project build invokes.

3)Build periodically:

If you want to schedule your project build periodically then you should select the **Build periodically** option from the build triggers.

You must specify the periodical duration of the project build in the scheduler field section This field follows the syntax of cron (with minor differences). Specifically, each line consists of

5 fields separated by TAB or whitespace:

MINUTE -Minutes within the hour (0–59)

HOUR -The hour of the day (0–23)

DOM -The day of the month (1–31)

MONTH -The month (1–12)

DOW -The day of the week (0–7) where 0 and 7 are Sunday.

4)GitHub webhook trigger for GITScm polling:

A webhook is an HTTP callback, an HTTP POST that occurs when something happens through a simple event-notification via HTTP POST

GitHub webhooks in Jenkins are used to trigger the build whenever a developer commits something to the branch.

Let's see how to add build a webhook in GitHub and then add this webhook in Jenkins.

1. Go to your project repository.

2. Go to "settings" in the right corner.

3. Click on "webhooks."

4. Click "Add webhooks."

5. Write the Payload URL as: http://e330c73d.ngrok.io/github-webhook

If you are running Jenkins on localhost then writing https://localhost:8080/github-webhook/ will not work because Webhooks can only work with the public IP. So if you want to make your localhost:8080 expose public then we can use some tools.

To know more on how to add webhook in Jenkins pipeline,

visit: https://blog.knoldus.com/opsinit-adding-a-github-webhook-in-jenkins-pipeline/ In this example, we used ngrok tool to expose my local address to the public.

Poll SCM periodically polls the SCM to check whether changes were made (i.e. new commits) and builds the project if new commits were pushed since the last

build. You must schedule the polling duration in the scheduler field. Like we explained above in the Build periodically section. You can see the Build periodically section to know how to schedule.

After successfully scheduled, the scheduler polls the SCM according to your specified duration in scheduler field and builds the project if new commits were pushed since the last build. LET'S INITIATE A PARTNERSHIP.

Job chaining and build pipelines: -

Job Chaining: -

Job chaining is a technique used to link multiple jobs or tasks together automate workflow.

In Jenkins job chaining means connecting jobs so that when one job finishes successfully it automatically start another job.

Job chaining in Jenkins refers to the process of linking multiple build jobs together in a sequence. When one job completes, the next job in the sequence is automatically triggered. This allows you to create a pipeline of builds that are dependent on each other, so you can automate the entire build process.

By chaining jobs in Jenkins, you can automate the entire build process and ensure that each step is completed before the next step is started. This can help to improve the efficiency and reliability of your build process, and allow you to quickly and easily make changes to your build pipeline.

Build pipelines: -

While basic Job Chaining is useful for many tasks sometimes you need more complex workflow know as pipeline.

Build Pipeline I a series of automation process that compiles, test, Deploy software application.

A build pipeline in DevOps is a set of automated processes that compile, build, and test software, and prepare it for deployment. A build pipeline represents the end-to-end flow of code changes from development to production.

A build pipeline can be managed using a continuous integration tool such as Jenkins, TravisCI, or CircleCI. These tools automate the build process, allowing you to quickly and easily make changes to the pipeline, and ensuring that the pipeline is consistent and reliable.

In DevOps, the build pipeline is a critical component of the continuous delivery process, and is used to ensure that code changes are tested, validated, and

deployed to production as quickly and efficiently as possible. By automating the build pipeline, you can reduce the time and effort required to deploy code changes, and improve the speed and quality of your software delivery process.

Jenkins has plugin like workflow which lets you create advance pipeline using a scripting language called Groovy.

These pipelines are helpful because they allow you to see and control build process.

Build servers and Infrastructure as code: -

Build servers: -

A build server is a centralized system that automates the build, test and deployment process.

When you're developing and deploying software, one of the first things to figure out is how to take your code and deploy your working application to a production environment where people can interact with your software.

Most development teams understand the importance of version control to coordinate code commits, and build servers to compile and package their software, but Continuous Integration (CI) is a big topic.

Build servers have 3 main purposes:

● Compiling committed code from your repository many times a day

● Running automatic tests to validate code

● Creating deployable packages and handing off to a deployment tool, like Octopus Deploy

Building servers in DevOps involves several steps:

Requirements gathering: Determine the requirements for the server, such as hardware specifications, operating system, and software components needed.

Server provisioning: Choose a method for provisioning the server, such as physical installation, virtualization, or cloud computing.

Operating System installation: Install the chosen operating system on the server. Software configuration: Install and configure the necessary software components, such as web servers, databases, and middleware.

Network configuration: Set up network connectivity, such as IP addresses, hostnames, and firewall rules.

Security configuration: Configure security measures, such as user authentication, access control, and encryption.

Monitoring and maintenance: Implement monitoring and maintenance processes, such as logging, backup, and disaster recovery.

Deployment: Deploy the application to the server and test it to ensure it is functioning as expected.

Infrastructure as code: -

Infrastructure as code is a practice that manages infrastructure configure through code (manages infrastructure like servers, database, networks, using code)

Infrastructure as code (IaC) uses DevOps methodology and versioning with a descriptive model to define and deploy infrastructure, such as networks, virtual machines, load balancers, and connection topologies. Just as the same source code always generates the same binary, an IaC model generates the same environment every time it deploys.

IaC is a key DevOps practice and a component of continuous delivery. With IaC, DevOps teams can work together with a unified set of practices and tools to deliver applications and their supporting infrastructure rapidly and reliably at scale.

IaC evolved to solve the problem of *environment drift* in release pipelines. Without IaC, teams must maintain deployment environment settings individually. Over time, each environment becomes a "snowflake," a unique configuration that can't be reproduced automatically.

Inconsistency among environments can cause deployment issues. Infrastructure administration and maintenance involve manual processes that are error prone and hard to track.

IaC avoids manual configuration and enforces consistency by representing desired environment states via well-documented code in formats such as JSON. Infrastructure deployments with IaC are repeatable and prevent runtime issues caused by configuration drift or missing dependencies.

Release pipelines execute the environment descriptions and version configuration models to configure target environments. To make changes, the team edits the source, not the target.

Benefits of IAC include:

Speed: IAC enables quick and efficient provisioning and deployment of infrastructure.

Consistency: By using code to define and manage infrastructure, it is easier to ensure consistency across multiple environments.

Repeatability: IAC allows for easy replication of infrastructure components in different environments, such as development, testing, and production.

Scalability: IAC makes it easier to scale infrastructure as needed by simply modifying the code.

Version control: Infrastructure components can be versioned, allowing for rollback to previous versions if necessary.

Building by dependency order: -

Building by dependency order in DevOps is the process of ensuring that the components of a system are built and deployed in the correct sequence, based on their dependencies.

This is necessary to ensure that the system functions as intended, and that components are deployed in the right order so that they can interact correctly with each other.

When building software some parts need to be build before other because they depend on each other. You can do it by using various build tools.

Many build tools have the concept of a build tree where dependencies are built in the order required for the build to complete, since parts of the build might depend on other parts.

Make Tool

In Make-like tools, this is described explicitly; for instance, like this:

a.out : b.o c.o

b.o : b.c

c.o : c.c

So, in order to build a.out, b.o and c.o must be built first.

Maven and Gradle

In tools such as Maven, the build graph is derived from the dependencies we set for an artifact. Gradle, another Java build tool, also creates a build graph before building.

Jenkins has support for visualizing the build order for Maven builds, which is called the **reactor** in Maven parlance, in the web user interface.

Build phases: -

In devops there are build phase

Building phases are sequential steps involved in compiling testing and deploying software application.

This is very useful for a large organization, since it won't need to invent its own build standards. Other build tools are usually much more lax regarding how to implement various build phases.

Types of build phase

1)Compile Phase: - compiling source code into executable files.

2)Build Phase: - Building executable files deployable artifacts.

3)Test Phase: - Testing artifacts for functionality and operations.

4)Package Phase: - Package artifacts for deployment.

5)Deploy Phase: - Deploy artifacts to Production server.

For Example: Maven Build Phases

Maven build lifecycle goes through a set of stages, they are called build phases.

For example, the default lifecycle is made up of the following phases. validate, compile, test, package, verify, install and deploy. The build phases are executed sequentially. When we run a maven build command, we specify the phase to be executed.

Alternative build servers: -

There are several alternative build servers in DevOps, including:

Jenkins - an open-source, Java-based automation server that supports various plugins and integrations.

Travis CI - a cloud-based, open-source CI/CD platform that integrates with Github.

CircleCI - a cloud-based, continuous integration and delivery platform that supports multiple languages and integrates with several platforms.

GitLab CI/CD - an integrated CI/CD solution within GitLab that allows for complete project and pipeline management.

Bitbucket Pipelines - a CI/CD solution within Bitbucket that allows for pipeline creation and management within the code repository.

AWS CodeBuild - a fully managed build service that compiles source code, runs tests, and produces software packages that are ready to deploy.

Azure Pipelines - a CI/CD solution within Microsoft Azure that supports multiple platforms and programming languages.

Collating quality measures: -

In DevOps, collating quality measures is an important part of the continuous improvement process. The following are some common quality measures used in DevOps to evaluate the quality of software systems:

Continuous Integration (CI) metrics - metrics that track the success rate of automated builds and tests, such as build duration and test pass rate.

Continuous Deployment (CD) metrics - metrics that track the success rate of deployments, such as deployment frequency and time to deployment.

Code review metrics - metrics that track the effectiveness of code reviews, such as review completion time and code review feedback.

Performance metrics - measures of system performance in production, such as response time and resource utilization.

User experience metrics - measures of how users interact with the system, such as click-through rate and error rate.

Security metrics - measures of the security of the system, such as the number of security vulnerabilities and the frequency of security updates.

Incident response metrics - metrics that track the effectiveness of incident response, such as mean time to resolution (MTTR) and incident frequency.

By regularly collating these quality measures, DevOps teams can identify areas for improvement, track progress over time, and make informed decisions about the quality of their systems.

----------XXXXXXXXX----------

DEVOPS (D75PE1C)

UNIT-V

Testing Tools and Deployment: Various types of testing, Automation of testing Pros and cons, Selenium - Introduction, Selenium features, JavaScript testing, Testing backend integration points, Test-driven development, REPL-driven development. Deployment of the system: Deployment systems, Virtualization stacks, code execution at the client, Puppet master and agents, Ansible, Deployment tools: Chef, Salt Stack and Docker.

Testing Tools and Deployment:

As we know, **software testing** is a process of analysing an application's functionality as per the customer prerequisite. If we want to ensure that our software is bug-free or stable, we must perform the various types of software testing because testing is the only method that makes our application bug-free.

Various types of testing: -

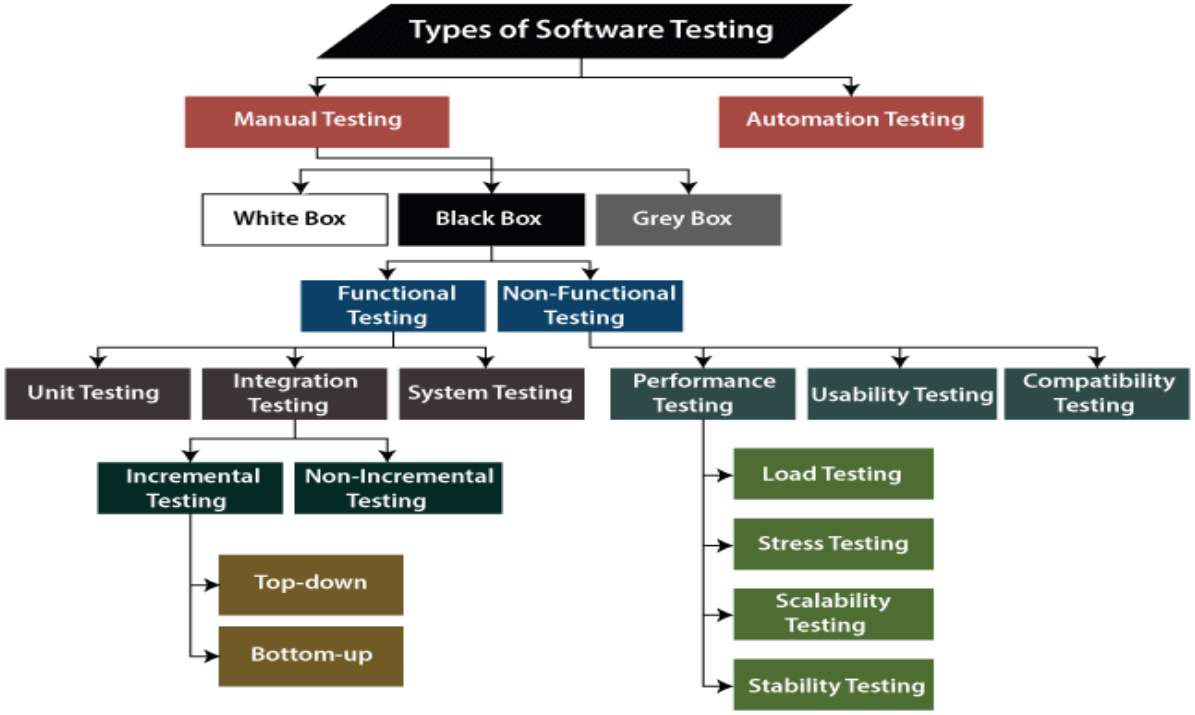
The categorization of software testing is a part of diverse testing activities, such as **test strategy, test deliverables, a defined test objective, etc**. And software testing is the execution of the software to find defects.

The purpose of having a testing type is to confirm the **AUT** (Application Under Test).

To start testing, we should have a **requirement, application-ready, necessary resources available**. To maintain accountability, we should assign a respective module to different test engineers.

The software testing mainly divided into two parts, which are as follows:

- **Manual Testing**
- **Automation Testing**



What is Manual Testing?

Testing any software or an application according to the client's needs without using any automation tool is known as **manual testing**.

In other words, we can say that it is a procedure of **verification and validation**. Manual testing is used to verify the behavior of an application or software in contradiction of requirements specification.

We do not require any precise knowledge of any testing tool to execute the manual test cases.

We can easily prepare the test document while performing manual testing on any application.

To get in-detail information about manual testing, click on the following link: https://www.javatpoint.com/manual-testing.

Classification of Manual Testing

In software testing, manual testing can be further classified into **three different types of testing**,

which are as follows:

o **White Box Testing**

o **Black Box Testing**

o **Grey Box Testing**

For our better understanding let's see them one by one:

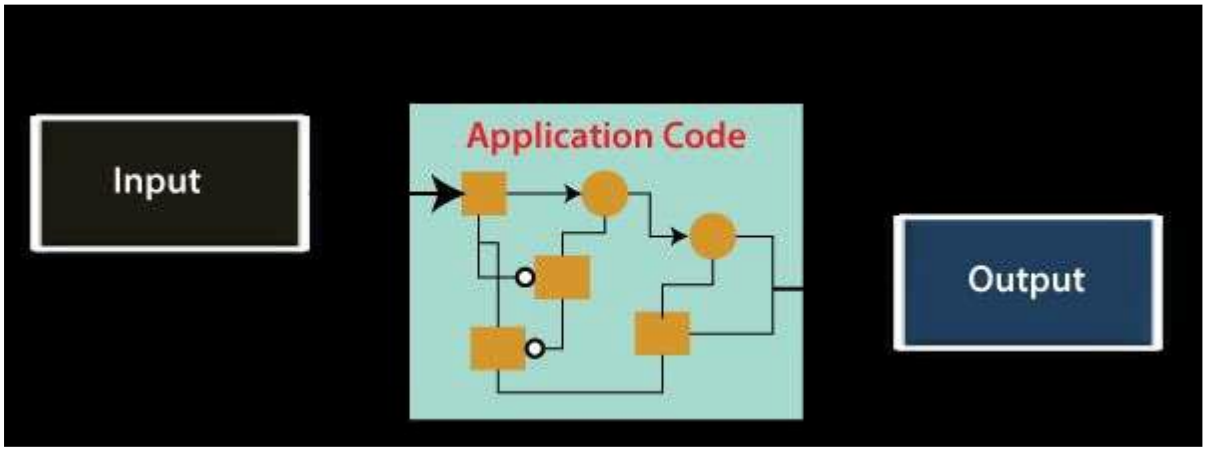
White Box Testing

In white-box testing, the developer will inspect every line of code before handing it over to the testing team or the concerned test engineers.

Subsequently, the code is noticeable for developers throughout testing; that's why this process is known as **WBT (White Box Testing)**.

In other words, we can say that the **developer** will execute the complete white-box testing for the particular software and send the specific application to the testing team.

The purpose of implementing the white box testing is to emphasize the flow of inputs and outputs over the software and enhance the security of an application.



White box testing is also known as **open box testing, glass box testing, structural testing,**

clear box testing, and transparent box testing.

Black Box Testing

Another type of manual testing is **black-box testing**. In this testing, the test engineer will analyze the software against requirements, identify the defects or bug, and sends it back to the development team.

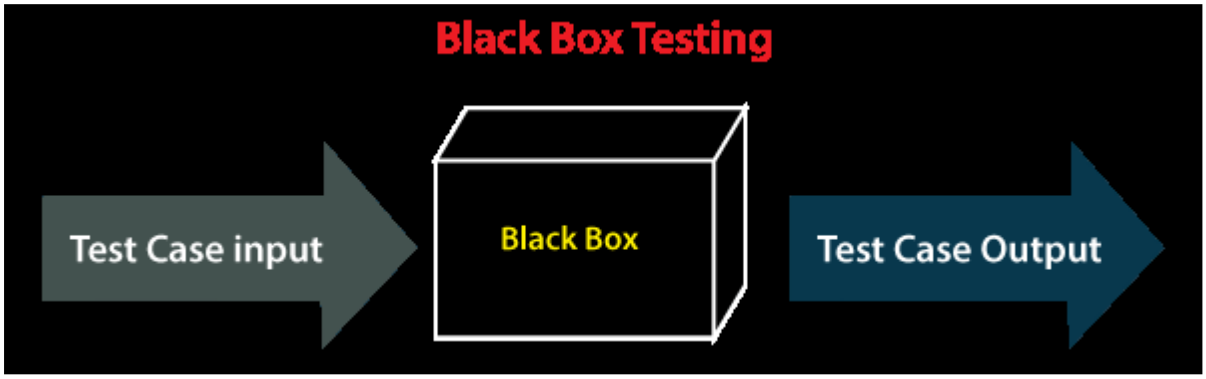
Then, the developers will fix those defects, do one round of White box testing, and send it to the testing team.

Here, fixing the bugs means the defect is resolved, and the particular feature is working according to the given requirement.

The main objective of implementing the black box testing is to specify the business needs or the customer's requirements.

In other words, we can say that black box testing is a process of checking the functionality of an application as per the customer requirement. The source code is not visible in this testing; that's

why it is known as **black-box testing**.

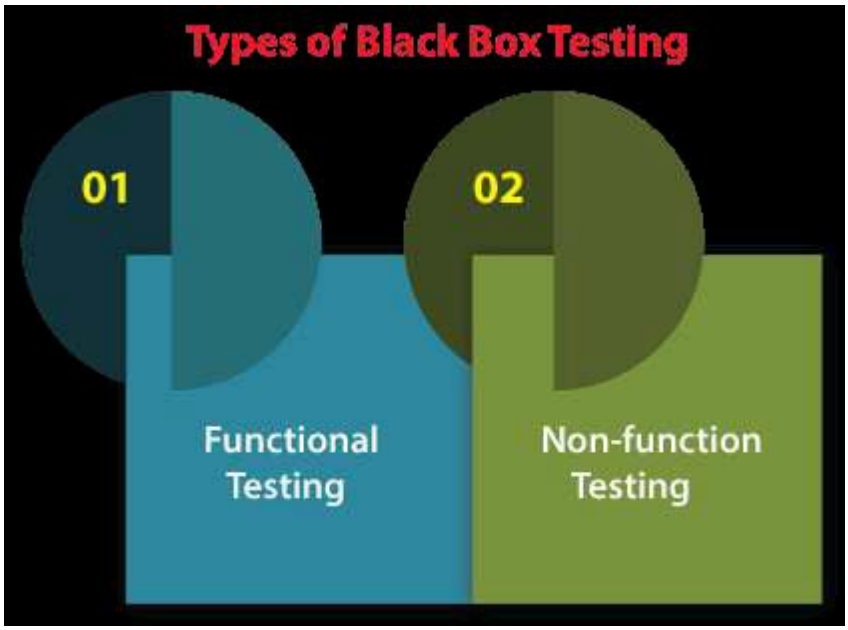


Types of Black Box Testing

Black box testing further categorizes into two parts, which are as discussed below:

o **Functional Testing**

o **Non-function Testing**



Functional Testing

The test engineer will check all the components systematically against requirement specifications is known as **functional testing**. Functional testing is also known as **Component testing**.

In functional testing, all the components are tested by giving the value, defining the output, and validating the actual output with the expected value.

Functional testing is a part of black-box testing as its emphases on application requirement rather than actual code. The test engineer has to test only the program instead of the system.

Types of Functional Testing

Just like another type of testing is divided into several parts, functional testing is also classified into various categories.

The diverse **types of Functional Testing** contain the following:

o **Unit Testing**

o **Integration Testing**

o **System Testing**

Now, Let's understand them one by one:

1. Unit Testing

Unit testing is the first level of functional testing in order to test any software. In this, the test engineer will test the module of an application independently or test all the module functionality is called **unit testing**.

The primary objective of executing the unit testing is to confirm the unit components with their performance. Here, a unit is defined as a single testable function of a software or an application. And it is verified throughout the specified application development phase.

2. Integration Testing

Once we are successfully implementing the unit testing, we will go integration testing. It is the second level of functional testing, where we test the data flow between dependent modules or interface between two features is called **integration testing**.

The purpose of executing the integration testing is to test the statement's accuracy between each module.

Types of Integration Testing

Integration testing is also further divided into the following parts:

o **Incremental Testing**

o **Non-Incremental Testing**

Incremental Integration Testing

Whenever there is a clear relationship between modules, we go for incremental integration testing. Suppose, we take two modules and analysis the data flow between them if they are working fine or not.

If these modules are working fine, then we can add one more module and test again. And we can continue with the same process to get better results.

In other words, we can say that incrementally adding up the modules and test the data flow between the modules is known as **Incremental integration testing**.

Types of Incremental Integration Testing

Incremental integration testing can further classify into two parts, which are as follows:

1. **Top-down Incremental Integration Testing**

2. **Bottom-up Incremental Integration Testing**

Let's see a brief introduction of these types of integration testing:

1. Top-down Incremental Integration Testing

In this approach, we will add the modules step by step or incrementally and test the data flow between them. We have to ensure that the modules we are adding are the **child of the earlier ones**.

2. Bottom-up Incremental Integration Testing

In the bottom-up approach, we will add the modules incrementally and check the data flow between modules. And also, ensure that the module we are adding is the **parent of the earlier ones**.

Non-Incremental Integration Testing/ Big Bang Method

Whenever the data flow is complex and very difficult to classify a parent and a child, we will go for the non-incremental integration approach. The non-incremental method is also known as **the Big Bang method**.

3. System Testing

Whenever we are done with the unit and integration testing, we can proceed with the system testing.

In system testing, the test environment is parallel to the production environment. It is also known as **end-to-end** testing.

In this type of testing, we will undergo each attribute of the software and test if the end feature works according to the business requirement. And analysis the software product as a complete system.

Non-function Testing

The next part of black-box testing is **non-functional testing**. It provides detailed information on software product performance and used technologies.

Non-functional testing will help us minimize the risk of production and related costs of the software.

Non-functional testing is a combination of **performance, load, stress, usability and, compatibility testing**.

Types of Non-functional Testing

Non-functional testing categorized into different parts of testing, which we are going to discuss further:

o **Performance Testing**

o **Usability Testing**

o **Compatibility Testing**

1. Performance Testing

In performance testing, the test engineer will test the working of an application by applying some load.

In this type of non-functional testing, the test engineer will only focus on several aspects, such as **Response time, Load, scalability, and Stability** of the software or an application.

Classification of Performance Testing

Performance testing includes the various types of testing, which are as follows:

o **Load Testing**

o **Stress Testing**

o **Scalability Testing**

o **Stability Testing**

o **Load Testing**

While executing the performance testing, we will apply some load on the particular application to check the application's performance, known as **load testing**. Here, the load could be less than or equal to the desired load.

It will help us to detect the highest operating volume of the software and bottlenecks.

o **Stress Testing**

It is used to analyze the user-friendliness and robustness of the software beyond the common functional limits.

Primarily, stress testing is used for critical software, but it can also be used for all types of software applications.

o **Scalability Testing**

To analysis, the application's performance by enhancing or reducing the load in particular balances is known as **scalability testing**.

In scalability testing, we can also check the **system, processes, or database's ability** to meet an upward need. And in this, the **Test Cases** are designed and implemented efficiently.

o **Stability Testing**

Stability testing is a procedure where we evaluate the application's performance by applying the load for a precise time.

It mainly checks the constancy problems of the application and the efficiency of a developed product. In this type of testing, we can rapidly find the system's defect even in a stressful situation.

2. Usability Testing

Another type of **non-functional testing** is **usability testing**. In usability testing, we will analyze the user-friendliness of an application and detect the bugs in the software's end-user interface.

Here, the term **user-friendliness** defines the following aspects of an application:

o The application should be easy to understand, which means that all the features must be visible to end-users.

o The application's look and feel should be good that means the application should be pleasant looking and make a feel to the end-user to use it.

3. Compatibility Testing

In compatibility testing, we will check the functionality of an application in specific hardware and software environments. Once the application is functionally stable then only, we go for **compatibility testing**.

Here, **software** means we can test the application on the different operating systems and other browsers, and **hardware** means we can test the application on different sizes.

Grey Box Testing

Another part of **manual testing** is **Grey box testing**. It is a **collaboration of black box and white box testing**.

Since, the grey box testing includes access to internal coding for designing test cases. Grey box testing is performed by a person who knows coding as well as testing.

In other words, we can say that if a single-person team done both **white box and black-box testing**, it is considered **grey box testing**.

Automation Testing

The most significant part of Software testing is Automation testing. It uses specific tools to automate manual design test cases without any human interference.

Automation testing is the best way to enhance the efficiency, productivity, and coverage of Software testing.

It is used to re-run the test scenarios, which were executed manually, quickly, and repeatedly**.**

In other words, we can say that whenever we are testing an application by using some tools is known as automation testing**.**

We will go for automation testing when various releases or several regression cycles goes on the application or software. We cannot write the test script or perform the automation testing without understanding the programming language.

Automation of testing Pros and cons: -

Pros of Automated Testing:

Automated Testing has the following advantages:

1. Automated testing improves the coverage of testing as automated execution of test cases is faster than manual execution.

2. Automated testing reduces the dependability of testing on the availability of the test engineers.

3. Automated testing provides round the clock coverage as automated tests can be run all time in 24*7 environment.

4. Automated testing takes far less resources in execution as compared to manual testing.

5. It helps to train the test engineers to increase their knowledge by producing a repository of different tests.

6. It helps in testing which is not possible without automation such as reliability testing, stress testing, load and performance testing.

7. It includes all other activities like selecting the right product build, generating the right test data and analyzing the results.

8. It acts as test data generator and produces maximum test data to cover a large number of input and expected output for result comparison.

9. Automated testing has less chances of error hence more reliable.

10. As with automated testing test engineers have free time and can focus on other creative tasks.

Cons of Automated Testing :

Automated Testing has the following disadvantages:

1. Automated testing is very much expensive than the manual testing.

2. It also becomes inconvenient and burdensome as to decide who would automate and who would train.

3. It has limited to some organisations as many organisations not prefer test automation.

4. Automated testing would also require additionally trained and skilled people.

5. Automated testing only removes the mechanical execution of testing process, but creation of test cases still required testing professionals.

Selenium

Introduction

Selenium is one of the most widely used open source Web UI (User Interface) automation testing suite.It was originally developed by Jason Huggins in 2004 as an internal tool at Thought Works.

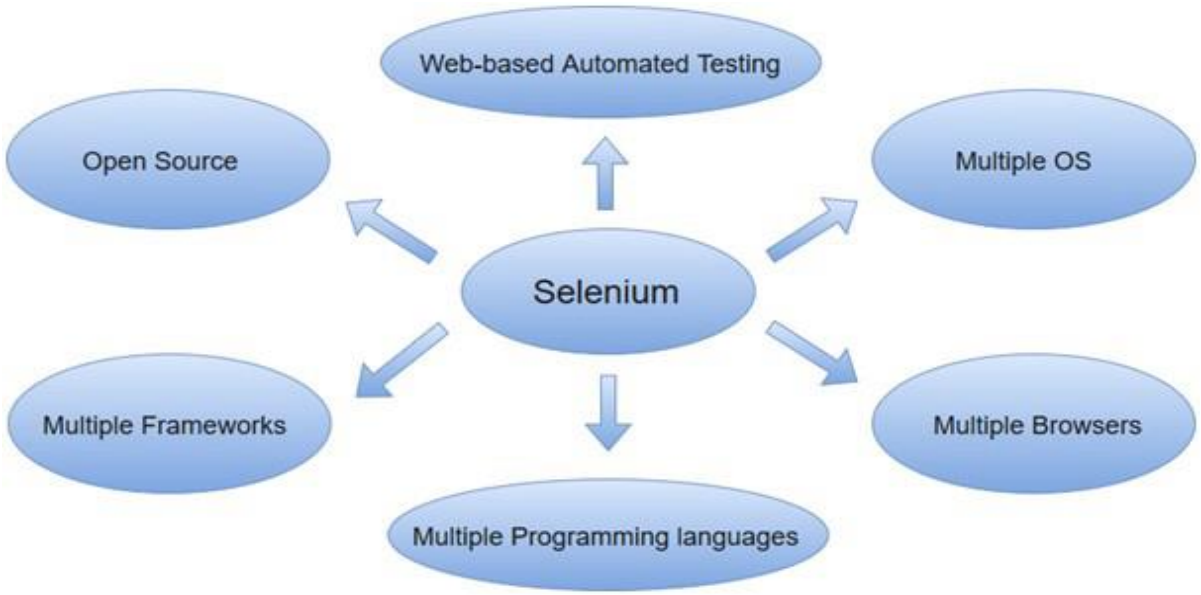
Selenium supports automation across different browsers, platforms and programming languages.

Selenium can be easily deployed on platforms such as Windows, Linux, Solaris and Macintosh. Moreover, it supports OS (Operating System) for mobile applications like iOS, windows mobile and android.

Selenium supports a variety of programming languages through the use of drivers specific to each Language.

Languages supported by Selenium include C#, Java, Perl, PHP, Python and Ruby.

Currently, Selenium Web driver is most popular with Java and C#. Selenium test scripts can be coded in any of the supported programming languages and can be run directly in most modern web browsers. Browsers supported by Selenium include Internet Explorer, Mozilla Firefox, Google Chrome and Safari.



Selenium can be used to automate functional tests and can be integrated with automation test tools such as **Maven**, **Jenkins**, **& Docker** to achieve continuous testing. It can also be integrated with tools such as **TestNG**, & **JUnit** for managing test cases and generating reports.

Selenium Features

o Selenium is an open source and portable Web testing Framework.

o Selenium IDE provides a playback and record feature for authoring tests without the need to learn a test scripting language.

o It can be considered as the leading cloud-based testing platform which helps testers to record their actions and export them as a reusable script with a simple-to-understand and easy-to-use interface.

o Selenium supports various operating systems, browsers and programming languages.

Following is the list:

o Programming Languages: C#, Java, Python, PHP, Ruby, Perl, and JavaScript

o Operating Systems: Android, iOS, Windows, Linux, Mac, Solaris.

o Browsers: Google Chrome, Mozilla Firefox, Internet Explorer, Edge, Opera,

Safari, etc.

o It also supports parallel test execution which reduces time and increases the efficiency of tests. Selenium can be integrated with frameworks like Ant and Maven for source code compilation.

o Selenium can also be integrated with testing frameworks like TestNG for application testing and generating rseports.

o Selenium requires fewer resources as compared to other automation test tools.

o WebDriver API has been indulged in selenium whichis one of the most important modifications done to selenium.

o Selenium web driver does not require server installation, test scripts interact directly with the browser.

o Selenium commands are categorized in terms of different classes which make it easier to understand and implement.

JavaScript testing

JavaScript testing is a crucial part of the software development process that helps ensure the quality and reliability of code. The following are the key components of JavaScript testing:

Test frameworks: A test framework provides a structure for writing and organizing tests. Some popular JavaScript test frameworks include Jest, Mocha, and Jasmine.

Assertion libraries: An assertion library provides a set of functions that allow developers to write assertions about the expected behavior of the code. For example, an assertion might check that a certain function returns the expected result.

Test suites: A test suite is a collection of related tests that are grouped together. The purpose of a test suite is to test a specific aspect of the code in isolation.

Test cases: A test case is a single test that verifies a specific aspect of the code. For example, a test case might check that a function behaves correctly when given a certain input.

Test runners: A test runner is a tool that runs the tests and provides feedback on the results. Test runners typically provide a report on which tests passed and which tests failed.

Continuous Integration (CI): CI is a software development practice where developers integrate code into a shared repository frequently. By using CI, developers can catch issues early and avoid integration problems.

The goal of JavaScript testing is to catch bugs and defects early in the development cycle, before they become bigger problems and impact the quality of the software. Testing also helps to ensure that the code behaves as expected, even when changes are made in the future.

There are different types of tests that can be performed in JavaScript, including unit tests, integration tests, and end-to-end tests. The choice of which tests to write depends on the specific requirements and goals of the project.

JavaScript can be used for testing web applications by writing test scripts that interact with the application's user interface and simulate user actions. Some of the popular JavaScript testing frameworks are:

Since there usually are web UI implementations of nearly every product these days, the JavaScript testing frameworks deserve special mention:

Karma is a test runner for unit tests in the JavaScript language

Jasmine is a Cucumber-like behavior testing framework

Protractor is used for AngularJS

Jasmine: Jasmine is a behavior-driven testing framework that is easy to set up and use. It provides a clean syntax for writing tests and supports asynchronous testing.

Mocha: Mocha is a testing framework that supports both synchronous and asynchronous testing. It provides various features like custom reporters, parallel testing, and support for multiple assertion libraries.

Jest: Jest is a JavaScript testing framework that is developed by Facebook. It provides built-in support for features like mocking, snapshot testing, and code coverage reporting.

Cypress: Cypress is a JavaScript-based end-to-end testing framework that allows developers to write tests that run in the browser. It provides features like real-time reloading, automatic waiting, and debugging tools.

Protractor: Protractor is a testing framework that is specifically designed for testing AngularJS applications. It uses Selenium WebDriver to automate browser interactions and provides built-in support for Angular-specific features like directives and services.

JavaScript testing frameworks like these allow developers to write automated tests for web applications to ensure that they are functioning correctly. These tests can help catch issues early in the development process and save time and resources in the long run.

Testing backend integration points: -

The term backend generally refers to **server-side deployment.** Here the process is entirely happening in the backend which is not shown to the user only the expected results will be shown to the user. In every web application, there will be a backend language to accomplish the task.

For Example, while uploading the details of the students in the database, the database will store all the details. When there is a need to display the details of the students, it will simply fetch all the details and display them. Here, it will show only the result, not the process and how it fetches the details.

Automated testing of backend functionality such as SOAP and REST endpoints is normally quite cost effective. Backend interfaces tend to be fairly stable, so the

corresponding tests will also require less maintenance effort than GUI tests, for instance.

SOAP means “Simple Object Acess Protocol”

REST means “Representational State Transfer”

The tests can also be fairly easy to write with tools such as soapUI, which can be used to write and execute tests. These tests can also be run from the command line and with Maven, which is great for Continuous Integration on a build server.

The soapUI is a good example of a tool that appeals to several different roles. Testers who build test cases get a fairly well-structured environment for writing tests and running them interactively. Tests can be built incrementally.

Developers can integrate test cases in their builds without necessarily using the GUI. There are Maven plugins and command-line runners.

The command line and Maven integration are useful for people maintaining the build server too.

Furthermore, the licensing is open source with some added features in a separate, proprietary version. The open source nature makes the builds more reliable. It is very stress-inducing when a build fails because a license has unexpectedly reached its end or a floating license has run out.

The soapUI tool has its share of flaws, but in general, it is flexible and works well. Here's what the user interface looks like:

The tests can also be fairly easy to write with tools such as soapUI, which can be used to write and execute tests. These tests can also be run from the command line and with Maven, which is great for Continuous Integration on a build server.

The soapUI is a good example of a tool that appeals to several different roles. Testers who build test cases get a fairly well-structured environment for writing tests and running them interactively. Tests can be built incrementally.

Developers can integrate test cases in their builds without necessarily using the GUI. There are Maven plugins and command-line runners.

The command line and Maven integration are useful for people maintaining the build server too.

Furthermore, the licensing is open source with some added features in a separate, proprietary version. The open source nature makes the builds more reliable. It is very stress-inducing when a build fails because a license has unexpectedly reached its end or a floating license has run out.

The soapUI tool has its share of flaws, but in general, it is flexible and works well. Here's what the user interface looks like:

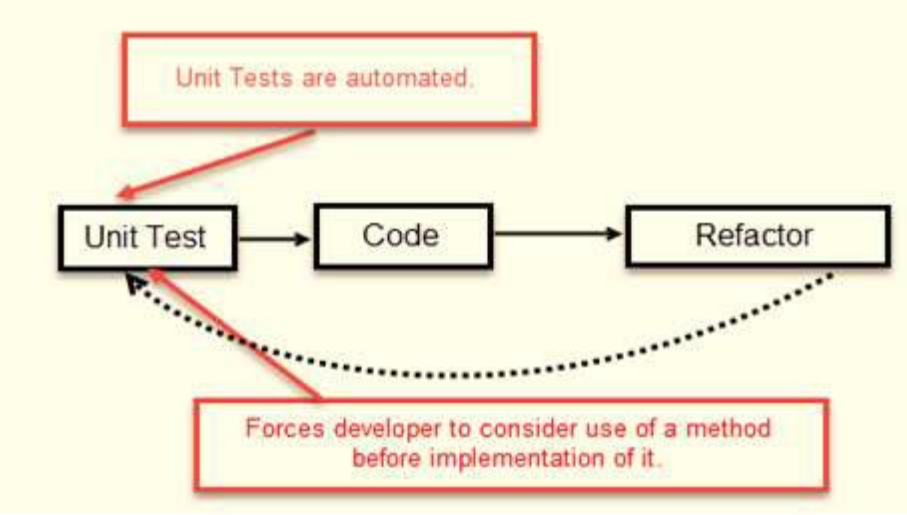
The soapUI user interface is fairly straightforward. There is a tree view listing test cases on the left. It is possible to select single tests or entire test suites and run them. The results are presented in the area on the right.

Test-driven development: -

Test-driven development (TDD) is a software development methodology that emphasizes writing automated tests before implementing the actual code.

In simple terms, testcases for each functionality are created and tested first and if the test fails then thenew code is written in order to pass the test and making code simple and bug-free.

Test-Driven Development starts with designing and developing tests for every small functionality of an application. TDD framework instructs developers to write new code only if an automated test has failed. This avoids duplication of code. The TDD full form is Test-driven development.



TDD is usually described as a sequence of events, as follows:

Implement the test: As the name implies, you start out by writing the test and write the code afterwards. One way to see it is that you implement the interface specifications of the code to be developed and then progress by writing the code. To be able to write the test, the developer must find all relevant requirement specifications, use cases, and user stories.

The shift in focus from coding to understanding the requirements can be beneficial for implementing them correctly.

Verify that the new test fails: The newly added test should fail because there is nothing to implement the behavior properly yet, only the stubs and interfaces needed to write the test. Run the test and verify that it fails.

Write code that implements the tested feature: The code we write doesn't yet have to be particularly elegant or efficient. Initially, we just want to make the new test pass.

Verify that the new test passes together with the old tests: When the new test passes, we know that we have implemented the new feature correctly. Since the old tests also pass, we haven't broken existing functionality.

Refactor the code: The word "refactor" has mathematical roots. In programming, it means cleaning up the code and, among other things, making it easier to understand and maintain. We need to refactor since we cheated a bit earlier in the development.

TDD is a style of development that fits well with DevOps, but it's not necessarily the only one. The primary benefit is that you get good test suites that can be used in Continuous Integration tests.

REPL-driven development: -

REPL-driven development, also known as Read-Eval-Print Loop-driven development

This style of development is very common when working with interpreted languages Such as Lisp, Python, Ruby and JavaScript.

REPL is a programming environment that allows developers to interactively write and test code snippets, where they can input expressions, have them evaluated and the results printed back.

In REPL-driven development, developers typically write code incrementally, testing and verifying its behavior in the REPL environment. They can experiment with different code snippets, evaluate them instantly, and receive immediate feedback.

Here are some key characteristics and benefits of REPL-driven development:

1.**Iterative development**: Developers can quickly write and test small pieces of code, getting immediate feedback on their changes. This iterative approach allows for faster development cycles.

2. **Exploration and experimentation**: REPLs provide a playground for developers to experiment with code snippets, try out different ideas, and

explore the behavior of the program in real-time. It encourages a more interactive and exploratory style of development.

3. **Incremental problem-solving**: Developers can break down complex problems into smaller sub-problems and solve them one step at a time. They can validate each step's correctness and build upon it iteratively.
4. **Rapid prototyping**: REPL-driven development allows developers to quickly sample ideas and test suggestions without the need for writing full-fledged programs. It enables faster validation of concepts and reduces the time required to see the results.
5. **Tight feedback loop**: The immediate feedback loop provided by a REPL environment helps in catching errors early, debugging code effectively, and gaining a better understanding of how code behaves.
6. **Learning and exploration**: REPLs can be valuable learning tools, especially for newcomers to a programming language or framework. They provide an interactive environment to experiment, ask "what if" questions, and understand how different language features work.
7. **Language exploration**: REPL-driven development is particularly popular in languages like Python, Ruby, Lisp, and JavaScript, where REPL environments are readily available and integrated into development tools. These languages' interactive nature aligns well with the REPL-driven workflow.

Deployment of the system: -

In DevOps, deployment systems are responsible for automating the release of software updates and applications from development to production. Some popular deployment systems include:

Jenkins: an open-source automation server that provides plugins to support building, deploying,and automating any project.

Ansible: an open-source platform that provides a simple way to automate software provisioning, configuration management, and application deployment.

Docker: a platform that enables developers to create, deploy, and run applications in containers.

Kubernetes: an open-source system for automating deployment, scaling, and management of containerized applications.

AWS Code Deploy: a fully managed deployment service that automates software deployments to a variety of compute services such as Amazon EC2, AWS Fargate, and on-premises servers.

Azure DevOps: a Microsoft product that provides an end-to-end DevOps solution for developing, delivering, and deploying applications on multiple platforms.

Virtualization stacks: -

Virtualization stacks are also known as virtualization software or hypervisors.

Virtualization stacks are software technologies that enable the creation and management of virtual machines (VMs) or virtualized environments on a physical host machine.

These stacks allow multiple operating systems or instances to run concurrently on the same hardware, providing isolation, resource allocation, and flexibility.

Here are some commonly used virtualization stacks:

1. **VMware vSphere**: VMware vSphere is a complete virtualization platform that includes the ESXi hypervisor and vCenter Server for centralized management. It supports a wide range of virtualization features, such as live migration, high availability, and distributed resource scheduling. VMware also offers other virtualization products like VMware Workstation and VMware Fusion for desktop virtualization.
2. **Microsoft Hyper-V**: Hyper-V is Microsoft's native hypervisor technology built into Windows Server and Windows 10 Pro/Enterprise. It provides virtualization capabilities for servers and desktops, supporting features like live migration, high availability, and clustering. Hyper-V integrates well with other Microsoft products and offers management tools like System Center Virtual Machine Manager (SCVMM).
3. **KVM (Kernel-based Virtual Machine)**: KVM is a virtualization solution for Linux that utilizes the Linux kernel's virtualization capabilities. It turns the host operating system into a hypervisor and allows running multiple VMs with different operating systems. KVM is open-source and widely used, and it can be managed using tools like libvirt and virt-manager.
4. Xen: Xen is an open-source hypervisor that supports paravirtualization and hardware-assisted virtualization. It provides strong isolation between virtual machines and allows running different operating systems concurrently. Xen is used in various commercial and open-source

virtualization solutions, including Citrix Hypervisor (formerly XenServer) and Oracle VM.

5. **Proxmox VE**: Proxmox Virtual Environment (VE) is an open-source virtualization platform that combines KVM and container- based virtualization using LXC (Linux Containers). Proxmox VE offers a web-based management interface and includes features like live migration, high availability, and backup/restore capabilities.
6. **Docker**: While not a traditional virtualization stack, Docker is a popular containerization platform that allows applications to run in lightweight, isolated containers. Docker containers share the host OS kernel, making them more lightweight and portable compared to VMs. Docker provides tools for container image creation, management, and deployment

These are just a few examples of virtualization stacks available today. Each stack has its own features, management tools, and compatibility requirements. The choice of a virtualization stack depends on factors such as the intended use case, hardware compatibility, management requirements, and the operating systems or applications to be virtualized.

Code execution at the client: -

In DevOps, code execution at the client refers to the process of executing code or scripts on client devices or machines. This can be accomplished in several ways, including:

Code execution at the client in a DevOps context typically refers to running code or scripts on the client machines or devices. This can be useful for various purposes such as deploying software updates, performing configurations, or executing specific tasks on client machines as part of a DevOps workflow.

Client-side scripting languages: JavaScript, HTML, and CSS are commonly used client-side scripting languages that run in a web browser and allow developers to create dynamic, interactive web pages.

Remote execution tools: Tools such as SSH, Telnet, or Remote Desktop Protocol (RDP) allow developers to remotely execute commands and scripts on client devices.

Configuration management tools: Tools such as Ansible, Puppet, or Chef use agent-based or agentless architectures to manage and configure client devices, allowing developers to execute code and scripts remotely.

Mobile apps: Mobile applications can also run code on client devices, allowing developers to create dynamic, interactive experiences for users.

These methods are used in DevOps to automate various tasks, such as application deployment, software updates, or system configuration, on client devices. By executing code on the client side, DevOps teams can improve the speed, reliability, and security of their software delivery process.

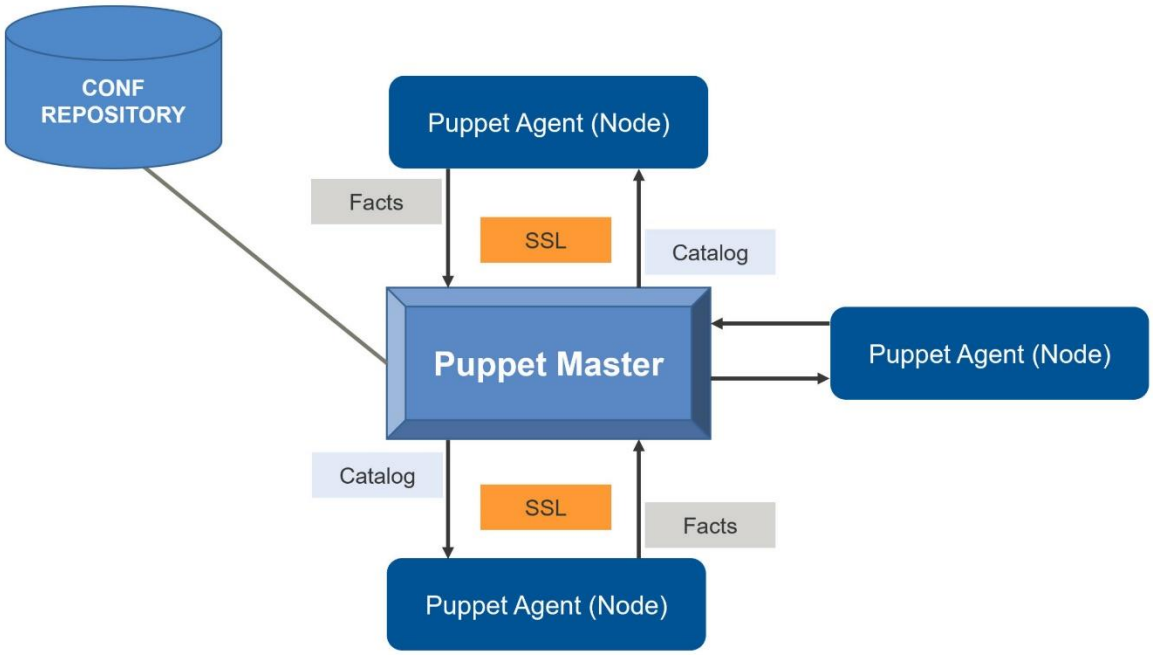
Puppet master and agents: -

Puppet is a configuration management technology to manage the infrastructure on physical or virtual machines. It is an open-source software configuration management tool developed using Ruby.

Puppet is a configuration management tool developed by Puppet Labs in order to automate infrastructure management and configuration

Puppet follows the client-server model, where one machine in any cluster acts as the server, known as puppet master and the other acts as a client known as a slave on nodes/puppet agents.

Puppet uses master-slave or client-server architecture. Puppet client and server interconnected by SSL, which is a secure socket layer. It is a model-driven system.



Here, the client is referred to as a Puppet agent/slave/node, and the server is referred to as a Puppet master.

Let's see the components of Puppet architecture:

Puppet Master

Puppet master handles all the configuration related process in the form of puppet codes. It is a Linux based system in which puppet master software is installed. The puppet master must be in Linux. It uses the puppet agent to apply the configuration to nodes. This is the place where SSL certificates are checked and marked.

Puppet Slave or Agent

Puppet agents are the real working systems and used by the Client. It is installed on the client machine and maintained and managed by the puppet master. They have a puppet agent service running inside them.

The agent machine can be configured on any operating system such as Windows, Linux, Solaris,or Mac OS.

Config Repository

Config repository is the storage area where all the servers and nodes related configurations are stored, and we can pull these configurations as per requirements.

Facts

Facts are the key-value data pair. It contains information about the node or the master machine. It represents a puppet client states such as operating system, network interface, IP address, uptime, and whether the client machine is virtual or not. These facts are used for determining the present state of any agent. Changes on any target machine are made based on facts. Puppet's facts are predefined and customized.

Catalog

The entire configuration and manifest files that are written in Puppet are changed into a compiled format. This compiled format is known as a catalog, and then we can apply this catalog to the target machine.

The above image performs the following functions:

o First of all, an agent node sends facts to the master or server and requests for a catalog.

o The master or server compiles and returns the catalog of a node with the help of some information accessed by the master.

o Then the agent applies the catalog to the node by checking every resource mentioned in the catalog. If it identifies resources that are not in their desired state, then makes the necessary

adjustments to fix them. Or, it determines in no-op mode, the adjustments would be required to reconcile the catalog.

o And finally, the agent sends a report back to the master.

Puppet Master-Slave Communication

Puppet master-slave communicates via a secure encrypted channel through the SSL (Secure Socket Layer). Let's see the below diagram to understand the communication between the master and slave with this channel:

The above diagram depicts the following:

o Puppet slave requests for Puppet Master Certificate.

o Puppet master sends the Master Certificate to the puppet slave in response to the client request.

o Puppet master requests to the Puppet slave for the slave certificate.

o Puppet slave sends the requested slave certificate to the puppet master.

o Puppet slave sends a request for data to the puppet master.

o Finally, the master sends the data to the puppet slave as per the request.

Ansible: -

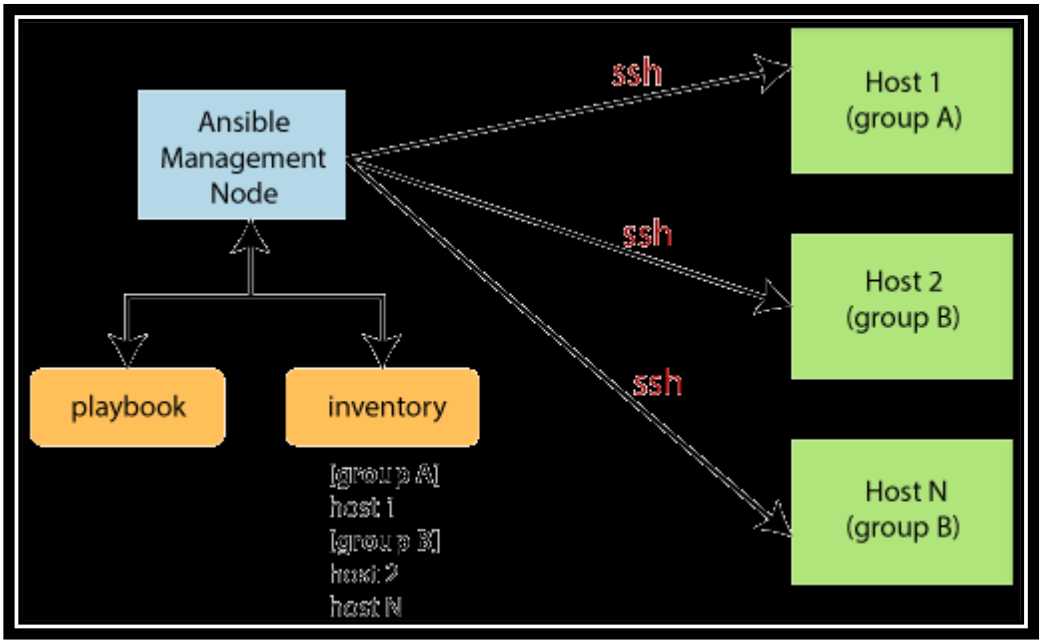
Ansible is simple open source IT engine which automates application deployment, intra service orchestration, cloud provisioning and many other IT tools.

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

It focuses on simplicity, ease of use, and agentless operation, making it popular among DevOps teams for managing large- scale infrastructure.

Ansible Workflow

Ansible works by connecting to your nodes and pushing out a small program called **Ansible modules** to them. Then Ansible executed these modules and removed them after finished. The library of modules can reside on any machine, and there are no daemons, **servers,** or **databases** required.



In the above image, the **Management Node** is the controlling node that controls the entire execution of the playbook. The **inventory** file provides the list of hosts where the Ansible modules need to be run. The **Management Node** makes an **SSH** connection and executes the small modules on the host's machine and install the software.

Ansible removes the modules once those are installed so expertly. It connects to the host machine executes the instructions, and if it is successfully installed, then remove that code in which one was copied on the host machine

Playbooks

Playbooks consist of your written code, and they are written in YAML format, which describes the tasks and executes through the Ansible. Also, you can launch the tasks synchronously and asynchronously with playbooks.

Key features and concepts of Ansible include:

1. **Playbooks**: Ansible uses Playbooks, which are written in YAML format, to define automation tasks and desired configurations. Playbooks describe a set of steps that should be executed on target systems to achieve a desired state.
2. **Inventory**: Ansible uses an Inventory file that lists the target systems or groups of systems (referred to as hosts) that Ansible manages. This file defines the hosts' connection details, such as IP addresses or hostnames.
3. **Modules**: Ansible provides a wide range of built-in modules that perform specific tasks on the target hosts. Modules can handle tasks like file management, package installation, service management, executing commands, or making API calls. Modules are executed by Ansible on the target hosts to bring them to the desired state.
4. **Play**: A Play in Ansible is a combination of one or more tasks that target a specific group of hosts defined in the Inventory. Each Play in a playbook targets a set of hosts and defines the tasks to be executed on those hosts.
5. **Roles**: Roles in Ansible provide a way to organize and reuse tasks, handlers, and variables across multiple playbooks. Roles encapsulate a specific functionality or configuration and can be shared and reused across projects.
6. **Ad-hoc Commands**: Ansible allows you to run ad-hoc commands on target hosts without the need for writing a playbook. Ad-hoc commands are useful for performing quick tasks, like executing a command, copying files, or restarting services.
7. **Idempotent Execution**: Ansible follows an idempotent execution model, meaning that if a task has been executed once and the system is already in the desired state, subsequent runs of the playbook will not make unnecessary changes. This ensures predictable and consistent configurations.
8. **Ansible Galaxy**: Ansible Galaxy is a hub for sharing and discovering reusable Ansible roles. It provides a repository of pre-built roles

contributed by the Ansible community, allowing you to leverage existing configurations and best practices.

Ansible's agentless architecture allows it to operate over SSH or PowerShell connections, making it easier to manage heterogeneous infrastructure. It offers powerful capabilities for automating infrastructure management, application deployment, and continuous delivery pipelines, enabling efficient and streamlined DevOps practices.

Deployment tools: -

Chef: -

Chef is an open source technology developed by Opscode. Adam Jacob, co-founder of Opscode is known as the founder of Chef. This technology uses Ruby encoding to develop basic building blocks like recipe and cookbooks. Chef is used in infrastructure automation and helps inreducing manual and repetitive tasks for infrastructure management.

Chef have got its own convention for different building blocks, which are required to manage and automate infrastructure.

Key Building Blocks of Chef

Recipe

It can be defined as a collection of attributes which are used to manage the infrastructure. These attributes which are present in the recipe are used to change the existing state or setting a particular infrastructure node. They are loaded during Chef client run and comparted with the existing attribute of the node (machine). It then gets to the status which is defined in the node resource of the recipe. It is the main workhorse of the cookbook.

Cookbook

A cookbook is a collection of recipes. They are the basic building blocks which get uploaded to Chef server. When Chef run takes place, it ensures that the recipes present inside it gets a giveninfrastructure to the desired state as listed in the recipe.

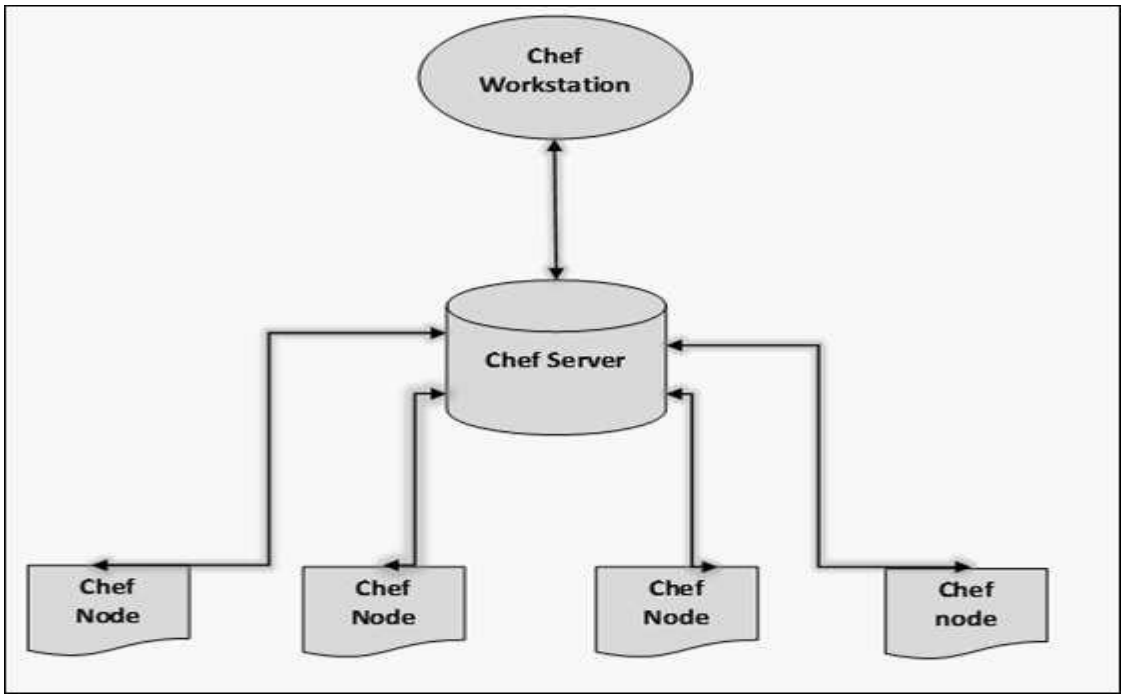
Resource

It is the basic component of a recipe used to manage the infrastructure with different kind of states. There can be multiple resources in a recipe, which will help in configuring and managing the infrastructure. For example −

- **package** − Manages the packages on a node
- **service** − Manages the services on a node
- **user** − Manages the users on the node
- **group** − Manages groups
- **template** − Manages the files with embedded Ruby template
- **cookbook_file** − Transfers the files from the files subdirectory in the cookbook to a location on the node
- **file** − Manages the contents of a file on the node
- **directory** − Manages the directories on the node
- **execute** − Executes a command on the node
- **cron** − Edits an existing cron file on the node

Chef – Architecture

Chef works on a three-tier client server model wherein the working units such as cookbooks are developed on the Chef workstation. From the command line utilities such as knife, they are uploaded to the Chef server and all the nodes which are present in the architecture are registered with the Chef server.



● In order to get the working Chef infrastructure in place, we need to set up multiple things in sequence.

● In the above setup, we have the following components.

● Chef Workstation

● This is the location where all the configurations are developed. Chef workstation is installed on the local machine. Detailed configuration structure is discussed in the later chapters of this tutorial.

● Chef Server

● This works as a centralized working unit of Chef setup, where all the configuration files are uploaded post development. There are different kinds of Chef server, some are hosted Chef server whereas some are built-in premise.

● Chef Nodes

● They are the actual machines which are going to be managed by the Chef server. All the nodes can have different kinds of setup as per requirement. Chef client is the key component of all the nodes, which helps in setting up the communication between the Chef server and Chef node. The other components of Chef node is Ohai, which helps in getting the current state of any node at a given point of time.

Salt Stack: -

Salt Stack is an open-source configuration management software and remote execution engine. Salt is a command-line tool. While written in Python, SaltStack configuration management is language agnostic and simple. Salt platform uses the push model for executing commands via the SSH protocol. The default configuration system is YAML and Jinja templates. Salt is primarily competing with Puppet, Chef and Ansible.

There is a convenient dockerized test environment for Salt, by Jackson Cage.

Salt provides many features when compared to other competing tools. Some of these important features are listed below.

● **Fault tolerance** − Salt minions can connect to multiple masters at one time by configuring the master configuration parameter as a YAML list of all the available masters. Any master can direct commands to the Salt infrastructure.

● **Flexible** − The entire management approach of Salt is very flexible. It can be implemented to follow the most popular systems management models such as Agent and Server, Agent-only, Server-only or all of the above in the same environment.

● **Scalable Configuration Management** − SaltStack is designed to handle ten thousand minions per master.

● **Parallel Execution model** − Salt can enable commands to execute remote systems in a parallel manner.

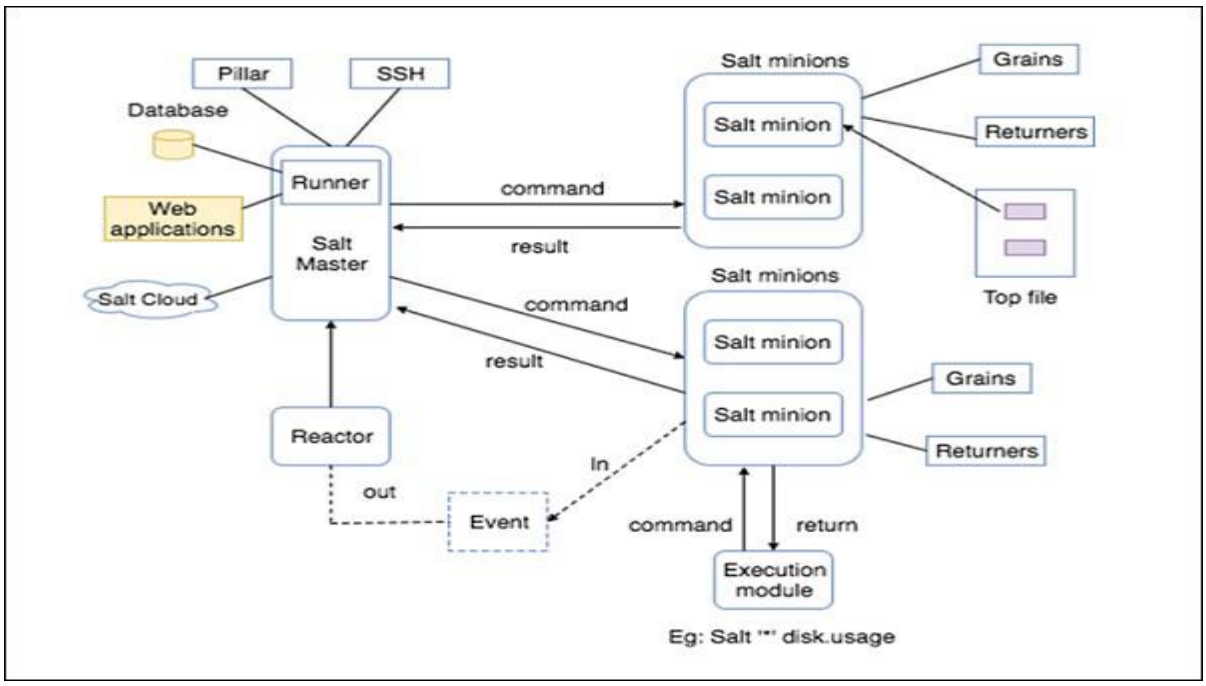
● **Python API** − Salt provides a simple programming interface and it was designed to be modular and easily extensible, to make it easy to mold to diverse applications.

● **Easy to Setup** − Salt is easy to setup and provides a single remote execution architecture that can manage the diverse requirements of any number of servers.

● **Language Agnostic** − Salt state configuration files, templating engine or file type supports any type of language.

SaltStack – Architecture

The architecture of SaltStack is designed to work with any number of servers, from local network systems to other deployments across different data centers. Architecture is a simple server/client model with the needed functionality built into a single set of daemons. Take a look at the following illustration. It shows the different components of SaltStack architecture.



● **SaltMaster** − SaltMaster is the master daemon. A SaltMaster is used to send commands and configurations to the Salt slaves. A single master can manage multiple masters.

● **SaltMinions** − SaltMinion is the slave daemon. A Salt minion receives commands and configuration from the SaltMaster.

● **Execution** − Modules and Adhoc commands executed from the command line against one or more minions. It performs Real-time Monitoring.

● **Formulas** − Formulas are pre-written Salt States. They are as open-ended as Salt States themselves and can be used for tasks such as installing a package, configuring and starting a service, setting up users or permissions and many other common tasks.

● **Grains** − Grains is an interface that provides information specific to a minion. The information available through the grains interface is static. Grains get loaded when the Salt minion starts. This means that the information in grains is unchanging. Therefore, grains information could be about the running kernel or the operating system. It is case insensitive.

● **Pillar** − A pillar is an interface that generates and stores highly sensitive data specific to a particular minion, such as cryptographic keys and passwords. It stores data in a key/value pair and the data is managed in a similar way as the Salt State Tree.

● **Top File** − Matches Salt states and pillar data to Salt minions.

● **Runners** − It is a module located inside the SaltMaster and performs tasks such as job status, connection status, read data from external APIs, query connected salt minions and more.

● **Returners** − Returns data from Salt minions to another system.

● **Reactor** − It is responsible for triggering reactions when events occur in your SaltStack environment.

● **SaltCloud** − Salt Cloud provides a powerful interface to interact with cloud hosts.

● **SaltSSH** − Run Salt commands over SSH on systems without using Salt minion.

Docker: -

Docker is a container management service. The keywords of Docker are **develop,**

ship and **run** anywhere. The whole idea of Docker is for developers to easily develop applications, ship them into containers which can then be deployed anywhere. The initial release of Docker was in March 2013 and since then, it has become the buzzword for modern world development, especially in the face of Agile-based projects. Features of Docker

● Docker has the ability to reduce the size of development by providing a smaller footprint of the operating system via containers.

● With containers, it becomes easier for teams across different units, such as development, QA and Operations to work seamlessly across applications.

● You can deploy Docker containers anywhere, on any physical and virtual machines and even on the cloud.

● Since Docker containers are pretty lightweight, they are very easily scalable.

Components of Docker

Docker has the following components

● **Docker for Mac** − It allows one to run Docker containers on the Mac OS.

● **Docker for Linux** − It allows one to run Docker containers on the Linux OS.

● **Docker for Windows** − It allows one to run Docker containers on the Windows OS.

● **Docker Engine** − It is used for building Docker images and creating Docker containers.

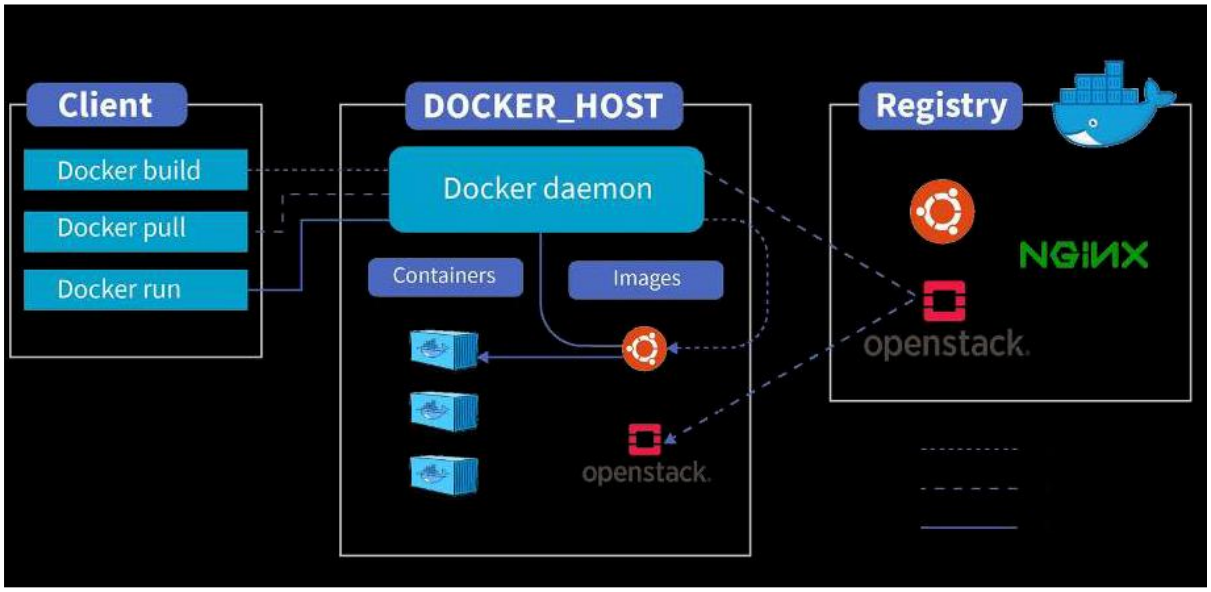
● **Docker Hub** − This is the registry which is used to host various Docker images.

● **Docker Compose** − This is used to define applications using multiple Docker containers.

Docker architecture

Docker uses a client-server architecture. The Docker *client* talks to the Docker *daemon*, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon *can* run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client

and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.



The Docker daemon

The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

The Docker client

The Docker client (docker) is the primary way that many Docker users interact with Docker. When you use commands such as docker run, the client sends these commands to dockerd, which carries them out. The docker command uses the Docker API. The Docker client can communicate with more than one daemon.

Docker Desktop

Docker Desktop is an easy-to-install application for your Mac, Windows or Linux environment that enables you to build and share containerized applications and microservices. Docker Desktop includes the Docker daemon (dockerd), the Docker client (docker), Docker Compose, Docker Content Trust, Kubernetes, and Credential Helper. For more information, see Docker Desktop.

Docker registries

A Docker *registry* stores Docker images. Docker Hub is a public registry that anyone can use, and Docker is configured to look for images on Docker Hub by default. You can even run yourown private registry.

When you use the docker pull or docker run commands, the required images are pulled from your configured registry. When you use the docker push command, your image is pushed to your configured registry.

Docker objects

When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.

Images

An *image* is a read-only template with instructions for creating a Docker container. Often, an image is *based on* another image, with some additional customization. For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.

You might create your own images or you might only use those created by others and published in a registry. To build your own image, you create a *Dockerfile* with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.

Containers

A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.

By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container's network, storage, or other underlying subsystems are from other containers or from the host machine.

A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that are not stored in persistent storage disappear.

The underlying technology

Docker is written in the Go programming language and takes advantage of several features of the Linux kernel to deliver its functionality. Docker uses a technology

called namespaces to provide the isolated workspace called the container. When you run a container, Docker creates a set of namespaces for that container.

These namespaces provide a layer of isolation. Each aspect of a container runs in a separate namespace and its access is limited to that namespace.

----------XXXXXXXX----------