DEVOPS

Course Objectives:

The main objectives of this course are to:

1. Describe the agile relationship between development and IT operations.

2. Understand the skill sets and high-functioning teams involved in DevOps and related methods to reach a continuous delivery capability.

3. Implement automated system update and DevOps lifecycle.

UNIT - I

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

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.

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.

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.

UNIT - V

Testing Tools and automation: 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

TEXT BOOKS:

1. Joakim Verona. Practical Devops, Second Edition. Ingram short title; 2ndedition (2018). ISBN10: 1788392574

2. Deepak Gaikwad, Viral Thakkar. DevOps Tools from Practitioner's Viewpoint. Wiley publications. ISBN: 9788126579952

REFERENCE BOOK:

1. Len Bass, Ingo Weber, Liming Zhu. DevOps: A Software Architect's Perspective. Addison Wesley; ISBN-10.

Course Outcomes: On successful completion of this course, students will be able to:

1. Identify components of Devops environment.

2. Describe Software development models and architectures of DevOps.

3. Apply different project management, integration, testing and code deployment tool.

4. Investigate different DevOps Software development models.

5. Assess various Devops practices, Collaborate and adopt Devops in real-time projects.

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.

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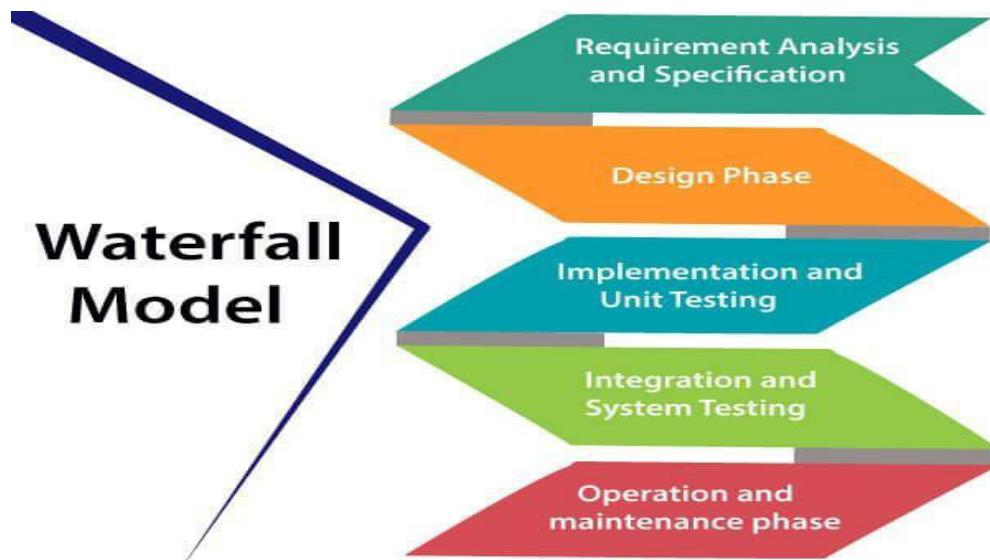
2. Design Phase: This phase aims to transform the requirements gathered in the SRS into a suitable form which permits further coding in a programming language. It defines the overall software architecture together with high level and detailed design. All this work is documented as a Software Design Document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

3

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.

4
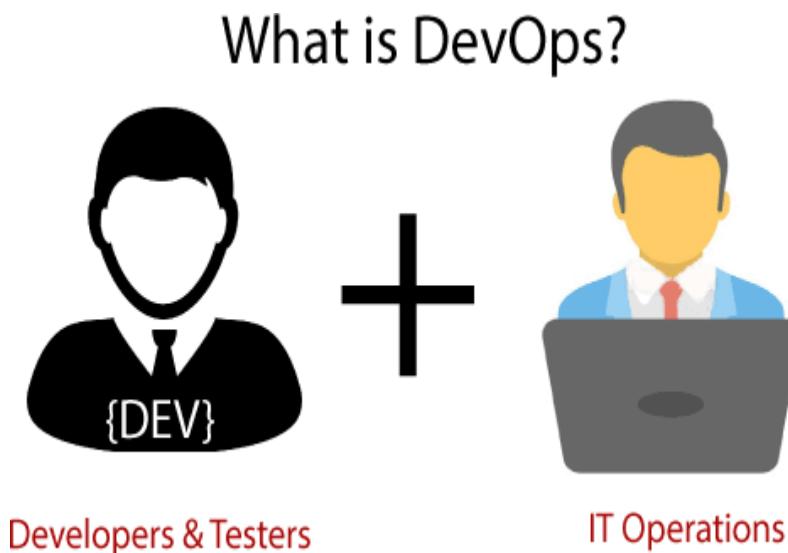
- This model cannot accept the changes in requirements during development.
- It becomes tough to go back to the phase. For example, if the application has now shifted to the coding phase, and there is a change in requirement, It becomes tough to go back and change it.
- Since the testing done at a later 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.

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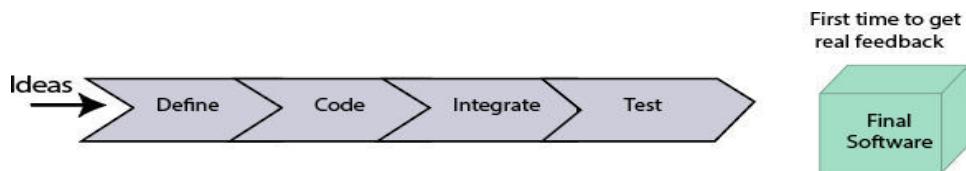
- Without the use of DevOps, the team members are spending a large amount of time on designing, testing, and deploying instead of building the project.
- Manual code deployment leads to human errors in production.
- Coding and operation teams have their separate timelines and are not in synch, causing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

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

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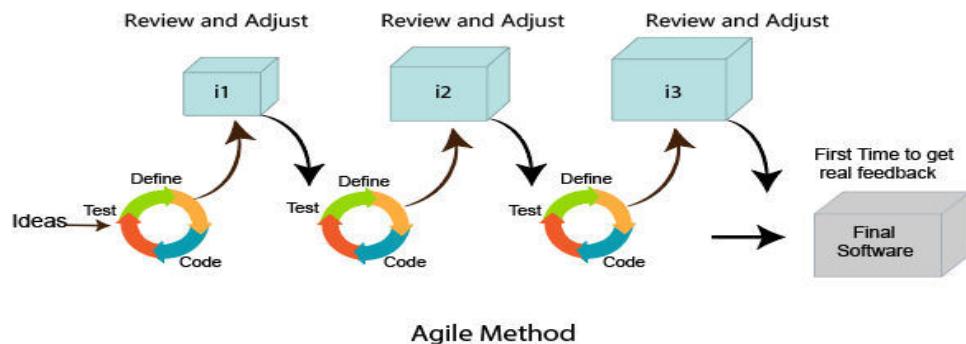
An agile methodology is an iterative approach to software development. Each iteration of agile methodology takes a short time interval of 1 to 4 weeks. The agile development process is aligned to deliver the changing business requirement. It distributes the software with faster and fewer changes.

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.

Traditional Method

Agile Method

ITIL

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

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.

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.

It is not a standard but a collection of best practices guidelines.

Service Lifecycle in ITIL

1. Service Strategy.
2. Service Design.
3. Service Transition.
4. Service Operation.
5. Continual Service Improvement.

Service Strategy

Service Strategy is the first and initial stage in the lifecycle of the ITIL framework. The main aim of this stage is that it offers a strategy on the basis of the current market scenario and business perspective for the services of IT.

This stage mainly defines the plans, position, patters, and perspective which are required for a service provider. It establishes the principles and policies which guide the whole lifecycle of IT service.

Following are the various essential services or processes which comes under the **Service Strategy** stage:

- o Financial Management
- o Demand Management
- o Service Portfolio Management
- o Business Relationship Management
- o Strategy Management

Strategy Management:

The aim of this management process is to define the offerings, rivals, and capabilities of a service provider to develop a strategy to serve customers.

According to the version 3 (V3) of ITIL, this process includes the following activities for IT services:

1. Identification of Opportunities
2. Identification of Constraints
3. Organizational Positioning
4. Planning
5. Execution

Following are the three sub-processes which comes under this management process:

1. Strategic Service Assessment
2. Service Strategy Definition
3. Service Strategy Execution

Financial Management:

This process helps in determining and controlling all the costs which are associated with the services of an IT organization. It also contains the following three basic activities:

1. Accounting 2. charging 3. Budgeting

Following are the four sub-processes which comes under this management process:

1. Financial Management Support
2. Financial Planning
3. Financial Analysis and Reporting
4. Service Invoicing

Demand Management

This management process is critical and most important in this stage. It helps the service providers to understand and predict the customer demand for the IT services. Demand management is a process which also work with the process of Capacity Management. Following are basic objectives of this process:

- o   This process balances the resources demand and supply.
- o   It also manages or maintains the quality of service.

According to the version 3 (V3) of ITIL, this process performs the following 3 activities:

1. Analysing current Usage of IT services
2. Anticipate the Future Demands for the Services of IT.
3. Influencing Consumption by Technical or Financial Means

Following are the two sub-processes which comes under this management process:

1. Demand Prognosis
2. Demand Control.

Business Relationship Management

This management process is responsible for maintaining a positive and good relationship between the service provider and their customers. It also identifies the needs of a customer. And, then ensure that the services are implemented by the service provider to meet those requirements.

This process has been released as a new process in the ITIL 2011.

According to the version 3 (V3) of ITIL, this process performs the following various activities:

- o   This process is used to represent the service provider to the customer in a positive manner.
- o   This process identifies the business needs of a customer.
- o   It also acts as a mediator if there is any case of conflicting requirements from the different businesses.

Following are the six sub-processes which comes under this management process:

1. Maintain Customer Relationships
2. Identify Service Requirements

3. Sign up Customers to standard Services
4. Customer Satisfaction Survey
5. Handle Customer Complaints
6. Monitor Customer Complaints.

Service Portfolio Management

This management process defines the set of customer-oriented services which are provided by a service provider to meet the customer requirements. The primary goal of this process is to maintain the service portfolio.

Following are the three types of services under this management process:

1. Live Services 2. Retired Services 3. Service Pipeline.

Following are the three sub-processes which comes under this management process:

1. Define and Analyse the new services or changed services of IT.
2. Approve the changes or new IT services
3. Service Portfolio review.

Service Design

It is the second phase or a stage in the lifecycle of a service in the framework of ITIL. This stage provides the blueprint for the IT services. The main goal of this stage is to design the new IT services. We can also change the existing services in this stage.

Following are the various essential services or processes which comes under the Service Design stage:

o Service Level Management
o Capacity Management
o Availability Management
o Risk Management
o Service Continuity Management
o Service Catalogue Management
o Information Security Management
o Supplier Management
o Compliance Management
o Architecture Management

Service Level Management

In this process, the **Service Level Manager** is the process owner. This management is fully redesigned in the ITIL 2011.

Service Level Management deals with the following two different types of agreements:

1. Operational Level Agreement
2. Service Level Agreement

According to the version 3 (V3) of ITIL, this process performs the following activities:

o It manages and reviews all the IT services to match service Level Agreements.
o It determines, negotiates, and agrees on the requirements for the new or changed IT services.

Following are the four sub-processes which comes under this management process:

1. Maintenance of SLM framework
2. Identifying the requirements of services
3. Agreements sign-off and activation of the IT services
4. Service level Monitoring and Reporting.

Capacity Management

This management process is accountable for ensuring that the capacity of the IT service can meet the agreed capacity in a cost-effective and timely manner. This management process is also working with other processes of ITIL for accessing the current infrastructure of IT.

According to the version 3 (V3) of ITIL, this process performs the following activities:

o It manages the performance of the resources so that the IT services can easily meet their SLA targets.
o It creates and maintains the capacity plan which aligns with the strategic plan of an organization.
o It reviews the performance of a service and the capacity of current service periodically.
o It understands the current and future demands of customer for the resources of IT.

Following are the four sub-processes which comes under this management process:

1. Business Capacity Management
2. Service Capacity Management

3. Component Capacity Management

4. Capacity Management Reporting

Availability Management

In this process, the **Availability Manager** is the owner. This management process has a responsibility to ensure that the services of IT meet the agreed availability goals. This process also confirms that the services which are new or changed does not affect the existing services.

It is used for defining, planning, and analysing all the availability aspects of the services of IT.

According to the version 3 (V3) of ITIL, this process contains the following two activities:

1. Reactive Activity
2. Proactive Activity

Following are the four sub-processes which comes under this management process:

1. Design the IT services for availability
2. Availability Testing
3. Availability Monitoring and Reporting

Risk Management

In this process, the **Risk Manager** is the owner. This management process allows the risk manager to check, assess, and control the business risks. If any risk is identified in the process of business, the risk of that entry is created in the ITIL Risk Register.

According to the version 3 (V3) of ITIL, this process performs the following activities in the given order:

o It identifies the threats.
o It finds the probability and impact of risk.
o It checks the way for reducing those risks.
o It always monitors the risk factors.

Following are the four sub-processes which comes under the Risk process:

1. Risk Management Support
2. Impact on business and Risk analysis
3. Monitoring the Risks.

4. Assessment of Required Risk Mitigation

Service Catalogue Management (SCM)

In this process, the **Service Catalogue Manager** is the owner. This management process allows the Catalogue Manager to give the huge information about all the other management processes.

It contains the services in the service operation phase which are presently active.

It is a process which certifies that the service catalogue is maintained, produced, and contains all the accurate information for all the operational IT services.

Following are the two types or aspects of service catalogue in ITIL framework:

1. BSC or Business Service Catalogue
2. TSC or Technical Service Catalogue

Under this management process, no sub-process is specified or defined.

Service Continuity Management

In this process, the **IT Service Continuity Manager** is specified as the owner. It allows the continuity manager to maintain the risks which could impact on the service of IT.

This process is bound with other processes of ITIL such as capacity and availability management to access and plan the resources which are needed to manage the desired service level.

The ITSCM consists of the following four activities or stages:

1. Initiation
2. Requirements and Strategy
3. Implementation
4. Ongoing Operation

Information Security Management

In this process, the **Information Security Manager** is specified as the owner. The main aim of this management process is to verify the confidentiality, integrity, and availability of the data, information, and services of an IT organization.

The main objective of this process is to control the access of information in the organizations.

According to the version 3 (V3) of ITIL, this process performs the following four activities:

1. Plan   2. Implement   3. Evaluation   4. Maintain

According to the version 3 (V3) of ITIL, following are the four sub-processes which comes under this management process:

1. Design of Security controls
2. Validation and Testing of Security
3. Management of Security Incidents
4. Security Review

Supplier Management

In this process, the **Supplier Manager** plays a role as an owner. The supplier manager is responsible to verify that all the suppliers meet their contractual commitments.

It also works with the Financial and knowledge management, which helps in selecting the suppliers on the basis of previous knowledge.

Following are the various activities which are involved in this process:

o It manages the sub-contracted suppliers.
o It manages the relationship with the suppliers.
o It helps in implementing the supplier policy.
o It also manages the supplier policy and supports the SCMIS.
o It also manages or maintains the performance of the suppliers.

According to the version 3 (V3) of ITIL, following are the six sub-processes which comes under this management process:

1. Provide the Framework of Supplier Management
2. Evaluation and selection of new contracts and suppliers
3. Establish the new contracts and suppliers
4. Process the standard orders
5. Contract and Supplier Review
6. Contract Renewal or Termination.

Compliance Management

In this process, the **Compliance Manager** plays a role as an owner. This management process allows the compliance manager to check and address all the issues which are associated with regulatory and non-regulatory compliances.

Under this compliance management process, no sub-process is specified or defined.

Here, the role of Compliance Manager is to certify that the guidelines, legal requirements, and standards are being followed properly or not. This manager works in parallel with the following three managers:

1. Information Security Manager
2. Financial Manager
3. Service Design Manager.

Architecture Management

In this process, the **Enterprise Architect** plays a role as an owner. The main aim of Enterprise Architect is to maintain and manage the architecture of the Enterprise.

This management process helps the **Enterprise Architect** by verifying that all the deployed services and products operate according to the specified architecture baseline in the Enterprise.

This process also defines and manages a baseline for the future technological development.

Under this Architecture management process, no sub-process is specified or defined.

Service Transition

Service Transition is the third stage in the lifecycle of ITIL Management Framework.

The main goal of this stage is to build, test, and develop the new or modified services of IT. This stage of service lifecycle manages the risks to the existing services. It also certifies that the value of a business is obtained.

This stage also makes sure that the new and changed IT services meet the expectation of the business as defined in the previous two stages of service strategy and service design in the lifecycle.

It can also easily manage or maintains the transition of new or modified IT services from the **Service Design** stage to **Service Operation** stage.

There are following various essential services or processes which comes under the Service Transition stage:

- o Change Management
- o Release and Deployment Management
- o Service Asset and Configuration Management
- o Knowledge Management
- o Project Management (Transition Planning and Support)
- o Service Validation and Testing
- o Change Evaluation

Change Management

In this process, the **Change Manager** plays a role as an owner. The Change Manager controls or manages the service lifecycle of all changes. It also allows the change Manager to implement all the essential changes to be required with the less disruption of IT services.

This management process also allows its owner to recognize and stop any unintended change activity. Actually, this management process is tightly bound with the process **"Service Asset and Configuration Management"**.

Following are the three types of changes which are defined by the ITIL.

1. Normal Change
2. Standard Change
3. Emergency Change

All these changes are also known as the Change Models.

According to the version 3 (V3) of ITIL, following are the eleven sub-processes which comes under this Change management process:

1. Change Management Support
2. RFC (Request for Change) Logging and Review
3. Change Assessment by the Owner (Change Manager)
4. Assess and Implement the Emergency Changes
5. Assessment of change Proposals

6. Change Scheduling and Planning
7. Change Assessment by the CAB
8. Change Development Authorization
9. Implementation or Deployment of Change
10. Minor change Deployment
11. Post Implementation Review and change closure

Release and Deployment Management

In this process, the **Release Manager** plays a role as an owner. Sometimes, this process is also known as the 'ITIL Release Management Process'.

This process allows the Release Manager for managing, planning, and controlling the updates & releases of IT services to the real environment.

Following are the three types of releases which are defined by the ITIL.

1. Minor release
2. Major Release
3. Emergency Release

According to the version 3 (V3) of ITIL, following are the six sub-processes which comes under this Change management process:

1. Release Management Support
2. Release Planning
3. Release build
4. Release Deployment
5. Early Life Support
6. Release Closure

Service Asset and Configuration Management

In this process, the **Configuration Manager** plays a role as an owner.

This management process is a combination of two implicit processes:

1. Asset Management
2. Configuration Management

The aim of this management process is to manage the information about the (CIs) Configuration Items which are needed to deliver the services of IT. It contains information about versions, baselines, and the relationships between assets.

According to the version 3 (V3) of ITIL, following are the five sub-processes which comes under this Change management process:

1. Planning and Management
2. Configuration Control and Identification
3. Status Accounting and reporting
4. Audit and Verification
5. Manage the Information

Knowledge Management

In this process, the **Knowledge Manager** plays a role as an owner. This management process helps the Knowledge Manager by analysing, storing and sharing the knowledge and the data or information in an entire IT organization.

Under this Knowledge Management Process, no sub-process is specified or defined.

Transition Planning and Support

In this process, the **Project Manager** plays a role as an owner. This management process manages the service transition projects. Sometimes, this process is also known as the Project Management Process.

In this process, the project manager is accountable for planning and coordinating resources to deploy IT services within time, cost, and quality estimates.

According to the version 3 (V3) of ITIL, this process performs the following activities:

o It manages the issues and risks.
o It defines the tasks and activities which are to be performed by the separate processes.
o It makes a group with the same type of releases.
o It manages each individual deployment as a separate project.

According to the version 3 (V3) of ITIL, following are the four sub-processes which comes under this Project management process:

1. Initiate the Project

2. Planning and Coordination of a Project
3. Project Control
4. Project Communication and Reporting

Service Validation and Testing

In this process, the **Test Manager** plays a role as an owner. The main goal of this management process is that it verifies whether the deployed releases and the resulting IT service meets the customer expectations.

It also checks whether the operations of IT are able to support the new IT services after the deployment. This process allows the Test Manager to remove or delete the errors which are observed at the first phase of the service operation stage in the lifecycle.

It provides the quality assurance for both the services and components. It also identifies the risks, errors and issues, and then they are eliminated through this current stage.

This management process has been released in the version 3 of ITIL as a new process.

Following are the various activities which are performed under this process:

o   Validation and Test Management
o   Planning and Design
o   Verification of Test Plan and Design
o   Preparation of the Test Environment
o   Testing
o   Evaluate Exit Criteria and Report
o   Clean up and closure

According to the version 3 (V3) of ITIL, following are the four sub-processes which comes under this management process:

1. Test Model Definition
2. Release Component Acquisition
3. Release Test
4. Service Acceptance Testing

Change Evaluation

In this process, the **Change Manager** plays a role as an owner. The goal of this management process is to avoid the risks which are associated with the major changes for reducing the chances of failures.

This process is started and controlled by the change management and performed by the change manager.

Following are the various activities which are performed under this process:

- o It can easily identify the risks.
- o It evaluates the effects of a change.

According to the version 3 (V3) of ITIL, following are the four sub-processes which comes under this management process:

1. Change the Evaluation prior to Planning
2. Change the Evaluation prior to Build
3. Change the Evaluation prior to Deployment
4. Change the Evaluation prior after Deployment

Service Operations

Service Operations is the fourth stage in the lifecycle of ITIL. This stage provides the guidelines about how to maintain and manage the stability in services of IT, which helps in achieving the agreed level targets of service delivery.

This stage is also responsible for monitoring the services of IT and fulfilling the requests. In this stage, all the plans of transition and design are measured and executed for the actual efficiency. It is also responsible for resolving the incidents and carrying out the operational tasks.

There are following various essential services or processes which comes under the stage of Service Operations:

- o Event Management
- o Access Management
- o Problem Management
- o Incident Management
- o Application Management

  o Technical Management

Event Management

In this process, the **IT Operations Manager** plays a role as an owner. The main goal of this management process is to make sure that the services of IT and CIs are constantly monitored. It also helps in categorizing the events so that appropriate action can be taken if needed.

In this Management process, the process owner takes all the responsibilities of processes and functions for the multiple service operations.

Following are the various purposes of Event Management Process:

  o It allows the IT Operations Manager to decide the appropriate action for the events.
  o It also provides the trigger for the execution of management activities of many services.
  o It helps in providing the basis for service assurance and service improvement.

The Event Monitoring Tools are divided into two types, which are defined by the Version 3 (V3) of ITIL:

  1. Active Monitoring Tool
  2. Passive Monitoring Tool

Following are the three types of events which are defined by the ITIL:

  1. Warning
  2. Informational
  3. Exception

According to the version 3 (V3) of ITIL, following are the four sub-processes which comes under this management process:

  1. Event Monitoring and Notification
  2. First level Correlation and Event Filtering
  3. Second level Correlation and Response Selection
  4. Event Review and Closure.

Access Management

In this process, the **Access Manager** plays a role as an owner. This type of Management process

is also sometimes called as the **'Identity Management'** or **'Rights Management'**.
The role of a process manager is to provide the rights to use the services for authorized users.

In this Management process, the owner of a process follows those policies and guidelines which are defined by the (ISM) '**Information Security Management**'.

Following are the six activities which come under this management process and are followed sequentially:

1. Request Access
2. Verification
3. Providing Rights
4. Monitoring or Observing the Identity Status
5. Logging and Tracking Status
6. Restricting or Removing Rights

According to the version 3 (V3) of ITIL, following are the two sub-processes which comes under this management process:

1. Maintenance of Catalogue of User Roles and Access profiles
2. Processing of User Access Requests.

Problem Management

In this process, the **Problem Manager** plays a role as an owner. The main goal of this management process is to maintain or manage the life cycle of all the problems which happen in the services of IT. In the ITIL Framework, the problem is referred to as "**an unknown cause or event of one or more incident**".

It helps in finding the root cause of the problem. It also helps in maintaining the information about the problems.

Following are the ten activities which come under this management process and are followed sequentially. These ten activities are also called as a lifecycle of Problem Management:

1. Problem Detection
2. Problem Logging
3. Categorization of a Problem
4. Prioritization of a Problem
5. Investigation and Diagnosis of a Problem
6. Identify Workaround

7. Raising a Known Error Record
8. Resolution of a Problem
9. Problem Closure
10. Major Problem Review

Incident Management

In this process, the **Incident Manager** plays a role as an owner. The main goal of this management process is to maintain or manage the life cycle of all the incidents which happen in the services of IT.

An incident is a term which is defined as the failure of any Configuration Item (CI) or reduction in the quality of services of IT.

This management process maintains the satisfaction of users by managing the qualities of IT service. It increases the visibility of incidents.

According to the version 3 (V3) of ITIL, following are the nine sub-processes which comes under this management process:

1. Incident Management Support
2. Incident Logging and Categorization
3. Pro-active User Information
4. First Level Support for Immediate Incident Resolution
5. Second Level Support for Incident Resolution
6. Handling of Major Incidents
7. Incident Monitoring and Escalation
8. Closure and Evaluation of Incident
9. Management Reporting of Incident

Application Management

In this function, the **Application Analyst** plays a role as an owner.

This management function maintains or improves the applications throughout the entire service lifecycle. This function plays an important and essential role in the applications and system management.

Under this management function, no sub-process is specified or defined. But, this management

function into the following six activities or stages:

1. Define   2. Design   3. Build   4. Deploy   5. Operate   6. Optimize

Technical Management

In this function, the **Technical Analyst** plays a role as an owner. This function acts as standalone in the IT organizations, which basically consists of technical people and teams. The main goal of this function is to provide or offer the technical expertise. And, it also supports for maintaining or managing of IT infrastructure throughout the entire lifecycle of a service.

The role of the Technical Analyst is to develop the skills, which are required to

operate the day-to-day operations of IT infrastructure. Under this management function, no sub-process is specified or defined.

Continual Service Improvement

It is the fifth stage in the lifecycle of ITIL service. This stage helps to identify and implement strategies, which is used for providing better services in future.

Following are the various objectives or goals under this CSI:

- o   It improves the quality services by learning from the past failures.
- o   It also helps in analyzing and reviewing the improvement opportunities in every phase of the service lifecycle.
- o   It also evaluates the service level achievement results.
- o   It also describes the best guidelines to achieve the large-scale improvements in the quality of service.
- o   It also helps in describing the concept of KPI, which is a process metrics-driven for evaluating and reviewing the performance of the services.

There are following various essential services or processes which comes under the stage of CSI:

- o   Service Review
- o   Process Evaluation
- o   Definition of CSI Initiatives
- o   Monitoring of CSI Initiatives

This stage follows the following six-step approach (pre-defined question) for planning, reviewing, and implementing the improvement process:

Service Review

In this process, the **CSI Manager** plays a role as an owner. The main aim of this management process is to review the services of business and infrastructure on a regular basis.

Sometimes, this process is also called as "**ITIL Service Review and Reporting**". Under this management process, no sub-process is specified or defined.

Process Evaluation

In this process, the **Process Architect** plays a role as an owner. The main aim of this management process is to evaluate the processes of IT services on a regular basis. This process accepts inputs from the process of **Service Review** and provides its output to the process of **Definition of CSI Initiatives**.

In this process, the process owner is responsible for maintaining and managing the process architecture and also ensures that all the processes of services cooperate in a seamless way.

According to the version 3 (V3) of ITIL, following are the five sub-processes which comes under this management process:

1. Process Management support
2. Process Benchmarking
3. Process Maturity Assessment
4. Process Audit
5. Process Control and Review

Definition of CSI Initiatives

In this process, the **CSI Manager** plays a role as an owner. This management process is also called/known as a "**Definition of Improvement Initiatives**".

Definition of CSI Initiatives is a process, which is used for describing the particular initiatives whose aim is to improve the qualities of IT services and processes.

In this process, the CSI Manager (process owner) is accountable for managing and maintaining the CSI registers and also helps in taking the good decisions regarding improvement initiatives.

Under this management process, no sub-process is specified or defined.

Monitoring of CSI Initiatives

In this process, the **CSI Manager** plays a role as an owner. This management process is also called as a "**CSI Monitoring**".

Under this management process, no sub-process is specified or defined.

Advantages of ITIL

Following are the various advantages or benefits of ITIL:
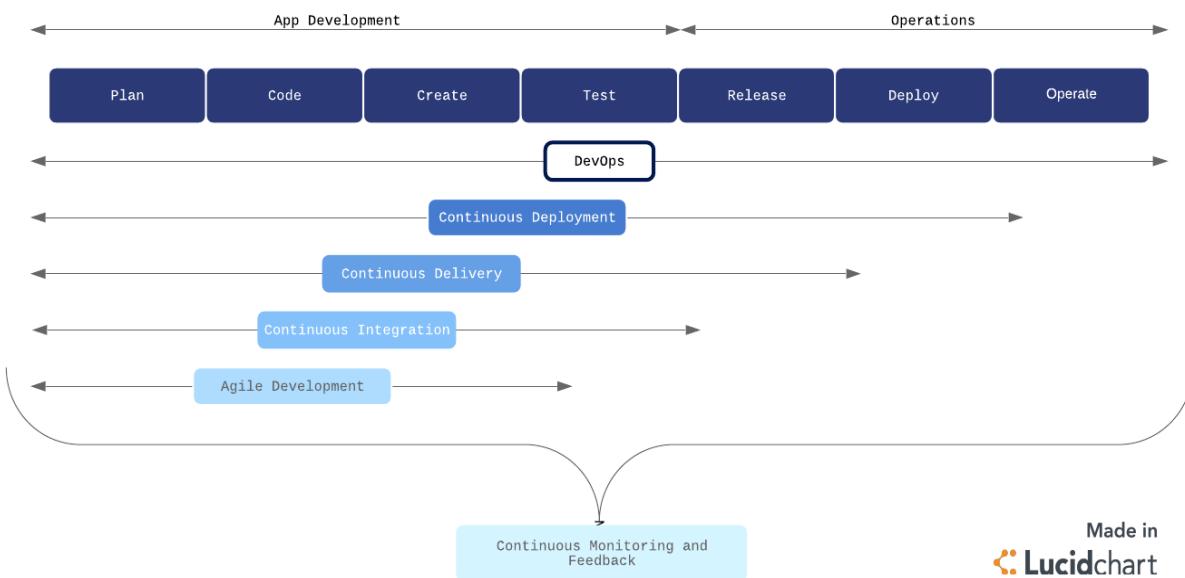
1. One of the best advantages of ITIL is that it helps in increasing the customer satisfaction.
2. It allows managers to improve the decision-making process.
3. It is also used for creating the clear structure of an organization.
4. It also helps managers by controlling the infrastructure services.
5. It improves the interaction between the customers and the service provider.
6. With the help of this framework, service delivery is also improved.
7. It establishes the framework of ITSM for the organization.

DevOps Process

The DevOps process flow

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.

DevOps Process Flow (Click on image to modify

Continuous development

Continuous development is an umbrella term that describes the iterative process for developing software to be delivered to customers. It involves continuous integration, continuous testing, continuous delivery, and continuous deployment.

By implementing a continuous development strategy and its associated sub-strategies, businesses can achieve faster delivery of new features or products that are of higher quality and lower risk, without running into significantly bandwidth barriers.

Continuous integration

Continuous integration (CI) is a software development practice commonly applied in the DevOps process flow. Developers regularly merge their code changes into a shared repository where those updates are automatically tested.

Continuous integration ensures the most up-to-date and validated code is always readily available to developers. CI helps prevent costly delays in development by allowing multiple developers to work on the same source code with confidence, rather than waiting to integrate separate sections of code all at once on release day.

This practice is a crucial component of the DevOps process flow, which aims to combine speed and agility with reliability and security.

Continuous testing

Continuous testing is a verification process that allows developers to ensure the code actually works the way it was intended to in a live environment. Testing can surface bugs and particular aspects of the product that may need fixing or improvement, and can be pushed back to the development stages for continued improvement.

Continuous monitoring and feedback

Throughout the development pipeline, your team should have measures in place for continuous monitoring and feedback of the products and systems. Again, the majority of the monitoring process should be automated to provide continuous feedback.

This process allows IT operations to identify issues and notify developers in real time. Continuous feedback ensures higher security and system reliability as well as more agile responses when issues do arises.

Continuous delivery

Continuous delivery (CD) is the next logical step from CI. Code changes are automatically built, tested, and packaged for release into production. The goal is to release updates to the users rapidly and sustainably.

To do this, CD automates the release process (building on the automated testing in CI) so that new builds can be released at the click of a button.

Continuous deployment

For the seasoned DevOps organization, continuous deployment may be the better option over CD. Continuous deployment is the fully automated version of CD with no human (i.e., manual) intervention necessary.
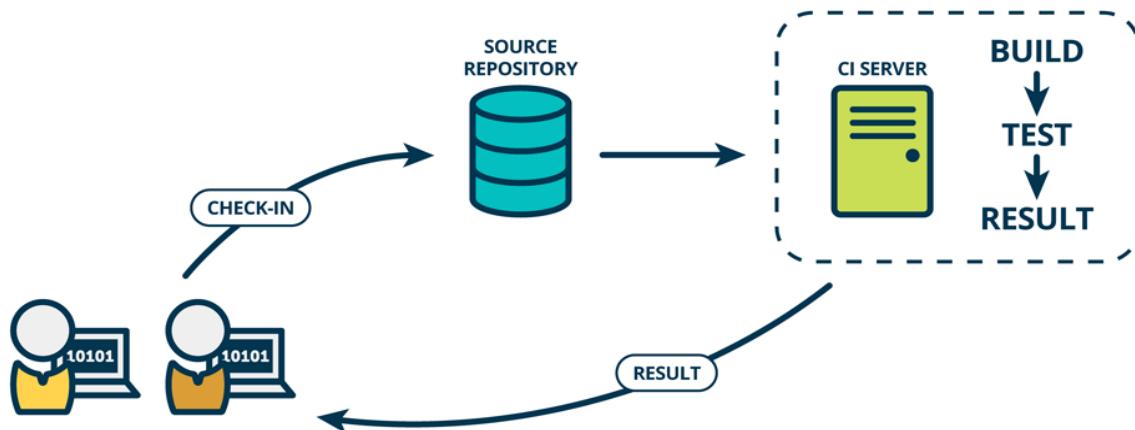
In a continuous deployment process, every validated change is automatically released to users. This process eliminates the need for scheduled release days and accelerates the feedback loop. Smaller, more frequent releases allow developers to get user feedback quickly and address issues with more agility and accuracy.

Continuous deployment is a great goal for a DevOps team, but it is best applied after the DevOps process has been ironed out. For continuous deployment to work well, organizations need to have a rigorous and reliable automated testing environment. If you're not there yet, starting with CI and CD will help you get there.

Continuous Delivery
Continuous delivery is an approach where teams release quality products frequently and predictably from source code repository to production in an automated fashion.

Some organizations release products manually by handing them off from one team to the next, which is illustrated in the diagram below. Typically, developers are at the left end of this spectrum and operations personnel are at the receiving end. This creates delays at every hand-off that leads to frustrated teams and dissatisfied customers. The product eventually goes live through a tedious and error-prone process that delays revenue generation.

How does continuous delivery work?

A continuous delivery pipeline could have a manual gate right before production. A manual gate requires human intervention, and there could be scenarios in your organization that require manual gates in pipelines. Some manual gates might be questionable, whereas some could be legitimate. One legitimate scenario allows the business team to make a last-minute release decision. The engineering team keeps a shippable version of the product ready after every sprint, and the business team makes the final call to release the product to all customers, or a cross-section of the population, or perhaps to people who live in a certain geographical location.

The architecture of the product that flows through the pipeline is a key factor that determines the anatomy of the continuous delivery pipeline. A highly coupled product architecture generates a complicated graphical pipeline pattern where various pipelines could get entangled before eventually making it to production.

The product architecture also influences the different phases of the pipeline and what artifacts are produced in each phase. The pipeline first builds components - the smallest distributable and testable units of the product. For example, a library built by the pipeline can be termed a component. This is the component phase.

Loosely coupled components make up subsystems - the smallest deployable and runnable units. For example, a server is a subsystem. A microservice running in a container is also an example of a subsystem. This is the subsystem phase. As opposed to components, subsystems can be stood up and tested.

The software delivery pipeline is a product in its own right and should be a priority for businesses. Otherwise, you should not send revenue-generating products through it. Continuous delivery adds value in three ways. It improves velocity, productivity, and sustainability of software development teams.

Velocity

Velocity means responsible speed and not suicidal speed. Pipelines are meant to ship quality products to customers. Unless teams are disciplined, pipelines can shoot faulty code to production, only faster! Automated software delivery pipelines help organizations respond to market changes better.

Productivity

A spike in productivity results when tedious tasks, like submitting a change request for every change that goes to production, can be performed by pipelines instead of humans. This lets scrum teams focus on products that wow the world, instead of draining their energy on logistics. And that can make team members happier, more engaged in their work, and want to stay on the team longer.

Sustainability

Sustainability is key for all businesses, not just tech. "Software is eating the world" is no longer true — software has already consumed the world! Every company at the end of the day, whether in healthcare, finance, retail, or some other domain, uses technology to differentiate and outmaneuver their competition. Automation helps reduce/eliminate manual tasks that are error-prone and repetitive, thus positioning the business to innovate better and faster to meet their customers' needs.

Release Management

Release management is the process of overseeing the planning, scheduling, and controlling of software builds throughout each stage of development and across various environments. Release management typically included the testing and deployment of software releases as well.

Release management has had an important role in the software development lifecycle since before it was known as release management. Deciding when and how to release updates was its own unique problem even when software saw physical disc releases with updates occurring as seldom as every few years.

Now that most software has moved from hard and fast release dates to the software as a service (SaaS) business model, release management has become a constant process that works alongside development. This is especially true for businesses that have converted to utilizing continuous delivery pipelines that see new releases occurring at blistering rates. DevOps now plays a large role in many of the duties that were originally considered to be under the purview of release management roles; however, DevOps has not resulted in the obsolescence of release management.

Advantages of Release Management for DevOps

With the transition to DevOps practices, deployment duties have shifted onto the shoulders of the DevOps teams. This doesn't remove the need for release management; instead, it modifies the data points that matter most to the new role release management performs.

Release management acts as a method for filling the data gap in DevOps. The planning of implementation and rollback safety nets is part of the DevOps world, but release management still needs to keep tabs on applications, its components, and the promotion schedule as part of change orders. The key to managing software releases in a way that keeps pace with DevOps deployment schedules is through automated management tools.

Aligning business & IT goals

The modern business is under more pressure than ever to continuously deliver new features and boost their value to customers. Buyers have come to expect that their software evolves and continues to develop innovative ways to meet their needs. Businesses create an outside perspective to glean insights into their customer needs. However, IT has to have an inside perspective to develop these features.

Release management provides a critical bridge between these two gaps in perspective. It coordinates between IT work and business goals to maximize the success of each release. Release management balances customer desires with development work to deliver the greatest value to users.

Minimizes organizational risk

Software products contain millions of interconnected parts that create an enormous risk of failure. Users are often affected differently by bugs depending on their other software, applications, and tools. Plus, faster deployments to production increase the overall risk that faulty code and bugs slip through the cracks.

Release management minimizes the risk of failure by employing various strategies. Testing and governance can catch critical faulty sections of code before they reach the customer. Deployment plans ensure there are enough team members and resources to address any potential issues before affecting users. All dependencies between the millions of interconnected parts are recognized and understood.

Direct accelerating change

Release management is foundational to the discipline and skill of continuously producing enterprise-quality software. The rate of software delivery continues to accelerate and is unlikely to slow down anytime soon. The speed of changes makes release management more necessary than ever.

The move towards CI/CD and increases in automation ensure that the acceleration will only increase. However, it also means increased risk, unmet governance requirements, and potential disorder. Release management helps promote a culture of excellence to scale DevOps to an organizational level.

Release management best practices

As DevOps increases and changes accelerate, it is critical to have best practices in place to ensure that it moves as quickly as possible. Well-refined processes enable DevOps teams to more effectively and efficiently. Some best practices to improve your processes include:

Define clear criteria for success

Well-defined requirements in releases and testing will create more dependable releases. Everyone should clearly understand when things are actually ready to ship.

Well-defined means that the criteria cannot be subjective. Any subjective criteria will keep you from learning from mistakes and refining your release management process to identify what works best. It also needs to be defined for every team member. Release managers, quality supervisors, product vendors, and product owners must all have an agreed-upon set of criteria before starting a project.

Minimize downtime

DevOps is about creating an ideal customer experience. Likewise, the goal of release management is to minimize the amount of disruption that customers feel with updates.

Strive to consistently reduce customer impact and downtime with active monitoring, proactive testing, and real-time collaborative alerts that enable you to quickly notify you of issues during a release. A good release manager will be able to identify any problems before the customer.

The team can resolve incidents quickly and experience a successful release when proactive efforts are combined with a collaborative response plan.

Optimize your staging environment

The staging environment requires constant upkeep. Maintaining an environment that is as close as possible to your production one ensures smoother and more successful releases. From QA to product owners, the whole team must maintain the staging environment by running tests and combing through staging to find potential issues with deployment. Identifying problems in staging before deploying to production is only possible with the right staging environment.
Maintaining a staging environment that is as close as possible to production will enable DevOps teams to confirm that all releases will meet acceptance criteria more quickly.

Strive for immutable

Whenever possible, aim to create new updates as opposed to modifying new ones. Immutable programming drives teams to build entirely new configurations instead of changing existing structures. These new updates reduce the risk of bugs and errors that typically happen when modifying current configurations.

The inherently reliable releases will result in more satisfied customers and employees.

Keep detailed records

Good records management on any release/deployment artifacts is critical. From release notes to binaries to compilation of known errors, records are vital for reproducing entire sets of assets. In most cases, tacit knowledge is required.

Focus on the team

Well-defined and implemented DevOps procedures will usually create a more effective release management structure. They enable best practices for testing and cooperation during the complete delivery lifecycle.
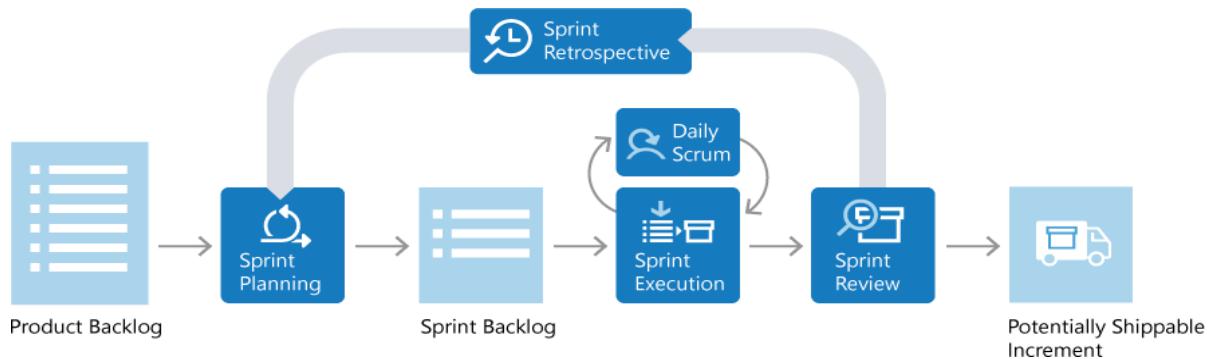
Although automation is a critical aspect of DevOps and release management, it aims to enhance team productivity. The more that release management and DevOps focus on decreasing human error and improving operational efficiency, the more they'll start to quickly release dependable services.

Scrum

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

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 roles

There are three key roles in Scrum: the product owner, the *Scrum master*, and the *Scrum team*.

Product owner

The product owner is responsible for what the team builds, and why they build it. The product owner is responsible for keeping the backlog of work up to date and in priority order.

Scrum master

The Scrum master ensures that the Scrum process is followed by the team. Scrum masters are continually on the lookout for how the team can improve, while also resolving impediments and other blocking issues that arise during the sprint. Scrum masters are part coach, part team member, and part cheerleader.

Scrum team

The members of the Scrum team actually build the product. The team owns the engineering of the product, and the quality that goes with it.

Product backlog

The product backlog is a prioritized list of work the team can deliver. The product owner is responsible for adding, changing, and reprioritizing the backlog as needed. The items at the top of the backlog should always be ready for the team to execute on.

Plan the sprint

In sprint planning, the team chooses backlog items to work on in the upcoming sprint. The team chooses backlog items based on priority and what they believe they can complete in the sprint. The *sprint backlog* is the list of items the team plans to deliver in the sprint. Often, each item on the sprint backlog is broken down into tasks. Once all members agree the sprint backlog is achievable, the sprint starts.

Execute the sprint

Once the sprint starts, the team executes on the sprint backlog. Scrum does not specify how the team should execute. The team decides how to manage its own work.
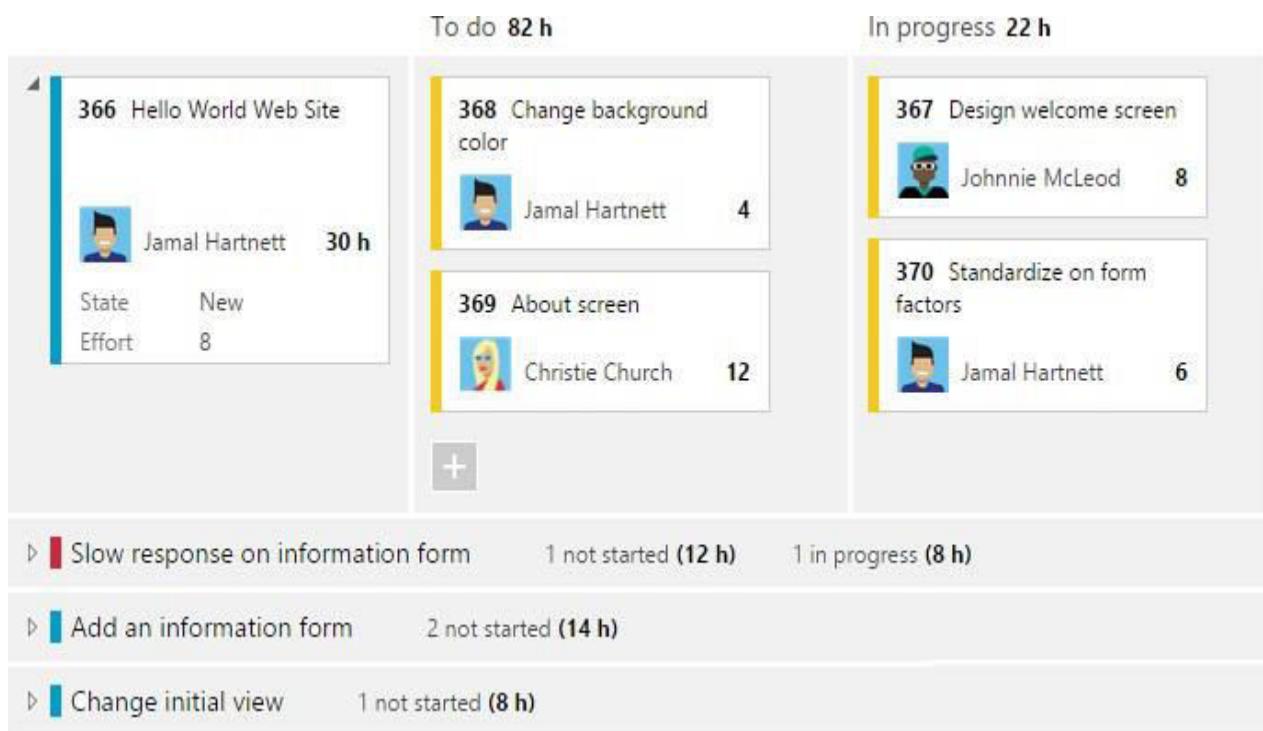
Scrum defines a practice called a *daily Scrum*, often called the *daily standup*. The daily Scrum is a daily meeting limited to fifteen minutes. Team members often stand during the meeting to ensure

it stays brief. Each team member briefly reports their progress since yesterday, the plans for today, and anything impeding their progress.

To aid the daily Scrum, teams often review two artifacts:

Task board

The task board lists each backlog item the team is working on, broken down into the tasks required to complete it. Tasks are placed in **To do**, **In progress**, and **Done** columns based on their status. The board provides a visual way to track the progress of each backlog item.

.

Sprint burndown chart

The sprint burndown is a graph that plots the daily total of remaining work, typically shown in hours. The burndown chart provides a visual way of showing whether the team is on track to complete all the work by the end of the sprint.

Sprint review and sprint retrospective

At the end of the sprint, the team performs two practices:

Sprint review

The team demonstrates what they've accomplished to stakeholders. They demo the software and show its value.

Sprint retrospective

The team takes time to reflect on what went well and which areas need improvement. The outcome of the retrospective are actions for the next sprint.

Increment

The product of a sprint is called the *increment* or *potentially shippable increment*. Regardless of the term, a sprint's output should be of shippable quality, even if it's part of something bigger and can't ship by itself. It should meet all the quality criteria set by the team and product owner.

Repeat, learn, improve

The entire cycle is repeated for the next sprint. Sprint planning selects the next items on the product backlog and the cycle repeats. While the team executes the sprint, the product owner ensures the items at the top of the backlog are ready to execute in the following sprint.

This shorter, iterative cycle provides the team with lots of opportunities to learn and improve. A traditional project often has a long lifecycle, say 6-12 months. While a team can learn from a traditional project, the opportunities are far less than a team who executes in two-week sprints, for example.
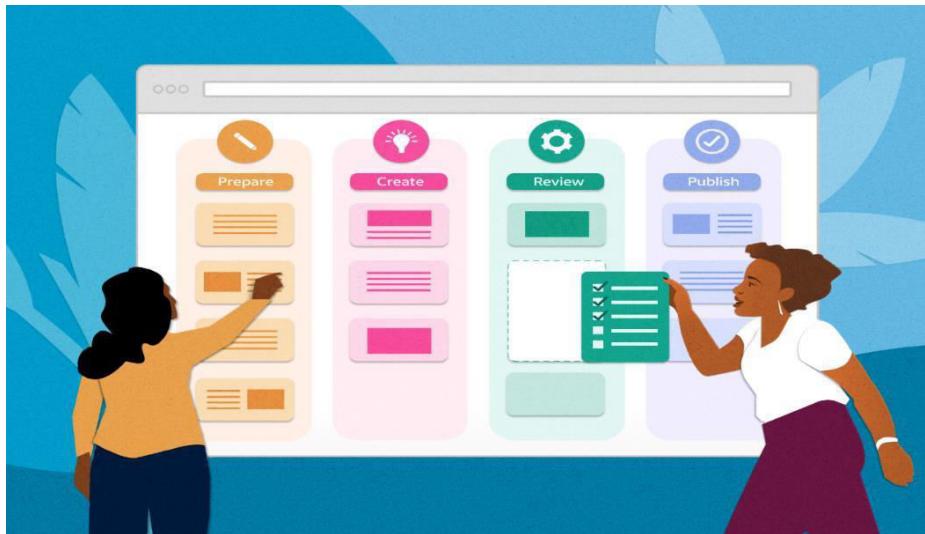
This iterative cycle is, in many ways, the essence of Agile.

Scrum is very popular because it provides just enough framework to guide teams while giving them flexibility in how they execute. Its concepts are simple and easy to learn. Teams can get started quickly and learn as they go. All of this makes Scrum a great choice for teams just starting to implement Agile principles.

Kanban

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.
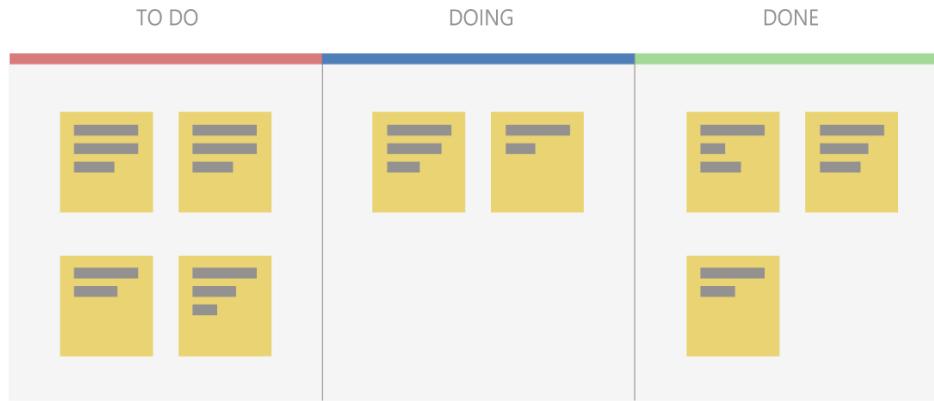

Kanban principles

Adopting Kanban requires adherence to some fundamental practices that might vary from teams' previous methods.

Visualize work

Understanding development team status and work progress can be challenging. Work progress and current state is easier to understand when presented visually rather than as a list of work items or a document.

Visualization of work is a key principle that Kanban addresses primarily through *Kanban boards*. These boards use cards organized by progress to communicate overall status. Visualizing work as cards in different states on a board helps to easily see the big picture of where a project currently stands, as well as identify potential bottlenecks that could affect productivity.

TO DO	DOING	DONE

Use a pull model

Historically, stakeholders requested functionality by pushing work onto development teams, often with tight deadlines. Quality suffered if teams had to take shortcuts to deliver the functionality within the timeframe.

Kanban focuses on maintaining an agreed-upon level of quality that must be met before considering work done. To support this model, stakeholders don't push work on teams that are already working at capacity. Instead, stakeholders add requests to a backlog that a team pulls into their workflow as capacity becomes available.

Impose a WIP limit

Teams that try to work on too many things at once can suffer from reduced productivity due to frequent and costly context switching. The team is busy, but work doesn't get done, resulting in unacceptably high lead times. Limiting the number of backlog items a team can work on at a time helps increase focus while reducing context switching. The items the team is currently working on are called *work in progress (WIP)*.

Teams decide on a *WIP limit*, or maximum number of items they can work on at one time. A well-disciplined team makes sure not to exceed their WIP limit. If teams exceed their WIP limits, they investigate the reason and work to address the root cause.
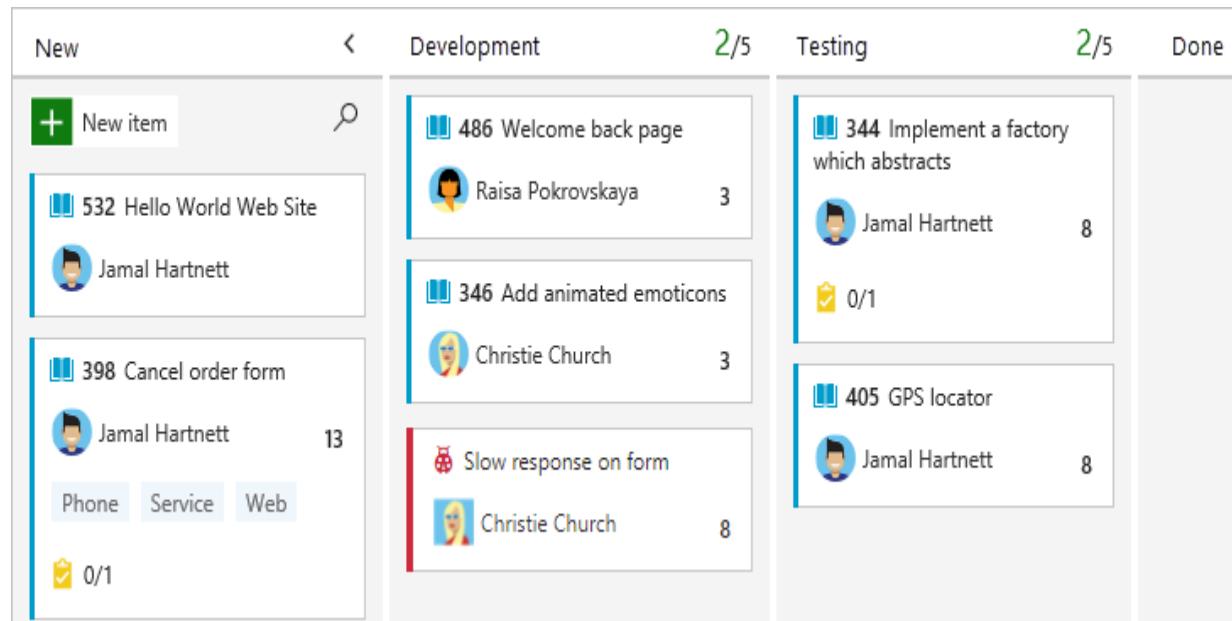
Measure continuous improvement

To practice continuous improvement, development teams need a way to measure effectiveness and throughput. Kanban boards provide a dynamic view of the states of work in a workflow, so teams can experiment with processes and more easily evaluate impact on workflows. Teams that embrace Kanban for continuous improvement use measurements like *lead time* and *cycle time*.

Kanban boards

The *Kanban board* is one of the tools teams use to implement Kanban practices. A Kanban board can be a physical board or a software application that shows cards arranged into columns. Typical column names are **To-do**, **Doing**, and **Done**, but teams can customize the names to match their workflow states. For example, a team might prefer to use **New**, **Development**, **Testing**, **UAT**, and **Done**.

Software development-based Kanban boards display cards that correspond to product backlog items. The cards include links to other items, such as tasks and test cases. Teams can customize the cards to include information relevant to their process.
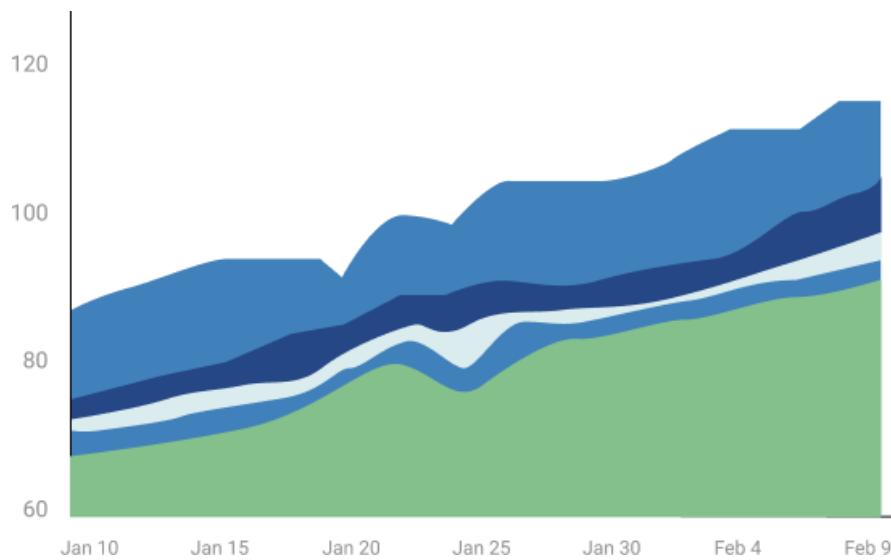

On a Kanban board, the WIP limit applies to all in-progress columns. WIP limits don't apply to the first and last columns, because those columns represent work that hasn't started or is completed. Kanban boards help teams stay within WIP limits by drawing attention to columns that exceed the limits. Teams can then determine a course of action to remove the bottleneck.
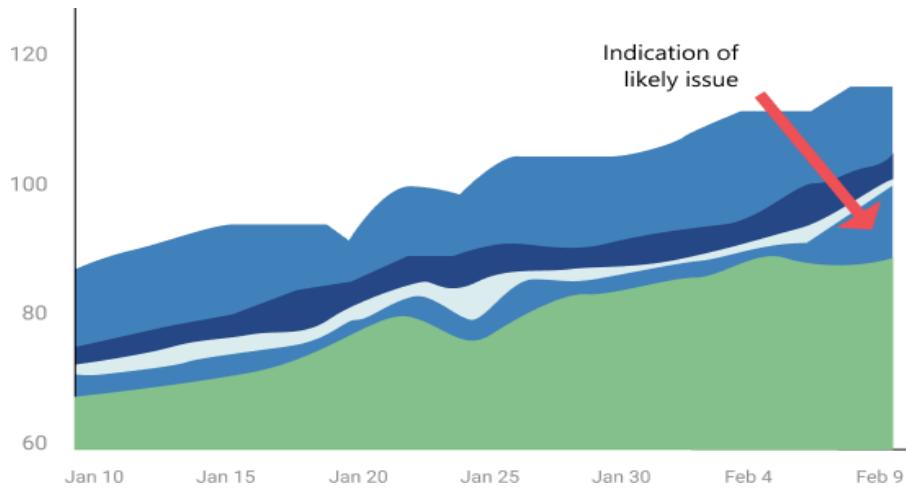
Cumulative flow diagrams

A common addition to software development-based Kanban boards is a chart called a *cumulative flow diagram (CFD)*. The CFD illustrates the number of items in each state over time, typically across several weeks. The horizontal axis shows the timeline, while the vertical axis shows the number of product backlog items. Colored areas indicate the states or columns the cards are currently in.

The CFD is particularly useful for identifying trends over time, including bottlenecks and other disruptions to progress velocity. A good CFD shows a consistent upward trend while a team is working on a project. The colored areas across the chart should be roughly parallel if the team is working within their WIP limits.

A bulge in one or more of the colored areas usually indicates a bottleneck or impediment in the team's flow. In the following CFD, the completed work in green is flat, while the testing state in blue is growing, probably due to a bottleneck.

Kanban and Scrum in Agile development

While broadly fitting under the umbrella of Agile development, Scrum and Kanban are quite different.

- Scrum focuses on fixed length sprints, while Kanban is a continuous flow model.
- Scrum has defined roles, while Kanban doesn't define any team roles.
- Scrum uses velocity as a key metric, while Kanban uses cycle time.

Teams commonly adopt aspects of both Scrum and Kanban to help them work most effectively. Regardless of which characteristics they choose, teams can always review and adapt until they find the best fit. Teams should start simple and not lose sight of the importance of delivering value regularly to users.

Kanban with GitHub

GitHub offers a Kanban experience through project boards (classic). These boards help you organize and prioritize work for specific feature development, comprehensive roadmaps, or release checklists. You can automate project boards (classic) to sync card status with associated issues and pull requests.

Kanban with Azure Boards

Azure Boards provides a comprehensive Kanban solution for DevOps planning. Azure Boards has deep integration across Azure DevOps, and can also be part of Azure Boards-GitHub integration.

- For more information, see Reasons to use Azure Boards to plan and track your work.
- The Learn module Choose an Agile approach to software development provides hands-on Kanban experience in Azure Boards.

Delivery Pipeline
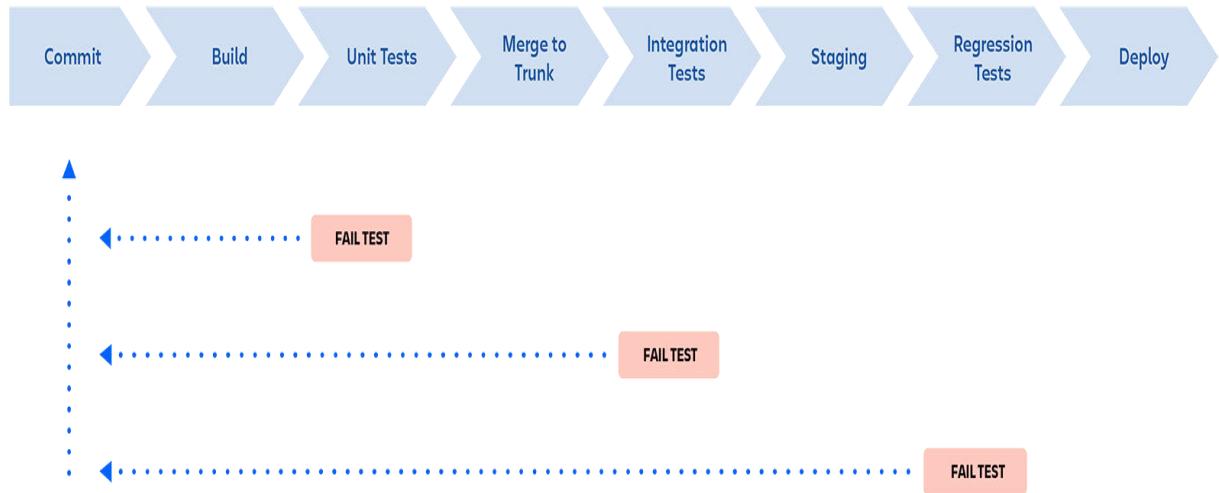
A DevOps pipeline is a set of automated processes and tools that allows both developers and operations professionals to work cohesively to build and deploy code to a production environment. While a DevOps pipeline can differ by organization, it typically includes build automation/continuous integration, automation testing, validation, and reporting. It may also include one or more manual gates that require human intervention before code is allowed to proceed.

Continuous is a differentiated characteristic of a DevOps pipeline. This includes continuous integration, continuous delivery/deployment (CI/CD), continuous feedback, and continuous operations. Instead of one-off tests or scheduled deployments, each function occurs on an ongoing basis.

Considerations for building a DevOps pipeline

Since there isn't one standard DevOps pipeline, an organization's design and implementation of a DevOps pipeline depends on its technology stack, a DevOps engineer's level of experience, budget, and more. A DevOps engineer should have a wide-ranging knowledge of both development and operations, including coding, infrastructure management, system administration, and DevOps toolchains.

Plus, each organization has a different technology stack that can impact the process. For example, if your codebase is node.js, factors include whether you use a local proxy npm registry, whether you download the source code and run `npm install` at every stage in the pipeline, or do it once and generate an artifact that moves through the pipeline. Or, if an application is container-based, you need to decide to use a local or remote container registry, build the container once and move it through the pipeline, or rebuild it at every stage.

While every pipeline is unique, most organizations use similar fundamental components. Each step is evaluated for success before moving on to the next stage of the pipeline. In the event of a failure, the pipeline is stopped, and feedback is provided to the developer.

Components of a DevOps pipeline

1. Continuous integration/continuous delivery/deployment (CI/CD)

Continuous integration is the practice of making frequent commits to a common source code repository. It's continuously integrating code changes into existing code base so that any conflicts between different developer's code changes are quickly identified and relatively easy to remediate. This practice is critically important to increasing deployment efficiency.

We believe that trunk-based development is a requirement of continuous integration. If you are not making frequent commits to a common branch in a shared source code repository, you are not doing continuous integration. If your build and test processes are automated but your developers are working on isolated, long-living feature branches that are infrequently integrated into a shared branch, you are also not doing continuous integration.

Continuous delivery ensures that the "main" or "trunk" branch of an application's source code is always in a releasable state. In other words, if management came to your desk at 4:30 PM on a Friday and said, "We need the latest version released right now," that version could be deployed with the push of a button and without fear of failure.

This means having a pre-production environment that is as close to identical to the production environment as possible and ensuring that automated tests are executed, so that every variable that might cause a failure is identified before code is merged into the main or trunk branch.

46

Continuous deployment entails having a level of continuous testing and operations that is so robust, new versions of software are validated and deployed into a production environment without requiring any human intervention.

This is rare and in most cases unnecessary. It is typically only the unicorn businesses who have hundreds or thousands of developers and have many releases each day that require, or even want to have, this level of automation.

To simplify the difference between continuous delivery and continuous deployment, think of delivery as the FedEx person handing you a box, and deployment as you opening that box and using what's inside. If a change to the product is required between the time you receive the box and when you open it, the manufacturer is in trouble!

3. **Continuous feedback**

4. The single biggest pain point of the old waterfall method of software development — and consequently why agile methodologies were designed — was the lack of timely feedback. When new features took months or years to go from idea to implementation, it was almost guaranteed that the end result would be something other than what the customer expected or wanted. Agile succeeded in ensuring that developers received faster feedback from stakeholders. Now with DevOps, developers receive continuous feedback not not only from stakeholders, but from systematic testing and monitoring of their code in the pipeline.

Continuous testing is a critical component of every DevOps pipeline and one of the primary enablers of continuous feedback. In a DevOps process, changes move continuously from development to testing to deployment, which leads not only to faster releases, but a higher quality product. This means having automated tests throughout your pipeline, including unit tests that run on every build change, smoke tests, functional tests, and end-to-end tests.

Continuous monitoring is another important component of continuous feedback. A DevOps approach entails using continuous monitoring in the staging, testing, and even development environments. It is sometimes useful to monitor pre-production environments for anomalous behavior, but in general this is an approach used to continuously assess the health and performance of applications in production.

Numerous tools and services exist to provide this functionality, and this may involve anything from monitoring your on-premise or cloud infrastructure such as server resources, networking, etc. or the performance of your application or its API interfaces.

3. Continuous operations

Continuous operations is a relatively new and less common term, and definitions vary. One way to interpret it is as "continuous uptime". For example in the case of a blue/green deployment

strategy in which you have two separate production environments, one that is "blue" (publicly accessible) and one that is "green" (not publicly accessible). In this situation, new code would be deployed to the green environment, and when it was confirmed to be functional then a switch would be flipped (usually on a load-balancer) and traffic would switch from the "blue" system to the "green" system. The result is no downtime for the end-users.

Another way to think of Continuous operations is as **continuous alerting**. This is the notion that engineering staff is on-call and notified if any performance anomalies in the application or infrastructure occur. In most cases, continuous alerting goes hand in hand with continuous monitoring.

One of the main goals of DevOps is to improve the overall workflow in the software development life cycle (SDLC). The flow of work is often described as WIP or work in progress. Improving WIP can be accomplished by a variety of means. In order to effectively remove bottlenecks that decrease the flow of WIP, one must first analyze the people, process, and technology aspects of the entire SDLC.

These are the 11 bottlenecks that have the biggest impact on the flow of work.

1. Inconsistent Environments

In almost every company I have worked for or consulted with, a huge amount of waste exists because the various environments (dev, test, stage, prod) are configured differently. I call this "environment hell". How many times have you heard a developer say "it worked on my laptop"? As code moves from one environment to the next, software breaks because of the different configurations within each environment. I have seen teams waste days and even weeks fixing bugs that are due to environmental issues and are not due to errors within the code. Inconsistent environments are the number one killer of agility.

Create standard infrastructure blueprints and implement continuous delivery to ensure all environments are identical.

2. Manual Intervention

Manual intervention leads to human error and non-repeatable processes. Two areas where manual intervention can disrupt agility the most are in testing and deployments. If testing is performed manually, it is impossible to implement continuous integration and continuous delivery in an agile manner (if at all). Also, manual testing increases the chance of producing defects, creating unplanned work. When deployments are performed fully or partially manual, the risk of deployment failure increases significantly which lowers quality and reliability and increases unplanned work.

Automate the build and deployment processes and implement a test automation methodology like test driven development (TDD)

3. SDLC Maturity

The maturity of a team's software development lifecycle (SDLC) has a direct impact on their ability to deliver software. There is nothing new here; SDLC maturity has plagued IT for decades. In the age of DevOps, where we strive to deliver software in shorter increments with a high degree of reliability and quality, it is even more critical for a team to have a mature process.

Some companies I visit are still practicing waterfall methodologies. These companies struggle with DevOps because they don't have any experience with agile. But not all companies that practice agile do it well. Some are early in their agile journey, while others have implemented what I call "Wagile": waterfall tendencies with agile terminology sprinkled in. I have seen teams who have implemented Kanban but struggle with the prioritization and control of WIP. I have seen scrum teams struggle to complete the story points that they promised. It takes time to get really good at agile.

Invest in training and hold blameless post mortems to continously solicit feedback and improve.

4. Legacy Change Management Processes

Many companies have had their change management processes in place for years and are comfortable with it. The problem is that these processes were created back when companies were deploying and updating back office solutions or infrastructure changes that happened infrequently. Fast forward to today's environments where applications are made of many small components or micro services that can be changed and deployed quickly, now all of a sudden the process gets in the way.

Many large companies with well-established ITIL processes struggle with DevOps. In these environments I have seen development teams implement highly automated CI/CD processes only to stop and wait for weekly manual review gates. Sometimes these teams have to go through multiple reviews (security, operations, code, and change control). What is worse is that there is often a long line to wait in for reviews, causing a review process to slip another week. Many of these reviews are just rubber stamp approvals that could be entirely avoided with some minor modifications to the existing processes.

Companies with legacy processes need to look at how they can modernize processes to be more agile instead of being the reason why their company can't move fast enough.

5. Lack of Operational Maturity

Moving to a DevOps model often requires a different approach to operations. Some companies accustomed to supporting back office applications that change infrequently. It requires a different mindset to support software delivered as a service that is always on, and deployed frequently. With DevOps, operations is no longer just something Ops does. Developers now must have tools so they can support applications. Often I encounter companies that only monitor infrastructure. In the DevOps model developers need access to logging solutions, application performance monitoring (APM) tools, web and mobile analytics, advanced alerting and notification solutions. Processes like change management, problem management, request management, incident management, access management, and many others often need to be modernized to allow for more agility and transparency. With DevOps, operations is a team sport.

Assess your operational processes, tools, and organization and modernize to increase agility and transparency.

6. Outdated testing practices

Too often I see clients who have a separate QA department that is not fully integrated with the development team. The code is thrown over the wall and then testing begins. Bugs are detected and sent back to developers who then have to quickly fix, build, and redeploy. This process is repeated until there is no time remaining and teams are left to agree on what defects they can tolerate and promote to production. This is a death spiral in action. With every release, they introduce more technical debt into the system lowering its quality and reliability, and increasing unplanned work. There is a better way.

The better way is to block bugs from moving forward in the development process. This is accomplished by building automated test harnesses and by automatically failing the build if any of the tests fail. This is what continuous integration is designed for. Testing must be part of the development process, not a handoff that is performed after development. Developers need to play a bigger part in testing and testers need to play a bigger part in development. This strikes fear in some testers and not all testers can make the transition.

Quality is everyone's responsibility, not just the QA team.

7. Automating waste

A very common pattern I run into is the automation of waste. This occurs when a team declares itself a DevOps team or a person declares themselves a DevOps engineer and immediately starts writing hundreds or thousands of lines of Chef or Puppet scripts to automate their existing

processes. The problem is that many of the existing processes are bottlenecks and need to be changed. Automating waste is like pouring concrete around unbalanced support beams. It makes bad design permanent.

Automate processes after the bottlenecks are removed.

8. Competing or Misaligned Incentives and Lack of Shared Ownership

This bottleneck has plagued IT for years but is more profound when attempting to be agile. In fact, this issue is at the heart of why DevOps came to be in the first place. Developers are incented for speed to market and operations is incented to ensure security, reliability, availability, and governance. The incentives are conflicting. Instead, everyone should be incented for customer satisfaction, with a high degree of agility, reliability, and quality (which is what DevOps is all about). If every team is not marching towards the same goals, then there will be a never-ending battle of priorities and resources. If all teams' goals are in support of the goals I mentioned above, and everyone is measured in a way that enforces those incentives, then everyone wins --- especially the customer.

Work with HR to help change the reward and incentives to foster the desired behaviors.

9. Dependence on Heroic Efforts

When heroic efforts are necessary to succeed, then a team is in a dark place. This often means working insane hours, being reactive instead of proactive, and being highly reliant on luck and chance. The biggest causes of this are a lack of automation, too much tribal knowledge, immature operational processes, and even poor management. The culture of heroism often leads to burnout, high turnover, and poor customer satisfaction.

If your organization relies on heroes, find out what the root causes are that creates these dependencies and fix them fast.

10. Governance as an Afterthought

When DevOps starts as a grassroots initiative there is typically little attention paid to the question "how does this scale?" It is much easier to show some success in a small isolated team and for an initial project. But once the DevOps initiative starts scaling to larger projects running on way more infrastructures or once it starts spreading to other teams, it can come crashing down without proper governance in place. This is very similar to building software in the cloud. How many times have you seen a small team whip out their credit card and build an amazing solution on AWS? Easy to do, right? Then a year later the costs are spiraling out of control as they lose sight of how many

servers are in use and what is running on them. They all have different versions of third party products and libraries on them. Suddenly, it is not so easy anymore.

With DevOps, the same thing can happen without the appropriate controls in place. Many companies start their DevOps journey with a team of innovators and are able to score some major wins. But when they take that model to other teams it all falls down. There are numerous reasons that this happens. Is the organization ready to manage infrastructure and operations across multiple teams? Are there common shared services available like central logging and monitoring solutions or is each team rolling their own? Is there a common security architecture that everyone can adhere to? Can the teams provision their own infrastructure from a self-service portal or are they all dependent on a single queue ticketing system? I could go on but you get the point. It is easier to cut some corners when there is one team to manage but to scale we must look at the entire service catalog. DevOps will not scale without the appropriate level of governance in place.

Assign an owner and start building a plan for scaling DevOps across the organization.

11. Limited to No Executive Sponsorship

The most successful companies have top level support for their DevOps initiative. One of my clients is making a heavy investment in DevOps training and it will run a large number of employees through the program. Companies with top level support make DevOps a priority. They break down barriers, drive organizational change, improve incentive plans, communicate "Why" they are doing Devops, and fund the initiative. When there is no top level support, DevOps becomes much more challenging and often becomes a new silo. Don't let this stop you from starting a grass roots initiative. Many sponsored initiatives started as grassroots initiatives. These grassroots teams measured their success and pitched their executives. Sometimes when executives see the results and the ROI they become the champions for furthering the cause. My point is, it is hard to get dev and ops to work together with common goals when it is not supported at the highest levels. It is difficult to transform a company to DevOps if it is not supported at the highest levels.

If running a grassroots effort, gather before and after metrics and be prepared to sell and evangelize DevOps upward.

Software Development Life Cycle models and Devops

Software Development Life Cycle models

- **Agile**
- **Lean**
- **Waterfall**
- **Iterative**
- **Spiral**
- **DevOps**

Each of these approaches varies in some ways from the others, but all have a common purpose: to help teams deliver high-quality software as quickly and cost-effectively as possible.

1. Agile

The Agile model first emerged in 2001 and has since become the de facto industry standard. Some businesses value the Agile methodology so much that they apply it to other types of projects, including nontech initiatives.

In the Agile model, fast failure is a good thing. This approach produces ongoing release cycles, each featuring small, incremental changes from the previous release. At each iteration, the product is tested. The Agile model helps teams identify and address small issues on projects before they evolve into more significant problems, and it engages business stakeholders to give feedback throughout the development process.

As part of their embrace of this methodology, many teams also apply an Agile framework known as Scrum to help structure more complex development projects. Scrum teams work in sprints, which usually last two to four weeks, to complete assigned tasks. Daily Scrum meetings help the whole team monitor progress throughout the project. And the ScrumMaster is tasked with keeping the team focused on its goal.

2. Lean

The Lean model for software development is inspired by "lean" manufacturing practices and principles. The seven Lean principles (in this order) are: eliminate waste, amplify learning, decide

as late as possible, deliver as fast as possible, empower the team, build in integrity and see the whole.

The Lean process is about working only on what must be worked on at the time, so there's no room for multitasking. Project teams are also focused on finding opportunities to cut waste at every turn throughout the SDLC process, from dropping unnecessary meetings to reducing documentation.

The Agile model is actually a Lean method for the SDLC, but with some notable differences. One is how each prioritizes customer satisfaction: Agile makes it the top priority from the outset, creating a flexible process where project teams can respond quickly to stakeholder feedback throughout the SDLC. Lean, meanwhile, emphasizes the elimination of waste as a way to create more overall value for customers — which, in turn, helps to enhance satisfaction.

3. Waterfall

Some experts argue that the Waterfall model was never meant to be a process model for real projects. Regardless, Waterfall is widely considered the oldest of the structured SDLC methodologies. It's also a very straightforward approach: finish one phase, then move on to the next. No going back. Each stage relies on information from the previous stage and has its own project plan.

The downside of Waterfall is its rigidity. Sure, it's easy to understand and simple to manage. But early delays can throw off the entire project timeline. With little room for revisions once a stage is completed, problems can't be fixed until you get to the maintenance stage. This model doesn't work well if flexibility is needed or if the project is long-term and ongoing.

Even more rigid is the related Verification and Validation model — or V-shaped model. This linear development methodology sprang from the Waterfall approach. It's characterized by a corresponding testing phase for each development stage. Like Waterfall, each stage begins only after the previous one has ended. This SDLC model can be useful, provided your project has no unknown requirements.

4. Iterative

The Iterative model is repetition incarnate. Instead of starting with fully known requirements, project teams implement a set of software requirements, then test, evaluate and pinpoint further requirements. A new version of the software is produced with each phase, or iteration. Rinse and repeat until the complete system is ready.

Advantages of the Iterative model over other common SDLC methodologies is that it produces a working version of the project early in the process and makes it less expensive to implement changes. One disadvantage: Repetitive processes can consume resources quickly.

One example of an Iterative model is the Rational Unified Process (RUP), developed by IBM's Rational Software division. RUP is a process product, designed to enhance team productivity for a wide range of projects and organizations.

RUP divides the development process into four phases:

- Inception, when the idea for a project is set
- Elaboration, when the project is further defined and resources are evaluated
- Construction, when the project is developed and completed
- Transition, when the product is released

Each phase of the project involves business modeling, analysis and design, implementation, testing, and deployment.

5. Spiral

One of the most flexible SDLC methodologies, Spiral takes a cue from the Iterative model and its repetition. The project passes through four phases (planning, risk analysis, engineering and evaluation) over and over in a figurative spiral until completed, allowing for multiple rounds of refinement.

The Spiral model is typically used for large projects. It enables development teams to build a highly customized product and incorporate user feedback early on. Another benefit of this SDLC model is risk management. Each iteration starts by looking ahead to potential risks and figuring out how best to avoid or mitigate them.

6. DevOps

The DevOps methodology is a relative newcomer to the SDLC scene. It emerged from two trends: the application of Agile and Lean practices to operations work, and the general shift in business toward seeing the value of collaboration between development and operations staff at all stages of the SDLC process.
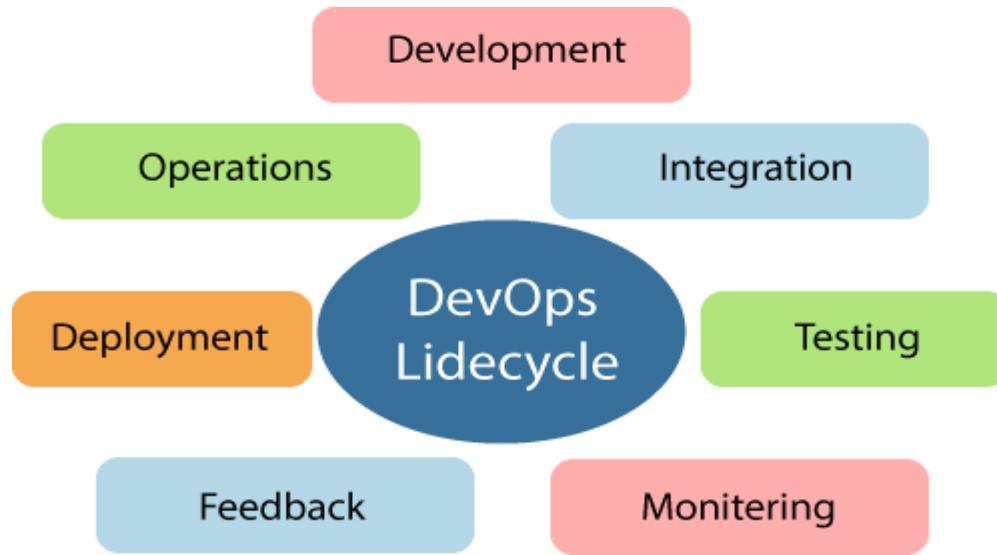
In a DevOps model, Developers and Operations teams work together closely — and sometimes as one team — to accelerate innovation and the deployment of higher-quality and more reliable software products and functionalities. Updates to products are small but frequent. Discipline, continuous feedback and process improvement, and automation of manual development processes are all hallmarks of the DevOps model.

Amazon Web Services describes DevOps as the combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity, evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes. So like many SDLC models, DevOps is not only an approach to planning and executing work, but also a philosophy that demands a nontraditional mindset in an organization.

Choosing the right SDLC methodology for your software development project requires careful thought. But keep in mind that a model for planning and guiding your project is only one ingredient for success. Even more important is assembling a solid team of skilled talent committed to moving the project forward through every unexpected challenge or setback.

DevOps Lifecycle

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

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

1) Continuous Development
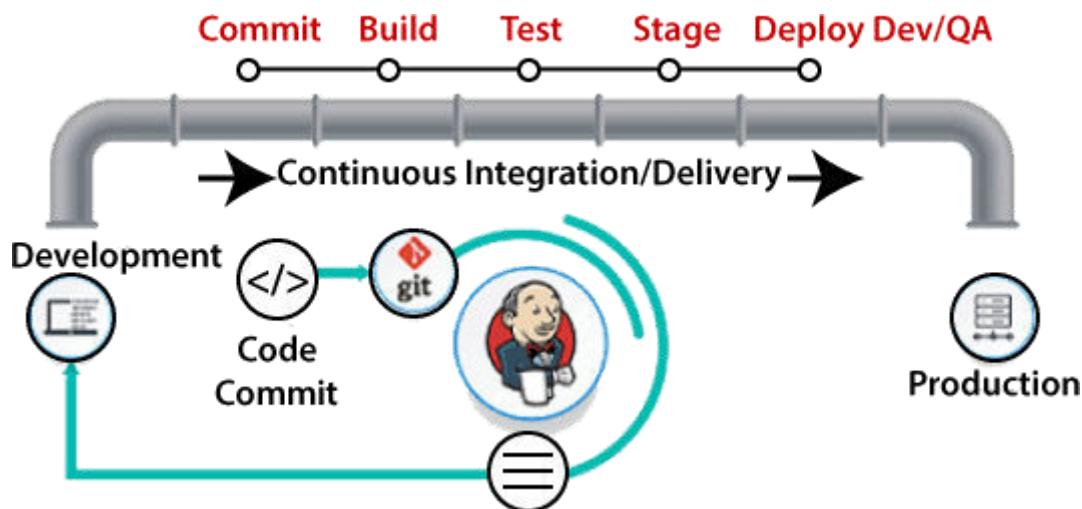
This phase involves the planning and coding of the software. The vision of the project is decided during the planning phase. And the developers begin developing the code for the application. There

are no DevOps tools that are required for planning, but there are several tools for maintaining the code.

2) Continuous Integration

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

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

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

3) Continuous Testing

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

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

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

4) Continuous Monitoring

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

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

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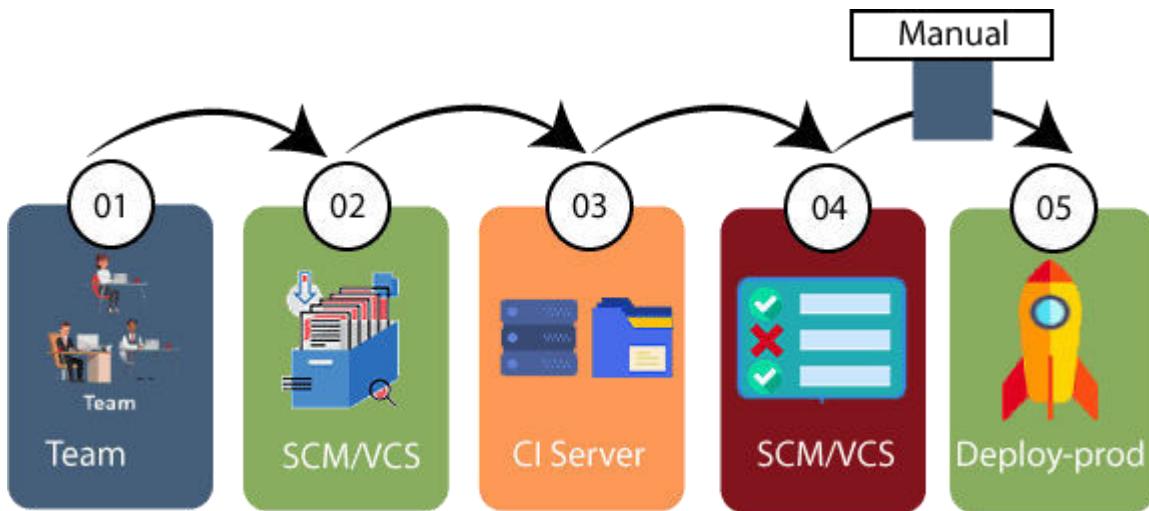
The application development is consistently improved by analyzing the results from the operations of the software. This is carried out by placing the critical phase of constant feedback between the operations and the development of the next version of the current software application.

The continuity is the essential factor in the DevOps as it removes the unnecessary steps which are required to take a software application from development, using it to find out its issues and then

producing a better version. It kills the efficiency that may be possible with the app and reduce the number of interested customers.

6) Continuous Deployment

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

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

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

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

Devops   influence   on   Architecture

Introducing   software   architecture

DevOps Model

The DevOps model goes through several phases governed by cross-discipline teams. Those phases are as follows:

Planning,Identify,andTrack Using the latest in project management tools and agile practices, track ideas and workflows visually. This gives all important stakeholders a clear pathway to prioritization and better results. With better oversight, project managers can ensure teams are on the right track and aware of potential obstacles and pitfalls. All applicable teams can better work together to solve any problems in the development process.

Development Phase Version control systems help developers continuously code, ensuring one patch connects seamlessly with the master branch. Each complete feature triggers the developer to submit a request that, if approved, allows the changes to replace existing code. Development is ongoing.

Testing Phase After a build is completed in development, it is sent to QA testing. Catching bugs is important to the user experience, in DevOps bug testing happens early and often. Practices like continuous integration allow developers to use automation to build and test as a cornerstone of continuous development.

Deployment Phase In the deployment phase, most businesses strive to achieve continuous delivery. This means enterprises have mastered the art of manual deployment. After bugs have been detected and resolved, and the user experience has been perfected, a final team is responsible for the manual deployment. By contrast, continuous deployment is a DevOps approach that automates deployment after QA testing has been completed.

Management Phase During the post-deployment management phase, organizations monitor and maintain the DevOps architecture in place. This is achieved by reading and interpreting data from users, ensuring security, availability and more.

Benefits of DevOps Architecture

A properly implemented DevOps approach comes with a number of benefits. These include the following that we selected to highlight:

Decrease Cost Of primary concern for businesses is operational cost, DevOps helps organizations keep their costs low. Because efficiency gets a boost with DevOps practices, software production increases and businesses see decreases in overall cost for production.

IncreasedProductivity and ReleaseTime With shorter development cycles and streamlined processes, teams are more productive and software is deployed more quickly.

Customers are Served User experience, and by design, user feedback is important to the DevOps process. By gathering information from clients and acting on it, those who practice DevOps ensure that clients wants and needs get honored, and customer satisfaction reaches new highs .

It Gets More Efficient with Time DevOps simplifies the development lifecycle, which in previous iterations had been increasingly complex. This ensures greater efficiency throughout a DevOps organization, as does the fact that gathering requirements also gets easier. In DevOps, requirements gathering is a streamlined process, a culture of accountability, collaboration and transparency makes requirements gathering a smooth going team effort where no stone is left unturned.

The monolithic scenario

Monolithic software is designed to be self-contained, wherein the program's components or functions are tightly coupled rather than loosely coupled, like in modular software programs. In a monolithic architecture, each component and its associated components must all be present for code to be executed or compiled and for the software to run.

Monolithic applications are single-tiered, which means multiple components are combined into one large application. Consequently, they tend to have large codebases, which can be cumbersome to manage over time.

Furthermore, if one program component must be updated, other elements may also require rewriting, and the whole application has to be recompiled and tested. The process can be time-consuming and may limit the agility and speed of software development teams. Despite these issues, the approach is still in use because it does offer some advantages. Also, many early applications were developed as monolithic software, so the approach cannot be completely disregarded when those applications are still in use and require updates.

What is monolithic architecture?

A monolithic architecture is the traditional unified model for the design of a <u>software</u> program. Monolithic, in this context, means "composed all in one piece." According to the Cambridge dictionary, the <u>adjective monolithic</u> also means both "*too large"* and *"unable to be changed.*"

Benefits of monolithic architecture

There are benefits to monolithic architectures, which is why many applications are still created using this development paradigm. For one, monolithic programs may have better throughput than modular applications. They may also be easier to test and debug because, with fewer elements, there are fewer testing variables and scenarios that come into play.

At the beginning of the software development lifecycle, it is usually easier to go with the monolithic architecture since development can be simpler during the early stages. A single codebase also simplifies logging, configuration management, application performance monitoring and other development concerns. Deployment can also be easier by copying the packaged application to a server. Finally, multiple copies of the application can be placed behind a load balancer to scale it horizontally.

That said, the monolithic approach is usually better for simple, lightweight applications. For more complex applications with frequent expected code changes or evolving scalability requirements, this approach is not suitable.

Drawbacks of monolithic architecture

Generally, monolithic architectures suffer from drawbacks that can delay application development and deployment. These drawbacks become especially significant when the product's complexity increases or when the development team grows in size.

The code base of monolithic applications can be difficult to understand because they may be extensive, which can make it difficult for new developers to modify the code to meet changing business or technical requirements. As requirements evolve or become more complex, it becomes difficult to correctly implement changes without hampering the quality of the code and affecting the overall operation of the application.

Following each update to a monolithic application, developers must compile the entire codebase and redeploy the full application rather than just the part that was updated. This makes continuous or regular deployments difficult, which then affects the application's and team's agility.

The application's size can also increase startup time and add to delays. In some cases, different parts of the application may have conflicting resource requirements. This makes it harder to find the resources required to scale the application.

Architecture Rules of Thumb

1. **There is always a bottleneck.** Even in a serverless system or one you think will "infinitely" scale, pressure will always be created elsewhere. For example, if your API scales, does your database also scale? If your database scales, does your email system? In modern cloud systems, there are so many components that scalability is not always the goal. Throttling systems are sometimes the best choice.

2. **Your data model is linked to the scalability of your application.** If your table design is garbage, your queries will be cumbersome, so accessing data will be slow. When designing a database (NoSQL or SQL), carefully consider your access pattern and what data you will have to filter. For example, with DynamoDB, you need to consider what "Key" you will have to retrieve data. If that field is not set as the primary or sort key, it will force you to use a scan rather than a faster query.

3. **Scalability is mainly linked with cost. When you get to a large scale, consider systems where this relationship does not track linearly.** If, like many, you have systems on RDS and ECS; these will scale nicely. But the downside is that as you scale, you will pay directly for that increased capacity. It's common for these workloads to cost $50,000 per month at scale. The solution is to migrate these workloads to serverless systems proactively.

4. **Favour systems that require little tuning to make fast.** The days of configuring your own servers are over. AWS, GCP and Azure all provide fantastic systems that don't need expert knowledge to achieve outstanding performance.

5. **Use infrastructure as code.** Terraform makes it easy to build repeatable and version-controlled infrastructure. It creates an ethos of collaboration and reduces errors by defining them in code rather than "missing" a critical checkbox.

6. **Use a PaaS if you're at less than 100k MAUs.** With Heroku, Fly and Render, there is no need to spend hours configuring AWS and messing around with your application build process. Platform-as-a-service should be leveraged to deploy quickly and focus on the product.

7. **Outsource systems outside of the market you are in. Don't roll your own CMS or Auth, even if it costs you tonnes.** If you go to the pricing page of many third-party systems, for enterprise-scale, the cost is insane - think $10,000 a month for an authentication system! "I could make that in a week," you think. That may be true, but it doesn't consider the long-term maintenance and the time you cannot spend on your core product. Where possible, buy off the shelf.

8. **You have three levers, quality, cost and time. You have to balance them accordingly.** You have, at best, 100 "points" to distribute between the three. Of course, you always want to maintain quality, so the other levers to pull are time and cost.
9. **Design your APIs as open-source contracts.** Leveraging tools such as OpenAPI/Swagger (not a sponsor, just a fan!) allows you to create "contracts" between your front-end and back-end teams. This reduces bugs by having the shape of the request and responses agreed upon ahead of time.
10. **Start with a simple system first (Gall's law).** Galls' law states, "all complex systems that work evolved from simpler systems that worked. If you want to build a complex system that works, build a simpler system first, and then improve it over time.". You should avoid going after shiny technology when creating a new software architecture. Focus on simple, proven systems. S3 for your static website, ECS for your API, RDS for your database, etc. After that, you can chop and change your workload to add these fancy technologies into the mix.

The Separation of Concerns

Separation of concerns is a software architecture design pattern/principle for separating an application into distinct sections, so each section addresses a separate concern. At its essence, Separation of concerns is about order. The overall goal of separation of concerns is to establish a well-organized system where each part fulfills a meaningful and intuitive role while maximizing its ability to adapt to change.

How is separation of concerns achieved

Separation of concerns in software architecture is achieved by the establishment of boundaries. A boundary is any logical or physical constraint which delineates a given set of responsibilities. Some examples of boundaries would include the use of methods, objects, components, and services to define core behavior within an application; projects, solutions, and folder hierarchies for source organization; application layers and tiers for processing organization.

Separation of concerns - advantages

Separation of Concerns implemented in software architecture would have several advantages:

1. Lack of duplication and singularity of purpose of the individual components render the overall system easier to maintain.
2. The system becomes more stable as a byproduct of the increased maintainability.
3. The strategies required to ensure that each component only concerns itself with a single set of cohesive responsibilities often result in natural extensibility points.

4. The decoupling which results from requiring components to focus on a single purpose leads to components which are more easily reused in other systems, or different contexts within the same system.
5. The increase in maintainability and extensibility can have a major impact on the marketability and adoption rate of the system.

There are several flavors of Separation of Concerns. Horizontal Separation, Vertical Separation, Data Separation and Aspect Separation. In this article, we will restrict ourselves to Horizontal and Aspect separation of concern.

Handling database migrations Introduction
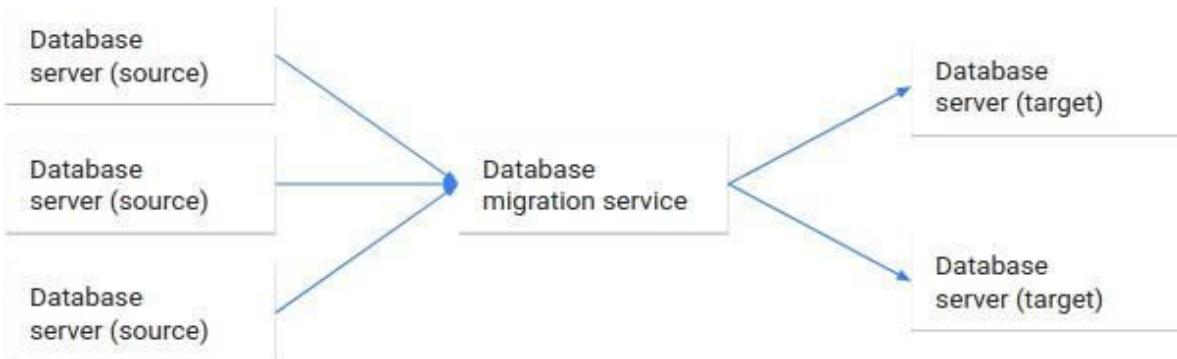
Database schemas define the structure and interrelations of data managed by relational databases. While it is important to develop a well-thought out schema at the beginning of your projects, evolving requirements make changes to your initial schema difficult or impossible to avoid. And since the schema manages the shape and boundaries of your data, changes must be carefully applied to match the expectations of the applications that use it and avoid losing data currently held by the database system.

What are database migrations?

Database migrations, also known as schema migrations, database schema migrations, or simply migrations, are controlled sets of changes developed to modify the structure of the objects within a relational database. Migrations help transition database schemas from their current state to a new desired state, whether that involves adding tables and columns, removing elements, splitting fields, or changing types and constraints.

Migrations manage incremental, often reversible, changes to data structures in a programmatic way. The goals of database migration software are to make database changes repeatable, shareable, and testable without loss of data. Generally, migration software produces artifacts that describe the exact set of operations required to transform a database from a known state to the new state. These can be checked into and managed by normal version control software to track changes and share among team members.

While preventing data loss is generally one of the goals of migration software, changes that drop or destructively modify structures that currently house data can result in deletion. To cope with this, migration is often a supervised process involving inspecting the resulting change scripts and making any modifications necessary to preserve important information.

What are the advantages of migration tools?

Migrations are helpful because they allow database schemas to evolve as requirements change. They help developers plan, validate, and safely apply schema changes to their environments. These compartmentalized changes are defined on a granular level and describe the transformations that must take place to move between various "versions" of the database.

In general, migration systems create artifacts or files that can be shared, applied to multiple database systems, and stored in version control. This helps construct a history of modifications to the database that can be closely tied to accompanying code changes in the client applications. The database schema and the application's assumptions about that structure can evolve in tandem.

Some other benefits include being allowed (and sometimes required) to manually tweak the process by separating the generation of the list of operations from the execution of them. Each change can be audited, tested, and modified to ensure that the correct results are obtained while still relying on automation for the majority of the process.

State based migration

State based migration software creates artifacts that describe how to recreate the desired database state from scratch. The files that it produces can be applied to an empty relational database system to bring it fully up to date.

After the artifacts describing the desired state are created, the actual migration involves comparing the generated files against the current state of the database. This process allows the software to analyze the difference between the two states and generate a new file or files to bring the current database schema in line with the schema described by the files. These change operations are then applied to the database to reach the goal state.

What to keep in mind with state based migrations

Like almost all migrations, state based migration files must be carefully examined by knowledgeable developers to oversee the process. Both the files describing the desired final state and the files that outline the operations to bring the current database into compliance must be reviewed to ensure that the transformations will not lead to data loss. For example, if the generated operations attempt to rename a table by deleting the current one and recreating it with its new name, a knowledgable human must recognize this and intervene to prevent data loss.

State based migrations can feel rather clumsy if there are frequent major changes to the database schema that require this type of manual intervention. Because of this overhead, this technique is often better suited for scenarios where the schema is well-thought out ahead of time with fundamental changes occurring infrequently.

However, state based migrations do have the advantage of producing files that fully describe the database state in a single context. This can help new developers onboard more quickly and works well with workflows in version control systems since conflicting changes introduced by code branches can be resolved easily.

Change based migrations

The major alternative to state based migrations is a change based migration system. Change based migrations also produce files that alter the existing structures in a database to arrive at the desired state. Rather than discovering the differences between the desired database state and the current one, this approach builds off of a known database state to define the operations to bring it into the new state. Successive migration files are produced to modify the database further, creating a series of change files that can reproduce the final database state when applied consecutively.

Because change based migrations work by outlining the operations required from a known database state to the desired one, an unbroken chain of migration files is necessary from the initial starting point. This system requires an initial state, which may be an empty database system or a files describing the starting structure, the files describing the operations that take the schema through each transformation, and a defined order which the migration files must be applied.

What to keep in mind with change based migrations

Change based migrations trace the provenance of the database schema design back to the original structure through the series of transformation scripts that it creates. This can help illustrate the evolution of the database structure, but is less helpful for understanding the complete state of the database at any one point since the changes described in each file modify the structure produced by the last migration file.

Since the previous state is so important to change based systems, the system often uses a database within the database system itself to track which migration files have been applied. This helps the software understand what state the system is currently in without having to analyze the current structure and compare it against the desired state, known only by compiling the entire series of migration files.

The disadvantage of this approach is that the current state of the database isn't described in the code base after the initial point. Each migration file builds off of the previous one, so while the changes are nicely compartmentalized, the entire database state at any one point is much harder to reason about. Furthermore, because the order of operations is so important, it can be more difficult to resolve conflicts produced by developers making conflicting changes.

Change based systems, however, do have the advantage of allowing for quick, iterative changes to the database structure. Instead of the time intensive process of analyzing the current state of the database, comparing it to the desired state, creating files to perform the necessary operations, and applying them to the database, change based systems assume the current state of the database based on the previous changes. This generally makes changes more light weight, but does make out of band changes to the database especially dangerous since migrations can leave the target systems in an undefined state.
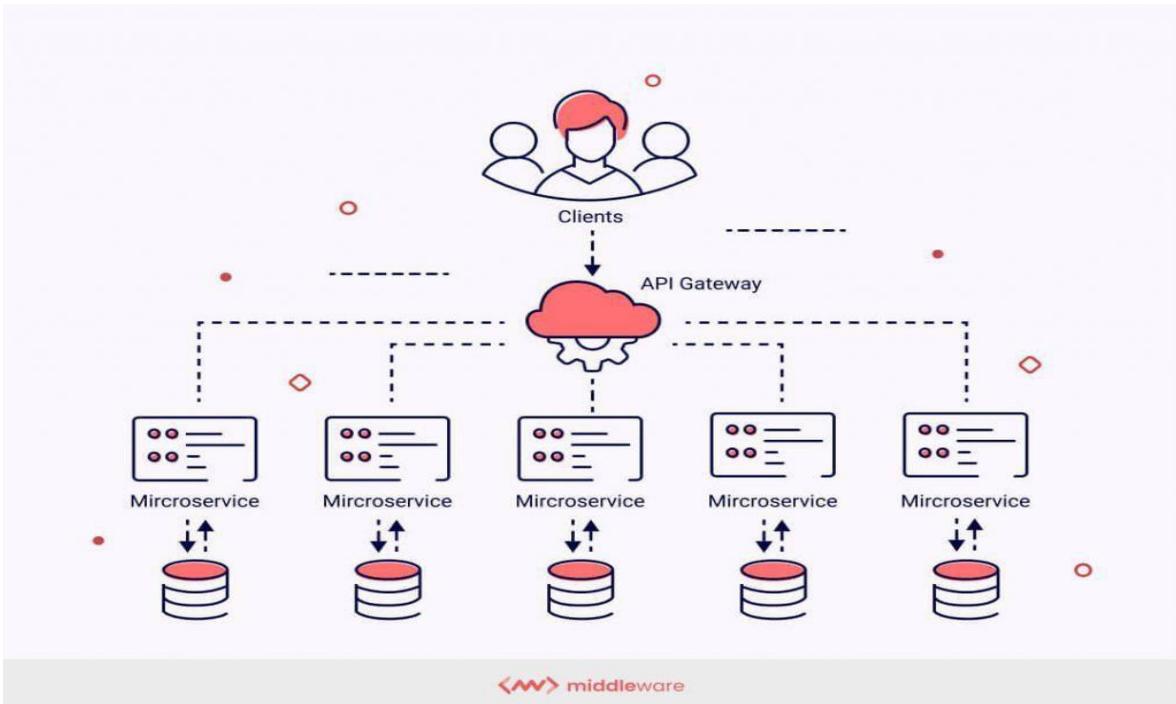
Microservices

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.
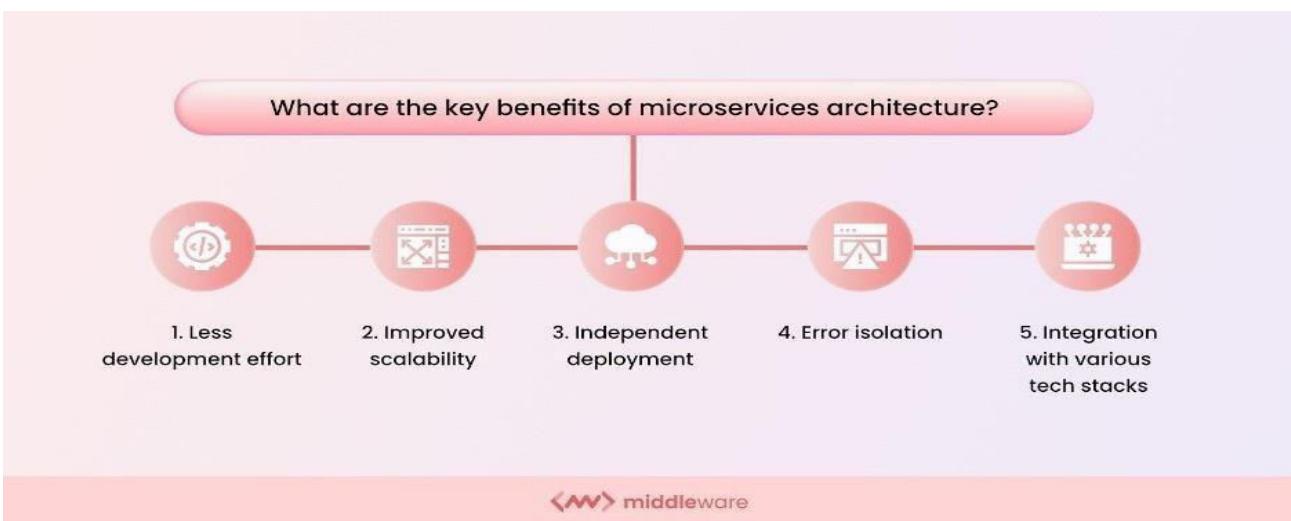
With micro services, however, each unit is independently deployable but can communicate with each other when necessary. Developers can now achieve the scalability, simplicity, and flexibility needed to create highly sophisticated software.

How does microservices architecture work?

The key benefits of microservices architecture

Microservices architecture presents developers and engineers with a number of benefits that monoliths cannot provide. Here are a few of the most notable.

1. Less development effort

Smaller development teams can work in parallel on different components to update existing functionalities. This makes it significantly easier to identify hot services, scale independently from the rest of the application, and improve the application.

2. Improved scalability

Microservices launch individual services independently, developed in different languages or technologies; all tech stacks are compatible, allowing DevOps to choose any of the most efficient tech stacks without fearing if they will work well together. These small services work on relatively less infrastructure than monolithic applications by choosing the precise scalability of selected components per their requirements.

3. Independent deployment

Each microservice constituting an application needs to be a full stack. This enables microservices to be deployed independently at any point. Since microservices are granular in nature, development teams can work on one microservice, fix errors, then redeploy it without redeploying the entire application.

Microservice architecture is agile and thus does not need a congressional act to modify the program by adding or changing a line of code or adding or eliminating features. The software offers to streamline business structures through resilience improvisation and fault separation.

4. Error isolation

In monolithic applications, the failure of even a small component of the overall application can make it inaccessible. In some cases, determining the error could also be tedious. With microservices, isolating the problem-causing component is easy since the entire application is divided into standalone, fully functional software units. If errors occur, other non-related units will still continue to function.

5. Integration with various tech stacks

With microservices, developers have the freedom to pick the tech stack best suited for one particular microservice and its functions. Instead of opting for one standardized tech stack encompassing all of an application's functions, they have complete control over their options.

What is the microservices architecture used for?

Put simply: microservices architecture makes app development quicker and more efficient. Agile deployment capabilities combined with the flexible application of different technologies drastically reduce the duration of the development cycle. The following are some of the most vital applications of microservices architecture.

Data processing

Since applications running on microservice architecture can handle more simultaneous requests, microservices can process large amounts of information in less time. This allows for faster and more efficient application performance.

Media content

Companies like Netflix and Amazon Prime Video handle billions of API requests daily. Services such as OTT platforms offering users massive media content will benefit from deploying a microservices architecture. Microservices will ensure that the plethora of requests for different subdomains worldwide is processed without delays or errors.

Website migration

Website migration involves a substantial change and redevelopment of a website's major areas, such as its domain, structure, user interface, etc. Using microservices will help you avoid business-damaging downtime and ensure your migration plans execute smoothly without any hassles.

Transactions and invoices

Microservices are perfect for applications handling high payments and transaction volumes and generating invoices for the same. The failure of an application to process payments can cause huge losses for companies. With the help of microservices, the transaction functionality can be made more robust without changing the rest of the application.

Microservices tools

Building a microservices architecture requires a mix of tools and processes to perform the core building tasks and support the overall framework. Some of these tools are listed below.

1. Operating system

The most basic tool required to build an application is an operating system (OS). One such operating system allows great flexibility in development and uses in Linux. It offers a largely self-contained environment for executing program codes and a series of options for large and small applications in terms of security, storage, and networking.

2. Programming languages

One of the benefits of using a microservices architecture is that you can use a variety of programming languages across applications for different services. Different programming languages have different utilities deployed based on the nature of the microservice.

3. API management and testing tools

The various services need to communicate when building an application using a microservices architecture. This is accomplished using application programming interfaces (APIs). For APIs to work optimally and desirably, they need to be constantly monitored, managed and tested, and API management and testing tools are essential for this.

4. Messaging tools

Messaging tools enable microservices to communicate both internally and externally. Rabbit MQ and Apache Kafka are examples of messaging tools deployed as part of a microservice system.

5. Toolkits

Toolkits in a microservices architecture are tools used to build and develop applications. Different toolkits are available to developers, and these kits fulfill different purposes. Fabric8 and Seneca are some examples of microservices toolkits.

6. Architectural frameworks

Microservices architectural frameworks offer convenient solutions for application development and usually contain a library of code and tools to help configure and deploy an application.

7. Orchestration tools

A container is a set of executables, codes, libraries, and files necessary to run a microservice. Container orchestration tools provide a framework to manage and optimize containers within microservices architecture systems.

8. Monitoring tools

Once a microservices application is up and running, you must constantly monitor it to ensure everything is working smoothly and as intended. Monitoring tools help developers stay on top of the application's work and avoid potential bugs or glitches.

9. Serverless tools Serverless tools further add flexibility and mobility to the various microservices within an application by eliminating server dependency. This helps in the easier rationalization and division of application tasks.

Microservices vs monolithic architecture

With monolithic architectures, all processes are tightly coupled and run as a single service. This means that if one process of the application experiences a spike in demand, the entire architecture must be scaled. Adding or improving a monolithic application's features becomes more complex as the code base grows. This complexity limits experimentation and makes it difficult to implement new ideas. Monolithic architectures add risk for application availability because many dependent and tightly coupled processes increase the impact of a single process failure.

With a microservices architecture, an application is built as independent components that run each application process as a service. These services communicate via a well-defined interface using lightweight APIs. Services are built for business capabilities and each service performs a single function. Because they are independently run, each service can be updated, deployed, and scaled to meet demand for specific functions of an application.

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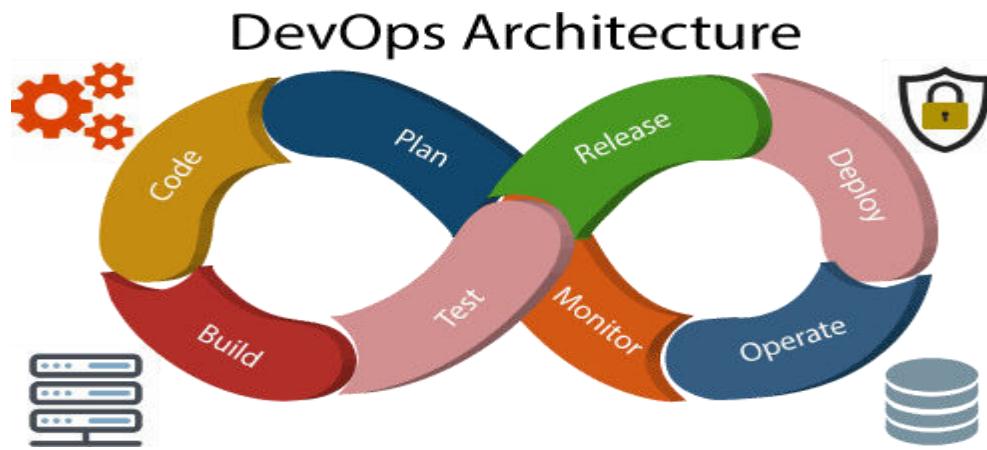
The data tier in DevOps refers to the layer of the application architecture that is responsible for storing, retrieving, and processing data. The data tier is typically composed of databases, data warehouses, and data processing systems that manage large amounts of structured and unstructured data.

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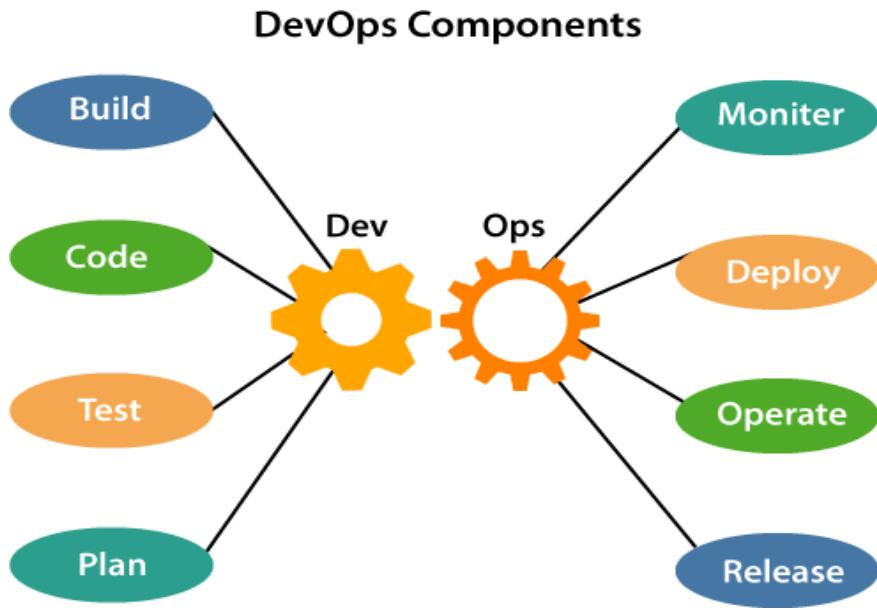
Development and operations both play essential roles in order to deliver applications. The deployment comprises analyzing the **requirements, designing, developing**, and **testing** of the software components or frameworks.

The operation consists of the administrative processes, services, and support for the software. When both the development and operations are combined with collaborating, then the DevOps architecture is the solution to fix the gap between deployment and operation terms; therefore, delivery can be faster.

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

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

DevOps Components

1) Build

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

2) Code

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

3) Test

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

4) Plan

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

5) Monitor

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

6) Deploy

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

7) Operate

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

8) Release

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

DevOps 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.

By focusing on these components, DevOps teams can create a resilient and adaptive DevOps system that is able to deliver high-quality applications and services, even in the face of failures and disruptions.

Unit 3

Introduction to project management

The need for source code control:

Source code control (also known as version control) is an essential part of DevOps practices. Here are a few reasons why:

Collaboration: Source code control allows multiple team members to work on the same codebase simultaneously and track each other's changes.

Traceability: Source code control systems provide a complete history of changes to the code, enabling teams to trace bugs, understand why specific changes were made, and roll back to previous versions if necessary.

Branching and merging: Teams can create separate branches for different features or bug fixes, then merge the changes back into the main codebase. This helps to ensure that different parts of the code can be developed independently, without interfering with each other.

Continuous integration and delivery: Source code control systems are integral to continuous integration and delivery (CI/CD) pipelines, where changes to the code are automatically built, tested, and deployed to production.

In summary, source code control is a critical component of DevOps practices, as it enables teams to collaborate, manage changes to code, and automate the delivery of software.

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.

Today, Git is the most widely used SCM system in the world, and is a critical component of DevOps practices. With the rise of cloud-based platforms, modern SCM systems also offer features like collaboration, code reviews, and integrated issue tracking.

Roles and code in Devops

In DevOps, roles and code play a critical role in the development, delivery, and operation of software.

Roles:

- Development team: responsible for writing and testing code.
- Operations team: responsible for the deployment and maintenance of the code in production.
- DevOps team: responsible for bridging the gap between development and operations, ensuring that code is delivered quickly and reliably to production.

Code:

- Code is the backbone of DevOps and represents the software that is being developed, tested, deployed, and maintained.
- Code is managed using source code control systems like Git, which provide a way to track changes to the code over time, collaborate on the code with other team members, and automate the build, test, and deployment process.
- Code is continuously integrated and tested, ensuring that any changes to the code do not cause unintended consequences in the production environment.

In conclusion, both roles and code play a critical role in DevOps. Teams work together to ensure that code is developed, tested, and delivered quickly and reliably to production, while operations teams maintain the code in production and respond to any issues that arise.

Overall, SCM has been an important part of the evolution of DevOps, enabling teams to collaborate, manage code changes, and automate the software delivery process.

Source code management system and migrations

- 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.

SCM systems provide a number of benefits for managing code migrations, including:

1. **Version control**
2. **Branching and merging**
3. **Rollback**
4. **Collaboration**
5. **Automation**

1) Version control: SCM systems keep a record of all changes to the code, enabling teams to track the code as it moves through different environments.

Purpose of Version Control:
- Multiple people can work simultaneously on a single project. Everyone works on and edits their own copy of the files and it is up to them when they wish to share the changes made by them with the rest of the team.
- It also enables one person to use multiple computers to work on a project, so it is valuable even if you are working by yourself.
- It integrates the work that is done simultaneously by different members of the team. In some rare cases, when conflicting edits are made by two people to the same line of a file, then human assistance is requested by the version control system in deciding what should be done.
- Version control provides access to the historical versions of a project. This is insurance against computer crashes or data loss. If any mistake is made, you can easily roll back to a previous version. It is also possible to undo specific edits that too without losing the work done in the meanwhile. It can be easily known when, why, and by whom any part of a file was edited.

Benefits of the version control system:

- Enhances the project development speed by providing efficient collaboration,
- Leverages the productivity, expedites product delivery, and skills of the employees through better communication and assistance,
- Reduce possibilities of errors and conflicts meanwhile project development through traceability to every small change,
- Employees or contributors of the project can contribute from anywhere irrespective of the different geographical locations through this VCS,

- For each different contributor to the project, a different working copy is maintained and not merged to the main file unless the working copy is validated. The most popular example is Git, Helix core, Microsoft TFS,
- Helps in recovery in case of any disaster or contingent situation,
- Informs us about Who, What, When, Why changes have been made.

Types of Version Control Systems:
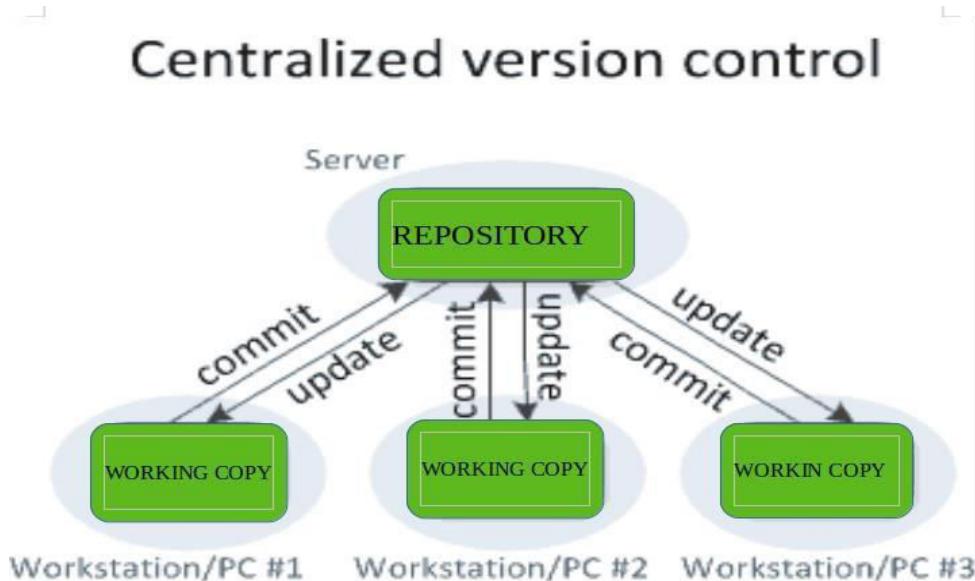
- Local Version Control Systems
- Centralized Version Control Systems
- Distributed Version Control Systems

Local Version Control Systems: It is one of the simplest forms and has a database that kept all the changes to files under revision control. RCS is one of the most common VCS tools. It keeps patch sets (differences between files) in a special format on disk. By adding up all the patches it can then re-create what any file looked like at any point in time.

Centralized Version Control Systems: Centralized version control systems contain just one repository globally and every user need to commit for reflecting one's changes in the repository. It is possible for others to see your changes by updating.

Two things are required to make your changes visible to others which are:

- You commit
- They update

The benefit of CVCS (Centralized Version Control Systems) makes collaboration amongst developers along with providing an insight to a certain extent on what everyone else is doing on the project. It allows administrators to fine-grained control over who can do what.
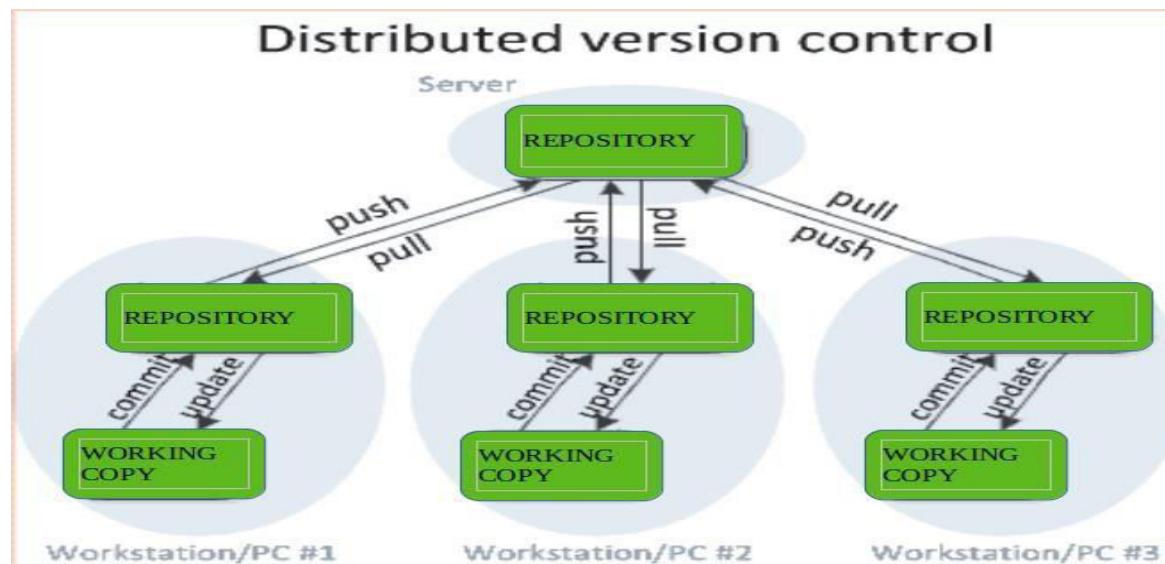
It has some downsides as well which led to the development of DVS. The most obvious is the single point of failure that the centralized repository represents if it goes down during that period collaboration and saving versioned changes is not possible. What if the hard disk of the central database becomes corrupted, and proper backups haven't been kept? You lose absolutely everything.

Distributed Version Control Systems:

Distributed version control systems contain multiple repositories. Each user has their own repository and working copy. Just committing your changes will not give others access to your changes. This is because commit will reflect those changes in your local repository and you need to push them in order to make them visible on the central repository. Similarly, When you update, you do not get others' changes unless you have first pulled those changes into your repository. To make your changes visible to others, 4 things are required:

- You commit
- You push
- They pull
- They update

The most popular distributed version control systems are Git, and Mercurial. They help us overcome the problem of single point of failure.

Distributed version control

2)Branching and merging: Teams can create separate branches of code for different environments, making it easier to manage the migration process.

Branching and merging are key concepts in Git-based version control systems, and are widely used in DevOps to manage the development of software.

Branching in Git allows developers to create a separate line of development for a new feature or bug fix. This allows developers to make changes to the code without affecting the main branch, and to collaborate with others on the same feature or bug fix.

Merging in Git is the process of integrating changes made in one branch into another branch. In DevOps, merging is often used to integrate changes made in a feature branch into the main branch, incorporating the changes into the codebase.

Branching and merging provide several benefits in DevOps:

Improved collaboration: By allowing multiple developers to work on the same codebase at the same time, branching and merging facilitate collaboration and coordination among team members.

Improved code quality: By isolating changes made in a feature branch, branching and merging make it easier to thoroughly review and test changes before they are integrated into the main codebase, reducing the risk of introducing bugs or other issues.

Increased transparency: By tracking all changes made to the codebase, branching and merging provide a clear audit trail of how code has evolved over time.

Overall, branching and merging are essential tools in the DevOps toolkit, helping to improve collaboration, code quality, and transparency in the software development process.

Rollback: In the event of a problem during a migration, teams can quickly revert to a previous version of the code.

Rollback in DevOps refers to the process of reverting a change or returning to a previous version of a system, application, or infrastructure component. Rollback is an important capability in DevOps, as it provides a way to quickly and efficiently revert changes that have unintended consequences or cause problems in production.

There are several approaches to rollback in DevOps, including:

Version control: By using a version control system, such as Git, DevOps teams can revert to a previous version of the code by checking out an earlier commit.

Infrastructure as code: By using infrastructure as code tools, such as Terraform or Ansible, DevOps teams can roll back changes to their infrastructure by re-applying an earlier version of the code.

Continuous delivery pipelines: DevOps teams can use continuous delivery pipelines to automate the rollback process, by automatically reverting changes to a previous version of the code or infrastructure if tests fail or other problems are detected.

Snapshots: DevOps teams can use snapshots to quickly restore an earlier version of a system or infrastructure component.

Overall, rollback is an important capability in DevOps, providing a way to quickly revert changes that have unintended consequences or cause problems in production. By using a combination of version control, infrastructure as code, continuous delivery pipelines, and snapshots, DevOps teams can ensure that their systems and applications can be quickly and easily rolled back to a previous version if needed.

Collaboration: SCM systems enable teams to collaborate on code migrations, with team members working on different aspects of the migration process simultaneously.

Collaboration is a key aspect of DevOps, as it helps to bring together development, operations, and other teams to work together towards a common goal of delivering high-quality software quickly and efficiently.

In DevOps, collaboration is facilitated by a range of tools and practices, including:

Version control systems: By using a version control system, such as Git, teams can collaborate on code development, track changes to source code, and merge code changes from multiple contributors.

Continuous integration and continuous deployment (CI/CD): By automating the build, test, and deployment of code, CI/CD pipelines help to streamline the development process and reduce the risk of introducing bugs or other issues into the codebase.

Code review: By using code review tools, such as pull requests, teams can collaborate on code development, share feedback, and ensure that changes are thoroughly reviewed and tested before they are integrated into the codebase.

Issue tracking: By using issue tracking tools, such as JIRA or GitHub Issues, teams can collaborate on resolving bugs, tracking progress, and managing the development of new features.

Communication tools: By using communication tools, such as Slack or Microsoft Teams, teams can collaborate and coordinate their work, share information, and resolve problems quickly and efficiently.

Overall, collaboration is a critical component of DevOps, helping teams to work together effectively and efficiently to deliver high-quality software. By using a range of tools and practices to facilitate collaboration, DevOps teams can improve the transparency, speed, and quality of their software development processes.

Automation: Many SCM systems integrate with continuous integration and delivery (CI/CD) pipelines, enabling teams to automate the migration process.

In conclusion, SCM systems play a critical role in managing code migrations. They provide a way to track code changes, collaborate on migrations, and automate the migration process, enabling teams to deliver code quickly and reliably to production.

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.

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.

Different Git server implementations

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 requirements.

Docker intermission

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

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

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

Docker vs. Virtual Machines

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

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

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

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

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

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

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

Gerrit

Gerrit is a web based code review tool which is integrated with Git and built on top of Git version control system (helps developers to work together and maintain the history of their work). It allows to merge changes to Git repository when you are done with the code reviews.

Gerrit was developed by *Shawn Pearce* at Google which is written in Java, Servlet, GWT(Google Web Toolkit). The stable release of Gerrit is 2.12.2 and published on March 11, 2016 licensed under *Apache License v2*.

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 commit.

Advantages of Gerrit
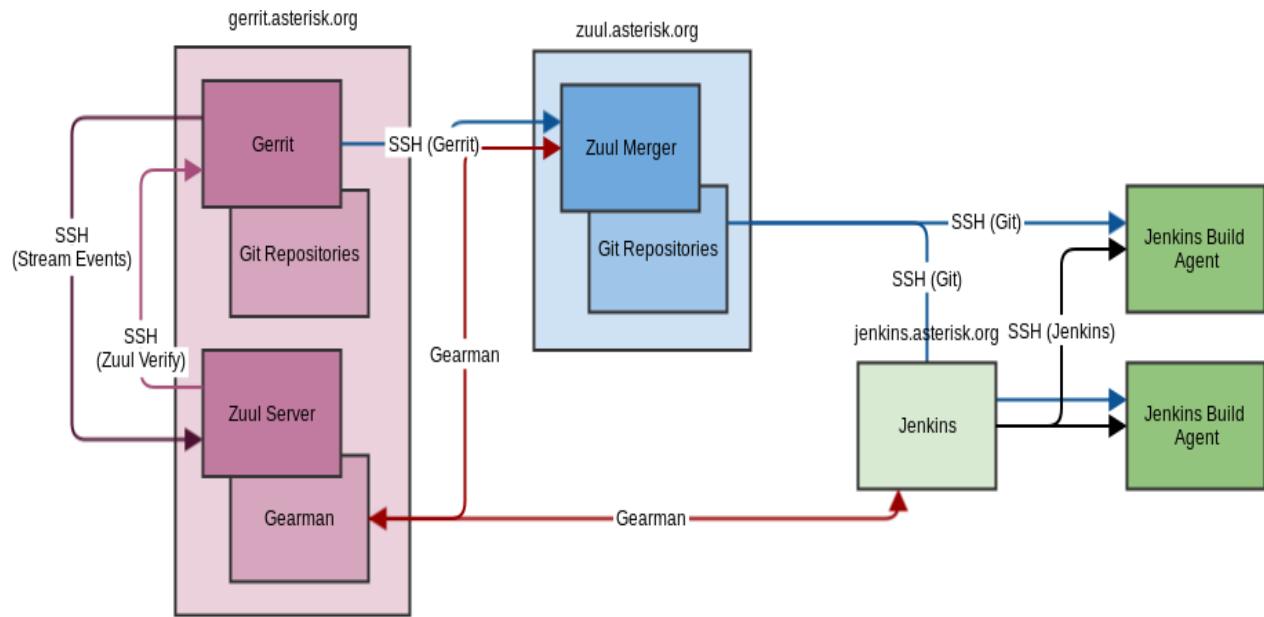
- Gerrit provides access control for Git repositories and web frontend for code review.
- You can push the code without using additional command line tools.
- Gerrit can allow or decline the permission on the repository level and down to the branch level.
- Gerrit is supported by Eclipse.

Disadvantages of Gerrit

- Reviewing, verifying and resubmitting the code commits slows down the time to market.
- Gerrit can work only with Git.
- Gerrit is slow and it's not possible to change the sort order in which changes are listed.
- You need administrator rights to add repository on Gerrit.

What is Gerrit?
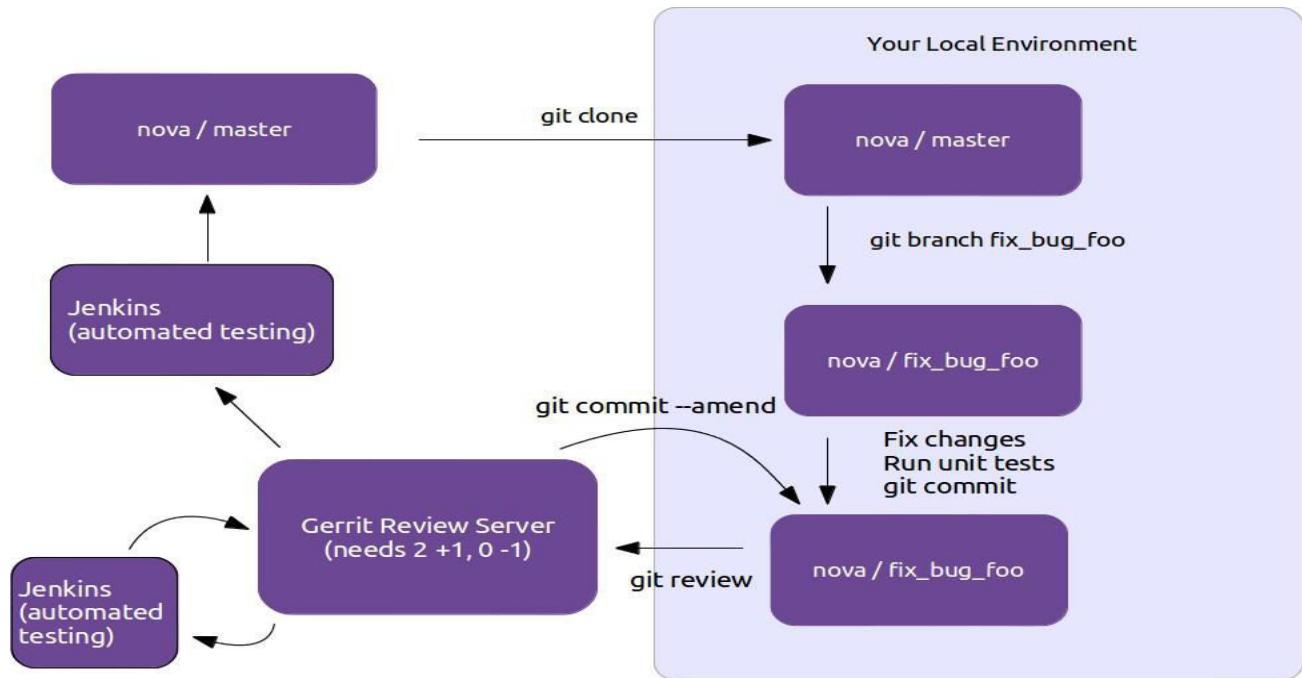
Gerrit is an exceptionally extensible and configurable apparatus for online code survey and storehouse the executives for projects utilizing the Git rendition control framework. Gerrit is similarly helpful where all clients are believed committers, for example, might be the situation with shut source business advancement.

It is used to store the merged code base and the changes under review that have not being merged yet. Gerrit has the limitation of a single repository per project.

Gerrit is first and foremost an arranging region where changes can be looked at prior to turning into a piece of the code base. It is likewise an empowering agent for this survey cycle, catching notes and remarks about the progressions to empower conversation of the change. This is especially valuable with conveyed groups where this discussion can't occur eye to eye.

How Gerrit Works Architecture?

Use case of Gerrit

- Knowledge exchange:

  o The code review process allows newcomers to see the code of other more experienced developers.

  o Developers can get feedback on their suggested changes.

  o Experienced developers can help to evaluate the impact on the whole code.

  o Shared code ownership: by reviewing code of other developers the whole team gets a solid knowledge of the complete code base.

The pull request model

Pull request is a feature of Git-based version control systems that allows developers to propose changes to a Git repository and request feedback or approval f rom other team members. It is widely used in DevOps to facilitate collaboration and code review in the software development process.

In the pull request model, a developer creates a new branch in a Git repository, makes changes to the code, and then opens a pull request to merge the changes into the main branch. Other team members can then review the changes, provide feedback, and approve or reject the request.

Pull Requests are a mechanism popularized by github, used to help facilitate merging of work, particularly in the context of open-source projects. A contributor works on their contribution in a fork (clone) of the central repository. Once their contribution is finished they create a pull request to notify the owner of the central repository that their work is ready to be merged into the mainline. Tooling supports and encourages code review of the contribution before accepting the request. Pull requests have become widely used in software development, but critics are concerned by the addition of integration friction which can prevent continuous integration.

Pull requests essentially provide convenient tooling for a development workflow that existed in many open-source projects, particularly those using a distributed source-control system (such as git). This workflow begins with a contributor creating a new logical branch, either by starting a new branch in the central repository, cloning into a personal repository, or both. The contributor then works on that branch, typically in the style of a Feature Branch, pulling any updates from Mainline into their branch. When they are done they communicate with the maintainer of the central repository indicating that they are done, together with a reference to their commits. This reference could be the URL of a branch that needs to be integrated, or a set of patches in an email.

Once the maintainer gets the message, she can then examine the commits to decide if they are ready to go into mainline. If not, she can then suggest changes to the contributor, who then has opportunity to adjust their submission. Once all is ok, the maintainer can then merge, either with a regular merge/rebase or applying the patches from the final email.

Github's pull request mechanism makes this flow much easier. It keeps track of the clones through its fork mechanism, and automatically creates a message thread to discuss the pull request, together with behavior to handle the various steps in the review workflow. These conveniences were a major part of what made github successful and led to "pull request" becoming a fundamental part of the developer's lexicon.

So that's how pull requests work, but should we use them, and if so how? To answer that question, I like to step back from the mechanism and think about how it works in the context of a source code management workflow. To help me think about that, I wrote down a series of patterns for managing source code branching. I find understanding these (specifically the Base and Integration patterns) clarifies the role of pull requests.
In terms of these patterns, pull requests are a mechanism designed to implement a combination of Feature Branching and Pre-Integration Reviews. Thus to assess the usefulness of pull requests we first need to consider how applicable those patterns are to our situation. Like most patterns,

they are sometimes valuable, and sometimes a pain in the neck - we have to examine them based on our specific context. Feature Branching is a good way of packaging together a logical contribution so that it can be assessed, accepted, or deferred as a single unit. This makes a lot of sense when contributors are not trusted to commit directly to mainline. But Feature Branching comes at a cost, which is that it usually limits the frequency of integration, leading to complicated merges and deterring refactoring. Pre-Integration Reviews provide a clear place to do code review at the cost of a significant increase in integration friction. [1]

That's a drastic summary of the situation (I need a lot more words to explain this further in the feature branching article), but it boils down to the fact that the value of these patterns, and thus the value of pull requests, rest mostly on the social structure of the team. Some teams work better with pull requests, some teams would find pull requests a severe drag on the effectiveness. I suspect that since pull requests are so popular, a lot of teams are using them by default when they would do better without them.

While pull requests are built for Feature Branches, teams can use them within a Continuous Integration environment. To do this they need to ensure that pull requests are small enough, and the team responsive enough, to follow the CI rule of thumb that everybody does Mainline Integration at least daily. (And I should remind everyone that Mainline Integration is more than just merging the current mainline into the feature branch). Using the ship/show/ask classification can be an effective way to integrate pull requests into a more CI-friendly workflow.
The wide usage of pull requests has encouraged a wider use of code review, since pull requests provide a clear point for Pre-Integration Review, together with tooling that encourages it. Code review is a Good Thing, but we must remember that a pull request isn't the only mechanism we can use for it. Many teams find great value in the continuous review afforded by Pair Programming. To avoid reducing integration frqency we can carry out post-integration code review in several ways. A formal process can record a review for each commit, or a tech lead can examine risky commits every couple of days. Perhaps the most powerful form of code review is one that's frequently ignored. A team that takes the attitude that the codebase is a fluid system, one that can be steadily refined with repeated iteration carries out Refinement Code Review every time a developer looks at existing code. I often hear people say that pull requests are necessary because without them you can't do code reviews - that's rubbish. Pre-integration code review is just one way to do code reviews, and for many teams it isn't the best choice.

The pull request model provides several benefits in DevOps:

Improved code quality: Pull requests encourage collaboration and code review, helping to catch potential bugs and issues before they make it into the main codebase.

Increased transparency: Pull requests provide a clear audit trail of all changes made to the code, making it easier to understand how code has evolved over time.

Better collaboration: Pull requests allow developers to share their work and get feedback from others, improving collaboration and communication within the development team.

Overall, the pull request model is an important tool in the DevOps toolkit, helping to improve the quality, transparency, and collaboration of software development processes.

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.

What is Git?

Git is a distributed version control system, which means that a local clone of the project is a complete version control repository. These fully functional local repositories make it easy to work offline or remotely. Developers commit their work locally, and then sync their copy of the repository with the copy on the server. This paradigm differs from centralized version control where clients must synchronize code with a server before creating new versions of code.

Git's flexibility and popularity make it a great choice for any team. Many developers and college graduates already know how to use Git. Git's user community has created resources to train developers and Git's popularity make it easy to get help when needed. Nearly every development environment has Git support and Git command line tools implemented on every major operating system.

Git basics

Every time work is saved, Git creates a commit. A commit is a snapshot of all files at a point in time. If a file hasn't changed from one commit to the next, Git uses the previously stored file. This
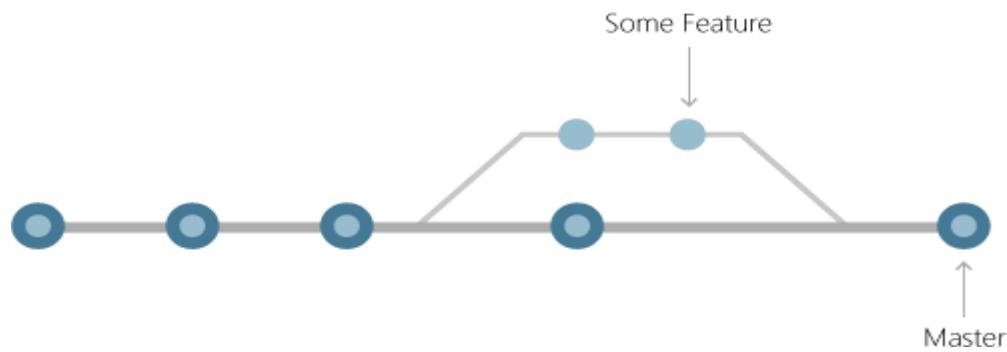
design differs from other systems that store an initial version of a file and keep a record of deltas over time.

Master

Commits create links to other commits, forming a graph of the development history. It's possible to revert code to a previous commit, inspect how files changed from one commit to the next, and review information such as where and when changes were made. Commits are identified in Git by a unique cryptographic hash of the contents of the commit. Because everything is hashed, it's impossible to make changes, lose information, or corrupt files without Git detecting it.

Branches
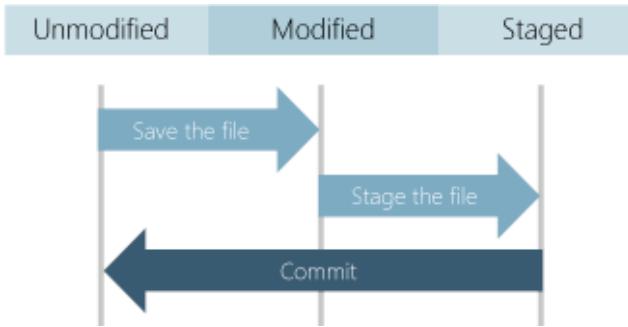
Each developer saves changes to their own local code repository. As a result, there can be many different changes based off the same commit. Git provides tools for isolating changes and later merging them back together. Branches, which are lightweight pointers to work in progress, manage this separation. Once work created in a branch is finished, it can be merged back into the team's main (or trunk) branch.

Some Feature

Master

Files and commits

Files in Git are in one of three states: modified, staged, or committed. When a file is first modified, the changes exist only in the working directory. They aren't yet part of a commit or the

development history. The developer must *stage* the changed files to be included in the commit. The staging area contains all changes to include in the next commit. Once the developer is happy with the staged files, the files are packaged as a *commit* with a message describing what changed. This commit becomes part of the development history.

Staging lets developers pick which file changes to save in a commit in order to break down large changes into a series of smaller commits. By reducing the scope of commits, it's easier to review the commit history to find specific file changes.

Benefits of Git

The benefits of Git are many.

Simultaneous development

Everyone has their own local copy of code and can work simultaneously on their own branches. Git works offline since almost every operation is local.

Faster releases

Branches allow for flexible and simultaneous development. The main branch contains stable, high-quality code from which you release. Feature branches contain work in progress, which are merged into the main branch upon completion. By separating the release branch from development in progress, it's easier to manage stable code and ship updates more quickly.

Built-in integration

Due to its popularity, Git integrates into most tools and products. Every major IDE has built-in Git support, and many tools support continuous integration, continuous deployment, automated testing, work item tracking, metrics, and reporting feature integration with Git. This integration simplifies the day-to-day workflow.

Strong community support

Git is open-source and has become the de facto standard for version control. There is no shortage of tools and resources available for teams to leverage. The volume of community support for Git compared to other version control systems makes it easy to get help when needed.

Git works with any team

Using Git with a source code management tool increases a team's productivity by encouraging collaboration, enforcing policies, automating processes, and improving visibility and traceability of work. The team can settle on individual tools for version control, work item tracking, and continuous integration and deployment. Or, they can choose a solution like GitHub or Azure DevOps that supports all of these tasks in one place.

Pull requests

Use pull requests to discuss code changes with the team before merging them into the main branch. The discussions in pull requests are invaluable to ensuring code quality and increase knowledge across your team. Platforms like GitHub and Azure DevOps offer a rich pull request experience where developers can browse file changes, leave comments, inspect commits, view builds, and vote to approve the code.

Branch policies

Teams can configure GitHub and Azure DevOps to enforce consistent workflows and process across the team. They can set up branch policies to ensure that pull requests meet requirements before completion. Branch policies protect important branches by preventing direct pushes, requiring reviewers, and ensuring clean builds.

Unit 4

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.

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

Compilation: The build system compiles the source code into a machine-executable format, such as a binary or an executable jar file.

Dependency Management: The build system ensures that all required dependencies are available and properly integrated into the build artifact. This can include external libraries, components, and other resources needed to run the application.

Testing: The build system runs automated tests to ensure that the code is functioning as intended, and to catch any issues early in the development process.

Packaging: The build system packages the compiled code and its dependencies into a single, deployable artifact, such as a Docker image or a tar archive.

Version Control: The build system integrates with version control systems, such as Git, to track changes to the code and manage releases.

Continuous Integration: The build system can be configured to run builds automatically whenever changes are made to the code, allowing for fast feedback and continuous integration of new code into the main branch.

Deployment: The build system can be integrated with deployment tools and processes to automate the deployment of the build artifact to production environments.

In DevOps, it's important to have a build system that is fast, reliable, and scalable, and that can integrate with other tools and processes in the software development and delivery pipeline. There are many build systems available, each with its own set of features and capabilities, and choosing the right one will depend on the specific needs of the project and team.

Jenkins build server

What is Jenkin?

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

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

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

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

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

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

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

Possible steps executed by Jenkins are for example:
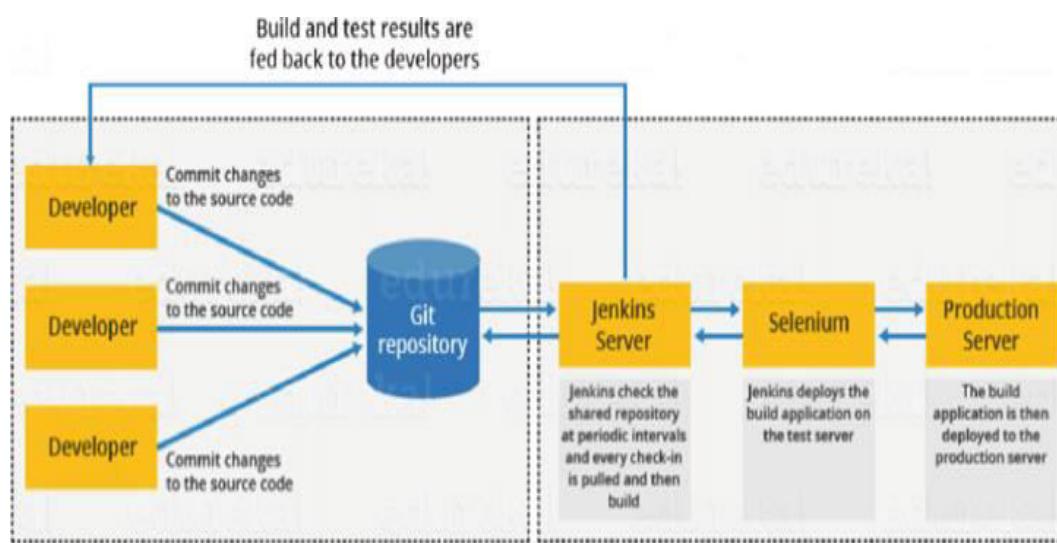
- o Perform a software build using a build system like Gradle or Maven Apache
- o Execute a shell script
- o Archive a build result
- o Running software tests

Jenkin workflow

Build and test results are fed back to the developers

Jenkins Master-Slave Architecture

As you can see in the diagram provided above, on the left is the Remote source code repository. The Jenkins server accesses the master environment on the left side and the master environment can push down to multiple other Jenkins Slave environments to distribute the workload.

That lets you run multiple builds, tests, and product environment across the entire architecture. Jenkins Slaves can be running different build versions of the code for different operating systems and the server Master controls how each of the builds operates.

Supported on a master-slave architecture, Jenkins comprises many slaves working for a master. This architecture - the Jenkins Distributed Build - can run identical test cases in different environments. Results are collected and combined on the master node for monitoring.

Jenkins Applications

Jenkins helps to automate and accelerate the software development process. Here are some of the most common applications of Jenkins:

1. Increased Code Coverage

Code coverage is determined by the number of lines of code a component has and how many of them get executed. Jenkins increases code coverage which ultimately promotes a transparent development process among the team members.

2. No Broken Code

Jenkins ensures that the code is good and tested well through continuous integration. The final code is merged only when all the tests are successful. This makes sure that no broken code is shipped into production.

What are the Jenkins Features?

Jenkins offers many attractive features for developers:

- **Easy Installation**

  Jenkins is a platform-agnostic, self-contained Java-based program, ready to run with packages for Windows, Mac OS, and Unix-like operating systems.

- **Easy Configuration**

  Jenkins is easily set up and configured using its web interface, featuring error checks and a built-in help function.

- **Available Plugins**

  There are hundreds of plugins available in the Update Center, integrating with every tool in the CI and CD toolchain.

- **Extensible**

  Jenkins can be extended by means of its plugin architecture, providing nearly endless possibilities for what it can do.

- **Easy Distribution**

  Jenkins can easily distribute work across multiple machines for faster builds, tests, and deployments across multiple platforms.

- **Free Open Source**

  Jenkins is an open-source resource backed by heavy community support.

As a part of our learning about what is Jenkins, let us next learn about the Jenkins architecture.

Jenkins build server

Jenkins is a popular open-source automation server that helps developers automate parts of the software development process. A Jenkins build server is responsible for building, testing, and deploying software projects.

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.

The build process in Jenkins typically starts with code being committed to a version control system (such as Git), which triggers a build on the Jenkins server. The Jenkins server then checks out the code, builds it, runs tests on it, and if everything is successful, deploys the code to a staging or production environment.

Jenkins has a large community of developers who have created hundreds of plugins that extend its functionality, so it's easy to find plugins to support specific tools, technologies, and workflows. For example, there are plugins for integrating with cloud infrastructure, running security scans, deploying to various platforms, and more.

Overall, a Jenkins build server can greatly improve the efficiency and reliability of the software development process by automating repetitive tasks, reducing the risk of manual errors, and enabling developers to focus on writing code.

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

Here are some common practices for managing build dependencies in Jenkins:

Dependency Management Tools: Utilize tools such as Maven, Gradle, or npm to manage dependencies and automate the process of downloading and installing required dependencies for a build.

Version Pinning: Specify exact versions of dependencies to ensure builds are consistent and repeatable.

Caching: Cache dependencies locally on the build server to improve build performance and reduce the time it takes to download dependencies.

Continuous Monitoring: Regularly check for updates and security vulnerabilities in dependencies to ensure the build environment is secure and up-to-date.

Automated Testing: Automated testing can catch issues related to dependencies early in the development process.

By following these practices, you can effectively manage build dependencies and maintain the reliability and consistency of your CI/CD pipeline.

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.

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.

Git Plugin

The Git Plugin is a popular plugin for Jenkins that integrates the Jenkins automation server with the Git version control system. This plugin allows you to pull code changes from a Git repository, build and test the code, and deploy it to production.

With the Git Plugin, you can configure Jenkins to automatically build and test your code whenever changes are pushed to the Git repository. You can also configure it to build and test code on a schedule, such as once a day or once a week.

The Git Plugin provides a number of features for managing code changes, including:

Branch and Tag builds: You can configure Jenkins to build specific branches or tags from your Git repository.

Pull Requests: You can configure Jenkins to build and test pull requests from your Git repository, allowing you to validate code changes before merging them into the main branch.

Build Triggers: You can configure Jenkins to build and test code changes whenever changes are pushed to the Git repository or on a schedule.

Code Quality Metrics: The Git Plugin integrates with tools such as SonarQube to provide code quality metrics, allowing you to track and improve the quality of your code over time.

Notification and Reporting: The Git Plugin provides notifications and reports on build status, failures, and other important events. You can configure Jenkins to send notifications via email, Slack, or other communication channels.

By using the Git Plugin, you can streamline your software development process and make it easier to manage code changes and collaborate with other developers on your team.

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

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.

When running Jenkins on a network or cloud-based infrastructure, the host server is responsible for providing resources for the Jenkins process, but the setup, configuration, and maintenance may be managed by other components of the infrastructure.

By providing the necessary resources and ensuring the stability and reliability of the host server, you can ensure the efficient operation of Jenkins and the success of your software development and deployment processes.

To host a server in Jenkins, you'll need to follow these steps:

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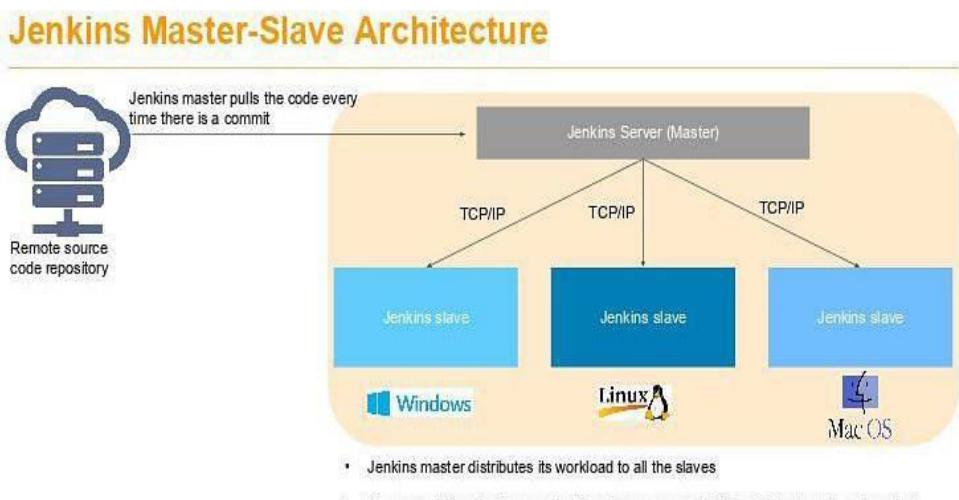
Create a Build Job: To build your project, you'll need to create a build job in Jenkins. This will define the steps involved in building your project, such as checking out the code from version control, compiling the code, running tests, and packaging the application.

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

Jenkins Master-Slave Architecture

As you can see in the diagram provided above, on the left is the Remote source code repository. The Jenkins server accesses the master environment on the left side and the master environment can push down to multiple other Jenkins Slave environments to distribute the workload.

That lets you run multiple builds, tests, and product environment across the entire architecture. Jenkins Slaves can be running different build versions of the code for different operating systems and the server Master controls how each of the builds operates.

Supported on a master-slave architecture, Jenkins comprises many slaves working for a master. This architecture - the Jenkins Distributed Build - can run identical test cases in different environments. Results are collected and combined on the master node for monitoring.

The standard Jenkins installation includes Jenkins master, and in this setup, the master will be managing all our build system's tasks. If we're working on a number of projects, we can run numerous jobs on each one. Some projects require the use of specific nodes, which necessitates the use of slave nodes.

The Jenkins master is in charge of scheduling jobs, assigning slave nodes, and sending builds to slave nodes for execution. It will also keep track of the slave node state (offline or online), retrieve build results from slave nodes, and display them on the terminal output. In most installations, multiple slave nodes will be assigned to the task of building jobs.
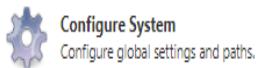
Before we get started, **let's double-check that we have all of the prerequisites in place for adding a slave node**:

- **Jenkins Server** is up and running and ready to use
- Another server for a slave node configuration
- The Jenkins server and the slave server are both connected to the same network

To configure the Master server, we'll log in to the Jenkins server and follow the steps below.

First, we'll go to **"Manage Jenkins -> Manage Nodes -> New Node" to create a new node**:

System Configuration

Configure System
Configure global settings and paths.

Global Tool Configuration
Configure tools, their locations and automatic installers.

Manage Plugins
Add, remove, disable or enable plugins that can extend the functionality of Jenkins.

Manage Nodes and Clouds
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

Install as Windows Service
Installs Jenkins as a Windows service to this system, so that Jenkins starts automatically when the machine boots.

On the next screen, we **enter the "Node Name" (slaveNode1), select "Permanent Agent", then click "OK"**:

Dashboard › Nodes ›

- Back to Dashboard
- Manage Jenkins
- New Node
- Configure Clouds
- Node Monitoring

Build Queue ^

No builds in the queue.

Node name

slaveNode1

🔘 **Permanent Agent**

Adds a plain, permanent agent to Jenkins. This is called "permanent" because Jenkins doesn't provide higher level of integration with these agents, such as dynamic provisioning. Select this type if no other agent types apply — for example such as when you are adding a physical computer, virtual machines managed outside Jenkins, etc.

OK

After clicking "OK", **we'll be taken to a screen with a new form where we need to fill out the slave node's information**. We're considering the slave node to be running on Linux operating systems, hence the launch method is set to "Launch agents via ssh".

In the same way, we'll add relevant details, such as the name, description, and a number of executors.

We'll save our work by pressing the "Save" button. The "Labels" with the name "slaveNode1" will help us to set up jobs on this slave node:

Name

    slaveNode1

Description

    Slave node to execute builds

Number of executors

    1

Remote root directory

    /home/user

Labels

    slaveNode1

Usage

    Only build jobs with label expressions matching this node

Launch method

    Launch agents via SSH

    Host

        20.14.XXX.XXX

    Credentials

        jenkinuser/****** (jenkinuser) ▾   ⊕Add ▾

4. Building the Project on Slave Nodes

Now that our master and slave nodes are ready, we'll discuss the steps for building the project on the slave node.

For this, we start by clicking "New Item" in the top left corner of the dashboard.

Next, we need to enter the name of our project in the "Enter an item name" field and select the "Pipeline project", and then click the "OK" button.

On the next screen, we'll enter a "Description" (optional) and navigate to the "Pipeline" section. Make sure the "Definition" field has the Pipeline script option selected.
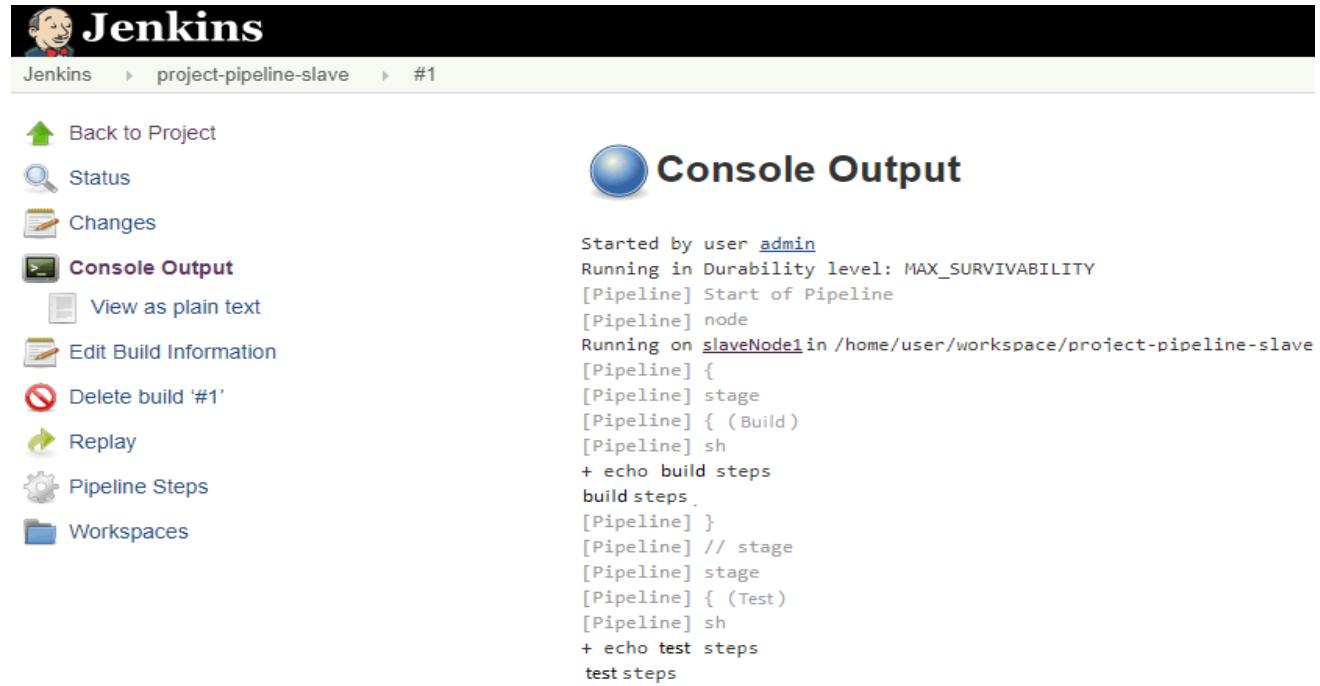
After this, we copy and paste the following declarative Pipeline script into a "script" field:

```
node('slaveNode1'){
   stage('Build') {
      sh '''echo build steps'''
   }
   stage('Test') {
      sh '''echo test steps'''
   }
}
Copy
```

Next, we click on the "Save" button. This will redirect to the Pipeline view page.

On the left pane, we click the "Build Now" button to execute our Pipeline. After Pipeline execution is completed, we'll see the Pipeline view:

We can verify the history of the executed build under the Build History by clicking the build number. As shown above, when we click on the build number and select "Console Output", we can see that the pipeline ran on our *slaveNode1* machine.

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: Jenkins is written in Java and requires Java to be installed on the host machine.

Git: If your project uses Git as the version control system, you'll need to have Git installed on the host machine.

Build Tools: Depending on the programming language and build process of your project, you may need to install build tools such as Maven, Gradle, or Ant.

Testing Tools: To run tests as part of your build process, you'll need to install any necessary testing tools, such as JUnit, TestNG, or Selenium.

Database Systems: If your project requires access to a database, you'll need to have the necessary database software installed on the host machine, such as MySQL, PostgreSQL, or Oracle.

Continuous Integration Plugins: To extend the functionality of Jenkins, you may need to install plugins that provide additional tools and features for continuous integration, such as the Jenkins GitHub plugin, Jenkins Pipeline plugin, or Jenkins Slack plugin.

To install these tools and software on the host machine, you can use a package manager such as apt or yum, or you can download and install the necessary software manually. You can also use a containerization tool such as Docker to run Jenkins and the necessary software in isolated containers, which can simplify the installation process and make it easier to manage the dependencies and tools needed for your build process.

Trigger

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.

predefined URL to trigger build remotely:

JENKINS_URL/job/JobName/build?token=TOKEN_NAME

JENKINS_URL: the IP and PORT which the Jenkins server is running

TOKEN_NAME: You have provided while selecting this build trigger.

//Example:

http://e330c73d.ngrok.io/job/test/build?token=12345

Whenever you will hit this URL from anywhere you project build will start.

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 HOUR DOM MONTH DOW

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.

To specify multiple values for one field, the following operators are available. In the order of precedence,

- * specifies all valid values
- M-N specifies a range of values
- M-N/X or */X steps by intervals of X through the specified range or whole valid range
- A,B,...,Z enumerates multiple values

Examples:

```
# every fifteen minutes (perhaps at :07,
        :22, :37, :52) H/15 * * *
                    * # every ten minutes in the first half of
every hour (three times, perhaps at :04, :14, :24) H(0-29)/10
        * * * * # once every
two hours at 45 minutes past the hour starting at 9:45 AM and finishing at
3:45 PM every weekday.
45              9-16/2          *          *          1-5
# once in every two hours slot between 9 AM and 5 PM every weekday (perhaps at
10:38 AM, 12:38 PM, 2:38 PM, 4:38 PM) H H(9-16)/2     *          *
                1-5
# once a day on the 1st and 15th of every month except
        December H H 1,15 1-11 *
```

After successfully scheduled the project build then the scheduler will invoke the build periodically according to your specified duration.

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

//This URL is a public URL where the Jenkins server is running

Here https://e330c73d.ngrok.io/ is the IP and port where my Jenkins is running.

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.

In this example, we used ngrok tool to expose my local address to the public.

To know more on how to add webhook in Jenkins pipeline, visit: https://blog.knoldus.com/opsinit-adding-a-github-webhook-in-jenkins-pipeline/

5)Poll SCM:

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

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.

There are several ways to chain jobs in Jenkins:

Build Trigger: You can use the build trigger in Jenkins to start one job after another. This is done

DevOps

by configuring the upstream job to trigger the downstream job when it completes.

Jenkinsfile: If you are using Jenkins Pipeline, you can write a Jenkinsfile to define the steps in your build pipeline. The Jenkinsfile can contain multiple stages, each of which represents a separate build job in the pipeline.
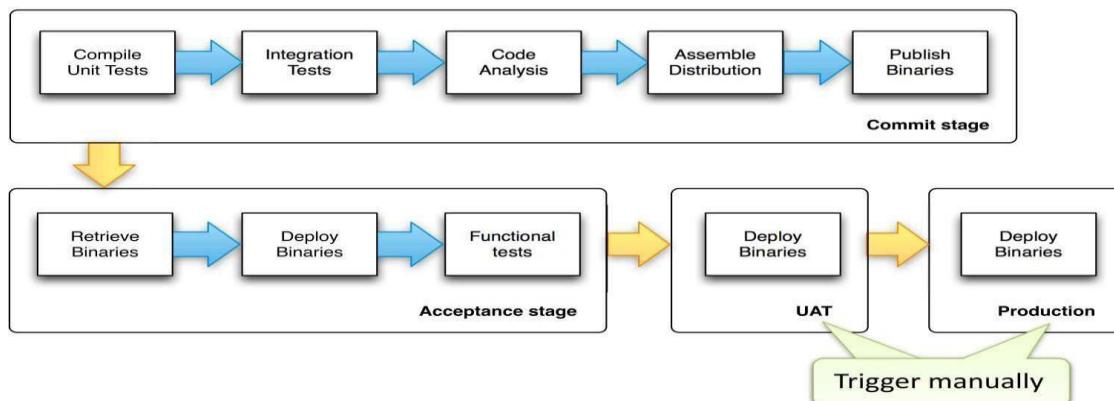
JobDSL plugin: The JobDSL plugin allows you to programmatically create and manage Jenkins jobs. You can use this plugin to create a series of jobs that are linked together and run in sequence.

Multi-Job plugin: The Multi-Job plugin allows you to create a single job that runs multiple build steps, each of which can be a separate build job. This plugin is useful if you have a build pipeline that requires multiple build jobs to be run in parallel.

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

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.

DevOps

The steps involved in a typical build pipeline include:

Code Commit: Developers commit code changes to a version control system such as Git.

Build and Compile: The code is built and compiled, and any necessary dependencies are resolved.

Unit Testing: Automated unit tests are run to validate the code changes.

Integration Testing: Automated integration tests are run to validate that the code integrates correctly with other parts of the system.

Staging: The code is deployed to a staging environment for further testing and validation.

Release: If the code passes all tests, it is deployed to the production environment.

Monitoring: The deployed code is monitored for performance and stability.

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.

Build servers

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.

Why build servers are important

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

DevOps

Without a build server you're slowed down by complicated, manual processes and the needless time constraints they introduce. For example, without a build server:

- Your team will likely need to commit code before a daily deadline or during change windows
- After that deadline passes, no one can commit again until someone manually creates and tests a build
- If there are problems with the code, the deadlines and manual processes further delay the fixes

Without a build server, the team battles unnecessary hurdles that automation removes. A build server will repeat these tasks for you throughout the day, and without those human-caused delays.

But CI doesn't just mean less time spent on manual tasks or the death of arbitrary deadlines, either. By automatically taking these steps many times a day, you fix problems sooner and your results become more predictable. Build servers ultimately help you deploy through your pipeline with more confidence.

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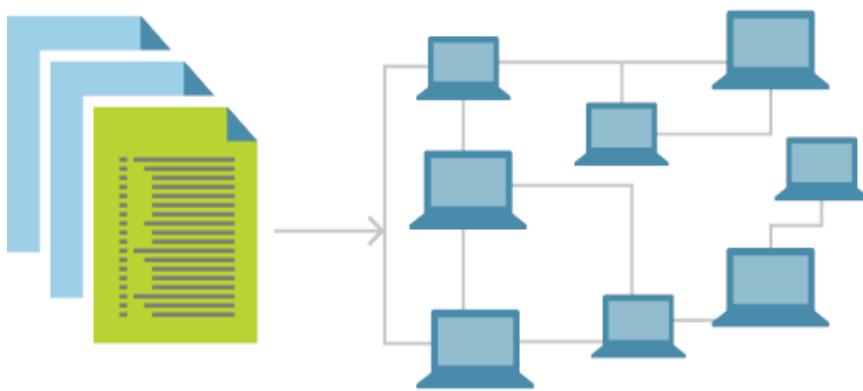
Deployment: Deploy the application to the server and test it to ensure it is functioning as expected.

Throughout the process, it is important to automate as much as possible using tools such as Ansible, Chef, or Puppet to ensure consistency and efficiency in building servers.

DevOps

Infrastructure as 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 <u>continuous delivery</u>. 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.

Idempotence, the ability of a given operation to always produce the same result, is an important IaC principle. A deployment command always sets the target environment into the same

DevOps

configuration, regardless of the environment's starting state. Idempotency is achieved by either

automatically configuring the existing target, or by discarding the existing target and recreating a fresh environment.

IAC can be achieved by using tools such as Terraform, CloudFormation, or Ansible to define infrastructure components in a file that can be versioned, tested, and deployed in a consistent and automated manner.

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.

Overall, IAC is a key component of modern DevOps practices, enabling organizations to manage their infrastructure in a more efficient, reliable, and scalable way.

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.

The steps involved in building by dependency order in DevOps include:

Define dependencies: Identify all the components of the system and the dependencies between them. This can be represented in a diagram or as a list.

Determine the build order: Based on the dependencies, determine the correct order in which components should be built and deployed.

Automate the build process: Use tools such as Jenkins, TravisCI, or CircleCI to automate the build and deployment process. This allows for consistency and repeatability in the build process.

Monitor progress: Monitor the progress of the build and deployment process to ensure that components are deployed in the correct order and that the system is functioning as expected.

DevOps

Test and validate: Test the system after deployment to ensure that all components are functioning as intended and that dependencies are resolved correctly.

Rollback: If necessary, have a rollback plan in place to revert to a previous version of the system if the build or deployment process fails.

In conclusion, building by dependency order in DevOps is a critical step in ensuring the success of a system deployment, as it ensures that components are deployed in the correct order and that dependencies are resolved correctly. This results in a more stable, reliable, and consistent system.

Build phases

In DevOps, there are several phases in the build process, including:

Planning: Define the project requirements, identify the dependencies, and create a build plan.

Code development: Write the code and implement features, fixing bugs along the way.

Continuous Integration (CI): Automatically build and test the code as it is committed to a version control system.

Continuous Delivery (CD): Automatically deploy code changes to a testing environment, where they can be tested and validated.

Deployment: Deploy the code changes to a production environment, after they have passed testing in a pre-production environment.

Monitoring: Continuously monitor the system to ensure that it is functioning as expected, and to detect and resolve any issues that may arise.
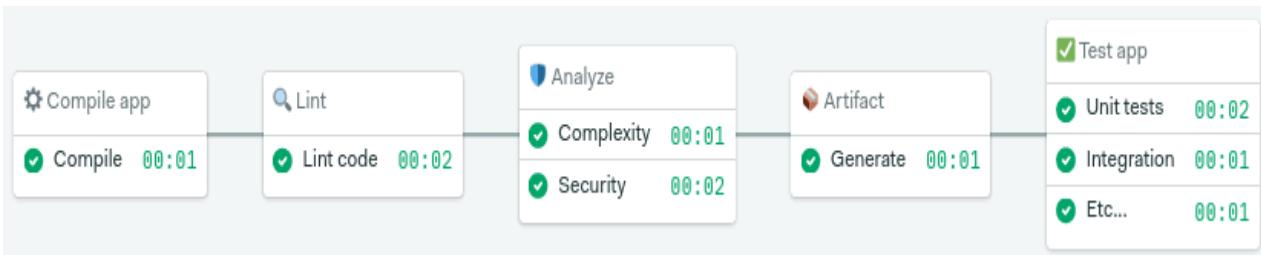
Maintenance: Continuously maintain and update the system, fixing bugs, adding new features, and ensuring its stability.

These phases help to ensure that the build process is efficient, reliable, and consistent, and that code changes are validated and deployed in a controlled manner. Automation is a key aspect of DevOps, and it helps to make these phases more efficient and less prone to human error.

In continuous integration (CI), this is where we build the application for the first time. The build stage is the first stretch of a CI/CD pipeline, and it automates steps like downloading dependencies, installing tools, and compiling.

Besides building code, build automation includes using tools to check that the code is safe and follows best practices. The build stage usually ends in the artifact generation step, where we create a production-ready package. Once this is done, the testing stage can begin.

DevOps

The build stage starts from code commit and runs from the beginning up to the test stage

We'll be covering testing in-depth in future articles (subscribe to the newsletter so you don't miss them). Today, we'll focus on build automation.

Build automation verifies that the application, at a given code commit, can qualify for further testing. We can divide it into three parts:

1. **Compilation**: the first step builds the application.
2. **Linting**: checks the code for programmatic and stylistic errors.
3. **Code analysis**: using automated source-checking tools, we control the code's quality.
4. **Artifact generation**: the last step packages the application for release or deployment.

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.

DevOps

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.

DevOps

Unit 5

Testing Tools and automation

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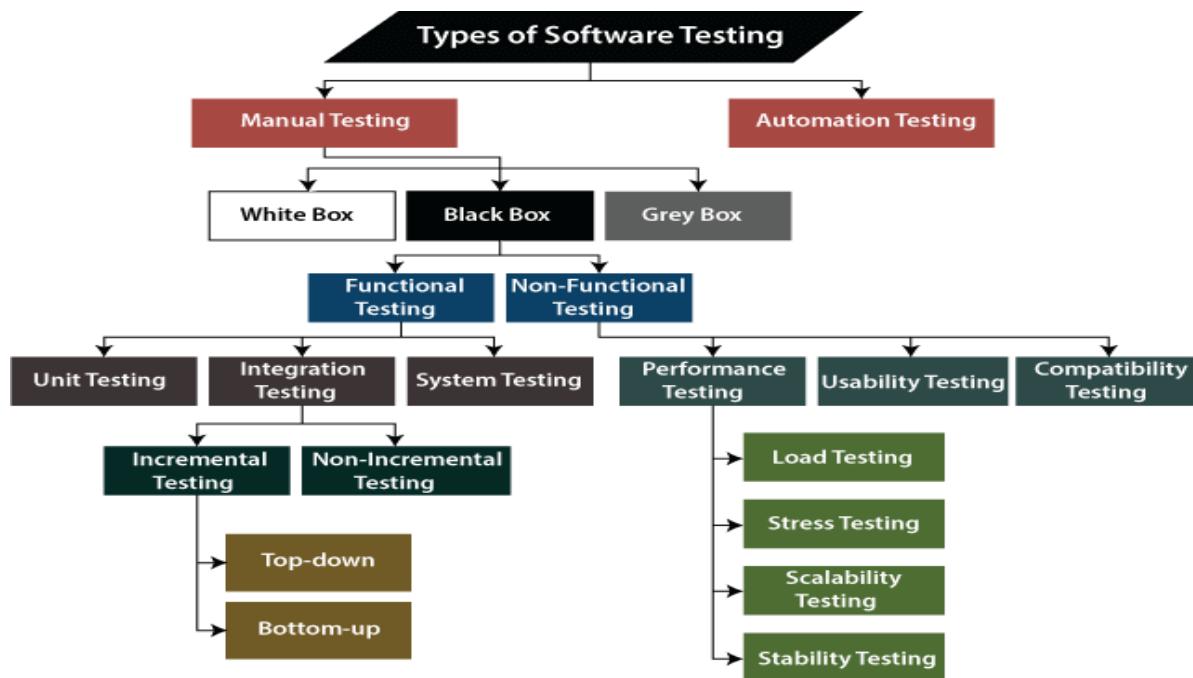
The categorization of software testing is a part of diverse testing activities, such as **test strategy, test deliverables, a defined test objective, etc**. And software testing is the execution of the software to find defects.

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:

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

DevOps

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**

DevOps

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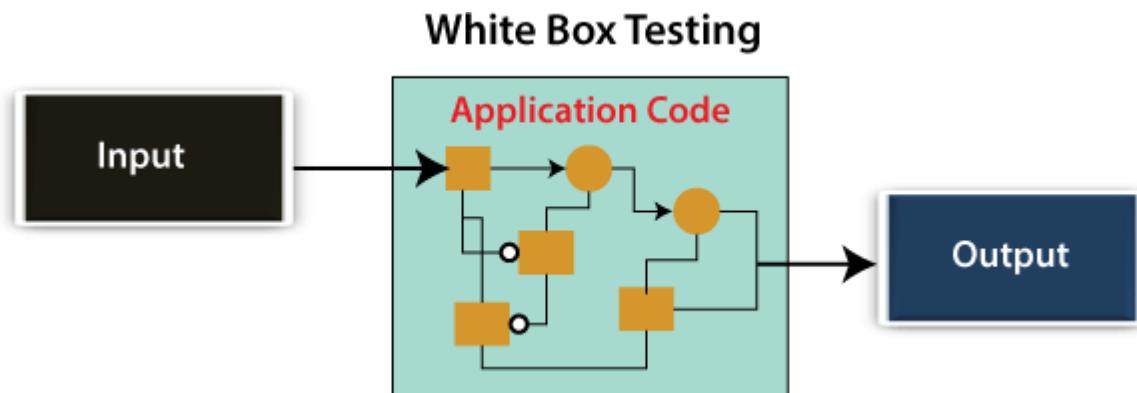
White Box Testing

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.

DevOps

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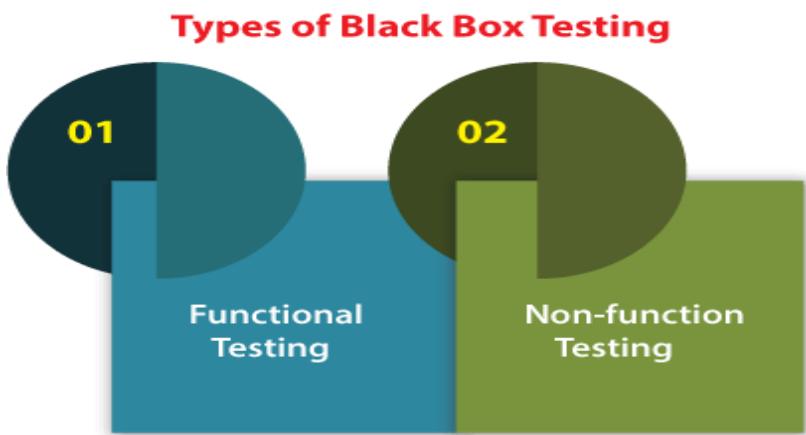
In other words, we can say that black box testing is a process of checking the functionality of an application as per the customer requirement. The source code is not visible in this testing; that's why it is known as **black-box testing**.

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**

DevOps

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:

DevOps

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:

DevOps

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.

DevOps

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.

DevOps

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**.
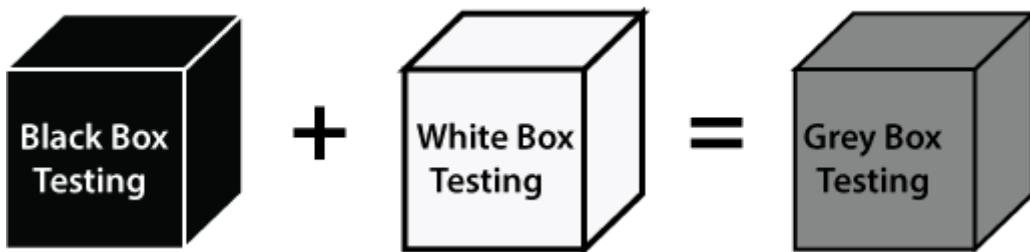
Here, **software** means we can test the application on the different operating systems and other browsers, and **hardware** means we can test the application on different sizes.

DevOps

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

Some other types of Software Testing

DevOps

In software testing, we also have some other types of testing that are not part of any above discussed testing, but those testing are required while testing any software or an application.

- o **Smoke Testing**
- o **Sanity Testing**
- o **Regression Testing**
- o **User Acceptance Testing**
- o **Exploratory Testing**
- o **Adhoc Testing**
- o **Security Testing**
- o **Globalization Testing**

Let's understand those types of testing one by one:

In **smoke testing**, we will test an application's basic and critical features before doing one round of deep and rigorous testing.

Or before checking all possible positive and negative values is known as **smoke testing**. Analyzing the workflow of the application's core and main functions is the main objective of performing the smoke testing.

Sanity Testing

It is used to ensure that all the bugs have been fixed and no added issues come into existence due to these changes. Sanity testing is unscripted, which means we cannot documented it. It checks the correctness of the newly added features and components.

Regression Testing

Regression testing is the most commonly used type of software testing. Here, the term **regression** implies that we have to re-test those parts of an unaffected application.

Regression testing is the most suitable testing for automation tools. As per the project type and accessibility of resources, regression testing can be similar to **Retesting**.

Whenever a bug is fixed by the developers and then testing the other features of the applications that might be simulated because of the bug fixing is known as **regression testing**.

DevOps

In other words, we can say that whenever there is a new release for some project, then we can perform Regression Testing, and due to a new feature may affect the old features in the earlier releases.

User Acceptance Testing

The User acceptance testing (UAT) is done by the individual team known as domain expert/customer or the client. And knowing the application before accepting the final product is called as **user acceptance testing**.

In user acceptance testing, we analyze the business scenarios, and real-time scenarios on the distinct environment called the **UAT environment**. In this testing, we will test the application before UAI for customer approval.

Exploratory Testing

Whenever the requirement is missing, early iteration is required, and the testing team has experienced testers when we have a critical application. New test engineer entered into the team then we go for the **exploratory testing**.

To execute the exploratory testing, we will first go through the application in all possible ways, make a test document, understand the flow of the application, and then test the application.

Adhoc Testing

Testing the application randomly as soon as the build is in the checked sequence is known as **Adhoc testing**.

It is also called **Monkey testing and Gorilla testing**. In Adhoc testing, we will check the application in contradiction of the client's requirements; that's why it is also known as **negative testing**.

When the end-user using the application casually, and he/she may detect a bug. Still, the specialized test engineer uses the software thoroughly, so he/she may not identify a similar detection.

Security Testing

It is an essential part of software testing, used to determine the weakness, risks, or threats in the software application.

DevOps

The execution of security testing will help us to avoid the nasty attack from outsiders and ensure our software applications' security.

In other words, we can say that security testing is mainly used to define that the data will be safe and endure the software's working process.

Globalization Testing

Another type of software testing is **Globalization testing.** Globalization testing is used to check the developed software for multiple languages or not. Here, the words **globalization** means enlightening the application or software for various languages.

Globalization testing is used to make sure that the application will support multiple languages and multiple features.

In present scenarios, we can see the enhancement in several technologies as the applications are prepared to be used globally.

Conclusion

In the tutorial, we have discussed various types of software testing. But there is still a list of more than 100+ categories of testing. However, each kind of testing is not used in all types of projects.

We have discussed the most commonly used types of Software Testing like **black-box testing, white box testing, functional testing, non-functional testing, regression testing, Adhoc testing, etc**.

Also, there are alternate classifications or processes used in diverse organizations, but the general concept is similar all over the place.

These testing types, processes, and execution approaches keep changing when the project, requirements, and scope change.

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.
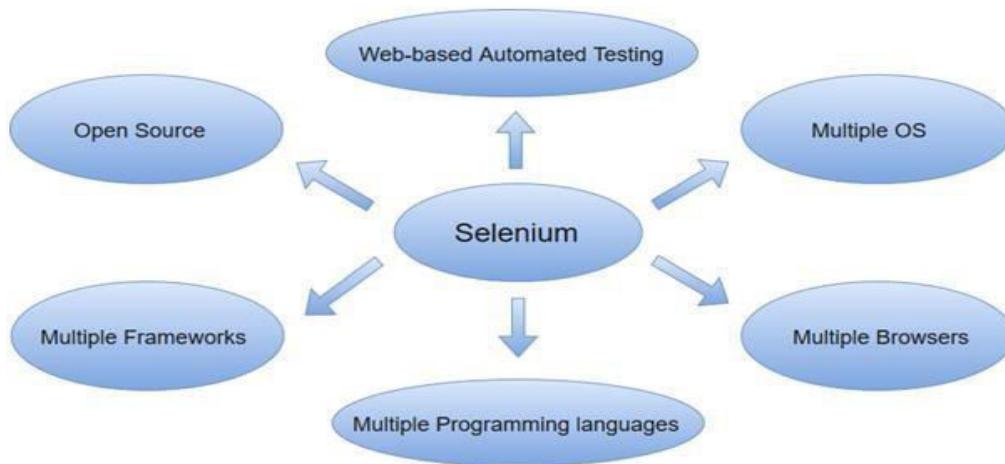
Selenium supports a variety of programming languages through the use of drivers specific to each Language.

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

DevOps

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.
- o 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.

DevOps

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.

What is Backend Testing?

Backend Testing is a testing method that checks the database or server-side of the web application. The main purpose of backend testing is to check the application layer and the database layer. It will find an error or bug in the database or server-side.

For implementing backend testing, the backend test engineer should also have some knowledge about that particular server-side or database language. It is also known as **Database Testing.**
Importance of Backend Testing: Backend testing is a must because anything wrong or error happens at the server-side, it will not further proceed with that task or the output will get differed or sometimes it will also cause problems such as data loss, deadlock, etc.,

Types of Backend Testing

The following are the different types of backend testing:

1. **Structural Testing**
2. **Functional Testing**
3. **Non-Functional Testing**

Let's discuss each of these types of backend testing.

1. Structural Testing
Structural testing is the process of validating all the elements that are present inside the data repository and are primarily used for data storage. It involves checking the objects of front-end developments with the database mapping objects.

Types of Structural Testing: The following are the different types of structural testing:

DevOps

a) Schema Testing: In this Schema Testing, the tester will check for the correctly mapped objects. This is also known as **mapping testing.** It ensures whether the objects of the front- end and the objects of the back-end are correctly matched or mapped. It will mainly focus on schema objects such as a table, view, indexes, clusters, etc., In this testing, the tester will find the issues of mapped objects like table, view, etc.,

b) Table and Column Testing: In this, it ensures that the table and column properties are correctly mapped.

- It ensures whether the table and the column names are correctly mapped on both the front- end side and server-side.
- It validates the datatype of the column is correctly mentioned.
- It ensures the correct naming of the column values of the database.
- It detects the unused tables and columns.
- It validates whether the users are able to give the correct input as per the requirement. For example, if we mention the wrong datatype for the column on the server-side which is different from the front-end then it will raise an error.

c) Key and Indexes Testing: In this, it validates the key and indexes of the columns.

- It ensures whether the mentioned key constraints are correctly provided. For example, Primary Key for the column is correctly mentioned as per the given requirement.
- It ensures the correct references of Foreign Key with the parent table.
- It checks the length and size of the indexes.
- It ensures the creation of clustered and non-clustered indexes for the table as per the requirement.
- It validates the naming conventions of the Keys.

d) Trigger Testing: It ensures that the executed triggers are fulfilling the required conditions of the DML transactions.

- It validates whether the triggers make the data updates correctly when we have executed them.
- It checks the coding conventions are followed correctly during the coding phase of the triggers.
- It ensures that the trigger functionalities of update, delete, and insert.

e) Stored Procedures Testing: In this, the tester checks for the correctness of the stored procedure results.

- It checks whether the stored procedure contains the valid conditions for looping and conditional statements as per the requirement.
- It validates the exception and error handling in the stored procedure.
- It detects the unused stored procedure.

DevOps

- It validates the cursor operations.
- It validates whether the TRIM operations are correctly applied or not.
- It ensures that the required triggers are implicitly invoked by executing the stored procedures.

f) Database Server Validation Testing: It validates the database configuration details as per the requirements.

- It validates that the transactions of the data are made as per the requirements.
- It validates the user's authentication and authorization.

For Example, If wrong user authentication is given, it will raise an error.

2. Functional Testing

Functional Testing is the process of validating that the transactions and operations made by the end-users meet the requirements.

Types of Functional Testing: The following are the different types of functional testing:

a) Black Box Testing:
- Black Box Testing is the process of checking the functionalities of the integration of the database.
- This testing is carried out at the early stage of development and hence It is very helpful to reduce errors.
- It consists of various techniques such as boundary analysis, equivalent partitioning, and cause-effect graphing.
- These techniques are helpful in checking the functionality of the database.
- The best example is the User login page. If the entered username and password are correct, It will allow the user and redirect to the next page.

b) White Box Testing:
- White Box Testing is the process of validating the internal structure of the database.
- Here, the specified details are hidden from the user.
- The database triggers, functions, views, queries, and cursors will be checked in this testing.
- It validates the database schema, database table, etc.,
- Here the coding errors in the triggers can be easily found.
- Errors in the queries can also be handled in this white box testing and hence internal errors are easily eliminated.

DevOps

3. Non-Functional Testing

Non-functional testing is the process of performing load testing, stress testing, and checking minimum system requirements are required to meet the requirements. It will also detect risks, and errors and optimize the performance of the database.
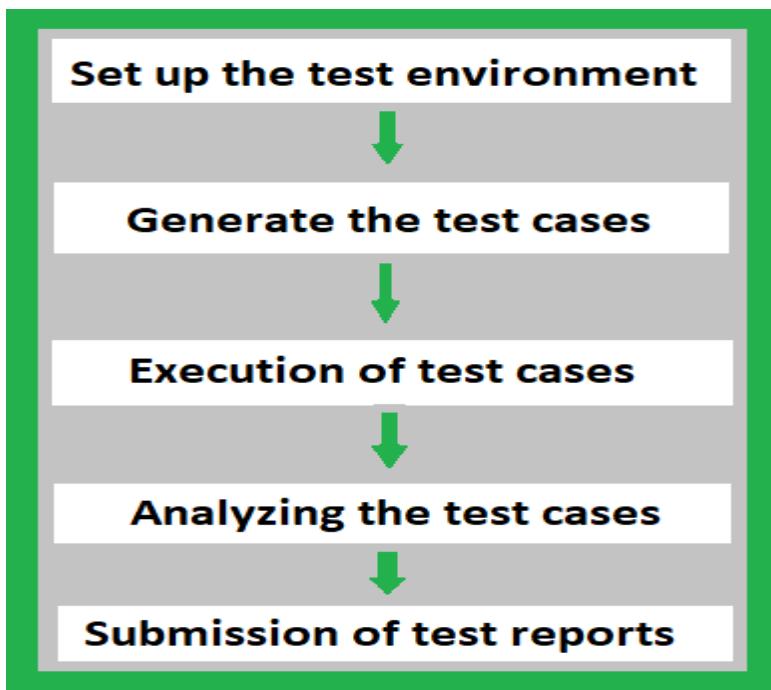
a) Load Testing:
- Load testing involves testing the performance and scalability of the database.
- It determines how the software behaves when it is been used by many users simultaneously.
- It focuses on good load management.
- For example, if the web application is accessed by multiple users at the same time and it does not create any traffic problems then the load testing is successfully completed.

b) Stress Testing:
- Stress Testing is also known as endurance testing. Stress testing is a testing process that is performed to identify the breakpoint of the system.
- In this testing, an application is loaded till the stage the system fails.
- This point is known as a breakpoint of the database system.
- It evaluates and analyzes the software after the breakage of system failure. In case of error detection, It will display the error messages.
- For example, if users enter the wrong login information then it will throw an error message.

DevOps

Backend Testing Process

1. Set up the Test Environment: When the coding process is done for the application, set up the test environment by choosing a proper testing tool for back-end testing. It includes choosing the right team to test the entire back-end environment with a proper schedule. Record all the testing processes in the documents or update them in software to keep track of all the processes.

2. Generate the Test Cases: Once the tool and the team are ready for the testing process, generate the test cases as per the business requirements. The automation tool itself will analyze the code and generate all possible test cases for developed code. If the process is manual then the tester will have to write the possible test cases in the testing tool to ensure the correctness of the code.

3. Execution of Test Cases: Once the test cases are generated, the tester or Quality Analyst needs to execute those test cases in the developed code. If the tool is automated, it will generate and execute the test cases by itself. Otherwise, the tester needs to write and execute those test cases. It will highlight whether the execution of test cases is executed successfully or not.

4. Analyzing the Test Cases: After the execution of test cases, it highlights the result of all the test cases whether it has been executed successfully or not. If an error occurs in the test cases, it will highlight where the particular error is formed or raised, and in some cases,

the automation tool will give hints regarding the issues to solve the error. The tester or Quality Analyst should analyze the code again and fix the issues if an error occurred.

5. Submission of Test Reports: This is the last stage in the testing process. Here, all the details such as who is responsible for testing, the tool used in the testing process, number of test cases generated, number of test cases executed successfully or not, time is taken to execute each test case, number of times test cases failed, number of times errors occurred. These details are either documented or updated in the software. The report will be submitted to the respective team.

Backend Testing Validation

The following are some of the factors for backend testing validation:

- **Performance Check:** It validates the performance of each individual test and the system behavior.
- **Sequence Testing:** Backend testing validates that the tests are distributed according to the priority.
- **Database Server Validations:** In this, ensures that the data fed through for the tests is correct or not.
- **Functions Testing:** In this, the test validates the consistency in transactions of the database.
- **Key and Indexes:** In this, the test ensures that the accurate constraint and the rules of constraints and indexes are followed properly.
- **Data Integrity Testing:** It is a technique in which data is verified in the database whether it is accurate and functions as per requirements.
- **Database Tables:** It ensures that the created table and the queries for the output are providing the expected result.
- **Database Triggers:** Backend Testing validates the correctness of the functionality of triggers.
- **Stored Procedures:** Backend testing validates the functions, return statements, calling the other events, etc., are correctly mentioned as per the requirements,
- **Schema:** Backend testing validates that the data is organized in a correct way as per the business requirement and confirms the outcome.

Tools For Backend Testing

The following are some of the tools for backend testing:

DevOps

1. LoadRunner:

- It is a stress testing tool.
- It is an automated performance and testing automation tool for analyzing system behavior and the performance of the system while generating the actual load.

2. Empirix-TEST Suite:

- It is acquired by Oracle from Empirix. It is a load testing tool.
- It validates the scalability along with the functionality of the application under heavy test.
- Acquisition with the Empirix -Test suite may be proven effective to deliver the application with improved quality.

3. Stored Procedure Testing Tools – LINQ:

- It is a powerful tool that allows the user to show the projects.
- It tracks all the ORM calls and database queries from the ORM.
- It enables to see the performance of the data access code and easily determine performance.

4. Unit Testing Tools – SQL Unit, DBFit, NDbUnit:

- **SQL UNIT:** SQLUnit is a Unit Testing Framework for Regression and Unit Testing of database stored procedures.
- **DBFit:** It is a part of FitNesse and manages stored procedures and custom procedures. Accomplishes database testing either through Java or .NET and runs from the command line.
- **NDbUnit:** It performs the database unit test for the system either before or after execution or compiled the other parts of the system.

5. Data Factory Tools:

- These tools work as data managers and data generators for backend database testing.
- It is used to validate the queries with a huge set of data.
- It allows performing both stress and load testing.

6. SQLMap:

- It is an open-source tool.
- It is used for performing Penetration Testing to automate the process of detection.
- Powerful detection of errors will lead to efficient testing and result in the expected behavior of the requirements.

7. phpMyadmin:

- This is the software tool and it is written in PHP.
- It is developed to handle the databases and we can execute test queries to ensure the correctness of the result as a whole and even for a separate table.

8. Automatic Efficient Test Generator (AETG):

- It mechanically generates the possible tests from user-defined requirements.
- It is based on algorithms that use ideas from statistical experimental design theory to reduce the number of tests needed for a specific level of test coverage of the input test space.

9. Hammer DB:

- It is an open-source tool for load testing.
- It validates the activity replay functionality for the oracle database.
- It is based on industry standards like TPC-C and TPC-H Benchmarks.

10. SQL Test:

- SQL Test uses an open-source tSQLt framework, views, stored procedures, and functions.
- This tool stores database object in a separate schema and if changes occur there is no need for clearing the process.
- It allows running the unit test cases for the SQL server database.

Advantages of Backend Testing

The following are some of the benefits of backend testing:

- Errors are easily detectable at the earlier stage.
- It avoids deadlock creation on the server-side.
- Web load management is easily achieved.
- The functionality of the database is maintained properly.
- It reduces data loss.
- Enhances the functioning of the system.
- It ensures the security and protection of the system.
- While doing the backend testing, the errors in the UI parts can also be detected and replaced.
- Coverage of all possible test cases.

Disadvantages of Backend Testing

The following are some of the disadvantages of backend testing:

- Good domain knowledge is required.
- Providing test cases for testing requires special attention.
- Investment in Organizational costs is higher.
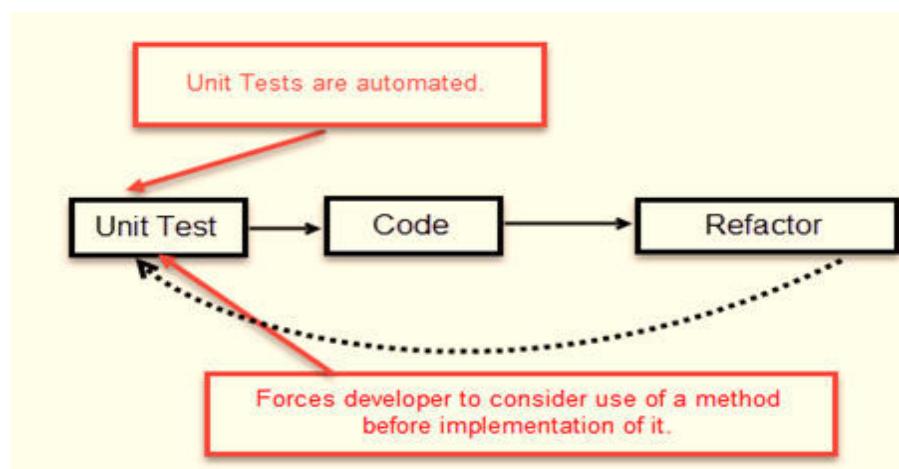
DevOps

- It takes more time to test.
- If more testing becomes fails then It will lead to a crash on the server-side in some cases.
- Errors or Unexpected results from one test case scenario will affect the other system results also.

Test-driven development

Test Driven Development (TDD) is software development approach in which test cases are developed to specify and validate what the code will do. In simple terms, test cases for each functionality are created and tested first and if the test fails then the new 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.


The simple concept of TDD is to write and correct the failed tests before writing new code (before development). This helps to avoid duplication of code as we write a small amount of code at a time in order to pass tests. (Tests are nothing but requirement conditions that we need to test to fulfill them).
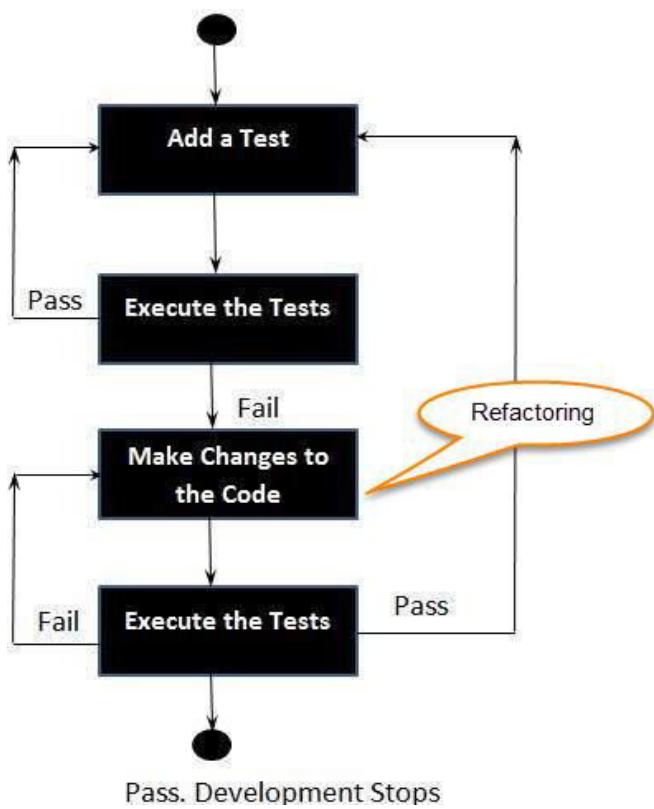
Test-Driven development is a process of developing and running automated test before actual development of the application. Hence, TDD sometimes also called as **Test First Development.**

DevOps

How to perform TDD Test

Following steps define how to perform TDD test,

1. Add a test.
2. Run all tests and see if any new test fails.
3. Write some code.
4. Run tests and Refactor code.
5. Repeat

Pass. Development Stops

TDD Vs. Traditional Testing

Below is the main difference between Test driven development and traditional testing:

TDD approach is primarily a specification technique. It ensures that your source code is thoroughly tested at confirmatory level.
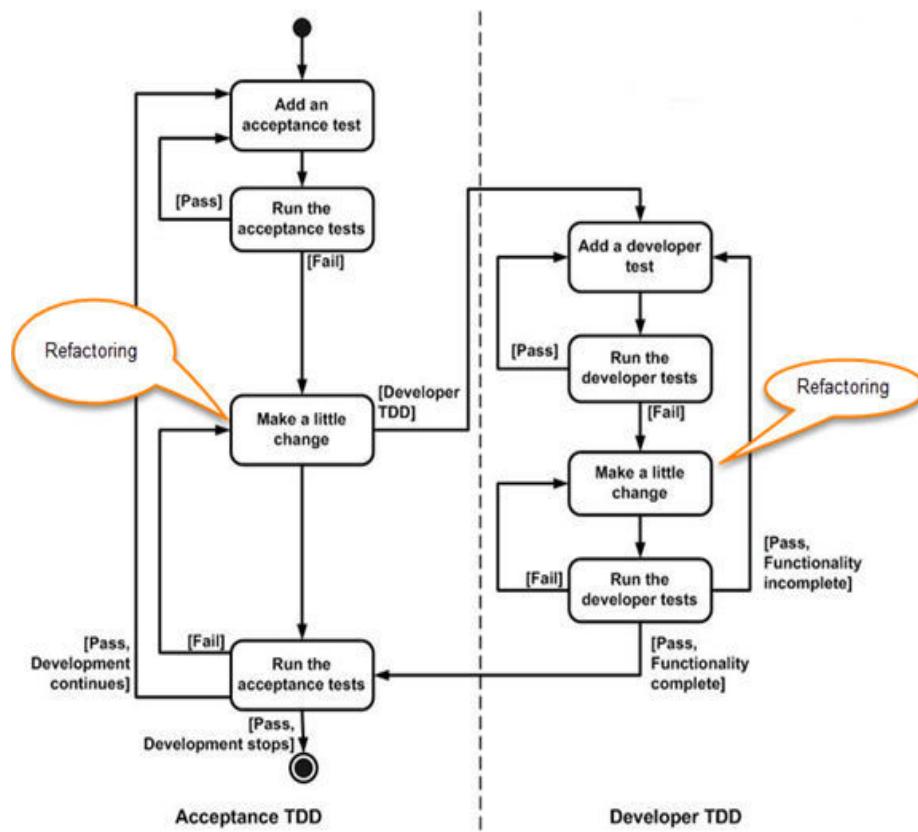
- With traditional testing, a successful test finds one or more defects. It is same as TDD. When a test fails, you have made progress because you know that you need to resolve the problem.
- TDD ensures that your system actually meets requirements defined for it. It helps to build your confidence about your system.
- In TDD more focus is on production code that verifies whether testing will work properly. In traditional testing, more focus is on test case design. Whether the test will show the proper/improper execution of the application in order to fulfill requirements.
- In TDD, you achieve 100% coverage test. Every single line of code is tested, unlike traditional testing.
- The combination of both traditional testing and TDD leads to the importance of testing the system rather than perfection of the system.
- In Agile Modeling (AM), you should "test with a purpose". You should know why you are testing something and what level its need to be tested.

What is acceptance TDD and Developer TDD

There are two levels of TDD

1. **Acceptance TDD (ATDD):** With ATDD you write a single acceptance test. This test fulfills the requirement of the specification or satisfies the behavior of the system. After that write just enough production/functionality code to fulfill that acceptance test. Acceptance test focuses on the overall behavior of the system. ATDD also was known as **Behavioral Driven Development (BDD).**
2. **Developer TDD:** With Developer TDD you write single developer test i.e. unit test and then just enough production code to fulfill that test. The unit test focuses on every small functionality of the system. Developer TDD is simply called as **TDD.**The main goal of ATDD and TDD is to specify detailed, executable requirements for your solution on a just in time (JIT) basis. JIT means taking only those requirements in consideration that are needed in the system. So increase efficiency.

DevOps

Acceptance TDD **Developer TDD**

REPL-driven development

REPL-driven development (Read-Eval-Print Loop) is an interactive programming approach that allows developers to execute code snippets and see their results immediately. This enables developers to test their code quickly and iteratively, and helps them to understand the behavior of their code as they work.

In a REPL environment, developers can type in code snippets, and the environment will immediately evaluate the code and return the results. This allows developers to test small bits of code and quickly see the results, without having to create a full-fledged application.

REPL-driven development is commonly used in dynamic programming languages such as Python, JavaScript, and Ruby. Some popular REPL environments include the Python REPL, Node.js REPL, and IRB (Interactive Ruby).

Benefits of REPL-driven development include:

DevOps

Increased efficiency: The immediate feedback provided by a REPL environment allows developers to test and modify their code quickly, without having to run a full-fledged application.

Improved understanding: By being able to see the results of code snippets immediately, developers can better understand how the code works and identify any issues early on.

Increased collaboration: REPL-driven development makes it easy for developers to share code snippets and collaborate on projects, as they can demonstrate the behavior of the code quickly and easily.

Overall, REPL-driven development is a useful tool for developers looking to improve their workflow and increase their understanding of their code. By providing an interactive environment for testing and exploring code, REPL-driven development can help developers to be more productive and efficient.

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

In DevOps, virtualization refers to the creation of virtual machines, containers, or environments that allow multiple operating systems to run on a single physical machine. The following are some of the commonly used virtualization stacks in DevOps:

Docker: An open-source platform for automating the deployment, scaling, and management of containerized applications.

DevOps

Kubernetes: An open-source platform for automating the deployment, scaling, and management of containerized applications, commonly used in conjunction with Docker.

VirtualBox: An open-source virtualization software that allows multiple operating systems to run on a single physical machine.

VMware: A commercial virtualization software that provides a comprehensive suite of tools for virtualization, cloud computing, and network and security management.

Hyper-V: Microsoft's hypervisor technology that enables virtualization on Windows-based systems.

These virtualization stacks play a crucial role in DevOps by allowing developers to build, test, and deploy applications in isolated, consistent environments, while reducing the costs and complexities associated with physical infrastructure.

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:

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.
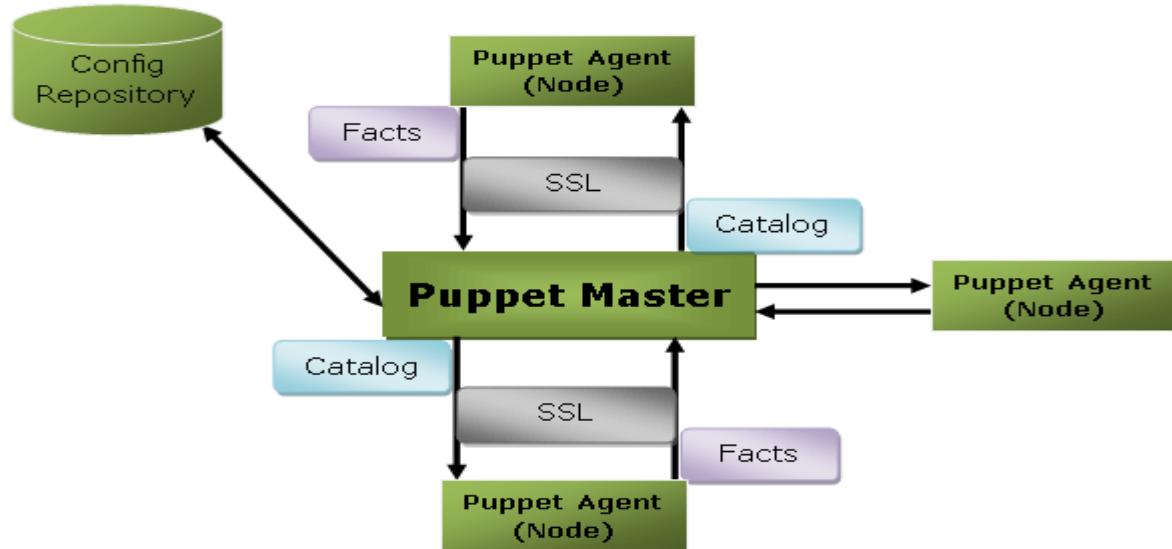
These methods are used in DevOps to automate various tasks, such as application deployment, software updates, or system configuration, on client devices. By executing code on the client side, DevOps teams can improve the speed, reliability, and security of their software delivery process.

Puppet master and agents:

Puppet Architecture

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.

DevOps

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.

DevOps

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:

DevOps

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.

Puppet Blocks

Puppet provides the flexibility to integrate Reports with third-party tools using Puppet APIs.

Four types of Puppet building blocks are

1. Resources
2. Classes
3. Manifest
4. Modules

Puppet Resources:

Puppet Resources are the building blocks of Puppet.

Resources are the **inbuilt functions** that run at the back end to perform the required operations in

puppet.

Puppet Classes:

A combination of different resources can be grouped together into a single unit called class.
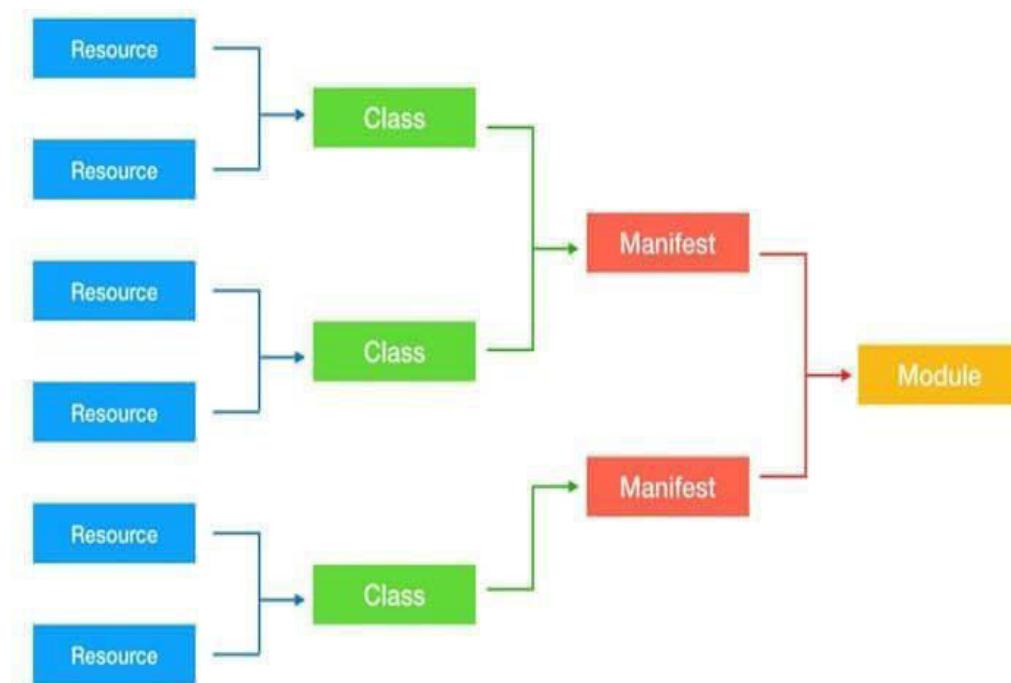
Puppet Manifest:

Manifest is a directory containing puppet DSL files. Those files have a .pp extension. The .pp extension stands for puppet program. The puppet code consists of definitions or declarations of Puppet Classes.

Puppet Modules:

Modules are a collection of files and directories such as Manifests, Class definitions. They are the re-usable and sharable units in Puppet.

For example, the MySQL module to install and configure MySQL or the Jenkins module to manage Jenkins, etc..

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.

DevOps

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

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

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

Sample Hosts File

This is the content of hosts file −

```
#File name: hosts
#Description: Inventory file for your application. Defines machine type abc
node to deploy specific artifacts
# Defines machine type def node to upload
metadata.

[abc-node]
#server1 ansible_host = <target machine for DU deployment> ansible_user = <Ansible
user> ansible_connection = ssh
server1 ansible_host = <your host name> ansible_user = <your unix user>
ansible_connection = ssh

[def-node]
#server2 ansible_host = <target machine for artifact upload>
ansible_user = <Ansible user> ansible_connection = ssh
server2 ansible_host = <host> ansible_user = <user> ansible_connection = ssh
```

DevOps

What is Configuration Management
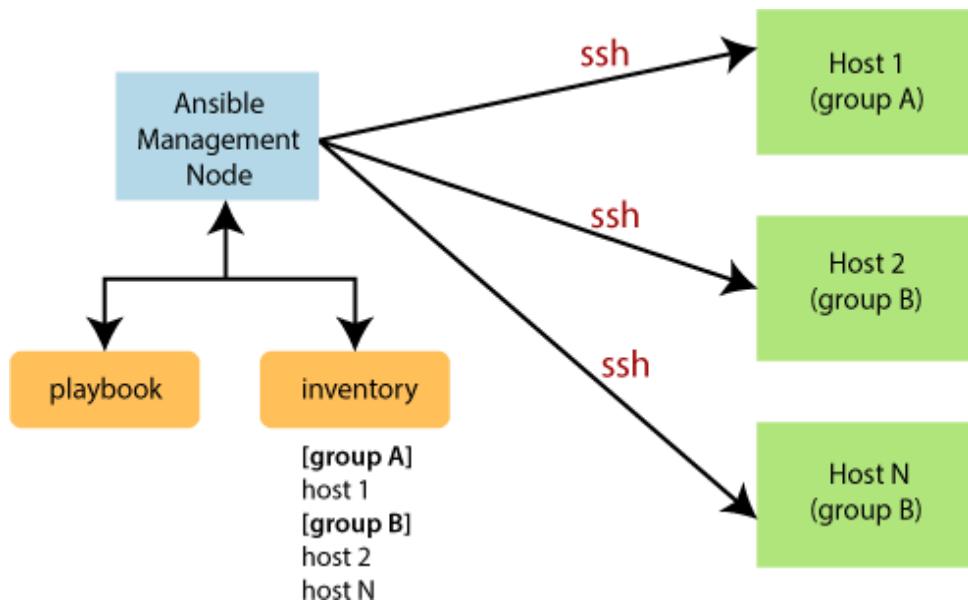
Configuration management in terms of Ansible means that it maintains configuration of the product performance by keeping a record and updating detailed information which describes an enterprise's hardware and software.

Such information typically includes the exact versions and updates that have been applied to installed software packages and the locations and network addresses of hardware devices. For e.g. If you want to install the new version of WebLogic/WebSphere server on all of the machines present in your enterprise, it is not feasible for you to manually go and update each and every machine.

You can install WebLogic/WebSphere in one go on all of your machines with Ansible playbooks and inventory written in the most simple way. All you have to do is list out the IP addresses of your nodes in the inventory and write a playbook to install WebLogic/WebSphere. Run the playbook from your control machine & it will be installed on all your nodes.

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

DevOps

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.
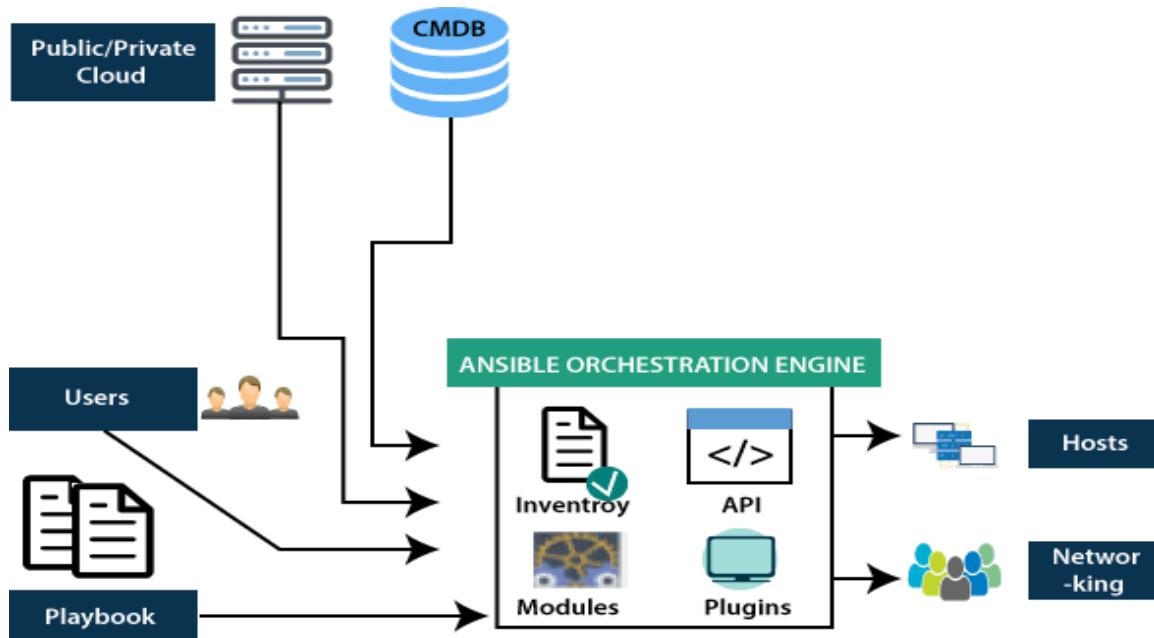
Terms used in Ansible

Terms	Explanation
Ansible Server	It is a machine where Ansible is installed and from which all tasks and playbooks will be executed.
Modules	The module is a command or set of similar commands which is executed on the client-side.
Task	A task is a section which consists of a single procedure to be completed.
Role	It is a way of organizing tasks and related files to be later called in a playbook.
Fact	The information fetched from the client system from the global variables with the gather facts operation.
Inventory	A file containing the data regarding the Ansible client-server.
Play	It is the execution of the playbook.
Handler	The task is called only if a notifier is present.
Notifier	The section attributed to a task which calls a handler if the output is changed.
Tag	It is a name set to a task that can be used later on to issue just that specific task or group of jobs.

DevOps

Ansible Architecture

The Ansible orchestration engine interacts with a user who is writing the Ansible playbook to execute the Ansible orchestration and interact along with the services of private or public cloud and configuration management database. You can show in the below diagram, such as:

Inventory

Inventory is lists of nodes or hosts having their IP addresses, databases, servers, etc. which are need to be managed.

API's

The Ansible API's works as the transport for the public or private cloud services.

Modules

Ansible connected the nodes and spread out the Ansible modules programs. Ansible executes the modules and removed after finished. These modules can reside on any machine; no database or servers are required here. You can work with the chose text editor or a terminal or version control system to keep track of the changes in the content.

DevOps

Plugins

Plugins is a piece of code that expends the core functionality of Ansible. There are many useful plugins, and you also can write your own.

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.

Hosts

In the Ansible architecture, hosts are the node systems, which are automated by Ansible, and any machine such as RedHat, Linux, Windows, etc.

Networking

Ansible is used to automate different networks, and it uses the simple, secure, and powerful agentless automation framework for IT operations and development. It uses a type of data model which separated from the Ansible automation engine that spans the different hardware quite easily.

Cloud

A cloud is a network of remote servers on which you can store, manage, and process the data. These servers are hosted on the internet and storing the data remotely rather than the local server. It just launches the resources and instances on the cloud, connect them to the servers, and you have good knowledge of operating your tasks remotely.

CMDB

CMDB is a type of repository which acts as a data warehouse for the IT installations.

Puppet Components

Following are the key components of Puppet:

- o Manifests
- o Module
- o Resource
- o Factor

DevOps

- o M-collective
- o Catalogs
- o Class
- o Nodes

Let's understand these components in detail:

Manifests
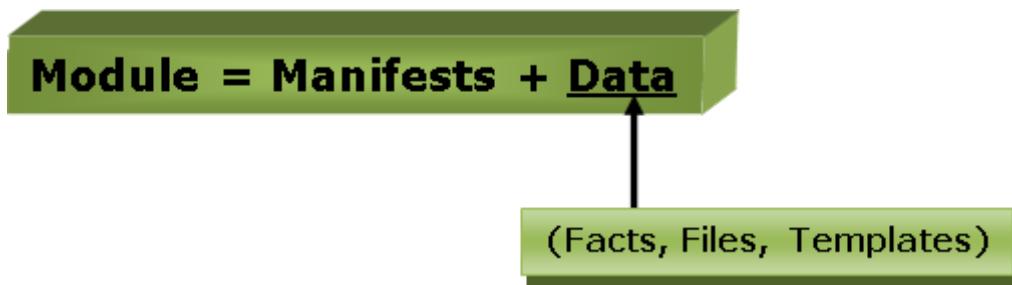
Puppet Master contains the Puppet Slave's configuration details, and these are written in Puppet's native language.

Manifest is nothing but the files specifying the configuration details for Puppet slave. The extension for manifest files is .pp, which means Puppet Policy. These files consist of puppet scripts describing the configuration for the slave.

Module

The puppet module is a set of manifests and data. Here data is file, facts, or templates. The module follows a specific directory structure. These modules allow the puppet program to split into multiple manifests. Modules are simply self-contained bundles of data or code.

Let's understand the module by the following image:

Resource

Resources are a basic unit of system configuration modeling. These are the predefined functions that run at the backend to perform the necessary operations in the puppet.

Each puppet resource defines certain elements of the system, such as some particular service or package.

DevOps

Factor

The factor collects facts or important information about the puppet slave. 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.

M-Collective

M-collective is a framework that enables parallel execution of several jobs on multiple Slaves. This framework performs several functions, such as:

- o This is used to interact with clusters of puppet slaves; they can be in small groups or very large deployments.
- o To transmit demands, use a broadcast model. All Slaves receive all requests at the same time, requests have filters attached, and only Slaves matching the filter can act on requests.
- o This is used to call remote slaves with the help of simple command-line tools.
- o This is used to write custom reports about your infrastructure.

Catalogs

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.

All the required states of slave resources are described in the catalog.

Class

Like other programming languages, the puppet also supports a class to organize the code in a better way. Puppet class is a collection of various resources that are grouped into a single unit.

Nodes

The nodes are the location where the puppet slaves are installed used to manage all the clients and servers.

DevOps

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 in reducing 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.

Why Chef?

Chef is a configuration management technology used to automate the infrastructure provisioning. It is developed on the basis of Ruby DSL language. It is used to streamline the task of configuration and managing the company's server. It has the capability to get integrated with any of the cloud technology.

In DevOps, we use Chef to deploy and manage servers and applications in-house and on the cloud.

Features of Chef

Following are the most prominent features of Chef −

- Chef uses popular Ruby language to create a domain-specific language.
- Chef does not make assumptions on the current status of a node. It uses its mechanisms to get the current status of machine.
- Chef is ideal for deploying and managing the cloud server, storage, and software.

Advantages of Chef

Chef offers the following advantages −

- **Lower barrier for entry** − As Chef uses native Ruby language for configuration, a standard configuration language it can be easily picked up by anyone having some development experience.
- **Excellent integration with cloud** − Using the knife utility, it can be easily integrated with any of the cloud technologies. It is the best tool for an organization that wishes to distribute its infrastructure on multi-cloud environment.

Disadvantages of Chef

Some of the major drawbacks of Chef are as follows −

- One of the huge disadvantages of Chef is the way cookbooks are controlled. It needs constant babying so that people who are working should not mess up with others cookbooks.

- Only Chef solo is available.
- In the current situation, it is only a good fit for AWS cloud.
- It is not very easy to learn if the person is not familiar with Ruby.
- Documentation is still lacking.

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 given infrastructure to the desired state as listed in the recipe.

Resource
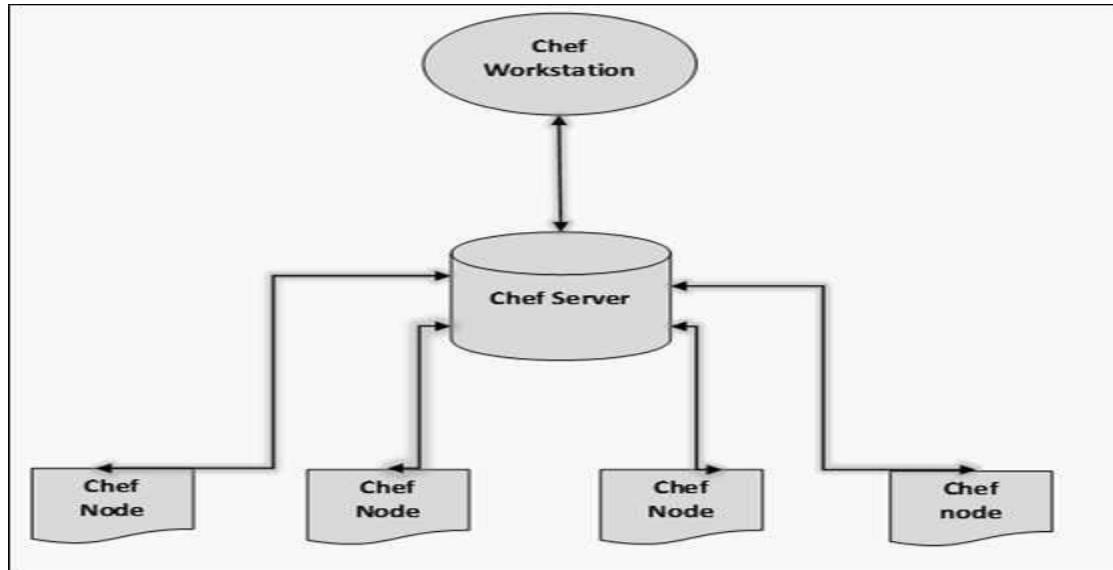
It is the basic component of a recipe used to manage the infrastructure with different kind of states. There can be multiple resources in a recipe, which will help in configuring and managing the infrastructure. For example −

- **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.

DevOps

- 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

DevOps

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**.

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.

Benefits of SaltStack

Being simple as well as a feature-rich system, Salt provides many benefits and they can be summarized as below −

- **Robust** − Salt is powerful and robust configuration management framework and works around tens of thousands of systems.
- **Authentication** − Salt manages simple SSH key pairs for authentication.
- **Secure** − Salt manages secure data using an encrypted protocol.
- **Fast** − Salt is very fast, lightweight communication bus to provide the foundation for a remote execution engine.
- **Virtual Machine Automation** − The Salt Virt Cloud Controller capability is used for automation.
- **Infrastructure as data, not code** − Salt provides a simple deployment, model driven configuration management and command execution framework.

DevOps

Introduction to ZeroMQ

Salt is based on the **ZeroMQ** library and it is an embeddable networking library. It is lightweight and a fast messaging library. The basic implementation is in **C/C++** and native implementations for several languages including **Java** and **.Net** is available.

ZeroMQ is a broker-less peer-peer message processing. ZeroMQ allows you to design a complex communication system easily.

ZeroMQ comes with the following five basic patterns −

- **Synchronous Request/Response** − Used for sending a request and receiving subsequent replies for each one sent.
- **Asynchronous Request/Response** − Requestor initiates the conversation by sending a Request message and waits for a Response message. Provider waits for the incoming Request messages and replies with the Response messages.
- **Publish/Subscribe** − Used for distributing data from a single process (e.g. publisher) to multiple recipients (e.g. subscribers).
- **Push/Pull** − Used for distributing data to connected nodes.
- **Exclusive Pair** − Used for connecting two peers together, forming a pair.
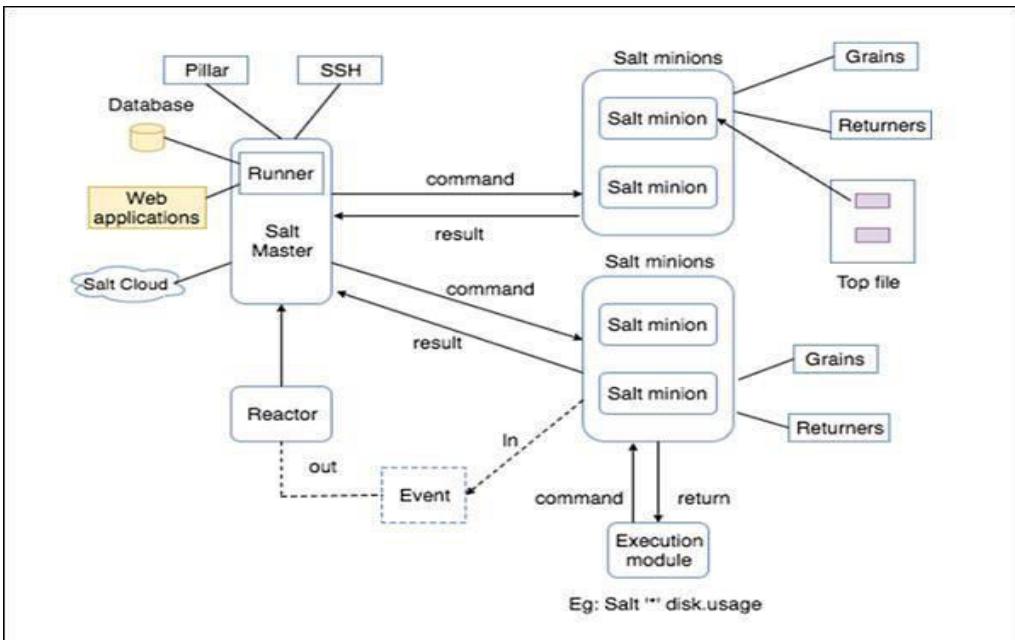
ZeroMQ is a highly flexible networking tool for exchanging messages among clusters, cloud and other multi system environments. ZeroMQ is the **default transport library** presented in SaltStack.

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.

DevOps

Eg: Salt '*' disk.usage

- **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
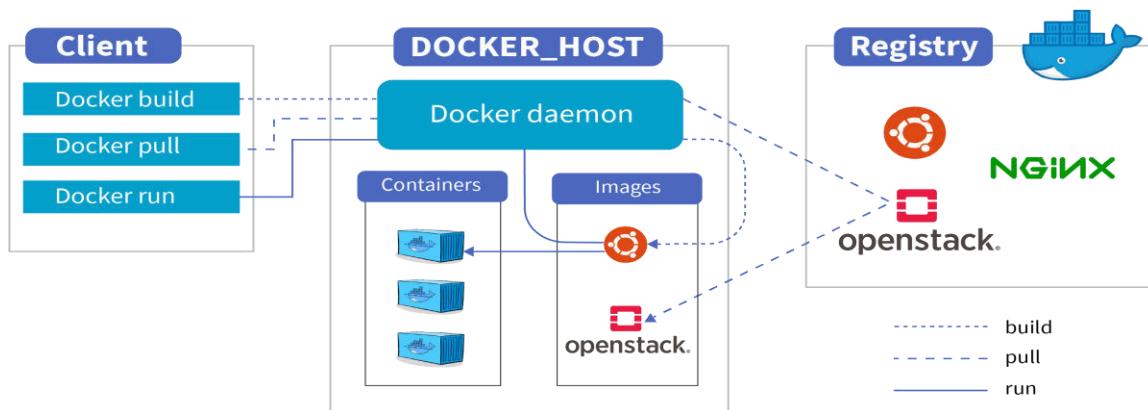
Docker has the following components

- **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 your own private registry.

DevOps

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.

Example docker run command
The following command runs an ubuntu container, attaches interactively to your local command-line session, and runs /bin/bash.
$ docker run -i -t ubuntu /bin/bash

When you run this command, the following happens (assuming you are using the default registry configuration):

1. If you do not have the ubuntu image locally, Docker pulls it from your configured registry, as though you had run docker pull ubuntu manually.
2. Docker creates a new container, as though you had run a docker container create command manually.
3. Docker allocates a read-write filesystem to the container, as its final layer. This allows a running container to create or modify files and directories in its local filesystem.

4. Docker creates a network interface to connect the container to the default network, since you did not specify any networking options. This includes assigning an IP address to the container. By default, containers can connect to external networks using the host machine's network connection.

5. Docker starts the container and executes /bin/bash. Because the container is running interactively and attached to your terminal (due to the -i and -t flags), you can provide input using your keyboard while the output is logged to your terminal.
6. When you type exit to terminate the /bin/bash command, the container stops but is not removed. You can start it again or remove it.

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.