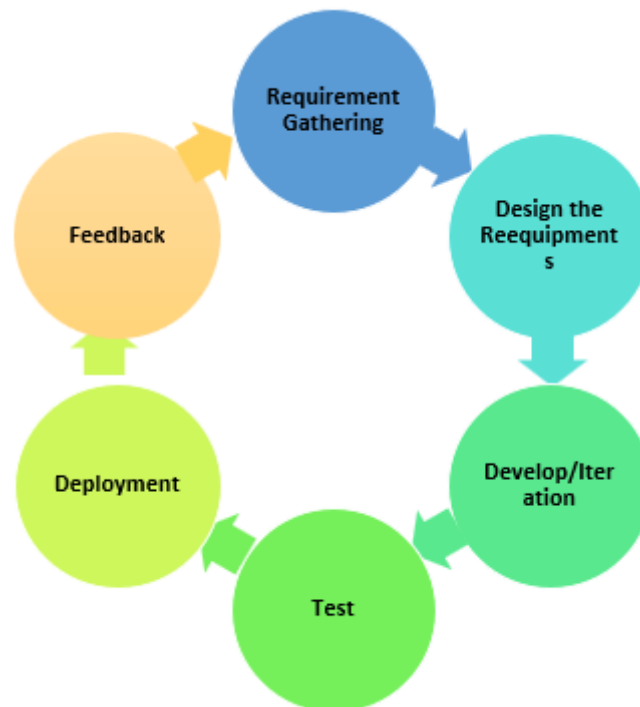


What is an Agile Development Model?

- The Agile development Model is an incremental and iterative process of software development. It defines each iteration's number, duration, and scope in advance. Every iteration is considered a short "frame" in the Agile process model, which mostly lasts from two to four weeks.
- Agile Model divides tasks into time boxes to provide specific functionality for the release. Each build is incremental in terms of functionality, with the final build containing all the attributes. The division of the entire project into small parts helps minimize the project risk and the overall project delivery time.

Phases of Agile Model

Here are the different phases of Agile:



Here are the important stages involved in the Agile Model process in the SDLC life cycle:

1.Requirements Gathering: In this Agile model phase, you must define the requirements. The business opportunities and the time, effort required for the project should also be discussed. By analyzing this information, you can determine a system's economic and technical feasibility.

2.Design the Requirements: Following the feasibility study, you can work with stakeholders to define requirements. Using the UFD diagram or high-level UML diagram, you can determine how the new system will be incorporated into your existing software system.

3.Develop/Iteration: The real work begins at this stage after the software development team defines and designs the requirements. Product, design, and development teams start working, and the product will undergo different stages of improvement using simple and minimal functionality.

4.Test: This phase of the Agile Model involves the testing team. For example, the Quality Assurance team checks the system's performance and reports bugs during this phase.

5.Deployment: In this phase, the initial product is released to the user.

6.Feedback: After releasing the product, the last step of the Agile Model is feedback. In this phase, the team receives feedback about the product and works on correcting bugs based on the received feedback.

Compared to Waterfall, Agile cycles are short. There may be many such cycles in a project. The phases are repeated until the product is delivered.

What are the important Agile Model Manifestos?

Here is the essential manifesto of the Agile Model:

- Individuals and interactions are given priority over processes and tools.
- Adaptive, empowered, self-organizing team.
- Focuses on working software rather than comprehensive documentation.
- Agile Model in software engineering aims to deliver complete customer satisfaction by rapidly delivering valuable software.
- Welcome changes in requirements, even late in the development phase.
- Daily co-operation between businesspeople and developers.
- Priority is customer collaboration over contract negotiation.
- It enables you to satisfy customers through early and frequent delivery.
- A strong emphasis is placed on face-to-face communication.
- Developing working software is the primary indicator of progress.
- Promote sustainable development pace.
- A continuous focus is placed on technical excellence and sound design.
- An improvement review is conducted regularly by the team.

Agile Testing Methods:

- Scrum
- Crystal
- Dynamic Software Development Method(DSDM)
- Feature Driven Development(FDD)
- Lean Software Development
- eXtreme Programming(XP)

1.Scrum

SCRUM is an agile development process focused primarily on ways to manage tasks in team-based development conditions.

There are three roles in it, and their responsibilities are:

- **Scrum Master:** The scrum can set up the master team, arrange the meeting and remove obstacles for the process.
- **Product owner:** The product owner makes the product backlog, prioritizes the delay and is responsible for the distribution of functionality on each repetition.
- **Scrum Team:** The team manages its work and organizes the work to complete the sprint or cycle.

eXtreme Programming(XP)

This type of methodology is used when customers are constantly changing demands or requirements, or when they are not sure about the system's performance.

Crystal:

Using Crystal methodology is one of the most straightforward and most flexible Approaches to developing software, recognizing that each project has unique characteristics.

Crystal methodologies are categorized as below:

- **CLEAR:** User for small and low critical efforts.
- **ORANGE:** User for moderately larger and critical projects.
- **ORANGE WEB:** Typically, electronic business

Dynamic Software Development Method(DSDM):

DSDM is a rapid application development strategy for software

development and gives an agile project distribution structure.

The essential features of DSDM are that users must be actively connected, and teams have been given the right to make decisions.

The techniques used in DSDM are:

1. Time Boxing
2. MoSCoW Rules
3. Prototyping

Feature Driven Development(FDD):

This method focuses on "Designing and Building" features.

FDD describes the small steps of the work that should be obtained separately per function.

Lean Software Development:

Lean software development methodology follows the principle

"just in time production." The lean method indicates the

Increasing speed of software development and reducing costs.

When to use the Agile Model?

- When frequent changes are required.
- When a highly qualified and experienced team is available.
- When a customer is ready to have a meeting with a software team all the time.
- When project size is small.

Advantage(Pros) of Agile Method:

1. Frequent Delivery
2. Face-to-Face Communication with clients.
3. Efficient design and fulfils the business requirement.
4. Anytime changes are acceptable.
5. It reduces total development time.

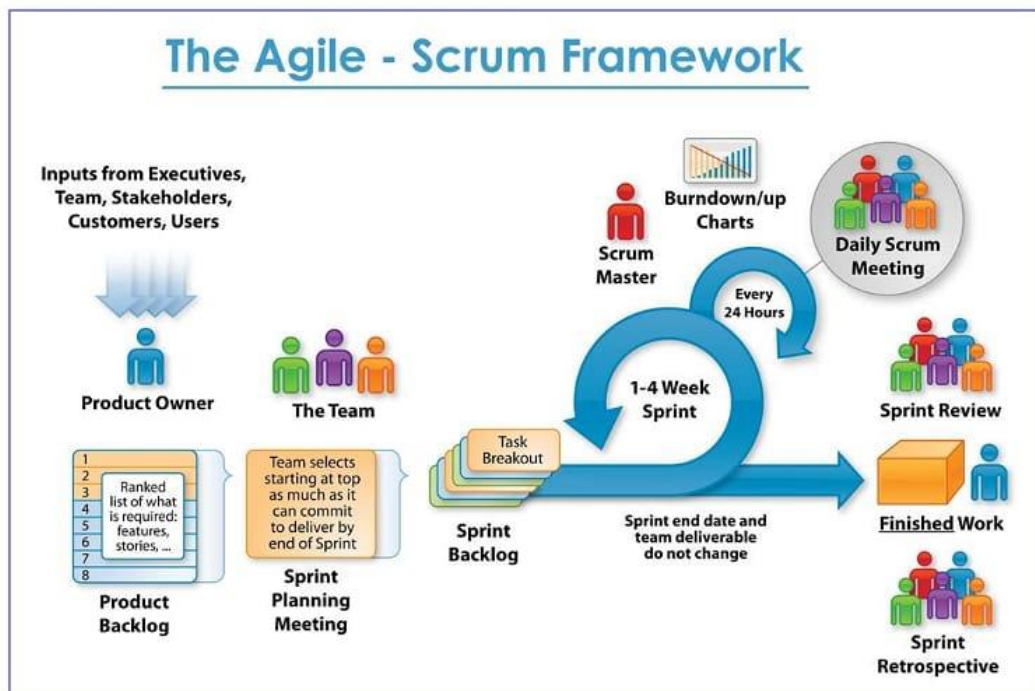
Disadvantages(Cons) of Agile Model:

1. Due to the shortage of formal documents, it creates confusion and

crucial decisions taken throughout various Phases.

2. Due to the lack of proper documentation,
maintenance of the finished project can become a difficulty.

Agile scrum methodology:



1. Scrum can easily be considered to be the most popular agile framework. The term 'scrum' is much considered synonymously to 'agile' by most practitioners. But that is a misconception. Scrum is just one of the frameworks by which you can implement agile.
 2. The goal of Scrum is to improve communication, teamwork, and speed of development.
 3. The Scrum methodology is one of many agile methodologies, all of which turn the historically predictive or waterfall project and product development triangle on its head.
- Thus, Scrum is a methodology to exhibit agile.

Yes. The scrum comply with agile manifesto principles as explained below:

-:) The Agile Manifesto is core to the Scrum methodology and can be considered the underlying mindset behind all agile methodologies. The Scrum methodology overlays on the Agile Manifesto to put how-tos to the whats.

-:) In other words, Scrum is one of the many how-to guides for developing outcomes in an agile mindset. Here's a breakdown of the Agile Manifesto coupled with a few core Scrum principles;

1. Individuals and interactions over processes and tools.
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation

4. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
5. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Benefits Of The Scrum Methodology:

The Scrum methodology is an agile project or product delivery technique that helps teams work more efficiently and effectively. It's a simple, yet powerful approach that can be used for software development, product design, marketing projects, and more.

Some of the benefits of using the Scrum methodology include:

Increased productivity: When teams are able to work in short bursts of time called "sprints," they can typically achieve more than if they were working on the project over a longer period of time.

Increased collaboration: The Scrum methodology encourages team members to work together closely to achieve common goals. This can help reduce misunderstandings and improve communication.

Increased flexibility: The scrum methodology is flexible and can be adapted to meet the specific needs of the project team. This can help ensure that the project prioritizes completion of the most important items while keeping commitments of cost and schedule.

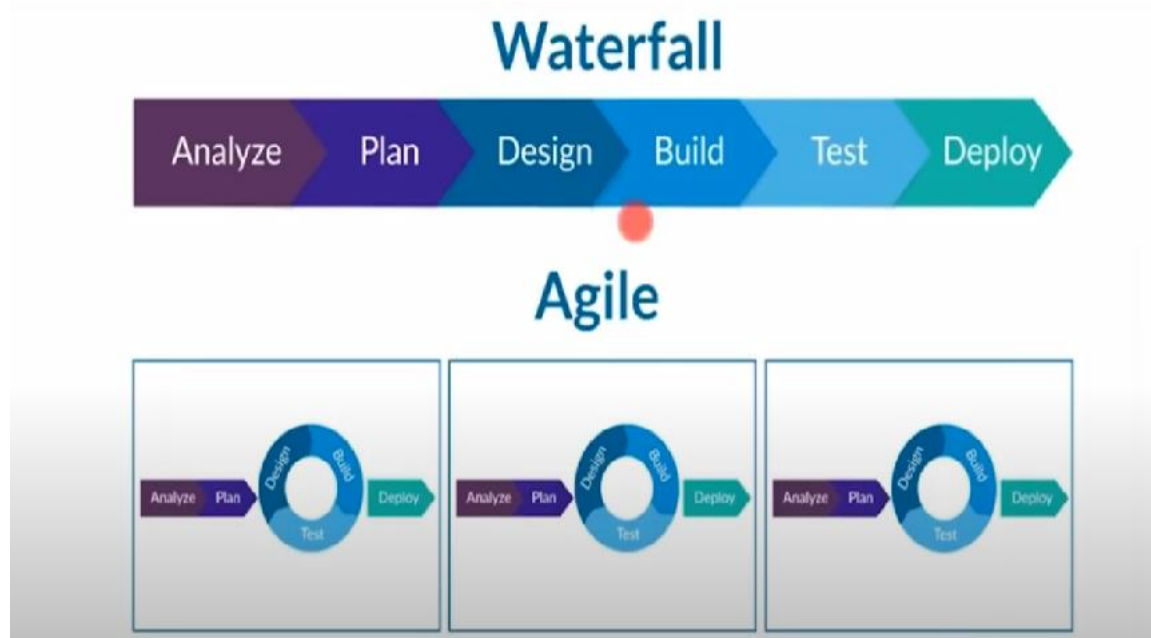
Improved quality: By using short sprints, teams are able to focus on completing specific tasks and ensuring that they are of the highest quality possible. This can help avoid costly mistakes and improve the overall quality of the final product or service.

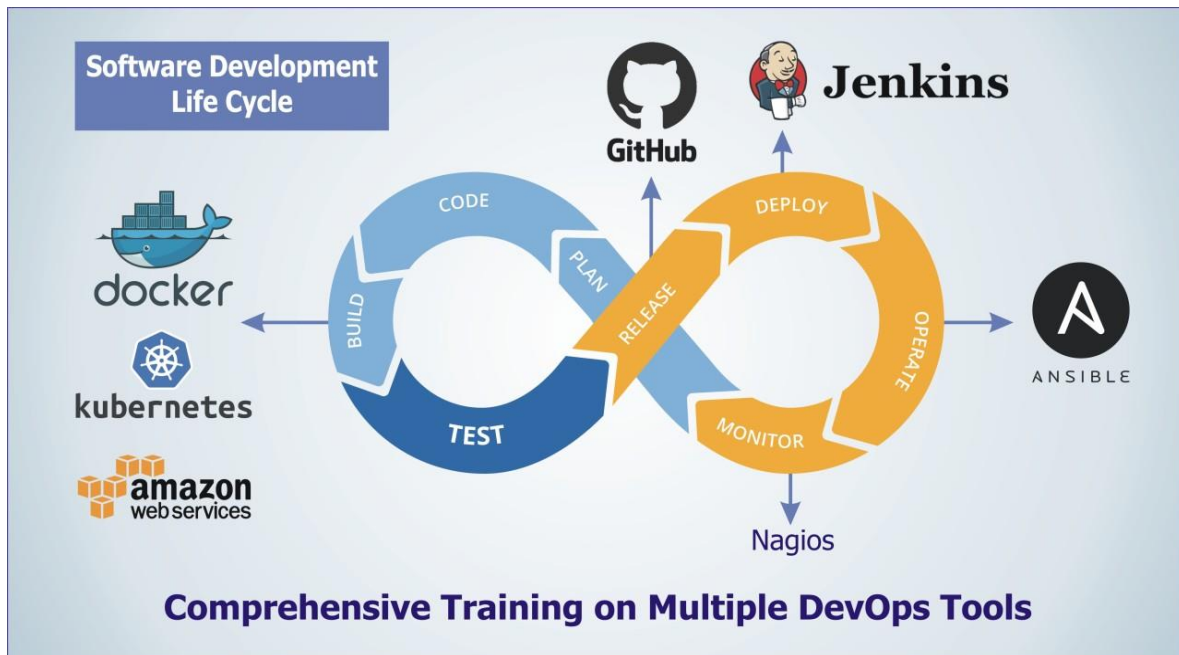
.

Agile Model Vs. Waterfall Model

Agile and Waterfall models are two different methods for the software development process. Despite their differences in approach, both methodologies can be used at times, depending upon the project and the requirements.

Agile Model	Waterfall Model
Agile methodologies propose incremental and iterative approaches to software design	Software development flows sequentially from start point to end point.
The Agile Model in software engineering is broken into individual models that designers work on	The design process is not broken into individual models
The customer has early and frequent opportunities to look at the product and make decisions and changes.	The customer can only see the product at the end of the project.
The Agile Model is considered unstructured compared to the waterfall model	Waterfall models are more secure because they are plan oriented
Small projects can be implemented very quickly. For large projects, it isn't easy to estimate the development time.	All sorts of the project can be estimated and completed.
Test plan is reviewed after each Sprint	The test plan is hardly discussed during the test phase.



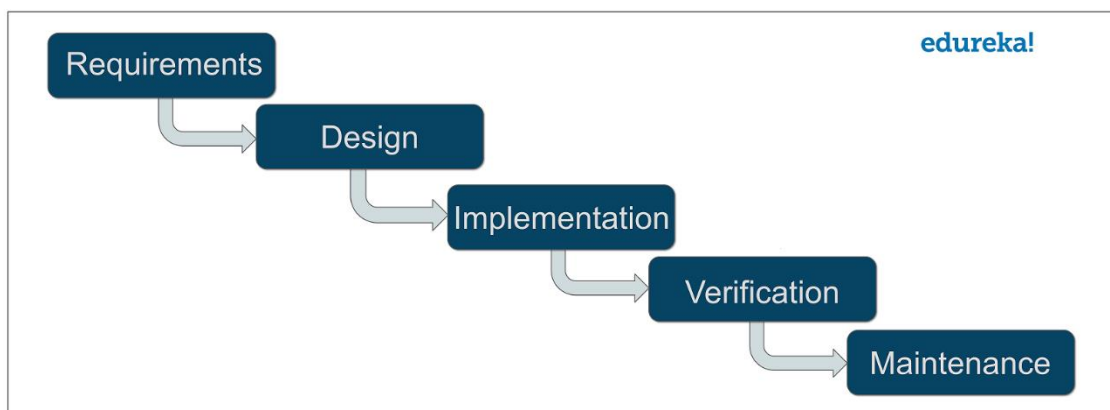


Why DevOps?

Before we know what DevOps is, it is very important to know how DevOps came into existence. Before DevOps, we had the waterfall model and the agile model for software development. So let us have a look at the waterfall model.

Waterfall Model

The waterfall model can be defined as a sequential process in the development of a system or software that follows a top-down approach. This model was a straight forward and linear model. The waterfall model had various phases such as Requirement Definition, Software Design, Implementation, Testing, Deployment, and Maintenance.

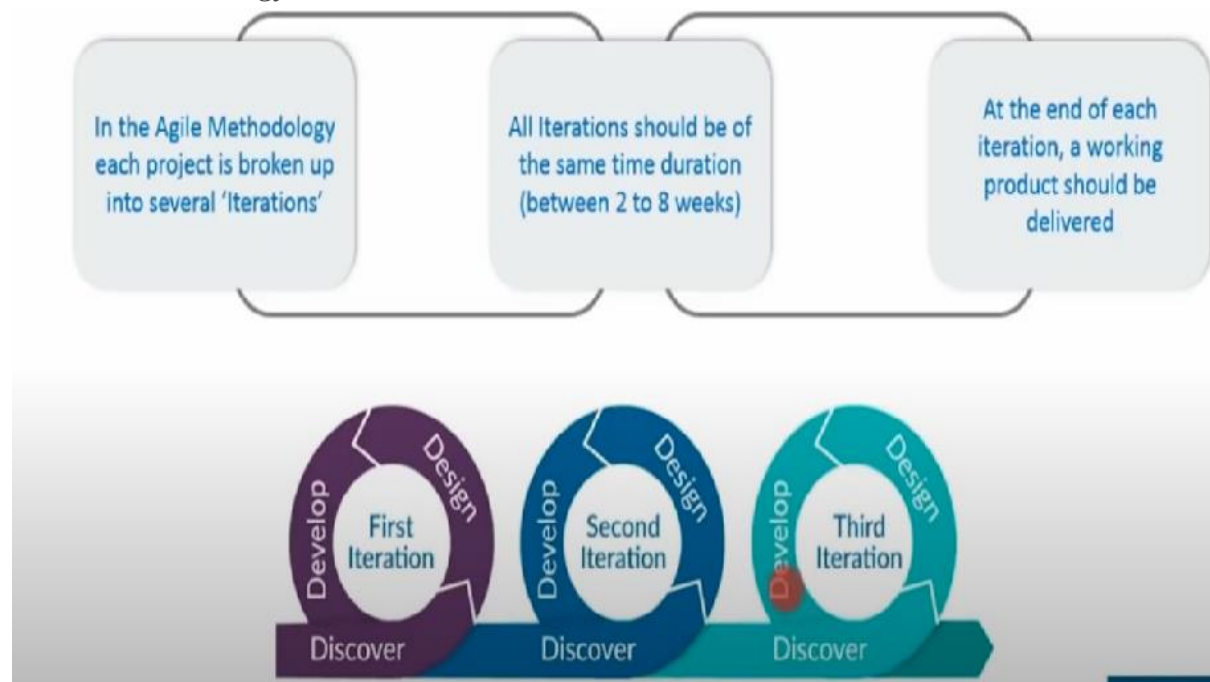


Software development companies that used the waterfall model approach, had to spend a lot of time to get their product right. That is because unless you complete a particular phase, you could not proceed to the next phase. Also, the working software was delivered only after the final phase.

This model was only suitable for projects which had stable requirements. By stable, mean that requirements will not change with the time. But in today's world, this is a very unlikely thing

because requirements keep on changing from time to time. These were a few drawbacks of the waterfall model.

AGILE Methodology



Next came the agile methodology of software development.

[Agile methodology](#) is a practice that promotes continuous iteration of development and testing throughout the software development life cycle of the project. Both development and testing activities are concurrent, unlike the Waterfall model. While the Agile approach brought agility to development, **it was lost on Operations which did not come up to speed with agile practices.**

There was a lack of collaboration between Developers and Operation Engineers and this slowed down the development process and releases. Software companies had begun to realize the need for better collaboration between the teams and faster delivery of software. **This gave birth to the DevOps approach.** DevOps enabled continuous software delivery with less complex problems to fix and faster resolution of problems

Limitation in Agile Model



- A very simple example might be the choice of systems used to report bugs. Quite often, development teams and quality assurance teams use different systems to handle tasks and bugs.
- This creates unnecessary friction between the teams and further separates them when they should really focus on working together instead. The operations team might, in turn, use a third system to handle requests for deployment to the organization's servers.

To overcome flaws in Agile introduce Devops

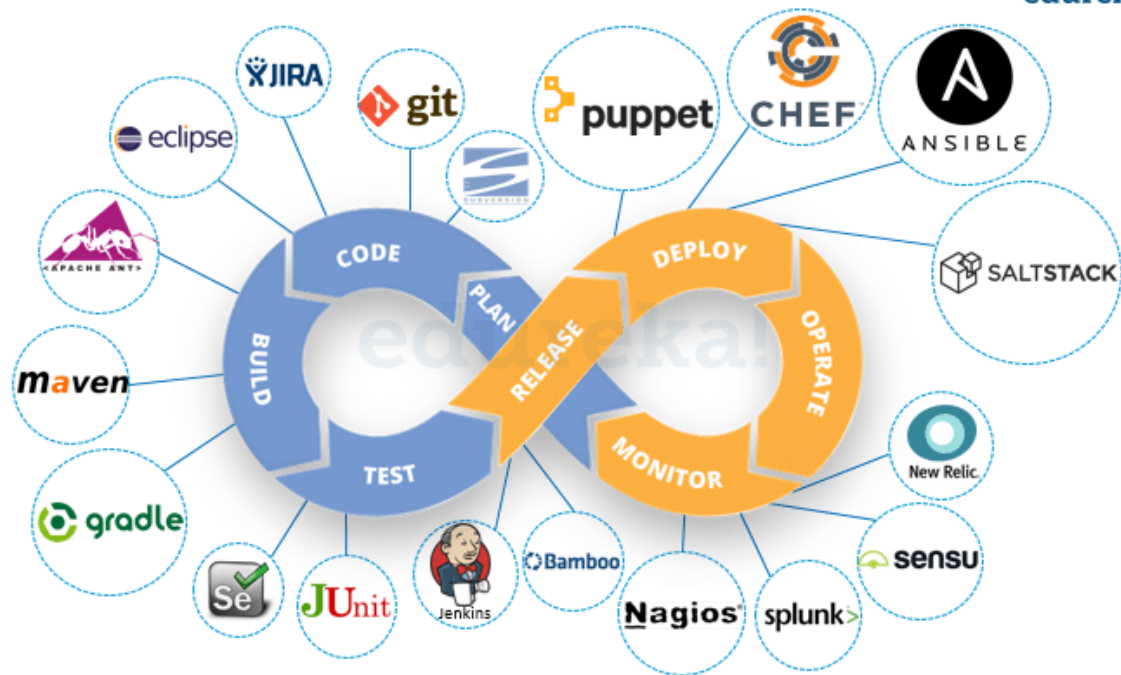
- maintenance costs
- three systems are replaced by one
- DevOps is automation and Continuous Delivery
- DevOps processes must be fast
- DevOps engineers work on making enterprise processes faster, more efficient, and more reliable.

What is DevOps?

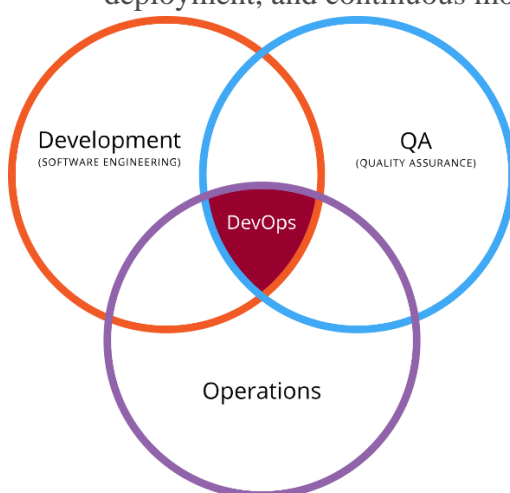
- DevOps is, by definition, a field that spans several disciplines. It is a field that is very practical and hands-on, but at the same time, you must understand both the technical background and the nontechnical cultural aspects.
- DevOps is a set of [practices](#), [tools](#), and a [cultural philosophy](#) that automate and integrate the processes between software development and IT teams. It emphasizes

team empowerment, cross-team communication and collaboration, and technology automation.

edureka!

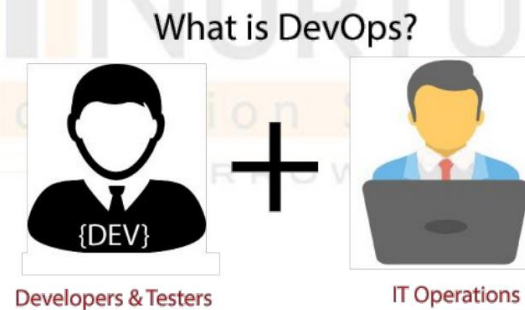


- The term **DevOps** is a combination of two words namely Development and Operations. DevOps is a practice that allows a single team to manage the entire application development life cycle, that is, development, testing, deployment, and operations.
- The aim of DevOps is to shorten the system's development life cycle while delivering features, fixes, and updates frequently in close alignment with business objectives.
- DevOps is a software development approach through which superior quality software can be developed quickly and with more reliability. It consists of various stages such as continuous development, continuous integration, continuous testing, continuous deployment, and continuous monitoring.

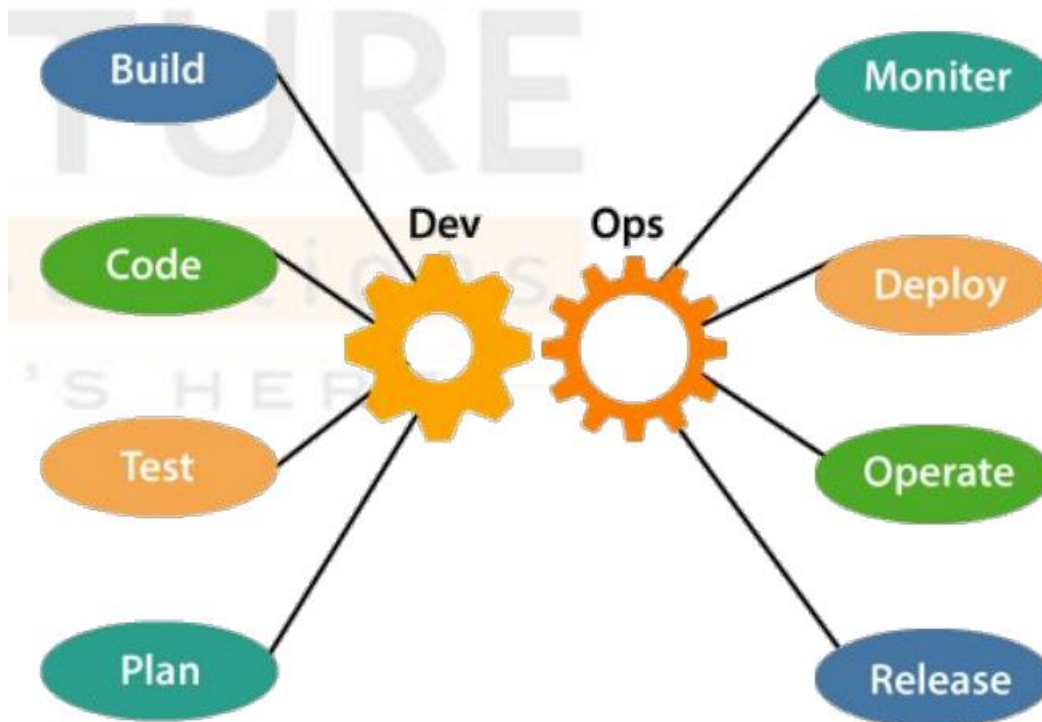


What Is DevOps?

It is the union of people, process, and technology to continuously give value to consumers. It is a blend of development (Dev) and operations (Ops).



DevOps Components



DevOps Main Objectives

Automate for Build

Automat Testing

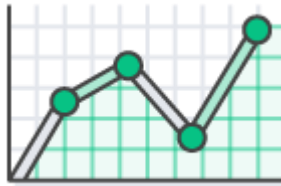
Automate Deployment

Automate Monitoring

Automate Issue Tracking

Automate Feedbacks

Benefits of DevOps



Speed

Move at high velocity so you can innovate for customers faster, adapt to changing markets better, and grow more efficient at driving business results. The DevOps model enables your developers and operations teams to achieve these results. For example, [microservices](#) and [continuous delivery](#) let teams take ownership of services and then release updates to them quicker.



Rapid Delivery

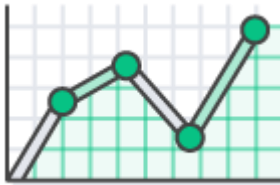
Increase the frequency and pace of releases so you can innovate and improve your product faster. The quicker you can release new features and fix bugs, the faster you can respond to your customers' needs and build competitive advantage. [Continuous](#)

[integration](#) and [continuous delivery](#) are practices that automate the software release process, from build to deploy.



Reliability

Ensure the quality of application updates and infrastructure changes so you can reliably deliver at a more rapid pace while maintaining a positive experience for end users. Use practices like [continuous integration](#) and [continuous delivery](#) to test that each change is functional and safe. [Monitoring and logging](#) practices help you stay informed of performance in real-time.



Scale

Operate and manage your infrastructure and development processes at scale. Automation and consistency help you manage complex or changing systems efficiently and with reduced risk. For example, [infrastructure as code](#) helps you manage your development, testing, and production environments in a repeatable and more efficient manner.



Improved Collaboration

Build more effective teams under a DevOps cultural model, which emphasizes values such as ownership and accountability. Developers and operations teams [collaborate](#) closely, share many responsibilities, and combine their workflows. This reduces inefficiencies and saves time (e.g. reduced handover periods between developers and operations, writing code that takes into account the environment in which it is run).



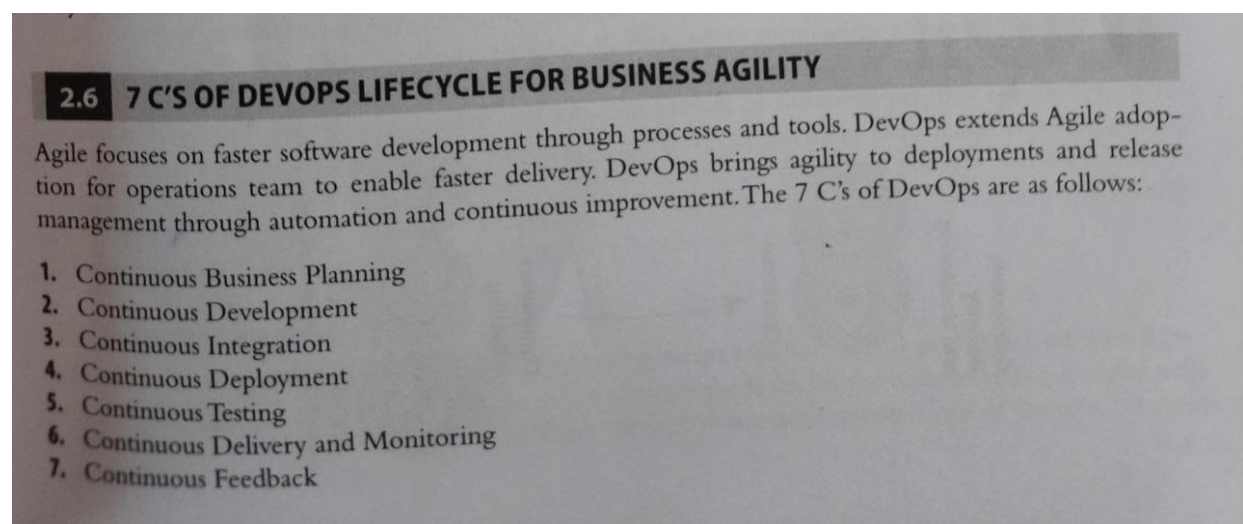
Security

Move quickly while retaining control and preserving compliance. You can adopt a DevOps model without sacrificing security by using automated compliance policies, fine-grained controls, and configuration management techniques. For example, using infrastructure as code and [policy as code](#), you can define and then track compliance at scale.

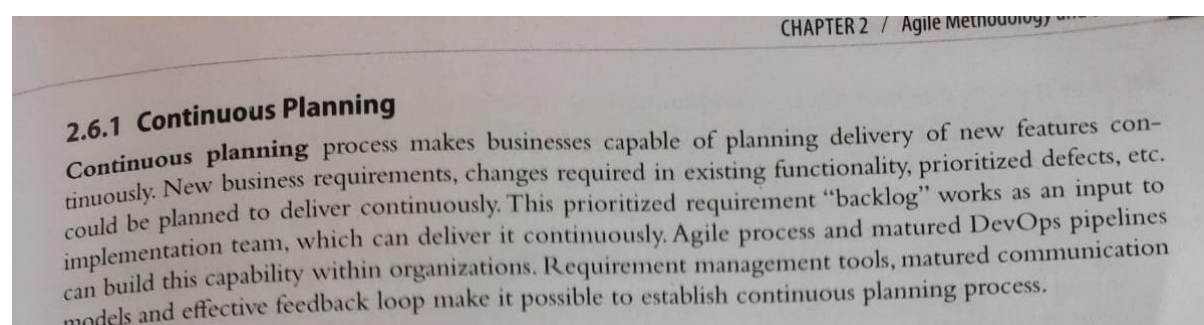
What is DevOps Life cycle?

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.

The DevOps lifecycle includes seven phases as given below:



1) Continuous Business Planning:



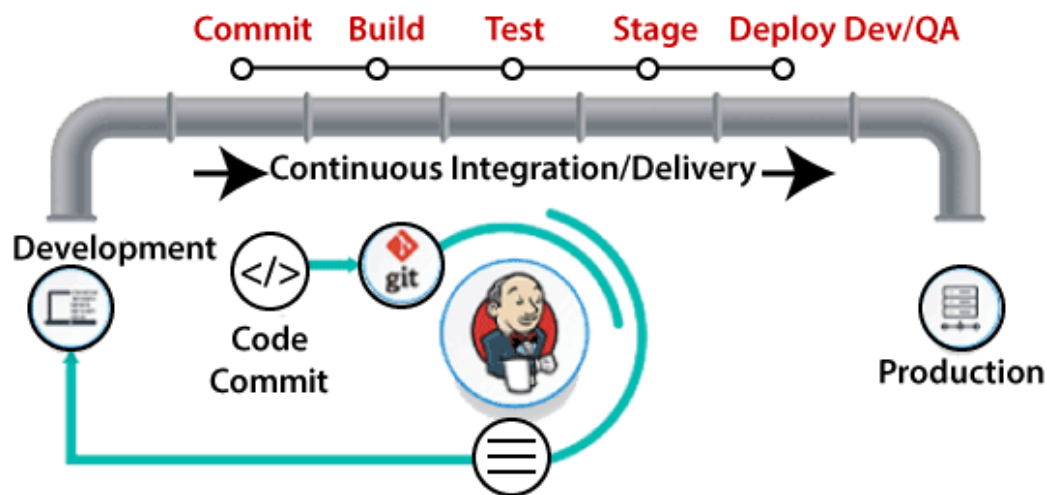
1) Continuous Development

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**.

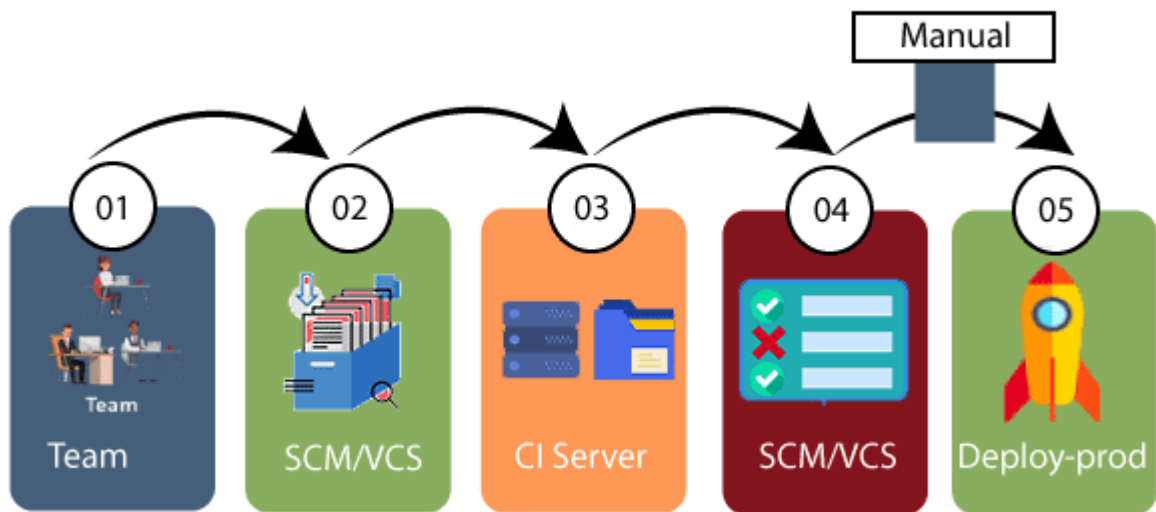
The code supporting new functionality is continuously integrated with the existing code. Therefore, there is continuous development of software. The updated code needs to be integrated continuously and smoothly with the systems to reflect changes to the end-users.



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.

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.

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 Delivery and 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.

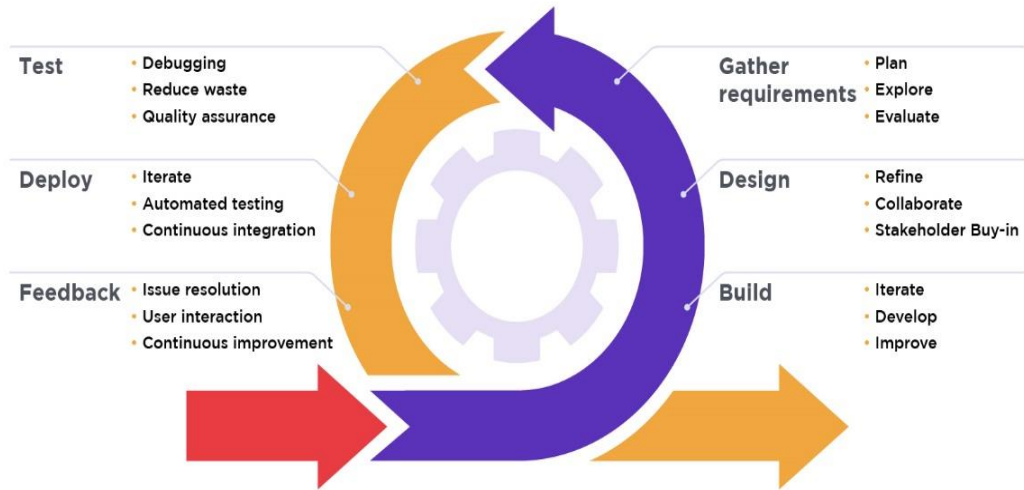
The continuity is the essential factor in the DevOps as it removes the unnecessary steps which are required to take a software application from development, using it to find out its issues and then producing a better version. It kills the efficiency that may be possible with the app and reduce the number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

7) Continuous Operations

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

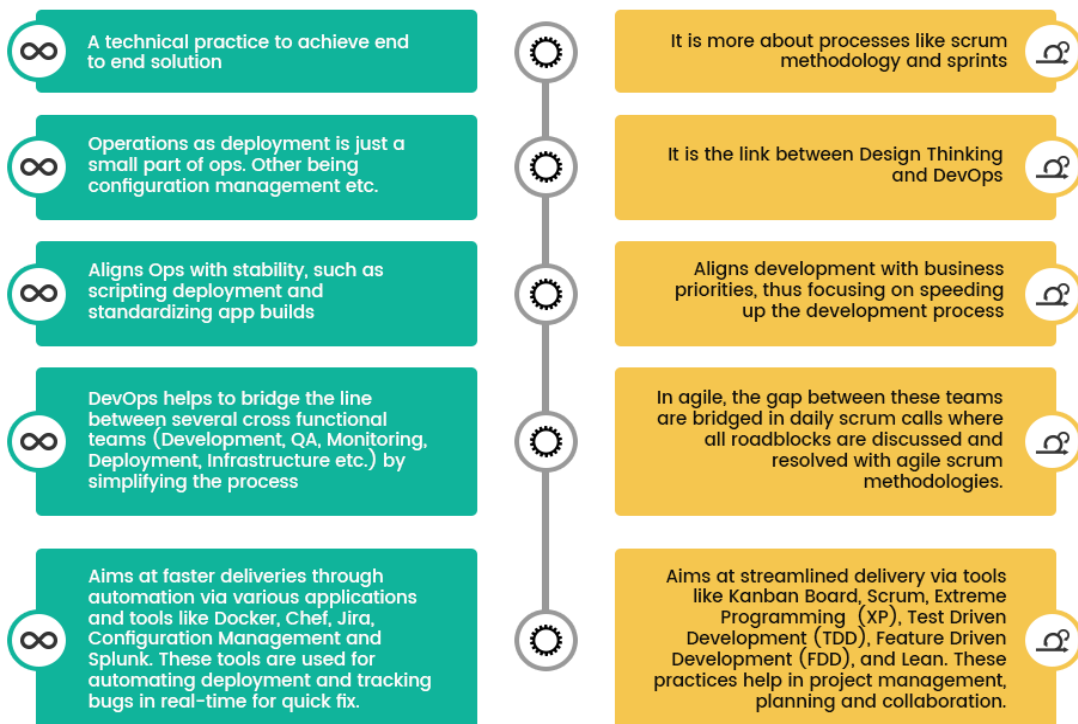
It is clear from the discussion that continuity is the critical factor in the DevOps in removing steps that often distract the development, take it longer to detect issues and produce a better version of the product after several months. With DevOps, we can make any software product more efficient and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

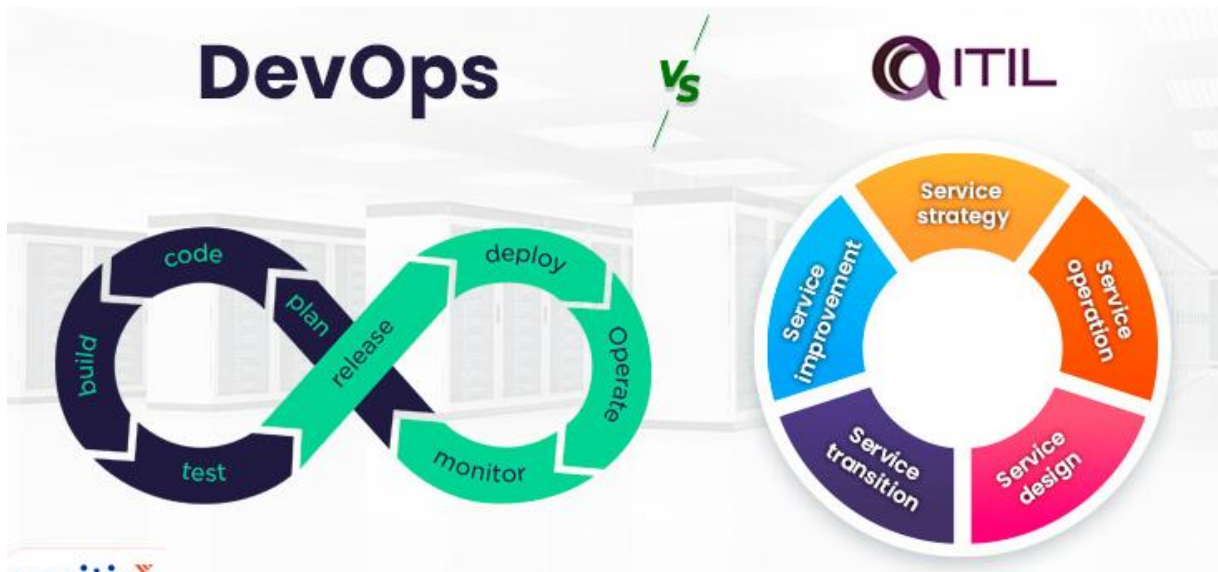
Stages of Agile Modelling





What's the difference?

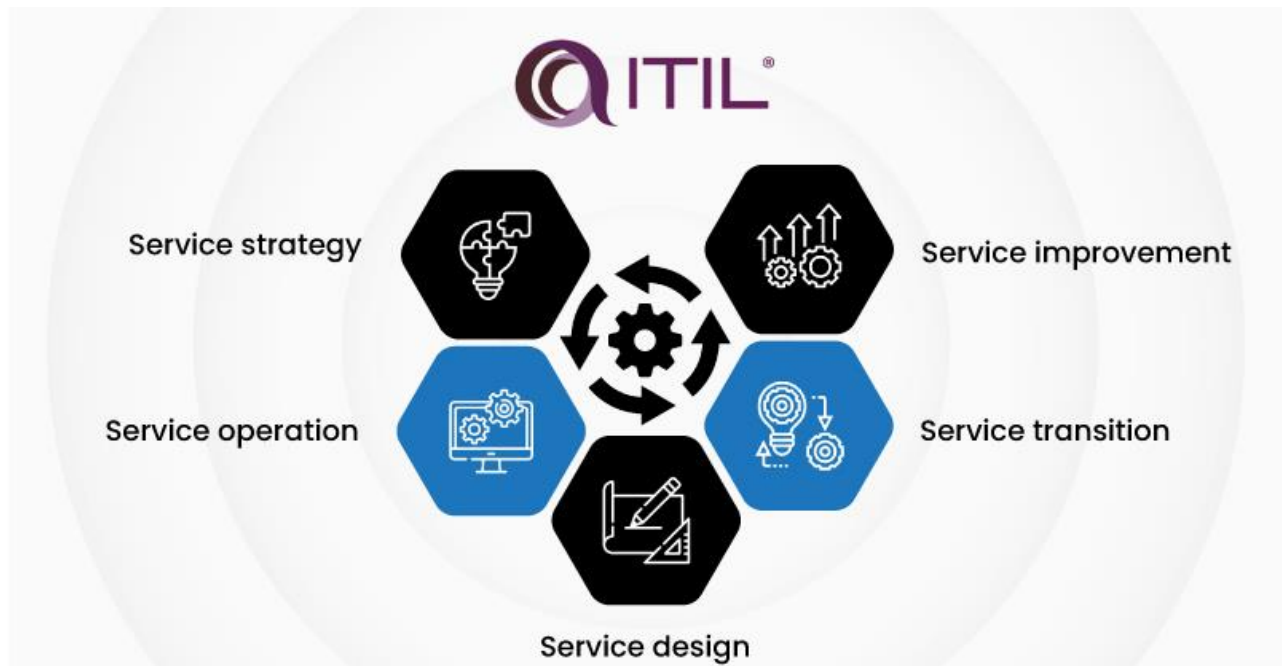




ITIL basics

- ITIL (Information Technology Infrastructure Library) is a set of principles for IT service management. It is a highly structured model built to boost productivity and offer statistics for IT teams. ITIL is a subset of ITSM; it primarily focuses on procedures for delivering, managing, and improving [IT services to businesses](#) and customers. It improves predictability and IT service delivery efficiency.
- ITIL defines and documents IT processes and procedures with a large focus on oversight and planning.
- ITIL (Information Technology Infrastructure Library) is a set of principles for IT service management. The guidelines cover best practices and tried-and-true processes for everything from [incident management](#) to [problem management](#) to [change management](#).
- The latest version, [ITIL 4](#), which debuted in 2019, continues to provide the guidance needed for organizations to address new service management challenges, using different technology in the era of [digital transformation](#), cloud, and DevOps.
- Businesses today continue to use ITIL as a high-level guide to support service management improvement initiatives. ITIL 4 puts a strong focus on customer experience and value.
- ITIL is so much more than documentation. It doesn't have to slow teams down. The core practices are sound and proven—and when approached in an agile way, they can streamline instead of clogging up the IT pipeline.

It is a framework for service management and is involved in five key stages. Let's dig in.



1) Service strategy

All managers follow instructions to develop a service strategy that ensures the company can manage all associated costs and risks. There are multiple roles involved in service strategy, and they can be defined as follows.

- A) Business relationship manager
- B) Finance manager
- C) IT steering group (ISG)
- D) Demand manager
- E) Service Strategy manager
- F) Service Portfolio manager

2) Service operation

Service operation includes technical support teams and application management that respond when an issue has an impact on the business.

3) Service design

It involves when the service's architecture is developed, and the business needs are translated into technical requirements.

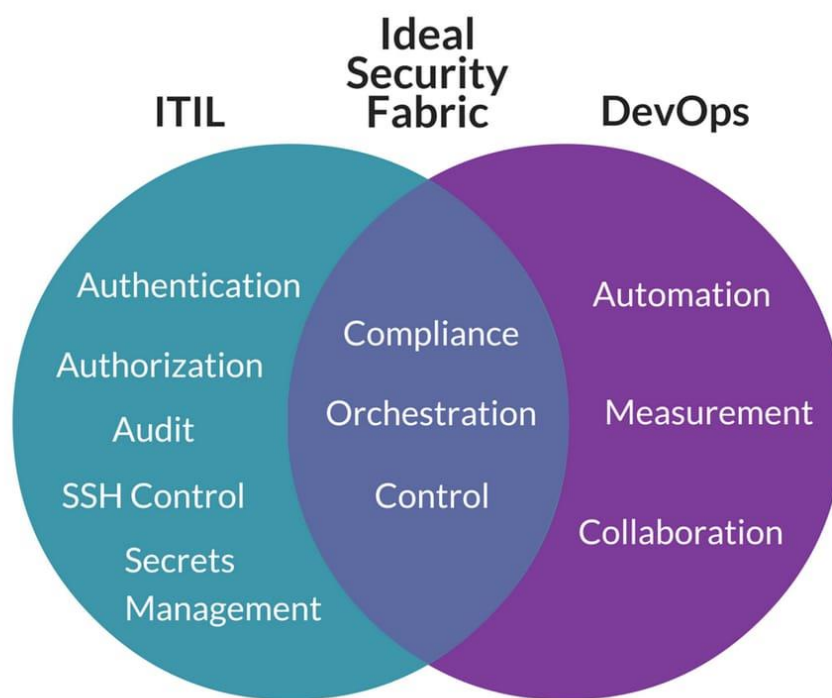
4) Service transition

In this stage, all assets are controlled to deliver a complete service for testing and integration.

5) Service improvement

It is a reflective approach that involves four stages to check the services are always in line with the demands of the business.

Can ITIL & DevOps be used together/Co-exist?



From a purely DevOps perspective, ITIL might seem like a big set of speed bumps that is going to do nothing but slow everything down and get in the way. And, initially, this might be true.

After all, the continuous processes and flexible relationships between teams using DevOps directly conflicts with the more specific principles of ITIL. However, DevOps does have the potential to expand upon the tried-and-true ITIL framework and strengthen it for better business outcomes.

ITIL also comes to the table with many benefits that can improve the DevOps mindset. With its highly understood processes and well-defined principles, ITIL can provide a different set of approaches as DevOps continues to [evolve across the enterprise](#).

At the end of the day, ITIL and DevOps *can* happily co-exist. By utilizing the best practices of each, organizations can find the balance of traditional procedures with forward-thinking agility and speed.

By implementing them in the right way can best benefit your business, both ITIL and DevOps are all aimed at the same goal: increased efficiency and enhanced collaboration. [DevOps](#) and [ITIL](#) are not mutually exclusive. They can be complementary approaches—each bringing its own benefits to the table. Agility and collaboration. Process and control. A mixed approach can benefit from the strengths of both.

The key for companies to use ITIL and DevOps successfully together is to adapt ITIL to work within the context of DevOps—without slowing it down.

[Misconceptions about DevOps vs. ITIL](#)

1. DevOps can replace ITIL

2. DevOps = continuous development, integration, and automated delivery

3. ITIL/ITSM is always documentation-heavy with onerous processes that slow teams down

4. ITIL/ITSM is only for large companies

Blue/green deployment Strategy:

- A blue/green deployment is a deployment strategy in which you create two separate, but identical environments.
- One environment (blue) is running the current application version and one environment (green) is running the new application version.
- Using a blue/green deployment strategy increases application availability and reduces deployment risk by simplifying the rollback process if a deployment fails. Once testing has been completed on the green environment, live application traffic is directed to the green environment and the blue environment is deprecated.
- A number of AWS deployment services support blue/green deployment strategies including Elastic Beanstalk, OpsWorks, CloudFormation, CodeDeploy, and Amazon ECS.
- Blue/green deployments provide releases with near zero-downtime and rollback capabilities. The fundamental idea behind blue/green deployment is to shift traffic between two identical environments that are running different versions of your application.
- The blue environment represents the current application version serving production traffic. In parallel, the green environment is staged running a different version of your application. After the green environment is ready and tested, production traffic is redirected from blue to green. If any problems are identified, you can roll back by reverting traffic back to the blue environment

Benefits of blue/green:

- Traditional deployments with in-place upgrades make it difficult to validate your new application version in a production deployment while also continuing to run the earlier version of the application.
- Blue/green deployments provide a level of isolation between your blue and green application environments. This helps ensure spinning up a parallel green environment

does not affect resources underpinning your blue environment. This isolation reduces your deployment risk.

- After you deploy the green environment, you have the opportunity to validate it. You might do that with test traffic before sending production traffic to the green environment, or by using a very small fraction of production traffic, to better reflect real user traffic. This is called canary analysis or canary testing. If you discover the green environment is not operating as expected, there is no impact on the blue environment. You can route traffic back to it, minimizing impaired operation or downtime and limiting the blast radius of impact.
- Blue/green deployments also work well with continuous integration and continuous deployment (CI/CD) workflows, in many cases limiting their complexity. During the deployment, you can scale out the green environment as more traffic gets sent to it and scale the blue environment back in as it receives less traffic. Once the deployment succeeds, you decommission the blue environment and stop paying for the resources it was using.

Scrum:



- Scrum is an agile project management framework that helps team structure and manage their work through a set of values, principles, and practices.
- Scrum encourages teams to learn through experiences, self-organize while working on a problem, and reflect on their wins and losses to continuously improve.
- Scrum is an agile development methodology used in the development of Software based on an iterative and incremental processes.

- Scrum is adaptable, fast, flexible and effective agile framework that is designed to deliver value to the customer throughout the development of the project.
- The primary objective of Scrum is to satisfy the customer's need through an environment of transparency in communication, collective responsibility and continuous progress.
- While scrum is structured, it is not entirely rigid. Its execution can be tailored to the needs of any organization.

Scrum artifacts

Scrum artifacts are important information used by the scrum team that helps define the product and what work to be done to create the product.

1. **Product Backlog**
2. **Sprint Backlog**
3. **Increment** (or Sprint Goal)

Definition: In Agile, Cadence is the number of days or weeks allocated in a Sprint or Release. In other way round, it is the length of the team's development cycle.

Different Roles in Scrum:

In Scrum, the team focuses on building quality software.

The Scrum team consists of the following roles:

Scrum master: The person who leads the team guiding them to comply with the rules and processes of the methodology. The Scrum Master is in charge of keeping Scrum up to date, providing coaching, mentoring and training to the teams in case it needs it.

Product owner (PO): Is the representative of the stakeholders and customers who use the software. They focus on the business part and is responsible for the ROI of the project.

Team: A group of professionals with the necessary technical knowledge who develop the project jointly carrying out the stories they commit to at the start of each sprint.

Benefits of Scrum Methodology

Scrum has many advantages , It is currently the most used and trusted framework of reference in the software industry. Below are some of the known benefits of Scrum:

- **Easily Scalable:** Scrum processes are iterative and are handled within specific work periods, which makes it easier for the team to focus on definite functionalities for each period.
- **Compliance of expectations:** The client establishes their expectations indicating the value about each requirement; the team estimates them and with this information the Product Owner establishes its priority.
- **Flexible to changes:** Quick reaction to changes in requirements generated by customer needs or market developments. The methodology is designed to adapt to the changing requirements that complex projects entail.
- **Time to Market reduction:** The client can start using the most important functionalities of the project before the product is completely ready.
- **Higher software quality:** The working method and the need to obtain a functional version after each iteration, helps to obtain a higher quality software.

- **Timely Prediction:** Using this methodology, we know the average speed of the team by sprint (story points), with which, consequently, it is possible to estimate when a certain functionality that is still in the backlog will be available.
- **Reduction of risks:** The fact of carrying out the most valuable functionalities in the first place and of knowing the speed with which the team advances in the project, allows to clear risks effectively in advance.

What are the six Scrum principles?

The Six Scrum Principles are:

1. Control over the empirical process
2. Self-organization
3. Collaboration
4. Value-based prioritization
5. Time-boxing
6. Iterative development

Devops software Architecture:

DevOps architecture is the combination of tools, processes, and practices that help organizations to rapidly produce software products and services. It typically includes:

Continuous integration and continuous delivery (CI/CD) pipeline - automates the process of building, testing, and deploying software.

Configuration management tools - manage and automate infrastructure, such as servers, databases, and network devices.

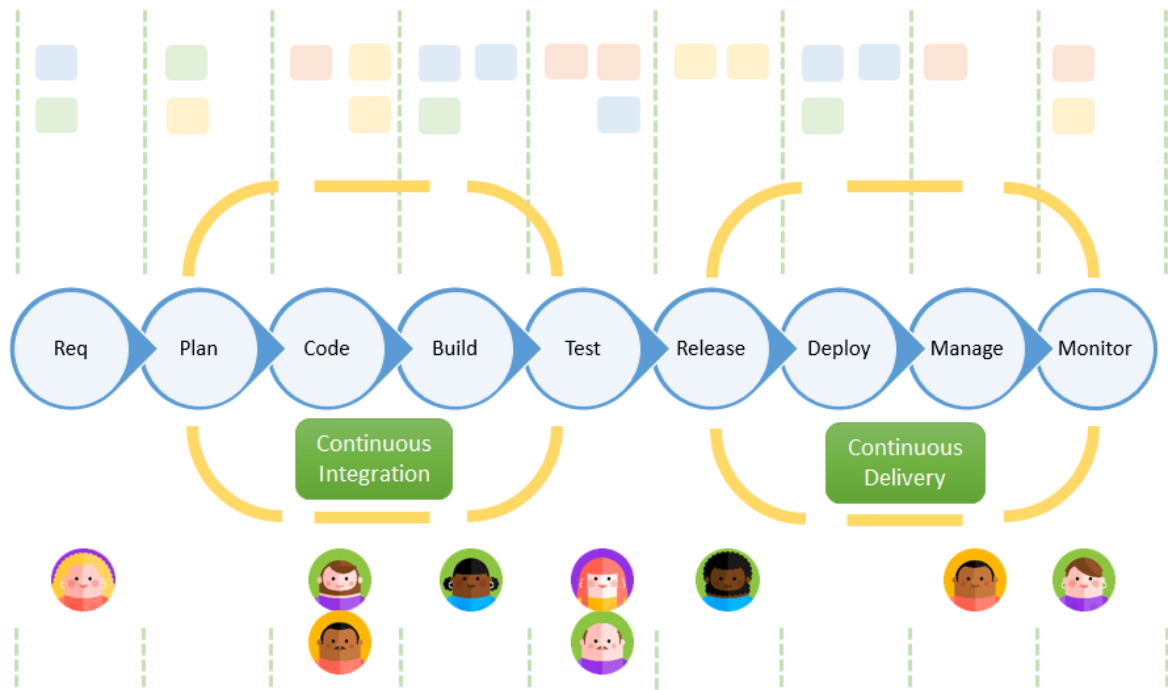
Monitoring and logging tools - track the performance and health of applications and infrastructure.

Containers and container orchestration - package and deploy applications consistently across environments.

Microservices - break down monolithic applications into smaller, loosely-coupled services that can be developed, deployed, and scaled independently.

These components work together to allow teams to quickly and reliably release new features and fixes, respond to incidents, and deliver better software faster.

Kanban



- Kanban is a popular framework used to implement [agile](#) and [DevOps](#) software development.
- It requires real-time communication of capacity and full transparency of work. Work items are represented visually on a [kanban board](#), allowing team members to see the state of every piece of work at any time.



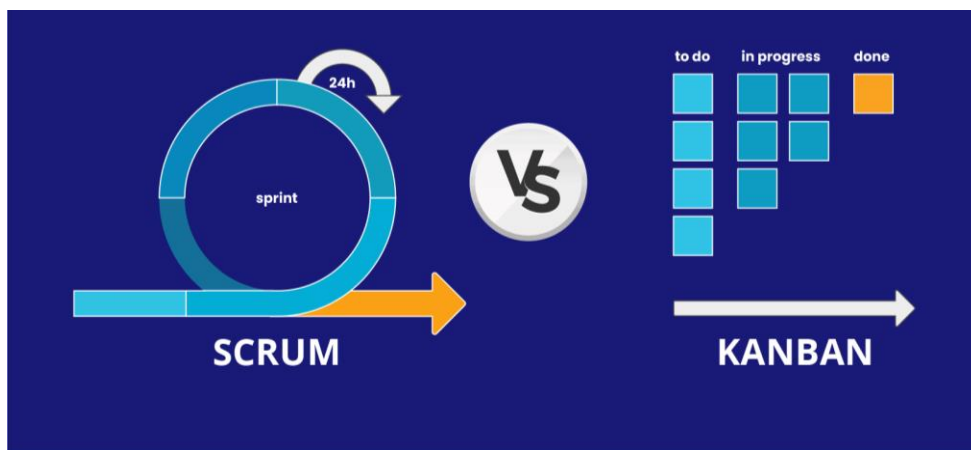
- Kanban is a visual system for managing work as it moves through a process. Kanban visualizes both the process (the workflow) and the actual work passing through that process.

- The goal of Kanban is to identify potential bottlenecks in your process and fix them, so work can flow through it cost-effectively at an optimal speed or throughput.

The benefits of kanban

Kanban is one of the most popular software development methodologies adopted by agile teams today. Kanban offers several additional advantages to task planning and throughput for teams of all sizes.

1. **Planning flexibility**
2. **Shortened time cycles**
3. **Fewer bottlenecks**
4. **Visual metrics**
5. **Continuous delivery**



Scrum vs. kanban

Kanban and [scrum](#) share some of the same concepts but have very different approaches. They should not be confused with one another.

	SCRUM	KANBAN
Cadence	Regular fixed length sprints (ie, 2 weeks)	Continuous flow
Release methodology	At the end of each sprint if approved by the product owner	Continuous delivery or at the team's discretion
Roles	Product owner, scrum master, development team	No existing roles. Some teams enlist the help of an agile coach.
Key metrics	Velocity	Cycle time
Change philosophy	Teams should strive to not make changes to the sprint forecast during the sprint. Doing so compromises learnings around estimation.	Change can happen at any time



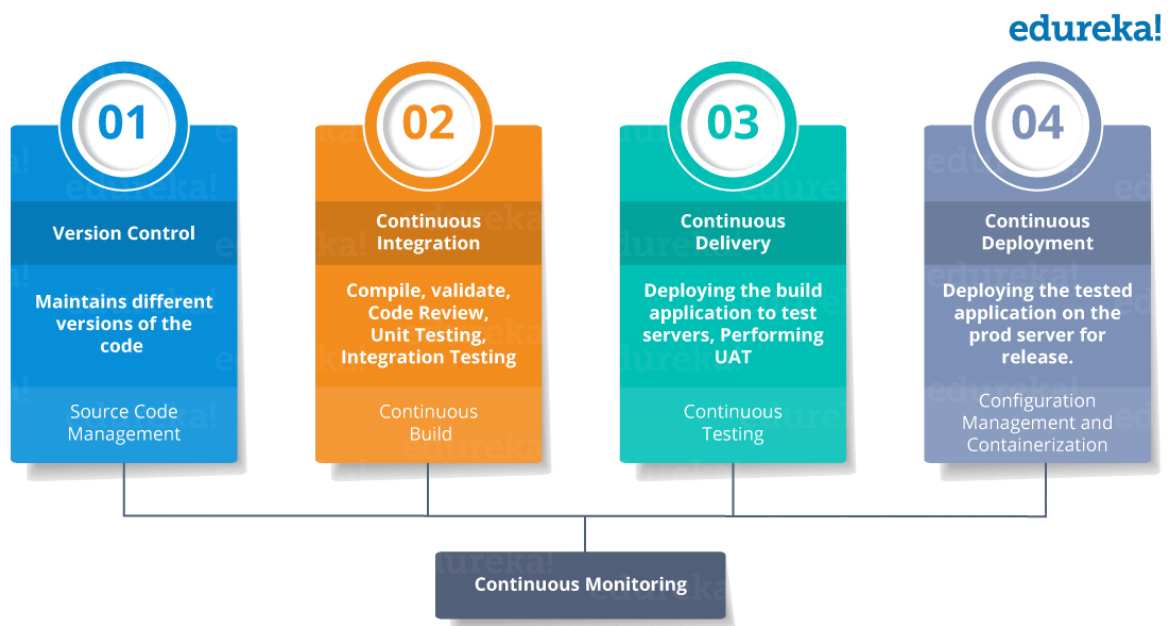
Scrum and Kanban

A comparison of Agile methodologies

	Scrum	Kanban
Origin	Software development	Lean manufacturing
Ideology	Solve complex problems while delivering valuable products	Use visuals to improve work flows and processes
Practices	Sprint planning Sprint Daily scrum Sprint review Sprint retrospective	Visualize the flow of work Limit work in progress Manage flow Make process policies explicit Implement feedback loops Improve, experiment
Roles	Product Owner Scrum Master Development Team	No formal roles
Metrics	Velocity	Cycle time Throughput

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

Delivery pipeline

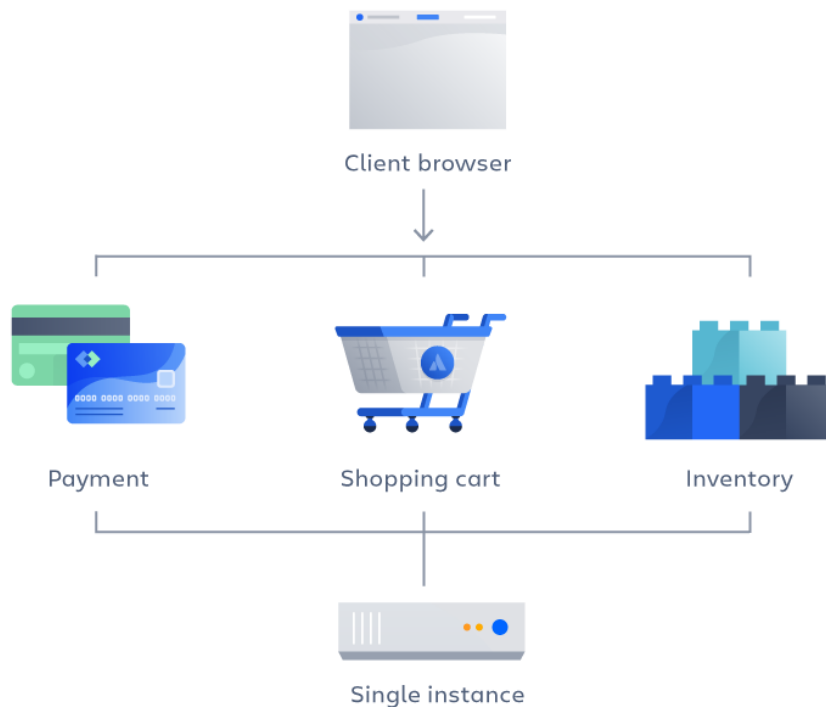


What is a monolithic architecture?

1. A monolithic architecture is a traditional model of a software program, which is built as a unified unit that is self-contained and independent from other applications.
2. The word “monolith” is often attributed to something large and glacial.
3. A monolithic architecture is a singular, large computing network with one code base that couples all of the business concerns together.
4. To make a change to this sort of application requires updating the entire stack by accessing the code base and building and deploying an updated version of the service-side interface. This makes updates restrictive and time-consuming.

Monoliths can be convenient early on in a project's life for ease of code management, cognitive overhead, and deployment. This allows everything in the monolith to be released at once.

Monolithic architecture



Advantages of a monolithic architecture

When developing using a monolithic architecture, the primary advantage is fast development speed due to the simplicity of having an application based on one code base. The advantages of a monolithic architecture include:

Easy deployment – One executable file or directory makes deployment easier.

Development – When an application is built with one code base, it is easier to develop.

Performance – In a centralized code base and repository, one API can often perform the same function that numerous APIs perform with microservices.

Simplified testing – Since a monolithic application is a single, centralized unit, end-to-end testing can be performed faster than with a distributed application.

Easy debugging – With all code located in one place, it's easier to follow a request and find an issue.

Disadvantages of a monolithic architecture

As with the case of Netflix, monolithic applications can be quite effective until they grow too large and scaling becomes a challenge. Making a small change in a single function requires compiling and testing the entire platform, which goes against the agile approach today's developers favor.

The disadvantages of a monolith include:

Slower development speed – A large, monolithic application makes development more complex and slower.

Scalability – You can't scale individual components.

Reliability – If there's an error in any module, it could affect the entire application's availability.

Barrier to technology adoption – Any changes in the framework or language affects the entire application, making changes often expensive and time-consuming.

Lack of flexibility – A monolith is constrained by the technologies already used in the monolith.

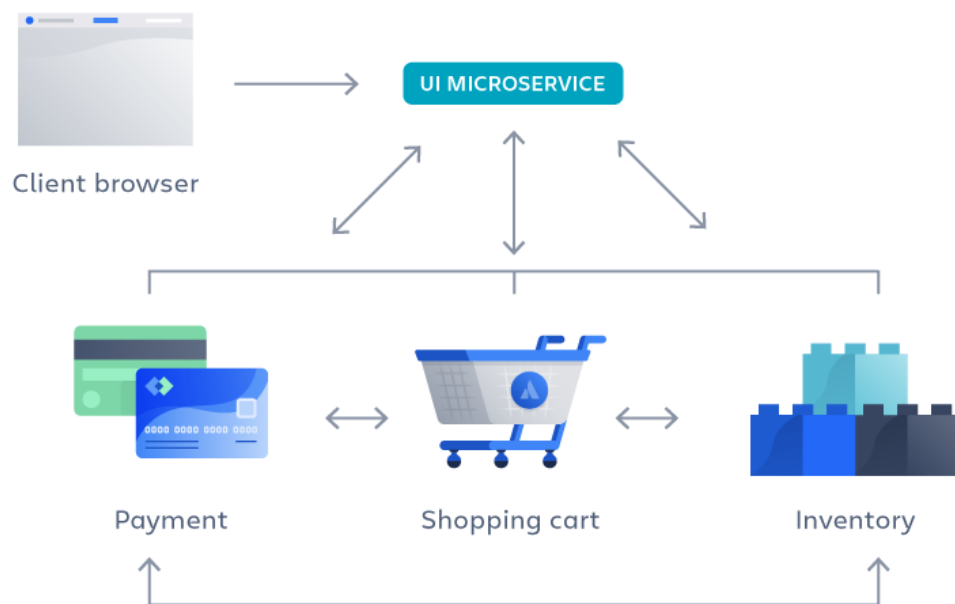
Deployment – A small change to a monolithic application requires the redeployment of the entire monolith.

What are microservices?

1. A microservices architecture, also simply known as microservices, is an architectural method that relies on a **series of independently deployable services**.
2. These services have their own business logic and database with a specific goal. Updating, testing, deployment, and scaling occur within each service.
3. Microservices decouple major business, domain-specific concerns into separate, independent code bases.
4. Microservices don't reduce complexity, but they make any complexity visible and more manageable by separating tasks into smaller processes that function independently of each other and contribute to the overall whole.

Adopting microservices often goes hand in hand with DevOps, since they are the basis for [continuous delivery](#) practices that allow teams to adapt quickly to user requirements.

Microservice architecture



Advantages of microservices:

Microservices are by no means a silver bullet, but they solve a number of problems for growing software and companies. Since a microservices architecture consists of units that run independently, each service can be developed, updated, deployed, and scaled without affecting the other services. Software updates can be performed more frequently, with improved reliability, uptime, and performance.

In short, the advantages of microservices are:

Agility – Promote agile ways of working with small teams that deploy frequently.

Flexible scaling – If a microservice reaches its load capacity, new instances of that service can rapidly be deployed to the accompanying cluster to help relieve pressure.

Continuous deployment – We now have frequent and faster release cycles. Before we would push out updates once a week and now we can do so about two to three times a day.

Highly maintainable and testable – Teams can experiment with new features and roll back if something doesn't work. This makes it easier to update code and accelerates time-to-market for new features. Plus, it is easy to isolate and fix faults and bugs in individual services.

Independently deployable – Since microservices are individual units they allow for fast and easy independent deployment of individual features.

Technology flexibility – Microservice architectures allow teams the freedom to select the tools they desire.

High reliability – You can deploy changes for a specific service, without the threat of bringing down the entire application.

Happier teams – teams who work with microservices are a lot happier, since they are more

autonomous and can build and deploy themselves without waiting weeks for a pull request to be approved.

Disadvantages of microservices **can include:**

Development sprawl – Microservices add more complexity compared to a monolith architecture, since there are more services in more places created by multiple teams. If development sprawl isn't properly managed, it results in slower development speed and poor operational performance.

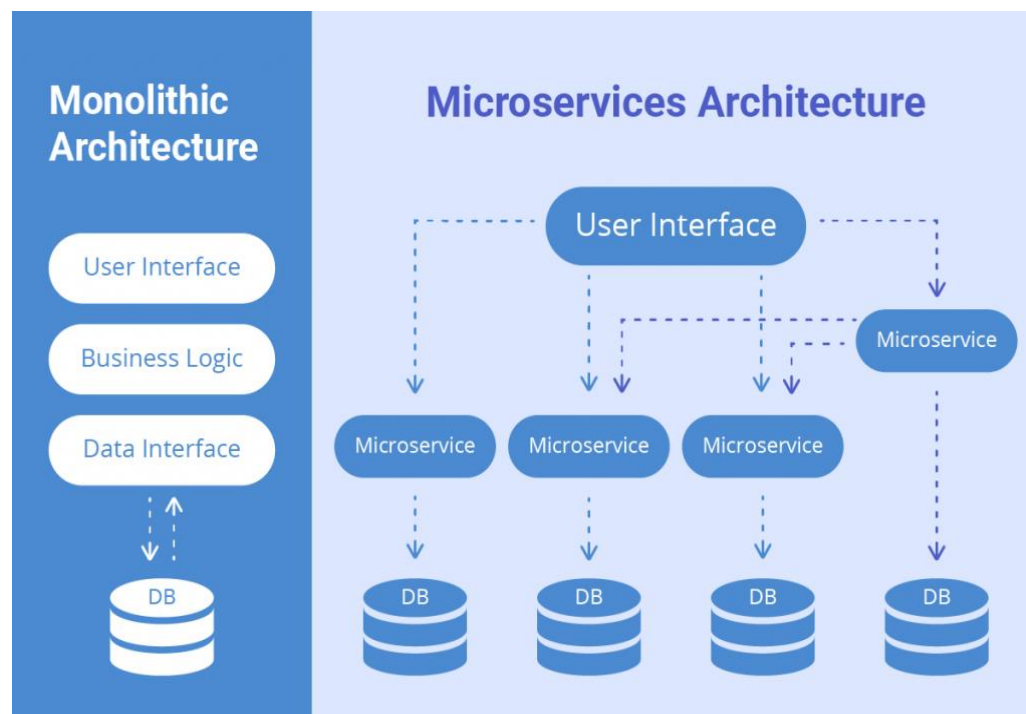
Exponential infrastructure costs – Each new microservice can have its own cost for test suite, deployment playbooks, hosting infrastructure, monitoring tools, and more.

Added organizational overhead – Teams need to add another level of communication and collaboration to coordinate updates and interfaces.

Debugging challenges – Each microservice has its own set of logs, which makes debugging more complicated.

Lack of standardization – Without a common platform, there can be a proliferation of languages, logging standards, and monitoring.

Lack of clear ownership – As more services are introduced, so are the number of teams running those services. Over time it becomes difficult to know the available services a team can leverage and who to contact for support.



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

Resilience

1. Resilience is about the creation and sustaining of various conditions that enable systems to adapt to unforeseen events.

2. Resilient DevOps teams are constantly improving, evolving, and finding better ways to resolve organizational challenges. This resilience allows organizations to achieve greater reliability, continuity, and compliance — all while reaping the benefits of faster release cycles.

3. “Resilience” is about what a system can do — including its capacity:


to anticipate — seeing developing signs of trouble ahead to begin to adapt early and reduce the risk of decompensation.

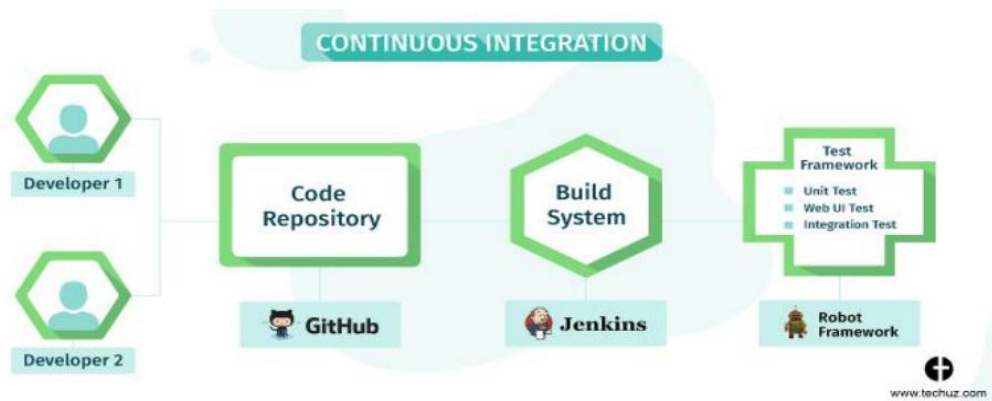
to synchronize — adjusting how different roles at different levels coordinate their activities to keep pace with tempo of events and reduce the risk of working at cross purposes

to be ready to respond — developing deployable and mobilizable response capabilities in advance of surprises and reduce the risk of brittleness.

for proactive learning — learning about brittleness and sources of resilient performance before major collapses or accidents occur by studying how surprises are caught and resolved.

CI:

Continuous integration  is a software development practice where members of a team use a version control system and frequently integrate their work to the same location, such as a main branch. Each change is built and verified to detect integration errors as quickly as possible. Continuous integration is focused on automatically building and testing code, as compared to *continuous delivery*, which automates the entire software release process up to production.




What You Need to Practice

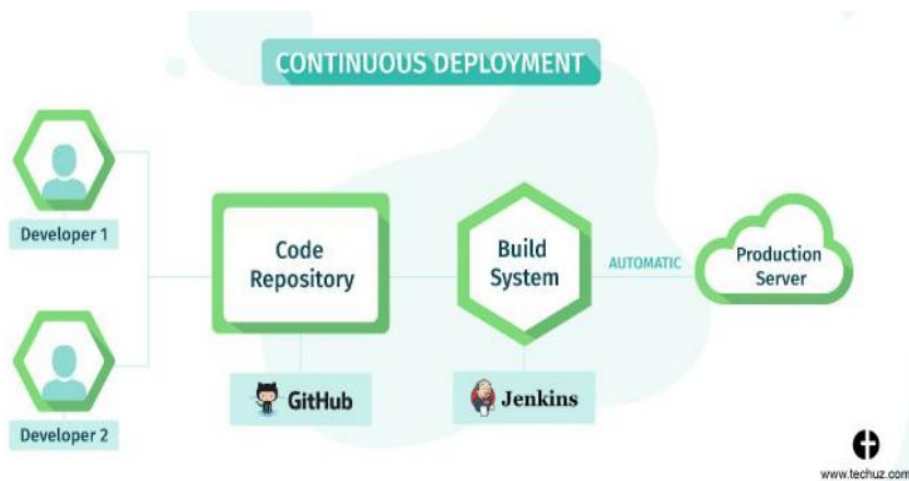
- Maintain a common code repository where the developers can commit code — such as GitHub and Gitlab.
- Automate the build.
- Configure the automated testing on the built package.
- Developers must commit the new code as often as possible typically several times a day.

Benefits of Continuous Integration

- Automated tests detect bugs at an early stage so that you can ship a clean final product to the customers.
- Continuous Integration makes the release way easier and helps to avoid the integration hell.
- Automated tests can run a number of tests at a time that reduces the overall cost of testing.
- Even the quality analysts can focus on improving the quality of application cohesively rather spending time on testing and finding bugs.

CD:

Continuous delivery  is a software development methodology where the release process is automated. Every software change is automatically built, tested, and deployed to production. Before the final push to production, a person, an automated test, or a business rule decides when the final push should occur. Although every successful software change can be immediately released to production with continuous delivery, not all changes need to be released right away.



Benefits of Continuous Delivery

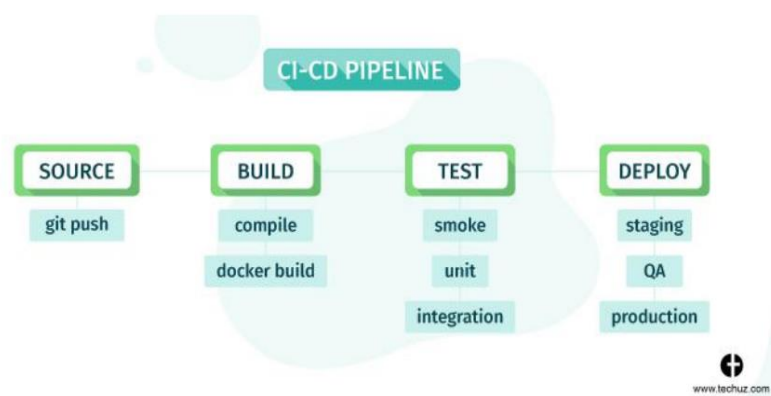
- Continuous Delivery makes the application deployment painlessly easier and the team no longer have to spend time on the release.
- It also reduces the risk of deploying the application with bugs as they are detected early on the production-like mock environment.
- Research has shown that CD making the releases easy and efficient reduces the burnout and ultimately makes the team happier.
- Releases can be made quicker and more often resulting in faster customer feedback and improving the product.

What You Need to Practice

- Your continuous integration must work well and the automated tests must cover enough codebase.
- Once the deployment is triggered, it must be fully automated without any need for human intervention.
- Incorporation of blue-green deployment or feature flags so that the incomplete features do not create downtime and affect the users.

What is CI/CD Pipeline?

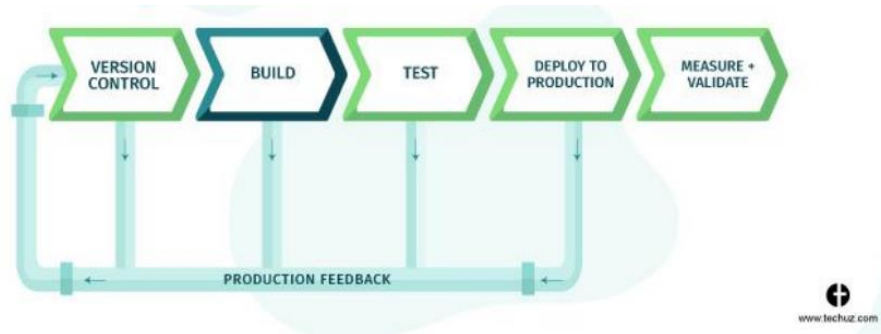
- Now you know that CI/CD is the strategies to ship well-tested applications frequently to the customers — and making the development process fast and efficient. However, the success of this discipline relies on a reliable CI/CD pipeline.
- A CI/CD pipeline is the structure that automates the steps of the application delivery process. Since the process is automated, it removes the chances of committing any manual error and provides a quick product iteration. From a high-level view, the CI/CD pipeline includes steps such as — code commit, build, automated testing and deployment on staging and production server.



1. Commit

The pipeline starts with a common code repository where multiple developers working on the application commits their code several times a day. Some of the popular and widely used code repositories

are [GitHub](#), [BitBucket](#), [GitLab](#), [Beanstalk](#) and [SourceForge](#).



2. Build

Now, the changes are built using a built management system. The source code is combined to build a runnable component of the product.

3. Automate Testing

- In this step, automated tests are run to check the codes. Automated testing plays an important role in the entire CI/CD pipeline and the success of delivering bug-free application depends on it.
- The code goes through varied tests depending on the complexity and needs of the project. We'll discuss the type of tests in a few moments. Popular tools for automated testing are [Telerik Test Studio](#), [Selenium](#), [Robotium](#), [SoapUI](#) and [TestDrive](#).

4. Deploy

- Once the built passes through the automated tests, they're ready to be deployed. The deployment can be either on the staging server, where the application is reviewed and additionally tested in the mock environment that is similar to the production environment.
- Or can be directly deployed onto the production server and made available to the users. [Jenkins](#), [Atlassian Bamboo](#), [AWS CodeDeploy](#) and [CICircle](#) are the majorly used tools.

Conclusion:

- The entire CI/CD pipeline is automated based on the needs of the project and organization.
- You can either adopt Continuous Delivery where the changes are uploaded on the mock server and then onto the production server or the highest level of automation — Continuous Deployment, where the changes are reflected in the production server automatically once the code is committed and passes the test.

Benefits of CI/CD:

1. Smaller code changes
2. Fault isolations
3. Faster mean time to resolution (MTTR)
4. More test reliability
5. Faster release rate
6. Smaller backlog
7. Customer satisfaction
8. Increase team transparency and Accountability
9. Reduce costs
10. Easy maintenance and updates

The different types of Testing:

- Test automation is the practice of automatically reviewing and validating a software product, such as a web application, to make sure it meets predefined quality standards, functionality (business logic), and user experience.
- Test automation helps development teams build, test, and ship faster and more reliably.

1. Unit tests

- Unit tests are very low-level and close to the source of an application. They consist in testing individual methods and functions of the classes, components, or modules used by your software.
- Unit tests are generally quite cheap to automate and can run very quickly by a continuous integration server.

2. Integration tests

- Integration tests verify that different modules or services used by your application work well together.
- For example, it can be testing the interaction with the database or making sure that microservices work together as expected.
- These types of tests are more expensive to run as they require multiple parts of the application to be up and running.

3. Functional tests

- Functional tests focus on the business requirements of an application. They only verify the output of an action and do not check the intermediate states of the system when performing that action.
- There is sometimes a confusion between integration tests and functional tests as they both require multiple components to interact with each other. The difference is that an integration test may simply verify that you can query the database while a functional test would expect to get a specific value from the database as defined by the product requirements.

4. End-to-end tests

- End-to-end testing replicates a user behaviour with the software in a complete application environment. It verifies that various user flows work as expected and can be as simple as loading a web page or logging in or much more complex scenarios verifying email notifications, online payments, etc...
- End-to-end tests are very useful, but they're expensive to perform and can be hard to maintain when they're automated. It is recommended to have a few key end-to-end tests and rely more on lower-level types of testing (unit and integration tests) to be able to quickly identify breaking changes.

5. Acceptance testing

- Acceptance tests are formal tests that verify if a system satisfies business requirements. They require the entire application to be running while testing and focus on replicating user behaviour.
- But they can also go further and measure the performance of the system and reject changes if certain goals are not met.

6. Performance testing

- Performance tests evaluate how a system performs under a particular workload. These tests help to measure the reliability, speed, scalability, and responsiveness of an application.
- For instance, a performance test can observe response times when executing a high number of requests, or determine how a system behaves with a significant amount of data.
- It can determine if an application meets performance requirements, locate bottlenecks, measure stability during peak traffic, and more.

7. Smoke testing

- Smoke tests are basic tests that check the basic functionality of an application. They are meant to be quick to execute, and their goal is to give you the assurance that the major features of your system are working as expected.

- Smoke tests can be useful right after a new build is made to decide whether or not you can run more expensive tests, or right after a deployment to make sure that they application is running properly in the newly deployed environment.

8. **Exploratory testing:** takes an unstructured approach to reviewing numerous areas of an application from the user perspective, to uncover functional or visual issues

Difference between Mercurial and bazaar

Mercurial and Bazaar are both version control systems used for software development and source code management.

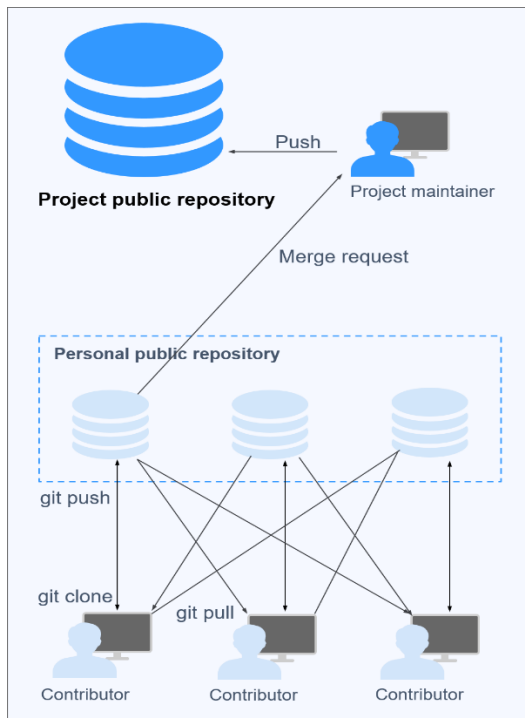
The key differences between the two are:

- **Architecture:** Mercurial uses a distributed architecture, meaning that every repository contains a full copy of the project history, while Bazaar uses a centralized architecture, where a central repository holds the history and changes are pushed to it.
- **Workflow:** Mercurial's workflow is focused on individual commits and changesets, while Bazaar emphasizes branching and merging, making it easier to work on multiple parallel streams of development.
- **Speed:** Mercurial is generally faster than Bazaar, especially when working with large projects with a lot of history.
- **Community:** Mercurial has a larger and more active user community, with a wider range of tools and extensions available, while Bazaar has a smaller user base and a more limited range of available tools.

Ultimately, the choice between Mercurial and Bazaar will depend on your specific needs and workflow.

Forking Workflow [strategy](#)

- The forking workflow is suitable for outsourcing, crowdsourcing, crowdfunding, and open-source projects.
- One of the features that distinguish this workflow is that every contracting developer has a personal public repository, which is forked from the project public repository.
- Developers can perform operations on the forks without the need of being authorized by the project maintainer. The following figure shows the process of the forking workflow.



Process

1. Developers fork the project public repository to create personal public ones.
2. The personal public repositories are cloned to their local computers for development.
3. After the development is complete, developers push changes to their personal public repositories.
4. Developers file merge requests to the project maintainer for merge to the project public repository.
5. The project maintainer pulls changes to the local computer and reviews the code. If the code is approved, it is pushed to the project public repository.

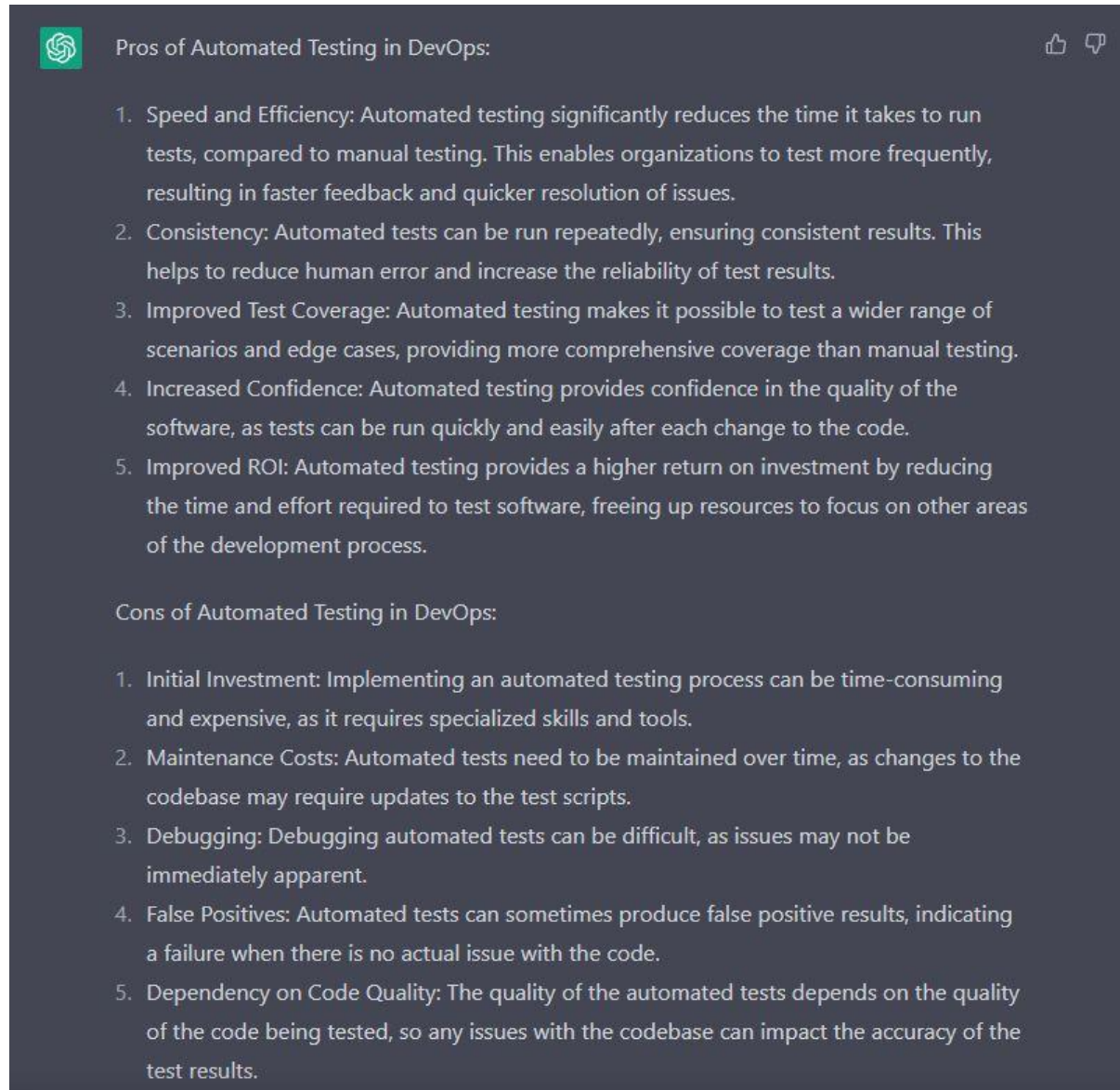
Advantages

- Code collaboration is easier. Developers can share their code by pushing it to their personal public repositories for others to pull, unlike some workflows where developers cannot see others' work until it is merged into the project repository.
- Project maintainers do not have to grant permissions on project public repositories to every contributor.
- Merge requests serve as an important guard for code security.
- The three workflows introduced previously can be incorporated into the forking workflow based on project requirements.

Disadvantages

It takes more steps and time before the code of developers gets merged into the project repository.

Pros and Cons of automated Testing:



The screenshot shows a GitHub repository page with a dark theme. At the top, there is a green GitHub logo and the title 'Pros of Automated Testing in DevOps'. Below the title, there are two thumbs up icons. The main content is a list of five pros of automated testing, followed by a section titled 'Cons of Automated Testing in DevOps' with another list of five cons. The text is white on a dark background.

Pros of Automated Testing in DevOps:

1. **Speed and Efficiency:** Automated testing significantly reduces the time it takes to run tests, compared to manual testing. This enables organizations to test more frequently, resulting in faster feedback and quicker resolution of issues.
2. **Consistency:** Automated tests can be run repeatedly, ensuring consistent results. This helps to reduce human error and increase the reliability of test results.
3. **Improved Test Coverage:** Automated testing makes it possible to test a wider range of scenarios and edge cases, providing more comprehensive coverage than manual testing.
4. **Increased Confidence:** Automated testing provides confidence in the quality of the software, as tests can be run quickly and easily after each change to the code.
5. **Improved ROI:** Automated testing provides a higher return on investment by reducing the time and effort required to test software, freeing up resources to focus on other areas of the development process.

Cons of Automated Testing in DevOps:

1. **Initial Investment:** Implementing an automated testing process can be time-consuming and expensive, as it requires specialized skills and tools.
2. **Maintenance Costs:** Automated tests need to be maintained over time, as changes to the codebase may require updates to the test scripts.
3. **Debugging:** Debugging automated tests can be difficult, as issues may not be immediately apparent.
4. **False Positives:** Automated tests can sometimes produce false positive results, indicating a failure when there is no actual issue with the code.
5. **Dependency on Code Quality:** The quality of the automated tests depends on the quality of the code being tested, so any issues with the codebase can impact the accuracy of the test results.

Architecture rules of thumb:

There are a number of architecture rules that might help us understand how to deal with the previous undesirable situation of dealing with monolithic type of architecture. i.e.

The system is so complex that it becomes difficult to think about the effects of changes, even though they might be trivial. Now, a change is usually more complex than a simple spelling correction. Thus, we need to exercise all aspects of the deployment chain, including manual verification, for all changes to a monolith.

The separation of concerns

- The renowned Dutch computer scientist Edsger Dijkstra first mentioned his idea of how to organize thought efficiently in his paper from 1974, On the role of scientific thought.
- He called this idea "the separation of concerns". To this date, it is arguably the single most important rule in software design.
- There are many other well-known rules, but many of them follow from the idea of the separation of concerns. The fundamental principle is simply that "We should consider different aspects of a system separately".

The principle of cohesion

- In computer science, cohesion refers to the degree to which the elements of a software module belong together.
- Cohesion can be used as a measure of how strongly related the functions in a module are.
- It is desirable to have strong cohesion in a module.
- We can see that strong cohesion is another aspect of the principle of the separation of concerns.

Coupling

- Coupling refers to the degree of dependency between two modules.
- We always want low coupling between modules.
- Again, we can see coupling as another aspect of the principle of the separation of concerns.
- Systems with high cohesion and low coupling would automatically have separation of concerns, and vice versa

Scalability: Design your architecture to be scalable, both vertically and horizontally, to accommodate increasing traffic and user demand.

Resilience: Ensure that your architecture is resilient and can handle failures gracefully, by using techniques such as load balancing and failover.

Security: Implement strong security measures, such as encryption, authentication, and access control, to protect sensitive data and systems.

Monitoring: Implement monitoring and logging to provide visibility into system performance and to facilitate troubleshooting and issue resolution.

Automation: Automate routine tasks and processes, such as deployment, testing, and scaling, to increase efficiency and reduce errors.

Continuous Integration and Delivery (CI/CD): Implement a CI/CD pipeline to streamline the software development and deployment process.

Microservices: Consider using a microservices architecture to enable faster and more flexible development, deployment, and scaling of individual components.

Observability: Implement observability practices to gain insight into the behavior and performance of your systems.

Cost Optimization: Optimize your infrastructure costs by using cost-effective services, such as cloud computing and containerization.

Database Migrations: Modifying Existing Databases

Database migrations, rather than simple builds, suddenly become necessary when a production database system must be upgraded in-situ. You can't just build a new version of the database: you have to preserve the data that is there, and you need to do the upgrade in a way that doesn't disrupt the service.

What is a database migration?

A database migration involves changing it from one defined version to another.

A **migration** script alters the metadata of a database, as defined by its constituent DDL creation scripts, from one database version to another, whilst preserving the data held within it. Migration scripts can be forward, or “up”, to upgrade a database to a newer version, or backward, or “down”, to downgrade to a previous version of a database.

The primary challenge when migrating an existing database is how to handle changes to the databases tables. The migration script must alter any table in the database that has changed while preserving data. The required change may be as simple as adding or removing a table column, or a complex refactoring task such as splitting tables or changing a column in a way that affects the data it stores.

Two approaches to database migrations

There are two basic approaches to migrating databases, There is no fundamental difference between the two approaches. Both use migration scripts to do the work.

The two approaches are not mutually exclusive: Tools used to migrate a database using the state-based approach occasionally need ‘help’ defining the correct route for migrations that affect existing data. Tools that use the migrations approach often need a way to “back fill” the state of each object, at each version, so that we can see easily how the object has changed over time.

1.State-based database migrations

Second one is based on versioning object CREATE scripts, commonly referred to as the **state-based** approach,

When using the state-based technique, we store in the VCS the source DDL scripts to CREATE each database object. Each time we modify a database object, we commit to the VCS the latest creation script for that object. In other words, we are versioning the current state of each object in the database.

We then derive a migration script based on a comparison between the static definition of the database in version control (the object DDL scripts) and the target database. We will need to automate the generation of this migration script, often referred to as an **automated synchronization script**.

Sometimes it is impossible for a tool to create a synchronization script that will preserve the data correctly. Although it knows what the initial and final versions of the database look like, there is sometimes ambiguity regarding the correct transition path.

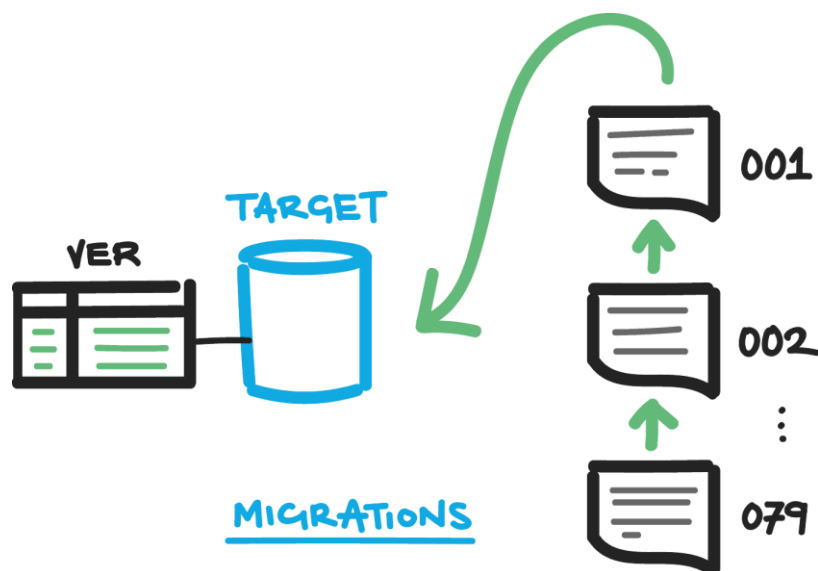
2. Change script-based migrations

Second one is based on storing in the VCS only the initial CREATE scripts for each object followed by a string of object change (ALTER) scripts, often referred to as the **migration-based** approach.

This technique migrates a database by means of one or more **change** scripts. Any DDL script that alters one or more existing database objects is a change script. For each migration between consecutive database versions, we need to store in version control the initial build scripts, plus change scripts that describe precisely the steps required to perform the necessary changes to migrate to the required database version.

An alternative scheme uses the [date and time at which the script was generated](#) to create a script sequence that still has ordering but does not need sequential integers, for example 20151114100523_CreateDatabase.sql, where the ordering is defined by a concatenation of year, month, day, hour, minute, and second in 24-hour format.

A change script-based approach generally rely on a metadata table held in the database itself that captures the database version and keeps tracks of which scripts have been run for that version. At the least, it stores the date of the migration, and the script version/name.



When running a migration, the framework or tool checks to see which script was last run by accessing the metadata version table, and then runs only those scripts that are numbered sequentially after that version.

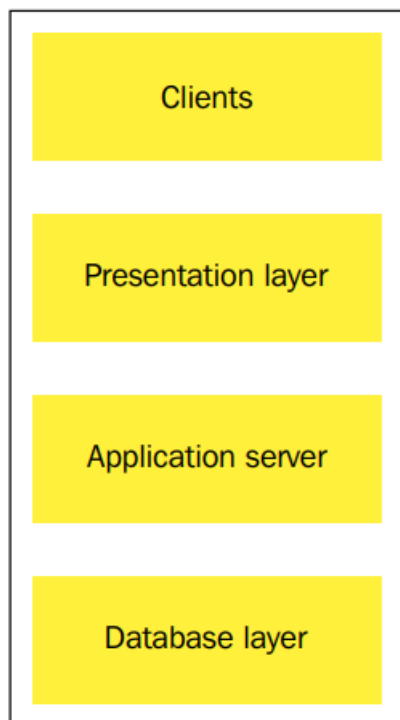
What is three-tier architecture?

Three-tier architecture is a well-established software application architecture that organizes applications into three computing tiers: the presentation tier, or user interface; the application tier, where data is processed; and the data tier, where the data associated with the application is stored and managed.

The chief benefit of three-tier architecture is that because each tier runs on its own infrastructure, each tier can be developed simultaneously by a separate development team, and can be updated or scaled as needed without impacting the other tiers.

Today, most three-tier applications are targets for [modernization](#), using [cloud-native](#) technologies such as [containers](#) and [microservices](#), and for [migration](#) to the cloud.

The three tiers in detail:



Presentation tier

The presentation tier is the user interface and communication layer of the application, where the end user interacts with the application. Its main purpose is to display information to and collect information from the user. This top-level tier can run on a web browser, as desktop application, or a graphical user interface (GUI), for example. Web presentation tiers are usually developed using HTML, CSS and JavaScript Or React. Desktop applications can be written in a variety of languages depending on the platform.

Application tier

The application tier, also known as the logic tier or middle tier, is the heart of the application. In this tier, information collected in the presentation tier is processed - sometimes against other information in the data tier - using business logic, a specific set of business rules. The application tier can also add, delete or modify data in the data tier.

The application tier is typically developed using Python, Java, Perl, PHP or Ruby, and communicates with the data tier using [API](#) calls.

Data tier

The data tier, sometimes called database tier, data access tier or back-end, is where the information processed by the application is stored and managed. This can be a [relational database management system](#) such as [PostgreSQL](#), MySQL, MariaDB, Oracle, DB2, Informix or Microsoft SQL Server, or in a [NoSQL](#) Database server such as Cassandra, [CouchDB](#) or [MongoDB](#).

In a three-tier application, all communication goes through the application tier. The presentation tier and the data tier cannot communicate directly with one another.

Benefits of three-tier architecture

The chief benefit of three-tier architecture is its logical and physical separation of functionality. Each tier can run on a separate operating system and server platform - e.g., web server, application server, database server. And each tier runs on at least one dedicated server hardware or virtual server, so the services of each tier can be customized and optimized without impact the other tiers.

Other benefits (compared to single- or two-tier architecture) include:

- **Faster development:** Because each tier can be developed simultaneously by different teams, an organization can bring the application to market faster, and programmers can use the latest and best languages and tools for each tier.
- **Improved scalability:** Any tier can be scaled independently of the others as needed.
- **Improved reliability:** An outage in one tier is less likely to impact the availability or performance of the other tiers.
- **Improved security:** Because the presentation tier and data tier can't communicate directly, a well-designed application tier can function as a sort of internal firewall, preventing SQL injections and other malicious exploits.

Version Control is the management of changes to documents, computer programs, large websites and other collection of information.

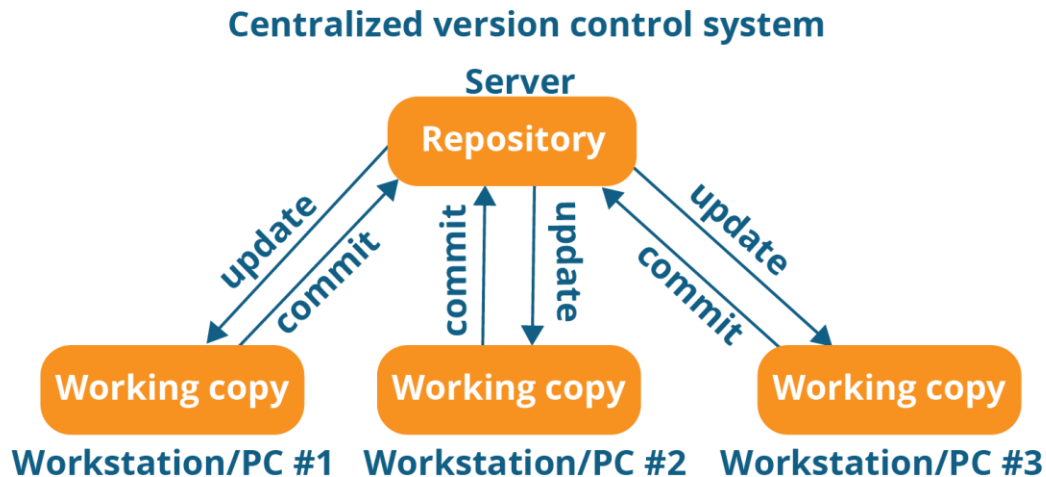
There are two types of VCS:

- Centralized Version Control System (CVCS)
- Distributed Version Control System (DVCS)

1. Centralized VCS

Centralized version control system (CVCS) uses a central server to store all files and enables team collaboration. It works on a single repository to which users can directly access a central server.

Please refer to the diagram below to get a better idea of CVCS:



The repository in the above diagram indicates a central server that could be local or remote which is directly connected to each of the programmer's workstation.

Every programmer can extract or **update** their workstations with the data present in the repository or can make changes to the data or **commit** in the repository. Every operation is performed directly on the repository.

Even though it seems pretty convenient to maintain a single repository, it has some major drawbacks. Some of them are:

- It is not locally available; meaning you always need to be connected to a network to perform any action.
- Since everything is centralized, in any case of the central server getting crashed or corrupted will result in losing the entire data of the project.

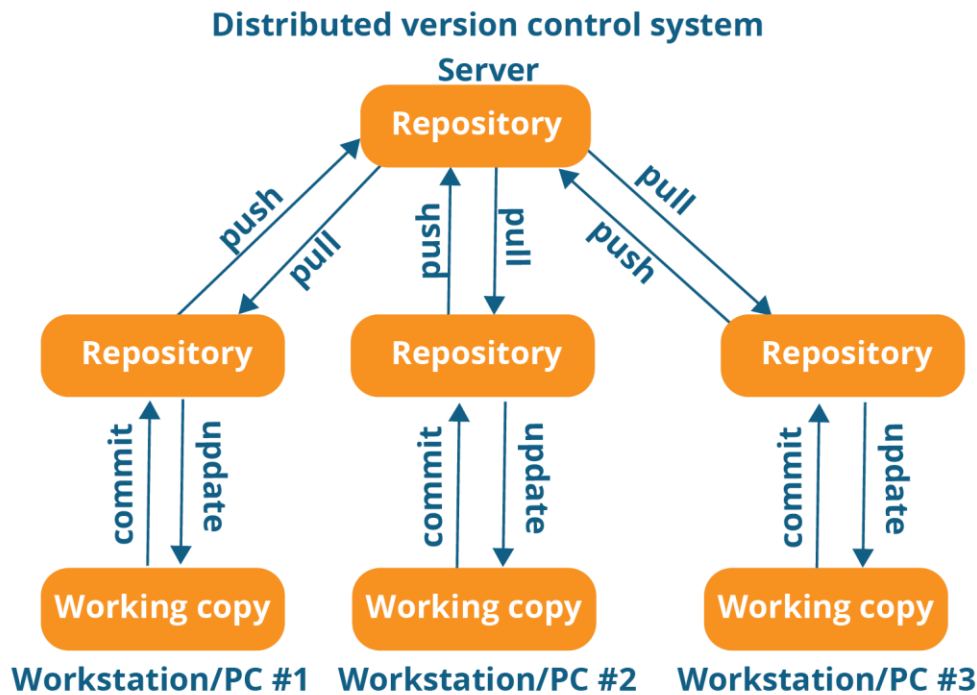
This is when Distributed VCS comes to the rescue.

2. Distributed VCS

These systems do not necessarily rely on a central server to store all the versions of a project file.

In Distributed VCS, every contributor has a local copy or "clone" of the main repository i.e. everyone maintains a local repository of their own which contains all the files and metadata present in the main repository.

You will understand it better by referring to the diagram below:



As you can see in the above diagram, every programmer maintains a local repository on its own, which is actually the copy or clone of the central repository on their hard drive. They can commit and update their local repository without any interference.

They can update their local repositories with new data from the central server by an operation called “**pull**” and affect changes to the main repository by an operation called “**push**” from their local repository.

The act of cloning an entire repository into your workstation to get a local repository gives you the following advantages:

- All operations (except push & pull) are very fast because the tool only needs to access the hard drive, not a remote server. Hence, you do not always need an internet connection.
- Committing new change-sets can be done locally without manipulating the data on the main repository. Once you have a group of change-sets ready, you can push them all at once.
- Since every contributor has a full copy of the project repository, they can share changes with one another if they want to get some feedback before affecting changes in the main repository.
- If the central server gets crashed at any point of time, the lost data can be easily recovered from any one of the contributor’s local repositories.

What Is Git?

Git is a free, open-source distributed version control system(DVCS) tool designed to handle everything from small to very large projects with speed and efficiency. It was created by Linus Torvalds in 2005 to develop Linux Kernel. Git has the functionality, performance, security and flexibility that most teams and individual developers need. It also serves as an important distributed version-control [*DevOps tool*](#). :-)

Git is a Distributed Version Control tool that supports distributed non-linear workflows by providing data assurance for developing quality software.

What is the purpose of Git?

Git is primarily used to manage your project, comprising a set of code/text files that may change.

Git provides with all the Distributed VCS facilities to the user. Git repositories are very easy to find and access. You will know how flexible and compatible Git is with your system when you go through the features mentioned below:

What is Git – Features of Git



1.Free and open source:

Git is released under GPL's (General Public License) open-source license. You don't need to purchase Git. It is absolutely free. And since it is open source, you can modify the source code as per your requirement.



2.Speed:

Since you do not have to connect to any network for performing all operations, it completes all the tasks really fast. Performance tests done by Mozilla showed it was an order of magnitude faster than other version control systems. Fetching version history from a locally stored repository can be one hundred times faster than fetching it from the remote server. The core part of Git is written in C, which avoids runtime overheads associated with other high-level languages.



3.Scalable:

Git is very scalable. So, if in future, the number of collaborators increase Git can easily handle this change. Though Git represents an entire repository, the data stored on the client's side is very small as Git compresses all the huge data through a lossless compression technique.



4.Reliable:

Since every contributor has its own local repository, on the events of a system crash, the lost data can be recovered from any of the local repositories. You will always have a backup of all your files.



5.Secure:

Git uses the **SHA1** (Secure Hash Function) to name and identify objects within its repository. Every file and commit are check-summed and retrieved by its checksum at the time of checkout. The Git history is stored in such a way that the ID of a particular version (a *commit* in Git terms) depends upon the complete development history leading up to that commit. Once it is published, it is not possible to change the old versions without it being noticed.



6.Economical:

In case of CVCS, the central server needs to be powerful enough to serve requests of the entire team. For smaller teams, it is not an issue, but as the team size grows, the hardware limitations of the server can be a performance bottleneck.

In case of DVCS, developers don't interact with the server unless they need to push or pull changes. All the heavy lifting happens on the client side, so the server hardware can be very simple indeed.



7.Supports

non-linear

development:

Git supports rapid branching and merging, and includes specific tools for visualizing and navigating a non-linear development history. A core assumption in Git is that a change will be

merged more often than it is written, as it is passed around various reviewers. Branches in Git are very lightweight.



8. Easy Branching:

Branch management with Git is very simple. It takes only few seconds to create, delete, and merge branches. Feature branches provide an isolated environment for every change to your codebase. When a developer wants to start working on something, no matter how big or small, they create a new branch. This ensures that the master branch always contains



production-quality code.

9. Distributed

development:

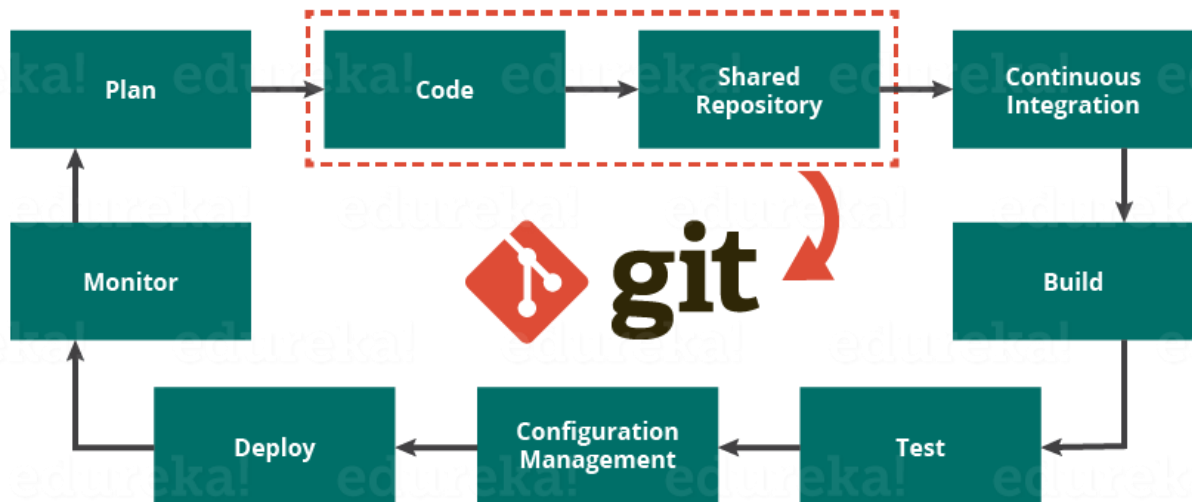
Git gives each developer a local copy of the entire development history, and changes are copied from one such repository to another. These changes are imported as additional development branches, and can be merged in the same way as a locally developed branch.



10. Compatibility with existing systems or protocol

Repositories can be published via http, ftp or a Git protocol over either a plain socket, or ssh. Git also has a Concurrent Version Systems (CVS) server emulation, which enables the use of existing CVS clients and IDE plugins to access Git repositories.

The diagram below depicts the Devops life cycle and displays how Git fits in Devops.



The diagram above shows the entire life cycle of DevOps starting from planning the project to its deployment and monitoring. Git plays a vital role when it comes to managing the code that the collaborators contribute to the shared repository. This code is then extracted for performing continuous integration to create a build and test it on the test server and eventually deploy it on the production.

Tools like Git enable communication between the development and the operations team. Commit messages in Git play a very important role in communicating among the team. The bits and pieces that we all deploy lies in the Version Control system like Git. To succeed in DevOps, you need to have all of the communication in Version Control. Hence, Git plays a vital role in succeeding at DevOps.

Who uses Git? – Popular Companies Using Git

Git has earned way more popularity compared to other version control tools available in the market like Apache Subversion(SVN), Concurrent Version Systems(CVS), Mercurial etc.

Some companies that use Git for version control are: Facebook, Yahoo, Zynga, Quora, Twitter, eBay, Salesforce, Microsoft and many more.

Docker Compose

- Compose is a tool for defining and running complex applications with Docker.
- Docker Compose is a tool for defining and running multi-container Docker applications. It's just that.
- With Docker compose, we will run more than one containers together and dependent another one. For example, you can define a container for database, web server and cache with depending on DB container. In this example, we will use Redis.

Docker compose can be used in many different requirements and/or in many different ways.

1. Development environments
2. Automated testing environments
3. Single host deployments

What is the difference between Git vs GitHub?

Git is just a version control system that manages and tracks changes to your source code whereas GitHub is a cloud-based hosting platform that manages all your Git repositories.

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

GitLab: The end-to-end DevOps Platform

[GitLab's One DevOps Platform](#) gives IT teams a single application that covers the entire software lifecycle, giving everyone an overview of projects as they progress from planning to deployment, monitoring, and documentation. GitLab simplifies the entire toolchain, allowing your teams to work together as a unified, collaborative system while maintaining a high level of security.

Plan

Identifying the work to be done is the first step in the DevOps toolchain. This allows tasks to be prioritized and tracked.

Build

Enabling developers to easily create feature branches, review code, merge branches, and fix bugs allows for a smooth development cycle.

Continuous integration and deployment

Running automated tests each time code is checked in or merged ensures that bugs are detected early and fixed before they get to production.

Monitor

Monitoring your application and production server performance, as well as managing incidents, is critical to the smooth operation of your software.

Operate

Ensuring the released system can scale automatically as needed is one of the ways to guarantee smooth system operations.

Continuous feedback

Distilling and sharing information empowers organizations to develop accurate insights into how well the software is received and used.

What is the difference between GitLab vs GitHub?

Parameters	GitLab	GitHub
Developed by	GitLab was developed by Dmitriy Zaporozhets and Valery Sizov.	GitHub was developed by Chris Wanstrath, Tom Preston-Werner, P. J. Hyett, and Scott Chacon.
Open-sourced	GitLab is open-source for community edition.	GitHub is not open source.
Public Repository	It allows users to make public repository.	It allows users to have unlimited free repository.
Private Repository	GitLab also provides free private repository.	GitHub allows users to have free private repository but with a maximum of three collaborators.
Navigation	GitLab provides the feature of navigation into the repository.	GitHub allows users to navigate usability.
Project Analysis	GitLab provides user to see project development charts.	GitHub doesn't have this feature yet but they can check the commit history.
Advantages	<ul style="list-style-type: none"> GitLab is freely available and open is source for community edition It is a cloud-native application and is highly secure. 	<ul style="list-style-type: none"> It helps us create an organized document for the project. It is used for sharing the work in front of the public.
Disadvantages	<ul style="list-style-type: none"> GitLab is available with many bugs and it makes the user experience sloppy. It is difficult to manage code reviews for first-timers. 	<ul style="list-style-type: none"> There is a limited private repository. It supports only Git version control.
Company	It is owned by GitLab Inc.	It is owned by Microsoft Corporation.
Security	More secure than Github.	It is less secure as security Dashboard, License Compliance is missing in GitHub.
Attachments	Gitlab supports adding other types of attachments.	GitHub does not allow adding other types of attachments.

Chef

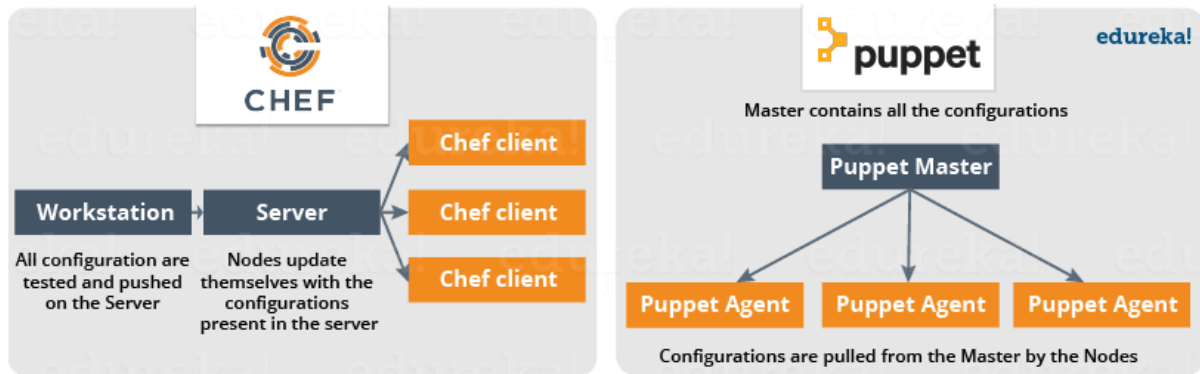
Chef is a tool used for Configuration Management and is closely competing with [Puppet](#)..

Chef is an automation tool that provides a way to define infrastructure as code. Infrastructure as code (IAC) simply means that managing infrastructure by writing code (Automating infrastructure) rather than using manual processes. It can also be termed as programmable infrastructure.

Chef uses a pure-Ruby, domain-specific language (DSL) for writing system configurations. Below are the types of automation done by Chef, irrespective of the size of infrastructure:

- Infrastructure configuration
- Application deployment
- Configurations are managed across your network

Like [Puppet](#) which has a Master-Slave architecture even Chef has a Client-Server architecture. But Chef has an extra component called Workstation.



In Chef, Nodes are dynamically updated with the configurations in the Server. This is called **Pull Configuration** which means that we don't need to execute even a single command on the Chef server to push the configuration on the nodes, nodes will automatically update themselves with the configurations present in the Server.

What Is Chef – Chef Key Metrics

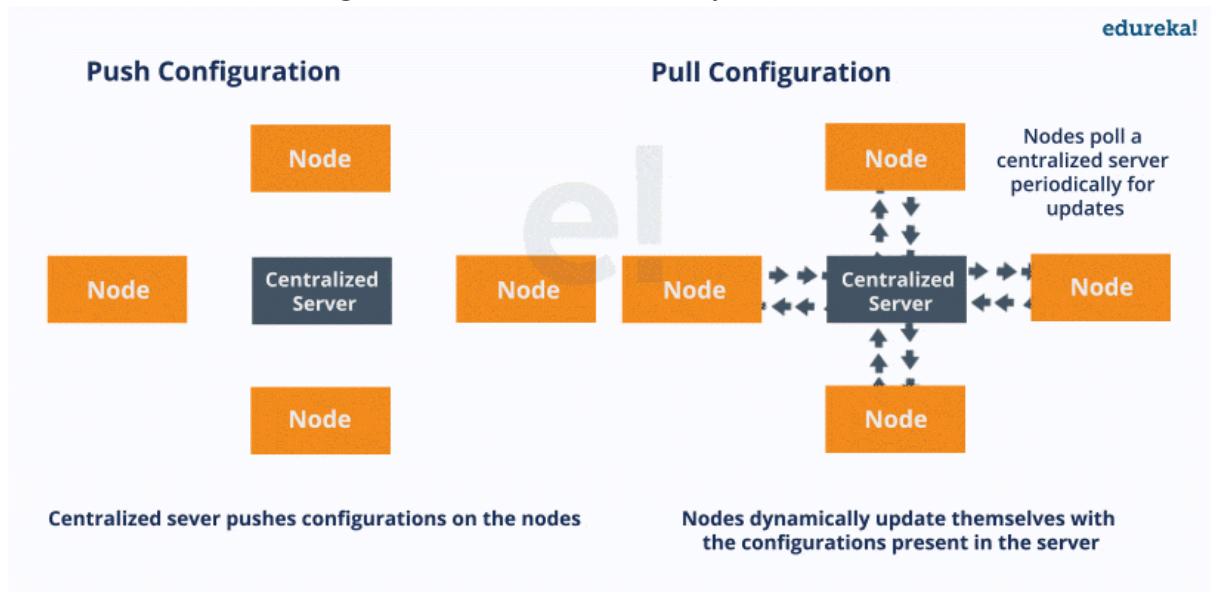
- Chef supports multiple platforms like AIX, RHEL/CentOS, FreeBSD, OS X, Solaris, Microsoft Windows and Ubuntu. Additional client platforms include Arch Linux, Debian and Fedora.
- Chef can be integrated with cloud-based platforms such as Internap, Amazon EC2, Google Cloud Platform, OpenStack, SoftLayer, Microsoft Azure and Rackspace to automatically provision and configure new machines.
- Chef has an active, smart and fast-growing community support.
- Because of Chef's maturity and flexibility, it is being used by giants like Mozilla, Expedia, Facebook, HP Public Cloud, Prezi, Xero, Ancestry.com, Rackspace, Get Satisfaction, IGN, Marshall University, Socrata, University of Minnesota, Wharton School of the University of Pennsylvania, Bonobos, Splunk, Citi, DueDil, Disney, and Cheezburger.

Configuration Management

There are broadly two ways to manage your configurations namely Push and Pull configurations.

- **Pull Configuration:** In this type of Configuration Management, the nodes poll a centralized server periodically for updates. These nodes are dynamically configured so basically they are pulling configurations from the centralized server. Pull configuration is used by tools like Chef, Puppet etc.
- **Push Configuration:** In this type of Configuration Management, the centralized Server pushes the configurations to the nodes. Unlike Pull Configuration, there are certain commands that have to be executed in the centralized server in order to configure the

nodes. Push Configuration is used by tools like Ansible.



DEFINITION

SaltStack

- SaltStack, also known as Salt, is a open-source [configuration management](#) and orchestration tool. It uses a central repository to provision new servers and other [IT infrastructure](#), to make changes to existing ones, and to install software in IT environments, including physical and virtual servers, as well as the cloud.
- SaltStack automates repeated [system administrative](#) and [code deployment](#) tasks, eliminating manual processes in a way that can reduce errors that occur when IT organizations configure systems.
- Salt is used in [DevOps](#) organizations because it **pulls** developer code and configuration information from a central code repository, such as GitHub or Subversion, and **pushes** that content remotely out to servers. Salt users can write their own scripts and programs.
- Salt is a very powerful automation framework. Salt architecture is based on the idea of executing commands remotely.
- Salt is designed to allow users to explicitly target and issue commands to multiple machines directly. Salt is based around the idea of a Master, which controls one or more **Minions**. Communications between a master and minions occur over the **ZeroMQ message bus**.

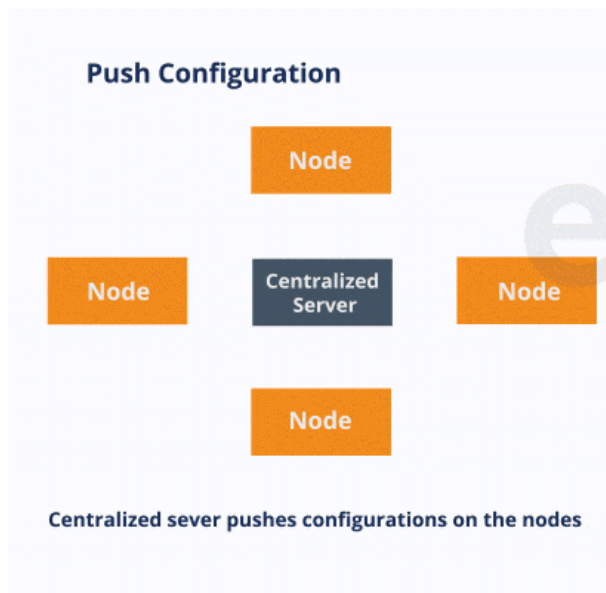
Need for SaltStack & It's features:

- ✓ SaltStack is built for speed and scale. This is why it is used to manage large infrastructures with tens of thousands of servers at LinkedIn, WikiMedia and Google.

Imagine that you have multiple servers and want to do things to those servers. You would need to login to each one and do those things one at a time on each one and then you might want to do complicated things like installing software and then configuring that software based on some specific criteria.

Let us assume you have ten or maybe even 100 servers. Imagine logging in one at a time to each server individually, issuing the same commands on those 100 machines and then editing the configuration files on all 100 machines becomes very tedious task. To overcome those issues, you would love to update all your servers at once, just by typing one single command. SaltStack provides you exactly the solution for all such problems.

- ✓ It is a remote execution engine. Salt is a command-line tool. Written in Python. And it is python based.
- ✓ Salt platform uses the **push model** for executing commands via the SSH protocol.



Benefits of SaltStack

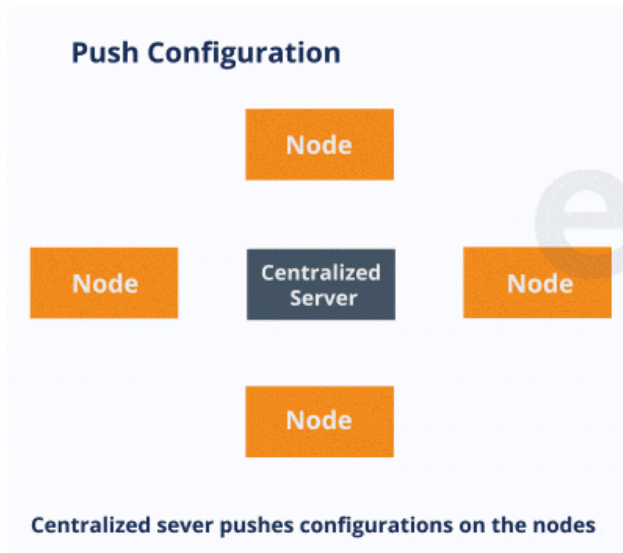
Being simple as well as a feature-rich system, Salt provides many benefits and they can be summarized as below –

- **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.
 - **Fault tolerance**
 - **Flexible**
 - **Scalable Configuration Management**
 - **Parallel Execution model**
 - **Python API**
 - **Easy to Setup.**

What Is Ansible?

- Ansible is a popular IT automation engine that automates tasks that are either repetitive or complex like configuration management, cloud provisioning, software deployment, and intra-service orchestration.
- Ansible is used for the multi-tier deployments and it models all of IT infrastructure into one deployment instead of handling each one separately. There are no agents and no custom security architecture is required to be used in the Ansible architecture.
- The deployment is simple plain English like language that is used in Ansible called YAML which stands for “YAML Ain’t Markup Language.”
- To work with Ansible is very easy; it pushes out small programs called “Ansible Modules” to your nodes to connect. It can deploy and connect using the SSH agent to execute the modules and then removes it when finished. Ansible has over 750 + modules built-in it.
- Ansible can use the inventory and variable information from other sources such as Rackspace, EC2, and Openstack, etc.
- If you need to write your code then also you can use Ansible in languages such as Python, Ruby, and Bash, etc which return JSON. You can write your modules, API, and Plugins.
- Playbooks are the simple and powerful automation language used to orchestrate multiple infrastructures in one goes. This can be done in Ansible.



Ansible for DevOps

Ansible is the most preferred DevOps tool for orchestration, automation, configuration, and managing the IT Infrastructure.

The following are the **benefits of Ansible in DevOps**:

- The feedback loop is accelerated at a faster rate
- The bugs are found sooner and not wait till the end
- Risk due to lack of sufficient knowledge is mitigated
- The deployments are reliable
- The IT infrastructure is coordinated
- The deployments are faster
- Need for automation
- Version control and configuration management
- Orchestration of the IT Infrastructure.
- [Continuous deployment](#)

Top 10 benefits of using Ansible

1. **Agentless** – There are no agents or software deployed on the clients/servers to work with Ansible. The connection can be done through the SSH or using the Python.
2. **English Like Language** – To use the Ansible, configure, and deploy the infrastructure is very simple and it is English like the language used called YAML.
3. **Modular** – The Ansible uses modules to automate, configure, deploy, and orchestrate the IT Infrastructure. There are around 750 + modules built-in Ansible.
4. **Efficient** – There are no servers, daemons, or databases required for Ansible to work.
5. **Features** – Ansible comes with a whole lot of features and can be used to manage the Operating systems, IT Infrastructure, the networks, the servers, and services in very less time.

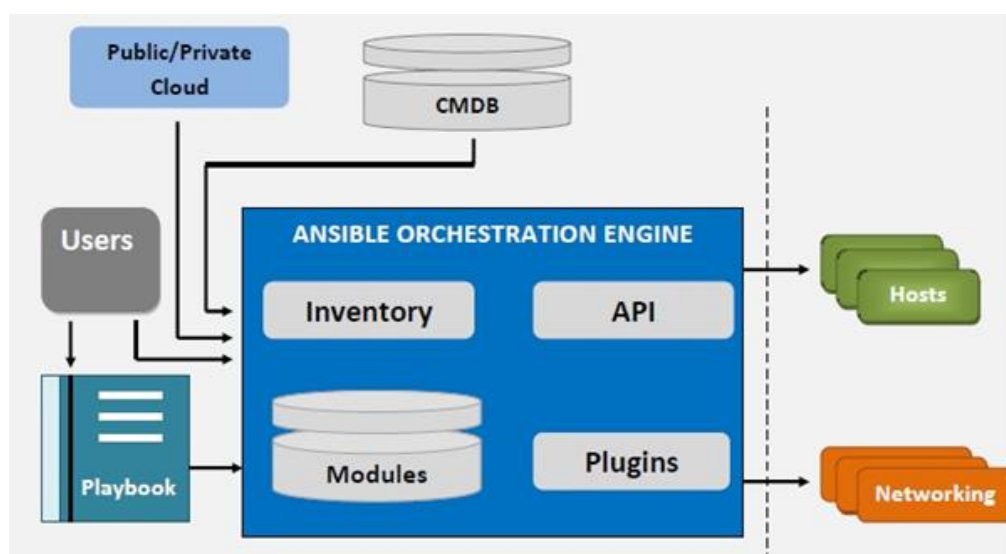
6. **Secure and consistent** – Since the Ansible uses SSH and Python it is very secure and the operations are flawless.
7. **Reliable** – The Ansible Playbook can be used to write programs or the modules and can be used to manage the IT without any downside.
8. **Performance**- The Ansible's performance is excellent and has very little latency.
9. **Low Overhead** – As it is agentless and does not require any servers, daemons, or databases it can provide a lot of space in the systems and has low overhead in terms of deployment.
10. **Simple** – It is very simple to use and is supported by YAML

What Ansible Can Do?

- **Configuration Management** - The enterprise hardware and software information is recorded and updated in detail, thus maintaining the consistency of the product performance.
- **Application Deployment** - The applications can be managed in Ansible from Development to Production.
- **Orchestration** - To manage as a whole and how the configurations interact.
- **Security and Compliance** - Wide security policy can be deployed across the infrastructure when the policy is defined in Ansible
- **Provisioning** - Helps to automate and manage the process

Ansible Architecture an overview

The Ansible architecture is shown below in the diagram.

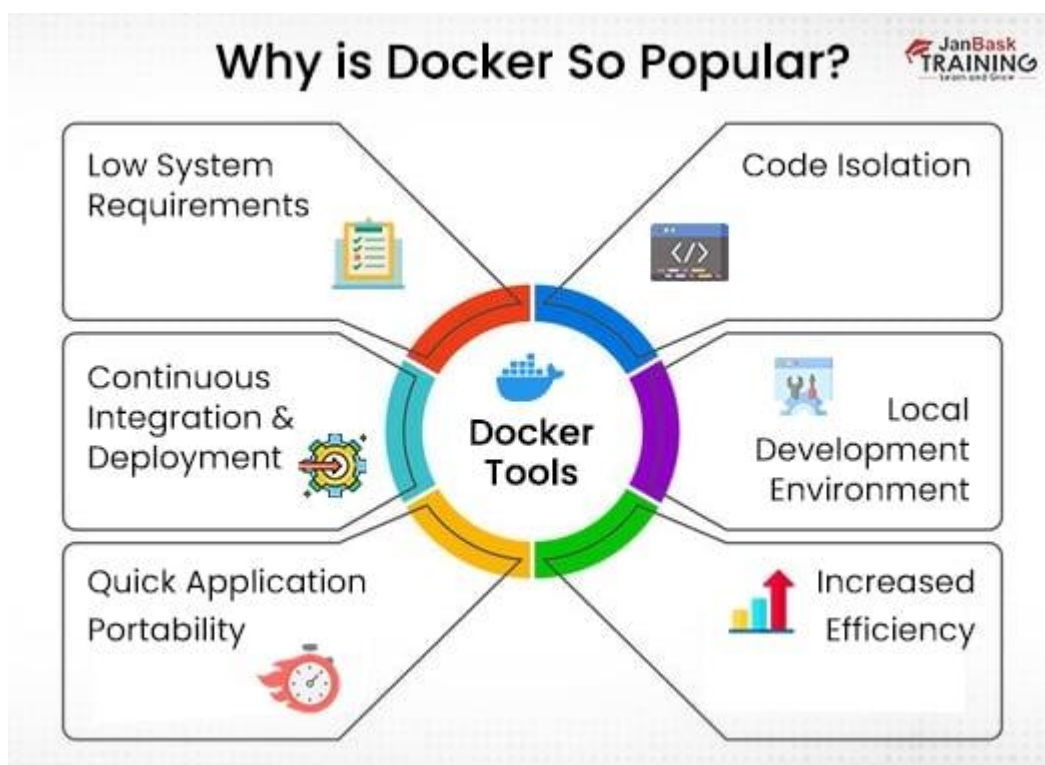


The Ansible Orchestration engine is the center of the Ansible tool.

- It consists of an Inventory table, API, Plugins, and modules written to configure, manage, automate, and orchestrate the process.
- It can get the inputs from the Playbook software, public/private cloud, and configuration management databases to do the networking, manage the hosts or the servers, operating systems, and manage security.

What next?

Ansible is mainly used as a [DevOps tool](#) and can perform a lot of tasks that otherwise are time-consuming, complex, repetitive, and can make a lot of errors or issues.



Differences Between Docker and Virtual Machine

Parameter	Docker	Virtual Machine
Definition	Group of processes that a shared kernel manages	Has an operating system that shares the host hardware via a hypervisor
Image Size	In MBs	In GBs
Boost-up speed	Very Fast	Very Slow
Efficiency	Higher	Lower

Efficiency	Higher	Lower
Scaling	Easy to scale up	Difficult to scale up
Space allocation	Share and reuse data volumes among various Docker containers	Cannot share data volumes with VMs
Portability	Docker is easily portable	May face compatibility issues across different platforms
Availability	Open Source	contains both paid and open-source (free) software

Some docker commands:

1). Command to Install the Process

`//sudo dnf install Docker`

2). Command to Start the Process

`sudo systemctl start Docker`

3). *Command to Enable the Process*

```
//sudo systemctl enable Docker
```

The steps for other Linux versions same steps will be used for this.

4). *How to Create a Container?*

```
$ sudo Docker run -it busybox is /bin/
```

- **Name:** We use Docker to create the containers and users can give a new and unique name to these containers. Docker can also give a default name to the Docker.
- **It:** It stands for interactive. This terminal gets connected to virtual **TTY** and so the running processes get interacted to the output terminal.
- **Busybox:** The base image is used to create the container. It is like a zip file that contains the necessary files to deploy and develop the application.
- **Echo:** It is a command that usually executes the commands that are contained in the busybox.

5). *Command to See the List of Cached Images*

In Docker, when images are used for the first time, they are downloaded and cached to speed up the things. To check the local images, we can use the following command:

```
// sudo Docker images
```

6). *Command to See Background Running Containers*

The status of any of the background running container can be checked by the following command:

```
//sudo Docker ps
```

7). *Command to Kill Running Containers*

Following command can be used to stop a container:

```
Sudo Docker stop [name of your container]
```

```
#example
```

```
Sudo Docker stop snooze
```

A running container is stopped through this command and the container is kept in cache even after deletion. The same command is executed again by the following command:

8). *The Command to Check Container Existence*

The existence of any container can be checked by the following command:

```
Docker ps
```

All running containers can be enlisted by following the above command. While to display, running and non-running containers can be checked by the following command:

```
Docker ps -a
```