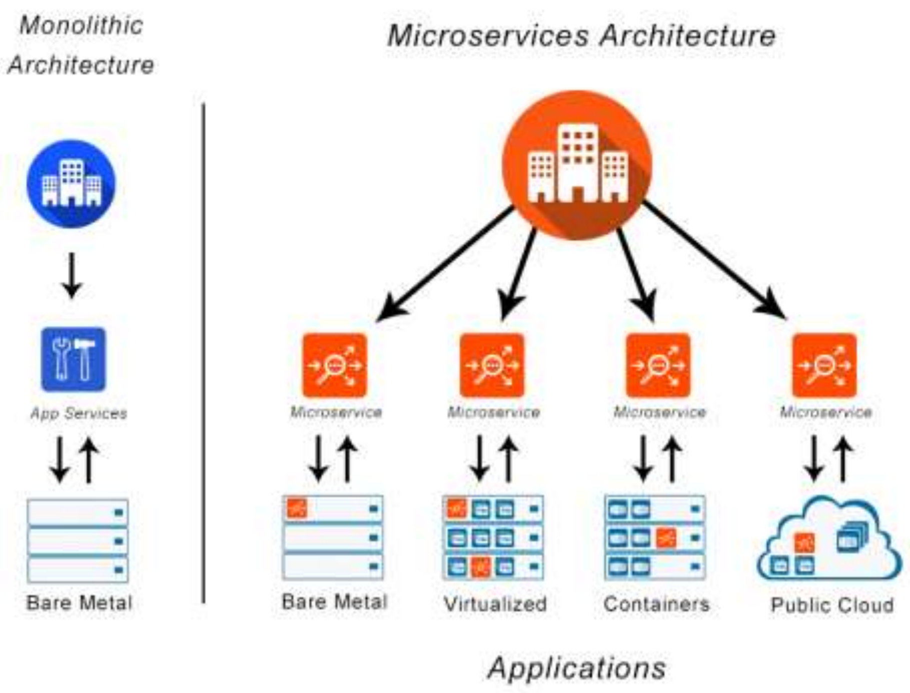
UNIT-1:-

INTRODUCTION TO MICROSERVICES

Definition of Microservices:

Microservices is an architectural design for building a distributed application using containers. They get their name because each function of the application operates as an independent service. This architecture allows for each service to scale or update without disrupting other services in the application. A microservices framework creates a massively scalable and distributed system, which avoids the bottlenecks of a central database and improves business capabilities, such as enabling continuous delivery/deployment applications and modernizing the technology stack.



What Is Microservices Architecture:

Microservices architecture treats each function of an application as an independent service that can be altered, updated or taken down without affecting the rest of the application.

The architecture was designed to solve this problem. All services are created individually and deployed separately from one another. This architectural style allows for scaling services based on specific business needs. Services can also be rapidly changed without affecting other parts of the application. Continuous delivery is one of the many advantages of microservices.

Microservices architecture has the following attributes:

• Application is broken into modular, loosely coupled components
• Application can be distributed across clouds and data centers
• Adding new features only requires those individual microservices to be updated
• Network services must be software-defined and run as a fabric for each microservice to be connect

When to Use Microservices?

Ultimately, any size company can benefit from the use of a microservices architecture if they have applications that need frequent updates, experience dynamic traffic patterns, or require near real-time communication.

Who Uses Microservices?

Social media companies like Facebook and Twitter, retailers like Amazon, media provider like Netflix, ride-sharing services like Uber and Lyft, and many of the world's largest financial services companies all use microservices. The trend has seen enterprises moving from a monolithic architecture to microservices applications, setting new standards for container technology and proving the benefits of using this architectural design.

How Are Microservices Deployed?

Deployment of microservices requires the following:

• Ability to scale simultaneously among many applications, even when each service has different amounts of traffic
• Quickly building microservices which are independently deployable from others
• Failure in one microservice must not affect any of the other services

Docker is a standard way to deploy microservices using the following steps:

• Package the microservice as a container image
• Deploy each service instance as a container
• Scaling is done based on changing the number of container instances

Using Kubernetes with an orchestration system like Docker in deployment allows for management of a cluster of containers as a single system. It also lets enterprises run containers across multiple hosts while providing service discovery and replication control. Large scale deployments often rely on Kubernetes.

Examples of Microservices:

1. Amazon: At the turn of the millennium, Amazon's retail website was a monolith with tightly knit connections between and within its multi-tiered services. This meant that developers had to work carefully to make sure nothing broke every time an upgrade or upscaling activity had to be undertaken.

2. **Uber:** Like Amazon and Netflix, Uber, too, decided to shift away from its monolithic structure due to growth hurdles. The challenges faced by the ride-sharing platform included inefficiency in the development and launch of new features, the inability to fix bugs quickly, and problems with integrating its fast-growing global operations. It reached a point where the complex application architecture needed extensively experienced developers to make minor changes and updates to the system.

Pros:-

- Give developers the freedom to independently develop and deploy service components.
- It can be developed by a fairly small team.
- Codes can be written in different languages for different services.
- Simple to integrate and automatic deployment.
- Developers can use the latest technologies.
- The code is organized for business capabilities.
- No long-term commitment is required for the tech stack.
- Easy to scale the apps and integrate them with third-party services.

Cons:-

- Distributed deployment can create problems for testing and make the job a bit tedious.
- An increased number of services can create an information barrier.
- Another con of a distributed system is it can result in duplication of efforts.
- With the increase in the number of services, managing the whole product becomes complicated.

Characteristics of Microservices:

1.Multiple Components*:*

Microservices software can be broken down into multiple component services. This is because each of these service components can be simultaneously tweaked, deployed, and redeployed independently without compromising the integrity of an app.

Thanks to this, you only need to change one or more distinct service components instead of redeploying the whole application. However, it does have its downsides which include expensive remote calls, coarser-grained remote APIs, and increased complexity between components.

2. Built for Business*:*

This architecture type is usually prioritized around business capabilities. In a monolithic approach, teams have to focus on UIs, technology layers, databases, or server-side logic. Whereas microservices architecture uses cross-functional teams. Each team was responsible for making specific products based on single or more services communicated via message bus.

3. Simple Routing*:*

Microservice architecture patterns act somewhat like a UNIX system where they receive requests, process them and generate responses. This is exactly the opposite of many other products such as ESBs. Thus, it’s safe to say that microservices have smart endpoints that process info.

4. Decentralized*:*

The microservice community favors the decentralized governance model because the developers strive to create helpful tools and systems to solve problems. Also, microservices favor decentralized data management, and software built on it usually manages its own unique database.

5. Failure Resistance*:*

Microservices are also designed to handle failure. There are several diverse services communicating with each other, it's possible that a service could fail for some or another reason. Here, clients should allow neighboring services to function. However, monitoring can help prevent failure.

6. Evolutionary*:*

This architecture is evolutionary and is ideal for developing evolutionary software systems. Due to unforeseen requirements, many monolithic applications are revamped to microservices that interact over an older monolithic architecture via API.

Microservices and Containers:

What are microservices

Microservices are individual units of software that combine to provide all of the functions required to run an application. Typically, each microservice handles a discrete type of functionality within an application. For example, one microservice handles logins, another generates the UI, another populates the interface with content specific to each user session and yet another interfaces with the database that stores user data.

Until about 2010, most applications were monolithic designs in which the entire application ran as a single unit and, in most cases, as a single process. Prior to wide adoption of the internet and APIs, a service-oriented architecture (SOA) approach evolved to break applications into somewhat smaller pieces. However, the individual services within an SOA are typically not as small or dynamic as microservices.

There are several reasons microservices have become popular over the past decade:

- **Agility.** Breaking applications into small pieces makes it easier to scale those different parts up and down. For example, if login requests spike, the application can run more instances of the specific microservice that handles logins.
- **Resilience.** If one microservice fails, the rest of the application remains intact, although part of its function might be unavailable. In this respect, microservices architectures are more resilient.
- **Easier updates.** Developers can update and redeploy microservices individually, which is more efficient than reinstalling the entire application each time an update becomes available.
- **Maintenance.** Typically, developers write and manage code separately for each microservice. This simplifies development work because it eliminates the need to sort through an expansive code base for every change.

What are containers:

Containers are semi-isolated environments in which applications, or parts of applications, can run. Unlike VMs which run entirely separate OSes, containers directly share resources with the OS of the server that hosts the containers. This makes containers more efficient than VMs because each containerized environment does not require a complete guest OS.

Moreover, containers are isolated at the process level from other containers, as well as noncontainerized processes that run on the server. This isolation makes containers more secure than multiple applications that run directly on a host server. Each container can have different environment parameters, rather than all containers sharing a common configuration.

Technology to deploy applications inside containers has existed since the introduction of the Unix chroot call in the 1970s. Containers became massively popular in the mid-2010s with the introduction of Docker and Kubernetes, which provided tooling that made it easier for developers to create and manage containerized applications.

Microservices vs. containers: Strengths and differences

The main difference between microservices and containers is that microservices are an architectural paradigm, while containers are a means to implement that paradigm. Containers host the individual microservices that form a microservices application.

However, an enterprise can host and deploy microservices in a variety of other ways:

- **Serverless functions.** These provide isolated environments that run code preconfigured to respond to triggers such as a user login request.
- **VMs.** It's uncommon to host microservices inside VMs. Nevertheless, it's technically feasible for developers to deploy a set of microservices inside individual VMs and then connect them together to form a microservices app. This provides even stricter isolation between microservices than do containers.
- **Directly on the OS.** There is also no technical reason why you can't deploy a set of microservices directly on the same OS and not isolate them inside of a container or VM.
- **Unikernels.** These lightweight, self-booting environments contain everything required to run a specific application or service. They can be used to deploy microservices on a server without a conventional OS.

How microservices and containers work together

Nevertheless, containers are the most popular way to implement a microservices architecture, for several reasons:

- **Fast start times.** VMs take up to a few minutes to start, but containers can typically start in just a few seconds. It's easier to maximize the agility of microservices when they are hosted inside containers.

- **Security.** Containers provide isolation for each containerized application or microservice, which reduces the risk that a security vulnerability can spread. Microservices deployed directly on a host OS are less secure in this respect.
- **Service discovery.** It's simpler for microservices to locate and communicate with each other when they all run in containers that are managed on the same platform. If you deploy microservices in VMs, serverless functions or unikernels, each host may have a different networking configuration. This makes it harder to design a network architecture for reliable service discovery.
- **Orchestration.** Along similar lines, microservices are easier to orchestrate -- schedule, start, stop and restart -- when they run in containers on a shared platform.
- **Tools.** The tools that support microservices deployment with containers have significantly matured over the past several years. Orchestration platforms for containers, such as Kubernetes, are massively popular and well supported in every major public cloud today. By comparison, there are few tools to orchestrate microservices hosted in unikernels or VMs.

When to choose containers vs. microservices

Put simply, for an enterprise that wants to deploy microservices, containers offer the best tradeoff among security, performance and management. In some scenarios, it makes more sense to deploy microservices without containers -- for instance, a workload may require especially strict isolation between microservices, or several microservices require different OS environments. VMs might be better suited for these situations. Also, keep in mind that containers can host monolithic applications, not just microservices.

For the foreseeable future, though, expect containers and microservices to fit together in a way that other technologies within the microservices landscape do not.

Interacting with Other Services:

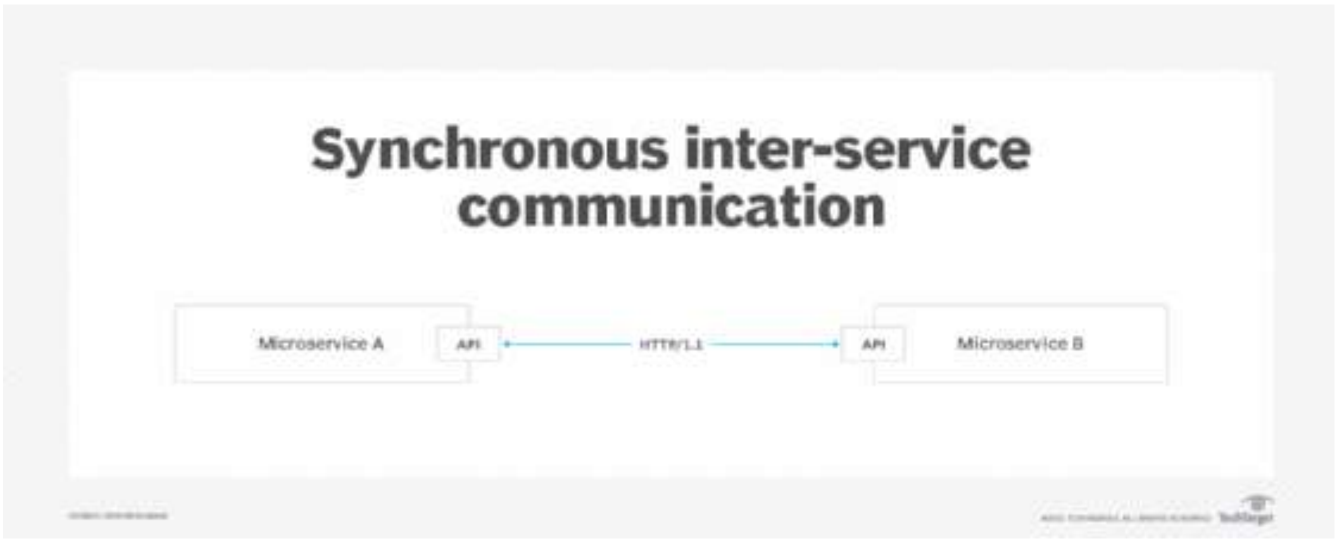
There are two basic messaging patterns that microservices can use to communicate with other microservices.

1. **Synchronous communication**: In this pattern, a service calls an API that another service exposes, using a protocol such as HTTP or gRPC. This option is a synchronous messaging pattern because the caller waits for a response from the receiver.

2. **Asynchronous message passing:** In this pattern, a service sends message without waiting for a response, and one or more services process the message asynchronously.

Synchronous microservices:

A synchronous microservice is one in which data moves to and from an endpoint of a service in a blocking interaction. A typical example of a synchronous data exchange is an HTTP request/response interaction, seen in Figure 1 below. When a request is made to an endpoint under HTTP, the caller is locked in the interaction until a response is received.



The caller might receive the response in a mere millisecond or in a few seconds. Regardless of the application latency, the caller cannot move forward to the next task until the response is received. REST is a good example of a synchronous microservice.

Communication is synchronous when one service sends a request to another service and waits for the response before proceeding further.

The most common implementation of Sync communication is over HTTP using protocols like REST, GraphQL, and gRPC.

When to use Synchronous Communication

- When you cannot proceed without a response from the other service
- When you want real-time responses
- When it takes less time to compute and respond

Advantages of Synchronous Communication

- It is simple and intuitive

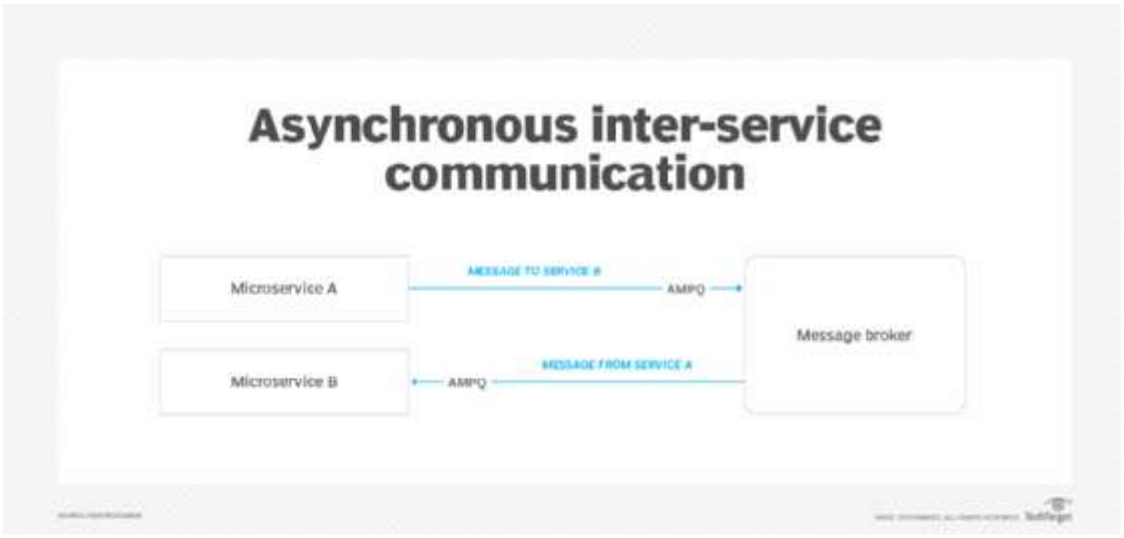
- Communication happens in realtime

Disadvantages of Synchronous Communication

- Caller is blocked until the response is received
- Servers need to be pro-actively provisioned for peaks
- There is a risk of cascading failures
- The participating services are strongly coupled

Asynchronous microservices:

An asynchronous microservice is one in which a request to a service and the subsequent response occur independently from each other. The general practice for implementing an asynchronous microservice is to use a message broker technology, such as Kafka or RabbitMQ, to act as a go-between for services, as seen in Figure 2 below. One service will publish a message to another service using the message broker.



Microservices that communicate in an asynchronous manner can use a protocol such as AMQP to exchange messages via a message broker.

The intended service receives the message in its own time. The sending service is not locked to the broker. It simply fires and forgets.

The communication is asynchronous when the one service sends a request to another service and does NOT wait for the response; instead, it continues with its own execution.

Async communication is most commonly implemented using a message broker like RabbitMQ, SQS, Kafka, Kinesis, etc.

When to use Asynchronous Communication:

- When delay in processing is okay
- When the job at hand is long-running and takes time to execute
- When multiple services need to react to the same event
- When it is okay for the processing to fail and you are allowed to retry

Advantages of Asynchronous Communication:

- Services do not need to wait for the response and can move on
- Services can handle surges and spikes better
- Servers do not need to be proactively provisioned
- No extra network hop due to Load Balancer
- No request drop due to target service being overwhelmed
- Better control over failures and retires is possible
- Services are truly decoupled

Disadvantages of Asynchronous Communication:

- Eventual consistency
- Broker could become a SPoF
- It is harder to track the flow of the message between services

Monitoring and Securing the Services:

Microservices is an application development approach in which large-scale applications are built as modular components, known as services. Each service supports a specific task or business goal and communicates with other services using simple, well-defined interfaces, typically application programming interfaces (APIs).

Monitoring and managing microservices can be particularly challenging, because there can be hundreds of services and thousands of service instances in a microservices application. Monitoring all service components and their interactions can be complex, and requires building observability into individual services, enabling failure detection, centralized log collection, and the ability to collect metrics from logs to identify a range of production issues.

Monitoring is the control system of the microservices. As the microservices are more complex and harder to understand its performance and troubleshoot the problems. Given the vivid changes to software delivery, it is required to monitor the service. There are **five** principles of monitoring microservices, as follows:

- Monitor container and what's inside them.
- Alert on service performance.
- Monitor services that are elastic and multi-location.
- Monitor APIs.
- Monitor the organizational structure.

These principles allow us to address technological changes associated with the microservices and organizational changes related to them.

Microservices Monitoring Challenges:

A major challenge to monitoring microservices environments is to ensure that all the services and components cooperate and deliver high performance and a smooth user experience. Measuring the functioning of these services and how they impact each other and the overall user experience is also challenging.

Traditional monitoring tools focus either on infrastructure, specific software components, or overall operational health. These tools are usually only moderately effective, but they may be sufficient for legacy systems with a monolithic architecture. However, microservices deployments expose the weaknesses of conventional monitoring tools.

Microservice architectures host components in containers or virtual machines distributed across a private, public, or hybrid cloud environment.Measuring the ability of services to communicate with each other and deliver the expected results requires specific monitoring capabilities:

- **End-user experience monitoring**—measures client operations and performance on browser and mobile devices.
- **System interaction monitoring**—measures the system interactions required to service each transaction. These include interactions between the end-user device and the microservices and other components involved in the user's request.
- **End-to-end monitoring**—helps isolate issues across the microservices environment.

A final challenge is managing the complexity of shared, dynamic services involving continuous, accurate documentation and awareness. It is important to train new employees to understand how each component interacts with the others.

Metrics to Monitor in Microservices:

Platform Metrics:

Monitoring platform metrics is critical to keeping microservices infrastructure running smoothly. This is low-level data that can indicate problems in the underlying compute, storage, or networking equipment. Careful monitoring of these metrics can highlight performance degradation and prevent system-wide failures.

Platform metrics include:

- Number of requests per second/minute
- Failed requests per second
- Average response time per service endpoint
- Distribution of time required for each request
- Average execution time for the fastest 10% and slowest 10% queries
- Success/failure rate by service

Resource Metrics:

The infrastructure provider typically provides resource metrics that are useful for monitoring infrastructure health. In the cloud these metrics are provided by a system like AWS CloudWatch; in an on-premise environment they could be Kubernetes metrics generated by a system like Prometheus.

Examples of resource metrics include:

- **CPU and memory utilization of nodes and containers**—monitoring the health and performance of microservices is not as easy as tracking CPU and memory usage on a single, known server. It requires tracking, multiple, ephemeral resources..
- **Host count**—the number of hosts or pods running the system (enables the identification of availability issues resulting from crashed pods).
- **Live threads**—the number of threads spawned by the service (enables the detection of multi-threading issues).
- **Heap usage**—statistics related to heap memory usage (for debugging memory leaks).

Golden Signals:

The concept of "golden signals" refers to metrics that are highly useful for monitoring the health of a microservice, or the entire microservices application, identifying and resolving problems. Examples of golden signals include:

- **Availability**—the system's state as measured from the client's perspective, such as the ratio of errors to total requests.
- **Health**—the system's state as measured using regular pings.
- **Request rate**—the rate of requests coming into the system.
- **Saturation**—the extent to which the system is free or loaded idle time or system load (e.g., available memory or queue depth).
- **Usage**—the system's usage level (CPU load, memory usage, etc.), expressed as a percentage.
- **Error rate**—the rate of errors the system is experiencing.
- **Latency**—the system's response time, usually measured in the 99th or 95th percentile.

5 Tips for Effective Microservice Monitoring:

1. Monitor Containers and What's Running Inside Them:

Containers have become a popular component of microservices. Their portability, speed, and isolation make them an important building block throughout the development lifecycle.

The traditional monitoring process—using a VM or an agent that runs in the host userspace—does not work well with containers because they are small, independent processes with minimal dependencies. Running multiple monitoring agents in a medium-sized deployment can also be expensive at large scale.

To overcome this, developers can either directly instrument their code or use common kernel-level instrumentation methods to monitor all container and application activity on the host.

2. Service Performance Monitoring:

To automate software deployment for containerized applications, DevOps teams use orchestration systems such as Kubernetes, which take a logical blueprint of an application and deploy it as a set of containers. Developers use Kubernetes to define microservices and understand the state of deployed services.

DevOps teams should configure alerts to focus on attributes closely related to the service experience. These alerts can immediately let operational staff know if anything is affecting the application or end users. Container-native monitoring solutions can use orchestration metadata to dynamically aggregate container and application data at a service level.

3. Monitoring Multi-location and Elastic Services:

Elastic services are not a new concept, but they change much faster in containerized environments than virtualized ones. A changing environment can make monitoring more difficult.

In monolithic applications, monitoring typically required manual tuning of metrics based on individual deployments—for example, configuring collection for a specific metric on specific servers. This approach is not feasible in a large microservices environment. Microservice monitoring should have fully automated metrics collection and be able to scale up and down without human intervention. It also needs to run dynamically in a cloud-native environment across multiple data centers, clouds, and edge locations.

4. Monitor APIs:

APIs are the common language in a microservices environment. In a properly-defined microservice, the API is the only element of the service exposed to other teams or external systems. In fact, API response and conformance can be the de-facto SLA of a microservice even when no formal SLAs are defined. This makes API monitoring extremely important.

The basic form of API monitoring is binary uptime checks. But this is not enough. Here are a few ways to extend API monitoring and make it more useful:

- **Monitoring most commonly used endpoints** at a given point in time. This allows the team to see if there are any noticeable changes to the use of the service due to changes in design of other microservices, or changes in the way users consume the service.
- **Monitoring slow endpoints**—identifying the slowest endpoints across the environment can reveal serious problems, or at least point to areas of your system that need optimization.

- **Distributed tracing of service calls** through the system is another important feature. This type of analysis helps understand the end-to-end user experience, and understand how requests interact with infrastructure and application components.

5. Map Monitoring to Your Organizational Structure:

As organizations adopt microservices, they typically reorganize teams in a microservices-compatible structure. These small, decoupled teams have a high degree of control over the languages they use, how errors are handled, and even operational responsibilities.

Monitoring needs to also reflect this structure. A microservices monitoring solution should allow individual teams to define their own alerts, metrics, and dashboards, while providing a broader view of the system that is shared by all teams.

Securing the Services:

Micro-Service is a very small or even micro-independent process that communicates and return message through mechanisms like Thrift, HTTPS, and REST API. Basically, micro-services architecture is the combination of lots of small processes which combine and form an application. In micro-services architecture, each process is represented by multiple containers. Each individual service is designed for a specific function and all services together build an application.
Now let's discuss the actual point of security in micro-service architecture, nowadays many applications use external services to build their application and with the greater demand, there is a need for quality software development and architecture design. Systems administrators, database administrators, cloud solution providers, and API gateway these are the basic services used by the application. Security of micro-services mainly focuses on designing secure communication between all the services which are implemented by the application.

How To Secure Micro-services :
(1) Password Complexity :
Password complexity is a very important part as a security feature is a concern. The mechanism implemented by the developer must be able to enforce the user to create a strong password during the creation of an account. All the password characters must be checked to avoid the combination of weak passwords containing only strings or numbers.
(2) Authentication Mechanism :
Sometimes authentication is not considered a high priority during the implementation of security features. It's important to lock users' accounts after a few numbers of fail login attempts. On login there must be rate-limiting is implemented to avoid the brute force attack. if the application is using any external service all APIs must be implemented with an authentication token to avoid interfering with the user in API endpoint communication. Use multi-factor authentication in micro-services to avoid username enumeration during login and password reset.
(3) Authentication Between Two Services :
The man-in-the-middle attack is may happen during encounters during the service-to-service

communication. Always use HTTPS instead of HTTP, HTTPS always ensures the data encryption between two services and also provides additional protection against penetration of external entities on the traffic between client-server.
It is difficult to manage SSL certificates on servers in multi-machine scenarios, and it is very complex to issue certificates on every device. There is a secure solution HMAC is available over HTTPS. HMAC consists of a hash-based messaging code to sign the request.

(4) Securing Rest Data :
It is very important to secure the data which not currently in use. If the environment is secure, the network is secure then we think that attackers can not reach stored data, but this is not case there are many examples of data breaches in the protected system only due to weak protection mechanisms on data security. All the endpoints of where data is stored must be non-public. Also, during development take care of the API key. All the API keys must be secret leakage of private API also leads to exposure of sensitive data in public. Don't expose any sensitive data, or endpoints in the source code.
(5) Penetration Testing :
It is always good practice to consider security features in the software development life cycle itself. but in general, this is not always true, considering this problem is always important to do penetration testing on the application after the final release. There are some important attack vectors released by OWASP always try these attacks during the penetrating testing of the application. Some of the important attack vectors are mentioned below.

- SQL Injection.
- Cross-Site Scripting (XSS).
- Sensitive Information Disclosure.
- Broken Authentication and Authorization.
- Broken Access Control.

Best Practices to Secure Microservices:
Here are eight of the best practices for securing microservices:

- **Integrate security right from the beginning**

Essentially, this means that security cannot be an "afterthought" but should be part of the development cycle right from the start of the project. Also referred to as "secure by design," microservices security needs to be incorporated at every stage including the design, build, and deployment phase.
How does one integrate architecture security right from the beginning? For a start, developers can perform continuous stress testing of the architecture at the time of writing the code. This can be done through security testing methods like:

1. Static analysis that detects any code vulnerabilities.
2. Dynamic analysis that tries to "ape" malicious attacks to identify any vulnerabilities.

- **Use the Defense in Depth (DiP) technique**

To avert complex and sophisticated attacks, organizations can no longer rely on parametrized solutions like the firewall. The Microservices architecture requires multiple levels of security incorporated at every data and service layer. This is what the Defense in Depth technique is all about.
Unlike firewalls, the DiP technique effectively applies security practices at multiple layers, thus making it difficult for cybercriminals to penetrate through all the layers. This technique uses a

combination of security-related tools including antivirus, firewalls, security patches, and anti-spam protection.
For microservices, the DiP technique can be applied to the most data-sensitive services before applying security layers to the rest of the services.

- **Focus on Container security**

Any microservices architecture is dependent on the underlying container security, which has its share of online threats. Container security can be compromised by a host of vulnerabilities, including application images, misconfigured registry, and runtime vulnerabilities.
Organizations need to focus on container security through a variety of practices including:

1. Using automated container security tools and technologies
2. Limiting permissions
3. Updating and configuring the Host OS to address any vulnerabilities
4. Limiting access to container resources

- **Implement multi-factor authentication**

A microservices security strategy is not complete without securing the endpoints and frontend applications. This makes user authentication and access control critical for securing a microservice application.
Multi-factor authentication (or MFA) is a proven technique for blocking malicious intent. For signing into microservices application accounts, users need to go through a two-step process that consists of entering their correct user credentials (username and password) followed by a unique verification code (that is sent only to their mobile phones or email address). Additionally, an effective MFA process can also “raise a red flag” for any intrusion.

- **Make use of automatic security updates**

This is probably the easiest and one of the most effective ways of ensuring the security of microservices architectures. Regular updates of third-party tools along with scanning them for any vulnerabilities can keep microservices safe and scalable at a time.
As a practice, automate your update process to avoid missing any major security fix or patch released by third-party developers. At the same time, ensure that the released updates are stable so that they do not end up “breaking” your application (or add more vulnerabilities to them).
Security tools like Dependabot from Github work efficiently at automating updates through pull requests.

- **Use the API gateway**

A microservices architecture typically does not allow service consumers to communicate directly with microservices. Instead, the architecture uses an API gateway as a single entry point for this traffic, directing it to the relevant microservices.

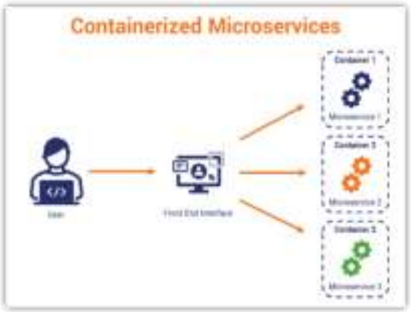
An API gateway usually employs token-based authentication for managing services’ data privileges and determining how they can interact with certain data. Since clients cannot directly access services, they cannot exploit them. Placing the API gateway behind a firewall extends this protection to ensure all the application’s microservices are secure.

Containerized Services:

- Containers virtualize multiple application runtime environments on the same operating system instance, providing isolation and portability.

- Containerized microservices offer reduced overhead, increased portability, faster application development, and easier adoption of a microservices architecture.
- Challenges in containerized microservices include container orchestration, service discovery and load balancing, network complexity, data consistency and synchronization, monitoring and observability, security and access control, and DevOps and continuous delivery.
- Container orchestration tools like Kubernetes and Docker Swarm help address the challenges of managing and automating containerized microservices.
- Strategies such as synchronous and asynchronous communication, service discovery mechanisms, API gateways, message queues, and event-driven architectures can ensure effective communication and coordination between microservices.

- Containers – or containerized environments – let you virtualize multiple application runtime environments on the same operating system instance (or, to be more technically precise, on the same kernel).



How Containerized Microservices Work:

A great way to introduce how containerized microservices work is to describe the other strategies for running microservices (and explain what's wrong with them):

1. **Each microservice runs on its own physical server with its own operating system instance:** This approach keeps the microservices isolated from each other, but it's wasteful. Modern servers have the processing power to handle multiple operating system instances, so a separate physical server for each microservice isn't necessary.
2. **Multiple microservices run on one operating system instance on the same server:** This is risky because it doesn't keep the microservices autonomous from each other. There is an increased chance of failures caused by conflicting application components and library versions. A problem with one microservice can lead to failure cascades that interrupt the operation of others.
3. **Multiple microservices run on different virtual machines (VM) on the same server:** This provides a unique execution environment for each microservice to run autonomously. However, a VM replicates the operating system instance, so you'll pay to license a separate OS for each VM. Also, running an entirely new OS is an unnecessary burden on system resources.
4. **Running microservices in containers with their necessary executables and libraries means that each microservice operates autonomously with reduced interdependency**

on the others. Moreover, multiple containers can run on a single OS instance, which eliminates licensing costs and reduces system resource burdens.

Tools for Containerized Microservices:

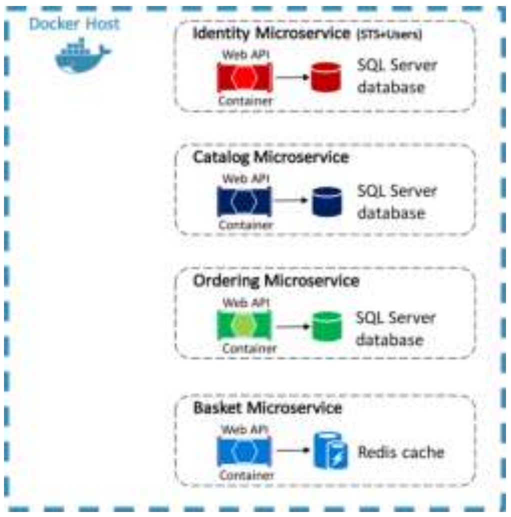
The most well-known tools for building and managing containerized microservices are Docker and Kubernetes. They automate the process of using Linux cgroups and namespaces to build and manage containers.

Docker focuses on creating containers, while Kubernetes focuses on container orchestration. Commonly abbreviated as CO, container orchestration is the automatic process of managing the work of individual containers for applications based on microservices within multiple clusters.

Here's a detailed comparison of these two popular tools for containerized microservices.

Docker:
Docker was released in 2013 as the first large-scale, open-source containerization solution. Built to make Linux container features such as cgroups and namespaces more accessible and easier to use, the Docker platform has since become synonymous with containers and containerized microservices.

Soon after Docker's release, forward-thinking enterprises began using the platform to build containerized runtime environments. This helped support their cloud migration, digital transformation, and microservices efforts. Docker-built containers can immediately reduce cloud-compute overhead expenses for an enterprise by replacing VMs with containers.



Docker's most popular benefits include:

- **Built-in security:** Containerizing an app with Docker automatically reduces the chances of an app outside the container running remote code. This offers built-in application security without needing to code any audits.
- **Scalability:** After running a web-based microservice in a Docker container, the application can run anywhere. Users can access the microservice via their smartphones,

tablets, laptops, or PCs. This significantly reduces your deployment and scalability challenges.

- **Reliable deployment infrastructure:** Docker's hard-coded deployment infrastructure with version control ensures that everyone on your development team stays on the same page.
- **Options:** Docker's premium Enterprise edition offers powerful extensions, a choice of tools and languages, and globally consistent Kubernetes environments. The enterprise edition also includes commercial phone support.

Kubernetes:
Kubernetes is another name that has become synonymous with containerization. It is a container orchestration tool that helps developers run and support containers in production. The open-source container workload platform lets you manage a group of containerized microservices as a "cluster." Within a Kubernetes cluster, you can distribute available CPU and memory resources according to the requirements of each containerized service. You can also move containers to different virtual hosts according to their load.

Working together with Docker, Kubernetes wraps containers in "pods." Then it automates container deployment and loads balancing with the following features:

- **Automatic bin-packing:** Kubernetes packages your microservices or applications into containers and automatically assigns available resources to the containers based on their requirements.
- **Service discovery and load balancing:** Kubernetes automatically configures IPs and ports, and manages traffic for your containers.
- **Storage orchestration:** Kubernetes allows you to mount whatever kind of storage system you want to use, whether it's cloud-based or on-premises.
- **Automatic "healing" to manage container crashes:** Kubernetes automatically spins up and deploys an identical container on a different node to replace containers that crash or fail.
- **Autoscaling:** Kubernetes includes a variety of features that automatically deploy new containers. They can expand the resources of container clusters and manage the resources assigned to containers within clusters. These automatically trigger according to preset thresholds to ensure overall system stability.

Kubernetes is a highly portable solution. They are now supported by the biggest cloud servers. These include Google Cloud Engine (GCE), Amazon Web Services (AWS), and Microsoft Azure Container Service. Private cloud providers – like OpenStack, Microsoft, Amazon, and Google – have also incorporated Kubernetes container services.

Common Containerized Microservices Challenges:

Containerized microservices also come with their own set of challenges. Here are some common challenges that organizations may face when they start adopting containerized microservices:

1. Container Orchestration: Managing a large number of containers and coordinating their deployment, scaling, and networking can be complex. Container orchestration tools like

Kubernetes help streamline this process, however they require a significant investment in learning and infrastructure setup.
2. Service Discovery and Load Balancing: As the number of microservices increases, it becomes crucial to have efficient service discovery mechanisms to locate and communicate with each service. Load balancing is also essential to distribute incoming traffic across multiple instances of a microservice. Implementing robust service discovery and load balancing strategies can be challenging in containerized environments.
3. Network Complexity: Microservices communicate with each other over a network, and managing the network architecture can be complicated. Containerized microservices often span multiple containers and hosts, requiring careful configuration and security considerations to ensure proper communication and data flow between services.
4. Data Consistency and Synchronization: In a distributed microservices architecture, maintaining data consistency and synchronization across services can be challenging. Each microservice may have its own data store or database, and ensuring data integrity and coherence can be complex. Implementing proper data synchronization strategies, such as event-driven architectures or distributed transaction management, becomes crucial in containerized microservices.
5. Monitoring and Observability: With a large number of microservices running in containers, monitoring and observability become essential for maintaining system health and diagnosing issues. Collecting and analyzing logs, metrics, and traces from various containers and services can be challenging without proper monitoring tools and strategies in place.

The Benefits of Containers:

Since containers have come into wide use, sysadmins and IT leaders alike have enjoyed the many benefits of containers. Here's a review of their characteristics and the value they bring to the enterprise:

Reduced overhead

Without the need for an operating system image, containers use fewer system resources than virtual machines. This translates into more efficient server utilization and no additional OS licensing costs.

Increased portability

Containerized microservices and apps are easy to deploy on the widest variety of platforms and operating systems. Run them on a PC, Mac, laptop, or smartphone, and they will operate consistently.

Faster application development

By dividing a monolithic application into a network of containerized, independently-running microservices, application development is faster and easier to organize. Small teams can work more granularly on different parts of an application to develop the highest quality code without the risk of coding conflicts. This results in faster deployment, patching, and scaling.

Easier adoption of a microservices architecture

Containers have become the de facto choice for adopting a microservices architecture because they're less expensive and less process heavy compared to other strategies.

Smaller than virtual machines, containers require less storage space. In many cases, they are 10% or less the size of a VM.

Faster startup than virtual machines

Containers start up in seconds – sometimes milliseconds – because they bypass the need for a VM to lumber through the process of spinning up the operating system each time it launches. Faster container initiation times offer an improved app user experience.

Deploying on Cloud:

Microservice deployments are executed directly via Talend Remote Engine. For each microservice, the Data Service Runner starts a separate Java process as a subprocess of the Remote Engine itself.

Each microservice is deployed with a unique identifier and dynamically gets a unique HTTP port when started. It is displayed on the **Task Details** page in Talend Cloud Management Console after a successful deloyment.

The deployment with a unique identifier and HTTP port allows you to deploy the same Talend Cloud artifact through multiple tasks with different configurations on the same Remote Engine.

By default, the deployment consists of the following parts:

- An artifact (the microservice ZIP based deployment)
- An application.properties file
- A context property file (depending on the Talend Cloud environment and the overwrite parameters set in Talend Cloud Management Console)
- Running a new Java sub-process of the current Talend Remote Engine

Adopting a Cloud Platform Solution refers to implementing a comprehensive infrastructure and service framework that leverages cloud technologies. It enables organizations to harness the benefits of scalability, flexibility, cost optimization, and streamlined operations, empowering them to innovate and thrive in the digital landscape.

In recent years developers are deploying microservices-based applications in the cloud instead of traditional monolithic applications. Because microservices architecture provides better scalability, flexibility, and fault tolerance.

Deploying Microservices in the Cloud:

Service Discovery

Imagine a big city with all similar-looking buildings housing thousands of businesses without any brand boards. Without a map or reliable directory, it would be impossible for you to find the service you are looking for. In the same way, service discovery is important for microservices in the cloud. Service discovery connects different microservices to work together seamlessly.

Service Discovery Best Practices

There are different methods of navigating a business in a big city. Likewise, service discovery has different methods to navigate and connect different microservices.

DNS-based Service Directory: In this method, service names are mapped out in front of their IP addresses. So services can query and find other services, just like an online phone directory.

Client-side Service Directory: In this method, each available service registers itself to the service discovery server. So clients can easily find and communicate with the required service.

Comparison of Cloud Platforms:

Here is a comparison of cloud application development services. Google Cloud platform has its own service discovery service called Cloud DNS. Cloud DNS creates DNS records and makes it easier to deploy microservices in google cloud. On the other hand, Amazon has its own Route 53. It creates DNS records and routes microservices making it easier to deploy Java microservices in AWS.

Nife is another cloud platform that provides a seamless service discovery solution that integrates with both Google Cloud and AWS. Nife's service discovery module automatically registers and updates microservices information in the service registry. It locates and enables communication between microservices.

Load Balancing:

Load balancing is another important part of microservices architecture. With multiple microservices applications working independently with varying loads, it becomes important to carefully manage these microservices for a streamlined workflow. Load balancing acts as a traffic controller and distributes the incoming request to all the available service instances.

Load Balancing Best Practices:

Just as there are different methods for controlling traffic there are different practices for load balancing in a microservices architecture.

Round Robin: In this load-balancing method requests are distributed among services in a rotating fashion. Services are queued and each new request is transferred to services instances following their position in the queue.

Weighted Round Robin: In this method, each service is given a weight, and requests are served proportionally among all the services.

Least Connections: In this load-balancing method requests are directed according to the load on service instances. Requests are directed towards the services handling the least amount of load.

Comparison of Cloud Platforms:

Here is a comparison of two renowned cloud application development services. Google Cloud platform has its own load balancing services which provide HTTP(S) Load Balancing, TCP/UDP Load Balancing, and Internal Load Balancing, and makes it easier to deploy microservices in google cloud. On the other hand, Amazon also has its own Elastic Load Balancing (ELB). It provides different types of load balancing options to handle load efficiently making it easier to deploy Java microservices in AWS.

Nife is another cloud platform that offers full load-balancing options. It can integrate with both google cloud and AWS. Nife leverages effective load-balancing techniques for microservices architecture for an efficient and streamlined workflow.

Scaling

Scaling is another important aspect of microservices deployment, especially in the case of cloud platforms for Middle East region. Microservices break down complex applications into smaller manageable services. The workload on each of these services can increase dramatically in case of increased demand. To manage these loads it is important to have a scalable infrastructure. Here are some primary scaling approaches.

Horizontal Scaling: In this practice, more microservices are added to handle increasing load.

Vertical Scaling: In this practice, the resources of microservices are increased. In this way, microservice can handle increasing demand with

Nife: Simplifying Microservices Deployment in the Cloud | Cloud Platform Solution:

Developers are always seeking efficient and streamlined solutions for deploying microservices. That’s where Nife comes in, a leading platform for cloud application development services. It simplifies the deployment of microservices and provides a wide range of features tailored according to the needs of developers. With Nife, you can enjoy a unified experience whether you want to deploy microservices in google cloud or deploy Java microservices in AWS.

By leveraging Nife's Cloud Platform for Middle East, developers can address the unique needs of that region. Nife's strength is its seamless integration of service discovery, load balancing, and scaling capabilities. Nife provides a service discovery mechanism to enable communication between microservices. Its seamless load balancing automatically distributes traffic across microservices. While automatic scaling ensures proper usage of cloud resources according to demand.

UNIT-2:-

MICROSERVICES ARCHITECTURE

Monolithic architecture:

In a monolithic architecture, the operating system kernel is designed to provide all operating system services, including memory management, process scheduling, device drivers, and file systems, in a single, large binary. This means that all code runs in kernel space, with no separation between kernel and user-level processes.

The main advantage of a monolithic architecture is that it can provide high performance, since system calls can be made directly to the kernel without the overhead of message passing between user-level processes. Additionally, the design is simpler, since all operating system services are provided by a single binary.

a monolithic architecture can provide high performance and simplicity but may come with some trade-offs in terms of security, stability, and flexibility. The choice between a monolithic and microkernel architecture depends on the specific needs and requirements of the operating system being developed

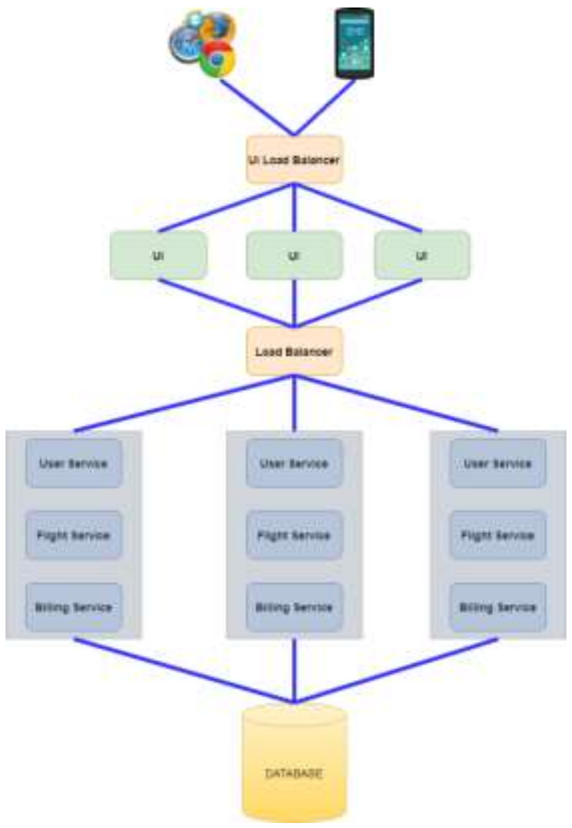
Characteristics of a monolithic architecture:

- **Single Executable:** The entire application is packaged and deployed as a single executable file. All components and modules are bundled together.
- **Tight Coupling:** The components and modules within the application are highly interconnected and dependent on each other. Changes made to one component may require modifications in other parts of the application.
- **Shared Memory:** All components within the application share the same memory space. They can directly access and modify shared data structures.
- **Monolithic Deployment:** The entire application is deployed as a single unit. Updates or changes to the application require redeploying the entire monolith.
- **Centralized Control Flow:** The control flow within the application is typically managed by a central module or a main function. The flow of execution moves sequentially from one component to another.

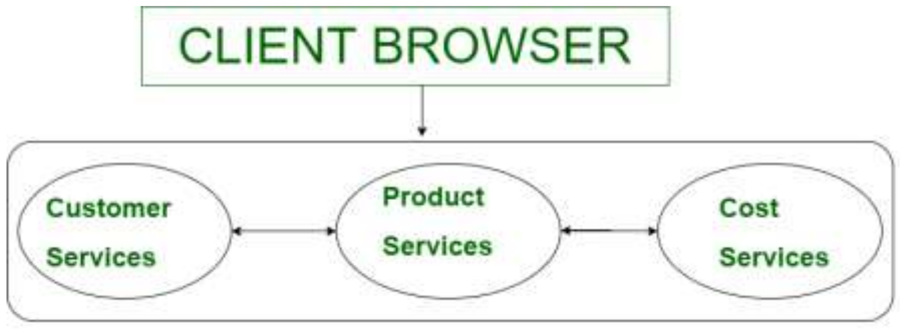
Monolithic architecture is a unified development model for software applications. It has three components:

- Client-side user interface
- Server-side application
- Data interface

All three parts interact with a single database. Software built on this model operates with one base of code. As a result, whenever stakeholders want to make updates or changes, they access the same set of code. This can have ripple effects that impact user-side performance.



Monolithic Architecture is like a big container, wherein all the software components of an app are assembled and tightly coupled, i.e., each component fully depends on each other. **Example:** Let's take an example of an e-commerce site-



As you can see in the example all the services provided by the application (Customer Services, Cost Services, Product Services) are directly connected. So if we want to change in code or something we have to change in all the services as well.

Advantages:

1. High performance: Monolithic kernels can provide high performance since system calls can be made directly to the kernel without the overhead of message passing between user-level processes.

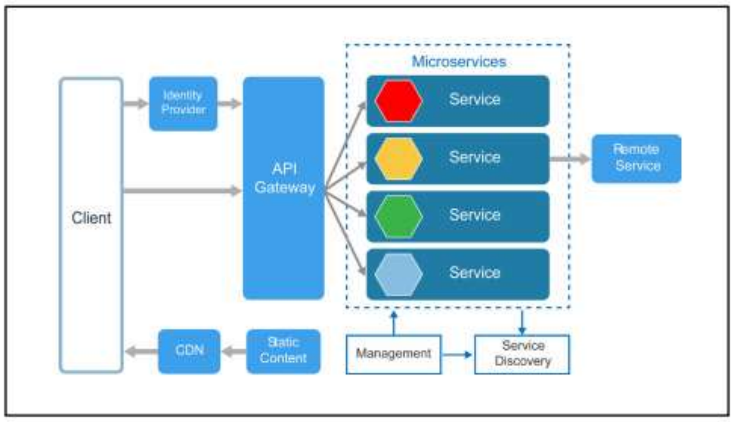
2. Simplicity: The design of a monolithic kernel is simpler since all operating system services are provided by a single binary. This makes it easier to develop, test and maintain.
3. Broad hardware support: Monolithic kernels have broad hardware support, which means that they can run on a wide range of hardware platforms.
4. Low overhead: The monolithic kernel has low overhead, which means that it does not require a lot of system resources, making it ideal for resource-constrained devices.
5. Easy access to hardware resources: Since all code runs in kernel space, it is easy to access hardware resources such as network interfaces, graphics cards, and sound cards.
6. Fast system calls: Monolithic kernels provide fast system calls since there is no overhead of message passing between user-level processes.

Disadvantage:

1. **Large and Complex Applications:** For large and complex application in monolithic, it is difficult for maintenance because they are dependent on each other.
2. **Slow Development:** It is because, for modify an application we have to redeploy whole application instead of updates part. It takes more time or slow development.
3. **Unscalable:** Each copy of the application will access the hole data which make more memory consumption. We cannot scale each component independently.
4. **Unreliable:** If one services goes down, then it affects all the services provided by the application. It is because all services of applications are connected to each other.
5. **Inflexible:** Really difficult to adopt new technology.It is because we have to change hole application technology.

<u>Microservices architectural style:</u>

The Microservices-based application architecture represents a collection of small, autonomous and self-contained services which are built to serve a single business functionality/capability. The following represents a sample microservices-styled application landscape representing a set of microservices. Pay attention to different micro-services accessed through API gateway.



In this post, you will learn about some of the **best practices** related to **microservices-styled architecture (MSA)** which can be adopted when creating the microservices-based application. The following are some of the best practices related to microservices which you may consider following while doing application implementation based on Microservices-styled architecture:

- **Model Services based on Domain-driven Design (DDD)**: Services should be modeled around the business domain. Check out some of the following great reads in relation to **domain-driven design**.
- **Consider separating data storage**: Data should be made private to each of the microservices. Microservice becomes the **owner of its data**. **Any access to data** owned by a specific service should **only happen through APIs**. Failing to do so would allow multiple services to access the database owned by a specific service leading to **coupling between services**. The architecture pattern such as CQRS (Command and Query Responsibility Segregation) comes handy in taking care of data which required to be read by different kinds of users.
- **Build separate teams for different microservices**: Teams should be divided based on microservices with one team working on one microservice. This consists of product manager, and DevOps staff (development, QA, and Ops staff). Recall that microservices shine when they could help organizations in building cloud-native applications which could be released to cloud frequently with very less lead time.
- **Design domain-driven APIs**: APIs should be designed keeping the business domain in mind. Also, implementation details should not be made part of API design.
- **Design cohesive services**: Consider grouping the functions requiring to change together as a single unit rather than separate services. Not doing so would lead to a lot of inter-service communications representing the hard-coupling.
- **Consider separating services for cross-cutting concerns**: One should consider designing separate services for cross-cutting concerns such as authentication and authorization.
- **Automate enough for independent deployment**: Nicely designed micro-services should be able to be deployed independently. And, build and release automation would enhance the deployment process thereby leading to quicker releases and shorter overall lead time. This would help build micro-services truly cloud-native in nature with micro-services wrapped within containers and deployed to any environment including cloud in easy manner. Good **DevOps** practice followed organization-wide would help achieve this objective.
- **Failure isolation**: Microservices-based architecture should consider adopting isolation of failure with independent micro-services. Architecture principles and design patterns such as some of the following would help achieve the same:
  - **Circuit-breaker** design pattern
  - Asynchronous communication
  - Loose coupling
  - Event-driven architecture
  - Stateless design
  - Self-contained services
  - Timeouts

API Gateway. The API gateway is the entry point for clients. Instead of calling services directly, clients call the API gateway, which forwards the call to the appropriate services on the back end.

- It decouples clients from services. Services can be versioned or refactored without needing to update all of the clients.
- Services can use messaging protocols that are not web friendly, such as AMQP.
- The API Gateway can perform other cross-cutting functions such as authentication, logging, SSL termination, and load balancing.
- Out-of-the-box policies, like for throttling, caching, transformation, or validation.

Benefits of Microservices architectural style:

- **Agility.** Because microservices are deployed independently, it's easier to manage bug fixes and feature releases. You can update a service without redeploying the entire application, and roll back an update if something goes wrong. In many traditional applications, if a bug is found in one part of the application, it can block the entire release process. New features might be held up waiting for a bug fix to be integrated, tested, and published.
- **Small, focused teams**. A microservice should be small enough that a single feature team can build, test, and deploy it. Small team sizes promote greater agility. Large teams tend be less productive, because communication is slower, management overhead goes up, and agility diminishes.
- **Small code base**. In a monolithic application, there is a tendency over time for code dependencies to become tangled. Adding a new feature requires touching code in a lot of places. By not sharing code or data stores, a microservices architecture minimizes dependencies, and that makes it easier to add new features.
- **Mix of technologies**. Teams can pick the technology that best fits their service, using a mix of technology stacks as appropriate.
- **Fault isolation**. If an individual microservice becomes unavailable, it won't disrupt the entire application, as long as any upstream microservices are designed to handle faults correctly. For example, you can implement the Circuit Breaker pattern, or you can design your solution so that the microservices communicate with each other using asynchronous messaging patterns.
- **Scalability**. Services can be scaled independently, letting you scale out subsystems that require more resources, without scaling out the entire application. Using an orchestrator such as Kubernetes or Service Fabric, you can pack a higher density of services onto a single host, which allows for more efficient utilization of resources.
- **Data isolation**. It is much easier to perform schema updates, because only a single microservice is affected. In a monolithic application, schema updates can become very challenging, because different parts of the application might all touch the same data, making any alterations to the schema risky.

Drawbacks of Microservices architectural style:

- **Complexity**. A microservices application has more moving parts than the equivalent monolithic application. Each service is simpler, but the entire system as a whole is more complex.

- **Development and testing**. Writing a small service that relies on other dependent services requires a different approach than writing a traditional monolithic or layered application. Existing tools are not always designed to work with service dependencies. Refactoring across service boundaries can be difficult. It is also challenging to test service dependencies, especially when the application is evolving quickly.
- **Lack of governance**. The decentralized approach to building microservices has advantages, but it can also lead to problems. You might end up with so many different languages and frameworks that the application becomes hard to maintain. It might be useful to put some project-wide standards in place, without overly restricting teams' flexibility. This especially applies to cross-cutting functionality such as logging.
- **Network congestion and latency**. The use of many small, granular services can result in more interservice communication. Also, if the chain of service dependencies gets too long (service A calls B, which calls C...), the additional latency can become a problem. You will need to design APIs carefully. Avoid overly chatty APIs, think about serialization formats, and look for places to use asynchronous communication patterns like queue-based load leveling.
- **Data integrity**. With each microservice responsible for its own data persistence. As a result, data consistency can be a challenge. Embrace eventual consistency where possible.
- **Management**. To be successful with microservices requires a mature DevOps culture. Correlated logging across services can be challenging. Typically, logging must correlate multiple service calls for a single user operation.
- **Versioning**. Updates to a service must not break services that depend on it. Multiple services could be updated at any given time, so without careful design, you might have problems with backward or forward compatibility.
- **Skill set**. Microservices are highly distributed systems. Carefully evaluate whether the team has the skills and experience to be successful.

Decomposing monolithic applications into Microservices:

After reading the previous two posts you all must be excited to implement microservice architecture to your application and that’s good you should be ready to implement something new that might boost the application performance, maintenance, scalability and will make your application future proof. But have you ever thought how will you be breaking your application into micro application i.e microservices. Think about it for some minute exercise your brain for some time before reading the below tutorial.

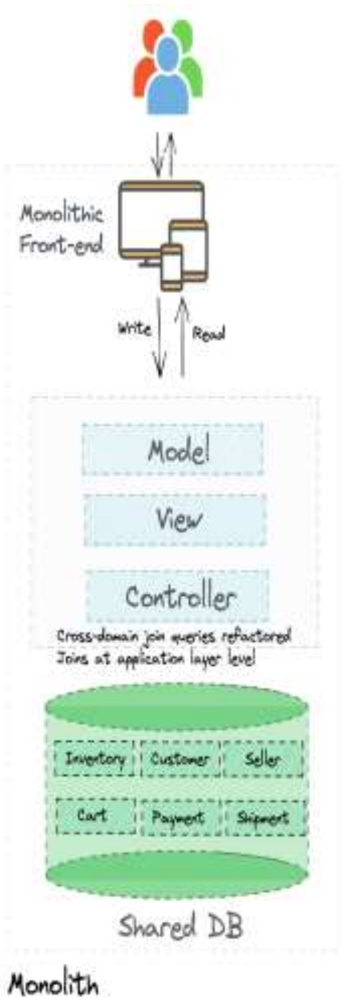
Now i am sure that you all might have come through excellent ideas for breaking up your application in microservices. Please feel free to mention your valuable ideas in comment so that other or even me can be benefited by your ideas.

Now before breaking your application into micro-services you should consider following points.

Decomposing an application into services is a key issue in the deployment and CI/CD architecture of microservices.

There is no single best decomposing strategy, but there are general methods to help you decompose your solution into multiple services.

- Divide by module. This method suggests isolating a component for each module or function and assigning them to separate teams that will be responsible for them.
- Split by domain. This method involves defining the region where your solution will be deployed. Each domain consists of several subdomains and each of them is responsible for a separate part of the business.



Points to be considered before creating microservices.

- Your each services should be easily testable.
- Your application should be Cohesive.
- Your application should be easily maintained by the help of small team(6-8 members).
- Each micro-services should be loosely coupled.

- Each team who are developing microservices should be self capable for end to end process from development to deployment.

Patterns to decompose your monolithic application into microservices

- Decompose on basis of business capability.
- Decompose on basis of subdomain.

Decomposition on basis of business

Whenever we develop microservices architecture we focus that our application is loosely coupled and support continuous delivery and development. Whenever changes come to our application it should impact only one service no multiple services as when the we have to deal with multiple services changes in order to comply with single change then we have to interact and depend on multiple team which will slow the process of development and cost productivity.

So when decomposing the application into microservices we can considers business capabilities. Business capability is something that business does to generate its value. We will take and example of school management system.

In School management system from where the scholl generates it value.

- Admission
- Students
- Stationery

Then we can decompose our application on igh label into 3 services on which will deal with admission of the student, Students management, and Stationery.

Main problem with this structure is before decomposing one should have proper understating of business and its organisation for successful decomposition of the application.

Decomposition on basis of Subdomain

We can follow Domain-Driven Design(DDD) to apply this type of decomposition. In this Decomposition we have to define our domains in which our business will be working and also we have to identify the subdomain for each domain so that our each services will be small and easily manageable. We will take same example as we took above, of Student Management.

Here we can have domain as.

- Admission
- Students
- Stationery

And further Admission can be decomposed into Accounts, Fees Management and etc

And Students can be decomposed into enrollment, Attendance and etc.By this we will be having small modules or services which are easily manageable and maintainable.

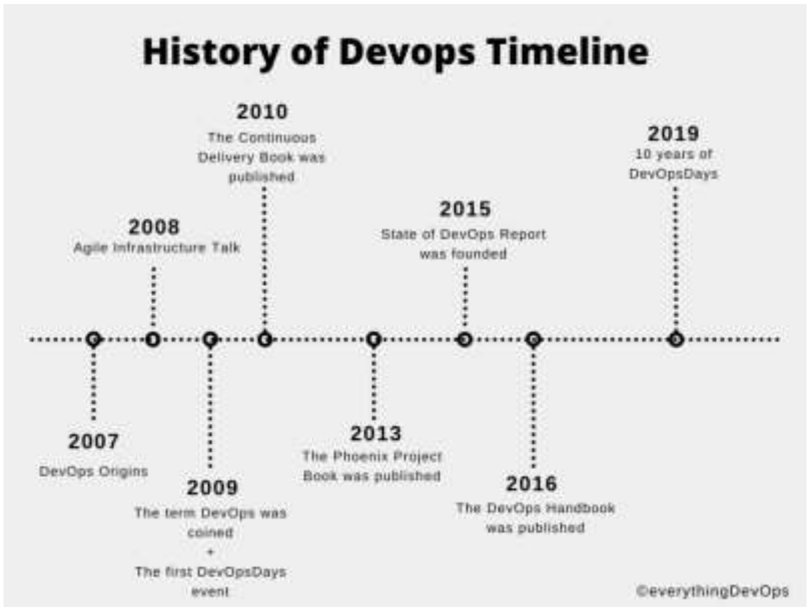
UNIT-3:-

DevOps Tools

History of DevOps:

As Agile development gained popularity, a new term emerged to describe the practices and principles used to streamline the development and deployment process: DevOps.

To share the history of DevOps, below is a detailed history of DevOps timeline, listing all the defining moments and summarizing the contributions of key influential people from 2007 to 2019.


History of Devops Timeline
2007
DevOps Origins
2008
Agile Infrastructure Talk
2009
The term DevOps was coined
+
The first DevOpsDays event
2010
The Continuous Delivery Book was published
2013
The Phoenix Project Book was published
2015
State of DevOps Report was founded
2016
The DevOps Handbook was published
2019
10 years of DevOpsDays
©everythingDevOps

2007:
DevOps started in 2007 when Patrick Debois — an IT consultant, recognized that development (Dev) and operations (Ops) teams were not working well together. While the gaps and conflicts between Dev and Ops have always been unsettling to him, the constant switching and back and forth on a large data center migration project where he was responsible for testing particularly frustrated him.

One day he was deep in the rhythm of Agile development. Next day he was firefighting and living the unpredictability of traditional operations. He knew there had to be a better way.

2008:
The following year, at the 2008 Agile Conference, Andrew Shafer created a birds of a feather meeting (BoF) to discuss "Agile Infrastructure". Andrew didn't think anybody would come, so he himself didn't show up to his own meeting. Patrick Debois showed up, and went looking for Andrew because he wanted to talk about Agile infrastructure being the solution to get operations to be as Agile like the developers were. This was where DevOps got started.

2009:
In 2009, at the Velocity conference, John Allspaw and Paul Hammond talked about “10+ deploys per day - Dev and Ops Cooperation at Flickr,” and the idea started gaining traction. This talk made people notice what was possible by adopting these early DevOps practices. Also, in October 2009, Patrick, held the first DevOpsDays conference in Ghent, Belgium. It was described as “The conference that brings development and operations together.” This is where the term "DevOps" was first used. DevOpsDays is now a local conference held internationally several times a year in different cities.

2010:
In 2010, Jez Humble and David Farley wrote a groundbreaking book called Continuous Delivery that sets out the principles and technical practices that enable rapid, incremental delivery of high-quality, valuable new functionality to users using a technique called Continuous Delivery.

Through automation of the build, deploy, and test processes, along with improved collaboration between developers, testers, and operations, delivery teams can release changes in a matter of hours—sometimes even minutes—no matter the size of a project or the complexity. The book is over 13 years old, but it still has a lot of great concepts that helped changed a lot of people's thinking about how to perform software delivery in a continuous fashion.

2013:
Two years later, in 2013, Gene Kim, Kevin Behr, and George Spafford published The Phoenix Project, based on Eliyahu Goldratt’s book, The Goal. The Goal is about a manufacturing plant about to go under and what they had to do to bring it back to life. It is a story about lean manufacturing principles. The Phoenix Project is about an information technology (IT) shop in a company about to go under and what it took to bring it back to life. This story is about applying lean manufacturing principles to software development and delivery.

2015:
In 2015, Dr. Nicole Forsgren, Gene Kim, and Jez Humble founded a startup called DORA (DevOps Research and Assessment) that produced what are now the largest DevOps studies to date called the State of DevOps Reports. Nicole was the CEO and is an incredible statistician.

Through this research, she found that taking an experimental approach to product development can improve your IT and organizational performance and that high-performing organizations are decisively outperforming their lower-performing peers in terms of throughput. The research shows that undertaking a technology transformation initiative can produce sizeable cost savings in any organization. If you haven't read the most recent State of DevOps report, I strongly urge you to do so.

2016:
The DevOps Handbook was published in 2016. It was written by Gene Kim, Jez Humble, Patrick Debois, and John Willis as a follow-on to The Phoenix Project and serves as a practical guide on implementing the concepts introduced in that book. John Willis, by the way, worked at Docker and Chef back then, and is a DevOpsDays coordinator after being at the original DevOpsDays in Ghent 2009 with Patrick Debois. If you only read one DevOps book, this is the

book to read. They looked at companies that have adopted DevOps and document what did work and what did not work. It's a great read.

2019 — 10 years of DevOpsDays:
Come 2019, 10 years after the first DevOpsDays in Ghent, Belgium, 60+ DevOpsDays events were held in 21 countries.

Patrick Debois led DevOpsDays from its inception in 2009 until 2014, and then Bridget Kromhout became the lead in 2015. She is also the co-host on the very popular podcast, Arrested DevOps. If you don't listen to it, you should. She stepped down in 2020 but stayed on the advisory board of DevOpsDays with Patrick.

The individuals mentioned above are some of the major influential people in the early DevOps movement. They weren't the only ones, but they went out and made a difference. They showed us how DevOps can be impactful.

Why is the DevOps History Important?

Knowing the DevOps History is important because DevOps was a grassroots effort started by people like Patrick Debois and Andrew Shafer, who just wanted to eliminate the roadblocks in software delivery and figure out how can development and operations work better together.

And as Damon Edwards, who co-hosted a podcast with John Willis called the 'DevOps Cafe', said it best in his talk on "The (Short) History of DevOps":

- DevOps is from the practitioners, by practitioners.
- It's not a product, a specification, or job title.
- It is an experience-based movement that is decentralized and open to all.

DevOps and Software Development Life Cycle:

DevOps:

DevOps is a collaboration between Development and IT Operations to make software production and Deployment in an automated & repeatable way. DevOps helps increase the organization's speed to deliver software applications and services. The full form of 'DevOps' is a combination of 'Development' and 'Operations.'

It allows organizations to serve their customers better and compete more strongly in the market. In simple words, DevOps can be defined as an alignment of development and IT operations with better communication and collaboration.

DevOps lifecycle functions at each step:

1. **Plan -** Professionals determine the commercial need and gather end-user opinions throughout this level. In this step, they design a project plan to optimize business impact and produce the intended result.

2. **Code** - During this point, the code is being developed. To simplify the design process, the developer team employ lifecycle DevOps tools and extensions like Git that assist them in preventing safety problems and bad coding standards.
3. **Build** - After programmers have completed their task, they use tools such as Maven and Gradle to submit the code to the common code source.
4. **Test** - To assure software integrity, the product is first delivered to the test platform to execute various sorts of screening such as user acceptability testing, safety testing, integration checking, speed testing, and so on, utilizing tools such as JUnit, Selenium, etc.
5. **Release** - At this point, the build is prepared to be deployed in the operational environment. The DevOps department prepares updates or sends several versions to production when the build satisfies all checks based on the organizational demands.
6. **Deploy** - At this point, Infrastructure-as-Code assists in creating the operational infrastructure and subsequently publishes the build using various DevOps lifecycle tools.
7. **Operate** - This version is now convenient for users to utilize. With tools including Chef, the management department take care of server configuration and deployment at this point.
8. **Monitor** - The DevOps workflow is observed at this level depending on data gathered from consumer behavior, application efficiency, and other sources. The ability to observe the complete surroundings aids teams in identifying bottlenecks affecting the production and operations teams' performance.

Software Development Life Cycle:

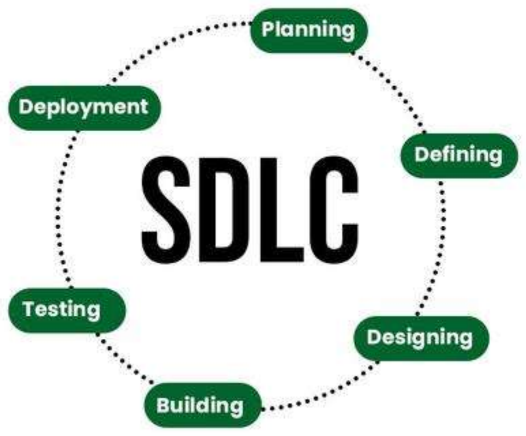
The software development life cycle (SDLC) is a structured process that is used to design, develop, and test good-quality software. SDLC, or software development life cycle is a methodology that defines the entire procedure of software development step-by-step. The goal of the SDLC life cycle model is to deliver high-quality, maintainable software that meets the user’s requirements. SDLC in software engineering models outlines the plan for each stage so that each stage of the software development model can perform its task efficiently to deliver the software at a low cost within a given time frame that meets users’ requirements.

SDLC stands for software development life cycle. It is a process followed for software building within a software organization. SDLC consists of a precise plan that describes how to develop, maintain, replace, and enhance specific software. The life cycle defines a method for improving the quality of software and the all-around development process.

Stages of the Software Development Life Cycle Model :

SDLC specifies the task(s) to be performed at various stages by a software engineer or developer. It ensures that the end product is able to meet the customer’s expectations and fits within the overall budget. Hence, it’s vital for a software developer to have prior knowledge of this software development process.

The **SDLC model involves six phases or stages** while developing any software. SDLC is a collection of these six stages, and the stages of SDLC are as follows:



Stages of Software Development Life Cycle Model

Stage-1: Planning and Requirement Analysis
Planning is a crucial step in everything, just as in software development. In this same stage, requirement analysis is also performed by the developers of the organization. This is attained from customer inputs, and sales department/market surveys.
The information from this analysis forms the building blocks of a basic project. The quality of the project is a result of planning. Thus, in this stage, the basic project is designed with all the available information.

Stage-2: Defining Requirements
In this stage, all the requirements for the target software are specified. These requirements get approval from customers, market analysts, and stakeholders.
This is fulfilled by utilizing SRS (Software Requirement Specification). This is a sort of document that specifies all those things that need to be defined and created during the entire project cycle.

Stage-3: Designing Architecture
SRS is a reference for software designers to come up with the best architecture for the software. Hence, with the requirements defined in SRS, multiple designs for the product architecture are present in the Design Document Specification (DDS).
This DDS is assessed by market analysts and stakeholders. After evaluating all the possible factors, the most practical and logical design is chosen for development.

Stage-4: Developing Product
At this stage, the fundamental development of the product starts. For this, developers use a specific programming code as per the design in the DDS. Hence, it is important for the coders to follow the protocols set by the association. Conventional programming tools like compilers, interpreters, debuggers, etc. are also put into use at this stage. Some popular languages like C/C++, Python, Java, etc. are put into use as per the software regulations.

Stage-5: Product Testing and Integration
After the development of the product, testing of the software is necessary to ensure its smooth execution. Although, minimal testing is conducted at every stage of SDLC. Therefore, at this stage, all the probable flaws are tracked, fixed, and retested. This ensures that the product confronts the quality requirements of SRS.

Documentation, Training, and Support: Software documentation is an essential part of the software development life cycle. A well-written document acts as a tool and means to information repository necessary to know about software processes, functions, and maintenance. Documentation also provides information about how to use the product. Training in an attempt to improve the current or future employee performance by increasing an employee's ability to work through learning, usually by changing his attitude and developing his skills and understanding.

Stage 6: Deployment and Maintenance of Products

After detailed testing, the conclusive product is released in phases as per the organization's strategy. Then it is tested in a real industrial environment. It is important to ensure its smooth performance. If it performs well, the organization sends out the product as a whole. After retrieving beneficial feedback, the company releases it as it is or with auxiliary improvements to make it further helpful for the customers. However, this alone is not enough. Therefore, along with the deployment, the product's supervision.

The waterfall model:

The waterfall model is a linear, sequential approach to the software development lifecycle (SDLC) that is popular in software engineering and product development.

The waterfall model uses a logical progression of SDLC steps for a project, similar to the direction water flows over the edge of a cliff. It sets distinct endpoints or goals for each phase of development. Those endpoints or goals can't be revisited after their completion.

The waterfall model continues to be used in industrial design applications. It's often cited as the first software development methodology. The model is also used more generally as a high-level project management methodology for complicated, multifaceted projects.

Phases of the waterfall model

When used for a software development process, the waterfall methodology has seven stages:

1. **Requirements**. Potential requirements, deadlines and guidelines for the project are analyzed and placed into a formal requirements document, also called a *functional specification*. This stage of development defines and plans the project without mentioning specific processes.
2. **Analysis.** The system specifications are analyzed to generate product models and business logic to guide production. This is also when financial and technical resources are audited for feasibility.

3. **Design.** A design specification document is created to outline technical design requirements, such as the programming language, hardware, data sources, architecture and services.
4. **Coding and implementation.** The source code is developed using the models, logic and requirement specifications designated in the prior phases. Typically, the system is coded in smaller components, or units, before being put together.
5. **Testing.** This is when quality assurance, unit, system and beta tests identify issues that must be resolved. This may cause a forced repeat of the coding stage for debugging. If the system passes integration and testing, the waterfall continues forward.
6. **Operation and deployment.** The product or application is deemed fully functional and is deployed to a live environment.
7. **Maintenance.** Corrective, adaptive and perfective maintenance is carried out indefinitely to improve, update and enhance the product and its functionality. This could include releasing patch updates and new versionproject.

Before moving to the next phase in the waterfall process, there's usually a review and sign off to ensure all defined goals have been met. For example, developers would ensure each unit of technology is properly integrated in the implementation phase before moving to the testing phase.

Who uses the waterfall model?

Project teams and project managers use the waterfall model to achieve goals based on the needs of their business. The model is used in many different project management contexts, such as in construction, manufacturing, IT and software development.

In the waterfall method, each step is dependent on the output of the previous step. There's a linear progression to the way these projects unfold.

For example, in construction, these three general steps are usually followed:

1. A building's physical design is created before any construction begins.

2. The foundation is poured before the skeleton of a building is erected.
3. The skeleton of the building is completed before the walls are built.

Advantages of the waterfall model

Today, Agile methodology is often used in place of the waterfall model. However, there are advantages to the waterfall approach, such as the following:

- enables large or changing teams to move toward a common goal that's been defined in the requirements stage;
- forces structured, disciplined organization;
- simplifies understanding, following and arranging tasks;
- facilitates departmentalization and managerial control based on the schedule or deadlines;
- reinforces good coding habits to define before implementing design and then code;
- enables early system design and specification changes to be easily done; and
- clearly defines milestones and deadlines.

Disadvantages of the waterfall model

Disadvantages of the waterfall model typically center around the risk associated with a lack of revision and flexibility. Specific issues include the following:

- Design isn't adaptive; when a flaw is found, the entire process often needs to start over.
- Method doesn't incorporate midprocess user or client feedback, and makes changes based on results.
- Waterfall model delays testing until the end of the development lifecycle.
- It doesn't consider error correction.
- The methodology doesn't handle requests for changes, scope adjustments and updates well.
- Waterfall doesn't let processes overlap for simultaneous work on different phases, reducing overall efficiency.

- No working product is available until the later stages of the project lifecycle.
- Waterfall isn't ideal for complex, high-risk ongoing projects.

Agile Model:

The Agile Model was primarily designed to help a project adapt quickly to change requests. So, the main aim of the Agile model is to facilitate quick project completion. To accomplish this task, agility is required. Agility is achieved by fitting the process to the project and removing activities that may not be essential for a specific project. Also, anything that is a waste of time and effort is avoided.
The Agile Model refers to a group of development processes. These processes share some basic characteristics but do have certain subtle differences among themselves.

In the Agile model, the requirements are decomposed into many small parts that can be incrementally developed. The Agile model adopts Iterative development. Each incremental part is developed over an iteration. Each iteration is intended to be small and easily manageable and can be completed within a couple of weeks only. At a time one iteration is planned, developed, and deployed to the customers. Long-term plans are not made.

Steps in the Agile Model
The agile model is a combination of iterative and incremental process models. The steps involve in agile SDLC models are:
- Requirement gathering
- Design the Requirements
- Construction / Iteration
- Testing / Quality Assurance
- Deployment
- Feedback

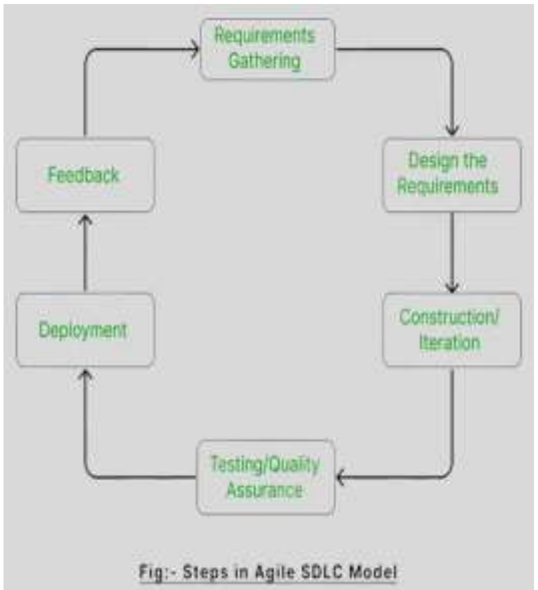
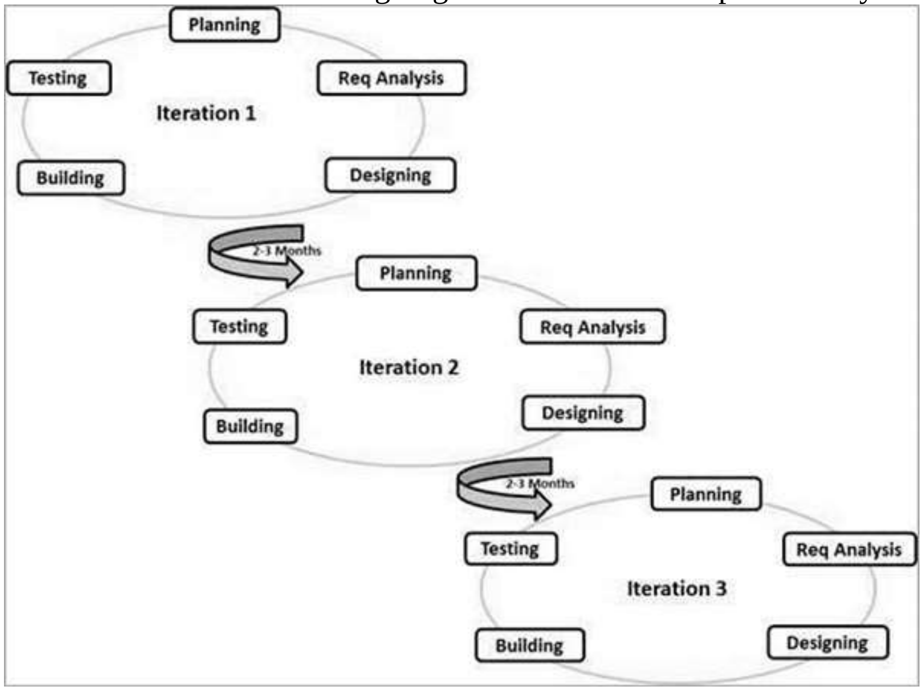


Fig:- Steps in Agile SDLC Model

1. Requirement Gathering:- In this step, the development team must gather the requirements, by interaction with the customer. development team should plan the time and effort needed to build the project. Based on this information you can evaluate technical and economical feasibility.
2. Design the Requirements:- In this step, the development team will use user-flow-diagram or high-level UML diagrams to show the working of the new features and show how they will apply to the existing software. Wireframing and designing user interfaces are done in this phase.
3. Construction / Iteration:- In this step, development team members start working on their project, which aims to deploy a working product.
4. Testing / Quality Assurance:- Testing involves Unit Testing, Integration Testing, and System Testing**.** A brief introduction of these three tests is as follows:
5. Unit Testing:- Unit testing is the process of checking small pieces of code to ensure that the individual parts of a program work properly on their own. Unit testing is used to test individual blocks (units) of code.

- **Integration Testing:-** Integration testing is used to identify and resolve any issues that may arise when different units of the software are combined.
- **System Testing:-** Goal is to ensure that the software meets the requirements of the users and that it works correctly in all possible scenarios.

5. Deployment:- In this step, the development team will deploy the working project to end users.
6. Feedback:- This is the last step of the **Agile Model.** In this, the team receives feedback about the product and works on correcting bugs based on feedback provided by the customer.



The time required to complete an iteration is known as a Time Box. Time-box refers to the maximum amount of time needed to deliver an iteration to customers. So, the end date for an iteration does not change.

Characteristics of the Agile Process:

- Agile processes must be adaptable to technical and environmental changes. That means if any technological changes occur, then the agile process must accommodate them.
- The development of agile processes must be incremental. That means, in each development, the increment should contain some functionality that can be tested and verified by the customer.
- The customer feedback must be used to create the next increment of the process.
- The software increment must be delivered in a short span of time.
- It must be iterative so that each increment can be evaluated regularly.

Advantages of the Agile Model

- Working through Pair programming produces well-written compact programs which have fewer errors as compared to programmers working alone.
- It reduces the total development time of the whole project.
- Agile development emphasizes face-to-face communication among team members, leading to better collaboration and understanding of project goals.
- Customer representatives get the idea of updated software products after each iteration. So, it is easy for him to change any requirement if needed.
- Agile development puts the customer at the center of the development process, ensuring that the end product meets their needs.

Disadvantages of the Agile Model:

- The lack of formal documents creates confusion and important decisions taken during different phases can be misinterpreted at any time by different team members.
- It is not suitable for handling complex dependencies.
- The agile model depends highly on customer interactions so if the customer is not clear, then the development team can be driven in the wrong direction.
- Agile development models often involve working in short sprints, which can make it difficult to plan and forecast project timelines and deliverables. This can lead to delays in the project and can make it difficult to accurately estimate the costs and resources needed for the project.
- Agile development models require a high degree of expertise from team members, as they need to be able to adapt to changing requirements and work in an iterative environment. This can be challenging for teams that are not experienced in agile development practices and can lead to delays and difficulties in the project.

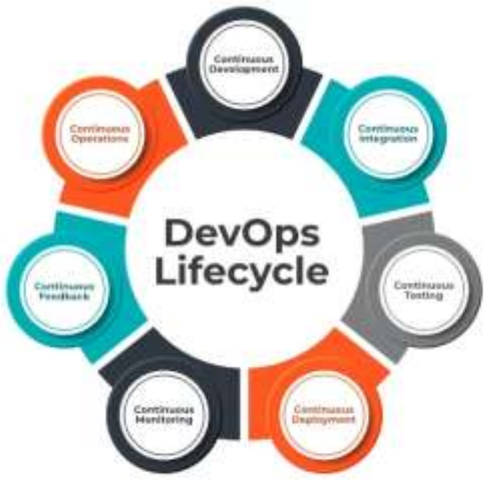
DevOps LifeCycle:

DevOps is a practice that enables a single team to handle the whole application lifecycle, including development, testing, release, deployment, operation, display, and planning. It is a mix of the terms “Dev” (for development) and “Ops” (for operations).

DevOps Lifecycle is the set of phases that includes DevOps for taking part in Development and Operation group duties for quicker software program delivery. DevOps follows positive techniques that consist of **code, building, testing, releasing, deploying, operating, displaying, and planning. DevOps lifecycle** follows a range of phases such as non-stop development, non-stop integration, non-stop testing, non-stop monitoring, and non-stop feedback. Each segment of the DevOps lifecycle is related to some equipment and applied sciences to obtain the process.

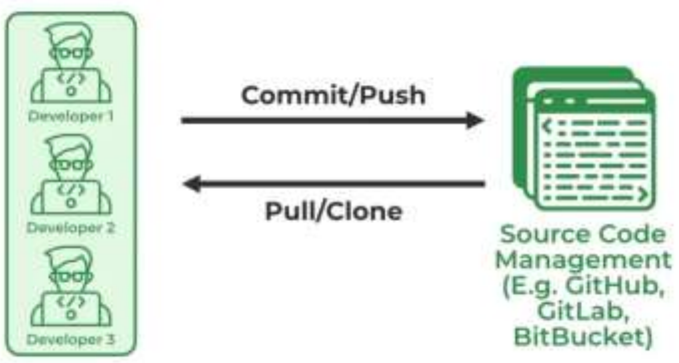
7Cs of DevOps

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



1. Continuous Development:

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

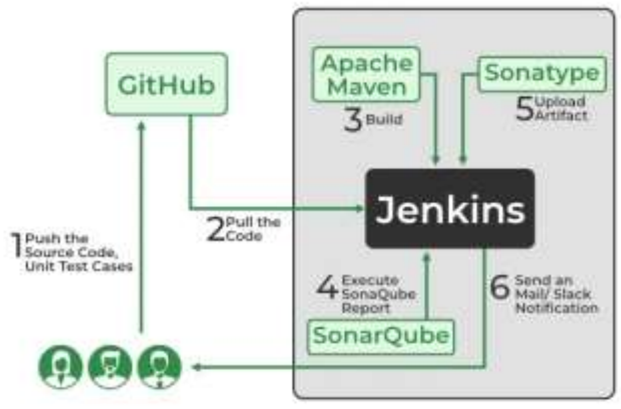


2. Continuous Integration:

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

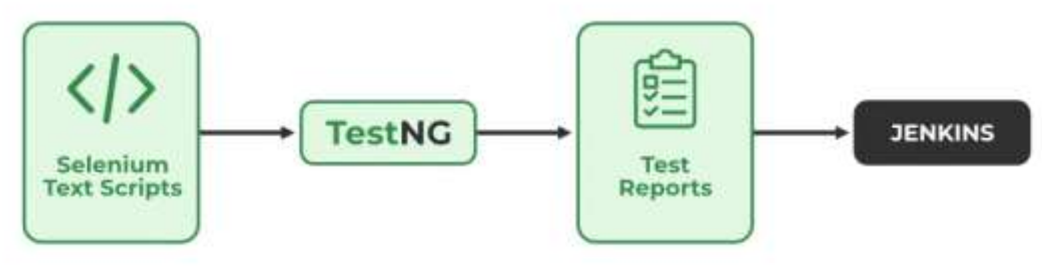
1. Getting the SourceCode from SCM
2. Building the code
3. Code quality review
4. Storing the build artifacts

The stages mentioned above are the flow of Continuous Integration and we can use any of the tools that suit our requirement in each stage and of the most popular tools are **GitHub for source code management(SCM)** when the developer develops the code on his local machine he pushes it to the remote repository which is GitHub from here who is having the access can Pull, clone and can make required changes to the code.
Nexus for storing the build artifacts will help us to store the artifacts that are build by using Maven and this whole process is achieved by using a Continuous Integration tool Jenkins.


GitHub
Apache Maven
Sonatype
3 Build
5 Upload Artifact
Jenkins
2 Pull the Code
1 Push the Source Code, Unit Test Cases
4 Execute SonaQube Report
6 Send an Mail/ Slack Notification
SonarQube

3. Continuous Testing

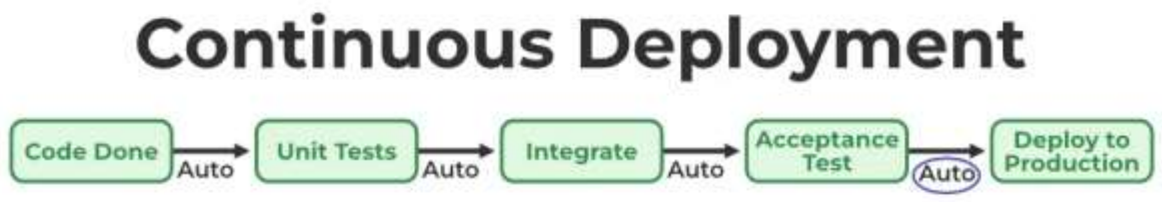
Any firm can deploy continuous testing with the use of the agile and DevOps methodologies. Depending on our needs, we can perform continuous testing using automation testing tools such as **Testsigma, Selenium, LambdaTest,** etc. With these tools, we can test our code and prevent problems and code smells,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



4. Continuous Deployment/ Continuous Delivery

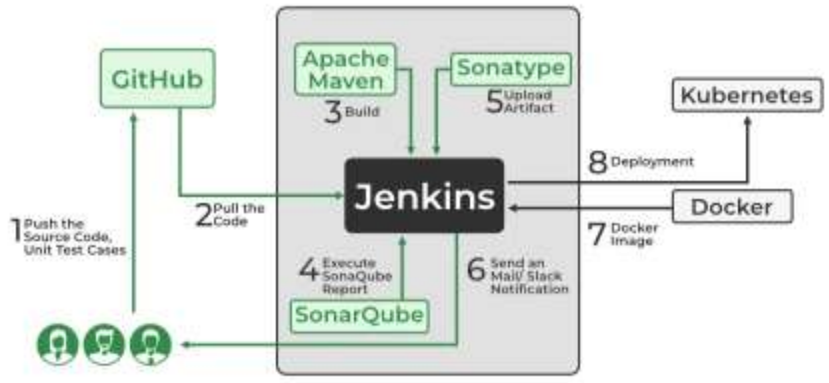
Continuous Deployment:

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



Continuous Delivery:

Continuous Delivery is the process of deploying an application into production servers manually when it has completed testing and the build stages. Here, we'll automate the continuous integration processes, however, manual involvement is still required for deploying it to the production environment.



5.Continuous Monitoring

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

6. Continuous Feedback

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

7. Continuous Operations

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

<u>DevOps Tools:</u>

DevOps tools help simplify and accelerate testing, configuration, deployment, and other software-related tasks required to implement DevOps processes.

Some DevOps tools provide the ability to identify and resolve errors and defects at high velocity and scale. Others add automation to processes such as monitoring, testing, deployment, updates, and infrastructure management, while others facilitate information sharing and improve collaboration.

An effective DevOps toolchain can improve the quality, stability, and reliability of applications, and can help identify and resolve problems earlier in the development lifecycle.

In general, DevOps tools provide the following benefits:

- **Reduce** repetitive work and optimize processes
- **Organize** work into structured processes
- **Improve** collaboration and communication between teams
- **Simplify** work processes, saving time and effort
- **Prevent** human errors and misjudgements via automation, while allowing human oversight of processes where necessary

Here are some most popular DevOps tools with brief explanation shown in the below such as:

1) Puppet

Puppet is the most widely used DevOps tool. It allows the delivery and release of the technology changes quickly and frequently. It has features of versioning, automated testing, and continuous delivery. It enables to manage entire infrastructure as code without expanding the size of the team.

Features

- Real-time context-aware reporting.
- Model and manage the entire environment.
- Defined and continually enforce infrastructure.
- Desired state conflict detection and remediation.
- It inspects and reports on packages running across the infrastructure.
- It eliminates manual work for the software delivery process.
- It helps the developer to deliver great software quickly.

2) Ansible

Ansible is a leading DevOps tool. Ansible is an open-source IT engine that automates application deployment, cloud provisioning, intra service orchestration, and other IT tools. It makes it easier for DevOps teams to scale automation and speed up productivity.

Features

- It is easy to use to open source deploy applications.
- It helps in avoiding complexity in the software development process.
- It eliminates repetitive tasks.
- It manages complex deployments and speeds up the development process.

3) Docker

Docker is a high-end DevOps tool that allows building, ship, and run distributed applications on multiple systems. It also helps to assemble the apps quickly from the components, and it is typically suitable for container management.

Features

- It configures the system more comfortable and faster.
- It increases productivity.
- It provides containers that are used to run the application in an isolated environment.
- It routes the incoming request for published ports on available nodes to an active container. This feature enables the connection even if there is no task running on the node.
- It allows saving secrets into the swarm itself.

4) Nagios

Nagios is one of the more useful tools for DevOps. It can determine the errors and rectify them with the help of network, infrastructure, server, and log monitoring systems.

Features

- It provides complete monitoring of desktop and server operating systems.
- The network analyzer helps to identify bottlenecks and optimize bandwidth utilization.
- It helps to monitor components such as services, application, OS, and network protocol.
- It also provides to complete monitoring of Java Management Extensions.

5.SALTSTACK

Stackify is a lightweight DevOps tool. It shows real-time error queries, logs, and more directly into the workstation. SALTSTACK is an ideal solution for intelligent orchestration for the software-defined data center.

Features

- It eliminates messy configuration or data changes.
- It can trace detail of all the types of the web request.
- It allows us to find and fix the bugs before production.
- It provides secure access and configures image caches.
- It secures multi-tenancy with granular role-based access control.
- Flexible image management with a private registry to store and manage images.

<u>Distributed version of control tool Git:</u>

Git is a distributed version control system. Each developer has a copy of the source repository on their dev machine. Developers can commit each set of changes on their dev machine and perform version control operations such as history and compare without a network connection.

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

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

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

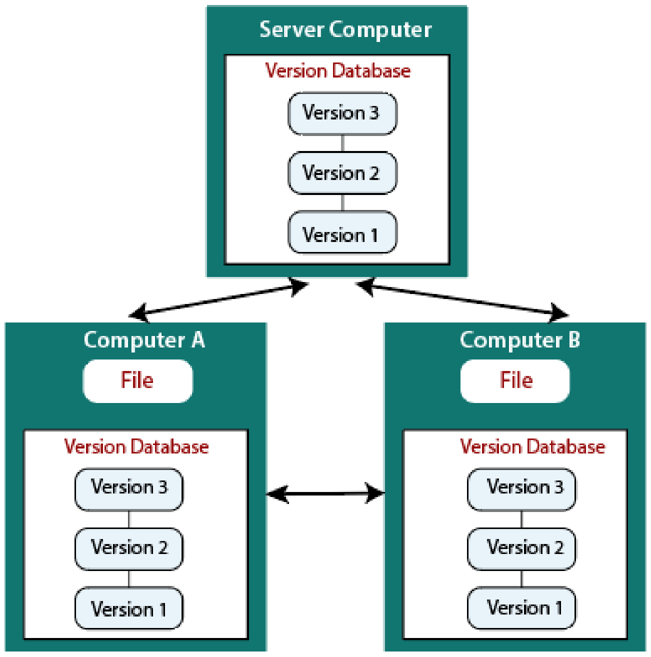
Types of Version Control System:

- Localized version Control System
- Centralized version control systems
- Distributed version control systems

Distributed Version Control System:

Centralized Version Control System uses a central server to store all the database and team collaboration. But due to single point failure, which means the failure of the central server, developers do not prefer it. Next, the Distributed Version Control System is developed.

In a Distributed Version Control System (such as Git, Mercurial, Bazaar or Darcs), the user has a local copy of a repository. So, the clients don't just check out the latest snapshot of the files even they can fully mirror the repository. The local repository contains all the files and metadata present in the main repository.



DVCS allows automatic management branching and merging. It speeds up of most operations except pushing and pulling. DVCS enhances the ability to work offline and does not rely on a single location for backups. If any server stops and other systems were collaborating via it, then any of the client repositories could be restored by that server. Every checkout is a full backup of all the data.

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

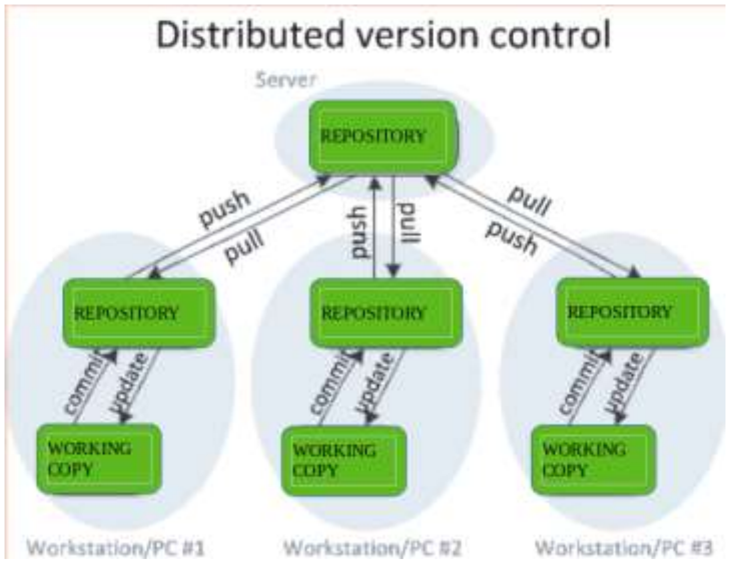
- In DVCS, Every user has a local copy of the repository in place of the central repository on the server-side.
- It is based on the client-server approach.
- It is flexible and has emerged with the concept that everyone has their repository.
- In DVCS, every user can check out the snapshot of the code, and they can fully mirror the central repository.
- DVCS is fast comparing to CVCS as you don't have to interact with the central server for every command.
- The popular tools of DVCS are Git and Mercurial.
- DVCS has some complex process for beginners.
- If any server fails and other systems were collaborating via it, that server can restore any of the client repositories

To make your changes visible to others, 4 things are required:

- You commit
- You push
- They pull

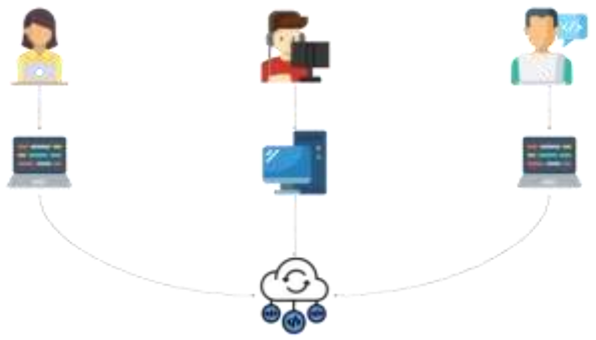
- They update

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



If the server becomes unavailable or dies, no worries! Any of the client repositories can send a copy of the project's version to any other client or back onto the server when it becomes available. We just need one client that contains a correct copy, which can then be distributed easily.

Git is a prime example of a distributed version control system.



Advantages of Distributed Version Control System:

- Other than push and pull, all actions can be performed very quickly, since it is the hard drive, and not the remote server that is accessed every time.
- Changesets can be committed to the local repository first and then a group of these changesets can be pushed to the central repository in a single shot.

- Only the pushing and pulling activities need internet connectivity; everything else can be managed locally.
- Every developer has a complete copy of the entire repository and the impact any change can be checked locally before the code is pushed to the central repository.
- DVCS is built to handle changes efficiently, since every change has a Global Unique Identifier (GUID) that makes it easy to track.
- Tasks like branching and merging can be done with ease, since every developer has their own branch and every shared change is like reverse integration
- DVCS is very easy to manage compared to CVCS.

Disadvantages of Distributed Version Control System:

- With many projects, large binary files that are difficult to compress, will occupy more space.
- Projects with a long history, i.e., a large number of changesets may take a lot of time and occupy more disk space.
- With DVCS, a backup is still needed, since the latest updated version may not be available to all the developers.
- Though DVCS doesn't prevent having a central server, not having a central server might cause confusions in identifying the right recent version.
- Though every repo has its own revision numbers, releases have to be tagged with appropriate names to avoid confusions.

Automation testing tools:

A test automation tool is a piece of software that enables people to define software testing tasks, that are afterwards run with as little human interaction as possible.

The automation testing is used to change the manual test cases into a test script with the help of some automation tools.

We have various types of automation testing tools available in the market. Some of the most commonly used automation testing tools are as follows:

- **Selenium**

- **Watir**
- **QTP**
- **Telerik Studio**
- **Testim**
- **Applitools**

Selenium:

It is an open-source and most commonly used tool in automation testing. This tool is used to test web-based applications with the help of test scripts, and these scripts can be written in any programming language such as java, python, C#, Php, and so on.

Features of Selenium

This tool is most popular because of its various features. Following are standard features of Selenium:

- Selenium supports only web-based application, which means that the application can be opened by the browser or the URL like Gmail, Amazon, etc.
- Selenium does not support a stand-alone application, which means that the application is not opened in the browser or URL like Notepad, MS-Word, Calculator, etc.
- Selenium web driver is the latest tool in the selenium community that removes all the drawbacks of the previous tool (selenium-IDE).
- Selenium web-driver is powerful because it supports multiple programming languages, various browsers, and different operating systems and also supports mobile applications like iPhone, Android.

Watir:

Watir stands for **web application testing in ruby** which is written in the Ruby programming language. testing in ruby. It is a web application testing tool, which is open-source and supports cross-browser testing tool and interacts with a platform like a human that can validate the text, click on the links and fill out the forms.

Features of Watir:

Following are the characteristics of Watir testing tools:

- It supports various browsers on different platforms like Google Chrome, Opera, Firefox, Internet Explorer, and Safari.
- Watir is a lightweight and powerful tool.
- We can easily download the test file for the UI.
- We can take the screenshots once we are done with the testing, which helps us to keep track of the intermediate testing.
- This tool has some inbuilt libraries, which helps to check the alerts, browser windows, page-performance, etc.

QTP:

QTP tool is used to test functional regression test cases of the web-based application. QTP stands for **Quick Test Professional**, and now it is known as **Micro Focus UFT [Unified Functional Testing]**. This is very helpful for the new test engineer because they can understand this tool in a few minutes. QTP is designed on the scripting language like VB script to automate the application.

Features of QTP:

Following are the most common features of QTP:

- This tool support record and playback feature.
- QTP uses the scripting language to deploy the objects, and for analysis purposes, it provides test reporting.
- Both technical and non-technical tester can use QTP.
- QTP supports multiple software development environments like Oracle, SAP, JAVA, etc.
- With the help of QTP, we can test both desktop and web-based applications.
- In this tool, we can perform BPT (Business process testing).

Telerik Studio:

It is modern web application which support functional test automation. With the help of this tool, we can test the load, performance, and functionality of the web and mobile applications and also identify the cross-browser issues.

Features of Telerik test studio:

Below are some essential features of Telerik test studio:

- Telerik test studio allows us to deliver quality products on time.
- This tool supports all types of applications, such as desktop, web, and mobile.
- This tool supports Asp.Net, AJAX, HTML, JavaScript, WPF, and Silverlight application testing.
- It supports multiple browsers like Firefox, Safari, Google Chrome, and Internet Explorer for the test execution process.
- With the help of this tool, we can do the sentence based UI validation.

Testim:

It is another automation testing tool, which can execute the test case in very little time and run them in various web and mobile applications. This tool will help us to enhance the extensibility and stability of our test suites. It provides the flexibility to cover the functionalities of the platform with the help of JavaScript and HTML.

Features of Testim:

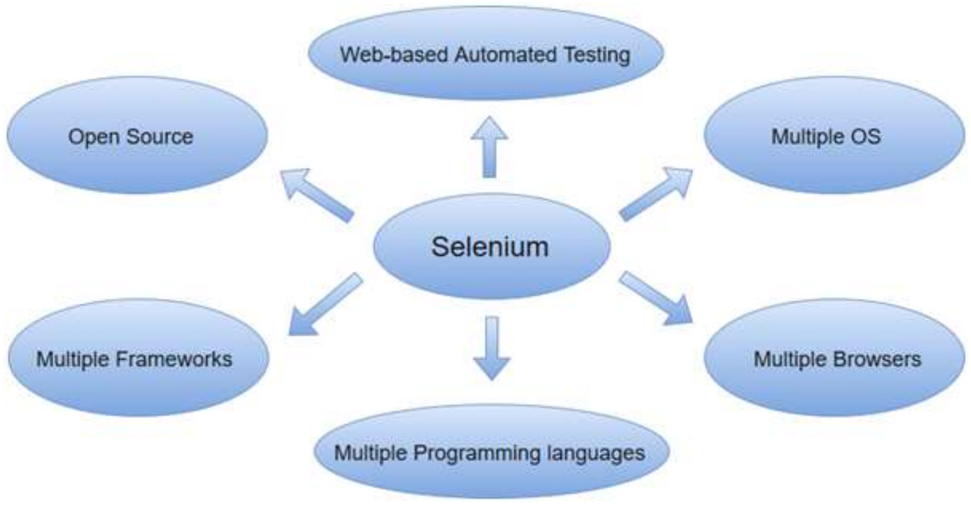
- The Test stability is very high in testim tool.
- This tool will support parallel execution.
- In this tool, we can capture the screenshots.
- This tool will automatically create the tests.
- With the help of this tool, we can perform requirements-based and parameterized testing.

Selenium:

Selenium is one of the most widely used open source Web UI (User Interface) automation testing suite.It was originally developed by Jason Huggins in 2004 as an internal tool at Thought Works. Selenium supports automation across different browsers, platforms and programming languages.

Selenium can be easily deployed on platforms such as Windows, Linux, Solaris and Macintosh. Moreover, it supports OS (Operating System) for mobile applications like iOS, windows mobile and android.

Selenium supports a variety of programming languages through the use of drivers specific to each language.Languages supported by Selenium include C#, Java, Perl, PHP, Python and Ruby.Currently, Selenium Web driver is most popular with Java and C#.



Selenium can be used to automate functional tests and can be integrated with automation test tools such as **Maven**, **Jenkins**, **& Docker** to achieve continuous testing. It can also be integrated with tools such as **TestNG**, & **JUnit** for managing test cases and generating reports.

Selenium Features:

- Selenium is an open source and portable Web testing Framework.
- Selenium IDE provides a playback and record feature for authoring tests without the need to learn a test scripting language.
- It can be considered as the leading cloud-based testing platform which helps testers to record their actions and export them as a reusable script with a simple-to-understand and easy-to-use interface.
- Selenium supports various operating systems, browsers and programming languages. Following is the list:
    - Programming Languages: C#, Java, Python, PHP, Ruby, Perl, and JavaScript
    - Operating Systems: Android, iOS, Windows, Linux, Mac, Solaris.
    - Browsers: Google Chrome, Mozilla Firefox, Internet Explorer, Edge, Opera, Safari, etc.
- It also supports parallel test execution which reduces time and increases the efficiency of tests.
- Selenium can be integrated with frameworks like Ant and Maven for source code compilation.

- Selenium can also be integrated with testing frameworks like TestNG for application testing and generating reports.
- Selenium requires fewer resources as compared to other automation test tools.
- WebDriver API has been indulged in selenium whichis one of the most important modifications done to selenium.
- Selenium web driver does not require server installation, test scripts interact directly with the browser.
- Selenium commands are categorized in terms of different classes which make it easier to understand and implement.
- Selenium Remote Control (RC) in conjunction with WebDriver API is known as Selenium 2.0. This version was built to support the vibrant web pages and Ajax.

Selenium Basic Terminology:

Before proceeding with this tutorial, let us first understand some of the key concepts associated with Automation testing of an application.

With the growing need for efficient software products, every software development group need to carry out a series of tests before launching the final product into the market.

Automation testing is the best way to increase the effectiveness, efficiency and coverage of your software testing.

Automation Testing:

Automation testing uses the specialized tools to automate the execution of manually designed test cases without any human intervention. Automation testing tools can access the test data, controls the execution of tests and compares the actual result against the expected result. Consequently, generating detailed test reports of the system under test.

Automation testing covers both functional and performance test on an application.

- Functional automation is used for automation of functional test cases. For example, regression tests, which are repetitive in nature, are automated.
- Performance automation is used for automation of non-functional performance test cases. For example, measuring the response time of the application under considerable (say 100 users) load.

Automation Testing tools which are used for functional automation:

- Quick Test Professional, provided by HP.
- Rational Robot, provided by IBM.
- Coded UI, provided by Microsoft.
- Selenium, open source.
- Auto It, open Source.

Automation Testing tools which are used for non-functional automation:

- Load Runner, provided by HP.
- JMeter, provided by Apache.
- Burp Suite, provided by PortSwigger.
- Acunetix, provided by Acunetix.

Selenium Limitations

- Selenium does not support automation testing for desktop applications.
- Selenium requires high skill sets in order to automate tests more effectively.
- Since Selenium is open source software, you have to rely on community forums to get your technical issues resolved.
- We can't perform automation tests on web services like SOAP or REST using Selenium.
- We should know at least one of the supported programming languages to create tests scripts in Selenium WebDriver.
- It does not have built-in Object Repository like UTF/QTP to maintain objects/elements in centralized location. However, we can overcome this limitation using Page Object Model.
- Selenium does not have any inbuilt reportingcapability; you have to rely on plug-ins like **JUnit** and **TestNG** for test reports.
- It is not possible to perform testing on images. We need to integrate Selenium with **Sikuli** for image based testing.
- Creating test environment in Selenium takes more time as compared to vendor tools like UFT, RFT, Silk test, etc.
- No one is responsible for new features usage; they may or may not work properly.
- Selenium does not provide any test tool integration for Test Management.

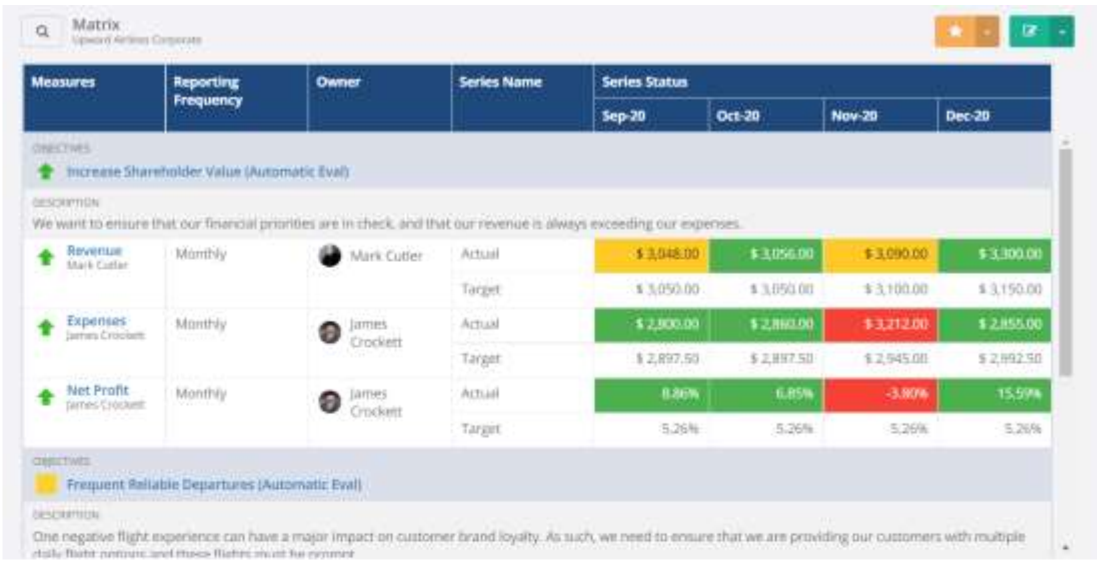
Report generation:

A computer program is referred to as a report generator. The purpose of this computer program is to accept information or data from the database, spreadsheet, or XML stream which are the source, and then utilize the data for producing a structured composition satisfying the readership of a specific human. The process in which reports are made by using a tool for users related to business is called **report generation** and the software used for this method is known as a **report generator**. For producing reports, a report generator is more proficient and suitable as compared to excel.

Need of Report Generation:

The necessity of making reports manually gets eliminated by report generation. Report generation also reduces the chance of mistakes. It allows us to examine the information/data. Explanation of the report which includes retrieving the type of data, data location, also the strategy for showing that data, all is done by the report generator.

Report generator permits operation of a report with the report processor and the data is removed with the given definition of the report and joining data with the layout of the report for report creation. The report generation tool not only allows the preview of the report but also its publication to a server to transfer later on. From that point onward, the data can be reported to your colleagues, directors, or accomplices.



Measures	Reporting Frequency	Owner	Series Name	Series Status			
				Sep-20	Oct-20	Nov-20	Dec-20
OBJECTIVES Increase Shareholder Value (Automatic Eval)							
DESCRIPTION We want to ensure that our financial priorities are in check, and that our revenue is always exceeding our expenses.							
Revenue Mark Cutler	Monthly	Mark Cutler	Actual	$ 3,048.00	$ 3,056.00	$ 3,090.00	$ 3,300.00
			Target	$ 3,050.00	$ 3,050.00	$ 3,100.00	$ 3,150.00
Expenses James Crockett	Monthly	James Crockett	Actual	$ 2,800.00	$ 2,860.00	$ 3,212.00	$ 2,855.00
			Target	$ 2,897.50	$ 2,897.50	$ 2,945.00	$ 2,992.50
Net Profit James Crockett	Monthly	James Crockett	Actual	8.86%	6.85%	-3.80%	15.59%
			Target	5.26%	5.26%	5.26%	5.26%
OBJECTIVES Frequent Reliable Departures (Automatic Eval)							
DESCRIPTION One negative flight experience can have a major impact on customer brand loyalty. As such, we need to ensure that we are providing our customers with multiple							

Features of Report Generator:

1. For every phase of report generation, the report generator is user-friendly and effective.
2. From numerous sources of data, report generators can easily extract information.
3. Report generator operators with real-time work. The reports are automatically generated after arranging templates and report the frequency of annual, quarterly, monthly, and day to day reports and are sent to the email address that is set.
4. Report generator supports the reuse of templates to generate reports.

5. Printing or exporting of reports is supported by the report generator. The report can be exported or printed in pdf, images, or excel.
6. Users can review the reports anywhere and anytime by their phones with the help of the report generator.

There are two types of the report generation process, which are as follows:

1. Full-automatic generation
2. Semi-automatic generation

1.Full-automatic Generation: Full automatic generation is a template-based process. YoY (Year over Year), MoM (Month over Month) ranking is the often utilized monetary insights. Take the most often involved for instance, including MoM, YoY Ranking. At the point when the report generator is used, layouts of monetary statistics have been created inside. The full-automatic generation just requires us to link to the database and drag it to the relating cell.

2. Semi-automatic Generation: Each module is generated automatically in semi-automatic generation by using professional functions. Semi-automatic generation is common and can more likely meet clients' customized requirements as compared to full-automatic report generation.

In general, the complete process of creating a report is divided into three steps:

• Connection to database – including opening designers, configuring data sources, designing a new report, and configuring private data sources.

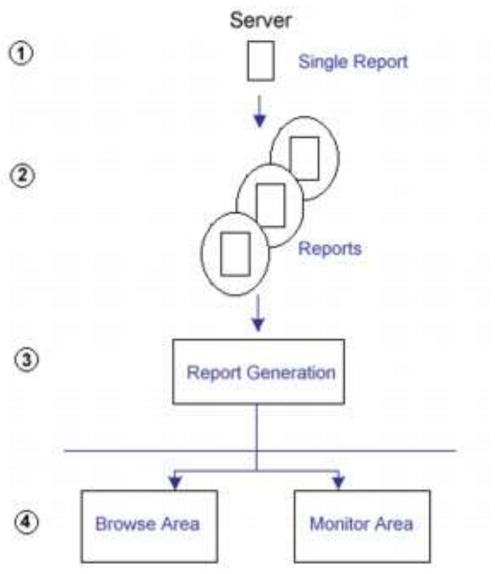
• Report design – including generating new reports, specifying data sources, binding data columns, summarizing, and formatting reports.

• Publishing and browsing reports – including viewing, saving, and publishing reports.

Report generation process:

Describes the procedure to generate a report after a report has been submitted to run.

The following figure shows the report generation process after a report has been submitted to run.



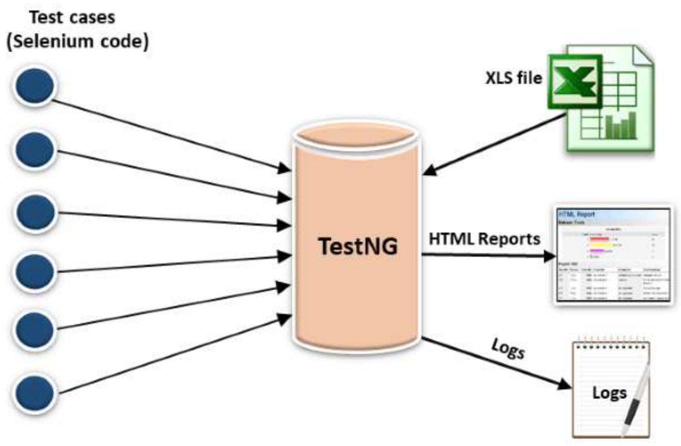
- Reports are submitted to run on the server.
- The Tivoli® Netcool® Performance Manager software treats all reports as report groups. Each report group is given a group number (job ID), and each report within the group is given a report number (task ID) and a report name. Report groups are queued to be run.
- A report group, and so all the reports within the group is run.
- If a report within the report group is successfully generated, the report output becomes available in **Monitor** tab. If it is a local report, it also becomes available in the **Browse** tab.

If report generation is unsuccessful, report status and auxiliary information about the run become available in the **Monitor** tab.

TestNG:

- TestNG is a very important framework when you are actually developing the framework from scratch level.
- TestNG provides you full control over the test cases and the execution of the test cases. Due to this reason, TestNG is also known as a testing framework.
- Cedric Beust is the developer of a TestNG framework.
- If you want to run a test case A before that as a pre-request you need to run multiple test cases before you begin a test case A. You can set and map with the help of TestNG so that pre-request test cases run first and then only it will trigger a test case A. In such way, you can control the test cases.

- TestNG framework came after Junit, and TestNG framework adds more powerful functionality and easier to use.
- It is an open source automated TestNG framework. In TestNG, NG stands for "**Next Generation**".
- TestNG framework eliminates the limitations of the older framework by providing more powerful and flexible test cases with help of easy annotations, grouping, sequencing and parametrizing.



TestNG Features:

- Supports annotations.
- TestNG uses more Java and OO features.
- Supports testing integrated classes (e.g., by default, no need to create a new test class instance for every test method).
- Separates compile-time test code from run-time configuration/data info.
- Flexible runtime configuration.
- Introduces ‘test groups’. Once you have compiled your tests, you can just ask TestNG to run all the "front-end" tests, or "fast", "slow", "database" tests, etc.
- Supports Dependent test methods, parallel testing, load testing, and partial failure.
- Flexible plug-in API.
- Support for multi threaded testing.

Uses of TestNG:

- **Flexibility and extensibility:** TestNG is more flexible than other frameworks, and it provides features that make it easier to It allows you run parallel execution of tests

- **Ease of use:** TestNG is designed to be more user-friendly, as it defines dependency between one method on another (so if a failure occurs in the middle).
- **Comprehensive coverage:** TestNG helps you write better, more extensive tests. In TestNG, we can group our methods together into groups with specific priorities for each purpose - for example: high/medium level checking versus low-level ones like logging exceptions or returning true conditions.

How To Install TestNG in eclipse and Configure TestNG:

This section covers all the details regarding how to install and configure TestNG on your eclipse IDE. To make sure your tests are designed to run efficiently, follow the steps below to set up TestNG on your machine:

Install TestNG:

To install TestNG, you can use the following command:

```
> Sudo apt-get install TestNG
```

Add TestNG to your project:

To add TestNG to your project, you can add the following dependency to your pom.xml file:

```
<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>6.12.3</version> </dependency>
```

Configure TestNG in your project:

To run tests with TestNG, you need to configure it in your project. You can do this by adding the following XML to your pom.xml file:

```
<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>
```

<artifactId>maven-surefire-plugin</artifactId>

<version>2.19</version>

<configuration>

<suiteXmlFiles>

<suiteXmlFile/> </suiteXmlFiles></configuration></plugin></plugins></build>

Run TestNG tests:

Once you have configured TestNG in your project, you can run your tests by using the following command:

mvn test

This will run all of the tests in your project. You can also use the following command to run a specific test:

mvn -Dtest=MyTest test

How to use TestNG Framework:

Now that we've taken a look at some of the key features of TestNG let's see how we can use it to write better tests. This section will look at how to use annotations, parameters, and groups in TestNG.

Annotations:

Annotation is a powerful feature of testing that allows you to specify how your test should be run. Annotations can be used to configure your test, set up dependencies, and group your tests. You can make your tests more modular and easier to maintain by using annotations.

TestNG uses annotations to identify tests, set priorities, and configure other aspects of how the tests should be run. This section will look at some of the most commonly used TestNG annotations.

@BeforeMethod:

The @BeforeMethod annotation denotes a method that should be run before each test. This can be used to set up the environment for the test or perform any other actions that need to be done before the test is run.

public void setup() {

// do something

}

@AfterMethod :

The @AfterMethod annotation denotes a method that should be run after each test. This can be used to clean up the environment after the test is run or perform any other actions that need to be done after the test is finished.

public void tearDown() {

// do something

}

@Test:

The @Test annotation is used to denote a test method. This method will be executed when the test is run.

public void myTest() {

// do something }

Advantages of TestNG:

- It provides a powerful and wide variety of annotations to support your test cases.
- Annotations are easy to understand.
- Supports parameterized and dependency tests.
- Support for data-driven testing.
- Supported by a variety of tools and plug-ins (Eclipse, IDEA, Maven, etc).
- It helps to perform parallel testing, dependent method testing.
- It offers the flexibility to run your tests through multiple data sets through the TestNG.xml file or via the data-provider concept.
- Test cases can be grouped and prioritized as per need basis.
- Provides access to HTML reports and can be customized through various plugins.
- With its Test logs can be generated across tests.

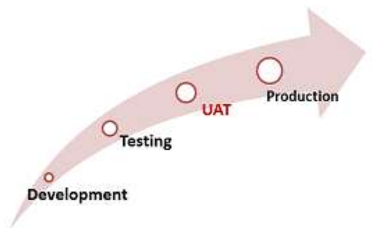
Disadvantages of TestNG:

- Setup of TestNG is time-consuming.
- If your project does not require test case prioritization that case TestNG is of no use.
- In the olden day, it was less commonly used compared to JUnit, so fewer people was having experience with it.

<u>User Acceptance Testing:</u>

User Acceptance Testing (UAT) is a type of testing performed by the end user or the client to verify/accept the software system before moving the software application to the production environment. UAT is done in the final phase of testing after functional, integration and system testing is done.

Purpose of UAT:



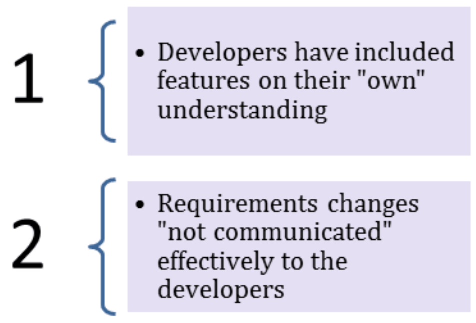
The main **Purpose of UAT** is to validate end to end business flow. It does not focus on cosmetic errors, spelling mistakes or system testing. User Acceptance Testing is carried out in a separate testing environment with production-like data setup. It is kind of black box testing where two or more end-users will be involved.

UAT is performed by –

- Client
- End users

Need of User Acceptance Testing:

Need of User Acceptance Testing arises once software has undergone Unit, Integration and System testing because developers might have built software based on requirements document by their own understanding and further required changes during development may not be effectively communicated to them, so for testing whether the final product is accepted by client/end-user, user acceptance testing is needed.


1
• Developers have included features on their "own" understanding
2
• Requirements changes "not communicated" effectively to the developers

- Developers code software based on requirements document which is their “own” understanding of the requirements and **may not actually be what the client needs from the software**.

- Requirements changes during the course of the project may not be communicated effectively to the developers.

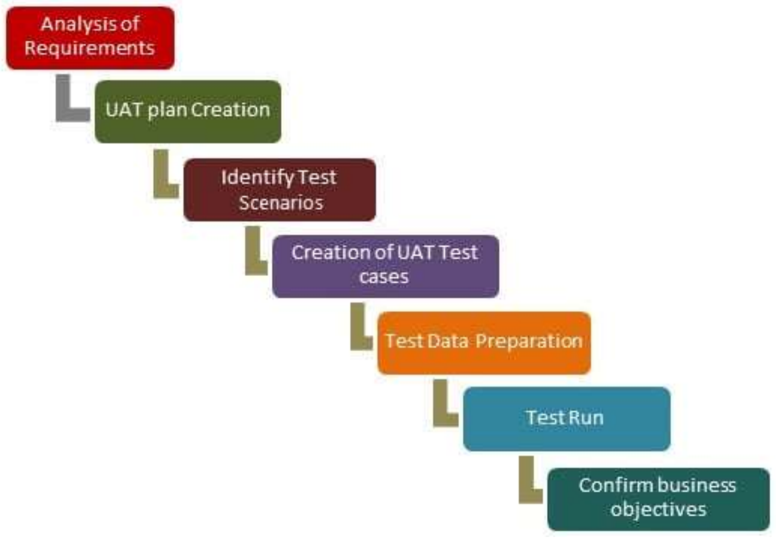
Prerequisites of User Acceptance Testing:

Following are the entry criteria for User Acceptance Testing:

- Business Requirements must be available.
- Application Code should be fully developed
- Unit Testing, Integration Testing & System Testing should be completed
- No Showstoppers, High, Medium defects in System Integration Test Phase –
- Only Cosmetic error is acceptable before UAT
- Regression Testing should be completed with no major defects
- All the reported defects should be fixed and tested before UAT
- Traceability matrix for all testing should be completed
- UAT Environment must be ready
- Sign off mail or communication from System Testing Team that the system is ready for UAT execution

How to execute UAT Tests:

UAT is done by the intended users of the system or software. This type of Software Testing usually happens at the client location which is known as Beta Testing. Once Entry criteria for UAT are satisfied, following are the tasks need to be performed by the testers:



UAT Process

- Analysis of Business Requirements
- Creation of UAT test plan
- Identify Test Scenarios
- Create UAT Test Cases

- Preparation of Test Data(Production like Data)
- Run the Test cases
- Record the Results
- Confirm business objectives

Step 1) Analysis of Business Requirements

One of the most important activities in the UAT is to identify and develop test scenarios. These test scenarios are derived from the following documents:

- Project Charter
- Business Use Cases
- Process Flow Diagrams
- Business Requirements Document(BRD)
- System Requirements Specification(SRS)

Step 2) Creation of UAT Plan

The UAT test plan outlines the strategy that will be used to verify and ensure an application meets its business requirements. It documents entry and **exit criteria for UAT, Test scenarios and test cases approach and timelines of testing**.

Step 3) Identify Test Scenarios and Test Cases

Identify the test scenarios with respect to high-level business process and create test cases with clear test steps. Test Cases should sufficiently cover most of the UAT scenarios. Business Use cases are input for creating the test cases.

Step 4) Preparation of Test Data

It is best advised to use live data for UAT. Data should be scrambled for privacy and security reasons. Tester should be familiar with the database flow.

Step 5) Run and record the results

Execute test cases and report bugs if any. Re-test bugs once fixed. Test Management tools can be used for execution.

Step 6) Confirm Business Objectives met

Business Analysts or UAT Testers needs to send a sign off mail after the UAT testing. After sign-off, the product is good to go for production. Deliverables for UAT testing are Test Plan, UAT Scenarios and Test Cases, Test Results and Defect Log

Example Guidelines for UAT

- Most of the times in regular software developing scenarios, UAT is carried out in the QA environment. If there is no staging or UAT environment
- UAT is classified into Beta and Alpha testing but it is not so important when software is developed for a service based industry
- UAT makes more sense when the customer is involved to a greater extent

Benefits of UAT:

Here's an overview of some benefits of conducting a UAT:

Improves product quality: User acceptance testing helps the product development team discover and resolve issues that affect user experience. They may also receive helpful feedback from users for optimizing the product's performance.

Reduces cost: It's much cheaper and easier to resolve issues and replace features when a product is still in development. Businesses can also avoid releasing products with major functionality issues, saving costs of recalling such products from the market.

Promotes brand image: Consistent and thorough user-acceptance tests ensure brands only put out top-quality products. This can improve brand image and customer loyalty.

Ensures compliance: User acceptance tests help to ensure that all aspects of a product abide by relevant laws and regulations. This ensures companies don't incur any legal liability.

Disadvantages of UAT often include:

- **It’s time-consuming**:the process takes longer than traditional functional testing because the users are doing their regular jobs.
- **UAT is expensive**:the project team must pay for the testing services (if not done internally).
- **Difficulting getting business buy-in**:skeptics may argue that all the time spent on User Acceptance Testing (UAT) is a complete waste since the software development has already occurred.
- **Privacy issues bubble up**:the user might have to disclose some sensitive information they would rather not use for testing purposes.
- It’s hard to use if the user must work in different environments or operating systems:if they require a specific setup to use the software, there may not be enough time in their schedule for testing.

<u>Jenkins:</u>

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

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

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

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

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

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

- Perform a software build using a build system like Gradle or Maven Apache
- Execute a shell script
- Archive a build result
- Running software tests

Features Of Jenkins:

Jenkins is more functionality-driven rather than UI-driven hence, there is a learning curve involved in getting to know what is Jenkins. Here are the powerful developer-centric features offered by Jenkins:

1. Easy Installation & Configuration

Jenkins is a self-contained Java program that is agnostic of the platform on which it is installed. It is available for almost all the popular operating systems such as Windows, different flavors of Unix, and Mac OS.

It is available as a normal installer, as well as a .war file. Once installed, it is easy to configure using its web interface.

2. Open-Source

As it is open-source, it is free for use. There is a strong involvement of the community which makes it a powerful CI/CD tool. You can take support from the Jenkins community, whether it is for extensibility, support, documentation, or any other feature related to Jenkins.

3. Thriving Plugin Ecosystem

The backbone of Jenkins is the community and the community members have been instrumental in the development (and testing) of close to 1500+ plugins available in the Update Center.

4. Easy Distribution

Jenkins is designed in such a manner that makes it relatively easy to distribute work across multiple machines and platforms for the accelerated build, testing, and deployment.

Architecture Of Jenkins:

Jenkins distributed architecture was created to meet the above requirements.
Furthermore, **Jenkins manages distributed builds using a Master-Slave architecture**. The TCP/IP protocol is used to communicate between the Master and the Slave in this design.

2.1. Jenkins Master

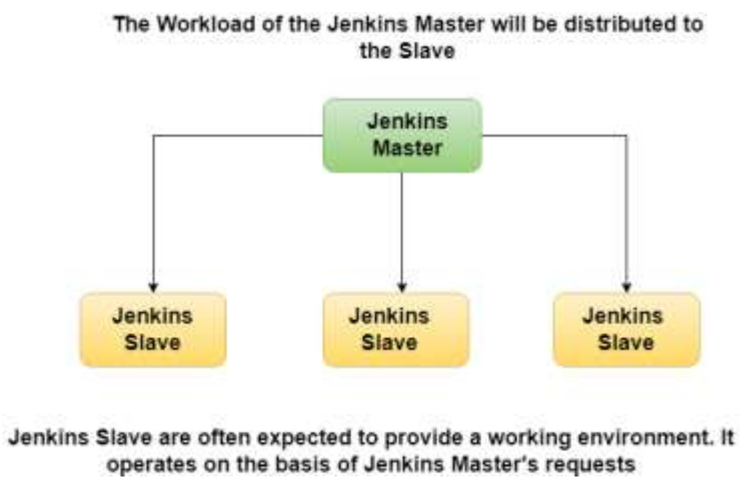
The Jenkins master is in charge of scheduling the jobs, assigning slaves, and sending builds to slaves to execute the jobs. It'll also keep track of the slave state (offline or online) and retrieve the build result responses from slaves and display them on the console output.

2.2. Jenkins Slave

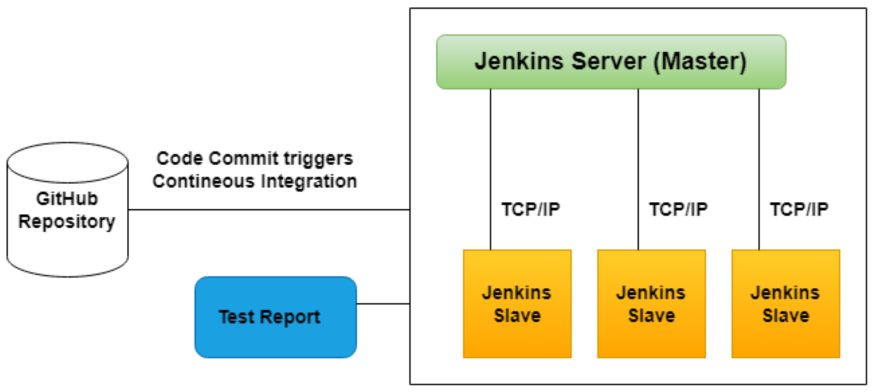
It runs on the remote server. **The Jenkins server follows the requests of the Jenkins master and is compatible with all operating systems**. Building jobs dispatched by the master are executed by the slave. Also, the project can be configured to choose a specific slave machine.

2.3. Distributed Master-Slave Architecture

Let's have a look at the Jenkins architecture using an example. One master and three Jenkins slaves are depicted in the diagram below:


The Workload of the Jenkins Master will be distributed to the Slave
Jenkins Master
Jenkins Slave
Jenkins Slave
Jenkins Slave
Jenkins Slave are often expected to provide a working environment. It operates on the basis of Jenkins Master's requests

Let's look at how we may utilize Jenkins for testing in various systems, such as Ubuntu, Windows, or Mac:



Advantages of Jenkins:

- It is an open source tool.
- It is free of cost.
- It does not require additional installations or components. Means it is easy to install.
- Easily configurable.
- It supports 1000 or more plugins to ease your work. If a plugin does not exist, you can write the script for it and share with community.
- It is built in java and hence it is portable.
- It is platform independent. It is available for all platforms and different operating systems. Like OS X, Windows or Linux.
- Easy support, since it open source and widely used.

- Jenkins also supports cloud based architecture so that we can deploy Jenkins in cloud based platforms.

Disadvantages of Jenkins:

- Its interface is out dated and not user friendly compared to current user interface trends.
- Not easy to maintain it because it runs on a server and requires some skills as server administrator to monitor its activity.
- CI regularly breaks due to some small setting changes. CI will be paused and therefore requires some developer's team attention.

UNIT-4:-

MICROSERVICES IN DEVOPS ENVIRONMENT

Evolution of Microservices and DevOps:

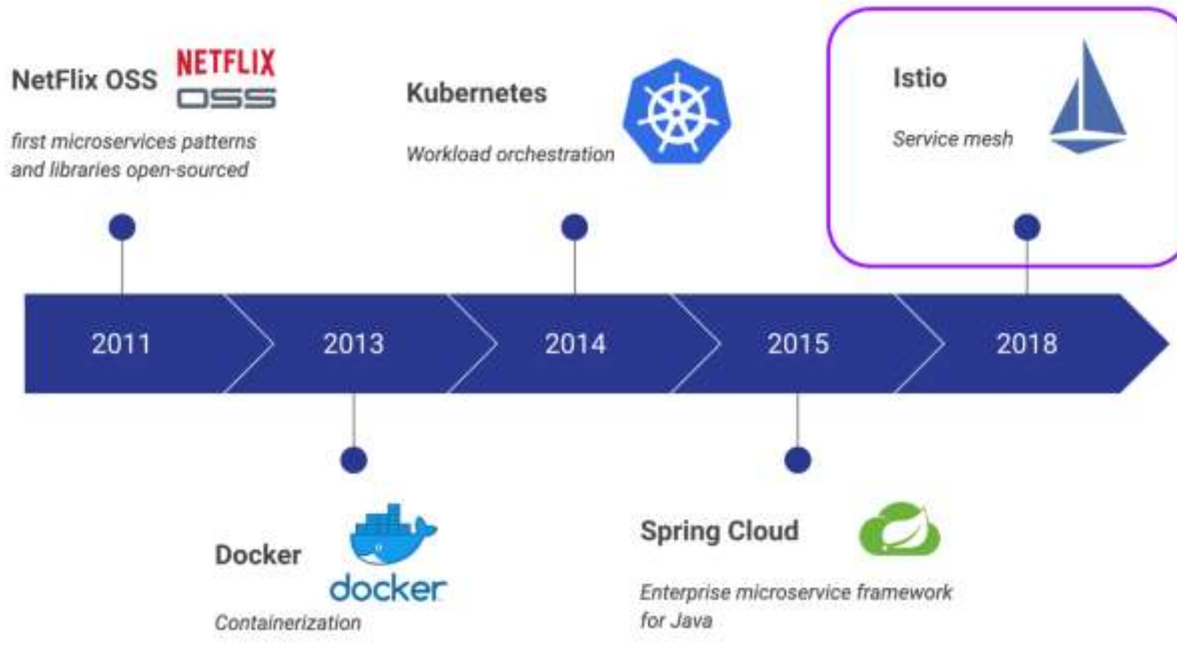
Evolution of Microservices:

Companies with large, monolithic applications are increasingly breaking these unwieldy apps into smaller, containerized microservices. Microservices are popular because they offer agility, speed, and flexibility, but they can be complex, which can be a hurdle for adoption.

In recent years evolving software development practices have fundamentally changed applications. These changes have impacted the requirements to the underlying infrastructure, tools, and processes to manage applications properly throughout the lifecycle.

Such microservices architectures provide agility to development teams. But, it also introduces **operational complexity**. Both Dev and Ops teams often struggle during the migration to microservices.

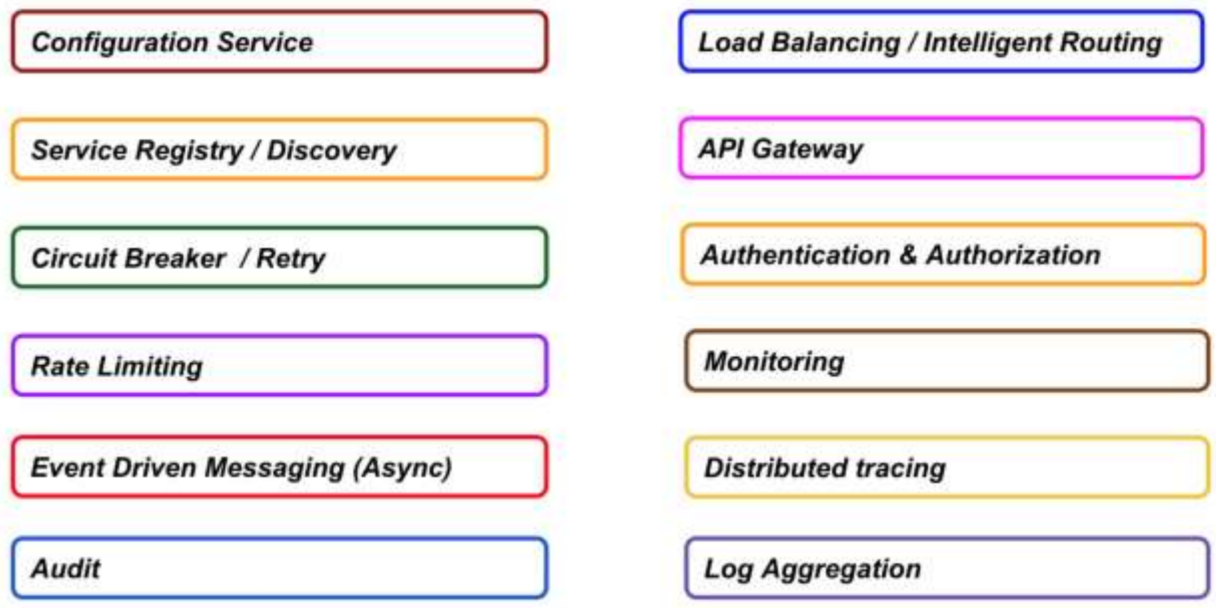
In a microservices world where a microservices architecture may seem like a distributed service mess, they must have these answers and more.



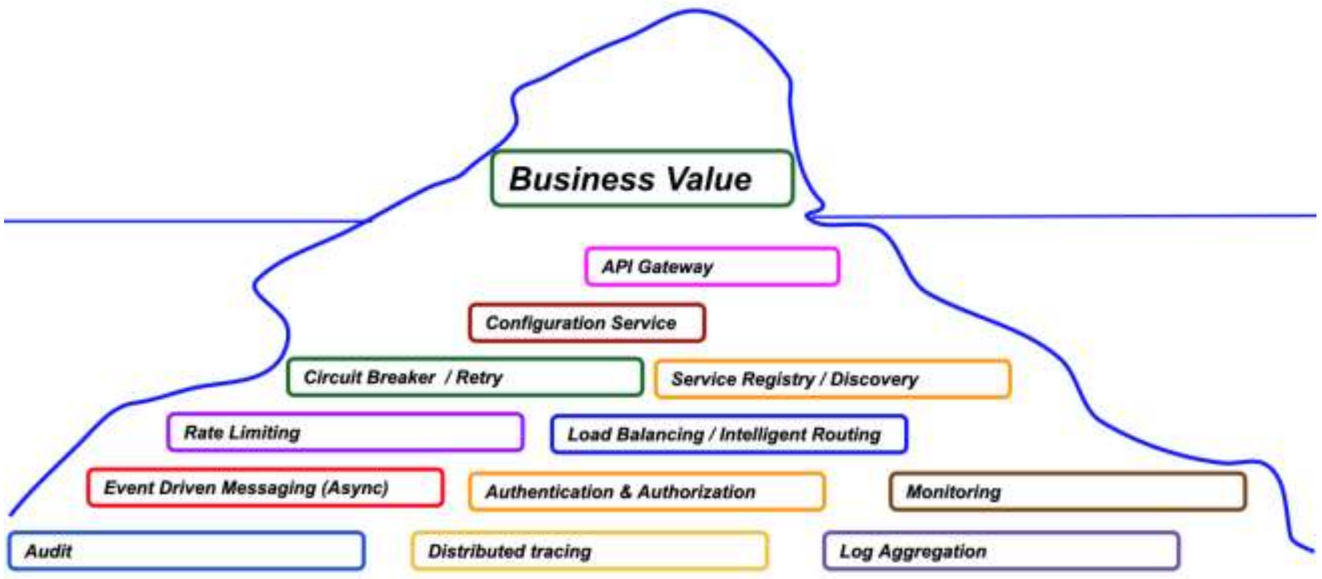
Microservices challenges:

Adopting a microservices oriented architecture brings new challenges:

- M to N communication.
- Distributed software interconnection and troubleshooting is hard.
- Containers should stay thin and platform agnostic.
- Upgrade of polyglot microservices is hard at scale.


Configuration Service
Load Balancing / Intelligent Routing
Service Registry / Discovery
API Gateway
Circuit Breaker / Retry
Authentication & Authorization
Rate Limiting
Monitoring
Event Driven Messaging (Async)
Distributed tracing
Audit
Log Aggregation

The only value visible from business stakeholders is the business service so we have a develop a bench of microservices concerns to deliver a new business value.



Evolution of DevOps:

DevOps is a software engineering culture and practice that aims to unify software development and operations. It is an approach to software development that emphasizes collaboration, communication, and integration between software developers and IT operations. DevOps has come a long way since its inception, and its evolution has had a significant impact on the way organizations develop and deliver software.

DevOps originated in the mid-2000s when organizations started to recognize the need for a more efficient and effective way of delivering software. The traditional approach to software development, where development and operations were separate departments, was proving to be too slow and cumbersome. The siloed nature of this approach led to poor communication and collaboration, resulting in long lead times, slow feedback cycles, and low-quality software.

Early Adoption and Growth of DevOps:

The early adopters of DevOps were organizations in the tech industry, particularly web-based companies. These organizations faced unique challenges in delivering software, such as frequent releases, short development cycles, and a large number of users. They saw the benefits of DevOps and were quick to adopt it to improve their software delivery process.

As the benefits of DevOps became more apparent, more and more organizations started to adopt it. DevOps became a key differentiator for organizations, and those that embraced it were able to deliver software faster, with higher quality, and at a lower cost. The adoption of DevOps was also driven by the rise of cloud computing, which provided the infrastructure and tools needed to support DevOps practices.

In the early 2010s, DevOps started to gain mainstream popularity, and its adoption accelerated. The rise of DevOps was also driven by the increasing importance of software in all aspects of business, from product development to customer engagement.

The year 2020, with its main feature being a global pandemic, has appeared to be a turning point for DevOps, one that will determine the trajectory of software development for years to come, beginning from the next, 2021.

This article explores how COVID-19, the shift to the cloud, and its implications for security are driving a new wave of demands for software development in the near future, from a DevOps perspective.

1. Cloud-Native Technology in Production

As a Forrester report notes, the cloud takes center stage in pandemic recovery efforts, enabling millions of workers to work from home, maintaining supply chains globally, and rapidly transforming business models. In 2021, it estimates that the global public cloud infrastructure market will grow by 35% to $120 billion, about $7 billion higher than its previous predictions, which changed after reviewing the revenue growth of the big cloud companies, AWS, Azure, Google Cloud, and Alibaba.

2. Integration of DevOps and Security

Security is defining the coming trends of DevOps significantly more than ever. The massive shift to the cloud occasioned by the pandemic means there must now be greater emphasis on cloud-based cybersecurity due to elevated security concerns. Amongst other lessons, the pandemic has shown that cloud cybersecurity for remote workers is tougher than expected. This has caused experts to suggest that the cyber pandemic might be next.

DevOps is tied strongly to security; teams can build more secure applications if they integrate security earlier in the application development. With continuous testing and delivery, DevOps ensures that nothing is left to chance.

3. Testing Shifts Further Left
Not only is security shifting left; testing, as a whole, will shift much further left in the coming years. Fewer bugs at the early stages mean that developers can detect and fix them easier. Vulnerabilities discovered at the production stage can cost up to around $7600 to fix whereas the same vulnerabilities discovered at the early stages of development could be fixed for only $80-$85 then. Hence, the need for shift-left testing. And developers are responding appropriately as evidenced by the rising trend.

Benefits of combining DevOps and Microservices:

Microservices enable continuous delivery and deployment, both of which accelerate the delivery of software. Most organizations are moving toward or implementing continuous delivery, which means releasing software in a matter of months versus years, weeks versus months, days versus weeks, or hours versus days.

Continuous deployment can result in hundreds or thousands of releases per day. It tends to be implemented by cloud natives such as Amazon, Facebook, and Google.

In fact, microservices emerged from the DevOps practices of cloud native companies that wanted to deliver software faster and improve product quality. Some of the benefits of microservices and DevOps are:

- More frequent software releases
- Better quality code
- Software reusability
- Scalability
- Resilience
- Higher levels of automation
- Flexibility (the ability to consume new data sources, frameworks, libraries and other resources)

- Freedom (different teams can use their favorite tools and languages)
- Cloud-based architecture
- Higher levels of DevOps productivity

How The Microservices-DevOps Combination Will Change In The Years Ahead
Microservices have already made DevOps teams more productive, as explained above. Moving forward, the combination of the two will result in even higher levels of efficiency. In addition, end users will experience less app-related friction:

- Cloud-native development will complement cloud-native deployment. While it's efficient to be able to run a microservice in any cloud, software generally is moving to the cloud. Where better to develop a cloud application than in the cloud itself? For one thing, it's easier to catch application errors prior to release.
- Machine learning and AI are making their way into DevOps and microservices tooling. One of the benefits of intelligent tooling is getting more in-depth insight into DevOps pipelines, code, configuration changes, performance, etc.
- More DevOps teams will embrace microservices to make enterprise applications more valuable, scalable, resilient and adaptable.
- Microservices help enable the "post-app" era in which users spend less time opening individual apps and more time outside the apps. For example, instead of opening a weather app, one can ask a virtual assistant such as Alexa, "What's the weather forecast today?" On a phone, a weather forecast could be pushed as a notification. The possibilities are virtually endless.

Working of DevOps and Microservices in Cloud environment:

DevOps is a software development approach that combines cultural principles, tools, and practices to increase the speed and efficiency of an organization's application delivery pipeline. It allows development and operations (DevOps) teams to deliver software and services quickly, enabling frequent updates and supporting the rapid evolution of products.

Cloud computing is a powerful technology that helps organizations implement DevOps strategies, enabling the crucial cultural and technological transformation needed to compete in the modern software marketplace.

There are three important ways DevOps and cloud work together:

1. **DevOps leverages the cloud**—DevOps organizations manage and automate infrastructure using cloud computing technology, enabling agile work processes.

2. **CloudSecOps** (cloud security operations)—an organizational pattern that moves processes to the cloud while tightly integrating security into the entire development lifecycle. It's like DevSecOps, only in the cloud.
3. **DevOps as a Service**—delivering integrated continuous integration and continuous delivery (CI/CD) pipelines via the cloud, in a software as a service (SaaS) model, to make DevOps easier to adopt and manage in an organization. We'll describe this model and also introduce DevOps as a Service offerings from Amazon, Microsoft Azure, and Google Cloud.

1. How DevOps Leverages the Cloud

DevOps processes can be very agile when implemented correctly, but they can easily grind to a halt when facing the limitations of an on-premise environment. For example, if an organization needs to procure and install new hardware in order to start a new software project or scale up a production application, it causes needless delays and complexity for DevOps teams.

Cloud infrastructure offers an important boost for DevOps and facilitates scalability. The cloud minimizes latency and enables centralized management via a unified platform for deploying, testing, integrating, and releasing applications.

The high level of automation provided by cloud-based DevOps services helps minimize human error and streamlines repeatable tasks. Cloud services and tools let developers automate specific tasks to use their time more efficiently.

2. What Is CloudSecOps (Cloud, Security, and Operations)?

SecOps is a merging of security and IT operations in a unified process. SecOps involves a team that combines skilled software engineers and security analysts that can assess and monitor risk and protect corporate assets. The SecOps team typically operates from an organization's security operations center (SOC).

Cloud security operations (CloudSecOps) is an evolution of the SecOps function that aims to identify, respond to, and recover systems from attacks targeting an organization's cloud assets. Security operations must reactively respond to attacks that the security tools detect, while proactively seeking out other attacks that ordinary detection methods may have missed.

CloudSecOps teams have several roles and functions:

- **Incident management**—identifies security incidents, responds to them, and coordinates the response with communication, legal, and other teams. In a cloud environment, incident management moves faster and involves many more moving parts than in an on-premise data center.
- **Prioritizing events**—this requires calculating risk scores for cloud systems, accounts, and devices, and identifying the sensitivity of cloud applications and data.
- **Using security technology**—traditional SOC tools include security information and event management (SIEM) solutions and other reactive systems. SOC teams are shifting from static log analysis using conventional tools to advanced analytics driven by new solutions such as extended detection and response (XDR). These solutions leverage behavioral analysis, machine learning, and threat intelligence capabilities to identify and respond to abnormal behavior.

- **Threat hunting**—a proactive effort to discover advanced security threats, usually triggered by a hypothetical threat scenario. Threat hunting involves tools that filter out the noise from security monitoring solutions, enabling advanced data investigation.
- **Metrics and objectives**—the role of a SecOps team requires keeping track of key performance indicators like mean time to detect, acknowledge, and remediate (MTTD, MTTA, and MTTR, respectively).

3. What Is DevOps as a Service?

DevOps as a Service is a set of cloud-based tools that enable collaboration between an organization's development and operations teams. The DevOps as a Service provider provides a toolset that covers all relevant aspects of the DevOps process and provides them as a unified platform.

DevOps as a Service is the opposite of a "best of breed" toolchain, where teams select the tools they like best for each purpose. It can make DevOps easier to implement for organizations new to agile processes because it does not require learning and integrating multiple-point solutions.

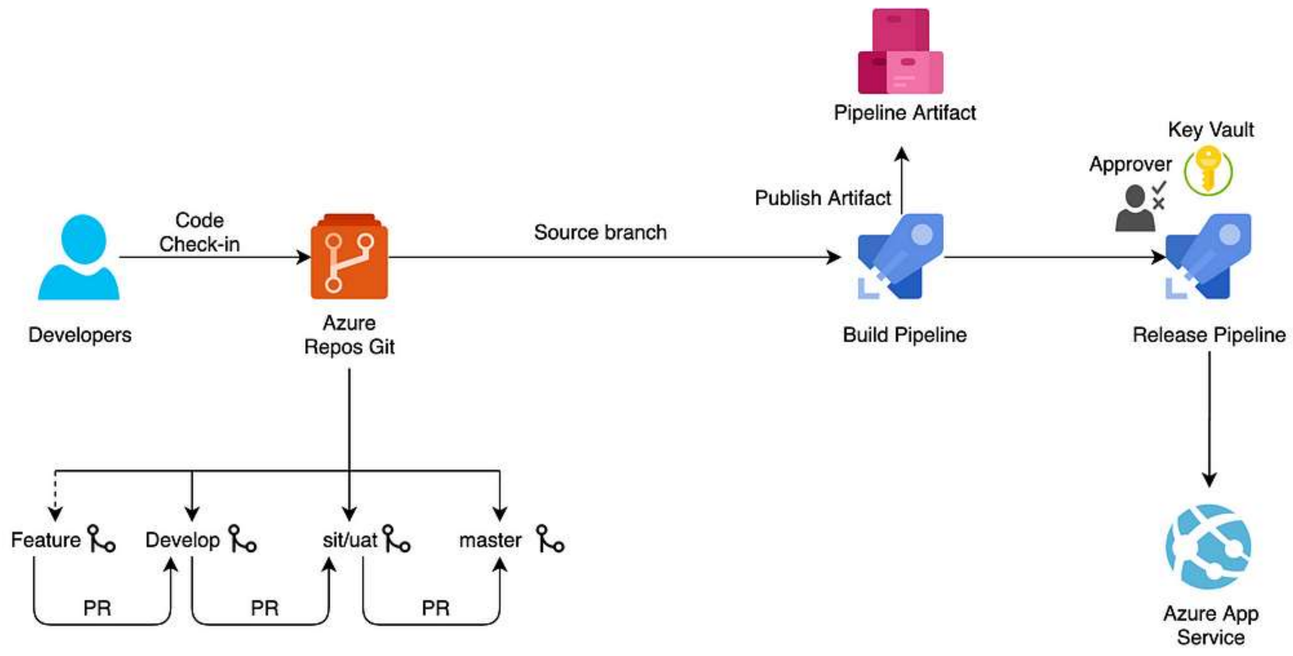
A DevOps as a Service platform enables tracking and management of every action taken in the software delivery process. It enables organizations to set up continuous integration / continuous delivery (CI/CD) systems and pipelines to increase development velocity and provide continuous feedback to developers.

For example, a DevOps as a Service solution provides access to source code management (SCM), build servers, deployment management, and application performance management (APM) in one interface, with centralized auditing and reporting.

DevOps Pipeline representation for a NodeJS based Microservices:

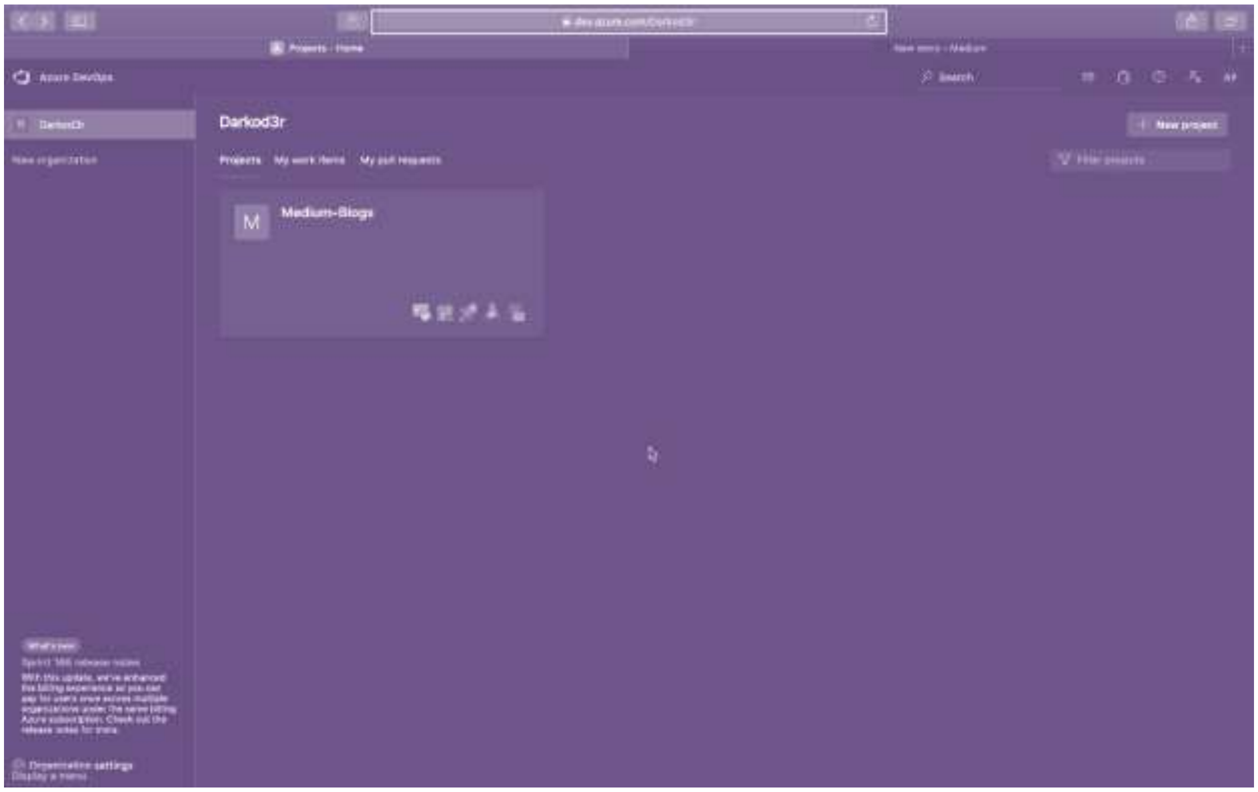
Use Azure DevOps pipelines to build and test Node.js apps, and then deploy or publish to azure app service. The following are the steps to be performed for a complete CI/CD workflow.

1. Develop and commit your code in the development branch.
2. Push code from the development branch to → test branch → master branch.
3. Deploy your code in different environments; Dev →Test →Prod using CI/CD pipelines in Azure DevOps.



Create a Build Pipeline

- Go to dev.azure.com/{organisation-name} → Select Project →Pipelines

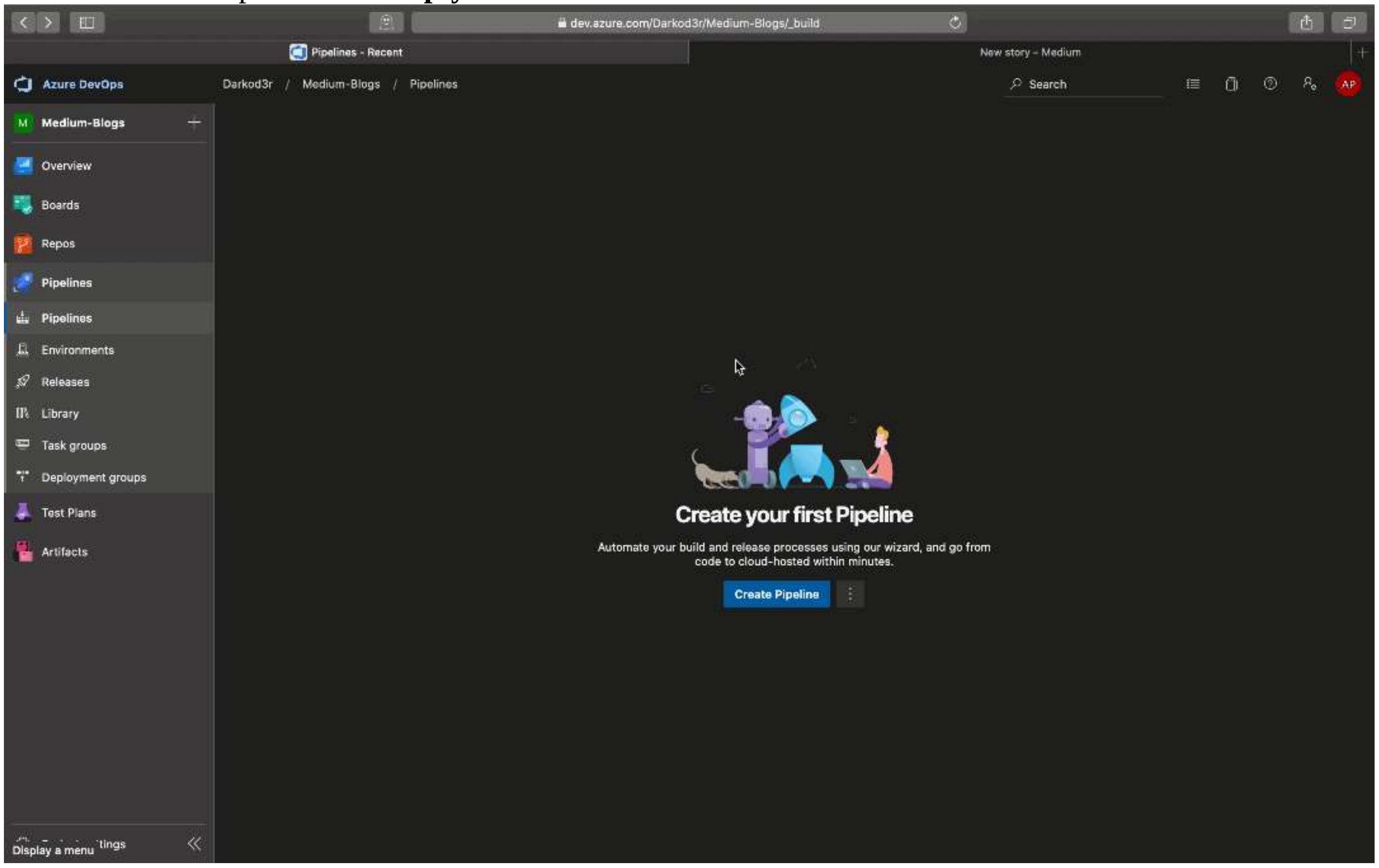


1.1 Go to project →Pipelines

- Create New Pipeline → Use Azure Git Repos (YAML) for creating a pipeline as code or use the **classic editor** for creating a pipeline from the visual designer. For this tutorial, we will be using the classic editor.

- Select a source repo → Select Project Name → Select Repository → Select Branch Name → Click on Continue.

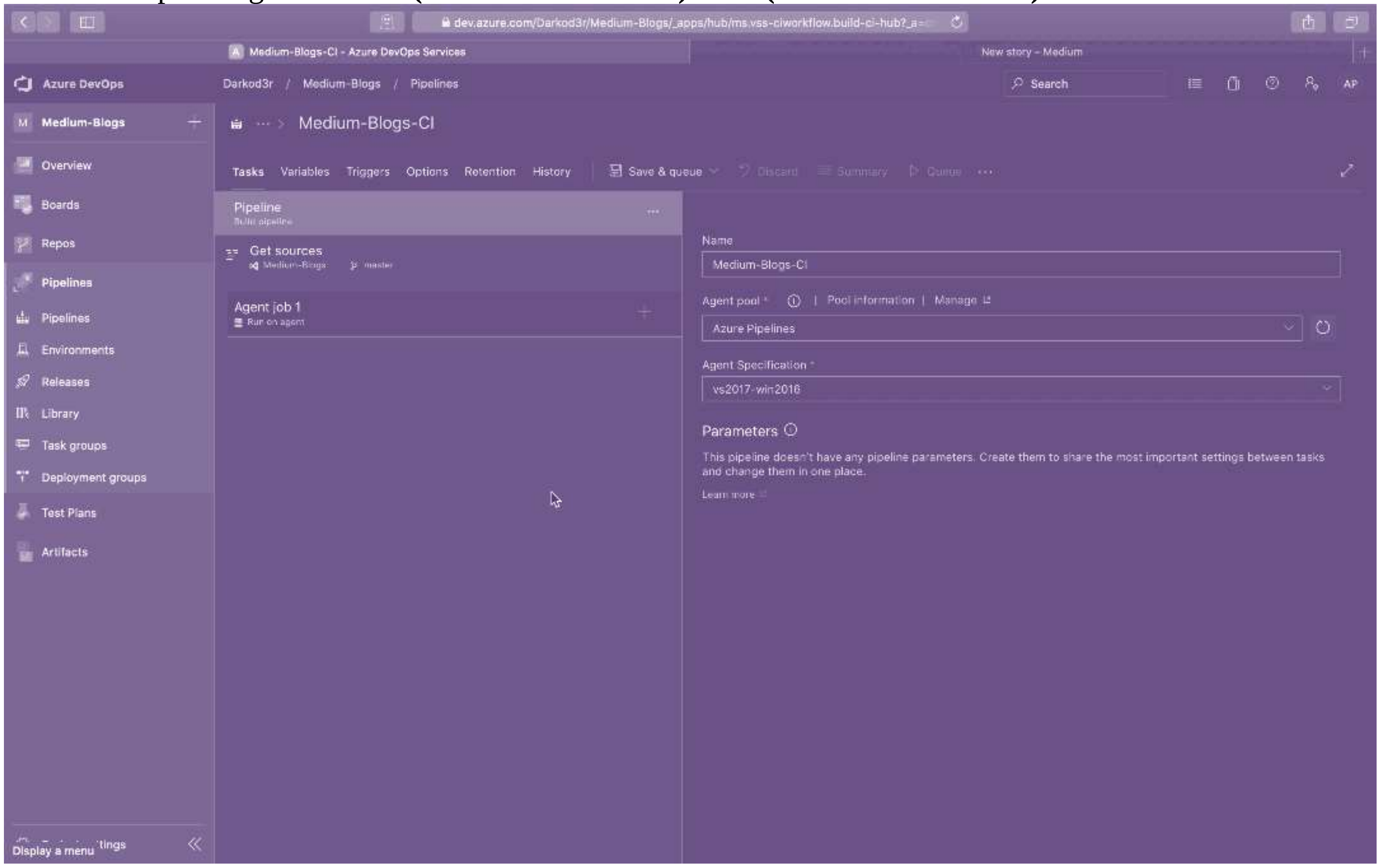
- Select the template as an **Empty Job.**


dev.azure.com/Darkod3r/Medium-Blogs/_build
Pipelines - Recent
New story – Medium
Azure DevOps
Darkod3r / Medium-Blogs / Pipelines
Search
Medium-Blogs
Overview
Boards
Repos
Pipelines
Pipelines
Environments
Releases
Library
Task groups
Deployment groups
Test Plans
Artifacts
Create your first Pipeline
Automate your build and release processes using our wizard, and go from code to cloud-hosted within minutes.
Create Pipeline

1.2 Create a new pipeline using the classic editor

- Change the name of the build pipeline as per the naming convention of your organization→ Select **Agent Pool** as per requirement { Hosted vs2017-win2016 for Windows Environment & Hosted Ubuntu 18.04 for Linux based environment }.

- Think of these agents as Virtual Machines with different OS flavors.
- Prefer to use Microsoft Hosted Agents instead of Self Hosted Agents unless you know what you are doing
- Select Tag Sources → on success to create git tags whenever your build gets succeeded. You can keep the tag format as **$(build.buildNumber) or v$(build.buildNumber).**



Name: Medium-Blogs-CI

Agent pool *: Azure Pipelines | Pool information | Manage

Agent Specification *: vs2017-win2016

Parameters

This pipeline doesn't have any pipeline parameters. Create them to share the most important settings between tasks and change them in one place.

Learn more

1.3 Select Agent Pool

- Click on Add Tasks(+) → search for task →Add Task
- You can add multiple tasks.
- If a task is not available in your organization, you can install it from Marketplace.

Scope for Improvements:

1. IaC (Infrastructure as Code)- This article is for beginners. If you are already familiar with CI/CD pipelines then multistage YAML pipeline is the way to go.

2. Increasing the Build pipeline performance by using npm-cache □□
3. Use of **static code analysis tools** like SonarQube in the build pipeline.
4. Build Validation of Pull Requests.

UNIT-5:-

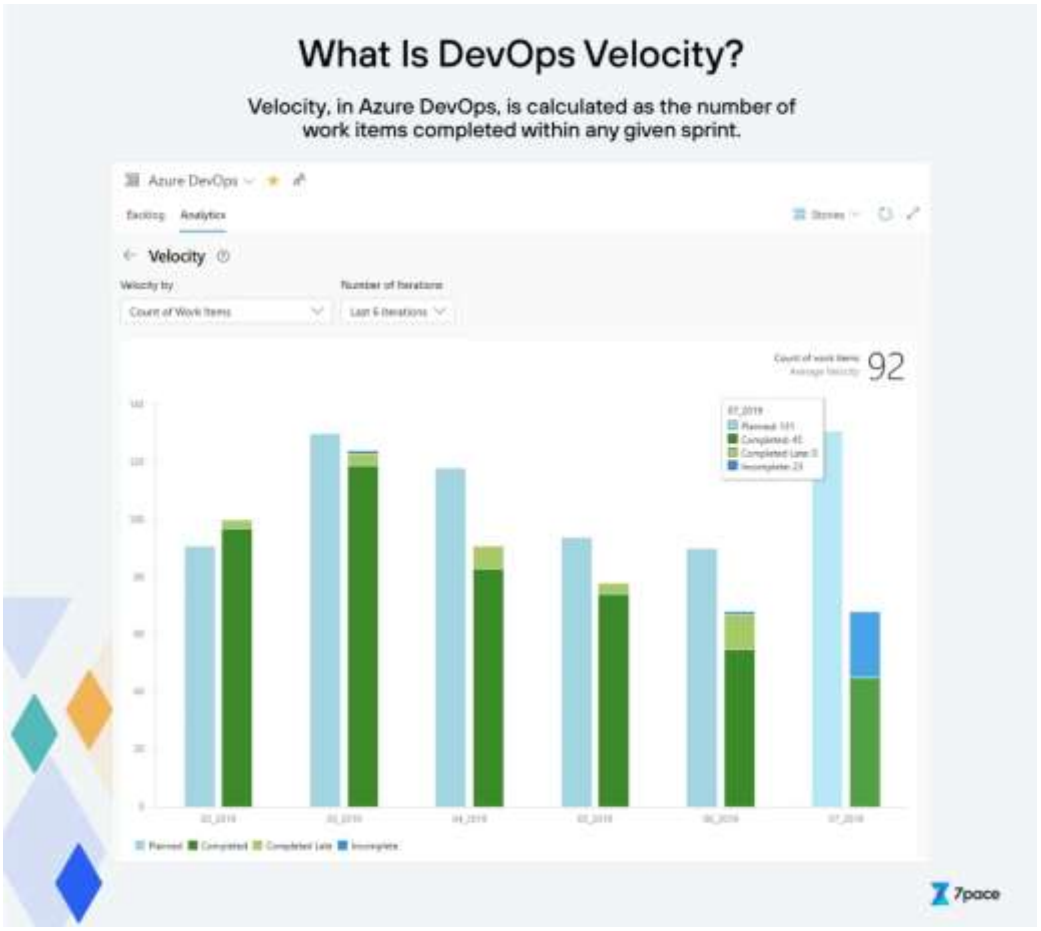
VELOCITY AND CONTINUOUS DELIVERY

Velocity:

Velocity, in the agile methodology, is defined as the rate of progress in software development from one sprint to the next. That is, it's the amount of work your team manages to get done within the current sprint, before moving on to the next sprint

Measuring velocity:
As you continue to keep track of your team's velocity from previous sprints, it'll help you establish a pattern that leads to better budget estimates and deadline forecasts. You'll have a better understanding of the time and effort required to complete a new project, which will empower you to make better decisions as a team lead.

You can access a report on your team's velocity both as an in-context chart in your product backlog, or as a universal widget on the team dashboard. Each report has its own advantages and offers different means to customize the way you estimate your team's velocity.



Getting Started With DevOps Velocity:

Before we dig deeper into Velocity in Azure DevOps, let's take a few common terms you should be familiar with:

- **Backlog:** Refers to the list of deliverables for any given software development project, sorted in order of priority.
- **Sprints:** time-boxed periods during which a development team tries to take on a specific task within the project, ranging from a week to one month.
- **Effort:** The number of work units, usually measured in person-hours, required to complete a specific task or project.
- **Stories:** Work items in your product backlog that each refer to a feature to be developed or a bug to be fixed. Also called user stories or story points.
- **Burndown:** It's the rate at which a development team works through the project backlog. That makes velocity the slope in the burndown chart.

Velocity, in Microsoft Azure DevOps, is calculated as the number of work items completed within any given sprint. Note that since velocity is based on the number of work items that have been actually completed, if you begin working on a story point but are unable to finish it, DevOps will automatically flag it as Incomplete.

Viewing the In-Context Report:

To view the in-context report, you must first access your project backlog. You can do this by visiting the Backlogs tab inside Azure Boards.

Velocity reports are available for both product and portfolio backlogs, all you have to do is visit your preferred backlog and choose *Analytics>Velocity>View full report*.

You can filter your velocity report based on the number of work items or the sum of estimates made to effort, story, or size. You can also select the number of iterations, between 1 to 15, that you wish to view this data for. Apart from the number of completed items per iteration, you'll also be able to view the number of items flagged as Planned, Incomplete, or Completed Late.

Configuring the Velocity Widget:

If you'd like to have a top-level view of your team's workflow and progress from sprint to sprint, you can also configure the velocity widget to appear on your main dashboard.

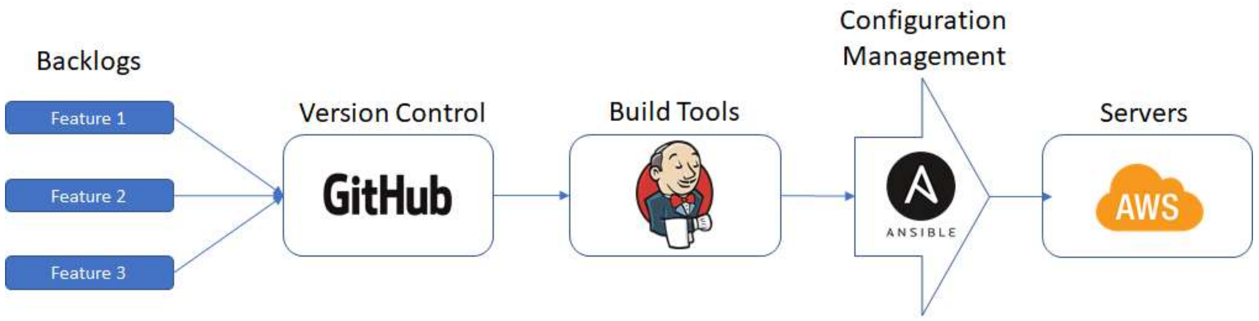
Once you're inside your admin panel in Azure DevOps, visit *Overview > Dashboards*. Then, click on the pencil-shaped edit icon to add a new widget to your dashboard. Choose the *Velocity widget* and click on *Add*.

To configure the Velocity widget, click on the *three dots (…)* on the upper right of the screen. This will open the configuration dialog, where you can add a title to your widget as well as specify what it is you want the widget to display.

Delivery Pipeline:

Delivery pipeline in software development life cycle is the process/workflow/strategy of driving the development process from product's story backlogs to deploying it to the live production system. This constructive workflow will ensure the conversion of backlogs -> features -> tasks -> build -> packages -> deployment. This entire life cycle needs a lot of collaboration and communication between teams and stakeholders. So the Delivery pipeline is creating a systematic flow in this cycle which will enable the communication between the team and logging mechanism for the reporting and auditing purpose.

Say we have a project of building e-commerce website. So, we are targeting couple of features to be delivered on the particular Sprint. Assume we have java based platform, GitHub as the version control system, Linux based machines for serving the website.



So, Delivery pipeline starts with assigning features to developers as tasks. Once Developers are done with the coding, Versions of their codes will be pushed to the version control (In our case GitHub), Once the code is merged with the respective branch (Environment specific), Build tool (In our case, Jenkins) should trigger the job to build the committed code & and get the Package out of it. Then Trigger the Configuration Management tool (In Our case Ansible) to deploy the package to the servers. So enabling tools like Jira, Slack and other tools will enable communication between the teams.

Stages of the Delivery Pipeline:

A continuous delivery pipeline consists of five main phases—build/develop, commit, test, stage, and deploy.

1.Build/Develop

A build/develop process performs the following:

1. **Pulls** source code from a public or private repository.
2. **Establishes** links to relevant modules, dependencies, and libraries.
3. **Builds** (compiles) all components into a binary artifact.

Depending on the programming language and the integrated development environment (IDE), the build process can include various tools. The IDE may offer build capabilities or require integration with a separate tool. Additional tools include scripts and a virtual machine (VM) or a Docker container.

2.Commit

The commit phase checks and sends the latest source code changes to the repository. Every check-in creates a new instance of the deployment pipeline. Once the first stage passes, a release candidate is created. The goal is to eliminate any builds unsuitable for production and quickly inform developers of broken applications.

Commit tasks typically run as a set of jobs, including:

- **Compile** the source code
- **Run** the relevant commit tests
- **Create** binaries for later phases
- **Perform** code analysis to verify health
- **Prepare** artifacts like test databases for later phases

These jobs run on a build grid, a facility provided by continuous integration (CI) servers. It helps ensure that the commit stage completes quickly, in less than five minutes, ideally, and no longer than ten minutes.

3.Test

During the test phase, the completed build undergoes comprehensive dynamic testing. It occurs after the source code has undergone static testing. Dynamic tests commonly include:

- **Unit or functional testing**—helps verify new features and functions work as intended.
- **Regression testing**—helps ensure new additions and changes do not break previously working features.

4.Stage

The staging phase involves extensive testing for all code changes to verify they work as intended, using a staging environment, a replica of the production (live) environment. It is the last phase before deploying changes to the live environment.

The staging environment mimics the real production setting, including hardware, software, configuration, architecture, and scale. You can deploy a staging environment as part of the release cycle and remove it after deployment in production.

5.Deploy

The deployment phase occurs after the build passes all testing and becomes a candidate for deployment in production. A continuous delivery pipeline sends the candidate to human teams for approval and deployment. A continuous deployment pipeline deploys the build automatically after it passes testing.

In most cases, developers do not deploy candidates fully as is. Instead, they employ precautions and live testing to roll back or curtail unexpected issues. Common deployment strategies include beta tests, blue/green tests, A/B tests, and other crossover periods.

Advantages:

- **More responsive development**. Many organizations that strive to deliver and operate at real-time speed are often hampered by slower traditional app development. With CI/CD, developers can rapidly prioritize and address each organizational need. This also requires a close relationship with the business side to prioritize areas with the biggest potential business impact, rather than what developers may view with higher technical preference.
- **Better code quality**. Smaller changes involve less code, and code quality can be checked to a greater degree before it reaches a full testing environment. Developers can more easily identify and solve problems earlier in the DevOps process. Code quality is still important, however -- CI/CD does not mean you can cut corners in the interests of speed.
- **Shorter testing cycles**. Less complex and smaller volumes of code to check means there's less need to repeatedly test functionality throughout the CI/CD process. New functionality created by the development community gets out into the operational environment far faster than with full retro-testing of mass changes. Again, this does not mean you should take any testing shortcuts -- only that a well-run CI/CD approach involves fewer interactions between what is changed and what remains the same.

- **Easier monitoring of change in the operational environment**. If something goes wrong during a rollout, it is usually far easier to identify the root cause when it involves only one or two changes. However, the right tools must be in place to ensure that monitoring is sufficient and effective.
- **Easier rollback if required**. If a problem occurs that requires the platform to return to a previous known position, smaller changes mean less effort to carry out the rollback. Again, ensure the right tools are in place, especially an effective orchestration tool.

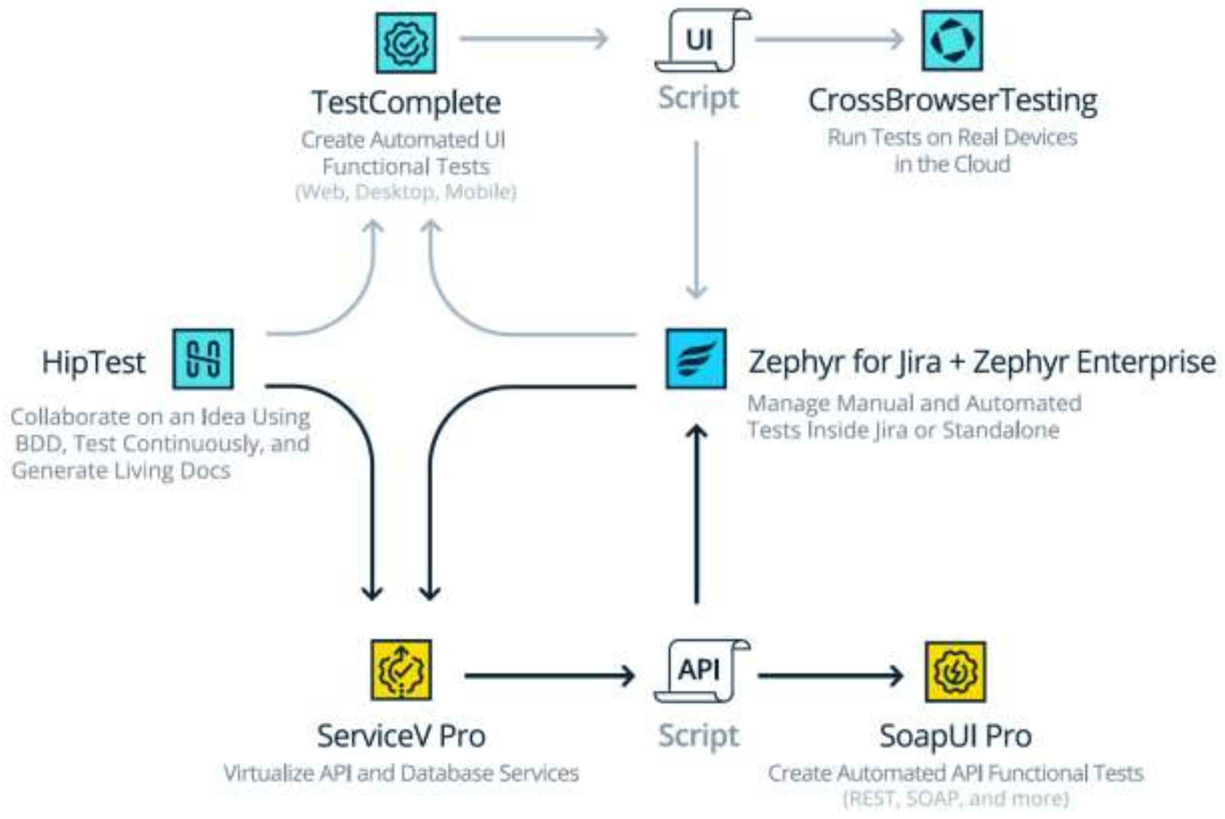
Disadvantages:

- **Changes create a domino effect with a microservices environment.** With CI/CD in a microservices environment, one small change can impact multiple different interactions. A change may work well for the intended microservices process, but it could cause problems within other service chains. Configuration management tracks all dependencies between different microservices, and orchestration can then help to ensure that any change does not impact other streams and that development teams can roll back changes if necessary.
- **Continuous change requires continuous monitoring and reporting.** CI/CD changes by their nature impact the platform to which they are rolled out. Real-time monitoring and reporting are necessary to understand and quickly address any problems. If a change misbehaves, you need to know immediately -- before problems cascade across other services and user complaints swamp the help desk.
- **Resource management must be responsive.** Without deep testing beforehand, developers and testers may not anticipate any resource and performance hits of a CI/CD change until it is rolled out.

Test stack:

A stack test measures the amount of a specific pollutant or pollutants being emitted through regulated stacks at facilities subject to the requirements of the Clean Air Act (CAA). Although stack testing is an important tool used to determine a facility's on-going compliance with emission limits established pursuant to the CAA, it is most valuable in determining whether a facility has the ability to comply with the requirements of the CAA in the first instance. However, this tool has not been consistently applied or utilized across the country by the U.S. Environmental Protection Agency (EPA), or delegated state/local agencies.

The OIG concluded that this lack of guidance and oversight had an adverse effect on the use of stack testing as a tool in determining compliance. As a result of their findings, OIG recommended that EPA develop national guidance that addresses issues such as: - recommended testing frequencies; - discrepancies in test procedures; and - inconsistent reporting of tests and results.


TestComplete
Create Automated UI Functional Tests
(Web, Desktop, Mobile)
UI
Script
CrossBrowserTesting
Run Tests on Real Devices in the Cloud
HipTest
Collaborate on an Idea Using BDD, Test Continuously, and Generate Living Docs
Zephyr for Jira + Zephyr Enterprise
Manage Manual and Automated Tests Inside Jira or Standalone
ServiceV Pro
Virtualize API and Database Services
API
Script
SoapUI Pro
Create Automated API Functional Tests
(REST, SOAP, and more)

For the purposes of this policy, stack testing is defined as any standardized procedure of actions using calibrated tools to determine a rate or concentration in order to verify emissions from a source or the accuracy of a monitor or gauge. It does not include visible emission observations.

The Integrated Test Tool Stack:

The capability to work with an integrated product suite that spans all the main QA and test disciplines is here. The SmartBear Test Stack delivers the best tool suite in the market to deliver the best performance on the ground.

If you're looking to take your test process and agile test management practices to the next level then the SmartBear UI automation, API testing and Test Management suite of products is the fastest way to achieve this. These 3 complex domains are well served by TestComplete, SoapUI and Zephyr. A well matched suite of tools that will deliver results for even the most challenging QA and development environment.

GOALS OF STACK TESTING:

• Expand upon the requirements of CMS and the HPV Policy to fully address the concerns raised by the Inspector General in his report on this issue.

• Improve uniformity on how stack tests are conducted.

• Improve coordination among EPA and state and local agencies.

• Enhance EPA oversight of state/local programs to ensure that the tool of stack testing is being used properly and sufficiently carried out.

Advantages of Stack Testing:

- **Easy implementation:** Stack data structure is easy to implement using arrays or linked lists, and its operations are simple to understand and implement.
- **Efficient memory utilization**: Stack uses a contiguous block of memory, making it more efficient in memory utilization as compared to other data structures.
- **Fast access time:** Stack data structure provides fast access time for adding and removing elements as the elements are added and removed from the top of the stack.
- **Helps in function calls:** Stack data structure is used to store function calls and their states, which helps in the efficient implementation of recursive function calls.
- **Supports backtracking:** Stack data structure supports backtracking algorithms, which are used in problem-solving to explore all possible solutions by storing the previous states.
- **Used in Compiler Design:** Stack data structure is used in compiler design for parsing and syntax analysis of programming languages.
- **Enables undo/redo operations**: Stack data structure is used to enable undo and redo operations in various applications like text editors, graphic design tools, and software development environments.

Disadvantages of Stack Testing:

- **Limited capacity:** Stack data structure has a limited capacity as it can only hold a fixed number of elements. If the stack becomes full, adding new elements may result in stack overflow, leading to the loss of data.
- **No random access:** Stack data structure does not allow for random access to its elements, and it only allows for adding and removing elements from the top of the stack. To access an element in the middle of the stack, all the elements above it must be removed.
- **Memory management:** Stack data structure uses a contiguous block of memory, which can result in memory fragmentation if elements are added and removed frequently.
- **Not suitable for certain applications:** Stack data structure is not suitable for applications that require accessing elements in the middle of the stack, like searching or sorting algorithms.
- **Stack overflow and underflow**: Stack data structure can result in stack overflow if too many elements are pushed onto the stack, and it can result in stack underflow if too many elements are popped from the stack.

- **Recursive function calls limitations:** While stack data structure supports recursive function calls, too many recursive function calls can lead to stack overflow, resulting in the termination of the program.

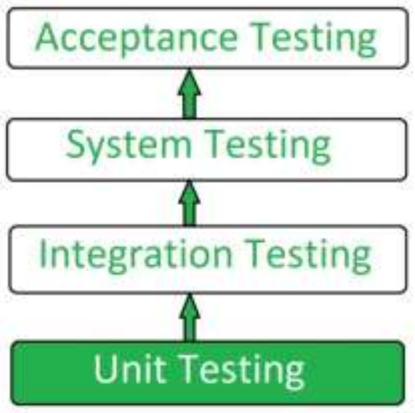
Small/Unit Test:

Unit Testing is a software testing technique by means of which individual units of software i.e. group of computer program modules, usage procedures, and operating procedures are tested to determine whether they are suitable for use or not. It is a testing method using which every independent module is tested to determine if there is an issue by the developer himself. It is correlated with the functional correctness of the independent modules. Unit Testing is defined as a type of software testing where individual components of a software are tested. Unit Testing of the software product is carried out during the development of an application. An individual component may be either an individual function or a procedure. Unit Testing is typically performed by the developer. In SDLC or V Model, Unit testing is the first level of testing done before integration testing. Unit testing is such a type of testing technique that is usually performed by developers. Although due to the reluctance of developers to test, quality assurance engineers also do unit testing.

Objective of Unit Testing:

The objective of Unit Testing is

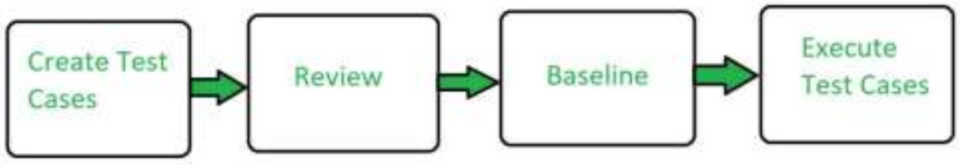
1. To isolate a section of code.
2. To verify the correctness of the code.
3. To test every function and procedure.
4. To fix bugs early in the development cycle and to save costs.
5. To help the developers to understand the code base and enable them to make changes quickly.
6. To help with code reuse.



Types of Unit Testing:

There are 2 types of Unit Testing: **Manual**, and **Automated**.

Workflow of Unit Testing:



Unit Testing Techniques:
There are 3 types of Unit Testing Techniques. They are

1. **Black Box Testing:** This testing technique is used in covering the unit tests for input, user interface, and output parts.
2. **White Box Testing:** This technique is used in testing the functional behavior of the system by giving the input and checking the functionality output including the internal design structure and code of the modules.
3. **Gray Box Testing:** This technique is used in executing the relevant test cases, test methods, test functions, and analyzing the code performance for the modules.

Unit Testing Tools:

Here are some commonly used Unit Testing tools:

1. Jtest
2. Junit
3. NUnit
4. EMMA
5. PHPUnit

Advantages of Unit Testing:

1. Unit Testing allows developers to learn what functionality is provided by a unit and how to use it to gain a basic understanding of the unit API.
2. Unit testing allows the programmer to refine code and make sure the module works properly.
3. Unit testing enables testing parts of the project without waiting for others to be completed.
4. Early Detection of Issues: Unit testing allows developers to detect and fix issues early in the development process, before they become larger and more difficult to fix.
5. **Improved Code Quality**: Unit testing helps to ensure that each unit of code works as intended and meets the requirements, improving the overall quality of the software.
6. **Increased Confidence:** Unit testing provides developers with confidence in their code, as they can validate that each unit of the software is functioning as expected.

Disadvantages of Unit Testing:

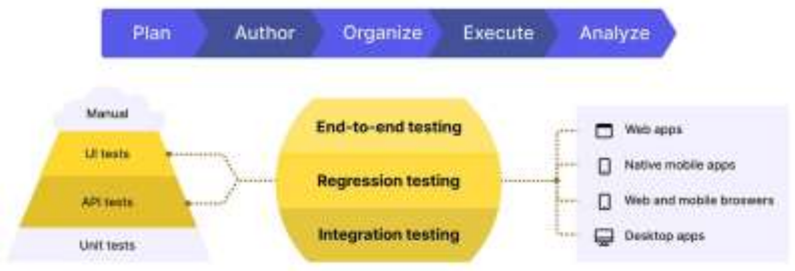
1. The process is time-consuming for writing the unit test cases.
2. Unit Testing will not cover all the errors in the module because there is a chance of having errors in the modules while doing integration testing.
3. Unit Testing is not efficient for checking the errors in the UI(User Interface) part of the module.
4. It requires more time for maintenance when the source code is changed frequently.
5. It cannot cover the non-functional testing parameters such as scalability, the performance of the system, etc.
6. **Time and Effor**t: Unit testing requires a significant investment of time and effort to create and maintain the test cases, especially for complex systems.

<u>Medium /integration testing:</u>

Integration Testing is a type of software testing in which the units, modules, and components of a software are integrated and tested as a cohesive unit.Essentially integration testing is a higher level than unit testing. With unit testing, we only validate if software components function well as a single unit. With integration testing, we want to see HOW those units work together and find out if there are code conflicts between them.

Integration Testing plays a crucial role in modern software testing as the development trend is gradually shifting towards cloud infrastructure, which calls for a higher volume of API testing to see if third-party integrations and APIs are interacting properly with the application itself.

Integration testing occurs after unit testing, and before system testing. The modules that have passed unit testing will be grouped into aggregates.



Why is integration testing important:

Software is often built from many individual software components or modules. These modules may pass unit testing and work perfectly fine individually, yet collapse when put together, for various reasons:

- **Inconsistent code logic:** They are coded by different programmers whose logic and approach to development differ from each other, so when integrated, the modules cause functional or usability issues. Integration testing ensures that the code behind these components is aligned, resulting in a functioning application.
- **Shifting requirements:** Clients change their requirements frequently. Modifying the code of module to adapt to new requirements sometimes means changing its code logic, which affects the entire application. In case that unit testing can't be performed due to time constraints, integration testing will be used to uncover the missing defects.
- **Erroneous data:** Data can change when transferred across developed modules. If not properly formatted when transferring, the data can't be read and processed, resulting in bugs. Integration testing is required to pinpoint where the issue lies for troubleshooting.
- **Third-party services and API integrations:** Since data can change when transferred, API and third-party services may receive false input and generate false responses. Integration testing ensures that these integrations can communicate well with each other.
- **Inadequate exception handling**: Developers usually account for exceptions in their code, but sometimes they can't fully see all of the exception scenarios until the modules are pieced together. Integration testing allows them to recognize those missing exception scenarios and make revisions.

- **External hardware interfaces:** Bugs may arise when there is software — hardware incompatibility, which can easily be found with proper integration testing.

Applications of integration testing:

1. **Identify the components:** Identify the individual components of your application that need to be integrated. This could include the frontend, backend, database, and any third-party services.
2. **Create a test plan:** Develop a test plan that outlines the scenarios and test cases that need to be executed to validate the integration points between the different components. This could include testing data flow, communication protocols, and error handling.
3. **Set up test environment:** Set up a test environment that mirrors the production environment as closely as possible. This will help ensure that the results of your integration tests are accurate and reliable.
4. **Execute the tests:** Execute the tests outlined in your test plan, starting with the most critical and complex scenarios. Be sure to log any defects or issues that you encounter during testing.
5. **Analyze the results:** Analyze the results of your integration tests to identify any defects or issues that need to be addressed. This may involve working with developers to fix bugs or make changes to the application architecture.
6. **Repeat testing:** Once defects have been fixed, repeat the integration testing process to ensure that the changes have been successful and that the application still works as expected.

Types of integration testing:

There are several strategies to use when performing integration testing, each of which has its own advantages and disadvantages, with the 2 most common approaches being:

1.Big Bang Approach: an approach in which all modules are integrated and tested at once, as a singular entity. The integration process is not carried out until all components have been successfully built and unit tested.

- **Advantage:** Suits smaller system
- **Disadvantage:** costly and time-consuming as testers have to wait until all modules have been developed to start the testing process

2.Incremental Approach: opposite to Big Bang approach, the Incremental approach involves strategically selecting 2 or more modules with closely related logic to integrate and test. This process is repeated until all software modules have been integrated and tested. The advantage of this is that we can test the application at an early stage.

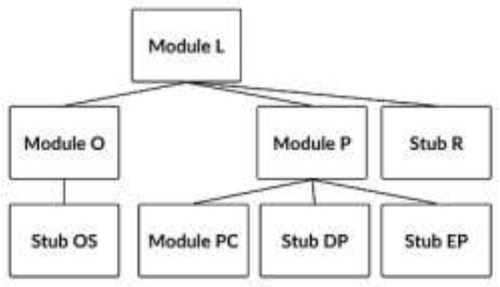
Incremental Approach is further categorized into 2 different methods:

1. Top-down approach

Testers first integrate and test the modules from the top layer, then sequentially integrate and test the lower-level modules until the entire application is tested. Underdeveloped or unavailable modules are substituted by "stubs".

Advantages: get to access early architecture defects easily

Disadvantages: hard to write different test environment conditions.



For example, the diagram above illustrates the modules required to complete an eCommerce purchase.

- **Module L:** Login action
- **Module O:** Order action
- **Stub OS:** Order Summary (yet to be developed)
- **Module P:** Payment Action
- **Module PC:** Payment By Cash option
- **Stub DP:** Debit Card Payment (yet to be developed)
- **Stub EP:** ePayment (yet to be developed)
- **Stub R:** Report (yet to be developed)

If we use the Top-down approach, we need 3 test cases:

- **Test Case 1**: Integrate and test Module L and Module O
- **Test Case 2**: Integrate and test Module L, O, and P
- **Test Case 3**: Integrate and test Module L, O, P, and

2. Bottom-up approach

Testers integrate and test the modules from the bottom layer first, then sequentially integrate and test higher and higher-level modules until the entire application has been tested. Underdeveloped or unavailable modules are replaced with “drivers”.

Advantages: can be used for applications with bottom-up design models.

Disadvantages: drivers are harder to code than stubs, and there is no working application until you have built the last module.

Both of these approaches inherit the advantages of Incremental Approach, which is its ease to accurately find interface errors with incremental testing.

3.Mixed Integration Testing – A mixed integration testing is also called sandwiched integration testing. A mixed integration testing follows a combination of top down and bottom-up testing approaches. In top-down approach, testing can start only after the top-level module have been coded and unit tested. In bottom-up approach, testing can start only after the bottom level modules are ready. This sandwich or mixed approach overcomes this shortcoming of the top-down and bottom-up approaches. It is also called the hybrid integration testing. also, stubs and drivers are used in mixed integration testing.

Advantages:

- Mixed approach is useful for very large projects having several sub projects.
- This Sandwich approach overcomes this shortcoming of the top-down and bottom-up approaches.
- Parallel test can be performed in top and bottom layer tests.

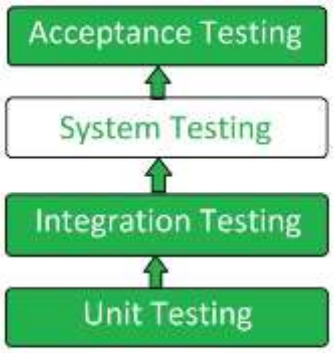
Disadvantages:

- For mixed integration testing, it requires very high cost because one part has a Top-down approach while another part has a bottom-up approach.
- This integration testing cannot be used for smaller systems with huge interdependence between different modules.

System testing:

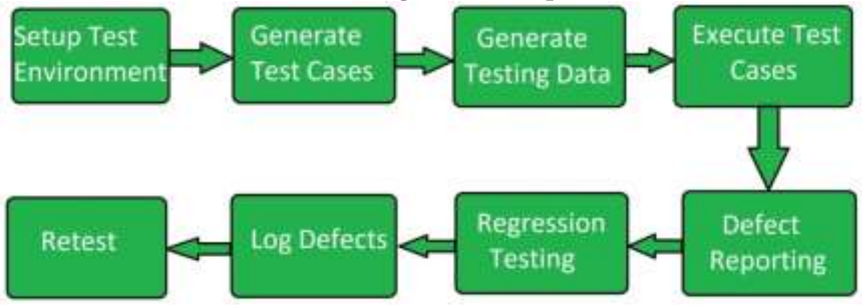
System Testing is a type of software testing that is performed on a complete integrated system to evaluate the compliance of the system with the corresponding requirements. In system testing, integration testing passed components are taken as input. The goal of integration testing is to detect any irregularity between the units that are integrated together. System testing detects defects within both the integrated units and the whole system.

System Testing is basically performed by a testing team that is independent of the development team that helps to test the quality of the system impartial. It has both functional and non-functional testing. **System Testing is a black-box testing**. System Testing is performed after the integration testing and before the acceptance testing.



System Testing Process: System Testing is performed in the following steps:

- **Test Environment Setup:** Create testing environment for the better quality testing.
- **Create Test Case:** Generate test case for the testing process.
- **Create Test Data:** Generate the data that is to be tested.
- **Execute Test Case:** After the generation of the test case and the test data, test cases are executed.
- **Defect Reporting:** Defects in the system are detected.
- **Regression Testing:** It is carried out to test the side effects of the testing process.
- **Log Defects:** Defects are fixed in this step.
- **Retest:** If the test is not successful then again test is performed.



Types of System Testing:

- **Performance Testing:** Performance Testing is a type of software testing that is carried out to test the speed, scalability, stability and reliability of the software product or application.
- **Load Testing:** Load Testing is a type of software Testing which is carried out to determine the behavior of a system or software product under extreme load.
- **Stress Testing:** Stress Testing is a type of software testing performed to check the robustness of the system under the varying loads.

- **Scalability Testing:** Scalability Testing is a type of software testing which is carried out to check the performance of a software application or system in terms of its capability to scale up or scale down the number of user request load.

Tools used for System Testing :

1. JMeter
2. Gallen Framework
3. Selenium

Here are a few common tools used for System Testing:

1. HP Quality Center/ALM
2. IBM Rational Quality Manager
3. Microsoft Test Manager
4. Selenium
5. Appium
6. LoadRunner
7. Gatling
8. JMeter
9. Apache JServ
10. SoapUI

Advantages of System Testing:

- Verifies the overall functionality of the system.
- Detects and identifies system-level problems early in the development cycle.
- Helps to validate the requirements and ensure the system meets the user needs.
- Improves system reliability and quality.
- Facilitates collaboration and communication between development and testing teams.
- Enhances the overall performance of the system.
- Increases user confidence and reduces risks.
- Facilitates early detection and resolution of bugs and defects.
- Supports the identification of system-level dependencies and inter-module interactions.
- Improves the system's maintainability and scalability.

Disadvantages of System Testing:

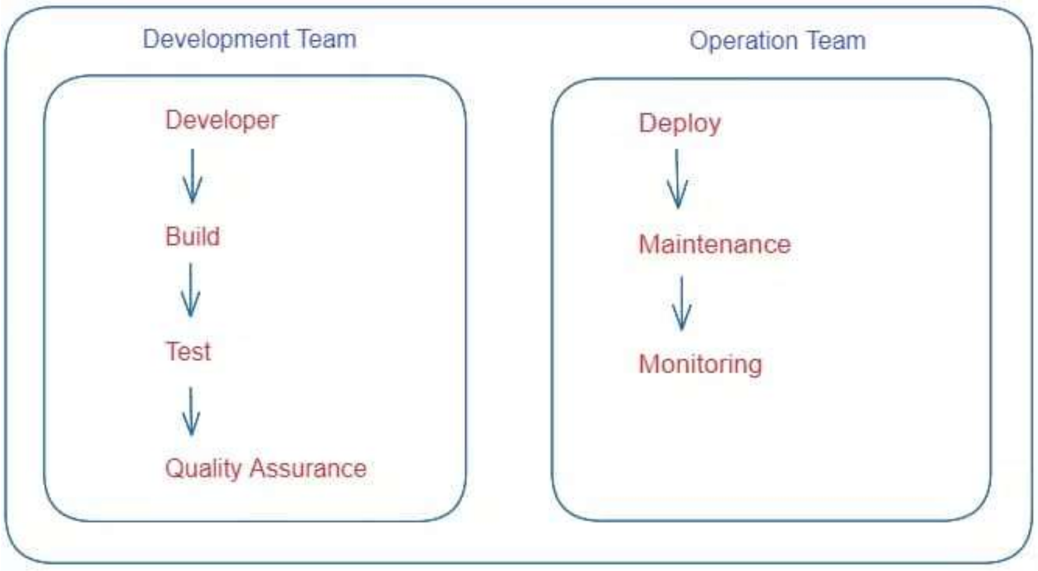
- Can be time-consuming and expensive.
- Requires adequate resources and infrastructure.
- Can be complex and challenging, especially for large and complex systems.
- Dependent on the quality of requirements and design documents.
- Limited visibility into the internal workings of the system.
- Can be impacted by external factors like hardware and network configurations.
- Requires proper planning, coordination, and execution.
- Can be impacted by changes made during development.
- Requires specialized skills and expertise.

- May require multiple test cycles to achieve desired results.

Job of Development and DevOps:

DevOps, at its core, focuses on implementing automation at each and every stage of the software development lifecycle.

The term **DevOps** is a combination of two terms: **Development** and **Operations**. DevOps aims to simplify and reduce the development life cycle of a system.
The DevOps Team is a combination of the Development Team and the Operations Team.



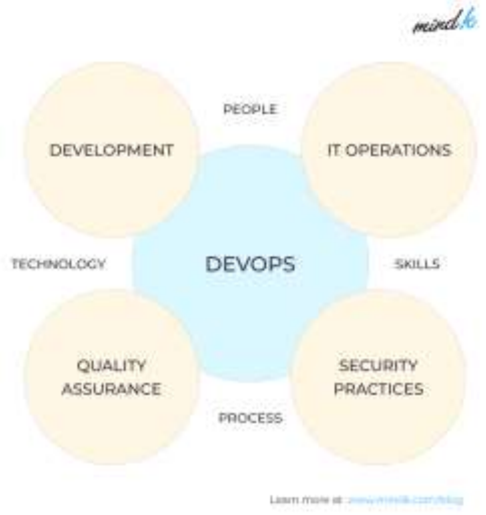
Development Team & Operations Team working separately

DevOps is a methodology that allows a single team to manage the entire application development life cycle – that is development, testing, deployment, and operations.
This approach can help your team produce superior quality software quickly and with more reliability.

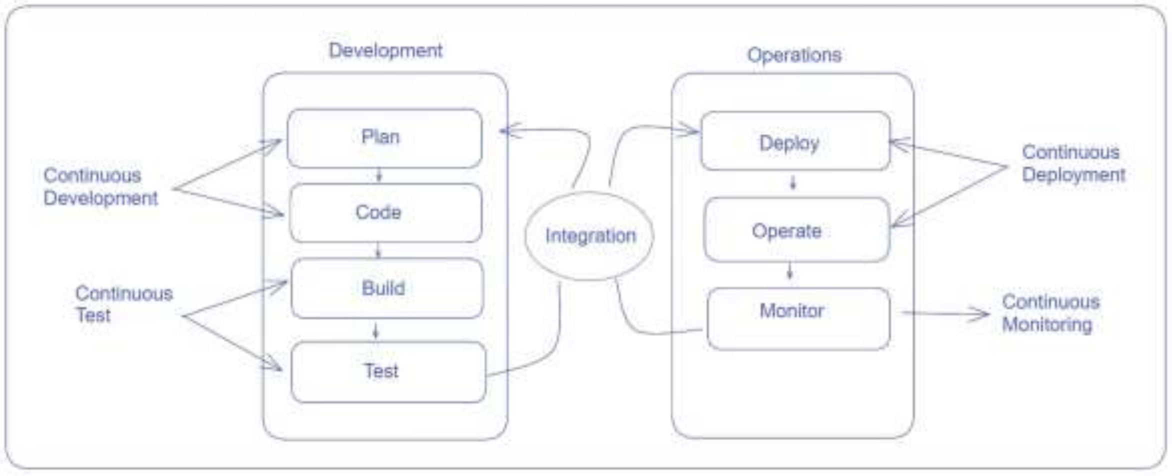
How Does DevOps Work?

Before DevOps, the development team and the operations team worked separately. This often created a lot of issues in production because both teams were working separately without consulting each other and integrating best practices.

Because of all of these issues, the DevOps team got created. Again, it combines the Development Team and the Operations Team and automates every step of the process.
Let's look at a sample workflow a DevOps team might follow:



Having a DevOps team doesn't necessarily mean all of its members are DevOps engineers. Companies are now favoring small, cross-functional teams that have all the capabilities you need to build a well-functioning product. As such, here's a typical DevOps team structure you can see in high-performing companies



Why Do You Need a DevOps Specialist?
A DevOps specialist plays an important role in achieving the goals of fast delivery, high quality, reduced costs, and improved reliability of software releases.

Fast delivery:
DevOps specialists help streamline the development process by automating build, test, and deployment pipelines, allowing the company to release software more quickly. They also help

identify and resolve bottlenecks in the development process, improving efficiency, and reducing delays.

High quality:
DevOps specialists work closely with development and operations teams to ensure that software releases meet the necessary quality standards. They also make sure that customer feedback is incorporated into future releases, allowing the company to continually improve the quality of its software.

Less Capex (Capital Expenditure) + Opex (Operational Expenditure):
DevOps specialists help identify and implement tools and processes that reduce the time and resources required for software releases. By automating manual tasks and reducing the need for manual intervention, organizations can save time and money, and improve overall efficiency.

Reduce outages:
DevOps specialists help reduce outages and improve reliability by implementing monitoring and alerting systems that proactively identify and address issues before they become major problems.

Job of Test and DevOps:

DevOps Testing is the continuous and automated process of software testing that enables continuous and faster delivery of software. The legacy approach of testing is performed manually. Manual testing involves more human activity, is more prone to error, and is more time-consuming. As DevOps focuses on automating processes, testing can fit into DevOps with the help of the right test automation tools.

Features of testing in a DevOps environment

Organizations are swiftly moving to DevOps for Agile teams. Both approaches focus on the automation of testing activity.

Let’s understand the features of DevOps testing

- Testing is automated, and it is continuous.
- Testing is carried out at different stages of SDLC.
- Easy to roll back and detect errors in the code as results/reports will be instant.
- Testing becomes a shared responsibility. Each team member is responsible for quality.

Who is Involved in DevOps Testing?

In DevOps Teams, everyone is equally responsible for the quality of the product. That means testing is done by the whole team. So no more blame game on the testing team. Designated testing team member will have expertise and skill set in the DevOps testing and he will drive the testing. Testing team member helps to choose the right tool, build the automation framework, automation code review, integrate testing activity with pipelines, etc. On the other hand, all team members will contribute to developing the automation scripts.

- In DevOps testing, the tester needs to have a broad understanding of development, testing, and tools.
- Some of the Key areas that DevOps QA members should focus on are given below.
- Source Control (using Git, Bitbucket, AzureDevOps, etc)
- Continuous Integration (using Jenkins, Bamboo, AzureDevOps)
- Deployment Automation & Orchestration (using Jenkins, AzureDevOps, Octopus Deploy)
- Container Concepts (Kubernetes, Docker)
- Cloud (using AWS, Azure, GoogleCloud, etc)

Apart from above knowledge, tester may also need to write code in specific languages such as Java, Javascript, Python, C#, etc.

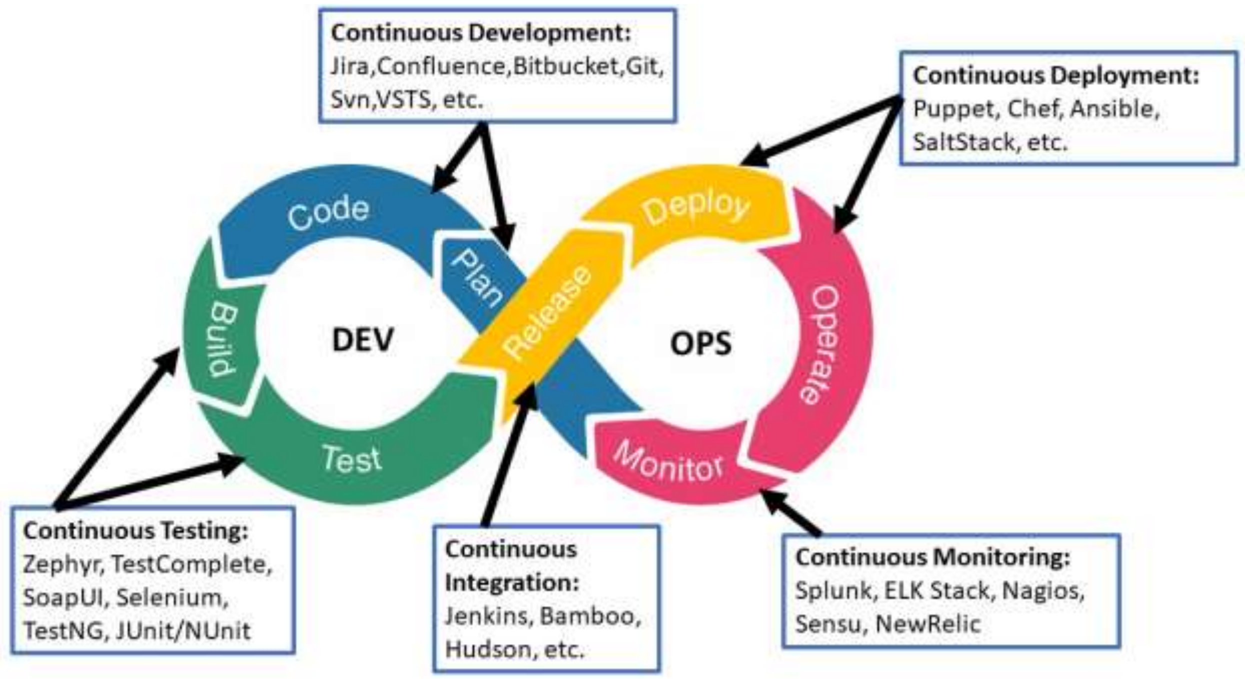
Role of Test Automation Tools:

DevOps is all about automating the process. Choosing the right tool at every stage is important. As we know more tools bring more maintenance and more complexity. Considering the DevOps testing you need to choose the tool carefully.

As there are many testing types you cannot have different testing tools for each type of testing. This may make your testing more complex and demands more skilled resources. Instead, choose the automation tool which supports different types of testing, less demanding skills, and is easy to integrate with the DevOps ecosystem. Testsigma is one such tool built with a DevOps mind.

DevOps Testing Strategy:

Choosing the right devops automation tools alone cannot make DevOps testing successful. DevOps testing strategy plays a key role in the success of DevOps testing. The strategy should be built carefully, observing the organization's skillsets, ecosystem, architecture, and release patterns. Below are some helpful pointers.



- Identify the test cases which need to be executed in specific builds and create a suite accordingly.
- Focus more on the unit tests, as it is faster and more stable.
- Use the test coverage tools and quality gates to ensure the unit testing criteria are met.
- End-to-end tests should focus on end-user scenarios only.
- Have quality checks for all levels and all environments such as QA, Staging Production, etc.
- Ensure the automation tool supports parallel testing and compatibility testing, and build the automation architecture around that.
- Testing should be carried out on different platforms such as Mac, Windows, and Linux with different supported browsers like Chrome, Firefox, Edge, etc.
- Make the development code author a primary responsible person to look at the failures. This way, testing responsibility will be delegated across the team members.
- Adapt the monitoring tools such as Splunk, Datadog, Graphana, etc., which help to continuously monitor the health of the application.
- Consider building the test harness tools to support the test cases which cannot be automated directly. Find a way to automate using the workarounds

Job of Op and Devops:

Operations can mean a lot of things and even different things to different people. DevOps is becoming more and more popular but a lot of people are confused to exactly what it is. So let's make a list of all the things operations traditionally does and figure out what developers should be doing, and which if any responsibilities should be shared.

DevOps vs Operations: Responsibilities

Operations responsibilities

- IT buying
- Installation of server hardware and OS
- Configuration of servers, networks, storage, etc…
- Monitoring of servers
- Respond to outages
- IT security
- Managing phone systems, network
- Change control
- Backup and disaster recovery planning
- Manage active directory
- Asset tracking

Shared Development & Operations duties

- Software deployments
- Application monitoring & support
- Server provisioning and configuration

Some of these traditional responsibilities have changed in the last few years. Virtualization and the cloud have greatly simplified buying decisions, installation, and configuration. For example, nobody cares what kind of server we are going to buy anymore for a specific application or project. We buy great big ones, virtualize them, and just carve out what we need and change it on the fly. Cloud hosting simplifies this even more by eliminating the need to buy servers at all.

So what part of the "Ops" duties should developers be responsible for?

- Be involved in selecting the application stack
- Configure and deploy virtual or cloud servers (potentially)
- Deploy their applications
- Monitor application and system health
- Respond to applications problems as they arise

Developers who **take ownership** of these responsibilities can ultimately deploy and support their applications more rapidly. DevOps processes and tools eliminate the walls between the teams and enables more agility for the business. This philosophy can enable the developers to potentially be responsible for the entire application stack from OS level and up in more a **self service mode**.

So what does the operations team do then?

- Manage the hardware infrastructure
- Configure and monitor networking
- Enforce policies around backup, DR, security, compliance, change control, etc
- Assist in monitoring the systems

- Manage active directory
- Asset tracking
- Other non-production application-related tasks
- Install & manage 3rd party software on-premise

Depending on the company size, the workload of these tasks will vary greatly. In large enterprise companies these operations tasks become complex enough to require specialization and dedicated personnel for these responsibilities. For small to midsize companies the IT manager and 1-2 system administrators can typically handle these tasks.

DevOps is evolving into letting the operations team focus on the infrastructure and IT policies while empowering the developers to exercise tremendous ownership from the OS level and up. This enables development teams to be more self-service and independent of a busy centralized operations team. DevOps enables more agility, better efficiency, and ultimately a higher level of service to their customers.

Infrastructure and the job of Ops:

IT infrastructure teams are responsible for managing the physical hardware that supports the systems, networks, and storage necessary for IT service delivery. This includes maintaining mainframes, system security, and network switches (network management), installing and patching hardware, monitoring assets, and configuring, deploying, or provisioning servers.

IT infrastructure is traditionally housed in an on-premises data center, where IT infrastructure teams are responsible for power and cooling. However, infrastructure is increasingly housed in third-party colocation facilities or virtual IT environments via a cloud provider. In such cases, **IT infrastructure** teams are primarily focused on integrating IT assets between cloud-based and on-premises environments, as well as maintaining the security and integrity of data across those environments.

What is IT Operations Management?

IT operations teams are responsible for the applications, processes, and platforms that support IT and business functions. This includes configuring, installing, and maintaining software, as well as database management, preventing downtime, disaster recovery, and deploying and integrating new technologies.

IT operations also manages help desks and devices to further support business needs.

IT Operations Management in the ITIL:

The information technology infrastructure library (ITIL) is a framework of best practices for providing IT services to the larger organization. ITIL breaks the IT service lifecycle into 5 stages, one of which is Service Operations, which contains ITOM. **Infrastructure management** is a sub-function of ITIL ITOM.

While the ITIL framework is by no means universally employed, it is widely regarded as a standard bearer that reflects larger trends within IT. With the recent release of ITIL 4 (2019), a few trends stand out:

- Integrating –not aligning– IT with the larger organization
- Focusing on collaboration within IT and between IT and other departments
- Implementing agile and DevOps methodologies

Bringing Infrastructure and Operations Together:

The need for rapid development is part of the reason why infrastructure and operations are increasingly lumped together, whether placing infrastructure within IT Operations Management or whether referring to **infrastructure and operations (I&O)**.

IT infrastructure has traditionally been an IT subset that few people (even within IT) ever had to worry about. But as data becomes the lifeblood of organizations, and as servers and cloud-computing become the arteries that move that data, infrastructure is suddenly critical to day-to-day operations.

New technologies such as infrastructure-as-code and low-code process automation platforms are making it easier for IT operations teams, developers, and even business personnel to quickly and reliably provision the servers, resources, and devices they need to complete their jobs.

Common IT Operations job titles:

Let's review some of the most common ITOps roles in an enterprise IT organization:

- **Operations Engineer** manages operations for environment implementation, application deployment and optimization of ITSM framework best practices.
- **Operations Manager** oversees the design and implementation of ITOps from a business and technical aspect.
- **Cloud Architect** designs the cloud environment with respect to the organization's business requirements. Additional responsibilities include Infrastructure deployment, maintenance, monitoring, management, and security.
- **Business Architect** interprets organizational policies, risks and goals, and bridges the gap between strategy and execution. Synthesizes the strategy development process across all departments to ensure that the collective vision of all teams are aligned with the organization.
- **Systems Integrator** designs, builds, and synthesizes all IT system subsets used collectively to deliver an IT service or solution.
- **Incident and Security Manager** develops strategies and systems that help reduce risk and maximize dependability of an IT service.
- **Applications Analyst** is responsible for monitoring, administration, and maintenance of software applications, interacting data and the underlying IT infrastructure.

ITOps + DevOps functions:

Of course, IT operations does have some intersecting responsibilities that also support application and software, including development and testing. In particular, DevOps does bring IT operations personnel closer to IT applications teams in the following areas:

- Application support, configurations, installations, and troubleshooting
- Database maintenance
- Application and infrastructure monitoring
- Network management
- Service desk support
- Financial auditing and reporting
- IT Operations help desk and support

These responsibilities are assigned across several roles, as your IT organization deems most appropriate and effective.

****