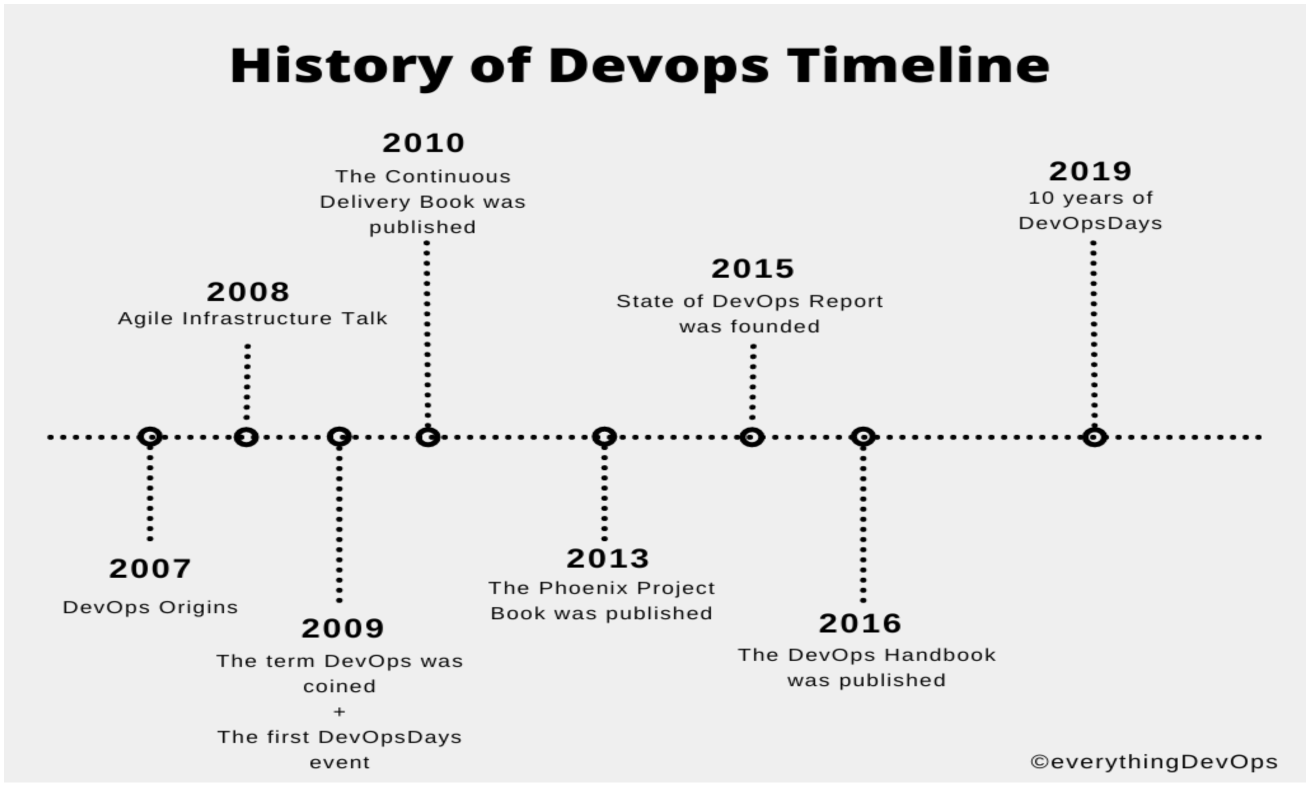
UNIT-III

1. History of DevOps

Patrick Debois is often referred to as the father of DevOps. Debois wanted to learn IT from all possible perspectives.

1. **Before DevOps (Pre-2000s):**
   - Software development and IT operations were separate. Developers wrote code, and operations teams managed the deployment and maintenance of that code.
   - This separation often led to communication gaps and inefficiencies.
2. **Agile Movement (2000s):**
   - The Agile Manifesto was published in 2001 by a group of software developers.
   - It introduced principles like iterative development, collaboration, and responsiveness to change, which began to challenge the traditional Waterfall approach.
   - This started breaking down barriers between developers and operations.
3. **Birth of DevOps (2009-2010):**
   - The term "DevOps" was coined to describe a new approach where development and operations teams work together more closely. The first DevOpsDays conference in 2009 in Belgium. This event helped to formalize the DevOps movement and gather practitioners to discuss and share ideas.
4. **Growth and Tools (2011-Present):**
   - DevOps practices became popular, focusing on automation, continuous integration, and continuous delivery. Tools like Jenkins, Docker, Kubernetes, Ansible, and Terraform. These tools helped automate various aspects of development, deployment and operations.
5. **Cultural Shift and Security (Recent Years):**
   - DevOps is not just about tools but also about a cultural shift toward collaboration and shared responsibility.
   - The rise of cloud computing and microservices architectures has further accelerated the adoption of DevOps.
   - Recently, there's been a focus on integrating security into DevOps practices (known as DevSecOps).


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

2. Definition of DEVOPS

DevOps is a set of practices, principles, and cultural philosophies that aim to improve collaboration between software development (Dev) and IT operations (Ops) teams. The goal of DevOps is to enhance the efficiency and quality of software delivery by fostering a more integrated and collaborative approach.

Purpose of DevOps:

1. **Improving Collaboration:**
   - **Breaking Down Silos:** DevOps encourages collaboration between development and operations teams, which traditionally worked in isolation. This helps to align goals and responsibilities, leading to more cohesive and efficient workflows.
2. **Accelerating Delivery:**
   - **Faster Releases:** By using practices like Continuous Integration (CI) and Continuous Delivery (CD), DevOps aims to speed up the

software release cycle. Automated testing and deployment processes enable frequent and reliable updates.

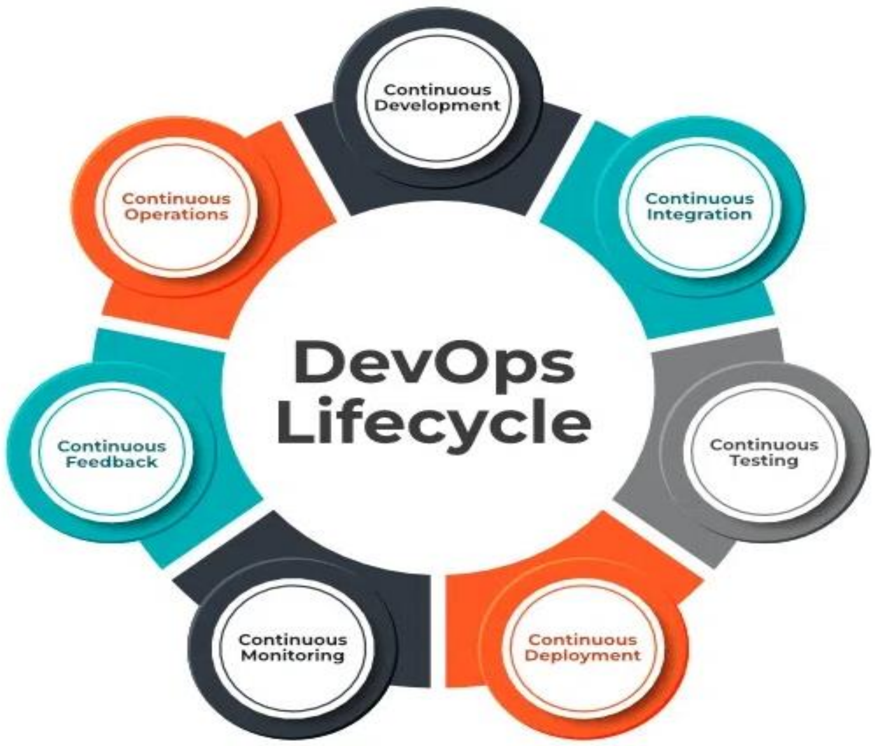
3. **Enhancing Quality:**
    - **Automated Testing and Monitoring:** DevOps practices include automated testing and continuous monitoring to catch issues early and ensure that software is stable and performs well in production environments.
4. **Increasing Efficiency:**
    - **Automation:** DevOps relies on automation to handle repetitive tasks such as testing, deployment, and infrastructure management. This reduces manual errors and frees up teams to focus on higher-value activities.
5. **Ensuring Reliability:**
    - **Infrastructure as Code (IaC):** DevOps promotes the use of IaC to manage infrastructure through code, making it easier to maintain, scale, and ensure consistency across different environments.
6. **Promoting Continuous Improvement:**
    - **Feedback Loops:** DevOps incorporates continuous feedback loops from monitoring and user experiences to drive ongoing improvements in software and processes.
7. **Integrating Security (DevSecOps):**
    - **Security as a Priority:** DevOps increasingly integrates security practices throughout the development lifecycle, known as DevSecOps, to address vulnerabilities early and ensure that security is a shared responsibility.

3. LIFE CYCLE OF DEVOPS

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. Some of the frequently used tools are open source and are carried out primarily based on commercial enterprise requirements. DevOps lifecycle is effortless to manipulate and it helps satisfactory delivery.

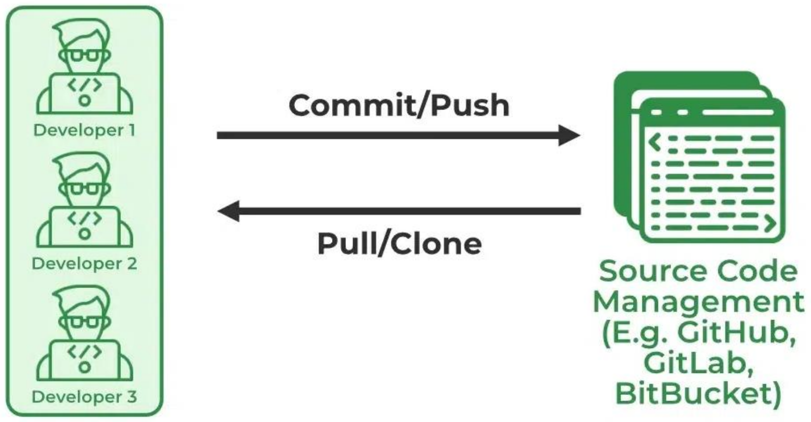
7 Cs of DevOps

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.

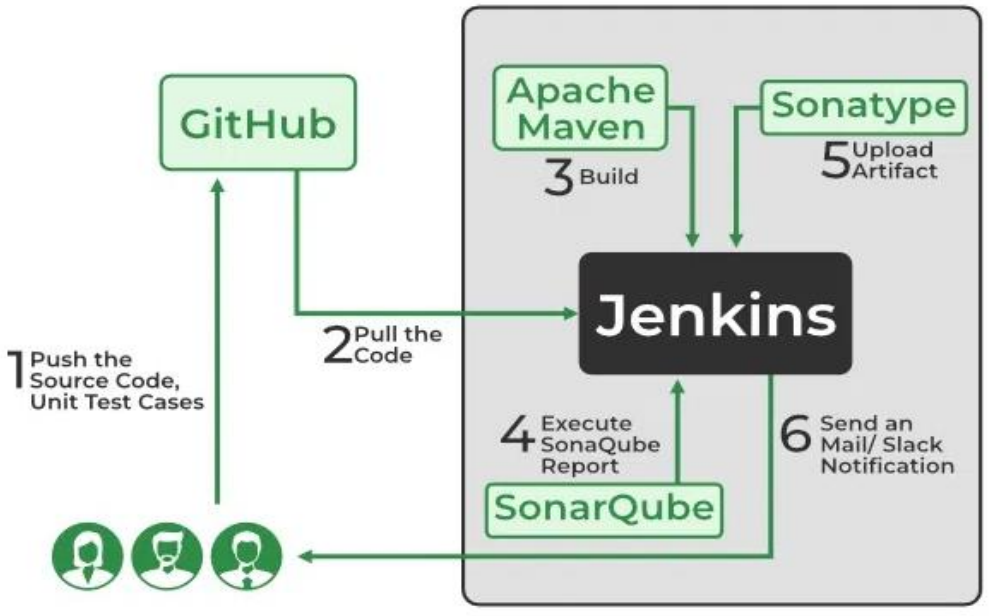


2. Continuous Integration

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

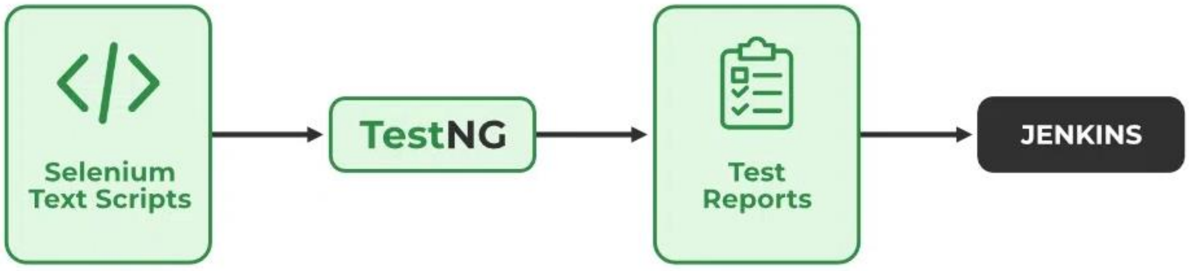
1. Getting the Source Code from SCM
2. Building the code
3. Code quality review
4. Storing the build artifacts

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. From there by using **Maven we can build** them into the required package (war, jar, ear) and can test the Junit cases. **SonarQube performs code quality reviews** where it will measure the quality of source code and generates a report in the form of HTML or PDF format. **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
3 Build
Sonatype
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

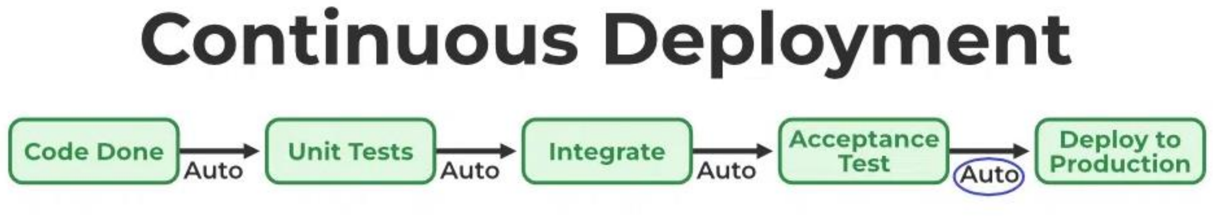
3. Continuous Testing

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 benefit.

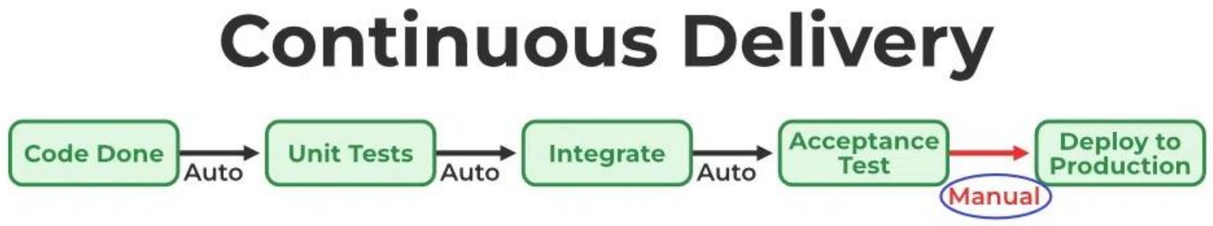


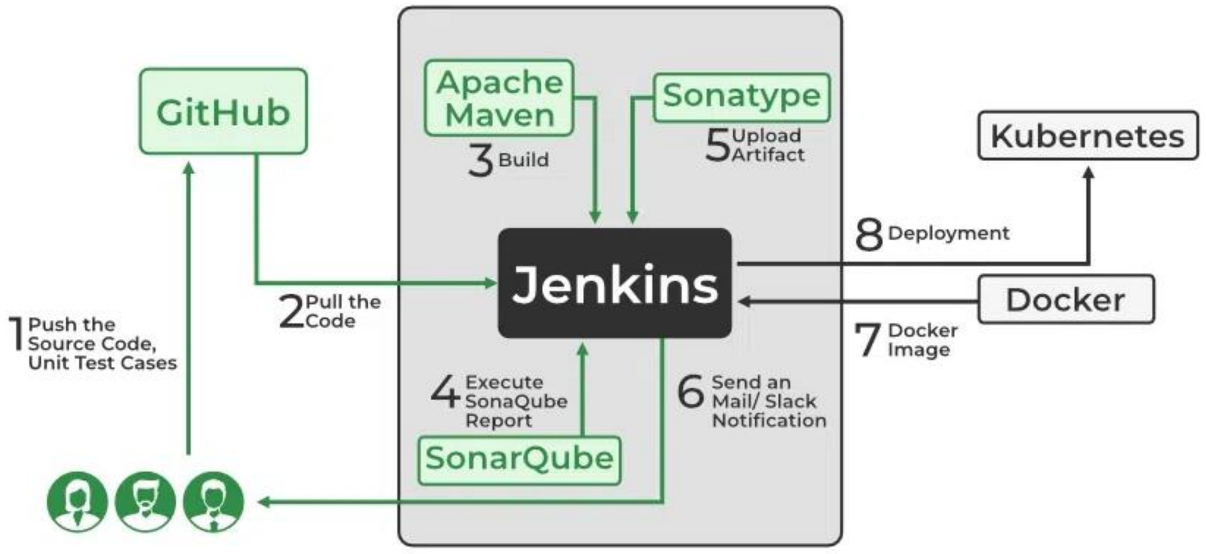
4. Continuous Deployment/ Continuous Delivery

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.

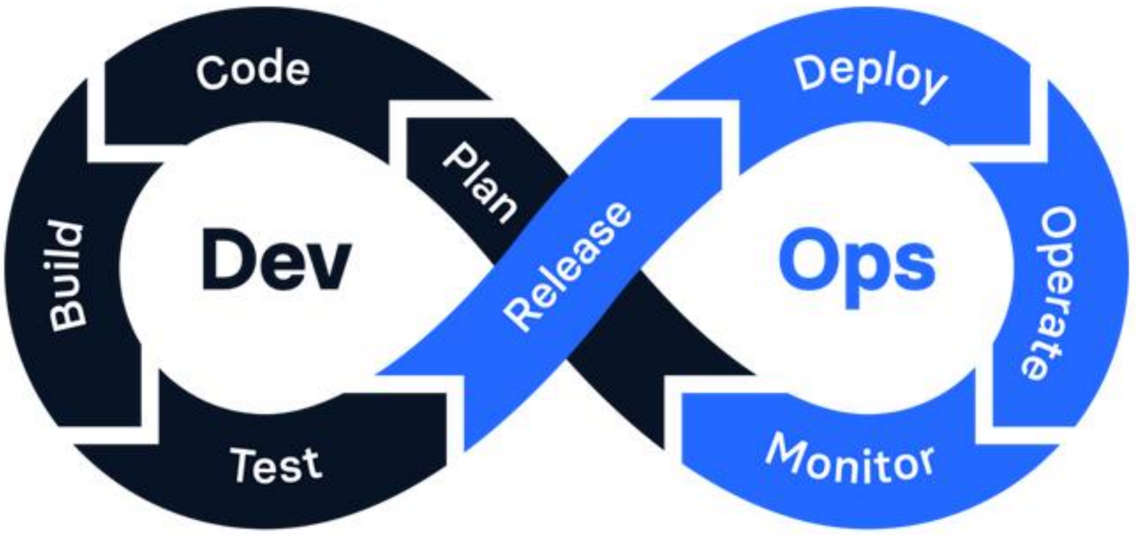
6. Continuous Feedback

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.

4.Different Phases of the DevOps Lifecycle



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 employs 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 tasks, 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.

5.Benefits and Challenges of DevOps

Benefits of DevOps

1. **Faster Delivery:** DevOps enables organizations to release new products and updates faster and more frequently, which can lead to a competitive advantage.

2. **Improved Collaboration:** DevOps promotes collaboration between development and operations teams, resulting in better communication, increased efficiency, and reduced friction.

3. **Improved Quality:** DevOps emphasizes automated testing and continuous integration, which helps to catch bugs early in the development process and improve the overall quality of software.

4. **Increased Automation:** DevOps enables organizations to automate many manual processes, freeing up time for more strategic work and reducing the risk of human error.

5. **Better Scalability:** DevOps enables organizations to quickly and efficiently scale their infrastructure to meet changing demands, improving the ability to respond to business needs.

6. **Increased Customer Satisfaction:** DevOps helps organizations to deliver new features and updates more quickly, which can result in increased customer satisfaction and loyalty.
7. **Improved Security**: DevOps promotes security best practices, such as continuous testing and monitoring, which can help to reduce the risk of security breaches and improve the overall security of an organization's systems.
8. **Better Resource Utilization**: DevOps enables organizations to optimize their use of resources, including hardware, software, and personnel, which can result in cost savings and improved efficiency.

Challenges While Adopting DevOps

- **High Initial Investment:** Implementing DevOps can be a complex and costly process, requiring significant investment in technology, infrastructure, and personnel.

- **Skills Shortage**: Finding qualified DevOps professionals can be a challenge, and organizations may need to invest in training and development programs to build the necessary skills within their teams.

- **Resistance to Change**: Some employees may resist the cultural and organizational changes required for successful DevOps adoption, which can result in resistance, resistance to collaboration, and reduced efficiency.

- **Lack of Standardization**: DevOps is still a relatively new field, and there is a lack of standardization in terms of methodologies, tools, and processes. This can make it difficult for organizations to determine the best approach for their specific needs.

- **Dependency on Technology**: DevOps relies heavily on technology, and organizations may need to invest in a variety of tools and platforms to support the DevOps process.

- **Need for Continuous Improvement**: DevOps requires ongoing improvement and adaptation, as new technologies and best practices emerge. Organizations must be prepared to continuously adapt and evolve their DevOps practices to remain competitive.

6. DevOps Engineer Job Description

A DevOps Engineer combines software development and IT operations to improve how software is built and deployed. This role involves creating and managing systems that help teams work together more efficiently, ensuring that updates and new features are released quickly and reliably.

Responsibilities

- **Build and Maintain Tools:** Create and manage tools that automate software development and deployment processes.

- **Collaborate with Teams:** Work closely with software developers and IT staff to ensure smooth and fast delivery of applications.

- **Monitor Systems:** Keep an eye on system performance and fix any issues that arise to ensure everything runs smoothly.

- **Improve Processes:** Continuously look for ways to make the software development and deployment processes more efficient.

- **Ensure Security:** Implement practices to keep systems secure from potential threats.

7. key components of Devops culture

DevOps culture involves closer collaboration and a shared responsibility between development and operations for the products they create and maintain. This helps companies align their people, processes, and tools toward a more unified customer focus.

It involves cultivating multidisciplinary teams who take accountability for the entire lifecycle of a product. DevOps teams work autonomously and embrace a software engineering culture, workflow, and toolset that elevates operational requirements to the same level of importance as architecture, design and development. The understanding that developers who build it, also run it, brings developers closer to the user, with a greater understanding of user requirements and needs. With operations teams more involved in the development process, they can add maintenance requirements and customer needs for a better product.

At the heart of DevOps culture is **increased transparency, communication, and** collaboration between teams that traditionally worked in siloes. But there are important cultural shifts that need to happen to bring these teams closer together. DevOps is an organizational culture shift that emphasizes continuous learning and continuous improvement, especially through team autonomy, fast feedback, high empathy and trust, and cross-team collaboration.

DevOps entails **shared responsibilities**. Development and operations staff should both be responsible for the success or failure of a product. Developers are expected to do more than just build and hand off to operations — they are expected to share the responsibility of overseeing a product through the entire course of its lifetime, adopting a "you build it, you run it" mentality. They test and operate software and collaborate more with QA and IT Ops. When they understand the challenges faced by operations, they are more likely to simplify deployment and maintenance. Likewise, when operations understand the system’s business goals, they can work with developers to help define the operational needs of a system and adopt automation tools.

Autonomous teams are another important aspect of DevOps. For the development and operations teams to collaborate effectively, they need to make decisions and implement changes without a cumbersome and lengthy approval process. This involves handing over trust to teams and establishing an environment where there is no fear of failure. These teams should have the right processes and tools in place to make decisions faster and easier, for each level of risk to the customer.

For example, a typical development workflow might require engagement from several contributors on different teams to deploy code changes. The developer makes a code change and pushes it to a source control repository, a build engineer builds and deploys the code to a test environment, a product owner updates the status of the work in an issue tracking tool, etc. An autonomous team will take advantage of tools that automate these processes, so that pushing new code triggers the building and deployment of a new feature into a test environment and the issue tracking tool is updated automatically.

For example, a team is handicapped by requirements like having to open a ticket with a separate operations team to make a trivial infrastructure change, such as a new DNS entry. A task that should take seconds to complete ends up taking days or weeks to satisfy. An autonomous team has the ability to implement such changes themselves, whether by having an individual on the team who has the correct skills and experience, or by having access to self-service tooling.

A DevOps team culture values fast **feedback** that can help with continuous improvement of a unified development and operations team. In an environment where the development and operations teams are in isolated silos, feedback about the performance and stability of application software in production is often slow to make it back to the development team, if it makes it at all. DevOps ensures that developers get the fast feedback they need to rapidly iterate and improve on application code by requiring collaboration between operations folks in designing and implementing application monitoring and reporting strategies. For example, any sufficiently capable continuous integration tool will enable automated build and testing of new code pushes and provide the developer with immediate feedback on the quality of their code.

Automation is essential to DevOps culture, since it allows great collaboration and frees up resources. Automating and integrating the processes between software development and IT teams helps them to build, test, and release software faster and more reliably

8. Similarities between agile and DevOps.

- Agile emphasizes collaboration between developers and product management — DevOps includes the operations team
- Agile centers the flow of software from ideation to code completion — DevOps extends the focus to delivery and maintenance
- Agile emphasizes iterative development and small batches — DevOps focuses more on test and delivery automation
- Agile adds structure to planned work for developers — DevOps incorporates unplanned work common to operations teams

The Agile Manifesto explicitly prioritizes individuals and interactions, working software, customer collaboration, and responding to change. These are clearly the same priorities of DevOps but extended beyond the development process and into the management of systems and running applications.

In addition, the Twelve Principles of Agile Software includes references to DevOps principles. For example, the emphasis on continuous integration and delivery, working in small batches with frequent releases, and using automation are all referenced in the Twelve Principles of Agile Software.

9. Differences between agile and DevOps

Agile is an iterative approach to project management and software development that focuses on collaboration, customer feedback, and rapid releases. It arose in the early 2000s from the software development industry, helping development teams react and adapt to changing market conditions and customer demands.

DevOps is an approach to software development that enables teams to build, test, and release software faster and more reliably by incorporating agile principles and practices, such as increased automation and improved collaboration between development and operations teams. Development, testing, and deployment occur in both agile and DevOps.

Now, it may look like DevOps and Agile do the same thing. But if we pay more attention, we'll notice they're not entirely similar. Here are their critical differences:

- **Implementation:** DevOps brings development and operations together. Meanwhile, Agile is a more interactive approach focused on continuous development, highlighting collaboration, customer feedback, and smaller, quick-fire releases.
- **Purpose**: Agile is about getting the software out there; DevOps is about delivering ready-to-use software reliably and securely.
- **Product/Service:** DevOps strives for "perfect readiness" better quality, owing to automation and early bug removal. Conversely, Agile is about creating better application suites with the desired requirements.
- **Focus**: DevOps focuses on end-to-end business solutions and fast deliveries; Agile focuses more on software development. The main focus of DevOps is business and operational readiness, while Agile is about functional and non-functional willingness.
- **Tasks**: The Agile process is about constant changes, whereas DevOps is more about continuous testing.

- **Team size**: Agile projects require smaller teams, while DevOps tend to require more extensive, well-coordinated teams.
- **Teams skill-set**: Agile teams tend to consist of individuals trained to possess more or less the same skills. In DevOps, those skills are divided between the operations and development teams. Thus, in Agile, everyone can do anything, which facilitates communication, and in DevOps, that communication can be more complex due to the differences in team skill sets.
- **Duration**: Agile development usually happens in “sprints,” less than a month. On the contrary, DevOps strives for deadlines and benchmarks for significant projects, with production-ready code expected daily.
- **Where feedback comes from**: In DevOps, feedback is provided by the internal team. In the case of Agile, it comes straight from the customer’s mouth.