Fundamentals of Devops:-

1. **What is Devops?**
2. **Devops Principles**
3. **Devops Lifecycle**
4. **Devops delivery pipeline**
5. **Devops Ecosystem and Tools**
6. **The Agile wheel of wheels.**

1. What is Devops?

The DevOps is a combination of two words, one is software Development, and second is Operations. This allows a single team to handle the entire application lifecycle, from development to **testing, deployment**, and **operations**. **In 2009, the first conference named DevOpsdays was held in Ghent Belgium. Belgian consultant and Patrick Debois founded the conference.**

DevOps helps you to reduce the disconnection between software developers, quality assurance (QA) engineers, and system administrators.DevOps promotes collaboration between Development and Operations team to deploy code to production faster in an automated & repeatable way.DevOps helps to increase organization speed to deliver applications and services. It also allows organizations to serve their customers better and compete more strongly in the market.

Fig.1.Devops

DevOps History

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

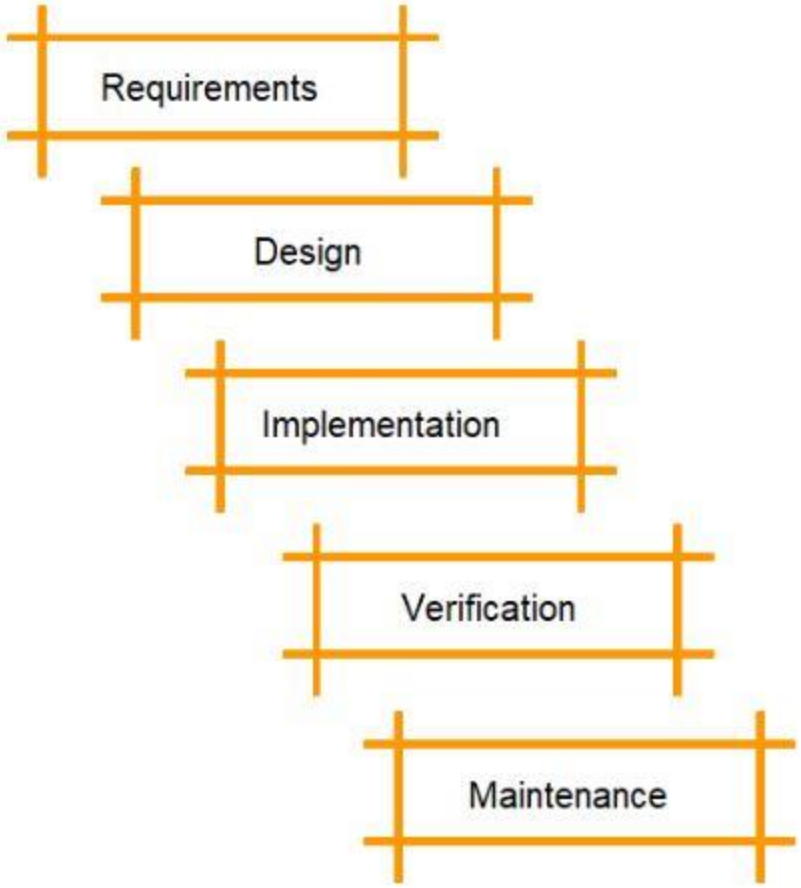
Why DevOps?

Before we get deep into what DevOps is and all the revolutions it brought with us, first understand why DevOps in the first place. and Before DevOps, there were two development models: Waterfall and Agile Method.

1. Waterfall Model

The waterfall model is the first model to be introduced in software development. It is a sequential process and very easy to understand. In this approach, software development is divided into several phases, and the output of one phase becomes the input for the next phase. This model is similar to a waterfall when the water flows off from the cliff; it cannot go back to its previous state.

The phases are; Requirements, Design, Implementation, Verification, and Maintenance.



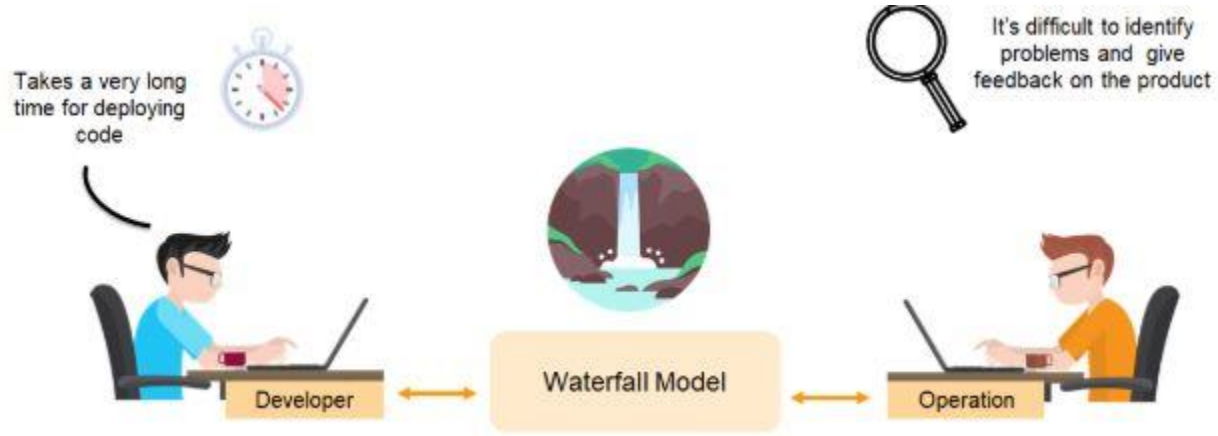
Drawbacks of the waterfall model:

- It’s difficult to make changes to the previous stage
- Not recommended for large-sized projects
- Developers and testers don’t work together (which can result in a lot of bugs at the end)
- Not recommended for projects that will likely have changing requirements

From the figure below, we can see the issues with the waterfall model:

- The developer took a very long time to deploy code

- On the operations side, the tester found it challenging to identify problems and give useful feedback

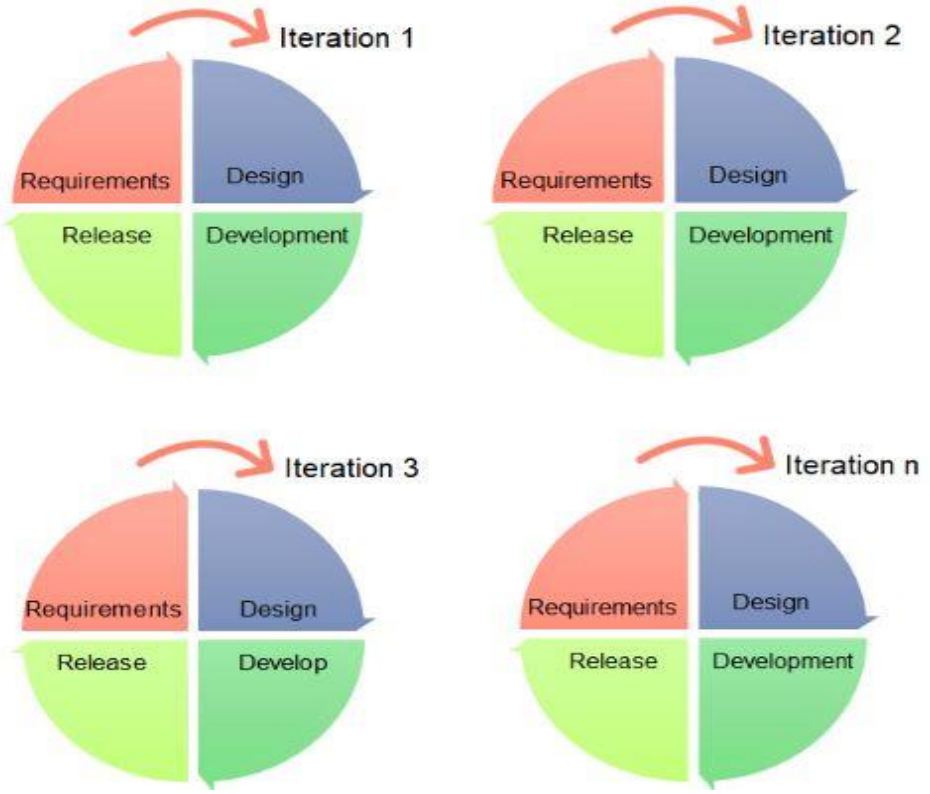


2. Agile Model

Agile is an approach in software development where each project splits into multiple iterations. As a result, at the end of each iteration, a software product is delivered. Each iteration lasts about one to three weeks. Every iteration involves functional teams working simultaneously on various areas, such as:

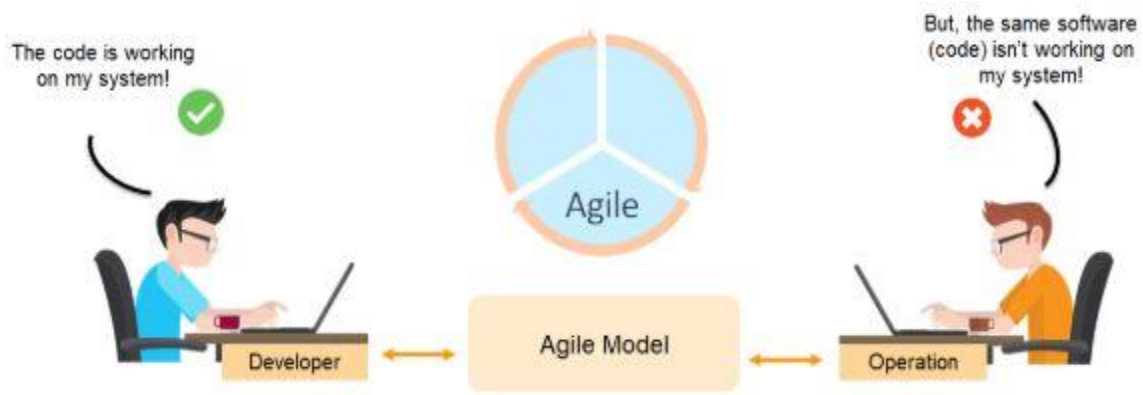
- Requirements
- Design
- Development
- Release

The figure below indicates that there can be a number of iterations needed to deliver a final product in agile method.



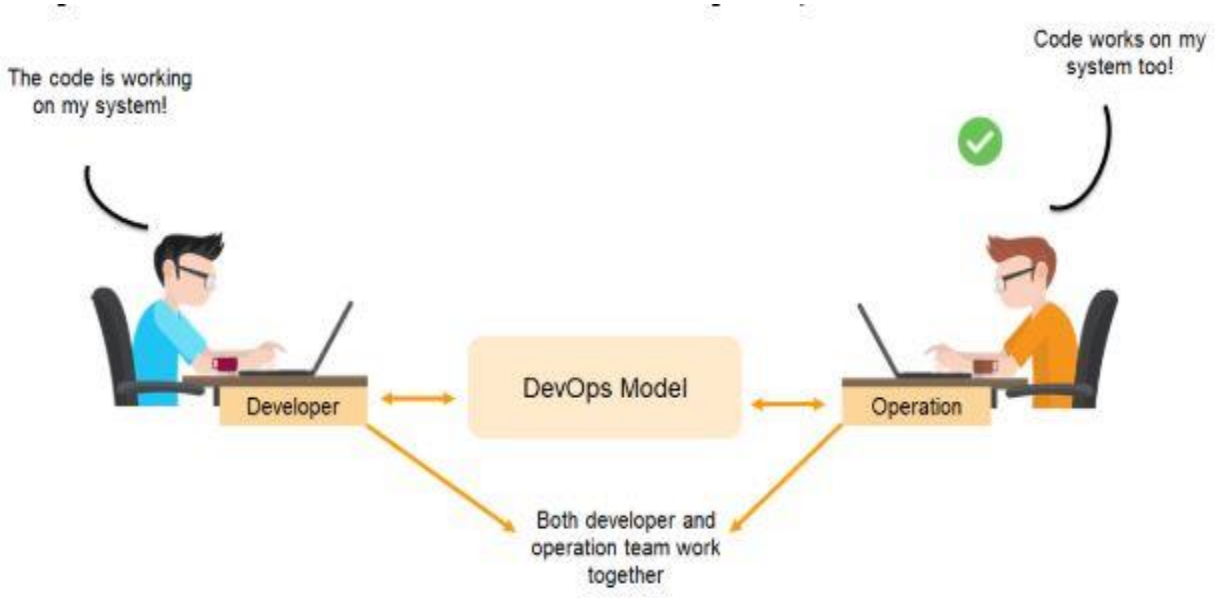
Using the agile method, the code that works for the developer may not work for the operations team.

So how can this issue be solved?



With DevOps, there is continuous integration between deployment of code and the testing of it. Near real-time monitoring and immediate feedback through a DevOps continuous monitoring tool enables both the developer and operations team work together.

The figure below shows how well the software is handled using DevOps.



Some more points for need devops:-

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

2. Devops Principles

1. Collaboration and Communication

DevOps emphasizes collaboration and open communication between development, operations, and other involved teams. By removing silos and fostering a culture where teams work together toward shared goals, DevOps encourages regular interaction, transparency, and joint ownership of the entire software development lifecycle, improving efficiency and resolving issues faster.

2. Automation

Automation is a core principle in DevOps, focusing on reducing manual tasks to increase speed and consistency. Continuous integration (CI) and continuous delivery (CD) automate the process of building, testing, and deploying software. Infrastructure automation using Infrastructure-as-Code (IaC) ensures that infrastructure is provisioned and managed consistently, eliminating human error and reducing time spent on repetitive tasks.

3. Continuous Improvement

DevOps promotes continuous improvement by iterating on processes, tools, and products. Teams regularly review and refine their workflows, incorporating feedback from various stakeholders. This iterative approach leads to faster adaptation to changes, encourages experimentation, and allows organizations to learn from failures, constantly evolving to meet new challenges.

4. Consistency and Standardization

In DevOps, maintaining consistency and standardization across environments is crucial to reduce errors and improve efficiency. By using tools like containers and automated configuration management, teams ensure that code behaves the same way in development, testing, and production. This consistency helps eliminate deployment issues like "it works on my machine" and reduces the risk of environmental discrepancies.

5. Lean and Agile Practices

DevOps integrates lean and agile practices to streamline development and delivery. Lean thinking focuses on minimizing waste and optimizing the flow of work, while agile methodologies encourage delivering small, incremental updates. This combination helps teams quickly respond to customer feedback, adapt to changes, and continuously improve product quality.

6. Continuous Testing

Continuous testing is integral to DevOps, ensuring that code changes are automatically validated early and throughout the development process. By automating testing at multiple levels (unit, integration, system), DevOps encourages early detection of bugs, reduces the cost of fixing defects, and ensures that software is always in a deployable state.

7. Monitoring and Logging

Monitoring and logging are critical components of the DevOps process, providing real-time insights into application performance and system health. Continuous monitoring helps teams detect issues proactively,

while centralized logging allows for quick diagnosis of problems across different systems. This visibility helps improve performance, reliability, and user experience.

8. Security (DevSecOps)

DevSecOps integrates security into the DevOps pipeline, making it an integral part of the development process rather than an afterthought. Automation of security checks, vulnerability assessments, and regular code scanning ensures that software is secure at every stage. This approach shifts security "left" to identify and address vulnerabilities early, making security a shared responsibility among development, operations, and security teams.

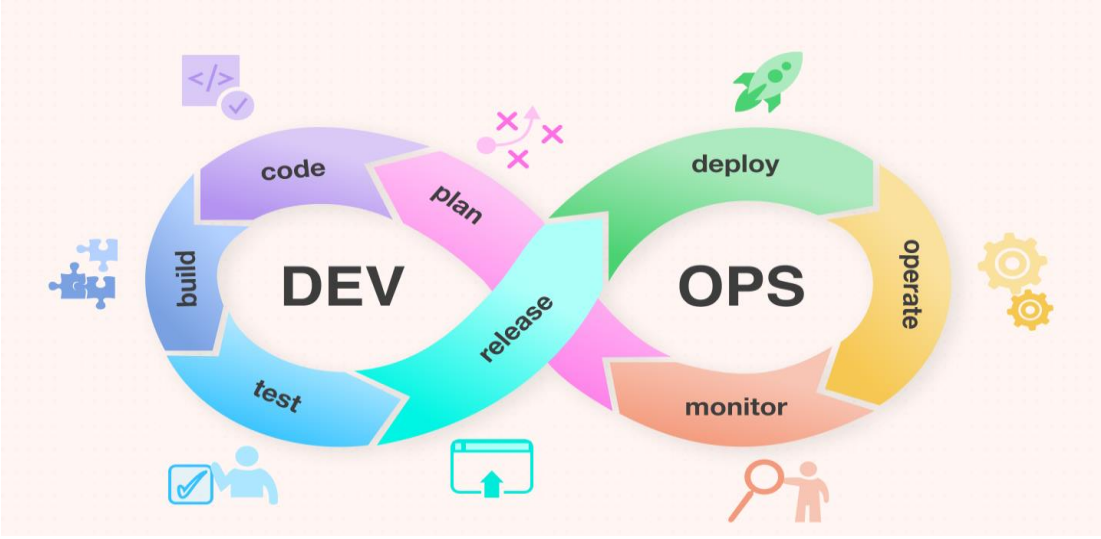
9. Culture of Empowerment

DevOps fosters a culture where teams are empowered to take ownership of both their code and the infrastructure that runs it. Teams are encouraged to collaborate, innovate, and make decisions autonomously, improving job satisfaction and accountability. This culture of empowerment leads to higher engagement, faster problem resolution, and better quality outcomes.

10. Infrastructure as Code (IaC)

Infrastructure as Code (IaC) is a key DevOps practice where infrastructure is managed and provisioned using code instead of manual processes. IaC enables teams to automate the setup of servers, networks, and other resources, ensuring consistency across environments. It allows for easy scaling, quick recovery from failures, and more efficient management of infrastructure, aligning with DevOps' goal of reducing manual intervention.

3. Devops Lifecycle



The **DevOps lifecycle** is a continuous process that involves several stages to automate and streamline the development, testing, deployment, and monitoring of software. The lifecycle focuses on collaboration, automation, and constant feedback to improve software delivery speed, quality, and reliability. Here are the main stages of the DevOps lifecycle:

1. Planning

In this stage, development teams and operations teams collaborate to plan the features, tasks, and overall roadmap for the software. Agile methodologies are often used to break down work into manageable sprints, and both teams ensure they align on requirements, timelines, and priorities. Continuous feedback from stakeholders helps to refine the planning process and adjust goals as needed.

2. Development

During the development stage, developers write the code, implement new features, and fix bugs. DevOps emphasizes the use of version control systems like Git to track code changes and ensure collaboration across teams. Developers use continuous integration (CI) practices to merge their code changes regularly, allowing for quicker detection of issues and improving the overall quality of the codebase.

3. Building

After code development, the building stage involves compiling the code into executable files or artifacts. Automation tools are used to streamline the build process, ensuring that the code can be compiled in a consistent manner every time. This is usually integrated with CI pipelines to run automated tests during the build process and catch any issues early, preventing faulty code from being deployed.

4. Testing

The testing phase is crucial for ensuring that the software works as expected and meets quality standards. DevOps encourages continuous testing by automating unit tests, integration tests, and acceptance tests. Automated tests are run as part of the CI pipeline to identify issues early in the development process. This reduces manual intervention and accelerates the testing process, ensuring that the code is of high quality before deployment.

5. Release

After testing, the software is ready to be released. Continuous Delivery (CD) tools automate the process of releasing code to various environments, such as staging or production. This ensures that code is always in a deployable state, and software can be released at any time. Release management tools help to track and monitor the deployment process, reducing the chances of errors during the release phase.

6. Deploy

Deployment involves moving the code into the production environment where end-users can access it. DevOps practices focus on automating deployment processes to ensure that code is deployed consistently and reliably. Infrastructure as Code (IaC) is often used to manage and provision production environments, ensuring that infrastructure is consistent and scalable. The deployment process is typically automated using CI/CD pipelines, enabling faster and more frequent releases.

7. Operate

Once the software is live, it enters the operations phase, where it is actively monitored for performance, uptime, and user experience. Teams use monitoring and logging tools to track application performance, system health, and detect any potential issues. This phase ensures that the software is running smoothly in production, and operational teams are ready to address any incidents, downtime, or scaling needs.

8. Monitor

Continuous monitoring plays a key role in the DevOps lifecycle. Monitoring tools help track application and system performance, logging errors, and gathering feedback from users. Insights from this data help teams identify potential problems, improve performance, and ensure high availability. Monitoring tools also provide feedback to developers and operations teams, enabling them to make data-driven decisions and continuously improve the application and infrastructure.

Feedback Loop

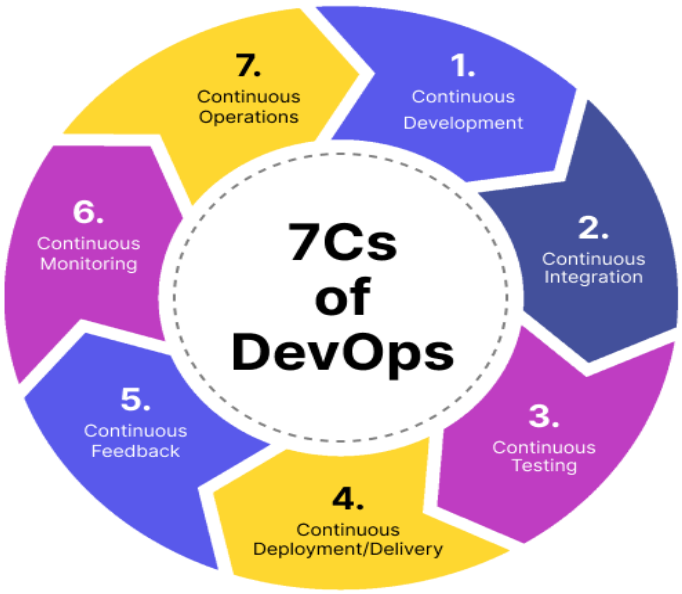
Throughout the entire DevOps lifecycle, continuous feedback is essential. Data from monitoring and logging, as well as feedback from users, operations, and quality assurance teams, helps guide the development of new features, bug fixes, and optimizations. This feedback is incorporated into the planning and development phases, creating a loop that drives constant improvement in the software and processes.

4. Devops delivery pipeline

The **DevOps Delivery Pipeline** is a set of automated processes and stages that facilitate the continuous delivery (CD) of software from development to production. It integrates various DevOps practices such as continuous integration (CI), continuous testing, and continuous deployment, enabling teams to deliver software in an efficient, consistent, and reliable manner. The pipeline automates the entire process, reducing manual effort, minimizing errors, and accelerating the software delivery lifecycle.

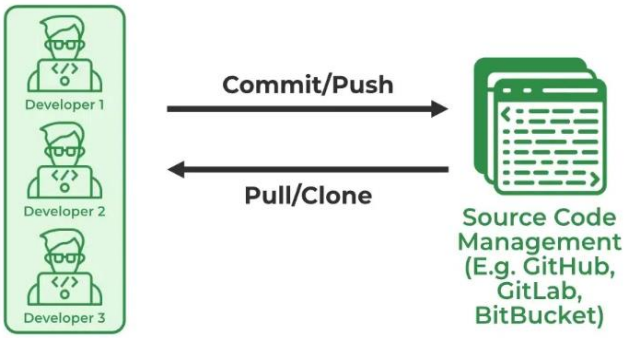
7 Cs of DevOps

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.



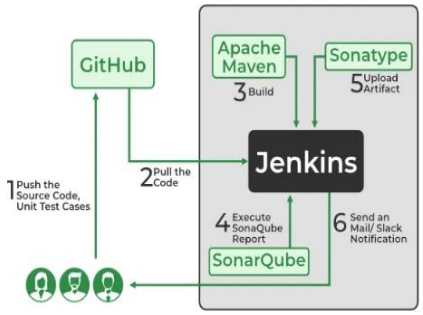
2. Continuous Integration

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

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

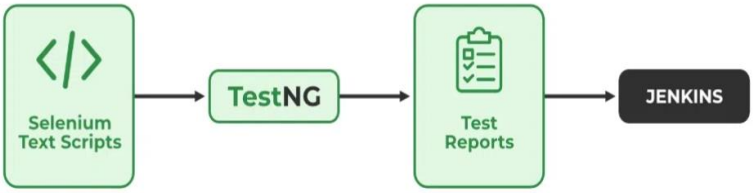
The stages mentioned above are the flow of Continuous Integration and we can use any of the tools that suit our requirement in each stage and of the most popular tools are **GitHub for source code management(SCM)** when the developer develops the code on his local machine he pushes it to the remote repository which is GitHub from here who is having the access can Pull, clone and can make required changes to the code. 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.



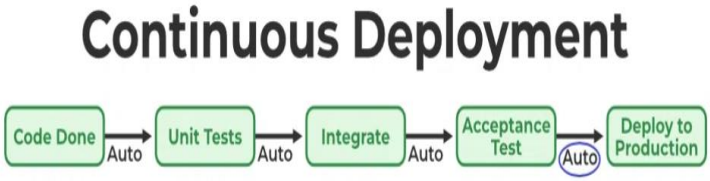
3. Continuous Testing

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 benefit.



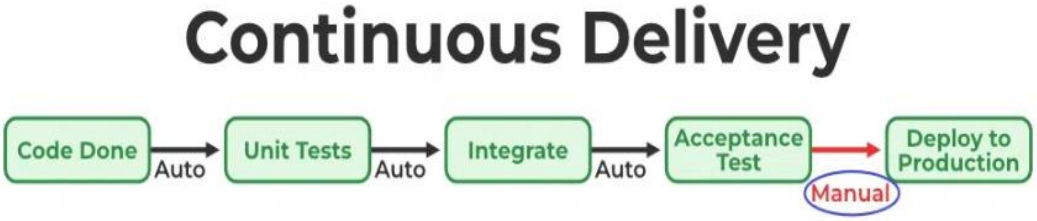
4.Continuous Deployment/ Continuous Delivery

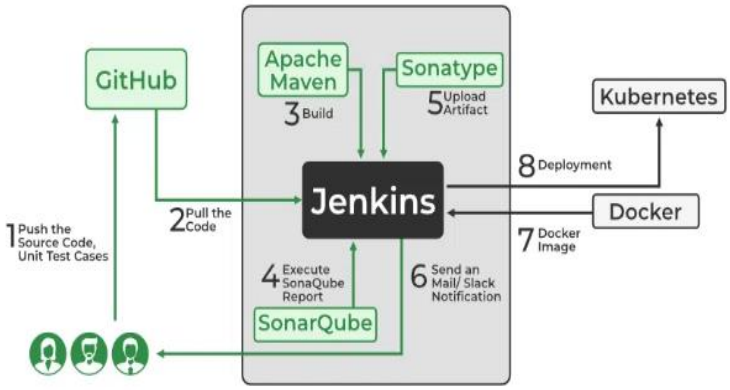
Continuous Deployment: Continuous Deployment is the process of automatically deploying an application into the production environment when it has completed testing and the build stages. Here, we'll automate everything from obtaining the application's source code to deploying it.



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.

5. Devops Ecosystem and Tools

The **DevOps Ecosystem** refers to the collection of tools, practices, and technologies that work together to support and automate the entire software development and delivery lifecycle. These tools are designed to help streamline development, continuous integration, testing, deployment, monitoring, and collaboration across development and operations teams. The goal is to enhance the speed, reliability, and quality of software delivery.

Below are key components of the **DevOps Ecosystem** and the tools commonly used in each stage:

1. Version Control

Version control systems help teams track changes in code, collaborate effectively, and maintain a history of modifications. They are critical for managing source code and ensuring collaboration across teams.

- **Popular Tools**:
  - **Git**: A distributed version control system used for tracking changes in source code.
  - **GitHub/GitLab/Bitbucket**: Hosting services for Git repositories, offering additional features like issue tracking, code reviews, and CI/CD integration.

2. Continuous Integration (CI) and Continuous Delivery (CD)

CI/CD tools automate the process of integrating code changes, running tests, and deploying software. CI focuses on automating the build and testing processes, while CD automates the deployment of code to production or staging environments.

- **Popular Tools**:
  - **Jenkins**: A widely-used automation server for building, testing, and deploying code.
  - **Travis CI**: A cloud-based CI tool for building and testing code, integrated with GitHub.
  - **CircleCI**: A cloud-native CI/CD tool that automates testing and deployment pipelines.
  - **GitLab CI/CD**: A built-in CI/CD solution within GitLab for automating testing and deployments.

3. Configuration Management and Infrastructure as Code (IaC)

These tools automate the configuration and provisioning of infrastructure. Infrastructure as Code (IaC) enables teams to define infrastructure using code, which makes it reproducible, scalable, and easier to manage.

- **Popular Tools**:
  - **Ansible**: A configuration management tool for automating server configuration and software deployment.
  - **Chef**: A configuration management tool that allows infrastructure to be defined as code.
  - **Puppet**: Used for automating the provisioning and management of servers.
  - **Terraform**: A popular IaC tool for provisioning and managing cloud infrastructure across multiple cloud providers.

4. Containerization and Orchestration

Containers package an application and its dependencies into a single unit, enabling portability across environments. Orchestration tools automate the deployment, scaling, and management of containerized applications.

- **Popular Tools**:
  - **Docker**: A platform for building, shipping, and running applications in containers.
  - **Kubernetes**: A container orchestration platform for automating the deployment, scaling, and management of containerized applications.
  - **Docker Compose**: A tool for defining and running multi-container Docker applications.
  - **OpenShift**: A Kubernetes-based container platform that provides enhanced security and scalability.

5. Monitoring and Logging

Monitoring and logging tools collect data from applications and infrastructure to provide insights into system performance, health, and user behavior. They enable proactive issue detection and help in troubleshooting.

- **Popular Tools**:
  - **Prometheus**: A monitoring and alerting toolkit designed for reliability and scalability.
  - **Grafana**: An open-source analytics platform that integrates with Prometheus to visualize and analyze monitoring data.
  - **ELK Stack (Elasticsearch, Logstash, Kibana)**: A powerful set of tools for searching, analyzing, and visualizing logs in real time.
  - **Splunk**: A platform for searching, analyzing, and visualizing machine-generated data (logs) in real time.
  - **Datadog**: A monitoring and analytics tool for cloud-scale applications, providing real-time observability.

6. Collaboration and Communication

DevOps requires strong communication and collaboration between development, operations, and other teams. Tools for collaboration ensure that everyone is on the same page and can quickly respond to issues.

- **Popular Tools**:
  - **Slack**: A messaging platform for teams that integrates with various DevOps tools and provides real-time communication.
  - **Microsoft Teams**: A collaboration platform that offers chat, video meetings, file sharing, and integrations with DevOps tools.
  - **Jira**: A project management tool used for issue tracking, agile planning, and team collaboration.
  - **Confluence**: A knowledge-sharing tool often used alongside Jira to document processes, guidelines, and best practices.

7. Security (DevSecOps)

DevSecOps integrates security practices into the DevOps pipeline. Security tools help automate vulnerability scanning, code analysis, and security testing to ensure that security is baked into the development process.

- **Popular Tools**:
  - **SonarQube**: A tool for static code analysis that helps detect bugs, vulnerabilities, and code smells in the codebase.
  - **OWASP ZAP**: A penetration testing tool that identifies security vulnerabilities in web applications.
  - **Aqua Security**: A security platform for securing containerized applications and cloud-native environments.
  - **Snyk**: A tool for finding and fixing vulnerabilities in open-source libraries, containers, and infrastructure.

8. Automated Testing

Automated testing is a key part of the DevOps process, allowing teams to ensure code quality and functionality early in the development cycle. It involves unit tests, integration tests, and acceptance tests.

- **Popular Tools**:
  - **Selenium**: A popular tool for automating web browsers to test web applications.
  - **JUnit**: A widely-used framework for unit testing in Java applications.
  - **TestNG**: A testing framework designed for test configuration and running tests in parallel.
  - **Cypress**: A tool for end-to-end testing of web applications, focusing on developer productivity and ease of use.

9. Artifact Repositories

Artifact repositories store build artifacts such as libraries, dependencies, and executables, which can be used during the build, test, and deployment stages.

- **Popular Tools**:
  - **Nexus Repository**: A repository manager for managing software artifacts and dependencies.
  - **Artifactory**: A universal artifact repository that supports Maven, Docker, and other technologies.
  - **AWS CodeArtifact**: A fully managed artifact repository service that integrates with AWS services.

Popular DevOps Tools:

Category	Tool Examples	Functionality
Planning & Tracking	Jira, Asana, Trello	Project management, task tracking, issue management
Development	Git, Visual Studio Code, Eclipse	Code development, version control, code editing
Testing	Selenium, JUnit, Pytest	Automated testing, test automation frameworks
Deployment	Terraform, Ansible, Jenkins	Infrastructure provisioning, application deployment, automation
Monitoring	Prometheus, Grafana, ELK Stack	Real-time monitoring, performance analysis, logging
CI/CD	Jenkins, CircleCI, GitHub Actions	Automated build, test, and deployment pipelines
Containerization	Docker	Packaging applications into containers
Container Orchestration	Kubernetes	Automating deployment, scaling, and management of containers
Infrastructure as Code (IaC)	Terraform, Ansible	Defining and managing infrastructure as code

6. The Agile wheel of wheels.

The **Agile Wheel of Wheels** is a visual metaphor that represents the various interconnected elements that drive Agile software development. It emphasizes that Agile is not a single linear process, but rather a continuous cycle of feedback, improvement, and collaboration. The wheel's "spokes" represent the principles, practices, and values that support Agile methods, and the "wheels" themselves represent different aspects of Agile that work together to form a cohesive, iterative development process.

The **Agile Wheel of Wheels** typically includes the following core components:

1. Agile Values

Agile is based on four key values:

- **Individuals and interactions over processes and tools**: Focus on people working together effectively rather than rigid processes or tools.
- **Working software over comprehensive documentation**: Prioritize delivering functional software that provides value.
- **Customer collaboration over contract negotiation**: Engage with customers throughout the development process to ensure their needs are met.
- **Responding to change over following a plan**: Be adaptable to changes in requirements and external conditions.

2. Agile Principles

Agile development is governed by 12 principles that guide teams toward delivering value more efficiently. Some of these principles include:

- Delivering working software frequently, with a preference for shorter timescales.
- Welcoming changing requirements, even late in development.
- Maintaining a sustainable pace of work.
- Fostering close, daily cooperation between business stakeholders and developers.

3. Collaboration

Collaboration is key in Agile. The "wheel" emphasizes teamwork between developers, business stakeholders, customers, and even between different teams. Agile relies on constant feedback and open communication channels to ensure that the software aligns with user needs and business goals.

4. Iteration and Increment

Agile processes are iterative and incremental. Development is broken down into small, manageable units (called **iterations** or **sprints**) that typically last 1-4 weeks. Each iteration results in a working increment of software. The short cycles allow for continuous improvement, feedback, and adaptation.

5. Continuous Improvement

Agile practices encourage continuous improvement of processes, workflows, and the product itself. Retrospectives, which are held at the end of each iteration, provide a forum for teams to reflect on their performance, discuss challenges, and plan improvements for the next cycle.

6. Customer-Centric Development

At the heart of Agile is the focus on delivering value to customers. The Agile wheel stresses continuous customer involvement and feedback, ensuring that the software being developed truly meets user needs and delivers meaningful outcomes.

7. Simplicity

Agile encourages **simplicity** in design and development. By focusing on the most essential features and avoiding overcomplication, Agile aims to deliver the maximum amount of value with the least amount of work. Simplicity also applies to decision-making, prioritizing tasks, and maintaining flexibility.

8. Self-Organizing Teams

Agile promotes **self-organizing teams** that have the autonomy to make decisions about how to best complete their work. This empowerment fosters creativity, accountability, and faster decision-making, which leads to better outcomes and higher morale.

9. Feedback Loops

Agile thrives on feedback loops, where progress is regularly assessed, and corrections are made in response to new insights. This includes both technical feedback (e.g., from automated tests) and business feedback (e.g., from stakeholders or customers). These loops ensure that the project stays aligned with its goals.

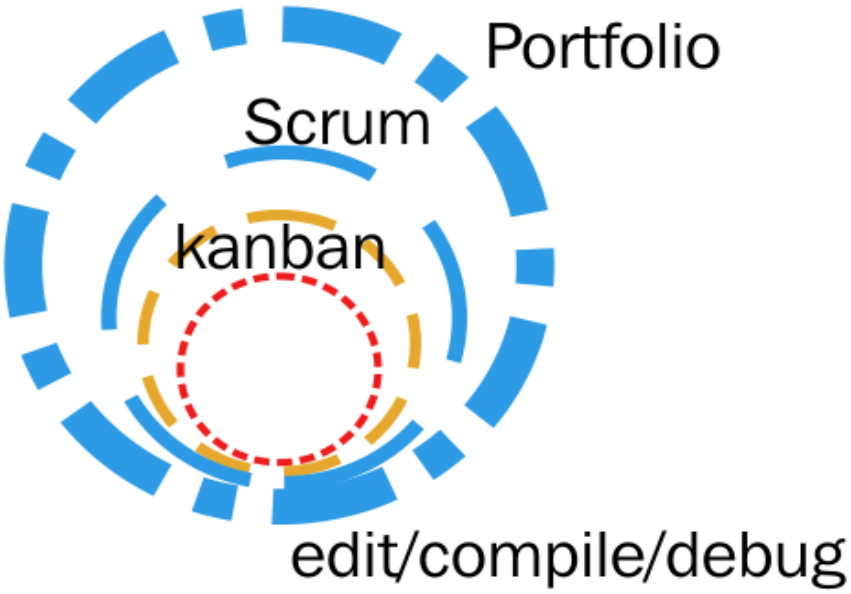
10. Sustainability

Agile methodologies emphasize sustainable development practices. Teams strive for a pace that can be maintained over the long term without burnout. The idea is to create a sustainable work environment where developers can keep producing quality work over extended periods.

The Agile Wheel of Wheels represents these interconnected elements in a circular, continuous manner, where each part supports and enhances the others. By iterating on all these aspects—values, principles, collaboration, and feedback—Agile teams are able to adapt, improve, and deliver value in an ever-changing environment. Each "wheel" turns independently, but they all drive toward the same goal: creating better software, faster.

The Agile wheel of wheels:-

There are several different cycles in Agile development, from the Portfolio level through to the Scrum and Kanban cycles and down to the **Continuous Integration** (**CI**) cycle. The emphasis on which cadence work happens in is a bit different depending on which Agile framework you are working with. Kanban emphasizes the 24-hour cycle, which is popular in operations teams. Scrum cycles can be two to four weeks and are often used by development teams using the Scrum Agile process. Longer cycles are also common and are called **Program Increments** (**PI**), which span several Scrum Sprint cycles, in the **Scaled Agile Framework** (**SAFe®**):



The Agile wheel of wheels

DevOps must be able to support all these cycles. This is quite natural given the central theme of DevOps: cooperation between disciplines in an Agile organization.

The most obvious and measurably concrete benefits of DevOps occur in the shorter cycles, which in turn makes the longer cycles more efficient.

Here are some examples of when DevOps can benefit Agile cycles:

- Deployment systems, maintained by DevOps engineers, make the deliveries at the end of Scrum cycles faster and more efficient. These can take place with a periodicity of two to four weeks.
- In organizations where deployments are done mostly by hand, the time to deploy can be several days. Organizations that have these inefficient deployment processes will benefit greatly from a DevOps mindset.
- The Kanban cycle is 24 hours, and it's therefore obvious that the deployment cycle needs to be much faster than that if we are to succeed with Kanban.
- A well-designed DevOps CD pipeline can deploy code from being committed to the code repository to production in a matter of minutes, depending on the size of the change.