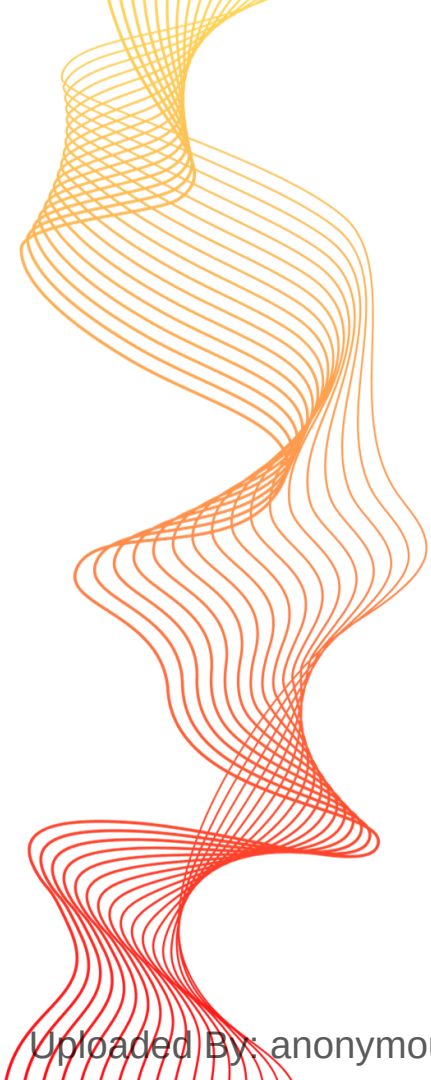
DevOps (Development and Operations) Teams

This presentation is about the role and importance of DevOps teams in software development projects from the testing perspective.

Introduction

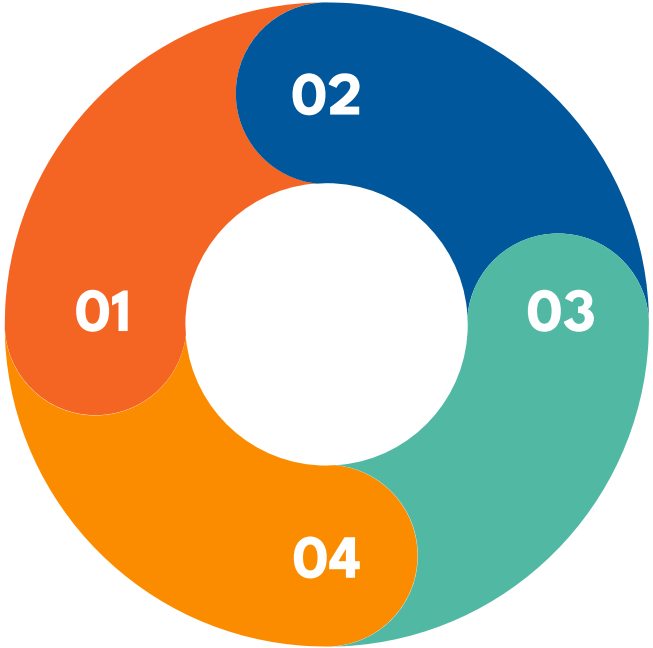
- DevOps is a set of practices that combines software development (Dev) and IT operations (Ops) to shorten the systems development life cycle and provide continuous delivery with high software quality.
- In contrast to traditional software development, development and operations teams work in silos. This can lead to delays, communication problems, and quality issues.
- DevOps breaks down these silos and fosters collaboration between the two teams



Introduction to DevOps

DevOps teams bridge the gap between software development and IT operations

The goal is to streamline the software development lifecycle

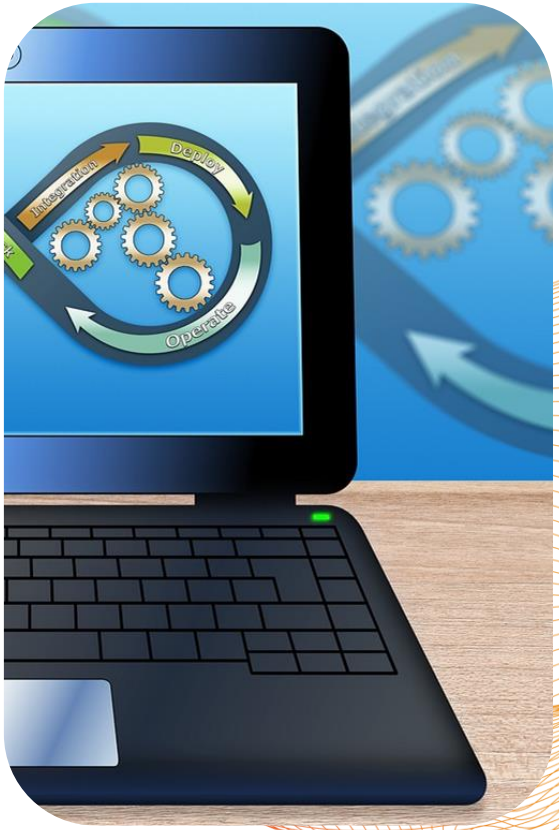


Improved collaboration between teams

Automate processes for efficient and reliable software delivery

Importance of DevOps

- Efficient software development and deployment
- Faster time to market for new features
- Improved quality and stability of software
- Enhanced customer satisfaction



Example: Developing E-Commerce Web App

-

Example Scenario: Imagine a small e-commerce company that wants to develop and deploy a new feature on its website that allows customers to track their orders in real-time.

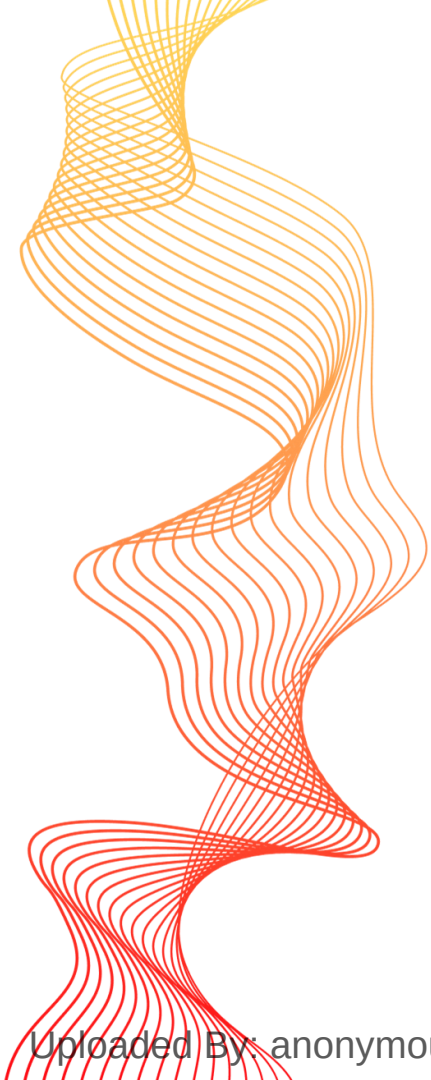
Collaboration

- DevOps teams foster collaboration between developers and IT operations.
- In this case, developers work on building the order tracking feature, while the DevOps team ensures that it can be seamlessly deployed and maintained in the production environment.

Continuous Integration (CI)

- The DevOps team sets up a CI system where code changes are automatically built, tested, and integrated into a shared repository.

- Whenever a developer makes a change to the order tracking feature, the CI system automatically checks if the code builds without errors and runs tests to catch any issues early.



Continuous Integration CI Process

- An agile software development team is working on a new feature for their product.
- The feature is broken down into small tasks, which are assigned to individual developers.
- Each developer works on their assigned tasks and commits their changes to the team's shared repository regularly.
- As soon as a developer commits their changes, a CI pipeline is triggered.

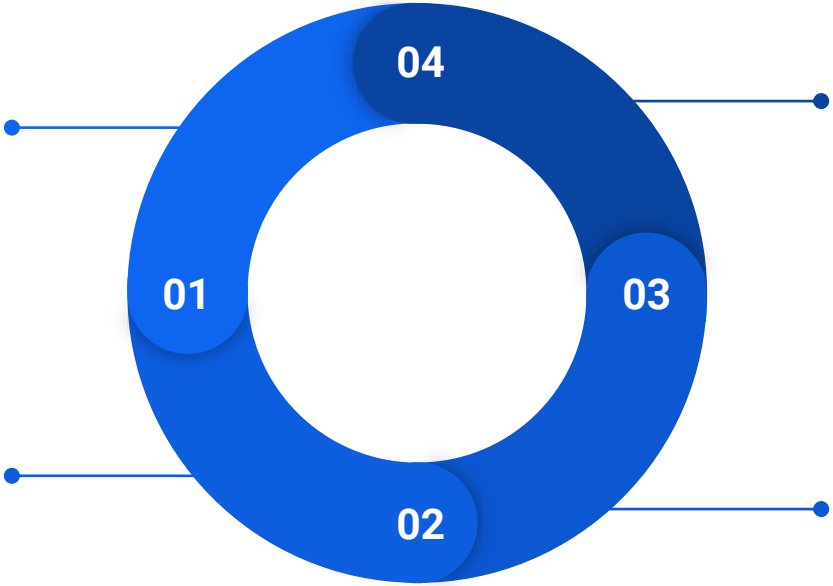
Example CI Pipeline

Building the code

The pipeline builds the code from the repository into a deployable artifact.

Running unit tests

The pipeline runs unit tests on the code to ensure that it is working as expected



Deploying the code to a test environment

The pipeline deploys the code to a test environment so that it can be tested by the team and other stakeholders.

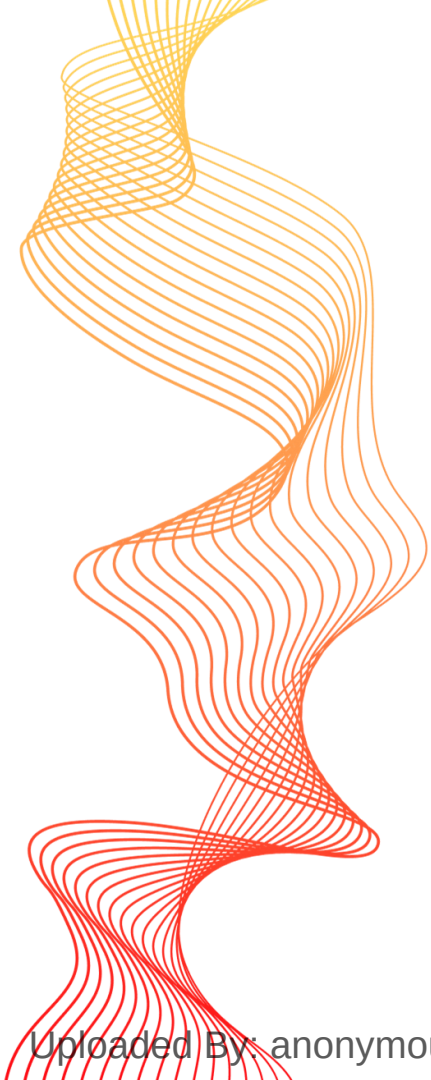
Running integration tests

The pipeline runs integration tests on the code to ensure that it works well with the other components of the system.

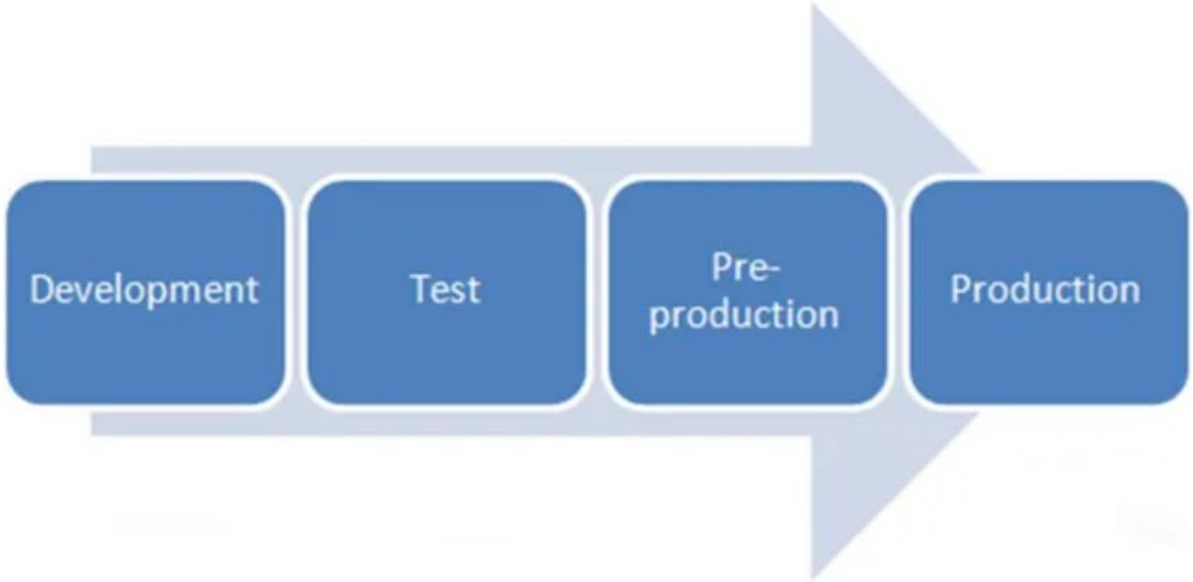
Continuous Delivery (CD)

- The DevOps team configures a CD pipeline to automate the deployment process.

- As soon as code changes pass CI checks, the CD pipeline deploys the feature to a staging environment for further testing and validation.



What is Staging Environment?



A staging environment is a copy of the production environment where software is tested before it is deployed to production. The staging environment is as close a replica of the production environment as possible, so that testers can be confident that the software will behave as expected in production.

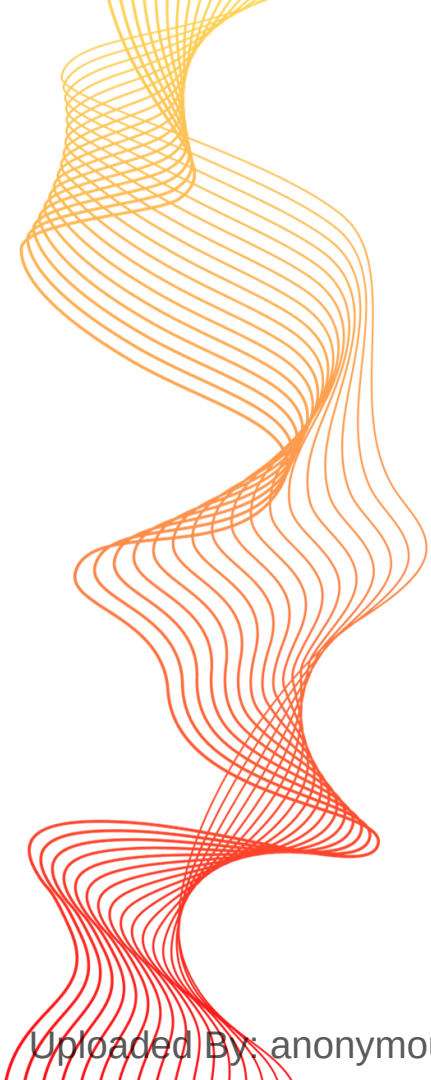
Types of tests applied at Staging Env.

- **Functional testing:** This type of testing ensures that the software meets all of its functional requirements.
- **Integration testing:** This type of testing ensures that the different components of the software work together correctly.
- **Performance testing:** This type of testing ensures that the software can handle the expected load and perform as expected under stress.
- **Security testing:** This type of testing identifies and fixes any security vulnerabilities in the software.

Infrastructure as a Code (IaC)

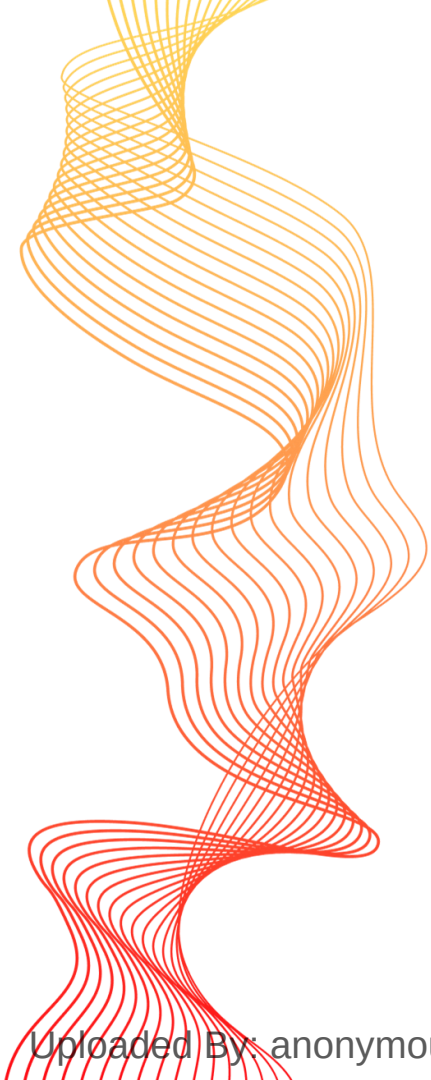
- DevOps engineers use infrastructure as code to define and manage the necessary server and networking resources for the application.

- This ensures that the development, staging, and production environments are consistent and can be recreated easily.



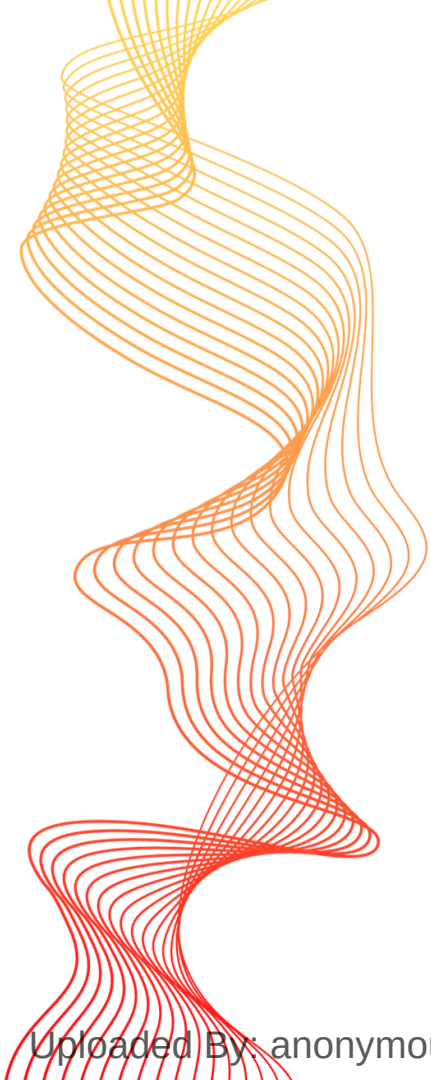
Automated Testing:

Automated tests are created and integrated into the CI/CD pipeline to verify that the order tracking feature functions correctly and doesn't introduce regressions into the existing application.



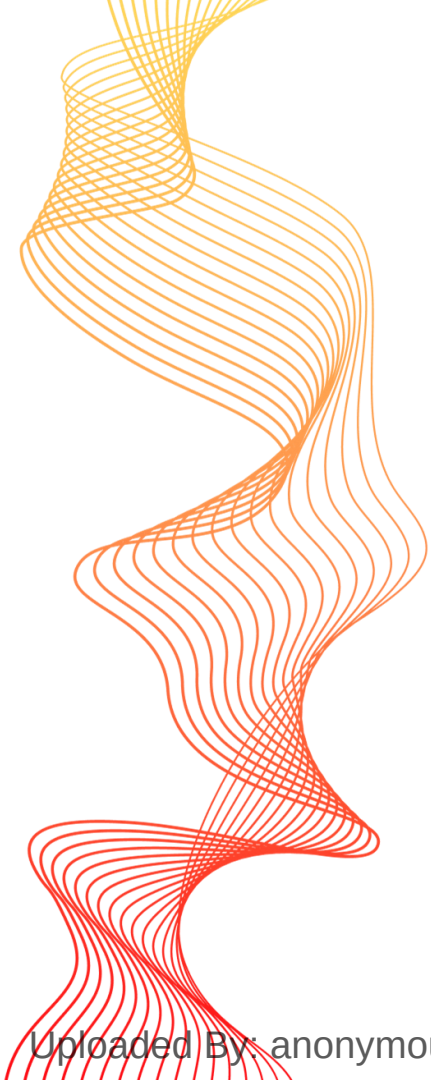
Monitoring and Logging

The DevOps team configures monitoring tools to track the performance of the feature in real-time. They set up logging to capture any errors or issues that may occur during usage.

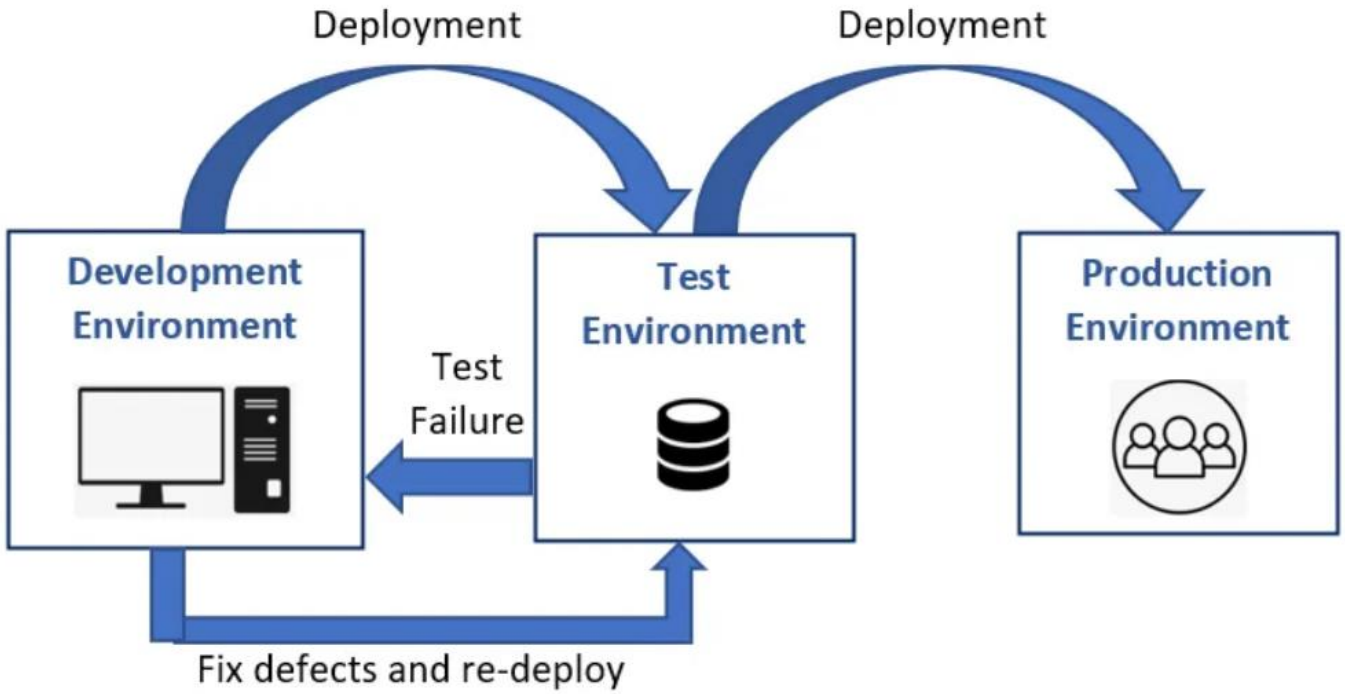


Deployment to Production

Once the feature passes all tests in the staging environment and receives approval from stakeholders, the DevOps team automates the deployment to the production environment. This ensures a smooth and consistent release of the new functionality to customers.



Testing Environments



Three main software environments

Overall CI/CD Pipeline Process

Here is a specific example of how CD might be used in a DevOps environment to deploy a new feature to production:

1. A developer commits changes to the code repository for the new feature.
2. The CI/CD pipeline is triggered and builds the code, runs unit tests and integration tests, and deploys the code to a staging environment.
3. The QA team tests the code in the staging environment and verifies that it is working as expected.
4. Once the QA team is satisfied with the code, they notify the operations team.
5. The operations team deploys the code to production.

Some example technologies and tools that are used in continuous integration and continuous delivery (CI/CD) in DevOps:

- **Version control systems:** Version control systems such as Git, Mercurial, and SVN are used to track changes to code and manage multiple versions of code.
- **Build tools:** Build tools such as Maven, Gradle, and Bazel are used to automate the process of building code into a deployable artifact.
- **Testing tools:** Testing tools such as JUnit, TestNG, and Cypress are used to automate the testing of code.
- **Deployment tools:** Deployment tools such as Ansible, Terraform, and Chef are used to automate the deployment of code to production environments.
- **CI/CD servers:** CI/CD servers such as Jenkins, CircleCI, and Travis CI are used to automate the CI/CD pipeline.