Softura

Best Practices for Implementing

CI/CD Pipelines

Introduction

According to Fortune Business Insights, the global market size for DevOps is expected to grow at a compounded annual growth rate of 19.1% from 2018 to 2026. By 2026, the market size for DevOps is expected to cross US$14.9 billion. This growth is largely attributed to the adoption of cloud as companies migrate their legacy workloads to the cloud and develop cloud native applications using Microservices architecture.

Capital One, has a 'cloud-first' policy which means Ideas for improving customer experience are quickly converted into digital products (software applications and services). All the applications are designed for the cloud using Microservices Architecture. Functionalities are built as loosely coupled services using DevOps on the cloud approach, enabled by AWS services. The company has reported that they have significantly reduced the time required to provision application infrastructure by using AWS services and DevOps. There are numerous such case studies where DevOps and more specifically, CI/CD pipelines have enabled collaboration and automation so that developers can focus on building functionalities without having to worry about infrastructure requirements.

Continuous Integration and Continuous Deployment (CI/CD) pipeline refers to the process followed to incrementally release new software features and functionalities. CI/CD pipelines are an important component of DevOps that aims to improve delivery through automation. While the steps in the CI/CD pipeline can be executed manually, automation ensures that the development team is able to deploy quality code into production faster.

Dr. Mik Kersten, Best-Selling Author and CEO at Tasktop, says, "CI/CD is about creating a fast feedback loop that enables teams to deliver value to their customers continuously, by automating the build, test, and deployment process and integrating code changes continuously."

In this blog post, we will look at the best practices for implementing CI/CD pipelines and conclude by listing the benefits of automating CI/CD pipelines.

Managing Dependencies

A dependency refers to a component or resource that is required to build, test, or deploy a software application or system. This can include external libraries or frameworks, build tools, testing tools, and other software or hardware components that are necessary for the development and deployment process. Here are a few best practices for managing dependencies:

- **Use a dependency management tool:** There are several tools available that can help manage dependencies in a CI/CD environment. For example, you can use tools like Maven, Gradle, or npm to manage dependencies for Java, Android, and JavaScript projects, respectively. These tools allow you to specify the dependencies for your project in a configuration file, and they will automatically download and manage the required dependencies for you.

- **Use a package registry:** A package registry is a centralized repository that stores packages that can be used as dependencies in your project. By using a package registry, you can easily manage the versions of your dependencies and ensure that the correct versions are being used in your project. Some examples of package registries include Maven Central, npmjs.com, and PyPI (Python Package Index).

- **Use containers:** Containerization technology such as Docker allows you to package your application and its dependencies into a single container image, which can be easily deployed and run in any environment. This can help ensure that your application always has the correct dependencies, regardless of the environment in which it is being run.

- **Use a dependency management service:** Some CI/CD platforms offer built-in dependency management services that allow you to manage your dependencies directly from the platform. For example, AWS CodePipeline offers a service called AWS CodeArtifact that can be used to store and manage dependencies for your project.

Leveraging Containerization

Containerization is a method of packaging software applications so that they can be run in multiple different environments without requiring any changes to the underlying infrastructure. This is achieved by packaging the application, along with all its dependencies, into a container. The container is a lightweight, standalone, executable package that includes everything the application needs to run, including the application code, system tools, libraries, and runtime. Jez Humble, Site Reliability Engineer at Google, says, "Containers provide a lightweight, portable way to package and distribute software, making it easier to deploy and run applications consistently across different environments."

Let us look at a case study to understand how companies are effectively leveraging containerization. Workflow management software provider, Weever Apps, uses Docker containers to empower developers to write code locally and deploy in the cloud without code changes. This approach saves time and helps the company accelerate feature releases. Here are some of the best practices for containerization:

- **Use versioned container images:** By using versioned container images, you can easily track and roll back changes to your application if necessary. This can be especially useful in a CI/CD environment where new versions of your application are being deployed regularly.
- **Use small and lightweight container images:** Smaller container images will be faster to build and deploy, which can help improve the efficiency of your CI/CD pipeline. You can use tools like multi-stage builds and image squashing to reduce the size of your container images.
- **Use a container registry:** A container registry is a centralized repository for storing and managing container images. By using a container registry, you can easily store and retrieve your container images as part of your CI/CD pipeline. Some examples of container registries include Docker Hub, Google Container Registry, and Amazon Elastic Container Registry (ECR).

- **Use a container orchestration platform:** A container orchestration platform, such as Kubernetes or AWS ECS, allows you to automate the deployment and management of your containerized applications. By using a container orchestration platform, you can ensure that your applications are deployed and scaled in a consistent and reliable manner.

- **Use security scanning:** Security scanning involves analyzing your container images for vulnerabilities and security issues. There are several tools available, such asanchore, Aqua Security, and Twistlock, that can be used to perform container scanning as part of your CI/CD pipeline.

Testing in Multiple Environments

In a CI/CD pipeline, testing plays a critical role in ensuring the quality and reliability of the software being developed. There are various types of tests that can be included in a CI/CD pipeline, including:

- **Unit tests :** Unit tests are small, isolated tests that verify the functionality of individual units of code.

- **Integration tests :** Integration tests test the integration of different units of code, and they ensure that these units are working together as expected.

- **End-to-end tests :** End-to-end tests, also known as acceptance tests, test the entire system from end to end to ensure that it is working as expected.

Key things to implement to test CI/CD Pipelines Effectively

- **Test Automation Framework:** A test automation framework allows you to automate the execution of your tests and makes it easy to run tests in multiple environments. There are several test automation frameworks available, such as Selenium, Appium, and Cypress, that can be used to automate tests for web, mobile, and API testing.

- **Cloud-based Testing Platform:** Cloud-based testing platforms, such as BrowserStack or Sauce Labs, allow you to run tests on a wide range of browser and device configurations without the need to maintain your own testing infrastructure. This can make it easier to test in multiple environments and reduces the overhead of maintaining your own test infrastructure.

- **Version Control System:** By storing your tests in a version control system such as Git, you can easily track changes to your tests and ensure that the correct versions are being run in different environments. This can be especially useful if you have multiple teams working on different environments and need to ensure that the correct tests are being run in each environment.

- **Environment Variables:** Environment variables allow you to configure your tests to run in different environments by setting variables that are specific to each environment. For example, you can use environment variables to set different URLs for testing in different environments, or to set different credentials for accessing different environments.

Integrating Security Controls

Integrating security controls into a CI/CD pipeline involves incorporating security checks and controls into the various stages of the pipeline to ensure that security is an integral part of the development and deployment process. Let us look at some of the best practices to build secure software using DevOps approach:

- **Static Code Analysis:** Static code analysis involves analyzing the source code of your application for vulnerabilities and security issues. There are several tools available, such as Fortify, Checkmarx, and SonarQube, that can be used to perform static code analysis as part of your CI/CD pipeline.

- **Dynamic Testing:** Dynamic testing involves executing your application and testing it for vulnerabilities and security issues. There are several tools available, such as ZAP, Burp Suite, and Nessus, that can be used to perform dynamic testing as part of your CI/CD pipeline.

- **Container Scanning:** Container scanning involves analyzing the container images used in your application for vulnerabilities and security issues. There are several tools available, such as Anchore, Aqua Security, and Twistlock, that can be used to perform container scanning as part of your CI/CD pipeline.

- **Secrets Management:** Secrets management involves securely storing and managing secrets, such as passwords and API keys, that are used in your application. There are several tools available, such as Hashicorp Vault, AWS Secrets Manager, and Azure Key Vault, that can be used to manage secrets as part of your CI/CD pipeline.

Conclusion - Role of Automation in DevOps

Automation plays a key role in a DevOps workflow, as it allows teams to automate various tasks and processes to improve efficiency, reduce errors, and speed up delivery. In conclusion, here are a list of tasks that can be automated in a DevOps workflow:

- **Build and deployment:** Automation tools, such as Jenkins, can be used to automate the build and deployment process, allowing teams to automatically build, test, and deploy code changes to production.
- **Testing:** Automation tools can be used to automate the testing process, allowing teams to automatically run unit tests, integration tests, and performance tests as part of the build process.
- **Infrastructure provisioning and configuration:** Tools such as Terraform and Ansible can be used to automate the provisioning and configuration of infrastructure, such as servers, networks, and storage.
- **Monitoring and Alerting:** Automation tools can be used to automate the monitoring and alerting process, allowing teams to set up alerts for important events and triggers, such as application errors or infrastructure issues.

Benefits of DevOps Approach to Software Development:

> According to Donovan Brown, Partner Program Manager at Microsoft, "DevOps is about bringing development and operations together in order to build, test, and release software faster and more reliably."

In a research to find out why companies are adopting DevOps as part of their agile software development strategy, Boston Consulting Group reported the following benefits:

- Reduce defects by 70%
- Release new features more frequently
- Accelerate product to market by 2 times
- Decrease cost by 30% through virtualized infrastructure

So, how do CI/CD pipelines help companies realize the DevOps benefits?

Continuous Integration and Continuous Delivery (CI/CD) pipelines play a critical role in enabling organizations to adopt a DevOps approach to software development and delivery, helping them to deliver high-quality software faster and more reliably.

CI/CD pipelines help to automate the software development and deployment process, making it easier for developers to build, test, and deploy their code changes. This approach reduces the time it takes to get new code changes to users and improves the overall quality and stability of the software.

- What your company can do to get started with automating CI/CD pipelines? Manage dependencies
- Leverage containerization
- Test software in multiple environments
- Integrate security controls

This article is brought to you by Softura. For more information, visit www.softura.com and follow us on LinkedIn. https://www.linkedin.com/company/softura/

Contact Us at

(844) 791- 0545 | info@softura.com | www.softura.com

Our Offices

Farmington Hills, MI | Chicago, IL | Indianapolis, IN | Pittsburgh, PA | Atlanta, GA | Houston, TX
Tampa Bay, FL | Charlotte, DC | Chennai, IN | Bangalore, IN | Vadodara, IN | Toronto, CA