Name : Sakshi Kamble

PRN : 2020BTEIT00205

Course : Agile software tools and practices

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

14. Use docker Containers and Kubernetes for Demonstration of CI/CD.

**Aim**:

To use docker containers and kubernetes for demonstration of CI/CD

# Objective:

1. To study docker and Kubernetes
2. Demonstration of CI/CD pipeline using Docker Container

# What is Docker

Docker is an open-source containerization platform used for developing, deploying, and managing applications in lightweight virtualized environments called containers. It is mainly used as a software development platform for developing [distributed applications](https://phoenixnap.com/blog/securely-connect-distributed-apps) that work efficiently in different environments. By making the software system agnostic, developers don’t have to worry about compatibility issues. Packaging apps into isolated environments (containers) also makes it easier to develop, deploy, maintain, and use applications.

# What is kubernetes

[Kubernetes](https://kubernetes.io/) is an open source project that has become one of the most popular container orchestration tools around; it allows you to deploy and manage

multi-container applications at scale. While in practice Kubernetes is most often used with [Docker](https://www.infoworld.com/article/3204171/what-is-docker-the-spark-for-the-container-revolution.html), the most popular containerization platform, it can also work with any container system that conforms to the Open Container Initiative (OCI) standards for container image formats and runtimes. And because Kubernetes is open source, with relatively few restrictions on how it can be used, it can be used freely by anyone who wants to run containers, most anywhere they want to run them—on-premises, in the public cloud, or both.

# What is CI/CD?

Continuous Integration/Continuous Deployment (CI/CD) describes the key stages in an automated software development and deployment flow. This flow typically includes design, coding, testing, integration, delivery, validation and phased deployment activities before operation in a target environment.

Since CI/CD attempts to automate the flow from design to deployment, each flow is shaped by the underlying value chain. Feedback from each stage flows back to earlier stages – creating a loop of continuous improvement.

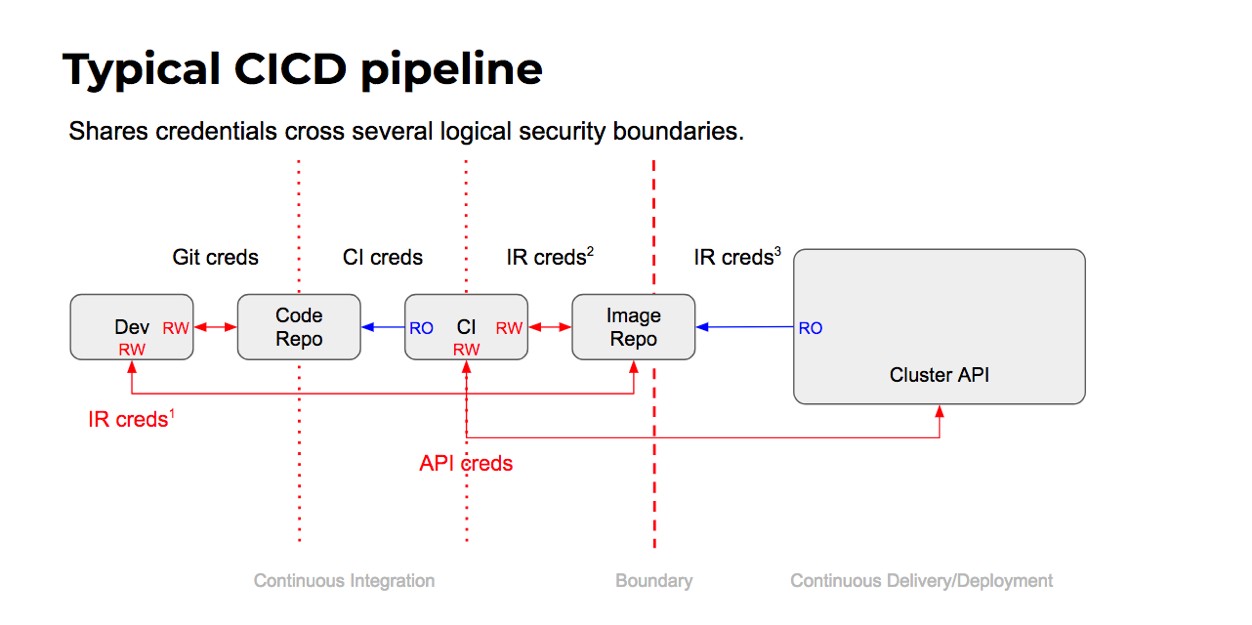
# Approaches to Continuous Delivery to Kubernetes

Even though there are a number of tools available, CI/CD vendors are still in the process of defining how a CI/CD Kubernetes pipeline can and should look like for engineers who create cloud native applications. At Weaveworks, we see two different paradigms emerging; one is a push type pipeline and the other is a pull pattern.

# Push Pipelines

Most of the CI/CD tools available today use a push-based model. What we mean by push-based is that code goes through the pipeline starting with the CI system and may continue its path through a series of encoded scripts or by using ‘kubectl’ by hand to push changes to the Kubernetes cluster.

The reason you don’t want to use your CI system as the deployment impetus or do it manually on the command line is because of the potential to expose credentials outside of your cluster. While it is possible to make those interactions secure in both your CI/CD scripts and manually on the command line, you are working outside the trust domain of your cluster. This is generally not good practice and is why CI systems can be known as attack vectors for production.



CI/CD tools work with a pipeline strategy and overtake the manual steps of getting the latest changes from the source code repository, compilation, test, verification, and deployment to the cluster. With Kubernetes, you need to extend CI/CD pipelines with a container registry, configuration manager, Helm, and multiple cluster environments, as shown in the image below.

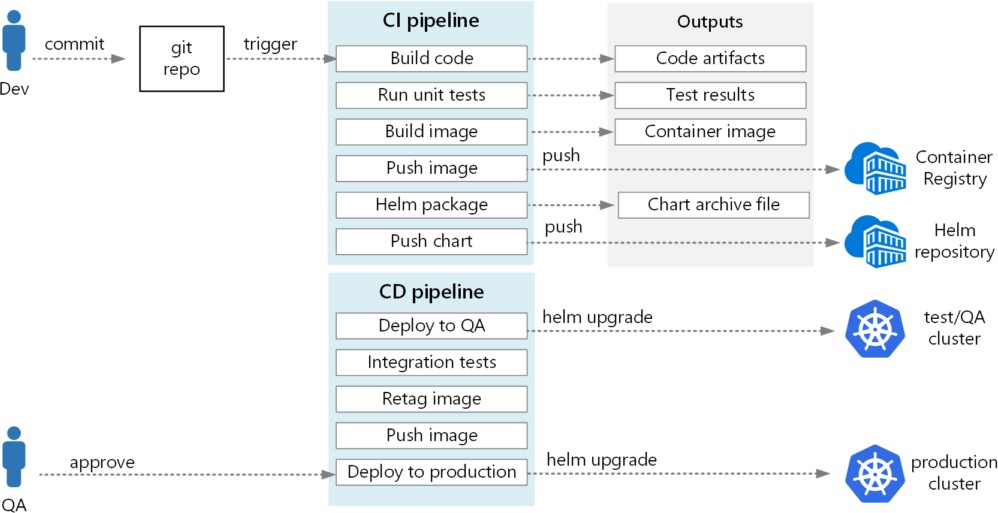


Figure 2: CI/CD pipeline with Kubernetes (Source: [Microsoft Azure](https://docs.microsoft.com/en-us/azure/architecture/microservices/ci-cd-kubernetes))

In this blog post, we will discuss the challenges as well as best practices for CI/CD pipelines for Kubernetes. We’ll also share a list of popular tools that you should not miss when building an effective CI/CD pipeline for Kubernetes.

Challenges of Kubernetes CI/CD Pipelines

Designing and using a CI/CD pipeline is not easy, due to the architecture of microservices and the restless nature of Kubernetes. So, if we would need to

summarize the challenges of a Kubernetes CI/CD pipeline, it would be in the following three categories:

* Automated testing: Automated testing is crucial when creating and deploying reliable applications in the cloud. You should test all of the developers’ commits thoroughly before moving to the following stages of the pipeline. This can be challenging while considering different cloud provider integrations and mocking Kubernetes API.
* Deployment and rollback: Kubernetes provides scalable and reliable deployment APIs so that it can release your applications to the cloud without downtime. Therefore, CD pipelines should also retain cloud-native deployment strategies, such as rolling upgrades, blue/green, or A/B testing. In addition, the pipeline should roll back to the “last working” version in case problems occur in the deployment. Therefore, designing an automated deployment and rollback strategy in your pipelines is critical.
* Scalability: When you use Kubernetes in production, it will be the home of tens—or even hundreds—of applications. In that case, the CI/CD tools you are using will be watching a large scale of source-code repositories and running multiple pipeline instances in parallel. Therefore, you need to design your CI/CD pipeline considering its scalability and allocate enough resources for its workers.

These challenges show that there is no silver bullet for designing and implementing a successful CI/CD pipeline for Kubernetes. However, there are widely accepted best practices, which we’ll discuss in the following section.

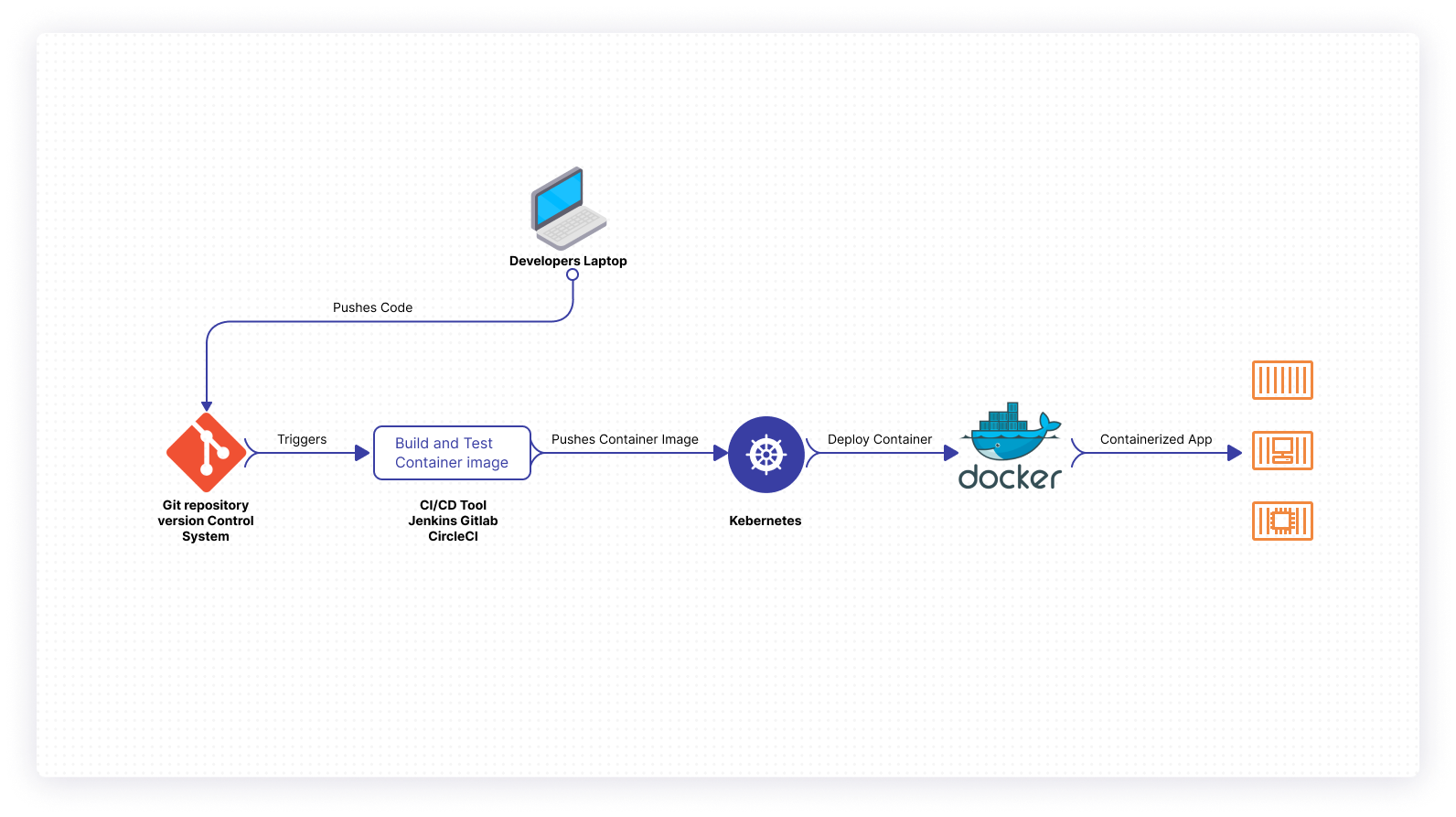
# Best Practices for Kubernetes CI/CD Pipelines

Here are three best practices you can follow for your Kubernetes CI/CD pipelines:

* GitOps: GitOps is one of the newest ways to manage infrastructure and cloud-native applications using the source version control system—namely, Git. In this practice, all application and pipeline configurations are kept in Git side-by-side with the application’s source code.
* Deployment strategies: Deployment strategies are the methods of release, upgrade, and rollback of the applications. Widely known strategies include blue/green, canary, and A/B testing. You need to choose a set of deployment strategies and include them in your CD pipelines, as they can help minimize downtime and the risk of failure.
* Overall observability: CI/CD pipelines are the bridges between your source code and Kubernetes clusters. Therefore, you need to monitor the status of pipeline stages and create alerts and notifications for the responsible people. Observability of the CI/CD systems is essential because when the pipeline stages fail, you will not be able to create new releases and bugfix patches.

# Kubernetes CI/CD pipeline example

Deploying a typical CI/CD pipeline to the Kubernetes cluster requires various components.



# Kubernetes CI/CD Pipeline

Version Control System – A code repository where code changes and updates are pushed from multiple developers. A CI/CD tool is usually triggered when developers push code changes to the version control system. Popular examples of version control systems are Git.

CI/CD Tool – An integration and test system that builds the docker image and runs a series of tests. It also pushes the built image to the Kubernetes cluster. Popular examples are Jenkins, Travis, or CircleCI.

Kubernetes Cluster – Kubernetes deploy docker containers for the software build verified by the CI/CD tool.

Docker – Docker containerizes the software application for seamless encapsulation and integration.

# Conclusion:

Implementing a CI/CD pipeline with Kubernetes automation capabilities has many benefits. Developers can easily patch updates, solve outage issues in case of an unforeseen traffic spike, and improve resource efficiency. With Kubernetes becoming more popular day by day, all the mature CI/CD tool vendors are developing new features to integrate with Kubernetes. When looking for a Kubernetes CI/CD pipeline tool, it is essential to determine factors such as deployment type (on-premises or cloud-based options), ease of use, and support for different operating systems.