CONTAINERS & DEVOPS: CONTAINERS FIT IN DEVOPS DELIVERY PIPELINES

DevOps came to prominence to meet the ever-increasing market and consumer demand for tech applications. It aims to create a faster development environment without sacrificing the quality of software. DevOps also focuses on improving the overall quality of software in a rapid development lifecycle. It relies on a combination of multiple technologies, platforms, and tools to achieve all these goals.

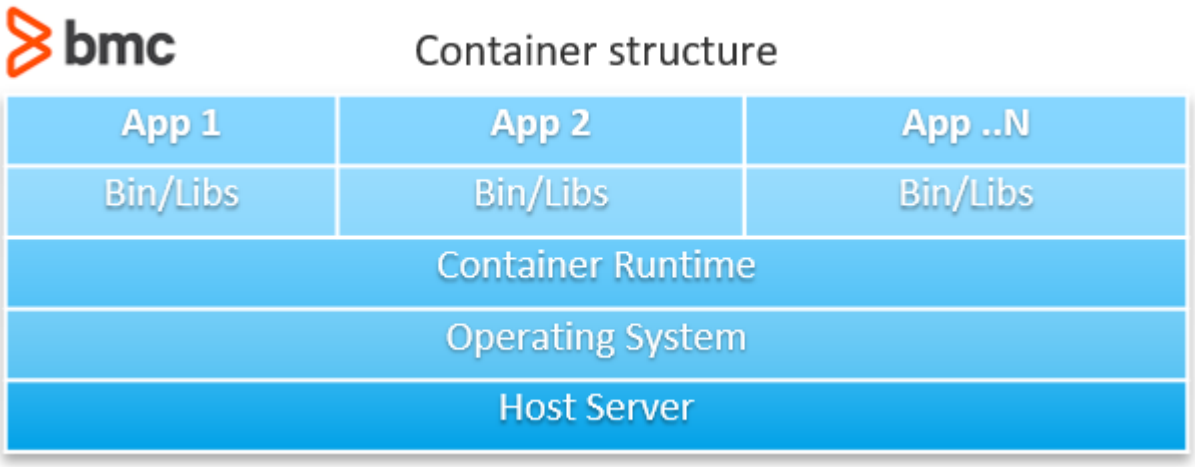
Containerization is one technology that revolutionized how we develop, deploy, and manage applications. In this post, we will look at how containers fit into the DevOps world and the advantages or disadvantages offered by a container-based DevOps delivery pipeline.

(This article is part of our DevOps Guide. Use the right-hand menu to navigate.)

What is a containerized application?

Virtualization helped users to create virtual environments that share hardware resources. Containerization takes this abstraction a step further by sharing the operating system kernel.

This leads to lightweight and inherently portable objects (containers) that bundle the software code and all the required dependencies together. These containers can then be deployed on any supported infrastructure with minimal or no external configurations.



One of the most complex parts of a traditional deployment is configuring the deployment environment with all the dependencies and configurations. Containerized applications eliminate these configuration requirements as the container packages everything that the application requires within the container.

On top of that, containers will require fewer resources and can be easily managed compared to virtual machines. This way, containerization leads to greatly simplified deployment strategies that can be easily automated and integrated into DevOps delivery pipelines. When this is combined with an orchestration platform like Kubernetes or Rancher, users can:

- Leverage the strengths of those platforms to manage the application throughout its lifecycle
- Provide greater availability, scalability, performance, and security

What is a continuous delivery pipeline?

DevOps relies on Continuous Delivery (CD) as the core process to manage software delivery. It enables software development teams to deploy software more frequently while maintaining the stability and reliability of systems.

Continuous Delivery utilizes a stack of tools such as CI/CD platforms, testing tools, etc., combined with automation to facilitate frequent software delivery. Automation plays a major role in these continuous delivery pipelines by automating all the possible tasks of the pipeline from tests, infrastructure provisioning, and even deployments.

In most cases, Continuous Delivery is combined with Continuous Integration to create more robust delivery pipelines called CI/CD pipelines. They enable organizations to integrate the complete software development process into a DevOps pipeline:

- Continuous Integration ensures that all code changes are integrated into the delivery pipeline.
- Continuous Delivery ensures that new changes are properly tested and ultimately deployed in production.

Both are crucial for a successful DevOps delivery pipeline.

(Learn how to set up a CI/CD pipeline.)

How does it all come together?

Now that we understand a containerized application and a delivery pipeline, let's see how these two relate to each other to deliver software more efficiently.

Traditional DevOps pipeline

First, let's look at a more traditional DevOps pipeline. In general, a traditional delivery pipeline will consist of the following steps.

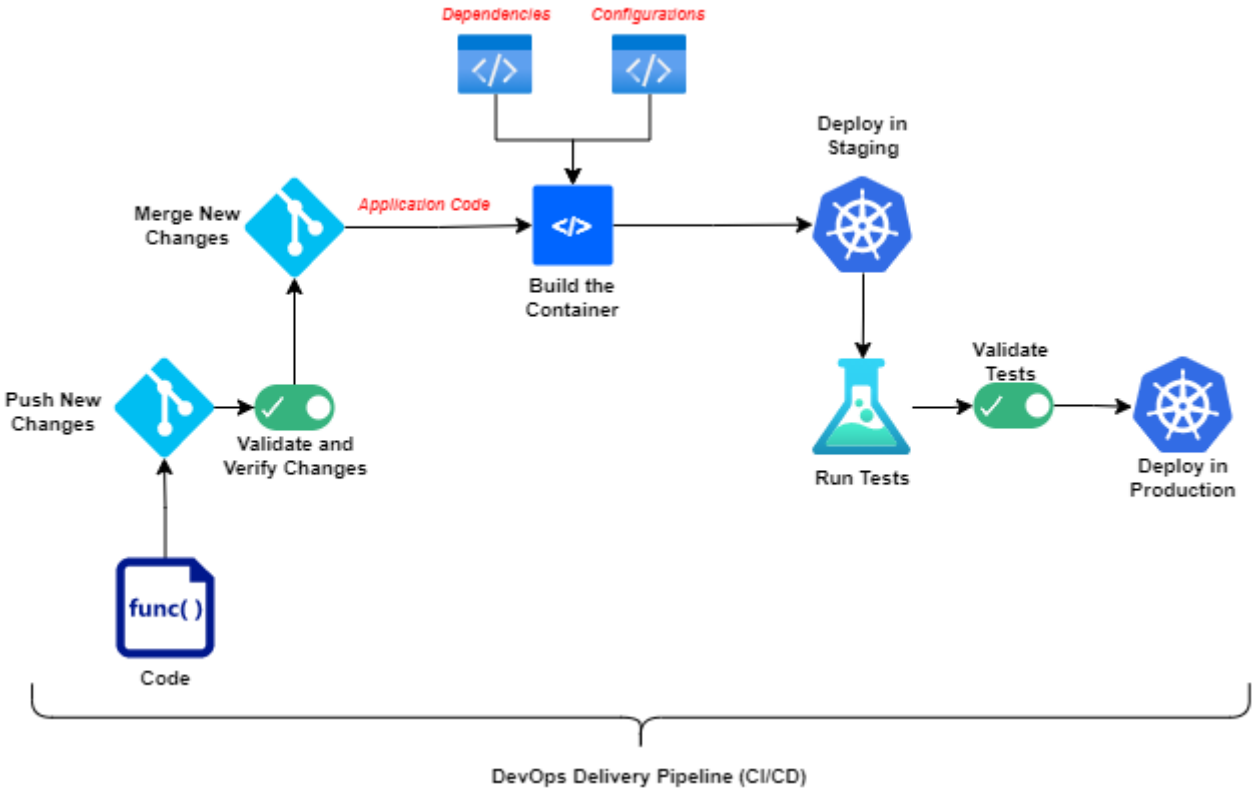
1. Develop software and integrate new changes to a centralized repository. (Version control tools come into play here.)
2. Verify and validate code and merge changes.
3. Build the application with the new code changes.
4. Provision the test environment with all the configurations and dependencies and deploy the application.
5. Carry out testing. (This can be both automated and manual testing depending on the requirement)
6. After all tests are completed, deploy the application in production. (This again requires provisioning resources and configuring the dependencies with any additional configurations required to run the application.)

Most of the above tasks can be automated, including provisioning infrastructure with IaC tools such as Terraform, CloudFormation, etc., and deployment can be simplified using platforms such as AWS Elastic Beanstalk and Azure App Service, etc. However, all these automated tasks still require careful configuration and management, and using provider-specific tools will lead to vendor lock-in.

Containerized delivery pipeline

Containerized application deployments allow us to simplify the delivery pipeline with less management overhead. A typical containerized pipeline can be summed up in the following steps.

1. Develop and integrate the changes using a version control system.
2. Verify and validate and merge the code changes.
3. Build the container. (At this stage, the code repository contains the application code and all the necessary configuration files and dependencies that are used to build the container.)
4. Deploy the container to the staging environment.
5. Carry out application testing.
6. Deploy the same container to the production environment.



Container-based DevOps delivery pipeline

As you can see in the above diagram, containerized application pipelines effectively eliminates most regular infrastructure and environment configuration requirements. However, the main thing to remember is that the container deployment environment must be configured beforehand. In most instances, this environment relates to either:

- A container orchestration platform like Kubernetes or Rancher
- A platform-specific orchestration service like Amazon Elastic Container Service (ECS), AWS Fargate, Azure Container services, etc.

The key difference

The main turning point of the delivery pipeline is the application build versus the containerization. Only the application is built in a normal delivery pipeline, while the complete container is built in a containerized application, which can be deployed in any supported environment.

The container includes all the application dependencies and configurations. It reduces any errors relating to configuration issues and allows delivery teams to quickly move these containers between different environments such as staging and production. Besides, containerization greatly reduces the scope of troubleshooting as developers only need to drill down applications within the container with little to no effect from external configurations or services.

Modern application architectures such as microservices-based architectures are well suited for containerization as they decouple application functionality to different services. Containerization allows users to manage these services as separate individual entities without relying on any external configurations.

There will be infrastructure management requirements even with containers, thought containers do indeed simplify these requirements. The most prominent infrastructure management requirement will be managing both the:

- Container orchestration platforms
- External services like load balancers and firewalls

However, using a managed container orchestration platform like Amazon Elastic Kubernetes Service (EKS) or Azure Kubernetes Service (AKS) eliminates any need for managing infrastructure for the container orchestration platform. These platforms further simplify the delivery pipeline and allow Kubernetes users to use them without being vendor-locked as they are based on Kubernetes.

(Determine when to use ECS vs AKS vs EKS.)

Container orchestration in DevOps delivery pipeline

Container Orchestration goes hand in hand with containerized applications as containerization is only one part of the overall container revolution. Container Orchestration is the process of managing the container throughout its lifecycle, from deploying the container to managing availability and scaling.

While there are many orchestration platforms, Kubernetes is one of the most popular options with industry-wide support. It can power virtually any environment, from single-node clusters to multi-cloud clusters. The ability of orchestration platforms to manage the container throughout its lifecycle while ensuring availability eliminates the need for manual intervention to manage containers.

As mentioned earlier, using a platform-agnostic orchestration platform prevents vendor-lock-in while allowing users to utilize managed solutions and power multi-cloud architectures with a single platform.

(Explore our multi-part Kubernetes Guide, including hands-on tutorials.)

Are containers right for your DevOps delivery pipeline?

The simple answer is yes. Containerization can benefit practically all application developments, with the only detractors including overly simple developments or legacy monolithic developments.

- DevOps streamlines rapid development and delivery while increasing team collaboration and improving the overall application quality.
- Containers help simplify the DevOps delivery process further by allowing users to leverage all the advantages of containers within the DevOps delivery pipelines without hindering the core DevOps practices.

Containers can support any environment regardless of the programming language, framework, deployment strategy, etc., while providing more flexibility for delivery teams to customize their environments without affecting the delivery process.

Related reading

- BMC DevOps Blog
- What Is a Container Pipeline?
- How To Run MongoDB as a Docker Container
- Containers vs Microservices: What's The Difference?
- GitHub, GitLab, Bitbucket & Azure DevOps: What's The Difference?