***The practice of development(dev) and operations(ops) engineers participating together in the entire service lifecycle, from design through the development process to production support to achieve Continuous Delivery and Continuous Deployments.***

DevOps Model Defined

DevOps is the combination of cultural philosophies, practices, and tools that increases an organization’s ability to **deliver applications and services at high velocity**: **evolving and improving products at a faster pace** than organizations using traditional software development and infrastructure management processes. This speed enables organizations to better serve their customers and compete more effectively in the market.



How DevOps Works

Under a DevOps model, development and operations teams are no longer “siloed/isolated.” Sometimes, these two teams are merged into a single team where the engineers work across the entire application lifecycle, from development and test to deployment to operations, and develop a range of skills not limited to a single function.

In some DevOps models, quality assurance and security teams may also become more tightly integrated with development and operations and throughout the application lifecycle. When security is the focus of everyone on a DevOps team, this is sometimes referred to as **DevSecOps**.

# Benefits of DevOps

### Speed | Rapid Delivery| Scale | Improved Collaboration | Security

## DevOps Practices

* [**Continuous Integration**](https://aws.amazon.com/devops/what-is-devops/#integration)

Continuous integration is a [DevOps](https://aws.amazon.com/devops/) software development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run. Continuous integration most often refers to the build or integration stage of the software release process and entails both an automation component (e.g. a CI or build service) and a cultural component (e.g. learning to integrate frequently). The key goals of continuous integration are to find and address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates.

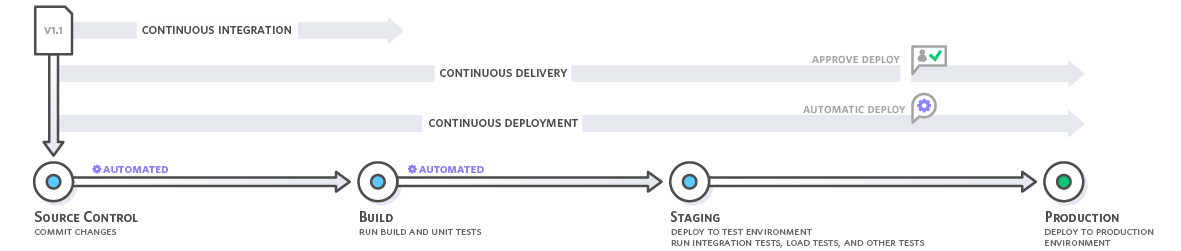
* [**Continuous Delivery**](https://aws.amazon.com/devops/what-is-devops/#cd)

Continuous delivery is a software development practice where code changes are automatically prepared for a release to production. A pillar of [modern application development](https://aws.amazon.com/modern-apps/), continuous delivery expands upon [continuous integration](https://aws.amazon.com/devops/continuous-integration/) by deploying all code changes to a testing environment and/or a production environment after the build stage. When properly implemented, developers will always have a deployment-ready build artifact that has passed through a standardized test process.

Continuous delivery lets developers automate testing beyond just unit tests so they can verify application updates across multiple dimensions before deploying to customers. These tests may include UI testing, load testing, integration testing, API reliability testing, etc. This helps developers more thoroughly validate updates and pre-emptively discover issues.

## Continuous Delivery vs. Continuous Deployment

With continuous delivery, every code change is built, tested, and then pushed to a non-production testing or staging environment. There can be multiple, parallel test stages before a production deployment. The difference between continuous delivery and continuous deployment is the presence of a manual approval to update to production. With continuous deployment, production happens automatically without explicit approval.



*Continuous delivery automates the entire software release process. Every revision that is committed triggers an automated flow that builds, tests, and then stages the update. The final decision to deploy to a live production environment is triggered by the developer.*

* [**Microservices**](https://aws.amazon.com/devops/what-is-devops/#microservices)

The microservices architecture is a design approach to build a single application as a set of small services. Each service runs in its own process and communicates with other services through a well-defined interface using a lightweight mechanism, typically an HTTP-based application programming interface (API). Microservices are built around business capabilities; each service is scoped to a single purpose. You can use different frameworks or programming languages to write microservices and deploy them independently, as a single service, or as a group of services.

* [**Infrastructure as Code**](https://aws.amazon.com/devops/what-is-devops/#iac)

Infrastructure as code is a practice in which infrastructure is provisioned and managed using code and software development techniques, such as version control and continuous integration. The cloud’s API-driven model enables developers and system administrators to interact with infrastructure programmatically, and at scale, instead of needing to manually set up and configure resources. Thus, engineers can interface with infrastructure using code-based tools and treat infrastructure in a manner similar to how they treat application code. Because they are defined by code, infrastructure and servers can quickly be deployed using standardized patterns, updated with the latest patches and versions, or duplicated in repeatable ways.

Three main task-

1. Infrastructure provisioning (setting new servers, network configuration on them, creating load balancers, configuration on infrastructure level)
2. Configuration of provisioning infrastructure (installing apps on servers, managing those apps, prepare the infrastructure/ server to deploy app)
3. Deployment of application

* [**Monitoring and Logging**](https://aws.amazon.com/devops/what-is-devops/#monitoring)
* [**Communication and Collaboration**](https://aws.amazon.com/devops/what-is-devops/#communication)

*Reference - https://aws.amazon.com/devops/what-is-devops/*

**Value Stream Map (VSM)**

Value Stream Map is a lean Manufacturing tool that seeks to map your process from supplier to customer, highlighting the flows of product and information and identifying delays and non-value adding processes.

**Types**- Operational VSM & Development VSM

**Security Testing Type**

Static Application Security Testing (SAST) | White box security testing

Dynamic Application Security Testing (DAST) | Black box security testing

Chart, bar chart

Description automatically generatedA screenshot of a computer

Description automatically generated with medium confidence

DevSecOps = Secured SDLC(SSDLC).

# Fundamentals

1. Code Quality.
2. Ability to Build in parallel.
3. A high degree of automation.

# DORA metrics

Measuring the scale and impact of your DevSecOps using DORA metrics (DevSecOps Research and Assessments)

**Throughput**

Measured using deployment frequency (DF), lead time for changes (MLT)

**Stability**

Measured using the time to restore service (MTTR), change failure rate (CFR)

Source control management

* Code versions are organized so users can track change over time.
* The version control workflow reduces the risk of multiple versions of an application spreading across multiple development machines or servers.
* The version control process reduces conflicts when working within teams.
* Tools: Git, SVN, mercurial

# Continuous integration

* Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day.
* Each check-in is then verified by an automated build, allowing teams to detect problems early.
* By integrating regularly, you can detect errors quickly, and locate them more easily.
* Tools: Jenkins, Travis CI, Circle CI

Continuous testing

* Continuous Testing’s primary goal is assessing business risk coverage
* Continuous Testing provides instant insight on whether a release candidate is too risky to proceed through the delivery pipeline
* Continuous Testing expects testing to be embedded within the development process, not tacked on at the end
* Continuous Testing evaluates each layer of a modern architecture at the appropriate stage of the delivery pipeline
* Tools: Selenium, Appium, SOAPUI, rest assured

Containers

* An easy solution for making development, testing and production environments consistent.
* Simple updates.
* Support for multiple frameworks.
* Tools: Docker

Configuration management and deployment

* Configuration management is the process of standardizing resource configurations and enforcing their state across IT infrastructure in an automated yet agile manner.
* CM enables ops to define their infrastructure in code
* Ensure a state of a machine
* Ensure policies and standards are in place
* Repeatable way of rebuilding a system
* Tools: Chef, Ansible, puppet

Monitoring tools

Infrastructure Monitoring

* Infrastructure monitoring lets you visualize events and get alerts in real time.
* Tools: Nagios, Zabbix, AWS cloudWatch, Google Stack Driver

# Application performance monitoring

* APM tools allow you to target bottlenecks with your application's framework.
* Tools: New Relic, AppDynamics