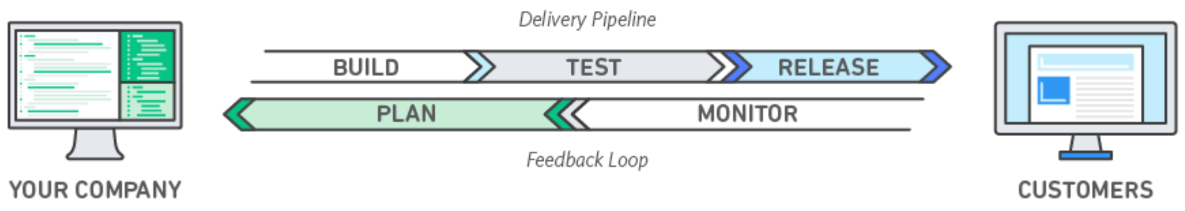
Week 9 / DevOps

Overview of DeVops:

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.



DeVops is all about the **unification and automation of processes**, and DevOps engineers are instrumental in combining code, application maintenance, and application management. All of these tasks rely on understanding not only development life cycles, but DevOps culture, and its philosophy, practices, and tools.

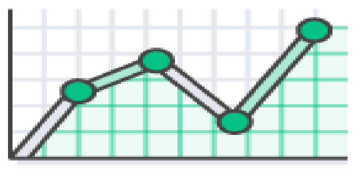
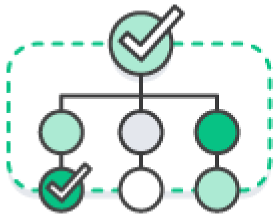
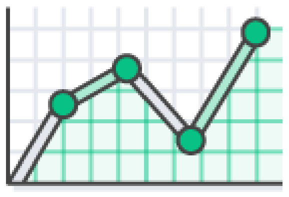
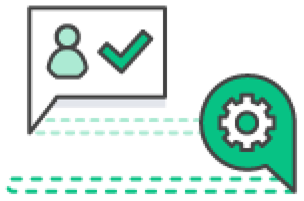
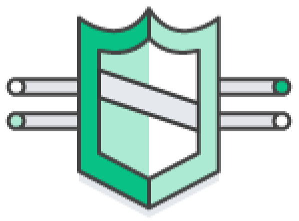
Working Principle:

Under a DevOps model, development and operations teams are no longer "siloed." 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.*

These teams use practices to automate processes that historically have been manual and slow. They use a technology stack and tooling which help them operate and evolve applications quickly and reliably. These tools also help engineers independently accomplish tasks (for example, deploying code or provisioning infrastructure) that normally would have required help from other teams, and this further increases a team's velocity.

Benefits of DevOps :

	**Speed :** Move at high velocity so that customer can Innovate faster, adapt to changing markets better, and grow more efficient at driving business results. Ex: Microservices and continuous delivery let teams take ownership of services and then release updates to them quicker.
	Rapid Delivery : Increase the frequency and pace of releases so one can innovate and improve your product faster. The quicker we release new features and fix bugs, the faster we can respond to customers' needs and build competitive advantage. Ex: Continuous integration and continuous delivery are practices that automate the software release process, from build to deploy.
	**Reliability :** Ensure the quality of application updates and infrastructure changes so you can reliably deliver at a more rapid pace while maintaining a positive experience for end users. Use practices like continuous integration and continuous delivery to test that each change is functional and safe. Monitoring and logging practices help you stay informed of performance in real-time.
	**Scale :** Operate and manage your infrastructure and development processes at scale. Automation and consistency help you manage complex or changing systems efficiently and with reduced risk. For example, infrastructure as code helps you manage your development, testing, and production environments in a repeatable and more efficient manner.
	**Improved Collaboration:** Build more effective teams under a DevOps cultural model, which emphasizes values such as ownership and accountability. Developers and operations teams collaborate closely, share many responsibilities, and combine their workflows. This reduces inefficiencies and saves time (e.g. reduced handover periods between developers and operations, writing code that takes into account the environment in which it is run).
	**Security :** Move quickly while retaining control and preserving compliance. You can adopt a DevOps model without sacrificing security by using automated compliance policies, fine-grained controls, and configuration management techniques. For example, using infrastructure as code and policy as code, you can define and then track compliance at scale**.**

DevOps culture/ How to Adopt a DevOps Model : DevOps Cultural Philosophy

Transitioning to DevOps requires a change in culture and mindset.

At its simplest, DevOps is about removing the barriers between two traditionally siloed teams, development and operations. In some organizations, there may not even be separate development and operations teams; engineers may do both.

With DevOps, the two teams work together to optimize both the productivity of developers and the reliability of operations. They strive to communicate frequently, increase efficiencies, and improve the quality of services they provide to customers.

Quality assurance and security teams may also become tightly integrated with these teams. Organizations using a DevOps model, regardless of their organizational structure, have teams that view the entire development and infrastructure lifecycle as part of their responsibilities.

DevOps practices:

Together, these practices help organizations deliver faster, more reliable updates to their customers. Here is an overview of important DevOps practices.The following are DevOps best practices:

- Continuous Integration
- Continuous Delivery
- Microservices
- Infrastructure as Code
- Monitoring and Logging
- Communication and Collaboration

	Continuous Integration Continuous integration is a software development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run. 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 Continuous delivery is a software development practice where code changes are automatically built, tested, and prepared for a release to production. It expands upon continuous integration by deploying all code changes to a testing environment and/or a production environment after the build stage. When continuous delivery is implemented properly, developers will always have a deployment-ready build artifact that has passed through a standardized test process.
	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 : 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.
	Monitoring and Logging : Organizations monitor metrics and logs to see how application and infrastructure performance impacts the experience of their product's end user. By capturing, categorizing, and then analyzing data and logs generated by applications and infrastructure, organizations understand how changes or updates impact users, shedding insights into the root causes of problems or unexpected changes. Active monitoring becomes increasingly

	important as services must be available 24/7 and as application and infrastructure update frequency increases. Creating alerts or performing real-time analysis of this data also helps organizations more proactively monitor their services.
	Communication and Collaboration :Increased communication and collaboration in an organization is one of the key cultural aspects of DevOps. The use of DevOps tooling and automation of the software delivery process establishes collaboration by physically bringing together the workflows and responsibilities of development and operations. Building on top of that, these teams set strong cultural norms around information sharing and facilitating communication through the use of chat applications, issue or project tracking systems, and wikis. This helps speed up communication across developers, operations, and even other teams like marketing or sales, allowing all parts of the organization to align more closely on goals and projects.

Continuous Integration:

Continuous Integration (CI) is a software development practice where developers regularly merge their code changes into a central code repository, after which automated builds and tests are run. CI helps find and address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates.

AWS offers the following services for continuous integration:

Topics

- AWS CodeCommit
- AWS CodeBuild
- AWS CodeArtifact

Continuous Delivery: Continuous delivery is a software development practice where code changes are automatically prepared for a release to production. A pillar of modern application development, continuous delivery expands upon 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 preemptively discover issues. With the cloud, it is easy and cost-effective to automate the creation and replication of multiple environments for testing, which was previously difficult to do on-premises.

AWS offers the following services for continuous delivery:

- AWS CodeBuild
- AWS CodeDeploy
- AWS CodePipeline

Version Control:

Version control systems like Git, Subversion, and Mercurial **provide a logical means to organize files and coordinate their creation, controlled access, updating, and deletion across teams and organizations**. Version control is closely related to automation.

A source control system, also called a version control system, **allows developers to collaborate on code and track changes**. Source control is an essential tool for multi-developer projects. Our systems support two types of source control: Git (distributed) and Team Foundation Version Control (TFVC)

Configuration Management :

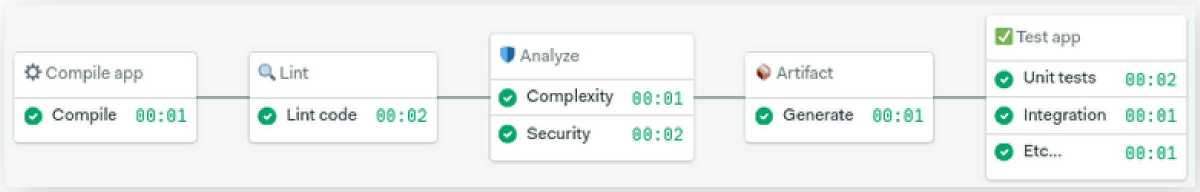
Developers and system administrators use code to automate operating system and host configuration, operational tasks, and more. The use of code makes configuration changes repeatable and standardized. It frees developers and systems administrators from manually configuring operating systems, system applications, or server software.

Build Process :

The build stage is the first stretch of a CI/CD pipeline, and it automates steps like downloading dependencies, installing tools, and compiling. Besides building code, build automation includes using tools to check that the code is safe and follows best practices. What happens in a build stage

In continuous integration (CI), this is where we build the application for the first time. The build stage is the first stretch of a CI/CD pipeline, and it automates steps like downloading dependencies, installing tools, and compiling.

Besides building code, build automation includes using tools to check that the code is safe and follows best practices. The build stage usually ends in the artifact generation step, where we create a production-ready package. Once this is done, the testing stage can begin.



The build stage starts from code commit and runs from the beginning up to the test stage

The build stage starts from code commit and runs from the beginning up to the test stage

Build automation verifies that the application, at a given code commit, can qualify for further testing. We can divide it into three parts:

1. **Compilation**: the first step builds the application.
2. **Linting**: checks the code for programmatic and stylistic errors.
3. **Code analysis**: using automated source-checking tools, we control the code's quality.
4. **Artifact generation**: the last step packages the application for release or deployment.

Step 1: Compilation

Step one of the build stage compiles the application. On compiled languages, it means that we can generate a working binary, whereas on interpreted languages, we confirm that we have the required dependencies and tools to successfully build the application.

The result can be a binary file, a code tarball, an installable package, a website, a container image — **the main thing is that we have something that we can run and test**.

Step 2: Analyzing Your Code

Programming requires discipline. We must solve hard problems while following good practices and abiding by a common coding style. Automated analysis tools help you keep out gnarly code at bay. In general terms, there are three classes of code analysis tools:

- **Linting**: linters improve code quality by pointing problematic and hard-to-maintain bits. Linters use a set of rules to enforce a unified standard that improves readability for the team.
- **Quality**: this class of tools uses metrics to find places where code can be improved. The collected metrics include the number of lines, documentation coverage, and complexity level.
- **Security**: security analysis tools scan the code, flag parts that may cause vulnerabilities, and check that dependencies don't have known security issues.

Since no one wants to ship unsafe or buggy code, we must design a CI pipeline that stops when errors are found. When the CI pipeline runs the right code analysis tools, **developers**

who review code can focus on more valuable questions such as "*Is this code solving the right problem?*" or "*Does this code increase our technical debt?*".

Step 3: Preparing your Application for Release

The released package will include everything needed to run or install the application, including:

- Application code.
- Installation scripts.
- Dependencies and libraries.
- Application metadata.
- Documentation.
- License information.

The final package can be published on a website, in a package database such as npmjs.com, in a container registry, or can be directly deployed into a QA or production environment.