TABLE OF CONTENTS

WHAT IS DEVOPS?

DevOps is a series of processes and tools, as well as a cultural practice within a company, that encourages collaboration between development (developers, QA, project management, and other associated teams) and operations (systems administrators, systems architects, security, and related). It is often characterized by its use of fast release cycles, common tools, and overall "DevOps" concepts that are reviewed below.

A company practicing DevOps does *not* need to use every tool or concept associated with DevOps; however, DevOps practices should permeate through the entire release cycle, from start to finish to any additional bug fixes and releases. The goal is to increase productivity through better communication and eliminate the issues that normally arise between traditionally siloed teams.

DEVOPS DICTIONARY

The following terms are in alphabetical order. Associated tools and products are noted where appropriate.

ACCEPTANCE TESTING

Reviewing a system to ensure it meets the necessary requirements for release. This includes any specifications laid out by the business or client and ensuring it meets the end users' needs. Items looked at can include usability, data integrity, scalability, and more.

AGILE

A methodology commonly associated with DevOps. Agile development encourages quick release cycles ("release early, release often") and the ability to respond quickly and effectively to change.

ARTIFACTS

Artifacts are the "byproducts" of a development cycle. This includes anything from compiled code, scripts, and test suites to written documentation, user stories, and class diagrams.

BUILD AUTOMATION

The process of automating tasks during the software development cycle. This can include checking out commits to version control repositories, compiling code, running tests, and packaging results.

Build automation products include Jenkins, Atlassian's Bamboo, Circle CI, and Travis CI.

CLIENT-SERVER/CLIENT-ONLY INFRASTRUCTURE

Client-server infrastructure involves a primary server (or servers for large infrastructures) that manages the resources used by the client. A common example of this is the relationship between a Puppet Master or Chef Server and Puppet Chef nodes. The "master" server tells the clients what to install, what services should be running, and overall how the client should be set up and function.

In contrast, client-only infrastructure does not have a "master" server orchestrating how things are set up. Instead, scripts or services such as those provided by AWS or Azure are used to set up the infrastructure.

CONFIGURATION DRIFT

When servers within the infrastructure change from their intended configuration through manual updates or even just general use. When undetected or uncorrected, configuration drift can cause failures, downtime, and other issues within the system.

Tools such as Chef, Puppet, Ansible, and Salt can help prevent configuration drift by periodically "checking in" on the servers they manage to ensure the client configurations match the described infrastructures.

CONFIGURATION MANAGEMENT

The process of creating and maintaining consistent infrastructure and services. This can be a combination of tools and practices use to ensure that a system or product stays within its required settings and can easily change and evolve as these requirements change.

Configuration management tools include Ansible, Chef, Puppet, and Salt.

CONTAINERS

A tool for isolating applications and associated frameworks and libraries on a system. Uses the host system's operating system and kernel to provide resources, while keeping the application siloed and unable to access the underlying system or any other containers located on the system.

Examples include Docker, LXC (Linux Containers), and FreeBSD Jails.

CONTINUOUS DELIVERY

An extension of continuous integration (see below) and the "deploy early, deploy often" philosophy often associated with DevOps and Agile development. When making frequent pushes to the mainline, continuous delivery calls for these pushes to be reviewed and quickly deployed if successful.

Tools such as Jenkins, Bamboo, Circle CI, and Travis CI all assist in continuous delivery.

CONTINUOUS DEPLOYMENT

Continuous deployment is the practice of automating various steps up to and including the deployment of the changes. This decreases lead time and allows releases to be wholly automated instead of relying on someone to manually "push out" the changes.

Products such as Jenkins, Ant, and Travis CI allow for the implementation of continuous deployment.

CONTINUOUS INTEGRATION

The practice of merging code to the main version control branch multiple times a day to ensure there are minimal integration problems when making changes to the code or system. This ensures developers and others are consistently working from a similar baseline.

Continuous integration benefits from breaking things down into smaller tasks that can be pushed to the mainline without being unfinished.

CONTINUOUS LEARNING

Learning and growth through work experiences and the overall need to keep learning and honing one's work skills. Continuous learning can work on three levels:

The individual level, in which a single employee works to expand their skills. This can be as simple as asking the right questions when the employee does not understand a task or as involved is taking classes and going through training programs. Continuous learning is not just about learning for yourself, either. Mentoring and teaching can also play an important role in one's continuous learning experience.

At a team level, a team or group within the company use and share their collective knowledge and experiences to further the team. This can also involve implementing certain practices to aid in learning, such as retrospectives or team Q&As.

Finally, on an organizational level, continuous learning involves reviewing and creating policies and processes to aid in the overall achievements of company goals. This often involves receiving both internal and external feedback from employees and customers.

IDEMPOTENCY

The ability for an operation or code to be run multiple times without any unintended or extraneous effects. For example, if on your configuration management platform, you have it set so your web servers will install Apache, Apache will only ever be installed once, regardless of how many times the configuration manager is run against a server in that group.

Must configuration management tools are idempotent, including Ansible, Chef, Puppet, and Salt.

INFRASTRUCTURE AS CODE

The ability to lay out one's infrastructure through machine-readable files that define the parameters of the system. For Chef, this is in recipes; for Puppet, manifests; etc. The configuration management tool is then able to take these parameters and configure the client servers to match the defined infrastructure.

Ansible, Chef, Puppet, and Salt all use the concept of "infrastructure as code" to configure and maintain systems.

INFRASTRUCTURE AS A SERVICE

In contrast to hosting your own data center, or renting out space in a data center to host your servers, infrastructure as a service providers offer managed hardware, servers, or storage that can than be customized to host an app, display a website, or otherwise work as regular, self-hosted infrastructure without the overhead of having to set up and manage the low-level aspects of keeping a server.

Infrastructure as a service providers often offer bare instances with only the base operating system installed, although products vary and can also feature more involved tools for setting up infrastructure.

MICROSERVICES

A type of service-oriented architecture in which the various parts of an application are built as loosely-coupled, smaller services that are modular to use. This goes hand-in-hand with the concepts of continuous integration, which often requires developers to parse down major tasks into smaller segments that can be pushed out as soon as they are complete.

MUTABLE/IMMUTABLE INFRASTRUCTURE

Mutable infrastructure is infrastructure that is updated, changed, and adapted. It is infrastructure that is expected to last, not be fully redeployed when changes are made; in contrast, immutable infrastructure is not intended to be changed. Instead, the instance or container is fully redeployed when changes are made, and then the initial instance or container is terminated.

ORCHESTRATION

The automated management of computer systems, software, or services. Policies are defined and used to do things such as set up workflows, bootstrap or maintain servers, and automatically scale services up or down as needed.

Orchestration tools include infrastructure orchestration, such as Chef and Puppet; container orchestration, such as Kubernetes for Docker; and others.

PROVISIONING

The act of preparing a server for use. This can include installing the operating system and services, making any configuration changes needed, and otherwise completing any tasks required before the server joins the greater infrastructure network.

SERVERLESS

Serverless architecture allows developers to feed their code directly into a service that handles all infrastructure for them. Serverless depends on the use of third-party vendors to provide either "backend as a service" or "functions as a service" features that allow developers to worry only about code, not the underlying system.

AWS Lambda and Azure Functions are two popular serverless offerings.

TEST-DRIVEN DEVELOPMENT

A development approach in which a test is written first and then code is created with the end goal of passing the test. The code is then run against the test and refactored as needed, then a next set of tests and goals are put in place. This approach encourages developers to think about the requirements and end goal of the code before putting in the work of writing functional code itself.