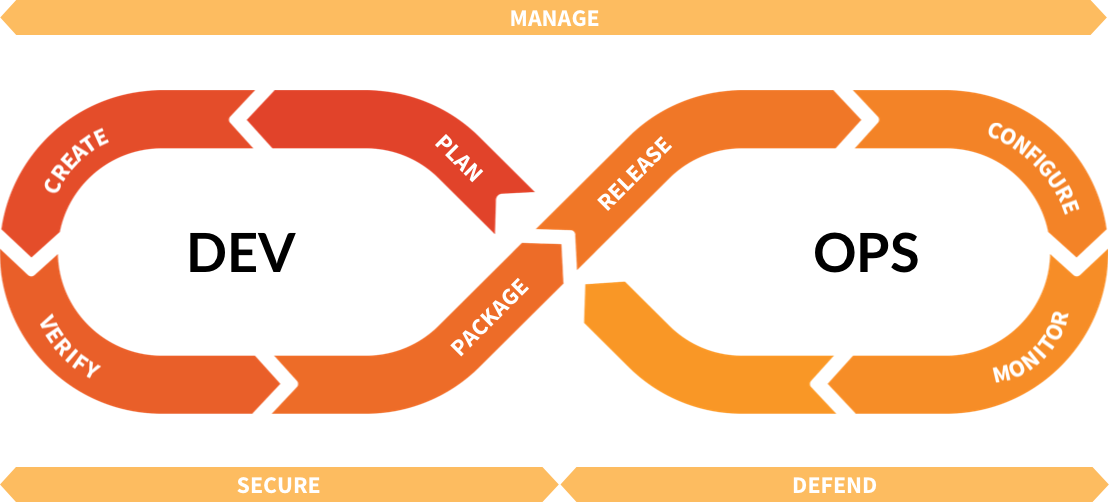
Lecture 1 - Why DevOps (Developer Operations) Matters



Why DevOps adoption?

DevOps culture brings the cohesion of autonomous teams to fast paced, high-demanding environments.

It has a set of processes that when applied correctly can:

1. Drastically reduce IT overhead
2. Remove manual tasks both of a mundane or complex nature
3. Enforce best security practices
4. Expedite product delivery
5. Improve communication and collaboration across teams

Perhaps it’s best put by a present-day pioneer in the movement, Gene Kim:

**"...its adoption can also enact changes in the culture of a business—for example, allowing small teams to work quickly and independently. You know, back in the bad old days, before DevOps started gaining ground within large and small enterprises, for a software company to deploy even small updates or upgrades or bug fixes, the work might involve literally hundreds of developers—and it might take weeks to get a new product out the door. Today's high-performing enterprises and DevOps adopters, on the other hand, deploy so much faster, and the products they release generally have higher rates of reliability."**

In their natural habitat, a DevOps engineer will cross collaborate with many people in different positions. An engineer has the ability to work autonomously and strive for improving the experience of product users, along with the overall development process.

An engineer helps optimize the environment they are working in. Be it – financially, increasing overall productivity, automating a task or solving a technical problem.

One part of the process we’ll be referring to is an application “build”. A build is a term commonly used in the industry to refer to source code compilation which provides an executable for a computer to run. Running a JavaScript build as an example, allows us to create a process which produces a binary for execution.

Employing a build process as a piece of the puzzle for delivering software end-to-end can provide a plethora of benefits. We’ll be able to automate things like linting for unconventional syntax, unit testing, checking for compilation errors or security vulnerabilities along with much, much more. We’ll be taking a close look at how a developer commonly ships code from their personal machine to a Production environment and the underlying process responsible for the highest efficacy possible.

I previously used JavaScript as an example for a build process because we’ll be using JS, specifically Node, throughout this course. If some terminology isn’t making sense, no worries or concerns needed. We’ll be unpacking terms as we move forward but if you’re puzzled about something, please have a look at the supplementary material for definitions of terms or feel free to email or bring something up in class.

We’ll be using Node through command line quite a bit. Node will allow us to install third-party dependencies for our application, define configurations, automate tasks and help-out with the build process. If you’ve perused GitHub repositories before, you may have come across some badges looking like this:

(fig.1)

Or this:

(fig.2)

Or maybe even this:

(fig.3)

The above “badges” are to provide transparent data on the applications most recent build. When the build shows as passing you see fig 1. A failed build would be represented by fig.2. Fig.3 is an example of a metric a Continuous Integration (CI) tool can collect and provide upon request.

This metric is known as **code coverage which is a representation of source code covered by automated tests**. One form of automated testing that a developer may consider introducing is Unit Testing. Unit Testing allows for developers to create test cases for the code they write. This helps ensure that a function is working as intended and all use cases are covered. It makes for a more robust codebase, one that’s easier to work with amongst teams, can act as documentation (when there is an unfortunate lack of documentation) and ultimately, prevents unintended behavior when source code changes are made.

We’ll be looking at Unit Testing through the lens of a DevOps engineer. We want an understanding of how automating Unit Tests can affect our pipeline. In this course, we won’t be talking about Test Driven Development (TDD). TDD is a methodology where Unit Tests are commonly used but it’s out of scope for this course.

Having previously mentioning “pipeline”, I owe you an explanation. **A “Pipeline” is a sequence of events triggered by a source code change or a manual action**. Typically this process starts on a developers laptop and ends on a QA or Production-like server (if the build does not fail). If a build makes it’s way through the pipeline from beginning to end, there can be a “clean-up” process automated to keep administration of the server clean. As an example; things like log files may need to be removed from storage, applications killed/spawned, or reports collected.

Since a DevOps engineer wouldn’t be responsible for developing a product, we won’t be working in the product development domain. Given this, we will be starting with a working web application. After the pre-requisites are installed during our lab, we’ll look under the hood of the application that we’ve been given to get Continuously Integrated and Continuously Deployed (CI/CD).

Even if you aren’t a Go, Ruby, JavaScript, or a \*fill in the blank\* developer, it’s crucial to understand the mechanics of the application you’re responsible for interacting with. To use Ruby as an example, if you’re working with a Rails app, it’s important to know what the Rails build process looks like because ultimately, you’d be responsible for automating any manual steps involved in the process.

To have something online we’ll need more than just an application; a server for hosting and a tool for us to automate the deployment processes must also be acquired. For hosting we’re going to look at a few options and determine which is the best one for us to use. Our tool of choice for the CI process will be Travis CI. We’ll be combining Travis with GitHub. Working with GitHub and Travis in conjunction will provide a different perspective when working with the two platforms in tandem.

Given the complexities of the industry and the demand for scalable products that can handle traffic for potentially millions of users, there are a lot of options and throughout the course, we’ll be performing an analysis to help us evaluate the right products. To make this demand realistic and enjoyable to work with, operational and automation tools have become a valuable skillset to possess when entering the job market.

With this course, you will be finishing with concepts and knowledge needed for interviews and environments where you may be exposed to the concepts previously mentioned. The chances are quite high given the current market demand for this type of knowledge. This course won’t make you a DevOps engineer, but it will shine some light in a part of the industry that developers will soon have to become much more familiar with.

Whether you’re a front-end developer working with a QA (Quality Assurance) analyst, a back-end developer working with Senior Architects or a network administrator supporting infrastructure, you will need to know the terminology commonly used and what’s considered to be best practices when it comes to successfully shipping software.

Along with some of the tools mentioned, we’ll also be leveraging and exploring some of the underlying protocols which make working with servers a cinch. This will remove some of the magic from the tools we use. Along with some lower-level knowledge, our intention is to spawn new ideas and explore creative solutions to common problems.

This course is not designed as a means for students to earn credits or boast their GPA. To successfully complete this course, you must attend every lab. **If you feel like lectures aren’t your thing** or there is something I can do better, shoot me an email and please let me know. I respond to all emails and consider all feedback offered. If you can’t make it to a lab, also please fire off an email to inform me of your absence. Not doing so will result in a lower mark for a class that is set up for your success!