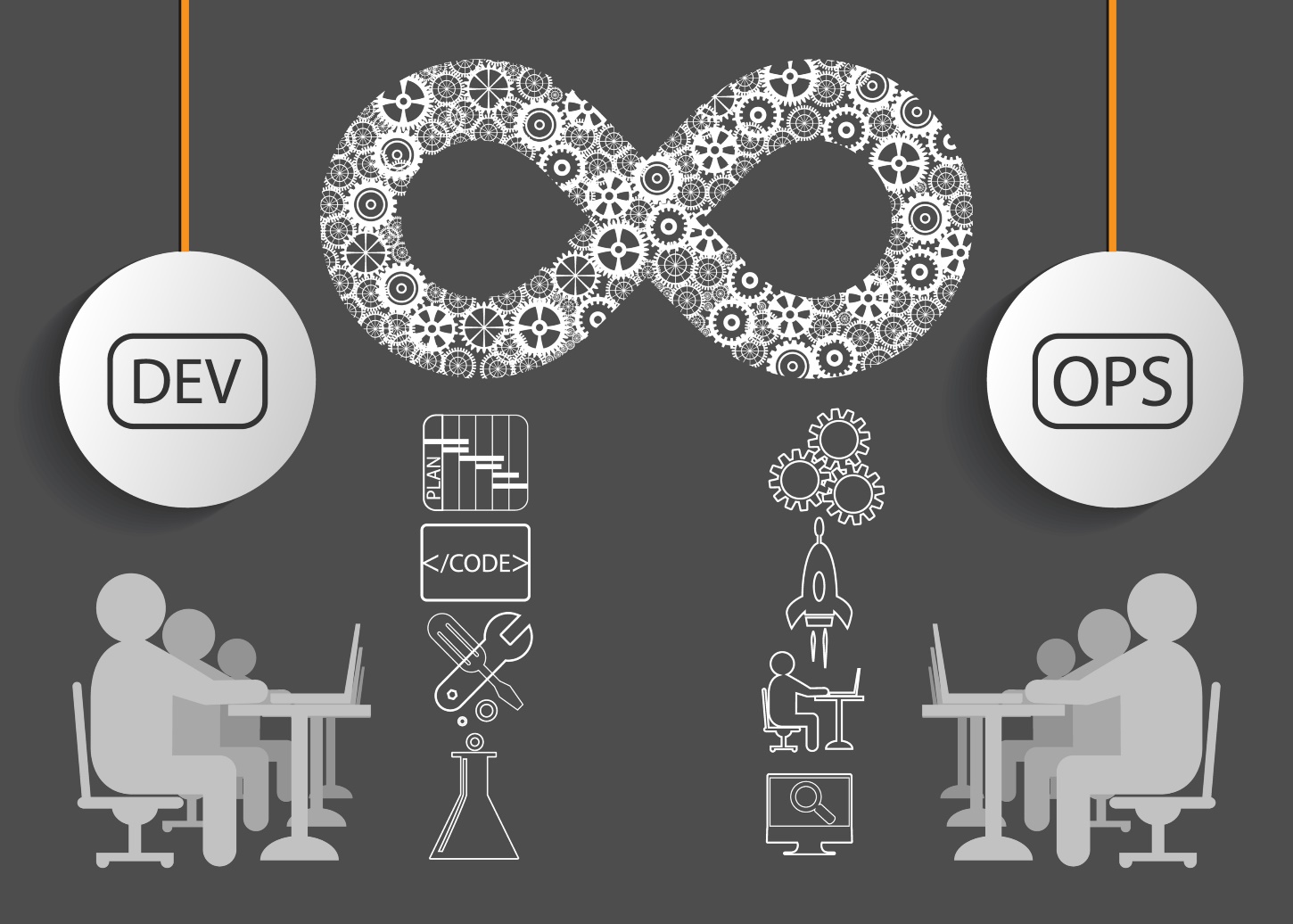
# Lecture 04 – The DevOps Landscape



This week we will be exploring the ins and outs of what it means to be a DevOps engineer. It’s not all fancy tooling and automating workflows, **an engineer is responsible to create and/or enforce software development strategies to bridge the gap between developer and operation teams.** Using DevOps processes, **organizations can release small features for an application very quickly with fewer failures** in the mix.

In the event of a failure, an engineer is responsible for the implementation of monitoring tools which help raise transparency and alert all teams involved regarding the failure. Monitoring, feedback and reporting solutions are integral to ensure problems are solved as soon as possible. This also leads to a shortened lead time between fixes.

If you’ve heard of the waterfall model before, DevOps helps overcome all the limitations that the paradigm is known for. It removes the linear phases that a waterfall model is dependent on by automating out the sequential tasks. A sound DevOps strategy should never block work for anyone. In the instance of an error, the strategy implemented should be capable of intelligently handling the failure by means of either an automatic revert to the latest working build, notifying teams or preventing broken code from ever reaching a development, staging or QA server.

Some of the commonly used tools which help make this possible are:

1. Jenkins - Automated build server with plugins built for creating CI/CD pipelines
2. Selenium – Automation test suite
3. Puppet – Enforces system configurations, define Infrastructure as Code (IaC), server management
4. Chef – Enforces system configurations, define Infrastructure as Code (IaC), server management
5. Ansible - Enforces system configurations, define Infrastructure as Code (IaC), server management
6. Docker – Software containerization platform
7. Kubernetes – Container orchestration tool
8. Nagios – Continuous monitoring

It’s great to know about all the tooling available and throughout the course we’ll continue to look more at the tools previously listed but first, let’s get a better perspective of exactly just what a DevOps engineer is. **A DevOps engineer is somebody who:**

1. **Understands the Software Development Lifecycle**
2. **Has adequate knowledge of various automation tools available for solving complex business problems**
3. **Works on increasing overall productivity amongst all teams**
4. **Focuses on reducing fiscal overhead through the optimization of the technical landscape**

A DevOps engineer works with both developers, operations and other IT staff members to oversee code releases. Typically, they’re background is that of a developer who got interested in the deployment process along with network operations or sysadmins with a passion for scripting and coding solutions for business problems. They’re geared towards the development side where they can improve planning, testing and deployments for software releases.

In the end, DevOps can be a little tricky to pigeonhole a definition for the term. To be successful as an engineer, like for most things, it requires an understanding of all the layers that are involved. It’s important to understand what a given DevOps implementation might contain or not contain. In the end, what DevOps hopes to bring to an Agile workplace is the understanding and practice that software isn’t done until it’s successfully delivered to a user. Going above and beyond that, it must also meet the users expectations around availability, performance and pace of change.

To provide a more specific explanation, I’ve come to believe there are three primary practices for an engineer to engage and focus in when discussing the context of DevOps.

* **Infrastructure Automation –** creating systems, OS configs and application deployments as code.
* **Continuous Delivery** – build, test, deploys of applications through automated practices
* **Site Reliability Engineer (SRE)** - operate your systems; monitoring and orchestration and optimizing the workplace landscape for operability

## Job Roles and Responsibilities for a DevOps Engineer

What DevOps-centric roles would be available at a company for an engineer to attain? Let’s look at some common roles along with a description of the position.

1. DevOps Evangelist – This is typically the leader or more officially recognized as the Principal Officer. An Evangelist is responsible for implementing DevOps in the work space.
2. Release Manager – This is the individual(s) who releases new features and also is responsible for ensuring post-release product stability
3. Automation Expert – If you like solving problems with software tools, this role may be an ideal position as an Automation Expert is responsible for optimizing automations and the orchestration of tools
4. Software Developer/Tester - This role is focused on the development side, whether it’s working with IaC, writing scripts and tests that the solution is adequate
5. Security Engineer – This position is designated to monitor the products safety & perform health checks

With some insight as to what niche roles are available for someone with interest in DevOps to reach for, let’s look at some of the skills required to become a successful engineer. For this examination, why don’t we look at what Cisco is looking for in an engineer and then breakdown some of the skills required to fulfill the position.

* Skills taken from a Cisco job posting for a DevOps Engineer:
* Deploy, upgrade and maintain the cloud (AWS) infrastructure for a reliable, highly available, secure and scalable platform for production
* Experience working with at least one of the following languages: Node.js, Python, PHP, Ruby and Java. Proficient with Git and Git workflows
* Monitor, upgrade and operate Cisco CVIM OpenStack private clouds in the lab
* Monitor, upgrade and operate Kubernetes platforms in the lab
* Monitor, upgrade and operate VMWare ESXi Clusters in the lab
* Provide Infrastructure-as-a-Service for engineering teams during the development cycle
* Provide CI/CD tools as a service for engineering teams (Jenkins, Docker Repo, Vault, GitHub)
* Troubleshoot and resolve issues in test, demo, and production environments as needed
* Participate in an on-call rotation and respond to escalations outside of normal working hours as needed
* Suggest product improvements to make the application high performing
* Contribute to and maintain a knowledge base
* Implement state of the art monitoring and logging
* Apply best practices and emerging concepts in DevOps and infrastructure automation
* Design and implement security best practices from day one
* Remain knowledgeable about emerging standards and trends

As you can see above, it’s more than just working with a fancy tech stack. Although there is some neat technology in the mix, there is also skill requirements for best practice insights when it comes to standards and trends in the industry. Also, notice how many times the word “monitor” is listed? Monitoring is a crucial piece in the DevOps landscape.

Some other key takeaways from the job description would be:

* Knowledge of a cloud platform
* Container orchestration tooling (Kubernetes)
* Proficient in scripting, and Git and Git workflows
* Experience in developing CI/CD pipelines

What about Salary expectations for a DevOps Engineer? It’s a lot that Cisco is asking for there, so naturally, not only is DevOps a highly sought-after craft but with that comes a high paying salary. Don’t take my word for it though, let’s see what Indeed.com is showing for some salaries in American dollars:



Indeed.com also collected insights from 18,434 employees, users, along with past and present job advertisements. In 2018, it was determined that an average DevOps engineer's salary was at the tune of $121,589 USD.

## DevOps & The Environments Which They Thrive

When you land your dream IT gig, chances are there will be environmental stages that the code is to live in and traverse through. Most commonly development will start on a local dev machine. Once the developer would like to deploy their changes, it is triggered through a VCS transaction and starts making its way to the source codes first home. Usually, the first home is some sort of development server that is available to an entire team or organization. It’s on the development server where the dev can first see there changes outside of their local environment.

After the code is pushed from a developer's laptop, passes any test case and/or code analysis and winds up on a development server, it’s going to provide the developer with a better idea as to how they’re change impacted the application. It’s a DevOps engineer who automates releases from a development server to a QA environment where a Quality Assurance Analyst can perform any sort of regression tests and qualify that the change meets the specificities of the demand for the change.

Once QA approves the change, sometimes the QA analyst will have the capabilities to schedule a release through a build platform like Jenkins. Often, releases are bundled together once they’ve been qualified and scheduled for release as a packaged version. This is where Git workflows like feature flagging, branching models or tags come in to play.

**Feature flags allow code to ship a project in different flavors by dynamically toggling certain functionality.** They enable teams to achieve Continuous Deployment by deploying new features to production at smaller batches for controlled testing. This allows for a complete separation of feature delivery from customer launch, helping reduce any associated risk with Production deployments.

Branching models are another Git workflow. It allows developers to “branch” off of the master repository for an isolated development environment. We’ve briefly touched upon this but to expand on it a little further, branching models provide clearly defined and intended purposes for a specific branch. To give an example; if you had to integrate a new feature, a developer might fetch the latest source from a remote repository and then create a new branch from it.

Let’s imagine the feature request is to include a new endpoint for a webhook. Rather working directly off the master branch, which is the single source of truth for Production, the developer would want to create a branch from the master copy called “webhook-endpoint”. Now the developer can freely experiment and once they have met the demands for the request, they can submit what's known as a “Pull Request” or “PR” for short.

Once a developer submits a PR, the rest of the team is notified of the request for the change to be merged into the master branch. **Best practice dictates that after a PR is submitted, the code is to be reviewed by the developer’s peers**. This is to ensure at least a second set of eyes can confirm that the code is following any desired convention, does not include any sort of syntactical error or if there may be a better approach. If a peer developer notices anything improper with the PR, they can deny the request and provide comments as to why it was denied. Should there be no issue with the request, another member of the team can approve the PR for a code merge. These additional steps aren’t intended to make life difficult, it’s just another measure of quality control implemented to ensure product stability. The number of approvals required is often two or more developers on the team.

Once the PR has been approved for a code merge, it’s the merge which triggers any sort of CI/CD pipeline process that will then move the change to an appropriate branch or environment for either further analysis or prepped for a Production release.

Preparing for Production release will vary by organization. Due to unique requirements or procedures put in place for products, it requires a release plan/schedule that best meets the demands of the business and the clients which will be engaging with the product. Some companies will schedule bi-weekly releases that coincide with Agile sprints, others monthly but typically, strategies need to be in place for immediate releases for hot-fixes. This is where DevOps protocols and procedures come in to play to ensure accommodation for business requirements. As requirements vary, the procedures need to be designed around the needs of the business.

## DevOps Toolchain

As you can see, DevOps is not just about software solutions. It’s also team engagement, helping raise transparency on product progress, Git workflows to provide solutions for optimization of the development lifecycle and configuring environments for software to thrive. Before we start looking at all the awesome tech that’s available for an engineer to take advantage of, I’d like to really hammer home the idea that DevOps is not just using cool tools and new technology.

I won’t try to argue this point or enforce an opinion in this course, but one thing to stay open minded on is that DevOps still needs a commonly accepted definition. This will prevent confusing and poorly structured decisions which increase the risk that people will start implementing common tools known to be “DevOps” without bearing in mind the principles.

While the implementation of DevOps tooling could be well intended, if a well-constructed thought process is not backing the reasoning for the introduction of a new technology, it certainly is an anti-pattern and a recipe for a lot of headaches. The point I’m trying to make, before using something new, shiny and cool, have a good reason for the use. I’ve witnessed people implement complex tooling for the wrong reasons which resulted in significant loss and disastrous consequences.

Disclaimer aside, let’s now look at some of the most common items found in a DevOps toolkit and what advantages the tool brings to an engineer and ultimately, the company and environment to which it will be introduced.

## Jenkins

Probably one of the most important parts of DevOps tooling is a good CI/CD platform. Jenkins is one of the most recognizable CI/CD platforms in the industry primarily due to the number of plugins built-in to the suite. With 563 open-source contributors at the time of writing and 13,306 stars given on GitHub, it is quite evident as to how highly adopted this automation server is.

It provides over 1000 plugins to support almost anything that an engineer would want to automate. Jenkins is used for producing application builds whether it’s a JAR file, a Node binary or a Microsoft application, Jenkins will automate out the mundanity of a process so humans can focus on completing things machines cannot yet do.

Jenkins can run tests to detect bugs and other issues as soon as they are introduced by a new change. It can also perform static code analysis which allows for source code to be analyzed without actually executing the software. Another feature that makes Jenkins, Jenkins – is the robust way in which it can handle deployments. Offering a plethora of configurations, almost any organization can count on Jenkins for release deployments.

## Selenium

Selenium is a framework for the automation of web application testing. It’s quite feature-rich in terms of the solutions it brings when having to work with web applications that are dynamically generated or heavily reliant on JavaScript, which in comparison to some of the other automated test suites out there, is the clear winner.

Selenium also offers a developer API to write scrapers and crawlers for performing mundane tasks like regression tests. Rather than having to go and check every UI component after a code change, Selenium test cases can be written to automate out the process.

It also offers the use of headless chrome, Firefox Gecko and other commonly used web drivers for real-time playback of what’s happening during execution of the test suite. Selenium is commonly used by QA but it’s not uncommon for a DevOps engineer to have to set up the infrastructure and associated pipeline integrations.

## Puppet, Chef and Ansible

Puppet, Chef and Ansible are all configuration management tools. They’re all designed to install and manage software on existing servers but the way in which they complete these tasks differ from tool-to-tool. When we were talking about **Infrastructure as Code (IaC)** earlier, it’s these platforms that make this possible.

We know that configuration management is the ability to manage software on existing servers but they can also be used for provisioning servers, along with the rest of your infrastructure, like load balancers, databases, networking configurations etc. Configuration management and provisioning are not mutually exclusive as most configuration management tools can be used to provision servers to some degree but it’s not what they are truly intended for. For true IaC provisioning, Amazon Web Services (AWS) offers a closed source service known as CloudFormation and Terraform but these services are out of scope for this course. CloudFormation and Terraform are true provisioning tools where their sole intention is to scaffold infrastructure and not so much take care of the configuration of the applications and services running on the server, like SSH.

The way in which Puppet, Chef and Ansible handle the configuration differs. **Puppet and Chef require a “Master” server** which is responsible for storing the state of your infrastructure. Every time you want to update a piece of infrastructure it requires a client (like a command line tool) to issue new commands to the master server. The master server is then responsible to push updates out to your infrastructure. There are some drawbacks with this model as you will accrue extra infrastructure, maintenance and security.

Ansible is neat and more like AWS’s CloudFormation as they are both masterless. Using Ansible as the example, it subverts the need for a master server to issue commands as Ansible has the concept of “playbooks” which allow you to write out the configurations in a YML file (just like our soon to be Travis file) and through instruction, it uses SSH to communicate directly with the servers that are to be configured.

## Docker

A true game changer in the world of DevOps. Docker brings the concept of application containerization which we briefly covered in last week's lecture. **Application containers allow a developer to bundle up an application with all the needed dependencies and ship it out as one isolated package containing only the bare necessities to run**. Using Node as an example, a Docker container for a Node app would contain Node dependencies, expose only the needed ports or services for the application to run along with a Linux kernel. All of this is configured with something called a Dockerfile.

Docker is often compared to a virtual machine but unlike a VM, it doesn’t require the creation of a whole virtual OS. Docker allows applications to use system level virtualization rather than the traditional hypervisor virtualization used by VM’s. The hypervisor virtualization runs on physical hardware or some sort of intermediate layer. Containers instead run user space on top of an operating systems kernel which makes for very fast and extremely lightweight products. **Another huge advantage to containerization is its use in the build pipeline and the ability to create isolated microservices.**

Along with Docker comes Docker Hub, Docker Hub is like GitHub in a sense where it’s an online repository that hosts configuration files that help developers create an application contain quite easily.

## Kubernetes

With all the awesome advantages Docker brought developers, it produced a new problem that needed solving. Think about our conversation about microservices last week, if we had containerized applications each hosting its own isolated functionality, how do they all work in unison? If one were to go down, how would it affect the rest of the services and how would the issue get resolved? This is where Kubernetes comes in, providing a service known as “container orchestration”.

Kubernetes also helps manage the automation of deployment, scaling, networking and so much more. Kubernetes has the concept of “Pod’s” which is a group of more than one container, using Docker as an example, 3 Docker containers sharing storage or a network would be considered a Pod. Containers within a Pod share an IP address and port space and can find each other via localhost. It’s essentially a group of containers with shared namespaces and filesystem volumes. Pods provide a pattern of multiple cooperating processes which form a cohesive unit of service.

Pods are just one benefit when using Kubernetes with Docker but it deserves its own attention as essentially, they are responsible for the simplification of application deployment and management by providing a higher-level of abstraction. They serve as a unit of deployment, horizontal scaling and also replication. We won’t be getting much deeper than the concept of a Kubernetes pod as it’s outside the scope for this course. Just remember that **Kubernetes is responsible for application container orchestration.**

## Nagios

Considered to be the industry standard when it comes to monitoring solutions for infrastructure regardless of platform specificities. As we know, monitoring is a crucial component for any solid DevOps strategy. With Nagios, teams can be notified for discrepancies found in network traffic, servers, applications and services to reduce downtime for application users. Like Jenkins, it’s a pretty robust platform that offers a dozens of open-source plugins to make setting up a cinch.

Depending on business needs they offer different solutions. Nagios provides visualizations of scalability which really helps solve problems that come when multiple servers are at work. Network management is one thing that is extremely simplified through the centralization of server data that may be coming from all over the world. It also provides the ability to quickly audit log data from any system which can often be a pain-staking yet critical process when troubleshooting or forecasting infrastructure changes.

When working in an enterprise environment it is quite common to find a subscription service such as Nagios in place to help with monitoring. Nagios prides itself in being a one-size-fits-all solution for IT monitoring and the industry heavyweights that rely on Nagios for their high demands seem to concur.