CloudCheckr

Essential Training for the DevOps Engineer

How To Build A Career and Training Strategy

So You Want to Be a DevOps Engineer

Who hasn't heard of DevOps in the IT industry? The concept is decades old, but the branding and the buzz of the title has only reached peak visibility in the past five years. It rings in the ears of every C-level executive: "We need: big data, containers, machine learning, DevOps ..."

It's not just a buzzword. It's not even a specific job. It's a work paradigm that brings together traditional IT development methods and operational administration, and merges them into one effort. DevOps breaks down bureaucratic walls which may exist in a company and removes finger pointing like "that's on the developer," or, "a sysadmin should fix that."

Most DevOps engineers were either developers or system administrators. They made a choice or stepped into a role that bridged the divide. If you are one of those professionals who has yet to cross the bridge to experience both disciplines, or perhaps are new to the industry, there is a lot to know. Here are some answers to get you on your way.

There Is No Silo

If you are a Linux administrator, a Windows mail sysadmin, a Java developer, or a PHP front-end programmer, you are no longer those things. Those are just things you can do. Being a DevOps engineer means that you will likely work with all the technologies in your stack. You'll write tests, you'll commit code, you'll create pipelines, and you'll automate server builds. Your focus will shift from managing your previous slice of the full pie to focus on deploying the entire solution through automation.

What if you don't know all the technologies in the stack? What if you've only worked in traditional data centers and haven't worked in the cloud yet? What if you've been a database administrator for 10 years and this all sounds like too much?

It doesn't have to be, at all. Training content is free or very inexpensive. In CloudCheckr's companion article in this series, "Essential Training for the Cloud Architect," there is a section highlighting popular online training vendors. The same sites are perfectly relevant for DevOps training. It's time to invest in yourself and dig your way out of the silo.

Vendor	Free Trial?	Mobile App?	Per month	Content
CloudAcademy	Yes	Yes	$59	V, L, Q, E
LinuxAcademy	Yes	Yes	$29	V, L, Q, E
Udemy	No	Yes	Per course	V - huge library
A Cloud Guru	Yes	Yes	$29	V, L, Q, E
Key	***V: Video***	***L: Labs***	***Q: Quizzes***	***E: Exams***

You Are the Automator

What does this mean? Here is a scenario:

At Company A, Tina the Windows administrator needs to deploy 10 additional Microsoft IIS servers for a web farm. She uses a VMware OVA that can bootstrap to the domain. When each server comes online, it attaches to SCCM and receives patches for the next hour. After multiple reboots, she uses a script on each server to add services such as WWW and IIS, mounts a network drive for content, loads certificates, and then she begins testing. She manually tests the functionality of each server until she is satisfied with the result, and then adds them to the web farm. The process can take about five hours.

This week is Black Friday and she needs to add 1,000 servers to meet the load. Sound like a nightmare? Of course that isn't going to happen. In most cases, companies would need to have those extra servers ready all year for the load. That is unnecessarily expensive for the cloud era.

At Company B, Minnie the DevOps engineer spends two weeks writing RSpec tests to outline the desired outcome of her configuration end state. She creates Puppet modules to load what is needed for a bare image, runs it through SCCM, syspreps, and then takes an AMI. She writes Puppet modules for configuring the services, mounting the drives, loading the content, and for adding it to the web farm. Next, she writes Beaker integration tests and runs them against Docker to locally test that all the code actually configures everything correctly. All of these tests and modules are checked into a code repository where it can be revised and reused.

Minnie then creates a deployment pipeline that cuts a release for her code, and a deployment AMI for an Auto Scaling Group in a Blue/Green environment. Minnie increases the web servers by 10. They take roughly 10 to 15 minutes to come online after being launched in AWS. It's Black Friday again. Minnie increases the scaling to 1,000 before the event. It takes roughly 10 to 15 minutes for the additional servers to come online (with some variance on the backend) and she doesn't worry about manually testing anything. Let's say 50 servers didn't provision correctly. So she terminates them and auto scaling recreates the next batch. Then, she writes a test and places it in the pipeline to check if any server didn't provision within 25 minutes to terminate. Seeing a trend?

Testing

If you are in operations moving toward DevOps, writing tests will likely be a new concept. Developers write tests and so do DevOps engineers, so it is important to understand why and what testing entails. Simply put, you will learn to automate using Test Driven Development (TDD). There are many testing frameworks out there, such as RSpec, Cucumber, Swagger, Serverspec, Pester, and Testkitchen; the list is quite extensive. Rather than go into any of them in depth, it's important to understand two very important testing concepts.

Business Driven Development (BDD) is the process of creating tests that satisfy business requirements. These tests are written prior to writing the actual code and often lay out what the code should do. These tests will pass when your code meets the what but they do not verify if automation, in fact, works.

Following are some examples of BBD tests:

- All patches should install without error.
- Microsoft IIS should be installed.
- Microsoft IIS should be running.
- HTTP should redirect to HTTPS.
- HTTPS calls against the webserver should return with a 200 (success).

Notice the “should” language. These tests perform tests in memory, making assumptions on how the code is constructed.

Integration (Acceptance) tests actually verify that what you coded actually did configure correctly. This testing can remove the need for extensive manual regression testing and is broken down by a unit of code. Following are some examples of Integration tests:

- 50 out of 50 patches installed successfully (exited with 0).
- Microsoft IIS is installed.
- Microsoft IIS running on PID 3400.

Always write tests first. Always run all tests before committing code. Always commit code often. Think like a developer!

Infrastructure as Code

Infrastructure as Code (IAC) is the bread and butter of a DevOps engineer. In the cloud, the network, servers, and services can all be manually created, but that would be following the pattern of old. If you resorted to manual configuration, then it wouldn’t be testable for changes, accurately repeatable, or easily tracked. If you built an entire business in AWS’ US East (Northern Virginia) region and needed to reproduce the entire offering in the US West (Oregon) region, you wouldn’t want to reproduce it manually.

The IAC tools are answers to that problem. AWS Cloudformation, Azure Resource Manager, and Google Cloud Deployment Manager can be used to create deployment templates. Those templates can do anything from configure the network access control lists, provision servers, create file services, launch databases, and set up message queues or containers—pretty much everything that is offered in each cloud. That said, the deployment templates are vendor specific. That is fine for one cloud, but if you're managing multiple clouds, it becomes two completely different code constructs to maintain.

Hashicorp's Terraform provides a cloud-agnostic tool to manage code from a single pane of glass. It is quick and easy to get running across many platforms. All these tools are great at managing the infrastructure, but what about the configuration of servers themselves? Believe it or not, the tools can be used for server configuration—however, it is not their strength. It's better to use Orchestrators and Desired State configuration tools. The most popular tools are Puppet, Chef, Ansible, and SaltStack. There is an excellent blogpost by Gruntwork comparing these tools. Interestingly enough, all four of the tools can provision and manage cloud infrastructure as well. It's up to you to decide what mix of technology is necessary to properly control and automate all of your services

Continuous Integration/Continuous Deployment

This is one of the most important and desired DevOps paradigms for businesses. In short, CI/ CD means testing, provisioning, and shipping code into an environment in an automated way. The "CD" is also frequently called "Continuous Delivery," which is a bit different than deployment. Continuous Integration means automatically shipping new code into a non-production environment. Continuous Delivery is bringing code through the pipe, and a manual step (often a button push) sends it to production. Continuous Deployment is shipping code from testing all the way through environments, and ending in production without any intervention. It is both powerful and dangerous. Continuous Delivery requires full end-to-end acceptance testing, as well as fully repeatable and identical stacks across environments. In other words, the only differences are environmental settings and configurations like passwords. It also means no manual checks and configurations. If you want to ship to production with confidence, your quality assurance is in the code and not in a QA-

engineer regression test sheet. There are many tools to manage CI/CD "pipelines." Jenkins is one of the most popular, but CircleCI, TravisCI, and Puppet Pipelines (formerly Distelli) have robust and cloud-hosted suites to achieve end-to-end automation.

These Are Not the Pets You're Looking For

Randy Bias coined an excellent phrase in the industry—"pets vs cattle"—and DevOps embraces this concept. When you are in the cloud and managing servers as a DevOps engineer, you shouldn't know the server names. You should know them by service. The idea behind the saying is that the admins of yesteryear knew their environment and servers so well, they treated them like pets. They knew which servers had quirks, which ones tended to crash, and which ones had special cron jobs. This familiarity does not scale. Unless the admin has a photographic memory, they will not know fine details of 500 or 5,000 servers. If a server is misbehaving, it is put to pasture so another replaces it. They are like cattle and they are maintained by service.

Unlike pets, it takes a lot of work to breed cattle. A short list of design paradigms includes:

- **Build security first**: Use automation to improve and add security on your services before deploying them. Never plan to add security later.
- **Building stateless:** This entails removing (or automating management of) the state of servers (such as agent IDs) and the state of application transactions. An application or web server should be picking up messages from queues or in a database, or a cache, and process the request. That way, if a processing server fails, another server picks up the state and processes the transaction.
- **Decoupling the monoliths:** Placing multiple applications on a server means that a single failure becomes multiple failures. It also means that if you need to scale one piece of a server, all of it scales—which creates waste and inflates

costs. To prevent this from occurring, break servers down into microservices, reducing them to a single service or container. In addition to helping speed troubleshooting, you also can scale exactly what you need.

- **Removing bottlenecks:** Every IT professional has heard this. The difference here is if you haven't scaled your environment to see what new bottlenecks occur, it can become a major problem when you need to scale for real. Some configurations may need to change when you're scaling a service from 10 servers to 500.
- **Full automation-ready to work:** When your servers scale up, they should be either a configured image, or use an IAC tool to fully configure and leave the server ready to handle traffic.
- **Ship all your logs:** Get in the mindset of shipping your logs. They can be used to troubleshoot, to perform trend analysis, and to review for compliance. Don't leave critical logs on servers— especially if they scale up and down—or they will be gone.

Summary

Does being a DevOps engineer sound awesome? It should, because it is awesome. Even though you are on your way to becoming an automation expert, it is still best to find valuable partnerships and use powerful tools during the journey. It is not always best to design solutions from the ground up. After all, you're creating, reusing, and contributing code. When looking for help to maintain your compliance, security, resource utilization and more in the stack through automation, look to CloudCheckr to help you to get your solutions to production faster while and prioritizing both compliance and security.

Need CloudCheckr for your organization?
Learn more at www.cloudcheckr.com.

CloudCheckr

About CloudCheckr

We deliver total visibility—across multiple public clouds and hybrid workloads—making the most complex cloud infrastructures easy to manage. From government agencies to large enterprise and managed service providers, CloudCheckr customers deploy our SaaS-based platform to secure, manage, and govern the most sensitive environments in the world. Our industry-leading products include CloudCheckr CMx, CMx High Security, CMx Federal, and CloudCheckr Finance Manager for cost management, billing & invoicing, cloud security, compliance, inventory & utilization, and cloud automation.

CLOUDCHECKR.COM