Linux Academy

DevOps Essentials

An Introduction

About This Course

- This course is designed to introduce you to the concept of DevOps in general as well as how it evolved
- We are going to cover terms and technology that either directly influence the DevOps movement or led us down that path
- DevOps is nothing without understanding some of the fundamental processes that are impacted by the new way of operating, we will cover those as well
- Not only is DevOps the driver of many new tools in the Open Source world, but a large consumer of them as well. We will talk about them and point out those we currently offer training on at Linux Academy

What is DevOps?

Although it may mean different things to different people, there is an official definition that we are going to use, from Wikipedia:

…is a software development method that stresses communication, collaboration (information sharing and web service usage), integration, automation, and measurement of cooperation between software developers and other information-technology (IT) professionals.

DevOps acknowledges the interdependence of software development, quality assurance, and IT operations, and aims to help an organization rapidly produce software products and services and to improve operations performance.

http://en.wikipedia.org/wiki/DevOps

So What Is It Really?

Well, it's a different way of doing things in Information Technology. What it is NOT is an excuse for staff reductions (which all too often it is used to justify or explain).

What it is exactly and how it came to be, we are going to talk about at length over this course. You will hear things like "continuous integration" and "build automation" and "treating your infrastructure like code".

Here at Linux Academy, we support the formal definition of DevOps and many of our courses are built around training you on the tools and processes that you will need in the new DevOps world!

Summary

Why are we spending the time on DevOps? Quite frankly, because it is here to stay and honestly, when applied correctly, it addresses traditional short comings and obstacles that we have all run into during our careers in Information Technology.

Stick around, we are going to talk about a little history first!

Linux Academy

DevOps Essentials

Traditional Responsibility Silos

Information Technology - Operations

Operations is simply the set of processes and services provisioned by IT personnel to their own internal or external clients in order to run their business.

It is generally delineated in several ways:

- Infrastructure and Monitoring
- Architecture and Planning
- Maintenance
- Support

Many times, the duties are limited to physical and virtual hardware and the "who and how" it is provisioned to provide services consumed by development.

Information Technology - Development

Although not necessarily exclusively, in IT, development generally refers to the process of creating software. It involves the programming, documenting, testing and debugging associated with application development and the associated software release lifecycle.

There are a number of methodologies for doing so:

- Prototyping
- Waterfall
- Agile
- Rapid

Just to name a few. These are the practices that define the activities around how software is developed and the order in which they occur.

Silos of Responsibility

In the past, these two beasts lived in the same zoo, but were separated and fed separately. That sounds strange, but it’s a fair analogy.

It is fair to say that until relatively recently, these two activities required COMPLETELY different sets of skills. Hardware and networking were not necessarily well understood by skilled software developers, and software development not well done by hardware staff.

When did that begin to change? Well, probably when things got a little bit “cloudy”.

Breaking Down Barriers

In about 2008, when Agile software development began to gain steam as a methodology, the concept of DevOps was introduced.

Around 2010 as Amazon’s relatively new “internet datacenter” became more popular, the skillsets of both of these silos began to converge.

Now that anyone could provision “images” to use for rapid development and prototyping, the skills necessary to manage those configurations started to be more well understood in general. These “crossover” skills began to create a new type of engineer that was exactly as described and those barriers began to break down.

Summary

Although the need for these barriers to be broken down was evident, there are those that have taken advantage of this new paradigm.

Routinely, companies are using the DevOps rallying cry as an excuse for not hiring the staff needed. They term it a new "culture" when sometimes all it is is an excuse for them to get more out of fewer personnel.

What it CAN be is an enabler for the business. Rapid development lifecycles are key in the internet age. Your ability to deploy new features on your site or in your application in a constant manner can be the difference between succeeding or failing in the market.

Linux Academy

DevOps Essentials

Terminology: IaaS

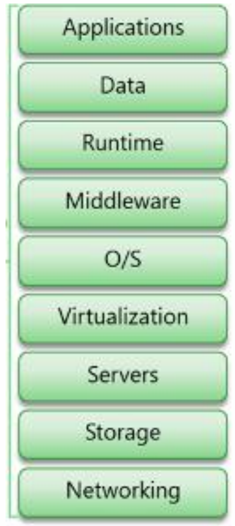
Terminology: IaaS?

One of the first steps in the DevOps revolution was when Information Technology began to look at it's Infrastructure differently.

IT in general, operations specifically, was always seen as a "cost center". A necessary budgetary evil that every company had to accept but was not well understood or appreciated.

Smart executives in IT began to change the model of how Infrastructure was consumed. It started with charging Infrastructure costs back to the business units that consumed them. In short, IT began ***I**nfrastructure **as a** **S**ervice.*

Traditional Services

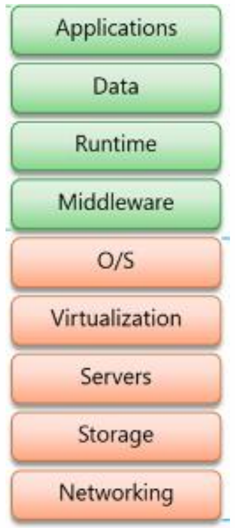


The traditional "stack" of technology services and the management of them looks something like the diagram to the left.

You are looking at a generalized view of everything from hardware to operating system to the data and applications that sit on top.

Whether the Infrastructure was provided internally or in a "colocation" model by a third party, IaaS took a different look at it.

Traditional Services – IaaS View



Here is the same "stack" with a slight difference. The perspective of what IT provides is now defined as a set of services related to those items highlighted, from the network to the operating system.

Now, infrastructure related items could be clearly identified to the consumers of those services. This was important because it was one of the first steps in defining IT lines of responsibility differently.

Summary

Infrastructure as a Service is now a routine term in the current IT nomenclature. It is a level of service and support that is used to clearly identify where the responsibility starts and ends when providing infrastructure to its consumers (be it the business directly or the developers who need to deploy on it).

Although this was not the driving force behind DevOps, it certainly has influenced the services and skills that DevOps requires. As virtualization has overtaken dedicated hardware resources in particular, IT has had to develop different (and faster) skills around the scale and speed of traditional infrastructure deployments.

Linux Academy

DevOps Essentials

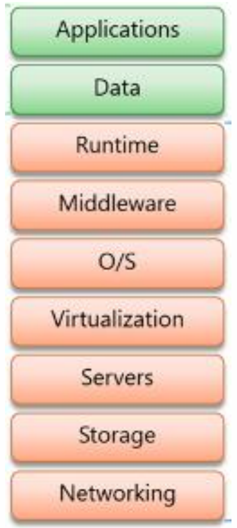
Terminology: PaaS

Terminology: PaaS?

Now we go further down the rabbit hole… the next step up in the redefinition of IT was to take a look at the Platform being delivered and how it was managed (and by whom).

In the ***P**latform **as a S**ervice* model, IT (or a vendor/cloud provider) delivers a “computing platform” for consumption. It generally includes everything from the previously detailed IaaS model as well as a few additions.

Traditional Services – PaaS View



You will notice that we have moved the “service” offering a few notches up the stack in the diagram to your left.

In addition to everything in the IaaS model, we have added Runtime and Middleware.

Sample platforms are database, web servers, runtimes, etc. These are independent of the Infrastructure but pushes the service offering up and takes more of the burden off the business or the developers to manage.

Why Is PaaS Important?

Again, although not the driving force behind the DevOps movement, it had significant influence as a further evolution of how IT was traditionally thought of.

Big name vendors like Microsoft (Azure) and Google (App Engine) were some of the first to offer the underlying computer and storage resources that could scale automatically to match application demand so manual allocation of resources was no longer necessary.

It also converged developer and operations skillsets even more than before.

Summary

Platform as a Service is now a routine term in the current IT nomenclature. Now that management of the stack has moved further up (or down depending on your view point), the skills necessary for each portion of that delineation have converged even more.

Although this was not the driving force behind DevOps, it certainly has influenced the services and skills that DevOps requires. Now that anyone can write and deploy software publicly on compute resources that were traditionally managed by operations only, we start to see how this evolution set us on the path to DevOps.

Linux Academy

DevOps Essentials

Terminology: SaaS

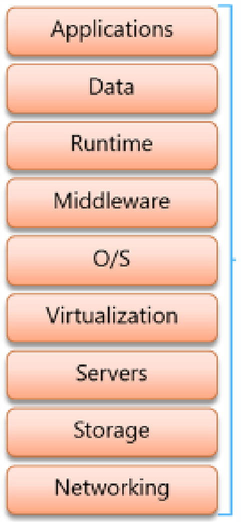
Terminology: SaaS (or “ugh not another one”)?

As someone in a famous science fictional movie may have once said “Now your journey to the dark side is complete.”

In the ***S****oftware* ***as a S****ervice* model, IT (or a vendor/cloud provider) delivers ACCESS to the software to be used without having to do anything to manage, configure, monitor or support it.

This eliminates the need to install or run applications locally and can be run entirely remotely (“in the cloud” so to speak). Since everything in the traditional stack is now consumed as a service, those lines, well, they disappear.

Traditional Services – SaaS View



Now, there doesn’t seem to be any “lines drawn in the sand”.

In addition to everything in the IaaS and PaaS model, we have Data and Applications.

The evolution has come full circle so to speak. Instead of the traditional lines drawn between software and hardware (often seen as operations and development) disappear, the skills needed to operate within this space have completely converged.

Why Is SaaS Important?

You are right, SaaS was not “the” driving force behind DevOps, but it was one of the most important final steps.

Virtualization and cloud technologies require automation in order to provision quickly enough for the service to be readily consumable. In order to do that, a ton of software has been written to manage those compute resources and allow the automatic scaling based on need.

We now have traditional software and hardware personnel that need the same skills to operate within this space.

Summary

Software as a Service is now a routine term in the current IT nomenclature. This evolution of how IT offers and consumes hardware and software turned the industry on its head.

All of these “service” platforms that we have talked about, drive cloud technology and those “cloud clients” are now the consumers where IT has converged in how it provides hardware (now largely virtualized) and software to be consumed by end users.

The skills necessary to manage these resources now cross over these previously traditional silos. No longer do we have the software developer who doesn’t understand basic networking or the hardware engineer who cannot develop, they are required skills and contained in the same space.

Linux Academy

DevOps Essentials

Process: Build Automation

Build Automation - Basics

Historically, the term has applied to software development. The process of "building" or compiling software that can then be deployed via script or cron jobs to various environments, including production systems.

In the DevOps world, it encompasses not only the software portion, but the process of automating the deployment of compute resources (physical or virtual, applications and data).

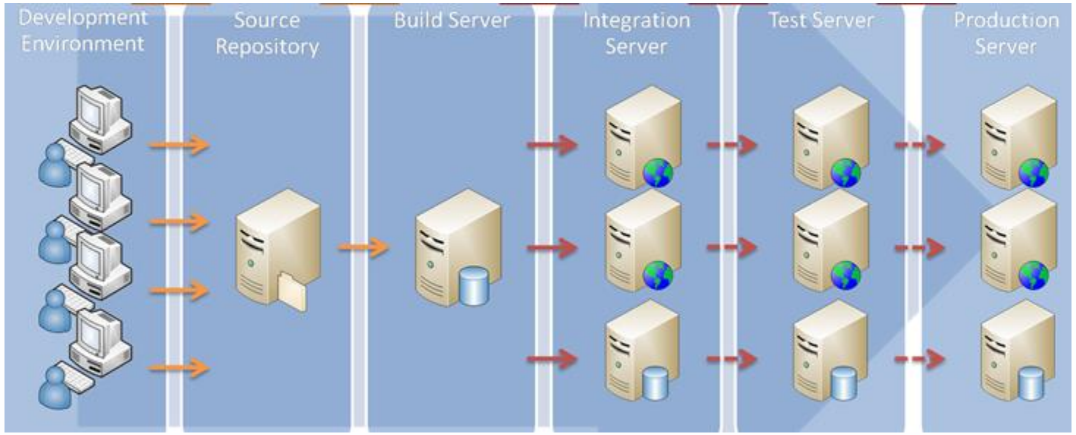
Whether the process is a unified tool or a set of them, build automation in DevOps terms allows the deployment and management of the entire stack of services, without manual intervention (well, mostly).

Treat Your Infrastructure As Code

DevOps erases those traditional lines between Operations and Development. It does so because everything is treated as a “compute resource” and can be managed with code.

When your compute resources are largely virtual (cloud consumed), your deployments can be automated throughout the stack with build automation tools.

What Does It Look Like?



Actually, it looks like this. Build automation is the process by which you initiate a software or hardware deployment automatically, using consistent methods, all the way through the environment stack.

It can and does include automated testing and rollback capabilities so that each environment remains stable and consistent.

Summary

Build automation is a key component in any DevOps organization. It consumes the services that IT has painstakingly made available in a consistent and repeatable manner.

Now that our compute resources (be they virtual or otherwise) can be consumed and expanded instantly and automatically, the build process can take advantage of those resources more efficiently.

Consistency and stability is the key in Build Automation. By removing the manual process necessary to deploy hardware and software, you eliminate potential inconsistencies amongst the environments and reduce troubleshooting time when there is a problem since rollback and new deployments are trivial.

Linux Academy

DevOps Essentials

Process: Continuous Integration And Continuous Delivery

Definition: Continuous Integration

This can be defined as the practice of merging development working copies (i.e. builds that are in flight on a development system or systems) with the shared source main (branch) multiple times per day.

The concept of multiple integrations per day on the main source branch is to prevent integration problems in large development teams where the odds of one change breaking the changes of another developer would be smaller.

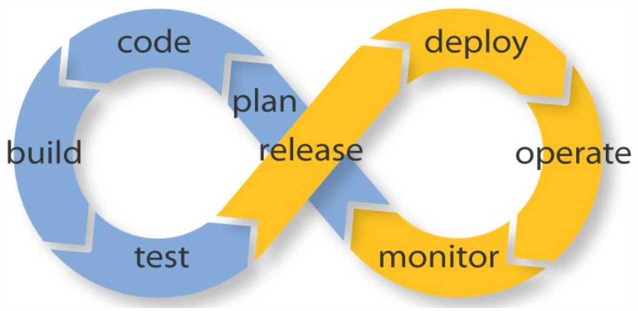
Continuous Integration works hand-in-hand with the previously discussed Build Automation. It takes advantage of those automated build processes to build and test each commit and reporting those results back to the development teams.

Definition: Continuous Delivery

This is an approach in which software teams keep producing valuable software in very short delivery cycles and ensures that those features can be reliably and consistently released at any point in time.

It is often mistakenly used interchangeably with the Continuous Integration approach. Whereas CI has more to do with how the software code is managed throughout the development lifecycle, CD is how valuable and how quickly that software can be released when it is determined that the aggregate features are valuable enough.

What Does It Look Like?



These topics are very cyclical in nature. Both are designed to take advantage of taking things in smaller chunks to increase stability and decrease release cycles.

As one progresses constantly, the outputs from the CI process feed into the CD process and the cycle begins again.

DevOps is driven largely by these two processes as they are the culmination of treating everything like code in rapid cycles.

Summary

Although related, Continuous Integration and Continuous Delivery are different in what they accomplish.

DevOps uses each one in and in turn feeds the next in the chain. Integration in small doses keeps each environment from developing competing relationships too quickly to manage. Delivery allows the reliable release of those small doses when they are determined to have enough value to move into production.

Since everything is treated as code from the infrastructure to the software being deployed, the DevOps tools being used apply equally to each part of the stack – we are going to talk about those very tools next up!

Linux Academy

DevOps Essentials

Tools: Jenkins

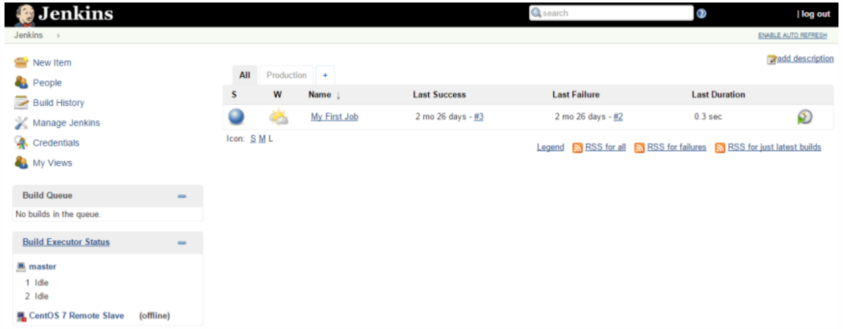
What is Jenkins?

In short, Jenkins is Build Automation on steroids. It also can be considered a Continuous Integration tool. Often considered a Continuous Deployment tool as well.

Wow. Although used more often in software development for build deployments, it supports a large number of plugins that enable anything from deploying scripts to launch virtual machines through VMWare or Vagrant to Docker containers across environments.

Was originally called Hudson in 2004 and started at Sun Microsystems. After big bad Oracle bought Sun and claimed the name, it was changed to Jenkins in 2011.

What Does It Look Like?



S	W	Name ↓	Last Success	Last Failure	Last Duration
		My First Job	2 mo 26 days - #3	2 mo 26 days - #2	0.3 sec

Build Queue

No builds in the queue.

Build Executor Status

master
1 Idle
2 Idle
CentOS 7 Remote Slave (offline)

Jenkins allows you to create build jobs that do anything from deploying a simple software build to the custom creation of a Docker container with specific build branches while doing performance and unit testing while reporting results back to the team

Most importantly, it does all of that WELL. The variety of plugins you can download and install as well as the way you can manage and distribute the build load across multiple slave servers ensures that you have a robust and performant environment for automating all facets of your build, integration and deployment process.

Where Can I Get It and How Do I Learn?

As far as the where, go to the Jenkins site at https://Jenkins-ci.org and download either the latest version or the current LTS (Long Term Support) version.

For training on its use in a DevOps environment, you can stay right here at Linux Academy. Navigate to our “DevOps” set of courses from the main page and check out “Jenkins and Build Automation” from there. We have more than three hours of Jenkins related DevOps training for you covering everything from installation and configuration to setting up full parameterized builds using plugins and build slaves!

Summary

The tools involved in DevOps can make or break a successful DevOps culture. By automating all of these integration and build tasks, anyone can work on stable environments and get consistent results.

Jenkins is one of the largest DevOps enabling tools available. It plays in every field across the board for deployments, integration, testing and builds.

If you had to make a list of the top three DevOps tools to use, Jenkins should absolutely make your list.

Linux Academy

DevOps Essentials

Tools: Docker

What is Docker?

According to Wikipedia (and others):

*Docker is an open-source project that automates the deployment of applications inside software containers, by providing an additional layer of abstraction and automation of operating system-level virtualization on Linux.**

So basically, it is a tool (or a set of tools depending on how you look at it) that packages up an application and all its dependencies in a "virtual container" so that it can be run on **any** Linux system or distribution.

** http://en.wikipedia.org/wiki/Docker_(software)*

Docker Architecture

Docker is a client-server application where both the daemon and client *can* be run on the same system or you can connect a Docker client with a remote Docker daemon.

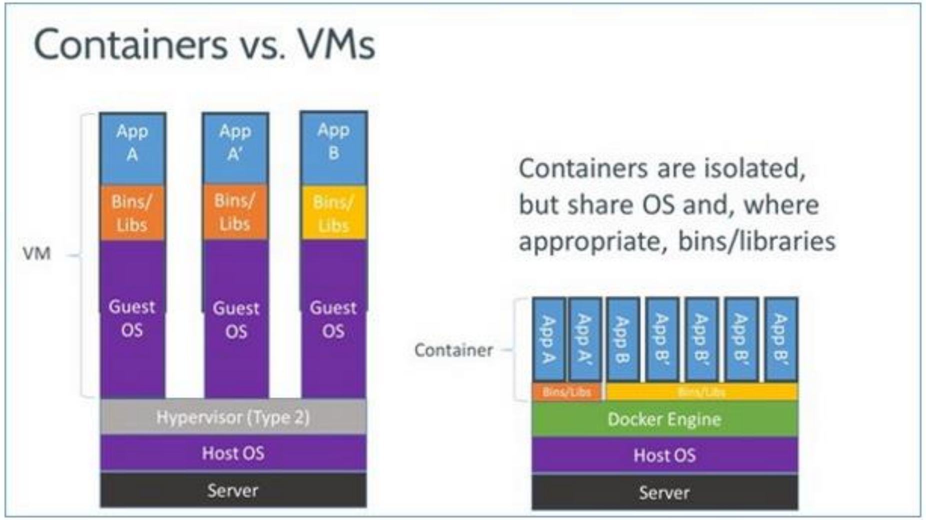
Docker clients and daemons communicate via sockets or through a RESTful API *(Representational State Transfer – it is a stateless transfer over HTTP of a web page containing an XML file that describes and includes the desired content).*

The main components of Docker are:

- Daemon
- Client
- Docker.io Registry

Container Architecture

Instead of virtualizing hardware, containers rest on top of a single Linux instance. This allows Docker to leave behind a lot of the bloat associated with a full hardware hypervisor. Here is a look at the architecture of each:



Don't mistake the *Docker Engine* as the equivalent of a hypervisor in a more traditional VM, it is simply the encapsulating process on the underlying system

When Would I Use Docker?

There are a lot of reasons to use Docker. Although you will generally hear about Docker used in conjunction with development and deployment of applications, there are a ton of examples for use:

- Configuration Simplification
- Enhance Developer Productivity
- Server Consolidation and Management
- Application Isolation
- Rapid Deployment
- Build Management

Keep in mind these are only a few use cases. We are going to explore many more during our course!

If it’s so great, why hasn’t anyone done it before?

Quite simply, they have. Containers are not a new concept in technology, it just appears that Docker has captured the buzz (right place, right time). If you look around, you will find that a number of companies and projects have been working on the concept of application virtualization for some time:

- FreeBSD – Jails
- Sun (and now Oracle) Solaris – Zones
- Google – lmctfy (Let Me Contain That For Your)
- LXC (Linux Containers)

We appear to have reached the point in time where software has caught up to hardware virtualization.

Summary

Docker offers you the ability to isolate your applications, standardize your build and deployment process and to create standard, repeatable processes in your software and infrastructure. We are going to take a look at Docker from the overall architecture to its use and application across all operating systems as well as exploring the use cases in Enterprise environments.

The architecture of Docker and the containers that it relies on are not new concepts, having been around since the early part of this century. However, hardware virtualization performance has now become almost indistinguishable from bare metal so that further virtualization on the technology stack can be realized.

Linux Academy

DevOps Essentials

Tools: Chef and Puppet

What is Chef?

It is a configuration management tool written in the Ruby and Erlang software languages. You create “recipes” using pure-Ruby (a domain specific language, or DSL).

Chef is most often used to streamline configuration tasks for maintaining servers and related infrastructure. It easily integrates with cloud-based platforms (AWS, Google Compute, Azure, OpenStack, etc) to provision and configure new “machines”, “images” or “instances” automatically.

It can also be used to configure almost anything that runs ON those instances as well.

What is Puppet?

Puppet is also a configuration management tool or utility that runs on Unix/Linux/Windows/OSX. It includes a custom “declarative” language that is used to “describe” system configurations and their state.

Although it uses its own declarative language, like Chef, you can also use a Ruby DSL (domain specific language). All of your configuration scripts are stored in files referred to as “Puppet Manifests”.

These manifests and scripts can then be applied directly to each system, or they can be compiled into a catalog. That catalog is that “published” to the targeted systems using a REST API.

Why are they important to DevOps?

These tools allow you to live one of the DevOps rallying cries – “treat your infrastructure as code”.

By abstracting the configuration needed to launch server instances with specific configurations into consistently repeatable “recipes” or “manifests”, the skillsets needed to manage the entire technology stack is converged – both operations and development skills are required.

Using one of these configuration management tools along with source control, Docker and/or Jenkins, you start to see how this all adds up to the DevOps processes we talked about earlier regarding Continuous Integration and Continuous Deployment.

Code Samples

CHEF

```
Vagrant::Config.run do |config|
  config.vm.box = “centos7_64"
   config.vm.provision :chef_solo do |chef|
     chef.cookbooks_path = “recipes"
     chef.add_recipe “apache_tomcat"
     chef.log_level = :debug
  end
end
```

PUPPET

```
# /home/user/manifests/init.pp

class sudo {
    file { "/etc/sudoers":
        owner => 'root',
        group => 'root',
        mode  => '0440',
    }
}
```

The Chef recipe uses Vagrant to spin up a CentOS 7 64 bit image. It then provisions itself, looks up the “recipes” path and adds (and executes) the recipe to install Tomcat. The Puppet manifest executes a change to the /etc/sudoers file from the /home/user/manifests directory on the user/owner/permissions of the file.

Summary

Both of these tools are important in the DevOps organization and they both can accomplish similar things, albeit in different ways using specific references (objects vs. procedures).

If you want to learn more, or better, show your company how valuable your DevOps skills are, go to the Linux Academy DevOps section and take the Puppet Certification course. This will prepare you completely for your Puppet Professional Certification Exam.

You will also find a Chef course and, taken together, you will get TWENTY FIVE hours of DevOps training that will make you a true DevOps professional!