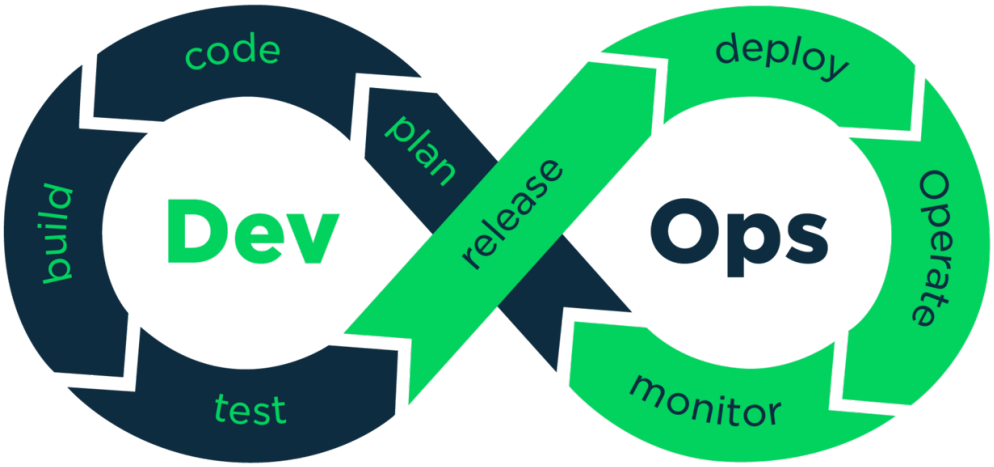


DevOps for Layman

Published on October 24, 2018

Aayush Shrut
Consultant at Deloitte

11 articles + Follow

We have come a long way in understanding complex cloud enabling technologies. We have understood what is Cloud, and its associated concepts of Virtualization. We then explored the de facto private cloud software called OpenStack. Applying same logic, we understood Containers, and how it is one step further then Virtualization. We then explored the de facto(s) Container softwares called Docker and Kubernetes.

In this article, we understand how this entire picture of Cloud, Virtualization and Containerization comes to a close with DevOps. We explore what is DevOps, why it is necessary, what are the important tools which enable DevOps, and most importantly, how to get a job in DevOps.

As usual, we try the "*Layman approach*", which is about using analogy, simple terms, and a sprinkle of humor to understand complex fancy terms. Just to make sure that there is an interest, a cheap marketing gimmick FYI: *Average salary of DevOps in USA is a massive $133,378 USD!*

Messaging

It's a tool, it's a phenonemon, it's a fancy term, it is, DEVOPS!

Like all big things in IT and Finance industry, DevOps had its humble origins by being a buzzword. People were excited by how a certain combination of tools was enabling faster delivery of software (and reducing delivery nightmares). This excitement soon led to a cultural revolution in IT, and with business folks getting involved, this revolution was coined a term called DevOps.

Loosely speaking, DevOps can be anything from "developer + operation collaboration", "automated delivery", "code/platform/function as a infrastructure", "toolchain approach to software delivery", "agile + lean software evolution", "matrix revolutions", etc. etc.

Strictly speaking, DevOps is yet another business strategy of underpaying developers by harboring additional responsibilities of operations, in disguise of fancy terms. Kind of same way how Full Stack Developers combined back end developers + front end developers, unfortunately without combining the pays. Seriously, look at any job posting for developers. Almost everyone demands knowledge of Docker, Kubernetes, CI/CD, DevOps etc. etc. Damn Business guys.

Accurately speaking, DevOps is defined as (according to WikiPedia):

> DevOps is a software development methodology that combines software development with information technology operations. The goal of DevOps is to shorten the systems development life cycle while also delivering features, fixes, and updates frequently in close alignment with business objectives.

Whatever DevOps is, there is no strict definition associated with it. We can however try to understand why it was needed in the first placed, besides its obvious benefit of cost savings.

Why DevOps was needed?

In the good old 90s, software was developed using the WaterFall model. People used to sit together in lengthy (and often pointless) discussions, chart out design, code, test, and finally, deliver. Obviously this process was really slow for fast moving business world. So to address this problem, came agile methodology. It introduced faster delivery time in the form of user stories and "sprint" sessions, and small, continuous (and incremental) development.

The problem with both was the increasing differences between Developers and Operations guys/gals. The developers were responsible for coding and developing features. The operators were responsible for setting up servers and integration environments, and testing the code. Sounds perfect, different departments, different verticals, the corporate style.

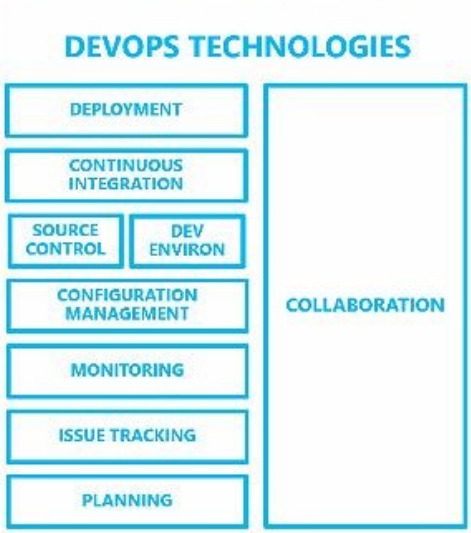
What actually happened was that developers often tested their code in their own environment (their laptop for example). The operations folks then had nightmares matching the production compatibility (lots of servers) with the development environment. It is like the game worked in your laptop, so you are assuming it must work in all other system (desktop, servers, etc.) of similar configuration. Insane right? Unfortunately that's how we worked so far.

Since a complete setup had so many moving parts, **diagnosing a problem was a nightmare**. This called for lots of unplanned work (weekend overtime without pay) for operations folks in executing a delivery. The developers also got stalled with their progress, as they had to wait for previous problems to be resolved before they added new changes. Even worse, since so many developers committed so many changes, it became impossible to find out which changes from which end (operations or developers?) are causing the problem. The worst case scenario (which unfortunately was the most common) was the **constant infighting between developers and operations**, with managers shouting their throat out.

To address this problem, and to increase the comradeship between developers and operations, DevOps came into picture. At its heart, DevOps follows following technologies:

 Messaging

- **Collaboration technologies.** Help teams work together more easily, regardless of location.
- **Planning technologies.** Provide transparency to stakeholders.
- **Issue tracking technologies.** Increase responsiveness and visibility.
- **Monitoring technologies.** Clear, shared responsibility for relevant parts of service health.
- **Configuration management technologies.** Enforcing desired state and consistency at scale.
- **Source control.** Accessible, controlled means for storing key assets.
- **Development environment technologies.** Accelerating development by reducing setup time and inconsistencies.
- **Continuous integration technologies.** Instant feedback by merging code regularly.
- **Deployment technologies.** Tools for building out environments and regularly updating systems.



Right now, with increasing adoption of containers, virtualization, and sophisticated tools enabling faster delivery, DevOps in practice has became easier. Now, with click of a button, developers can test their code in a common setup (no more laptops). With containers, operations folks can deploy it anywhere. With monitoring tools, finding the bottlenecks has become much much easier. So all these principles mentioned above, they have become a reality thanks to technology.

Think of DevOps as a modernization of the great IT factory. Just like in a modern factory, if an order is placed, the raw materials roll in the conveyor belt, and you get the finished product on their way to the client. In the same way, if client demands certain changes (like those usual banner resizing requests), a developer can quickly code in the changes, test them, and deploy them across all the servers hosting the client's software. This ultra fast deployment (and any tools or technology enabling it) is in essence what DevOps is all about.

In fact, Amazon shares their stats for a month of deployment using DevOps:

1. **11.6 seconds** : Mean time between deployments (weekday)
2. **1,079** : Max # of deployments in a single hour
3. **10,000** : Mean # of hosts simultaneously receiving

Messaging

4. **30,000** : Max # of hosts simultaneously receiving a deployment

From 1 day to 11.6 seconds, there is no other compelling reason for any enterprise to not adopt DevOps practice.

What are the tools enabling DevOps?

Just like the modern factory revolution happened because of sophisticated machines (LMWS pulsed fiber laser welding system anyone?), **DevOps happens because of certain tools**. Many folks confuse DevOps with toolchains, but it is not. Always remember, DevOps in principle is faster delivery of software. DevOps in practice is using tools to enable just that. Let's discuss a few of the most important tool chains.

1.) Version Control: GIT

In a modern factory, you need certain tagging mechanism to know which raw material is arriving from where. Like for example, you can tag your oranges as level 1: China, level 2: India. Now, if level 1 oranges turn sour in quality test, you can quickly revert to level 2 oranges.

In the same way, you need some software to manage different versions of source code (the raw materials for IT industry). So if some problem arises in some version (or tags), you can know exactly which version of the code is causing problem, and you can easily revert back to earlier changes.

GIT serves as the industry standard for version control. Other popular tools are *Perforce, Subversion* etc.

2.) Configuration Orchestration and Management: Terraform and Ansible

In a modern factory, we need some mechanisms to quickly expand our machinery without affecting workflow. Suppose, demand for new orders increase exponentially. So we must have a mechanism to quickly fit in more conveyor belts, more sophisticated machines (Mega New Baby Multipurpose Collet Chuck 25N anyone?), and also manage the configuration of our factory machinery (extra drilling power, faster conveyor belts).

In an almost same way, we need tools to quickly scale our IT factory servers and their configuration. This follows ***Infrastru***

Messaging

principle. There are two sides of it. One is **configuration orchestration (CO)**, which simply means automating deployment of more servers (similar to acquiring new machines). Second is **configuration management (CM)**, which means automating configuration of build environments on the existing machines (similar to configuring drill machine power based on demand).

Both terms are usually used together as some tools perform orchestration and management both, but it is important to know the difference. CM and CO are what most people relate with DevOps, automating and managing servers at scale. Usually, using a master server, we can code our way to create any infrastructure with any configuration, and manage them at any time.

So if a client demands we need Windows OS, Direct X 11, Python library and Fancy Framework #1, we can quickly create a server with these requirements without any variance. In a way, our laptop (or developer environment) configuration is replicated exactly across different servers. No more compatibility nightmare or "it worked on my system excuse" yay!

Let's just take a moment to appreciate how far we have come from the Iron Age of server management. Just 10 years back, configuring a server to test your software used to take weeks (purchasing, cabling, setting up it in racks, installing software, fixing networking, debugging sudden issues etc. etc.). Now? Minutes. Technology!

Ansible is one of the tools for configuration management. It is agentless, uses SSH, and has python instead of proprietary language (Puppet for eg.). Other such tools are *Chef, Puppet, SaltStack* etc. *Terraform* is a tool for configuration orchestration. It is cloud agnostic, which means it can be used to deploy servers on any cloud provider. Other tool is *AWS Cloudformation*.

3.) Continuous Integration and Continuous Delivery (CI/CD): Jenkins

In a modern factory, you need several raw materials to mix fast (water + sugar + carbon + optional secret coke recipe), tested quickly for quality control, and be packaged in the end. This happens almost 24*7, in a well defined process running on a conveyor belt.

In the same way, the IT factory needs a mechanism where several

Messaging

developers are merging in their code (**Continuous Integration or CI**), with these combined changes being tested (on the created servers using CM) and quickly deployed on the production servers (**Continuous Delivery or CD**). So no more infighting between developers on who is merging what. Also, no more breakdowns in integration testing or Integration Hell, yay!

The best tool for it is *Jenkins*. It is very easy to use, has tons of plugins to chose any environment (or conveyor belt), and has a very active community. Other such tools are *Travis CI* and *Bamboo*.

4.) Monitoring: Nagios and App Dynamics.

This one is obvious. Without proper logging mechanisms, engineers will be blind men facing an army of archers while running on a tightrope. In other words, they will always fail. Not only we need proper understandable logs, we need them to be shared to the right people (developers, operators, managers etc.). Monitoring can also enable us to spot trends and prevent future overloads.

There are two kinds of monitoring tools, **Infrastructure monitoring** (for servers) and **Application monitoring** (for softwares). Market leaders of them are *Nagios* and *New Relic* respecively. Other such tools are *Zabbix, Sensu, Prometheus, SysDig* for IM, and *AppDynamics, Compuware APM & Boundary* for AM.

5.) (OPTIONAL) Project Management Tools: JIRA

While not entirely part of DevOps, project management tools helps a lot in reducing the infighting and knowing the progress of the project. These tools allow for the big project to be broken down into small components or tasks. Each tasks are assigned to engineers and are tracked with a proper reporting mechanism. It must be noted that these tools have been in existence long before DevOps revolution.

Putting it all together

We know the associated principles of DevOps. We know the tools enabling these principles. Now, let's try to put it all together, using the great Modern Factory as an example.

The big billion sale and black Friday sale demands for greater production of goods. This sets the factory manager in a frenzy. He first procures the raw materials from the warehouse (*github*), properly tags them according to quantity and quality (*git*), and then unleashes them into the factory floor to bring forth the finished goods. Before he can do that, he quickly sets up new machinery and extra conveyor belts with high speed and some other sophisticated customization (*Ansible, Configuration Management and Terraform, Configuration Orchestration*). He then unleashes his tagged raw materials on the factory floor.

Things are moving fast on the floor. Lots of different raw materials are quickly getting mixed into the finished product (*Continuous Integration*). The products themselves are being tested rigorously for quality control. Finally, they are all being dispatched with colorful packaging (*Continuous Delivery*). One smooth system, from production to delivery.

The factory manager is monitoring everything from his Master Control Panel (*Jenkins*). He controls the mixing, the testing machines, and the dispatch location. Suddenly, it shows that the products

Messaging

the tests and are causing his high speed conveyor belt to stop. Again, using his Master Control Panel (*Jenkins*) and tagging system (*git*), he realizes that Raw Material S9 v5.7 is causing the defect. He replaces that raw material with v4.2, and almost quickly, things return back to normal.

He uses other monitoring panels to know about the production quota, efficiency, outages, machine performance etc. (*Nagios*) and the performance of the finished products (*App Dynamics*). Finally, to coordinate efficiently with his workers, he uses another panel to chart out tasks and get updated status on them (*Jira*).

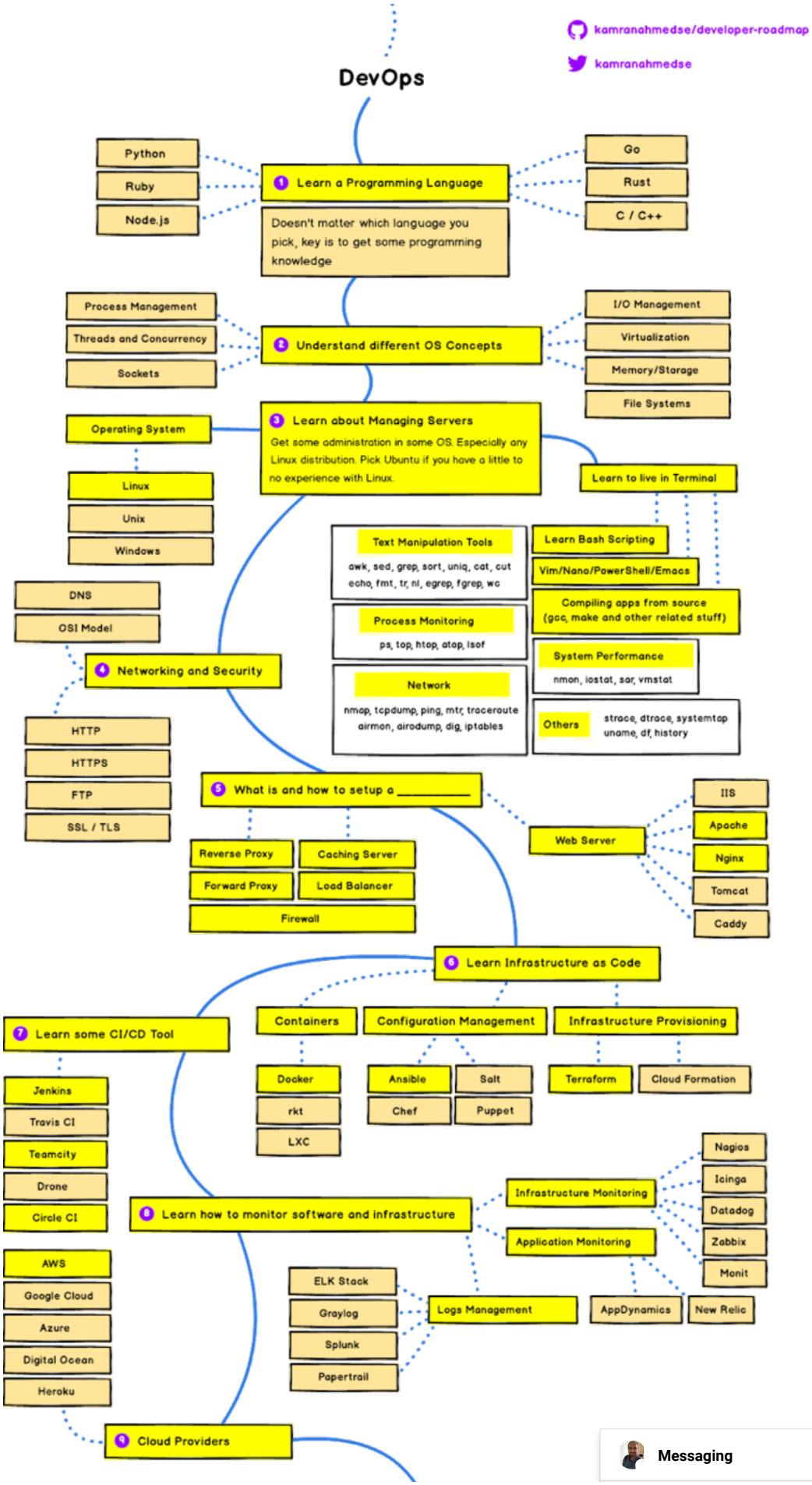
Suffice to say, our manager exceeds his production quota and receives a good promotion :).

So this is how DevOps works in a nutshell. We procure our source code, mix and tag them, use CM/CO tools to automate custom configured servers on the fly. We then use CI/CD tools to quickly test and deploy our code to production. Finally, we use monitoring and project management tools to have a bird's eye view on performance and tasks respectively.

Bottom Line: How to get job in DevOps

Friends from hackernoon has put the best answer, so I am just going to share the roadmap. A basic primer on the associated concepts is enough to become a good DevOps engineer.

 Messaging


kamranahmedse/developer-roadmap
kamranahmedse
DevOps
Python
Ruby
Node.js
1 Learn a Programming Language
Doesn't matter which language you pick, key is to get some programming knowledge
Go
Rust
C / C++
Process Management
Threads and Concurrency
Sockets
2 Understand different OS Concepts
I/O Management
Virtualization
Memory/Storage
File Systems
Operating System
3 Learn about Managing Servers
Get some administration in some OS. Especially any Linux distribution. Pick Ubuntu if you have a little to no experience with Linux.
Learn to live in Terminal
Linux
Unix
Windows
Text Manipulation Tools
awk, sed, grep, sort, uniq, cat, cut echo, fmt, tr, nl, egrep, fgrep, wc
Learn Bash Scripting
Vim/Nano/PowerShell/Emacs
Compiling apps from source (gcc, make and other related stuff)
Process Monitoring
ps, top, htop, atop, lsof
System Performance
nmon, iostat, sar, vmstat
Network
nmap, tcpdump, ping, mtr, traceroute airmon, airodump, dig, iptables
Others
strace, dtrace, systemtap uname, df, history
DNS
OSI Model
4 Networking and Security
HTTP
HTTPS
FTP
SSL / TLS
5 What is and how to setup a _____
Reverse Proxy
Caching Server
Forward Proxy
Load Balancer
Firewall
Web Server
IIS
Apache
Nginx
Tomcat
Caddy
6 Learn Infrastructure as Code
Containers
Configuration Management
Infrastructure Provisioning
Docker
rkt
LXC
Ansible
Salt
Chef
Puppet
Terraform
Cloud Formation
7 Learn some CI/CD Tool
Jenkins
Travis CI
Teamcity
Drone
Circle CI
8 Learn how to monitor software and infrastructure
Infrastructure Monitoring
Application Monitoring
Nagios
Icinga
Datadog
Zabbix
Monit
ELK Stack
Graylog
Splunk
Papertrail
Logs Management
AppDynamics
New Relic
AWS
Google Cloud
Azure
Digital Ocean
Heroku
9 Cloud Providers
Messaging

For those pesky interviews, the easiest way is to create a simple web application and automate its deployment on a simple two server setup. Use the tools suggested in this article, or any other tool for your fancy. For developer environment, you can use *Vagrant*. If you can create a simple IT factory on your laptop or public cloud servers, you are way ahead than others, and definitely interview material.

Conclusion

So this concludes DevOps for Layman. We started with introducing the disputed definitions of DevOps. We then explored why DevOps was and is needed in the first place. We then explored the different tools enabling DevOps. We finally concluded with an analogy of a modern factory, and how DevOps is nothing but a modern IT factory. Hopefully readers got a bird's eye view on what is this DevOps movement all about.

Going forward, DevOps movement is evolving on the fly. With Serverless Computing, AI writing codes, and many other advanced technologies, DevOps is all set for massive transformation for the better (much faster code to production time). With more and more enterprises adopting DevOps, we can safely assume that it is the future of any IT engineer (developers or operators alike). Only time will tell us about the future, but it surely is going to be fast.

Writer's Note: *If this post helped you in any way possible, do take a minute to like, share or comment :). You can also read some of my previous work here.* As always, *Sharing is Caring!*

Report this

Published by

Aayush Shrut
Consultant at Deloitte
Published • 1yr

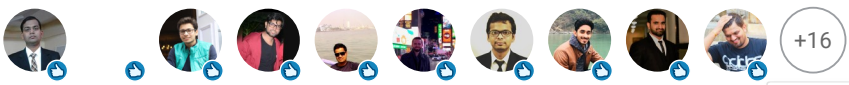
11 articles + Follow

#dev #ops #devops #agile #technology #transformation #interview #highest #paid #job #IT #evolution #configuration #management #CI/CD #continous #integration #deployment #ansible #chef #puppet #git #github #jira #aws #cloud #jenkins #bamboo #travis #nagios #app #dynamics #toolchain #revolution #approach #factory #article #hire #me #please #hashtag #howto #management

Like Comment Share

28 • 3 Comments

Reactions



3 Comments

Messaging

Most Relevant

Add a comment...

Anirudh Singh • 3rd+
at Ernst and Young Consulting
1mo ...

Thanks aayush for this extremely simple and infomrative article.

Bill Michelson • 3rd+
R&D Scientist & Software Architect at Xtreme Solutions
9mo ...

Nice take

1 Like

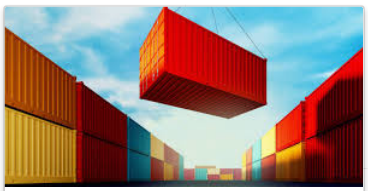
Load more comments

Aayush Shrut
Consultant at Deloitte

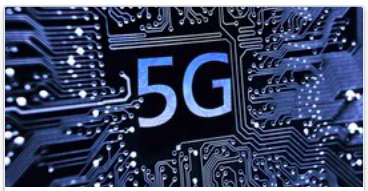
+ Follow

n Aayush Shrut

l Kubernetes for Layman
ut on LinkedIn



Containerization for Layman!
Aayush Shrut on LinkedIn



5G and Network Slicing for Layman
Aayush Shrut on LinkedIn

LTE for Layman (part 3) - The Complete Picture!
Aayush Shrut on LinkedIn

rticles

Messaging