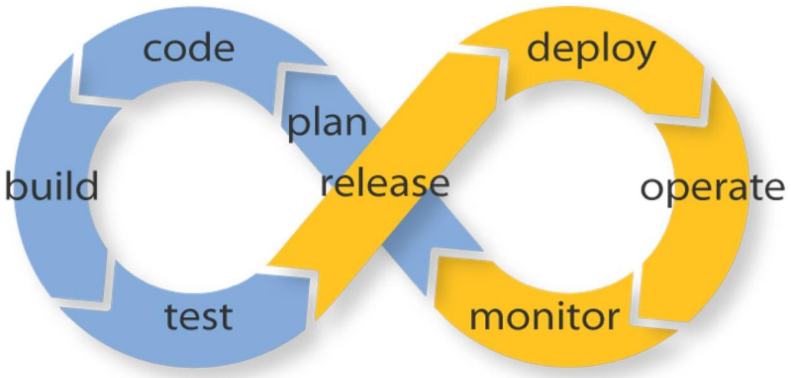
# DevOps Introduction

## What is DevOps ?

You can ask 10 people for a definition of DevOps and likely get 10 different answers.

If you want to find a definition of your own, your research will probably begin by asking Google, "what is DevOps". Naturally, Wikipedia is one of the first result so that is where we will begin. The first sentence on Wikipedia defines DevOps as "a software development method that stresses communication, collaboration and integration between software developers and information technology (IT) professionals." Well, that's a fairly dense definition, yet still pretty vague.

So let's delve into how DevOps works and can better adjoin two traditionally opposing departments.
I think DevOps can be explained simply as operations working together with engineers to get things done faster in an automated and repeatable way. This interesting graphic shows the interaction loop between the two sides.



Endless Possibilities: DevOps can create an infinite loop of release and feedback for all your code and deployment targets.

## From Developer to Operations – One and the Same?

As a developer I have always dabbled lightly in operations. I always wanted to focus on making my code great and let an operations team worry about setting up the production infrastructure. Real applications are much more complex. As I evolved my skillset I started to do more and expand my operations knowledge.

## Infrastructure Automation

You no longer have to build a server from scratch, buy power and connectivity in a data center, and manually plug a machine into the network. After wearing the operations hat for a few years I have learned many operations tasks are mundane, manual, and often have to be done at two in the morning once something has gone wrong. DevOps is predicated on the idea that all elements of technology infrastructure can be controlled through code. With the rise of the cloud it can all be done in real-time via a web service.

## Growing Pains

When you are responsible for large distributed applications the operations complexity grows quickly.

- How do you provision virtual machines?
- How do you configure network devices and servers?
- How do you deploy applications?
- How do you collect and aggregate logs?
- How do you monitor services?

- How do you monitor network performance?
- How do you monitor application performance?
- How do you alert and remediate when there are problems?

## Combining the Power of Developers and Operations

The focus on the developer/operations collaboration enables a new approach to managing the complexity of real world operations. I believe the operations complexity breaks down into a few main categories: infrastructure automation, configuration management, deployment automation, log management, performance management, and monitoring. Below are some tools I have used to help solve these tasks.

## Infrastructure Automation

Infrastructure automation solves the problem of having to be physically present in a data center to provision hardware and make network changes. The benefits of using cloud services is that costs scale linearly with demand and you can provision automatically as needed without having to pay for hardware up front.

- Amazon Web Services
- Windows Azure
- RackSpace Cloud
- HP Cloud
- OpenShift by Red Hat
- Ubuntu Cloud
- Citrix Cloud Platform
- Heroku
- Engine Yard

## Configuration Management

Configuration management solves the problem of having to manually install and configure packages once the hardware is in place. The benefit of using configuration automation solutions is that servers are deployed exactly the same way every time. If you need to make a change across ten thousand servers you only need to make the change in one place.

- Chef
- Puppet
- Ansible
- Salt Stack
- Pallet
- Bcfg2

There are other vendor-specific DevOps tools as well:

- Amazon’s OpsWorks and CloudFormation
- Ubuntu’s JuJu

## Deployment Automation

Deployment automation solves the problem of deploying an application with an automated and repeatable process.

- Jenkins
- Fabric
- Capistrano

## Log Management

Log management solves the problem of aggregating, storing, and analyzing all logs in one place.

- Splunk
- SumoLogic
- Loggly
- LogStash

## Performance Management

Performance management is about ensuring your network and application are performing as expected and providing intelligence when you encounter problems.

- AppDynamics
- Boundary
- Cloudweaver

## Monitoring

Monitoring and alerting are a crucial piece to managing operations and making sure people are notified when infrastructure and related services go down.

- Nagios
- Ganglia
- Sensu
- Icinga
- Zabbix
- PagerDuty

The Times They Are A-Changin

## Out With the Old

In the operations environments I have worked in there were always strict controls on who could access production environments, who could make changes,
when changes could be made, who could physically touch hardware, and who could access what data centers. In these highly regulated and process oriented enterprises the thought of blurring the lines between development and operations seems like a non-starter. There is so much process and tradition standing in the way of using a DevOps approach that it seems nearly impossible. Let's break it down into small pieces and see if could be feasible.
Here are the basic steps to getting a new application built and deployed from scratch (from an operations perspective) in a stodgy financial services environment. If you've never worked in this type of environment

some of the timing of these steps might surprise you. We are going to assume this new application project has already been approved by management and we have the green light to proceed.

1. Place order for development, testing, user acceptance testing, production, and infrastructure. Usually about an 8-week lead time.
2. Development team does works on testing while ops personnel are filling out miles of virtual paperwork to get the infrastructure in place. Much discussion occurs about failover, redundancy, disaster recovery, data center locations, and storage requirements. None of this discussion includes developers, just operations and architects.
3. New application is added to change management database to include new infrastructure components, application components, and dependencies.
4. Operations is hopeful the developers are making good progress in the 8 weeks lead time provided by the operational request process. Servers have landed and are being racked and stacked. Hopefully we correctly estimated the number of users, efficiency of code, and storage requirements that were used to size this hardware. In reality we will have to see what happens during load testing and make adjustments
5. We're closing in on one week until the scheduled go-live date but the application isn't ready for testing yet. It's not the developers fault the functional requirements keep changing but it is going to squeeze the testing and deployment phases.

6. The monitoring team has installed their standard monitoring agents (usually just traditional server monitoring) and marked off the checkbox from the deployment checklist.
7. It's 2 days before go-live and we have an application to test. The load test team has coded some form of synthetic load to be applied to the servers. Functional testing showed that the application worked. Load testing shows slow response times and lot's of errors. Another test is scheduled for tomorrow while the development team works frantically to figure out what went wrong with this test.
8. One day until go-live, load test session 2, still some slow response time and a few errors but nothing that will stop this application from going into production. We call the load test a "success" and give the green light to deploy the application onto the production servers. The app is deployed, functional testing looks good, and we wait until tomorrow for the real test: production users!
9. Go-Live — Users hit the application, the application stalls and/or crashes, the operations team check the infrastructure and get the developers the log files to look at. Management is upset. Everyone is asking if we have any monitoring tools that can show what is happening in the application.
10. Week one is a mess with the application working, crashing, restarting,
working again, and new emergency code releases going into production to fix the problems. Week 2 and each subsequent week will get better until new functionality gets released in the next major change window.

## In With the New

Part of the problem with the scenario above is the development and operations teams are so far removed from each other there is little to no communication during the build and test phases of the development lifecycle. What if we took a small step towards a more collaborative approach as recommended by DevOps? How would this process change? Let's explore (modified process steps are highlighted using bold font):

1. Place order for development, testing, user acceptance testing, production, and infrastructure. Usually about an 8-week lead time.
2. Development and operations personnel fill out virtual paperwork together which creates a much more accurate picture of infrastructure requirements. Discussions about failover, redundancy, disaster recovery, data center locations, and storage requirements progress more quickly with better estimations of sizing and understanding of overall environment.
3. New application is added to change management database to include new infrastructure components, application components, and dependencies.
4. Operations is fully aware of the progress the developers are making. This gives the operations staff an opportunity to discuss monitoring requirements from both a business and IT perspective with the developers. Operations starts designing the monitoring architecture while the servers have arrived and are being racked and stacked. Both the development and operations teams are comfortable with the hardware requirement estimates but understand that they will have to see what happens during load testing and make adjustments. Developers start using the monitoring tools in their dev environment to identify issues before the application ever makes it to test.
5. We're closing in on one week until the scheduled go-live date but the application isn't ready for testing yet. It's not the developers fault that the functional requirements keep changing but it is going to squeeze the testing and deployment phases.
6. The monitoring team has installed their standard monitoring agents as well as the more advanced application performance monitoring (APM) agents across all environments. This provides the foundation for rapid triage during development, load testing, and production.
7. It's 2 days before go-live and we have an application to test. The load test team has coded a robust set of synthetic load based upon application monitoring data gathered during development. This load is applied to the application which reveals some slow response times and some errors. The developers and operations staff use the APM tool together during the load test to immediately identify the problematic code and have a new release available by the end of the original load test. This process is repeated until the slow response times and errors are resolved.
8. One day until go-live, we were able to stress test overnight and everything looks good. We have the green light to deploy the application onto the production servers. The app is deployed, functional testing looks good, business and IT metric dashboard looks good, and we wait until tomorrow for the real test… production users!
9. Go-Live — Users hit the application, the application works well for the most part. The APM tool is showing some slow response time and a couple of errors to the developers and the operations staff. The team agrees to implement a fix after business hours as the business dashboard shows that things are generally going well.

After hours the development and operations team collaborate on the build, test, and deploy of the new code to fix the issues identified that day. Management is happy.

10. Week one is highly successful with issues being rapidly identified and dealt with as they come up. Week 2 and each subsequent week are business as usual and the development team is actively focused on releasing new functionality while operations adapts monitoring and dashboards when needed.

So what scenario sounds better to you? Have you ever been in a situation where increased collaboration caused more problems than it solved? In this example the overall process was kept mostly intact to ensure compliance with regulatory audit procedures. Developers were never granted access to production (regulatory issue for financial services companies) but by being tightly coupled with operations they had access to all of the information they needed to solve the issues.

It seems to me you can make a big impact across the lifecycle of an application by implementing parts of the DevOps philosophy in even a minor way. In this example we didn't even touch the automation aspects of DevOps. That's where all of those fun and useful tools come into play so that is where we will pick up next time.

## Inception and Working With a Product Team

From an operational perspective, my first instinct is to understand the application architecture so I can start thinking about the proper deployment model for the infrastructure components. Here are some of my operational questions and considerations for this stage:

- Are we using a public or private cloud?
- What is the lead time for spinning up each component and ensuring they comply with my companies regulations?
- When do I need to provide a development environment to my dev team or will they handle it themselves?
- Does this application perform functions that other applications or services already handle? Operations should have high-level visibility into the application and service portfolio.

From a development perspective, my first milestone is to make sure the ops team fully understands the application and what it takes to deploy it to a pre-production environment. This is where we the developers sync with the product and ops team and make sure we are aligned.

**Planning for the product team:**

- Is the project scope well defined? Is there a product requirements document?
- Do we have a well defined product backlog?
- Are there mocks of the user experience?

**Planning for the ops team:**

- What tools will we use for deployment and configuration management?
- How will we automate the deployment process and does the ops team understand the manual steps?
- How will we integrate our builds with our continuous integration server?
- How will we automate the provisioning of new environments?
- Capacity Planning – Do we know the expected production load?

There's not a ton of activity at this stage for the operations team. This is really where the DevOps synergy comes into play. DevOps is simply operations working together with engineers to get things done faster in an automated and repeatable way. When it comes to scaling, the more automation in place the easier things will be in the long run.

## Development and Scoping Production

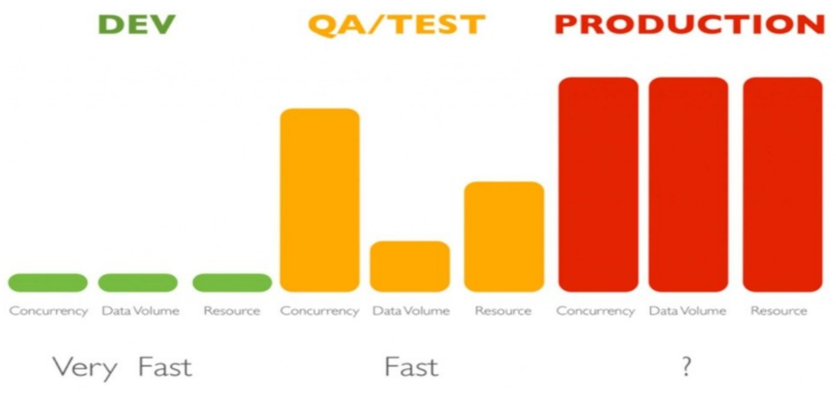
This should start with a conversation between the dev and ops teams to control domain ownership. Depending on your organization and peers strengths this is a good time to decide who will be responsible for automating the provisioning and deployment of the application. The ops questions for deploying complex web applications:

- How do you collect and aggregate logs?
- How do you monitor services?
- How do you monitor network performance?
- How do you monitor application performance?
- How do you alert and remediate when there are problems?

During the development phase the operations focused staff normally make sure the development environment is managed and are actively working to set up the test, QA and Prod environments. This can take a lot of time if automation tools aren't used.

## Testing and Quality Assurance

Once developers have built unit and functional tests we need to ensure the tests are running after every commit and we don't allow regressions in our promoted environments. In theory, developers should do this before they commit any code, but often times problems don't show up until you have production traffic running under production infrastructure. The goal of this step is really to simulate as much as possible everything that can go wrong and find out what happens and how to remediate.



The next step is to do capacity planning and load testing to be confident the application doesn't fall over when it is needed most. There are a variety of tools for load testing:

- Apica Load Test - Cloud-based load testing for web and mobile applications
- Soasta – Build, execute, and analyze performance tests on a single, powerful, intuitive platform.
- Bees with Machine Guns – A utility for arming (creating) many bees (micro EC2 instances) to attack (load test) targets (web applications).
- MultiMechanize – Multi-Mechanize is an open source framework for performance and load testing. It runs concurrent Python scripts to generate load (synthetic transactions) against a remote site or service. Multi-Mechanize is most commonly used for web performance and scalability testing, but can be used to generate workload against any remote API accessible from Python.
- Google PageSpeed Insights - PageSpeed Insights analyzes the content of a web page, then generates suggestions to make that page faster. Reducing page load times can reduce bounce rates and increase conversion rates.

The last step of testing is discovering all of the possible failure scenarios and coming up with a disaster recovery plan. For example, what happens if we lose a database or a data center or have a 100x surge in traffic?

During the test and QA stages operations needs to play a prominent role. This is often overlooked by ops teams but their participation in test and QA can make a meaningful difference in the quality of the release into production. Here's how:

If the application is already in production (and monitored properly), operations has access to production usage and load patterns. These patterns are essential to the QA team for creating a load test that properly exercises the application. I once watched a functional test where 20+ business transactions were tested manually by the application support team. Directly after the functional test I watched the load test that ran the same 2 business transactions over and over again. When I asked the QA team why there were only 2 transactions they said, "because that is what the application team told us to model."

The development and application support teams usually don't have time to sit with the QA team and give them an accurate assessment of what needs to be modeled for load testing. Operations teams should work as the middle man and provide business transaction information from production or from development if this is an application that has never seen production load.

**Here are some of the operational tasks during testing and QA:**

- Ensure monitoring tools are in place.
- Ensure environments are properly configured
- Participate in functional, load, stress, leak, etc… tests and provide analysis and support
- Providing guidance to the QA team

## Production

Production is traditionally the domain of the operations team. For as long as I can remember, the development teams have thrown applications over the production wall for the operations staff to deal with when there are problems. Sure, some problems like hardware issues, network issues, and cooling issues are purely on the shoulders of operations – but what about all of those application specific problems? For example, there are problems where the application is consuming way too many resources, or when the application has connection issues with the database due to a misconfiguration, or when the application just locks up and has to be restarted.

I recall getting paged in the middle of the night for application-related issues and thinking how much better each release would be if the developers had to support their applications once they made it to production. It was really difficult back in those days to say with any certainty that the problem was application related and that a developer needed to be involved. Today's monitoring tools have changed that and allow for problem isolation in just minutes. Since developers in financial services organizations are not allowed access to production servers, it makes having the proper tools all the more important.

**Production DevOps is all about:**

- deploying code in a fast, repeatable, scalable manner
- rapidly identifying performance and stability problems
- alerting the proper team when a problem is detected
- rapidly isolating the root cause of problems

- automatic remediation of known problems and rapid manual remediation of new problems (runbooks and runbook automation)

Your application must always be available and operating correctly during business hours (this may be 24×7 for your specific application).

**Alerting Tools:**

Pager Duty

In case of failures alerting tools are crucial to notify the ops team of serious issues. The operations team will usually have a runbook to turn to when things go wrong. A best practice is to collaborate on incident response plans.

**Maintenance**

Finally we've made it to the last major category of the software development life cycle, maintenance. My mind focuses on the following tasks:

- Capacity planning – Do we have enough resources available to the application? If we use dynamic scaling, this is not an issue but a task to ensure the scaling is working properly.
- Patching – Are we up to date with patches on the infrastructure and application components? This is supposed to help with performance, security, and/or stability but it doesn't always work out that way.
- Support – Are we current with our software support levels?
- New releases – New releases always made me cringe since I assumed the release would have issues the first week. I learned this reaction from some very late nights immediately following those new releases.

As a developer the biggest issues during the maintenance phase is working with the operations team to deploy new versions and make critical bug fixes. The other primary concern is troubleshooting production problems. Even when no new code has been deployed, sometimes failures happen. If you have a great process, application performance monitoring, and a DevOps mentality, resolving the root cause of failures becomes easy.

As you can see, the dev and ops perspectives are pretty different, but that's exactly why those 2 sides of the business need to tear down the walls and work together. DevOps isn't just a set of tools, but a philosophical shift that needs that requires buy-in from all folks involved to truly succeed. It's only through a high-level of collaboration that things will change for the better. AppDynamics can't change the mindset of your organization, but it is a great way to foster collaboration across all of your organizational silos.

**7 Habits of Highly Effective (DevOps) People**

As you may have heard by now, Stephen Covey – the author of "Seven Habits of Highly Effective People" passed away recently. He published this book for the first time in 1989, just as I was entering the workforce and had a strong influence on how I analyzed my own effectiveness. To honor him and his contribution to professional development and the millions of lives that he has touched, I thought I'd tailor his Seven Habits to make them even more relevant and prescriptive to App dynamics users – Application Operations, DevOps & IT Operations professionals.

Managing mission-critical apps isn't easy. There is a lot of pressure to "keep the trains running on time" in an environment where change is a given. Thus, it can be very easy to spend your day on what's "urgent" or "hot" instead of what's most "important". This relates to habit #3 of the Seven Habits and perhaps my favorite. Luckily, we work with some really strong App Operations & DevOps teams who have figured out how to be the most effective and have the most impact at their company.

So, what are their secrets and how can Stephen Covey's habits help Application Operations folks be more effective? What do the most effective Application Operations groups do to differentiate themselves and their company? Let's dig in and look at each habit one at a time.

**1. Be Proactive**

Ok. This concept is a no-brainer for folks in Operations. If you let things happen to-you (ie reactive), you'll spend your whole day responding to angry users and line-of-business folks…trapped in endless war-room conference calls trying to figure out why something broke after-the-fact. At the most basic level, being proactive can only happen if you have enough visibility into how your applications are working before any problem reaches a Severity-1 level. Pilots can't fly a plane without the right set of gauges and instrumentation, so don't put your team in the position of operating a mission-critical app without the right level of visibility. Having visibility puts you in control and allows you to be proactive.

**2. Begin With the End In Mind**

Define what success is and what impact you want to have on your organization. Go beyond uptime and availability measures and think about how your work can contribute to the company's revenue achievement, revenue growth, competitive advantage, customer satisfaction, etc. Thinking in these terms will better align you with your Line-of-Business (LOB) colleagues as well as enhance your job satisfaction. Our users who can say: "my company would have never been able to grow web sales by 35% last year without my contributions to scaling and operating our app" have a lot of job satisfaction.

**3. Put First Things First**

In a nutshell, this habit instructs us to prioritize tasks based on importance, rather than urgency. It is an easy trap to "live in your inbox" or see half the day get wasted by chasing down false alarms. I like Covey's 2×2 matrix and it's a good five minute exercise to put all of our projects/tasks in this matrix to ensure we are spending the bulk of our week on things that are both important and urgent. Certainly, the right Operations tools and automation can help provide focus by prioritizing what work will have the most impact on the success of a mission critical application.

**4. Think Win-Win**

This is also a major tenet of the DevOps movement – If Dev, Ops, and LOB all have divergent goals and objectives…there will always be tension and conflict. As the Operations guy, take the initiative to find out what matters to your Dev and LOB counterparts and how they measure success. Once you've done that, align your goals and objectives with theirs and co-build a plan to get there. What most of the successful DevOps shops are doing is picking shared goals/objectives that matter to Dev/Ops/ LOB and then aligning their efforts to achieve those goals. When everyone is committed to the same goals and a shared plan to get there, a lot of the daily conflict/political infighting goes away.

**5. Think First To Understand, Then To Be Understood**

Application Operations can be hard. For most of our App Ops users, they didn't architect or code the application they are responsible for, but there is a lot they must understand about it to operate it successfully. On top of that, if their shop is agile, their app is likely changing all the time and it can be near impossible to know how it has changed each time. With agile, the days of complete knowledge transfer and gated exit criteria seem to be gone, so you'll need different approaches to understand what you need to know about the App to succeed. The right tools/ automation can help by automatically discovering what has changed and whether or not the new build is performing as well as the last build. Once you have this understanding – you'll be in a better position to communicate to Dev/Test if/how/ where the new build is performing poorly….and be listened to.

**6. Synergize**

This habit can often mean to use teamwork to reach goals unattainable by one person working alone. Often we hear Dev & Ops folks talk about the "blame game" that goes on inside their company and the acronym MTTI – "Mean-time-to-Innocence"…where each silo (network, database, application, server, storage) are all trying to prove that their area isn't to blame for the latest problem. When the culture that exists between Dev and Ops is one that leads to finger-pointing rather than collaboration, then there is an opportunity for improvement. The DevOps movement has some good materials on how to enact a cultural change to enhance collaboration and teamwork.

**7. Sharpen Your Saw**

Continually look for ways to improve your knowledge and the way you do Operations. What we've seen a lot in the last 2 years is Infrastructure Ops folks re-defining themselves as AppOps or DevOps people. And they aren't just changing their title. What this means is that they've changed the way they work, added to their skills, gained experience with new tools/automation (AppDynamics, Puppet, Chef, etc..). What I often see from folks that have embraced these new skills is a new level of confidence and job satisfaction. I hear a new pride when they say they are "Ops people who can code" or "Ops people who can troubleshoot App issues better than their peers". If you are interested in taking this same journey, study the presentations on slideshare by Netflix and others on how they run Operations and what this has meant to their job definitions.

In the end, the habits and behaviors of highly effective Application Operation require a range of skills that go beyond learning a new way to use automation to improve availability and performance – these skills also include effective relationship management and goal alignment across teams, prioritization/time management, and continuous improvement. The good news is that most of the App Ops folks we work with seem naturally inclined to continually improve.

## A Short History of DevOps

**2008**

Software developer Patrick Debois - developer, network specialist, system administrator, tester and project manager.Debois helps plant the seeds of the DevOps movement at the Agile conference in Toronto, resolve the conflict between the software developers and the operations teams when it comes to getting great work done quickly.

**2009**

At the O'Reilly Velocity Conference, two Flickr employees—John Allspaw, senior vice president of technical operations, and Paul Hammond, director of engineering—deliver a seminal talk known as "10+ Deploys per Day: Dev and Ops Cooperation at Flickr."
Debois launches the first Devopsdays event, in Ghent, Belgium. Early supporters include John Willis, an enterprise system management expert, and Kris Buytaert, a Linux and open source consultant.

**2010**

The first US Devopsdays is organized, with the help of Willis.The events soon become a regular global series of community-organized conferences and a major force driving the DevOps community forward.

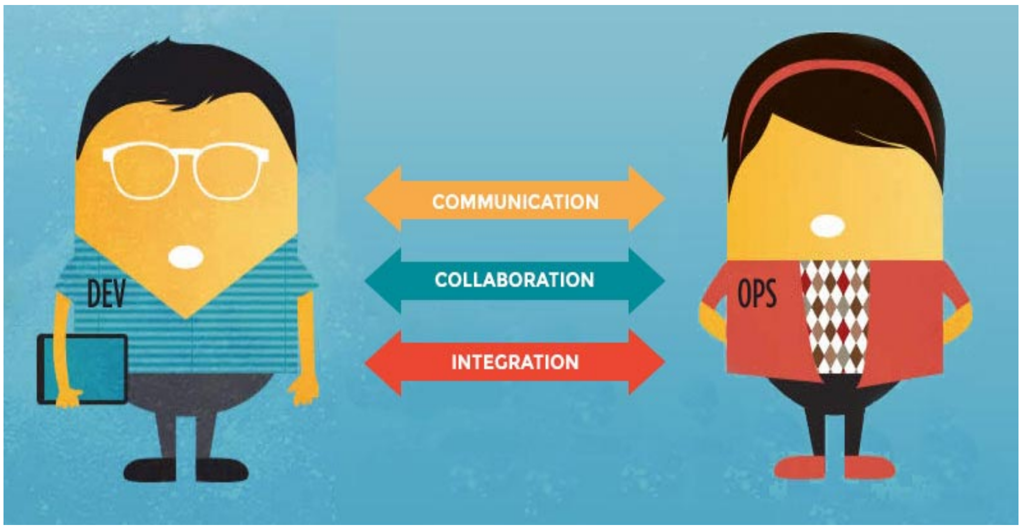
**2011**

The DevOps community starts to build open source tools like Vagrant (for creating and configuring virtual development environments) that work with existing configuration management tools like Puppet and Chef.

### A Short DevOps Definition

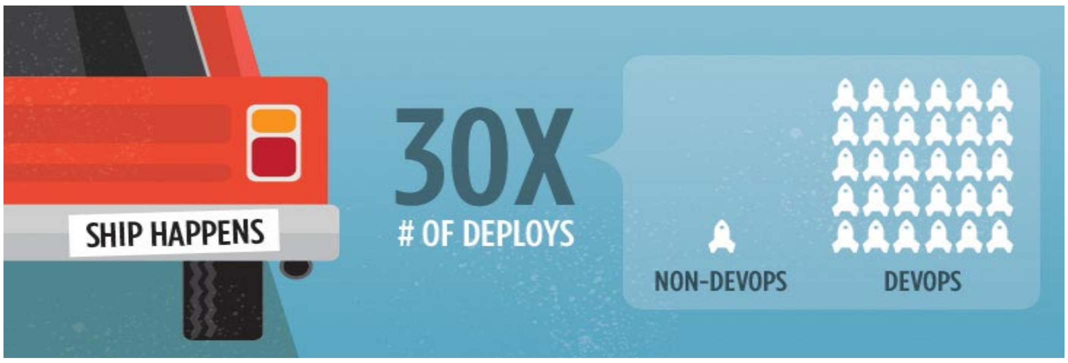
DevOps is the philosophy of unifying Development and Operations at the culture, practice, and tool levels, to achieve accelerated and more frequent deployment of changes to Production.

Culture=behaviour, teamwork, responsibility/accountability, trust...
Practice=policy, roles, processes/procedures, metrics/reporting...
Tools=shared skills, tool making for each other, common technology platforms...



## Modern applications IN the cloud and OUT.

- DevOps found initial traction within many large public cloud service providers.  infrastructure is now part of the code.
- Classic big WebOps shops like Google, Amazon, Twitter and Etsy are known to do deployments multiple times a day.
- DevOps helps ensure frequent deploys with a low failure rate.
- Companies of all sizes are beginning to implement DevOps practices.
- DevOps adoption increased from 66 percent in 2015 to 74 percent in 2016.
-

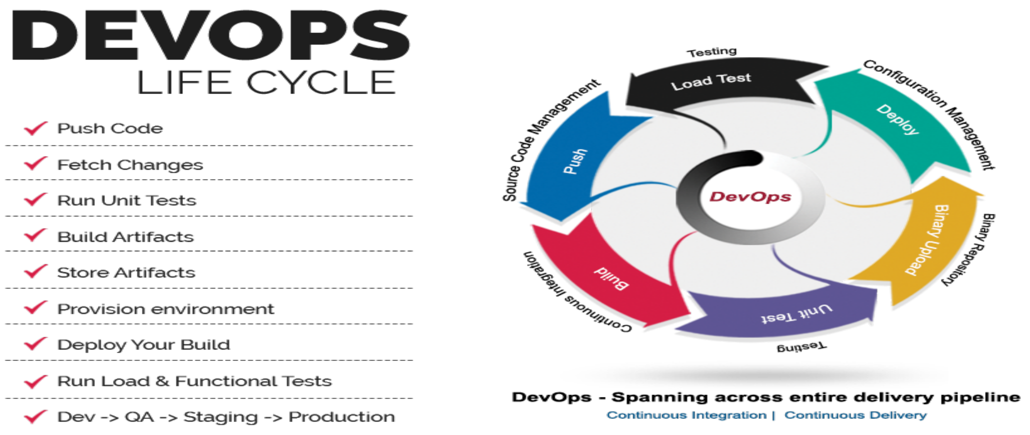


## 5 Things DevOps is NOT

1. DevOps is not simply combining Development & Operations teams
2. DevOps is not a separate team
3. DevOps is not a tool
4. DevOps is not a one-size-fits-all strategy
5. DevOps is not just automation

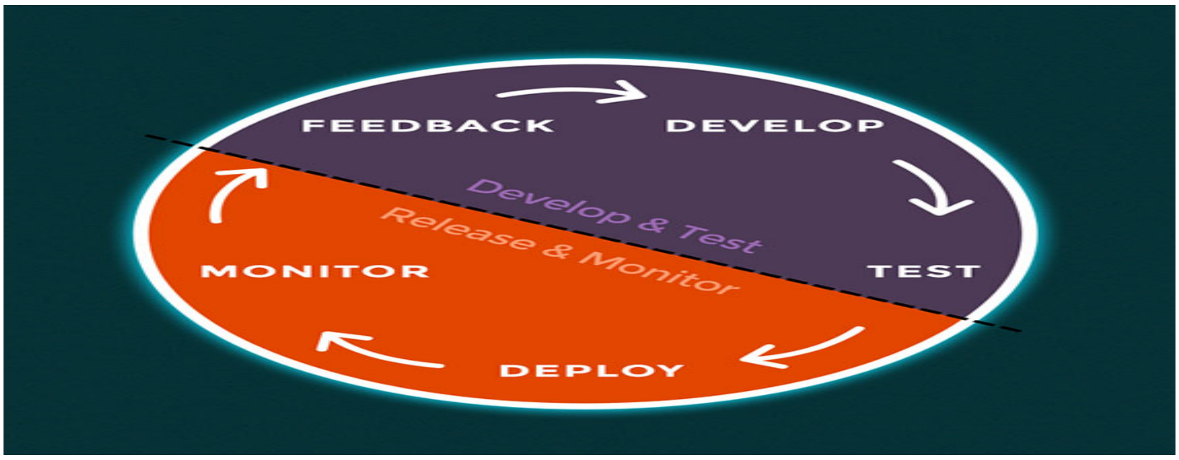
## DevOps and Software Development Life Cycle

- The DevOps Lifecycle Looks Like This:
- Check in code
- Pull code changes for build
- Run tests (continuous integration server to generate builds & arrange releases): Test individual models, run integration tests, and run user acceptance tests.
- Store artifacts and build repository (repository for storing artifacts, results & releases)
- Deploy and release (release automation product to deploy apps)
- Configure environment
- Update databases
- Update apps
- Push to users – who receive tested app updates frequently and without interruption
- Application & Network Performance Monitoring (preventive safeguard)
- Rinse and repeat



### The DevOps Lifecycle = A Rapid Release Cycle with a Strong Feedback Loop

- Utilizing a DevOps lifecycle, products can be continuously deployed in a feedback loop through:
- Infrastructure Automation
- Configuration Management
- Deployment Automation
- Infrastructure Monitoring
- Log Management
- Application & Performance Management



### Infrastructure As A Code

- With the arrival of tools like  Puppet, and Chef, the concept of Infrastructure as Code was born
- Infrastructure as code, or programmable infrastructure, means writing code (which can be done using a high level language or any descriptive language) to manage configurations and automate provisioning of infrastructure in addition to deployments.

### DevOps on the Cloud

- What is the relationship between Cloud Computing and DevOps? Is DevOps really just “IT for the Cloud”? Can you only do DevOps in the cloud? Can you only do cloud using DevOps? The answer to all three questions is “no”. Cloud and DevOps are independent but mutually reinforcing strategies for delivering business value through IT

## Prerequisites for DevOps

- There’s no formal career track for becoming a DevOps engineer. They are either developers who get interested in deployment and network operations, or sysadmins who have a passion for scripting and coding, and move into the development side where they can improve the planning of test and deployment.
- Either way, these are people who have pushed beyond their defined areas of competence and who have a more holistic view of their technical environments.”

## Tools in Dev Ops

- Ansible
- Scripting
- Jenkins
- Chef
- Git
- Nexus
- Docker
- Nagios
- Vagrant….etc

## Continuous Integration

- Continuous integration (CI) is a software engineering practice in which isolated changes are immediately tested and reported on when they are added to a larger code base. The goal of CI is to provide rapid feedback so that if a defect is introduced into the code base, it can be identified and corrected as soon as possible.
- Time frames are crucial. Integration should be divided into three steps:
- commit new functionality and build new application
- run unit tests
- run Integration/System tests

## Continuous Release and Deployment

- The relevant terms here are “Continuous Integration” and “Continuous Deployment”, often used together and abbreviated as CI/CD . Originally Continuous Integration means that you run your “integration tests” at every code change while Continuous Delivery means that you automatically deploy every change that passes your tests.
- **The Software Development Pipeline**
- From a high level, a CI/CD pipeline usually consists of the following discrete steps:
- **Commit**. When a developer finishes a change to an application, he or she commits it to a central source code repository.
- **Build**. The change is checked out from the repository and the software is built so that it can be run by a computer. This steps depends a lot on what language is used and for interpreted languages this step can even be absent.
- **Automated tests**. This is where the meat of the CI/CD pipeline is. The change is tested from multiple angles to ensure it works and that it doesn’t break anything else.
- **Deploy**. The built version is deployed to production.