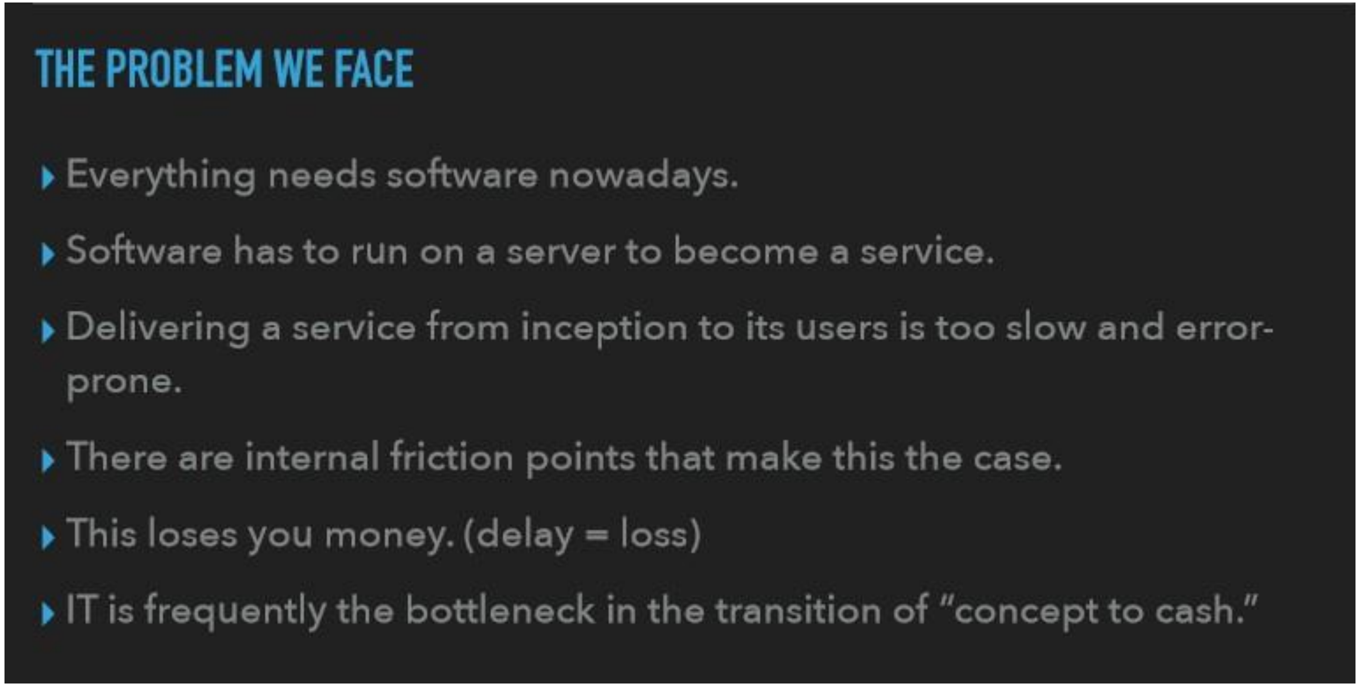
DevOps Fundamental

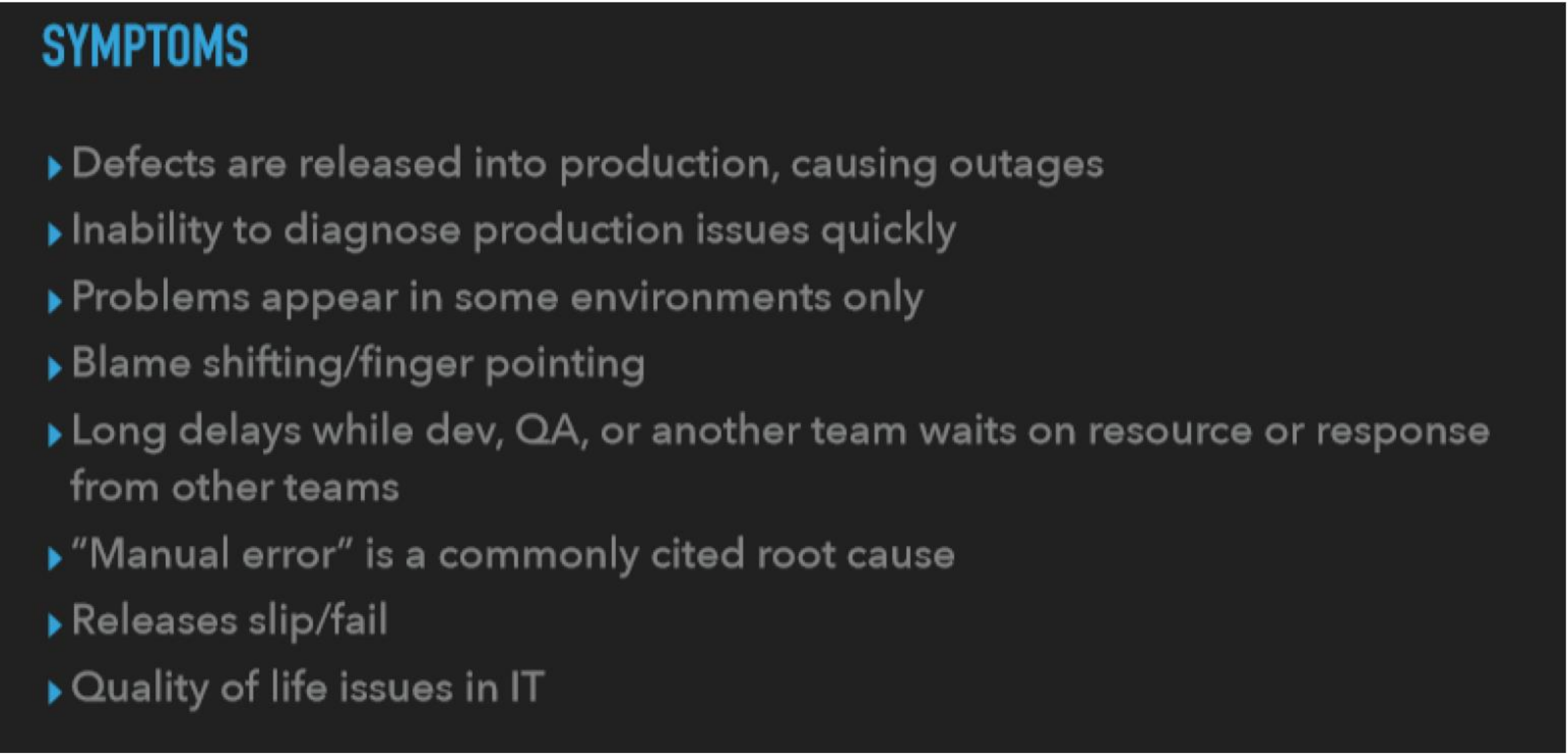
AKM Rafiqul Alam
Software Engineer
KAZ Software

- What is DevOps?
- What problems does DevOps address?
- How does DevOps relate to Agile?
- What can DevOps do for you



THE PROBLEM WE FACE

- Everything needs software nowadays.
- Software has to run on a server to become a service.
- Delivering a service from inception to its users is too slow and error-prone.
- There are internal friction points that make this the case.
- This loses you money. (delay = loss)
- IT is frequently the bottleneck in the transition of "concept to cash."

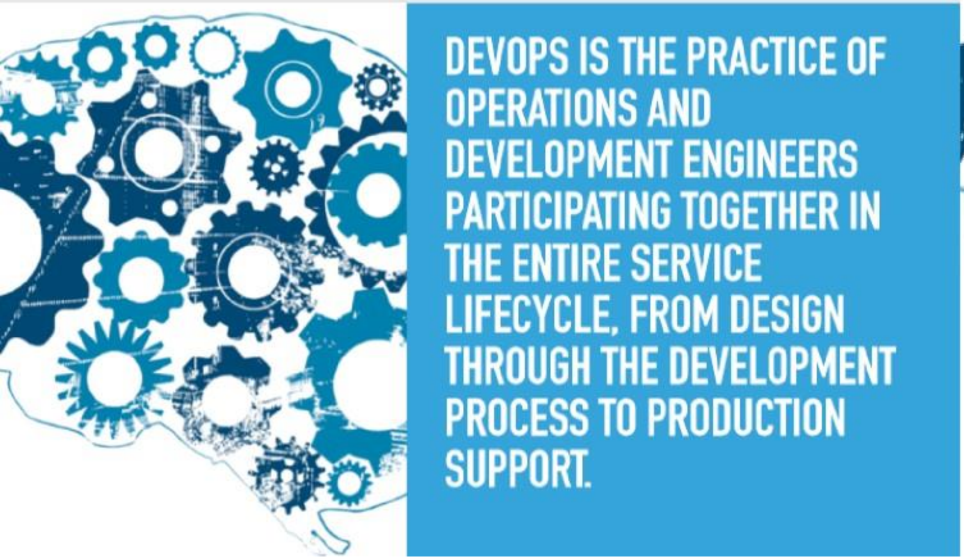


SYMPTOMS

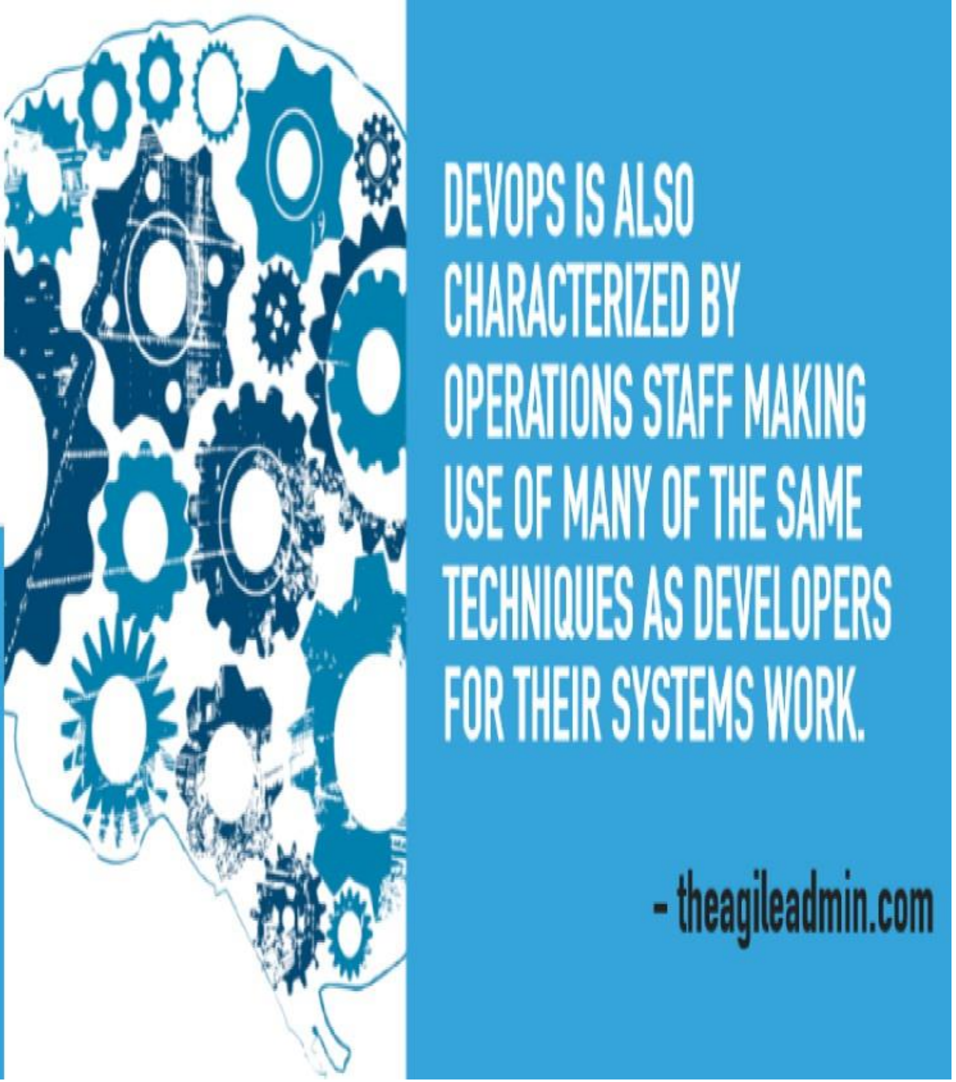
- Defects are released into production, causing outages
- Inability to diagnose production issues quickly
- Problems appear in some environments only
- Blame shifting/finger pointing
- Long delays while dev, QA, or another team waits on resource or response from other teams
- "Manual error" is a commonly cited root cause
- Releases slip/fail
- Quality of life issues in IT

WORKED FINE IN DEV....
...OPS PROBLEM NOW

What is DevOps?



DEVOPS IS THE PRACTICE OF OPERATIONS AND DEVELOPMENT ENGINEERS PARTICIPATING TOGETHER IN THE ENTIRE SERVICE LIFECYCLE, FROM DESIGN THROUGH THE DEVELOPMENT PROCESS TO PRODUCTION SUPPORT.



DEVOPS IS ALSO CHARACTERIZED BY OPERATIONS STAFF MAKING USE OF MANY OF THE SAME TECHNIQUES AS DEVELOPERS FOR THEIR SYSTEMS WORK.

- theagileadmin.com

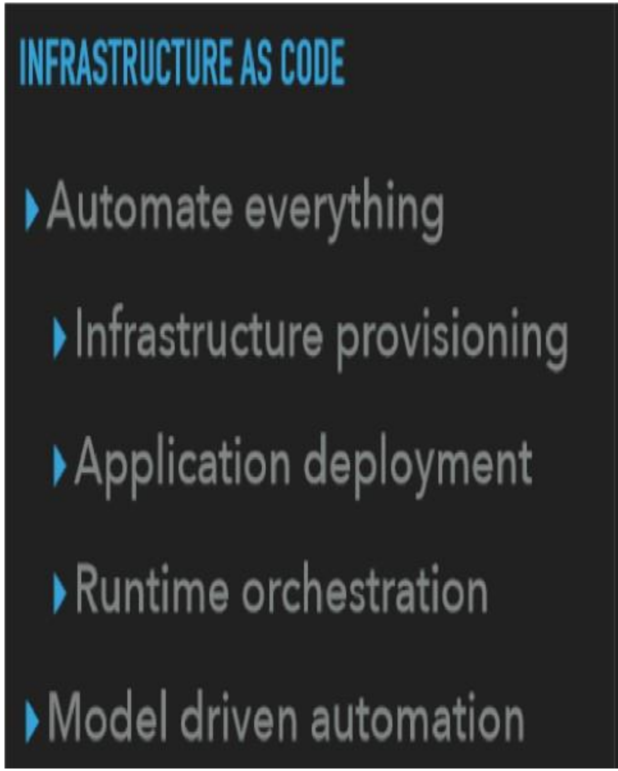
DevOps

- Philosophy more than set of tools or procedure
- Engineers are exposed to full life cycle of product
- Requires lot of discipline but gives enormous amount of control over what is being built
- If engineers understand problem — they will solve it

THE THREE PILLARS OF DEVOPS

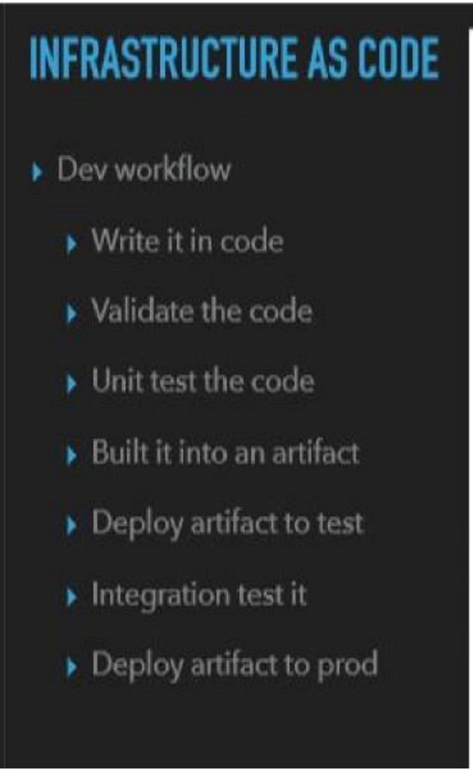
- Infrastructure Automation
- Continuous Delivery
- Reliability Engineering

INFRASTRUCTURE AUTOMATION



INFRASTRUCTURE AS CODE

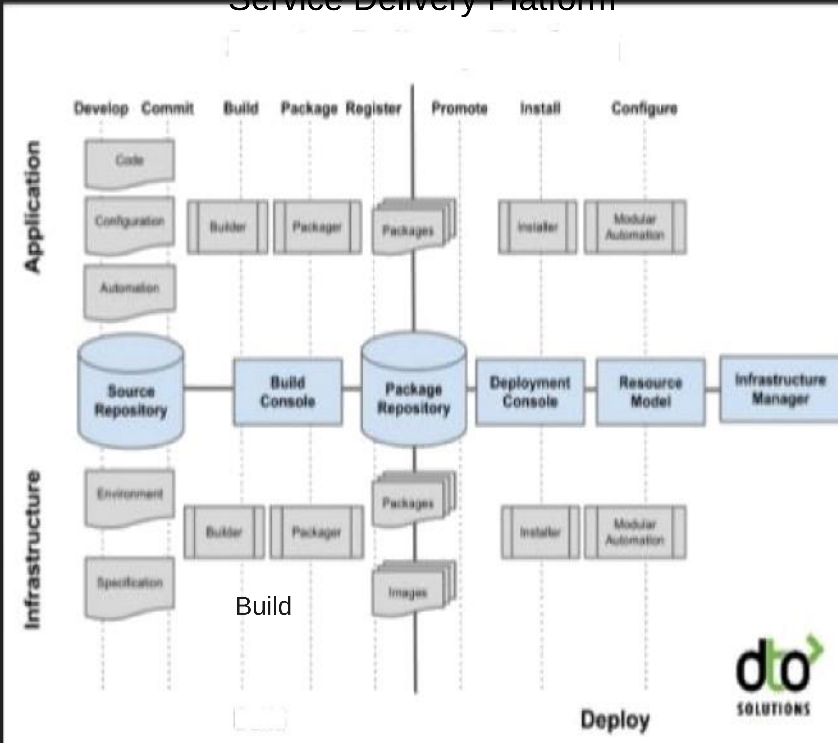
- Automate everything
  - Infrastructure provisioning
  - Application deployment
  - Runtime orchestration
- Model driven automation



INFRASTRUCTURE AS CODE

- Dev workflow
  - Write it in code
  - Validate the code
  - Unit test the code
  - Built it into an artifact
  - Deploy artifact to test
  - Integration test it
  - Deploy artifact to prod

Service Delivery Platform



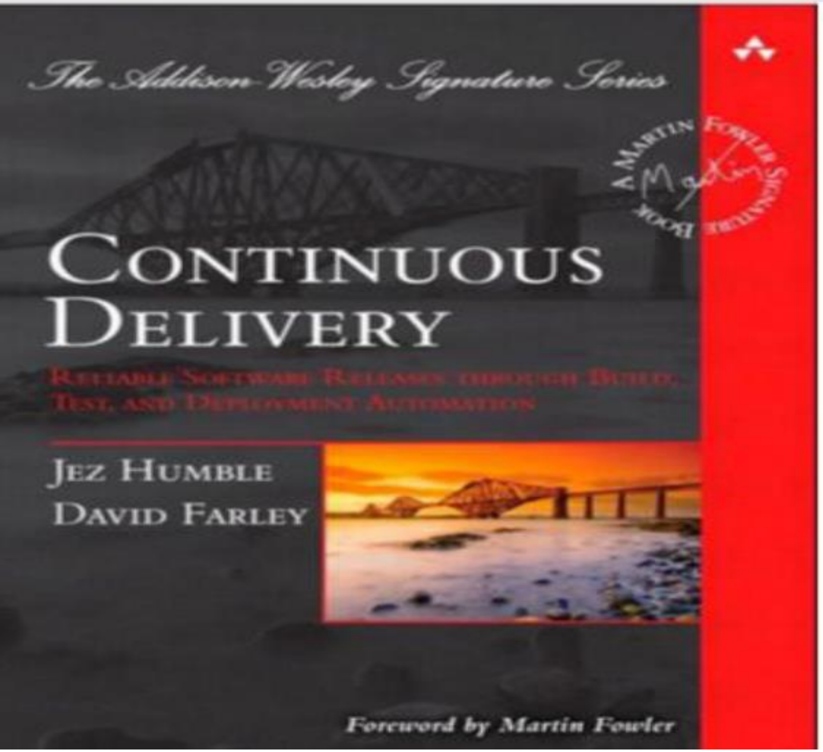
INFRASTRUCTURE AUTOMATION TOOLING

- Infrastructure Models - AWS Cloudformation, Terraform, Azure ARM Templates, Ubuntu Juju
- Hardware Provisioning - Packer, Foreman, MaaS, Cobbler, Crowbar, Digital Rebar
- Configuration Management - Puppet, Chef, Ansible, Salt, CFEngine
- Integration Testing - rspec, serverspec
- Orchestration - Rundeck, Ansible, Kubernetes (for docker)

SPECIAL TOPICS

- Immutable deployment with docker
- Serverless with AWS Lambda/Azure Functions/Google Cloud Functions
- Single model for infra and apps is best (Juju, Kubernetes)

CONTINUOUS DELIVERY


The Addison-Wesley Signature Series
A Martin Fowler Signature Book
Continuous Delivery
Reliable Software Releases through Build, Test, and Deployment Automation
Jez Humble
David Farley
Foreword by Martin Fowler

<- All You Need To Know

- Integration (CI): Build and test
- Deployment (CD): Deploy and integration test
- Delivery: All the way to production, baby

WORKING SOFTWARE, ALL THE TIME

- Builds should pass the coffee test (< 5 minutes)
- Commit really small bits
- Don’t leave the build broken
- Use a trunk/master based development flow (branch less than a day - use feature flags to branch by abstraction)
- Don't allow flaky tests, fix them!
- The build should return a status, a log, and an artifact.

HOW IT WORKS

THE CD PIPELINE

- Only build artifacts once
- Artifacts should be immutable
- Deployment should go to a copy of production before going into production
- Stop deploys if it a previous step fails
- Deployments should be idempotent

TEST FOR SUCCESS

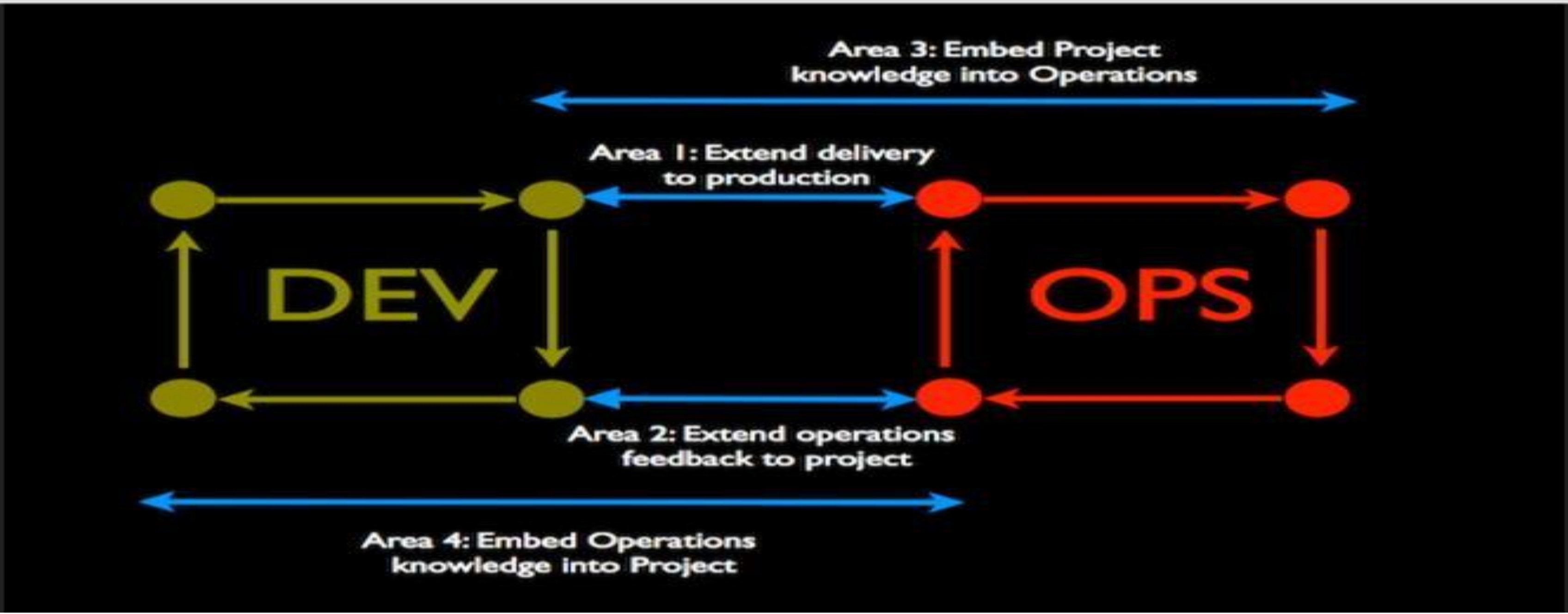
- Automated testing is CI table stakes
- Unit tests
- Integration tests
- Crossbrowser, performance, security, etc.
- Test driven development, ideally (TDD/BDD/ATDD)
- Do it
- Stop being a crybaby
- Really, do it

SHINY TRINKETS

TOOLING

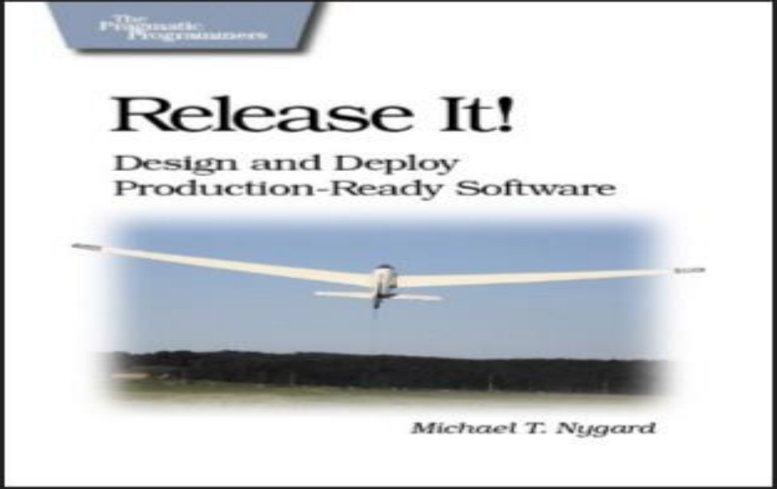
- Version Control (git, others, doesn't matter as long as you use it)
- CI System (jenkins, bamboo, circleCI, travisCI)
- Build (whatever your platform needs - make/rake, maven, gulp, packer)
- Test (*unit, robot/protractor, cucumber)
- Artifact Repository (artifactory, nexus, dockerhub, S3)
- Deployment (rundeck, ansible, your CM system)

RELIABILITY ENGINEERING


Area 3: Embed Project knowledge into Operations
Area 1: Extend delivery to production
DEV
OPS
Area 2: Extend operations feedback to project
Area 4: Embed Operations knowledge into Project

DESIGN FOR OPERATION

- Design patterns exist for creating resilient systems.
- If your app sucks, there's only so much an ops team can do about it once it's deployed.
- Devs need to take responsibility for their app through deployment.


The Pragmatic Programmers
Release It!
Design and Deploy Production-Ready Software
Michael T. Nygard

ADDING OPS INTO DEV

- Enhance Service Design With Operational Knowledge
  - Reliability
  - Performance
  - Security
  - Test These
  - Design Patterns (Release It!, 12-factor apps)
- Foster a Culture of Responsibility
  - Whether your code passes test, gets deployed, and stays up for users is your responsibility – not someone else's
- Make Development Better With Ops
  - Productionlike environments
  - Power tooling - vagrant, etc.

How does DevOps relate to Agile?

What is DevOps?
DevOps is a software development method which focuses on communication, integration, and collaboration among IT professionals to enables rapid deployment of products.

DevOps is a culture that promotes collaboration between Development and Operations Team. This allows deploying code to production faster and in an automated way. It helps to increases an organization's speed to deliver application and services. It can be defined as an alignment of development and IT operation.

What is Agile?
Agile Methodology involves continuous iteration of development and testing in the SDLC process. This software development method emphasizes on iterative, incremental, and evolutionary development.

Agile development process breaks the product into smaller pieces and integrates them for final testing. It can be implemented in many ways, including scrum, Kanban, scrum, XP, etc.

DevOps and agile aren't same thing but they are complementary to each other. Agile development is development that adheres to the principles stated in The Agile Manifesto. In brief, agile is the word for an environment in which the priorities, according to the authors of the manifesto, are

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation and
- Responding to change over following a plan.

The point of DevOps is to deliver technology to business units in a timely fashion and ensure the technology runs without interruption or disruption. To achieve its goal. DevOps mostly focus on-

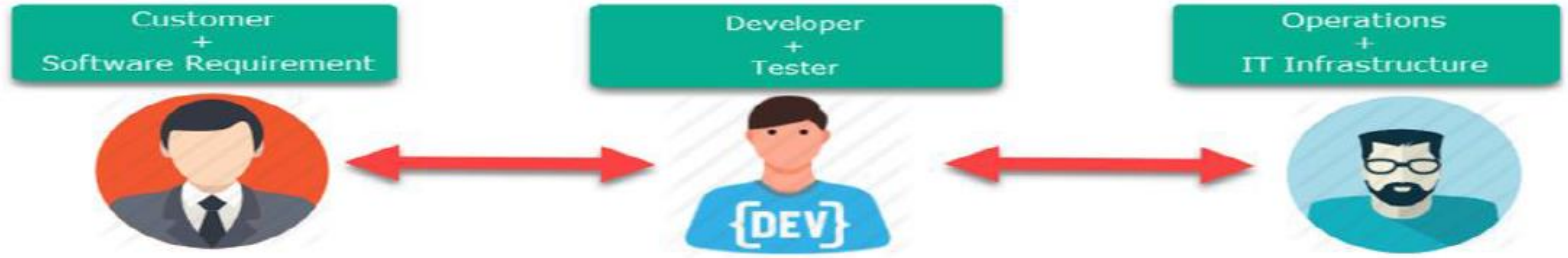
- deep communication between software development and IT operational groups
- automated deployment processes

Below are list of relation/differences with agile and DevOps-

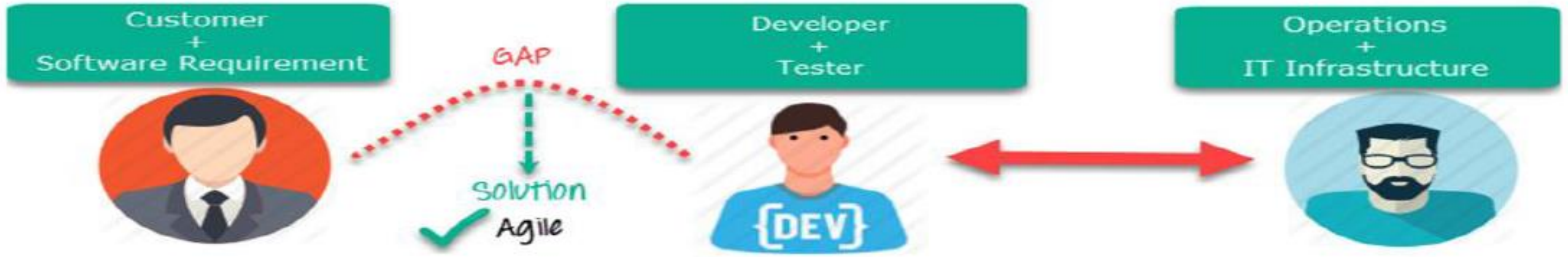
- Agile software development is a methodology for developing software, but DevOps, on the other hand, is all about taking software which is ready for release and deploying it in the safest, most reliable manner possible.
- Agile is all about development. Sometimes, it takes over the entire company. Even when it does, agile discipline doesn't inevitably lead to DevOps. The practice of DevOps involves a separate discipline and methodology from those of agile.
- Agile practices, like Continuous Delivery practices, can be part of a DevOps activities.

Agile Vs. DevOps

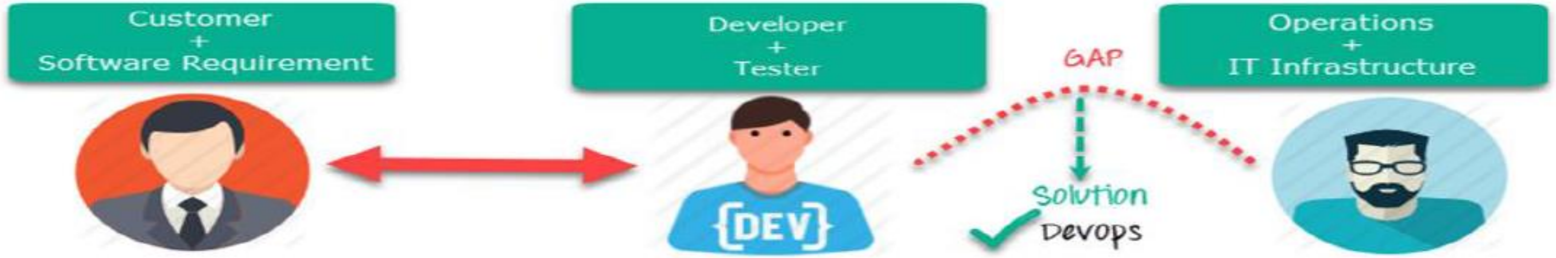
Stakeholders and communication chain in a typical IT process.


Customer + Software Requirement
Developer + Tester
{DEV}
Operations + IT Infrastructure

Agile addresses gaps in Customer and Developer communications


Customer + Software Requirement
GAP
Solution
Agile
Developer + Tester
{DEV}
Operations + IT Infrastructure

DevOps addresses gaps in Developer and IT Operations communications


Customer + Software Requirement
Developer + Tester
{DEV}
GAP
Solution
Devops
Operations + IT Infrastructure

DevOps Engineer Need the Following Skills

1. Flexibility

Coding is an on-going process, ever changing and always needing updating. To be a successful and effective DevOps engineer the ideal candidate must have the ability to continuously develop
and integrate new systems and operations into the code. A DevOps engineer must have flexible working skills and adapt to the changing code.

Engineers must be comfortable moving from one area of software construction to another, be it integration, testing, releasing or deployment.

Continuous Integration, for example, requires the technical skills to manage change quickly and efficiently, as well as being
able to work collaboratively in a team to guarantee everyone is working towards the same goal.

2. Security Skills

As with many other skilled areas security is always of the utmost importance, especially in coding. An easy way for hackers to get into systems is through vulnerabilities, undermining the
system that is in place to get to the data.

DevOps brings a faster cycle of development and deployment of code, which means that vulnerabilities are at higher risk of being introduced to the code much quicker than they have ever been able to before. Therefore, engineers must have the skills to write secure code to protect applications from unwanted attacks, in addition to ensuring systems have defense mechanisms in place against common cybersecurity vulnerabilities.

A DevOps engineer must have security skills when being employed because it is paramount to build-in secure software from the start of deployment, as opposed to adding it in later. If security is not in place from the outset, then here is more chance that hackers could write in damaging code to the network. Therefore, when recruiting for the next DevOps engineer ensure security is on the top of the list of skills.

3. Collaboration

When it comes to a successful DevOps engineer, the ability to perform as a one-man band will not pass muster – collaboration is in fact central to the DevOps concept, bringing together software development and software operation. A DevOps engineer must have the ability to work in a team, with collaboration providing more cross-functionality within the DevOps process.

4. Scripting Skills

Though it may sound obvious, any developer must have high-quality skills in scripting code. Whether its JavaScript, Python, Perl, or Ruby, a successful DevOps engineer must be able to write code. From writing manual
code to replacing manual processes such as assigning IP addresses or DNS codes, there must be someone with the ability to write them and that’s what the perfect candidate should be able to do.

5. Decision-making

An indecisive candidate is not one that you want for your business DevOps engineer. The successful candidate will have the ability to confidently and quickly make a decision is the hustle-bustle environment a DevOps engineer works in.

The ever-changing nature of code brings the necessity to quickly make the decision on how to fix any incoherent elements of the code. Decisiveness must be an element to consider when employing a DevOps engineer, because making quick decisions allows engineers to maintain the ability of rapid development and deployment of new coding changes.

6. Infrastructure Knowledge

Scripting is just one of the key skills a developer should have, just ahead of cloud and infrastructure experience. Engineers should have a working understanding of data centre-based and cloud infrastructure components. This includes elements such as how software is networked to running virtual networks.

Without the ability to understand infrastructure it could prove somewhat difficult to be the full package DevOps engineer. Incorporating infrastructure skills will enable an effective DevOps engineer to design and deploy applications effectively using the best of the best platforms.

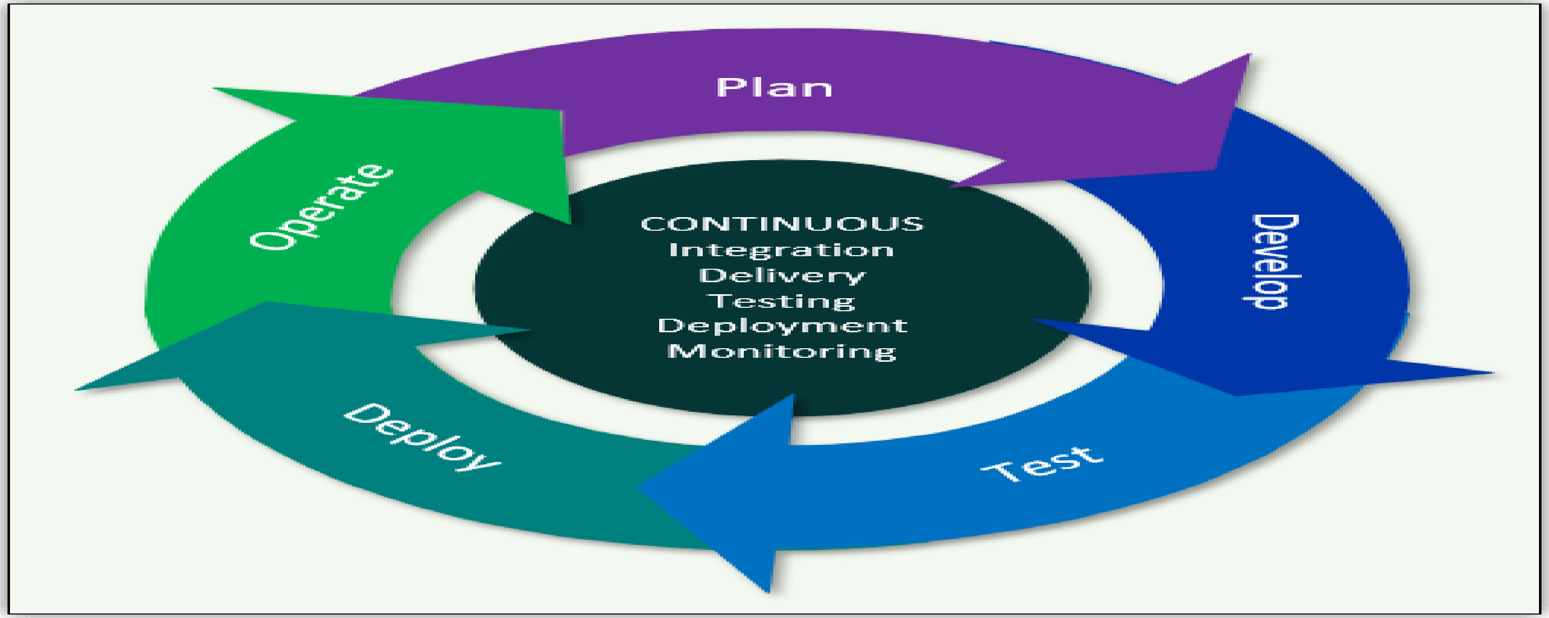
7. Soft Skills

As mentioned above being a DevOps engineer is no one-man job, so in that case, any future employee must have soft skills as well as technical. Bound on trust, DevOps culture enables all workers to be communicative and understanding to the process and if changes need to be made.

When developers communicate with each other effectively, applications can be delivered in a much shorter period of time than if some workers were absent to information. As well as quicker market deployment, having
good communication will lead to fewer errors and therefore lower costs and improve the quality of code.

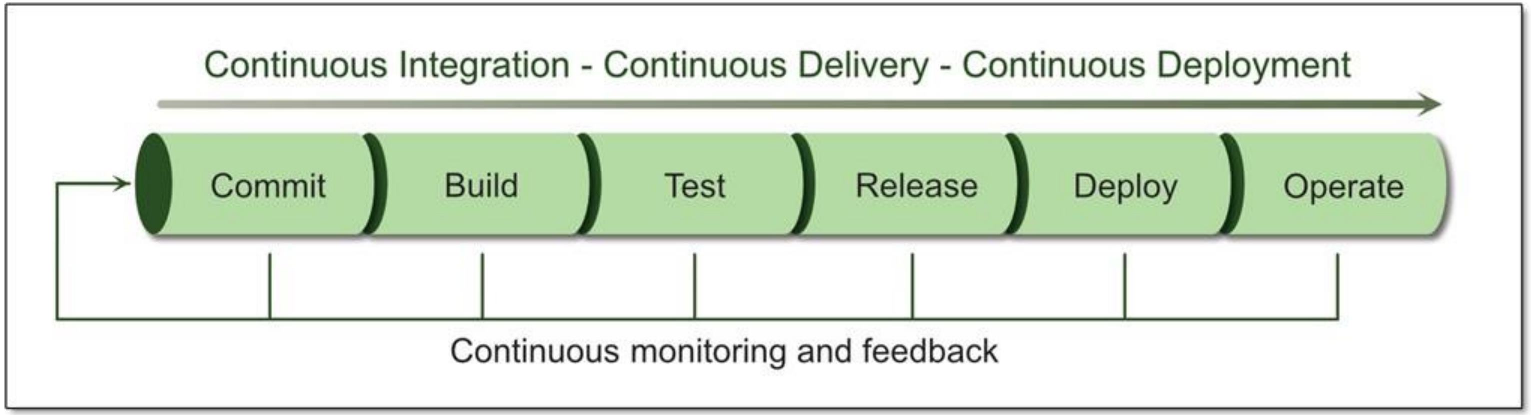
DevOps Delivery Pipeline

DevOps unifies the application delivery process into a continuous flow that incorporates planning, development, testing, deployment and operations, with the goal of delivering applications more quickly and easily. Figure 1 provides an overview of this delivery flow and its emphasis on continuous, integrated services.



The continuous services—integration, delivery, testing, deployment, and monitoring—are essential to the DevOps process flow. Before I go further with this discussion, however, I should point out that documentation about DevOps practices and technologies often vary from one source to the next, leading to confusion about specific terms and variations in the process flow. Despite these differences, the underlying principles remain the same—development and operations coming together to optimize, speed up and in other ways, improve application delivery.

With this in mind, consider the application delivery pipeline that drives the DevOps process flow. One way to represent the pipeline is to break it down into six distinct stages—Commit, Build, Test, Release, Deploy, and Operate. The pipeline begins with code being committed to a source control repository and ends with the application being maintained in production, as shown in Figure 2. Notice that beneath the pipeline is a monitoring and feedback loop that ties the six stages together and keeps application delivery moving forward while supporting an ongoing, iterative process for regularly improving the application.


Continuous Integration - Continuous Delivery - Continuous Deployment
Commit
Build
Test
Release
Deploy
Operate
Continuous monitoring and feedback

You'll find plenty of variations of how the pipeline is represented, but the idea is the same: to show a continuous process that incorporates the full application lifecycle—from developing the applications to operating them in production. What the figure does not reflect is the underlying automation that makes the process flow possible, although it's an integral part of the entire operation.

Also integral to the pipeline are the concepts of continuous integration (CI), continuous delivery (CD), and continuous deployment (CD). My use of the same acronym for both continuous delivery and continuous deployment is no accident. In fact, it points to one of the most common sources of confusion when it comes to the DevOps delivery pipeline.

You'll often see the delivery process referred to as the CI/CD pipeline, with no clear indication whether CD stands for continuous delivery or continuous deployment. Although you can rely on CI referring to continuous integration in this context, it's not always clear how CI differs from the two CDs, how the two CDs differ from each other, or how the three might be related. To complicate matters, the terms *continuous delivery* and *continuous deployment* are often used interchangeably, or one is used to represent the other.

To help bring clarity to this matter, I'll start by dropping the use of acronyms and then give you my take on these terms, knowing full well you're likely to stumble upon other interpretations.

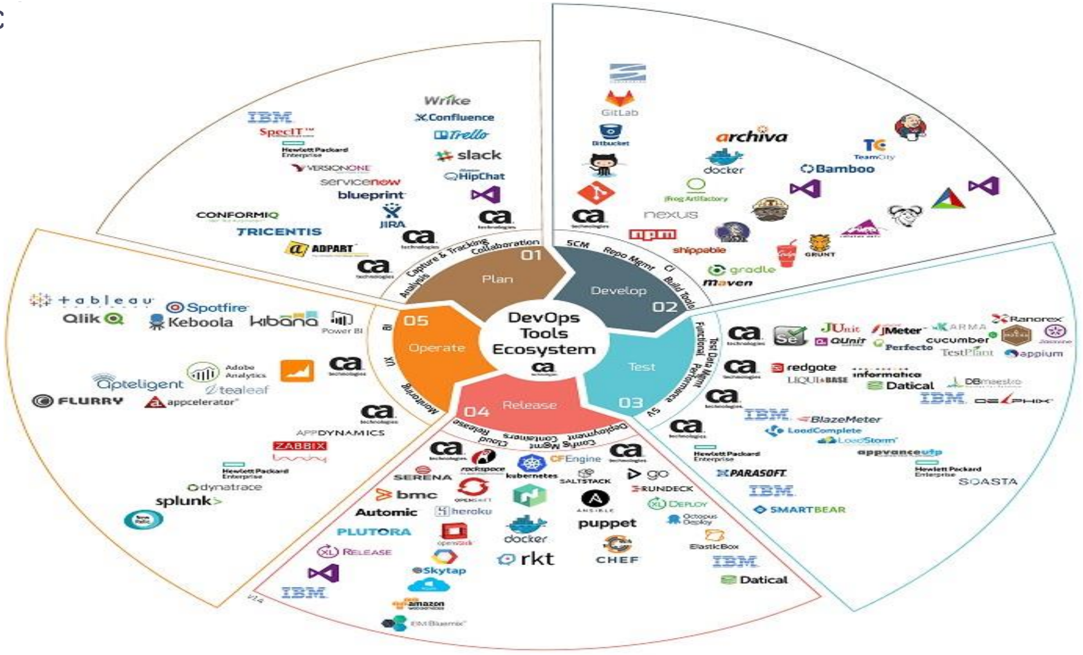
I'll start with continuous integration, which refers to the practice of automating application build and testing processes. After developers create or update code, they commit their changes to a source control repository. This launches an automated build and testing operation that validates the code whenever changes are checked into the repository. (The automated testing includes such tests as unit and integration tests.)

Developers can see the results of the build/testing process as soon as it has completed. In this way, individual developers can check in frequent and isolated changes and have those changes verified shortly after they commit the code. If any defects were introduced into the code, the developer knows immediately. At the same time, the source control system might roll back the changes committed to the central repository, depending on how continuous integration has been implemented.

With continuous integration, development teams can discover problems early in the development cycle, and the problems they do discover tend to be less complex and easier to resolve.

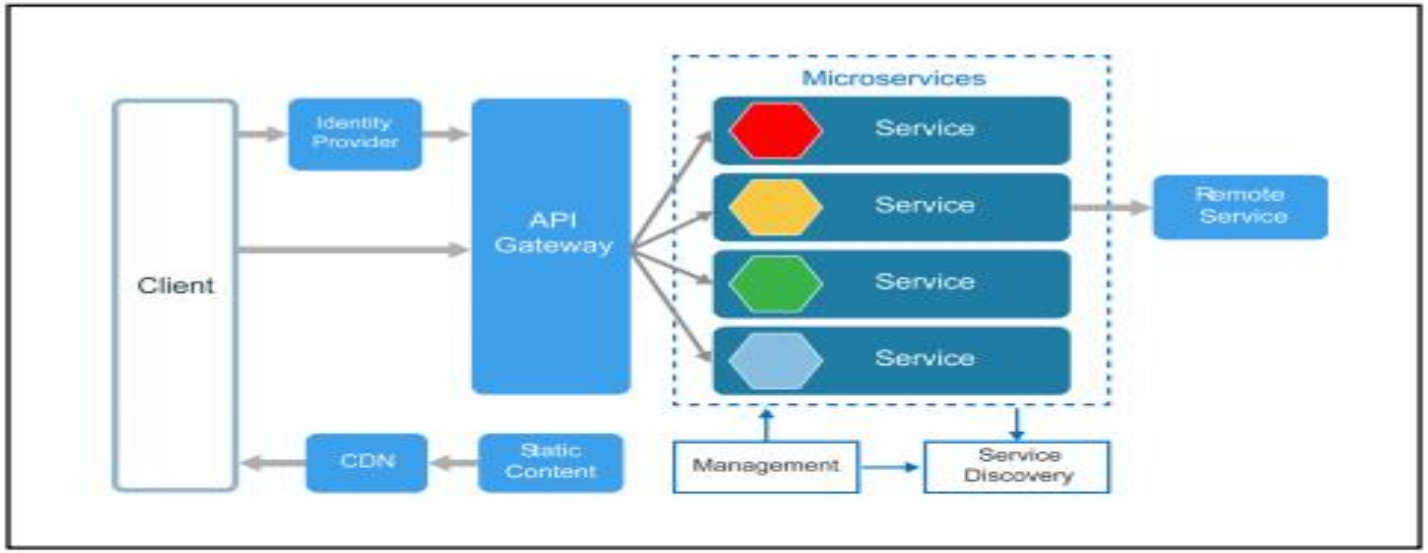
DevOps Eco System

But the existence of so many DevOps tools can be confusing. The tools will be divided into the five stages of the work cycle: Plan, Develop, Test, Release and Operate. We will cover a different section each time, for a total of six blog posts (including this introductory one). For your convenience, you can see the complete DevOps Ecosystem infographic



1. Plan - Planning is the initial stage, and it covers the first steps of project management. The project and product ideas are presented and analyzed, in groups, alone or on whiteboards. The developer, team and organization decide what they want and how they want it and assign tasks to developers, QA engineers, product managers, etc. This stage requires lots of analysis of problems and solutions, collaboration between team members and the ability to capture and track all that is being planned.
2. Develop - Developing is the stage where the ideas from planning are executed into code, or in other words - the ideas come to life as a product. This stage requires software configuration management, repository management and build tools, as well as automated Continuous Integration (CI) tools for incorporating this stage with the following ones.
3. Test - A crucial part that examines the product and service and makes sure they work in real time and under different conditions, even extreme ones sometimes. This stage requires many different kinds of tests, mainly functional tests, performance or load tests and service virtualization tests. It’s also important to test compatibility and integrations with 3rd party services. The data from the tests needs to be managed and analyzed in rich reports, for improving the product according to test results.
4. Release - Once a stage that stood out on its own and caused many a night with no sleep for developers, now the release stage is becoming agile and integrating with the Continuous Delivery process. Therefore, the discussion of this part can’t revolve only around tools, but rather needs to discuss methodologies as well. Regarding tools, this stage requires deployment tools, containers and release tools, as well as configuration management and abilities to work in the Cloud. The cloud provides us the foundation that we all use to achieve scalability.
5. Operate - We now have a working product. But how can we maximize the features we’ve planned, developed, tested and released? This is what this stage is for. By implementing the best UX is a big part of this, monitoring infrastructure, APMs and aggregators, and analyzing Business Intelligence (BI), this stage ensures our users get the most out of the product and can use it error-free.

Microservice


Client
Identity Provider
API Gateway
Microservices
Service
Service
Service
Service
Remote Service
CDN
Static Content
Management
Service Discovery