Course and Labs

Introduction to DevOps

Mohanraj Shanmugam

Topics Covered

DevOps Architecture

DevOps Technologies

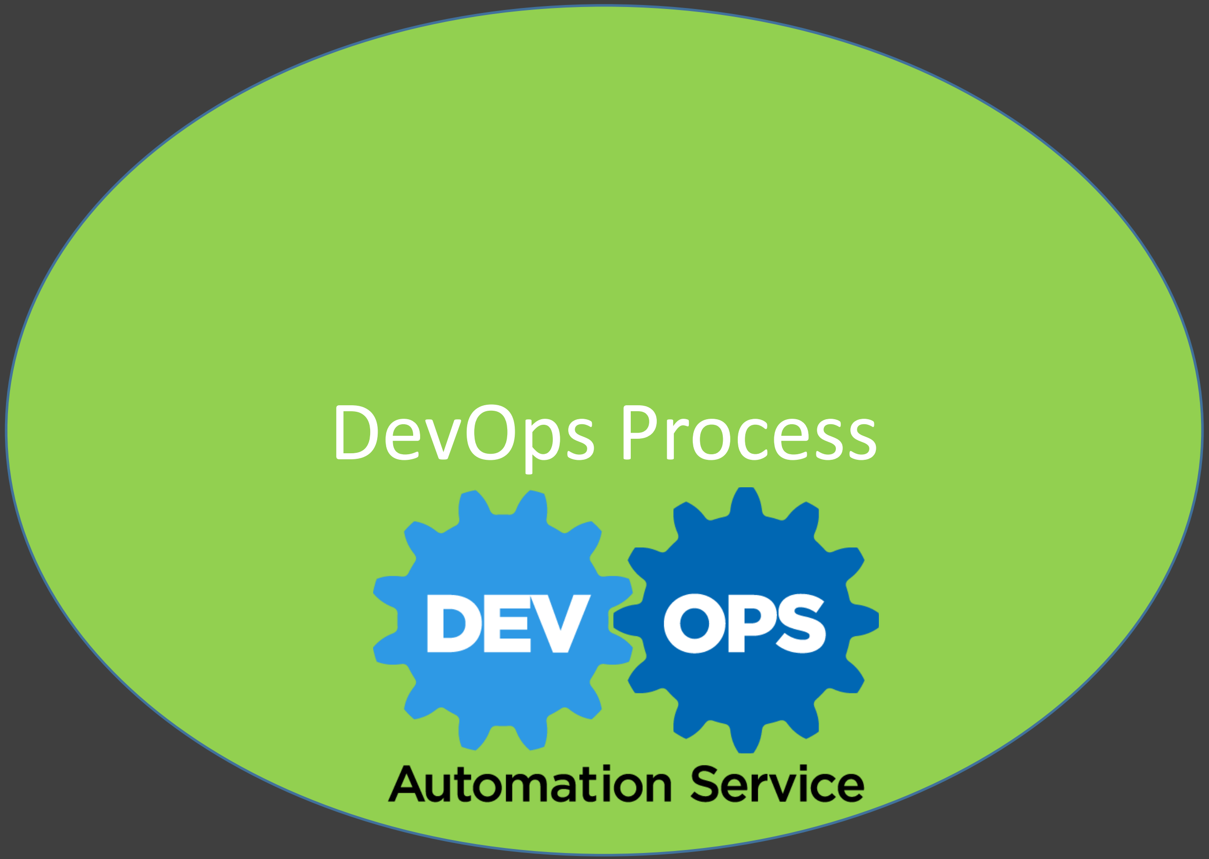
DevOps Process

DEV OPS

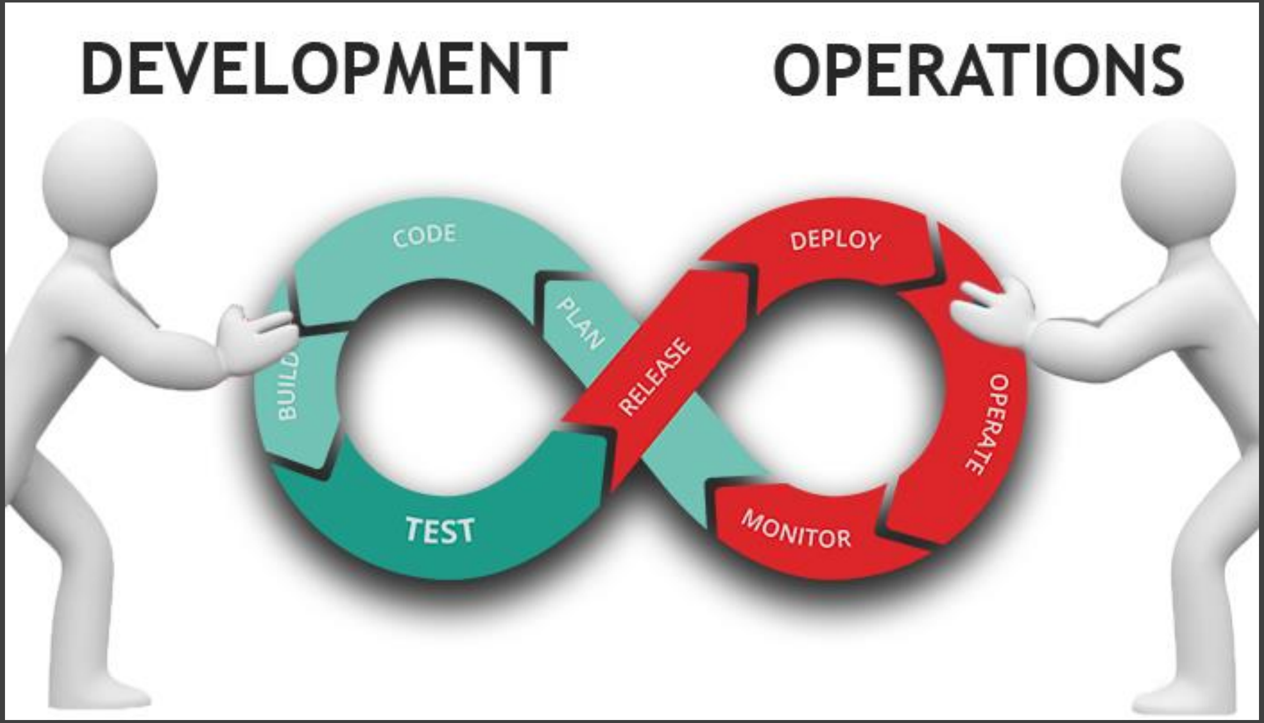
Automation Service

DevOps Teams

DevOps Process

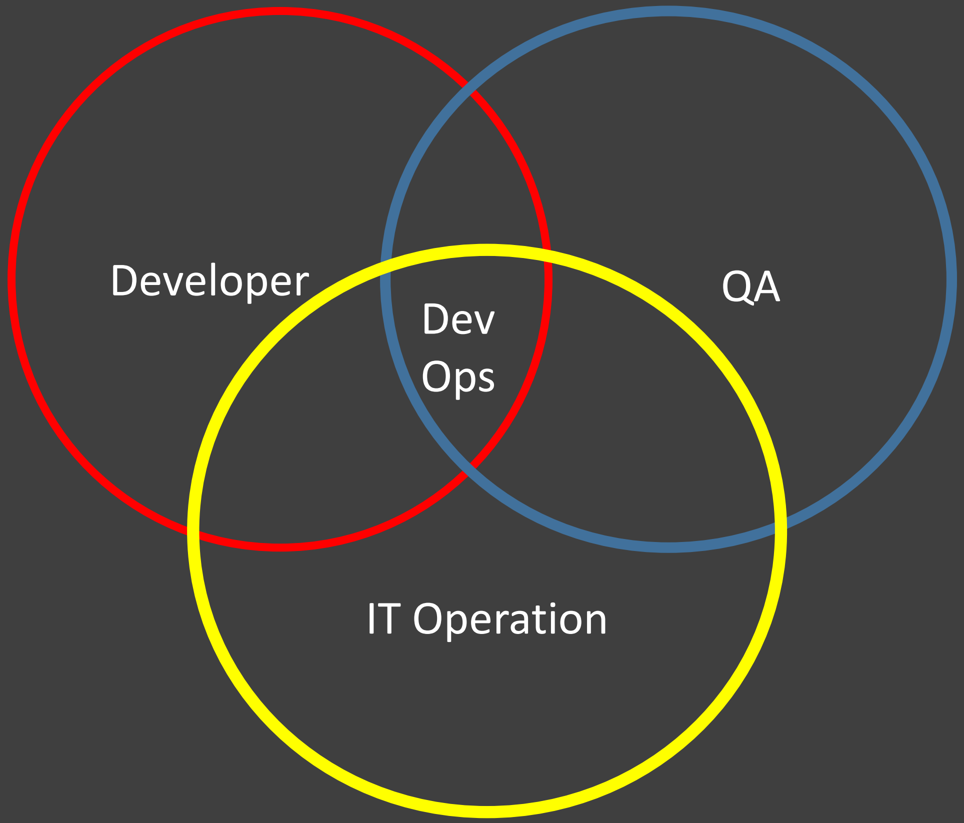


What is DevOps ?



- It is a Software Product Development, Deployment and Management method that stresses on
  - communication,
  - collaboration and
  - integration
- Between **software developers** and Information Technology(IT) Operations professionals.

What is DevOps?



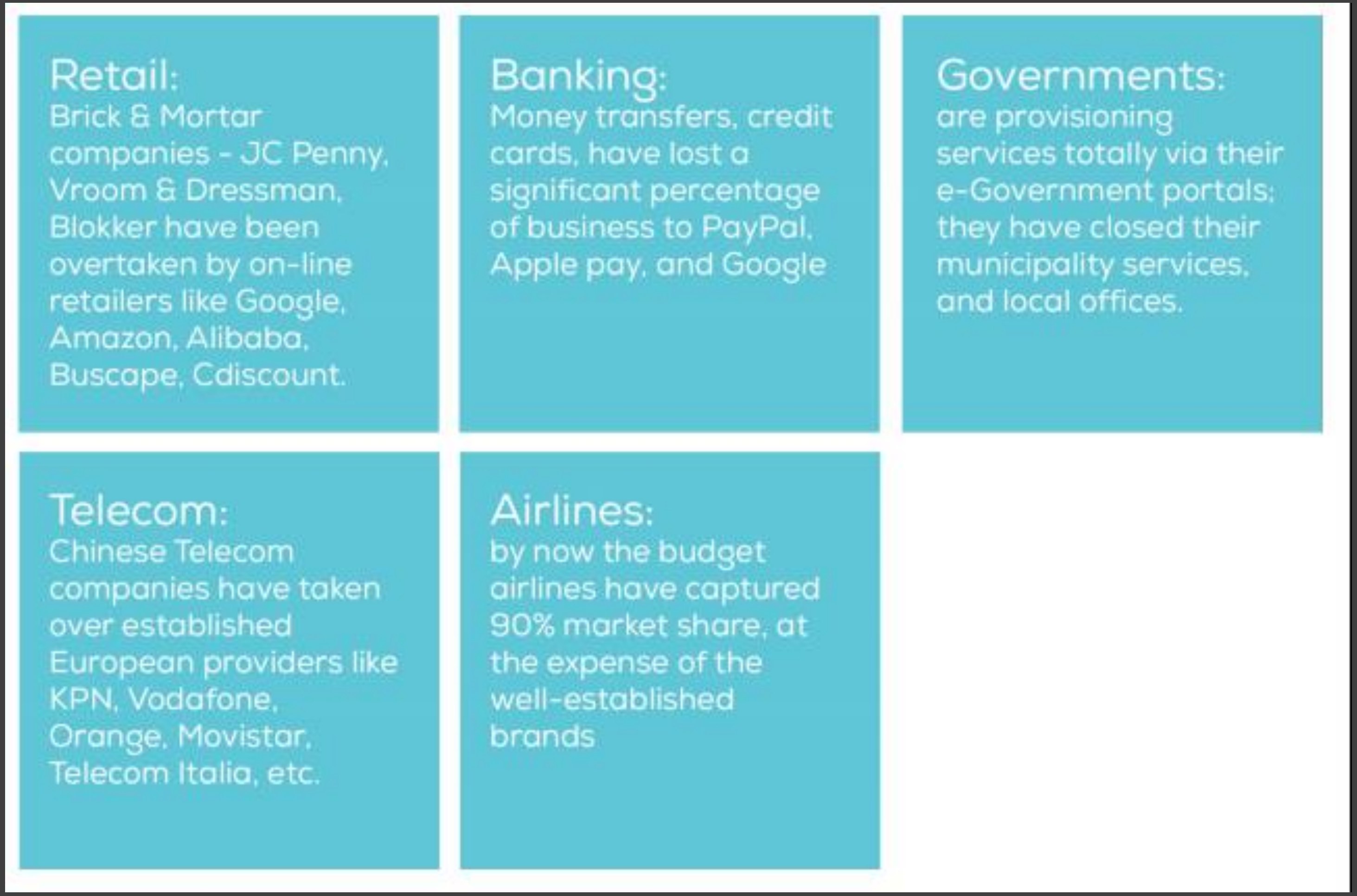
- Software developer means inclusive of Developer and QA
- IT Infrastructure Operation means Infra Engineering and Operation
- It aims at establishing a culture and environment where building, testing, and releasing software can happen rapidly, frequently, and more reliably
- By Automating the process of software delivery and infrastructure

Course and Labs

A Brief History of DevOps

- The first seeds of the DevOps movement were sown in Belgium back in 2007.
- An IT consultant named Patrick Debois was frustrated by the conflict, lack of communication and the disconnection between development and traditional IT operations departments.
- While working on a large data centre migration project for the Belgium Government Ministry, Patrick found himself straddled between development and operations.
- He was, on one hand, firefighting in the unpredictable world of IT operations, while, on the other, he would be working on agile development.

Why Enterprise have to adopt Dev Ops ?



Retail:
Brick & Mortar companies - JC Penny, Vroom & Dressman, Blokker have been overtaken by on-line retailers like Google, Amazon, Alibaba, Buscape, Cdiscount.

Banking:
Money transfers, credit cards, have lost a significant percentage of business to PayPal, Apple pay, and Google

Governments:
are provisioning services totally via their e-Government portals; they have closed their municipality services, and local offices.

Telecom:
Chinese Telecom companies have taken over established European providers like KPN, Vodafone, Orange, Movistar, Telecom Italia, etc.

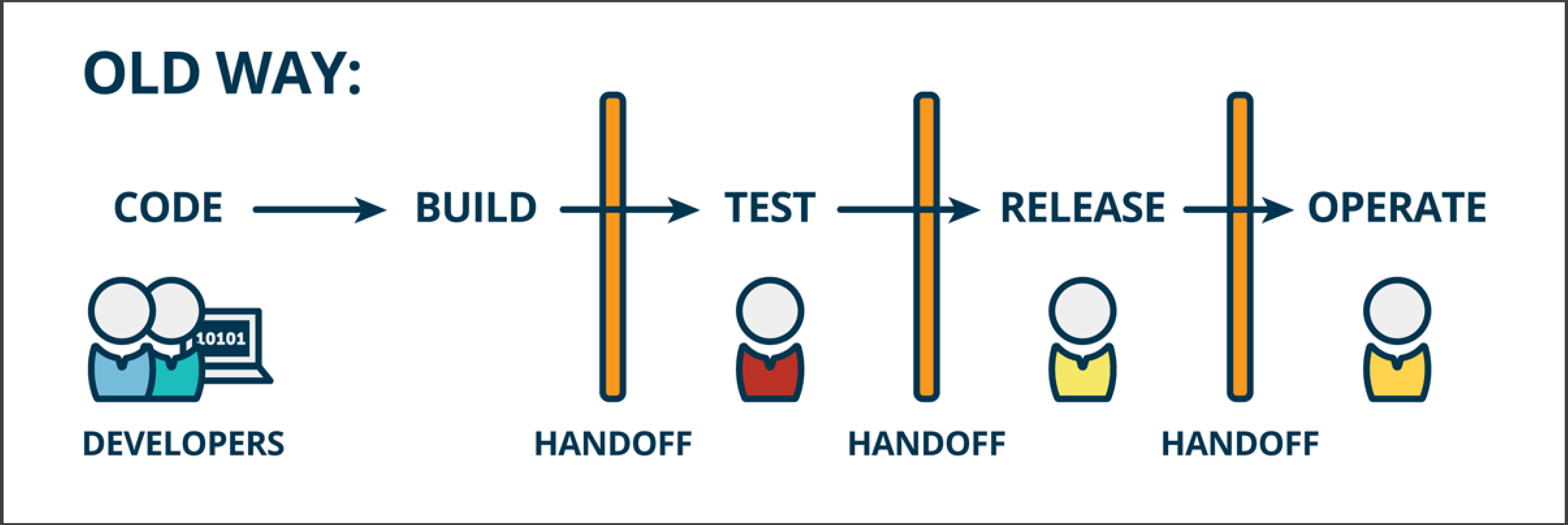
Airlines:
by now the budget airlines have captured 90% market share, at the expense of the well-established brands

- The urgency for enterprises to become a fast-mover
- low Time to Market
- Fast Concept to Cash
- Lowering Operating and Capital Expenditure

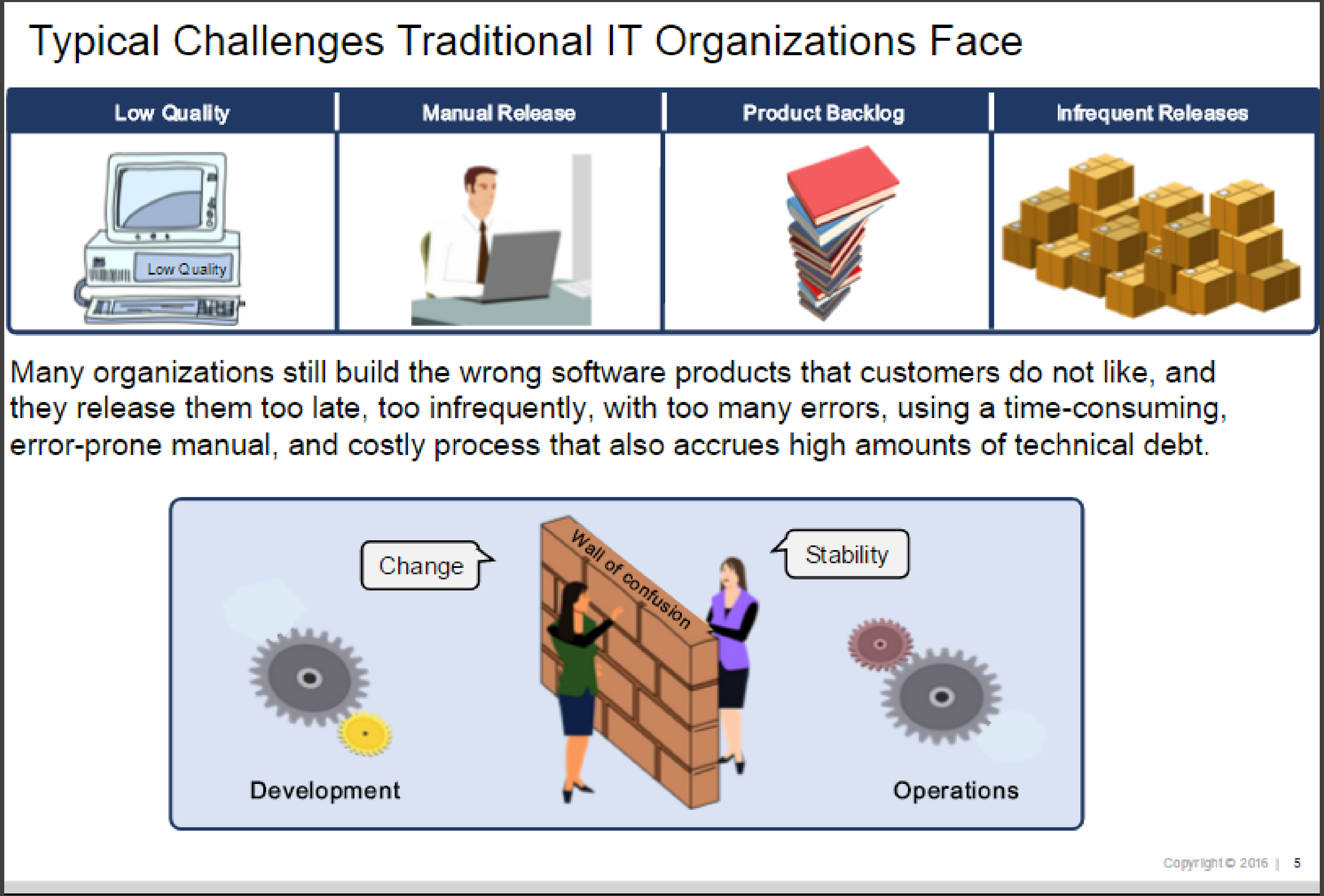
DevOps Driving Factors

- The adoption of DevOps is being driven by factors such as:
  - Use of agile and other development processes and methodologies
  - Demand for an increased rate of production releases from application and business stakeholders
  - Wide availability of virtualized and cloud infrastructure from internal and external providers
  - Increased usage of data center automation and configuration management tools

Traditional Product Development Method



Challenges in Traditional Product Development



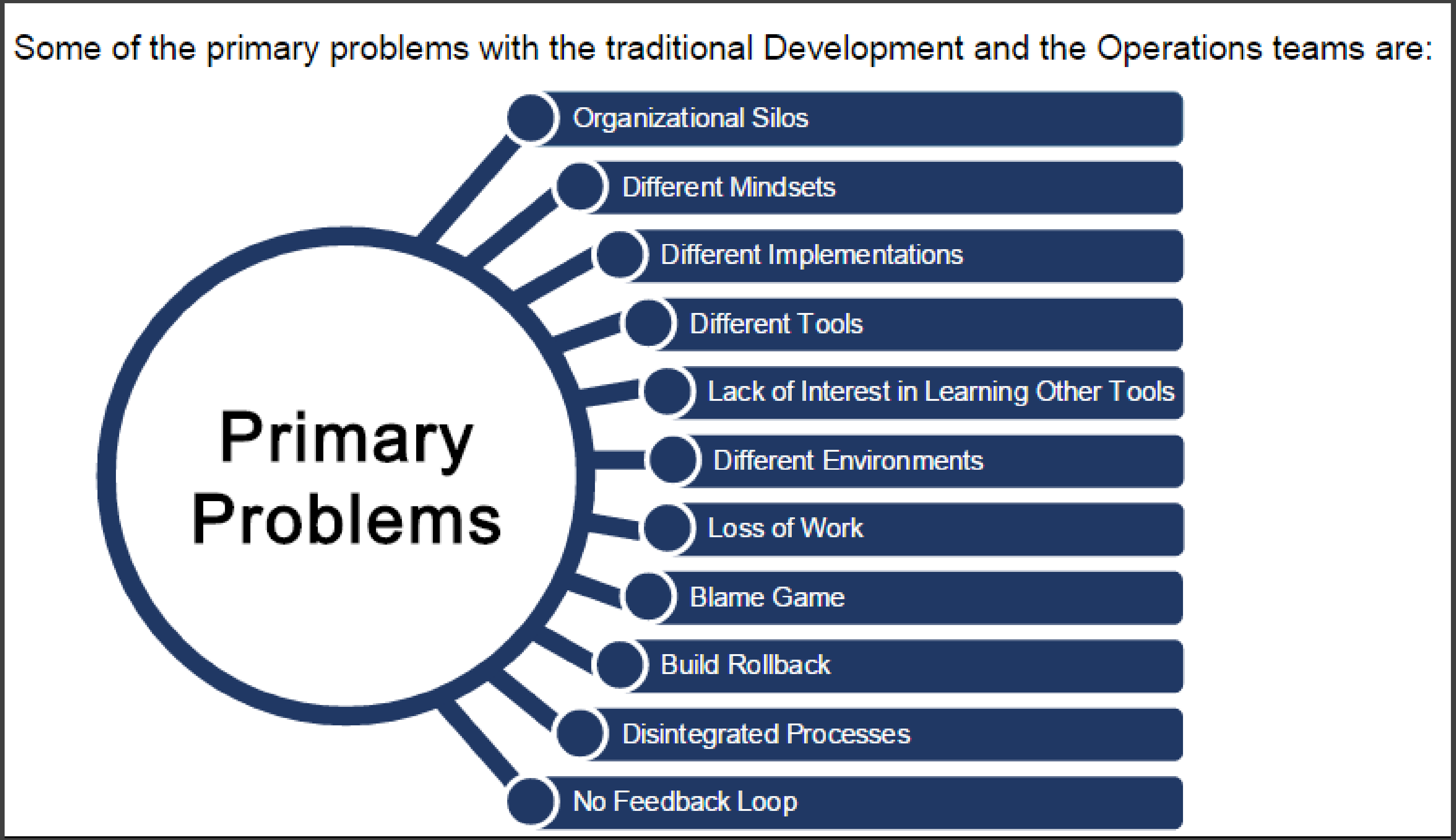
Typical Challenges Traditional IT Organizations Face

Low Quality	Manual Release	Product Backlog	Infrequent Releases

Low Quality

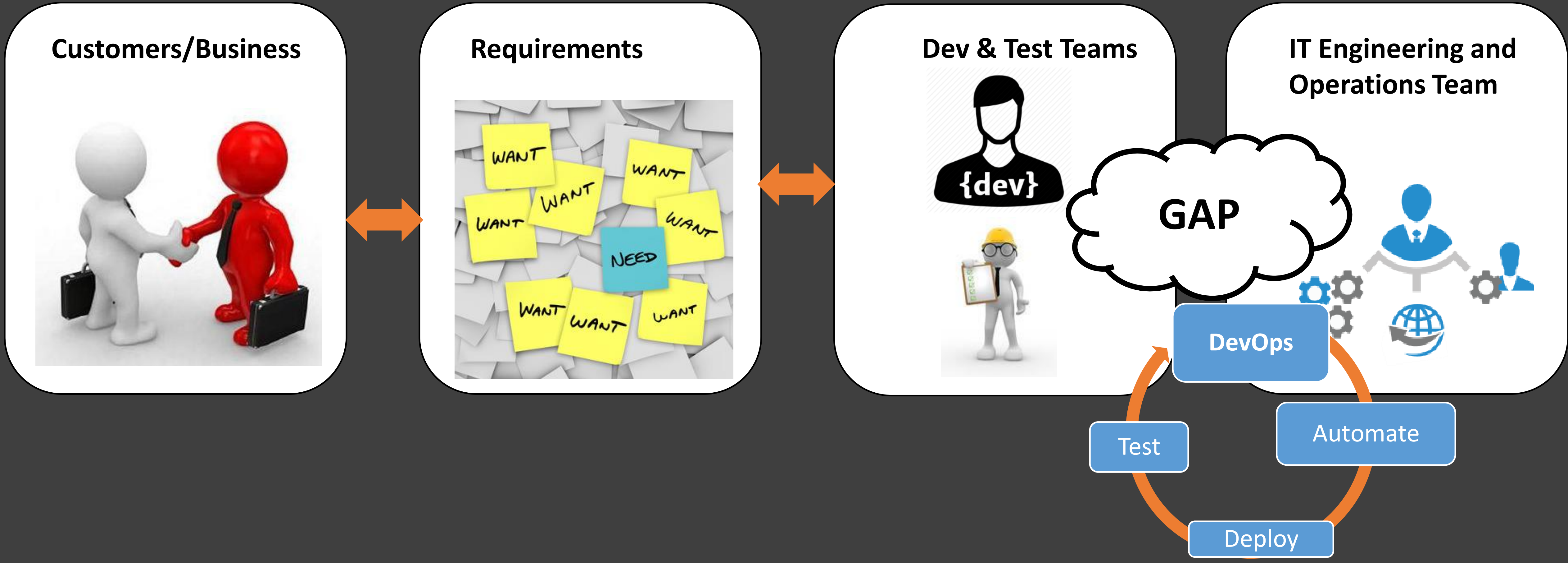
Many organizations still build the wrong software products that customers do not like, and they release them too late, too infrequently, with too many errors, using a time-consuming, error-prone manual, and costly process that also accrues high amounts of technical debt.

Problems in Traditional Product Development



Some of the primary problems with the traditional Development and the Operations teams are:

Dev Ops in End to End Product Lifecycle



Why Gaps

- Dev View:
  - Mostly delivers features after testing in development systems
  - Dev systems may not be same as production system
  - Developers will have faster turn around time w.r.t features
  - Not much concerned about the infrastructural as well as deployment impact because of the code changes

Why Gaps

- Ops View:
  - Worries more about Production Server Availability
  - Rewarded mainly for uptime
  - Lesser turn around time w.r.t feature
  - Deployment and testing due to large number of dev builds coming their way
  - Very much concerned about the infrastructural as well as deployment impact because of the code changes

Dev and Ops

- Developers work with Ops to understand the impact of code changes
- Developers now work more closely with production-equivalent systems
- Developers focuses on metrics required by Ops team
- Ops now have more clarity on infrastructure needs
- More automation on deployment
- Closely monitors the Dev – Test – Prod pipeline for each deployment with immediate feedback
- Better collaboration and communication

Principles of DevOps

- Customer-centric action.
  - It is imperative nowadays to have short feedback loops with real customers and end-users, and that all activity in building products and services centers around these clients.
  - To be able to meet these customers’ requirements, DevOps organizations require the guts to act as lean startups that innovate continuously, pivot when an individual strategy is not (or no longer) working, and constantly invests in products and services that will receive a maximum level of customer delight.

Principles of DevOps

- Create with the end in mind.
  - Organizations need to let go of waterfall and process-oriented models where each unit or individual works only for a particular role/function, without overseeing the complete picture.
  - They need to act as “product companies” that explicitly focus on building working products sold to real customers,
  - All employees need to share the engineering mindset that is required actually to envision and realize those products.

Principles of DevOps

- End-to-End responsibility.
  - Where traditional organizations develop IT solutions and then hand them over to Operations to deploy and maintain these solutions
  - In a DevOps environment teams are vertically organized such that they are fully accountable from “concept to grave”.
  - IT products or services created and delivered by these teams remain under the responsibility of these stable groups.
  - These teams also provide performance support, until they become end-of-life, which greatly enhances the level of responsibility felt and the quality of the products engineered.

Principles of DevOps

- Continuous Improvement.
  - End-to-end responsibility also means that organizations need to adapt continuously in the light of changing circumstances (e.g. customer needs, changes in legislation, new technology becomes available).
  - In a DevOps culture, a strong focus is put on continuous improvement to minimize waste optimize for speed, costs and ease of delivery, and to continuously improve the products/services offered.
  - Experimentation is therefore an important activity to embed and develop a way of learning from failures is essential.
  - A good rule to live by in that respect is “if it hurts, do it more often”.

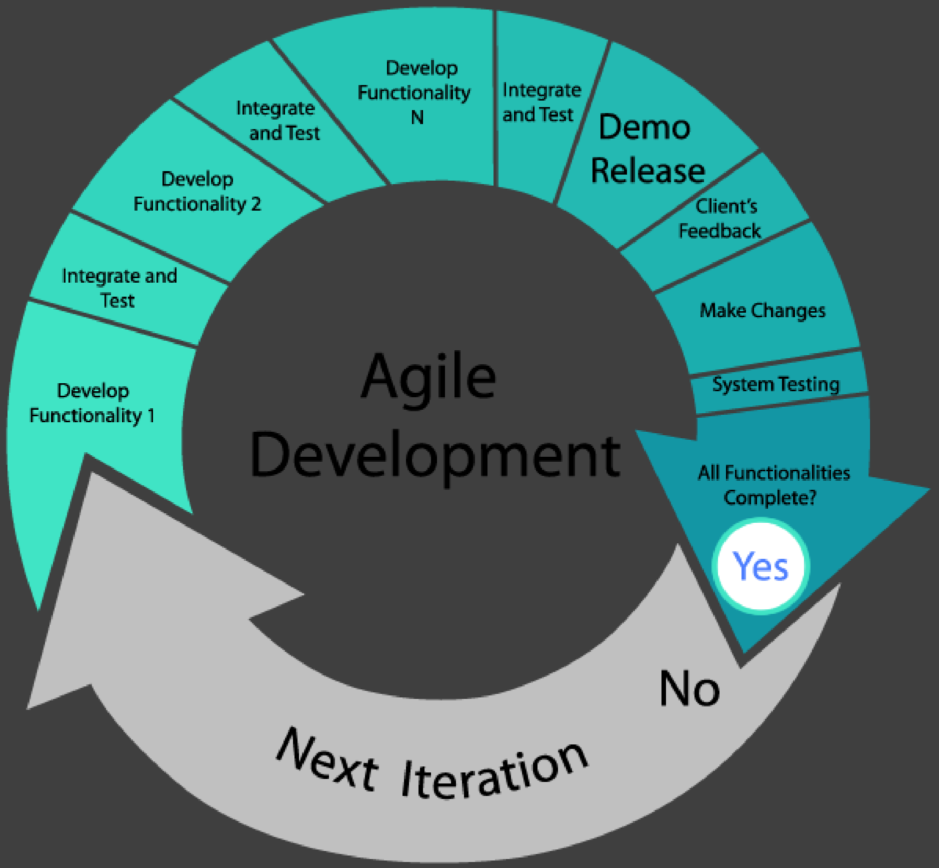
Principles of DevOps

- Automate everything you can.
  - To adopt a continuous improvement culture with high cycle rates and to create an IT organization that receives instant feedback from end users or customers, many organizations have quite some waste to eliminate.
  - Fortunately, in the past years, enormous gains in IT development and operations can be made in that respect.
  - Think of automation of not only the software development process (continuous delivery, including continuous integration and continuous deployment) but also of the whole infrastructure landscape by building next-gen container-based cloud platforms that allow infrastructure to be versioned and treated as code as well.
  - Automation is synonymous with the drive to renew the way in which the team delivers its services.

Principles sum-up

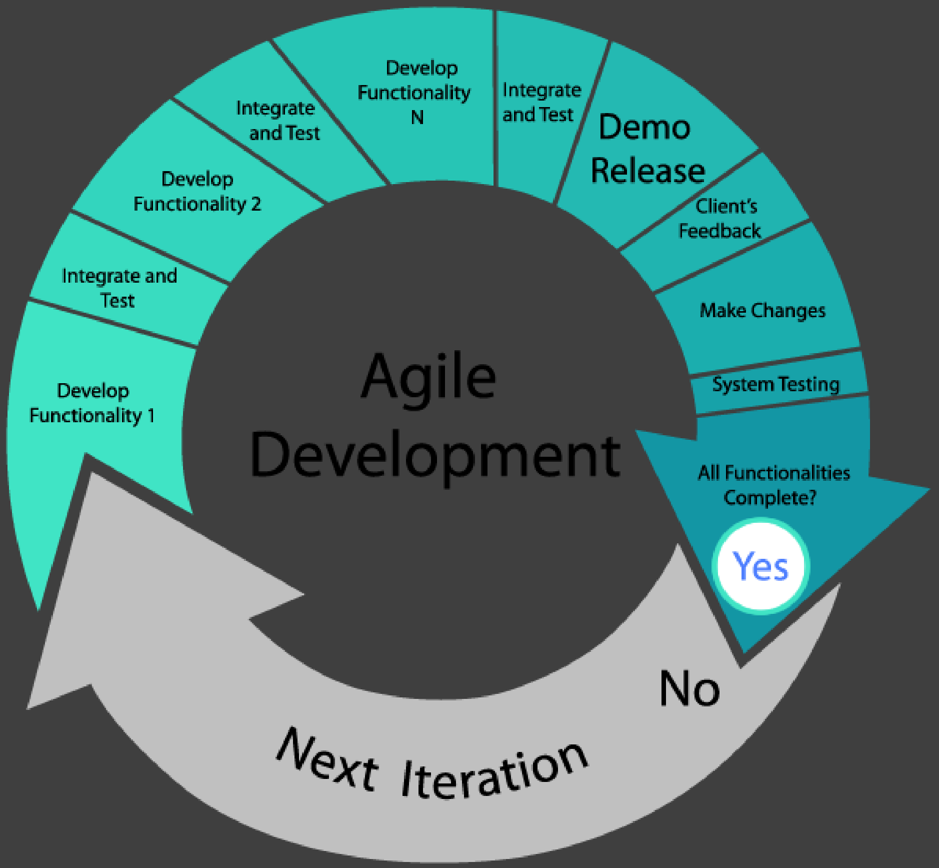
- Develop and test in an environment similar to production
- Automating Infrastructure
- Automating Workflows
- Deploy builds frequently
- Validate operation quality continuously
- Continuously Monitor Application Performance

What is Agile Development



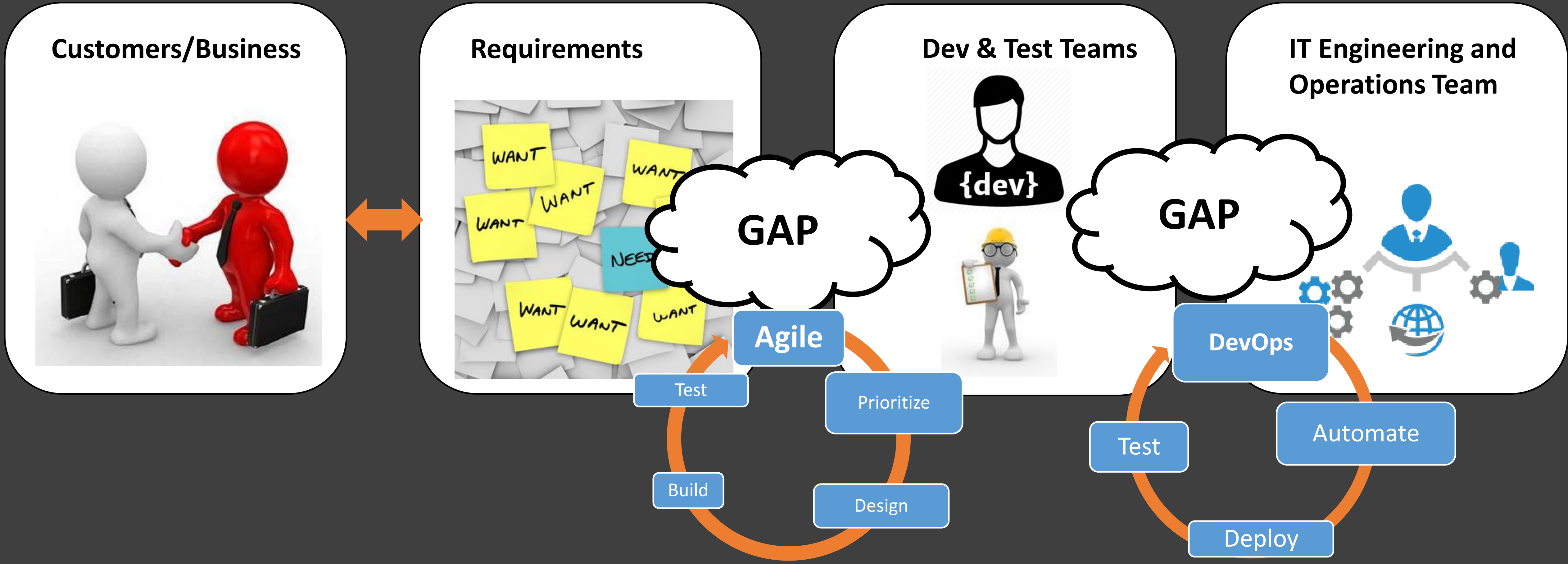
- The Agile movement seeks alternatives to traditional project management.
- Agile approaches help teams respond to unpredictability through incremental, iterative work cadences and empirical feedback.
- Agile methodologies are an alternative to waterfall, or traditional sequential development.

What is Agile Development



- It is principle for software development where requirements and solutions evolve through the collaborative effort of self-organizing cross-functional teams.
- It promotes adaptive planning, evolutionary development, early delivery, and continuous improvement, and it encourages rapid and flexible response to change

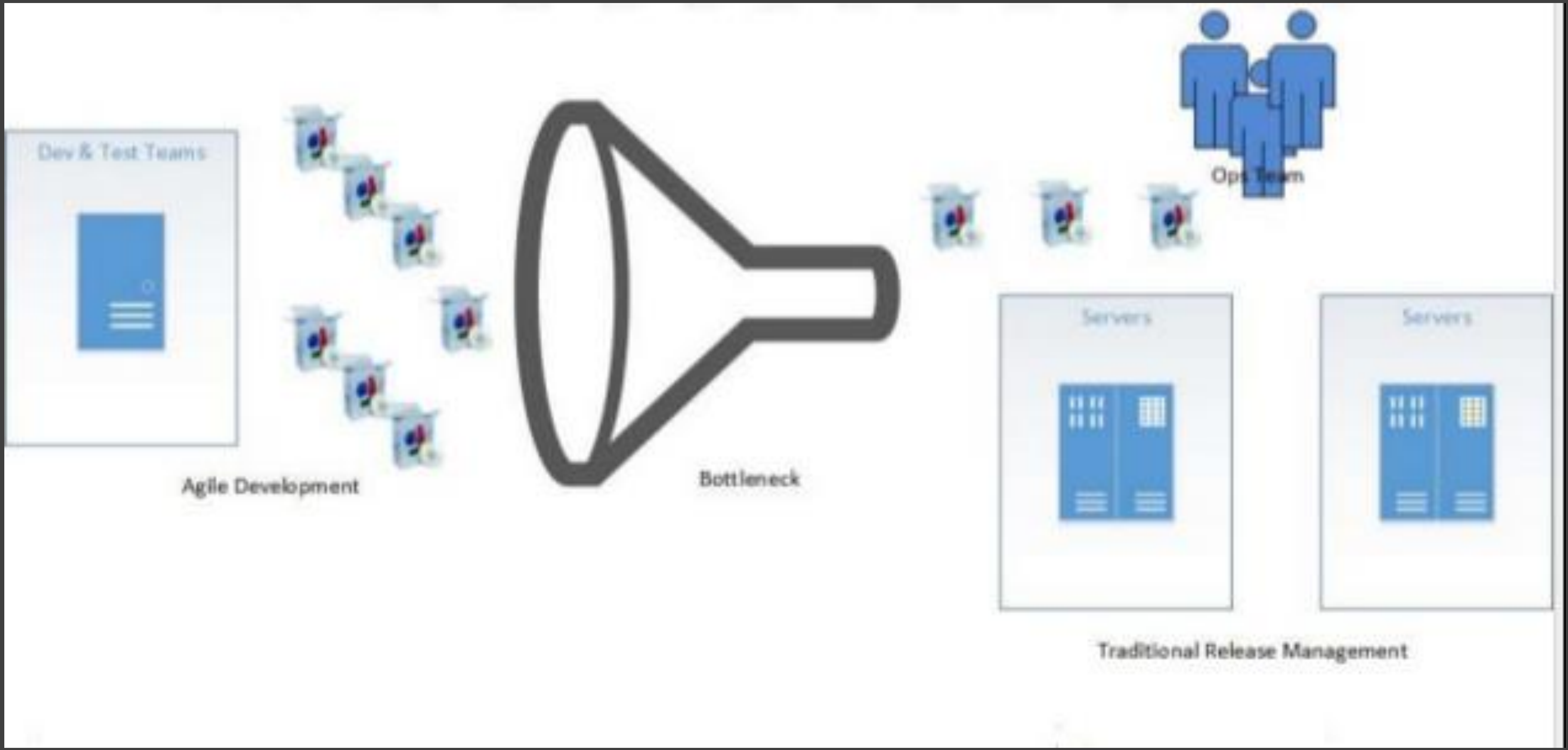
What is Agile ?



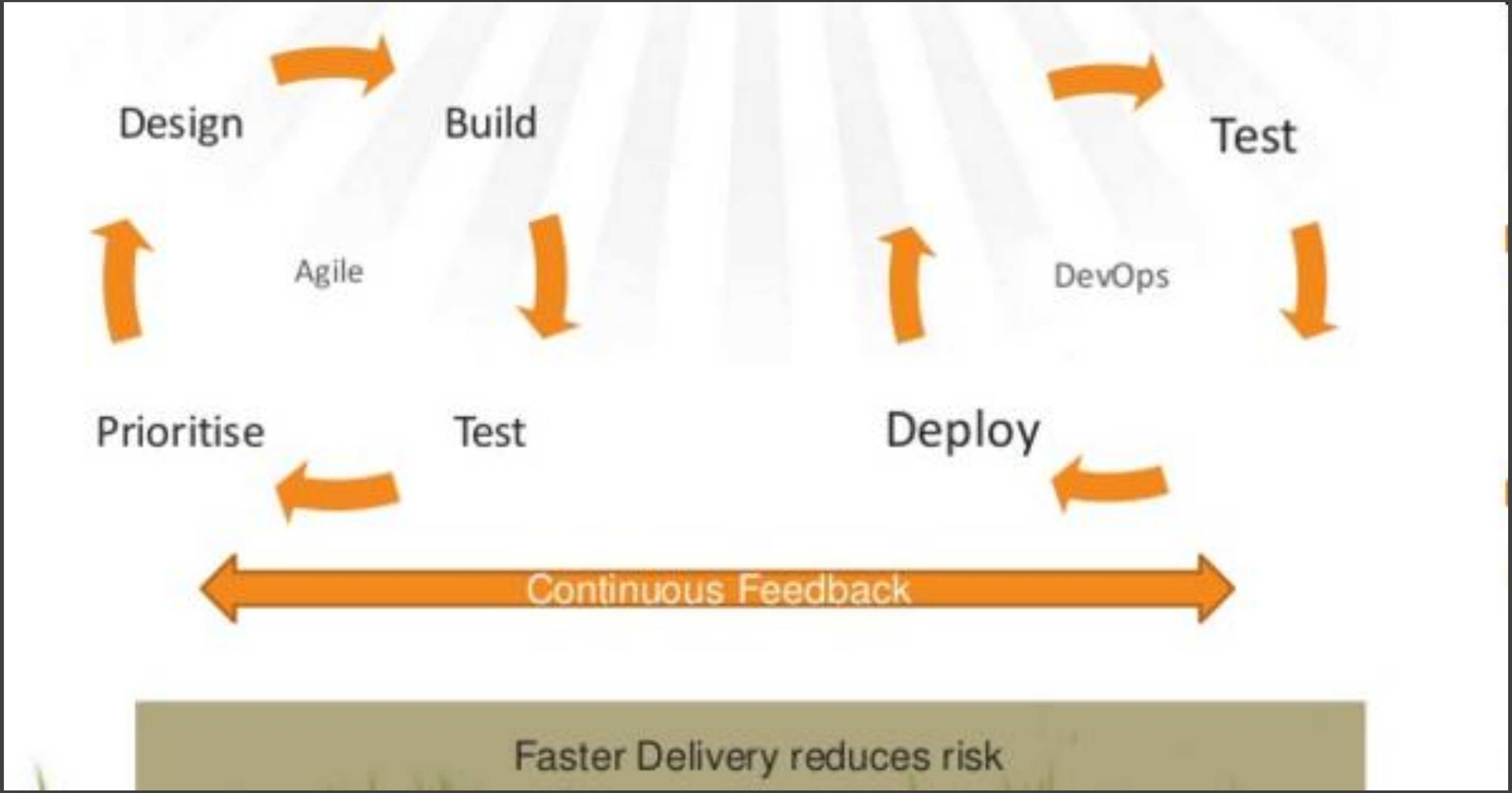
Agile and DevOps

- Agile Development–
  - Addresses the gap between customer requirements and dev + testing teams
  - Cross-functional teams to design, develop, and test features/stories prioritized by the Customer
  - Focuses more on functional and non-functional readiness
- DevOps–
  - Addresses the gap between dev + testing and Ops
  - Automated release management
  - Focuses on functional and non-functional plus operational and business readiness
  - Intensifies reusability and automation

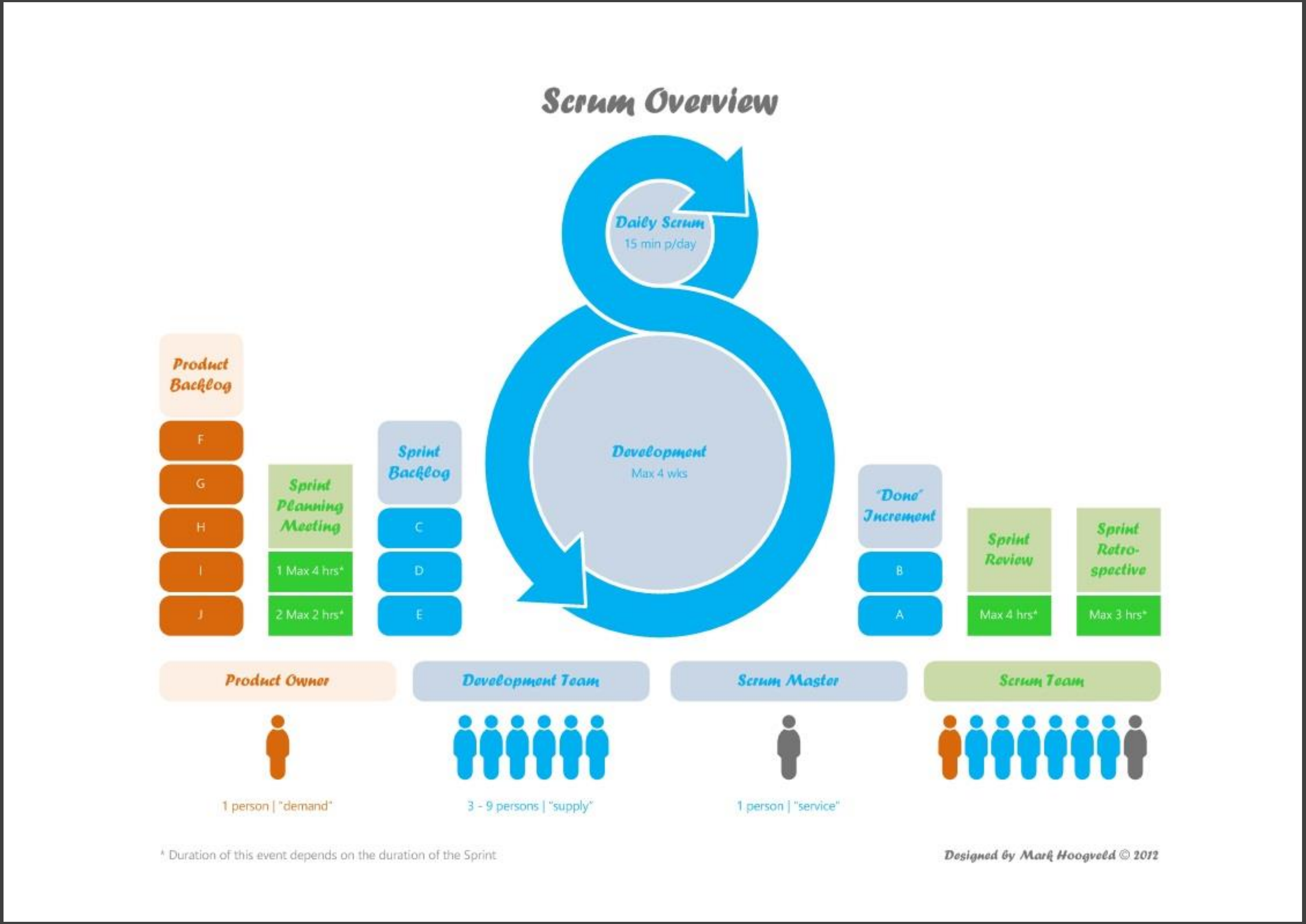
Agile and Traditional Operations Model



Agile with Dev Ops



What is Scrum?



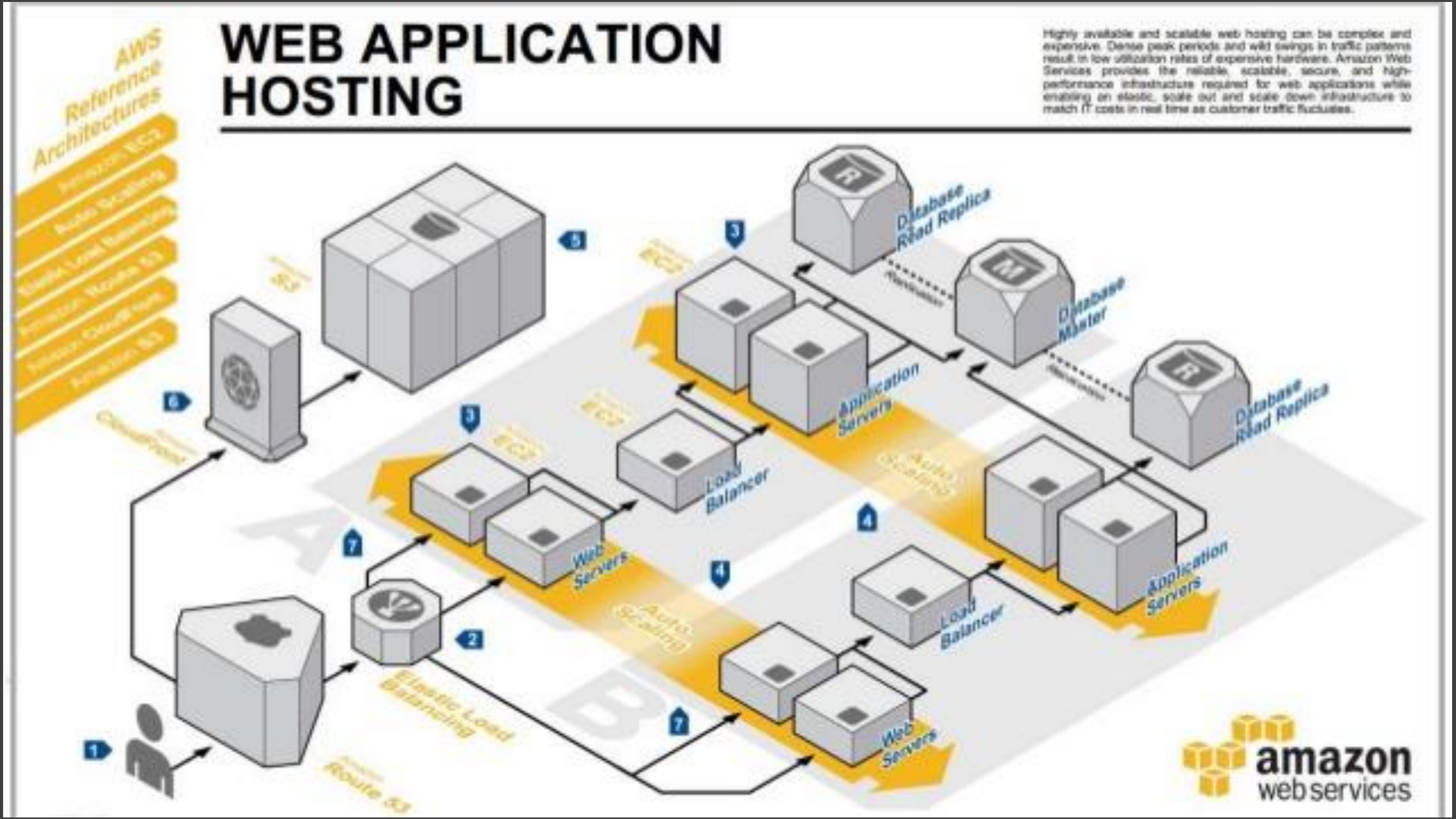
What is Scrum?

- Scrum is an Agile framework for completing complex projects.
- Scrum originally was formalized for software development projects, but it works well for any complex, innovative scope of work.
- Scrum is a process framework that has been used to manage complex product development since the early 1990s.
- Scrum is not a process or a technique for building products; rather, it is a framework within which you can employ various processes and techniques.
- Scrum makes clear the relative efficacy of your product management and development practices so that you can improve.
- The Scrum framework consists of Scrum Teams and their associated roles, events, artifacts, and rules.
- Each component within the framework serves a specific purpose and is essential to Scrum's success and usage.
- The rules of Scrum bind together the events, roles, and artifacts, governing the relationships and interaction between them.

Application Architecture for DevOps

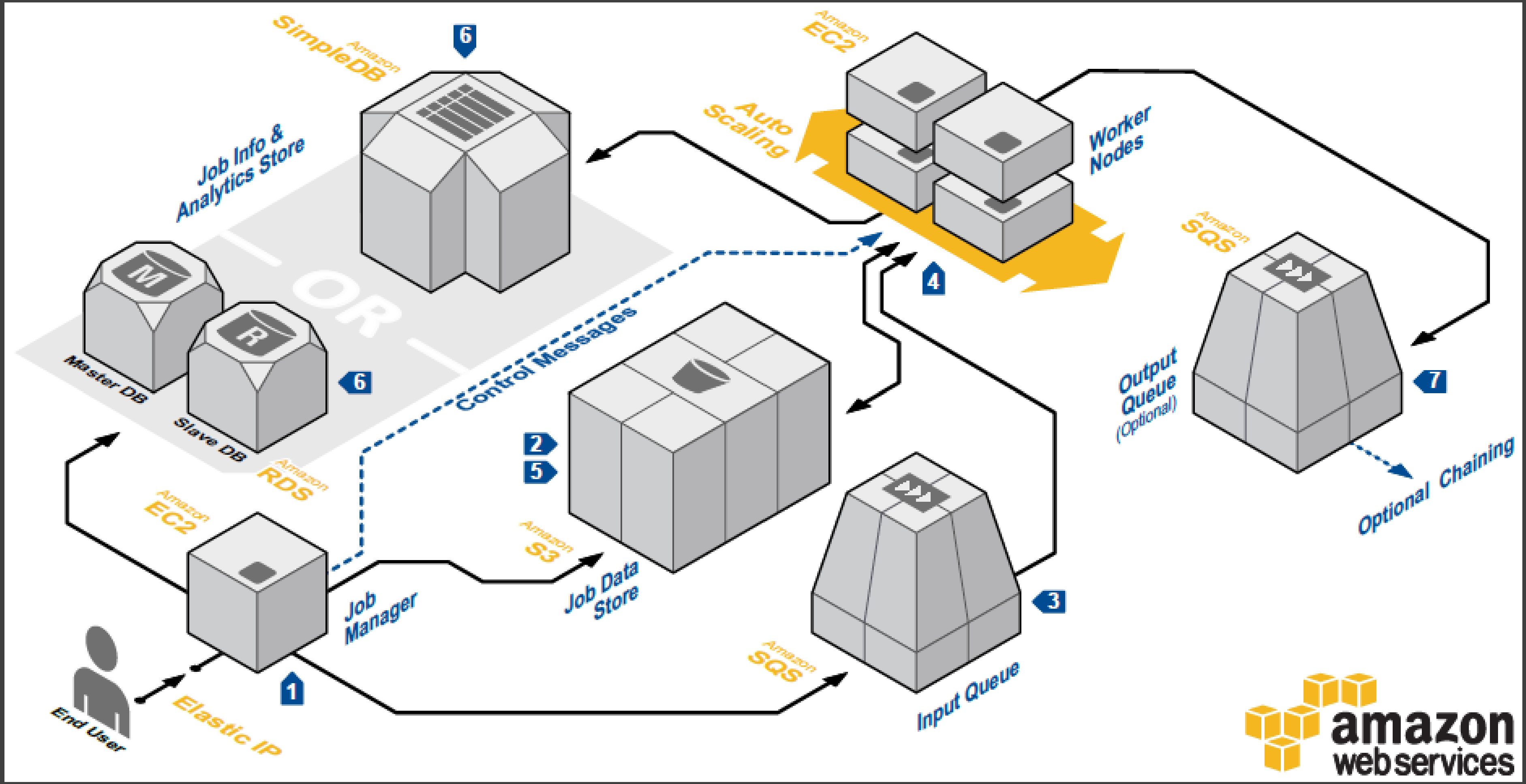
- There is need for Re Architecture the application in order to deploy Successful DevOps in your Organization.
  - Cloud Based Scalable Architecture
  - State Less Application for Web and App
  - Full Rest FULL API Compliant
  - Microservices Architecture

Cloud based Scalable Architecture

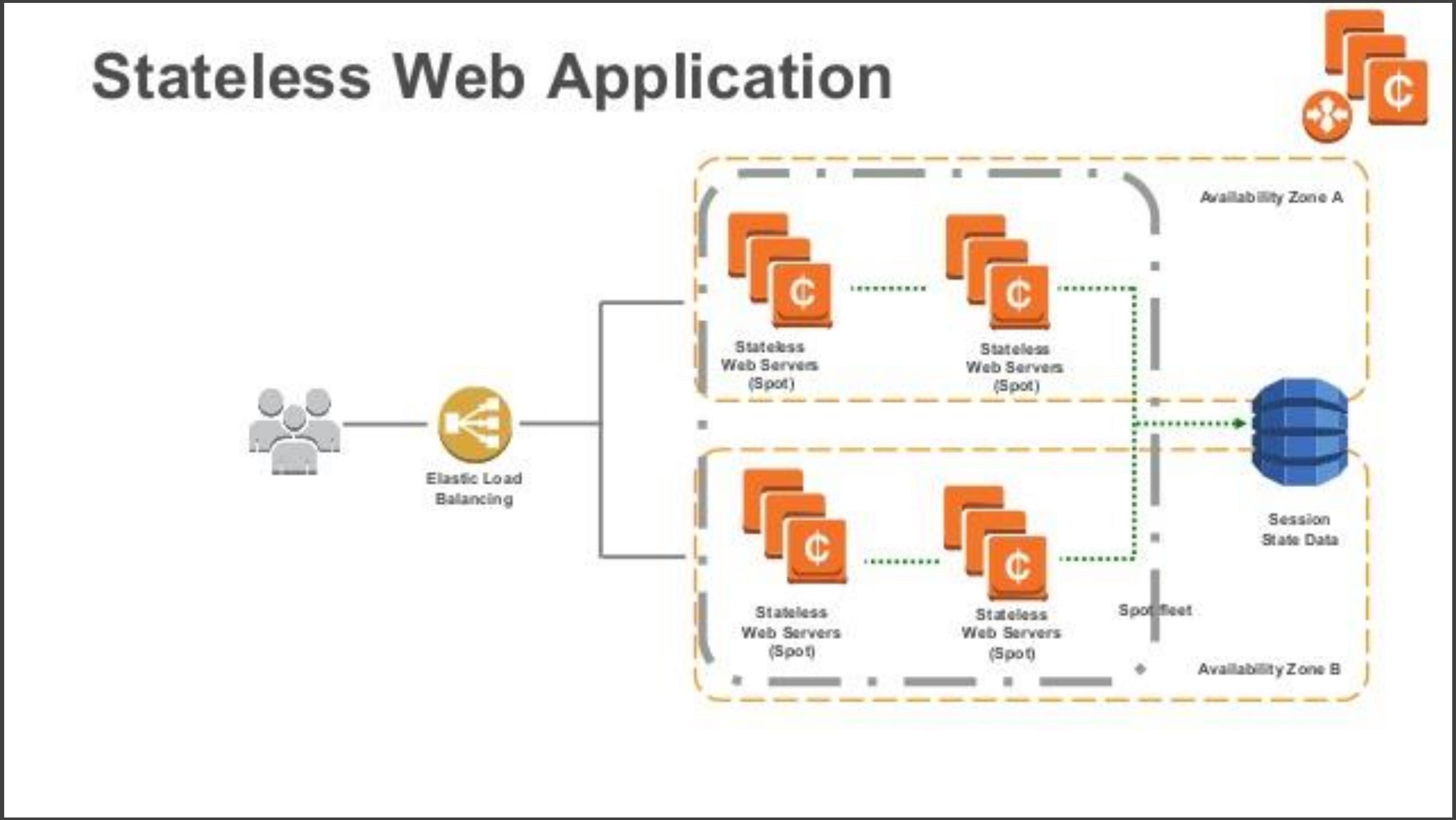


Highly available and scalable web hosting can be complex and expensive. Dense peak periods and wild swings in traffic patterns result in low utilization rates of expensive hardware. Amazon Web Services provides the reliable, scalable, secure, and high-performance infrastructure required for web applications while enabling an elastic, scale out and scale down infrastructure to match IT costs in real time as customer traffic fluctuates.

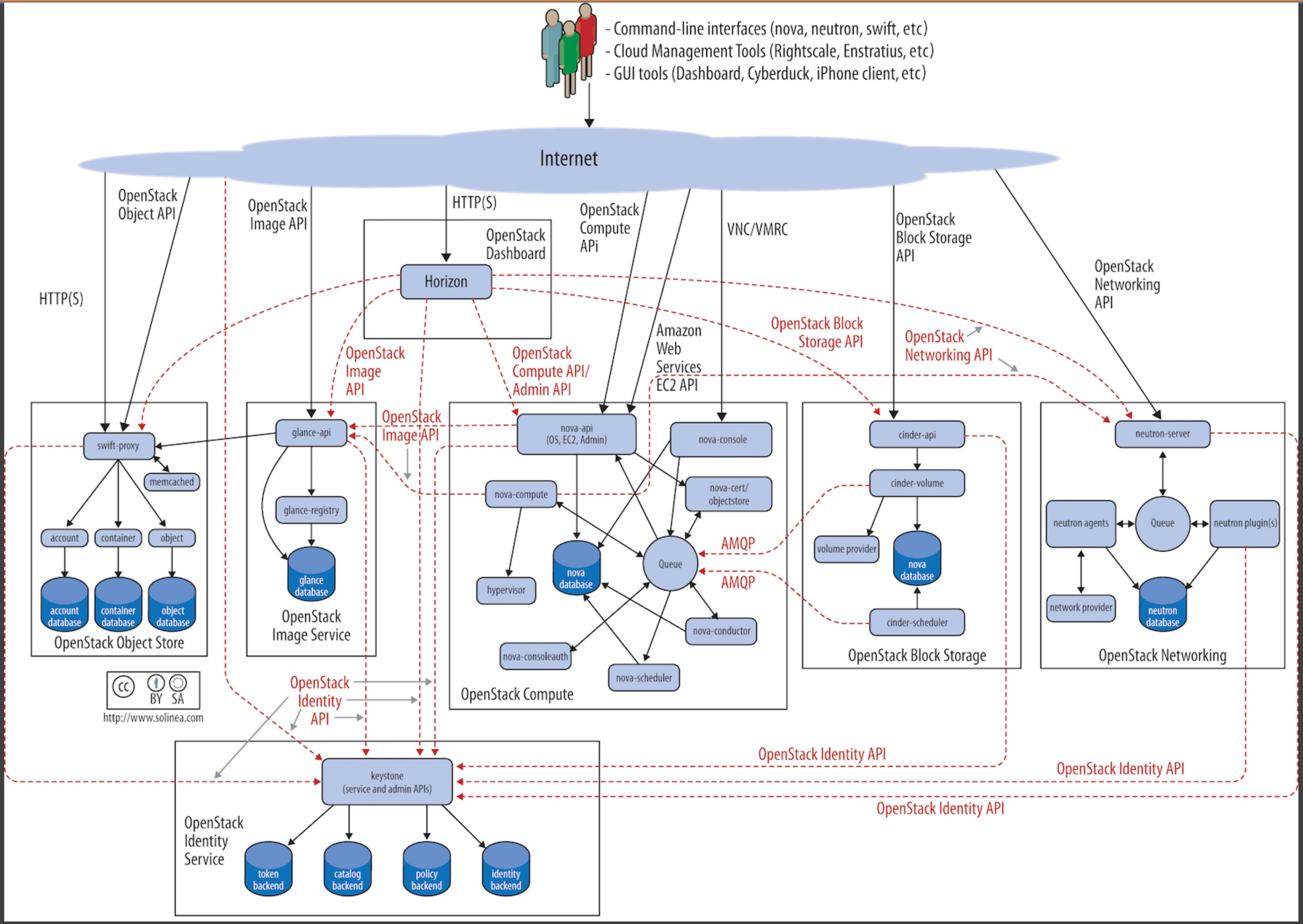
Cloud based Scalable Architecture



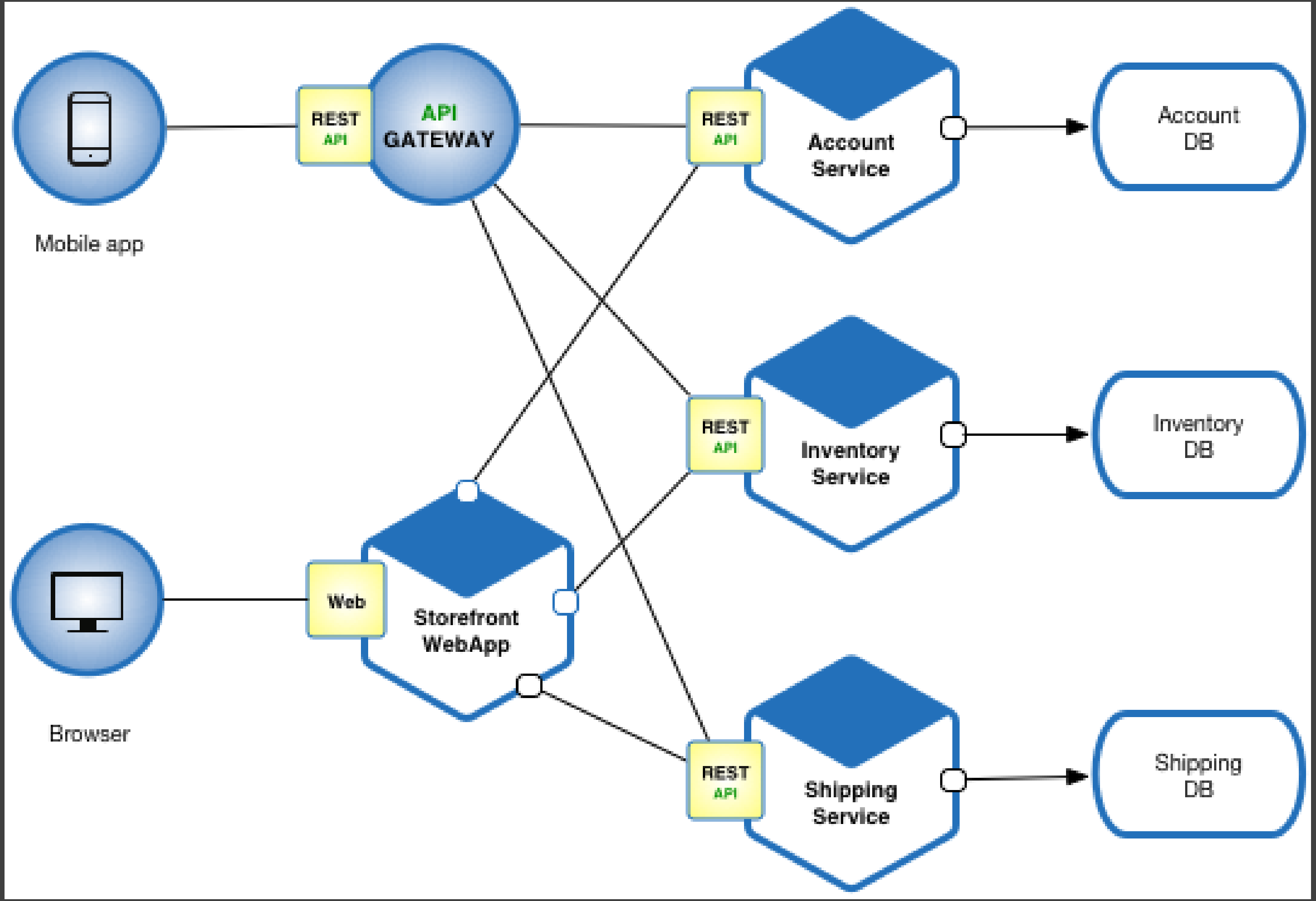
Stateless Architecture


Stateless Web Application
Availability Zone A
Stateless Web Servers (Spot)
Stateless Web Servers (Spot)
Elastic Load Balancing
Session State Data
Stateless Web Servers (Spot)
Stateless Web Servers (Spot)
Spot fleet
Availability Zone B

Restfull API Architecture



Microservices Architecture

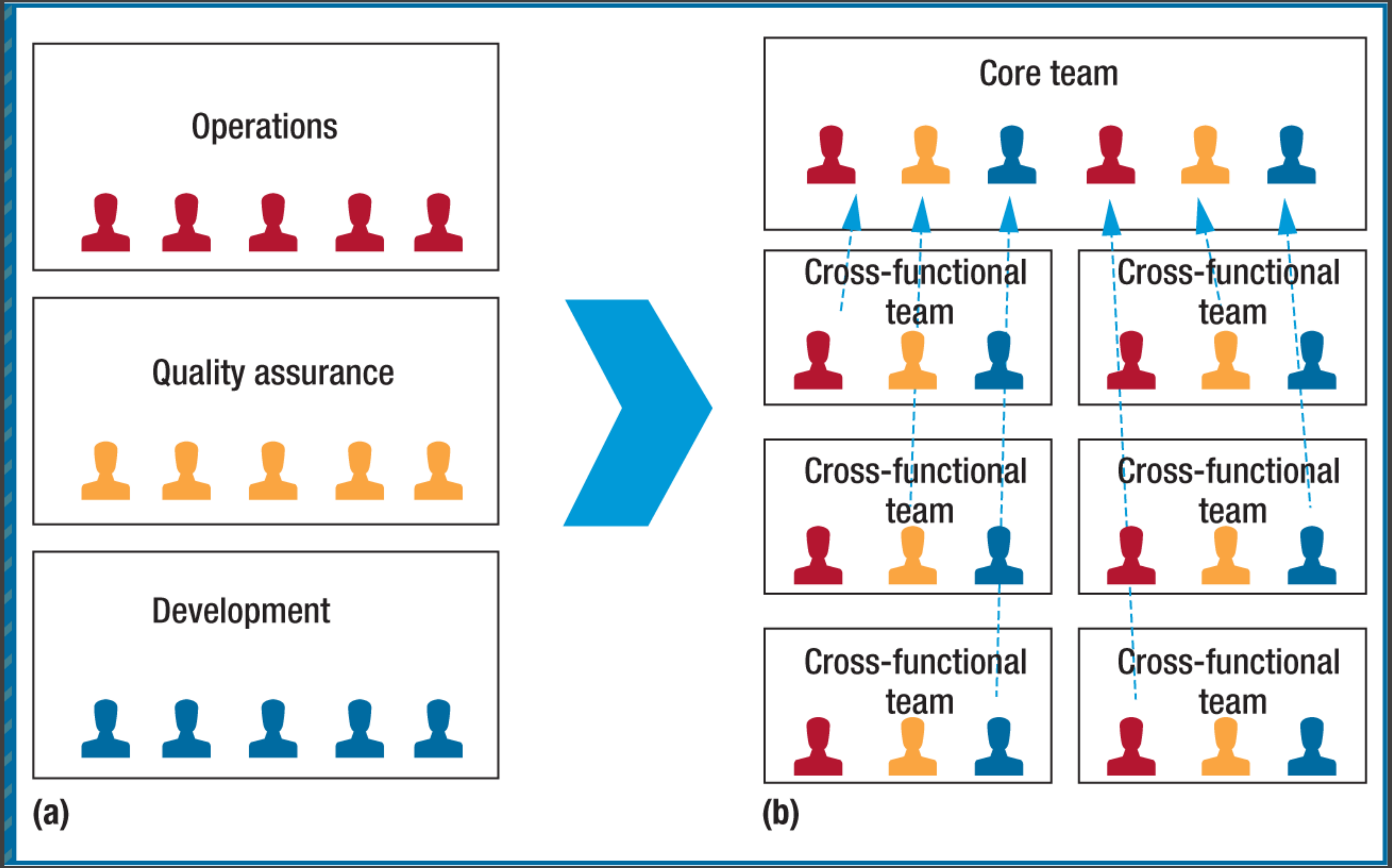


DevOps Team

DevOps Team

- Cross-functional autonomous teams.
  - In product organizations with vertical, fully responsible teams, these teams need to be entirely independent throughout the whole lifecycle.
  - That requires a balanced set of skills and also highlights the need for team members with “T-shaped” all-round profiles instead of old-school IT specialists who are only knowledgeable or skilled in for example testing, requirements analysis or coding.
  - These teams become a hotbed of personal development and growth.

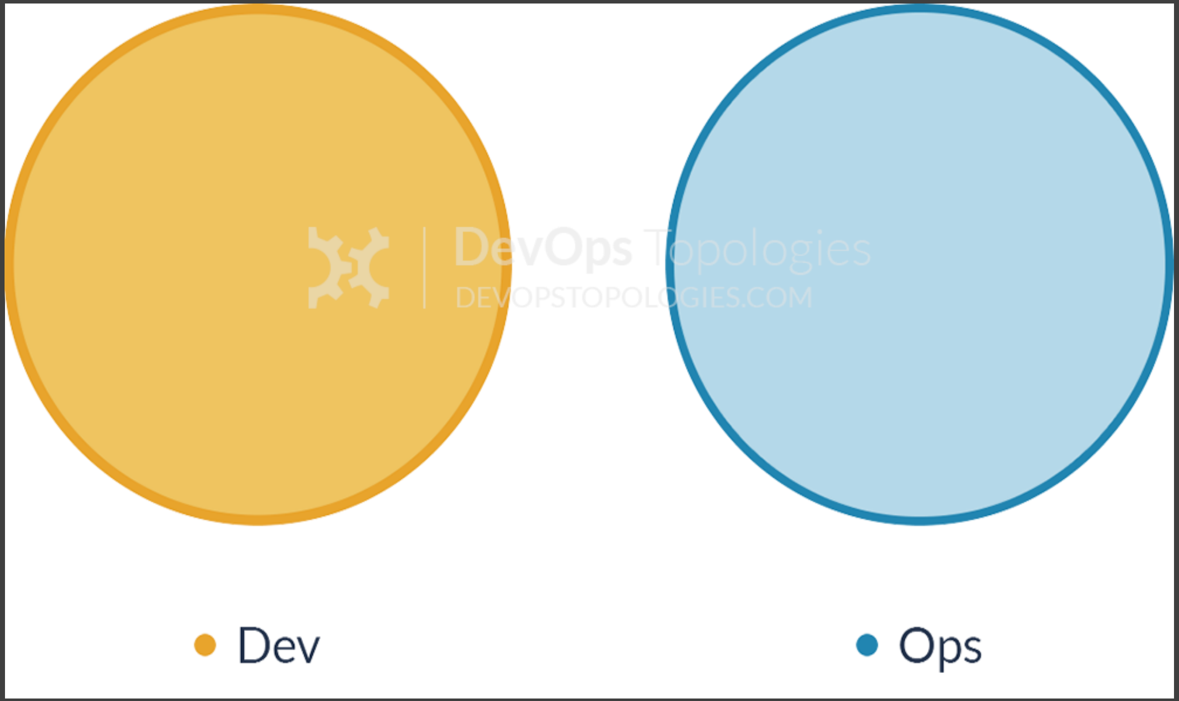
DevOps Team


Operations
Quality assurance
Development
(a)
Core team
Cross-functional team
Cross-functional team
Cross-functional team
Cross-functional team
Cross-functional team
Cross-functional team
(b)

DevOps Anti-Types

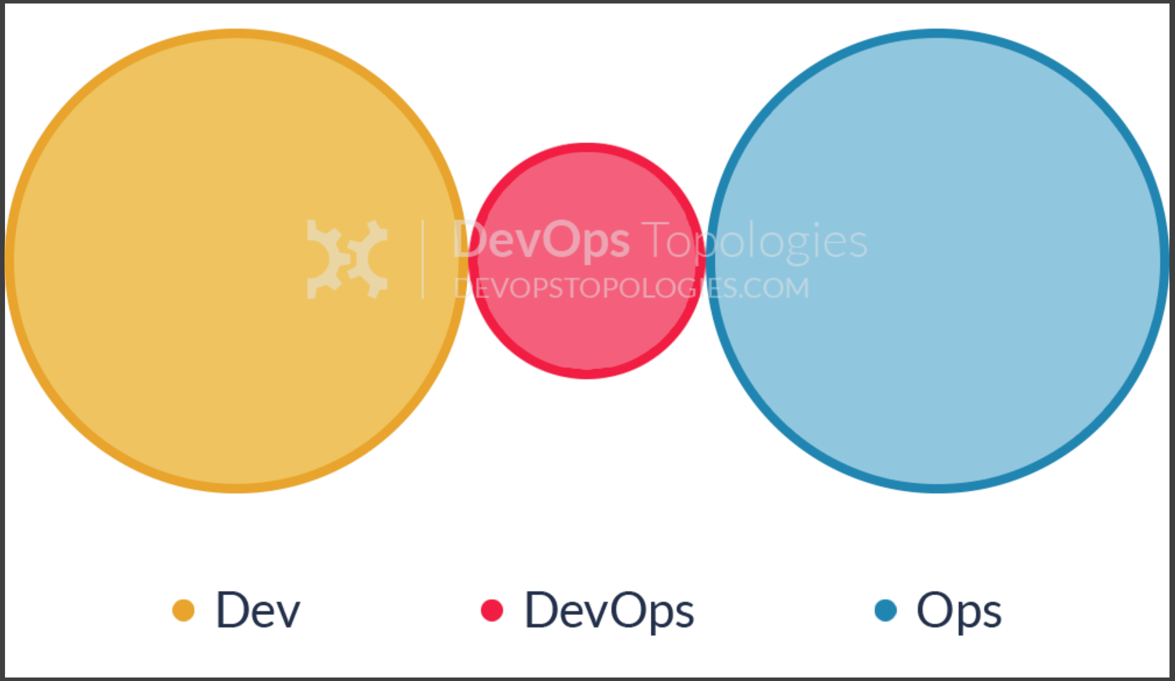
- **Anti-Type A:** Dev and Ops Silos

  Typical Dev and Ops Working in Silos



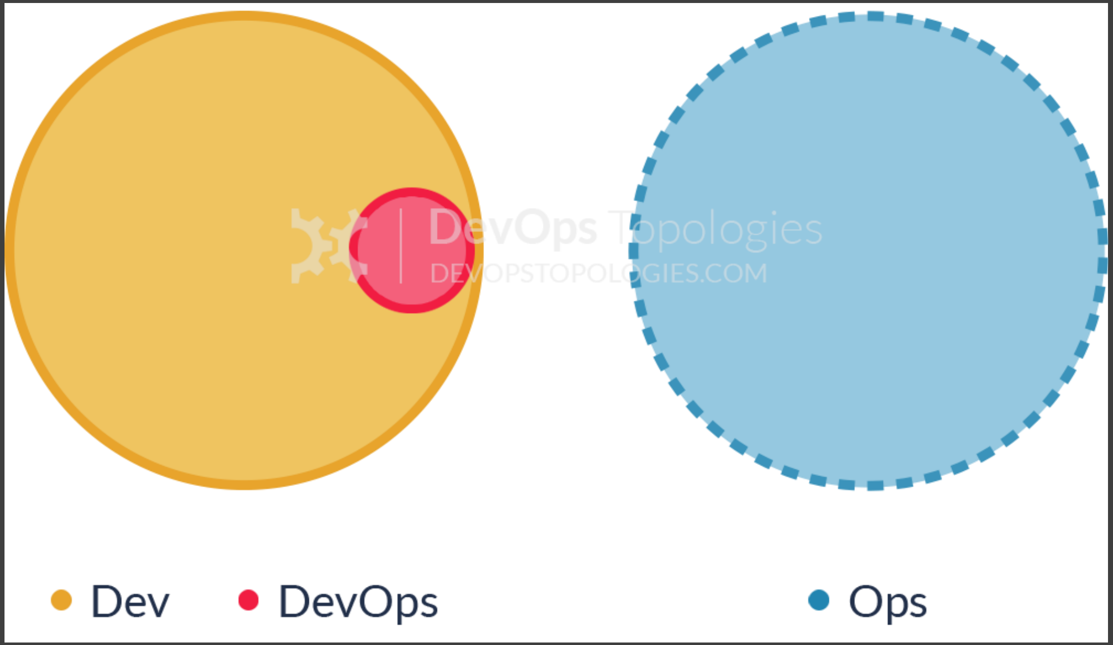
Anti-Type B: DevOps Team Silo

- The DevOps Team Silo (Anti-Type B) typically results from a manager or exec deciding that they "need a bit of this DevOps thing" and starting a 'DevOps team' (probably full of people known as 'a DevOp').
- The members of the DevOps team quickly form another silo, keeping Dev and Ops
- They defend their corner, skills, and toolset from the 'clueless Devs' and 'dinosaur Ops' people.



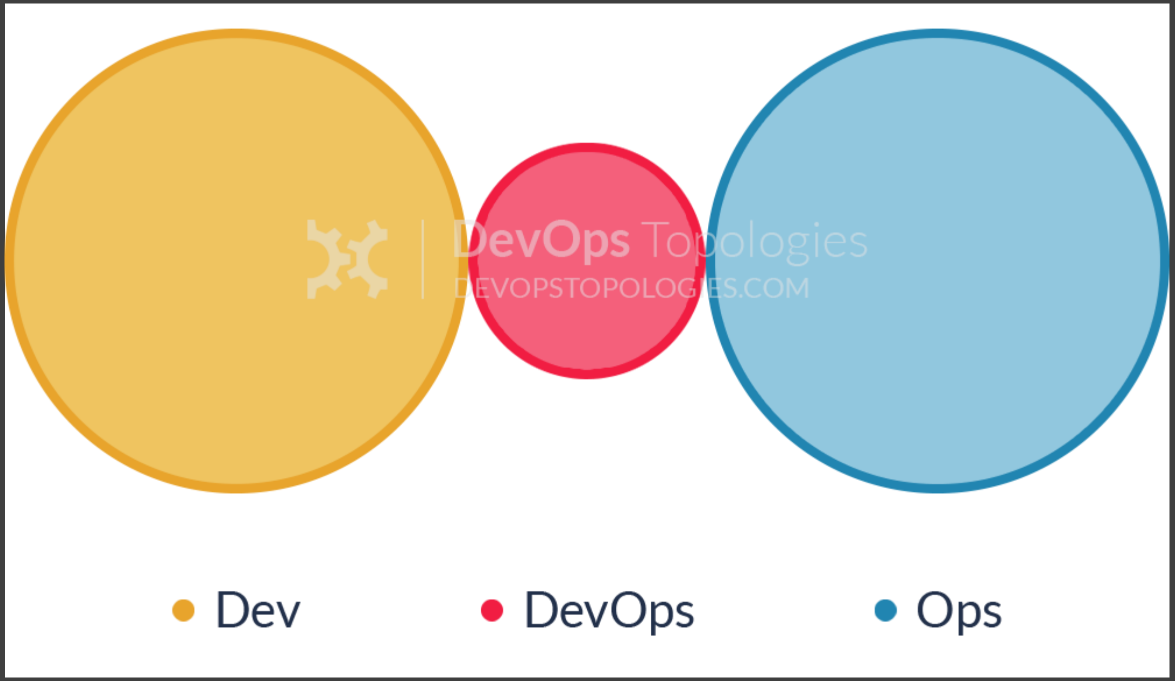
Anti-Type C: Dev Don't Need Ops

- The DevOps Team Silo (Anti-Type B) typically results from a manager or exec deciding that they "need a bit of this DevOps thing" and starting a 'DevOps team' (probably full of people known as 'a DevOp').
- The members of the DevOps team quickly form another silo, keeping Dev and Ops
- They defend their corner, skills, and toolset from the 'clueless Devs' and 'dinosaur Ops' people.



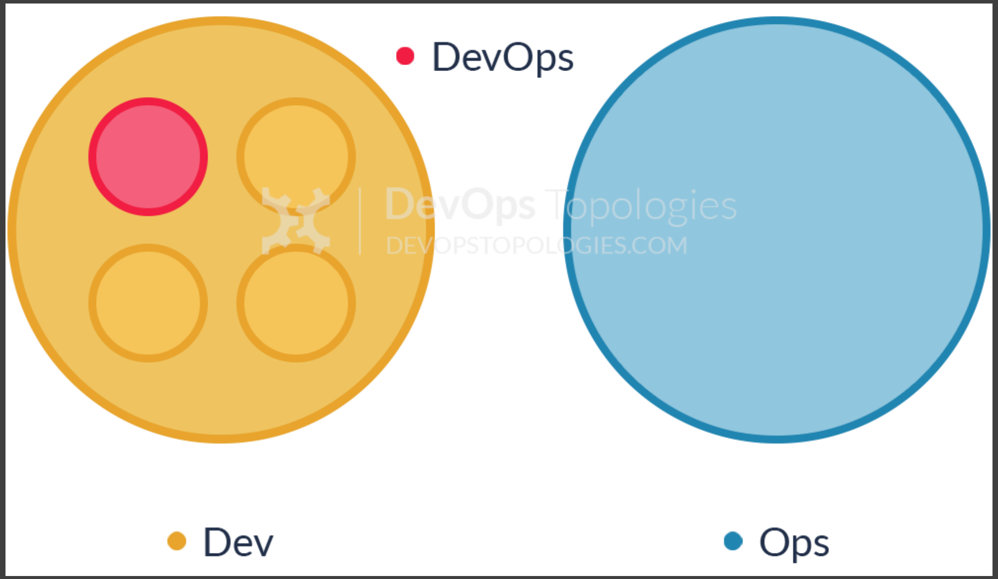
Anti-Type C: Dev Don't Need Ops

- This topology is borne of a combination of naivety and from developers and development managers particularly when starting on new projects or systems.
- Assuming that Ops is now a thing of the past ("we have the Cloud now, right?"), the developers wildly underestimate the complexity and importance of operational skills and activities, and believe that they can do without them, or just cover them in spare hours.



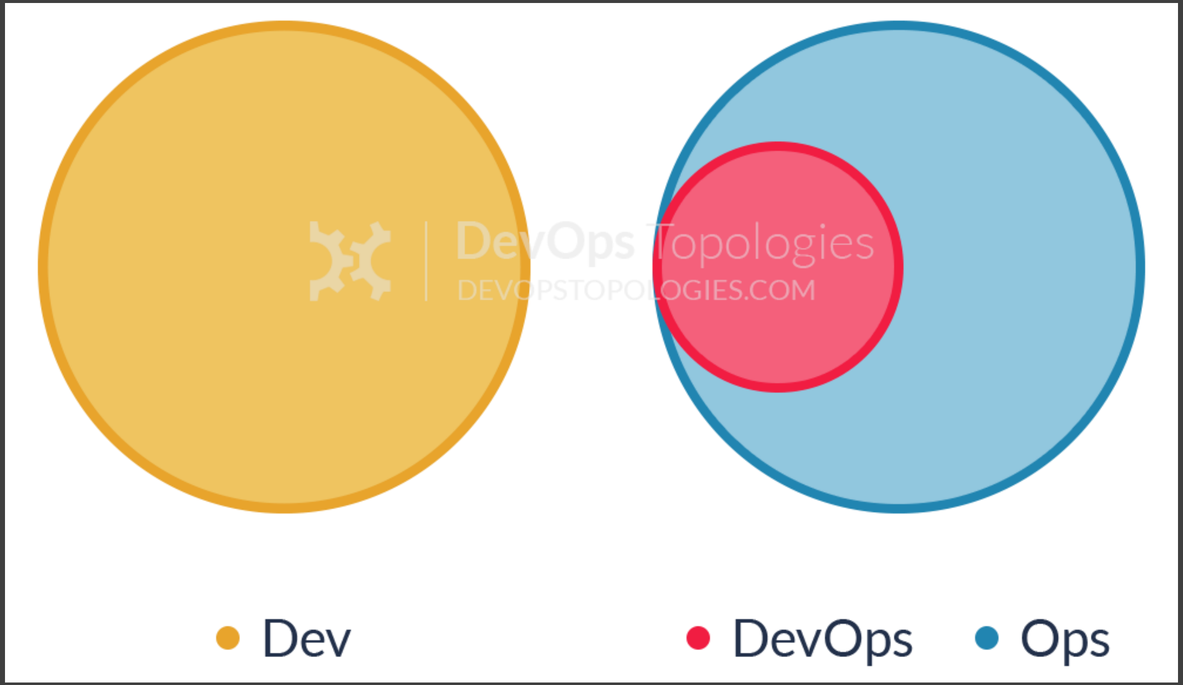
Anti-Type D: DevOps as Tools Team

- In order to "become DevOps" without losing current dev teams velocity.
- A DevOps team is set up to work on the tooling required for deployment pipelines, configuration management, environment management, etc.
- Meanwhile Ops folks continue to work in isolation and Dev teams continue to throw them applications "over the wall".



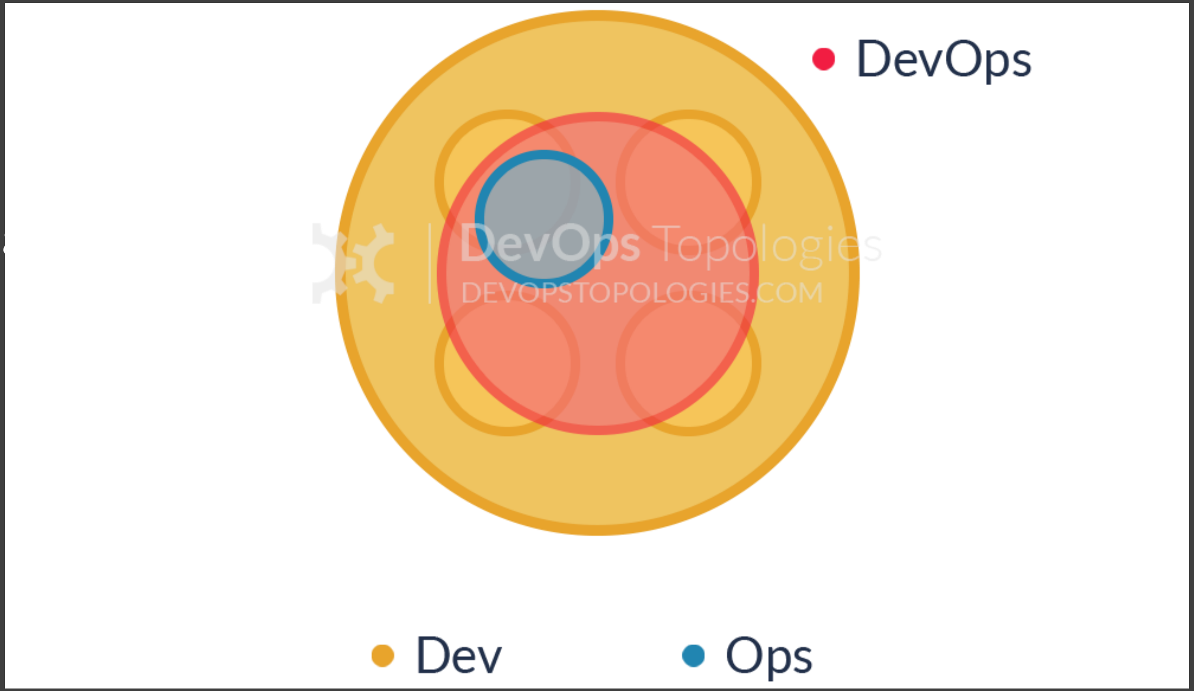
Anti-Type E: Rebranded SysAdmin

- This anti-type is typical in organizations with low engineering maturity.
- They want to improve their practices and reduce costs, yet they fail to see IT as a core driver of the business.
- Because industry successes with DevOps are now evident, they want to "do DevOps" as well.
- Unfortunately, instead of reflecting on the gaps in the current structure and relationships, they take the elusive path of hiring "DevOps engineers" for their Ops team(s).



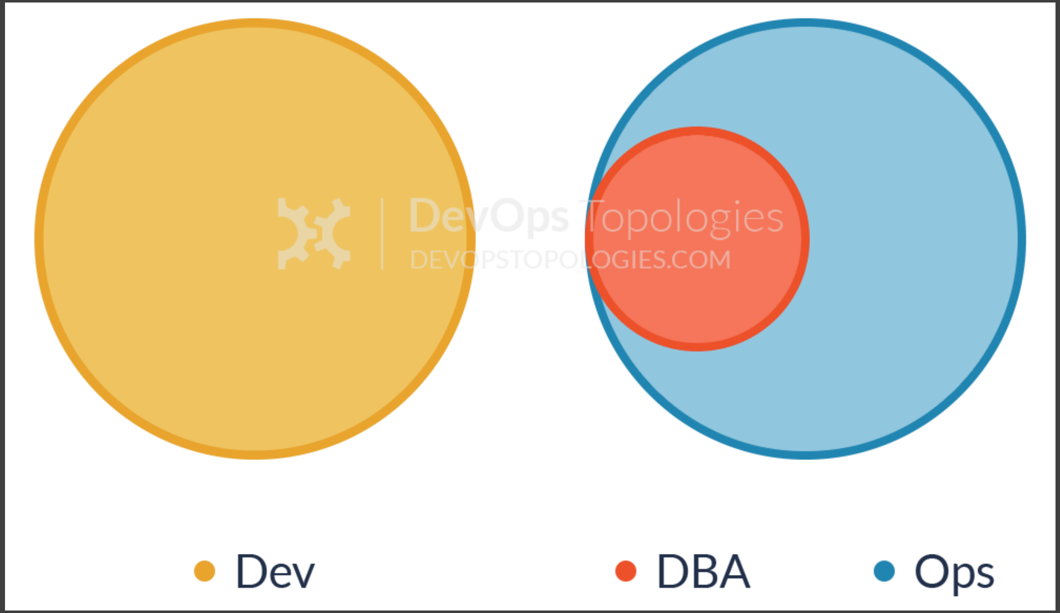
Anti-Type F: Ops Embedded in Dev Team

- The organization does not want to keep a separate Ops team, so development teams take responsibility for infrastructure, managing environments, monitorin etc.
- However, doing so in a project or product-driven way means those items are subject to resource constraints and re-prioritizations which lead to subpar approaches and half-baked solutions.
- In this anti-type the organization shows lack of appreciation for the importance and skills required for effective IT operations.



Anti-Type G: Dev and DBA Silos

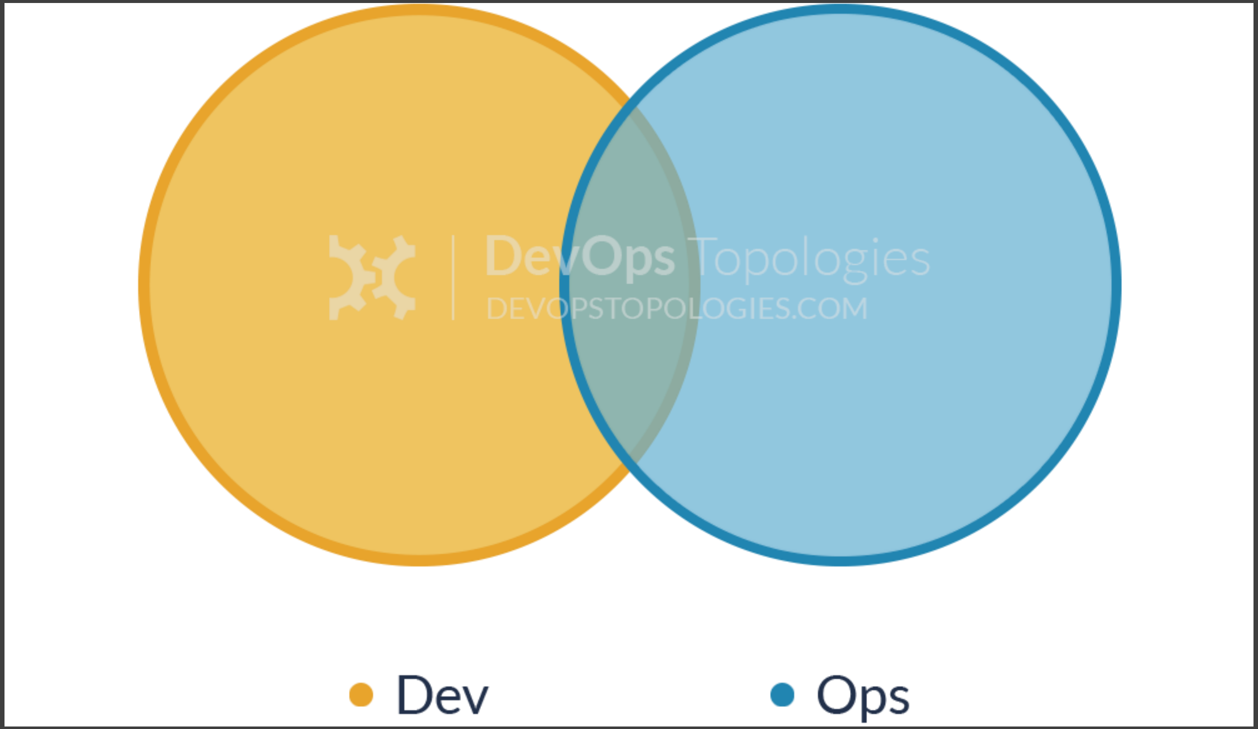
- This is a form of which is prominent in medium-to-large companies where multiple legacy systems depend on the same core set of data.
- Because these databases are so vital for the business, a dedicated DBA team, often under the Ops umbrella, is responsible for their maintenance, performance tuning and disaster recovery. That is understandable.
- The problem is when this team becomes a gate keeper for any and every database change, effectively becoming an obstacle to small and frequent deployments (a core tenet of DevOps and Continuous Delivery).
- the DBA team is not involved early in the application development, thus data problems (migrations, performance, etc) are found late in the delivery cycle. Coupled with the overload of supporting multiple applications databases, the end result is constant firefighting and mounting pressure to deliver.



DevOps Team Structures

Type 1: Dev and Ops Collaboration

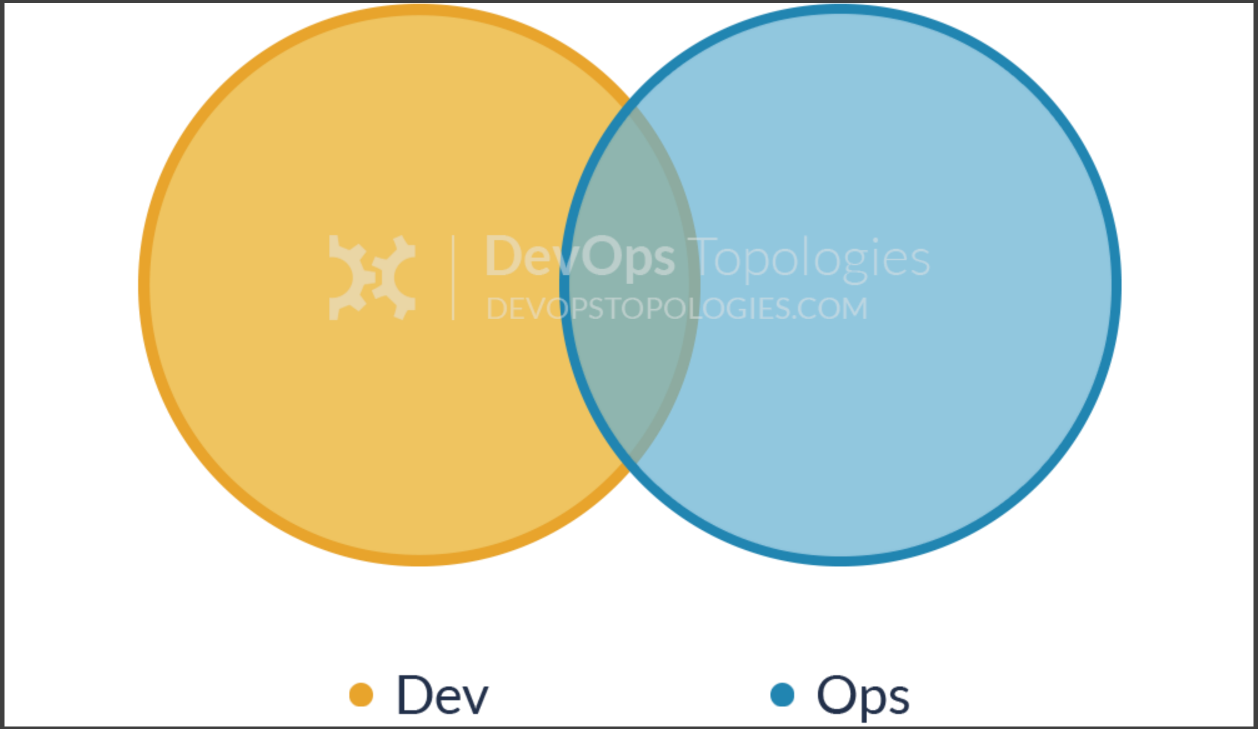
- This Model is implemented in an organisation with strong technical leadership.
- Smooth collaboration between Dev teams and Ops teams, each specialising where needed, but also sharing where needed.
- There are likely many separate Dev teams, each working on a separate or semi-separate product stack.



Potential effectiveness: HIGH

Type 1: Dev and Ops Collaboration

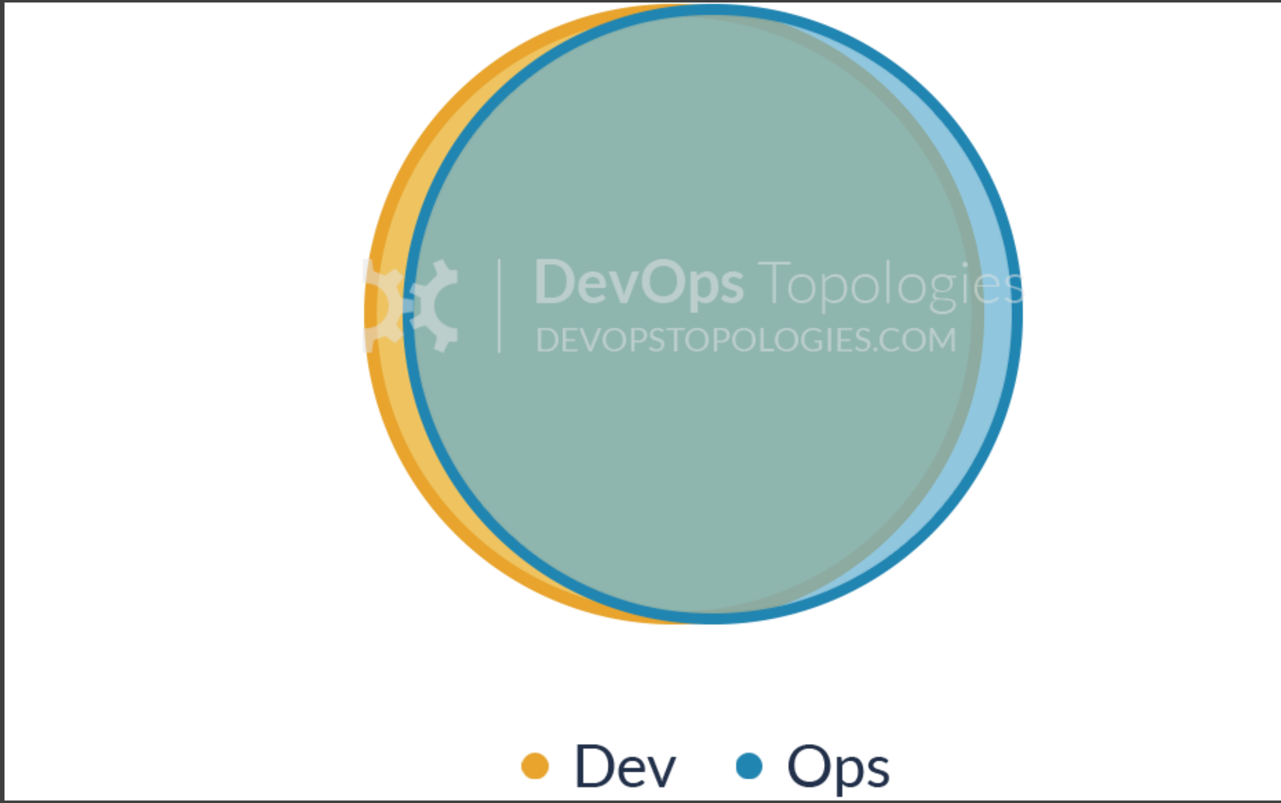
- Model needs quite substantial organisational change to establish it, and a good degree of competence higher up in the technical management team.
- Dev and Ops must have a clearly expressed and demonstrably effective shared goal.
- Ops folks must be comfortable pairing with Devs and get to grips with test-driven coding and Git,
- Devs must take operational features seriously and seek out Ops people for input into logging implementations, and so on, all of which needs quite a culture change from the recent past.



Potential effectiveness: **HIGH**

Type 2: Fully Shared Ops Reponsibilities

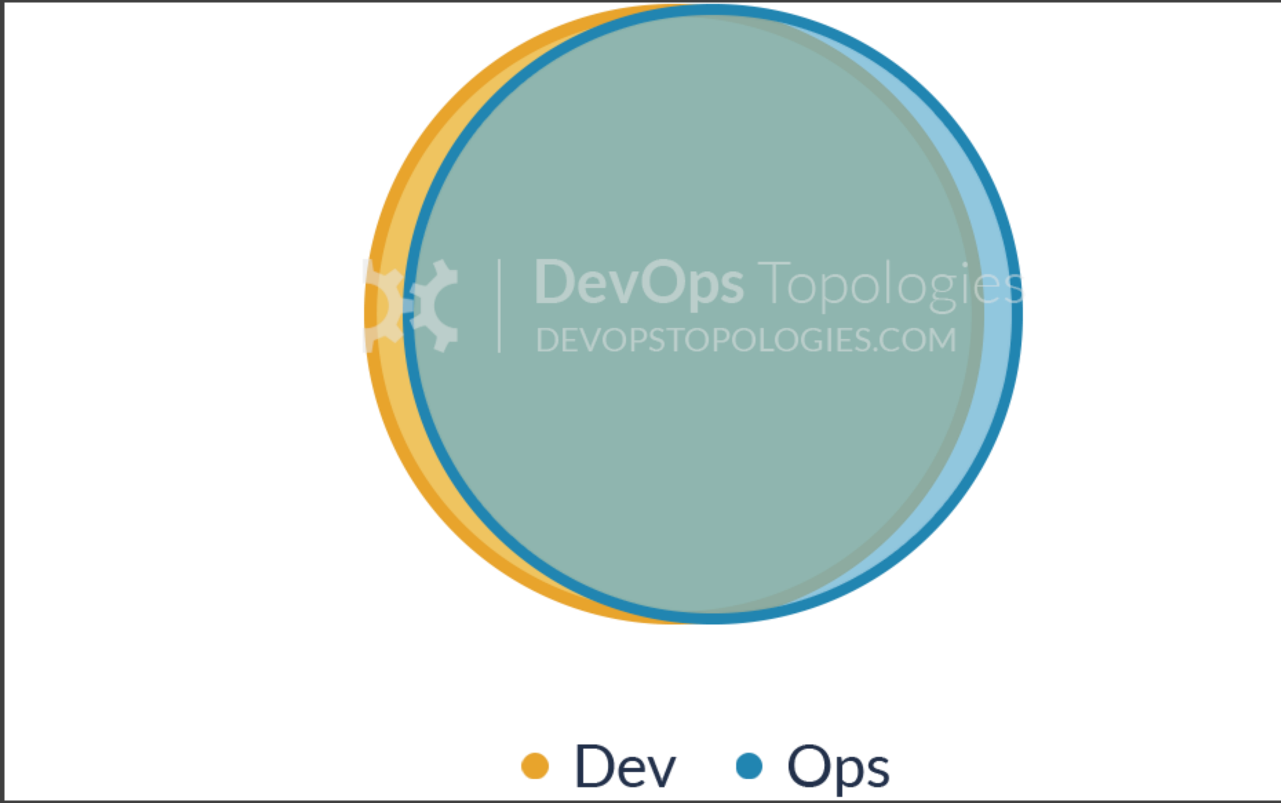
- Operations people have been integrated in product development teams
- There is so little separation between Dev and Ops that all people are highly focused on a shared purpose
- Organisations with a single main web-based product or service.



Potential effectiveness: **HIGH**

Type 2: Fully Shared Ops Reponsibilities

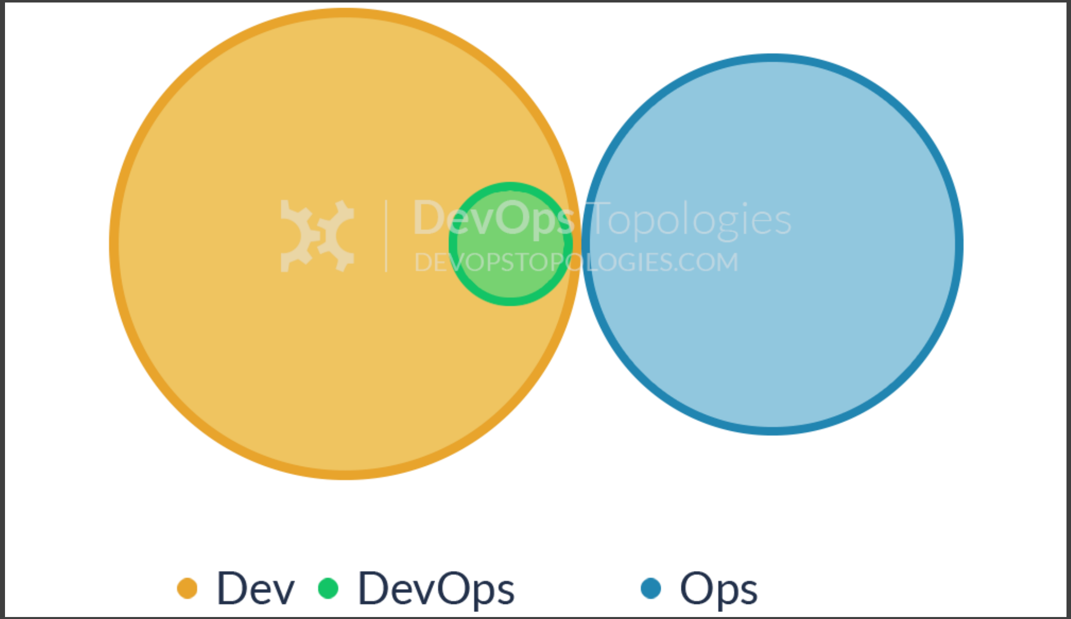
- Organisations such as Netflix and Facebook with effectively a single web-based product have achieved this Type 2 topology
- This topology might also be called ‘NoOps‘, as there is no distinct or visible Operations team



Potential effectiveness: **HIGH**

Type 3: Ops as Infrastructure-as-a-Service

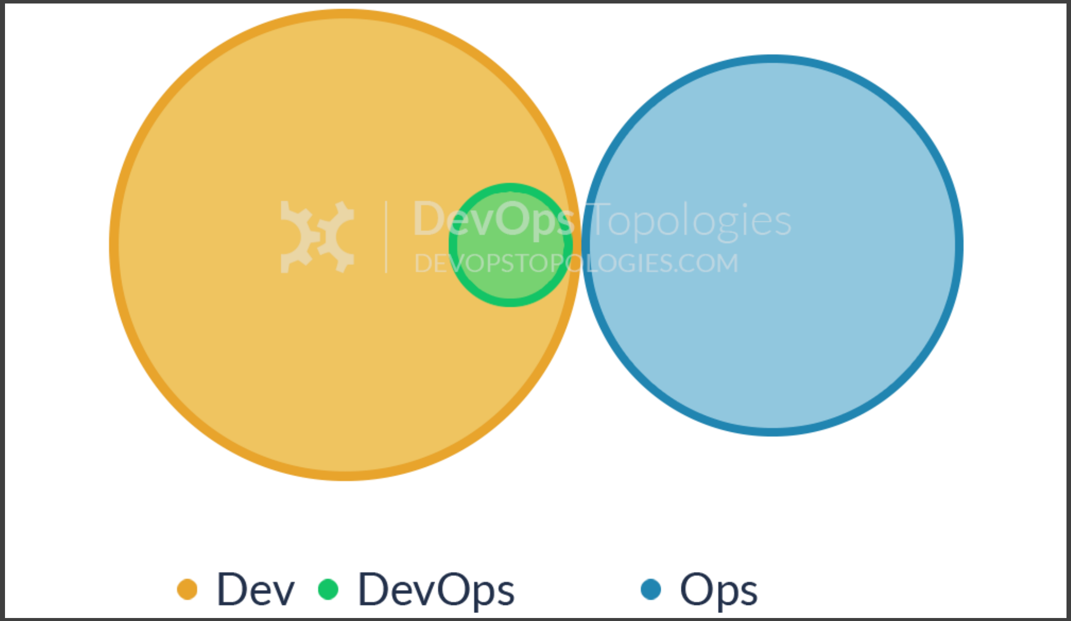
- For organisations with a fairly traditional IT Operations department which cannot or will not change rapidly , and
- For organisations who run all their applications in the public cloud (Amazon EC2, Rackspace, Azure, etc.),
- probably helps to treat Operations as a team who simply provides the elastic infrastructure on which applications are deployed and run;
- the internal Ops team is thus directly equivalent to Amazon EC2, or Infrastructure-as-a-Service.



Potential effectiveness: **MEDIUM**

Type 3: Ops as Infrastructure-as-a-Service

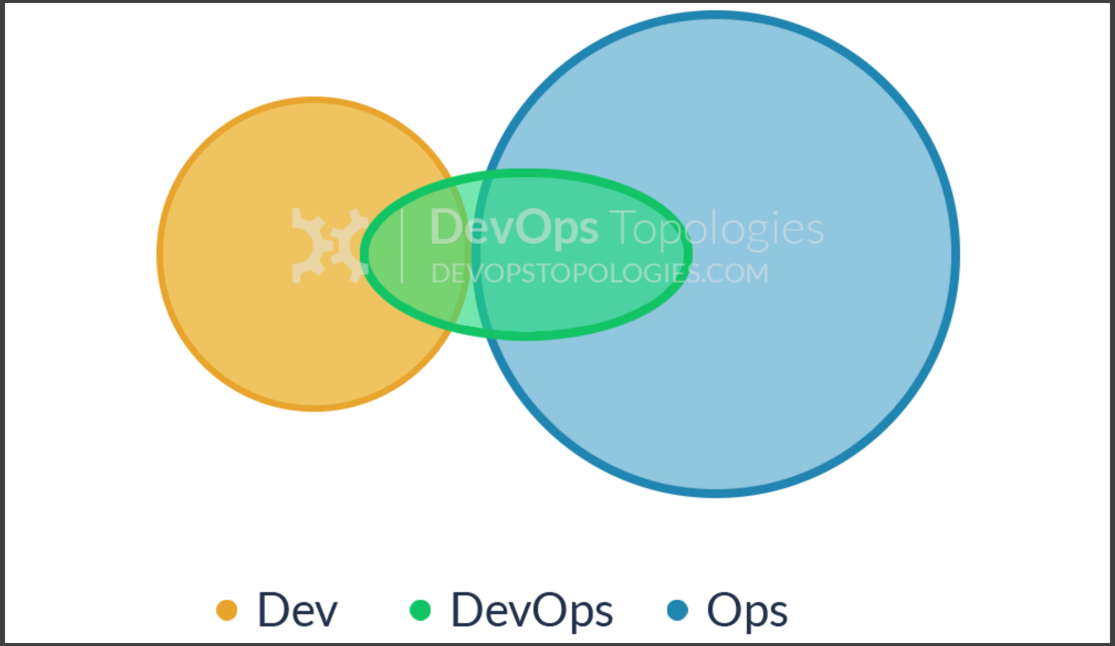
- A team within Dev then acts as a source of expertise about operational features, metrics, monitoring, server provisioning, etc., and
- Does most of the communication with the IaaS team.
- This team is still a Dev team, however, following standard practices like CI, iterative development, coaching, etc.
- Organisations with several different products and services, with a traditional Ops department, or whose applications run entirely in the public cloud.



Potential effectiveness: MEDIUM

Type 4: DevOps as an External Service

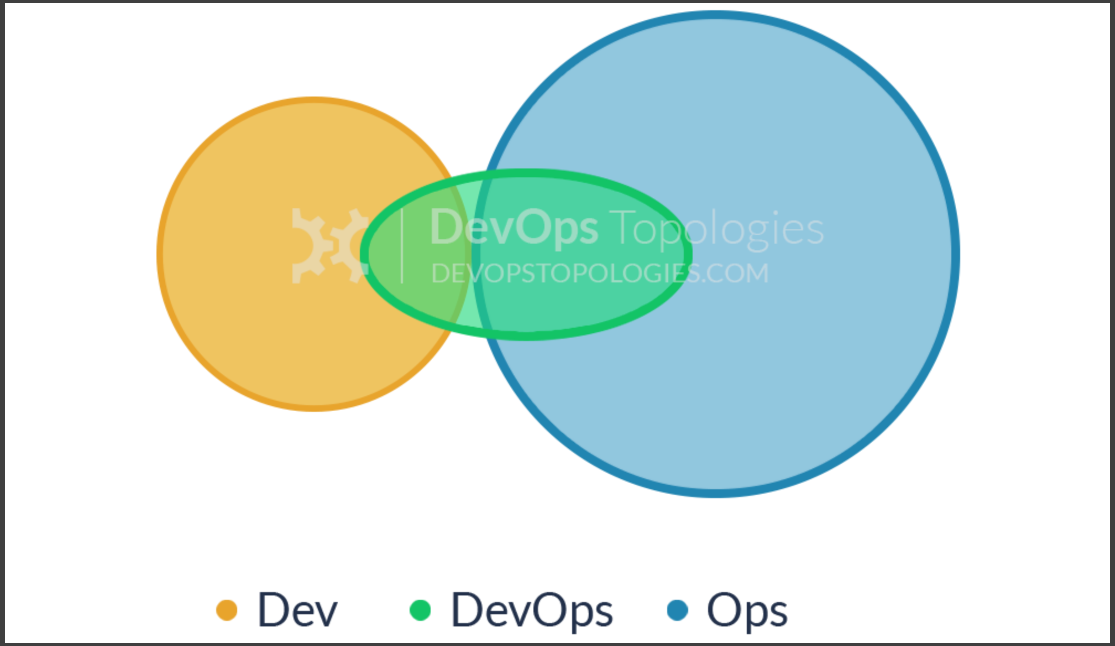
- Some organisations, particularly smaller ones, might not have the finances, experience, or staff to take a lead on the operational aspects of the software they produce.
- The Dev team might then reach out to a service provider like Rackspace to help them build test environments and automate their infrastructure and monitoring, and advise them on the kinds of operational features to implement during the software development cycles.



Potential effectiveness: MEDIUM

Type 4: DevOps as an External Service

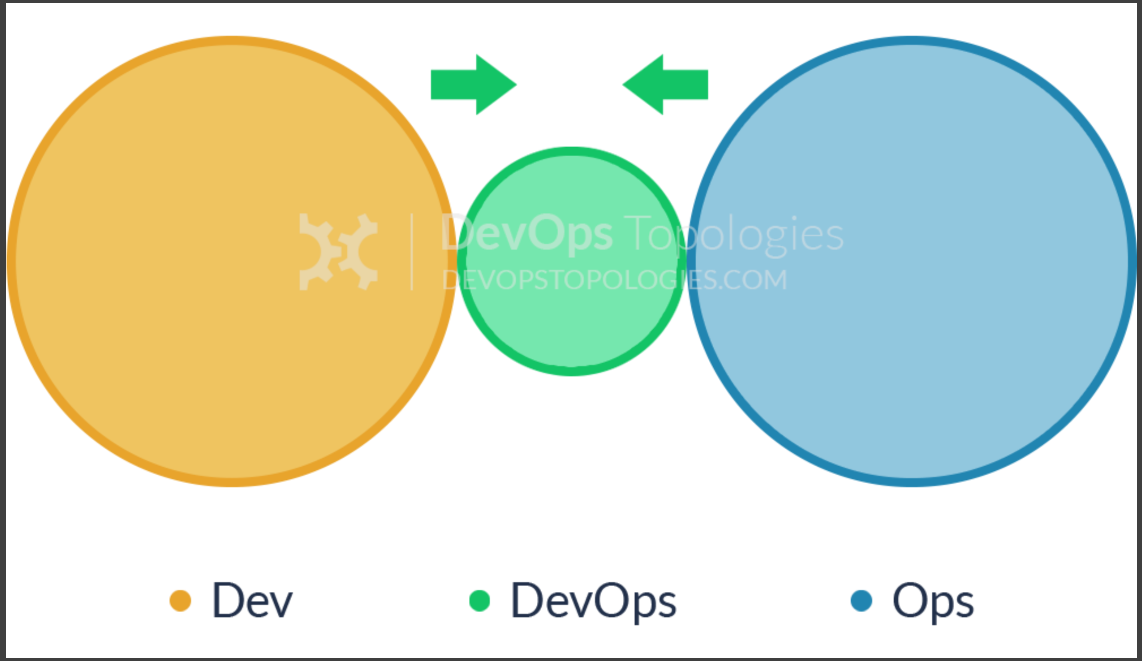
- What might be called DevOps-as-a-Service could be a useful and pragmatic way for a small organisation or team to learn about automation, monitoring, and configuration management, and then perhaps move towards a **Type 3 (Ops as IaaS)** or even **Type 1 (Dev and Ops Collaboration)** model as they grow and take on more staff with operational focus.
- Suitable for smaller teams or organisations with limited experience of operational issues.



Potential effectiveness: **MEDIUM**

Type 5: DevOps Team with an Expiry Date

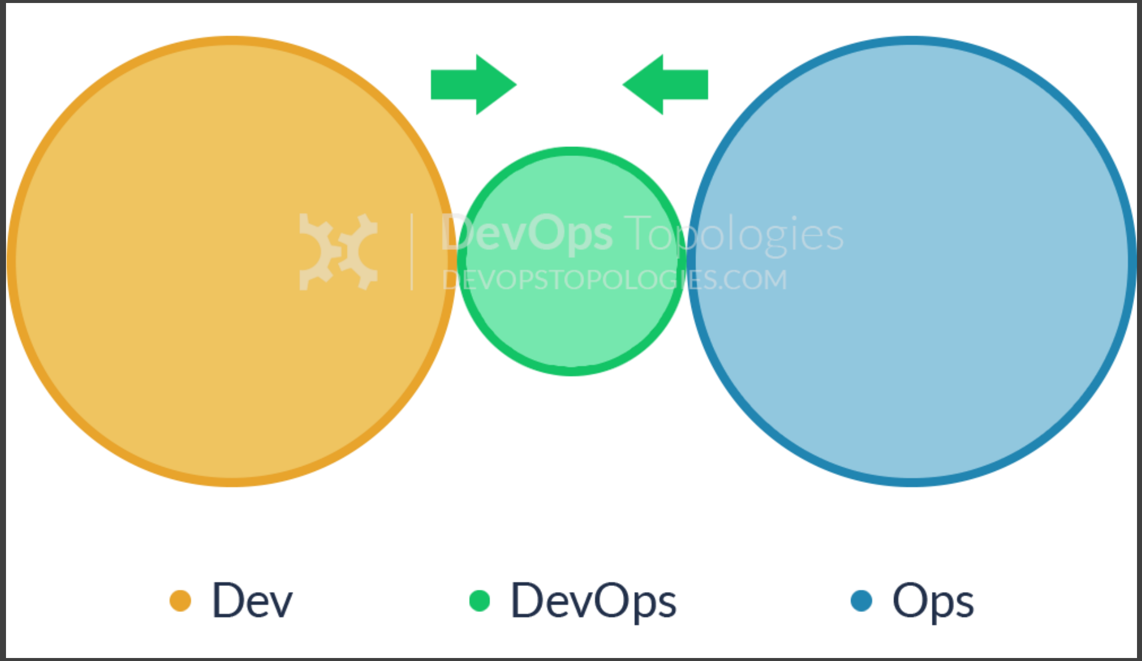
- The DevOps Team with an Expiry Date
- This temporary team has a mission to bring Dev and Ops closer together, ideally towards a **Type 1 (Dev and Ops Collaboration)** or **Type 2 (Fully Shared Ops Reponsibilities)** model, and eventually make itself obsolete.



Potential effectiveness: **MEDIUM**

Type 5: DevOps Team with an Expiry Date

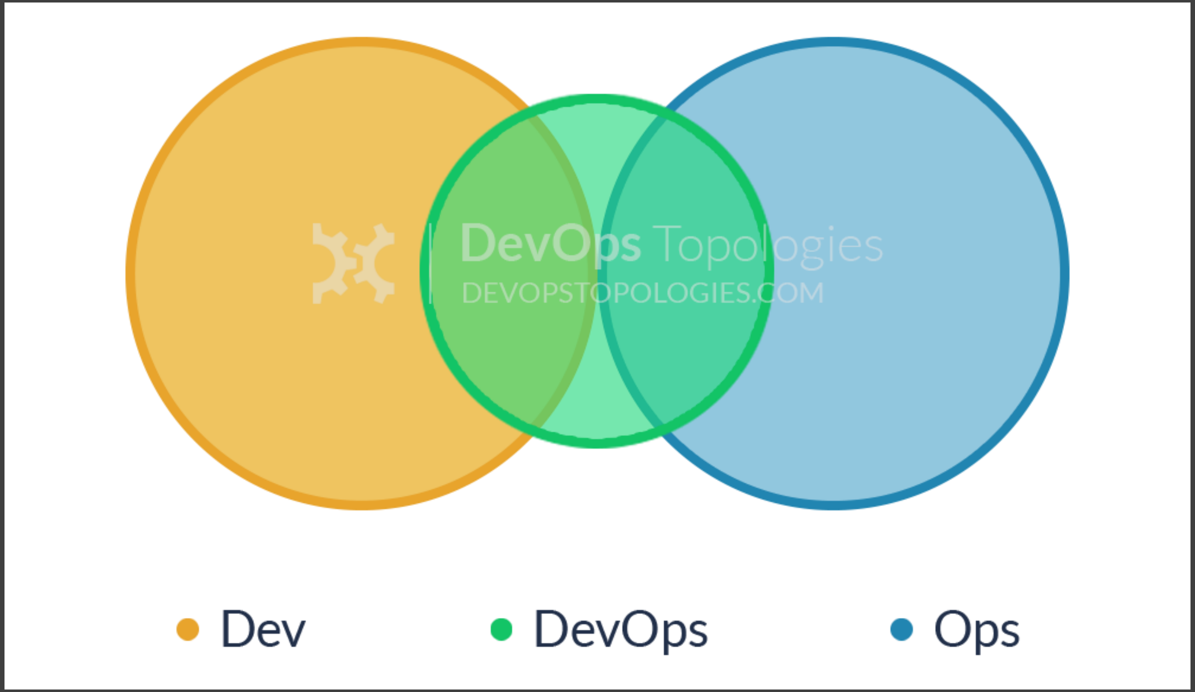
- The members of the temporary team will 'translate' between Dev-speak and Ops-speak, introducing crazy ideas like stand-ups and Kanban for Ops teams, and thinking about dirty details like load-balancers, management NICs, and SSL offloading for Dev teams.
- If enough people start to see the value of bringing Dev and Ops together, then the temporary team has a real chance of achieving its aim;
- Crucially, long-term responsibility for deployments and production diagnostics should not be given to the temporary team, otherwise it is likely to become a **DevOps Team Silo (Anti-Type B).**



Potential effectiveness: **Low to HIGH**

Type 6: DevOps Evangelists Team

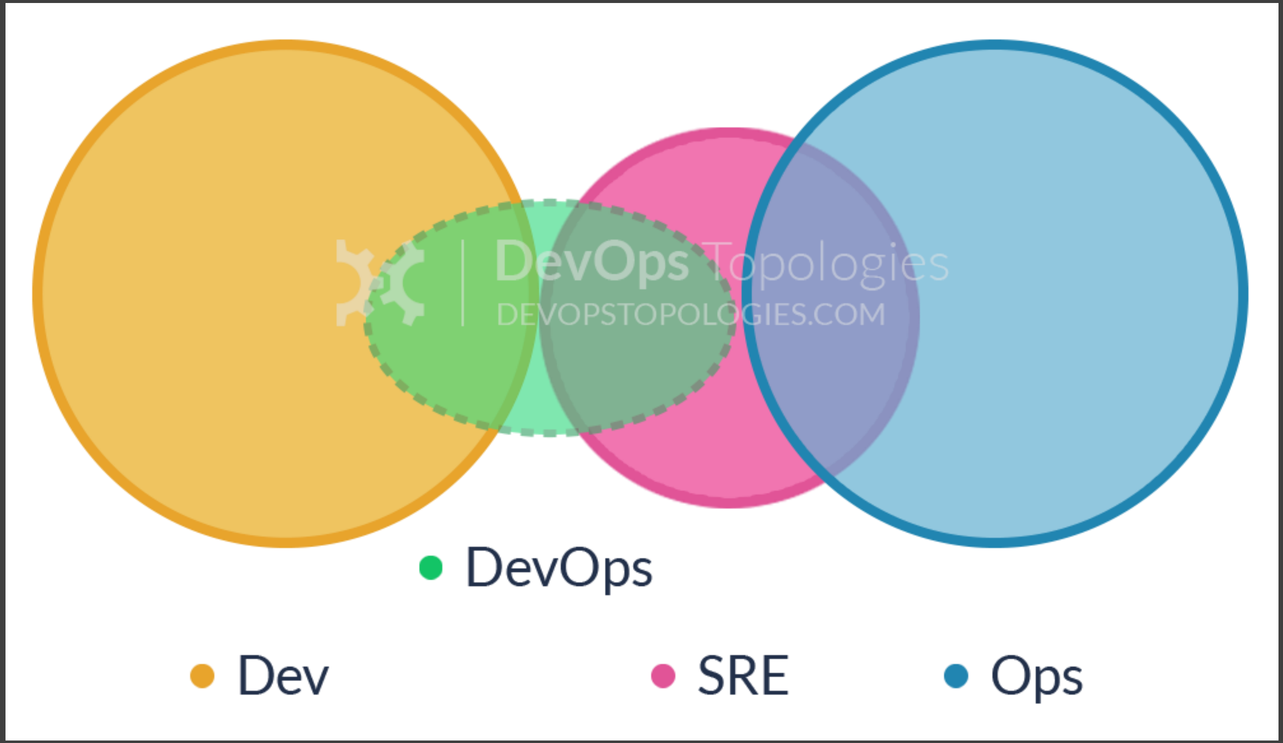
- Within organisations that have a large gap between Dev and Ops,
- it can be effective to have a 'facilitating' DevOps team that keeps the Dev and Ops sides talking.
- This is a version of **Type 5 (DevOps Team with an Expiry Date)** but where the DevOps team exists on an ongoing basis with the specific remit of facilitating collaboration and cooperation between Dev and Ops teams.
- Members of this team are sometimes called 'DevOps Evangelists', because they help to spread awareness of DevOps practices.



Potential effectiveness: **MEDIUM to HIGH**

Type 7: SRE Team (Google Model)

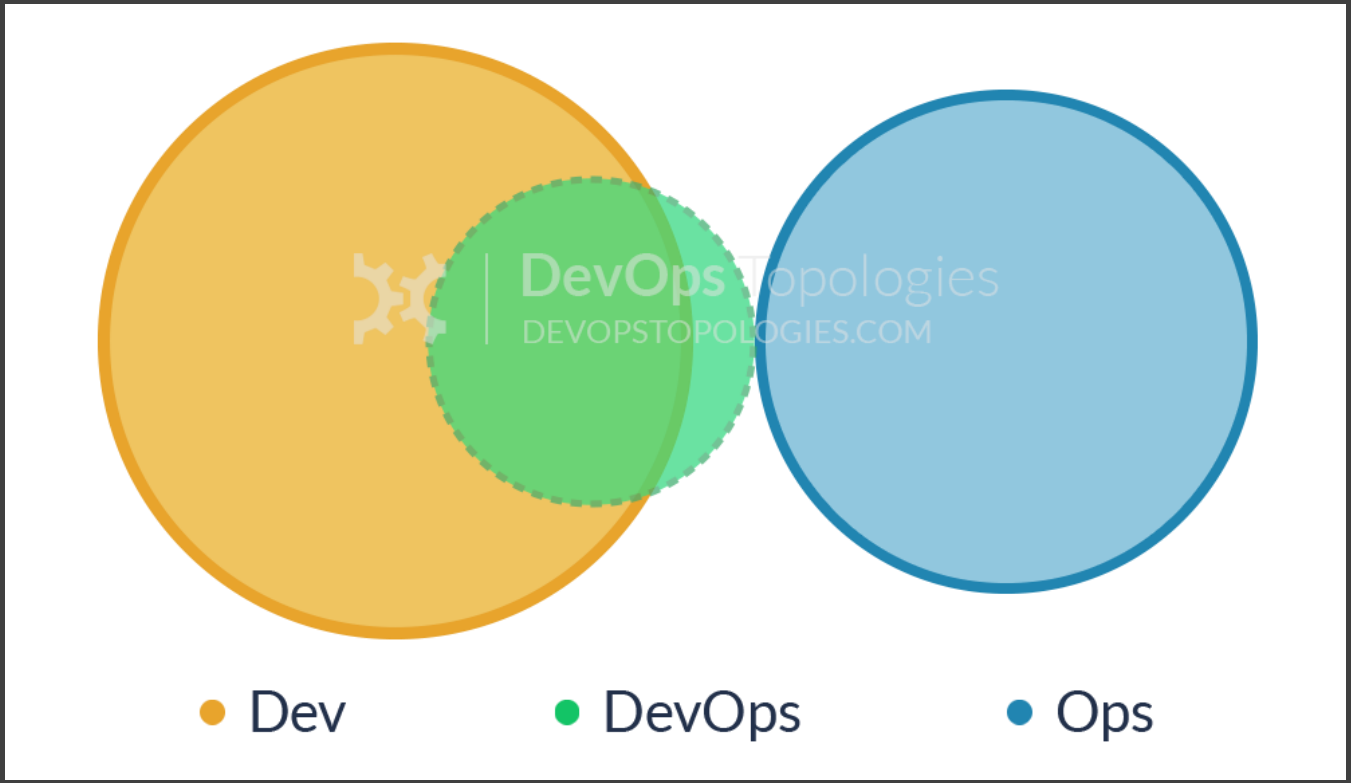
- DevOps often recommends that Dev teams join the on-call rotation, but it's not essential. In fact, some organisations (including Google) run a different model, with an explicit 'hand-off' from Development to the team that runs the software, the Site Reliability Engineering (SRE) team.
- In this model, the Dev teams need to provide test evidence (logs, metrics, etc.) to the SRE team showing that their software is of a good enough standard to be supported by the SRE team.



Potential effectiveness: **LOW to HIGH**

Type 8: Container-Driven Collaboration

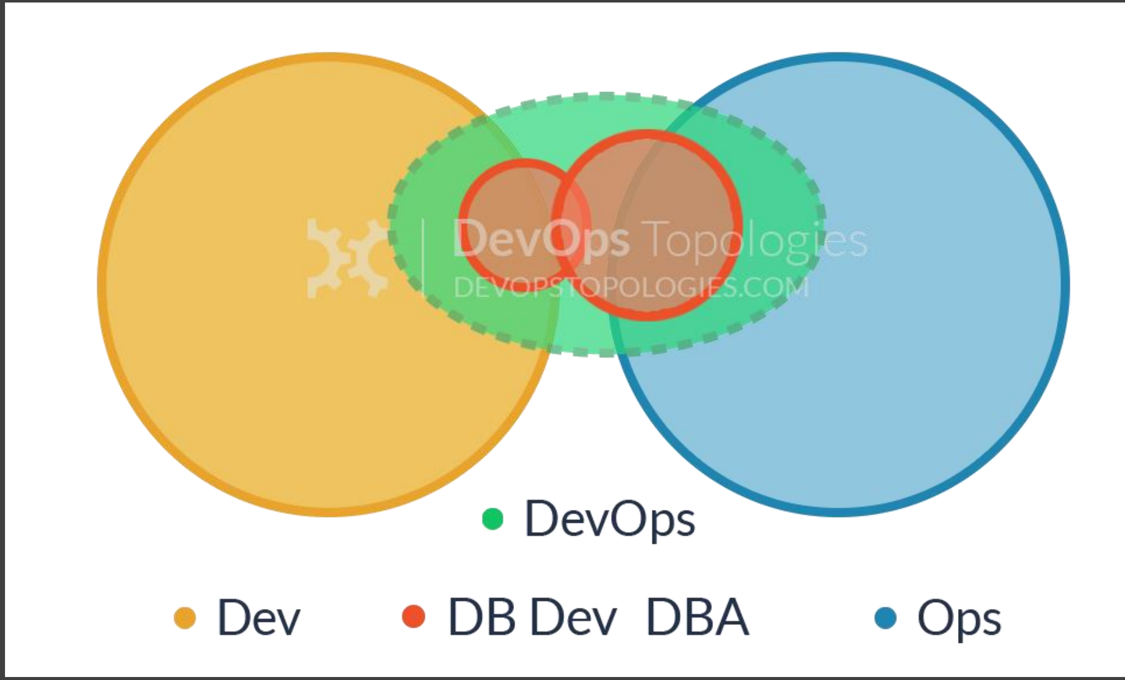
- Containers remove the need for some kinds of collaboration between Dev and Ops by encapsulating the deployment and runtime requirements of an app into a container. In this way, the container acts as a boundary on the responsibilities of both Dev and Ops.
- With a sound engineering culture, the Container-Driven Collaboration model works well, but if Dev starts to ignore operational considerations this model can revert towards to an adversarial 'us and them'.



Potential effectiveness: MEDIUM to HIGH

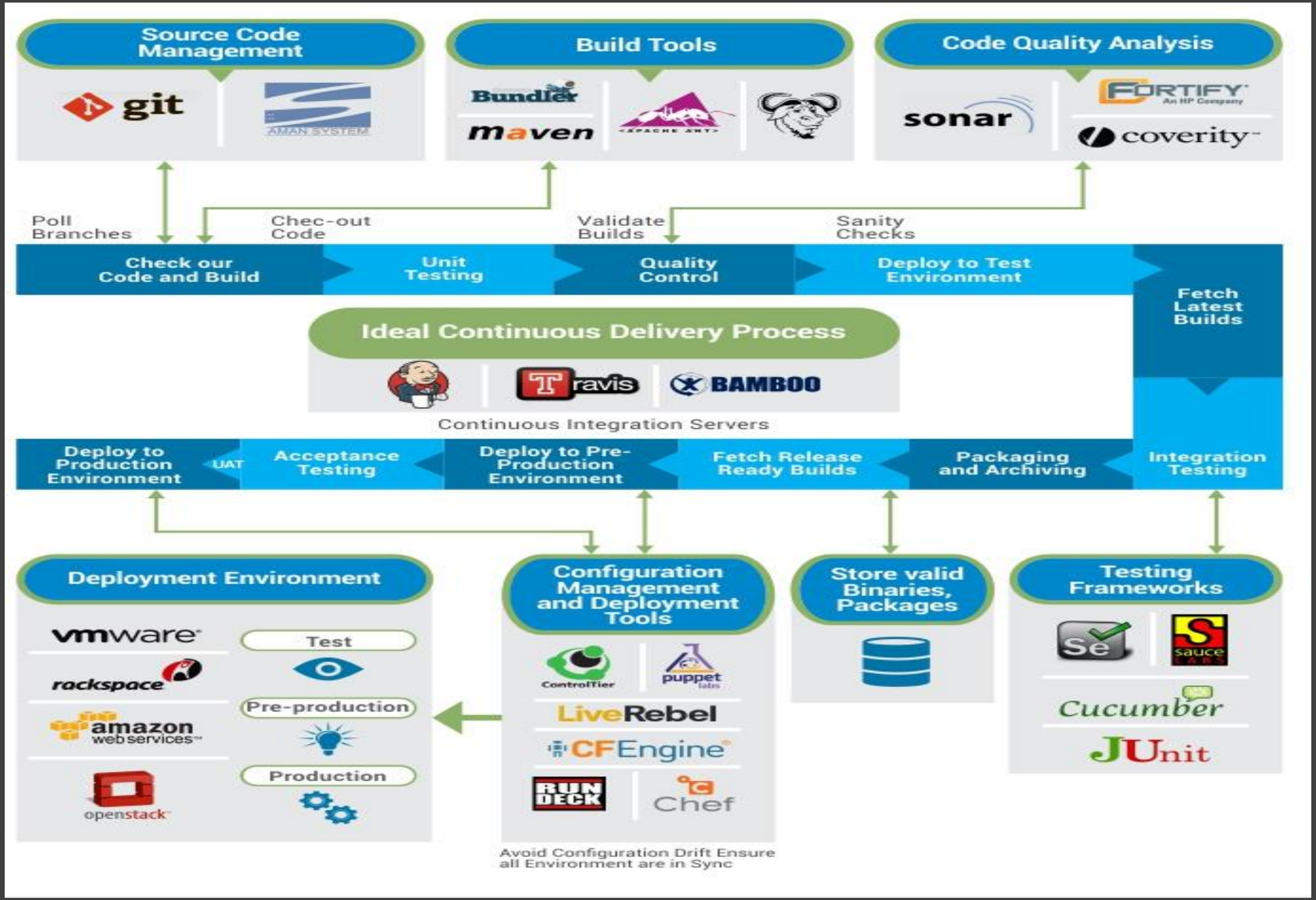
Type 9: Dev and DBA Collaboration

- In order to bridge the Dev-DBA chasm, some organisations have experimented with something like Type 9, where a database capability from the DBA team is complimented with a database capability (or specialism) from the Dev team.
- This seems to help to translate between the Dev-centric view of databases (as essentially dumb persistence stores for apps) and the DBA-centric view of databases (smart, rich sources of business value).

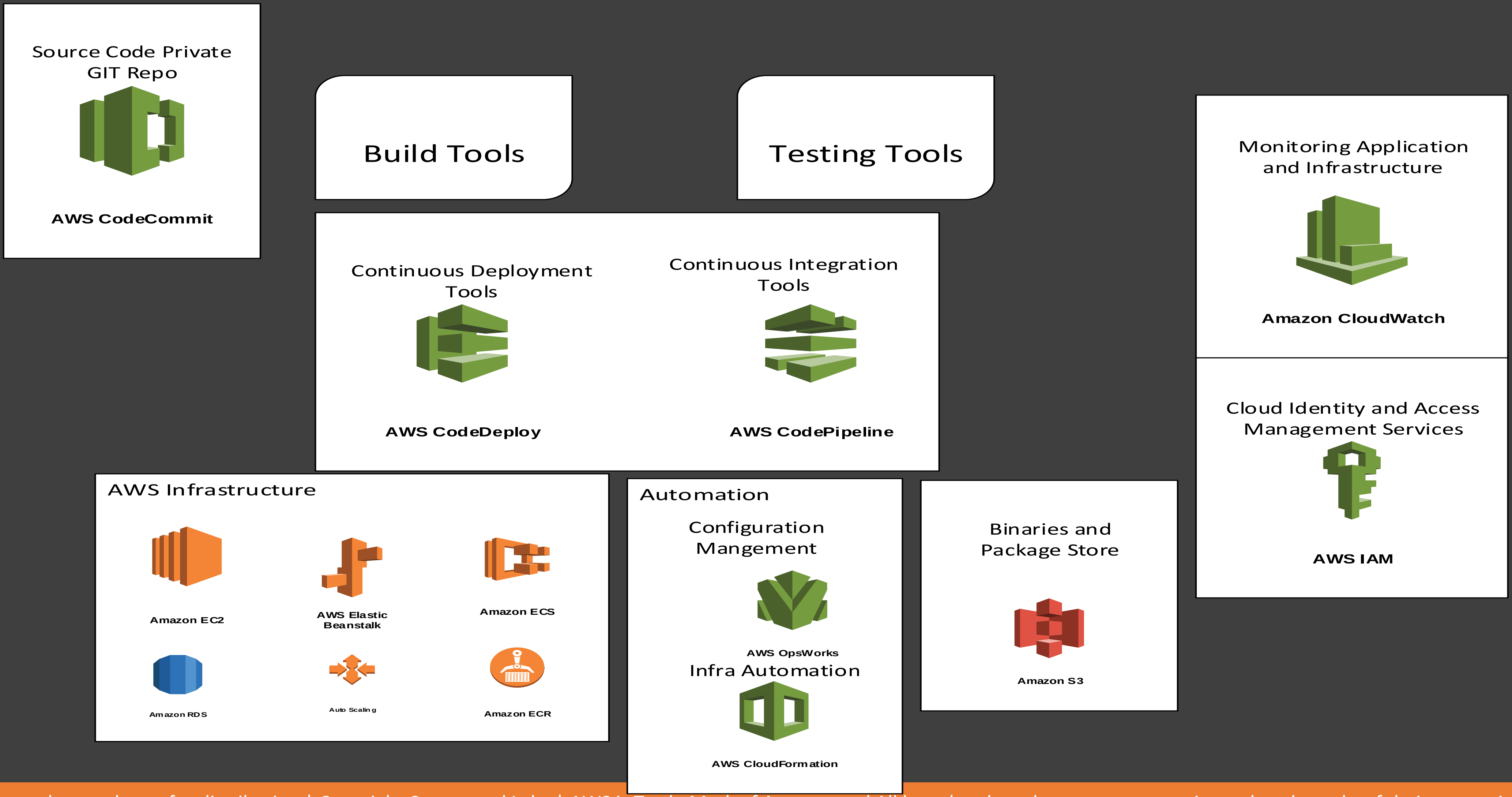


Potential effectiveness: MEDIUM

Dev Ops Open source Reference Architecture



AWS Dev Ops Reference Architecture



Infrastructure as a Service

- 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.
- 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.
- 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.

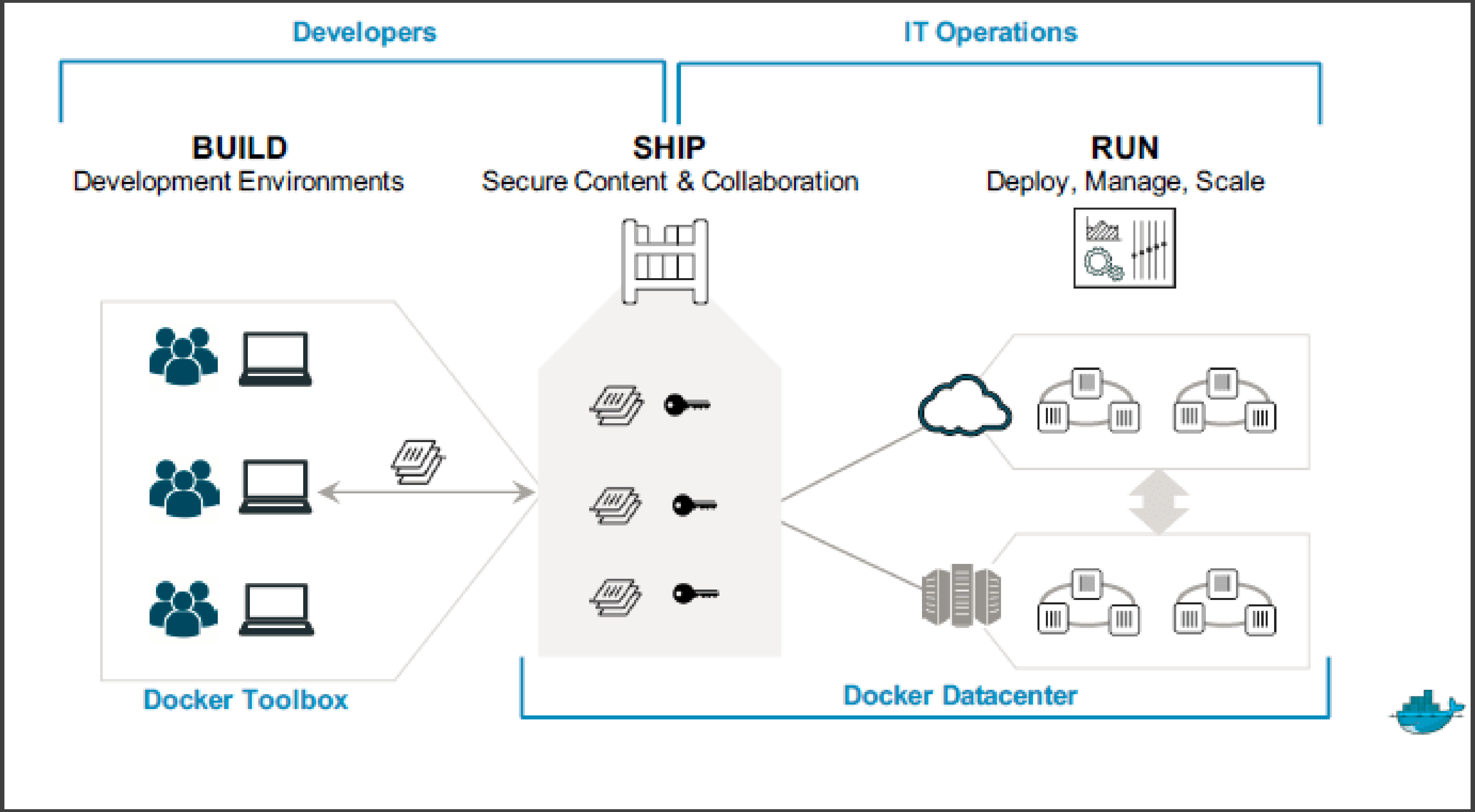
Infrastructure as a Service

- Public Cloud Providers
  - Amazon Web Services
  - Windows Azure
  - RackSpace Cloud
  - HP Cloud
  - OpenShift by Red Hat
  - Ubuntu Cloud
  - Heroku
  - EngineYard

Infrastructure as a Service

- Build your own private clouds
  - Openstack
  - VMWare
  - Microsoft Private Cloud

Container as a Service



Container as a Service

- CAAS Solutions
  - Docker
  - Swarm and Compose
  - Apache Mesos, Mesos DCOS
  - CoreOS and etcd
  - Kubernetes

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.

Configuration Management Tools

- Chef
- Ops Works (Using AWS)
- Puppet
- Ansible
- Salt Stack
- Pallet
- Bcfg2

Source Code Management

- To achieve the benefits of DevOps, it's essential to Manage Source code not just your application code but your infrastructure, configurations, and databases.
- This requires scripting all of your source artifacts, but the payoff should be a single source of truth for both your application code and your IT systems and databases, allowing you to quickly identify where things went wrong, and recreate known states with the push of a button.
- No more having to play Sherlock Holmes to figure out which versions of your application code goes with which environments or databases.
- While commonly used version-control tools include Git, Perforce, and Subversion, they differ widely on how well they support DevOps-style collaboration.

Continuous Integration

- Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day.
- Each check-in is then verified by an automated build, allowing teams to detect problems early.
- By integrating regularly, you can detect errors quickly, and locate them more easily.
- Continuous Integration emerged in the Extreme Programming (XP) community, and XP advocates Martin Fowler and Kent Beck first wrote about continuous integration circa 1999.

Continuous Integration

- *"Continuous Integration doesn't get rid of bugs, but it does make them dramatically easier to find and remove."*
    - Martin Fowler
- Because you're integrating so frequently, there is significantly less back-tracking to discover where things went wrong, so you can spend more time building features.
- Continuous Integration is cheap. Not continuously integrating is costly.
- If you don't follow a continuous approach, you'll have longer periods between integrations. This makes it exponentially more difficult to find and fix problems. Such integration problems can easily knock a project off-schedule, or cause it to fail altogether.

Continuous Integration benefits

- Say goodbye to long and tense integrations
- Increase visibility which enables greater communication
- Catch issues fast and nip them in the bud
- Spend less time debugging and more time adding features
- Proceed in the confidence you're building on a solid foundation
- Stop waiting to find out if your code's going to work
- Reduce integration problems allowing you to deliver software more rapidly

CI Practices

- Maintain a single source repository
  - Check in all the code in a Single Source repository
  - You can use any source code repository like GIT, SVN, PERFORCE, Clearcase etc
  - Check in Everything required for a build
    - test scripts,
    - properties files,
    - database schema,
    - install scripts, and
    - third party libraries.

CI Practices

- Automate the Build
- Make Your Build Self-Testing
- Everyone Commits To the Mainline Every Day
- Every Commit Should Build the Mainline on an Integration Machine
- Fix Broken Builds Immediately
- Keep the Build Fast
- Test in a Clone of the Production Environment
- Make it Easy for Anyone to Get the Latest Executable
- Everyone can see what's happening
- Automate Deployment

CI Process

- Developers check out code into their private workspaces.
- When done, commit the changes to the repository.
- The CI server monitors the repository and checks out changes when they occur.
- The CI server builds the system and runs unit and integration tests.
- The CI server releases deployable artefacts for testing.
- The CI server assigns a build label to the version of the code it just built.
- The CI server informs the team of the successful build.
- If the build or tests fail, the CI server alerts the team.
- The team fix the issue at the earliest opportunity.
- Continue to continually integrate and test throughout the project.

Teams Responsibility

- Check in frequently
- Don’t check in broken code
- Don’t check in untested code
- Don’t check in when the build is broken
- Don’t go home after checking in until the system builds

Continuous Deployment

- **Continuous delivery** (**CD**) is a software engineering approach in which teams produce software in short cycles, ensuring that the software can be reliably released at any time.
- It aims at building, testing, and releasing software faster and more frequently.
- The approach helps reduce the cost, time, and risk of delivering changes by allowing for more incremental updates to applications in production.
- A straightforward and repeatable deployment process is important for continuous delivery.
- Continuous Deployment is closely related to Continuous Integration and refers to the release into production of software that passes the automated tests.

Course and Labs

CI/CD Tools

- Jenkins
- Travis
- Bamboo
- AWS Code Deploy / Code Pipeline

Build

Build is the process of integrating, building and compiling the software that is produced.

For large software teams, there is generally a continuous integration process where software developers check-in their changes, and if anyone breaks the build, it needs to be addressed.

The integration process can take a lot of processing power, and several hours. This infrastructure needs to be monitored and optimized.

Commonly used Build tools

- Maven
- Bundle
- Ant
- Build master
- Quick Build

Quality Assurance

- **QA owns continuous improvement and quality tracking** across the entire development cycle. They are the ones who are primarily responsible for identifying problems not just in the product but also in the process, and recommending changes wherever they can.
- **Tests are code**, as any test automation expert will tell you. It's a necessity, of course. If your process is designed to publish a new release every day (or every hour) there is no room for manual testing. You must develop automation systems, through code, that can ensure quality standards are maintained.
- **Automation rules**. Anything that can be automated, should be automated. When Carl describes Unbounce's deployment process as "push-button easy," this is what he's talking about.
- **Testers are the quality advocates**, influencing both development and operational processes. They don't just find bugs. They look for any opportunity to improve repeatability and predictability.

QA Tools

- Code testing tools
  - Sonar
  - Fortify
  - Coverty
- Automated Testing tools
  - Selinium
  - Sause
  - Cucumber
  - Junit

Continuous Monitoring

- DevOps teams then face significant challenges in guaranteeing expected application behavior:
  - Understanding application performance before and after new code is pushed and pinpointing defects early, before they spread.
  - Diving back into the history of deployments, determining the impact on the infrastructure throughput and response time.
  - Forecasting infrastructure utilization bottlenecks due to changes into the code or variations in the workload.
  - Rather than waiting for production performance data to analyze what went wrong, the DevOps team is able to develop performance analytics models that can anticipate operational and quality problems before the delivery phase.

Continuous Monitoring Tools

- Infrastructure Monitoring
  - **Nagios, Zabbix & Sensu**
- IaaS Monitoring
  - AWS Cloud watch, Openstack Celiometer, Stack driver
- Application Performance Monitoring
  - New Relic and App Dynamics