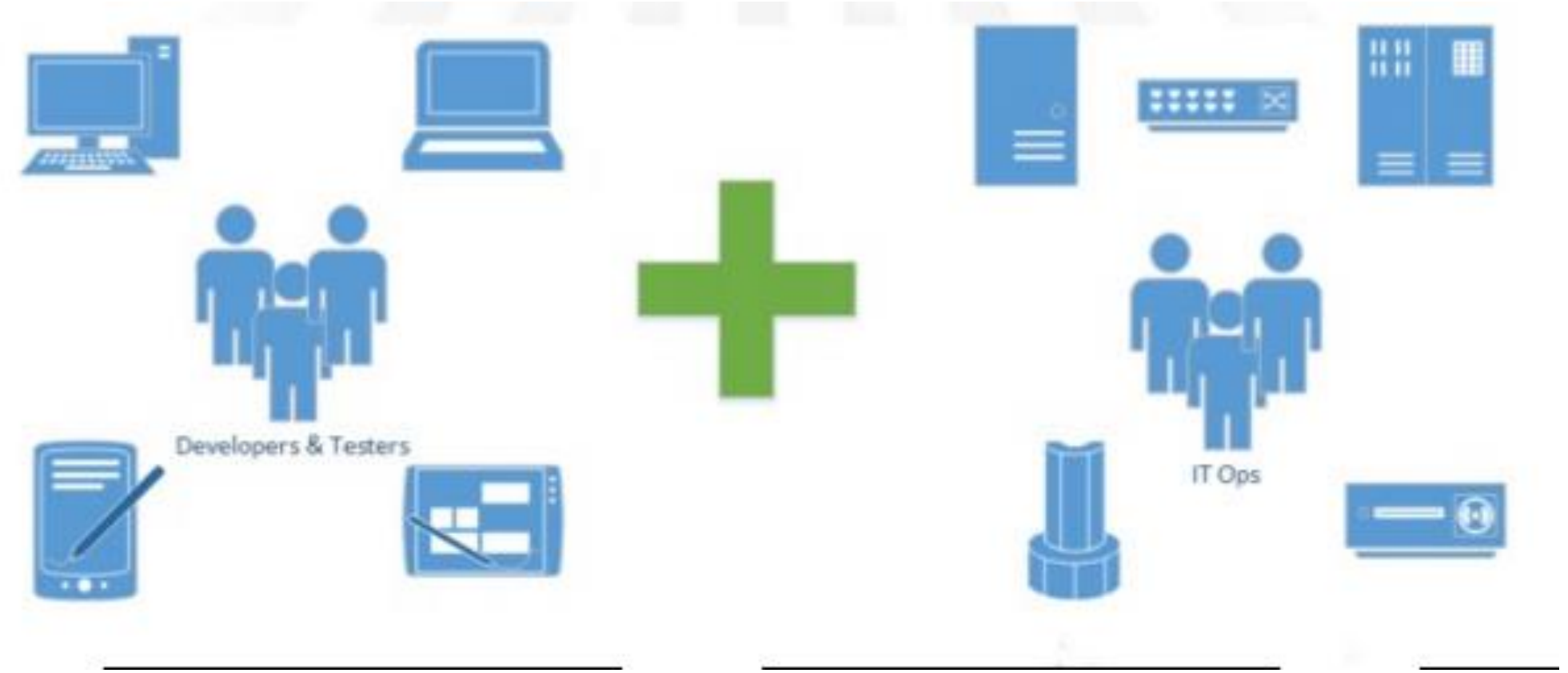
Introduction to DevOps

Mohanraj Shanmugam

What is DevOps



What is DevOps

- DevOps (a combination of development and operations) is a software development method that stresses on communication, collaboration and integration between software developers and information Technology(IT) professionals.
- Thereby– Enable rapid evolution of products or services–
  - Reduce risk,
  - Improve quality across portfolio, and
  - Reduce costs

What is DevOps

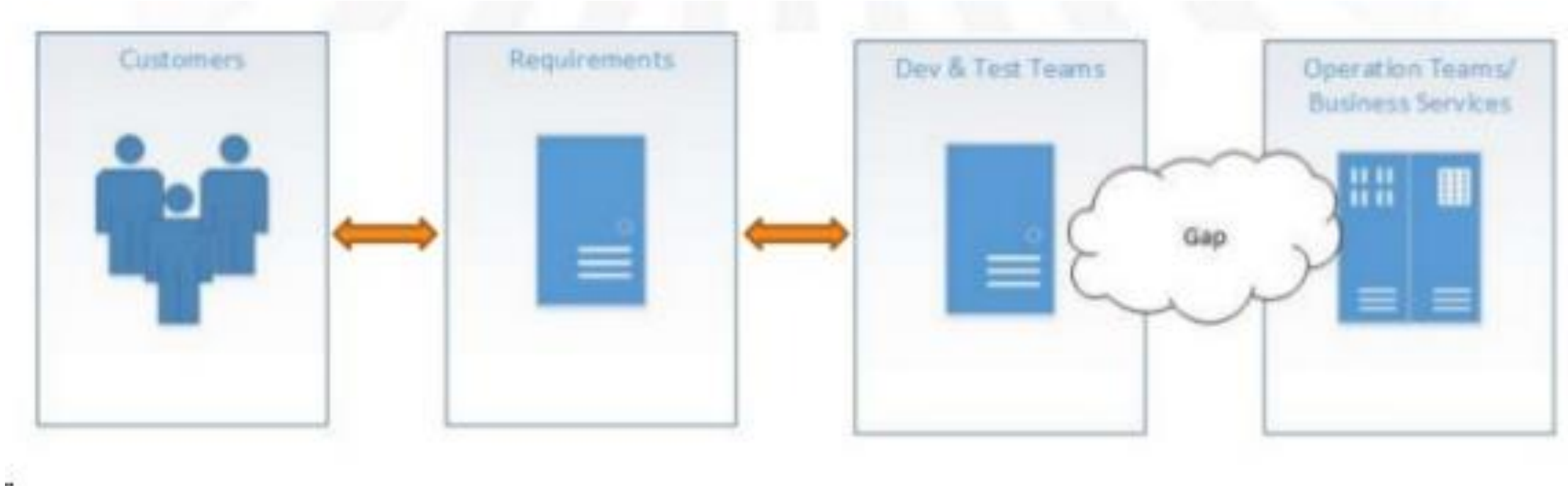
- DevOps integration targets
  - Product delivery,
  - Quality testing,
  - Feature development and
  - Maintenance releases

  in order to improve reliability and security and faster development and deployment cycles.
- 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

Principles of DevOps

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

Dev Ops Delivery Challenges



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

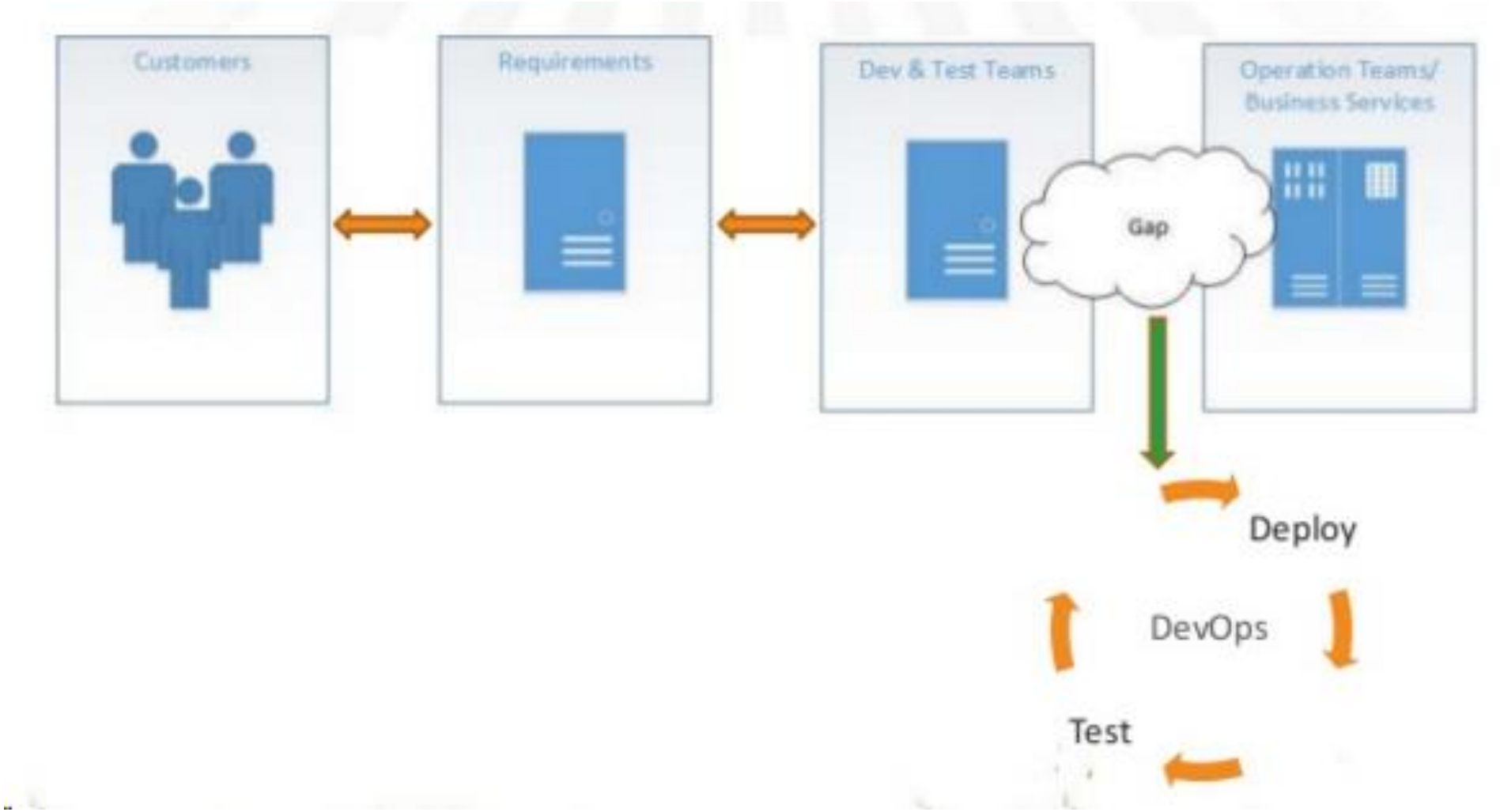
Dev and Ops

- 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 like PSR
- 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

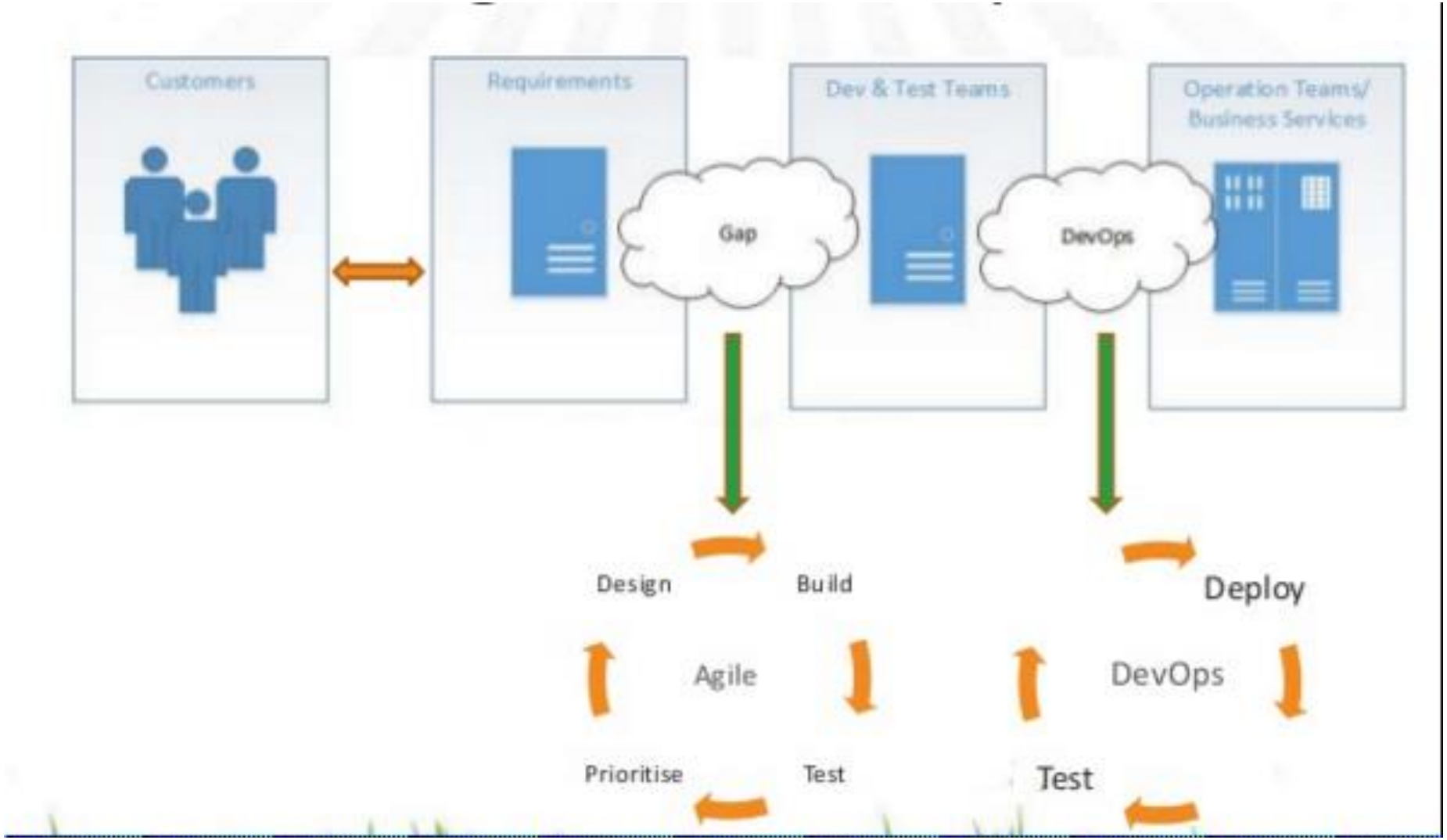
DevOps Delivery Challenges

- Top 3 delivery challenges
- Release management– Better understanding of risks, dependencies, compliance issues
- Release/Deployment coordination–
  - Better tracking of discrete activities, faster escalation of issues, documented process control and granular reporting
- Release/Deployment Automation–
  - Usually have existing automation but want to flexibly manage and drive this automation that can be invoked by non-operations resources in specific non-production environments

DevOps Delivery Challenges



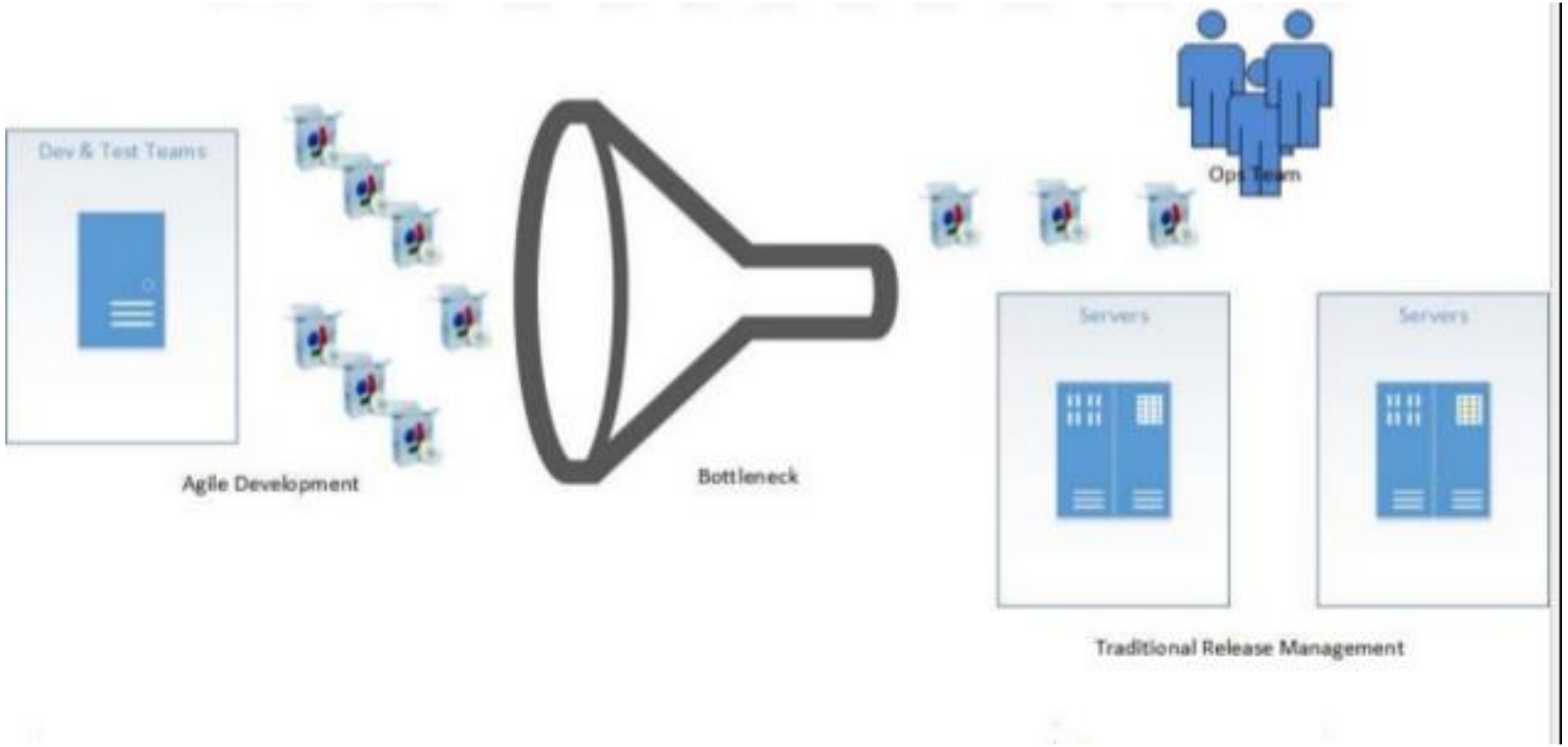
Agile and DevOps



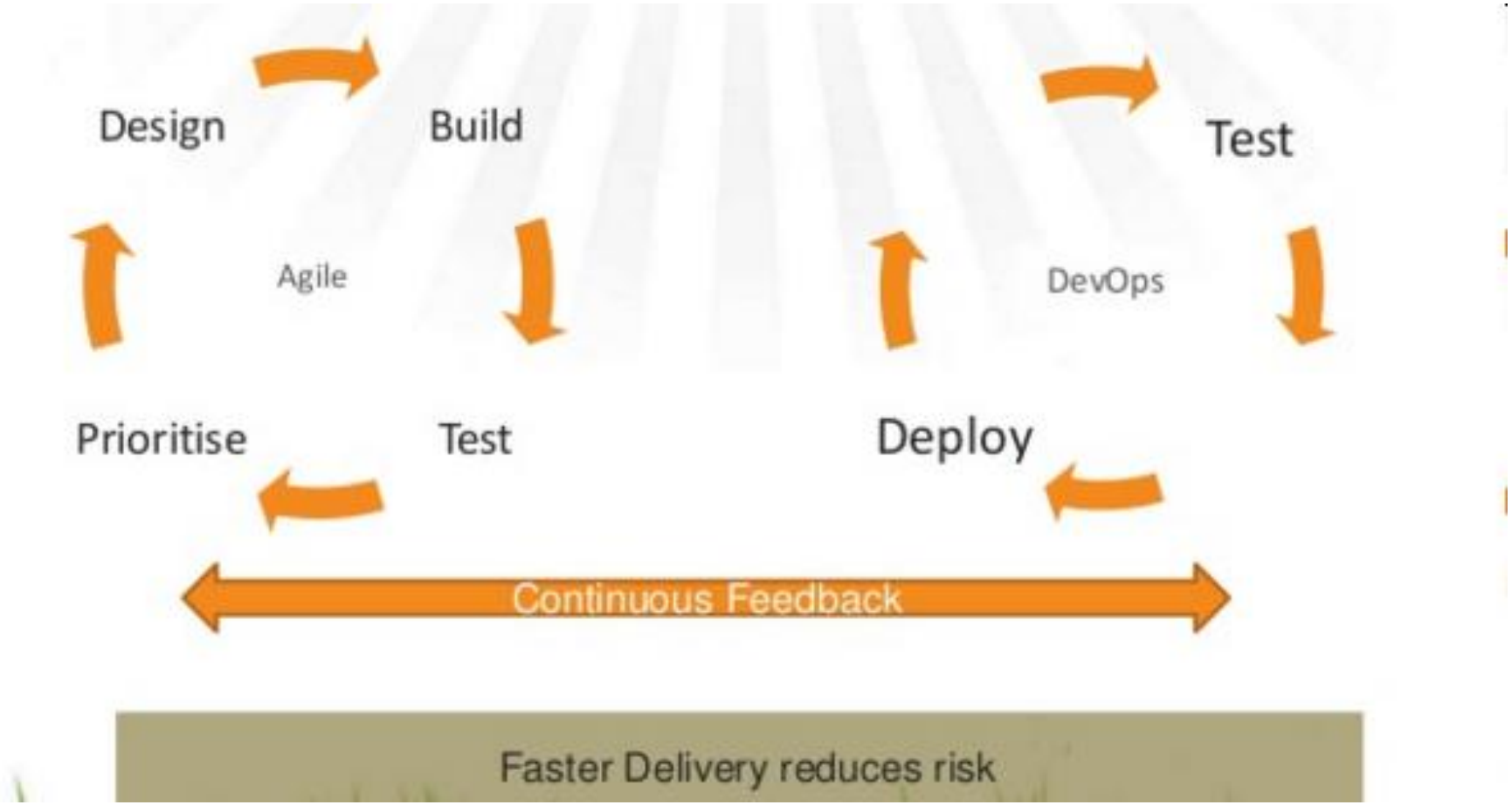
Agile and DevOps

- Agile Development–
  - Addresses the gap between customer requirements and dev + testing teams
  - ross-functional teams to design, develop, and test features/stories prioritised 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

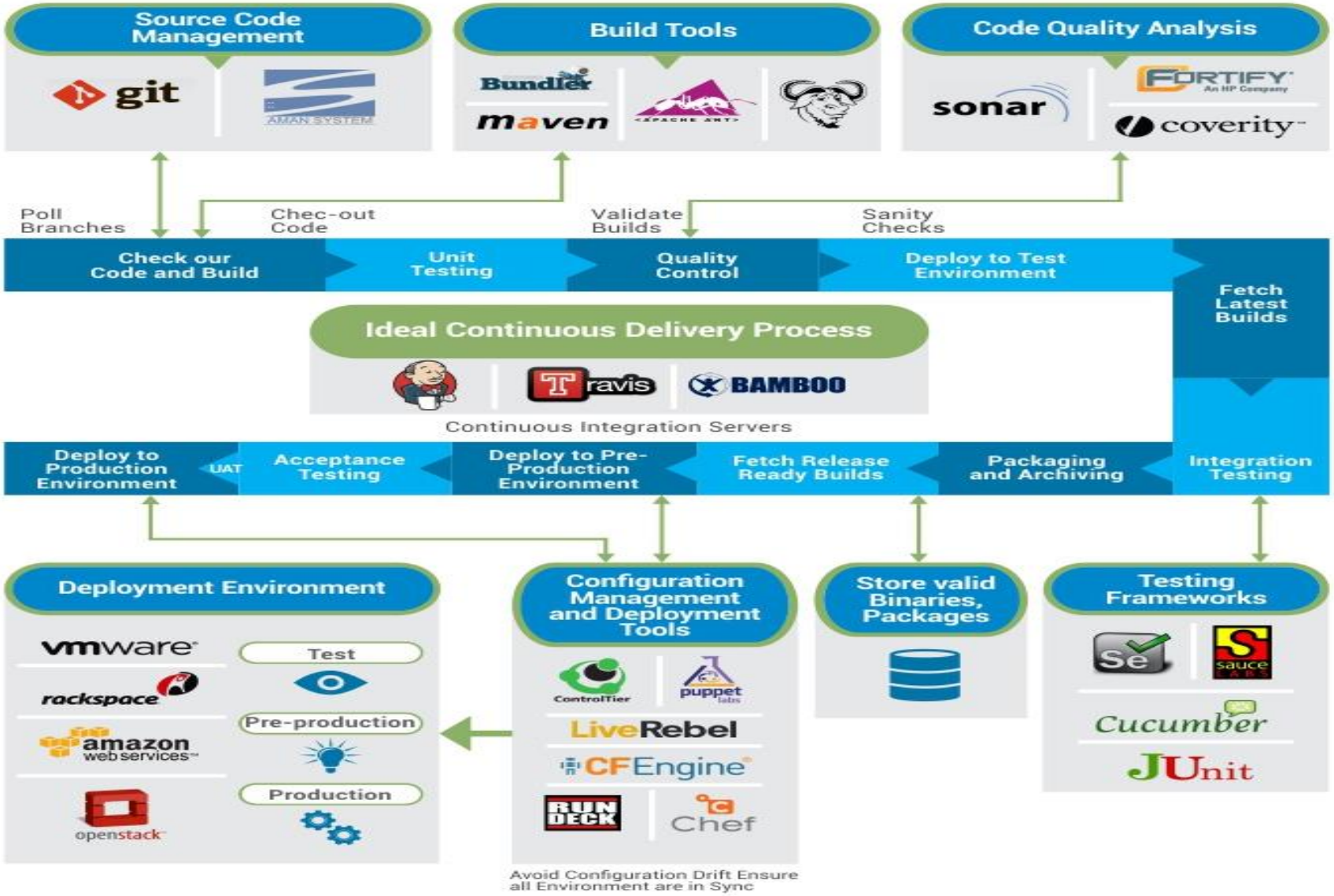


Agile with Dev Ops



Dev Ops Environment and Infrastructure

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.

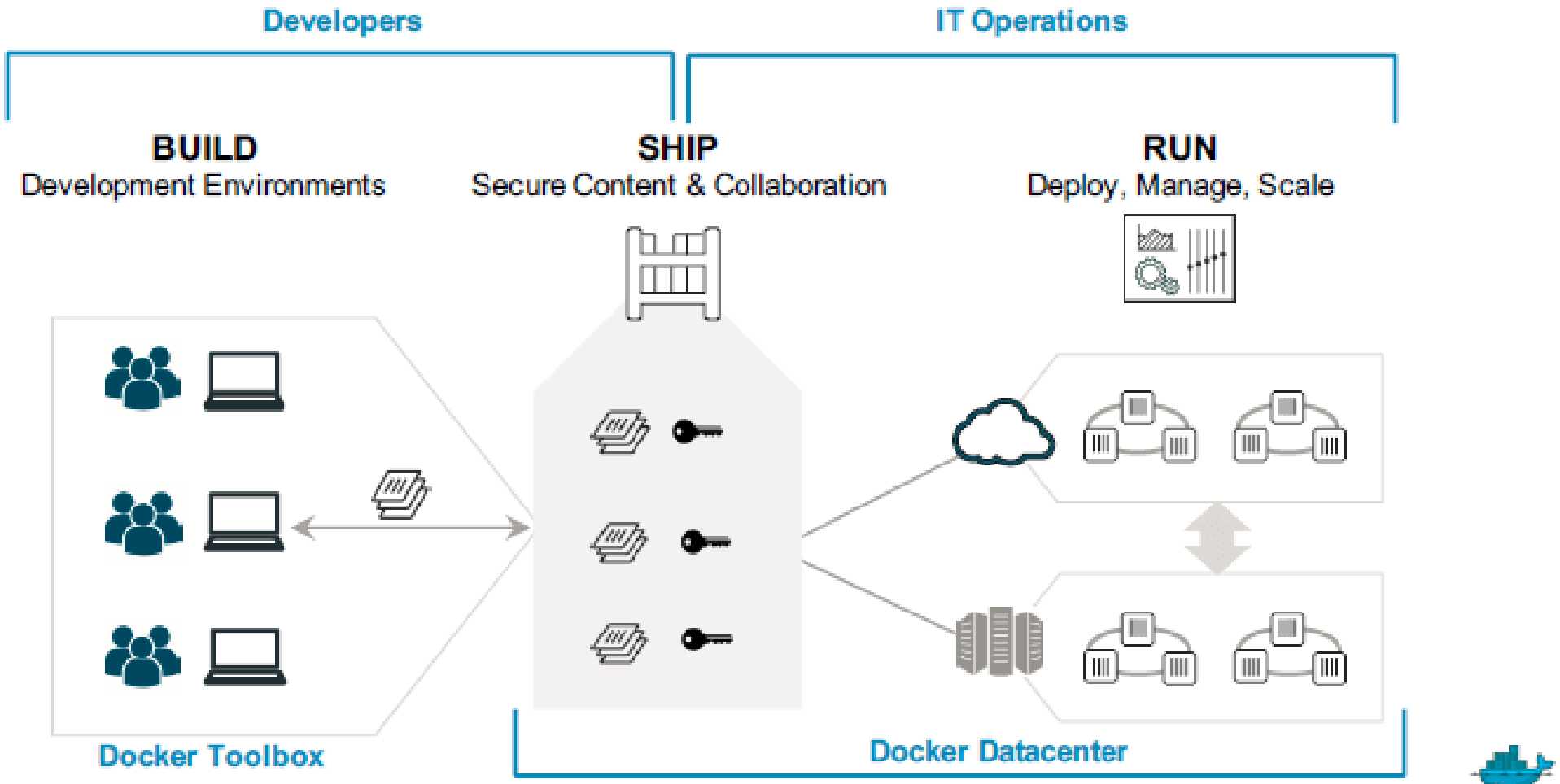
Infrastructure as a Service

- 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
- 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.

CI/CD Tools

- Jenkins
- Travis
- Bamboo

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

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 Assurence

- **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