What is DevOps?

DevOps is the key to Continuous Delivery. How is this achieved? First it is useful to consider the evolution of the previous models of project management, namely Waterfall and Agile.

AGILE

Agile addressed the gap between client requirements and development, but a disconnect remained between the developers and the operations teams, i.e. applications were being developed on different systems to the ones they would ultimately be deployed upon with the assumption that the production infrastrcuture was bigger and more powerful than the development laptop so it would all be fine.

Client + Requirements <–> Developers + Testers <- X -> Operations + Infrastructure

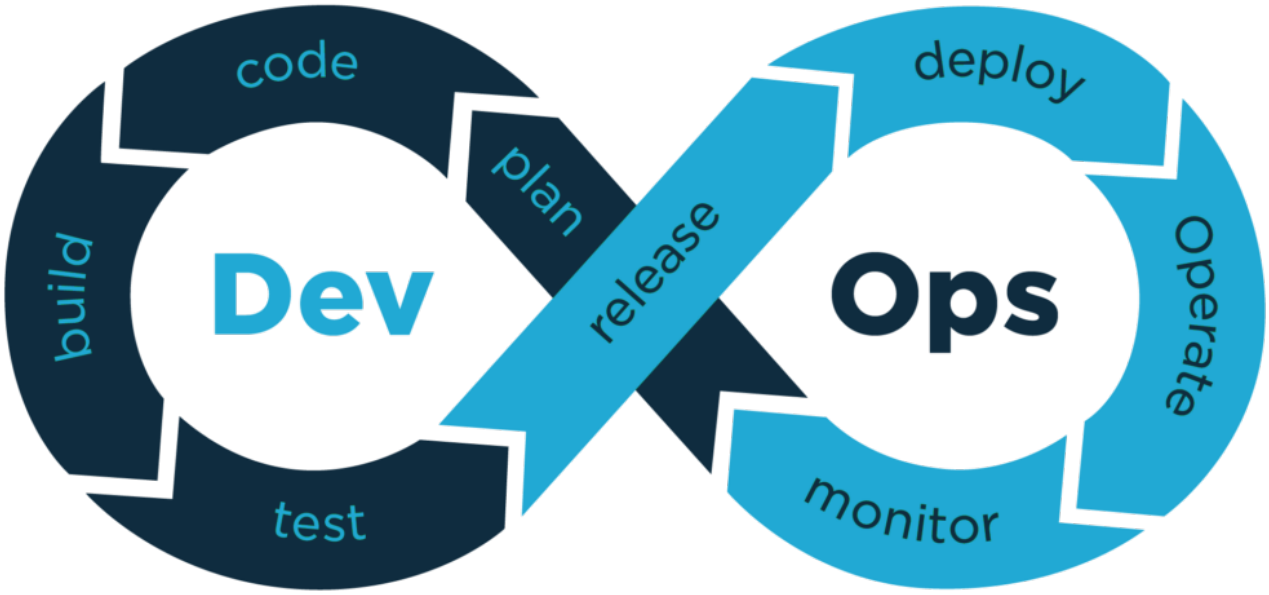
DEVOPS

DevOps is a logical evolution of the Agile shift and addresses the link between developers and operations so that continuous delivery and continuous integration can be achieved along with the promise of fast product to market times and quicker return of value to the client. This utopia is further realised since much infrastructure is now hosted in the cloud and is in itself code (infrastructure as code). This doesn’t so much bring the operations and dev teams closer together as blur the

divide between them, since they now use many common tools.

Client + Requirements <–> Developers + Testers <–> Operations + Infrastructure

It also facilitates a feedback loop rather than a left-to-right delivery paradigm.



The figure of 8 diagram above shows the sequence of phases in a DevOps environment that facilitate continuous delivery, continuous integration and continous deployment. More on what that means below… First lets have a quick look at each of the phases shown in the diagram, starting with the Planning Phase.

It’s worth noting at this point that many open-source tools are used in each stage of the DevOps process. We’ll cover some of the more commonly used tools as we go along.

It is also worth noting that many of these tools are designed to automate the functions of a build engineer, tester or operator.

PLAN

To sit down with business teams and understand their goals. Tools used in this phase are **Subversion** and **IntelliJIDEA**.

CODE

Programmers design code using **git** to carefully control versioning and branches of code that may be a collaborative effort, ultimately merging the branches into a new build. More on the elementary use of *git* here. Code may be shell script, python, powershell or any other language and git can maintain version control of developers local repositories of code and the projects main private and public repositories held online at github that collaborating devs keep in sync with.

BUILD

Build tools such as **Maven** and **Gradle** take code from different repositories and combine them to build the complete application.

TEST

Testing of code is automated using tools such as **Selenium,**

JUnit to ensure software quality. The testing environment is scripted just as the build environment is.

INTEGRATE

Jenkins integrates new features once testing is complete, to the already existing codebase. Another tool used in the integration phase is **Bamboo**.

DEPLOY

BMC XebiaLabs can be used to package the application after Jenkins release and is deployed from the Developement Server to the Production Server.

OPERATE

Operations elements such as Servers, VM's and Containers are deployed using tools such as **Puppet, Chef** and their configuration managed and maintained using tools such as **Ansible** and **Docker**. Like the application hosted on the platform, these tools are used to execute code in the cloud and that code can be maintained using git etc just the same as application code for a consistent, self-healing deployment where the scale of the application may require many identically configured elements to host an application that could also be subjected to attacks.

MONITOR

Monitoring frameworks such as **Nagios** are used to schedule scripted checks of parts of the solution and the consequences of the results collated by **Groundwork** and/or **Splunk>**. A nagios monitoring script should exit with a status of 0, 1 or 2 (OK, Error, Warning) and may be displayed on a board for operators to see, but may also feed back into the development cycle automatically.

So, all these tools, all this code and the utopia of entirely automated, cloud infrastructure is often described as Continuous Integration/ Continuous Delivery/Continous Deployment or "CI/CD" for short. Lets make the distinction between the three…

CONTINUOUS DELIVERY

This is effectively the outcome of the PLAN, CODE, BUILD and TEST phases.

CD = PLAN <–> CODE <–> BUILD <–> TEST

CONTINUOUS INTEGRATION

This is effectively the outcome of the CD phases above plus the outcome of the RELEASE phase, i.e.

CI = CD <–> RELEASE

or

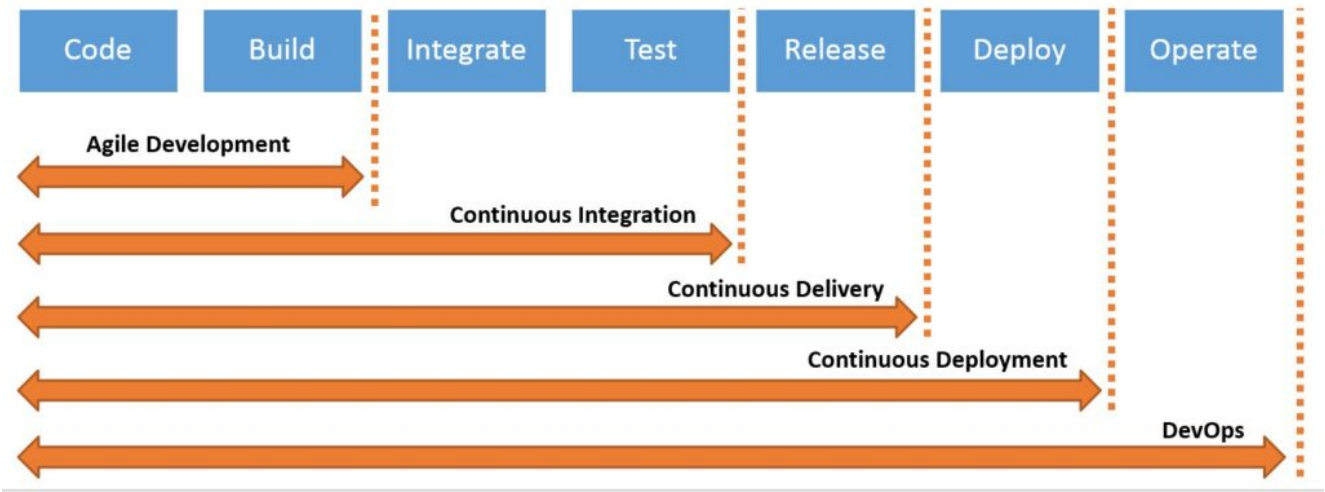
CI = PLAN <–> CODE <–> BUILD <–> TEST <–> RELEASE

whereby the outcome of the RELEASE phase *(defect or success)* is fed back into the Continuous Delivery phases above or moved into the DEPLOY, OPERATE and MONITOR phases or “Continuous Deployment” phases respectively, i.e.

CONTINUOUS DEPLOYMENT

“*success*” outcome from **CI –> DEPLOY <–> OPERATE <–> MONITOR**

So in summary, the terms Continuous Delivery, Continous Integration and Continuous Deployment is simply a collective term for multiple phases of the DevOps cycle…



COMPARISON WITH WATERFALL and AGILE

Waterfall projects can take weeks, months or years before the first deployment of a working product, only to find bugs when released into the wild on a much larger user base than the

developers and testing teams. This is an extremely stressful time if you're the dev tasked with finding and fixing the root cause of the bugs in the days after go-live, especially when the production system is separate in every sense of the word from the developers working environment, not to mention the potential for poor public image, poor customer PR and spiralling costs after heavy upfront costs.

REQUIREMENTS ANALYSIS –> DESIGN –> DEVELOPMENT –> TESTING –> MAINTENANCE

Agile projects use kanban boards to monitor tasks in the Pending, Active, Complete and Resolved columns. Agreed Sprints lasting 2 weeks or 4 weeks *(sprint cadence)* ultimately resulting in a new release, drives value back to the customer in a guaranteed schedule, with outstanding tasks and bugs still being worked on during the next sprint.

SPRINT = [PLAN <–> CODE <–> TEST <–> REVIEW] + SCRUM

DevOps in comparison, heavily leverages automation and a diverse toolset to bring the sprint cadence down to days or even a daily release.

-> PLAN <–> CODE <–> BUILD <–> TEST <–> INTEGRATE <–> DEPLOY <–> OPERATE <–> MONITOR <-

ADVANTAGES OF DEVOPS

As an example, Netflix accounts for a third of all network

traffic on the internet, yet it’s DevOps team is just 70 people.

The time taken to create and deliver software is greatly reduced.

The complexity of maintaining an application is reduced via automation and scripting.

Teams aren’t silo’d according to discrete skill sets. They work cohesively at various phases in the loop, their roles assigned during daily scrums.

Value is delivered more readily to the customer and up-front costs reduced.