



# What is DevOps?

DevOps Practitioner

transforming performance  
through learning

## Outline

- **What is DevOps?**
  - Culture change
  - Keep CALMS and carry on
- **DevOps Tooling**
  - An ideal world
  - Categories
- **The DevOps tool chain**

## What is DevOps?

“a cross-disciplinary community of practice dedicated to the study of building, evolving and operating rapidly-changing resilient systems at scale.”

Jez Humble

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.

Ernest Mueller

DevOps is also characterized by operations staff making use many of the same techniques as developers for their systems work.

Ernest Mueller

3

<http://theagileadmin.com/what-is-devops/>

DevOps (a clipped compound of "software DEvelopment" and "information technology OPerationS") is a term used to refer to a set of practices that emphasize the collaboration and communication of both software developers and information technology (IT) professionals while automating the process of software delivery and infrastructure changes. It aims at establishing a culture and environment where building, testing, and releasing software can happen rapidly, frequently, and more reliably.

## What is DevOps

- **DevOps is a catalysation of concepts into a technical movement**
  - We could sum it up as a movement of **agile system administration**
  - It sprang from applying **Agile** and **Lean** approaches to operations
- **Beyond taking agile to traditional Sys Admin roles**
  - It tries to bridge the gap between **dev** and **ops** roles
  - Understanding the benefits of collaboration between the roles
- **It is a methodology that actively de-silos the traditional separation**
  - Trying to undo the harm that has been done by separating them
  - Allowing us to treat DevOps as an outgrowth of Agile

4

The goal of DevOps is to deliver software to production, in an efficient manner, minimizing risk, and striving for continuous improvement.

Applying Lean principles and especially Lean measurements to a delivery pipeline or even better, the entire DevOps lifecycle, helps progress towards these goals, in a proven, and mathematically astute manner.

DevOps has strong affinities with Agile and Lean approaches.

The old view of operations tended towards the “Dev” side being the “makers” and the “Ops” side being the “people that deal with the creation after its birth” – the realization of the harm that has been done in the industry of those two being treated as soloed concerns is the core driver behind DevOps. In this way, DevOps can be implemented as an outgrowth of Agile –

Agile software development prescribes close collaboration of customers, product management, developers, and (sometimes) QA to fill in the gaps and rapidly iterate towards a better product –

DevOps says “yes, but service delivery and how the app and systems interact are a fundamental part of the value proposition to the client as well, and so the product team needs to include those concerns as a top level item.”

From this perspective, DevOps is simply extending Agile principles beyond the boundaries of “the code” to the entire delivered service.

## What is Agile and Lean

- **Agile manifesto:**
  - *Individuals and interactions* over processes and tools
  - *Working software* over comprehensive documentation
  - *Customer collaboration* over contract negotiation
  - *Responding to change* over following a plan
- **Lean**
  - The core idea is to maximize customer value while minimizing waste
- **Lean thinking is about:**
  - Increasing customer value
  - Eliminating waste (work that does not add value)
  - Management as a facilitator
  - The involvement of all employees
  - Continual improvement
  - Focusing on long-term goals

5

Agile manifesto: <http://agilemanifesto.org/>

Lean: <https://www.lean.org/WhatsLean/>

Lean:

[https://www.ibm.com/developerworks/community/blogs/invisiblethread/entry/lean\\_assessment?lang=en](https://www.ibm.com/developerworks/community/blogs/invisiblethread/entry/lean_assessment?lang=en)

## Agile principles

- Our *highest priority* is to satisfy the customer through early and continuous delivery of *valuable software*.
- *Welcome changing requirements*, even late in development.
- *Deliver working software* frequently, from a couple of weeks to a couple of months, with a preference to the shorter time scale.
- Business people and developers must *work together* daily throughout the project.
- Build *projects around motivated individuals*.
- The most efficient and effective method of conveying information to and within a development team is *face-to-face conversation*.
- *Working software* is the *primary measure* of progress.
- Agile processes promote *sustainable development*
- *Continuous attention* to technical excellence and *good design* enhances agility.
- *Simplicity* – the art of maximizing the amount of work not done – is essential.
- The best architectures, requirements, and designs emerge from *self-organizing teams*.
- At regular intervals, the *team reflects* on how to become more effective, then tunes and adjusts its behavior accordingly.

6

## The Agile Principles

-

## Recap – The Agile Manifesto

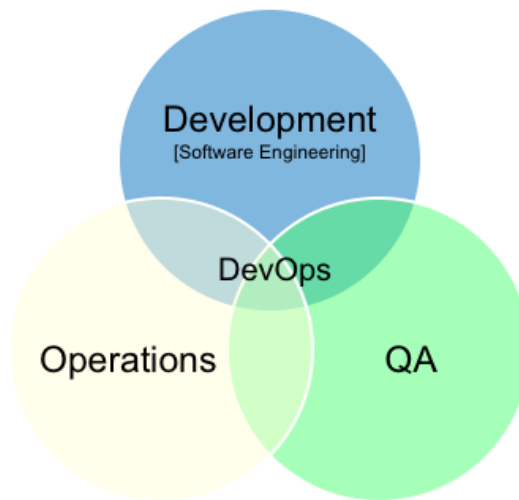
- **The core values in Agile as defined by the Agile Manifesto are:**
  - Agile Values
  - Agile Principles
  - Agile Methods
    - Process specific implementations XP, Scrum, Lean
  - Agile Practices
    - Specific techniques used in conjunction with agile implementation
      - i.e. artefacts used for Agile by developers, planning poker, standups
  - Agile Tools
    - Specific technical implementations of these practices
      - JIRA Agile (grasshopper), planningpoker.com

7

## The Agile Principles

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter time scale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity – the art of maximizing the amount of work not done – is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

## What is DevOps?

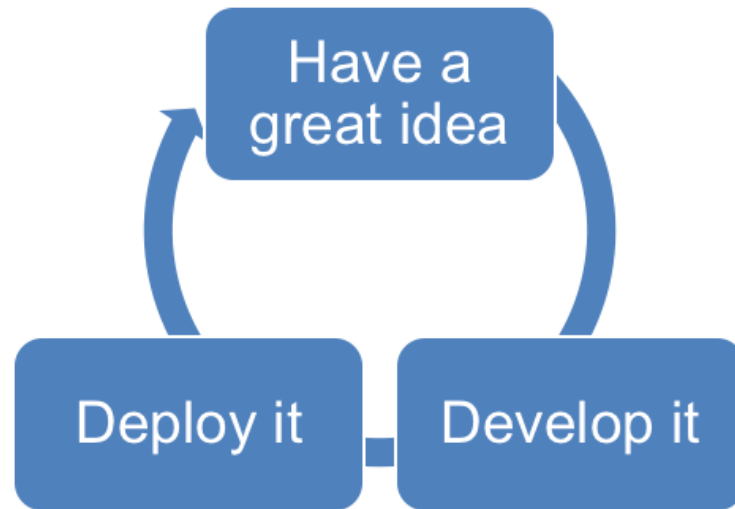


8

**DevOps** promotes a set of processes and methods for thinking about communication and collaboration – between departments of development, QA (quality assurance), and IT operations. In some organisations, this collaboration involves embedding IT operations specialists within software development teams, thus forming a cross-functional team – this may also be combined with matrix management.



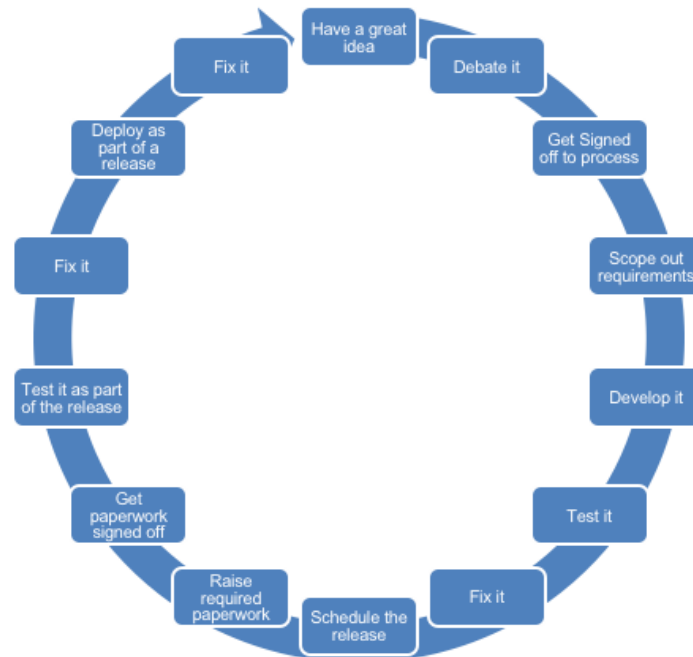
## Startup



9

- No barriers between teams
- Everyone communicates
- Everyone can monitor what's happening
- All areas of business are focused on the same thing
- Speed of delivery is of the essence
- When things break, everyone swarms around to fix the problem
- The software evolves quickly and features are added in incremental chunks
- The ways of working are normally very agile

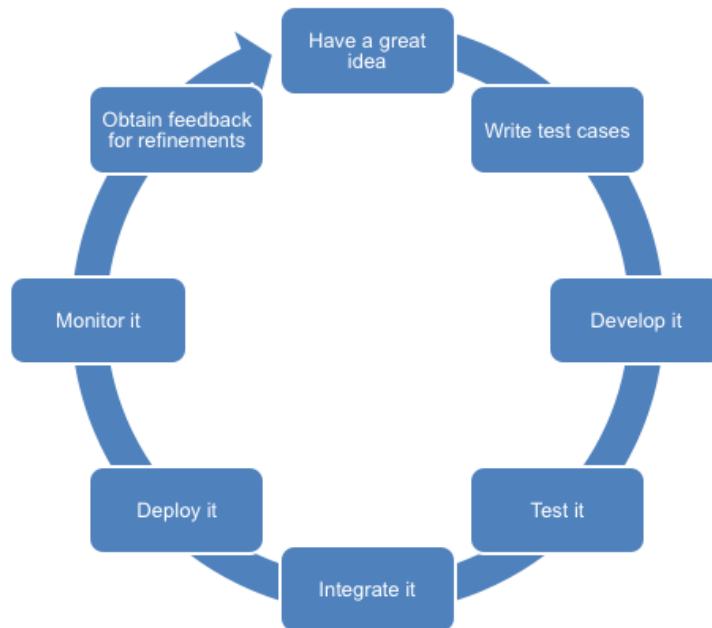
## Corporate



10

- Everyone must be agreed
- Complexity of system slows down production and shipping
- Split in job roles
- Changing scope for external clients
- Deployments are causing system downtime– planned and unplanned

## Best of Both - DevOps



11

No more silos:

- Everyone knows what's going on
- Everyone communicates
- Everyone helps with the problems
- No more 'throwing it over the wall'

Constant change:

- What worked? What didn't? How do we fix it

Automate everything:

- If it works once – it will work thousand times
- TDD and Test Drive Operations

Evangelizing across all areas of the business to share and sell the overall vision and value of Continuous Development and DevOps

Because DevOps is a cultural shift and collaboration (between development, operations and testing), there is no single "DevOps tool": it is rather a set (or "[DevOps toolchain](#)"), consisting of multiple tools.

Generally, DevOps tools fit into one or more of these categories, which is reflective of key aspects of the [software development](#) and [delivery process](#):

- Code — Code development and review, [version control](#) tools, code merging (Git);

- Build — [Continuous integration](#) tools, build status (Maven);
- Test — Test and results determine performance (TDD);
- Package — [Artifact repository](#), application pre-deployment staging;
- Release — Change management, release approvals, [release automation](#);
- Configure — Infrastructure configuration and management, [Infrastructure as Code](#) tools;
- Monitor — [Applications performance monitoring](#), end-user experience.

Tools such as:

- [Docker](#) ([containerization](#)),
- [Jenkins](#) (continuous integration),
- [Puppet](#) (Infrastructure as Code)

## Who should be Included in the DevOps Process?

- **Almost everyone should be involved in the DevOps analysis?**
  - Anyone involved with defining, building, testing, shipping and supporting your product
    - Product owners/managers
    - Program managers
    - Software developers
    - Team leads
    - System administrators
    - Database administrators
    - Testers, QAs, and QCs
    - Business analysts
    - Scrum masters
    - Change controllers
    - Release managers
    - SCM administrators

## Keep CALMS and carry on

- **The pillars of the movement**
- **CULTURE**
  - No more blaming
  - No silos
- **AUTOMATION**
  - Automate everything we can
  - Leave nothing for human error
- **LEAN**
  - Product value for the end user
  - Less waste in the system
- **METRICS**
  - Measure everything
  - What works, what didn't, how do we fix it?
- **SHARING**
  - Learning from mistakes

## DevOps Tooling

- **The DevOps tool set is aimed at supporting the pillars**
  - Automation is key
  - Testing everything as we go through
  - Monitoring what we have in place and catch problems before they escalate
- **There are many tools available for the job**
  - Each company will have its own set
  - Use what is best for you
  - You should be able to swap one tool for another without the entire chain breaking

14

<https://blog.appdynamics.com/devops/devops-scares-me-part-2/> has a list of useful tools! We will be exploring some of them

[illegible]



## The Pillars of Tooling

- **Continuous Integration and Continuous Delivery**
  - We want to be able to deploy changes as they happen
- **Cloud and Virtualisation or Infrastructure**
  - No need for costly data centers, let someone else deal with this for us
- **Automation and Orchestration**
  - Deployment should be reliable, testable and repeatable
- **Monitoring and Logging**
  - We need to know if something is going wrong, notice and adapt
- **Optimisation, Testing and Performance**
  - Test drive by default

16

1. CONTINUOUS INTEGRATION – to enable people work on the same set of info (no worries how it will be merged together) Git – integrates various people work to single code.

1. Continuous Delivery – all code in the single format should build. Notices news, picks it and push it through build process using tools like Maven, Jenkins, Gradle.

2. Cloud and Virtualization – place, where your code will run (like server, cloud)

3. **Automation and Orchestration (key)** – “we don't want people to be involved in this process” E.g. to deploy new software. It's a tricky process. Automation of it means less to worry about it, while we don't have missing stages, nothing is forgotten or incorrectly typed. Orchestration – means that setting up new machine is a pressing of a button. Cloud and Virtualization sets up a new machine and orchestrations makes sure, that correct packages are installed, with right packages and right patches are included etc.

4. Monitoring and Logging – to know what's going with your machines (log pretty much everything, then filter important events); without monitoring you don't know if something goes wrong. Monitoring today – if something is about to happen as well if something happened in the past. Without monitoring is difficult to check if something is going wrong and to correct it on time.

5. Optimisation, Testing and Performance – how to make process faster, better, and with less errors. Optimization – how to make faster delivery. Monitor are changes you done negative or positive. Testing – you can test anything what is written in logical sequence (it includes orchestration, monitoring, logging)

## Aims

- **Consistent, fast and safe delivery of software**
  - We want new changes to be able to be pushed into production as fast as possible
  - You should be able to add and remove as many servers as required
  - Automate as much as possible!

17

We don't want situation: delivering new changes makes to fool down the system.

DevOps tries to make it reliable. Automation helps you to have a lot of reproducibility.

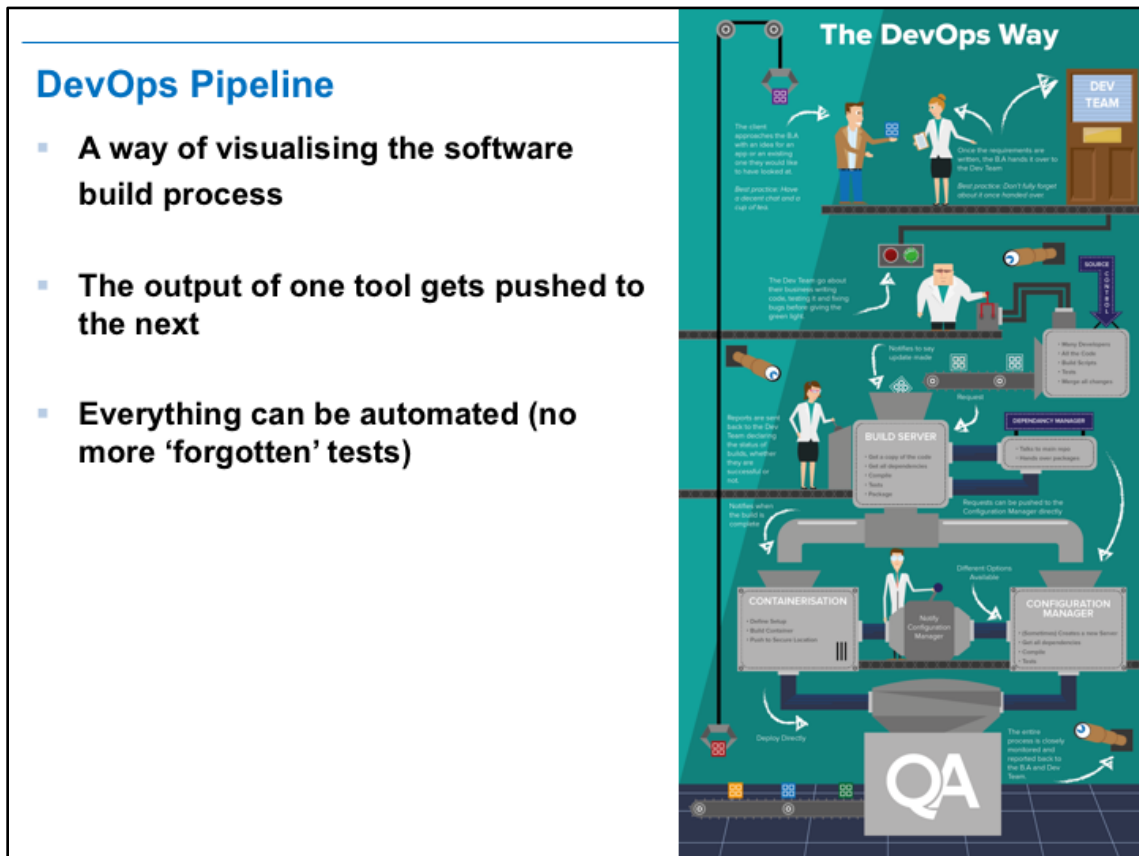
If it worked once – it will work a lot times.

To produce new code its easier with DevOps. Instead to have issues and recover after – our automated system can do in seconds.

DevOps is the latest idea about how to create and deploy software, and it's an entire cultural shift for companies,

bringing an agile approach to the software development lifecycle. The tools are simply a way of helping that process.

Without a culture change, to paraphrase a colleague of mine, you simply have a method of failing to deploy software even faster.



**The aim of the pipeline is to get something that is repeatable, trustable and reliable.**

It is based on the same idea as Test Driven Development - if we test everything and the tests pass, then the code is working as expected.

If a developer submits some code it should run through all the tests automatically,

if it passes then it can be pushed through to deployment without worrying that it is about to bring down the entire system.

The quality of the pipeline is all down to the quality of the testing at each stage.

The tools tend to fall into a series of categories:

1. Initial stages
2. Source Control
3. Building Engines
4. Containerization
5. Configuration Management
6. Monitoring

Automation is key here; we do not want people involved in the middle stages of the pipeline if at all possible.

People are rubbish. They will forget to do things, or decide it's too much hassle

to run those tests at 6pm on a Friday so will skip that step.

We have computers now which are capable of doing boring, repetitive tasks for us, so why not let them do that?

## Initial Stages



- **A client has an idea for a great new product – they talk it over with the company (usually the Business Analysts) who write up requirements and send them to the Developers**

19

<https://www.qa.com/blogs/devops-pipeline>

Before any code is written we have the client and (usually) the Business Analyst (BA).

They sit down, have a cup of tea, and chat about what the client is after.

The outcome of these meetings is to work out the requirements of the project and to manage expectations from the client.

There is no point scoping out the Moon on a stick!

The BA will then go to the development team and hand over the requirements. It is important that they don't then ignore the project from this point on.

If the client does not want to be directly involved with talking to the developers then the BA will need to act as a go between.

Even if this isn't needed, they should stay on side and be helping out with the project. A good set of requirements is one that can be tested automatically.

## Source Control



- The developers write code and submit to source control
- Source control allows many people to work on the same set of files without overwriting each other's changes

20

**The aim with source control is to have a single place where everyone can get a copy of the code for a project.**

Multiple people can work on it at the same time, adding their changes, and then all the changes are merged together into one place.

Conflicts between different people's work are sorted straight away. Git is one of the most popular solutions for this.

It uses a Distributed Model. **Meaning that there is no single point of failure, but this model does open up more change merger issues.**

**Source control is a key part of the pipeline as this is where the developer submits code and tests into the pipeline.**

Everything that is required to build and test (and later, even deploy) a project should be stored in the source repositories.

When new code is submitted to the repository, this sets off the next step of the pipeline - The Build Process.

This can be done by the **git server** creating a push notification and telling the build machine to start, or the build machine can be watching **git for changes**.

## Build Engines



- The build engine automates compiling the code
- Dependency managers are important here, each piece of software is based on many building blocks – dependencies – this is other people's code which does a specific purpose

21

There are two types of tools grouped together in this category.

1-st are the **local build tools and dependency managers** such as Maven , Gradle, sbt or npm.

2-nd are **server based solutions (or dedicated servers)** such as Jenkins, Travis CI or Bamboo.

**Local build tools all work in a similar way.** There is a file which will describe how to build your project.

Maven uses the project object model file - pom.xml whereas sbt uses a build.sbt file. The contents are formatted differently, but you get the same effect.

**This build file describes how to;** compile the software, what dependencies are required and where to get them from.

The build file can also look at code style and report test coverage statistics.

**Importantly, what these build tools do, is they always build your project in the same way; using the same building blocks.**

No more issues with different versions of libraries being used, as the build tool will pick them up for you.

**Server based build tools work alongside the local build tools.**

They will notice when a change is made to the source repository, clone a copy of the code and run the Maven, Gradle or sbt jobs to build the project.

They can also set off pre or post-build steps, such as deploying to a server, or

informing another service that the built project is ready.

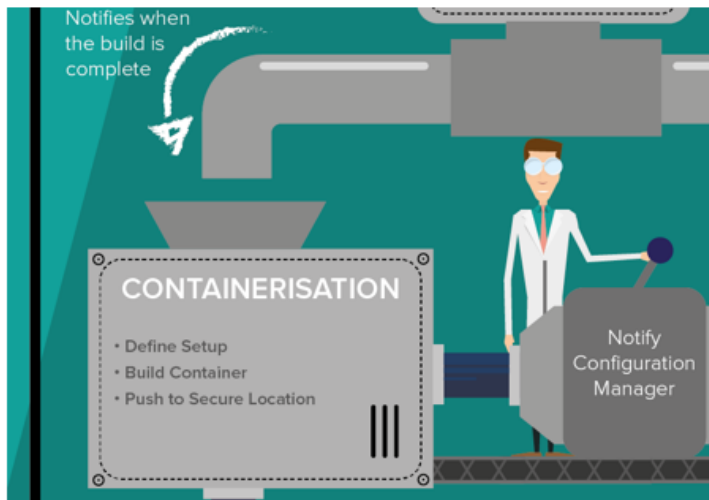
Jenkins is one of the most popular options out there *for build management*; it was originally used for mostly Java projects, but it has a very active community behind it meaning there will be a plugin for just about any: language, reporting system, test or deployment tool.

**After the project is built, the pipeline splits.** Some companies deploy their code directly from the build server.

Others send a notification to a config management tool to pick up the built code and deploy it. A new kid on the block is Containerisation.



## Containers



The process then splits. Some people use containers, others jump to config management. Some use both!

- **Containerisation creates a box with the software and everything needed to run the software inside it. This can then be handed over to the servers as a single unit, rather than saying “Oh, did you install x, y and z first?”**

22

Once your code has been build, it can ten go 1 or 2 direction:

1. **Containers - we can have something like Docker engine, what will pick up your build artifacts, put the container around it and ready to push it directly to the production. It could go straight away in to the cloud or to configuration management side.**
2. Configuration management – there are systems that ensures that everything is installed that should be there: is your OS patched, after it – is software deployed correctly. This can come either from the container, or directly from the build artifacts and deploy them directly using something like tomcat.

Software containers are much like real containers. A box does not care about the contents inside it.

If you have 10 boxes, all the same size, they will still stack the same if they contain the same thing or vastly different things.

And so containers for software are boxes which contain code and everything that code relies on to run correctly.

**They are small, self contained units which can be loaded and unloaded on any operating system which has an engine to do so.**

The most popular of these tools is Docker.

Docker **builds a container from a script known as the Dockerfile**. This script is just a text file, so it can be stored along with the code in source control.

**Build systems like Jenkins** can even set off the Docker build process as a post-build step.

Containers are designed to be able to be run on any system, they are entirely infrastructure and content agnostic.

This means that I can build a container on a Windows desktop machine, transfer it to a Linux machine in the Cloud and it will still run in the same way on both systems.

All the files, libraries, settings and even the operating system the Container need, are all inside it.

The best practice with Containers is to make them as small as possible so they are quick to start or transfer between machines.

Containers should be able to be started, stopped and removed at will. **So there should be no data stored inside the Container.**

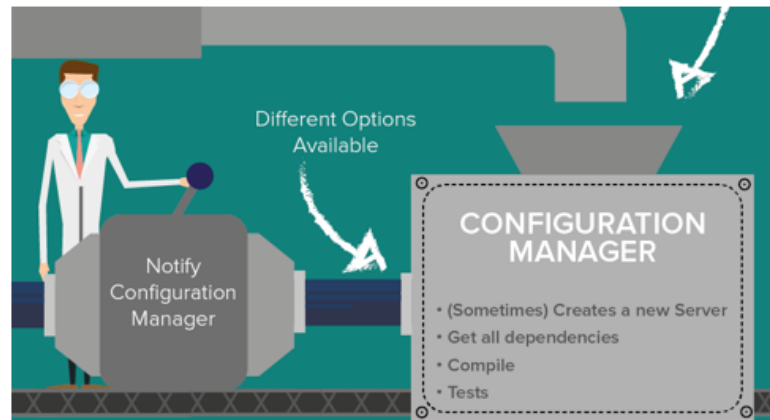
Docker and all the other containerization groups have started a standard for Containers - [runC](#).

This means that in the future you will be able to create Containers with one system and move it to another.

Then we have another split. Containers can be deployed directly (see Docker Machine, Swarm or [Kubernetes](#)) or they can be passed over to the config management tools.

## Config Management

The built code can skip the container stage and be deployed directly to a server using config management tools



- **Configuration Management ensures that everything is installed on a server that needs to be there. It then checks that everything stays in the expected state over time**

23

Once your code has been build, it can then go 1 or 2 direction:

1. Containers - we can have something like Docker engine, what will pick up your build artifacts, put the container around it and ready to push it directly to the production. It could go straight away in to the cloud or to configuration management side.
2. **Configuration management – there are systems that ensures that everything is installed that should be there: is your OS patched, after it – is software deployed correctly. This can come either from the container, or directly from the build artifacts and deploy them directly using something like tomcat. This will have a bunch of tests: is your container build correctly or your config management tools has happened.**

**Configuration Management is all about ensuring that servers** (or other machines) **are in the state they are expected to be in.**

Tools such as [Chef](#), [Puppet](#) or [Ansible](#) are the key here (although, like everything, there are probably a dozen more being used!).

**Config Management tools** generally involve a master server of some variety, which holds the configuration for all the agents.

**The two styles here are a push and pull.**

- **Push style** systems send a notification to all the agents telling them they need to update or to check their current configuration is in line with what is

required.

- Chef and Puppet operate a **pull style notification** where the agent checks in every half an hour to see if there are any changes.

Again, (and there is a pattern here), **all the configuration is held in a series of text files.**

Chef and Puppet use a Ruby-like syntax to express how a machine should be set up (see my [previous blog post](#) for how to get started with Puppet!).

As these files are all just plain text they can also be stored in source control, alongside the code, the tests and the Docker container build file.

There are modules for configuration management tools that also deal with the idea of Infrastructure as Code.

This means even the number of servers, and how they are networked together is expressed in terms of files.

There are different systems for different Clouds; AWS uses CloudFormation and Azure uses resource management templates. Both do the same job, describing the setup for each of the machines in the network.

There are also other systems that offer an abstraction layer on top of this, such as [Terraform](#) . ***This tool can link together systems in different clouds or even in your data centres.***

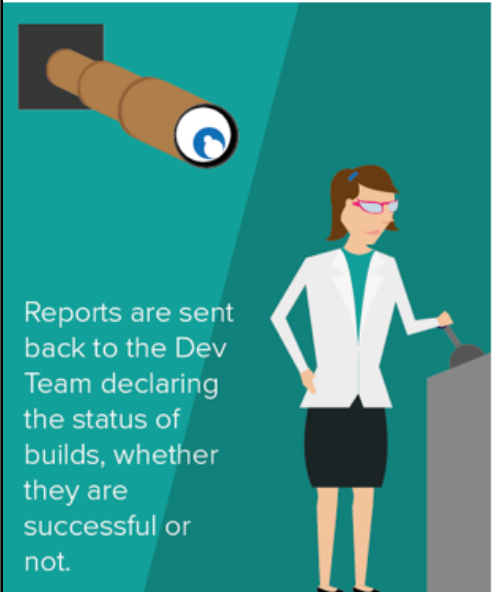
There is a **Docker plugin for Puppet** which will pull and start up containers on a base system.

The split tends to be the people who are running only microservices who can use something like Docker throughout and the people who run legacy systems, or anything where data is to be stored.

**Container and Config Management systems have a lot of cross over, but they are both worth having in different situations.**

## Monitoring

The entire process is monitored end to end




Reports are sent back to the Dev Team declaring the status of builds, whether they are successful or not.

If anything goes wrong in the pipeline then the developers need to know, as well as the operations team.

Problems are fixed by the whole team, not just one person's role

The entire process is closely monitored and reported back to the B.A and Dev Team.



Monitoring is across of each of these systems. We should be watching everything starting from “what is going on on your Git machine”, “what is happening when people submits new code”, “are the push notifications ben generated correctly”, “whether the infrastructure is set up and your config management tooling stuff is installing things in right order”.

Monitor should save a logs. You never know if something will go wrong until it will happened. And after then you can go and find what problem was happened and think how to solve it.

Is always better to have more monitoring then less.

***Monitoring is often seen as a bit dull*** - something to leave until the end, but it is key in any successful pipeline.

Changes can be made to the infrastructure via changes to text files.

The monitoring tools tell you whether that change was beneficial.

Monitoring tools can tell you if a machine is overloaded with requests and then instruct the config management tools

to create some more machines and add them to the load balancer until the traffic calms down again.

Monitoring is also the first line of defence against something going wrong.

In an ideal world, the **monitoring system will pick up any problems with a system before users have a chance to report anything is wrong.**

Then either automated systems can kick in to fix the problem, or a notification is sent to the people running the pipeline to have a look and solve issues.

*The monitoring systems should be available for anyone to look at.*

The Developer should be just as interested in why part of their program is taking a long time to complete as the database staff

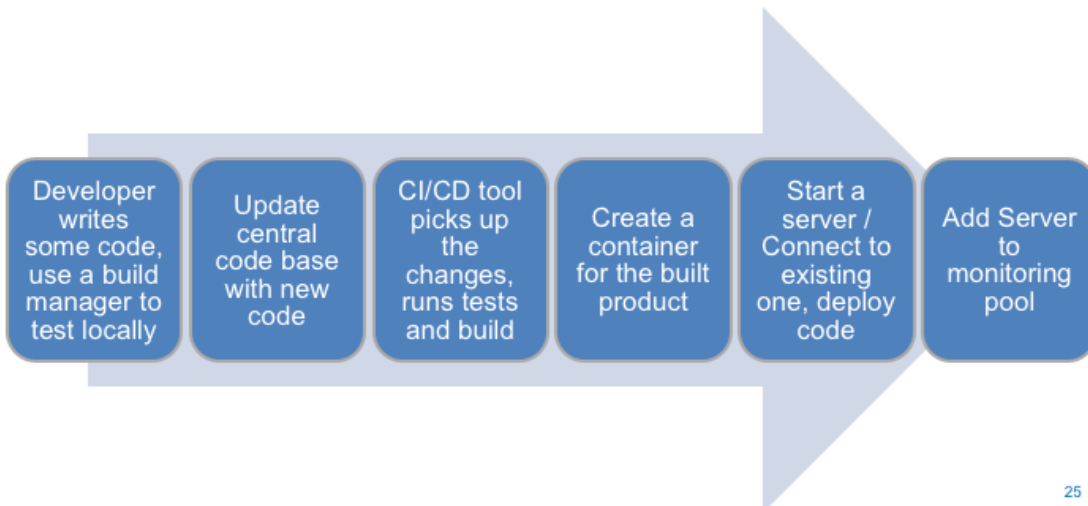
who are trying to speed up data retrieval and the ops staff who are trying to ensure that there is a 99.999% uptime on the system.

These metrics can also feed back through to the client and the business analyst as they sit down and talk about the next step of the pipeline. Maybe over another cup of tea.

***The pipeline is just a tool, made up of other tools. It allows for faster, safer and successful software delivery.***

## Discussion – What tools are available

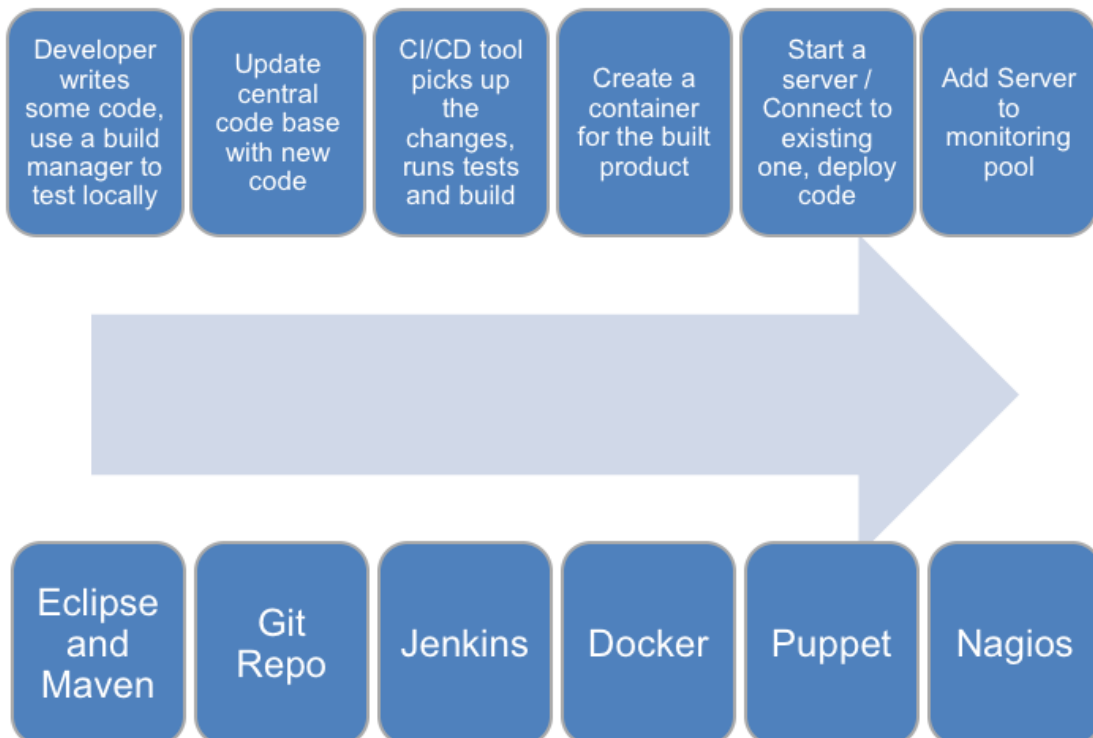
- **What have you heard of?**
- **What have you used?**
  - What was good? Bad?



25

CI/CD tool Docker (continuous integration/continuous deployment)

## The Tool Chain



The source will be build a Maven and tested locally.

Then will be pushed it to Git repository (GitLab or BitBucket)

Jenkins will pick up changes, that happened on GitLab/BitBucket and build the software for us. This will start containerization process with Docker automatically

Jenkins will build a container and pull it to the Docker.

Puppet will be watching. Puppet agent will be checking every 30 min. Puppet will getting new copy of the code from the DockerHub and put it into production

Nagios will watch entire process and monitor end to end process what is going on with this.



## To read more about DevOps

### Articles:

- Top DevOps Tools: 50 Reliable, Secure, and Proven Tools for All Your DevOps Needs (link: <https://stackify.com/top-devops-tools/>)
- What Is DevOps? (link: <https://theagileadmin.com/what-is-devops/>)
- How to choose the right DevOps tools (link: <https://www.atlassian.com/blog/devops/how-to-choose-devops-tools>)
- Periodic table of devops tools (link: <https://xebialabs.com/periodic-table-of-devops-tools/>)
- DevOps Efficiencies for 2018 and Beyond (link: <https://medium.com/@MentorMate/devops-efficiencies-for-2018-and-beyond-2715c10de632>)
- DevOps Predictions for the Future (link: <https://devops.com/devops-predictions-future/>)
- Technology trends 2017: These are the most popular tools (link: <https://jaxenter.com/technology-trends-2017-these-are-the-most-popular-tools-132109.html>)

## Summary

- **What is DevOps?**
  - Culture change
  - Keep CALMS and carry on
- **DevOps Tooling**
  - An ideal world
  - Categories
- **The DevOps tool chain**