(Oltre lo) Sviluppo Software

Introduzione

Davide Salomoni – davide@infn.it

About this introduction

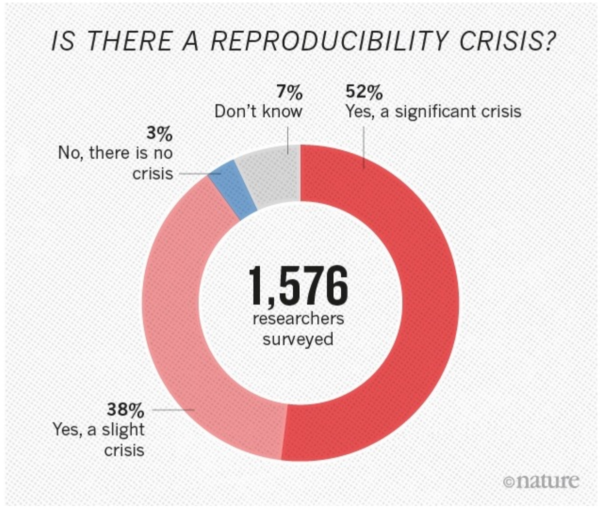
- This course will **not** discuss tools that one uses to develop applications (e.g., compilers, languages, algorithms and their optimization, etc.)
- It will rather focus on **tools that support** the development, testing and deployment of applications in a modern environment.
- Note that it will also **not** discuss details about distributed resources and infrastructures, but it will assume that we live "in the Cloud".
  - **Here, Cloud: a "software-defined world"** (software-defined networking, software-defined storage, software-defined computing).
  - This course, that will mostly be hands-on oriented, will then tap on what it means to write **"cloud-native applications"**.
- This introduction **covers the general points** that you will explore in further lectures.

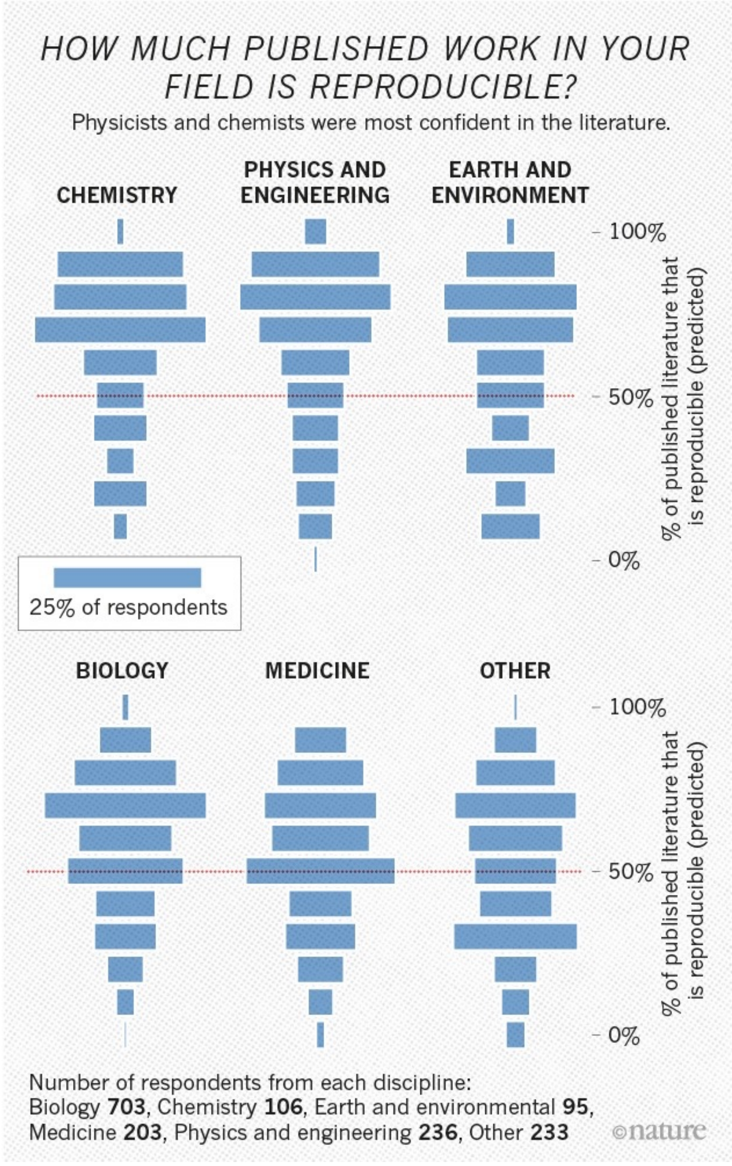
An essential point: Cloud Automation

- Given the complexity of modern applications and environments, **automating development processes** is essential.
- Cloud Automation is therefore **a set of processes and technologies** that allow to *automatize* several operations related to Cloud computing.
- Doing things *by hand* is rarely a good idea when complexity increases, and we will see in this course some relatively complex technologies.
- This is closely linked to a key issue, that is, **reproducibility**.

Cloud automation as key to reproducibility

- For examples of Cloud automation connected to reproducibility, see e.g. "Cloud Computing May be Key to Data Reproducibility".
  - See also Nature, Vol. 533, 26 May 2016, pp. 452-454, "1,500 scientists lift the lid on reproducibility".





Number of respondents from each discipline: Biology **703**, Chemistry **106**, Earth and environmental **95**, Medicine **203**, Physics and engineering **236**, Other **233**

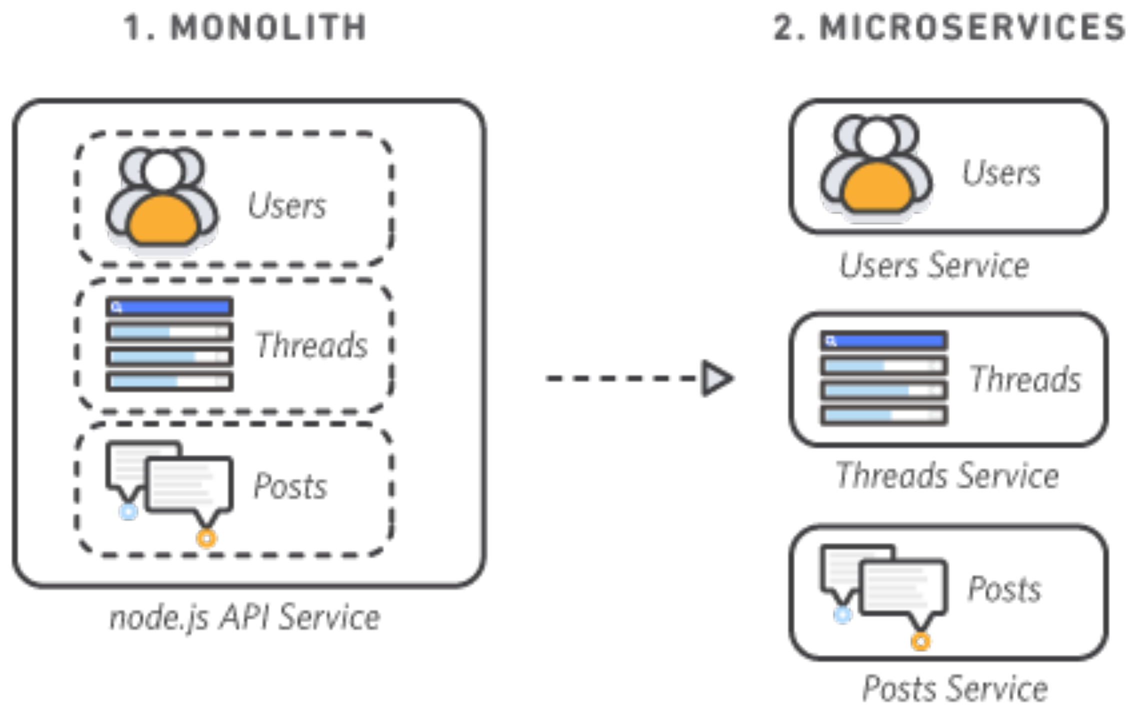
Microservices (1)

- When we start thinking about applications designed for the Cloud (where we have potentially many identical resources at our disposal), a common analogy that is made is one of picturing **traditional apps as pets** (each one is unique and irreplaceable), compared to **Cloud-native apps as cows** (many identical instances of a functionally equivalent "item").
- **Microservices** are a way to build applications as **a collection of (potentially many) small autonomous services**. This is the opposite of creating a big software or service, or anyway a few *fat ones*, called sometimes *monoliths*.

Microservices (2)

- At high level, microservices reflect at the architectural level a **culture of autonomy and responsibility**: a single microservice can be developed and managed independently by different teams, using maybe different development environments (for example, different languages).
- In microservices architectures, **multiple, independent processes communicate with each other through the network**.
- As we will see later, these processes are often encapsulated in *Docker containers*.

Application architectures



Monoliths vs. microservices

Monolithic Applications

- Do everything
- Single application
- You distribute the entire application
- Single database
- Keep state in each application instance
- Single stack with a single technology

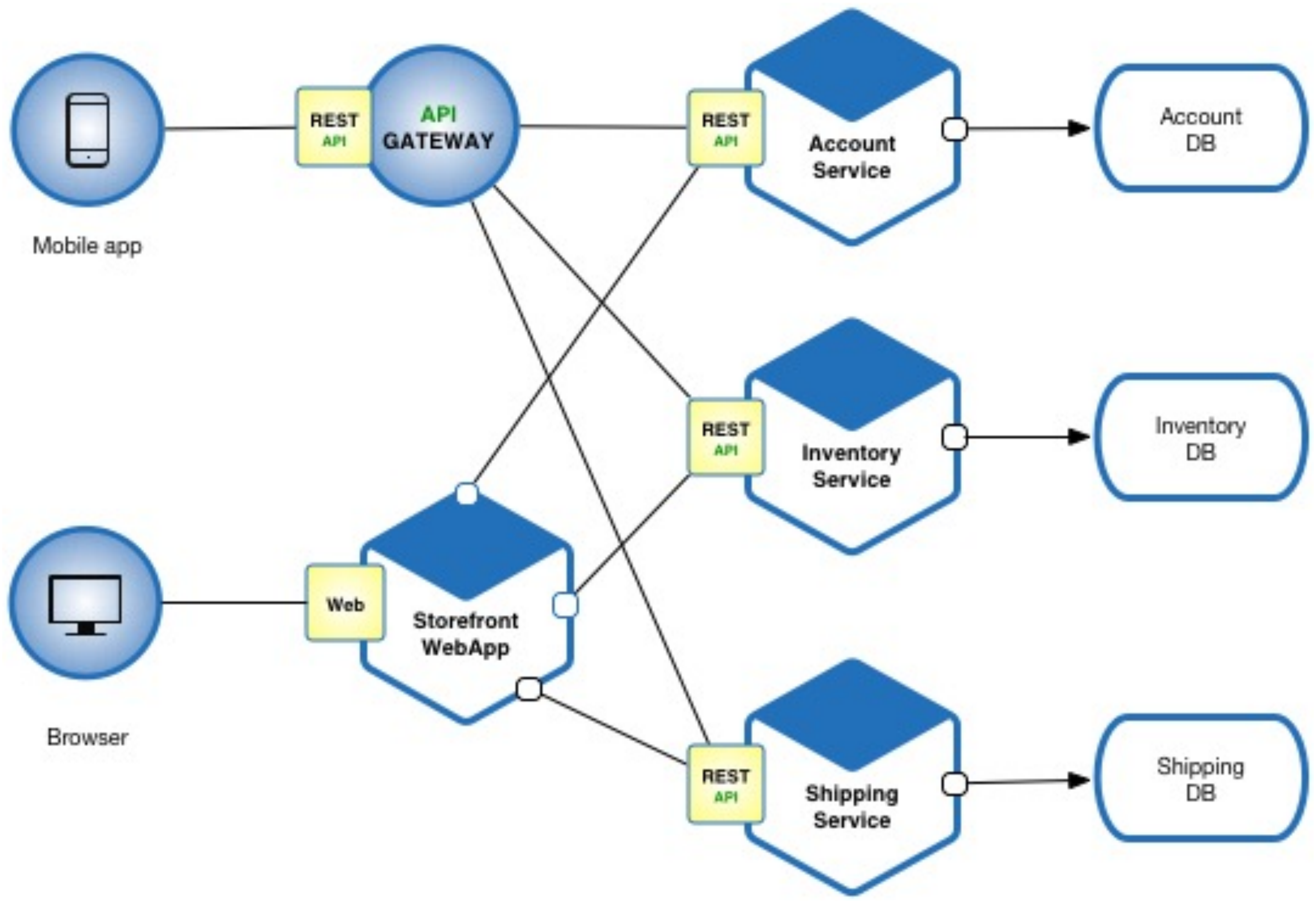
Microservices

- Each has a dedicated task
- Minimal services for each function
- Can be distributed individually
- Each has its own database
- State is external
- Each microservice can adopt its own preferred technology

Adapted from AWS

An example of a microservice architecture

- This is the structure of a typical **e-commerce application** (from https://microservices.io/patterns/microservices.html)
- The microservice pattern, however, applies in a similar way to any type of applications.



All good with microservices?

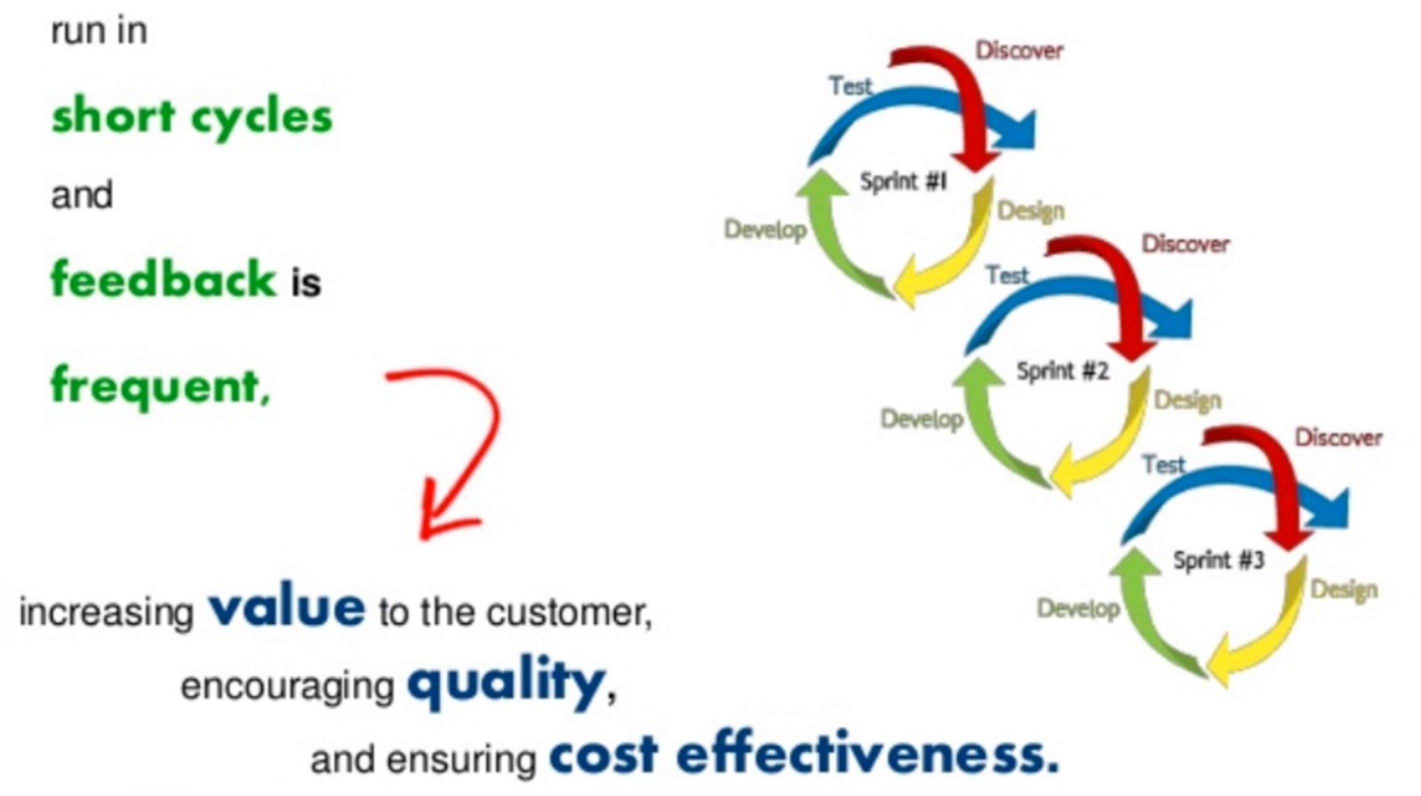
- Of course not. **There are cases when monolithic applications might make more sense**.
- With microservices, **you must typically consider**:
  - Deployment of each microservice independently.
  - Microservice orchestration.
  - Unification of the format of software integration and deployment pipelines.
  - Compared to monolithic systems, there are more services to monitor.
  - Since they form a distributed system, the model is more complex than with monoliths.
- **However, with microservices:**
  - Reliability is much easier, because if for example you happen to break one microservice, you will affect only one part, not the entire app.
  - Scalability is much better. With monoliths, horizontal scaling might be impossible and, when possible, it is connected to scaling the entire app, which is typically inefficient.

Automation of the release pipelines

- Strictly related to the microservice architecture is the concept of **DevOps**.
- DevOps is a pattern for developing applications where **Development and Operation practices tightly integrate**.
  - In other words, rather than (1) writing a full “production level” application, (2) releasing it and then (3) waiting for operational feedback, the DevOps application release process is much more **agile**, and it follows **tight release and feedback schedules**.
  - This is a concept that extends beyond the people who practically do development and operations. It includes end users as soon as this is possible.

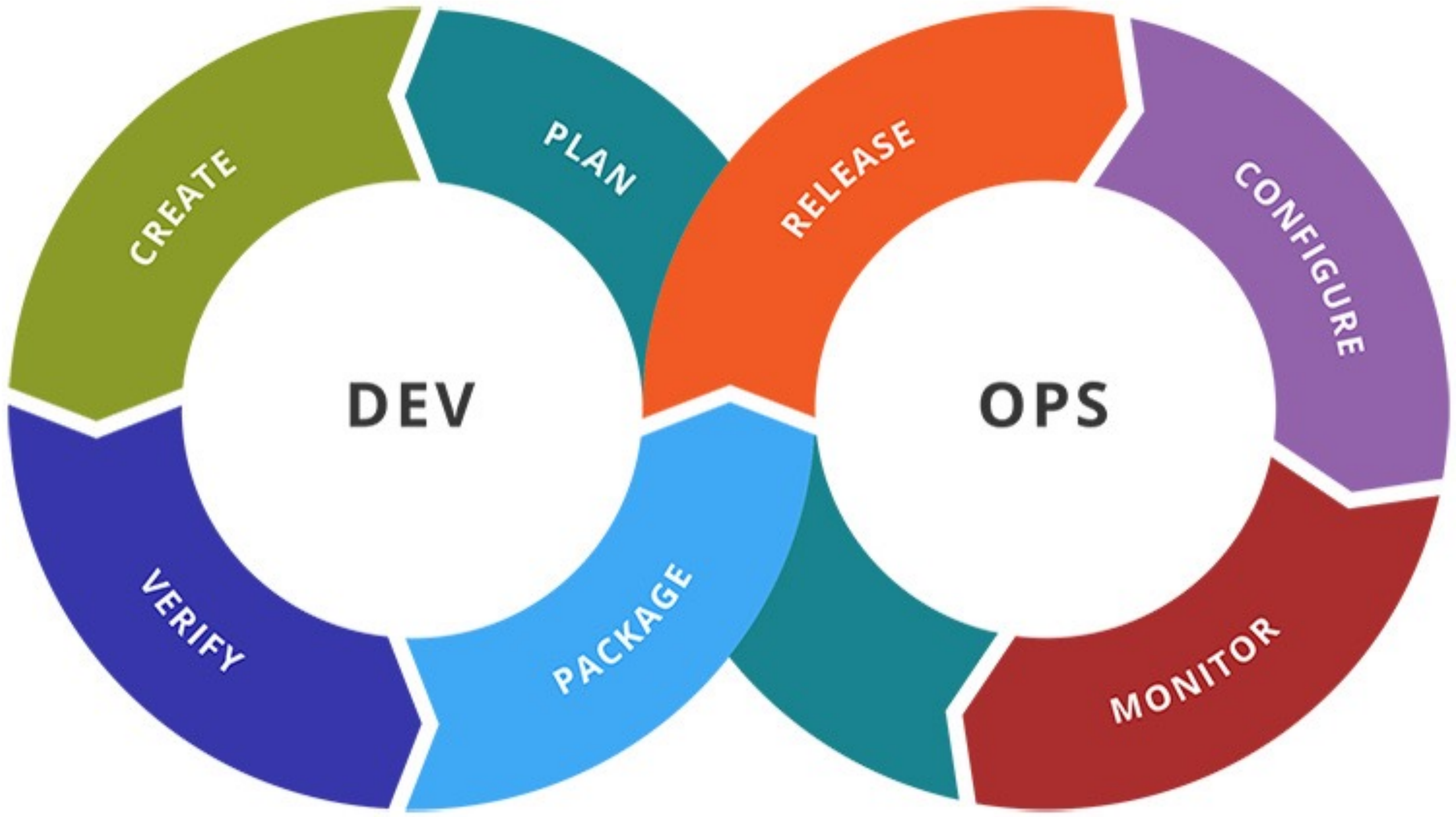
Release early, release often

- The DevOps *mantra* is "**release early, release often**": this implies utilizing a set of *tools and processes* to facilitate **automation, monitoring and continuous integration** of all the involved components (microservices, for example) to quickly complete the development and delivery cycles.
- This is an example of **risk reduction** in software development.


run in **short cycles** and **feedback** is **frequent**, increasing **value** to the customer, encouraging **quality**, and ensuring **cost effectiveness.**

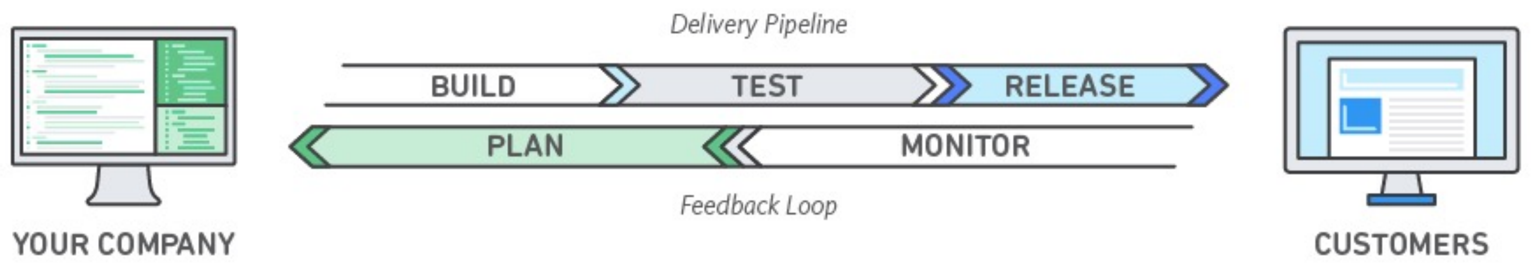
Source: https://medium.com/@warren2lynch/scrum-philosophy-release-early-release-often-a5b864fd62a8

DevOps



Source: https://nickjanetakis.com/blog/what-is-devops

DevOps benefits



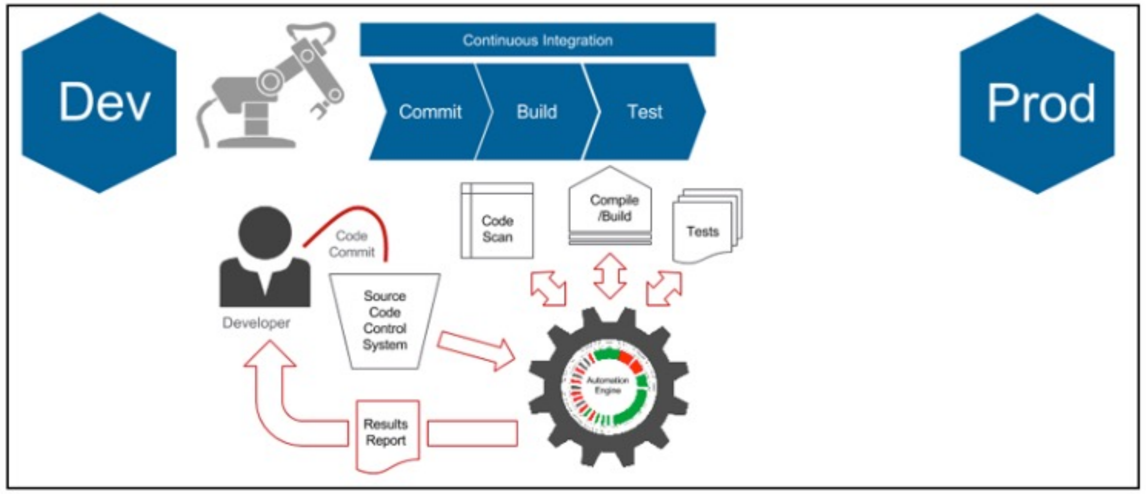
- **Speed** – microservices & continuous delivery
  - Innovate faster
  - Better adapt to changing requirements
- **Rapid Delivery** – continuous integration and delivery
  - Higher release frequency
- **Reliability** – continuous integration and delivery, monitoring & logging
  - Ensure the quality of application updates
- **Scale** – automation, treat infrastructures *as code*
  - Operate and manage infrastructures and development processes at scale
- **Improved Collaboration**
  - Less friction, more effective teams
- **Security** - automated compliance policies, fine-grained controls, configuration management
  - Move quickly but preserve control and compliance

The DevOps principles

- DevOps is a comprehensive way of thinking **covering all the stages of an application lifetime**.
- It is particularly applicable to **distributed, microservices-based applications**, which we typically find in Cloud environments.
- It is therefore important to know its main principles and **try to apply them whenever we write applications, be they small or big**. Let's now see them in some detail.

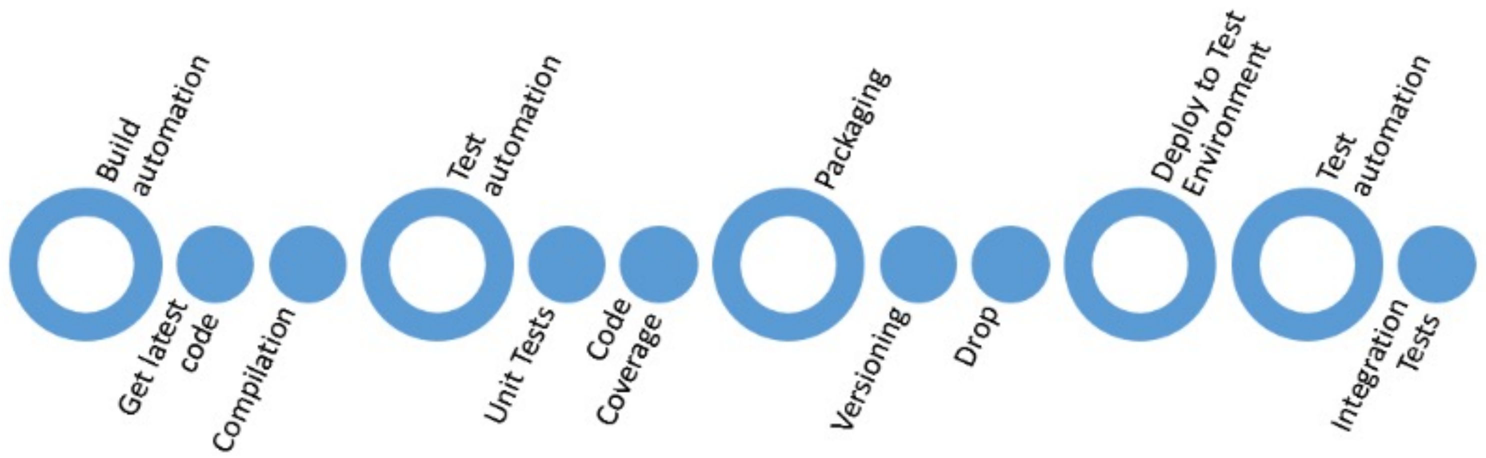
Continuous Integration

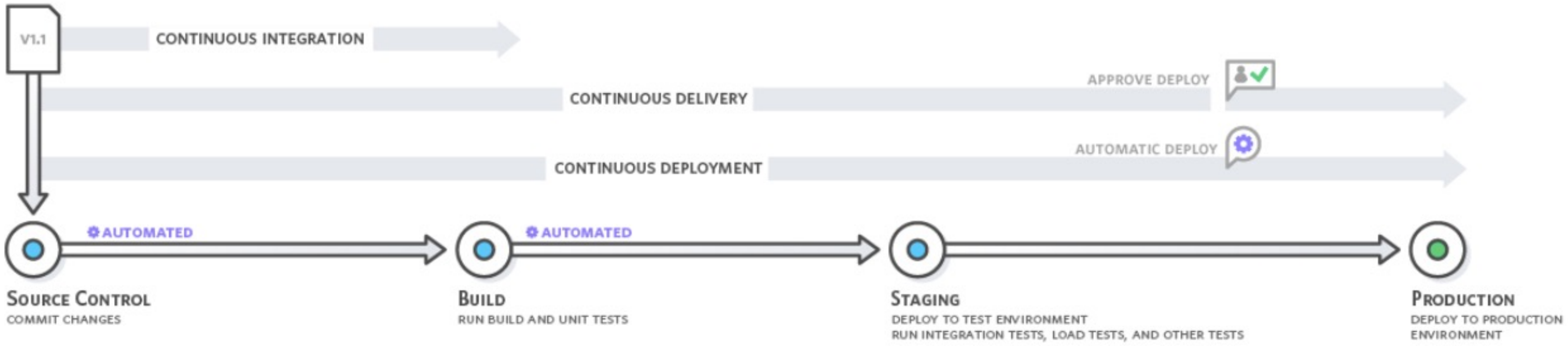
- **Continuous Integration** is a software development practice where developers **regularly merge** their code changes into a **central repository**, after which **automated builds** and **tests** are run
  - The result: **deployment packages** that can be used by Continuous Deployment (discussed later) for deployment to multiple environments.
  - A widely used tool for this: **Jenkins** (https://jenkins.io).



Source: https://jaxenter.com/how-to-move-from-ci-to-cd-with-jenkins-workflow-128135.html

Continuous Integration





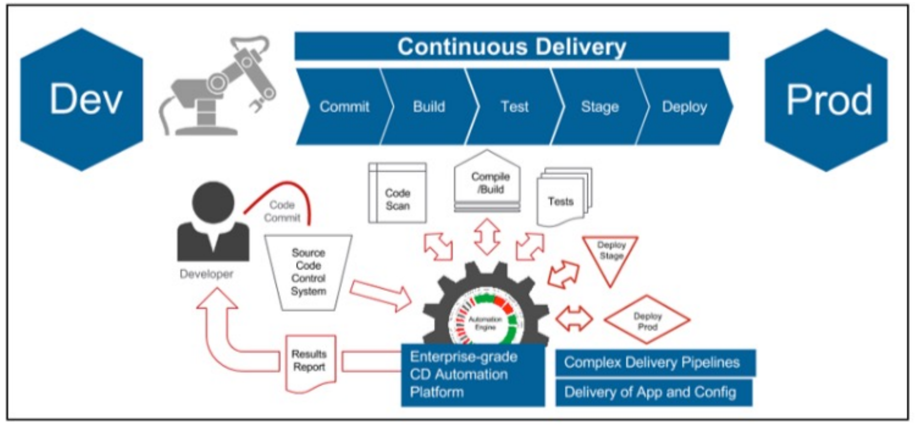
Which Continuous Integration if I use Python?

- Python does not need a "compilation step". However, you can and should still use some Continuous Integration best practices in your projects, even if you only use Python. For example, you most likely want to perform some Quality Assurance tests to be run *automatically*, such as:
  - **sloccount** to count the lines of code (that is, non-blank, non-comment) in a program, not only in Python. This seems simplistic, but it can give you an estimate about the complexity of a project. See https://dwheeler.com/sloccount/.
  - **Pylint** is "a Python static code analysis tool which looks for programming errors, helps enforcing a coding standard, sniffs for code smells and offers simple refactoring suggestions". It is sometimes annoying, but I would say it is a must use. See https://pypi.org/project/pylint/.
  - **Pytest** (https://docs.pytest.org/en/latest/index.html) and **Nose2** (https://github.com/nose-devs/nose2) make it easy to write tests for code coverage. Never underestimate the importance of writing tests in your programs!

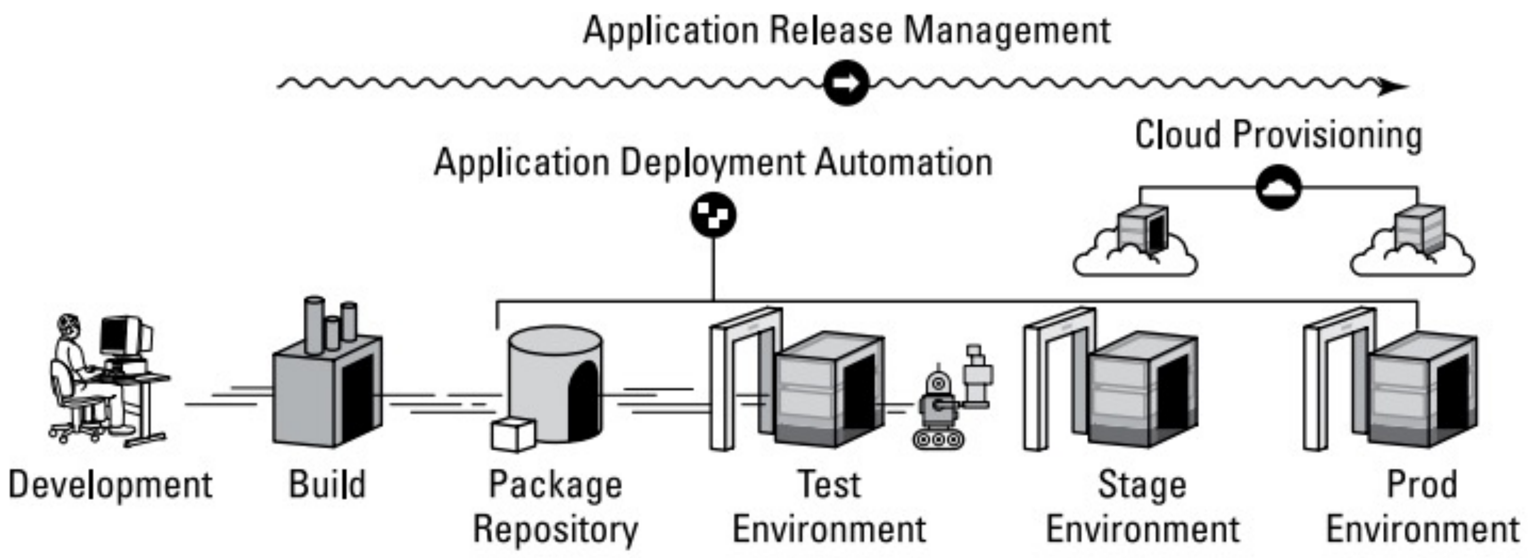
Continuous Delivery

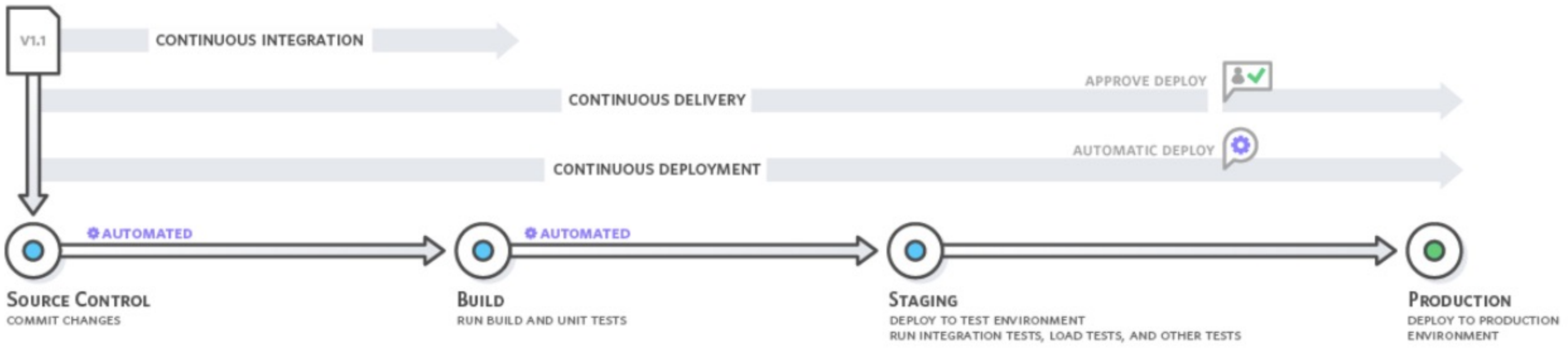
- **Continuous Delivery** is a software development practice where code changes are *automatically*:
  - Built,
  - Tested,
  - **Prepared** for a release to production.
- It expands upon continuous integration by deploying all code changes to a **testing** environment **and/or** a **production** environment after the build stage.
- When continuous delivery is **implemented properly**, developers will **always** have a **deployment-ready** build artifact that has passed through a standardized test process.

Source: https://jaxenter.com/how-to-move-from-ci-to-cd-with-jenkins-workflow-128135.html



Continuous Delivery

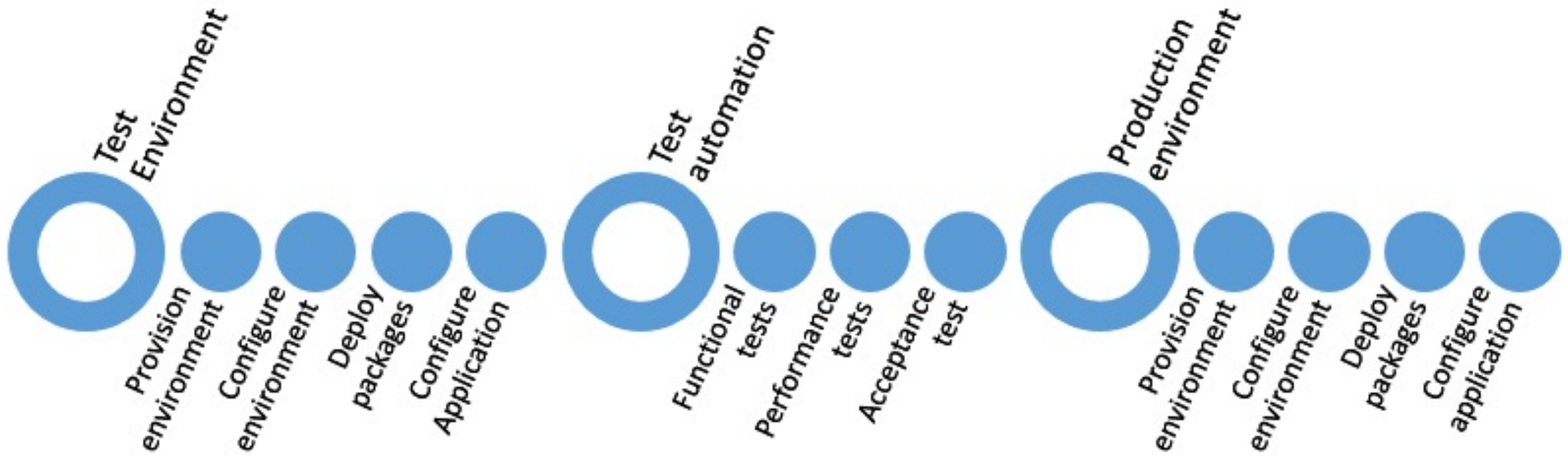


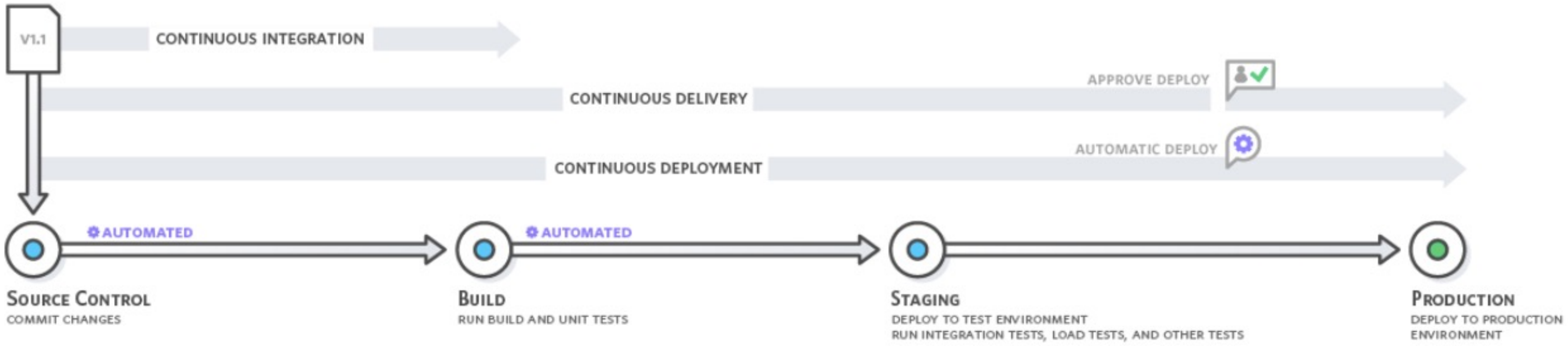


Continuous Deployment

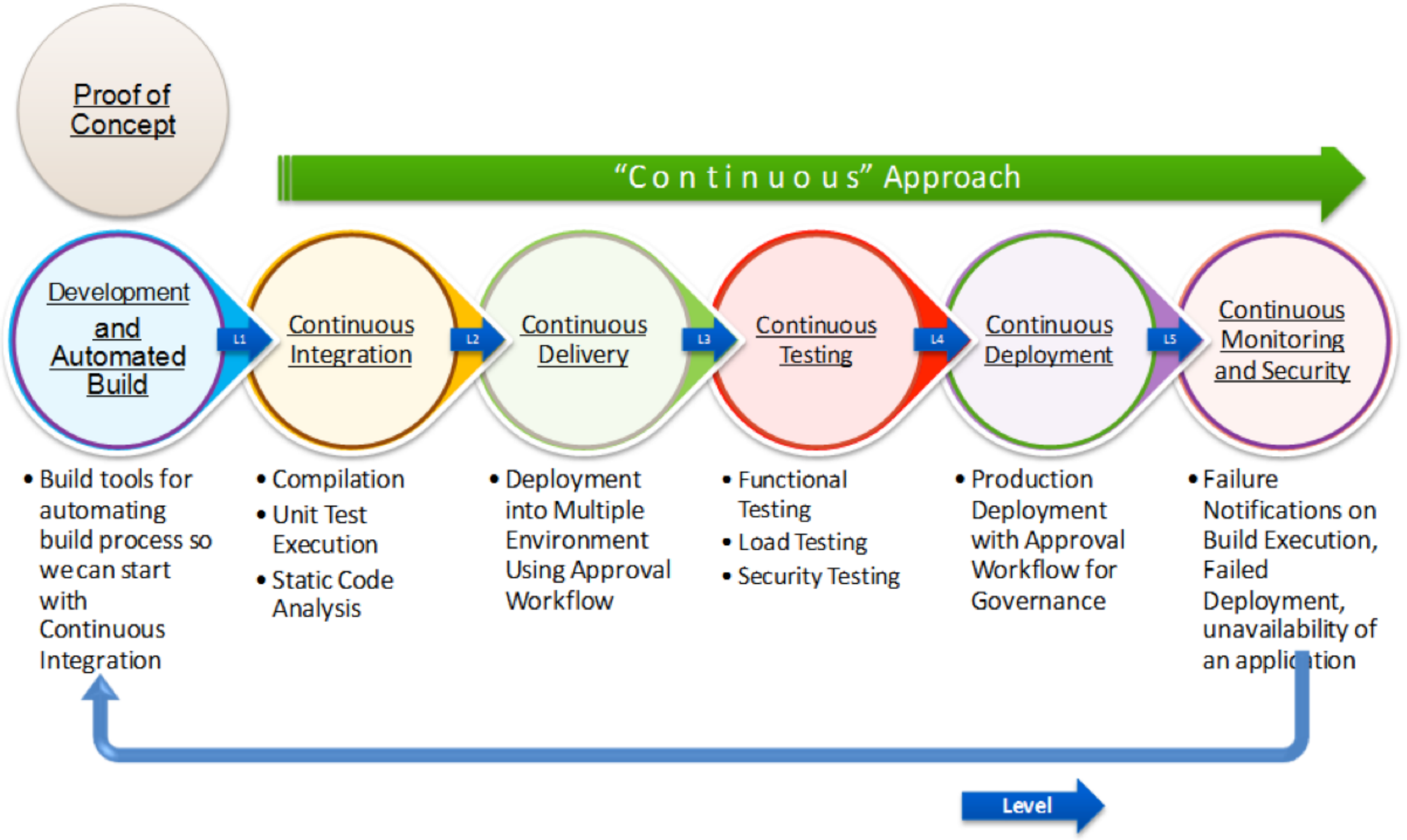
- **Continuous Deployment** refers to the capability to deploy applications and services to pre-production and production environments through automation.
- This means:
  - Provisioning and configuring an environment.
  - Deploying and configuring an application on top of it.
- This is normally done after conducting multiple validations (functional performance tests) on a **pre-production** environment.
  - Provision and configure the **production** environment.
  - An application is deployed to production environments through **automation.**

Continuous Deployment





The “Continuous” mantra



- Build tools for automating build process so we can start with Continuous Integration

- Compilation
- Unit Test Execution
- Static Code Analysis

- Deployment into Multiple Environment Using Approval Workflow

- Functional Testing
- Load Testing
- Security Testing

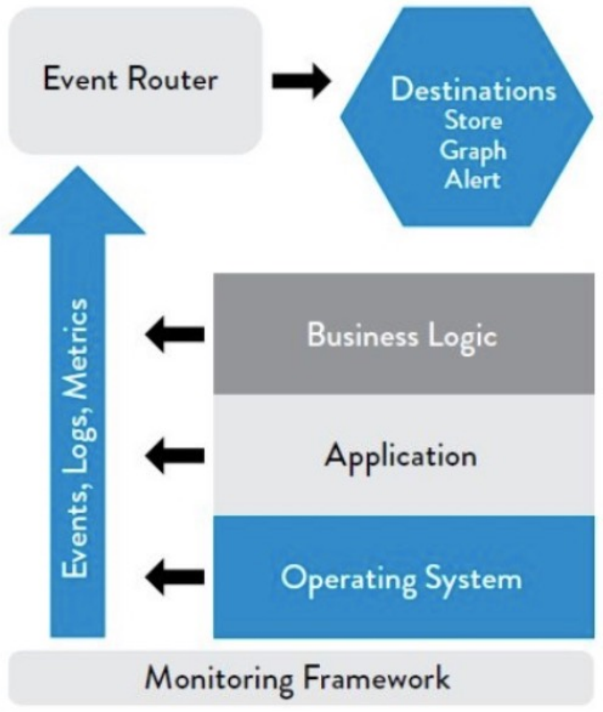
- Production Deployment with Approval Workflow for Governance

- Failure Notifications on Build Execution, Failed Deployment, unavailability of an application

Remember the DevOps motto:
“Release early, release often”

Continuous Learning

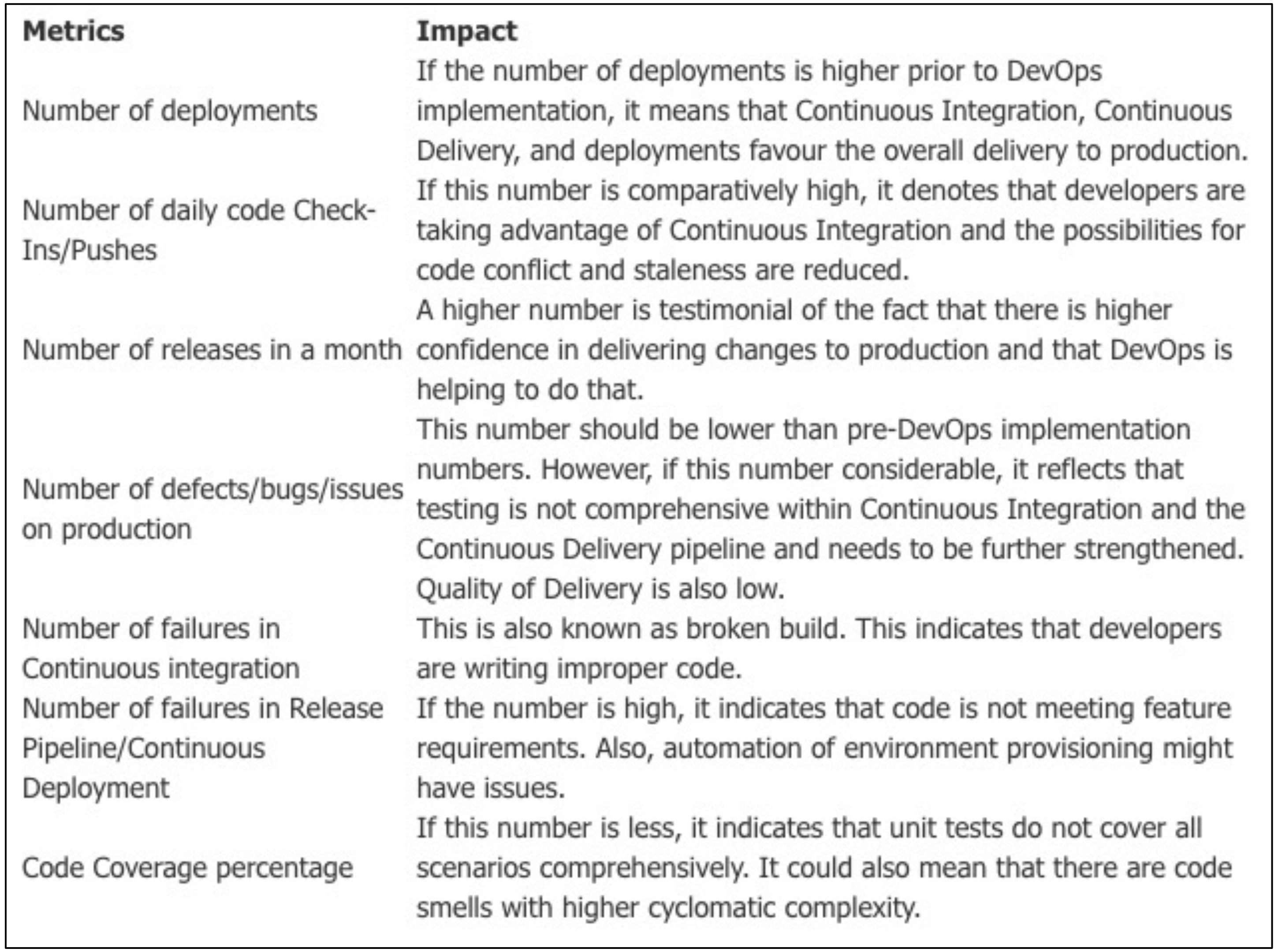
- The benefits of DevOps will not last for long if a **continuous improvement and feedback** principle is not in place.
  - This means to have real-time feedback about the application's behavior.
- Applications should therefore be built with:
  - Monitoring;
  - Auditing;
  - Telemetry in mind.



Continuous Monitoring

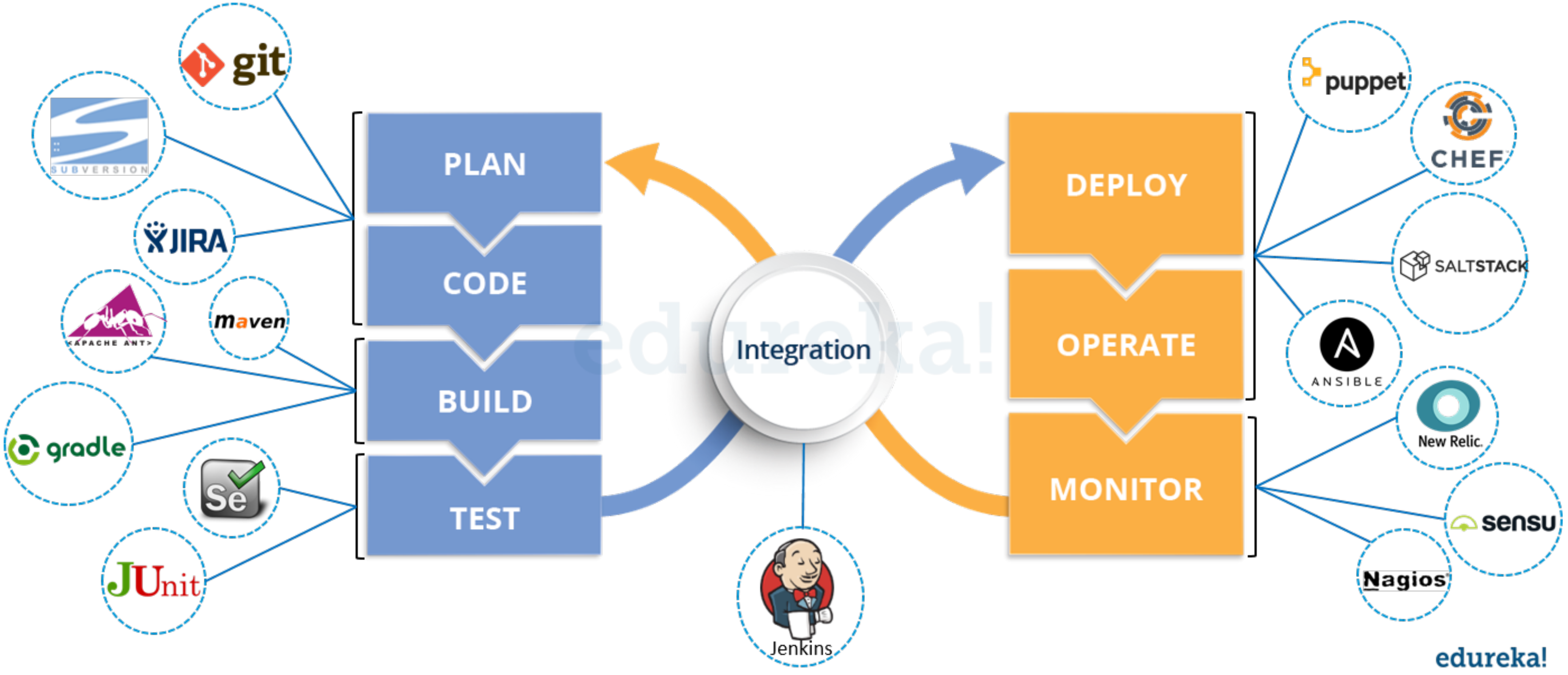
- **Monitoring** starts in the development phase.
  - The same tools that monitor the production environment can be employed in development to spot performance problems *before* they hit production.
- Two kinds of monitoring are required for DevOps:
  - Server monitoring.
  - Application performance monitoring.
- This means **measuring DevOps effectiveness:**
  - **Monitoring, audit and collection of metrics** should be developed and deployed.
  - There should be **a regular baselining of data** for effective comparison. Metrics should be **captured over a period** and then compared with the baseline.

Examples of monitoring metrics



Metrics	Impact
Number of deployments	If the number of deployments is higher prior to DevOps implementation, it means that Continuous Integration, Continuous Delivery, and deployments favour the overall delivery to production.
Number of daily code Check-Ins/Pushes	If this number is comparatively high, it denotes that developers are taking advantage of Continuous Integration and the possibilities for code conflict and staleness are reduced.
Number of releases in a month	A higher number is testimonial of the fact that there is higher confidence in delivering changes to production and that DevOps is helping to do that.
Number of defects/bugs/issues on production	This number should be lower than pre-DevOps implementation numbers. However, if this number considerable, it reflects that testing is not comprehensive within Continuous Integration and the Continuous Delivery pipeline and needs to be further strengthened. Quality of Delivery is also low.
Number of failures in Continuous integration	This is also known as broken build. This indicates that developers are writing improper code.
Number of failures in Release Pipeline/Continuous Deployment	If the number is high, it indicates that code is not meeting feature requirements. Also, automation of environment provisioning might have issues.
Code Coverage percentage	If this number is less, it indicates that unit tests do not cover all scenarios comprehensively. It could also mean that there are code smells with higher cyclomatic complexity.

The DevOps tool chain



Docker containers, microservices and orchestration

- **Docker containers** help to easily create and share applications that are – as the name says – self-contained.
- On the other hand, we just saw that **microservice architectures** are based on the composition of many independent (but communicating) services.
- **Combining these two points**, containers can greatly help with the creation of a microservice architecture. For example, through `docker-compose` you can easily create multiple containers linked together in **Application Stacks**.
- However, `docker-compose` <u>is limited to the composition of containers within a single host</u>. On the other hand, microservices are often deployed across multiple hosts.
- We therefore need to know how to effectively *orchestrate* many containers across **multiple, distributed hosts**. This is called **container orchestration**.

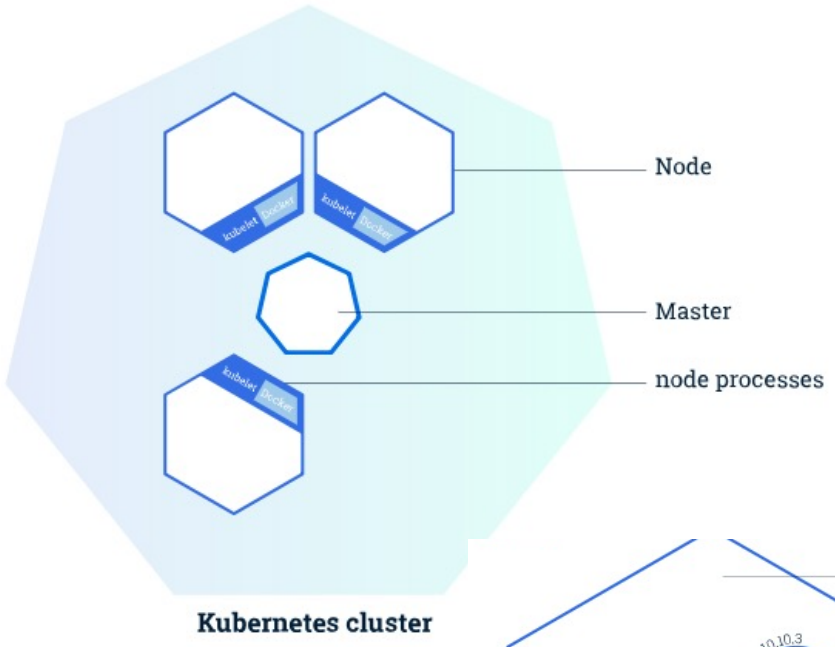
Kubernetes

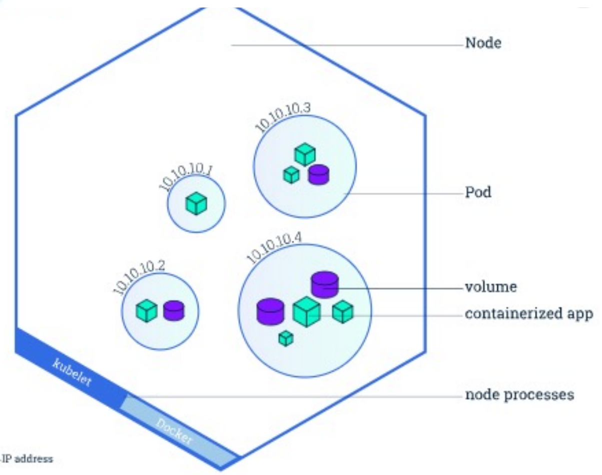
- Probably the most famous container orchestration toolset in use today is **Kubernetes**, or K8s (https://kubernetes.io/).
- Kubernetes [*] was initially developed at Google to scale container applications over a Google-scale infrastructure.
- There will be several hands-on exercises on Kubernetes. Let's quickly cover its main concepts here in one slide.

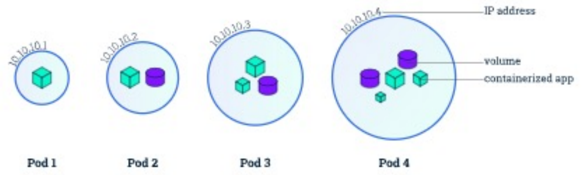
[*] Kubernetes: κυβερνήτης, Greek for "helmsman" or "pilot" or "governor" (https://en.wikipedia.org/wiki/Kubernetes)

Kubernetes clusters

- A Kubernetes *cluster* consists of two types of resources:
  - One or more **Masters** coordinate the cluster
  - **Nodes** are the workers that run containerized applications
- The **Master** is responsible for managing the cluster.
  - It coordinates all activities in the cluster, such as scheduling applications, maintaining applications' desired state, scaling applications and rolling out new updates.
- A **Node** is a VM or a physical computer that runs containerized applications by special processes called **pods**.



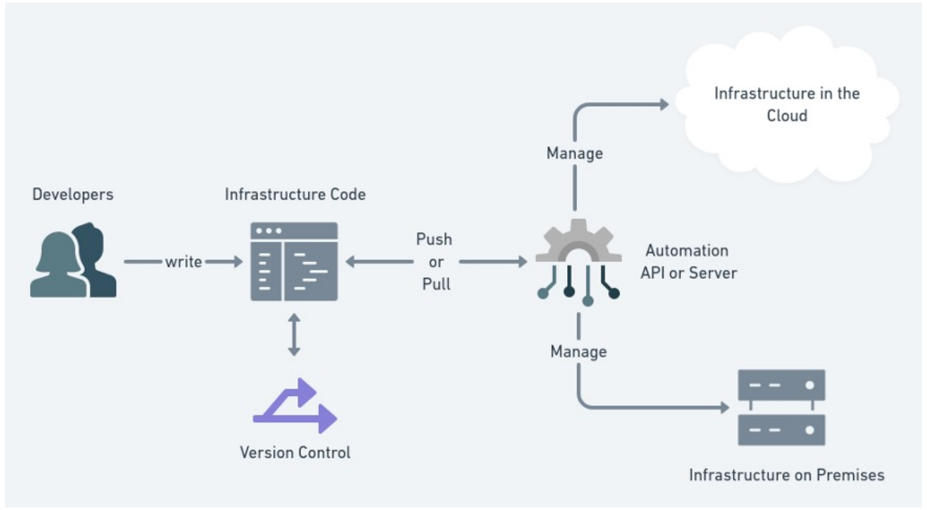




Not covered in detail in this course

Infrastructure as Code

- With the idea of **Infrastructure as Code (IaC)**, instead of manually *creating or provisioning the infrastructure we need* for our applications (e.g., install and configure virtual machines, disk volumes, clusters of servers, etc.), we **define what we want** through machine-readable definition files.
  - IaC assumes that "**Complexity kills Productivity**": it therefore aims to simplify how you can realize complex infrastructures and set-ups, without having you to learn infrastructural details.
- With IaC, all the specifications for the physical or virtual infrastructure that we want to generate are explicitly **described through configuration files**, often stored in **high-level templates**.
  - IaC focuses on "what we need", rather than on "how to create an infrastructure".



https://blog.stackpath.com/infrastructure-as-code-explainer/

Not covered in detail in this course

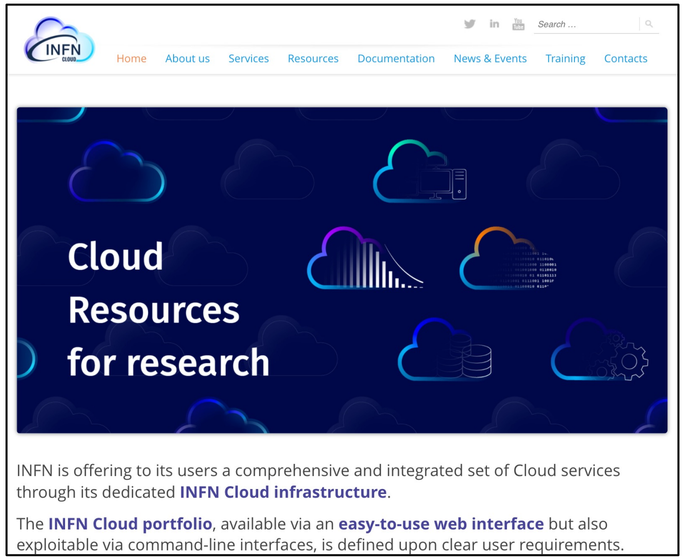
Template-based orchestration

- IaC makes use of **templating mechanisms** to describe and provision ("orchestrate") resources needed by an application in a fully distributed Cloud infrastructure.
- This extends what we said about Kubernetes to cover *any requirements* that your applications might have, automatizing your app deployments in a Cloud.
- The templating concept also links to the idea of **reusing and extending know-how**, rather than reinventing the wheel every time.
- IaC is therefore also part of a modern software development process, and it connects tightly with the DevOps concepts mentioned earlier.

Not covered in detail in this course

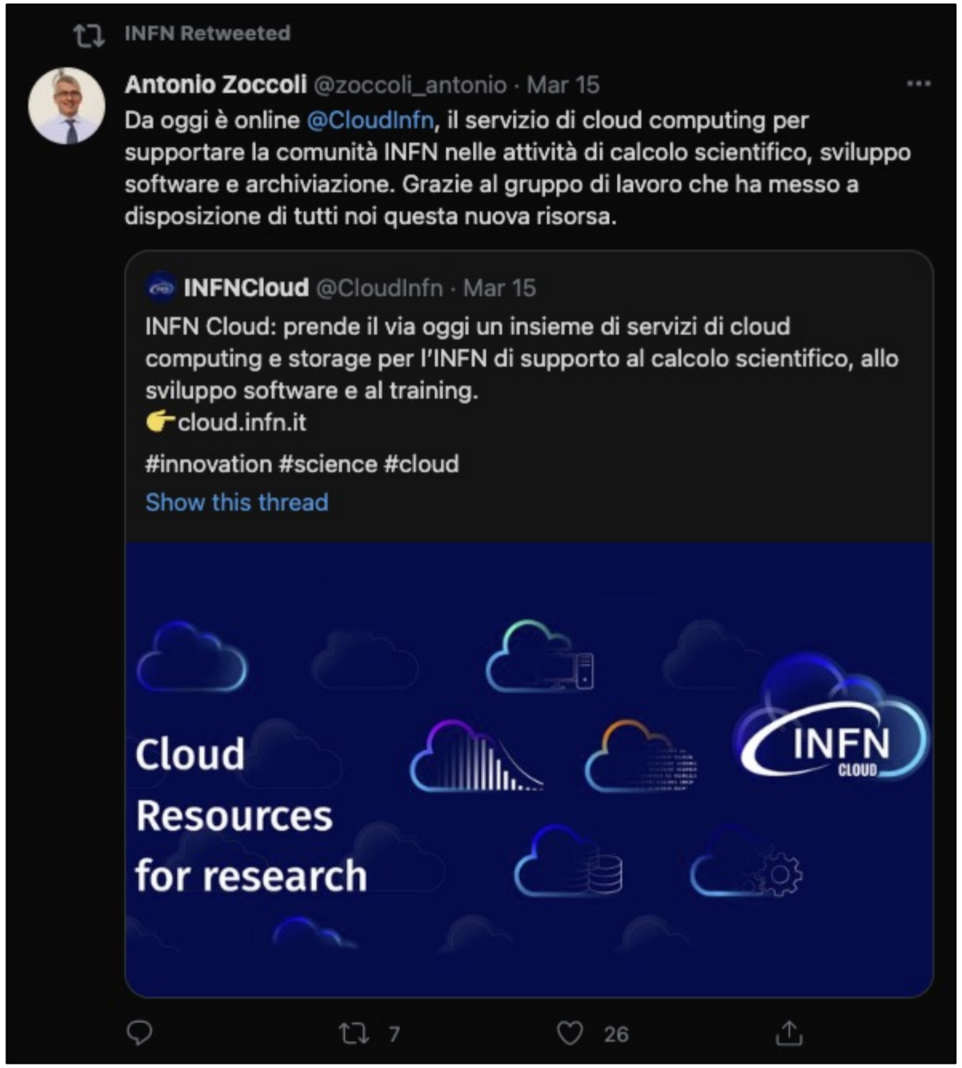
An example: INFN Cloud

- Since 15/3/2021 it is possible for INFN personnel and associates to require access to resources available on the INFN Cloud infrastructure. For more info: https://www.cloud.infn.it



INFN is offering to its users a comprehensive and integrated set of Cloud services through its dedicated **INFN Cloud infrastructure**.

The **INFN Cloud portfolio**, available via an **easy-to-use web interface** but also exploitable via command-line interfaces, is defined upon clear user requirements.



INFN Retweeted

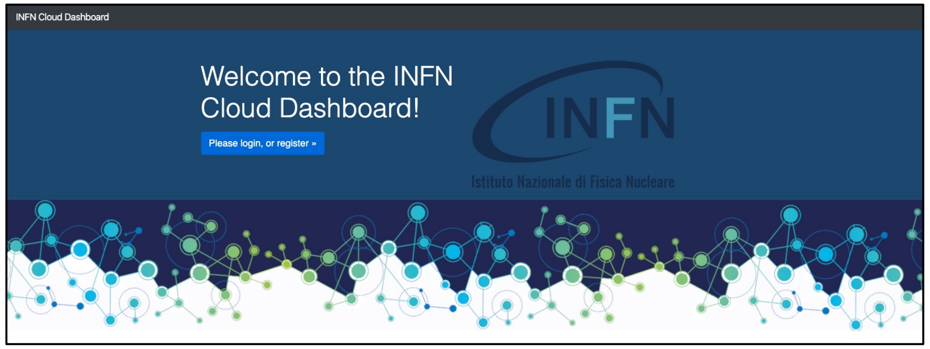
Antonio Zoccoli @zoccoli_antonio · Mar 15

Da oggi è online @CloudInfn, il servizio di cloud computing per supportare la comunità INFN nelle attività di calcolo scientifico, sviluppo software e archiviazione. Grazie al gruppo di lavoro che ha messo a disposizione di tutti noi questa nuova risorsa.

INFNCloud @CloudInfn · Mar 15

INFN Cloud: prende il via oggi un insieme di servizi di cloud computing e storage per l'INFN di supporto al calcolo scientifico, allo sviluppo software e al training.
👉cloud.infn.it
#innovation #science #cloud
Show this thread

The INFN Cloud dashboard

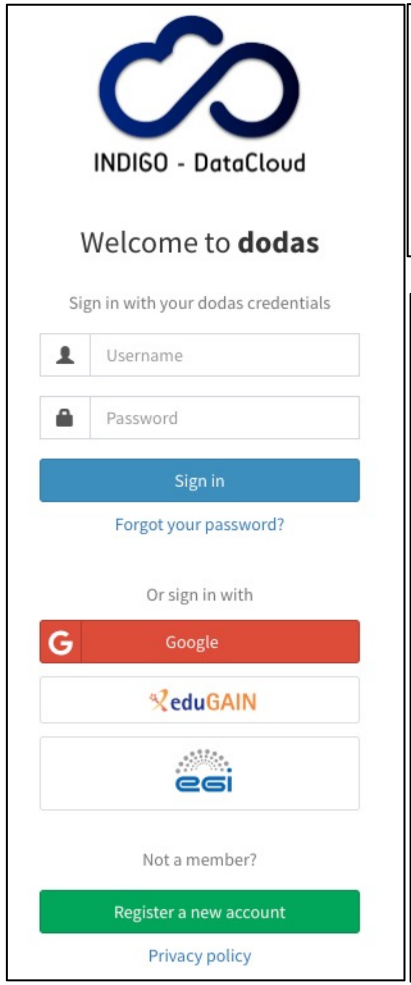

INFN Cloud Dashboard

Welcome to the INFN Cloud Dashboard!

Please login, or register »

Transparent, multi-site **federation** for users of Cloud resources belonging to INFN and/or to other Cloud providers (private or public)

IaC Templates



Welcome to **dodas**

Sign in with your dodas credentials

Username

Password

Sign in

Forgot your password?

Or sign in with

Google

eduGAIN

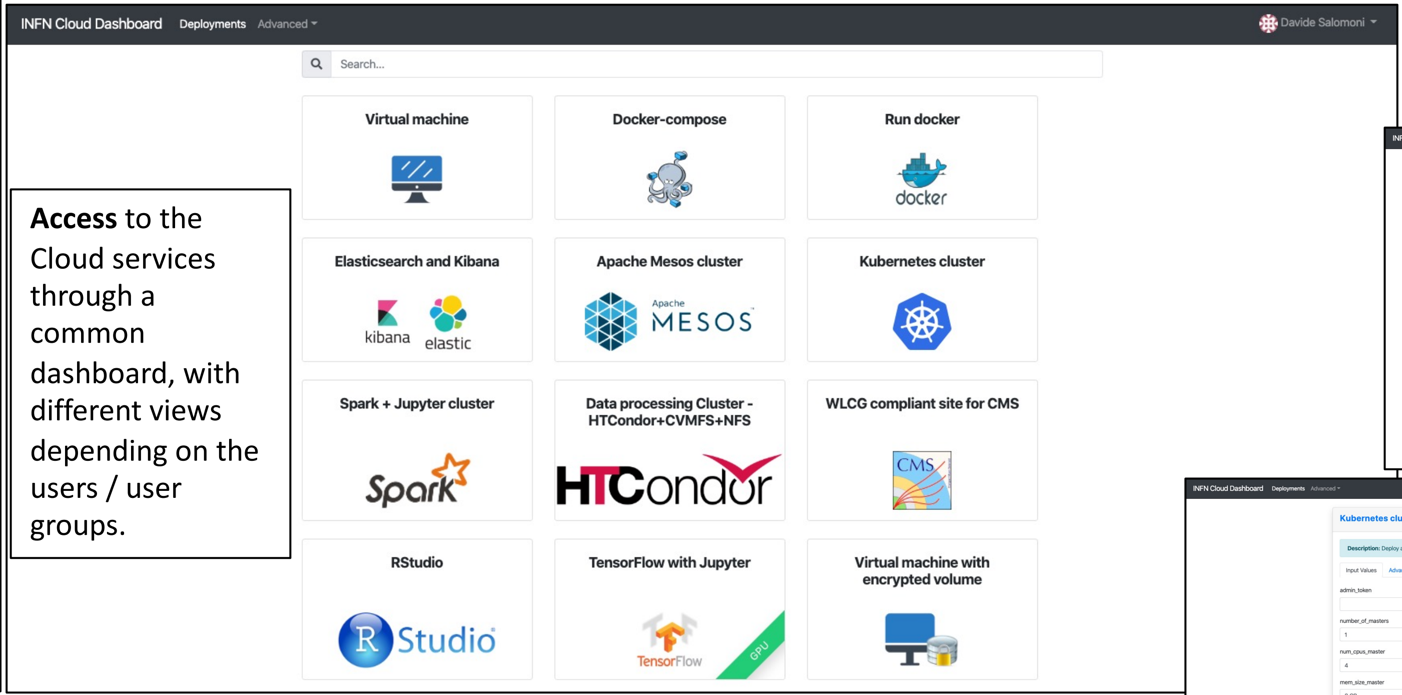
Not a member?

Register a new account

Privacy policy

Authentication *can* be enabled for:

- Local username/password
- Google accounts
- eduGAIN (e.g. Universities, research centers, etc.)
- Other OIDC providers

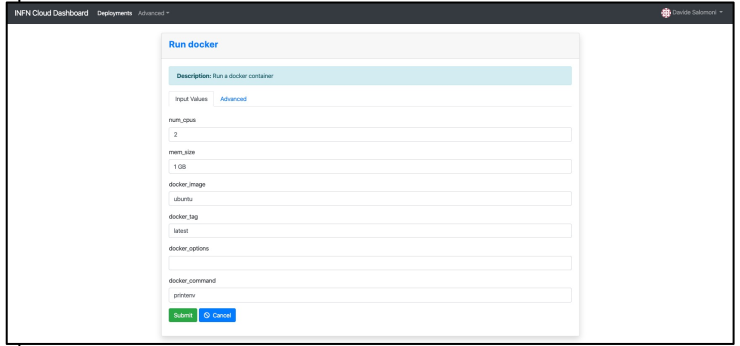

INFN Cloud Dashboard Deployments Advanced ▾ Davide Salomoni ▾

Search...

Virtual machine	Docker-compose	Run docker
Elasticsearch and Kibana	Apache Mesos cluster	Kubernetes cluster
Spark + Jupyter cluster	Data processing Cluster - HTCondor+CVMFS+NFS	WLCG compliant site for CMS
RStudio	TensorFlow with Jupyter	Virtual machine with encrypted volume

Access to the Cloud services through a common dashboard, with different views depending on the users / user groups.

Composed, high-level services easily customizable and configurable directly by users


INFN Cloud Dashboard Deployments Advanced ▾ Davide Salomoni ▾

Run docker

Description: Run a docker container

Input Values Advanced

num_cpus: 2

mem_size: 1 GB

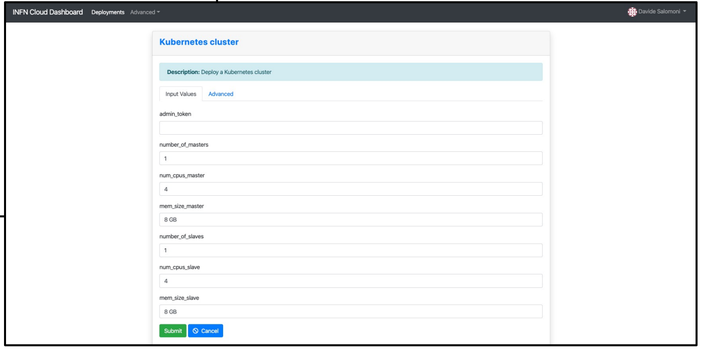
docker_image: ubuntu

docker_tag: latest

docker_options:

docker_command: printenv

Submit Cancel


INFN Cloud Dashboard Deployments Advanced ▾ Davide Salomoni ▾

Kubernetes cluster

Description: Deploy a Kubernetes cluster

Input Values Advanced

admin_token:

number_of_masters: 1

num_cpus_master: 4

mem_size_master: 8 GB

number_of_slaves: 1

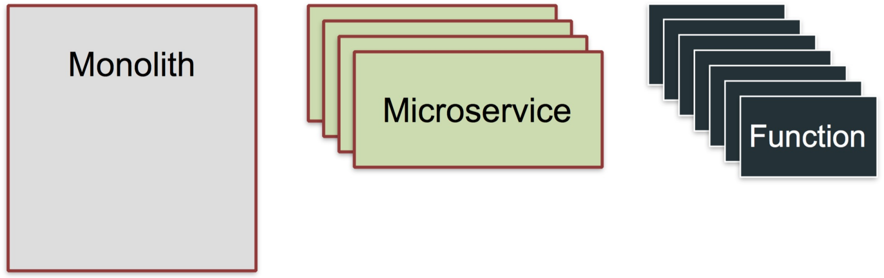
num_cpus_slave: 4

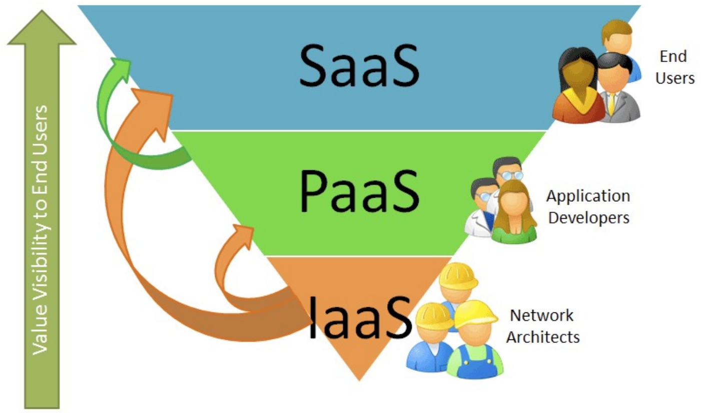
mem_size_slave: 8 GB

Submit Cancel

The whole application landscape so far

- Let’s recap the evolutions:
  - We started from applications running on single data centers running on **physical hardware**…
  - … then we introduced **virtualization** to optimize resource usage…
  - … then we moved to the Cloud to instantiate basic infrastructural **resources** (for instance VMs)…
  - … then we introduced **Docker containers** to improve efficiency and portability…
  - … then we exploited Docker, breaking down monolithic applications into stateless applications based on **microservices through Container orchestration…**
  - … then we said we can use **DevOps** to manage the entire application lifecycle.
- With **IaC** (not covered in detail in this course) we are now abstracting from the infrastructural layers even more, expressing application requirements in high-level templates, making use of pluggable PaaS components.

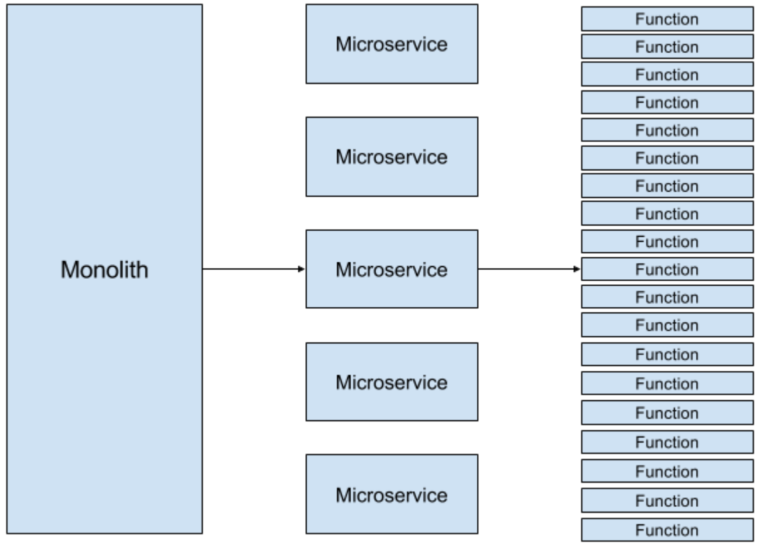




Not covered in detail in this course

Serverless technologies

- With **serverless technologies**, we perform another step toward automating and facilitating writing and using applications and Cloud resources.
    - Remember that **what eventually matters are the applications, not the infrastructure.**
- With **serverless**, a Cloud provider is responsible for executing a piece of code, written by you, by **dynamically finding and allocating the resources needed by the code**.
- In serverless, your code is typically structured around a set of stateless functions. Thus, serverless computing is also called **Functions as a Service, or FaaS**. We won't cover FaaS in detail in this course, but it is an important concept.
    - The running of the serverless functions can be **triggered** by some conditions, such as for example database events, file uploads, scheduled events, various alerts, etc.
    - Structuring an app around stateless functions is consistent with the idea of **microservices** we have already seen. This time, however, we focus just on the app code, and deal as little as possible with resource provisioning and deployment.



https://www.iron.io/

Conclusions

- We have mentioned tools and concepts that deal with automatizing the writing, testing and deploying of modern applications in a Cloud-centric world. This has positive consequences also for the all-important topic of reproducibility.
- You will now spend the rest of the course exploring in some more details most of what it was said, also through several hands-on sessions.
- So, learn and have fun!
- For any questions, send me an email at davide@infn.it.

Thanks!