

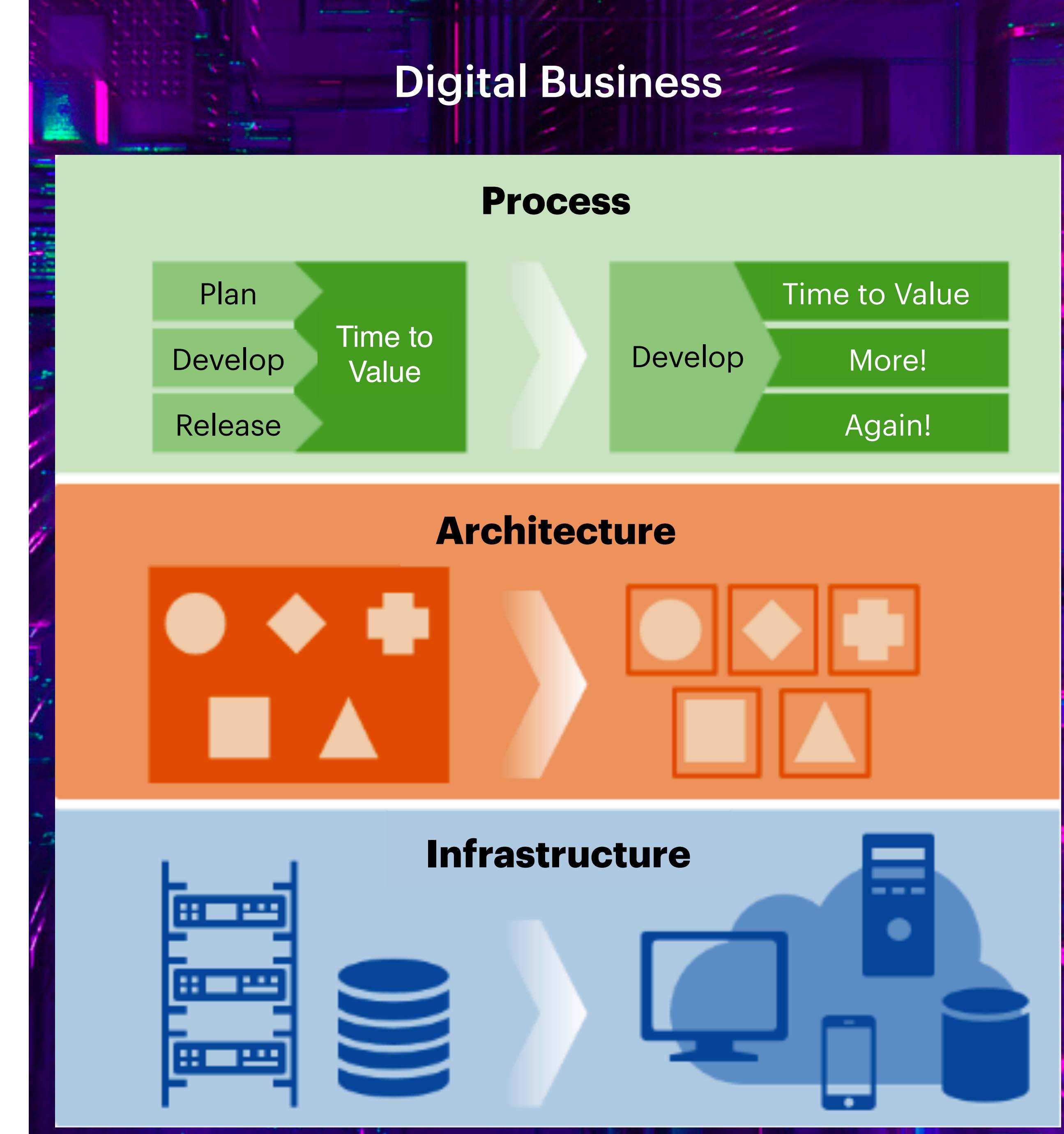
Smart Workflows

How to develop your application

sara.vallero@to.infn.it

A paradigm shift

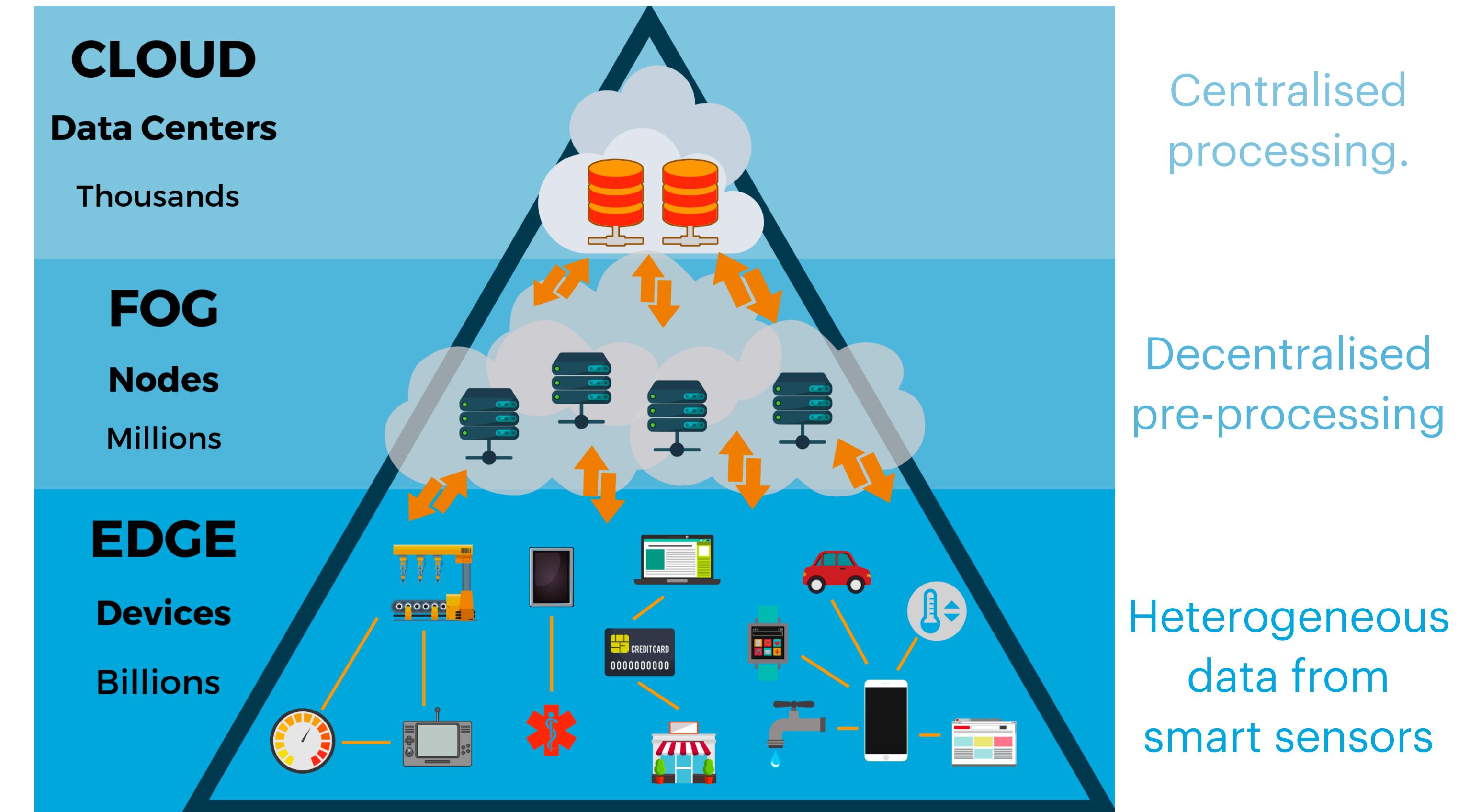
- In the last decades the approach to computing has changed, not only because of new technologies, but also in its philosophy.
- We often hear about:
 - DevOps approach and Continuous Integration (CI)
 - Microservices
 - Internet of Things (IoT) and the Digital Continuum



Infrastructure

Internet of Things (IoT) and the Digital Continuum

- We used to have servers and storage disks...
- Now, we still have servers and storage disks, but we also talk about:
 - **IoT**, where you generate, store and process data on an amount of objects: computers, smartphones, sensors, the washing machine...
 - **Digital Continuum**: a design space that creates continuity between elements that exist separately, with the goal to transform data into information as soon as possible.



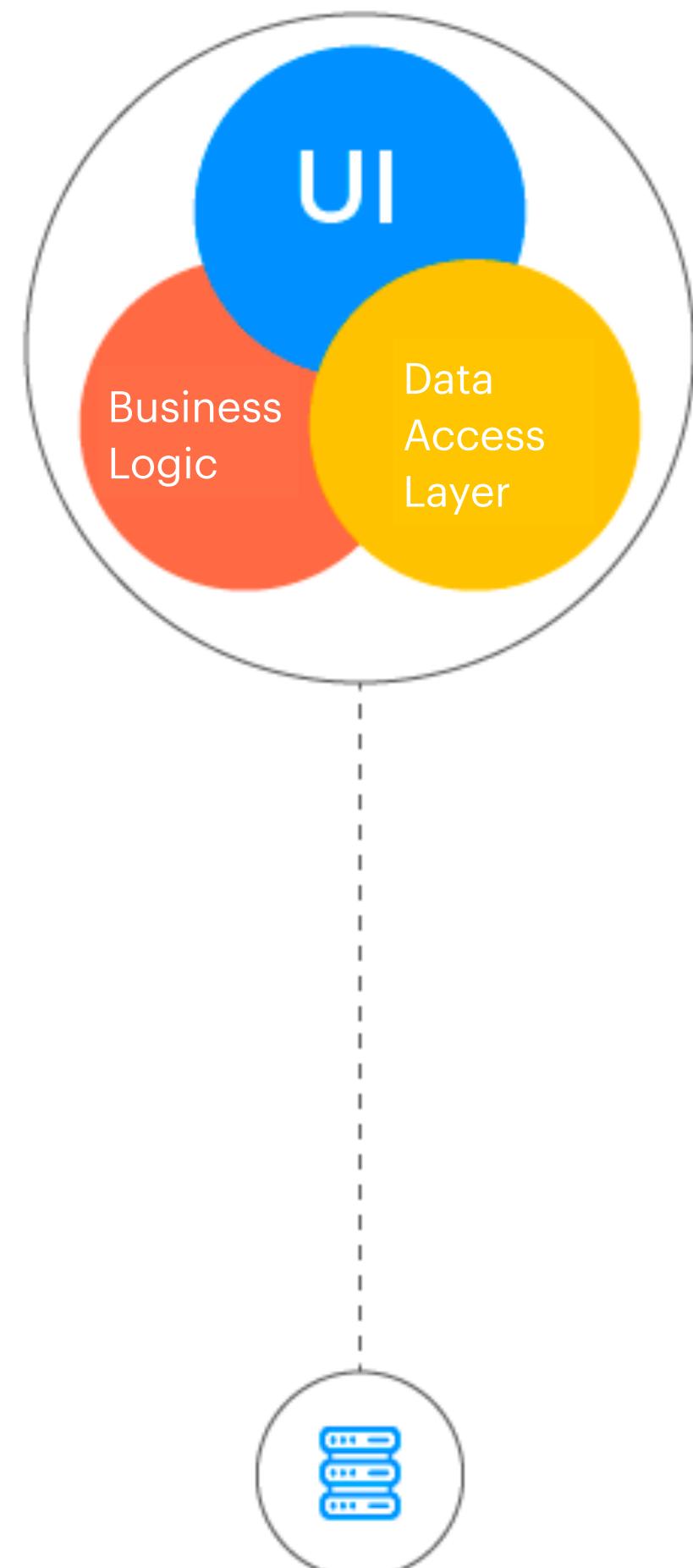
Architecture

Microservices

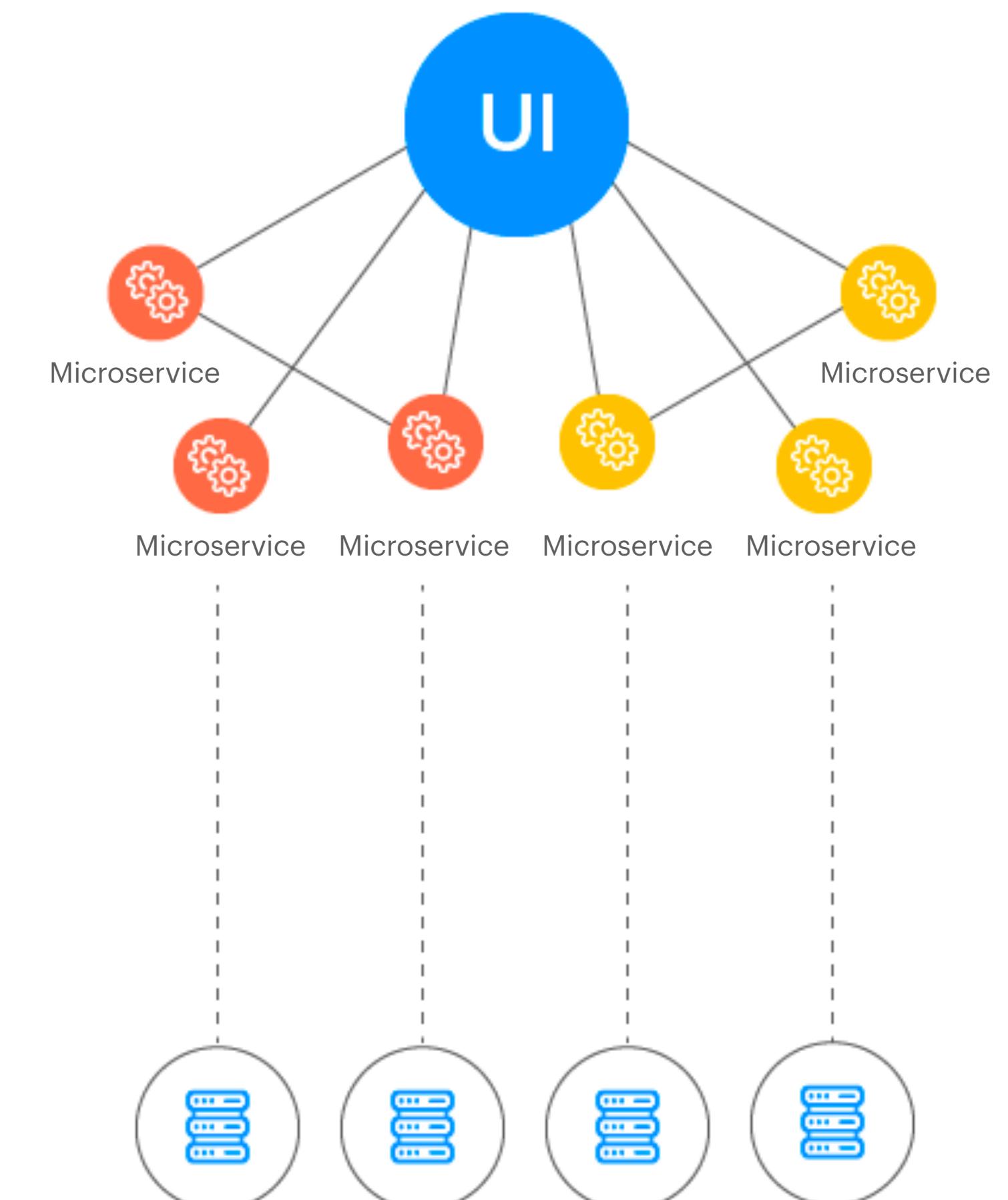
The way people structure a complex stack of computing applications has changed.

- **Monolithic architecture:** several “programs” installed on a server. Difficult to understand dependencies, do updates, re-use work done.
- **Microservices:** Service Oriented Architecture (SOA) that structures an application as a collection of loosely coupled services.
 - Makes the application easier to understand, develop and test.
 - Parallelises development.
 - Orchestration layer needed to coordinate operations.

Monolithic Architecture



Microservice Architecture



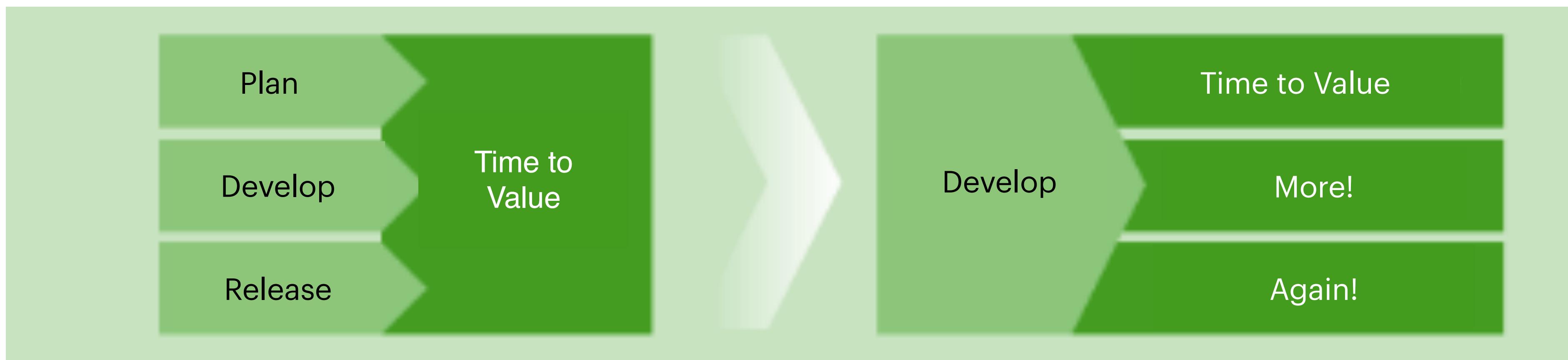
Process

Time to Value

Time to value: from the original idea to have a product available for other people to use or to produce a scientific result.

Before: I have an idea, I write code, I release.

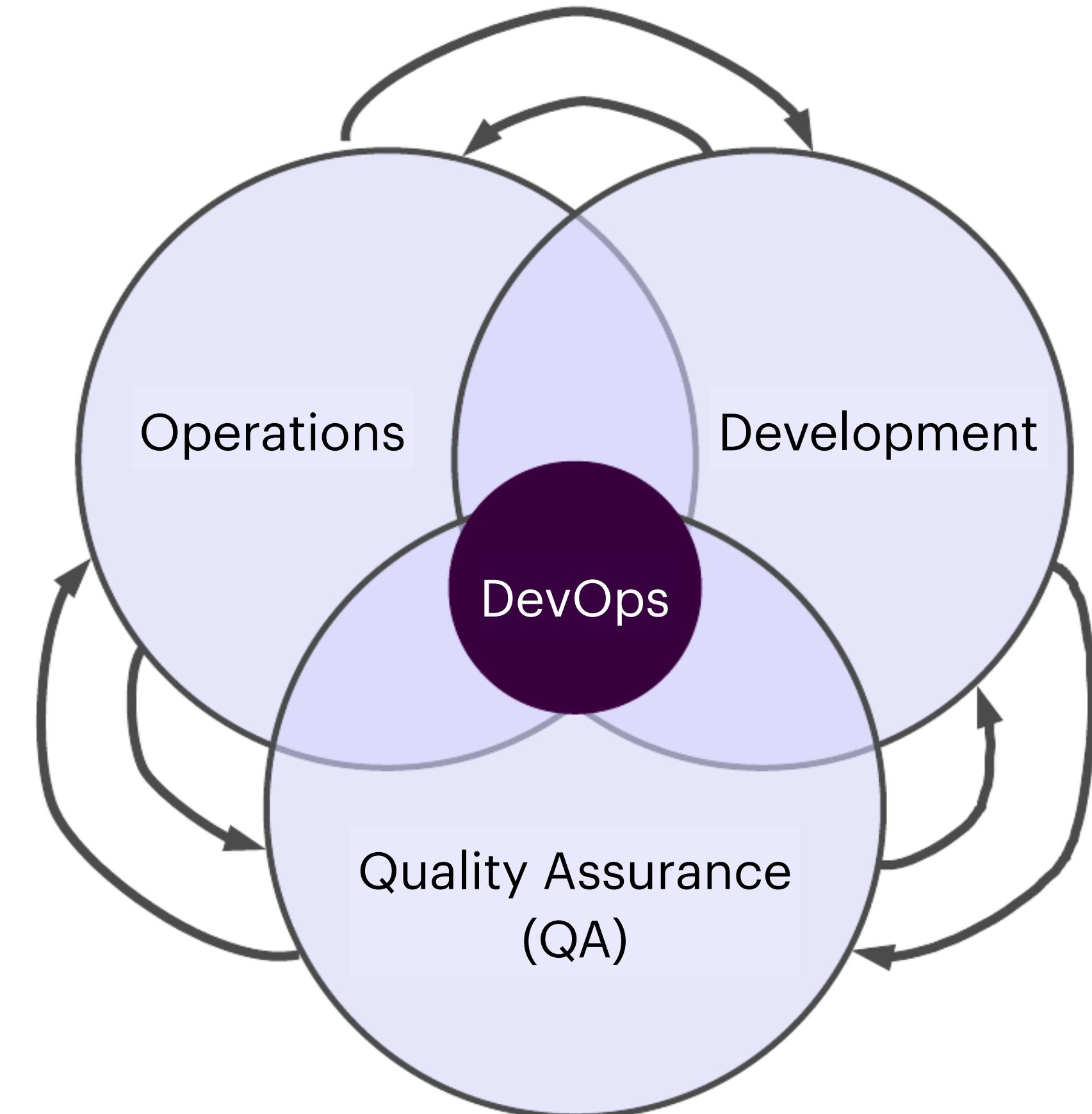
Now: keep time to value as short as possible. Make a change available whenever it is ready and tested, with the possibility to rollback.



Process

DevOps approach

- DevOps (Development and Operations) is a philosophy and practice focused on agility, collaboration, and automation within IT and development team processes.
- The goal is to **bridge the gap between IT operations and development** to improve communication and collaboration, create more seamless processes, and align strategy and objectives for faster and more efficient delivery.
- **DevOps philosophy principles:**
 - Automation
 - Iteration
 - Self-service
 - Continuous improvement
 - Collaboration
 - Continuous testing

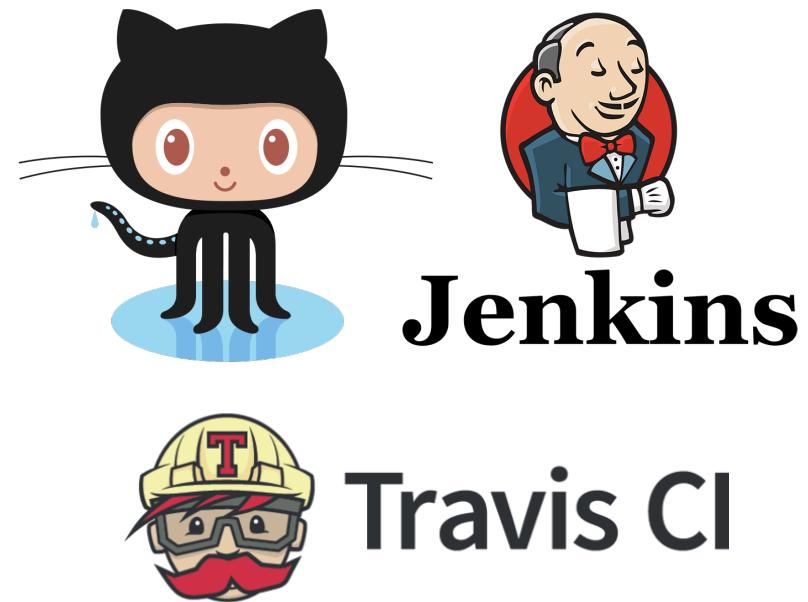


Process

Continuous Integration, Delivery and Deployment

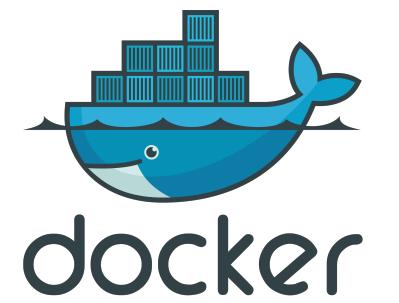
Continuous Integration (CI):

- a software development practice in which developers regularly merge their code changes into a shared repository where those updates are automatically tested.
- ensures that the most up-to-date and validated code is always readily available to developers.



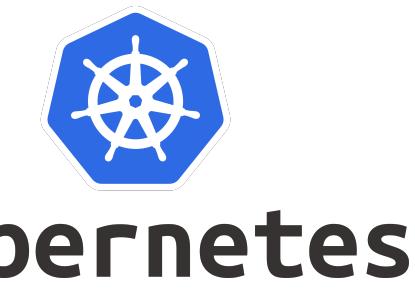
Continuous Delivery:

- code changes are automatically built, tested, and packaged for release into production.



Continuous Deployment:

- every validated change is automatically released to users.



Continuous Monitoring and Feedback.



Git:

- A Version Control System:
- You make constant changes to the code, releasing new versions
- Keep the revisions straight, storing modifications in a central repository
- Make it easy to collaborate: download/upload new revisions
- Efficient storage of file changes
- File integrity checks

The Hub (GitHub):

A Web Platform to:

- Store your projects
- Collaborate
- Host your project's documentation
- Implement Continuous Development
- Automatic builds
- WebHooks

My repos

farmcontroller
Kube operator to enforce dynamic quotas on farm.
Go Updated 7 days ago

controller-runtime
Forked from kubernetes-sigs/controller-runtime
Repo for the controller-runtime subproject of kubebuilder (sig-apimachinery)
Go 231 Apache License 2.0 Updated 21 days ago

MLCourse [Private]
Notebooks and stuff for the ML graduate course.
Jupyter Notebook Updated on Oct 3

Profile Summary

Sara Vallero
svallero
Edit profile
INFN and Università degli Studi
Torino svallero AT to.infn.it

farmcontroller Repository

Code Issues 0 Pull requests 0 Projects 0 Wiki Security Insights Settings

Kube operator to enforce dynamic quotas on farm.

Manage topics

9 commits 2 branches 0 releases 1 contributor

Branch: master New pull request

Switch branches/tags
Find or create a branch...
Branches Tags
master default
devel-farm-manager

Latest commit 53df899 7 days ago

Ready to be deployed on Kube 7 days ago

Ready to be deployed on Kube 7 days ago

Ready to be deployed on Kube 7 days ago

First commit My commits 15 days ago

Ready to be deployed on Kube 7 days ago

Ready to be deployed on Kube 7 days ago

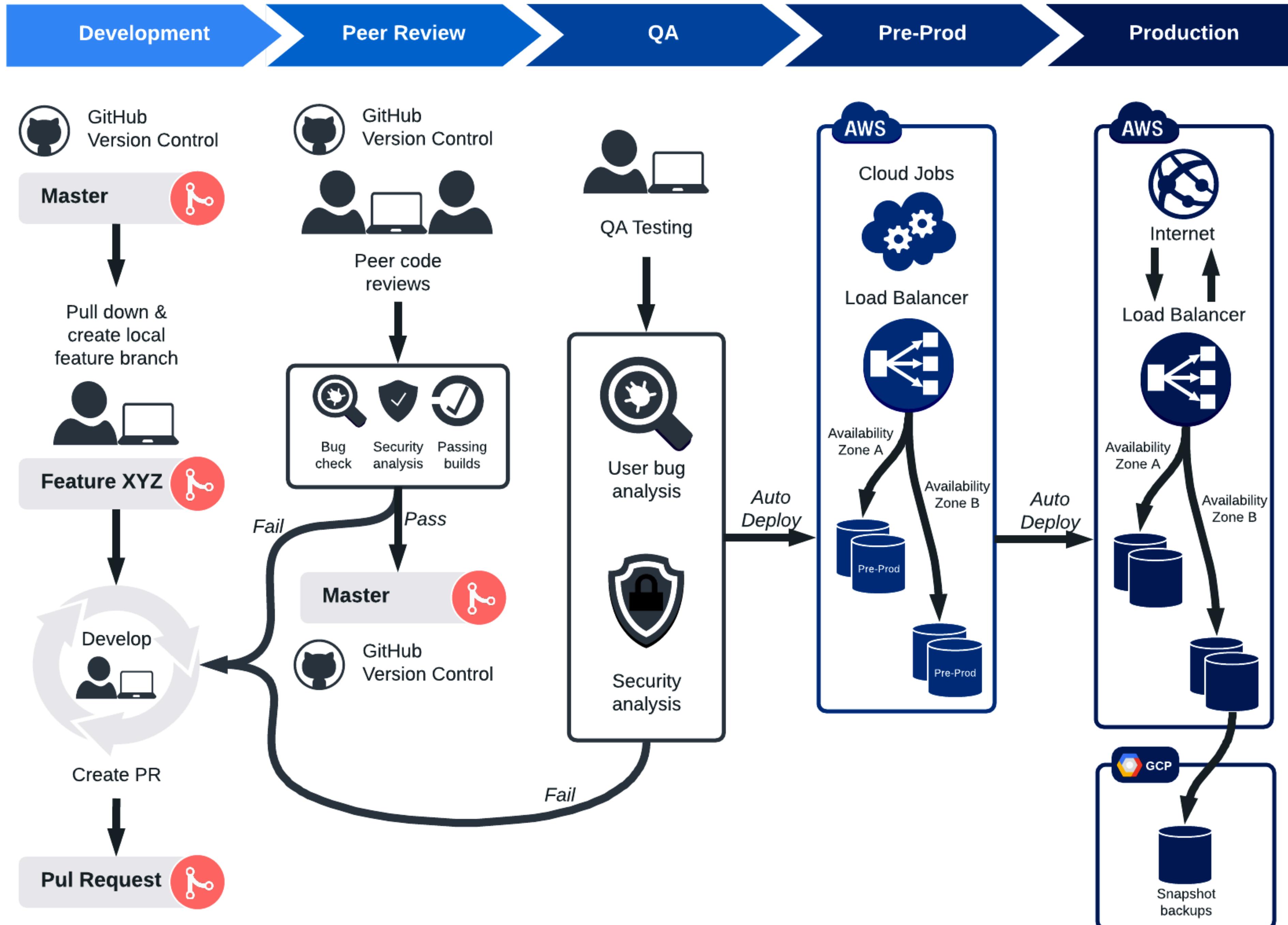
Ready to be deployed on Kube 7 days ago

.gitignore Dockerfile Dockerfile.build

DevOps workflow

Atomic commits:

implement a new feature, commit. If something goes wrong you can always go back. Each commit version is working with its own set of features.



The more sophisticated science becomes, the harder it is to communicate results. Papers today are longer than ever and full of jargon and symbols. They depend on chains of computer programs that generate data, and clean up data, and plot data, and run statistical models on data. These programs tend to be both so sloppily written and so central to the results that it's contributed to a replication crisis, or put another way, a failure of the paper to perform its most basic task: to **report what you've actually discovered, clearly enough that someone else can discover it for themselves.**

- James Somers



Reproducibility



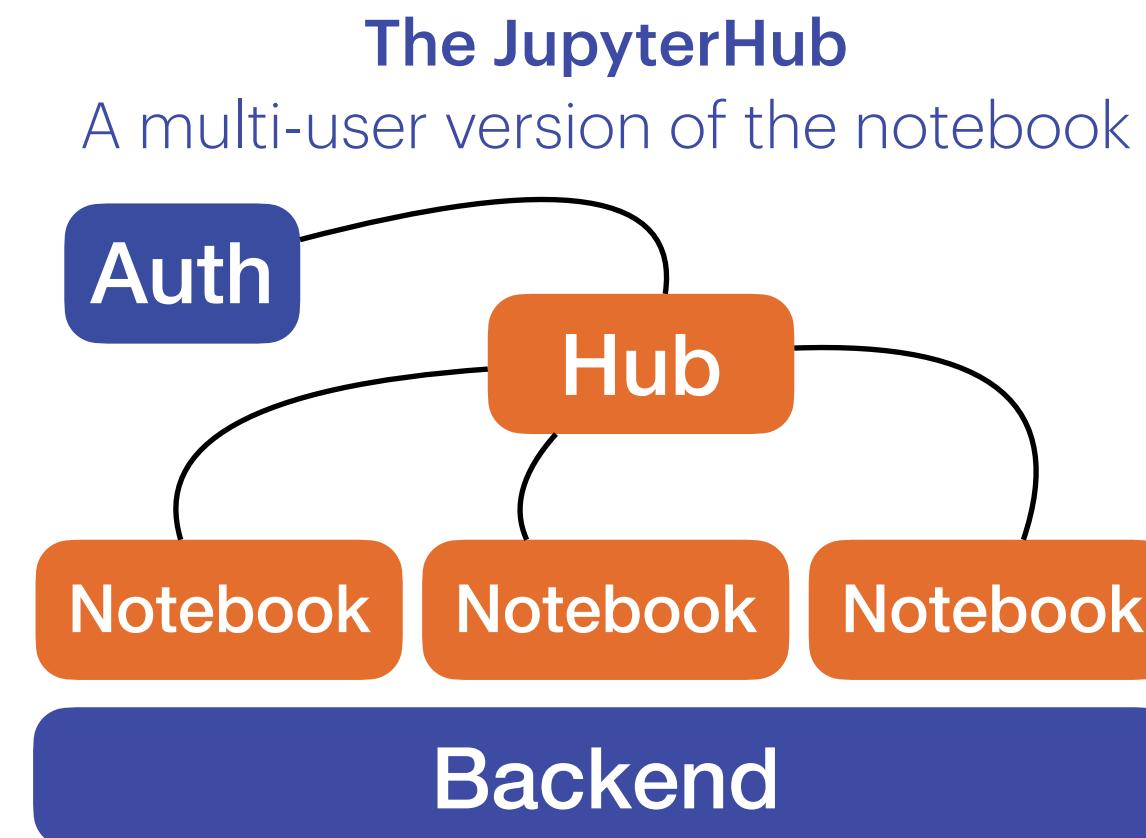
JupyterHub

The **Jupyter Notebook** is an open-source web application that allows you to interactively create and share documents that contain:

- live code
- equations
- visualisations
- narrative text

Uses include:

- data cleaning and transformation
- numerical simulation
- statistical modeling
- data visualisation
- machine learning



Multiple language support (kernels)

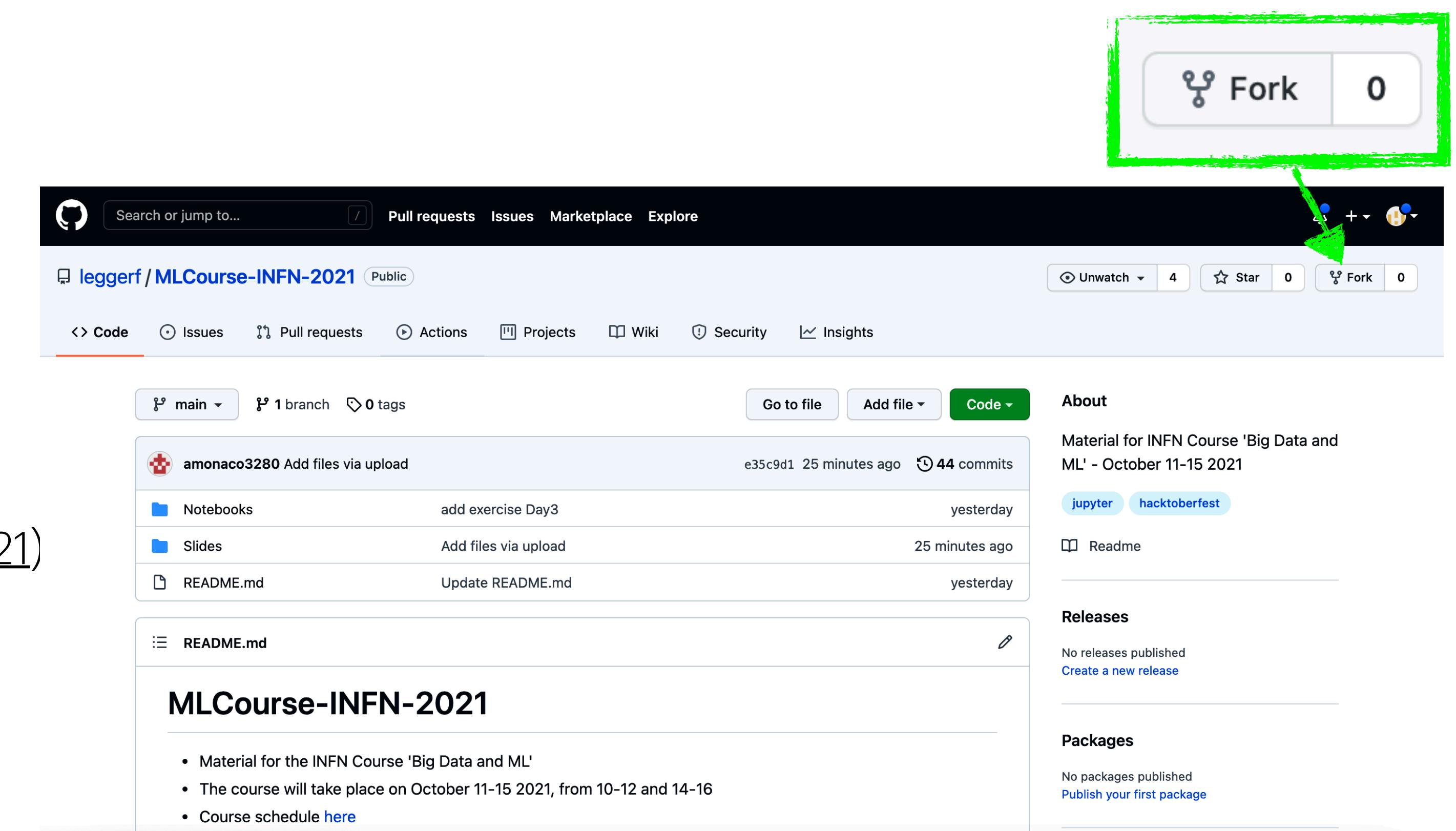
The image shows two screenshots of the JupyterHub interface. The top screenshot displays a notebook titled "Big data and ML - Hands-on Day 1" with sections for "course slides", "You'll learn", "Input data", "Visualization", and "Dataset description". A red arrow points from the text "Start the Spark Context with this white button" to a white rectangular button in the toolbar. The bottom screenshot shows a notebook titled "Exercise 2" with a code cell containing `#plot invariant mass distribution
plotSignalvsBg(df, 'm_bb')`. Below the code is a histogram titled "Distribution of m_bb" with two overlapping distributions: "signal" (blue) and "background" (orange). The x-axis is labeled "m_bb" and ranges from 0.0 to 3.0, while the y-axis is labeled "counts" and ranges from 0 to 6000.

A very simple (and useful) example of DevOps workflow

During the hands-on sessions, you will not have a persistent storage to save your Notebooks, but you can save them on GitHub!

STEP 1: Fork

- login to GitHub (<https://github.com/>)
- go to the course repository (<https://github.com/leggerf/MLCourse-INFN-2021>)
- Fork the repository in your private space



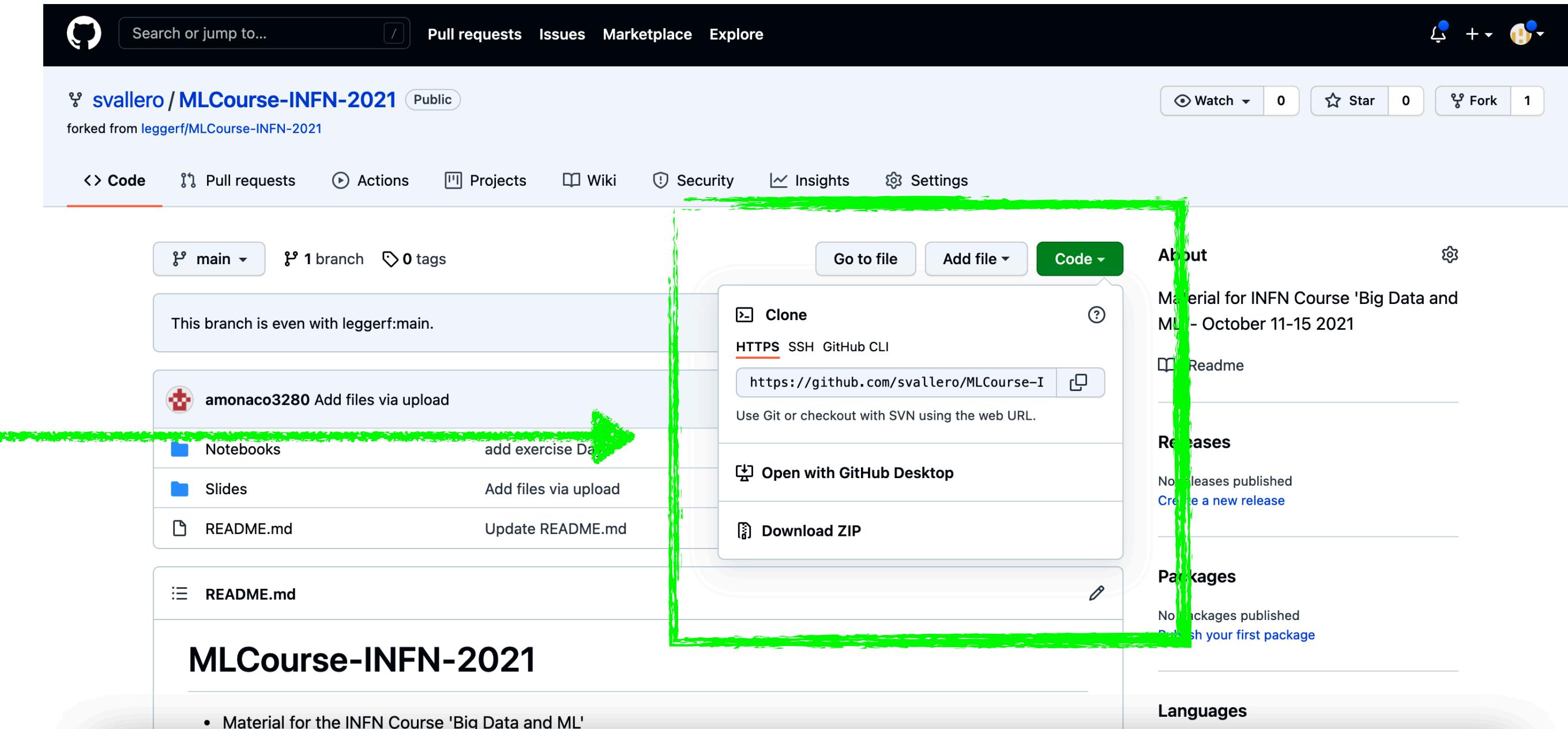
STEP 2: Clone

- You have been redirected to your private Fork of the repository
- Copy the Clone URL
- Clone your repository using the command line:

```
git clone https://github.com/yourusername/  
MLCourse-INFN-2021.git
```

(This will be executed in a *Jupyter Terminal* as explained in the upcoming hands-on session)

- You should now have a directory called:
MLCourse-INFN-2021



The directory is available in your JupyterHub space!



STEP 3: Status

- Browse the directory and start a Notebook.
- Make local modifications to the Notebooks (i.e. solve the exercises) and save them
- Now you can go back to the terminal and see what happened...

```
$ cd MLCourse-INFN-2021
```

```
$ git status
```

On branch main

Your branch is up to date with 'origin/main'.

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: Notebooks/Day1/inputForML.ipynb



This is the file you have changed
and you want to commit .

Untracked files:

(use "git add <file>..." to include in what will be committed)

Notebooks/Day1/.ipynb_checkpoints/

Notebooks/Day1/Higgs100k.parquet

Notebooks/Day1/__pycache__/



These are the files that will not
go in the repository unless you
explicitly add them (and you
don't want to do that).

no changes added to commit (use "git add" and/or "git commit -a")

STEP 4: Commit

- Add your changes
- Commit your changes
(remember the atomic commits)
- Check the status
- So far you made only local modifications

```
$ git add Notebooks/Day1/inputForML.ipynb
```

```
$ git status
```

On branch main

Your branch is up to date with 'origin/main'.

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

modified: Notebooks/Day1/inputForML.ipynb

Untracked files:

(use "git add <file>..." to include in what will be committed)

Notebooks/Day1/.ipynb_checkpoints/

Notebooks/Day1/Higgs100k.parquet

Notebooks/Day1/__pycache__/

STEP 5: Push and Pull

- Send your changes to the central repository: `git push`
- Get the latest version of the code, including changes made by others: `git pull`

```
$ git commit -m "All exercises solved"
```

```
$ git status
```

[main 942ef43] All exercises solved

1 file changed, 7354 insertions(+), 22 deletions(-)

```
$ git status
```

On branch main

Your branch is ahead of 'origin/main' by 1 commit.

(use "git push" to publish your local commits)

...

To take home

- DevOps approach for fast time to value
- Version Control System (Git) to book-keep code changes and ease collaboration
- Make your workflow understandable and reproducible (Notebooks)

