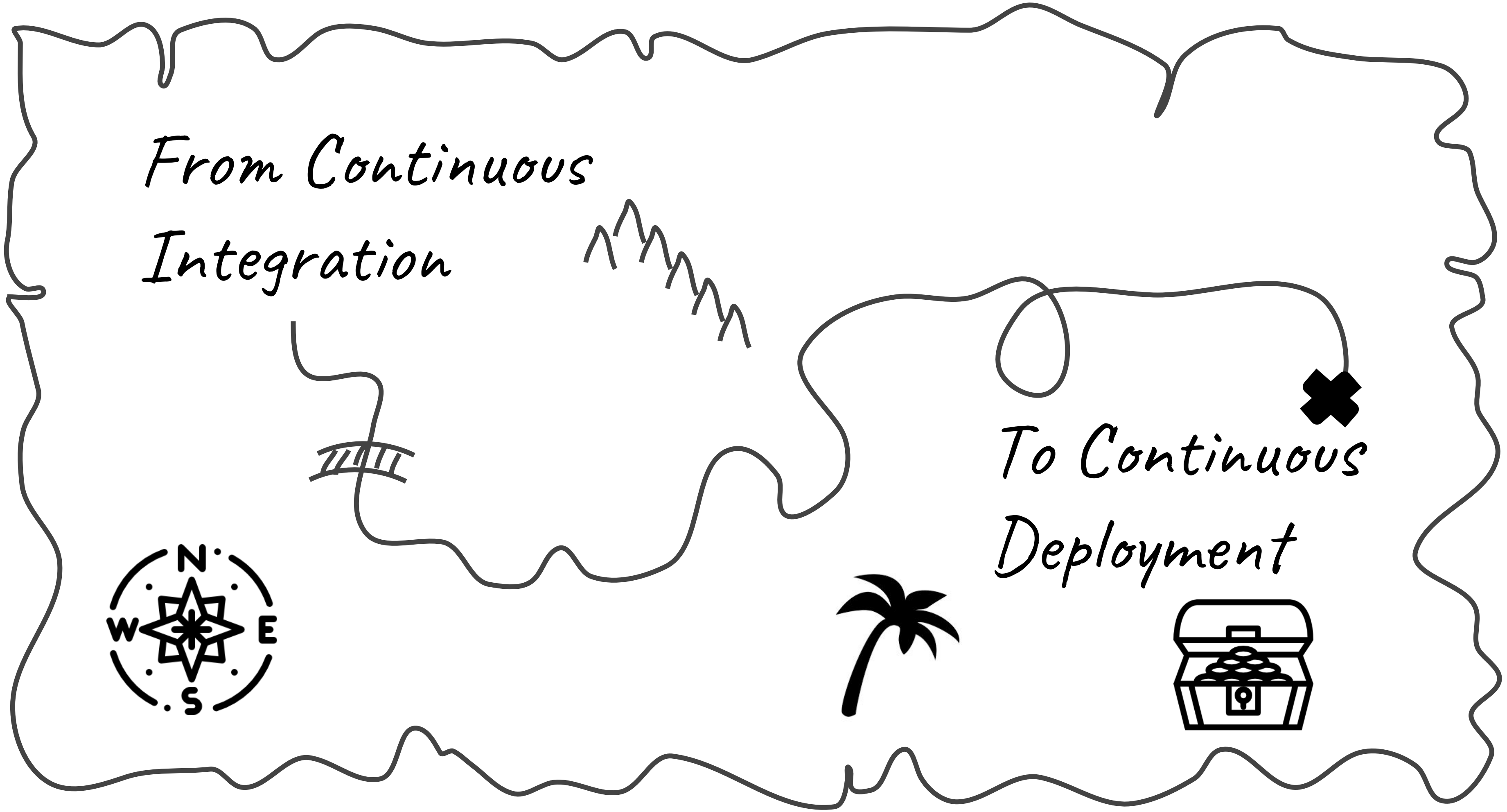
From Continuous Integration To Continuous Deployment



Julien Vey

Operational Excellence Manager

radiofrance

radiofrance ...

7 national radio stations

44 local radio stations

4 musical groups

~ 4200 employees

~ 100 different jobs

radiofrance digital

6 websites

5 mobile apps

49 millions visits (sept 2018)

53 millions downloaded podcasts (sept 2018)

Once Upon a Time,

We decided to move to a microservice architecture

...to get away from the monolith

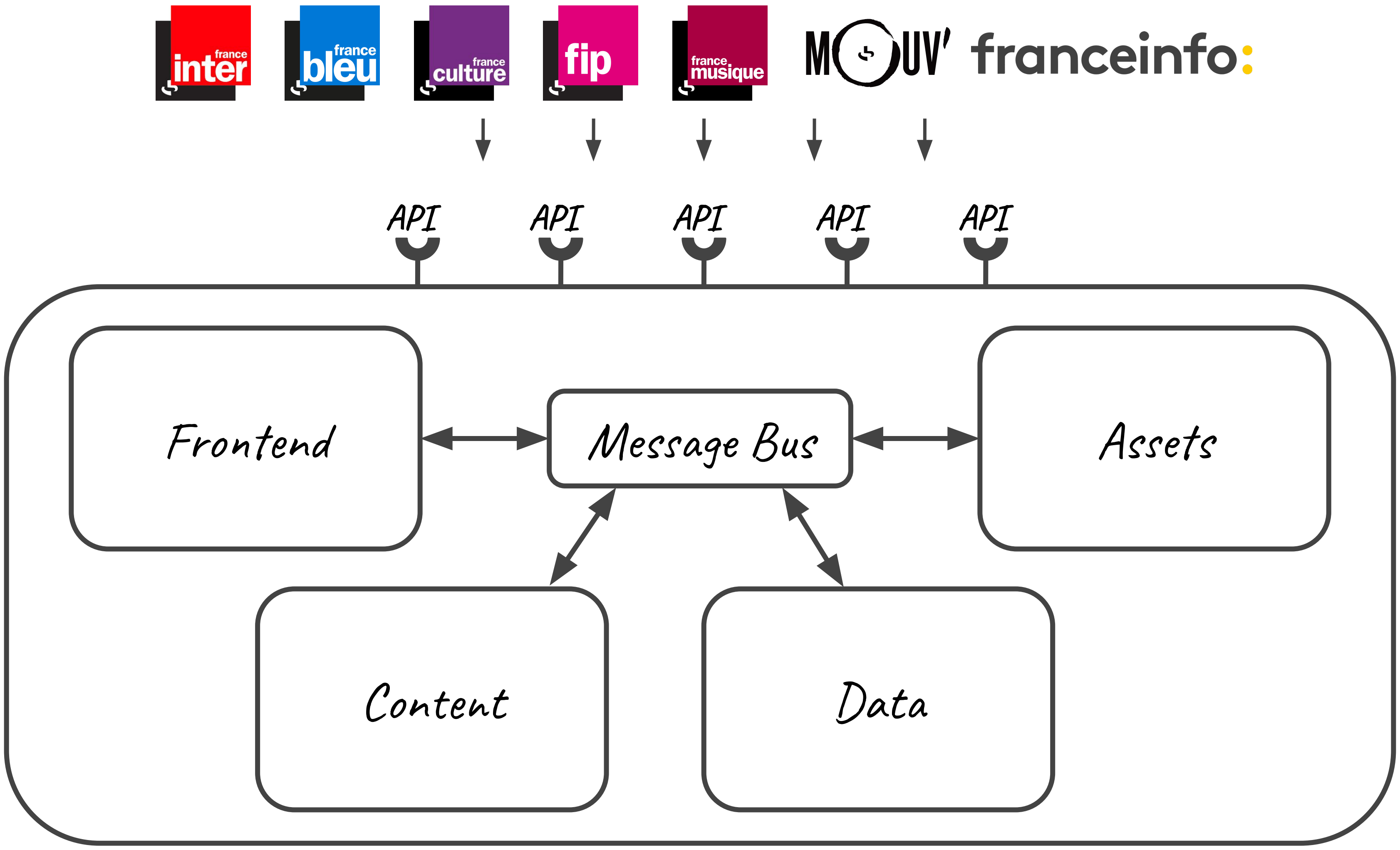
Why a Microservice architecture

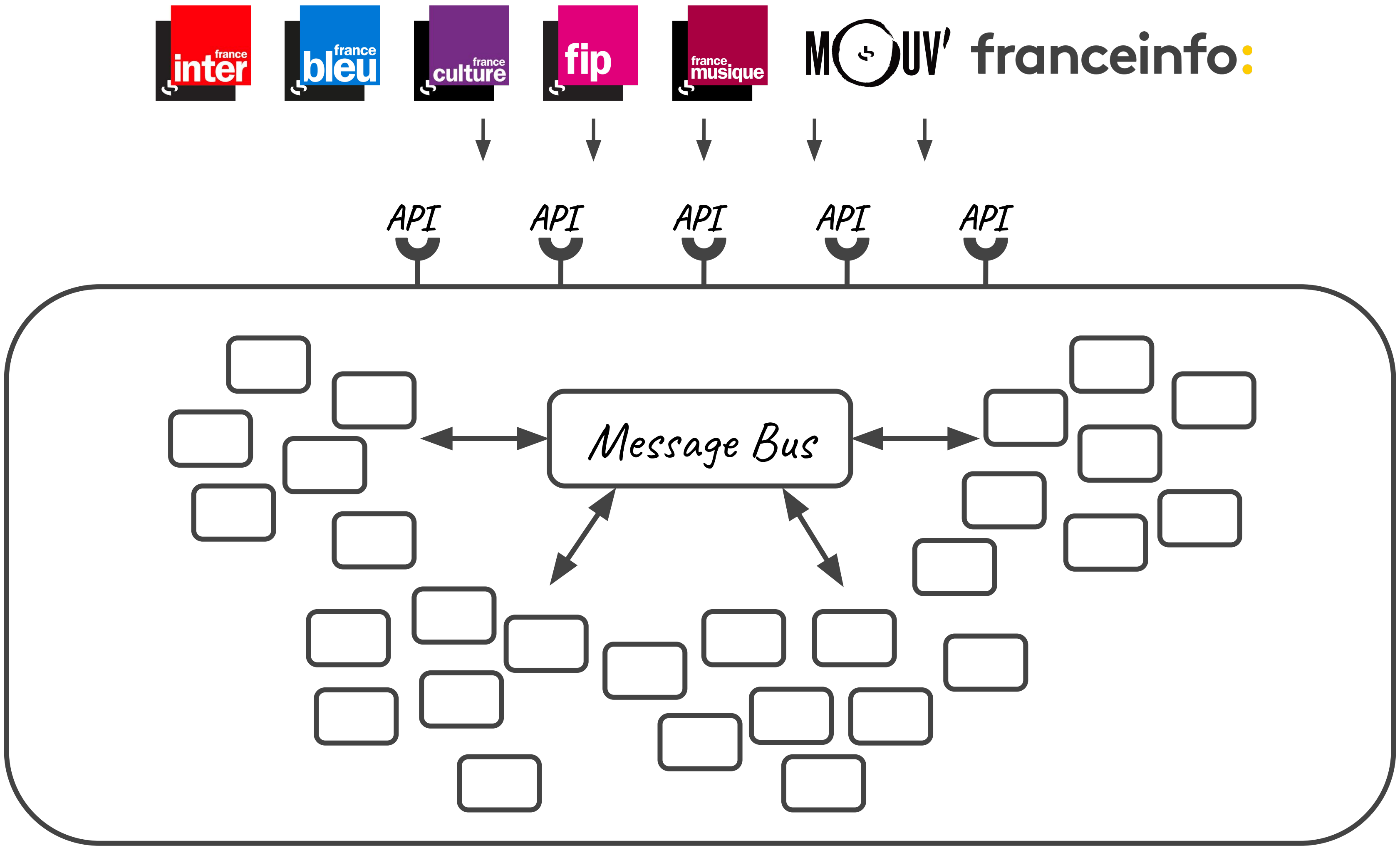
API First

Mutualize work on data, assets & content management

Standardize data format between stations

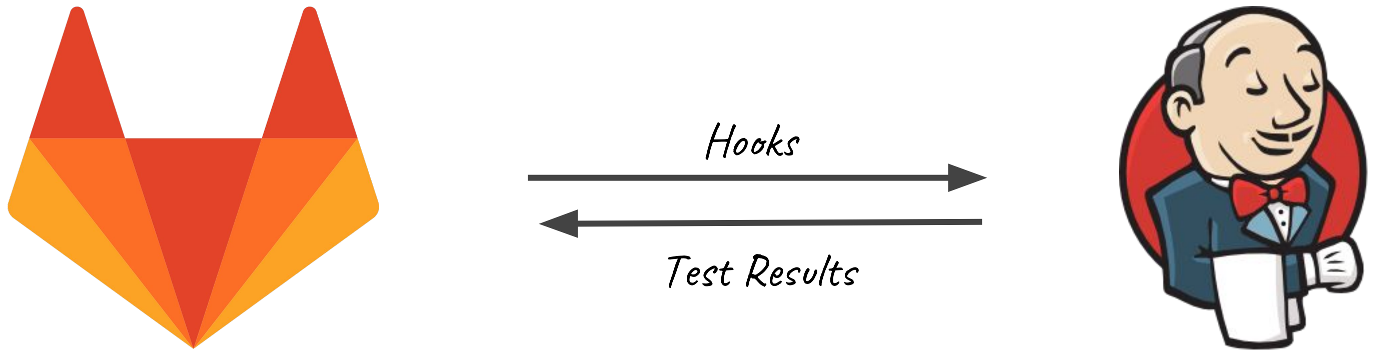
Shared infrastructure (+ logs and monitoring)


API
API
API
API
API
Frontend
Message Bus
Assets
Content
Data



API
API
API
API
API
Message Bus

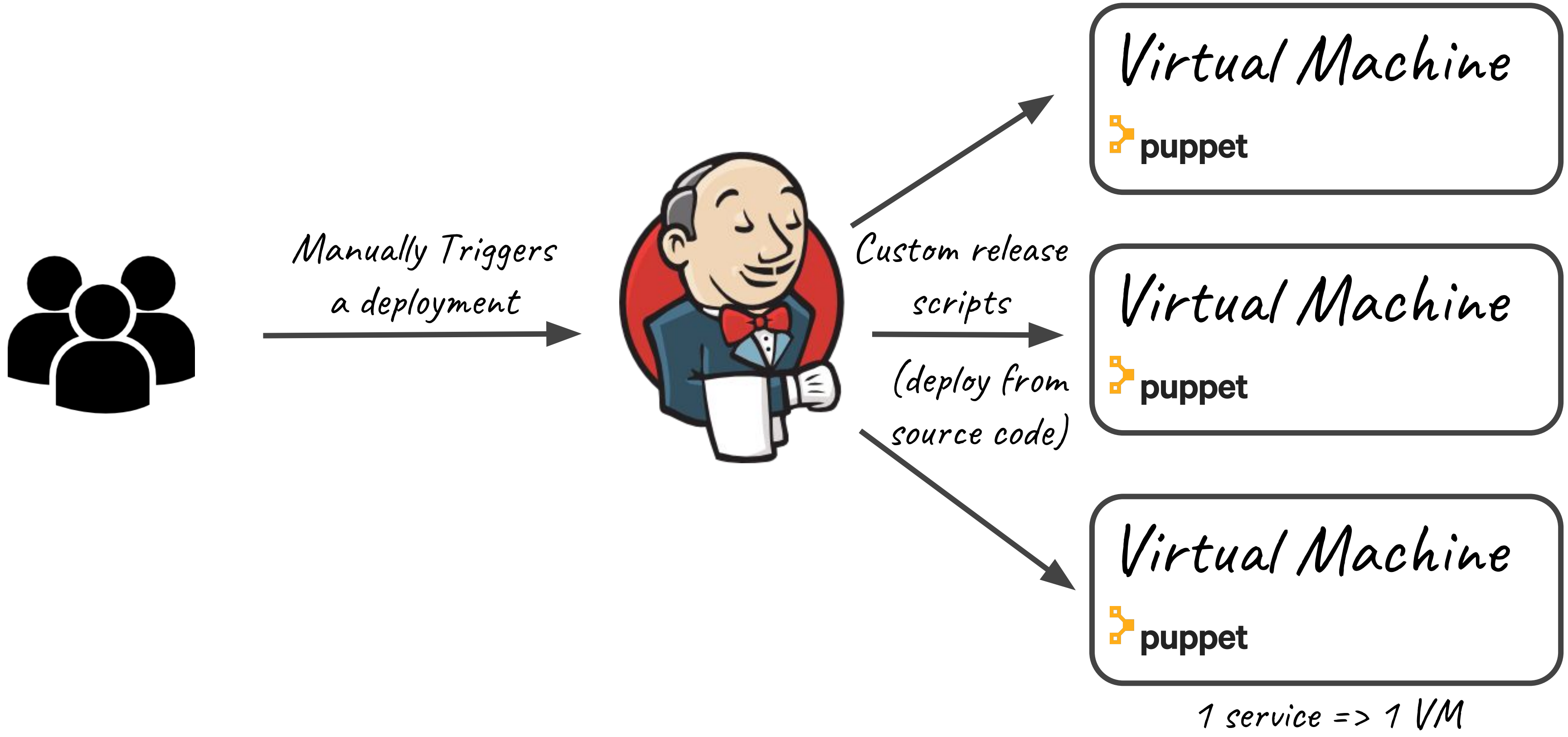
“Classic” Continuous Integration



Source Code on Gitlab
1 service = 1 repository

Jenkins to run tests

Not Continuous Deployment



Feedback

Not easy to scale each service independently

Failover is complex to automate

Custom scripts, custom problems

Hard to customize Jenkins per project

Once Upon a Time (again),

We decided to move to Kubernetes

Challenges

We had to rethink our build process (containers !)

We had to rethink our deployment process

on both Dev and Ops side

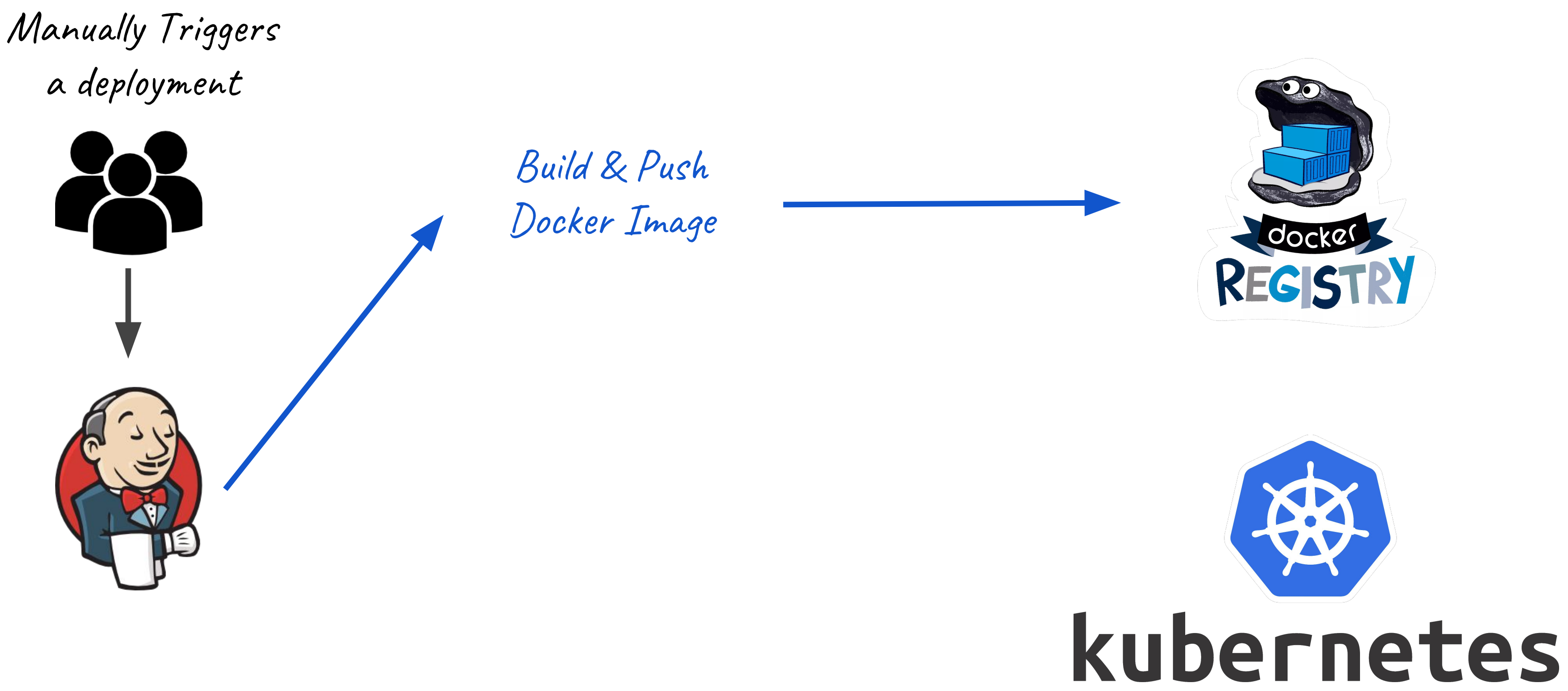
We no longer deploy code
We deploy containers

How to release a new version

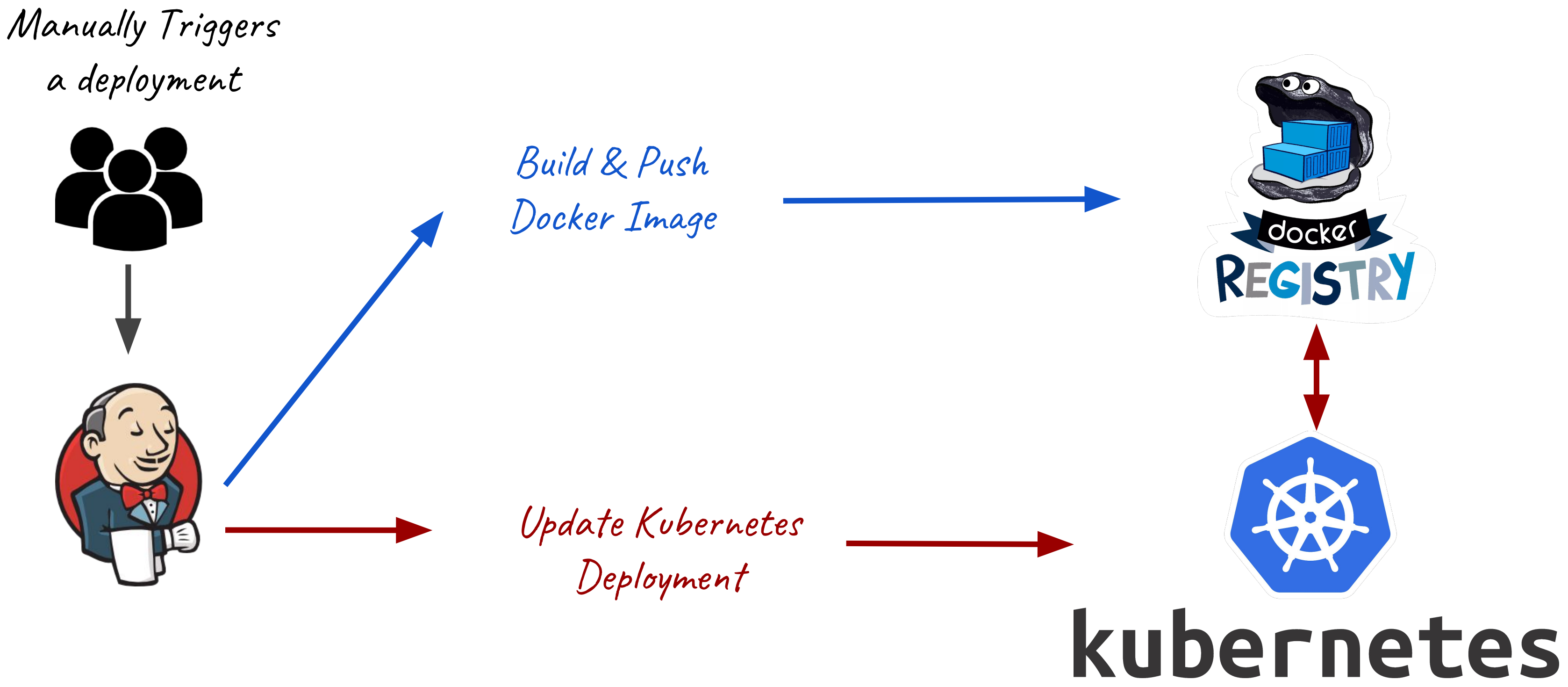
1. Build the docker image for the version
2. Tell Kubernetes to use this new image

That's all !

Not (yet) Continuous Deployment



Not (yet) Continuous Deployment

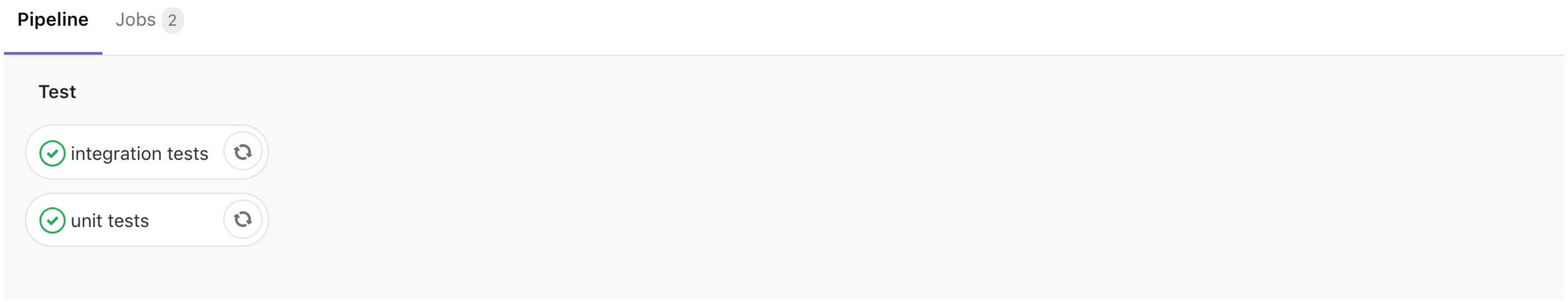

Manually Triggers a deployment
Build & Push Docker Image
Update Kubernetes Deployment
docker REGISTRY
kubernetes

Once Upon a Time (again),

We decided to move away
from Jenkins

...to use modern CI tools

Gitlab Pipelines (v1) "Continuous Integration"

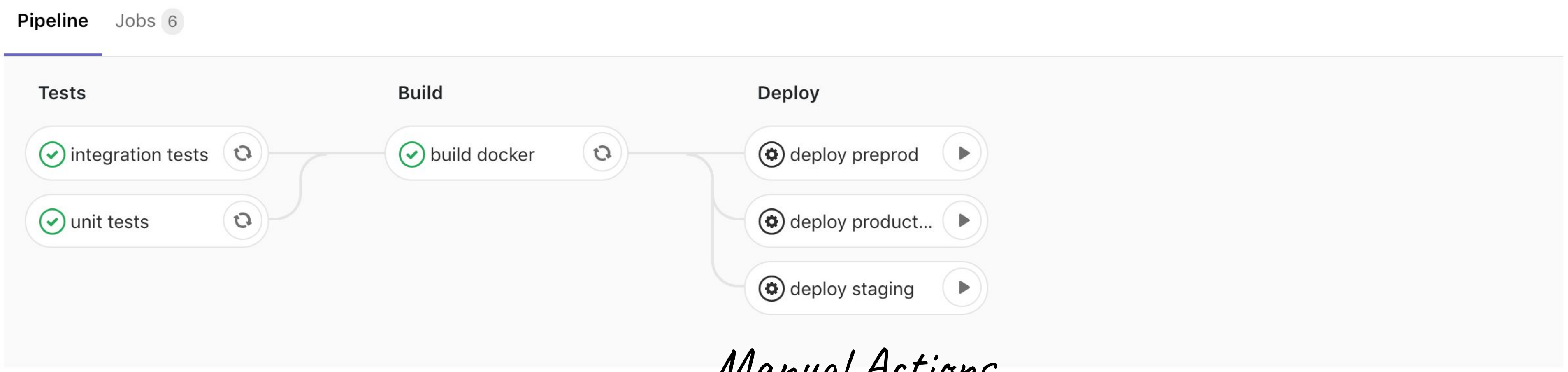


Test

✓ integration tests

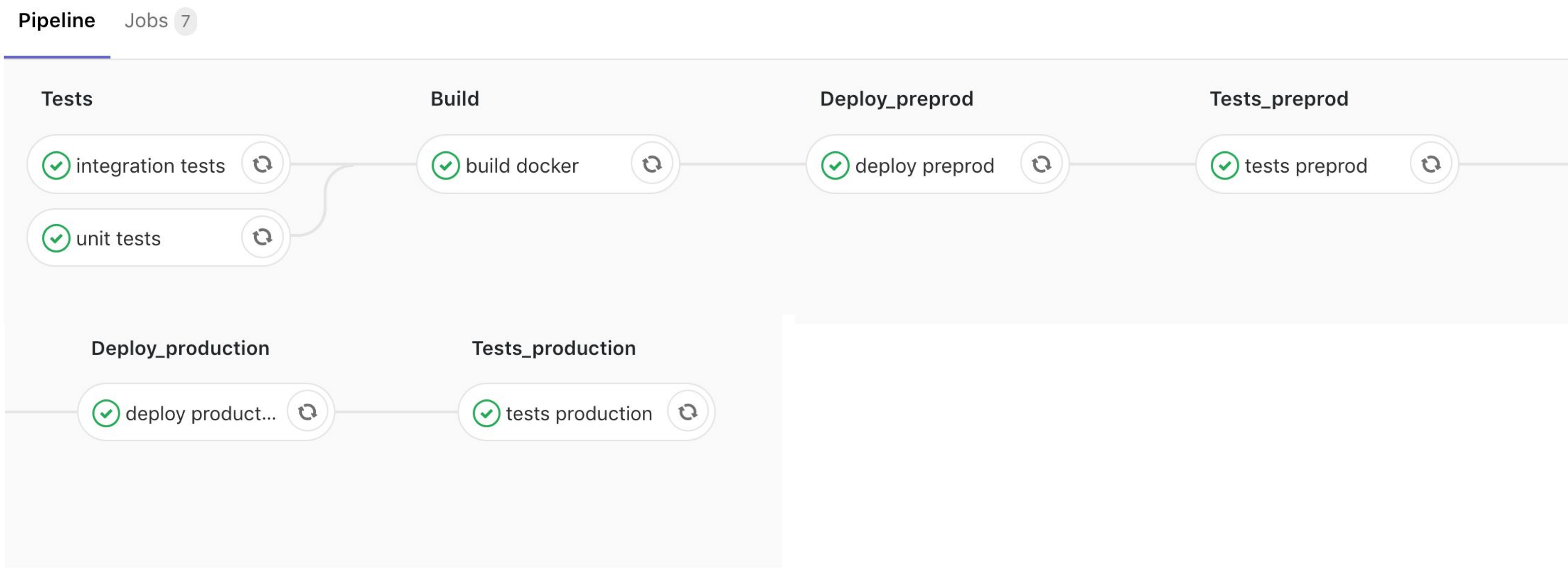
✓ unit tests

Gitlab Pipelines (v2) "Continuous Delivery"



Manual Actions

Gitlab Pipelines (v3) "Continuous Deployment"



Challenges

Test the application, and its dependencies ?

We have to keep backward compatibility

Gitlab CI Feedback

Continuous Integration and Continuous Deployment "as-code"

Everything is a container (it's great !)

We encapsulate the delivery logic in a docker container

The hardest part,

getting rid of the "Manual" action

Give Confidence & Visibility

Your release process is a software component

Test your code, but also your release process

Monitoring & Metrics give visibility

Critical services are hard to let go

What's next ?

Other types of deployment (Canary, A/B Tests, Blue Green)

Chaos Engineering (for both applications & deployment)

Thank you !