# DevOps approach and Infrastructure as Code for the SVOM mission

D. Corre on behalf of SVOM
French Science Ground Segment

HEPiX Spring 2024 - 18 April 2024

# Outline

- SVOM brief overview
  - Objectives
  - Global infrastructure
- FSGS collaboration
  - Overview
  - DevOps approach
- FSGS infrastructure
  - Automatic deployment
  - Migration to Kubernetes

# SVOM objectives

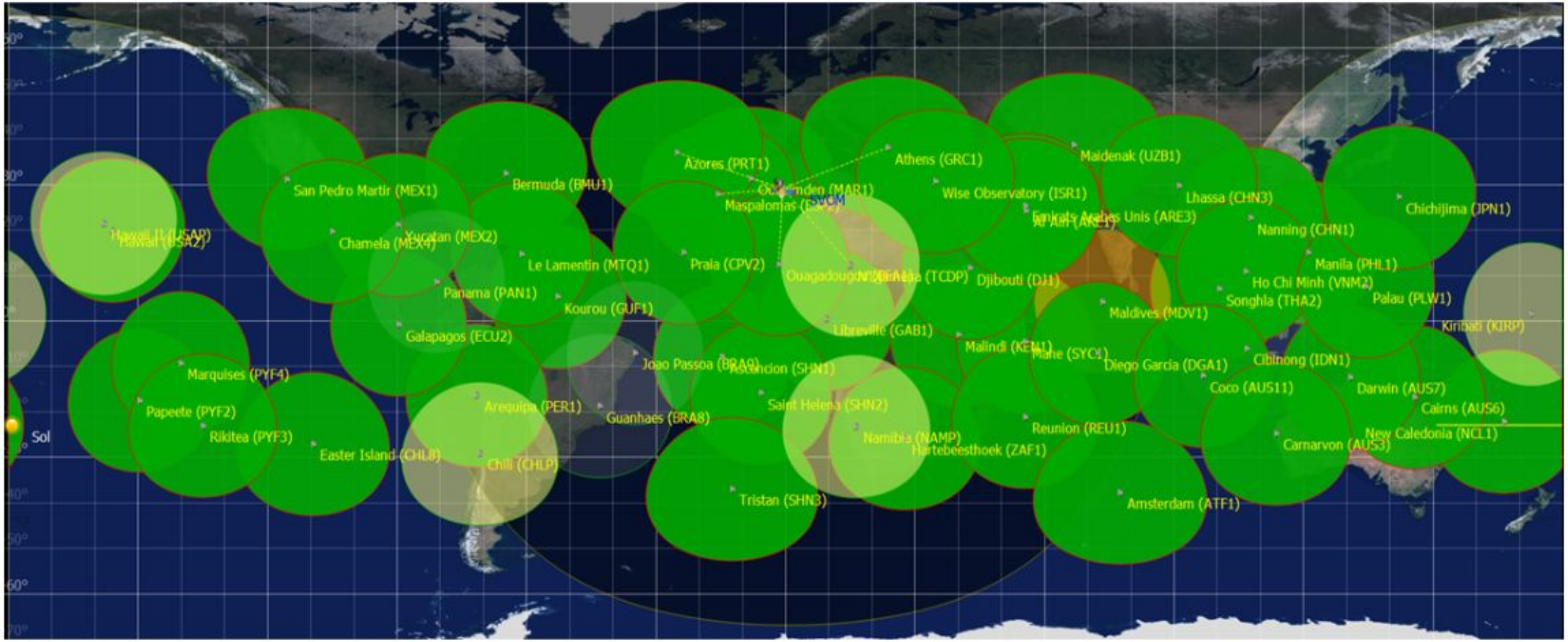
- **S**pace-based multi-band astronomical **V**ariable **O**bjects **M**onitor

- **Gamma-Ray Bursts observation**
  - Most energetic events observed since the Big Bang
  - **Can appear anywhere in the sky**
  - 2 types of emission
    - Prompt emission in gamma-rays (few seconds) → detection
    - Afterglow emission spanning wide range of wavelengths (few sec to min / hours) → characterisation
  - **Afterglow emission fades very quickly**
  - Coordinated observations over a wide range of wavelengths from space and from the ground are the key to fullest understanding this astronomical phenomenon.
  - GRB detection and localization estimation: **70 GRBs detected per year**

- **Major constraints : race against the time**
  - Continuous monitoring of the Sky + rare brief events → **high availability infrastructure**
  - Near Real time alert processing + short events → **automated services**

# Near Real-Time VHF Network

## 56 stations ~ 1 packet (100 bytes) / 1.8s



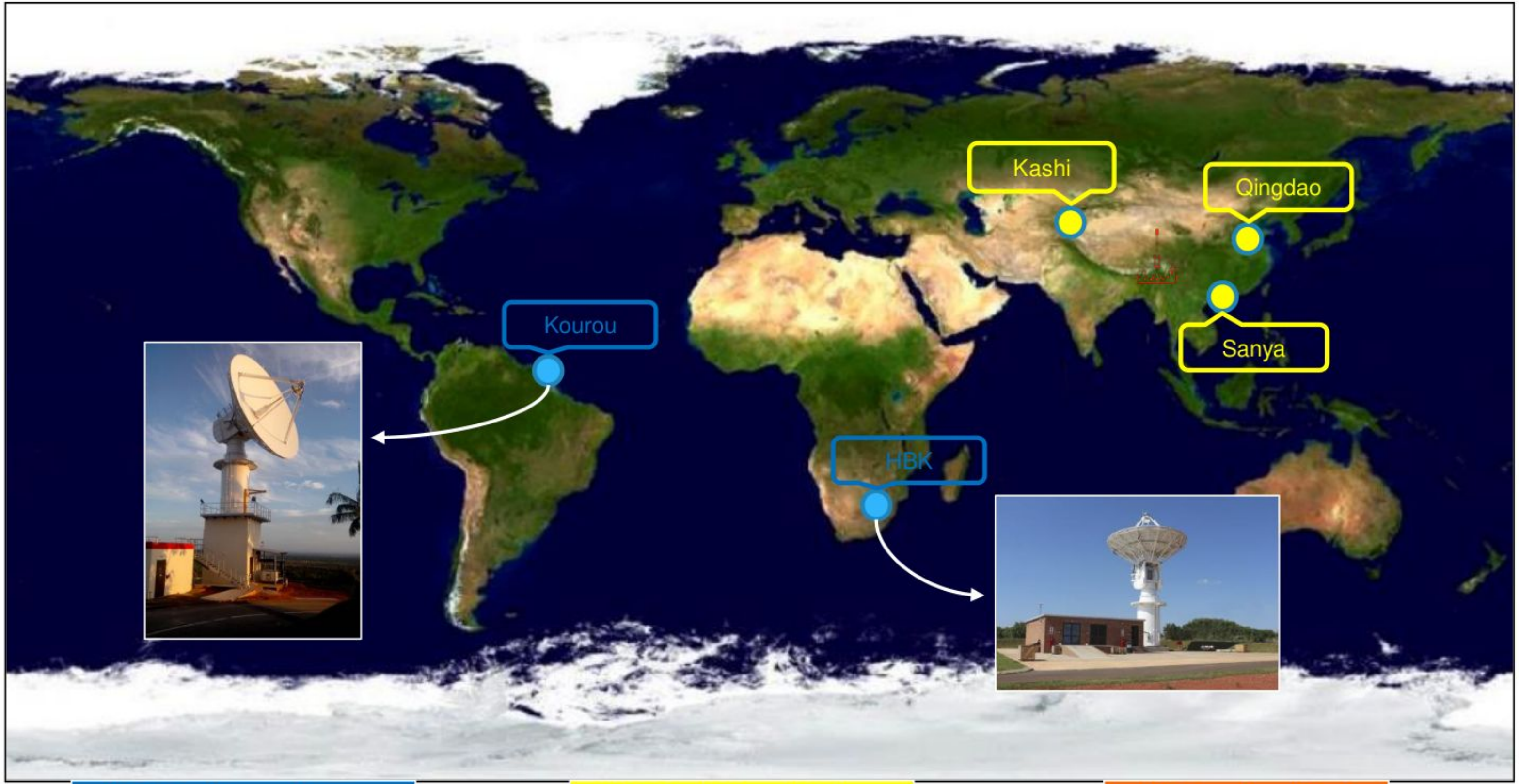
🟩 Station Installed

🟨 Station Ready to be installed

○ Discussions

# S & X band Ground Stations

## full raw data download 2GB / 6h

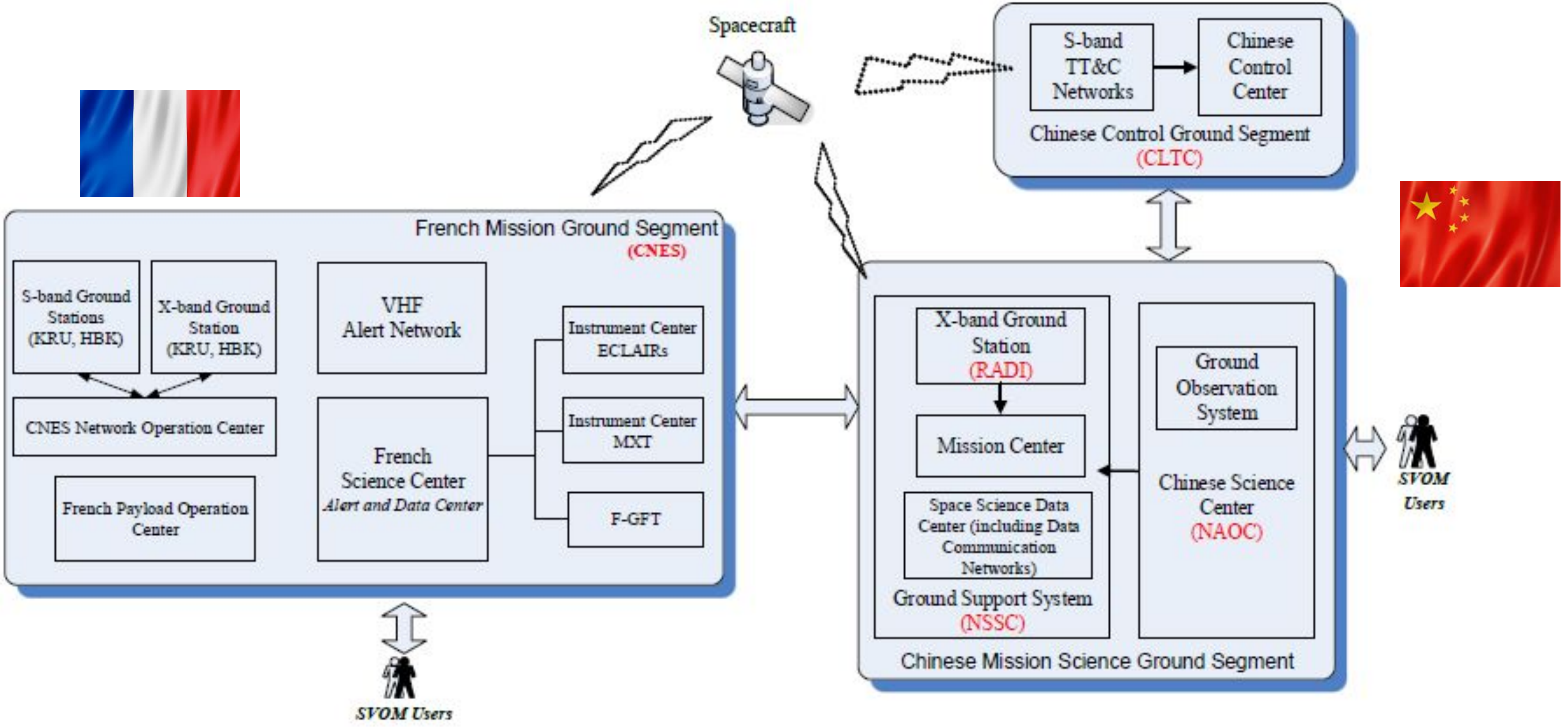


X & S Band Ground Stations

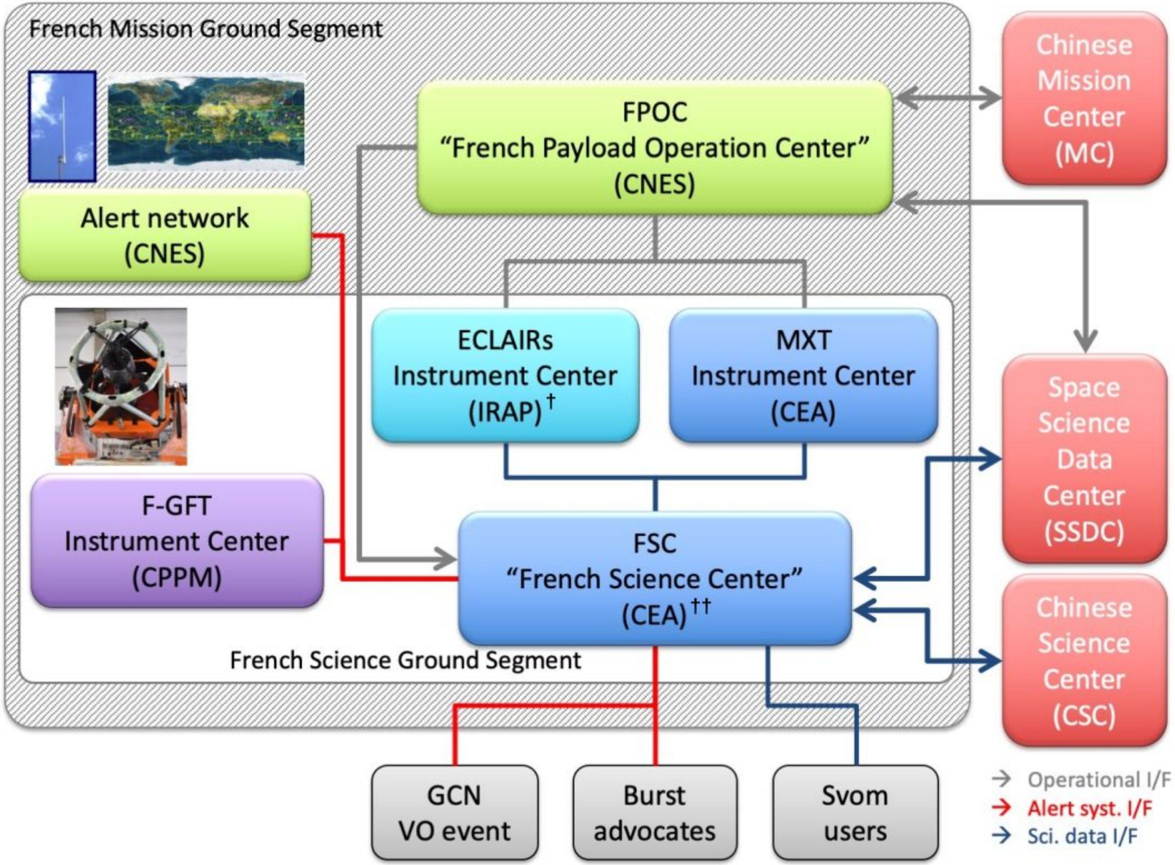
X & S Band Ground Stations

Xi'Chang Lauch Pad

# Mission segments


Spacecraft
S-band TT&C Networks
Chinese Control Center
Chinese Control Ground Segment
(CLTC)
French Mission Ground Segment
(CNES)
S-band Ground Stations (KRU, HBK)
X-band Ground Station (KRU, HBK)
VHF Alert Network
Instrument Center ECLAIRs
Instrument Center MXT
F-GFT
CNES Network Operation Center
French Science Center
Alert and Data Center
French Payload Operation Center
SVOM Users
X-band Ground Station
(RADI)
Mission Center
Space Science Data Center (including Data Communication Networks)
Ground Support System
(NSSC)
Ground Observation System
Chinese Science Center
(NAOC)
Chinese Mission Science Ground Segment
SVOM Users

# French segment



French Mission Ground Segment

FPOC
"French Payload Operation Center"
(CNES)

Alert network
(CNES)

Chinese Mission Center (MC)

ECLAIRs Instrument Center (IRAP)†

MXT Instrument Center (CEA)

Space Science Data Center (SSDC)

F-GFT Instrument Center (CPPM)

FSC
"French Science Center"
(CEA)††

Chinese Science Center (CSC)

French Science Ground Segment

GCN VO event

Burst advocates

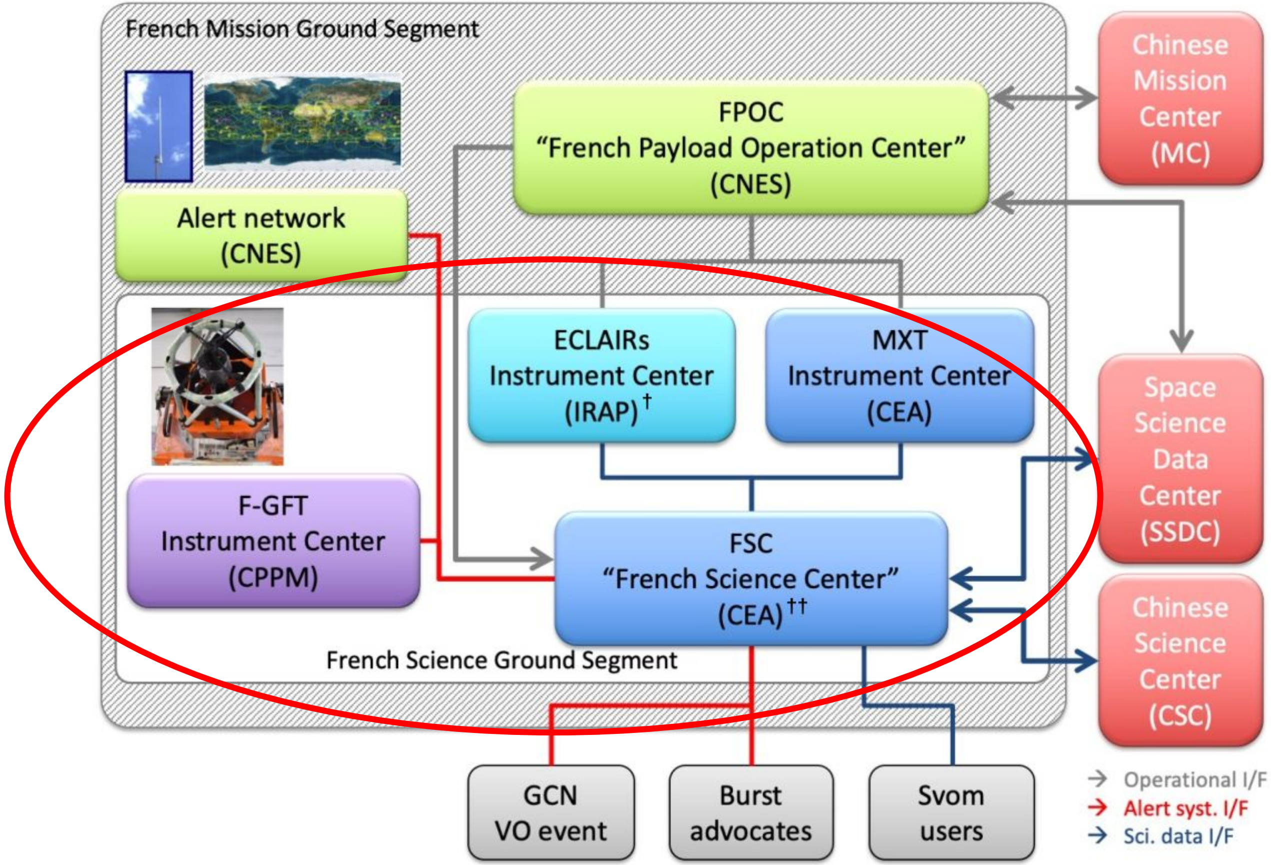
Svom users

→ Operational I/F
→ Alert syst. I/F
→ Sci. data I/F

† with contributions from:
CEA (Saclay)

†† with contributions from:
APC (Paris)
CPPM (Marseille)
GEPI (Meudon)
IAP (Paris)
IJCLab (Orsay)
LAM (Marseille)
LUPM (Montpellier)
ObAS (Strasbourg)

# French segment



† with contributions from:
CEA (Saclay)

†† with contributions from:
APC (Paris)
CPPM (Marseille)
GEPI (Meudon)
IAP (Paris)
IJCLab (Orsay)
LAM (Marseille)
LUPM (Montpellier)
ObAS (Strasbourg)

# Architecture principles

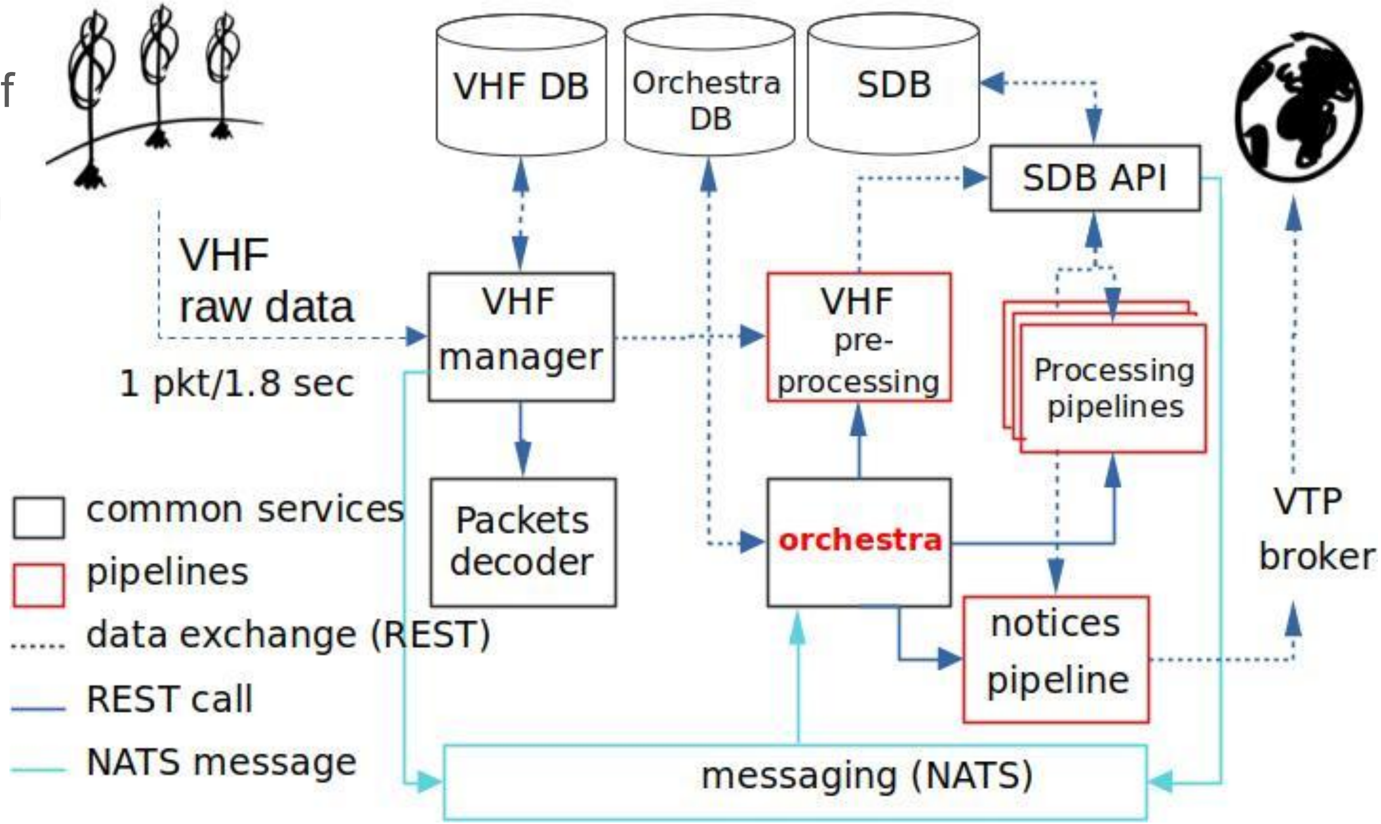
- **SOA** (service oriented architecture)
  - Define small services dedicated to specific tasks
    - **DB management :** satellite data, science products, etc
    - **Data processing :** algorithm processing data at different level
    - **Services Orchestration** : triggers specific services based on data availability
    - **Services Monitoring**
  - First (?) French space scientific segment using fully automated containerised microservices approach

- **Architecture and Protocols**
  - REST API :
    - **FSC data services offer REST APIs to external (and internal) clients**
    - Standardised API for different processing pipelines
    - This allow synchronous operations (data upload and retrieval)
  - **Messaging (NATS, MQTT) :**
    - **In several cases, an asynchronous communication is important:**
      - external clients need to be notified about new products @ FSC
      - FSC needs to be notified about new products (Beidou data, XBAND data)
    - Internally, orchestration relies on messaging to gather information on existing data and trigger pipeline processing on specific inputs

# FSGS Services Orchestrator

## Fully automated services orchestrated upon data availability

Simplified overview of VHF data processing



# DevOps approach: objectives

- **Objectives**
  - Let developers focus on development only
  - Common git workflow adopted by all actors
  - Homogeneous and automated integration processes
  - Automated services deployment

# DevOps approach: facts / constraints

- **Roadmap of facts / constraints**
  - Multi languages (Python, Java, Angular, React)
  - Several laboratories involved in France and China, ~30 developers (**small collaboration**)
  - ~150 projects hosted on a GitLab self instance at CEA Irfu
  - Fully containerised micro-services (~140 services)
  - Container orchestration technology / platform : Docker Swarm / Portainer
  - **Infrastructure deployed using OpenStack IaaS (CC IN2P3, IJCLab)**
  - High availability required
  - No money → open source only
  - **Satellite launch : June 2024**
  - Human resources for DevOps & Infrastructure : **1 person**

# DevOps approach: status

- **Status 1 year ago: custom agile +++++++**
  - Lack of global configuration management
  - No common git workflow
  - Each GitLab project responsible for its own GitLab CI/CD pipeline
  - Still manual steps:
    - deployment of infrastructure by different people, without documentation
    - deployment of services
    - OpenStack projects configuration
  - Not all micro-services containerised yet
  - Most containers were running as root
  - **There were many good things too** 😌

# DevOps approach

- **Improvement Priorities**
  - **Continuous Integration**
    - Set up common git workflow
    - Set up common CI templates to let developers develop!
  - **Continuous Deployment**
    - Automatise services deployment with **rootless** container
  - **Infrastructure**
    - Automatise infrastructure deployment on OpenStack
    - Automatise VMs configuration
    - Add a third OpenStack project to have 3 deployment sites: integration, pre-production and production
  - **Migrate to Kubernetes**

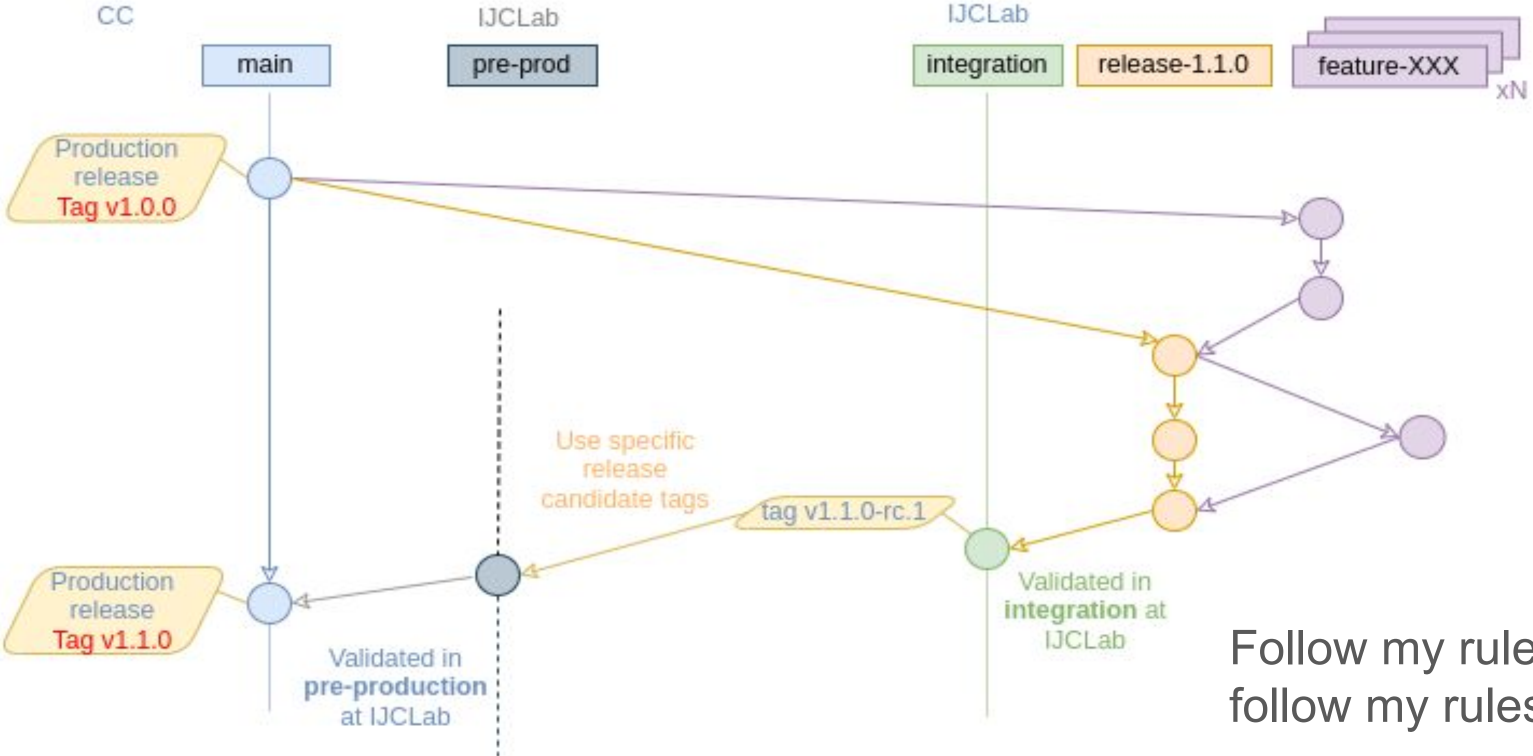
irfu

# DevOps approach: Continuous Integration

SVOM FSGS

- **Common git worflow: I must be cruel to be kind!**

3 environments : integration, pre-production and production


CC
IJCLab
IJCLab
main
pre-prod
integration
release-1.1.0
feature-XXX
xN
Production release Tag v1.0.0
Use specific release candidate tags
tag v1.1.0-rc.1
Validated in integration at IJCLab
Production release Tag v1.1.0
Validated in pre-production at IJCLab

Follow my rules or follow my rules! 😈

# DevOps approach: CI/CD

- **Common CI templates: kindness moment**
  - Ensure git workflow by implementing specific trigger rules for push / Merge Request

  - Remove responsibility of maintaining the GitLab CI/CD pipelines to developers

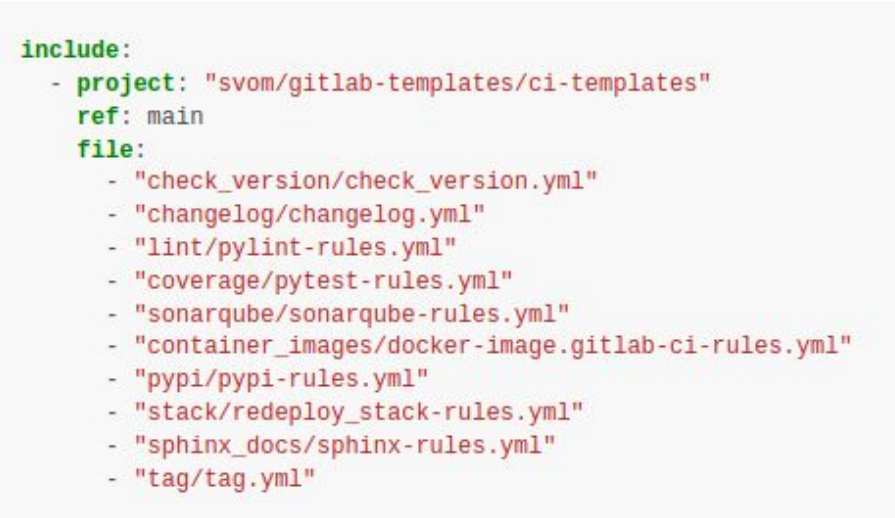
  - Ensure homogeneity of CI/CD jobs
    - quality report (lint tools, ruff, SonarQube)
    - unit testing
    - container building / pushing (Docker, Kaniko)
    - vulnerability scans (SonarQube, Trivy)
    - tag format
    - documentation (Sphinx)
    - packaging (private PyPI)
    - changelogs (towncrier)
    - release version conformity

  - Ensure automated deployment of services into OpenStack trough Portainer

  - Homogeneity allows GitLab projects configuration checks (end of kindness moment 💔)

# DevOps approach: CI templates

- **Make life easy for developers**
  - Simply include the CI templates in .gitlab-ci.yml



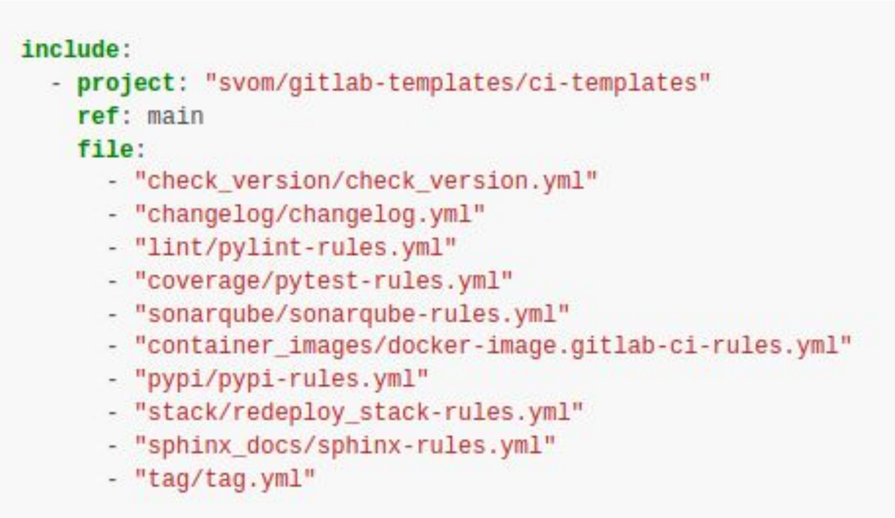
```yaml
include:
  - project: "svom/gitlab-templates/ci-templates"
    ref: main
    file:
      - "check_version/check_version.yml"
      - "changelog/changelog.yml"
      - "lint/pylint-rules.yml"
      - "coverage/pytest-rules.yml"
      - "sonarqube/sonarqube-rules.yml"
      - "container_images/docker-image.gitlab-ci-rules.yml"
      - "pypi/pypi-rules.yml"
      - "stack/redeploy_stack-rules.yml"
      - "sphinx_docs/sphinx-rules.yml"
      - "tag/tag.yml"
```

irfu

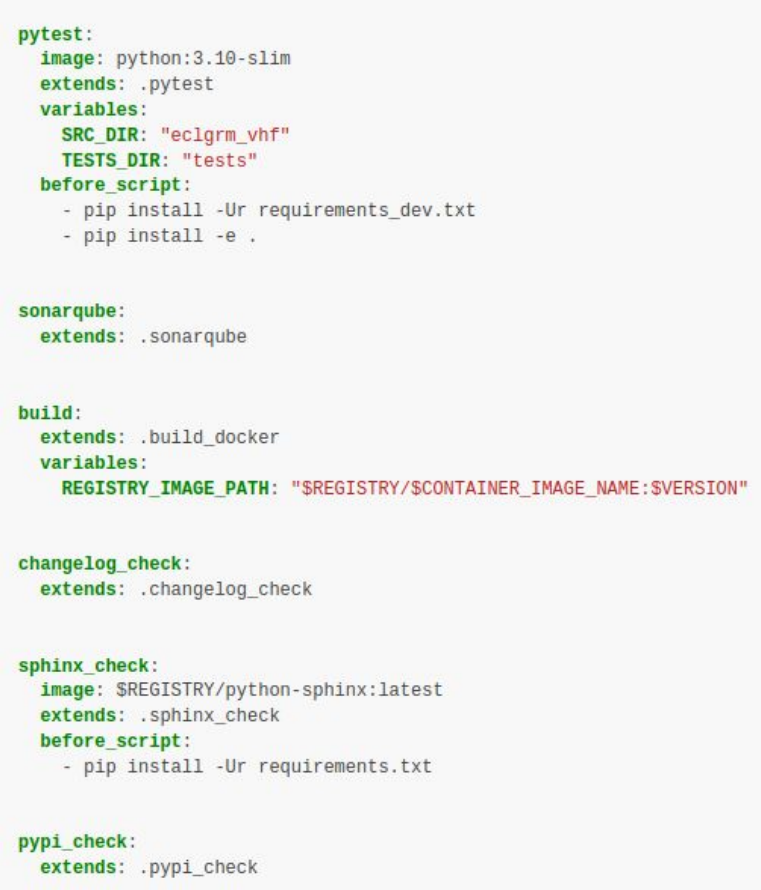
# DevOps approach: CI templates

SVOM FSGS

- **Make life easy for developers**
  - Simply include the CI templates in .gitlab-ci.yml
  - And add the desired CI jobs (copy/paste)



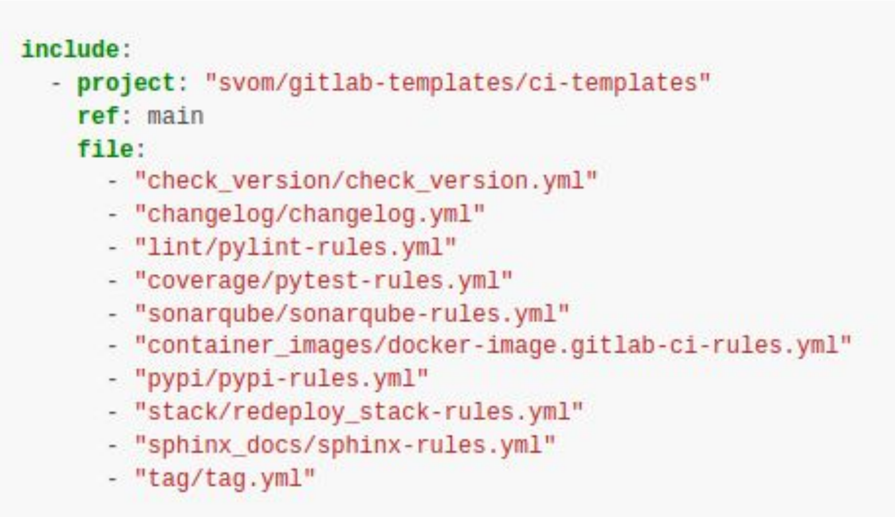
```yaml
include:
  - project: "svom/gitlab-templates/ci-templates"
    ref: main
    file:
      - "check_version/check_version.yml"
      - "changelog/changelog.yml"
      - "lint/pylint-rules.yml"
      - "coverage/pytest-rules.yml"
      - "sonarqube/sonarqube-rules.yml"
      - "container_images/docker-image.gitlab-ci-rules.yml"
      - "pypi/pypi-rules.yml"
      - "stack/redeploy_stack-rules.yml"
      - "sphinx_docs/sphinx-rules.yml"
      - "tag/tag.yml"
```



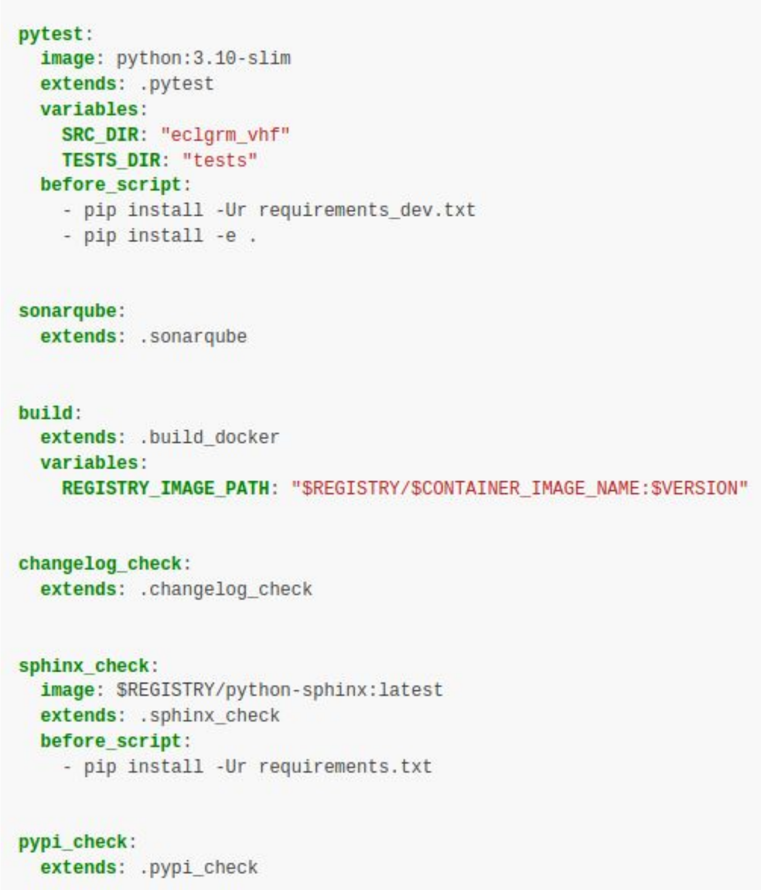
```yaml
pytest:
  image: python:3.10-slim
  extends: .pytest
  variables:
    SRC_DIR: "eclgrm_vhf"
    TESTS_DIR: "tests"
  before_script:
    - pip install -Ur requirements_dev.txt
    - pip install -e .


sonarqube:
  extends: .sonarqube


build:
  extends: .build_docker
  variables:
    REGISTRY_IMAGE_PATH: "$REGISTRY/$CONTAINER_IMAGE_NAME:$VERSION"


changelog_check:
  extends: .changelog_check


sphinx_check:
  image: $REGISTRY/python-sphinx:latest
  extends: .sphinx_check
  before_script:
    - pip install -Ur requirements.txt


pypi_check:
  extends: .pypi_check
```

# DevOps approach: CI templates

- **Make life easy for developers**
  - Simply include the CI templates in .gitlab-ci.yml
  - And add the desired CI jobs (copy/paste)
  - Developers have flexibility to configure other jobs



```yaml
include:
  - project: "svom/gitlab-templates/ci-templates"
    ref: main
    file:
      - "check_version/check_version.yml"
      - "changelog/changelog.yml"
      - "lint/pylint-rules.yml"
      - "coverage/pytest-rules.yml"
      - "sonarqube/sonarqube-rules.yml"
      - "container_images/docker-image.gitlab-ci-rules.yml"
      - "pypi/pypi-rules.yml"
      - "stack/redeploy_stack-rules.yml"
      - "sphinx_docs/sphinx-rules.yml"
      - "tag/tag.yml"
```



```yaml
pytest:
  image: python:3.10-slim
  extends: .pytest
  variables:
    SRC_DIR: "eclgrm_vhf"
    TESTS_DIR: "tests"
  before_script:
    - pip install -Ur requirements_dev.txt
    - pip install -e .


sonarqube:
  extends: .sonarqube


build:
  extends: .build_docker
  variables:
    REGISTRY_IMAGE_PATH: "$REGISTRY/$CONTAINER_IMAGE_NAME:$VERSION"


changelog_check:
  extends: .changelog_check


sphinx_check:
  image: $REGISTRY/python-sphinx:latest
  extends: .sphinx_check
  before_script:
    - pip install -Ur requirements.txt


pypi_check:
  extends: .pypi_check
```

# Infrastructure as Code

- **Terraform using OpenStack provider**
  - Use Terraform scripts to automate provisioning of idempotent infrastructure:
    - Networking (networks, routers, subnets)
    - Security Groups (acting as virtual firewall)
    - Server Groups (to spread critical VMs over different hypervisors)
    - Virtual Machines (CPUs, RAM)
    - OS images (Rocky Linux 9.3)
    - Volumes
    - Floating and Virtual IPs
  - Use modules to factorise code for our 3 OpenStack projects / environments while accounting for cloud provider specificities / differences
  - Easy integration in GitLab CI/CD (lint, validation, deployment)
  - Whole FSGS infrastructure (re)deployment duration : few minutes
  - VMs IPs are stored in config files for further use by Ansible

# Infrastructure as Code

- **Ansible for VMs / software configuration**
  - Set up inventories adapted to each of our environment (integration, pre-production, production)

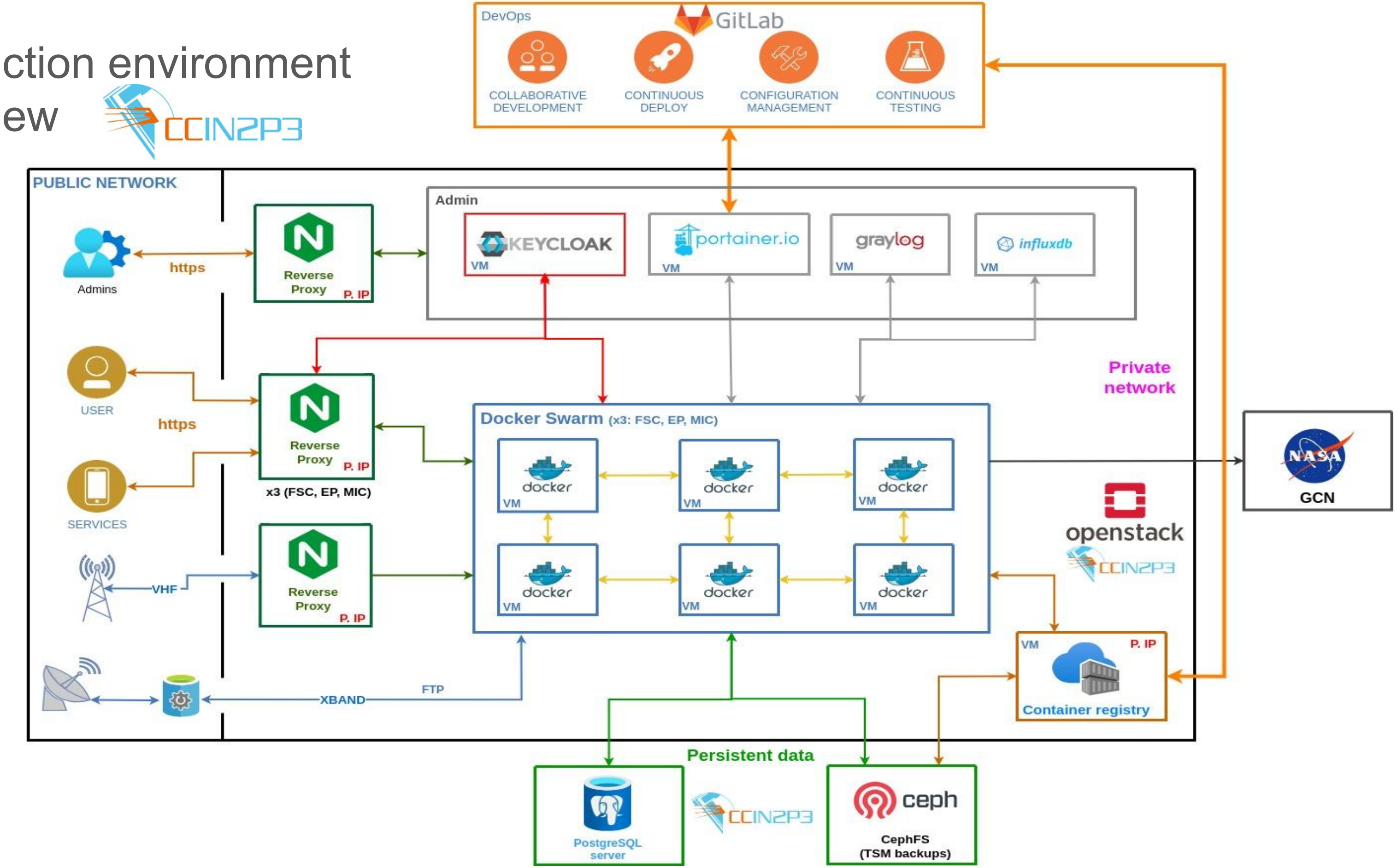
  - Use host variables to link instances defined in inventories with IP addresses retrieved by Terraform

  - Usage of custom roles for configuring:
    - OS hardening
    - FSGS services deployment
    - Keycloak
    - Portainer
    - Docker Swarm
    - Kubernetes
    - nginx
    - Certificates (letsencrypt)
    - NFS
    - cephFS mount
    - PostgreSQL server
    - Many others

  - Whole FSGS infrastructure configuration duration : <~ 1 hour

# FSGS with Docker Swarm

Production environment overview



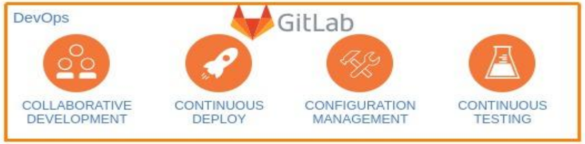
# Migration to Kubernetes

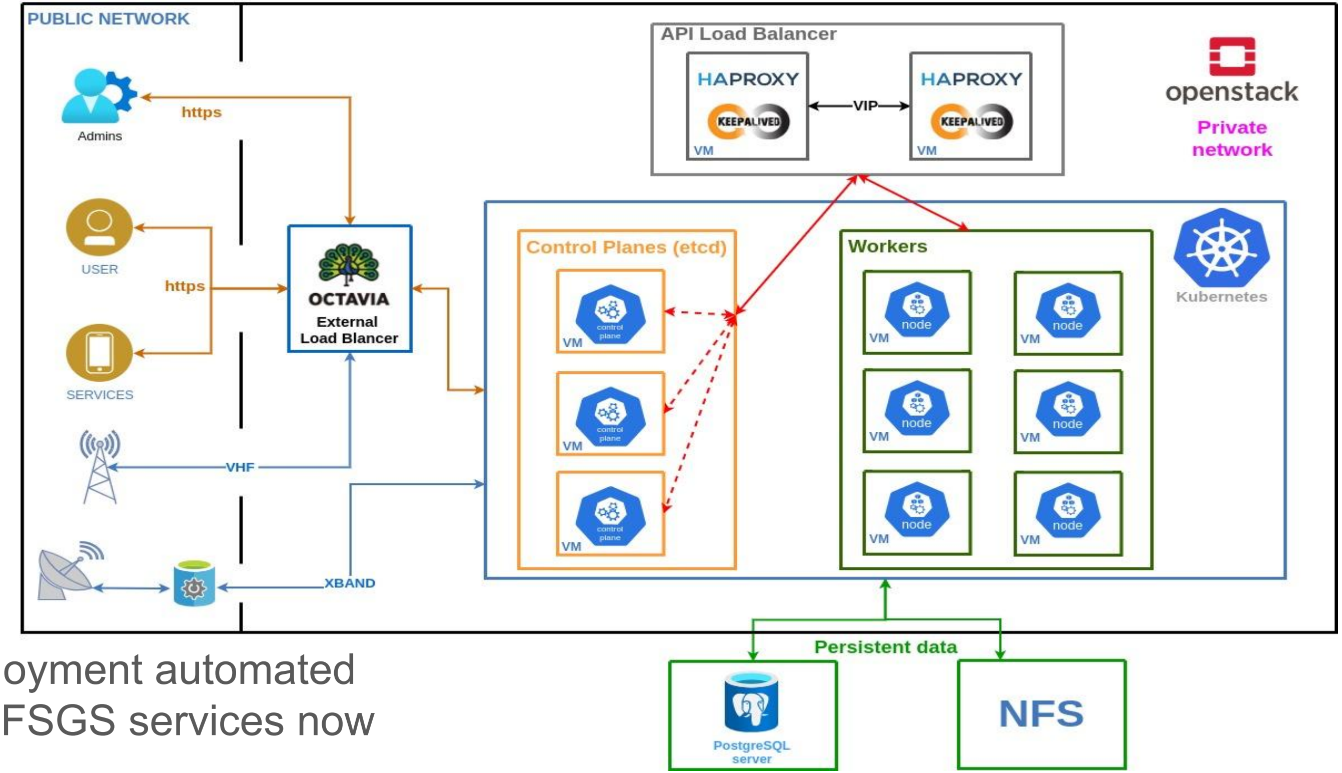
- **Motivation (deadline October 2024)**
  - Improve network stability
  - Better scalability management
- **Swarm → k8s**
  - Use of **kompose** to help converting docker-compose.yml into k8s manifests
  - Use of **kubespray** Ansible collection to configure the k8s cluster (CNI, CRI, LB, CephFS, etc)
  - Use of **nginx controller**
  - Use of **Portainer** to deploy k8s deployments from GitLab repos
  - Use of **Keycloak** fo authentication and user management
- **Change of technologies**
  - **Prometheus** instead of Graylog
  - **ELK** instead of InfluxDB
  - LBaaS **Octavia**
  - ArgoCD / FluxCD / Flamingo instead of Portainer ?
  - Gateway API instead of Ingress API?
  - OPA or kyverno?
- **Others**
  - **Harbor** instead of Docker Registry
  - Set up scalable gitlab runners fleet

Feedback / tips on Kubernetes deployment are welcomed!

# FSGS with Kubernetes

Testing environment overview





Main structure deployment automated
Ready to integrate FSGS services now

# Summary

- **DevOps**
  - Micro-services are great but can be a chaotic nightmare without common configuration management
  - Set up a git workflow **in the early stages of the mission**
  - Invest efforts on a common strategy for CI/CD pipelines **ASAP**
    - Let developers develop!
    - Homogeneous CI allows configuration checks and provides a better quality overview
    - Automates Continuous Deployment

# Summary

- Micro-services are great but can be a chaotic nightmare without common configuration management
- Set up a git workflow **in the early stages of the mission**
- Invest efforts on a common strategy for CI/CD pipelines **ASAP**
- **IaaS (OpenStack) + IaC (Terraform, Ansible) very useful for limited manpower collaboration**
- Investing (reasonable) efforts on IaC is a good idea for the **present** but **also** the **future** → easily re-usable for future projects / collaborations

# Summary

- **Cloud deployment**
  - **Use of IaaS platforms as OpenStack**
  - Eases and fastens deployment
  - **Use of IaC tools as Terraform and Ansible**
    - ensure idempotency
    - allows rapid automated (re-)deployment
  - **IaaS + IaC very useful for limited manpower collaboration**
  - **IaC is easily re-usable for other projects / collaborations**

irfu

SVOM FSGS

# Thank you!