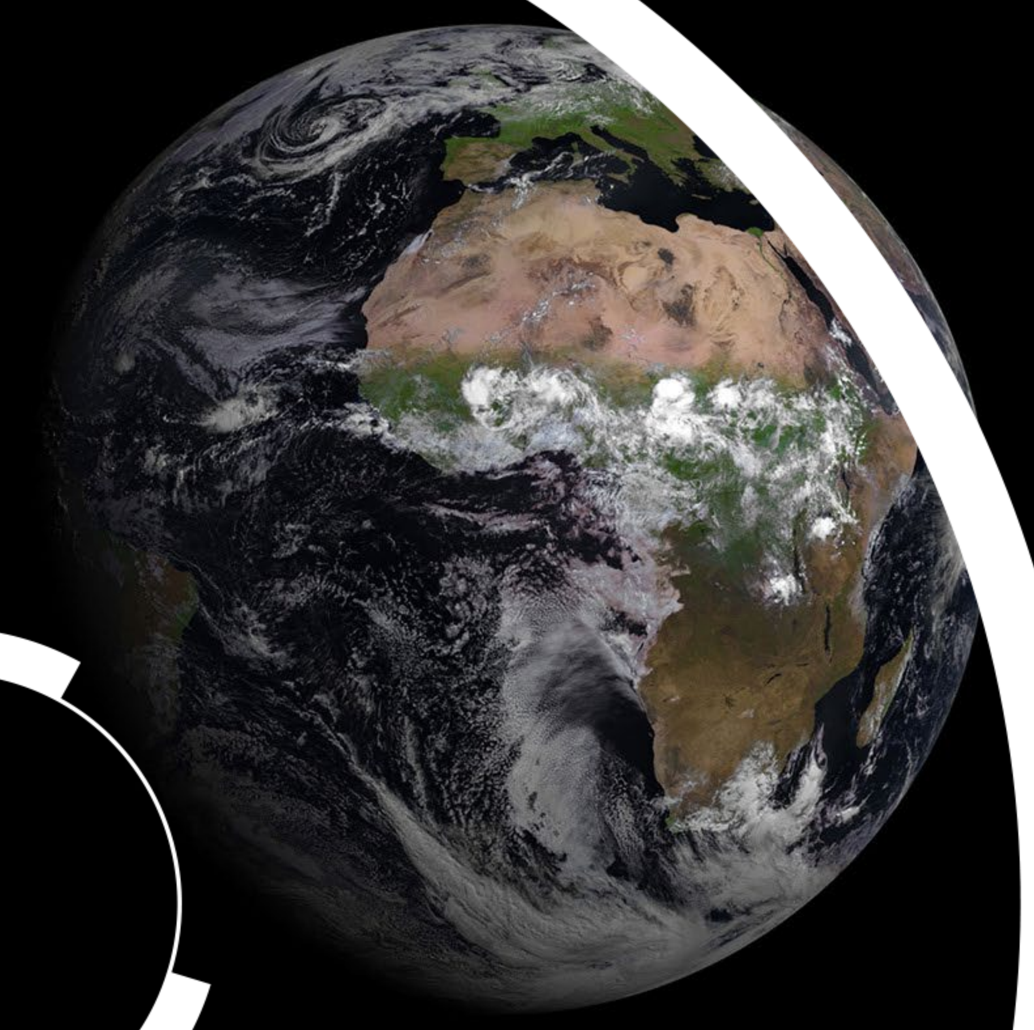
EUMETSAT



MME REP: Climate Data Records and EO data processing in a server-less computing paradigm

Salvatore Pinto, Mike Grant, Fernando Ibanez
Data Reprocessing Engineer, EUMETSAT

ECMWF, 20th ECMWF workshop on high performance computing in meteorology - 13/10/2023

Agenda

The problem

Lots of missions, lots of different mission products + climate data processing, diversity of processors

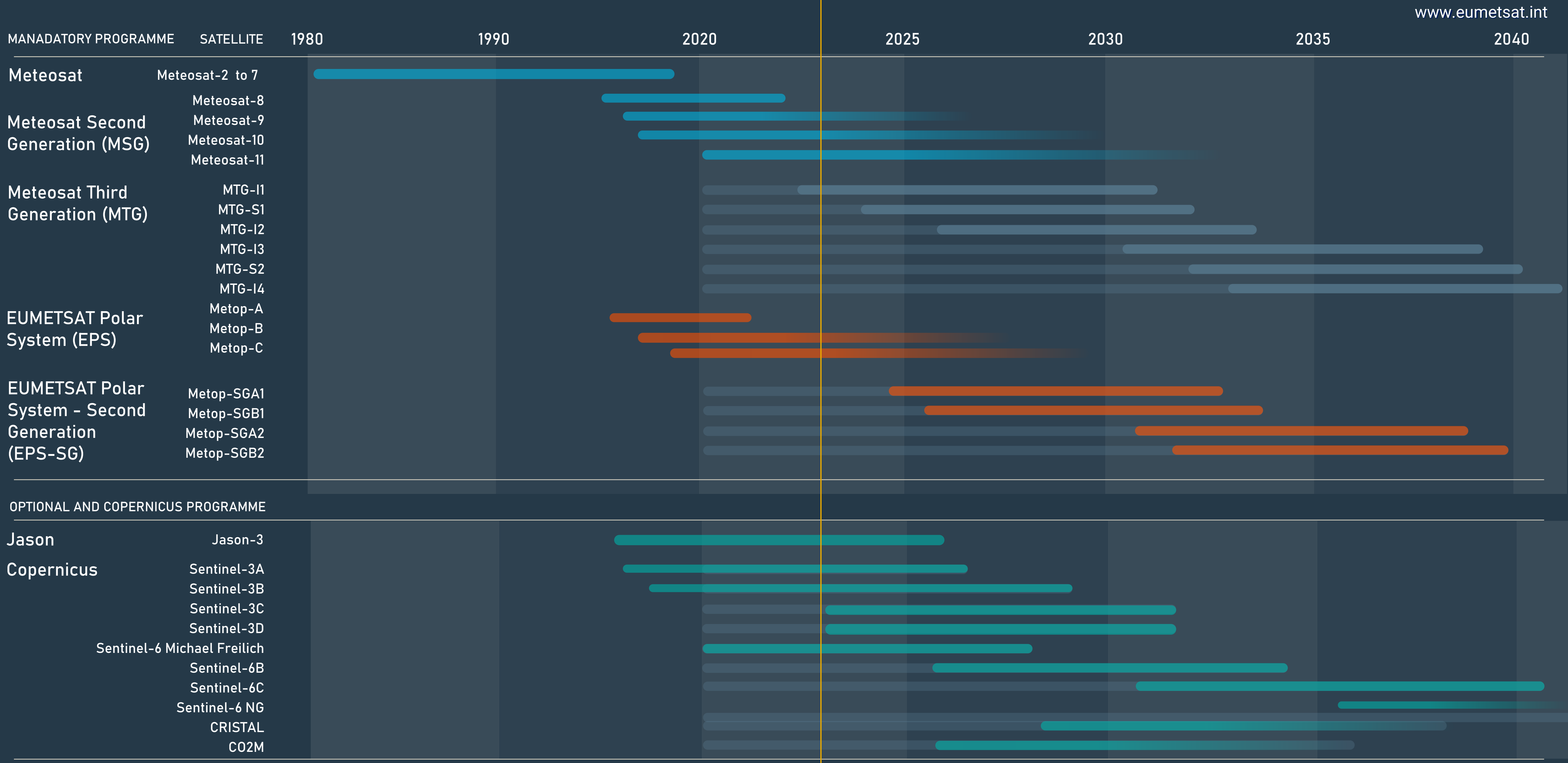
The solution

Transition to serverless computing, cloud and container technologies

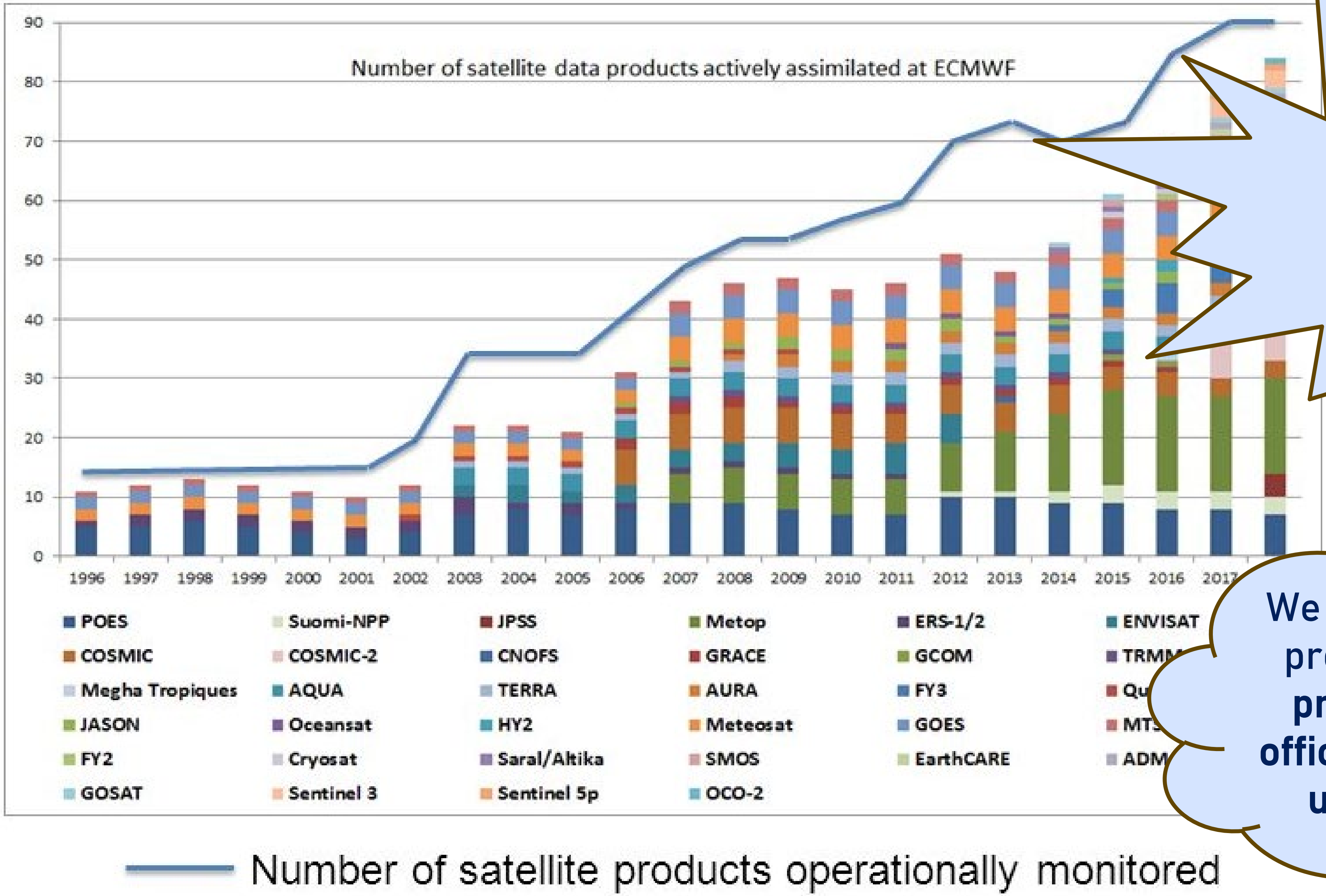
The system

What we have, how it works

Lots of Satellite missions (producing data to be re/processed)



Lots of data products (to be assimilated in NWP and more)



EUMETSAT contributes to about **1/3 of all data** assimilated at ECMWF

We produce other data products for **internal processing, the met offices directly or other user communities**

Processing at EUMETSAT

- **Near-Real-Time:**
  - Satellite measurements “as soon as they are acquired”
  - Disseminated primary though EUMETcast

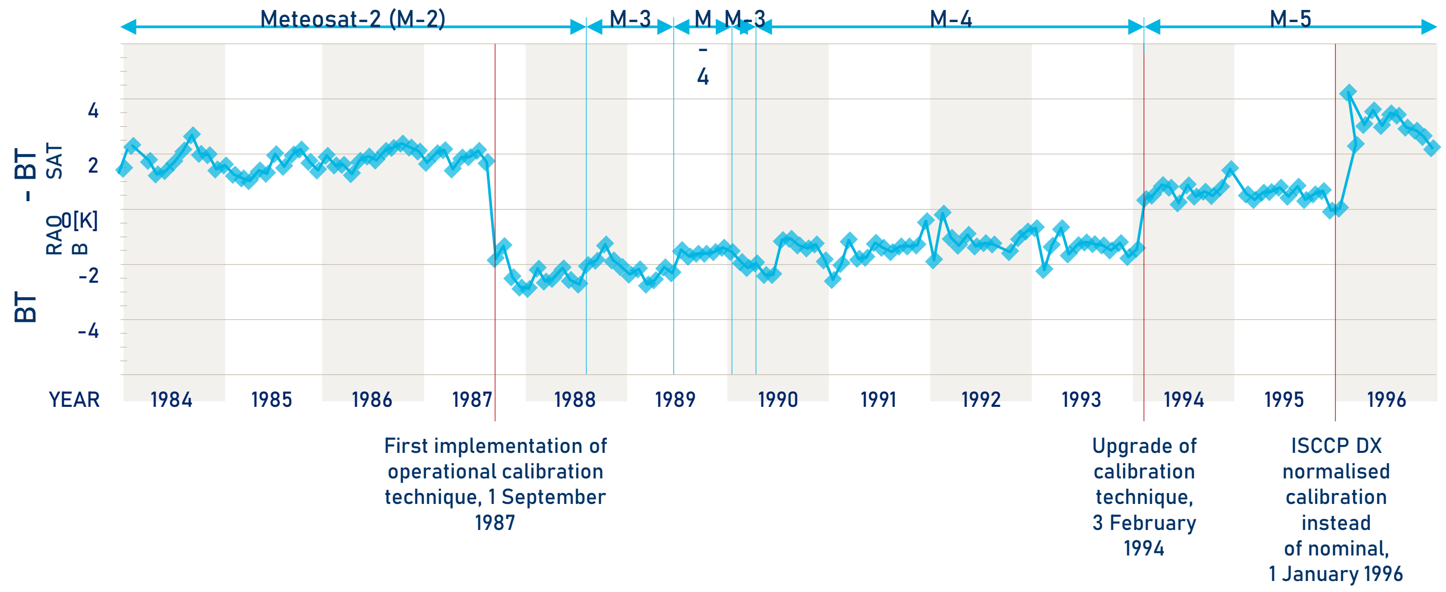
- **Climate Data Records (this presentation)**
  - Processing on the entire archive
  - Disseminated primarily through EUMETSAT Data Store

EUMETSAT Data Centre (in the center)

Climate Data Records – (re)processing full datasets

- **Fundamental Climate Data Records (FCDRs)**, consistent and calibrated time series of "direct" observations, e.g. Meteosat radiances FCDR
- **Climate Data Records (CDRs)**, long timeseries of uncertainty-quantified "derived" values of a geophysical variable or related indicator (e.g. wind vectors)

Remove inconsistencies due to satellite or algorithm changes



Improve the quality of products with better algorithms or cleaned-up/reprocessed inputs

Number of reprocessed products extracted from Meteosat imaging with a quality index > 80%, 2005-2013. Reprocessed (purple) has more high-quality products than original (blue).

- **Each reprocessing produces the complete dataset size again (or more!)**

Diversity of processors/software (just some examples)

~10 processors

10-20 processors

10-30 processors each

10-15 processors

~8 processors

+ a plethora of "custom" and prototype processors

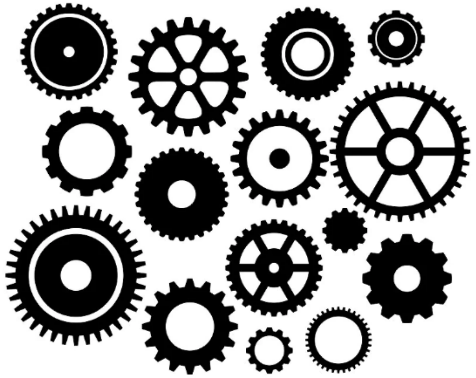
Diversity of processing framework and dependencies

Batch processing

Streaming, micro-batch

Specific OS dependency

Databases (and side services)



Custom frameworks and dependencies (developed from scratch or built on top of OSS)

With some missions, there is a limited to null possibility to adapt or recode the algorithm to run it on a different processing framework

Our key needs

www.eumetsat.int

Flexibility

- Several different software approaches needing different processing frameworks (batch, streaming and a lot of custom services)

Simplicity of use

- most of our processor developers are not fluent with optimization for different platforms and they want to worry "only about the science"
  - it may be "cheaper" to run a bit longer/slower than optimizing the processor

Performance

- FCDR/CDR complete run (entire archive) in 3 months
  - the maximum still allowing iteration and experimentation in feasible wait times
- no supercomputer numbers, but still a processing cluster in the order of thousands of CPU cores

Agenda

The problem

Lots of missions, lots of different mission products
+ climate data processing, diversity of processors

The solution

Transition to serverless computing, cloud and container technologies

The system

What we have, how it works

Bulk data processing in EUMETSAT, a bit of history

~2014

CDR-1

PBS

Computing Cluster (static)

ORACLE

Database

NAS

POSIX Datastore

- ✗ **Flexibility**
- ✗ **Scaling**

~2019

CDR-2

HTCondor

Computing Cluster (virtualized, dynamic)

Data analysis nodes

ORACLE

Database

NAS

POSIX Datastore

Object storage

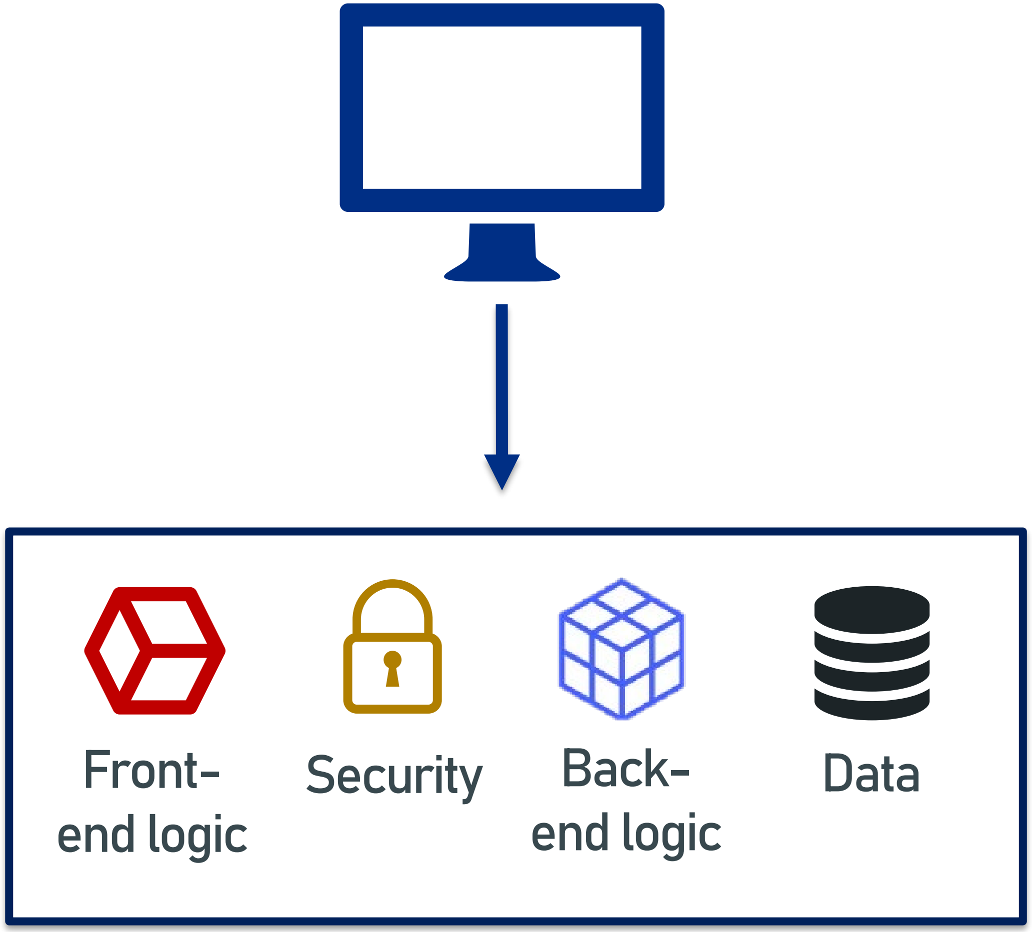
- ✚ **Flexibility** (data analysis nodes)
- ✚ **Scaling** (start VMs on demand, object storage for IN/OUT)

- **Flexibility** (run "everything we want/need")
- **Scalability** (2 order of magnitude, on-demand)

Traditional vs Serverless computing

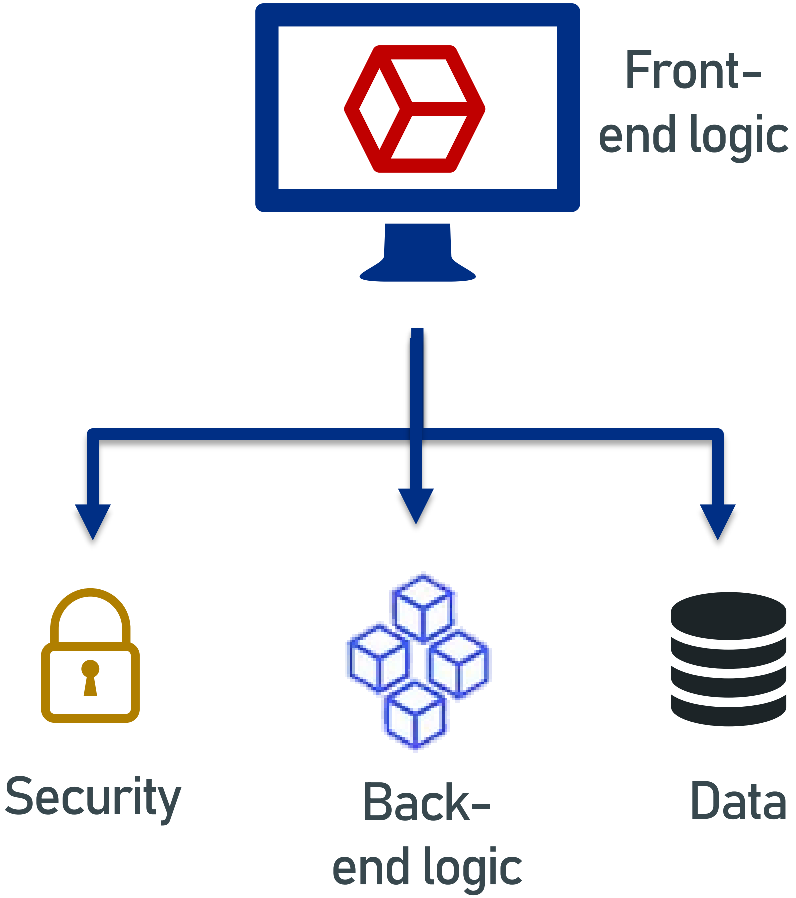
Traditional

Fully managed server(s) hosting the application



Serverless

Client-side logic and thirty-party services



Bulk data processing in EUMETSAT, past and present

~2014 — CDR-1

PBS

Computing Cluster (static)

ORACLE

Database

NAS

POSIX Datastore

- ✗ **Flexibility**
- ✗ **Scaling**

Traditional

~2019 — CDR-2

HTCondor

Computing Cluster (virtualized, dynamic)

jupyter

Data analysis nodes

ORACLE

Database

NAS

POSIX Datastore

Object storage

- ✚ **Flexibility** (data analysis nodes)
- ✚ **Scaling** (start VMs on demand, object storage for IN/OUT)

2023 — MME-REP

(Multi-Mission Element for REProcessing)

Computing/storage/network services

kubernetes

HTCondor

jupyter

Multi-tier storage

kafka

APACHE STORM

EUMETSAT Data Lake

- **Flexibility** (run “everything we want/need”)
- **Scalability** (2 order of magnitude, on-demand)

Serverless

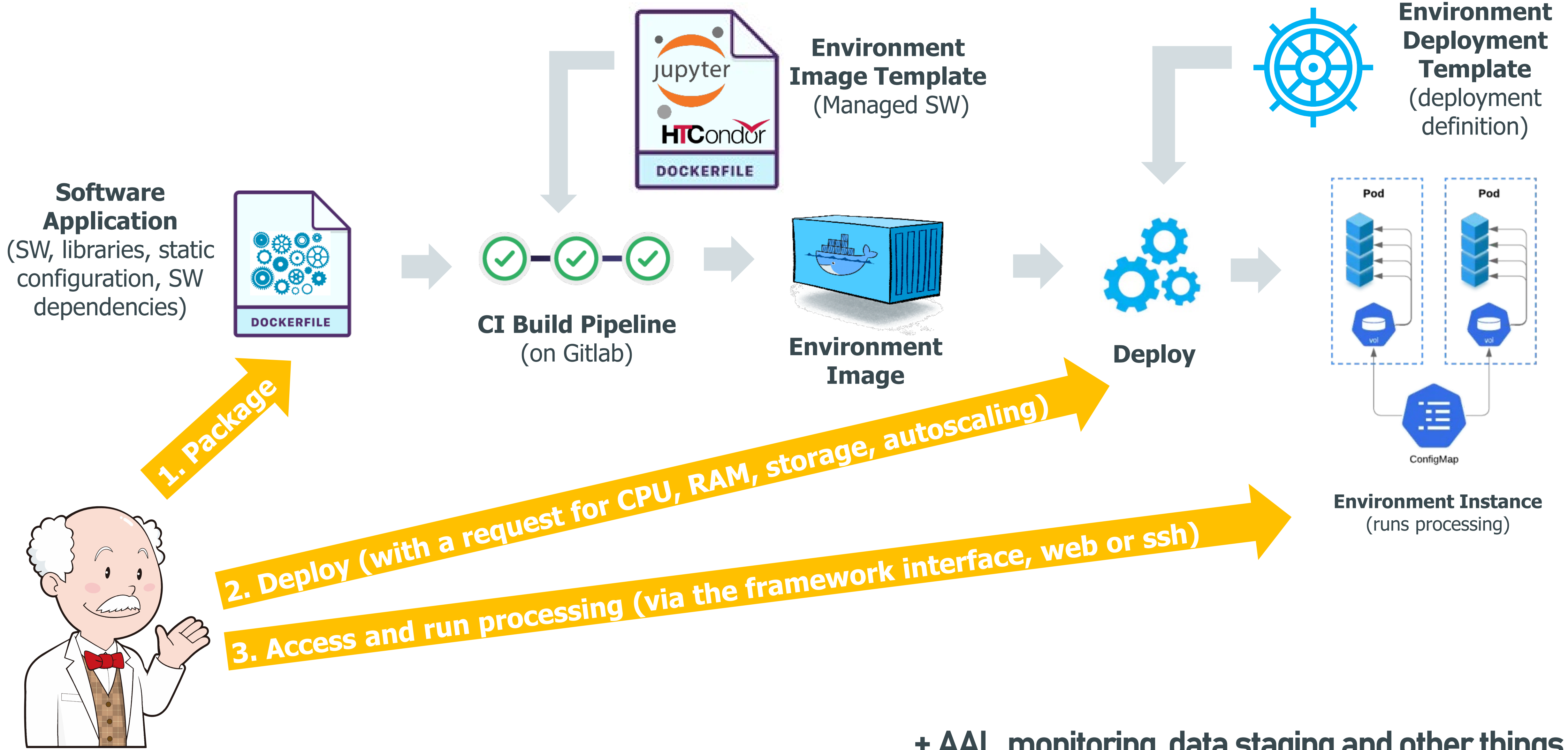
MME REP, serverless computing + more

MME REP (Multi-Mission Element for REProcessing) is the latest EUMETSAT system for bulk data processing (everything which cannot run on a single PC)

- Based on a **Kubernetes** infrastructure (& multiple K8S clusters)
  - Designed to scale by 2 orders of magnitude
  - 3 tiers of storage (performance/local -> bulk/shared)
    - + EUMETSAT Data Lake

- Includes tools to ease transition to serverless computing/containers:
  - **Automatic package and deployment of applications** (simplicity of use)
  - Pre-defined environment image templates (installing general SW)
    - JupyterHub, Interactive, Batch processing with HTCondor, …
  - Built-in security, automatic scaling, reliability and monitoring

MME REP, the idea behind



+ AAI, monitoring, data staging and other things

Agenda

The problem

Lots of missions, lots of different mission products
+ climate data processing, diversity of processors

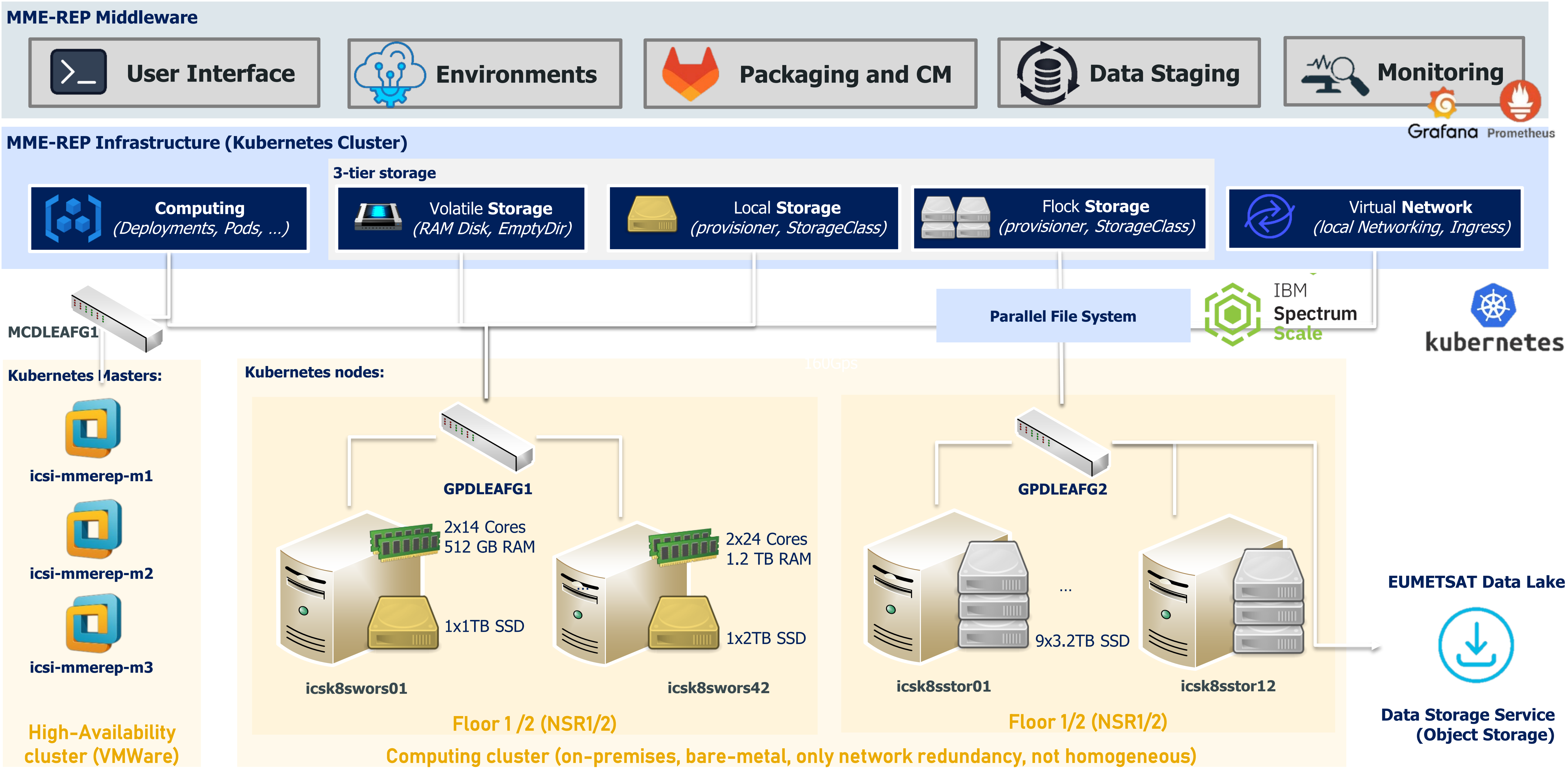
The solution

Transition to serverless computing, cloud and
container technologies

The system

What we have, how it works

MME REP design

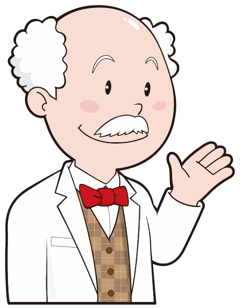


Security and access

www.eumetsat.int

- We do not run containers as "root"
  - Adds complexity in running some daemon-like software
  - We are experimenting on running Kubernetes root-less for this

- User management is done via the shared EUMETSAT AAI, integrated in Kubernetes



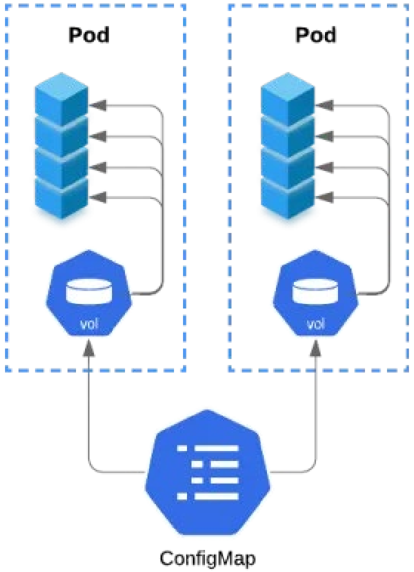
Access to Web services on environment (e.g. Jupyter)

Kubernetes Ingress

(plus AAI)

Access to command-line services on environments (e.g. SSH for job submission)

Kubernetes Exec communication channel (includes AAI)



Reprocessing is non-critical, so:

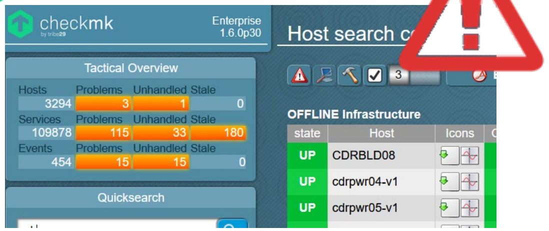
- No redundancy (except K8S core infrastructure)
- No strict availability commitments
  - Computing nodes can be down for scheduled or unscheduled maintenance for weeks
  - Loose requirement of no more than 5 nodes over 100 down for one week
  - NOTE: We do have redundancy for storage and overall network

How do we ensure smooth operations then?

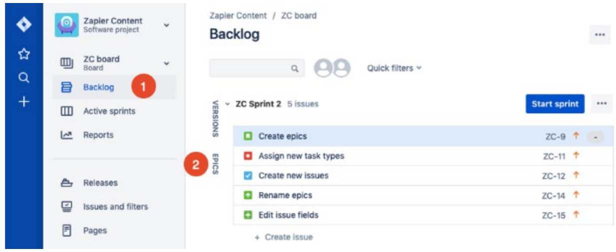
- Monitoring and (automatic) reaction

Active/Passive Monitoring

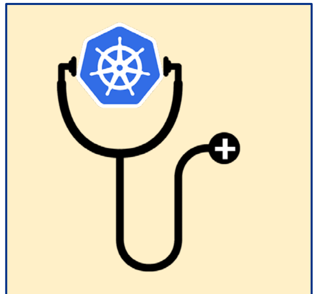
Hardware level



Hardware issue raised as JIRA ticket



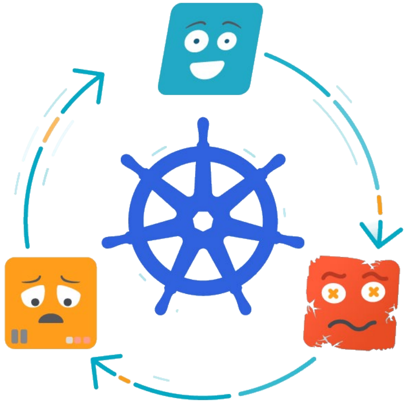
Kubernetes level



Liveliness probe

Software or node HW issue

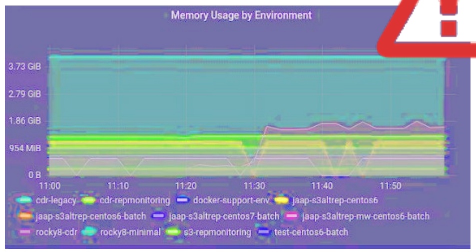
K8S Crash-Loop-Back-Off



Node restart

Environment level

Grafana Prometheus



MME-REP **monitoring of environments** (e.g. job resource usage, job restarts, job frozen)

Environment probes

HTCondor

Restart job (from breakpoint)

Processor level

Application specific monitoring (eq. quality of output products), defined by the users

App Logs

Traditional

Serverless

Scaling (is very easy and fast)

Environment Level

- User can scale the nodes (pod) vertically

- Nodes can scale horizontally **automatically** (e.g. for batch processing environments with full job queues), or manually

System Level

- Multiple Kubernetes Clusters sharing load

- Elasticity on the cloud (Kubernetes/Rancher can provision new nodes on the cloud)

An example (from the user of batch processing) (1/3)

Step 1. User push a (Dockerfile) and Gitlab CI configuration (.gitlab-CI.yml):

Build a htcondor-batch environment and an interactive environment

Processor Application (Dockerfile):

```
FROM centos:7

#Update packages and install basic package requirements
RUN yum update -y && yum install -y bash gawk sed python3 sqlite unzip zip rsync
 git && pip3 install pyyaml && yum clean all && rm -rf /var/cache/yum

#Copy S3 processors (with COTS into a dedicate layer, so that if the COTS stay t
he same, we do not need to re-deploy the entire image)
COPY s3ipf/cots /usr/local/cots
COPY s3ipf/components /usr/local/components
COPY s3ipf/conf /usr/local/conf

#Install S3 IPF command-line application (which we need to generate joborders)
COPY ipfcmd /usr/local/ipfcmd
~
~
~
~
~
~
~
~
~
"Dockerfile" 12L, 576C
```

Gitlab CI configuration (.gitlab-CI.yml):

```
stages:
  #Base image software tests to be implemented
  #- test
  - build

build:
  image: harbor.eumetsat.int/mme-rep/base/mme-rep-environment-tools:latest
  stage: build
  tags: [ "dind" ]
  variables:
    GIT_SUBMODULE_STRATEGY: recursive
  script:
  - /mmereptools/build/envbuild.sh -T htcondor-batch
  - /mmereptools/build/envbuild.sh -T interactive
~
~
~
~
~
~
~
~
~
".gitlab-ci.yml" 14L, 356C
```

Step 2. Automatic pipeline builds the Docker images for the environment

passed	#16802 latest	master -o- 9eb5e636 added old processor ...	✓
passed	#16783 latest	v0.0.1 -o- bdc7bea0 updated ipfcmd	✓

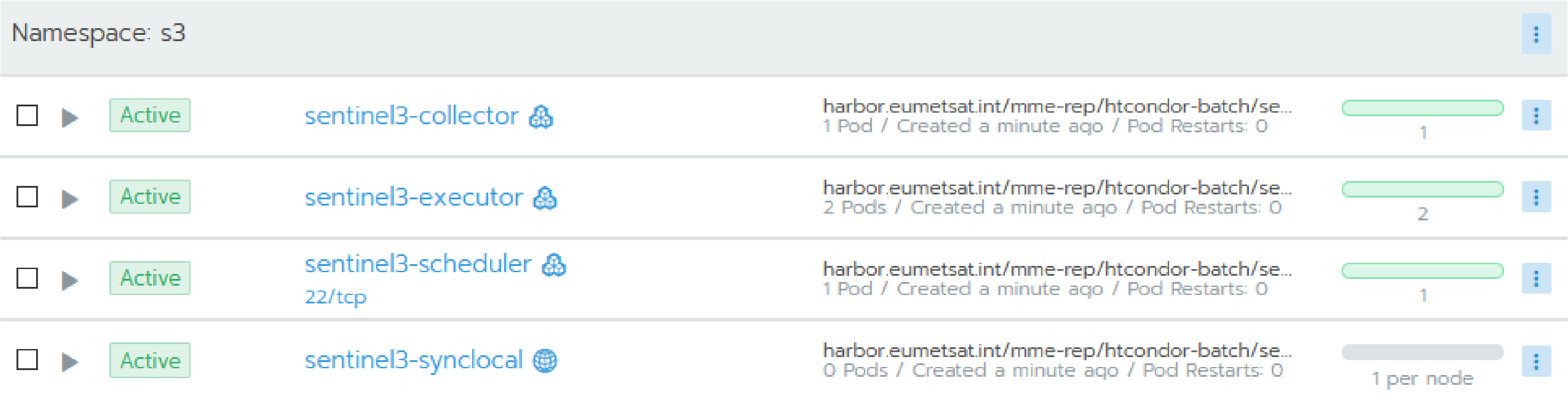
Git tags are mapped to Docker image tags

An example (from the user of batch processing) (2/3)

Step 3. Deploy environment

Operator deploys htcondor-batch environment using sentinel3 base image version v0.0.1 build in the previous step

```
!$ ./mme-rep.sh env deploy -F dev-s3 htcondor-batch sentinel3:v0.1.0
```

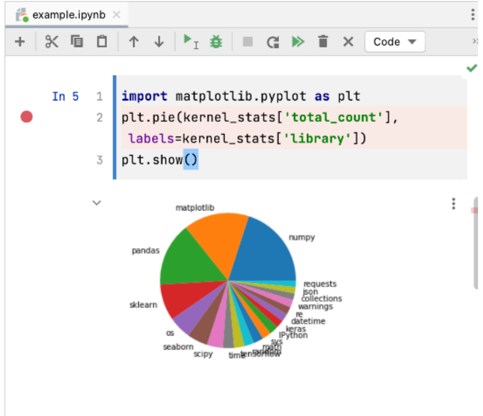


Step 4. Access environment

Environment is deployed and can be accessed via SSH (for batch environments, using kubectl as a ProxyCommand)

```
!$ ssh sentinel2-scheduler.dev-s3.mmerep-general-dev
```

*If you deployed a web environment, like Jupyter, you would get a web address to connect



An example (from the user of batch processing) (3/3)

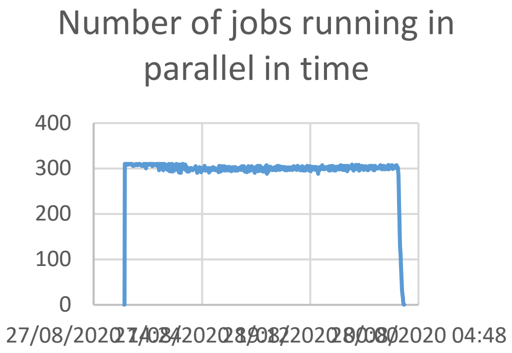
www.eumetsat.int

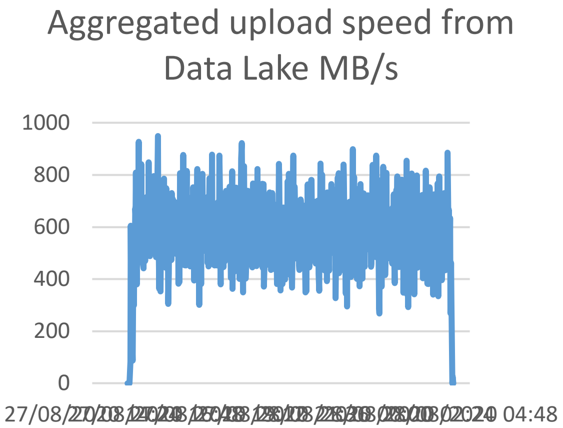
Step 6. Scale environment

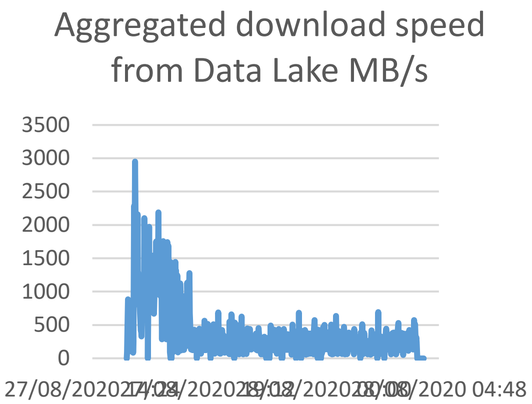
You can scale environments vertically or horizontally, manually or automatically, from a console or web interface

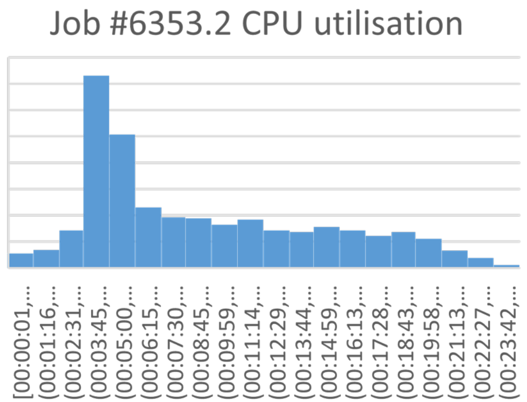
```
[climproc@sentinel3-scheduler-847dc59d-56tpf ~]$ #See the currently assigned resources
[climproc@sentinel3-scheduler-847dc59d-56tpf ~]$ /configs/scaler.sh
Daemon is: off
Cluster size:     2 (0 busy nodes)
Environment size: 2
Nodes are set for:
  Requests: "1" CPU / "10G" RAM
  Max: "" CPU / "" RAM
[climproc@sentinel3-scheduler-847dc59d-56tpf ~]$ #Scale vertically to 2 CPU and 8GB per processing node
[climproc@sentinel3-scheduler-847dc59d-56tpf ~]$ /configs/scaler.sh vscale 2 8G
Scaling deployment vertically to CPU/RAM requests 2/8G and limits / ...

[climproc@sentinel3-scheduler-847dc59d-56tpf ~]$ #Scale horizontally to 310 processing nodes
[climproc@sentinel3-scheduler-847dc59d-56tpf ~]$ /configs/scaler.sh scale 310
Scaling deployment up from 2 to 310...
```

Step 7. Monitor processing









- Deploy "anything we want" ("exotic" dependencies), when we need it
- Simple for the scientist (whoever wants just a batch cluster can still get it)
  - ✗ Not so simple for the service provider
- Performance
  - very lean virtualization and limited OS overhead (container vs VM)
  - 3-tier storage improves I/O demanding applications (most of our SW)
  - ✗ we had to write a custom K8S provisioner to fully exploit the local SSD storage
- Cheaper & easier to scale
  - ✚ scales on anything you can get your hands on (local resources, private/commercial cloud, …)
  - ✗ handling finite resource allocation conflicts is not as mature as in a batch processing cluster
- ✚ Better control of what's running (Gitlab, Tags, Container images, security scans)
- Monitoring at deep level allows more reliability and better tuning
  - ✚ Automatic restarts, easier recovery (you can "reinstall" in one click)

Thank you!

Questions are welcome.

Contacts:

Mike Grant – Michael.Grant@eumetsat.int

Fernando Ibanez – Fernando.Ibanez@eumetsat.int

Salvatore Pinto – Salvatore.Pinto@eumetsat.int