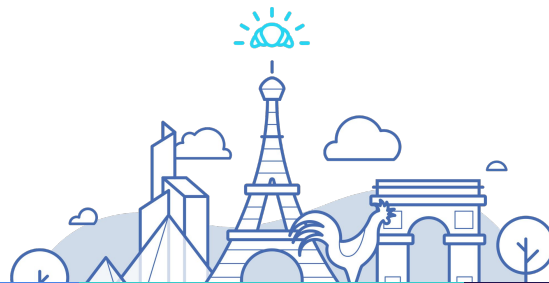


Ingestion de l'univers visible dans une base de données distribuée Cloud-Native

Speaker(s) :

Fabrice Jammes

Gabriele Mainetti



1. The Vera C. Rubin Observatory
2. The Rubin Science Platform
3. Qserv: the petascale Database
4. Cloud-Native: Kubernetes Operators
5. Cloud-Native: Workflows
6. What we have learnt



The Rubin Observatory

Telescope, catalogs and the IN2P3 role

The Vera C. Rubin Observatory

A revolutionary telescope
The **largest digital camera** in the world

Cerro Pachón at Chili (2647m slm)
8.4m primary mirror
3.2 Gpixel camera

Three main Data Facilities: USDF, UKDF and FrDF
(@ CC-IN2P3)

Funding
~\$1 billion, 20% dedicated to data management

Main Objective:
Define the nature of dark energy and dark matter



The largest astronomical catalog

Raw Data: 20TB/night



Sequential 30s images covering the entire visible sky every few days



Prompt Data Products

Alerts: up to 10 million per night

Raw & Processed Visit Images, Difference Images, Templates

Transient and variable sources from Difference Image Analysis

Solar System Objects: ~ 6 million

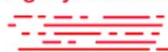
Data Release Data Products

Final 10yr Data Release:

- Images: 5.5 million x 3.2 Gpixels
- Catalog: 15PB, 37 billion objects



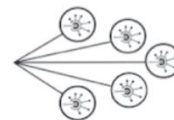
via nightly alert streams



via Prompt Products DB



via Data Releases



Community Brokers

Rubin Data Access Centres (DACs)

USA (USDF)
Chile (CLDF)
France (FRDF)
United Kingdom (UKDF)

Independent Data Access Centers (IDACs)

Source: Rubin Observatory

LSST will produce a catalog of **37 billion galaxies and stars** and their associated physical properties, i.e. **2 EB** of data

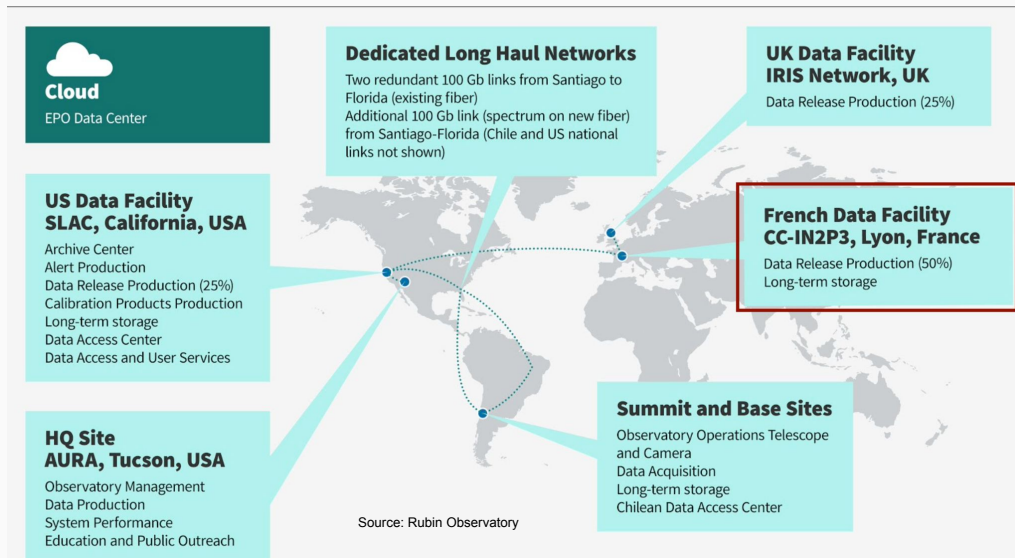
The IN2P3 Role

Development of Qserv and integration with Kubernetes

F. Jammes

CC-IN2P3 is the FrDF

50% of data produced by the telescope will be processed



900 Pb of data

30 Pb for the astronomical catalog

The CC-IN2P3 dedicated infrastructure

K8S physical cluster dedicated to Rubin:

- Shared by 2 applications: **Qserv** and **Rubin Science Platform**
- 25 workernodes
- 5 DELL PowerEdge R440
- 20 DELL PowerEdge R540
- 3 K8S Masters (R440)
- 2 Qserv Masters (R440)
- 15 Qserv Workers (R540)
- 5 Rubin Science Platform Workers (R540)
- QSERV nodes selected via taint

K8S OpenStack (small) cluster for test purpose:

- Shared by 2 applications: Qserv and Rubin Science Platform
- 8 VM
- 3 K8S masters
- 1 Qserv Master
- 3 Qserv Workers (1T of local storage per worker)
- 1 RSP Worker

R540:

- 40 Intel Xeon Silver 4210 CPU @ 2.20GHz
- 256 GB of RAM
- 50 TB of local storage

R440:

- 40 Intel Xeon Silver 4114 CPU @ 2.20GHz
- 256 GB of RAM
- 18 TB of local storage

Clusters managed via Puppet and Ansible

- Puppet used to configure the machines
- Ansible used to manage K8S: install, configuration etc.

Caddy web server to expose data to Qserv

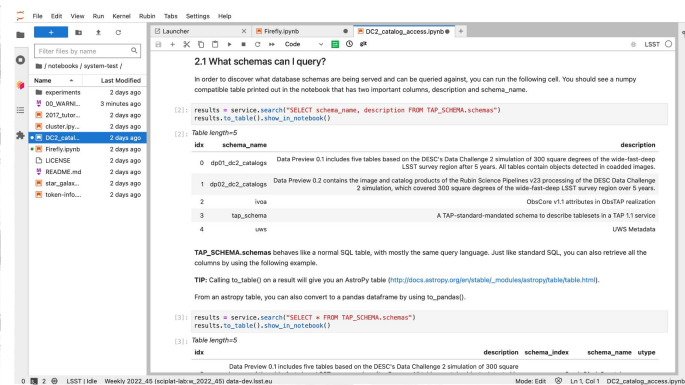
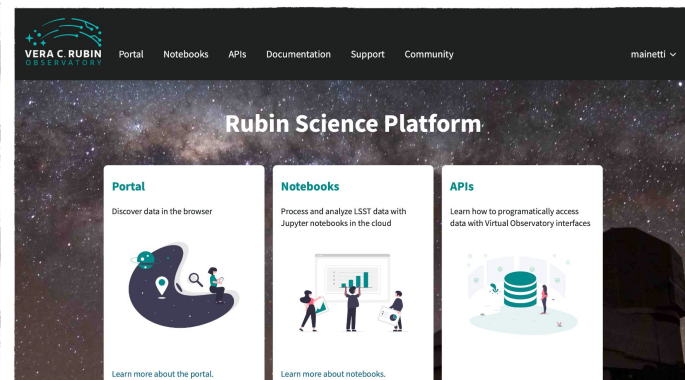
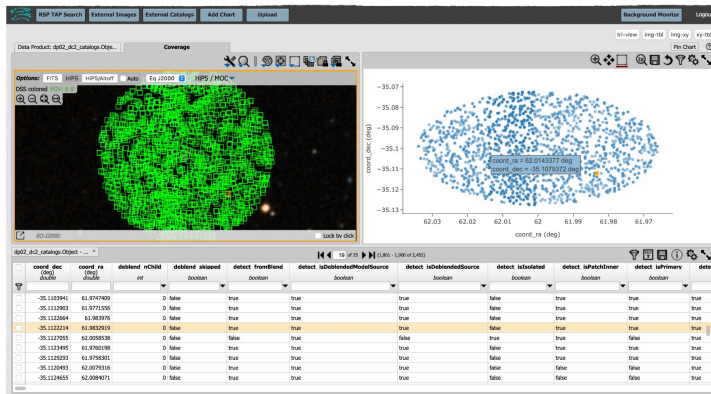
- 4 worker nodes

The Rubin Science Platform

The Rubin Science Platform (RSP)

Web environment for interactive data analysis

- Qserv catalogs access via VO Table Access Protocol (UI or script)
- Table visualization and exploitation, graph, images visualization
- Access to external astronomical catalogs (Gaia, Planck, ...) and images (SDSS, 2MASS, ...)
- Integrated Jupyter platform (Python API)
- Qserv access portal for all the Virtual Observatory components (Topcat, DS9, Aladdin,...)



RSP Configuration and Deployment

- Configuration via Helm Chart
- A config for each environment (e.g. CC-IN2P3, SLAC, UKDF, ...) and a config for each application

```
config:
  gcsBucket: "async-results.lsst.codes"
  gcsBucketUrl: "https://cccephs3.in2p3.fr:8080"
  gcsBucketType: "S3"
  jvmMaxHeapSize: "31G"

qserv:
  host: "ccqserv201.in2p3.fr:30040"
  mock:
    enabled: false
```

- Deployed via ArgoCD
- HashiCorp Vault used to manage secrets

<https://github.com/lsst-sqre/phalanx>

<https://phalanx.lsst.io/>

templates	Clean up the sqlproxy service
Chart.yaml	Fix typo in science-platform Chart.yaml
README.md	Updated missed values to reflect new naming
values-base.yaml	Delete obstap service
values-ccin2p3.yaml	activate datalinker
values-idfdev.yaml	adjusted naming to be more generic
values-idfint.yaml	Update values-idfint.yaml
values-idfprod.yaml	Delete obstap service

The screenshot shows the Argo CD web interface with a grid of application tiles. The tiles are:

- argocd/argocd**: Project: default, Labels: argocd, argoproj.io/instance-science-platform, Status: Healthy Synced, Repository: https://github.com/gabirmarine/phalanx.git, Target Rev.: ccin2p3, Path: services/argocd, Destination: in-cluster, Namespace: argocd.
- argocd/cachemachine**: Project: default, Labels: argocd, argoproj.io/instance-science-platform, Status: Healthy Synced, Repository: https://github.com/gabirmarine/phalanx.git, Target Rev.: ccin2p3, Path: services/cachemachine, Destination: in-cluster, Namespace: cachemachine.
- argocd/cert-manager**: Project: default, Labels: argocd, argoproj.io/instance-science-platform, Status: Healthy Synced, Repository: https://github.com/gabirmarine/phalanx.git, Target Rev.: ccin2p3, Path: services/cert-manager, Destination: in-cluster, Namespace: cert-manager.
- argocd/datalinker**: Project: default, Labels: argocd, argoproj.io/instance-science-platform, Status: Healthy Synced, Repository: https://github.com/gabirmarine/phalanx.git, Target Rev.: ccin2p3, Path: services/datalinker, Destination: in-cluster, Namespace: datalinker.
- argocd/gafsealr**: Project: default, Labels: argocd, argoproj.io/instance-science-platform, Status: Healthy Synced, Repository: https://github.com/gabirmarine/phalanx.git, Target Rev.: ccin2p3, Path: services/gafsealr, Destination: in-cluster, Namespace: gafsealr.
- argocd/ingress-nginx**: Project: default, Labels: argocd, argoproj.io/instance-science-platform, Status: Healthy Synced, Repository: https://github.com/gabirmarine/phalanx.git, Target Rev.: ccin2p3, Path: services/ingress-nginx, Destination: in-cluster, Namespace: ingress-nginx.

Qserv

The Petascale database

International context



Data Access and Database (DAX)



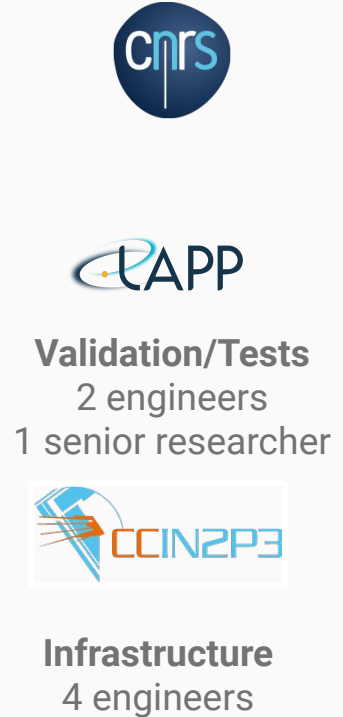
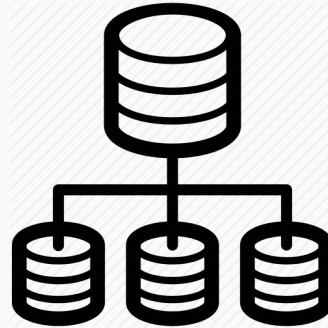
Devops
6 engineers



**Google Cloud
Infrastructure**
3 engineers

R&D
7 engineers

QSERV

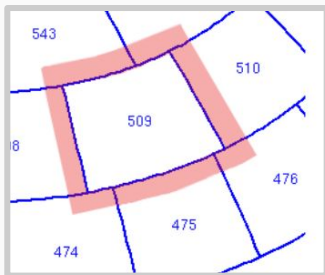
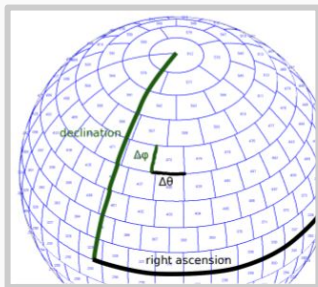


Qserv design



Relational database, 100% open source

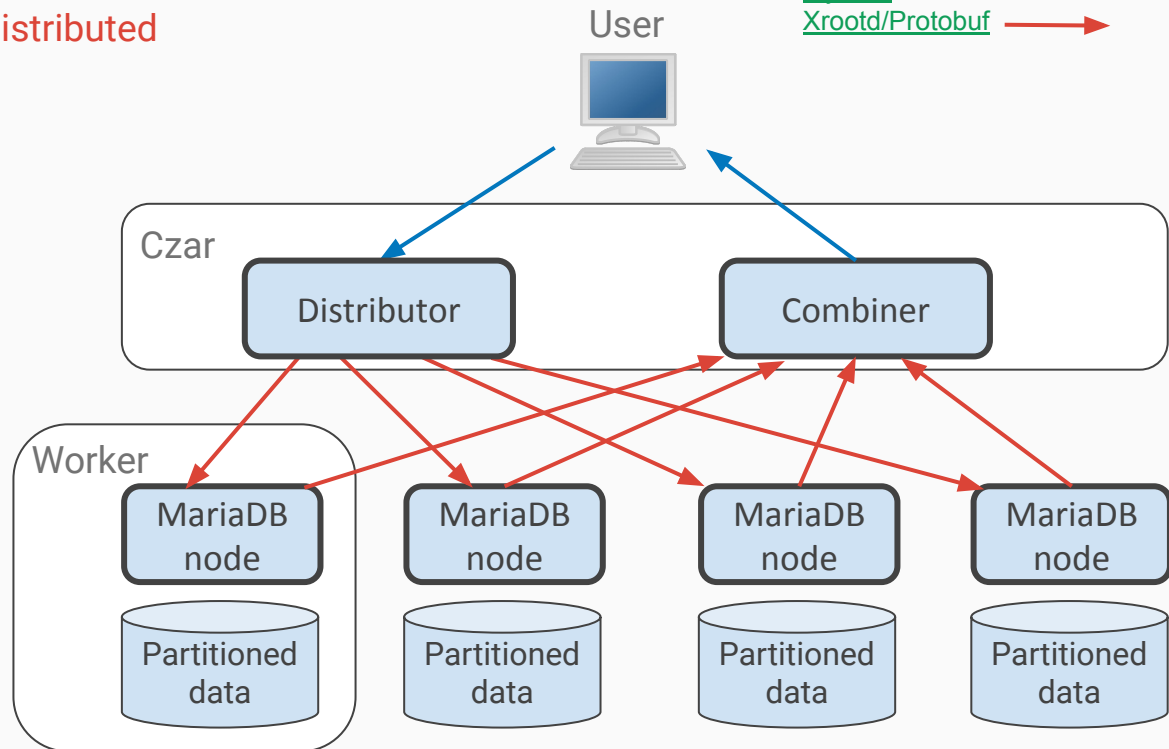
Spatially-sharded with overlaps

Map/reduce-like processing, highly distributed



Legend:

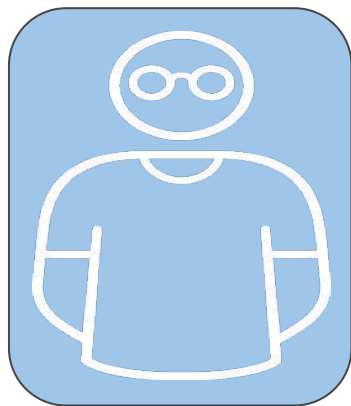
MySQL 
Xrootd/Protobuf 



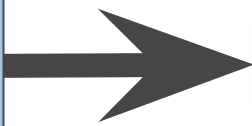
~1000 workers, 20 chunks/5TB per workers

Cloud-Native Kubernetes operators

Operators embed ops knowledge from the experts

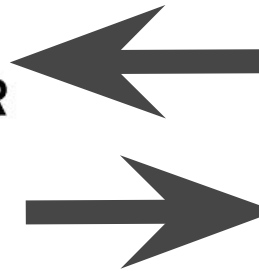


ops knowledge from the experts



**OPERATOR
SDK**

operator
implementation
i.e. k8s controller



K8s standard API:
Deployments
StatefulSets
Autoscalers
Secrets
Config maps

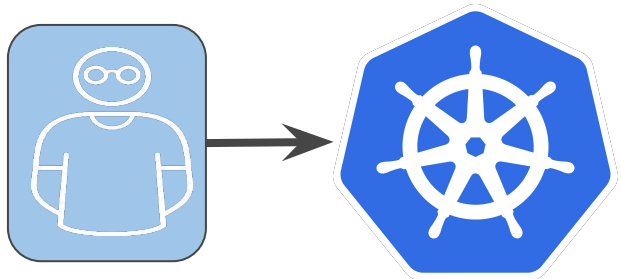
See

- <https://kubernetes.io/docs/concepts/extend-kubernetes/operator/>
- <https://cloud.google.com/blog/products/containers-kubernetes/best-practices-for-building-kubernetes-operators-and-stateful-apps>

Credits: Daniel Messer Product Manager, OpenShift - Guilherme Barros Product Manager, Cloud BU



How does an operator works?



Allow to deploy a complex application with only a few lines of yaml

Software Developer
Kubernetes user

K8s API



Kubernetes operator

Native Kubernetes
resources

Custom resource

```
apiVersion: qserv.lsst.org/v1alpha1
kind: Qserv
metadata:
  name: qserv
  namespace: database
spec:
  czar:
    image: qserv/lite-qserv:2021.10.1-rc1
  replicas: 1
  storage: 1Ti
  worker:
    image: qserv/lite-qserv:2021.10.1-rc1
  replicas: 10
  ...
```

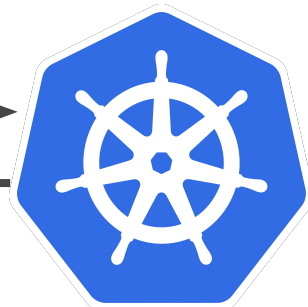
Custom Kubernetes controller

Watch Event

Reconcile

Custom resource definition

here Qserv



Deployments
StatefulSets
Autoscalers
Secrets
Config maps

What brings the Qserv operator?

Operators: both sysadmin + application experts

🔗 Easy Qserv install

On-prem, Cloud, CI, laptop

🔗 Easy configuration

Based on Qserv CRD

Admission Control Plugin:

manage default values and

update to configuration

🔗 Automated upgrade

Including database schemas

Install qserv-operator

```
kubectl apply -f manifests/operator.yaml
```

Install a qserv instance

```
kubectl apply -k manifests/base
```

Monitor Qserv instance and related Pods

```
fjammes@clrinfopo18 ~/src/qserv-operator main kubectl get qservs.qserv.lsst.org
```

NAME	CZAR	INGEST-DB	REPL-CTL	REPL-DB	REPL-REGISTRY	WORKER	XROOTD	AGE
qserv	1/1	1/1	1/1	1/1	1/1	2/2	1/1	28m

```
fjammes@clrinfopo18 ~/src/qserv-operator main kubectl get pod
```

NAME	READY	STATUS	RESTARTS	AGE
qserv-czar-0	2/2	Running	0	28m
qserv-ingest-db-0	1/1	Running	0	28m
qserv-repl-ctl-0	1/1	Running	0	28m
qserv-repl-db-0	1/1	Running	0	28m
qserv-repl-registry-78cc7b9cc-2jh2h	1/1	Running	0	28m
qserv-worker-0	4/4	Running	0	28m
qserv-worker-1	4/4	Running	0	28m
qserv-xrootd-redirector-0	2/2	Running	0	28m

Full demo <https://asciinema.org/a/uWCPVytEy2ravJs27aqj2UqpB>

Automated deployment: On Prem

Qserv repository



Build/Validate/Push Qserv images





GitHub Actions



Docker Registry

Legend:

CI Workflow 
Containers images 



Infrastructure:
IN2P3 Datacenter

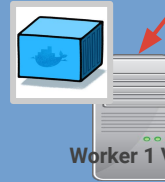


Storage:
~ 45TB Catalogs
Local disks
(DAS)

QSERV operator



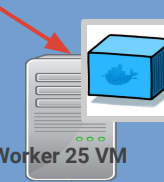
Master VM



Worker 1 VM



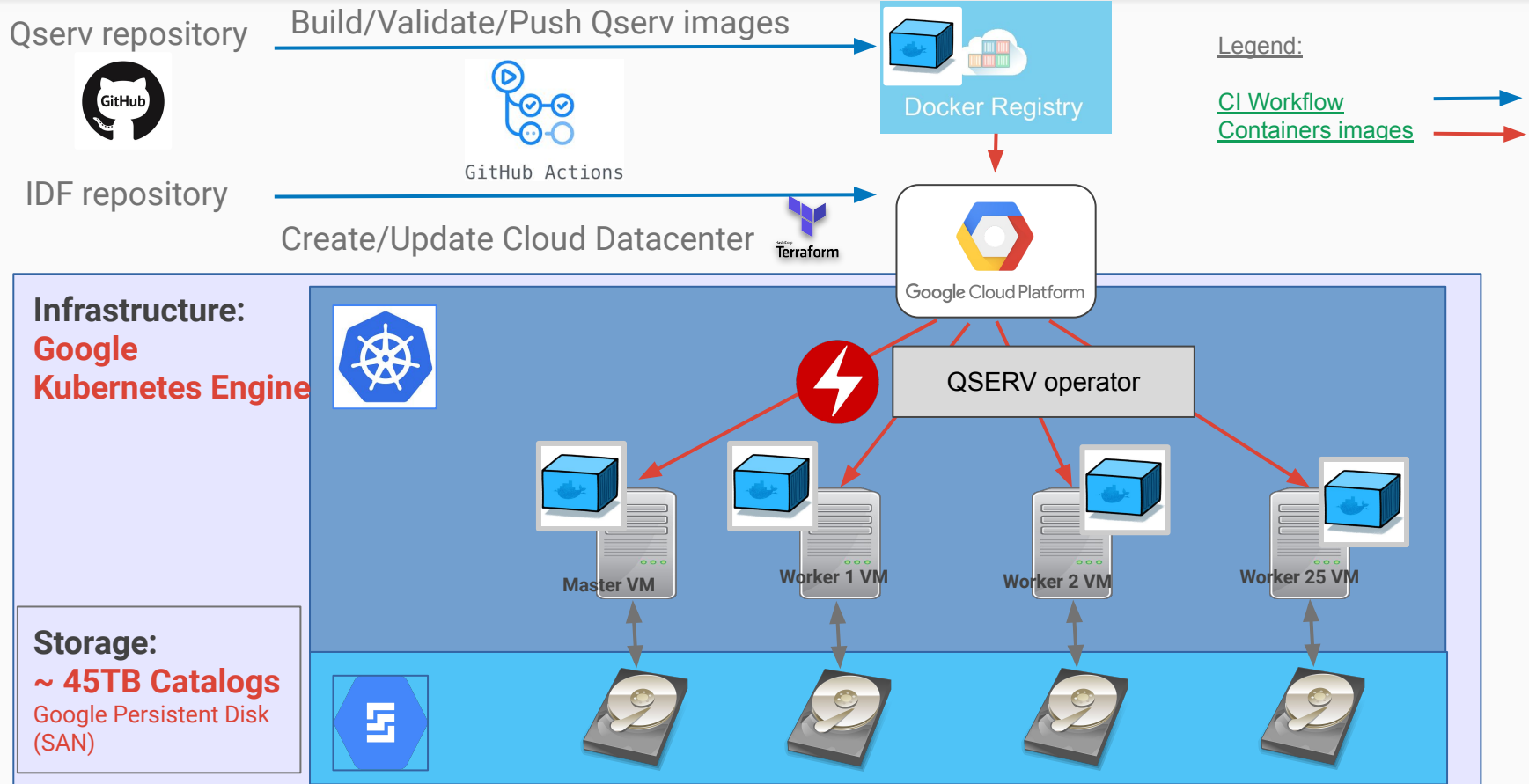
Worker 2 VM



Worker 25 VM



Automated deployment: Cloud Native

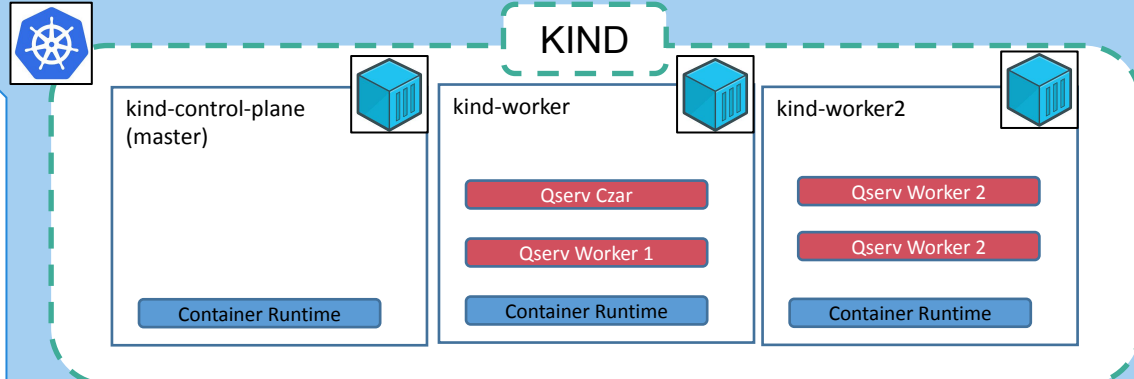


CI Setup with KIND



For each commit

- Build Qserv image
- Start kind
- Start Qserv
- Launch integration tests
- Push image to registry



Docker runtime



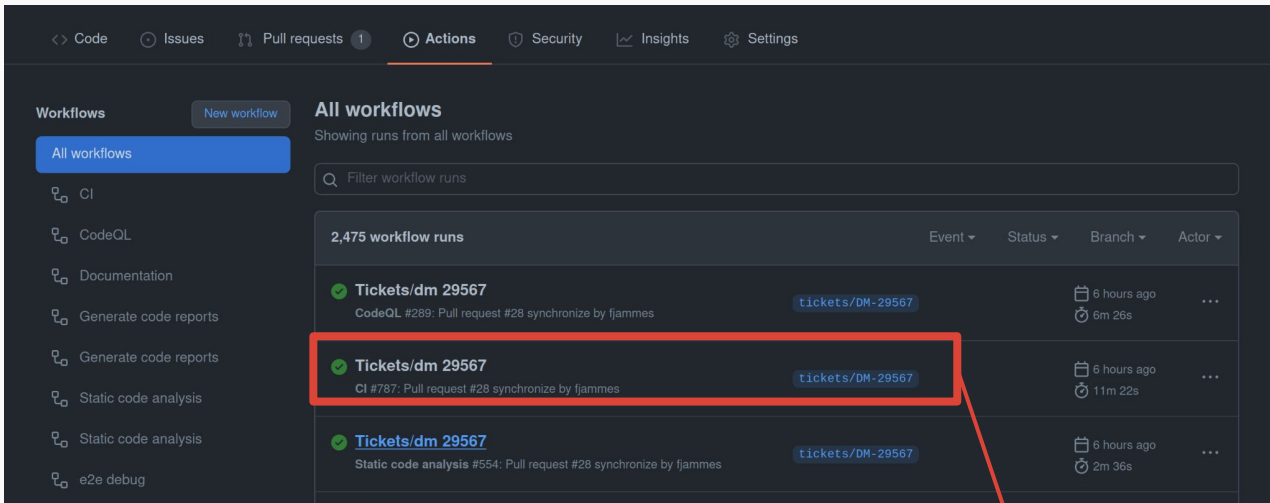
QSERV operator

Github Action VM

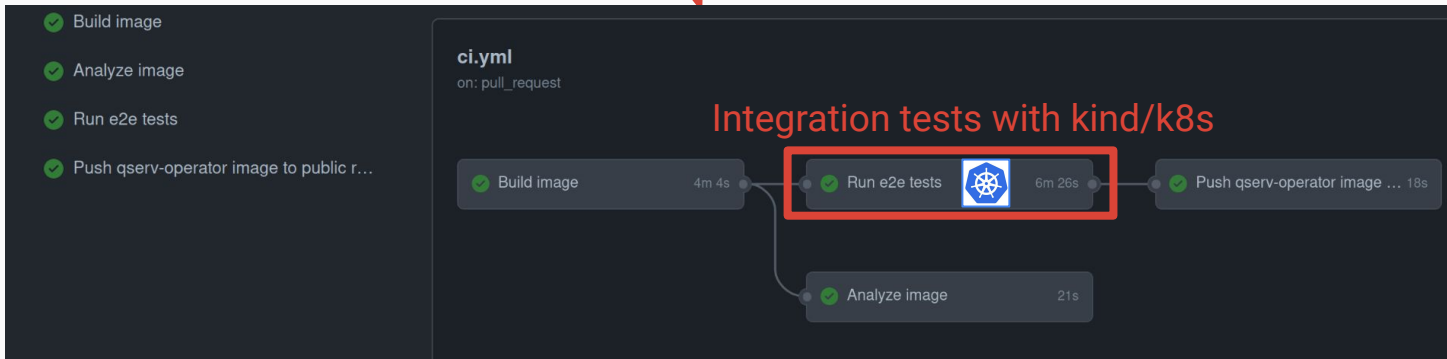
<https://kind.sigs.k8s.io/>

Wrapper for CI: <https://github.com/k8s-school/kind-helper>

CI in practice: Qserv integration tests



The screenshot shows the GitHub Actions interface. The top navigation bar includes links for Code, Issues, Pull requests (1), Actions (selected), Security, Insights, and Settings. On the left, a sidebar lists various workflow categories: CI, CodeQL, Documentation, Generate code reports, Static code analysis, and e2e debug. The main area is titled "All workflows" and shows a list of 2,475 workflow runs. A red box highlights a specific run: "Tickets/dm 29567" (CI #787: Pull request #28 synchronize by fjammes), which is 6 hours old and took 11m 22s to complete. An arrow points from this run to the detailed workflow view below.



The screenshot displays a workflow diagram for a file named `ci.yml` triggered on `pull_request`. The workflow consists of several steps: "Build image" (4m 4s), "Analyze image" (21s), "Run e2e tests" (6m 26s), and "Push qserv-operator image ..." (18s). The "Run e2e tests" step is highlighted with a red box and labeled "Integration tests with kind/k8s". The diagram also shows a list of workflow runs on the left, including "Build image", "Analyze image", "Run e2e tests", and "Push qserv-operator image to public r...".

Cloud-Native Workflows

A powerful data ingest workflow

Qserv has a powerful distributed ingest algorithm

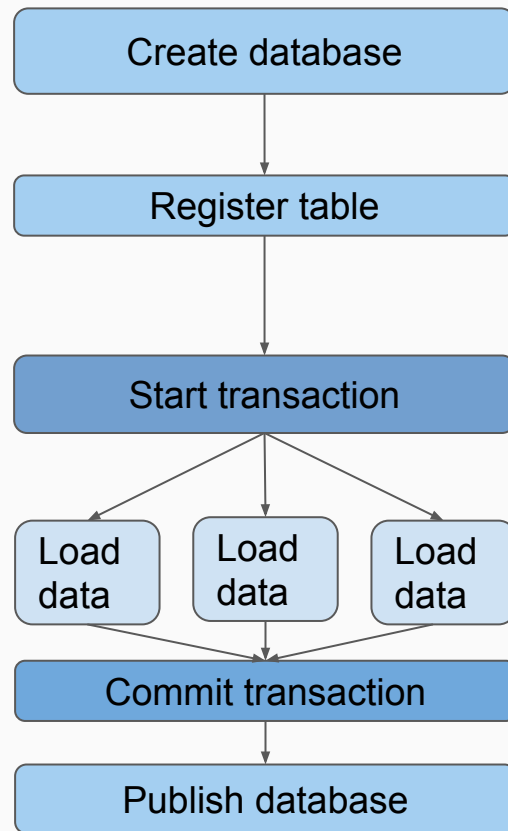
Flexible but require orchestrating tasks (DAG)

Argo Workflow project help us a lot



Case study 2022: Implementation of a large-scale data loading algorithm

Ingestion of 2M files and ~40TB in 5h

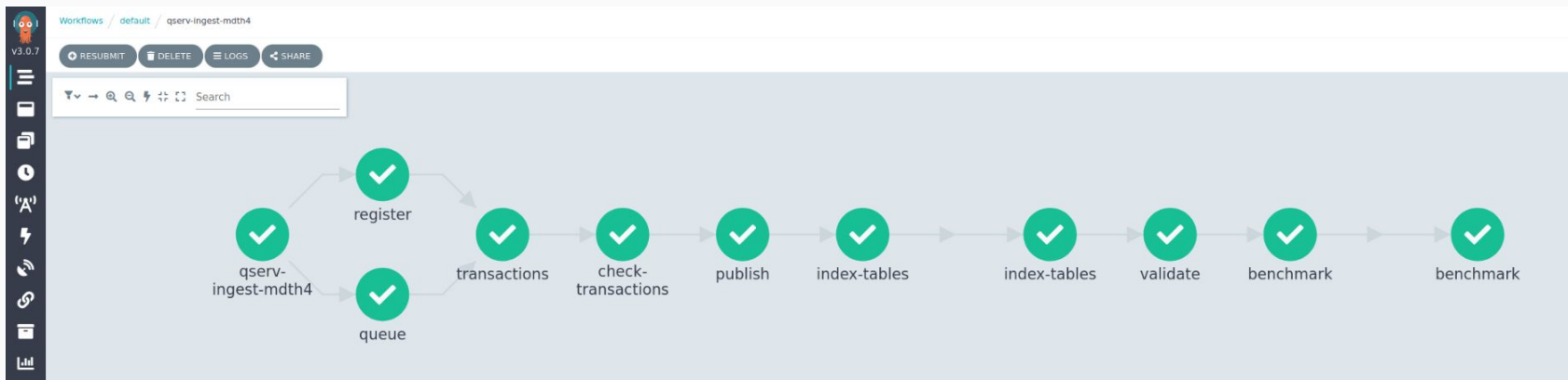


Argo: screenshots

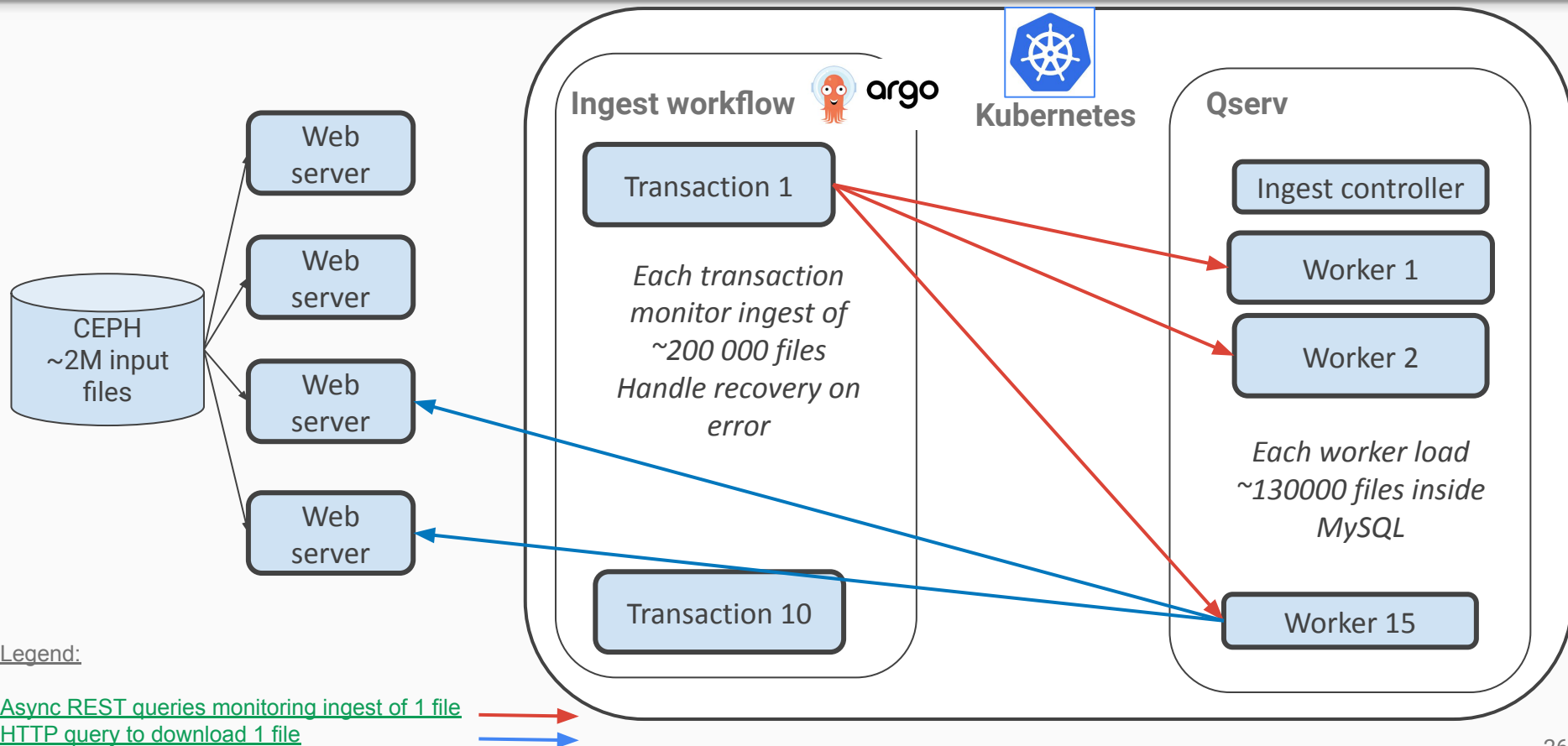


```
fjammes@clrinport18 ~$ argo get @latest | tail -n 15
```

STEP	TEMPLATE	PODNAME	DURATION
✓ qserv-ingest-mdth4	main		
└─ ✓ queue	ingest-step	qserv-ingest-mdth4-2075476264	3s
└─ ✓ register	ingest-step	qserv-ingest-mdth4-964548720	4s
└─ ✓ transactions	transactions	qserv-ingest-mdth4-1041421248	14s
└─ ✓ check-transactions	ingest-step	qserv-ingest-mdth4-3195504171	2s
└─ ✓ publish	ingest-step	qserv-ingest-mdth4-4256901816	12s
└─ ✓ index-tables	index-tables		
└─└─ ✓ index-tables	ingest-step	qserv-ingest-mdth4-1866502525	2s
└─ ✓ validate	ingest-step	qserv-ingest-mdth4-493206715	2s
└─ ✓ benchmark	benchmark		
└─└─ ✓ benchmark	ingest-step	qserv-ingest-mdth4-1797710727	5s



Technical details about ingest process



Qserv catalogs now

Catalog	Size (TB)	# of lines (Billions)
DP0.2	36.6	139
DP0.1	1.1	1.7
SkySim 5000	13.6	20.5
Cosmo DC2	3.7	5.5

Database	Data [GB]																					
	#chunks		in unique chunks										in all replicas									
			chunks		overlaps			regular					chunks		overlaps			regular				
	unique	replicas	data	index	Σ	data	index	Σ	data	index	Σ	Σ	data	index	Σ	data	index	Σ	data	index	Σ	Σ
cosmoDC2_v1_1_4_image	1730	1744	3569.4	69.4	3638.7	41.9	<0.1	41.9	0.0	0.0	0.0	3680.7	3569.4	69.4	3638.7	41.9	<0.1	41.9	0.0	0.0	0.0	3680.7
dp01_dc2_catalogs	1398	1412	915.3	58.9	974.2	114.3	<0.1	114.3	0.0	0.0	0.0	1088.5	915.3	58.9	974.2	114.3	<0.1	114.3	0.0	0.0	0.0	1088.5
dp02_dc2_catalogs	1478	1492	31746.3	2737.4	34483.7	2138.3	<0.1	2138.3	0.0	0.0	0.0	36622.0	31746.3	2737.4	34483.7	2138.3	<0.1	2138.3	0.0	0.0	0.0	36622.0
skysim5000_v1_1_1_parquet	18738	18752	13171.2	261.5	13432.7	157.9	<0.1	158.0	0.0	0.0	0.0	13590.7	13171.2	261.5	13432.7	157.9	<0.1	158.0	0.0	0.0	0.0	13590.7
Total [TB for data]	23344	23400	49.4	3.1	52.5	2.5	<0.1	2.5	0.0	0.0	0.0	55.0	49.4	3.1	52.5	2.5	<0.1	2.5	0.0	0.0	0.0	55.0

1 Keep operator simple

*Handle complexity inside application
And expose simple management API
to operator*

2 Scale up with asynchronous queries

*Easier to handle network failures which
Always occur at scale*

3 Error recovery is a MUST HAVE

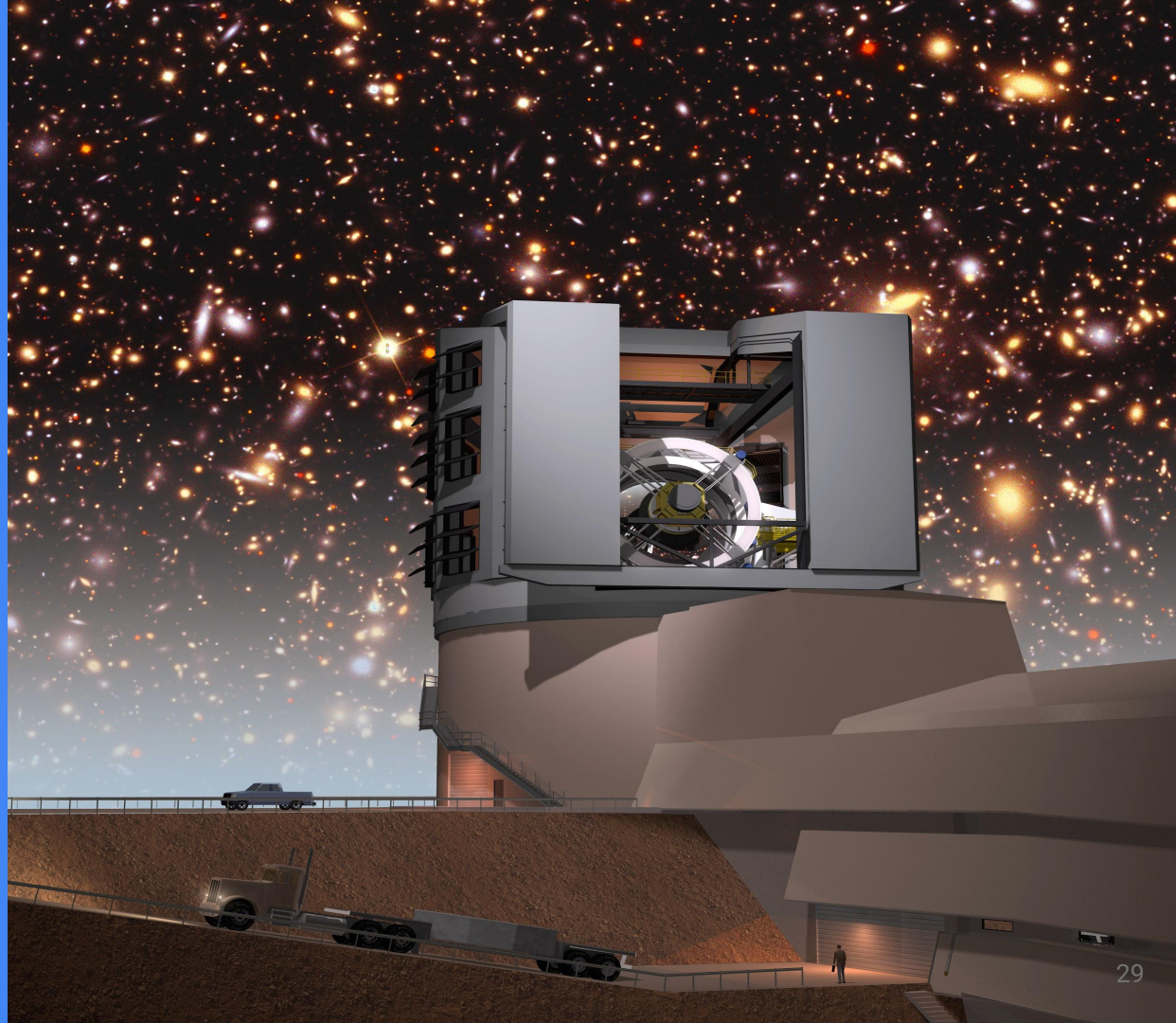
Transaction and Mariadb partitioning

What we
have learnt

Q&A

Gabriele MAINETTI
Centre de Calcul de l'IN2P3

Fabrice JAMMES
Laboratoire de Physique de
Clermont



Contact us!



Fabrice Jammes

<https://www.k8s-school.fr>



Gabriele Mainetti

gabrielemainetti [AT] in2p3.fr