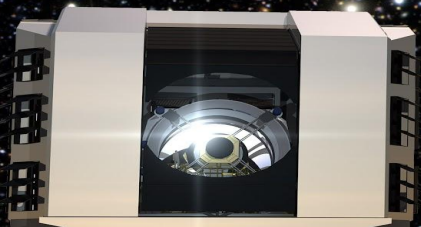


Large Synoptic Survey Telescope From Cloud-Native to Dark Energy



Fabrice Jammes

Scalable Data Systems Expert

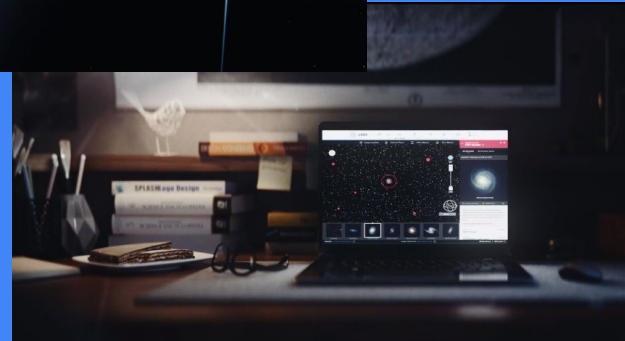
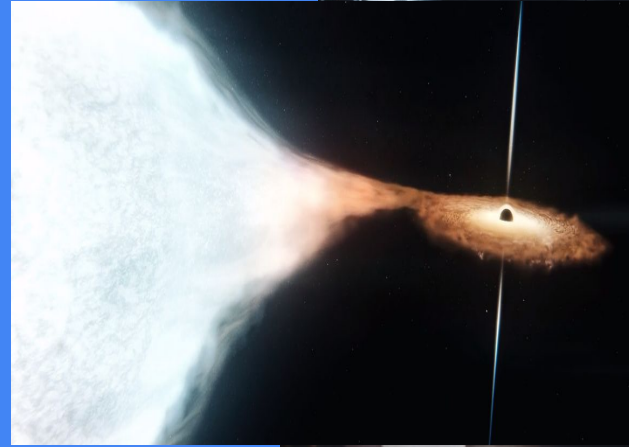
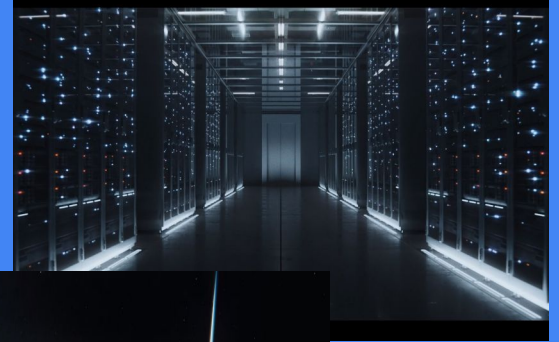
IN2P3 - Laboratoire de Physique de Clermont

Fritz Mueller

Technical manager

SLAC National Accelerator Laboratory

- 1 Large Synoptic Survey Telescope
- 2 The largest astronomical catalog
- 3 Cloud-Native: Kubernetes
- 4 Cloud-Native: Gitops & CI
- 5 Cloud-Native: Kubernetes Operators
- 6 Cloud-Native: Storage management
- 7 Cloud-Native: Workflows



A project that makes you dream

A revolutionary telescope

The **largest digital camera** in the world

The **largest celestial catalogs** ever made

Funding

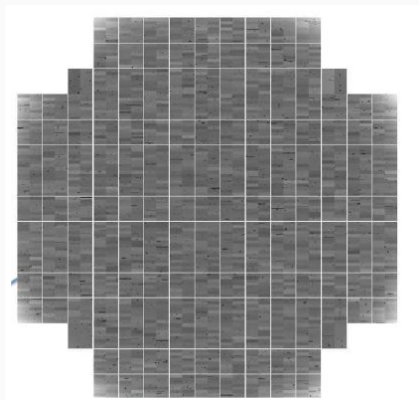
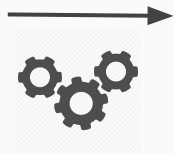
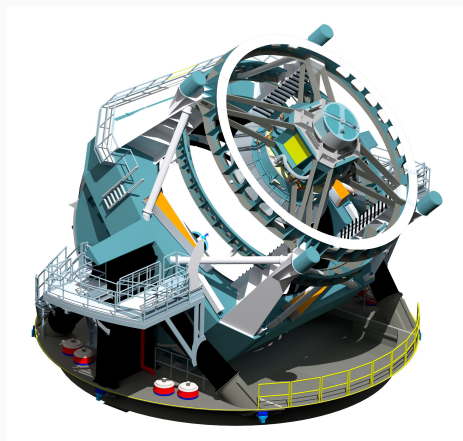
~\$1 billion, 20% dedicated to data management

Key role of CNRS/IN2P3

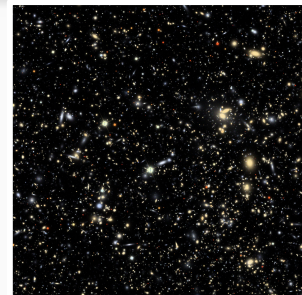
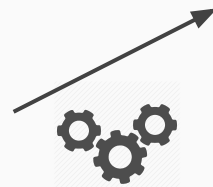
Objective:
Define the nature of dark energy



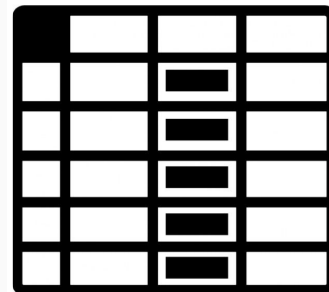
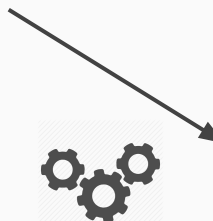
The largest astronomical catalog



Raw data



Processed image



Catalog (stars, galaxies, objects, sources, transients, exposures, etc.)

LSST will produce a catalog of **40 billion galaxies and stars** and their associated physical properties, i.e. **500 PB** of data

Data

Images

Persisted: **~38 PB**

Temporary: **~½ EB**



- ★ **~3 million “visits”**
- ★ **~47 billion “objects”**
- ★ **~9 trillion “detections”**

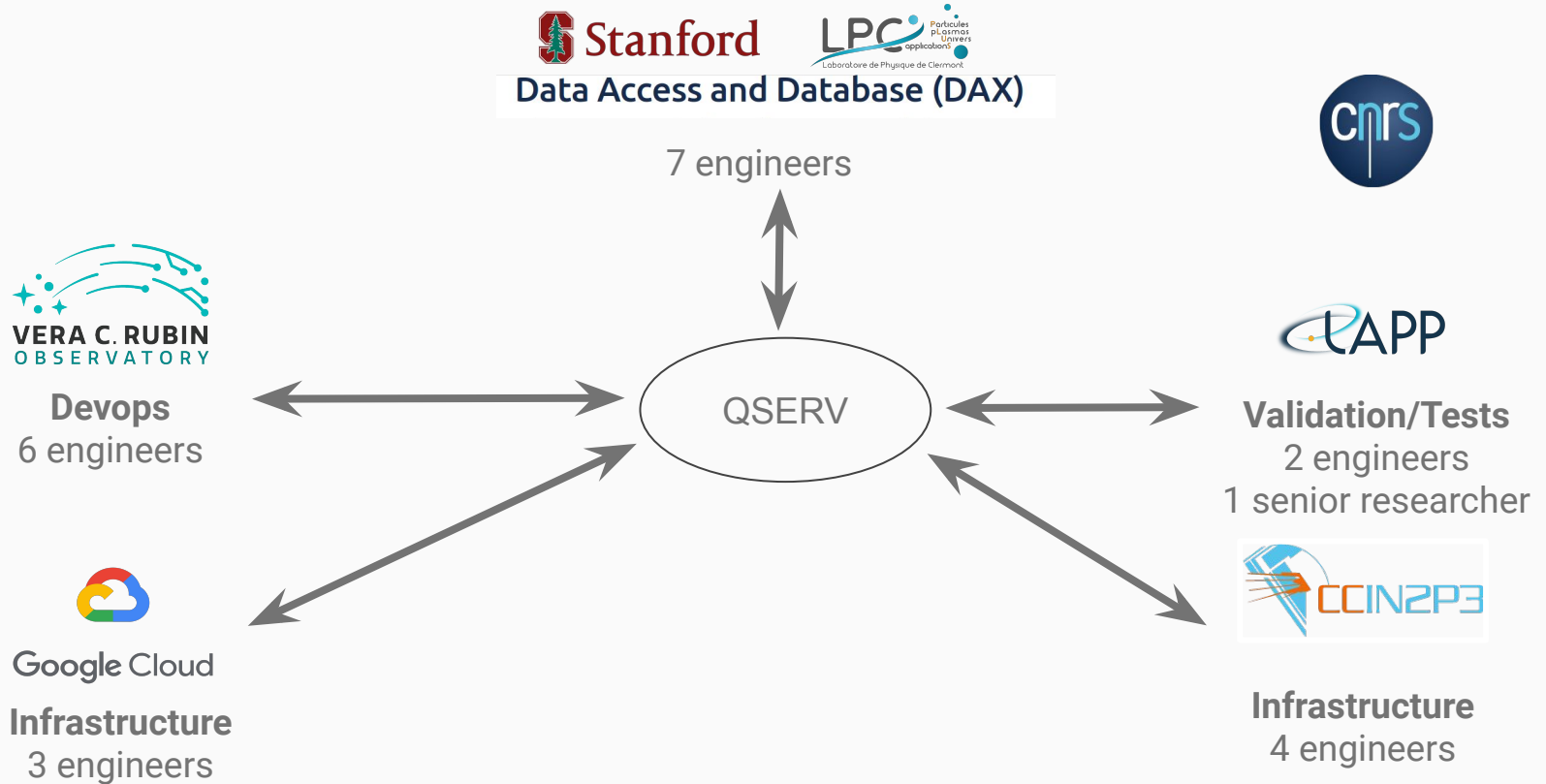
- ★ **Largest table: ~5 PB**
- ★ **Tallest table: ~50 trillion rows**
- ★ **Total (all data releases, compressed):
~83 PB**

Ad-hoc user-generated data
Rich provenance

Qserv

The Petascale database

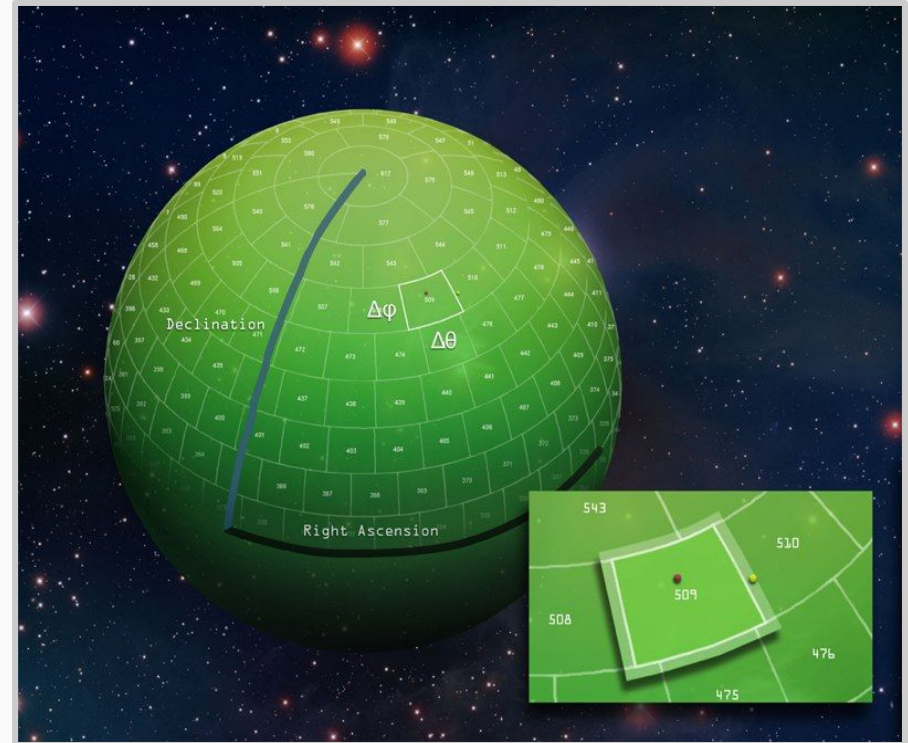
International context



The DAX team

Data Access and Database

- ★ Data and metadata
- ★ Images and databases
- ★ Persisting and querying
- ★ For pipelines and users
- ★ Real time Alert Production and annual Data Release Production
- ★ For Archive Center and all Data Access Centers
- ★ For USA, France and international partners
- ★ Persisted and virtual data
- ★ **Estimating, designing, prototyping, building, and productizing**

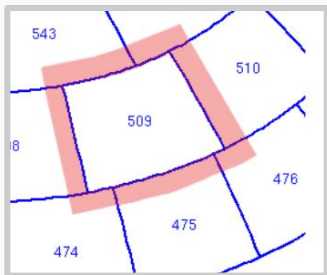
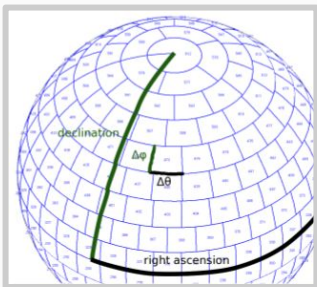


Qserv design



Relational database, 100% open source

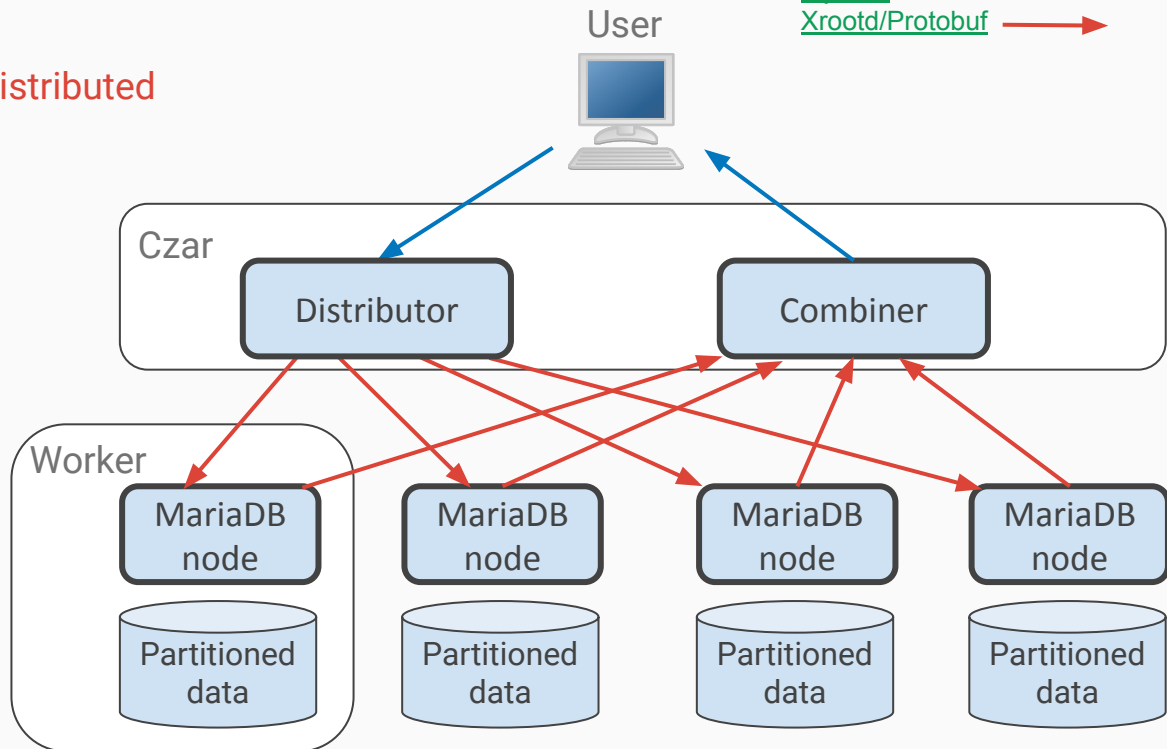
Spatially-sharded with overlaps

Map/reduce-like processing, highly distributed



Legend:

MySQL 
Xrootd/Protobuf 



~1000 workers, 20 chunks/5TB per workers

Highly automated deployment

Goals

In France

CC-IN2P3 will analyze **50% of the data** stream and provide **access to the entire catalog**

In the US

Google hosts the **Interim Data Facility**

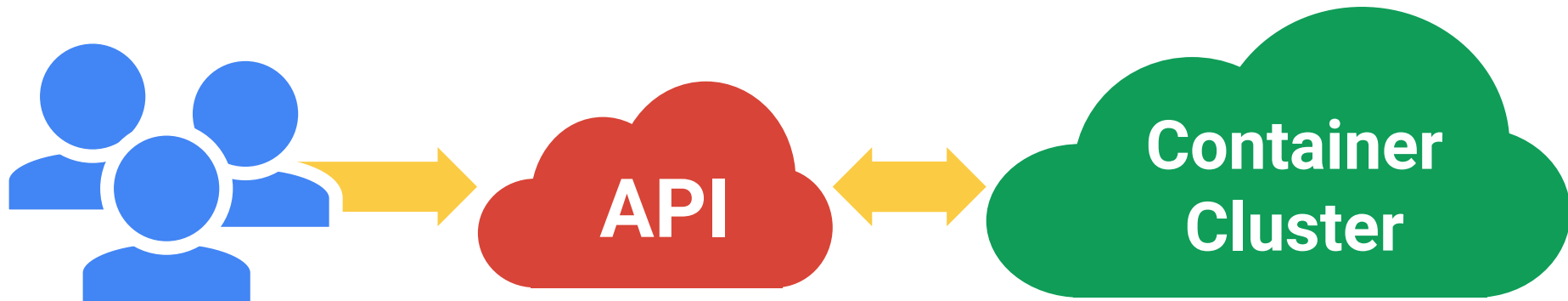
~1000 machines per database instance

Coordination of Rubin Observatory, IN2P3 and Google
Kubernetes accepted by the project and validated for **20%** of the target



Cloud-Native Kubernetes

All you really care about



Workload portability

Portability

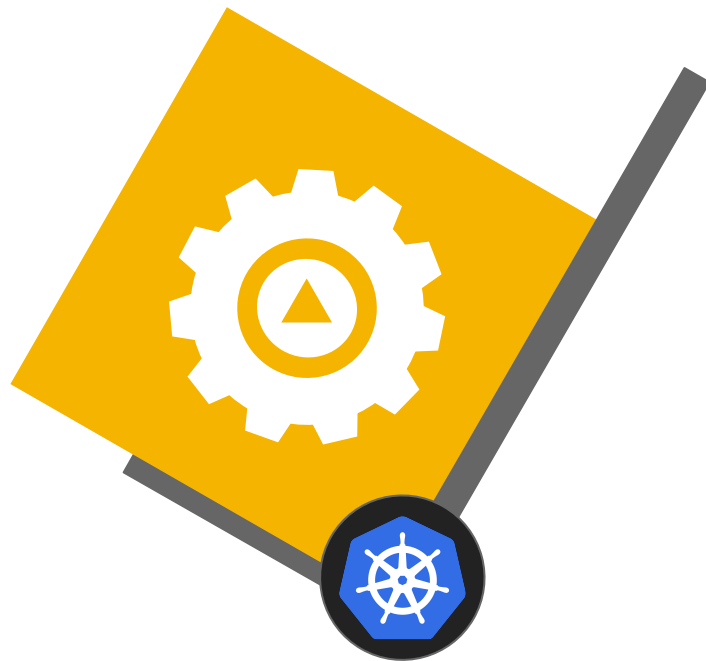
Build your apps on-prem,
lift-and-shift into cloud when you
are ready

Before Kubernetes

~3 months to deploy Qserv inside a new
cluster

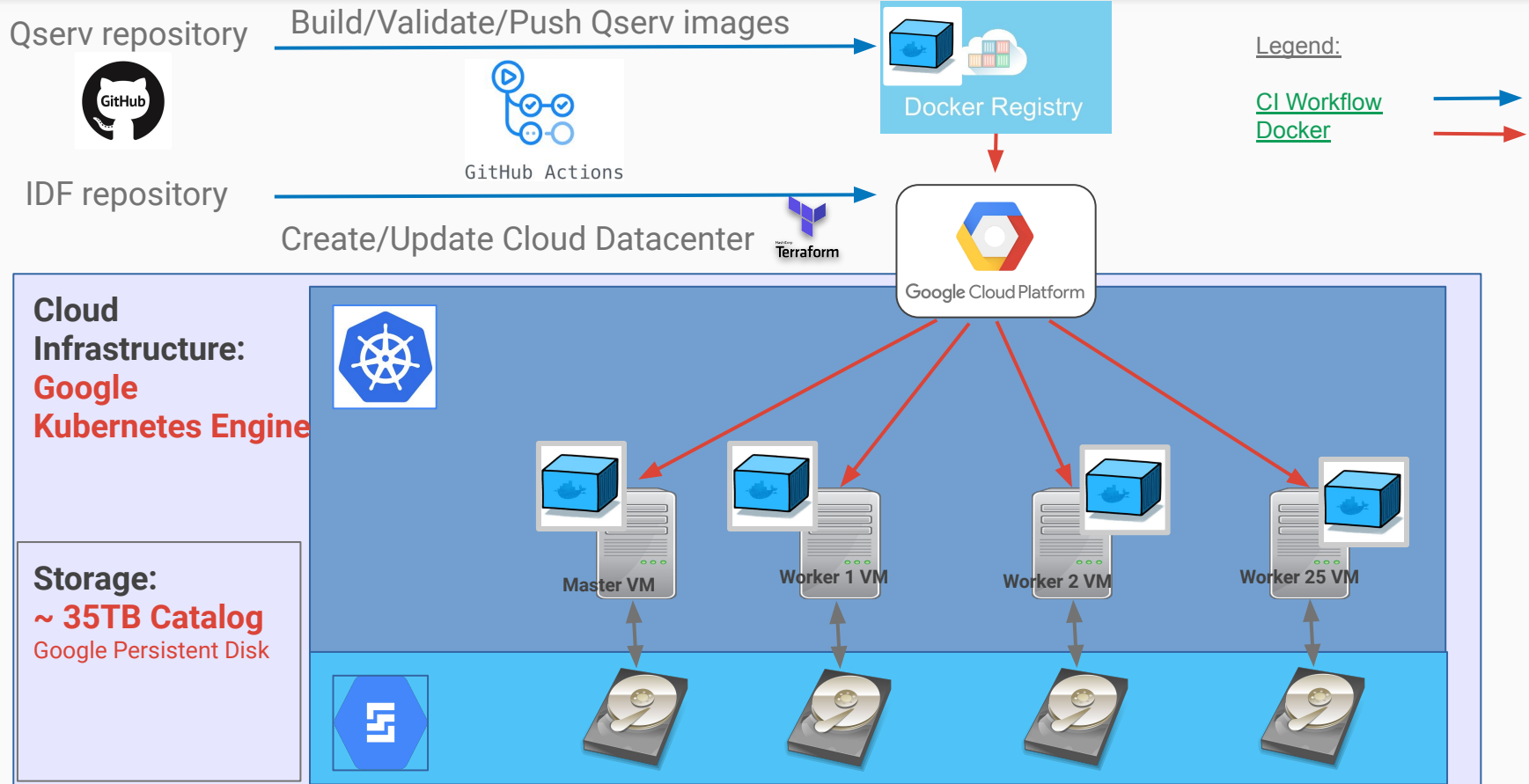
With Kubernetes

5 minutes to 1 day



Cloud-Native Gitops & CI

Automated deployment: Cloud Native

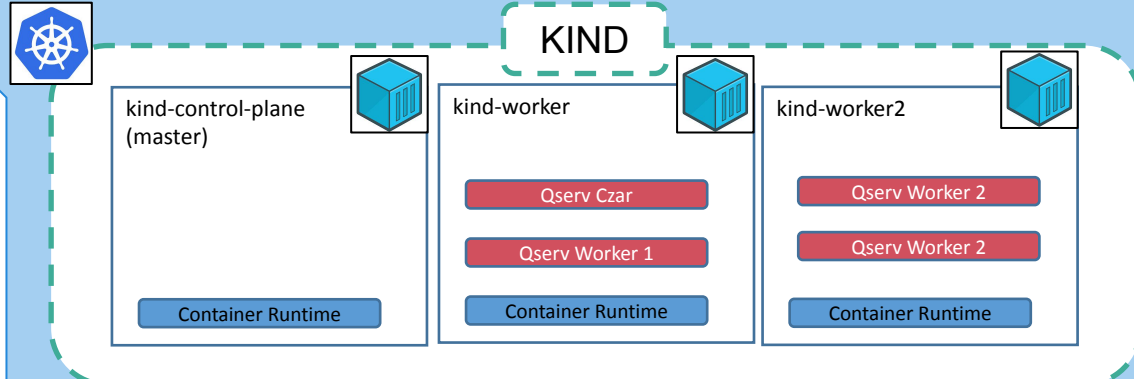


CI Setup with KIND



For each commit

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



Docker runtime

Github Action VM

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

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

CI in practice: Qserv integration tests

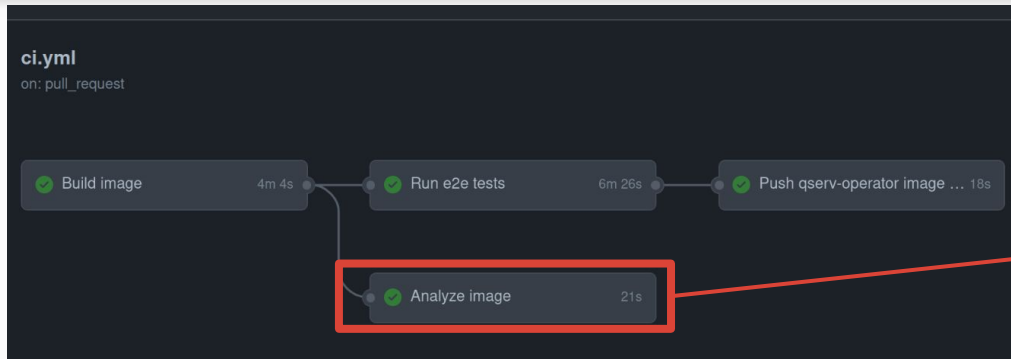
The screenshot shows the GitHub Actions interface. The top navigation bar includes links for Code, Issues, Pull requests (1), Actions (selected), Security, Insights, and Settings. On the left, a sidebar lists various workflow categories: CI, CodeQL, Documentation, Generate code reports, Static code analysis, 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) with a duration of 11m 22s. An arrow points from this run to the workflow diagram below.

The screenshot displays a workflow diagram for a file named `ci.yml` triggered on `pull_request`. The workflow consists of four steps:

- Build image (4m 4s)
- Run e2e tests (6m 26s) - This step is highlighted with a red box.
- Analyze image (21s)
- Push qserv-operator image ... 18s

The text "Integration tests with kind/k8s" is overlaid in red above the workflow diagram.

CI in practice: Qserv image scanning



Analyze image

succeeded 6 hours ago in 21s

- > ✓ Set up job
- > ✓ Download image
- > ✓ Load image in local registry
- > ✓ Scan operator image
- > ✓ upload Anchore scan SARIF report
- > ✓ Complete job

anchore

Vulnerability
Scanning &
Policy-Compliance
for Containers

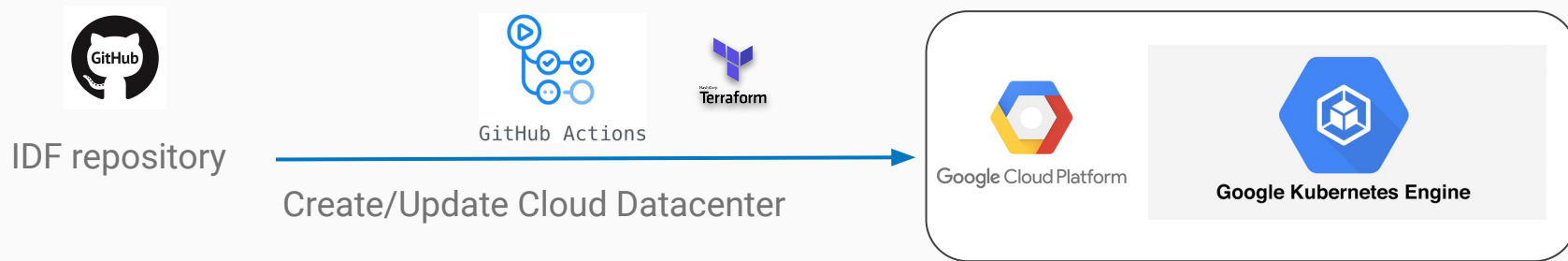
Requests 1 Actions Security Insights Settings

Code scanning

Add more scans

Latest scan	Pull request	Workflow	Lines scanned	Duration	Result
43 minutes ago	#28	CodeQL	2.48k / 2.6k ⓘ	5m 49s	0 alerts

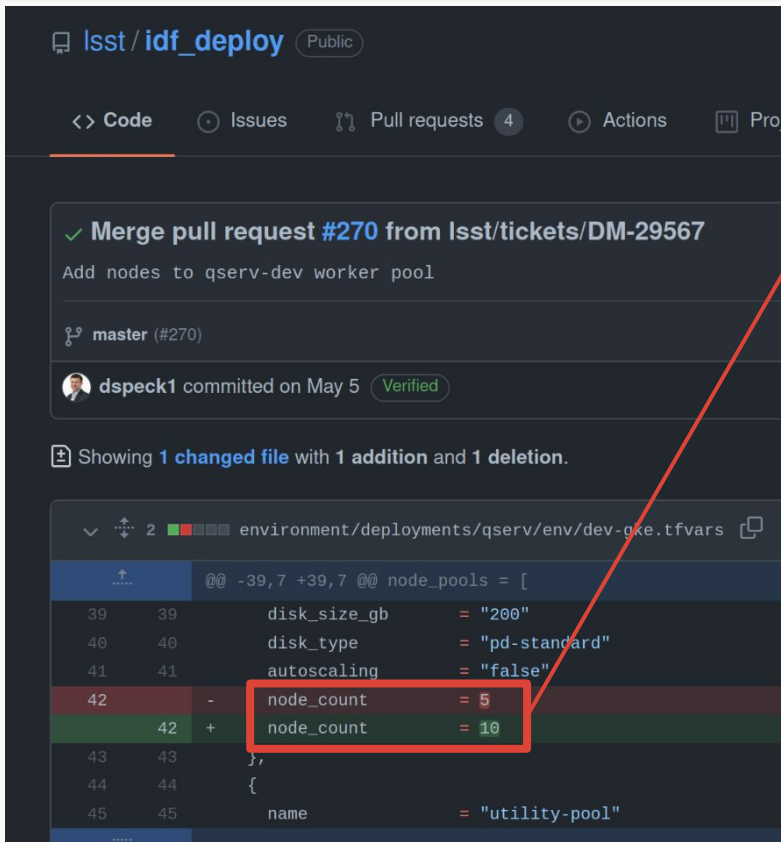
Gitops: CI + IaC



- Delegate access to infrastructure management
- Track who does what on infrastructure
- Recreate infrastructure from scratch
- Ease Kubernetes upgrade

Kubernetes is fully managed by Google Cloud / GKE

In practice



Isst / idf_deploy Public

<> Code Issues Pull requests 4 Actions Pro

✓ Merge pull request #270 from Isst/tickets/DM-29567

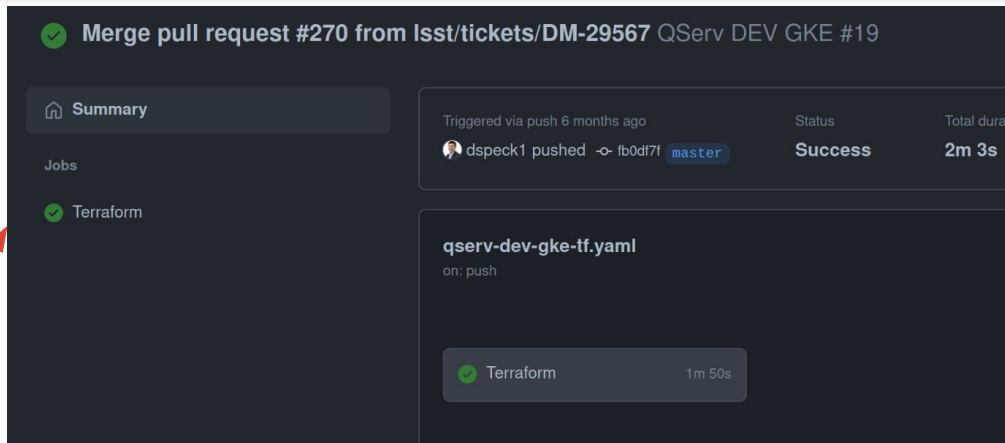
Add nodes to qserv-dev worker pool

master (#270)

dspeck1 committed on May 5 Verified

Showing 1 changed file with 1 addition and 1 deletion.

```
environment/deployments/qserv/env/dev-gke.tfvars
@@ -39,7 +39,7 @@ node_pools = [
39 39     disk_size_gb     = "200"
40 40     disk_type        = "pd-standard"
41 41     autoscaling      = "false"
42 42     node_count       = 5
43 43   },
44 44   {
45 45     name              = "utility-pool"
```



✓ Merge pull request #270 from Isst/tickets/DM-29567 Qserv DEV GKE #19

Summary

Triggered via push 6 months ago Status Success Total duration 2m 3s

Jobs

✓ Terraform

qserv-dev-gke-tf.yaml

on: push

✓ Terraform 1m 50s

Add five nodes to the GKE cluster
Kubernetes will then allow to easily scale Qserv

Gitops: the Google Cloud Console

Kubernetes Engine Clusters [EDIT](#) [DELETE](#) [ADD NODE POOL](#) [DEPLOY](#) [CONNECT](#) [DUPLICATE](#)

✓ **qserv-dev**

DETAILS **NODES** STORAGE LOGS

Node Pools

Filter Filter node pools

Name ↑	Status	Version	Number of nodes	Machine type	Image type	Autoscaling	
czar-pool-976f	✓ Ok	1.21.5-gke.1302	1	n2-standard-32	Container-Optimized OS with Containerd (cos_containerd)	Off	🗑️
utility-pool-7221	✓ Ok	1.21.5-gke.1302	1	n2-standard-4	Container-Optimized OS with Containerd (cos_containerd)	Off	🗑️
worker-pool-a64a	✓ Ok	1.21.5-gke.1302	10	n2-standard-16	Container-Optimized OS with Containerd (cos_containerd)	Off	🗑️

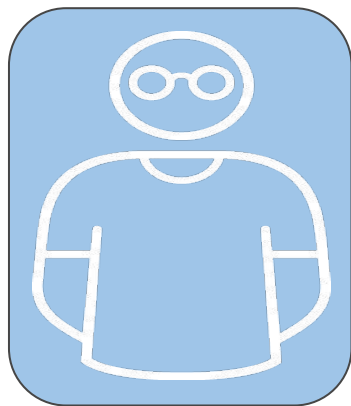
Nodes

Filter Filter nodes

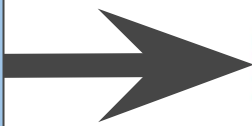
Name ↑	Status	CPU requested	CPU allocatable	Memory requested	Memory allocatable	Storage req
gke-qserv-dev-czar-pool-976f-80236054-71b4	✓ Ready	663 mCPU	31.85 CPU	503.32 MB	125.25 GB	
gke-qserv-dev-utility-pool-7221-c20d1f43-rdpx	✓ Ready	303 mCPU	3.92 CPU	367 MB	13.94 GB	
gke-qserv-dev-worker-pool-a64a-01bc5238-117x	✓ Ready	303 mCPU	15.89 CPU	367 MB	61.5 GB	
gke-qserv-dev-worker-pool-a64a-01bc5238-55xm	✓ Ready	303 mCPU	15.89 CPU	367 MB	61.5 GB	
gke-qserv-dev-worker-pool-a64a-01bc5238-d9ug	✓ Ready	303 mCPU	15.89 CPU	367 MB	61.5 GB	
gke-qserv-dev-worker-pool-a64a-01bc5238-11qm	✓ Ready	563 mCPU	15.89 CPU	482.34 MB	61.5 GB	
gke-qserv-dev-worker-pool-a64a-01bc5238-lv4s	✓ Ready	303 mCPU	15.89 CPU	367 MB	61.5 GB	
gke-qserv-dev-worker-pool-a64a-01bc5238-mwrj	✓ Ready	323 mCPU	15.89 CPU	377.49 MB	61.5 GB	
gke-qserv-dev-worker-pool-a64a-01bc5238-oq6x	✓ Ready	313 mCPU	15.89 CPU	387.97 MB	61.5 GB	
gke-qserv-dev-worker-pool-a64a-01bc5238-qfvs	✓ Ready	303 mCPU	15.89 CPU	367 MB	61.5 GB	

Cloud-Native Kubernetes operators

Operators embed ops knowledge from the experts

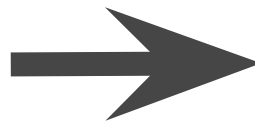
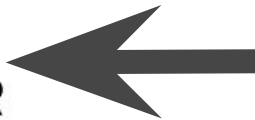


ops knowledge from the experts



**OPERATOR
SDK**

operator
implementation
i.e. k8s controller

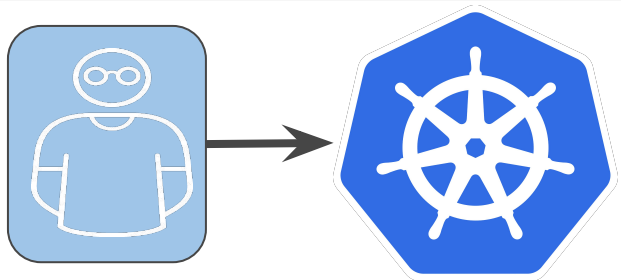


K8s standard API:
Deployments
StatefulSets
Autoscalers
Secrets
Config maps

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>

How does an operator works?



Software Developer
Kubernetes user

K8s API

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



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-rc1-37-g45646a1cb
    replicas: 1
    storage: 1Ti
  worker:
    image:
      qserv/lite-qserv:2021.10.1-rc1-37-g45646a1cb
    replicas: 10
  ...
```

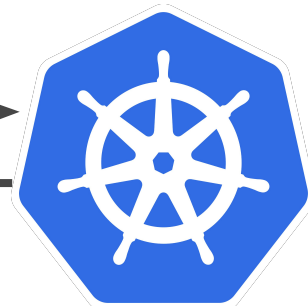
Custom Kubernetes controller

Watch Event

Reconcile

Custom resource definition

here Qserv



Deployments
StatefulSets
Autoscalers
Secrets
Config maps

Why should you use an operator?

Operators: both sysadmin + application experts

🔗 **Resize/Upgrade**

🔗 **Reconfigure**

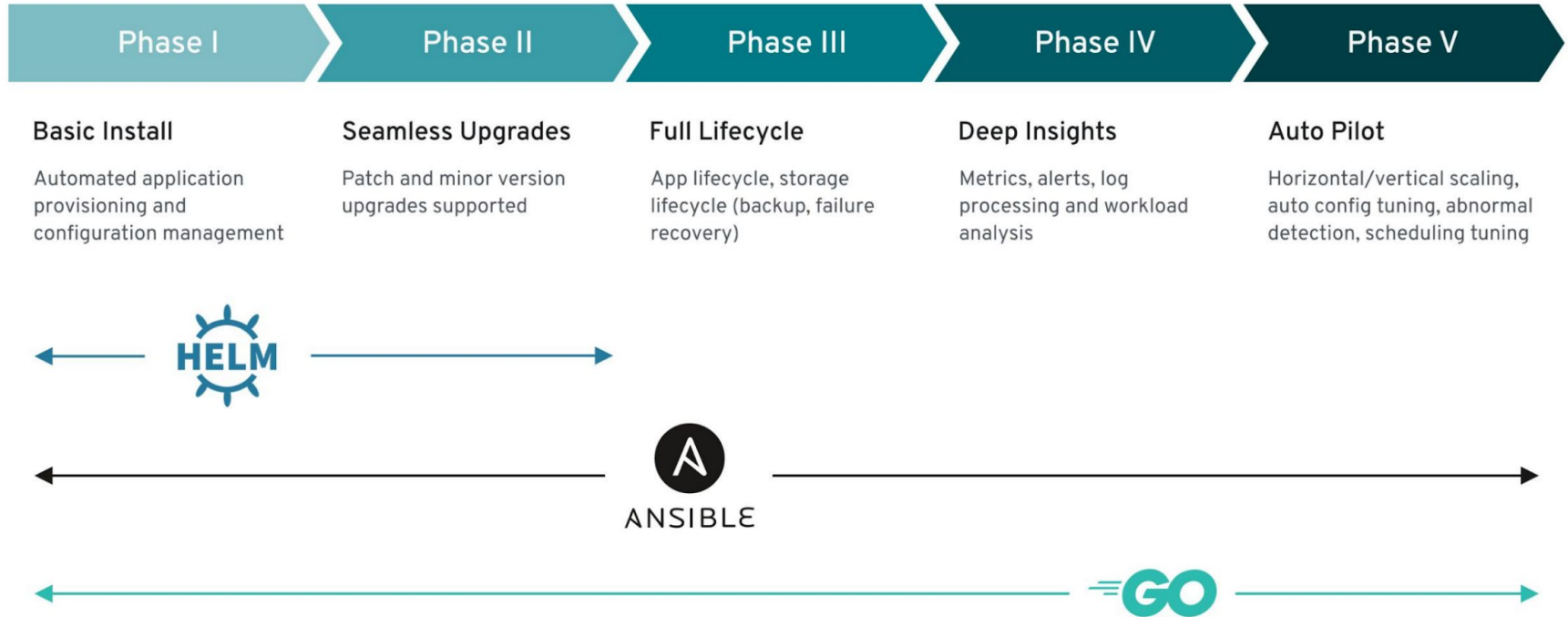
🔗 **Backup**

🔗 **Healing**



The Sysadmin

Types of operators



Operator across the industry

OperatorHub.io | The registry for Kubernetes Operators



Qserv is available on operatorHub

<https://operatorhub.io/operator/qserv-operator>



OperatorHub.io

Qserv operator

Create and maintain highly-available Qserv clusters on Kubernetes

Search OperatorHub... Contribute

Home > Qserv operator

Qserv operator

Install

Overview

Qserv Operator manages the full lifecycle of Qserv at scale, in order to ease and fully automate deployment and management of Qserv clusters.

The operator aims to provide the following:

- Basic Install to Qserv components.
- Out-of-box Intra-Cluster discovery support.

Mind you, this is still a work-in-progress implementation.

CHANNEL
alpha

VERSION
2021.9.1-rc1 (Current) ▾

CAPABILITY LEVEL ⓘ

- Basic Install
- Seamless Upgrades
- Full Lifecycle
- Deep Insights
- Auto Pilot

PROVIDER
Vera C. Rubin Observatory

Seamless upgrade is a work in progress

Cloud-Native Storage management

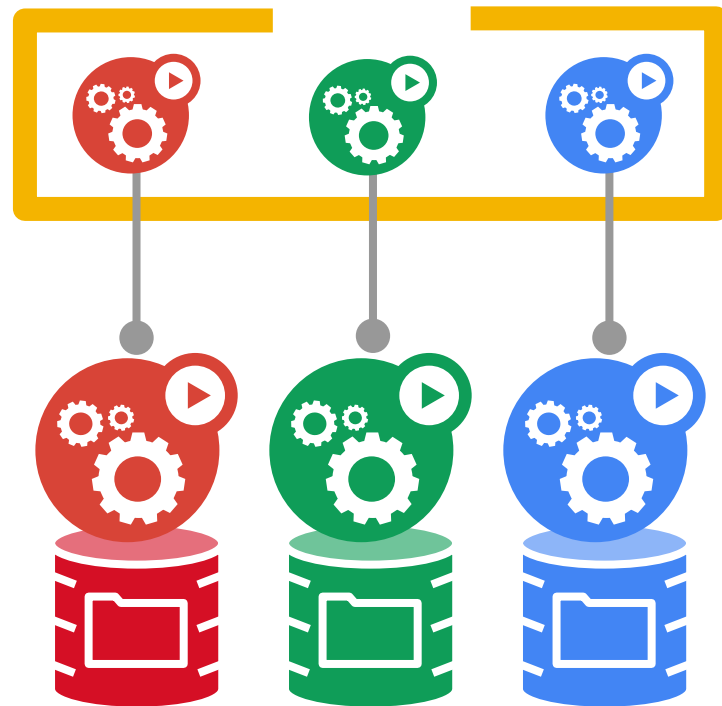
Qserv is deployed using StatefulSets

Goal: enable clustered software on Kubernetes

- Qserv, but also mysql, redis, zookeeper, ...

Clustered apps need “identity” and have sequencing constraints

- stable hostname, available in DNS
- an ordinal index
- stable storage: linked to the ordinal & hostname
- discovery of peers for quorum
- startup/teardown ordering



Storage management

GKE: Dynamic storage provisioning

User deploy Qserv instance

Create PVClaims

Google Storage creates automatically PersistentVolume+Google Disks (ssd+hdd)

On-premise:

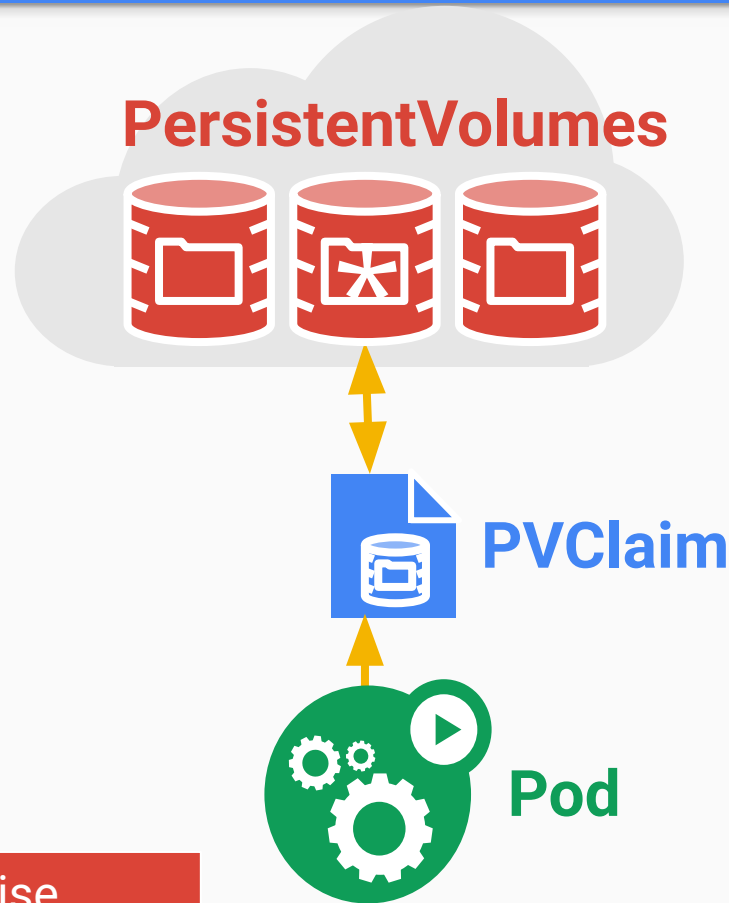
Storage is manually declared to Kubernetes (via PV) and created



Cluster Admin



User



Easier on GKE, but better performance on-premise

What we need to improve

Backup and restore

Use Kubernetes standard API (i.e. VolumeSnapshots)?

Use more advanced but non standard tools like Portworx, StorageOS, Longhorn?

Dynamic provisioning on-premise

Looking at CNCF Landscape, but no standard solution yet



Objective:

Use the very same procedures to manage the storage On-Premise and on Commercial Cloud

Cloud-Native Workflows

A powerful data ingest workflow

Qserv has a powerful distributed ingest algorithm

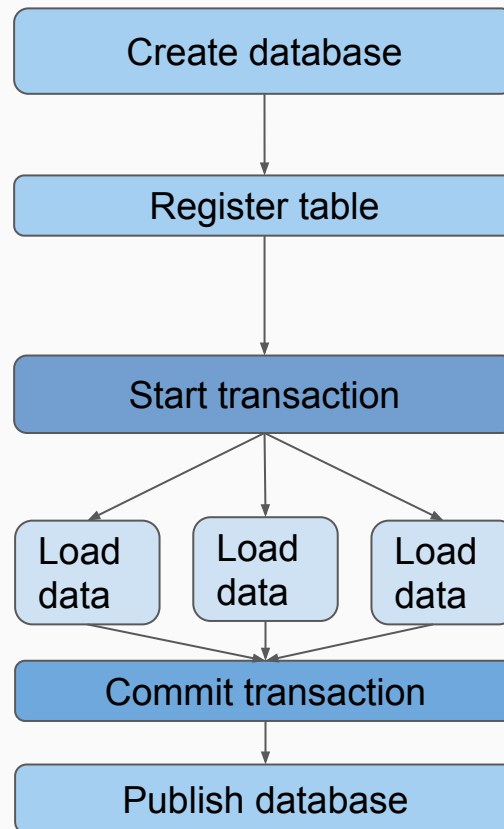
Flexible but require orchestrating tasks (DAG)

Argo Workflow project help us a log



Case study 2021: Implementation of a large-scale data loading algorithm

Ingestion of 15000 files and 15TB in 1h30

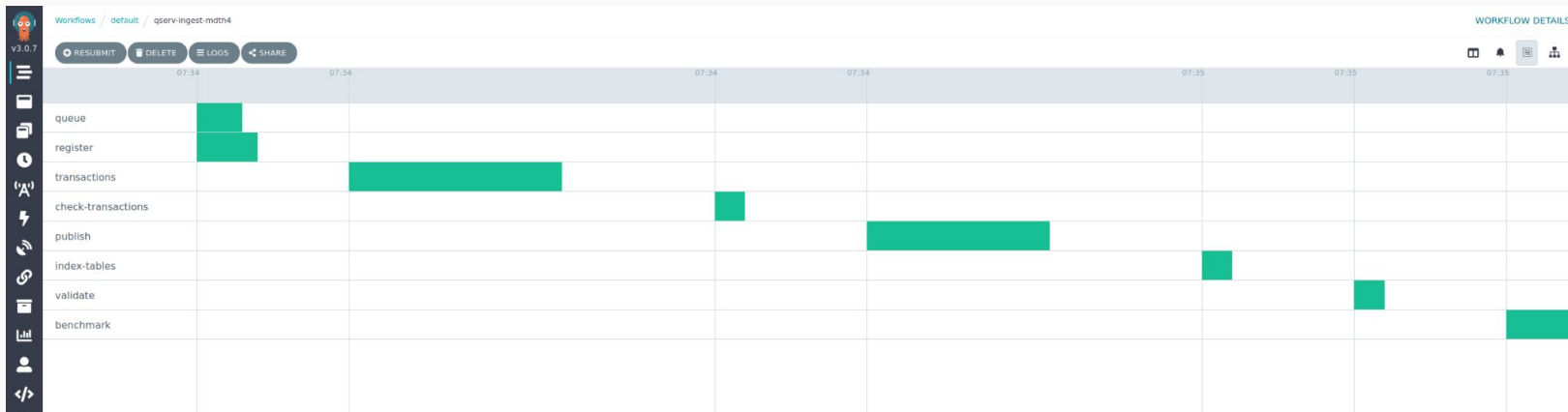


Argo: screenshots

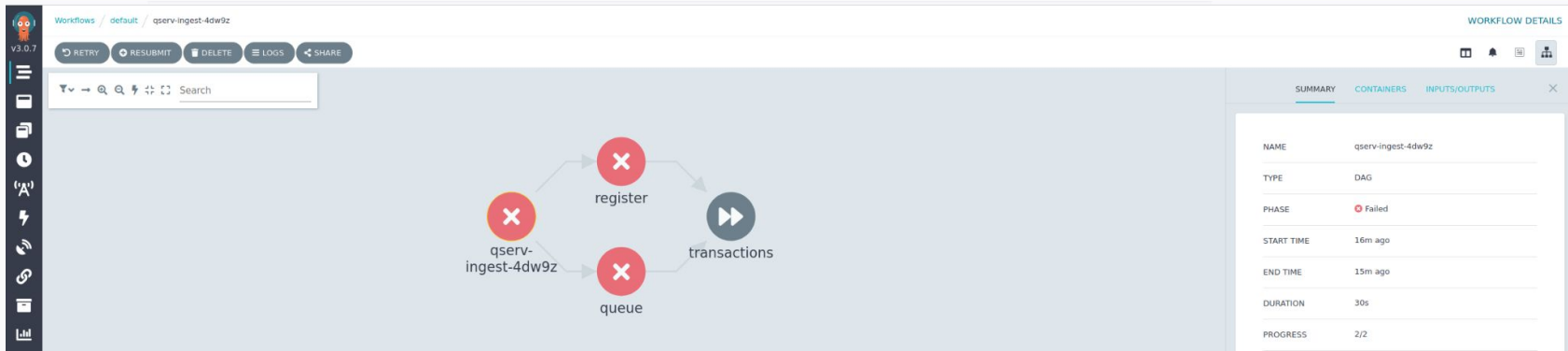
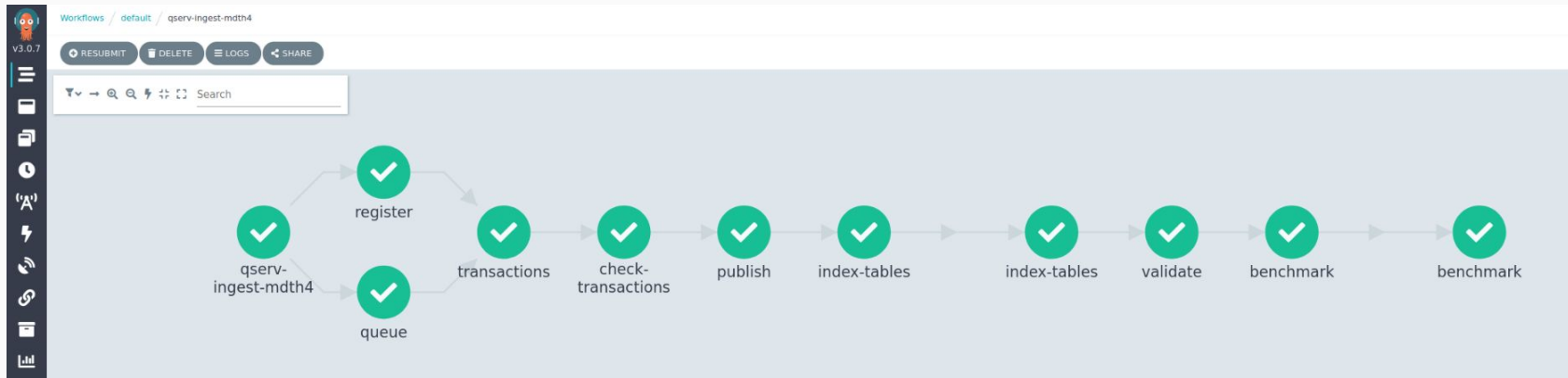


```
fjammes@clrinport18 ~$ argo get @latest | tail -n 15
```

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



Argo: screenshots



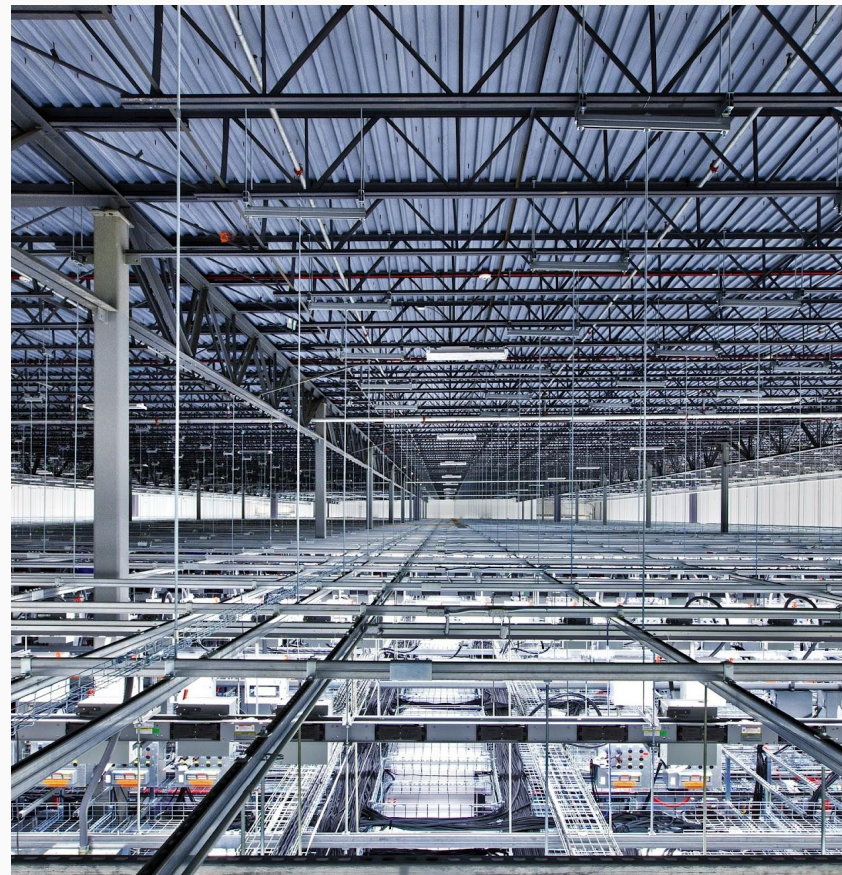
Public cloud: pros and cons

Pros

- ★ Flexibility (access, provisioning)
- ★ Excellent support
- ★ Low maintenance
- ★ Cool proprietary features

Cons

- ★ Expensive / Cost difficult to manage
- ★ Vendor lock-in
- ★ Hide Kubernetes internals (black box)
- ★ Run slower than bare-metal (~25%)



- 1 Qserv is going on
- 2 Container orchestration helps a lot
- 3 Commercial cloud might be an alternative

Conclusion

Q&A

Fabrice JAMMES
Laboratoire de Physique de
Clermont

Fritz Mueller
SLAC National Accelerator
Laboratory

