Fermilab

Sep 14, 2026

Operational Evolution of the Fermilab FTS3 Service for non-CMS experiments: Modernizing Deployment, Automation, and Elastic Operations

Lorena Lobato Pardavila
(in collaboration with Victor Leopoldo Muñoz Flores)

U.S. DEPARTMENT of ENERGY
Fermi National Accelerator Laboratory is managed by FermiForward for the U.S. Department of Energy Office of Science

FERMI NATIONAL ACCELERATOR LABORATORY

What I'll cover today

Agenda

Where we started

Why evolve it?

FTS3 worked - why change it?

The evolution

How we have modernized our configuration

Did it work?

Operational Challenges & Lessons Learned

What's next?

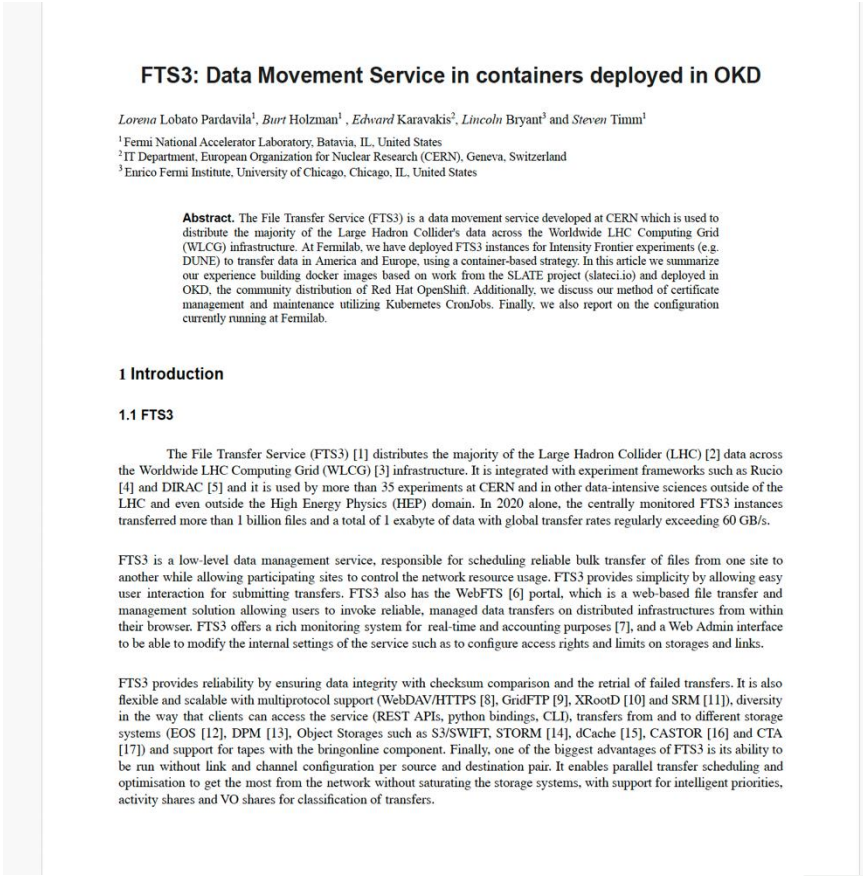
Current status & Future plans

Conclusions

Takeaways and Q&As

Where did the journey start?

Before the evolution



FTS3: Data Movement Service in containers deployed in OKD

Lorena Lobato Pardavila[1], *Burt* Holzman[1] , *Edward* Karavakis[2], *Lincoln* Bryant[3] and *Steven* Timm[1]

[1] Fermi National Accelerator Laboratory, Batavia, IL, United States
[2] IT Department, European Organization for Nuclear Research (CERN), Geneva, Switzerland
[3] Enrico Fermi Institute, University of Chicago, Chicago, IL, United States

Abstract. The File Transfer Service (FTS3) is a data movement service developed at CERN which is used to distribute the majority of the Large Hadron Collider's data across the Worldwide LHC Computing Grid (WLCG) infrastructure. At Fermilab, we have deployed FTS3 instances for Intensity Frontier experiments (e.g. DUNE) to transfer data in America and Europe, using a container-based strategy. In this article we summarize our experience building docker images based on work from the SLATE project (slateci.io) and deployed in OKD, the community distribution of Red Hat OpenShift. Additionally, we discuss our method of certificate management and maintenance utilizing Kubernetes CronJobs. Finally, we also report on the configuration currently running at Fermilab.

1 Introduction

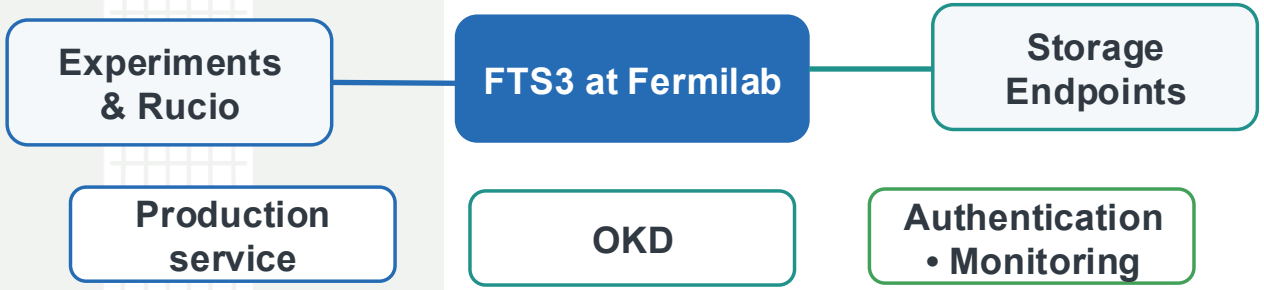
1.1 FTS3

The File Transfer Service (FTS3) [1] distributes the majority of the Large Hadron Collider (LHC) [2] data across the Worldwide LHC Computing Grid (WLCG) [3] infrastructure. It is integrated with experiment frameworks such as Rucio [4] and DIRAC [5] and it is used by more than 35 experiments at CERN and in other data-intensive sciences outside of the LHC and even outside the High Energy Physics (HEP) domain. In 2020 alone, the centrally monitored FTS3 instances transferred more than 1 billion files and a total of 1 exabyte of data with global transfer rates regularly exceeding 60 GB/s.

FTS3 is a low-level data management service, responsible for scheduling reliable bulk transfer of files from one site to another while allowing participating sites to control the network resource usage. FTS3 provides simplicity by allowing easy user interaction for submitting transfers. FTS3 also has the WebFTS [6] portal, which is a web-based file transfer and management solution allowing users to invoke reliable, managed data transfers on distributed infrastructures from within their browser. FTS3 offers a rich monitoring system for real-time and accounting purposes [7], and a Web Admin interface to be able to modify the internal settings of the service such as to configure access rights and limits on storages and links.

FTS3 provides reliability by ensuring data integrity with checksum comparison and the retrial of failed transfers. It is also flexible and scalable with multiprotocol support (WebDAV/HTTPS [8], GridFTP [9], XRootD [10] and SRM [11]), diversity in the way that clients can access the service (REST APIs, python bindings, CLI), transfers from and to different storage systems (EOS [12], DPM [13], Object Storages such as S3/SWIFT, STORM [14], dCache [15], CASTOR [16] and CTA [17]) and support for tapes with the bringonline component. Finally, one of the biggest advantages of FTS3 is its ability to be run without link and channel configuration per source and destination pair. It enables parallel transfer scheduling and optimisation to get the most from the network without saturating the storage systems, with support for intelligent priorities, activity shares and VO shares for classification of transfers.

The question was not “Can FTS3 run?” — it was “Can we operate it sustainably?”



FERMI NATIONAL ACCELERATOR LABORATORY

What did the architecture look like?

One Pod, many responsibilities

Single FTS3 pod

- DeploymentConfig, BuildConfig & ImageStreams
- Manual deployment and configuration
- Fixed resources and startup dependencies

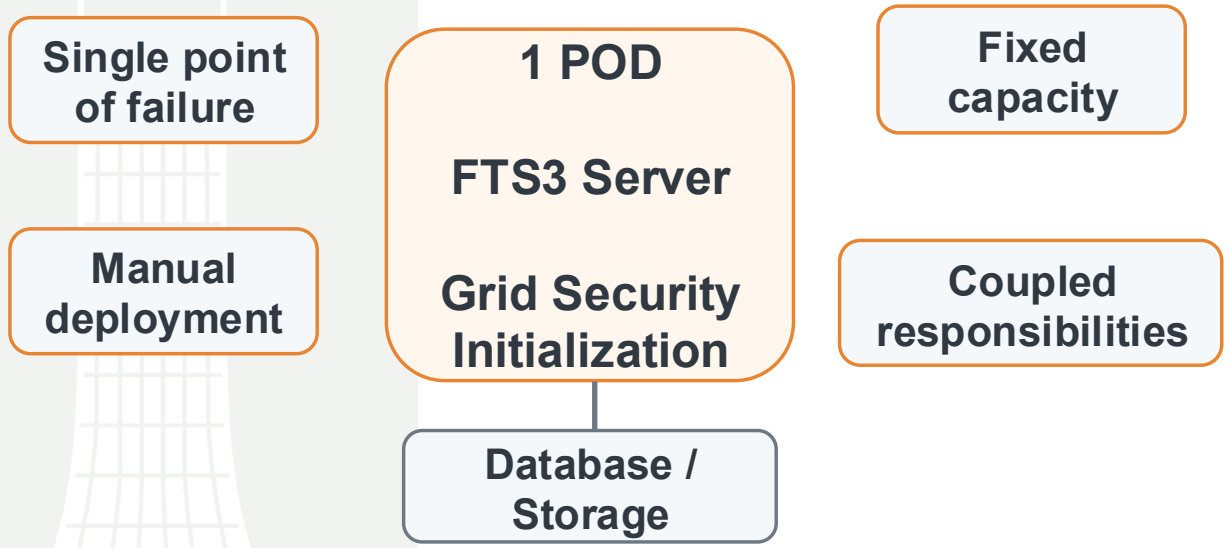
Platform & storage

- OKD / OpenShift-based deployment
- Persistent volumes backed by GlusterFS

Certificate management

- FTS3 required shared CAs, CRLs and certificates
- OSG repository + osg-ca-manager
- Startup-coupled CRL updates via CronJobs

The initial architecture worked, but operational limitations were accumulating.



FERMI NATIONAL ACCELERATOR LABORATORY

The service worked. The operational model didn't scale with it

Why evolve?

One Pod is one point of failure

- A Pod failure meant temporarily losing the service endpoint.
- Maintenance could require service intervention.
- We needed multiple replicas without creating new operational problems.

ONE POD: A failure could remove the service endpoint

Couple Startup

- The old startup process mixed service startup with Grid Security initialization.
- Multiple replicas could potentially perform initialization independently.
- We separated service execution from security configuration.

COUPLED STARTUP
Multiple replicas could repeat initialization

Why pay for capacity we don't need?

- Workload varies over time.
- Static resource allocation either wastes resources or risks resource pressure.
- HPA dynamically adjusts FTS3 replicas based on CPU/memory utilization.

FIXED CAPACITY
Either waste resources or hit resource pressure

(A lot of) Manual Operation

- Deployment, migration and configuration depended on manual intervention
- Updates and maintenance were operationally expensive
- Automation became necessary as the service evolved

MANUAL OPS
Deployment and migration depended on human intervention

From legacy deployment to elastic operations

MORE RESILIENT • MORE ELASTIC • MORE REPRODUCIBLE

1. Modern containers – the roadmap

- SL7 → AlmaLinux 9
- Upgrade FTS3 version (3.14)
- Supported base images

2. Kubernetes-native architecture

- Deployment instead of DeploymentConfig
- Multiple replicas
- Explicit Pod lifecycle management

3. Elastic operations - autoscaling

- Horizontal Pod Autoscaling (HPA)
- 2–5 Pods
- CPU / memory-based scaling

4. DevOps / GitOps practices - automation

- GitLab CI
- Kustomize
- Argo CD
- Git as the source of truth

From one pod to an elastic service

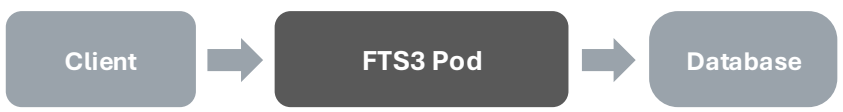
The biggest architectural change: from a single instance to a multi-replica service.

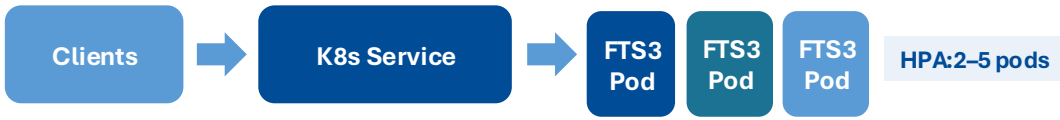
BEFORE — Single Pod

- One FTS3 Pod served all production traffic
- A Pod failure meant losing the service endpoint
- Fixed capacity, sized for peak load

AFTER — Elastic Service

- Multiple replicas run behind a Kubernetes Service
- Startup, readiness & pre-stop hooks keep traffic safe
- HPA scales capacity automatically with demand



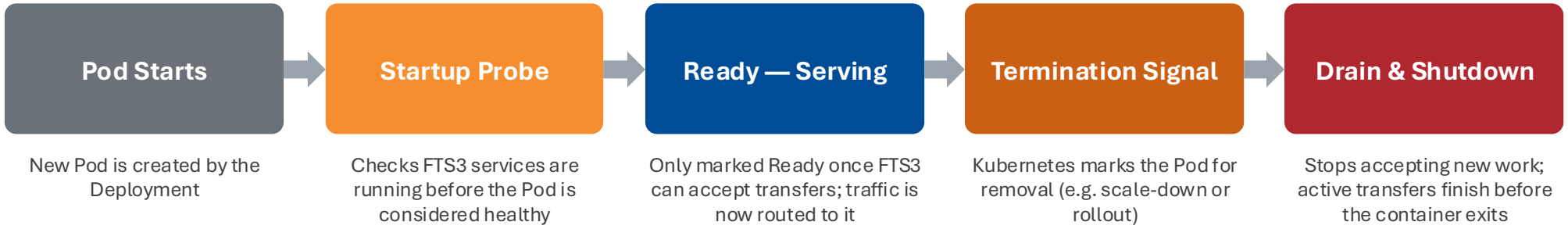


HPA scales the deployment between 2 and 5 replicas, based on CPU and memory utilization

Scaling is easy. Scaling safely is not.

Autoscaling only works if the Pod lifecycle is controlled.

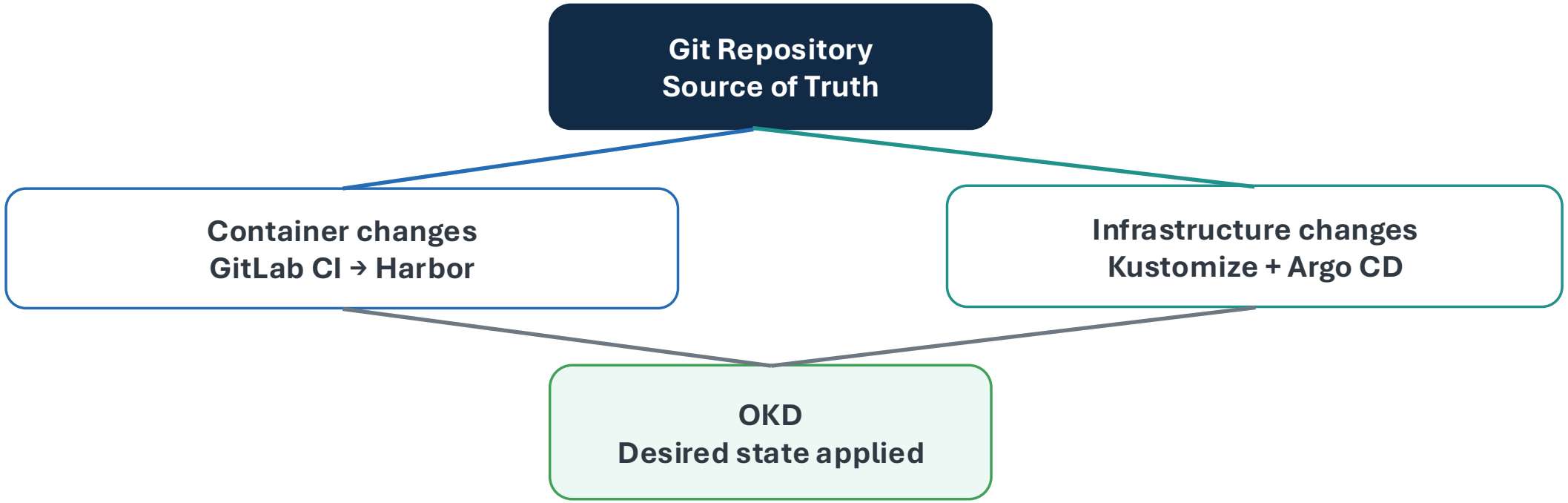
- **Startup probe**: Is FTS3 actually running? Is the pod actually healthy?
  - Make sure the services are actually running.
- **Readiness probe:** Is FTS3 ready for traffic? Should it receive new transfers?
  - Don't send requests until FTS3 is ready.
- **Pre-stop hook/drain mode:** Can we shut down gracefully?
  - When a Pod is terminated, allow active transfers to finish before shutting down



Startup + readiness + drain mode = safe autoscaling

From manual operations to GitOps

The modernization was an operational redesign — not just a software upgrade.



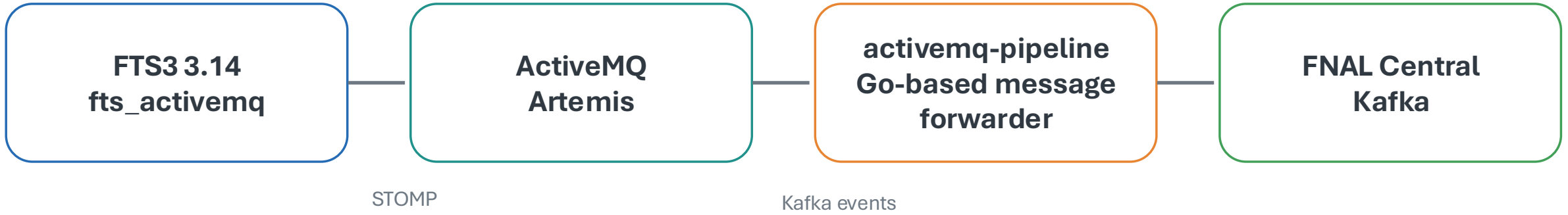
FERMI NATIONAL ACCELERATOR LABORATORY

Keeping monitoring compatible with FTS3 evolution

Modernizing the monitoring pipeline

Why did we need it?

- FTS3 3.14 changed the messaging path — the surrounding monitoring infrastructure had to evolve too.
- Monitoring infrastructure required **direct Kafka integration**
- We introduced a dedicated **ActiveMQ broker + lightweight Go pipeline**



Small supporting pipeline → compatible monitoring without changing FTS3 itself

What did we actually gain?

From Operational Pain to Operational Control

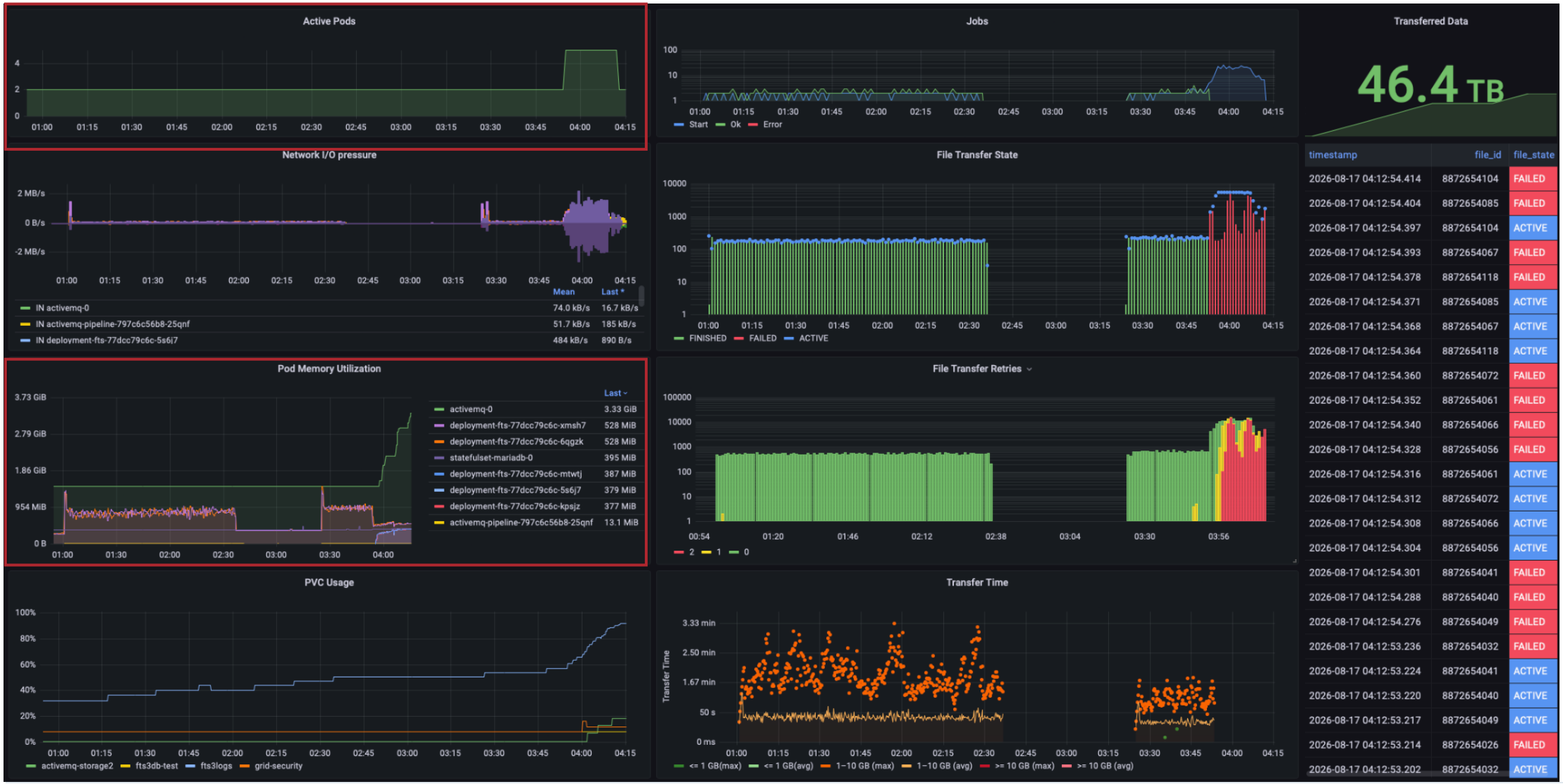
BEFORE	AFTER
Single point of failure	Multiple replicas
Fixed resources	Elastic scaling
Manual startup/configuration	Automated lifecycle
Mixed responsibilities	Separated initialization
Manual deployment	GitOps / CI/CD
Hard to reproduce	Version-controlled configuration

Not just a newer FTS3 deployment. A new way of operating it.

Did it actually scale? Yes.

The HPA reacted to increased resource pressure without manual intervention

① **Low workload: 2 Pods**
Stable CPU/memory utilization
② **Load increases: Memory rises**
More jobs → higher resource utilization
③ **HPA reacts: +3 Pods (no manual intervention)**
Deployment scales from **2** → **5**



What we learned

Three lessons matter beyond this specific deployment.

Autoscaling is not enough

Safe lifecycle management is part of the design.

lifecycle management

readiness

graceful shutdown

shared configuration

database considerations

Automation is more than convenience

Reproducibility and traceability reduce operational dependency.

GitOps makes changes traceable and repeatable.

FTS3 is one part of the ecosystem

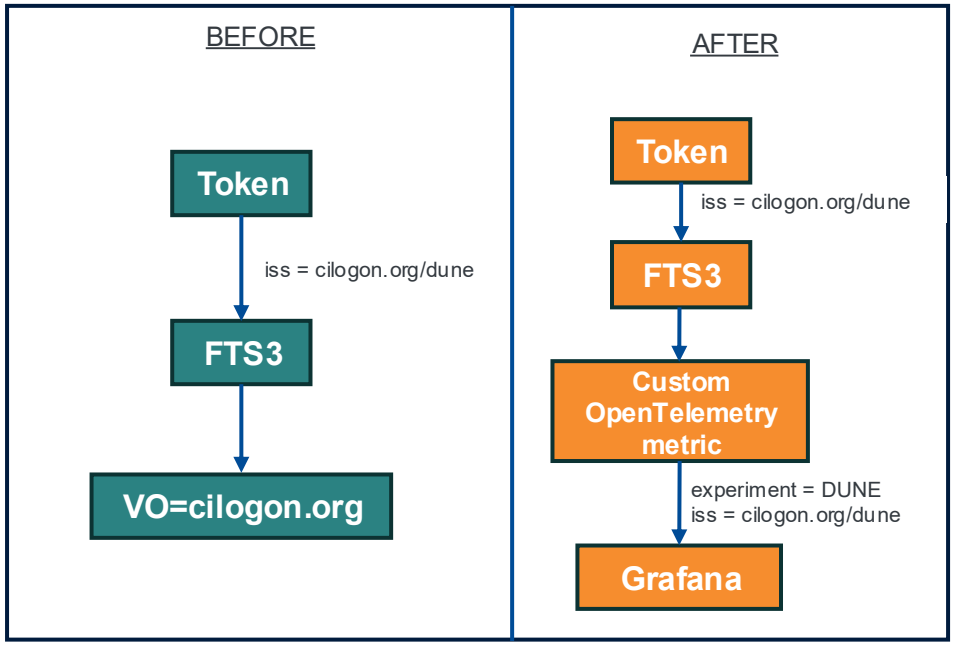
Scaling FTS3 does not automatically scale storage endpoints.

Modernizing infrastructure reduces some operational effort, but Kubernetes introduces many components and configurations that still need to be managed.

The whole transfer path matters, not just the FTS3 Pods

From monitoring to operational intelligence

- **Improve observability**
  - Identify token-authenticated transfers by **experiment**
  - Expose **source and destination transfer errors**
  - Add custom **OpenTelemetry metrics**
  - Visualize through **Grafana**

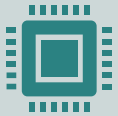


- **AI-assisted operations**
  - Collect logs, metrics, incidents, and maintainer knowledge
  - Build a **RAG-based operational knowledge base**
  - Help operators investigate anomalies and previous incidents

The Goal Wasn't Just Kubernetes

FTS3 Modernization: We moved from a legacy, manually intensive deployment toward a Kubernetes-native architecture.

Elasticity and resilience: Multiple replicas, HPA, and controlled Pod lifecycle management improved service continuity and enabled capacity to adapt to workload.



Reproducible operations: GitOps and CI/CD made infrastructure and application changes more automated, traceable, and maintainable.

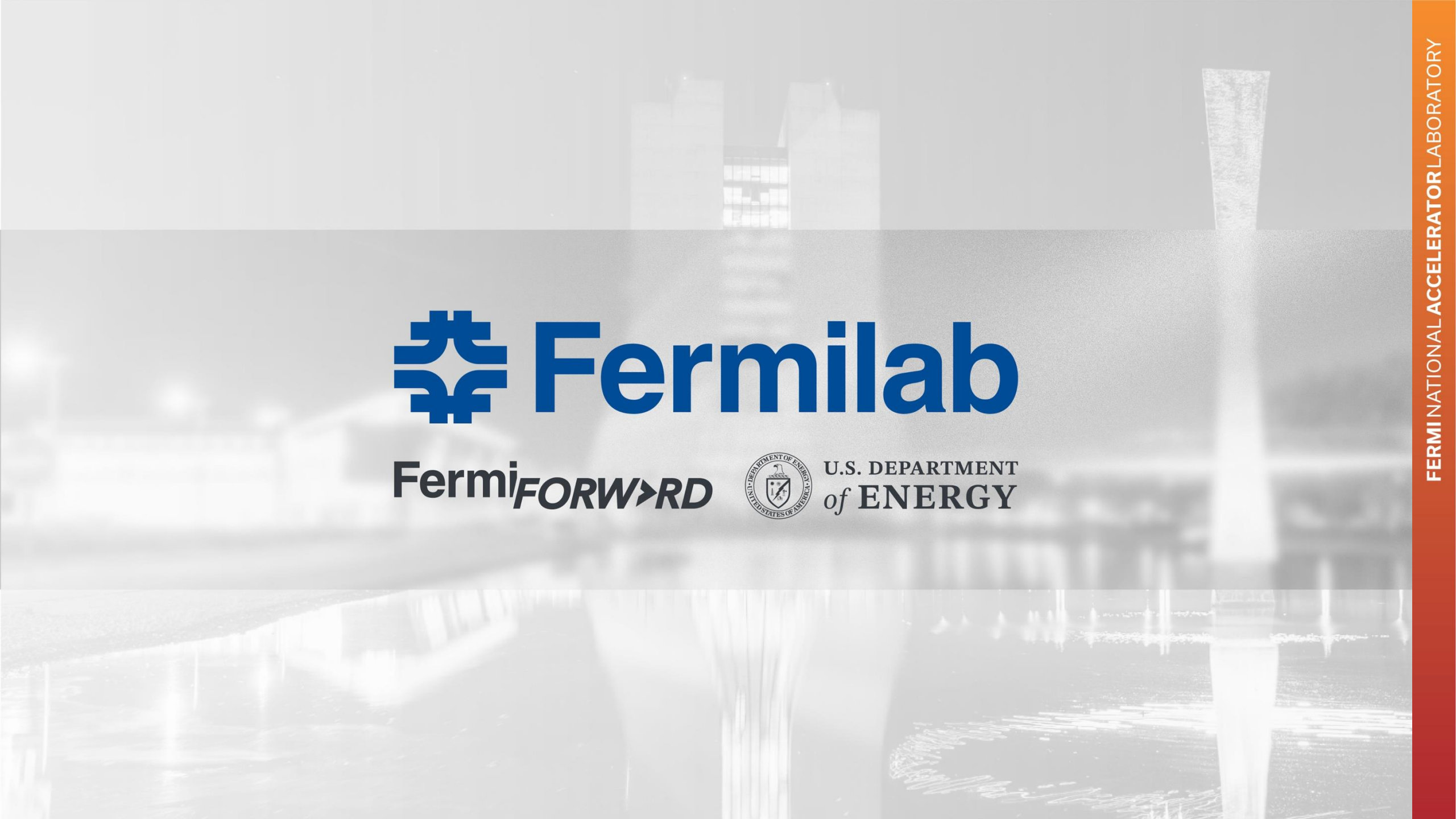
The goal is not simply to run FTS3 on Kubernetes, but to make the service easier to operate, reproduce, scale, and evolve.

Operational Evolution of the Fermilab FTS3 Service
for non-CMS experiments

Thank you!

Q&A

FERMI NATIONAL ACCELERATOR LABORATORY


Fermilab
FermiFORWARD
U.S. DEPARTMENT of ENERGY
FERMI NATIONAL ACCELERATOR LABORATORY