

# Managing the OSG Fabric of Services the GitOps Way

Brian Bockelman, on behalf of the co-authors

Brian Bockelman (Morgridge)

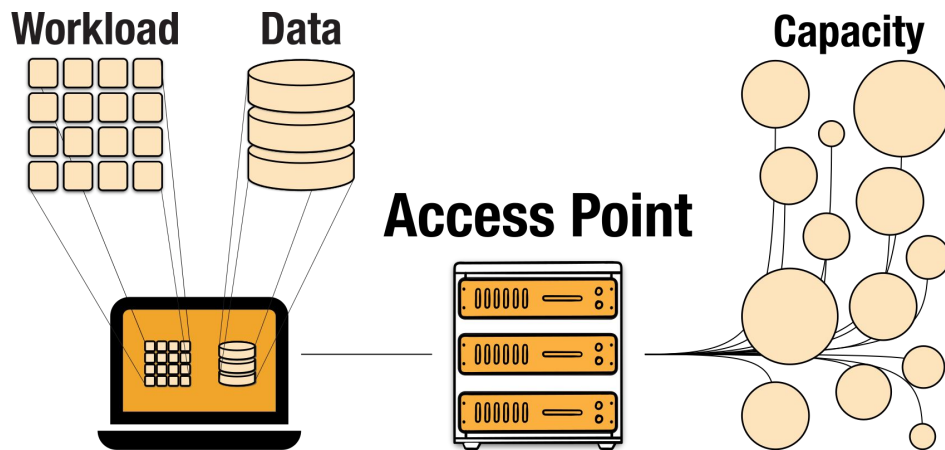
Brian Lin (UW-Madison)

John Thiltges (UNL)

Fengping Hu (U. Chicago)

# The OSG Consortium

- OSG Consortium operates a Fabric of Services to enable distributed High Throughput Computing (dHTC).
- These services are run by a small set of staff funded through projects such as IRIS-HEP and PATH.
  - Beyond the operational services, the projects teams also produce technologies (HTCondor Software Suite) and provide consulting/facilitation services.



The OSG Consortium needs to innovate quickly and be flexible – all on a “research budget”!



# OSG Fabric of Services - at a glance

The OSG Consortium operates more than

**150 services**

(internal and external) across

**4 large sites**

for the main central services.

Of these, there are

**55 Hosted CEs**

and caches that are distributed across

**2 dozen small sites**

# OSG Fabric of Services - Challenges

- Services and operators are spread across timezones.
- Not every location has a local operator.
- Varies widely in complexity (multi-TB databases to stateless webapps).
- Strong desire for portability as a means for disaster recovery!



# Example - Hosted Compute Entrypoints

- CEs provide a service that can **accept capacity allocation requests** and translate to a glidein in the local site batch system.
- Can be daunting to run! Requires ‘care and feeding’ as well as expertise on how the distributed system works.
- **Observation:** almost every cluster already runs a remote access protocol, SSH.
- With the ‘hosted CE’ service, OSG staff will run the CE, connecting to the batch system over SSH.

All of this is run by two operators, Colby and Jeff, at about 1 FTE total!

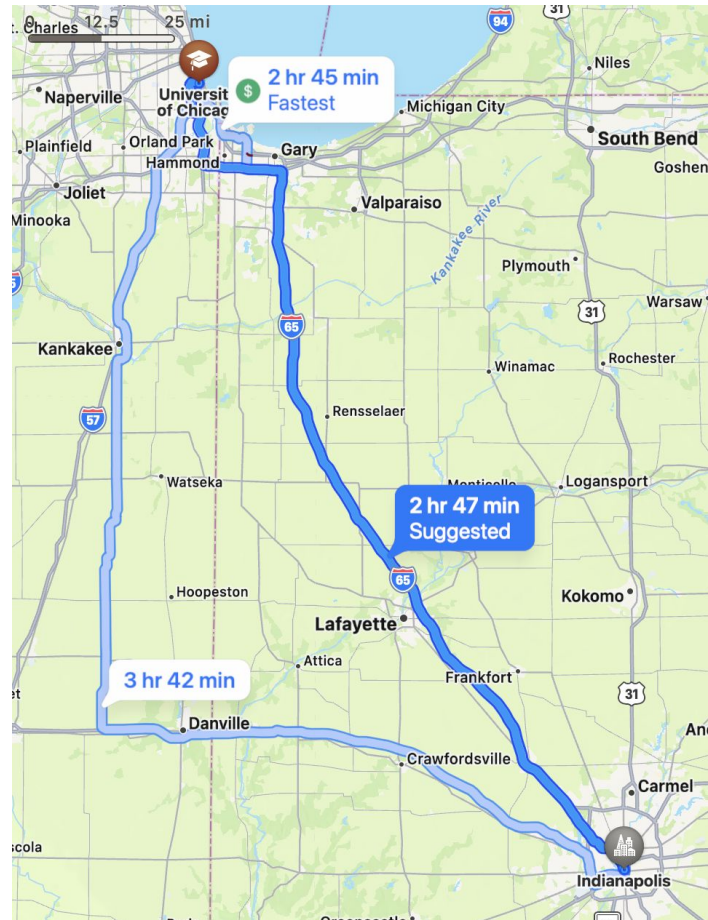


**The price of popularity: ~50 sites use this service.**

# The Fabric of Services, B.K. (Before Kubernetes)

Slide on how we ran the FoS before Kubernetes, pros-vs-cons.

- Services were deployed by the local team according to the local team's philosophy.
  - Puppet, Ansible, by hand, containers, non-containers, local DNS infrastructure...
- Services were indecipherable to those outside the local team.
  - Devs at other universities were unable
- Services were non-portable. Want to move a service from Indiana to Chicago?
  - *Perhaps put the server in the trunk...*



# The Advent of **kubernetes**

Kubernetes has been a game-changer for how we run services:

- **Management of software environment:** Identical software environment between instances - no “special tweaks” of the OS based on site prefs.
- **Service Orchestration:** Operators can describe not just software but also the larger service deployment - network needs, firewall policies, required storage subsystem.
- **Commonality between sites:** Same toolsets to manage services across all sites.

No “siloing” of operators: anyone can help with any service

## ... But Kubernetes is not enough!

Kubernetes is not magic, however:

- “What’s the desired state of this service?”: You can query Kubernetes for the current state – is that the *desired* state? Or is it just where the last operator “left off”?
- “Why was this change made?”: What’s the history of how the service got to this configuration?
- “Oops, I deleted the cluster”: In case of disaster, how do we rebuild?

We found operators were *still* being siloed because they were lacking collaboration tools.



# Introducing GitOps

**Key concept:**

All cluster configuration is kept in a git repository whose state is synchronized to Kubernetes.



# flux

## GitOps: Powered by Flux

All Kubernetes-based services for OSG are synchronized from YAML files kept in Git.

- Operator makes a change, commits it to git, sends in a pull request.
- Buddy reviews the PR, decides it's a good idea, merges.
- The flux operator, which is told to monitor the repository, notices the commit and executes the corresponding changes to the cluster's objects.

```
1  apiVersion: source.toolkit.fluxcd.io/v1beta1
2  kind: GitRepository
3  metadata:
4    name: tiger-osg-config
5  spec:
6    interval: 1m0s
7    ref:
8      branch: master
9    secretRef:
10     name: fluxv2-deploy-key
11    url: ssh://git@github.com/opensciencegrid/tiger-osg-config.git
12  ---
13  apiVersion: kustomize.toolkit.fluxcd.io/v1beta2
14  kind: Kustomization
15  metadata:
16    name: osgdev-config
17  spec:
18    interval: 1m0s
19    path: ./manifests/development
20    prune: false
21    serviceAccountName: flux
22    sourceRef:
23      kind: GitRepository
24      name: tiger-osg-config
25      namespace: osgdev
```

# Other Supporting Roles

Each cluster runs additional operators to provide services that complement the GitOps style:

- **Sealed-secrets**: Allows one to keep encrypted secrets safely in a repo.
- **cert-manager**: Creates host certificates for services as needed
- **External DNS**: Integrates the clusters' services with global DNS.
- **Dex**: Allows users to fetch k8s credentials through SSO
- Useful, but not critical:
  - **Postgres Operator**: Allocates Postgres databases on demand.
  - **Rook**: Provides persistent volumes or filesystems, as needed, from Ceph.
  - **OpenEBS**: Provisions raw block devices for pods.
  - **Traefik**: HTTP layer-7 load balancing.

# GitOps on Tour

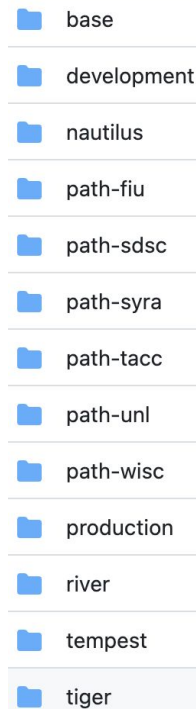
# Core Services

The two biggest clusters used by OSG are Tiger (UW-Madison) and Tempest\* (U. Chicago).

These are plain, **boring** Kubernetes clusters run by GitOps.

Highlights:

- Tiger and Tempest are both in the same Git repository. We can move services between sites – including DNS! – by migrating imports between two different directories.
- In 2021, Tiger had a complete loss of the underlying Kubernetes database. Was able to redeploy the cluster and all services from scratch within two days (while I was on vacation).



- base
- development
- nautilus
- path-fiu
- path-sdsc
- path-syra
- path-tacc
- path-unl
- path-wisc
- production
- river
- tempest
- tiger

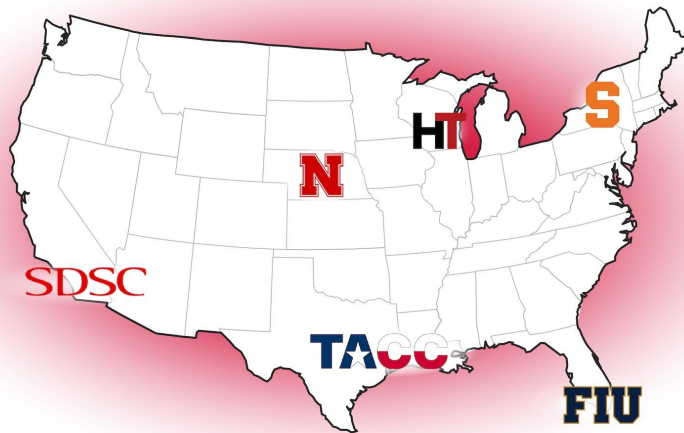
\* Actually in transition from an older cluster at Chicago, River.

# PATH Facility: PATH Production + OSG

- National-scale dHTC service: 30k cores, 36 A100 GPUs
  - Each site maintains hardware and networking
  - PATH Production Services Team provisions Kubernetes at remote sites:
    - Florida International University
    - Syracuse University
    - University of Nebraska
    - University of Wisconsin
  - San Diego Supercomputer Center and Texas Advanced Computing Center provision their own hosts with Kubernetes
- Developers in PATH Production Services deploy dHTC services across distributed Kubernetes infrastructure

## The PATH Facility

*Powering distributed high throughput computing*



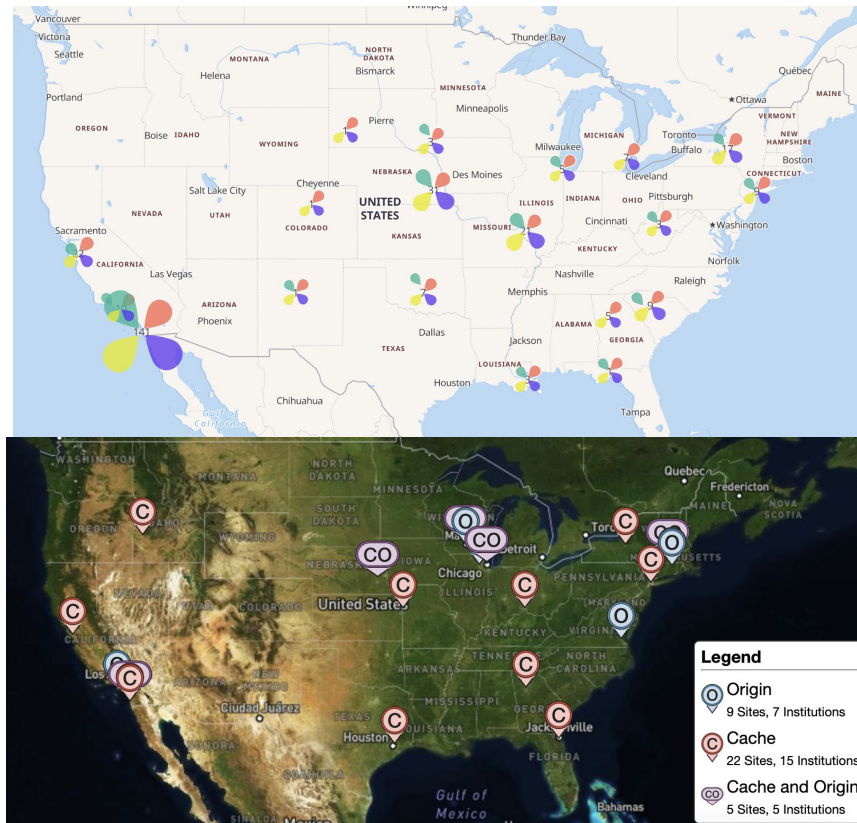
# PATH Facility Challenges

- **One cluster or many?** The PATH project maintains ~6 Kubernetes clusters total to avoid the headache of a single, distributed cluster...
  - ... which may or may not be a good idea! Simpler, yet each upgrade must be done 6x.
- In GitOps, each customization is one patch per cluster.
  - **Death by YAML.**
- There's "Kubernetes" and then there's "Kubernetes": in a complex setup, you notice the subtle difference in deployment styles between sites.
- System complexity can result in difficult-to-troubleshoot issues
  - Potentially mitigated via Operations and architecture manuals
  - New technologies complex at first → organizational education opportunities

## OSG on Nautilus

The Nautilus cluster, run by the NRP, is a distributed Kubernetes cluster spanning the globe.

- OSG runs about a dozen OSDF caches at various network POPs as pods on Nautilus.
- The Nautilus hosts are deployed via GitOps.
- **Interesting twist:** Flux is not installed on Nautilus. Instead, the operator on Tiger synchronizes changes to the remote cluster.





# The Big Picture

# Open Issues & Irritations

No silver bullets, right? What's the catch?

- Flux V2 is configured via custom objects (CRDs) – a learning curve for new operators.
- Since Flux isn't part of Kubernetes core, it sometimes interacts strangely with other operators (its “name prefixes” feature breaks the sealed secrets operator)
- Should environments be setup in branches or directories? We use directories - unclear if that's the right thing. Hard to do one-off environments as in Puppet.
- **Shared repositories means shared breakage:** A bad commit can break deployments for all. Good CI is needed (but difficult).

# Summary

Two technologies - Kubernetes (service orchestration) and Flux (GitOps) - have transformed how the OSG Consortium runs its Fabric of Services.

- Kubernetes enables uniform tooling.
- GitOps wrangles our configuration; while “GitOps” is a trendy name, it’s the same discovery that keeping Puppet configs in git is a good idea.

The approach is flexible, allowing the operations team manage **services in 10 clusters** spanning dozens of locations around the planet.

# Thanks

This material is based upon work supported by the National Science Foundation under Grant No. 2030508. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.