How SLATE uses Kubernetes

Chris Weaver (on behalf of the SLATE team)
University of Chicago

k8s-hep meetup
January 30, 2020

The Need for Federated Edge Platforms

- Science collaborations have computing needs which extend across many sites' computing facilities
  - Especially relevant for Large Facility user communities
- Various types of distributed infrastructure are necessary to make this work efficiently:
  - Caches to manage data distribution (CVMFS, Squid)
  - Data transfer endpoints (Globus, XRootD, Rucio, iRODS)
  - 'Compute elements' to route batch jobs (HTCondor-CE, ARC-CE)
- Managing this infrastructure can be difficult when changes must be coordinated among many parties
  - When administrators at each site must be responsible for running services there is a high barrier to starting a new service, or even to make routine updates to existing services

The Current Model

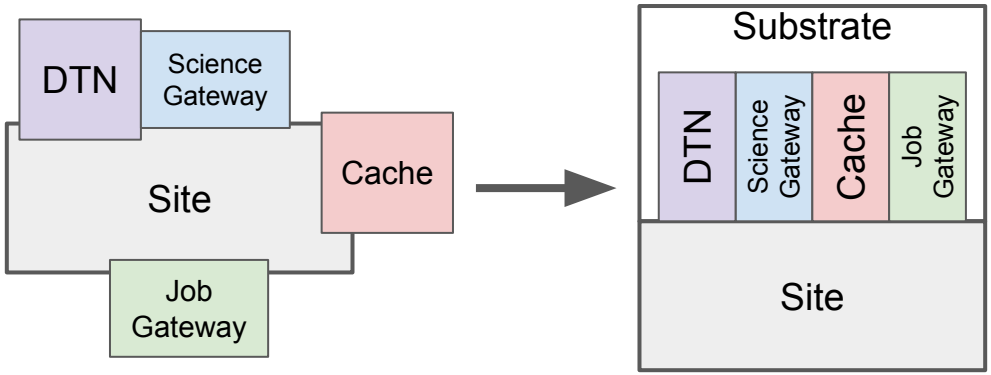
- >150 sites integrated into a production fabric delivering billions of CPU-hours and transferring hundreds of PB/year
- However, it's costly to operate, difficult to innovate, thus slow to change
- Also, software is being "upgraded" to meet the enormous challenge of the HL-LHC era

Re-federating the Edge

- In the WLCG edge of **today** we have
  - **Network diagnostics** (e.g. PerfSONAR)
  - **Data transfer (storage)** endpoints
  - **Compute elements** to route jobs
  - **Software & conditions data caches** (CVMFS, Frontier-squid)
  - **Data caches** (e.g. Xrootd Cache)
- In future?
  - New data delivery services
  - Facility API's (IaaS)
  - Platform APIs (PaaS)
  - ?

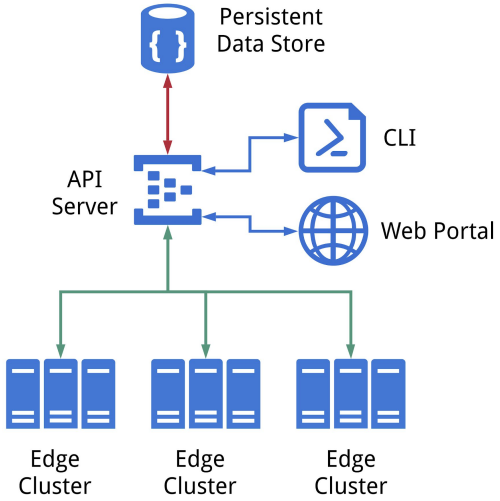
Standardizing a Service Substrate

- By adding a consistent substrate that is common to sites and modular service components which use it, labor can be reduced
- Offers possibility federated operation
  - Or a mix of local and federated



The SLATE Platform for Edge Services

- SLATE (Services Layer at the Edge) provides a substrate for this type of infrastructure
- Docker, Kubernetes, and Helm are used to package and deploy service applications
- A central server component is used to mediate user requests being sent to participating edge Kubernetes clusters
- Command line and web interfaces are provided



https://slateci.io

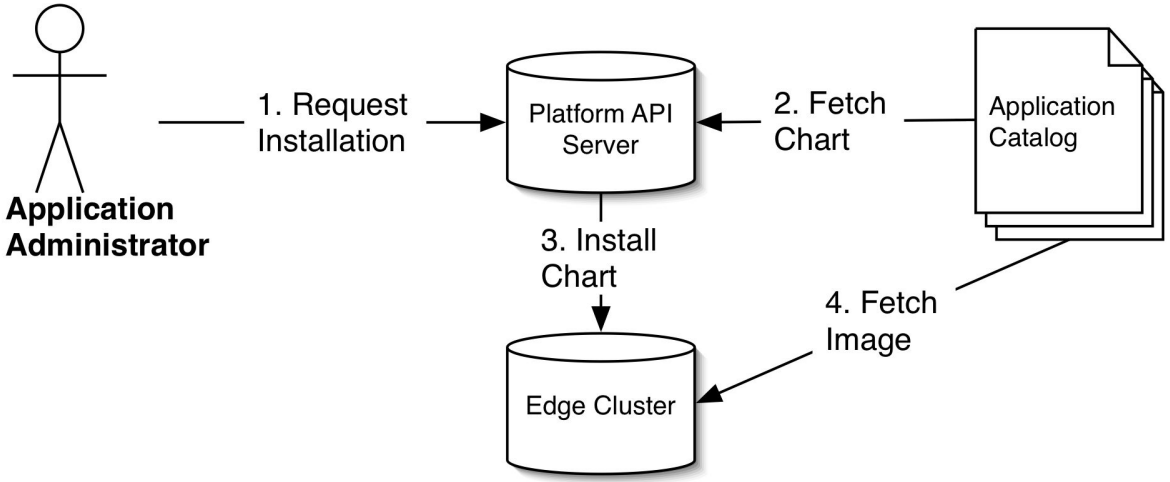
SLATE Goals in using Kubernetes

- Install Helm charts from a trusted repository/catalog
- Prevent users from direct/arbitrary use of Kubernetes
  - Suited to building trustworthy infrastructure, not intended for end-user computation
- Avoid disturbing anything already happening on federated clusters
  - Use the minimum possible privileges
- Separate users' applications from each other

Application (containerized service) Packaging

- SLATE makes use of Helm to package applications for Kubernetes
- Only limited configuration settings for each application are exposed by its Helm chart
  - Hides complexity users don't want to see
  - Can be used to enforce required aspects of configuration
  - Provides a consistent interface which all participants in the federation can inspect and agree on
- SLATE maintains its own catalog of charts, and allows only those applications to be installed
- In future, a WLCG federation may curate & maintain such apps

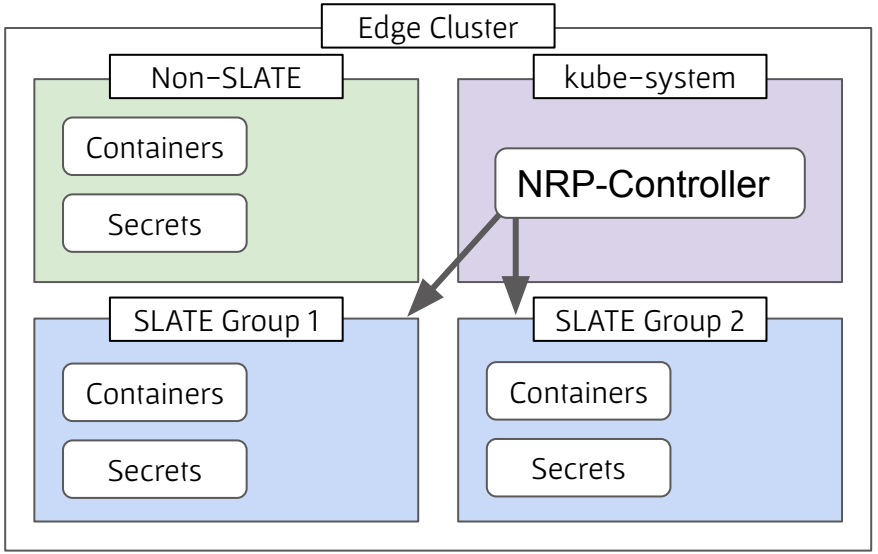
Application Install Process



- The SLATE API server mediates requests to install applications
  - Fetches applications only from the curated catalog
  - Enforces rules set by the administrators of the target cluster

Multi-tenancy with the NRP-Controller

- SLATE uses Kubernetes' namespaces, secrets, Role-Based Access Control (RBAC)
- The SLATE API server is granted access only to its own subset of namespaces
- SLATE places applications belonging to different user groups into separate namespaces
- Kubernetes forbids containers in one namespace from reading secrets in other namespaces



https://gitlab.com/ucsd-prp/nrp-controller

Some Difficulties Encountered

- Federation mechanisms have been very slow to mature
  - This is why SLATE opted for a custom solution, which is mostly outside of Kubernetes
- LoadBalancers are key components for 'nice, easy-to-use' clusters, but are hard for users to get insight into
  - How do you tell if there is a LoadBalancer?
  - How do you tell if it has addresses to give out?
- Helm templating has been half highly useful, and half maddeningly restrictive
  - Kind of poor documentation has made things harder

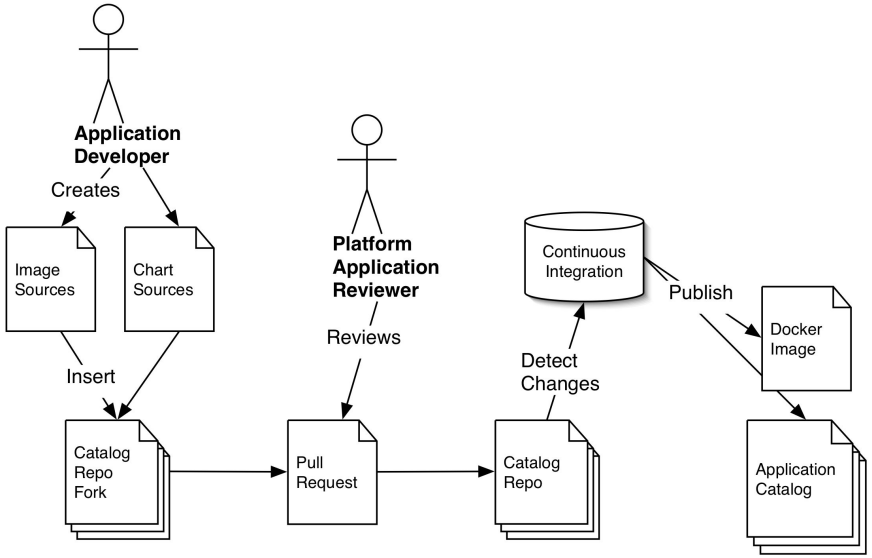
Future Work in SLATE

- Fully exposing PriorityClasses and StorageClasses to SLATE users
  - A trivial change to RBAC, but we need to improve our mechanisms for rolling that out
- Monitoring with Prometheus as an optional add-on
  - Prometheus can do a lot; we are still learning about how to use it well,
- Supporting resource limits and requests
  - Want to expose+enforce this at the SLATE level, so that cluster admins can specify how many cores or how much RAM a guest group may use, etc.
- Formalizing security and application curation rules

Thanks

This work is supported by the National Science Foundation Office of Advanced Cyberinfrastructure (OAC), grant number 1724821

Application Curation



- Much of the value of the centralized application catalog derives from the overesight applied to the applications added to it
- Some amount of human attention is required, but maximizing automation is highly desirable

Curation Considerations

- Both Helm charts and the container images they reference must be taken into consideration
- The review process must be deep enough to be able to prevent problems, but not so slow or restrictive that potential users of the platform cannot get appropriate applications into production
  - Reviewer effort is also a limited resource!
- Some trusted sources are needed as a basis
  - The community already trusts major OS distributions (CentOS, Ubuntu, etc.)
  - Some applications are provided by major groups within the wider community which already have their own processes for trustworthy releases (Apache httpd, NGINX)

We are using SLATE to manage squids...

- OSG announced a vulnerability in Frontier Squid on July 26
- SLATE instances were all updated within the hour with this script:

```
for i in $(slate instance list | grep squid | awk '{print $4}'); do
   slate instance restart $i
done
```

Evolving Trust and Privilege in WLCG

- Infrastructure services are qualitatively different from the batch jobs that many sites already accept from outside users—they must run persistently, and must often accept network connections from outside
- To admit such services, site administrators need strong guarantees that:
  - Only suitable persons will be able to deploy services
  - Only appropriate software for providing a relevant service will be run
  - Services will use appropriately secure software and configuration
  - External users running services will not interfere with existing uses of resources
- Users running services also want separation between their applications and others'

Broader Policy Concerns

- The SLATE Team has been working on an engagement with **TrustedCI.org**, with one major goal being to design security policies and procedures
- Incident Response and Disaster Recovery have been identified as particularly critical areas
  - Incident Response, in particular can involve multiple sites, and a need to share information in a timely manner
- We think that getting these policy areas structured correctly is key building a useful platform
- Eventually, we hope to have policies which can themselves be considered sufficiently standard for broad adoption by the community
  - This means that we need to form a clear picture of what sites' concerns are
  - The WLCG (CERN Large Hadron Collider) has set up a working group to investigate these ideas as well

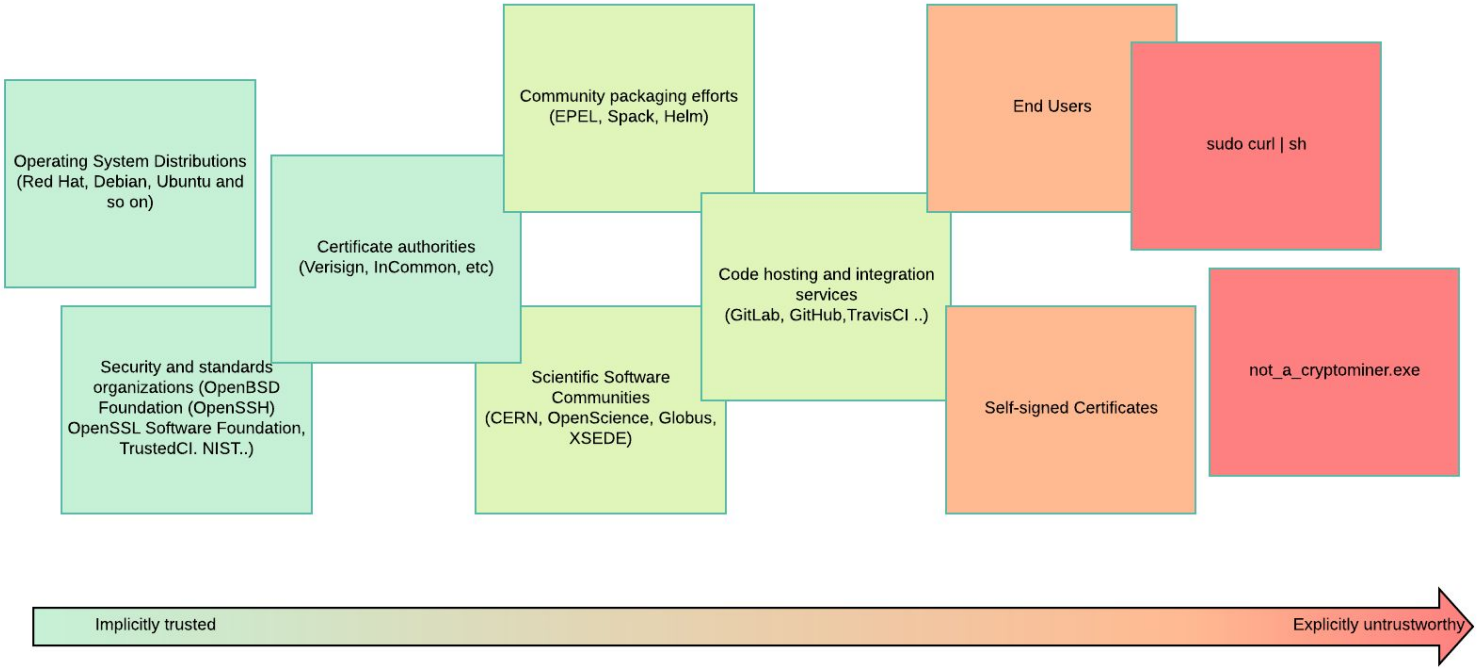
An Aside On Containers

- Linux Containers are a useful technology for implementing this type of platform, but are not fundamental to the general strategy of building a unified abstraction across multiple sites' resources
- It would be conceptually equivalent to build a substrate using Virtual Machines (leveraging OpenStack, for example), or using configuration management tools (such as Puppet or Ansible)
- Security concerns can be divided into two categories:
  - Those which are generally applicable to any distributed service platform, such as how users are granted access to sites' resources
  - Those which are specific to using containers to implement such a platform, such as the provenance of the container images and details of the container runtime

Internal Permissions Model

- SLATE organizes users into groups, and permissions apply per-group
- Every participating cluster is administered by a group
  - When a cluster first joins the federation, only its administering group has access
- The administrators of a cluster can:
  - Grant access to other groups to deploy applications on their cluster
  - Set up per-group whitelists of which applications guest groups are authorized to deploy
- The site administrator always retains the capability to directly work with the underlying Kubernetes layer to perform actions beyond what SLATE directly supports
  - This means that local admins have no restriction on inspecting, editing, or removing components if needed

Variation in Trust Levels



Special Challenges of Container Images

- Container images are a snapshot of a system state, so they do not tend to be aware of security patches since their creation
  - This implies that periodic rebuilding of images is necessary, and possibly that containers should be periodically restarted
- Typical distribution mechanisms (Docker) allow the data referred to by a particular image 'tag' to be replaced—an image which was previously reviewed may be replaced by one with different contents
  - This is why we prefer to have SLATE manage image sources, build and publish the images to a repository itself
- Automated image scanning tools can help with review, but are not a complete answer
  - Only images containing package manager data can be scanned
  - Scans may find large numbers of low-importance vulnerabilities for which no patched packages are available from the base distribution

Risks of containers as defined by NIST

- Broadly, NIST has identified* 5 areas of risk with application containers:
  - Image Risks
    - Configuration defects, malware, embedded secrets, untrusted software
  - Registry Risks
    - Insecure connections, stale images, insufficient authentication and authorization
  - Orchestrator Risks
    - Unbounded administrative access, unauthorized access, mixed sensitivity of workloads and poor separation between workloads
  - Container Risks
    - Vulnerabilities in the Container runtime, insecure runtime configurations, unbounded network access
  - Host OS Risks
    - Large attack surface, shared kernel, host filesystem tampering, host component vulnerabilties

* https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-190.pdf

Risks and Mitigations

- Image Risks
  - SLATE uses a curated application catalog with specific requirements for containers that may be in the catalog.
- Registry Risks
  - The SLATE team is currently considering running a registry independent of the common ones, e.g. DockerHub to have more control over images delivered via the platform
- Orchestrator Risks
  - SLATE allows the operator of the cluster to limit which applications may be launched
  - Additionally, the Kubernetes API need only be open publicly to a few specific IPs for SLATE access
- Container (runtime) Risks / Host OS risks
  - SLATE will offer best-practices for runtime and host configuration, but largely this is left up to the Cluster administrator

* https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-190.pdf

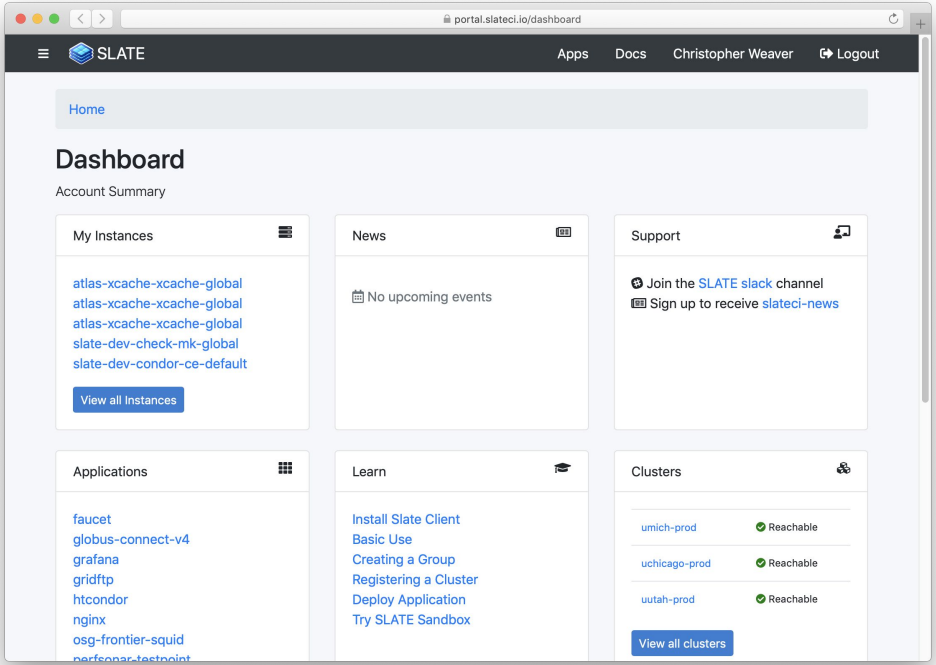
Application Configuration Example

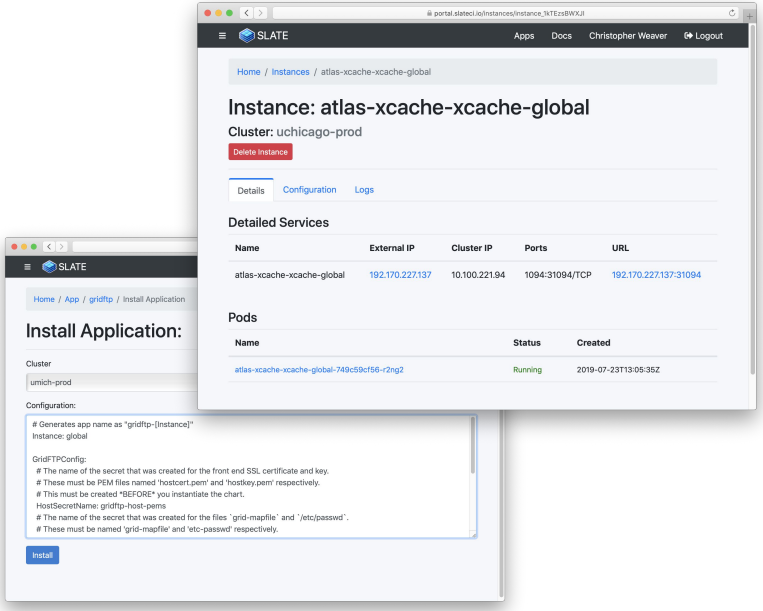
```
# Instance to label use case of Frontier Squid deployment
# Generates app name as "osg-frontier-squid-[Instance]"
# Enables unique instances of Frontier Squid in one namespace
Instance: global

SquidConf:
  # The amount of memory (in MB) that Frontier Squid may use on the machine.
  # Per Frontier Squid, do not consume more than 1/8 of system memory with Frontier Squid
  CacheMem: 128
  # The amount of disk space (in MB) that Frontier Squid may use on the machine.
  # The default is 10000 MB (10 GB), but more is advisable if the system supports it.
  # Current limit is 999999 MB, a limit inherent to helm's number conversion system.
  CacheSize: 10000
  # The range of incoming IP addresses that will be allowed to use the proxy.
  # Multiple ranges can be provided, each separated by a space.
  # Example: 192.168.1.1/32 192.168.2.1/32
  # The default set of ranges are those defined in RFC 1918 and typically used
  # within kubernetes clusters.
  IPRange: 10.0.0.0/8 172.16.0.0/12 192.168.0.0/16
```

SLATE Command Line Interface

```
# Find the PerfSONAR testpoint application
$ slate app list | grep 'Name\|perfsonar'
Name App Version Chart Version Description
perfsonar-testpoint 4.2.0 1.0.3 Perfsonar Testpoint Deployment
# Get the default configuration
$ slate app get-conf perfsonar-testpoint > ps.yaml
# Customize the configuration
$ vi ps.yaml
# Do the install
$ ./slate app install perfsonar-testpoint --cluster uchicago-prod --group slate-dev --conf ps.yaml
Successfully installed application perfsonar-testpoint as instance slate-dev-perfsonar-testpoint-cnw- test
with ID instance_U-2KiIGqFKs
# Query instance information
$ ./slate instance info instance_U-2KiIGqFKs
Name                          Started                  Group     Cluster       ID
perfsonar-testpoint-cnw-test 2019-Jul-15 18:06:39 UTC slate-dev uchicago-prod instance_U-2KiIGqFKs
Pods:
  slate-dev-perfsonar-testpoint-cnw-test-84596d7c85-ns8xk
    Status: Running
    Created: 2019-07-15T18:06:44Z
    Host: sl-uc-xcache1.slateci.io
    Host IP: 192.170.227.137
# Run a test against the new endpoint
$ pscheduler task rtt --dest 192.170.227.137
Waiting for result...
1       192.170.227.137  64 Bytes  TTL 64  RTT   0.2690 ms
...
0% Packet Loss  RTT Min/Mean/Max/StdDev = 0.117000/0.190000/0.269000/0.051000 ms
```

SLATE Web Interface

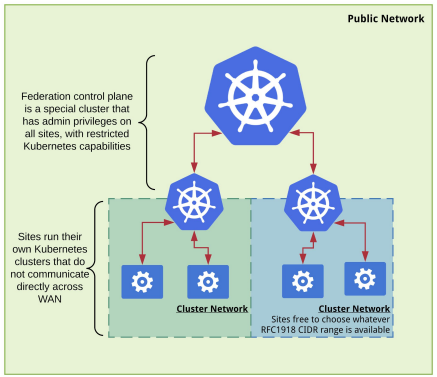




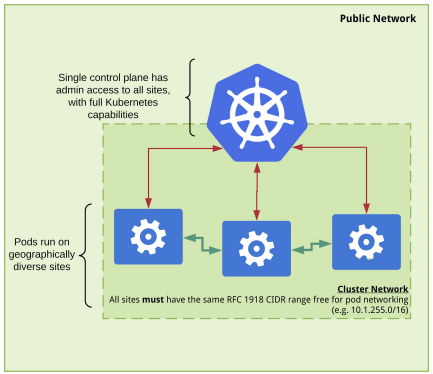
- Can be used by the local admin team, experiment Ops teams, etc
- Who can do what is totally under local admin control

Approaches to Kubernetes Federation

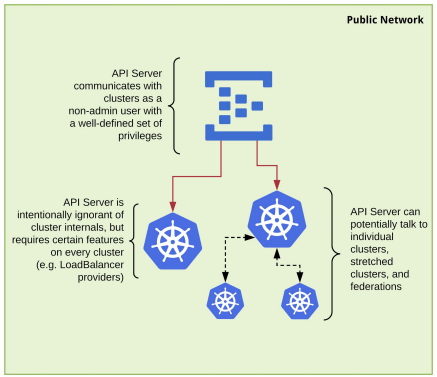
- Native federation (KubeFed) is still not mature (in alpha testing as of July 2019)
- 'Stretched' Kubernetes clusters become unwieldy at large scales, and have implications for networking
- For SLATE, giving users direct `kubectl` access to participating clusters was not a specific goal (and restricting what users can do is much easier without it)



FEDERATION CONTROL PLANE



STRETCHED KUBERNETES (a la PRP)



SLATE PSEUDO-FEDERATION