Lightweight WLCG Sites: Remote DevOps with SLATE

Rob Gardner
University of Chicago

ATLAS Sites Jamboree
March 4-8, 2019

Outline

- Goal
- What is SLATE
- SLATE Architecture
- Provisioning Options
- Deploying services (e.g. XCache)
- Developing with MiniSLATE
- Getting Started
- Extras

Goal

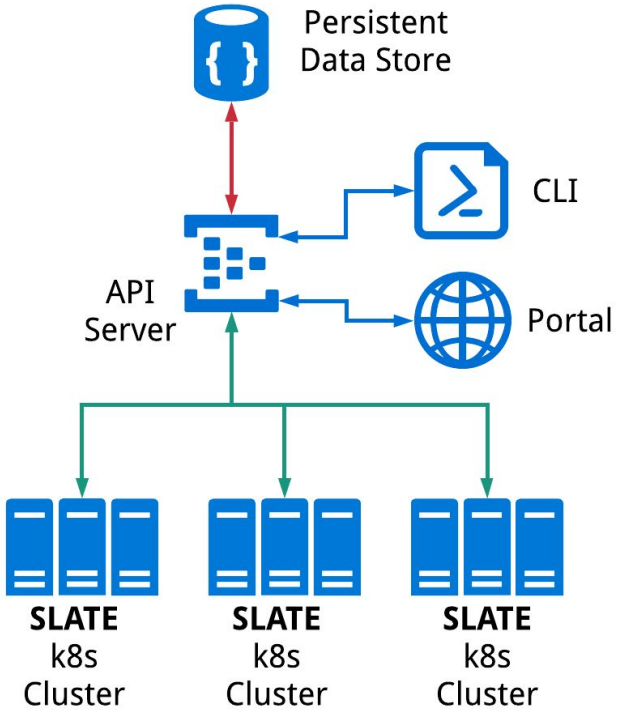
- Remotely manage edge services at sites (e.g. squid, xcache, ...) with central expert teams
- Deploy updates more quickly
- Introduce new services more easily
- Save time and effort for the local admins

Create a federation of edge clusters

- **SLATE: S**ervices **L**ayer **A**t **T**he **E**dge
- Distributed service orchestration platform
- Kubernetes-based
- Start with a single server and scale as needed
- Loosely federated, share projects/users/applications across institutions
- Good for any site but "**lightweight**" sites might find it particularly useful

Basic SLATE Architecture

- Lightweight federation and application catalog layer on top of Kubernetes
  - Security-conscious, site autonomous
  - Sites retain administrative control
- Single entrypoint using institutional identity
- Simple UNIX-like permissions model (Users + Groups)
- **Application catalog** provides natural boundary between configuration knobs users actually want to change and complex Kubernetes configurations
- SLATE is an infrastructure **and software**



Create & manage your own federation over independently managed Kubernetes clusters

SLATE provisioning options

- **SLATE**Lite (for a quick evaluation using Docker):
  - https://github.com/slateci/slatelite
- Zero to k8s+**SLATE** script on a bare edge server:
  - Installs everything necessary starting from a fresh CentOS system
    http://jenkins.slateci.io/artifacts/scripts/install-slate.sh
- "Managed" install
  - **We** will SSH to your site, set it up, and hand you the configured machine.
- Full install
  - **You** install Kubernetes, download **SLATE** client and register your cluster

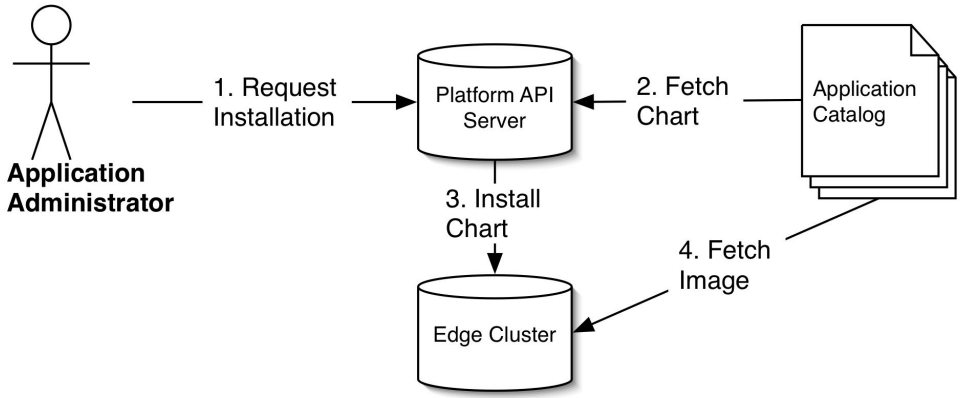
Registering a cluster

```
$ slate cluster create atlas-t2-xyz \
  --group atlas-xyz-admins \
  --org "ATLAS Tier 2 XYZ"
$ slate cluster allow-group atlas-xcache
```

- Join a kubernetes cluster to a SLATE federation
  - Specifying the group which will administer it and the organization which owns the resource
- Grant users access to deploy applications on the cluster
  - In this case, just the atlas-xcache group

Deploying Services ("Applications" in k8s)

- A "central" service expert deploys & operates many sites
- Helm charts and Docker images
- Command line or web interface (in dev)



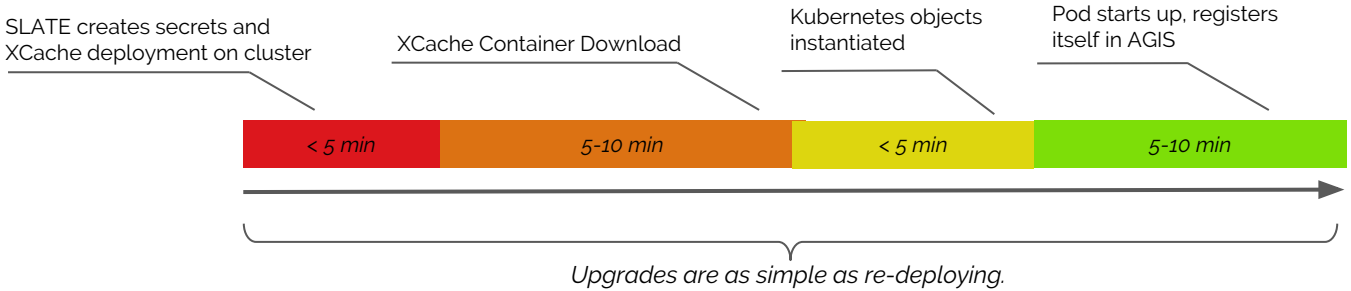
Deployment experience in ATLAS

- Goal: build an XCache-based caching network as part of the DOMA activity
- SLATE-based deployment will simplify operations and allow for rapid development and debugging
- SLATE services already operational at MWT2, AGLT2, LRZ
- XCache application already in SLATE catalog
  - Ilija is developing & testing daily

XCache deployment process

- Register a cluster with SLATE and allow the `atlas-xcache` group
- Apply a few special extra steps for XCache:
  - Node labeled in Kubernetes (`xcache-capable=true`)
  - One or more storage volumes mounted (e.g. `/xcache`) & communicated to Ilija
  - Endpoint protocol registered in AGIS
- Test suite containerized
  - Launch a very realistic stress test from Google Compute Engine in minutes

XCache Deployment & Upgrade Cycle:



XCache fully deployed in less than 20 minutes.

XCache deployment process (more details)

- As XCache requires special resources this has to be communicated between Ilija and the site but is done only once:
  - Dedicated node labeled in K8s.
  - Storage should be JBODs organized.
  - Endpoint protocol registered in AGIS.
- Ilija takes over and creates secrets, server, reporting, monitoring, activates protocol in AGIS.
  - All of that is two commands and takes 30 seconds.
- Full update of all the caches in SLATE should take less than 20 min.
  - 10-15 minutes for dockerhub to rebuild image
  - 1 minute to stop running instances
  - 1 minute to start them again
  - 3 minutes to check everything worked
  - Even stress testing is containerized and Ilija can run a very realistic stress test against any xcache in matter of minutes (from Google Computing Engine)

XCache updates

- Even simpler
- Completely transparent to site admin.

```
$ slate instance list
$ slate instance delete <instance name>
$ slate app install --group atlas-xcache --cluster uchicago-prod --conf MWT2.yaml xcache
```

Additional benefits:

- Automatic core dump collection
- Containerized environment makes it easier to debug

Monitoring

Wealth of information collected even without any direct XRootD monitoring (summary or detailed stream).

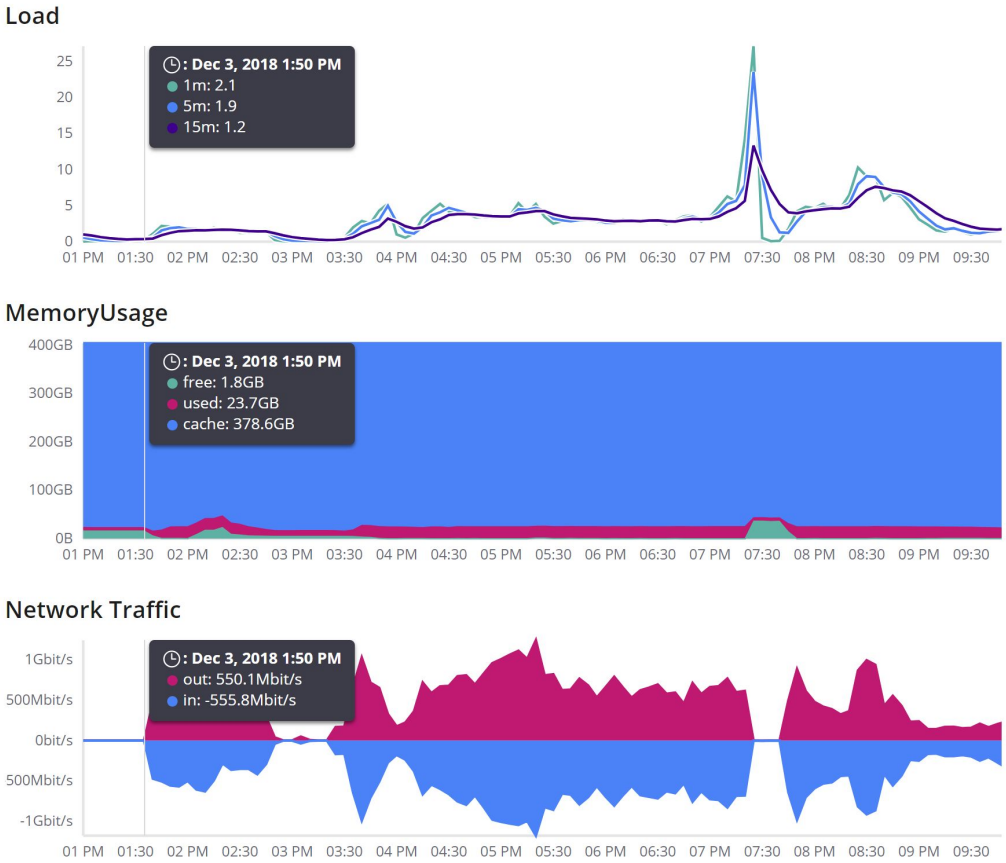
Node state (load, mem, network).

Per pod/container event and resource usage.

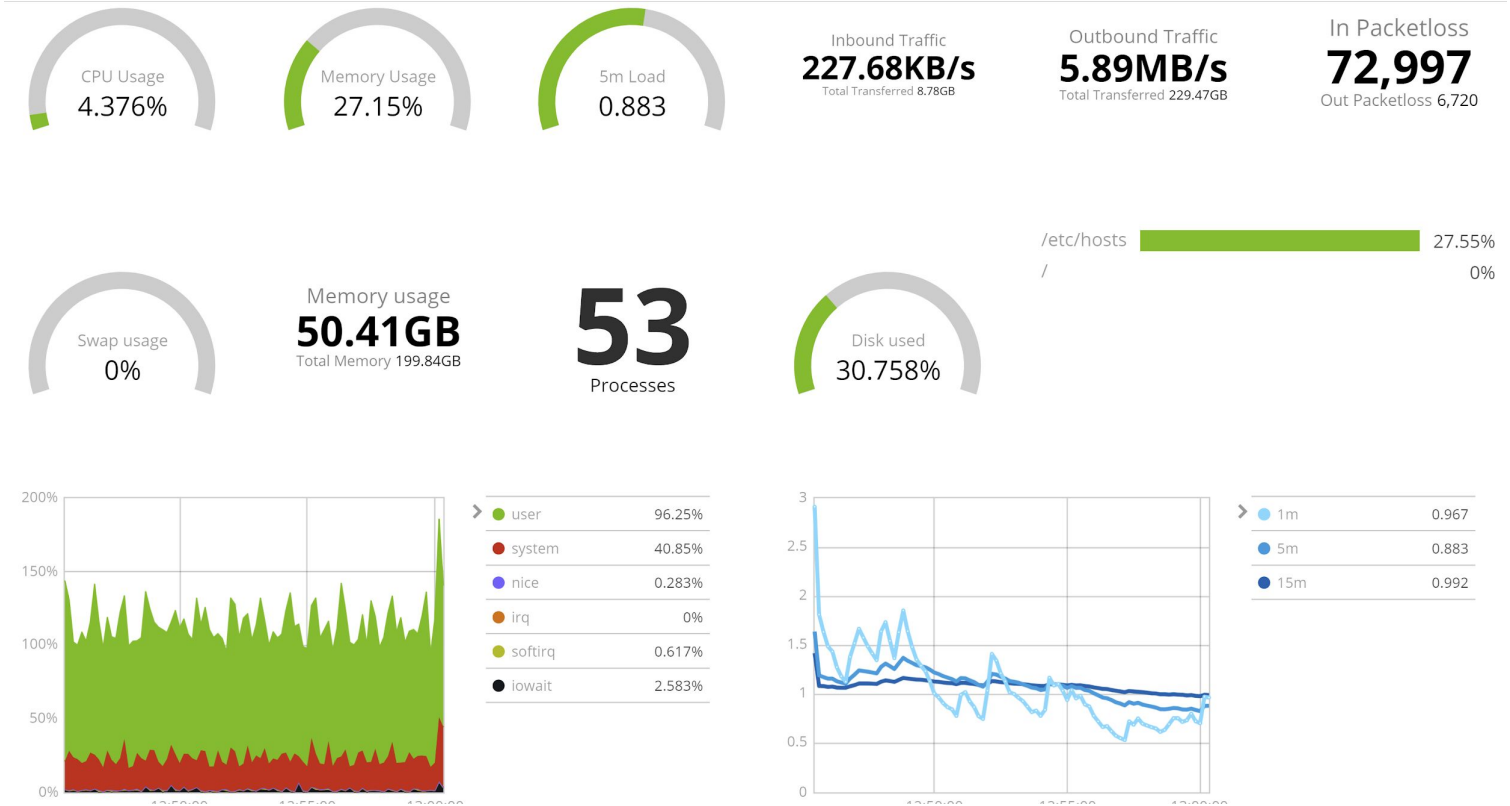
Logs. Fully searchable.

All info shipped to Elasticsearch at UChicago.

WIP - Prometheus-based monitoring



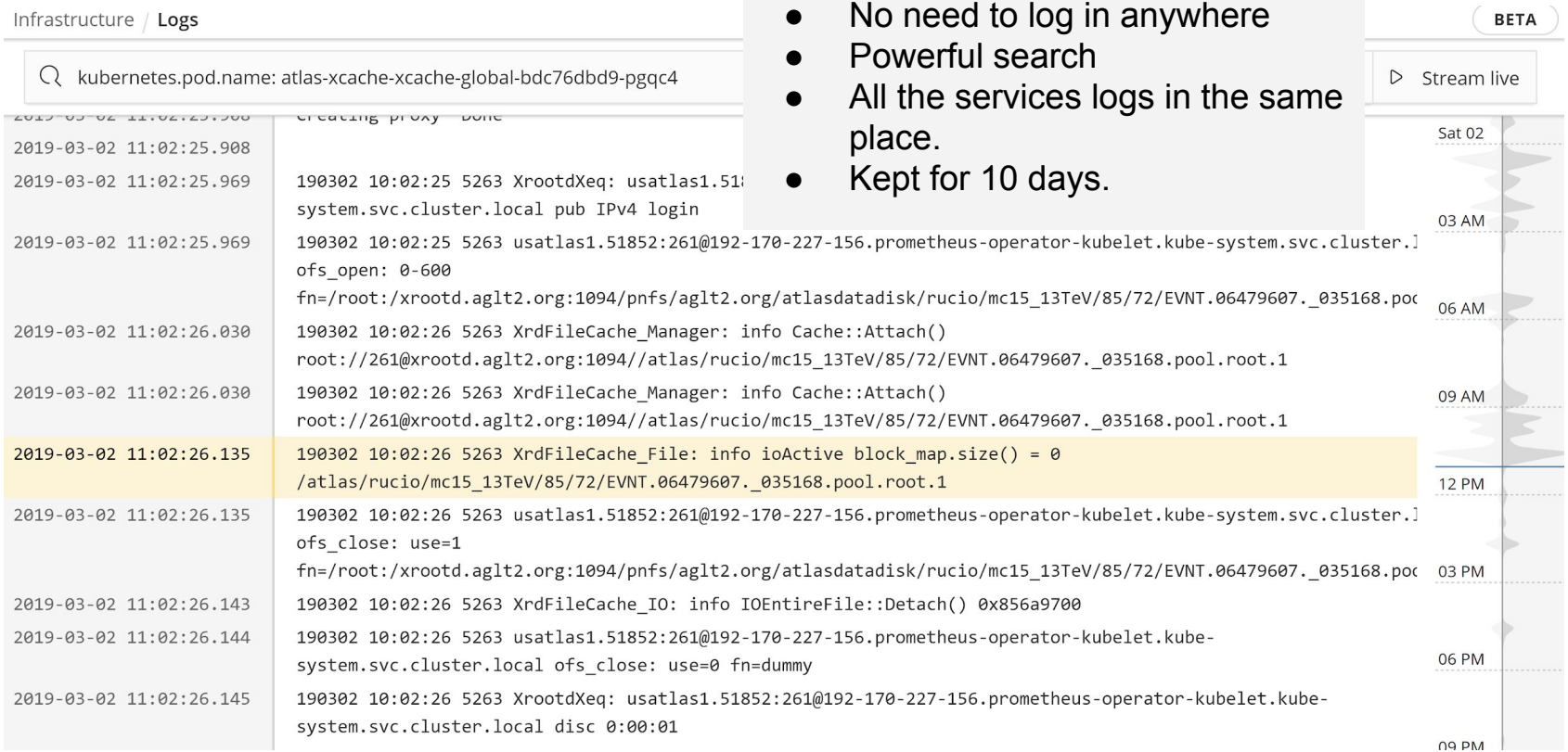
Monitoring - ES & Kibana



Monitoring continued

Really convenient logging

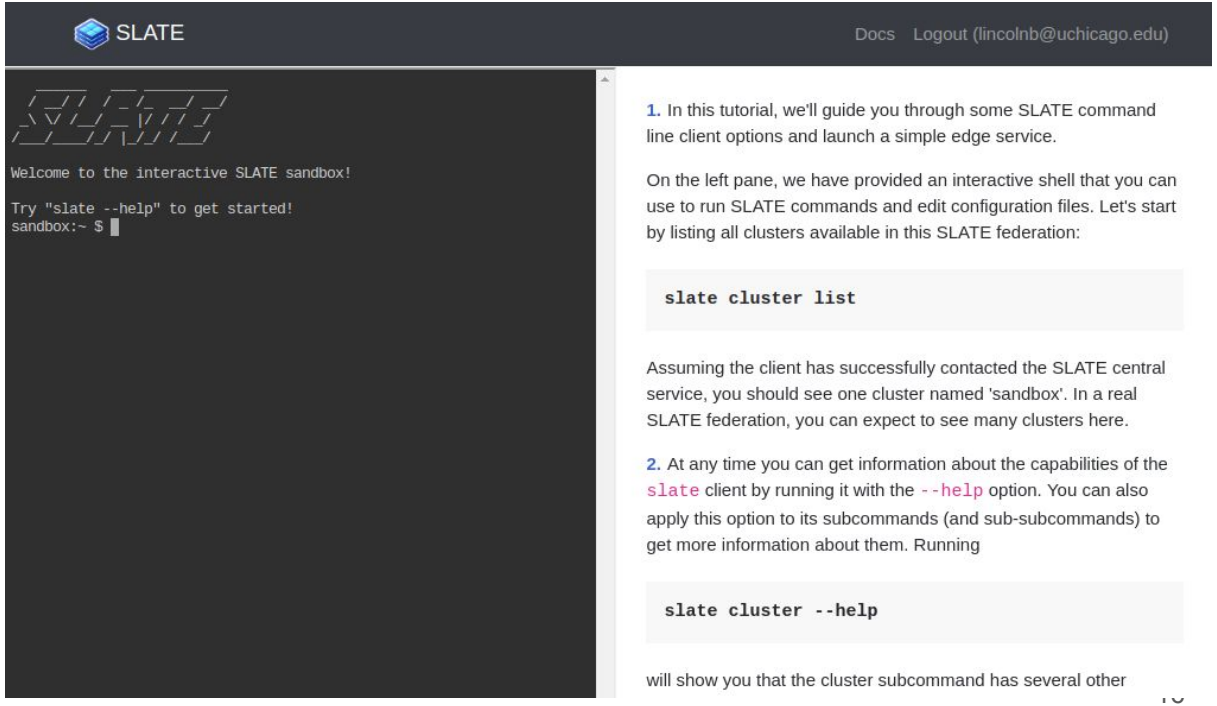
- No need to contact anyone
- No need to log in anywhere
- Powerful search
- All the services logs in the same place.
- Kept for 10 days.



Get a feel for it - SLATE "Sandbox"

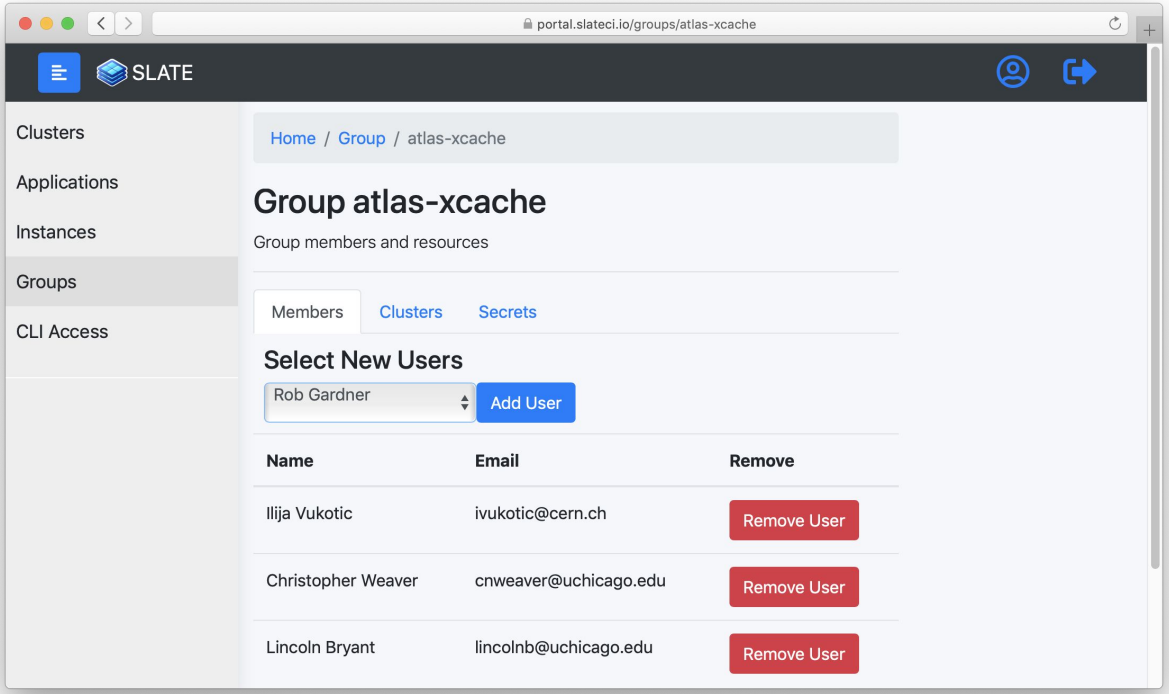
- Starts a tutorial environment inside a kubernetes pod with the slate client
  - Runs an instance of the SLATE API and exposes the cluster
- Anyone can make a temporary account, try out the command line interface, and deploy a simple web server application

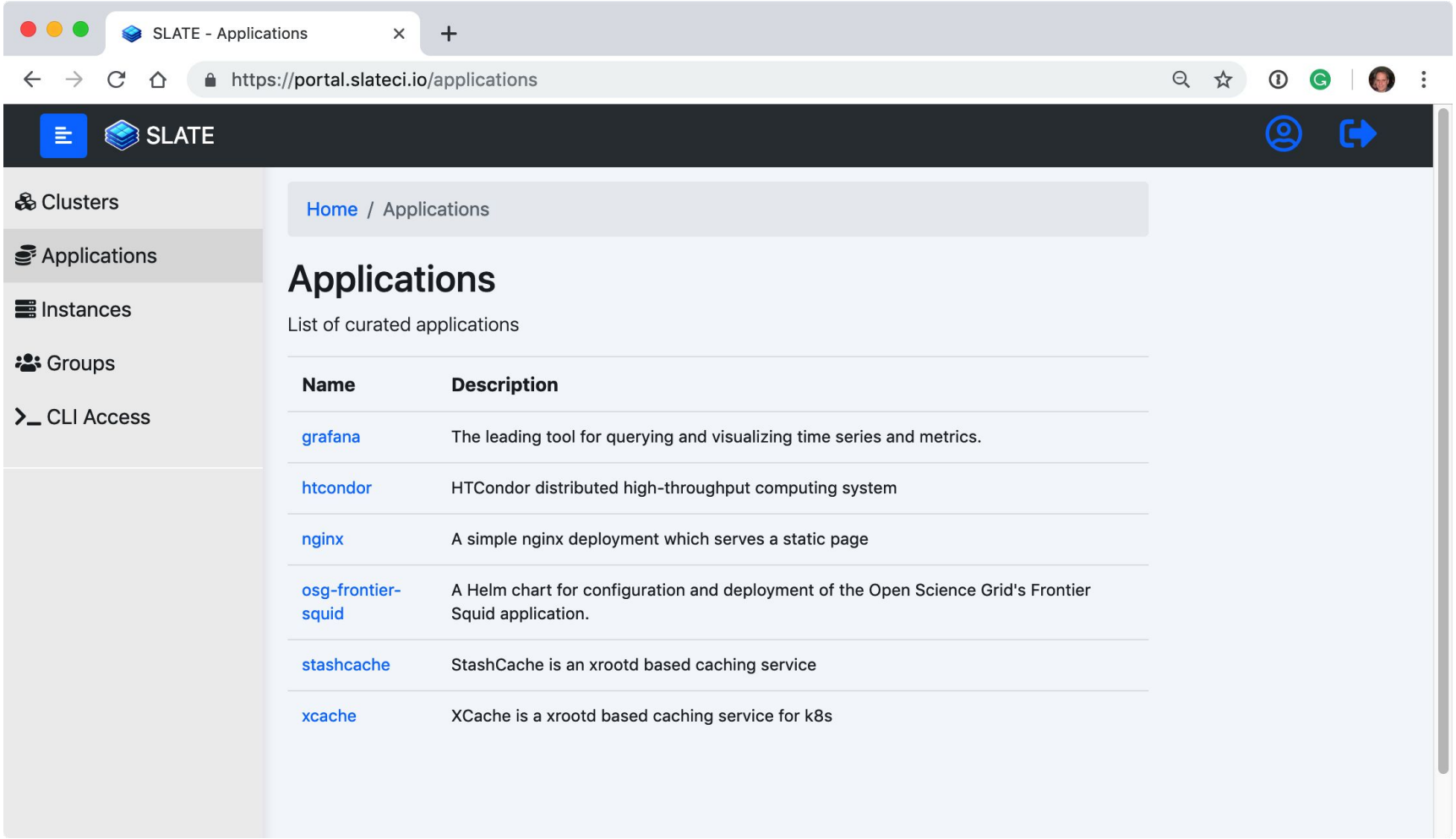
https://sandbox.slateci.io:5000/

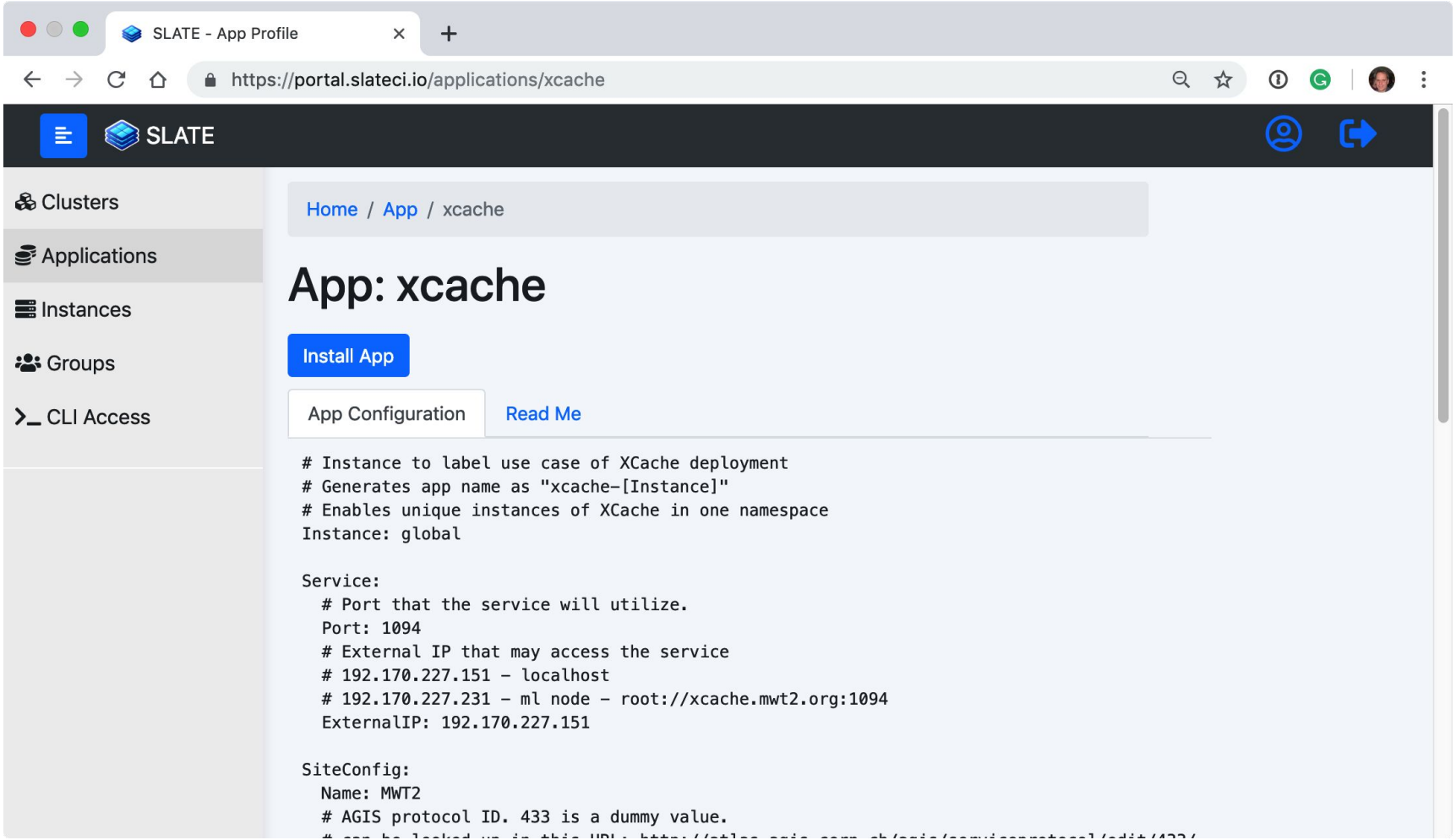


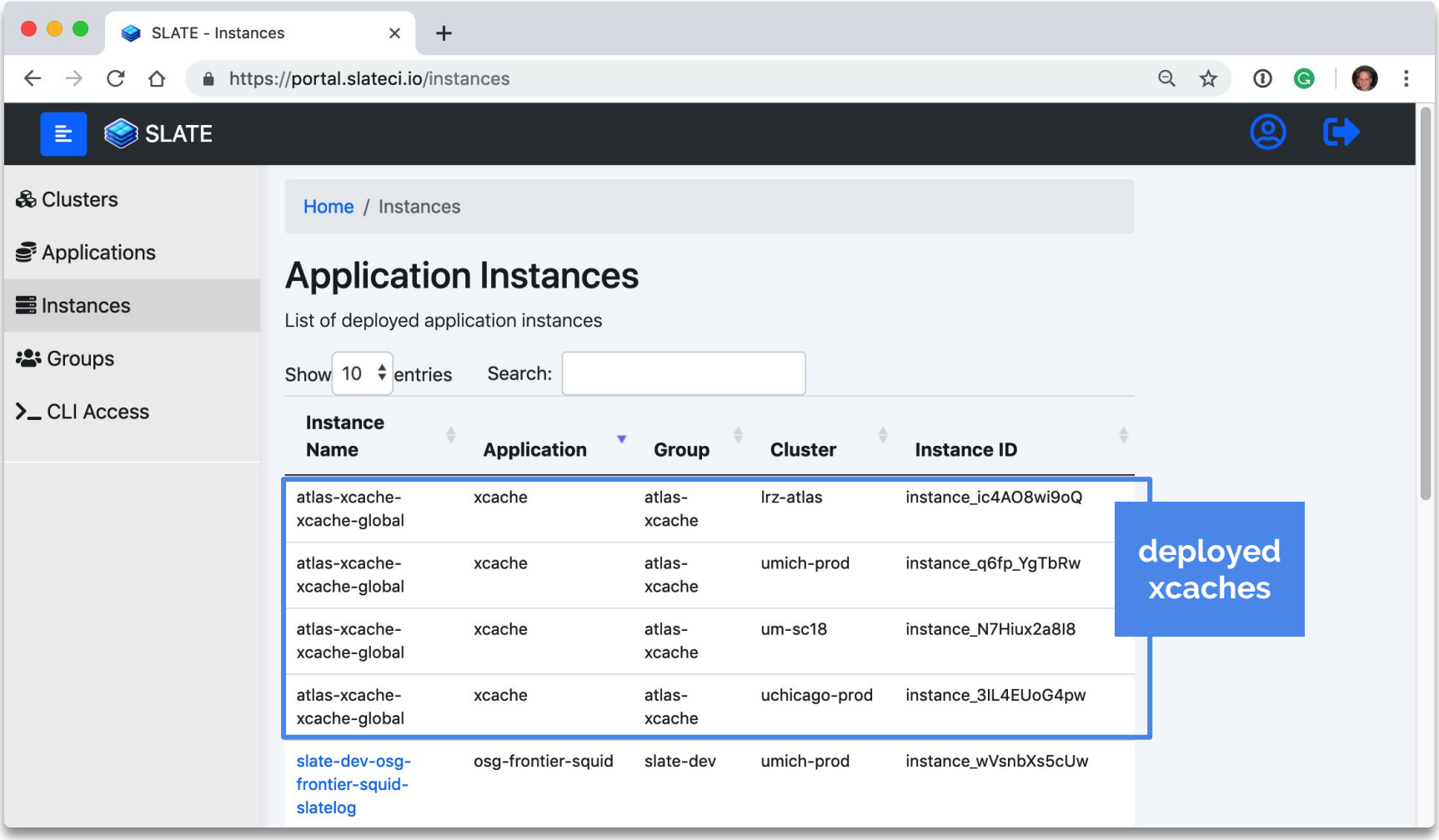
SLATE portal (in dev)

- A convenient graphical interface for most functions of SLATE
- For example, managing the users who belong to a group




SLATE - Applications
https://portal.slateci.io/applications
SLATE
Clusters
Applications
Instances
Groups
CLI Access
Home / Applications
Applications
List of curated applications
Name
Description
grafana
The leading tool for querying and visualizing time series and metrics.
htcondor
HTCondor distributed high-throughput computing system
nginx
A simple nginx deployment which serves a static page
osg-frontier-squid
A Helm chart for configuration and deployment of the Open Science Grid's Frontier Squid application.
stashcache
StashCache is an xrootd based caching service
xcache
XCache is a xrootd based caching service for k8s

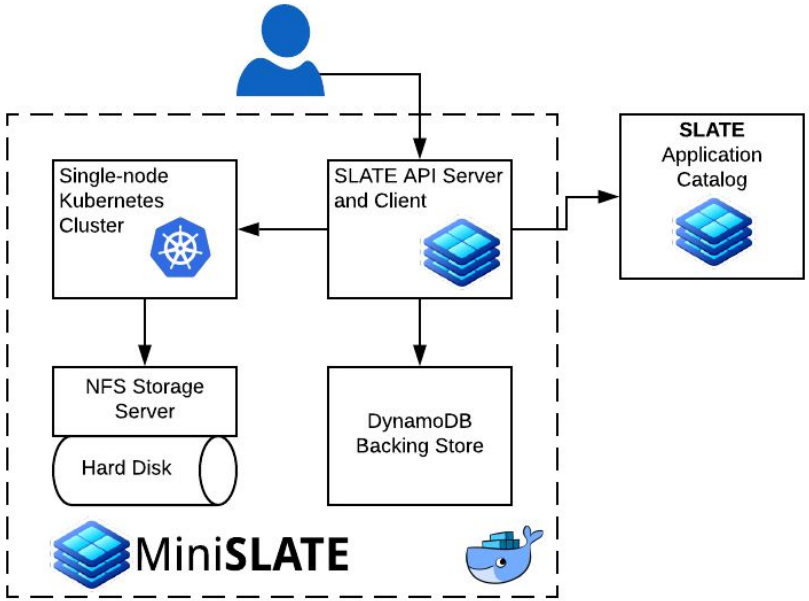

SLATE - App Profile
https://portal.slateci.io/applications/xcache
SLATE
Clusters
Applications
Instances
Groups
CLI Access
Home / App / xcache
App: xcache
Install App
App Configuration
Read Me
Instance to label use case of XCache deployment
Generates app name as "xcache-[Instance]"
Enables unique instances of XCache in one namespace
Instance: global
Service:
Port that the service will utilize.
Port: 1094
External IP that may access the service
192.170.227.151 - localhost
192.170.227.231 - ml node - root://xcache.mwt2.org:1094
ExternalIP: 192.170.227.151
SiteConfig:
Name: MWT2
AGIS protocol ID. 433 is a dummy value.


SLATE - Instances
https://portal.slateci.io/instances
SLATE
Clusters
Applications
Instances
Groups
CLI Access
Home / Instances
Application Instances
List of deployed application instances
Show 10 entries
Search:
Instance Name
Application
Group
Cluster
Instance ID
atlas-xcache-xcache-global
xcache
atlas-xcache
lrz-atlas
instance_ic4AO8wi9oQ
atlas-xcache-xcache-global
xcache
atlas-xcache
umich-prod
instance_q6fp_YgTbRw
atlas-xcache-xcache-global
xcache
atlas-xcache
um-sc18
instance_N7Hiux2a8I8
atlas-xcache-xcache-global
xcache
atlas-xcache
uchicago-prod
instance_3IL4EUoG4pw
slate-dev-osg-frontier-squid-slatelog
osg-frontier-squid
slate-dev
umich-prod
instance_wVsnbXs5cUw
deployed xcaches

Mini**SLATE**

A development environment for SLATE

- **Create a stand alone, miniature SLATE federation for development**
- Follows an Infrastructure as Code pattern
- Enclosed within Docker
  - Little dependency clutter
    - Python, Docker, Docker-Compose
  - Environment consistency
- Completely Destructible
  - Destroy and recreate at will
  - Mount code from host safely
- Batteries Included
  - Full development kit
  - All required software and useful tools are installed when the Docker image is built



Installing Mini**SLATE** (https://github.com/slateci/minislate)

```
$ git clone https://github.com/slateci/minislate.git
Cloning into 'minislate'...
$ cd minislate
$ ./minislate init
(...)
DONE! MiniSLATE is now initialized.
$ ./minislate slate app install nginx --group ms-group --cluster ms-c

Installing application...
...
Successfully installed application nginx as instance ms-group-nginx-default with ID
instance_tey72YzGYuw
```

State of SLATE for ATLAS

- Three sites operational (AGLT2, MWT2, LRZ)
  - Deployment in development at Innsbruck (OpenStack)
- API server and client relatively stable at this point
  - Adding features as needed by use-case
- Web portal under development
- Platform monitoring under development
- Application catalog starting to gel
  - XCache, HTCondor CE, Squid, various other apps stabilizing
- Best practices evolving and being documented

If you'd like to try out SLATE

- Homepage: http://slateci.io
- Slack: http://bit.ly/slate-slack-03
- Discussion list
- Or just email robert.w.gardner@cern.ch

Acknowledgements

- SLATE team members in particular who did the work and provided input
  - Lincoln Bryant
  - Ben Kulbertis
  - Chris Weaver
  - Jason Stidd
- SLATE dashboard
  - Ilija Vukotic
- SLATE portal
  - Jeremy Van
- SLATE website
  - Shelly Johnson

Extra slides

SLATE Hardware - example ATLAS edge server

Standard SLATE config:
http://slateci.io/docs/slate-hardware/atlas-node.html

- (2) Xeon Silver 4110 (32 HT cores)
- (12) 16GB RDIMMs (192 GB RAM)
- (4) 2TB NVMe + NVMe holder/adapter
- (12) 12TB disks (144TB raw storage)
- (1) 240GB BOSS card
- (1) 2x10Gb NIC

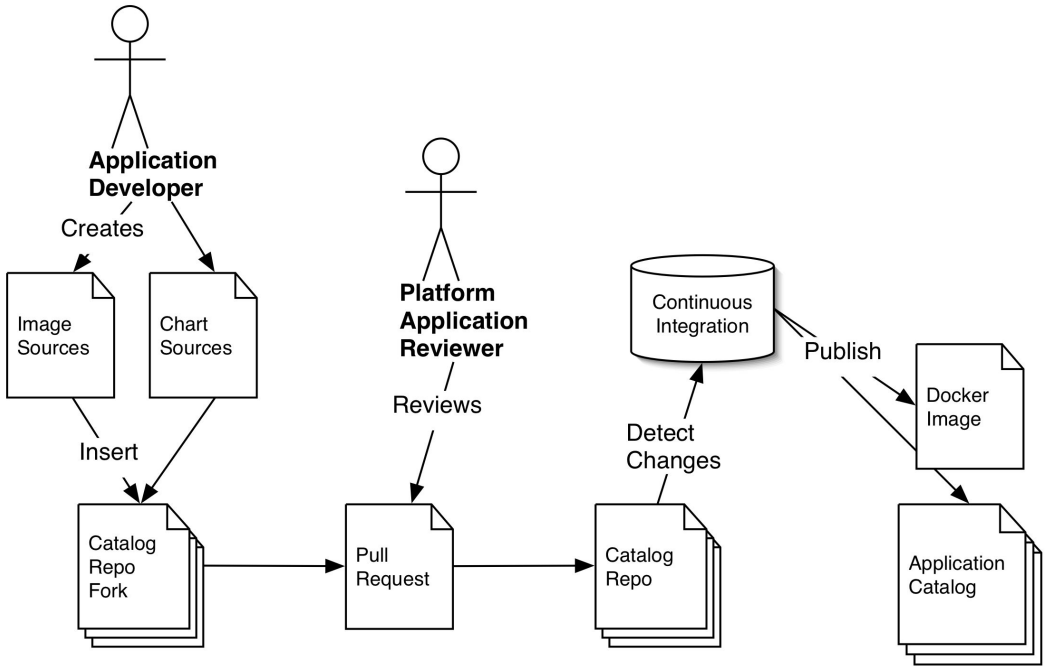
Suitable for hosting multiple edge services for ATLAS (e.g. Squid, XCache)

Currently testing with XCache

Developing for the SLATE platform

Application Security for the Edge

- We have considered the question of meeting sites' cybersecurity policies
- Discussions with community started: http://bit.ly/app-sec-edge
- Feel free to directly comment



Dedicated Development Environments

Pros

- No setup for developers
- Little sysadmin experience required for developers, even for advanced configs
- Consistent for all developers

Cons

- Consistency issues with production (especially without IaC)
- Volatility (single bad deploy brings down environment for all developers)
- Maintenance (OS management, “refreshes”, etc.)
- Security / IAM requirements
- Requires dedicated resources

Local Development Environments

Pros

- Limited volatility (one developer, one environment)
- No maintenance (environments are codified and destructible)
- Developer flexibility (reset environment at will, modify environment as needed)
- Ease of use (less security barriers, no need to push code remotely)

Cons

- Environment variability (high configurability required)
- Limited resources (software must run on local machines)
- Machine clutter (local machine can easily become a dedicated development machine when many dependencies are required)

Best of Both Worlds

- Little setup for developers
- Little sysadmin experience required for developers
- Consistent for all developers
- Limited volatility (one developer per environment)
- No maintenance (environments are codified and destructible)
- Flexible (reset environment at will, modify environment as needed)
- Easy to use (less security barriers, no need to push code remotely)
- Runs wherever developers want to develop

YAML deployments

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  creationTimestamp: null
  generation: 1
  labels:
    app: nginx
    chart: nginx-1.0.0
    instance: default
    release: ms-group-nginx-default
  name: ms-group-nginx-default

...
```

- Very verbose
- Not terribly human readable or writable
- No templating capabilities
- Requires knowledge or extensive documentation reference of Kubernetes object types

Why Helm?

- Kubernetes is complex
  - Application developers write once for users
  - End-users require less deep Kubernetes knowledge
- Environments are different
  - Take advantage of templating for configuration variables
  - Developers need not worry about exact deployment details
- Package management
  - Keep a curated catalog of charts
  - “Push button” deployment and deletion of apps
- This results in improved productivity
  - Improved efficiency for core and application developers

Helm charts

Chart.yaml

```
name: nginx
description: A basic NGINX HTTP server
version: 0.1.0
kubeVersion: ">=1.2.0"
keywords:
  - http
  - nginx
  - www
  - web
home: https://github.com/helm/helm
sources:
  - https://hub.docker.com/_/nginx/
maintainers:
  - name: technosophos
    email: mbutcher@deis.com
```

Values.yaml

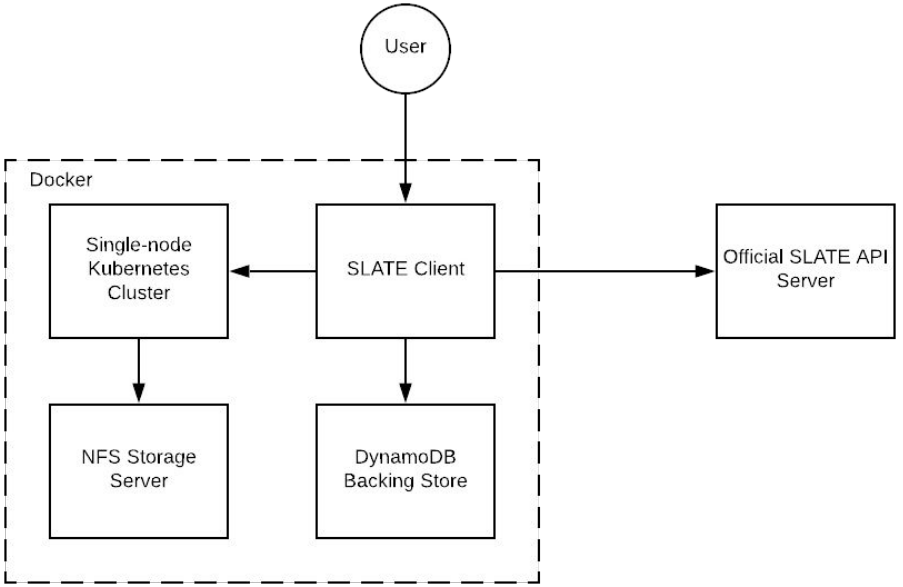
```
replicaCount: 1
restartPolicy: Never

index: >-
  <h1>Hello</h1>
  <p>This is a test</p>

image:
  repository: nginx
  tag: alpine
  pullPolicy: IfNotPresent
```

SLATElite

A lightweight k8s cluster federated with SLATE

- **Create a single-node k8s cluster in Docker at your site and register it with SLATE**

- Enclosed within Docker
  - Little dependency clutter
    - Python, Docker, Docker-Compose
  - Environment consistency
- Deploy in Minutes
  - SLATElite will do the heavy lifting
  - Full SLATE node with tooling
- Try SLATE on your infrastructure
  - Little investment to get started



MiniSLATE / SLATElite Support

MiniSLATE / SLATElite run on:

- Linux with DockerCE
- MacOS with “Docker for Mac”

More Info

MiniSLATE - https://github.com/slateci/minislate

SLATElite - https://github.com/slateci/slatelite