

EESSI : BEHIND THE SCENES

INFRASTRUCTURE

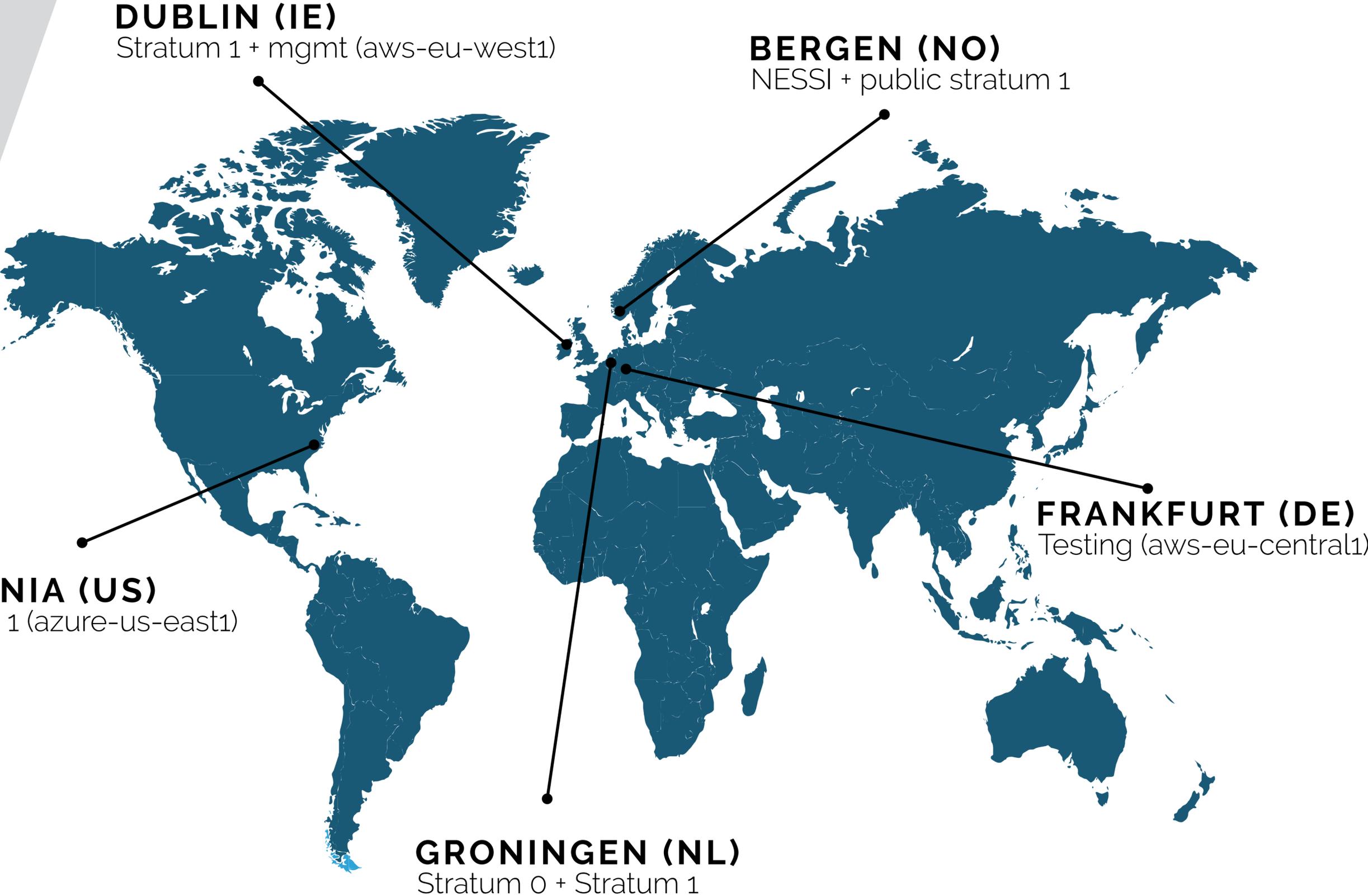


ANYWHERE, ANYTIME

Where are we today?

Multiple providers
Multiple locations

And more to come...



PROVISIONING OVERVIEW

- The chosen tool is Terraform.
- Domain specific language, under (over)active development, rich ecosystem.
- Used to deploy nodes into AWS (and Azure).
- Modularised and reusable.
- Deployment under "four eyes" using Atlantis.
- Conceptually provider agnostic. Conceptually.

```
static > terraform > dns.tf > resource "aws_route53_zone" "testing-eessi-infra-org" > tags > Environn
You, 4 months ago | 1 author (You)
2 resource "aws_route53_zone" "testing-eessi-infra-org" {
3   name = "testing.eessi-infra.org"
4   tags = {}
5   Environment = "testing"
6 }
7
8 resource "aws_route53_record" "testing-eessi-infra-org" {
9   zone_id = var.aws_route53_infra_zoneid
10  name     = "testing.eessi-infra.org"
11  type     = "NS"
12  ttl      = 30
13
14  records = aws_route53_zone.testing-eessi-infra-org.name_servers
15 }
16
17 # Stratum 0
18 You, 12 months ago | 1 author (You)
19 resource "aws_route53_record" "stratum0_rug" {
20   zone_id = var.aws_route53_infra_zoneid
21   name     = "rug-nl.stratum0.cvmfs.eessi-infra.org"
22   type     = "A"
23   ttl      = "300"
24   records = [ "129.125.60.179" ]
25 }
26
27 You, 12 months ago | 1 author (You)
28 resource "aws_route53_record" "stratum0_cname" {
29   zone_id = var.aws_route53_infra_zoneid
30   name     = "cvmfs-s0.eessi-infra.org"
31   type     = "CNAME"
32   ttl      = "5"
33   records = [ aws_route53_record.stratum0_rug.name ]
34 }
35
36 # Stratum 1
37
38 # Run by Rijksuniversiteit Groningen (University of Groningen), Netherlands
39 # Contacts: @bedroge
40 You, 12 months ago | 1 author (You)
41 resource "aws_route53_record" "stratum1_rug" {
42   zone_id = var.aws_route53_infra_zoneid
43   name     = "rug-nl.stratum1.cvmfs.eessi-infra.org"
44   type     = "A"
45   ttl      = "300"
46   records = [ "129.125.55.102" ]
47 }
```

CONFIGURATION OVERVIEW

- The chosen tool is Ansible.
- YAML-based, human "readable", under (over)active development, rich ecosystem.
- Due to repositories being layer-based (a repo for software layer, another for the combat layer, etc), the ansible scripts themselves are distributed across *many* repositories...
- No central playbook deployment or structure.
- (In what repo do I find, or would I expect to find, the ansible scripts to set up a client?)

```
ansible > playbooks > roles > compatibility_layer > tasks > ! main.yml
Bob Dröge, 21 months ago | 4 authors (Bob and others)
1 # Main task which:
2 # -- checks the given path for a Prefix installation, and installs it if necessary
3 # -- starts (and publishes at the end) a CVMFS transaction, if requested;
4 # -- calls the tasks for adding the overlay and installation of sets and package
5 # -- does some fixes and other modifications in the Prefix installation (e.g. se
6 ---
7
8 - name: Check if a Prefix installation is found at the specified location
9   stat:
10     path: "{{ gentoo_prefix_path }}/usr/bin/emerge"
11     register: emerge
12
13 - include_tasks: install_prefix.yml
14   when: not emerge.stat.exists
15
16 - name: Start transaction
17   command: "cvmfs_server transaction {{ cvmfs_repository }}"
18   when: cvmfs_start_transaction
19
20 - block:
21   - include_tasks: prefix_configuration.yml
22
23   - include_tasks: create_host_symlinks.yml
24
25   - include_tasks: add_overlay.yml
26     args:
27       apply:
28         become: False
29         environment:
30           PYTHON_TARGETS: "{{ prefix_python_targets }}"
31
32   - include_tasks: install_packages.yml
33
34   - name: Publish transaction
35     command: "cvmfs_server publish {{ cvmfs_repository }}"
36     when: cvmfs_start_transaction and cvmfs_publish_transaction
37
38   rescue:
39     - name: Abort transaction
40       command: "cvmfs_server abort {{ cvmfs_repository }}"
41       when: cvmfs_start_transaction and cvmfs_abort_transaction_on_failures
42
```

CONFIGURATION INVENTORY AND ROLES

Ansible, inventories and roles.

```
# file: inventory
[clients]
client01
client02
hpc-node[01-50]

[proxies]
eessi-proxy01
eessi-proxy02

[stratum1]
stratum1
```

proxies.yml — ansible-eessi-roles-example

EXPLORER

inventory single-site

README.md single-site

inventory multiple

README.md

proxies.yml

clients.yml

OPEN EDITORS

single-site > inventory

single-site > proxies.yml

```
1 - hosts: proxies
2   roles:
3     - eessi.proxy
4     vars:
5       # List of clients allowed to access your local proxies
6       # Add individual IPs and/or use CIDR notation.
7       local_cvmsf_http_proxies_allowed_clients:
8         - 192.168.0.0/12
9         - 10.0.0.15
10
11       # If you want to use your own Squid configuration template
12       # local_proxies_cvmsf_squid_conf_src: "/path/to/your/
```

clients.yml

single-site > clients.yml

```
You, 5 months ago | 1 author (You)
1 - hosts: clients
2   roles:
3     - eessi.client
4
5   vars:
6     # List of all http proxies that should be configured for
7     # Remove or comment the line if you do not want to use a
8     # case it will be set to "DIRECT" in the client configuration
9     local_cvmsf_http_proxies:
10     - eessi-proxy01:3128
11     - eessi-proxy02:3128
```

> OUTLINE

> TIMELINE

main* 2↓0↑ 0 0 38

You, 5 months ago Ln 1, Col 1 Spaces: 2 UTF-8 LF YAML 4 Spell

CREATING INFRASTRUCTURE

PLANNING DEFINE NEEDS



What infrastructure is needed? Virtual machines? Storage buckets? What access rules are required? Cost/benefit analysis, and so on.

PROVISION TERRAFORM



Produce terraform code, test directly against the test environment, push PR to a branch in the infrastructure repo on GitHub to have Atlantis test the code. Repeat until Atlantis stops complaining.

APPROVE GIT+ATLANTIS



Approving terraform code to have it applied is done by approving a PR in the infrastructure repo. Atlantis then executes the changes.

INVENTORY
GIT



Add to or create a new infrastructure inventory. As of now this is a manual task.

CONFIGURE
ANSIBLE



Run the playbooks to deploy the configuration. Currently performed manually. A long term goal is to have a master deployment node.

VALIDATE
MONITORING

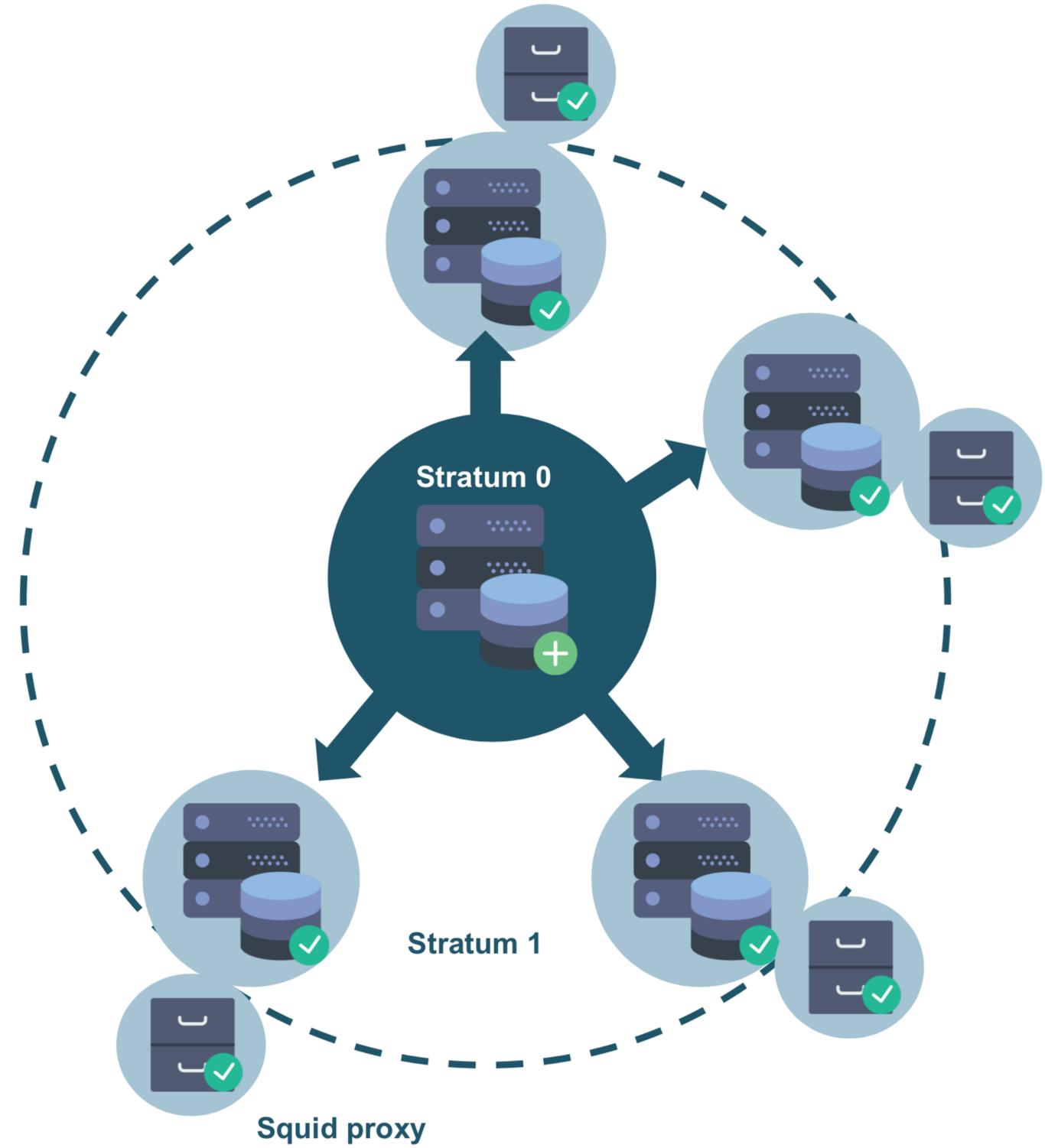


Validate the changes by adding the new services to the monitoring stack.



WHAT'S IN AN EESSI?

A familiar high-level overview

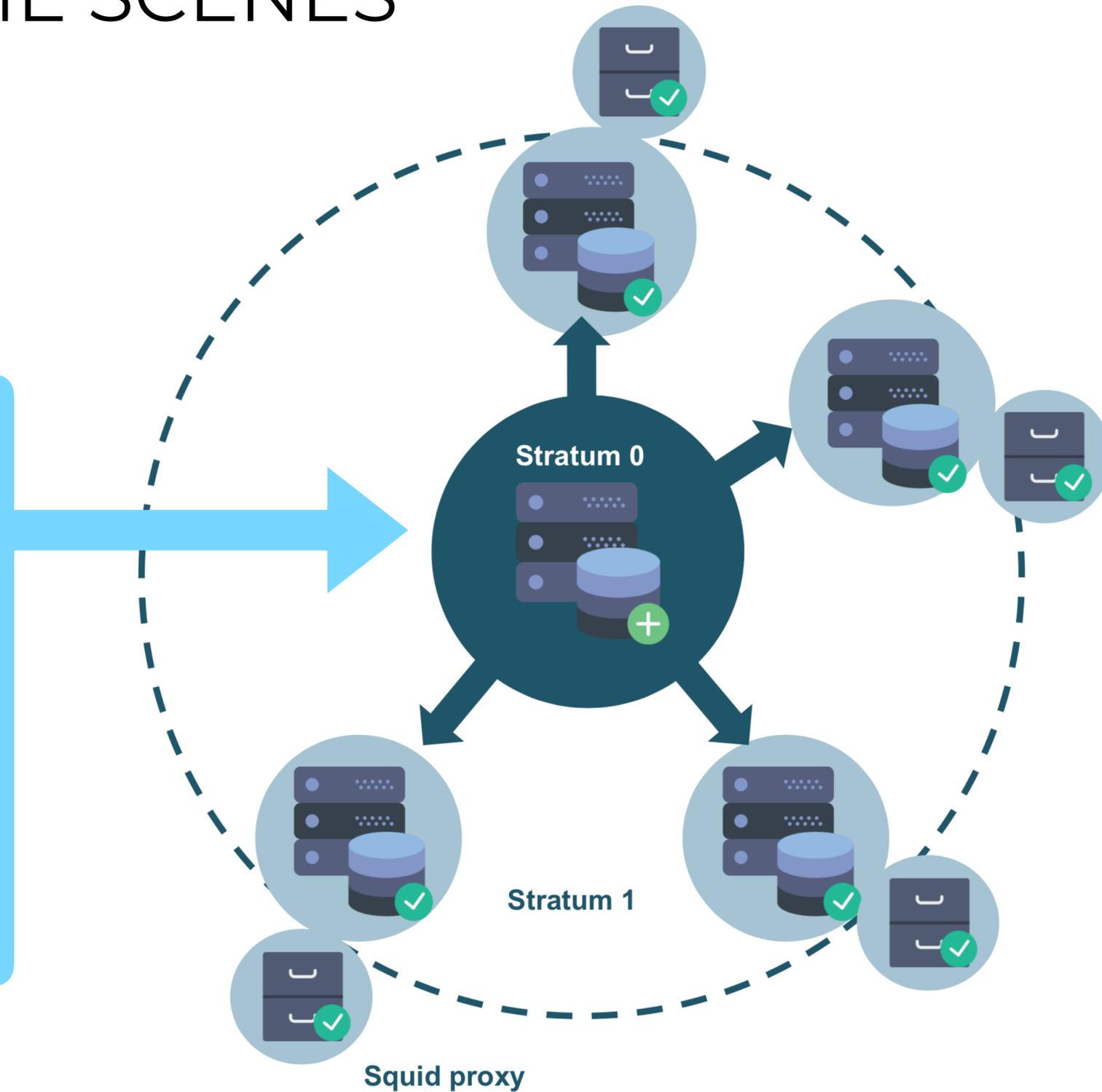


EESSI : BEHIND THE SCENES

STRATUM 0

Stratum 0:

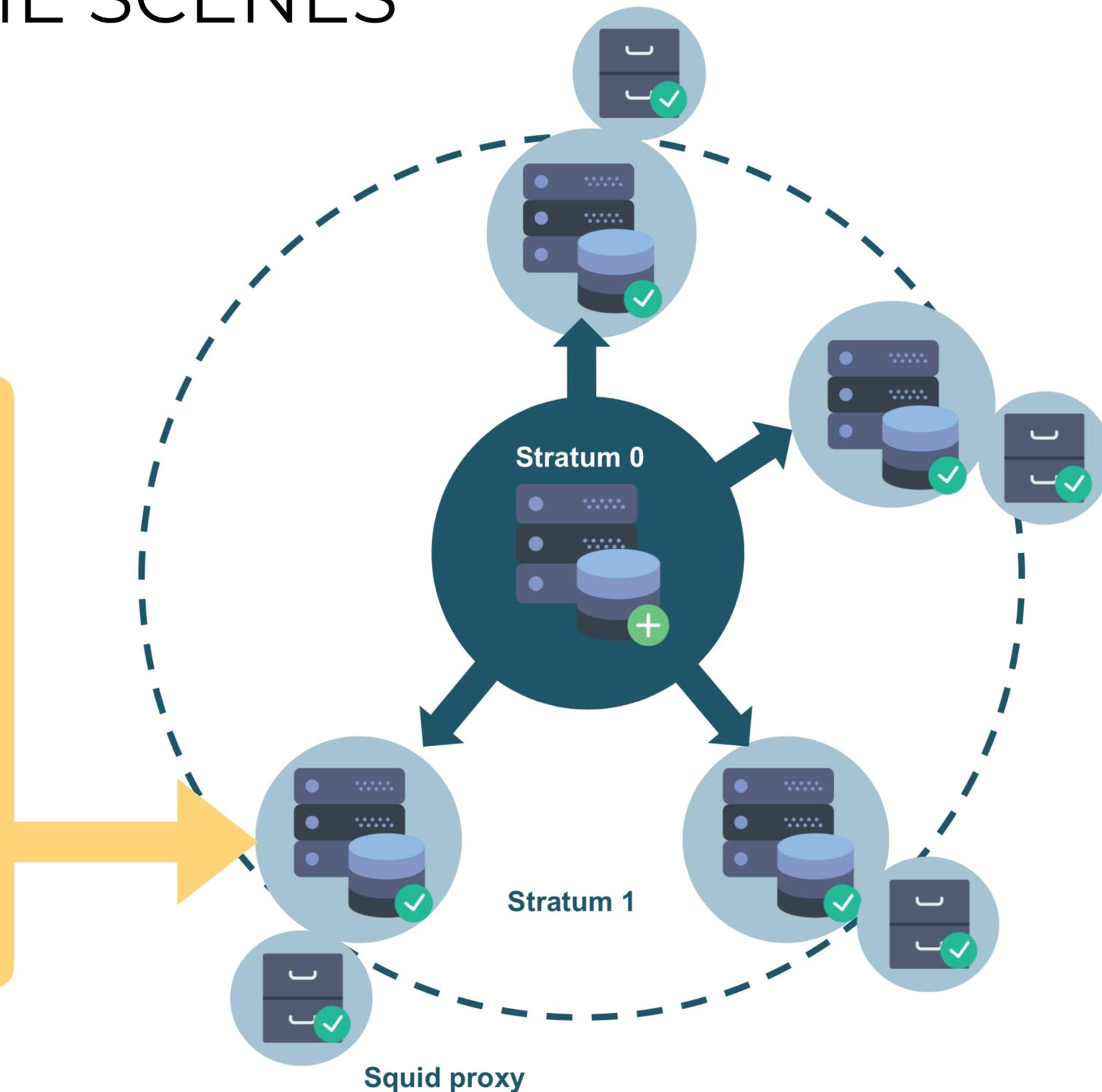
- Central server
- Unique
- Hosts the CVMFS volumes
- Mostly automated day-to-day
- Extremely limited access
- May be provisioned via terraform
- Partial configuration management via Ansible



EESSI : BEHIND THE SCENES

STRATUM 1

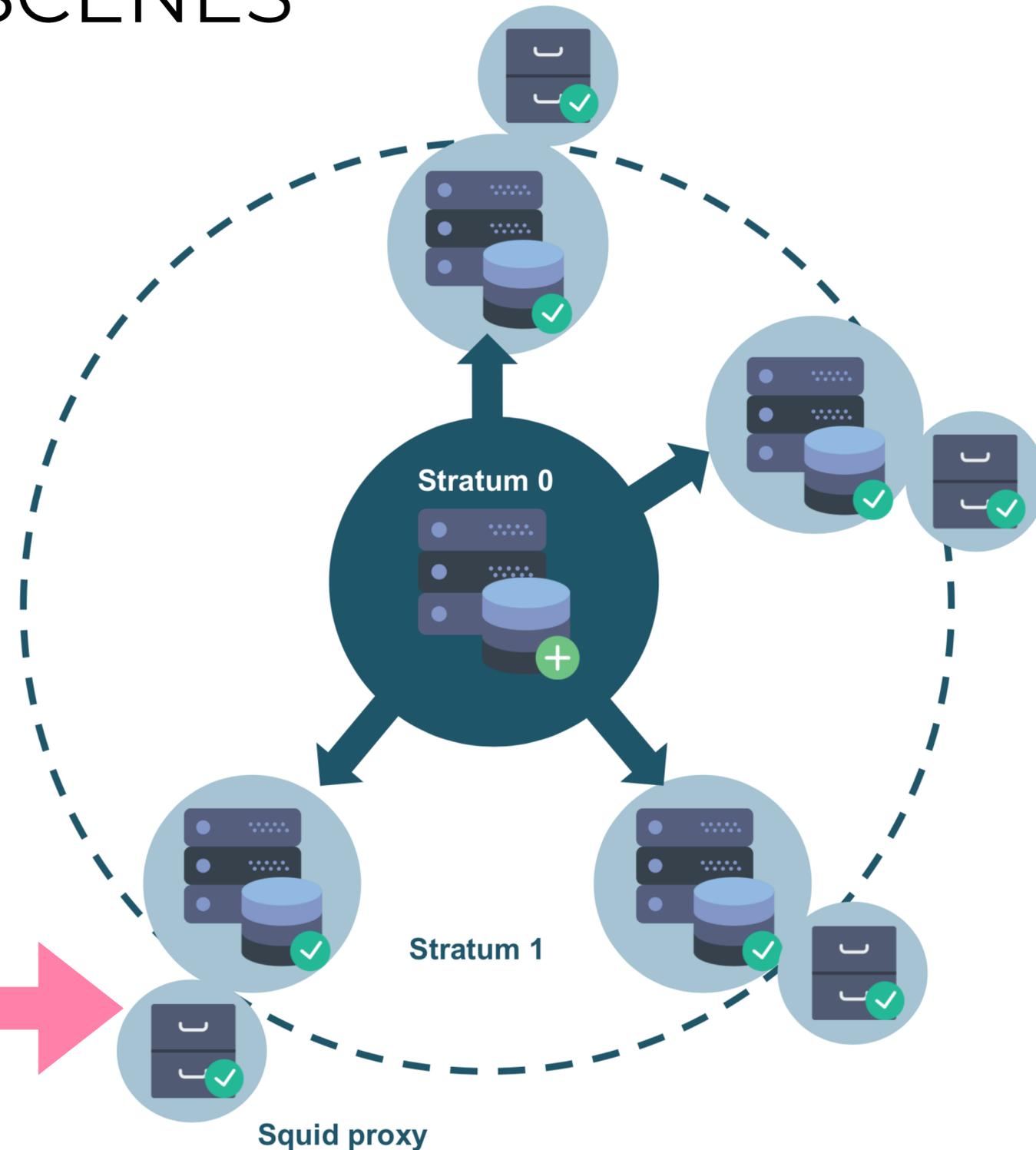
- Replicates stratum 0
- Complete copy of volumes
- Serves data read-only
- A number of these worldwide
- Geographically distributed
- Runs a standard webserver
- Reduces load on stratum 0
- Offers redundancy
- Provisioned via Terraform
- Configured via Ansible



EESSI : BEHIND THE SCENES

PROXY

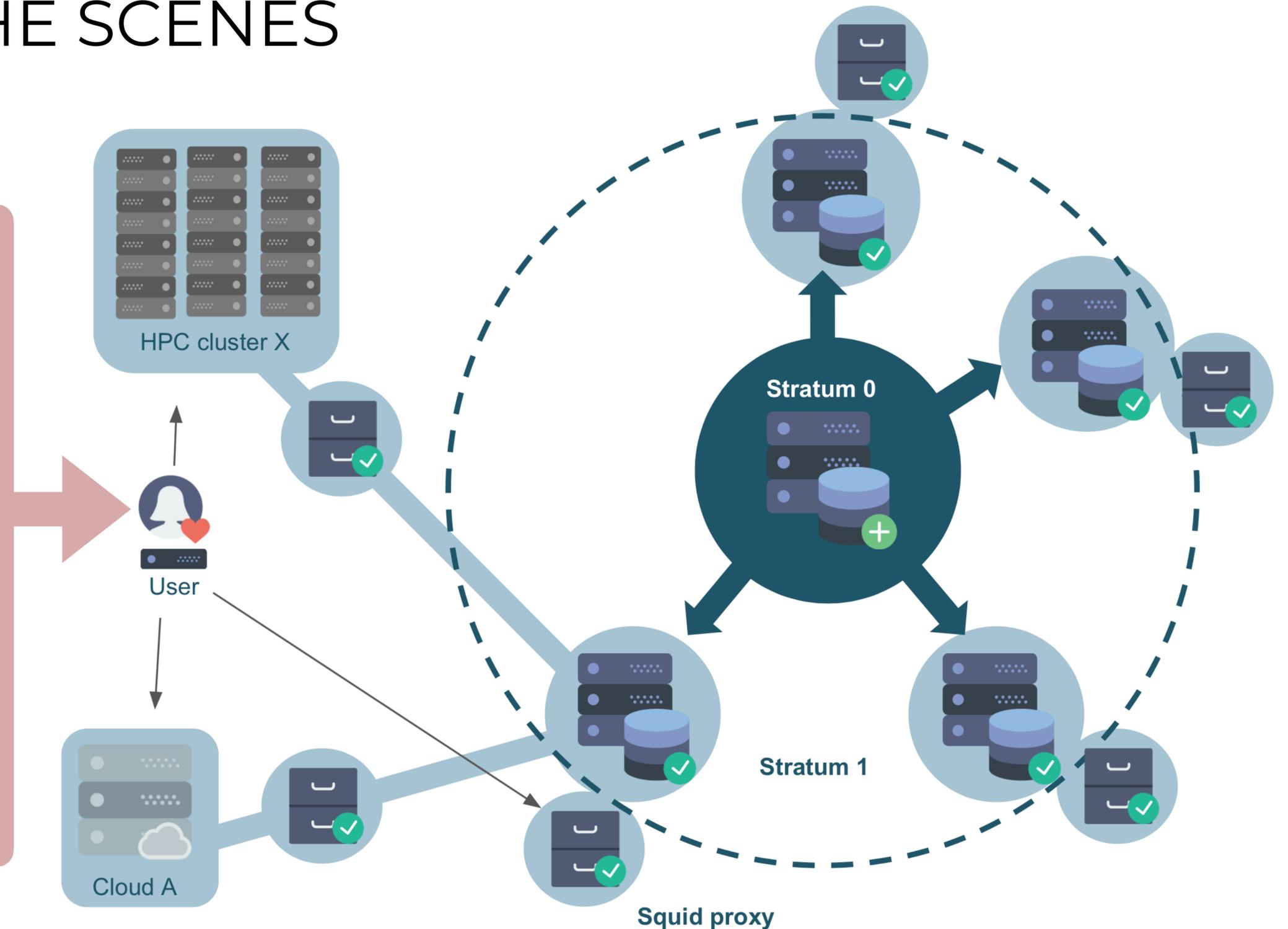
- Reverse proxy for stratum 1 
- I/O cache for clients
- Improved user experience
- Load balancing is an option
- Primary contact point for clients
- Lots and lots everywhere
- Provisioned via Terraform
- Configured via Ansible



EESSI : BEHIND THE SCENES

CLIENTS

- Fetches software from a squid proxy or a stratum 1
- Laptops, workstations, HPC-clusters, cloud machines, etc
- Private, personal, or managed devices
- A local filesystem cache provides performance
- All clients experience the same EESSI software stack everywhere!
- Can be configured via Ansible



EESSI : BEHIND THE SCENES

OTHER STUFF

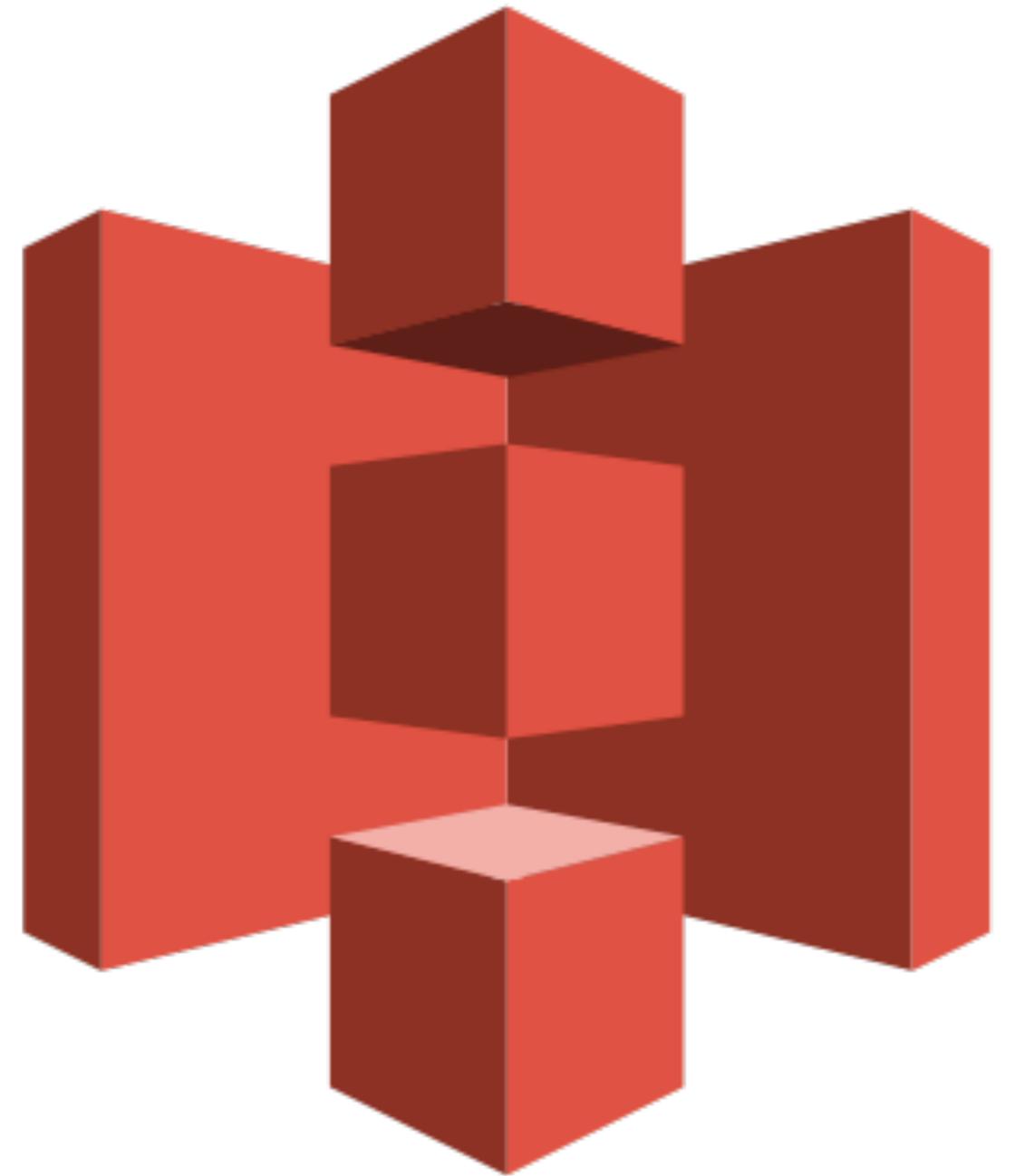
- Monitoring (prometheus + grafana)
- Atlantis (deployment)
- Login node with local persistent storage
- S3 buckets (compatibility layer + software layer tarballs, interaction with GitHub tools, logging, and more)
- Status page
- Ephemeral nodes

- Identity providers
- Access control (hosts, services, networks, users, roles, groups...)
- ...

OTHER STUFF

S3 STORAGE

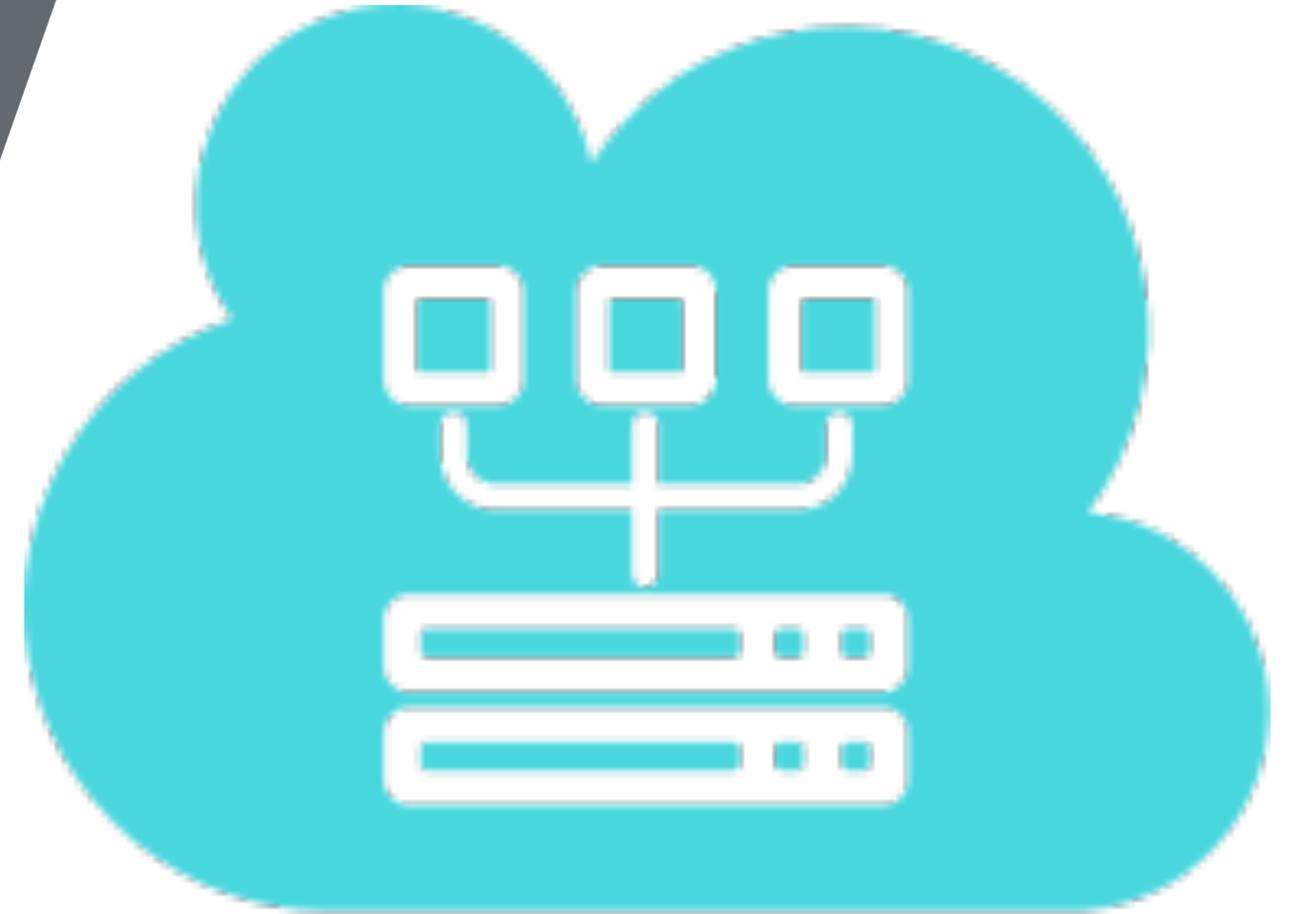
- Provisioned and access controlled via terraform
- Used for staging tarballs, gentoo snapshot backups, and logging
- Currently uses AWS as its provider
- Lots of lifecycle management baked into the terraform modularisation
- The ACL environment is complex, allowing users or specific nodes or specific tokened nodes access.



OTHER STUFF

CITC

- Cluster in the Cloud
- Build/test/deploy EESSI-related “stuff”
- Good for hackathons
- Automatic scale-out (spins up worker nodes on demand)
- Off the shelf software, customisations



**CLUSTER IN
THE CLOUD**



OTHER STUFF STATUS PAGE

- URL: <http://status.eessi-infra.org>
GitHub: <https://github.com/EESSI/status-page>
- Uses a scraper to test CVMFS server:
<https://github.com/EESSI/cvmfs-server-scraper>
- Reports on stratum servers, their repos, and repo sync status (version equality between nodes)
- Open source and generic for CVMFS servers
- Automated provisioning, manual configuration (Ansible role very doable)
- Runs as a cron job every two minutes

Normal service

EESSI services operating without issues.

Warning

Failed

Maintenance

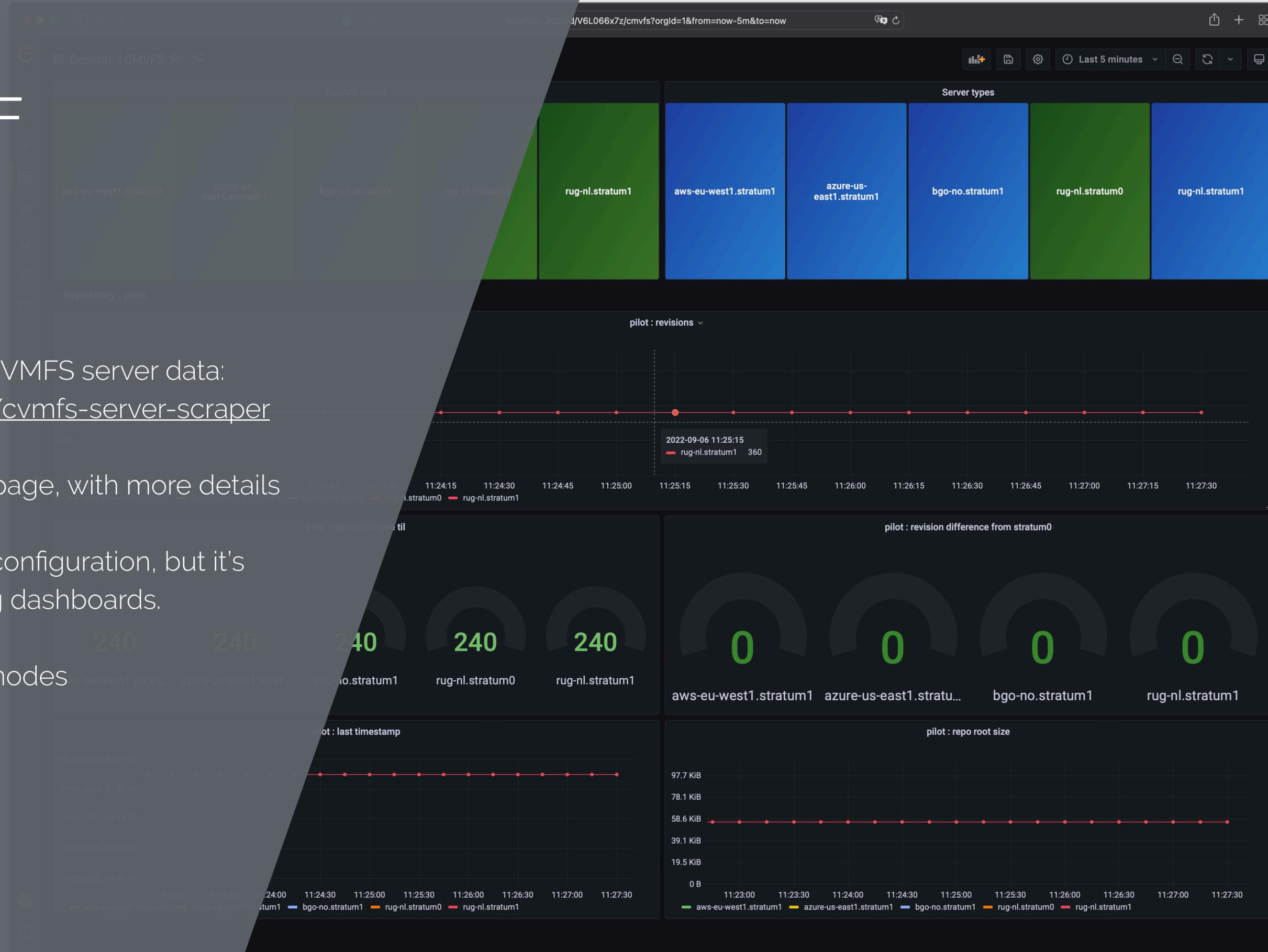


Repositories



OTHER STUFF MONITORING

- Prometheus + Grafana
- Uses a scraper to collect CVMFS server data: <https://github.com/EESSI/cvmfs-server-scraper>
- Historic view of the status page, with more details
- Veeeeeeeeeeeery manual configuration, but it's actually solvable, including dashboards.
- Only monitors production nodes



THE FUTURE

PLANS



EESSI : BEHIND THE SCENES

CURRENT PLANS : STRATUM SERVERS

- New physical stratum0
- Use security devices (yubi-key) to approve signing keys rotated regularly
- Admin access to all Stratum servers based on GitHub team membership (except maybe stratum0)
- Locked down default OS image and access control for all stratum servers
- Reconsider image creation (Packer) and instead configure very basic images
- World-wide public Stratum1 service, utilising AWS, Azure, and possibly other providers
- Automated monitoring of public stratum1 via scrapers

EESSI : BEHIND THE SCENES

CURRENT PLANS : OTHER STUFF

- Migration to eessi.io as our primary domain
- Further work to unify access control across services (github tasks/stratum servers/monitoring/atlantis)
- Dedicated management node with node-based access to relevant infrastructure
- Clean up Ansible playbook structure, ensure regular deployment
- Better CD/CI pipelining for playbooks
- Centralize critical meta data (what are the public stratum1s etc) and use this data *everywhere*
- Try to stabilise the infrastructure code base, possibly a clean slate?

THE FUTURE

CHALLENGES



EESSI : BEHIND THE SCENES

CURRENT CHALLENGES : CODE BASE

- There are many ways of making things work
- Terraform and Ansible have best practices
- Following them is a good thing
- You won't be following them
- But you can try
- Versions change, code needs to change with it
- Nothing rots like infrastructure code

EESSI : BEHIND THE SCENES

CURRENT CHALLENGES : MONITORING

- Monitoring needs further automation based on an authoritative source of public stratum1s
- How do we tag resources for severity?
- When do we alert?
- Why do we alert?
- Who do we alert?

(Also, how do we alert in this day and age? E-mail? Slack? SMS? For what severity? For what security? How do we inform end users?)

EESSI : BEHIND THE SCENES

CURRENT CHALLENGES : TESTING

- Testing environment needs work
- Stratum1 isn't trivial to test (size, time, etc)
- A complete mirror of prod is possible but will carry a financial and logistical cost
- It's never going to be a perfect mirror
- What do we do with login nodes, management resources, log buckets, etc?
- Ideally we monitor the testing environment as if it was prod but report things in different channels
- This is one of the big outstanding bits of the current environment

EESSI : BEHIND THE SCENES

CURRENT CHALLENGES : DESIGN

- *We need* multi-provider redundancy for EESSI infrastructure
- IaaS is nice, but it isn't enough
- Most provisions are automated
- Lots of configuration isn't automated, even if it is scripted
- The solution is to move everything to "GitOps"
- Atlantis (and/or Terraform Cloud? Github actions?) for provisioning
- Github actions, Semaphore or Rundeck for Ansible?
- If we run self-hosted services we need redundancy with load balancers with health checks in front (ELB)



LESSONS

Infrastructure is a living thing.

Infrastructure is never complete.

Automation isn't.

Monitoring is hard. Reporting is harder.

Testing and production are sometimes similar, but never the same.
(And if it is, one has other problems.)