EarthScope Investigation

Overview

This is an addendum to the initial incident report (https://github.com/2i2c-org/incident-reports/blob/main/reports/2025-05-12-earthscope-resource-provisioning-issue.md), detailing some additional investigation and follow-up actions we took to reduce the likelihood of similar issues repeating.

Summary

On May 12 2025 we received a Freshdesk support ticket about user's servers taking >15min to start and their kernels being killed. This was happening during a workshop and was also impacting other instructors trying to use the hub for different purposes.

Upon investigation, we found there was interplay between two factors:

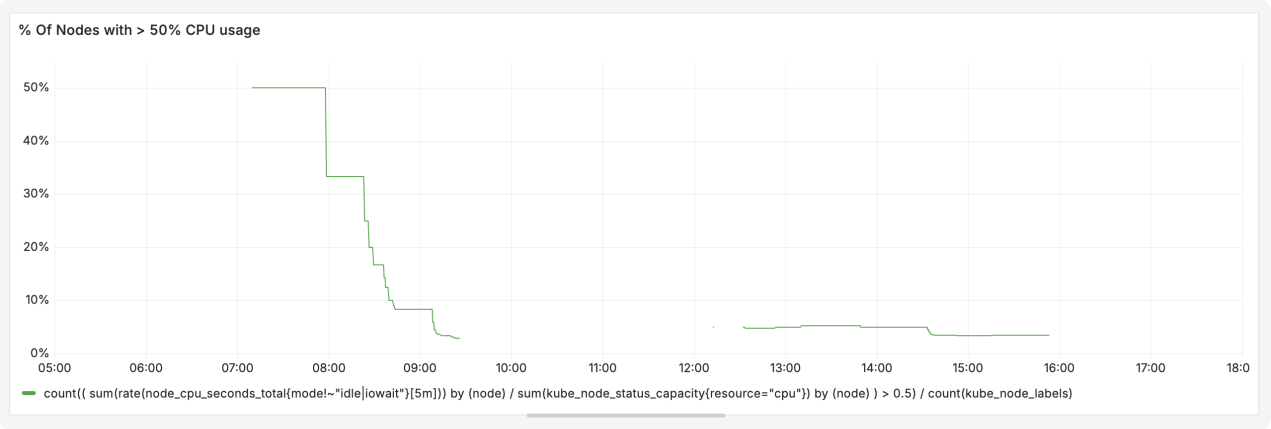
1. Many users coming on at the same time, causing some slow-down in new node spin-ups and user startup (expected)
2. Resource contention for existing users on existing nodes (due to lack of appropriate resource limits set) was causing a splew of unexpected errors *after* the users have started (like blank screens, unstarted kernels, etc)

Further, we also discovered some gaps in our instrumentation that would have helped us diagnose and make better choices earlier on, which are since addressing. We also developed some additional tooling during the course of this investigation that helps us provide more *confidence* that our infrastructure can handle specific workloads.

What Happened

At the time of the incident, we did not have explicit CPU or memory limits set for user servers - we set only guarantees. This meant that when you have a number

of users trying to use all the CPU (or memory) available to them, *some* users (non-deterministically) may not get much CPU (or memory). We believe this is what happened here at the start, and why our team's actions after we were notified resolved the issue for the next days.



This graph (times in PST) illustrates the total number of nodes with a high CPU usage. After our team's intervention, users picked a profile option (https://github.com/2i2c-org/infrastructure/pull/6037) that's more appropriate for their use case (medium), thus spreading themselves out across nodes a lot better. This resulted in less resource contention and thus a better end user experience for everyone.

Resolution

We were already aware this (notice the PR description (https://github.com/2i2c-org/infrastructure/pull/6047) calling the existing setup 'legacy') could cause potential issues, and had developed a new set of resource allocation options that would prevent this. Specifically, we were aware of both user friendliness issues (making servers behave differently from how users expect based on the profile description) as well as resource contention issues (as servers had burstable (https://kubernetes.io/docs/concepts/workloads/pods/pod-qos/#burstable) rather than guaranteed (https://kubernetes.io/docs/concepts/workloads/pods/pod-qos/#guaranteed) quality of service). However, since that changed how options were presented to end users, we were rolling them out slowly, community by community. During the incident itself, we modified the profile options in small ways to enable the workshop to move forward. We co-ordinated with earthscope and moved to the

new resource selection options on July 2 (https://github.com/2i2c-org/infrastructure/pull/6204 (https://github.com/2i2c-org/infrastructure/pull/6204)), thus preventing recurrance of this in the future.

Other hypotheses explored

We explored a number of different hypotheses, and will describe them briefly here. We ruled out all of these, but investigating them helped increase our confidence in what happened + improved our infrastructure's reliability in other ways.

`jupyterhub-home-nfs` migration causing issues

We had moved from using AWS EFS to AWS EBS (with jupyterhub-home-nfs (https://github.com/2i2c-org/jupyterhub-home-nfs/)) for home directory storage, and wanted to rule out that as being a potential issue. We were stymied a little bit by lack of some metrics (since remedied (https://github.com/2i2c-org/infrastructure/pull/6220)).

We attempted to recreate the issue and stress test the home directory setup by:

1. Creating the jupyterhub-simulator (http://github.com/2i2c-org/jupyterhub-simulator) project
2. Simulate a large number of users starting up, and doing a lot of disk iops
3. Manually try to start a server, and see if we can reproduce the reported symptoms while the disk was fully saturated.

Despite multiple attempts, we were not able to replicate the described symptoms. We ruled out this hypothesis, but added a saturation alert (https://github.com/2i2c-org/infrastructure/pull/6209) to watch for this issue in the future. Our experience with this alert in the meantime has increased our conviction that this hypothesis is not what happened.

Realtime collaboration in JupyterLab causing slow startup

JupyterLab's real time collaboration feature relies on a sqlite database to store state, and this causes issues on NFS (https://discourse.jupyter.org/t/jupyterlab-collaboration-configuring-sqliteystore-db-path/34851). We investigated this (as sometimes this may

result in upto 10s delays in startup). We already had workarounds (https://github.com/2i2c-org/infrastructure/issues/2245) in place, but realized that a new version of JupyterLab required a little more workarounds.

We ruled this out because the earthscope image in use did not have the new version of JupyterLab with this functionality! We still added additional mitigation (https://github.com/2i2c-org/infrastructure/pull/6210) so JupyterLab upgrades don't cause mitigation issues.

`cluster-autoscaler` brings up nodes sequentially, not parallelly

We use the cluster-autoscaler (https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler/) upstream kubernetes project to bring up new nodes when users come online. We wanted to check to see if it had weird behavior when a large number of users came on, potentially scaling up new nodes sequentially after prior nodes came up (so node count increases only by 1 at a time, even though more than 1 is needed).

We tested this with `jupyterhub-simulator`, and observed that `cluster-autoscaler` brings up nodes in parallel, rather than sequentially. So this hypotheses was also ruled out.

Systemic Improvements

Ultimately outages don't have any *single* cause, but are systemic failures. As such, we tried to make systemic improvements based on our experience mitigating this particular outage, applied both to earthscope and other communities. Here's a short list of them:

1. Move as many communities as possible out of the 'legacy' setup with no resource limits to the newer, more user friendly resource selection options. This is the most critical improvement.
2. Improve our logging and metrics collection so we can do better post-hoc analysis to understand what went wrong where. This is an ongoing process.
3. Do two quarters of work in increasing our overall systemic resiliency, with a particular focus on alerting and incident response process.

4. Increase our infrastructure engineering capacity (Angus Hollands is now an infrastructure engineer at 2i2c!)
5. Work on systematizing our community engagement process better, so we can understand community needs and address them in more consistent, systematic ways.
6. Create and invest in the jupyterhub-simulator (http://github.com/2i2c-org/jupyterhub-simulator) project so we can slowly but surely simulate *exact* workshop workloads before a workshop, and have more faith in the infrastructure.