Back to blog

Stytch postmortem 2023-02-23

Engineering
Mar 16, 2023
Author: Edwin Lim
Author: Ovadia Harary

2023-02-23

Stytch postmortem

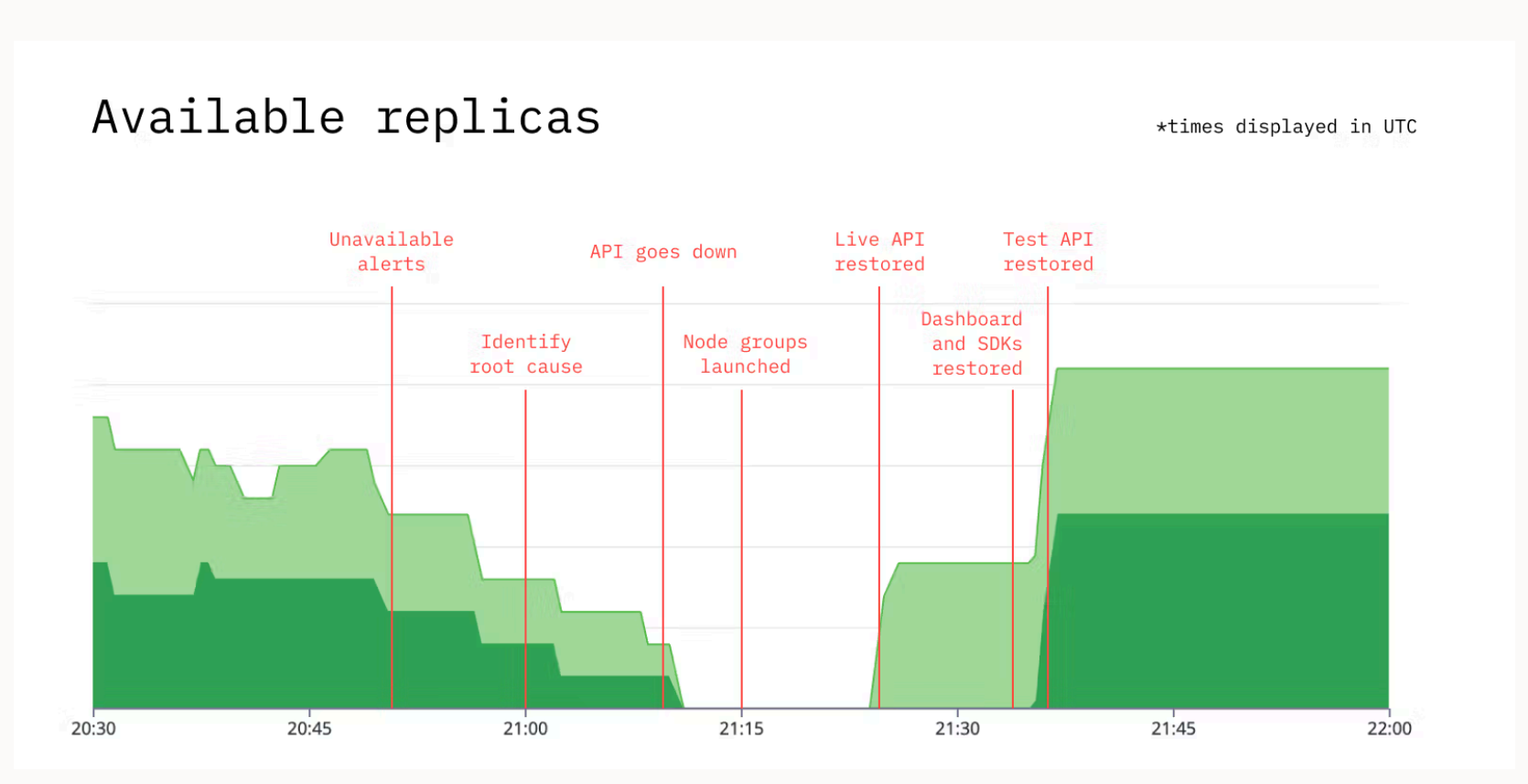
On February 23, 2023, an infrastructure configuration change took down all our Kubernetes worker nodes resulting in a full system outage of the **Live Stytch API for 14 minutes**, the **Stytch Frontend SDKs for 17 minutes**, and the **Stytch Dashboard for 17 minutes**.

We have published our full, internal RCA here.

We're sharing these technical details to give our community an understanding of the root cause, how we addressed it, and what we are doing to prevent similar issues from happening again.

Table of contents

Timeline of outage



Available replicas in the production environment.

All times in the postmortem are in UTC.

- **20:48** – Deployment Replicas Unavailable Alerts begin firing in production.
- **21:00** – Engineers identify node group change as likely root cause.
- **21:10** – Production Live and Test API go down.
- **21:14** – Support team flags outage.
- **21:15** – New node groups are launched.
- **21:17** – Dashboard and SDKs go down.
- **21:24** – Production Live API service is restored.
- **21:34** – Dashboard and SDK services are restored.
- **21:36** – Production Test API service is restored. All systems are operational.

Background

A primer on Stytch infrastructure

Stytch uses AWS EKS, a managed Kubernetes service, to manage our compute infrastructure. AWS EKS offers a few different ways to host nodes that will run the containers – each have their own strengths and unique qualities. Self-managed nodes provide complete control over the instance at the expense of full management of the instance itself. Managed node groups handle the provisioning and lifecycle of Elastic Compute Cloud (EC2) instances through Auto Scaling Groups (ASGs).

AWS Fargate, as a serverless platform, can offload the complete node management to AWS.

Stytch initially used managed node groups for shared resources such as CoreDNS, log forwarding, ingress controllers, etc. and Fargate to run major data plane services such as the Stytch API.

To create these cloud resources (EKS cluster, managed node groups, Fargate profiles, etc.), Stytch uses Crossplane, an open-source tool, which allows us to manage our clusters in code. It also utilizes the Kubernetes control plane to consistently reconcile differences between our clusters and the desired state. Our Crossplane is set up to utilize a "management cluster" which provisions our clusters across environments and accounts.

Migrating from Fargate to EC2 and Karpenter

In H1 2022, we began an effort to migrate from Fargate to EC2 nodes to run our pods and workloads ourselves. Fargate's slow time to spin up a new pod both impeded our internal development process and hampered our agility to deploy in the case of an incident. We had two major milestones in the project:

1. Move to intentionally over-provisioned managed node groups which were statically sized.
2. Use Karpenter to dynamically provision and manage self-managed nodes as our scaling solution for worker nodes to achieve the elastic compute parity that we had with Fargate.

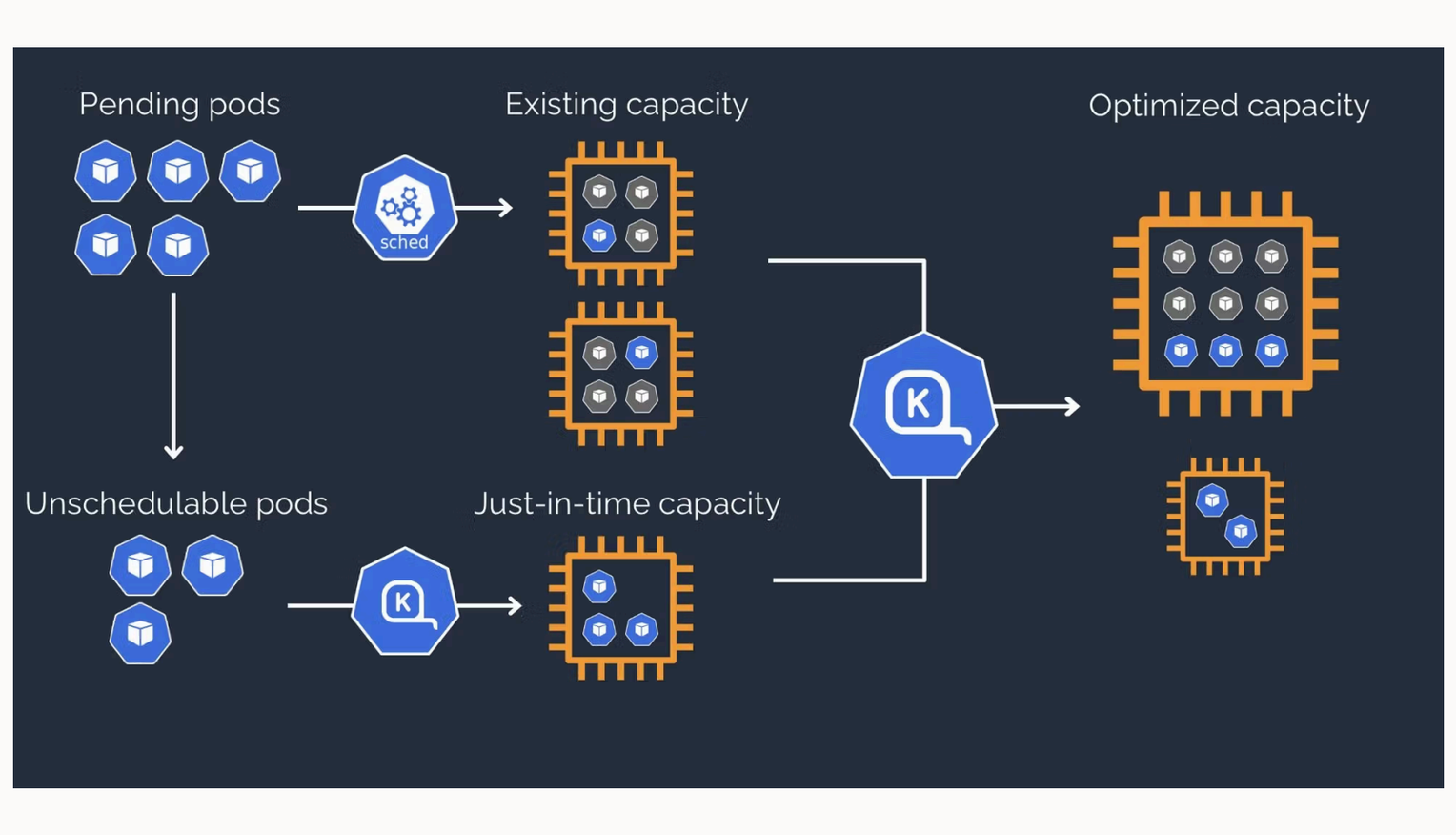


Illustration of how Karpenter operates (source: Karpenter).

Both milestones created positive results for us. Our move to EC2 nodes with managed node groups reduced our complete deploy time from 25+ minutes to under five minutes. Meanwhile, Karpenter's just-in-time provisioning of a dynamic number of nodes proved to be highly available, quick to schedule new pods, and resilient. We not only made our internal development teams much more efficient, but also improved our ability to scale to meet peak customer load and greatly reduced our time to resolution in the event of an incident.

Using managed node groups *and* Karpenter, we had both static compute and dynamic compute. After about a year with this redundancy in place, we felt confident enough in Karpenter's dynamic compute that the managed node groups were no longer needed. We decided to move fully to just-in-time-provisioned nodes by Karpenter and to remove managed node groups for our data plane applications.

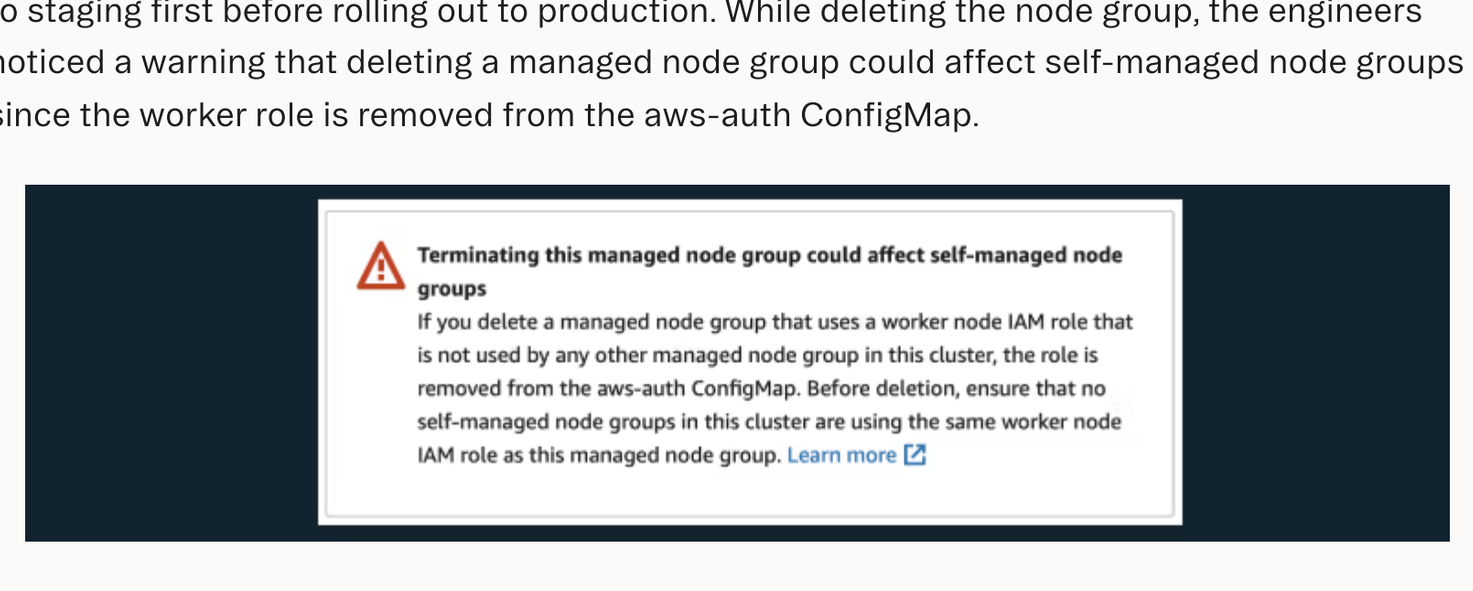
What happened

Root cause: deletion of instance profile (14:21 - 14:50)

14:21 – We landed a change to remove and delete unused managed node groups.

As a safeguard, our Crossplane resources (XRDs) are set up to use an orphan deletion policy – meaning that, if the Crossplane Kubernetes resource is deleted, the resource is not deleted from the provider. Because of our Crossplane orphan deletion policy, this required us to do some operations in the AWS console.

In line with established team norms, two engineers paired on the process to make the changes to staging first before rolling out to production. While deleting the node group, the engineers noticed a warning that deleting a managed node group could affect self-managed node groups since the worker role is removed from the aws-auth ConfigMap.



Warning in AWS console.

Like the rest of our infrastructure, this ConfigMap is managed by Crossplane, and we were confident that even if the role was removed from the config map it would be replaced immediately. After noticing no impact in staging, we applied the same changes in production**.**

However, unknown to our engineers and absent from AWS documentation, in the background, the AWS API was making opaque clean up calls after a node group deletion. Upon inspection of our CloudTrail logs, we were able to confirm that the node group deletion resulted in the deletion of the instance profile.

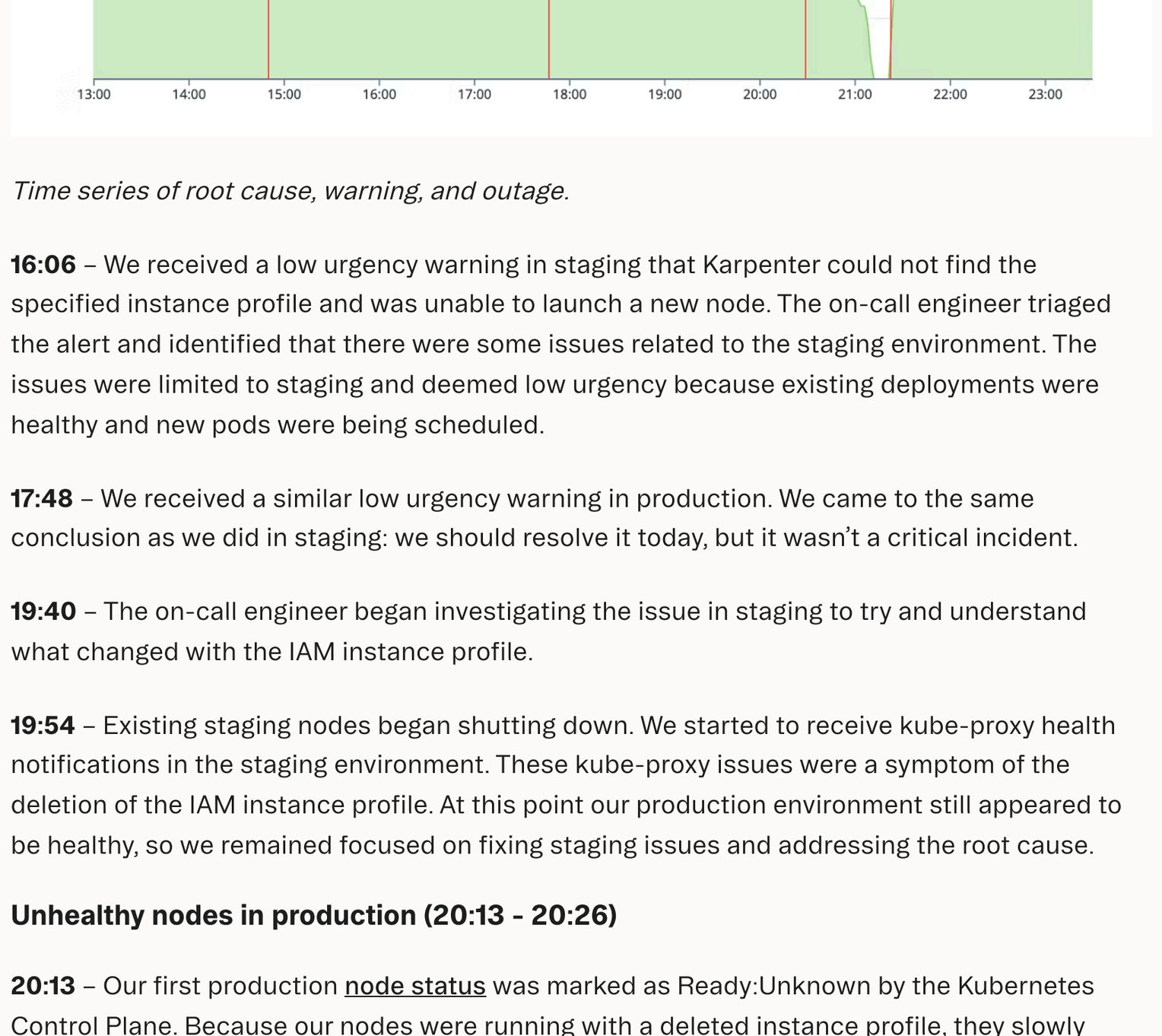
14:24 – The instance profile in staging was deleted.

14:50 – The instance profile in production was deleted.

Managing our resources through Crossplane should have saved us here by automatically recreating the deleted instance profile. But while the role was recreated, our Crossplane Cluster Composition used a randomly generated name instead of the previous instance's profile name which our Karpenter configuration depended on. With these incompatible names, Karpenter did not have the permissions to launch new nodes.

We would encounter issues several hours later once the deleted instance profile needed to be used again – by either new or existing nodes.

First warning and nodes shutting down (16:06 - 19:54)



Time series of root cause, warning, and outage.

16:06 – We received a low urgency warning in staging that Karpenter could not find the specified instance profile and was unable to launch a new node. The on-call engineer triaged the alert and identified that there were some issues related to the staging environment. The issues were limited to staging and deemed low urgency because existing deployments were healthy and new pods were being scheduled.

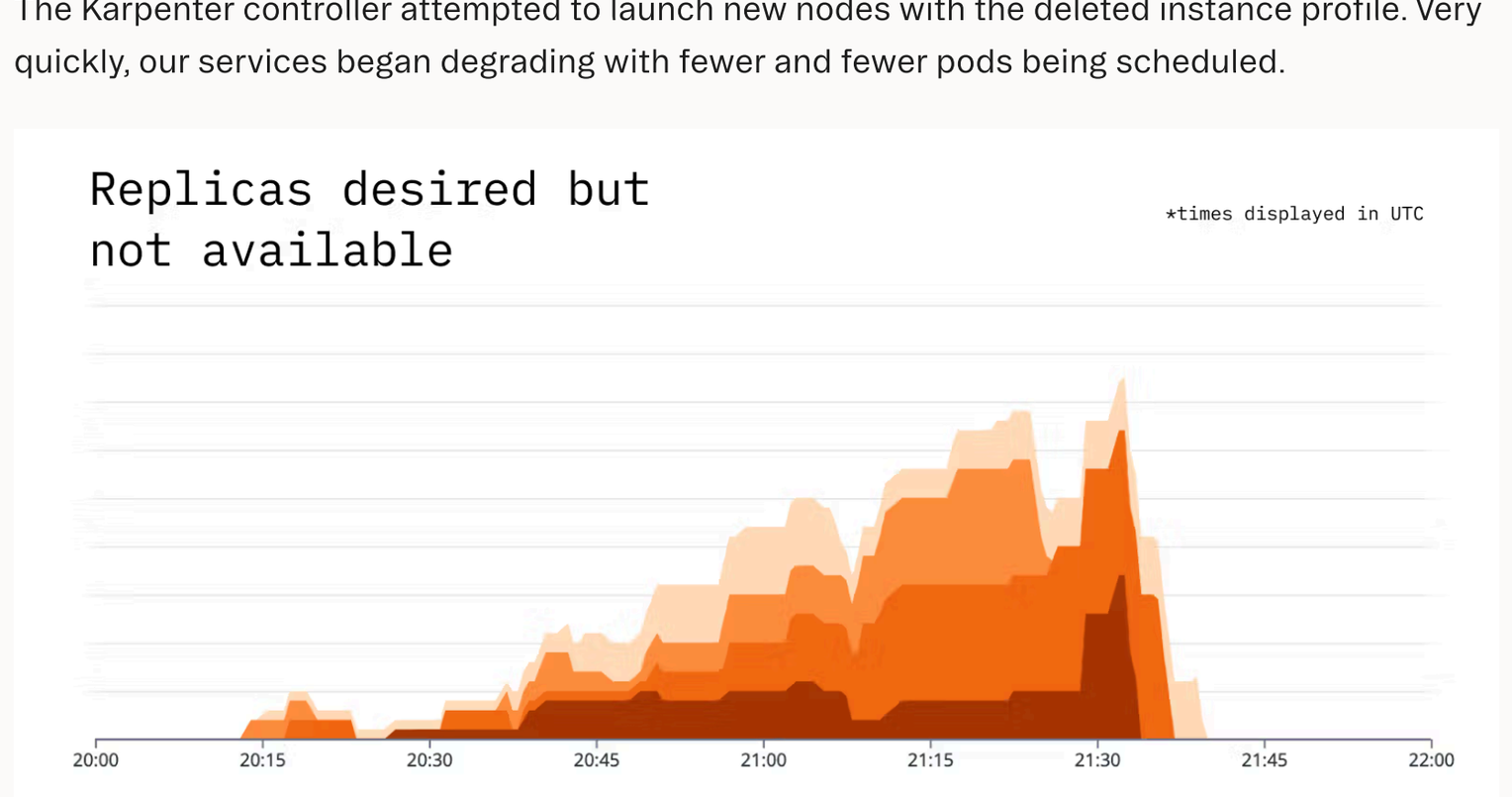
17:48 – We received a similar low urgency warning in production. We came to the same conclusion as we did in staging: we should resolve it today, but it wasn't a critical incident.

19:40 – The on-call engineer began investigating the issue in staging to try and understand what changed with the IAM instance profile.

19:54 – Existing staging nodes began shutting down. We started to receive kube-proxy health notifications in the staging environment. These kube-proxy issues were a symptom of the deletion of the IAM instance profile. At this point our production environment still appeared to be healthy, so we remained focused on fixing staging issues and addressing the root cause.

Unhealthy nodes in production (20:13 - 20:26)

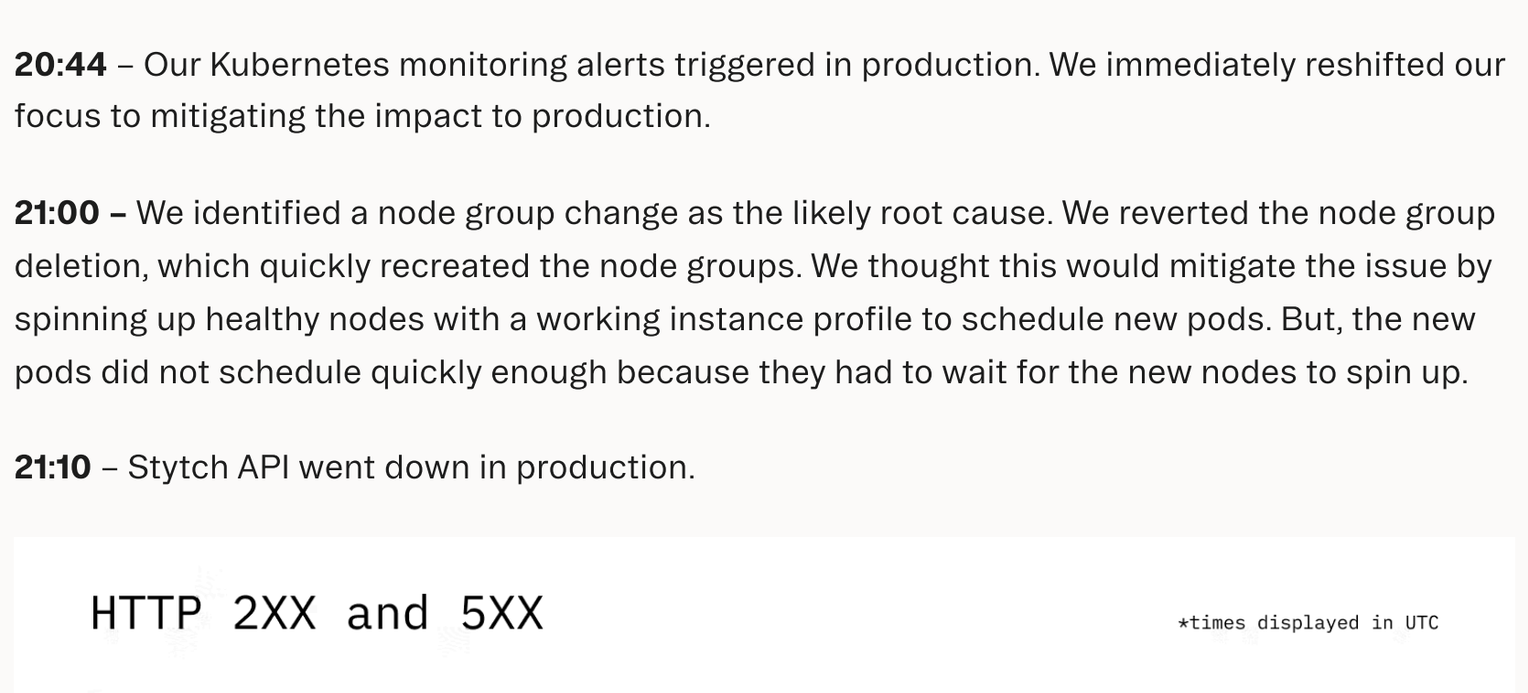
20:13 – Our first production node status was marked as Ready:Unknown by the Kubernetes Control Plane. Because our nodes were running with a deleted instance profile, they slowly began to terminate as they attempted to use the profile.



Rise of unhealthy nodes.

Typically, when a node is in Unknown ready status, and after a five minute wait, the taint controller in the Kubernetes control plane adds an unreachable taint to the node. The taint causes the eviction of all pods running on that node. In our case, our existing nodes were all slowly shut down, then evicted, and we were unable to schedule new pods.

The Karpenter controller attempted to launch new nodes with the deleted instance profile. Very quickly, our services began degrading with fewer and fewer pods being scheduled.



Pods not being scheduled.

20:26 – These node launching failures spiked our Karpenter error rate. We kicked off our incident process and froze deployments.

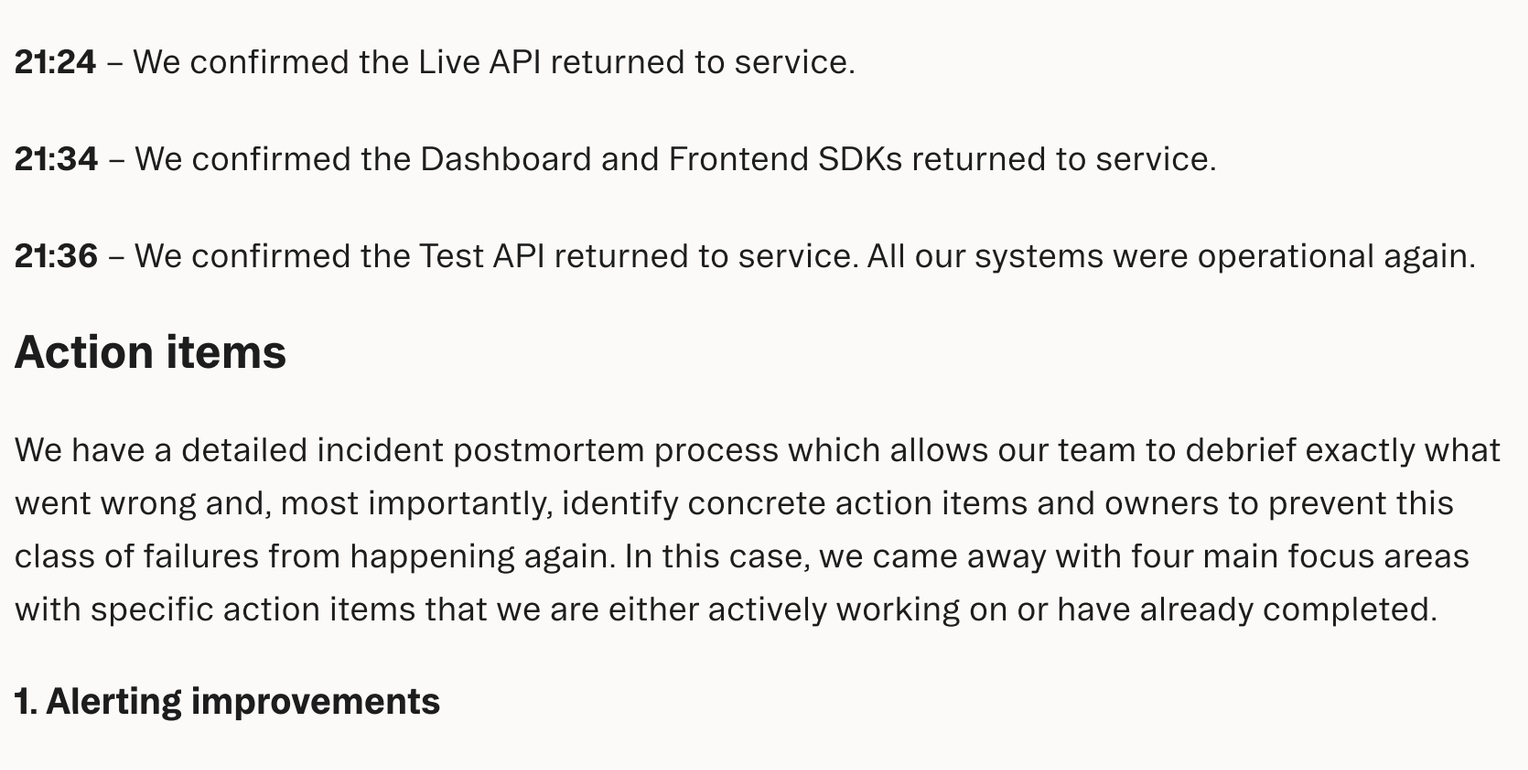
At this point in time, the Stytch API and SDKs were still operational.

Outage and return to service (20:44 - 21:36)

20:44 – Our Kubernetes monitoring alerts triggered in production. We immediately reshifted our focus to mitigating the impact to production.

21:00 – We identified a node group change as the likely root cause. We reverted the node group deletion, which quickly recreated the node groups. We thought this would mitigate the issue by spinning up healthy nodes with a working instance profile to schedule new pods. But, the new pods did not schedule quickly enough because they had to wait for the new nodes to spin up.

21:10 – Stytch API went down in production.



Stytch API calls all returning 5XX.

In order to return to service quickly, we manually scaled up one of the Live API node groups with more nodes that were significantly bigger in compute size and removed taints from the node group. We then removed node selectors from the application consecutively so they could schedule. Within minutes, each application was successfully scheduled with the correct number of nodes.

21:24 – We confirmed the Live API returned to service.

21:34 – We confirmed the Dashboard and Frontend SDKs returned to service.

21:36 – We confirmed the Test API returned to service. All our systems were operational again.

Action items

We have a detailed incident postmortem process which allows our team to debrief exactly what went wrong and, most importantly, identify concrete action items and owners to prevent this class of failures from happening again. In this case, we came away with four main focus areas with specific action items that we are either actively working on or have already completed.

1. Alerting improvements

- Raise the severity of Karpenter errors and treat them as high urgency alerts that go straight to the on-call pager.
- Tune the cluster alerting thresholds to send a high urgency page earlier.
- Add alerts for nodes marked as NotReady or Unknown for long periods to better keep track of the health of existing nodes.
- Raise the thresholds for other deployment-related alerts including unschedulable pods and replicas not available.

2. Cloud configuration and IAC improvements

- Replace the IAM instance profile used by Karpenter to be distinct from the IAM profile used by the node groups.
- Separate out other AWS resources (specifically IAM and security group) that were being overloaded in usage.
- Overhaul our EKS and Karpenter configuration so that new system instance profiles are unique and static per cluster and generated by Karpenter, reducing complexity.
- Introduce a cloud visualization tool to further improve the tools available to our developer team when making infrastructure changes.

3. Status page improvements

- Add live system and product metrics to the status page to provide a real-time view of how our Stytch API is behaving.

4. AWS improvements and learnings

- Reach out to AWS to confirm the undocumented actions and side effects.
- Correct overloaded use of IAM roles and security groups in our infrastructure.

In the wake of the incident, we reached out to AWS in hopes of getting a better understanding of how they handle managed node group deletions. AWS confirmed that the Delete Node Group Action sparked off a set of (undocumented) cascading actions:

1. DeleteNodegroup API invoked > worker nodes are drained
2. Pods running on the target node are evicted from draining nodes
3. Service role cleans up (deletes) any resources associated with the node group (i.e. roles, policies, security groups, etc.)
4. Worker node instances are terminated

Although it's not obvious that the instance profile would get deleted, in retrospect, we should never have used the same IAM role for Karpenter self managed nodes as the node groups because overloading the usage of an IAM role would cause the exact issue we encountered during the incident. During our initial configuration of Karpenter, we missed separating out these roles in the interest of expediency and simplicity.

What made this incident difficult to assess as a whole was the delay between cause and effect. It took on the order of hours until the configuration change started to impact our clusters. We have confirmed with AWS that this is expected behavior and will take this into account when making future IAM instance profile changes.

Conclusion

We are committed to improving our platform and incident process to reduce frequency and duration of incidents, ensure a smoother response and tighter, more detailed communications. In the event of any major disruption of service, Stytch will continue to publish an incident postmortem with context, learnings, and follow ups to share with the community.

Visit and subscribe to Stytch's status page and changelog for future updates.

Share this article

LinkedIn
X
Facebook

Related Articles

Product
Sep 12, 2022

How we (re)made Passwords for next-generation auth

Company
Oct 6, 2022

Stytch named as top authentication provider by Cybernews

Company
Engineering
Dec 2, 2022

Engineering the engineering team

Get started with Stytch

Start building for free
Explore our docs