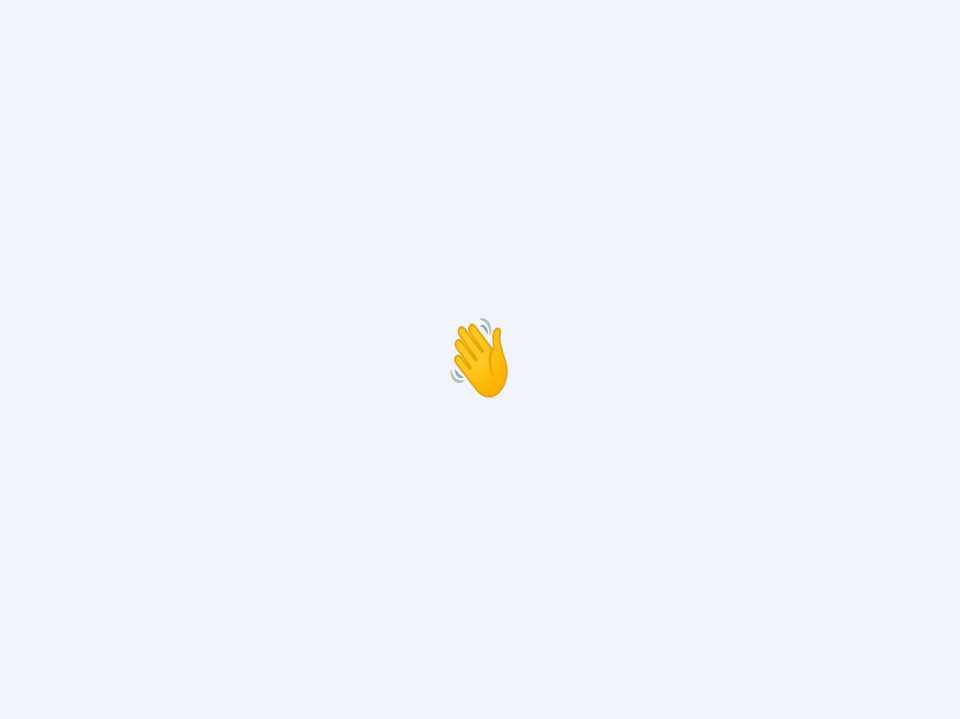
Hello and welcome 👋 Not a pitch for a new discipline or conference - there will be no FREcon, at least not organised by me!

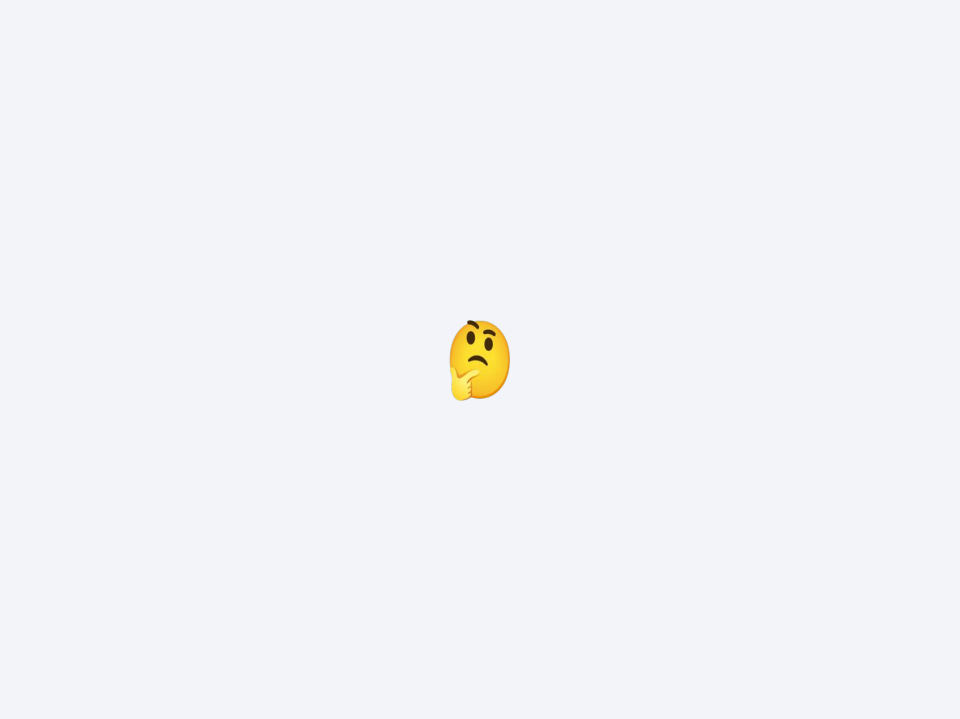
I’m here to talk about some work Shopify has done in the last little while to get a better handle on our cloud infra costs - how we brought them down, and how we’re going to keep a grip on them.



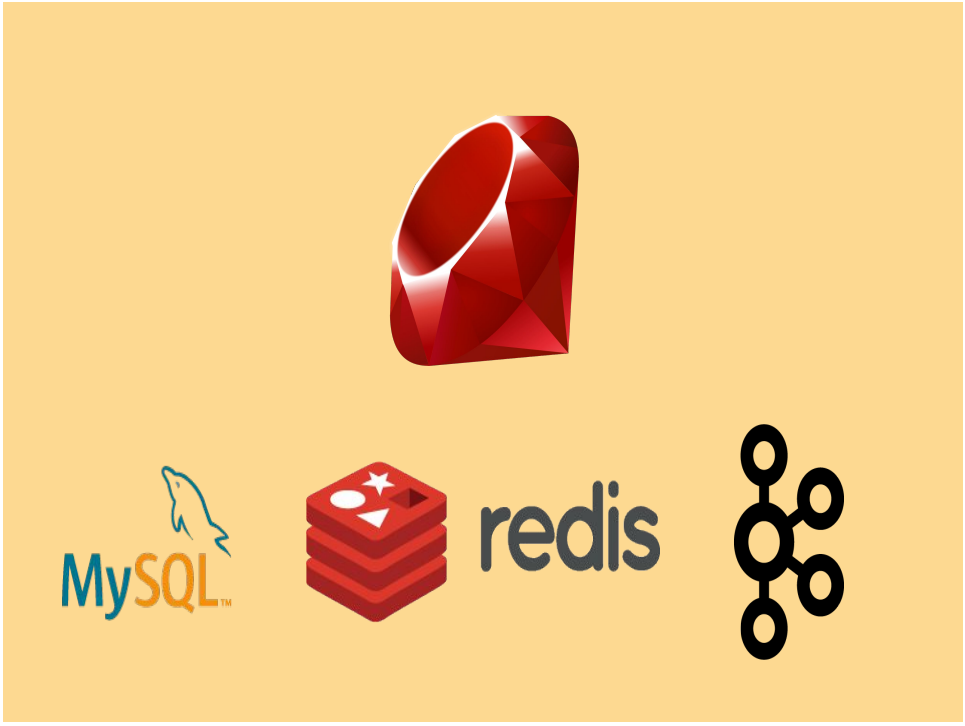
Who am I? Darren Worrall, sysadmin type things for 20 years. Starting with computers under tables, moving through computers in racks, to now herding lots of computers in the cloud, and increasingly, groups of people too.

Working at Shopify for 6+ on Infrastructure / Production Engineering

Shopify is a commerce platform - we build our systems so buyers can browse storefronts, place orders, complete checkouts, and be the backend for the merchants business - fulfilment, marketing, inventory etc

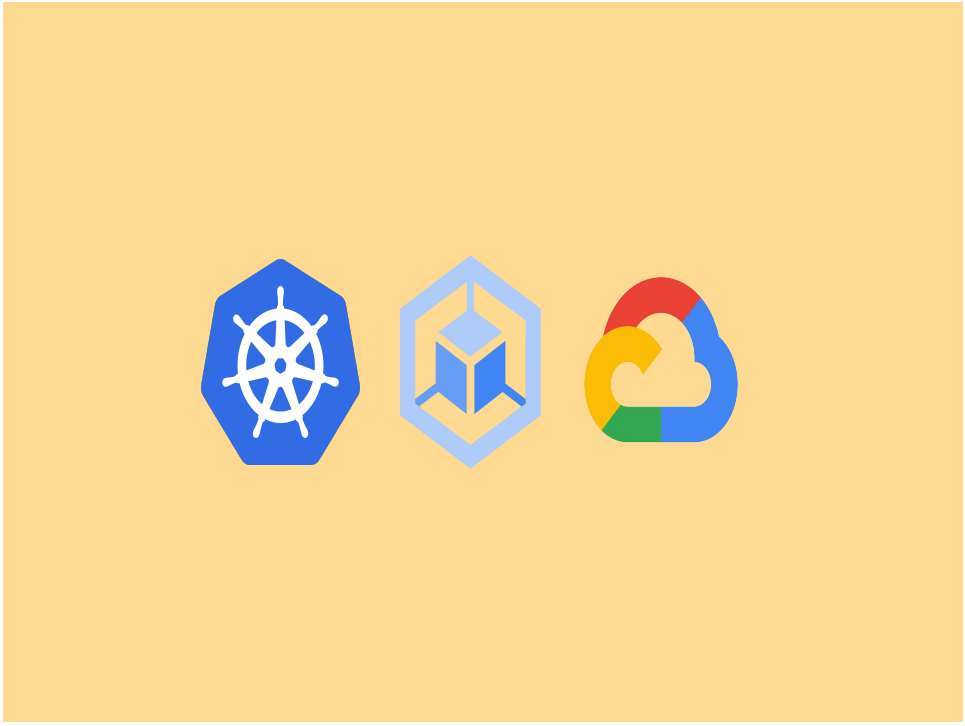


Why? SRE gets involved with the non-functional requirements of our systems - availability, performance, security etc - I think we need to have efficiency on that list - specifically cost efficiency. Like the others, if we don’t get it right - operate sustainably - then eventually we will fail to meet our business objectives.

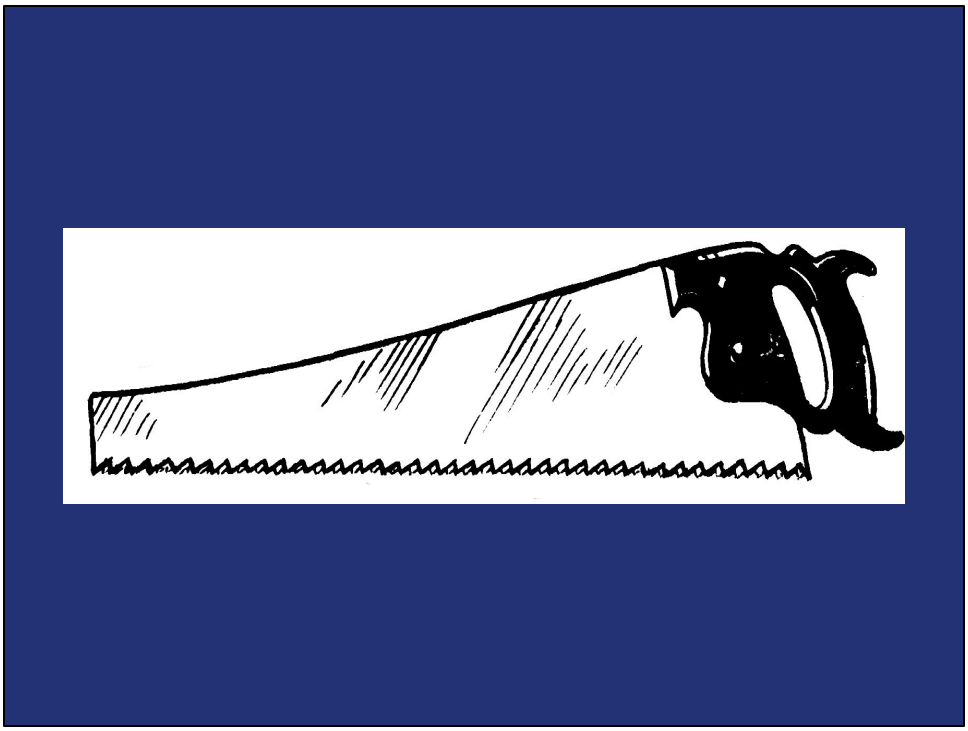


Context about Shopify. We have a Ruby monolith at the core of the app. There are several important infrastructure components to support that Ruby app up and downstream - HTTP routing and load balancing, sharded relational databases, streaming data, caching, proxies for these things etc

And there are _also_ other large applications we consider tier 1 each with their own upstream and downstream dependencies. And beyond that we a have a long tail - hundreds - of smaller services.

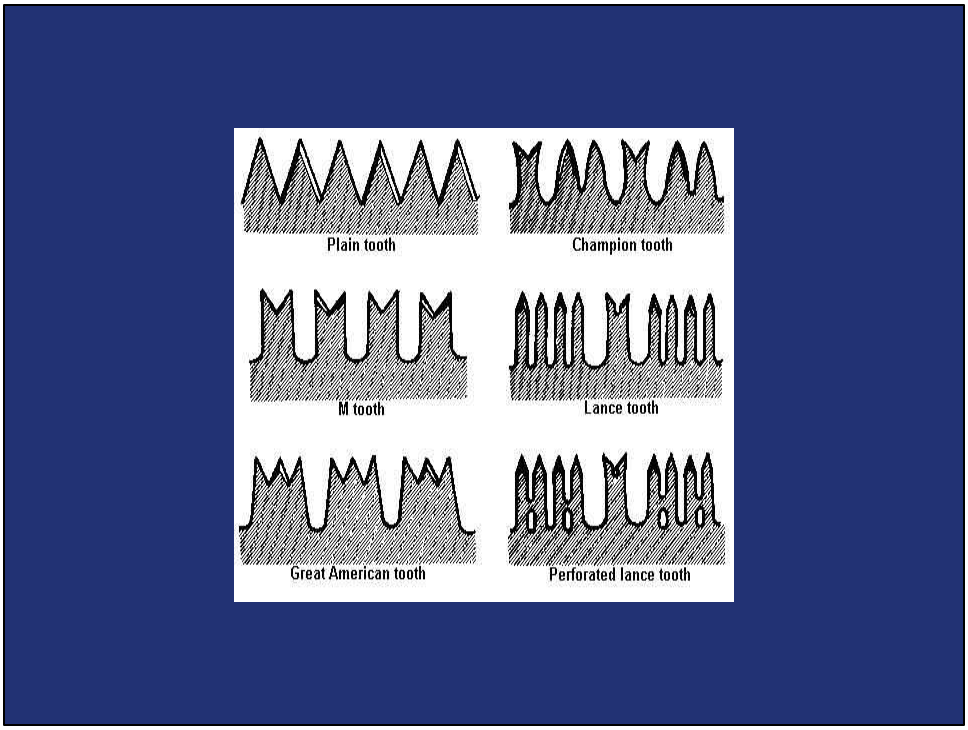


All that runs mostly on Kubernetes on GCP - hundreds of clusters, distributed globally. The detail we will talk about here will apply to that stack, but the principles should apply to any cloud provider or stack. I hope a lot of this will be useful and applicable.

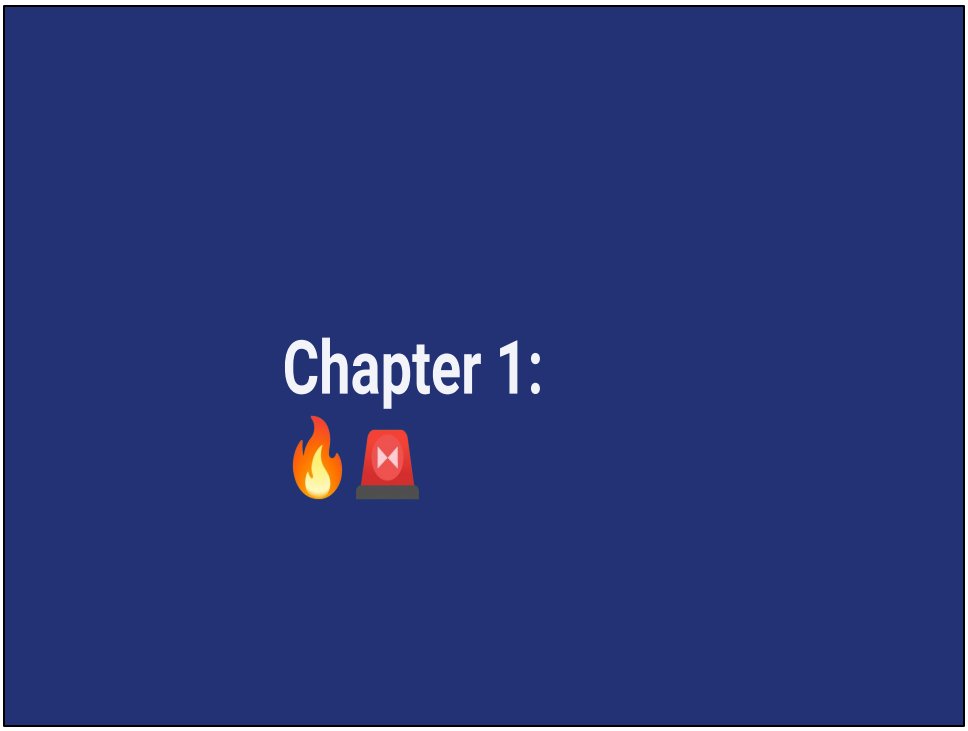


Before this our efficiency efforts were a bit ad-hoc and had varying cycle times. Corey Quinn, cloud economics superstar, has talked about this process looking like a saw tooth - slow rises then sharp declines as attention and effort is spent to interrogate the bill - and that this is to be expected. The cloud is a utility.

Public domain image:
https://commons.wikimedia.org/w/index.php?curid=84873213

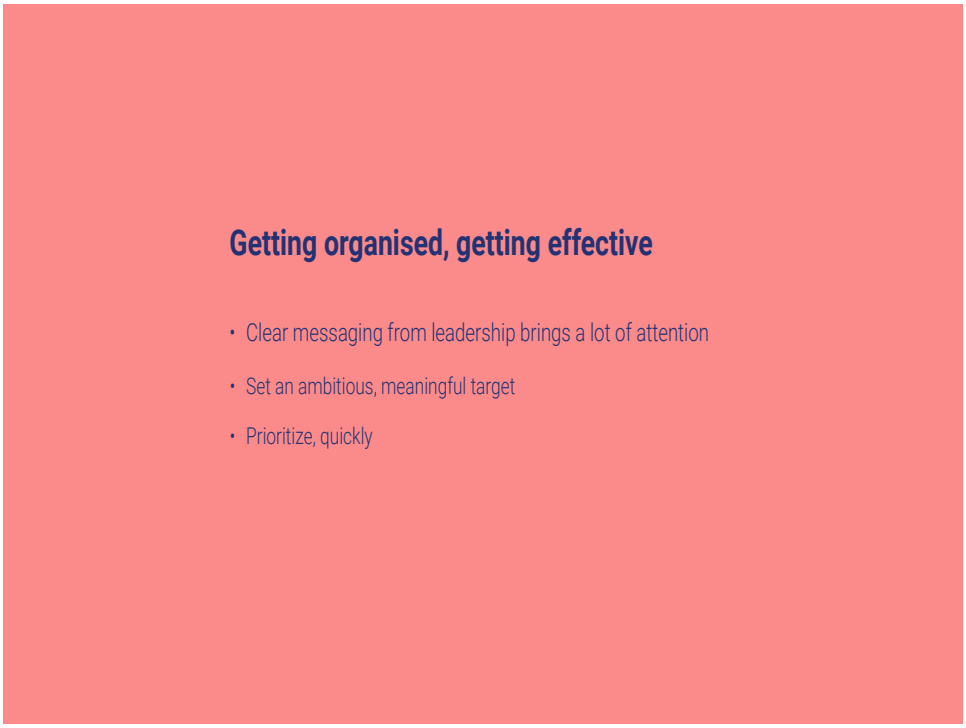


For us, if we're being honest, our efforts were not very clean or consistent. Depending on the business priorities at the time, efficiency had varying priority.



And so our story begins with someone sounding the alarm.

Leadership make clear that cost savings are a priority, a sense that we are not getting good value for everything we're spending. How do we start?



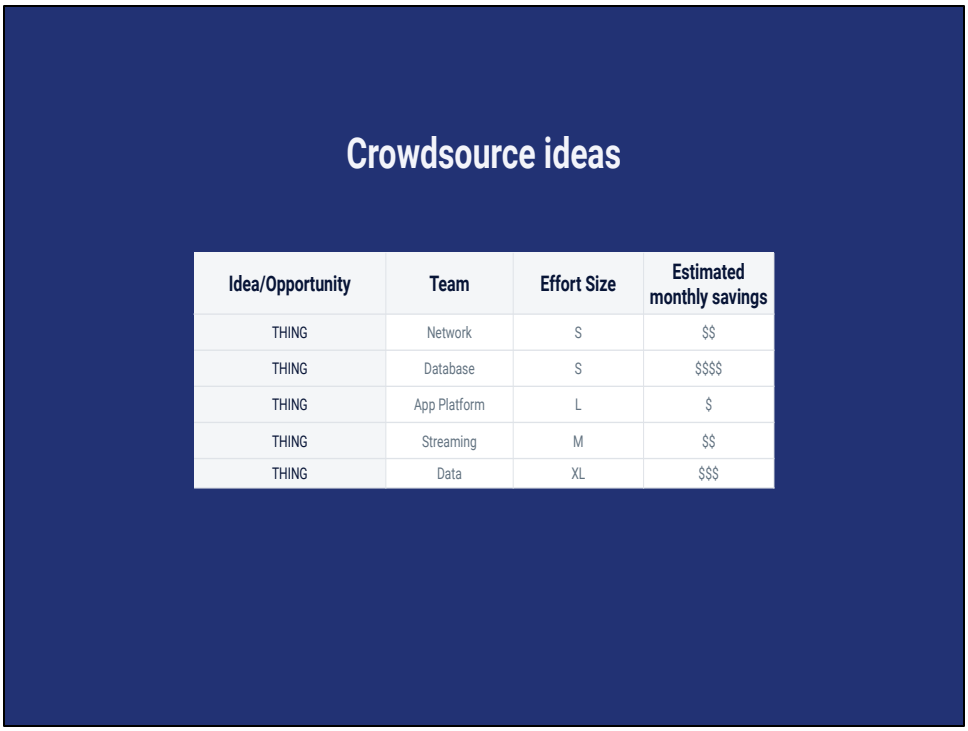
Loud and clear messaging from leadership brings people to the table, this is an opportunity we want to maximise.

We set a target in absolute dollar terms - bring the monthly spend down to between X and Y - a meaningful reduction we could rally an effort around, not 2% - 5%. There was some concern that setting an absolute target could risk people cutting too deep or in the wrong places, but we trusted that we would get pushback if people felt that we were impacting our resiliency boundaries - and we have our SLIs and SLOs to help us see where those are - and for this effort really wanted to set the expectation that we were really looking to move the needle. Setting an unambiguous and amitious goal was important to achieve that.

We know we needed to prioritise the best work, and that we needed to do that quickly. We wanted to understand what work was being done so we could measure its effectiveness - we don’t want teams of people spending days or weeks to save $50/day - but we can’t take long to do this. Capitalize on

momentum and deliver.

In the great traditions of ad-hoc projects, it was time for a spreadsheet



Cast a wide net for ideas, but time box it e.g. a week. Write down everything you are doing now in response to the call from leadership, or could do in the near future with some priority shifting. We quickly got a lot of ideas on this list, teams had a pretty good idea where their low hanging fruit was, and where there was good scope to optimize.

T-Shirt size S(1-2 days)
M(1-2 weeks)
L(2-4 weeks)
XL(1+ month)

And also: estimate the savings. There were some other fields on this sheet, e.g. GCP project, a link to a tracking issue, more notes

So we take a short period to aim, and tell people to… pick the best options from this list and go do the things.

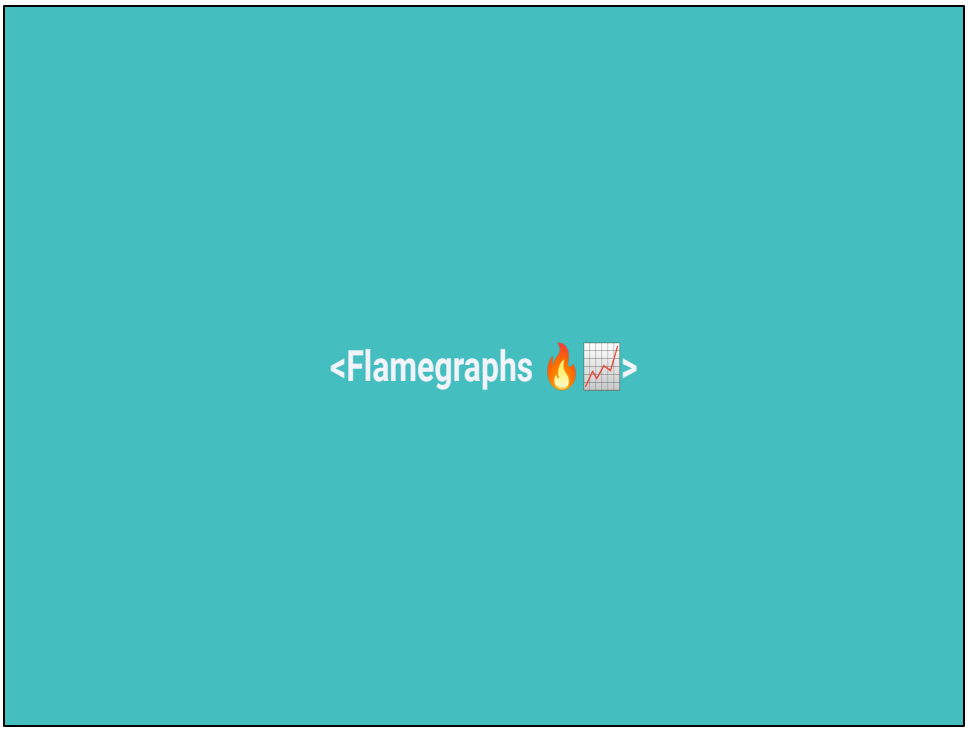
Things started happening immediately - it was actually pretty chaotic, which I'm sure was somewhat intentional. There was a real buzz about the work, turns out It was not hard to get people involved, lots of engineers actually like optimization work. Lorin talked about this yesterday - about the lack of time to reflect - and to some people this was permission/license to go and look at what was done and to try to make it better.

1 week, 2 weeks, 3 weeks go by - the small and medium sized work starts to land. Optimization work was happening in every corner of the infrastructure org, but the impact of that work was inconsistent - some effort was delivering the expected results, but others not. Some of this was expected - when engineers started doing the work, they learned some things and it turned out that they couldn't remove as much resource as they expected at the outset. But sometimes the engineers were surprised too - everything went as they expected, but the savings did not materialize.

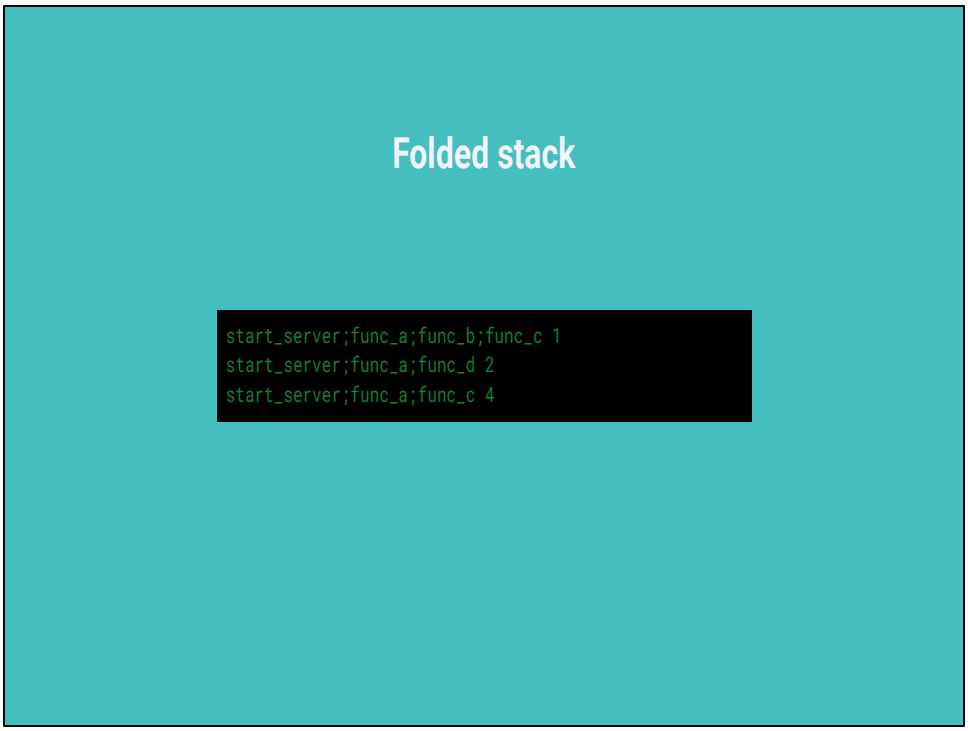
One reason we found for that was that individuals and teams didn’t have enough context to accurately size their opportunities; not the _scale_ of the work - those turned out to be pretty accurate - but the impact on the bill.

Key context teams were missing: what the real costs of things were - we have discounts, CUDs etc - and the size of their workloads relative to the rest of the infra.

We needed better tools to help with this.

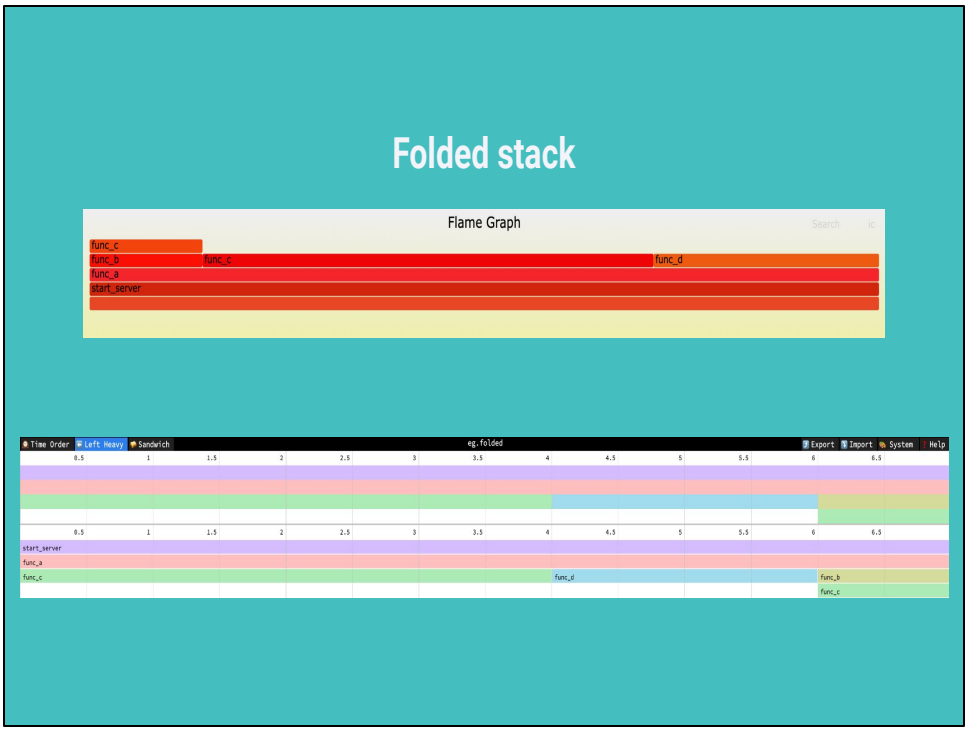


A quick diversion into flamegraphs - let’s define this as a collective noun for performance visualisation tools. Taking hierarchical tracing data and letting us explore and understand them.

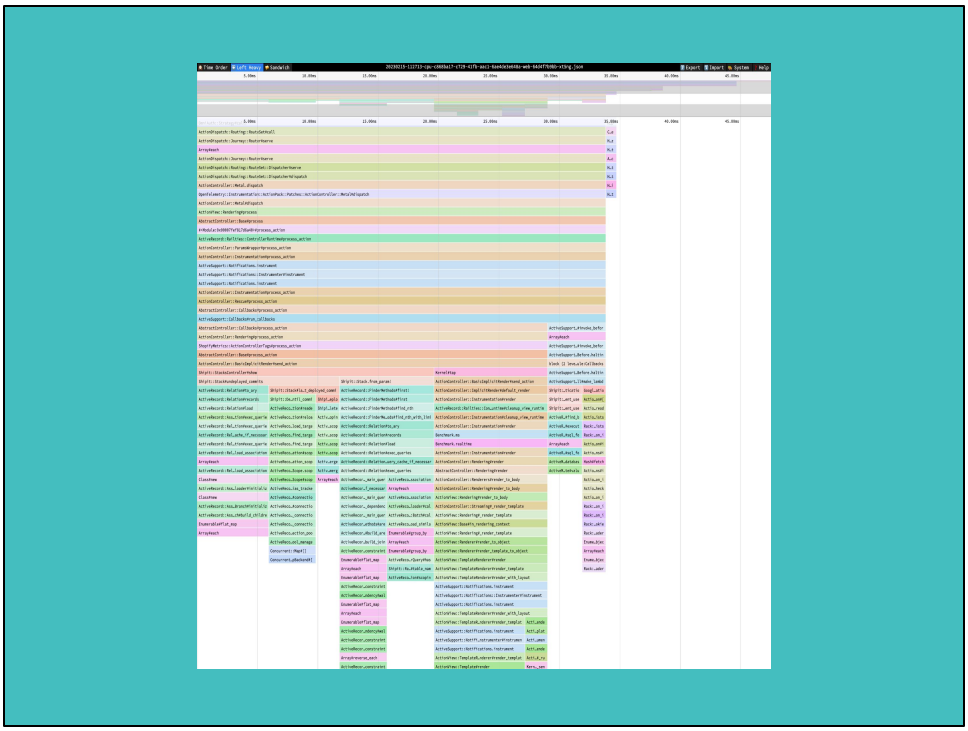


Folded stack is one example of an intermediate representation of trace data - puts stack traces on a single line, with functions separated by semicolons, followed by a space and then a unit - e.g a count.

This intermediate format can be read by a bunch of tools e.g. flamegraph.pl https://github.com/brendangregg/FlameGraph /ht Brendan Gregg

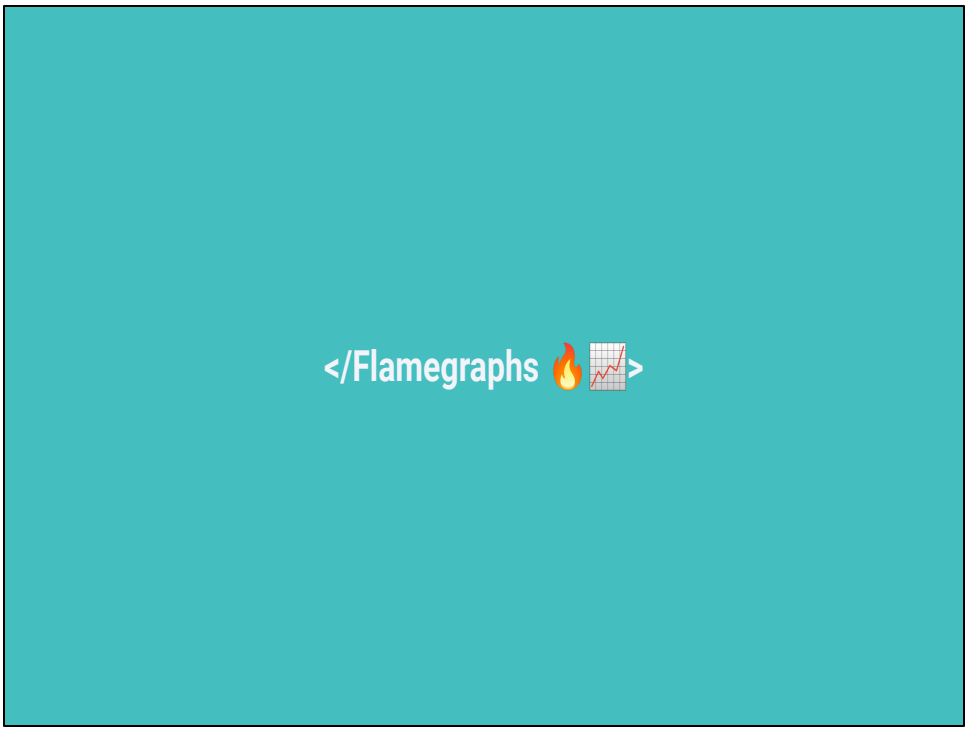


This intermediate format can be read by a bunch of tools e.g. flamegraph.pl https://www.speedscope.app/

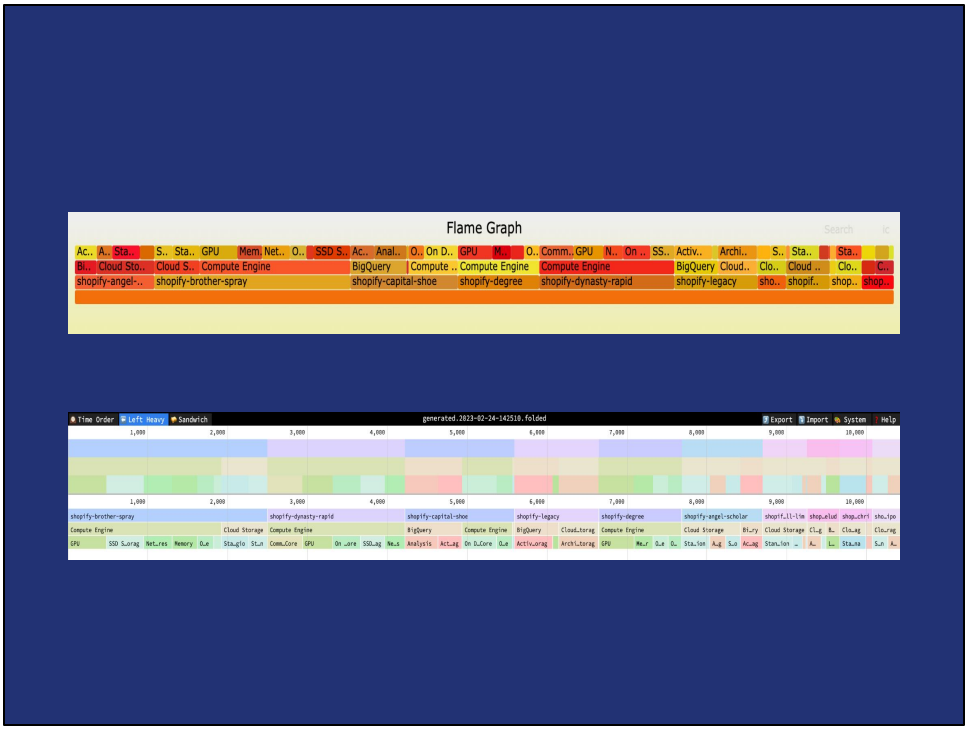


We typically use these tools to understand how our applications are behaving and performing - this is an example request profile from https://github.com/Shopify/shipit-engine

Are we spending time where we expect, or are there things we can optimize.

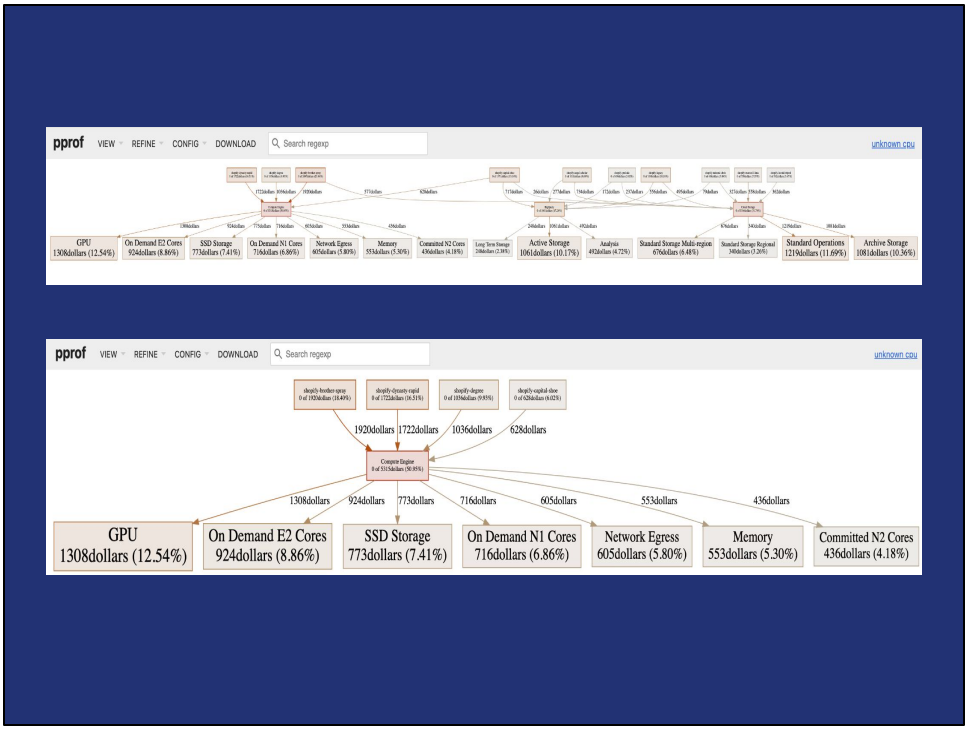


So, profiling tools + flamegraphs are useful for exploring hierarchical data to identify areas for optimisation - *expensive* operations 🤔 Someone on our team had the inspired idea to see if we could turn our existing perf tools to our billing data.



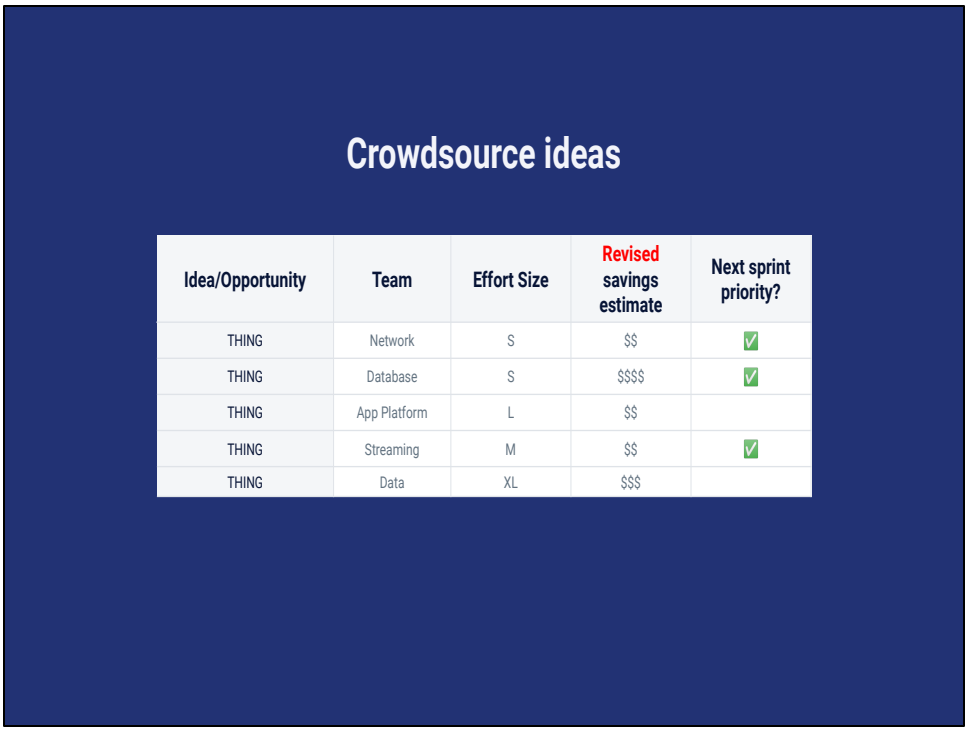
So we quickly figured out how we could export a view of our billing and service data into one of these intermediate formats and turned our performance tools on them - it worked really well.

These names and SKUs are completely fabricated - fever dreams of a random number generator.



Same data but converted to pprof. Again, this data is completely fictional, pay no attention to the names or numbers. We can quickly see that compute engine is more than half the bill here, and the fictional GPU SKU is the biggest part of that.

This turned out to be a great find - engineers were able to use tools they were already familiar with to get the context they needed to make better estimates.



Teams were good at estimating t-shirt sizes, but less good at estimating opportunity size - e.g. missing context like discounts we have negotiated. Items were being completed but impact varied.

We did a second pass, teams used the new tools to reevaluate their ideas to make different decisions on priority, and bring *resource reduction* estimates instead. We worked with engops/finops to estimate the savings ourselves.

Planned comms around a big push/new sprint - keep the momentum going, refocus efforts on work items we were more confident would be impactful.

This approach really accelerated the impact, bending the daily spend curve downward. We were much more successful in identifying the best ideas to work on.

So in general, what sort of opportunities turned out to be the

right things to focus on?

Sounds easy to say and obvious, but is something we had to take a beat to sort out. Without a bit of thought and focus it's very easy to spend effort on things that aren't actually impactful for what you're trying to achieve.

The large things were obvious and well known. Incremental wins on these systems are impactful in their own right, but also in our case unlocked some even bigger wins, such as removing two whole cluster per region for this particular app.

We were able to find projects/clusters/nodepools with low utilisation pretty easily from the center, but we had to cast a wide net through the org to find the people with the right context to find where the real opportunities were. We couldn't know e.g. some experimental infra could be turned off, when we found the right person with that context we could make that change happen pretty quickly.

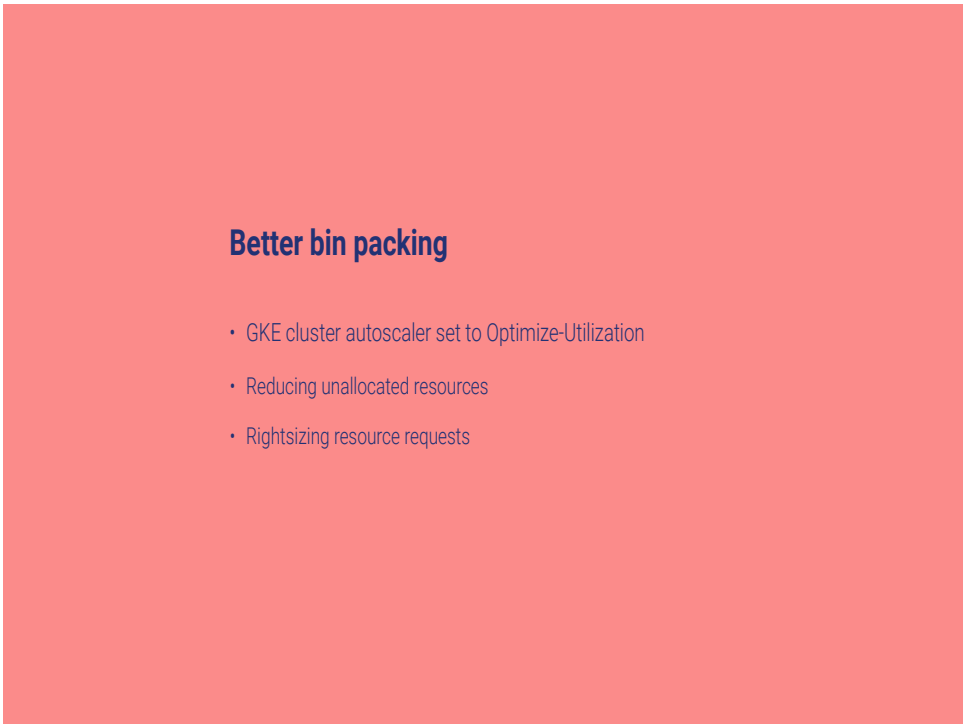
This is probably specific to our org, but I think it will be interesting regardless, and maybe some lessons will apply to you.

First win: The bin packing problem - extreme Tetris.

You have a number of things of varying size, that we need to efficiently pack into a finite number of bins of fixed size.

For us this means packing container based workloads - Kubernetes pods - onto compute nodes. How well (or not!) this is done can have a large impact on efficiency.

Image credit: Glynn Clements. GPL, https://commons.wikimedia.org/w/index.php?curid=3702251



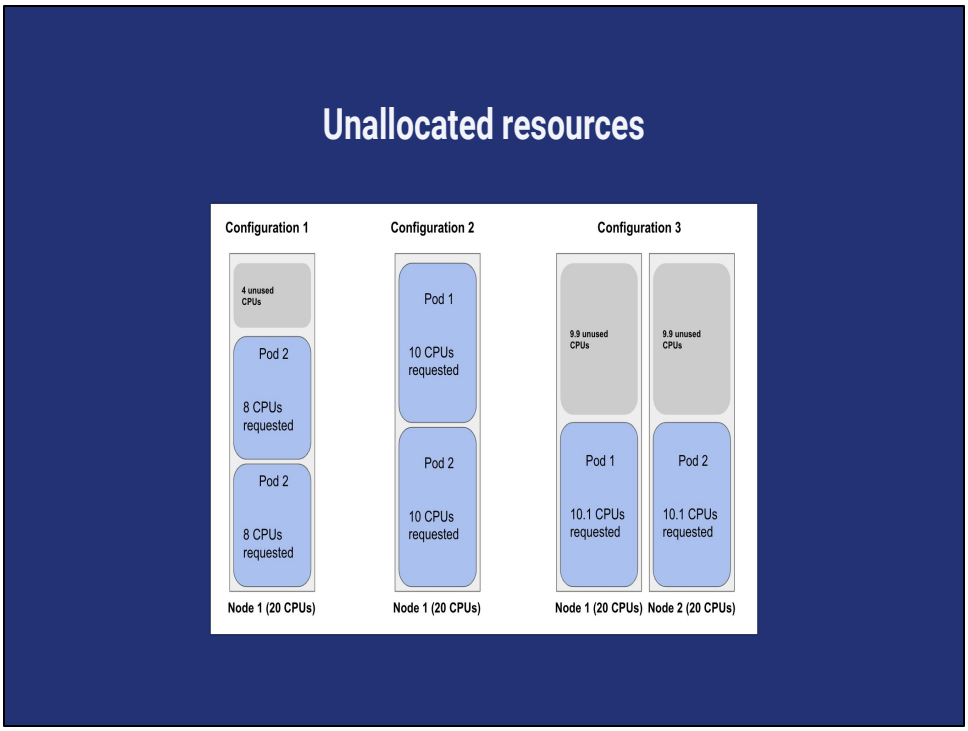
GKE autoscaler - add or remove nodes to match scheduling demands of pods. It has different profile which influence its behaviour, which are a bit of a black box. Prioritises increasing utilisation/allocation when scheduling new pods - rather than evenly distributing workloads - and more aggressively evict pods to achieve the first goal, so removes un[der]-utilised nodes can be removed.

Trade-offs:

- More pod churning, check your disruption budgets
- Higher utilization might impact workloads (i.e. resource requests might be too low)
- May not play well with stateful workloads

Unallocated resources - CPU and memory on a node which is not requested by a workload
Rightsizing - Using our observability tooling + VPA to find workloads requesting too much

Let’s look at those last 2 a little closer.



This is a contrived example.

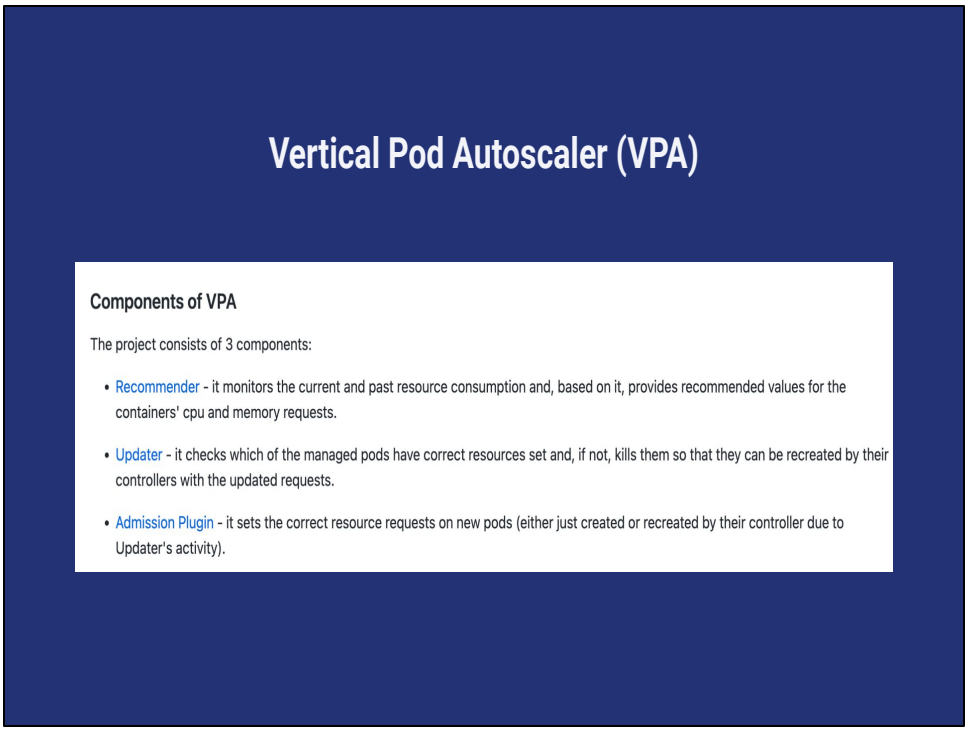
If a cluster has nodes with 20 CPUs allocatable per node and a workload with replicas that request 8 CPUs. This means at most 2 replicas can fit on one node, leaving 4 CPU cores unused but paid for. If the workload is scaled to 100 replicas the cluster will provision 50 nodes and 200 CPU cores will be wasted. If the requested CPU were to be increased to 10 CPUs per replica then 50 nodes would still be enough and 100% of the CPU resources would be used. But if the requested CPU is then raised to 10.1 CPUs per replica, it becomes impossible to deploy 2 replicas per node and 100 nodes will be provisioned where only 50.5% of the CPU are allocated. That small change has a disproportionately large impact on the waste of resources.

No autoscaler strategy or configuration can help with this.

This is how that looks when visualised using our profiling tools. This is a real example and compared to the fabricated one from earlier you can see it goes a little deeper - we also add cluster, app/namespace/owner, and individual workloads.

Summing kube_unallocated - attributing it to a ‘virtual’ workload - across the whole fleet told us there was a real opportunity here - it was one of our biggest ‘apps’ - visualising this way help us figure out the best places to start.

Even with a more aggressive node autoscaler, we could see we still had work to do, and reducing unallocated resource was a real opportunity. How to start? First things first: are workloads requesting the right amount of resources?



Second win: rightsizing resource requests. A great help with this is the VPA - vertical pod autoscaler

https://github.com/kubernetes/autoscaler/tree/master/vertical-pod-autoscaler

Ultimate goal: “frees users from the necessity of setting up-to-date resource limits and requests for the containers in their pods.” - this is a worthwhile problem. Resource requests bitrot very quickly after an application is first deployed and especially at somewhere like Shopify where we are constantly deploying new revisions with new features and behaviours.

Made up of several components which can be used independently.

The updater and admission plugin components are ‘handle with care’ tools.

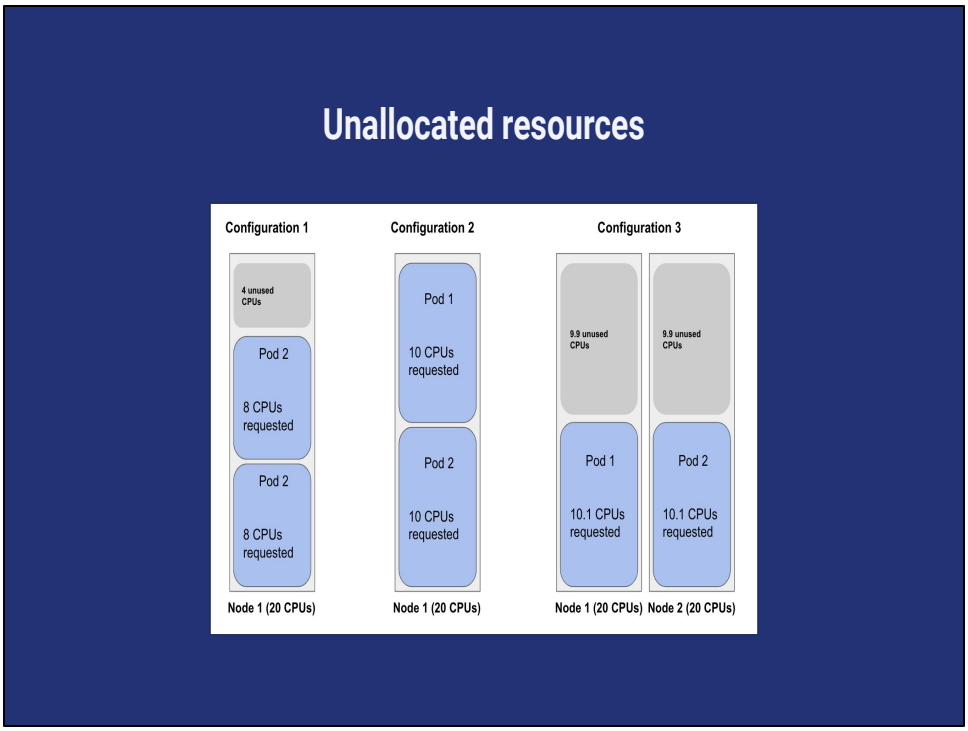
VPA recommender is a really good tool to help understand where resources requests and limits should be set - these can become very out of date over the lifecycle of an app. Bin packing well is hindered if the workloads are poorly sized or rapidly changing. We had actually already deployed the recommender to a lot of the fleet, so we had data to interrogate

The recommender works by deploying a custom resource targeting a real workload - typically a deployment - which tells the recommender to start monitoring the resource usage of its pods, and to make recommendations.. The resource is updated with various values over time, we want to focus on the target block - I could talk more about the other values but this isn't a VPA tutorial, and are mostly relevant if you're running with the other components - Target has the values that the VPA recommends for this workload, based on current and historical data

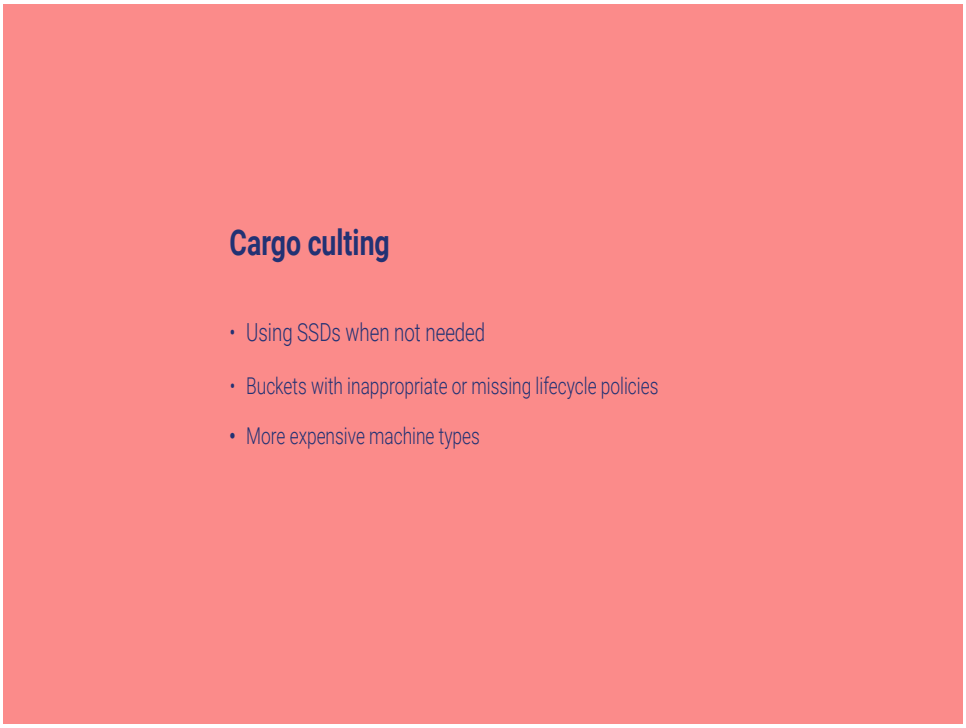
We built some extra tools to scrape this data into our observability systems, so we can track them overtime and overlay them with other data - like requests, limits, and actual usage - or compare across different deployments (other clusters or geographies). The `usage` metric here is a max aggregation - we care about the peaks.

Anywhere you see a big gap between the current request and recommendation levels is a good place to start. Note the resolution of the recommender might be too low for bursty workloads, so double check the recommendation against actual usage as measured by something else.

We also have some tools to notify owners where there is a big discrepancy/opportunity, and help people apply recommendations to their workloads (e.g. automatically open a pull request).



Going back to this view - pod sizing and node sizing have to be considered together. The most perfect resource requests for any particular workload might make it harder to effectively pack nodes. Having more, smaller pods - or fewer, larger ones - could give you bigger bang for buck. For your environment standard pod sizes might be appropriate to make this easier to reason about and tool for. Or, maybe you already have similar or predictable pod sizes - low variance - and you could change the shape of your nodes to better accommodate them. This is local vs global maxima stuff. YMMV, but remember that at the end of the day, your costs are a function of the size and number of nodes - not the size and number of pods.

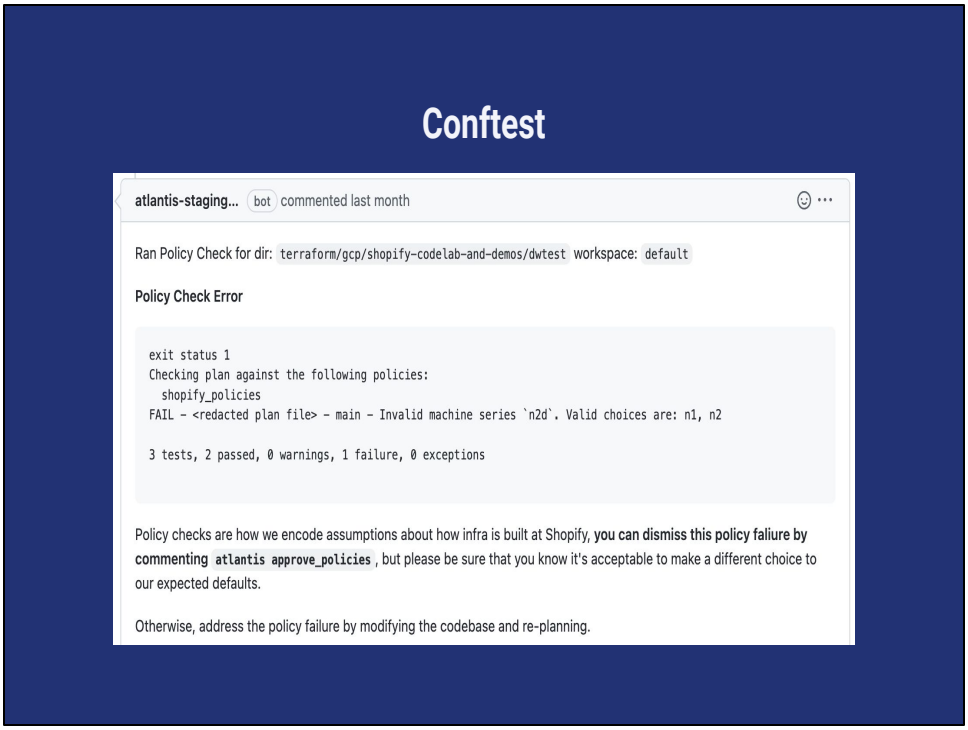


Third win: cargo culting.

Copying and pasting terraform! These are some of the inefficiencies we found.

Engineers at Shopify have lots of freedom by design, but in this case it was starting to hurt us, so we needed some guardrails - to give more context.

Discounts are a good example of context that very few people have, but can really impact the bottom line.



We use Atlantis, a great tool to orchestrate terraform on GitHub pull requests.

We have implemented some guardrails using conftest integrated with atlantis, like this example. Anyone can dismiss the policy checks - we always trust people to decide they know better - but this allows us to give context about the things we care about right in the critical path, not buried in a doc, chat, or a person's head.

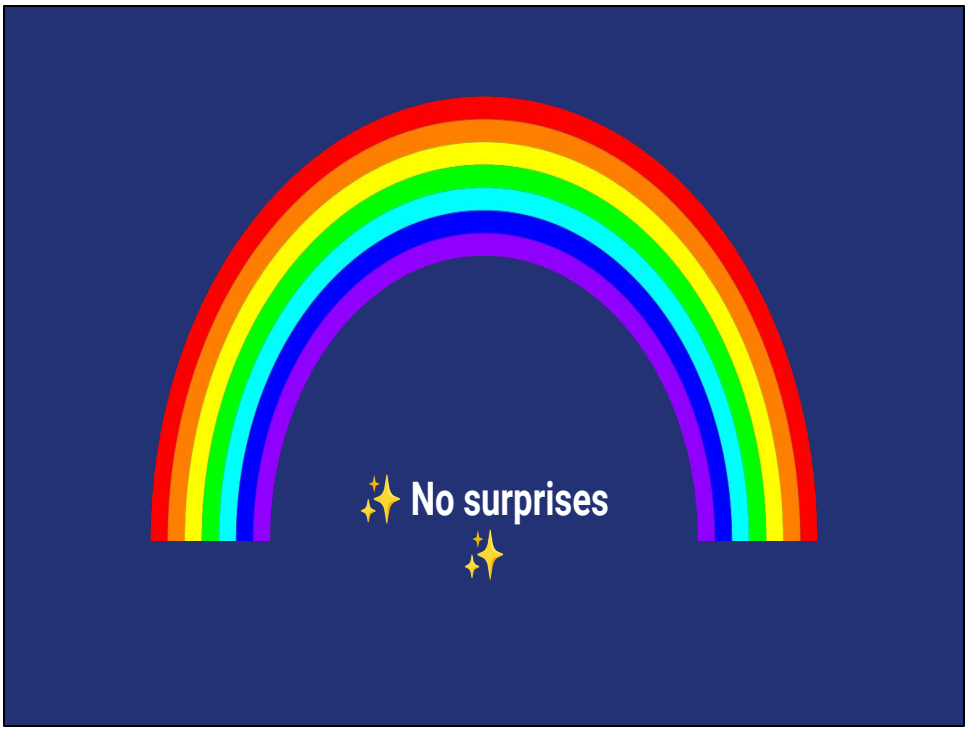
We can also look at the policy violations after the fact, to see if our policies - our expectations - are out of date.

Chapter 3: Getting to good

Taking a step back from what we did, thinking about what lessons we can draw.

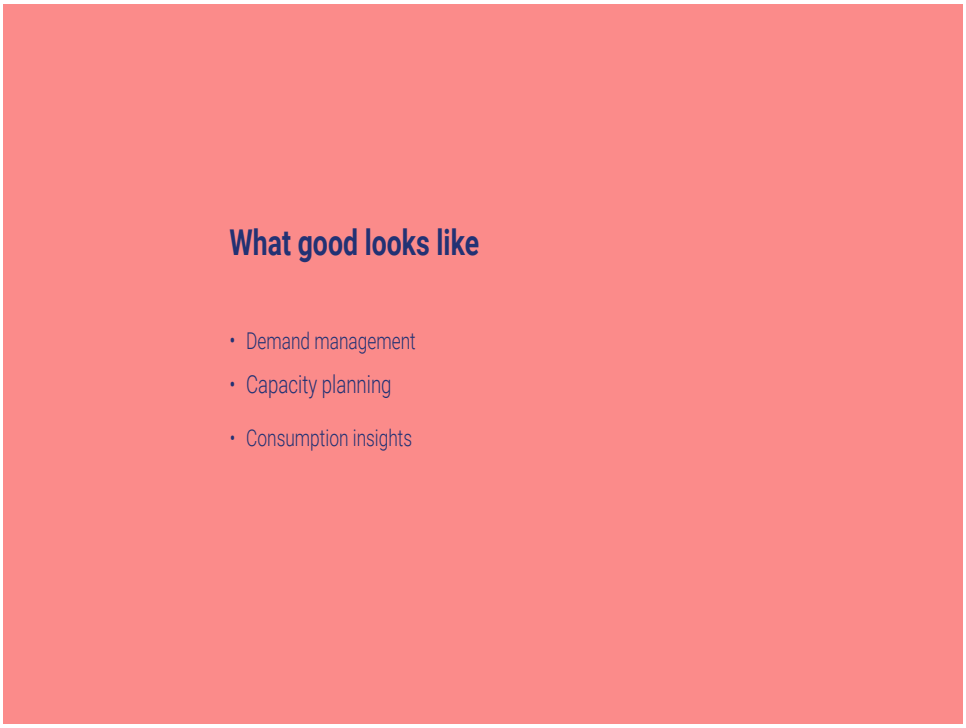
We don’t want to slip back to bad - not only for the business but also our teams, it was not a sustainable way to operate. “Save X$ or Y%” style cuts over and over will eventually cross resiliency or performance boundaries we don’t want to cross, and we don’t wan’t to keep intruding on teams existing roadmap.

This is the present and the work in progress for us, and is not all figured out. Selfish interest in sharing - I would love to talk to you if you have opinions on this!



So what might good look like?

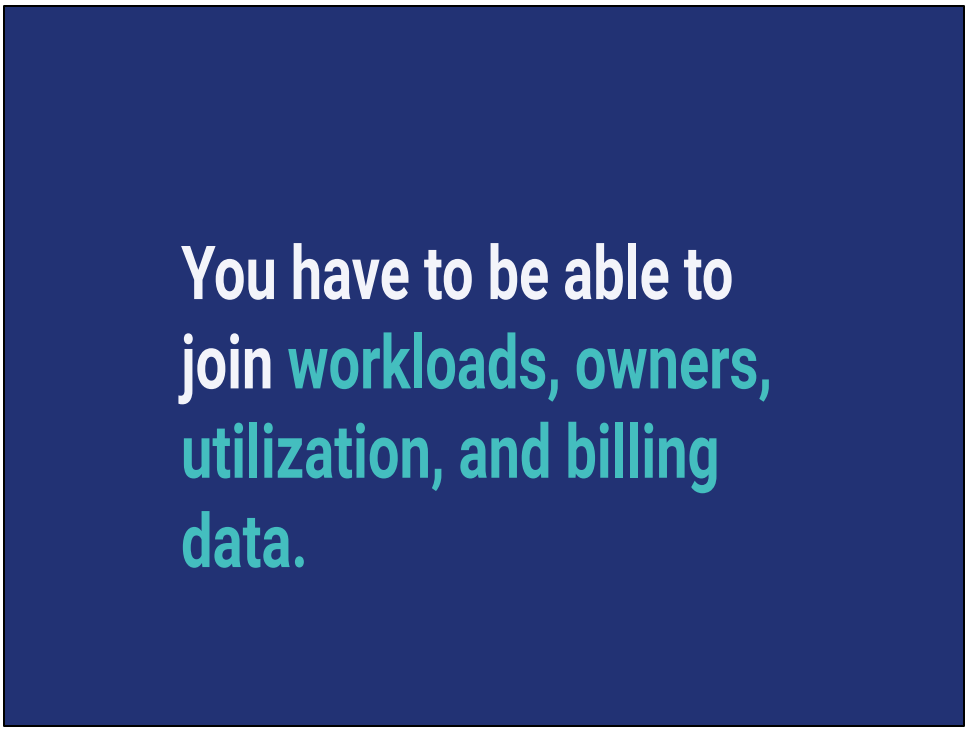
Overall theme for this is no surprises - and in order to not be surprised about something, you need to set some expectations ahead of time.



Demand management: Traffic forecasts, HPA/VPA/Cluster autoscaler configuration, can we scale to our peak demand requirements, scale tests

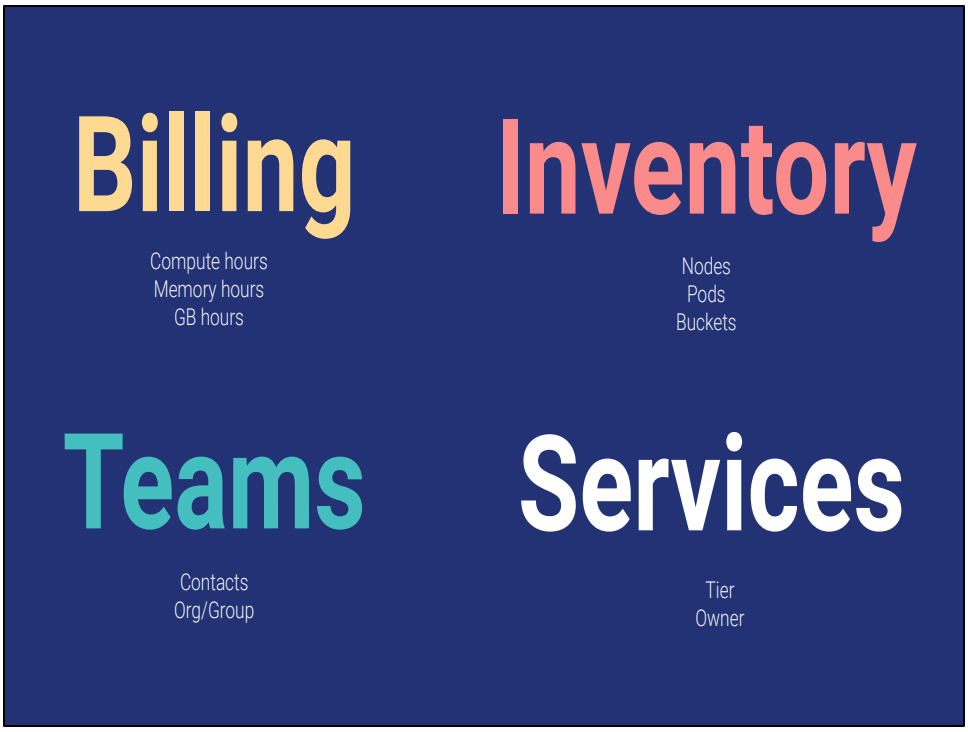
Capacity planning: resource forecasts, reservations, committed use discounts

Consumption insights: Attribution, utilization, anomaly detection

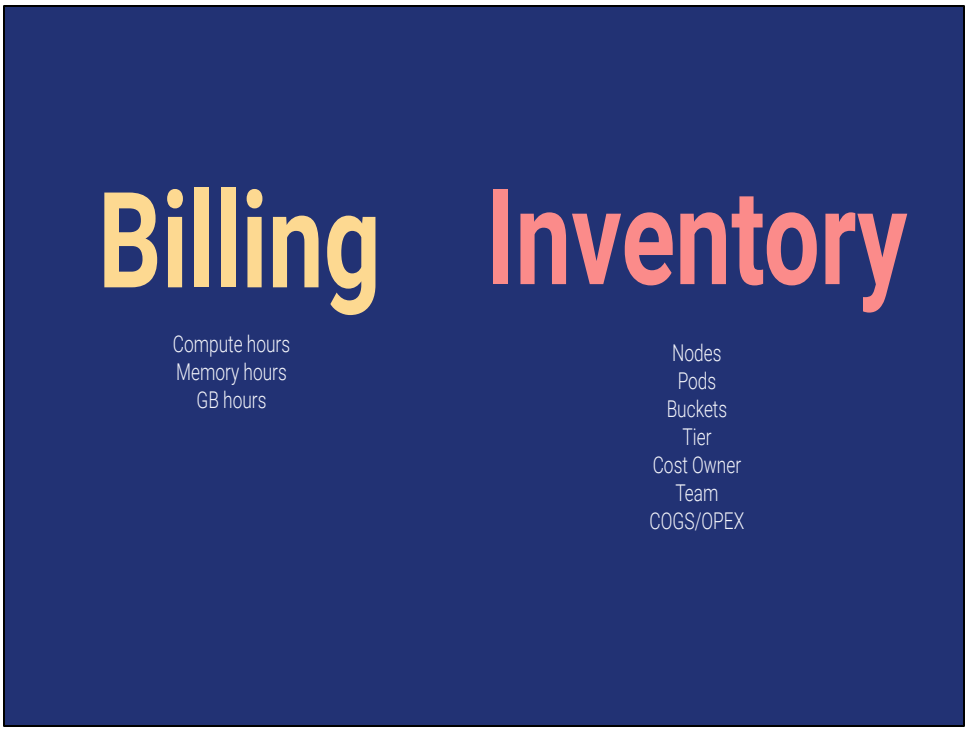


Just to talk about attribution a little more - I wanted to emphasise this learning for us. We needed this to ensure we could get the insights we needed

We could do subsets of this, but not all of it, and not all of it in the same place. This hindered our ability to find the right opportunities centrally.



We were joining data across these 4 - and more - data sources. Lowest common denominator becomes working by hand in spreadsheets - fine for ad-hoc things but not very scalable. It also had gaps.

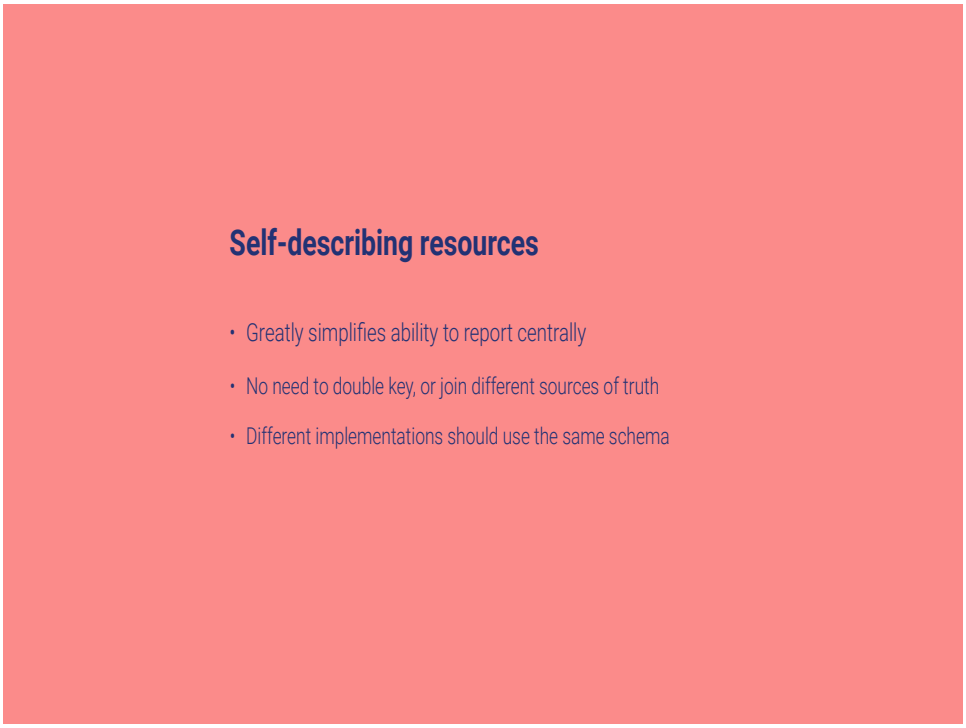


Adding more of this data directly onto the resources as metadata, e.g labels on Kubernetes resources and namespaces, GCP instances, storage buckets

Especially helpful when the unit you want to group by is not neatly contained in e.g. a single namespace, node pool, cluster etc

Resources should describe at least - what their costs should be attributed to, and who is responsible for maintaining them (i.e. who you would ping on an action item/chat message/issue tracker)

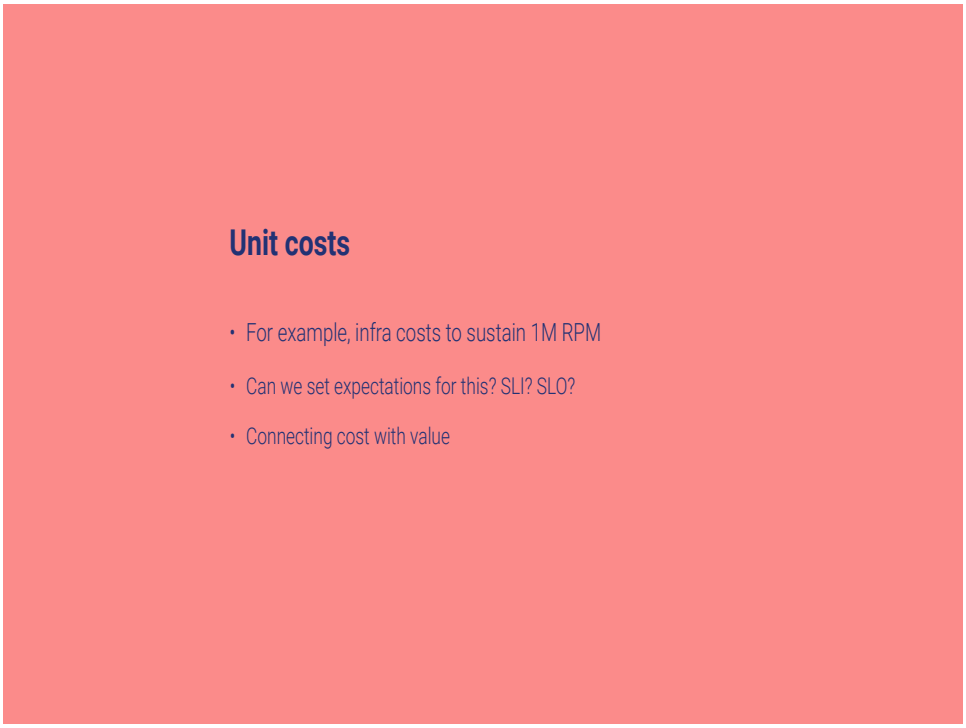
Also consider: compliance information (in scope for PCI?) or jurisdiction (GDPR?)



Reducing the difficulty level in understanding our infra on different dimensions.

So we can use the same keys in Kubernetes metadata - resource labels - as that in other systems. This needs some planning - constraints on character set and sizes/length can vary between systems, so a lowest common denominator needs to be found.

Foundation for different multi-tenant systems to attribute their usage accurately and meaningfully.



Cost in one side of the equation - a numerator without a denominator. Unit costs can quantify what your organization is buying in terms of the outputs of your applications - for example, capacity to support 1M RPM.

Over a long time horizon you need to get better economies of scale the more demand your infrastructure handles - otherwise the business will eventually fail - unit costs can help show that.

Connect cost with value - this is where I want to end

Value is where I want to end this talk, and where I think we should start.

Saving money is easy - turn it all off, delete all the data; there I fixed it - but that's not going to be a positive outcome for your business.

This doesn't exist in a vacuum.

Pick the low hanging fruit, given enough time passing you will always find some - back to the saw tooth. Then sure, look to optimise, but understand the trade offs. Chesterton's Fence!

This isn't quite precise enough either, if any of you can articulate this better I would love to hear about it!

Cost is one part of a nuanced equation to determine value. What are we buying with our spend, is that valuable to our business objectives - answering this is not simple arithmetic, and requires input from customers and stakeholders.

What is 100ms of p99 latency worth? 30 minutes of recovery time objective (RTO)? 12 months of retention? "What business objective does this support", "what value does this drive" are key to the value judgement. Like good SLOs - connecting what we're doing to the value we are trying to deliver to our users and customers.

So that's what good might look like: no surprises, and trying to connect as much of the spend as possible to the value it delivers.

I’m lucky enough to work with some very clever people at Shopify, I am here today representing their efforts so thanks to them, and thank you all for coming today.