Simon Eskildsen Interview on ScaleYourCode.com

7 mins summary

THE
@ScaleYourCode Show
Episode 14 - Scaling Shopify's Infrastructure and Rails app
"What would you do if you had 100x more visitors?"

ScaleYourCode.com

View original post

Simon Eskildsen has been working at Shopify for 2 years now. He works on Site Reliability, Performance, and Infrastructure. They jump around a lot in the infrastructure department, and Simon is currently working on setting up new data centers and completely isolating clusters of shops from one another.

A lot of his work revolves around resiliency. He also worked on moving Shopify to Docker, and has given a few talks about this at conferences.

[Speaker Deck] GOTO Chicago: Patterns for Docker Success (https://speakerdeck.com/sirupsen/goto-chicago-patterns-for-docker-success)

[Video] GOTO Chicago: Patterns for Docker Success (https://www.youtube.com/watch?v=ZnIiFWD7yUw)

[Speaker Deck] DockerCon 2015: Resilient Routing and Discovery (https://speakerdeck.com/sirupsen/dockercon-2015-resilient-routing-and-discovery)

[Video] DockerCon 2015: Resilient Routing and Discovery (https://www.youtube.com/watch?v=ZDeAEZHby_A)

Shopify's scaling challenges

Just to put it into perspective, Shopify handles about 300 million unique visitors a month.

One of his team's biggest challenge is what they call "flash sales". These flash sales are when tremendously popular stores sell something at a specific time. For example, Kanye West might sell shoes and combined with Kim Kardashian, they have a Twitter following of about 50 million people.

They also have customers who advertise on the Superbowl. Because of this, they have no idea how much traffic to expect. It could be 100,000 people showing up at 2:00 for a special sale that ends within a few hours.

How do they scale to these sudden increases in traffic, and even if they can't scale that well for a particular sale, how can they make sure it doesn't affect other stores? This is what we will discuss in the next sections, after covering Shopify's architecture.

Shopify's architecture

Despite Shopify's move to Docker last year, they have not moved away from their Monolithic application to Microservices.

(When asked about this, Simon told me that they do not take the overhead of Microservices lightly. However this move has positioned them for the future, perhaps...)

The architecture is fairly simple:

When a request comes in, it goes to Nginx and then to a cluster of servers running Docker with a Rails app.

Their data tier consists of things like:

- Memcached
- Redis
- ElasticSearch
- MySQL
- Kafka
- ZooKeeper

Other interesting points:

They run on their own hardware which is distributed among multiple data centers which are on different continents. However, they do run a few things on AWS.

To reduce cost, Shopify runs a multi-tenancy platform. This means they have multiple stores on a single server -- shopA.com and shopB.com are served from the same server, for example.

View original post

Moving to Docker has enabled them to run Continuous Integration on hundreds of thousands of lines of Ruby in about 5 minutes (15 minutes before Docker) and deploys to 300-400 nodes across 3 data centers take just 3 minutes (15 minutes before).

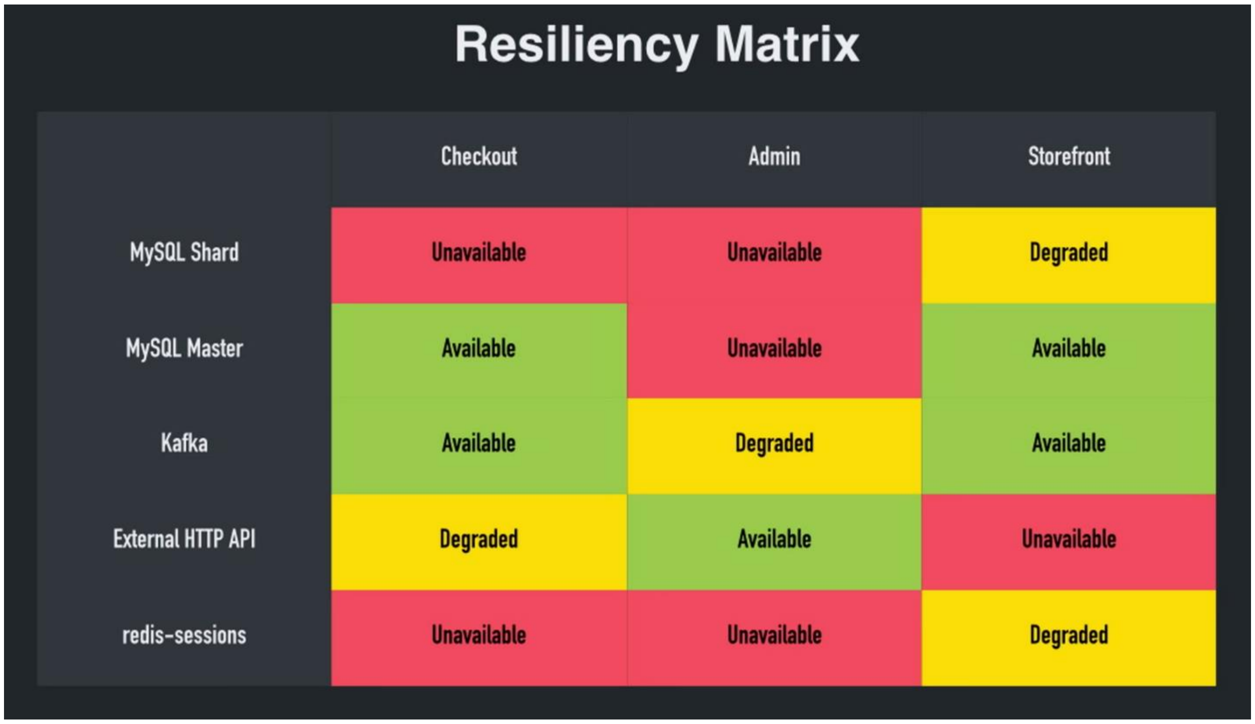
How they solve their challenges

With this architecture and facing these intense flash sales, how does the team at Shopify conquer its challenges?

Ideally, the platform could handle these spikes by itself. This is not completely implemented yet, so they use a checklist for performance checks.

In Kanye's case, they sat down for 2 weeks and did extensive passive load testing and performance optimizations by combing through critical parts of the platform. They also ran experiments by putting the entire storefront on [Fastly]() (CDN) instead of serving it from their own data center.

To run various tests, they use something called the Resiliency Matrix



Resiliency Matrix

	Checkout	Admin	Storefront
MySQL Shard	Unavailable	Unavailable	Degraded
MySQL Master	Available	Unavailable	Available
Kafka	Available	Degraded	Available
External HTTP API	Degraded	Available	Unavailable
redis-sessions	Unavailable	Unavailable	Degraded

View original post

The Resiliency Matrix helps you determine what happens to systems when a service goes down.

Say Redis goes down, and Redis is used as an integral part of checkouts. Do we take the whole site down and put it into maintenance mode? No, we can just sign everyone out and still allow them to go through checkout without their customer account. Then, once Redis is back up, associate email addresses with the customer account and fill in the gaps.

Go down the list of systems (like storefronts, administration, etc... in Shopify's case) and see what happens when a service goes down.

To help with this, Simon and the Shopify team have Open Sourced two great tools: Toxiproxy and Semian. Toxiproxy introduces controlled latency in whatever system you choose, and Semian is used to verify that you have no single points of failure.

In addition to this, they are able to over-provision since they own their own hardware. This is cheap for them.

Another big challenge for just about any company that has run into scaling issues is with the data store. Shopify is no different, especially considering that they handle financial transactions. Their databases cannot be out of sync.

To combat issues, Shopify sharded MySQL about 2 years ago. They shard pretty aggressively and are aiming at having more, and smaller, shards over time.

Simon went on to say that scaling databases is hard, though, especially sharding. It almost always should be a last resort. You can probably cache a lot before you have to do that.

But the good thing about shards is that they help isolate incidents. If something crazy is going on with a customer in a single shard, only a very small subset of the entire platform suffers from it.

View original post

Going back to resiliency testing, Simon reinforces that most of their database scaling issues have been fixed with a lot of resiliency work and automatic failovers (which I'm going to touch on in this next section).

Moving forward

Going forward, the team is looking at isolating tiers that serve the applications from each other. Another big thing they are working on is getting shops to run out of multiple different data centers on different continents at the same time. This is big for data locality and also to protect against unexpected events.

Protecting against unexpected events is also something Netflix has invested a substantial amount of resources in as Jeremy Edberg told me in his interview.

In addition to these plans, they are also looking at making failovers something they could easily perform multiple times a day. If you are curious about how they perform failover tests between entire data centers, refer to the interview page.

As it stands, they cannot failover an entire data center without temporarily taking down checkouts. However, they are working on a solution to overcome this:

When a request goes to a shard that is currently non-writable, they can queue the request at the load balancer. Once the shard is writable again, they can drain that queue. This means customers just see their browser spinning a little bit longer while maintenance is performed, and then the request goes through. This literally means they could perform an entire failover test without losing a single checkout.

Career tips

This is a question I get a lot: "How do you get experience working on large scale apps when you don't have access to a large scale app?"

Simon answered that a lot of things are not written down in blog posts, but organizations practice internally, when it comes to scaling. A good way of getting this knowledge is reading a blog like Highscalability.com, and listening to podcasts like ScaleYourCode.com (thanks for the plug, Simon!)

Constantly question how apps end up with a certain architecture. Write down words you don't understand and spend time researching.

Ultimately, you have to practice it to get good at it.

I followed this up by asking him what kind of skills he looked for in engineers they hire. He says they look for endless curiosity.

During the interview process, he likes to ask what projects people are working on. If they are working on a small app, he will ask "what would you do if you had 100 times more visitors?" "How would your database react?"

They might answer "it would be important to have an index." to which Simon will follow up with "Ok, so how do you make sure that this is indexed? How does an index work? How is it stored in the database?"

The best engineers that he sees are creating distractions for themselves -- be it debugging something or creating some sort of a mental challenge and allowing these distractions to be an opportunity.

Conclusion

I have to try and fit as much information in this summary as possible without making it too long. This means I cut out some information that could really help you out one day. I highly recommend you view the full interview.

You can watch it, read it, or even listen to it. There's also an option to download the MP3 so you can listen on your way to work. Of course, the show is also on iTunes (https://itunes.apple.com/tt/podcast/scale-your-code-podcast/id987253051?mt=2) and Stitcher (http://www.stitcher.com/podcast/scaleyourcode?refid=stpr)

View original post

Thanks for reading, and please let me know if you have any feedback!

- *Christophe Limpalair (@ScaleYourCode)*

View original post