Whooo’s calling Whoooo?

Jodi Spacek
Hootsuite
March 11, 2016

Part 1: Microservice Migration
How we adjust to our ever-changing environment leading to reasons why microservice calls are hard to track

Part 2: Microservice Mystery
Take a look at a case study and come up with some techniques to diagnose problems
Hootsuite collects, organizes and interacts with social network data
10+ million users 5000+ requests / sec

Lots of interesting Distributed Systems problems!

Business Uses: customer support, data analytics, predictions

One of our largest concerns today is dealing with legacy code and outdated infrastructure

#chillax
Part 1: Microservice Migration

The Code
Legacy Code: older code we’ve inherited that is 4-5 years old

Is it a good idea to remove and replace legacy code all at once?

- No, we need to consider how drastic changes affect 10 million users currently in the system
- We need to get all of the developers on board with the changes gradually

#hootdogs Baby Monty!!

#hootdogs Wise Monty
#YodaPug
and the ragtag team of HootDogs
How did Hootsuite adjust to hyper-growth?
Hacking together a solution in the monolith to keep up with this drastic growth in the user base

Why are we getting rid of the Monolith?

- PHP monolith doesn’t scale well and is ill-suited for enterprise use
- it’s difficult to keep code neat and tidy, it allows for bad coding behaviour
How have we addressed **consistent growth** in our user base?

- We can take the time now to move away from the PHP monolith code to Service Oriented Architecture
- Microservices fall under the umbrella of SOA

These isolated components help us to (1) **distribute network traffic** (2) **replace legacy code incrementally** and (3) **distribute work in our team**

We have **5000+ user calls per second**. How many **microservice** calls per user call? 5000+ requests/s multiplied by the complexity of the type of call

#soaftw
**monolith**: these ingredients (components) are baked into one big pie

**microservices**: pick and choose your ingredients individually

How easy is it to remove the apple and replace with raspberries?

- Should we remove the apple completely so that there’s a moment of time where nothing is on the plate?
- Should we put all of the raspberries on the plate and then after a certain amount of time remove the apple?
- Should we put one raspberry on the plate at a time? Remove parts of the apple that match the weight of the raspberry so that the weight is the same?

#deliciousdeliciousmicroservices
**blue-green deployments**: switch from old to new (blue to green)

What happens if we have problems with the green environment?

- switching environments is a quick change that is less complex than performing rollbacks ~ faster than a full redeployment of code which must be thoroughly tested before going to production
- the state in the green (new) environment may be corrupted and unusable even if we replace the new code with the old code version
**canary deployment:** Riskier changes where we want to discover the behaviour on production

What happens to the canary server if it starts failing?

- stop routing any requests to the canary
- swap out the canary server with the old version of the service
Part 1: Microservice Migration

Infrastructure

Where the Code Lives
Infrastructure Redesign

Motivation

Volume of retweets causes an outage of our entire system

this guy >>>

#supnotyourapps
Microservice Architecture

**Brokers:** queue and transform messages

**Routers:** determine best location to send messages

**Service Discovery:** automatically detect the location of a microservice

**Workers:** microservice nodes in a cluster

**Fault Tolerance:** operates correctly even when component fails
Monolith Infrastructure Migration

- Most of our older services & THE MONOLITH live in EC2 Classic
- Use a bridge to direct traffic to our new services in VPC (Virtual Private Cloud)
- ASG (auto scaling group) lives in VPC and it helps us to deal with changes in the volume of requests
- How? ASG can scale up by adding new nodes in the cluster, and scale down when traffic is lower to save $$

#evenmoremigrations
Code Name “Back to the Future”

- HTTP is an oldie but a goodie
- Service discovery & load balancing with Nginx & Consul
- Nginx: HTTP proxy with caching
- Consul: Distributed (K, V) store

If one of the nodes goes down, but the request was already sent (in-flight), nginx can redispacht it to another node

Can any request ever be dropped?

- If the number of requests sent to a downed node exceeds the Nginx buffer storage size, requests will be dropped
Part 2: Microservice Mystery

What do we know?
#somanycalls
How micro is micro? Microservices are of varying size & complexity, they handle small pieces of logical functionality that make microservices easier to distribute and replace.

How micro is too micro? You don’t want to make your microservices so tiny that the advantages of this design are overshadowed by having to make so many calls that it’s a networking nightmare.

#micromicromicro
Part 2: Microservice Mystery

What can we see?
**logstash**: centralizes log data and standardizes them for elastic search

**elasticsearch**: real time data analytics

**kibana**: visualization tool for elasticsearch

#elkstack
What kinds of problems are caused by decentralization of our logs?
Logs are spread all over our servers and are hard to track

What kinds of features does the ELK stack provide?
Functionality to coordinate different log formats, regardless of the tag placement and format

What can everyone understand in the ELK stack?
Kibana’s visual clues for behaviour changes in graphs

#elkstack
How do we make connections between calls that are logged?

- We need to make the logical connections between microservice calls ourselves by searching for keywords to view logs in a list

Is this an easy thing to do?

- This could be an easy task if the microservice calls are simple
- But simple calls don’t usually cause complex issues that are difficult to track!
Part 2: Microservice Mystery

The Case
Send a request to update Twitter, Instagram, and Facebook

What happens after the first few calls from the first microservice?
Where would you start your investigation?
<table>
<thead>
<tr>
<th>Name</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>am-i-up-docker</td>
<td>/etc/sensu/plugins/check-am-i-up-for-containers.rb</td>
</tr>
<tr>
<td>apigateway1_disk</td>
<td>/etc/sensu/plugins/check-snmp.rb -C ::api_gateway.community:: -o 1.3.6.1.4.1.2021.4.11.0 -w 1024 -c 512 -h gateway1.externalapi.us-east-1.hootops.com</td>
</tr>
<tr>
<td>apigateway1_health</td>
<td>/etc/sensu/plugins/check-http.rb -k -u <a href="https://gateway1.externalapi.us-east-1.hootops.com:8443/sgg/ping">https://gateway1.externalapi.us-east-1.hootops.com:8443/sgg/ping</a> --response-code 200</td>
</tr>
<tr>
<td>apigateway1_memory</td>
<td>/etc/sensu/plugins/check-snmp.rb -C ::api_gateway.community:: -o le -O 1.3.6.1.4.1.2021.4.11.0 -w 1024 -c 512 -h gateway1.externalapi.us-east-1.hootops.com</td>
</tr>
</tbody>
</table>

#sensuhealth
How many **different places** do we need to check?

How many **developers** would need to do this?

How would we **coordinate** their efforts to put together a **hypothesis**?

Can we get rid of some of the **stress points** in this process?

Can these **clues** be **connected** in some way to help our analysis?

#whatwouldsherlockdo
Activity: Sherlock & Watson

Connect the Clues

Can these microservice clues be connected in some way to help our analysis?

Let’s take a couple of minutes to work in pairs and brainstorm on a solution!
Microservice Inspiration

What would Sherlock do?

RESEARCH
What are other companies doing?

- Inspiration from Google’s Dapper
- Constant deployments mean we need a dynamic solution
- Can understand real-time system behaviour
- Helps to understand exceptions
Bright Idea: what if we link our microservice calls?

This will help to:

- Troubleshoot issues
- Find points of stress in the system
- Allocate resources (people and systems)
<table>
<thead>
<tr>
<th>Hootsuite’s Feather Finder</th>
<th>Google’s Dapper</th>
</tr>
</thead>
<tbody>
<tr>
<td>● UUID List in the request header</td>
<td>● Instrument RPC code</td>
</tr>
<tr>
<td>● <strong>in-band</strong>: trace is inside of the request itself</td>
<td>● <strong>out-of-band</strong>: trace is outside of the request tree</td>
</tr>
<tr>
<td>● 2 points of contact with duration</td>
<td>● 4 points of contact</td>
</tr>
<tr>
<td></td>
<td>● more accurate timing data</td>
</tr>
</tbody>
</table>

Is this enough information for us to deduce, Sherlock style?

Yes, the **duration** of each call and the complete list of microservices in a call is helpful for most cases.
Microservice Mystery

Back to the Case: Let’s try out our Call Tree
Can you spot the microservice call that failed?

The call from Data to Push has failed for Instagram.

What are the implications of this problem?

This is a very difficult problem to solve, and can result in a dangling reference.
Project Feather Finder?

This is a great idea! Let’s code it up!

#socloseyetsofar
How can we show the usefulness of this tool to all developers?

- 2-day company wide hackathon
- Integrate a tracing system by reusing ELK stack
- Embed information in our requests
- Reuse the existing logging mechanisms in PHP and Scala

#thisisprojectfeatherfinder
Project Feather Finder Growth
Resources


http://twitter.github.io/zipkin/

The Verification of a Distributed System (short overview from Twitter)

No compromises: distributed transactions with consistency, availability, and performance (RDMA)
Thank you! Questions?

#hazquestions?