Containerizing your monolith

Jano González (@janogonzalez)
Introduction
What's SoundCloud
What’s SoundCloud

> 200M Tracks

> 20M Creators

Many SoundCloud Rappers
Motivation

2017-2018

Migration

Keep moving
What's the balance?
Contents

Our migration to microservices

The monolith

Containerizing the monolith

Conclusion
Our migration to microservices
How we started

$ rails new soundcloud
Around 2012

3rd Party Apps

iOS / Android

Soundcloud.com

The Mothership

Public API

Domain Logic

MySQL Cluster
Around 2012

- 3rd Party Apps
- iOS / Android
- Soundcloud.com
- Public API
- Domain Logic
- MySQL Cluster
- Microservice
- Microservice
- Microservice
Around 2012
Around 2012
After

Public Internet

- Mobile Clients
- Web Clients
- 3rd Party Clients
- api-mobile
- api-web
- api-public-api

Microservice

Microservice

Microservice

Microservice
But what about deployment?
Deployment
Our abstractions

Component
Environment
Zone
## Deployment

### The process

<table>
<thead>
<tr>
<th>Instance</th>
<th>Action</th>
<th>Test</th>
<th>Publish</th>
<th>Promote</th>
<th>Deploy</th>
</tr>
</thead>
<tbody>
<tr>
<td>29-d09ca91</td>
<td>VSM</td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Green" /></td>
</tr>
<tr>
<td>28-b854736</td>
<td>VSM</td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Green" /></td>
</tr>
<tr>
<td>27-9d16a0b</td>
<td>VSM</td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Green" /></td>
</tr>
<tr>
<td>26-9d16a0b</td>
<td>VSM</td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Green" /></td>
</tr>
</tbody>
</table>

*Revision details and Local Time are also included.*
The monolith
The monolith
Core entities

Track
User
Playlist
The monolith

The technology

360 Chef provisioned bare-metal machines

Rails 2.3

Capistrano deployment
The monolith
The architecture

- Public Web
- Public API
- Strangler
- Many microservices
- Internal API caching/strangler
- Another BFF
- Internal API
- Workers
The monolith
The components

Public API

MoshiMoshi (Internal API)

Workers

Cron

Shell

Migration

Public Web

MoshiMoshi Comments (Internal API)

Assets
The monolith
The hosts

- component
- mtail
- statsd
- statsd-exporter
- passenger-exporter
- ...
The monolith
The issues

Utilization
Deployment
Lack of confidence
Containerizing the monolith
Throw it into Kubernetes
Congrats, your monolith is a microservice now
Thank You!
The project

1.5 Engineers

1 year until the last bit was cleaned up
The first milestone
The first milestone

Docker development container

Tests on GoCD
The proof of concept
The proof of concept
Does it even work?

First staging component
The proof of concept
Where are my env variables?
The proof of concept
Env variables with The Perl Hack™

# Declares env variables
<% config.env_variable_names.each do |name| %>
  env <%= name %>;
<% end %>

# Sets a Perl handler for each env variables, to made them available in passenger without having to write its values on this config file
<% config.env_variable_names.each do |name| %>
  perl_set $<%= name.downcase %> 'sub { return $ENV{"<%= name %>"}; }';
<% end %>
Productionizing
Productionizing
Does it run where it matters?

Deployment

Monitoring

Logs
Productionizing
Anatomy of a traffic serving pod

- component
- statsd
- statsd-exporter
- passenger-exporter
- mtail
- twemproxy
- twemproxy-cu
- twemproxy-exporter
- init
Productionizing
Sizing the pods

Don't choke service discovery

Be allocatable
Productionizing
Sizing the pods

3

16

CPU units for main container

Passenger processes
Productionizing
Stdout v/s the log metrics exporter

Component

STDOUT

Log aggregator

File???

Mtail
Productionizing
Mtail with The Rotatelogs Hack™

```bash
11  exec 2>&1
12  exec &> >(rotatelogs -e -L "/tmp/stdout.lnk" -n 2 "/tmp/stdout.log" "50M")
13
14  exec /usr/sbin/nginx \
15    -c /srv/mothership/app/config/nginx/nginx-moshimoshi.conf \
16    -g "daemon off;"
```
Productionizing Public API
Productionizing Public API

Orchestration
Productionizing Public API
DNS latency and our excessive usage
Productionizing Public API
CoreDNS and the DNS Hack™

class Resolv
class DNS
  class Config
    # Internal optimization, we consider those domains absolute
    FORCED_ABSOLUTE_DOMAINS = /\.(com|net|io)\z/

    def generate_candidates(name)
      candidates = nil
      if FORCED_ABSOLUTE_DOMAINS =~ name
        name = "#{name}."
      end
      name = Name.create(name)
Productionizing Internal API
Productionizing Internal API

Highest throughput
Productionizing Internal API

Latency was too high
Productionizing Internal API
Optimize GC and make cheaper SQL queries
Productionizing Internal API
Errors spikes during deployment
lifecycle:
  preStop:
    exec:
      command: ['
/bin/sh', '-c', 'sleep 15 && /usr/sbin/nginx -c /srv/mothership/app/config/nginx/nginx-{{ component }}.conf -s quit']
Productionizing Internal API
Errors spikes during deployment (still???)
Productionizing Internal API
The Pre Start Trick™

readinessProbe:
  httpGet:
    path: /-/health
    port: 8200
    initialDelaySeconds: 60

passenger_pre_start http://<%= config.hostname %>:8200/-/health;
The rest
The rest

- Workers
- Cron jobs
- Shell / Migration hosts
- Cleanup!
Remove capistrano deploy #4259

Merged janogonzalez merged 2 commits into master from remove-capistrano-deploy on 12 Sep 2018

Conversation 0  Commits 2  Checks 0  Files changed 25

janogonzalez commented on 7 Sep 2018 • edited

Removing the Capistrano deployment. Now the `config/database.yml` and `config/migration_database.yml` are versioned.

This can be merged when the migration node is ready for the new migration procedure.

👍 2
Current status
Current status
Number of pods

On-prem: ~1000
Cloud: ~140
Current status
Traffic

On-prem RPS: 25K
Cloud RPS: 3K
## Current status

Many deploys

<table>
<thead>
<tr>
<th>Instance</th>
<th>Init</th>
<th>deploy-canary-and-promote-stable</th>
<th>deploy-non-workers-deploy-and-configure-related-systems</th>
<th>deploy-workers</th>
</tr>
</thead>
<tbody>
<tr>
<td>126-e9300e7</td>
<td>VSM</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>125-2366626</td>
<td>VSM</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>124-d18bc6f</td>
<td>VSM</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>123-02f6db2</td>
<td>VSM</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>122-485b6f7</td>
<td>VSM</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>121-64329b6</td>
<td>VSM</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conclusions
Conclusions
What we solved

One Infrastructure

One Delivery Process
Conclusions
How we did it

Step by step
Controlled rollouts
Managing expectations
Conclusions

Benefits

Improved utilization

Increased confidence

Enabling new initiatives
Conclusions
Should you do it?

Assess current progress
Evaluate costs and benefits
Thank You!
@janogonzalez

https://soundcloud.com/janogonzalez