What's Yelp?

- connect people with great local businesses
- website, apps, mobile site
- 142 million monthly unique visitors
- 77 million reviews
Yelp for Biz Owners

• measure visitor activity on your page
• interact with customers
• upload photos
whoami

backend developer for the Biz Owner App
worked on the main Yelp app backend before that
Python user since 2008
did a lot of Django work in the past
Yelp: a brief history lesson

• founded in 2004
• all code in one central repository (‘yelp-main’)
• web, mobile web, mobile backend, business owner site
• a lot of homegrown code
• new abstractions introduced without removing the old ones
• as Yelp grew, this started to become a bottleneck
The Yelp push process

Code deployments ("pushes") are done several times a day

Run by a pushmaster, an engineer with production system access

People join a push ("pickme")

<table>
<thead>
<tr>
<th>Time</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Friday Morning</td>
<td>pickme by 9:45 am</td>
</tr>
<tr>
<td>Pushmaster</td>
<td>sjensch</td>
</tr>
<tr>
<td>Push Type</td>
<td>morning</td>
</tr>
<tr>
<td>Branch</td>
<td>deploy-78-cent-doughnuts</td>
</tr>
<tr>
<td>Created</td>
<td>07/09/15 16:35:20</td>
</tr>
<tr>
<td>Modified</td>
<td>07/10/15 14:18:09</td>
</tr>
<tr>
<td>Requests</td>
<td>Load</td>
</tr>
<tr>
<td>Friday Early -</td>
<td>pickme by Thursday 6PM. Verify and be on #yelp by Friday 8AM <em>sharp</em></td>
</tr>
<tr>
<td>Pushmaster</td>
<td>sjensch</td>
</tr>
<tr>
<td>Push Type</td>
<td>morning</td>
</tr>
<tr>
<td>Branch</td>
<td>deploy-bang-keyboard-get-limit</td>
</tr>
<tr>
<td>Created</td>
<td>07/09/15 09:17:45</td>
</tr>
<tr>
<td>Modified</td>
<td>07/10/15 09:21:36</td>
</tr>
<tr>
<td>Requests</td>
<td>Load</td>
</tr>
<tr>
<td>Friday EuroPush</td>
<td>pickme by 10:30am CEST</td>
</tr>
<tr>
<td>Pushmaster</td>
<td>sjensch</td>
</tr>
<tr>
<td>Push Type</td>
<td>regular</td>
</tr>
<tr>
<td>Branch</td>
<td>deploy-genisys</td>
</tr>
<tr>
<td>Created</td>
<td>07/10/15 00:11:22</td>
</tr>
<tr>
<td>Modified</td>
<td>07/10/15 05:34:27</td>
</tr>
<tr>
<td>Requests</td>
<td>Load</td>
</tr>
<tr>
<td>Thursday afternoon</td>
<td>pickme my 1:30</td>
</tr>
<tr>
<td>Pushmaster</td>
<td>sjensch</td>
</tr>
<tr>
<td>Push Type</td>
<td>regular</td>
</tr>
<tr>
<td>Branch</td>
<td>deploy-nutritious-shopkeeper</td>
</tr>
<tr>
<td>Created</td>
<td>07/09/15 10:28:35</td>
</tr>
<tr>
<td>Modified</td>
<td>07/09/15 17:14:14</td>
</tr>
<tr>
<td>Requests</td>
<td>Load</td>
</tr>
</tbody>
</table>
Running a push

Automatic checks make sure there are no merge conflicts
deployment branch is deployed to a stage system
after verification, it's sent to production
~2 hour process, with no upper bound

Deploy Dashboard v2
Yelp Hack'16, brought to you by ztm

- **Bounce Summary**

  - **sfo2**
    - `continue_makeLive` in progress on version `r201507140746-17e18e955-deploy-sf-robots`, started at 2015-07-14 08:20:23
    - 38 hosts completed / 9 hosts in progress / 15 hosts waiting / 0 hosts failed

  - **iad1**
    - `continue_makeLive` in progress on version `r201507140746-17e18e955-deploy-sf-robots`, started at 2015-07-14 08:20:23
    - 44 hosts completed / 9 hosts in progress / 20 hosts waiting / 0 hosts failed

  - **uswest1prod**
    - `continue_makeLive` in progress on version `r201507140746-17e18e955-deploy-sf-robots`, started at 2015-07-14 08:20:23
    - 50 hosts completed / 9 hosts in progress / 30 hosts waiting / 0 hosts failed
I HAVE A SOLUTION
FOR THAT
Modularize

you can run only so many pushes a day
so let’s build services!
Why Services?

each service is developed and deployed independently

services are usually small, covering only one aspect or set of features

easy to parallelize thanks to async HTTP requests, so it might even speed your code up

http://bit.do/fowler-service
http://bit.do/microservices
https://github.com/Yelp/service-principles
Why Not Services?
Why Not Services?

consistency is really hard
no clear dependency / usage graph
need to maintain interfaces “forever”

testing one huge, mostly self-contained codebase is easy; how do you test services?
How to make sure it doesn't break

unit tests

...are great, but not enough

a lot of breakage if interfaces change

our solution: acceptance tests

as close to production as possible without using dedicated stage environments
Testing SOA at Yelp

spin up all components you need, using production code
done with docker-compose
heavyweight: take time to run, setup grows with the number of services you call
Setting up acceptance testing

```
configs:
build: acceptance/configs/
volumes:
  - "/logs:/tmp/logs"
bizapp:
build: .
links:
  - bizfeed
  - businessmedia
  - internalapi
  - sessionsservice
  - ruleserv
volumes_from:
  - configs
ports:
  - 13849

internalapi:
image: docker-dev/internalapi-testing
links:
  - gearman
  - memcache
  - databaseprimary
  - databaseaux
  - databasebatch
  - geocoderservice
environment:
  YELP_USE_GEARMAND: True
```
The Yelp service stack

originally we used tornado; didn’t work well

now: Pyramid, uWSGI, SQLAlchemy

HTTP and JSON for communication

Swagger to specify the API and do the inter-service calls
**default**

### Path

<table>
<thead>
<tr>
<th>Method</th>
<th>Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>/business/{business_id}/slideshow</td>
<td>(getSlideshowMedia) Get a sorted list of slideshow media items by business_id</td>
</tr>
<tr>
<td>POST</td>
<td>/business/{business_id}/slideshow</td>
<td>(updateSlideshowMedia) Update the sorted list of slideshow media items by business_id</td>
</tr>
<tr>
<td>GET</td>
<td>/business/{business_id}/yelp-sorted</td>
<td>(getYelpSortedMedia) Get a sorted list of yelp-sorted media items by business_id, paged by offset/limit</td>
</tr>
</tbody>
</table>

### Response Class (Status 200)

Model | Model Schema
---|---
BusinessMediaListObject | {
  total_items (integer): The total number of items,
  media_items (Array(MediaItem)): Sorted list of media items for this business
}
MediaItem | {
  media_type (string, optional): Type of media item = ['photo', 'video'],
  id (integer, optional): photo_id or video_id
}

### Response Content Type

- application/json

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
<th>Parameter Type</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>business_id</td>
<td>(required)</td>
<td>Business ID</td>
<td>path</td>
<td>long</td>
</tr>
<tr>
<td>limit</td>
<td>(required)</td>
<td>Limit</td>
<td>query</td>
<td>long</td>
</tr>
<tr>
<td>offset</td>
<td>(required)</td>
<td>Offset</td>
<td>query</td>
<td>long</td>
</tr>
</tbody>
</table>

**Try it out!**
Swagger

does request and response validation
data structure and basic type checking of the individual fields
works dynamically by reading a service’s spec, no need to generate and update client libraries
The Biz App service

a special snowflake since it’s one of the very few services reachable from the outside

not constrained to one area (like business media)

no local datastore

oftentimes just a proxy, calling yelp-main and other services
The Biz App service API

- RESTy model
- do multiple calls (to different endpoints) to match related resources
- get concurrency for free (if using async calls)
- some say a lot of simple calls are easier to scale than fewer complicated ones
The Biz App service API

one endpoint per client (app) page

for write (POST) endpoints, also send the client the data it needs to display the follow-up page

aggregate and enrich data we retrieve from yelp-main and other services

a high-level interface that translates to our low-level service APIs
Developing a mobile app backend

Mobile apps have releases
in our case, they’re synchronized, both in time and in features
iOS apps need to be reviewed; might take 10+ days
you probably also want to test before releasing
meaning: API needs to be done sooner than client implementation
way sooner than release date
It's not web development
you can’t upgrade apps whenever you upgrade the server
actually, some users never upgrade
so your APIs need to be backwards compatible - forever

GET /business/{business_id}/feed/v1
GET /business/{business_id}/feed/v2
Multi-version API

maintaining multiple versions can become costly
adding fields is backwards compatible

```
"longitude": {
  "type": "number",
  "format": "float",
  "description": "Business longitude"
},

"timezone": {
  "type": "string",
  "description": "Timezone the business is in. This is a pytz timezone (e.g. America/Los Angeles) and has the same format as the timezone sent to the consumer apps."
},

"rating": {
  "type": "number",
  "format": "float",
  "description": "Business rating"
}
```
Monitoring

number of requests, server errors, task queues, sent push notifications...

ElastAlert: it's open source!

app crashes: Crashlytics

you need an on-call rotation: we use PagerDuty
More about services @Yelp

Scott Triglia: Arrested Development - surviving the awkward adolescence of a microservices-based application

Friday, 11am, Python Anywhere room
The shameless plugs

We're hiring! Check out yelp.com/careers

Interested? Contact me even if you don't find an open job position that fits you, we're always looking for talented people!

yelp.com/engineering aggregates the blog posts, open source projects and more

follow us on Twitter: @Yelp, @YelpEngineering
Have fun and win prizes

The Yelp Dataset Challenge: yelp.com/dataset_challenge

Want to work with data, but have no data lying around?

The Challenge Dataset:
- 1.6M reviews and 500K tips by 366K users for 61K businesses
- 481K business attributes, e.g., hours, parking availability, ambience.
- Social network of 366K users for a total of 2.9M social edges.
- Aggregated check-ins over time for each of the 61K businesses

Get the Data
THANK YOU

questions?