A Year* With Apache Aurora:
Cluster Management at Chartbeat

Rick Mangi

Director of Platform Engineering

@rmangi / rick@chartbeat.com

ABOUT US

Chartbeat is the content intelligence platform that empowers storytellers, audience builders and analysts to drive the stories that change the world.

Key Innovations

- Real Time Editorial Analytics
- Focus on Engaged Time
- Solving the Social News Gap
- NEW: Intelligent Reporting

Power to the press.

THIS TALK

- Who we are
- What our architecture looks like
- Why we adopted Aurora / Mesos
- How we use Aurora
- A deeper look at a few interesting features

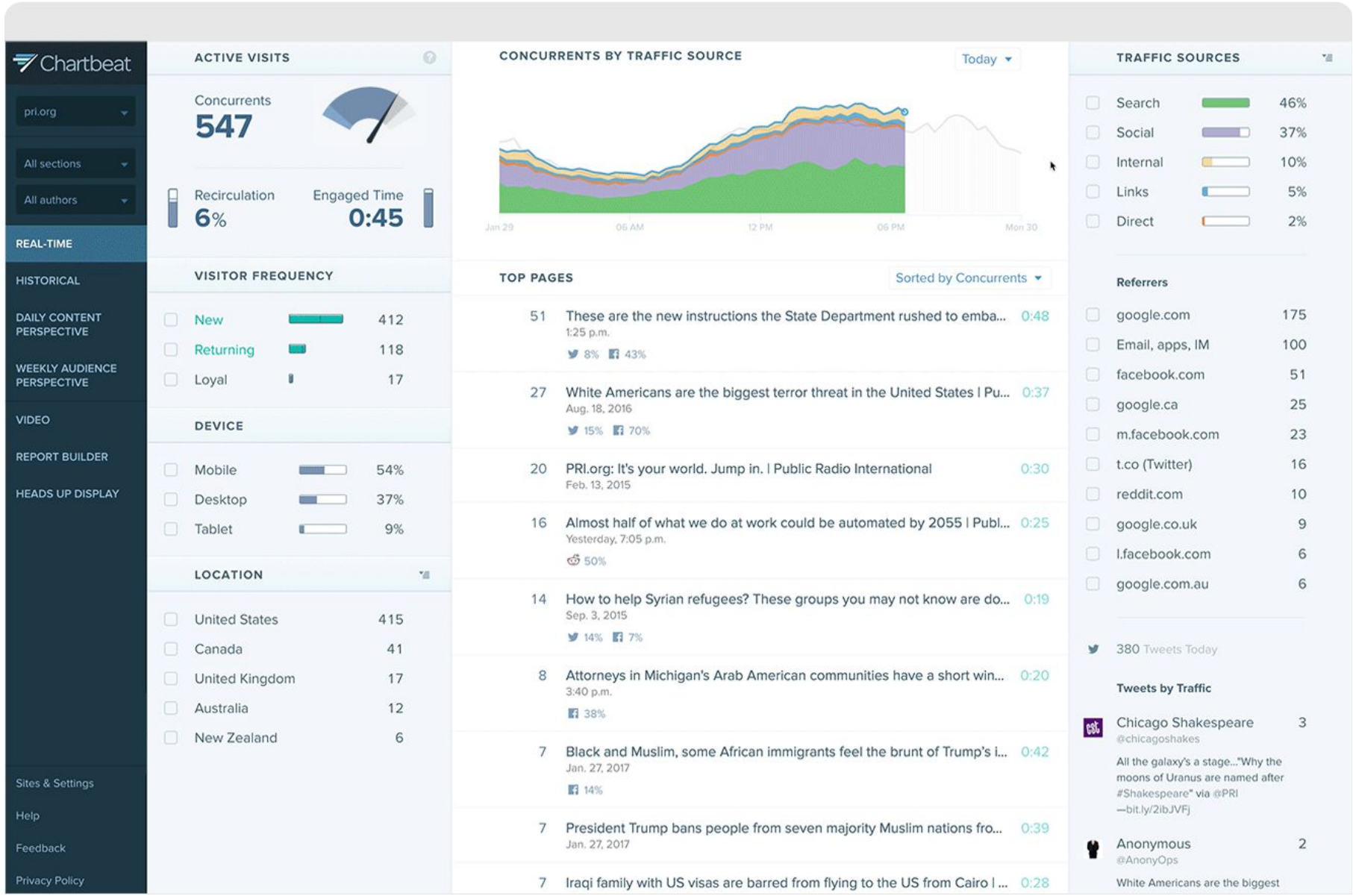
ABOUT US: OUR TEAM

- 75 employees
- 8 year old, VC backed startup
- 20-ish engineers
- 5 Platform/DevOps engineers
- Office in NYC
- Hosted on AWS
- Every engineer pushes code. Frequently

What does Chartbeat do?

Dashboards

- Real Time
- Historical
- Video



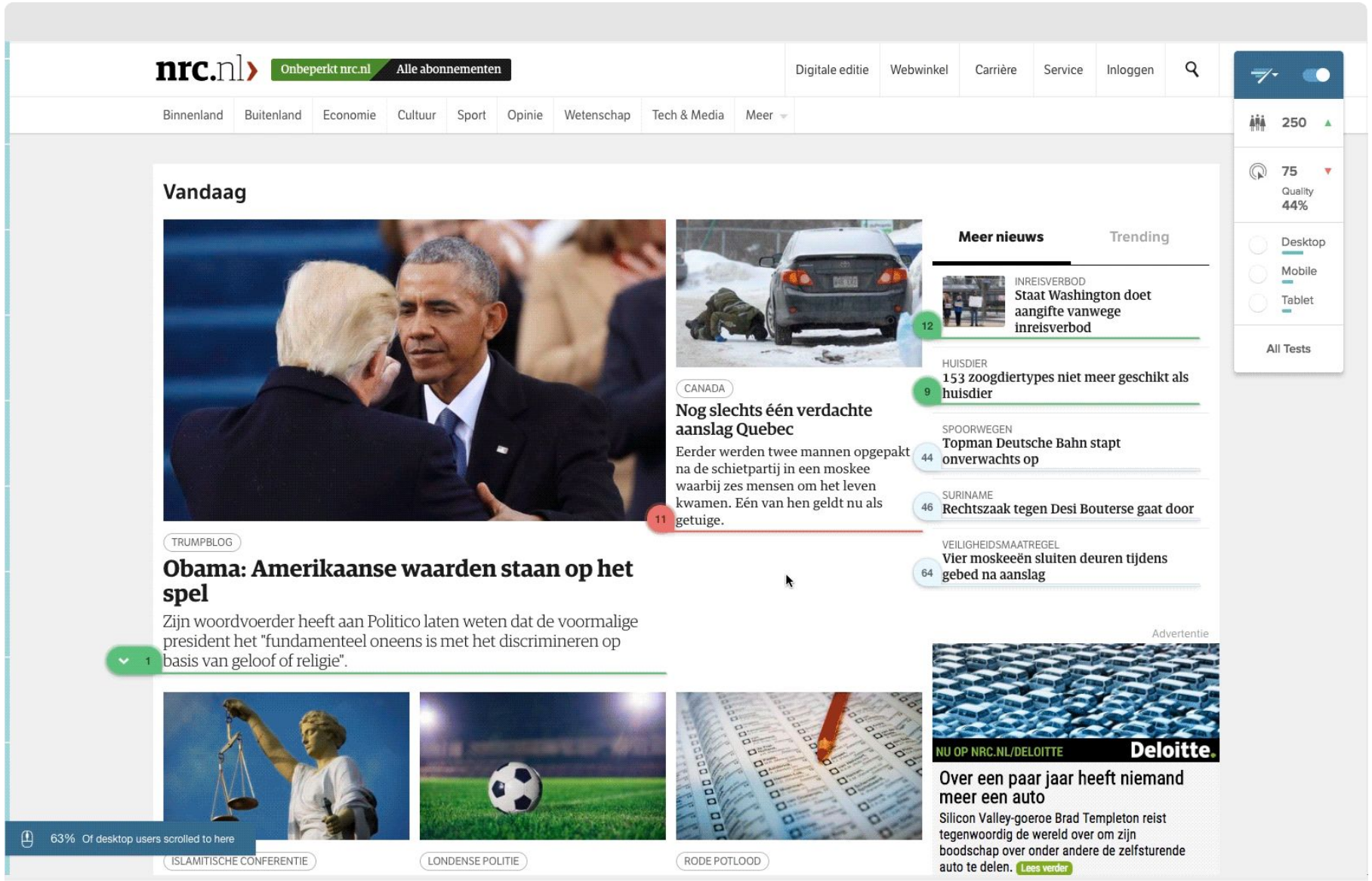
What does Chartbeat do?

Optimization

- Heads Up Display
- Headline Testing

Reporting

- Automated Reports
- Advanced Querying
- APIs



We Get a Lot of Traffic.

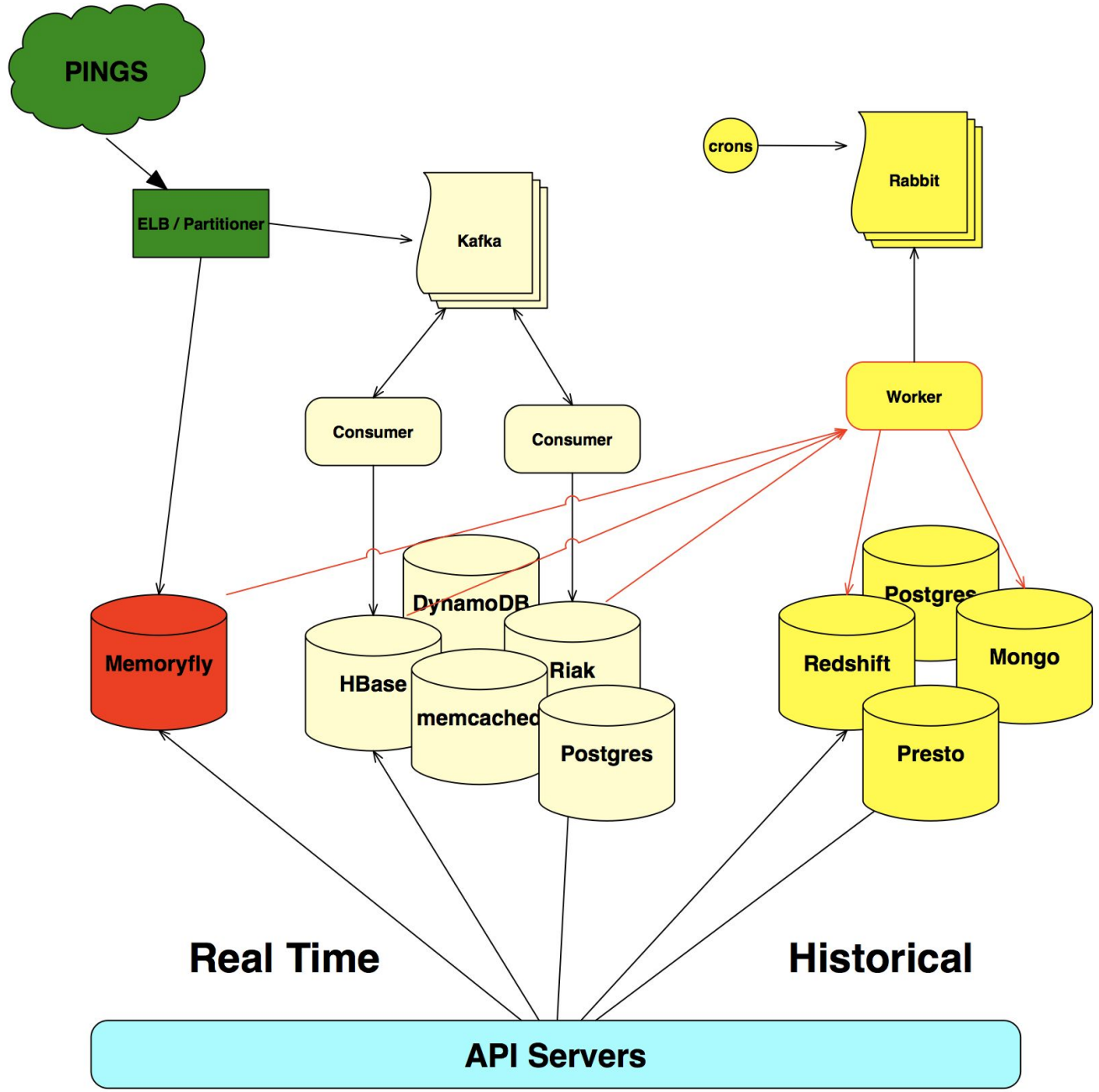
Some #BigData Numbers

Sites Using Chartbeat	Pings/Sec	Tracked Pageviews/Month
50k+	300K	50B

Our Stack

Most of the code is python, clojure or C

It's not all pretty, but we love it.



Why Mesos?
Why Now?

GOALS OF THE PROJECT

Freedom to innovate is the result of a successful product.

Setting ourselves up for the next 5 years.

Goals

- Reduce server footprint
- Provide faster & more reliable services to customers
- Migrate most jobs in a year
- Make life better for engineering team
- Currently - 1200 cores in our cluster, almost all jobs migrated

Happy Engineers?

Happy engineers are productive engineers.

They like:

- Uneventful on-call rotations
- Quick and easy pushes to production
- Easy to use monitoring and debugging tools
- Fast scaling and configuration of jobs
- Writing product code and not messing with DevOps stuff
- Self Service DevOps that’s easy to use

... to build an efficient, effective, and secure development platform for Chartbeat engineers.

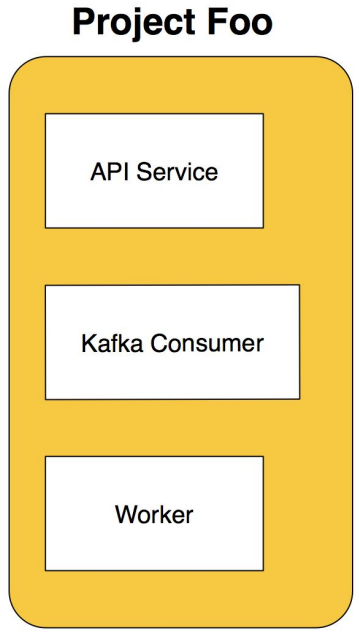
We believe an efficient and effective development platform leads to fast execution.

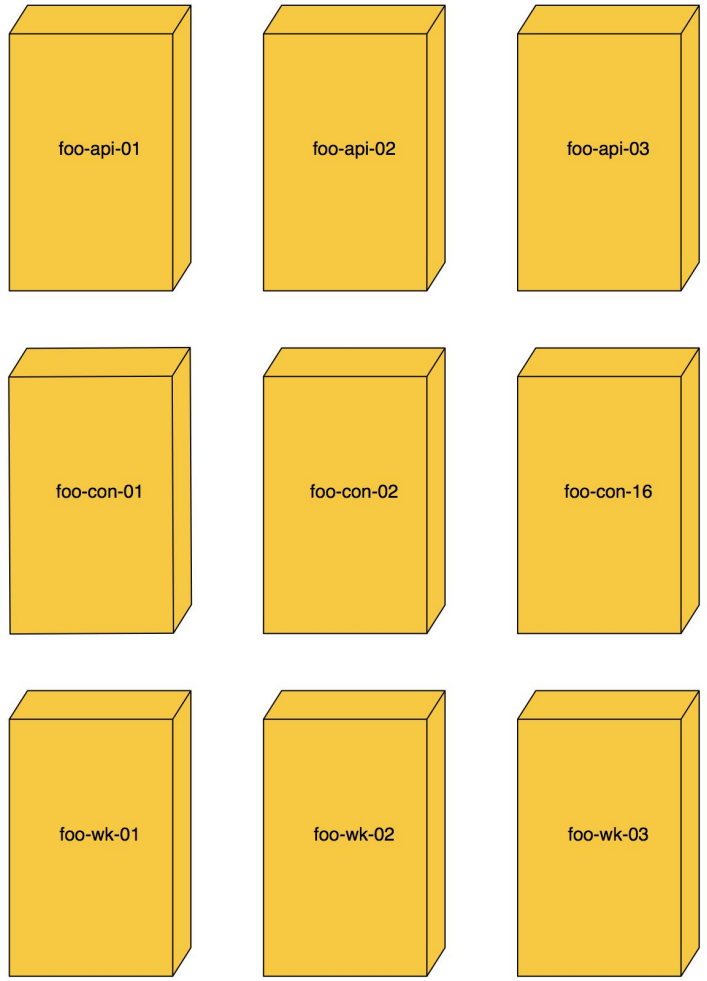
Platform Team Mission Statement

Source: *Platform Team V2MOM, OKR, KPI or some such document c. 2017*

Before Mesos there was Puppet*

- Hiera roles -> AWS tag
- virtual_env -> .deb
- Mostly single purpose servers
- Fabric based DevOps CRUD
- Flexible, but complicated



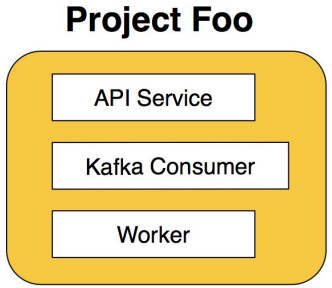


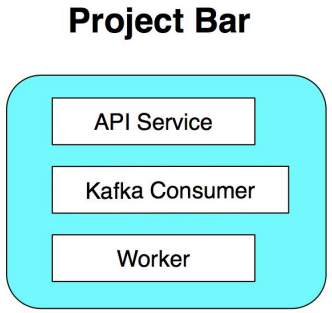
*We still use puppet to manage our mesos servers :-)

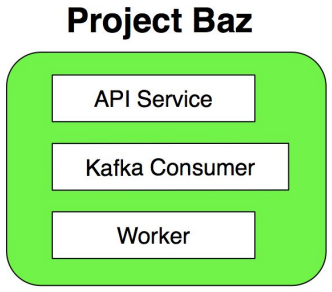
Which “scales” like this

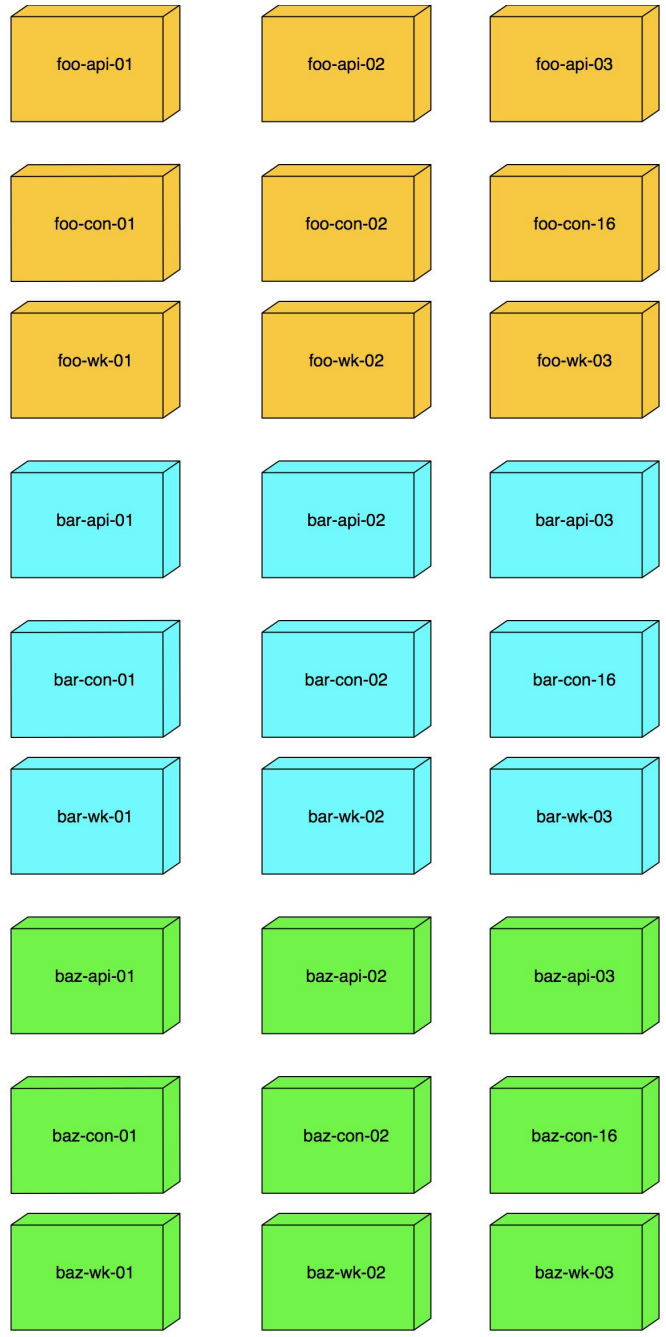
- Jan 2016: 773 EC2 Instances*
- 125 Different Roles
- Hard on DevOps
- Confusing for Product Engineers
- Wasted Resources
- Slow to Scale

* Today we have about 500









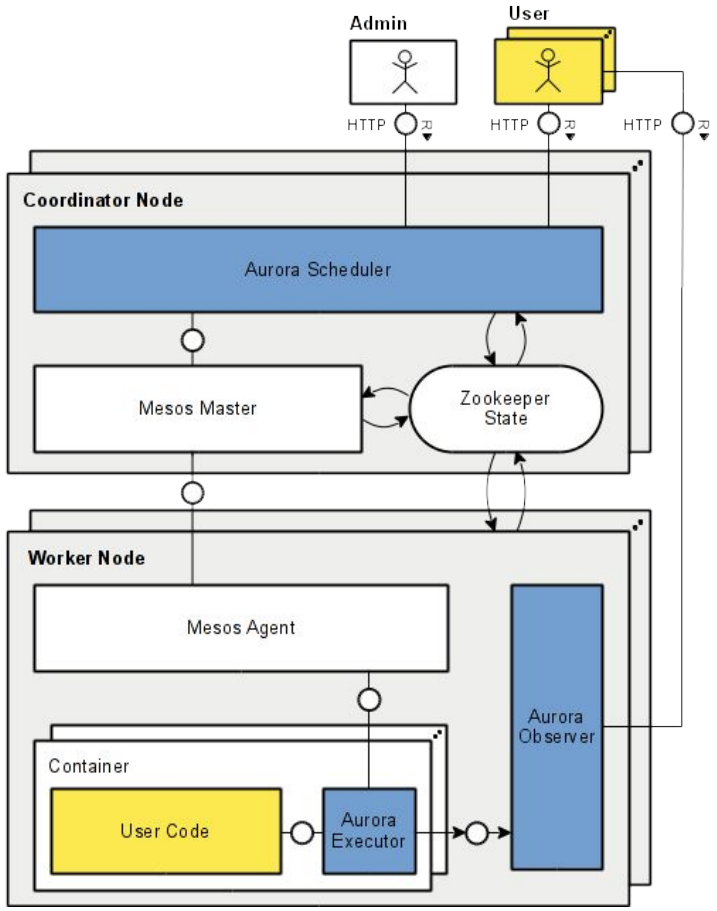
Whatever solution we choose must...

- Allow us to solve python dependency management for once and for all
- Play nicely with our current workflow and be hackable
- Be OSS and supported by an active community using the product irl
- Allow us to migrate jobs safely and over time
- Make our engineers happy

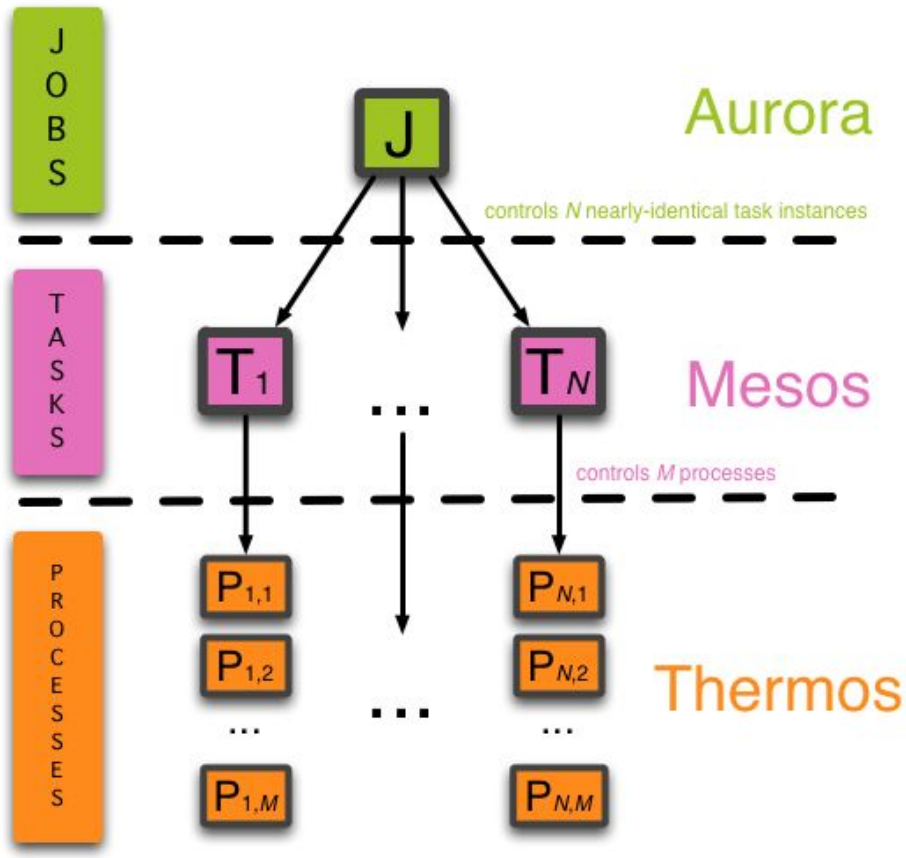
We Chose Aurora

This talk will not be about that decision vs other mesos frameworks.
Read my blog post or let's grab a beer later.

Aurora in a Nutshell



Components



Jobs / Tasks and Processes

Aurora User Features

an incomplete list of ones we have found useful

- Job Templating in Python
- Support for Crons and Long Running Jobs - Autorecovery!
- Hackable CLI for Job Management
- Service Discovery through Zookeeper
- Flexible Port Mapping
- Rich API for Monitoring
- Job Organization and Quotas by User/Environment/Job

Aurora Hello World

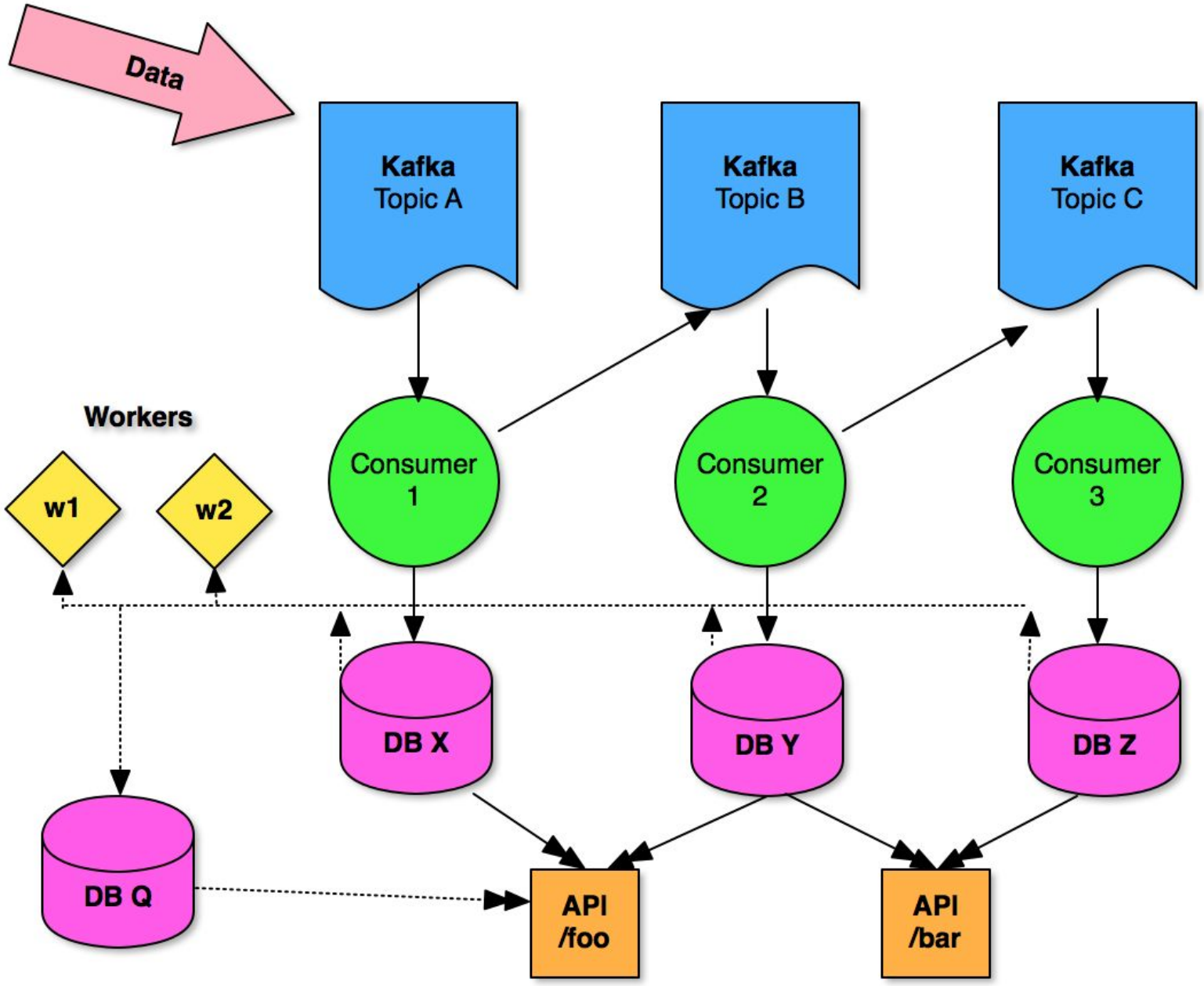
```
pkg_path = '/vagrant/hello_world.py'
import hashlib
with open(pkg_path, 'rb') as f:
  pkg_checksum = hashlib.md5(f.read()).hexdigest()

# copy hello_world.py into the local sandbox
install = Process(
  name = 'fetch_package',
  cmdline = 'cp %s . && echo %s && chmod +x hello_world.py'  % (pkg_path, pkg_checksum))


# run the script
hello_world = Process(
  name = 'hello_world',
  cmdline = 'python -u hello_world.py' )


# describe the task
hello_world_task = SequentialTask(
  processes = [install, hello_world],
  resources = Resources(cpu = 1, ram = 1*MB, disk=8*MB))


jobs = [
  Service(cluster = 'devcluster',environment = 'devel', role = 'www-data', name = 'hello_world', task =
hello_world_task)]
```

- Processes run unix commands
- Tasks are pipelines of processes
- A Job binds it all together

Take a step back and understand the problem you're trying to solve

It turns out that the vast majority of our jobs follow one of 3 patterns:

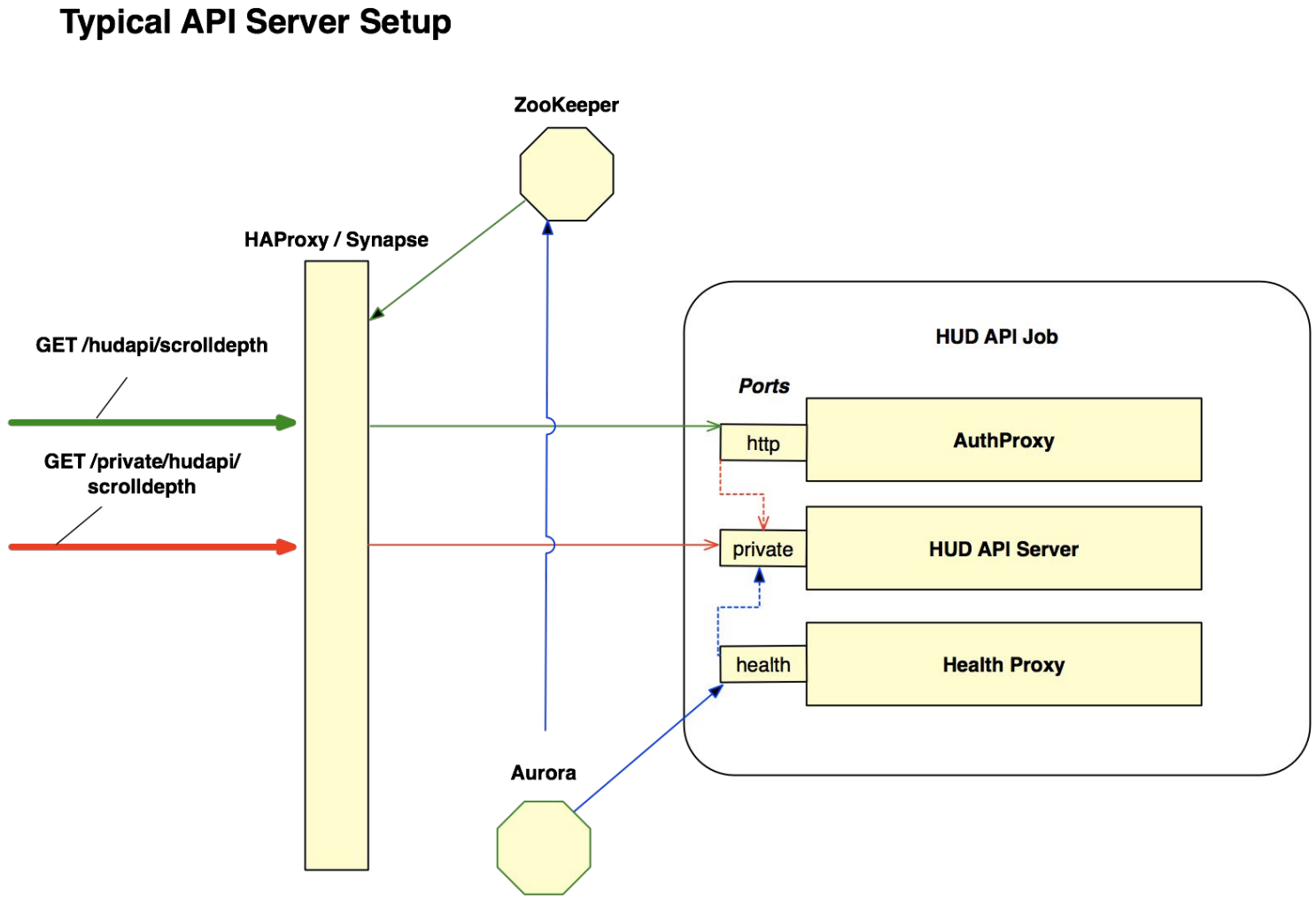
1. a clojure kafka consumer
2. a python worker
3. a python api server



Good DevOps is a Balance Between Flexibility and Reliability and Sometimes it Takes a Lot of Work

Our API Servers follow this pattern:

1. AuthProxy bound on HTTP Port
2. API Server Bound on Private Port
3. Some Health Check Bound on Health Port



How do We Integrate Aurora With Our Workflow?

INTEGRATE WITH OUR WORKFLOW

what does our workflow feel like?

- git is source of truth for code and configurations
- Deployed code tagged with git hash
- Individual projects can run in prod / dev / local environments
- Do everything from the command line
- Prefer writing scripts to memorizing commands
- Don’t reinvent things that work - Make templates for common tasks

We will encourage you to develop the three great virtues of a programmer: laziness, impatience, and hubris.

Larry Wall, Programming Perl

Source:

wiki.c2.com/?LazinessImpatienceHubris

Major Decision Time

1. Adopt Pants
2. Wrap Aurora CLI with our own client
3. Create a library of Aurora templates
4. Let Aurora keep jobs running and disks clean
5. Dive in and embrace sandboxes for isolation

Step 1. Make Aurora Fit In

Our Aurora Wrapper

- Separate common config options from aurora configs into <job>.yaml file
- Require versioned artifacts built by CI server to deploy
  - Require git master to push to prod
- 1 to 1 mapping between yaml file and job (prod or dev)
  - Many to 1 mapping between yaml file and aurora configs
- Allow for job command line options to be set in yaml
- All configs live in single directory in repo - easy to find jobs
- Additional functionality for things like tailing output from running jobs

Aurora CLI

Start a job named aa/cbops/prod/fooserver defined in ./aurora-jobs/fooserver.aurora:

Aurora:

```
> aurora create aa/cbops/prod/fooserver ./aurora-jobs/fooserver.aurora
```

Chartbeat:

```
> aurora-manage create fooserver --stage=prod
```

1. All configs are in one location
2. Production deploys require explicit flag
3. Consistent mapping between job name and config file(s)
4. All aurora client commands use aurora-manage wrapper

Aurora + YAML - eightball.yaml

info about the job and build artifact

Resource requirements

```
file: eightball
user: cbe
buildname: eightball
hashtype: git
config:
 cpu: 0.25
 num_instances: 1
 ram: 300
 disk: 5000
taskargs:
 workers: 10
envs:
 prod:
   cpu: 1.5
   num_instances: 12
   taskargs:
     workers: 34
   githash: ABC123
 devel:
   githash: XYZ456
```

Options for use in aurora template

Stage specific overrides

githash of artifact being deployed. Can be top level as well.

Step 2: Write Templates

CUSTOM AURORA TEMPLATES

Python modules to generate aurora templates for common use cases:

- Artifact installers (jars, tars, pex’es)
- JVM/JMX/Logging configs
- General environment configs and setups
- Local dynamic config file creation
- Access credentials to shared resources (DBs, ZKs, Kafka brokers, etc.)
- Common supporting tasks (AuthProxy, Health Checkers)

Aurora + YAML - eightball.aurora

setup pystachio

```
PROFILE = make_profile()
PEX_PROFILE = make_pexprofile('eightball')
SERVICES = get_service_struct()

install_pex = pex_install_template
```

options to job process

```
opts = {
  '--port': '{{thermos.ports[private]}}',
  '--memcache_servers':'{{services.[memcache]}}',
  '--workers={{profile.taskargs[CB_TASK_WORKERS]}}
'
  '--logstash_format': 'True'
}
```

server process

```
run_server = Process(
  name='eightball',
  cmdline=make_cmdline('./{{pex.pexfile}}
server',opts)
)
```

get helper processes

```
auth_proxy_processes= get_authproxy_processes()

health_check_processes= get_proxy_hc_processes(
      url="/private/stats/",
port_name='private')
```

generate correctly ordered processes

```
MAIN = make_main_template(
  ([install_eightball, eightball_server],

auth_proxy_processes,health_check_processes,),
res=resources_template)
```

Apply templates and run

```
jobs = [
   job_template(task=MAIN,
       health_check_config =
health_check_config,
       update_config = update_config
   ).bind(pex=PEX_PROFILE,
          profile=PROFILE,
          services=SERVICES)
]
```

Aurora Templates++

Most workers are built off of the same python framework.

Each job gets its own git-hash named pex file with its specific dependencies.

Command line arguments determine the work to be done.

Engineers simply define their worker jobs in a few lines of yaml

Engineers are happy

```
groot in ~/chartbeat/aurora/configs

± |master {1} ?:2 ✗| → ls igor_worker.aurora

igor_worker.aurora

± |master {1} ?:2 ✗| → grep igor_worker *.yaml|wc -l

     104

± |master {1} ?:2 ✗| → grep igor_worker *.yaml|head -n 3

content_es_article_index.yaml:file: igor_worker

content_es_cluster_maintenance.yaml:file: igor_worker

content_es_fill_storyid.yaml:file: igor_worker
```

bb/cbp/prod/content_es_fill_storyid and bb/cbp/devel/content_es_fill_storyid

CUSTOM AURORA TEMPLATES+++

Our new ETL pipeline “Deep Water”

- Steps defined in python classes
- Each step receives a set of independent aurora jobs (defined in yaml)
- Pipeline state stored in Postgres for consistency

Non-Mesos Components

Before deploying anything, we needed solutions for the following

- Build, Packaging & Deployment
- Request Routing
- Metrics / Monitoring
- Logfile Collection & Analysis
- Configuration Management
- Probably some other stuff

Question #1: Build, Packaging & Deployment

We like our git mono-repo / Jenkins workflow
Can we make this work for python dependencies?

Actually we really don't like virtualenv that much...

Answer: Yes.
Put on your pants

Pants in one slide

A build system for big repos, especially python ones

- pantsbuild.io

- Maven for Python (and Java...)
- Creates PEX files with dependencies bundled in (3rd party and intra-repo)
- Directory level BUILD files
- Incremental builds in mono-repo
- Artifacts can include git-hash in filename
- No more repo level dependency conflicts
- Happens to be how Aurora is built :-)
- Huge migration effort, huge benefits

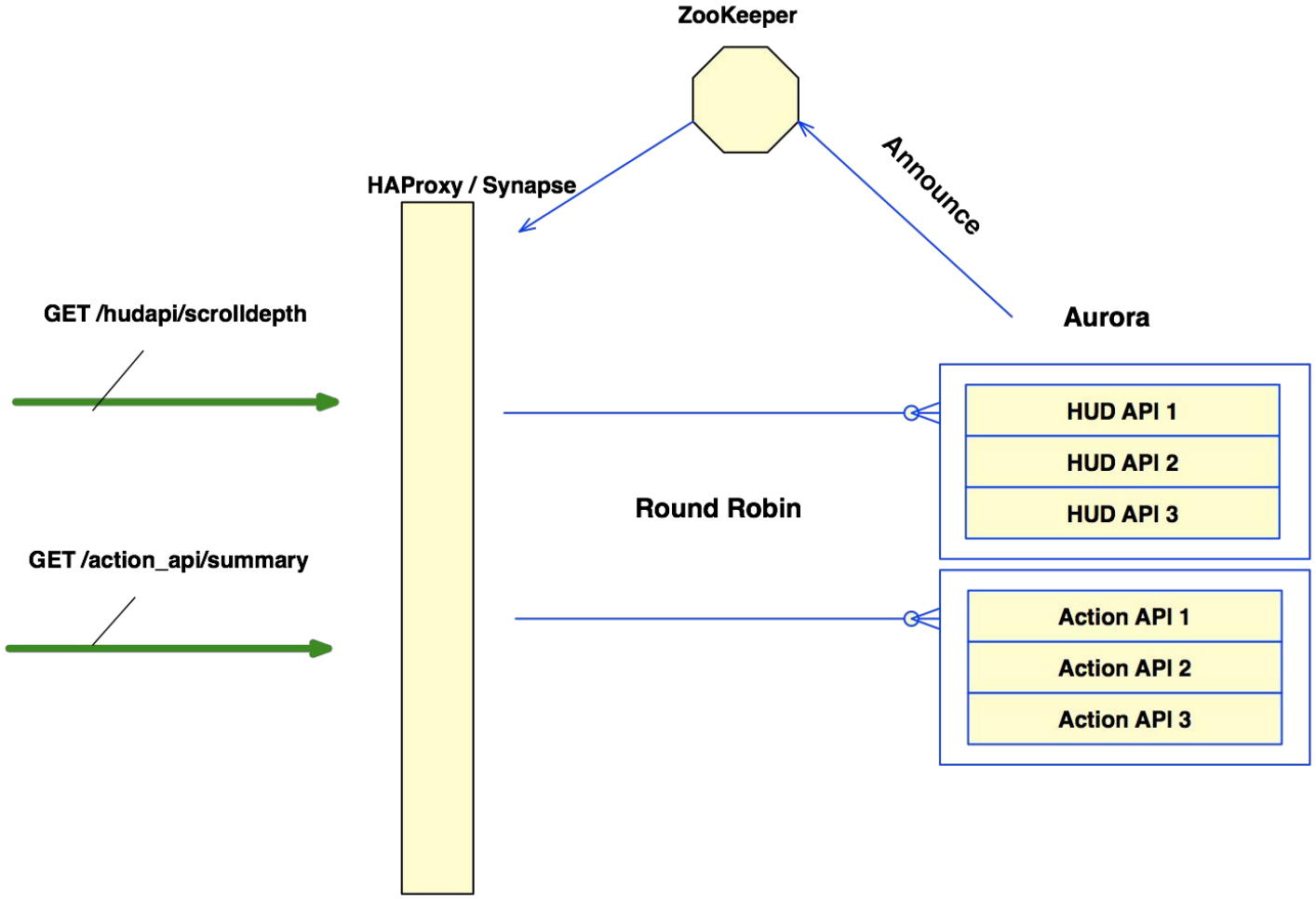
Question #2: Routing

How are we going to route traffic as jobs move around the cluster?

Answer:
HAProxy & Synapse

Synapse in a Nutshell

- Config is yaml superset of HAProxy config
- Aurora updates zookeeper with list of task/port mappings
- Synapse discovers service changes in zk and updates HAProxy
- Synapse generates HAProxy config
- Puppet pushes synapse changes to HAProxy servers



https://github.com/airbnb/synapse

Question #3: Metric Collection, Reporting and Monitoring

Can we easily collect metrics for all of our jobs? It's kinda ad-hoc now.

Answer:
Consolidate on:
OpenTSDB + Grafana

How We Collect and Report Metrics

OpenTSDB -> Grafana / Nagios -> PagerDuty

- Consistent job naming makes everything easier

- Automatic collection of aurora job resource utilization
- Automatic collection of HAProxy metrics
- Libraries for python/clojure auto tag TSDB metrics with job info
- Custom JMX collector pulls metrics from JVM jobs
  - Discovers jobs in ZK just like Synapse
- Grafana dashboards for all
- Nagios -> Pagerduty alerting
  - most simple failures are just restarted by aurora!

Question #3: Logfile Analysis

Users like to ssh and tail. How do we make that easy for them?

Answer:
Flume / Athena and tailll

How We Read Log Files
It turns out log file aggregation is hard

We didn’t like ELK

- Users want “polysh” - aurora-manage tailll <jobname>
- Aurora Web UI allows “checking” on logs
- Aurora CLI allows ssh to a single instance
- Flume -> S3 -> Athena for historical forensics
- Don’t rotate logs - let aurora kill sandboxes that fill up disk is cheap

SUMMARY

Almost 2 years later - we couldn’t be happier

- Huge reduction in frequency of “on-call events”
- Reduced EC2 instance costs by 1/3
- Engineers survey shows they “rarely” experience blockers deploying
- Changed our entire approach to DevOps and architecture

Thank you.

Rick Mangi
rick@chartbeat.com
@rmangi

medium.com/chartbeat-engineering