



Evan Krall
2015-06-12



Who is this person?

- Evan Krall
- SRE = development + sysadmin
- 4+ years at Yelp





Connecting people with great local businesses

Founded in 2004



Jeremy Stoppelman

Review Distribution



Around the World

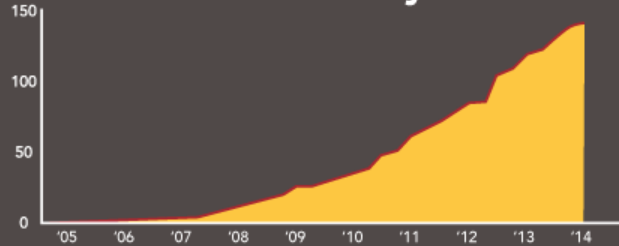
We're operating in 29 locales



72 million via mobile



135 million monthly visitors



You can find out more at yelp.com/careers

Paasta

Application Delivery at Yelp



Why?



History

- Yelp started out as monolithic python app
- Builds/pushes take a long time
- Messing up is painful
- So we build process to avoid messing up
- Process makes pushes even slower



Service Oriented Architecture

- Pull features out of monolith
- Split into different applications
- Smaller services -> faster pushes
 - fewer issues per push
 - total # of issues increases,
 - but we can fix issues faster.
- Smaller parts -> easier to reason about
- Bonus: can scale parts independently



SOA comes with challenges

- Lots of services means lots of dependencies
- Now your code is a ~distributed system~
- If you thought running 1 app was hard, try 100



What is a service?

- Standalone application
- Stateless
- Separate git repo
- Usually, at Yelp:
 - HTTP API
 - Python, Pyramid, uWSGI
 - virtualenv



Yelp SOA before Paasta

- services responsible for providing init script
 - often not idempotent
- central list of which hosts run which services
- pull-model file transfers
 - reasonably reliable
- push-model control (for host in hosts: ssh host ...)
 - What if hosts are down?
 - What if transfer hasn't completed yet?



What is
Paasta?



What is Paasta?

Internal PaaS

- Builds services
- Deploys services
- Interconnects services
- Monitors services



What is Paasta?

Deploying services should be better!

Servers are not snowflakes!

Continuous integration is awesome!

Declarative control++



Monitor your services!

Design goals



Make ops happy

- Fault tolerance
 - no single points of failure
 - recover from failures
- Efficient use of resources
- Simplify adding/removing resources



Make devs happy

- We need adoption, but can't impose on devs
- Must be possible to seamlessly port services
 - Must work in both datacenter and AWS
- Must be compelling to developers
 - Features
 - Documentation
 - Flexibility



Make ourselves happy

(paasta devs)

- Pick good abstractions
- Avoid hacks
- Write good tests
- Don't reinvent the wheel: use open-source tools
- Enforce opinions when necessary for scale



How



What runs in production?

(or stage, or dev, or ...)



What parts do we need?

- Scheduling: Decide where to run the code
- Delivery: Get the code + dependencies onto boxes
- Discovery: Tell clients where to connect
- Alerting: Tell humans when things are wrong



What parts do we need?

- Scheduling: Decide where to run the code
- Delivery: Get the code + dependencies onto boxes
- Discovery: Tell clients where to connect
- Alerting: Tell humans when things are wrong



Scheduling:

Decide where to run the code

- Static: humans decide
 - puppet/chef: role X gets service Y
 - static mappings: boxes [A,B,C,...] get service Y
- simple, reliable
- slow to react to failure, resource changes



Scheduling:

Decide where to run the code

- Dynamic: computers decide
 - Mesos, Kubernetes, Fleet, Swarm
 - IaaS: dedicated VMs for service, let Amazon figure it out.
- Automates around failure, resource changes
- Makes discovery/delivery/monitoring harder



Scheduling in Paasta: Mesos + Marathon

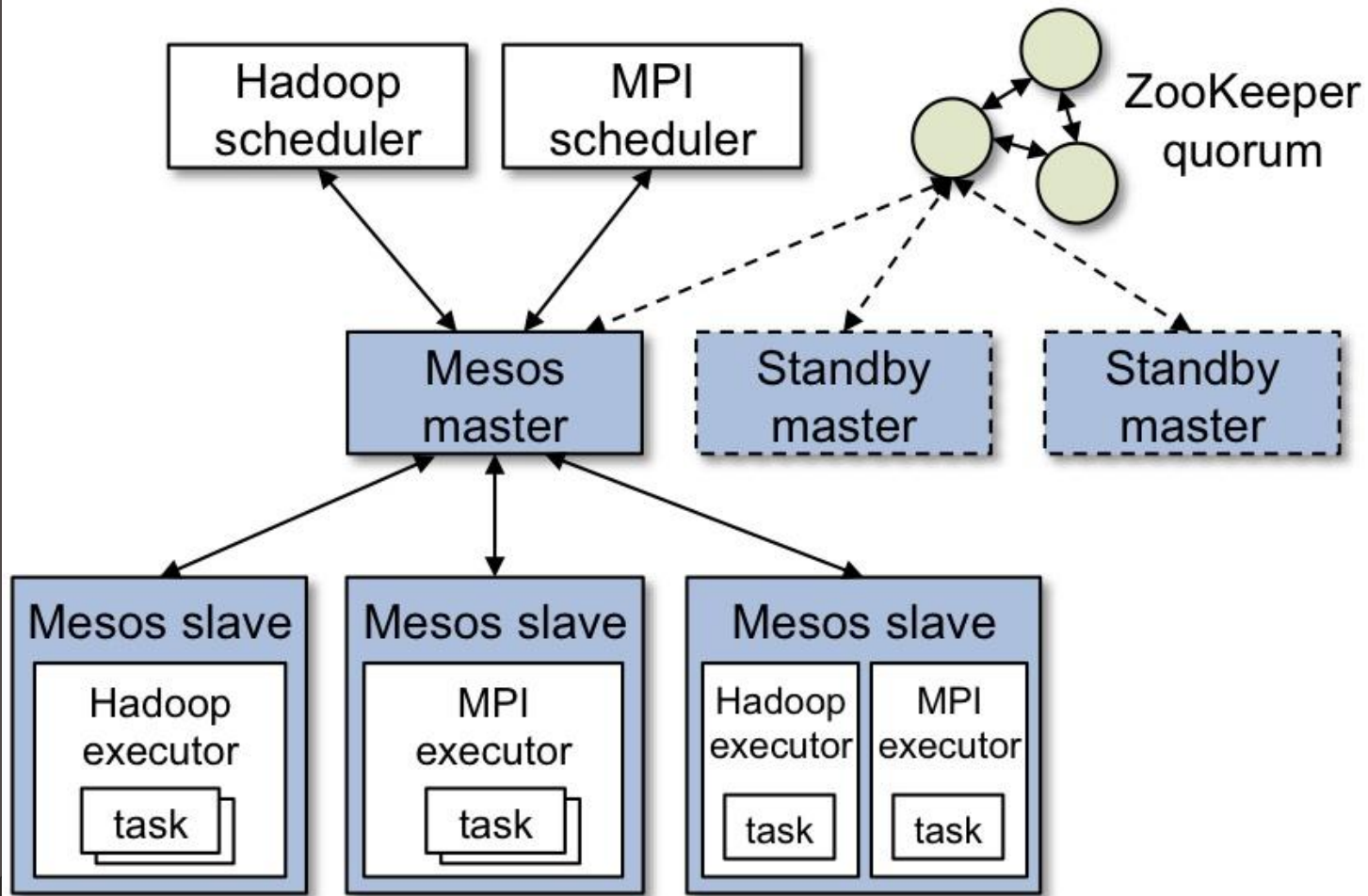
- Mesos is an "SDK for distributed systems", not batteries-included.
- Requires a framework
 - Marathon (ASGs for Mesos)
- Can run many frameworks on same cluster
- Supports Docker as task executor

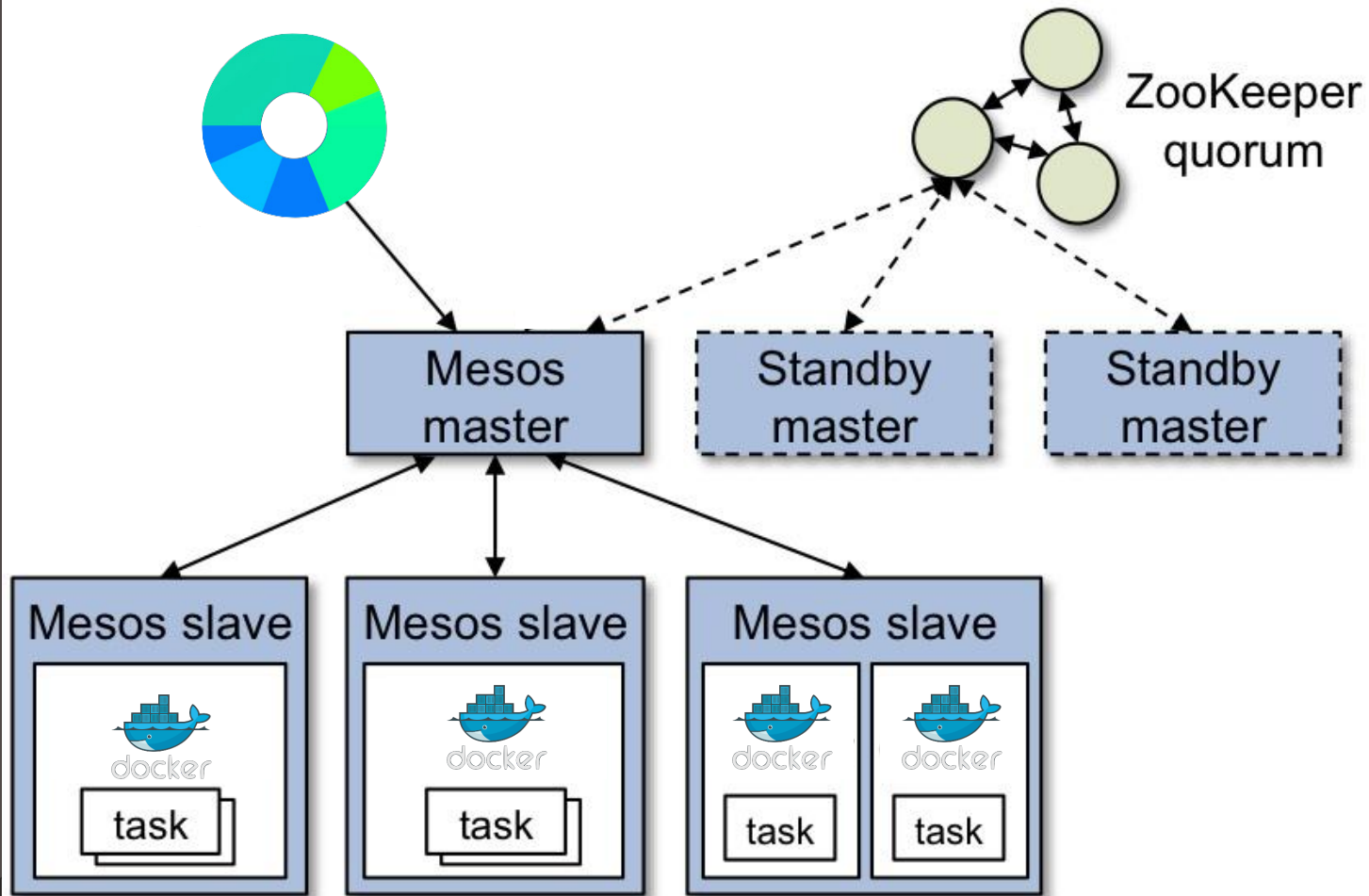
mesosphere.io
mesos.apache.org



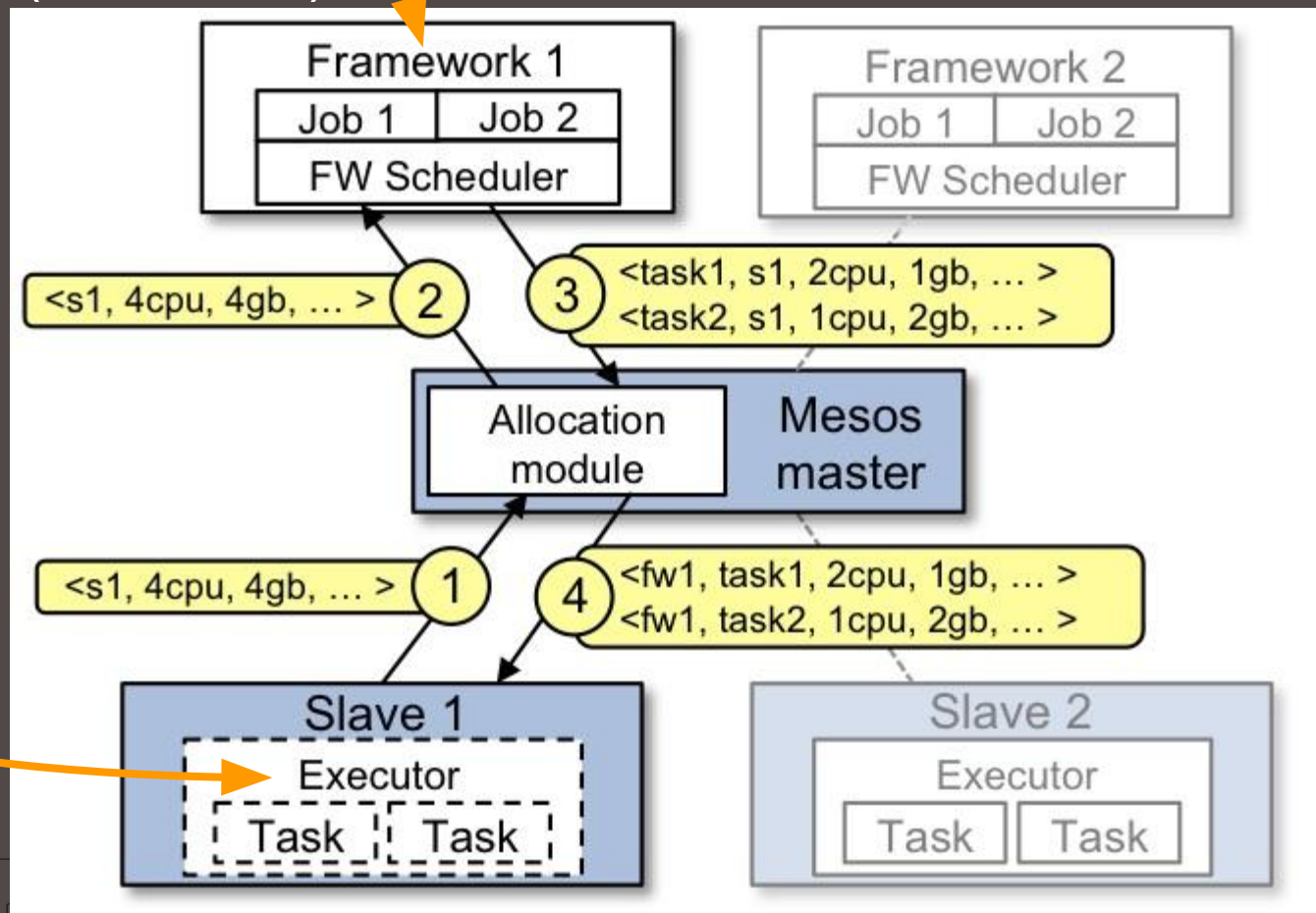
MARATHON







(Marathon)



(Docker)

What parts do we need?

- Scheduling: Decide where to run the code
- Delivery: Get the code + dependencies onto boxes
- Discovery: Tell clients where to connect
- Alerting: Tell humans when things are wrong



Delivery:

Get the code + dependencies onto boxes

- Push-based:
 - `for box in $boxes; do rsync code $box:/code`
 - Simple, easy to tell when finished
 - what about failures?
 - retry, but how long?
 - how do we make sure new boxes get code?
 - cron deploys



Delivery:

Get the code + dependencies onto boxes

- Pull-based:
 - cron job on every box downloads code
 - built-in retries
 - new boxes download soon after boot
 - have to wait for cron job
 - baked VM/container images
 - container/VM can't start on failure
 - ASG, Marathon will retry



Delivery:

Get the code + dependencies onto boxes

- Shared
 - `sudo {gem,pip,apt-get} install`
 - lots of tooling exists already
 - shared = space/bandwidth savings
 - what if two services need different versions?
 - how to update a library that 20 services need?



Delivery:

Get the `code` + dependencies onto boxes

- Isolated
 - virtualenv / rbenv / VM-per-service / Docker
 - more freedom for dev
 - services don't step on each others' toes
 - more disk/bandwidth
 - harder to audit for vulnerabilities



Delivery in Paasta: Docker

- Containers: like lightweight VMs
- Provides a language (Dockerfile) for describing container image
- Reproducible builds (mostly)
- Provides software flexibility



What parts do we need?

- Scheduling: Decide where to run the code
- Delivery: Get the code + dependencies onto boxes
- Discovery: Tell clients where to connect
- Alerting: Tell humans when things are wrong



Discovery:

Tell clients where to connect

- Static:
 - Constants in code
 - Config files
 - Static records in DNS
- Simple, reliable
- Slow reaction time



Discovery:

Tell clients where to connect

- Dynamic:
 - Dynamic DNS zone
 - ELB
 - Zookeeper, Etcd, Consul
 - Store IPs in database, not text files
- Reacts to change faster, allows better scheduling
- Complex, can be fragile
- Recursive: How do you know where ZK is?

Discovery:

Tell clients where to connect

- in-process
 - DNS
 - Everyone supports DNS
 - TTLs are rarely respected, limit update rate
 - Lookups add critical-path latency
 - Talking to ZK, Etcd, Consul in service
 - Tricky. Risk of worker lockup if ZK hangs
 - Delegate to library
 - Few external dependencies

**IF YOU NEED TO DO SERVICE
DISCOVERY MID-REQUEST**



YOU'RE GONNA HAVE BAD TIMINGS

Discovery:

Tell clients where to connect

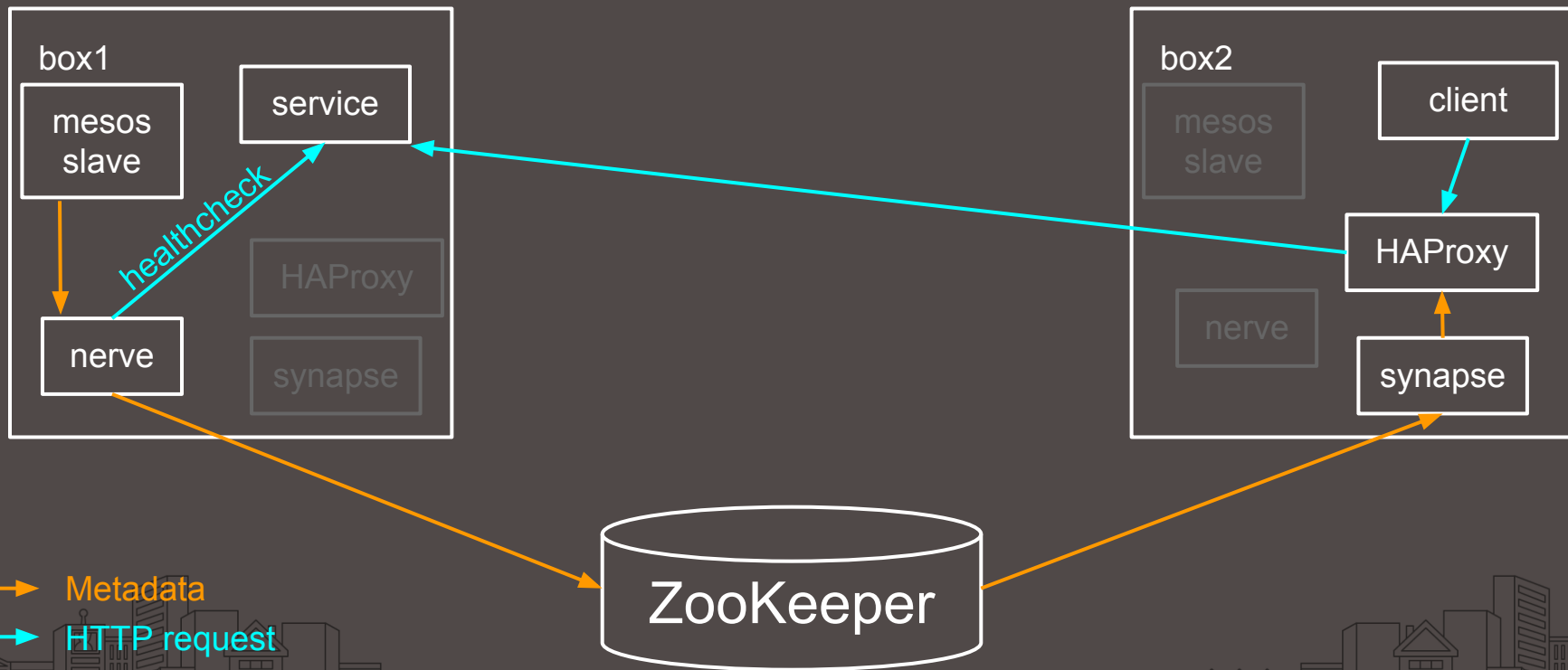
- external
 - SmartStack, consul-template, vulcand
 - Reverse proxy on local box
- Simpler client code (just hit localhost:\$port)
- Avoids library headaches
- Easy cross-language support
- Must be load-balanceable



Discovery in Paasta: Smartstack

- Nerve registers services in ZooKeeper
- Synapse discovers from ZK + writes HAProxy config
- Registration, discovery, load balancing
 - Hard problems! Let's solve them once.
- Provides migration path:
 - port legacy version to SmartStack
 - have Paasta version register in same pool

Discovery in Paasta: Smartstack

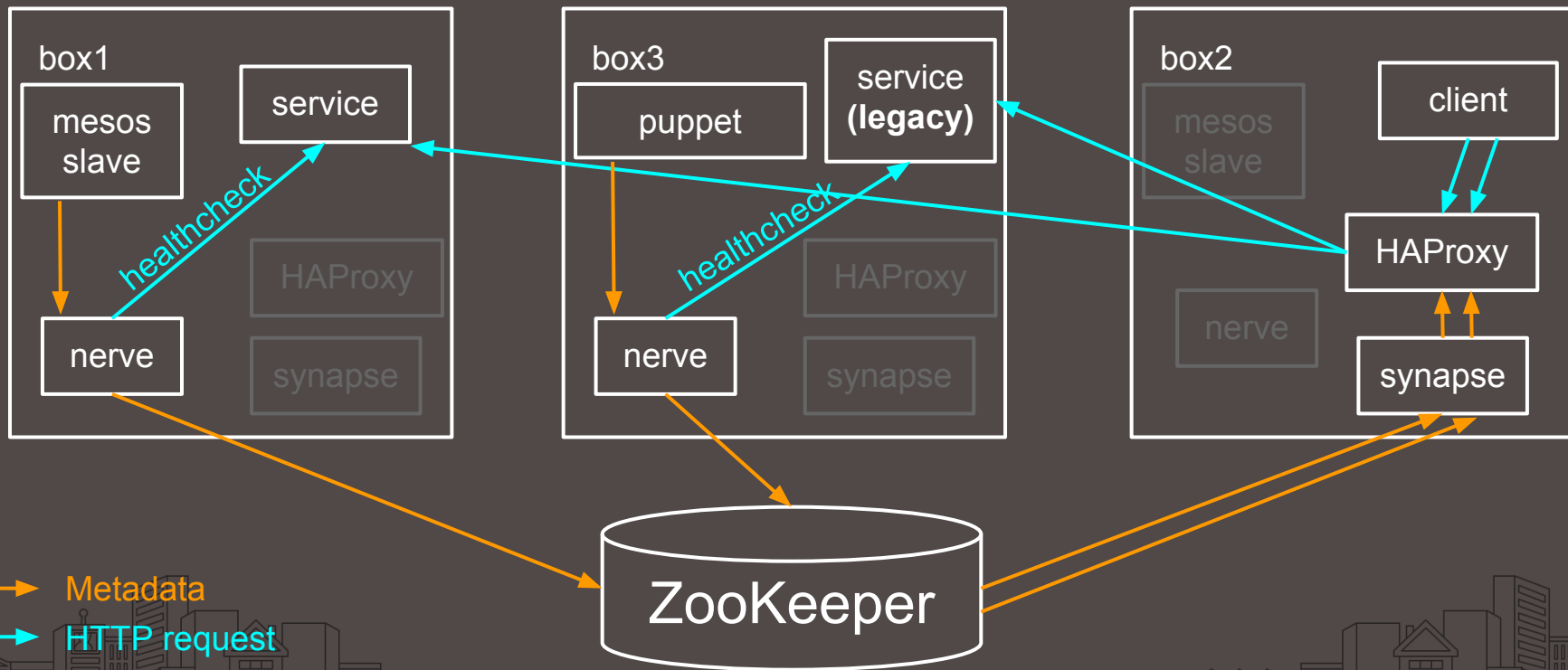


why bother with registration?
why not ask your scheduler?

Scheduler portability!



Discovery in Paasta: Smartstack

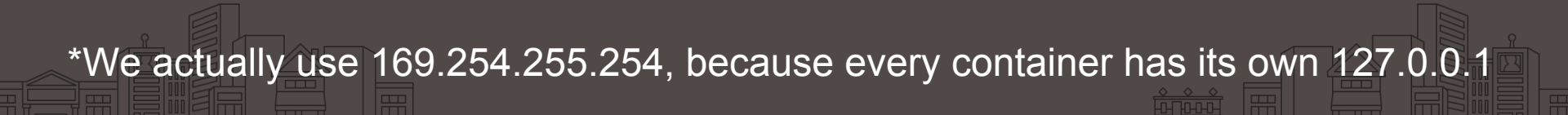


There's no place like 127.0.0.1*

- Every box runs HAProxy
- Paper over network issues with retries
- Load balancing scales with # of clients

- Downside: lots of healthchecks
 - hacheck caches to avoid hammering services
- Downside: many LBs means LB algorithms don't work as well

*We actually use 169.254.255.254, because every container has its own 127.0.0.1



What parts do we need?

- Scheduling: Decide where to run the code
- Delivery: Get the code + dependencies onto boxes
- Discovery: Tell clients where to connect
- Alerting: Tell humans when things are wrong



Alerting:

Tell humans when things are wrong

- Static
 - E.g. nagios, icinga
 - File-based configuration
- Simple, familiar
- Often not highly available
- Hard to dynamically generate checks/alerts



Alerting:

Tell humans when things are wrong

- Dynamic
 - e.g. Sensu, Consul
 - Allows you to add hosts & checks on the fly
- Flexible
- Generally newer, less battle-tested
- Newer software is often built for high availability



Alerting in Paasta: Sensu



- Based around event bus
- Replication monitoring
 - how many instances are up in HAProxy?
- Marathon app monitoring
 - is service failing to start?
- Cron jobs on master boxes do checks, emit results.

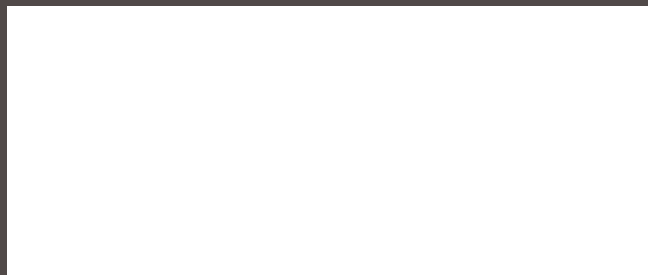
Runtime Components

- Scheduling: Mesos+Marathon
- Delivery: Docker
- Discovery: SmartStack
- Alerting: Sensu



How do we
control this thing?





- Primary control plane
- Convenient access controls (via gitolite, etc)
- deploys, stop/start/restart indicated by tags
- less-frequently changed metadata stored in a repo



Declarative control

- Describe **end goal**, not **path**
- Helps us achieve fault tolerance.

"Deploy 12abcd34 to prod"

vs.

"Commit 12abcd34 should be running in prod"

Gas pedal vs. Cruise Control



metadata repo

- editable by service authors
- `marathon-$cluster.yaml`
 - how many instances to run?
 - canary
 - secondary tasks
- `smartstack.yaml`
- `deploy.yaml`
 - list of deploy steps
- Boilerplate can be generated with `paasta fsm`



paasta_tools

- python 2.7 package, dh-virtualenv
- CLI for users
 - control + visibility
- cron jobs:
 - Collect information from git
 - Configure Marathon
 - Configure Nerve
 - Resilient to failure
- This is how we build higher-order systems



Bounce strategies

- up-then-down:
 - wait for new version to start; kill old
- down-then-up
 - wait for old version to die; start new
- crossover
 - as instances of new version start, kill old instances



Jenkins

- Builds Docker images
- Pushes to Docker registry
- Marks image for deployment

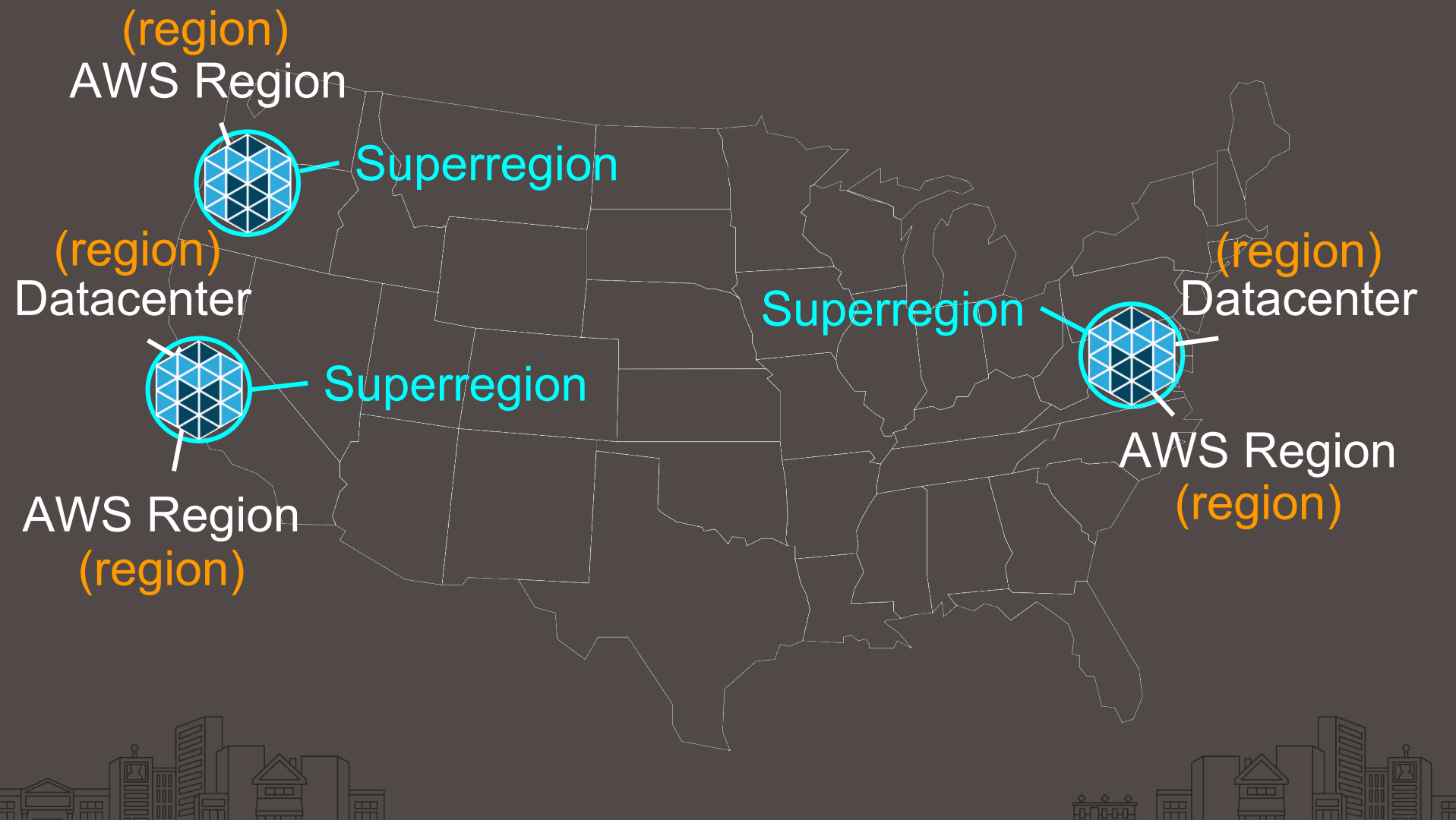
- GUI configuration is a Bad Idea, so we automate it (deploy.yaml)
- Most build steps call paasta command



Multi-Region Datacenter Environment

habitat	AZ or cage	0.3ms
region	AWS region, nearby cages	1ms
superregion	nearby regions	<5ms
ecosystem	copy of site (dev/stage/prod)	

- Mesos cluster per superregion
- Services choose at which level SmartStack works



Walkthrough

Let's deploy a service



```
krall@dev4-devc:~/pg$ mkdir paasta_demo
krall@dev4-devc:~/pg$ cd paasta_demo/
krall@dev4-devc:~/pg/paasta_demo$ git init
Initialized empty Git repository in /nail/home/krall/pg/paasta_demo/.git/
krall@dev4-devc:~/pg/paasta_demo (master) $ cat > Dockerfile
FROM docker-dev.yelpcorp.com/trusty_yelp

RUN apt-get -y --force-yes install python3 wget
RUN wget -r html5zombo.com

CMD cd /html5zombo.com && python3 -m http.server 9999

EXPOSE 9999
krall@dev4-devc:~/pg/paasta_demo (master) $ cat > Makefile
DOCKER_TAG ?= $(USER)-dev

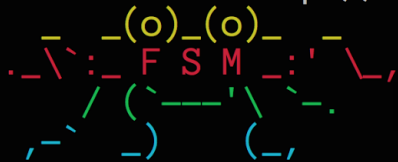
itest:
    docker build -t $(DOCKER_TAG) .
krall@dev4-devc:~/pg/paasta_demo (master) $ git add Dockerfile Makefile
krall@dev4-devc:~/pg/paasta_demo (master) $ git commit -m "initial commit"
[master (root-commit) cedd0b8] initial commit
2 files changed, 13 insertions(+)
create mode 100644 Dockerfile
create mode 100644 Makefile
krall@dev4-devc:~/pg/paasta_demo (master) $
```



```
krall@dev4-devc:~/pg/paasta_demo (master) $ paasta check
X Failed to locate yelpsoa-config directory for paasta_demo.
  Please follow the guide linked below to get boilerplate. http://y/paasta-deploy
X No deploy.yaml exists, so your service cannot be deployed.
  Push a deploy.yaml and run `paasta generate-pipeline`.
  More info: http://y/yelpsoa-configs
X No 'security-check' entry was found in your deploy.yaml.
  Please add a security-check entry *AFTER* the itest entry in deploy.yaml
  so your docker image can be checked against known security vulnerabilities.
  More info: http://servicedocs.yelpcorp.com/docs/paasta\_tools/paasta\_cli/security\_check.html
X No 'performance-check' entry was found in your deploy.yaml.
  Please add a performance-check entry *AFTER* the security-check entry in deploy.yaml
  so your docker image can be checked for performance regressions.
  More info: http://servicedocs.yelpcorp.com/docs/paasta\_tools/paasta\_cli/performance\_check.html
X Jenkins build pipeline missing. Please run 'paasta generate-pipeline'
  More info: http://y/paasta-deploy
✓ Git repo found in the expected location.
✓ Found Dockerfile
✓ Your Dockerfile pulls from the standard Yelp images.
X Couldn't find 'EXPOSE 8888' in Dockerfile. The Dockerfile should
  expose that per the doc linked below.
  More info: http://y/paasta-contract
✓ A Makefile is present
✓ The Makefile contains a tab character
✓ The Makefile contains a docker tag
✓ The Makefile responds to `make itest`
X The Makefile does not have a `make test` target. Jenkins needs
  this and expects it to run unit tests. More info: http://y/paasta-contract
X No marathon.yaml exists, so your service cannot be deployed.
  Push a marathon-[superregion].yaml and run `paasta generate-pipeline`.
  More info: http://y/yelpsoa-configs
✓ All entries in deploy.yaml correspond to a marathon entry
✓ All marathon instances have a corresponding deploy.yaml entry
X Your service is not using Sensu (monitoring.yaml).
  Please setup a monitoring.yaml so we know where to send alerts.
  More info: http://y/monitoring-yaml
X Your service is not setup on smartstack yet and will not be automatically load balanced.
  More info: http://y/smartstack-cep323
```

```
krall@dev4-devc:~/pg/paasta_demo (master) $ █
```

```
krall@dev4-devc:~/pg$ paasta fsm -y yelpsoa-configs -s paasta_demo
Smartstack proxy_port? 20931
Team responsible for this service? paasta
One line description of this service? Anything is possible in Paasta
Link to your CEP or SCF? http://html5zombo.com
```



With My Noodly Appendage I Have Written Configs For

paasta_demo

Customize Them If It Makes You Happy -- <http://y/paasta> For Details
Remember To Add, Commit, And Push When You're Done:

```
cd yelpsoa-configs/paasta_demo
# Review And/Or Customize Files
git add .
git commit -m'Initial Commit For paasta_demo'
git push origin HEAD # Pushmaster Or Ops Deputy Privs Required
```

```
krall@dev4-devc:~/pg$
```



```
krall@dev4-devc:~/pg/paasta_demo (master) $ paasta check
✓ yelpsoa-config directory for paasta_demo found in /nail/etc/services
✓ deploy.yaml exists for a Jenkins pipeline
✓ Found a security-check entry in your deploy pipeline
✓ Found a performance-check entry in your deploy pipeline
✗ Jenkins build pipeline missing. Please run 'paasta generate-pipeline'
  More info: http://y/paasta-deploy
✓ Git repo found in the expected location.
✓ Found Dockerfile
✓ Your Dockerfile pulls from the standard Yelp images.
✗ Couldn't find 'EXPOSE 8888' in Dockerfile. The Dockerfile should
expose that per the doc linked below.
  More info: http://y/paasta-contract
✓ A Makefile is present
✓ The Makefile contains a tab character
✓ The Makefile contains a docker tag
✓ The Makefile responds to `make itest`
✗ The Makefile does not have a `make test` target. Jenkins needs
this and expects it to run unit tests. More info: http://y/paasta-contract
✓ Found marathon.yaml file.
✓ All entries in deploy.yaml correspond to a marathon entry
✓ All marathon instances have a corresponding deploy.yaml entry
✓ monitoring.yaml found for Sensu monitoring
✓ Your service uses Sensu and team 'paasta' will get alerts.
✓ Found smartstack.yaml file
✓ Instance 'main' of your service is using smartstack port 20931 and will be automatically load
balanced
krall@dev4-devc:~/pg/paasta_demo (master) $ █
```

```
krall@dev4-devc:~/pg/paasta_demo (master) $ paasta check
✓ yelpsoa-config directory for paasta_demo found in /nail/etc/services
✓ deploy.yaml exists for a Jenkins pipeline
✓ Found a security-check entry in your deploy pipeline
✓ Found a performance-check entry in your deploy pipeline
✓ Jenkins build pipeline found
✓ Git repo found in the expected location.
✓ Found Dockerfile
✓ Your Dockerfile pulls from the standard Yelp images.
✓ Found 'EXPOSE 8888' in Dockerfile
✓ A Makefile is present
✓ The Makefile contains a tab character
✓ The Makefile contains a docker tag
✓ The Makefile responds to `make itest`
✓ The Makefile responds to `make test`
✓ Found marathon.yaml file.
✓ All entries in deploy.yaml correspond to a marathon entry
✓ All marathon instances have a corresponding deploy.yaml entry
✓ monitoring.yaml found for Sensu monitoring
✓ Your service uses Sensu and team 'paasta' will get alerts.
✓ Found smartstack.yaml file
✓ Instance 'main' of your service is using smartstack port 20931 and will be automatically load
balanced
krall@dev4-devc:~/pg/paasta_demo (master) $
```



```
krall@dev4-devc:~/pg/paasta_demo (master) $ paasta test-run
Building container from Dockerfile in /nail/home/krall/pg/paasta_demo
----> 7d0d345d863d
Step 1 : RUN apt-get -y --force-yes install python3 wget
----> Using cache
----> dacf277dba29
Step 2 : RUN wget -r html5zombo.com
----> Using cache
----> 293b5d7e13f2
Step 3 : CMD cd /html5zombo.com && python3 -m http.server 8888
----> Using cache
----> 9b7b81d9601b
Step 4 : EXPOSE 8888
----> Running in 88ddefa2f98a
----> f038b08af3a3
Successfully built f038b08af3a3
Warning! You're running a container in non-interactive mode.
This is how Mesos runs containers. Some programs behave differently
with no tty attach.
```

Mesos would have healthchecked your service via
<http://dev4-devc.dev.yelpcorp.com:33548>

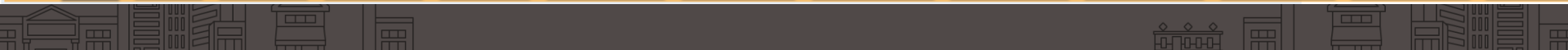
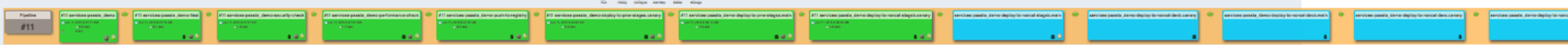
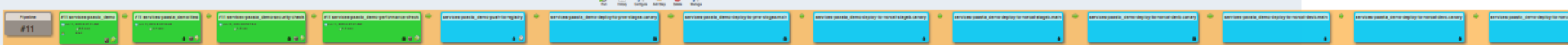
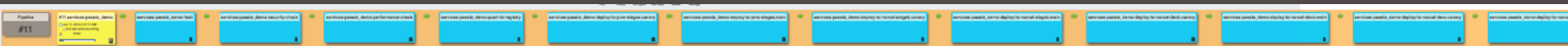
Zombo.com



```
krall@dev4-devc:~/pg/paasta_demo (master) $ git push origin HEAD
Counting objects: 6, done.
Delta compression using up to 24 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 638 bytes | 0 bytes/s, done.
Total 6 (delta 2), reused 0 (delta 0)
To git@git.yelpcorp.com:services/paasta_demo
 575aef6..8f90a52 HEAD -> master
krall@dev4-devc:~/pg/paasta_demo (master) $
```



Wait for Jenkins...



```
krall@dev4-devc:~/pg/paasta_demo (master) $ paasta status -c norcal-devc
```

```
Pipeline: https://jenkins.yelpcorp.com/view/services-paasta\_demo
```

```
cluster: norcal-devc
```

```
instance: canary
```

```
Git sha: 6d1101bb
```

```
State: Running - Desired state: Started
```

```
Marathon: Healthy - up with (1/1) instances. Status: Deploying.
```

```
Mesos: Healthy - (1/1) tasks in the TASK_RUNNING state.
```

```
Smartstack: N/A - canary is announced in the main namespace.
```

```
instance: main
```

```
Git sha: 6d1101bb
```

```
State: Running - Desired state: Started
```

```
Marathon: Healthy - up with (3/3) instances. Status: Deploying.
```

```
Mesos: Healthy - (3/3) tasks in the TASK_RUNNING state.
```

```
Smartstack: Critical - in haproxy with (0/4) total backends UP in this namespace.
```

```
krall@dev4-devc:~/pg/paasta_demo (master) $ paasta status -c norcal-devc -v
```



```
krall@dev4-devc:~/pg/paasta_demo (master) $ paasta status -c norcal-devc
Pipeline: https://jenkins.yelpcorp.com/view/services-paasta\_demo
```

```
cluster: norcal-devc
```

```
instance: canary
```

```
Git sha: 1432707d
```

```
State: Running - Desired state: Started
```

```
Marathon: Healthy - up with (1/1) instances. Status: Running.
```

```
Mesos: Healthy - (1/1) tasks in the TASK_RUNNING state.
```

```
Smartstack: N/A - canary is announced in the main namespace.
```

```
instance: main
```

```
Git sha: 1432707d
```

```
State: Running - Desired state: Started
```

```
Marathon: Healthy - up with (3/3) instances. Status: Running.
```

```
Mesos: Healthy - (3/3) tasks in the TASK_RUNNING state.
```

```
Smartstack: Healthy - in haproxy with (4/4) total backends UP in this namespace.
```

```
krall@dev4-devc:~/pg/paasta_demo (master) $ █
```

ZOMBO



dev4-devc:20931

Search



Zombo.com



Questions?

