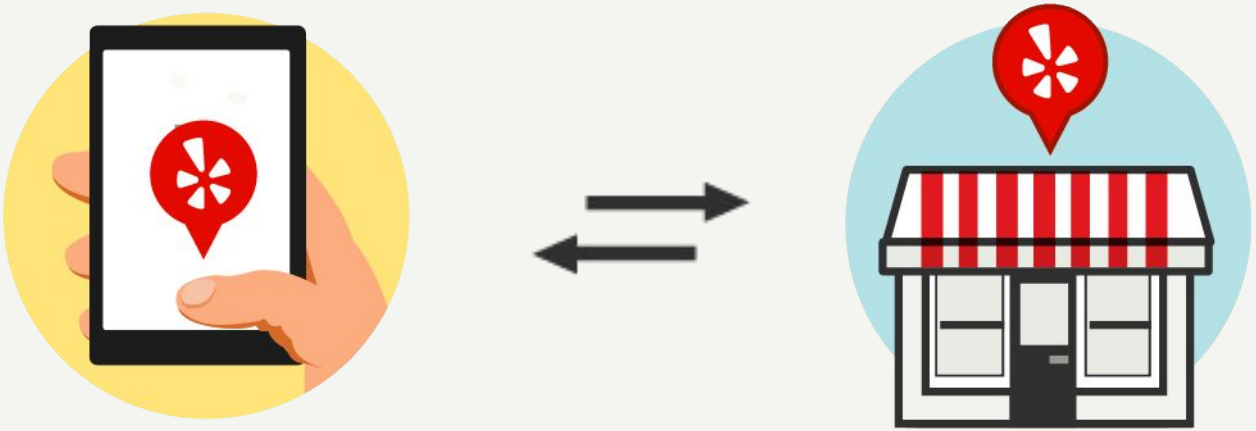
# Seagull: A distributed, fault tolerant, concurrent task runner

Sagar Patwardhan
sagarp@yelp.com

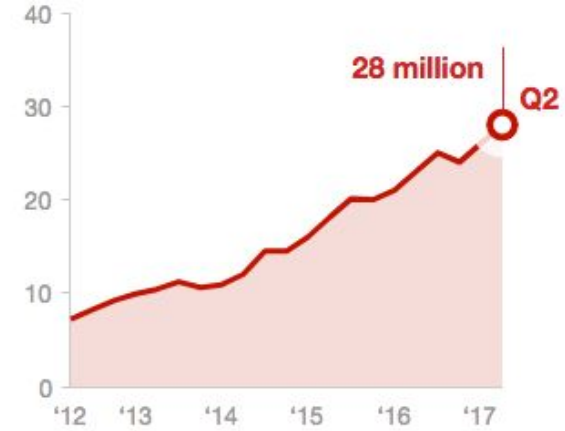
# Yelp's Mission

Connecting people with great local businesses.



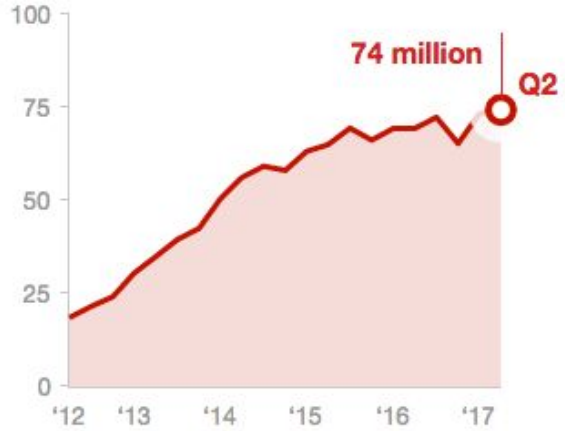
# Yelp scale

**Average monthly mobile app unique users**



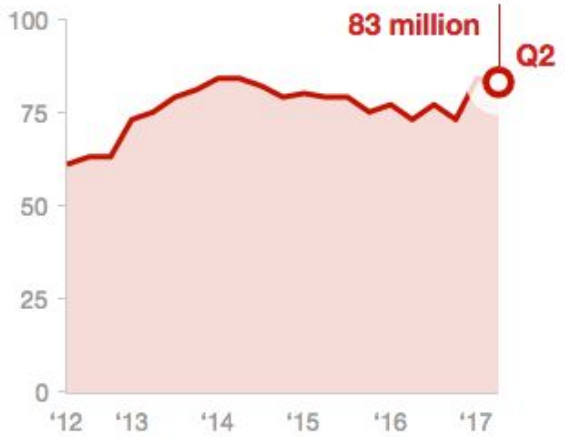

Calculated as the number of unique devices accessing the app on a monthly average basis over a given three-month period, according to internal Yelp logs.

**Average monthly mobile web unique visitors**




Calculated as the number of "users," as measured by Google Analytics, accessing Yelp via mobile website on a monthly average basis over a given three-month period.

**Average monthly desktop unique visitors**




Calculated as the number of "users," as measured by Google Analytics, accessing Yelp via desktop computer on an average monthly basis over a given three-month period.

# Outline

What is Seagull? Why did we build it?

Deep dive into Seagull

Fleetmiser: Yelp’s in-house cluster autoscaler

Challenges faced and lessons learned

Future of Seagull

# Testing at Yelp

Yelp needs to run ~100,000 tests for its applications.

Tests take ~2 days to run if executed serially.

North of 500 developers.

Directly impacts developer productivity.

# Seagull

# Current seagull scale

~**350** seagull runs every day. Average run time ~**10-15 mins**.

**~2.5** million ephemeral containers every day.

Cluster scales from **~70** instances to **~450** instances.

**All spot** instances.

**~25** million tests executed every day.

# Applications of seagull

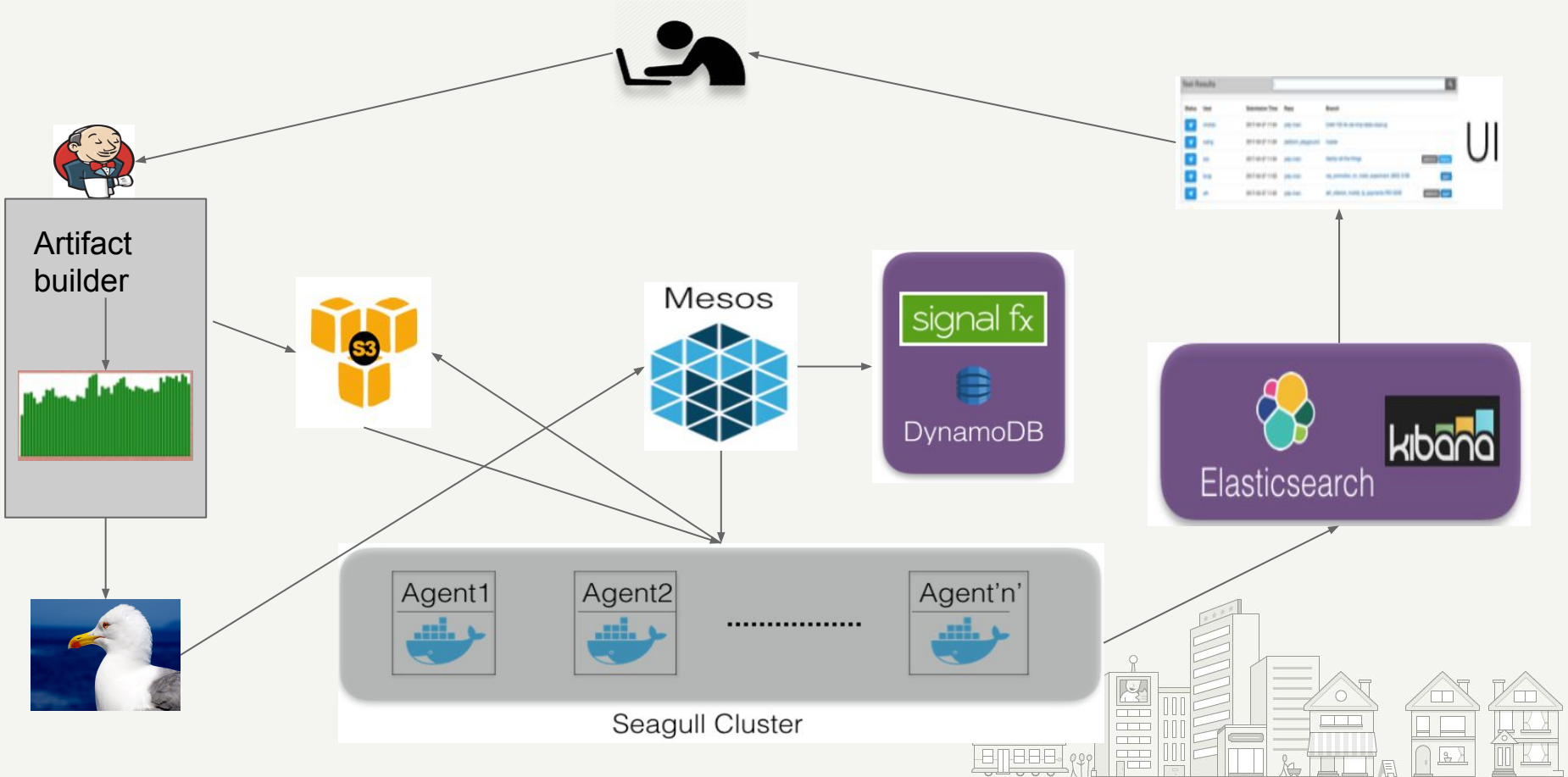
Run Dockerized integration, acceptance tests

Locust: Yelp’s load testing framework.

Photo classification: Classify tens of millions of photos in less than a day.

# Deep dive into seagull

# Seagull workflow for testing



# Seagull Mesos scheduler

Written in python; Uses libmesos

One scheduler per test suite per run

~40-50 schedulers running simultaneously at peak

Customizable concurrency

Fault tolerant

# Placement strategies

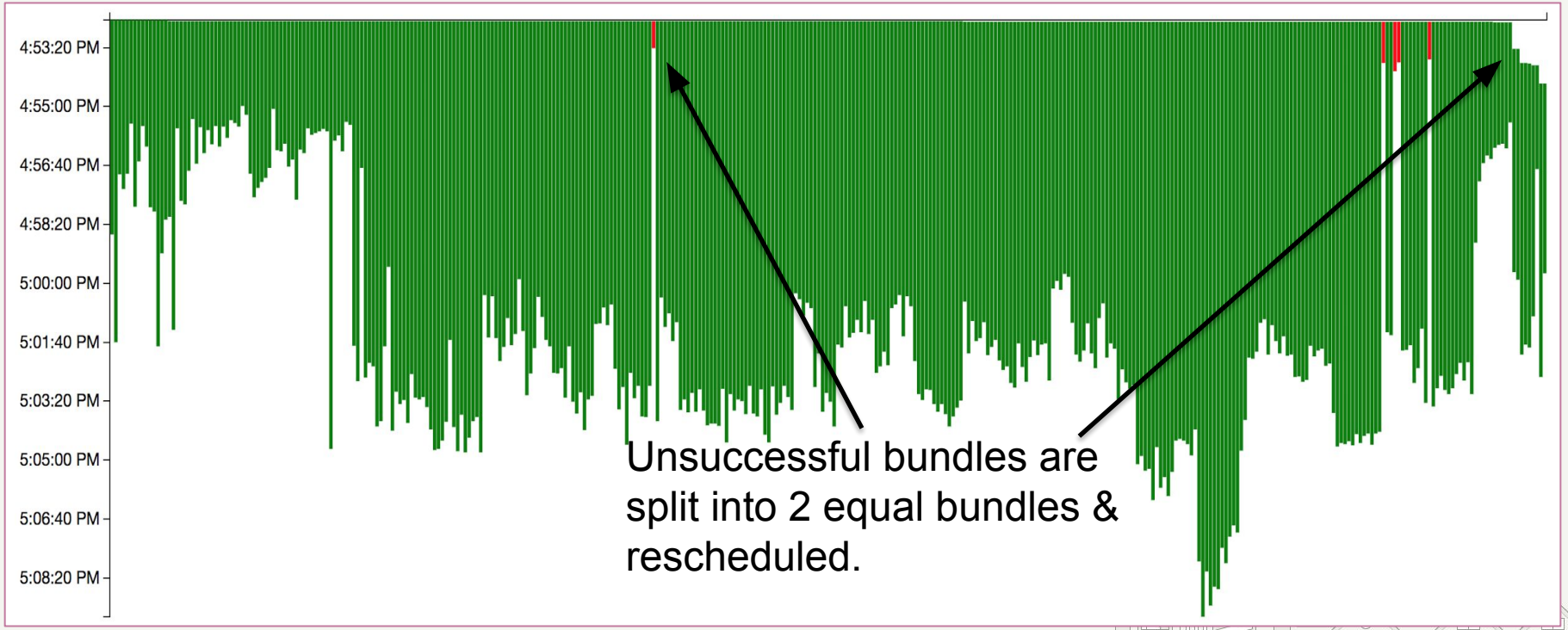
Aim: Optimize for seagull bundle setup time.

Affinity for already used agents.

Use as many resources in an offer as possible.

This also simplifies the scale down.

# Unsuccessful tasks/bundles



# Seagull executor

Custom mesos executor written in python.

Uses Mesos containerizer and cgroups isolator.

Does setup and teardown of bundles.

Reports resource utilization stats.

Uploads log files to s3, sends metrics to ElasticSearch and SignalFx.

# Clusterwide resources

Clusterwide resources: selenium and database connections

Resources are not tied to specific agents.

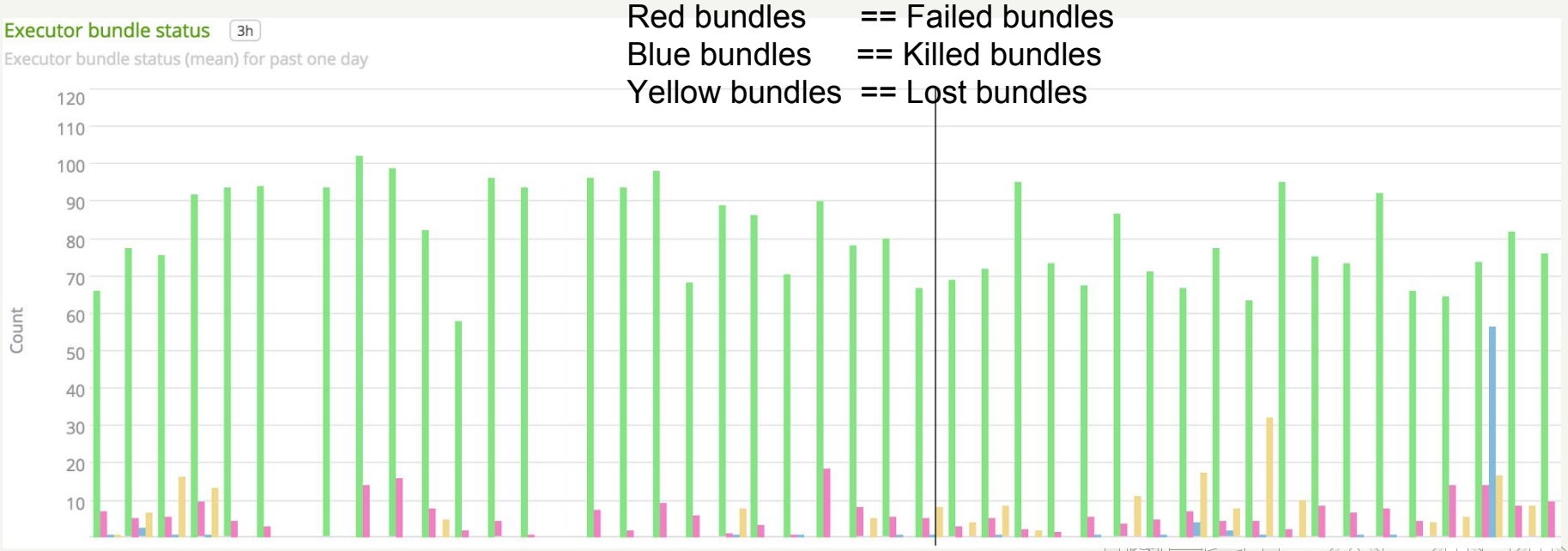
ZooKeeper ephemeral znodes to keep track of how many connections are being used.

ZooKeeper locks for atomic access.

Resources are freed up when executors go away.

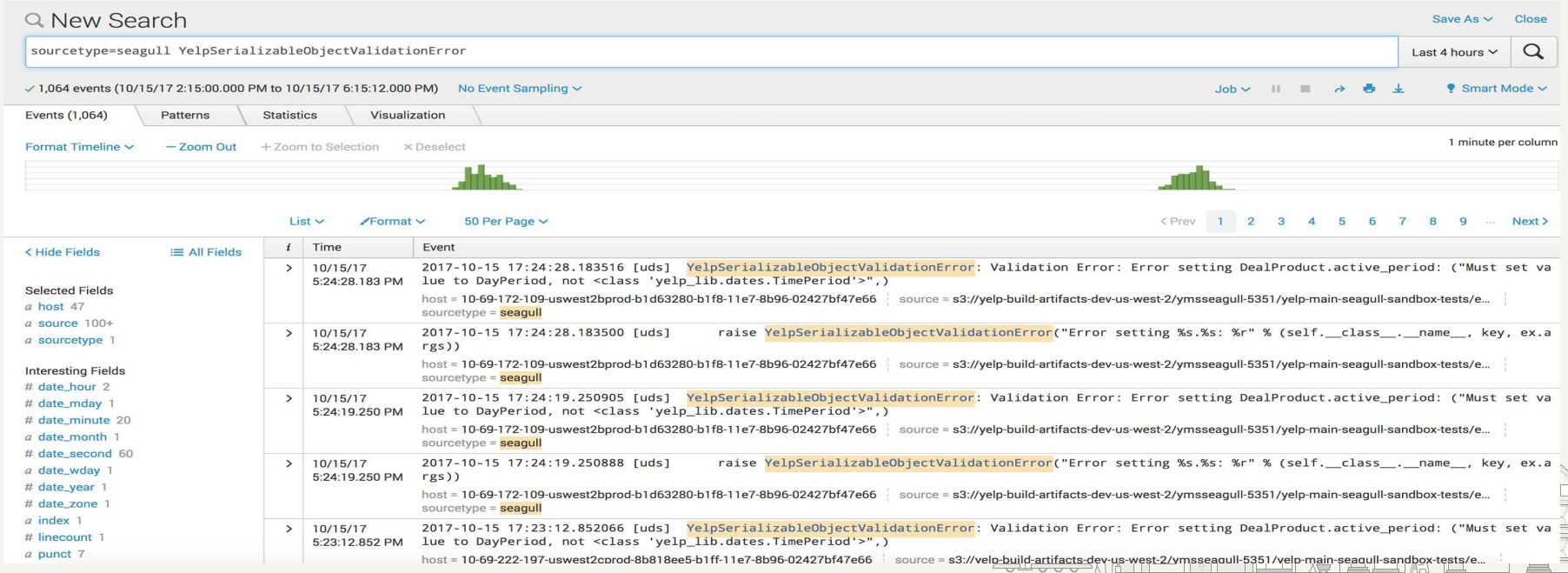
# Monitoring & Alerting

# Real time monitoring & alerting using SignalFx



# Log aggregation in splunk

stdout & stderr of all the executors is stored in Splunk which allows us to see failure trends across multiple seagull runs.



# Efficient bundling of tasks for Seagull

# Greedy Algorithm

Test timings are stored in ElasticSearch.

P90 of test timings for last one week are stored in DynamoDB every day.

The list is sorted in ascending order of test timings.

Tests are packed into bins of 10 minutes.

# Linear Programming algorithm

Handle test dependencies. Some tests cannot be run together. Some tests need to run together.

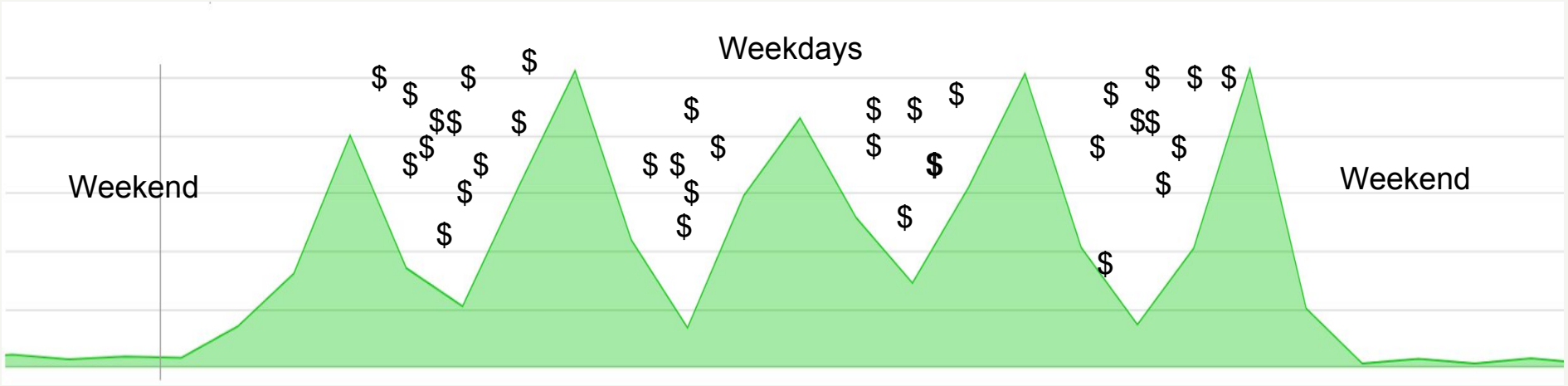
We use the PuLP LP solver.

Goals:

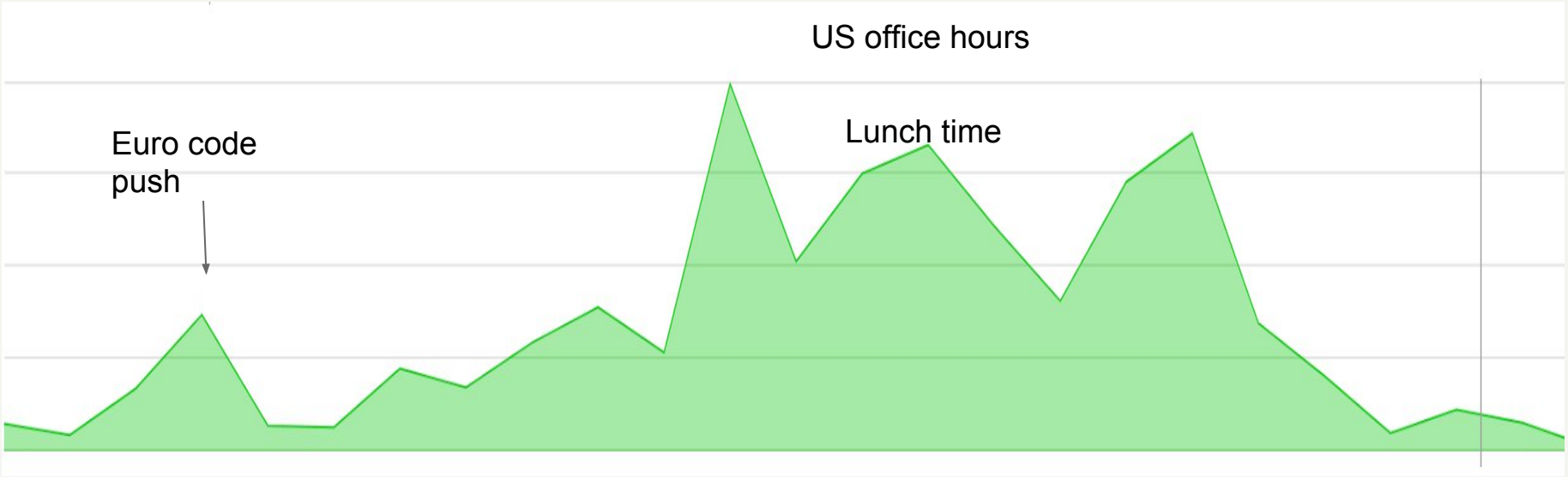
1. Minimize the number of bundles created.
2. A test is present in only one bundle.
3. A single bundle’s work is less than 10 mins.

# Autoscaling the cluster

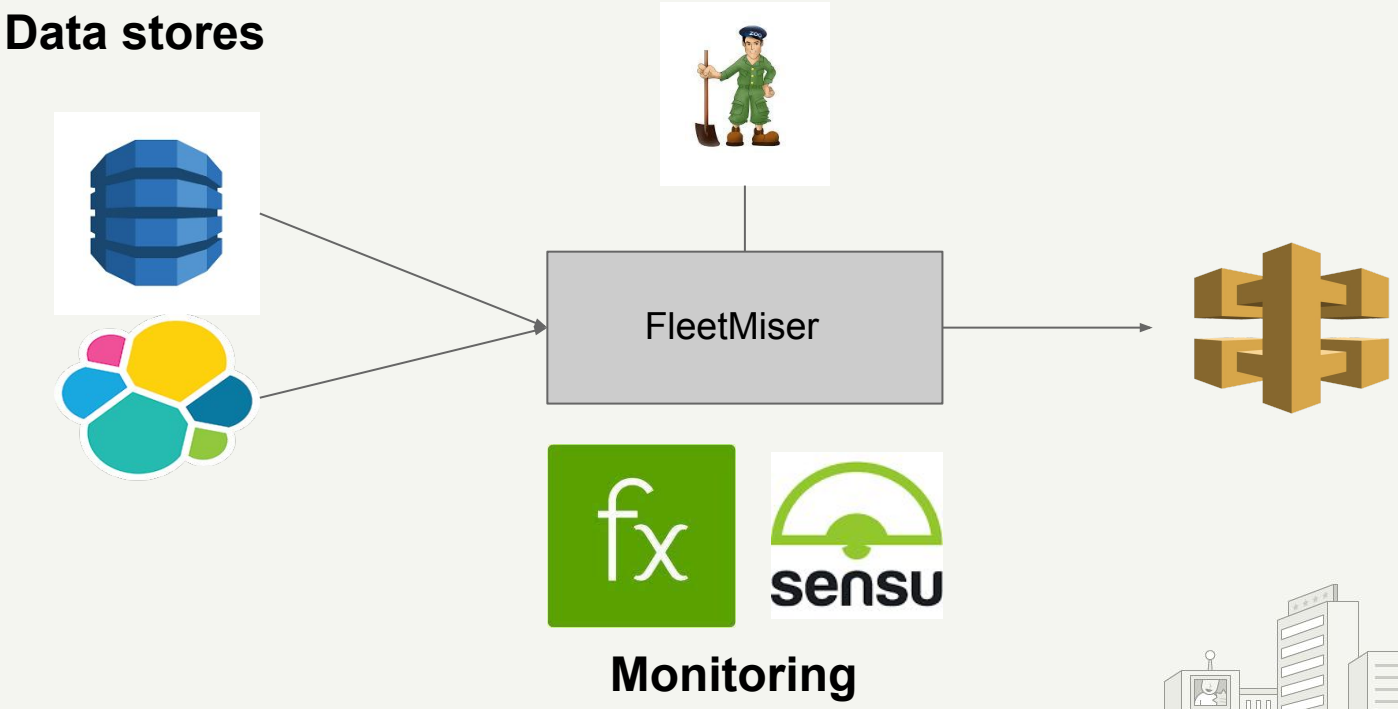
# Savings!!!




# Daily usage trends




# FleerMiser: Yelp's in-house autoscaler



# Auto scaling strategies

CPU utilization

Seagull runs in flight

# Based on CPU utilization

Our tasks are CPU bound

Autoscaler tracks the CPU utilization in the cluster, and makes decisions based on that.

If the cluster utilization > 65% for 15 minutes, then we scale up.

If the cluster utilization is < 35% for 30 mins, then we scale down.

# Based on the number of Seagull run submitted

Whenever a new Seagull run is submitted, autoscaler gets notified about it.

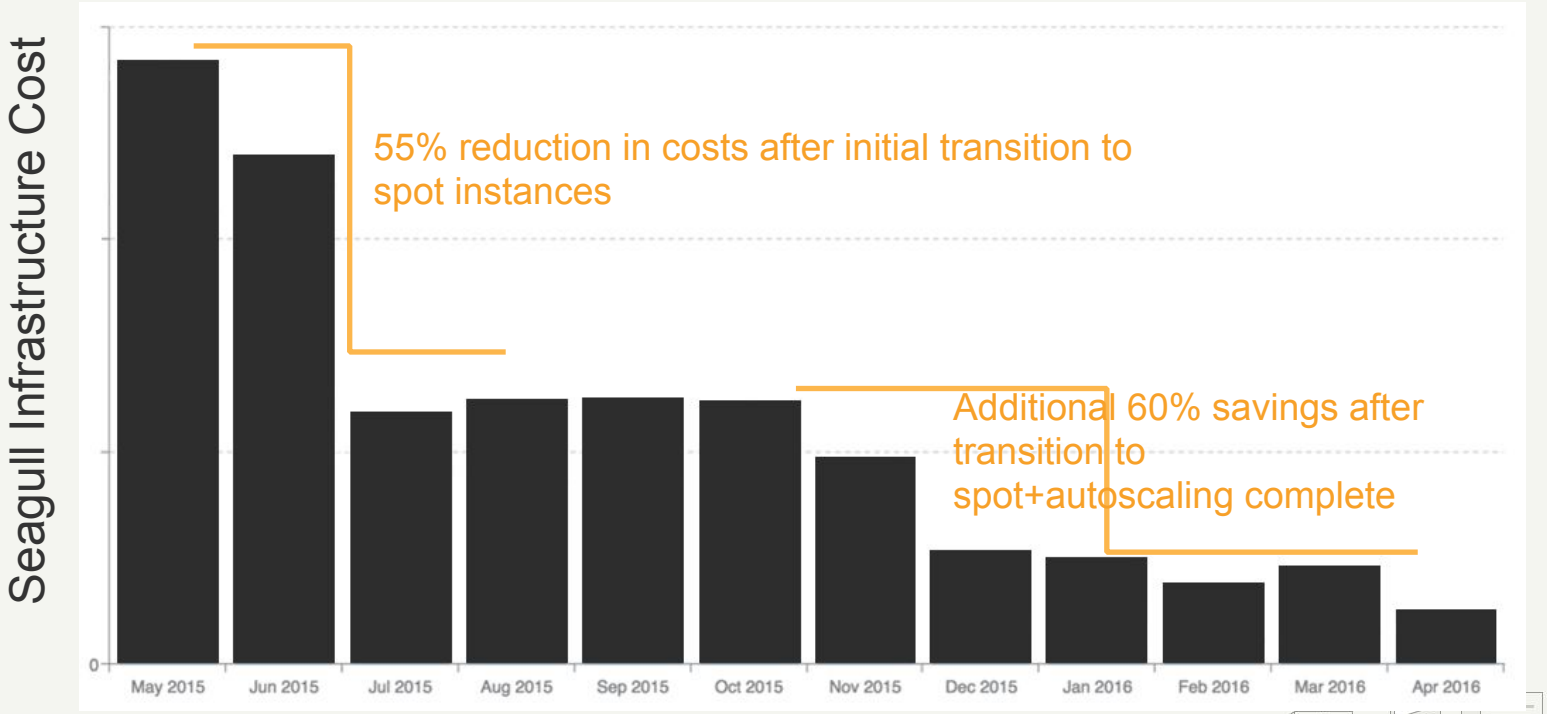
Autoscaler anticipates the resources required for seagull runs triggered and adds resources to the cluster.

# Scaling down is difficult!

AWS Spotfleet does not allow us to specify which instances to terminate.

Autoscaler finds and terminates the idle instances, and readjusts the Spotfleet capacity.

# 80% in cost savings in compute cost



# Key challenges and solutions

# Bandwidth issues while talking to s3

Artifact and docker image download takes a long time causing seagull runs to be delayed.

Other applications in the VPC are affected by this.

# Use VPC S3 endpoints

Fast and secure access to S3 without any limitations on bandwidth.

Traffic does not leave Amazon network.

*Caveat*: It can be only enabled for the S3 buckets in the same AWS region.

# Central Docker registries get overwhelmed

Setup: Multiple Docker registries on a single host behind an nginx proxy.

It failed to cope up with requests being made.

Solution: Run Docker registries on every agent. Use /etc/hosts for address resolution.

# Spot instances

AWS gives a warning 2 mins before reclaiming spot instances.

Solution:
A cron job terminates all the running executors upon receiving a termination notice.

mesos-agent process is killed to prevent new tasks from getting scheduled.

# Spot markets are volatile

Fluctuations in spot prices of instances in certain markets can have an adverse effect on your application.

Getting the bid price right is hard. Trade-off between availability and cost savings.

Solutions:
Make your application fault tolerant.

Diversify! Add more spot markets.

# Issues with Docker daemon

Docker daemon gets locked up and does not respond to requests.

Deadlock in Docker daemon.

Docker daemon randomly fails to resolve DNS.

AUFS causes soft CPU lockup.

# Orphaned Docker containers

Cannot kill containers because docker daemon gets busy which leads to orphaned docker containers.

Containers take up resources that are not accounted for in Mesos.

Boxes eventually OOM.

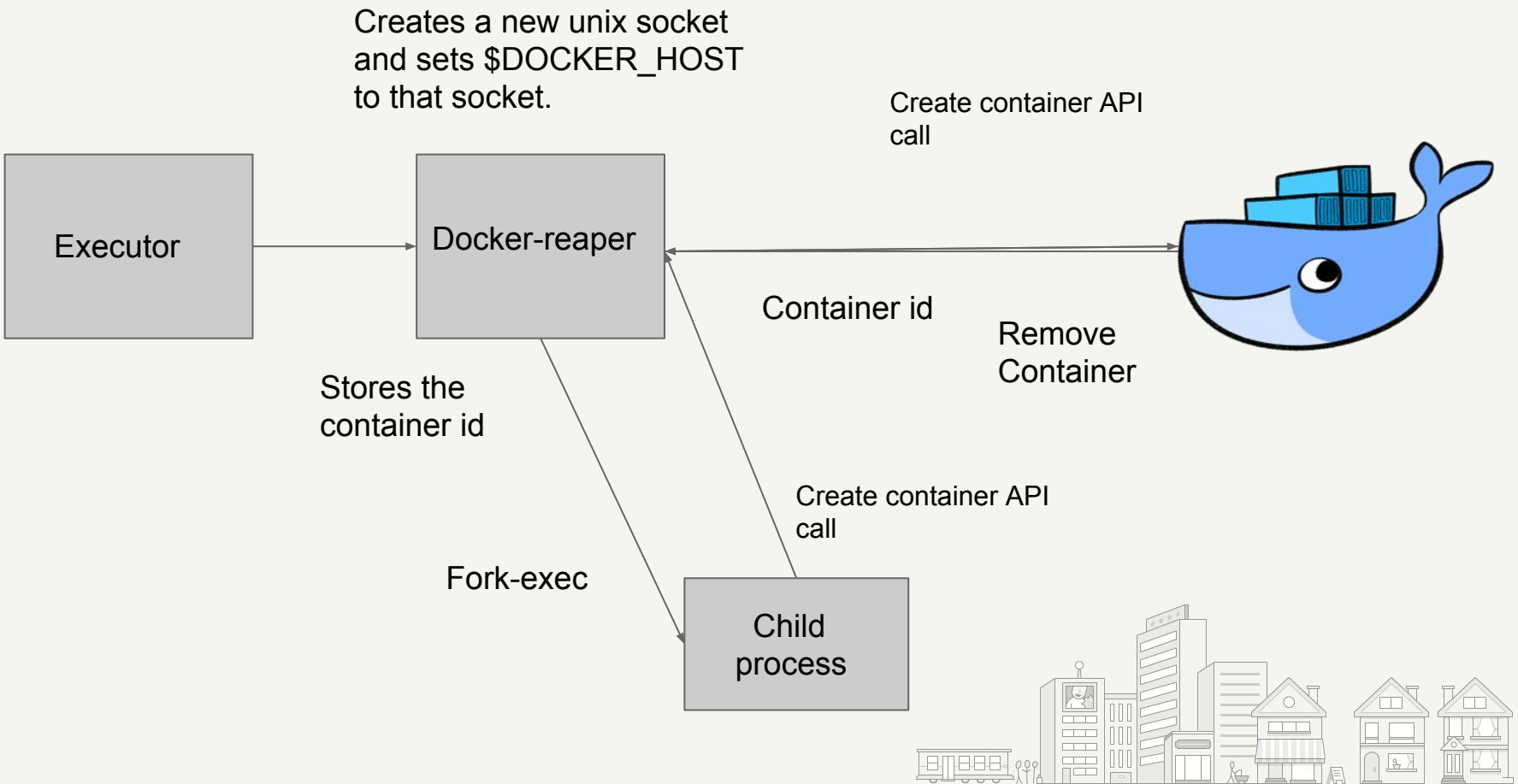
# docker-reaper

Proxy for Docker daemon.

Written in go.

Forwards all the signals to its children.

Cleans up all the containers after the child process exits.


Creates a new unix socket and sets $DOCKER_HOST to that socket.
Create container API call
Executor
Docker-reaper
Container id
Remove Container
Stores the container id
Create container API call
Fork-exec
Child process

# Mesos maintenance mode

Designed to be used by a single operator.

Need external locking mechanism to make it work for multiple operators.

# Future of Seagull

# Scheduler improvements

Use oversubscription.

Use task_processing library to replace the core-component of the scheduler.

Use CSI plugin to implement clusterwide resources.

Make it easier for other services/applications to use seagull for parallelizing tasks.

# Executor improvements

Containerize everything!!!

Use Docker runtime in Mesos containerizer and eliminate the need to talk to Docker daemon.

Experiment with nested containers and pods.

# Autoscaler improvements

More advanced autoscaling for better resource utilization

Use multiple spot fleets. We may save more money?

Use more instance types in the cluster.

# We are hiring in Europe!

- Offices in London or Hamburg, remote workers also welcome!

- Engineers or managers with dist-sys experience:
  - Strong knowledge of systems and application design.
  - Ability to work closely with information retrieval/machine learning experts on big-data problems.
  - Strong understanding of operating systems, file systems and networking.
  - Fluency in Python, C, C++, Java, or a similar language.
  - Technologies we use: Mesos, Marathon, Docker, ZooKeeper, Kafka, Cassandra, Flink, Spark, Elasticsearch

Apply at https://www.yelp.com/careers or come say hi!


fb.com/YelpEngineers

@YelpEngineering

engineeringblog.yelp.com

github.com/yelp