

How to Cost Effectively and Reliably Build Infrastructure for Machine Learning

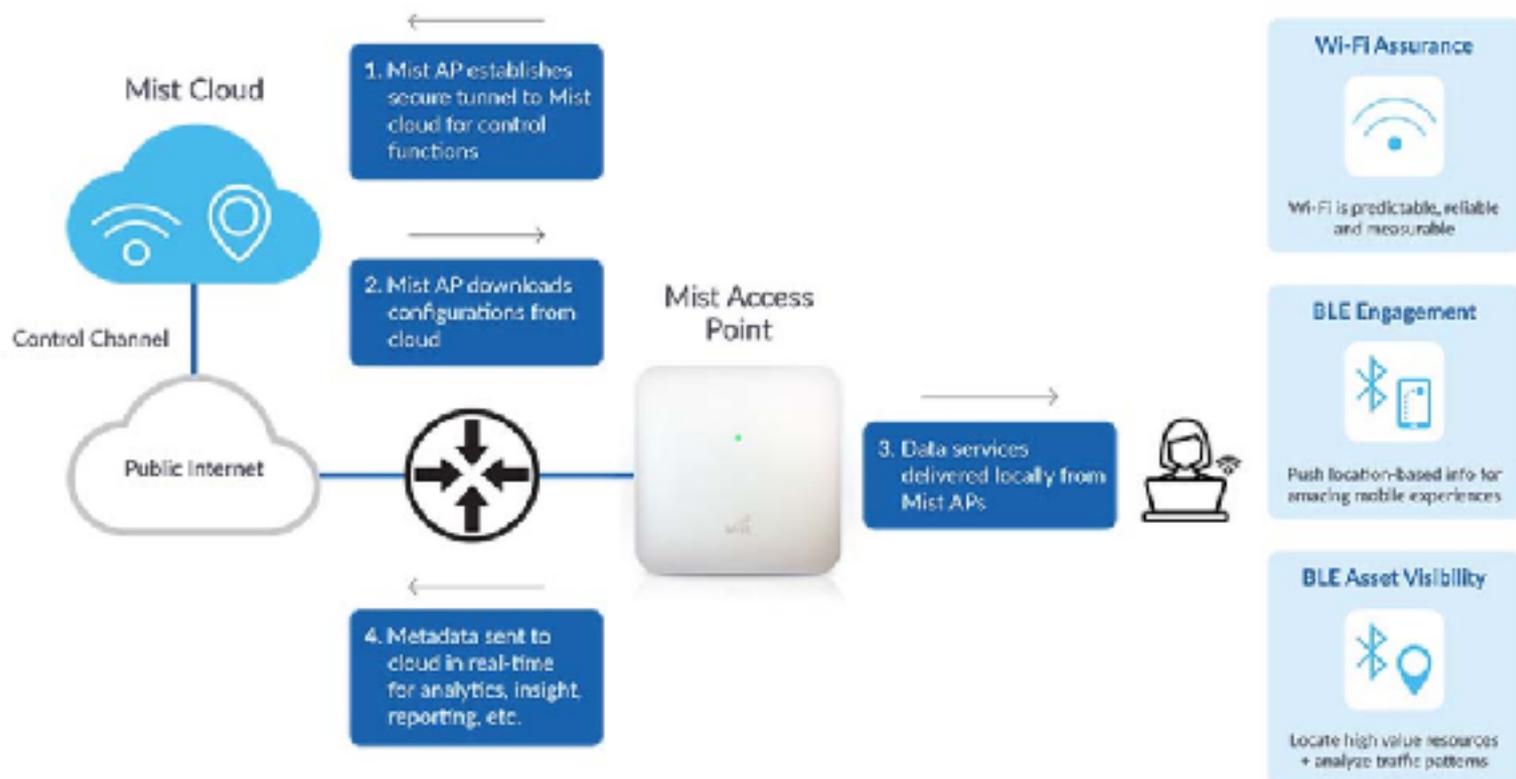
Osman Sarood
(Mist Systems)

Strata
DATA CONFERENCE



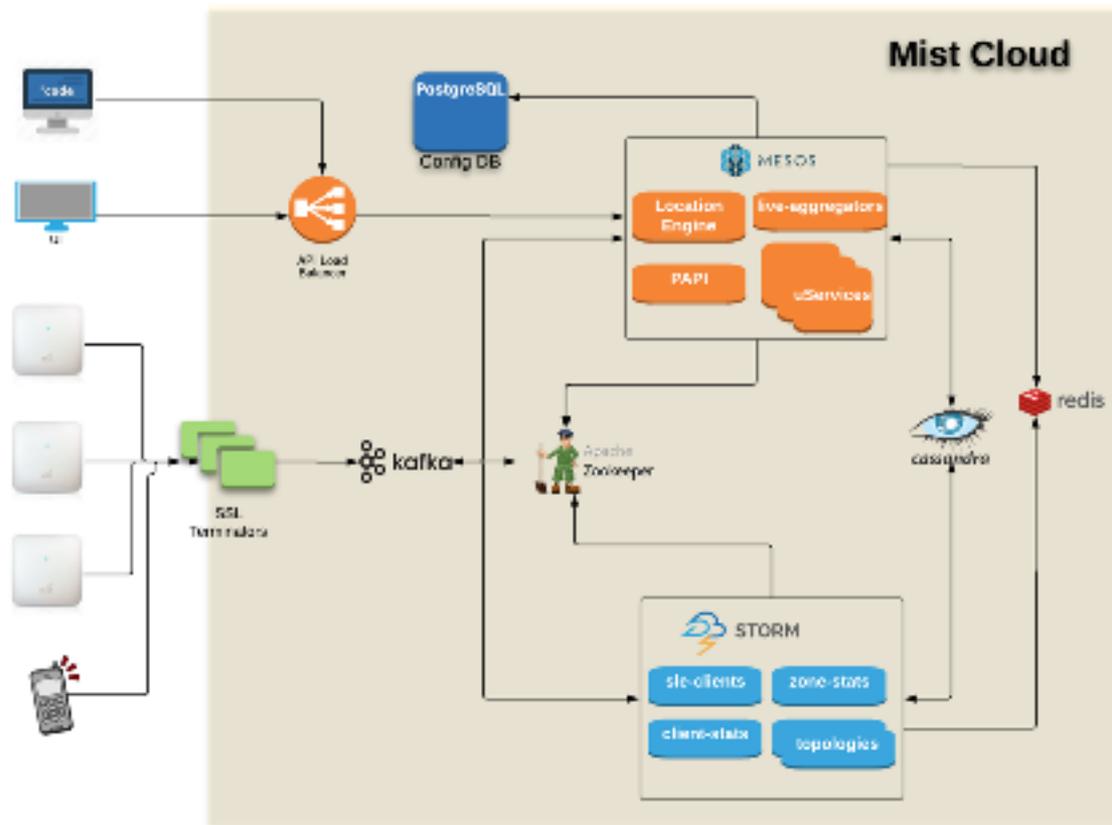
- Fan of performance, reliability, load balancing, scheduling and cost
 - PhD @ UIUC (High Performance Computing)
 - Authored 20+ research papers
 - Speaker at several academic and industry conferences
- Currently @ Mist System
 - Managing infrastructure, data platform and operations teams
 - Tech lead for infrastructure, data platform and operations
 - Software Engineer
 - Seen several 10X growth spurts :)

What's Mist?



Mist Architecture

-  1 TB+
-  kafka 500+ partitions
10 Billion+ Msgs
-  VPC 10's TB+



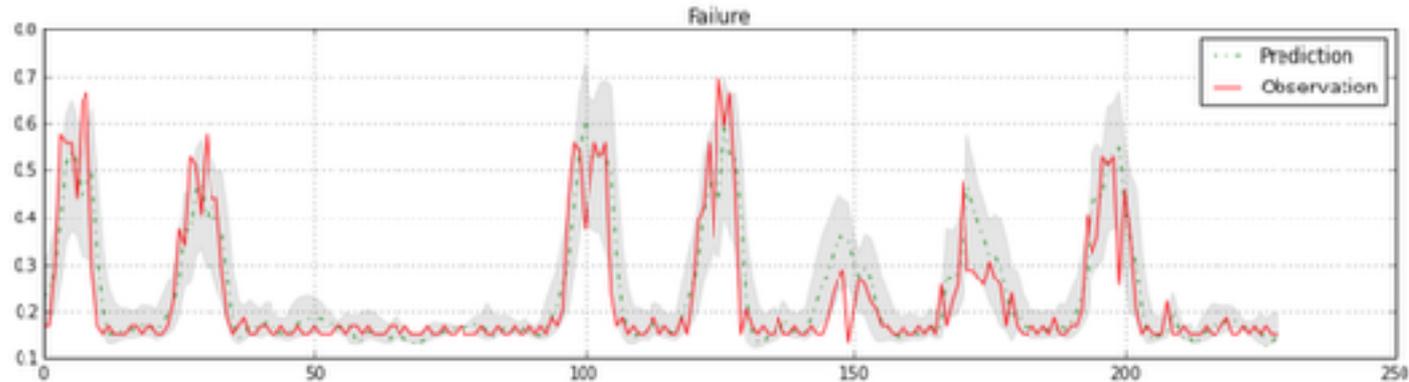
Marvis: Virtual Network Assistant - Inference Engine

- Based On:
 - Probabilistic programming
 - Bayesian statistics
- Feeds on preprocessed streaming data from:
 - Apache Storm
 - Live-aggregators



Anomaly Detection

- Consumes preprocessed-aggregated data
- Intelligent spatio-temporal monitoring
- Analyzing network data across multiple dimensions
- Seasonal ARIMA and Tensor Flow models



What to expect?

- Minimize cost using the volatile AWS Spot instances
- Ensuring reliability amidst unpredictable server terminations
- Container right-sizing for reliability and autoscaling
- How to monitor real-time applications under chaotic conditions



Heat Map for CPU Load (Hotspots)



AWS EC2 Contract Types

- On-demand
- Reserved
- Spot



Contract Type	Cost	Reliable	Commitment Required?
On-demand	Very High	Yes	No
Reserved	High	Yes	Yes
Spot	Low	No	No

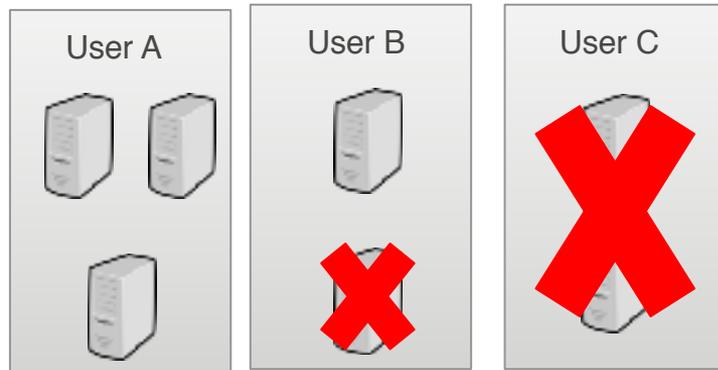
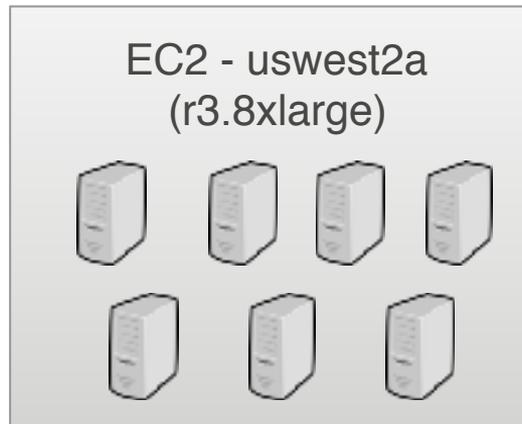
Spot Instances

- Unused AWS EC2 capacity
- Bidding based
- If used correctly, they can be ~ 80% cheaper
- Be careful! Naive usage may end up costing more than on-demand

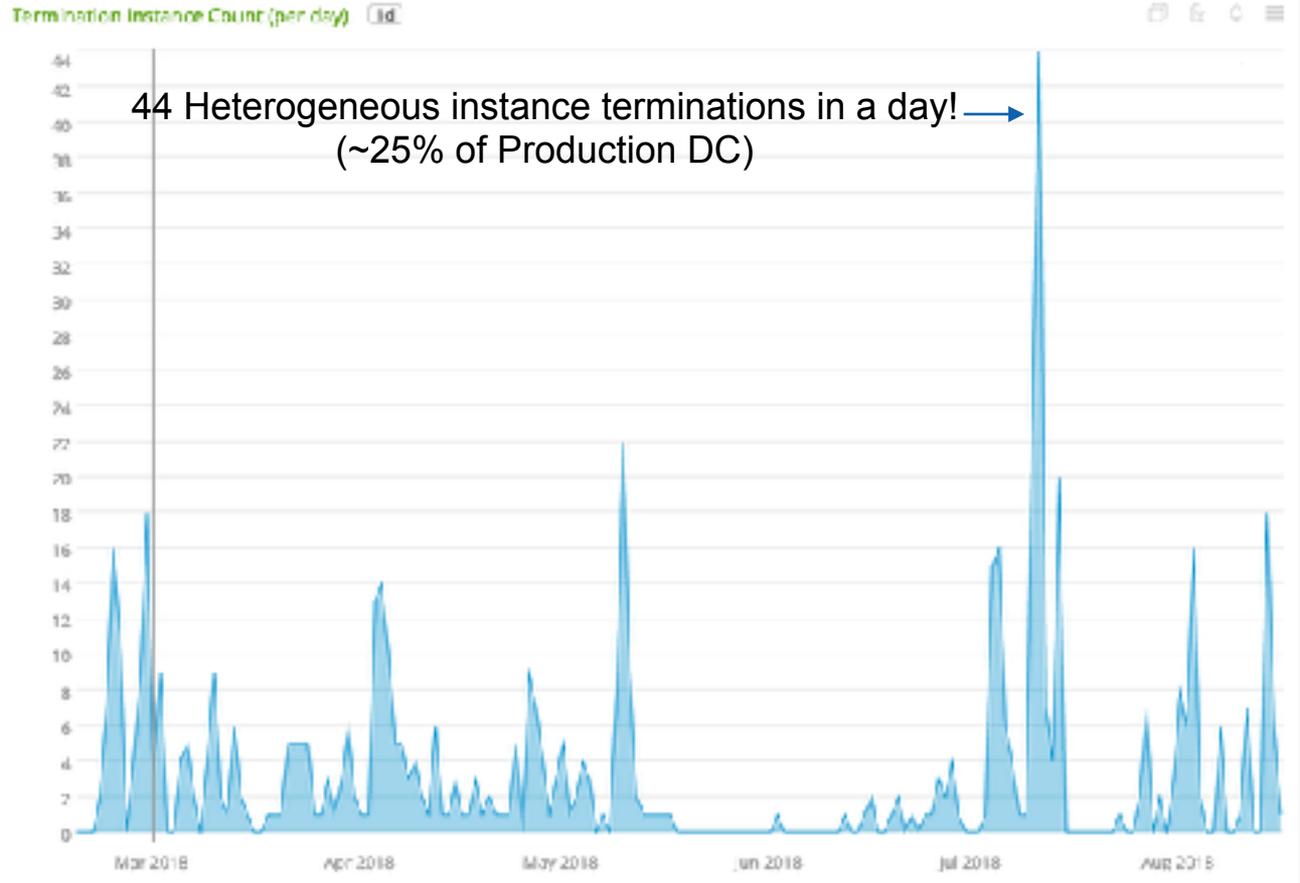


How does Spot Pricing Work?

- Three users make the requests
 - A: 3 machines bidding @ \$10
 - B: 2 machines bidding @5
 - C: 4 machines bidding @0.4
- User 'A' gets 3 machines
- User 'B' gets 2 machines
- User 'C' gets *just* 2 machines
- Spot price is the bid price of last fulfilled request, i.e., \$0.4 (every pays \$0.4)
- On-demand demand increases by 3:
 - 3 spot machines terminated
 - New spot price: \$5

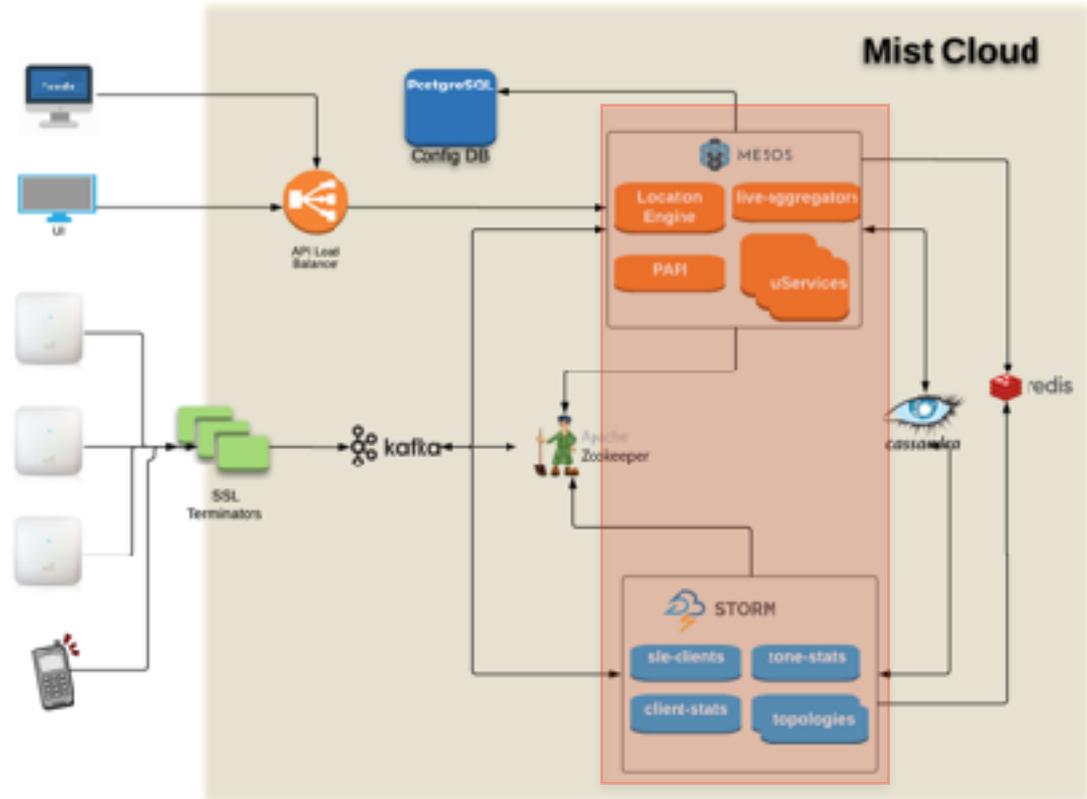


Spot Market Volatility



Mist Architecture Running On Spot

- 100% on Spot
- 80% of Datacenter
- Huge savings \$\$\$
- Allows us to do more
- Forces reliability



Cost Benefits of Using Spot Instances

System	vCPUs	Percentage of total EC2 cost	Savings compared to On-demad	Savings compared to RI
Storm	1096 (31%)	17%	74%	60%
Mesos	1864 (52%)	30%	75%	62%
Total Cores	3590		56%	35%

17% non-spot infra constitutes 53% of our cost!



*Includes EBS cost as well

How to Use Spot Instances?

- Spot market = (instance type, AZ)
- Prices for different spot markets are independent
- AWS Spot Fleet
 - Uses spot instances
 - Ensures capacity
- Caution: Cost saving scheme of AWS Spot Fleet is risky!
- **Diversify across markets!**
- **Applications can restart**

Request id	af-20e66fa
Request type	test
Created	1/16/2017, 3:47:28 AM
State	active
Status	Unfilled
Total target capacity	340 (0% filled)
On-Demand capacity	0 (0% filled)
Allocation strategy	Diversified
Instance types	c3.xlarge weight=4, c3.2xlarge weight=5, c3.4xlarge weight=75, c3.xlarge weight=4, i3.2xlarge weight=6, i3.4xlarge weight=15, r3.xlarge weight=4, r3.2xlarge weight=4, r3.xlarge weight=4, d3.xlarge weight=3, r3.4xlarge weight=16
AWS ID	ami-
Subnet	subnet-
IAM fleet role	aws-ec2-spot-test-role

How Much To Overprovision?

- Lead time for spot instance: < 5 mins
- Max number of instances terminated in 5 mins = 9
- Overprovision by $9 * 3 = 27$ (where 3 is a magic multiple)
- The bigger the cluster the better it is



Unreliable Servers Help Building Reliable Systems!?

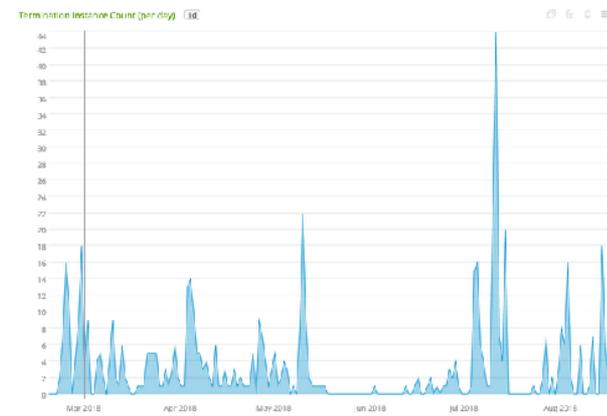
- Chaos Monkey

- Pioneer of chaos engineering
- Controlled Chaos (can be stopped and resumed)



- Use of Spot Instances

- Uncontrolled 'Real' Chaos
- Can't be turned off and depends on spot prices
- Enables building reliable infrastructure. Wait what?
- Can be volatile



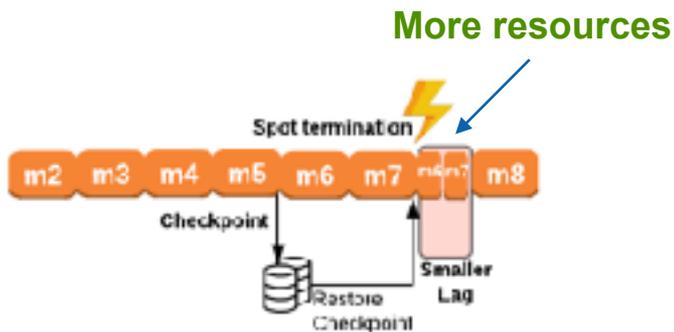
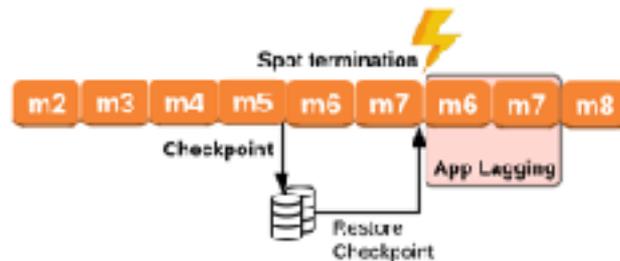
Coping with Spot's Unreliability

- Mist Systems relies on both stateful and stateless applications
- Live-aggregators:
 - 1.2+ TB in-memory state
 - ~ 1000 Mesos containers
 - CPU Utilization 60%-80%
 - Per-container memory footprint
 - minimum: 32MB
 - maximum: 21GB
 - mean: 1.3GB

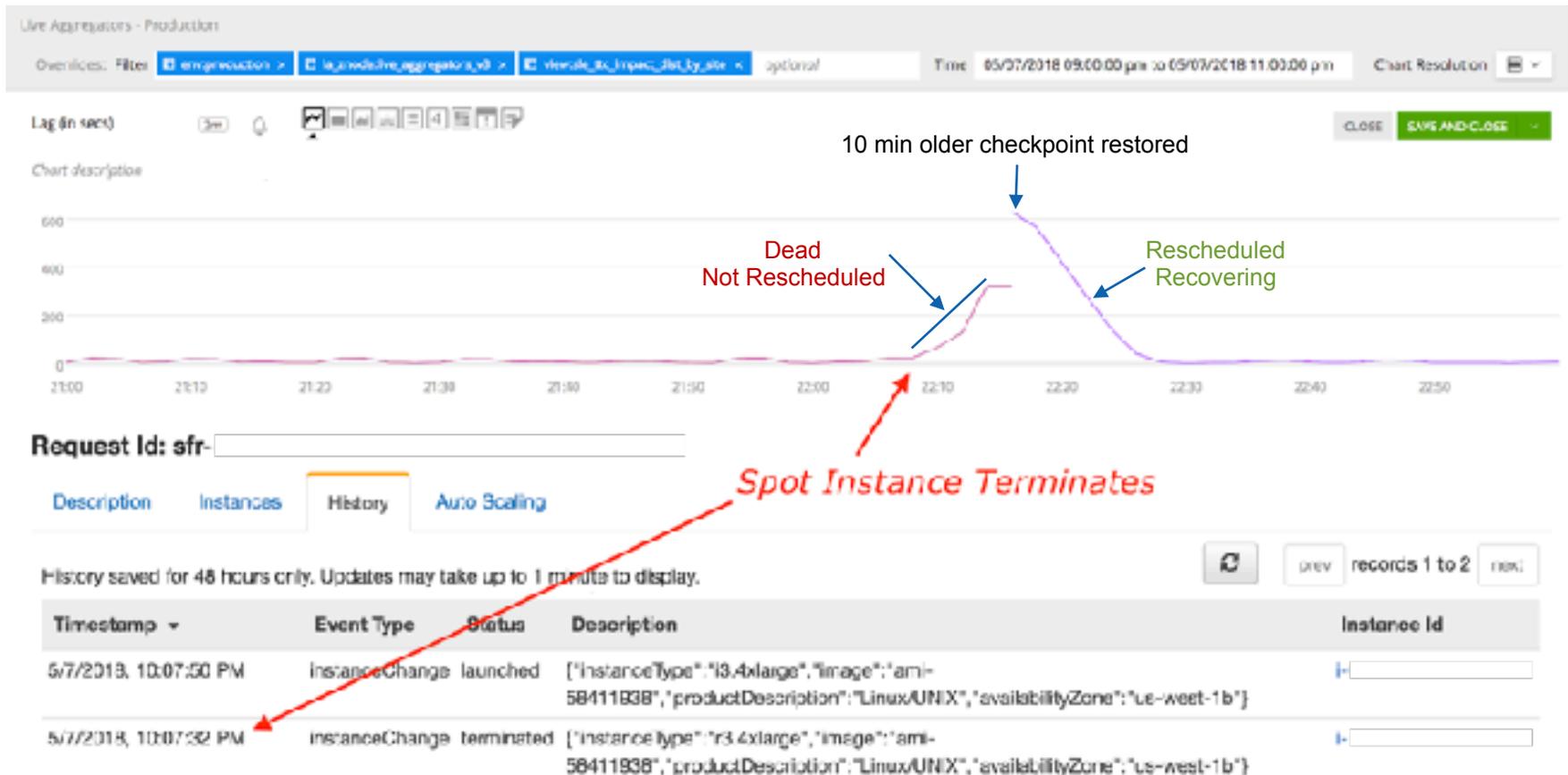


Coping with Spot's Unreliability: Checkpointing

- How far a realtime application is lagging?
- Checkpointing is critical!
- Each platform we use has a mechanism for checkpointing:
 - Storm: Redis
 - Mesos:
 - Live-aggregators: S3
 - Real-time: Redis
 - API: Stateless



Spot Termination: Lag and then Recovery



Difference in Recovery Times

- High Message Rate
- Type of computation
- Amount of data/msg



PLOT EDITOR	CHART OPTIONS	AXES	DATA TABLE	EVENTS (0)
Pinned Value	Value		executor_id	view
7.657000	4.857000		LAExecutor9160	sle_coverage_impact_dist_by_cl
5.755000	9.386000		LAExecutor9159	sle_coverage_impact_by_classifier_wl_cl
5.160000	1.372000		LAExecutor9161	fcc_classifier_trend_hist_by_cl

Why Do We Monitor?

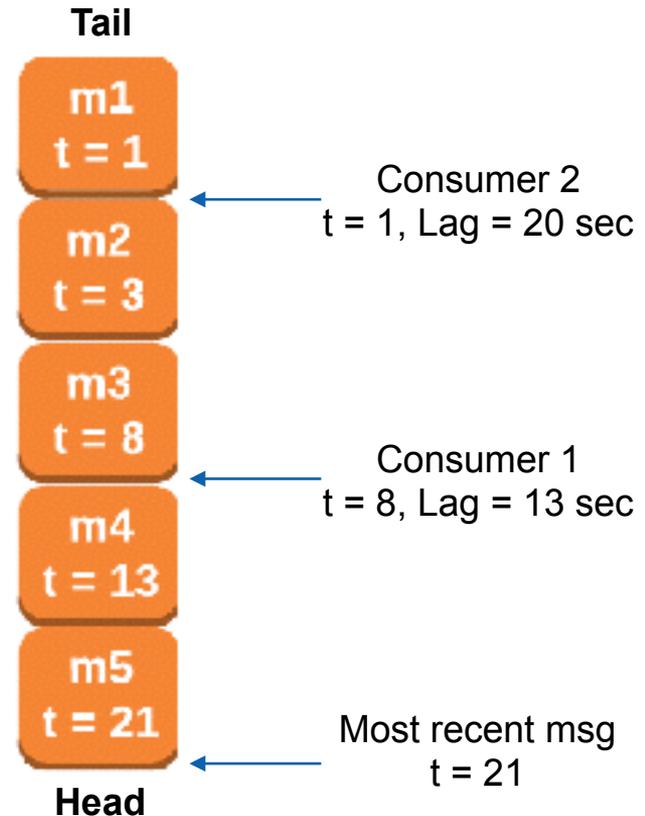
- Detecting the problem: Is a real time application falling behind?
- Troubleshooting:
 - Resource bottlenecks
 - Bottlenecked on an external system
 - Application code errors
- Use SignalFX, Graphana, Graphite and Cloudwatch

SignalFx

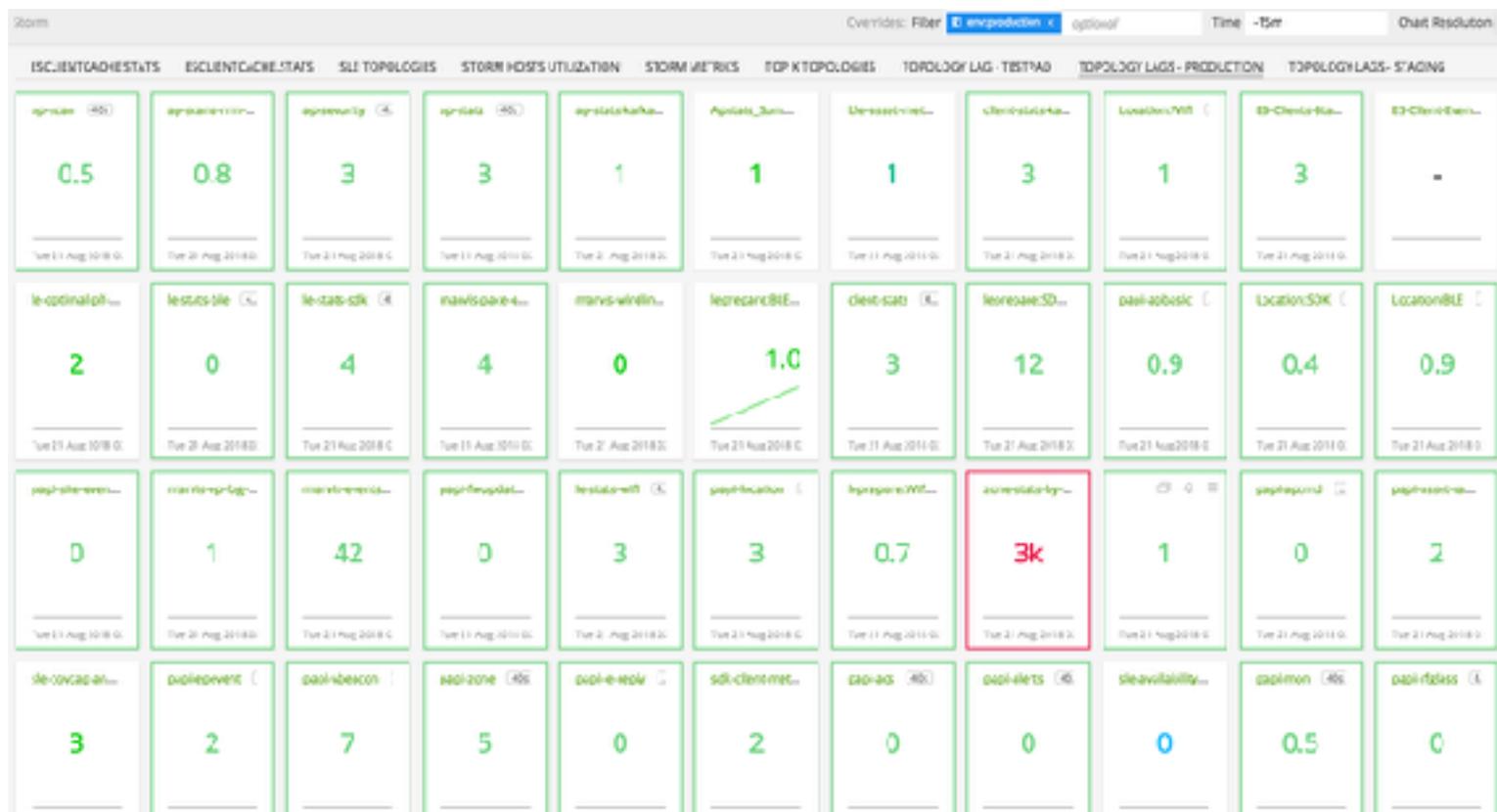


Detection: Realtime Lag

- How far behind is a realtime service/topology?
- Difference between Kafka timestamp between:
 - Most recent message
 - Last consumed message for an application
- How much lag is acceptable? nano secs, millisecs, secs?



Realtime Lags

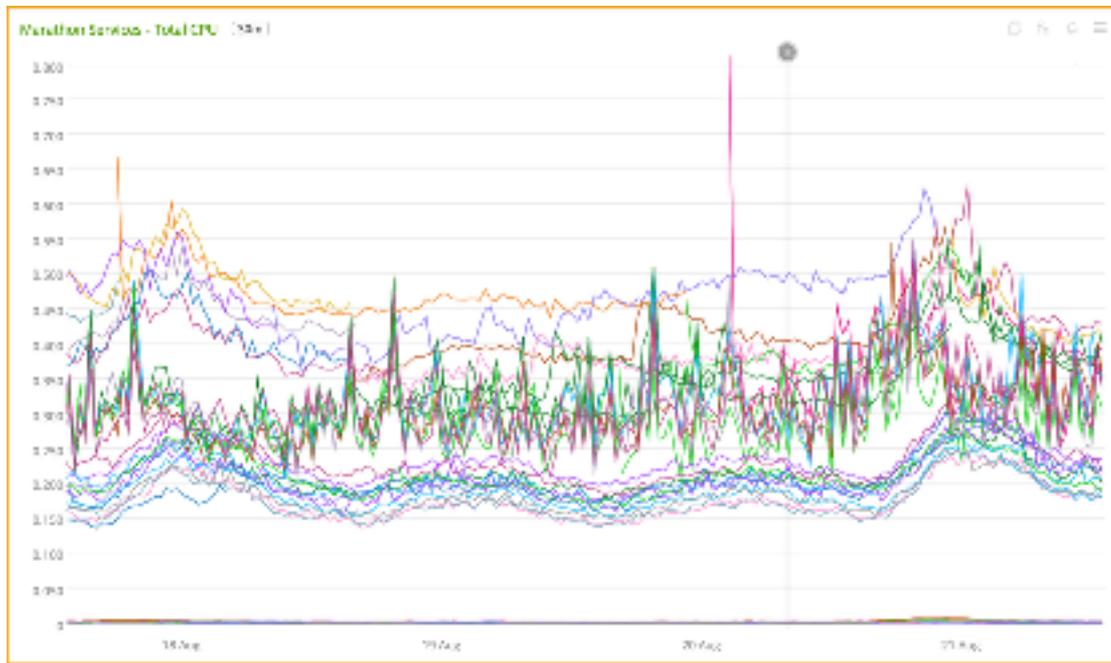


Attribution: Resource Monitoring

- Track key metrics per container/worker
 - CPU, Memory, Network IO
 - CPU Kafka stream
 - Timing individual code blocks
- Write metrics using different dimensions to filter and aggregate
 - Host name
 - Container ID
 - Service/Application name
 - Environment
 - Instance Type



CPU Metrics for Microservices



DATA TABLE DWDITS (6)

Pinac Value	Value	service	hostname	env	container_id
0.000000	-	le_b_a_jocost...	microfront-172.31.127-158-prod-ci-stun...	production	le_ba_ba0e4e428c-f1a8-b0b-02237531107
0.000000	-	le_b_a_jocost...	microfront-172.31.127-158-prod-ci-stun...	production	le_ba_ba0e4e428c-f1a8-b0b-02237531107
0.0311750	-	le_b_a_jocost...	microfront-172.31.127-158-prod-ci-stun...	production	le_ba_ba0e4e428c-f1a8-b0b-02237531107
0.000000	-	le_b_a_jocost...	microfront-172.31.127-158-prod-ci-stun...	production	le_ba_ba0e4e428c-f1a8-b0b-02237531107
0.0223160	-	le_b_a_jocost...	microfront-172.31.127-158-prod-ci-stun...	production	le_ba_ba0e4e428c-f1a8-b0b-02237531107
0.0203482	0.2490916	pepi	microfront-172.31.127-158-prod-ci-stun...	production	pepi16190d1-e38f-f1a8-b0b-02237531107
0.0194656	-	le_b_a_jocost...	microfront-172.31.127-158-prod-ci-stun...	production	le_ba_ba0e4e428c-f1a8-b0b-02237531107
0.0260007	0.2512948	pepi	microfront-172.31.127-158-prod-ci-stun...	production	pepi16190d1-e38f-f1a8-b0b-02237531107
0.000000	0.0000000

Performance Variations Across Instance Types

CPU utilization (%) for consuming the same stream



Case Study: Lagging Application



SignalFx APP

Critical Alert: Lag (in secs) Detector V3 (Lag (in secs) Detector V3)
Rule "Lag (in secs) Detector V3" in detector "Lag (in secs) Detector V3" triggered at Sun, 26 Aug 2018 02:30:20 GMT.

Triggering condition: The value of live_agg_lags_in_secs - Maximum by view,partition,env - Maximum(5m) is above 1200 for 10m.

Case Study: High CPU Utilization

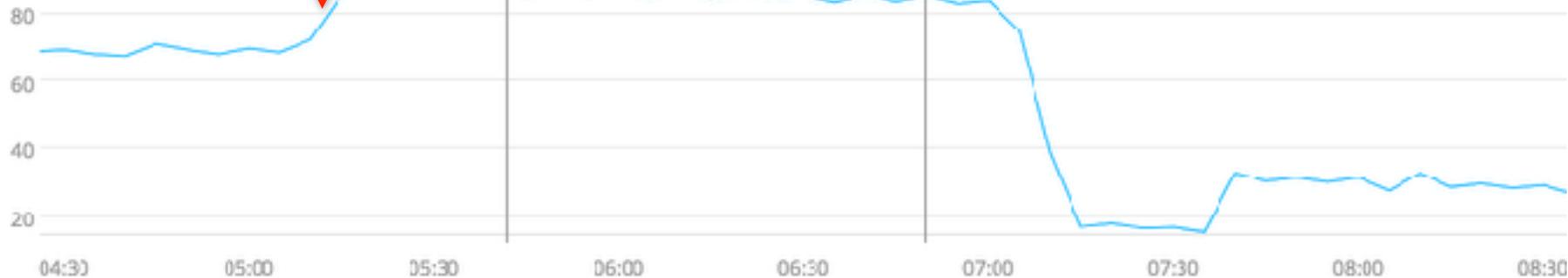
Mesos Host CPU Utilization (%)

5m



Chart description

~ 5:15



PLOT EDITOR CHART OPTIONS AXES DATA TABLE EVENTS (0)

Pinned Value ▾	Value	Plo: Name	host
85.36253	87.60300	Storm Max CPU	mesos-slave-172-31-26-207-production.mistsys.net

Case Study: Host CPU Utilization Details

Marathon Services Production

Overrides: Filter

env:production

hostname:mesos-slave-172-31-26-207-production.mistsys.net

Marathon Services - Total CPU

25m



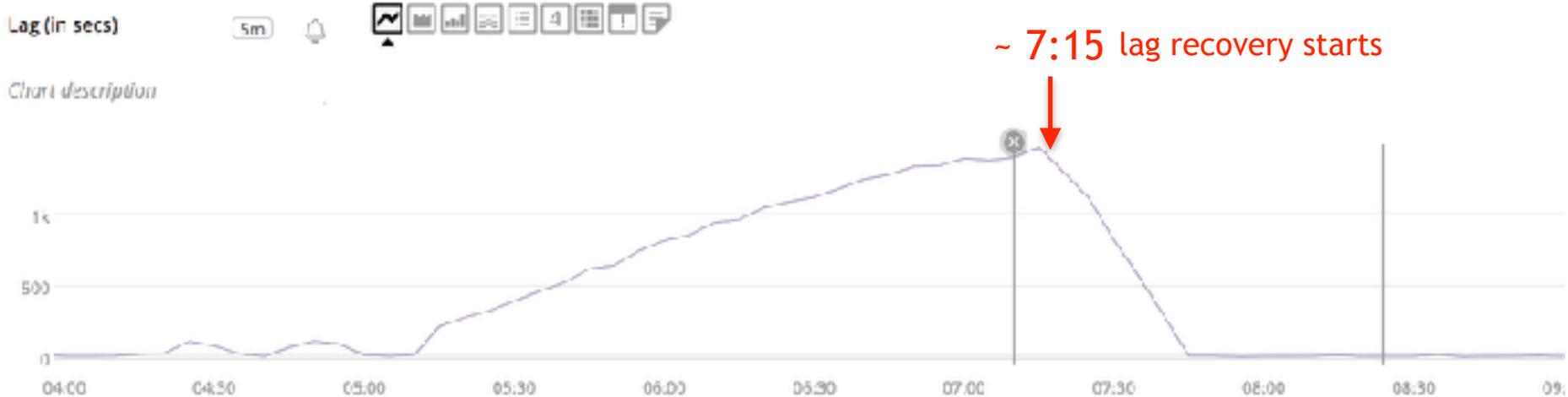
Chart description



PLOT EDITOR CHART OPTIONS AXES DATA TABLE EVENTS (0)

Plotted Value	Value	framework_id	service	executor_id
9.408766	9.266495	97f57f43-de4e-4e97-Rar...	le_wifi_leprepare	le_wifi_leprepare-02f6b990-f9f7-11e6-846b-06203f753107
0.8028556	-	la-framework-1528343712	LAExecutor7468	LAExecutor7468
0.3932275	0.3051193	97f57f43-de4e-4e97-Dec...	opstats_summarizer	opstats_summarizer-f744b7dc-6a30-11e0-bc6b-06203f753107
0.9449751	0.2827740	la-framework-1528343712	LAExecutor9879	LAExecutor9879
0.2857536	0.2843128	la-framework-1528343712	LAExecutor4361	LAExecutor4361
0.0105130	0.0103297	la-framework-1528343712	LAExecutor44	LAExecutor44
0.0008706	0.0008551	97f57f43-de4e-4e97-Rar...	shuffle	shuffle-852b36e6-8f67-11e6-b9bb-06203f753107

Case Study: Lag Recovers



PLOT EDITOR CHART OPTIONS AXES DATA TABLE EVENTS (0)

Filtered Value ▾	Value	view	sf_metric	host
1,402	6323500	site_ftc_impact_cist_by_site	live_eggs.lags_in_secs	mesos-slave-172-31-25-207-production.mistsys.net

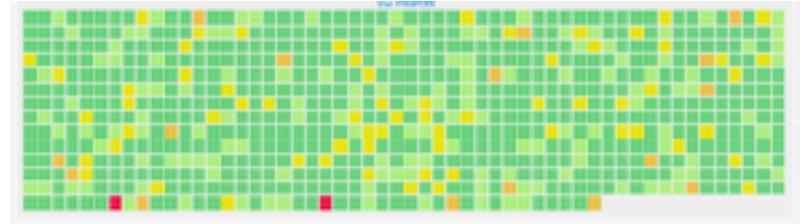
- Predicting resource requirements is difficult
- How much CPU, memory and network IO?
- Recent resource managers are typically book keepers:
 - Mesos
 - Kubernetes
- Lying factor is the difference between (1 - 2 = -1 cores)
 - Reserved resources (1 cores)
 - Actual used resources (2 cores)

Noisy Neighbor

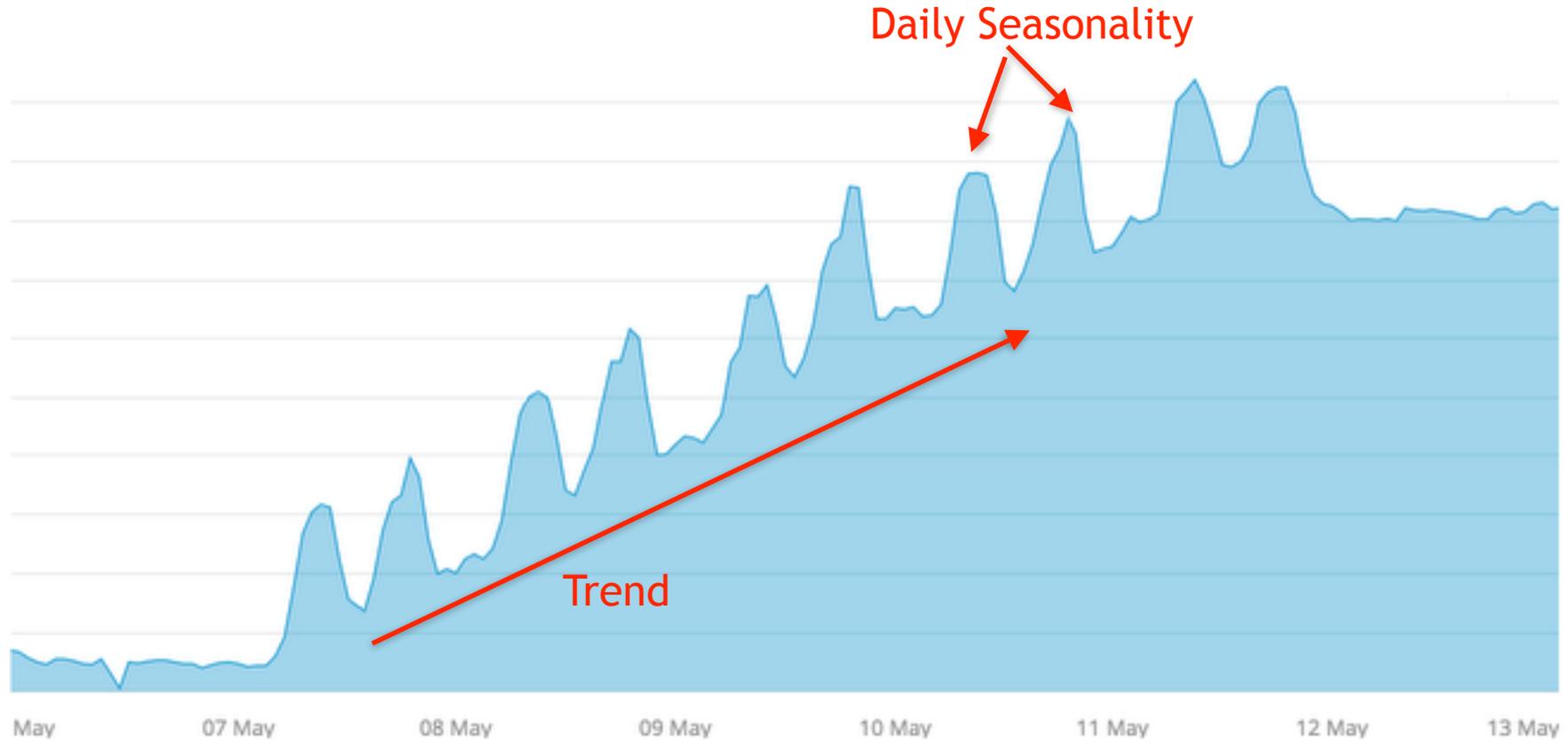


Right-sizing Containers

- Lying Factor for a container:
 - Negative: Dangerous! Using more resources than it reserved (hotspots)
 - Positive: Resources wasted
- Ideally should be 0 (reserved = actual/consumed)
- Manually update resources using marathon

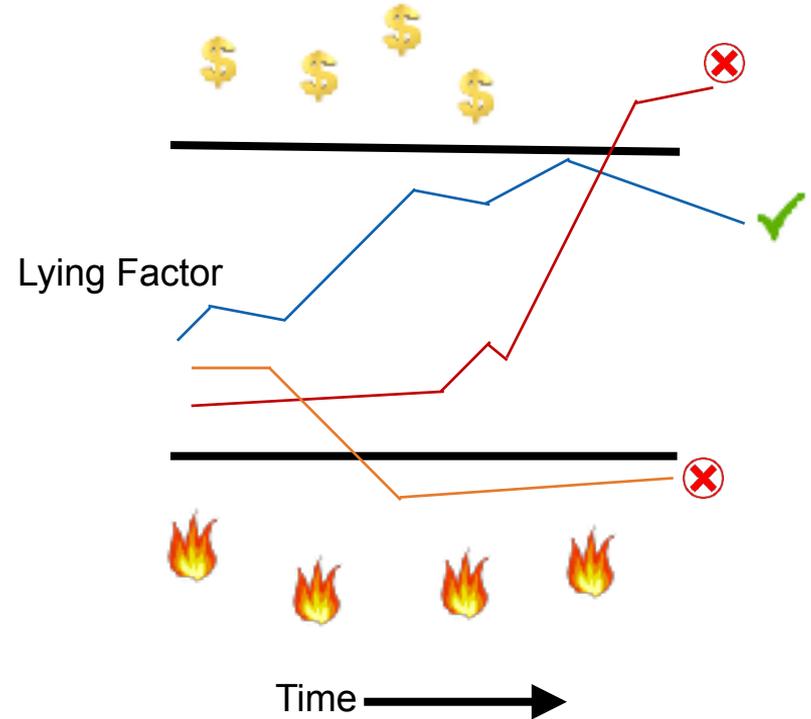


Dynamic Load (\propto Incoming data stream)

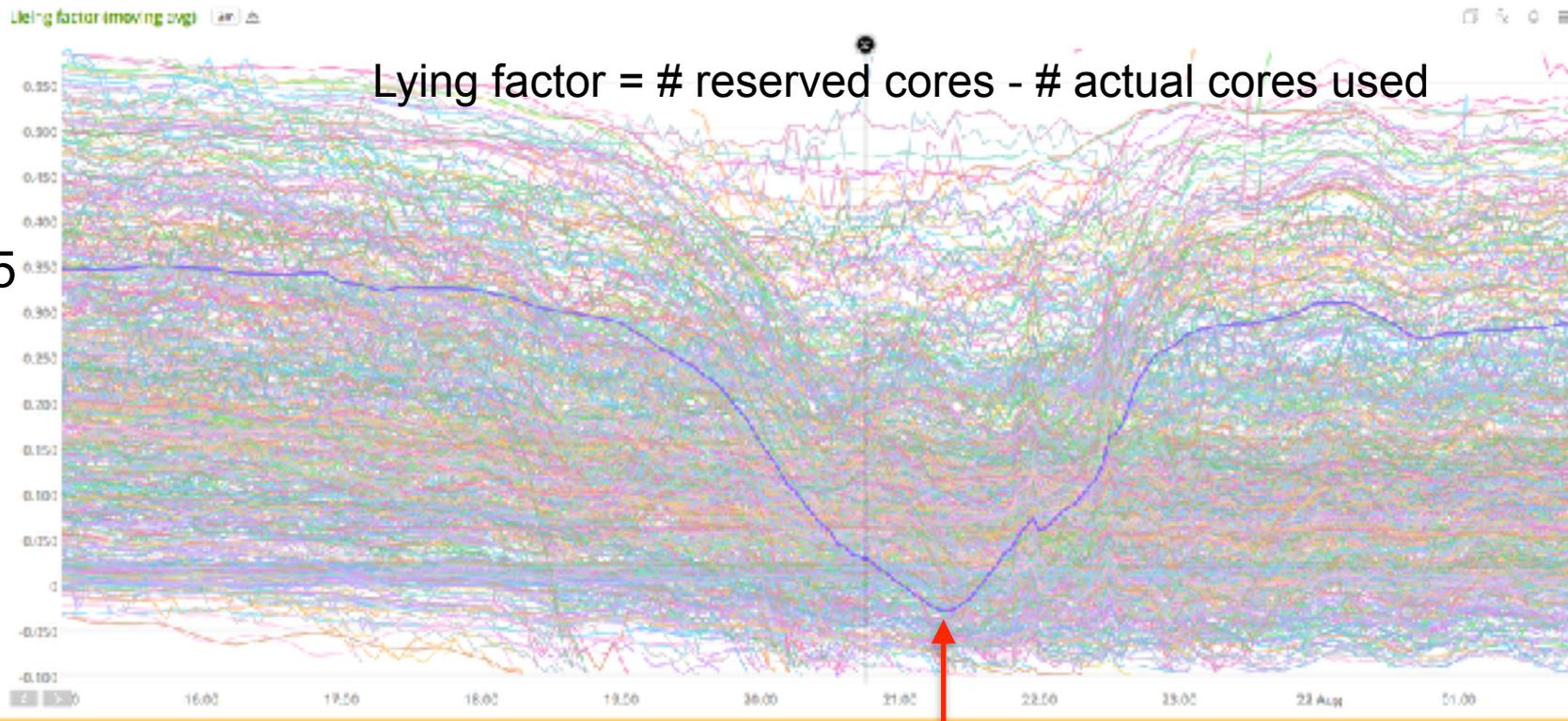


Autoscaling Using Lying Factor

- Changing load common in data streams
- Complicated due to changing load:
 - Seasonality (daily peaks)
 - Trend (load increasing overtime)
- Autoscaling based on lying factor for containers
 - Reduce resources when Lying factor crosses a positive threshold
 - Increase resources when Lying factor below a number
- Maintain Lying factor within a band



Lying Factor vs Load



Lying factor = # reserved cores - # actual cores used

0.35

Upper threshold: 0.6

Peak Load

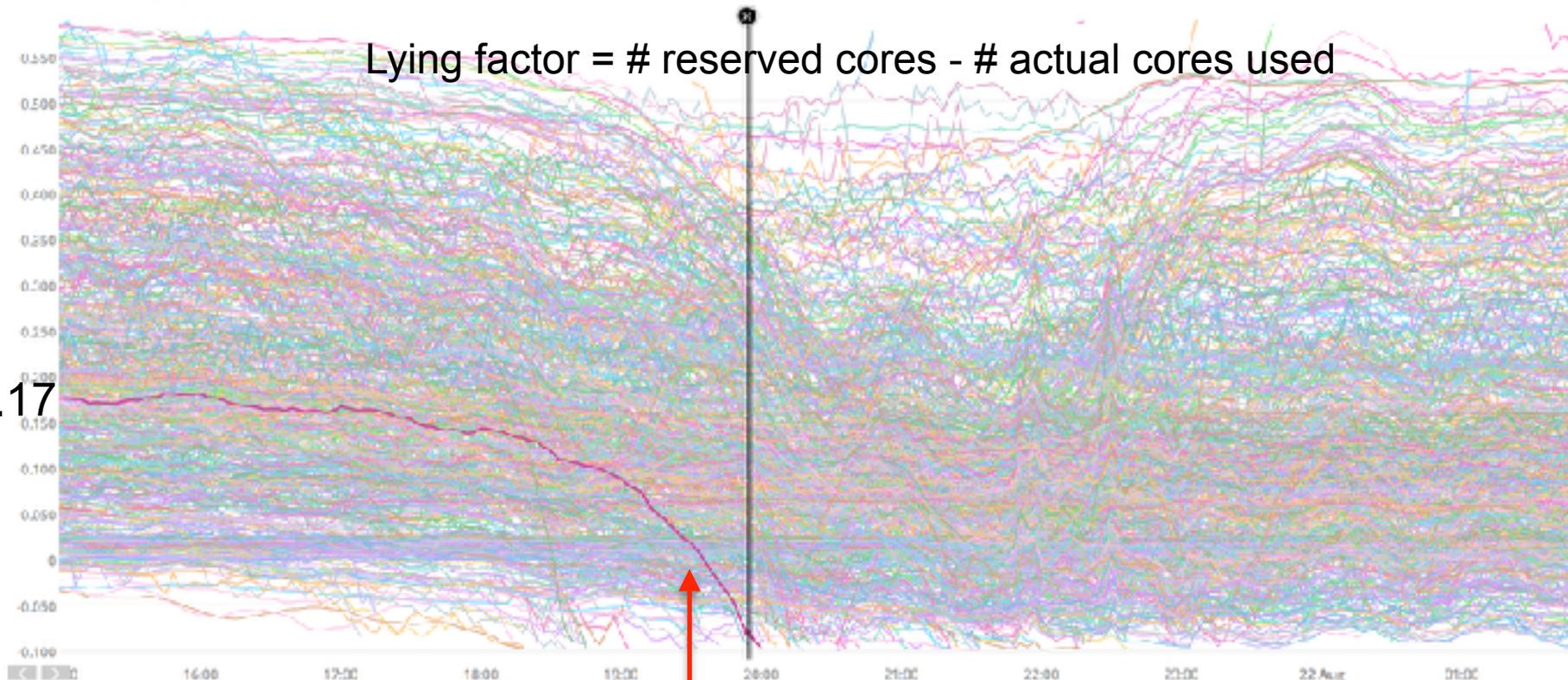
Upper threshold: -0.1

Containers Up-sized Automatically

Lying factor (moving avg)

Lying factor = # reserved cores - # actual cores used

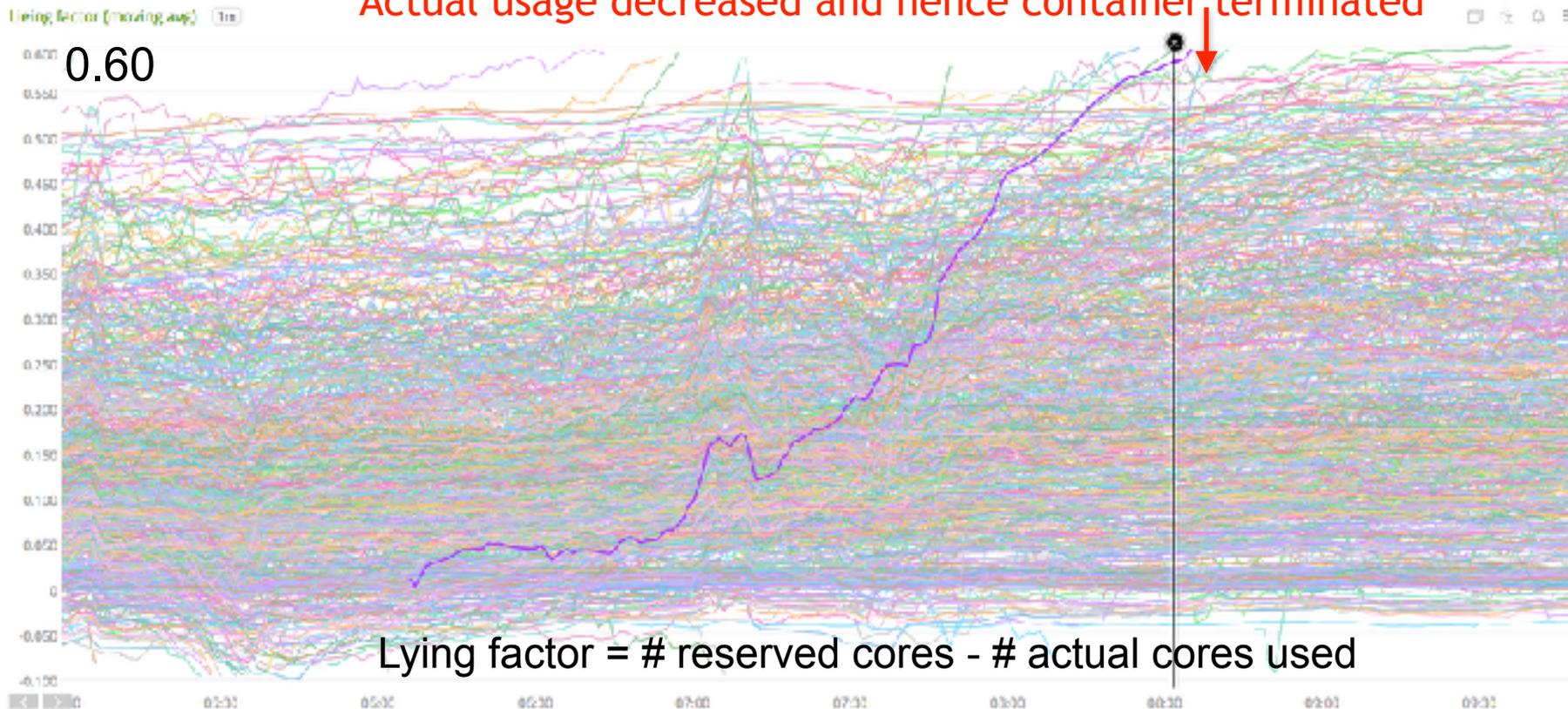
0.17



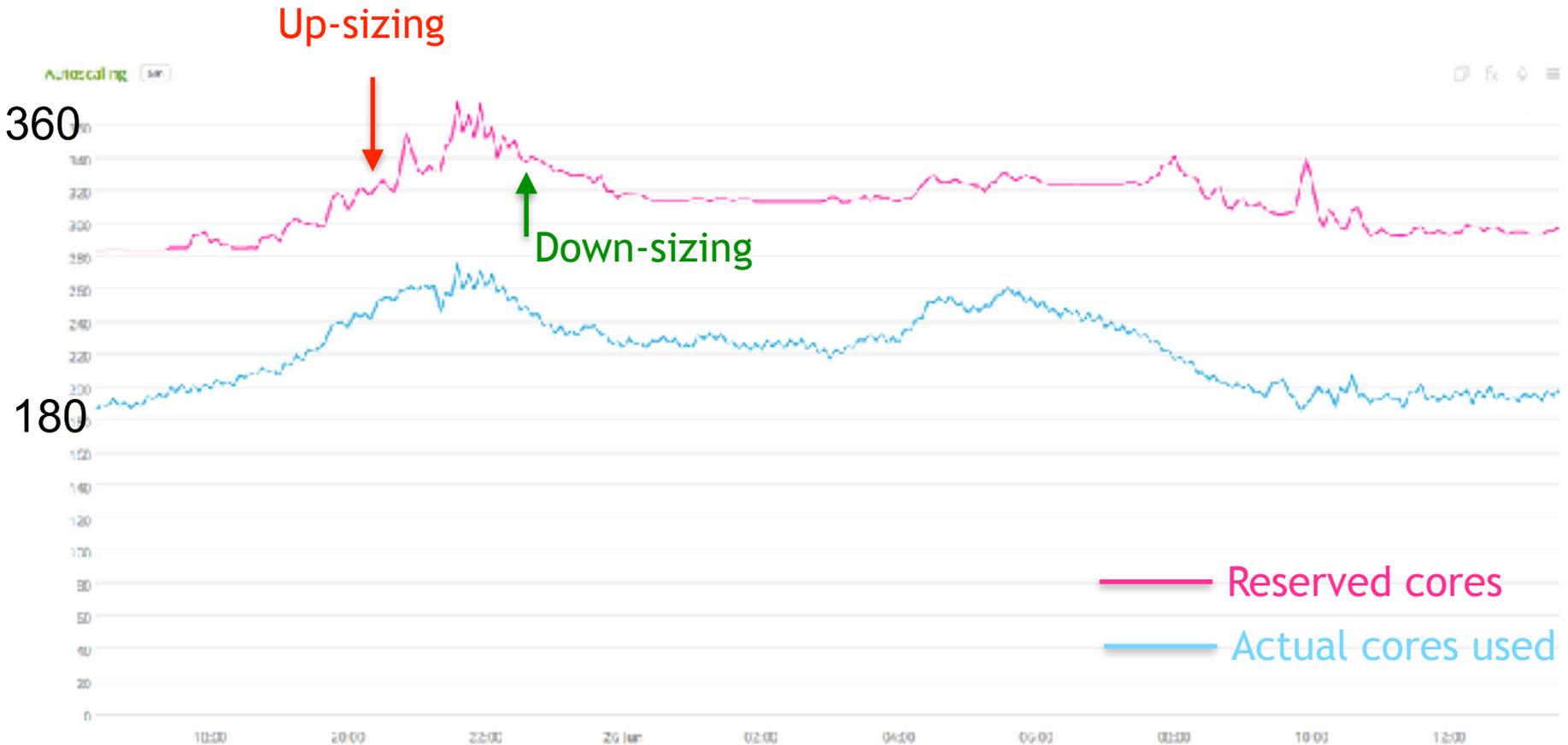
Actual usage increasing and hence container terminated

Container Down-sized Automatically

Actual usage decreased and hence container terminated



Autoscaling in Action: Reserved vs Actual Usage



What next?

- Using Spot instances for:
 - Kafka
 - Cassandra
- Intelligently and automatically selecting Spot instance types:
 - Least cost
 - Least volatile
- Autoscaling the remaining applications

- Site Reliability Engineer
 - Please email: osman@mist.com
 - Contact me on LinkedIn (Osman Sarood)
 - Fresh position hence not on the website yet 😊

- Data Scientist
 - Please email: osman@mist.com
 - Contact me on LinkedIn (Osman Sarood)
 - On the website (www.mist.com/careers)

Questions

