AKAMAS

Let the machines optimize the machines: ML-driven automated performance tuning

Conf42 SRE

Stefano Doni (Akamas)

AKAMAS

Agenda

1. Why SREs should care about system configurations
2. A new approach: ML-driven performance tuning
3. Real-world example: optimize Kubernetes and JVM
4. Conclusions

Stefano Doni

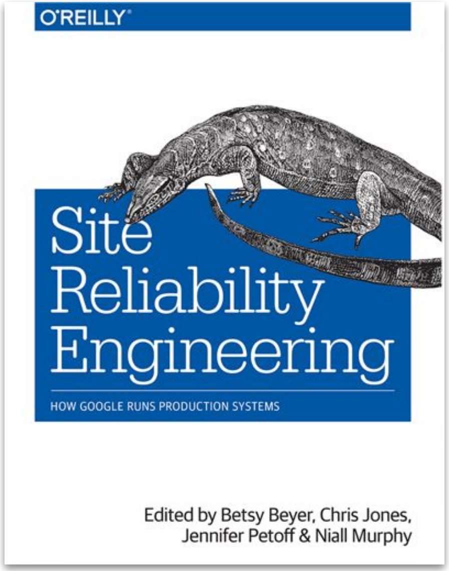
CTO at Akamas

15 years in performance engineering

2015 CMG Best Paper Award Winner

Why SREs should care about system configurations

SREs care about efficiency and performance



O'REILLY

Site Reliability Engineering

HOW GOOGLE RUNS PRODUCTION SYSTEMS

Edited by Betsy Beyer, Chris Jones,
Jennifer Petoff & Niall Murphy

https://sre.google/books

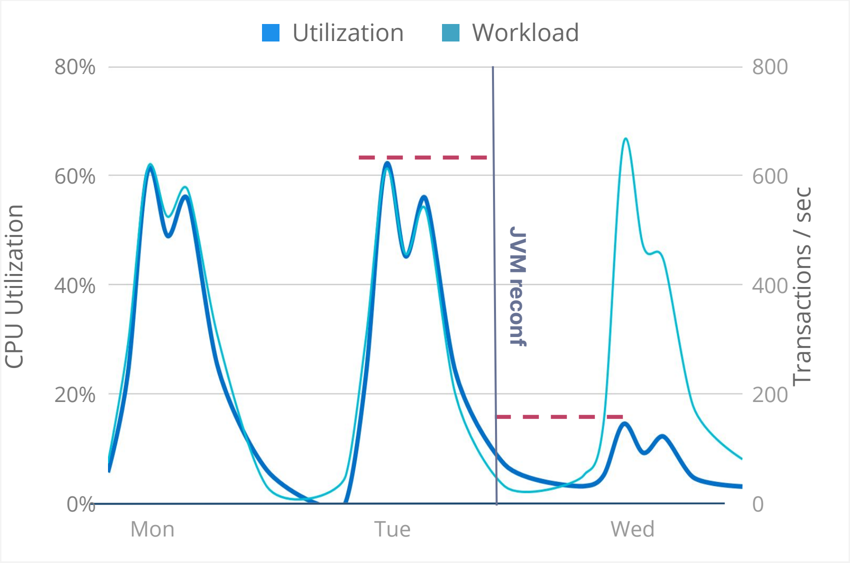
*"an **SRE team** is responsible for the availability, latency, performance, efficiency, change management, monitoring, emergency response, and capacity planning of their service(s)"*

The **core SRE tenets** include:

- Pursuing maximum change velocity without violating SLOs
- Demand Forecasting and Capacity Planning
- Efficiency and performance

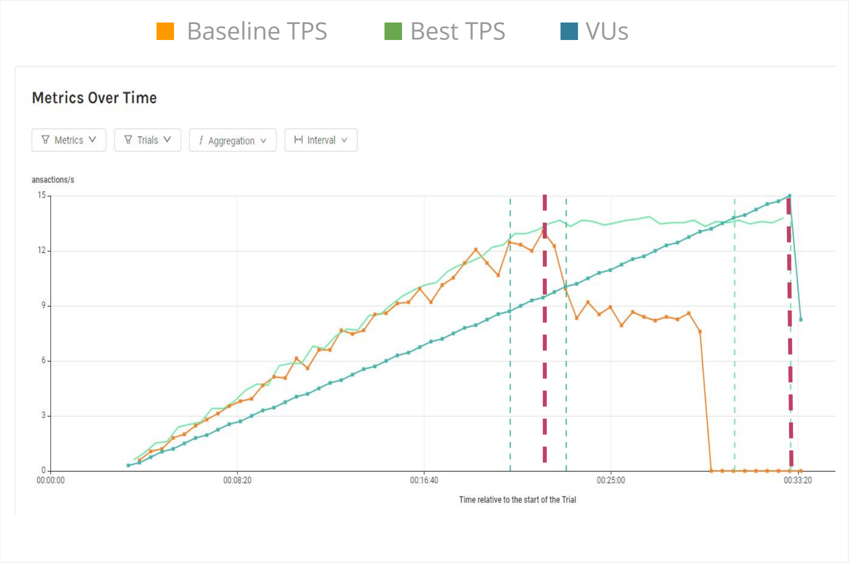
Tuning system configuration matters...

performance and efficiency



higher application performance and lower infrastructure cost

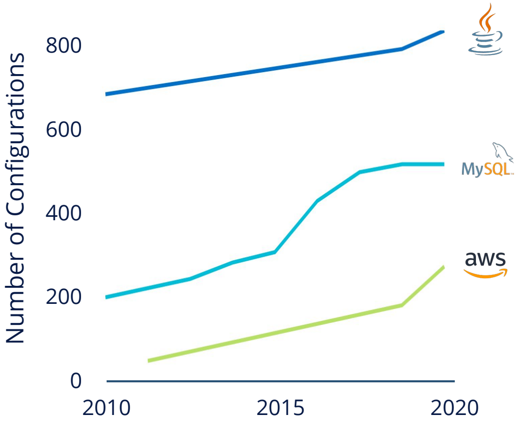
... and service availability



higher transaction throughput and improved service resilience

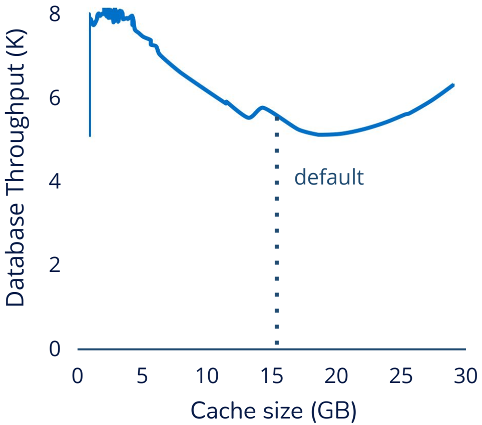
… but it is getting harder and harder

Configuration Explosion



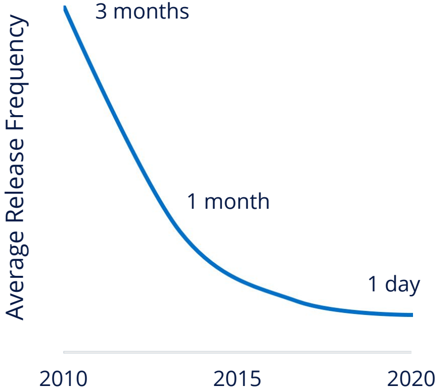
properly configuring the IT stack requires analyzing thousands of configurations

Unpredictable Effects



effect of changes can be counterintuitive + default values not always appropriate

Faster Deployments



acceleration of release pace makes manual approach infeasible/useless

A new approach:
ML-driven performance tuning

Akamas ML-driven optimization

AKAMAS

Full-Stack

any application, any middleware, any database, any cloud - any system

Smart Exploration

patented ML quickly identifying optimal configurations beyond any manual tuning

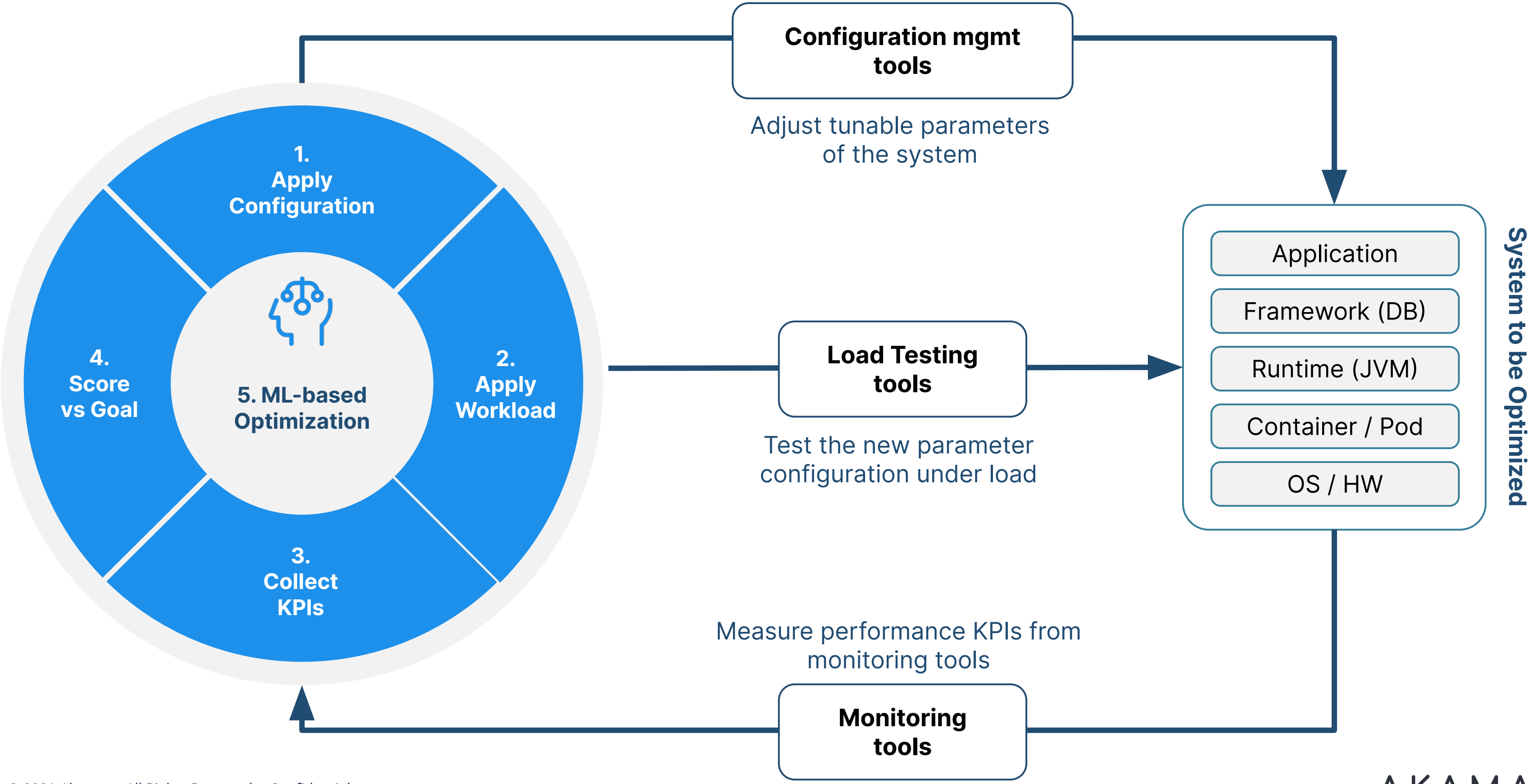
Goal-oriented

custom-defined goals translating SLOs and business & technical constraints

Fully Automated

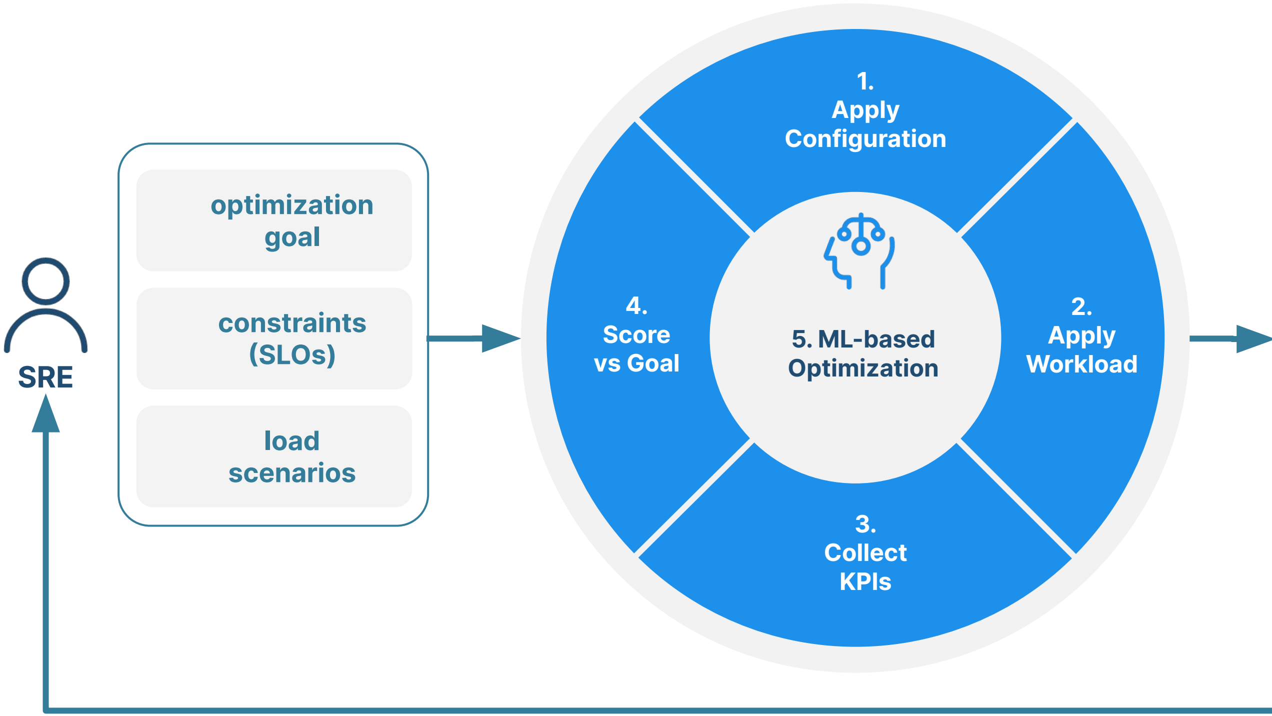
custom workflows automating parameter changes, load testing & telemetry collection

ML enables automated performance tuning...

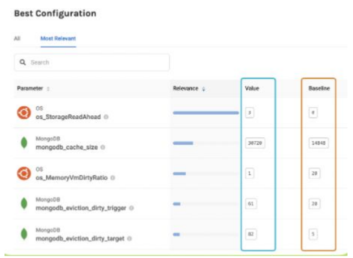


AKAMAS

… and a new performance tuning process



optimal configuration



Best Configuration

All | Most Relevant

Search

Parameter	Relevance	Value	Baseline
OS os_StorageReadAhead		5	0
MongoDB mongodb_cache_size		30720	14848
OS os_MemoryVmDirtyRatio		1	20
MongoDB mongodb_eviction_dirty_trigger		61	20
MongoDB mongodb_eviction_dirty_target		82	5

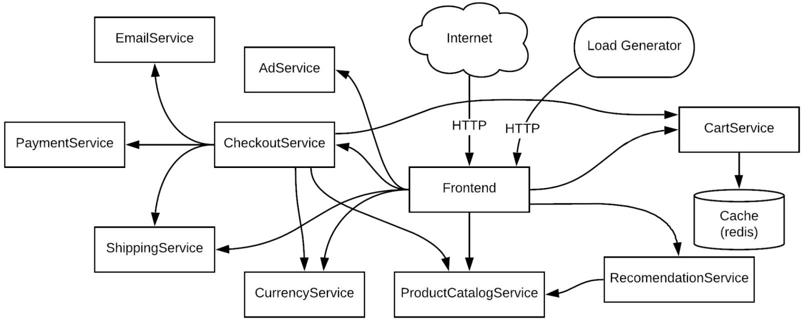
AKAMAS

Real world example: optimize Kubernetes and JVM

AKAMAS

The target system: Online Boutique

- **Cloud-native application** by Google made of **10 microservices**
- Realistic sample web-based **e-commerce service**
- Features a **modern software stack** (Go, Node.js, Java, Python, Redis)
- Includes a Load Generator (Locust) to inject **realistic workloads**



https://github.com/GoogleCloudPlatform/microservices-demo

Use Case: optimizing cost of K8s microservices while ensuring reliability

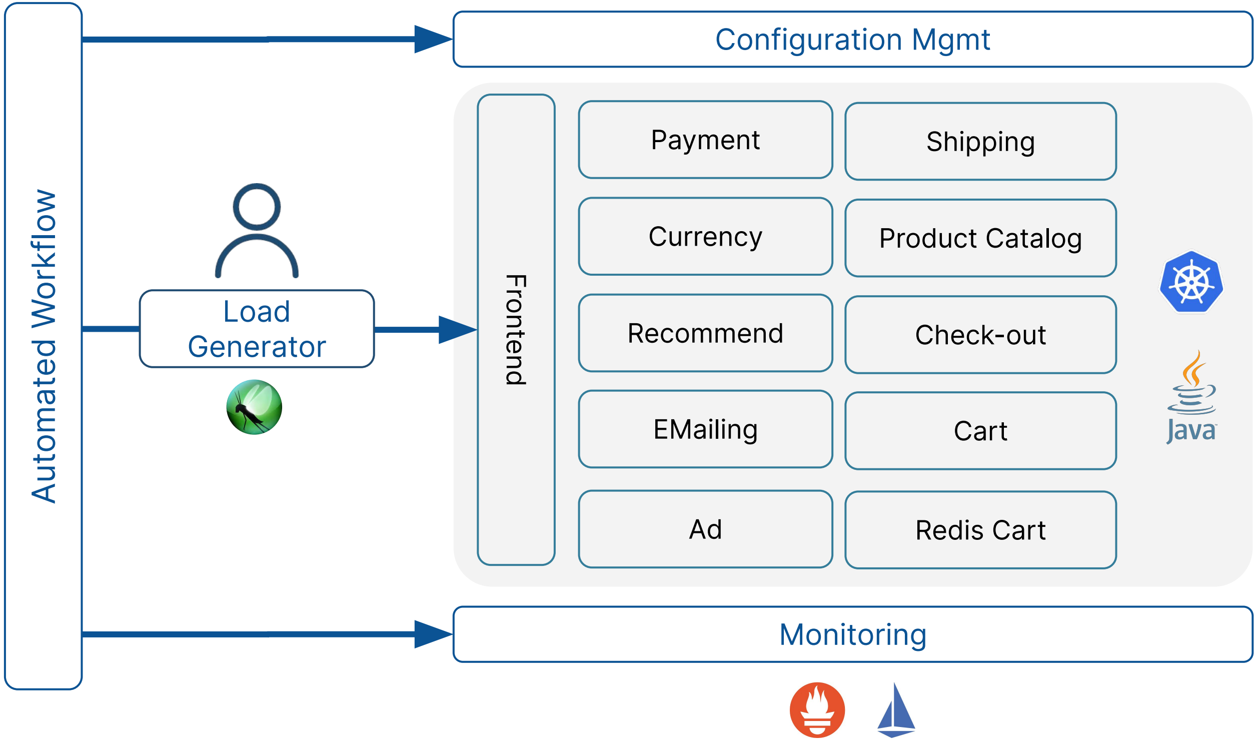
SRE

Challenge for SRE

How to provision the optimal resources to your application made of several **Kubernetes** microservices, so that you can trust the overall service

- ➔ will sustain the expected **target load**
- ➔ while matching the defined **Service-Level Objectives** (SLOs)
- ➔ at the **minimum cost**
- ➔ while minimizing the operational effort
- ➔ and matching delivery milestones

The reference architecture



22
TUNABLE PARAMETERS (CPU & Memory limits)

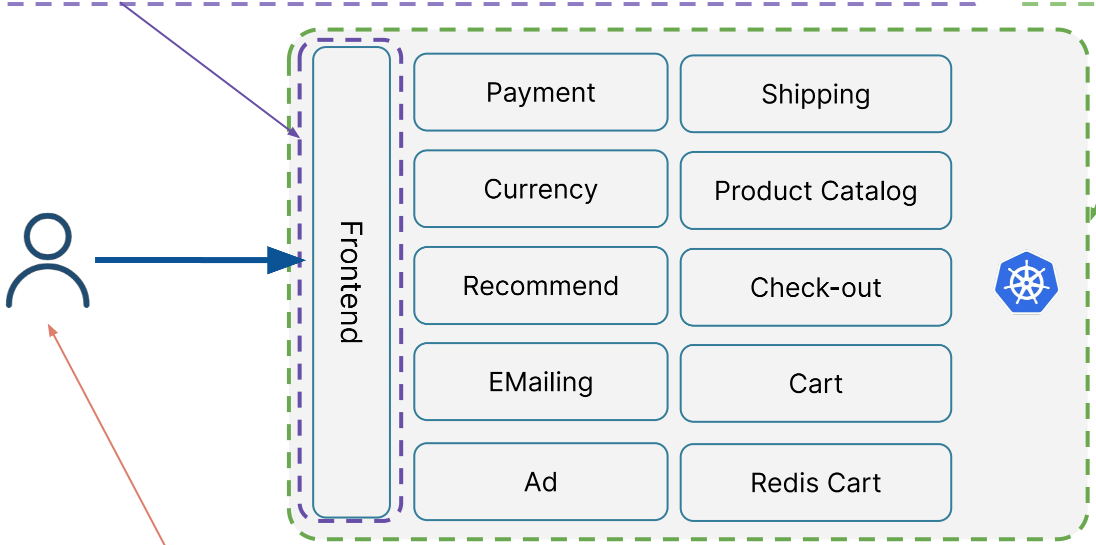
10
MICROSERVICES

275
MEASURED KPIs

AKAMAS

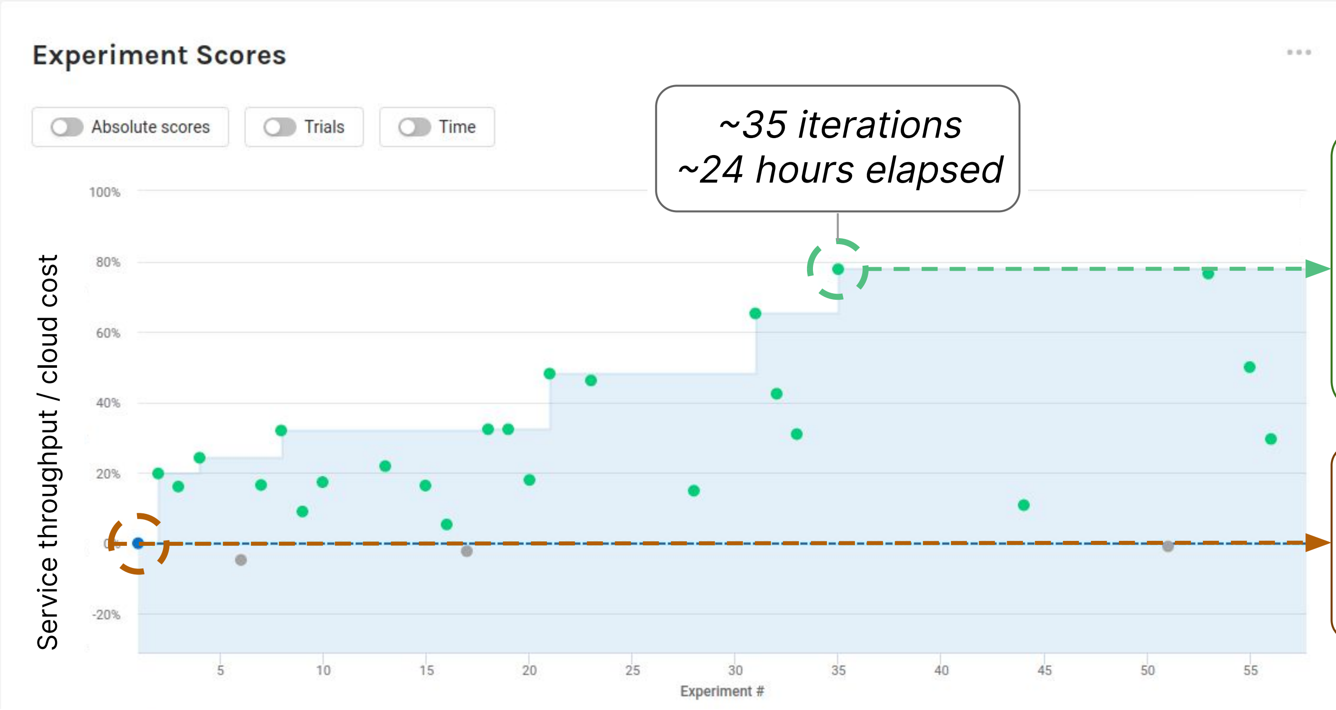
The optimization goals & constraints

GOAL:
MAXIMIZE frontend_boutique.istio_incoming_success_transactions / application_cost



CONSTRAINTS: loadgenerator_locust.locust_fail_ratio <= 2% **AND**
frontend_boutique.istio_incoming_response_time_90pct <= 500ms

Best configuration found by ML in 24H improves cost efficiency by 77%



~35 iterations
~24 hours elapsed

Best configuration

Perf/Cost:
***0.52** TPS/$/mo*

+77%

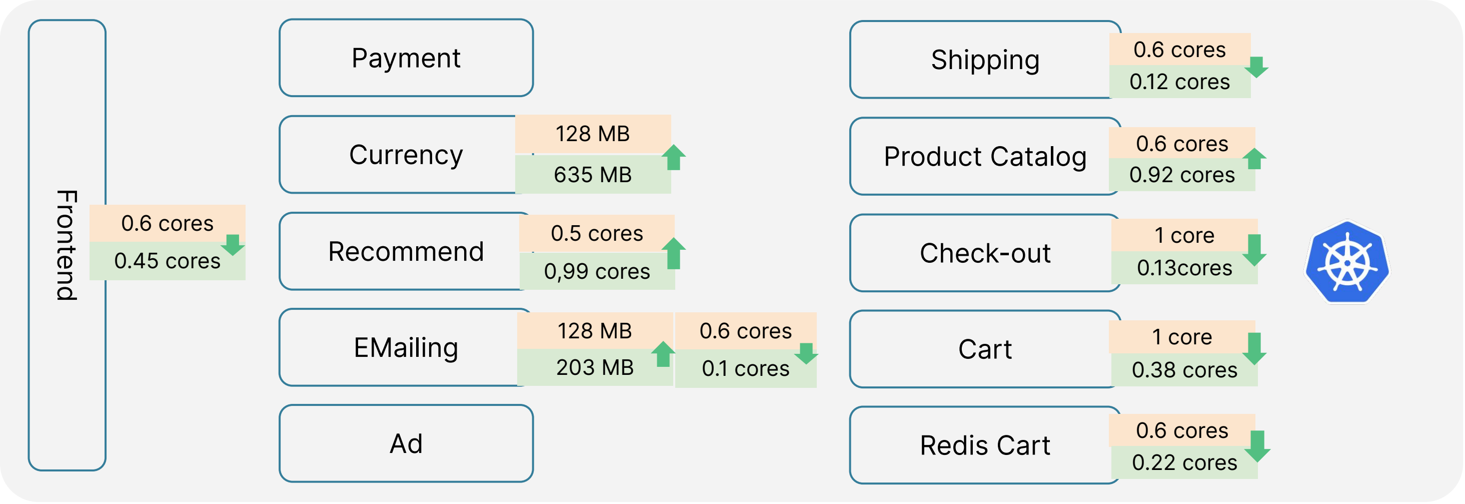
Baseline configuration

Perf/Cost:
***0.29** TPS/$/mo*

Best config: optimal resources assigned to microservices

10
TOP IMPACT PARAMETERS

Baseline
Best



Frontend: 0.6 cores → 0.45 cores ↓

Payment

Currency: 128 MB → 635 MB ↑

Recommend: 0.5 cores → 0,99 cores ↑

EMailing: 128 MB → 203 MB ↑; 0.6 cores → 0.1 cores ↓

Ad

Shipping: 0.6 cores → 0.12 cores ↓

Product Catalog: 0.6 cores → 0.92 cores ↑

Check-out: 1 core → 0.13cores ↓

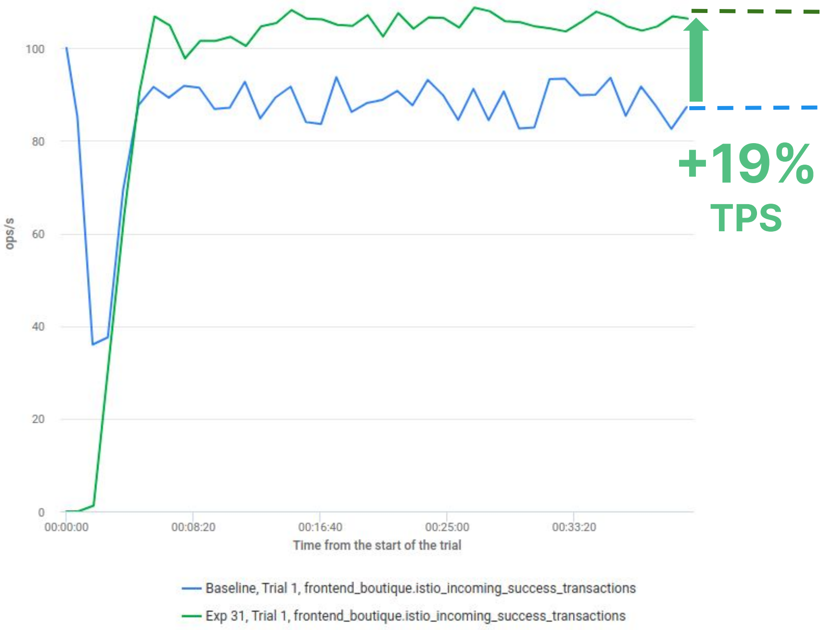
Cart: 1 core → 0.38 cores ↓

Redis Cart: 0.6 cores → 0.22 cores ↓

- decreased CPU limits set for almost all containers
- increased CPU assigned to 2 microservices
- all these changes to achieve max cost efficiency and match SLOs

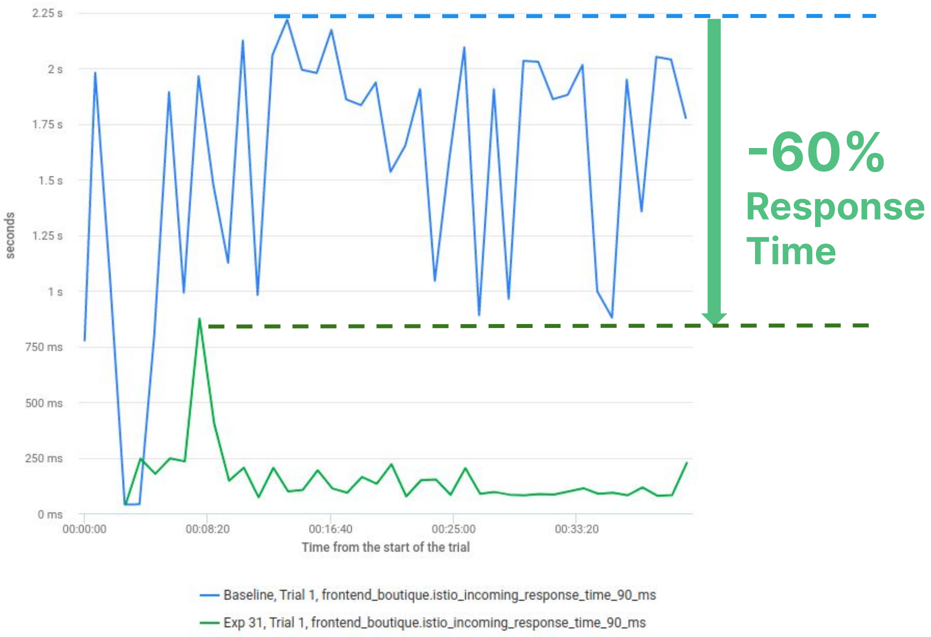
Best config: higher performance & efficiency for the overall service

Baseline vs Best: Service throughput



+19%
TPS

Baseline vs Best: Service p90 response time



-60%
Response Time

Use Case: maximizing service performance & efficiency with JVM tuning

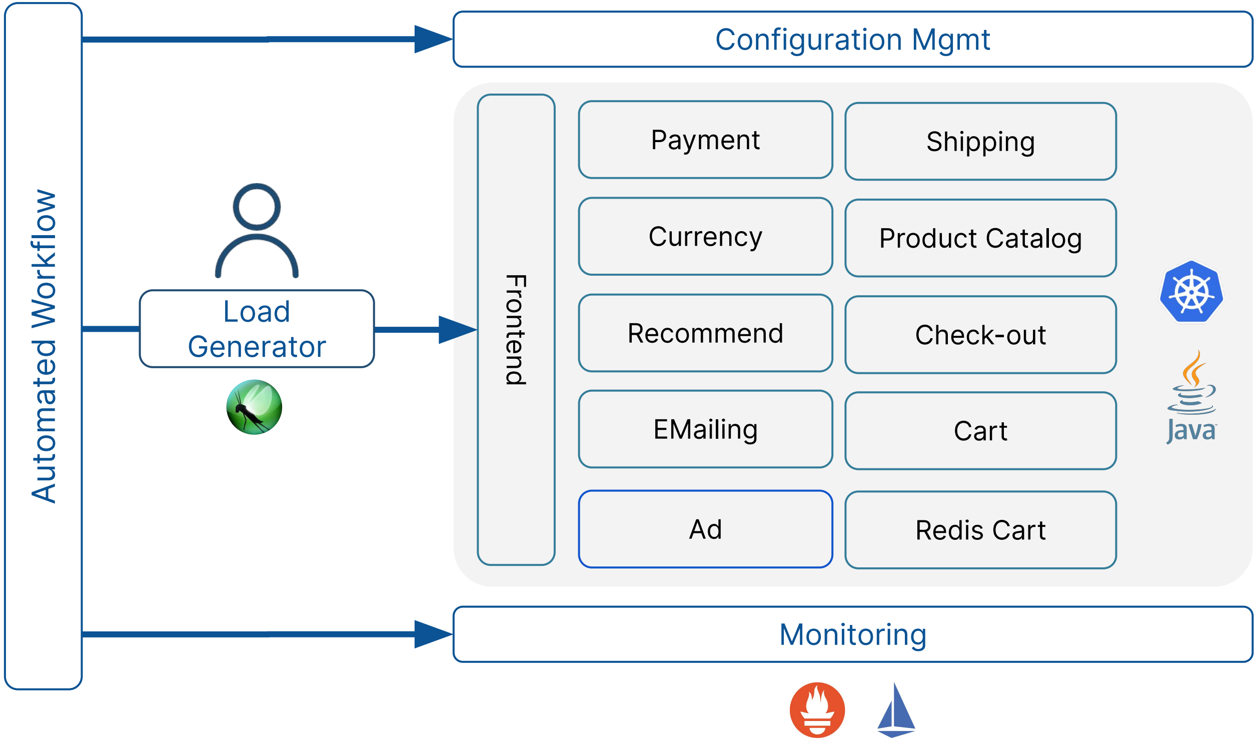
SRE

Challenge for SRE

How to ensure a reliable product launch, by properly configuring JVM options, so that you can trust the overall service

- will sustain the expected **target load**
- while matching the defined **Service-Level Objectives** (SLO)
- at the **minimum cost**
- while minimizing the operational effort
- and staying aligned product launch milestones

The reference architecture



32
TUNABLE PARAMETERS (JVM options)

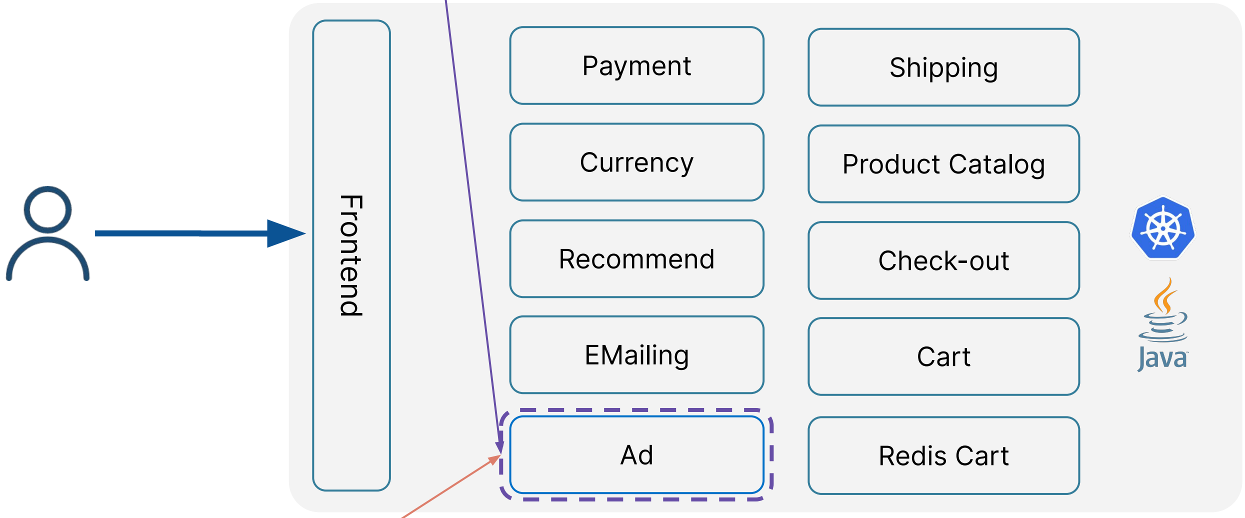
10
MICROSERVICES

275
MEASURED KPIs

AKAMAS

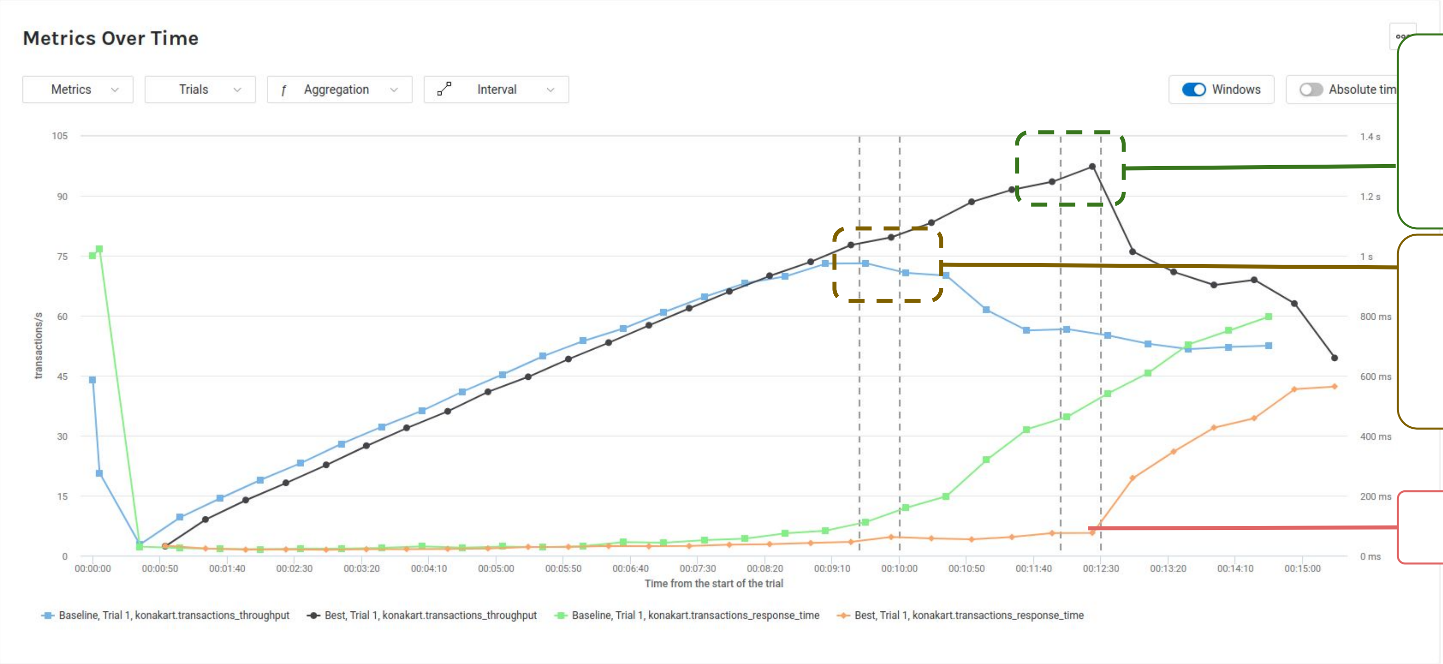
The optimization goals & constraints

GOAL:
MAXIMIZE ad.istio_incoming_success_transactions



CONSTRAINTS: ad.transaction_response_time <= 100ms

Best config:
+28% throughput, and meeting SLOs



Best configuration

+28%

*Peak Throughput matching SLO: **95** TPS*

Baseline configuration

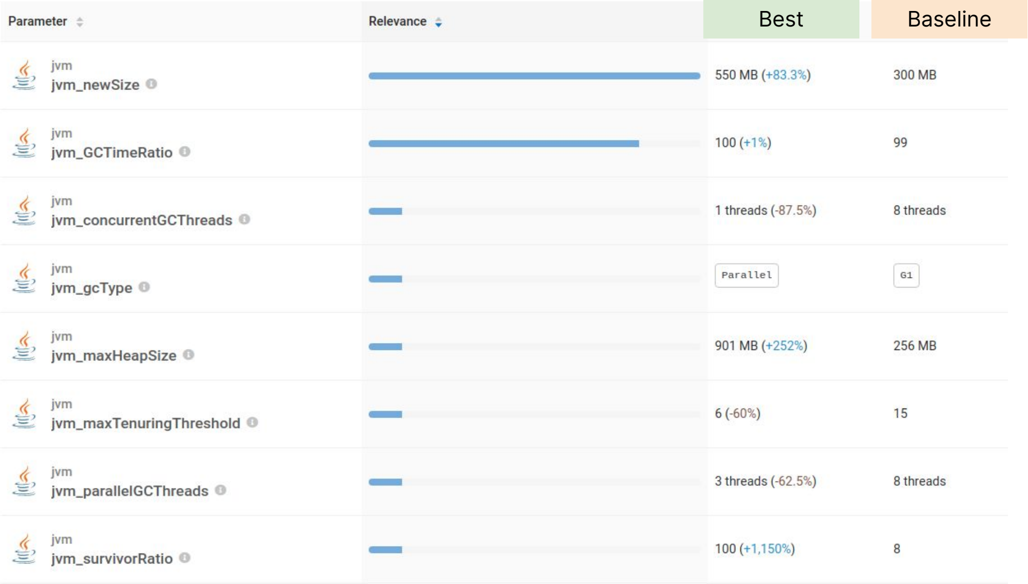
*Peak Throughput matching SLO: **74** TPS*

***SLO** breaking at 100ms*

Best config: optimal JVM options

8

TOP IMPACT PARAMETERS



Parameter	Relevance	Best	Baseline
jvm jvm_newSize		550 MB (+83.3%)	300 MB
jvm jvm_GCTimeRatio		100 (+1%)	99
jvm jvm_concurrentGCThreads		1 threads (-87.5%)	8 threads
jvm jvm_gcType		Parallel	G1
jvm jvm_maxHeapSize		901 MB (+252%)	256 MB
jvm jvm_maxTenuringThreshold		6 (-60%)	15
jvm jvm_parallelGCThreads		3 threads (-62.5%)	8 threads
jvm jvm_survivorRatio		100 (+1,150%)	8

- increased max heap memory
- changed Garbage Collector type
- decreased number of Garbage Collector threads
- adjusted heap regions & object aging thresholds

Conclusions

Key takeaways

Tuning modern applications for increasing their efficiency, performance and reliability is a **complex problem** that represent a **relevant toil** for SRE teams

A new approach leveraging fully-automated **ML-based optimization** enables SRE teams to ensure applications will have **higher performance & reliability**

Leveraging this new **ML-based optimization** approach, SRE teams can **reduce the operational toil** and **stay aligned to release milestones**

Contacts

Italy HQ
Via Schiaffino 11
Milan, 20158
+39-02-4951-7001

USA East
211 Congress Street
Boston, MA 02110
+1-617-936-0212

USA West
12655 W. Jefferson Blvd
Los Angeles, CA 90066
+1-323-524-0524

Singapore
5 Temasek Blvd
Singapore 038985

LinkedIn

@akamaslabs

Twitter

@AkamasLabs

Email
info@akamas.io

BACKUP SLIDES