keptn

SRE CON AMERICAS

Automatically Detect the Performance & Scalability Issues in Distributed Architectures

"And integrate this in your delivery pipeline with keptn*"*

Andreas Grabner

DevOps Activist at Dynatrace

DevRel for Keptn

@grabnerandi, https://www.linkedin.com/in/grabnerandi

keptn

CLOUD NATIVE COMPUTING FOUNDATION
We are a Cloud Native Computing Foundation Sandbox project.

Follow us @keptnProject

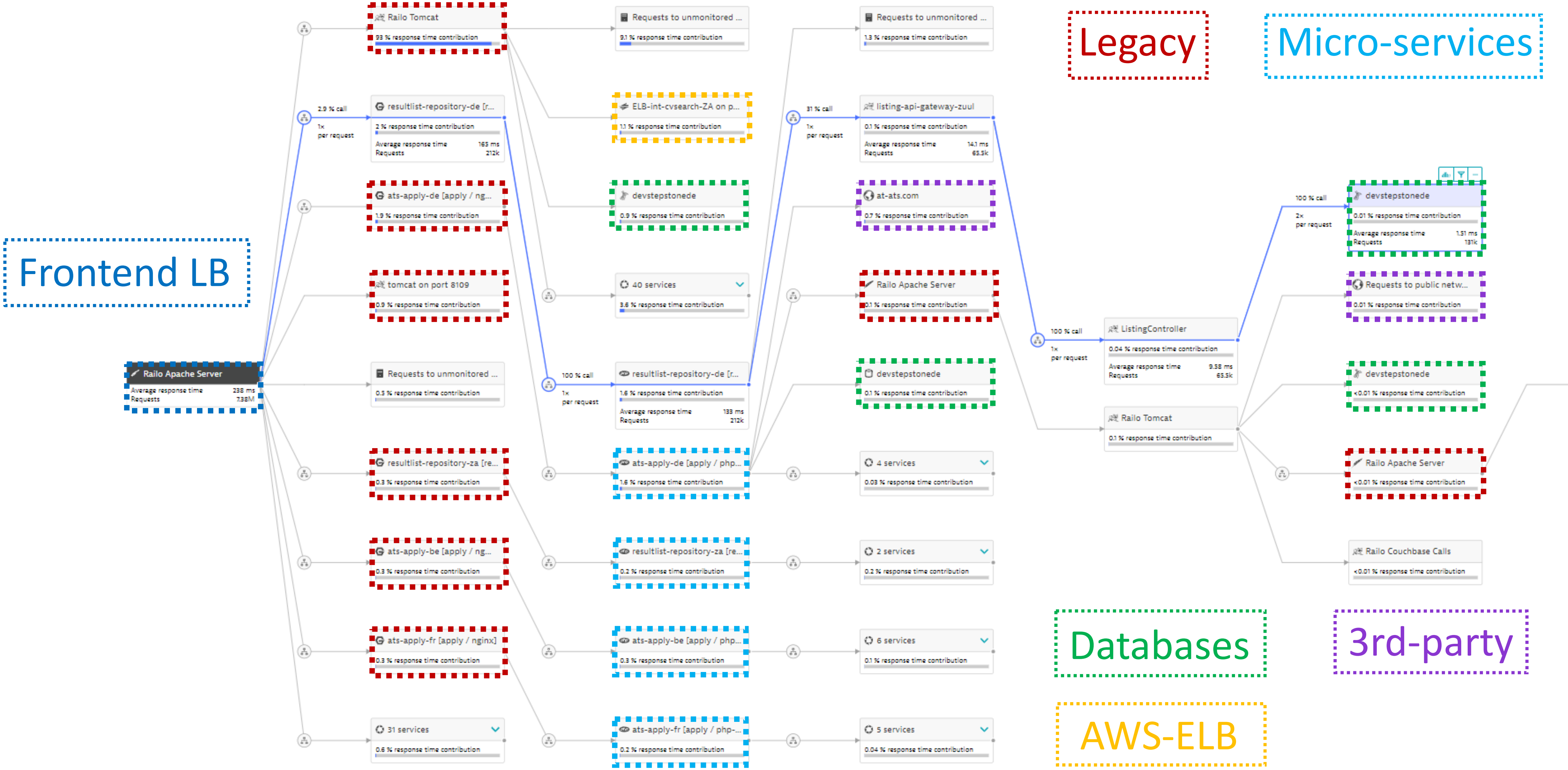
Star us @ https://github.com/keptn/keptn

Slack Us @ https://slack.keptn.sh

How distributed systems look like!

Or how they shouldn‘t ...

Distributed Trace Example from StepStone (AWS Summit Berlin 2019)


Railo Apache Server
Average response time 238 ms
Requests 7.38M
Railo Tomcat
93 % response time contribution
2.9 % call
1x per request
resultlist-repository-de [r...
2 % response time contribution
Average response time 165 ms
Requests 212k
ats-apply-de [apply / ng...
1.9 % response time contribution
tomcat on port 8109
0.9 % response time contribution
Requests to unmonitored ...
0.5 % response time contribution
resultlist-repository-za [re...
0.3 % response time contribution
ats-apply-be [apply / ng...
0.3 % response time contribution
ats-apply-fr [apply / nginx]
0.3 % response time contribution
31 services
0.6 % response time contribution
Requests to unmonitored ...
9.1 % response time contribution
ELB-int-cvsearch-ZA on p...
1.1 % response time contribution
devstepstonede
0.9 % response time contribution
40 services
3.6 % response time contribution
100 % call
1x per request
resultlist-repository-de [r...
1.6 % response time contribution
Average response time 133 ms
Requests 212k
ats-apply-de [apply / php...
1.6 % response time contribution
resultlist-repository-za [re...
0.2 % response time contribution
ats-apply-be [apply / php...
0.3 % response time contribution
ats-apply-fr [apply / php-...
0.2 % response time contribution
Requests to unmonitored ...
1.3 % response time contribution
31 % call
1x per request
listing-api-gateway-zuul
0.1 % response time contribution
Average response time 14.1 ms
Requests 65.5k
at-ats.com
0.7 % response time contribution
Railo Apache Server
0.1 % response time contribution
devstepstonede
0.1 % response time contribution
4 services
0.03 % response time contribution
2 services
0.2 % response time contribution
6 services
0.1 % response time contribution
5 services
0.04 % response time contribution
100 % call
1x per request
ListingController
0.04 % response time contribution
Average response time 9.58 ms
Requests 65.5k
Railo Tomcat
0.1 % response time contribution
100 % call
2x per request
devstepstonede
0.01 % response time contribution
Average response time 1.51 ms
Requests 131k
Requests to public netw...
0.01 % response time contribution
devstepstonede
<0.01 % response time contribution
Railo Apache Server
<0.01 % response time contribution
Railo Couchbase Calls
<0.01 % response time contribution

Frontend LB

Legacy

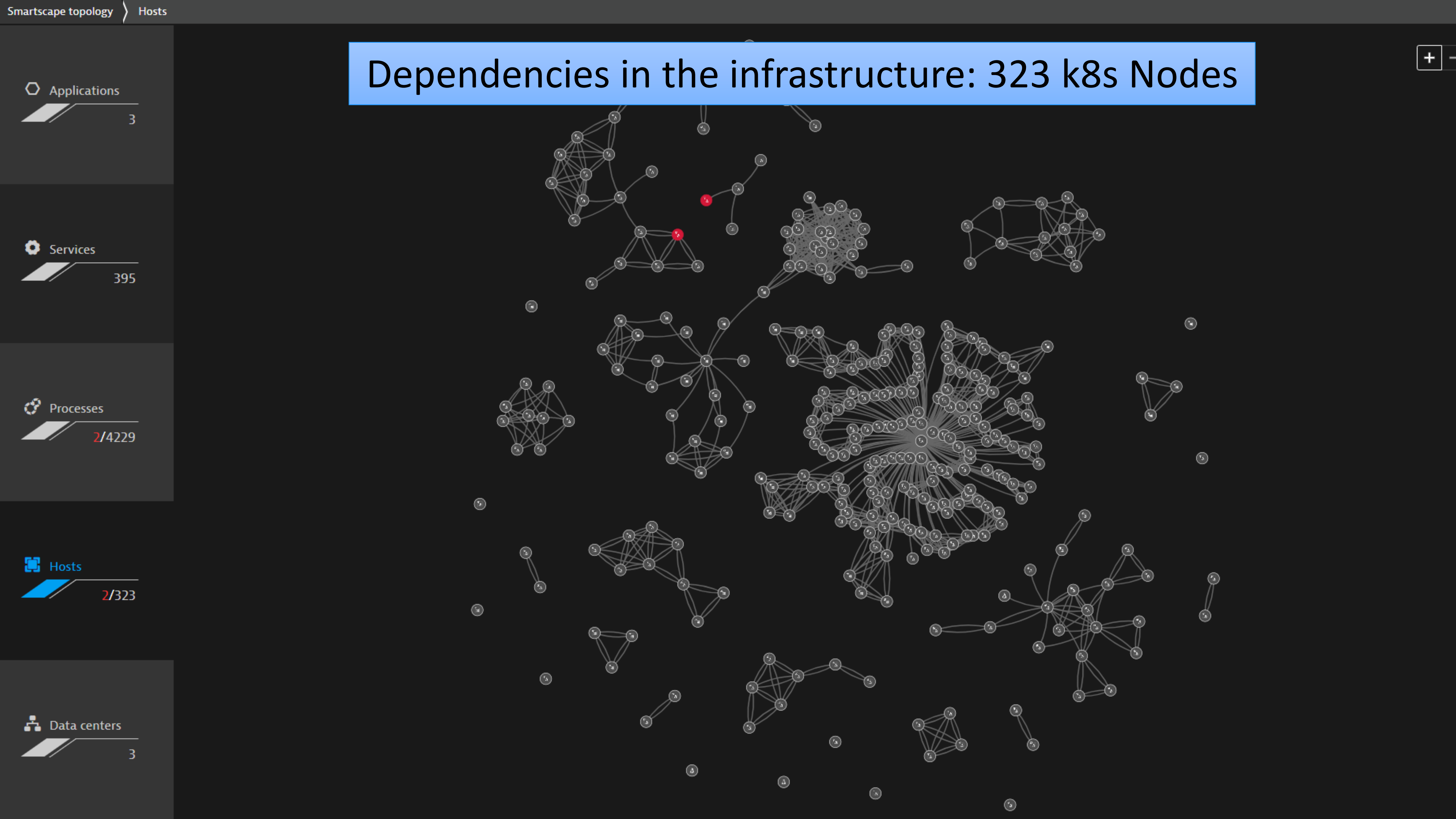
Micro-services

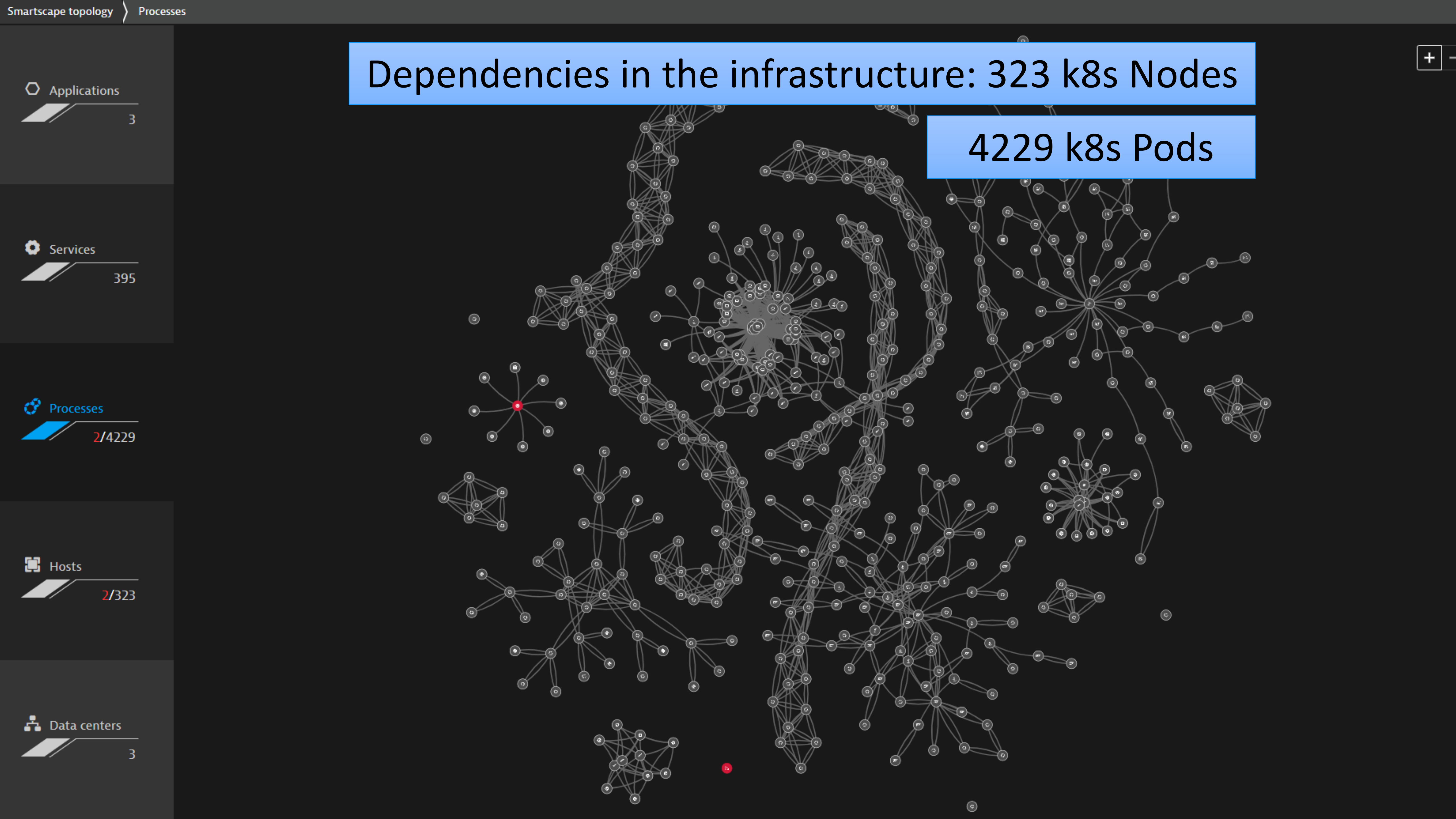
Databases

3rd-party

AWS-ELB

Dependencies in the infrastructure: 323 k8s Nodes

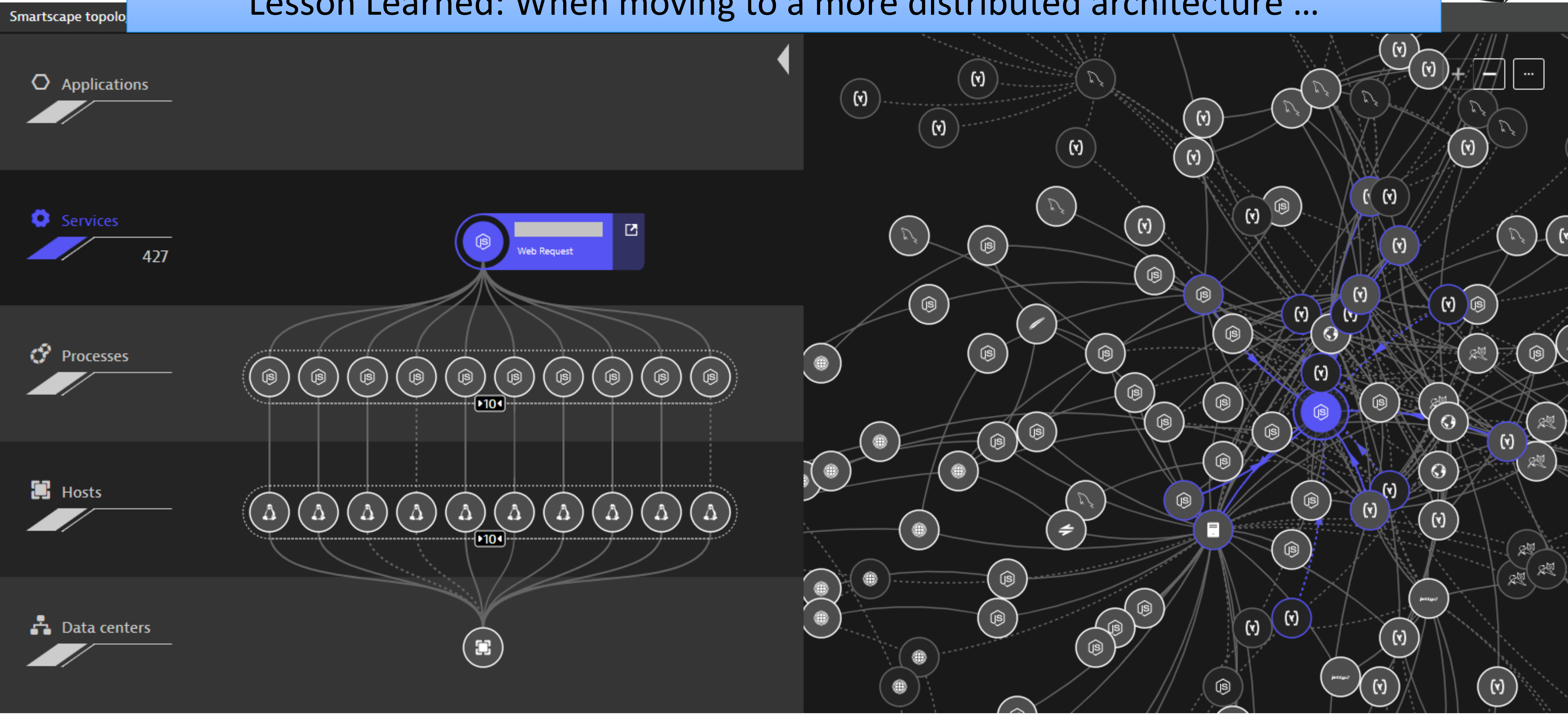




Dependencies in the infrastructure: 323 k8s Nodes

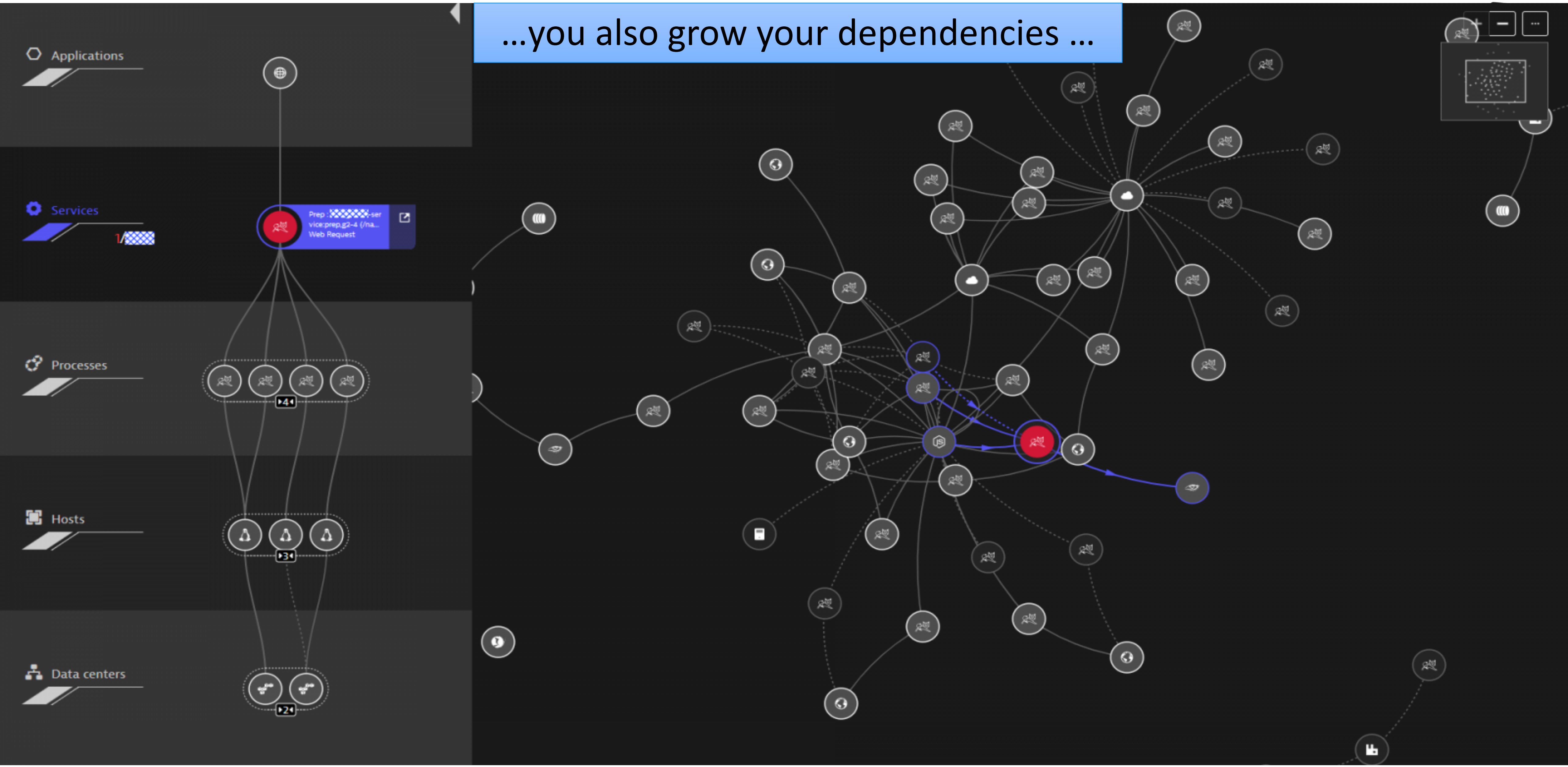
4229 k8s Pods

Lesson Learned: When moving to a more distributed architecture …



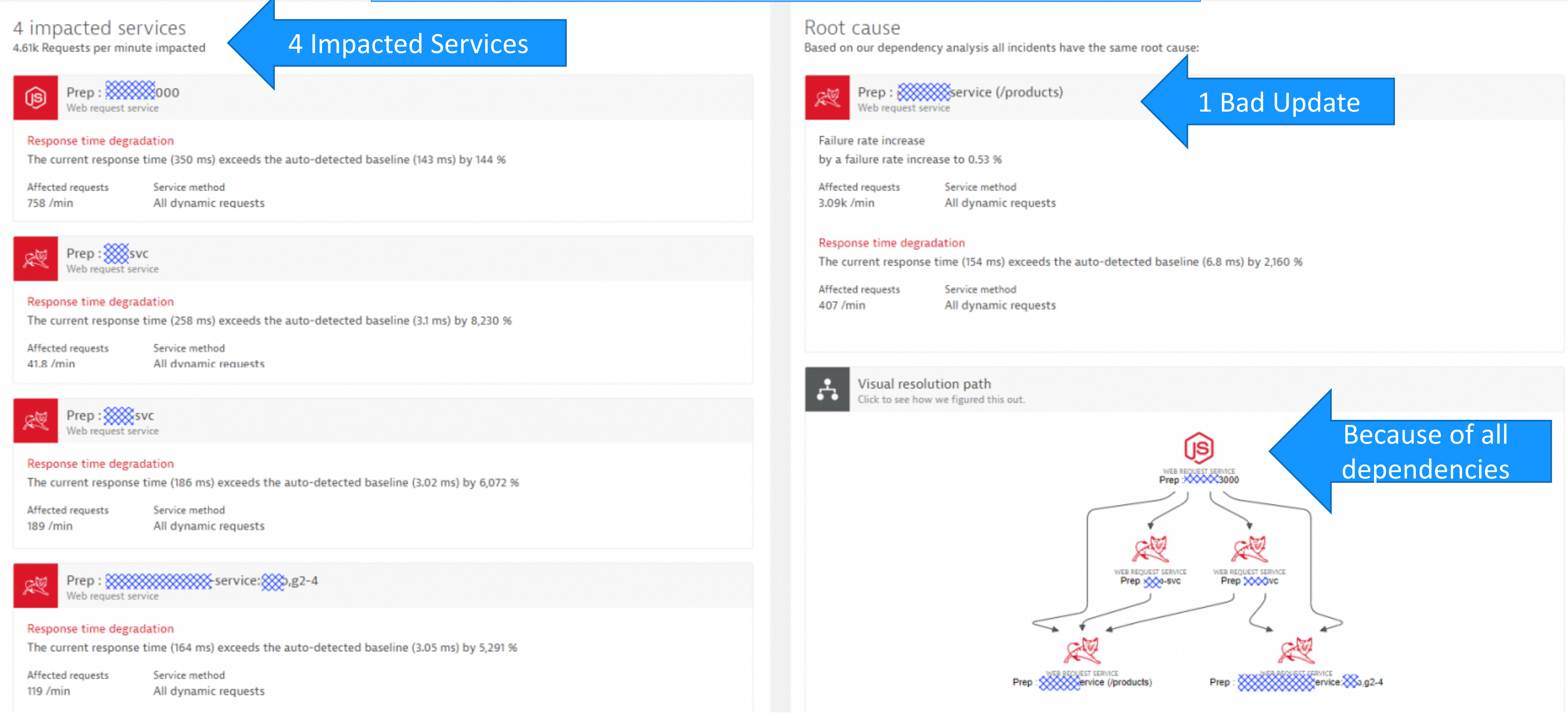
https://www.dynatrace.com/news/blog/monitoring-aws-fargate-with-dynatrace-testing-it-in-the-field/

…you also grow your dependencies …



https://www.dynatrace.com/news/blog/enterprise-cloud-ecs-microservices-and-dynatrace-at-neiman-marcus/

… and the potential impact of a failure grows!



4 Impacted Services

1 Bad Update

Because of all dependencies

4 impacted services

4.61k Requests per minute impacted

Prep : [redacted]000
Web request service

Response time degradation
The current response time (350 ms) exceeds the auto-detected baseline (143 ms) by 144 %

Affected requests	Service method
758 /min	All dynamic requests

Prep : [redacted]svc
Web request service

Response time degradation
The current response time (258 ms) exceeds the auto-detected baseline (3.1 ms) by 8,230 %

Affected requests	Service method
41.8 /min	All dynamic requests

Prep : [redacted]svc
Web request service

Response time degradation
The current response time (186 ms) exceeds the auto-detected baseline (3.02 ms) by 6,072 %

Affected requests	Service method
189 /min	All dynamic requests

Prep : [redacted]service:[redacted],g2-4
Web request service

Response time degradation
The current response time (164 ms) exceeds the auto-detected baseline (3.05 ms) by 5,291 %

Affected requests	Service method
119 /min	All dynamic requests

Root cause

Based on our dependency analysis all incidents have the same root cause:

Prep : [redacted]service (/products)
Web request service

Failure rate increase
by a failure rate increase to 0.53 %

Affected requests	Service method
3.09k /min	All dynamic requests

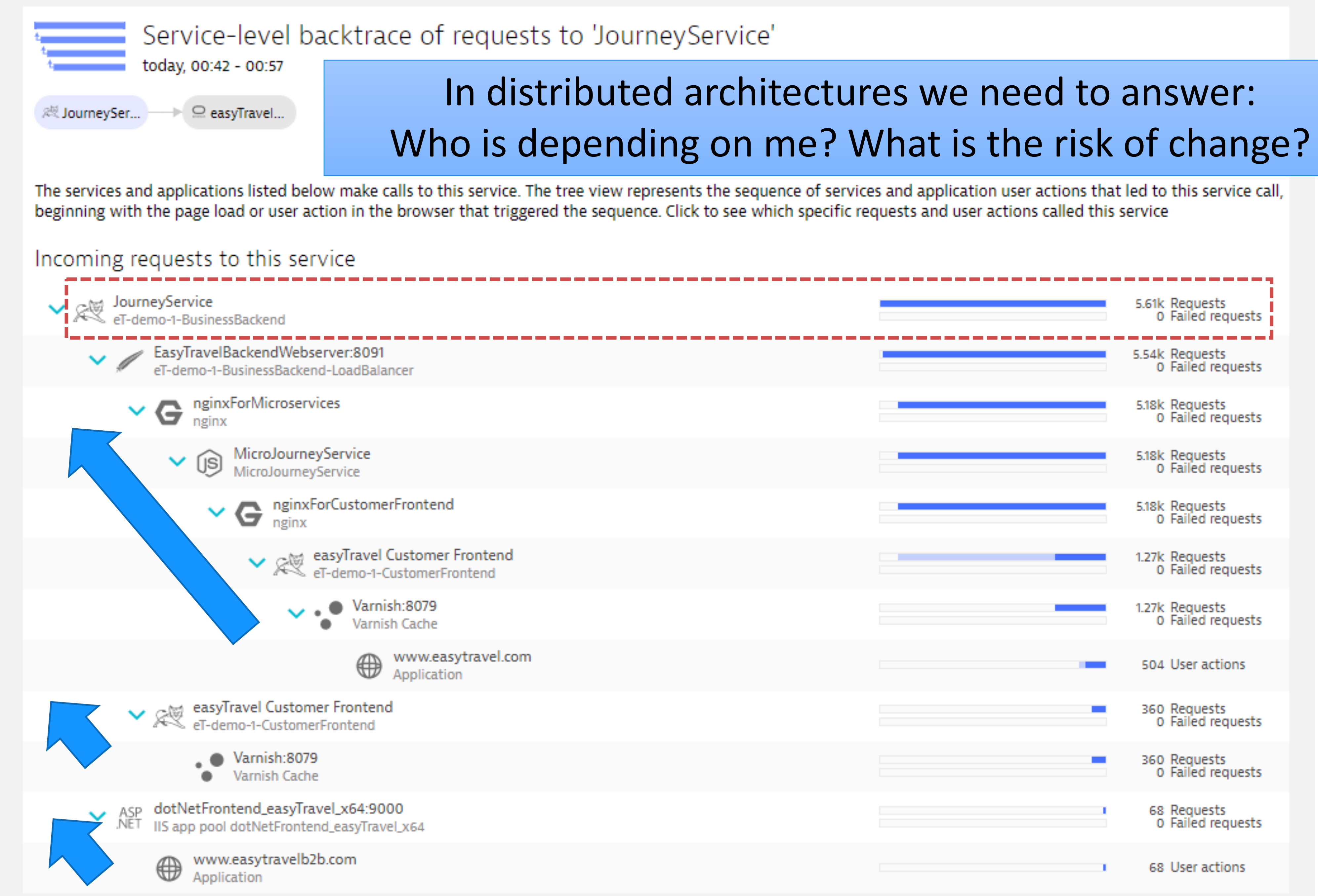
Response time degradation
The current response time (154 ms) exceeds the auto-detected baseline (6.8 ms) by 2,160 %

Affected requests	Service method
407 /min	All dynamic requests

Visual resolution path
Click to see how we figured this out.

https://www.dynatrace.com/news/blog/enterprise-cloud-ecs-microservices-and-dynatrace-at-neiman-marcus/

In distributed architectures we need to answer: Who is depending on me? What is the risk of change?



Service-level backtrace of requests to 'JourneyService'

today, 00:42 - 00:57

The services and applications listed below make calls to this service. The tree view represents the sequence of services and application user actions that led to this service call, beginning with the page load or user action in the browser that triggered the sequence. Click to see which specific requests and user actions called this service

Incoming requests to this service

- JourneyService – eT-demo-1-BusinessBackend – 5.61k Requests, 0 Failed requests
  - EasyTravelBackendWebserver:8091 – eT-demo-1-BusinessBackend-LoadBalancer – 5.54k Requests, 0 Failed requests
    - nginxForMicroservices – nginx – 5.18k Requests, 0 Failed requests
      - MicroJourneyService – MicroJourneyService – 5.18k Requests, 0 Failed requests
        - nginxForCustomerFrontend – nginx – 5.18k Requests, 0 Failed requests
          - easyTravel Customer Frontend – eT-demo-1-CustomerFrontend – 1.27k Requests, 0 Failed requests
            - Varnish:8079 – Varnish Cache – 1.27k Requests, 0 Failed requests
              - www.easytravel.com – Application – 504 User actions
    - easyTravel Customer Frontend – eT-demo-1-CustomerFrontend – 360 Requests, 0 Failed requests
      - Varnish:8079 – Varnish Cache – 360 Requests, 0 Failed requests
  - dotNetFrontend_easyTravel_x64:9000 – IIS app pool dotNetFrontend_easyTravel_x64 – 68 Requests, 0 Failed requests
    - www.easytravelb2b.com – Application – 68 User actions

Common Distributed Architectural Patterns

Patterns I‘ve seen in > 90% of the problems I analyzed

There are more – and we only have time to cover some today

1. N+1 call
2. N+1 query
3. Payload flood
4. Granularity
5. Tight Coupling
6. Inefficient Service Flow
7. Timeouts, Retries, Backoff
8. Dependencies

More recorded presentations on problem patterns:

- Java and Performance: Biggest Mistake - https://www.youtube.com/watch?v=IBkxiWmjM-g (SFO Java Meetup)
- Top Performance Challenges: https://www.youtube.com/watch?v=QypHTQr2RXk (Confitura 2019)

N + 1 Call Pattern

Or better: 1 + N

1 initial call + 1 Call per N results

N+1 Call Pattern

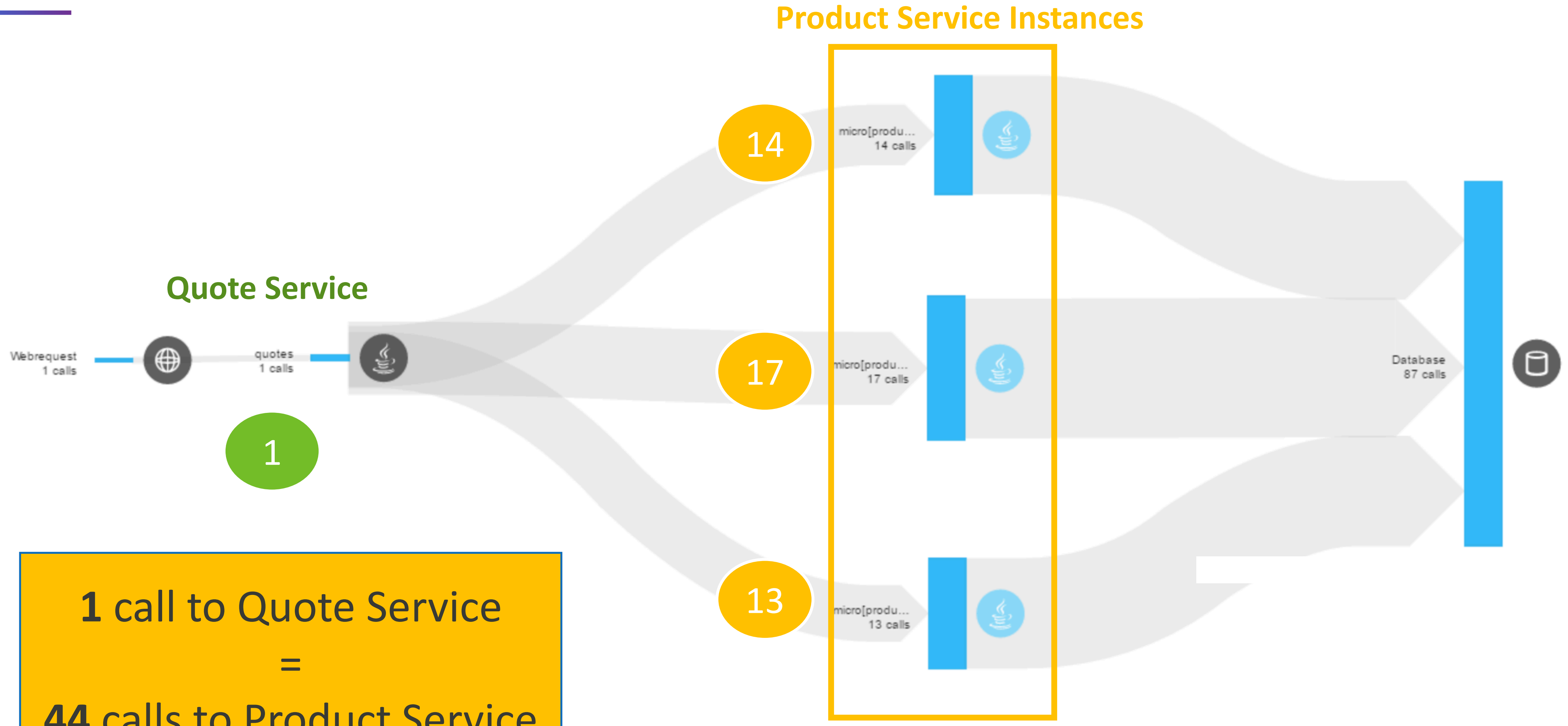
Monolithic Code

```
public double getTotalQuote(Products[] products) {
    double quote=0;
    for (Product product: products) {
        quote += product.getQuote();
    }
    return quote;
}
```

“Works” well within a single process

Extract into Service?

N+1 Call Pattern across distributed “Product Service”



Product Service Instances

Quote Service

1

14

17

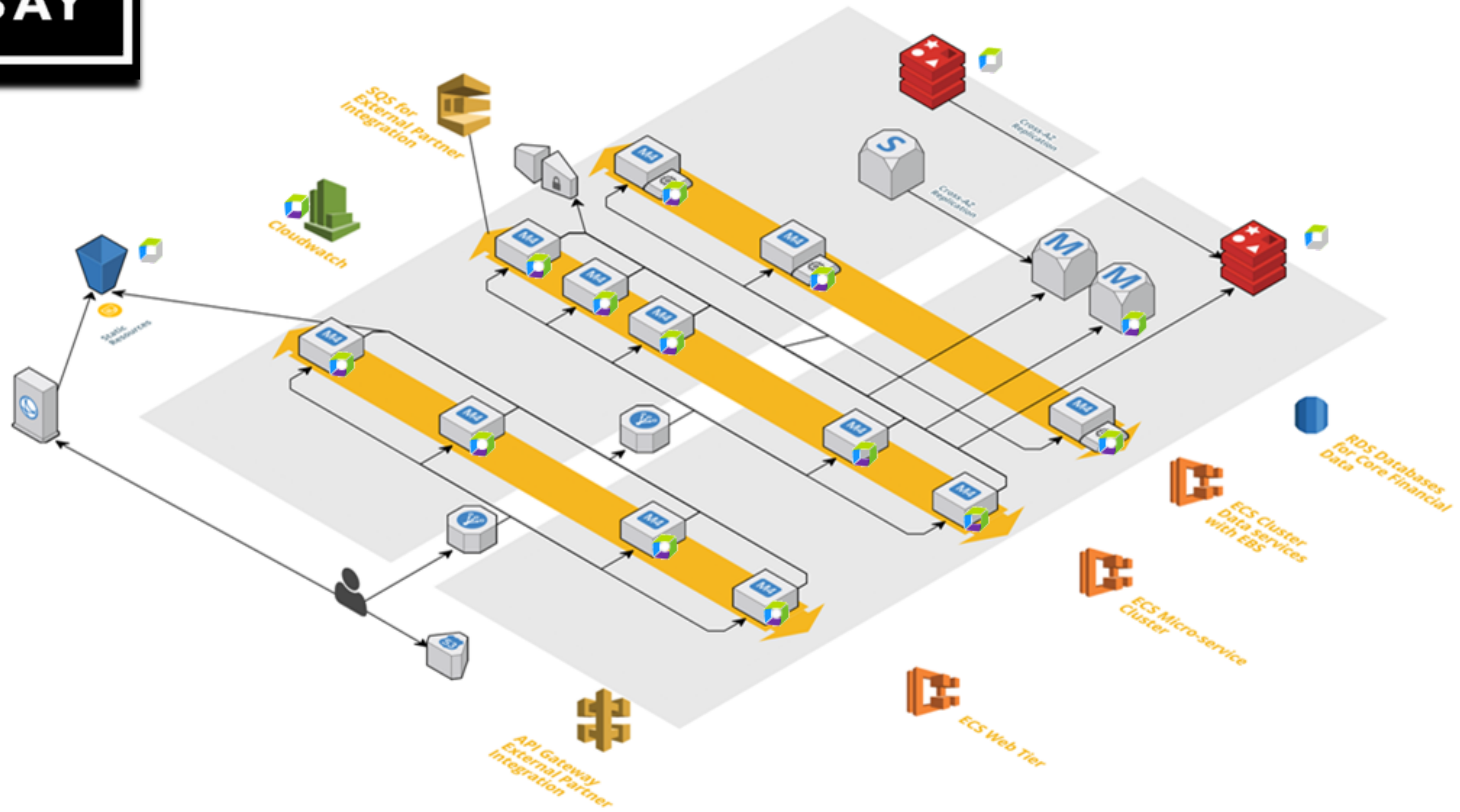
13

1 call to Quote Service

=

44 calls to Product Service

LANDBAY


SQS for External Partner Integration
Cloudwatch
Static Resources
Cross-AZ Replication
Cross-AZ Replication
M4
RDS Databases for Core Financial Data
ECS Cluster Data services with EBS
ECS Micro-service Cluster
ECS Web Tier
API Gateway External Partner Integration

https://aws.amazon.com/solutions/case-studies/landbay/

LANDBAY

24

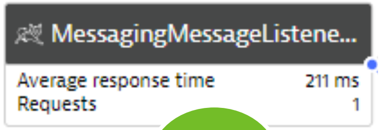
24

24

24

Subtotal: 243

74



MessagingMessageListene...

Average response time 211 ms

Requests 1

1

22

22

24

1

1

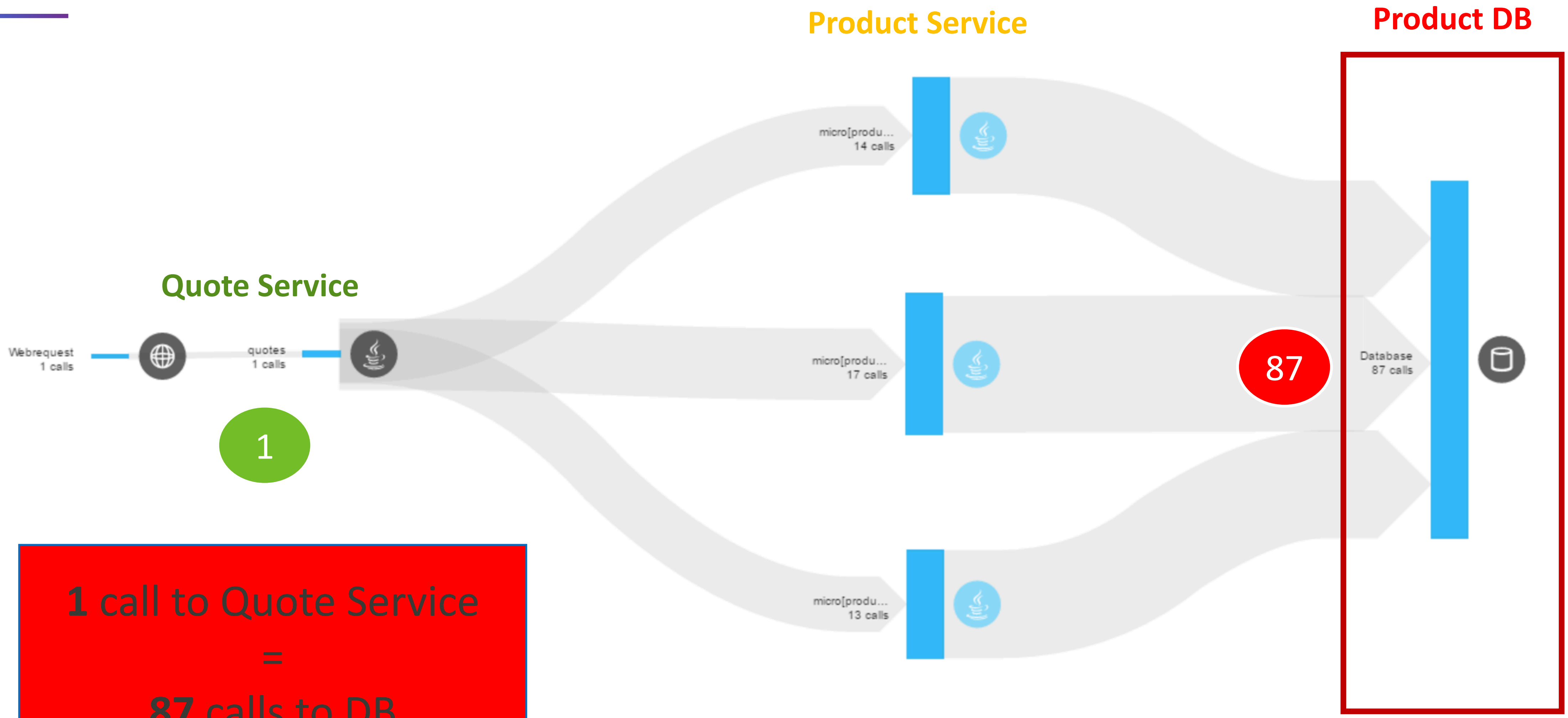
1

1

N + 1 Query Pattern

Similiar to N +1 Call Pattern but focused on database queries

N+1 Query Pattern

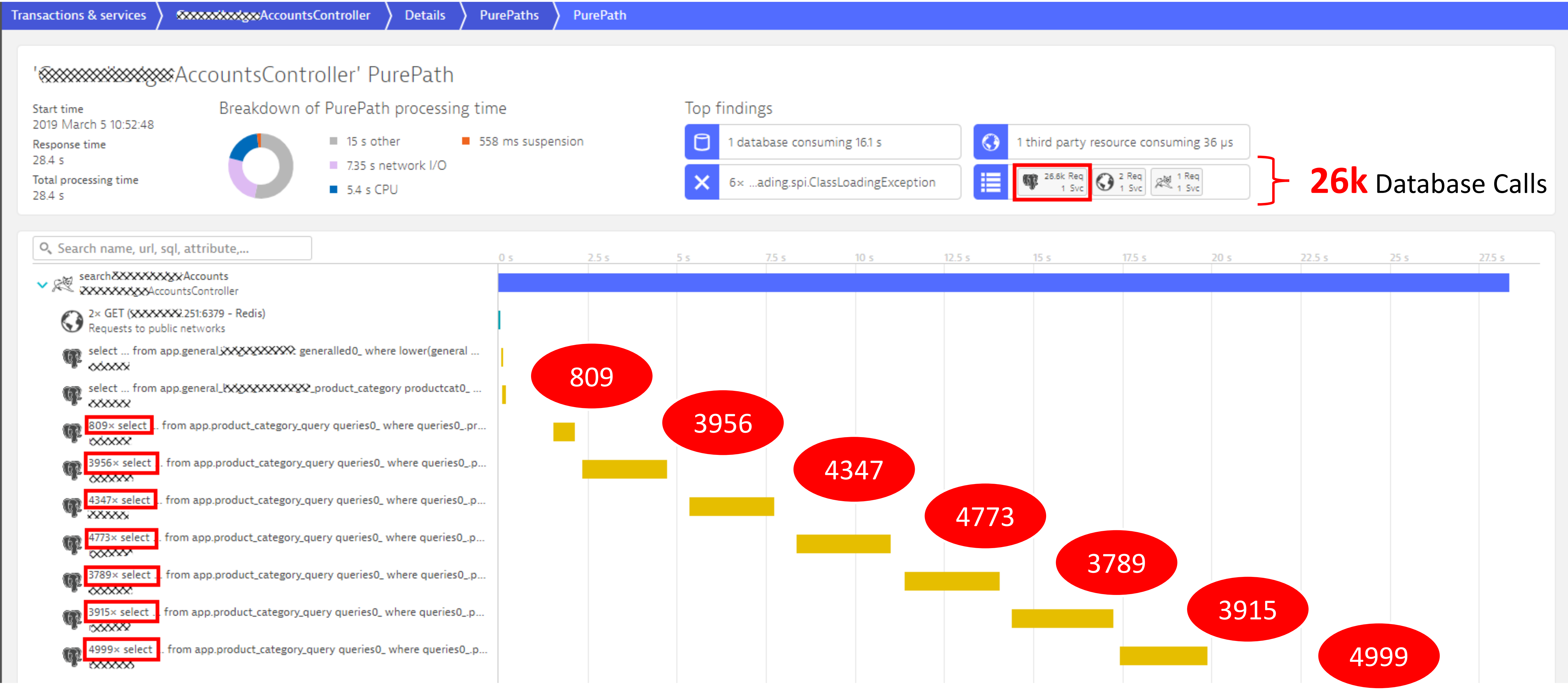


1 call to Quote Service

=

87 calls to DB

Cascading N+1 Query Pattern: This is a single End-2-End Distributed Trace



Transactions & services 〉 [redacted]AccountsController 〉 Details 〉 PurePaths 〉 PurePath

'[redacted]AccountsController' PurePath

Start time
2019 March 5 10:52:48

Response time
28.4 s

Total processing time
28.4 s

Breakdown of PurePath processing time

- 15 s other
- 558 ms suspension
- 7.35 s network I/O
- 5.4 s CPU

Top findings

- 1 database consuming 16.1 s
- 1 third party resource consuming 36 µs
- 6× ...ading.spi.ClassLoadingException
- 26.6k Req 1 Svc
- 2 Req 1 Svc
- 1 Req 1 Svc

26k Database Calls

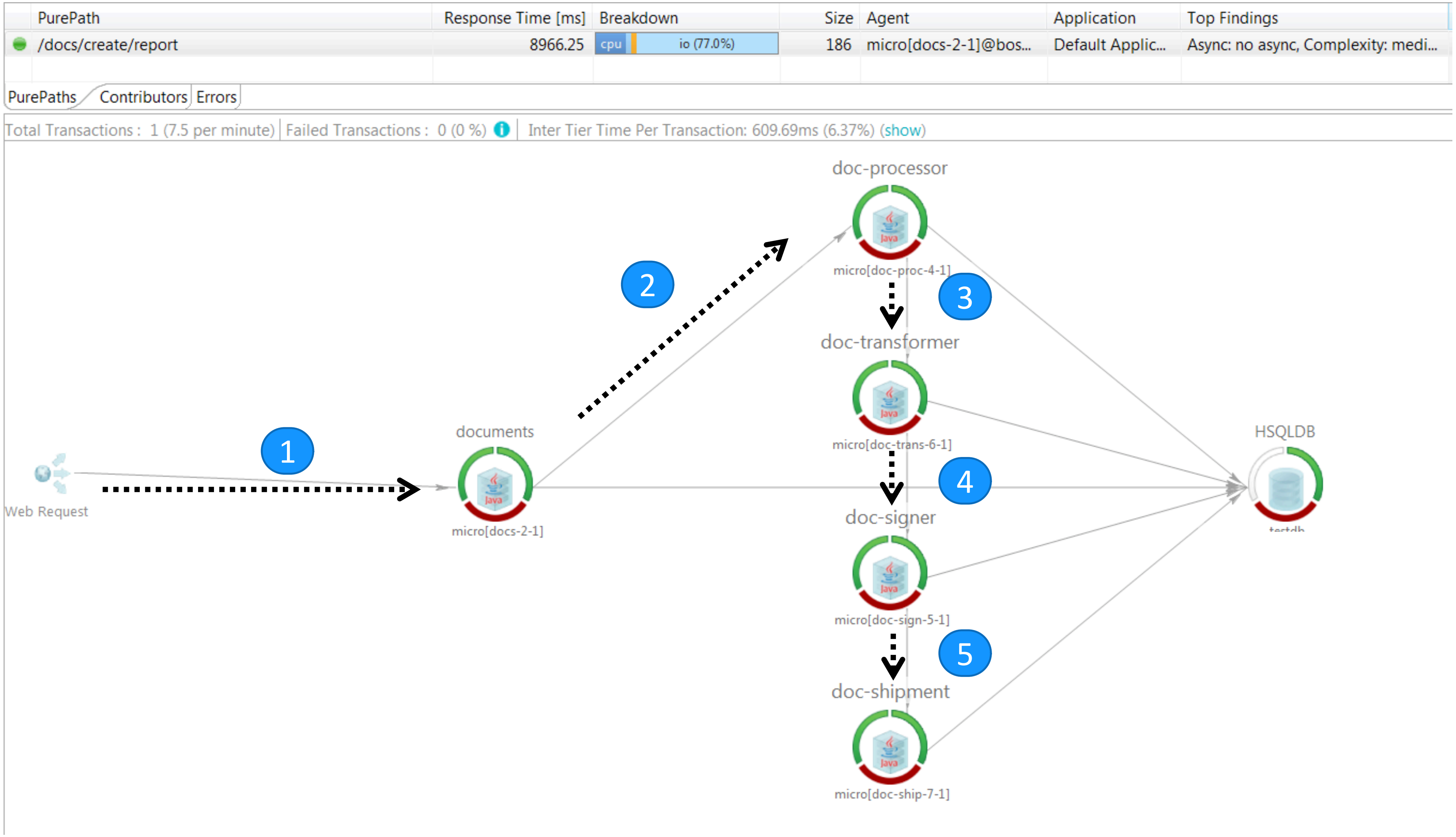
Search name, url, sql, attribute,...

- search[redacted]Accounts
  [redacted]AccountsController
- 2× GET ([redacted].251:6379 - Redis)
  Requests to public networks
- select ... from app.general_[redacted] generalled0_ where lower(general ...
  [redacted]
- select ... from app.general_[redacted]_product_category productcat0_ ...
  [redacted]
- 809× select ... from app.product_category_query queries0_ where queries0_.pr...
  [redacted]
- 3956× select ... from app.product_category_query queries0_ where queries0_.p...
  [redacted]
- 4347× select ... from app.product_category_query queries0_ where queries0_.p...
  [redacted]
- 4773× select ... from app.product_category_query queries0_ where queries0_.p...
  [redacted]
- 3789× select ... from app.product_category_query queries0_ where queries0_.p...
  [redacted]
- 3915× select ... from app.product_category_query queries0_ where queries0_.p...
  [redacted]
- 4999× select ... from app.product_category_query queries0_ where queries0_.p...
  [redacted]

Payload Flood

AKA – sending useless information across the network

Payload Flood: “Doc Creation” sequential across distributed services

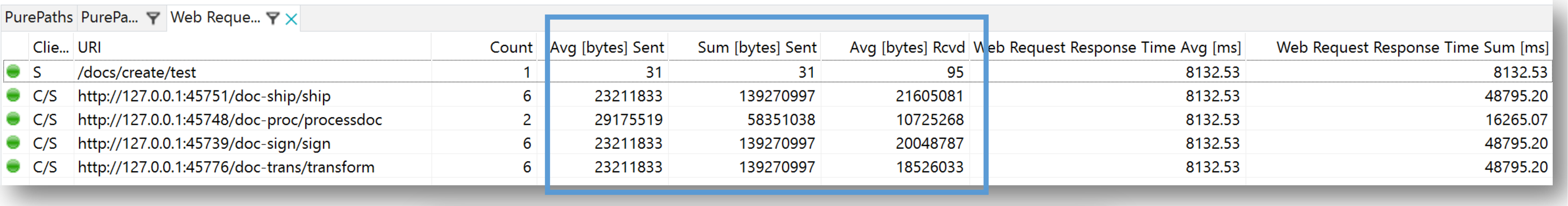


PurePath	Response Time [ms]	Breakdown	Size	Agent	Application	Top Findings
● /docs/create/report	8966.25	cpu io (77.0%)	186	micro[docs-2-1]@bos...	Default Applic...	Async: no async, Complexity: medi...

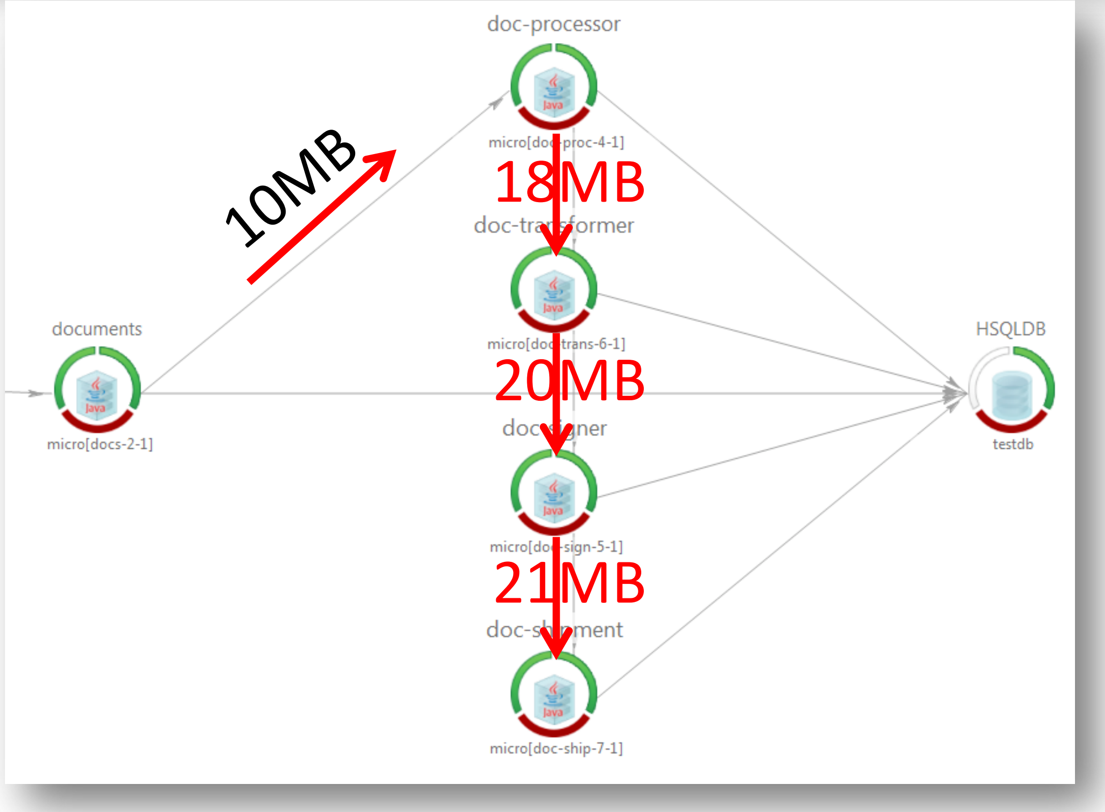
PurePaths | Contributors | Errors

Total Transactions : 1 (7.5 per minute) | Failed Transactions : 0 (0 %) | Inter Tier Time Per Transaction: 609.69ms (6.37%) (show)

Payload Flood in numbers: Full DOC sent between distributed services



	Clie...	URI	Count	Avg [bytes] Sent	Sum [bytes] Sent	Avg [bytes] Rcvd	Web Request Response Time Avg [ms]	Web Request Response Time Sum [ms]
●	S	/docs/create/test	1	31	31	95	8132.53	8132.53
●	C/S	http://127.0.0.1:45751/doc-ship/ship	6	23211833	139270997	21605081	8132.53	48795.20
●	C/S	http://127.0.0.1:45748/doc-proc/processdoc	2	29175519	58351038	10725268	8132.53	16265.07
●	C/S	http://127.0.0.1:45739/doc-sign/sign	6	23211833	139270997	20048787	8132.53	48795.20
●	C/S	http://127.0.0.1:45776/doc-trans/transform	6	23211833	139270997	18526033	8132.53	48795.20

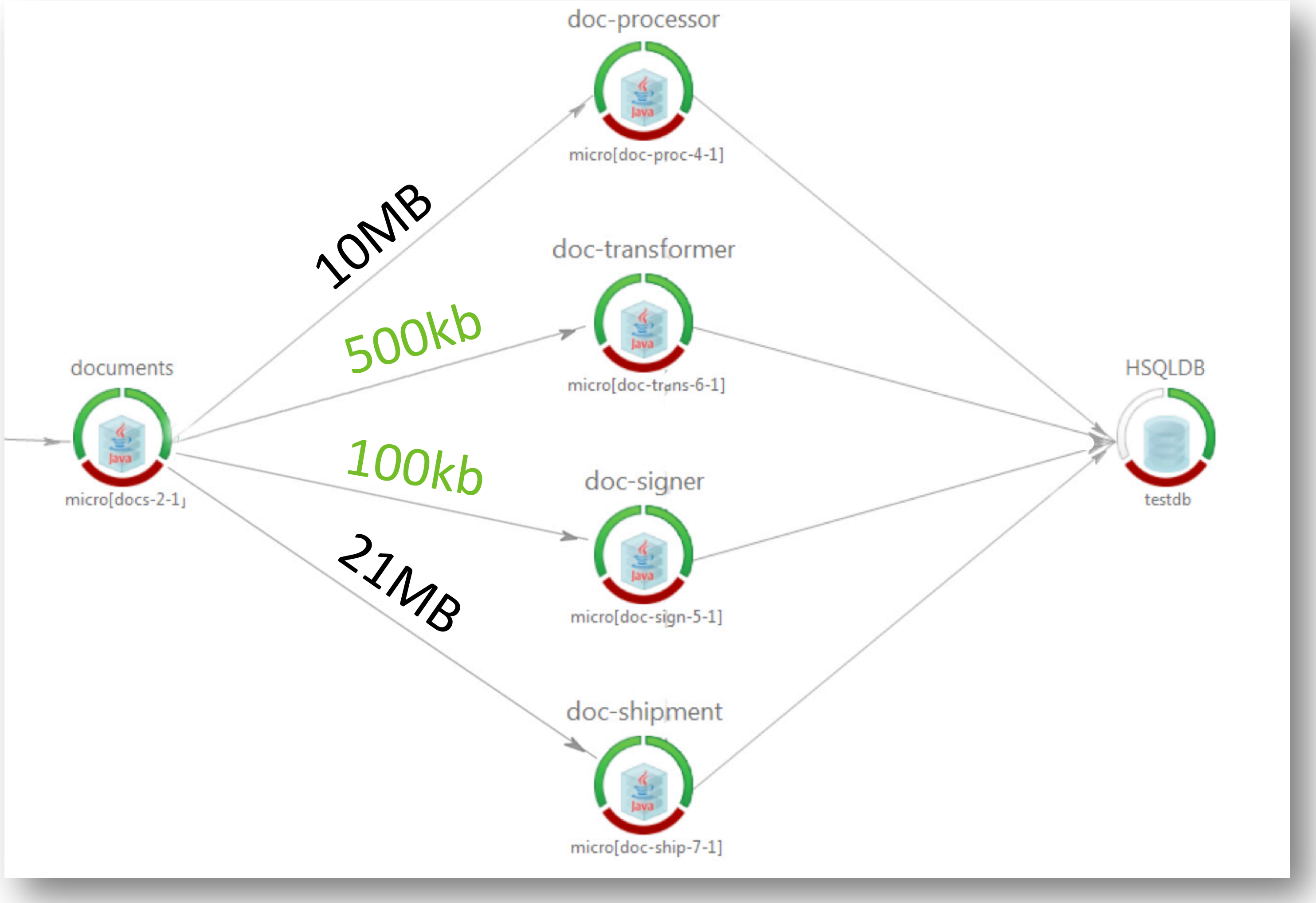


Refactor: Only send relevant data to specialized services

69MB

vs

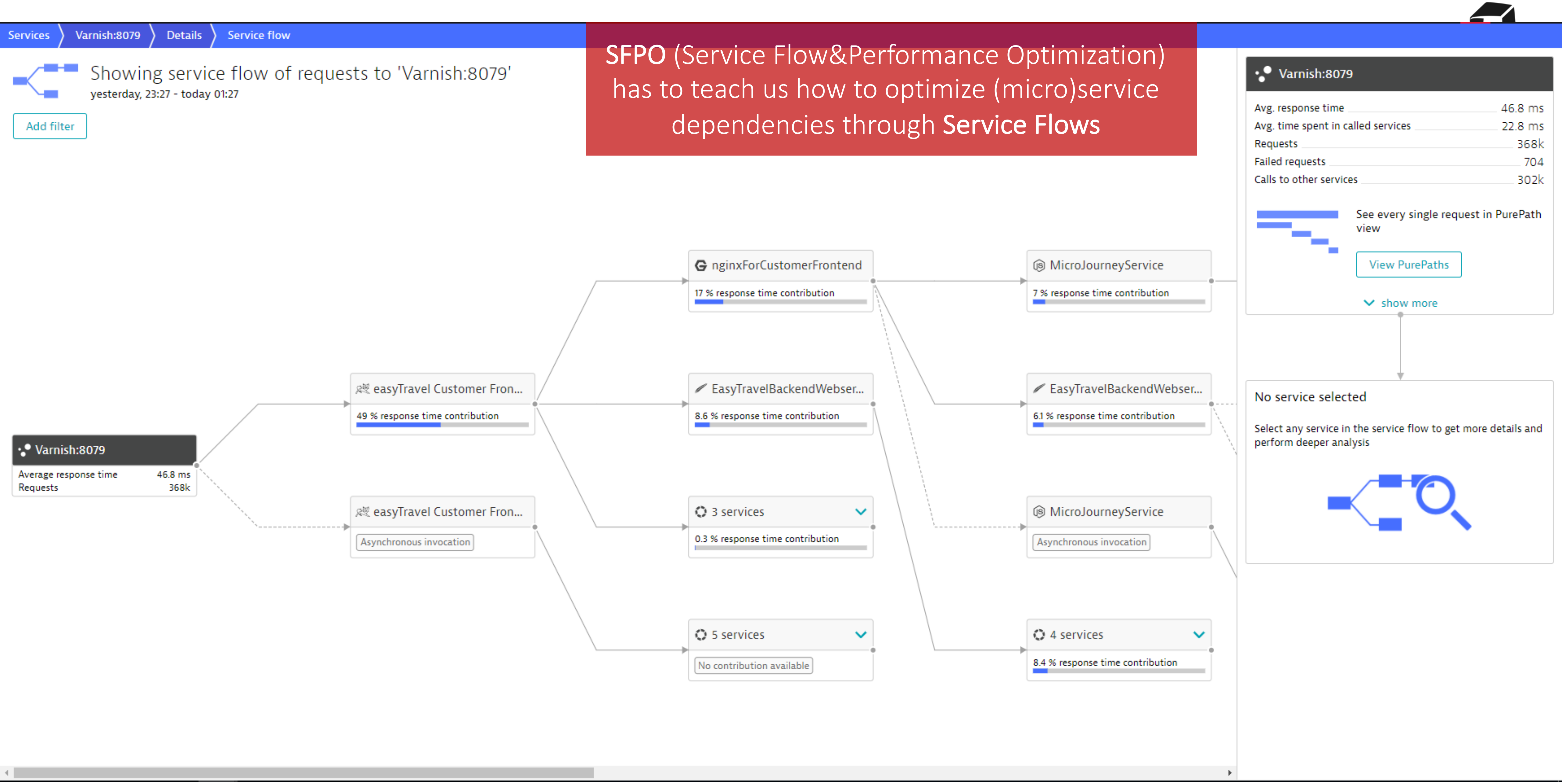
31.6MB



Inefficient Service Flow

drawing parallels to Web Performance Optimization

SFPO (Service Flow&Performance Optimization) has to teach us how to optimize (micro)service dependencies through **Service Flows**



Services › Varnish:8079 › Details › Service flow

Showing service flow of requests to 'Varnish:8079'
yesterday, 23:27 - today 01:27

Add filter

Varnish:8079
Average response time 46.8 ms
Requests 368k
easyTravel Customer Fron...
49 % response time contribution
easyTravel Customer Fron...
Asynchronous invocation
nginxForCustomerFrontend
17 % response time contribution
EasyTravelBackendWebser...
8.6 % response time contribution
3 services
0.3 % response time contribution
5 services
No contribution available
MicroJourneyService
7 % response time contribution
EasyTravelBackendWebser...
6.1 % response time contribution
MicroJourneyService
Asynchronous invocation
4 services
8.4 % response time contribution

Varnish:8079

Avg. response time	46.8 ms
Avg. time spent in called services	22.8 ms
Requests	368k
Failed requests	704
Calls to other services	302k

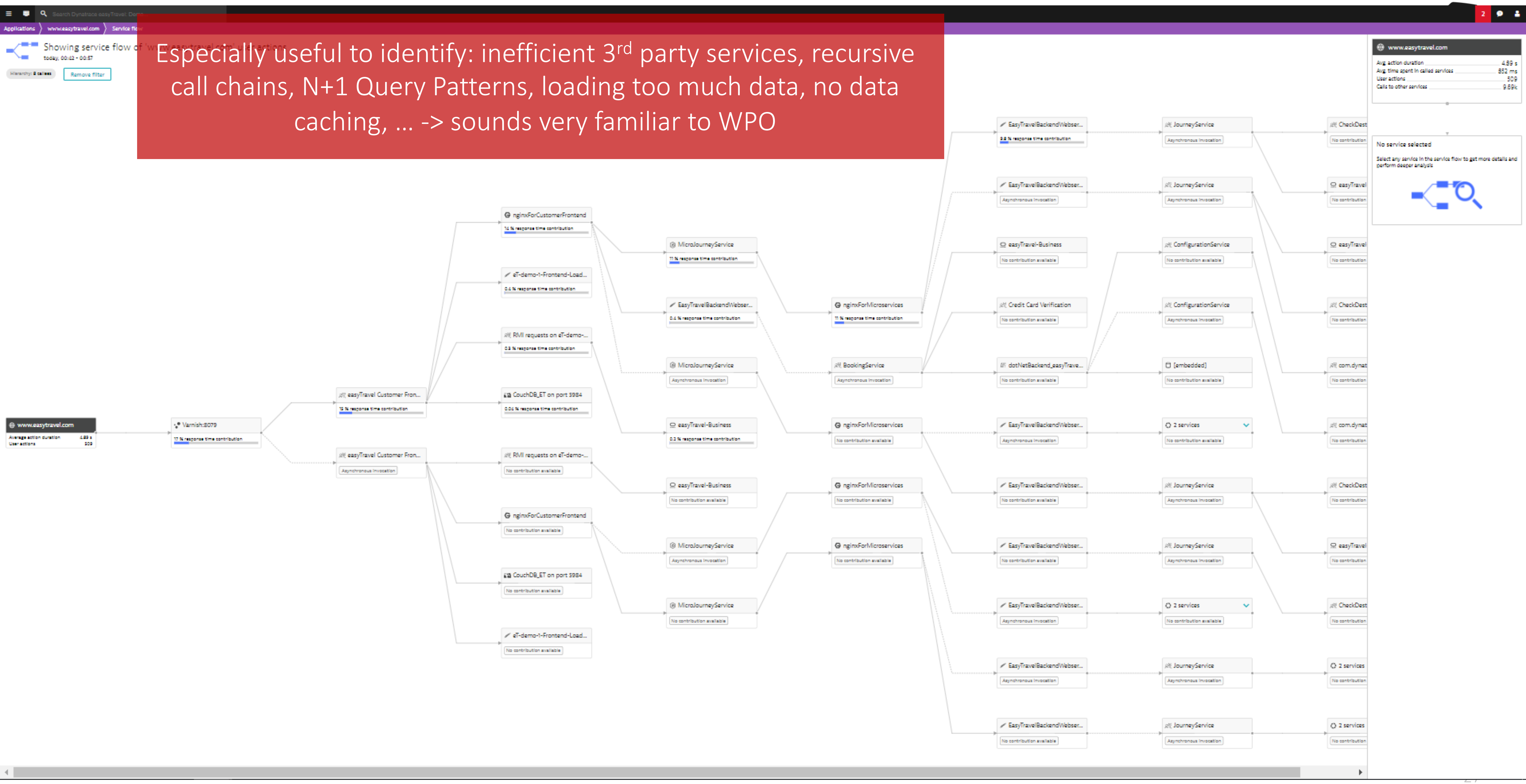
See every single request in PurePath view

View PurePaths

show more

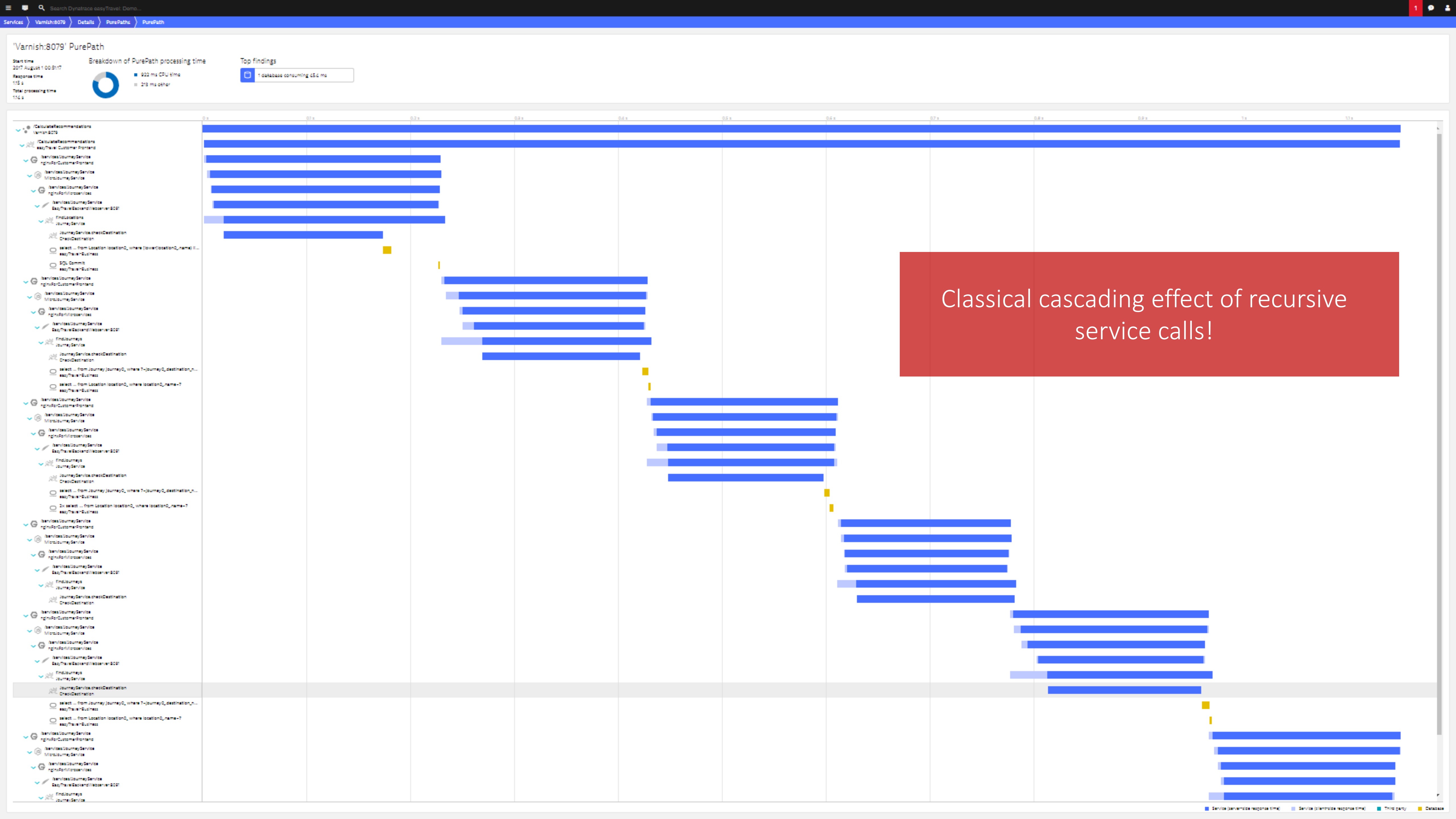
No service selected

Select any service in the service flow to get more details and perform deeper analysis



Applications
www.easytravel.com
Service flow
Showing service flow of 'www.easytravel.com' user actions
today, 00:42 - 00:57
Hierarchy: 8 callees
Remove filter
www.easytravel.com
Average action duration 4.89 s
User actions 509
Varnish:8079
17 % response time contribution
easyTravel Customer Fron...
15 % response time contribution
Asynchronous invocation
nginxForCustomerFrontend
14 % response time contribution
eT-demo-1-Frontend-Load...
0.4 % response time contribution
RMI requests on eT-demo-...
0.3 % response time contribution
CouchDB_ET on port 5984
0.04 % response time contribution
MicroJourneyService
11 % response time contribution
EasyTravelBackendWebser...
easyTravel-Business
nginxForMicroservices
BookingService
Credit Card Verification
dotNetBackend_easyTrave...
JourneyService
ConfigurationService
[embedded]
2 services
CheckDest
com.dynat
No contribution available
www.easytravel.com
Avg. action duration 4.89 s
Avg. time spent in called services 852 ms
User actions 509
Calls to other services 9.69k
No service selected
Select any service in the service flow to get more details and perform deeper analysis

Especially useful to identify: inefficient 3rd party services, recursive call chains, N+1 Query Patterns, loading too much data, no data caching, … -> sounds very familiar to WPO



'Varnish:8079' PurePath

Start time
2017 August 1 00:51:17

Response time
1.15 s

Total processing time
1.14 s

Breakdown of PurePath processing time

- 922 ms CPU time
- 218 ms other

Top findings

1 database consuming 45.4 ms

/CalculateRecommendations
Varnish:8079

/CalculateRecommendations
easyTravel Customer Frontend

/services/JourneyService
nginxForCustomerFrontend

/services/JourneyService
MicroJourneyService

/services/JourneyService
nginxForMicroservices

/services/JourneyService
EasyTravelBackendWebserver:8091

findLocations
JourneyService

JourneyService.checkDestination
CheckDestination

select ... from Location location0_ where (lower(location0_.name) l...
easyTravel-Business

SQL Commit
easyTravel-Business

/services/JourneyService
nginxForCustomerFrontend

/services/JourneyService
MicroJourneyService

/services/JourneyService
nginxForMicroservices

/services/JourneyService
EasyTravelBackendWebserver:8091

findJourneys
JourneyService

JourneyService.checkDestination
CheckDestination

select ... from Journey journey0_ where ?=journey0_.destination_n...
easyTravel-Business

select ... from Location location0_ where location0_.name=?
easyTravel-Business

/services/JourneyService
nginxForCustomerFrontend

/services/JourneyService
MicroJourneyService

/services/JourneyService
nginxForMicroservices

/services/JourneyService
EasyTravelBackendWebserver:8091

findJourneys
JourneyService

JourneyService.checkDestination
CheckDestination

select ... from Journey journey0_ where ?=journey0_.destination_n...
easyTravel-Business

2x select ... from Location location0_ where location0_.name=?
easyTravel-Business

/services/JourneyService
nginxForCustomerFrontend

/services/JourneyService
MicroJourneyService

/services/JourneyService
nginxForMicroservices

/services/JourneyService
EasyTravelBackendWebserver:8091

findJourneys
JourneyService

JourneyService.checkDestination
CheckDestination

/services/JourneyService
nginxForCustomerFrontend

/services/JourneyService
MicroJourneyService

/services/JourneyService
nginxForMicroservices

/services/JourneyService
EasyTravelBackendWebserver:8091

findJourneys
JourneyService

JourneyService.checkDestination
CheckDestination

select ... from Journey journey0_ where ?=journey0_.destination_n...
easyTravel-Business

select ... from Location location0_ where location0_.name=?
easyTravel-Business

/services/JourneyService
nginxForCustomerFrontend

/services/JourneyService
MicroJourneyService

/services/JourneyService
nginxForMicroservices

/services/JourneyService
EasyTravelBackendWebserver:8091

findJourneys
JourneyService

Service (serverside response time)
Service (clientside response time)
Third party
Database

Classical cascading effect of recursive service calls!

Common Distributed Architectural Patterns

Recap and overview of Metrics used for pattern detection!

Recap - Common Distributed Patterns + Metrics to look at

1. N+1 call: ***# same Service Invocations per Request***
2. N+1 query: ***# same SQL Invocations per Request***
3. Payload flood: ***Transfer Size!***
4. Granularity: ***# of Service Invocations across End-2-End Transaction***
5. Tight Coupling: ***Ratio between Service Invocations***
6. Inefficient Service Flow: ***# of Involved Services, # of Calls to each Service***
7. Timeouts, Retries, Backoff: ***Pool Utilization, …***
8. Dependencies: ***# of Incoming & Outcoming Dependencies***

More recorded presentations on problem patterns:

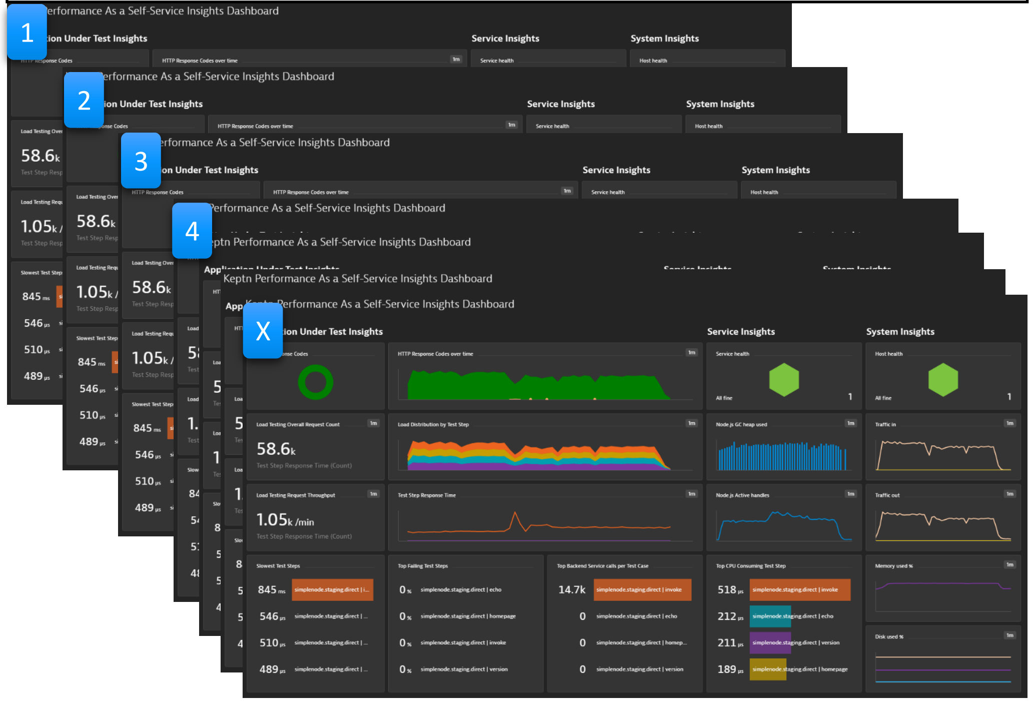
- Java and Performance: Biggest Mistake - https://www.youtube.com/watch?v=IBkxiWmjM-g (SFO Java Meetup)
- Top Performance Challenges: https://www.youtube.com/watch?v=QypHTQr2RXk (Confitura 2019)

Can we automate pattern detection?

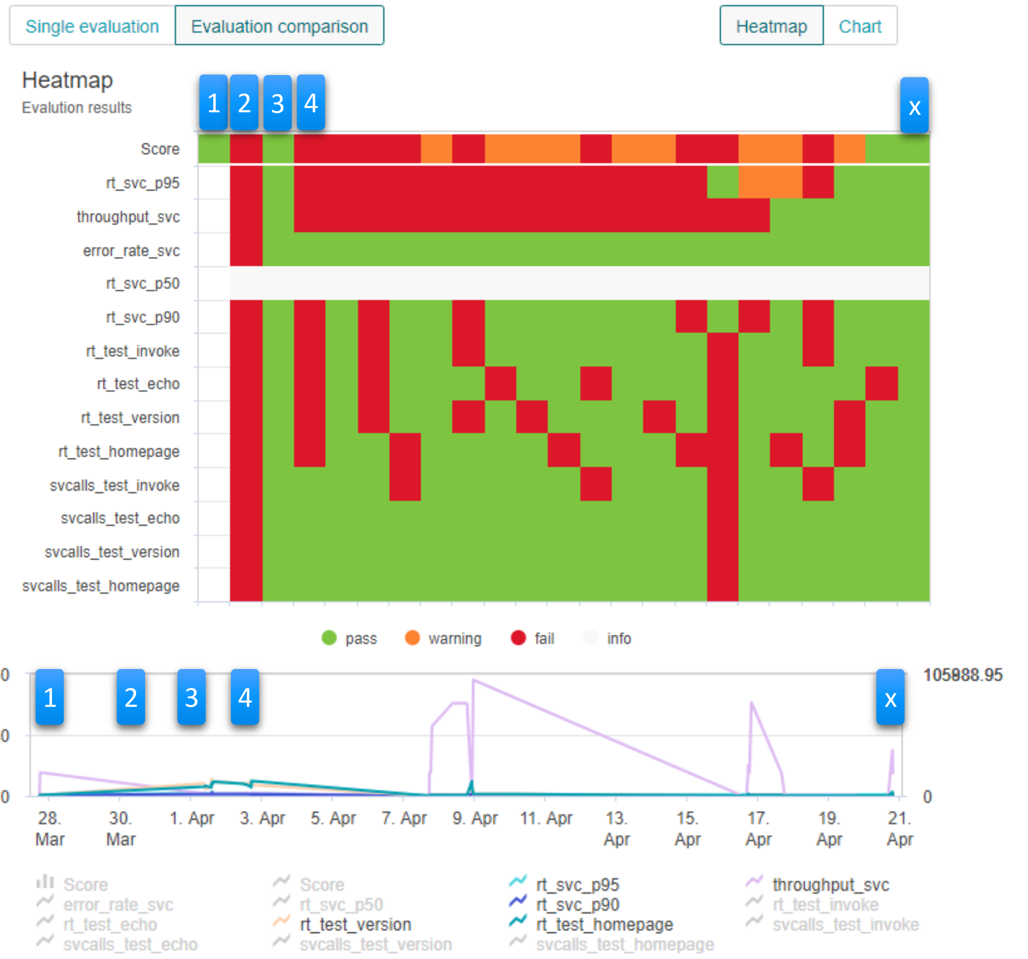
If we can detect them on a dashboard – we should be able to automate!

Keptn automates analysis through SLIs/SLOs

Instead of manually detecting patterns and comparing metrics



keptn automates that process based on SLIs & SLOs



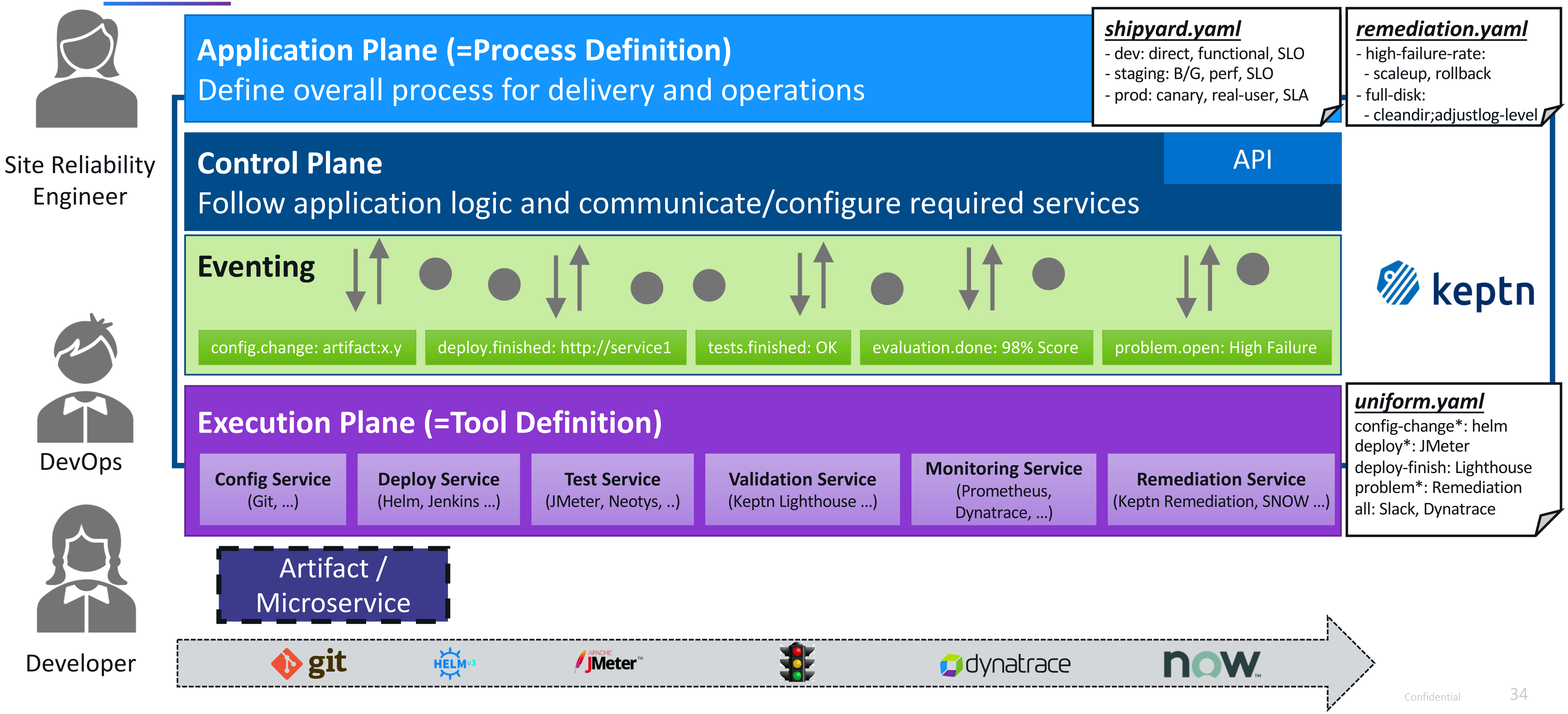
Integrate in Testing, Delivery & Auto-Remediation

Introducing Keptn

Declarative, extensible automation of SLO-driven delivery, quality gates & remediation

https://github.com/keptn, www.keptn.sh

Keptn from 10000ft: Declarative Workflows + Event-Triggered Actions



Site Reliability Engineer

DevOps

Developer

Application Plane (=Process Definition)
Define overall process for delivery and operations

shipyard.yaml
- dev: direct, functional, SLO
- staging: B/G, perf, SLO
- prod: canary, real-user, SLA

remediation.yaml
- high-failure-rate:
  - scaleup, rollback
- full-disk:
  - cleandir;adjustlog-level

Control Plane
Follow application logic and communicate/configure required services

API

Eventing

config.change: artifact:x.y

deploy.finished: http://service1

tests.finished: OK

evaluation.done: 98% Score

problem.open: High Failure

Execution Plane (=Tool Definition)

Config Service (Git, ...)

Deploy Service (Helm, Jenkins ...)

Test Service (JMeter, Neotys, ..)

Validation Service (Keptn Lighthouse ...)

Monitoring Service (Prometheus, Dynatrace, ...)

Remediation Service (Keptn Remediation, SNOW ...)

uniform.yaml
config-change*: helm
deploy*: JMeter
deploy-finish: Lighthouse
problem*: Remediation
all: Slack, Dynatrace

Artifact / Microservice

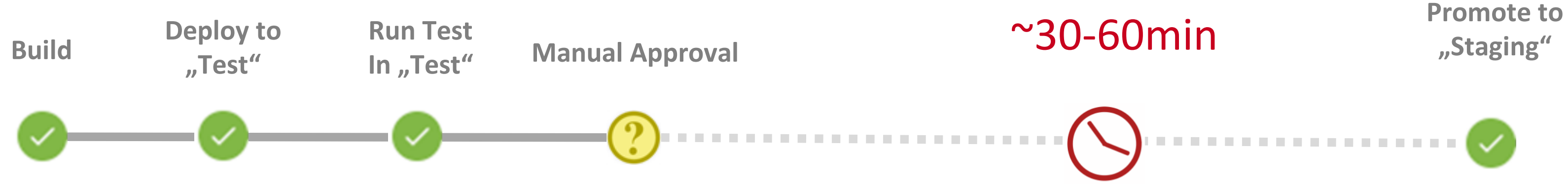
Use Case #1

Automated Architecture & Performance Validation

Through event-based SLI/SLO-based Quality Gates

https://github.com/keptn, www.keptn.sh

Root Cause: Lengthy manual approval in existing delivery pipelines



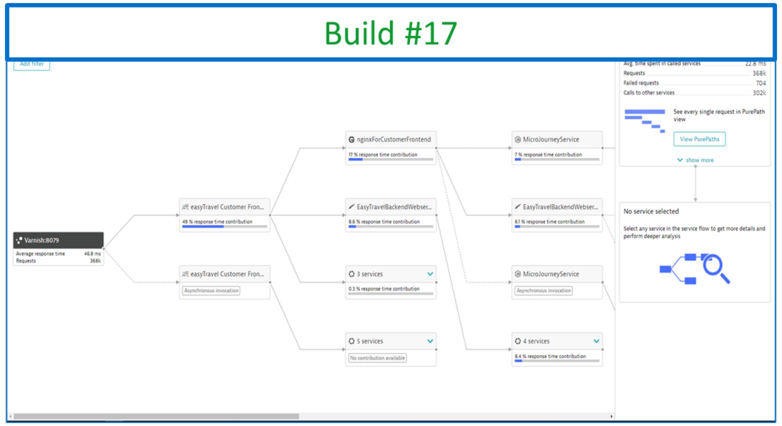
Looking at all these dashboards and data points is time-consuming and slows down the process!

Identify / Optimize Architectural Patterns

Recursive Calls, N+1 Call Pattern, Chatty Interfaces, No Caching Layer …

Identify Performance Hotspots

CPU, Memory, I/O, …


Build #17

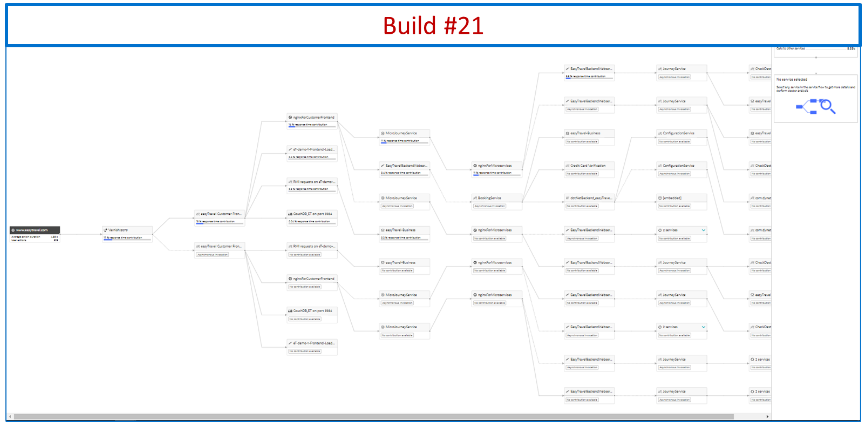
Avg. time spent in called services	22.8 ms
Requests	368k
Failed requests	704
Calls to other services	302k

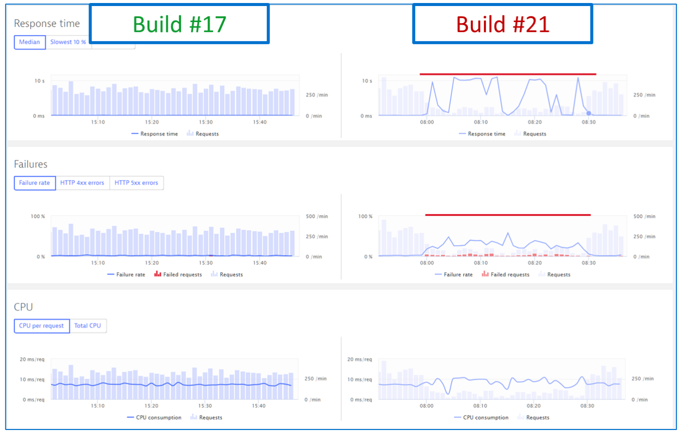
See every single request in PurePath view

No service selected

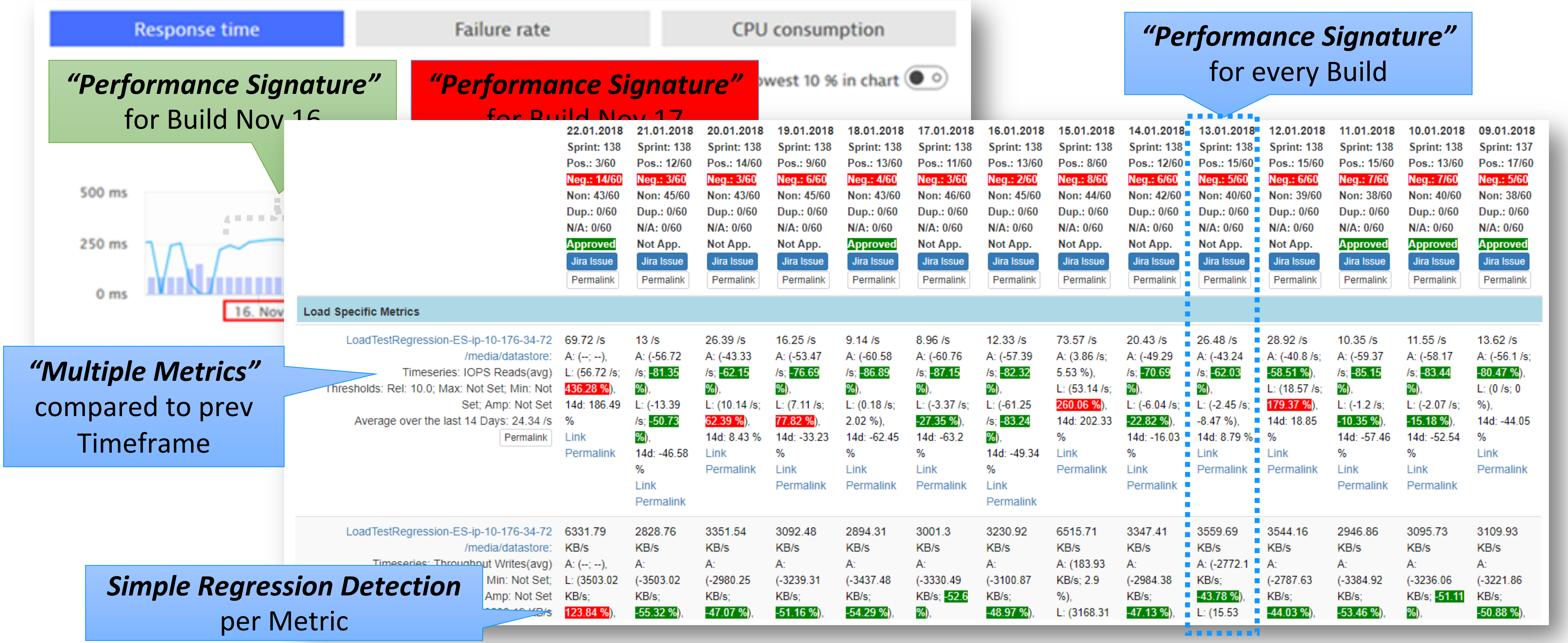
Select any service in the service flow to get more details and perform deeper analysis

Add filter
View PurePaths
show more
Varnish:8079
Average response time 46.8 ms
Requests 368k
easyTravel Customer Fron...
49 % response time contribution
easyTravel Customer Fron...
Asynchronous invocation
nginxForCustomerFrontend
17 % response time contribution
EasyTravelBackendWebser...
8.6 % response time contribution
3 services
0.3 % response time contribution
5 services
No contribution available
MicroJourneyService
7 % response time contribution
EasyTravelBackendWebser...
6.1 % response time contribution
MicroJourneyService
Asynchronous invocation
4 services
8.4 % response time contribution


Build #21



Inspired by Dynatrace‘s internal „Performance Signature as Code“



“Performance Signature”
for Build Nov 16

“Performance Signature”
for Build Nov 17

“Performance Signature”
for every Build

“Multiple Metrics”
compared to prev Timeframe

Simple Regression Detection
per Metric

	22.01.2018	21.01.2018	20.01.2018	19.01.2018	18.01.2018	17.01.2018	16.01.2018	15.01.2018	14.01.2018	13.01.2018	12.01.2018	11.01.2018	10.01.2018	09.01.2018
	Sprint: 138	Sprint: 138	Sprint: 138	Sprint: 138	Sprint: 138	Sprint: 138	Sprint: 138	Sprint: 138	Sprint: 138	Sprint: 138	Sprint: 138	Sprint: 138	Sprint: 138	Sprint: 137
	Pos.: 3/60	Pos.: 12/60	Pos.: 14/60	Pos.: 9/60	Pos.: 13/60	Pos.: 11/60	Pos.: 13/60	Pos.: 8/60	Pos.: 12/60	Pos.: 15/60	Pos.: 15/60	Pos.: 15/60	Pos.: 13/60	Pos.: 17/60
	Neg.: 14/60	Neg.: 3/60	Neg.: 3/60	Neg.: 6/60	Neg.: 4/60	Neg.: 3/60	Neg.: 2/60	Neg.: 8/60	Neg.: 6/60	Neg.: 5/60	Neg.: 6/60	Neg.: 7/60	Neg.: 7/60	Neg.: 5/60
	Non: 43/60	Non: 45/60	Non: 43/60	Non: 45/60	Non: 43/60	Non: 46/60	Non: 45/60	Non: 44/60	Non: 42/60	Non: 40/60	Non: 39/60	Non: 38/60	Non: 40/60	Non: 38/60
	Dup.: 0/60	Dup.: 0/60	Dup.: 0/60	Dup.: 0/60	Dup.: 0/60	Dup.: 0/60	Dup.: 0/60	Dup.: 0/60	Dup.: 0/60	Dup.: 0/60	Dup.: 0/60	Dup.: 0/60	Dup.: 0/60	Dup.: 0/60
	N/A: 0/60	N/A: 0/60	N/A: 0/60	N/A: 0/60	N/A: 0/60	N/A: 0/60	N/A: 0/60	N/A: 0/60	N/A: 0/60	N/A: 0/60	N/A: 0/60	N/A: 0/60	N/A: 0/60	N/A: 0/60
	Approved	Not App.	Not App.	Not App.	Approved	Not App.	Not App.	Not App.	Not App.	Not App.	Not App.	Approved	Approved	Approved
	Jira Issue	Jira Issue	Jira Issue	Jira Issue	Jira Issue	Jira Issue	Jira Issue	Jira Issue	Jira Issue	Jira Issue	Jira Issue	Jira Issue	Jira Issue	Jira Issue
	Permalink	Permalink	Permalink	Permalink	Permalink	Permalink	Permalink	Permalink	Permalink	Permalink	Permalink	Permalink	Permalink	Permalink
Load Specific Metrics														
LoadTestRegression-ES-ip-10-176-34-72 /media/datastore: Timeseries: IOPS Reads(avg) Thresholds: Rel: 10.0; Max: Not Set; Min: Not Set; Amp: Not Set Average over the last 14 Days: 24.34 /s Permalink	69.72 /s A: (--; --), L: (56.72 /s; 436.28 %), 14d: 186.49 % Link Permalink	13 /s A: (-56.72 /s; -81.35 %), L: (-13.39 /s; -50.73 %), 14d: -46.58 % Link Permalink	26.39 /s A: (-43.33 /s; -62.15 %), L: (10.14 /s; 62.39 %), 14d: 8.43 % Link Permalink	16.25 /s A: (-53.47 /s; -76.69 %), L: (7.11 /s; 77.82 %), 14d: -33.23 % Link Permalink	9.14 /s A: (-60.58 /s; -86.89 %), L: (0.18 /s; 2.02 %), 14d: -62.45 % Link Permalink	8.96 /s A: (-60.76 /s; -87.15 %), L: (-3.37 /s; -27.35 %), 14d: -63.2 % Link Permalink	12.33 /s A: (-57.39 /s; -82.32 %), L: (-61.25 /s; -83.24 %), 14d: -49.34 % Link Permalink	73.57 /s A: (3.86 /s; 5.53 %), L: (53.14 /s; 260.06 %), 14d: 202.33 % Link Permalink	20.43 /s A: (-49.29 /s; -70.69 %), L: (-6.04 /s; -22.82 %), 14d: -16.03 % Link Permalink	26.48 /s A: (-43.24 /s; -62.03 %), L: (-2.45 /s; -8.47 %), 14d: 8.79 % Link Permalink	28.92 /s A: (-40.8 /s; -58.51 %), L: (18.57 /s; 179.37 %), 14d: 18.85 % Link Permalink	10.35 /s A: (-59.37 /s; -85.15 %), L: (-1.2 /s; -10.35 %), 14d: -57.46 % Link Permalink	11.55 /s A: (-58.17 /s; -83.44 %), L: (-2.07 /s; -15.18 %), 14d: -52.54 % Link Permalink	13.62 /s A: (-56.1 /s; -80.47 %), L: (0 /s; 0 %), 14d: -44.05 % Link Permalink
LoadTestRegression-ES-ip-10-176-34-72 /media/datastore: Timeseries: Throughput Writes(avg) Min: Not Set; Amp: Not Set	6331.79 KB/s A: (--; --), L: (3503.02 KB/s; 123.84 %),	2828.76 KB/s A: (-3503.02 KB/s; -55.32 %),	3351.54 KB/s A: (-2980.25 KB/s; -47.07 %),	3092.48 KB/s A: (-3239.31 KB/s; -51.16 %),	2894.31 KB/s A: (-3437.48 KB/s; -54.29 %),	3001.3 KB/s A: (-3330.49 KB/s; -52.6 %),	3230.92 KB/s A: (-3100.87 KB/s; -48.97 %),	6515.71 KB/s A: (183.93 KB/s; 2.9 %), L: (3168.31	3347.41 KB/s A: (-2984.38 KB/s; -47.13 %),	3559.69 KB/s A: (-2772.1 KB/s; -43.78 %), L: (15.53	3544.16 KB/s A: (-2787.63 KB/s; -44.03 %),	2946.86 KB/s A: (-3384.92 KB/s; -53.46 %),	3095.73 KB/s A: (-3236.06 KB/s; -51.11 %),	3109.93 KB/s A: (-3221.86 KB/s; -50.88 %),

https://www.neotys.com/performance-advisory-council/thomas_steinmaurer

SLI/SLO-based evaluation implementation in Keptn

SLIs defined per SLI Provider as YAML

SLI Provider specific queries, e.g: Dynatrace Metrics Query

```
indicators:
  error_rate:     "builtin:service.errors.total.count:merge(0):avg"
  count_dbcalls:  "calc:service.toptestdbcalls:merge(0):sum"
  jvm_memory:     "builtin:tech.jvm.memory.pool.committed:merge(0):sum"
```

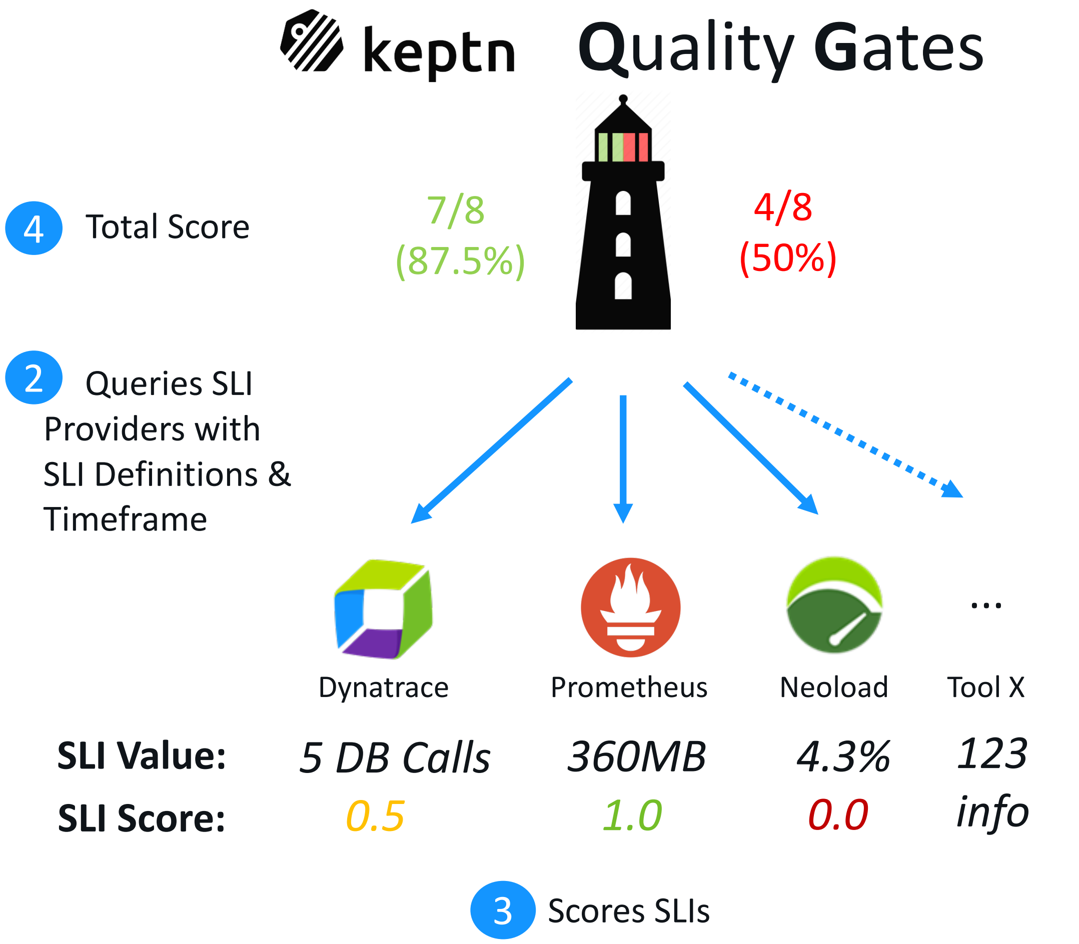
SLOs defined on Keptn Service Level as YAML

List of objectives with fixed or relative pass & warn criteria

```
objectives:
  - sli: error_rate
    pass:
    - criteria:
      - "<=1“  # We expect a max error rate of 1%
  - sli: jvm_memory
  - sli: count_dbcalls
    pass:
    - criteria:
      - "=+2%"  # We allow a 2% increase in DB Calls to previous runs
    warning:
    - criteria:
      - "<=10"  # We expect no more than 10 DB Calls per TX
total_score:
  pass: "90%"
  warning: "75%"
```

1

```
$ keptn start-evaluation 30m myservice sli.yaml slo.yaml
```



keptn Quality Gates

4 Total Score

2 Queries SLI Providers with SLI Definitions & Timeframe

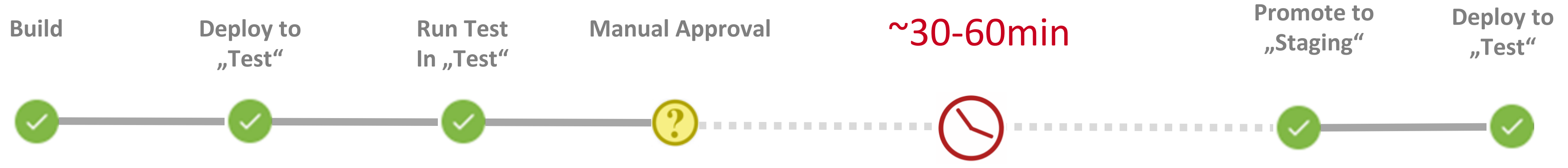
	Dynatrace	Prometheus	Neoload	Tool X
SLI Value:	*5 DB Calls*	*360MB*	*4.3%*	*123 info*
SLI Score:	*0.5*	*1.0*	*0.0*	

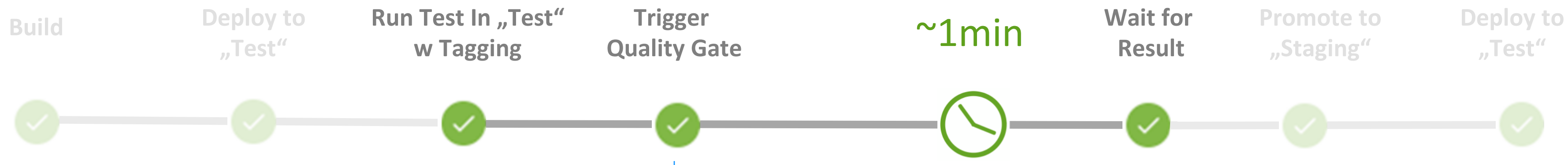
3 Scores SLIs

Solution: Automate Approval through SLI/SLO-based Quality Gates

GitLab

Jenkins





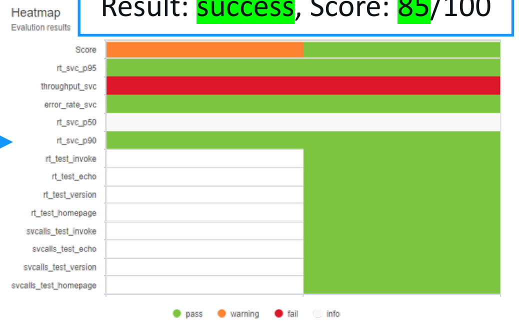
Tagging

SLI & SLO

```
Rt(p95)  <  500ms
#ofSQLs  <=   5
cpu(max)<   80%
Java GC  <    2%
...
```

Validate SLOs

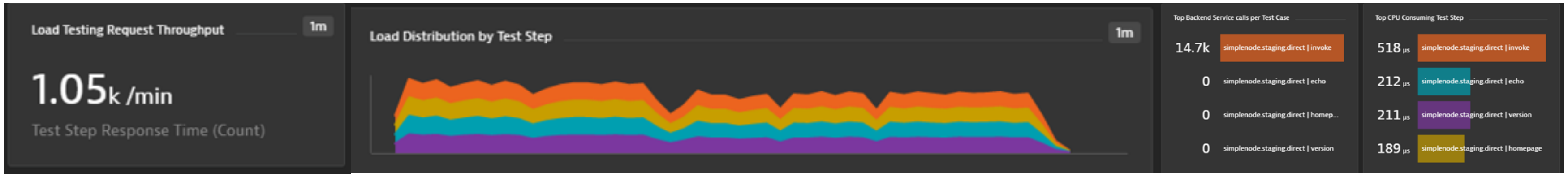
Result: success, Score: 85/100



Pull SLIs for Testing time frame

keptn

Observability



Load Testing Request Throughput 1m

1.05k /min

Test Step Response Time (Count)

Load Distribution by Test Step 1m

Top Backend Service calls per Test Case

14.7k	simplenode.staging.direct | invoke
0	simplenode.staging.direct | echo
0	simplenode.staging.direct | homep...
0	simplenode.staging.direct | version

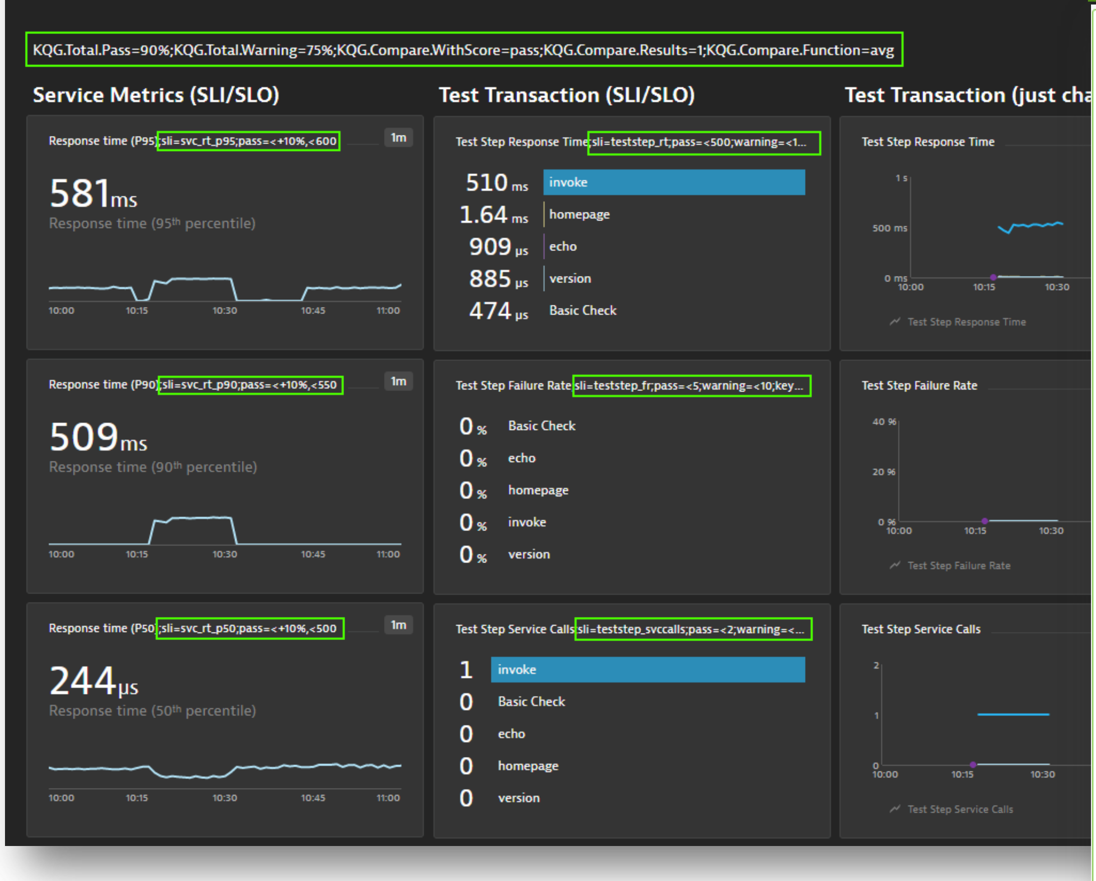
Top CPU Consuming Test Step

518 µs	simplenode.staging.direct | invoke
212 µs	simplenode.staging.direct | echo
211 µs	simplenode.staging.direct | version
189 µs	simplenode.staging.direct | homepage

Demo: Automated SLI/SLO Validation based on Dynatrace Dashboards

You: Just build a dashboard!

keptn: Automates the analysis!



KQG.Total.Pass=90%;KQG.Total.Warning=75%;KQG.Compare.WithScore=pass;KQG.Compare.Results=1;KQG.Compare.Function=avg

Service Metrics (SLI/SLO)

Response time (P95);sli=svc_rt_p95;pass=<+10%,<600 1m

581ms
Response time (95th percentile)

Response time (P90);sli=svc_rt_p90;pass=<+10%,<550 1m

509ms
Response time (90th percentile)

Response time (P50;sli=svc_rt_p50;pass=<+10%,<500 1m

244µs
Response time (50th percentile)

Test Transaction (SLI/SLO)

Test Step Response Time;sli=teststep_rt;pass=<500;warning=<1...

510 ms invoke
1.64 ms homepage
909 µs echo
885 µs version
474 µs Basic Check

Test Step Failure Rate sli=teststep_fr;pass=<5;warning=<10;key...

0 % Basic Check
0 % echo
0 % homepage
0 % invoke
0 % version

Test Step Service Calls sli=teststep_svccalls;pass=<2;warning=<...

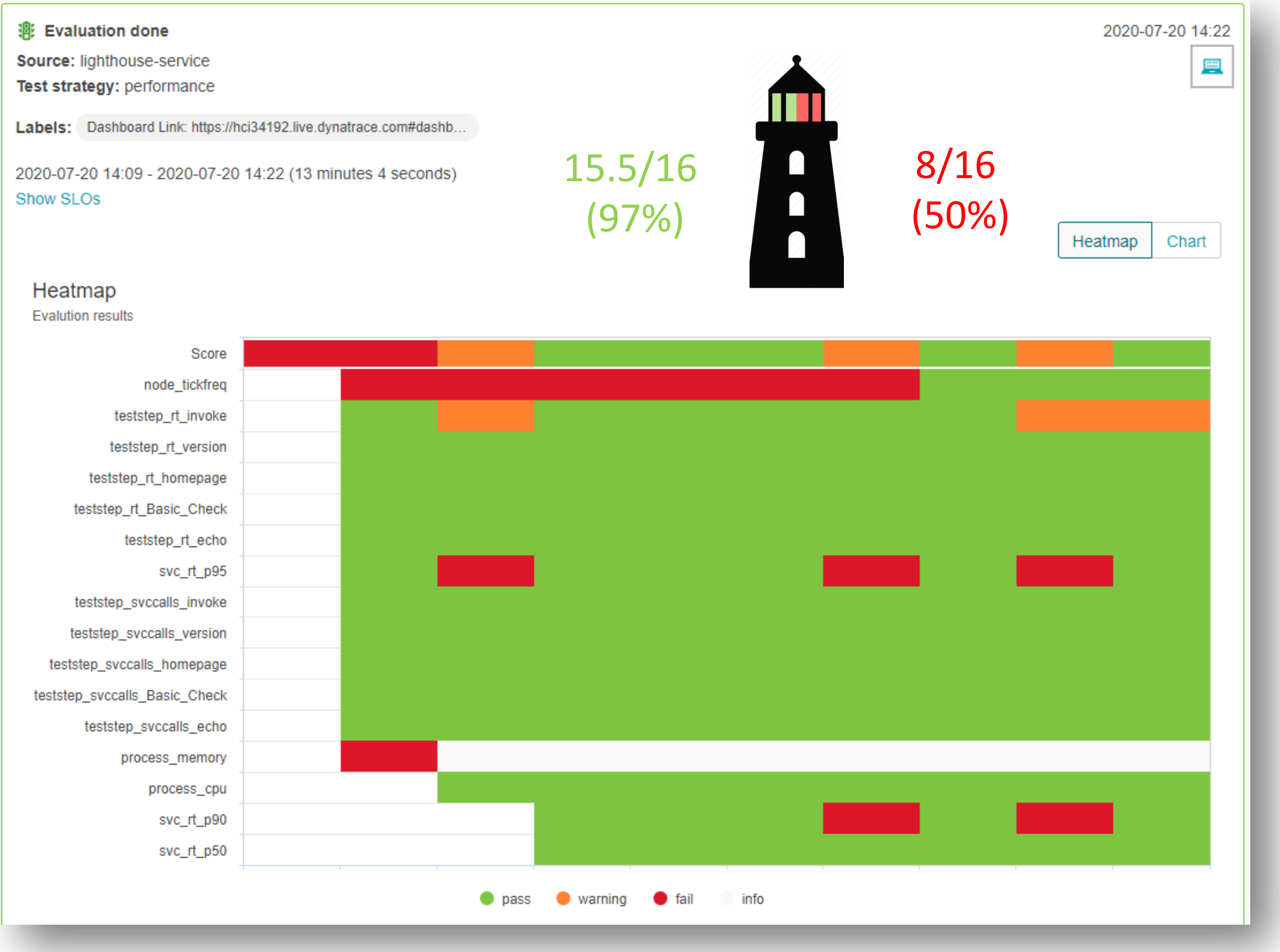
1 invoke
0 Basic Check
0 echo
0 homepage
0 version

Test Transaction (just cha

Test Step Response Time

Test Step Failure Rate

Test Step Service Calls



Evaluation done 2020-07-20 14:22

Source: lighthouse-service
Test strategy: performance

Labels: Dashboard Link: https://hci34192.live.dynatrace.com#dashb...

2020-07-20 14:09 - 2020-07-20 14:22 (13 minutes 4 seconds)
Show SLOs

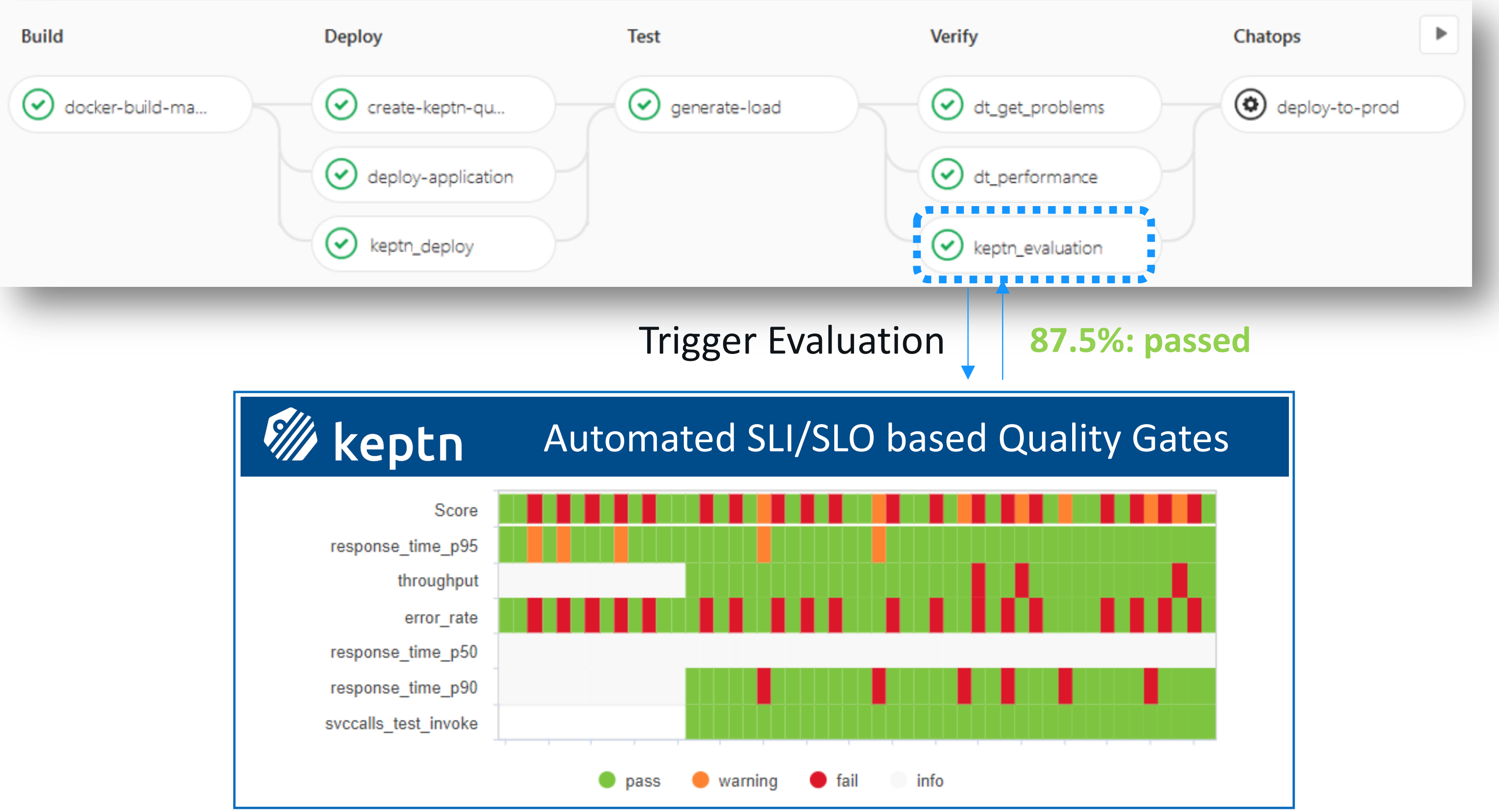
15.5/16 (97%)

8/16 (50%)

Heatmap | Chart

Heatmap
Evalution results

User Example: Automating Build Approvals using Keptn‘s SLIs/SLOs in GitLab



Build
- docker-build-ma...

Deploy
- create-keptn-qu...
- deploy-application
- keptn_deploy

Test
- generate-load

Verify
- dt_get_problems
- dt_performance
- keptn_evaluation

Chatops
- deploy-to-prod

Trigger Evaluation

87.5%: passed

keptn Automated SLI/SLO based Quality Gates

Christian Heckelmann

Senior Systems Engineer

ERT Confidence at every turn™

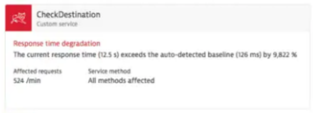
Use Case #2

Automated Remediation

Through a closed loop event-driven remediation workflow

https://github.com/keptn, www.keptn.sh

Keptn – Closed-Loop Remediation with Keptn 0.7



CheckDestination
Custom service

Response time degradation

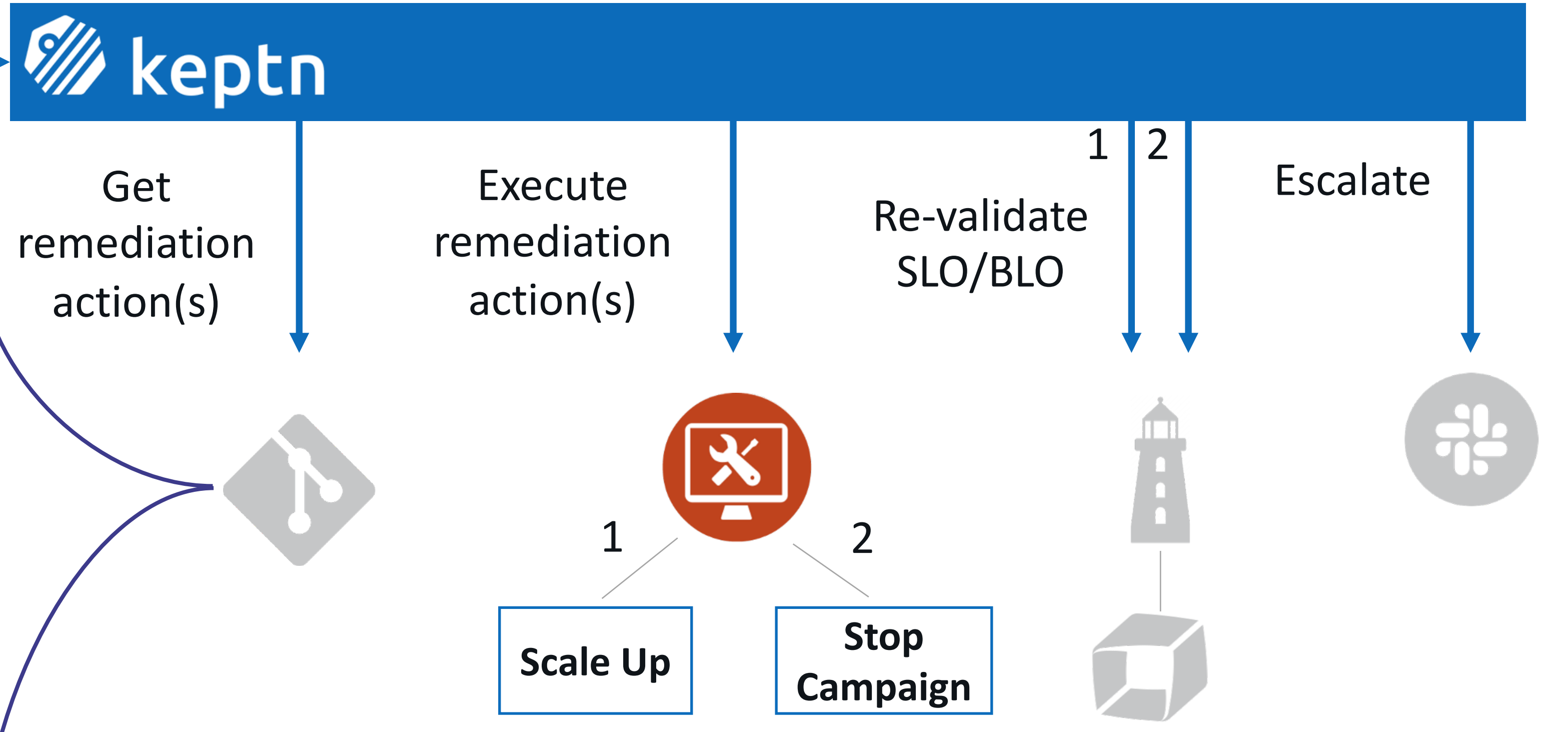
The current response time (12.5 s) exceeds the auto-detected baseline (126 ms) by 9,822 %

Affected requests
524 /min

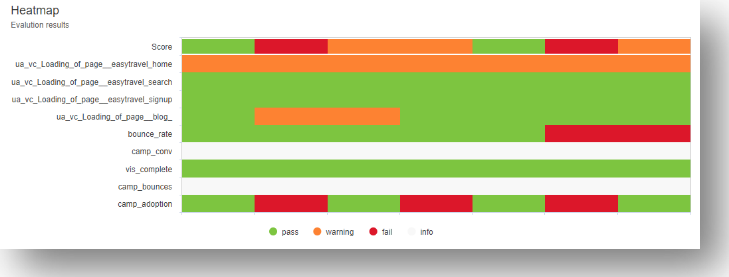
Service method
All methods affected

Problem: Conversion Rate Dropped

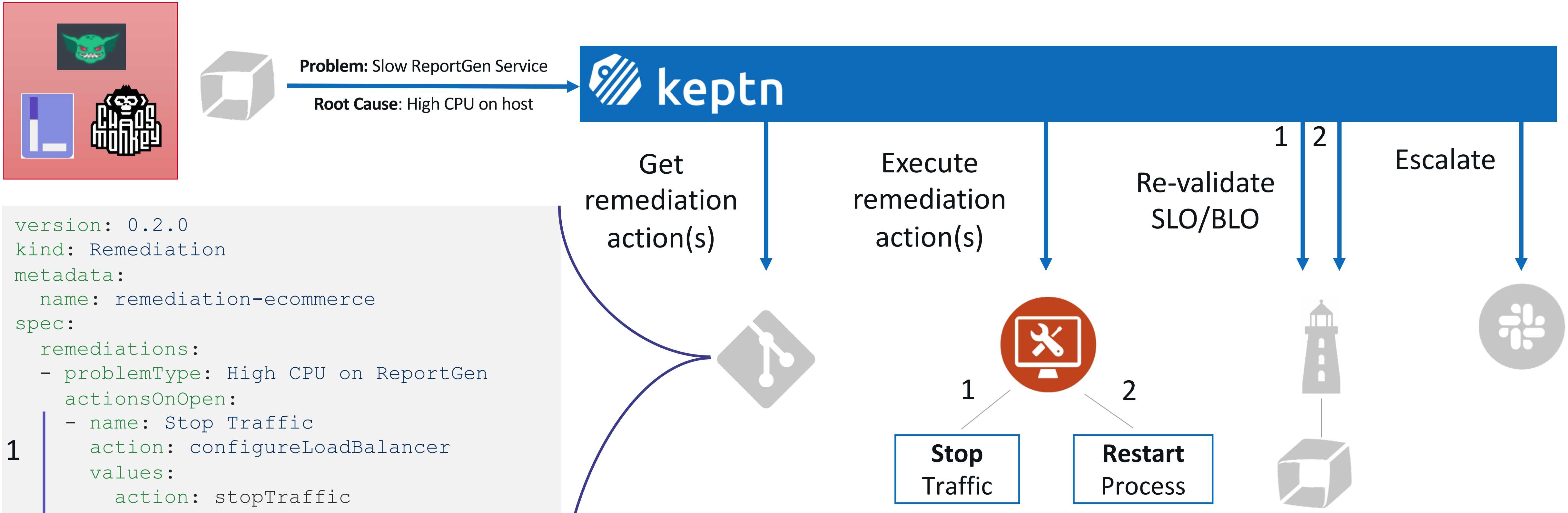
Root Cause: CPU Pressure



```
  version: 0.2.0
  kind: Remediation
  metadata:
    name: remediation-ecommerce
  spec:
    remediations:
    - problemType: Conversion Rate Dropped
      actionsOnOpen:
      - name: Scaling ReplicaSet by 1
1       action: scaling
        values:
          increment: +1
      - name: Stop Ad Campaign
2       action: googleadtoggle
        values:
          enable: off
          campaign: $campaignid
```

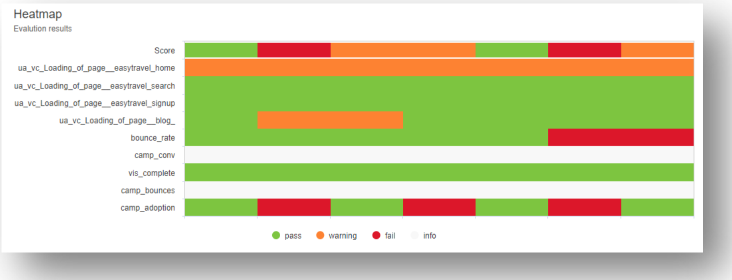


Too risky? Start in Pre-Prod leveraging Chaos Engineering to define & test Auto-Remediation



Problem: Slow ReportGen Service
Root Cause: High CPU on host
keptn
Get remediation action(s)
Execute remediation action(s)
Re-validate SLO/BLO
1
2
Escalate
1
2
Stop Traffic
Restart Process

```
  version: 0.2.0
  kind: Remediation
  metadata:
    name: remediation-ecommerce
  spec:
    remediations:
    - problemType: High CPU on ReportGen
      actionsOnOpen:
      - name: Stop Traffic
1       action: configureLoadBalancer
        values:
          action: stopTraffic
          ip: $problem.hostIp
2     - name: Restart Process
        action: executeAnsible
        values:
          script: restartProcess
          process: $problem.processID
```

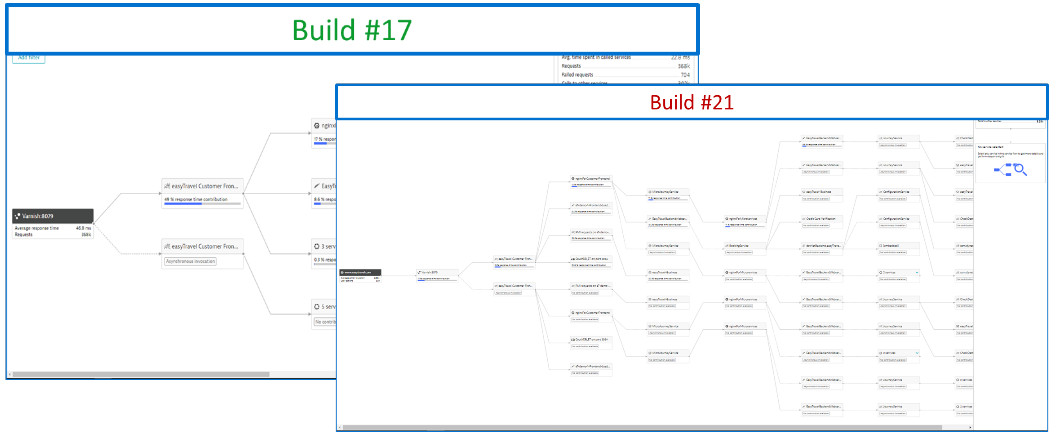


To wrap it up …

What you should have learned today is that

Automate Distributed Problem Detection & Remediation

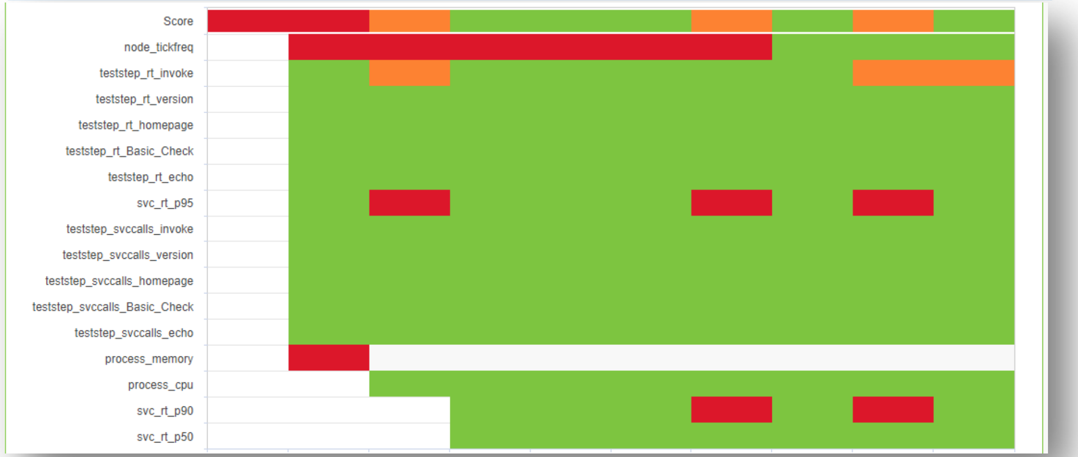
#1 Understand your Patterns & Drive Metrics



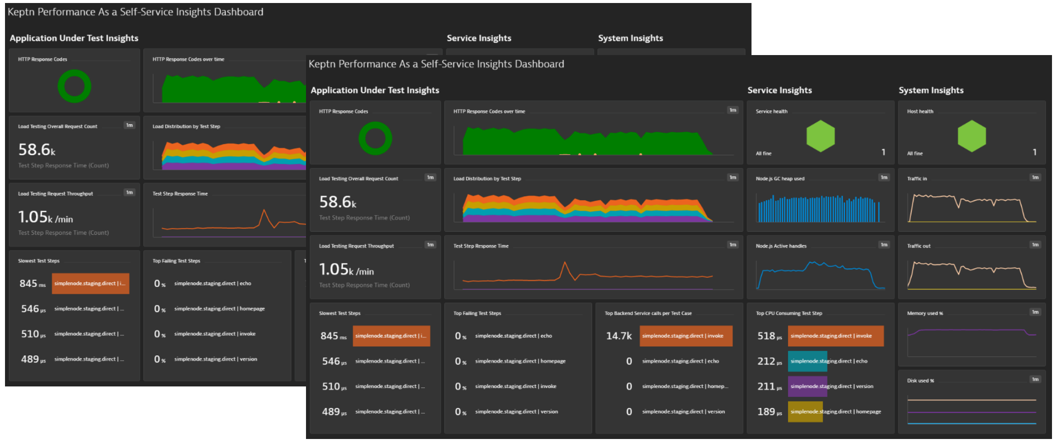
Build #17

Build #21

#3 Let Keptn automate the analysis



#2 Derive and monitor your metrics (SLIs/SLOs)



Keptn Performance As a Self-Service Insights Dashboard

Application Under Test Insights

Service Insights

System Insights

HTTP Response Codes

HTTP Response Codes over time

Service health: All fine 1

Host health: All fine 1

Load Testing Overall Request Count: 58.6k Test Step Response Time (Count)

Load Distribution by Test Step

Node.js GC heap used

Traffic in

Load Testing Request Throughput: 1.05k /min Test Step Response Time (Count)

Test Step Response Time

Node.js Active handles

Traffic out

Slowest Test Steps	
845 ms	simplenode.staging.direct | i...
546 µs	simplenode.staging.direct | ...
510 µs	simplenode.staging.direct | ...
489 µs	simplenode.staging.direct | ...

Top Failing Test Steps	
0 %	simplenode.staging.direct | echo
0 %	simplenode.staging.direct | homepage
0 %	simplenode.staging.direct | invoke
0 %	simplenode.staging.direct | version

Top Backend Service calls per Test Case	
14.7k	simplenode.staging.direct | invoke
0	simplenode.staging.direct | echo
0	simplenode.staging.direct | homep...
0	simplenode.staging.direct | version

Top CPU Consuming Test Step	
518 µs	simplenode.staging.direct | invoke
212 µs	simplenode.staging.direct | echo
211 µs	simplenode.staging.direct | version
189 µs	simplenode.staging.direct | homepage

Memory used %

Disk used %

#4 Integrate Keptn into Delivery & Operations

GitLab

Jenkins

APACHE JMeter™

NeoLoad

THANK YOU!

Automatically Detect the Performance & Scalability Issues in Distributed Architectures

"And integrate this in your delivery pipeline with keptn*"*

Andreas Grabner

DevOps Activist at Dynatrace

DevRel for Keptn

@grabnerandi, https://www.linkedin.com/in/grabnerandi

Follow us @keptnProject

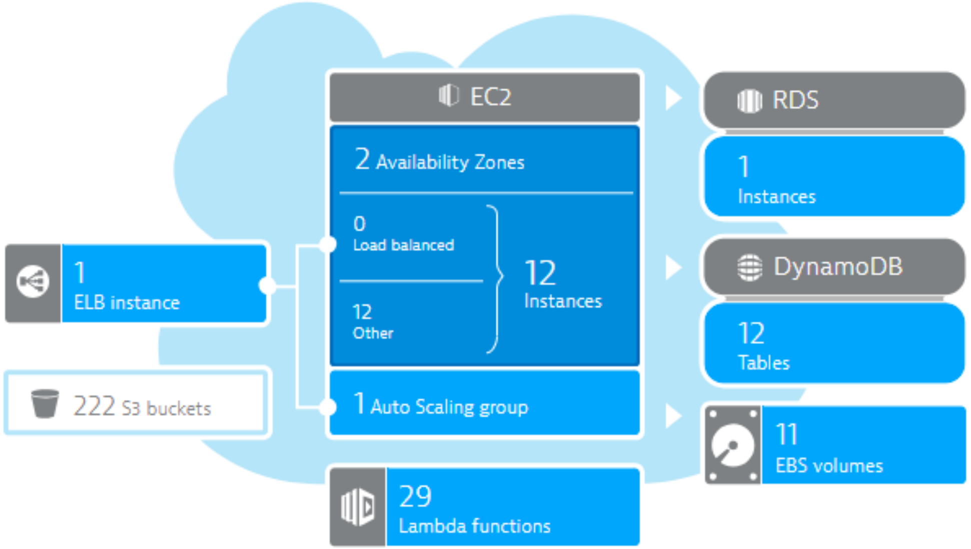
Star us @ https://github.com/keptn/keptn

Slack Us @ https://slack.keptn.sh

We are a Cloud Native Computing Foundation Sandbox project.

More examples

Example #1: Building Monitoring for AWS



EC2

2 Availability Zones

0 Load balanced

12 Other

12 Instances

1 Auto Scaling group

1 ELB instance

222 S3 buckets

RDS

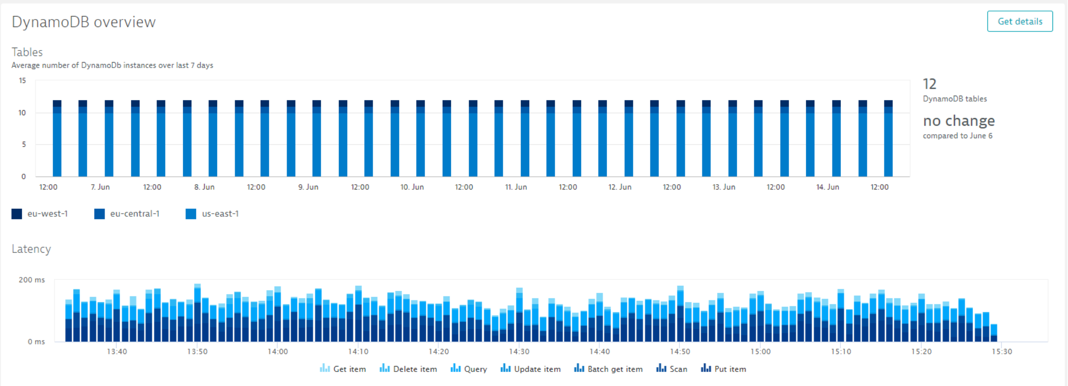
1 Instances

DynamoDB

12 Tables

11 EBS volumes

29 Lambda functions



$0.01 / 1000 Calls

907 Calls

41 sec

97 threads

$$$$

Single Fetch

AWS CloudWatch API

104 Calls

21 sec

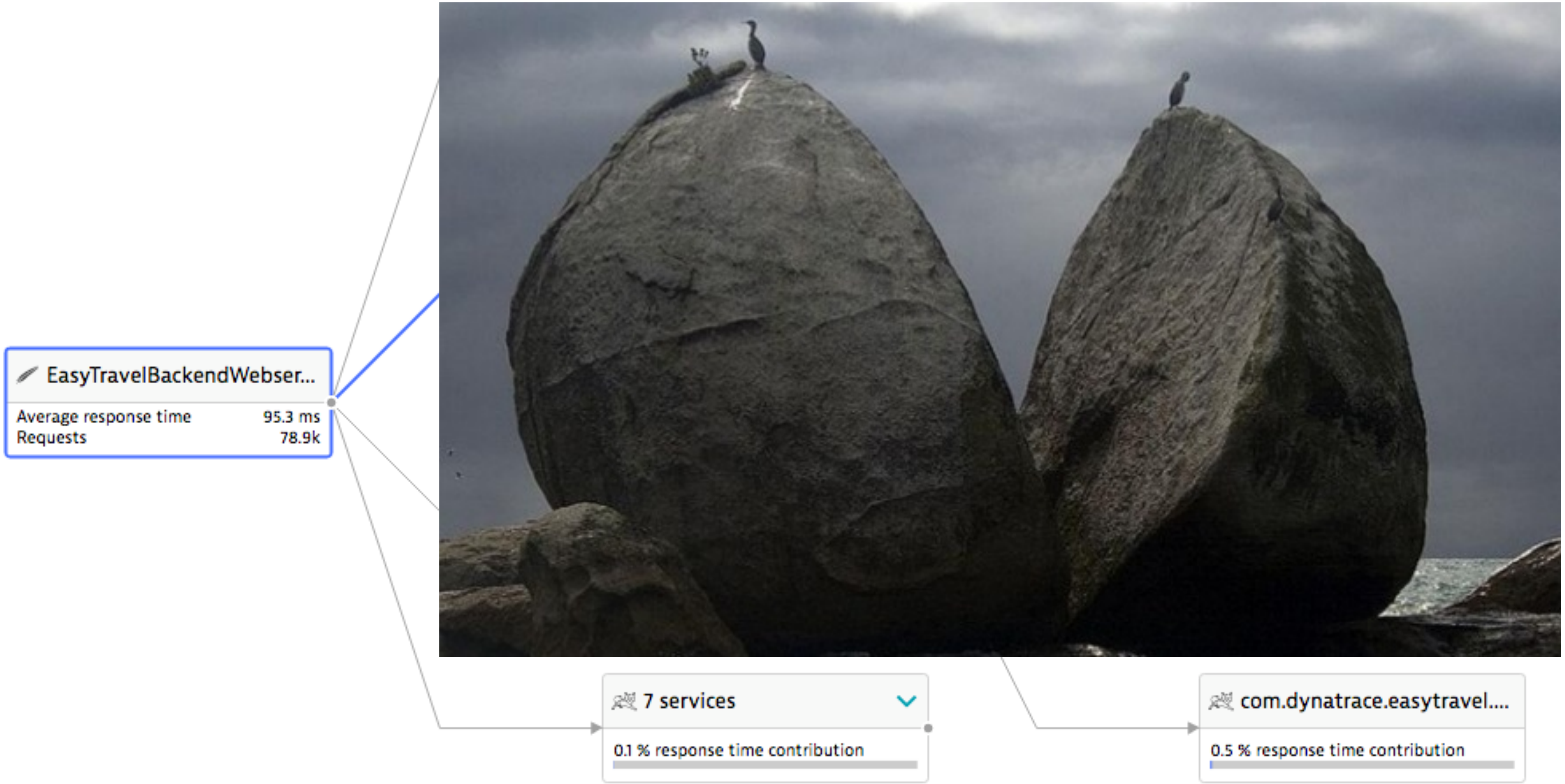
92 threads

Bulk Fetch

$$

Tight Coupling

When “Breaking the Monolith” be aware …



EasyTravelBackendWebser...

Average response time 95.3 ms

Requests 78.9k

7 services

0.1 % response time contribution

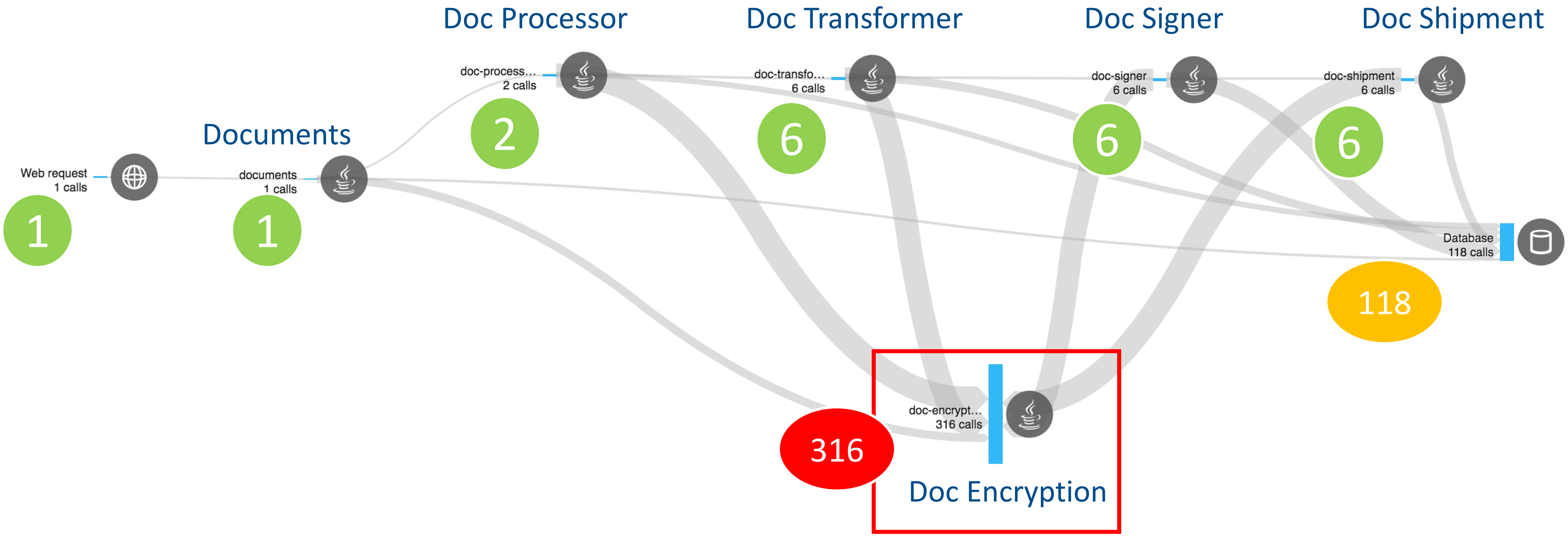
com.dynatrace.easytravel....

0.5 % response time contribution

https://www.dynatrace.com/news/blog/breaking-up-the-monolith-while-migrating-to-the-cloud-in-6-steps/

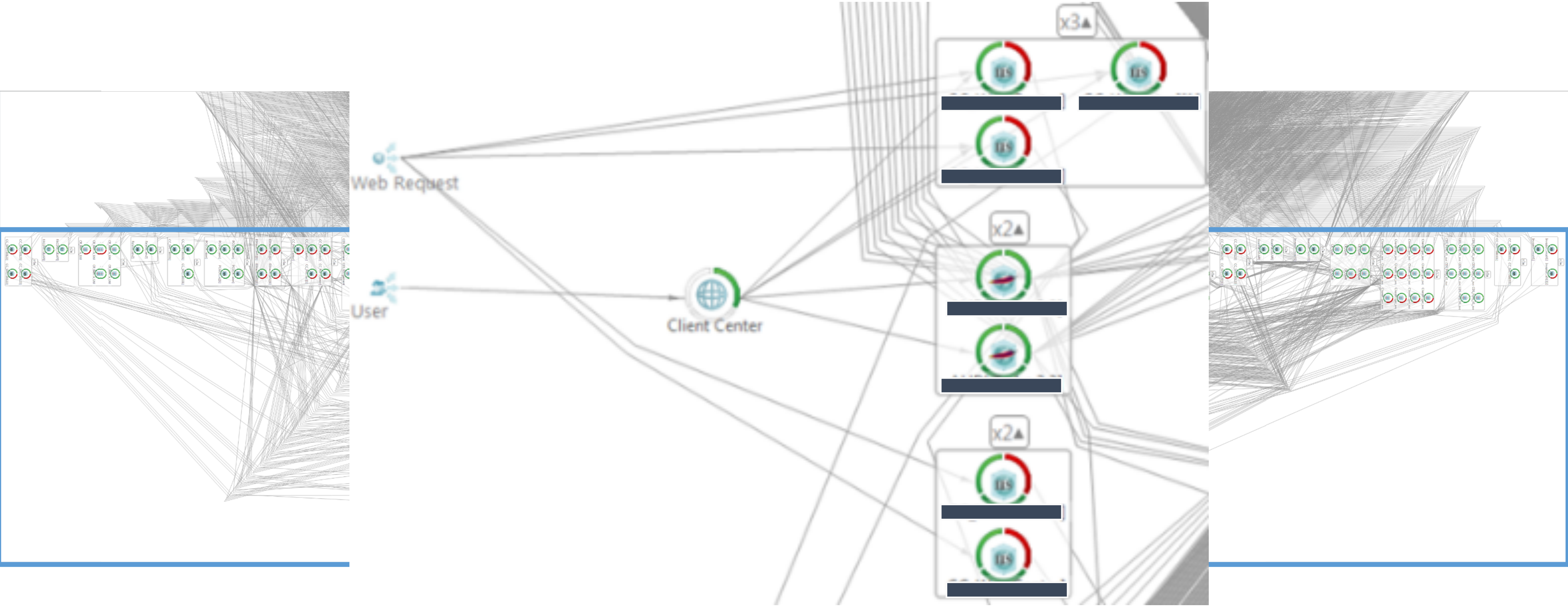
Granularity

Granularity: Encryption carved out into separate service

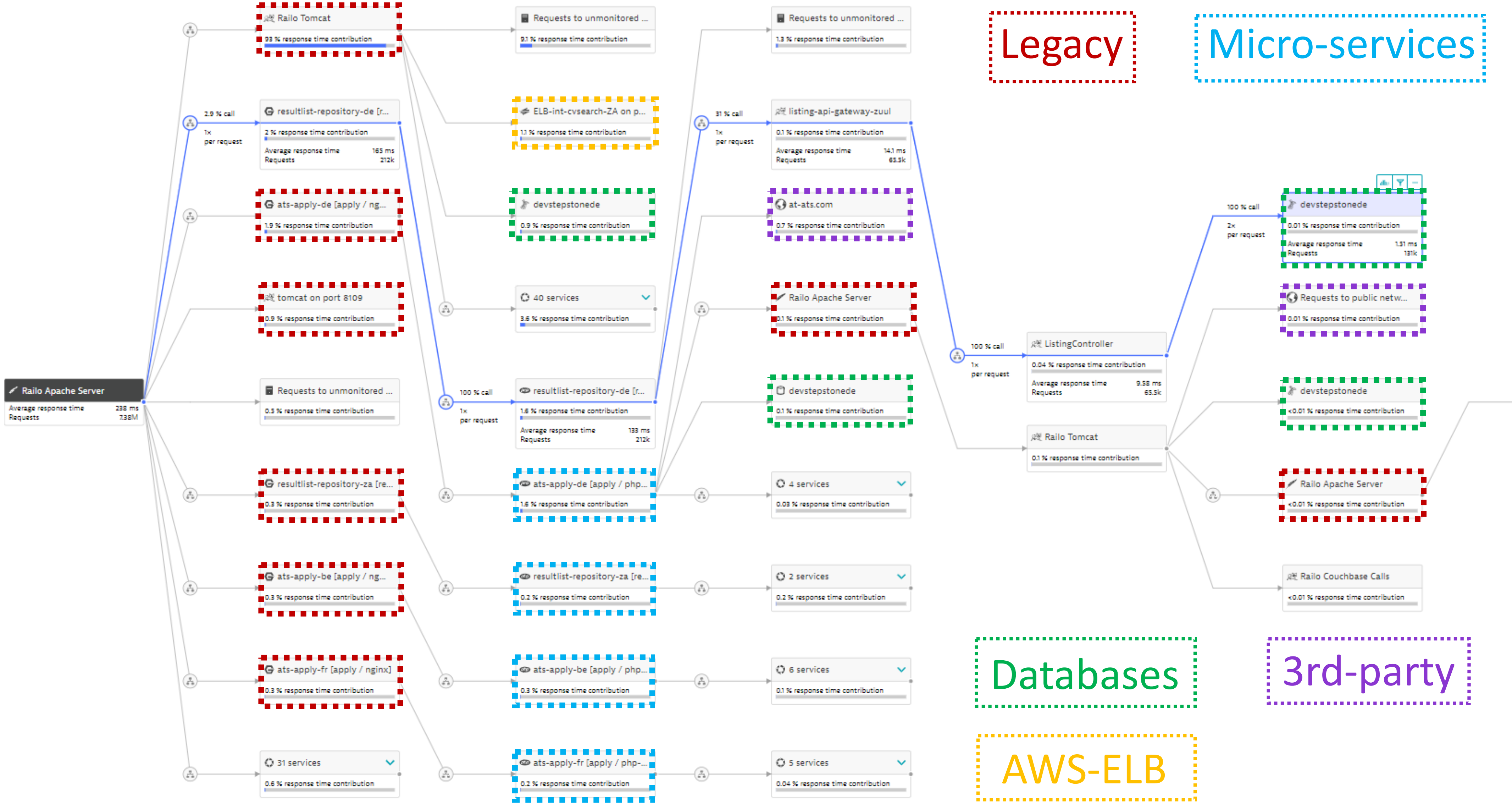


Dependencies

Look beyond the “Tip of the Iceberg”: Understanding Dependencies is critical!



Example from StepStone (AWS Summit Berlin 2019)


Railo Apache Server
Average response time 238 ms
Requests 7.38M
Railo Tomcat
93 % response time contribution
2.9 % call
1x
per request
resultlist-repository-de [r...
2 % response time contribution
Average response time 165 ms
Requests 212k
ats-apply-de [apply / ng...
1.9 % response time contribution
tomcat on port 8109
0.9 % response time contribution
Requests to unmonitored ...
0.5 % response time contribution
resultlist-repository-za [re...
0.3 % response time contribution
ats-apply-be [apply / ng...
0.3 % response time contribution
ats-apply-fr [apply / nginx]
0.3 % response time contribution
31 services
0.6 % response time contribution
Requests to unmonitored ...
9.1 % response time contribution
ELB-int-cvsearch-ZA on p...
1.1 % response time contribution
devstepstonede
0.9 % response time contribution
40 services
3.6 % response time contribution
100 % call
1x
per request
resultlist-repository-de [r...
1.6 % response time contribution
Average response time 133 ms
Requests 212k
ats-apply-de [apply / php...
1.6 % response time contribution
resultlist-repository-za [re...
0.2 % response time contribution
ats-apply-be [apply / php...
0.3 % response time contribution
ats-apply-fr [apply / php-...
0.2 % response time contribution
Requests to unmonitored ...
1.3 % response time contribution
31 % call
1x
per request
listing-api-gateway-zuul
0.1 % response time contribution
Average response time 14.1 ms
Requests 65.5k
at-ats.com
0.7 % response time contribution
Railo Apache Server
0.1 % response time contribution
devstepstonede
0.1 % response time contribution
4 services
0.03 % response time contribution
2 services
0.2 % response time contribution
6 services
0.1 % response time contribution
5 services
0.04 % response time contribution
100 % call
1x
per request
ListingController
0.04 % response time contribution
Average response time 9.58 ms
Requests 65.5k
Railo Tomcat
0.1 % response time contribution
100 % call
2x
per request
devstepstonede
0.01 % response time contribution
Average response time 1.51 ms
Requests 131k
Requests to public netw...
0.01 % response time contribution
devstepstonede
<0.01 % response time contribution
Railo Apache Server
<0.01 % response time contribution
Railo Couchbase Calls
<0.01 % response time contribution

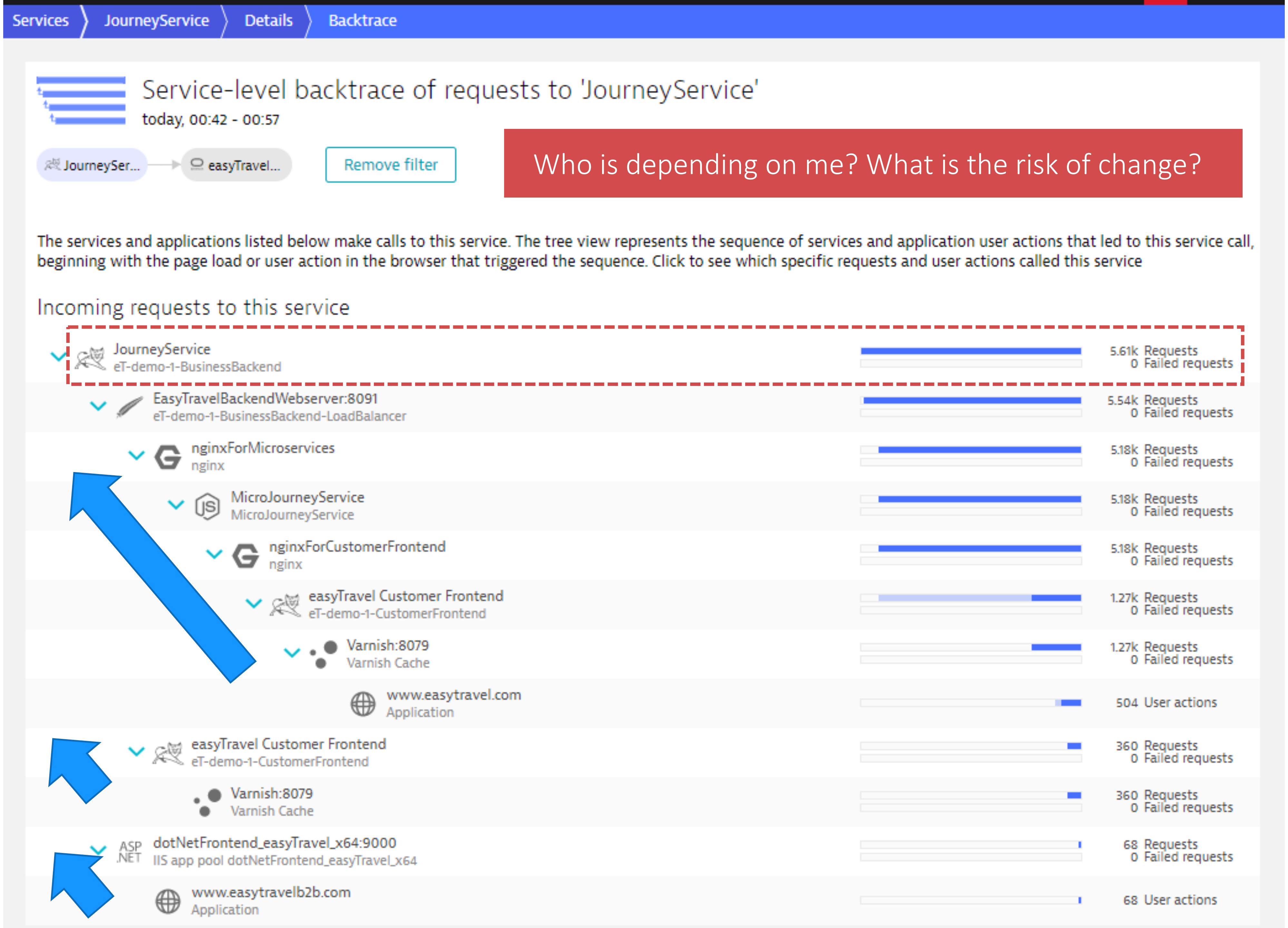
Legacy

Micro-services

Databases

3rd-party

AWS-ELB



Service-level backtrace of requests to 'JourneyService'

today, 00:42 - 00:57

Who is depending on me? What is the risk of change?

The services and applications listed below make calls to this service. The tree view represents the sequence of services and application user actions that led to this service call, beginning with the page load or user action in the browser that triggered the sequence. Click to see which specific requests and user actions called this service

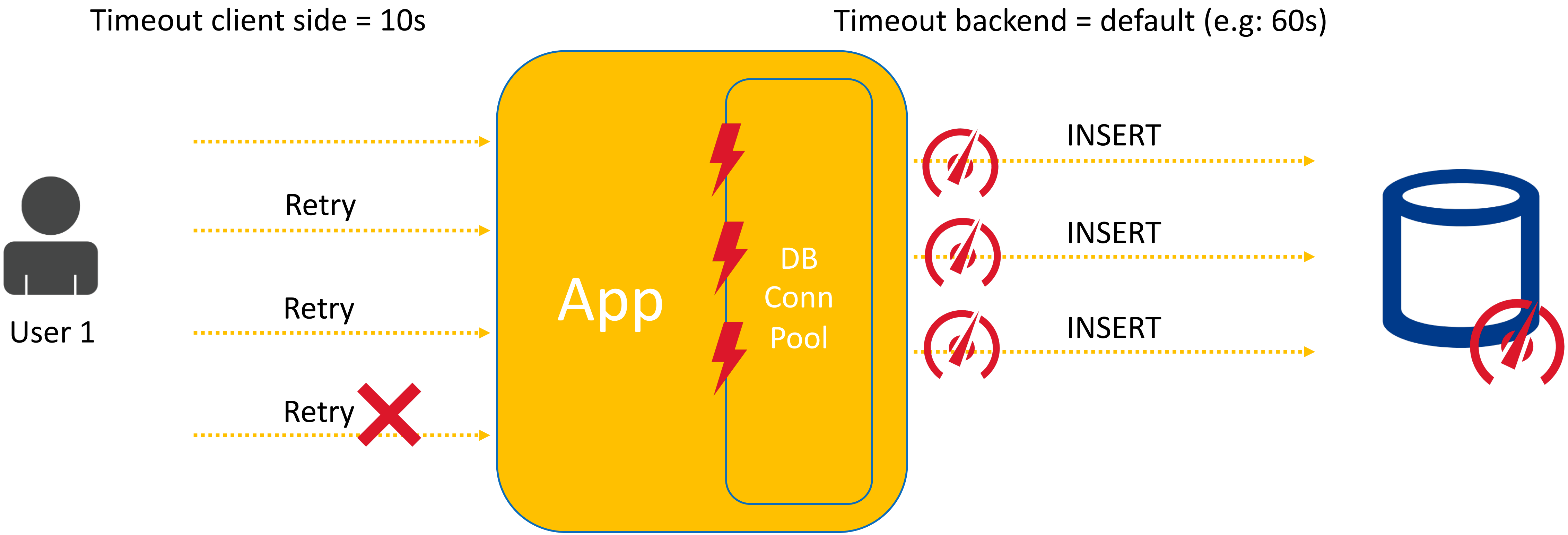
Incoming requests to this service

- JourneyService — eT-demo-1-BusinessBackend — 5.61k Requests, 0 Failed requests
  - EasyTravelBackendWebserver:8091 — eT-demo-1-BusinessBackend-LoadBalancer — 5.54k Requests, 0 Failed requests
    - nginxForMicroservices — nginx — 5.18k Requests, 0 Failed requests
      - MicroJourneyService — MicroJourneyService — 5.18k Requests, 0 Failed requests
        - nginxForCustomerFrontend — nginx — 5.18k Requests, 0 Failed requests
          - easyTravel Customer Frontend — eT-demo-1-CustomerFrontend — 1.27k Requests, 0 Failed requests
            - Varnish:8079 — Varnish Cache — 1.27k Requests, 0 Failed requests
              - www.easytravel.com — Application — 504 User actions
    - easyTravel Customer Frontend — eT-demo-1-CustomerFrontend — 360 Requests, 0 Failed requests
      - Varnish:8079 — Varnish Cache — 360 Requests, 0 Failed requests
  - dotNetFrontend_easyTravel_x64:9000 — IIS app pool dotNetFrontend_easyTravel_x64 — 68 Requests, 0 Failed requests
    - www.easytravelb2b.com — Application — 68 User actions

Timeouts, Retries & Backoff

Credits go to Adrian Hornsby (@adhorn)

Bad Timeout & Retry Settings

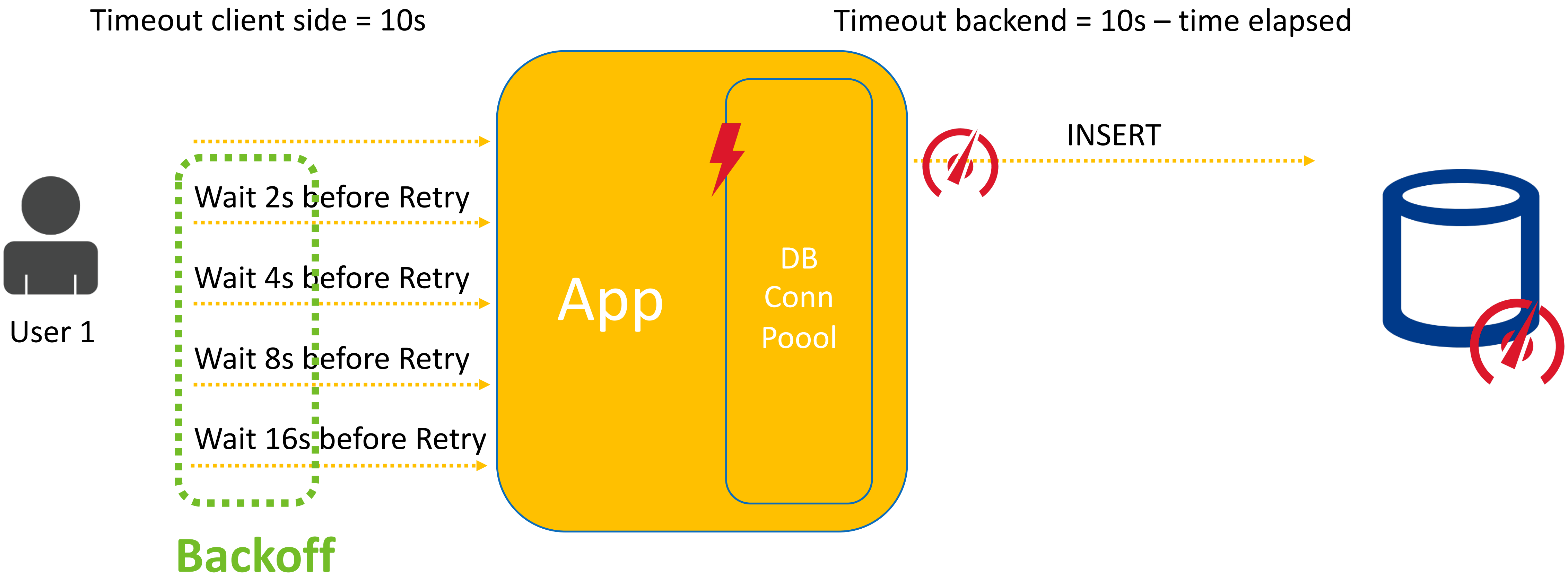


Timeout client side = 10s
Timeout backend = default (e.g: 60s)
User 1
Retry
Retry
Retry
App
DB
Conn
Pool
INSERT
INSERT
INSERT

ERROR: Failed to get connection from pool

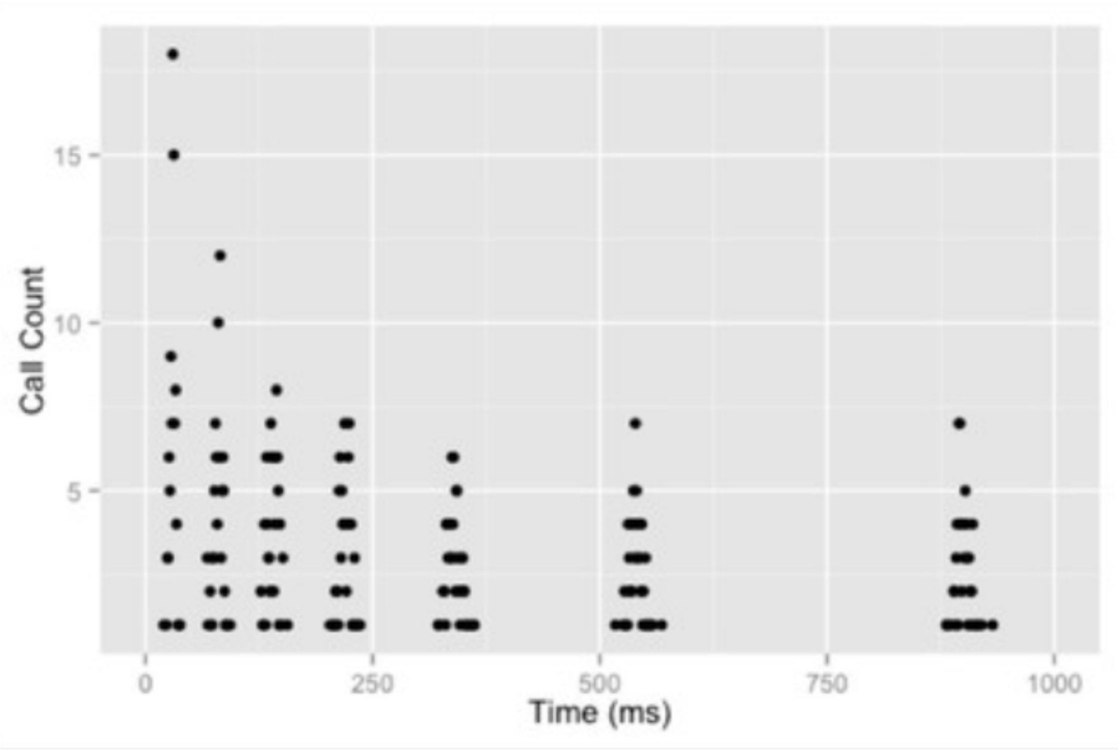
From Adrian Hornsby (@adhorn): https://speakerdeck.com/adhorn/resiliency-and-availability-design-patterns-3742b5ba-e013-4f50-8512-00a65775f478?slide=31

Backoff between Retries

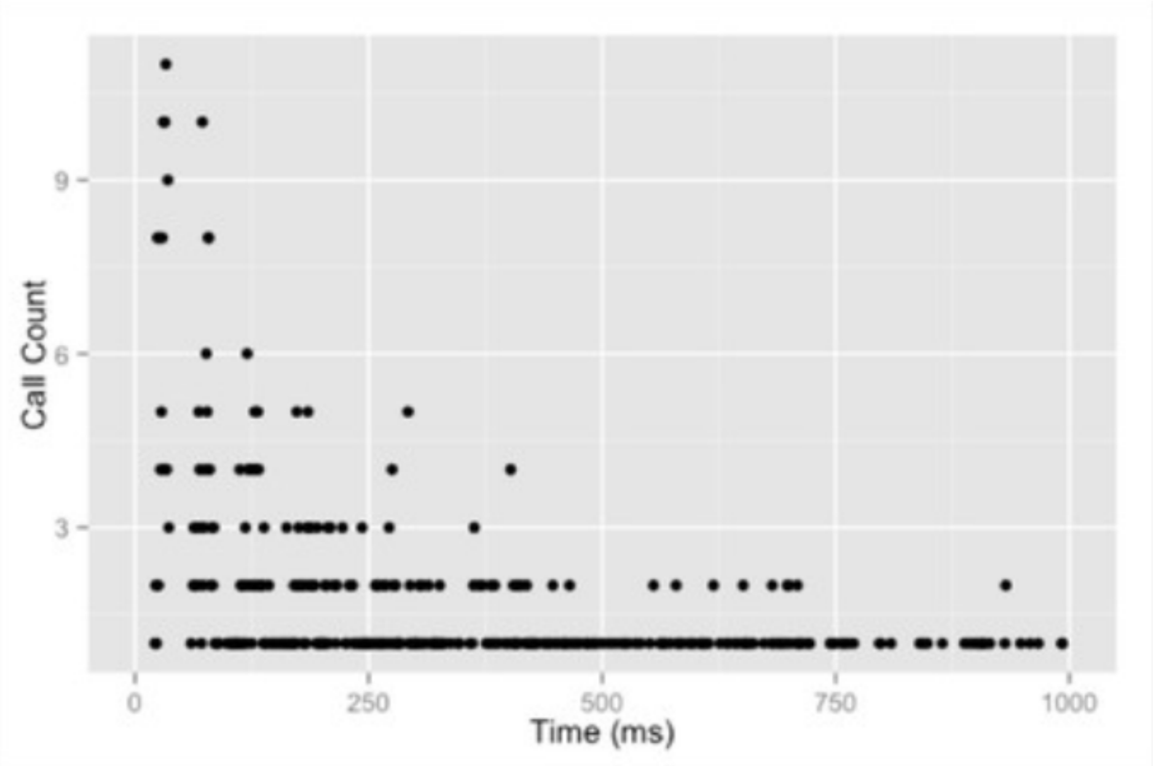


From Adrian Hornsby (@adhorn): https://speakerdeck.com/adhorn/resiliency-and-availability-design-patterns-3742b5ba-e013-4f50-8512-00a65775f478?slide=33

Simple Exponential Backoff is not enough: Add Jitter



No jitter



With jitter

From Adrian Hornsby (@adhorn): https://speakerdeck.com/adhorn/resiliency-and-availability-design-patterns-3742b5ba-e013-4f50-8512-00a65775f478?slide=34