Pinpoint 로 어플리케이션에 대한 Naver 수준의 가시성 확보하기

김성욱
정현길

NAVER

Index

- Microservice Architecture
- Observability in Naver
- Enhance Observability with Pinpoint
- Open source Pinpoint
- Open source Community
- Troubleshooting Distributed Systems
- How Pinpoint Works
- Troubleshooting with Pinpoint
- Future Plans

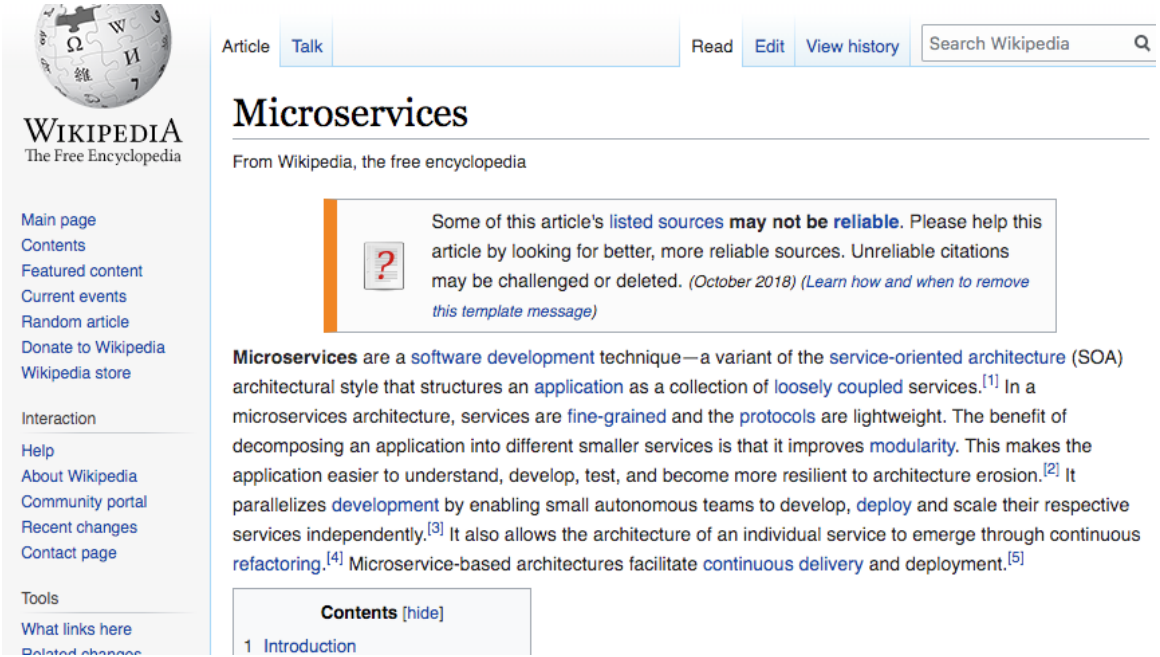
Microservice Architecture

Microservice Architecture

- Development
- Maintenance
- Reliability
- Scalability
- Cost
- Deployment
- Releasing

Microservice Architecture

- Development
- Maintenance
- Reliability
- Scalability
- Cost
- Deployment
- Releasing



WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction

Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools

What links here

Article Talk Read Edit View history Search Wikipedia

Microservices

From Wikipedia, the free encyclopedia

Some of this article's listed sources **may not be reliable**. Please help this article by looking for better, more reliable sources. Unreliable citations may be challenged or deleted. *(October 2018) (Learn how and when to remove this template message)*

Microservices are a software development technique—a variant of the service-oriented architecture (SOA) architectural style that structures an application as a collection of loosely coupled services.[1] In a microservices architecture, services are fine-grained and the protocols are lightweight. The benefit of decomposing an application into different smaller services is that it improves modularity. This makes the application easier to understand, develop, test, and become more resilient to architecture erosion.[2] It parallelizes development by enabling small autonomous teams to develop, deploy and scale their respective services independently.[3] It also allows the architecture of an individual service to emerge through continuous refactoring.[4] Microservice-based architectures facilitate continuous delivery and deployment.[5]

Contents [hide]

1 Introduction

NAVER

Microservice Architecture

- **Development**
- Maintenance
- Reliability
- Scalability
- Cost
- **Deployment**
- Releasing

Microservice-based architectures facilitate continuous delivery and deployment.

Microservice Architecture

New Research Shows 63% of Enterprises Are Adopting Microservices Architectures

Despite new research, 50% of enterprises are unaware of microservices' impact on revenue-generating business processes.

by Tom Smith · Sep. 20, 18 · Microservices Zone · Interview

Like (4) Comment (0) Save Tweet 4,775 Views

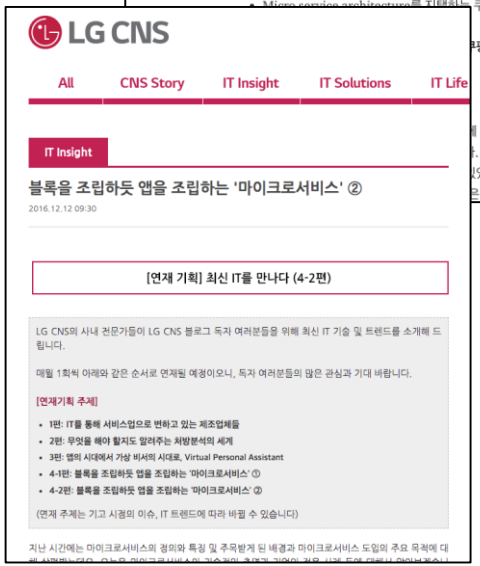
Join the DZone community and get the full member experience. JOIN FOR FREE

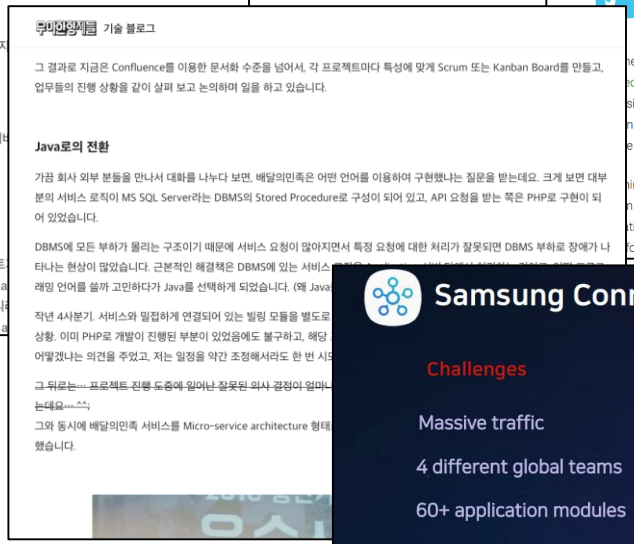
Download DZone's 2019 Scaling DevOps Trend Report to learn how to ensure security as you scale DevOps, why "blameless retrospectives" are a myth, and the key considerations for getting the most out of DevOps as you scale.
Download Now

Presented by DZone

https://dzone.com/articles/right-strategies-for-microservices-deployment

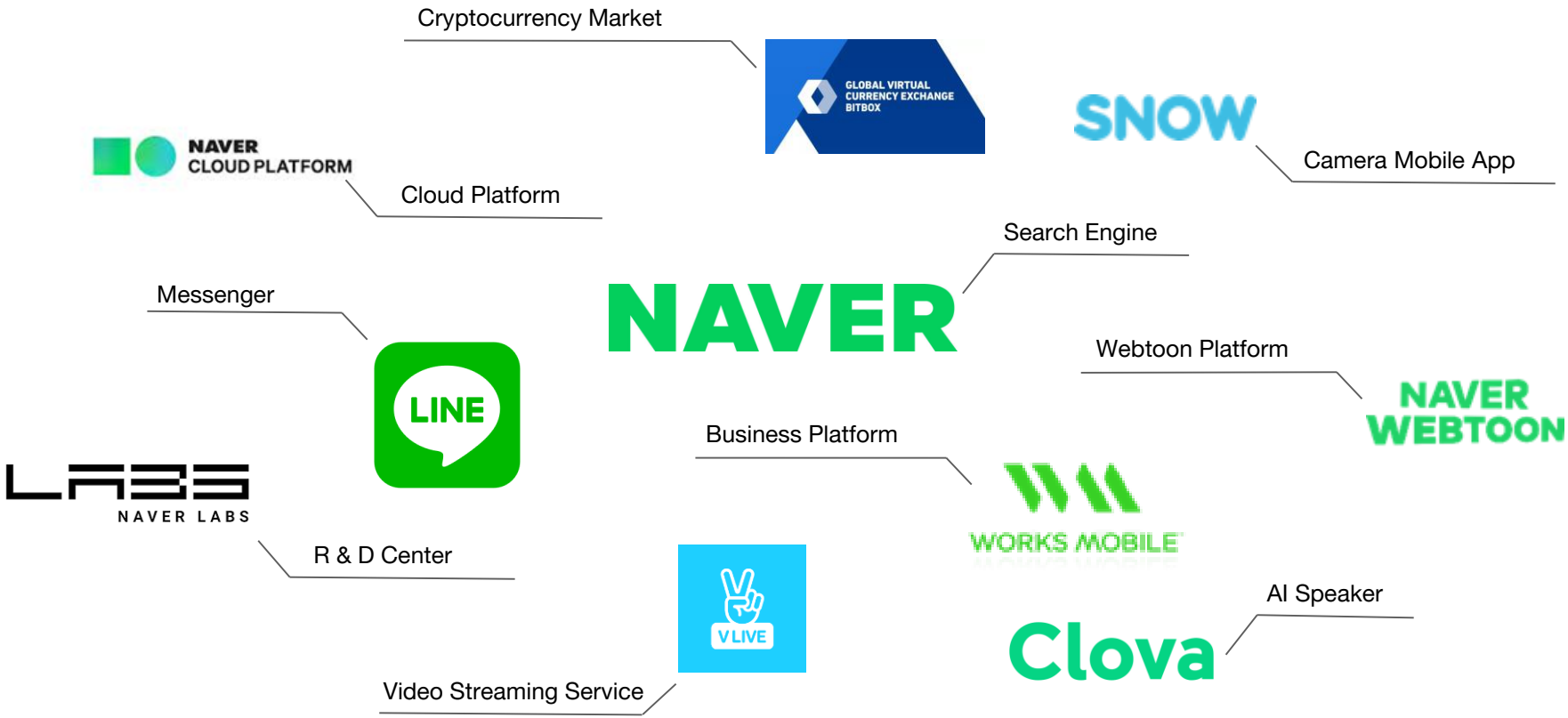
Microservice Architecture





NAVER

Microservice Architecture

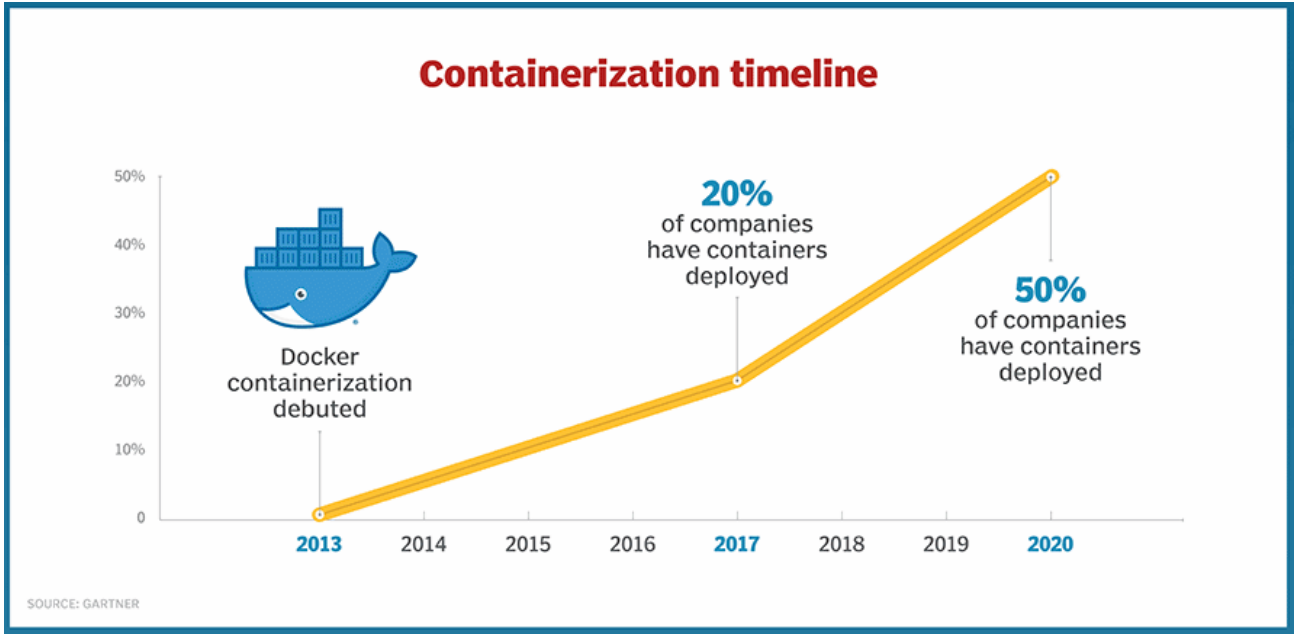


Microservice Architecture

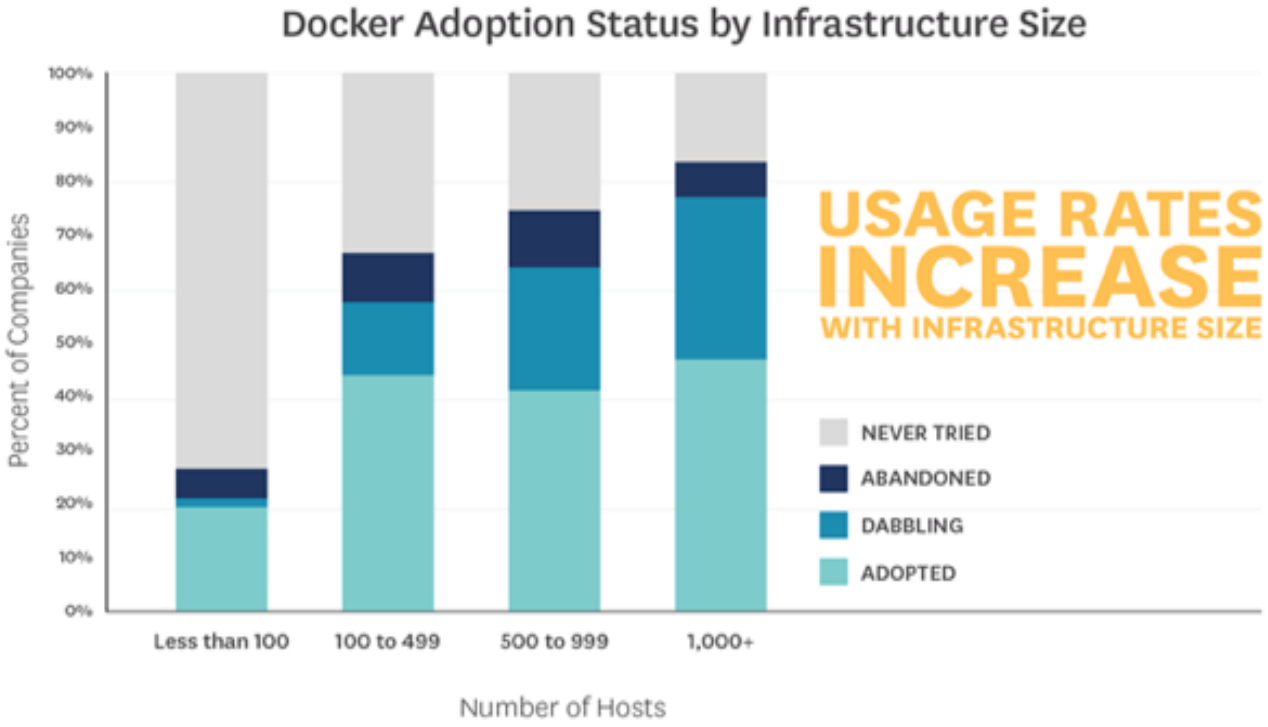
- Deployment Management
- Debugging

Microservice Architecture

- ~~Deployment Management~~
- Debugging

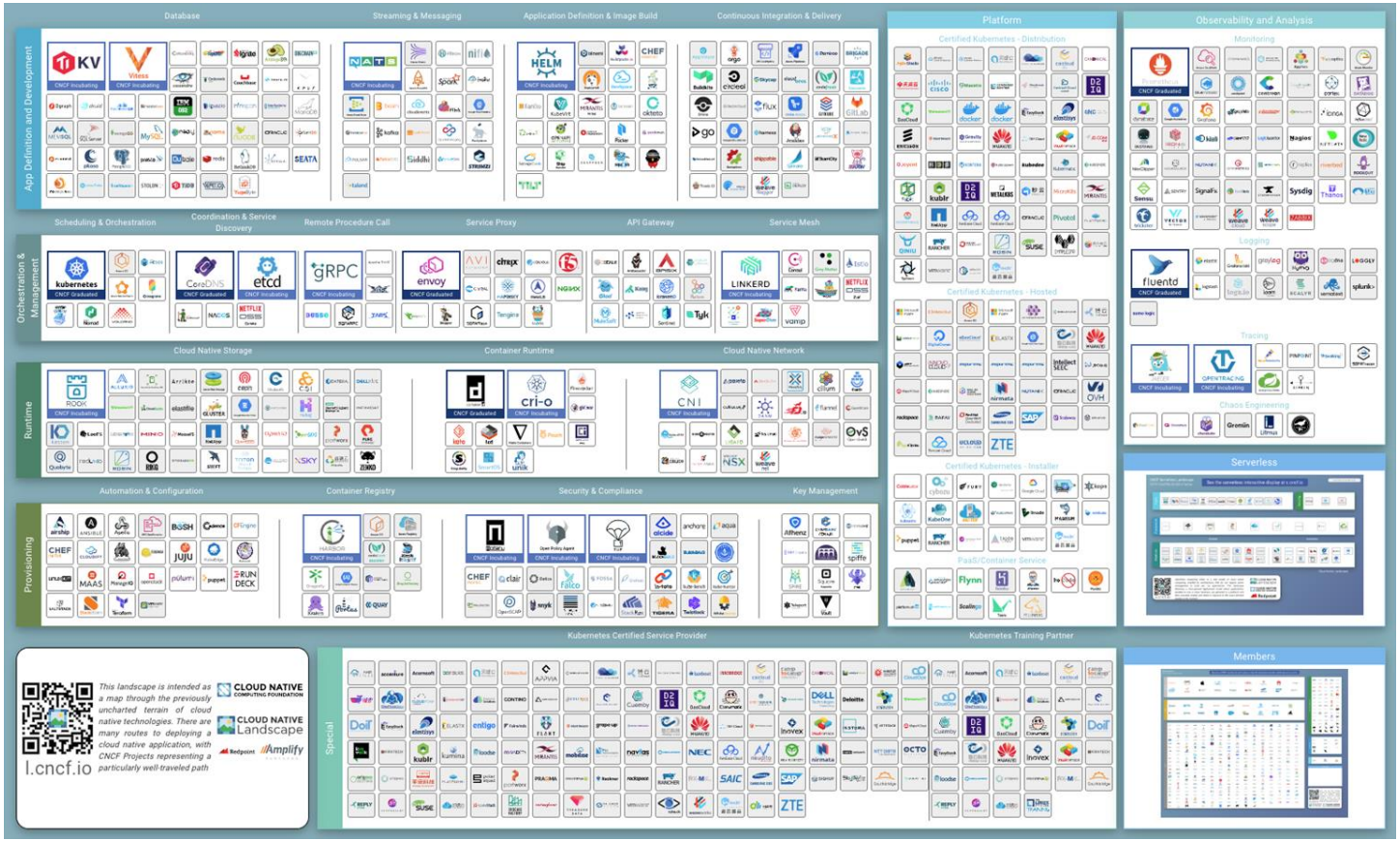


Microservice Architecture



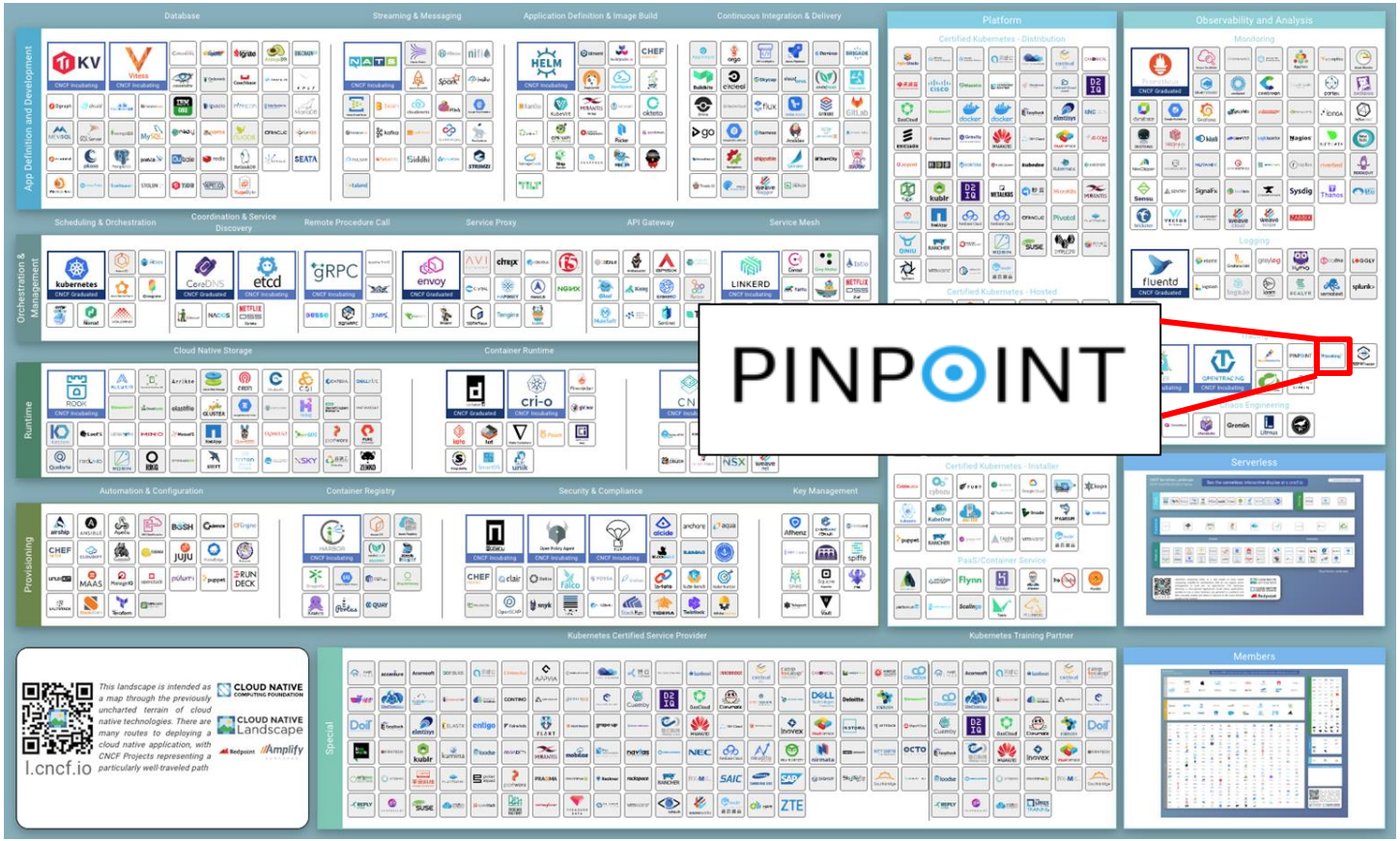
Source: Datadog

Microservice Architecture



NAVER

Microservice Architecture



NAVER

Microservice Architecture

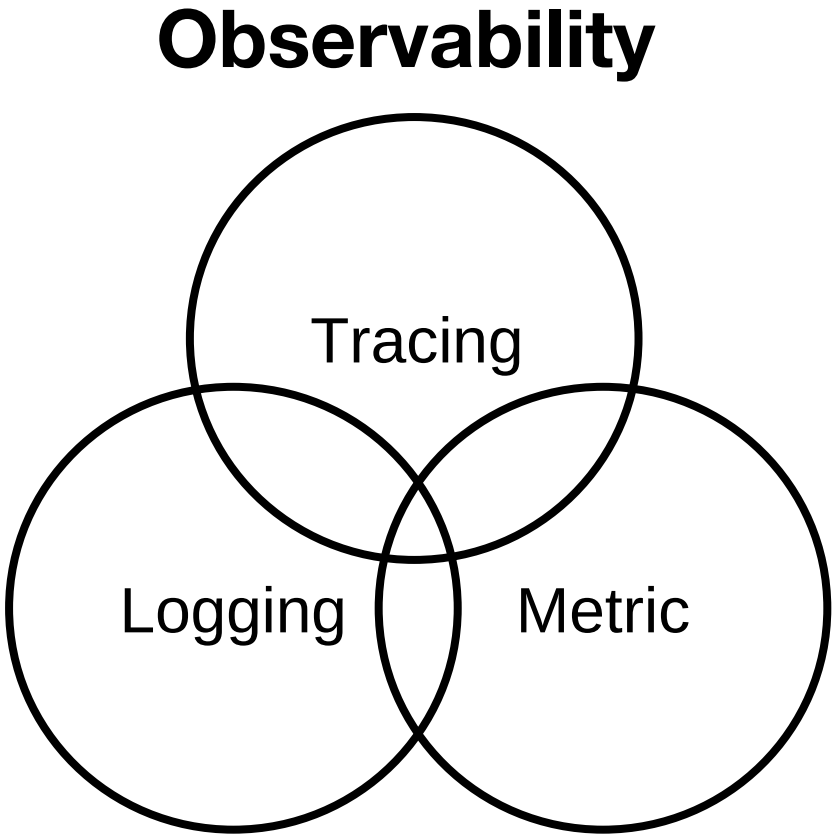
- ~~Deployment Management~~
- Debugging

Observability in Naver

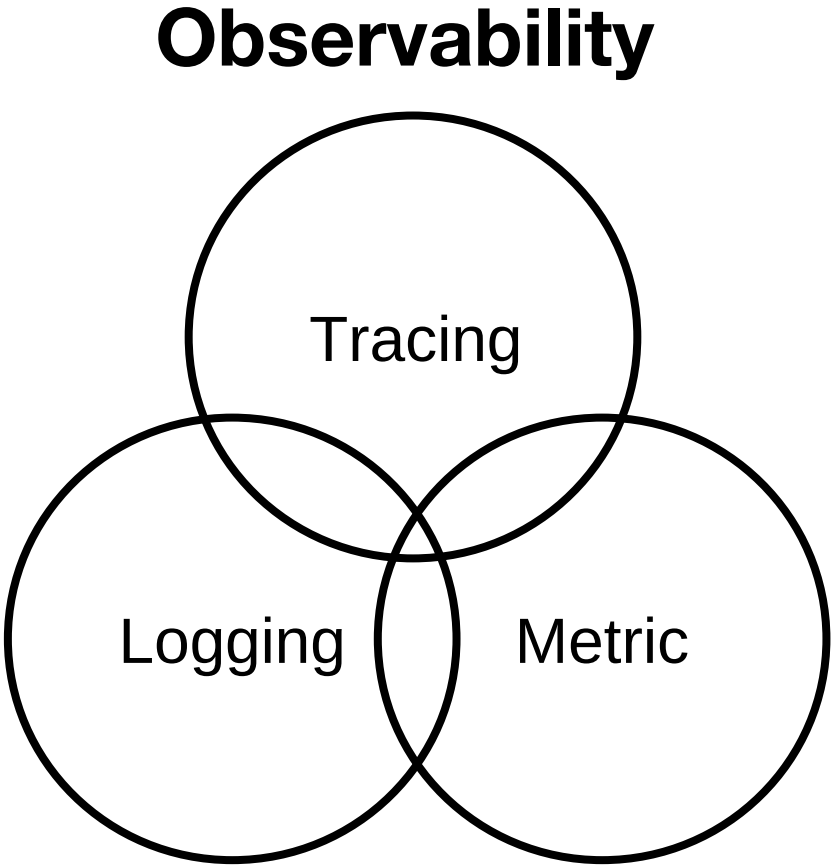
- ~~Deployment Management~~
- ~~Debugging~~

Observability ↑

Observability in Naver



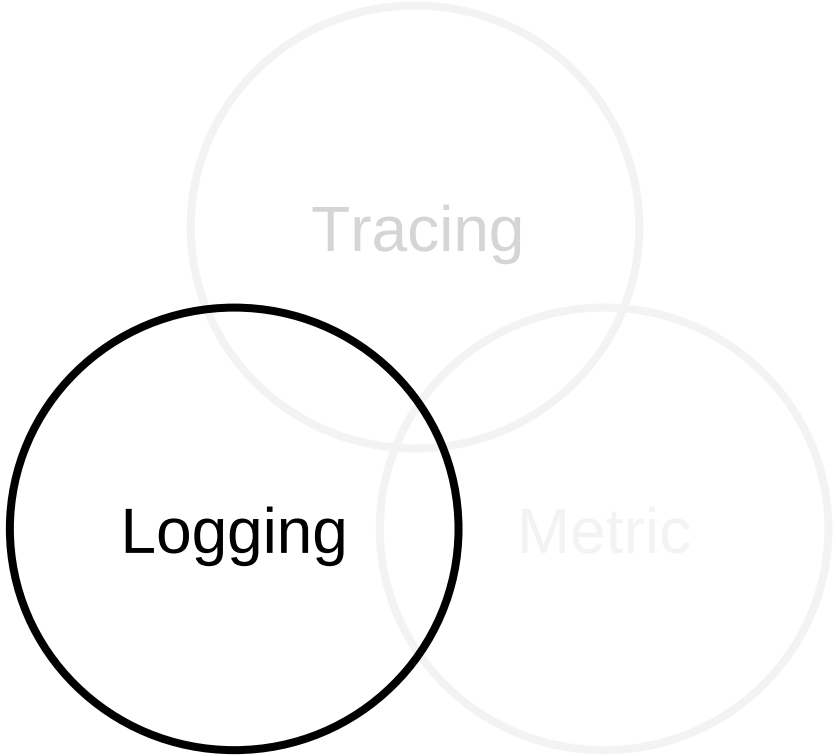
Observability in Naver



그리고 사람

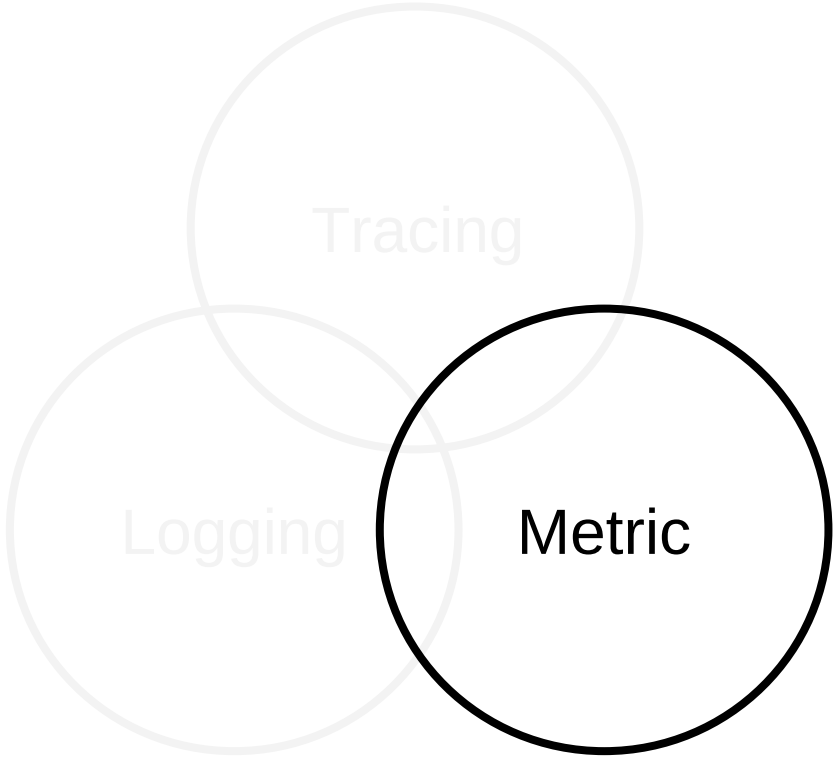
Observability in Naver

Observability



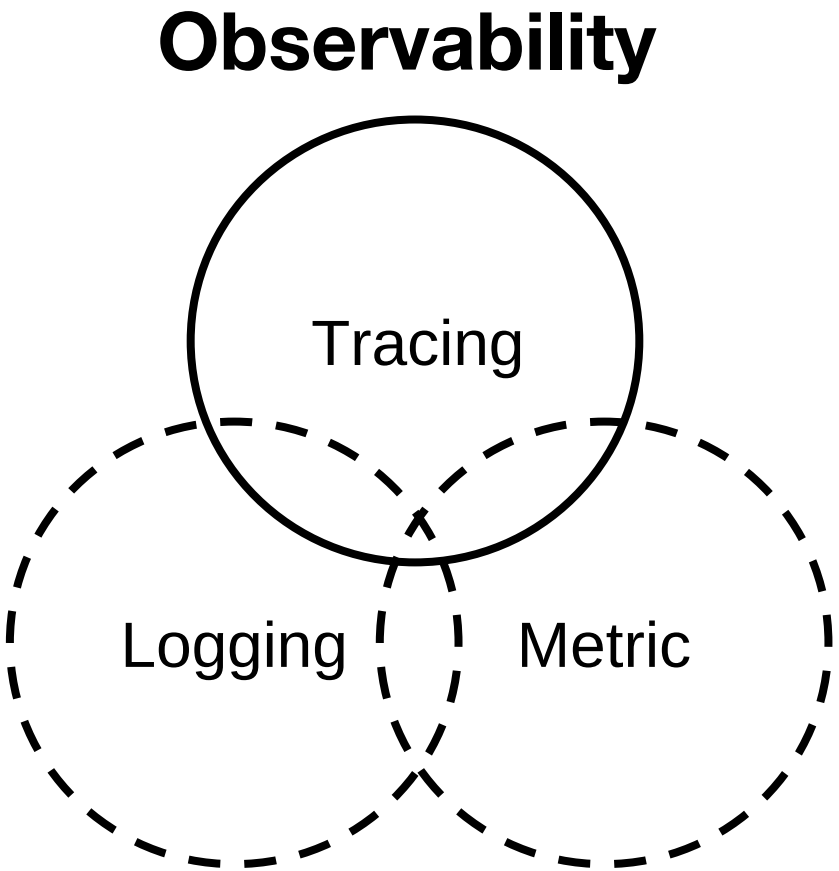
NELO2

Observability



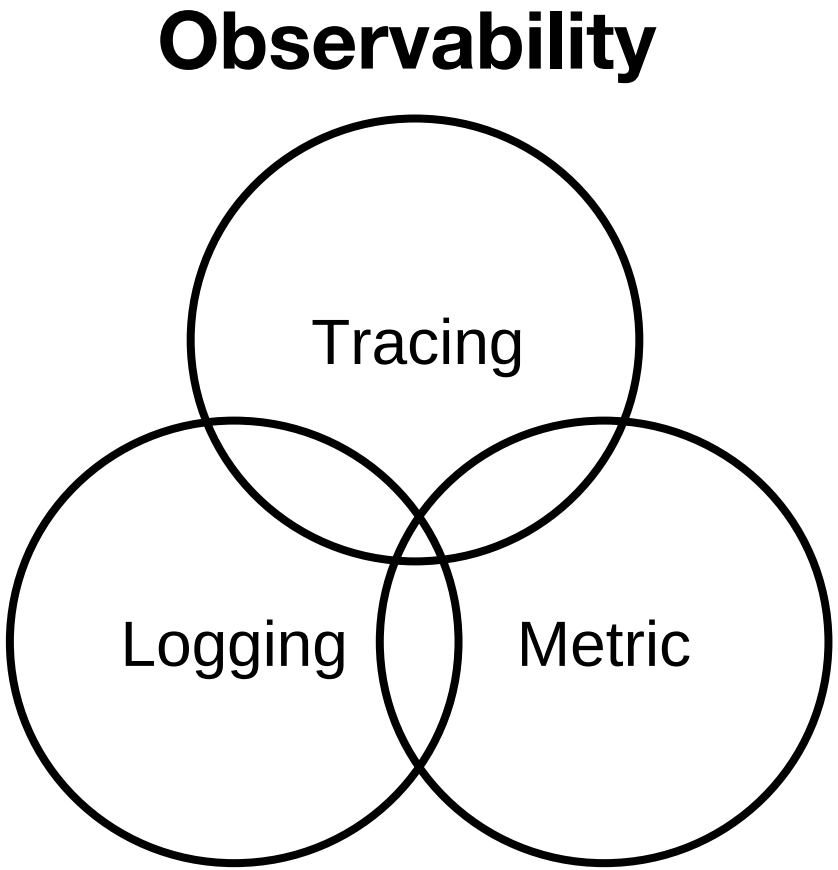
NPOT

Observability in Naver



PINPOINT

Observability in Naver

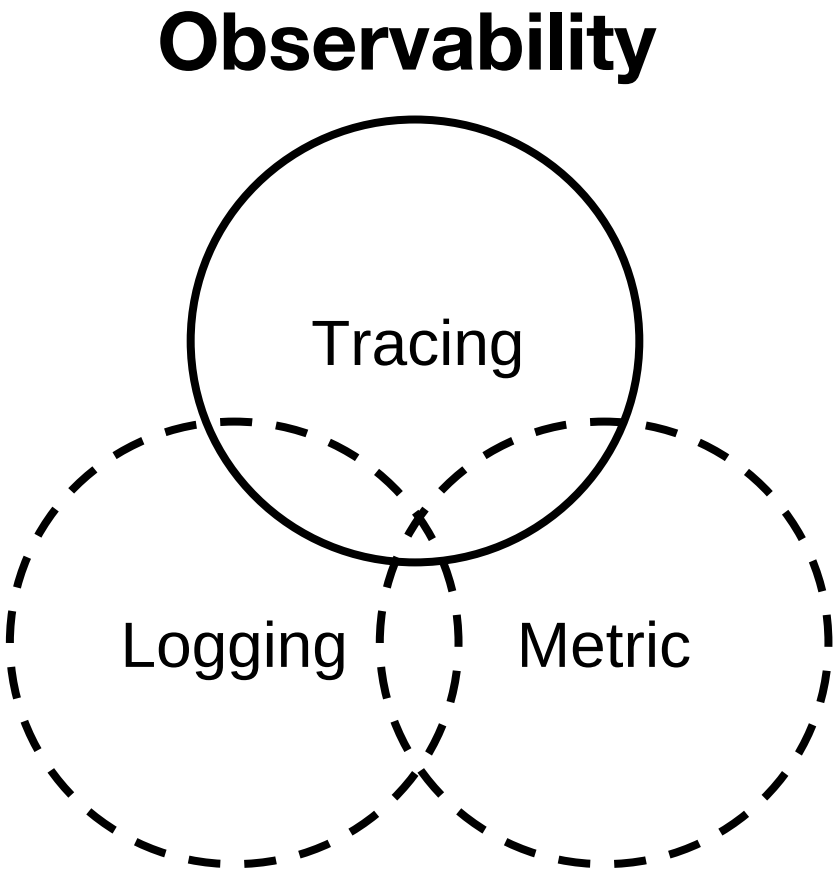


NELO2

PINPOINT

NPOT

Observability in Naver



PINPOINT

Enhance Observability

Bird Eye View

Finding Slow Transactions

Distributed Tracing

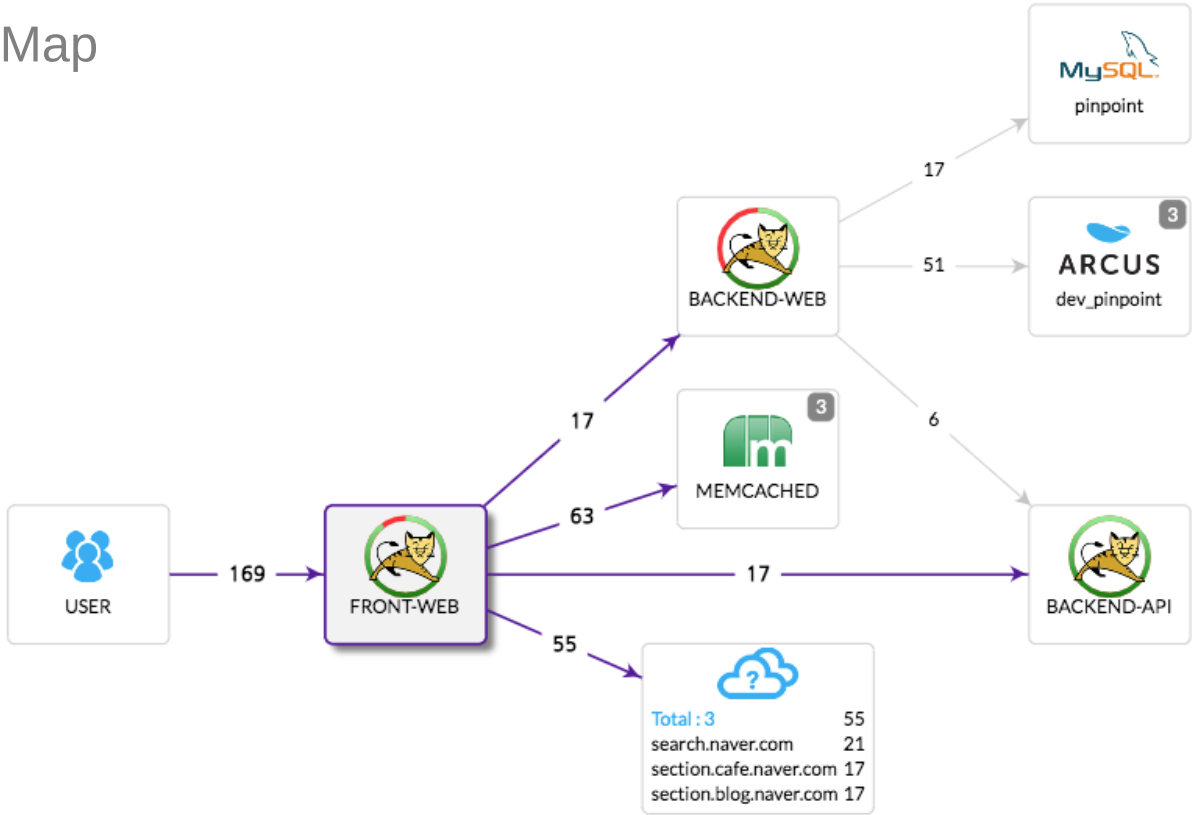
DevOps

Scalable

Minimum Overload

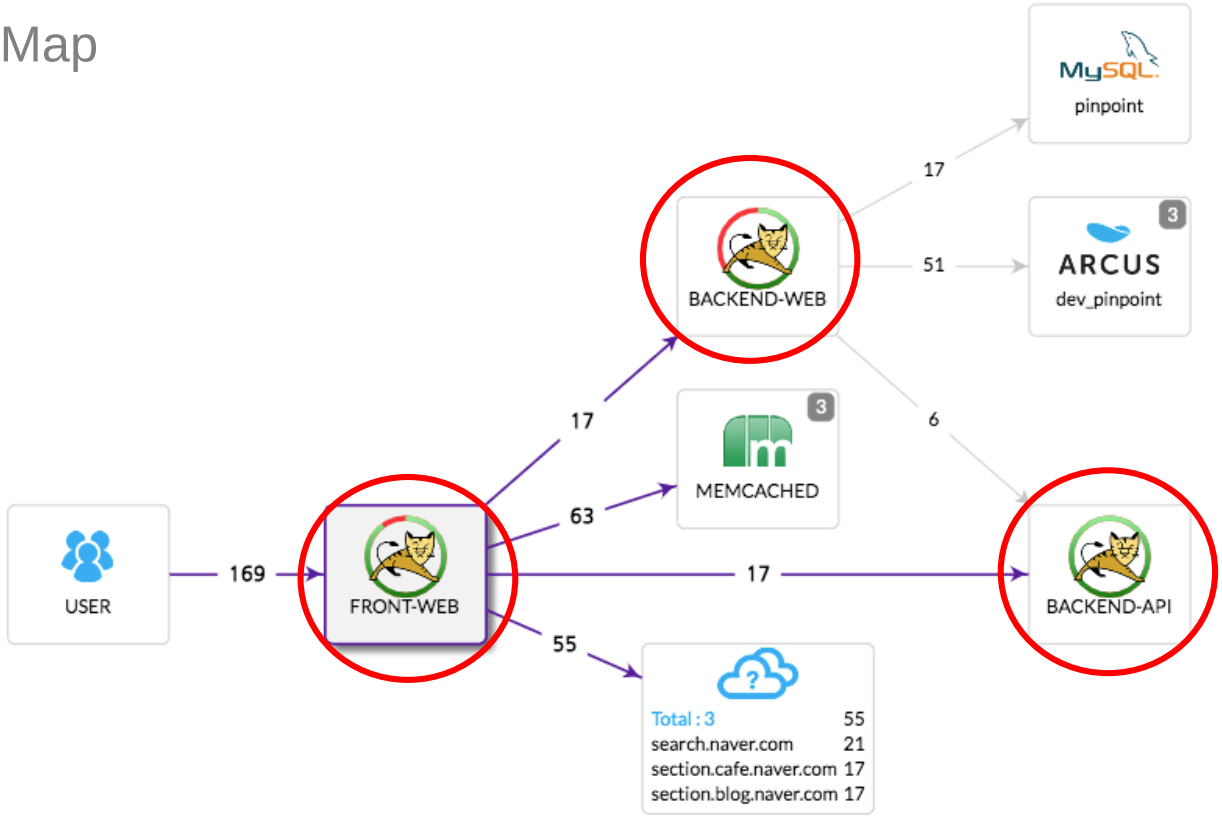
Bird Eye View

- Server Map



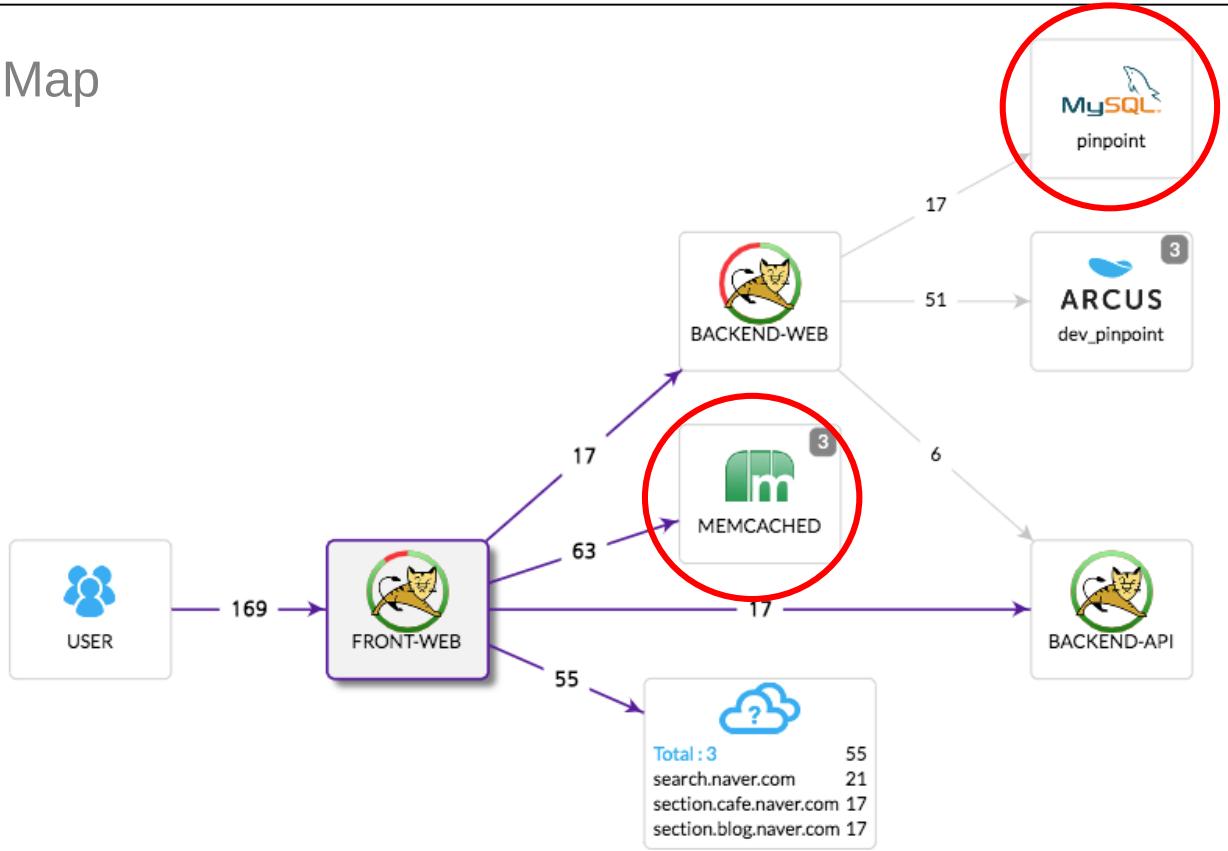
Bird Eye View

- Server Map



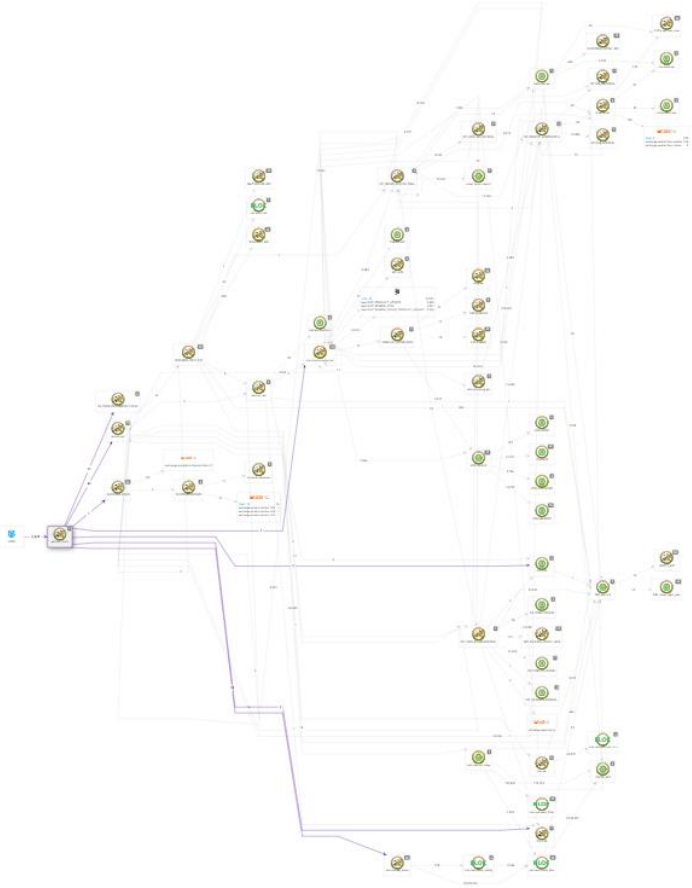
Bird Eye View

- Server Map



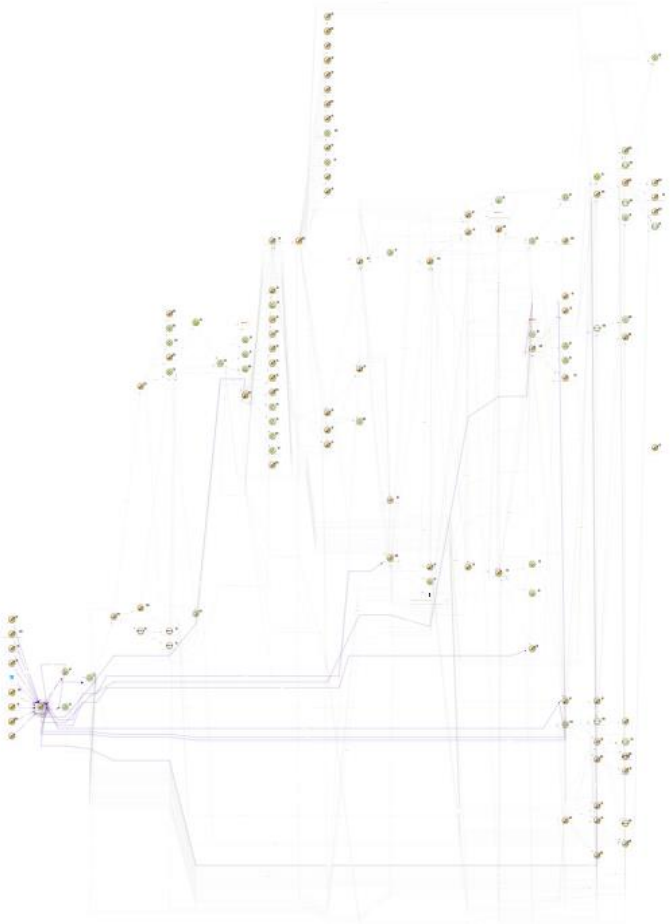
Bird Eye View

Bird Eye View



NAVER

Bird Eye View



NAVER

Enhance Observability

Bird Eye View

Finding Slow Transactions

Distributed Tracing

DevOps

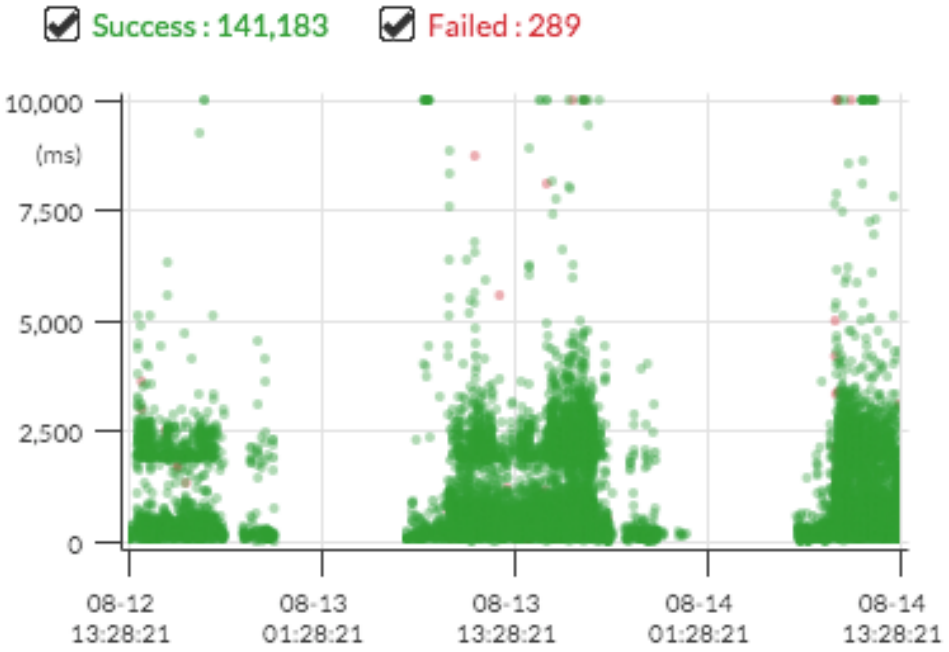
Scalable

Minimum Overload

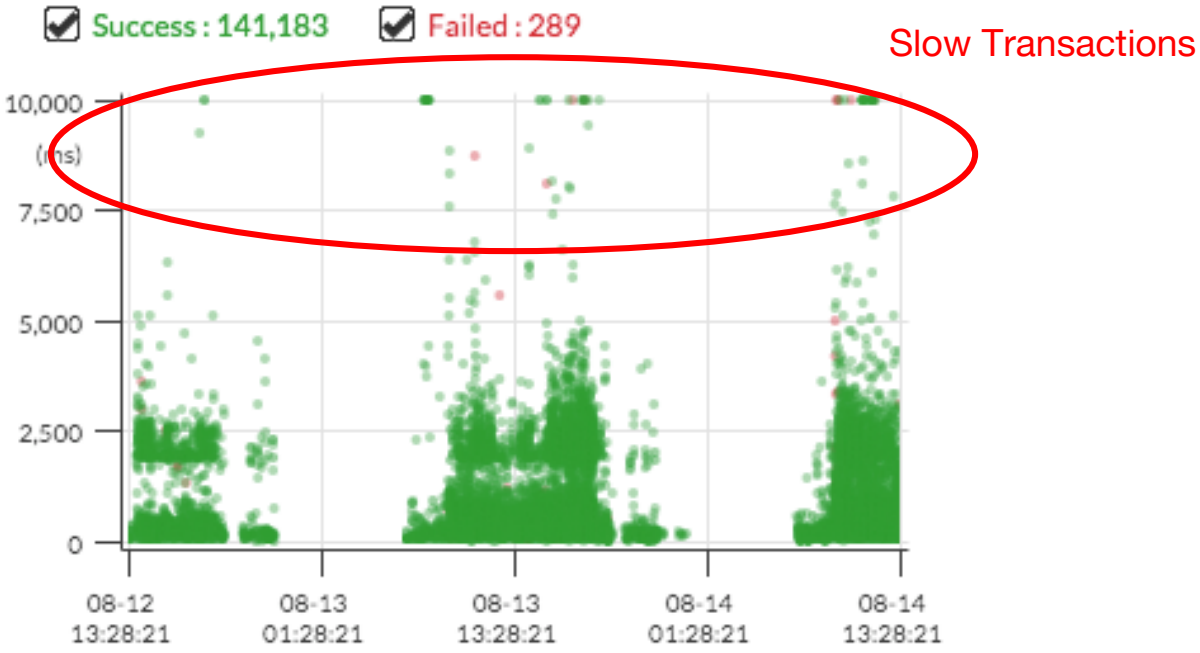
Finding Slow Transaction

- Scatter Chart

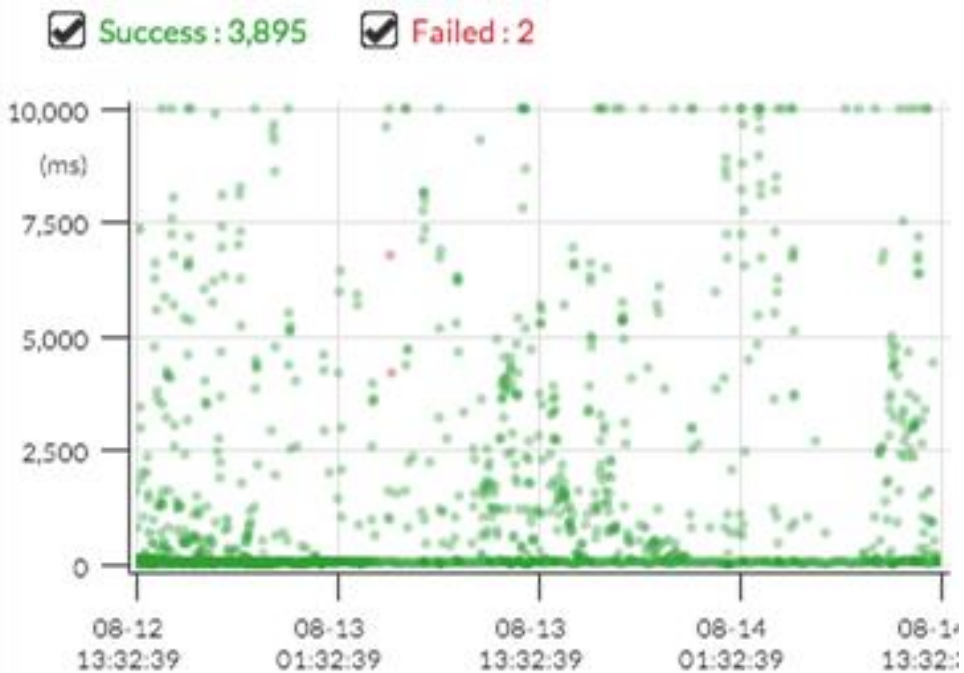
Finding Slow Transaction



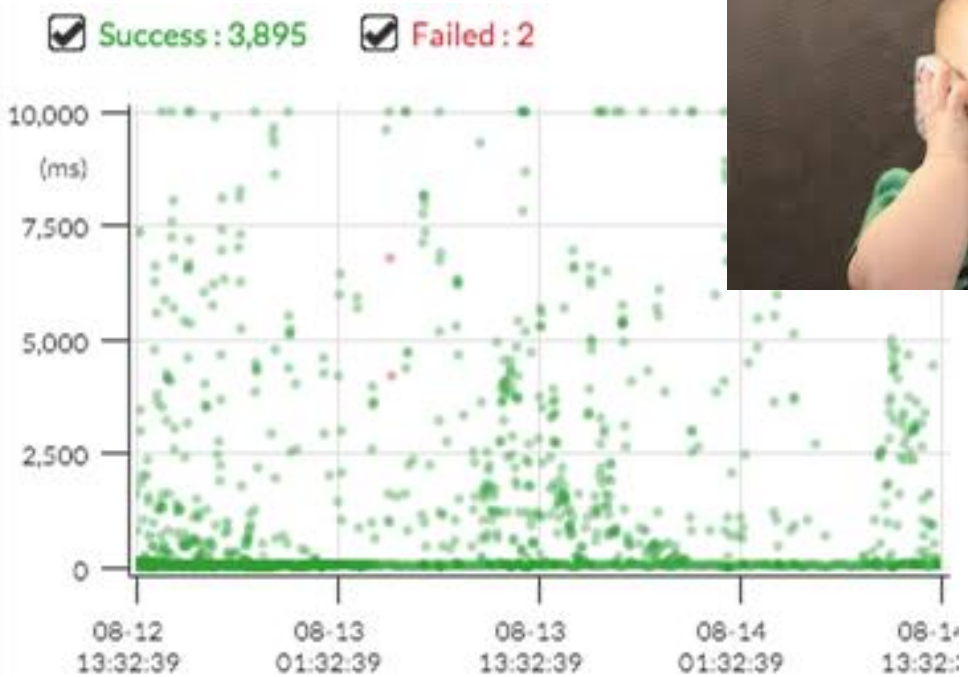
Finding Slow Transaction



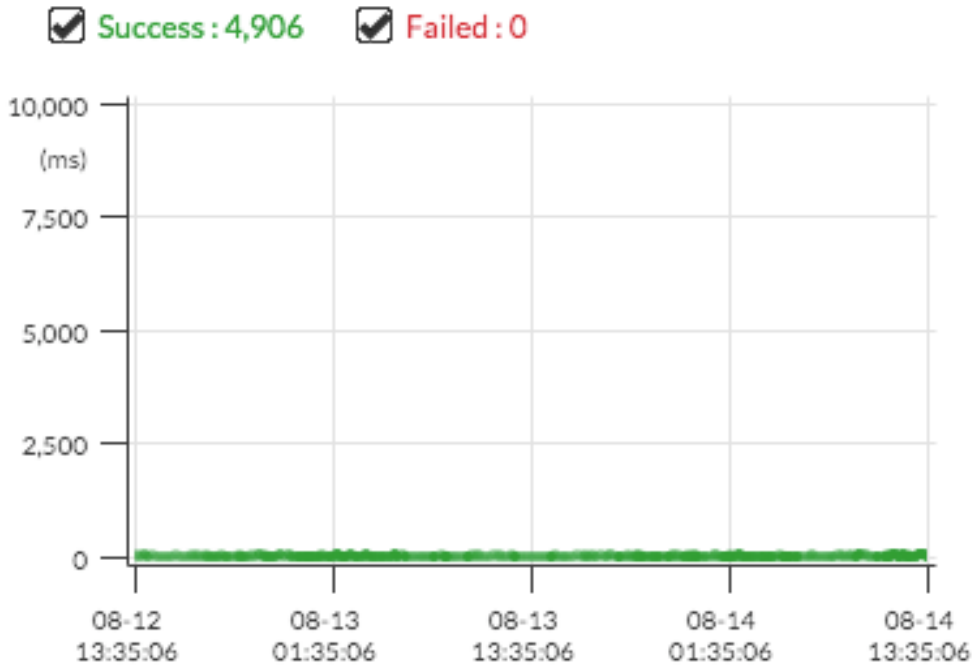
Finding Slow Transaction



Finding Slow Transaction

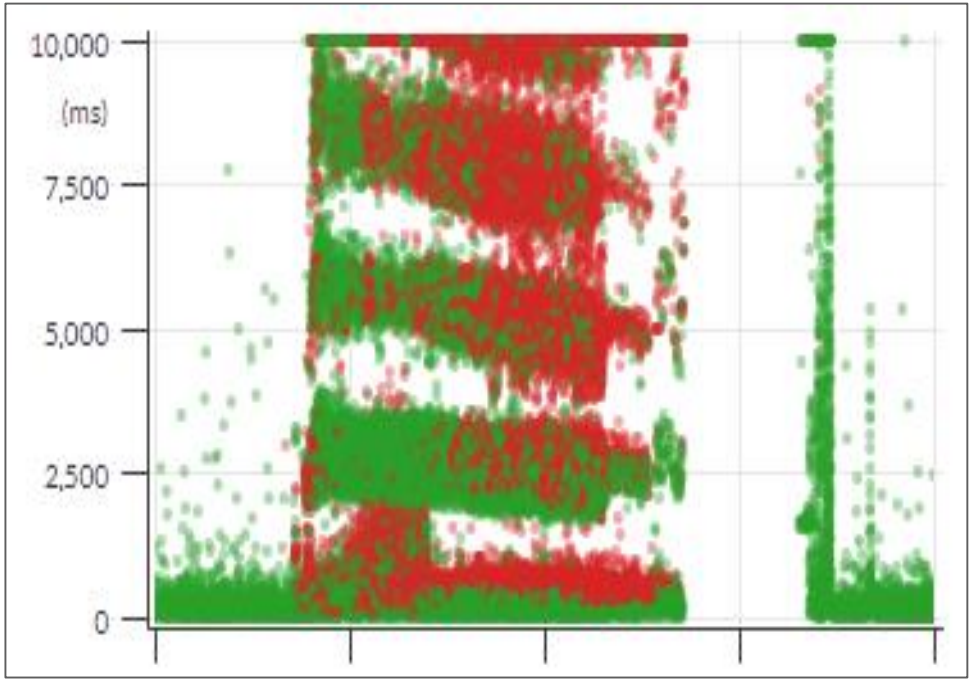


Finding Slow Transaction



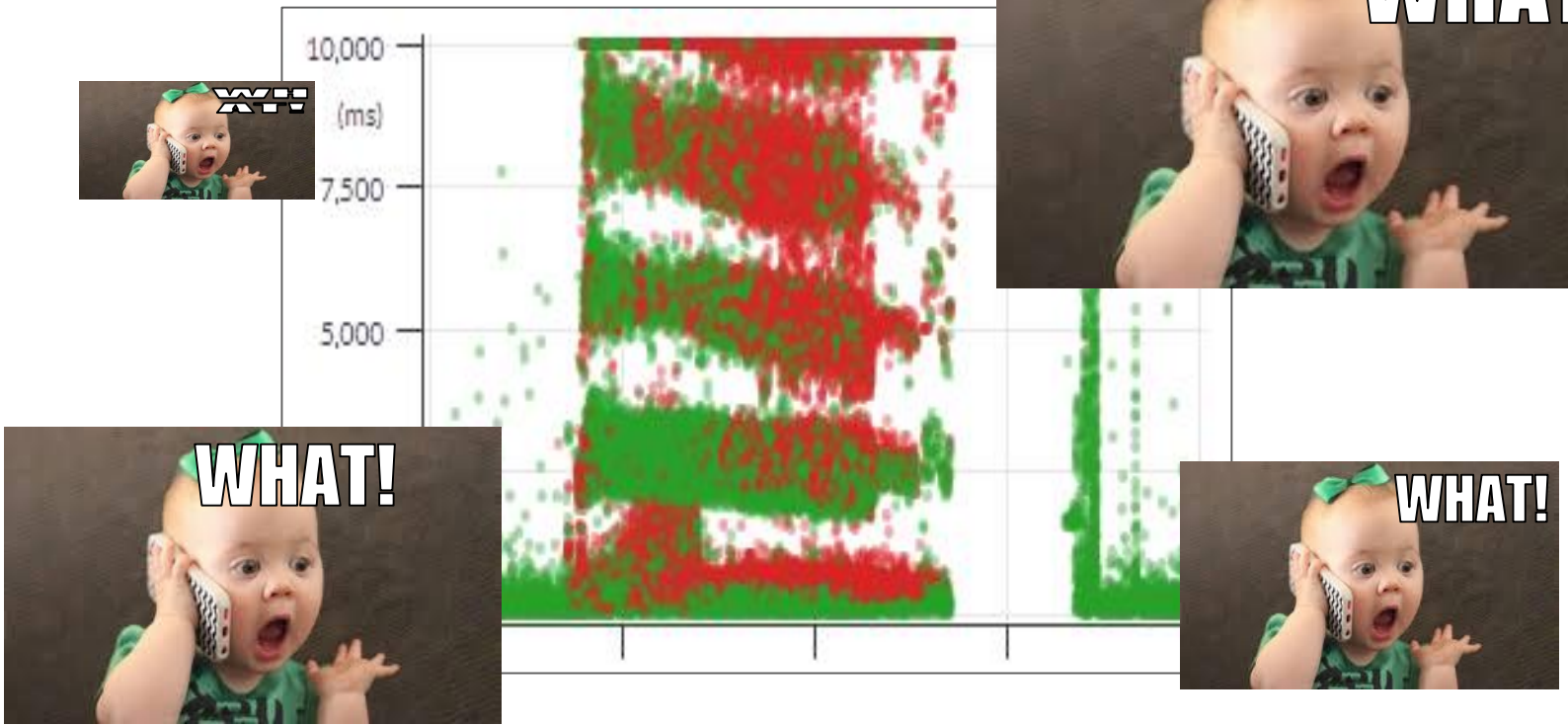
Finding Slow Transaction

- Malfunctioning in Real Server



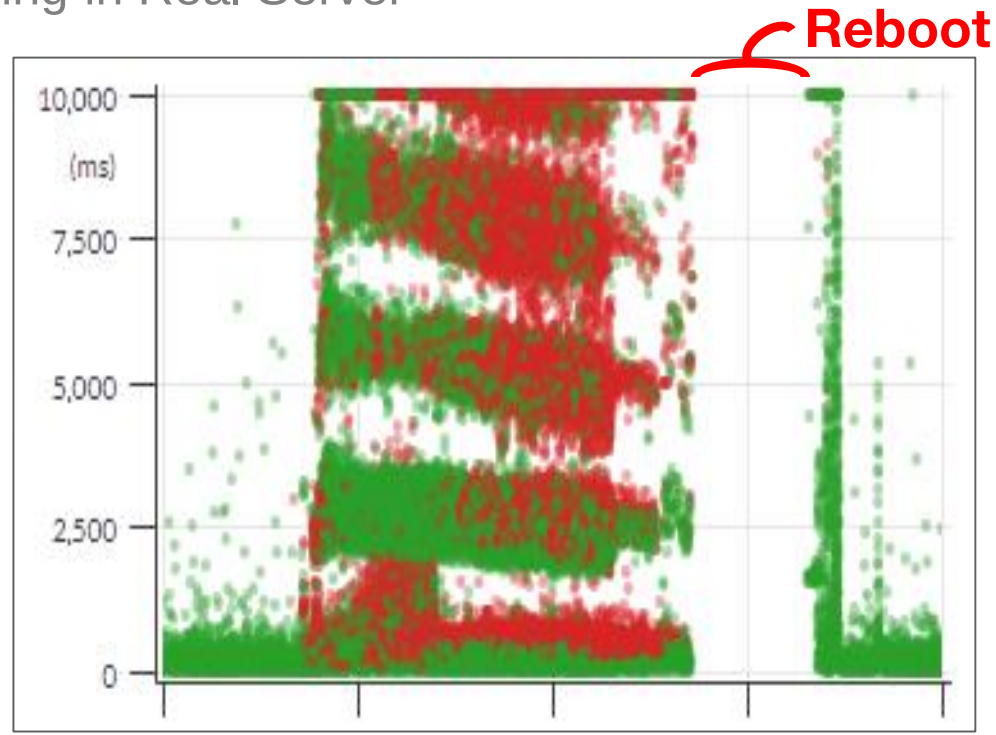
Finding Slow Transaction

- Malfunctioning in Real Server



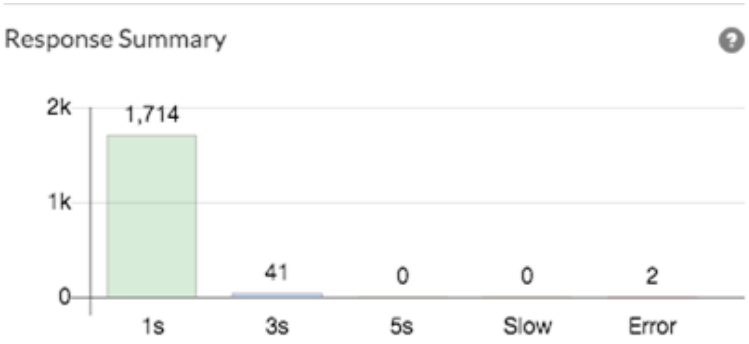
Finding Slow Transaction

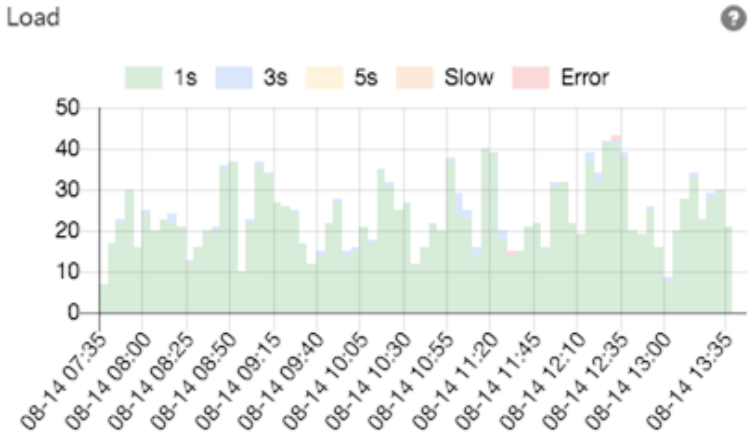
- Malfunctioning in Real Server



Finding Slow Transaction

- Response Summary chart
- Load chart





Enhance Observability

Bird Eye View

Finding Slow Transactions

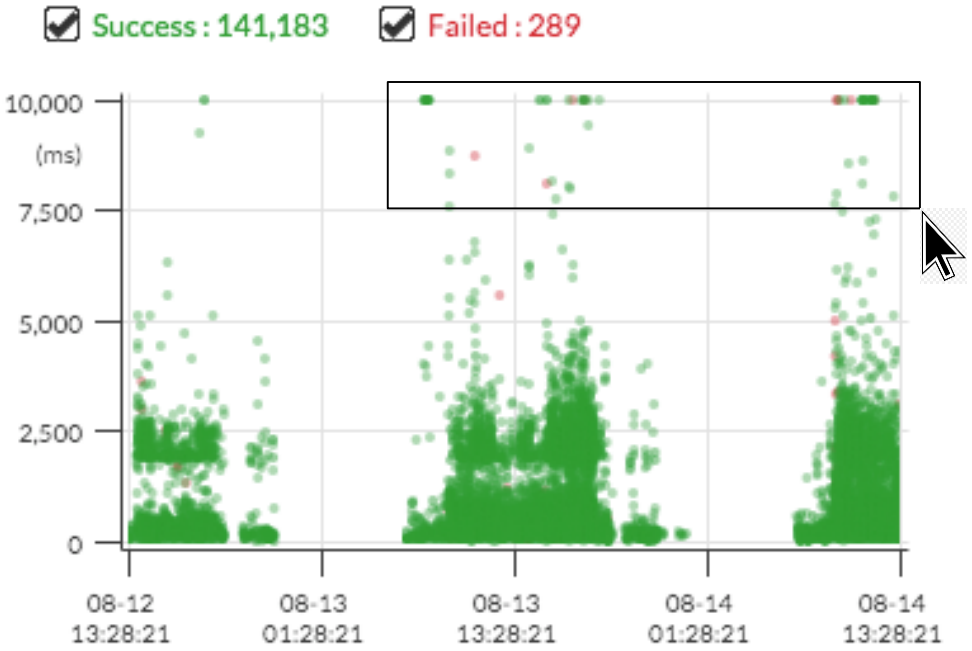
Distributed Tracing

DevOps

Scalable

Minimum Overload

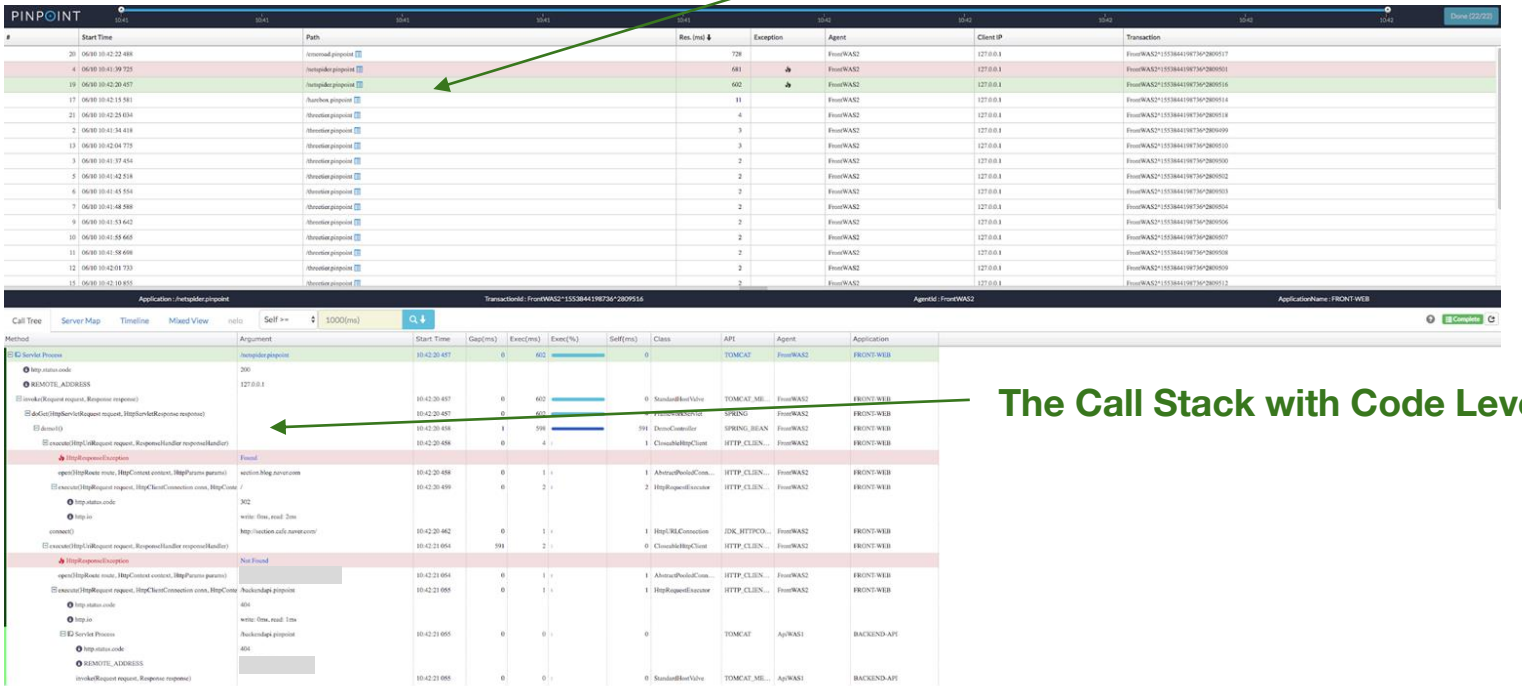
Finding Slow Transaction



Distributed Tracing

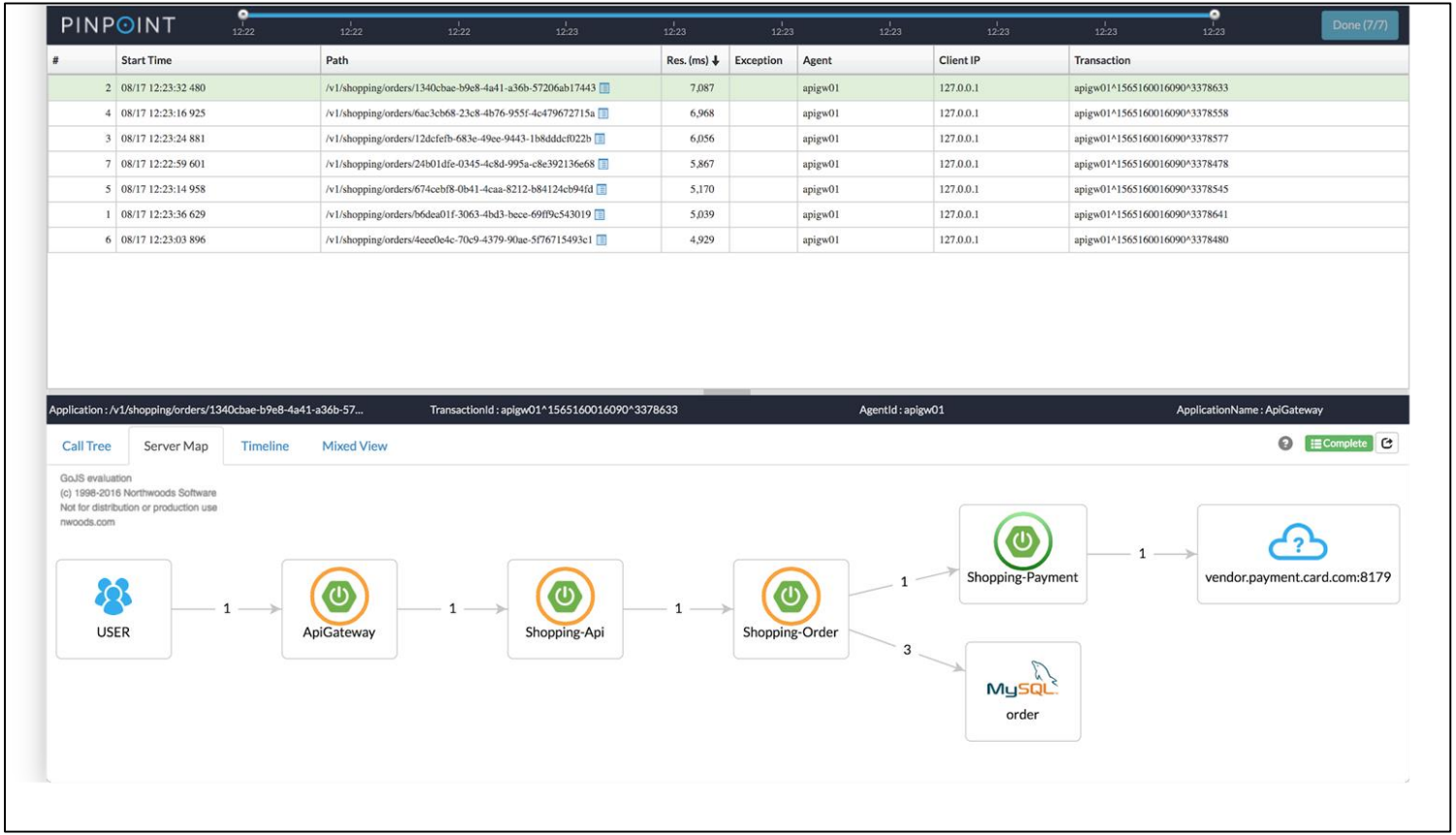
- Call Stack

List of Selected Transaction

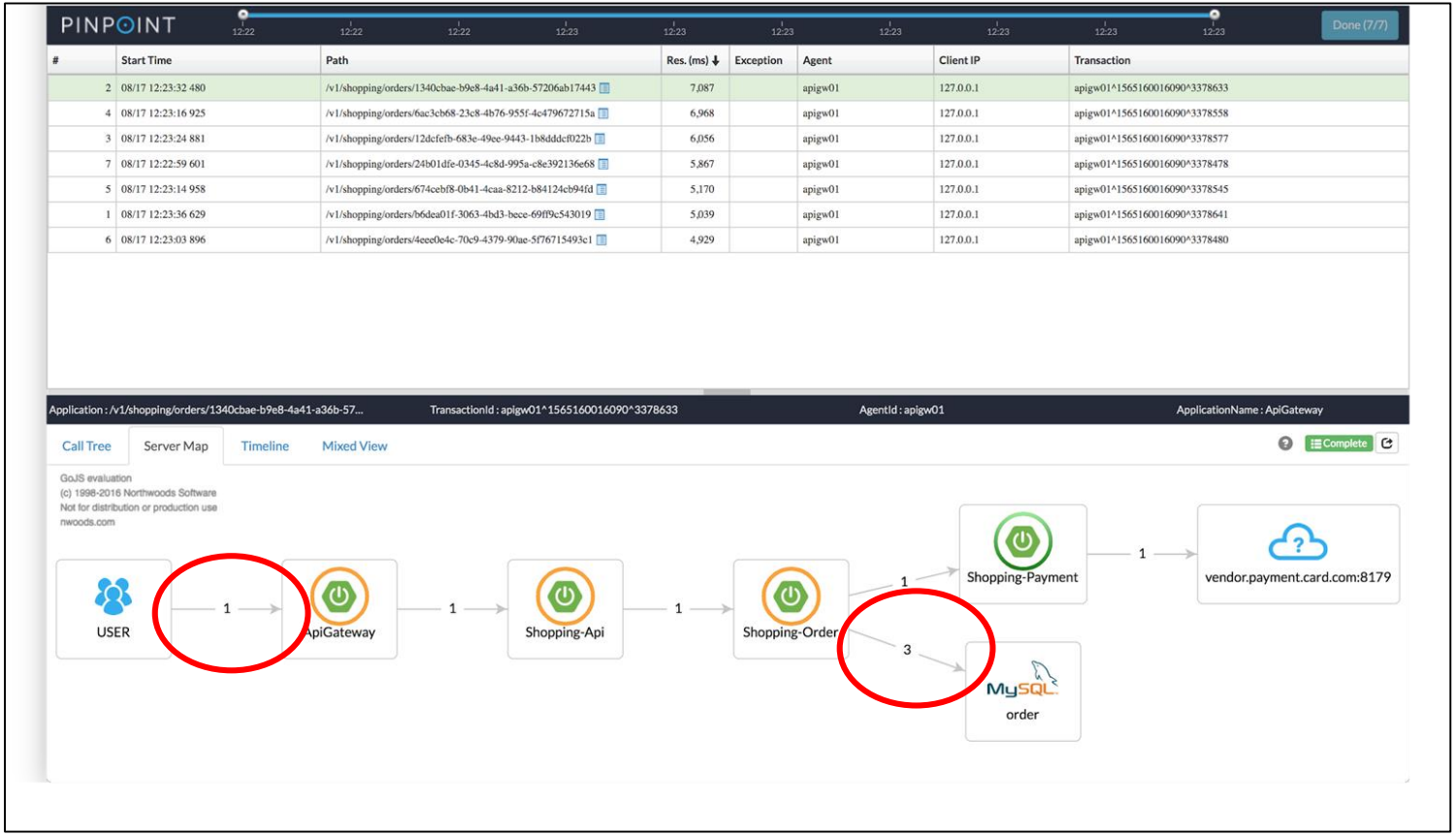


The Call Stack with Code Level

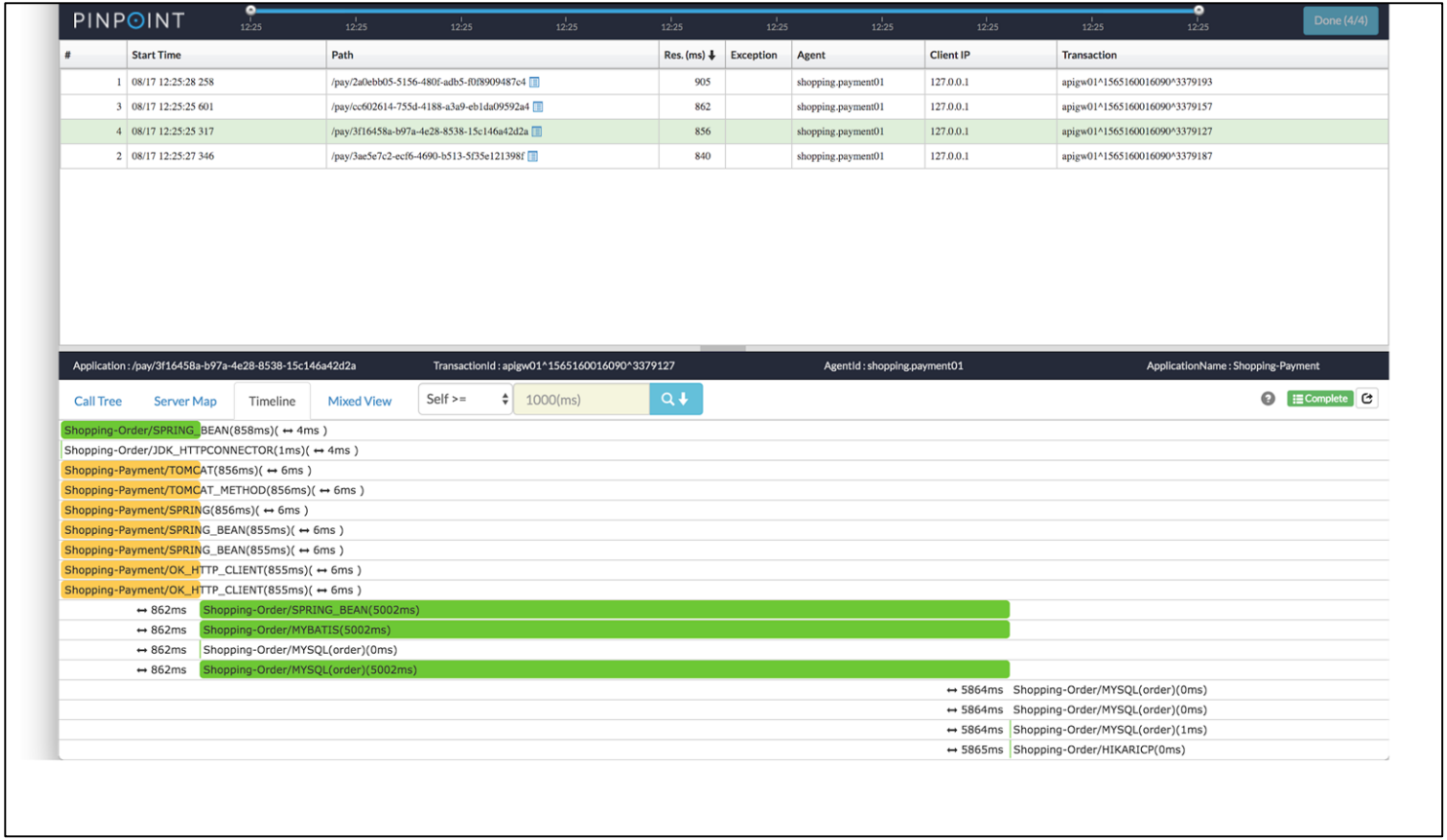
Distributed Tracing



Distributed Tracing



Distributed Tracing

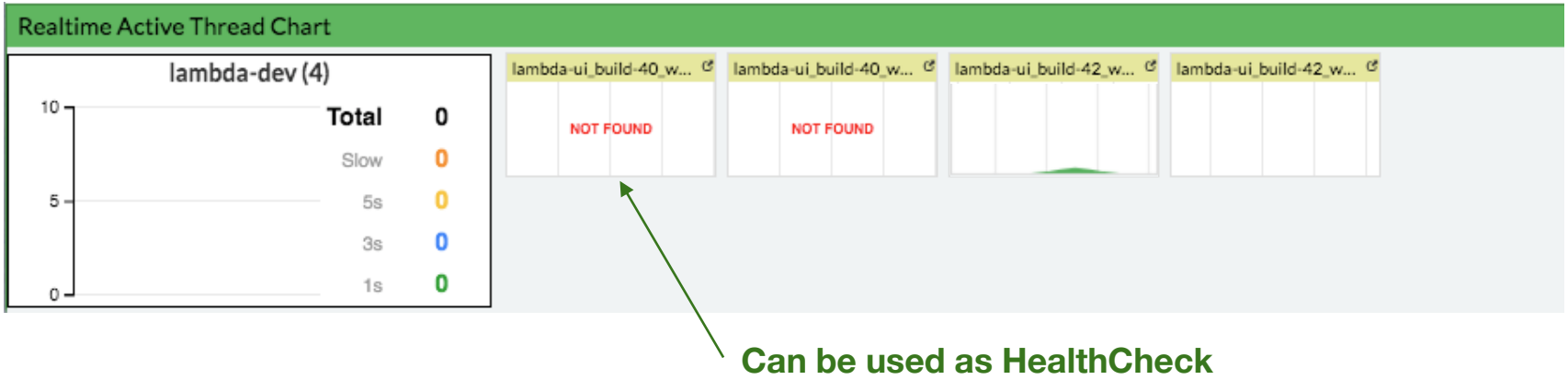


Enhance Observability

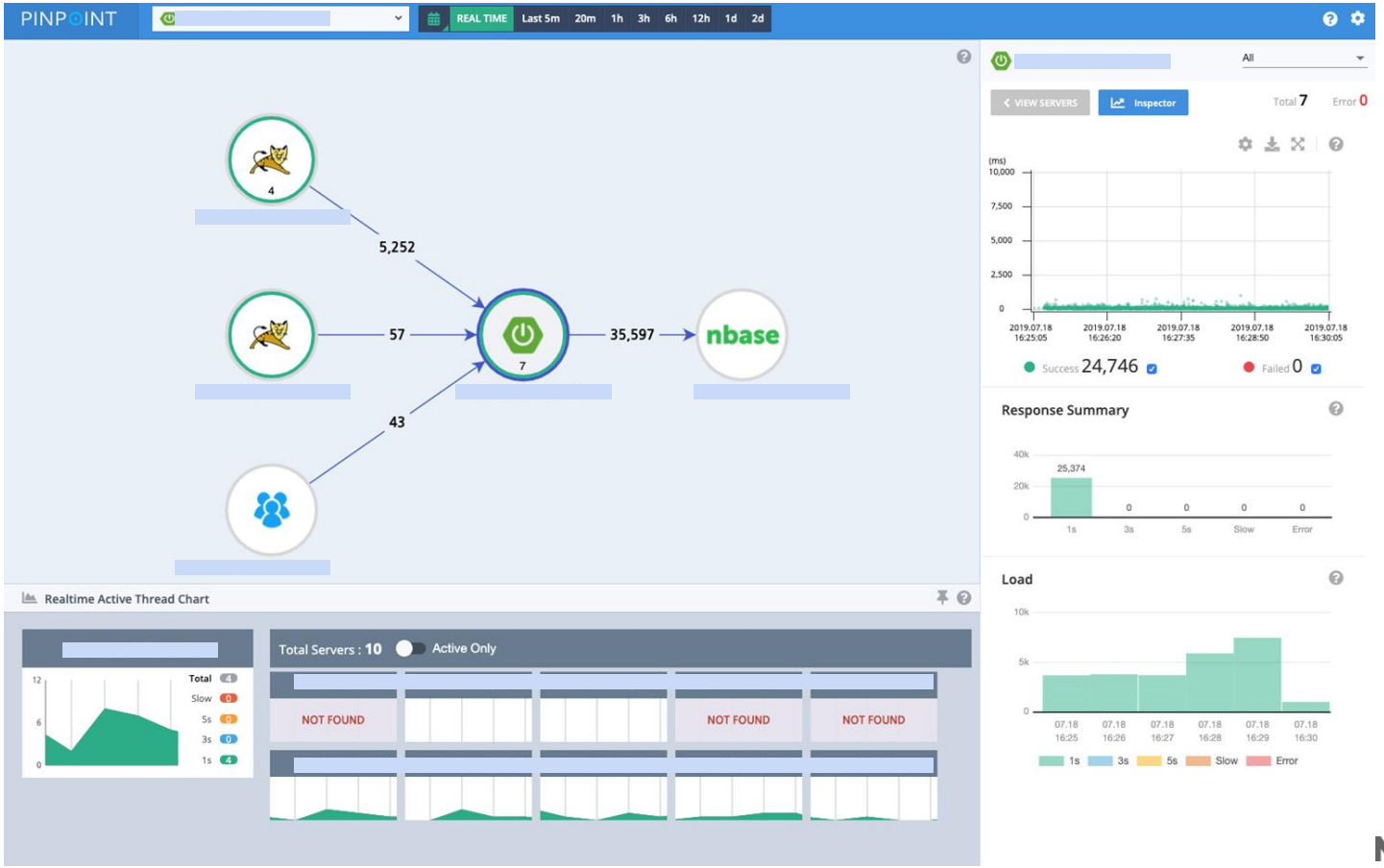
Bird Eye View
Finding Slow Transactions
Distributed Tracing
DevOps
Scalable
Minimum Overload

DevOps

- Realtime Active Thread Chart

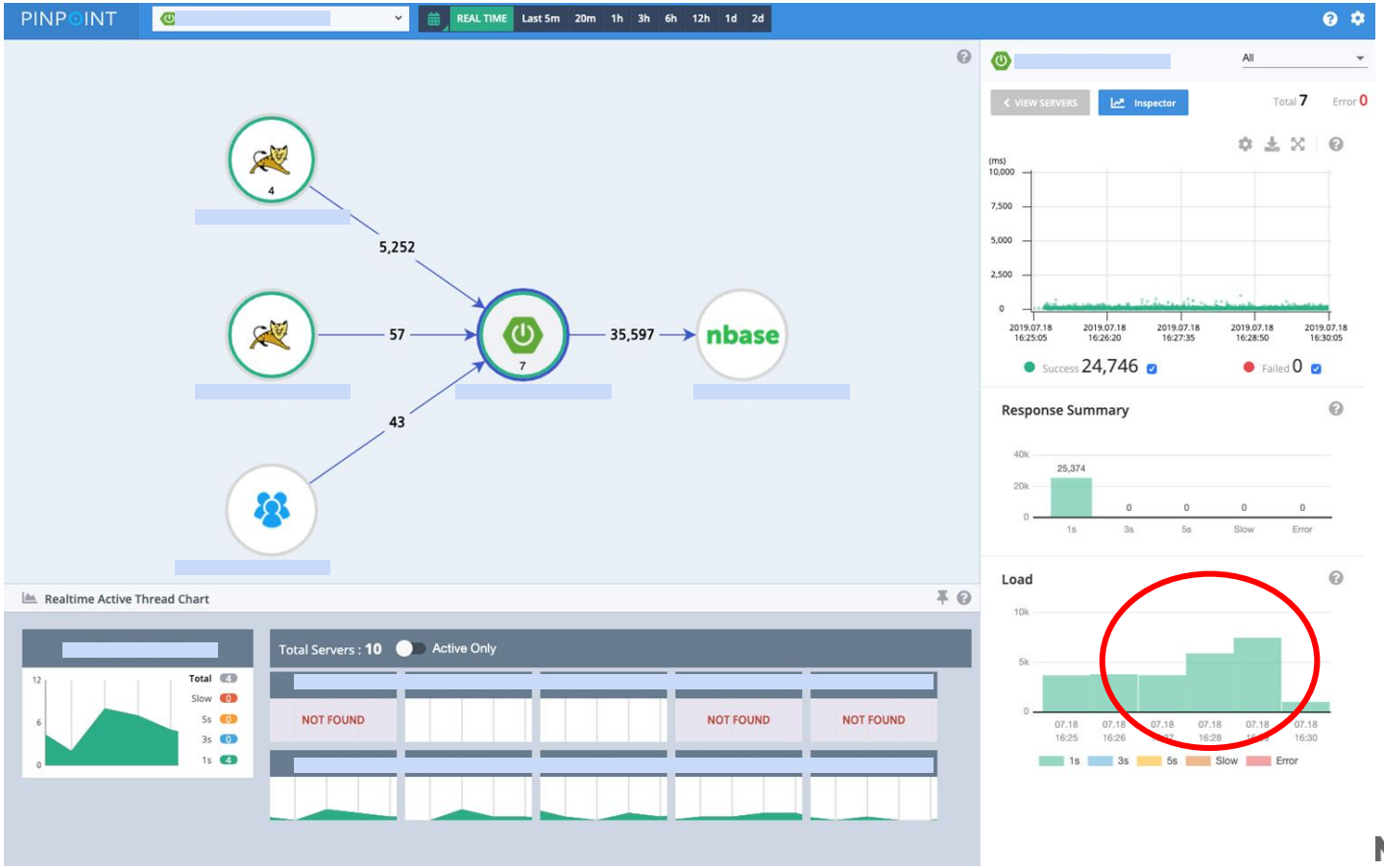


DevOps



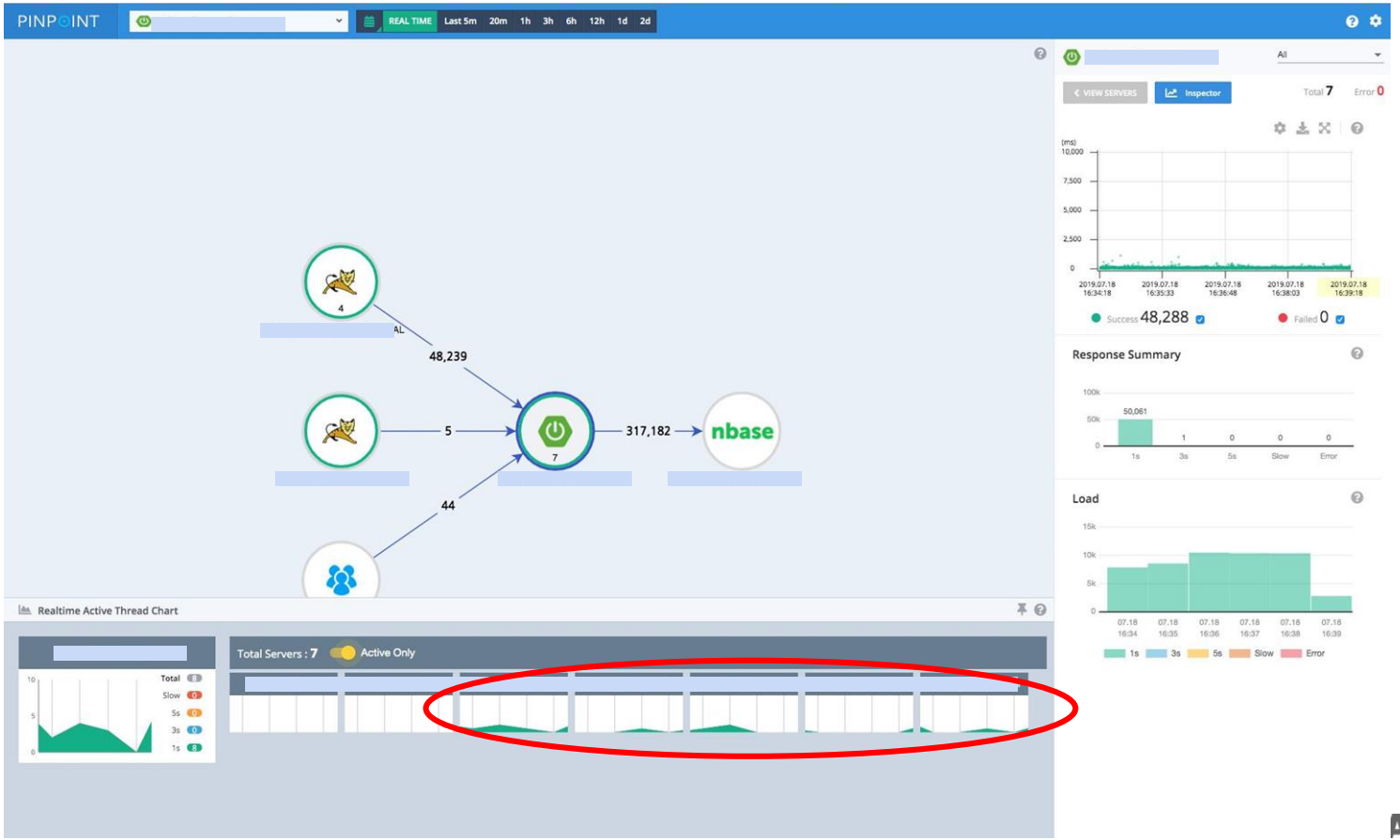
NAVER

DevOps

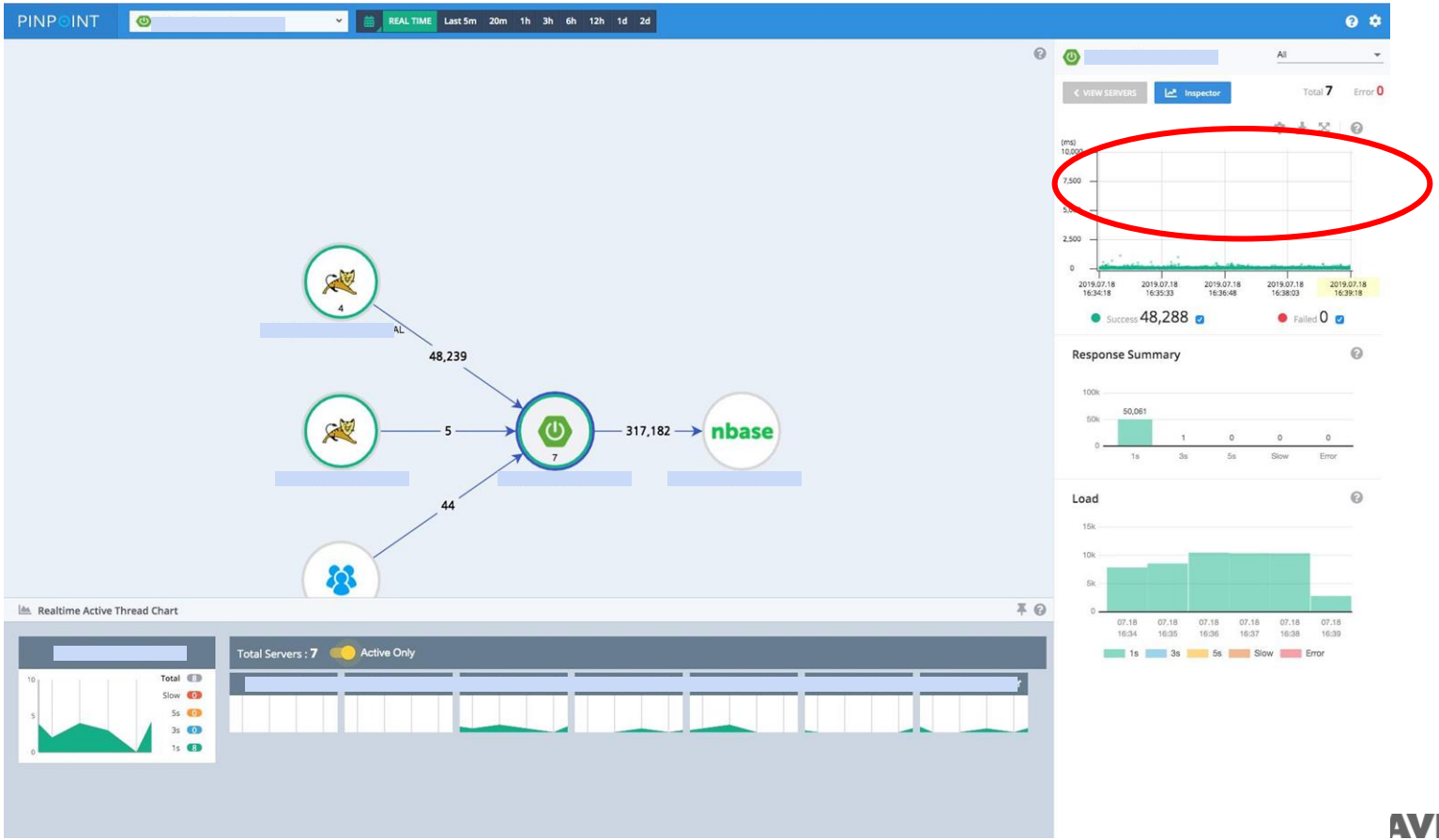


NAVER

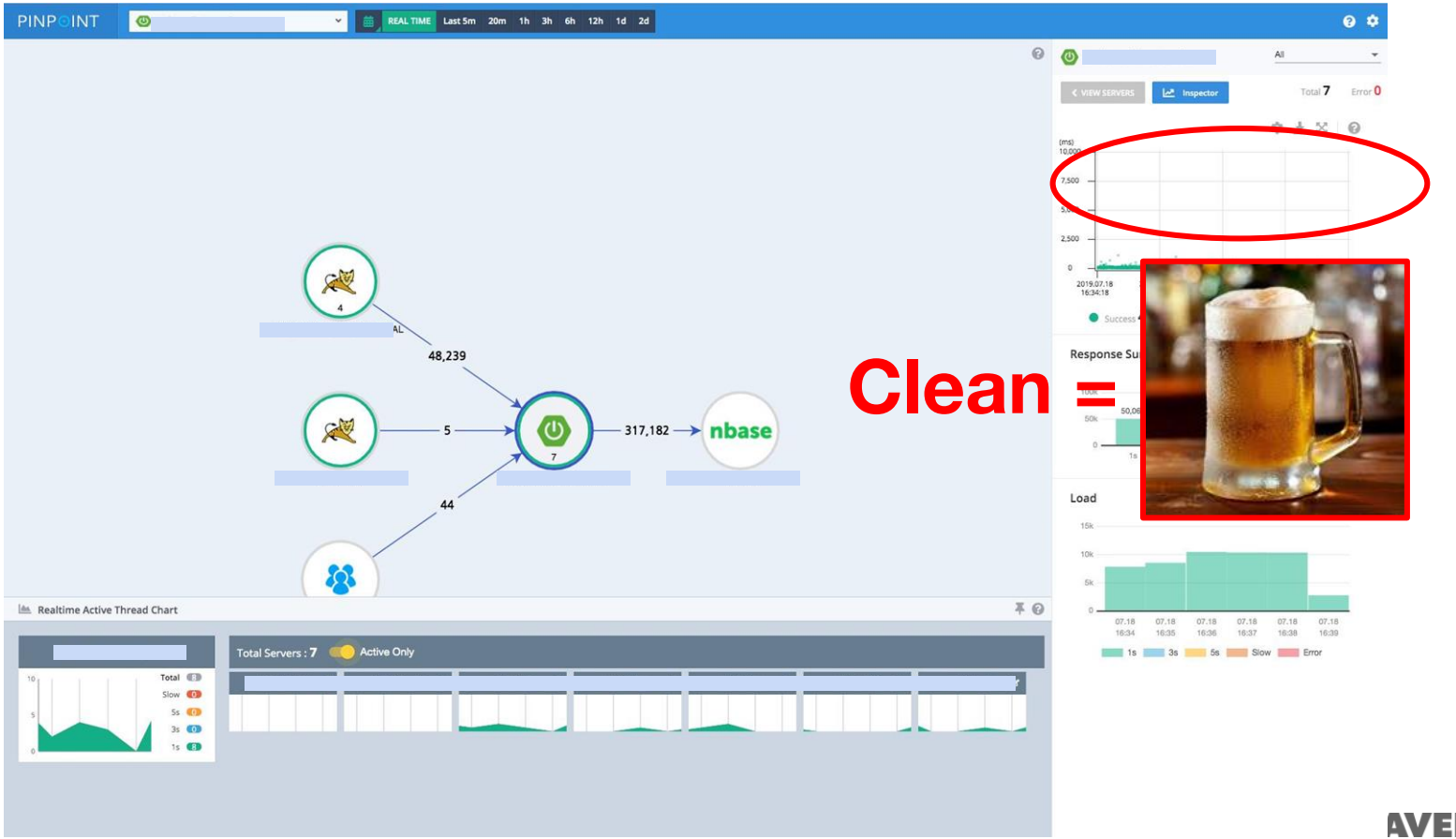
DevOps


PINPOINT
REAL TIME
Last 5m
20m
1h
3h
6h
12h
1d
2d
48,239
5
317,182
44
nbase
Realtime Active Thread Chart
Total Servers : 7
Active Only
Response Summary
Load
Success 48,288
Failed 0

DevOps


PINPOINT
REAL TIME
Last 5m
20m
1h
3h
6h
12h
1d
2d
48,239
5
317,182
nbase
44
Realtime Active Thread Chart
Total Servers : 7
Active Only
VIEW SERVERS
Inspector
Total 7
Error 0
Success 48,288
Failed 0
Response Summary
Load

DevOps


PINPOINT
REAL TIME
Last 5m
20m
1h
3h
6h
12h
1d
2d
48,239
5
317,182
nbase
44
Realtime Active Thread Chart
Total Servers : 7
Active Only
Clean =
Load

DevOps

- Inspector

- Heap, Non-Heap Memory
- JVM/SYSTEM CPU
- JVM GC
- TPS, Active Thread
- Response Time
- File Descriptor
- Direct/Mapped Buffer
- Data Source

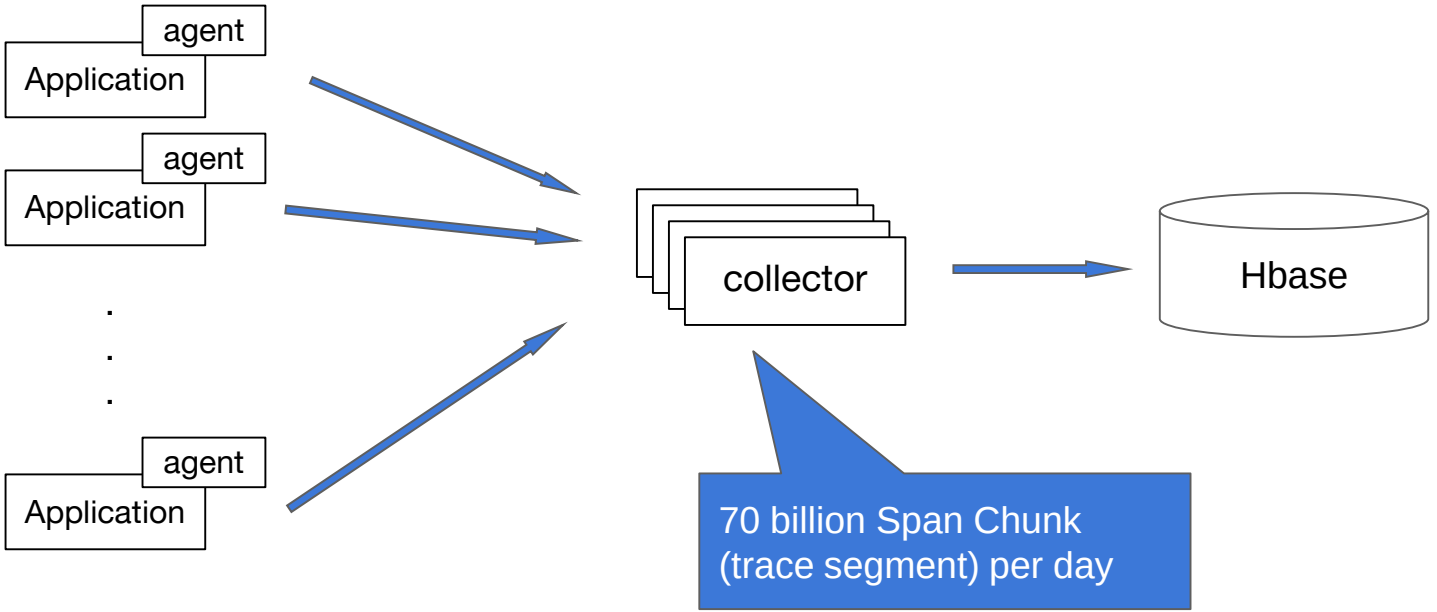
NAVER

Enhance Observability

Bird Eye View
Finding Slow Transactions
Distributed Tracing
DevOps
Scalable
Minimum Overload

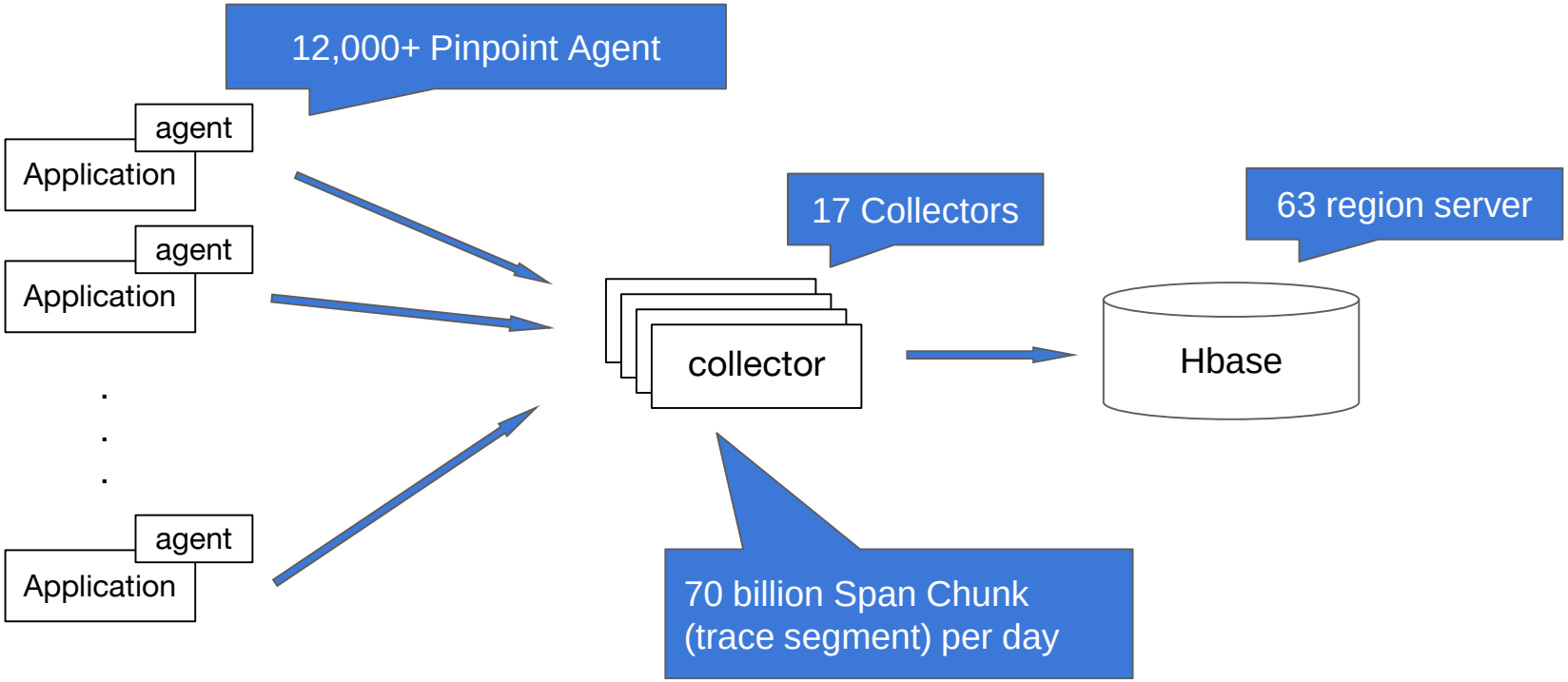
Scalable

- Fully functioning in Naver



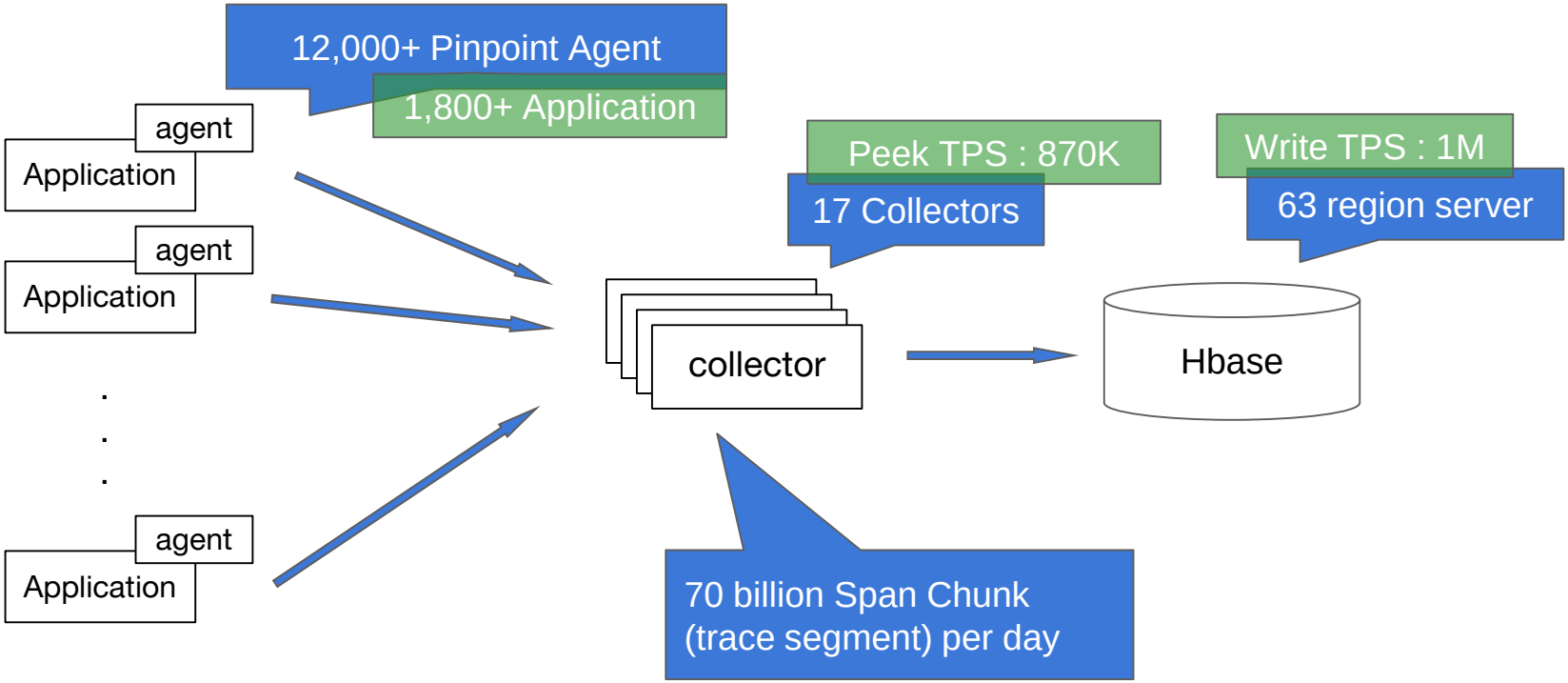
Scalable

- Fully functioning in Naver



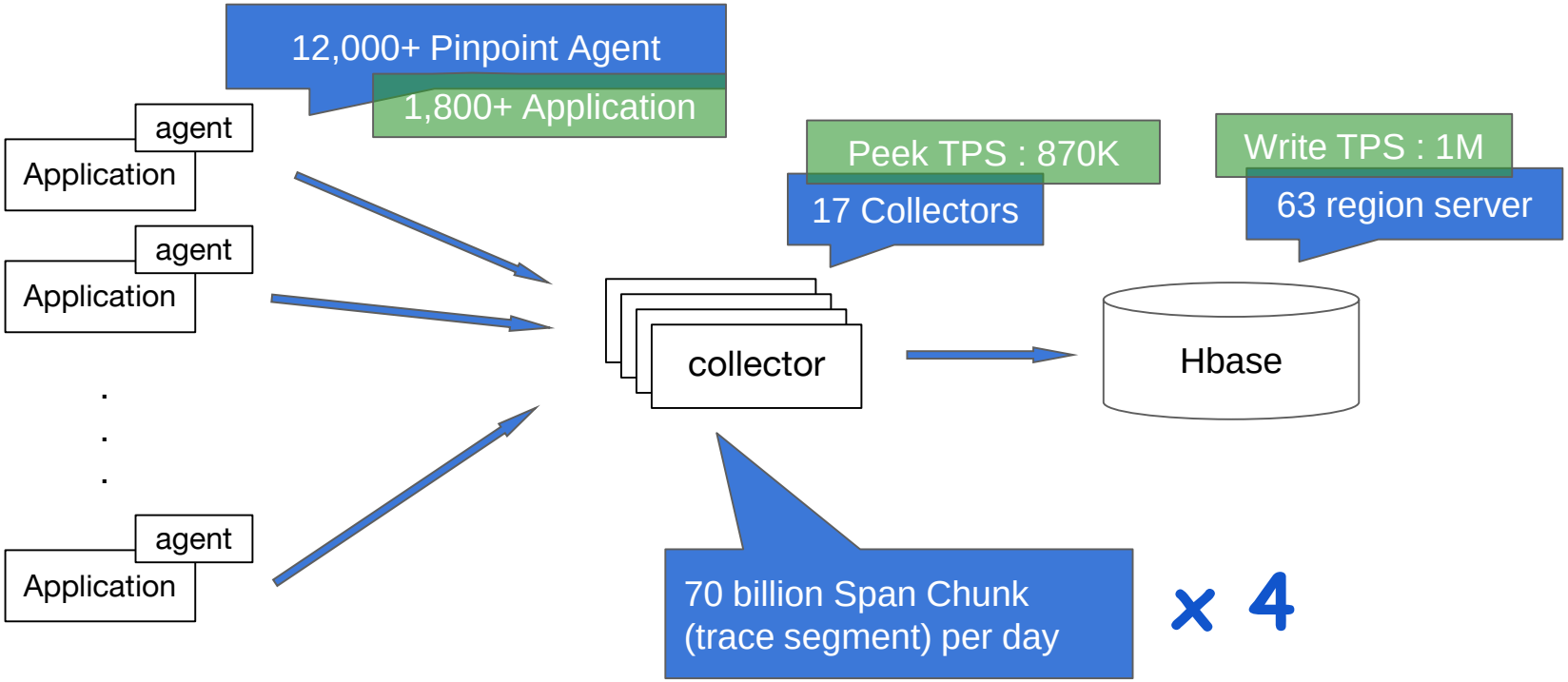
Scalable

- Fully functioning in Naver



Scalable

- Fully functioning in Naver



Enhance Observability

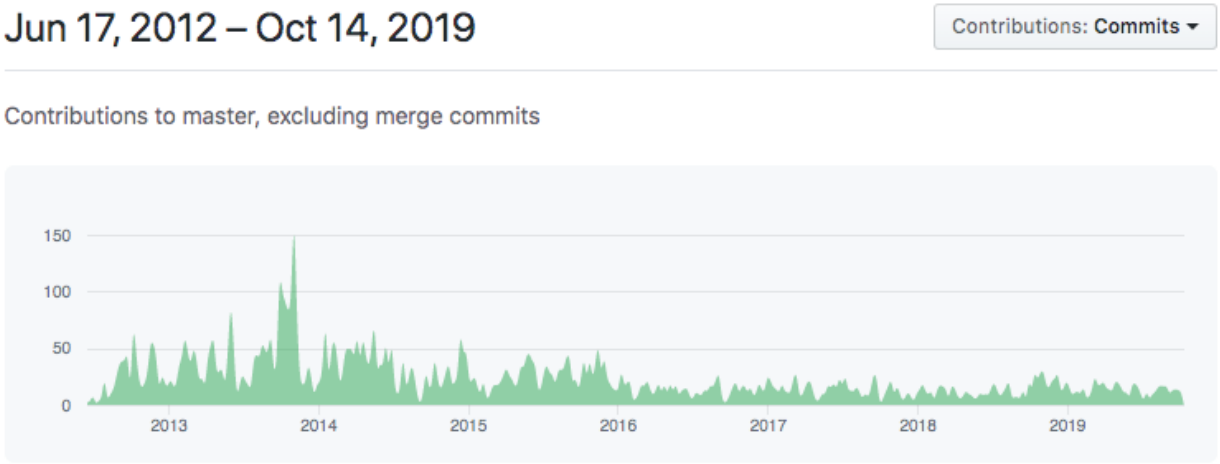
Bird Eye View
Finding Slow Transactions
Distributed Tracing
DevOps
Scalable
Minimum Overload

Minimum Overload

- Execute Integration Tests Periodically
  - **Less than 3%** difference in performance
  - Sampling ‘No-Agent’, ‘5%’, ‘100%’

Tomcat Only																					
VUser 4						VUser 32						VUser 64						VUser 128					
Duration (ms)	Tests	Error	Mean Test Time (ms)	TPS	nGrinder	Duration (ms)	Tests	Error	Mean Test Time (ms)	TPS	nGrinder	Duration (ms)	Tests	Error	Mean Test Time (ms)	TPS	nGrinder	Duration (ms)	Tests	Error	Mean Test Time (ms)	TPS	nGrinder
300000	135003	0	6.87	571.92	link	300000	421252	20	17.63	1784.54	link	300000	427943	11	34.78	1828.37	link	300000	427923	0	69.95	1828.08	link
300000	125282	0	7.42	530.74	link	300000	422774	23	17.51	1791.0	link	300000	415028	7	35.84	1773.2	link	300000	424081	0	70.61	1811.98	link
300000	127462	0	7.29	539.97	link	300000	427575	21	17.34	1811.3	link	300000	428504	8	34.75	1830.7	link	300000	403144	656	72.5	1722.34	link

Open Source Pinpoint



PINPOINT

2012.07 Development Start

2014.01 First use of Pinpoint

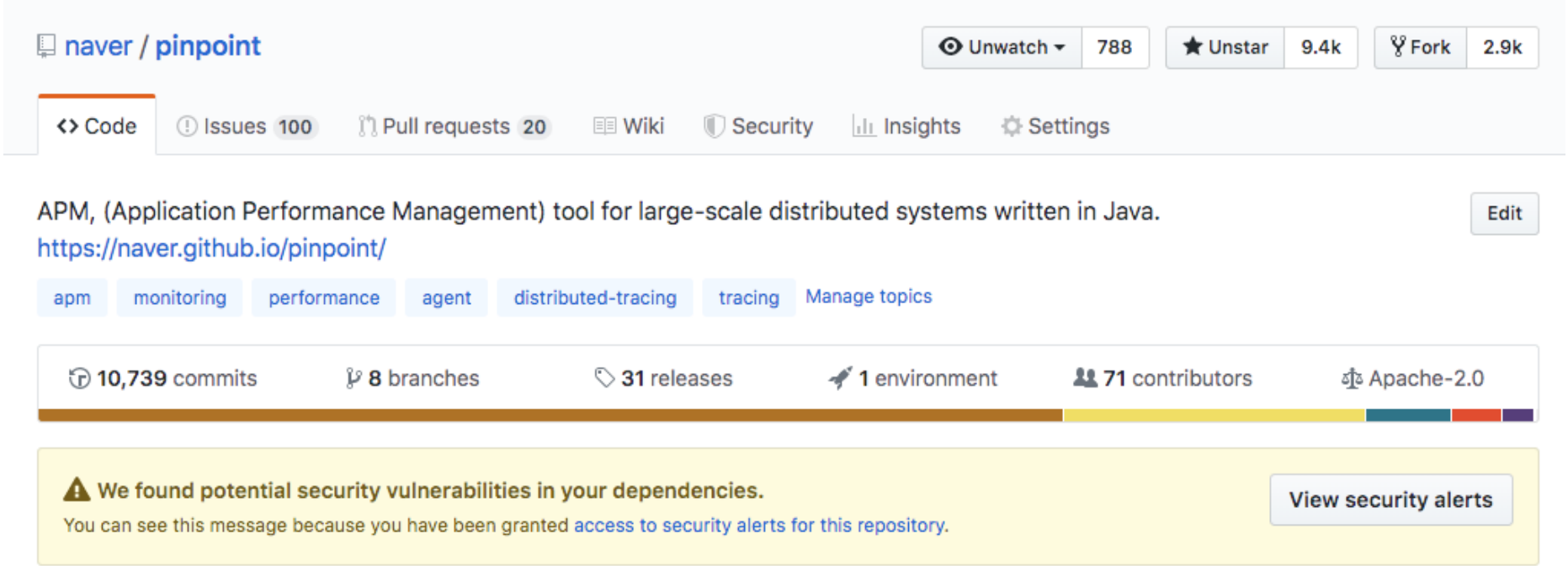
2014.12 90% teams switched from commercial APM

2015.01 Starting open source

2017.12 Reached 5000 stars

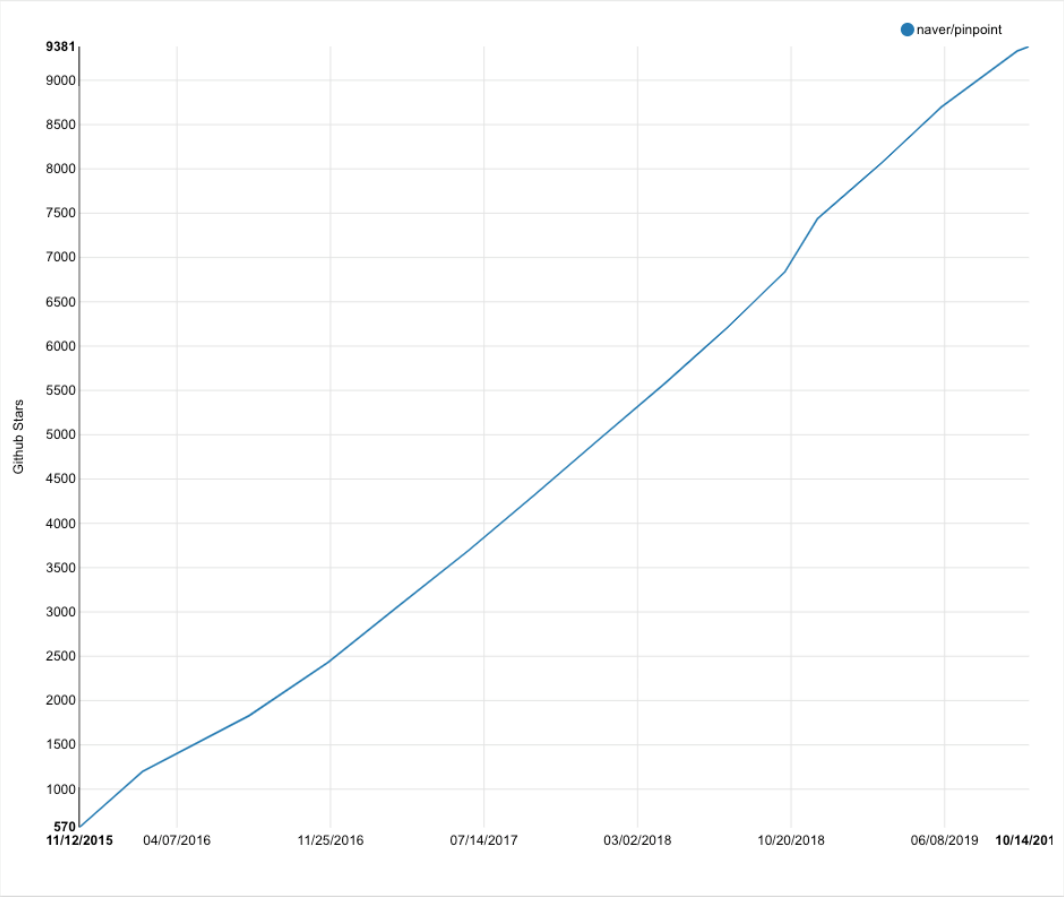
NAVER

Open Source Pinpoint



NAVER

Open Source Pinpoint



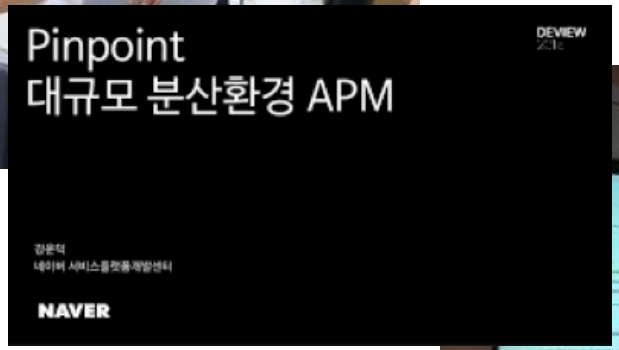
NAVER

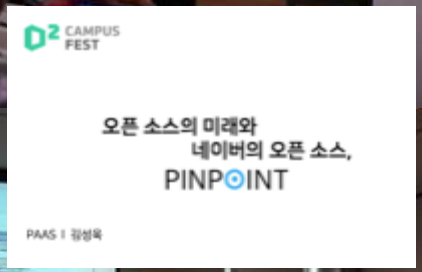
Open Source Pinpoint

- Lost of Top 10 IT companies in China
- Global enterprises in Korea
- Various IT companies in USA
- Companies in Financial Industry

NAVER

Open Source Community





NAVER

Open Source Community

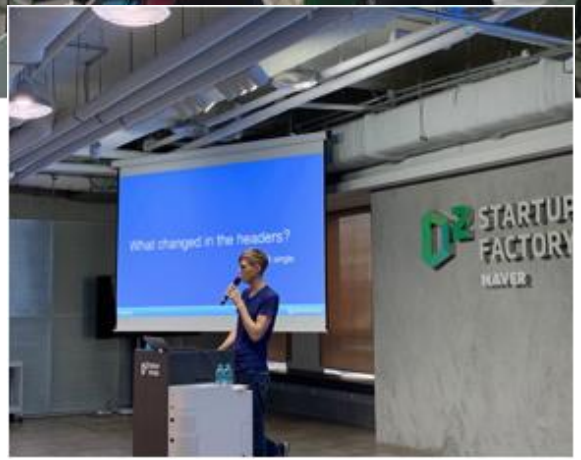
Open Source Community

Open Source Community

NAVER

Open Source Community

Open Source Community



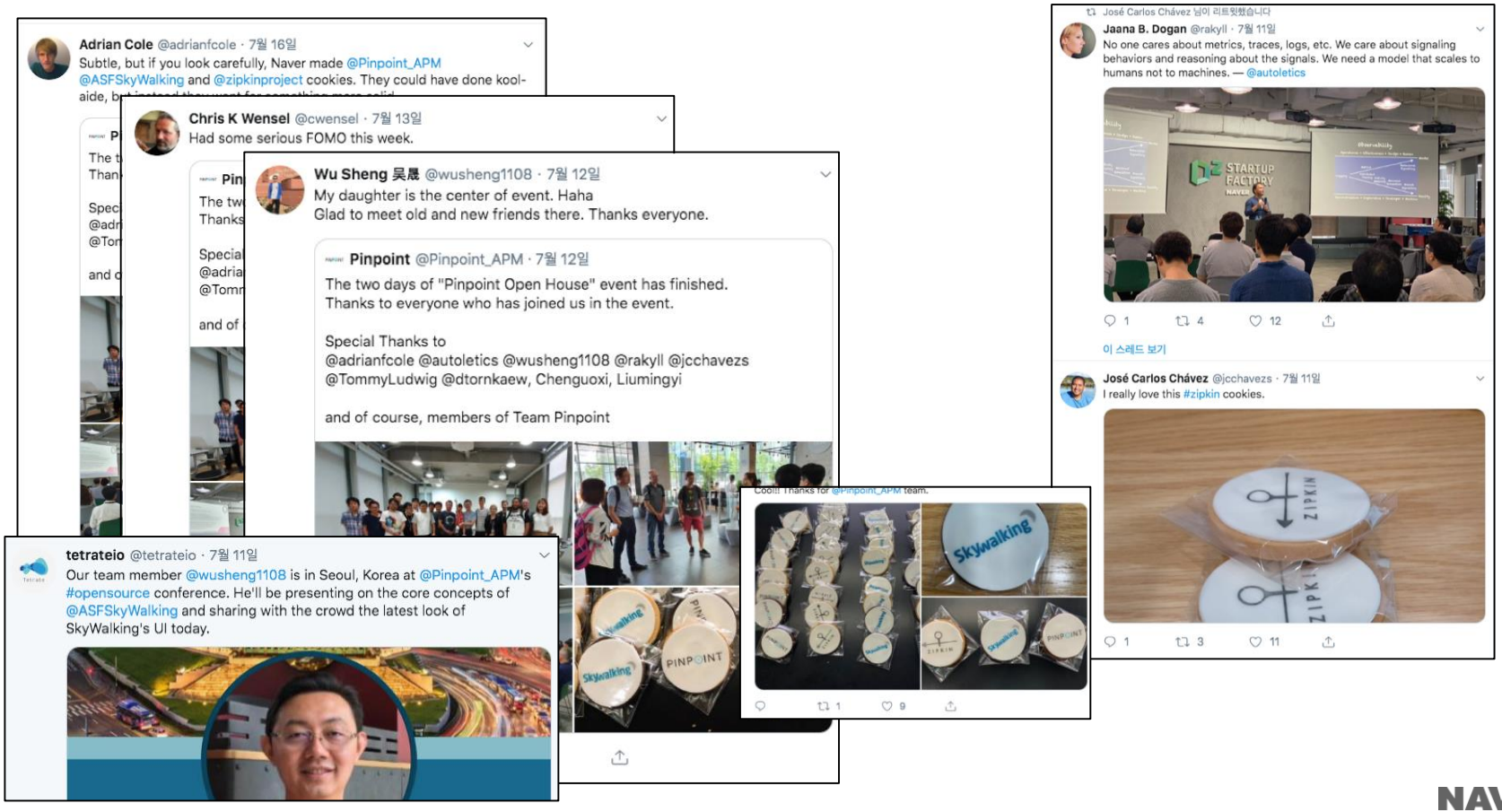
Open Source Community

Open Source Community

NAVER

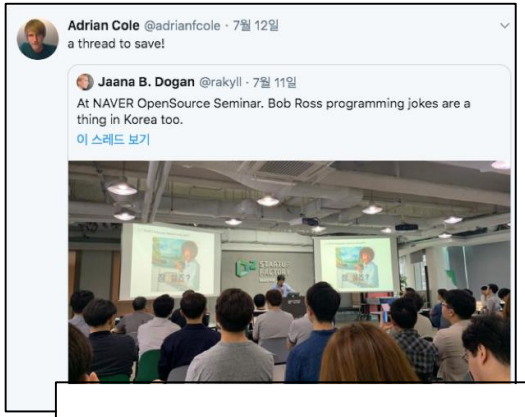
GOOD HOSTING!!!!

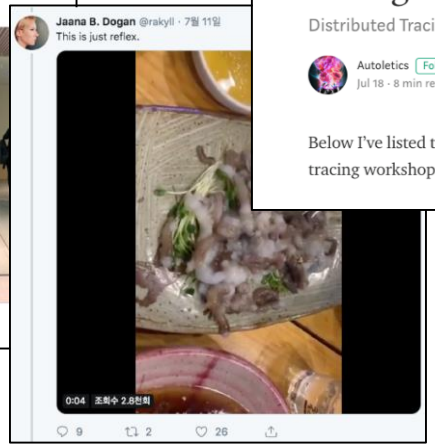
Open Source Community

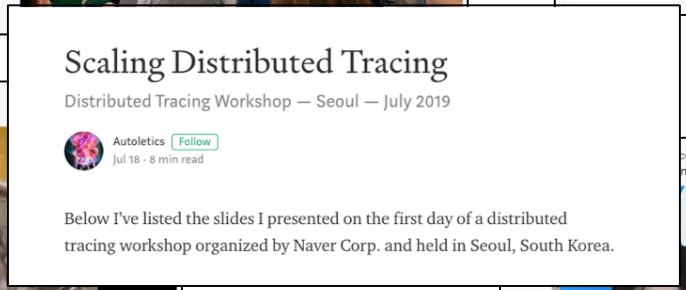


NAVER

Open Source Community



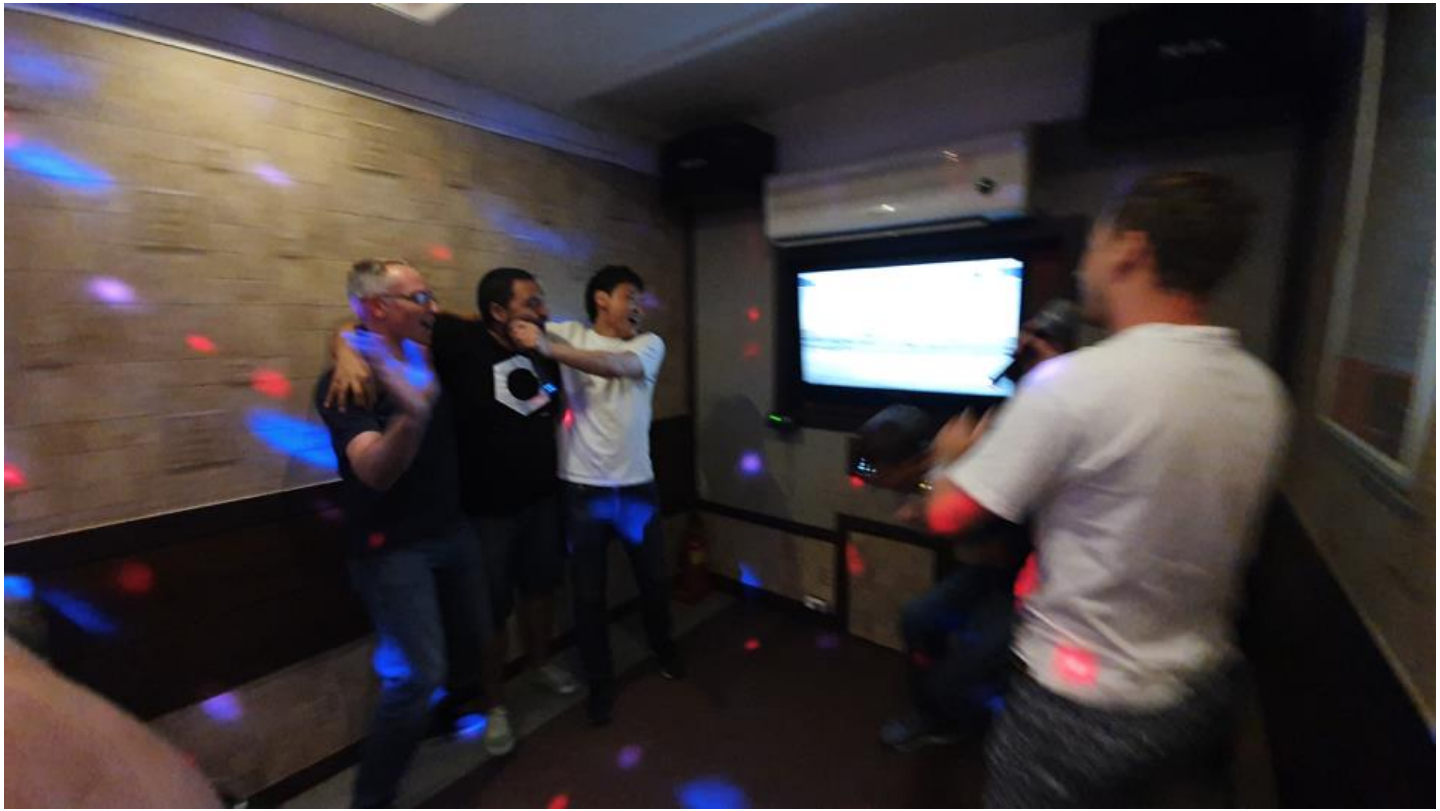




NAVER

Open Source Community

Open Source Community



NAVER

Open Source Community

Microservice Challenges

More areas that can fail

99% success rate over 10 components -> 90.5% success rate

Network error much more relevant

Increase in latency

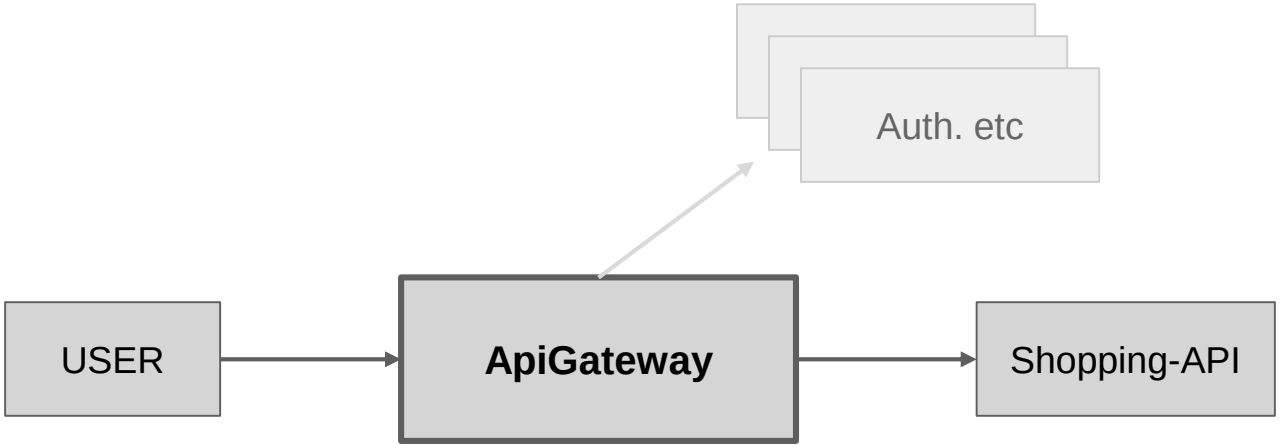
1s 99th percentile over 10 components -> approx. 80th percentile

Observability greatly reduced

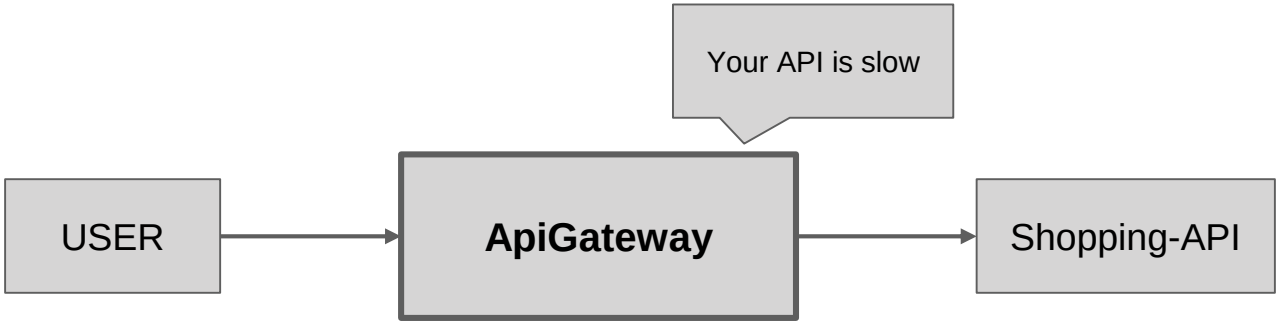
Individual servers does not tell the whole story

Traditional way of troubleshooting no longer works

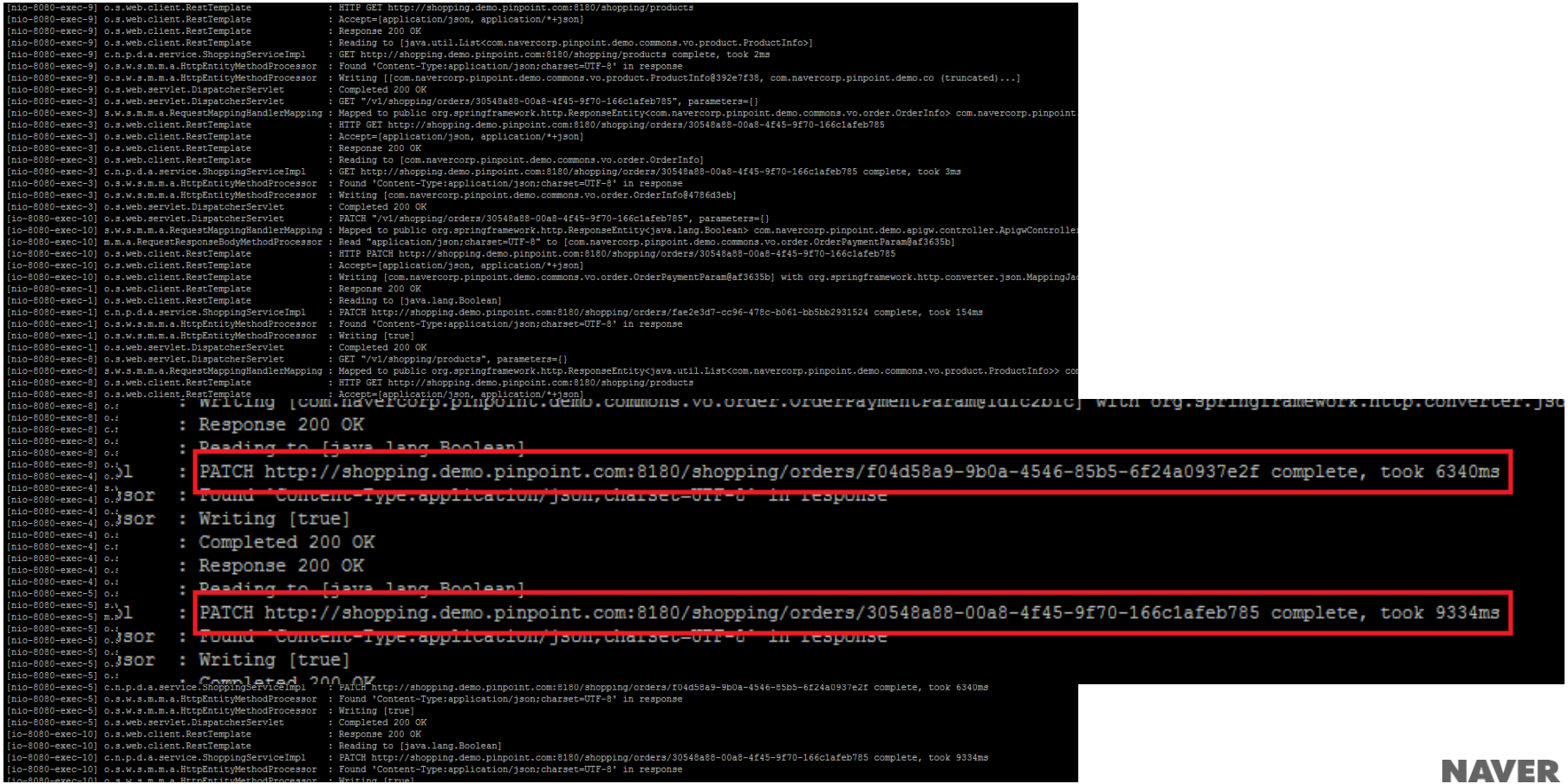
Troubleshooting Microservices



Troubleshooting Microservices

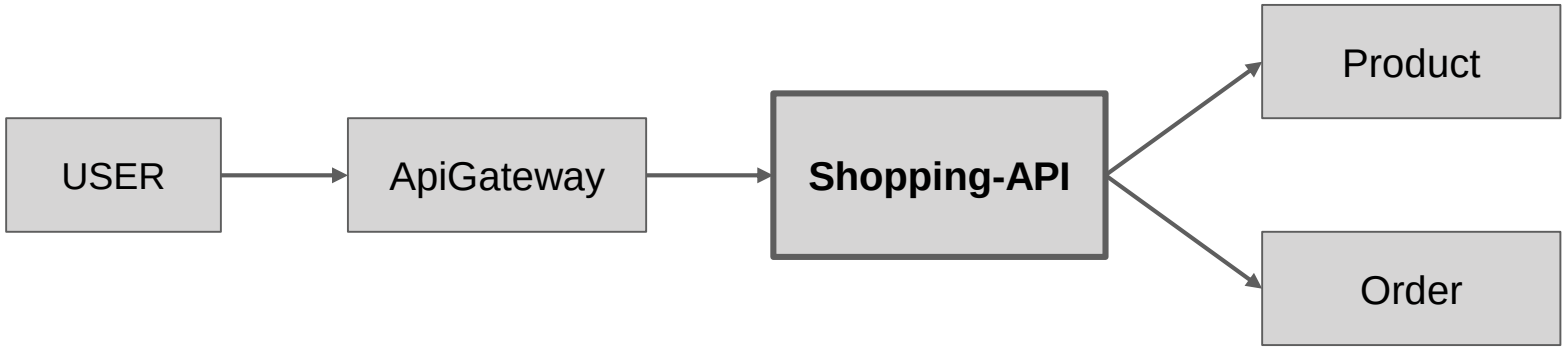


Troubleshooting Microservices

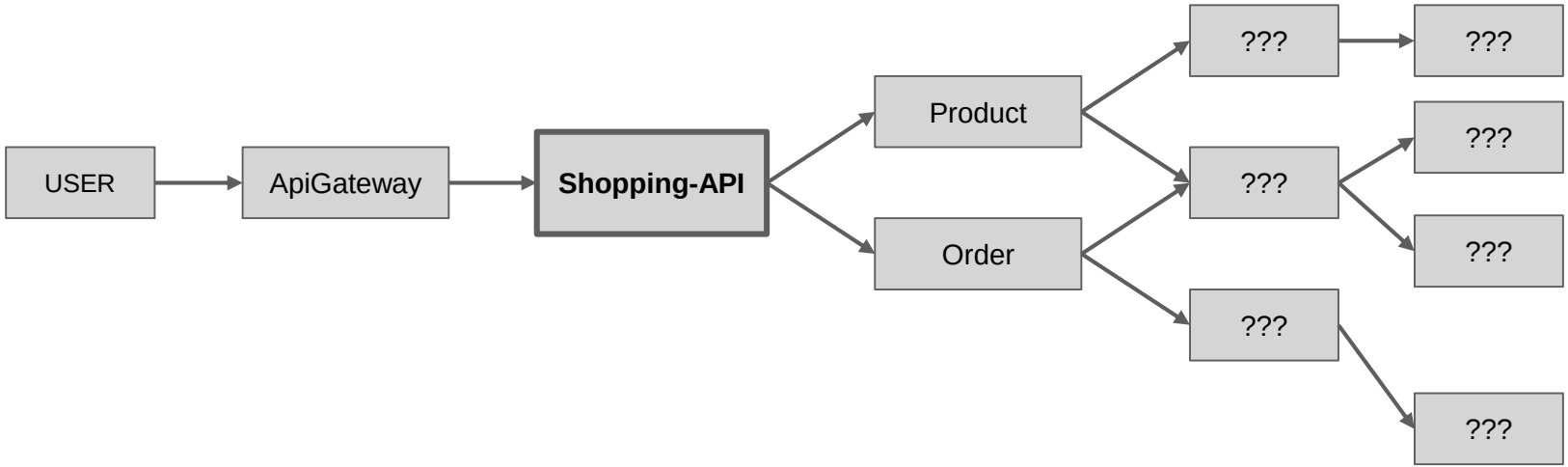


NAVER

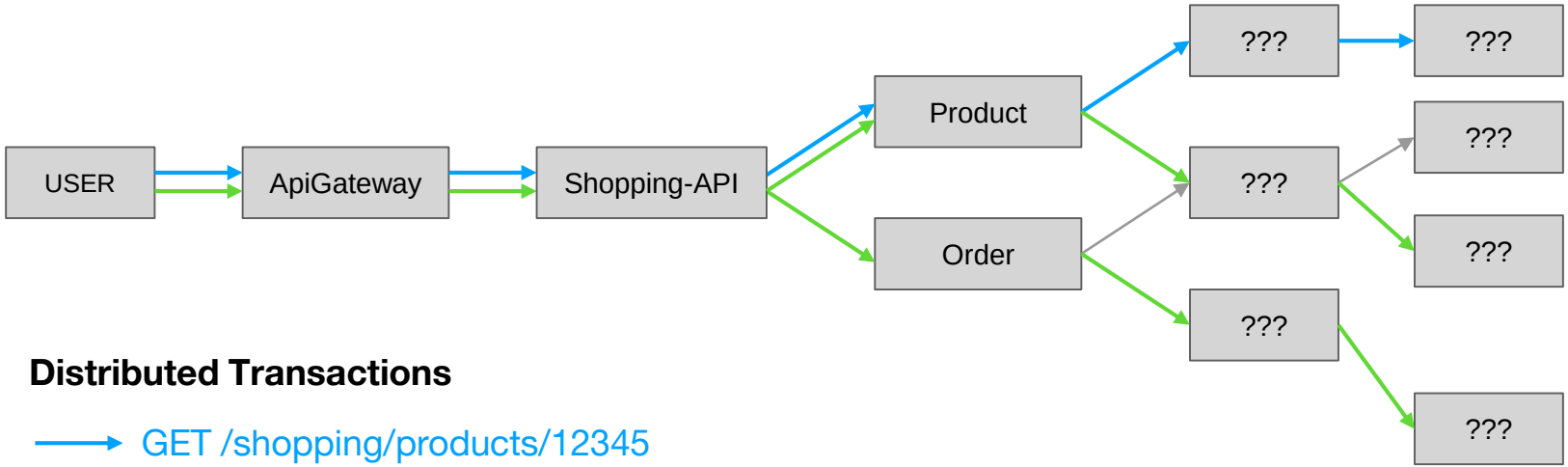
Troubleshooting Microservices



Troubleshooting Microservices



Troubleshooting Microservices

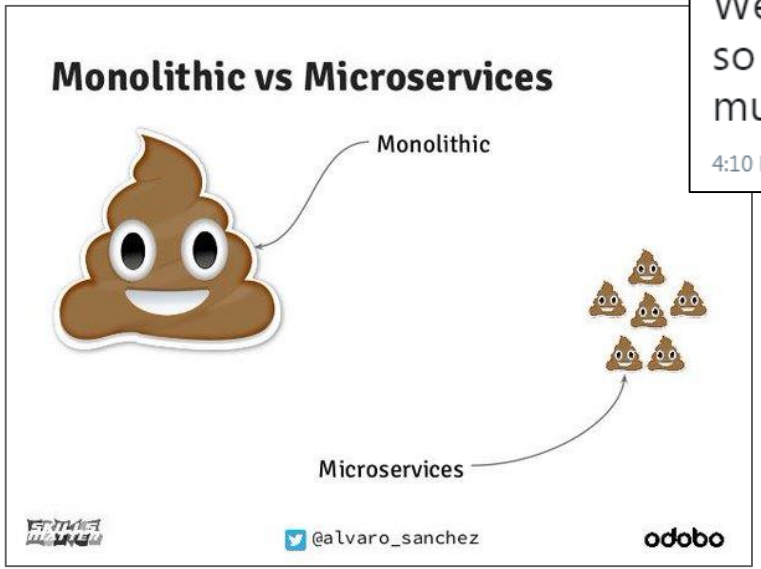


Distributed Transactions

GET /shopping/products/12345

POST /shopping/orders

Troubleshooting Microservices



www.slideshare.net/alvarosanchezmariscal/stateless-authentication-for-microservices

@honest_update 트위터

So what do we need?

Context

- Identify as part of the same request within a single service instance
- Identify as part of the same transaction within distributed service instances
- Context propagation across thread, or process boundary

Order

- Timestamp not enough
- Distributed Nodes - time skew
- Asynchronous Processing - delayed execution

Structure

- Order - 1 dimensional
- Call tree requires depth as well

Pinpoint

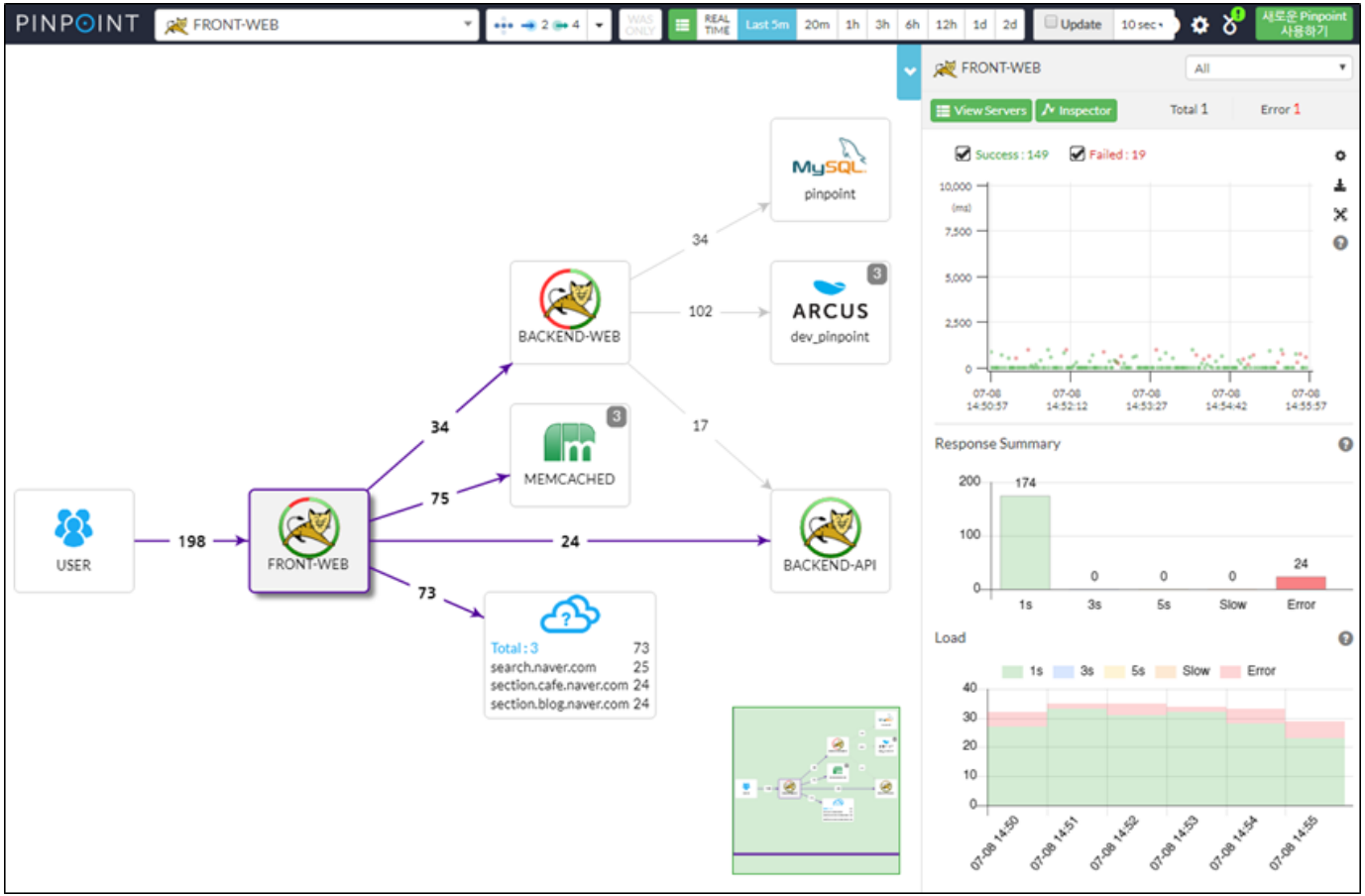
Call Stack Trace

- Traces everything that happens in a single instance
- Context propagated via thread-local
- Order and structure inherently provided by emulating the call stack

Distributed Transaction Trace

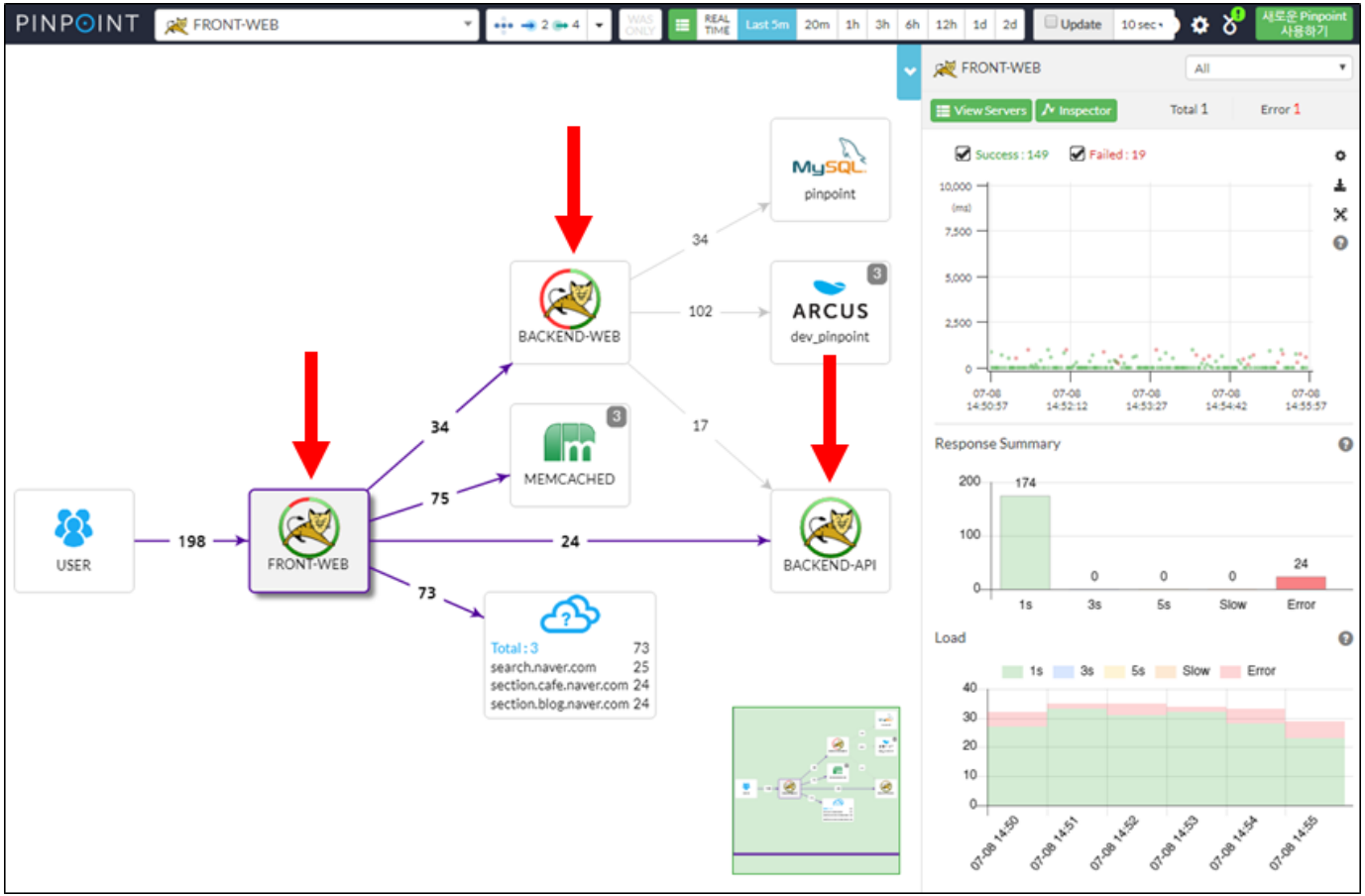
- Stitches multiple call stacks under the same transaction
- Context propagated via RPC
- Order and structure via parent-child relationship of distributed call stacks

Server Map



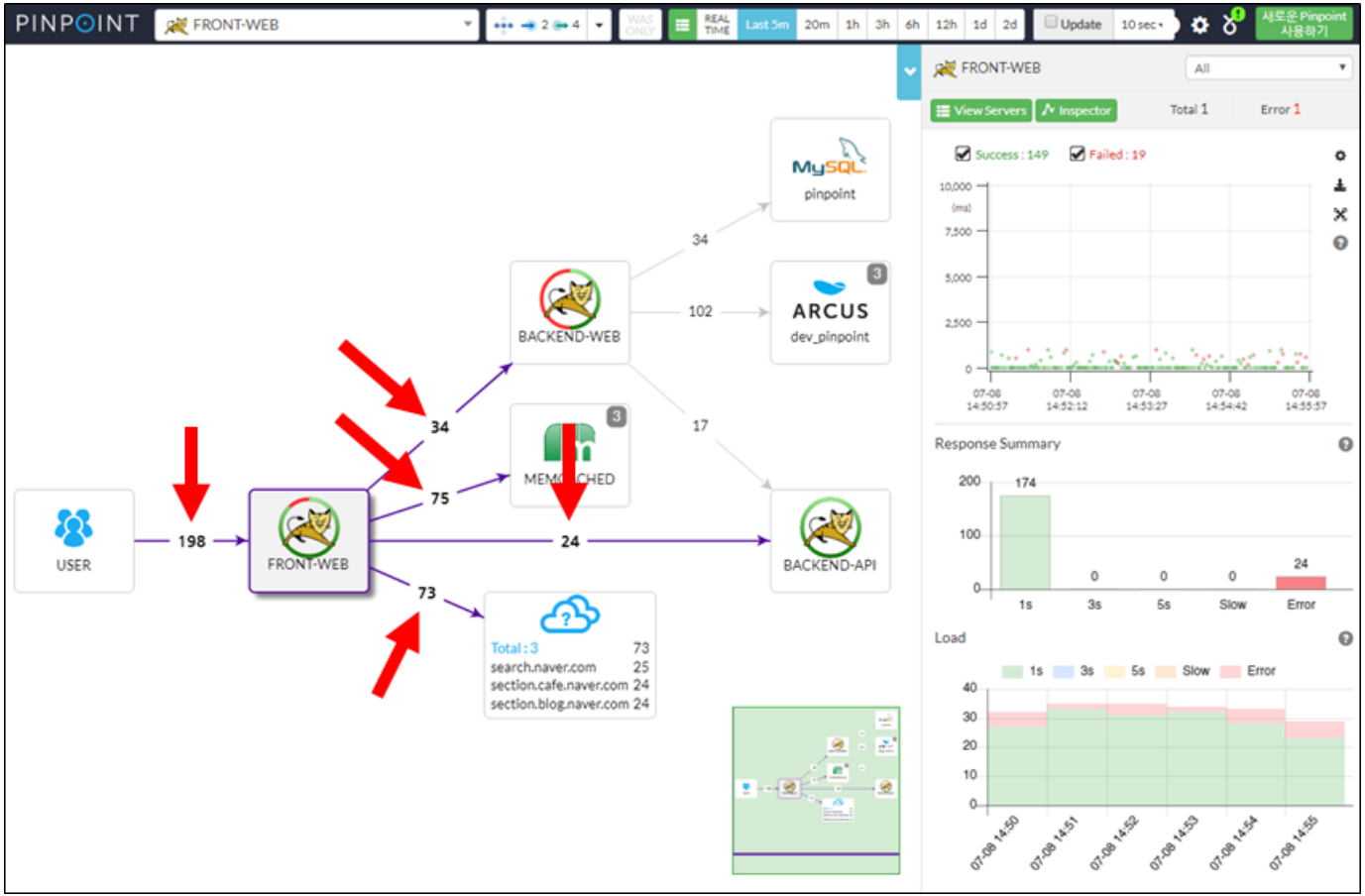
NAVER

Server Map



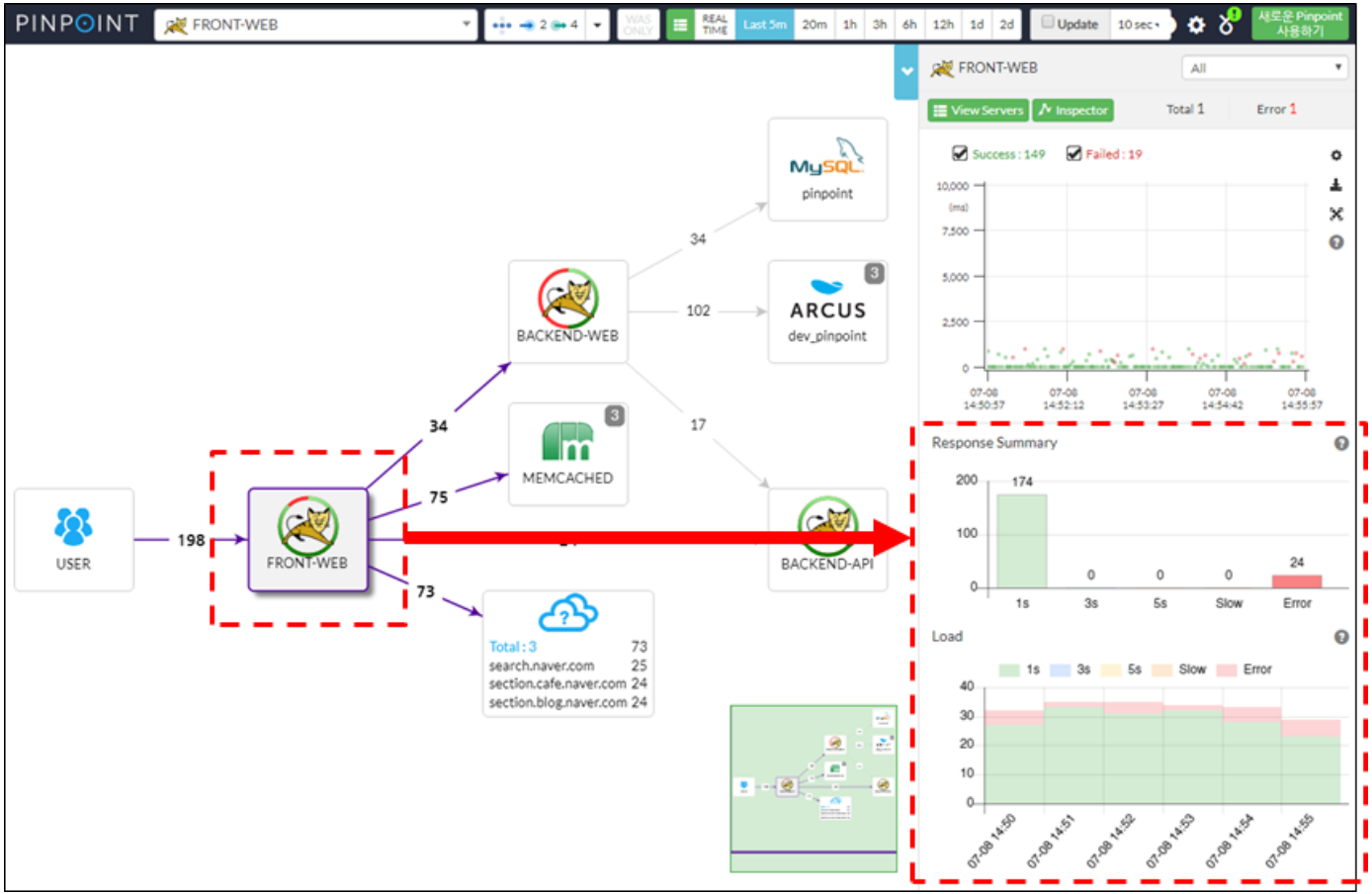
NAVER

Server Map



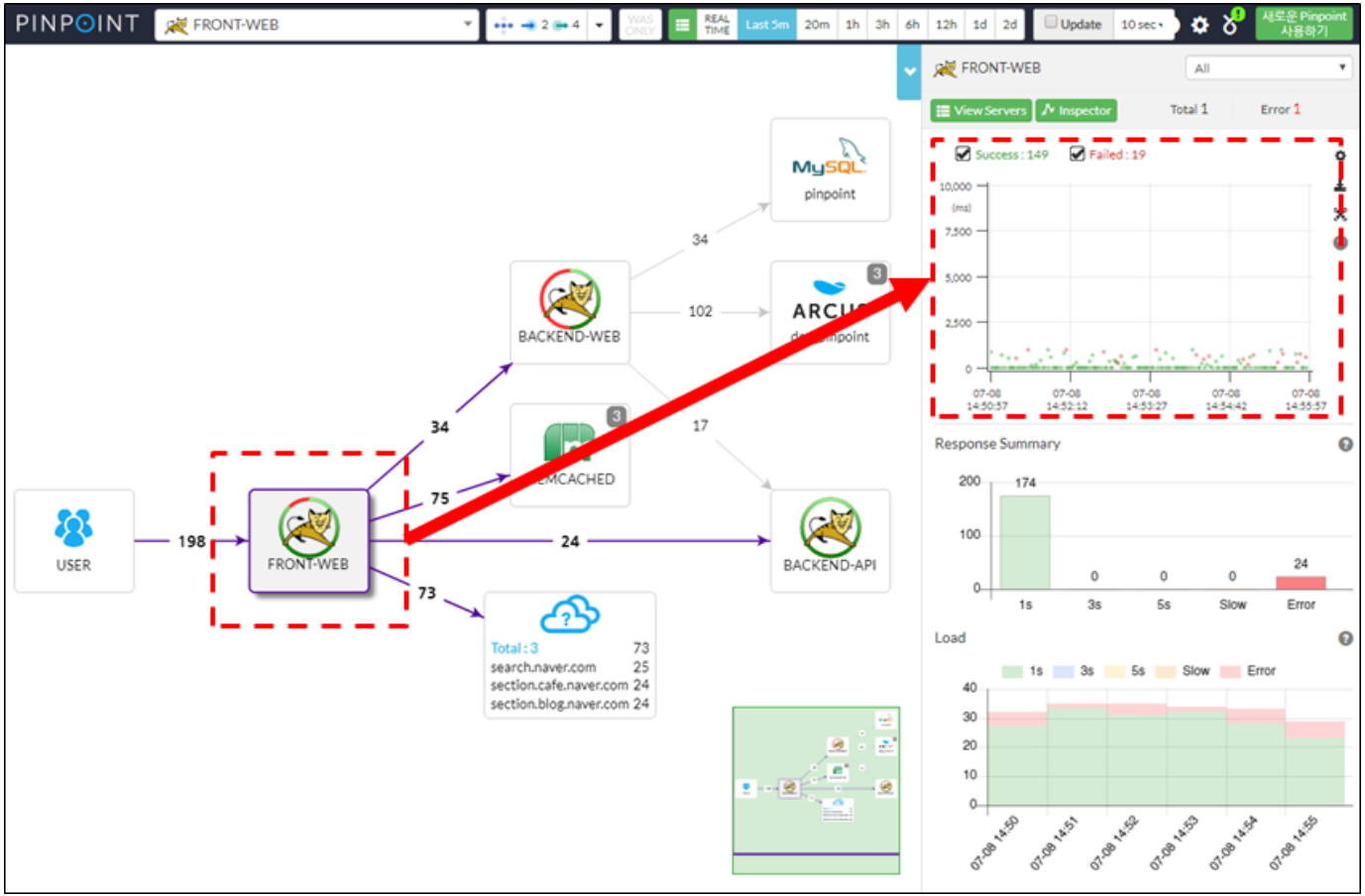
NAVER

Server Map



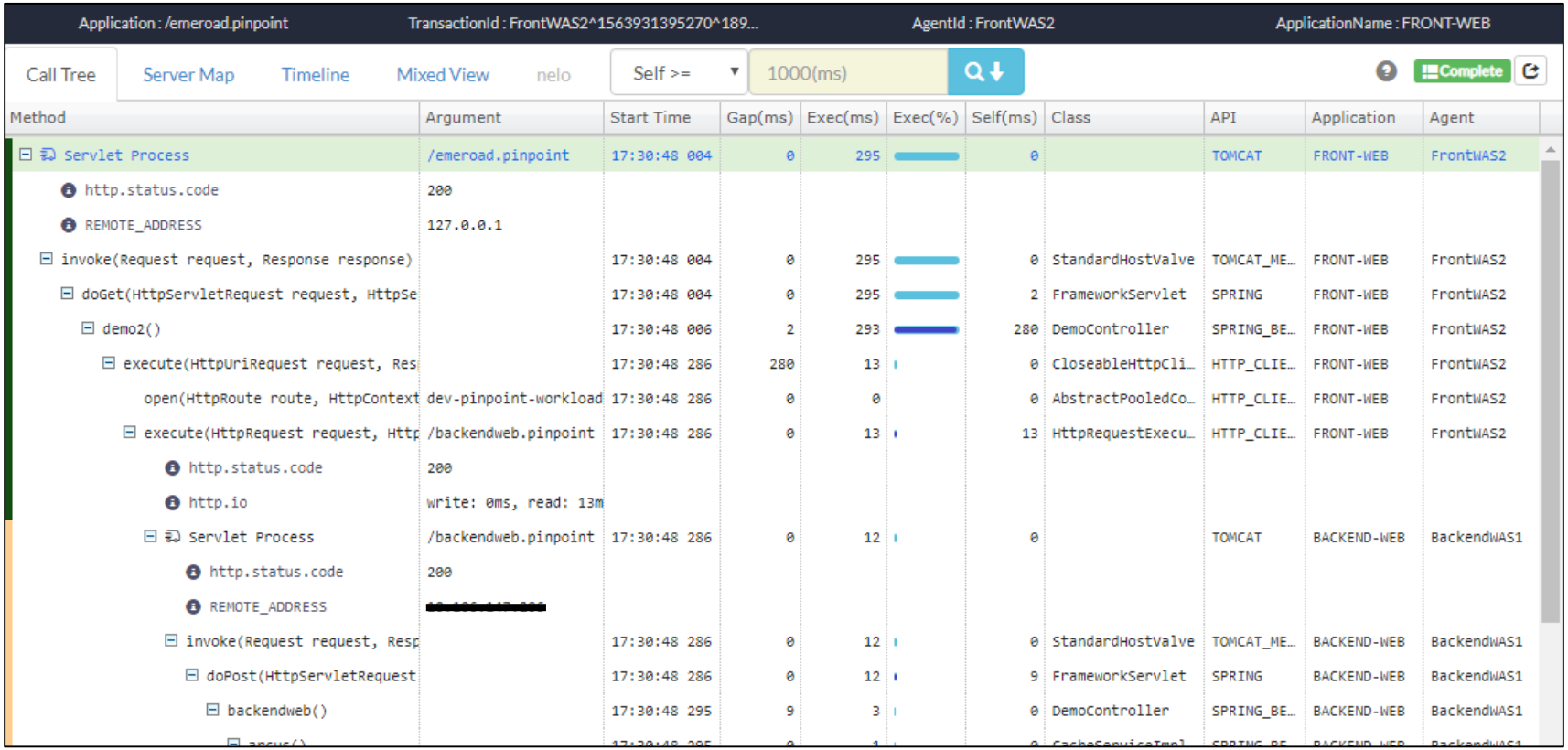
NAVER

Server Map



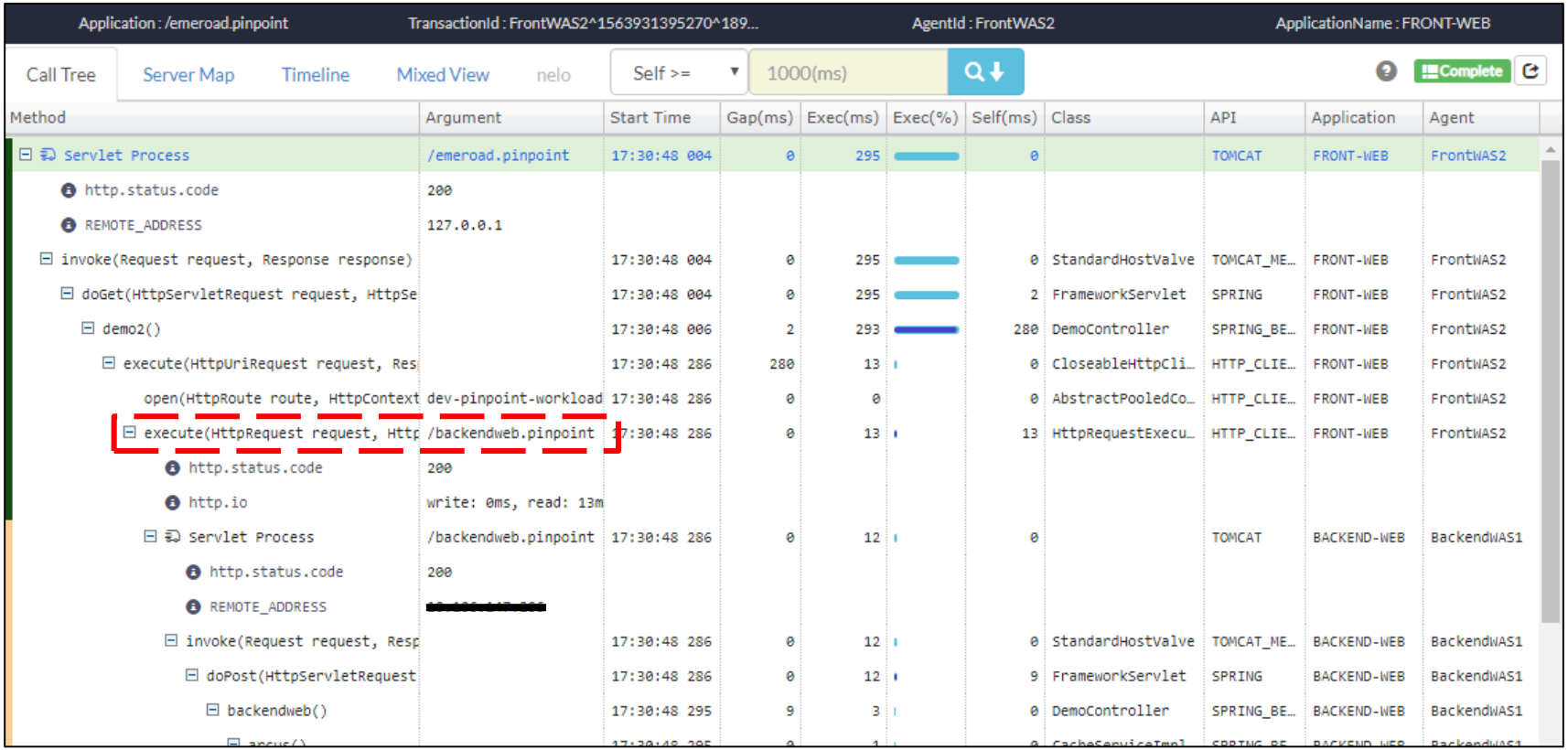
NAVER

Distributed Call Tree



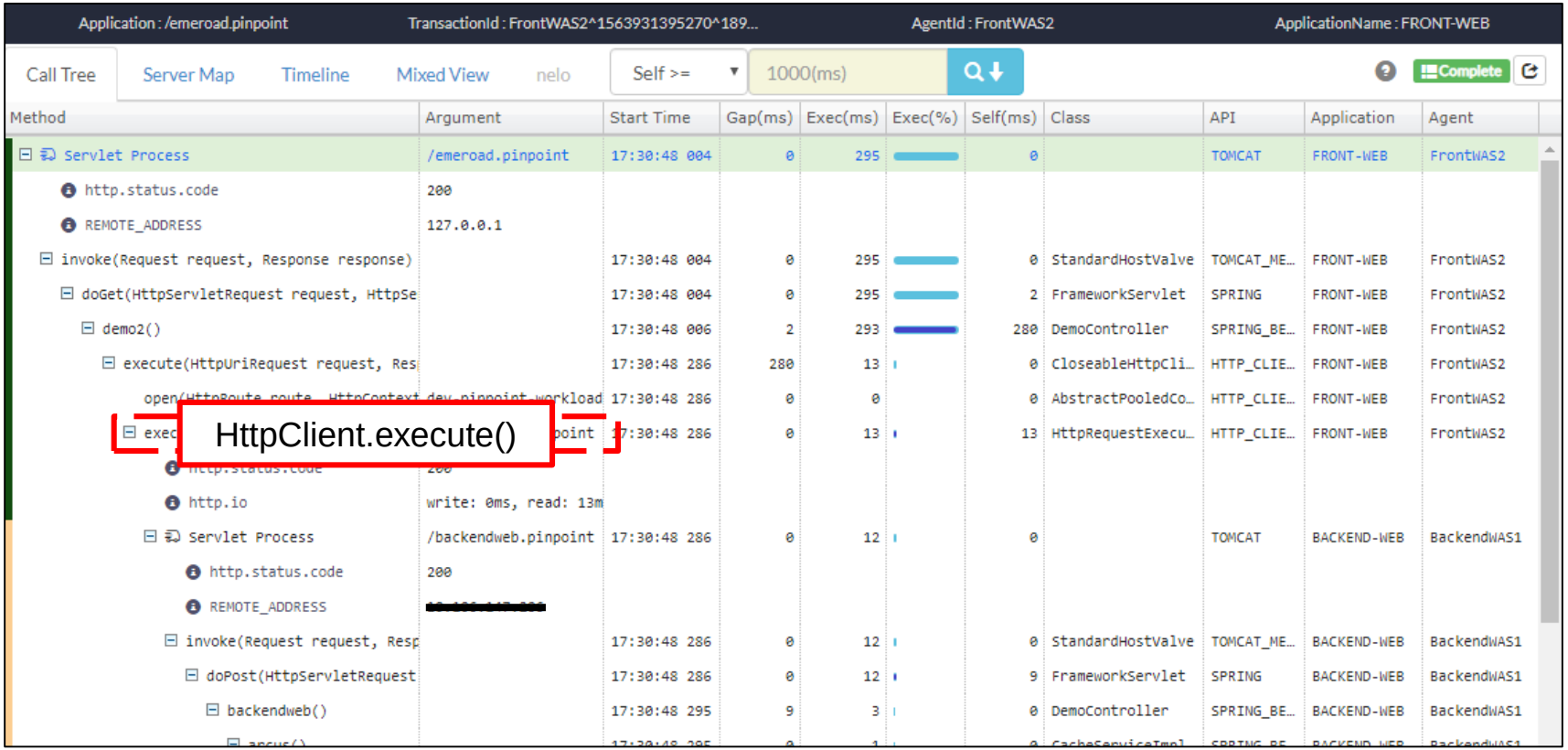
NAVER

Distributed Call Tree

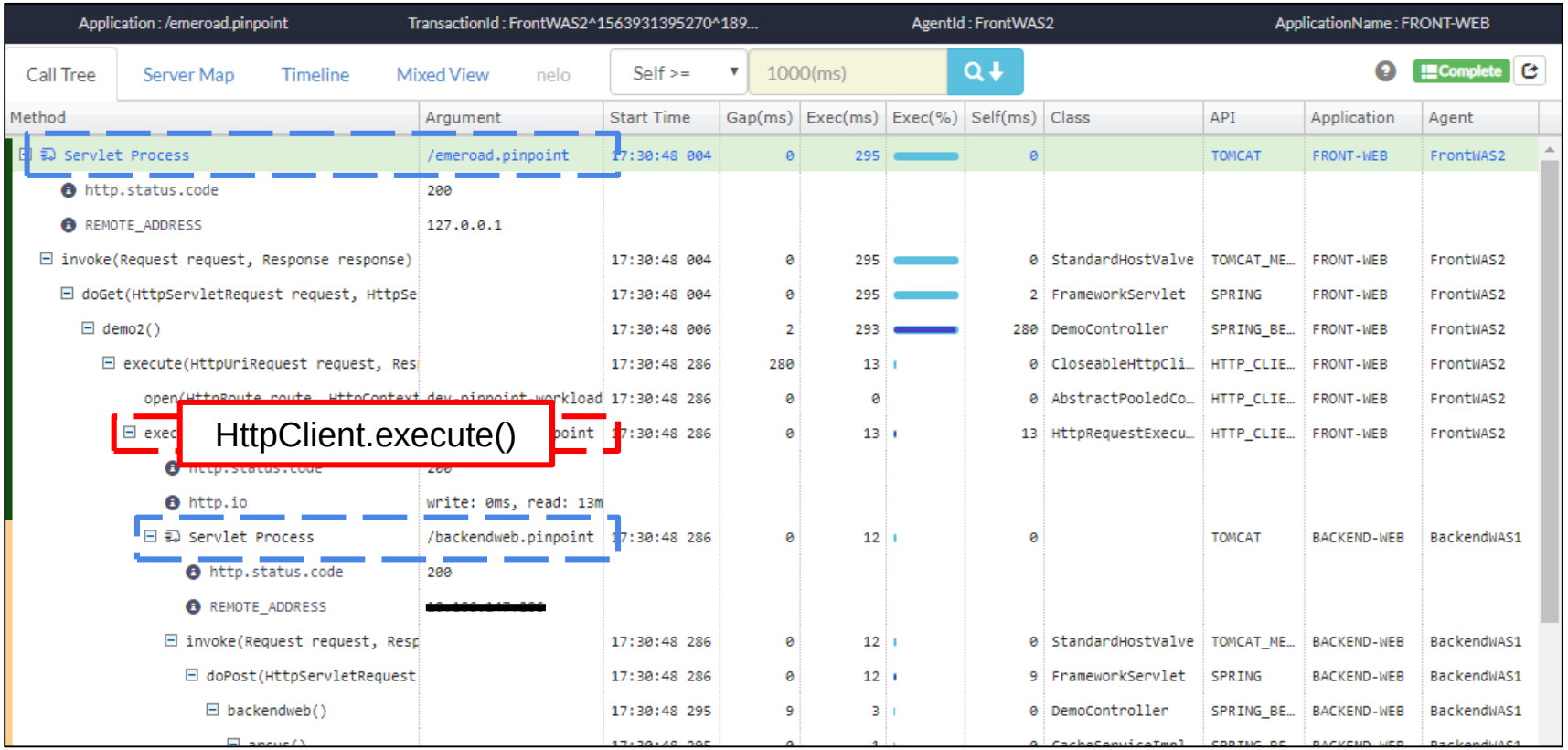


NAVER

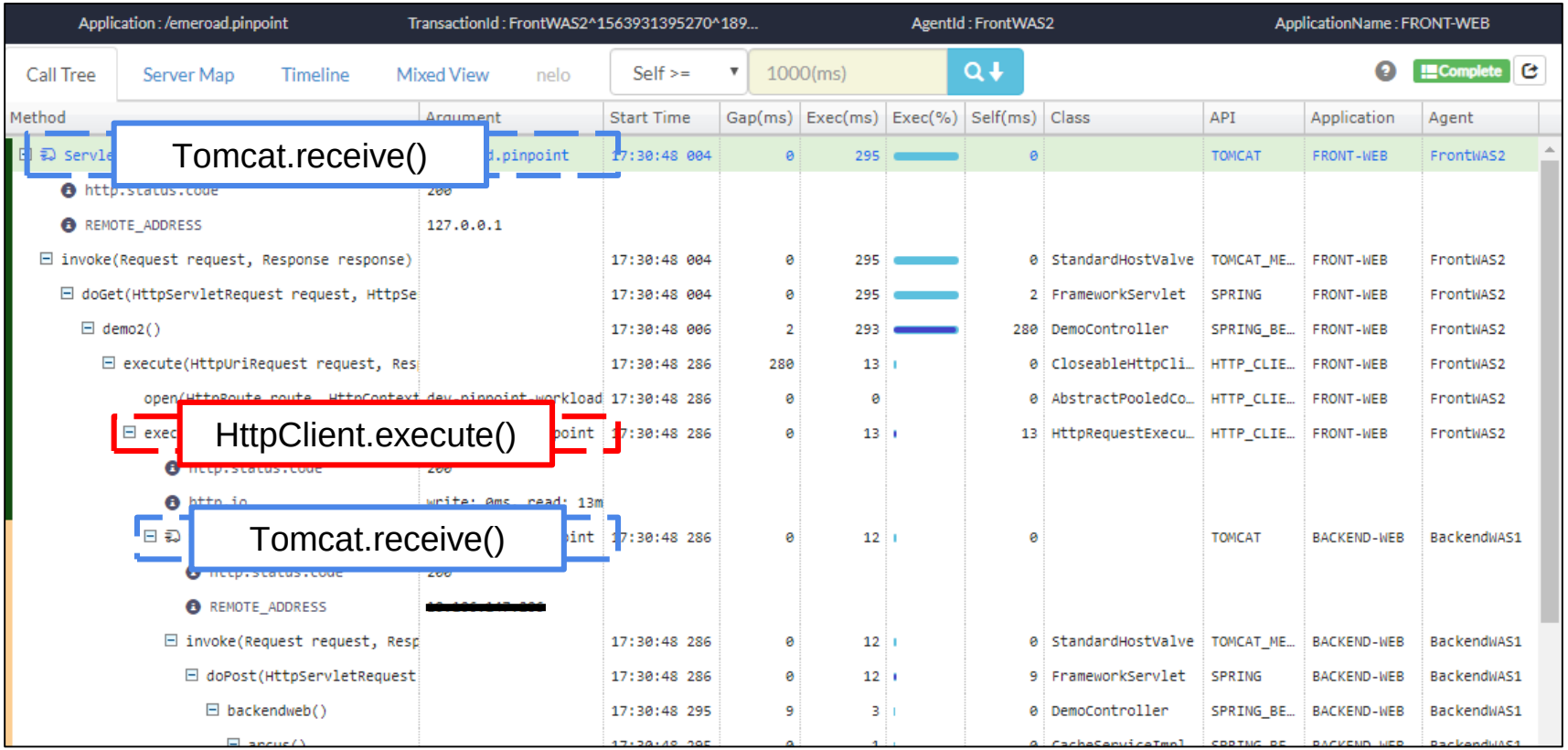
Distributed Call Tree



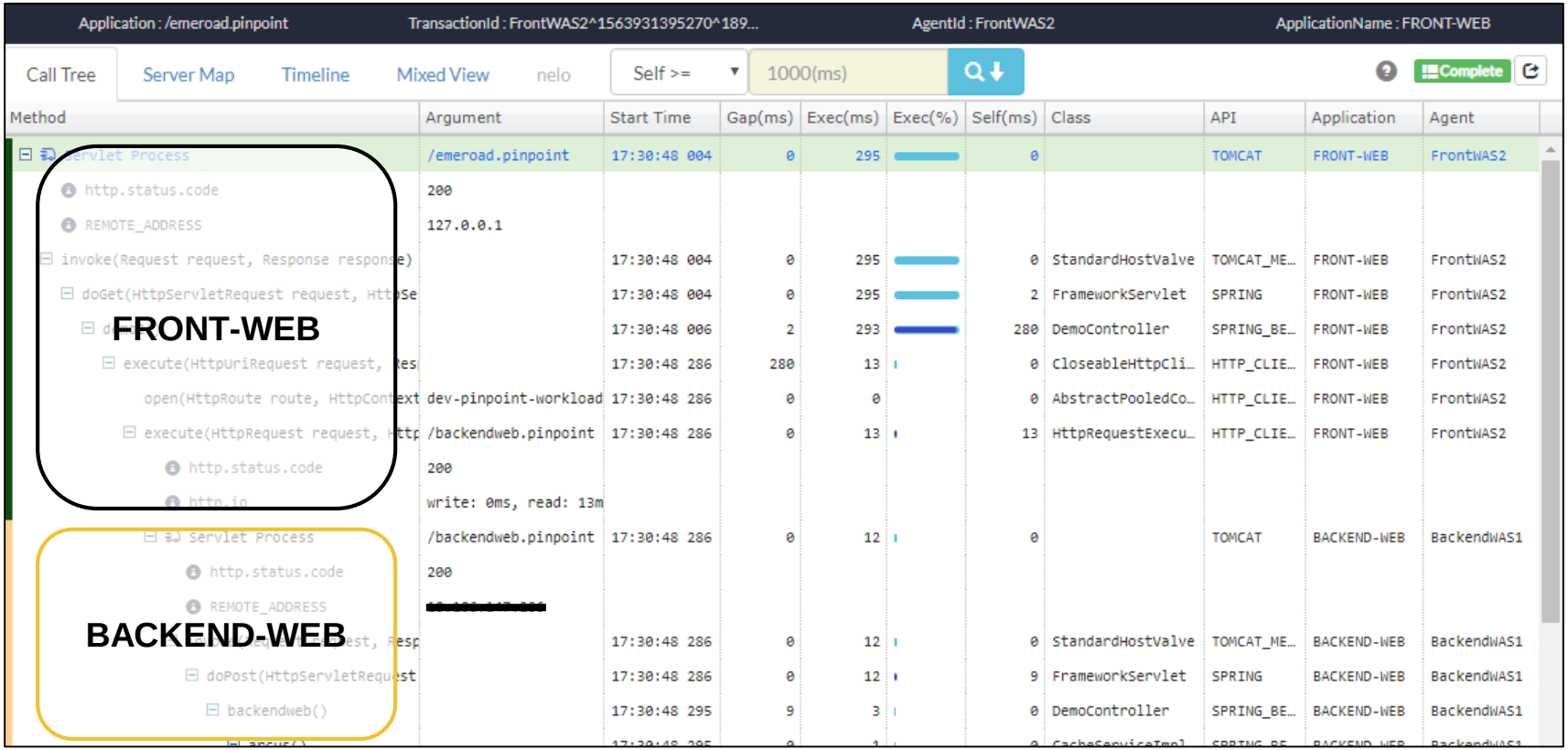
Distributed Call Tree



Distributed Call Tree



Distributed Call Tree



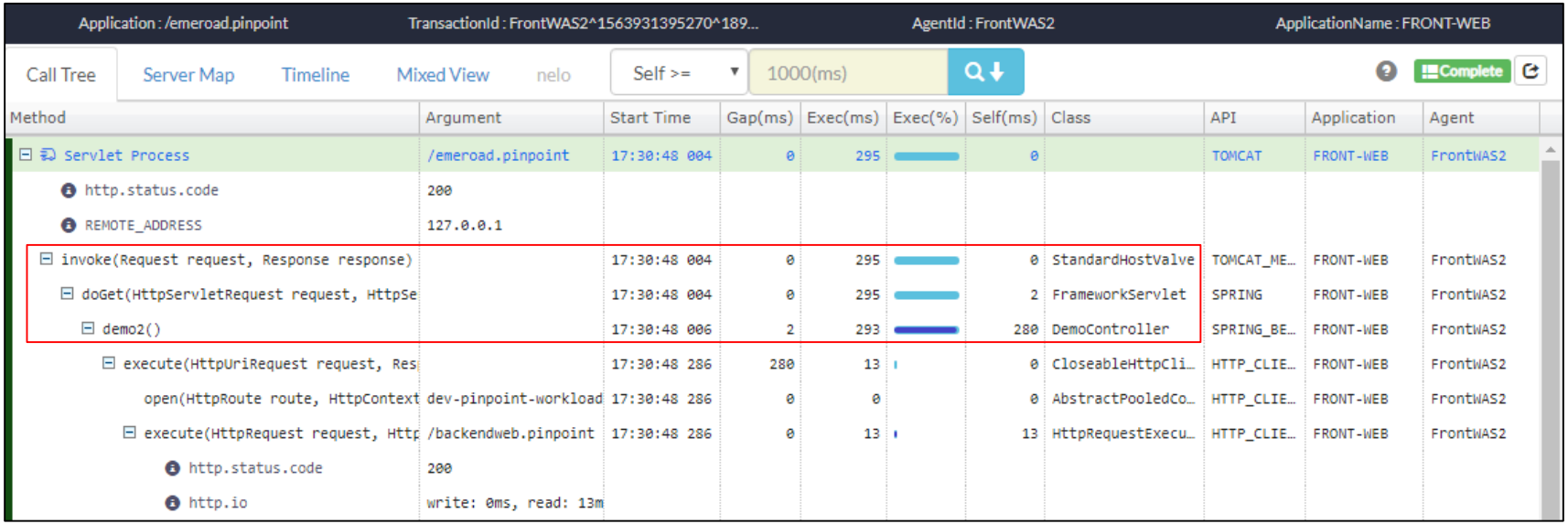
NAVER

How it all works

Call Stack Trace

Distributed Transaction Trace

Call Stack Trace



Method	Argument	Start Time	Gap(ms)	Exec(ms)	Exec(%)	Self(ms)	Class	API	Application	Agent
Servlet Process	/emeroad.pinpoint	17:30:48 004	0	295		0		TOMCAT	FRONT-WEB	FrontWAS2
http.status.code	200									
REMOTE_ADDRESS	127.0.0.1									
invoke(Request request, Response response)		17:30:48 004	0	295		0	StandardHostValve	TOMCAT_ME…	FRONT-WEB	FrontWAS2
doGet(HttpServletRequest request, HttpSe		17:30:48 004	0	295		2	FrameworkServlet	SPRING	FRONT-WEB	FrontWAS2
demo2()		17:30:48 006	2	293		280	DemoController	SPRING_BE…	FRONT-WEB	FrontWAS2
execute(HttpUriRequest request, Res		17:30:48 286	280	13		0	CloseableHttpCli…	HTTP_CLIE…	FRONT-WEB	FrontWAS2
open(HttpRoute route, HttpContext	dev-pinpoint-workload	17:30:48 286	0	0		0	AbstractPooledCo…	HTTP_CLIE…	FRONT-WEB	FrontWAS2
execute(HttpRequest request, Http	/backendweb.pinpoint	17:30:48 286	0	13		13	HttpRequestExecu…	HTTP_CLIE…	FRONT-WEB	FrontWAS2
http.status.code	200									
http.io	write: 0ms, read: 13m									

```
invoke(); -> doGet(); -> demo2();
```

Call Stack Trace

```
invoke(); -> doGet(); -> demo2();
```

```
invoke() {          doGet() {          demo2() {
  doGet();            demo2();         }
}                   }
```

Call Stack Trace

```
invoke(); -> doGet(); -> demo2();
```

```
invoke() {          doGet() {          demo2() {
  doGet();            demo2();         }
}                   }
```

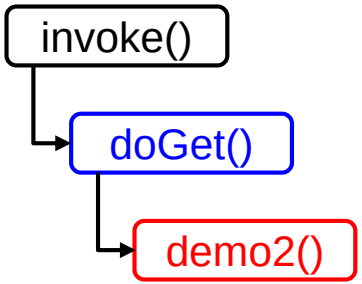
```
invoke() {
  doGet() {
    demo2() {
    }
  }
}
```

Call Stack Trace

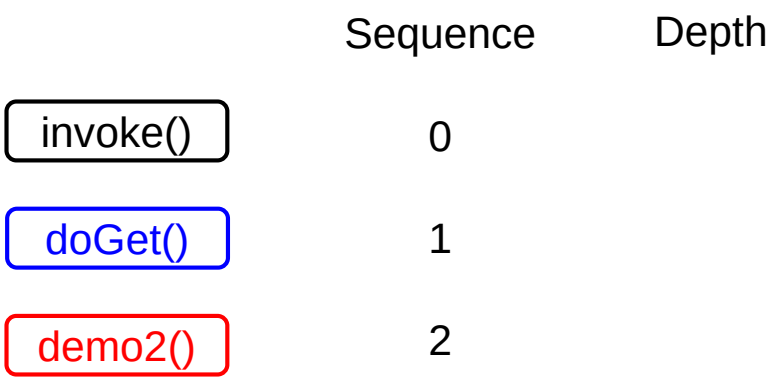
```
invoke(); -> doGet(); -> demo2();
```

```
invoke() {          doGet() {          demo2() {
  doGet();            demo2();         }
}                   }
```

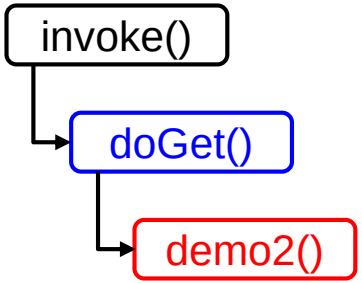
```
invoke() {
  doGet() {
    demo2() {
    }
  }
}
```



Call Stack Trace

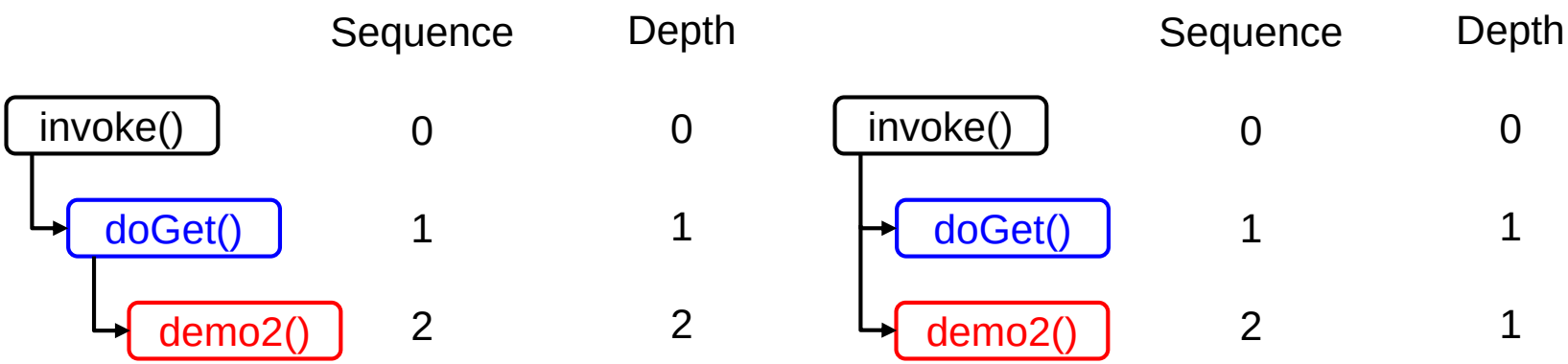


Call Stack Trace



	Sequence	Depth
invoke()	0	0
doGet()	1	1
demo2()	2	2

Call Stack Trace



Call Stack Trace

```
foo() {
  fooInterceptor.before();
  …
  fooInterceptor.after();
}
```

Call Stack Trace

```
invoke() {
  invokeInterceptor.before();
  doGet() {
    doGetInterceptor.before();
    demo2() {
      demo2Interceptor.before();
      demo2Interceptor.after();
    }
    doGetInterceptor.after();
  }
  invokeInterceptor.after();
}
```

Call Stack Trace

```
invoke() {
  invokeInterceptor.before();
  doGet() {
    doGetInterceptor.before();
    demo2() {
      demo2Interceptor.before();
      demo2Interceptor.after();
    }
    doGetInterceptor.after();
  }
  invokeInterceptor.after();
}
```

before()

Increment sequence

Push on to the call stack

Seq.	Event	Depth
		3
		2
		1
push 0	invoke()	0

Call Stack Trace

```
invoke() {
  invokeInterceptor.before();
  doGet() {
    doGetInterceptor.before();
    demo2() {
      demo2Interceptor.before();
      demo2Interceptor.after();
    }
    doGetInterceptor.after();
  }
  invokeInterceptor.after();
}
```

before()

Increment sequence

Push on to the call stack

Seq.	Event	Depth
		3
		2
push 1	doGet()	1
0	invoke()	0

Call Stack Trace

```
invoke() {
  invokeInterceptor.before();
  doGet() {
    doGetInterceptor.before();
    demo2() {
      demo2Interceptor.before();
      demo2Interceptor.after();
    }
    doGetInterceptor.after();
  }
  invokeInterceptor.after();
}
```

before()

Increment sequence

Push on to the call stack

push

Seq.	Event	Depth
		3
2	demo2()	2
1	doGet()	1
0	invoke()	0

Call Stack Trace

```
invoke() {
  invokeInterceptor.before();
  doGet() {
    doGetInterceptor.before();
    demo2() {
      demo2Interceptor.before();
      demo2Interceptor.after();
    }
    doGetInterceptor.after();
  }
  invokeInterceptor.after();
}
```

after()

Pop off the call stack

Buffer to write queue

Seq.	Event	Depth
		3
		2
1	doGet()	1
0	invoke()	0

pop

Write Queue

demo2()

Call Stack Trace

```
invoke() {
  invokeInterceptor.before();
  doGet() {
    doGetInterceptor.before();
    demo2() {
      demo2Interceptor.before();
      demo2Interceptor.after();
    }
    doGetInterceptor.after();
  }
  invokeInterceptor.after();
}
```

after()

Pop off the call stack

Buffer to write queue

Seq.	Event	Depth
		3
		2
		1
0	invoke()	0

pop

Write Queue

demo2() doGet()

Call Stack Trace

```
invoke() {
  invokeInterceptor.before();
  doGet() {
    doGetInterceptor.before();
    demo2() {
      demo2Interceptor.before();
      demo2Interceptor.after();
    }
    doGetInterceptor.after();
  }
  invokeInterceptor.after();
}
```

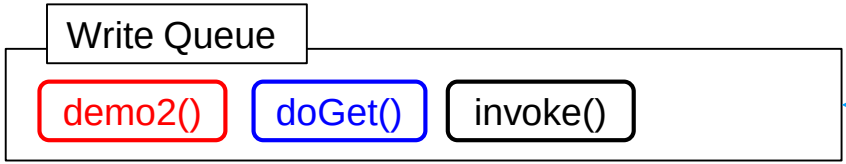
after()

Pop off the call stack

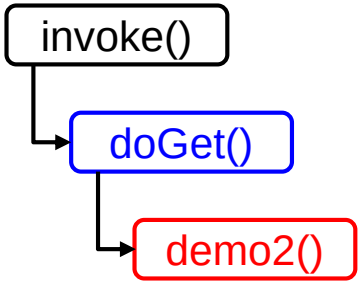
Buffer to write queue

Seq.	Event	Depth
		3
		2
		1
		0

pop

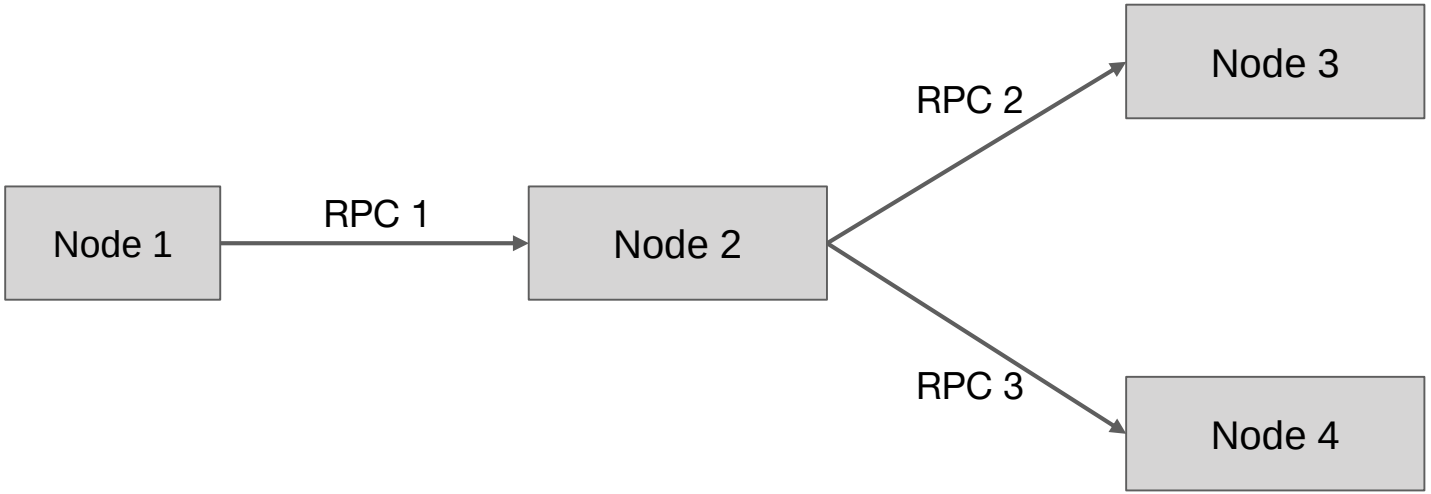


Call Stack Trace



	Sequence	Depth
invoke()	0	0
doGet()	1	1
demo2()	2	2

Distributed Transaction Trace

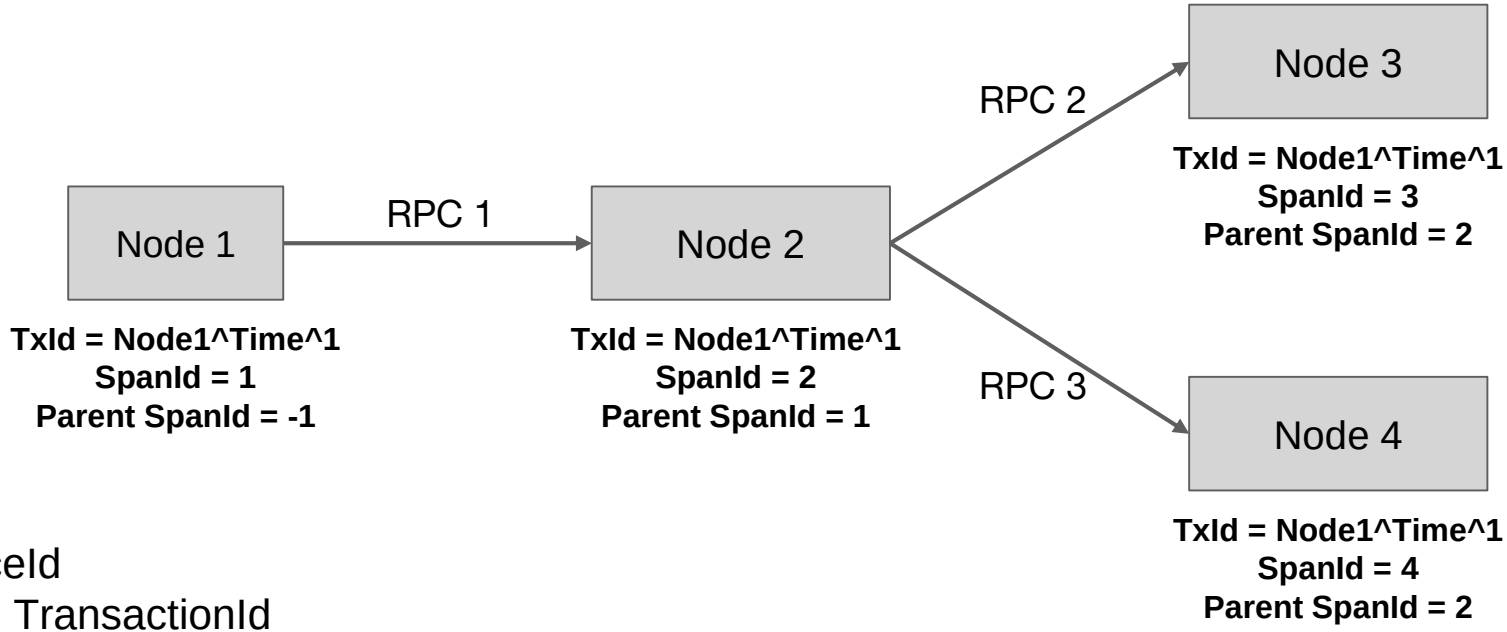


Find out the relationship between nodes connected by RPCs for a given transaction

Trace tag added to each requests

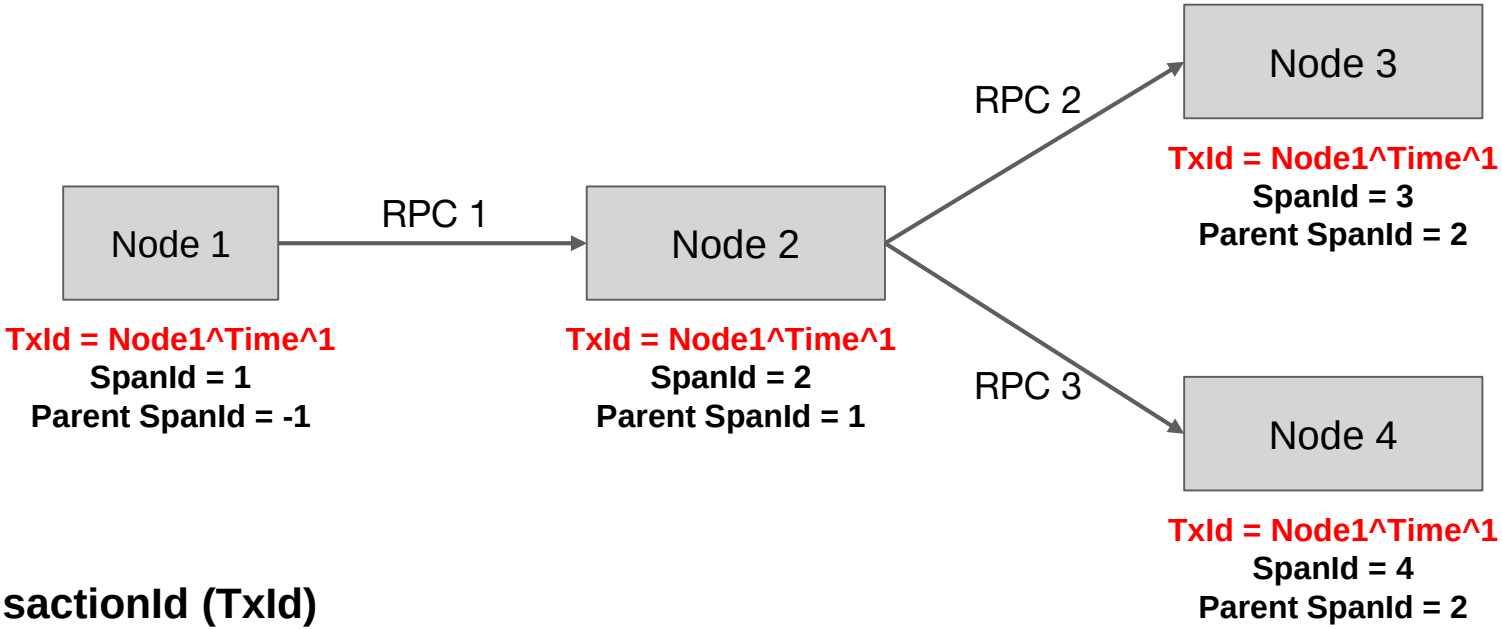
- HTTP : HttpHeader

Distributed Transaction Trace



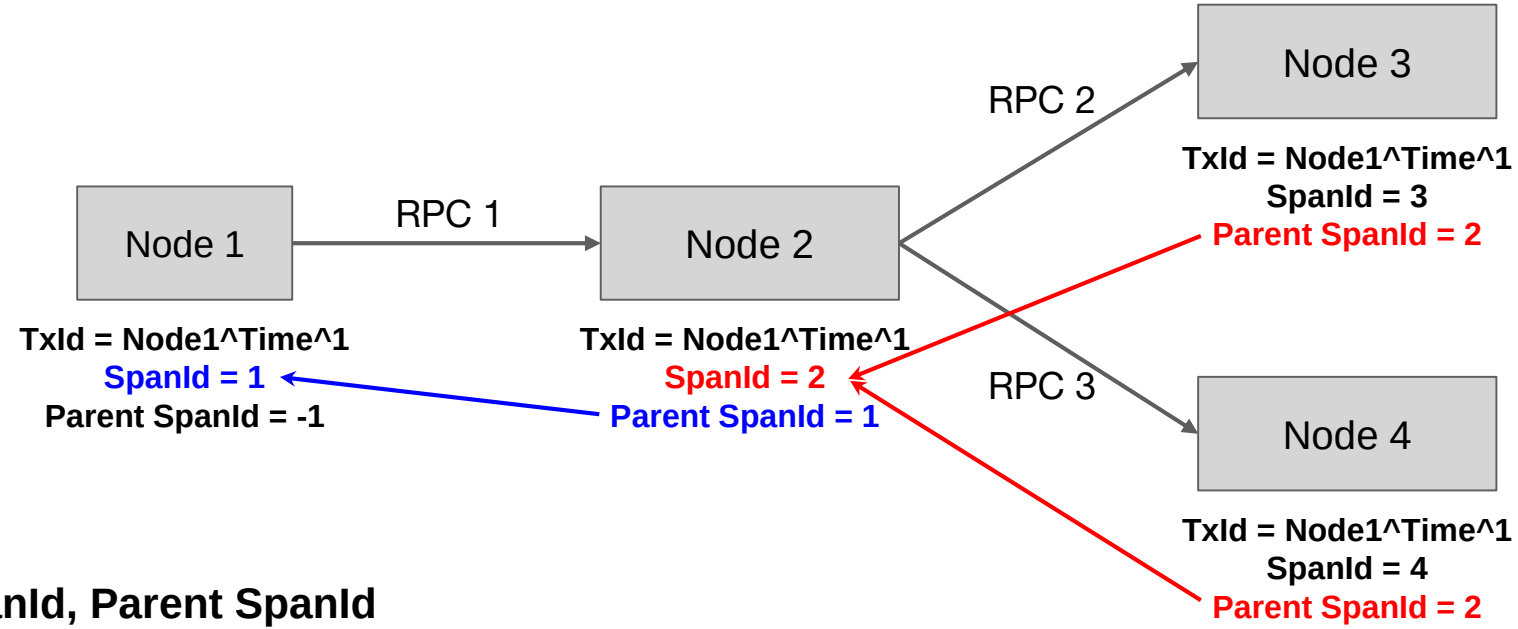
TraceId
- TransactionId
- SpanId
- Parent SpanId

Distributed Transaction Trace



TransactionId (TxId)
Globally unique ID for a single transaction

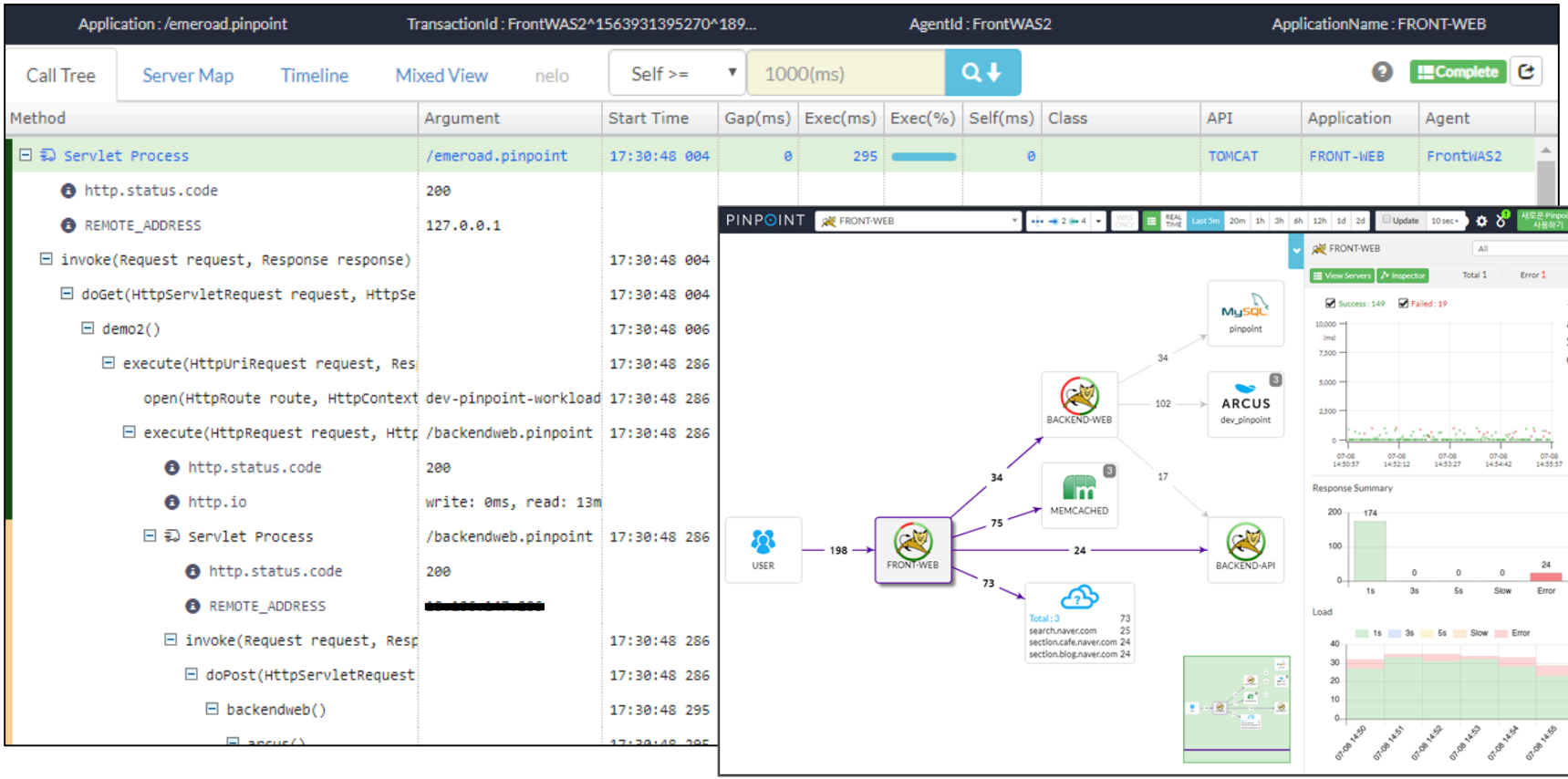
Distributed Transaction Trace



SpanId, Parent SpanId

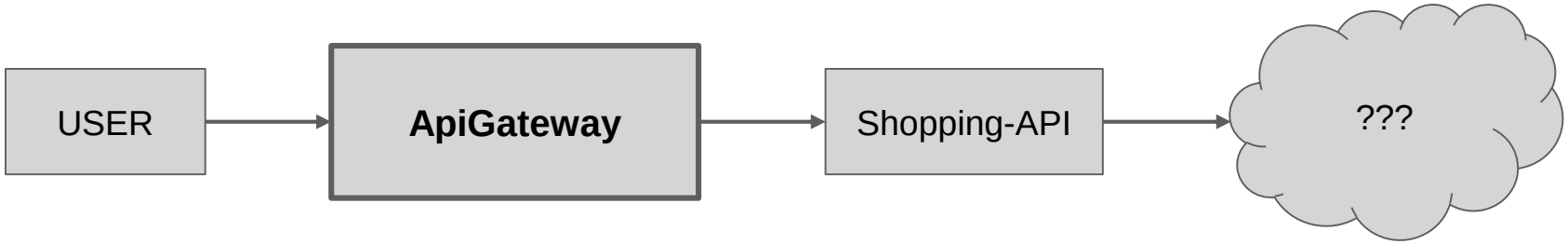
Id used to encode parent-child relationship between nodes

Distributed Call Tree



NAVER

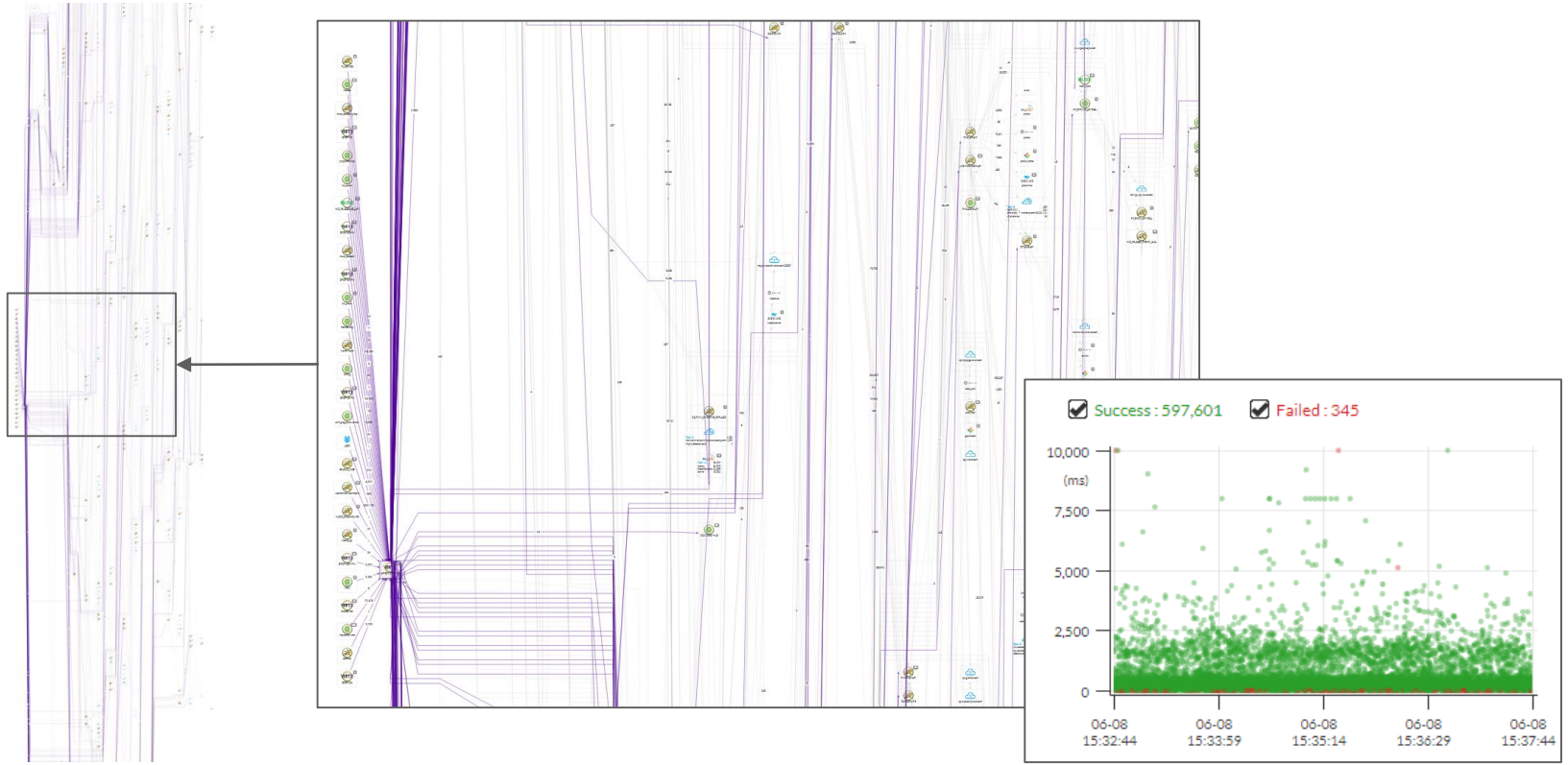
Troubleshooting with Pinpoint



Let’s troubleshoot our system using Pinpoint

(There’s a link @ www.github.com/naver/pinpoint)

Troubleshooting with Pinpoint



Future plans

Future Plans

Future Plans

kubernetes

Istio X

Thank you

Q & A

Homepage : https://naver.github.io/pinpoint

Github : https://github.com/naver/pinpoint

Twitter : https://twitter.com/Pinpoint_APM

E-mail : roy.kim@navercorp.com, hyungil.jeong@navercorp.com