End-to end Tracing, tracing Kubernetes

Aditya Chechani
Joshua Manning
Reet Chowdhary
Runzhou Han

2/8/2018

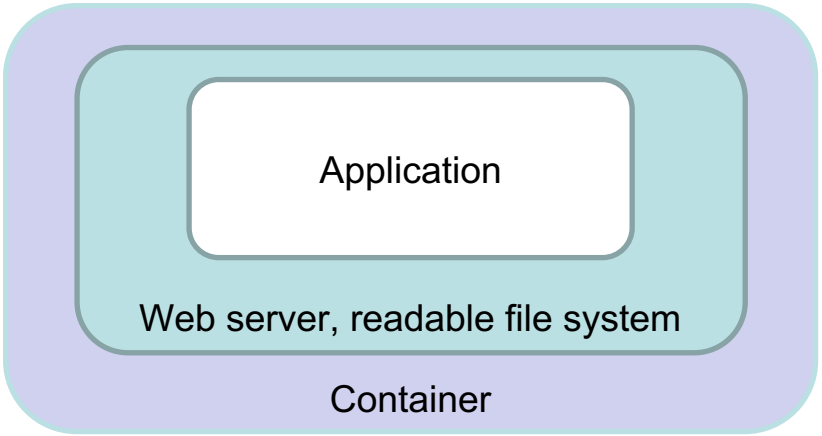
Taiga: https://tree.taiga.io/project/msdisme-2018-bucs528-template-7/

GitHub: https://github.com/BU-NU-CLOUD-SP18/Cross-Layer-Tracing-in-Kubernetes

BOSTON UNIVERSITY

Container

- Containers provide an isolated environment with a unique namespace
  - The environment satisfies a description of a set of resources required by an app
- Each App runs in a different container
  - An app can only use the resources defined in this namespace
  - App doesn't know what is outside its container



Kubernetes

- Kubernetes is an orchestration system coordinating a highly available cluster of computers for deploying, scaling and managing containerized components of a distributed application in a datacenter. It makes these applications agnostic and isolated from other containers deployed on the same node e.g. machine.
  - It allocates resources to containers to have them meet the required description
  - It offers a unique namespace to each container
  - It can scale in & scale out an application and make replication) when required
  - Provides an abstraction through which each container is able to communicate with the outside world

But why Kubernetes?

- Better management through modularity.
  - Faster Development.
  - Achieved using Pods.

- Deploying and updating software at scale.
  - Scalability
  - Visibility
  - Time Saving
  - Version Control

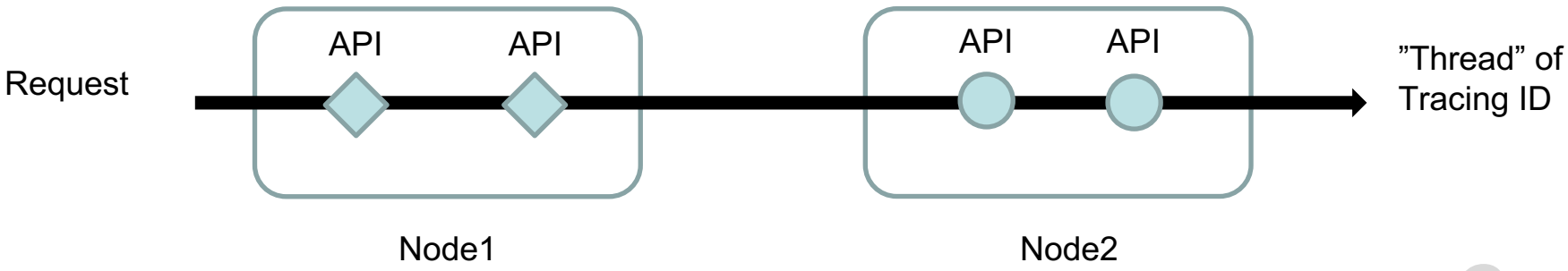
Components of Kubernetes

- Containers
- Control Plane
  - The decision making part of Kubernetes which decides what has to be done with each containers according to its description
- Data Plane
  - This is the part which enforces all of control plane's decisions
  - Allows applications to remain agnostic to their surroundings
- e.g: Post Service

Tracing

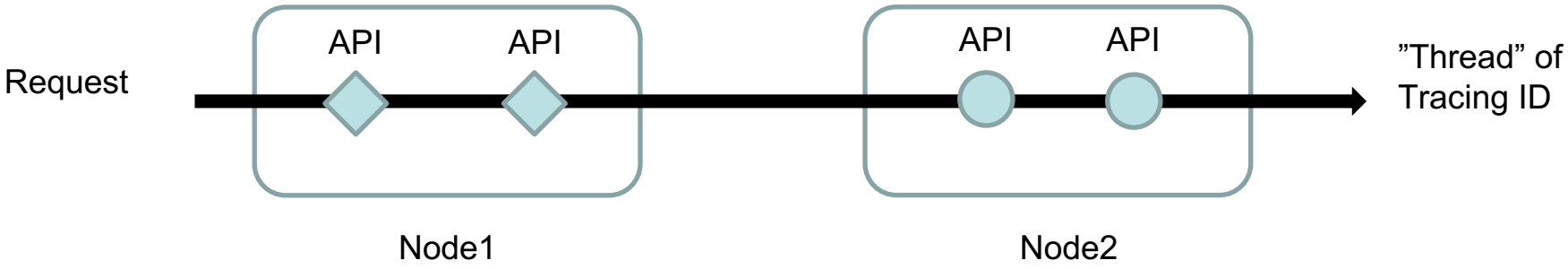
- To understand the performance/correctness of deploying distributed applications, many parts e.g application, networks, dataplane, control plane etc have to be understood.

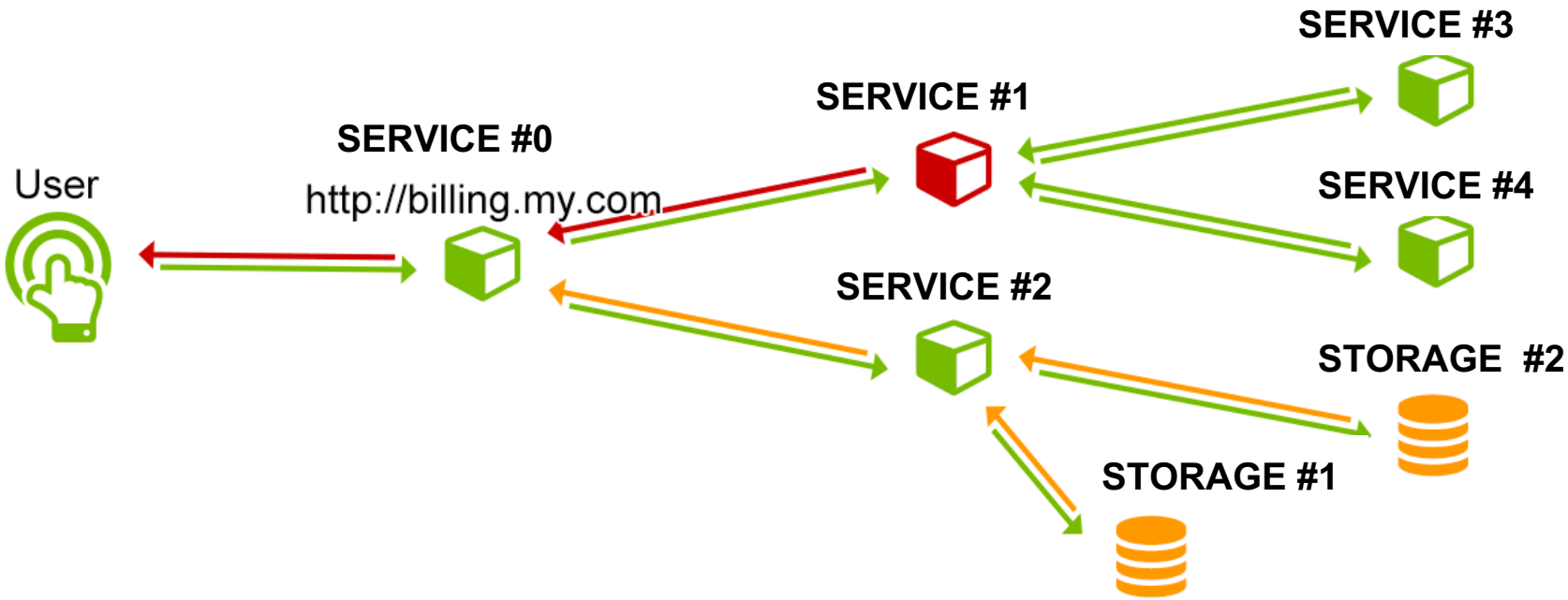
- Tracing: Trace events labeled by a unique ID of a request to know apps' behavior
  - Tracing ID is a local variable generated by an app
  - Propagation of request IDs tied to a specific request



Tracing

- Three major components of Tracing
  - Trace points
  - Tracing ID propagation
  - Huge repository storage to store tracing information




SERVICE #3
SERVICE #1
SERVICE #0
User
http://billing.my.com
SERVICE #4
SERVICE #2
STORAGE #2
STORAGE #1

End to End Tracing

- Tracing is determining the behaviour of each application/container. But what if it is not the application/container which has a latency issue?

- End to End (e2e) Tracing is to follow the execution of requests infrastructure along the entirety of its "PATH" of propagation (including its dataplane), to provide detailed tracing for capacity planning and performance analysis

Vision & Goals

- End-to-end tracing for app behavior on Kubernetes
  - Trace points (start time, stop time, annotations for remote procedure call)
  - RPC IDs
- Identify issues (bottle necks, latency issues etc.)
- Add in trace points to “data plane” features
  - Ingress - An API object that manages external access to the services in a cluster, typically HTTP.
  - Services - An abstraction which defines a logical set of **Pods** and a policy by which to access them - sometimes called a micro-service.

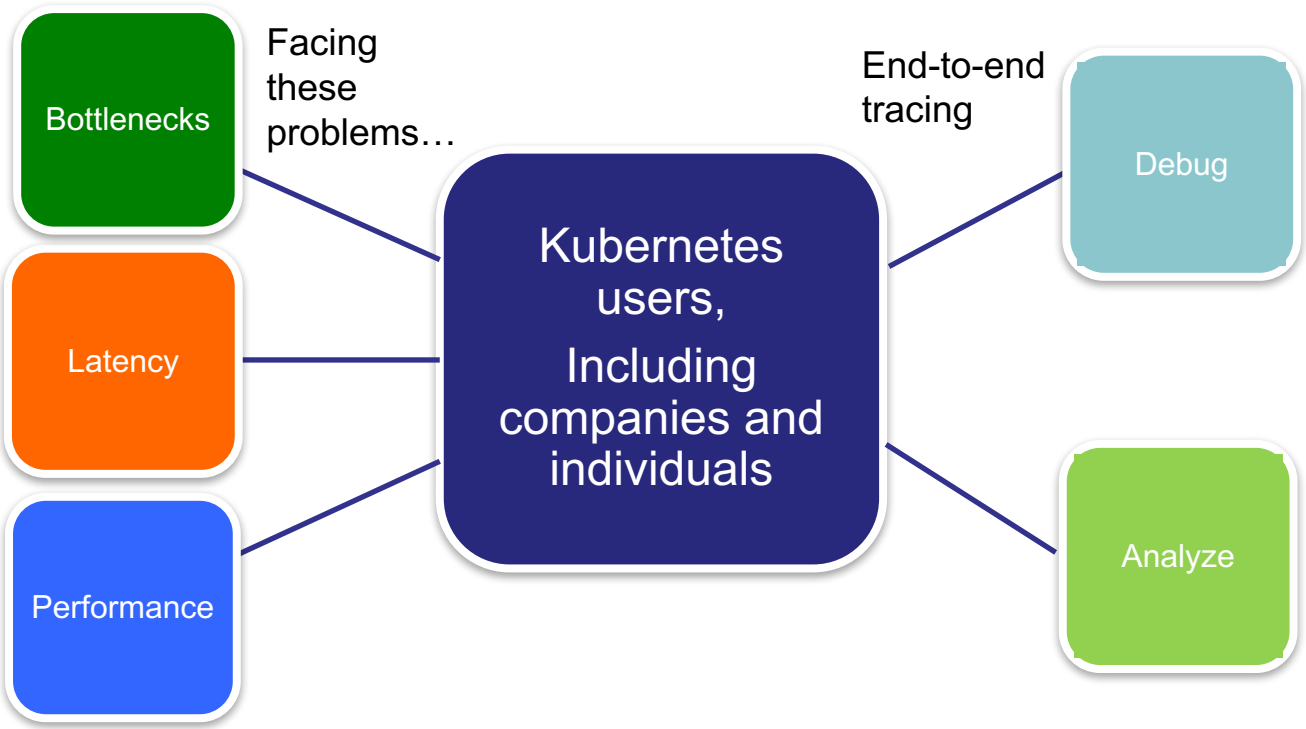
Scope & Features

- Instrument the services section of Kubernetes with trace points
- Instrument the ingress features of Kubernetes with trace points
- Focus on data-plane portion

User Stories

- A developer would be able to debug and analyze their deployed distributed applications
- Kubernetes developers can look into which requests are present in a particular data plane at some time T.
- Performance of a node can be determined
- Cloud providers and data centers

Our Users(This part is not sure)



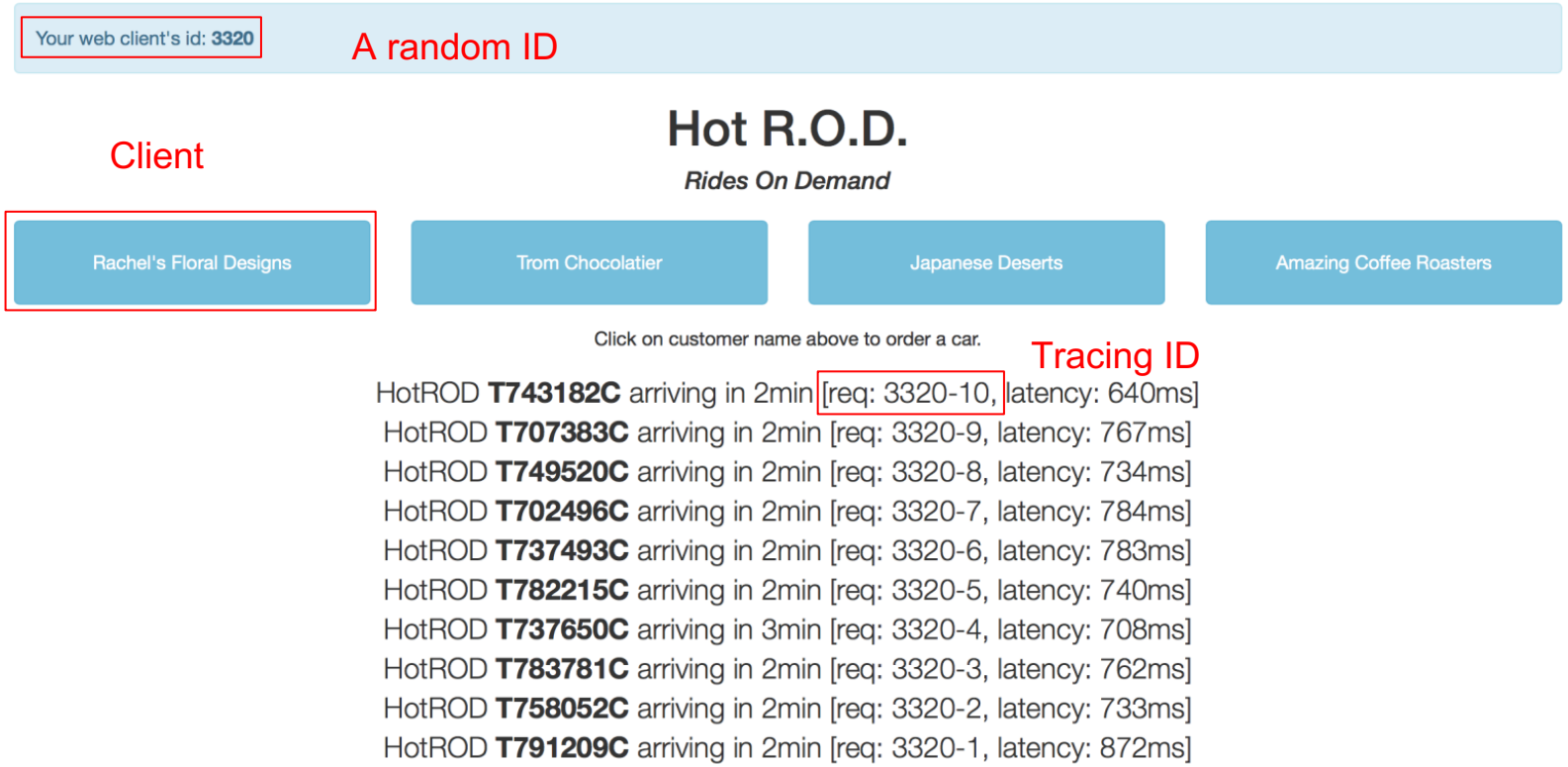
Challenges

- Tracing ID is from an app inside a container, how to let it propagate between containers and nodes?
- How to make tracing events visible and record them?

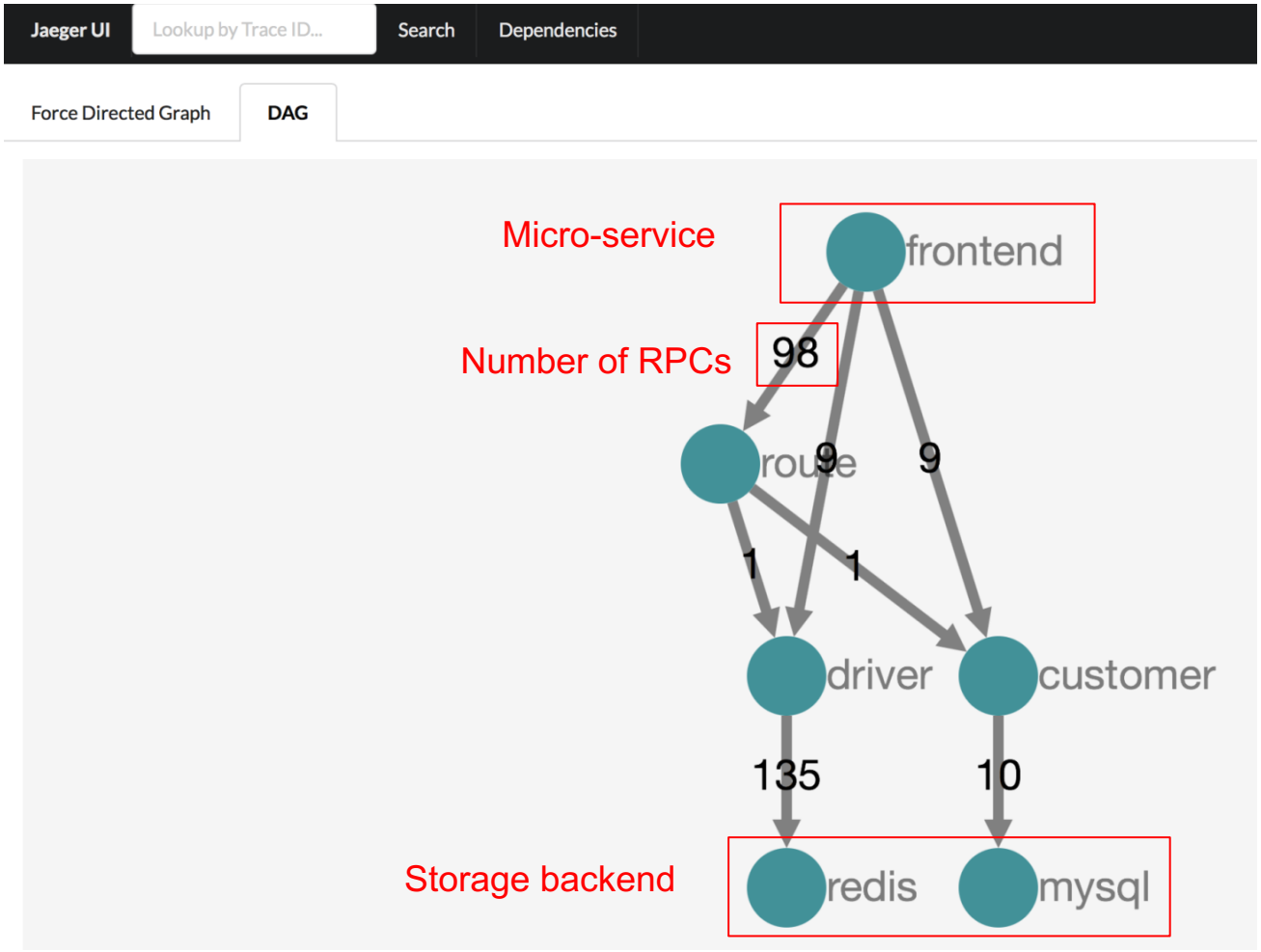
Example

- Jaeger: a distributed tracing system used for monitoring microservices-based distributed systems donated by Uber.
- OpenTracing: A new, open standard for instrumenting applications and OSS packages for distributed tracing and monitoring.
- HotRod: a sample application that can find out drivers nearby.

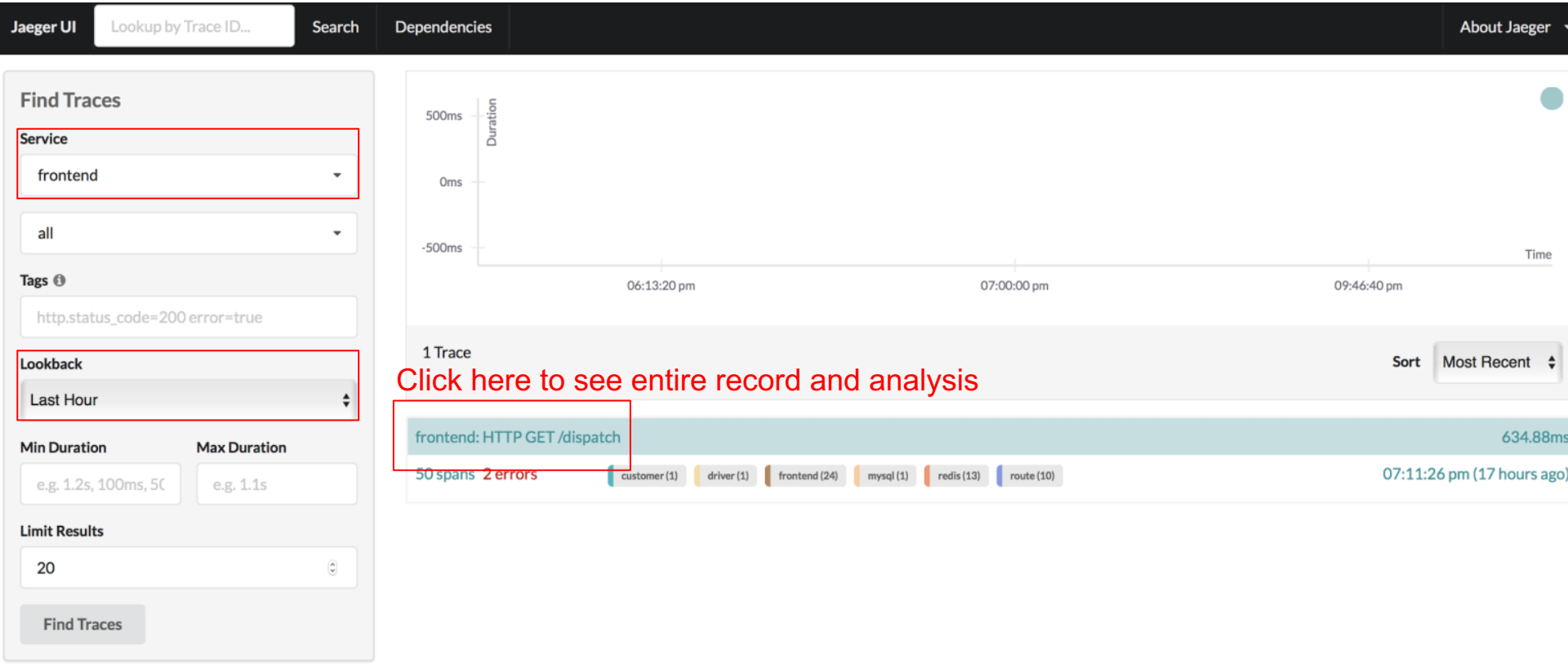
Example


Your web client's id: 3320
A random ID
Client
Hot R.O.D.
Rides On Demand
Rachel's Floral Designs
Trom Chocolatier
Japanese Deserts
Amazing Coffee Roasters
Click on customer name above to order a car.
Tracing ID
HotROD T743182C arriving in 2min [req: 3320-10, latency: 640ms]
HotROD T707383C arriving in 2min [req: 3320-9, latency: 767ms]
HotROD T749520C arriving in 2min [req: 3320-8, latency: 734ms]
HotROD T702496C arriving in 2min [req: 3320-7, latency: 784ms]
HotROD T737493C arriving in 2min [req: 3320-6, latency: 783ms]
HotROD T782215C arriving in 2min [req: 3320-5, latency: 740ms]
HotROD T737650C arriving in 3min [req: 3320-4, latency: 708ms]
HotROD T783781C arriving in 2min [req: 3320-3, latency: 762ms]
HotROD T758052C arriving in 2min [req: 3320-2, latency: 733ms]
HotROD T791209C arriving in 2min [req: 3320-1, latency: 872ms]

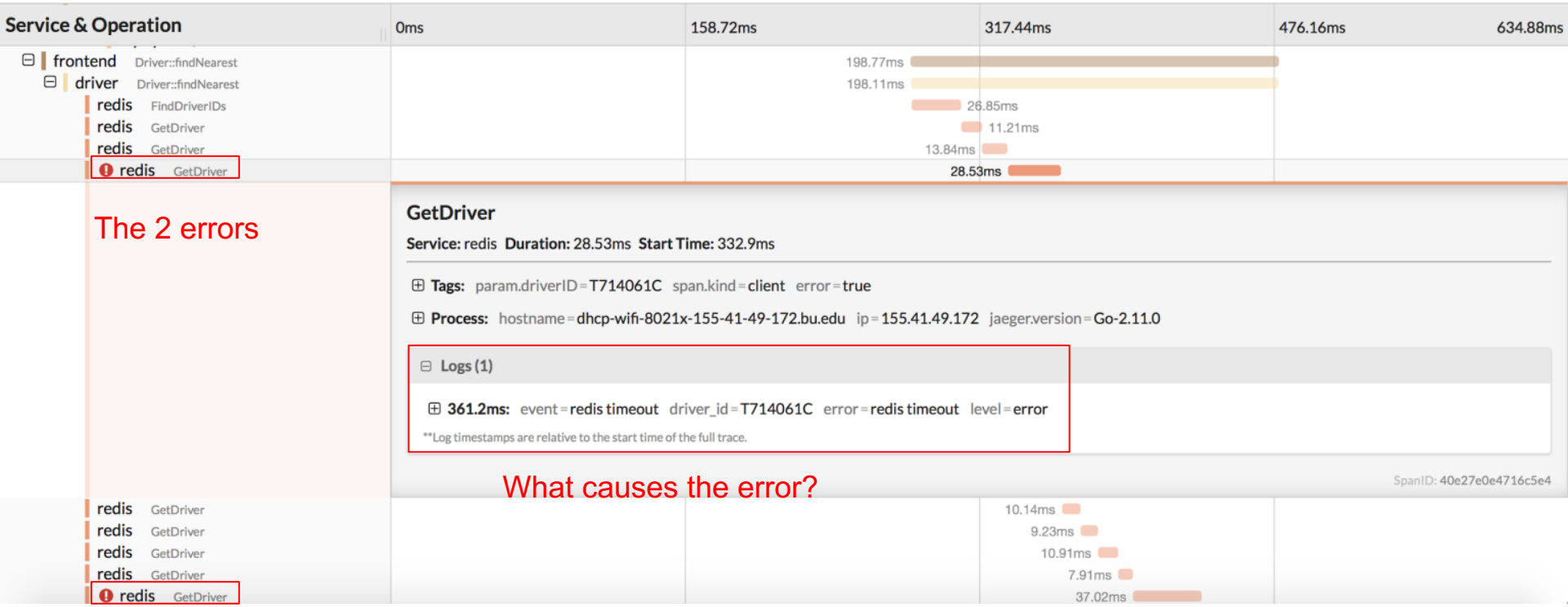
Example


Jaeger UI
Lookup by Trace ID...
Search
Dependencies
Force Directed Graph
DAG
Micro-service
frontend
Number of RPCs
98
route
9
9
1
1
driver
customer
135
10
Storage backend
redis
mysql

Example


Jaeger UI
Lookup by Trace ID...
Search
Dependencies
About Jaeger
Find Traces
Service
frontend
all
Tags
http.status_code=200 error=true
Lookback
Last Hour
Min Duration
e.g. 1.2s, 100ms, 5(
Max Duration
e.g. 1.1s
Limit Results
20
Find Traces
500ms
0ms
-500ms
Duration
Time
06:13:20 pm
07:00:00 pm
09:46:40 pm
1 Trace
Sort
Most Recent
Click here to see entire record and analysis
frontend: HTTP GET /dispatch
634.88ms
50 spans 2 errors
customer (1)
driver (1)
frontend (24)
mysql (1)
redis (13)
route (10)
07:11:26 pm (17 hours ago)

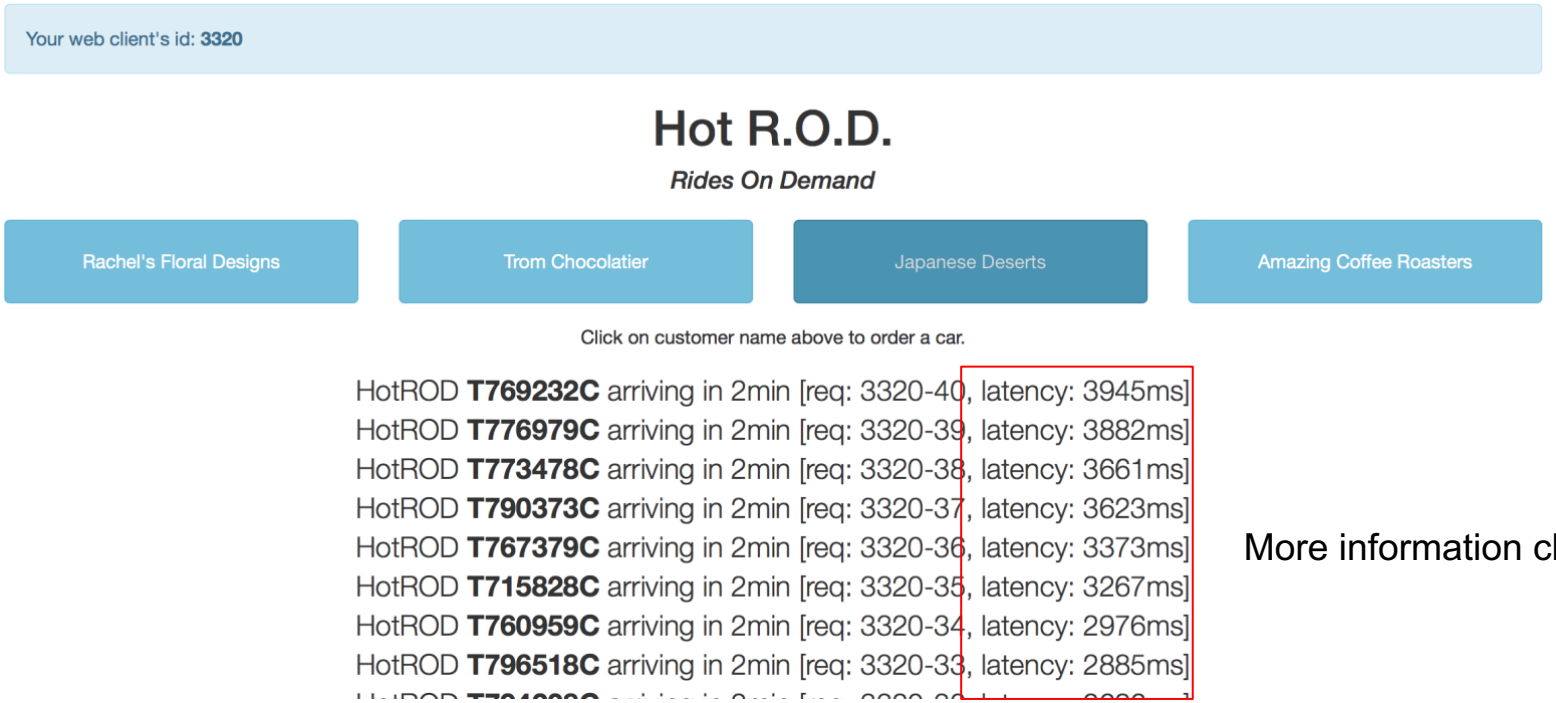
Example


Service & Operation
0ms
158.72ms
317.44ms
476.16ms
634.88ms
frontend Driver::findNearest
driver Driver::findNearest
redis FindDriverIDs
redis GetDriver
redis GetDriver
redis GetDriver
198.77ms
198.11ms
26.85ms
11.21ms
13.84ms
28.53ms
The 2 errors
GetDriver
Service: redis Duration: 28.53ms Start Time: 332.9ms
Tags: param.driverID = T714061C span.kind = client error = true
Process: hostname = dhcp-wifi-8021x-155-41-49-172.bu.edu ip = 155.41.49.172 jaeger.version = Go-2.11.0
Logs (1)
361.2ms: event = redis timeout driver_id = T714061C error = redis timeout level = error
**Log timestamps are relative to the start time of the full trace.
What causes the error?
SpanID: 40e27e0e4716c5e4
redis GetDriver
redis GetDriver
redis GetDriver
redis GetDriver
redis GetDriver
10.14ms
9.23ms
10.91ms
7.91ms
37.02ms

Example

Jaeger can detect latency and bottlenecks and find out how to deal with them.

Let's go to the HotROD UI and click on one of the buttons repeatedly (and quickly).we can see, the more requests are being processed concurrently, the longer it takes for the backend to respond.



More information click here

Acceptance Criteria

Propagating a request through the data plane of Kubernetes while making Kubernetes knowledgeable about the metadata being propagated along with the request.

Stretching the capabilities of the already existing tracing systems to the network namespace layer of Kubernetes which could prove to cause a substantial increase in latency and reduction in performance.

MVP(minimum value product)

- Adding trace points to Kubernetes on TWO data paths of a distributed application in parallel and then have them show up in Jaeger.

Release Planning

Short Term Goals:

- Get Kubernetes on our machines
- Tutorials on Kubernetes & tracing
- Containerize Hotrod using a container providing platform. e.g docker.
- Implement Jaeger in this containerized Hotrod application as backend to look up the traces and trace points.
- Study up on Kubernetes dataplane subsystems e.g HAProxy.

THANK YOU