Red Hat Summit

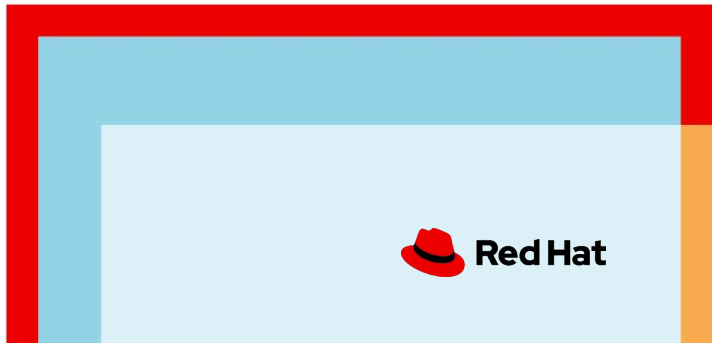
Connect

Distributed Tracing with Tempo and OpenTelemetry Auto-Instrumentation

Red Hat

Andreas Gerstmayr

Senior Software Engineer
Red Hat



Agenda

- Introduction to Distributed Tracing
- Manual and Automatic Instrumentation
- New Distributed Tracing Stack on OpenShift
- Example

Distributed Tracing

Red Hat

What is distributed tracing?

- Distributed Tracing records the execution of individual requests in distributed systems (through proxies, microservices, databases, etc.)
- a trace is a data/execution path through the system and contains one or more spans
- a span represents a single unit of work, with an operation name, start, duration, and optionally custom attributes and logs

JAEGER UI | Search | Compare | System Architecture | Monitor | Lookup by Trace ID... | About Jaeger

frontend: HTTP GET /dispatch 5967a56

Find... | Trace Timeline

Trace Start **October 4 2023, 13:42:05.130** | Duration **1.41s** | Services **6** | Depth **5** | Total Spans **50**

0µs | 352.9ms | 705.79ms | 1.06s | 1.41s

Service & Operation	0µs	352.9ms	705.79ms	1.06s	1.41s
frontend HTTP GET /dispatch					
frontend HTTP GET: /customer					1.07s
frontend HTTP GET					1.07s
customer HTTP GET /customer					1.07s
mysql SQL SELECT					1.07s
frontend /driver.DriverService/FindNearest				163.95ms	
driver /driver.DriverService/FindNearest				163.53ms	
redis FindDriverIDs				7.22ms	
redis GetDriver				9.19ms	
redis GetDriver				10.17ms	
redis GetDriver				33.12ms	
redis GetDriver				10.13ms	
redis GetDriver				6.16ms	
redis GetDriver				13.18ms	
redis GetDriver				8.15ms	
redis GetDriver				27.49ms	
redis GetDriver				9.2ms	
redis GetDriver				9.08ms	
redis GetDriver				11.13ms	
redis GetDriver				9.07ms	
frontend HTTP GET: /route				41.78ms	
frontend HTTP GET				41.77ms	
route HTTP GET /route				41.37ms	

Red Hat

Why distributed tracing?

- reduce mean time to detect (MTTD) and mean time to remediate (MTTR)
- optimize performance
- understand how data flows through a system

OpenTelemetry

- Collection of APIs and SDKs, Data Model and semantic conventions (“k8s.pod.name”)
- Protocol (OTLP)
- Collector (receive, process and export telemetry data)

Instrumentation

Red Hat

Manual Instrumentation with OTEL SDK

```go
func rolldice(ctx context.Context) int {
    ctx, span := tracer.Start(ctx, "rolldice")
    defer span.End()

    roll := 1 + rand.Intn(6)
    span.SetAttributes(attribute.Int("roll.value", roll))
    return roll
}

func rolldiceHandler(w http.ResponseWriter, r *http.Request) {
    ctx, span := tracer.Start(r.Context(), "rolldiceHandler")
    defer span.End()

    roll1 := rolldice(ctx)
    roll2 := rolldice(ctx)
    roll3 := rolldice(ctx)

    io.WriteString(w, fmt.Sprintf("%d,%d,%d\n", roll1, roll2, roll3))
}
```

Manual Instrumentation with OTEL wrappers

```
import io.opentelemetry.instrumentation.jdbc.datasource.OpenTelemetryDataSource;

@Configuration
public class DataSourceConfig {

  @Bean
  public DataSource dataSource() {
    BasicDataSource dataSource = new BasicDataSource();
    dataSource.setDriverClassName("org.postgresql.Driver");
    dataSource.setUrl("jdbc:postgresql://127.0.0.1:5432/example");
    dataSource.setUsername("postgres");
    dataSource.setPassword("root");
    return new OpenTelemetryDataSource(dataSource);
  }

}
```

Manual Instrumentation with OTEL wrappers

```
KafkaConsumer<String, String> consumer = new KafkaConsumer<>(properties);
KafkaProducer<String, String> producer = new KafkaProducer<>(properties);

KafkaTelemetry telemetry = KafkaTelemetry.create(GlobalOpenTelemetry.get());
Producer<String, String> tracingProducer = telemetry.wrap(producer);
Consumer<String, String> tracingConsumer = telemetry.wrap(consumer);
```

Automatic Instrumentation with OTEL

- Captures telemetry data from popular libraries and frameworks
- Red Hat OpenShift distributed tracing data collection supports injecting auto-instrumentation agents for the following languages/applications (Dev Preview):
  - Java
  - .NET
  - NodeJS
  - Go
  - Python
  - Apache HTTPD

Distributed Tracing on OpenShift

Red Hat

New Distributed Tracing Stack on OpenShift

- **Instrumentation:** OpenTelemetry, Jaeger, OpenCensus or Zipkin
- **Collection:** OpenTelemetry Collector (managed by OpenShift distributed tracing data collection operator)
- **Storage:** Grafana Tempo (managed by Tempo Operator), object storage
- **Visualization:** Jaeger UI (managed by Tempo Operator)

Tempo Operator

- Deployment of Grafana Tempo instances
- Authentication and Authorization, Multitenancy
- Jaeger UI
- Managed upgrades

Red Hat OpenShift distributed tracing data collection

- Deployment of OpenTelemetry Collector instances as sidecar, daemon set or regular deployment
- Auto-Instrumentation injection
- Managed upgrades

OpenShift Service Mesh

- OpenShift Service Mesh supports creating spans of the interactions between services in the service mesh
- Services must propagate trace context between inbound and outbound requests in order to correlate spans

Example

Installed Operators

Installed Operators are represented by ClusterServiceVersions within this Namespace. For more information, see the Understanding Operators

Name ▾ Search by name... /

Name	Managed Namespaces	Status
Red Hat OpenShift distributed tracing data collection 0.81.0-2 provided by Red Hat	All Namespaces	Succeeded Up to date
Tempo Operator 0.3.0-2 provided by Red Hat	All Namespaces	Succeeded Up to date

Red Hat

Storage Configuration for Tempo

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: tempo-storage
type: Opaque
stringData:
  endpoint: http://minio:9000
  bucket: tempo
  access_key_id: tempo
  access_key_secret: supersecret
```

Tempo Deployment

```yaml
apiVersion: tempo.grafana.com/v1alpha1
kind:  TempoStack
metadata:
  name: prod
spec:
  storage:
    secret:
      name: tempo-storage
      type: s3
  storageSize: 1Gi
  template:
    queryFrontend:
      jaegerQuery:
        enabled: true
        ingress:
          type: route
```


Red Hat

OpenTelemetry Collector Deployment

```yaml
apiVersion: opentelemetry.io/v1alpha1
kind: OpenTelemetryCollector
metadata:
  name: otel
spec:
  config: |
    receivers:
      otlp:
        protocols:
          grpc:

    exporters:
      otlp:
        endpoint: tempo-prod-distributor:4317
        tls:
          insecure: true

    processors:
      batch:

    service:
      pipelines:
        traces:
          receivers: [otlp]
          processors: [batch]
          exporters: [otlp]
```


Red Hat

Auto-Instrumentation

```yaml
apiVersion: opentelemetry.io/v1alpha1
kind: Instrumentation
metadata:
  name: petclinic
spec:
  exporter:
    endpoint: http://otel-collector:4317
```


Red Hat

PetClinic Deployment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: petclinic
spec:
  selector:
    matchLabels:
      app: petclinic
  template:
    metadata:
      labels:
        app: petclinic
      annotations:
        instrumentation.opentelemetry.io/inject-java: petclinic
    spec:
      containers:
      - image: springcommunity/spring-framework-petclinic:6.0.3
        name: petclinic
```


Red Hat

spring petclinic HOME FIND OWNERS VETERINARIANS ERROR

New Pet

Owner
George Franklin
Name

Birth Date

Type

- bird
- cat
- dog
- hamster
- lizard

Add Pet

spring by Pivotal

Red Hat

JAEGER UI | Search | Compare | System Architecture | Monitor | Lookup by Trace ID... | About Jaeger

Search | Upload

Service (1)

petclinic

Operation (49)

all

- all
- CrashController.triggerException
- GET /
- GET /oups
- GET /owners
- GET /owners/{ownerId}

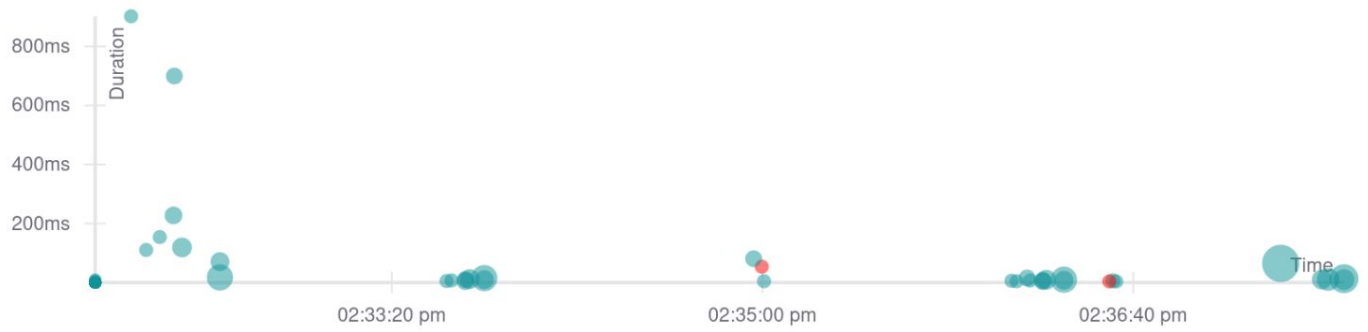
Max Duration: e.g. 1.2s, ...

Min Duration: e.g. 1.2s, ...

Limit Results

200

Find Traces



107 Traces

Sort: Most Recent | Download Results | Deep Dependency Graph

Compare traces by selecting result items

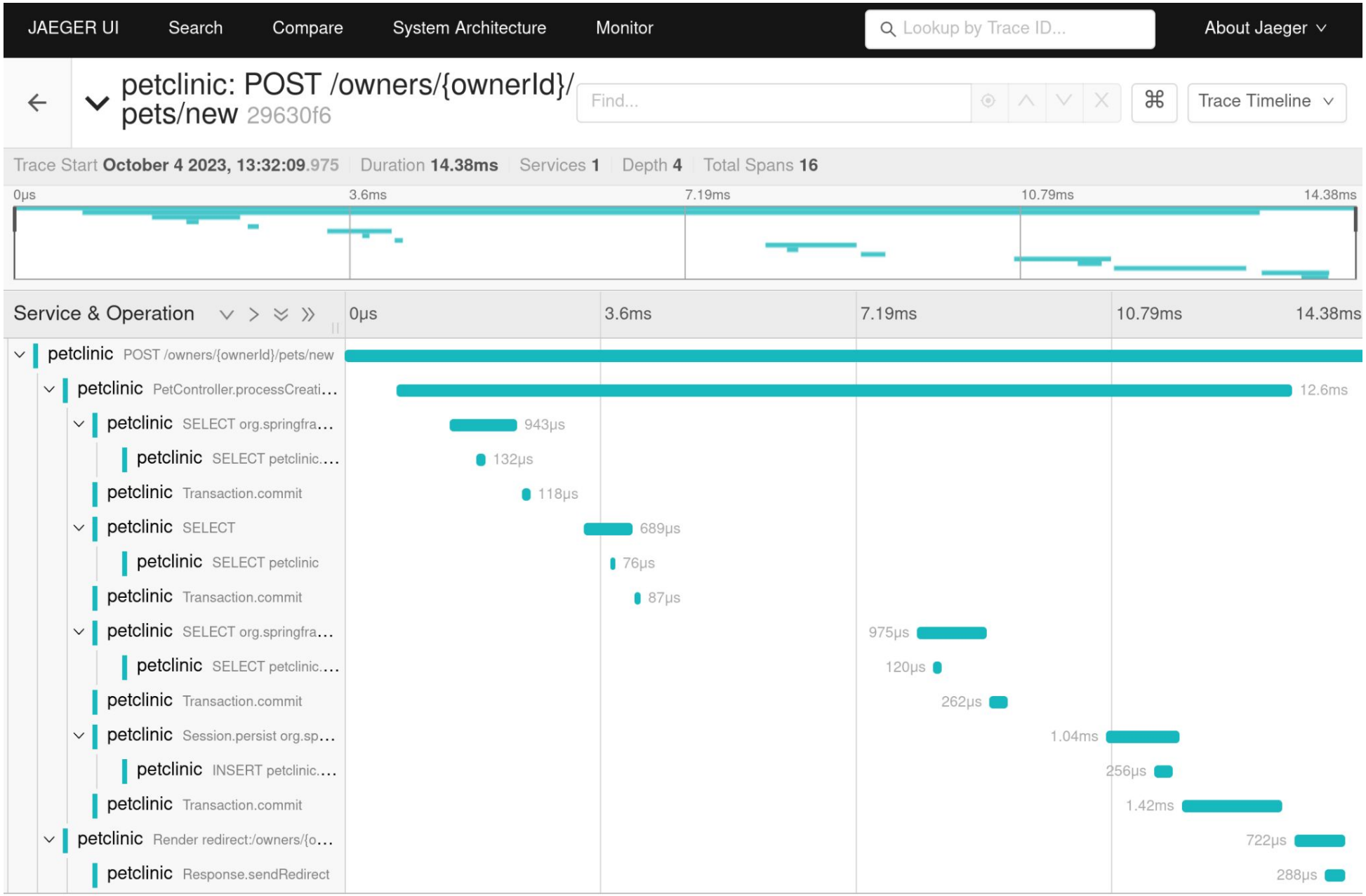
petclinic: GET /owners/{ownerId} a52d96c — 9.85ms

10 Spans | petclinic (10) | Today 2:37:36 pm | 4 minutes ago

petclinic: POST /owners/{ownerId}/pets/new 4c06eac — 13.17ms

19 Spans | petclinic (19) | Today 2:37:36 pm | 4 minutes ago

Red Hat


JAEGER UI
Search
Compare
System Architecture
Monitor
Lookup by Trace ID...
About Jaeger
petclinic: POST /owners/{ownerId}/pets/new 29630f6
Find...
Trace Timeline
Trace Start October 4 2023, 13:32:09.975
Duration 14.38ms
Services 1
Depth 4
Total Spans 16
0µs
3.6ms
7.19ms
10.79ms
14.38ms
Service & Operation
petclinic POST /owners/{ownerId}/pets/new
petclinic PetController.processCreati...
12.6ms
petclinic SELECT org.springfra...
943µs
petclinic SELECT petclinic....
132µs
petclinic Transaction.commit
118µs
petclinic SELECT
689µs
petclinic SELECT petclinic
76µs
petclinic Transaction.commit
87µs
petclinic SELECT org.springfra...
975µs
petclinic SELECT petclinic....
120µs
petclinic Transaction.commit
262µs
petclinic Session.persist org.sp...
1.04ms
petclinic INSERT petclinic....
256µs
petclinic Transaction.commit
1.42ms
petclinic Render redirect:/owners/{o...
722µs
petclinic Response.sendRedirect
288µs

Red Hat

JAEGER UI | Search | Compare | System Architecture | Monitor | Lookup by Trace ID... | About Jaeger

petclinic: POST /owners/{ownerId}/pets/new 29630f6

Find... | Trace Timeline

Trace Start **October 4 2023, 13:32:09.975** | Duration **14.38ms** | Services **1** | Depth **4** | Total Spans **16**

0µs | 3.6ms | 7.19ms | 10.79ms | 14.38ms

Service & Operation	0µs	3.6ms	7.19ms	10.79ms	14.38ms
petclinic POST /owners/{ownerId}/pets/new					
petclinic PetController.processCreati…	12.6ms				
petclinic SELECT org.springfra…	943µs				
petclinic SELECT petclinic.…	132µs				
petclinic Transaction.commit	118µs				
petclinic SELECT	689µs				
petclinic SELECT petclinic	76µs | petclinic::SELECT petclinic				

SELECT petclinic

Service: **petclinic** | Duration: **76µs** | Start Time: **3.74ms**

Tags

db.connection_string	`h2:mem:`
db.name	`petclinic`
db.operation	`SELECT`
db.statement	`select o1_0.id,o1_0.address,o1_0.city,o1_0.first_name,o1_0.last_name,p1_0.owner_id,p1_0.id,p1_0.birth_date,p1_0.name,p1_0.type_id,o1_0.telephone from owners o1_0 left join pets p1_0 on o1_0.id=p1_0.owner_id where o1_0.id=?`

Red Hat

Java Auto-Instrumentation: Behind the Scenes

The OpenShift distributed tracing data collection operator performs the following modifications to a pod:

- It attaches a new emptyDir volume
- It adds a new init container, which copies javaagent.jar to this volume
- This volume is mounted in the container of the application
- The JAVA_TOOL_OPTIONS environment variable is modified to load javaagent.jar

Q & A

Red Hat

Red Hat Summit

Connect

Thank you

linkedin.com/company/red-hat

facebook.com/redhatinc

youtube.com/user/RedHatVideos

twitter.com/RedHat

Red Hat