Advanced Application Management Using Red Hat OpenShift Service Mesh

Observability

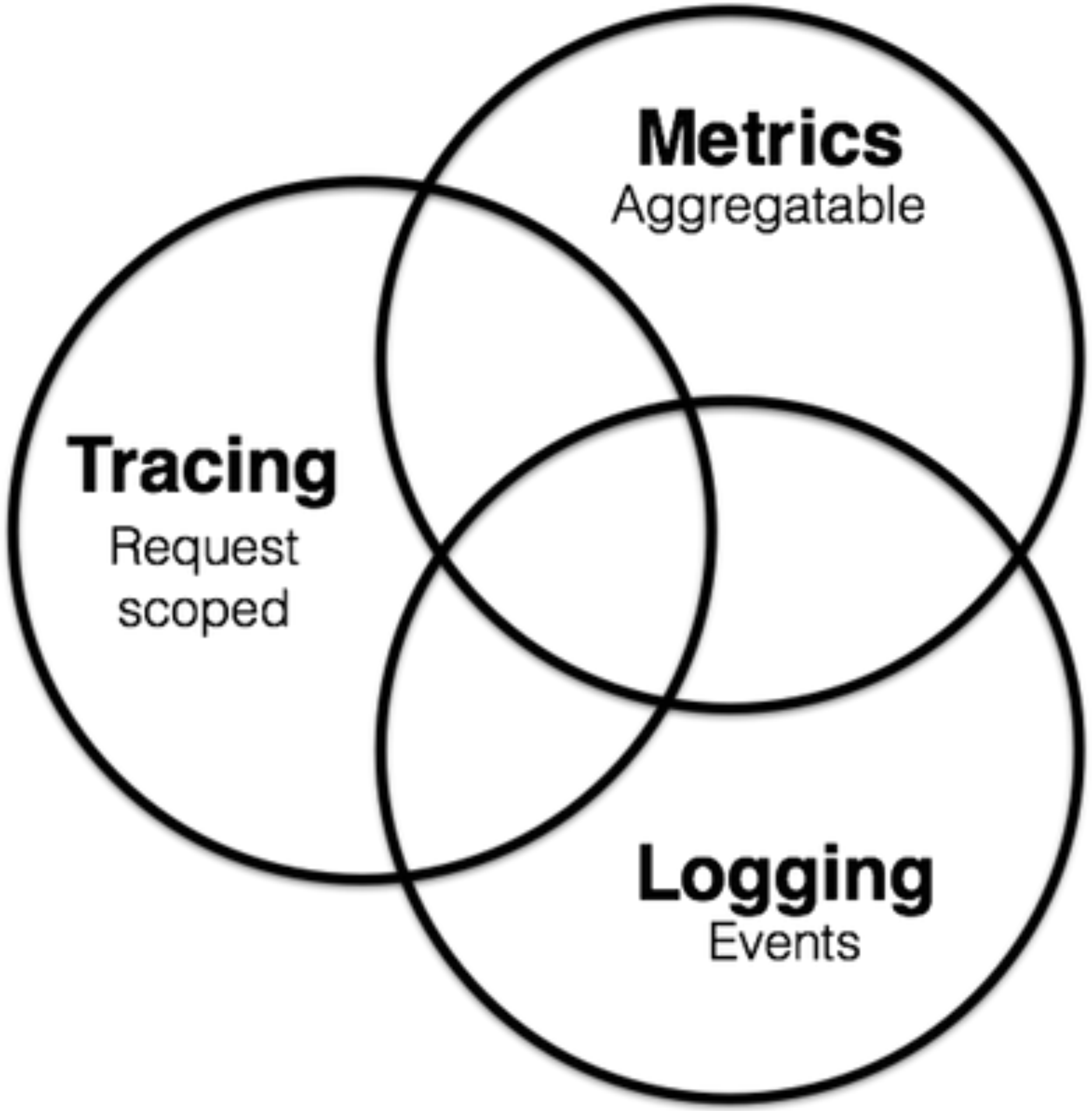
Module Topics

- Observability
- Logging
- Distributed Tracing
- Monitoring
- Visualization

Observability

- Ability to understand how system behaves in production
- System is observable if current state can be understood from outside
- Important characteristic of cloud-native distributed systems, like availability and scalability
- Techniques:
  - Logging
  - Tracing
  - Monitoring
  - Visualization
- Goal: Gain understanding of application services to help understand, operate, maintain, evolve system

Observability



Source: http://peter.bourgon.org/blog/2017/02/21/metrics-tracing-and-logging.html

Logging

- Messages representing events
- Aggregation and correlation:
    - By process: Thread name

```
Thread-1 2018-09-03T15:52:54+02:00 Request started
Thread-2 2018-09-03T15:52:55+02:00 Charging credit card x321
Thread-1 2018-09-03T15:52:55+02:00 Order submitted
Thread-1 2018-09-03T15:52:56+02:00 Charging credit card x123
Thread-1 2018-09-03T15:52:57+02:00 Changing order status
Thread-1 2018-09-03T15:52:58+02:00 Dispatching event to inventory
Thread-1 2018-09-03T15:52:59+02:00 Request finished
```

    - By transaction: Correlation ID

```
abc123 Order     2018-09-03T15:52:58+02:00 Dispatching event to inventory
def456 Order     2018-09-03T15:52:58+02:00 Dispatching event to inventory
abc123 Inventory 2018-09-03T15:52:59+02:00 Received `order-submitted` event
abc123 Inventory 2018-09-03T15:53:00+02:00 Checking inventory status
abc123 Inventory 2018-09-03T15:53:01+02:00 Updating inventory
abc123 Inventory 2018-09-03T15:53:02+02:00 Preparing order manifest
```

Distributed Tracing

- Story of request across services
  - Which services were touched, when, in which order, etc.
- With context information: Routing info, version, tags, etc.
- Not replacement for logging
  - Process/thread-level details
  - Not all traces sampled

Distributed Tracing

Use cases

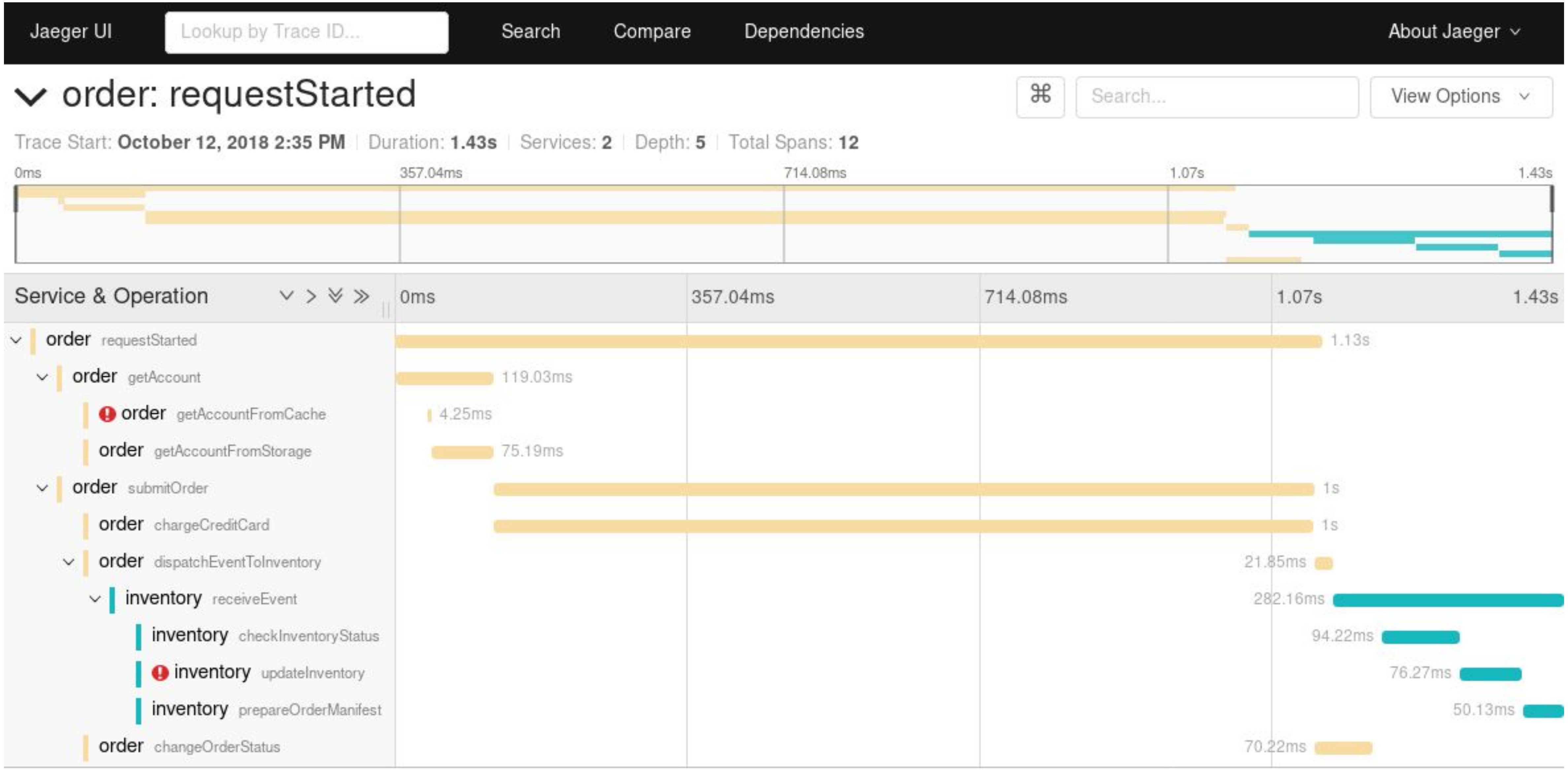
- Root cause analysis
    - Where did first failure happen?
    - Trace contains error code and message
- Performance optimization
    - Is a service behaving badly?
    - Impact of performance optimization
- Service dependency visualization

Distributed Tracing

Concepts

- Measures unit of work
- Data stored in *span*
- References other spans
  - Causality
- Context propagation
  - Similar to correlation identifiers
  - Typically autogenerated
- Additional information: Tags, baggage

Distributed Tracing Visualization



order: requestStarted

Trace Start: **October 12, 2018 2:35 PM** | Duration: **1.43s** | Services: **2** | Depth: **5** | Total Spans: **12**

Service & Operation | 0ms | 357.04ms | 714.08ms | 1.07s | 1.43s

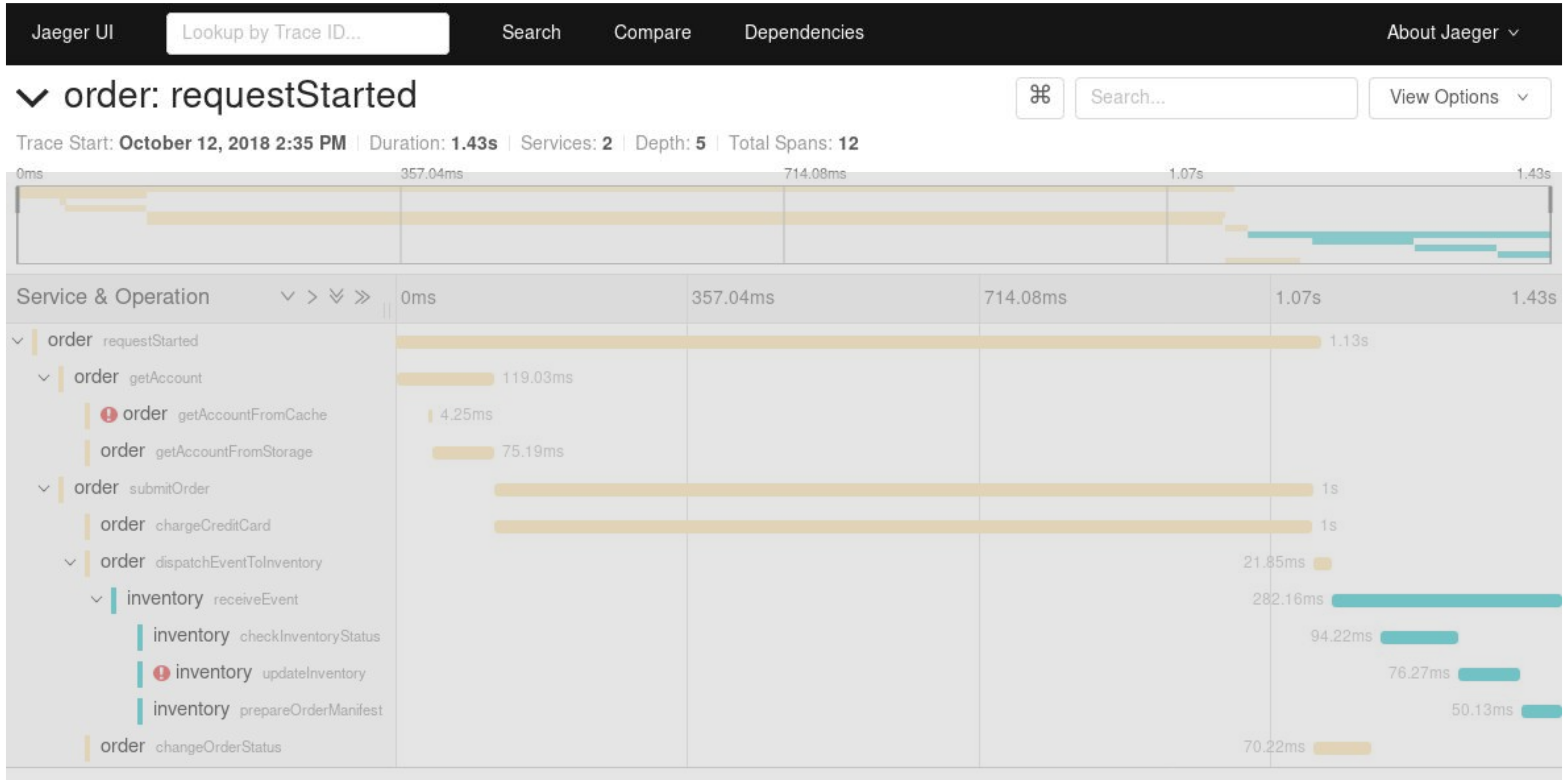
- order requestStarted 1.13s
  - order getAccount 119.03ms
    - order getAccountFromCache 4.25ms
    - order getAccountFromStorage 75.19ms
  - order submitOrder 1s
    - order chargeCreditCard 1s
    - order dispatchEventToInventory 21.85ms
      - inventory receiveEvent 282.16ms
        - inventory checkInventoryStatus 94.22ms
        - inventory updateInventory 76.27ms
        - inventory prepareOrderManifest 50.13ms
    - order changeOrderStatus 70.22ms

Distributed Tracing Visualization

Request Summary

order: requestStarted

Trace Start: **October 12, 2018 2:35 PM** | Duration: **1.43s** | Services: **2** | Depth: **5** | Total Spans: **12**



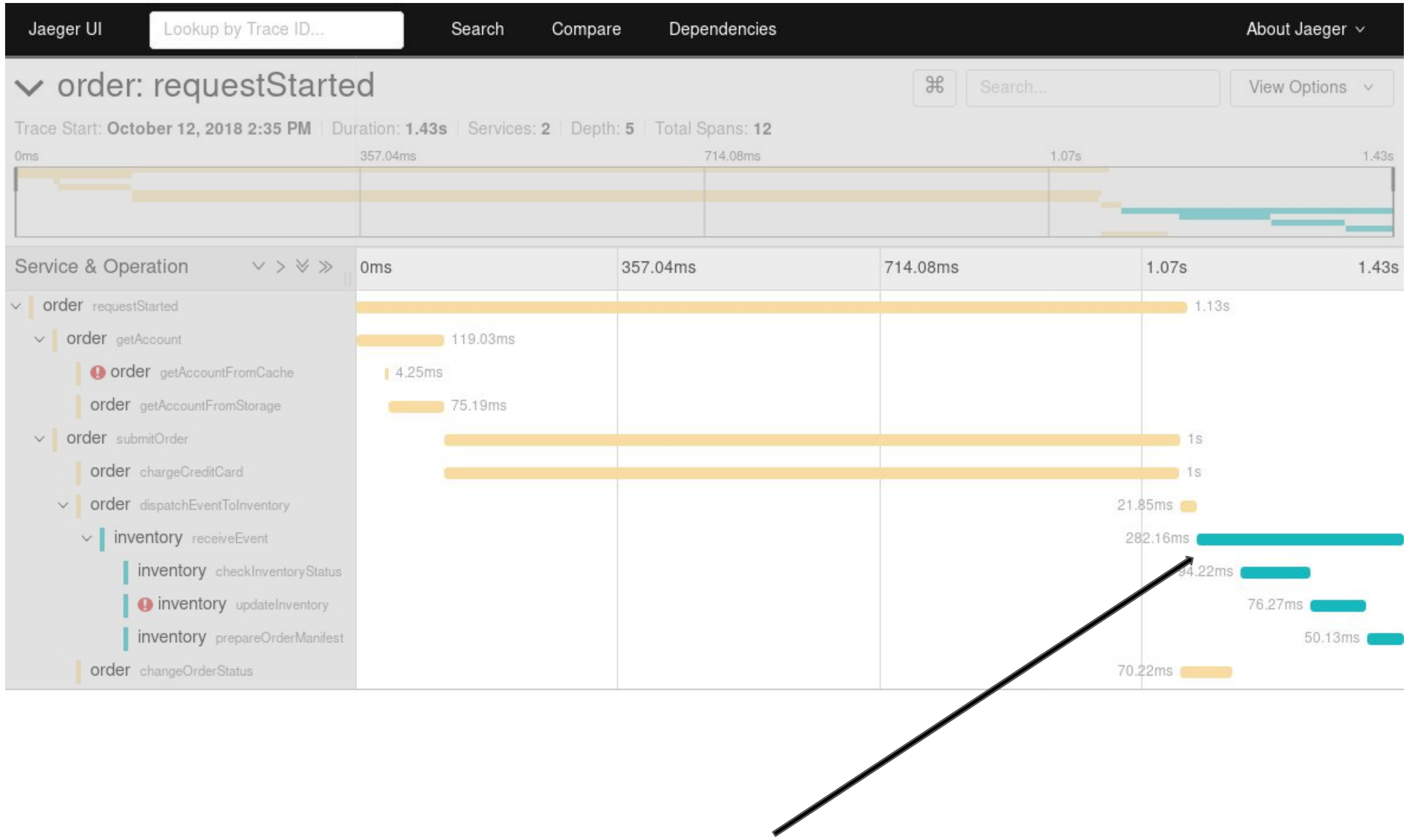
order: requestStarted

Trace Start: **October 12, 2018 2:35 PM** | Duration: **1.43s** | Services: **2** | Depth: **5** | Total Spans: **12**

Service & Operation	Duration
order requestStarted	1.13s
order getAccount	119.03ms
order getAccountFromCache	4.25ms
order getAccountFromStorage	75.19ms
order submitOrder	1s
order chargeCreditCard	1s
order dispatchEventToInventory	21.85ms
inventory receiveEvent	282.16ms
inventory checkInventoryStatus	94.22ms
inventory updateInventory	76.27ms
inventory prepareOrderManifest	50.13ms
order changeOrderStatus	70.22ms

Distributed Tracing Visualization

Process Boundary



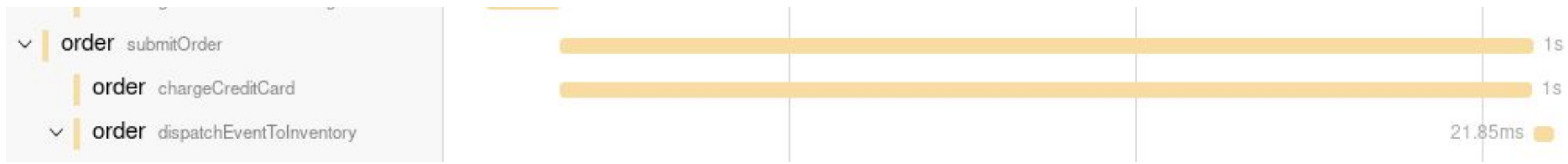
order: requestStarted

Trace Start: **October 12, 2018 2:35 PM** | Duration: **1.43s** | Services: **2** | Depth: **5** | Total Spans: **12**

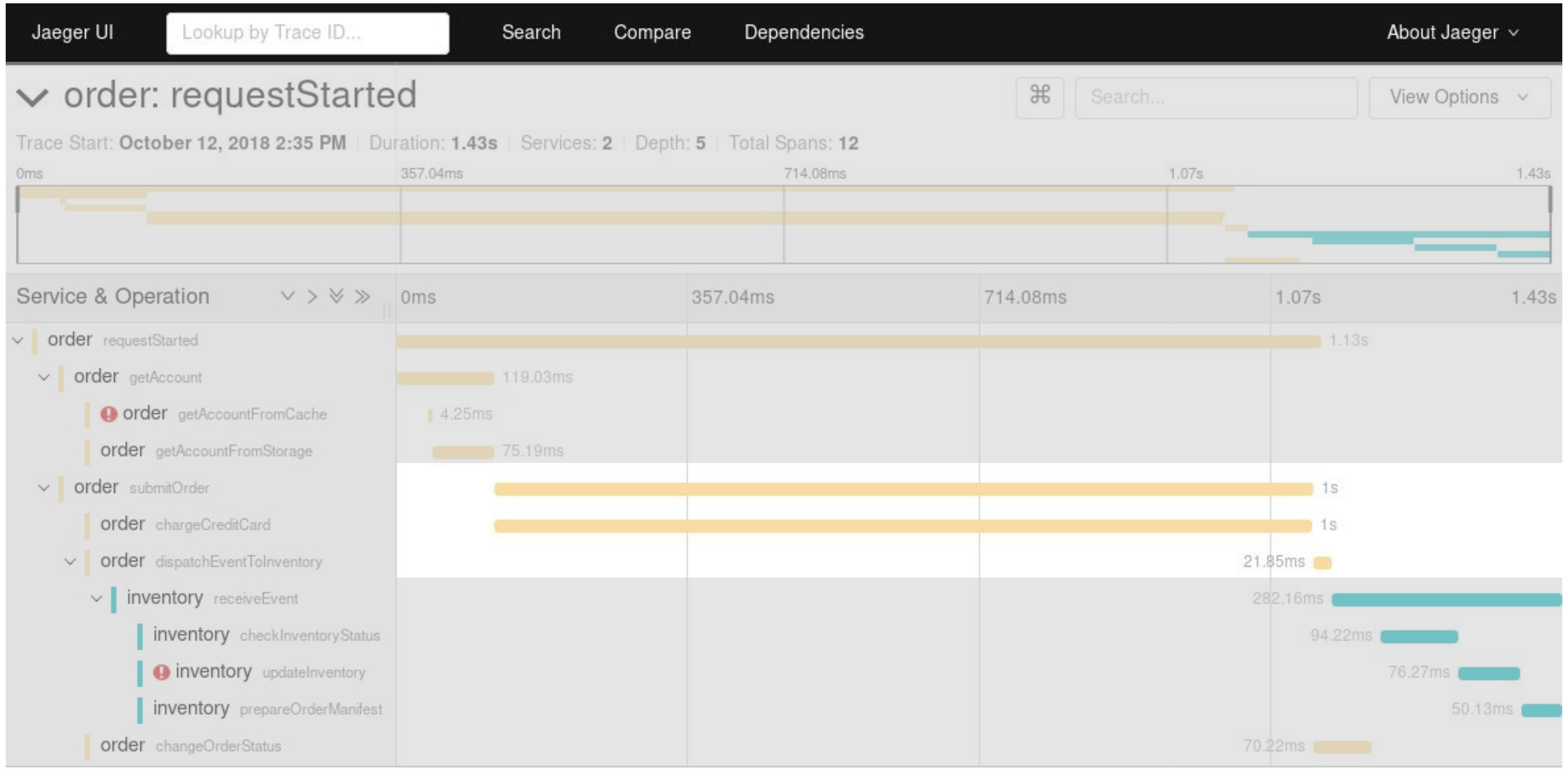
Service & Operation	
order requestStarted	1.13s
order getAccount	119.03ms
! order getAccountFromCache	4.25ms
order getAccountFromStorage	75.19ms
order submitOrder	1s
order chargeCreditCard	1s
order dispatchEventToInventory	21.85ms
inventory receiveEvent	282.16ms
inventory checkInventoryStatus	94.22ms
! inventory updateInventory	76.27ms
inventory prepareOrderManifest	50.13ms
order changeOrderStatus	70.22ms

Distributed Tracing Visualization

Performance Bottleneck



order	submitOrder	1s
order	chargeCreditCard	1s
order	dispatchEventToInventory	21.85ms



order: requestStarted

Trace Start: **October 12, 2018 2:35 PM** | Duration: **1.43s** | Services: **2** | Depth: **5** | Total Spans: **12**

Service & Operation		
order	requestStarted	1.13s
order	getAccount	119.03ms
order	getAccountFromCache	4.25ms
order	getAccountFromStorage	75.19ms
order	submitOrder	1s
order	chargeCreditCard	1s
order	dispatchEventToInventory	21.85ms
inventory	receiveEvent	282.16ms
inventory	checkInventoryStatus	94.22ms
inventory	updateInventory	76.27ms
inventory	prepareOrderManifest	50.13ms
order	changeOrderStatus	70.22ms

Code Instrumentation

- Explicit
  - Included in source code
- Implicit
  - Span generation and propagation done automatically
  - Web frameworks, REST libraries, Spring Boot, JMS, etc.
- Span propagation
  - Mechanism depends on protocol
  - HTTP: Span context propagated to upstream services as HTTP headers

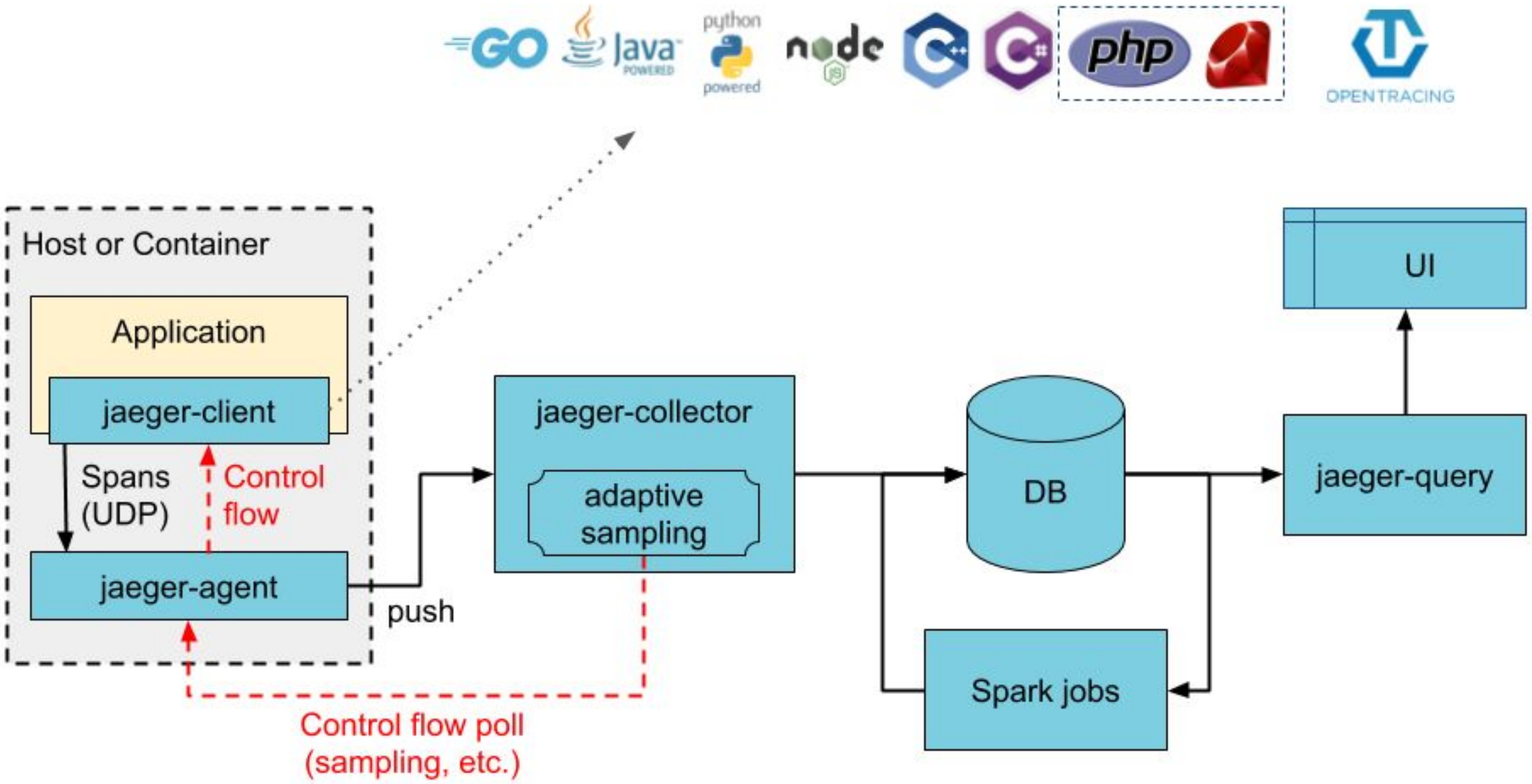
OpenTracing

- Open standard specification for distributed tracing
- Defines semantics: What is a trace, what is a span, etc.
- Instrumentation API
    - Official APIs for Java™, Go, Ruby
    - Can be used with compatible tracers—e.g., Zipkin, Jaeger
- Hosted at Cloud Native Computing Foundation
- Integrations
    - Frameworks, stack, platform
        - JAX-RS, JDBC, JMS, others
        - Spring Boot, MicroProfile, others
    - Infrastructure
        - Istio

Jaeger

- Concrete OpenTracing implementation
  - Native OpenTracing semantics
- Originated at Uber
- Back-end components
  - Agent, Collector, Query, UI
- Production-ready
  - Bare metal, Kubernetes, OpenShift

Jaeger Architecture



Service Mesh and Jaeger

- Jaeger optionally installed as part of OpenShift Service Mesh
  - All-in-one
  - Production Elasticsearch
- Envoy proxy responsible for generating initial trace headers
  - Uses Zipkin format headers.
  - Default sampling rate: 1%
- Envoy proxy sends tracing information directly to tracing back ends
- Application responsible for propagating trace context to upstream services
  - Envoy proxy not compatible with Jaeger agent sidecar
  - Applications must send traces directly to Jaeger collector
- Integration with other tracers (e.g. StackDriver) possible through Envoy WASM modules

Monitoring

- Gathering, processing, storing metric data about system
- Blackbox monitoring
  - Tests externally visible behavior as user sees it
  - HTTP endpoints, ICMP, overall response times
- Whitebox monitoring
  - Based on internal system metrics
  - CPU, memory, JVM, application-specific metrics
- Monitoring data typically stored as time series
- Requirements for monitoring data:
  - Aggregation
  - Mathematical processing and transformation
  - Examples: Averages, histograms

Monitoring Solutions for Cloud Native Apps

- Commercial, closed source
    - AppDynamics
    - New Relic
    - Dynatrace
- Open source
    - Zabbix
    - Heapster
    - Prometheus

Prometheus

- Open source monitoring solution
- Inspired by Google's *Borgmon* monitoring tool
- Graduated project of CNCF
  - Second after Kubernetes
- Inclusive monitoring: Blackbox/whitebox, client libraries
- Time series collection implemented via pull model over HTTP
- Multi-dimensional data model with time series data identified by metric name and key/value pairs
- Powerful query language
  - Multiply, add, aggregate, join, compute quantiles

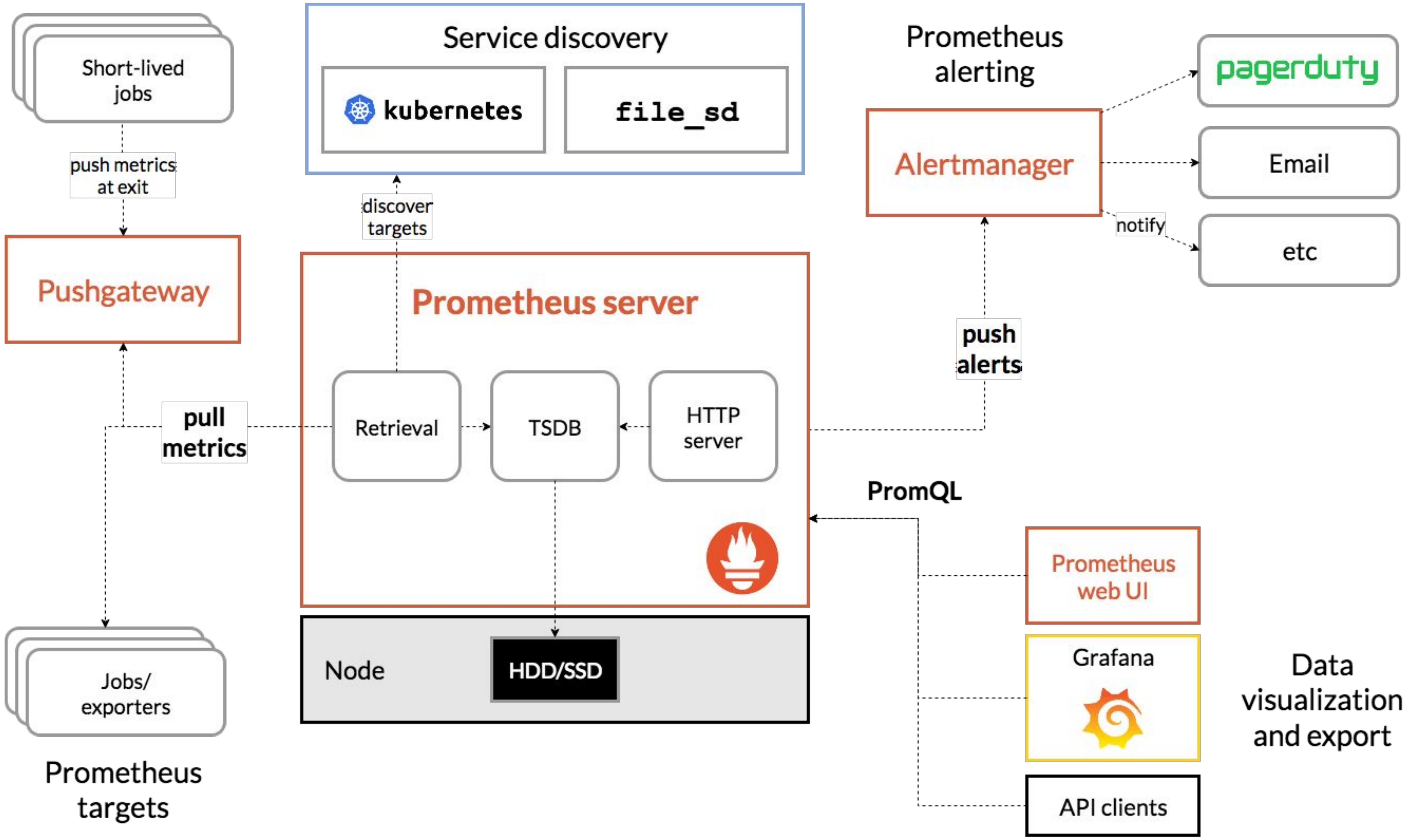
Prometheus

Features

- Open source monitoring solution
- Monitoring targets discovered via service discovery or static configuration
- Kubernetes integration: Service discovery through Kubernetes API
- No reliance on distributed storage: Local on-disk time series database
- Efficient:
  - Minimal datapoint size
  - Single server can handle millions of metrics
- Recording rules to precompute frequently needed or computationally expensive expressions
- Scale out through sharding and federation

Prometheus

Architecture



Service Mesh and Monitoring

- Prometheus and Grafana installed as part of OpenShift Service Mesh
- Prebuilt Grafana dashboards for metrics generated by Envoy proxies and Istio control plane components
- Telemetry V2
  - Integration with Prometheus through Envoy WebAssembly Module (WASM) extension
  - Metrics exposed by Envoy and scraped by Prometheus
- Integration with other monitoring solutions through WASM extensions

Visualization

- Jaeger UI
- Grafana dashboards
- Kiali
  - OpenShift Service Mesh console
  - Visualizes service mesh topology in real time
  - Provides visibility into features like request routing, circuit breakers, request rates, latency, etc.
  - Inline edition of YAML representation of Istio resources, with powerful semantic validation
  - Actions to create, update, delete Istio configuration resources, driven by wizards
  - Custom metrics dashboards
  - Integrated with distributed tracing

Module Summary

- Observability
- Logging
- Distributed Tracing
- Monitoring
- Visualization