KubeCon | CloudNativeCon

China 2018

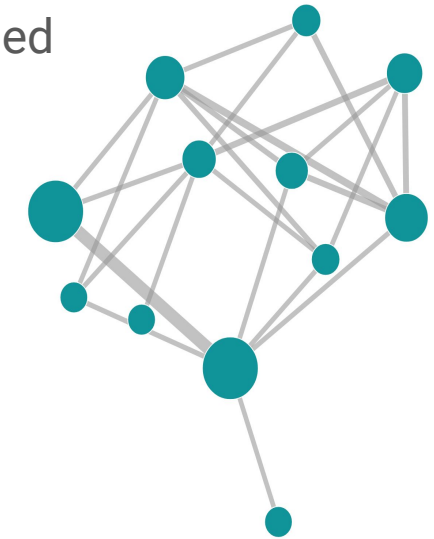
Navigating the Distributed Systems Execution Maze with OpenTracing

Ashlie Martinez

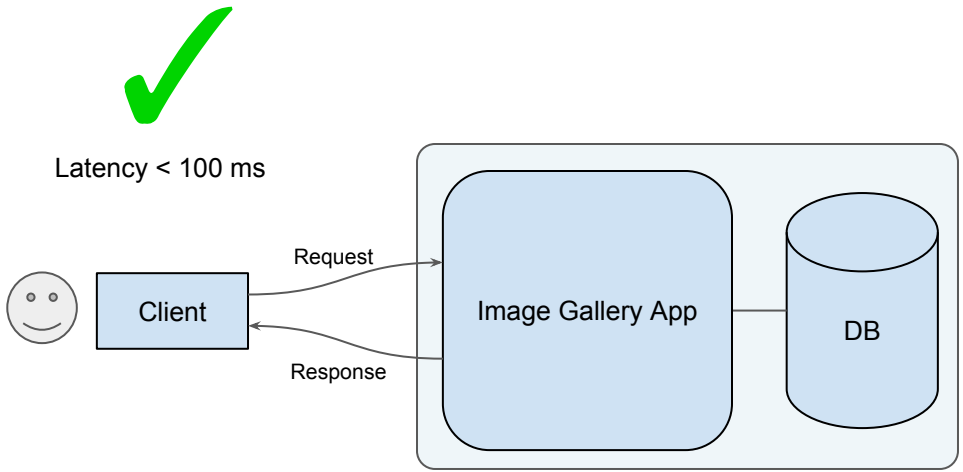
Ilya Kislenko (on behalf of Julio López)

Why Do We Need Distributed Tracing?

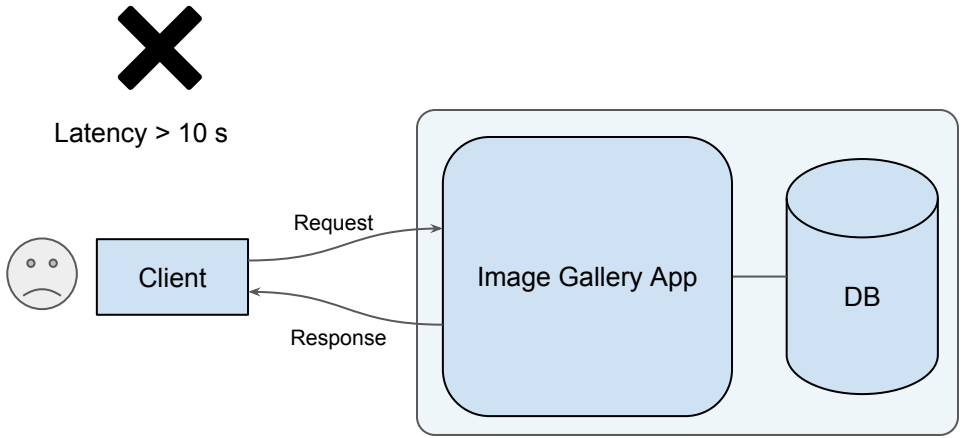
- K10 (Data Management for Cloud-Native Applications) has 13 microservices
- Don’t want to redesign existing K10 microservices just for tracing
- Logging exposes some information, but cannot show time spent in services
- Require tracing library that can be incrementally added



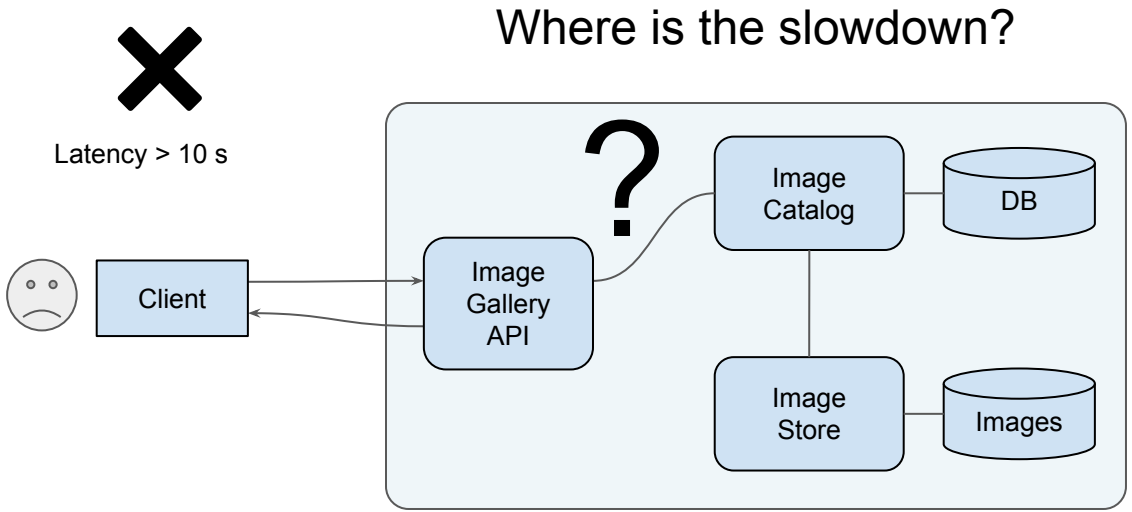
Sample App: Image Gallery



Sample App: Image Gallery



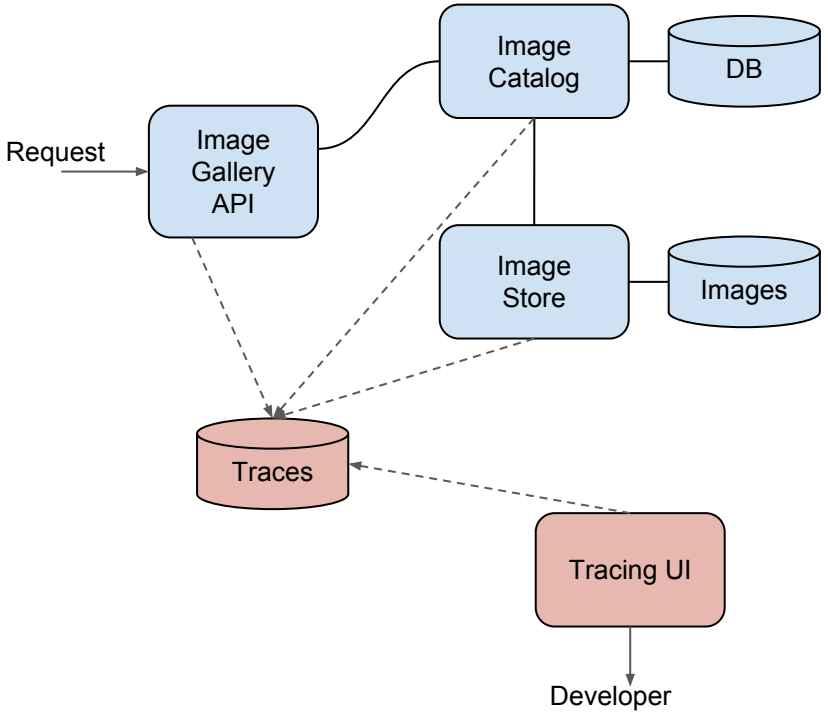
Sample App: Image Gallery



It becomes much harder to troubleshoot and debug a set of intertwined distributed microservices

Distributed Tracing

- Automatically aggregate traces for a request from multiple microservices
- Highlights the execution path of client requests in the backend
- Can help pinpoint where failures occur and what causes poor performance
- Most libraries support probabilistic sampling of requests
- Complements distributed log and metrics collection tools



Leveraging Distributed Tracing: Outline

Steps for tracing applications using REST-based Go microservices running in K8s

- How does distributed request tracing work?
- Our choice for tracing: OpenTracing and Jaeger
- Installing Jaeger in a K8s cluster
- Instrumenting Go microservices using OpenTracing SDKs and APIs
  - Request IDs and spans
- Dealing with services external to the application: Cloud Providers and K8s API

How Does Distributed Tracing Work?

General approach:

- Instrument parts of services with tracing framework to record information
  - Can instrument any part of service, though request level gives reasonable visibility into system
- Configure services to send tracing data to a central database for display
- Database correlates traces from different services for the same request
- Use separate UI to display and search tracing data

OpenTracing & Jaeger

Open Tracing:

- CNCF distributed tracing library for Go, C#, Java, and other languages
- Instrument existing code with OpenTracing calls to collect tracing information

Jaeger:

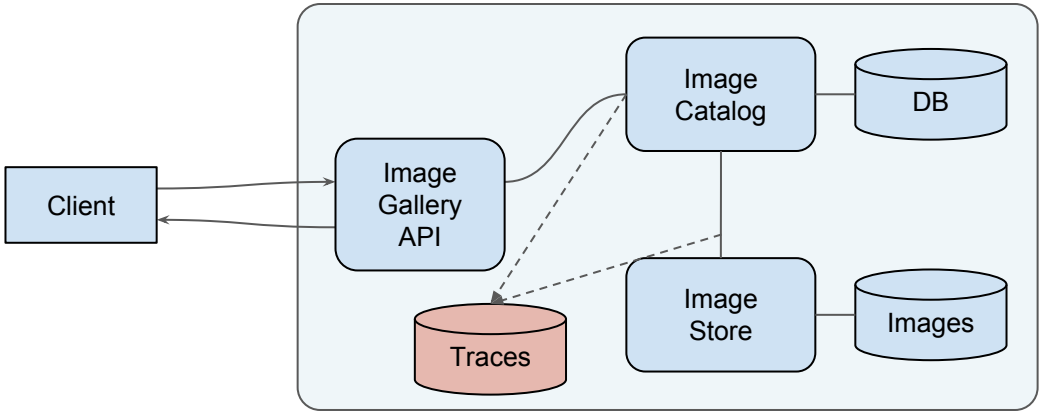
- CNCF UI for visualizing and searching tracing data
- Uses coalesced tracing data stored in a database like Cassandra
- Deployable via helm chart and K8s yaml

Other tracing options: Zipkin, Google OpenCensus

Instrumenting Image Gallery App: Part 1

Instrumenting the Image Gallery API microservice with custom Middleware

```
func Middleware(next http.Handler) http.Handler {
        // requests that go through it.
        return nethttp.Middleware(opentracing.GlobalTracer(),
                next,
                nethttp.OperationNameFunc(func(r *http.Request) string {
                        return "HTTP " + r.Method + " " + r.URL.String()
                }))
}
```



OpenTracing Go SDK

- Each trace collected by a single service is called a “span”
  - Spans can be nested to show one service calling another
- OpenTracing leverages Go’s Context object to carry info about traces
  - Code being traced must propagate Context to be traced
- Information like HTTP status codes or request IDs can be added to traces
  - Allows developers to get more information about the state of the system for that trace
  - Can help the developer associate a specific trace with other debug information like logs
- Deploy Jaeger with in-memory database via Helm chart

Instrumenting Image Gallery App: Part 1

And we got 1 lonely trace

Instrumenting Image Gallery App: Part 2

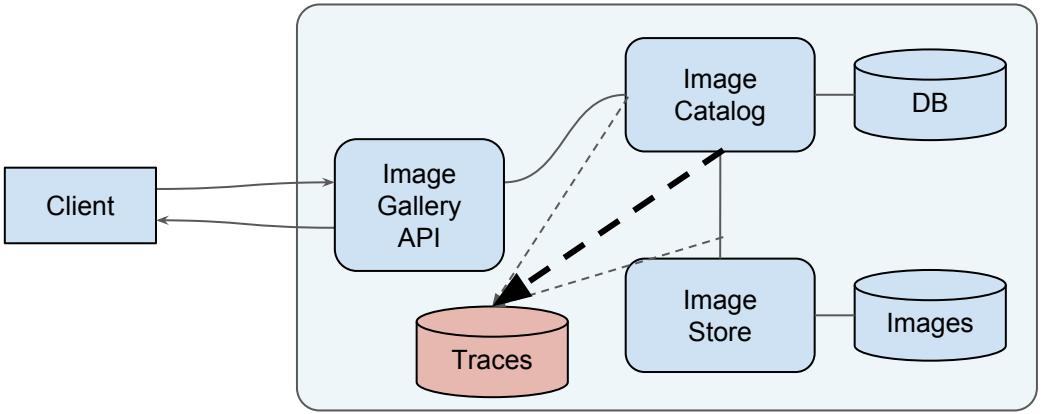
Instrumenting the Image Catalog and Image Store microservices' outgoing requests

```
func (t *tracingTransport) RoundTrip(r *http.Request) (*http.Response, error) {
        ctx := r.Context()
        span, ctx2 := opentracing.StartSpanFromContext(ctx, "HTTP Request")
        defer span.Finish()

        r.WithContext(ctx2)
        carrier := opentracing.HTTPHeadersCarrier(r.Header)
        span.Tracer().Inject(span.Context(), opentracing.HTTPHeaders, carrier)

        resp, err := t.transport.RoundTrip(r)

        return resp, err
}
```



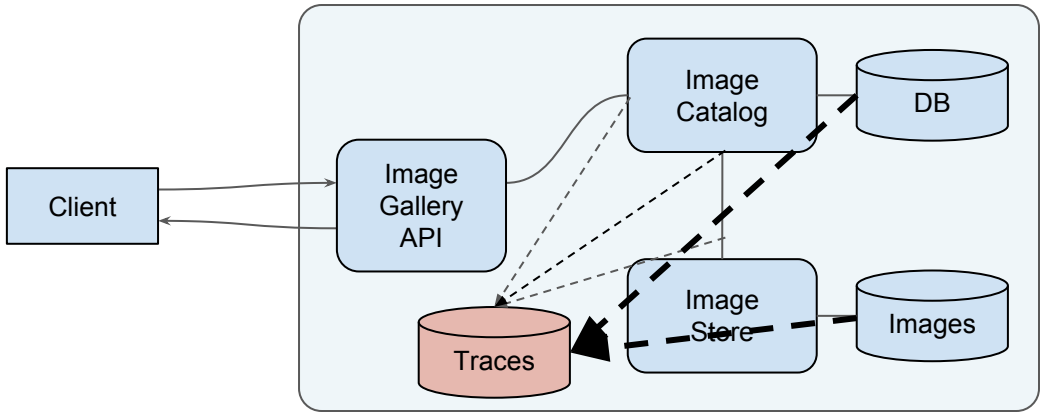
Instrumenting Image Gallery App: Part 2

Now we can see that metadata is calling store service

Instrumenting Image Gallery App: Part 3

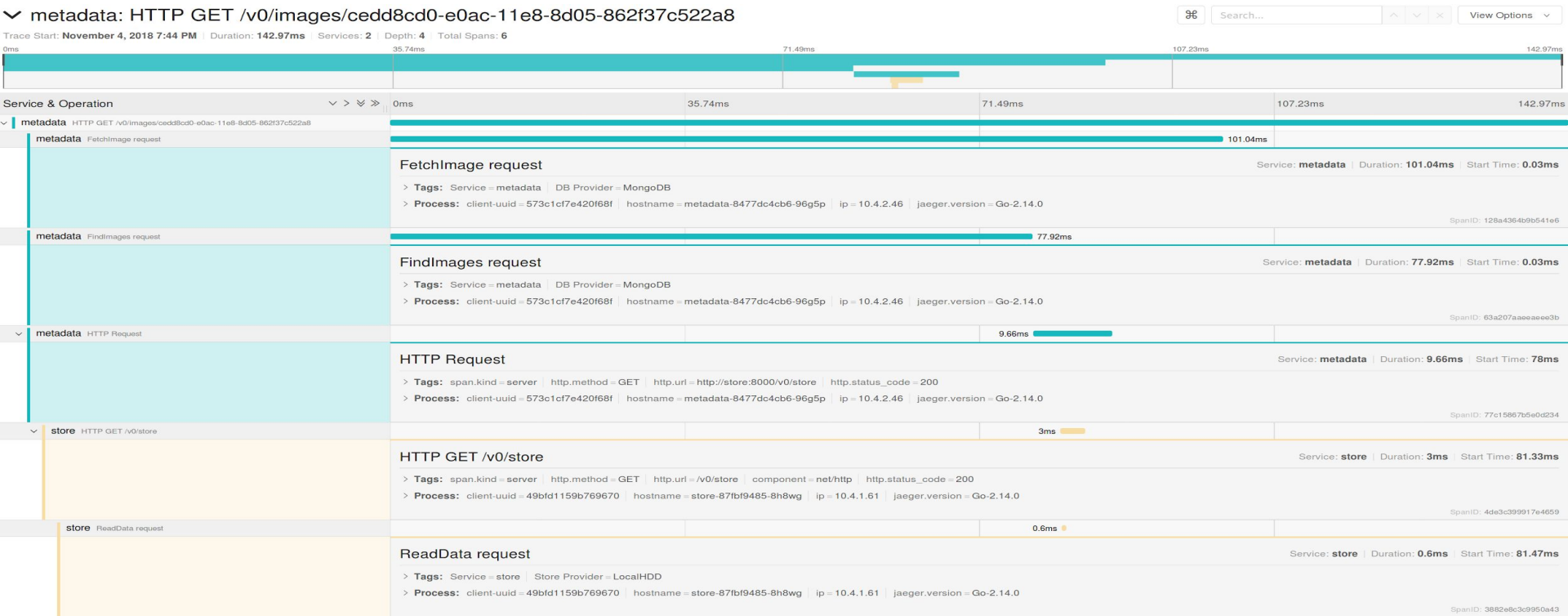
Internal instrumentation example: Calls into the DB

```
func (s *Mongo) GetAllImages(ctx context.Context) (models.ImageList, error) {
        span, _ := opentracing.StartSpanFromContext(ctx, "GetAllImages request")
        defer span.Finish()
        addSpanTags(span)
        err := s.Ping()
        if err != nil {
                return models.ImageList{}, err
        }
        c := s.Conn.DB(dbName).C(collName)
        imgs := models.ImageList{}
        return imgs, c.Find(nil).All(&imgs)
}
```



Instrumenting Image Gallery App: Part 3

Here we can see everything.



Discussion

General approach: start with coarse tracing and add finer granularity as needed

Trade-offs:

- Pro: Fine granularity and detailed request information
- Con: Additional resource requirements.
  - Request processing in each of the services, and additional network traffic
  - Additional processing and storage requirements for the traces

Other thoughts:

- Service meshes, such as Istio, give you coarse grained tracing
- Complements and overlaps with metrics and logging

Bringing it all Together

Today we discussed

- Using OpenTracing Go SDK to add instrumentation microservices
- Instrumenting calls to other services: DB, cloud provider, K8s API
- Installing Jaeger tracing collector and UI in k8s cluster
- Using Jaeger UI to visualize, analyze and dig into traces

Final Thoughts

Tracing can give insights into system bottlenecks, but need to balance with time spent adding instrumentation

Trade-offs:

- Pro: Fine granularity and detailed request information
- Con: Additional resource requirements.
  - Request processing in each of the services, and additional network traffic
  - Additional processing and storage requirements for the traces

Other thoughts:

- Service meshes, such as Istio, give you coarse grained tracing
- Complements and overlaps with metrics and logging

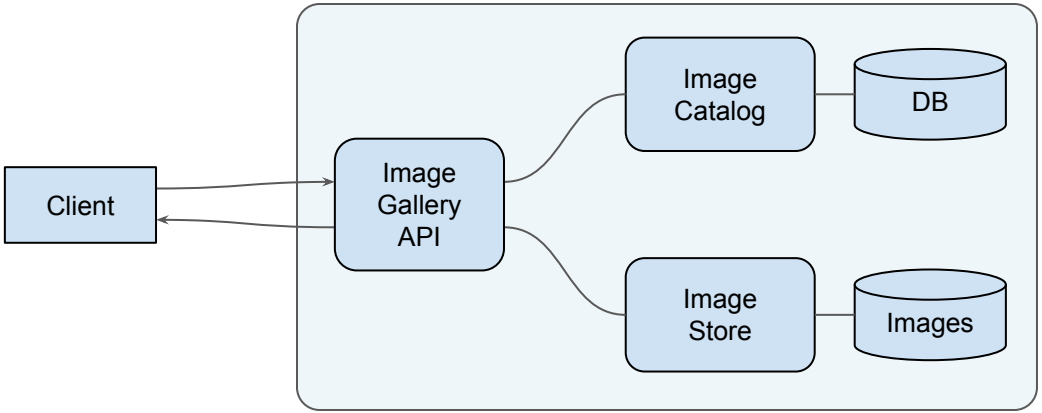
KubeCon

CloudNativeCon

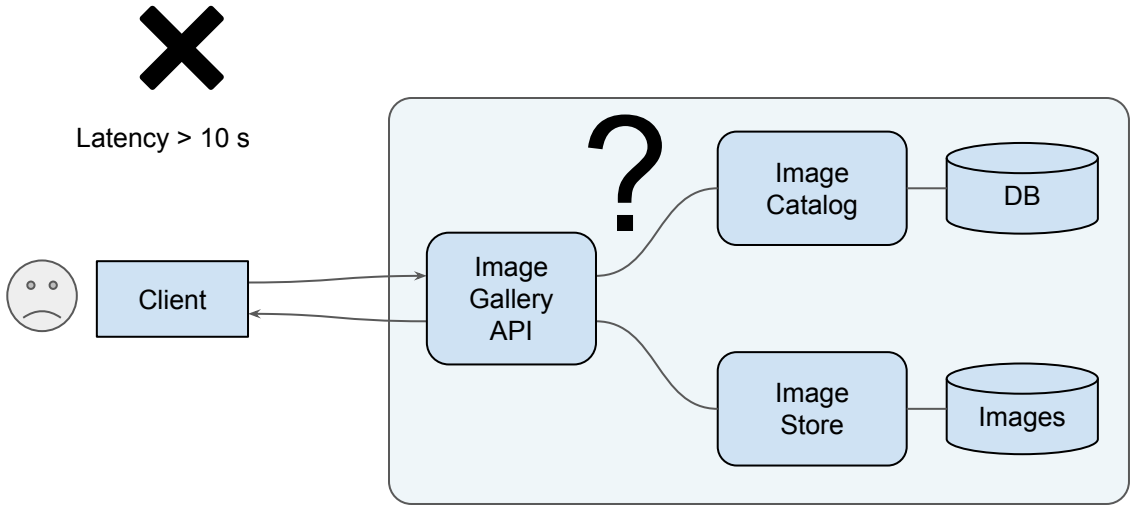
China 2018

Questions?

Sample App: Image Gallery



Sample App: Image Gallery



Where is the slowdown?