AWS

SUMMIT

Deep Dive: AWS X-Ray

London 2017

Randall Hunt, Technical Evangelist AWS
Ashley Sole, Software Engineer, Skyscanner

amazon web services

Who am I and why am I here?

- Software Engineer at AWS, previously at SpaceX and NASA (and MongoDB where I learned to code)
- Follow me on twitter @jrhunt for demo and serverless hot takes
- Email me: randhunt@amazon.com
- Thanks to all the people at AWS who helped build these slides!

Agenda

- Overview
- Concepts
- API
- Use cases
- Getting started
- Q & A

Challenges

Deploying and managing service-oriented applications is more work compared to monolithic applications

Services such as AWS Lambda, Amazon EC2 Container Service, AWS Elastic Beanstalk, AWS CloudFormation, etc. make it easier to deploy and manage applications consisting of hundreds of services

Still hard to debug application issues in production applications due to:

- Cross-service interactions
- Varying log formats across services
- Collecting, aggregating, and collating logs from services

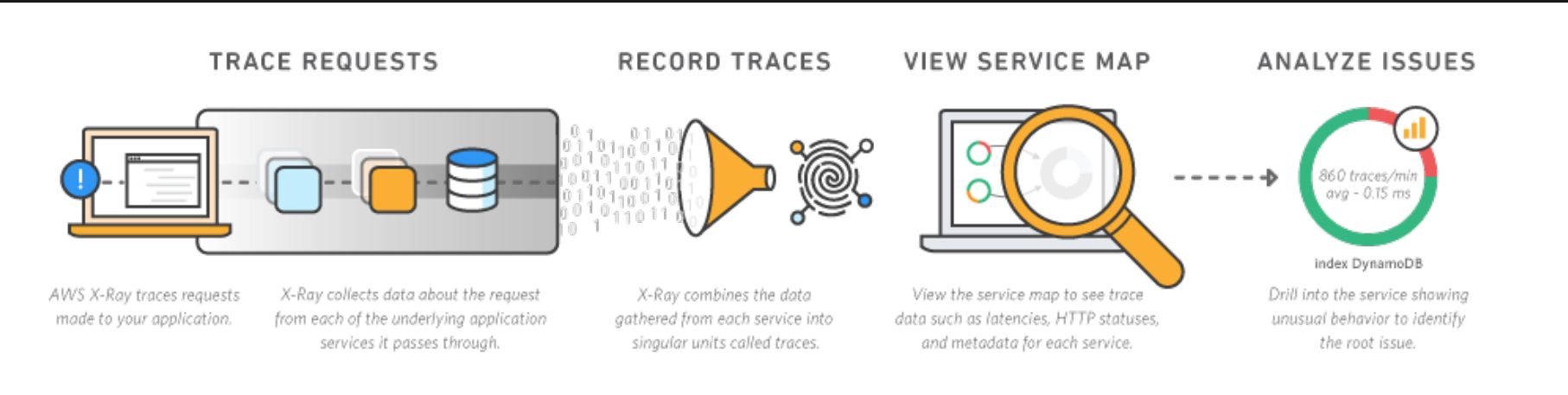
How does X-Ray help?

Solution

AWS X-Ray makes it easy to:

- Identify performance bottlenecks and errors
- Pinpoint issues to specific service(s) in your application
- Identify impact of issues on users of the application
- Visualize the service call graph of your application

X-Ray service



X-Ray SDK

Available for Java, .NET, and Node.js

Adds filters to automatically captures metadata for calls to:

- AWS services using the AWS SDK
- Non-AWS services over HTTP and HTTPS
- Databases (MySQL, PostgreSQL, and Amazon DynamoDB)
- Queues (Amazon SQS)

Enables you to get started quickly without having to manually instrument your application code to log metadata about requests

X-Ray daemon

Receives data from the SDK over UDP and acts as a local buffer. Data is flushed to the backend every second or when the local buffer fills.

Available for Amazon Linux AMI, RHEL, Ubuntu, OS X, and Windows

Concepts

X-Ray Concepts

Trace	End-to-end data related a single request across services
Segments	Portions of the trace that correspond to a single service
Sub-segments	Remote call or local compute sections within a service
Annotations	Business data that can be used to filter traces
Metadata	Business data that can be added to the trace but not used for filtering traces
Errors	Normalized error message and stack trace

Sampling configuration

```
{
  "rules": {
    "move": {
      "id": 1,
      "service_name": "*",
      "http_method": "*",
      "url_path": "/api/move/*",
      "fixed_target": 0,
      "rate": 0.05
    },
    "base": {
      "id": 2,
      "service_name": "*",
      "http_method": "*",
      "url_path": "*",
      "fixed_target": 1,
      "rate": 0.1
    }
  }
}
```

This example defines two rules.

The first rule applies a five-percent sampling rate with no minimum number of requests to trace to requests with paths under `/api/move`

The second overrides the default sampling rule with a rule that traces the first request each second and 10 percent of additional requests.

APIs

X-Ray API

X-Ray provides a set of APIs to enable you to send, filter, and retrieve trace data

You can send trace data directly to the service without having to use our SDKs (i.e. you can write your own SDKs for languages not currently supported)

Raw trace data is available using batch get APIs

You can build your own data analysis applications on top of the data collected by X-Ray

X-Ray API

PutTraceSegments	Uploads segment documents to AWS X-Ray
BatchGetTraces	Retrieves a list of traces specified by ID
GetServiceGraph	Retrieves a document that describes services in your application and their connections
GetTraceSummaries	Retrieves IDs and metadata for traces available for a specified time frame using an optional filter

Segment document

Minimal example

```
{
  "name" : "example.com",
  "id" : "70de5b6f19ff9a0a",
  "start_time" : 1.478293361271E9,
  "trace_id" : "1-581cf771-a006649127e371903a2de979",
  "end_time" : 1.478293361449E9
}
```

Example showing an in-progress segment

```
{
  "name" : "example.com",
  "id" : "70de5b6f19ff9a0b",
  "start_time" : 1.478293361271E9,
  "trace_id" : "1-581cf771-a006649127e371903a2de979",
  “in_progress”: true
}
```

Traced Request

Optional: Parent segment ID

Ashley Sole

Software Engineer - Skyscanner

Software Engineer and technical leader with over 10 years experience.

Worked in various industries in business from Defence to Gaming.

I believe in a blameless culture and that the most important thing to delivering good quality products is a strong happy team.

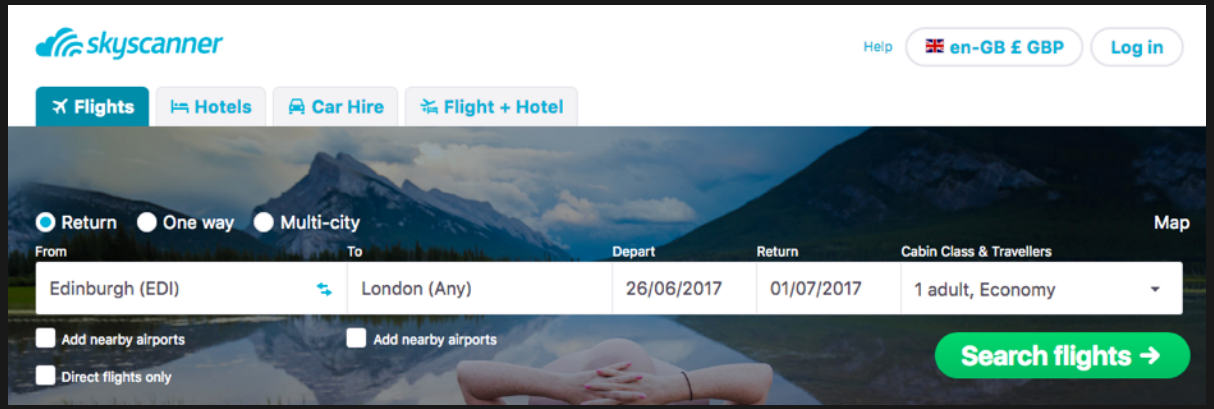
X-Ray @ Skyscanner

Skyscanner

We have a goal to be the most used and the most trusted online travel brand in the world.

Skyscanner

Approaching 100 million monthly unique visitors
Getting towards 1,000 microservices in AWS
1000’s of deploys every day in multiple AWS regions
This complexity means we need a way to understand our system



100% Availability

Consumers everywhere expect Skyscanner to be always available and reliable

In order to increase the availability of our services in production we need to be able to monitor the communication happening between them to anticipate any failures and diagnose the root cause of them if/when they happen.

Tracing the path of Requests

Request tracing gives us the ability to track a single request end-to-end: from a user searching in our website to returning a list of flights and everything in between.

What have we built?

We've orchestrated http clients to add X-Ray tracing to outgoing requests

`JerseyClientRegistry` to provide Clients that are orchestrated with tracing.

Wrappers for `request` and `request-promise-native`

What have we built?

We promote functions for service developers to easily wrap particular sections of code in tracing

```
// subsegments as lambda functions
String operationResult = Tracing.callWithSubsegment("long operation", subsegment -> {
    String result = performLongOperation();
    segment.putAnnotation("result", result);
    return result;
});


// subsegments as an object - this is equivalent to above
TracingSubsegment subsegment = Tracing.beginSubsegment("long operation 2");
try {
    String result = performLongOperation();
    subsegment.putAnnotation("result", result);
} catch (Exception e) {
    subsegment.addException(e);
    throw e;
} finally {
    subsegment.end();
}
```

What have we built?

We've added new annotations to simply add tracing to specific functions

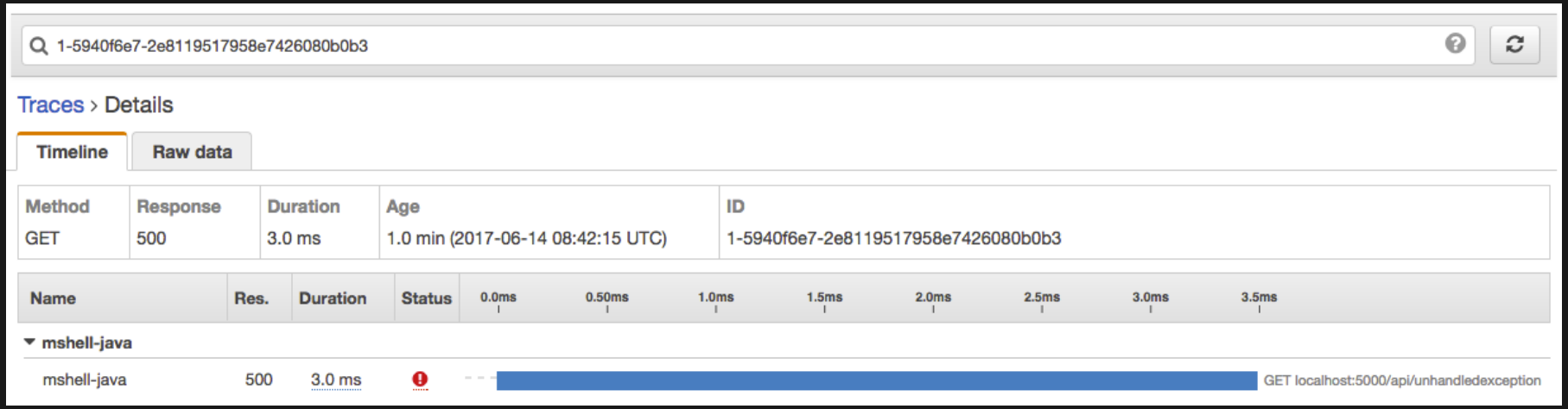
```
class Foo {
    @TracedSubsegment
    public void bar() { ... } // subsegment name = Foo::bar

    @TracedSubsegment("custom")
    public void baz() { ... } // subsegment name = custom
}
```

What have we built?

Deep linking into X-Ray from log messages

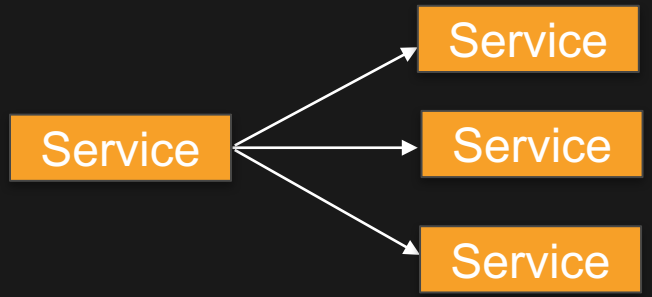
Time	message	severity	fields.Tracing-Trace-Link
June 14th 2017, 09:42:15.661	ERROR [2017-06-14 08:42:15,659] [dw-17 - GET /api/unhandledexception] net.skyscanner.mshell.exceptions.LoggingTracedExceptionMapper: Error handling a request: 3e315ba54e1b11d8. This request has been traced with X-Ray and can be seen here: https://eu-west-1.console.aws.amazon.com/xray/home?region=eu-west-1#/traces/1-5940f6e7-2e8119517958e7426080b0b3 ! java.lang.UnsupportedOperationException: Example exception message ! at net.skyscanner.mshell.api.HelloWorldResource.unhandledException(HelloWorldResource.java:76) ! at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)	ERR	https://eu-west-1.console.aws.amazon.com/xray/home?region=eu-west-1#/traces/1-5940f6e7-2e8119517958e7426080b0b3



How Does X-Ray Help Us?

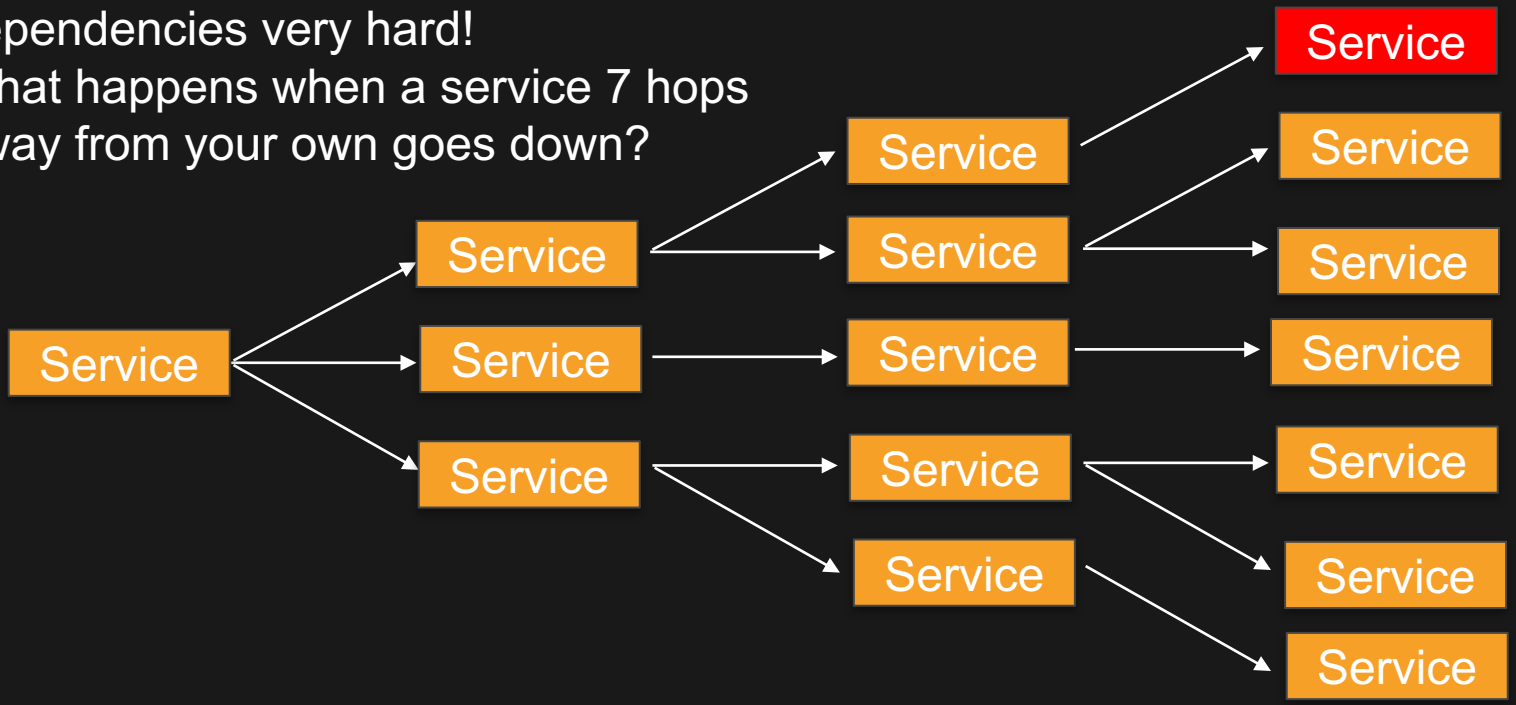
Use Case 1 – Understanding your dependencies

When your system is simple, it’s easy to understand your dependencies



Use Case 1 – Understanding your dependencies

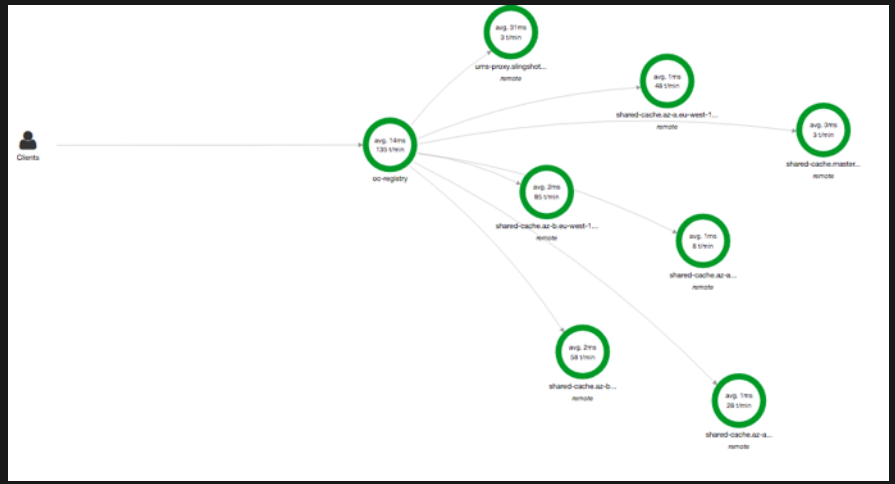
But microservice architecture has made understanding your dependencies very hard!
What happens when a service 7 hops away from your own goes down?



Use Case 1 – Understanding your dependencies

Tracing helps to visualize the reality of your production system

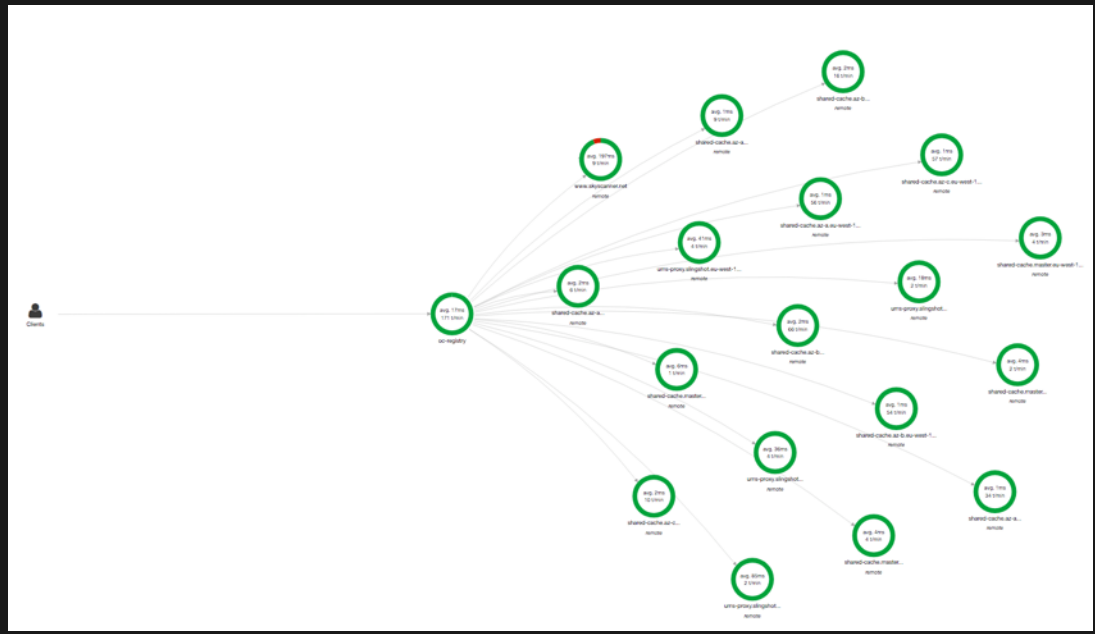
Going from what you think your system looks like…



Use Case 1 – Understanding your dependencies

…to understanding what your system actually looks like!

Who knew your service was connected to that legacy service that was about to be decommissioned



Use Case 1 – Understanding your dependencies

X-Ray helps us understand the complexity of our distributed system

It allows us to visualise the system as a whole and dive in and out of what we need

Use Case 2 – Troubleshooting slow performance

You get an alert that something has decreased in performance

How would you typically debug this on a live complex distributed system? Look at the logs?

Let’s try using X-Ray

The following scenario is a genuine bug, found using X-Ray

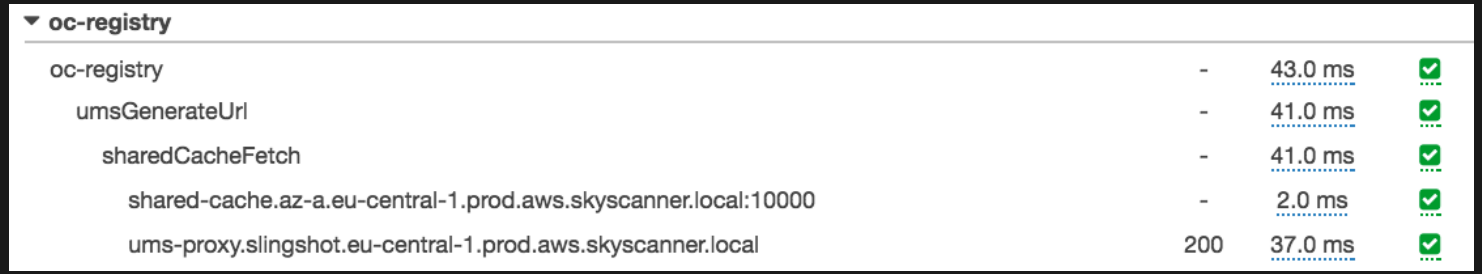
Use Case 2 – Troubleshooting slow performance

Look into a trace of the service that is performing slow

What does this tell us?

Use Case 2 – Troubleshooting slow performance

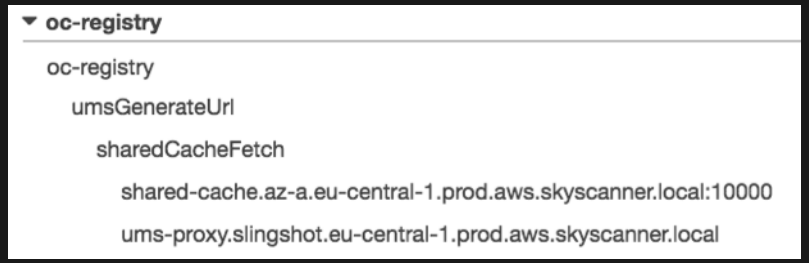
Calls to umsGenerateUrl are taking 41.0ms – slower than we would like!



Use Case 2 – Troubleshooting slow performance

What's actually being called?

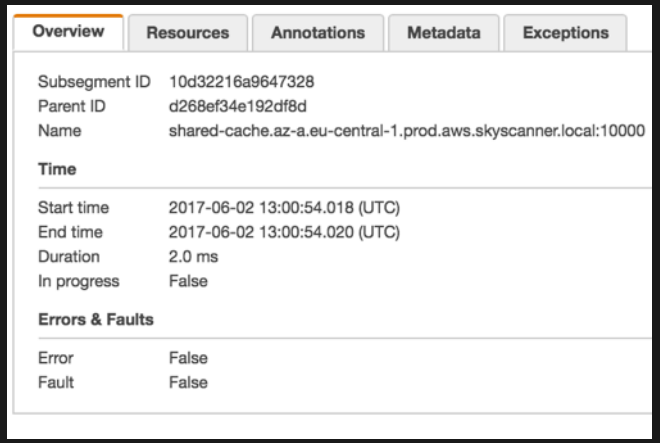
oc-registry is making a call to umsGenerateUrl
umsGenerateUrl calls sharedCacheFetch
sharedCacheFetch makes two calls…

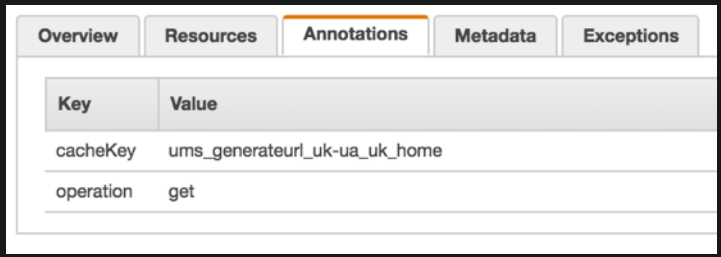


Use Case 2 – Troubleshooting slow performance

The first of these two calls is a call to get a key from the cache

X-Ray tells us this takes 2.0ms and gives us the request details

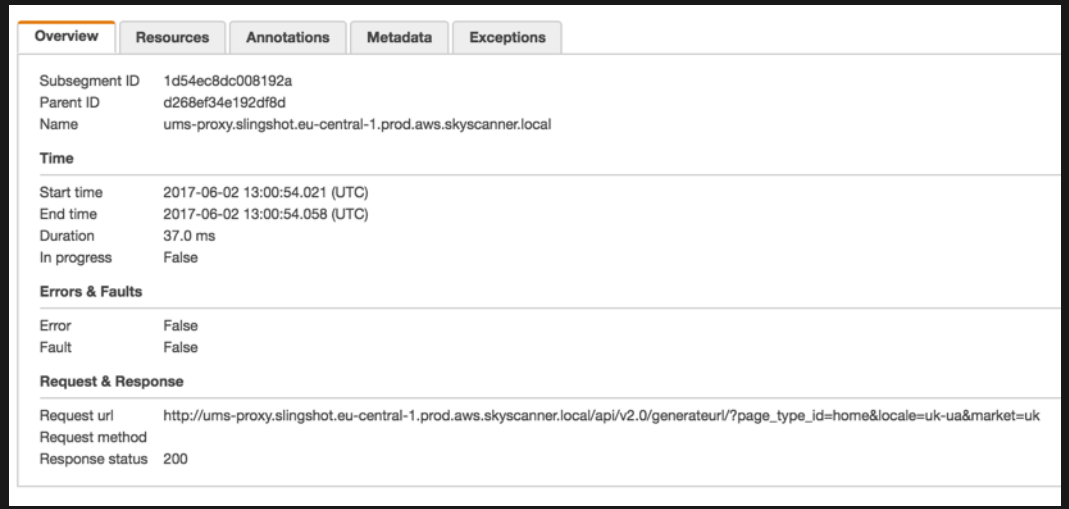




Use Case 2 – Troubleshooting slow performance

The second call is a remote call to retrieve the requested value from an API

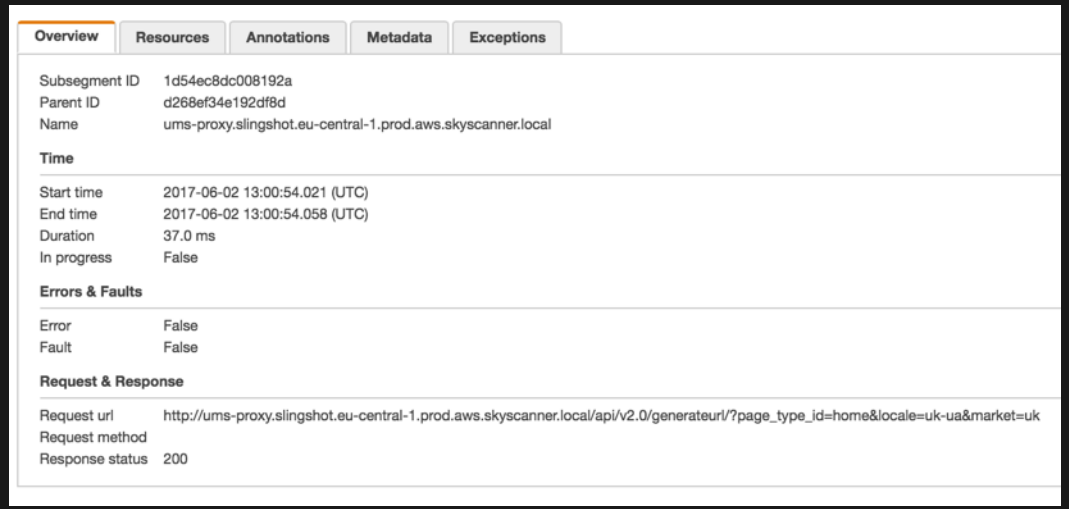
X-Ray tells us this takes 37.0 ms and returns with a 200 response



Use Case 2 – Troubleshooting slow performance

What's actually happening here?

We've got a cache miss, so are making a http GET request



Use Case 2 – Troubleshooting slow performance

We seem to be getting an awful lot of cache misses…

Going back to our original trace – can we spot a problem?

Use Case 2 – Troubleshooting slow performance

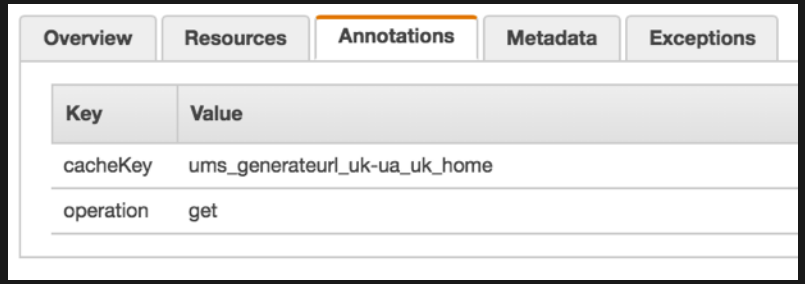
We’re doing a cache “get”, but no cache “set”!

umsGenerateUrl

sharedCacheFetch

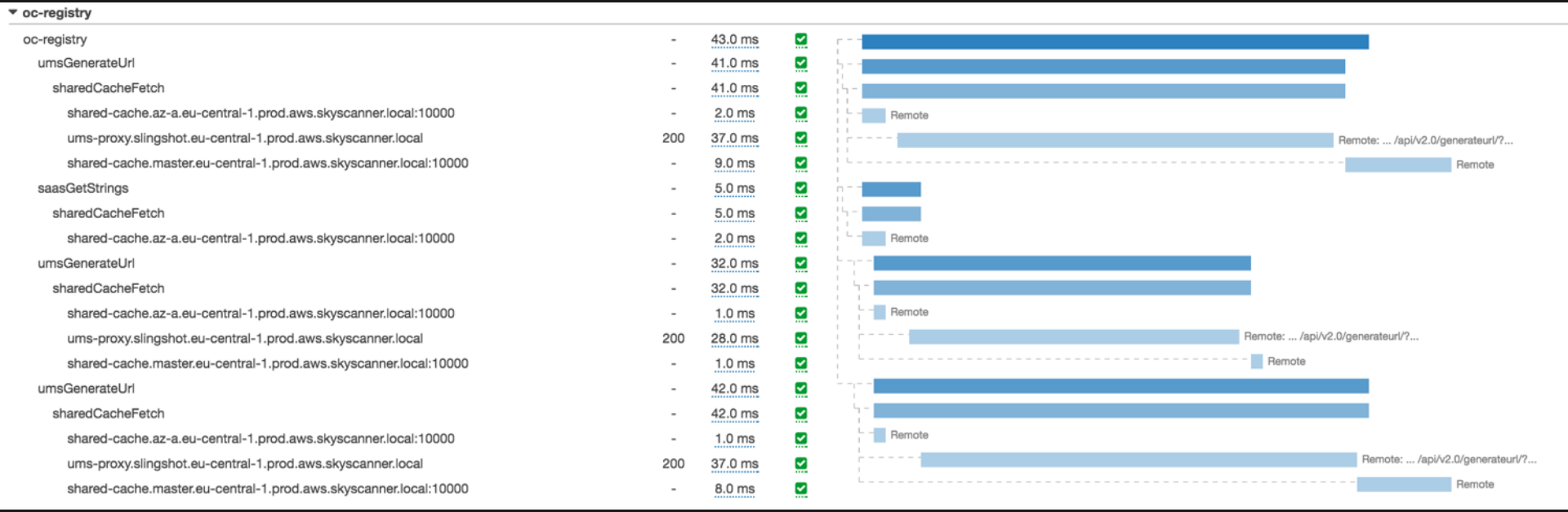
shared-cache.az-a.eu-central-1.prod.aws.skyscanner.local:10000

ums-proxy.slingshot.eu-central-1.prod.aws.skyscanner.local



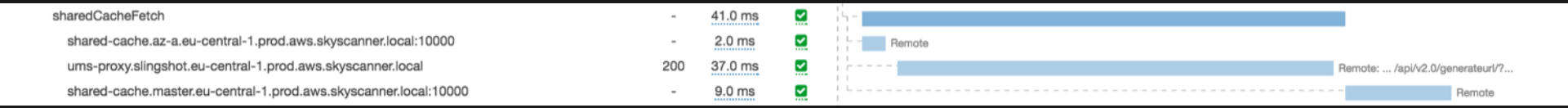
Use Case 2 – Troubleshooting slow performance

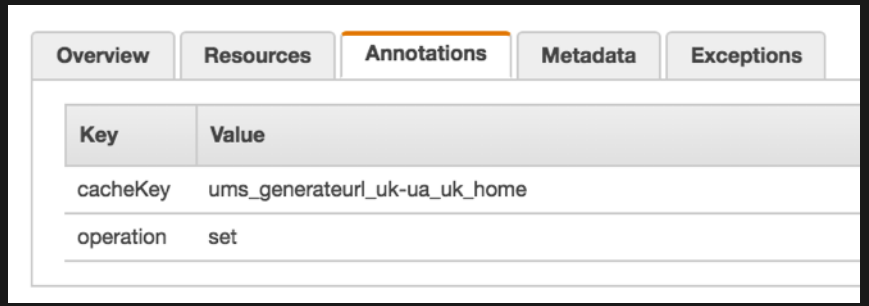
Fix the bug! Deploy! Check X-Ray again



Use Case 2 – Troubleshooting slow performance

We can see that we've added a call to do an asynchronous "set" to add our value to the cache in the event of a cache miss.

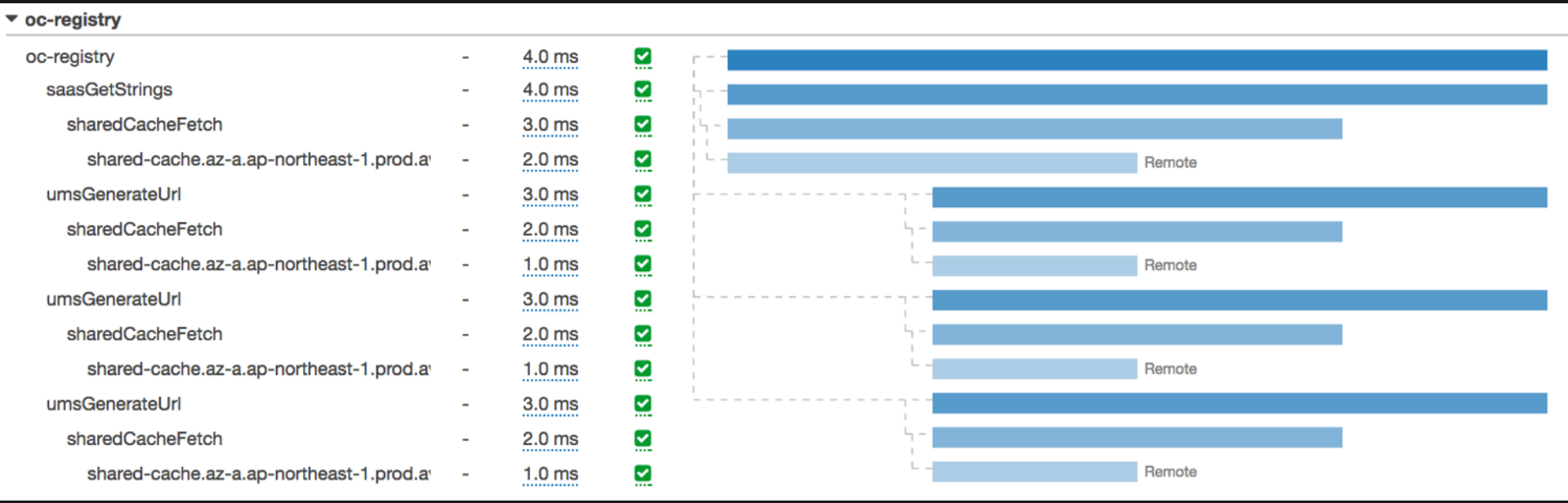




Use Case 2 – Troubleshooting slow performance

So that subsequent calls get many more cache hits!

And our umsGenerateUrl function now returns in 4.0ms instead of 41ms!



Summary

X-Ray gives us **deep insight** into our services to investigate when things go wrong

Analysis of the traces allows us to understand detailed information about how services are interacting with each other

We can track down bugs by more ways than trawling the logs

Briefly back to Randall

X-Ray Pricing

Free during the preview. After that:

Free tier

- The first 100,000 traces recorded per month are free
- The first 1,000,000 traces retrieved or scanned per month are free

Additional charges

- Beyond the free tier, traces recorded cost $5.00 per million per month
- Beyond the free tier, traces retrieved or scanned cost $0.50 per million per month

Currently AWS Supported SDKs and Frameworks

- Java
  - Tomcat, Spring Boot, Servlet filters
- Node.js (express filters)
- C# (.NET, .NET core)

Currently (Natively) Supported Services

- ELB
- Lambda
- API Gateway
- EC2
- Elastic Beanstalk

AWS

SUMMIT

Thank You!

@jrhunt

amazon web services