

# AWS X-Ray

(with Fargate and Lambda)

Doug Toppin  
AWS DC Meetup  
15-Oct-2019

# Agenda

- What is it
- Why use it
- What does it cost
- How does it work
- Where did it help me
- How did I use it
- Demo
- Lessons learned
- Links

# What is it

“AWS X-Ray helps developers analyze and debug production, distributed applications, such as those built using a microservices architecture. With X-Ray, you can understand how your application and its underlying services are performing to identify and troubleshoot the root cause of performance issues and errors. X-Ray provides an end-to-end view of requests as they travel through your application, and shows a map of your application’s underlying components. You can use X-Ray to analyze both applications in development and in production, from simple three-tier applications to complex microservices applications consisting of thousands of services.”

<https://aws.amazon.com/xray/>

# Why use it

- Collect and display application performance metrics
- Gives insight into how an application is performing and at granular levels
- Metrics comparison after deployments provides immediate indication of performance differences or where issues might be occurring
- Very cost effective
- If it saves any labor/time in awareness or while troubleshooting it is worth the cost
- Cost can be dialed up/down by changing sampling level

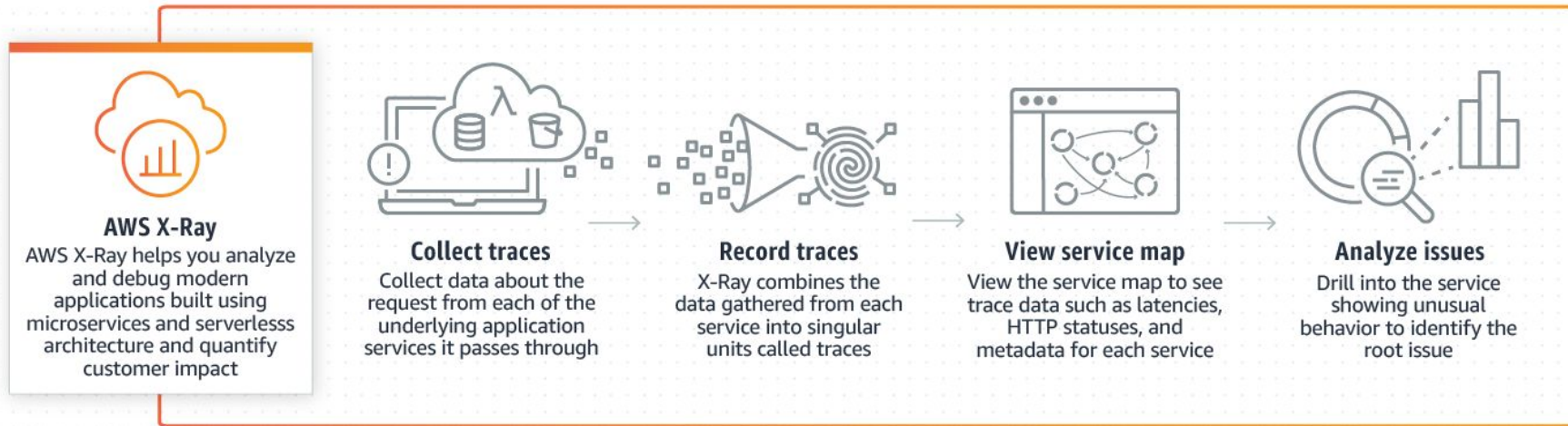
# What does it cost

- The first 100,000 traces recorded each month are free.
- The first 1,000,000 traces retrieved or scanned each month are free.
- If you have an application which receives 2,000 incoming requests per hour and you're using a 10% sampling rate, then your cost would be calculated as follows:
  - Traces Recorded
    - Traces Recorded per Month = 2,000 requests per hour x 24 hours x 31 days x 10% = 148,800 traces
    - Billable Traces Recorded per Month = 148,800 traces - 100,000 traces in free tier = 48,800 traces
    - Monthly Traces Recorded Charges = 48,800 traces \* \$0.000005 = \$0.24

# How does it work

- Container, my Python Flask use example
  - Instrumentation data sent to an agent installed in the container, it could have been an independent container in the Fargate task
  - Agent forwards to service
- Lambda, my email processing function
- EC-2
  - Agent installed on instance and forwards to service
- Non-AWS resource, such as your machine
  - Agent installed and running
- Application can send directly to service without using an agent

# How does it work



# Where did it help me

- Within the first 2 days of use it revealed where performance bottlenecks in aws s3 calls were occurring
- On 3 other occasions it provided an immediate indication of where performance issues were occurring or improvements had appeared after a deployment



# How did I use it

- Python Flask apps running in Fargate
  - S3
  - DynamoDB
  - SNS/SQS
- Lambda functions
  - S3
  - SES

# Demo

- Python Flask application running in Fargate
  - Created and deployed using the AWS Fargate cli
- Lambda function
  - Created and deployed using AWS cli

# Demo system map

## AWS X-Ray

Getting started

**Service map**

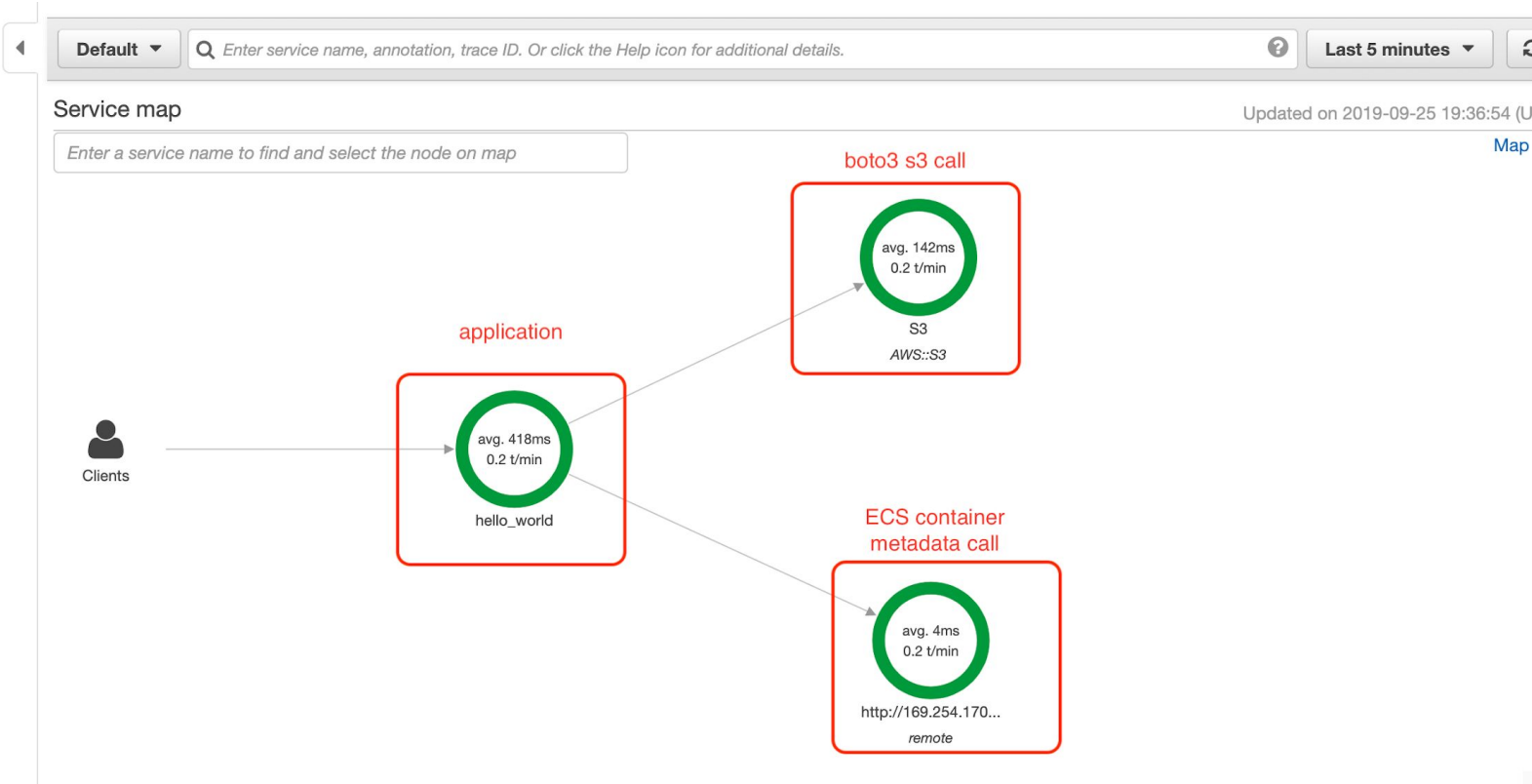
Traces

Analytics new

Configuration

Sampling

Encryption



# Demo traces

## AWS X-Ray

Getting started

Service map

### Traces

Analytics new

Configuration

Sampling

Encryption

Default

Q Enter service name, annotation, trace ID. Or click the Help icon for additional details.

?

Last 15 minutes

Trace overview

Group by: URL

Done 100% scanned (found 2)

URL	Avg response time	% of Traces	Response
-	261 ms	100.00%	2 OK, 0 Throttled, 0 Errors, 0 Faults

Trace list

ID	Age	Method	Response	Response time	URL	Client IP	Annotations
...2c3adb14	5.5 min			418 ms			0
...cb49c7cb	5.4 min			103 ms			0

2 request traces

# Demo trace

Traces &gt; Details

## Timeline

## Raw data

Method	Response	Duration	Age	ID
--	--	418 ms	7.7 min (2019-09-25 23:36:28 UTC)	1-5d8bf9fc-13b3055210080b0f2c3adb14

Name	Res.	Duration	Status	0.0ms	50ms	100ms	150ms	200ms	250ms	300ms	350ms	400ms	450ms
▼ hello_world	segments/subsegments			time breakdowns									
hello_world	-	418 ms	✓	[Timeline bar for hello_world]									
get-s3-data-files	-	418 ms	✓	[Timeline bar for get-s3-data-files]									
http://169.254.170.2/v3/fc6a55a7-1d45-42b4-	200	4.4 ms	✓	[Timeline bar for http request]									
S3	200	142 ms	✓	[Timeline bar for S3 response]									
▶ http://169.254.170.2/v3/fc6a55a7-1d45-42b4-a6d1-470b37fe09c3/task (Client Response)													
▶ S3 AWS::S3 (Client Response)													

# Demo comparisons after a deployment

Traces > Details

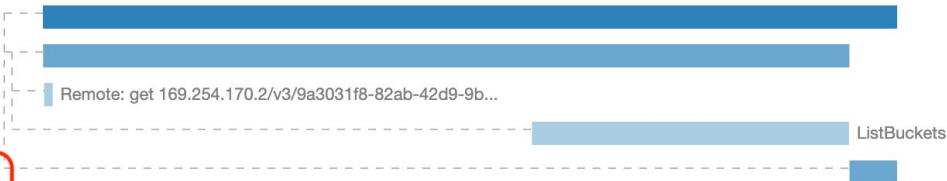
Timeline

Raw data

Method	Response	Duration	Age	ID
--	--	460 ms	1.0 min (2019-09-26 01:30:06 UTC)	1-5d8c149e-0a94607b0e1567e01f8d186d

Name	Res.	Duration	Status	0.0ms	50ms	100ms	150ms	200ms	250ms	300ms	350ms	400ms	450ms	500ms
------	------	----------	--------	-------	------	-------	-------	-------	-------	-------	-------	-------	-------	-------

## ▼ hello\_world

hello_world	-	460 ms	✓											
get-s3-data-files	-	434 ms	✓											
http://169.254.170.2/v3/9a3031f8-82ab-42d9-9b01-31ecfc730c99/task	200	4.6 ms	✓											
S3	200	171 ms	✓											
get-a-file	-	25.6 ms	!											

► http://169.254.170.2/v3/9a3031f8-82ab-42d9-9b01-31ecfc730c99/task (Client Response)

► S3 AWS::S3 (Client Response)

# Demo Lambda

## Service map

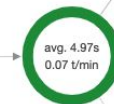
Enter a service name to find and select the node on map



Clients



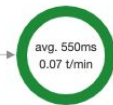
kindle-filter-email  
AWS::Lambda



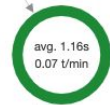
kindle-filter-email  
AWS::Lambda::Function



dougtoppin-email  
AWS::S3::Bucket



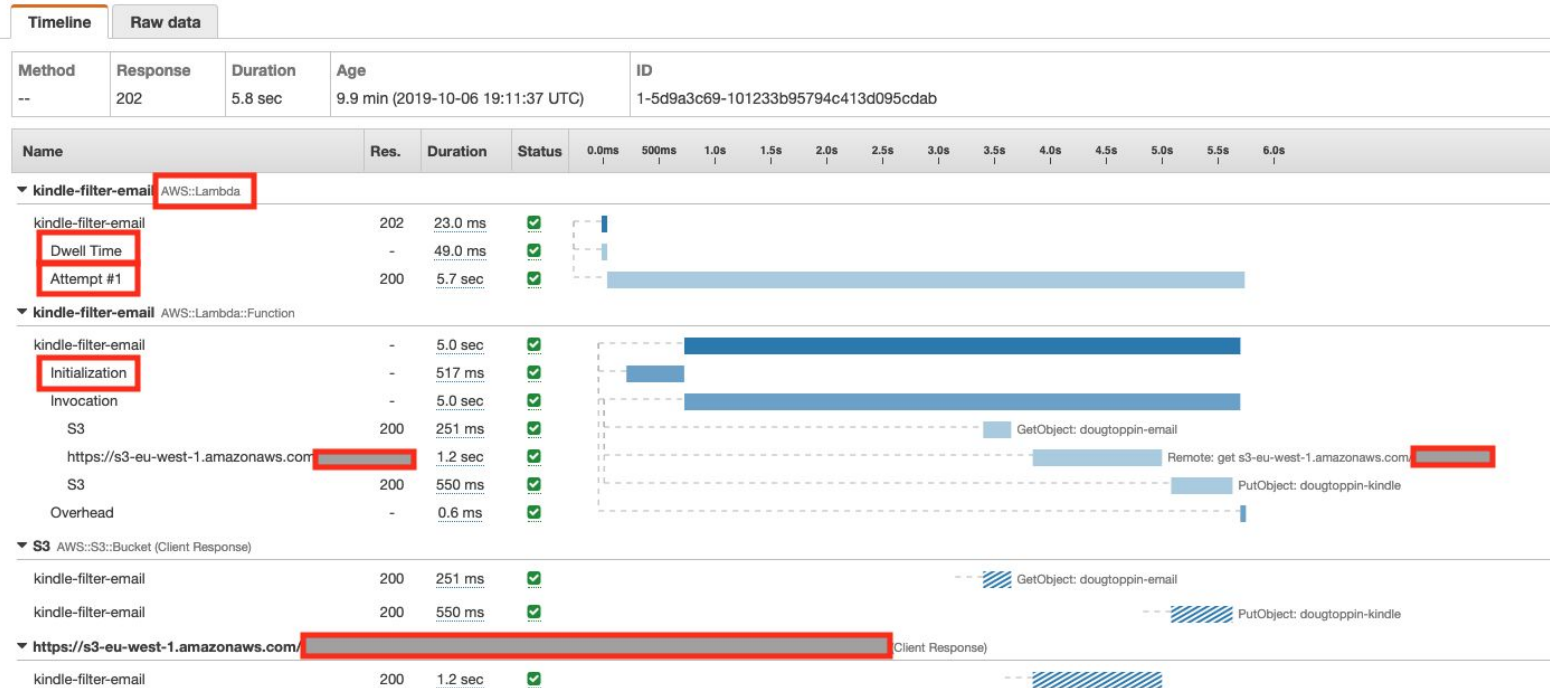
dougtoppin-kindle  
AWS::S3::Bucket



https://s3-eu-west-1...  
remote

# Demo Lambda metrics

Traces > Details





# Demo Lambda metrics definitions

- Dwell time - time spent in Lambda service queue
- Invocation attempts - retries
  - <https://docs.aws.amazon.com/lambda/latest/dg/retries-on-errors.html>
  - “**Asynchronous Invocation** – Lambda retries function errors twice. If the function doesn't have enough capacity to handle all incoming requests, events may wait in the queue for hours or days to be sent to the function. You can configure a dead-letter queue on the function to capture events that were not successfully processed. For more information, see [Asynchronous Invocation](#).”
- Initialization - function initialization code run before the handler

# Demo Lambda attempts - default timeout too short

Traces > Details

Timeline

Raw data

Method	Response	Duration	Age	ID
--	202	3.1 min	18.3 hr (2019-10-06 01:19:46 UTC)	1-5d994132-180c464d7a4721357dd14601

Name	Res.	Duration	Status	0.0ms	20s	40s	1.0m	1.3m	1.7m	2.0m	2.3m	2.7m	3.0m	3.3m
------	------	----------	--------	-------	-----	-----	------	------	------	------	------	------	------	------

▼ **kindle-filter-email** AWS::Lambda

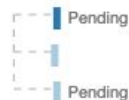
kindle-filter-email	202	26.0 ms	✓
Dwell Time	-	50.0 ms	✓
Attempt #1	200	4.2 sec	⚠
Attempt #2	200	3.0 sec	⚠
Attempt #3	200	3.7 sec	⚠



▼ **kindle-filter-email** AWS::Lambda::Function

kindle-filter-email	-	Pending	⌚
kindle-filter-email	-	Pending	⌚
Initialization	-	494 ms	✓
Invocation	-	Pending	⌚

-- Pending



# Demo comparisons over time

All traces in the group ⓘ 23 traces in the group. [Show in charts](#) ⓘ

Complete 100% scanned (found 23 traces)

## Retrieved traces ⓘ

23 traces

## Filtered trace set A ⓘ

To add a filter, click and drag one of the charts below or click one of the table rows.

[+ Compare](#)  
(Copy filter trace set A)

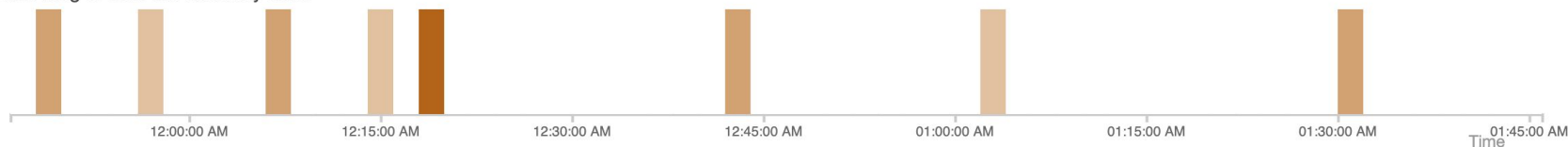
## Response time distribution ⓘ

Click and drag to filter the traces by response time.



## Time series activity ⓘ

Click and drag to filter the traces by time.



# Demo deployment with fargate cli

## Makefile

### run:

```
fargate task run app --subnet-id ${SUBNET} --security-group-id ${SG} --task-role ${ROLE}
```

### stop:

```
fargate task stop app
```

### info:

```
$(eval IP:= $(shell fargate task info app --no-color --no-emoji | grep IP | sed -n 's/^.*: //p'))
```

```
@curl http://${IP}:8080
```

# Lessons learned

- Instrument all external service calls
- Instrument all application compute bound functions
- Name segments and subsegments in a fashion that makes sense later
- Name segments in a manner that makes them easy to distinguish or filter on when you have multiple parallel environments, example is cluster name prefixes
- Make sampling level easily configurable to keep cost down
- Plan to include additional metadata in traces to allow more filtering later
- Decide what you want to be a segment, container start or request processing start?
- Filter out health check path using local sampling configuration

# Links

- <https://github.com/dougtoppin/presentation-aws-xray>
- <https://aws.amazon.com/xray/>
- <https://aws.amazon.com/xray/pricing/>
- <https://github.com/aws-labs/fargatecli>
- <https://read.iopipe.com/how-far-out-is-aws-fargate-part-2-e87088f3ee26>
- <https://docs.aws.amazon.com/xray/latest/devguide/xray-daemon-ecs.html>