

The road to observable systems

January 2022



cinch

Hi, I'm Toli.



Stumbled into software engineering



Worked in the software industry for 10+ years



Experience across a few industries (Logistics, Energy, Automotive)



Touched all traditional tech stacks

My current role



Principal Practice Engineer at cinch



Advocating for good DevOps and oily patterns & practices

**Cinch offers the UK's
biggest range of cars
to buy entirely online**

**We remove the faff
from finding, buying
and owning a car**

We don't just let
people nail it when
they buy a new car

We make sure they
absolutely cinch it.

Hence the name*

cinch¹ [sinch] [SHOW IPA](#)  

See synonyms for: [cinch](#) / [cinched](#) / [cinching](#) on Thesaurus.com



High School Level

noun

- 1 a strong girth used on stock saddles, having a ring at each end to which a strap running from the saddle is secured.
- 2 a firm hold or tight grip.

3 *Informal.*

- a something sure or easy:

This problem is a cinch.

- a person or thing certain to fulfill an expectation, especially a team or contestant certain to win a sporting event:

The Giants are a cinch to win Sunday's game.

*<https://www.dictionary.com/browse/cinch>

The road to observable systems



Why did we focus so much on observability?



What impact did this have on our organization and engineering practices?

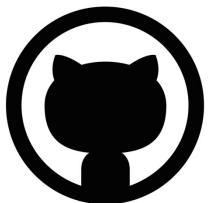


What lessons did we learn along the way?

Tech stack



TypeScript



GitHub Actions



Engineering principles



Serverless, Event Driven Architecture

Build serverless, asynchronous, resilient systems to serve team autonomy



Test Driven Development

Automate as much testing as possible



Flow

Fast flow, small batches, frequent changes



Feedback

Build observable systems for optimal feedback

Engineering principles



Serverless, Event Driven Architecture

Build asynchronous, resilient systems to serve team autonomy



Test Driven Development

Automate as much testing as possible



Flow

Fast flow, small batches, frequent changes



Feedback

Build observable systems for optimal feedback

Engineering principles



Serverless, Event Driven Architecture

Build asynchronous, resilient systems to serve team autonomy



Test Driven Development

Automate as much testing as possible



Flow

Fast flow, small batches, frequent changes



Feedback

Build observable systems for optimal feedback

Engineering principles



Serverless, Event Driven Architecture

Build asynchronous, resilient systems to serve team autonomy



Test Driven Development

Automate as much testing as possible



Flow

Fast flow, small batches, frequent changes



Feedback

Build observable systems for optimal feedback

requests

Count

10/08 10/08

10/09 10/09

10/10 10/10

10/11 10/11

10/12 10/12

Count 5XXError 4XXError



latency

Serverless scales beautifully



Milliseconds

10/08 10/08

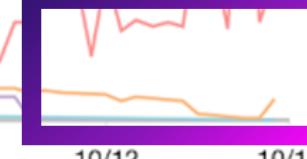
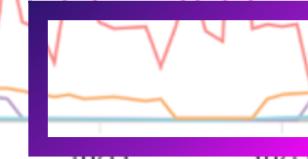
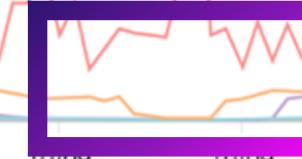
10/09 10/09

10/10 10/10

10/11 10/11

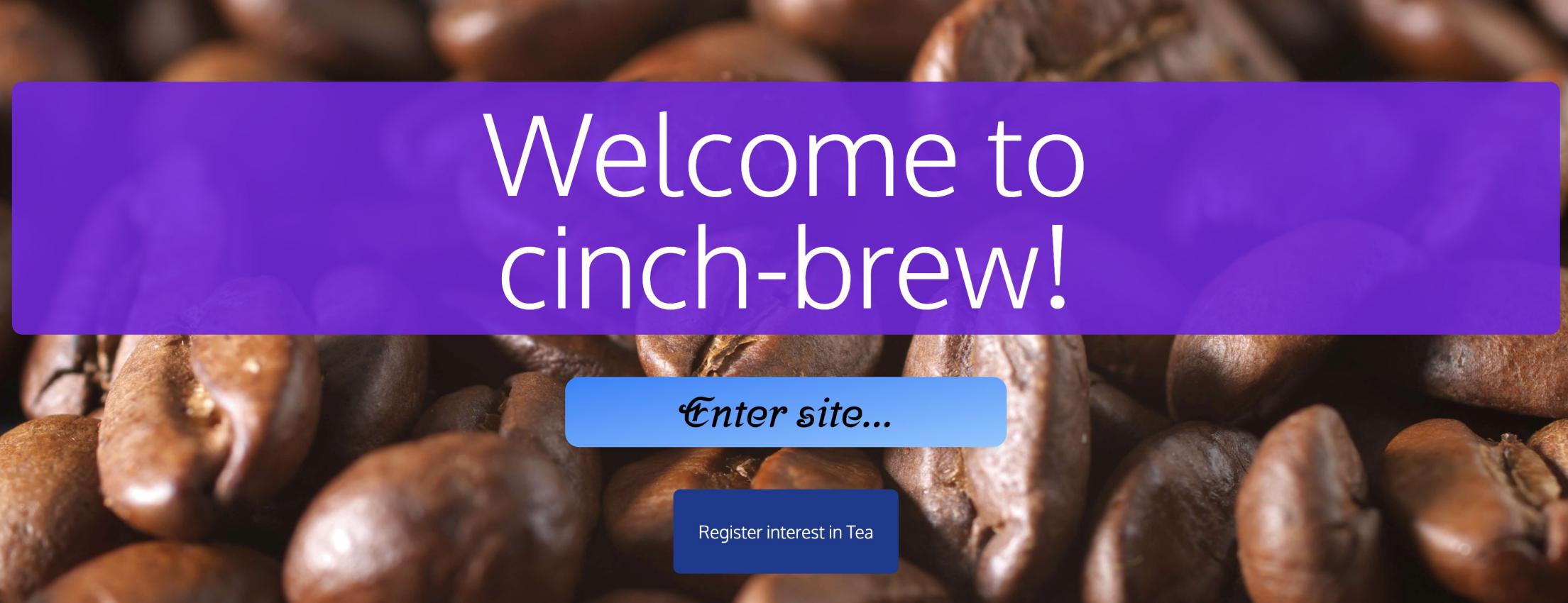
10/12 10/12

p95 Average Maximum p99



What does serverless look like?

This is not real, don't get excited!*



Welcome to
cinch-brew!

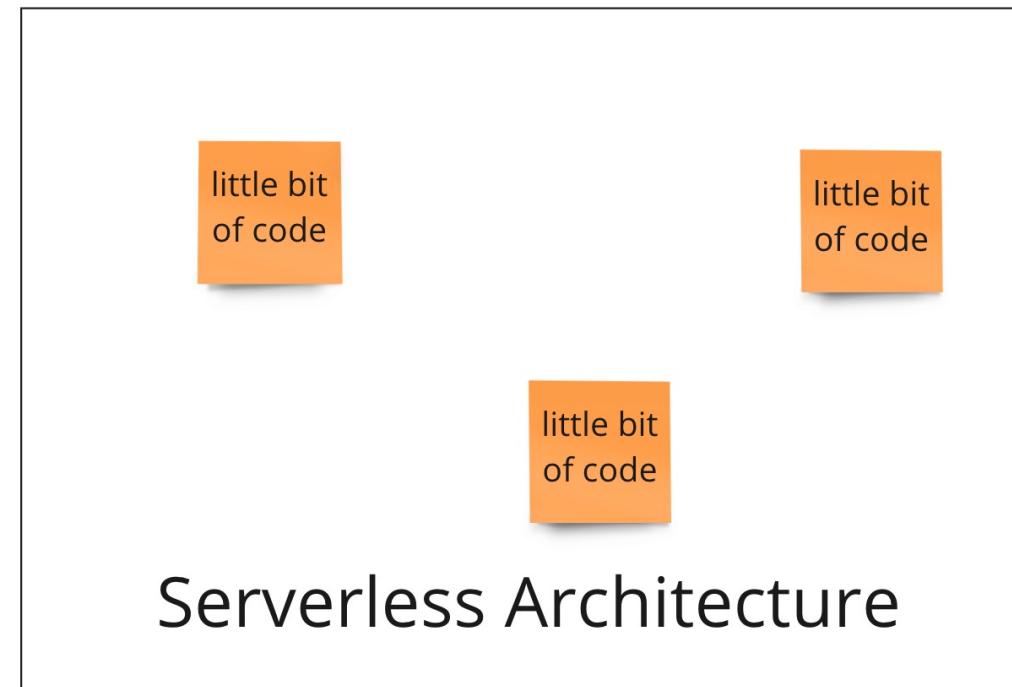
Enter site...

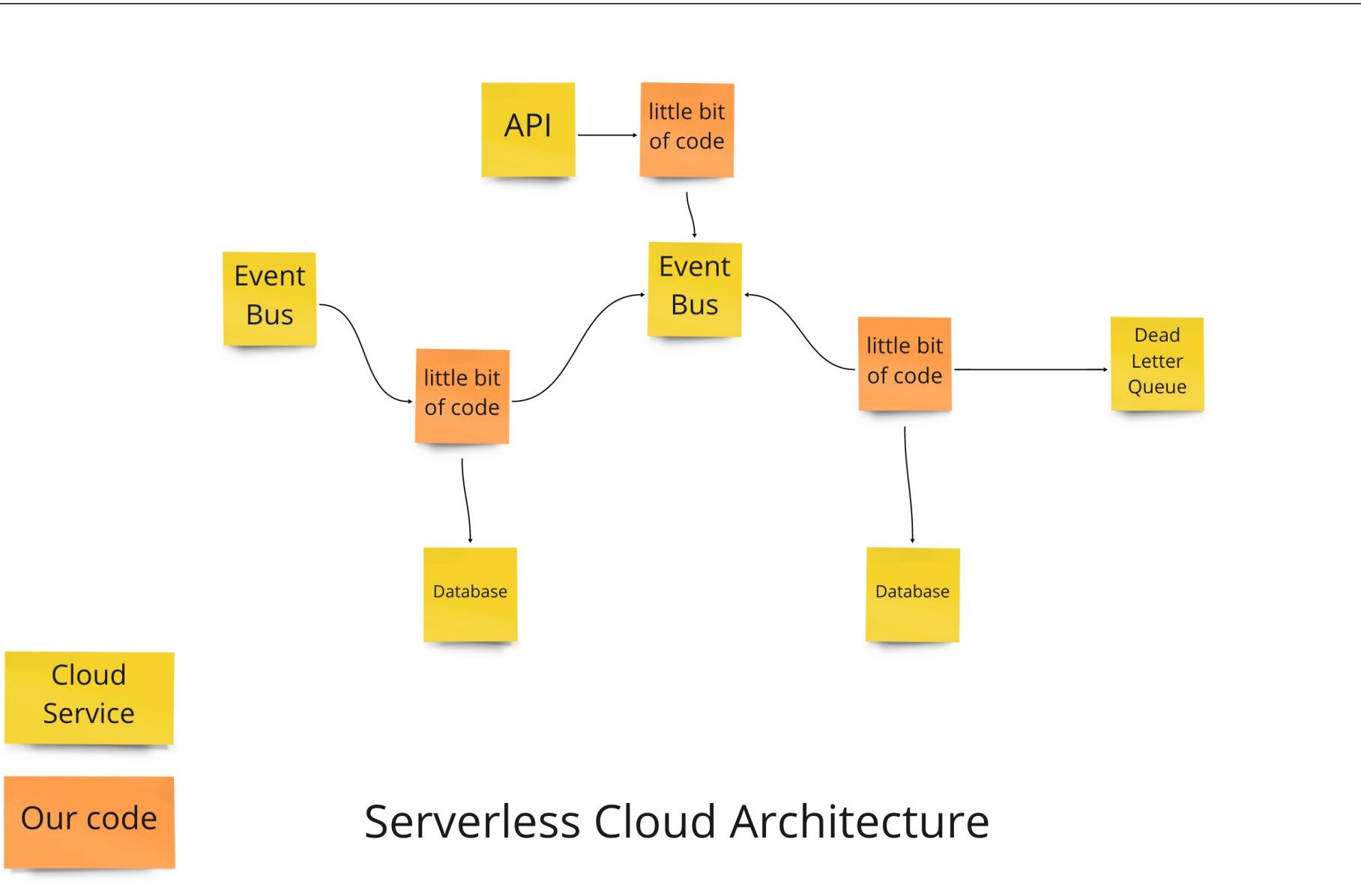
Register interest in Tea

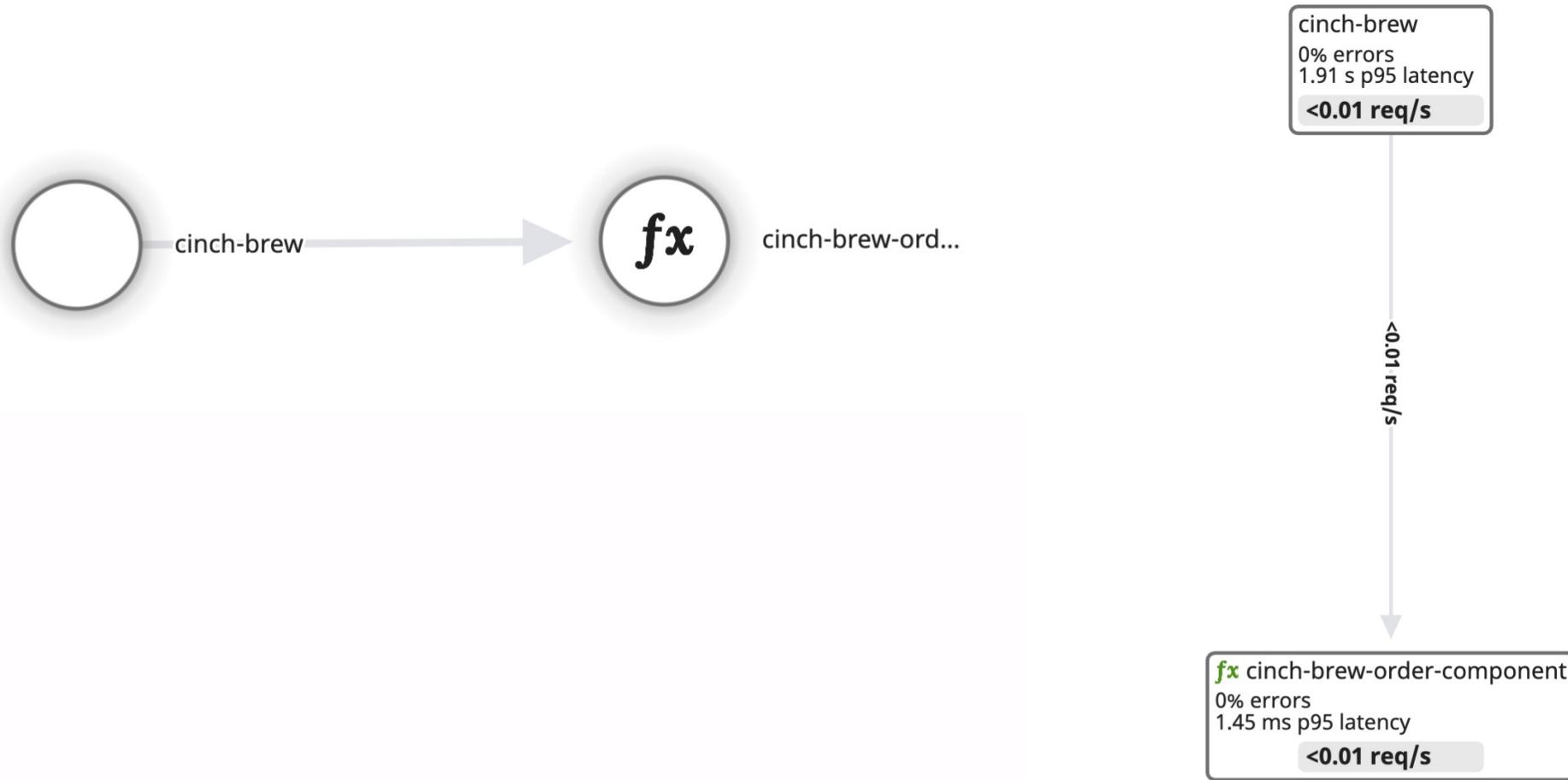
*We're going to use a fictional software system to take you on our serverless observability journey - it's called 'cinch brew'

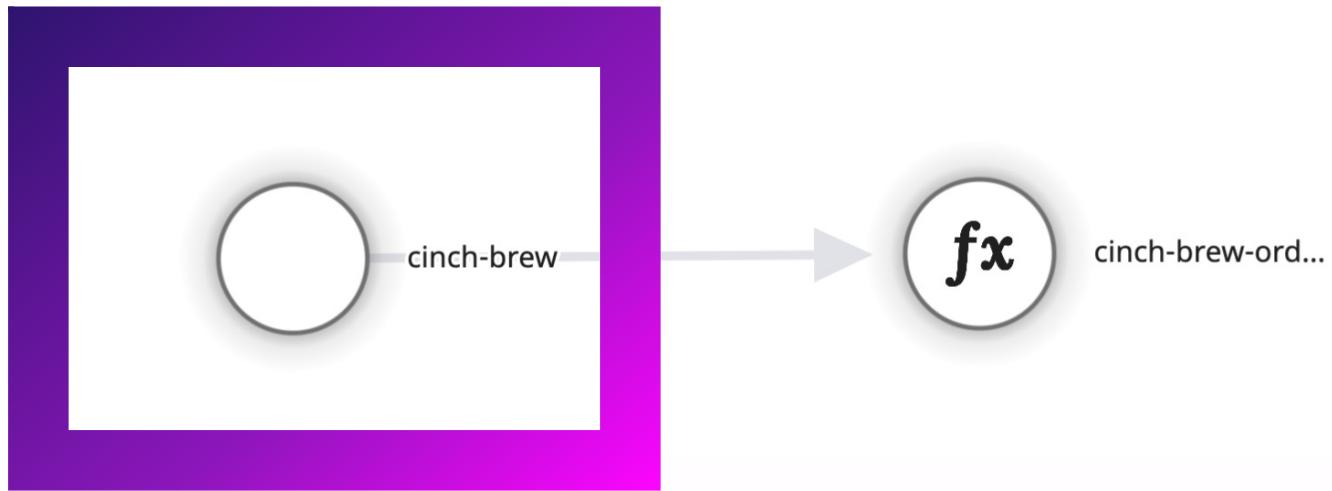


Little bits of code here & there

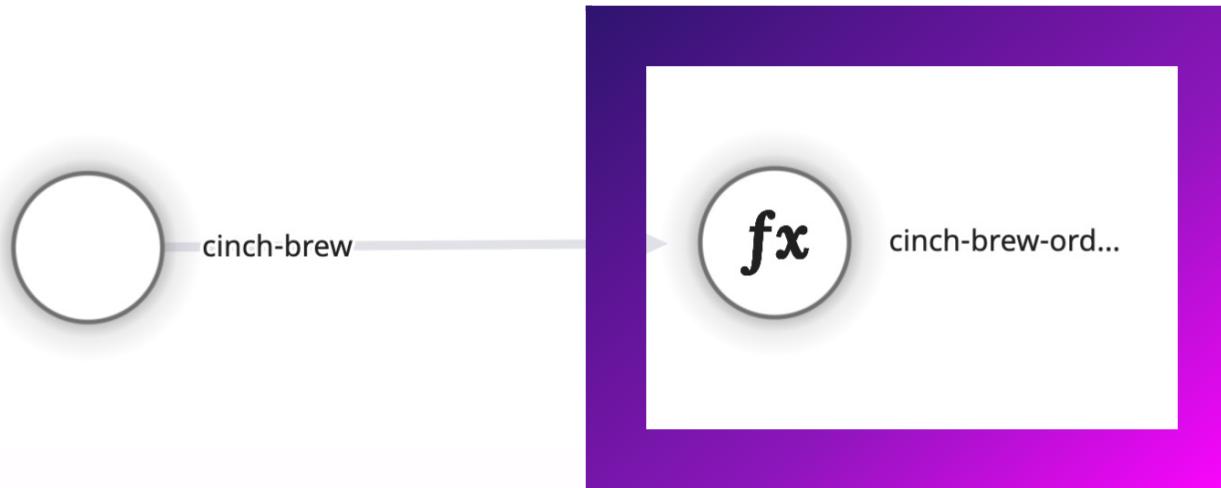






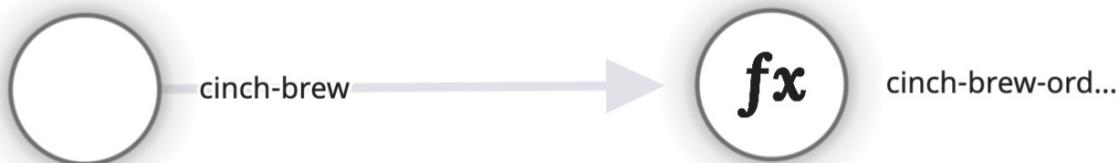


Frontend

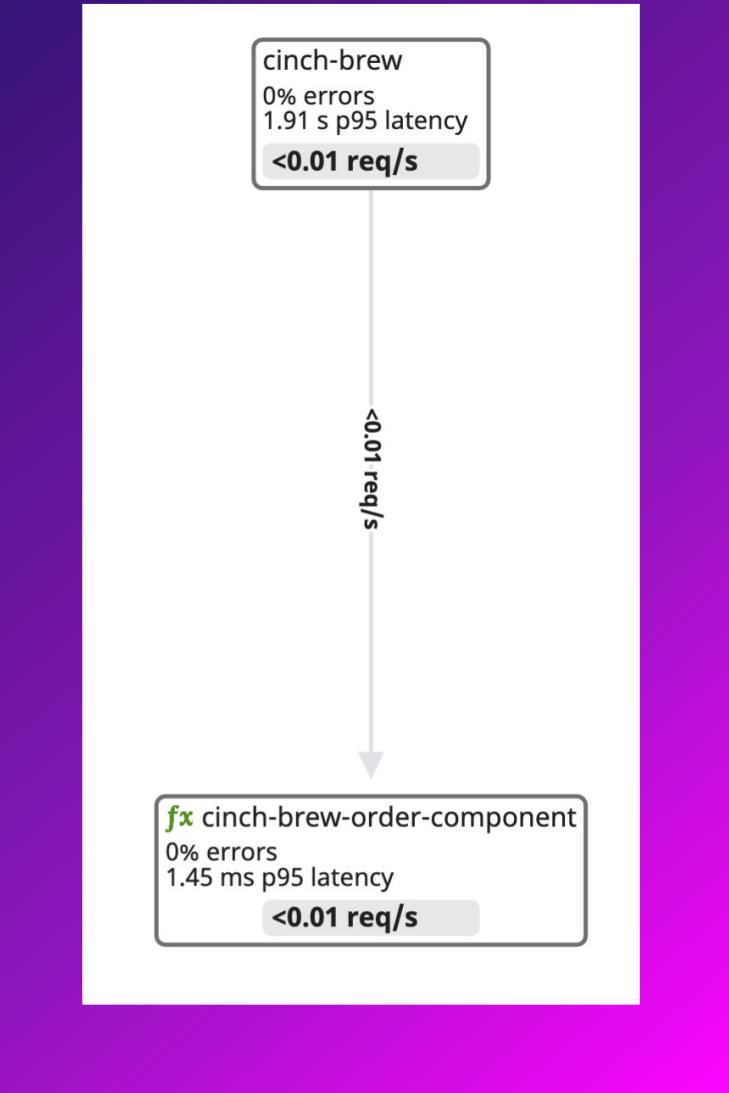


Backend

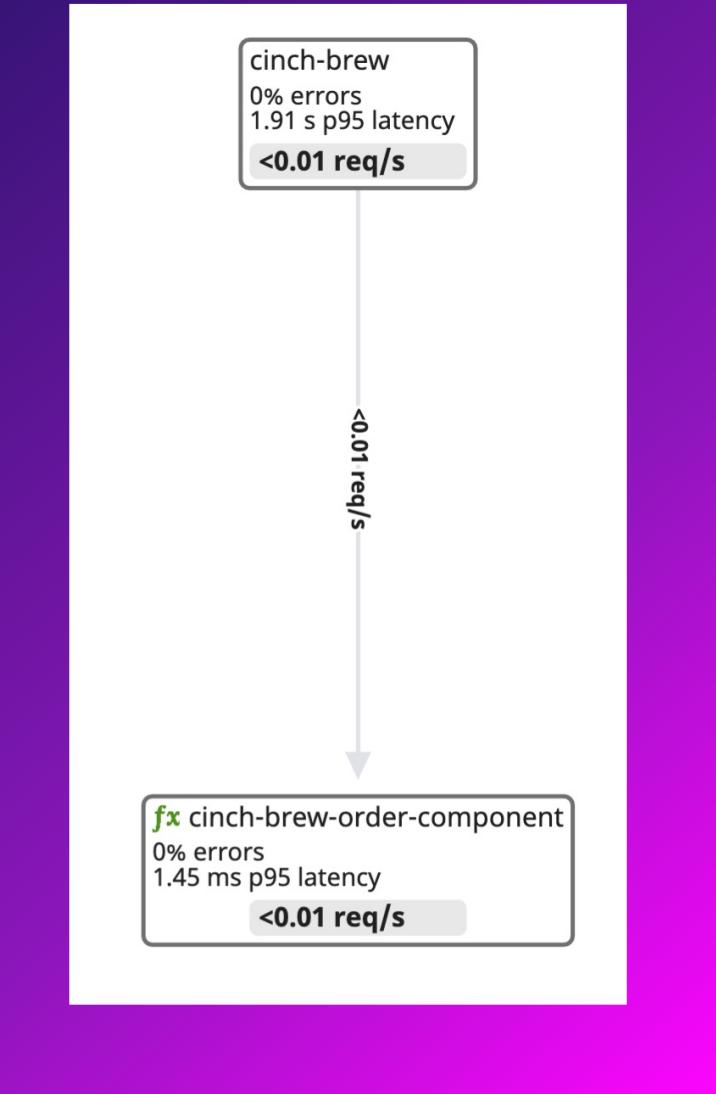
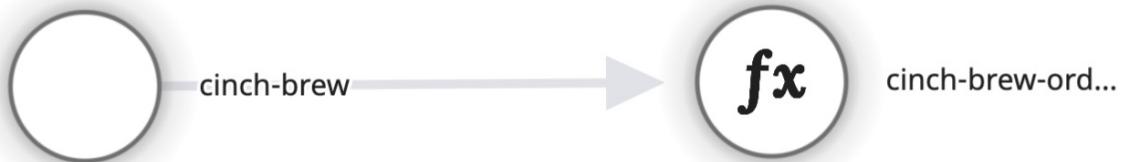
Requests, Errors, Duration



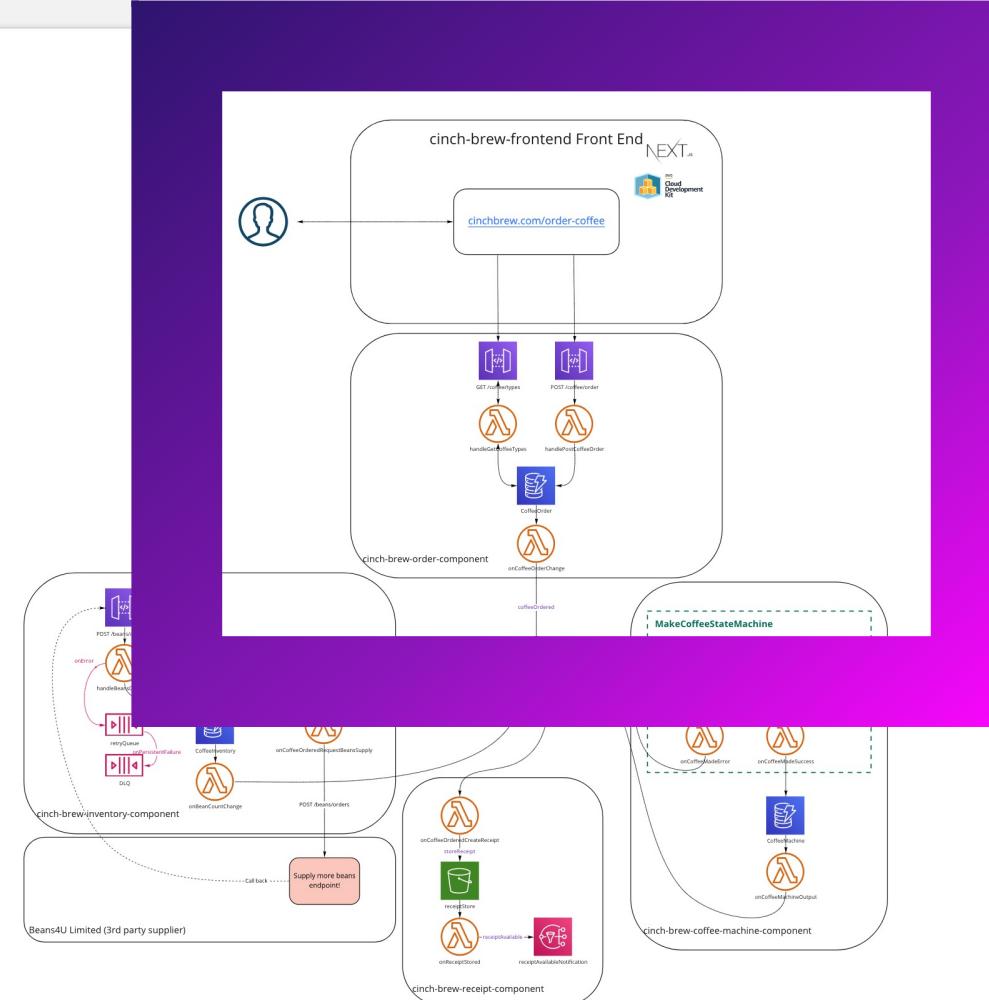
cinch-brew-ord...



Serverless observability is simple, right?



- Legend**
- API Gateway
 - Lambda function
 - DynamoDB table
 - SNS topic
 - S3 bucket
 - EventBridge event bus
 - SQS



In (our fictitious) reality,
systems are more complex

There is this bit too

Legend

API Gateway



Lambda function



DynamoDB table



SNS topic



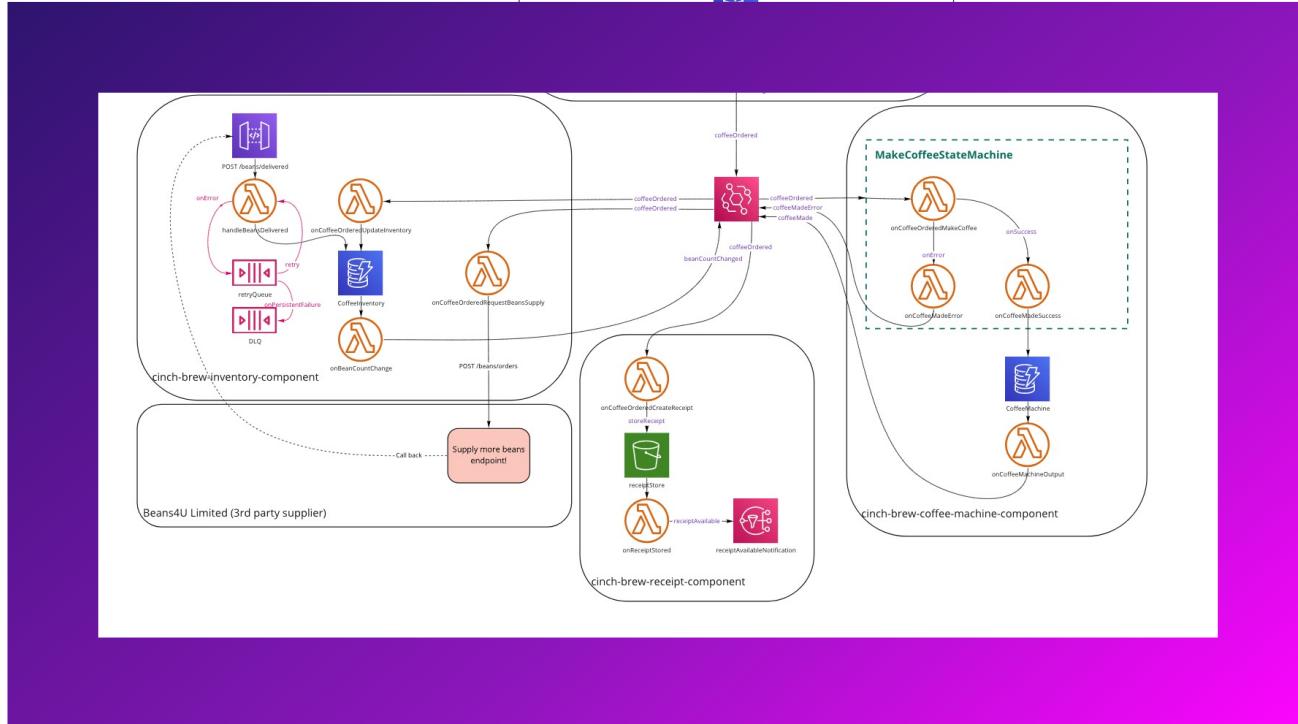
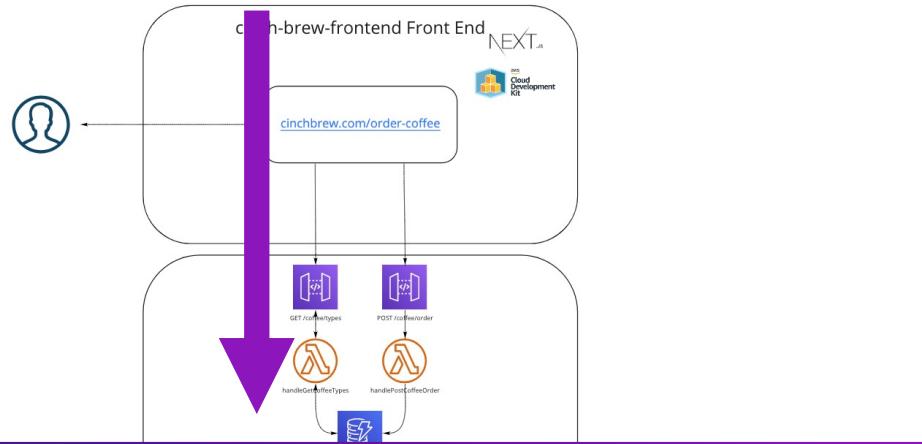
S3 bucket



EventBridge event bus



SQS



Serverless in production



Can you see faff creeping in?

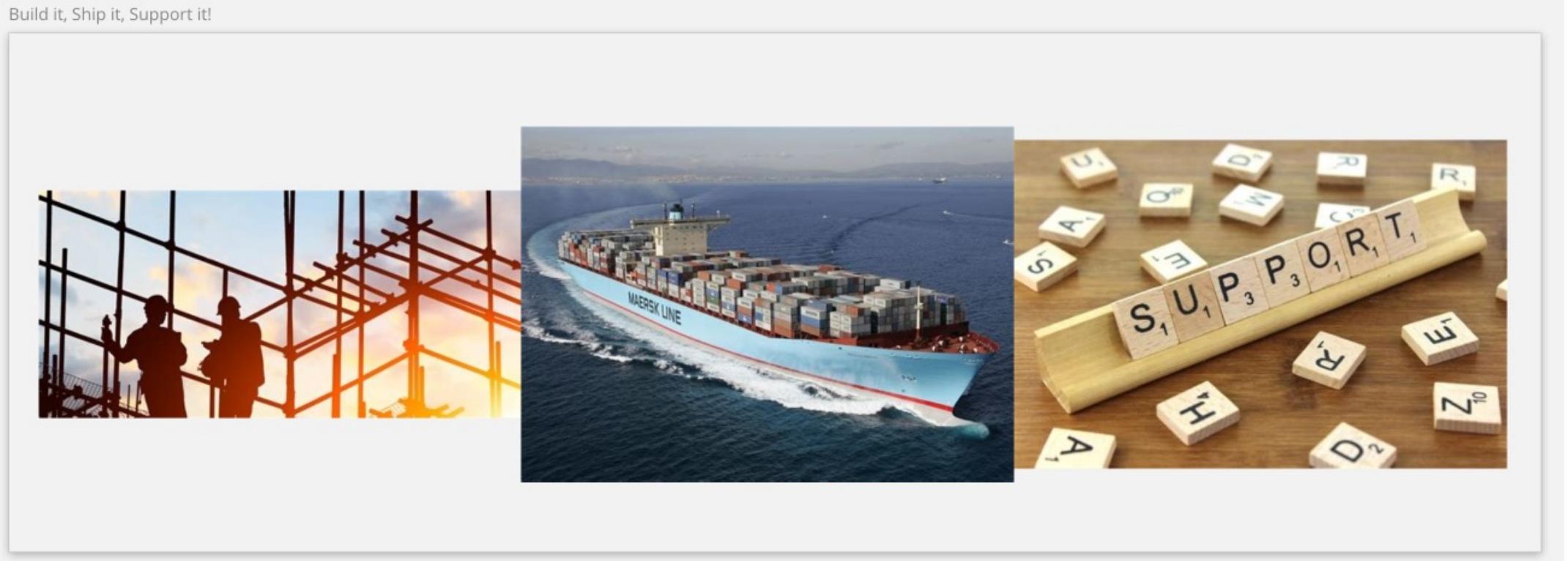


We don't do faff.

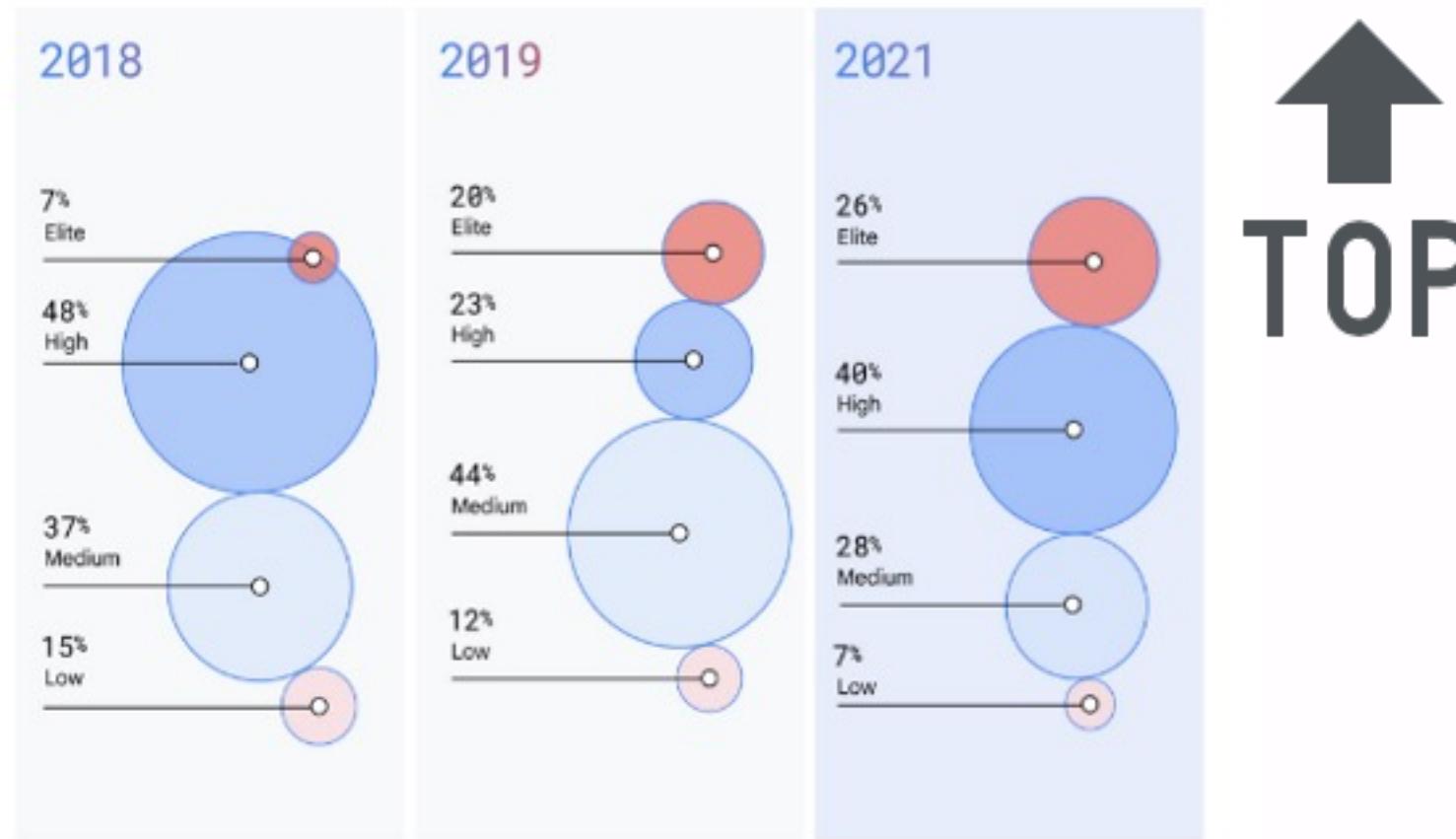


How do we know our services are keeping their promises?

Our model: Build it, Ship it, Support it



The industry has advanced



Serverless means we are elite, right?

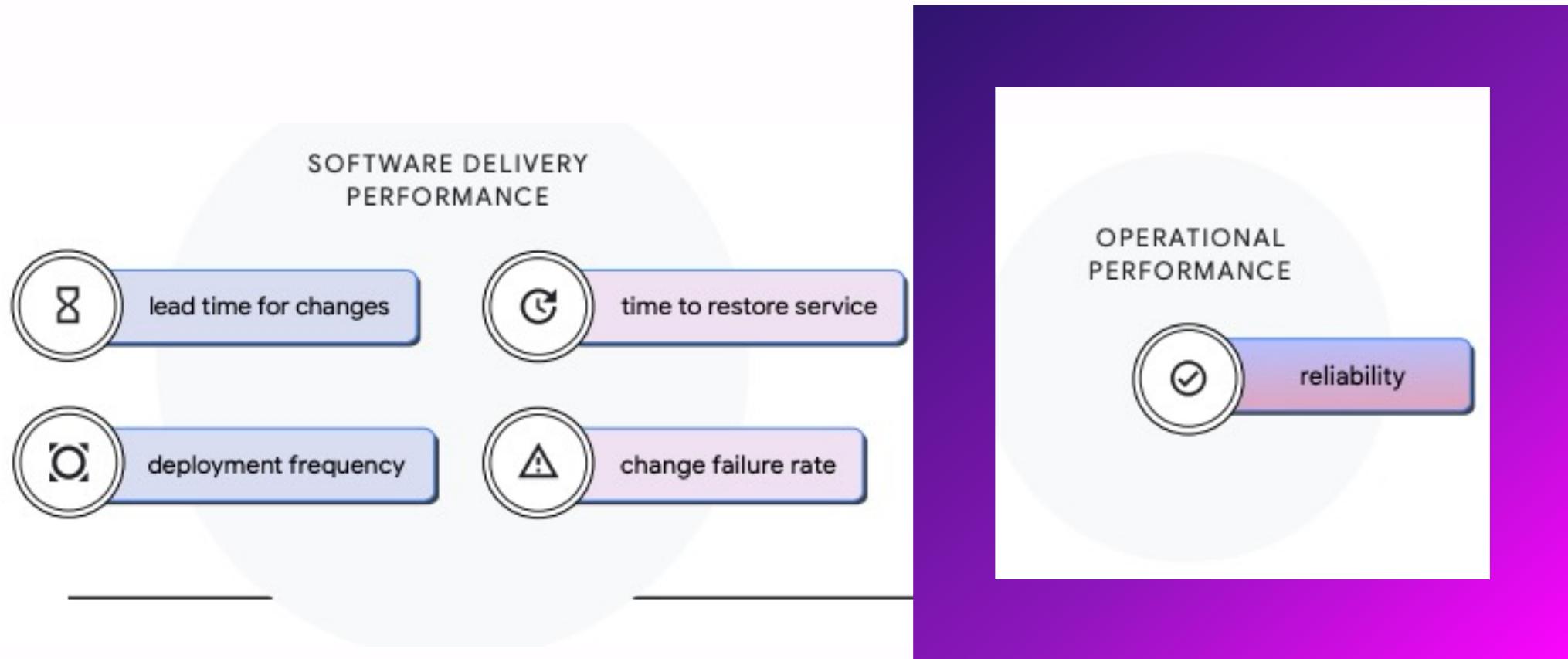
Build it, Ship it, Support it!



There is a fifth Accelerate metric



The fifth metric: Reliability*



* Reliability does not equal observability. But serverless helps. And observability helps too.

Reliability = serverless + o11y*

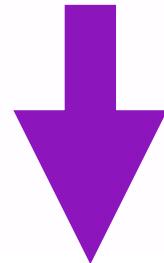
*observability



Glossary

Term	Definition
Observability	<p>“Observability” is being able to fully understand our systems. In control theory, observability is a measure of how well internal states of a system can be inferred from knowledge of its external outputs.</p> <p>Honeycomb coined the term in software from control theory and has popularised it across the industry.</p>

Charity Majors, Honeycomb CEO



<https://charity.wtf/2020/03/03/observability-is-a-many-splendored-thing/>

CHARITY.WTF

OBSERVABILITY IS A MANY-SPLENDORED DEFINITION

Last weekend, [@swyx](#) posted a great little [primer](#) to instrumentation titled “Observability Tools in JavaScript”. A friend sent me the link and suggested that I might want to respond and clarify some things about observability, so I did, and we had a great conversation! Here is a lightly edited transcript of my [reply tweet storm](#).

First of all, confusion over terminology is understandable, because there are some big players out there actively trying to confuse you! Big Monitoring is indeed actively trying to define observability down to “metrics, logs and traces”. I guess they have been paying attention to the interest heating up around observability, and well... they have metrics, logs, and tracing tools to sell? So they have hopped on the bandwagon with some undeniable zeal.

But metrics, logs and traces are just data types. Which actually has nothing to do with observability. Let me explain the difference, and why I think you should care about this.

“OBSERVABILITY? I DO NOT THINK IT MEANS WHAT YOU THINK IT MEANS.”



Observability is a borrowed term from mechanical engineering/control theory. It means, paraphrasing: “can you understand what is happening inside the system — can you understand ANY internal state the system may get itself into, simply by asking questions from the outside?” We can apply this concept to software in interesting ways, and we may end up using some data types, but that’s putting the cart before the horse.

It’s a bit like saying that “database replication means structs, longints and elegantly diagrammed English sentences.” Er, no.. yes.. missing the point much?

cinch

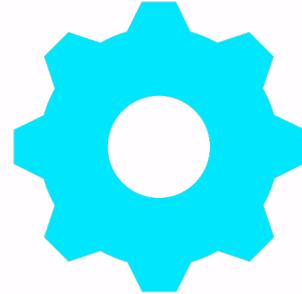
**Observable systems are built
by teams**

Instrument, don't log

Instrumentation

"Instrumentation" is the practice of systematically configuring our codebase to emit telemetry data that provide insights into how our code is exercised by data in production.

You can auto-instrument, be more ambitious



Auto-instrumentation



Custom instrumentation

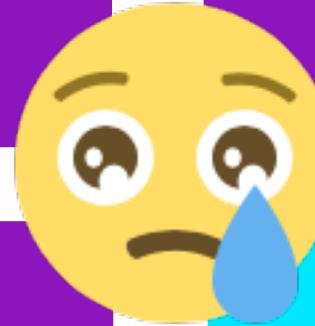
Sadly, there are many telemetry data types

Traces

Metrics

Logs

RUM Events



We use logs, but minimally

Traces

high context &
traceability

Metrics

high value signals
with limited context

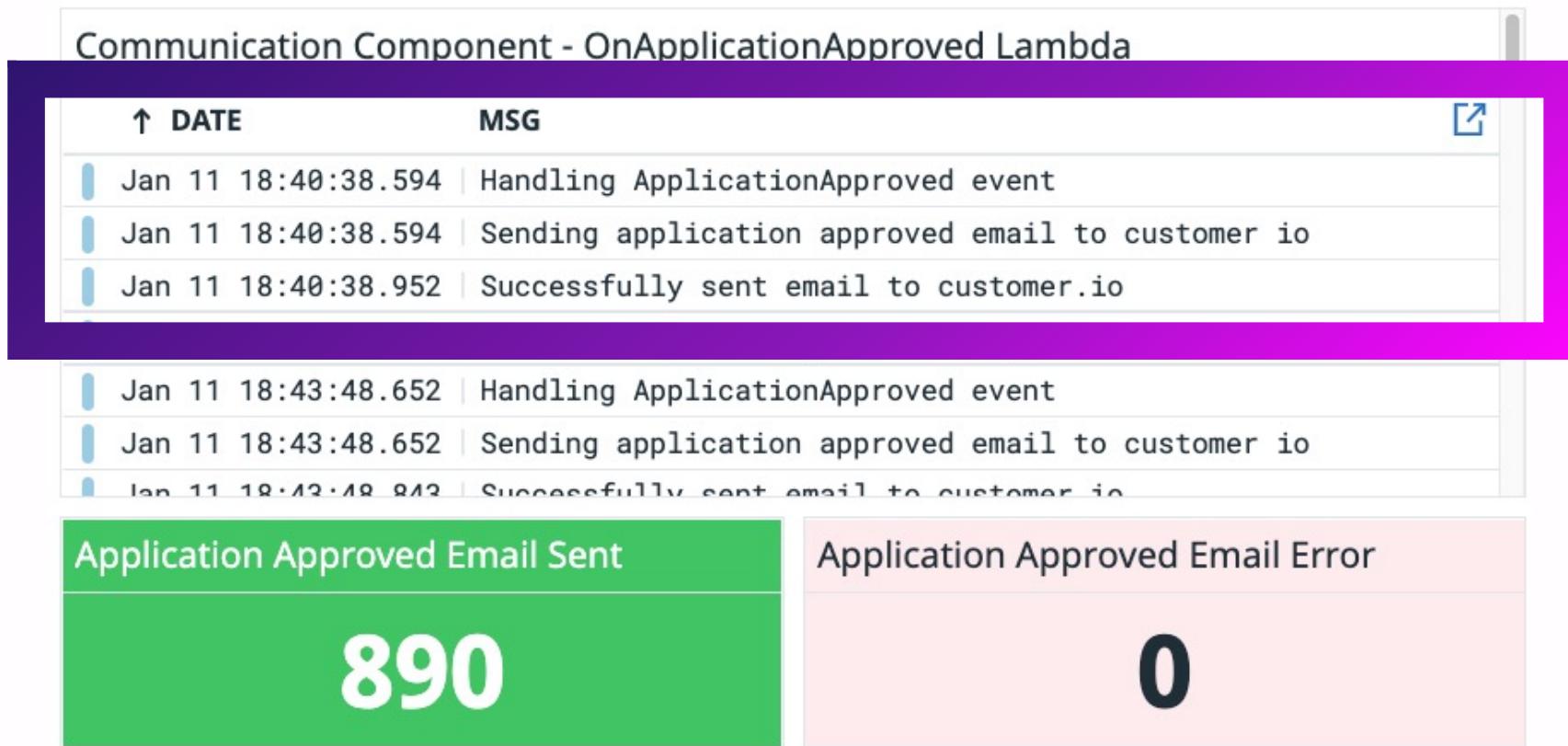
Logs

high context with
free text included &
links to traces

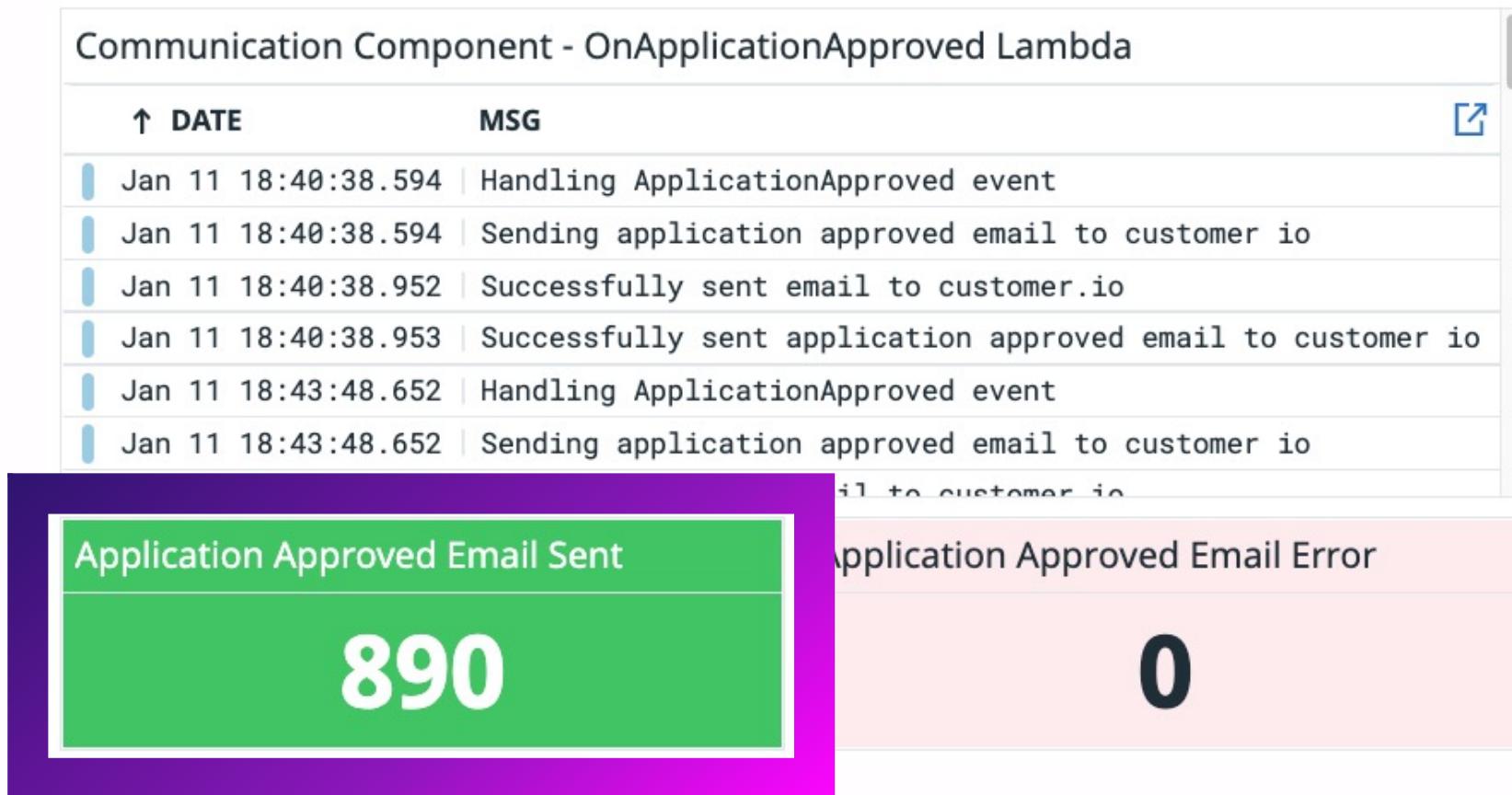
RUM Events

high context & links
to traces

Logs are useful for non techies



They can be turned into numbers too



Our favourite type: traces (and spans)

Traces

high context &
traceability

Metrics

high value signals
with limited context

Logs

high context with
free text included &
links to traces

RUM Events

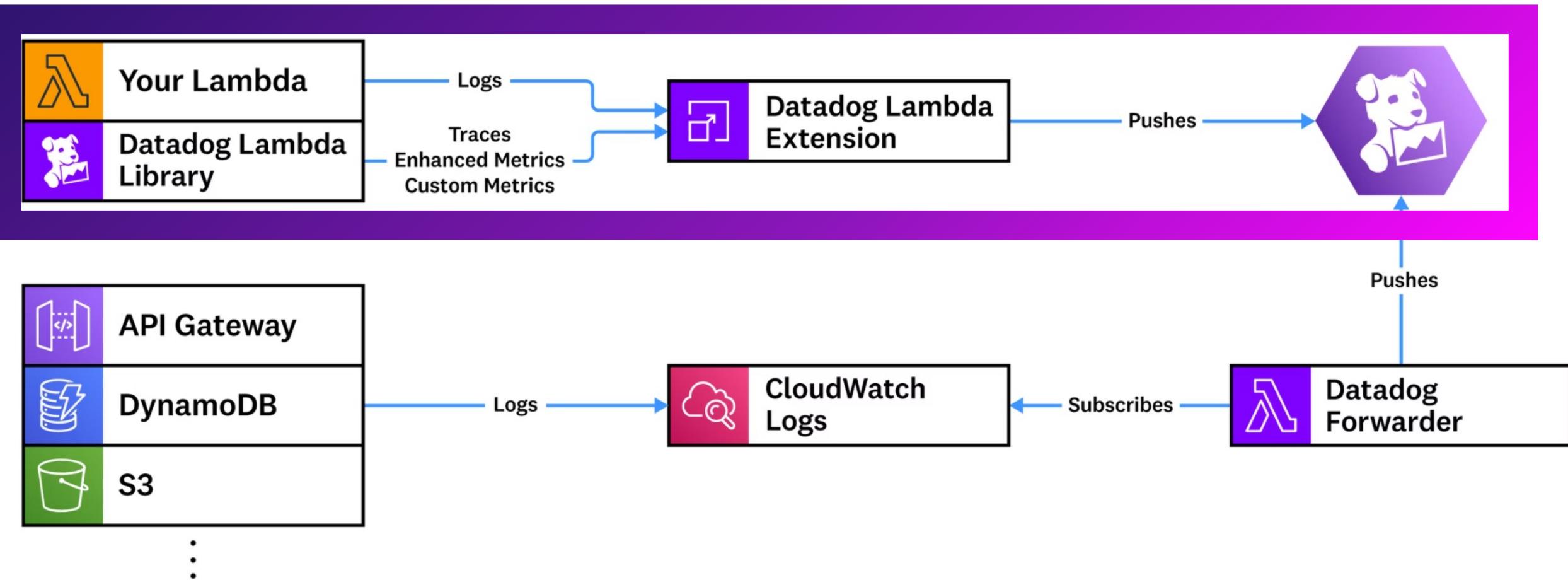
high context & links
to traces

Auto-instrumentation

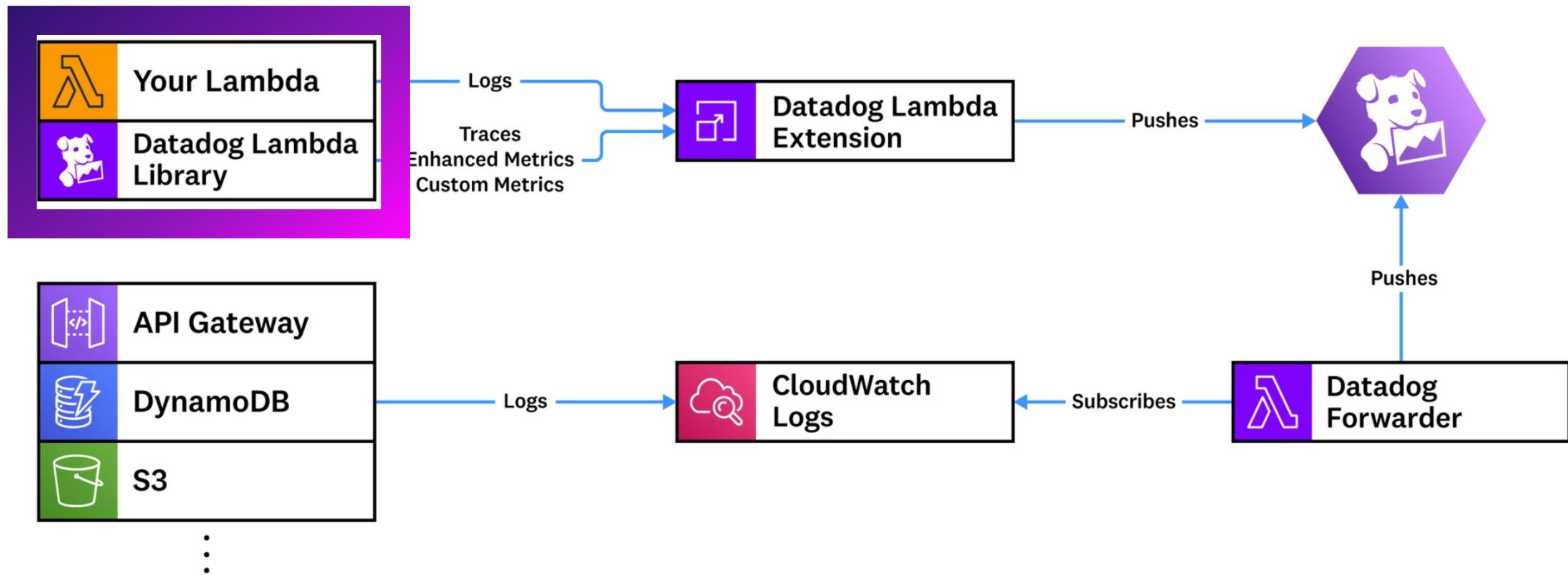
```
64      const datadog = new Datadog(this, 'Datadog', {  
65        addLayers: true,  
66        apiKey: props.datadogApiKey,  
67        extensionLayerVersion: 12,  
68        site: 'datadoghq.eu',  
69        nodeLayerVersion: 64,  
70      });
```

```
153      datadog.addLambdaFunctions([listCoffeeBeanTypes]);
```

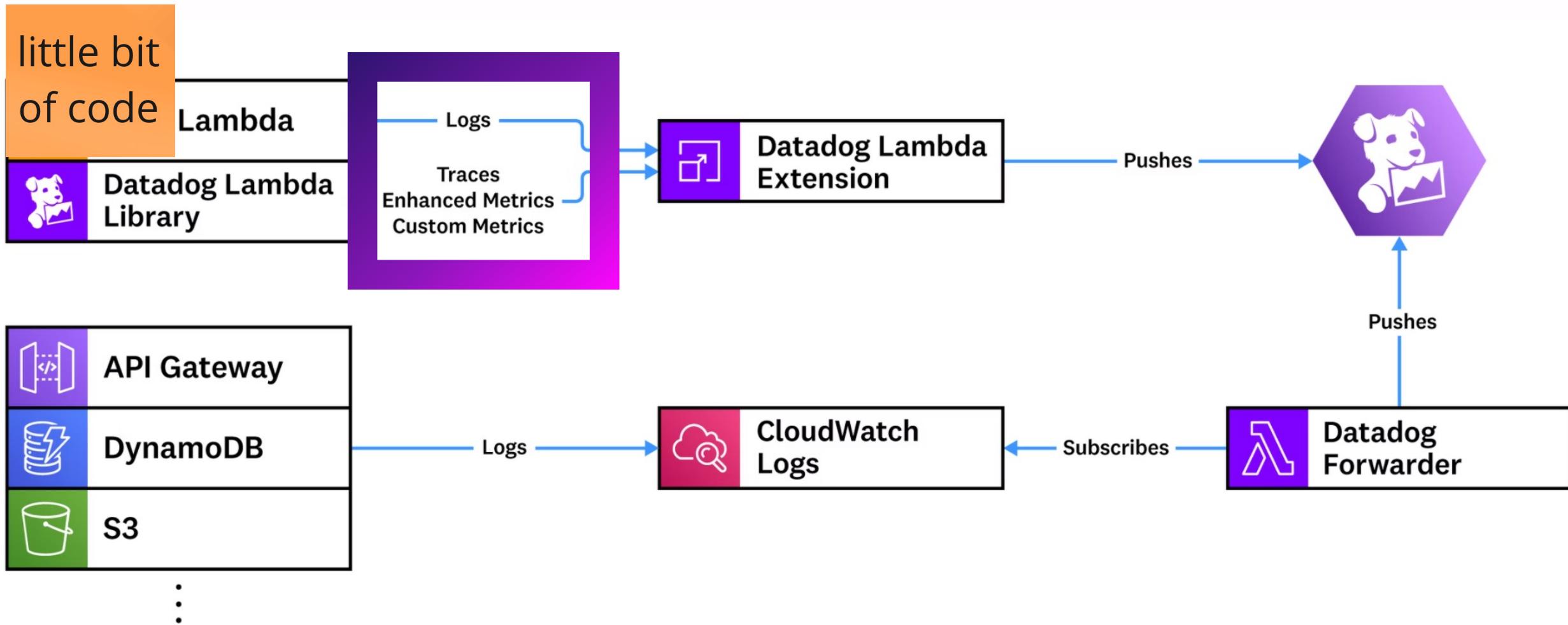
Hook into lambda runtime



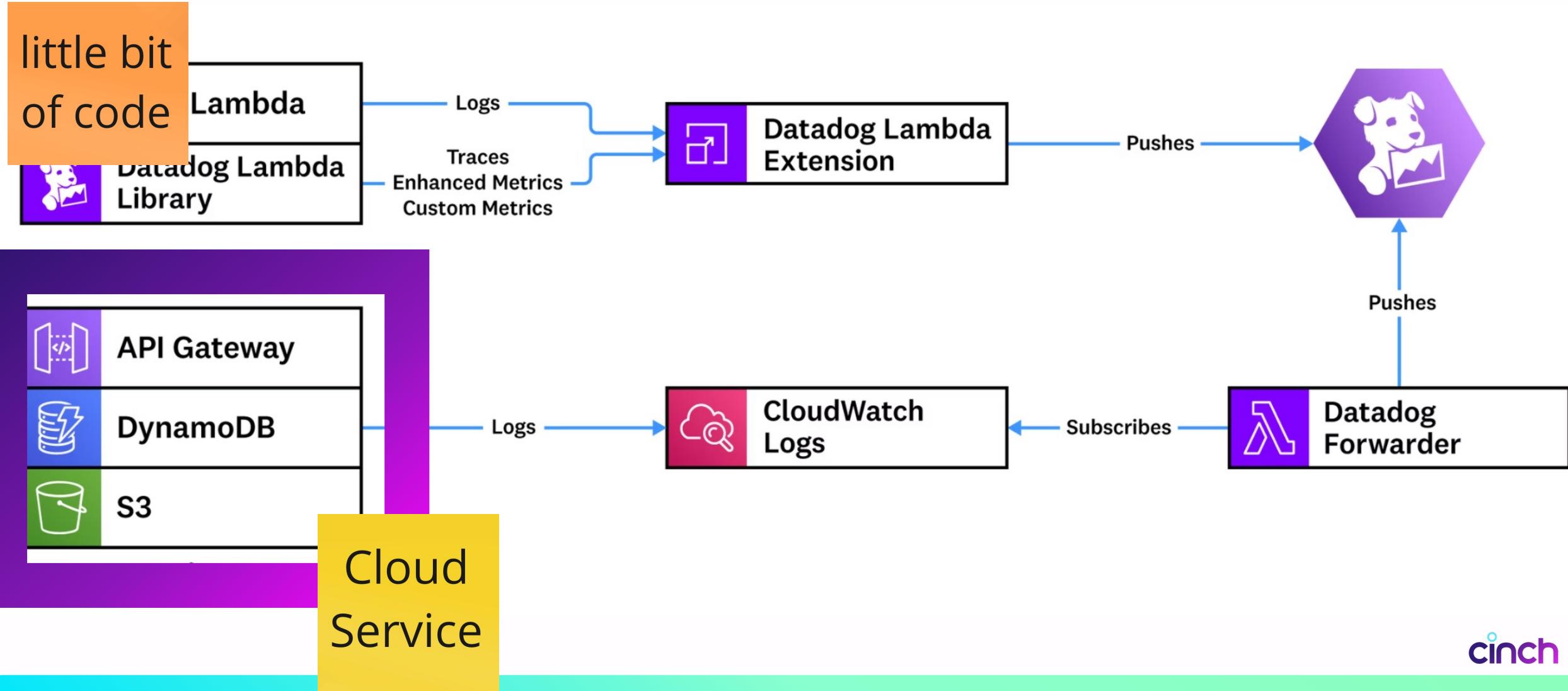
We extend our lambdas



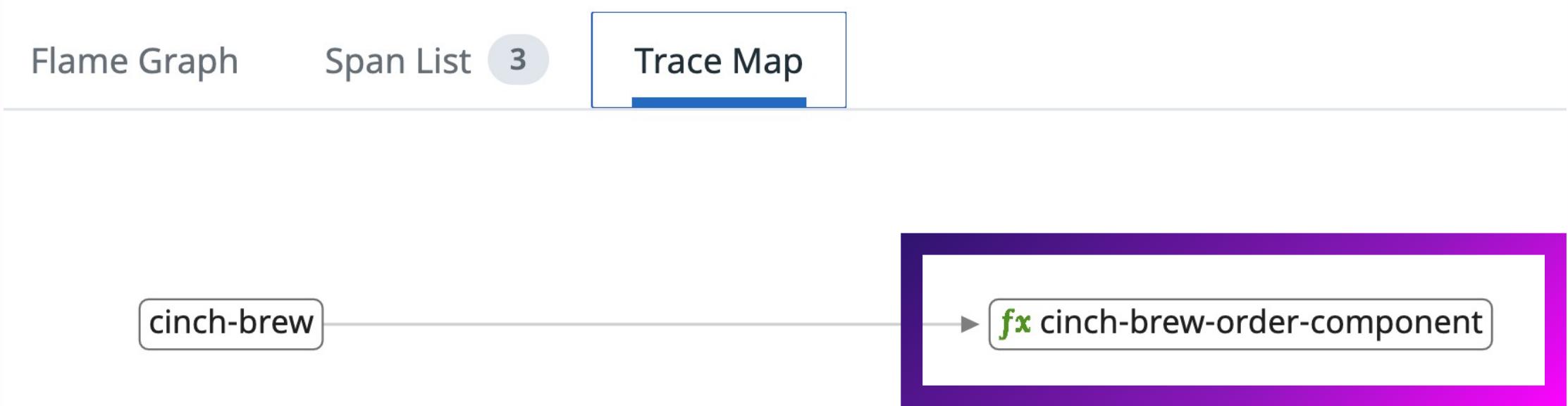
Who needs CloudWatch?



We pull logs for Cloud services

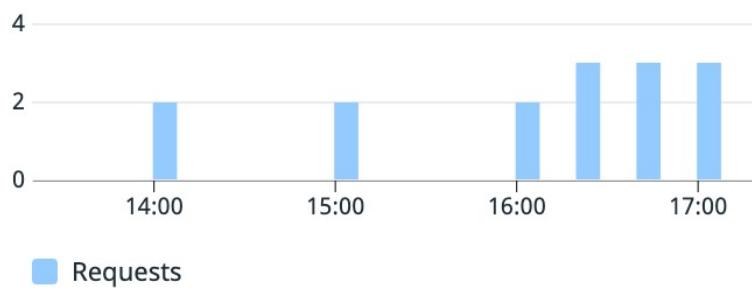


Out of the box trace map



Out of the box top level metrics

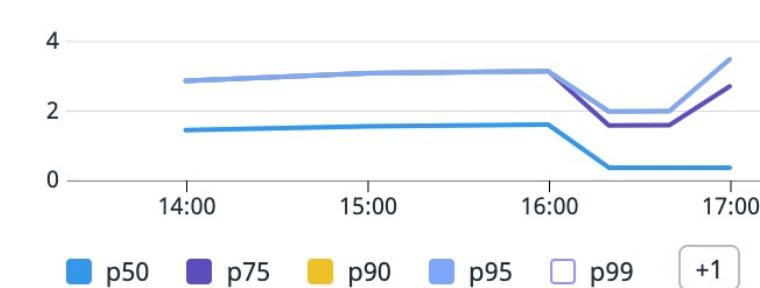
Requests 15 total (< 0.1 req/s)



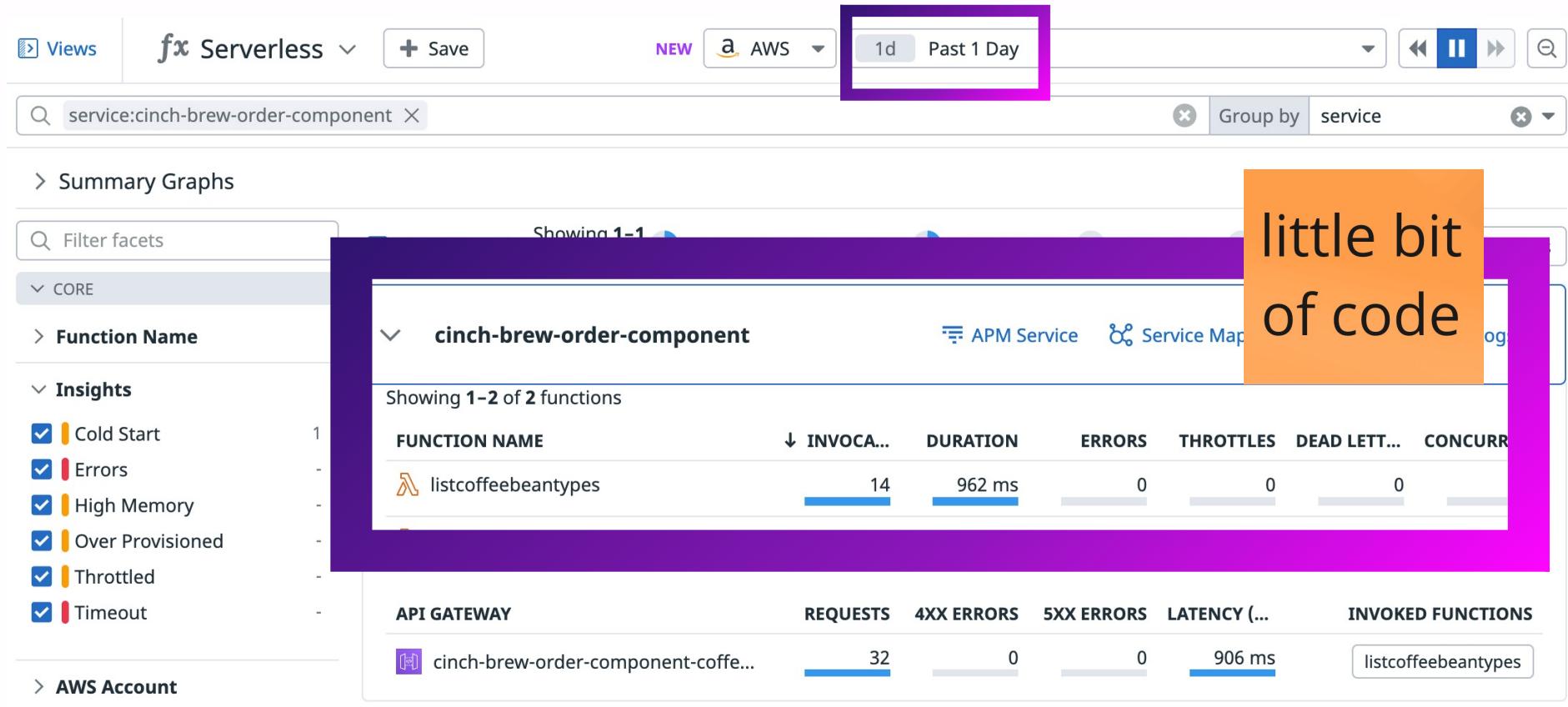
Errors



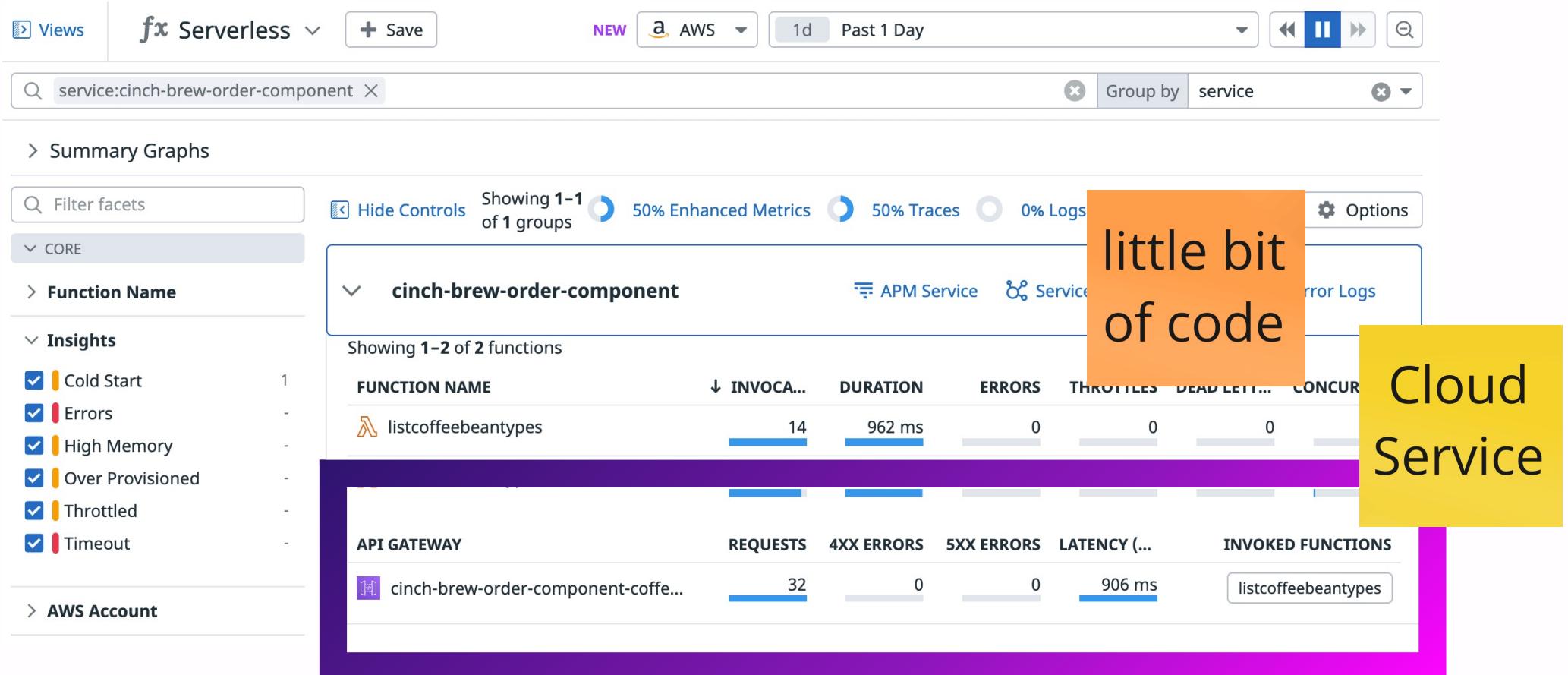
Latency



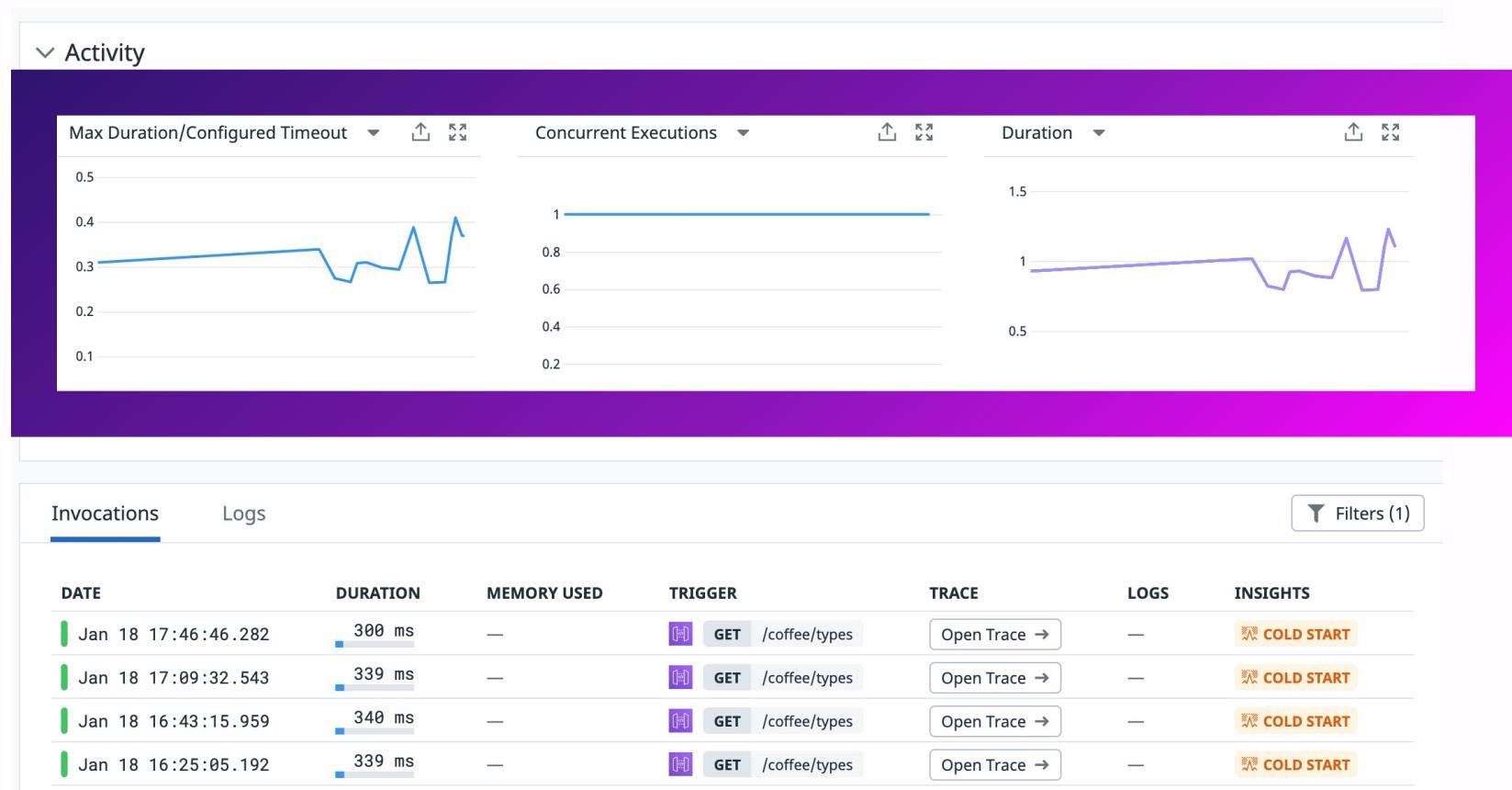
Out of the box stack overview



Cloud service too!



Serverless metrics



All your invocations

Activity

Max Duration/Configured Timeout

Concurrent Executions

Duration

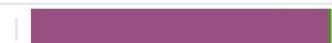
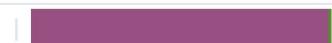


Invocations Logs

Filters (1)

DATE	DURATION	MEMORY USED	TRIGGER	TRACE	LOGS	INSIGHTS
Jan 18 17:46:46.282	300 ms	—	GET /coffee/types	Open Trace →	—	COLD START
Jan 18 17:09:32.543	339 ms	—	GET /coffee/types	Open Trace →	—	COLD START
Jan 18 16:43:15.959	340 ms	—	GET /coffee/types	Open Trace →	—	COLD START
Jan 18 16:25:05.192	339 ms	—	GET /coffee/types	Open Trace →	—	COLD START

All our spans* too!

↓ DATE	SERVICE	RESOURCE	DURATION	METHOD	STATUS CODE	LATENCY BREAKDOWN
Jan 18 17:46:46.584	cinch-brew	/coffee/types	2.23 s	GET	200	
Jan 18 17:46:46.282	fx cinch-brew-order-com...	LISTCoffeeBeanTypes	300 ms	GET	200	
Jan 18 17:09:33.358	cinch-brew	/coffee/types	3.46 s	GET	200	
Jan 18 17:09:32.543	fx cinch-brew-order-com...	LISTCoffeeBeanTypes	339 ms	GET	200	
Jan 18 17:09:32.484	🌐 cinch-brew-order-com...	POST	139 ms	POST	200	
Jan 18 16:43:16.257	cinch-brew	/coffee/types	1.96 s	GET	200	
Jan 18 16:43:15.959	fx cinch-brew-order-com...	LISTCoffeeBeanTypes	340 ms	GET	200	
Jan 18 16:43:15.922	🌐 cinch-brew-order-com...	POST	141 ms	POST	200	
Jan 18 16:25:05.334	cinch-brew	/coffee/types	1.96 s	GET	200	
Jan 18 16:25:05.192	fx cinch-brew-order-com...	LISTCoffeeBeanTypes	339 ms	GET	200	
Jan 18 16:25:05.134	🌐 cinch-brew-order-com...	POST	159 ms	POST	200	
Jan 18 16:02:27.939	cinch-brew	/coffee/types	3.11 s	GET	200	
Jan 18 16:02:27.110	fx cinch-brew-order-com...	LISTCoffeeBeanTypes	37.7 ms	GET	200	

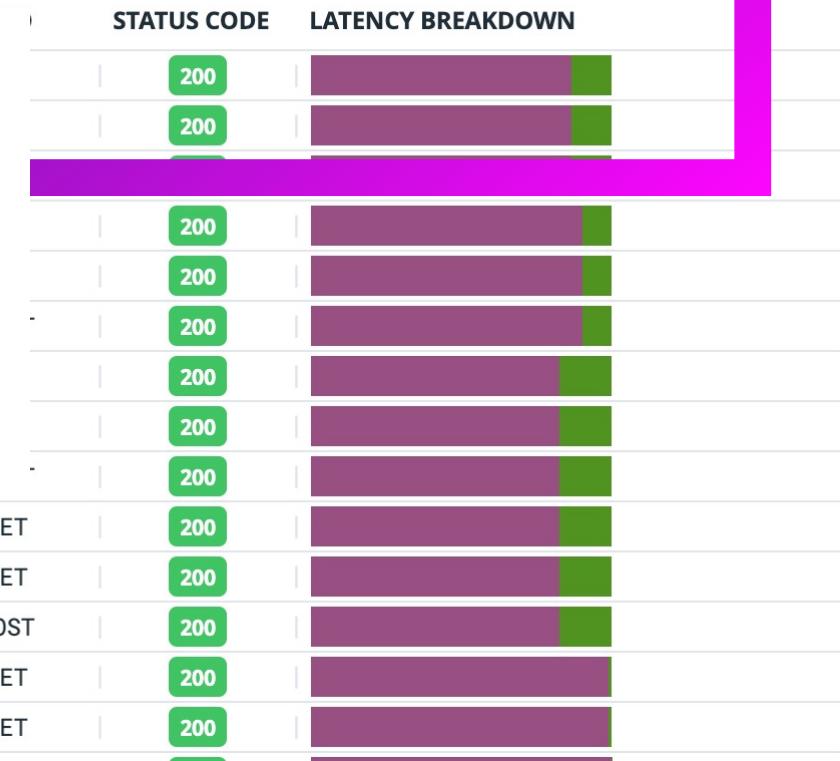
*I'm not going to get into spans, look them up



All our spans* too!

↓ DATE	SERVICE	RESOU
Jan 18 17:46:46.584	cinch-brew	/coffee
Jan 18 17:46:46.282	cinch-brew-order-com...	LISTCoffe

Jan 18 17:09:33.358	cinch-brew	/coffee
Jan 18 17:09:32.543	cinch-brew-order-com...	LISTCoffe
Jan 18 17:09:32.484	cinch-brew-order-com...	POST
Jan 18 16:43:16.257	cinch-brew	/coffee
Jan 18 16:43:15.959	cinch-brew-order-com...	LISTCoffe
Jan 18 16:43:15.922	cinch-brew-order-com...	POST
Jan 18 16:25:05.334	cinch-brew	/coffee/types
Jan 18 16:25:05.192	cinch-brew-order-com...	LISTCoffeeBeanTypes
Jan 18 16:25:05.134	cinch-brew-order-com...	POST
Jan 18 16:02:27.939	cinch-brew	/coffee/types
Jan 18 16:02:27.110	cinch-brew-order-com...	LISTCoffeeBeanTypes



*Here's a useful link

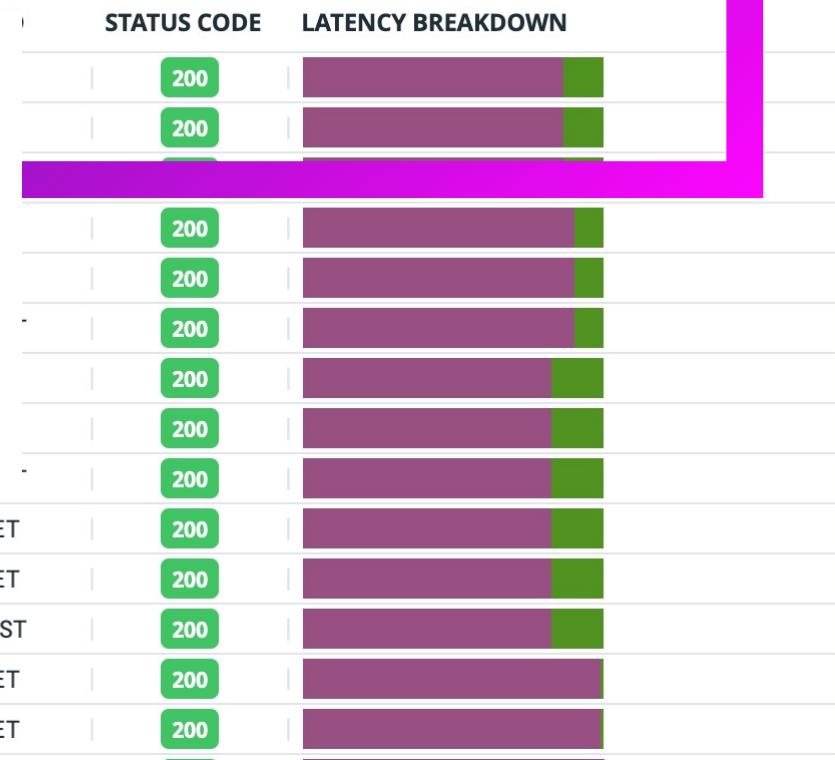
All our spans* too!

↓ DATE	SERVICE	RESOURCE	DURATION	METHOD	STATUS CODE	LATENCY BREAKDOWN
Jan 18 17:46:46.584	cinch-brew	/coffee/types	2.23 s	GET	200	
Jan 18 17:46:46.282	fx cinch-brew-order-com...	LISTCoffeeBeanTypes	300 ms	GET	200	
Jan 18 17:09:33.358	cinch-brew	/coffee/types	3.46 s	GET	200	
Jan 18 17:09:32.543	fx cinch-brew-order-com...	LISTCoffeeBeanTypes	339 ms	GET	200	
Jan 18 17:09:32.484	🌐 cinch-brew-order-com...	POST	139 ms	POST	200	
Jan 18 16:43:16.257	cinch-brew	/coffee/types	1.96 s	GET	200	
Jan 18 16:43:15.959	fx cinch-brew-order-com...	LISTCoffeeBeanTypes	340 ms	GET	200	
Jan 18 16:43:15.922	🌐 cinch-brew-order-com...	POST	141 ms	POST	200	
Jan 18 16:25:05.334	cinch-brew	/coffee/types	1.96 s	GET	200	
Jan 18 16:25:05.192	fx cinch-brew-order-com...	LISTCoffeeBeanTypes	339 ms	GET	200	
Jan 18 16:25:05.134	🌐 cinch-brew-order-com...	POST	159 ms	POST	200	
Jan 18 16:02:27.939	cinch-brew	/coffee/types	3.11 s	GET	200	
Jan 18 16:02:27.110	fx cinch-brew-order-com...	LISTCoffeeBeanTypes	37.7 ms	GET	200	

*Seriously, next slide can wait.

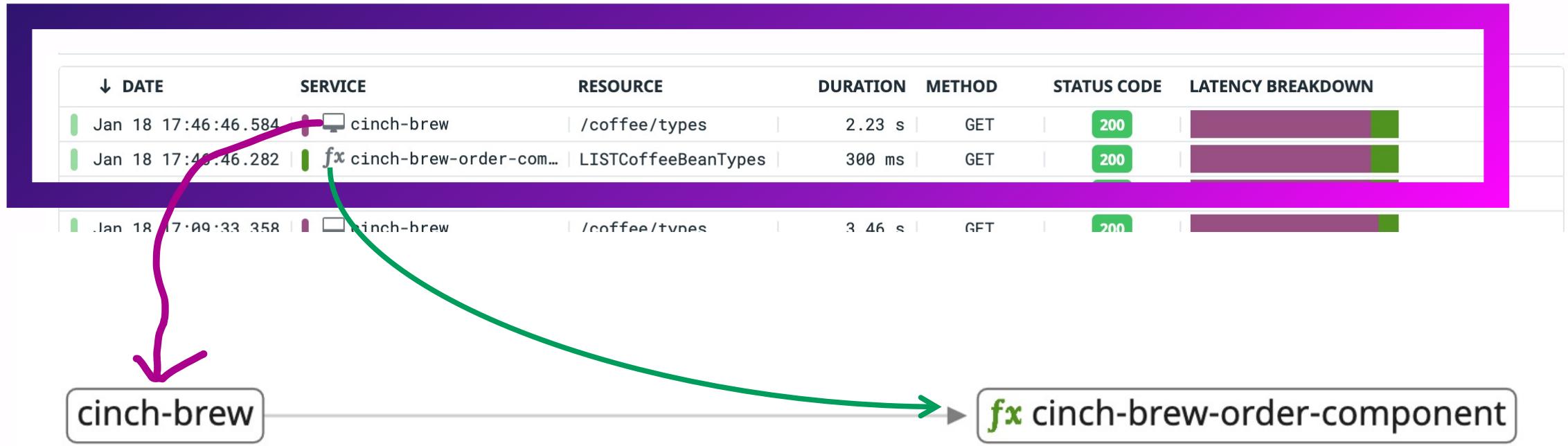
All our spans* too!

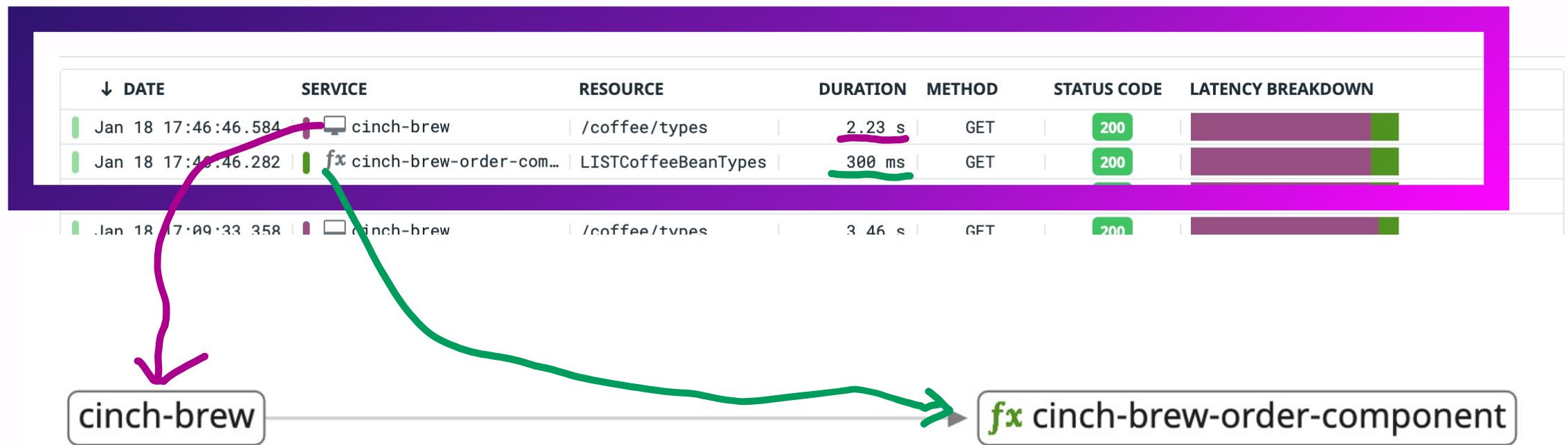
↓ DATE	SERVICE	RESOU
Jan 18 17:46:46.584	(cinch-brew)	/coffee
Jan 18 17:46:46.282	(fx cinch-brew-order-com...)	LISTC
Jan 18 17:09:33.358	(cinch-brew)	/coffee
Jan 18 17:09:32.543	(fx cinch-brew-order-com...)	LISTC
Jan 18 17:09:32.484	(@ cinch-brew-order-com...)	POST
Jan 18 16:43:16.257	(cinch-brew)	/coffee
Jan 18 16:43:15.959	(fx cinch-brew-order-com...)	LISTC
Jan 18 16:43:15.922	(@ cinch-brew-order-com...)	POST
Jan 18 16:25:05.334	(cinch-brew)	/coffee/types
Jan 18 16:25:05.192	(fx cinch-brew-order-com...)	LISTCoffeeBeanTypes
Jan 18 16:25:05.134	(@ cinch-brew-order-com...)	POST
Jan 18 16:02:27.939	(cinch-brew)	/coffee/types
Jan 18 16:02:27.110	(fx cinch-brew-order-com...)	LISTCoffeeBeanTypes



*<https://opentelemetry.lightstep.com/spans/>

Anyway, these are two spans





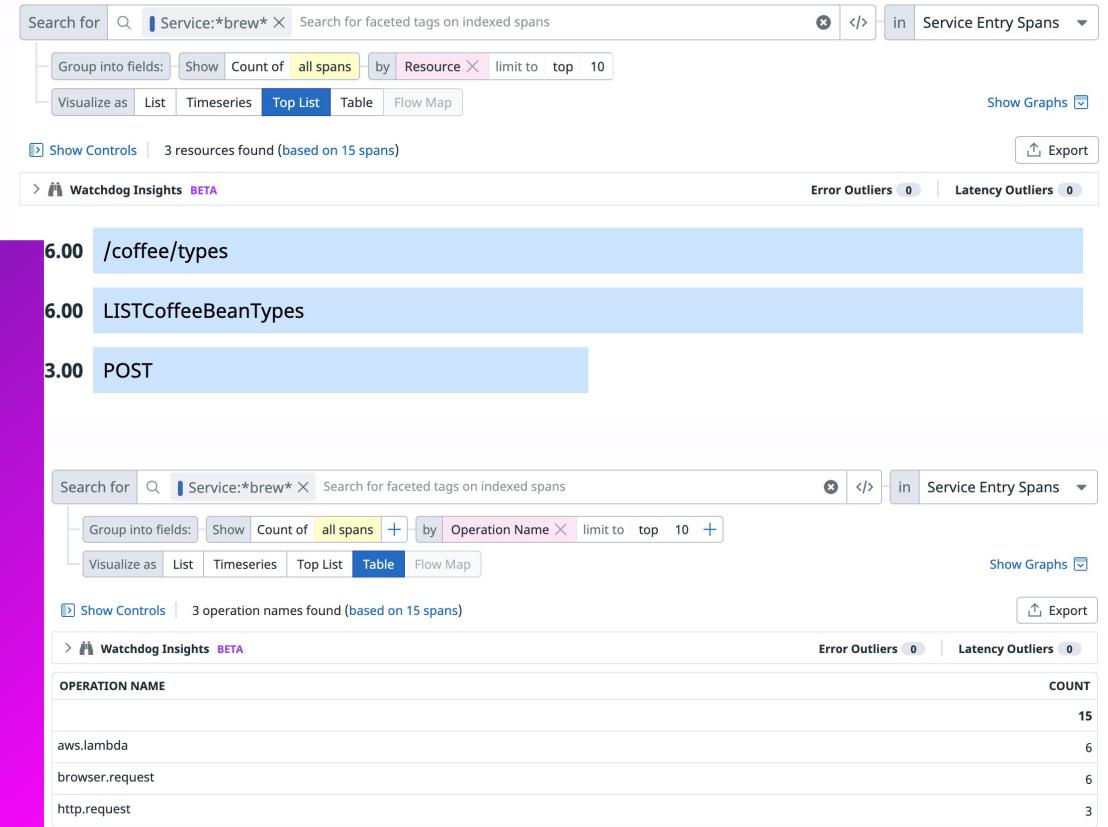
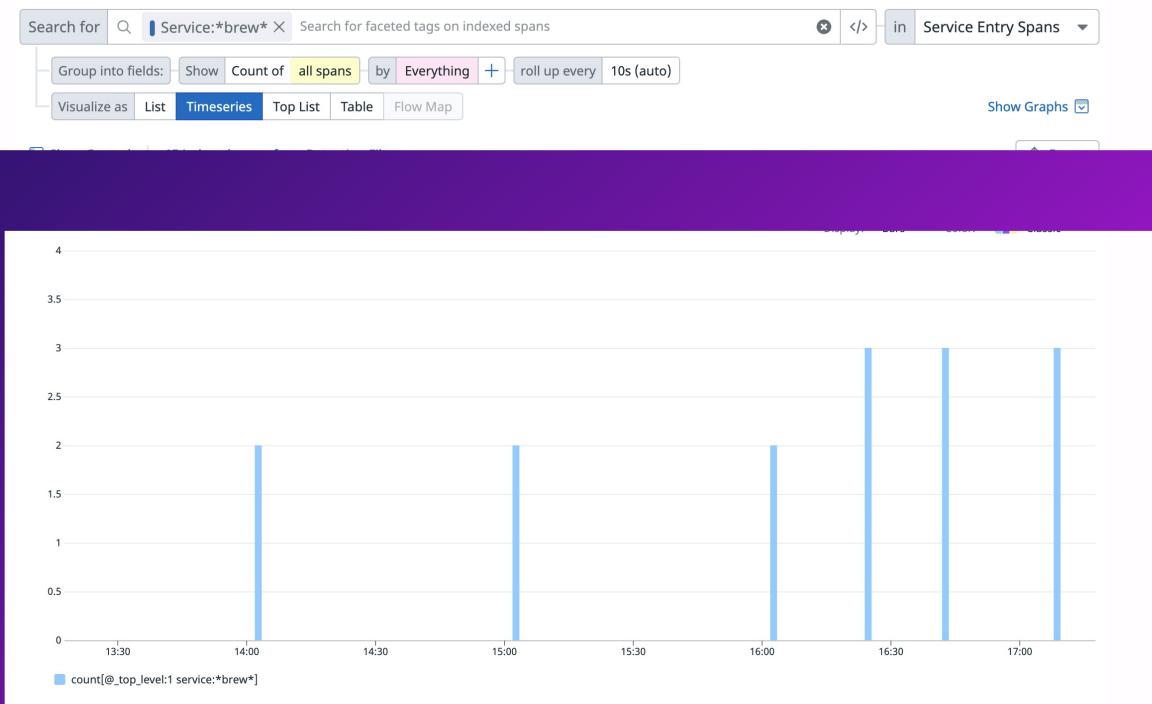
Frontend span

Backend span

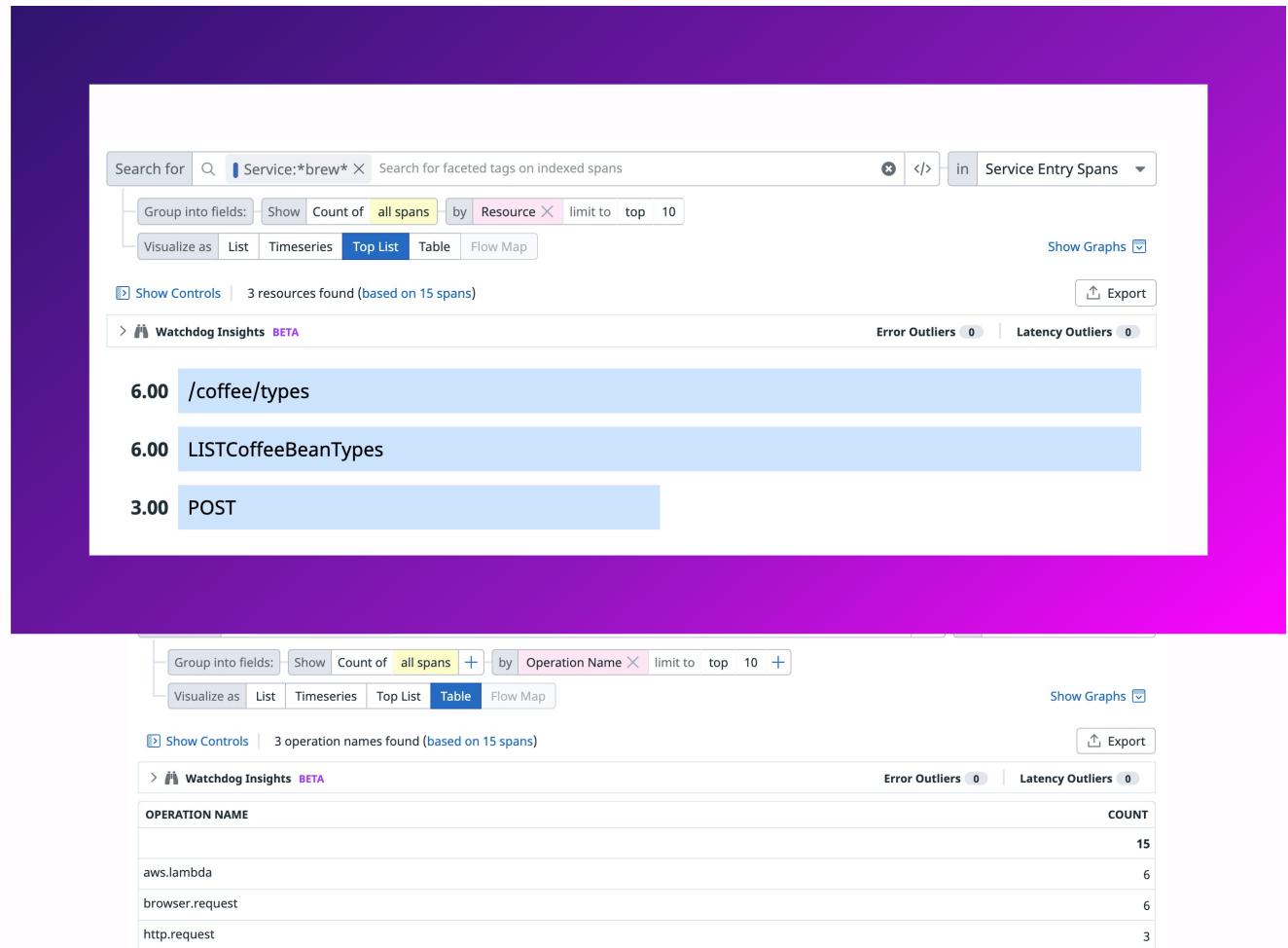
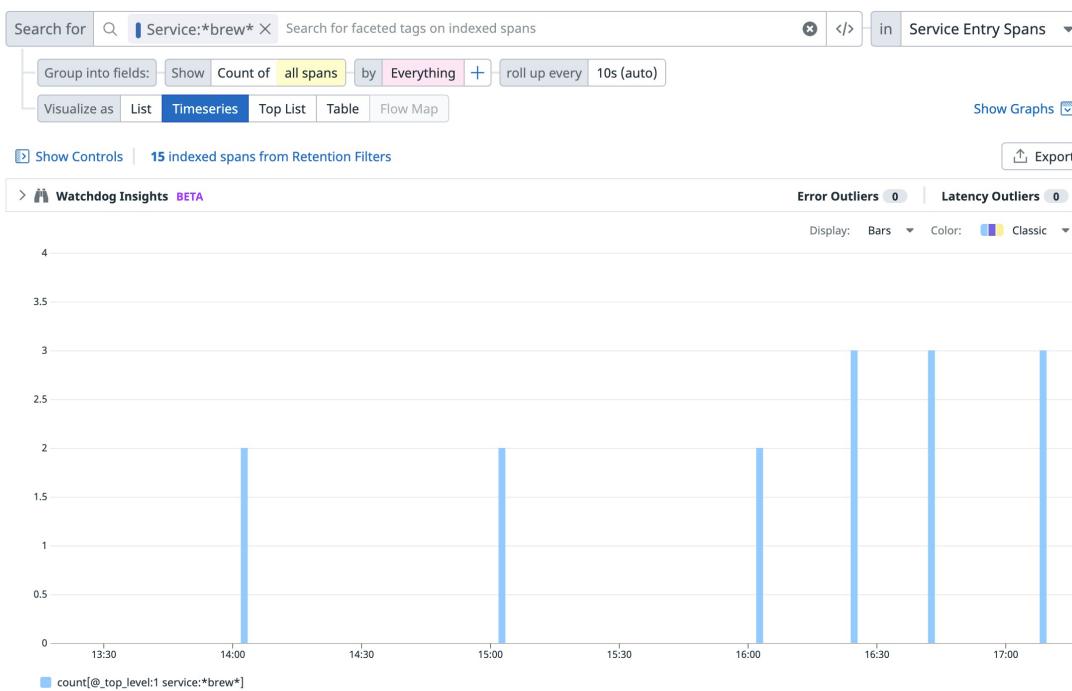
Flame graph: a better way of visualising spans



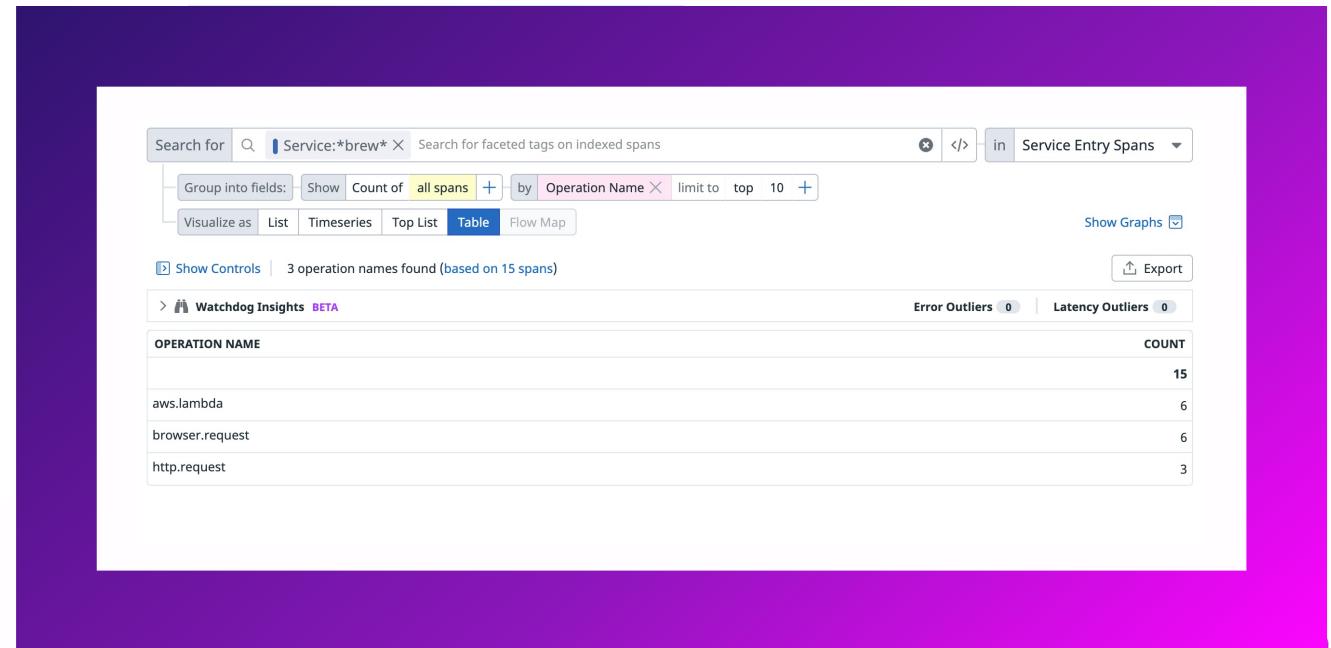
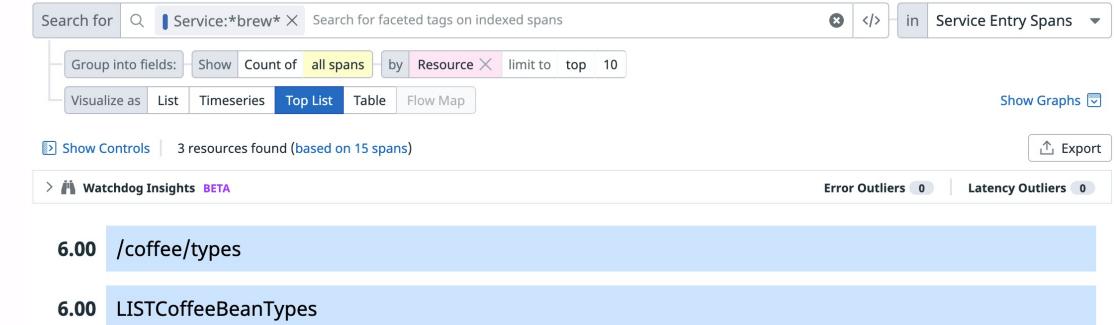
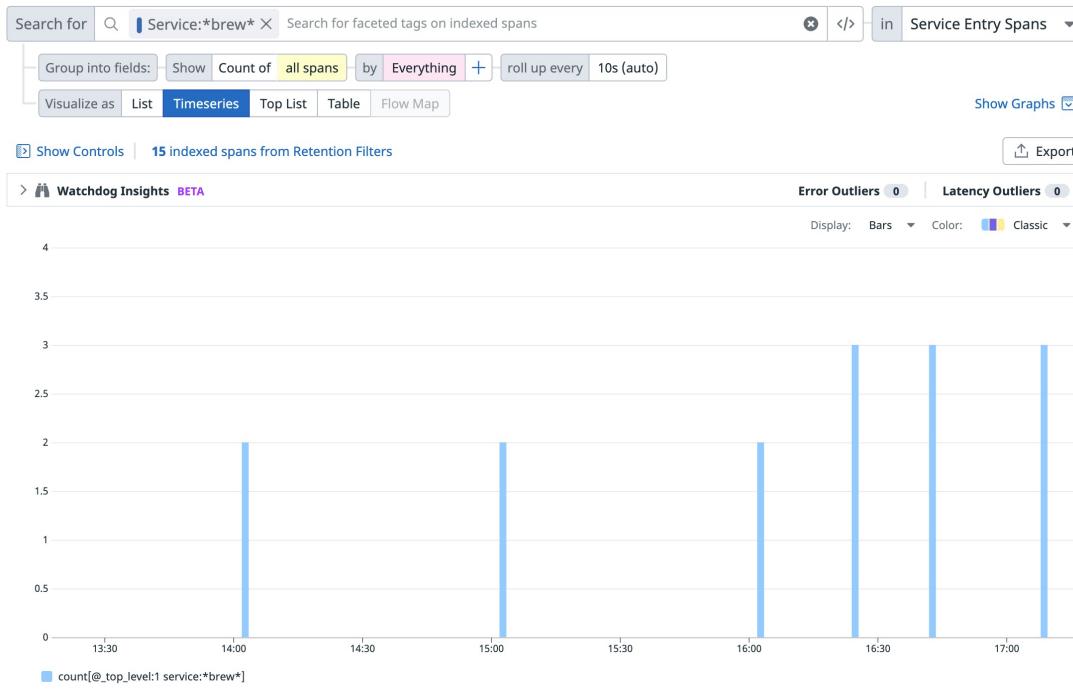
Spans as time series



Spans as top list



Even spans as tables



**Custom instrumentation makes
the difference.**

Custom instrumentation: set tags

```
4 import { setTag, sendMetric } from '../../../../../helpers/observability';
5 import { GetCommand } from '@aws-sdk/lib-dynamodb';
6 import { dynamodbDocClient } from '../../../../../helpers/aws-clients';
7
8 export const handler: Handler<APIGatewayProxyEventV2, APIGatewayProxyStructuredResultV2> = async () => {
9     setTag('response_hardcoded', false);
10
11    const params = {
12        TableName: 'cinch-brew-order-component-store',
13        Key: { id: 'TYPES', sk: 'COFFEE_BEANS' },
14    };
15    const coffeeBeanTypes = await dynamodbDocClient.send(new GetCommand(params)).then((res) => res.Item?.options || []);
16    sendMetric('available_coffee.Bean_types', coffeeBeanTypes.length, 'metricType:count');
17
18    if (coffeeBeanTypes.length > 0) {
19        return apiGatewayResult({ body: { types: coffeeBeanTypes } });
20    } else {
21        return apiGatewayResult({ status: 404 });
22    }
23};
```

Our custom tag on the span

Traces >  **cinch-brew** > `/coffee/types` > trace_id: 7965058752485519897

Jan 18 17:09:29.902 | 3.46 s | GET https://cinch-brew-api.dev.api.cinch.co.uk/coffee/types | 200 OK

 cinch-brew | Electron | 🐧 Linux | 🇺🇸 United States

Flame Graph | Span List (3) | Trace Map





⌚ 339 ms | >p99 🔍 (9.81% of total)

Tags	Metrics	Logs
env	dev	
function_arn	arn:aws:lambda:eu-west-1:563314216495:function:listcoffeebeantypes	
function_trigger {		
event_source	api-gateway	
event_source_arn	arn:aws:apigateway:eu-west-1::/restapis/blf0zukdpk/stages/prod	
}		
function_version	\$LATEST	
functionname	listcoffeebeantypes	
http {		
host	cinch-brew-api.dev.api.cinch.co.uk	
method	GET	
path_group	/	
status_code	200	
url	cinch-brew-api.dev.api.cinch.co.uk	
url_details {		
path	/coffee/types	
}		
}		
memorysize	256	
region	eu-west-1	
request_id	8f8f1342-81c0-42e3-ba4d-3d48c80a9b5f	
response_type	listcoffeebeantypes	
response_hardcoded	false	
runtime	nodejs14.x	
service	cinch-brew-order-component	
squad	enablement	
tribe	enablement	
version	1	

We use metrics heavily*

Traces

high context &
traceability

Metrics

high value signals
with limited context

Logs

high context with
free text included &
links to traces

RUM Events

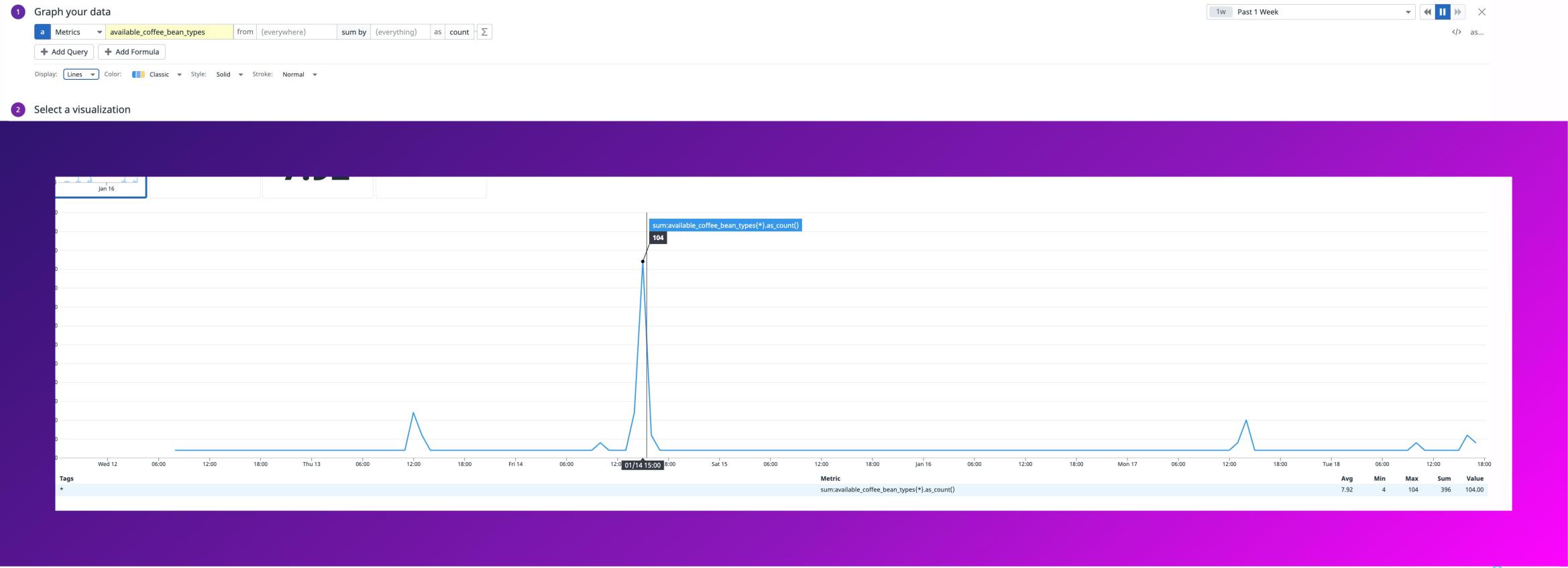
high context & links
to traces

* Although we do prefer traces and spans

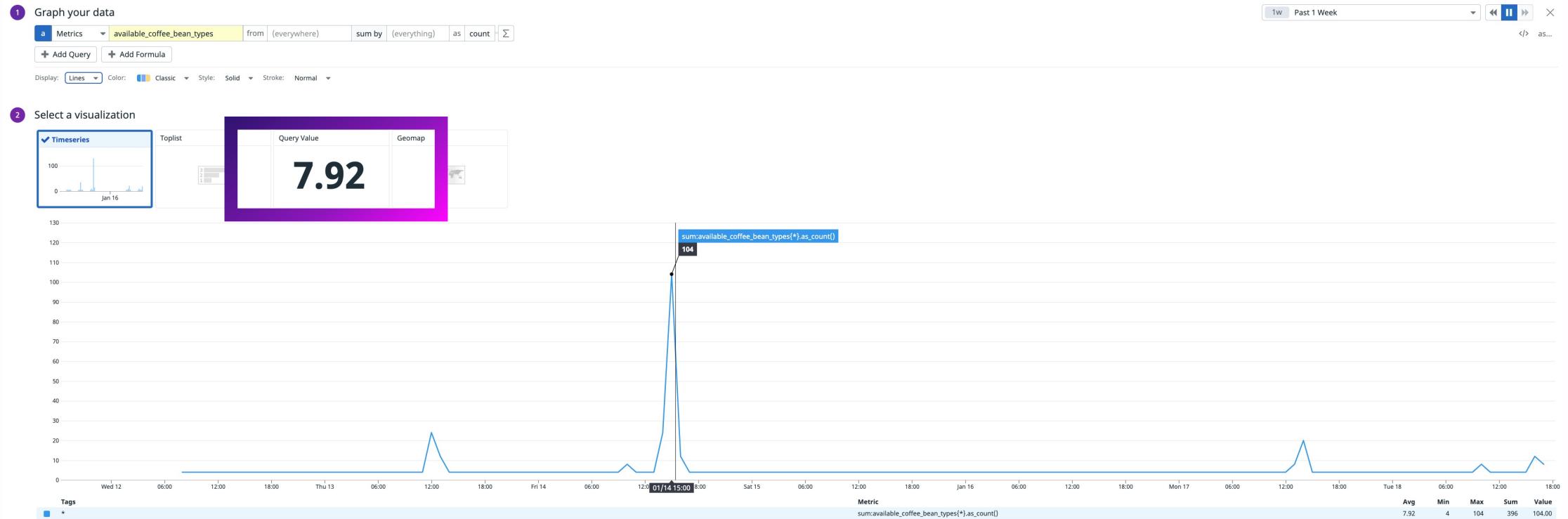
We send metrics directly from code

```
4 import { setTag, sendMetric } from '../../../../../helpers/observability';
5 import { GetCommand } from '@aws-sdk/lib-dynamodb';
6 import { dynamodbDocClient } from '../../../../../helpers/aws-clients';
7
8 export const handler: Handler<APIGatewayProxyEventV2, APIGatewayProxyStructuredResultV2> = async () => {
9     setTag('response_hardcoded', false);
10    const params = {
11        TableName: 'cinch-brew-order-component-store',
12        Key: { id: 'TYPES', sk: 'COFFEE_BEANS' },
13    };
14    const coffeeBeanTypes = await dynamodbDocClient.send(new GetCommand(params)).then((res) => res.Item?.options || []);
15
16    sendMetric('available_coffee_bean_types', coffeeBeanTypes.length, 'metricType:count');
17
18    if (coffeeBeanTypes.length > 0) {
19        return apiGatewayResult({ body: { types: coffeeBeanTypes } });
20    } else {
21        return apiGatewayResult({ status: 404 });
22    }
23};
```

We can then graph metric values over time



Or create a single number



RUM* events: the tracing of the frontend

Traces

high context &
traceability

Metrics

high value signals
with limited context

Logs

high context with
free text included &
links to traces

RUM Events

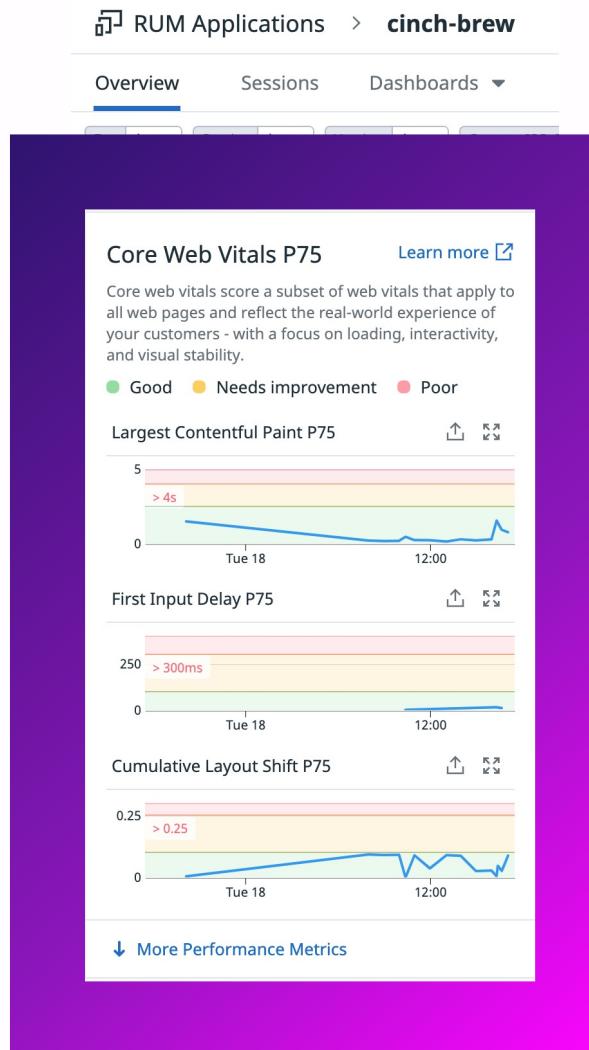
high context & links
to traces

* RUM stands for Real User Monitoring. I think it deserves to be abbreviated.

Auto-instrumentation

```
1 import { datadogRum } from "@datadog/browser-rum";
2
3 datadogRum.init({
4   applicationId: "████████████████████████████████",
5   clientToken: "████████████████████████████████",
6   sampleRate: 100,
7   service: "cinch-brew",
8   env: "dev",
9   version: process.env.NEXT_PUBLIC_BUILD_ID,
10  sampleRate: 100,
11  replaySampleRate: 100,
12  trackInteractions: true,
13  defaultPrivacyLevel: "mask-user-input",
14  allowedTracingOrigins: [
15    "https://cinch-brew████████████████████████████████.cinch.co.uk",
16    "/https:\//.*\.\cinch\.co\.uk/",
17  ],
18 });
19
20 datadogRum.startSessionReplayRecording();
```

Out of the box analytics



Out of the box performance (and many more)

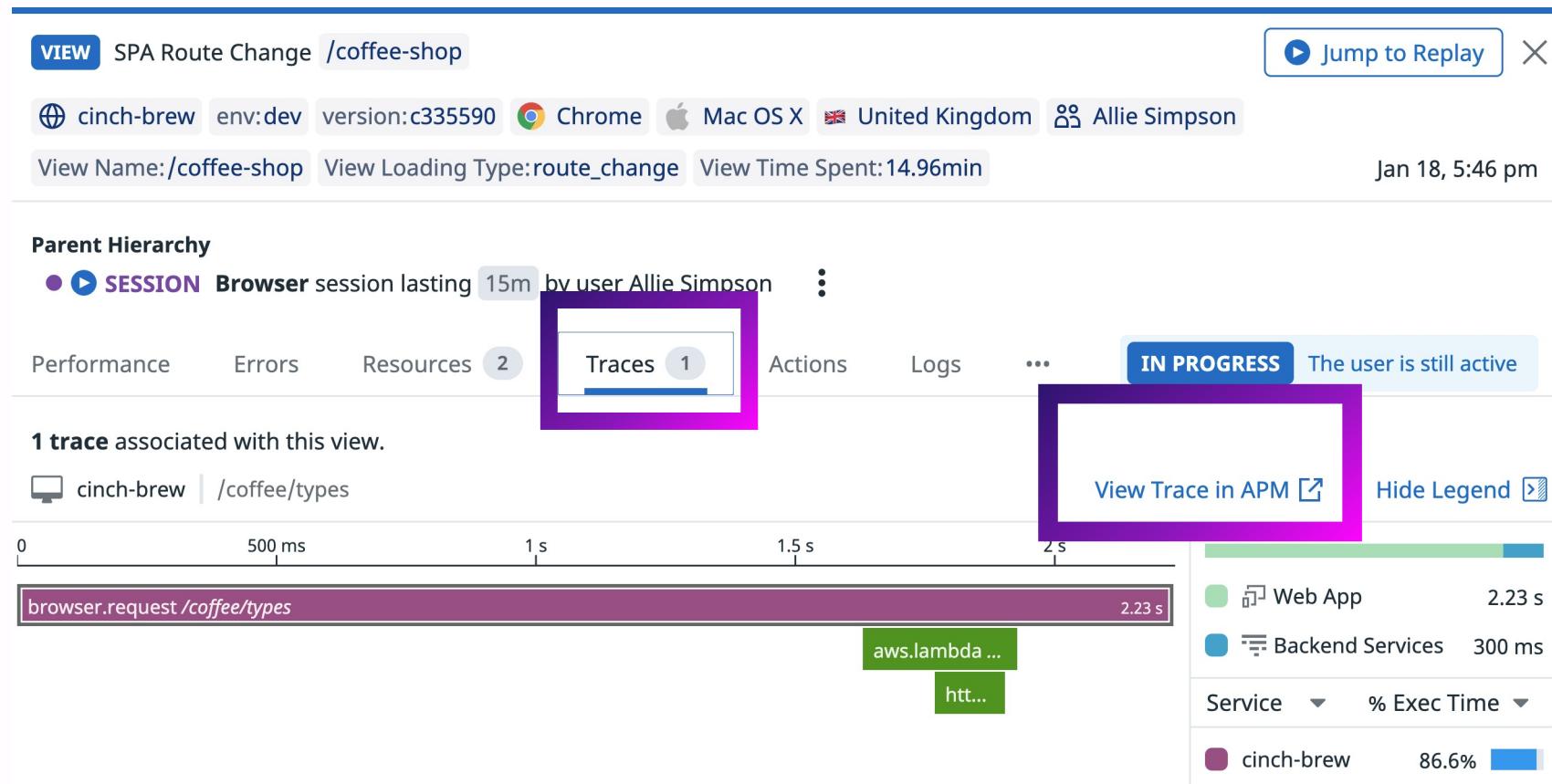
▼ Most viewed pages



Performance overview of most popular pages

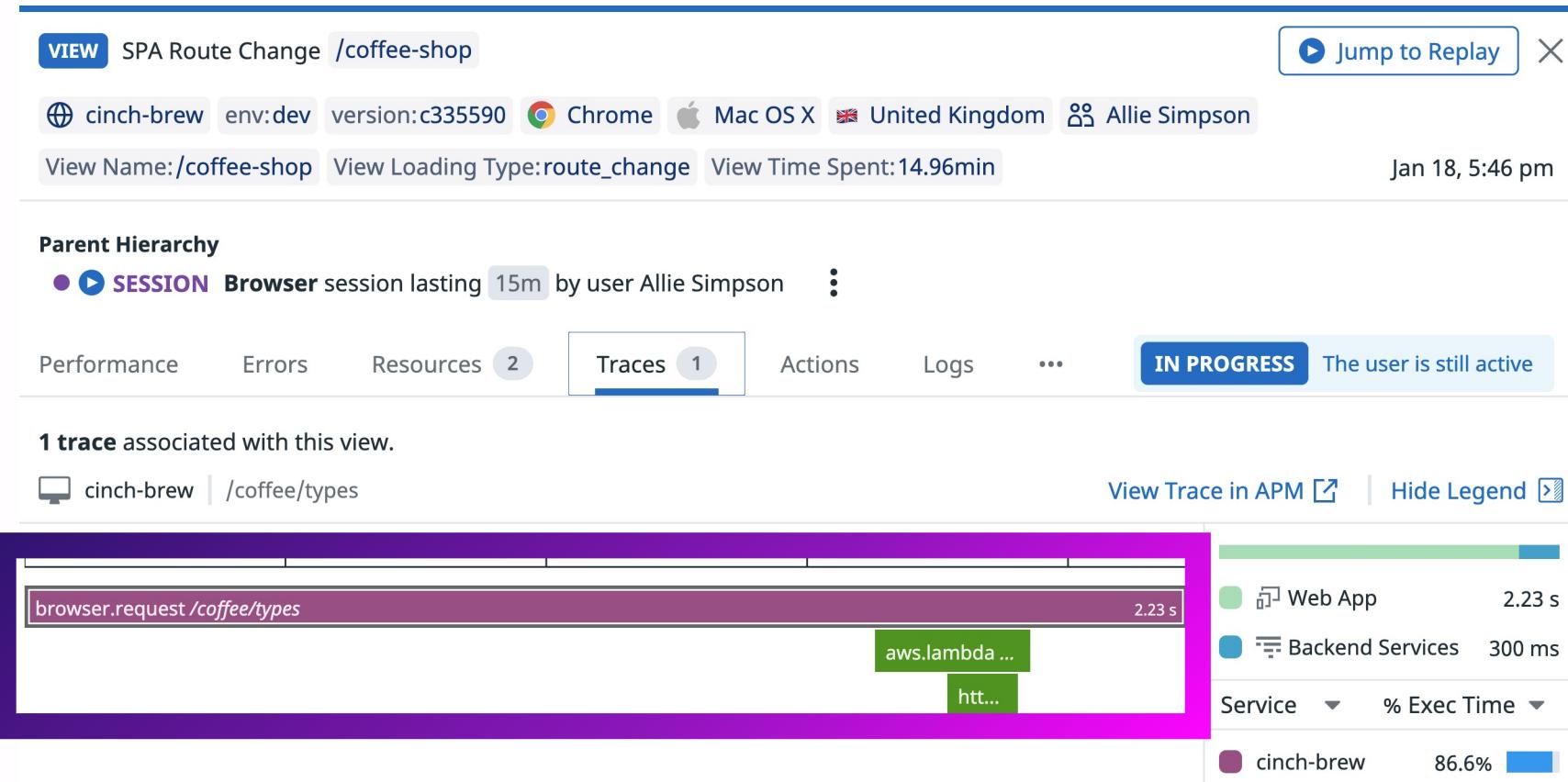
VIEW NAME	↓ COUNT	PC75:LOADING TIME	PC75:LARGEST CONTENTFUL PAINT	PC75:FIRST INPUT DELAY	PC75:CUMULATIVE LAYOUT SHIFT
/	16	1.55s	635.85ms	13.33ms	0.024
/login	14	1.26s	N/A	N/A	0.036

Tracing frontend to backend*

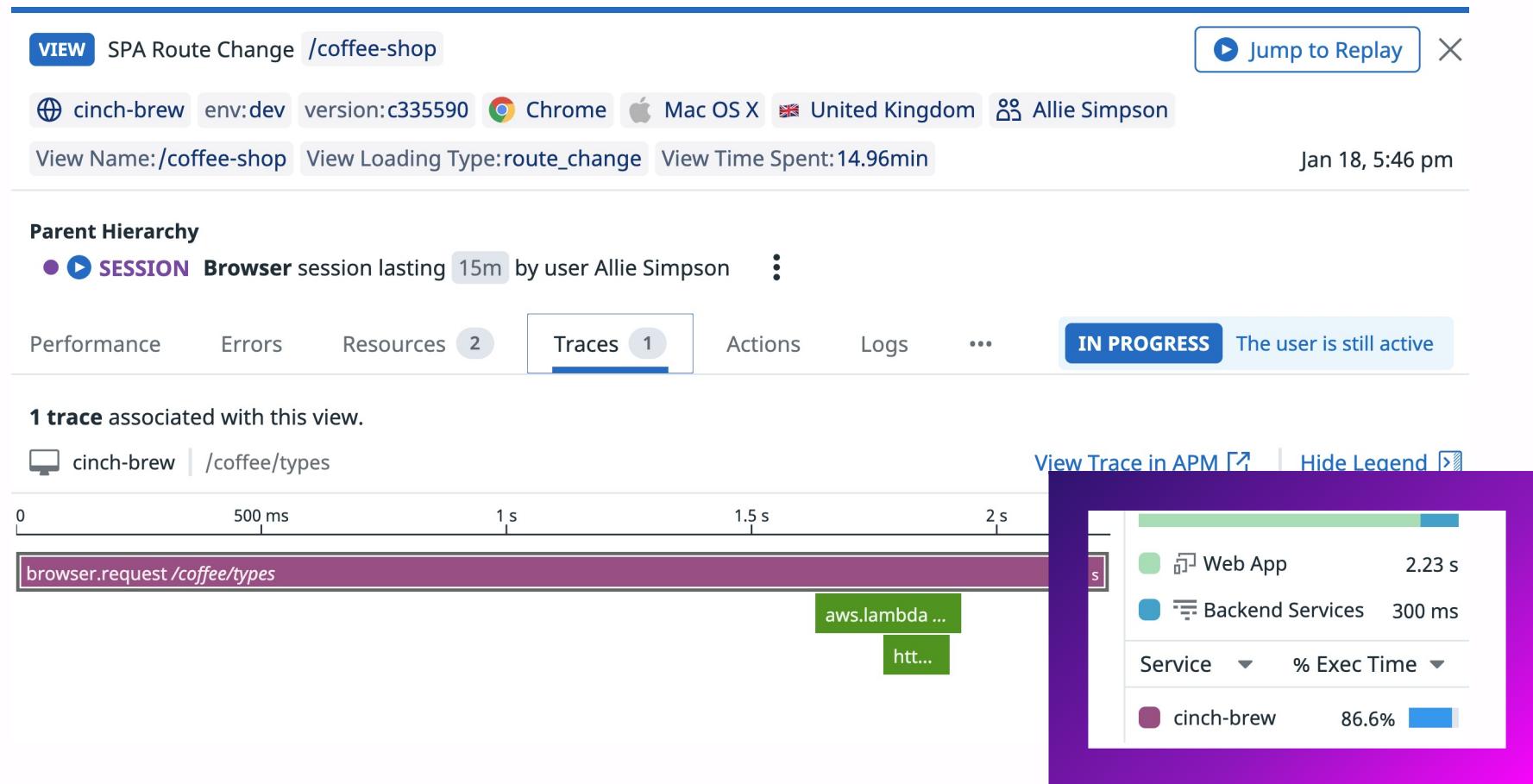


* I have hinted it before. I'm making it explicit here

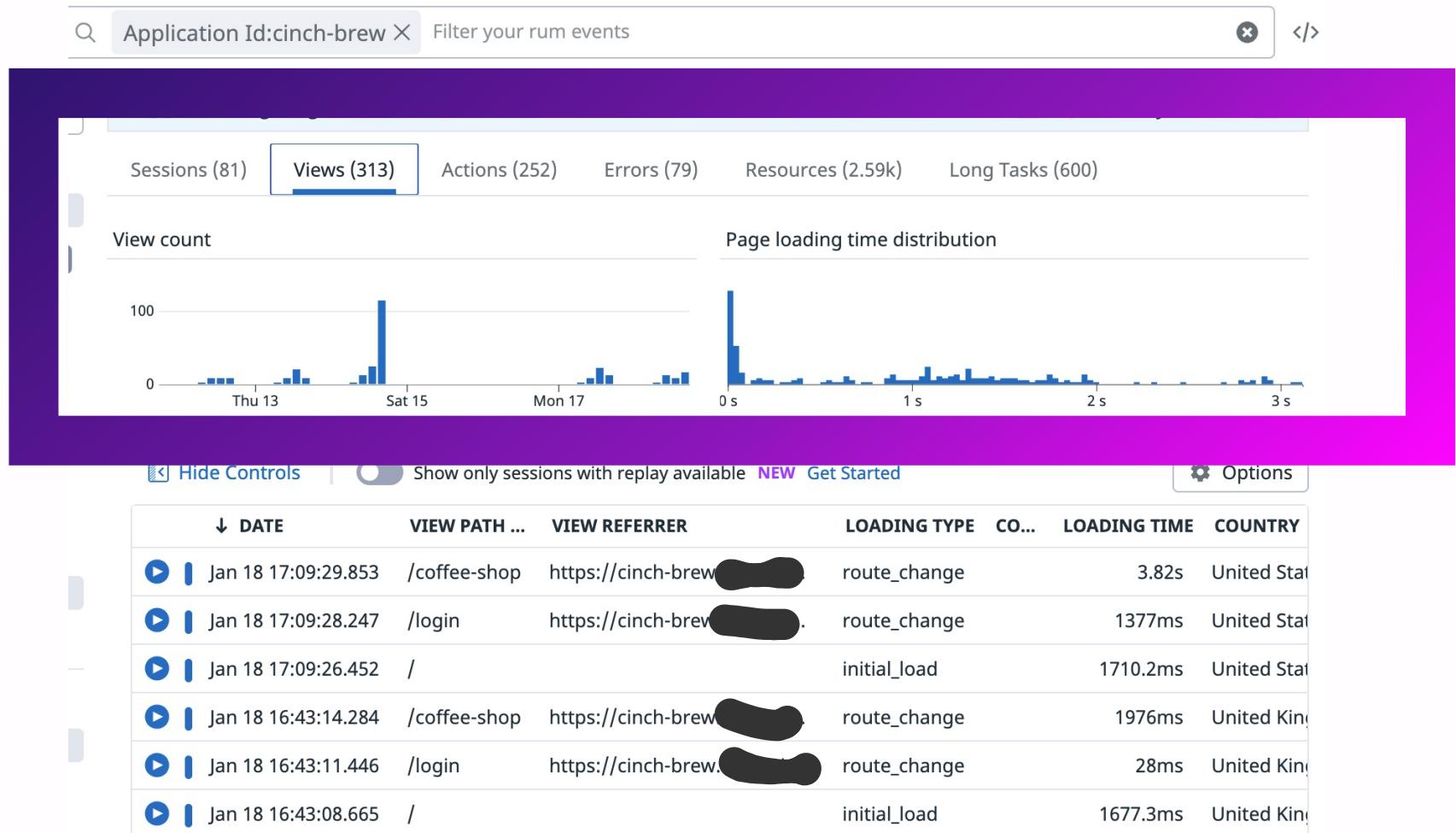
Spans with both frontend and backend



86% taken on the frontend



Like spans, you can list views



Each row is an individual view

Application Id:cinch-brew X Filter your rum events x </>

> **Watchdog Insights BETA** Error Outliers 0 | Latency Outliers 0

Sessions (81) Views (313) Actions (252) Errors (79) Resources (2.59k) Long Tasks (600)

View count Page loading time distribution

A screenshot of the Watchdog Insights interface for the application 'cinch-brew'. The top navigation bar includes a search field, a filter for rum events, and session statistics. Below this is a dashboard with two main charts: 'View count' (a histogram showing spikes in views on Saturday, January 15th) and 'Page loading time distribution' (a histogram showing most pages load within 1 second). The bottom section displays a table of individual view events, each represented by a purple box. The table columns include DATE, VIEW PATH ..., VIEW REFERRER, LOADING TYPE, CO..., LOADING TIME, and COUNTRY. The first few rows show events like a route_change from a referral to '/coffee-shop' at 17:09:29.853 and another at 17:09:28.247, both taking approximately 3.8 seconds.

DATE	VIEW PATH ...	VIEW REFERRER	LOADING TYPE	CO...	LOADING TIME	COUNTRY
Jan 18 17:09:29.853	/coffee-shop	https://cinch-brew...	route_change		3.82s	United States
Jan 18 17:09:28.247	/login	https://cinch-brew...	route_change		1377ms	United States
Jan 18 17:09:26.452	/		initial_load		1710.2ms	United States
Jan 18 16:43:14.284	/coffee-shop	https://cinch-brew.dev.api....	route_change		1976ms	United Kingdom
Jan 18 16:43:11.446	/login	https://cinch-brew.dev.api....	route_change		28ms	United Kingdom
Jan 18 16:43:08.665	/		initial_load		1677.3ms	United Kingdom

**Again, custom instrumentation
makes the difference.**

Add a custom action*

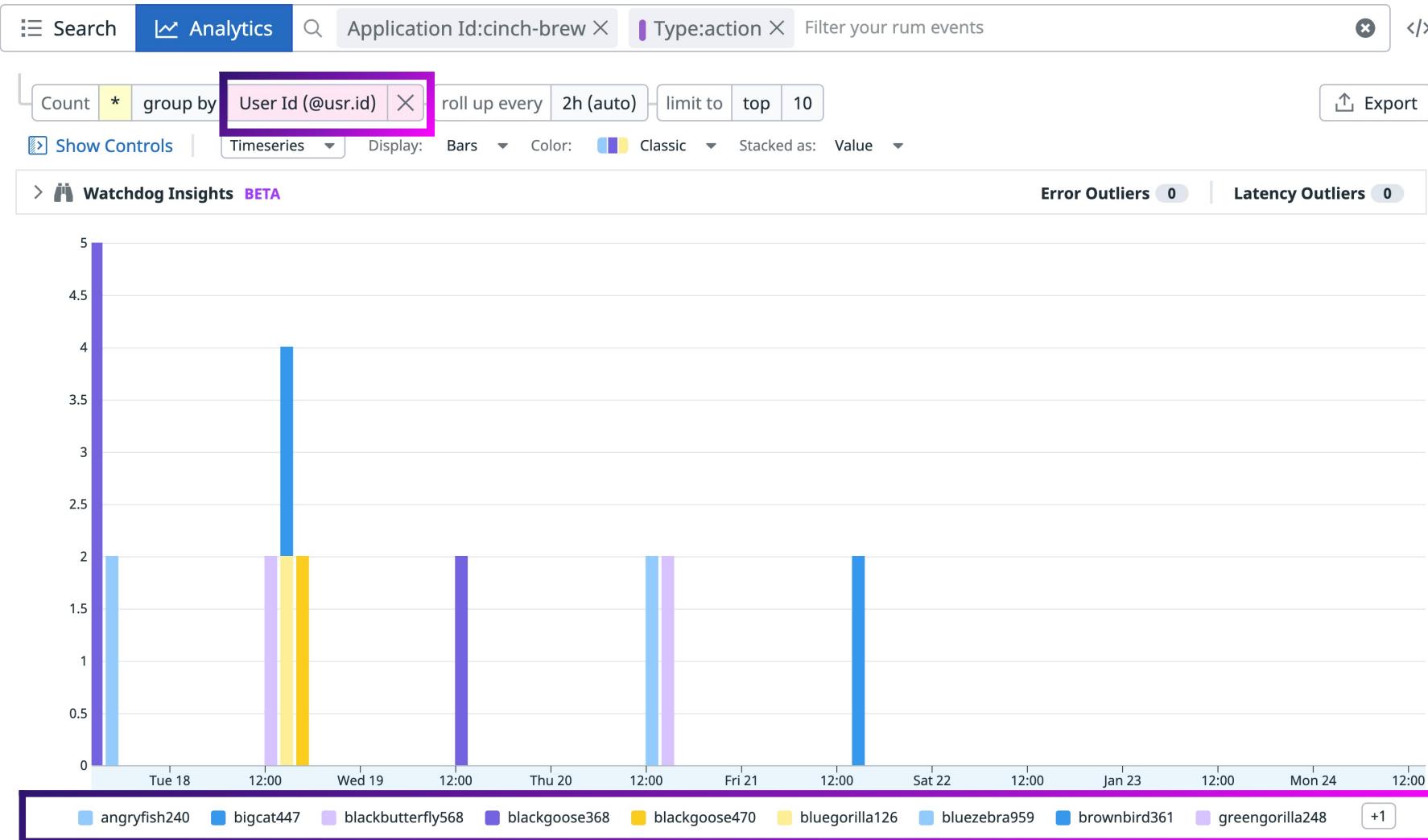
```
7     userTimeout(() -> {
8         // create a custom action to datadog rum to notify that the user has not done much
9         datadogRum.addAction("userSleeping", { inactivitySeconds: 5 });
10        , 5_000);
```

* I don't have an example of this in action right now.

Or set a user

```
26      if (user) {  
27          localStorage.setItem("user", JSON.stringify(user));  
28          datadogRum.setUser({  
29              id: user.username,
```

Graph all views by user*



* We use ids by default. We mask sensitive data by default. We don't set PII.

Instrumentation / tips



Use traces & RUM events as a rich dataset of events



Adopt custom instrumentation practices



Get your service boundaries right

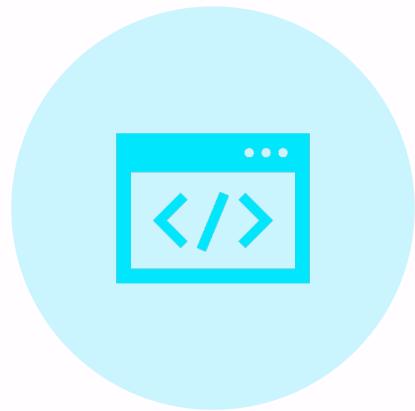


Follow the ollie tool's principles

Instrumentation / challenges



TOO MANY DATA TYPES ON THE BACKEND IS CONFUSING



ENGINEERS ARE VERY USED TO LOGS



ENGINEERS NEED TO 'SEE' TRACES & RUM EVENTS WORKING TO BE CONVINCED

Instrumentation / challenges

01

Native SDKs
very divergent
from olly tool
SDKs

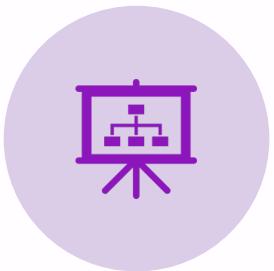
02

Overuse of
metrics loses
context

03

Getting
everything right
at once is hard

Milestone #2: Dashboards



**SQUADS OWN & EVOLVE
THEIR DASHBOARDS**



A VEHICLE TO GET
SIGNALS FROM
PRODUCTION



FOCAL POINTS AT STAND
UPS WITHIN A SQUAD



INFORMATION
RADIATORS FOR
EXTERNAL TO THE
SQUAD AUDIENCE

Inclusive

The screenshot shows a dashboard titled "Orders Squad - Motherboard". At the top, there are search and filter fields for "Search...", "Saved Views", "\$aws_account" (set to [REDACTED]), "\$env" (set to "prod"), "\$aws_account_id" (set to [REDACTED]), and "\$environment" (set to "production"). There are also time controls for "1h" and "Past 1 Hour", and navigation icons for back, forward, and search.

The dashboard displays six cards, each with a question and the number of widgets:

- Are orders looking healthy?** 23 widgets
- Is our part exchange service healthy?** 12 widgets
- What changes have been released?** 6 widgets
- Do we have any incidents?** 1 widget
- Are we seeing errors?** 5 widgets
- Are my APIs healthy?** 12 widgets

Comprehensive

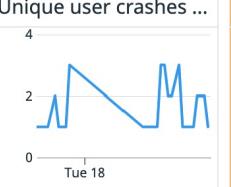
★ 🔎 Search Squad Standup Board ▾ [+ Add Widgets Cmd + E](#) 1d Past 1 Day ⏪ ⏴ ⏵ ⏵

🔍 Search... | Saved Views | \$env prod | \$no-version-conflict | version_conflict_engine_exception | ⚙️

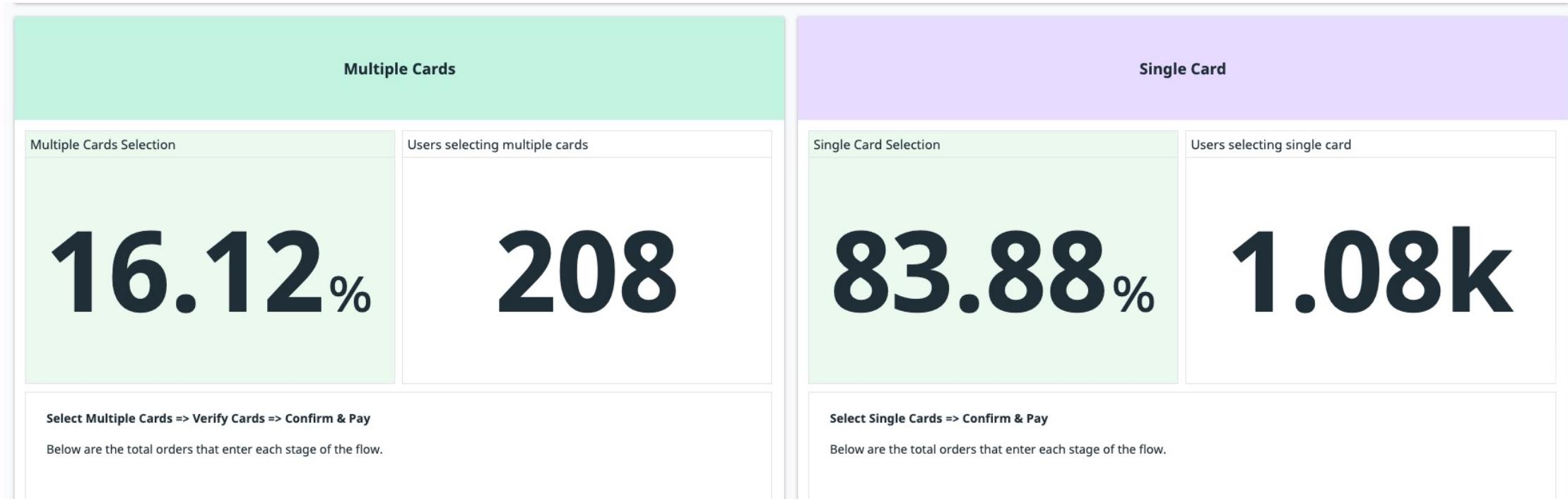
▼ 🔎 Search squads Links

Jira 📈	Repos 🛡️	Miro 🎨	Dashboards	Docs 📄	AWS 🤖
APM - Search 🚶	APM - Rec 🚶	CloudTrail	RUM 🍻	Sonar 🐛	Logs 🌴
Event	More ▾				

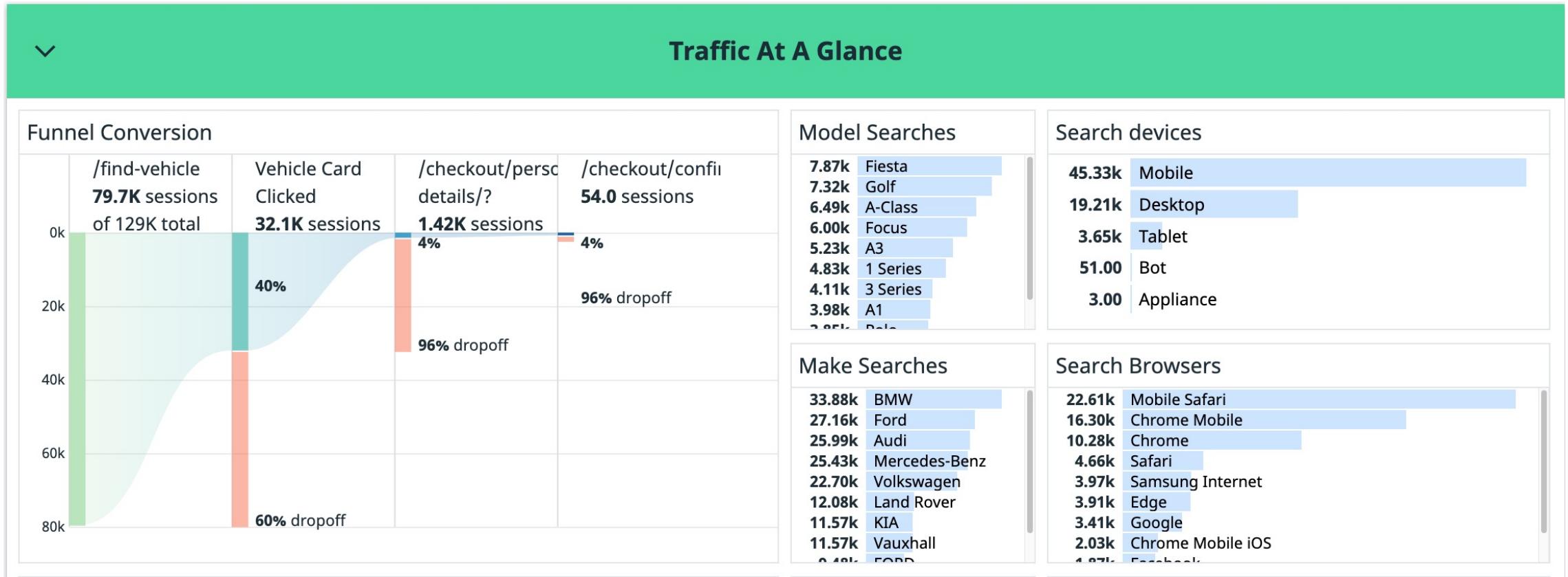
▼ Health at a Glance

Monitor statuses			Alerted monitors	LCP average	Unique user crashes
STATUS	MONITOR NAME	TRIGGERED			
OK	35		No matching entries found		
OK	Prod - Someone has messed with s...	4h	Learn more about events query syntax		32
OK	Prod - Vehicles get lambda duratio...	1w			
OK	Prod - API Gateway Logs are no lon...	1w			
OK	[Synthetics] Search API - Ford Fiesta	1w			

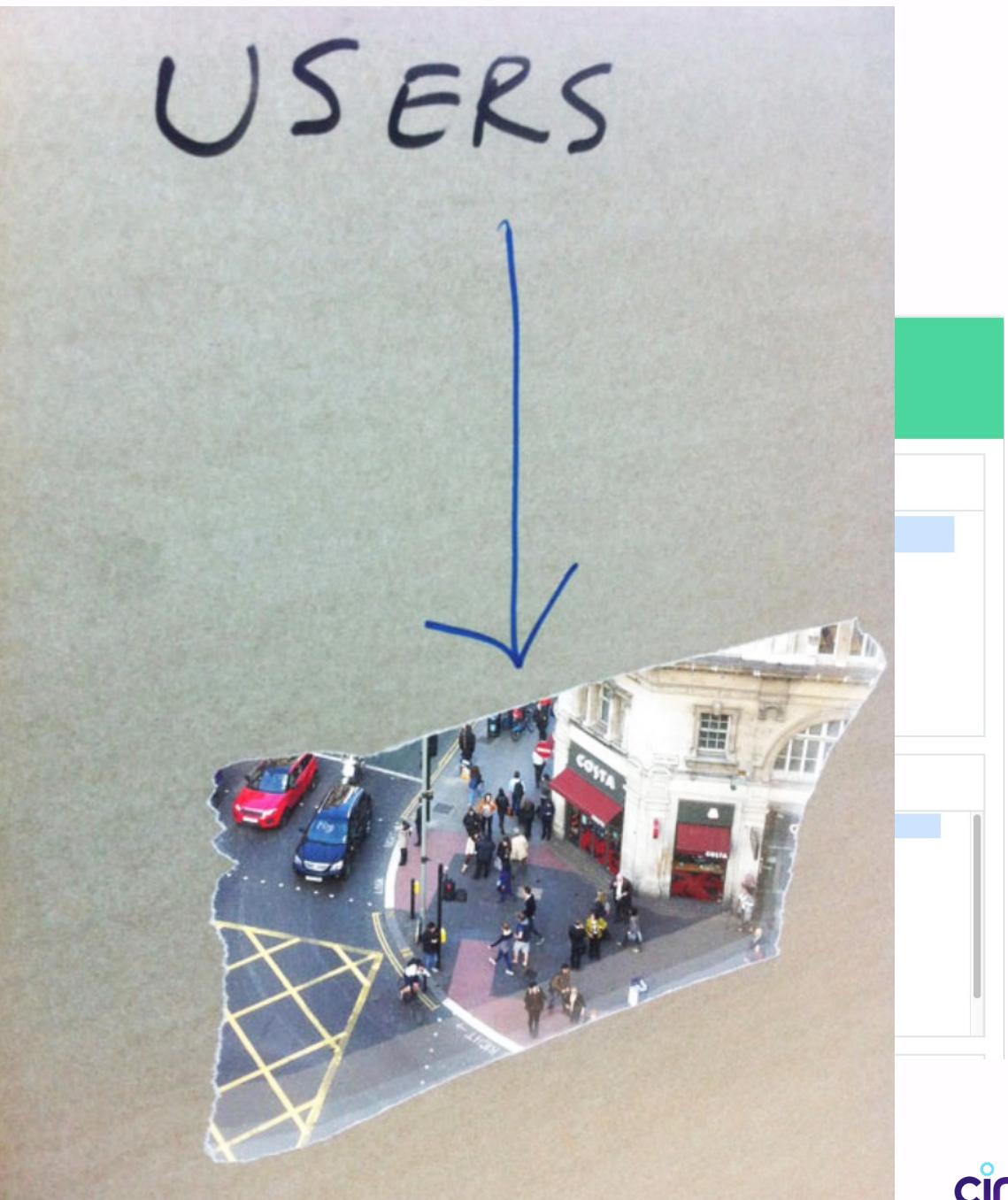
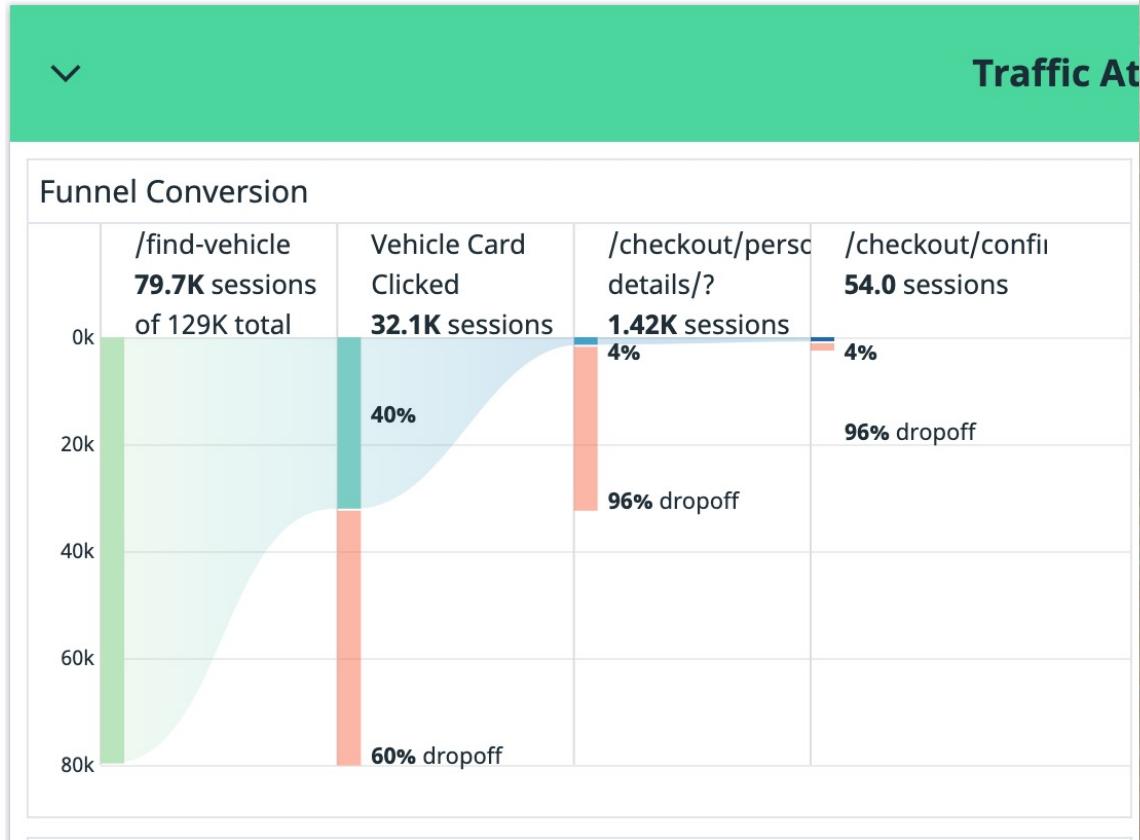
Customer centric



Teams care about funnels

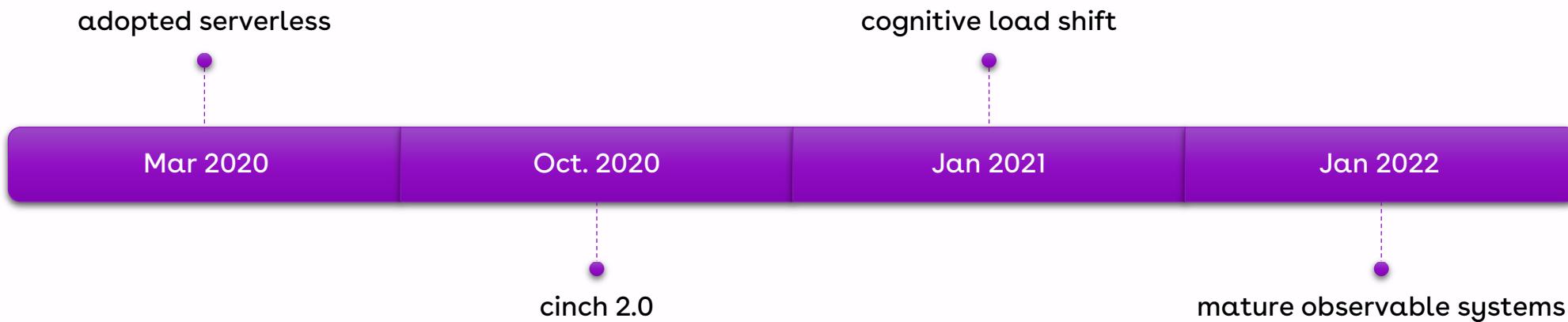


USERS



A collective cognitive load shift.

Mature Observability



Serverless...



Reduces

Caring about target infrastructure

Cognitive load



Requires

New engineering practices

New tooling

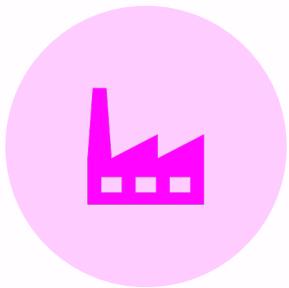


Upgrades

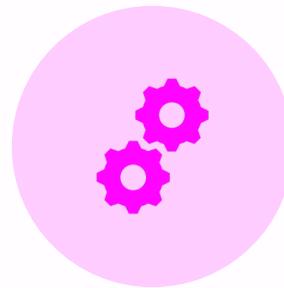
DevOps patterns and practices

Observability of systems

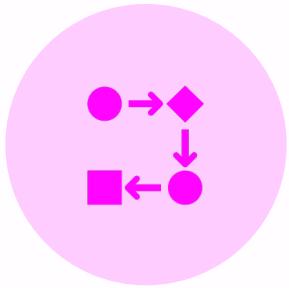
Organizational wins



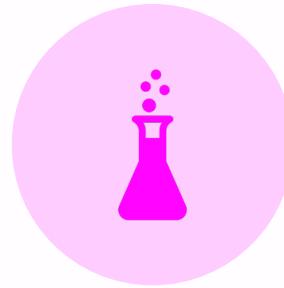
Focused on
production, rather
than platform



Squads own the
reliability of their
production systems



Fast flow, instant,
short feedback loops



Experimentation led

Emergent practices



Review dashboards after/during stand ups & stand downs



Review monitoring and observability weekly



In-hours, on-call rotas

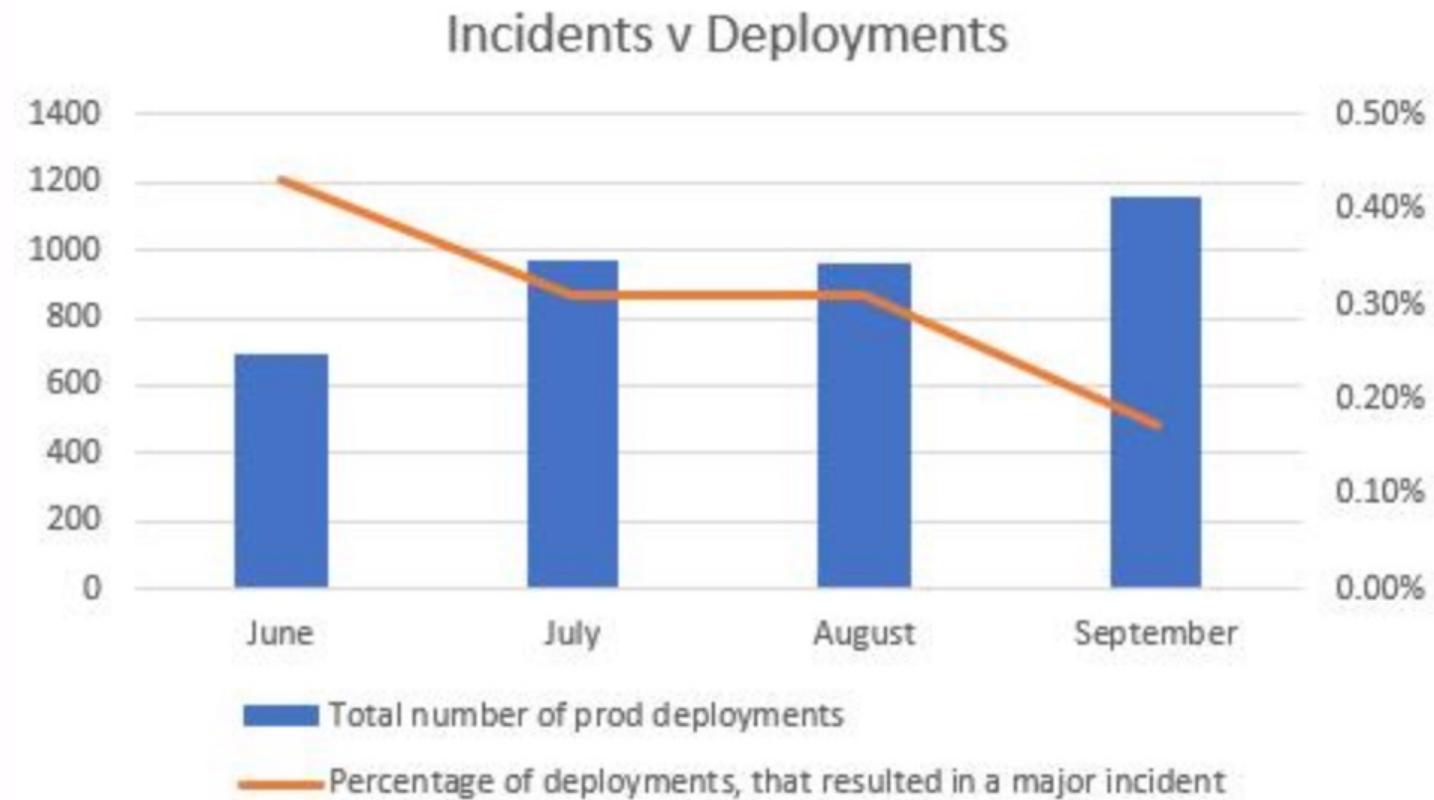


Observability workshops



Surface various CI/CD quality metrics

Incidents are very rare



What next?

What next?



DevOps Community of Practice



Principal Engineers to support squads

What next?



Rely more on SLOs



Establish
observability
blueprints (cinch brew)



Optimize incident
management



Deployment & Test
observability



Security cognitive
load shift

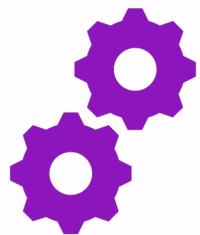


Public dashboards

My final 2 cents.



Choose tech stack that allows
for a cognitive load shift
towards observability



**Adopt engineering practices &
patterns** that enable teams to
truly have a DevOps mindset



Trust your teams, be patient

**Thank
you!**

Get in touch



dev@toli.io (personal)

toli@cinch.co.uk (work)

@apostolis09 (twitter)