

# Introduction to Observability

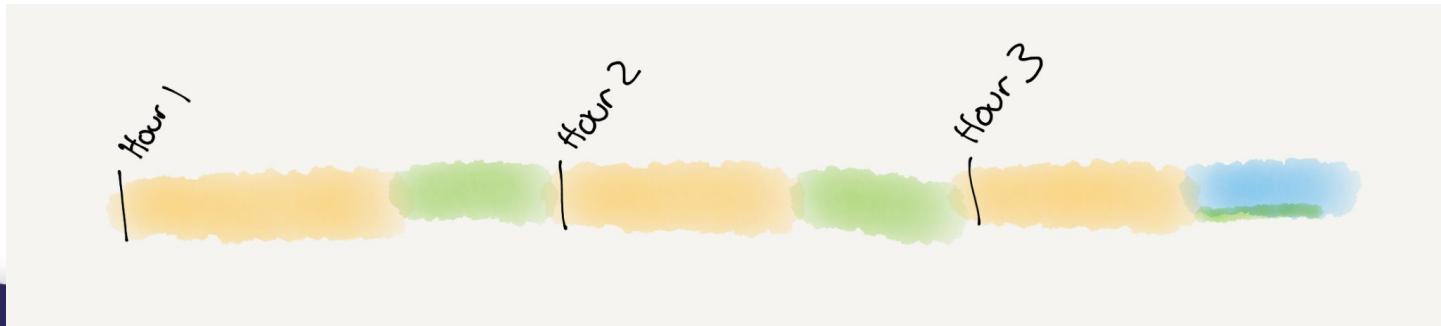
---

15 June 2022

# Today's agenda

---

- Housekeeping & Intros
- What is Observability?
- Get some data in!
- Answer questions with the data!
- What else might you do?
- Q&A



# Some housekeeping items

---

- We are not recording this.
- Captions are available.
- In the main room: raise hands using reactji, or ask questions in zoom chat
- In breakout rooms, you are welcome to unmute video/audio.
- Also ask questions in Slack! They persist





Please join us in Pollinators Slack  
at **#intro-to-olly-workshop**

---



## Jessica Kerr

Principal Developer Advocate

## David Marchante

Implementation Engineer





**Martin Thwaites**  
Developer Advocate



**Phillip Carter**  
Senior PM, Telemetry

**Mike Terhar**  
Customer Architect



# **Why does Observability matter?**

---

and how can it make your job easier?

# Developers have a lot on our plates.

---

We want devs to:

- Ship more often
- Produce better code
  - Spend less time debugging and more time creating value
- Decrease downtime



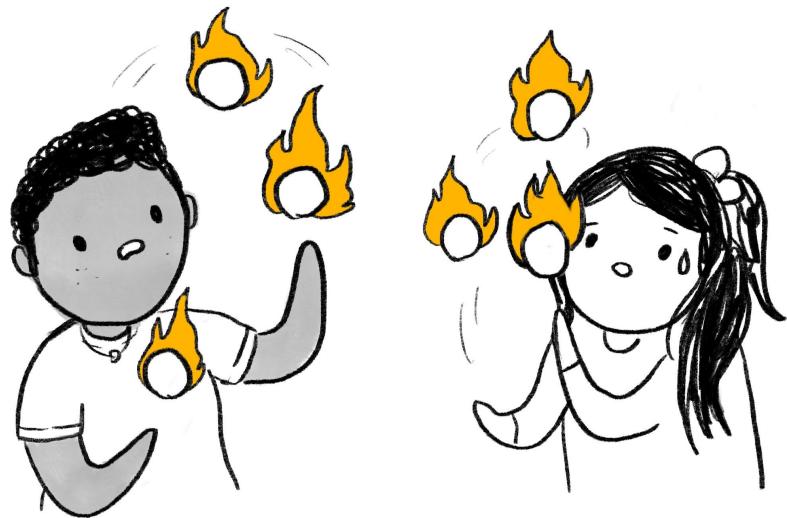
# Working in production is challenging.

---

We're afraid we'll break things.

We've forgotten what we've built by the time it ships.

We can't figure out what's wrong and fix it.



# But some organizations have solved it!

---

DevOps & Progressive Delivery culture have the answers to some of these problems.

And a growing fraction of our industry is moving faster *and* safer!

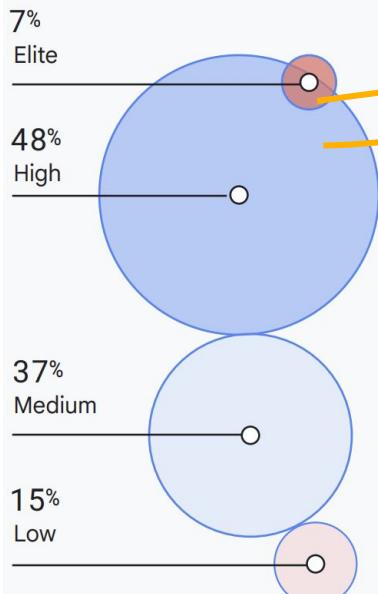


Software delivery performance metric	Elite	High	Medium	Low
<b>⌚ Deployment frequency</b> <p>For the primary application or service you work on, how often does your organization deploy code to production or release it to end users?</p>	On-demand (multiple deploys per day)	Between once per week and once per month	Between once per month and once every 6 months	Fewer than once per six months
<b>⌛ Lead time for changes</b> <p>For the primary application or service you work on, what is your lead time for changes (i.e., how long does it take to go from code committed to code successfully running in production)?</p>	Less than one hour	Between one day and one week	Between one month and six months	More than six months
<b>⌚ Time to restore service</b> <p>For the primary application or service you work on, how long does it generally take to restore service when a service incident or a defect that impacts users occurs (e.g., unplanned outage or service impairment)?</p>	Less than one hour	Less than one day	Between one day and one week	More than six months
<b>⚠ Change failure rate</b> <p>For the primary application or service you work on, what percentage of changes to production or released to users result in degraded service (e.g., lead to service impairment or service outage) and subsequently require remediation (e.g., require a hotfix, rollback, fix forward, patch)?</p>	0%-15%	16%-30%	16%-30%	16%-30%

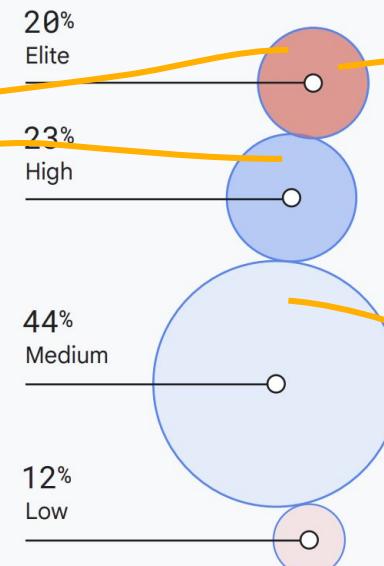
Accelerate: State of DevOps 2021, Smith et al, Google



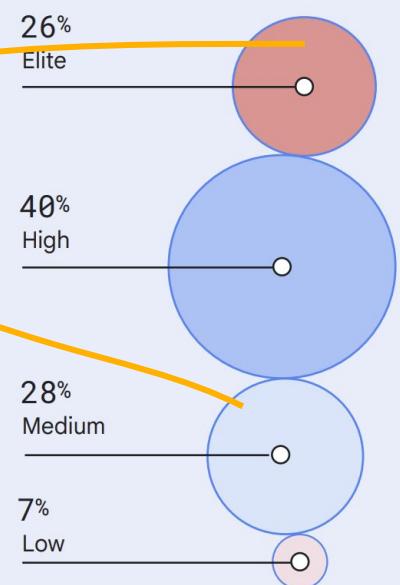
# 2018



# 2019

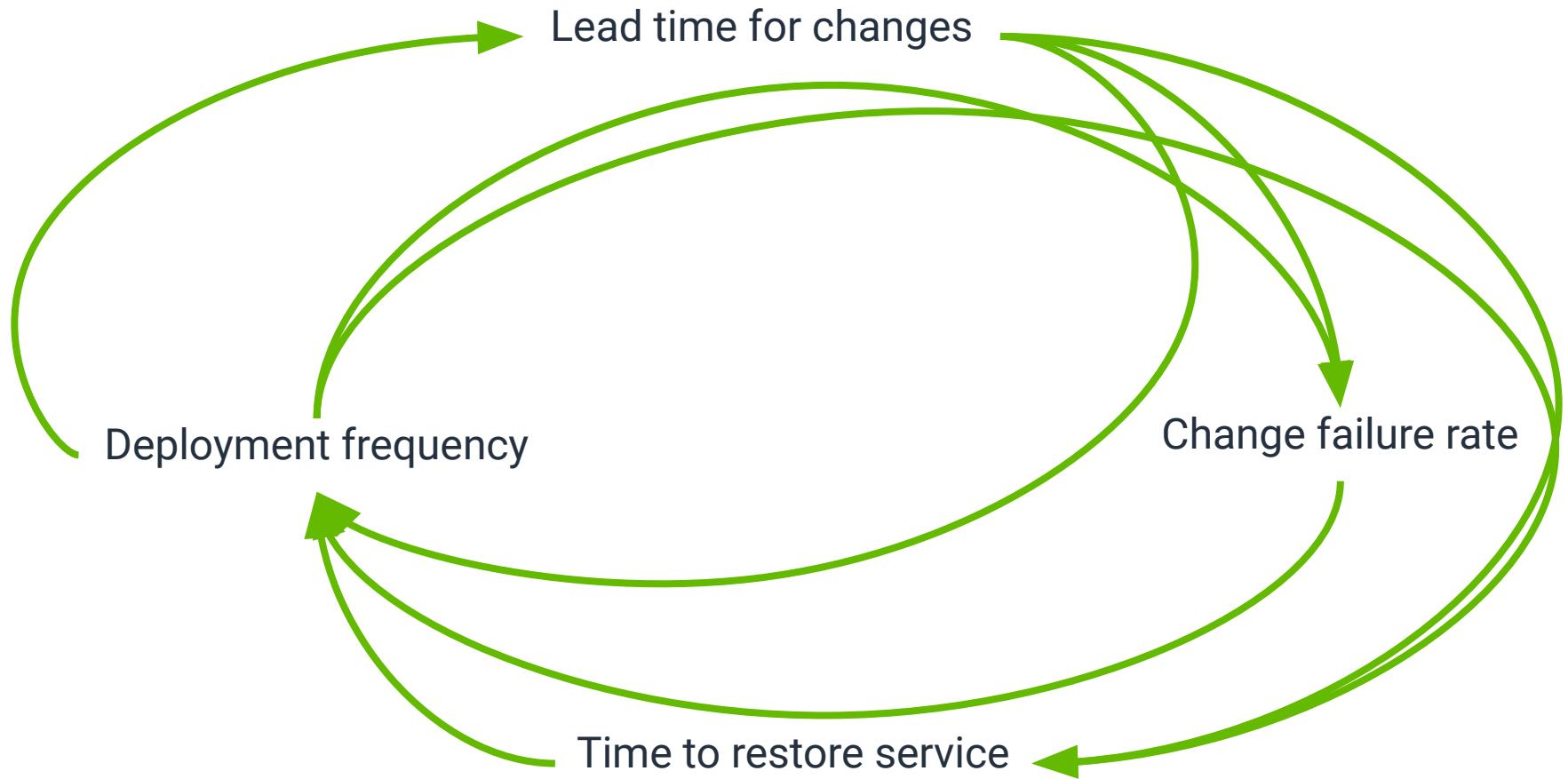


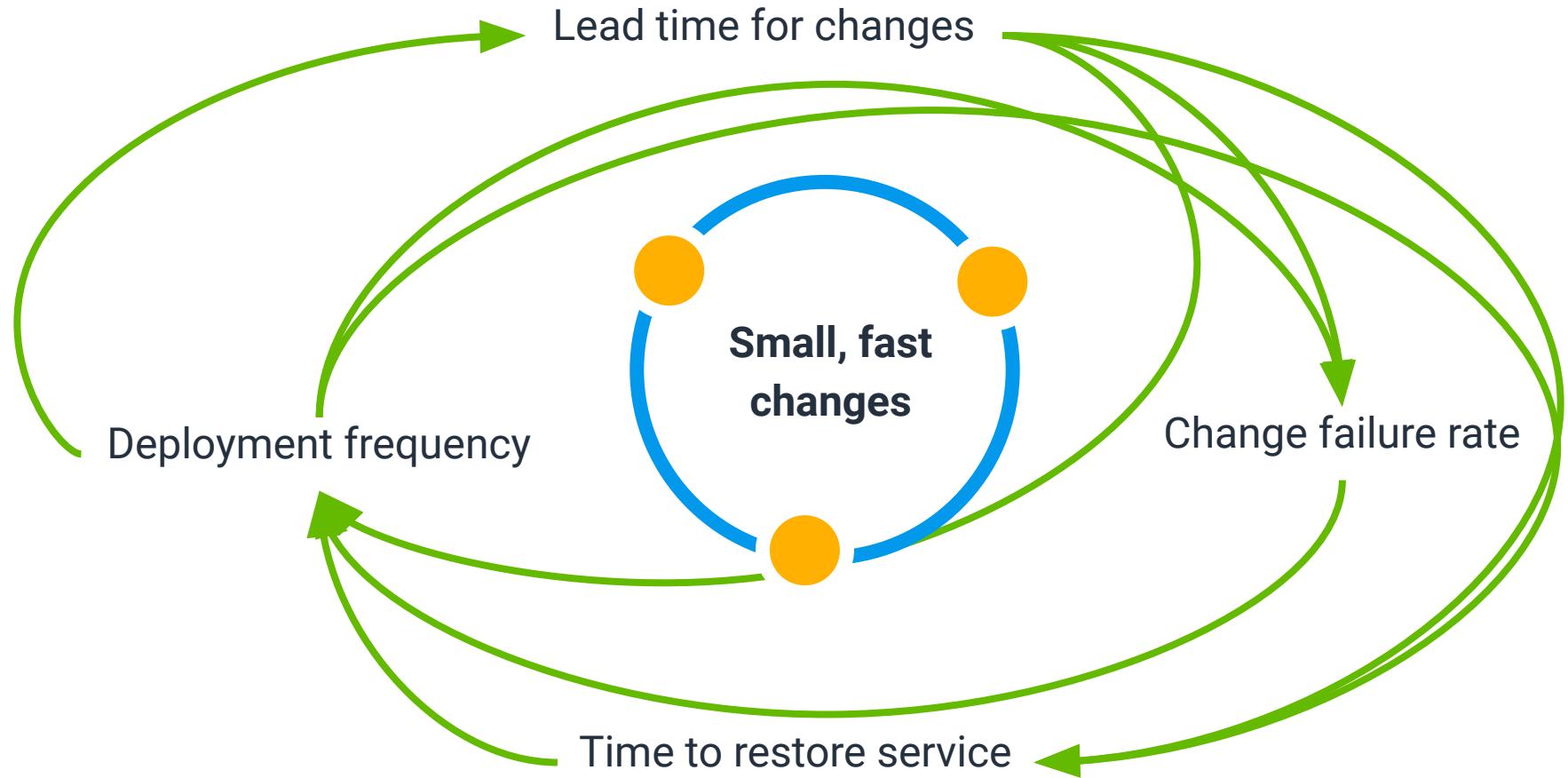
# 2021



Accelerate: State of  
DevOps 2021, Smith et  
al, Google









## Monitoring and observability

As with previous years, we found that monitoring and observability practices support continuous delivery. Elite performers who successfully meet their reliability targets are 4.1 times more likely to have solutions that incorporate observability into overall system health. Observability practices give your teams a better understanding of your systems, which decreases the time it takes to identify and troubleshoot issues. Our research also indicates that teams with good observability practices spend more time coding. One possible explanation for this finding is that implementing observability practices helps shift developer time away from searching for causes of issues toward troubleshooting and eventually back to coding.

## Monitoring and observability

As with previous years, we found that monitoring and observability practices support continuous delivery. Elite performers who successfully meet their reliability targets are 4.1 times more likely to have solutions that incorporate observability into overall system health. Observability practices give your teams a better understanding of your systems, which decreases the time it takes to identify and troubleshoot issues. Our research also indicates that teams with good observability practices spend more time coding. One possible explanation for this finding is that implementing observability practices helps shift developer time away from searching for causes of issues toward troubleshooting and eventually back to coding.

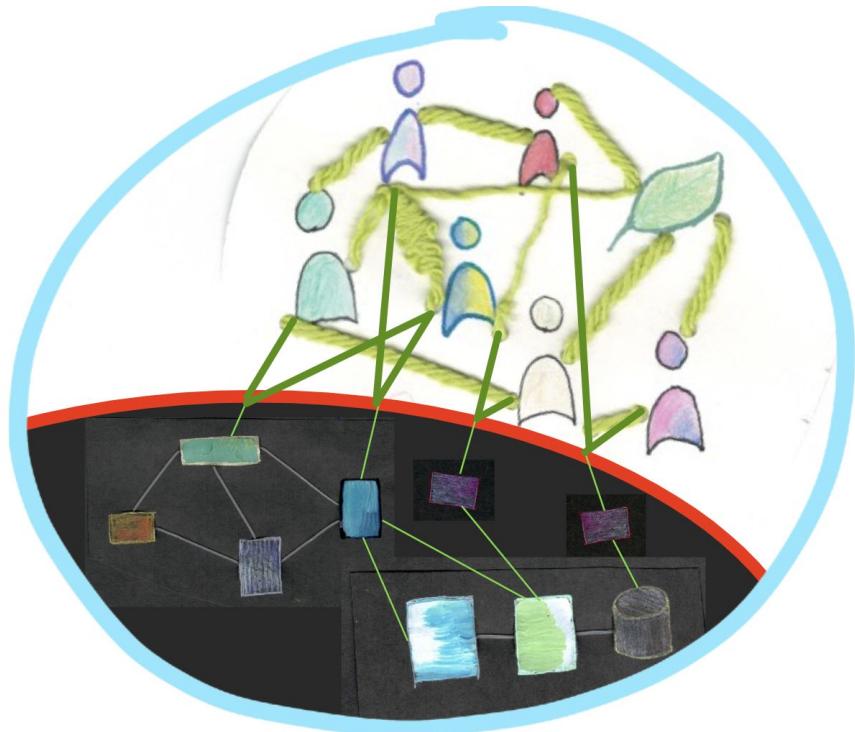
# Your software and tools are on your team.

The running software, plus the tools, plus the people on our team form a system,

a *sympathesy*: a learning system made of learning parts.

Our software and tools learn from us, because we change them.

Make the software teach you about itself and the world!



# so adapt your culture, not just the tools.

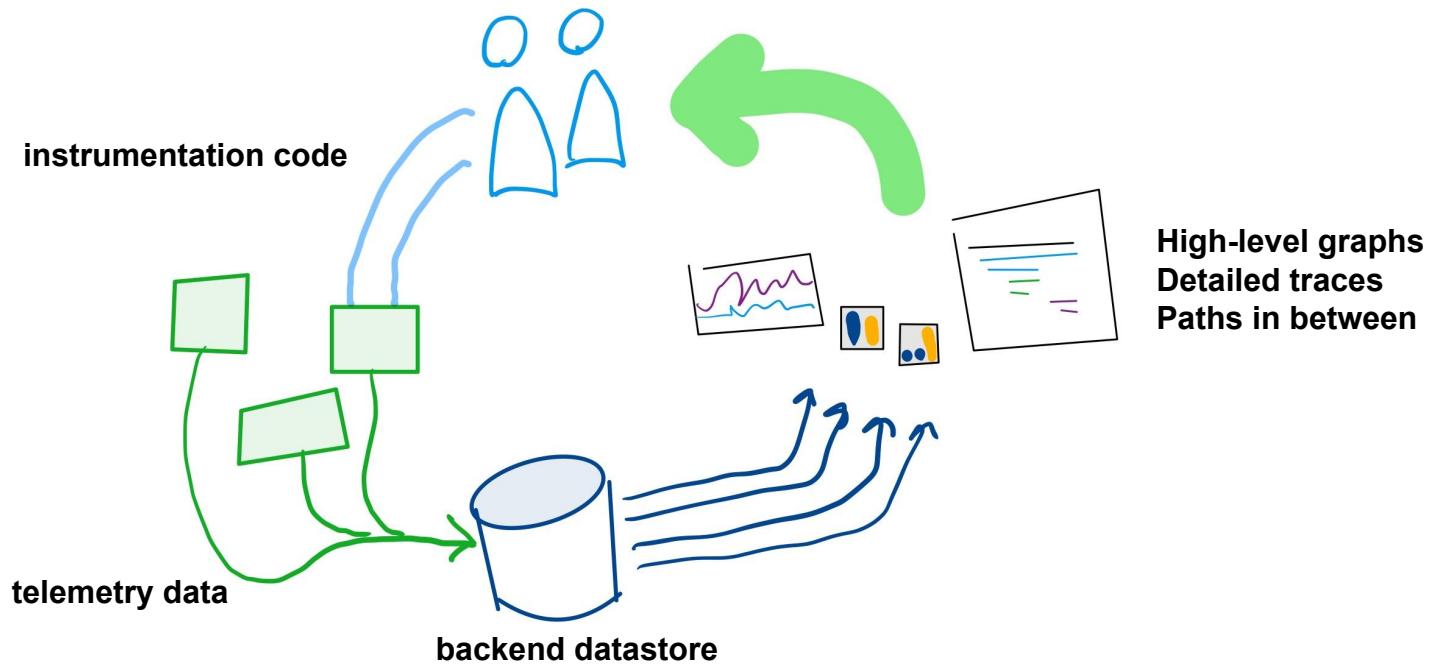
---

We need a culture of *curiosity*, not of fear.

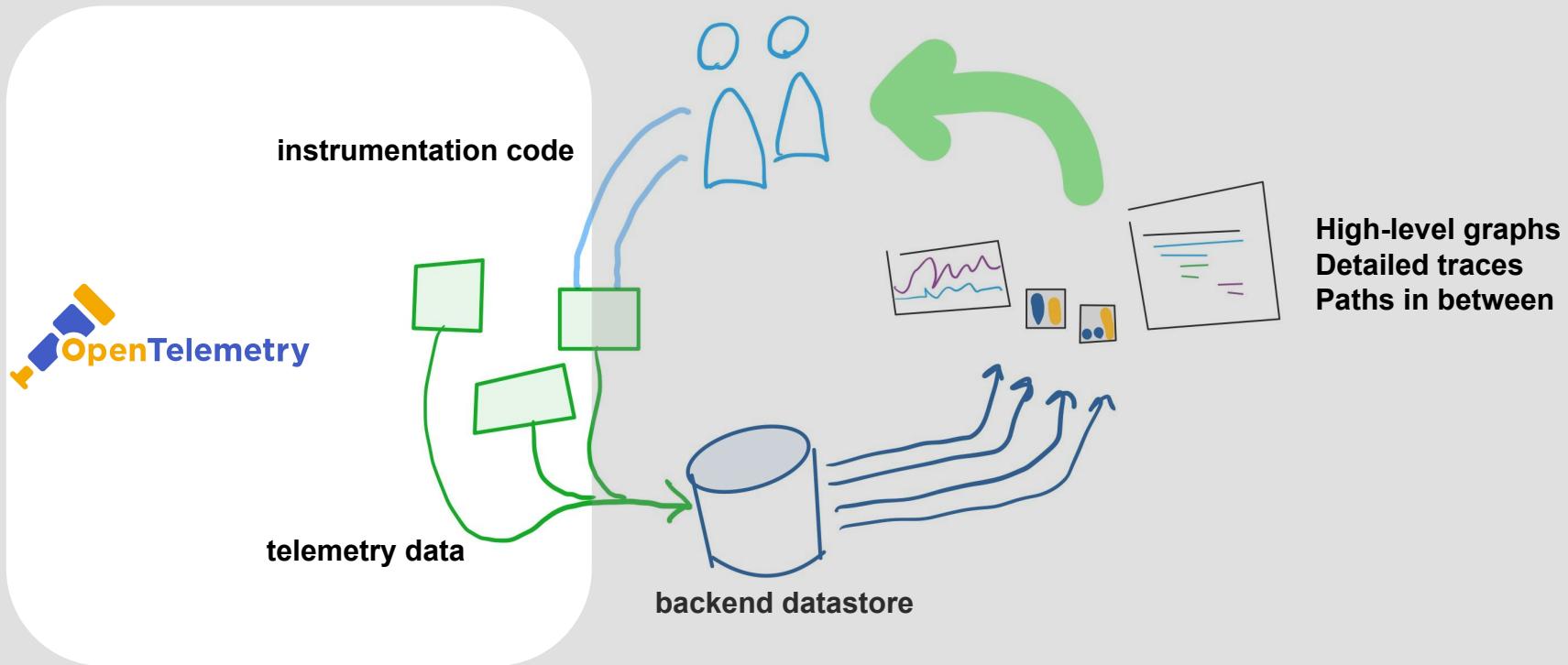
We need to accelerate feedback cycles.



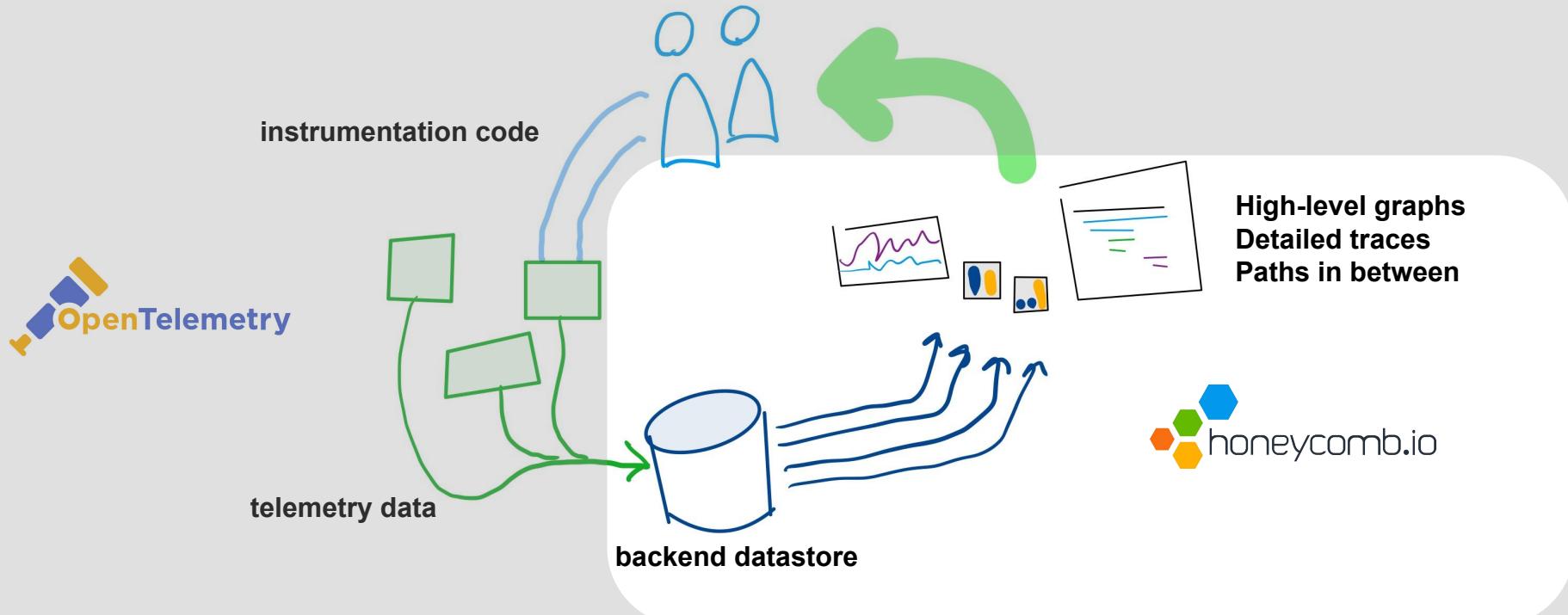
# How does observability work?



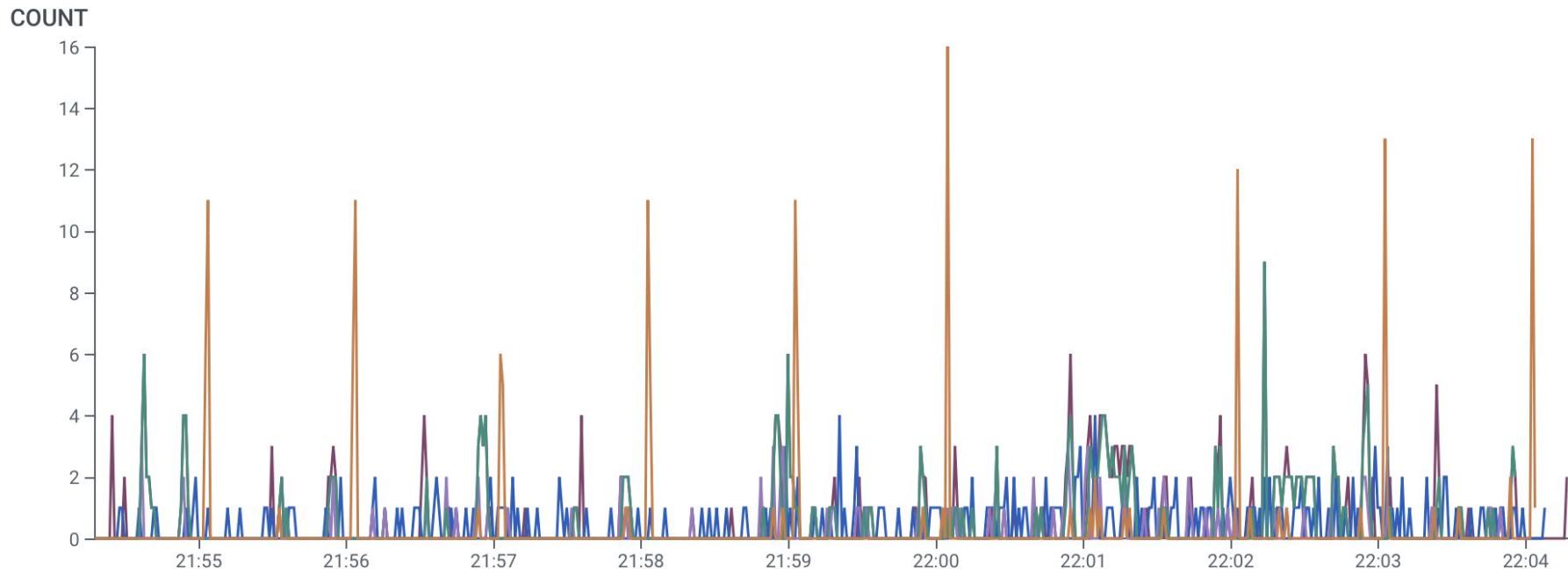
# How does observability work?



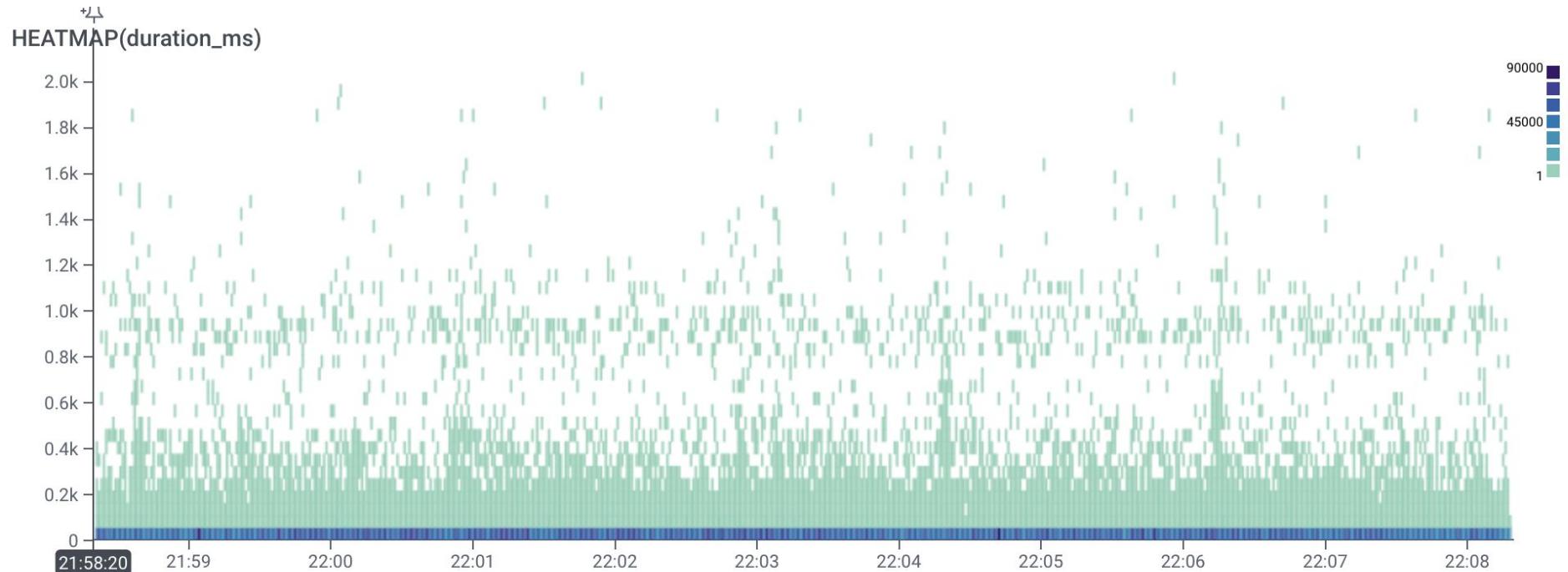
# How does observability work?



# Telemetry feeds graphs



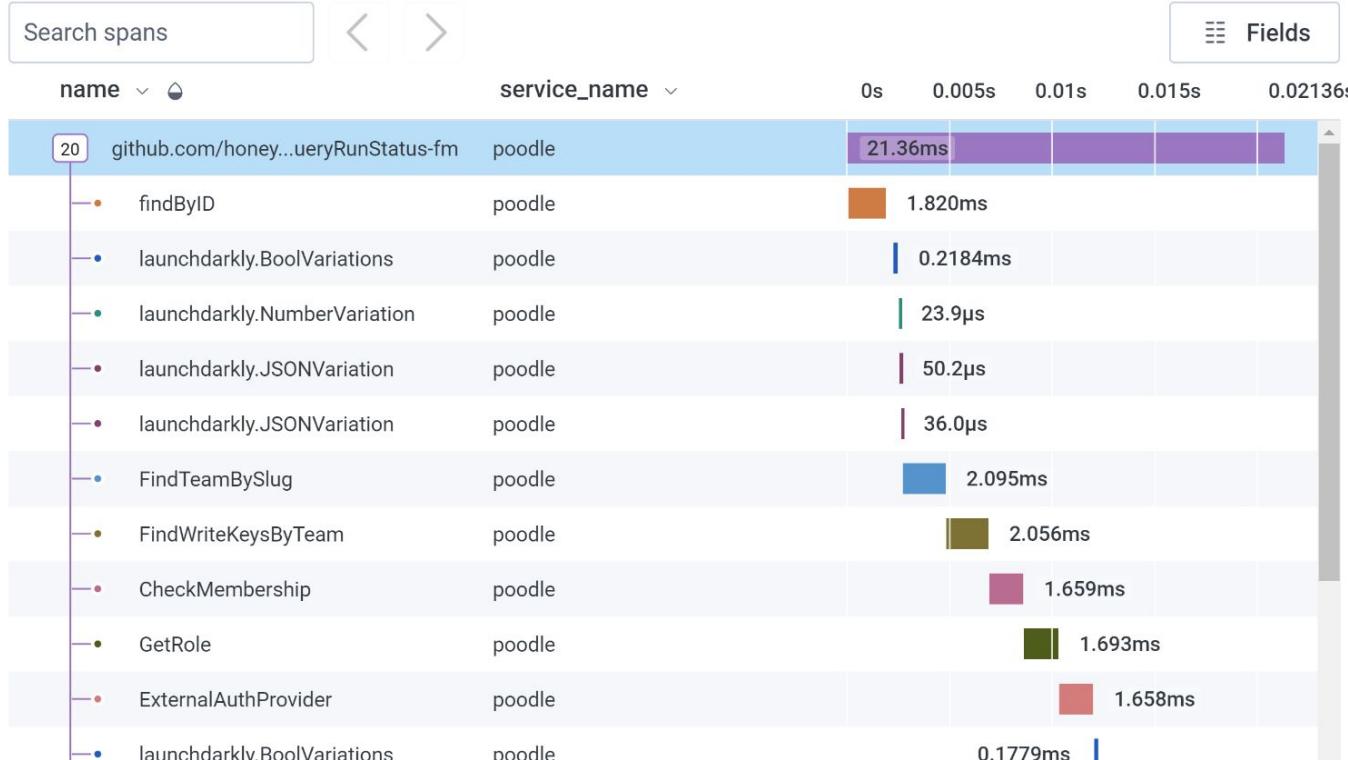
# Telemetry feeds heatmaps



# Telemetry feeds traces

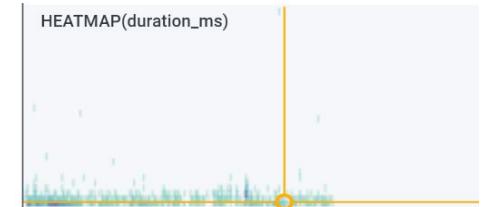
Trace 041f1cbd0b6095e0bdc73889ac20120c at 2021-10-29 10:42:44

Rerun



poole >  
github.com/honeycombi...andler).QueryRunStat...fm

Distribution of span duration



Fields

trace.

[str] trace.span\_id  
df9fc50200e3ad1c

[str] trace.trace\_id  
041f1cbd0b6095e0bdc73889ac20120c

# **Observability requires telemetry.**

---

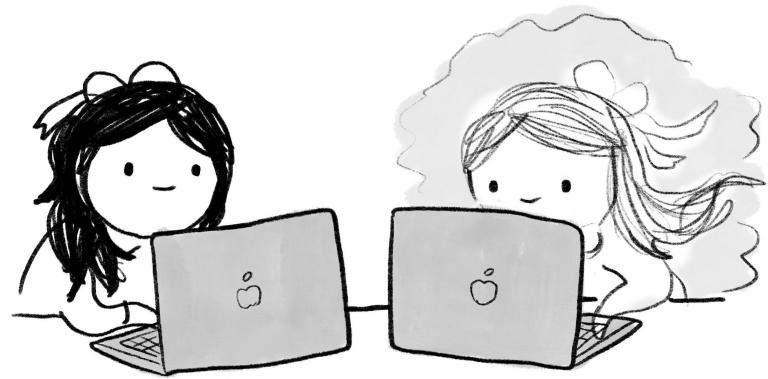
Let's get some data in!

# Section Overview

---

In this section, we will:

- Run a demo app instrumented with OpenTelemetry
- Get a Honeycomb Account
- Send telemetry from the demo app to Honeycomb



# Get the Demo App

---



[glitch.com/@honeycombio](https://glitch.com/@honeycombio)

- Node.js
- Python
- Go



<https://github.com/orgs/honeycombio/repositories?q=intro>

- Java
- .NET
- Node.js
- Python
- Go



**GitHub**  
(clone & run locally)

- Java
- .NET
- Node.js
- Python
- Go

# Choose the GitHub Repo for your language

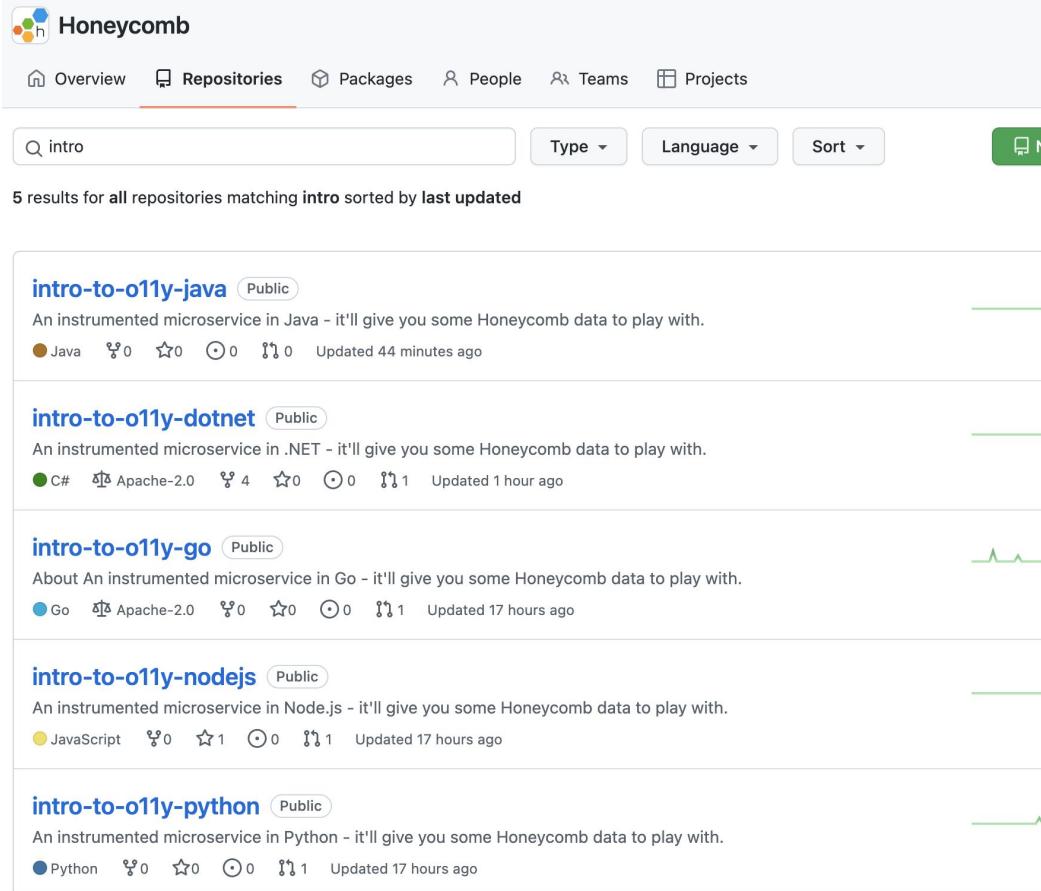
---

Go to <https://github.com/honeycombio>

Find a repository...

“intro”

Choose the repository (repo) for your language.



The screenshot shows a GitHub search results page for the query "intro". The results are sorted by "last updated". There are five repositories listed:

- intro-to-o11y-java** (Public)
 

An instrumented microservice in Java - it'll give you some Honeycomb data to play with.

Java 0 stars 0 forks 0 issues 0 Updated 44 minutes ago
- intro-to-o11y-dotnet** (Public)
 

An instrumented microservice in .NET - it'll give you some Honeycomb data to play with.

C# Apache-2.0 4 stars 0 forks 0 issues 1 Updated 1 hour ago
- intro-to-o11y-go** (Public)
 

About An instrumented microservice in Go - it'll give you some Honeycomb data to play with.

Go Apache-2.0 0 stars 0 forks 1 issues 1 Updated 17 hours ago
- intro-to-o11y-nodejs** (Public)
 

An instrumented microservice in Node.js - it'll give you some Honeycomb data to play with.

JavaScript 0 stars 1 forks 1 issues 1 Updated 17 hours ago
- intro-to-o11y-python** (Public)
 

An instrumented microservice in Python - it'll give you some Honeycomb data to play with.

Python 0 stars 0 forks 1 issues 1 Updated 17 hours ago



# Choose the GitHub Repo for your language

Go to <https://github.com/orgs/honeycombio/repositories?q=intro>

Choose the repository  
(repo) for your language.

Scroll down and  
select the Gitpod button  
in the repo's README.

update-dependencies.sh Local running 2 months ago

README.md

**Intro to Observability: OpenTelemetry in Go**

This application is here for you to try out tracing. It consists of a microservice that calls itself, so you can simulate a whole microservice ecosystem with just one service!

**What to do**

Recommended:

 Open in Gitpod

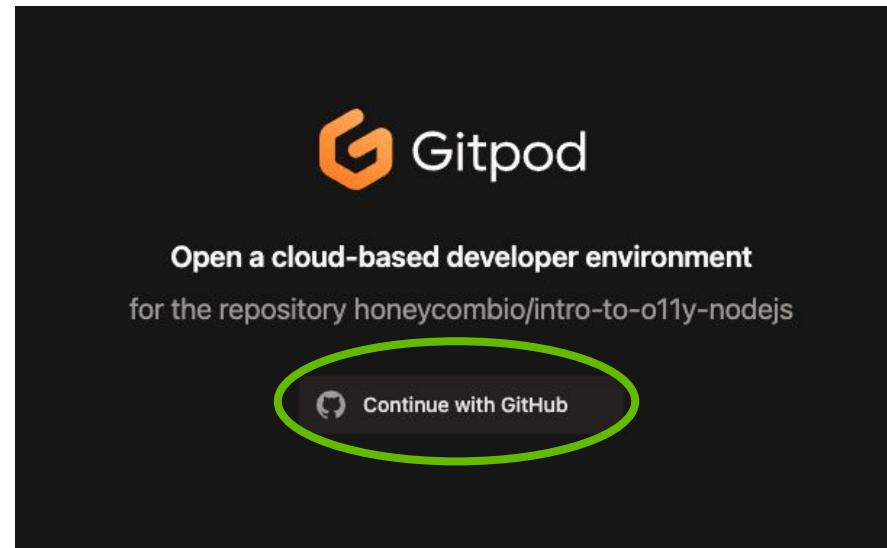


# Log in

---

A screen will appear and prompt you to sign-in with GitHub.

Select **Continue with GitHub**



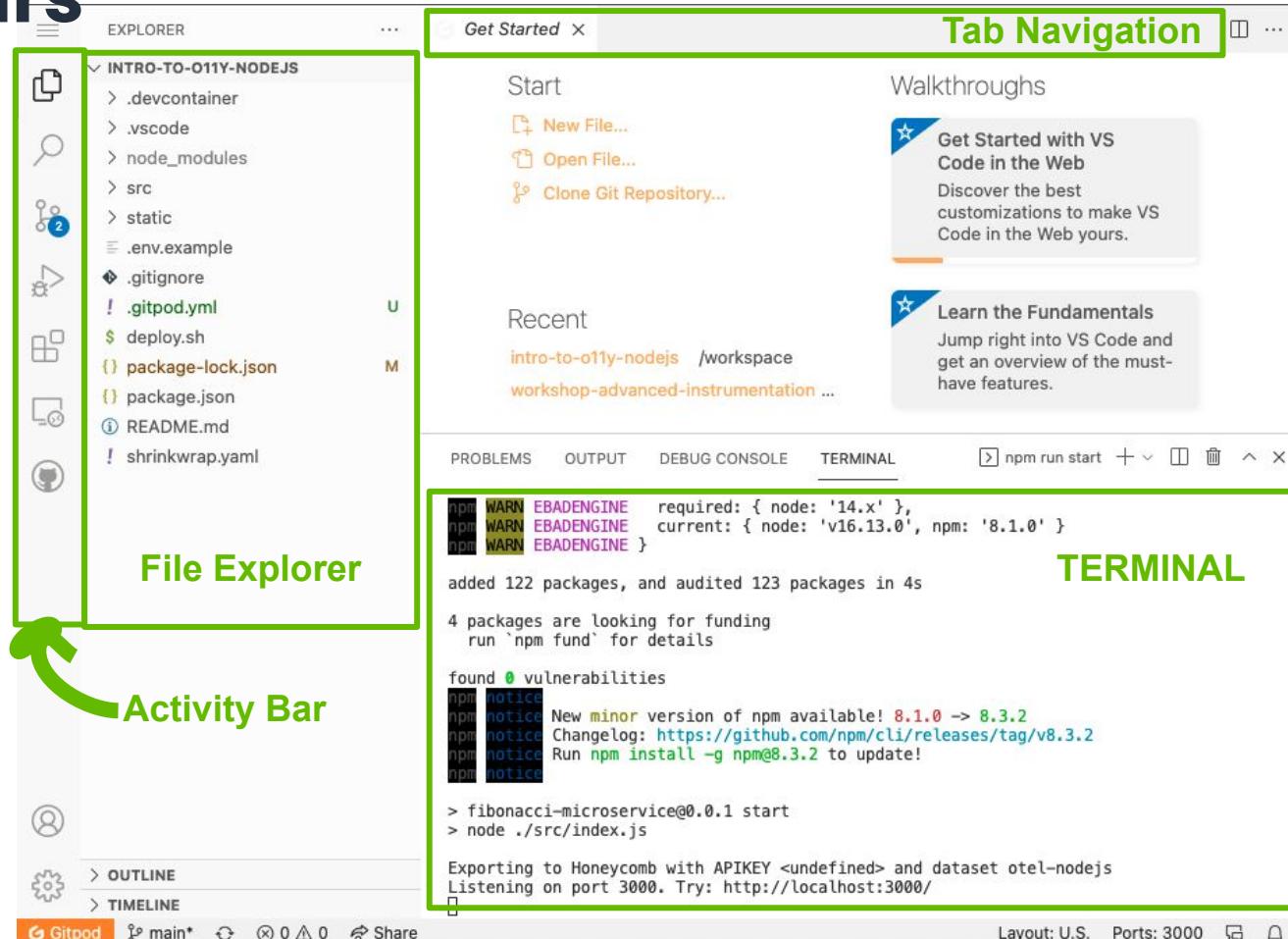
# Gitpod Appears

Layout Reflects VSCode

Important areas include:

- Activity Bar
- File Explorer
- Tab Navigation
- Terminal

Select README.md in the File Explorer to see instructions.



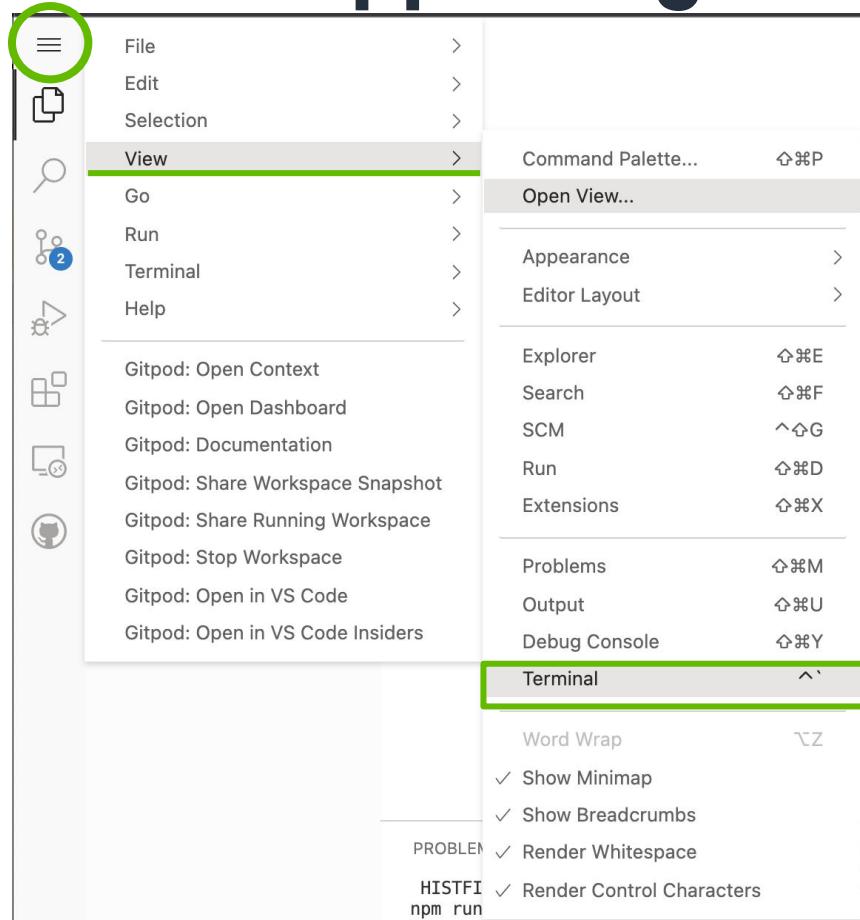
# Troubleshooting: Terminal not appearing

Go to the upper left corner to the

**Hamburger Menu > View > Terminal**

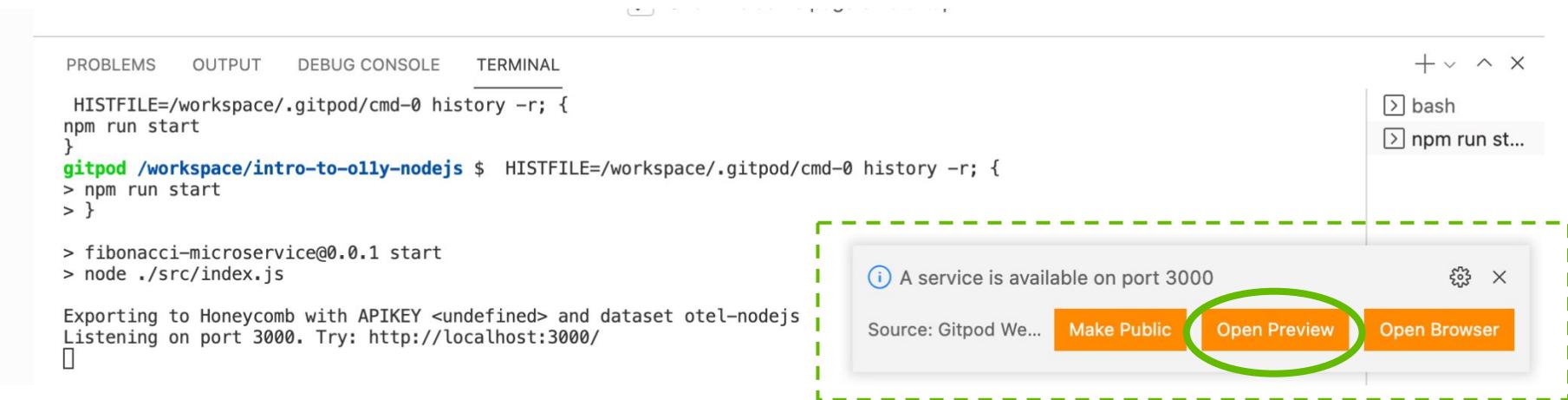
The terminal window will appear in the bottom half of the screen

Shortcut: Ctrl+J (Cmd+J on Mac)



# How to Open an App Preview

In the Terminal area, a "Service is available on port xxxx" pop-up may appear. Select **Open Preview** to open the app display in a new tab.



The screenshot shows a terminal window within a code editor interface. The terminal tab is active, displaying the following command history:

```
HISTFILE=/workspace/.gitpod/cmd-0 history -r; {  
npm run start  
}  
gitpod /workspace/intro-to-olly-nodejs $ HISTFILE=/workspace/.gitpod/cmd-0 history -r; {  
> npm run start  
> }  
  
> fibonacci-microservice@0.0.1 start  
> node ./src/index.js  
  
Exporting to Honeycomb with APIKEY <undefined> and dataset otel-nodejs  
Listening on port 3000. Try: http://localhost:3000/  
[ ]
```

A green dashed box highlights a pop-up message in the terminal area:

A service is available on port 3000

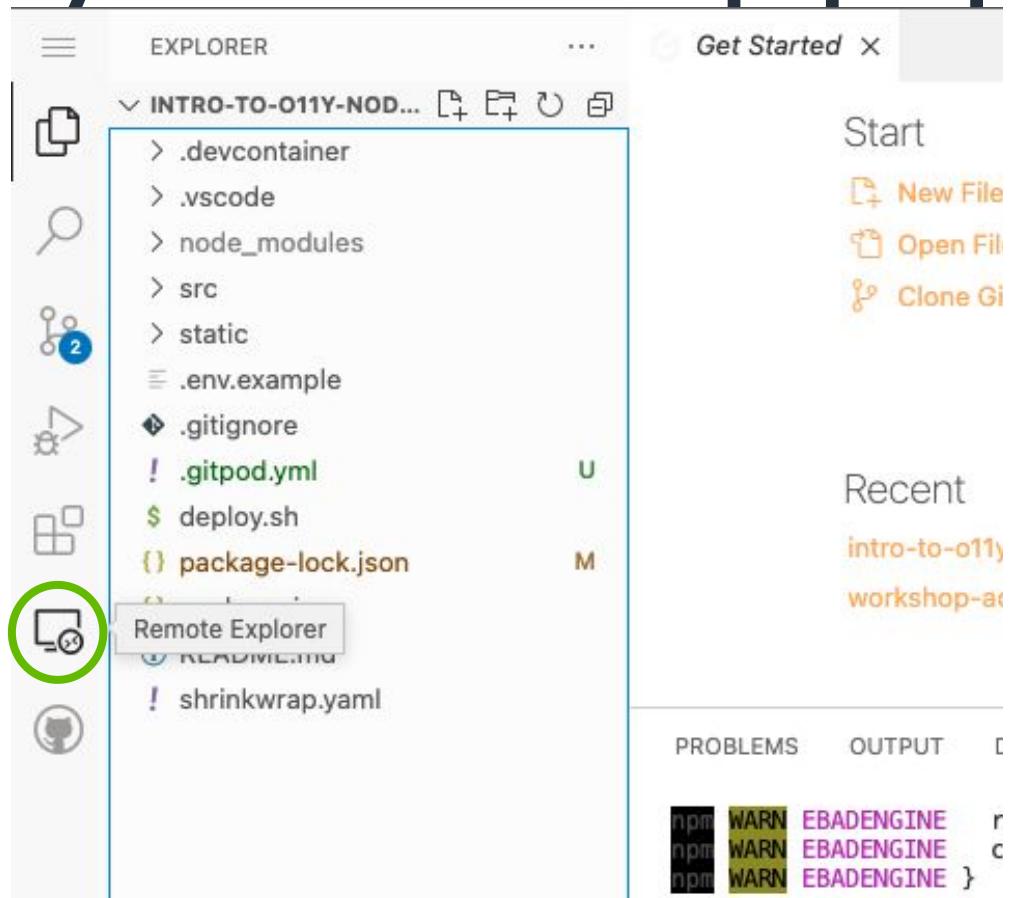
Source: Gitpod We... **Make Public** **Open Preview** **Open Browser**

The **Open Preview** button is circled with a green oval.



# Troubleshooting: If you don't see the pop-up

Go to **Remote Explorer**.

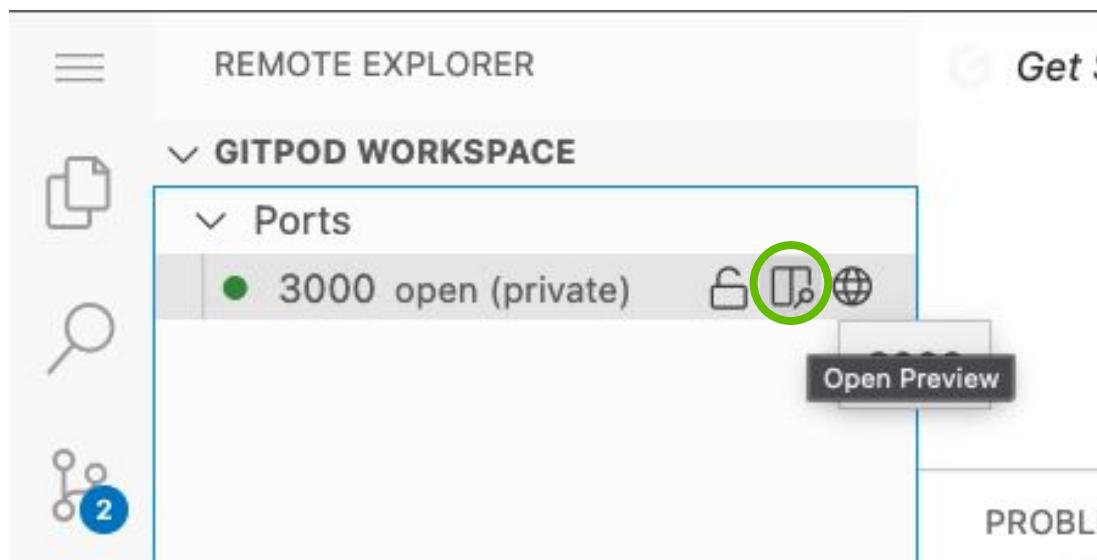


# Troubleshooting: If you don't see the pop-up

Go to Remote Explorer.

Expand *Ports* to see the listed port number. For example, 3000

Choose the middle ***Open Preview*** icon for the app to appear in a new tab.



# Try out the app

---

A sequence of numbers:

Go

Push Go!

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 

Stop

Push Stop!

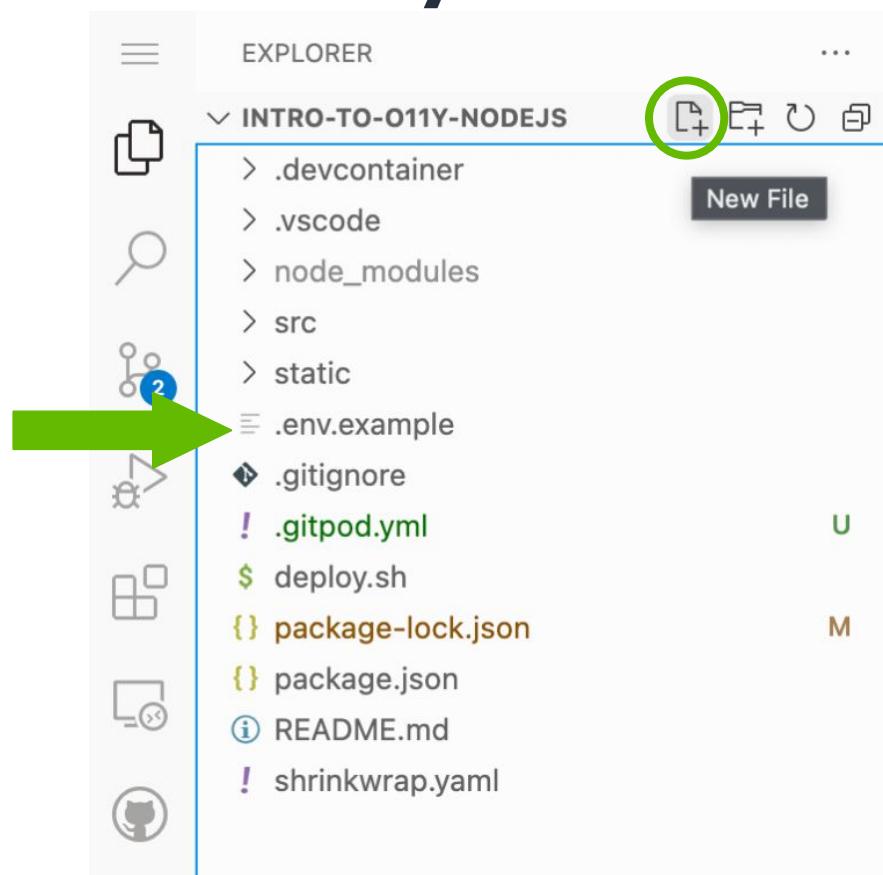


# Add credentials to send to Honeycomb

Provide your credential variables in the **.env** file.

Using the File Explorer:

- Create a new file named **.env**
- Copy the contents of **.env.example** into the **.env** file
- Replace the credential variables in the **.env** file as needed



# Add credentials to send to Honeycomb

Provide your credentials!

- HONEYCOMB\_API\_KEY identifies your team and environment.
  - Enter your API key from the Honeycomb UI
- SERVICE\_NAME groups your data



```
1 # Honeycomb API key.  
2 export HONEYCOMB_API_KEY=your-api-key-goes-here  
3  
4 # Service name (opentelemetry standard field). Can b  
5 SERVICE_NAME=fib-microsvc  
6
```



# Get a Honeycomb account

---

<https://honeycomb.io/signup>

Use the same email you used to register for this workshop.



## Create account

No credit card required.

First name

Last name

Email address (work)

By signing up, I agree to the Honeycomb Terms of Service.

**Sign up**

or



**Sign up with Google**

Already have an account? [Sign in](#)



# Create a team

---

*Note to Existing Users:*  
Go to [ui.honeycomb.io/teams](https://ui.honeycomb.io/teams)  
At the bottom, find “Create Team”

## Create new team

Teams organize members' access to data and shared work history in Honeycomb. We recommend using your organization name, if you'll be inviting collaborators to your account later.

We've provided a default team name for you based on your email address. You can change it here if you'd prefer a different name.

Team name

jessitron

Create Team



# Get a Honeycomb API Key

The first time you create a team as a new user, it takes you directly to your API Key.



We are waiting for you to send us data. Please follow the instructions below to send events from your application

Last checked 9:49:10 PM

[Check now](#)

## Send data to test

Honeycomb's auto-instrumentation allows you to send basic data quickly using the OpenTelemetry industry standard, before adding custom context.

[View these instructions in Honeycomb Docs.](#) ↗

API key ⓘ

.....

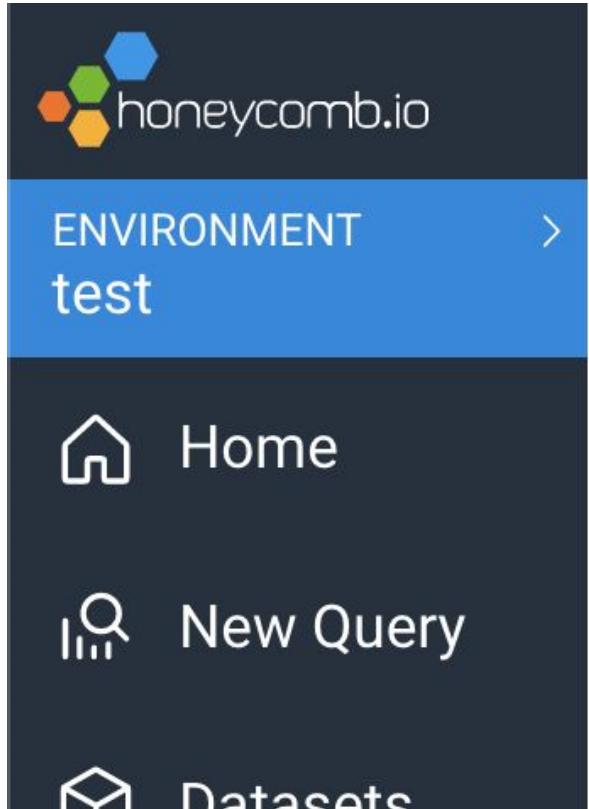
[Manage API keys](#)



# Get a Honeycomb API Key

If you already have an account:

1. Click the environment selector at the top left
2. Choose **Manage environments** at the bottom.
3. View API Keys.



# Add credentials to send events to Honeycomb

Provide your credentials!

- HONEYCOMB\_API\_KEY identifies your team and environment.
  - Enter your API key from the Honeycomb UI
- SERVICE\_NAME groups your data



```
1 # Honeycomb API key.  
2 export HONEYCOMB_API_KEY=your-api-key-goes-here  
3  
4 # Service name (opentelemetry standard field). Can b  
5 SERVICE_NAME=fib-microsvc  
6
```



# Re-run Your App

---

1) If your app is still running, stop it with  
Ctrl+C

2) Save your .env file.

3) In the terminal, run the app with:

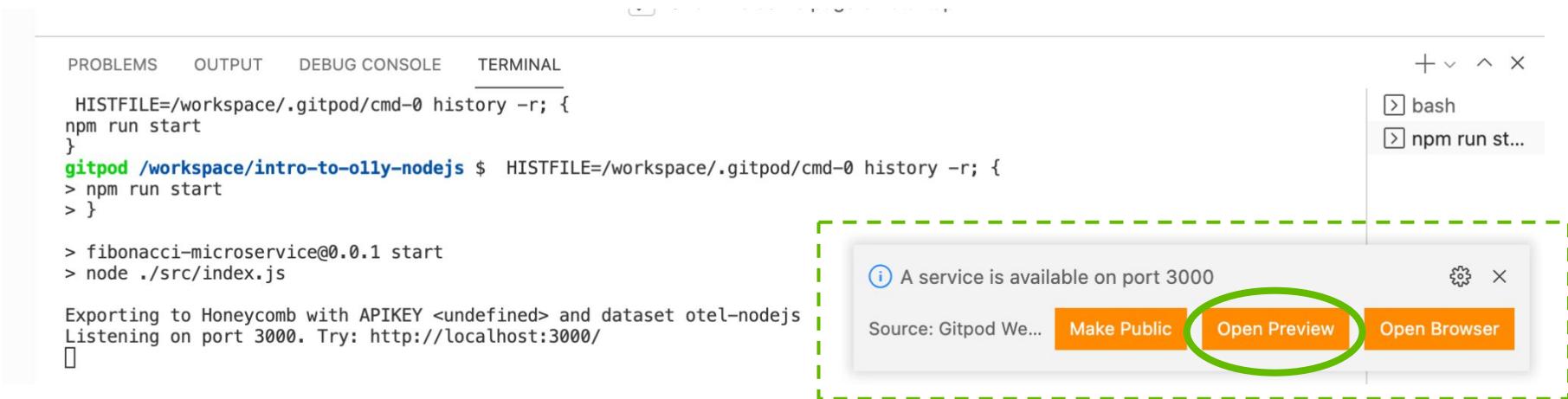
**./run**

Then, click “Go” and “Stop” in the app.



# How to Open an App Preview

In the Terminal area, a "Service is available on port xxxx" pop-up may appear. Select **Open Preview** to open the app display in a new tab.



The screenshot shows a terminal window within a development environment. The terminal tab is active, displaying the following command history:

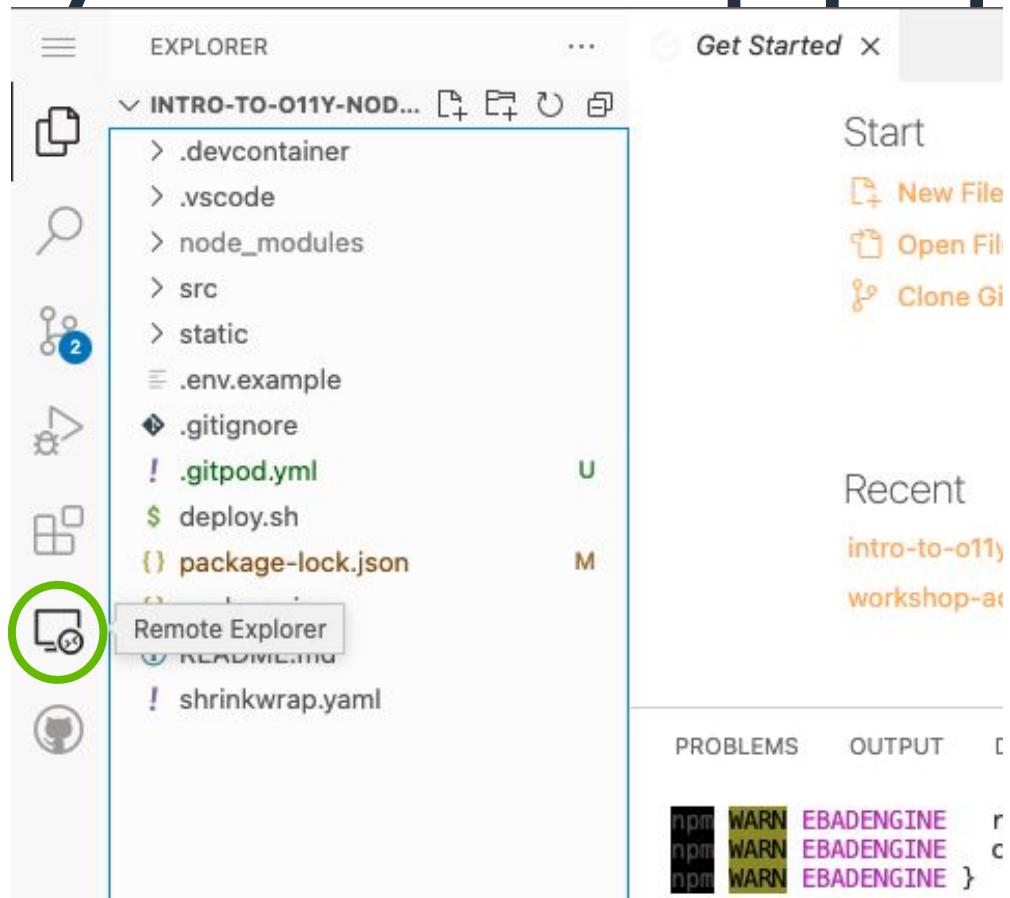
```
HISTFILE=/workspace/.gitpod/cmd-0 history -r; {  
npm run start  
}  
gitpod /workspace/intro-to-olly-nodejs $ HISTFILE=/workspace/.gitpod/cmd-0 history -r; {  
> npm run start  
> }  
  
> fibonacci-microservice@0.0.1 start  
> node ./src/index.js  
  
Exporting to Honeycomb with APIKEY <undefined> and dataset otel-nodejs  
Listening on port 3000. Try: http://localhost:3000/  
[ ]
```

A green dashed box highlights a tooltip that appears over the terminal output. The tooltip contains the message: "A service is available on port 3000". It includes three buttons: "Source: Gitpod We...", "Make Public", "Open Preview", and "Open Browser". The "Open Preview" button is circled with a green oval.



# Troubleshooting: If you don't see the pop-up

Go to **Remote Explorer**.

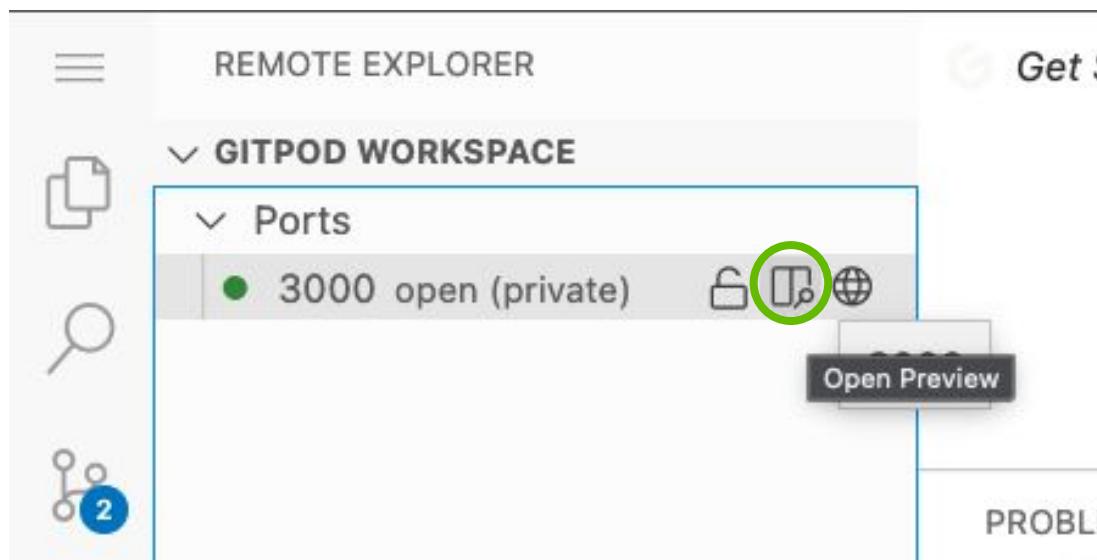


# Troubleshooting: If you don't see the pop-up

Go to Remote Explorer.

Expand *Ports* to see the listed port number. For example, 3000

Choose the middle ***Open Preview*** icon for the app to appear in a new tab.



# Use the Gitpod App again

---

Make the sequence of numbers appear.

Select **Go!**

... and then select **Stop**, after 5-7 numbers.

Repeat as needed to generate data.

## A sequence of numbers:

**Go**

0, 1, 1, 2, 3, 5, 8, 13, 21, 34,



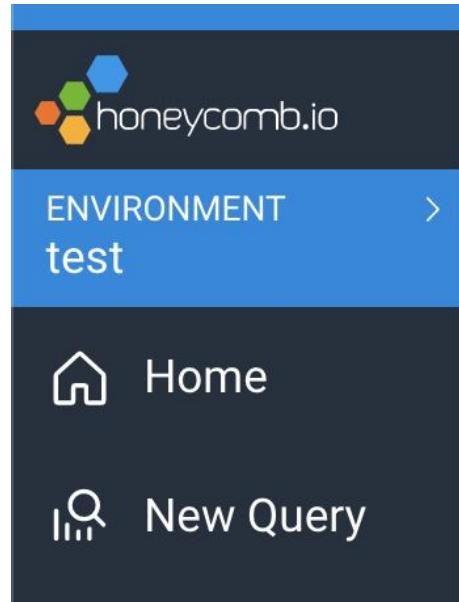
**Stop**

# Data should start flowing!

The blue box turns green

- ✓ Data received! Events from your application are now available to explore in Honeycomb. [Explore your data](#)

Click “Home”



# Home

---

Got data?  
Take a break.

Ask questions in  
break-out if you  
get stuck.



# Home

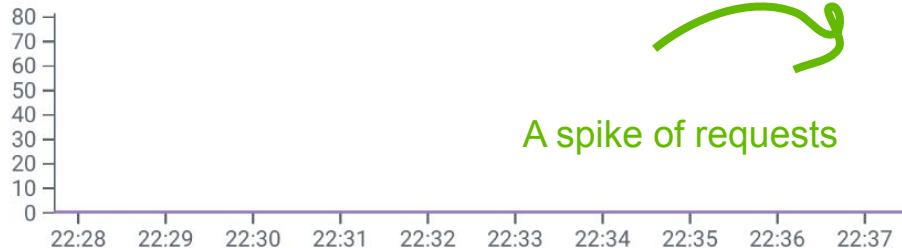
sequence-of-numbers

New Query

Dataset Settings

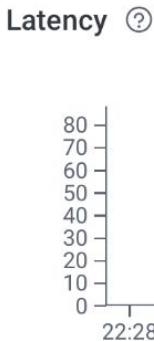
Overall    HTTP Status Code    Service    Route    Error    User    [?](#)

Total spans [?](#)



Error rate [?](#)

Based on http status code



# Summary

---

In this section, we:

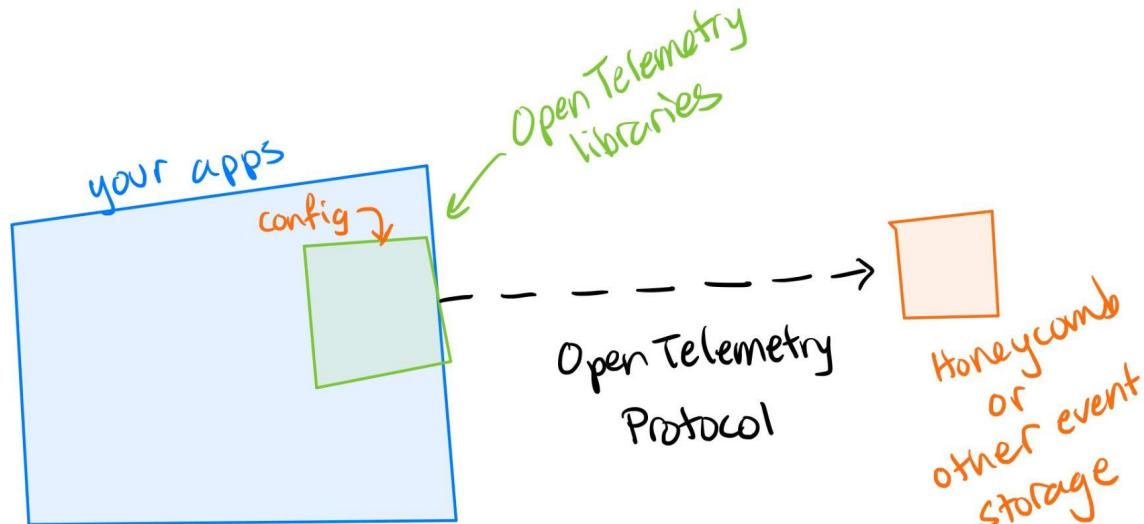
- Ran a demo app that is instrumented with OpenTelemetry
- Got a Honeycomb Account
- Sent telemetry from the demo app to Honeycomb
- Confirmed data ingestion from the Honeycomb home screen



# This is OpenTelemetry

OpenTelemetry is a vendor-neutral standard for instrumentation.

It has an instrumentation API, various SDKs, proxies, and wire formats pre-standardized.



# Using data to answer questions

---

Learn to use Honeycomb's product UI!

# Section Overview

---

In this section, we will:

- Learn about Honeycomb UI and tools
- Find out what makes some calls to our Fibonacci app slow



# Home

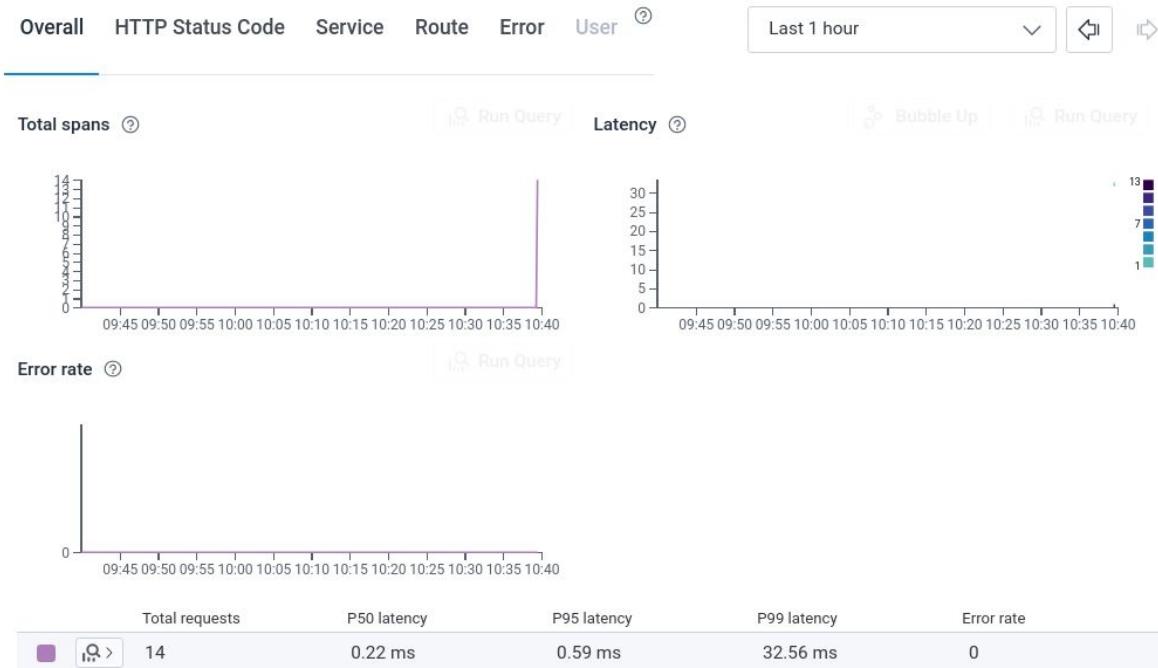
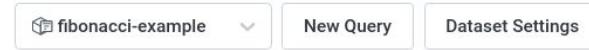
---

The basics:

Rate, error, duration graphs

You can always come to Home via the Honeycomb logo at top left.

## Home



# Let's find & examine a trace!

After using the app...

Look for the trace in Recent Traces tab at the bottom of the Home screen.

Use the **View Trace** button on the row you want to examine further.

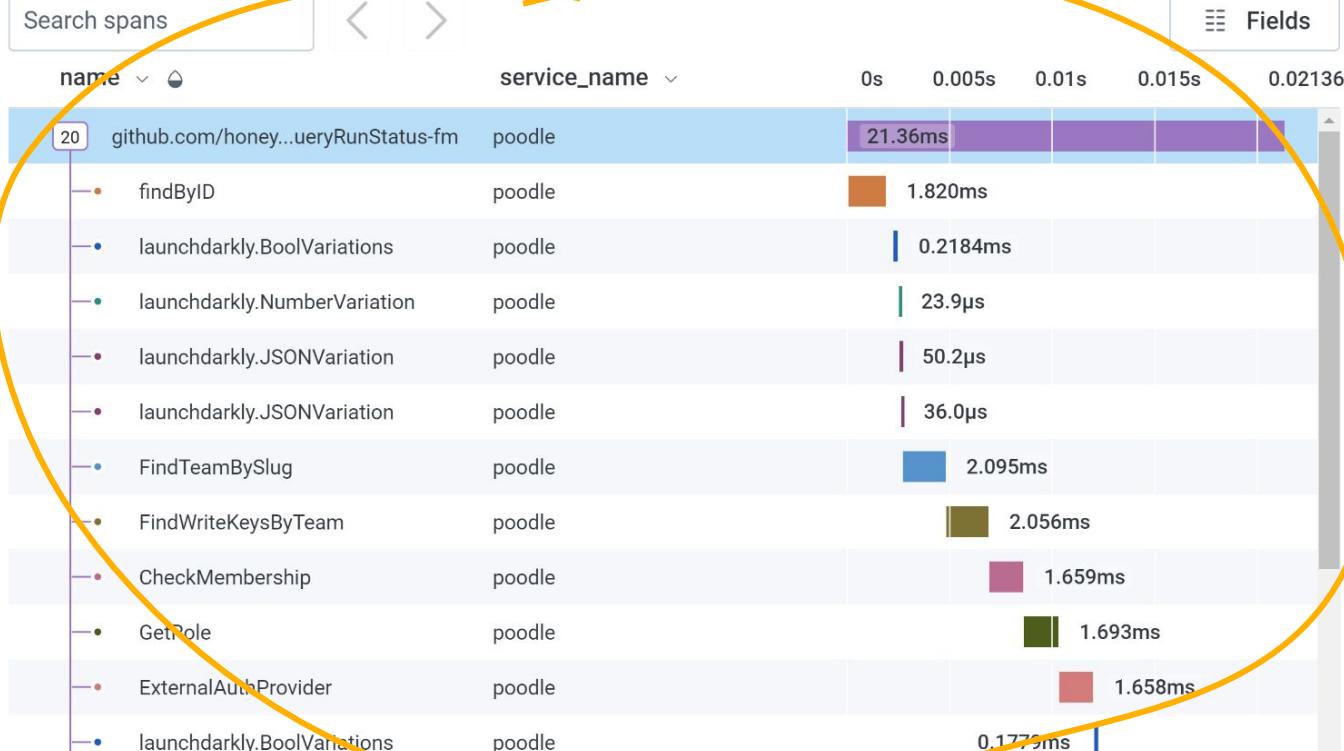
The screenshot shows the Hound app's home screen. At the top, there are two tabs: "Recent Traces" (which is highlighted with a green oval) and "Recent Events". Below the tabs, a subtitle reads "The traces with the most recent root spans". A large green arrow points from the text "Use the View Trace button on the row you want to examine further." towards the screenshot. In the main area, there is a table with five rows, each representing a trace. The columns are labeled "Time" and "Service". The "Time" column shows "7 minutes ago" for all rows. The "Service" column shows a service icon for each row. The first row has a "View Trace" button (represented by a square with a right-pointing arrow) circled in green, indicating it is the target for examination.

Time	Service
7 minutes ago	

# Parts of a trace

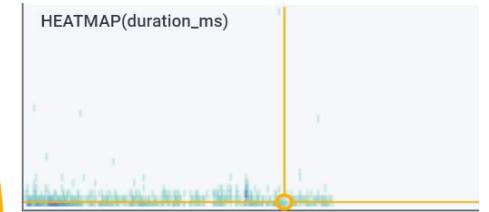
Trace 041f1cbd0b6095e0bdc73889ac20120c at 2021-10-29 10:42:44

Rerun



poole >  
github.com/honeycombi  
o...andler).QueryRunStat  
fm

Distribution of span duration



Fields

trace.|

[str] trace.span\_id  
df9fc50200e3ad1c  
[str] trace.trace\_id  
041f1cbd0b6095e0bdc73889ac20120c

# Parts of a trace

Trace 041f1cbd0b6095e0bdc73889ac20120c at 2021-10-29 10:42:44

Rerun

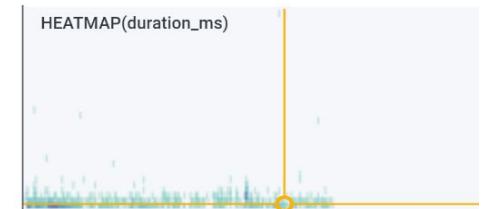


Fields

name	service_name	span	0s	0.005s	0.01s	0.015s	0.02136s
20 github.com/honey...QueryRunStatus-fm	poodle	span	21.36ms				
• findByID	poodle	span	1.820ms				
• launchdarkly.BoolVariations	poodle	span	0.2184ms				
• launchdarkly.NumberVariation	poodle	span	23.9µs				
• launchdarkly.JSONVariation	poodle	span	50.2µs				
• launchdarkly.JSONVariation	poodle	...	36.0µs				
• FindTeamBySlug	poodle		2.095ms				
• FindWriteKeysByTeam	poodle		2.056ms				
• CheckMembership	poodle		1.659ms				
• GetRole	poodle		1.693ms				
• ExternalAuthProvider	poodle		1.658ms				
• launchdarkly.BoolVariations	poodle		0.1779ms				

poole >  
github.com/honeycombi...andler).QueryRunStat...fm

Distribution of span duration



Fields

trace.

[str] trace.span\_id  
df9fc50200e3ad1c

[str] trace.trace\_id  
041f1cbd0b6095e0bdc73889ac20120c

# Parts of a trace

Trace 041f1cbd0b6095e0bdc73889ac20120c at 2021-10-29 10:42:44

Rerun

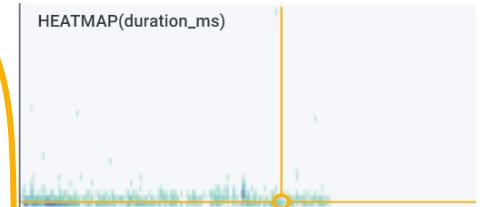


Fields

name	service_name		0s	0.005s	0.01s	0.015s	0.02136s
20 github.com/honey...QueryRunStatus-fm	poole	root span	21.36ms				
• findById	poole		1.820ms				
• launchdarkly.BoolVariations	poole		0.2184ms				
• launchdarkly.NumberVariation	poole		23.9µs				
• launchdarkly.JSONVariation	poole		50.2µs				
• launchdarkly.JSONVariation	poole		36.0µs				
• FindTeamBySlug	poole		2.095ms				
• FindWriteKeysByTeam	poole		2.056ms				
• CheckMembership	poole		1.659ms				
• GetRole	poole		1.693ms				
• ExternalAuthProvider	poole		1.658ms				
• launchdarkly.BoolVariations	poole		0.1779ms				

poole >  
github.com/honeycombi...andler).QueryRunStat...fm

Distribution of span duration



Fields

trace.

[str] trace.span\_id  
df9fc50200e3ad1c  
  
[str] trace.trace\_id  
041f1cbd0b6095e0bdc73889ac20120c

# Parts of a trace

Trace 041f1cbd0b6095e0bdc73889ac20120c at 2021-10-29 10:42:44

Rerun

Search spans



Fields

name ▾

service\_name ▾

0s

0.005s

0.01s

0.015s

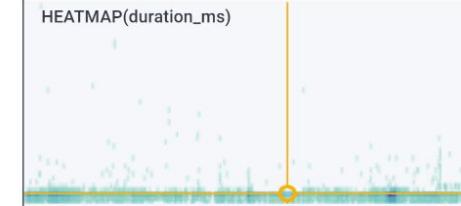
0.02136s

20	github.com/honey...ueryRunStatus-fm	poodle	21.36ms
•	findById	poodle	1.820ms
•	launchdarkly.BoolVariations	poodle	0.2184ms
•	launchdarkly.NumberVariation	poodle	23.9µs
•	launchdarkly.JSONVariation	poodle	50.2µs
•	launchdarkly.JSONVariation	poodle	36.0µs
•	FindTeamBySlug	poodle	2.095ms
•	FindWriteKeysByTeam	poodle	2.056ms
•	CheckMembership	poodle	1.659ms
•	GetRole	poodle	1.693ms
•	ExternalAuthProvider	poodle	1.658ms
•	launchdarkly.BoolVariations	poodle	0.1770ms

span

poole >  
FindTeamBySlug

Distribution of span duration



Fields

trace.

str trace.parent\_id

df9fc50200e3ad1c

str trace.span\_id

4da124d8596548ff

str trace.trace\_id

041f1cbd0b6095e0bdc73889ac20120c

# Raw data view

Also, the **Recent Events** tab at bottom of **Home** view shows raw JSON for events.

The screenshot shows the 'Recent Events' tab selected in a user interface. Below the tab, a message reads 'The most recent events sent to your dataset' with a help icon. A green arrow points from the text 'raw JSON for events.' in the previous slide to the JSON event data shown here. The data is presented in a table with columns: Time, HTTP Status Code, Service, and Route. Two events are listed:

Time	HTTP Status Code	Service	Route
5:04 AM			/sequence.js
5:04 AM			/

Below the table, a large green box highlights the first event's JSON object, which is partially visible:

```
{  
  "duration_ms": 0.305322,  
  "http.client_ip": "195.213.102.18,::ffff:10.10.10.42,::ffff:10.10.92.17",  
  "http.flavor": "1.1",  
  "http.host": "opentelemetry-instructor.glitch.me",  
  "http.method": "GET",  
  "http.route": "/",  
  "http.scheme": "http",  
  "http.server_name": "root",  
  "http.status_code": 200,  
  "http.target": "/",  
  "http.user_agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like  
  "http.wrote_bytes": 36,  
  "library.name": "go.opentelemetry.io/contrib/instrumentation/net/http/otelhttp",  
  "library.version": "semver:0.20.0",  
}
```



# Raw data view

Sometimes you need to look at spans as wide events/structured logs!

Honeycomb supports showing a table of spans and fields on those spans.

Go to the **Raw Data** tab.

**RESULTS**

**BubbleUp**

**Traces**

**Raw Data**

**Graph Settings**

Sep 23 2021, 2:18 PM – Sep 23 2021, 2:52 PM (Granularity: 5 sec)

DOWNLOAD	Timestamp	duration_ms	http.client_ip	http.flavor	http.host
CSV   JSON	2021-09-23 14:38:01.636	1,391.055	24.125.233.208	1.1	intro-to-o11y-nodejs.glitch.
		1			
	2021-09-23 14:38:00.579	1,007.479	24.125.233.208	1.1	intro-to-o11y-nodejs.glitch.
		81			
	2021-09-23 14:38:00.039	479.43142	24.125.233.208	1.1	intro-to-o11y-nodejs.glitch.
	2021-09-23 14:37:59.619	367.54432	24.125.233.208	1.1	intro-to-o11y-nodejs.glitch.
	2021-09-23 14:37:59.307	268.29466	24.125.233.208	1.1	intro-to-o11y-nodejs.glitch.

**FIELDS**

All (30)

- Timestamp
- duration\_ms
- http.client\_ip
- http.flavor
- http.host
- http.method
- http.route
- http.status\_code
- http.status\_text
- http.target
- http.url
- http.user\_agent
- library.name
- library.version
- name
- net.host.ip
- net.host.name
- net.host.port
- net.peer.ip
- net.peer.port
- net.transport



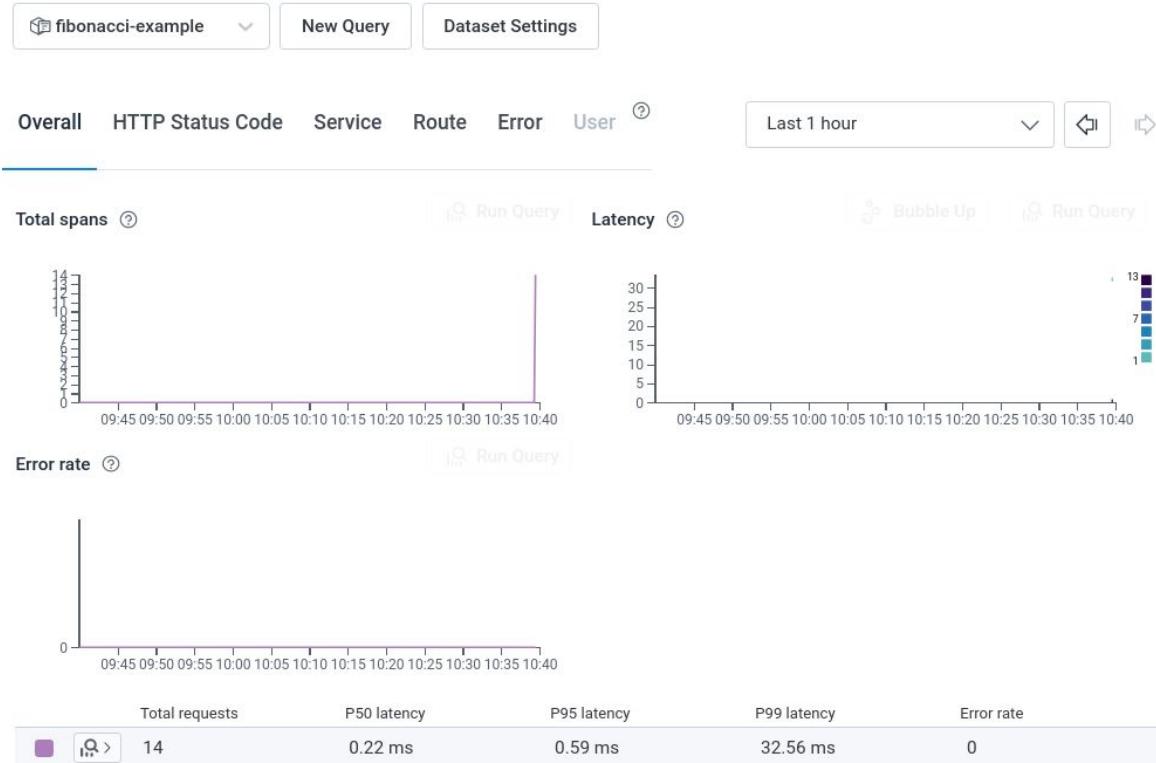
# Hello data

---

Click the house to get back to Home

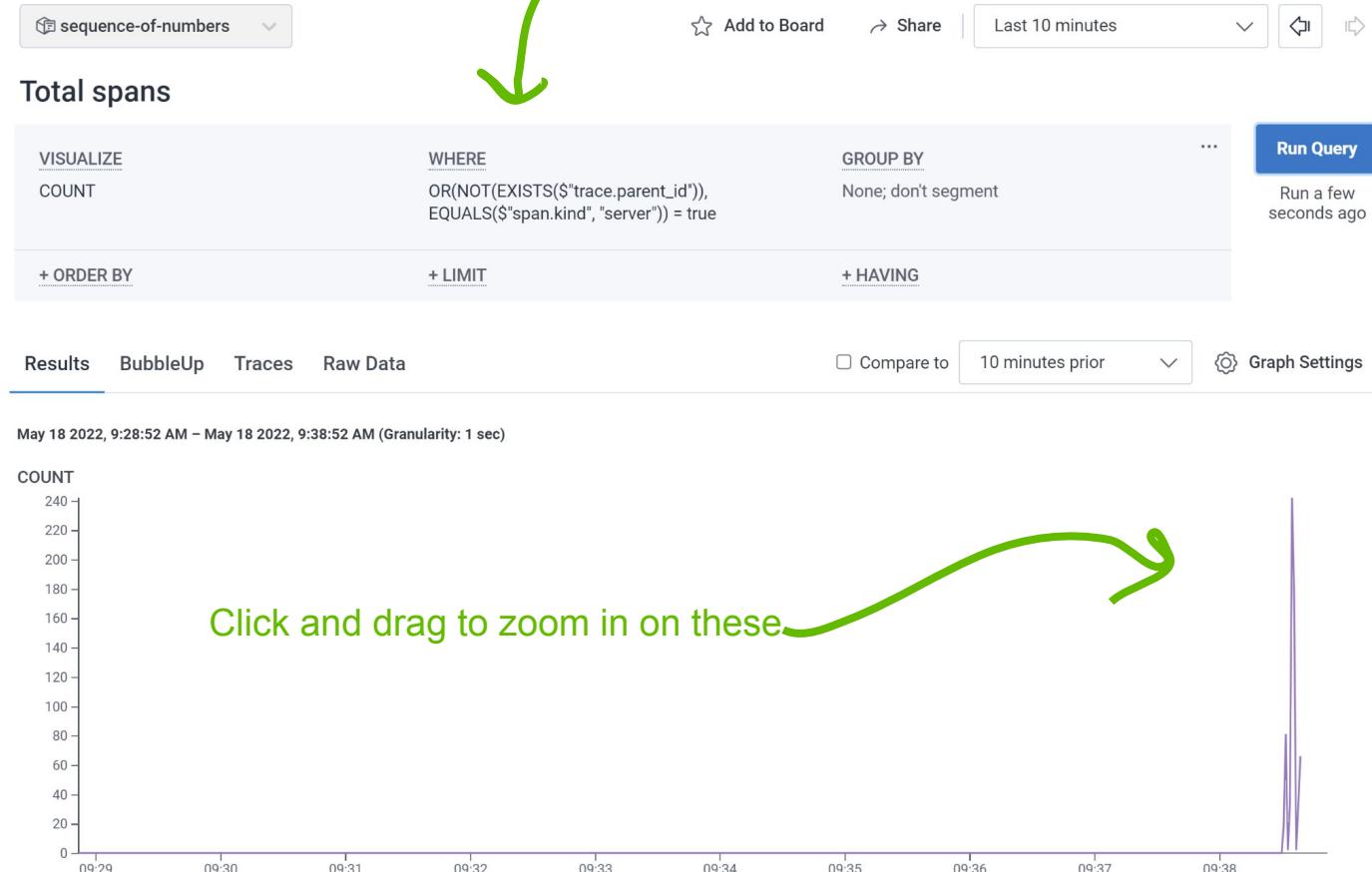
Click the first graph to zoom into it.

## Home



# Interactive Querying

Your service



# Total spans

Run Query

[Clear](#) | [Cancel](#)

VISUALIZE

[X](#) COUNT

WHERE

[X](#) OR(NOT(EXISTS(\$"trace.parent\_id")), EQUALS(...

AND

GROUP BY

...

+ ORDER BY

+ LIMIT

+ HAVING

Results

BubbleUp

Traces

Raw Data

Compare to

42 seconds prior

▼

 Graph Settings

May 18 2022, 9:38:10 AM – May 18 2022, 9:38:52 AM (Granularity: 100 ms)

Click into the  
“where” clause  
to change it

Click the X to  
remove the  
filter

COUNT

90  
80  
70  
60  
50  
40  
30  
20  
10  
0

:10

:15

:20

:25

:30

:35

:40

:45

:50

09:38:32

COUNT

653



© 2022 Hound Technology, Inc. All rights reserved.

# Find only the root spans

The screenshot shows a search interface with a light gray background. At the top left is the word "WHERE" in bold capital letters. To its right is the text "AND ▾" followed by an open parenthesis. Below "WHERE" is a horizontal dotted line. A blue rectangular box encloses the text "trace.parent\_id does-not-exist". This box has a thin blue border and is positioned above another box. Below the first box is a blue box containing the same text "trace.parent\_id does-not-exist". At the bottom left of the interface is the text "+ LIMIT" followed by a horizontal dotted line.

WHERE

AND ▾ (

trace.parent\_id does-not-exist

trace.parent\_id does-not-exist

+ LIMIT

VISUALIZE

COUNT

WHERE

trace.parent\_id does-not-exist

GROUP BY

None; don't segment

...

Run Query

+ ORDER BY

+ LIMIT

+ HAVING

Run a few seconds ago

Results

BubbleUp

Traces

Raw Data

Compare to

42 seconds prior



Graph Settings

May 18 2022, 9:38:10 AM – May 18 2022, 9:38:52 AM (Granularity: 100 ms)

COUNT

1.0  
0.9  
0.8  
0.7  
0.6  
0.5  
0.4  
0.3  
0.2  
0.1  
0



Push to run the new query

COUNT

27



© 2022

# Add more visualizations!

✓ Define aggregate function(s) to visualize events

VISUALIZE

x COUNT  
avg(duration\_ms)

AVG(duration\_ms)  
RATE\_AVG(duration\_ms)

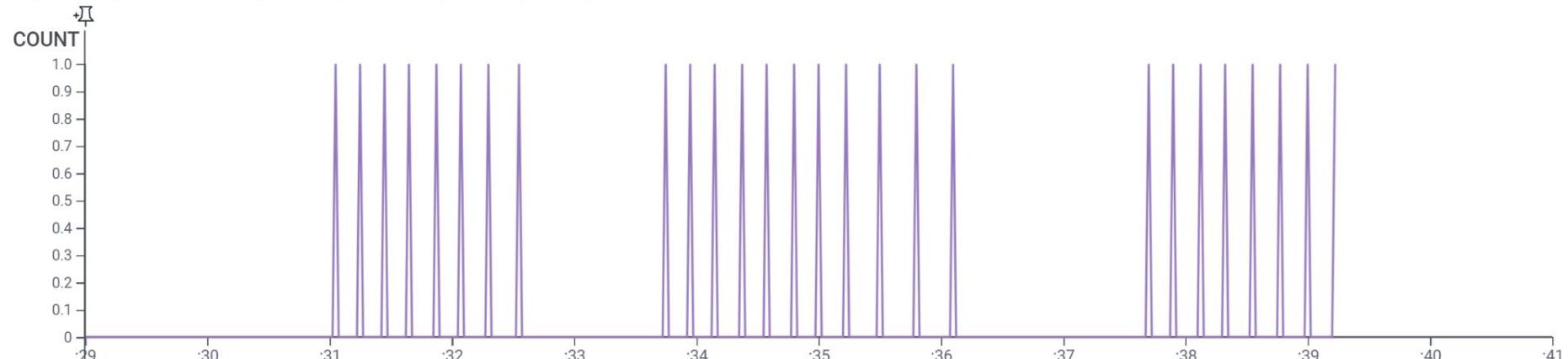
WHERE

x trace.parent\_id

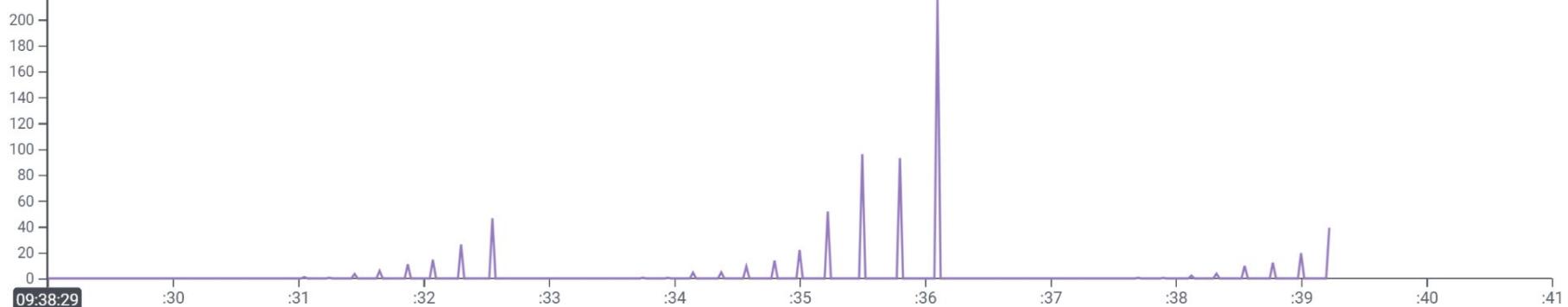
+ LIMIT

The screenshot shows a user interface for defining visualizations. At the top, a dark callout box contains the text "Define aggregate function(s) to visualize events" with a checkmark icon. Below this, the "VISUALIZE" section is active, indicated by a blue border around its input field. The input field contains the text "x COUNT" and "avg(duration\_ms)". A dropdown menu is open over this field, showing two options: "AVG(duration\_ms)" and "RATE\_AVG(duration\_ms)", with "AVG(duration\_ms)" highlighted in blue. To the right of the VISUALIZE section is the "WHERE" section, which contains the text "x trace.parent\_id". Below the WHERE section is a "+ LIMIT" button.

May 18 2022, 9:38:29 AM – May 18 2022, 9:38:41 AM (Granularity: 25 ms)



AVG(duration\_ms)



# Now add a heatmap

Under VISUALIZE, add

HEATMAP(duration\_ms)

then push Run Query.

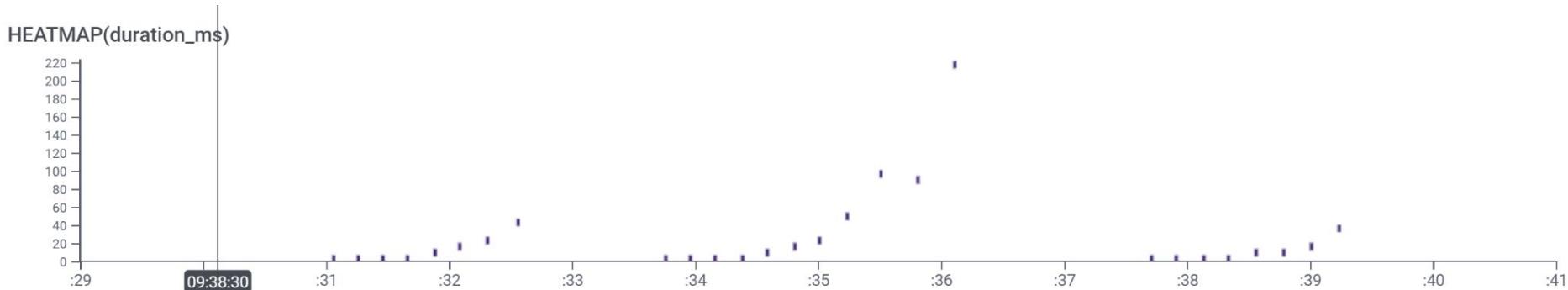
It doesn't look like much...

VISUALIZE

COUNT

AVG(duration\_ms)

HEATMAP(duration\_ms)



# Generating a little more data...

---

In your app, select **Go!** ... and then make it **Stop** after 5-7 numbers several times.

**A sequence of numbers:**

Go

0, 1, 1, 2, 3, 5, 8, 13, ●

Stop



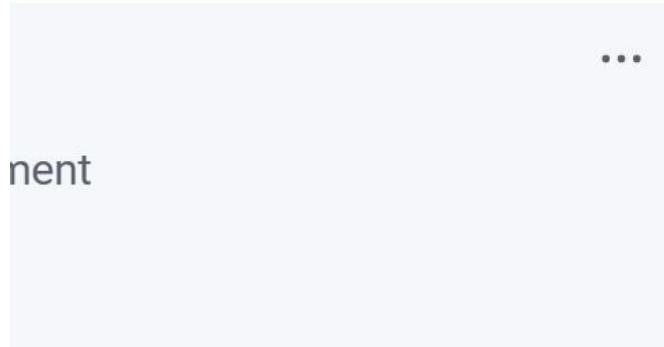
# Run the query again

Change the timeframe back to  
“Last 10 minutes”

Last 10 minutes



Then zoom in again on the part of  
the timeline with data.

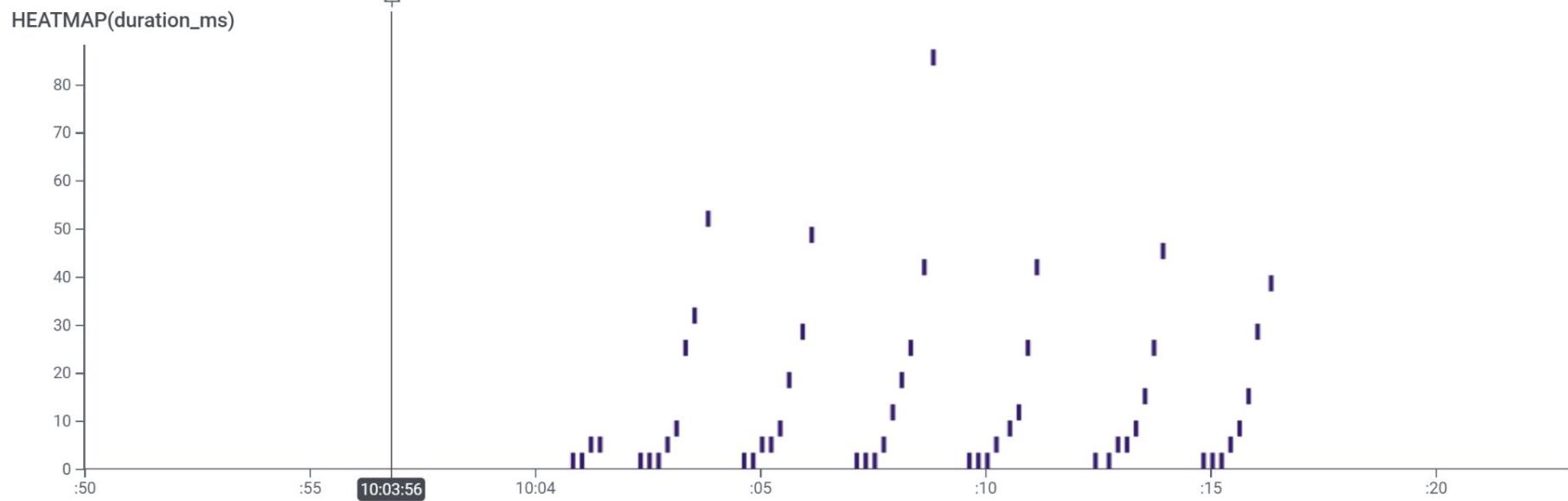


**Run Query**

Run a few  
seconds ago



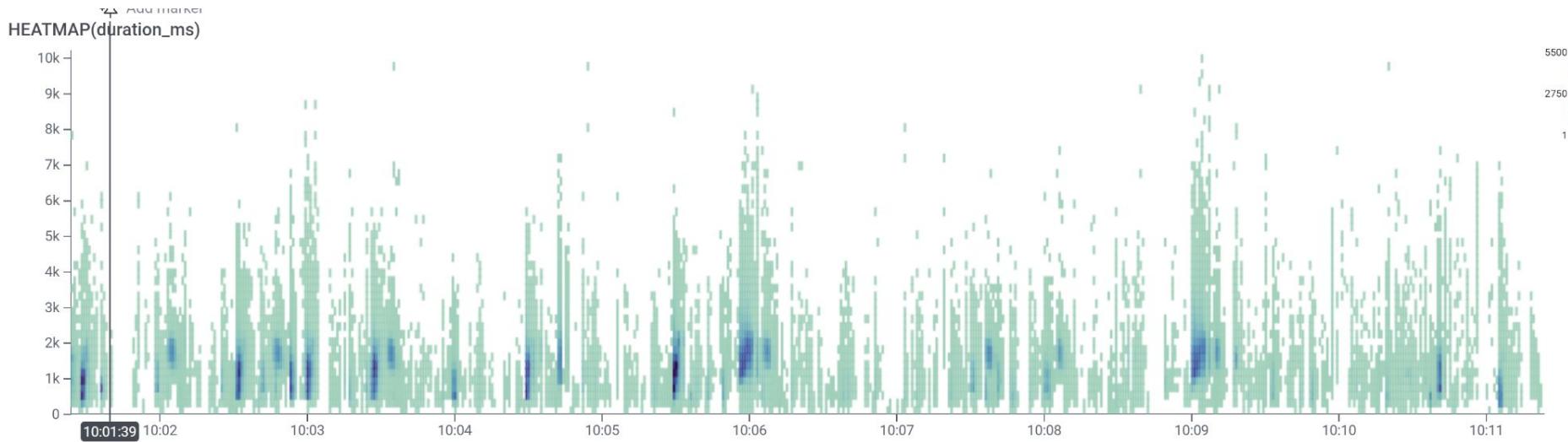
# The higher the dot, the slower the request.



# Heatmaps

Each area on the heatmap corresponds to a time of day, and a slowness of requests. (higher is slower)

The histogram's color shows the density in that area: how many requests took this long, at this time. (darker is more of them)



# Accessing BubbleUp

Go to the **BubbleUp** tab and draw a box around a group of slow requests!

Honeycomb finds fields whose values are very different between **your selected events** and all the rest

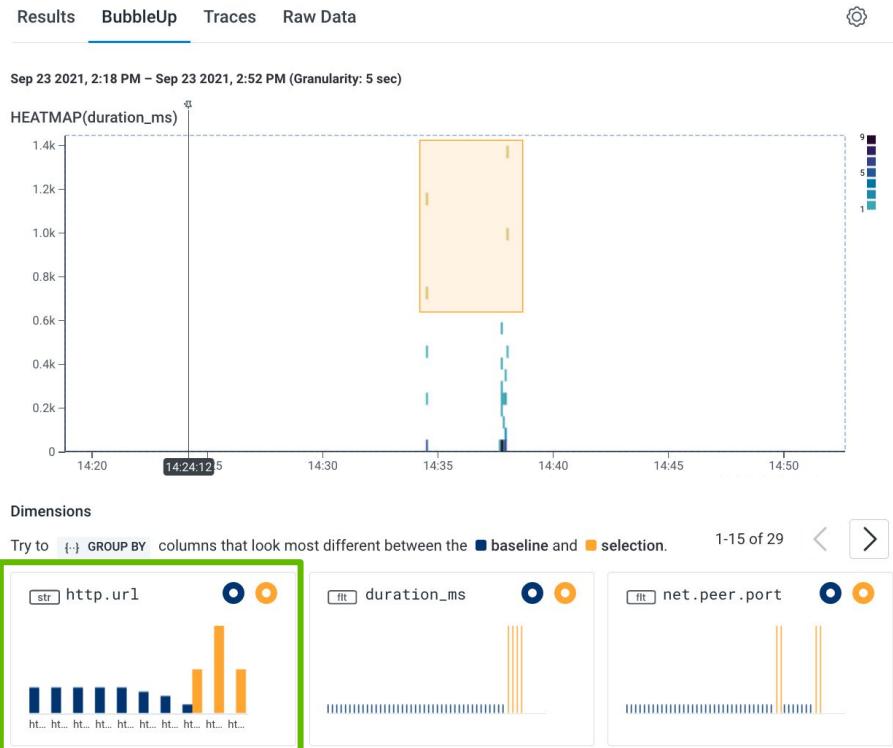
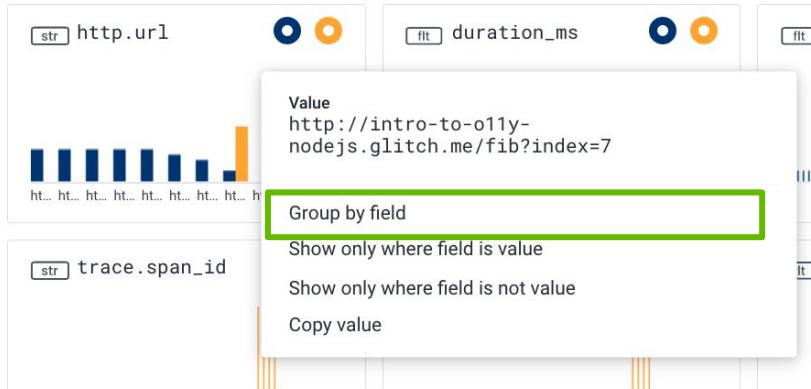
The yellow bars reflect fields and values disproportionately represented inside the area you drew.

The blue bars reflect the rest of the dataset.



# Drilling down with BubbleUp

Click on the field that makes a difference  
(`http.url` or `http.target`).  
Select **Group by field** in the menu.

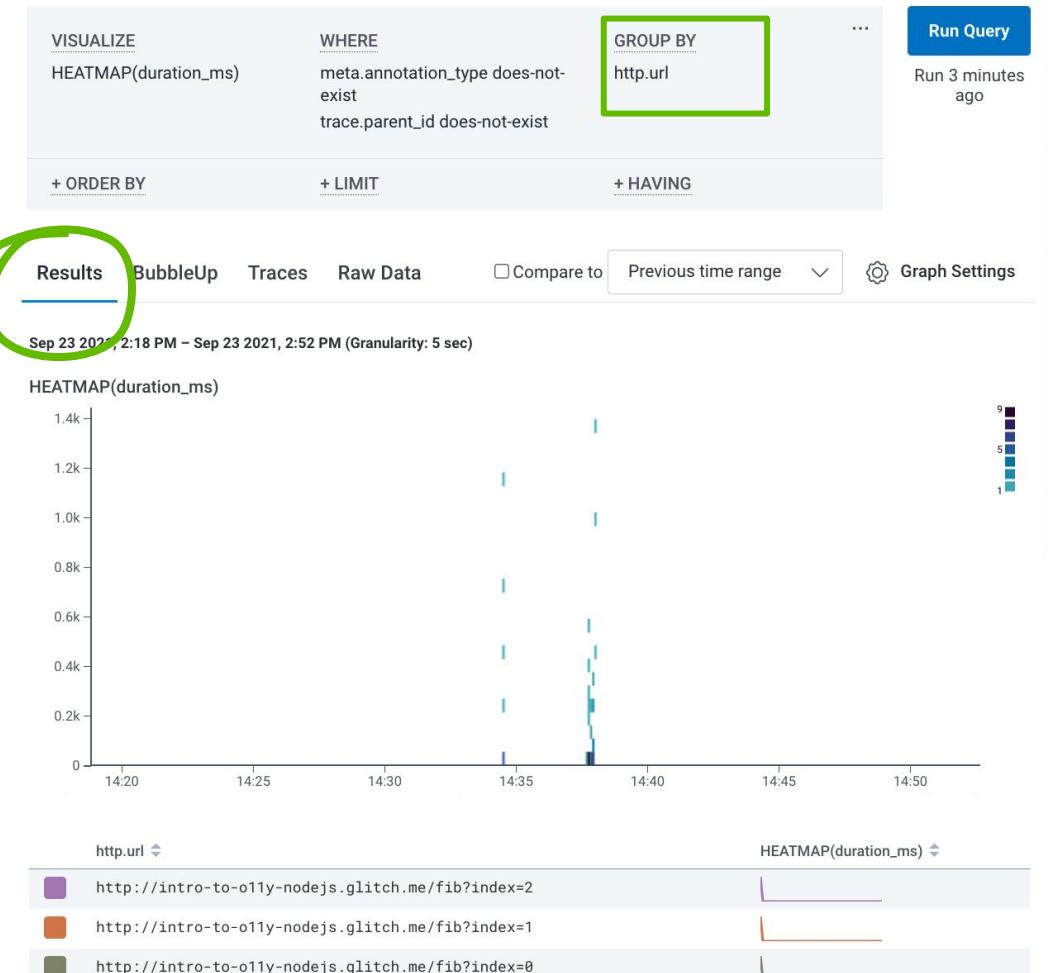
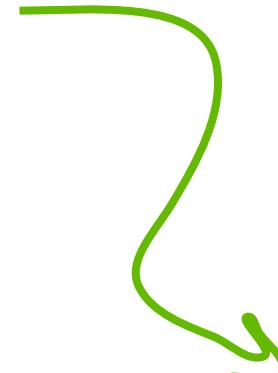


# What did that do?

It doesn't look much different.

But the query has been customized...

And if you select the **Results** tab,  
you will see the table below the graph.



# GROUP BY and HEATMAP

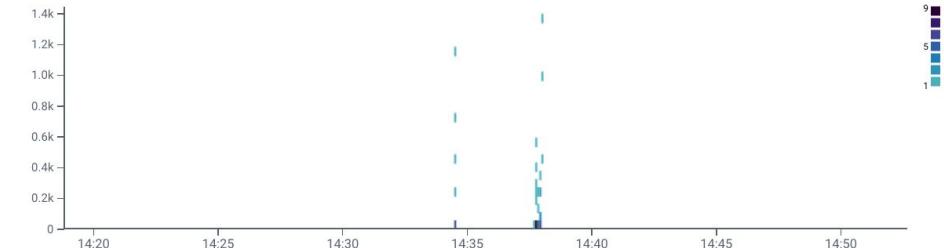
When in the **Results** tab...

You can hover over a group of results to highlight just those entries in the heatmap.

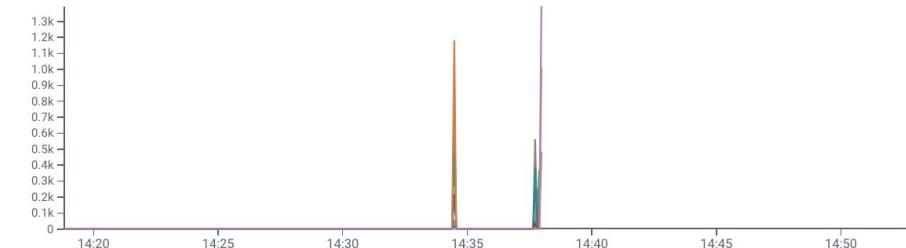
Results   BubbleUp   Traces   Raw Data    Compare to   Previous time range       Graph Settings

Sep 23 2021, 2:18 PM – Sep 23 2021, 2:52 PM (Granularity: 5 sec)

HEATMAP(duration\_ms)



P99(duration\_ms)



http.url	HEATMAP(duration_ms)	P99(duration_ms)
http://intro-to-o11y-nodejs.glitch.me/fib?index=9		1,391.0551
http://intro-to-o11y-nodejs.glitch.me/fib?index=8		1,176.9633
http://intro-to-o11y-nodejs.glitch.me/fib?index=7		728.68045
http://intro-to-o11y-nodejs.glitch.me/fib?index=6		470.62451



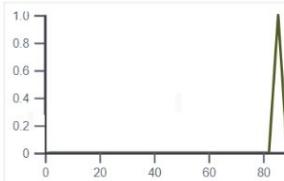
# Click on the heatmap to see a trace

HEATMAP(duration\_ms)

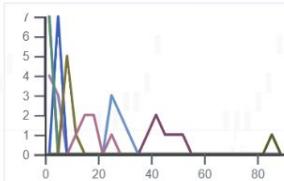


HEATMAP(duration\_ms)

This time bucket



Entire time range



# See an example!

← Trace d6ba7420b6bbd9b8e094dff4784e07d7 at 2022-05-18 10:04:08

Rerun

Search spans



name service.name

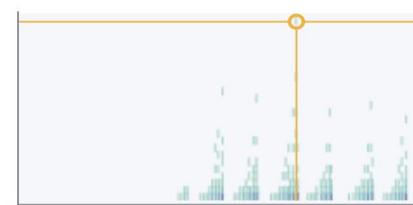
0 spans with errors

Fields

		0s	0.02s	0.04s	0.06s	0.0848s
...	5 GET /fib	sequence-of-numbers	84.80ms			
..	middleware - query	sequence-of-numbers	38.4µs			
..	middleware - expressInit	sequence-of-numbers	23.6µs			
..	request handler - /fib	sequence-of-numbers	2.82µs			
1	HTTP GET	sequence-of-numbers	59.75ms			
5	GET /fib	sequence-of-numbers	59.06ms			
..	middleware - query	sequence-of-numbers	23.3µs			
..	middleware - expressInit	sequence-of-numbers	17.4µs			
..	request handler - /fib	sequence-of-numbers	2.05µs			
1	HTTP GET	sequence-of-numbers	43.70ms			
5	GET /fib	sequence-of-numbers	42.48ms			
..	middleware - query	sequence-of-numbers	22.0µs			

sequence-of-numbers >  
GET /fib

Distribution of span duration



Fields

Filter fields and values in span

Timestamp  
2022-05-18T15:04:08.873779968Z

duration\_ms  
84.803072

http.flavor  
1.1

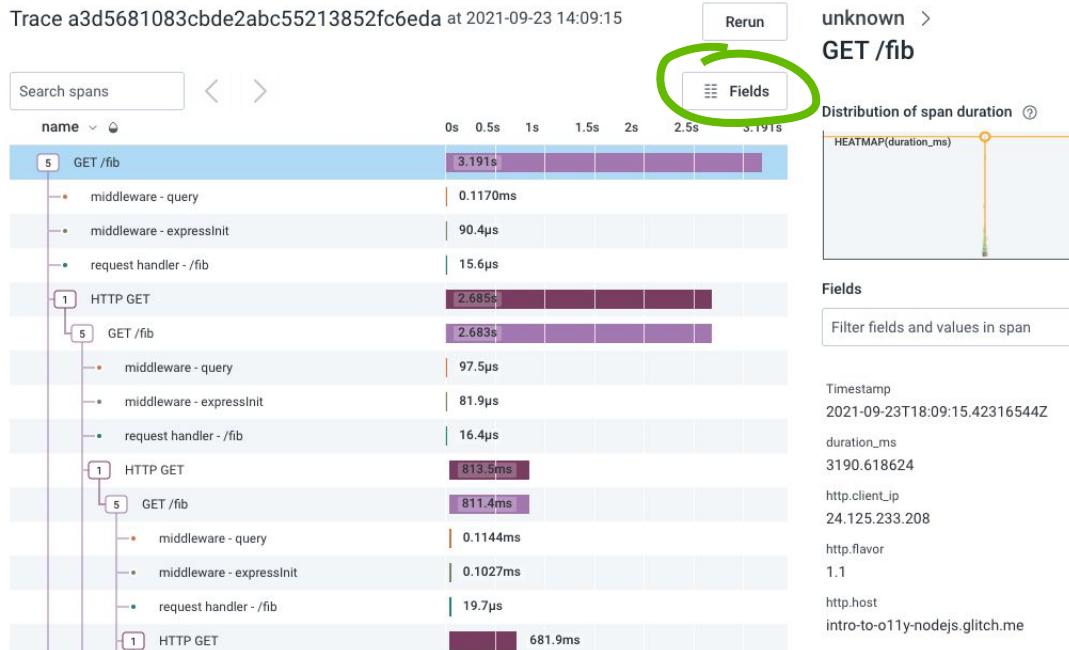
http host

# Examine a trace

To customize:

Select the **Fields** button in the upper right corner and enter **http.target**.

**NOTE:** For Python and Go implementations, use **http.target**  
For others, use **http.url**

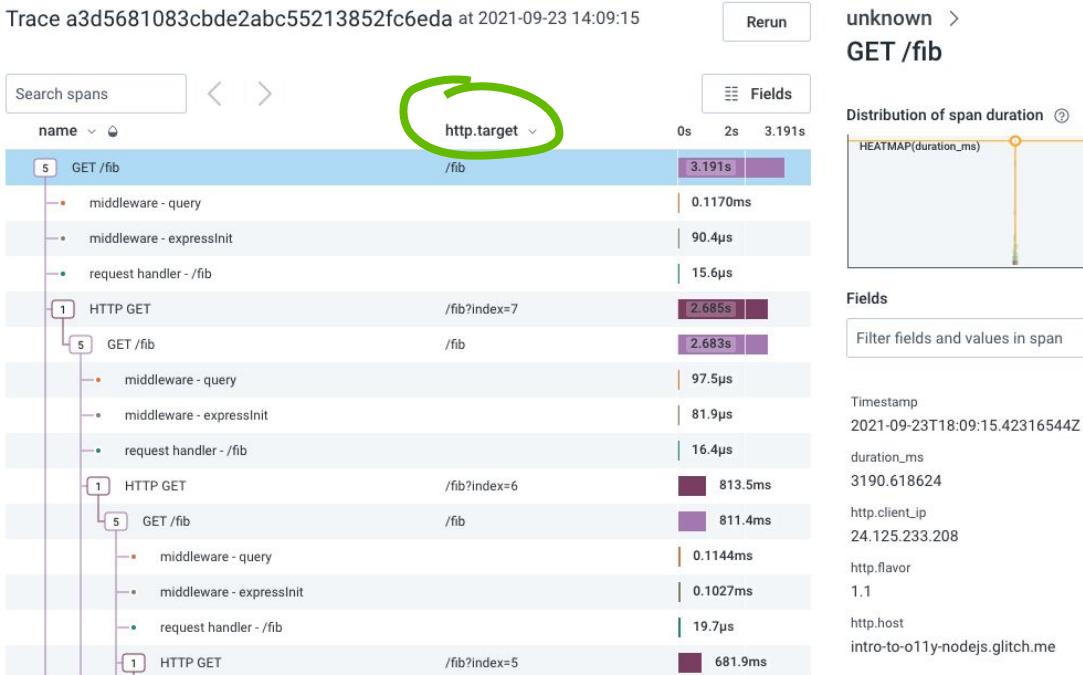


# Examine a trace

To customize:

Select the **Fields** button in the upper right corner and enter **http.target** (for Python and Go)

The display will update to show a **http.target** column.



# Search a trace

Search for spans that contain a phrase anywhere in their fields

The screenshot shows a trace visualization interface. At the top, there is a search bar with the placeholder "index=" and a green oval highlighting it. To the right of the search bar are two arrows pointing left and right, and the text "Selected: 1 of 133 spans found." Below the search bar is a dropdown menu set to "name". To the right of the search bar is a dropdown menu set to "http.url". The main area displays a list of spans. The first span is highlighted with a yellow background and has a purple border around its number (5). It is labeled "GET /fib" and "http://localhost:3000/fib?index=8". This span has three child spans: "middleware - query" (orange dot), "middleware - expressInit" (blue dot), and "request handler - /fib" (green dot). Below this is another span with a yellow background and a purple border (number 1) labeled "HTTP GET" and "http://127.0.0.1:3000/fib?index=7". It has five child spans: "GET /fib" (purple box), "middleware - query" (orange dot), "middleware - expressInit" (blue dot), and "request handler - /fib" (green dot). Further down is another span with a yellow background and a purple border (number 1) labeled "HTTP GET" and "http://127.0.0.1:3000/fib?index=6". It has five child spans: "GET /fib" (purple box) and "middleware - query" (orange dot). The right side of the interface shows numerical values (8, 3, 2, 2, 5, 5, 2, 1, 2, 4, 4, 2) which likely represent span IDs or sequence numbers.

Span ID	Span Type	Parent Span ID	URL
5	GET /fib		http://localhost:3000/fib?index=8
1	HTTP GET		http://127.0.0.1:3000/fib?index=7
5	GET /fib	1	http://127.0.0.1:3000/fib?index=7
1	HTTP GET		http://127.0.0.1:3000/fib?index=6
5	GET /fib	1	http://127.0.0.1:3000/fib?index=6



# Search a span

On the right, search all the fields in the selected span.

## Fields

/fib

str http.url

http://localhost:3000/fib?index=8

str http.route

/fib

str http.target

/fib

str name

GET /fib



# Query by a value

After searching for a field (say, trace.trace\_id)

Click the three dots next to the value to get a handy menu

“Show only where field is value” gets you to a query

## Fields

trace

[str] trace.span\_id

143911d7ff0976dd

[str] trace.trace\_id

d6ba7...

Group by field

Show only where field is value

Show only where field is not value

Copy field name

Copy value



Run Q

Run a second

VISUALIZE

COUNT

HEATMAP(duration\_ms)

WHERE

```
service.name = sequence-of-numbers
trace.trace_id =
d6ba7420b6bb9b8e094dff4784e07d7
```

GROUP BY

None; don't segment

...

ORDER BY

COUNT desc

LIMIT

None

HAVING

None; include all results

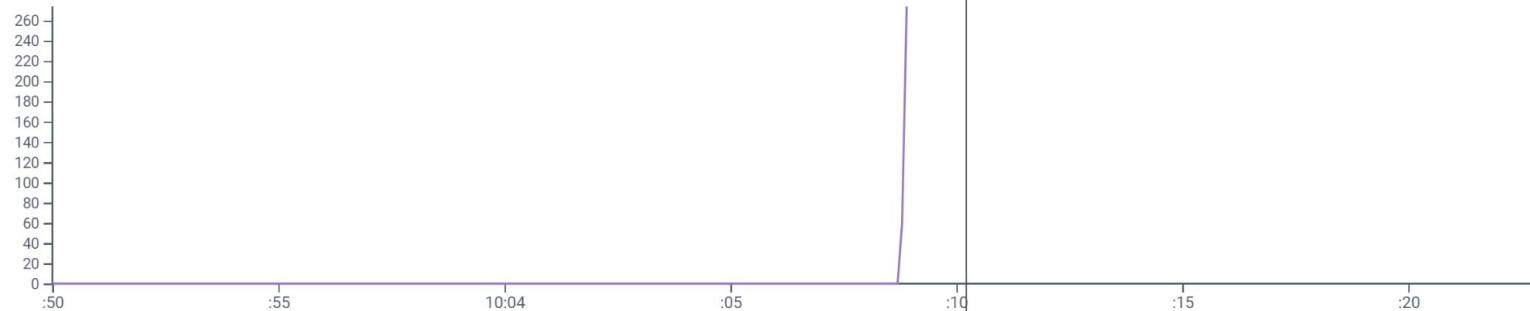
**Results****BubbleUp****Traces****Raw Data** Compare to

33 seconds prior



Graph Set

May 18 2022, 10:03:50 AM – May 18 2022, 10:04:23 AM (Granularity: 100 ms)

**COUNT****HEATMAP(duration\_ms)**

# Try customizing with GROUP BY

Add “name” to the GROUP BY then Run Query

VISUALIZE

WHERE AND ↴ GROUP BY ...

COUNT  HEATMAP(duration\_ms)

service.name = sequence-of-numbers  
 trace.trace\_id = d6ba7420b6bb9b8e094dff47...

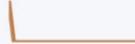
name

Run Query

Clear | Cancel



Then scroll down to the table of results

name	COUNT	HEATMAP(duration_ms)
middleware - expressInit	67	
middleware - query	67	
GET /fib	67	
request handler - /fib	67	
HTTP GET	66	

# Add visualizations to see more results

In VISUALIZE, add:

P99(duration\_ms)

Then add that to the ORDER BY too.

Now see the slowest spans in the result table!

## VISUALIZE

x COUNT x P99(duration\_ms)

## ORDER BY

x P99(duration\_ms) desc

name ▲

GET /fib	67	59.06406
HTTP GET	66	43.69894
middleware - query	67	0.05222
middleware - expressInit	67	0.04173
request handler - /fib	67	0.00333

# Putting it all together: custom queries

VISUALIZE

WHERE

GROUP BY

ORDER BY

LIMIT

HAVING

The screenshot shows a user interface for building custom queries. At the top, there are four main sections: 'VISUALIZE', 'WHERE', 'AND ▾', and 'GROUP BY'. Below each section is a text input field. Under 'VISUALIZE' is 'COUNT, SUM(..., HEATMAP(...'. Under 'WHERE' is 'attribute = value, attribute exists...'. Under 'GROUP BY' is 'attribute(s)'. Below these are three additional buttons: '+ ORDER BY', '+ LIMIT', and '+ HAVING'.

VISUALIZE	WHERE	AND ▾	GROUP BY
COUNT, SUM(..., HEATMAP(...	attribute = value, attribute exists...	AND ▾	attribute(s)

+ ORDER BY      + LIMIT      + HAVING

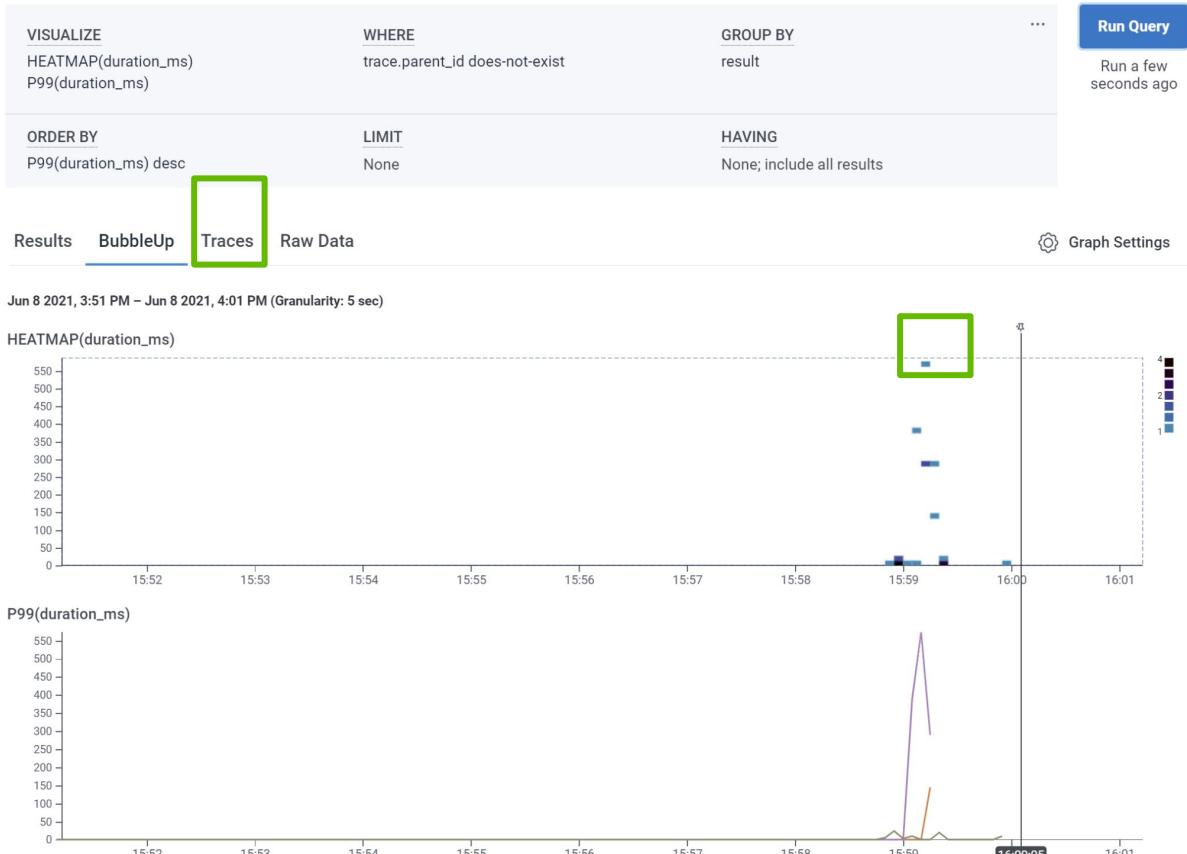


# Find an especially slow trace

Go to the **Traces** tab now.

Click on the slowest trace in the heatmap.

The next screen will be your detailed Trace display.



# Summary

---

In this section, we:

- Examined a trace for a specific request
- Identified outliers using Heatmaps and BubbleUp
- Customized our queries and used multiple visualizations
- Inspected our raw data and found the slowest traces



# Puzzles for the class

---

How many times is `/fib?index=1` *inside* a call to `/fib?index=5`?

How much longer (*P99*) does it take to evaluate `/fib` of 5 compared to of 4?

Why is this so slow as *index* increases?



# Puzzles for the class

---

**How many times is /fib?index=1 inside a call to /fib?index=5?**

Hint: Find an example trace.

Use “WHERE http.url CONTAINS index=5”

Then, choose “Traces” tab.

**How much longer (P99) does it take to evaluate /fib of 5 compared to of 4?**

Hint: Use both

VISUALIZE P99(duration\_ms)

GROUP BY http.url

**Why is this so slow as index increases?**

Don't forget to select the **Run Query** button!

There is **more than one method** to arrive at the answer.

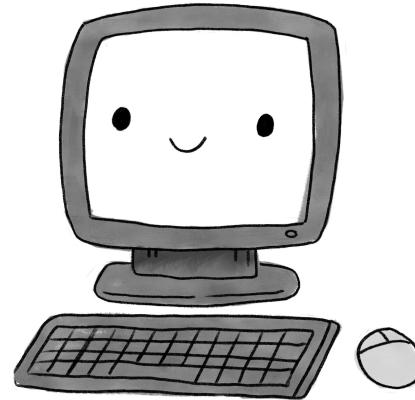


# Answered the puzzles? You're done for now!

---

Come back at top of the hour.

If you're stuck, speak up on Slack or join a help breakout room!



# Puzzles for the class

---

**How many times is /fib?index=1 inside a call to /fib?index=5?**

**Hint:** Find an example trace.

Use “WHERE http.url CONTAINS index=5”  
Then, choose “Traces” tab.

In Query Builder, start with:  
**VISUALIZE COUNT**

**A Solution:**

WHERE trace.parent\_id does-not-exist  
GROUP BY http.url

[For Python and Go apps, use *http.target*]

Focus on */fib?index=5*.

Below the graph, **select** [...] button in  
*http.url* column in the */fib?index=5* row.  
**Choose** "Show only where http.url =  
<http://intro-to-o11y-nodejs.glitch.me/fib?index=5>

**Select any point on graph** to see the Trace Detail View and **count** how many */fib?index=1* you see.



# Puzzles for the class

---

How much longer (**P99**) does it take to evaluate /fib of 5 compared to of 4?

**Hint** - Use Query Builder & add to your query:  
VISUALIZE P99(duration\_ms)  
GROUP BY http.url

[For Python and Go apps, use *http.target* instead.]

## A Solution:

**Return to your query builder display** by selecting the arrow next to the trace name in the upper left corner.

## Add to your query:

VISUALIZE P99(duration\_ms)  
GROUP BY http.url

Two graphs now appear.

In the table of results, sort and compare duration\_ms for /fib of 5 vs. /fib of 4.



# Puzzles for the class

---

**Why is this so slow as index increases?**

**Examine a Trace in Trace Detail view.**

Notice the double recursion and  
no caching present...



# **Where to go from here**

---

# Section Overview

---

In this section, we will:

- Add custom instrumentation
- Explore the dataset schema
- See how querying combined with collaboration tools help to quickly diagnose a problem



# Add more fields and instrumentation

---

Edit or add at least one attribute name and one attribute value. (e.g. "\$firstName was here")

Look for and change the key and value parameters to SetAttributes()



# Check your fields in Honeycomb

---

Look on the right hand side for Dataset Details > Schema to see field names & types, and annotate field descriptions.

Go to Dataset Settings > Schema to coerce types.



# The core instrumentation loop

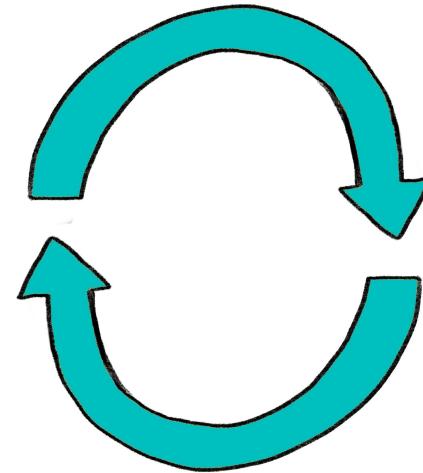
---

Start with auto-instrumentation.

Manually instrument key fields,  
expensive function calls, & mystery  
slowness.

Look at production to verify  
instrumentation correctness.

Repeat!



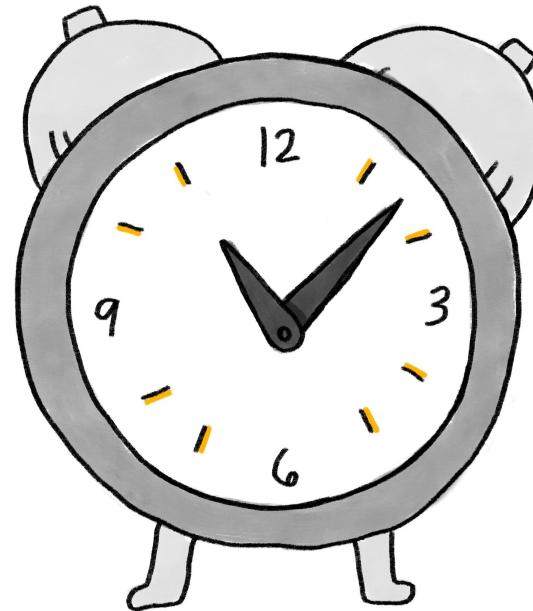
# Using triggers

---

Set up a trigger to send you an email if someone triggers fib with a *parameter* of 5 or higher

(and then test it)

Set up a trigger on your account to send you an email if someone's /fib query takes longer than 10 seconds.



# The core debugging loop

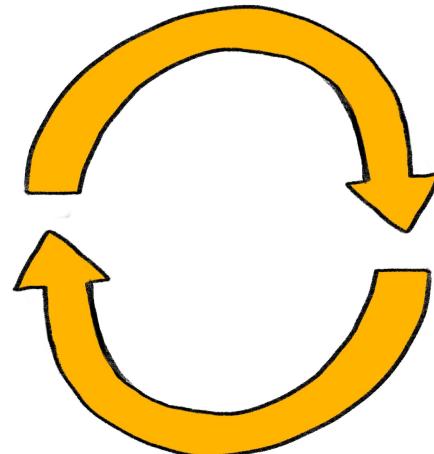
---

<https://ui.honeycomb.io/quickstart/assets/tracing-tour>

Start with a heatmap. Run BubbleUp on outliers to find interesting groups.

Group by / filter by fields (especially instrumented fields) to find the culprit.

Look at an example trace or two.



# Don't panic, you have Honeycomb!

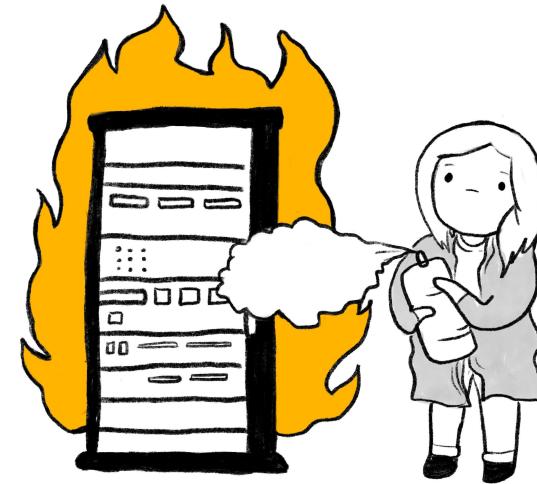
---

Remember remediation comes first.

Markers help you find what changed most recently.

Use bubbleup, group by, and traces to navigate to the problem.

(and Honeycomb SLOs can also help as you get more advanced!)



# Using query history

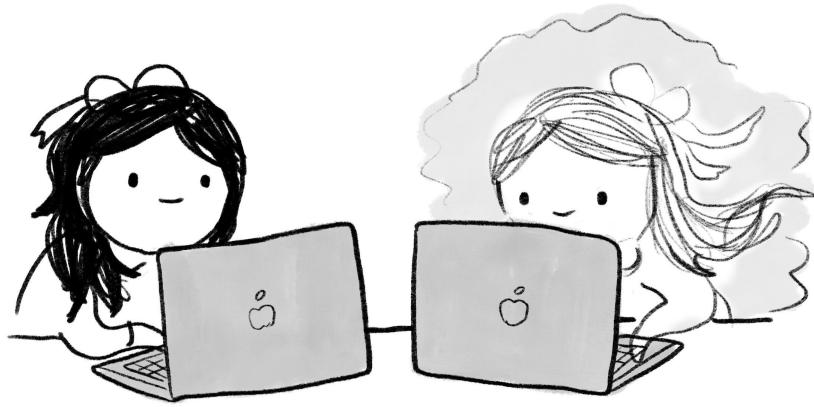
---

Honeycomb is a virtual lab notebook!

Pop open query history on the right.

Pop open saved queries on the left.

Shared with future teammates & colleagues! (and yourself, too)

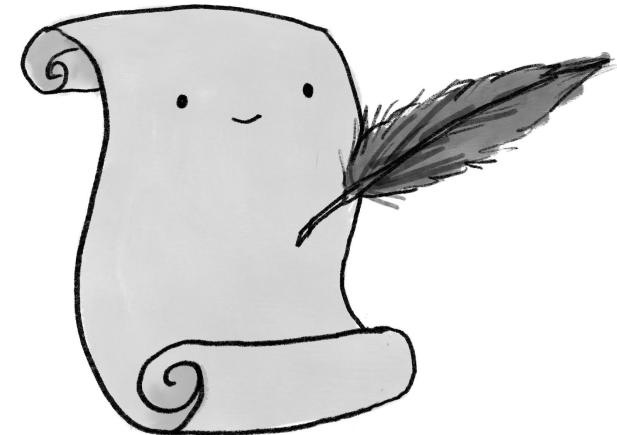


# Summary

---

In this section, we:

- Learned about adding custom instrumentation
- Explored Honeycomb features that keep everyone in the loop, like query history and triggers



# **Share with your colleagues**

---

# Show and tell

---

If you have a coworker here, you can share with them!

Otherwise, add a coworker to your team and show them!



# Bee a hero, make others heroes!

---

Instrumenting with OTel is easy

Sending data to Honeycomb is free!

Querying reveals insights

We're here to help you!

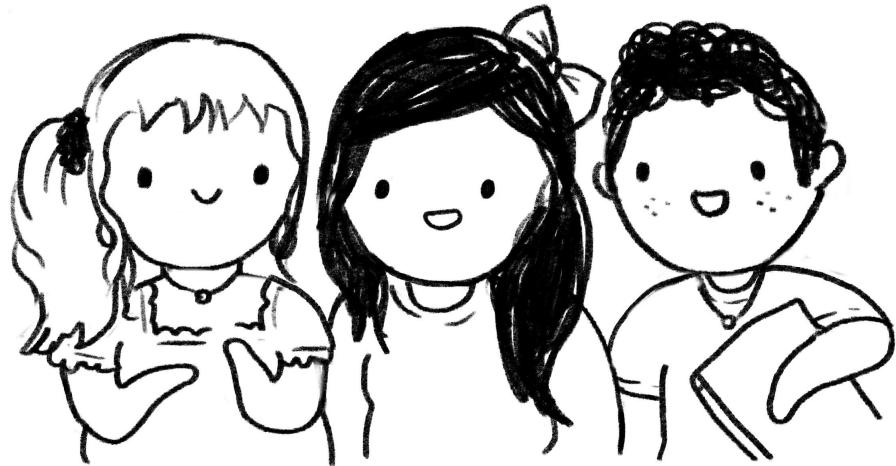
Feel free to add OTel and Honeycomb to  
your own apps!



# Here's what you learned today

---

- Why observability matters
- How to add instrumentation and get telemetry data
- How to use Honeycomb to answer questions and achieve observability
- How to add Honeycomb to your debugging workflow
- How to share these lessons with your team!



# Event Survey + i test in prod t-shirt!

---

We value your feedback and read every comment!

All respondents will receive a t-shirt upon completion. The survey closes tomorrow, June 16 at 12 p.m. PT.

**Secret word:** opentelemetry



Please see the survey link in the Zoom chat window now!





# Thank you

---

We'd love to hear your feedback!

Instructors will be around Slack, give us a 





[www.honeycomb.io](http://www.honeycomb.io)

