OpenTelemetry End-User SIG: Developer Experience Survey

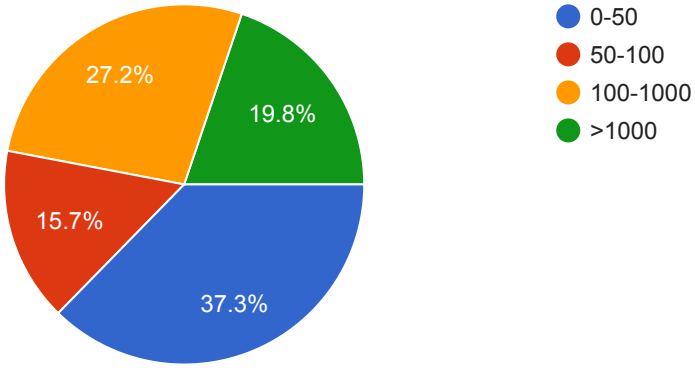
217 responses

Publish analytics

How large is your engineering organization or department?

Copy

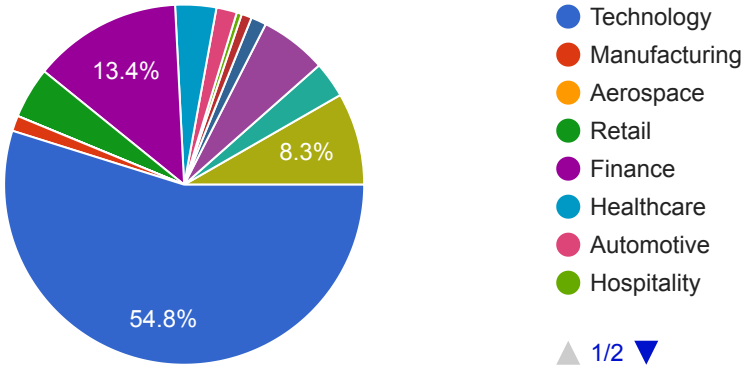
217 responses



What industry do you work in?

Copy

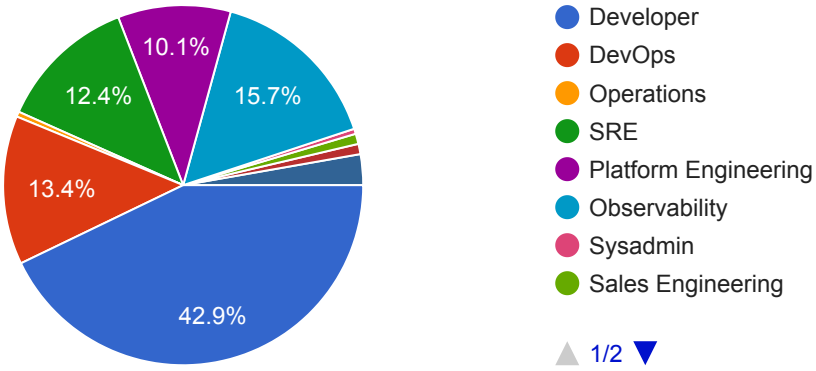
217 responses



What type of team do you work on?

Copy

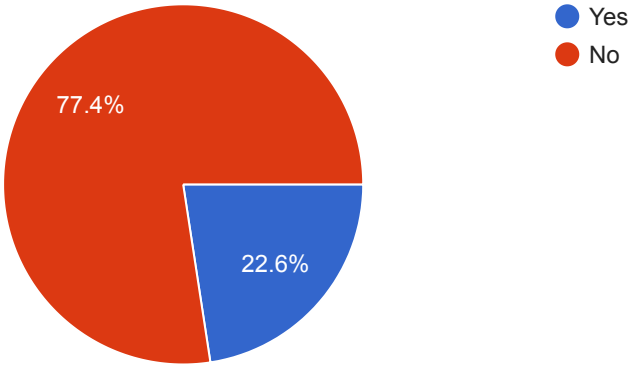
217 responses



Do you work for an Observability or APM (monitoring) vendor?

Copy

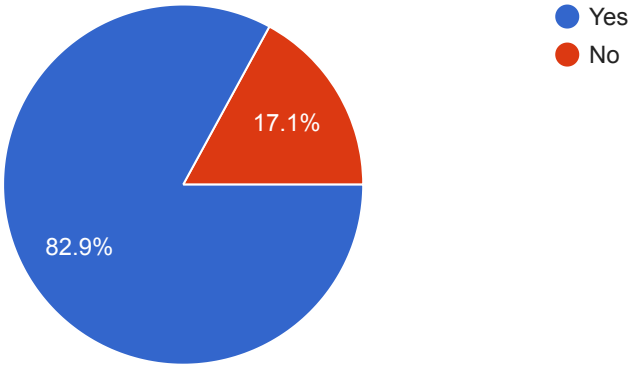
217 responses



Are you running OpenTelemetry in Production?

Copy

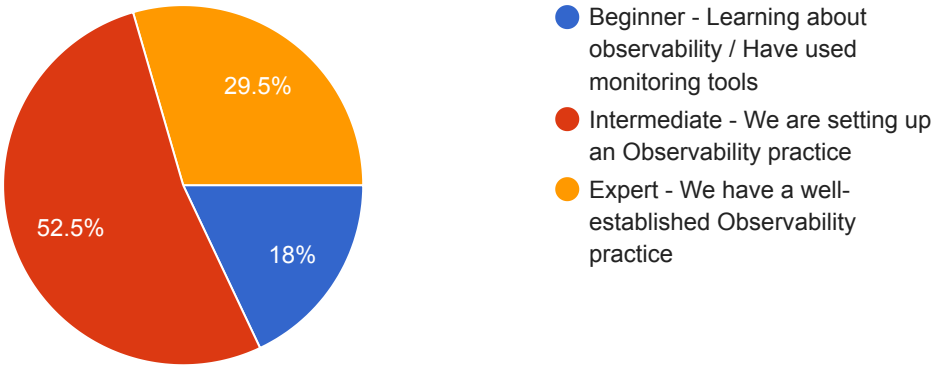
217 responses



Where in your Observability journey is your organization?

Copy

217 responses

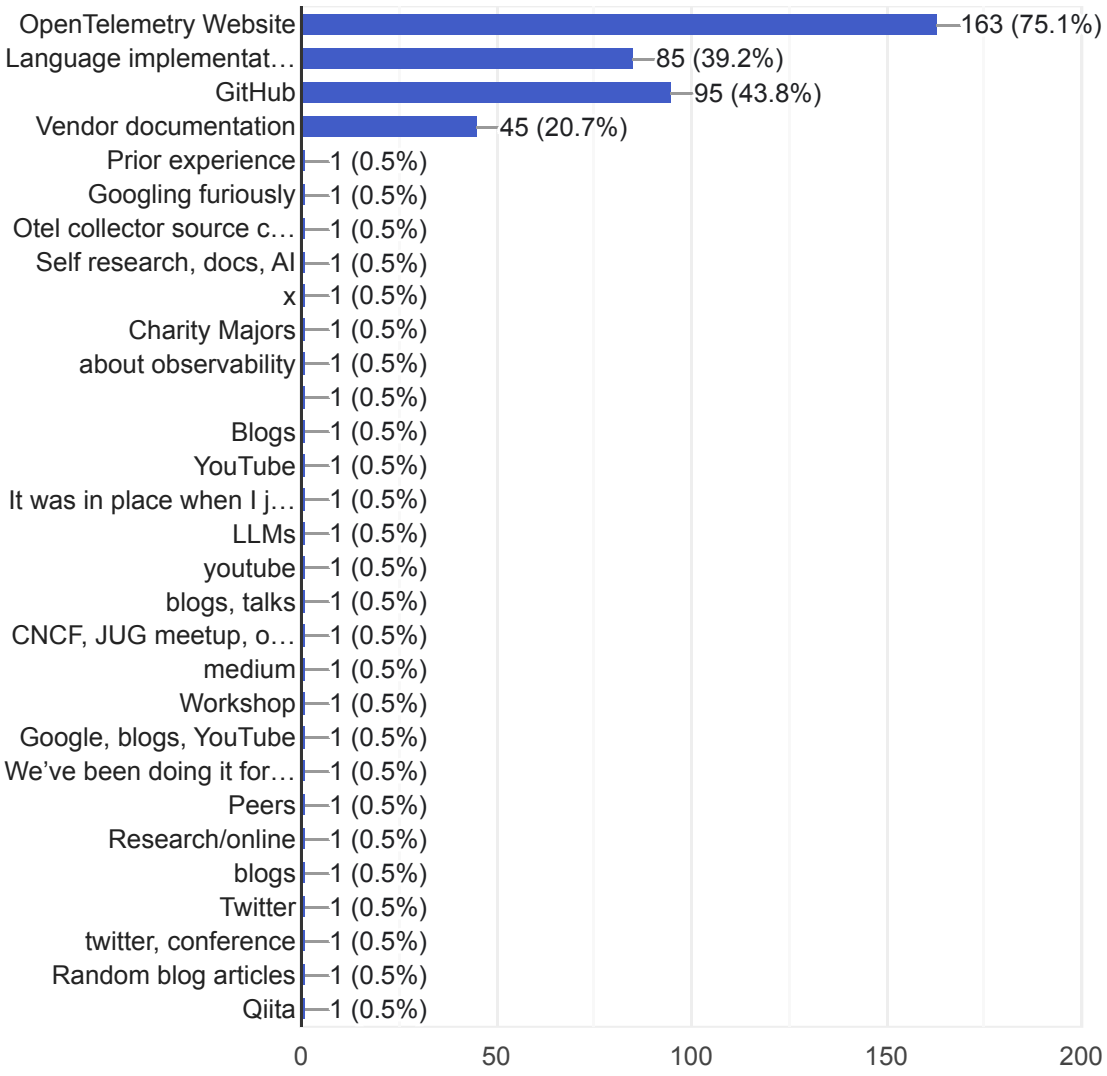


General OTel info

Where did you get your day 1 information on getting started with OpenTelemetry? (Select all that apply)

Copy

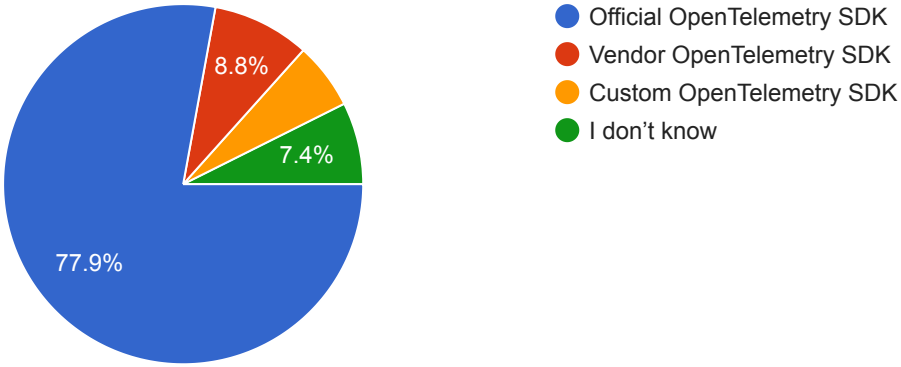
217 responses



Do you use OpenTelemetry SDK directly or via vendor SDK of the languages you use?

Copy

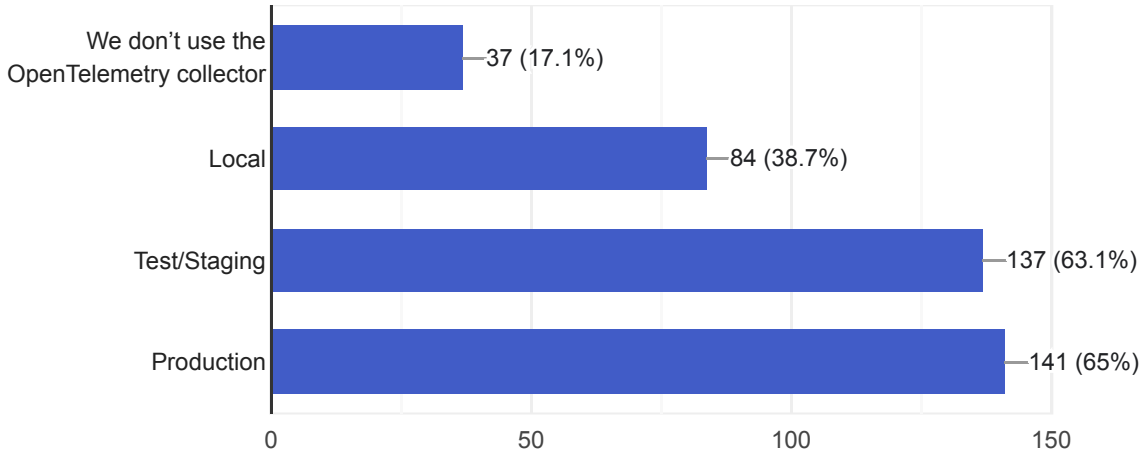
217 responses



What environments are you using the OpenTelemetry collector? (select all that apply)

Copy

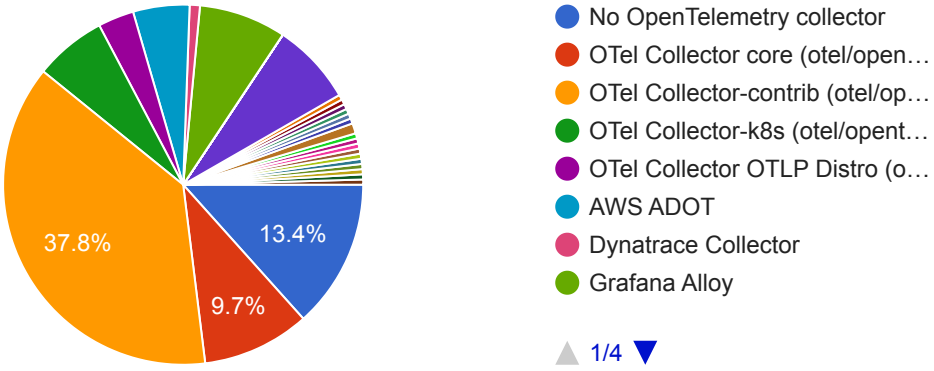
217 responses



Which collector distribution do you use?

Copy

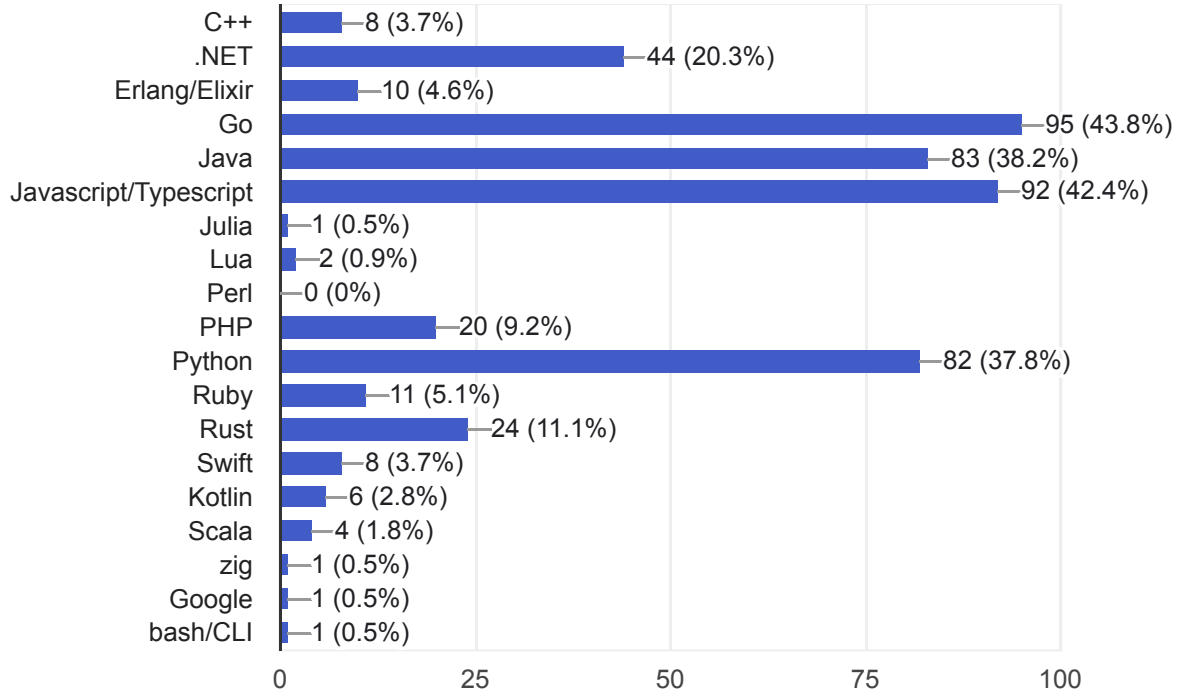
217 responses



What programming languages are you using with OpenTelemtery? (select all that apply)

Copy

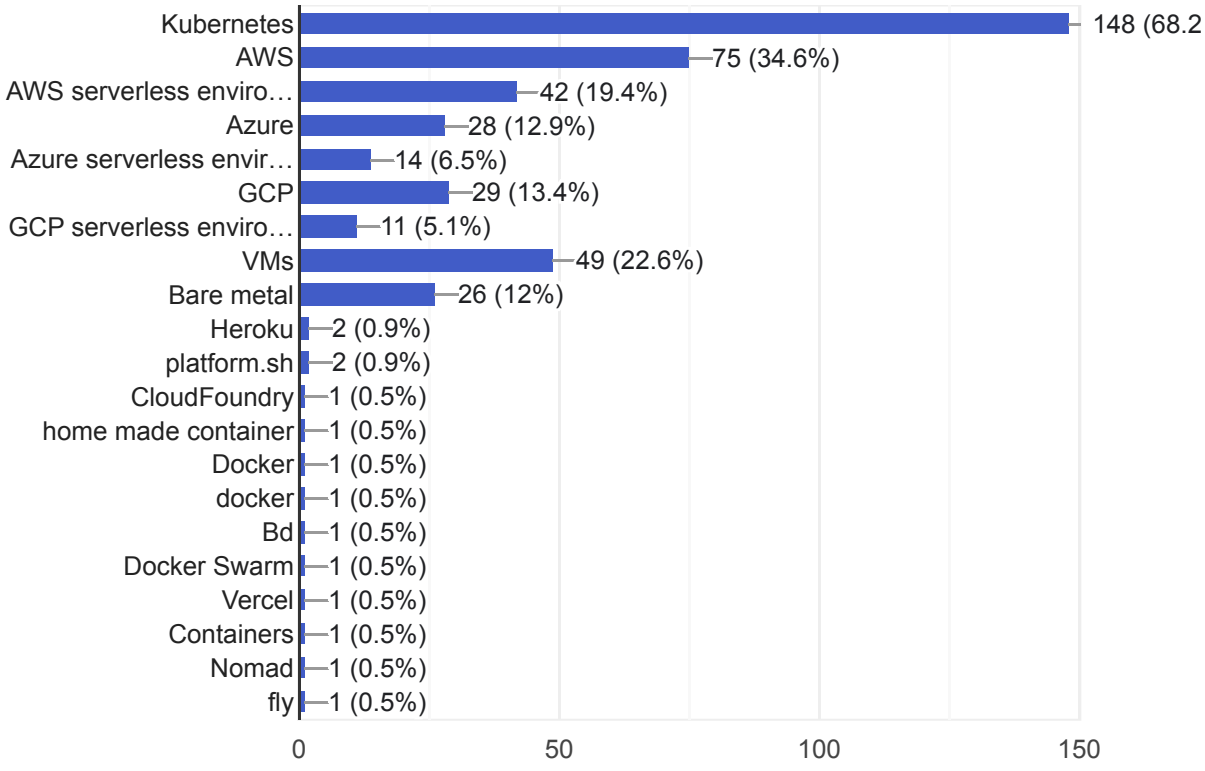
217 responses



Which platforms are you running OpenTelemety on? (select all that apply)

Copy

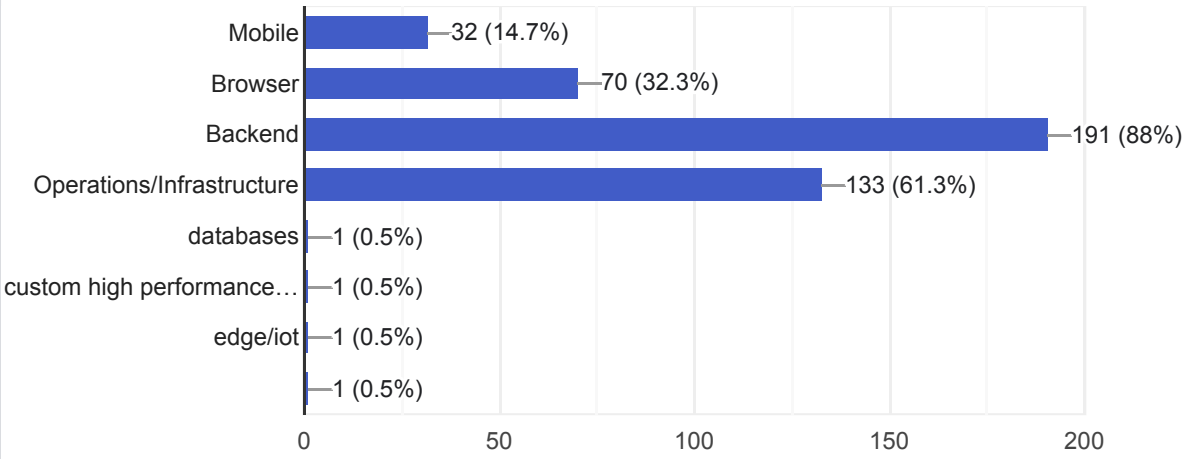
217 responses



What kind of service do you regularly work on? (select all that apply)

Copy

217 responses

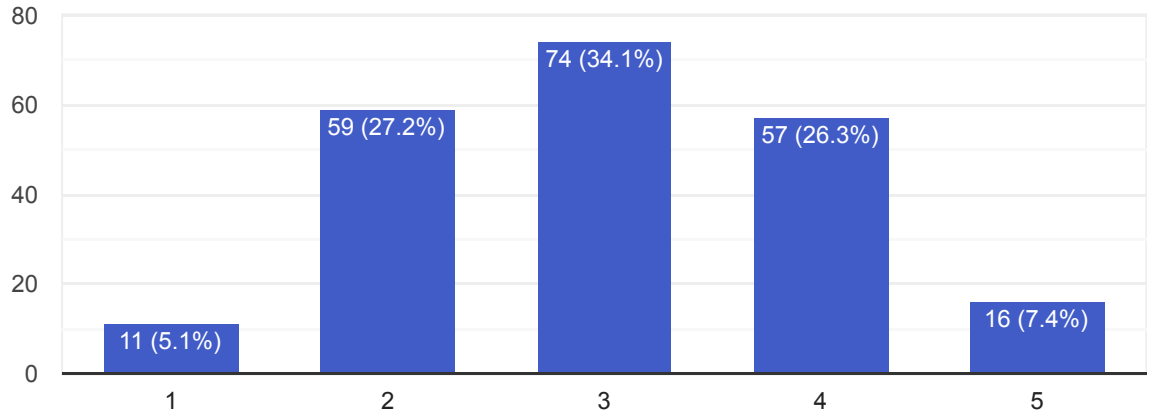


SDK installation

How complex was it to manually instrument your application with OpenTelemetry?

Copy

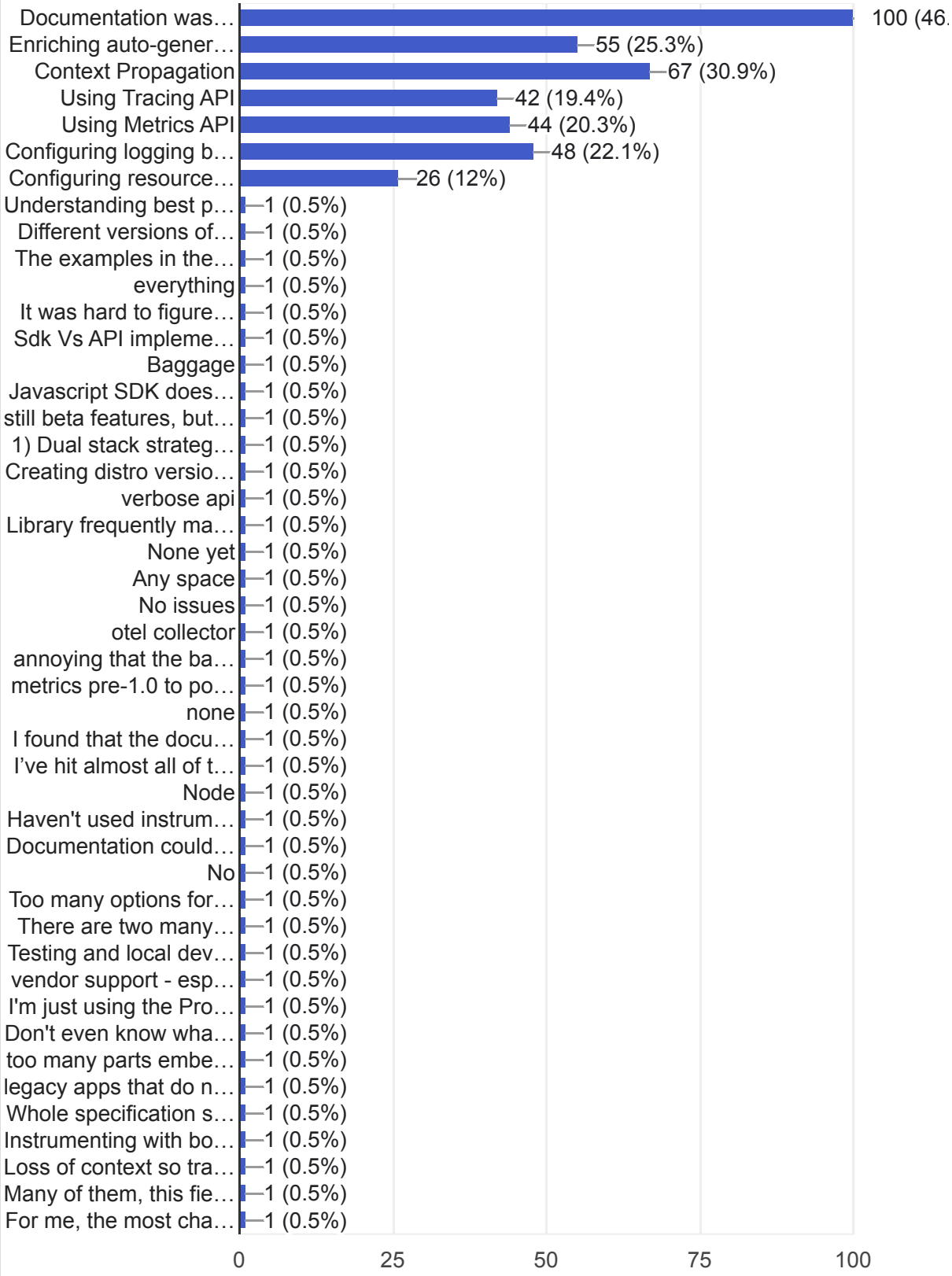
217 responses



Did you encounter any issues when manually instrumenting your application?

Copy

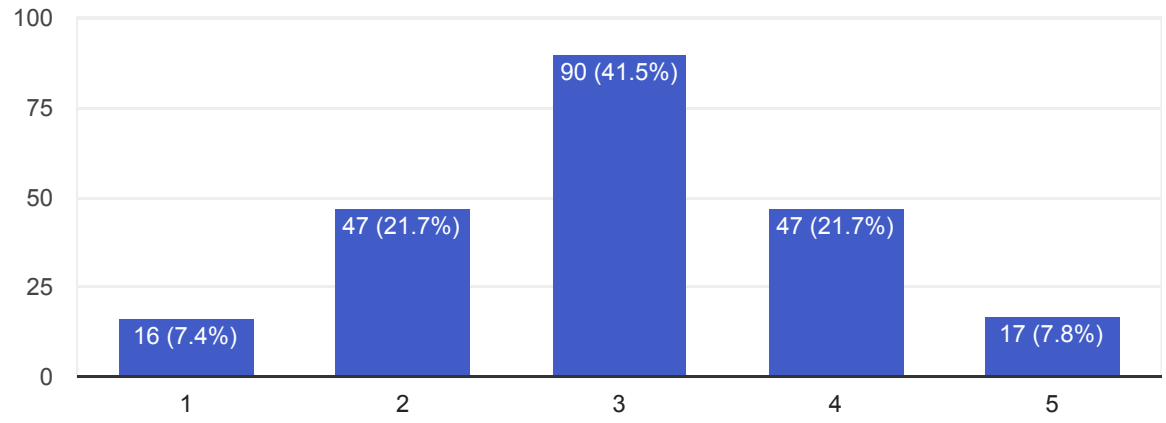
217 responses



How complex was it to get started with OpenTelemetry SDK? (skip if you don't use)

Copy

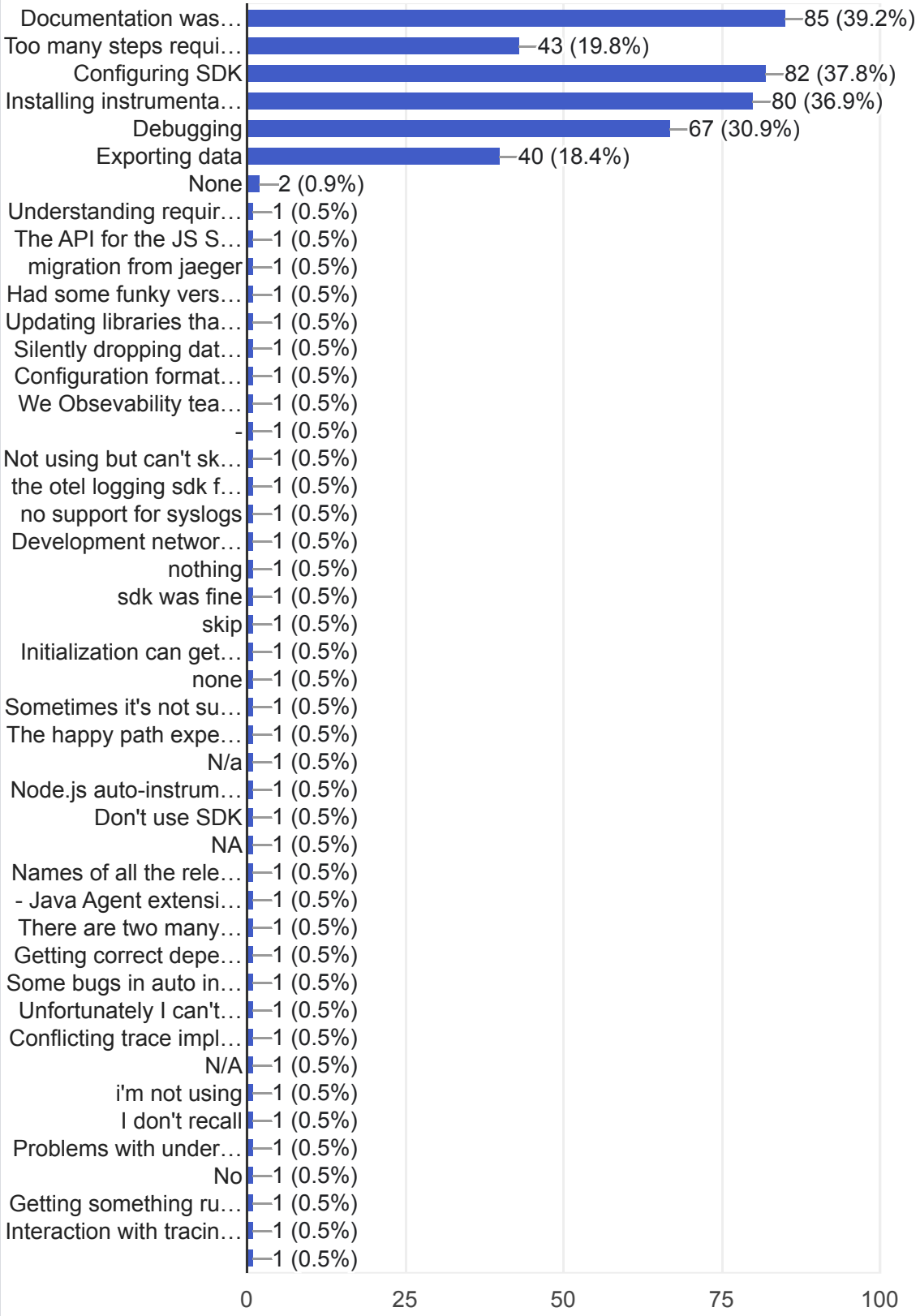
217 responses



Did you encounter any issues when installing OpenTelemetry SDK in your application? (select all that apply)

Copy

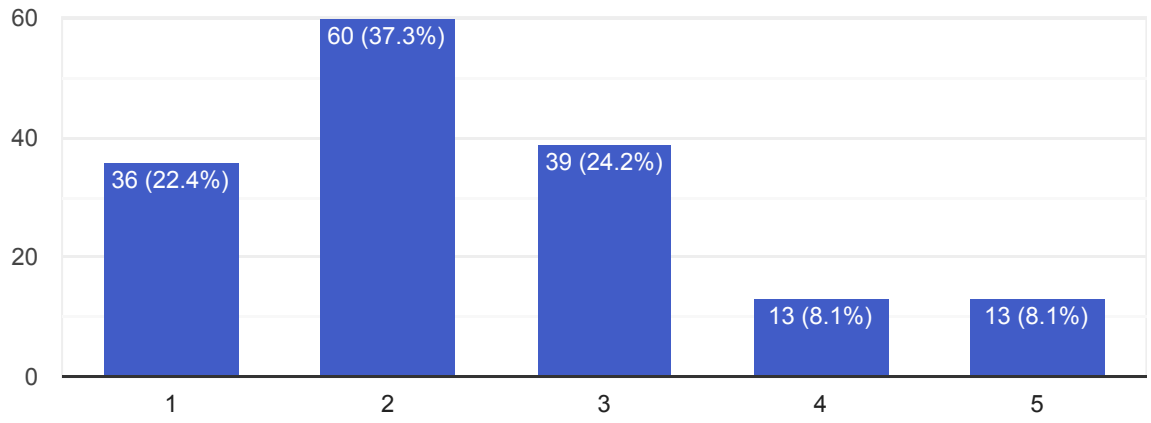
217 responses



How complex was it to get started with auto-instrumentation? (skip if you don't use)

Copy

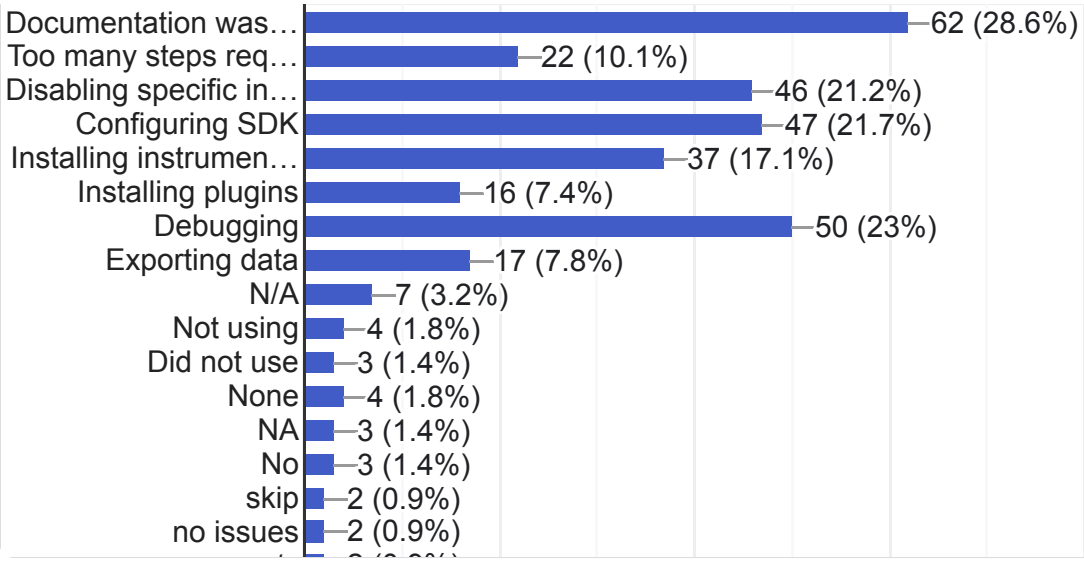
161 responses



Did you encounter any issues when using auto-instrumentation in your application? (select all that apply)

Copy

217 responses

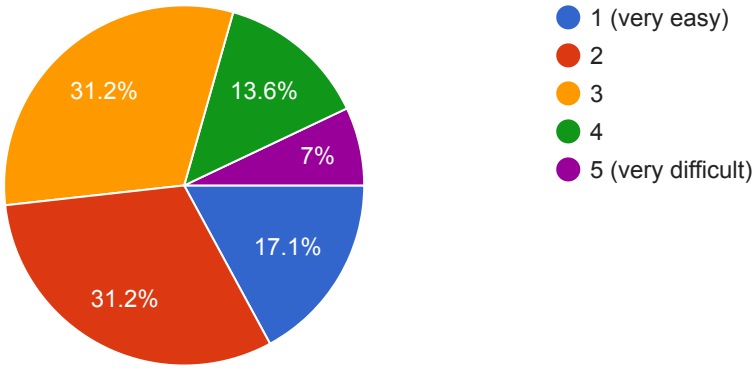


Collector usage

How complex was it to get started operating the OpenTelemetry Collector?

Copy

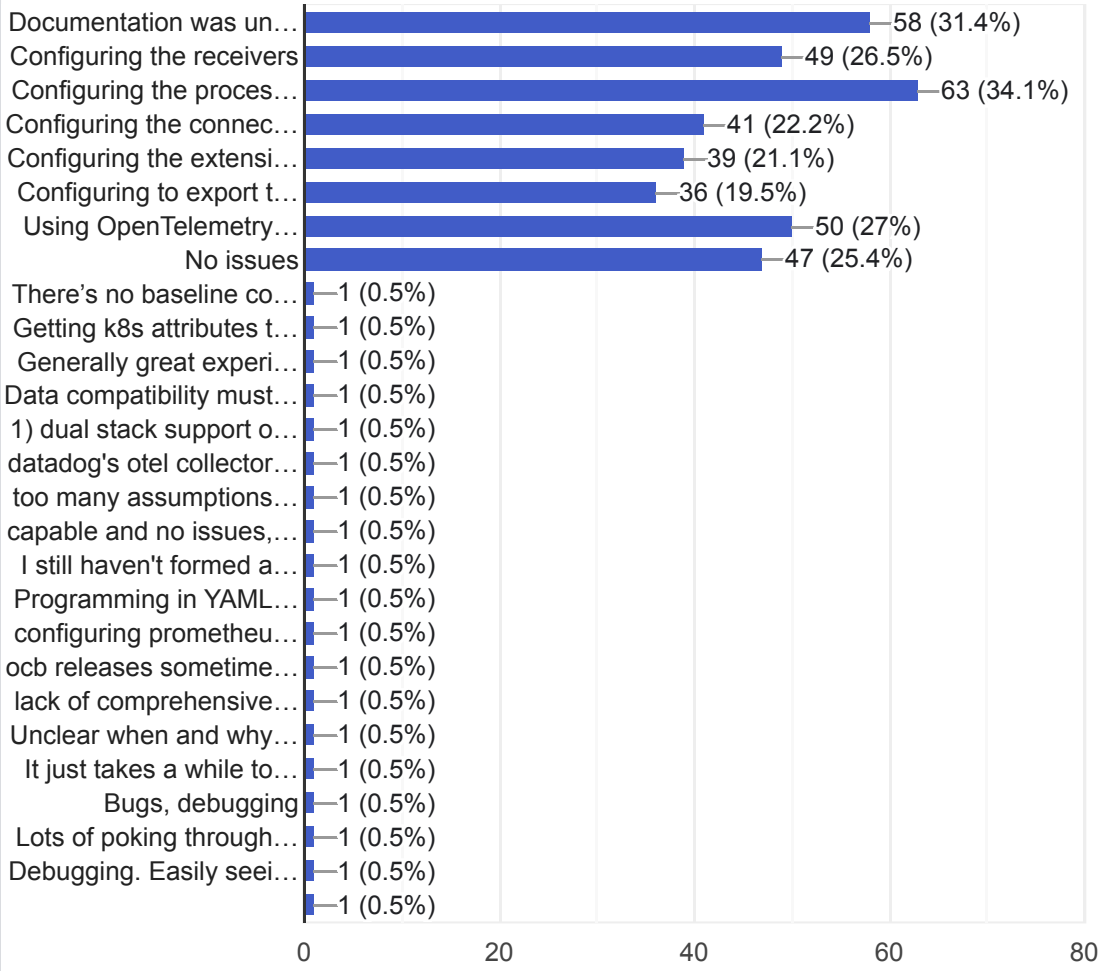
199 responses



Did you encounter any issues when operating the OpenTelemetry Collector? (select all that apply or skip if you don't use a Collector)

Copy

185 responses



Overall experience

What could be changed to improve the developer experience? For example:

- Installation of the SDK for the first time

- Debugging instrumentation

- Setting up telemetry pipelines using the Collector

- Getting telemetry out of your application

- Configuring and using resource detectors

- Troubleshooting problems during SDK setup

- Configuring instrumentation

- Too difficult to adopt OpenTelemery for all signals (tracing, metrics, logs)

217 responses

Configuring instrumentation

Debugging instrumentation

Troubleshooting problems during SDK setup

Too difficult to adopt OpenTelemery for all signals (tracing, metrics, logs)

Getting telemetry out of your application

NA

-

Best practices should be language specific. Otel metrics and logs arent supported by elixir. And auto instrumentation isnt supported.

I would _love_ an easier local developer set up to help engineers get started and debug and use OTel locally to get the most value from it. We don't yet support tracing centrally, but there's no reason engineers couldn't benefit from it locally! I know the collector can do it, but not many teams are using docker here so bringing all the components together is a bit of an exercise.

Improve code samples

Improvise documentation.. sample complex application based workshop for practicing implementation from basic to advance in different environment like AWS, azure, kubernetes etc. especially using different otel components

Too difficult to adopt otel for all signals, and also improve the documentation.

Better OTTL documentation. Having basic examples is nice but having more real world examples for pipeline transformations and structuring multiple pipelines for performance reasons would have been very helpful. Same with language specifics I find examples to trump documentation especially when the documentation is telling you directions and telling you the why's.

More complete and detailed documentation.
Ability to view logs or screens of input and final output results

Keep doc updated and examples, configs in examples should always be complete not only focus on a given processor, Receiver or exporter

- documentation needs A LOT more REAL examples
- clarify when to use what
- generally pretty messy and too many layers of indirection
- setting up pipelines beyond hello world
- best practices

The docs need to be more consistent and allow you to get answers to questions like "wait, what is that portion of the config doing?" I also think they could stand to be a bit more opinionated for someone setting this up for the first time (http vs grpc, etc).

* Prioritizing OpAmp, enables it to be used in production with very less custom code to manage org wide configuration.

* Stabalize Logs exporter (Datadog)

more complete examples

Otel-go tracing API is difficult to discover, partially because of having too many packages that interact with each other.

Better examples of connecting to third-party tracing services

/

More production examples and real production best practices showing how to name operations, for what to create spans and for what not.
Also, having some nice (not Jaeger) local tool to inspect data would be great. Something like datadog, but just to inspect one file.

More example usages documented

Better DX on how to get started but still be able to go down and customize as needed

Integration of manual instrumentation with auto instrumentation. How they can work together.

Troubleshooting problems during SDK setup

I love that the model is the same across all languages, but do wish that api Vs ask split didn't exist, or at least the names given to API structures were different

Documentation

Improve the OTLP HTTP exporter to have dynamic number of concurrent requests (e.g. dynamic consumers from queue).

I have already passed the threshold of being a beginner, and it is somewhat difficult to recall what challenges I faced at the beginning.

We need auto instrumentation for all technologies like java today .

More explanations on sampling rates and sampling policies,
and how sampling settings can impact performance.

Provide recommendations and guidance for businesses on how to evaluate sampling rates and sampling policies.

x

Documentation for languages and plugins in those languages

No working examples of applications, to run them locally on docker and debug, to understand how it works.

Setting up the underlying protocol libraries is quite hard i most languages.

First Time is hard

Documentation most of all things

K

Before the Apire dashboard, we found it hard to observe generated telemetry during local dev

.NET instrumentation

Making a more clear path and documentation. Having clear sections with best practices. There are many aspects that you need to put effort into understanding, for example the baggage api or the tracer shim in .net. The explanation is poor. Also the contrib vs core repos make things even more confusing. I understand it's work in progress.

Observing the collector itself is confusing, and only Honeycomb had even a starting point for setting the whole thing up in a way where it had good defaults without completely overwhelming volume coming out.

It’s way too hard to apply an attribute to all child spans of a trace. Very common use case and we ended up writing a bunch of iterations of custom processors to achieve it.

Debugging the collector is often a case of “deploy to staging and hope it worked” for us.

Make configuration and customization of auto instrumentation easier

Documentation and profiling signals

Pipeline

Javascript ecosystem is soo behind and not well organized. I guess everyone must be using Java or C# right.

for provisioning observability infrastructure, like ansible terraform, etc.

no auto-instrumentation by default, I prefer explicitly mention which libs to instrument.

1) Documentation is not clear and not uniform 2) Troubleshooting problems during SDK setup

Overall the documentation was clear. Initial learning curve was hard as we started the adoption couple of years back. Tackling breaking changes was bit of a challenge. Configuring extension with distro is still a challenge as there is no clear documentation on this topic. If we have more SIG meetings during APAC timezone it will be more useful.Thanks a ton for all the hardwork and bringing OpenTelemetry to a state that we can confidently use it in production.

In the case of PHP SDK, by default all instrumetations are enabled, so to disable it the OTEL_PHP_DISABLED_INSTRUMENTATIONS=all env var must be set, but I think this should be opposite and by default nothing should happen

Stop making breaking changes to the libraries every other release. If you insist on having so many different crates for the rust project also just have one crate where we pull things in via features and more example projects. I'm kind of tired of finding a crate I used is now deprecated, moved as a feature into another crate or moved out of a crate into it's own feature.

Also you frequently break the documentation links either by using sealed traits or some sort of semi-private obfuscation method so the docs are near impossible to navigate i.e. getting to builder methods. https://docs.rs/opentelemetry-otlp/0.27.0/opentelemetry_otlp/struct.SpanExporter.html#method.builder take me to the SpanExporterBuilder... oh wait you can't

The configuration is relatively complex and the version is updated too quickly

Where is Recovery me ?

Nothing yet- we are not far enough in the journey to get our customers' feedback in a productive manner

Grameen and Robi

Get someone to redesign the docs site and rewrite the docs

Better docs, single source of docs or just information where to find docs related to an instrumentation library, now getting started docs are on the opentelemetry.io page, but to actually understand how to use specific instrumentation libraries you need to find the docs on github.

Now I see that entries on the Registry page link to docs on github. But the registry page includes a lot of libraries, I'm mostly interested in the "official" libraries from https://github.com/open-telemetry/opentelemetry-dotnet-contrib, maybe libraries from opentelemetry-dotnet-contrib should be displayed first on the Registry page, instead of ordering by name.

Also each entry on the Registry page take a lot of space so it is not easy to see what libraries are available, just list of directories on github https://github.com/open-telemetry/opentelemetry-dotnet-contrib/tree/main/src shows a better overview of available libraries.

focusing on the go sdks:
* tracing is very nice, feels natural to use
* metrics are middle of the road, the api feels clunkier than prometheus, so I needed to develop wrappers to offer a more prometheus-like experience on top of otel metrics
* logging feels just bad, slog is an amazing interface and at least there's the slog bridge, but it's not something I enjoy using

overall documentation is just very lackluster, it only has "toy" examples and there's barely any info about how to correctly manage the lifecycle of providers, how to correctly inject and extract info from the contexts, especially on api environments, how to trace and get metrics for api calls, with correct error statuses, etc.

documentation and cleaner clients

Nothing comes to mind

i dont want to have to configure instrumentation.js for nodejs environment

Installation of the SDK for the first time: was very difficult to get the C++ implementation built on an air-gapped network.

Better documentation of the contrib ecosystem. Less SDK churn.

Breaking changes in attribute names between otel versions makes upgrading difficult. Breaking changes in the erlrang/elixir sdk cause issues when clients depend on different versions and libraries can't co-exist.

Overall it has been a good experience

simpler Collector cofigurations

Better documentation, especially for logs in Python.

More tools and examples

My biggest request is more common examples in cookbooks. Simple things like starting traces, propagating baggage. Every time I feel I have to reference my old code or GitHub search for examples

Most of the issues were with AWS specific stuff, disabling conflict with xray

None come to mind

making it easier to use auto instrumentation and to test it/debug it

otel is a fantastic idea, would like to see it have a more 'basic' / core implementation rather than all the abstraction layers, everywhere, all at once.

1) More test tooling, e.g. verify that proper signals are being emit during unit tests; 2) making more of the 2nd order tools visible (e.g. resource detectors, I only stumbled upon them accidentally, the docs I originally read did not point them out as a tool); 3) FE / React tracing is pretty horrible; 4) more human-language examples of how to use/aggregate/display/monitor Metrics signals, especially for complex and/or distributed use-cases.

Debugging the otel collector

The WP auto instrumentation could really be improved.

Easy documentation

First of all: I think OpenTelemetry is great and I try to use it wherever I can. The semantic conventions are very helpful.

But there are some issues I encountered:

* Sometimes it is diffucult to find the relevant documentation. I have to switch between the OTel website and the GitHub repository to find all information.

* It is confusing to have to many OTel Collector distributions. Also the versioning (1.0.0-RCx vs. 0.115.0 for example) is complicated. Why not build one distribution with all modules and only enable the needed modules via configuration?

* I miss concrete configuration examples for OTel Collector in production. For example how to make sure that no data is lost when the backend is not available.

* Suggestion: A collection of ready-to-use configuration snippets for logs of common applications would be helpful. Especially for unstructured logfiles or logs with non-standard named attributes. These snippets could parse the logs and also use attributes that follow the

semantic conventions.

* Suggestion: An overview table of all semantic conventions names on one page. This could be helpful because it is searchable in the browser. Also user does not have to know the category.

AutoInstrumentation for go

We started pretty early on when there wasn't much available in terms of bootstrapping the SDK. In Golang e.g. we needed to write a hub CJ of boilerplate code to bootstrap the SDK and ended up building a wrapper library. Our users found the API too complicated to use and would make common errors like not sharing the context properly. The code to instrument is very verbose and "gets in the way" of your domain logic. It's also seemingly impossible to tune the SDK. We don't know if we're dropping data or overflowing the buffer without staring at diagnostic logs. We've avoided adoption OTel Metrics until the API stabilized but I think we find it extremely complicated compared to statsd. The Tracing API is also just different enough from more familiar log or event style APIs. It adds more mental overhead by disambiguating something that is effectively a structured event with some attributes. Verbosity in semantic conventions has also been challenging. The guidance around using our own company domains for namespaces makes it verbose and adds a lot of overhead to payloads as we push them around the network. It's a tough trade off from having structured Any types that we could include in messages. I know that these are non trivial problems but i think there are some opportunities to simplify things for OTel 2.0

Better Documentation

Instrumentation

Auto-instrumentation is too limited and manual instrumentation is too contextual to the application and its classes / methods.

Difficult to get good telemetry signals OOB without extensive configuration (transforming traces to metrics to cover basic metrics most observability agent vendors provide by default, easier sensitive data redaction)

Microsoft integration is still in beta, some integrations need opt -in which isn't well documented.

Knowing all the general otel concepts seems to be required in order to begin using otel. Can be a large barrier to entry

I don't have much here. After coming from a number of different providers, I was fairly satisfied with how easy it was to get a minimal ecosystem up and running.

Debug and testing complex rules

I don't find installing and using OTEL related components very difficult. I've sometimes felt a bit lost in the documentation though. One thing that puzzled me initially was the fact that receivers/processors/exporters are split across main and contrib repos, but then I got used to it.

Possibility to change the configuration of the Otel Zero-code Agents runtime without restarting the processes.

Make the happy path easier. Work with library, language, framework, authors to build in the otel-api. Work with platforms to make OTLP streams first-class. Better local tooling like otel-desktop-viewer, otel-cli. I can pipe logs to a file. I should be able to pipe telemetry into something that lets me visualize and explore it. Don't make me run docker containers or figure out what a jaeger is. Make a VS Code extension. Give me a native app.

Clearer debugging story, better language specific documentation, explanations of core concepts written for mere mortal consumption.

Getting good local debugging output: being able to see traces/metrics locally easier without needing to set up a collector + grafana

Configuring

Why do much contrib packages? Having a well structured lib would lead to a better dev xp. Also it would be awesome to have examples on how to export telemetry directly to the backend without using a collector (for serverless applications)

Debugging for certain. Other than that, go sdk is unfortunately bloated and it would be nice to get the ability to work with custom contexts in golang as opposed to having to do a huge refactor of legacy code

Nothing

Manual span instrumentation to enrich traces and context propagation.

103 more responses are hidden