



オブザーバビリティ実践編

株式会社サイバーエージェント AI事業本部

岩見彰太

GitHub: [@Blwashi](#)

X: [@B_Sardine](#)

自己紹介



岩見彰太 / Iwamin 株式会社サイバーエージェント

2022年度新卒入社

AI事業本部 協業リテールメディア Div.

アプリ運用カンパニー



 @Blwashi

 @B_Sardine



自動生成を活用した、運用保守コストを抑える Error/Alert/Runbook の一元集約管理



Feature Flag Deep Dive - Speaker Deck

注意

- ハンズオンでは少し Go を書きます

多言語でも似たように記述したりするので雰囲気を感じてもらえればOKです

- 全員共通の AWS、Datadog アカウントを使います

Perman で入れるようになっていればOK

- ツール群に関してはさわりぐらいしか解説できません

ハンズオンの時間に遊んでみましょう！

突然ですが

旅行で忘れ物をした時
どうやって見つけますか？

北海道旅行 財布紛失編

北海道旅行に行こう



新千歳空港



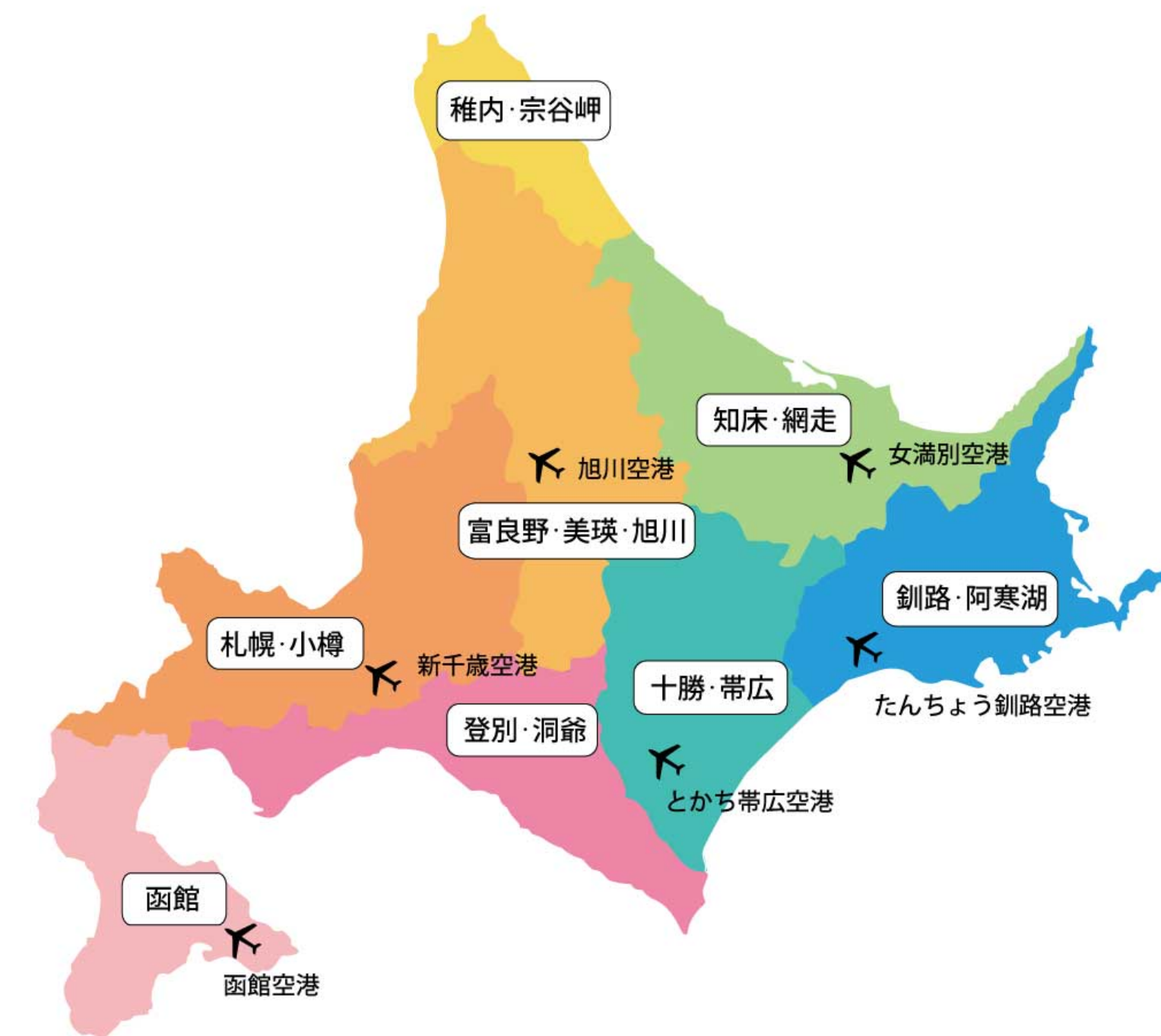
北海道旅行に行こう



新千歳空港



札幌観光



北海道旅行に行こう



新千歳空港



札幌観光



小樽観光



北海道旅行に行こう



新千歳空港



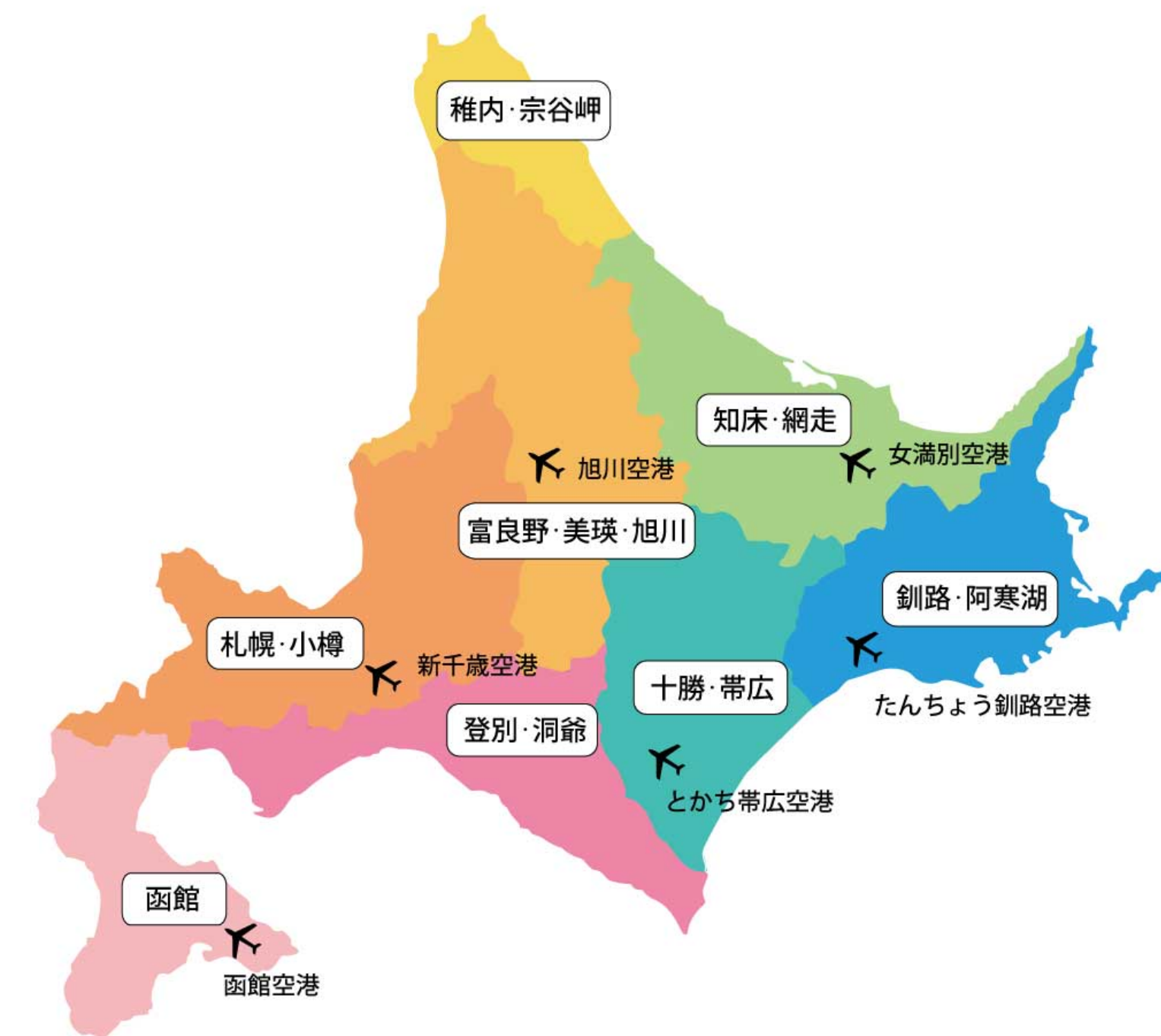
札幌観光



小樽観光



網走観光



北海道旅行に行こう



新千歳空港



札幌観光



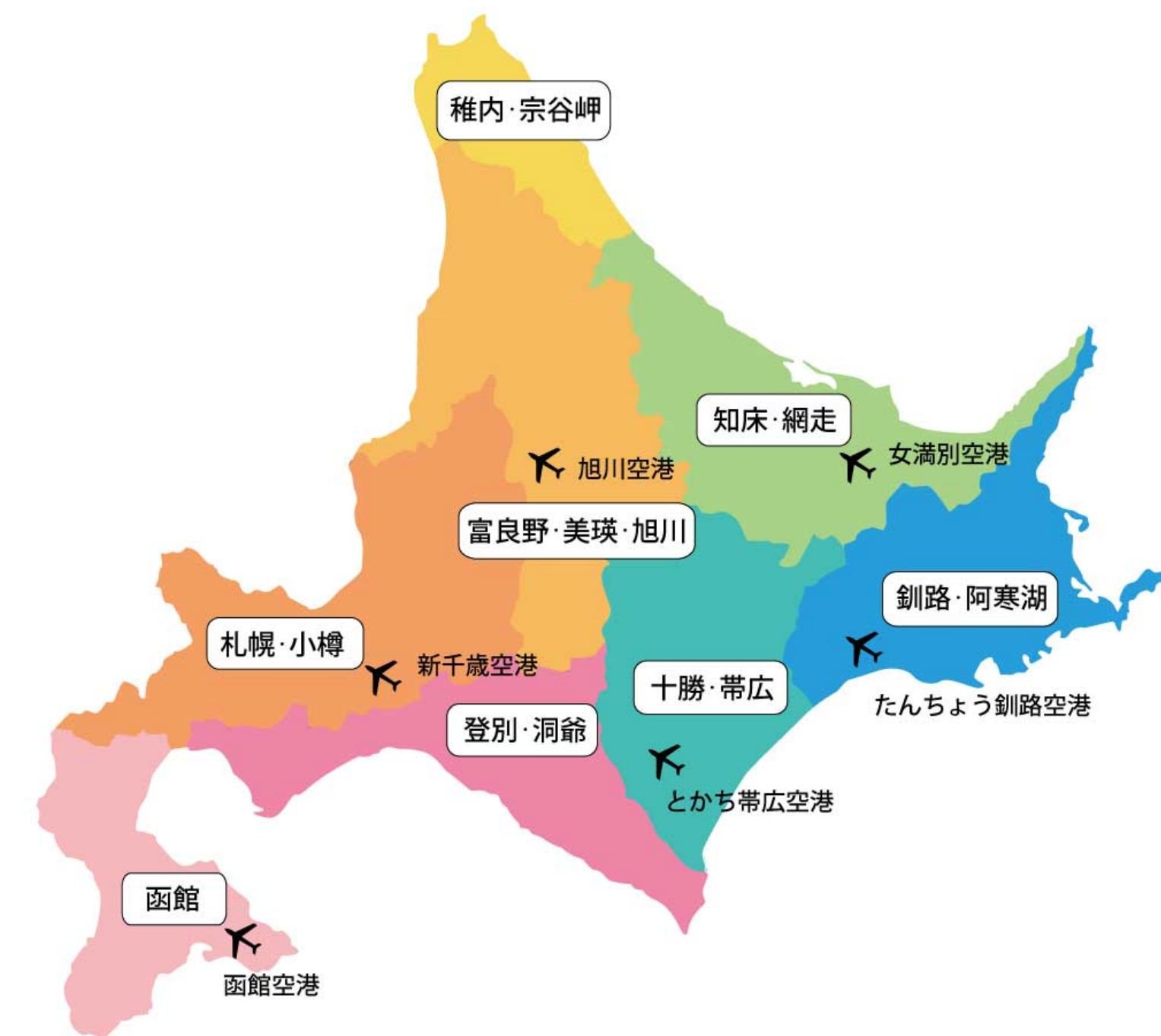
小樽観光



網走観光



旭山動物園



北海道旅行に行こう



新千歳空港



札幌観光



小樽観光



網走観光



旭山動物園



旭川空港



北海道旅行に行こう



札幌・小樽

知床・網走

富良野・美瑛・旭川



新千歳空港



札幌観光



小樽観光



網走観光



旭山動物園



旭川空港



北海道旅行に行こう



札幌・小樽

空港

札幌

小樽

知床・網走

網走監獄

富良野・美瑛・旭川

旭山動物園

空港



新千歳空港



札幌観光



小樽観光



網走観光



旭山動物園



旭川空港



北海道旅行に行こう



札幌・小樽

空港

札幌

小樽

知床・網走

網走監獄

富良野・美瑛・旭川

旭山動物園

空港



財布紛失!



新千歳空港



札幌観光



小樽観光



網走観光



旭山動物園



旭川空港



北海道旅行に行こう



Service A

func 1

func 2

func 3

Service B

func 4

Service C

func 5

func 5



Error



新千歳空港



札幌観光



小樽観光



網走観光



旭山動物園



旭川空港

旅行（リクエスト）して返ってきたら
財布がなくなっていた（エラー発生）



北海道旅行に行こう



Service A

func 1

func 2

func 3

Service B

func 4

Service C

func 5

func 5



どこで何が起きていたか把握するのが大切

旅行（リクエスト）して返ってきたら
財布がなくなっていた（エラー発生）



1

Observability とは



Observability is なに？

Observability = 可観測性…?

あの行列のやつ…?



航空宇宙工学科出身

2. 可観測性

システムの入力を有限時間観測することによって、観測開始時のシステムのすべての状態変数の成分を知ることができるか否かという特性を**可観測性**という。可観測性を有するシステムに対し、「システムは可観測である」、「可観測なシステム」という言い方をする。

式(1)の線形時不変システムにおいて、初期時間0から有限時間 t_f までの入力 $\mathbf{u}(t)$ と出力 $\mathbf{y}(t)$ から、初期状態 $\mathbf{x}(0)$ を一意に求めることができるならば、システムは可観測である。可観測であるならば、有限な時間区間 $0 \leq t \leq t_f$ での入力 $\mathbf{u}(t)$ と出力 $\mathbf{y}(t)$ の推移から、その時間区間におけるすべての状態 $\mathbf{x}(t)$ の推移を計算できる。可観測性は、入力が既知であることを仮定しているので行列 \mathbf{B} 、 \mathbf{D} には無関係となり、行列 \mathbf{A} 、 \mathbf{C} のみによって定まるので、「 (\mathbf{A}, \mathbf{C}) は可観測である」ということもある。

定理 式(1)のシステムが可観測であるための必要十分条件は、

$$M_o = \begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \\ \vdots \\ \mathbf{CA}^{n-1} \end{bmatrix} \quad (20)$$

として定義される可観測行列（可観測性行列） M_o がフルランクをもつこと、すなわち、

$$\text{rank } M_o = n \quad (21)$$

である。

制御工学、機械工学における可観測性

$$\begin{cases} \dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u} \\ \boldsymbol{y} = \boldsymbol{C}\boldsymbol{x} + \boldsymbol{D}\boldsymbol{u} \end{cases} \quad (1)$$

式(1)の線形時不変システムにおいて、初期時間0から有限時間 t_f までの入力 $\boldsymbol{u}(t)$ と出力 $\boldsymbol{y}(t)$ から、初期状態 $\boldsymbol{x}(0)$ を一意に求めることができるならば、システムは可観測である。可観測であるならば、有限な時間区間 $0 \leq t \leq t_f$ での入力 $\boldsymbol{u}(t)$ と出力 $\boldsymbol{y}(t)$ の推移から、その時間区間におけるすべての状態 $\boldsymbol{x}(t)$ の推移を計算できる。可観測性は、入力が既知であることを仮定しているので行列 \boldsymbol{B} 、 \boldsymbol{D} には無関係となり、行列 \boldsymbol{A} 、 \boldsymbol{C} のみによって定まるので、「 $(\boldsymbol{A}, \boldsymbol{C})$ は可観測である」ということもある。

- システムの出力を観測することで、そのシステムの全ての状態変数の成分を知ることができるか否かという特性

システムの振る舞い^{（赤文字）}を出力から知ることができるか否か

ソフトウェアアシステムにおける

Observability is なに？

ソフトウェアシステムにおけるo11y

簡単にいうと、私たちが考えるソフトウェアシステムの「オブザーバビリティ」とは、システムがどのような状態になったとしても、それがどんなに斬新で奇抜なものであっても、どれだけ理解し説明できるかを示す尺度です。

また、そのような斬新で奇抜な状態に対しても、事前にデバッグの必要性を定義したり予測したりすることなく、システムの状態データのあらゆるディメンションやそれらの組み合わせについてアドホックに調査し、よりデバッグが可能になりようにする必要があります。

もし、新しいコードをデプロイする必要がなく、どんな斬新で奇抜な状態でも理解できるなら、オブザーバビリティがあると言えます。

by オブザーバビリティ・エンジニアリング



ソフトウェアシステムにおけるo11y

簡単にいうと、私たちが考えるソフトウェアシステムの「オブザーバビリティ」とは、システムがどのような状態になったとしても、それがどんなに斬新で奇抜なものであっても、**どれだけ理解し説明できるか**を示す尺度です。

また、そのような斬新で奇抜な状態に対しても、事前にデバッグの必要性を定義したり予測したりすることなく、システムの状態データのあらゆるディメンションやそれらの組み合わせについてアドホックに調査し、よりデバッグが可能になりようにする必要があります。

もし、新しいコードをデプロイする必要がなく、どんな斬新で奇抜な状態でも理解できるなら、オブザーバビリティがあると言えます。

by オブザーバビリティ・エンジニアリング



ソフトウェアシステムにおけるo11y

簡単にいうと、私たちが考えるソフトウェアシステムの「オブザーバビリティ」とは、システムがどのような状態になったとしても、それがどんなに斬新で奇抜なものであっても、どれだけ理解し説明できるかを示す尺度です。

また、そのような斬新で奇抜な状態に対しても、事前にデバッグの必要性を定義したり予測したりすることなく、システムの状態データのあらゆるディメンションやそれらの組み合わせについてアドホックに調査し、よりデバッグが可能になりようにする必要があります。

もし、**新しいコードをデプロイする必要がなく、どんな斬新で奇抜な状態でも理解できる**なら、オブザーバビリティがあると言えます。

by オブザーバビリティ・エンジニアリング



ソフトウェアアシステムにおけるo11y

- 制御理論の尺度として使われていたオブザーバビリティをソフトウェア領域に拡張
- ソフトウェアにオブザーバビリティを持たせるための要件

by オブザーバビリティ・エンジニアリング

アプリケーションの内部構造を理解する

予想できないことが起こったとしても、どのような状態に陥っているか理解する

外部ツールを使って観測・調査し、内部構造を理解する

コードを改修することなく、内部情報を理解する

Monitoring と Observability の違い

	モニタリング（監視）	オブザーバビリティ（可観測性）
調査	システムの状態を既知の閾値と照合	問題がどこでなぜ発生しているか反復探査的な調査が可能
アプローチ	リアクティブなアプローチ	既知・未知に関わらず、積極的なアプローチ
最高のデバッガー	組織に長くいる者	最も好奇心が高いエンジニア
隠れた問題や発見に対して	本当の問題が見えにくい、状態の緩和をまずしてしまう	正しい答えを導き出すためにデータを追う
ツール間や被疑箇所間の相関	固有の非互換や矛盾が発生し、相関を導き出すのことに苦勞	明確なデータになっており、どのエンジニアでも探索可能
障害領域の予想	荒いレベルのメトリクスとひらめきを組み合わせる	分散トレースや複数のシグナルを組み合わせることで俯瞰

Monitoring と Observability の違い

	モニタリング (監視)	オブザーバビリティ (可観測性)
調査	システムの状態を既知の閾値と照合	問題がどこでなぜ発生しているか反復探査的な調査が可能
<p>システムの複雑さや規模が増して、モニタリングの限界が現れ始めた オブザーバビリティの誕生 🧒</p>		
ツール間や被疑箇所間の相関	固有の非互換や矛盾が発生し、相関を導き出すのことに苦勞	明確なデータになっており、どのエンジニアでも探索可能
障害領域の予想	荒いレベルのメトリクスとひらめきを組み合わせる	分散トレースや複数のシグナルを組み合わせることで俯瞰

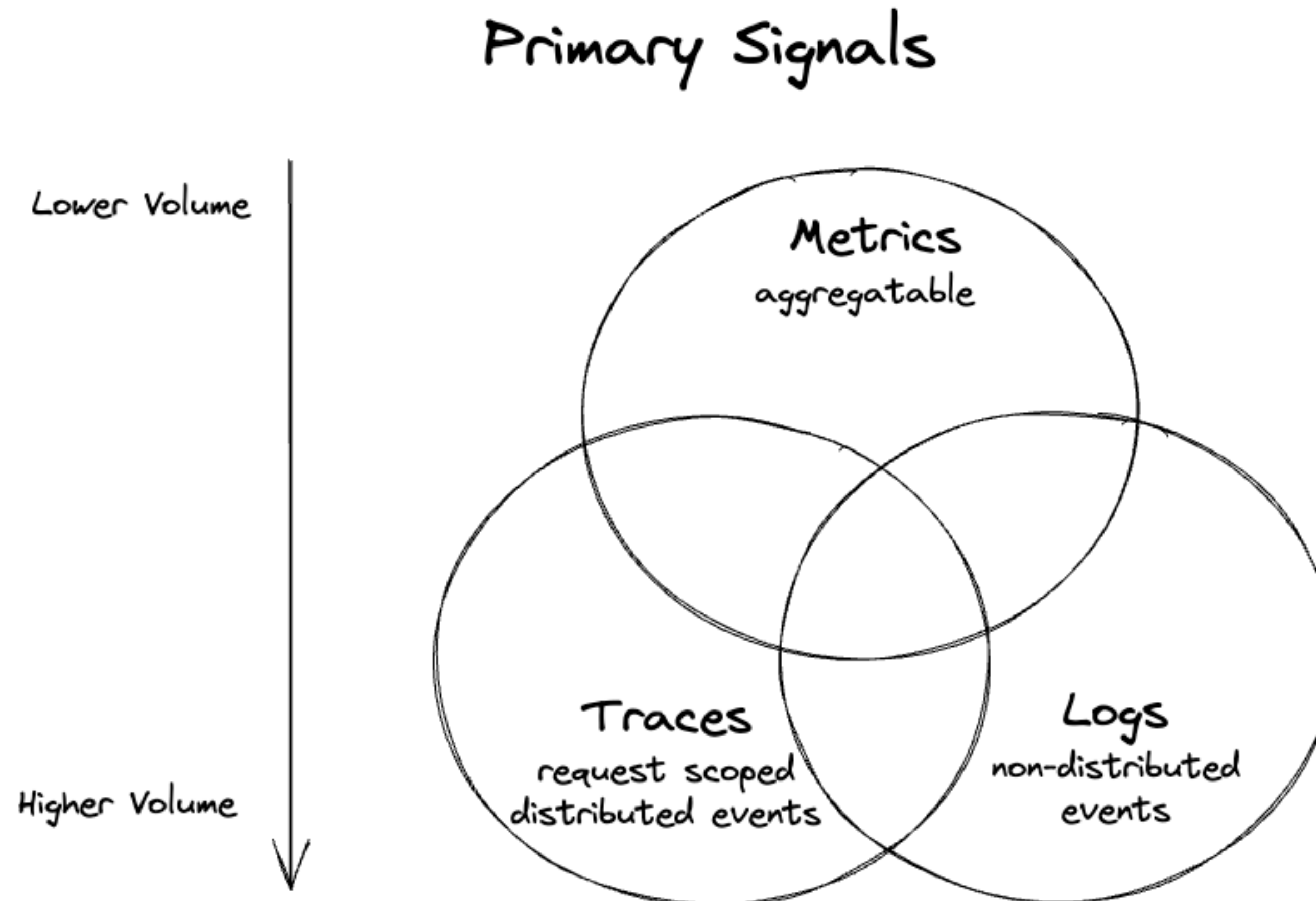
2

Observability の基礎



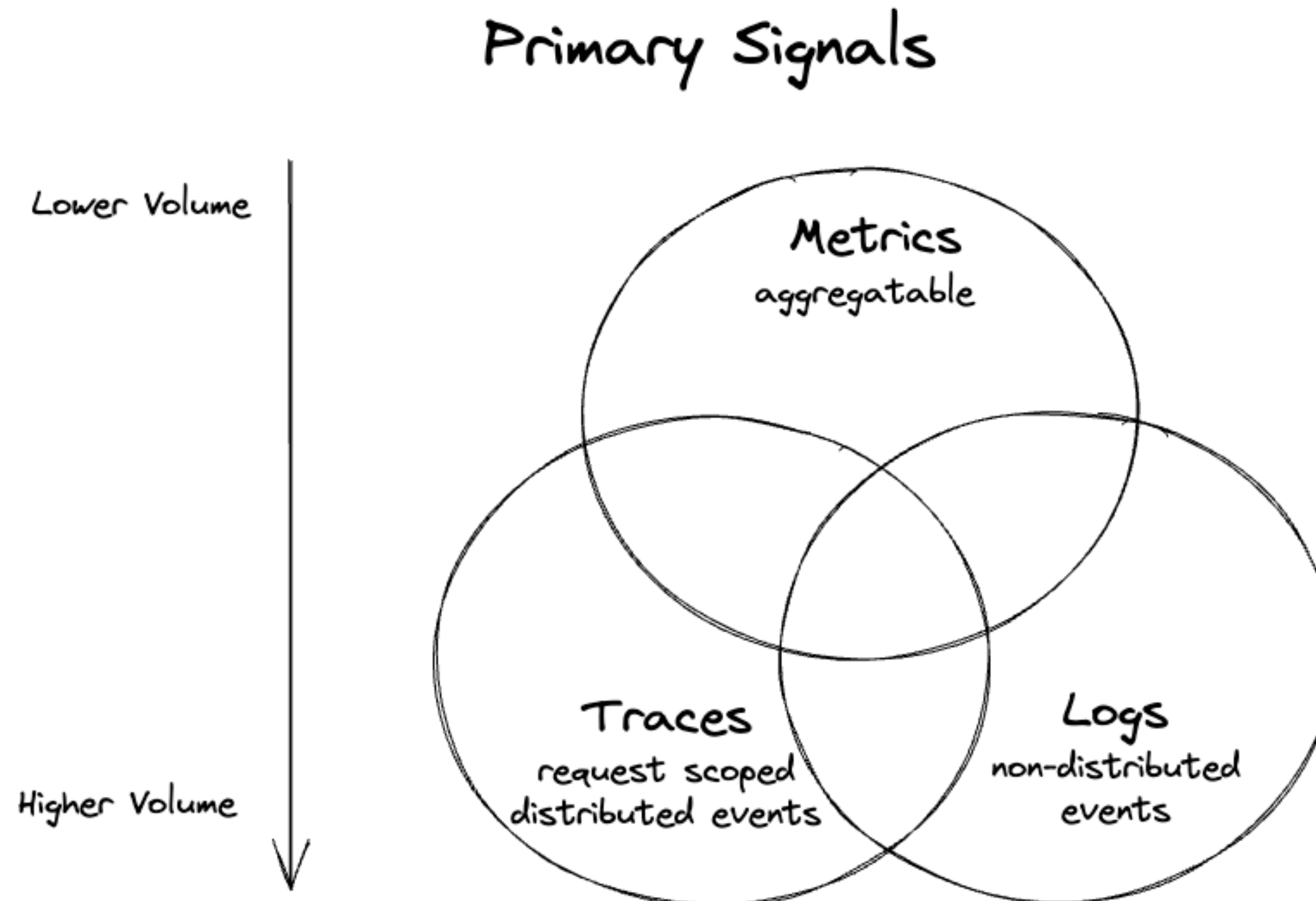
Primary Signals

- オブザーバビリティの主要なシグナル (not 三本柱)



Primary Signals

- オブザーバビリティの主要なシグナル (not 三本柱)



Primary Signals

- オブザーバビリティの主要なシグナル (not 三本柱)

There is a really good chance that you have heard about the "Three Observability Pillars", which are metrics, logs, and traces. They are commonly mentioned and probably what you're going to start with. We like to think of them as the "primary signals" instead of "three pillars" for two reasons:

Pillars carry an implicit meaning of being foundational. They are a safe place to start, yet are not always required at the same time. In fact, basing on one or two signals with a small mix of others can be a valid trade-off to improve cost efficiency (e.g. metrics and logs with ad-hoc tracing). Recently, more signals are becoming popular in open-source communities like application profiles (continuous profiling) and crash dumps. New signals with new semantics may also arise in the near future, and those interested in this topic should keep an eye open for them.

[by cndf/tag-Observability](#)

Primary Signals

- オブザーバビリティの主要なシグナル (not 三本柱)

メトリクス、ログ、トレースである「可観測性の3つの柱」について聞いたことがある可能性は非常に高いです。これらは一般的に言及されており、おそらく最初に始めるものです。私たちはこれらを「3つの柱」ではなく「主要なシグナル」と考えたいと考えています。その理由は次の2つです。

柱には、基礎となるという暗黙の意味が込められています。これらは安全に始めることができますが、必ずしも同時に必要になるわけではありません。実際、1つまたは2つの信号に基づいて他の信号を少し混ぜることは、コスト効率を向上させるための有効なトレードオフとなる可能性があります（アドホック トレースによるメトリクスとログなど）。最近、オープンソース コミュニティでは、アプリケーション プロファイル（継続的プロファイリング）やクラッシュダンプなどのシグナルが人気を集めています。近い将来、新しいセマンティクスを備えた新しいシグナルも出現する可能性があるため、このトピックに興味がある人は、常に目を光らせておく必要があります。

[by cndf/tag-Observability](#)

Primary Signals

- オブザーバビリティの主要なシグナル (not 三本柱)

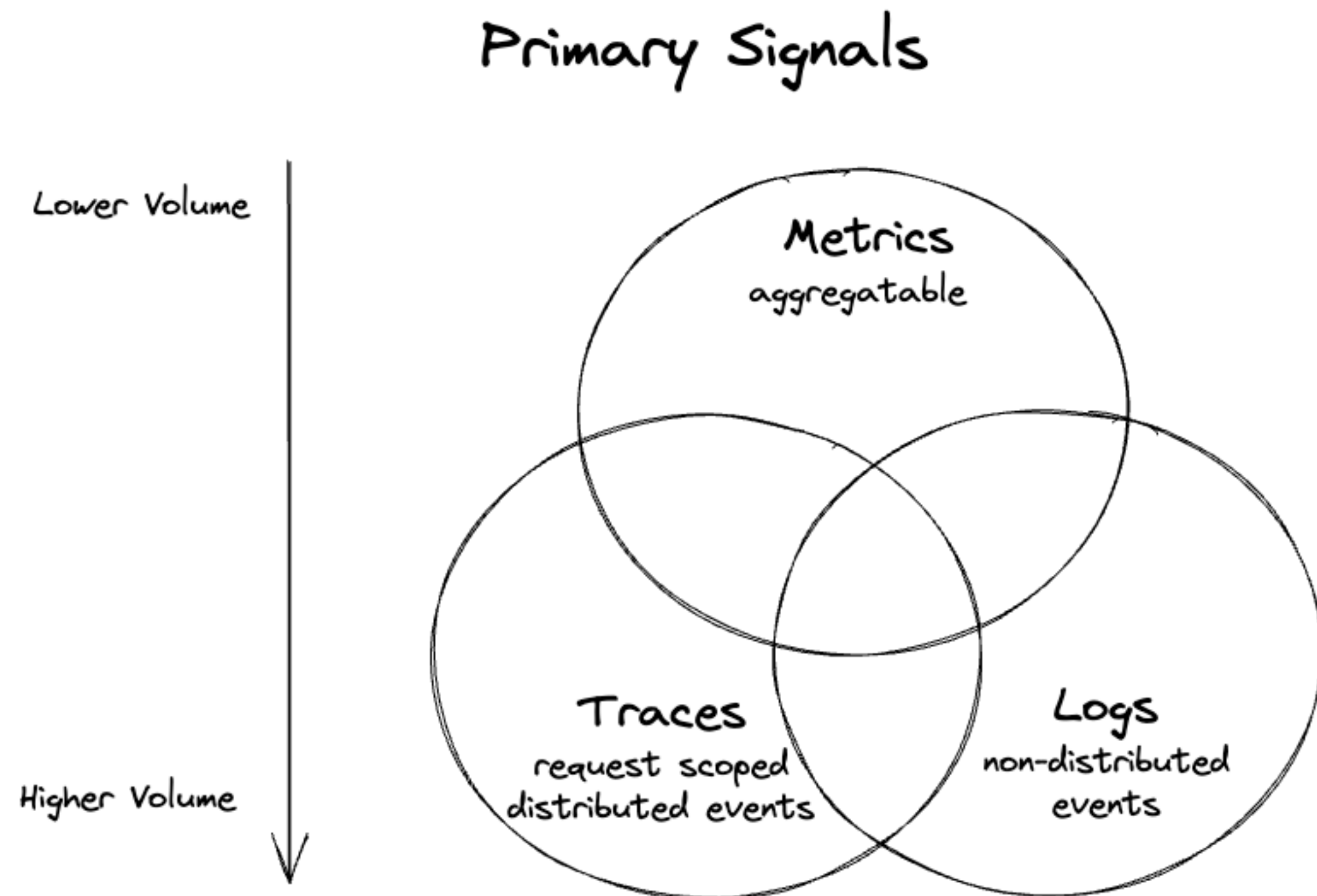
メトリクス、ログ、トレースである「可観測性の3つの柱」について聞いたことがある可能性は非常に高いです。これらは一般的に言及されており、おそらく最初に始めるものです。私たちはこれらを「3つの柱」ではなく「主要なシグナル」と考えたいと考えています。その理由は次の2つです。

柱には、**基礎となるという暗黙の意味が込められています**。これらは安全に始めることができますが、必ずしも同時に必要になるわけではありません。実際、**1つまたは2つの信号に基づいて他の信号を少し混ぜることは、コスト効率を向上させるための有効なトレードオフとなる**可能性があります (アドホック トレースによるメトリクスとログなど)。最近、オープンソース コミュニティでは、アプリケーション プロファイル (継続的プロファイリング) やクラッシュダンプなどのシグナルが人気を集めています。**近い将来、新しいセマンティクスを備えた新しいシグナルも出現する可能性がある**ため、このトピックに興味がある人は、常に目を光らせておく必要があります。

[by cndf/tag-Observability](#)

Primary Signals

- オブザーバビリティの主要なシグナル (not 三本柱)



[tag-observability/whitepaper.md](https://tag-observability.github.io/whitepaper.md) at main · cncf/tag-observability

メトリクス

ログ

トレース

メトリクス

- システムの**健康状態**
- システムの状態を常に記録する
 - CPU、メモリ、I/O、レスポンスタイム、エラーレート etc...
- 「**既知の未知のもの**」に対する最も効率的なシグナル
 - 例
 - CPU使用率が徐々に増加している
 - レスポンスタイムがとあるデプロイから悪化している
- トラフィックが増加してもボリュームが増えるようなものではないので、**コストが予測可能**

ログ

- あらゆる操作などを保存するテキストデータ
- 全てのイベントを記録しておくこと、**根本的な分析**をする際や**障害時にアプリケーションの状態を理解**できる
- **種類**

アプリケーションログ

アプリケーション内で発生したイベント

セキュリティーログ

ログインの失敗、パスワード変更、認証の失敗、リソースアクセス、リソースの変更など

システムログ

物理デバイスと論理デバイスと扱うカーネルレベルのメッセージなど

監査ログ

ユーザーのアクティビティの記録、システムの応答

インフラストラクチャログ

API、syslog やオンプレ・クラウドなどのインフラ管理部分

ログレベル

DEBUG	<p>トラブルシューティングや原因の分析に使用</p> <p>パフォーマンスに影響する可能性があるため、トラブルシューティングや障害発生時な短期間の使用を心がける</p> <p>Loggerのログレベルを調整してDev環境のみ表示されるようにするなど</p>
INFO	<p>システムがどのように動作しているかを把握するために使用</p> <p>複雑で時間を要する処理にはステップ毎に入れたりもする</p>
WARNING	<p>失敗ではないが注意を要する高レベルのメッセージに使用</p>
ERROR	<p>障害が発生した理由と詳細を確認するために使用</p> <p>スタックトレースを含めることでより詳細な調査が可能</p> <p>調査に有用な情報はできるだけ含めているのが良い</p>

構造化イベント（ログ）

- システムがどのような状態であっても、データを**任意かつ詳細に繰り返し分析**できなくてはならない by cndf/tag-Observability

- イベントとは

サービスへの影響を理解するために、ある特定のリクエストがそのサービスとやり取りをしている間に発生した全ての記録のこと
by オブザーバビリティ・エンジニアリング

つまりとあるリクエストが来てレスポンスを返すまでの間に内部に発生した全ての記録

- 任意かつ詳細に繰り返し分析

ユニークなリクエストIDや変数の値、callしたサービスなどを簡単に検索できるようにする

構造化イベント (ログ)

```
time=2024-05-13T16:36:31.168Z  
level=INFO msg=dbQuery  
service=handson env=dev  
trace.id=hoge span.id=foo  
dd.git.commit.sha=sha  
dd.git.repository=github.com/  
example/handson
```

```
{  
  "time": "2024-05-  
13T16:39:39.585656469Z",  
  "level": "INFO",  
  "msg": "dbQuery",  
  "service": "handson",  
  "env": "dev",  
  "trace.id": "hoge",  
  "span.id": "foo",  
  "dd.git.commit.sha": "sha",  
  "dd.git.repository":  
  "github.com/example/handson"  
}
```


構造化イベント (ログ)

```
time=2024-05-13T16:36:31.168Z  
level=INFO msg=dbQuery  
service=handson env=dev
```

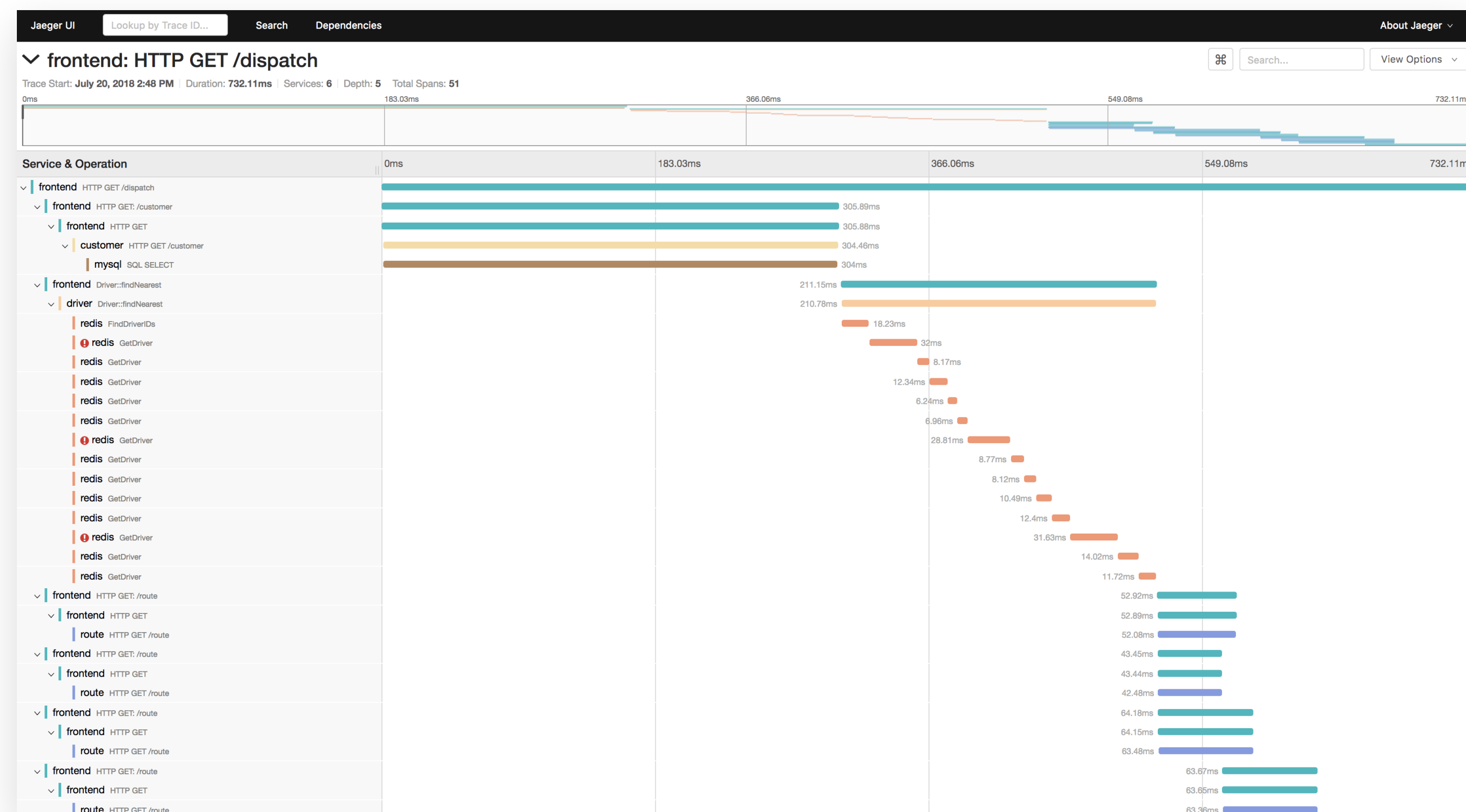
```
{  
  "time": "2024-05-  
13T16:39:39.585656469Z",  
  "level": "INFO"
```

Key-value にすると Key で検索などできる

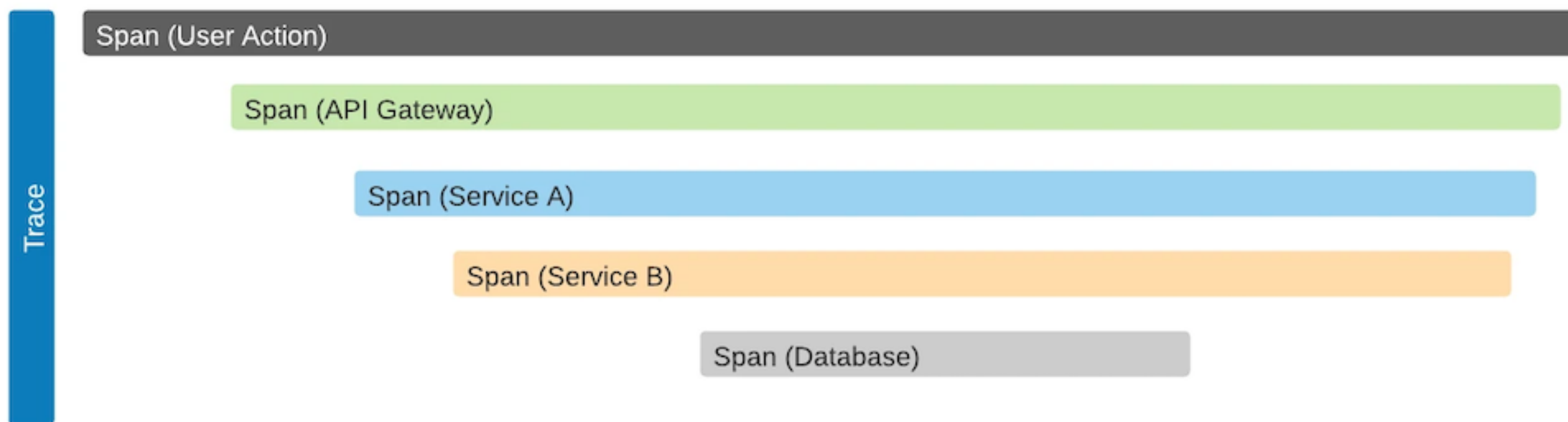
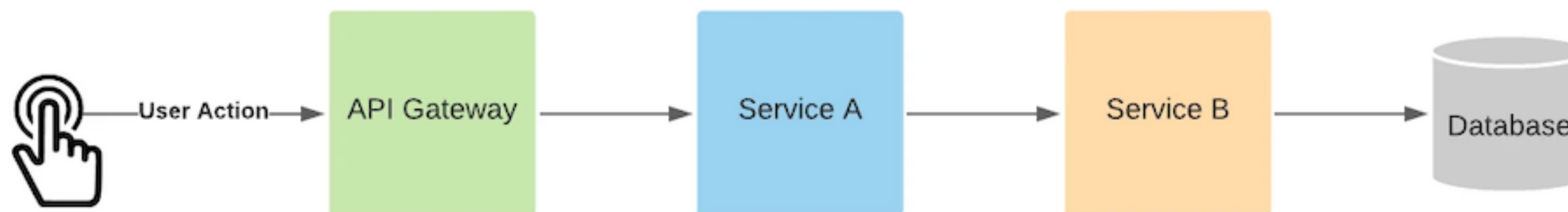
```
  "dd.git.commit.sha": "sha",  
  "dd.git.repository":  
  "github.com/example/handson"  
}
```

トレース（今日のメイン）

- エンドユーザーによって開始されたリクエストとその結果アクセスされたダウンストリーム、マイクロサービス、データストアへのリクエストなど、分散トランザクション中になのが起こったかを理解するためのシグナル



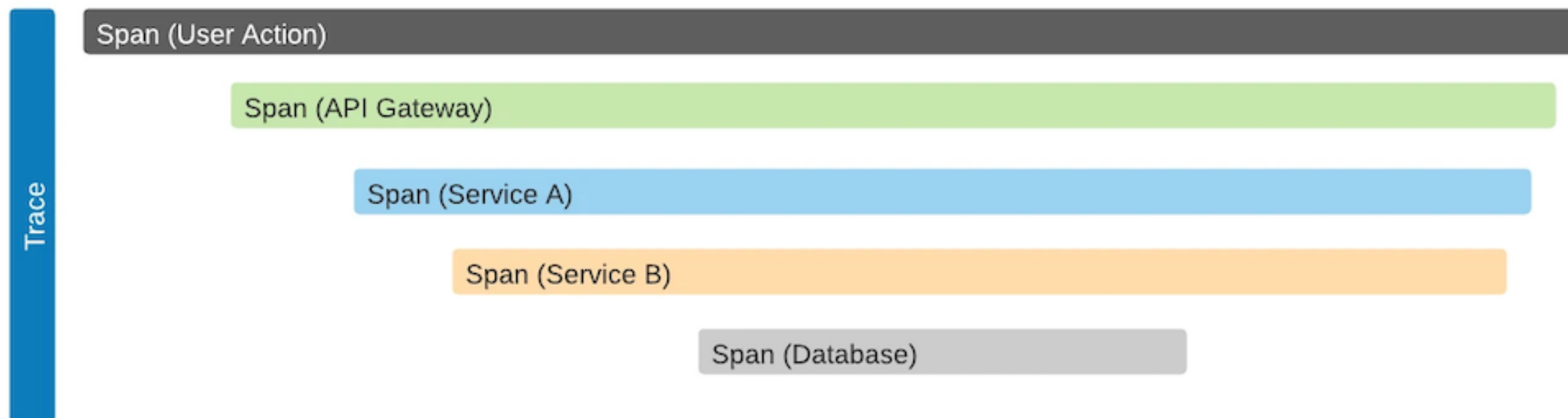
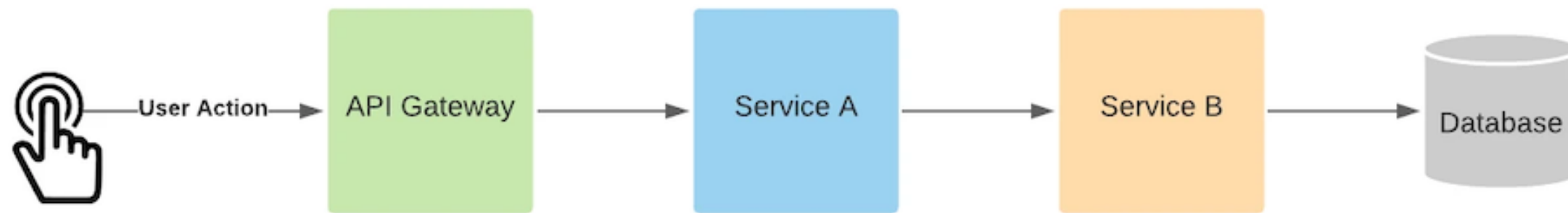
Trace と Span



Trace と Span

- Trace

Span によって盲目的に定義される
全ての Span の大元、集約



[OpenTelemetry.Trace](#)

- Span

トランザクション内の操作
次の状態をカプセル化

操作名

開始と終了のタイムスタンプ

属性値

親 span の識別子

Span参照に必要な SpanContext

SpanContext

親 span

```
{
  "name": "hello",
  "context": {
    "trace_id": "0x5b8aa5a2d2c872e8321cf37308d69df2",
    "span_id": "0x051581bf3cb55c13"
  },
  "parent_id": null,
  "start_time": "2022-04-29T18:52:58.114201Z",
  "end_time": "2022-04-29T18:52:58.114687Z",
  "attributes": {
    "http.route": "some_route1"
  },
  "events": [
    {
      "name": "Guten Tag!",
      "timestamp": "2022-04-29T18:52:58.114561Z",
      "attributes": {
        "event_attributes": 1
      }
    }
  ]
}
```

子 span

```
{
  "name": "hello-greetings",
  "context": {
    "trace_id": "0x5b8aa5a2d2c872e8321cf37308d69df2",
    "span_id": "0x5fb397be34d26b51"
  },
  "parent_id": "0x051581bf3cb55c13",
  "start_time": "2022-04-29T18:52:58.114304Z",
  "end_time": "2022-04-29T22:52:58.114561Z",
  "attributes": {
    "http.route": "some_route2"
  },
  "events": [
    {
      "name": "hey there!",
      "timestamp": "2022-04-29T18:52:58.114561Z",
      "attributes": {
        "event_attributes": 1
      }
    },
    {
      "name": "bye now!",
      "timestamp": "2022-04-29T18:52:58.114585Z",
      "attributes": {
        "event_attributes": 1
      }
    }
  ]
}
```

SpanContext

親 span

```
{
  "name": "hello",
  "context": {
    "trace_id": "0x5b8aa5a2d2c872e8321cf37308d69df2",
    "span_id": "0x051581bf3cb55c13"
  },
  "parent_id": null,
  "start_time": "2022-04-29T18:52:58.114201Z",
  "end_time": "2022-04-29T18:52:58.114687Z",
  "attributes": {
    "http.route": "some_route1"
  },
  "events": [
    {
      "name": "Guten Tag!",
      "timestamp": "2022-04-29T18:52:58.114561Z",
      "attributes": {
        "event_attributes": 1
      }
    }
  ]
}
```

子 span

```
{
  "name": "hello-greetings",
  "context": {
    "trace_id": "0x5b8aa5a2d2c872e8321cf37308d69df2",
    "span_id": "0x5fb397be34d26b51"
  },
  "parent_id": "0x051581bf3cb55c13",
  "start_time": "2022-04-29T18:52:58.114304Z",
  "end_time": "2022-04-29T22:52:58.114561Z",
  "attributes": {
    "http.route": "some_route2"
  },
  "events": [
    {
      "name": "hey there!",
      "timestamp": "2022-04-29T18:52:58.114561Z",
      "attributes": {
        "event_attributes": 1
      }
    },
    {
      "name": "bye now!",
      "timestamp": "2022-04-29T18:52:58.114585Z",
      "attributes": {
        "event_attributes": 1
      }
    }
  ]
}
```

3

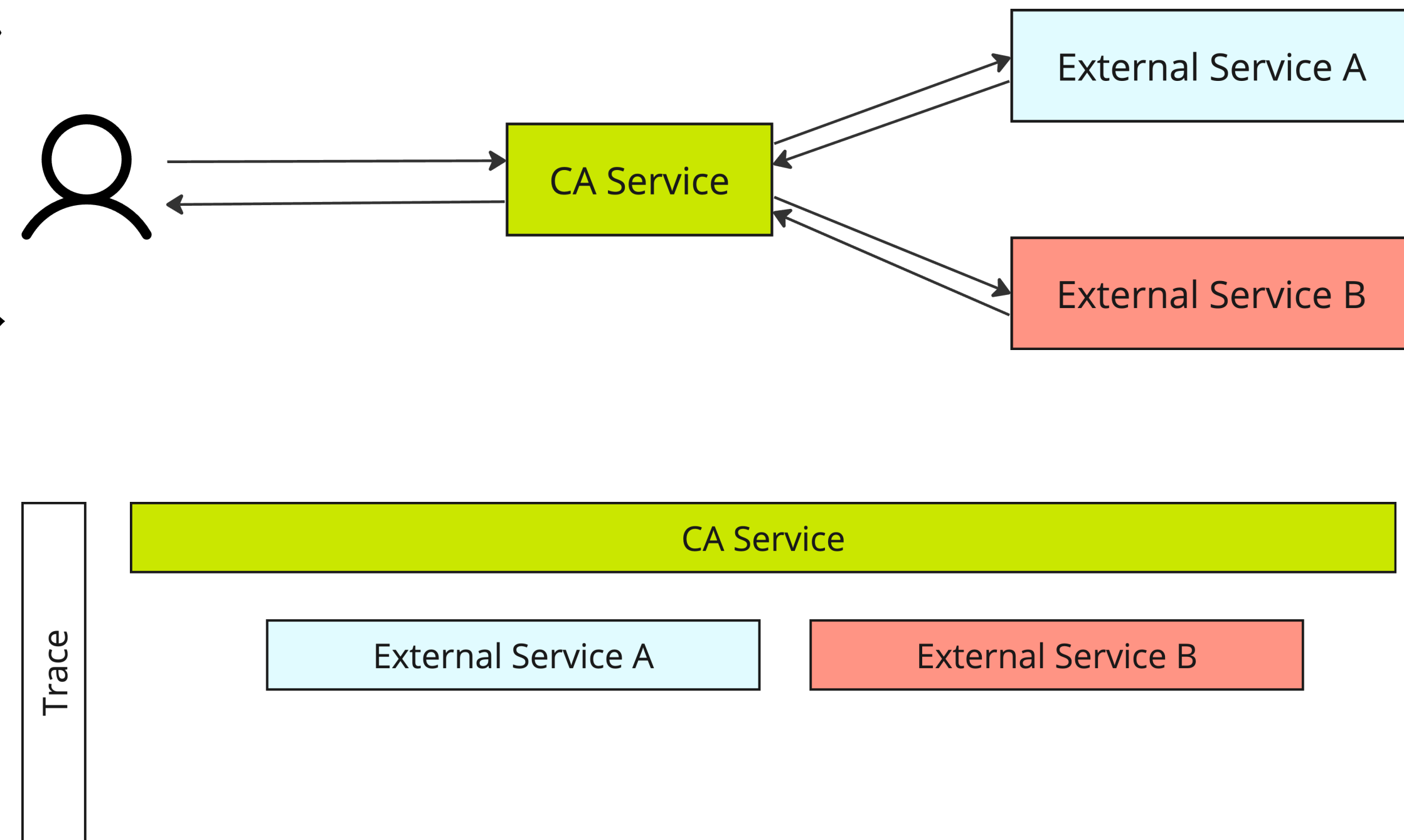
Observability が必要な ユースケース



実際に必要・役立ったケース

アプリ運用カンパニー

- 協業で開発しているサービス内部で別ベンダーのAPIを複数叩く必要があった
- 非機能要件で「自分達で管理している部分でのRPS」があったため、外部APIを除いた時間を計測する必要があった



Only CA Service

= CA Service - (External Service A + External Service B)

ABEMA (FIFA W杯カタル2022)

- マイクロサービスをまたぐ負荷試験をしていたが、分散トレースを入れていなかったためボトルネックを調査するには、各サービスのメトリクスやプロファイラ情報、実際のソースコードから仮説を立てながら調査する必要があったのを改善

導入方法

前提

- Application は Kubernetes (GKE) で稼働
- Anthos Service Mesh (ASM) 導入済み
 - Istio ベースの Service Mesh
 - 基本的に Application の通信は Sidecar として挿入されている istio-proxy (envoy) 経由で送受信される

ABEMA

72

オープニング	LT ① 「ABEMA」のレコメンドへの大規模アクセスを支えるGo製サーバーの裏側	LT ② コメント時のTwitter連携機能で動画がシェアされる際のGo製サーバー開発手法	LT ③ 「ABEMA」の安定稼働を支えたOpenTelemetryの導入事例	ご案内
--------	--	--	--	-----

導入方法

Instrumentation (OpenTelemetry)

LB

HTTP

gRPC request

istio-proxy container

application container

microservice A

microservice B

Cloud Trace

概念図

ABEMA

73

オープニング	LT ① 「ABEMA」のレコメンドへの大規模アクセスを支えるGo製サーバーの裏側	LT ② コメント時のTwitter連携機能で動画がシェアされる際のGo製サーバー開発手法	LT ③ 「ABEMA」の安定稼働を支えたOpenTelemetryの導入事例	ご案内
--------	--	--	--	-----

4

計装

Instrument



CyberAgent®

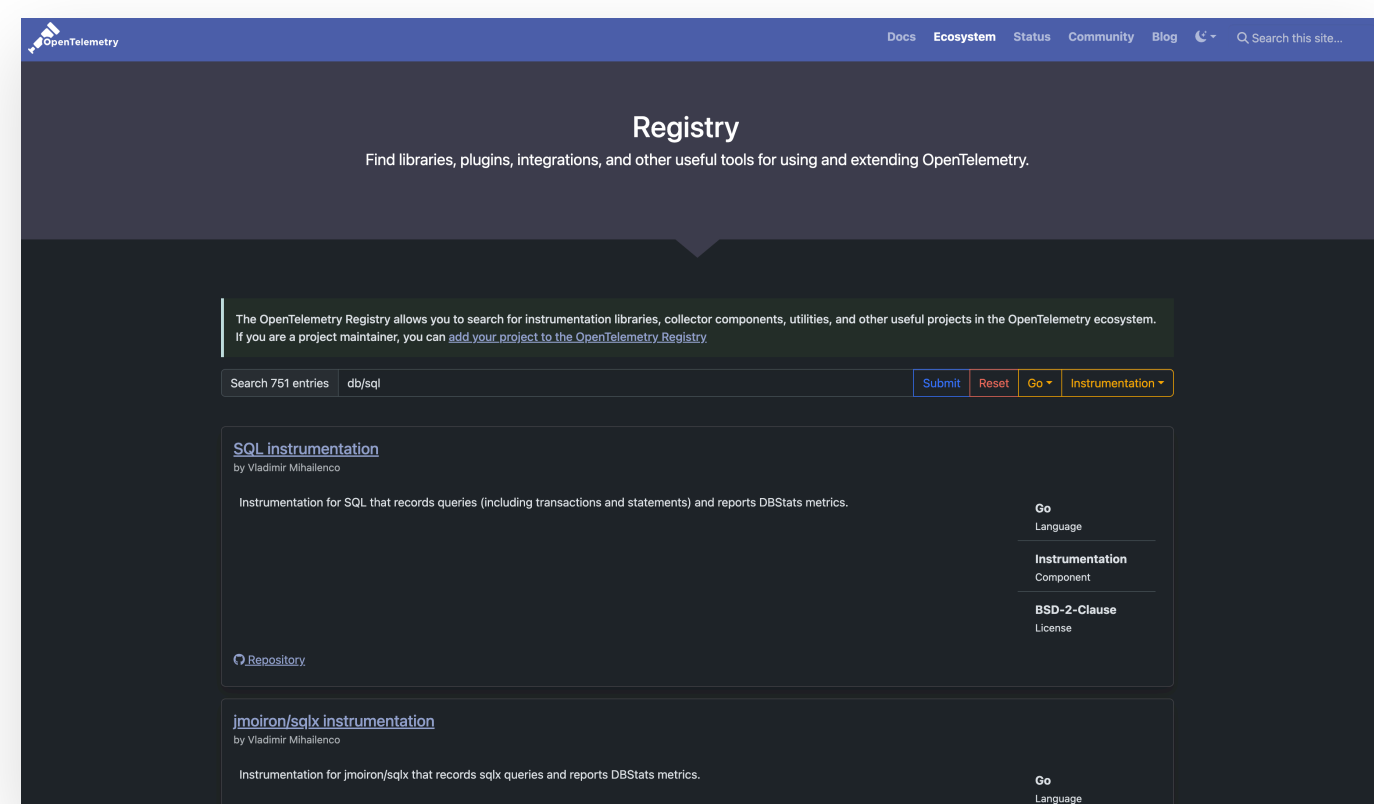
計装 Instrument

- Trace や Span などのシグナルを送るためにロジックを実装すること

手動計装

コードの変更が必要

OTel Registry のように、既存のライブラリを wrap することで計装してくれるものなどもある



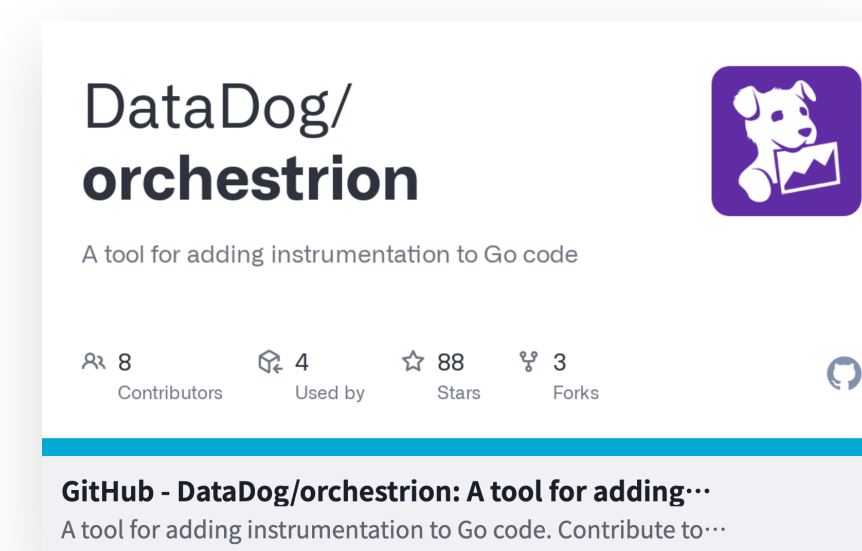
Registry | OpenTelemetry

自動計装

既存のアプリケーションコードを変更することなく計装

eBPF（後述）などを使用

DataDog/orchestrion などの静的解析を使用して自動計装するなどもある



DataDog/orchestrion: A tool for adding instrumentation to Go code

5

Observability を 支える技術



Observability を支える技術

- OpenTelemetry (後で詳細に説明)

ベンダーやツールに依存せず、メトリック信号を作成および管理するために設計されたo11y
フレームワークおよびツールキット

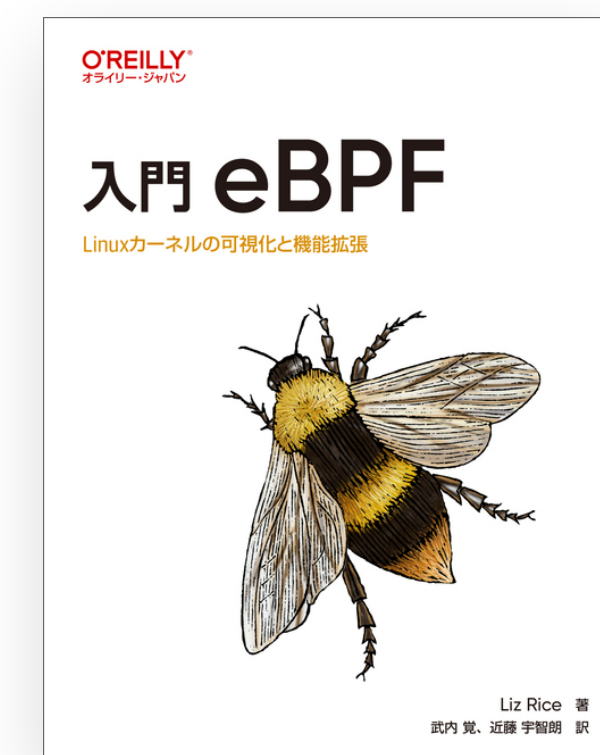
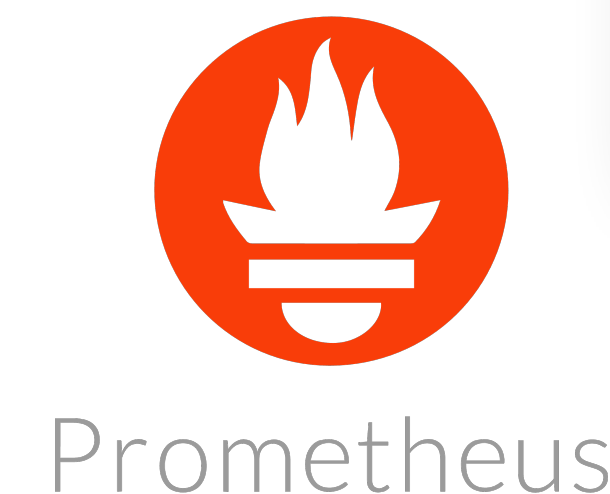
- eBPF

Extend Berkeley Packet Filter

Linux カーネルのソースコード変更やモジュールを追加することなく Linux カーネル内部でプログラム
を実行できる技術

アプリケーション側のコードを変更することなく o11yを計装できるとして注目

- Datadog, Grafana, Jaeger, Prometheus etc...



O'Reilly Japan - 入門 eBPF

6

OpenTelemetry入門



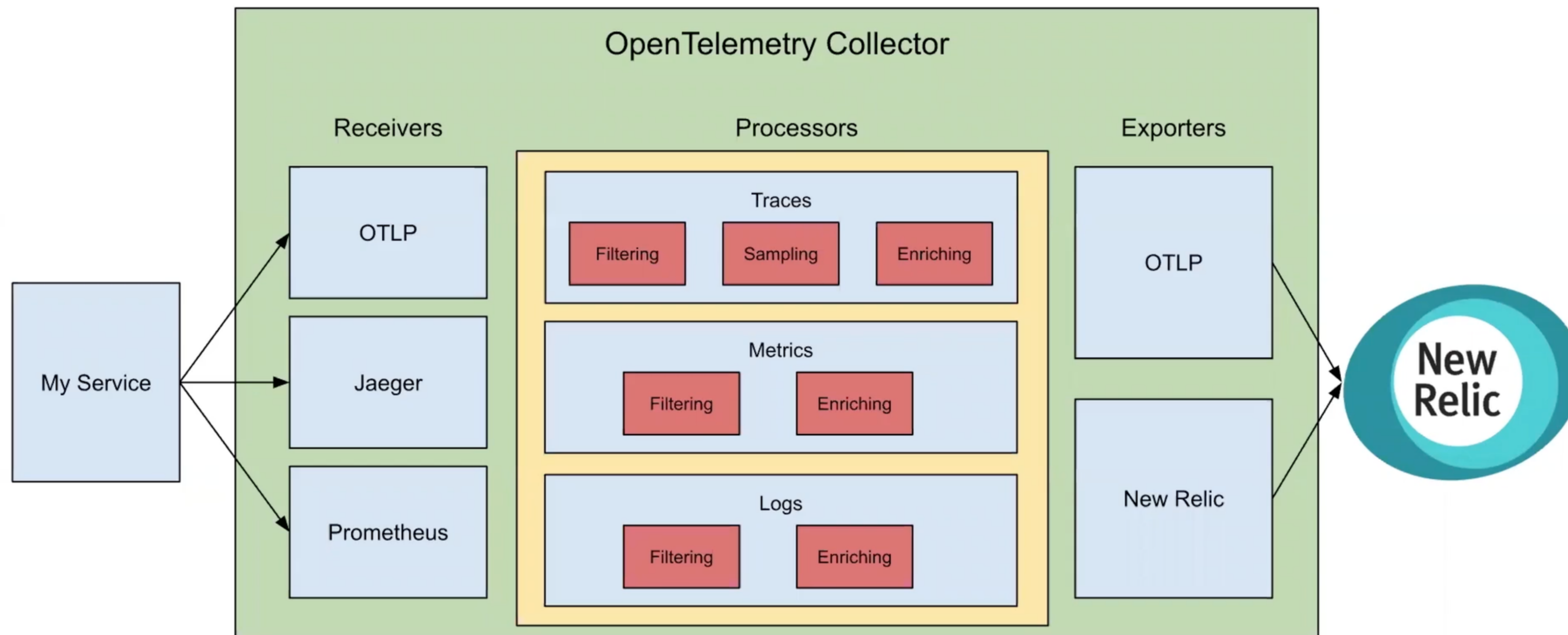
OpenTelemetry Protocol (OTLP)

- Protocol Buffers で定義され gRPC を使用する、テレメトリデータのプロトコル
- ベンダーに依存せずに各サービス間でテレメトリデータをやり取りする際に使用可能

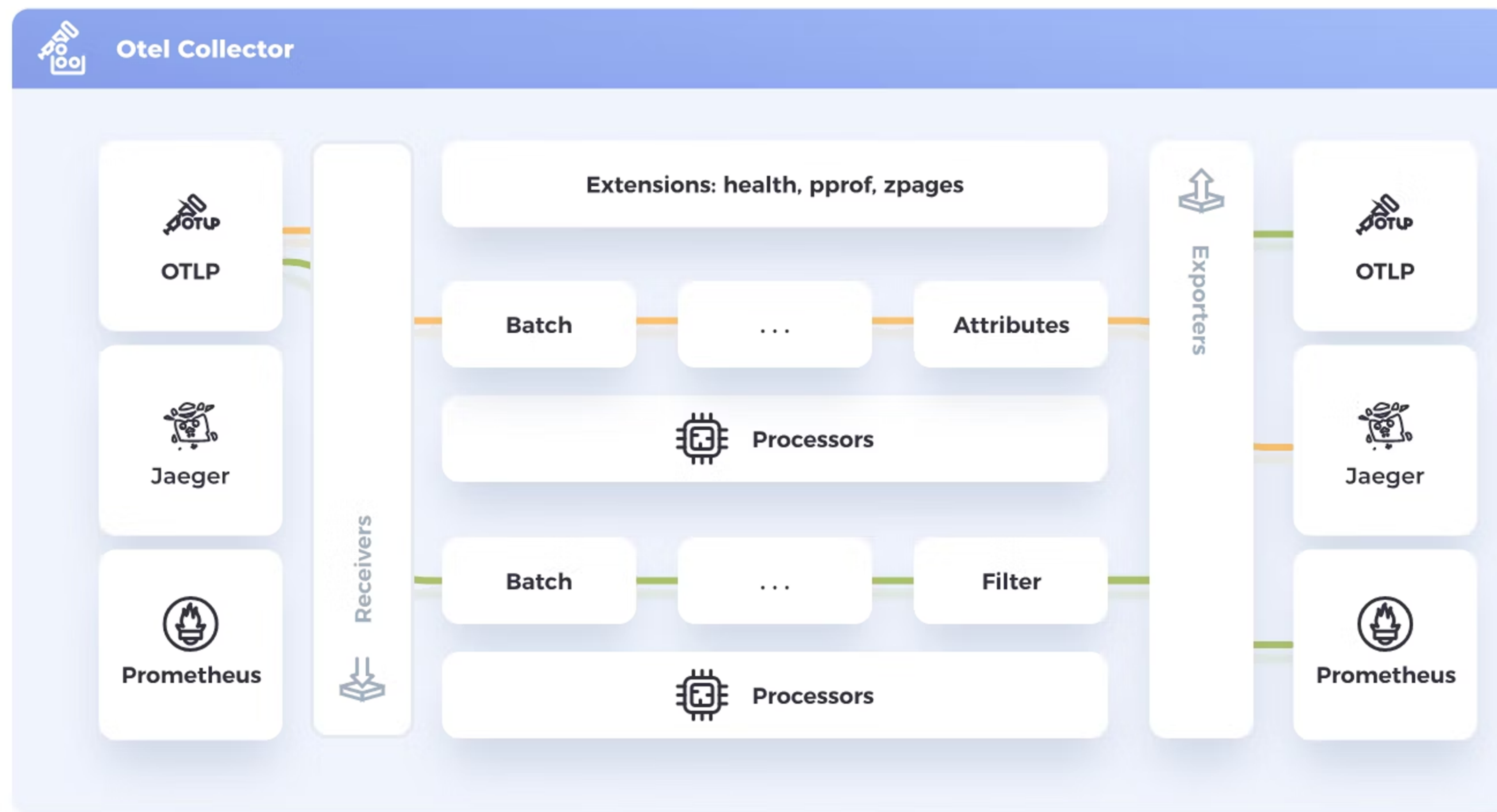


OpenTelemetry Collector

- テレメトリデータ（Signal）を集約する gateway 的な役割をする
- 4つの component から構成

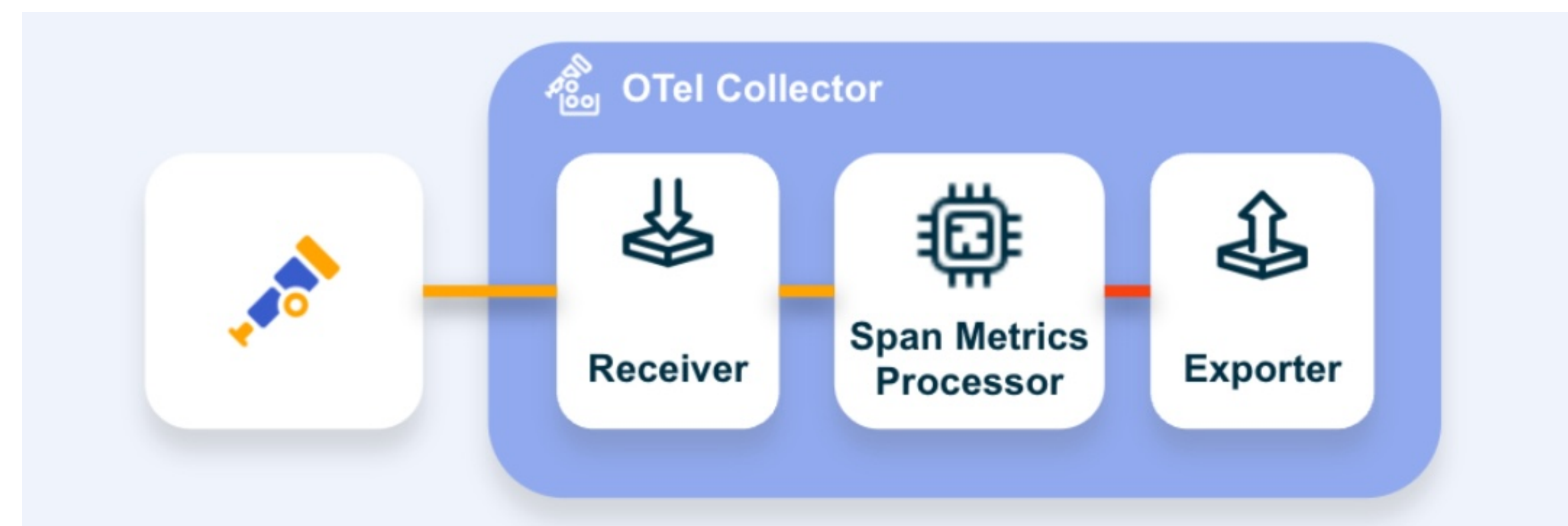
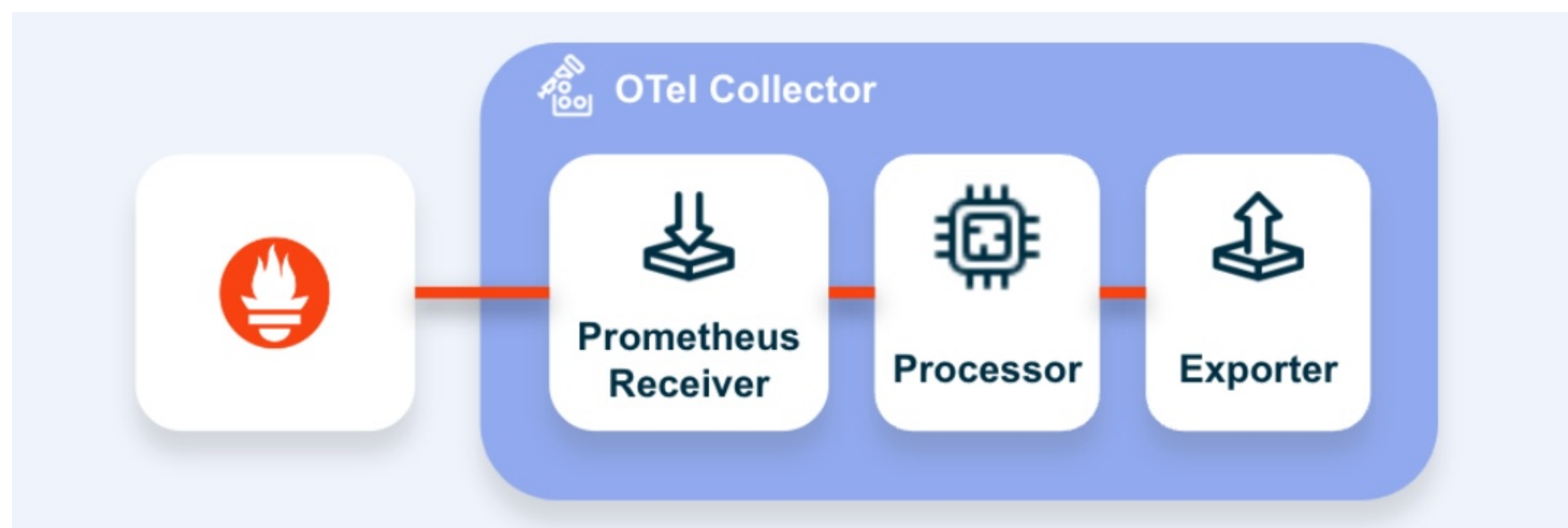


OpenTelemetry Collector



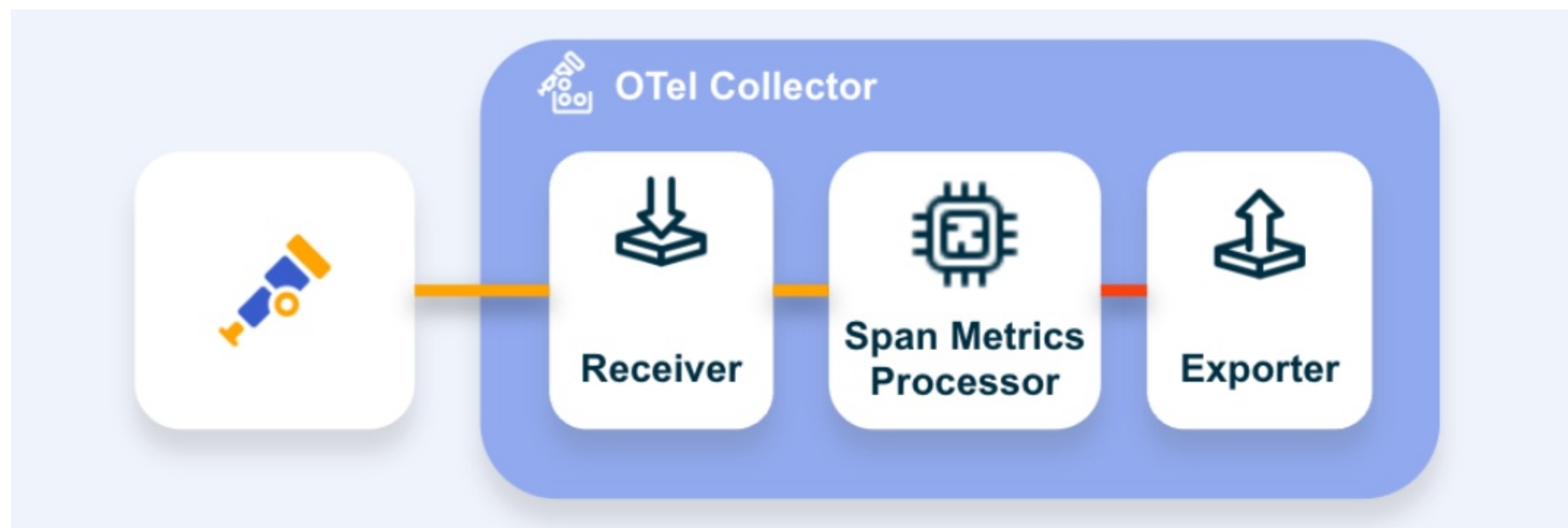
Receiver

- 送信されてきた signal を受け取り変換する
- OTLP はもちろん Prometheus 形式（一部サポート外）にも対応



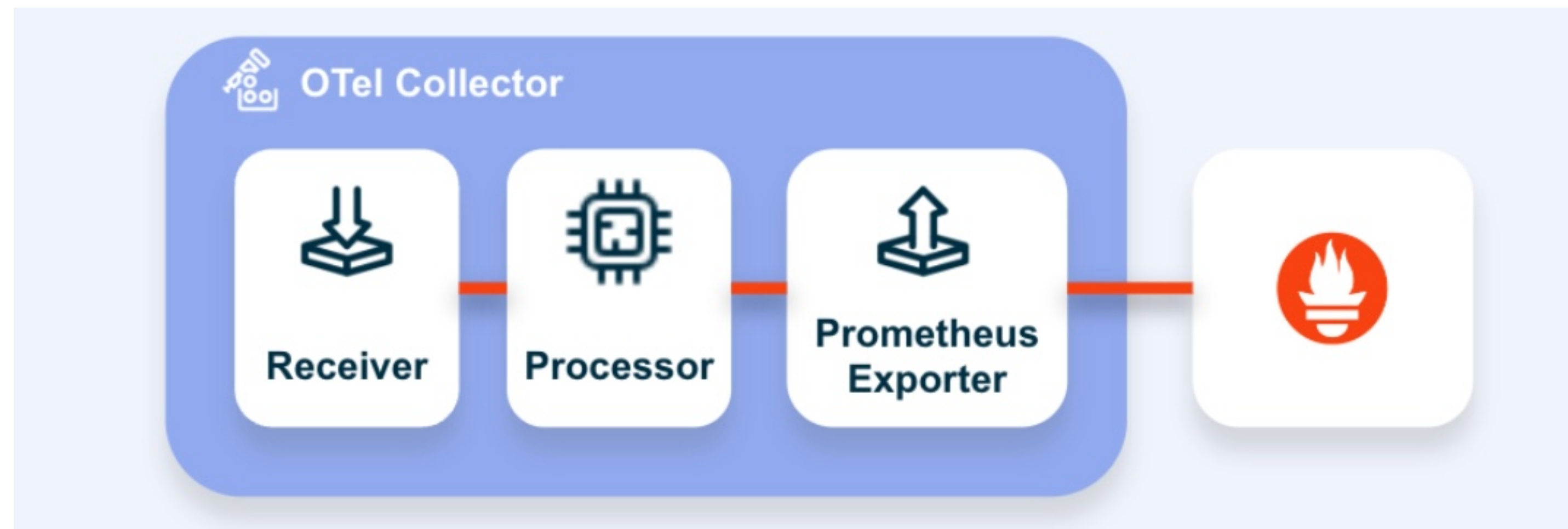
Processor

- フィルタリングやサンプリング、エンリッチングなど Exporter に送信する前に処理を行いデータを変換する
- Datadog でも Datadog Agent 内部や Datadog に取り込んだ後の Grock パーサを用いた Log Pipeline でも 同じようなことができる



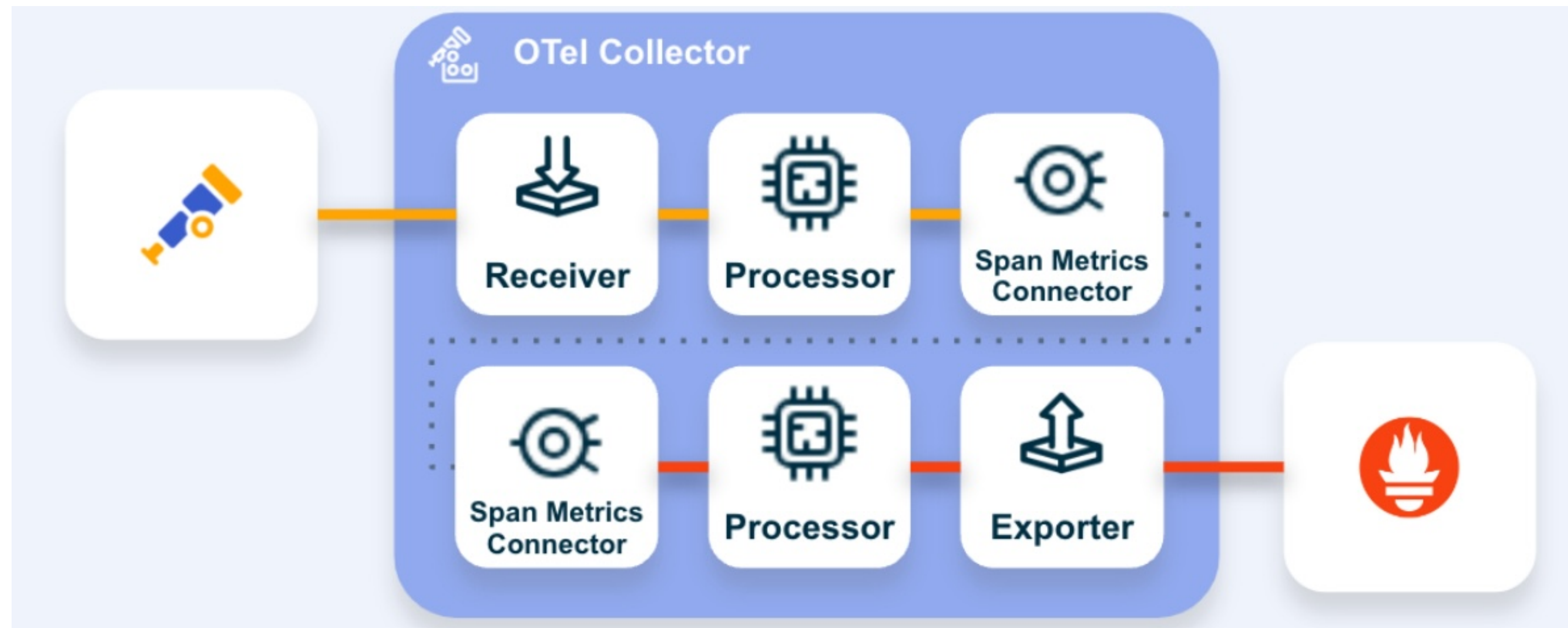
Exporter

- 1つ以上のバックエンドのオブザーバビリティプラットフォームにデータをプッシュ・プル形式で送信



Connector

- 二つのテレメトリパイプラインを繋ぐ
- それぞれの Exporter、Receiver としての機能を持つ
- Connector で扱うデータの型は同一のままでも変換することも可能
- データ型の変換を行う場合はこの component を使用する（Processor で行うのは非推奨）



7

OpenTelemetryを 手動計装してみる



OpenTelemetry x Golang

- OpenTelemetry による計装例です
- ハンズオンの時に実際に計装してもらいます



Tracer

Tracer の作成

- サービスで一つ tracer を初期化

```
import (  
    ...  
    "go.opentelemetry.io/otel/exporters/otlp/otlptrace/  
otlptracehttp"  
    "go.opentelemetry.io/otel/trace"  
    ...  
)  
var (  
    tracer trace.Tracer  
    collectorURL = os.Getenv("OTEL_EXPORTER_OTLP_ENDPOINT")  
)  
  
func Init(ctx context.Context) (exporter, error) {  
    httpExporter, _ := otlptracehttp.New(  
        ctx,  
        otlptracehttp.WithEndpoint(collectorURL),  
        otlptracehttp.WithInsecure(),  
    )  
  
    exporter := httpExporter  
  
    r, _ := newResource()  
    tp := sdktrace.NewTracerProvider(  
        sdktrace.WithResource(r),  
        sdktrace.WithSampler(sdktrace.AlwaysSample()),  
        sdktrace.WithBatcher(exporter),  
    )  
  
    tracer = tp.Tracer(version.Get().ServiceName)  
    otel.SetTracerProvider(tp)  
  
    return exporter, nil  
}
```

Tracer の作成

- **Exporter を作成**

今回のハンズオンでは、sidecar している Datadog Agent に対して送信)

```
import (  
    ...  
    "go.opentelemetry.io/otel/exporters/otlp/otlptrace/  
otlptracehttp"  
    "go.opentelemetry.io/otel/trace"  
    ...  
)  
var (  
    tracer trace.Tracer  
    collectorURL = os.Getenv("OTEL_EXPORTER_OTLP_ENDPOINT")  
)  
  
func Init(ctx context.Context) (exporter, error) {  
    httpExporter, _ := otlptracehttp.New(  
        ctx,  
        otlptracehttp.WithEndpoint(collectorURL),  
        otlptracehttp.WithInsecure(),  
    )  
  
    exporter := httpExporter  
  
    r, _ := newResource()  
    tp := sdktrace.NewTracerProvider(  
        sdktrace.WithResource(r),  
        sdktrace.WithSampler(sdktrace.AlwaysSample()),  
        sdktrace.WithBatcher(exporter),  
    )  
  
    tracer = tp.Tracer(version.Get().ServiceName)  
    otel.SetTracerProvider(tp)  
  
    return exporter, nil  
}
```

Tracer の作成

- Tracer Provider を作成

- OTLP を吐き出す Provider を作成

```
import (  
    ...  
    "go.opentelemetry.io/otel/exporters/otlp/otlptrace/  
otlptracehttp"  
    "go.opentelemetry.io/otel/trace"  
    ...  
)  
var (  
    tracer trace.Tracer  
    collectorURL = os.Getenv("OTEL_EXPORTER_OTLP_ENDPOINT")  
)  
  
func Init(ctx context.Context) (exporter, error) {  
    httpExporter, _ := otlptracehttp.New(  
        ctx,  
        otlptracehttp.WithEndpoint(collectorURL),  
        otlptracehttp.WithInsecure(),  
    )  
  
    exporter := httpExporter  
  
    r, _ := newResource()  
    tp := sdktrace.NewTracerProvider(  
        sdktrace.WithResource(r),  
        sdktrace.WithSampler(sdktrace.AlwaysSample()),  
        sdktrace.WithBatcher(exporter),  
    )  
  
    tracer = tp.Tracer(version.Get().ServiceName)  
    otel.SetTracerProvider(tp)  
  
    return exporter, nil  
}
```


Tracer の作成

- Tracer を作成

```
import (  
    ...  
    "go.opentelemetry.io/otel/exporters/otlp/otlptrace/  
otlptracehttp"  
    "go.opentelemetry.io/otel/trace"  
    ...  
)  
var (  
    tracer trace.Tracer  
    collectorURL = os.Getenv("OTEL_EXPORTER_OTLP_ENDPOINT")  
)  
  
func Init(ctx context.Context) (exporter, error) {  
    httpExporter, _ := otlptracehttp.New(  
        ctx,  
        otlptracehttp.WithEndpoint(collectorURL),  
        otlptracehttp.WithInsecure(),  
    )  
  
    exporter := httpExporter  
  
    r, _ := newResource()  
    tp := sdktrace.NewTracerProvider(  
        sdktrace.WithResource(r),  
        sdktrace.WithSampler(sdktrace.AlwaysSample()),  
        sdktrace.WithBatcher(exporter),  
    )  
  
    tracer = tp.Tracer(version.Get().ServiceName)  
    otel.SetTracerProvider(tp)  
  
    return exporter, nil  
}
```

Span

Span の実装

- Span の開始

Span を入れたい部分にこれを差し込む

```
import (  
    ...  
    "go.opentelemetry.io/otel/trace"  
    ...  
)  
  
var (  
    tracer trace.Tracer  
)  
  
func StartSpan(ctx context.Context, name string) (context.Context, trace.Span) {  
    return tracer.Start(ctx, name)  
}
```

Span の実装

- Span の計装

Span を入れたい部分にこれを差し込む

```
func exampleFunction(ctx context.Context) {  
    // NOTE: OpenTelemetry span  
    ctx, span := tracer.StartSpan(ctx, "exampleFunction")  
    defer span.End()  
    ...  
}
```

Span をあらゆる場所に
計装していくの大変すぎでは？

Registry

- 公式やサードパーティが色々なライブラリをラップして計装してくれているものを探せる
- ここにあった場合、いつも使っているものをこれに置き換えると勝手にいい感じに計装してくれる

The screenshot shows the OpenTelemetry Registry website. At the top, there is a navigation bar with links for Docs, Ecosystem, Status, Community, and Blog, along with a search bar. The main heading is "Registry" with a subtitle: "Find libraries, plugins, integrations, and other useful tools for using and extending OpenTelemetry." Below this, a search bar shows "net/http" with 751 entries found. Two search results are visible:

- splunkhttp – Instrumentation for net/http** by Splunk Inc.
Description: Splunk specific instrumentation for the Golang `net/http` package.
Metadata: Go Language, Instrumentation Component, Apache 2.0 License. Includes a "Repository" link.
- Go package net/http instrumentation** by OpenTelemetry Authors
Description: Package `http` provides a `http.Handler` and functions that are intended to be used to add tracing by wrapping existing handlers (with `Handler`) and routes `WithRouteTag`.
Metadata: Go Language, Instrumentation Component.

Registry

- 公式が対応しているPackage

ginとかechoとかnet/httpとかある

Instrumentation Packages

The [OpenTelemetry registry](#) is the best place to discover instrumentation packages. It will include packages outside of this project.

The following instrumentation packages are provided for popular Go packages and use-cases.

Instrumentation Package	Metrics	Traces
github.com/aws/aws-sdk-go-v2		✓
github.com/emicklei/go-restful		✓
github.com/gin-gonic/gin		✓
github.com/gorilla/mux		✓
github.com/labstack/echo		✓
go.mongodb.org/mongo-driver		✓
google.golang.org/grpc	✓	✓
gopkg.in/macaron.v1		✓
host	✓	
net/http	✓	✓
net/http/httptrace		✓
runtime	✓	

Registry

- 例：net/http

Span を入れたい部分にこれを差し込む

Transport に OTel のライブラリを噛ませる

```
import (  
    ...  
    "net/http"  
    "go.opentelemetry.io/contrib/instrumentation/net/http/otelhttp"  
    ...  
)  
...  
httpClient := &http.Client{  
    // NOTE: OpenTelemetry instrumentation  
    Transport: otelhttp.NewTransport(http.DefaultTransport),  
}  
  
req, _ := http.NewRequestWithContext(ctx, http.MethodGet, "https://httpbin.org/get", nil)  
resp, _ := h.httpClient.Do(req)  
defer resp.Body.Close()  
...
```

Registry

- 例：net/http

Span を入れたい部分にこれを差し込む

Transporter に OTel のライブラリを噛ませる

```
import (  
    ...  
    "net/http"  
    "go.opentelemetry.io/contrib/instrumentation/net/http/otelhttp"  
    ...  
)  
...  
httpClient := &http.Client{  
    // NOTE: OpenTelemetry instrumentation  
    Transport: otelhttp.NewTransport(http.DefaultTransport),  
}  
  
req, _ := http.NewRequestWithContext(ctx, http.MethodGet, "https://httpbin.org/get", nil)  
resp, _ := h.httpClient.Do(req)  
defer resp.Body.Close()  
...
```

番外編 Datadog

- 実は Datadog にも同じようなものがある
Datadog の方が多く対応していそう
OSSなのでコントリビュートもできる

インテグレーション

フレームワークの互換性

次のヘルパーパッケージのいずれかを使用して、Go トレーサーと次のリストのウェブフレームワークを統合します。

注: インテグレーションドキュメントは、サポートされているパッケージとその API の詳細な概要を使用例とともに提供します。

フレームワーク	サポートの種類	GODOC DATADOG ドキュメント
Gin	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/gin-gonic/gin
Gorilla Mux	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/gorilla/mux
gRPC	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/google.golang.org/grpc
gRPC v1.2	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/google.golang.org/grpc.v12
chi	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/go-chi/chi
echo v4	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/labstack/echo.v4
echo v3	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/labstack/echo
Fiber	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/gofiber/fiber.v2

ライブラリの互換性

Go トレーサーには、次のデータストアとライブラリのサポートが含まれています。

ライブラリ	サポートの種類	例とドキュメント
AWS SDK	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/aws/aws-sdk-go/aws
AWS SDK v2	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/aws/aws-sdk-go-v2/aws
Elasticsearch	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/olivere/elastic
Cassandra	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/gocql/gocql
GraphQL	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/graph-gophers/graphql-go
HTTP	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/net/http
HTTP router	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/julienschmidt/httprouter
Redis (go-redis)	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/go-redis/redis
Redis (go-redis-v8)	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/go-redis/redis.v8
Redis (redigo)	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/garyburd/redigo
Redis (new redigo)	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/gomodule/redigo
SQL	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/database/sql
SQLx	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/jmoiron/sqlx
MongoDB	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/go.mongodb.org/mongo-driver/mongo
[MongoDB (mgo)]73	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/globalsign/mgo
BuntDB	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/tidwall/buntdb
LevelDB	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/syndtr/goleveldb/leveldb
miekg/dns	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/miekg/dns
Kafka (confluent)	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/confluentinc/confluent-kafka-go
Kafka (sarama)	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/Shopify/sarama
Google API	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/google.golang.org/api
go-restful	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/emicklei/go-restful
Twirp	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/twitchtv/twirp
Vault	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/hashicorp/vault
Consul	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/hashicorp/consul
Gorm	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/jinzhu/gorm
Gorm v2	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/gorm.io/gorm.v1
Kubernetes	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/k8s.io/client-go/kubernetes
Memcache	完全対応	gopkg.in/DataDog/dd-trace-go.v1/contrib/bradfitz/gomemcache/memcache

8

Datadog 入門



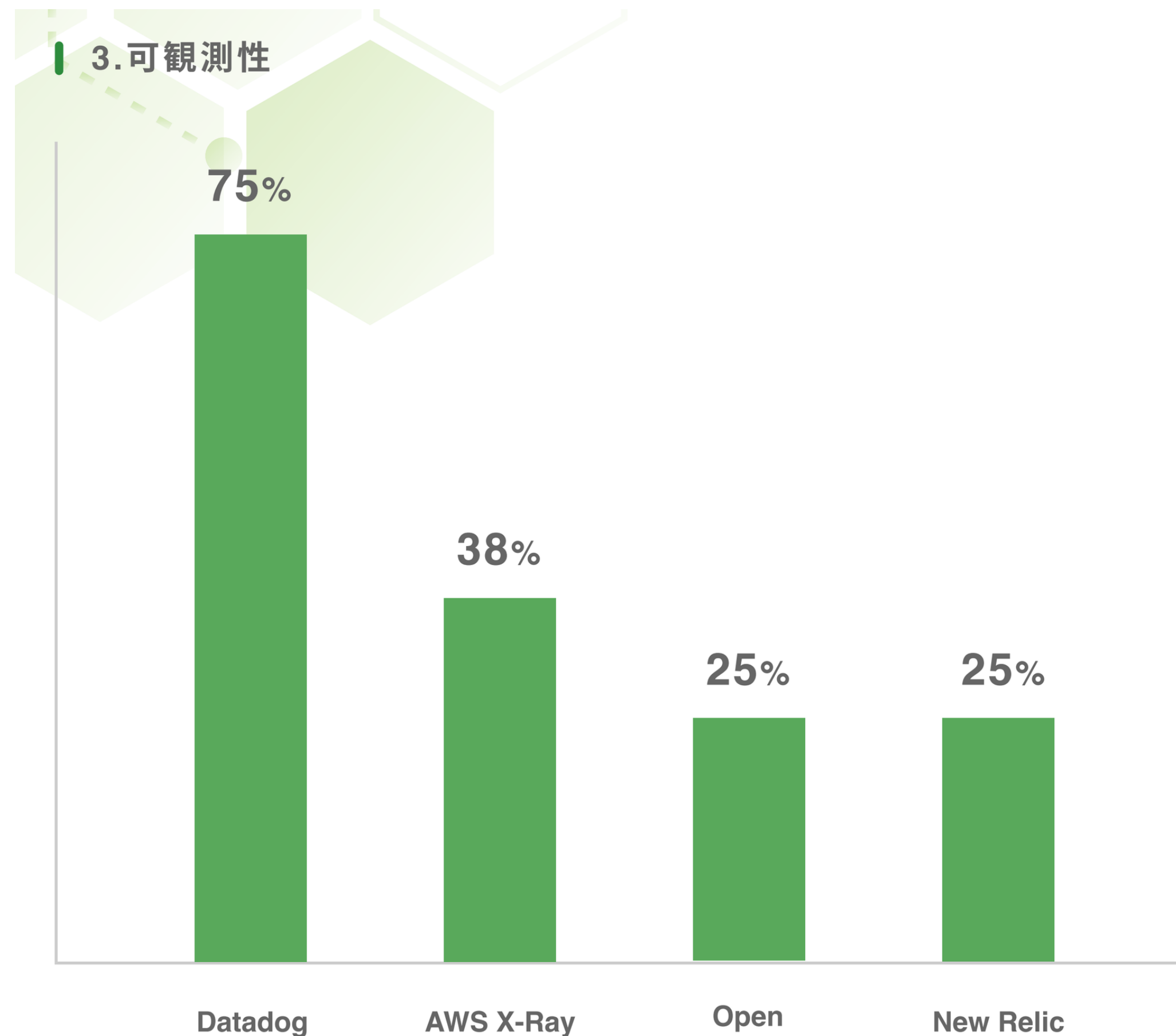
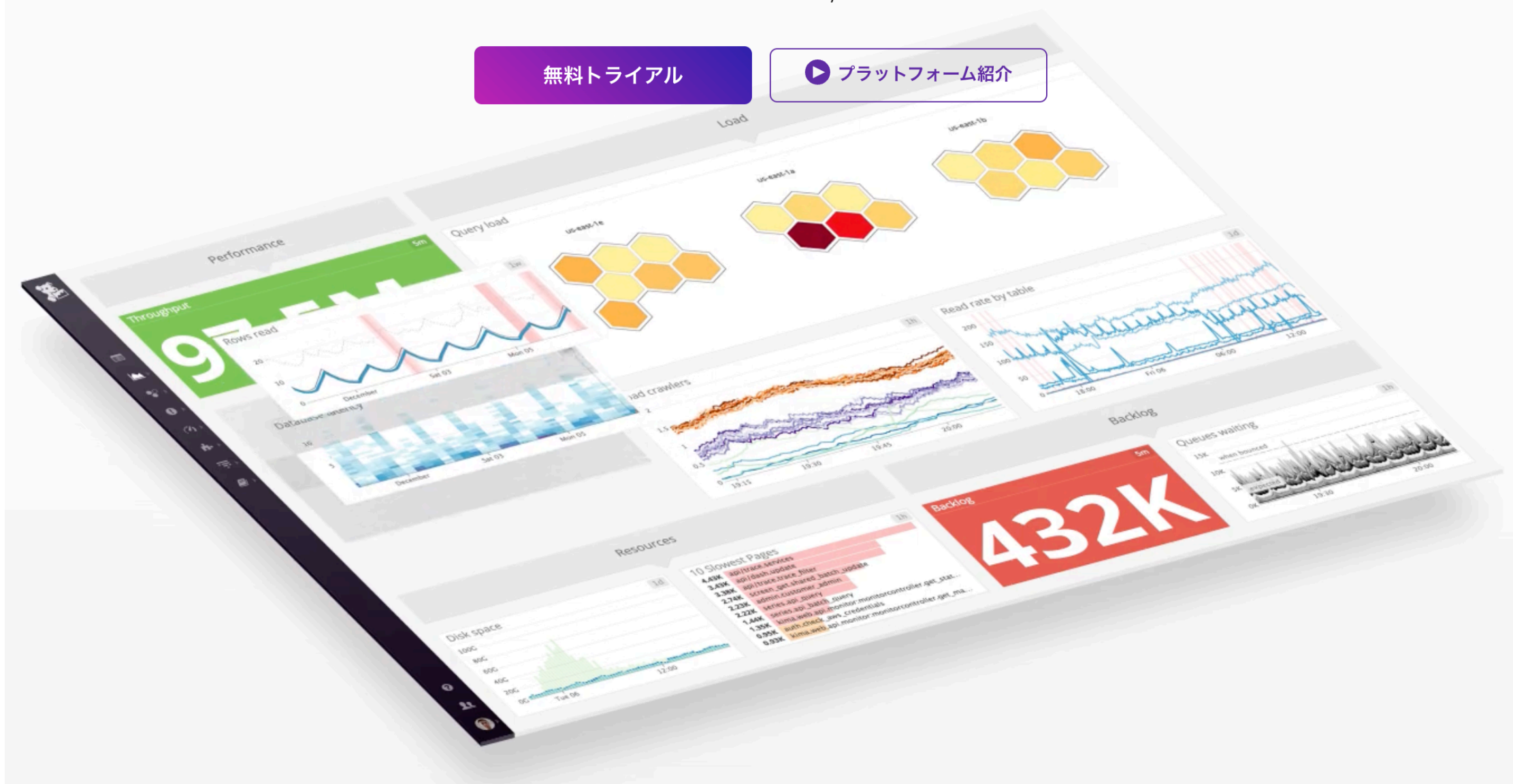
Datadog

- モニタリング SaaS の一つ
- 社内だと一番使われてそう

AI事業本部でもかなり採用されている印象

モダンなモニタリングとセキュリティ

場所や規模に関係なく、あらゆるスタック/アプリケーションの内部を監視



Datadog

- クラウドベンダーをはじめ、大体のサービスの Integration に対応している
なくとも作れる
- 高度な Log 検索やグラフ描写ができる

The screenshot shows the Datadog documentation website for integrations. The main content area is titled 'インテグレーション' (Integrations) and includes a search bar and a grid of integration cards. The sidebar on the left contains a navigation menu with categories like '重要な情報' (Important Information), 'はじめに' (Getting Started), '用語集' (Glossary), and 'インテグレーション' (Integrations). The integration cards are organized into categories such as 'MARKETPLACE', 'SERVICE MANAGEMENT', 'INFRASTRUCTURE', 'APPLICATION PERFORMANCE', 'SOFTWARE DELIVERY', 'LOG MANAGEMENT', 'SECURITY', and 'DIGITAL EXPERIENCE'.

Datadog

- 詳しい使い方が知りたければ Datadog Learning Center がおすすめ

2週間ほど有効な Datadog アカウントが付与

(期限を過ぎても再度アクセスすればまた使える)

instruqt というラーニングツールを用いて実際に vm を立てて、そこに対して Agent を入れたり、WEBアプリのログを吐かせたり、はたまた動いているWEBアプリに対して Synthetic Testing を実行したり、実際に RUM を入れて見たりと一通りの Datadog の機能を試すことができる

- Datadog の一通りの使い方はこれでわかる

Datadog 認定資格 (Datadog Fundamentals) を取ってみた



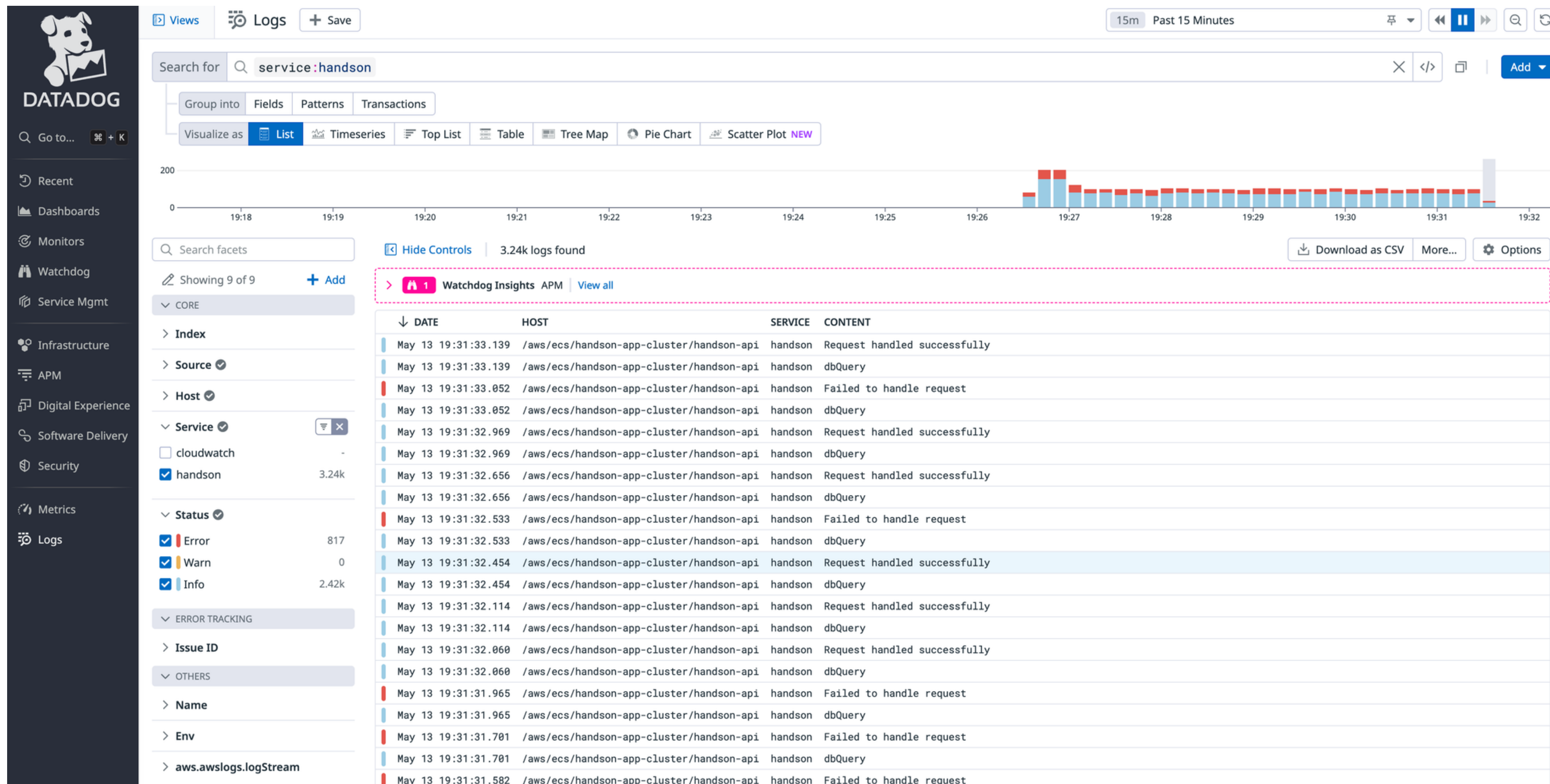
Shota Iwami



機能の軽い紹介

Log

- CloudWatch Logs や Datadog Agent など経由で送った Log を見れる
- Log を構造化すると、facets など key で絞り込めて便利



The screenshot displays the Datadog Logs interface. The search bar contains 'service:handson'. The left sidebar shows the navigation menu with 'Logs' selected. The main area shows a search bar with 'service:handson' and a search button. Below the search bar, there are tabs for 'Group into' (Fields, Patterns, Transactions) and 'Visualize as' (List, Timeseries, Top List, Table, Tree Map, Pie Chart, Scatter Plot). A bar chart shows the log volume over time, with a peak around 19:27. Below the chart, there are buttons for 'Download as CSV', 'More...', and 'Options'. The log entries are displayed in a table with columns for DATE, HOST, SERVICE, and CONTENT. The table shows a mix of successful requests and failed requests.

DATE	HOST	SERVICE	CONTENT
May 13 19:31:33.139	/aws/ecs/handson-app-cluster/handson-api	handson	Request handled successfully
May 13 19:31:33.139	/aws/ecs/handson-app-cluster/handson-api	handson	dbQuery
May 13 19:31:33.052	/aws/ecs/handson-app-cluster/handson-api	handson	Failed to handle request
May 13 19:31:33.052	/aws/ecs/handson-app-cluster/handson-api	handson	dbQuery
May 13 19:31:32.969	/aws/ecs/handson-app-cluster/handson-api	handson	Request handled successfully
May 13 19:31:32.969	/aws/ecs/handson-app-cluster/handson-api	handson	dbQuery
May 13 19:31:32.656	/aws/ecs/handson-app-cluster/handson-api	handson	Request handled successfully
May 13 19:31:32.656	/aws/ecs/handson-app-cluster/handson-api	handson	dbQuery
May 13 19:31:32.533	/aws/ecs/handson-app-cluster/handson-api	handson	Failed to handle request
May 13 19:31:32.533	/aws/ecs/handson-app-cluster/handson-api	handson	dbQuery
May 13 19:31:32.454	/aws/ecs/handson-app-cluster/handson-api	handson	Request handled successfully
May 13 19:31:32.454	/aws/ecs/handson-app-cluster/handson-api	handson	dbQuery
May 13 19:31:32.114	/aws/ecs/handson-app-cluster/handson-api	handson	Request handled successfully
May 13 19:31:32.114	/aws/ecs/handson-app-cluster/handson-api	handson	dbQuery
May 13 19:31:32.060	/aws/ecs/handson-app-cluster/handson-api	handson	Request handled successfully
May 13 19:31:32.060	/aws/ecs/handson-app-cluster/handson-api	handson	dbQuery
May 13 19:31:31.965	/aws/ecs/handson-app-cluster/handson-api	handson	Failed to handle request
May 13 19:31:31.965	/aws/ecs/handson-app-cluster/handson-api	handson	dbQuery
May 13 19:31:31.701	/aws/ecs/handson-app-cluster/handson-api	handson	Failed to handle request
May 13 19:31:31.701	/aws/ecs/handson-app-cluster/handson-api	handson	dbQuery
May 13 19:31:31.582	/aws/ecs/handson-app-cluster/handson-api	handson	Failed to handle request

Log

- スタックトレースなども出して
おくといい感じに表示してくれる

ERROR May 13, 2024 at 7:31:33.052 pm (5 minutes ago) View in Context

HOST /aws/ecs/handson-app-cluster/hands... **SERVICE** handson **SOURCE** cloudwatch

ALL TAGS
env:dev component:handson-api-infra datadog.index:main datadog.submission_auth:api_key forwarder_memorysize:1024
forwarder_version:3.110.0 forwardername:datadogintegration-forwarderstack-seh6ro-forwarder-qoypow8z4hsy
name:ecs-handson-app-cluster-handson-api source:cloudwatch sourcecategory:aws terraform:true

Failed to handle request

Event Attributes **Trace 1** Metrics Processes

ERROR KIND
handson-api-error

Error

Pretty Raw

handson-api-error: random error

```
github.com/BIwashi/otel-handson/backend/app/api/handson.(*Handler).randomError  
github.com/BIwashi/otel-handson/backend/app/api/handson/handson.go:135  
  
github.com/BIwashi/otel-handson/backend/app/api/handson.(*Handler).handson  
github.com/BIwashi/otel-handson/backend/app/api/handson/handson.go:54  
  
github.com/BIwashi/otel-handson/backend/app/api/handson.(*Handler).Register.(*Handler).loggingMiddleware.func1  
github.com/BIwashi/otel-handson/backend/app/api/handson/handler.go:32
```

Show More

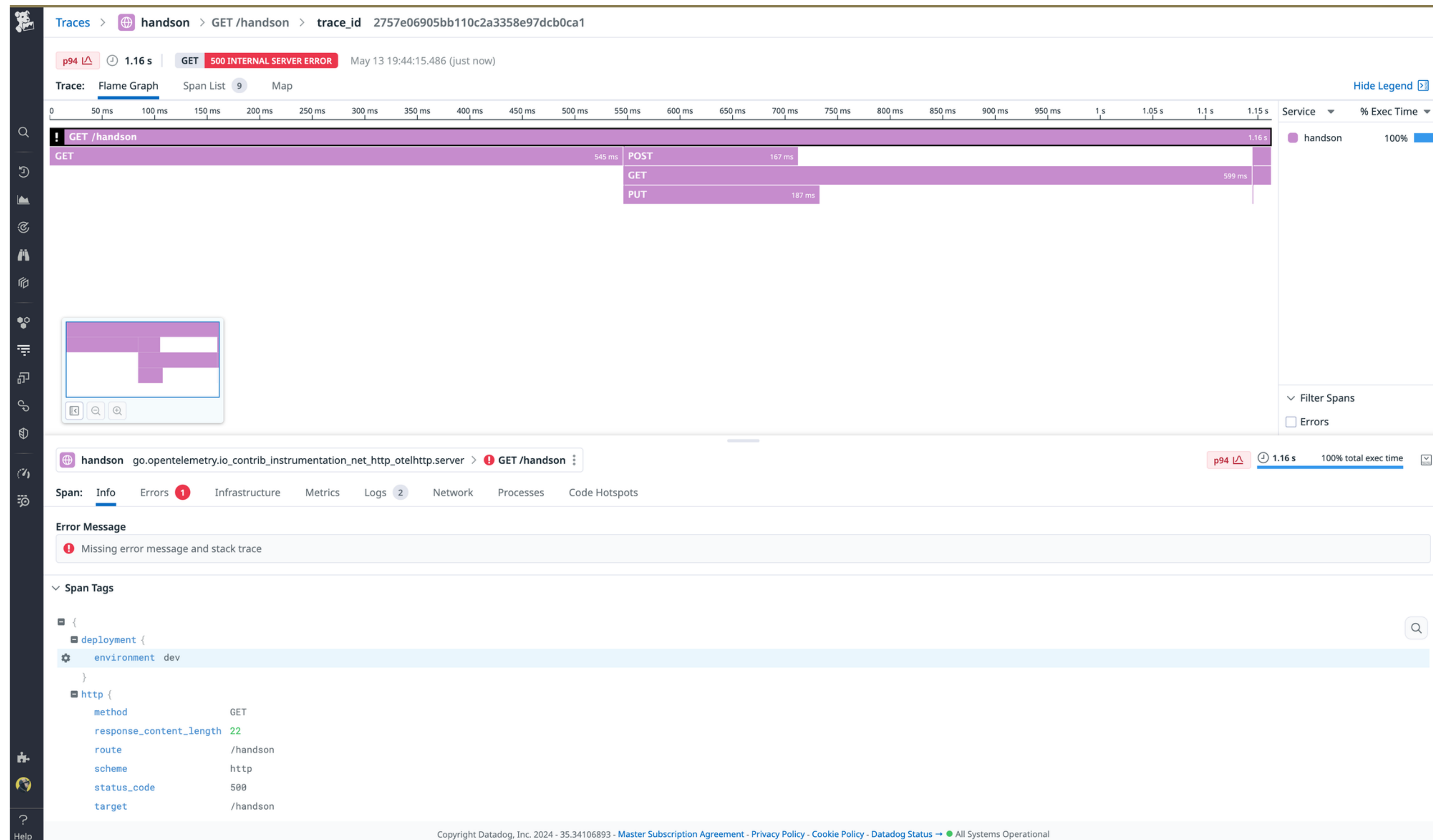
```
{  
  aws: {  
    awslogs: {  
      logGroup: /aws-logs-2024-05-13-13:00:00-000000000000-us-east-1  
      logStreamName: /aws-logs-2024-05-13-13:00:00-000000000000-us-east-1/2024-05-13T07:31:33.052Z  
      source: /aws-logs-2024-05-13-13:00:00-000000000000-us-east-1/2024-05-13T07:31:33.052Z  
    }  
  }  
}
```

APM Trace

- APM = Application Performance Monitoring

- Traceを見れる

これを後で作ってもらいます

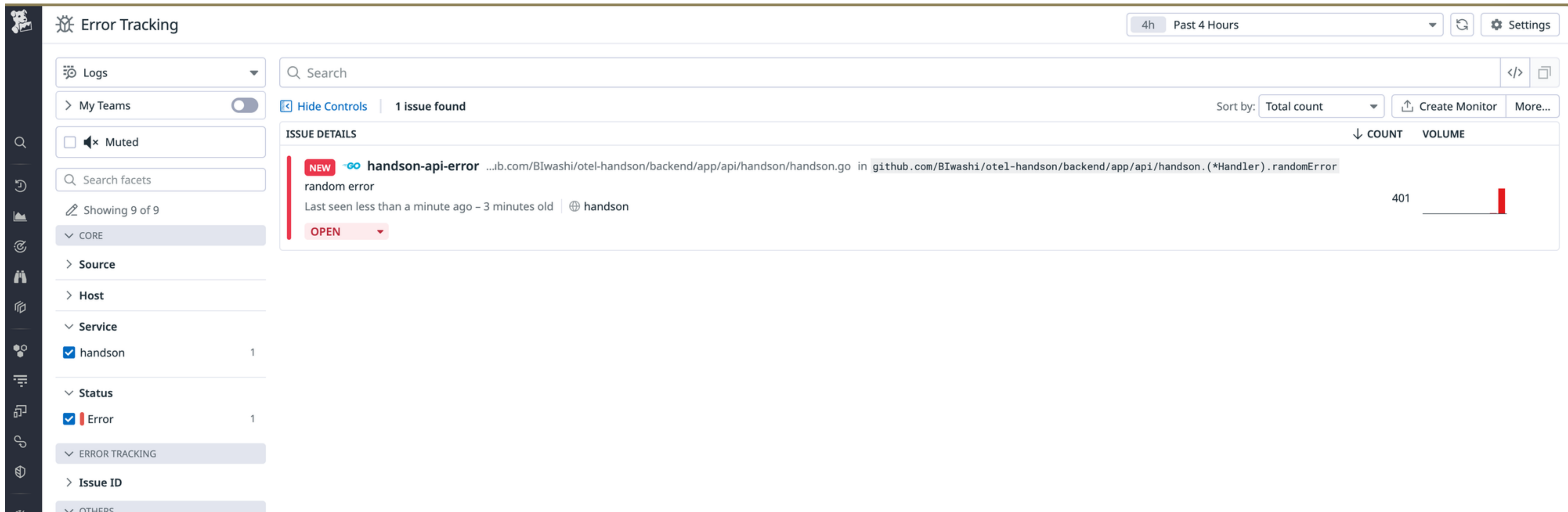


APM Error Tracking

- 同じエラーをまとめて集約してくれる

特定の Error を無視したりもできる

Error Log をまとめてくれるので、全体の Error を俯瞰して見れる



The screenshot displays the APM Error Tracking dashboard. The interface includes a sidebar with navigation options like 'Logs', 'My Teams', and 'Muted'. The main area shows a search bar and a list of error issues. The top right corner has a time filter set to '4h Past 4 Hours' and a 'Settings' button. The search bar contains the text 'Search'. Below the search bar, there are controls for 'Hide Controls' and '1 issue found'. The 'Sort by' dropdown is set to 'Total count'. There are buttons for 'Create Monitor' and 'More...'. The 'ISSUE DETAILS' section shows a single issue with the following information:

- NEW** [handson-api-error](#) ...ib.com/BIwashi/otel-handson/backend/app/api/handson/handson.go in [github.com/BIwashi/otel-handson/backend/app/api/handson.\(*Handler\).randomError](#)
- random error
- Last seen less than a minute ago - 3 minutes old | [handson](#)
- OPEN**

The issue is associated with a count of 401 and a volume bar chart.

APM Error Tracking

- Error に **error.message / error.kind / error.stack** を入れると認識

4 Format and send errors

To properly enable Error Tracking ensure that you use the following attribute names when logging an error: `error.stack`, `error.message`, `error.kind`.

[Learn more about Error Tracking attributes](#) 

Logrus

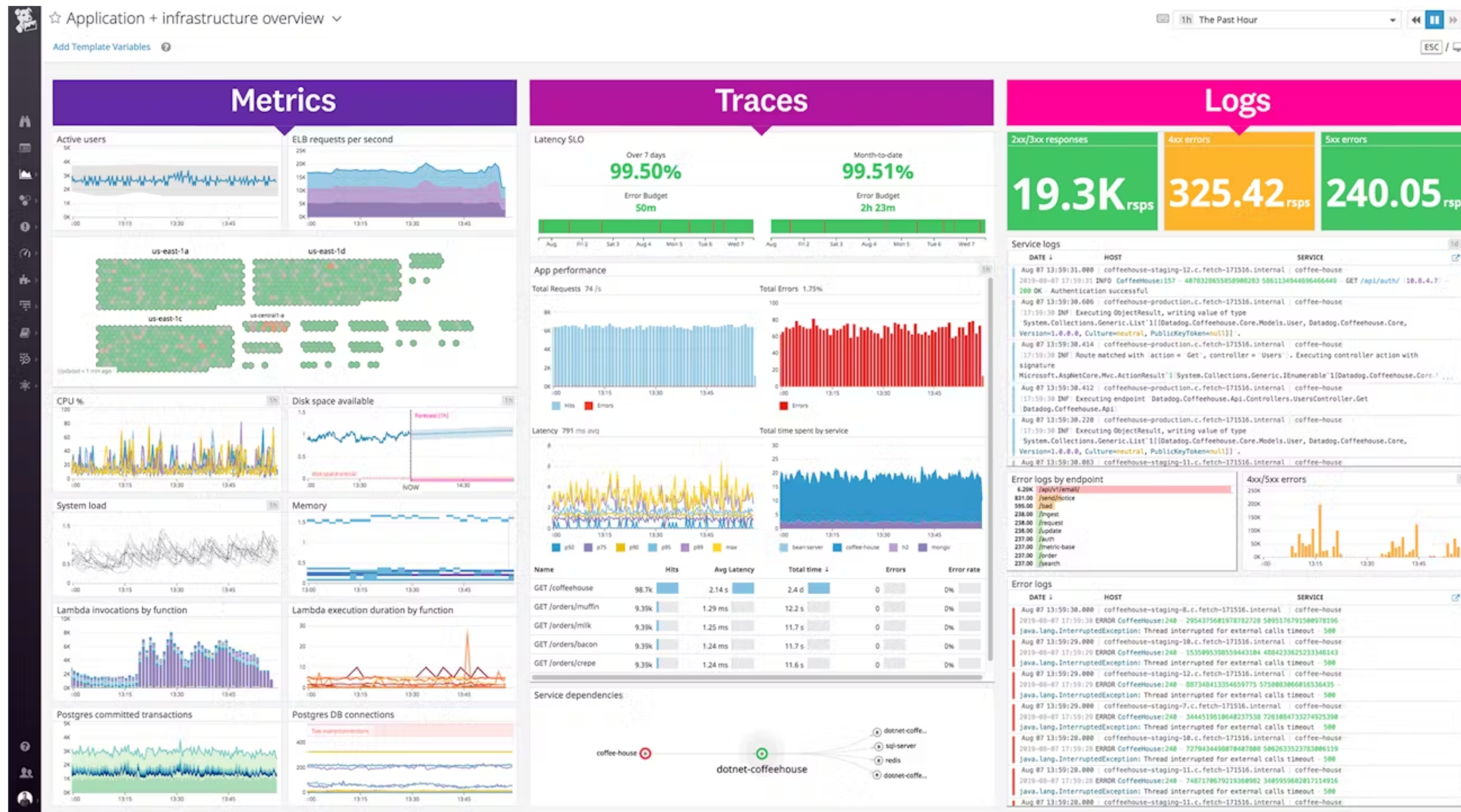
```
// for https://github.com/pkg/errors
type stackTracer interface {
    StackTrace() errors.StackTrace
}

type errorField struct {
    Kind    string `json:"kind"`
    Stack   string `json:"stack"`
    Message string `json:"message"`
}

func ErrorField(err error) errorField {
    var stack string
    if serr, ok := err.(stackTracer); ok {
        st := serr.StackTrace()
        stack = fmt.Sprintf("%+v", st)
        if len(stack) > 0 && stack[0] == '\n' {
```


Dashboards

- ログやメトリクス、トレースなどをグラフとして描画できる



Monitors

- ログやメトリクスなどの閾値を元に slack などにアラートを飛ばせる
- On-call などにもトリガーできる

The screenshot shows the Datadog Monitors configuration page for an alert titled "High number of errors in APM Traces". The alert is currently in an "OK" state. The configuration details are as follows:

- Properties:**
 - Monitor ID:** 144820378
 - Created at:** May 13, 2024, 8:12 pm
 - Created by:** 影太 岩見
- QUERY:** `error-tracking-traces("service:handson").rollup("count").by("@issue.id").last("5m") > 1`
- RECIPIENT:** slack-ai_kenshu24_o11y_handson
- MESSAGE:** High number of errors on [issue]({{ issue.link }}) detected.
 `{{#is_alert}}`
 `{{span.error.type}}: {{span.attributes.error.message}}`
- TAGS:** (None)
- TEAMS:** (None)
- PRIORITY:** Not Defined

Below the configuration, there is a "Status & History" section with a graph showing the monitor's status over time. The graph currently shows "no groups found".

The screenshot shows a Slack notification message from the Datadog app, triggered at 20:16. The message content is as follows:

Datadog アプリ 20:16

Triggered: [TEST] High number of errors in APM Traces on @issue.id:N/A

High number of errors on [issue]() detected.

:

@slack-AI_Business_Unit-ai_kenshu24_o11y_handson

Test notification triggered by [kwami_shota@cyberagent.co.jp](#).

The count of trace errors matching service:handson, grouped by @issue.id, was > 1 during the last 5m.

Tags

@issue.id:N/A

Notified

@slack-AI_Business_Unit-ai_kenshu24_o11y_handson

Monitors

- 適切なアラートをしようという話

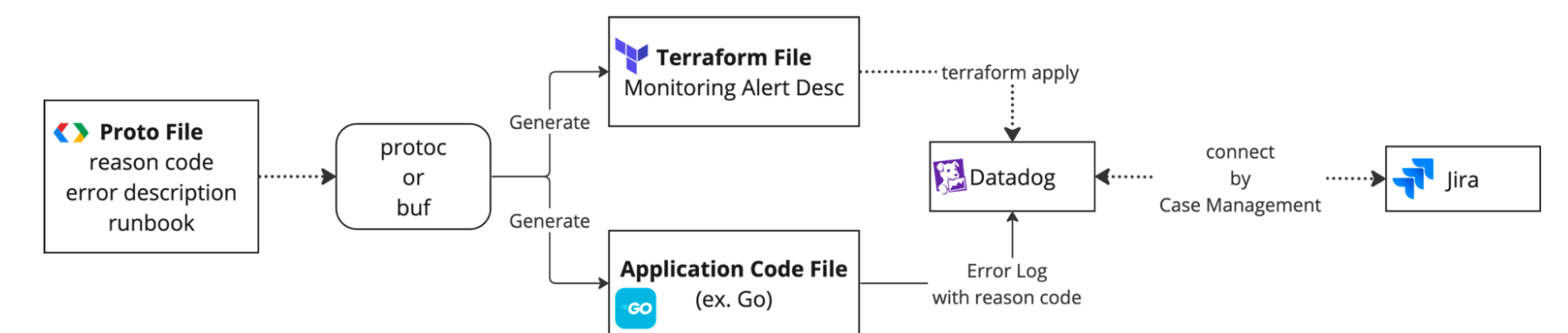
is_match によるMessage分岐

- Datadog のMessageでは **log.attributes** で構造化されたログのフィールドの値を取得できる
- error に埋め込んだ **reason_codes** を is_match で検知して、message(runbook)を表示する

```
{{#is_match "log.attributes.error.reason_codes" "RC00000" }} reason_code: RC00000  
  
hogeが発生しました  
`http://example.com` を参照してリソースを確認してください  
fooだった場合はfooしてください  
  
{{/is_match}}
```

配列フィールドの場合、以下のように検知されるため " " で囲っている
"RC00000", "RC00001", ...

全体図



自動生成を活用した 運用保守コストを抑える Error/Alert/Runbook の一元集約管理

株式会社サイバーエージェント AI事業本部
岩見彰太

GitHub: @Blwashi
X: @B_Sardine



DevOpsDays Tokyo 2024

その他

- Watchdog、Metrics Explorer、Synthetics Test、NPM etc...
- その他にも機能がたくさん！

ぜひ実践編の時に色々触ってみよう！



9

PipeCD 入門



PipeCD

- ハンズオンで使うため少し紹介します

GitOps

GitOps とは

GitOps is an operational framework that takes DevOps best practices used for application development such as version control, collaboration, compliance, and CI/CD, and applies them to infrastructure automation.
by [GitLab](#)

GitOps とは

GitOps は、バージョン管理、コラボレーション、コンプライアンス、CI/CD などのアプリケーション開発に使用される DevOps のベスト プラクティスを取り入れ、インフラストラクチャの自動化に適用する運用フレームワークです。

by [GitLab](#)

- **GitOps は Weaveworks 社が提唱**
- **インフラとアプリケーションの両方を含めたシステム全体のコードを Git を使って管理する**
- **Git を信頼できる唯一の情報源とする**
- **Git をみると全ての情報が分かる（宣言的に管理されている）**

GitOps のメリット

- 直接環境にアクセスしなくていい
- 全ての構成変更は Git (PR) を通して行われる
- 今 Git で確認できる構成 == 今動いている構成
- 全ての変更が追跡可能

PipeCD

PipeCD



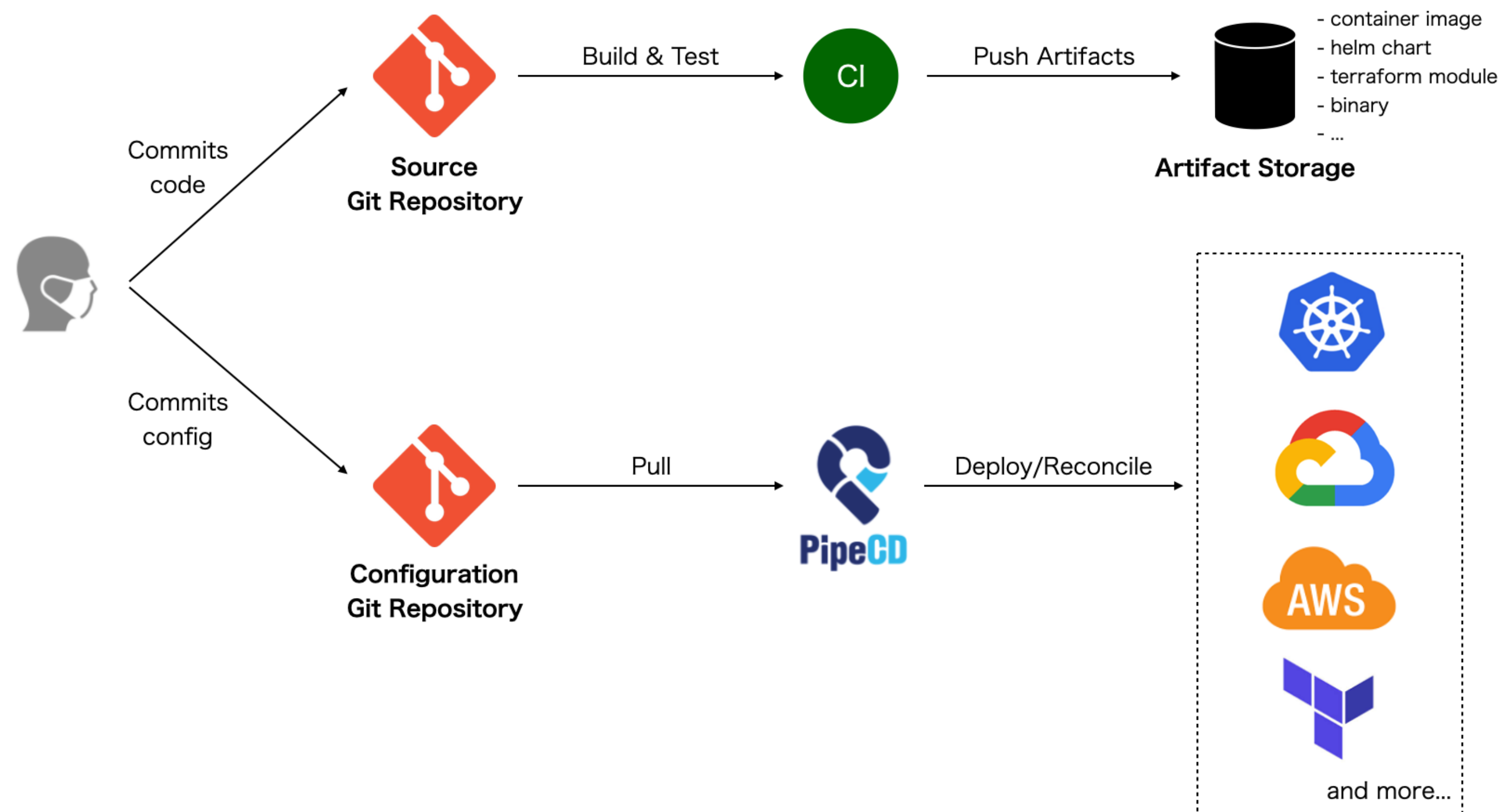
- GitOps スタイルの CD
- K8s だけでなく ECS、Lambda、CloudRun、Terraform などを統一したUX
で管理可能
- 社内の DP 室がメインでメンテナンスしているOSS

CNCF Sandboxにも採択

The screenshot shows the PipeCD web interface for a deployment named 'bluegreen'. The deployment is in a 'FAILURE' state, indicated by a red exclamation mark icon. The failure occurred 4 minutes ago while rolling out primate workloads. The application is 'bluegreen', the piped is 'nghialv-local', and the commit is 'Update helloworld image to v0.5.0 (5e05202)'. The deployment was triggered by 'Le Van Nghia'. The summary states: 'Sync progressively because of updating image helloworld from v0.4.0 to v0.5.0'. The deployment flow is shown as a sequence of steps: 'K8S_CANARY_ROLLOUT' (success), 'K8S_TRAFFIC_ROUTING' (success), 'WAIT_APPROVAL' (success, approved by nghialv), 'K8S_PRIMARY_ROLLOUT' (failure), 'K8S_TRAFFIC_ROUTING' (failure), 'K8S_CANARY_CLEAN' (failure), and 'ROLLBACK' (success). A 'ROLLBACK' dialog box is open, showing a log of the rollback process. The log shows the following steps: 1. Loading manifests at running commit d274075d71be7ecaf8595d6e3233a77f10883b05 for handling. 2. Successfully loaded 2 manifests. 3. Start applying 2 manifests. 4. Applied manifest: name='bluegreen', kind='Deployment', namespace='default', apiVersion='apps/v1'. 5. Applied manifest: name='bluegreen', kind='Service', namespace='default', apiVersion='v1'. 6. Successfully applied 2 manifests. 7. Start checking to ensure that the CANARY variant should be removed. 8. Starting finding and deleting service resources of CANARY variant. 9. No resources to delete. 10. Starting finding and deleting workload resources of CANARY variant. 11. Start deleting 1 resources. 12. Deleted resource: name='bluegreen-canary', kind='Deployment', namespace='default', apiVersion='apps/v1'. 13. Successfully deleted 1 resources. 14. Start checking to ensure that the BASELINE variant should be removed.

PipeCD Overview

- Git repository の変更を PipeCD が検知
- Git の内容を元にインフラリソースを PipeCD が変更する



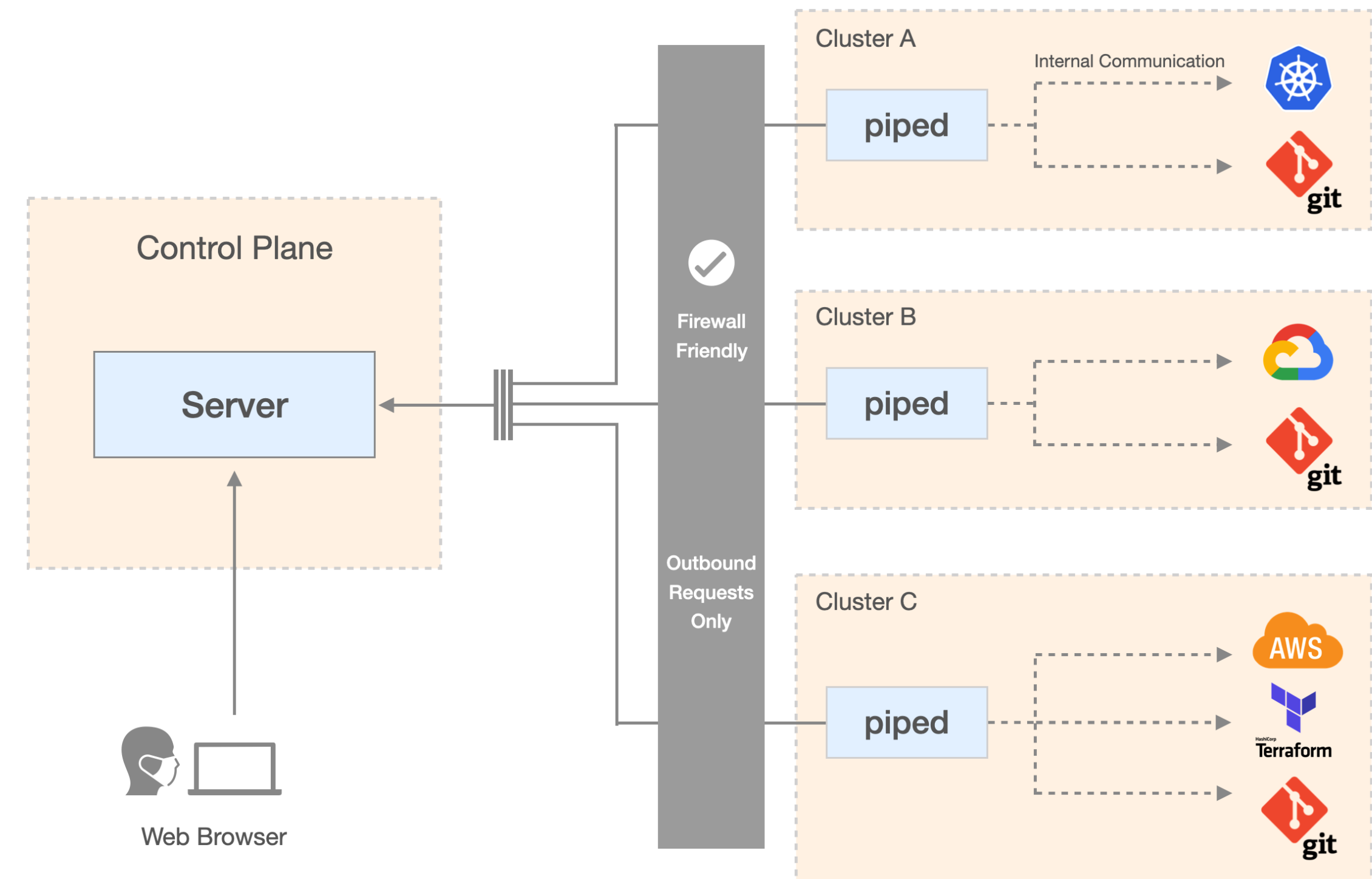
PipeCD Concept

• Piped

展開するネットワークであるクラスター内で実行するバイナリコンポーネント
ステートレスなので単一のVMやローカルマシンなどでも実行できる

• Control Plane

デプロイデータを一元管理
社内では SaaS としても提供されている
自分で立てることもできる

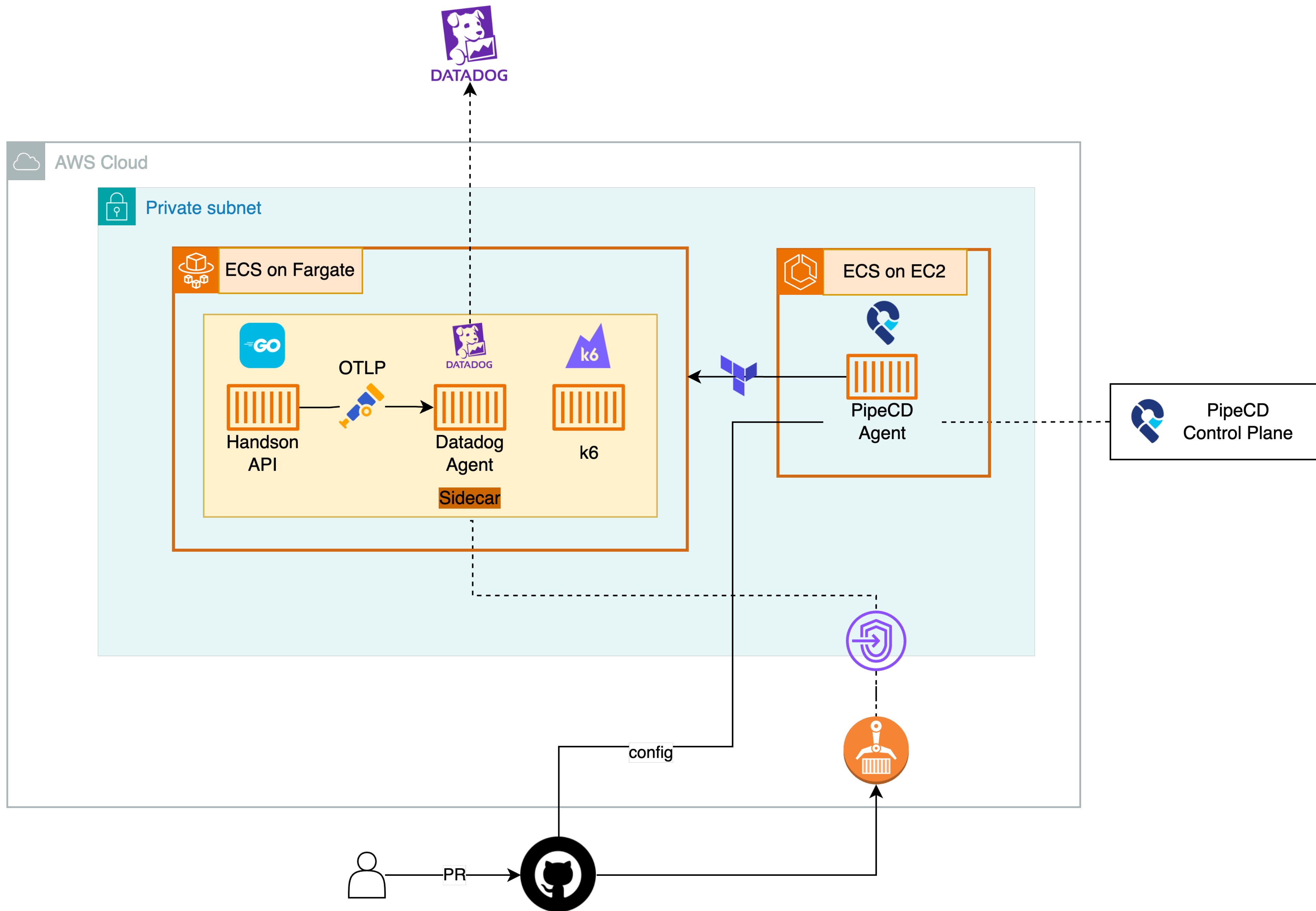


10

ハンズオン



OpenTelemetry を計装して
Datadog に送ってみよう！



概要

- OpenTelemetry を実際に計装してみよう
- OTLP をDatadog Agent を経由して Datadog APM Trace で見れるようにしてみよう
- Log と Trace を紐づけてみよう
- Monitors でアラートを設定してみよう
- ダッシュボードを作ってみよう
- (時間が余った人) PipeCD のカナリアリリースを試してみよう
- (時間が余った人) OTel SDK から Datadog SDK に載せ替えてみよう

事前準備

AWS

- 全員で一つのアカウントを使います

社内向け

Datadog

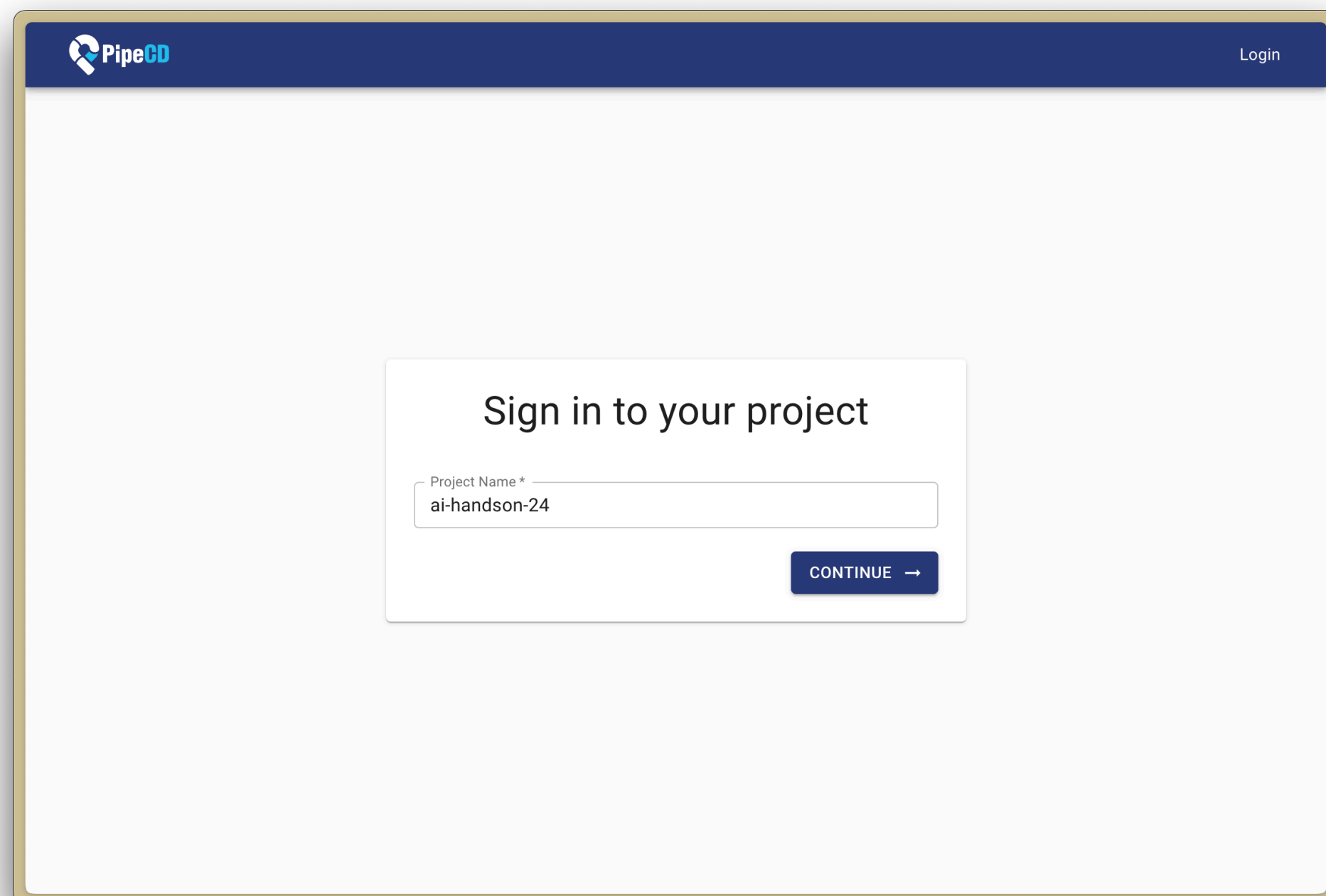
- 全員で一つのアカウントを使います

社内向け

PipeCD Control Plane

社内向け

- 自分のアプリケーションの登録などをしてもらいます
- Github でログインしてください（Github の SSO を使用しています）

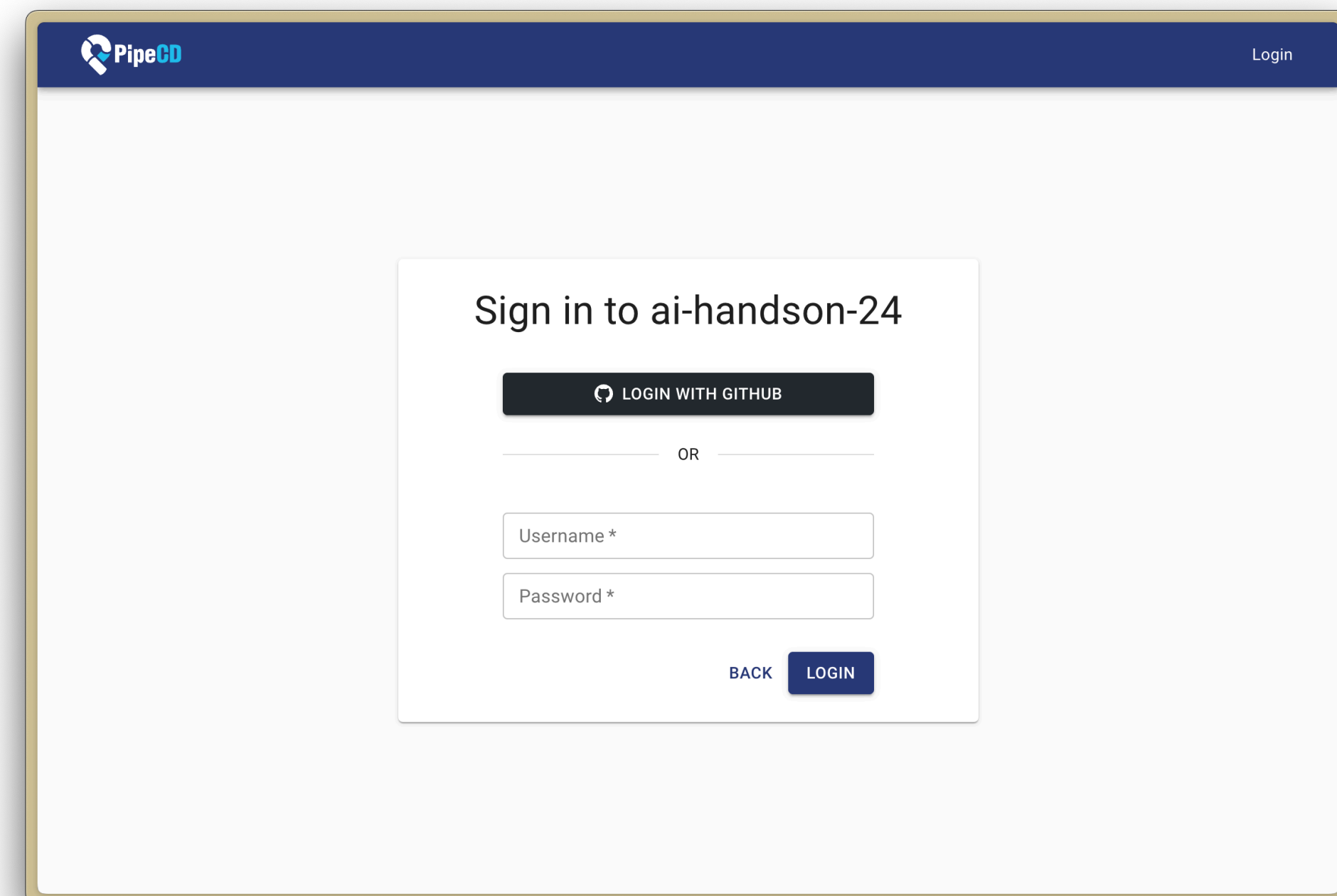


PipeCD Login

Sign in to your project

Project Name *
ai-handson-24

CONTINUE →



PipeCD Login

Sign in to ai-handson-24

LOGIN WITH GITHUB

OR

Username *

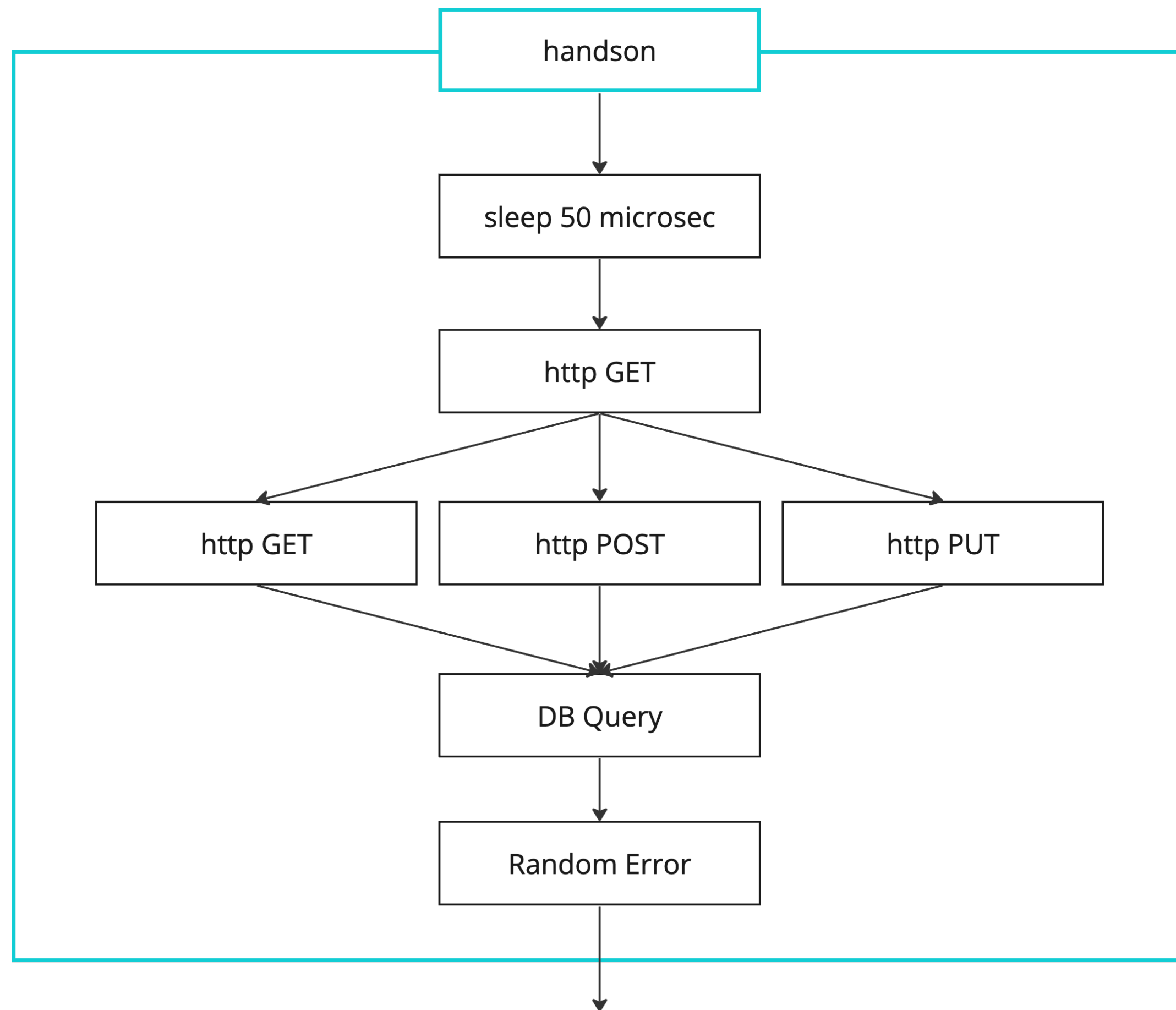
Password *

BACK LOGIN

API

/handson

- 一つだけエンドポイントが生えてる



```
func (h *Handler) handson(w http.ResponseWriter, r *http.Request) (int, error) {
    ctx := r.Context()

    // Custom Function
    h.sleep50microsec(ctx)

    // External API Call
    if err := h.restHttpBin(ctx, http.MethodGet, "get"); err != nil {
        return http.StatusInternalServerError, err
    }
    // External API Call concurrently
    errGroup, errCtx := errgroup.WithContext(ctx)

    errGroup.Go(func() error {
        return h.restHttpBin(errCtx, http.MethodGet, "get")
    })

    errGroup.Go(func() error {
        return h.restHttpBin(errCtx, http.MethodPost, "post")
    })

    errGroup.Go(func() error {
        return h.restHttpBin(errCtx, http.MethodPut, "put")
    })

    if err := errGroup.Wait(); err != nil {
        return http.StatusInternalServerError, err
    }

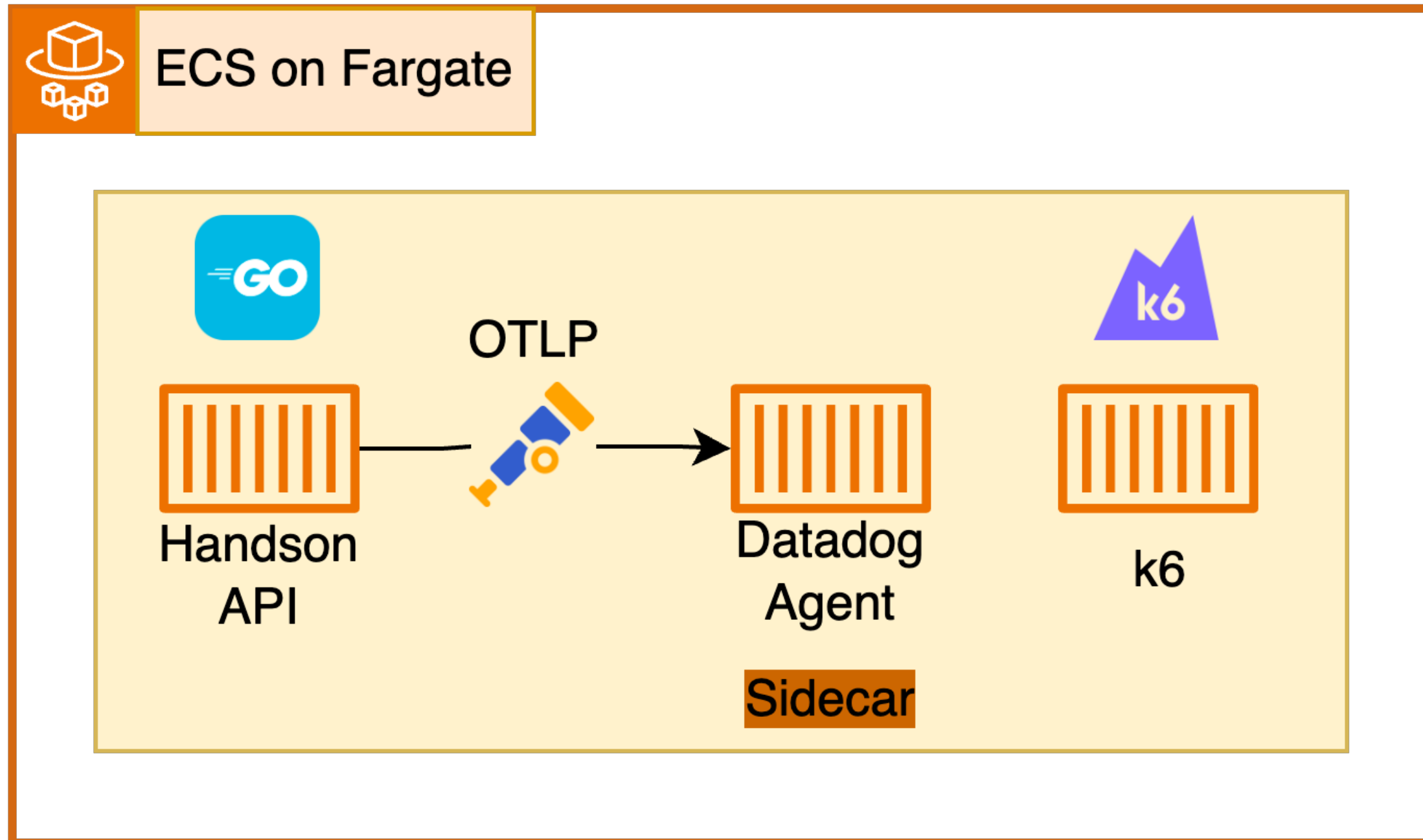
    // Database Query
    if err := h.dbQuery(ctx); err != nil {
        return http.StatusInternalServerError, err
    }

    // Random Error
    if err := h.randomError(radomErrorRate); err != nil {
        return http.StatusInternalServerError, err
    }

    return http.StatusOK, nil
}
```

k6

- 5 rps でサイドカーコンテナがずっとリクエストを送り続けている



```
import http from 'k6/http';
import { check } from 'k6';

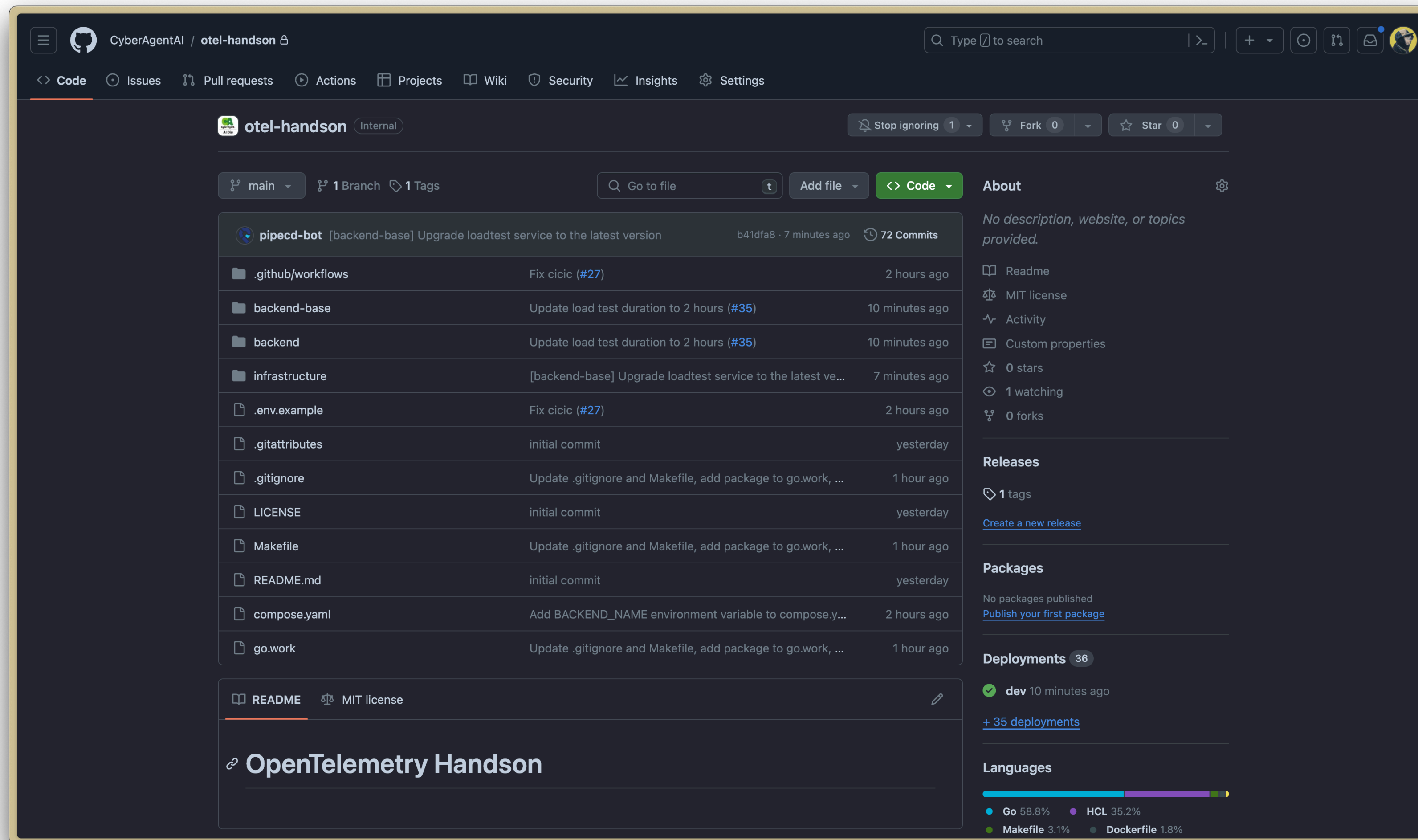
export const options = {
  scenarios: {
    contacts: {
      executor: 'constant-arrival-rate',
      rate: 5,
      timeUnit: '1s',
      duration: '2h',
      preAllocatedVUs: 50,
    }
  }
}

export default function () {
  let url = 'http://localhost:8080/handson';
  let response = http.get(url);
  check(response, {
    'is status 200': (r) => r.status === 200,
  });
}
```


ECS on Fargate に
自分のサービスを Deploy する

Github から Repository を Clone

- <https://github.com/CyberAgentAI/otel-handson>



Repository

- **backend/**

すでに計装されたAPI

分からなくなったらカンニングしてください

- **backend-base/**

計装されていないベースのAPI

これをコピーして使う

- **infrastructure/**

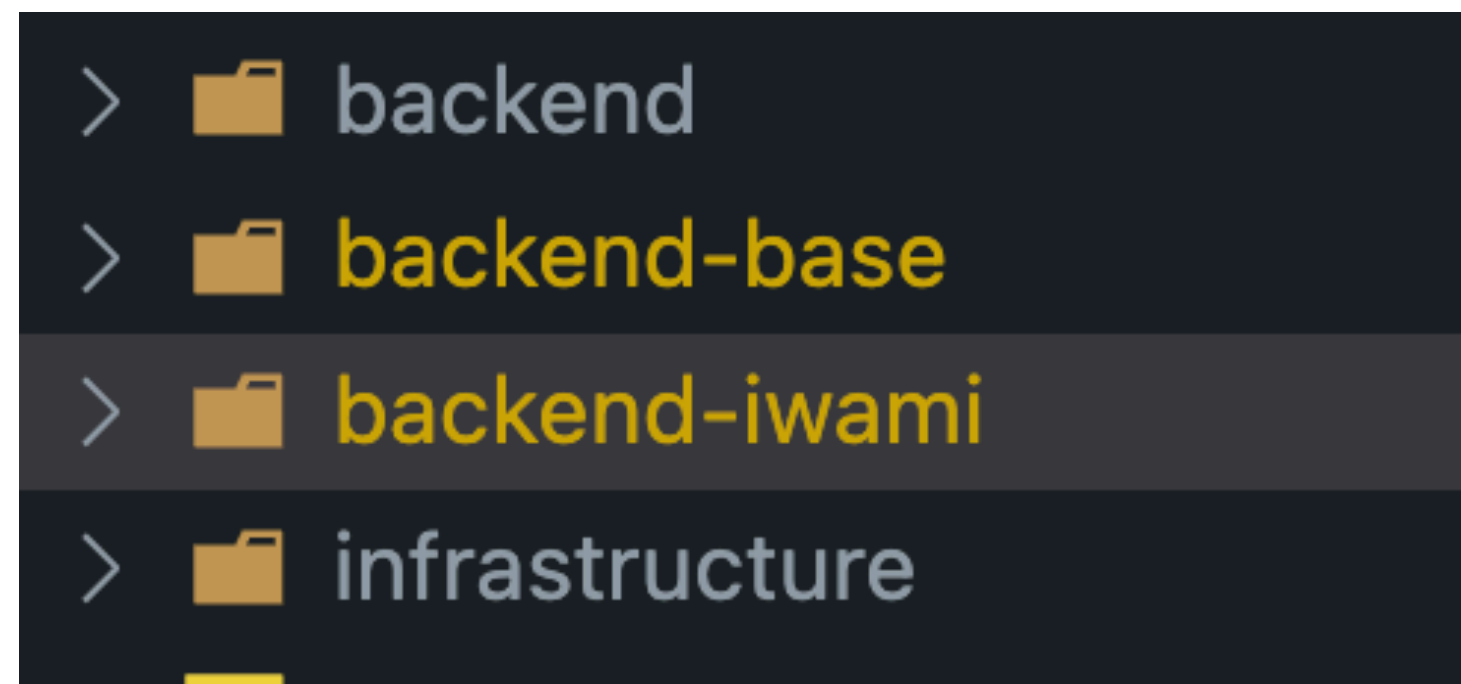
Terraform などが入っています

少し修正してもらいます

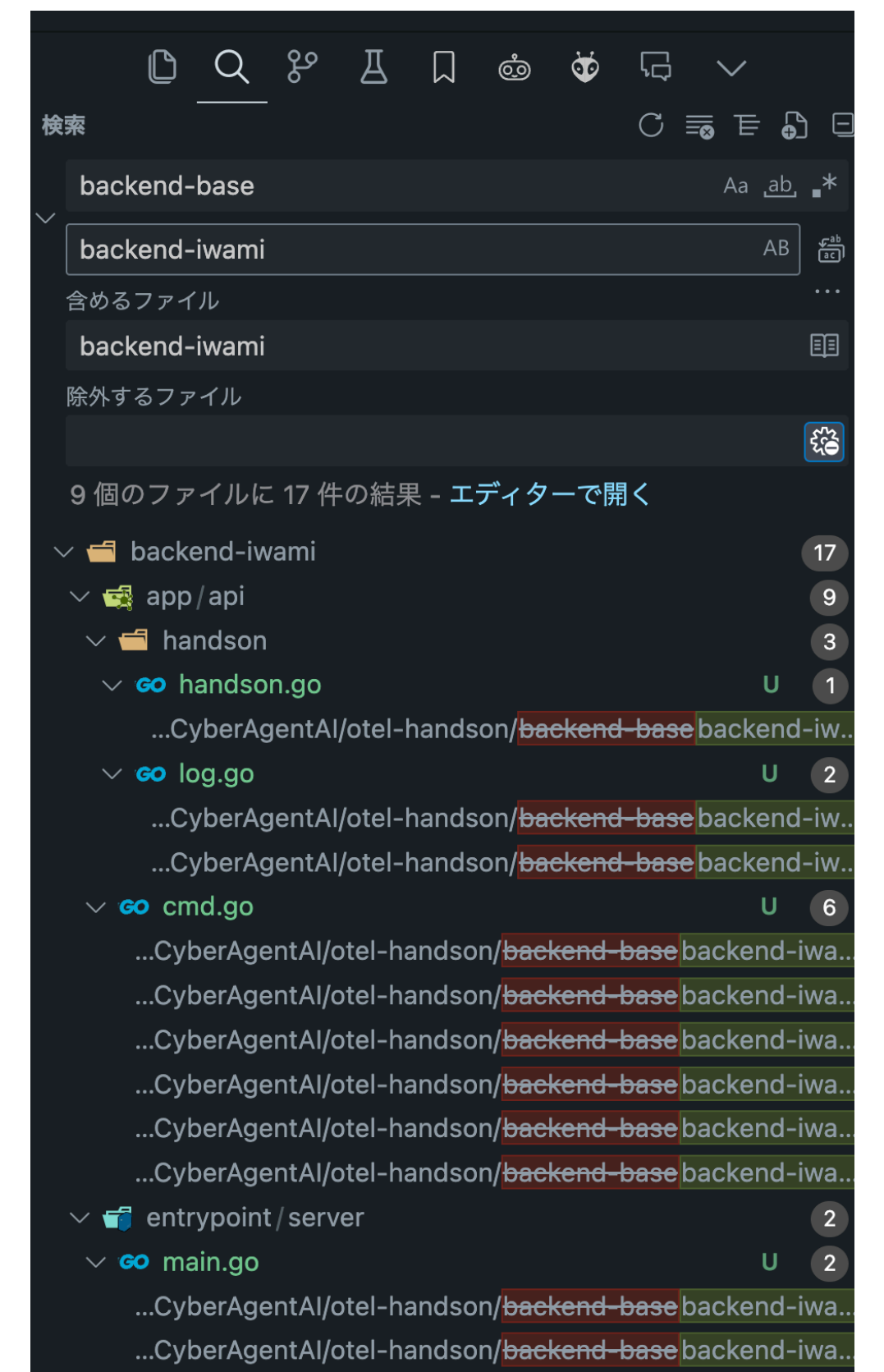
```
.
├── backend
│   ├── app
│   ├── ...
│   └── ...
├── backend-base
│   ├── app
│   ├── ...
│   └── ...
├── compose.yaml
├── ...
├── infrastructure
│   ├── env
│   └── modules
├── ...
└── ...
```

backend-base をコピーする

- backend-base をコピーして backed-{sukina-name} に変える
- コピーしたディレクトリ内の backend-base を全て backed-{sukina-name} に置換する
- .env.example をコピーして .env を作成する
BACKEND_NAME に backed-{sukina-name} を入れる
- go.work に backed-{sukina-name} を追加



```
DD_API_KEY=your_api_key_here
DD_ENV=local
BACKEND_NAME=backend-iwami
```



backend-base をコピーする

- 以下コマンドが実行できて動くことを確認する (docker必須)
 - Hello, handson-{sukina-name} が返ってきたら正解
- ここまでできたら PR を出して merge する

```
> make run/backend
```

```
> curl localhost:8080/handson
```

```
{"message": "Hello, handson-iwami"}
```


ECR を作成

- `infrastructure/env/base-infra/main.tf` の `locals.prefixes` に `backed-{sukina-name}` を追加 (PR出す、コンフリクトするので仲良く)
 - `{prefix}-server-image` : handson-api 用
 - `{prefix}-loadtest-image` : loadtest 用

```
locals {  
  prefixes = [  
    "backend",  
    "backend-base" ## TODO: Add your ecr repository name  
  ]  
}
```

ECR を作成

- main にマージされると、PipeCD Agent が自動で terraform apply

The screenshot displays the PipeCD interface for a deployment. At the top, it indicates the Piped version is v0.47.2. The deployment is marked as SUCCESS, completed 5 minutes ago, in the dev environment for the infra layer. The deployment ID is 27b4800d-631d-4f21-a0c9-0e11d6a1a45e. The commit message is "Add backend-iwami to prefixes in main.tf (#44)" by Shota Iwami. The deployment summary is "Sync with the specified progressive pipeline".

The deployment process is visualized as a sequence of two steps: TERRAFORM_PLAN and TERRAFORM_APPLY, both of which are shown as completed with green checkmarks.

The TERRAFORM_APPLY step is expanded to show the following Terraform output:

```
Plan: 4 to add, 0 to change, 0 to destroy.
66 [2024-05-15 17:58:46 +09:00] module.base.aws_ecr_repository.repositories["backend-iwami-loadtest-image"]: Creating...
67 [2024-05-15 17:58:46 +09:00] module.base.aws_ecr_repository.repositories["backend-iwami-server-image"]: Creating...
68 [2024-05-15 17:58:47 +09:00] module.base.aws_ecr_repository.repositories["backend-iwami-server-image"]: Creation complete after 0s [id=backend-iwami-server-image]
69 [2024-05-15 17:58:47 +09:00] module.base.aws_ecr_repository.repositories["backend-iwami-loadtest-image"]: Creation complete after 0s [id=backend-iwami-loadtest-image]
70 [2024-05-15 17:58:47 +09:00] module.base.aws_ecr_lifecycle_policy.repository_policies["backend-iwami-server-image"]: Creating...
    module.base.aws_ecr_lifecycle_policy.repository_policies["backend-iwami-loadtest-image"]: Creating...
71 [2024-05-15 17:58:47 +09:00] module.base.aws_ecr_lifecycle_policy.repository_policies["backend-iwami-loadtest-image"]: Creation complete after 0s [id=backend-iwami-loadtest-image]
72 [2024-05-15 17:58:47 +09:00] module.base.aws_ecr_lifecycle_policy.repository_policies["backend-iwami-server-image"]: Creation complete after 0s [id=backend-iwami-server-image]
73 [2024-05-15 17:58:47 +09:00]
    Apply complete! Resources: 4 added, 0 changed, 0 destroyed.

Outputs:

base = {
  "vpc_id" = "vpc-0151c2a580454e945"
}
74 [2024-05-15 17:58:47 +09:00] Successfully applied changes
```

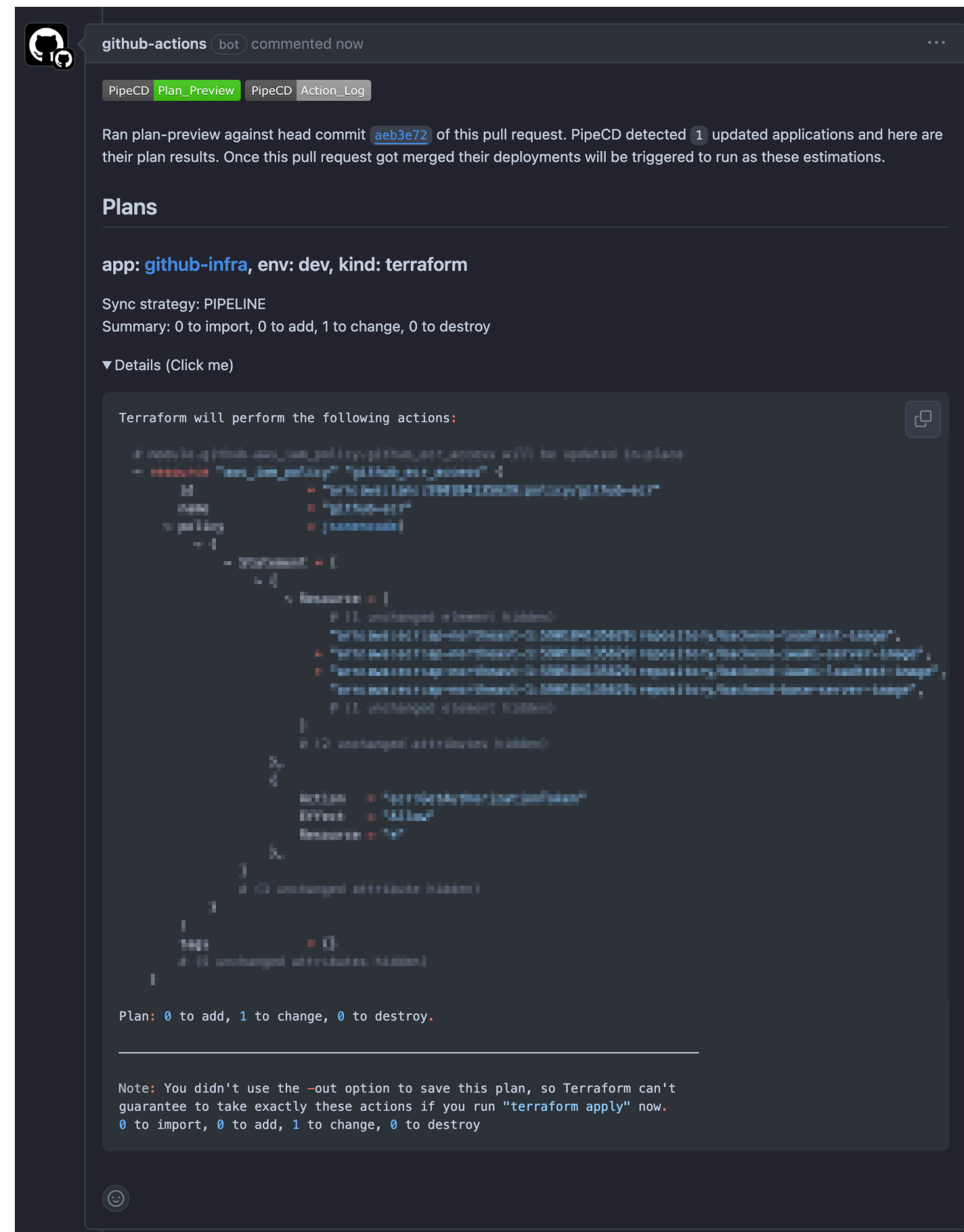
ECR に GitHub からアクセスできるようにする

- `infrastructure/env/github-infra/main.tf` の `locals.prefixes` に `backed-{sukina-name}` を追加 (PR出す、コンフリクトするので仲良く)
- OIDC を使用して AWS 認証できるようにする

```
locals {  
  prefixes = [  
    "backend",  
    "backend-base" ## TODO: Add your ecr repository name  
  ]  
}
```

ECR に GitHub からアクセスできるようにする

- PR の pipecd plan preview で terraform plan の結果も見れる

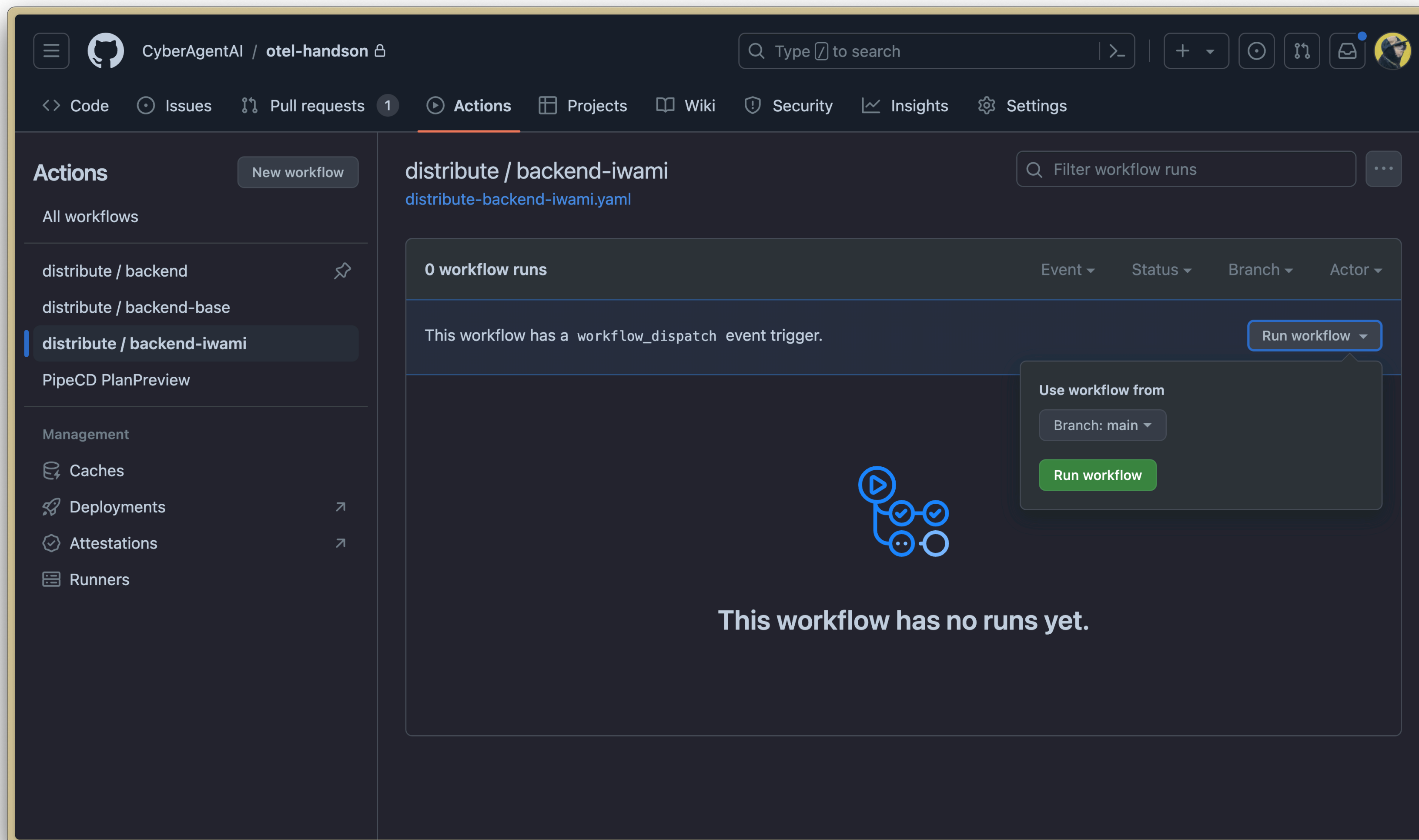


ECR に image を push する actions を作成

- `.github/workflows/distribute-backend-base.yaml` を copy して `.github/workflows/distribute-backend-{sukina-name}.yaml` とかにする
- TODO コメントがついている場所を `backend-base` → `backend-{sukina-name}` に変更する
- できたら PR を出して merge する

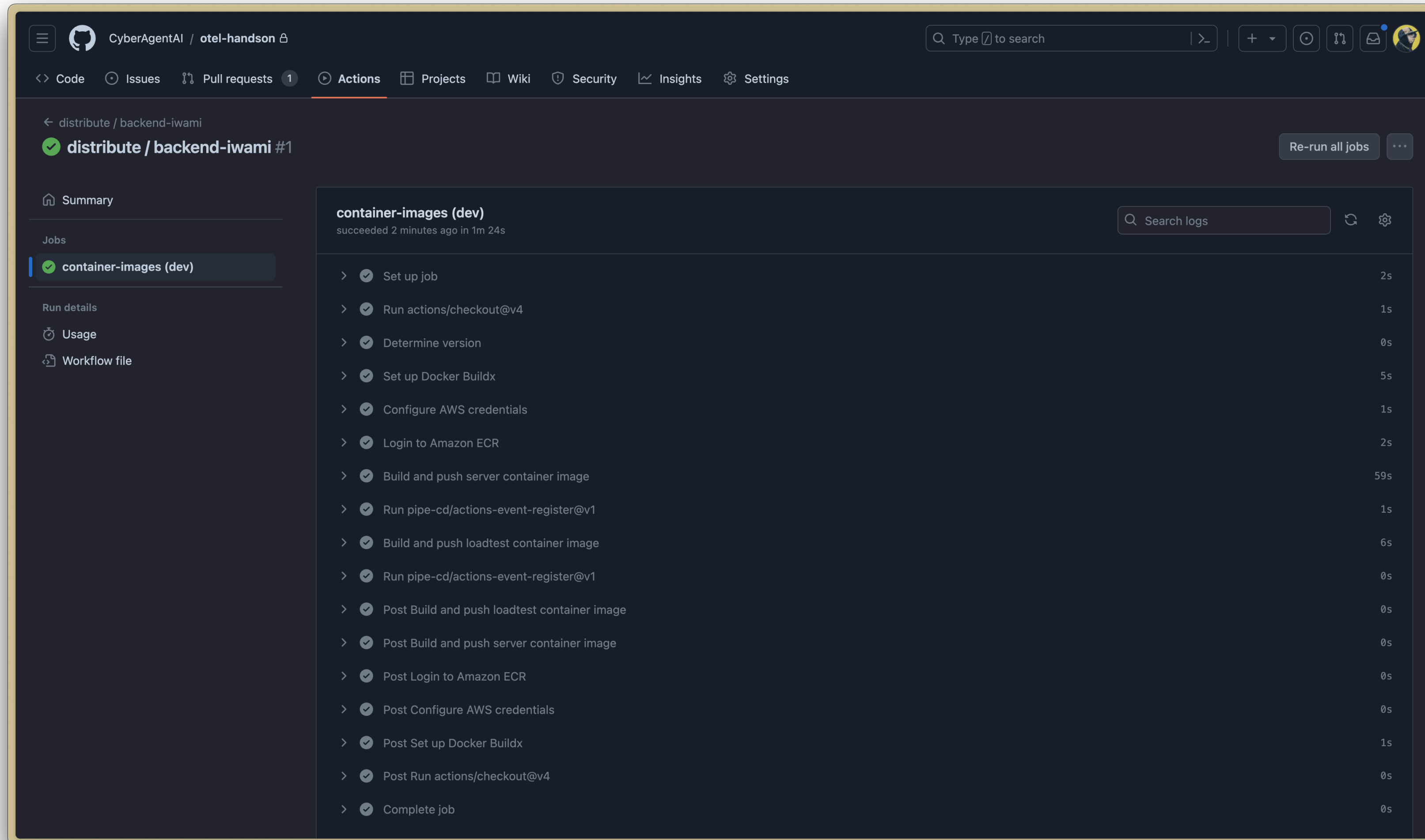
ECR に image を push する actions を作成

- できたら workflow dispatch で起動してみる



ECR に image を push する actions を作成

- 成功してればOK



The screenshot displays the GitHub Actions interface for a workflow named 'container-images (dev)'. The workflow has successfully completed, as indicated by the green checkmark and the text 'succeeded 2 minutes ago in 1m 24s'. The left sidebar shows the workflow file and run details. The main area lists the steps of the workflow, each with a green checkmark and a duration.

Step	Duration
Set up job	2s
Run actions/checkout@v4	1s
Determine version	0s
Set up Docker Buildx	5s
Configure AWS credentials	1s
Login to Amazon ECR	2s
Build and push server container image	59s
Run pipe-cd/actions-event-register@v1	1s
Build and push loadtest container image	6s
Run pipe-cd/actions-event-register@v1	0s
Post Build and push loadtest container image	0s
Post Build and push server container image	0s
Post Login to Amazon ECR	0s
Post Configure AWS credentials	0s
Post Set up Docker Buildx	1s
Post Run actions/checkout@v4	0s
Complete job	0s

PipeCD 用の ECS の config を追加

- `infrastructure/env/handson-api-base` を copy して `infrastructure/env/handson-api-{sukina-name}` にする
- 以下ファイルの TODO 部分を `base` → `{sukina-name}` に変更する

`app.pipcd.yaml`

PipeCD がアプリケーションを認識するための config

`servicedef.yaml`、`taskdef.yaml`

ECS の設定

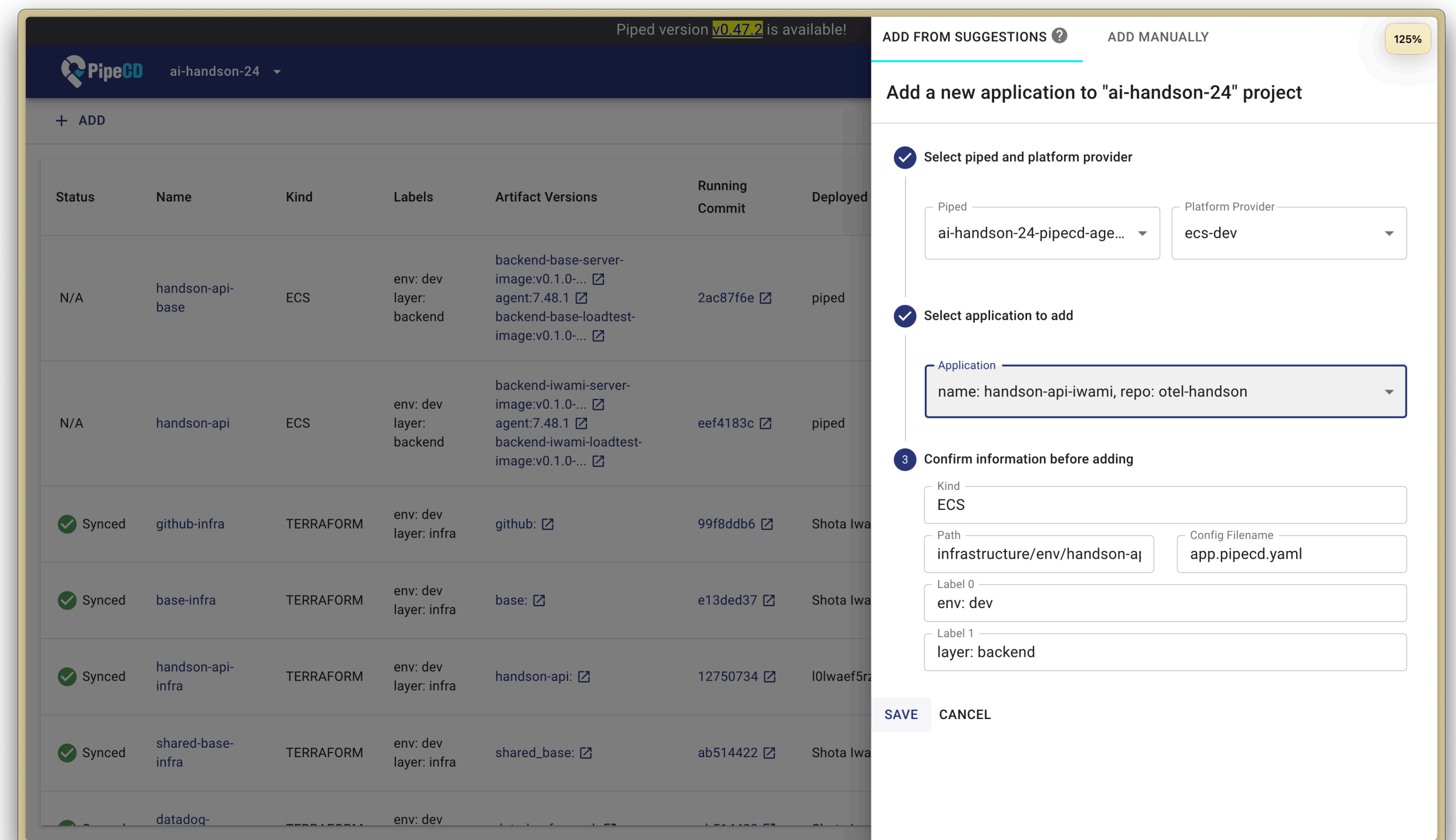
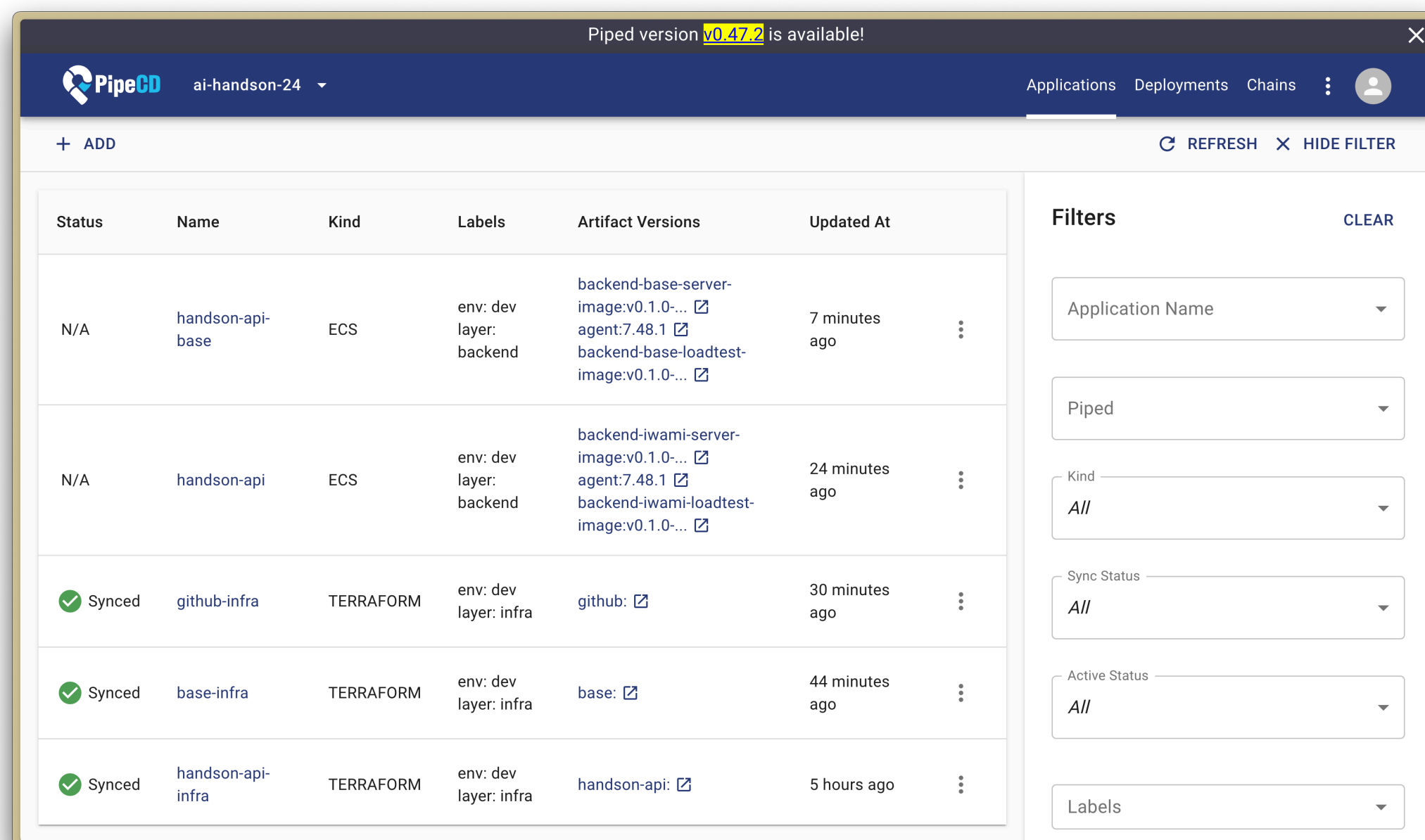
app.pipECD.yaml

- **spec.name**
 - PipeCD 側で認識する component の名前
- **spec.eventWatcher**
 - 実はさっきの actions の最後にコントロールプレーンに対して event を送っている
 - Event Name が一致した際に config のデータを書き換える
 - ECR のイメージが更新されたら、自動的に ECS Task で指定している image の version も変えてくれるようになる

```
apiVersion: pipECD.dev/v1beta1
kind: ECSApp
spec:
  name: handson-api-base
  labels:
    env: dev
    layer: backend
  input:
    serviceDefinitionFile: servicedef.yaml
    taskDefinitionFile: taskdef.yaml
  description: |
    Service to providing hands-on REST APIs to end-
    users.
  eventWatcher:
    - matcher:
        name: handson-api-image-update-base
        labels:
          app: handson-api-base
          env: dev
      handler:
        type: GIT_UPDATE
        config:
          commitMessage: "[backend-base] Upgrade
handson-api service to the latest version"
          replacements:
            - file: taskdef.yaml
              yamlField: $.containerDefinitions[0].image
    ...
```


PipeCD CP で application を登録

- app.piped.yaml を main に merge したので管理対象にいれる
- Applications ので +ADD を選択
- Platform Provider で ecs-dev を選択、Application で先ほど name に入れたものを選択する



PipeCD CP で application を登録

- しばらくすると ECS に Service を追加してくれる

The screenshot shows the PipeCD dashboard for the 'ai-handson-24' workspace. The top navigation bar includes 'Applications', 'Deployments', and 'Chains'. A notification at the top right states 'Piped version v0.47.2 is available!'. The main content area displays a list of applications with the following details:

Status	Application Name	Platform Provider	Environment	Layer	Last Update
PENDING	handson-api-iwami	ECS	env: dev	layer: backend	a few seconds ago
SUCCESS	handson-api-base	ECS	env: dev	layer: backend	11 minutes ago
SUCCESS	handson-api-base	ECS	env: dev	layer: backend	14 minutes ago
SUCCESS	handson-api	ECS	env: dev	layer: backend	28 minutes ago
SUCCESS	handson-api-base	ECS	env: dev	layer: backend	28 minutes ago
SUCCESS	github-infra	TERRAFORM	env: dev	layer: infra	34 minutes ago
SUCCESS	base-infra	TERRAFORM	env: dev	layer: infra	an hour ago

On the right side, there is a 'Filters' panel with dropdown menus for 'Application Name', 'Application Kind' (set to 'All'), 'Application Id', 'Deployment Status' (set to 'All'), and 'Labels'. A 'CLEAR' button is located at the top right of the filters panel.

The screenshot shows the deployment details for the 'handson-api-iwami' application. The top navigation bar is the same as in the previous screenshot. The main content area displays the following information:

- Status:** SUCCESS (a minute ago)
- Environment:** env: dev
- Layer:** layer: backend
- Deployment ID:** 4fe4e38c-6898-4ccb-bc67-cb4ebd414d16
- Commit:** Upgrade handson-api and loadtest services to the latest version (#48) (440814c)
- Application:** handson-api-iwami
- Triggered by:** Shota Iwami
- Piped:** ai-handson-24-pipecd-agent
- Platform Provider:** ecs-dev
- Summary:** Quick sync to deploy image v0.1.0-84-gf51709e and configure all traffic to it (pipeline was not configured)

Below the summary, there is a green 'ECS_SYNC' button. Underneath, a log entry for 'ECS_SYNC' is shown: 'Deploy the new version and configure all traffic to it'. The bottom section of the page contains a detailed log of the deployment process:

```
1 [2024-05-15 18:46:15 +09:00] Preparing deploy source at target commit (440814cc4be6e1bbe49700f7aa703756e5494caa)
2 [2024-05-15 18:46:17 +09:00] Successfully cloned the target commit
3 [2024-05-15 18:46:17 +09:00] Successfully loaded the application configuration file
4 [2024-05-15 18:46:17 +09:00] Successfully prepared deploy source at target commit (440814cc4be6e1bbe49700f7aa703756e5494caa)
5 [2024-05-15 18:46:17 +09:00] Loading task definition manifest at commit 440814cc4be6e1bbe49700f7aa703756e5494caa
6 [2024-05-15 18:46:17 +09:00] Successfully loaded the ECS task definition at commit 440814cc4be6e1bbe49700f7aa703756e5494caa
7 [2024-05-15 18:46:17 +09:00] Loading service manifest at commit 440814cc4be6e1bbe49700f7aa703756e5494caa
8 [2024-05-15 18:46:17 +09:00] Successfully loaded the ECS service definition at commit 440814cc4be6e1bbe49700f7aa703756e5494caa
9 [2024-05-15 18:46:17 +09:00] Loading target groups config at the commit 440814cc4be6e1bbe49700f7aa703756e5494caa
10 [2024-05-15 18:46:17 +09:00] No target groups were set at commit 440814cc4be6e1bbe49700f7aa703756e5494caa
11 [2024-05-15 18:46:17 +09:00] Start applying the ECS task definition
12 [2024-05-15 18:46:17 +09:00] Start applying the ECS service definition
13 [2024-05-15 18:46:17 +09:00] Start rolling out ECS task set
14 [2024-05-15 18:46:18 +09:00] Successfully applied the service definition and the task definition for ECS service handson-api-iwami and task definition of family handson-app-cluster-handson-api
```

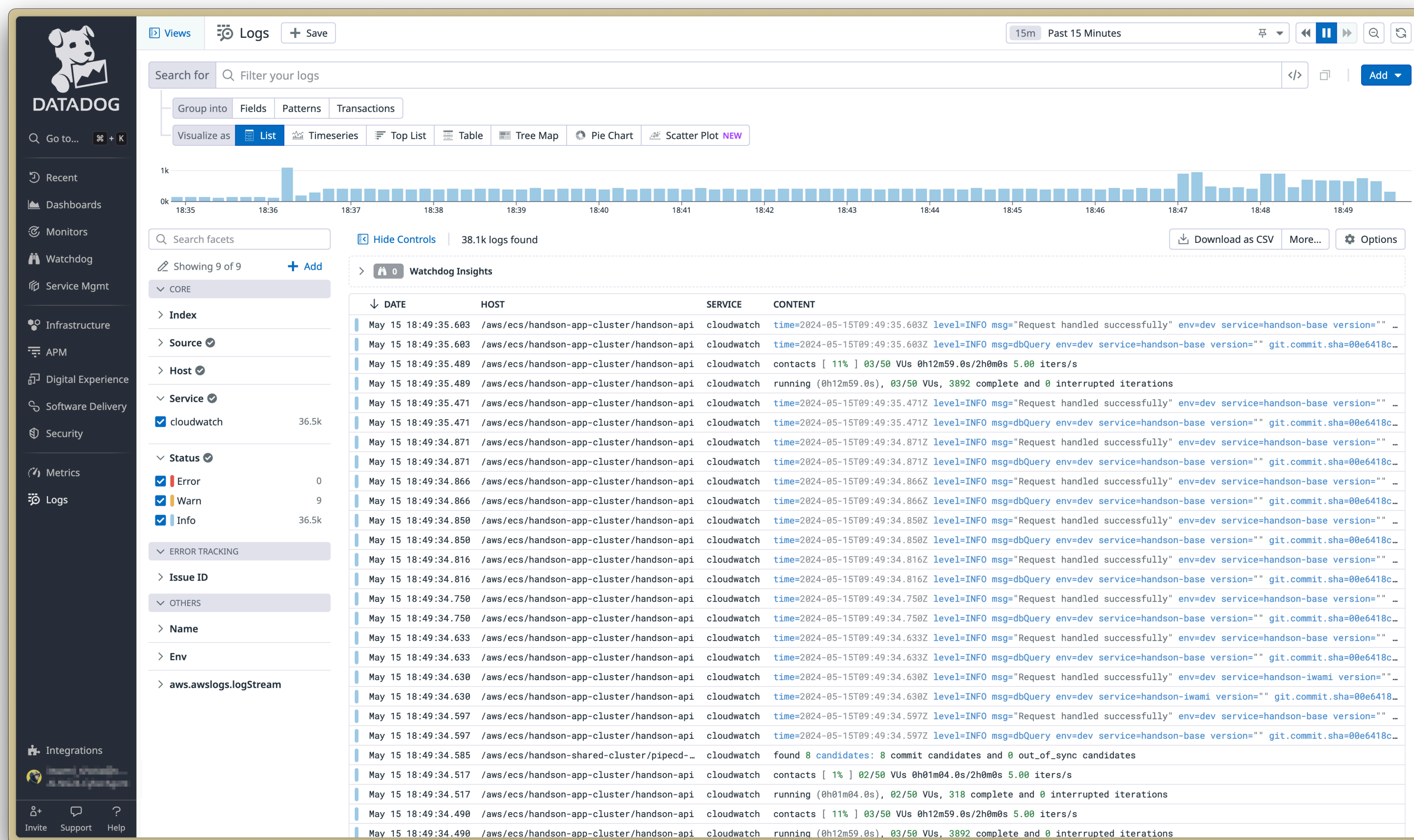
ECS に追加されているか見に行く

- ECS -> handson-app-cluster に Service が追加されていれば成功

社内向け

Datadog に Log を見に行く

- Datadog にログインしてログを見に行く

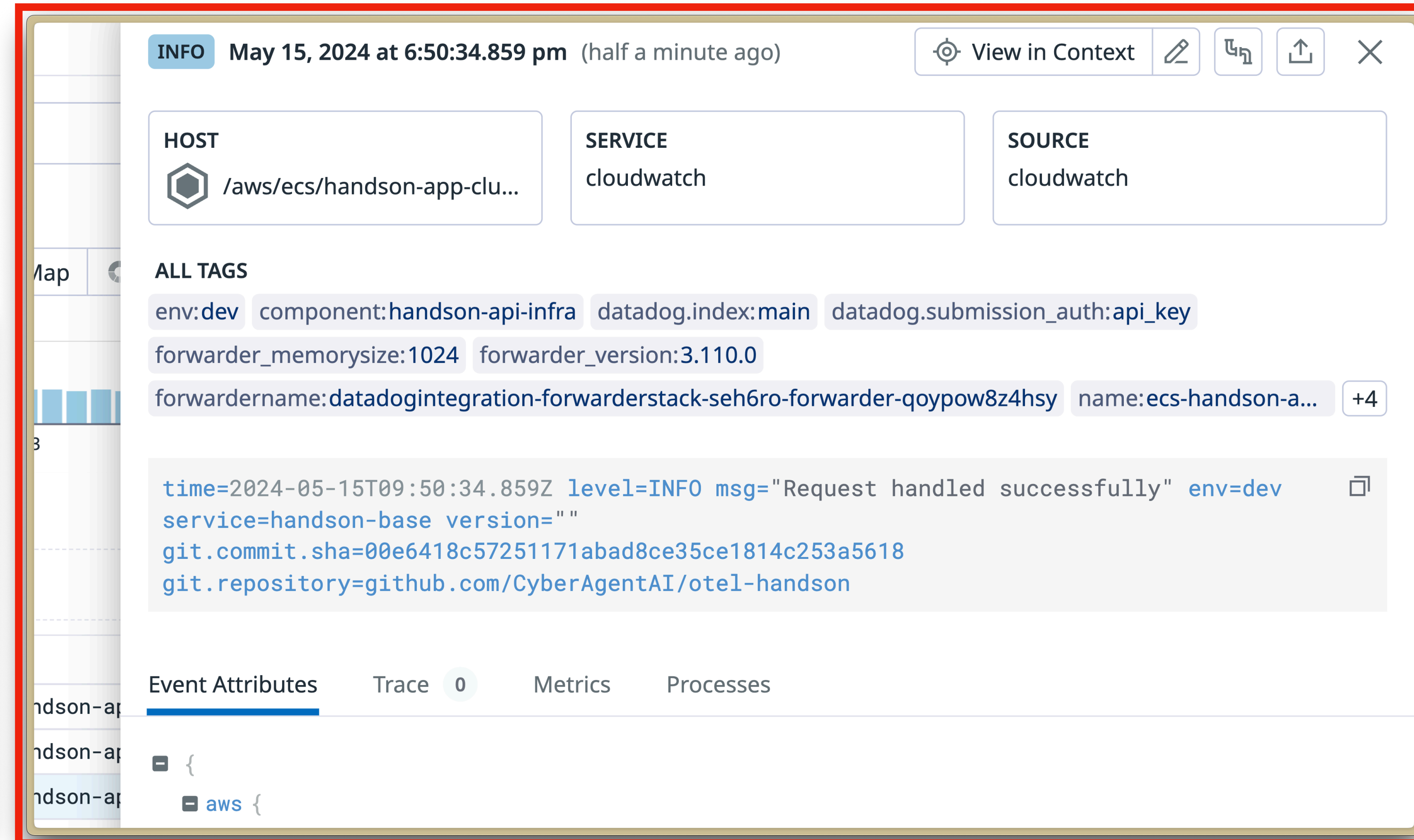
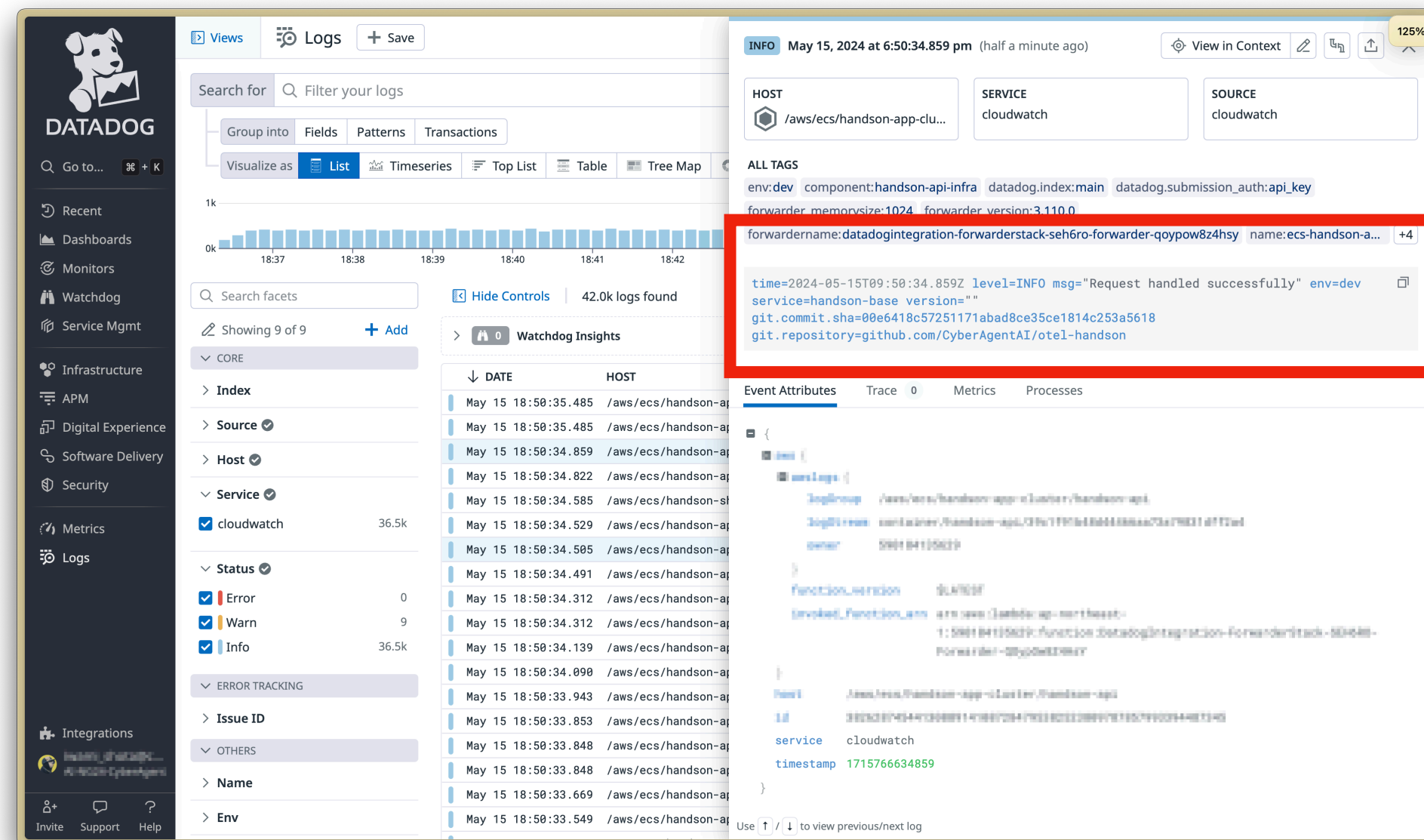


The screenshot displays the Datadog Logs interface. On the left is a navigation sidebar with the Datadog logo and various menu items like 'Recent', 'Dashboards', 'Monitors', 'Watchdog', 'Service Mgmt', 'Infrastructure', 'APM', 'Digital Experience', 'Software Delivery', 'Security', 'Metrics', 'Logs', and 'Integrations'. The main area shows a search bar, a visualization selector (currently set to 'List'), and a time range of '15m Past 15 Minutes'. Below this is a bar chart showing log volume over time. The log stream itself is a table with columns for DATE, HOST, SERVICE, and CONTENT. The logs show various messages, including 'Request handled successfully', 'dbQuery', and 'Request handled successfully'.

DATE	HOST	SERVICE	CONTENT
May 15 18:49:35.603	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	time=2024-05-15T09:49:35.603Z level=INFO msg="Request handled successfully" env=dev service=handson-base version="" ...
May 15 18:49:35.603	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	time=2024-05-15T09:49:35.603Z level=INFO msg=dbQuery env=dev service=handson-base version="" git.commit.sha=00e6418c...
May 15 18:49:35.489	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	contacts [11%] 03/50 VUs 0h12m59.0s/2h0m0s 5.00 iters/s
May 15 18:49:35.489	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	running (0h12m59.0s), 03/50 VUs, 3892 complete and 0 interrupted iterations
May 15 18:49:35.471	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	time=2024-05-15T09:49:35.471Z level=INFO msg="Request handled successfully" env=dev service=handson-base version="" ...
May 15 18:49:35.471	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	time=2024-05-15T09:49:35.471Z level=INFO msg=dbQuery env=dev service=handson-base version="" git.commit.sha=00e6418c...
May 15 18:49:34.871	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	time=2024-05-15T09:49:34.871Z level=INFO msg="Request handled successfully" env=dev service=handson-base version="" ...
May 15 18:49:34.871	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	time=2024-05-15T09:49:34.871Z level=INFO msg=dbQuery env=dev service=handson-base version="" git.commit.sha=00e6418c...
May 15 18:49:34.866	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	time=2024-05-15T09:49:34.866Z level=INFO msg="Request handled successfully" env=dev service=handson-base version="" ...
May 15 18:49:34.866	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	time=2024-05-15T09:49:34.866Z level=INFO msg=dbQuery env=dev service=handson-base version="" git.commit.sha=00e6418c...
May 15 18:49:34.850	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	time=2024-05-15T09:49:34.850Z level=INFO msg="Request handled successfully" env=dev service=handson-base version="" ...
May 15 18:49:34.850	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	time=2024-05-15T09:49:34.850Z level=INFO msg=dbQuery env=dev service=handson-base version="" git.commit.sha=00e6418c...
May 15 18:49:34.816	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	time=2024-05-15T09:49:34.816Z level=INFO msg="Request handled successfully" env=dev service=handson-base version="" ...
May 15 18:49:34.816	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	time=2024-05-15T09:49:34.816Z level=INFO msg=dbQuery env=dev service=handson-base version="" git.commit.sha=00e6418c...
May 15 18:49:34.750	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	time=2024-05-15T09:49:34.750Z level=INFO msg="Request handled successfully" env=dev service=handson-base version="" ...
May 15 18:49:34.750	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	time=2024-05-15T09:49:34.750Z level=INFO msg=dbQuery env=dev service=handson-base version="" git.commit.sha=00e6418c...
May 15 18:49:34.633	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	time=2024-05-15T09:49:34.633Z level=INFO msg="Request handled successfully" env=dev service=handson-base version="" ...
May 15 18:49:34.633	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	time=2024-05-15T09:49:34.633Z level=INFO msg=dbQuery env=dev service=handson-base version="" git.commit.sha=00e6418c...
May 15 18:49:34.630	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	time=2024-05-15T09:49:34.630Z level=INFO msg="Request handled successfully" env=dev service=handson-iwami version="" ...
May 15 18:49:34.630	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	time=2024-05-15T09:49:34.630Z level=INFO msg=dbQuery env=dev service=handson-iwami version="" git.commit.sha=00e6418...
May 15 18:49:34.597	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	time=2024-05-15T09:49:34.597Z level=INFO msg="Request handled successfully" env=dev service=handson-base version="" ...
May 15 18:49:34.597	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	time=2024-05-15T09:49:34.597Z level=INFO msg=dbQuery env=dev service=handson-base version="" git.commit.sha=00e6418c...
May 15 18:49:34.585	/aws/ecs/handson-shared-cluster/pipecd-...	cloudwatch	found 8 candidates: 8 commit candidates and 0 out_of_sync candidates
May 15 18:49:34.517	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	contacts [1%] 02/50 VUs 0h01m04.0s/2h0m0s 5.00 iters/s
May 15 18:49:34.517	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	running (0h01m04.0s), 02/50 VUs, 318 complete and 0 interrupted iterations
May 15 18:49:34.490	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	contacts [11%] 03/50 VUs 0h12m59.0s/2h0m0s 5.00 iters/s
May 15 18:49:34.490	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	running (0h12m59.0s), 03/50 VUs, 3892 complete and 0 interrupted iterations

Datadog に Log を見に行く

- 構造化されていないログが出ているはず



ログを構造化してみよう

ログを構造化してみよう

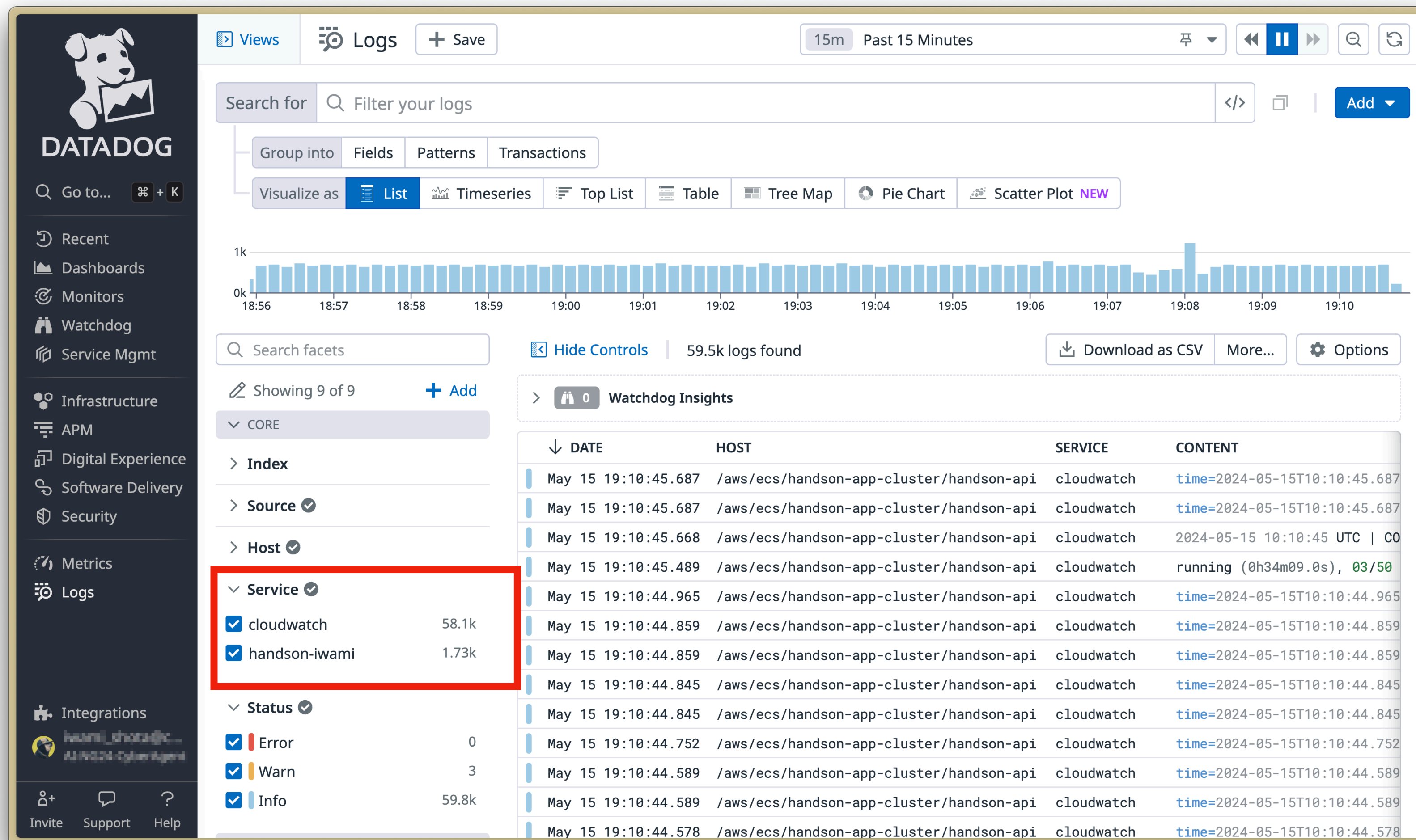
- backend-{sukina-name}/pkg/cli/entrypoint.go の handler を JSON Handler に変更する
- 再び make run/backend を実行して、テナ内のログが json 形式になっていることを確認する
- PR を出して merge する
 - PipeCD によって勝手に新しくデプロイされるようになっている

```
// Before
logger := slog.New(slog.NewTextHandler(os.Stdout,
    &slog.HandlerOptions{
        Level: slog.LevelFromString(flags.LogLevel),
    }),
)

// After
logger := slog.New(slog.NewJSONHandler(os.Stdout,
    &slog.HandlerOptions{
        Level: slog.LevelFromString(flags.LogLevel),
    }),
)
```

再び Datadog の Log を見ると

- 左側の Service にうまく行っていれば自分の Service が出ているはず

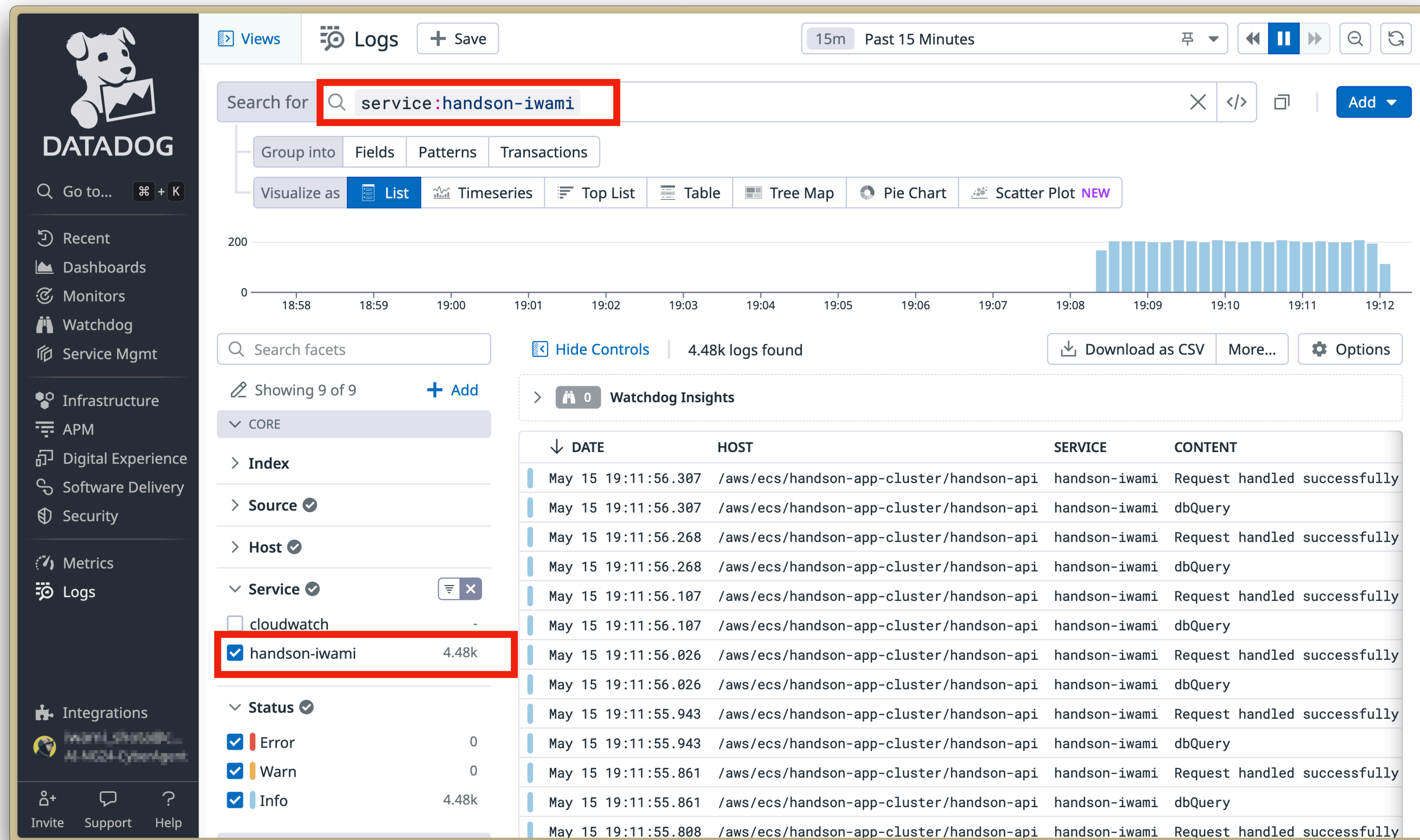


The screenshot displays the Datadog Logs interface. On the left sidebar, the 'Service' filter is expanded, showing 'cloudwatch' (58.1k) and 'handson-iwami' (1.73k) selected. The main panel shows a search bar with 'Filter your logs', a visualization bar with 'List' selected, and a bar chart showing log volume over time. Below the chart, a table of log entries is displayed, with columns for DATE, HOST, SERVICE, and CONTENT. The table shows multiple entries for the 'cloudwatch' service, with content including timestamps and system information.

DATE	HOST	SERVICE	CONTENT
May 15 19:10:45.687	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	time=2024-05-15T10:10:45.687
May 15 19:10:45.687	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	time=2024-05-15T10:10:45.687
May 15 19:10:45.668	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	2024-05-15 10:10:45 UTC CO
May 15 19:10:45.489	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	running (0h34m09.0s), 03/50
May 15 19:10:44.965	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	time=2024-05-15T10:10:44.965
May 15 19:10:44.859	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	time=2024-05-15T10:10:44.859
May 15 19:10:44.859	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	time=2024-05-15T10:10:44.859
May 15 19:10:44.845	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	time=2024-05-15T10:10:44.845
May 15 19:10:44.845	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	time=2024-05-15T10:10:44.845
May 15 19:10:44.752	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	time=2024-05-15T10:10:44.752
May 15 19:10:44.589	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	time=2024-05-15T10:10:44.589
May 15 19:10:44.589	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	time=2024-05-15T10:10:44.589
May 15 19:10:44.578	/aws/ecs/handson-app-cluster/handson-api	cloudwatch	time=2024-05-15T10:10:44.578

再び Datadog の Log を見ると

- 自分の Service の Log だけを絞り込めるようになっている

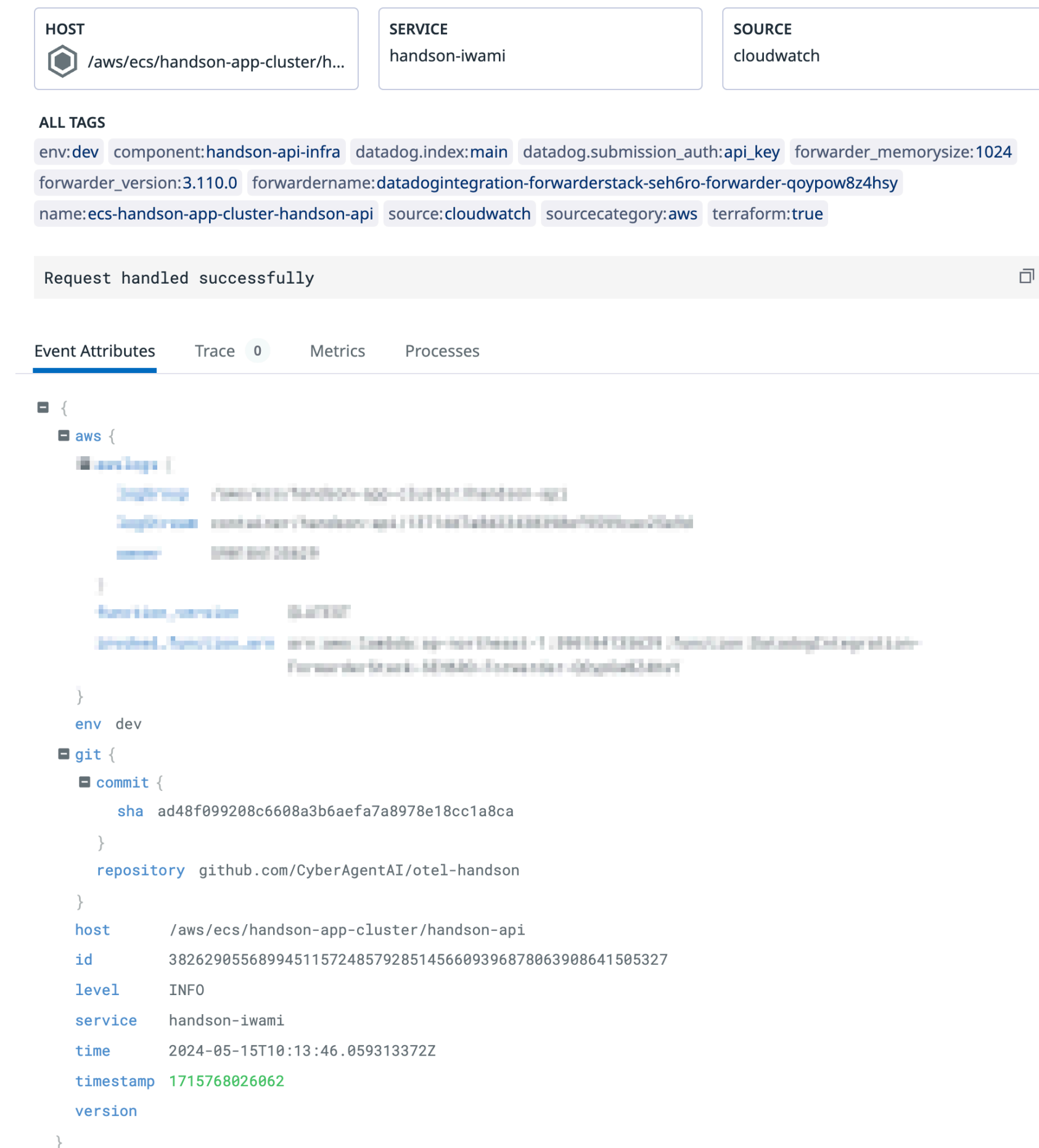


The screenshot shows the Datadog Logs interface. The search bar contains the query `service:handson-iwami`. The interface displays a bar chart showing log volume over time, a search facets section, and a table of log entries. The table has columns for DATE, HOST, SERVICE, and CONTENT. The SERVICE column is filtered to show only 'handson-iwami' logs. The log entries include 'Request handled successfully' and 'dbQuery'.

DATE	HOST	SERVICE	CONTENT
May 15 19:11:56.307	/aws/ecs/handson-app-cluster/handson-api	handson-iwami	Request handled successfully
May 15 19:11:56.307	/aws/ecs/handson-app-cluster/handson-api	handson-iwami	dbQuery
May 15 19:11:56.268	/aws/ecs/handson-app-cluster/handson-api	handson-iwami	Request handled successfully
May 15 19:11:56.268	/aws/ecs/handson-app-cluster/handson-api	handson-iwami	dbQuery
May 15 19:11:56.107	/aws/ecs/handson-app-cluster/handson-api	handson-iwami	Request handled successfully
May 15 19:11:56.107	/aws/ecs/handson-app-cluster/handson-api	handson-iwami	dbQuery
May 15 19:11:56.026	/aws/ecs/handson-app-cluster/handson-api	handson-iwami	Request handled successfully
May 15 19:11:56.026	/aws/ecs/handson-app-cluster/handson-api	handson-iwami	dbQuery
May 15 19:11:55.943	/aws/ecs/handson-app-cluster/handson-api	handson-iwami	Request handled successfully
May 15 19:11:55.943	/aws/ecs/handson-app-cluster/handson-api	handson-iwami	dbQuery
May 15 19:11:55.861	/aws/ecs/handson-app-cluster/handson-api	handson-iwami	Request handled successfully
May 15 19:11:55.861	/aws/ecs/handson-app-cluster/handson-api	handson-iwami	dbQuery
May 15 19:11:55.808	/aws/ecs/handson-app-cluster/handson-api	handson-iwami	Request handled successfully

再び Datadog の Log を見ると

- ログを見ると、key value で入れたものが event attribute として認識



The screenshot shows the Datadog log viewer interface. At the top, there are three filter boxes: HOST (/aws/ecs/handson-app-cluster/h...), SERVICE (handson-iwami), and SOURCE (cloudwatch). Below these, the 'ALL TAGS' section lists various attributes like env:dev, component:handson-api-infra, and name:ecs-handson-app-cluster-handson-api. The main log entry is 'Request handled successfully'. Below the log, the 'Event Attributes' tab is selected, displaying a JSON object with the following structure:

```
{
  "aws": {
    "aws_logs": {
      "log_group": "/aws/ecs/handson-app-cluster/handson-api",
      "log_stream": "ecs-logs-handson-api-1177887346224682988994511572485792851456609396878063908641505327",
      "message": "Request handled successfully"
    }
  },
  "function_name": "lambda",
  "invoked_function_arn": "arn:aws:lambda:ap-northeast-1:1177887346224682988994511572485792851456609396878063908641505327:function:handson-api",
  "env": {
    "dev": true
  },
  "git": {
    "commit": {
      "sha": "ad48f099208c6608a3b6aefa7a8978e18cc1a8ca"
    },
    "repository": "github.com/CyberAgentAI/otel-handson"
  },
  "host": "/aws/ecs/handson-app-cluster/handson-api",
  "id": "38262905568994511572485792851456609396878063908641505327",
  "level": "INFO",
  "service": "handson-iwami",
  "time": "2024-05-15T10:13:46.059313372Z",
  "timestamp": "1715768026062",
  "version": "1.0.0"
}
```

HOST
 /aws/ecs/handson-app-cluster/h...

SERVICE
 handson-iwami

SOURCE
 cloudwatch

ALL TAGS

env:dev component:handson-api-infra datadog.index:main datadog.submission_auth:api_key forwarder_memorysize:1024
forwarder_version:3.110.0 forwardername:datadogintegration-forwarderstack-seh6ro-forwarder-qoypow8z4hsy
name:ecs-handson-app-cluster-handson-api source:cloudwatch sourcecategory:aws terraform:true

Request handled successfully 📄

Event Attributes Trace 0 Metrics Processes

```

{
  aws {
    awslogs {
      log_group /aws/ecs/handson-app-cluster/handson-api
      log_stream handson-api-117157680260626062
      version 3.110.0
    }
    host_id_version 38262905568994511572485792851456609396878063908641505327
    resolved_host_id_version aws-logs-2024-05-15-117157680260626062-1-117157680260626062-aws-logs-datadog-integration-lab-forwarder-stack-117157680260626062-forwarder-qoypow8z4hsy
  }
  env dev
  git {
    commit {
      sha ad48f099208c6608a3b6aefa7a8978e18cc1a8ca
    }
    repository github.com/CyberAgentAI/otel-handson
  }
  host /aws/ecs/handson-app-cluster/handson-api
  id 38262905568994511572485792851456609396878063908641505327
  level INFO
  service handson-iwami
  time 2024-05-15T10:13:46.059313372Z
  timestamp 1715768026062
  version
}

```

```

// backend-{sukina-name}/app/api/handson/log.go

func LogAttributes(ctx context.Context, attrs []any) []any {
  lattrs := []any{
    slog.String("env", "dev"),
    slog.String("service", version.Get().ServiceName),
    slog.String("version", version.Get().Version),
    slog.String("git.commit.sha", version.Get().GitCommit),
    slog.String("git.repository", version.Get().GitRepository),
    // TODO: Add span and trace IDs to log attributes
  }

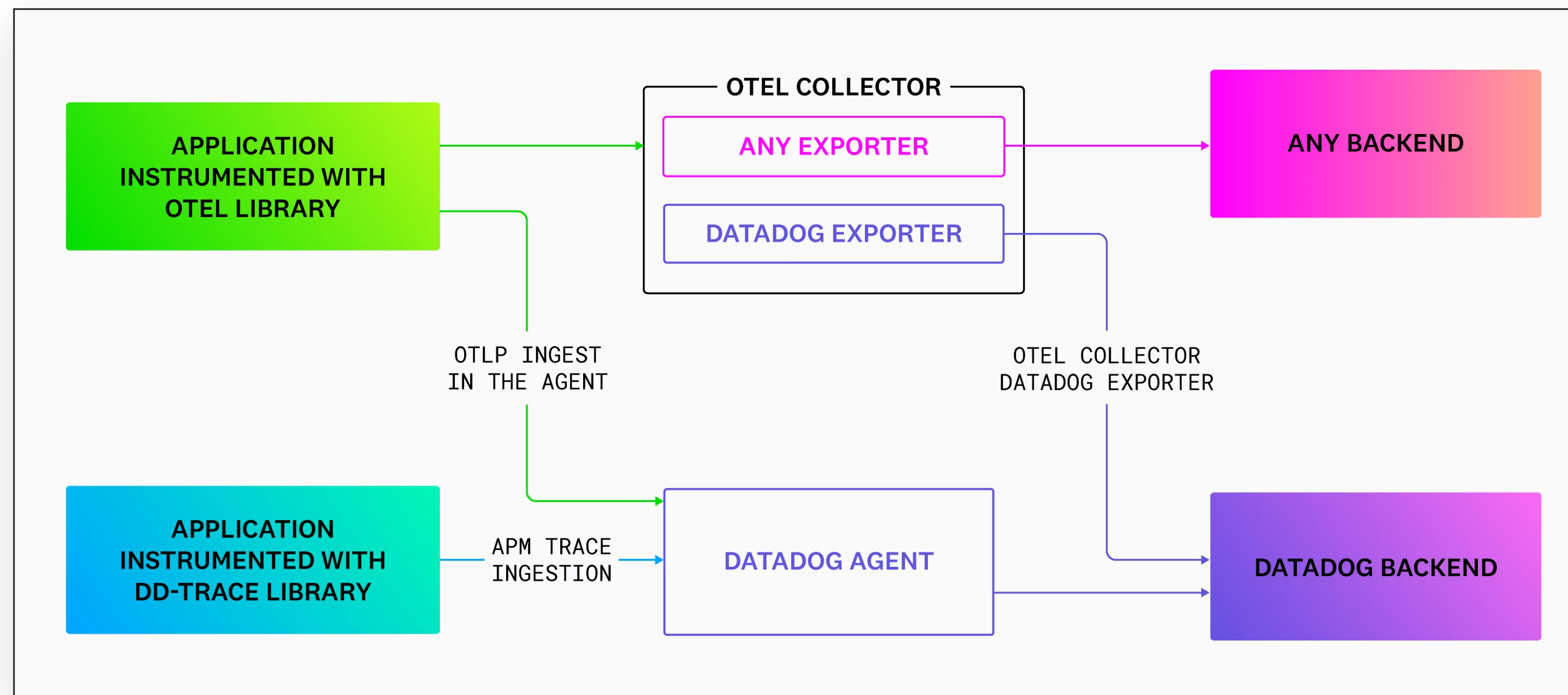
  return append(lattrs, attrs...)
}

```


OpenTelemetry を計装してみよう

OpenTelemetry × Datadog

- シグナルと OTEL Collector を経由するパターンと Datadog Agent に直接送るパターンに2種類存在
- 今回は下の「Datadog Agent を経由」を採用



Datadog の OpenTelemetry

Tracerの初期化

- backend-`{sukina-name}/app/api/cmd.go` に tracer の初期化処理を追加
- Init の中身は 「7. OpenTelemetry を手動計装してみる」 を参照

```
logger.InfoContext(ctx, "Version", slog.String("version",
v.String()))

// NOTE: Init OpenTelemetry
cleanup, err := tracer.Init(ctx)
if err != nil {
    logger.ErrorContext(
        ctx,
        "Failed to initialize tracer",
        handson.ErrorLogAttributes(ctx, err)...,
    )

    return err
}
defer cleanup.Shutdown(ctx)

httpServer.HandleFunc("/healthz", func(w
http.ResponseWriter, r *http.Request) {
```

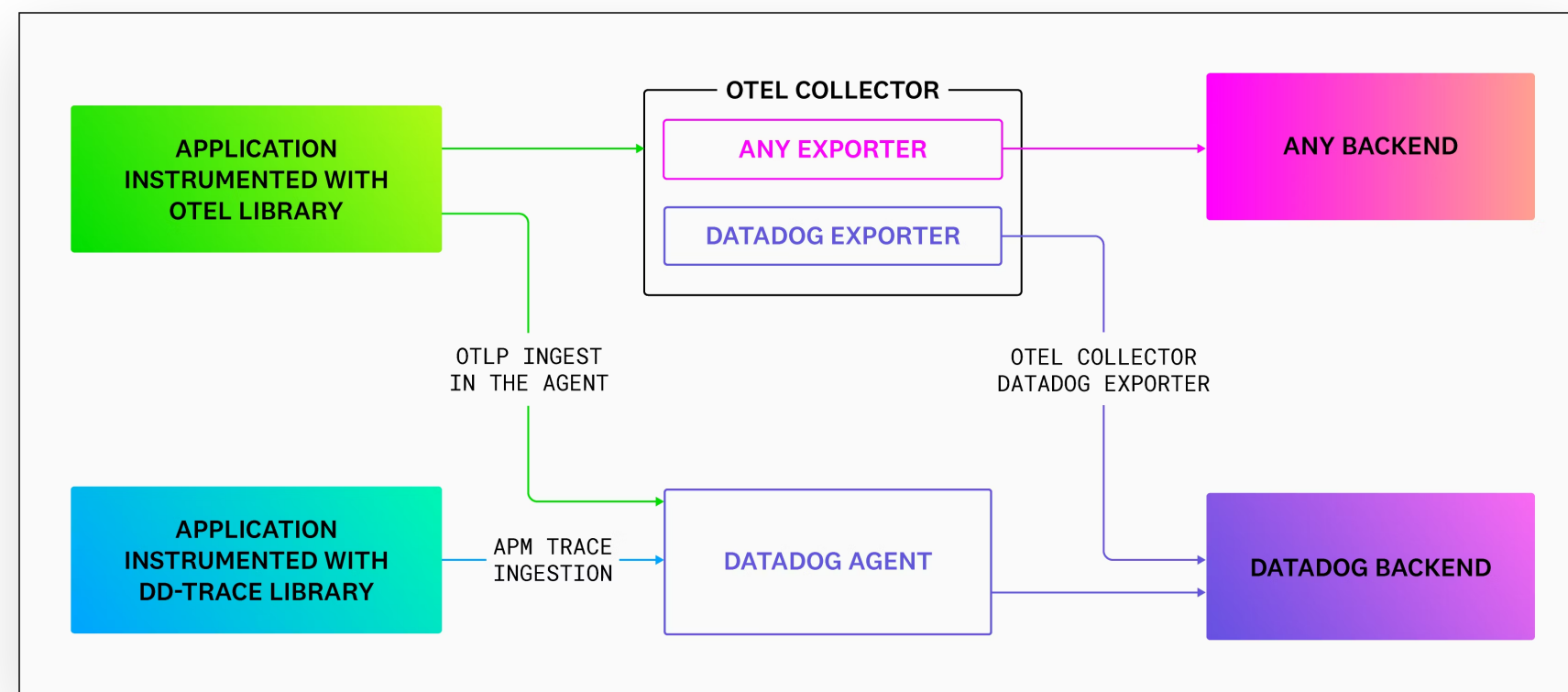
Spanの計装

- backend-{sukina-name}/app/api/handson/handson.go の `function sleep50micrisec` に `span` を計装してみよう！
- できたらPRをマージしてみよう

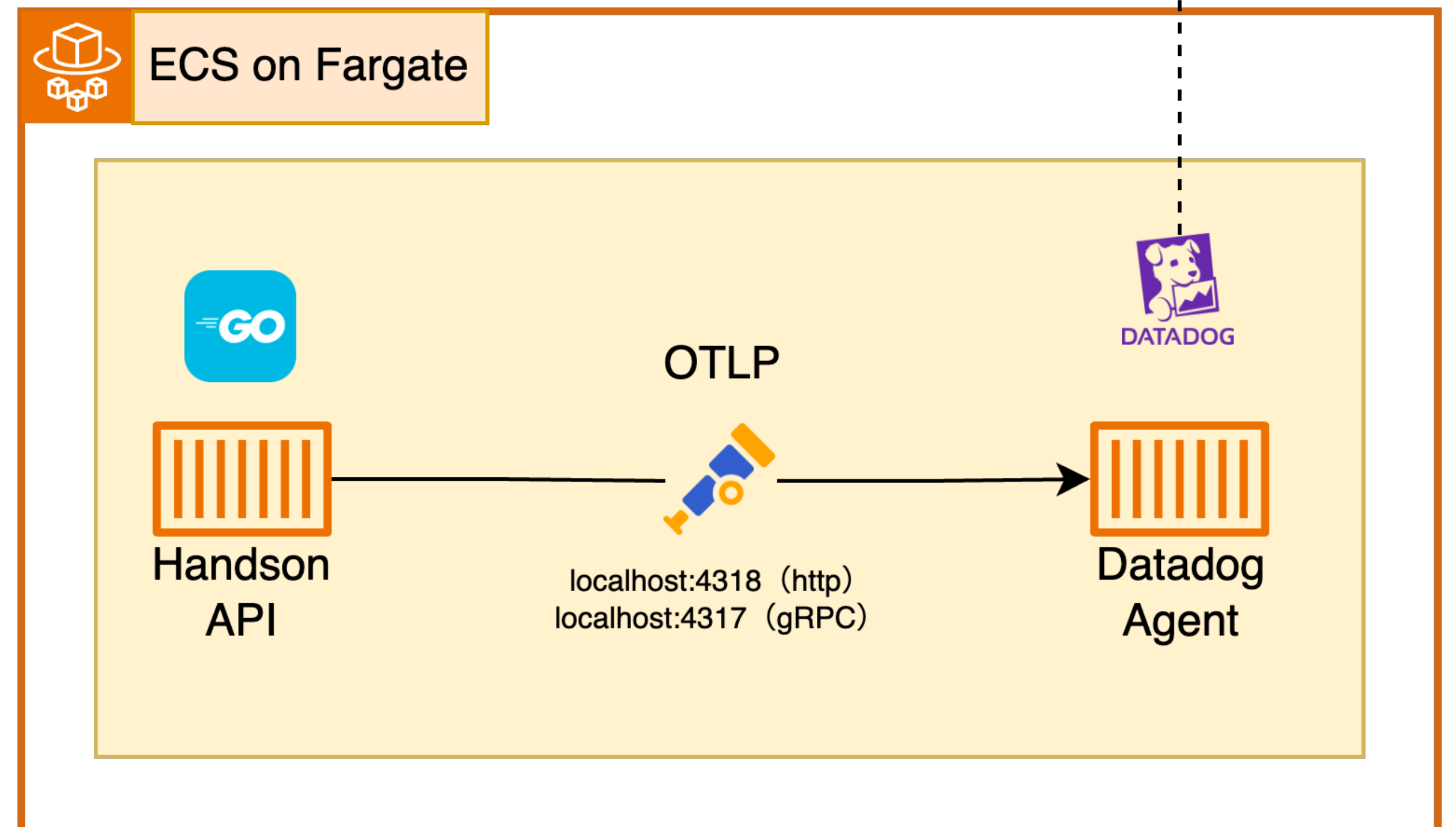
```
func (h *Handler) sleep50microsec(ctx
context.Context) {
    // TODO: Instrument OpenTelemetry span
    time.Sleep(50 * time.Microsecond)
}
```

仕組み

- Datadog Agent の OTLP Receiver で待ち受ける
- サイドカー経由でシグナルを送る
- taskdef.yaml の環境変数を覗いてみてください



Datadog の OpenTelemetry



Datadog の APM

- Service に自分のものが出てたら送れている

The screenshot displays the Datadog APM interface for the service 'AI-NG24-CyberAgent'. The left sidebar contains navigation options such as 'Go to...', 'Recent', 'Dashboards', 'Monitors', 'Watchdog', 'Service Mgmt', 'Infrastructure', 'APM', 'Digital Experience', 'Software Delivery', 'Security', 'Metrics', 'Logs', 'Integrations', 'Invite', 'Support', and 'Help'. The main content area is divided into three sections:

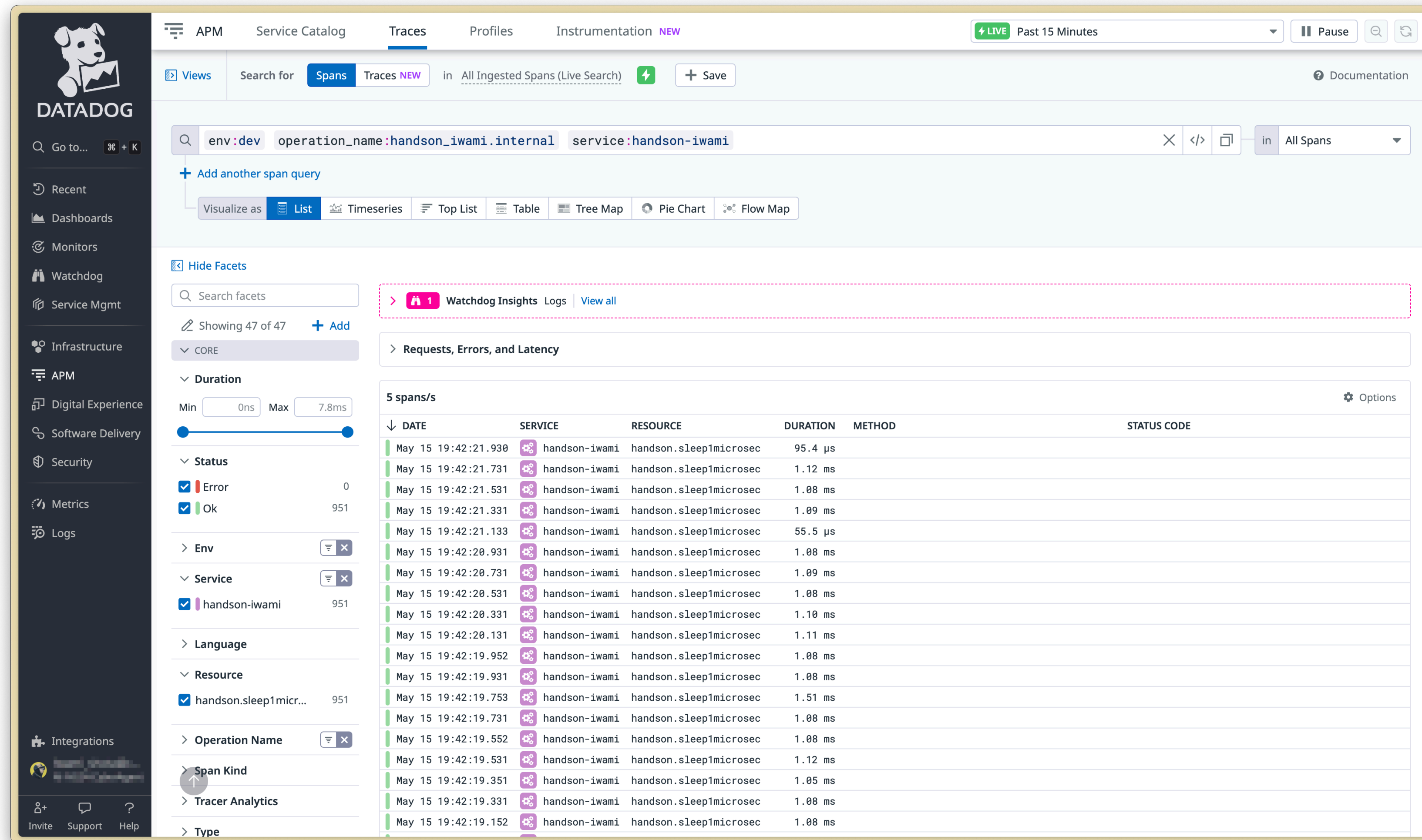
- APM Services (1h Past 1 Hour):** A table showing service metrics for the environment 'env:dev'.

★	TYPE	SERVICE	REQUESTS	P99 LATENCY	↓ ERROR RATE
☆	🔧	handson-iwami	0.2 req/s	1.14 ms	—
- Watchdog (2d Past 2 Days):** A message indicating that Watchdog hasn't detected any unusual behavior for the environment 'env:dev'.
- Issues:** A section for viewing issue details and error counts.
- Dashboards:** A list of available dashboards including Host Counts, Containers - Overview, Amazon Fargate Overview, Amazon EBS, Amazon DynamoDB, Amazon Data Firehose, Amazon CodeWhisperer, and Amazon API Gateway.

Copyright Datadog, Inc. 2024 - 35.34335801 - Master Subscription Agreement - Privacy Policy - Cookie Policy - Datadog Status → ● All Systems Operational

Datadog の APM

- Traceに入っている



The screenshot displays the Datadog APM interface. The top navigation bar includes 'APM', 'Service Catalog', 'Traces', 'Profiles', and 'Instrumentation'. A search bar at the top contains the query: `env:dev operation_name:handson_iwami.internal service:handson-iwami`. Below the search bar, there are options to 'Visualize as' (List, Timeseries, Top List, Table, Tree Map, Pie Chart, Flow Map) and a 'Hide Facets' section.

The 'Hide Facets' section shows the following filters:

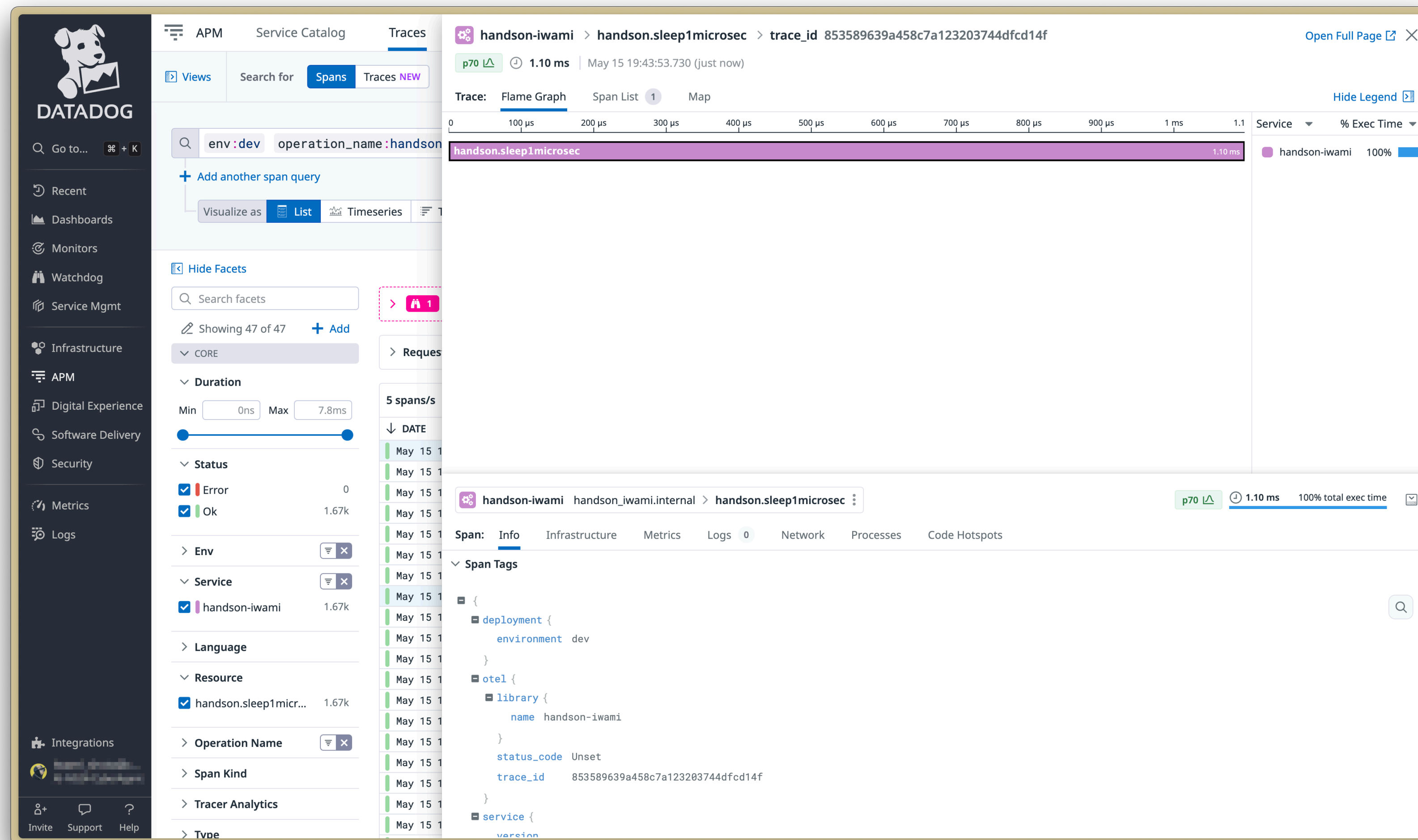
- Showing 47 of 47
- CORE
- Duration: Min 0ns, Max 7.8ms
- Status: Error (0), Ok (951)
- Env: (empty)
- Service: handson-iwami (951)
- Language: (empty)
- Resource: handson.sleep1micr... (951)
- Operation Name: (empty)
- Span Kind: (empty)
- Tracer Analytics: (empty)
- Type: (empty)

The main content area displays a table of spans with the following columns: DATE, SERVICE, RESOURCE, DURATION, METHOD, and STATUS CODE. The table shows 5 spans per second.

DATE	SERVICE	RESOURCE	DURATION	METHOD	STATUS CODE
May 15 19:42:21.930	handson-iwami	handson.sleep1microsec	95.4 μs		
May 15 19:42:21.731	handson-iwami	handson.sleep1microsec	1.12 ms		
May 15 19:42:21.531	handson-iwami	handson.sleep1microsec	1.08 ms		
May 15 19:42:21.331	handson-iwami	handson.sleep1microsec	1.09 ms		
May 15 19:42:21.133	handson-iwami	handson.sleep1microsec	55.5 μs		
May 15 19:42:20.931	handson-iwami	handson.sleep1microsec	1.08 ms		
May 15 19:42:20.731	handson-iwami	handson.sleep1microsec	1.09 ms		
May 15 19:42:20.531	handson-iwami	handson.sleep1microsec	1.08 ms		
May 15 19:42:20.331	handson-iwami	handson.sleep1microsec	1.10 ms		
May 15 19:42:20.131	handson-iwami	handson.sleep1microsec	1.11 ms		
May 15 19:42:19.952	handson-iwami	handson.sleep1microsec	1.08 ms		
May 15 19:42:19.931	handson-iwami	handson.sleep1microsec	1.08 ms		
May 15 19:42:19.753	handson-iwami	handson.sleep1microsec	1.51 ms		
May 15 19:42:19.731	handson-iwami	handson.sleep1microsec	1.08 ms		
May 15 19:42:19.552	handson-iwami	handson.sleep1microsec	1.08 ms		
May 15 19:42:19.531	handson-iwami	handson.sleep1microsec	1.12 ms		
May 15 19:42:19.351	handson-iwami	handson.sleep1microsec	1.05 ms		
May 15 19:42:19.331	handson-iwami	handson.sleep1microsec	1.08 ms		
May 15 19:42:19.152	handson-iwami	handson.sleep1microsec	1.08 ms		

Datadog の APM

- Spanをうまく計装できていれば `handson.sleep50microsec` などが表示されるはず



Spanの計装

- こんな感じに入れていれば正解

```
func (h *Handler) sleep50microsec(ctx context.Context) {  
    // NOTE: OpenTelemetry span  
    ctx, span := tracer.StartSpan(ctx, "handson.sleep50microsec")  
    defer span.End()  
  
    time.Sleep(50 * time.Microsecond)  
}
```

OpenTelemetry Registry を使って
計装してみよう

net/http

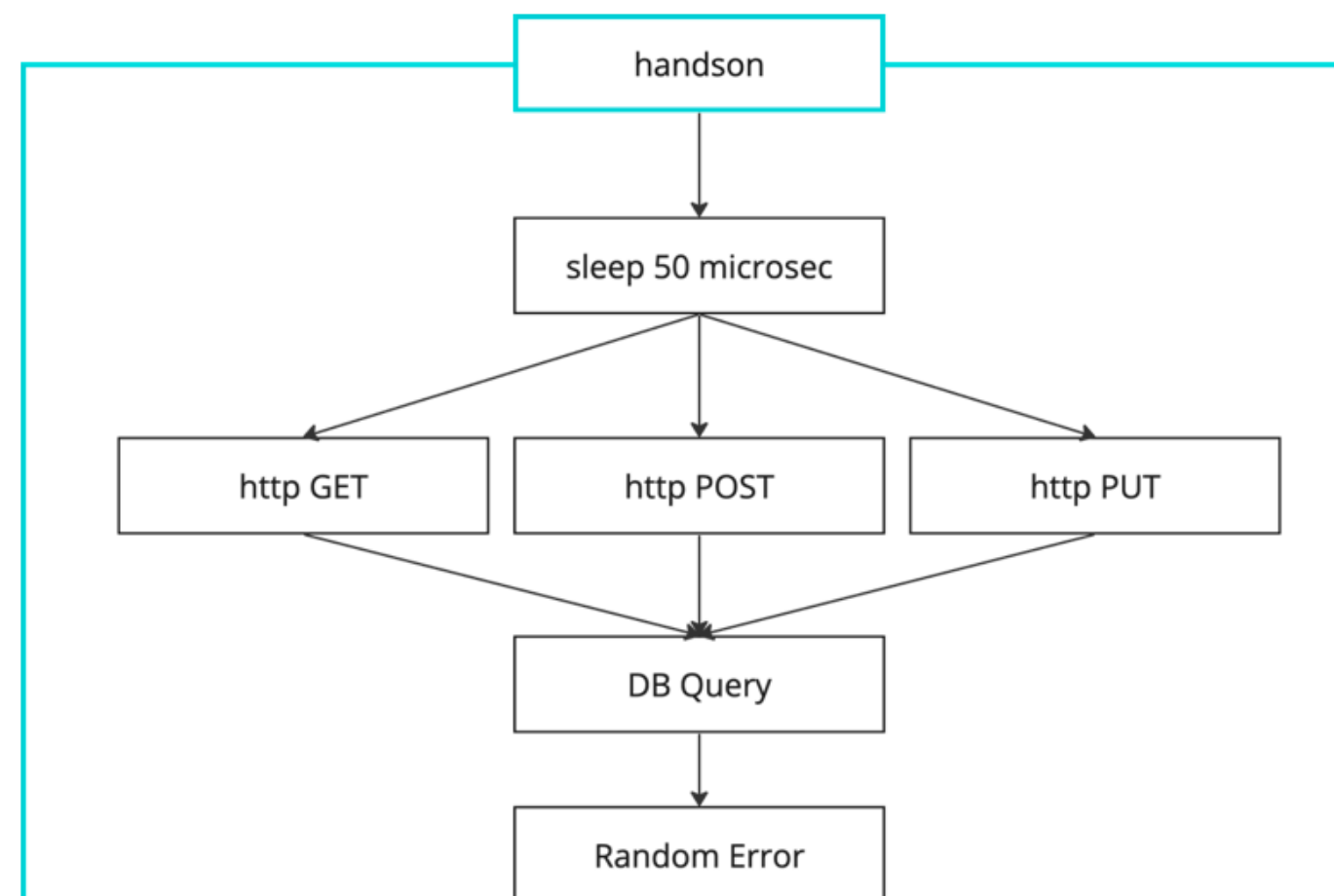
- handson api は net/http を用いて作られている
- この API のトレースを取りたい

net/http

- <https://httpbin.org> というモックサーバに net/http を使用してアクセスしている

/handson

- 一つだけエンドポイントが生えてる



```
func (h *Handler) handson(w http.ResponseWriter, r *http.Request) (int, error) {
    ctx := r.Context()

    // Custom Function
    h.sleep50microsec(ctx)

    // External API Call
    if err := h.restHttpBin(ctx, http.MethodGet, "get"); err != nil {
        return http.StatusInternalServerError, err
    }

    // External API Call concurrently
    errGroup, errCtx := errgroup.WithContext(ctx)

    errGroup.Go(func() error {
        return h.restHttpBin(errCtx, http.MethodGet, "get")
    })

    errGroup.Go(func() error {
        return h.restHttpBin(errCtx, http.MethodPost, "post")
    })

    errGroup.Go(func() error {
        return h.restHttpBin(errCtx, http.MethodPut, "put")
    })

    if err := errGroup.Wait(); err != nil {
        return http.StatusInternalServerError, err
    }

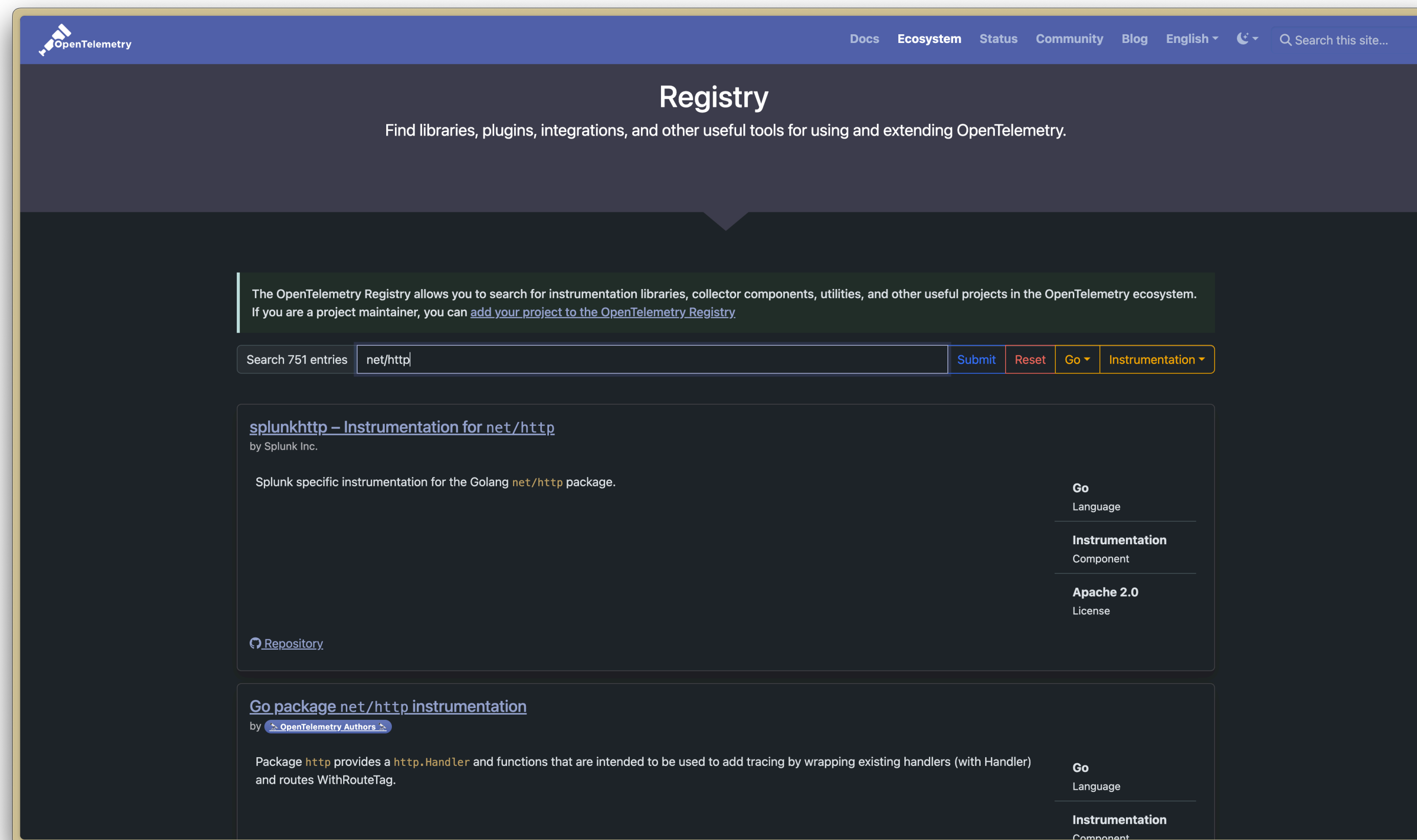
    // Database Query
    if err := h.dbQuery(ctx); err != nil {
        return http.StatusInternalServerError, err
    }

    // Random Error
    if err := h.randomError(radomErrorRate); err != nil {
        return http.StatusInternalServerError, err
    }

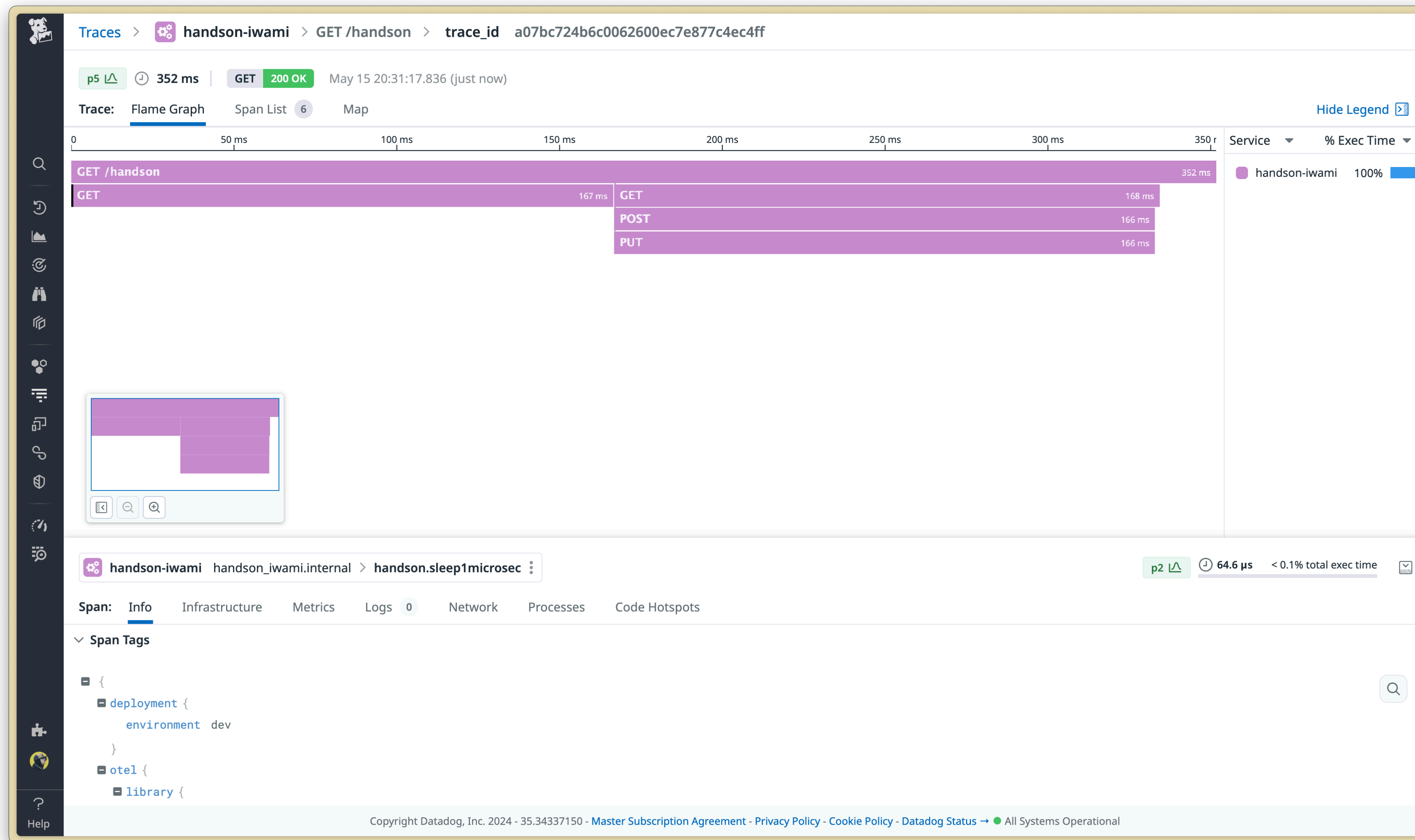
    return http.StatusOK, nil
}
```

Registry で検索して計装してみよう

- 言語に Go、検索欄に net/http など入れて検索してみよう
- backend-{sukina-name}/pkg/http/httpserver/server.go あたり



計装できているとこんな感じになる



net/http

The screenshot shows the Go package documentation page for `net/http`. The page is dark-themed and features a navigation bar at the top with the Go logo, a search bar, and links for "Why Go", "Learn", "Docs", "Packages", and "Community". The breadcrumb trail indicates the package path: `go.opentelemetry.io/contrib/instrumentation/net/http`. The package name `http` is highlighted, with "package" and "module" tabs. Metadata includes the version `v0.11.0` (marked as "Latest"), published on Aug 25, 2020, under the Apache-2.0 license, with 12 imports and 3 imported by.

The main content area is divided into three sections:

- Documentation:** A sidebar on the left with a "Jump to ..." search box and a list of navigation items: Overview, Index, Constants, Variables, Functions, Types, Source Files, and Directories.
- Overview:** A paragraph stating: "Package `http` provides a `http.Handler` and functions that are intended to be used to add tracing by wrapping existing handlers (with `Handler`) and routes `WithRouteTag`."
- Index:** A list of symbols with their signatures:
 - Constants
 - `func NewHandler(handler http.Handler, operation string, opts ...Option) http.Handler`
 - `func WithRouteTag(route string, h http.Handler) http.Handler`
 - `type Config`
 - `func NewConfig(opts ...Option) *Config`
 - `type Filter`
 - `type Handler`
 - `func (h *Handler) ServeHTTP(w http.ResponseWriter, r *http.Request)`
 - `type Option`
 - `func WithFilter(f Filter) Option`
 - `func WithMessageEvents(events ...event) Option`
 - `func WithMeter(meter metric.Meter) Option`
 - `func WithPropagators(ps propagation.Propagators) Option`
 - `func WithPublicEndpoint() Option`
 - `func WithSpanNameFormatter(f func(operation string, r *http.Request) string) Option`
 - `func WithSpanOptions(opts ...trace.StartOption) Option`
 - `func WithTracer(tracer trace.Tracer) Option`
 - `type OptionFunc`
 - `func (o OptionFunc) Apply(c *Config)`
 - `type Transport`
 - `func NewTransport(base http.RoundTripper, opts ...Option) *Transport`
 - `func (t *Transport) RoundTrip(r *http.Request) (*http.Response, error)`
- Examples:** A list of links for `NewHandler` and `NewTransport`.
- Details:** A sidebar on the right showing package quality checks: Valid `go.mod` file, Redistributable license, Tagged version, and Stable version. It also includes a link to "Learn more about best practices", a "Repository" link to `github.com/open-telemetry/opentelemetry-g...`, and a "Links" section with "Open Source Insights".

net/http

```
// backend-{sukina-name}/pkg/http/httpserver/server.go

func NewServer(port int, gracePeriod time.Duration, logger slog.Logger) *Server {
    s := &Server{
        ..
    }

    s.mux = http.NewServeMux()

    // s.server.Handler = s.mux
    s.server.Handler = otelhttp.NewHandler(s.mux, "httpServer")

    return s
}
```

net/http

```
// backend-{sukina-name}/pkg/http/httpserver/server.go

func (s *Server) HandleFunc(...) {
    // s.mux.Handle(pattern, http.HandlerFunc(handler))
    otelHandler := otelhttp.WithRouteTag(pattern, http.HandlerFunc(handler))
    s.mux.Handle(pattern, otelHandler)
}
```

net/http

```
// backend-{sukina-name}/app/api/cmd.go

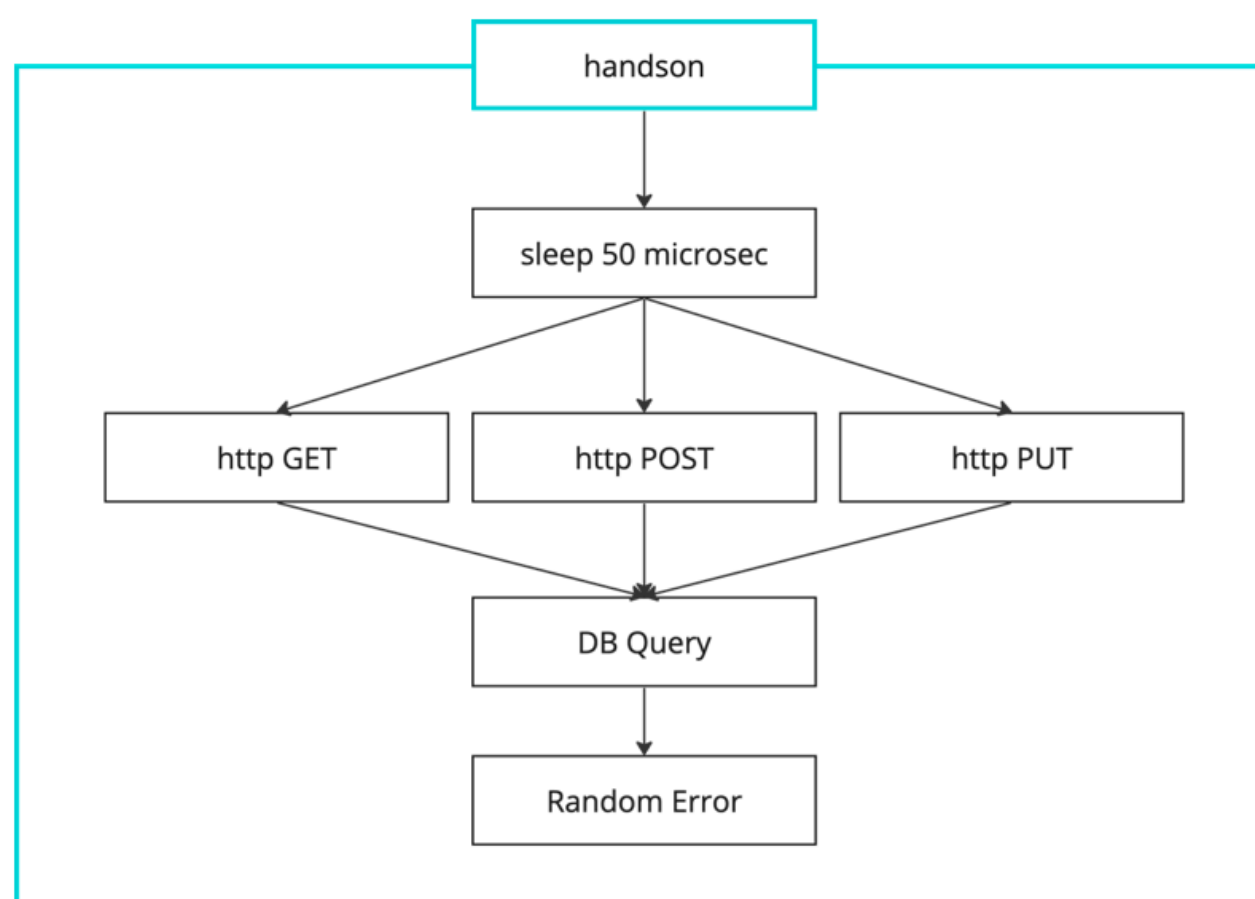
httpClient := &http.Client{
    // NOTE: OpenTelemetry instrumentation
    Transport: otelhttp.NewTransport(http.DefaultTransport),
}
```

database/sql

- sqlite3 に対してアクセスしている
- これも同じように計装してみよう

/handson

- 一つだけエンドポイントが生えてる



```
func (h *Handler) handson(w http.ResponseWriter, r *http.Request) (int, error) {
    ctx := r.Context()

    // Custom Function
    h.sleep50microsec(ctx)

    // External API Call
    if err := h.restHttpBin(ctx, http.MethodGet, "get"); err != nil {
        return http.StatusInternalServerError, err
    }
    // External API Call concurrently
    errGroup, errCtx := errgroup.WithContext(ctx)

    errGroup.Go(func() error {
        return h.restHttpBin(errCtx, http.MethodGet, "get")
    })
    errGroup.Go(func() error {
        return h.restHttpBin(errCtx, http.MethodPost, "post")
    })
    errGroup.Go(func() error {
        return h.restHttpBin(errCtx, http.MethodPut, "put")
    })

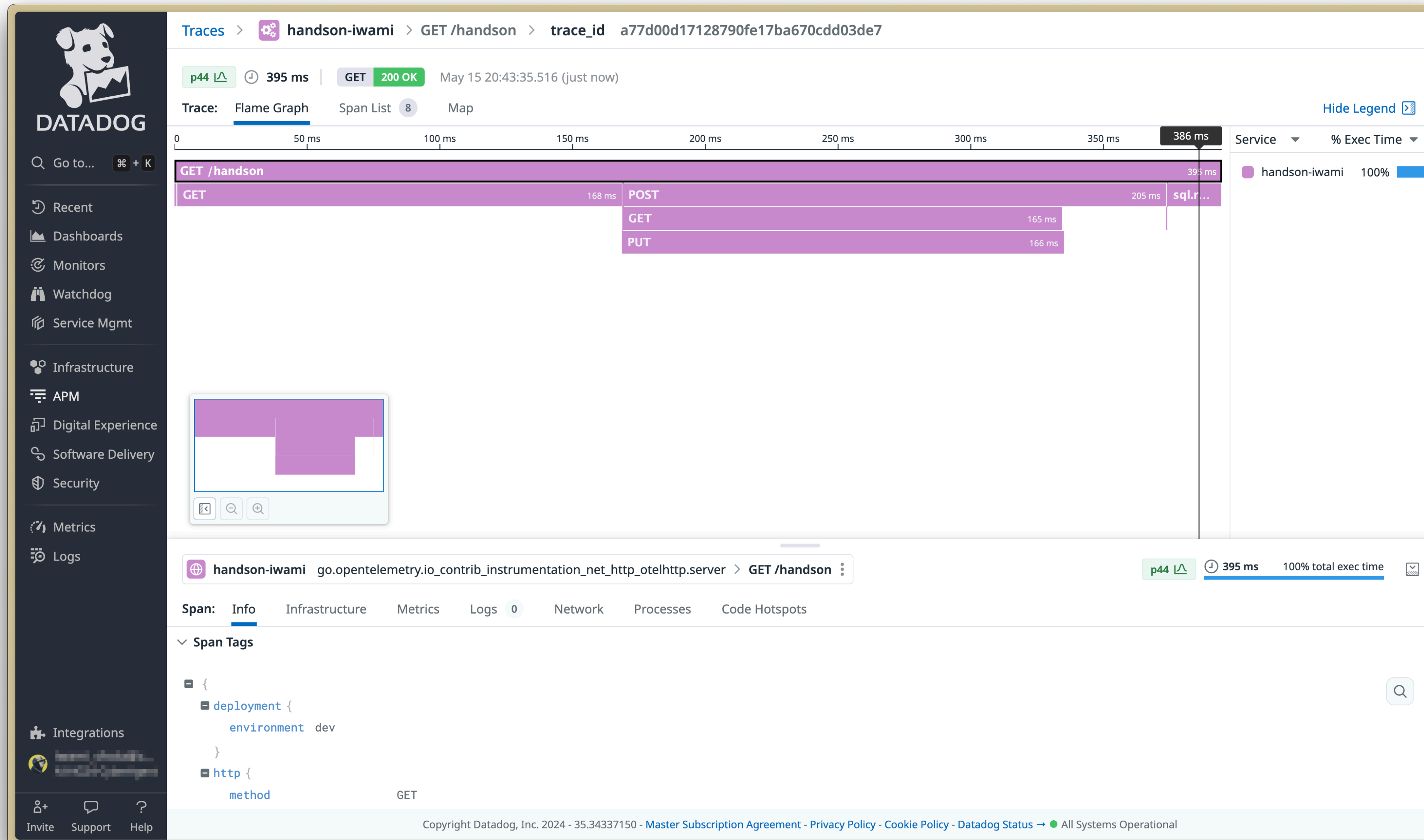
    if err := errGroup.Wait(); err != nil {
        return http.StatusInternalServerError, err
    }

    // Database Query
    if err := h.dbQuery(ctx); err != nil {
        return http.StatusInternalServerError, err
    }

    // Random Error
    if err := h.randomError(radomErrorRate); err != nil {
        return http.StatusInternalServerError, err
    }

    return http.StatusOK, nil
}
```

計装できているとこんな感じになる



database/sql

```
func Init(ctx context.Context, logger slog.Logger) (*sql.DB, error) {
    ...

    // db, err := sql.Open("sqlite3", "file::memory:?cache=shared")
    db, err := otelsql.Open(
        "sqlite3",
        "file::memory:?cache=shared",
        otelsql.WithAttributes(semconv.DBSystemSqlite),
    )
    if err != nil {
        return nil, cerror.Wrap(err, "failed to open sqlite")
    }

    ...

    return db, nil
}
```

ログとトレースを紐付けよう

Span ID/Trace ID

- ログに Span ID、Trace ID を仕込む
- ログが発生した際にどこの Trace かがわかるようになる
- backend-{sukina-name}/app/api/handson/log.go に追加してみよう
- Context の中に trace 情報が入っているので取り出して使う

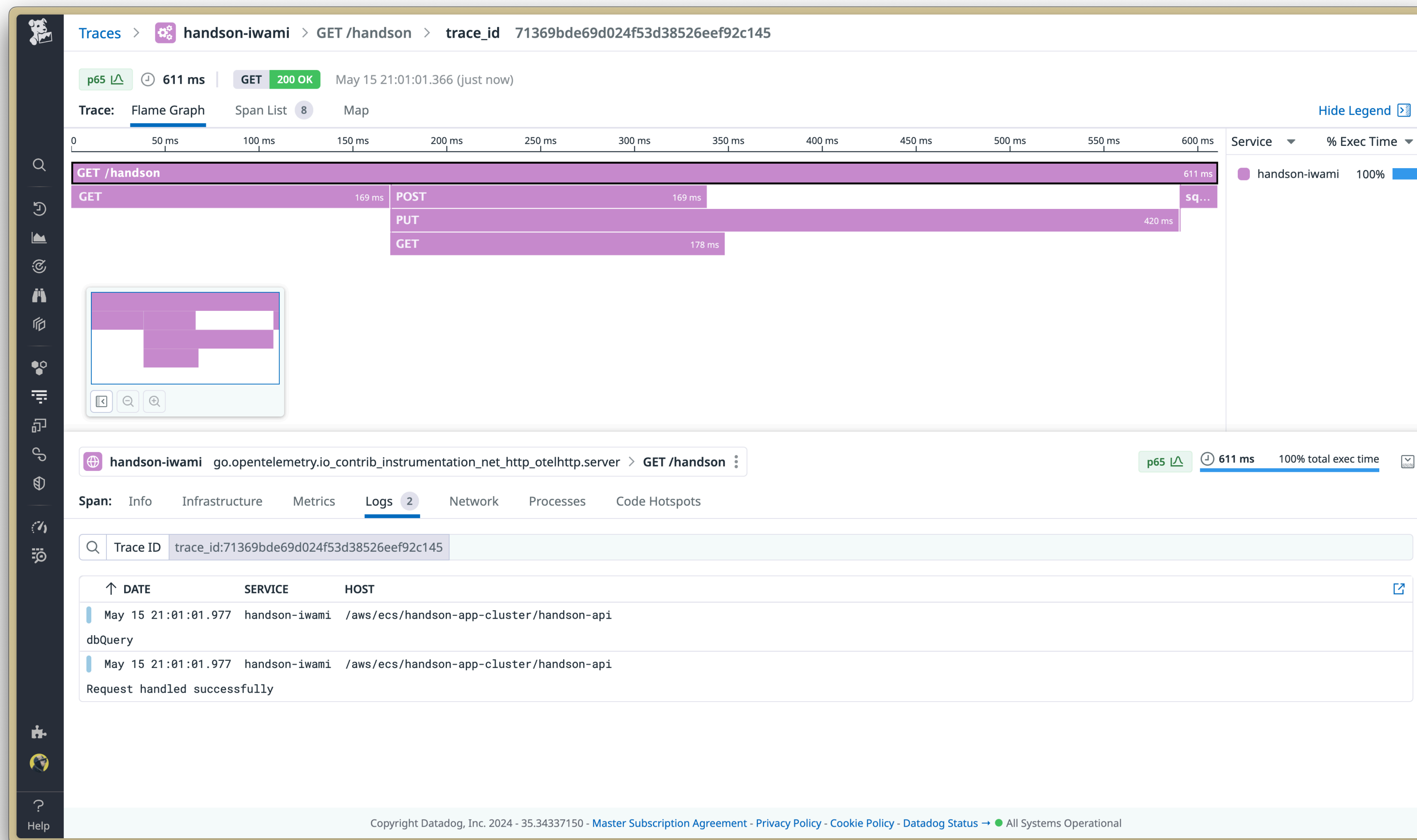
```
func LogAttributes(ctx context.Context, attrs []any) []any {
    span := trace.SpanFromContext(ctx)
    lattrs := []any{
        slog.String("env", "dev"),
        slog.String("service", version.Get().ServiceName),
        slog.String("version", version.Get().Version),
        slog.String("git.commit.sha", version.Get().GitCommit),
        slog.String("git.repository",
            version.Get().GitRepository),
        slog.String("trace.id",
            span.SpanContext().TraceID().String()),
        slog.String("span.id",
            span.SpanContext().SpanID().String()),

        slog.String("dd.service", version.Get().ServiceName),
        slog.String("dd.version", version.Get().Version),
        slog.String("dd.env", "dev"),
        slog.String("dd.git.commit.sha",
            version.Get().GitCommit),
        slog.String("dd.git.repository",
            version.Get().GitRepository),
        slog.String("dd.trace_id",
            span.SpanContext().TraceID().String()),
        slog.String("dd.span_id",
            span.SpanContext().SpanID().String()),
    }

    return append(lattrs, attrs...)
}
```

Trace → Log

- Trace の下に Log が出てる



The screenshot displays the Datadog Traces interface for a specific trace ID: 71369bde69d024f53d38526eef92c145. The trace is for a GET request to /handson, which took 611 ms and returned a 200 OK status. The trace is visualized as a flame graph, showing the following spans:

- GET /handson (611 ms)
- GET (169 ms)
- POST (169 ms)
- PUT (420 ms)
- GET (178 ms)

The right sidebar shows the service 'handson-iwami' with 100% execution time. Below the flame graph, the 'Logs' tab is selected, showing two log entries for the span:

DATE	SERVICE	HOST
May 15 21:01:01.977	handson-iwami	/aws/ecs/handson-app-cluster/handson-api
May 15 21:01:01.977	handson-iwami	/aws/ecs/handson-app-cluster/handson-api

The log messages are:

- dbQuery
- Request handled successfully

The footer of the interface includes the Datadog logo, a help icon, and the following text: Copyright Datadog, Inc. 2024 - 35.34337150 - Master Subscription Agreement - Privacy Policy - Cookie Policy - Datadog Status → All Systems Operational

Log → Trace

- Log の画面で Trace がみれる

The screenshot displays the Datadog interface for logs and traces. On the left, the 'Logs' view is active, showing a search for `trace_id:71369bde69d024f53d38526eef92c145`. The search results show 2 logs found. A log entry is highlighted with a red dashed box, containing the text `Request handled successfully`. The log entry is associated with the service `handson-iwami` and the host `/aws/ecs/handson-app-clu...`. The log entry is also associated with the tag `env:dev`.

On the right, the 'Trace' view is active, showing the details of the trace for the log entry. The trace is for the endpoint `GET /handson` and has a status of `200 OK` with a response time of `611 ms`. The trace is associated with the service `handson-iwami` and the host `/aws/ecs/handson-app-clu...`. The trace is also associated with the tag `env:dev`.

The trace details show a waterfall chart of the request and response times for various services. The services and their execution times are:

Service	% Exec Time	Execution Time
handson-iwami	100%	611 ms
GET		169 ms
POST		169 ms
PUT		420 ms
GET		178 ms

エラーのアラート飛ばしてみよう

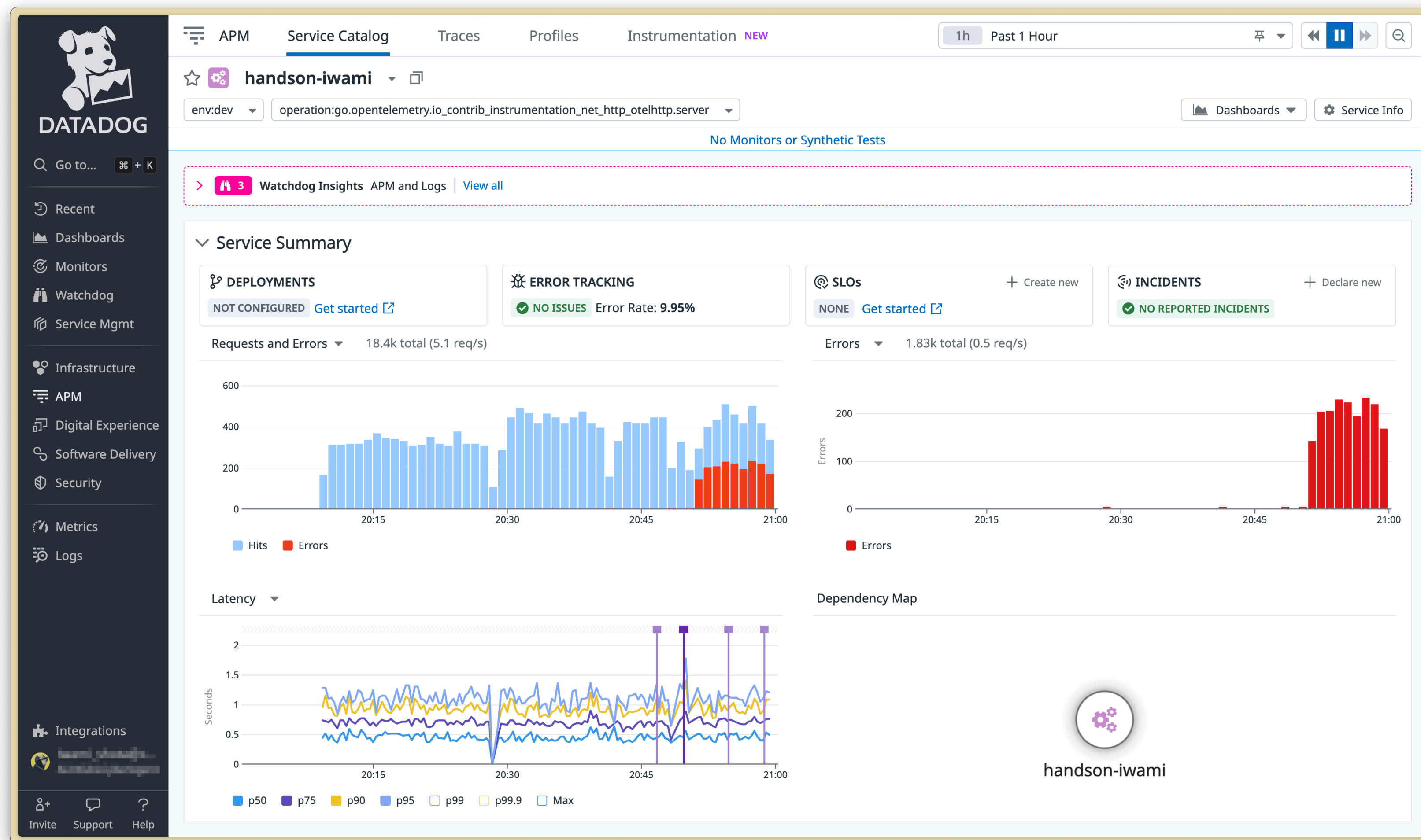
randomError

- backend-{sukina-name}/app/
api/handson/handson.go で一定
確率で Error が発生する function
がある
- この rate を 0.5 にしてエラーを発生
させてみよう

```
const (  
    // radomErrorRate = 0.0  
    radomErrorRate = 0.5  
)  
  
func (h *Handler) randomError(rate float64) error {  
    // TODO: Instrument OpenTelemetry span  
    if rate == 0 {  
        return nil  
    }  
  
    if rand.Float64() < rate {  
        return cerror.New("random error")  
    }  
  
    return nil  
}
```

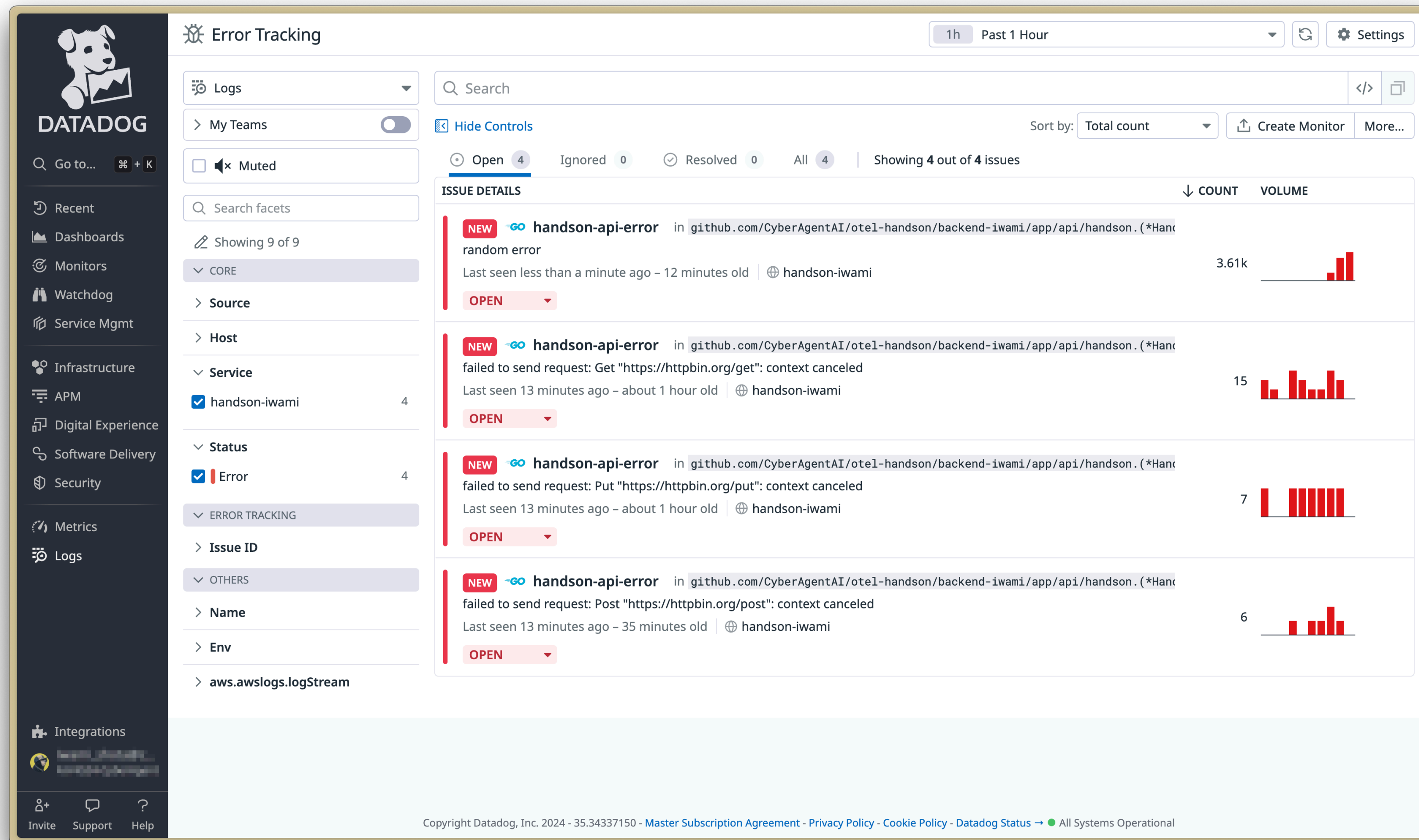
APM

- Error が出ているのが分かるはず



APM Error Tracking

- Error が Issue になっているのが分かる



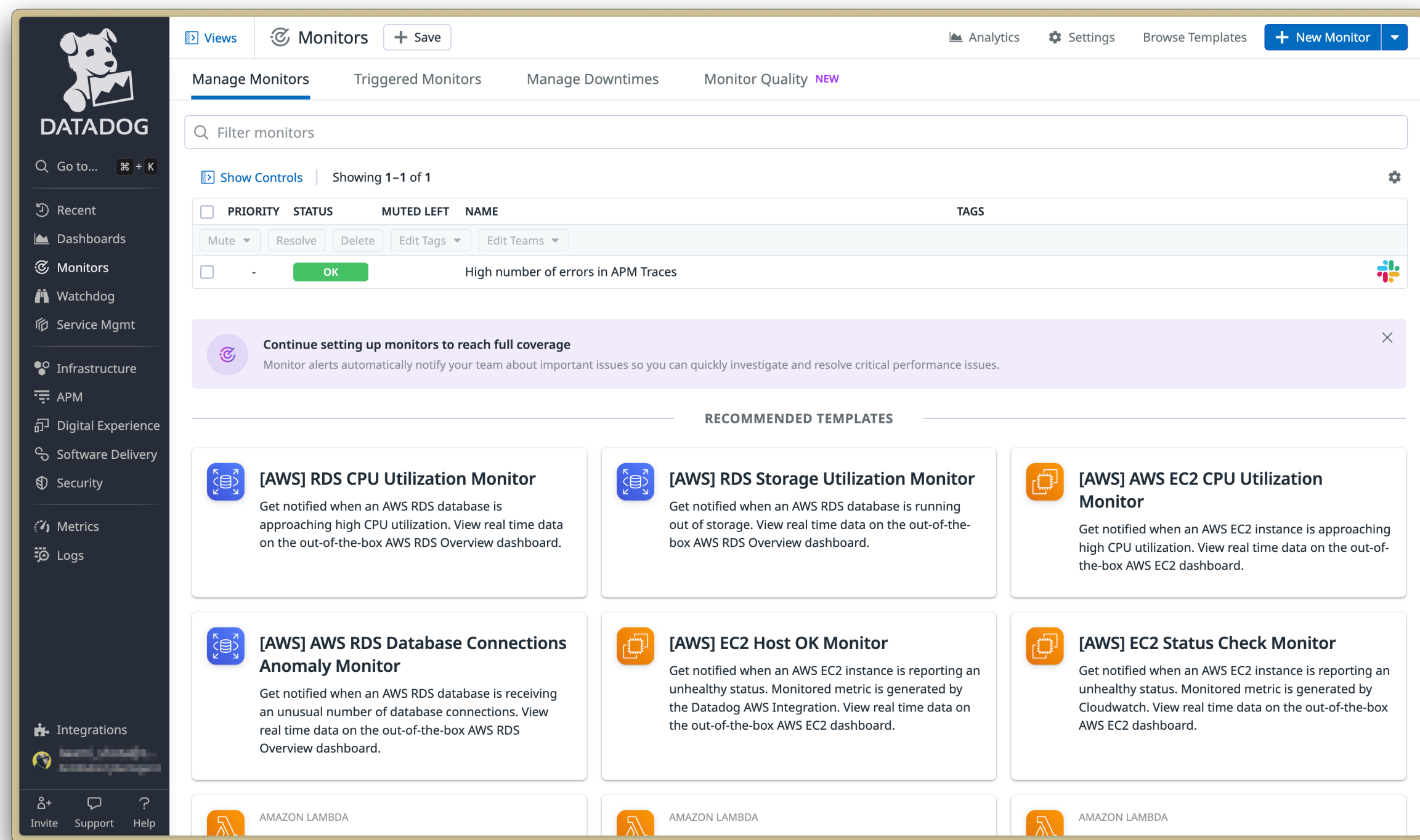
The screenshot displays the Datadog Error Tracking dashboard. The left sidebar contains navigation options like Recent, Dashboards, Monitors, Watchdog, Service Mgmt, Infrastructure, APM, Digital Experience, Software Delivery, Security, Metrics, and Logs. The main content area shows a list of error issues with columns for COUNT and VOLUME. The issues are categorized by status (NEW, OPEN) and include details such as the error message, source, and last seen time.

Issue ID	Count	Volume
handson-api-error (random error)	3.61k	3.61k
handson-api-error (failed to send request: Get "https://httpbin.org/get": context canceled)	15	15
handson-api-error (failed to send request: Put "https://httpbin.org/put": context canceled)	7	7
handson-api-error (failed to send request: Post "https://httpbin.org/post": context canceled)	6	6

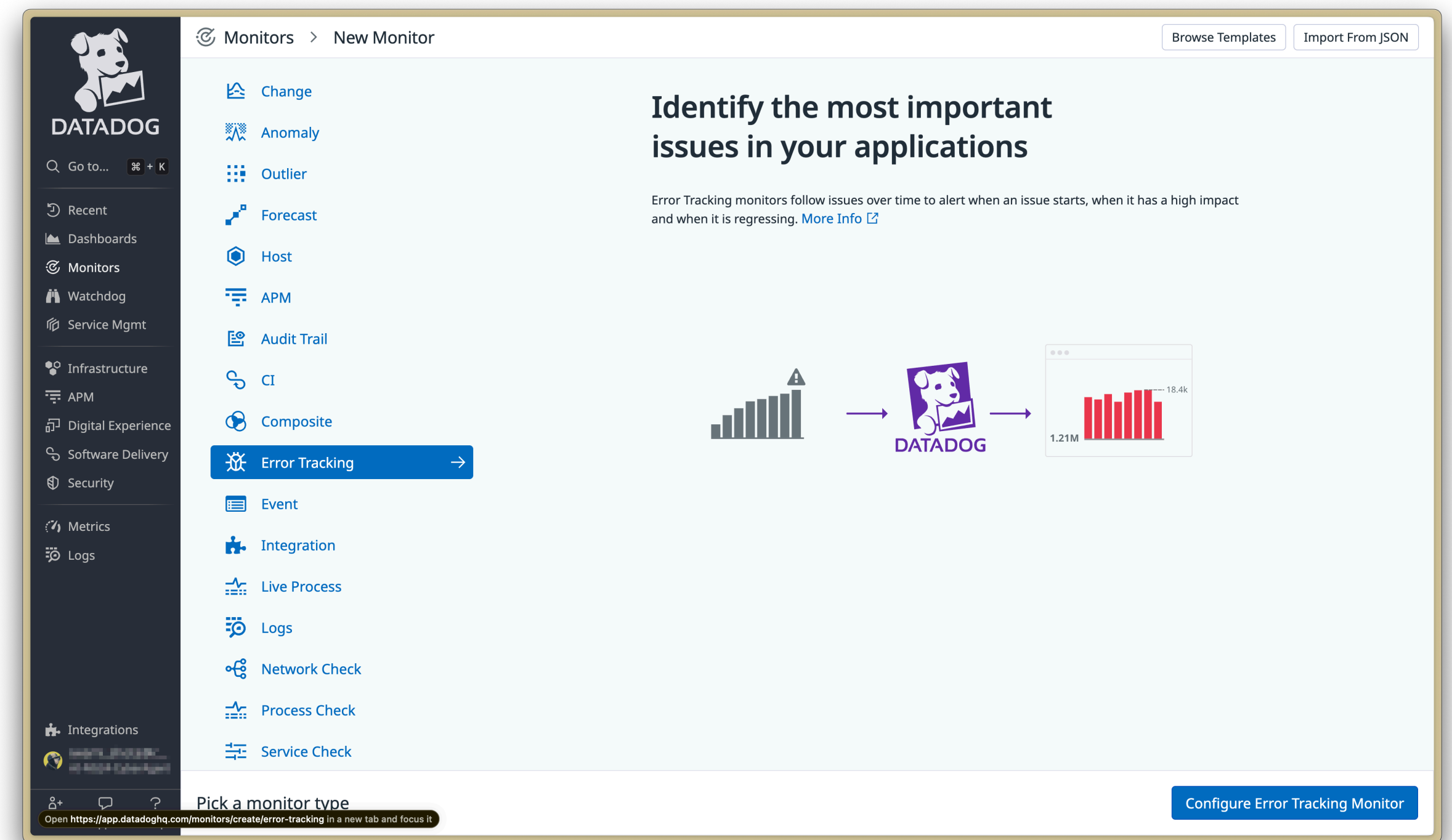
Copyright Datadog, Inc. 2024 - 35.34337150 - Master Subscription Agreement - Privacy Policy - Cookie Policy - Datadog Status → All Systems Operational

Datadog の Monitors

- Error Tracking をベースに Monitor を作成してみよう
- チャンネルは [#ai_kenshu24_o11y_handson](#) に飛ばしてみよう



The screenshot shows the Datadog Monitors page. The top navigation bar includes 'Views', 'Monitors', and '+ Save'. Below the navigation bar, there are tabs for 'Manage Monitors', 'Triggered Monitors', 'Manage Downtimes', and 'Monitor Quality'. A search bar for 'Filter monitors' is present. A table shows one monitor with a status of 'OK' and the name 'High number of errors in APM Traces'. Below the table, there is a notification to 'Continue setting up monitors to reach full coverage'. The main content area is titled 'RECOMMENDED TEMPLATES' and features six cards for different AWS services: [AWS] RDS CPU Utilization Monitor, [AWS] RDS Storage Utilization Monitor, [AWS] AWS EC2 CPU Utilization Monitor, [AWS] AWS RDS Database Connections Anomaly Monitor, [AWS] EC2 Host OK Monitor, and [AWS] EC2 Status Check Monitor. Each card provides a brief description of the monitor's function.

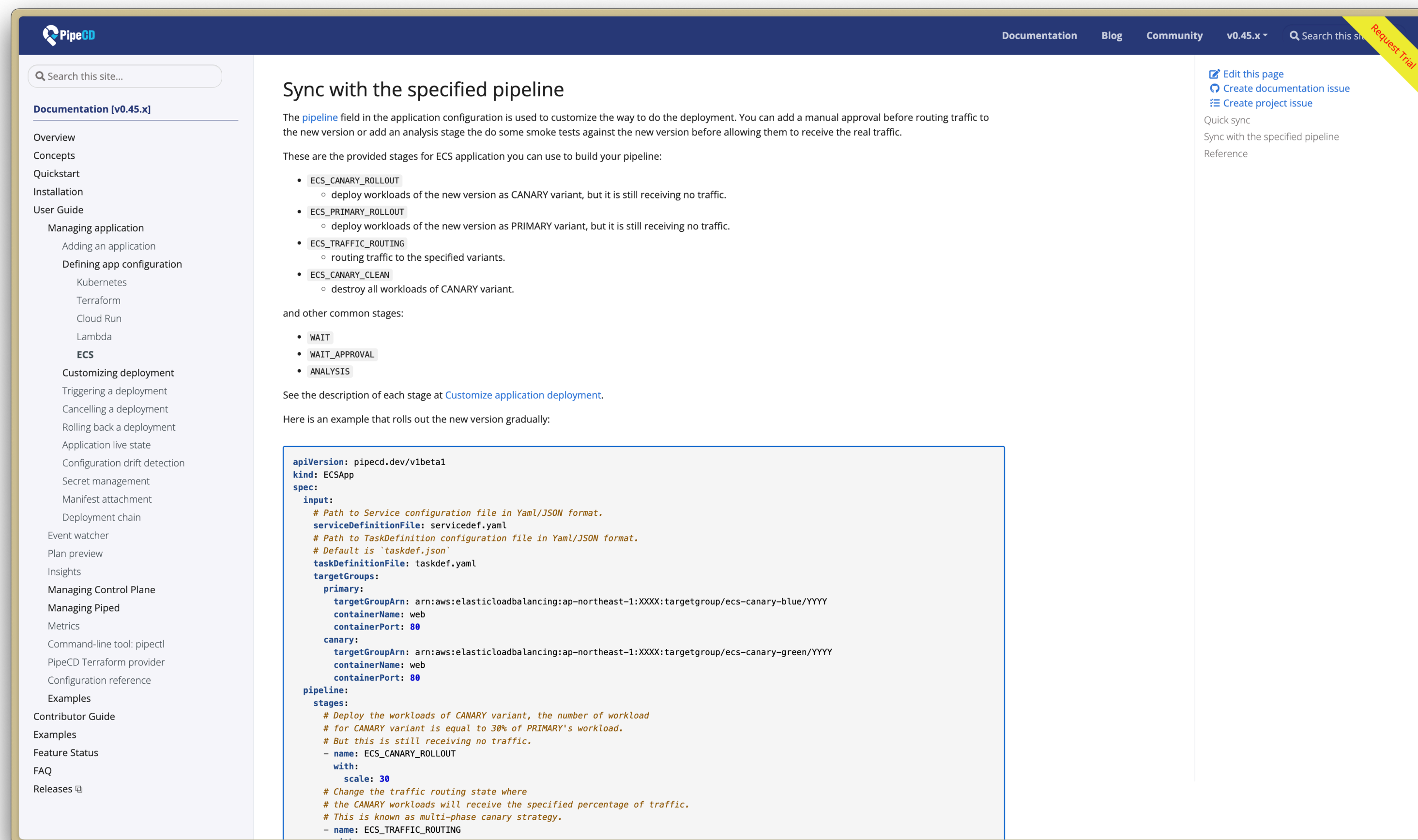


The screenshot shows the 'New Monitor' page in Datadog. The left sidebar lists various monitor types, with 'Error Tracking' highlighted in blue. The main content area is titled 'Identify the most important issues in your applications' and includes a sub-header 'Error Tracking monitors follow issues over time to alert when an issue starts, when it has a high impact and when it is regressing.' Below this, there is a diagram showing a bar chart with an upward arrow, the Datadog logo, and another bar chart with a value of 1.21M. At the bottom, there is a 'Pick a monitor type' section and a 'Configure Error Tracking Monitor' button.

以下時間が余った人

PipeCD のカナリアリリース

- ドキュメントを見ながら ECS にカナリアリリースを導入してみよう



The screenshot shows the PipeCD documentation page for "Sync with the specified pipeline". The page is in Japanese and provides instructions on how to configure a pipeline for a canary release on ECS. The main content includes a list of stages and a code example for a pipeline configuration.

Sync with the specified pipeline

The `pipeline` field in the application configuration is used to customize the way to do the deployment. You can add a manual approval before routing traffic to the new version or add an analysis stage to do some smoke tests against the new version before allowing them to receive the real traffic.

These are the provided stages for ECS application you can use to build your pipeline:

- `ECS_CANARY_ROLLOUT`
 - deploy workloads of the new version as CANARY variant, but it is still receiving no traffic.
- `ECS_PRIMARY_ROLLOUT`
 - deploy workloads of the new version as PRIMARY variant, but it is still receiving no traffic.
- `ECS_TRAFFIC_ROUTING`
 - routing traffic to the specified variants.
- `ECS_CANARY_CLEAN`
 - destroy all workloads of CANARY variant.

and other common stages:

- `WAIT`
- `WAIT_APPROVAL`
- `ANALYSIS`

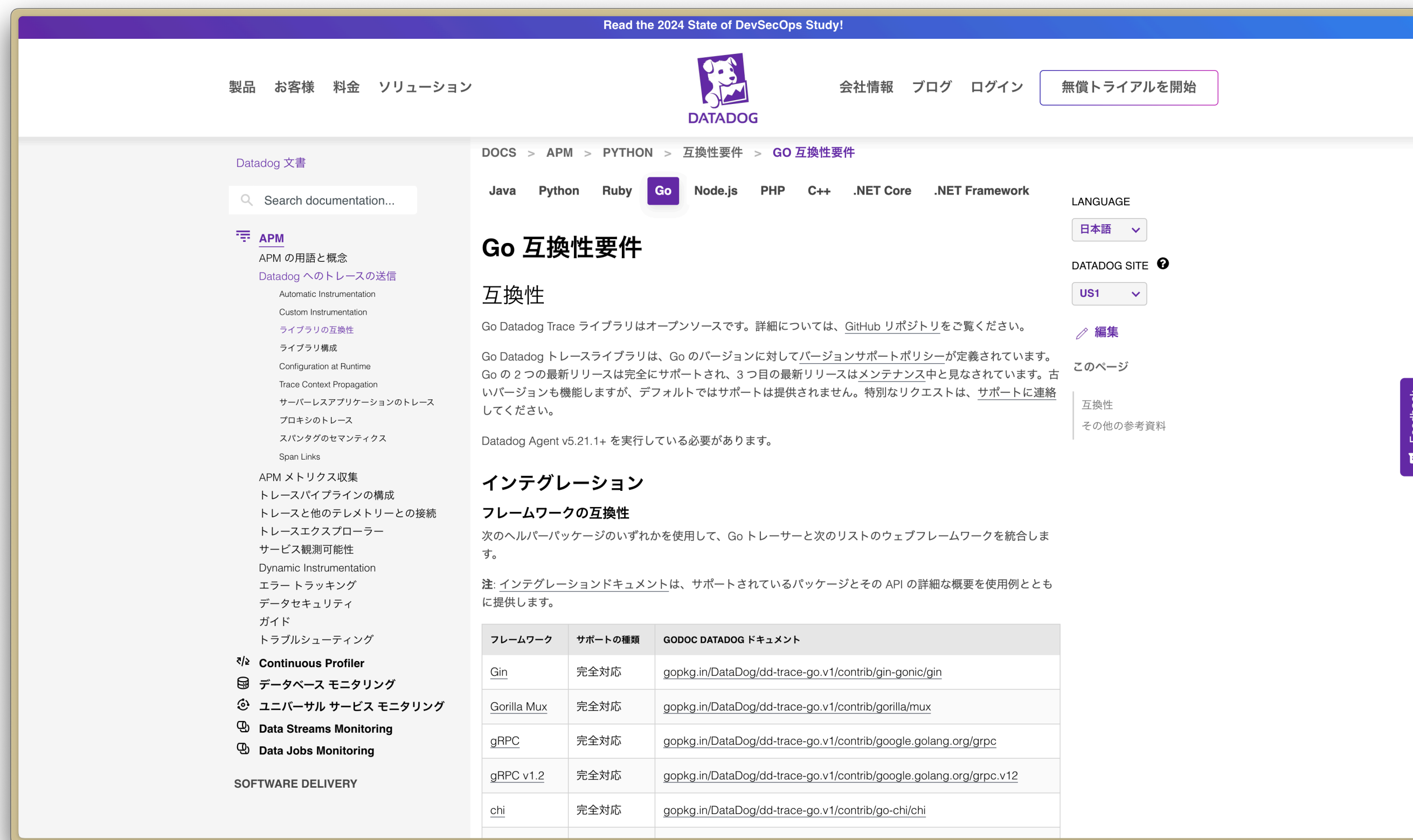
See the description of each stage at [Customize application deployment](#).

Here is an example that rolls out the new version gradually:

```
apiVersion: pipecd.dev/v1beta1
kind: ECSApp
spec:
  input:
    # Path to Service configuration file in Yaml/JSON format.
    serviceDefinitionFile: servicedef.yaml
    # Path to TaskDefinition configuration file in Yaml/JSON format.
    # Default is `taskdef.json`
    taskDefinitionFile: taskdef.yaml
  targetGroups:
    primary:
      targetGroupArn: arn:aws:elasticloadbalancing:ap-northeast-1:XXXX:targetgroup/ecs-canary-blue/YYYY
      containerName: web
      containerPort: 80
    canary:
      targetGroupArn: arn:aws:elasticloadbalancing:ap-northeast-1:XXXX:targetgroup/ecs-canary-green/YYYY
      containerName: web
      containerPort: 80
  pipeline:
    stages:
      # Deploy the workloads of CANARY variant, the number of workload
      # for CANARY variant is equal to 30% of PRIMARY's workload.
      # But this is still receiving no traffic.
      - name: ECS_CANARY_ROLLOUT
        with:
          scale: 30
      # Change the traffic routing state where
      # the CANARY workloads will receive the specified percentage of traffic.
      # This is known as multi-phase canary strategy.
      - name: ECS_TRAFFIC_ROUTING
```

Datadog の SDK 使用した計装

- OpenTelemetry の SDK から Datadog の SDK に計装し直してみよう
- ベンダーロックインがどんなことか分かるが、同時に恩恵も分かるはず



まとめ

- オブザーバビリティは飽くなき探究心…
- コストとトレードオフしながら、システム完全理解になるようなオブザーバビリティをぜひ実践してみてください！



出典・参考資料

- [北海道旅行・北海道ツアーなら格安旅行のJ-TRIP](#)
- [航空機の運動方程式 土屋研究室](#)
- [オブザーバビリティ・エンジニアリング](#)
- [【書評】オブザーバビリティ・エンジニアリング | DevelopersIO](#)
- [tag-observability/whitepaper.md at main · cncf/tag-observability](#)
- [cndf/tag-Observability](#)
- [Jaeger documentation](#)
- [OpenTelemetry.Trace](#)
- [「CA.go ～ABEMAのGoを活用したFIFA ワールドカップ生中継の舞台裏～」を開催しました！ #CAgo | CyberAgent Developers Blog](#)
- [Registry | OpenTelemetry](#)
- [DataDog/orchestrion: A tool for adding instrumentation to Go cod](#)
- [O'Reilly Japan - 入門 eBPF](#)
- [OpenTelemetryを理解する 第2回: コアのコンポーネント | New Relic](#)
- [Collector | OpenTelemetry](#)
- [OpenTelemetry と Prometheus @AoTo0330 \(Kento Kimura\) - Qiita](#)
- [opentelemetry-go-contrib/instrumentation/README.md at main · open-telemetry/opentelemetry-go-contrib](#)
- [Datadog Go 互換性要件](#)
- [SRE Technology Map | 株式会社サイバーエージェント](#)
- [Datadog 認定資格 \(Datadog Fundamentals\) を取ってみた](#)

出典・参考資料

- [Datadog バックエンドエラーの追跡](#)
- [GitLab](#)
- [Overview | PipeCD](#)
- [OpenTelemetry入門 - zenn](#)
- [仕様と実装から理解するOpenTelemetryの全体像](#)
- [OpenTelemetryのTraceをGoで試してみる - Carpe Diem](#)
- [cilium/ebpf: ebpf-go is a pure-Go library to read, modify and load eBPF programs and attach them to various hooks in the Linux kernel.](#)
- [APM の用語と概念 Datadog](#)
- [OpenTelemetryとGrafanaでLogsとMetricsとTracesを接続する #grafana - Qiita](#)
- [eBPF Introduction - Speaker Deck](#)
- [The Datadog Learning Center](#)