マイクロサービスの運用・管理

Oracle Cloud Hangout Café #2

日本オラクル株式会社
ソリューション・エンジニアリング統括　ソリューションエンジニア

茂 こと（Koto Shigeru）

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

自己紹介

@cotoc88

- 茂 こと（しげる こと）
- 日本オラクルのソリューションエンジニア
  - Oracle Database 数年
  - Docker/Kubernetes 6ヶ月くらい
  - Vitess 3週間
- 仕事のモチベーションをあげるために自宅のデスクチェアを エルゴヒューマン（中古）にアップグレードしました

撮影：JapanContainerDays実行委員会

Vitess

ORACLE

Developer Summit 2019

2月14日 11:05～11:50

【14-B-2】

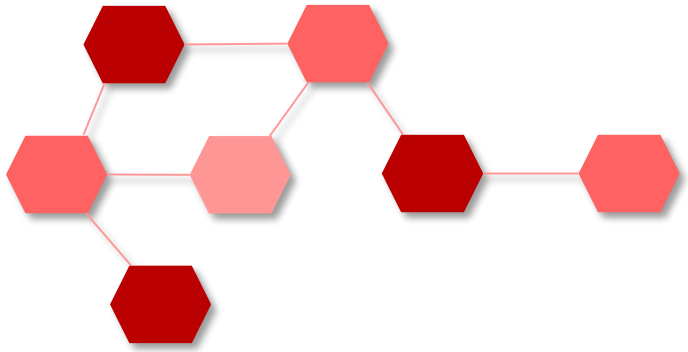
Cloud Native アプリケーションに最適！Oracle Cloud Infrastructureの魅力！

本日のテーマ:
マイクロサービスの運用・管理

マイクロサービスとは

目的・用途毎にアプリケーションを分離し、1つのシステムを構成するアーキテクチャ

- Monolithic
  - システムを単一のアプリケーションとして実装
    - 一部の変更が全体にどう影響するかが把握しにくい
    - 規模が大きくなると更新に時間がかかる
    - スケールアウトしづらい
    - コミュニケーションコスト高



- Microservices
  - 複数の小さなアプリケーション同士を連携させシステムを実装
    - 変更の影響範囲をサービスに留められる
    - 素早く更新できる
    - サービス単位でスケールアウトしやすい
    - 大規模開発しやすい

ORACLE

マイクロサービスに特化した問題

マイクロサービスはサービス同士をつなぐ“境界”が存在する

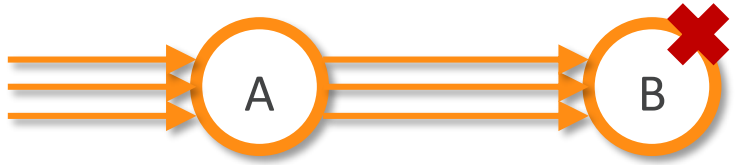
- 互いに疎結合なサービス同士が連携する箇所で発生する問題を、「サービス境界の問題」と呼ぶことにします
  - サービス同士の依存関係によっては、一箇所の障害が広範囲な障害に発展したり、また障害やパフォーマンス劣化の原因究明が複雑になりがちです

サービス境界の問題により起こる問題

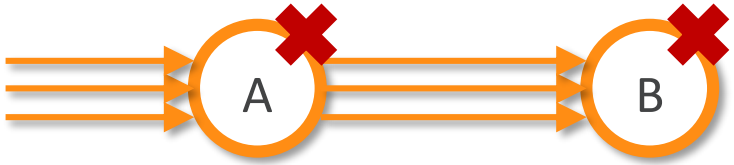
障害の連鎖によりシステムの広範囲が機能しなくなるケースも

1. サービスAが依存するサービスBで障害発生

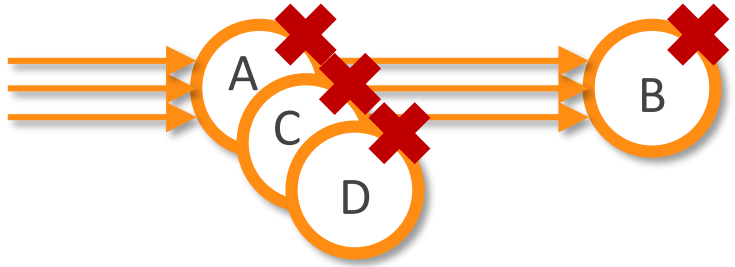
2. サービスAでサービスBの応答待ちのリクエストが累積



3. 応答待ちリクエストがAのリソースを専有し、Aも停止



4. Bへのアクセス集中が続き、Bが復旧困難に。Bに依存する他サービスもAと同様に次々停止



サービス境界の問題解決に向けた取り組み

- 影響を事前に把握する
  - カオスエンジニアリング（障害・パフォーマンス劣化の影響を知る）
  - カナリア・リリース（リリースの影響を知る）
- 監視する
  - 依存関係やデータフローを可視化する
  - ログ・メトリック可視化する
- 分析する
  - ログ・メトリックを収集する
  - 因果関係を追跡する

サービス境界の問題解決に向けた取り組み

	取り組み	活用できるツール （本日紹介するもの）
事前に把握する	カオスエンジニアリング カナリアリリース	Istio
監視する	ログ・メトリック可視化する 依存関係やデータフローを可視化する	Istio / Prometheus / Grafana / Kiali
分析する	ログ・メトリックを収集する 因果関係を追跡する	Istio / Prometheus / Jaeger / Kiali

Istio

サービス・メッシュを実現するOSS

- サービス・メッシュ
  - マイクロサービスのような複雑なサービス境界の問題を解消するネットワーキングモデル
- Istio
  - Google, IBM, Lyft社によって開発、2017年にオープンソース化された
    - IstioはCloud Native Computing Foundation(CNCF)がホストするプロジェクト
  - デファクトスタンダードになりつつある?

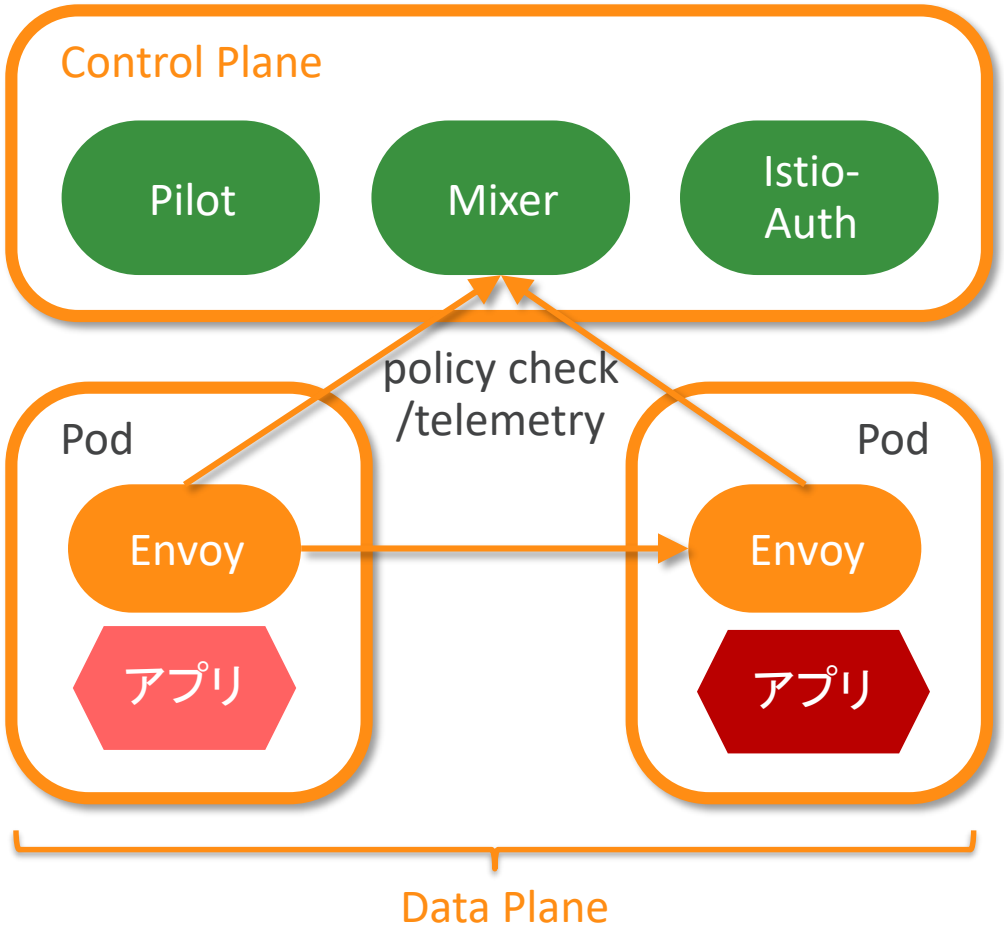
Istio

Connect, secure, control, and observe services.

Istioのアーキテクチャ

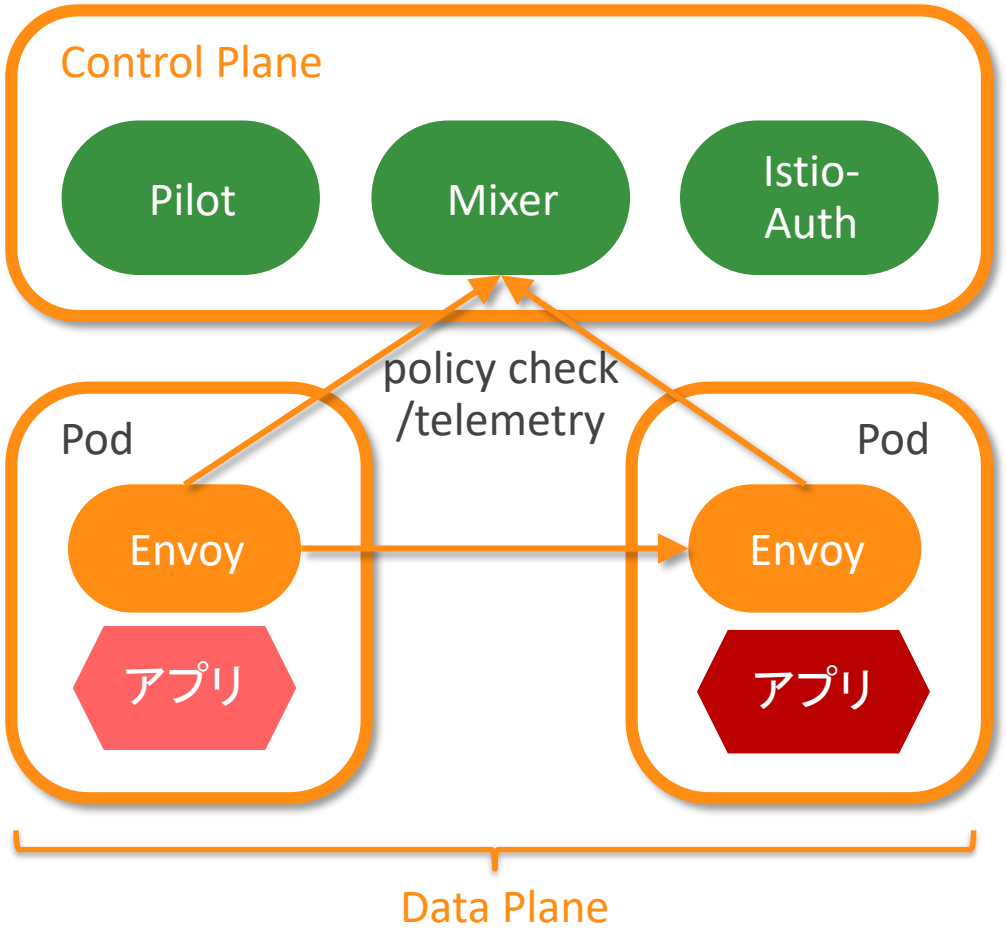
アプリケーションに影響なくサービス境界の問題対策を導入

- Control Plane
  - ポリシーやプロキシ設定の管理を行う
- Data Plane
  - プロキシサーバによりマイクロサービス間の通信を管理する
  - ネットワークトラフィックを仲介し、サーキットブレーカーなどの機能を担う
- 各アプリケーションの手前にプロキシサーバを注入（サイドカーパターン）



Istioを構成するコンポーネント

- Envoy
  - トラフィックを仲介をおこなうプロキシサーバの実態（KubernetesではPodにサイドカーとしてデプロイ）
- Mixer
  - Envoyを通して各サービスのデータを収集し、その情報を元にアクセスコントロールを実施
- Pilot
  - Envoyにサービスディスカバリの機能や各種ルーティングルールを提供
- Istio-Auth
  - サービス同士認証、ユーザー-サービス間の認証サービスのための各種情報（TLS証明書など）を提供



サービス・メッシュ“Istio”が行うことの例

Istioはマイクロサービスを実現するために必要なサービスメッシュを実現

- トラフィック管理
  - ロードバランシング,ルーティング
- サービス間のセキュリティ
  - 認証認可、ポリシー
- 可観測性（Observability）
  - 依存関係やデータフローの可視化
- サービスディスカバリ
  - 追加されたサービスを検出

参考：サーキットブレーカー

障害が起きたサービスを負荷から開放して素早く復旧

- サービス呼び出しに対する応答が一定の条件を満たした場合に、障害発生と判断し、以降の呼び出しを遮断する機構
- 呼び出し元のサービスに早期にエラーが返されるので、呼び出し元でのリソースの専有を回避。適切なエラーにフォールバックできる
- リクエストが遮断されている間に障害を起こしたサービスが回復

(1) 一定数のエラーを観測したら

(2) リクエストを遮断

参考：その他のサービス・メッシュを実現するソフトウェア

- Linkerd
  - Buoyant社により開発
- CONDUIT
  - 2018年7月にバージョン0.5.0がConduitとしての最後のリリースとなり、Linkerd 2.0にマージ
- Hashicorp Consul
  - Hashicorp社により開発、Service Discoveryを実現するソフトウェアとして提供
  - 2018年6月に発表されたバージョン1.2からサービスメッシュを実現する機能（Connect）が追加された

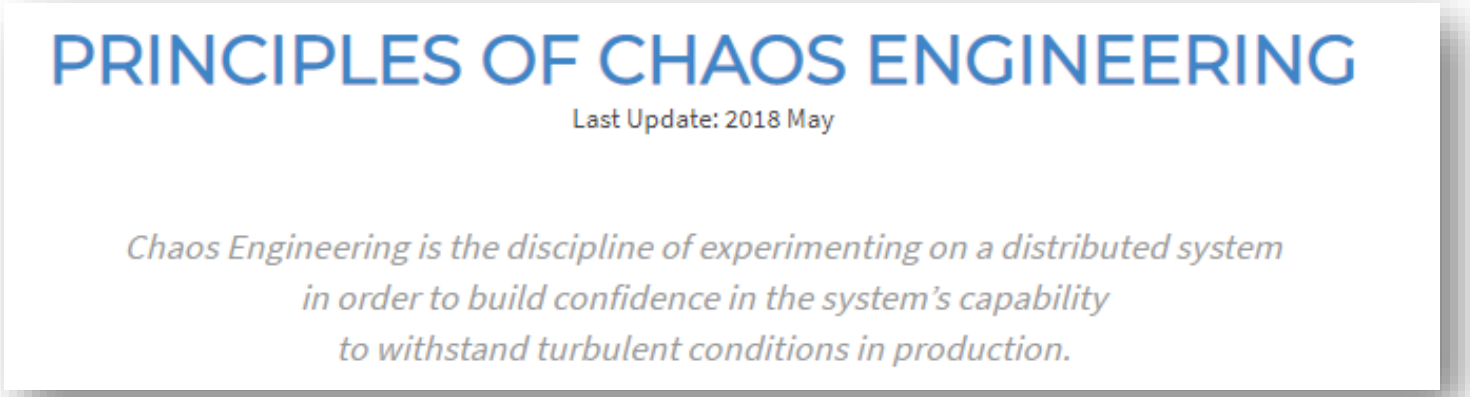
サービス境界の問題解決に向けた取り組み

取り組み	手段	活用できるツール （本日紹介するもの）
事前に把握する	カオスエンジニアリング カナリアリリース	Istio
監視する	ログ・メトリック可視化する 依存関係やデータフローを可視化する	Istio / Prometheus / Grafana / Kiali
分析する	ログ・メトリックを収集する 因果関係を追跡する	Istio / Prometheus / Jaeger / Kiali

影響を事前に把握するには?

“障害・パフォーマンスダウン時”を実際に発生させる

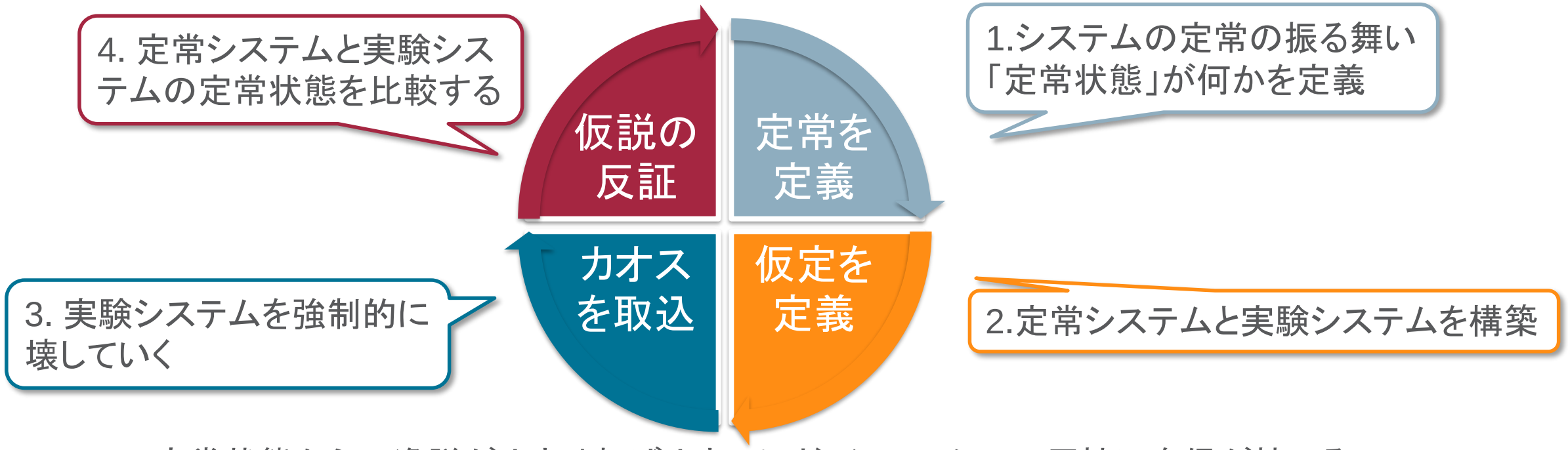
- カオスエンジニアリングとは
  - Netflix社が発表した論文“PRINCIPLE OF CHAOS ENGINEERING”
    - 「プロダクション環境の過酷な状況に耐えられるというシステムの能力に自信を持つため、分散システムで実験するという規律」
    - ⇒ 制御された実験の間にそれを観察することによって、分散システムのふるまいについて学ぶ。これをカオスエンジニアリングと呼びます



http://principlesofchaos.org/

カオスエンジニアリングの原則

カオスエンジニアリングは、分散システムの弱点を発見するための実験



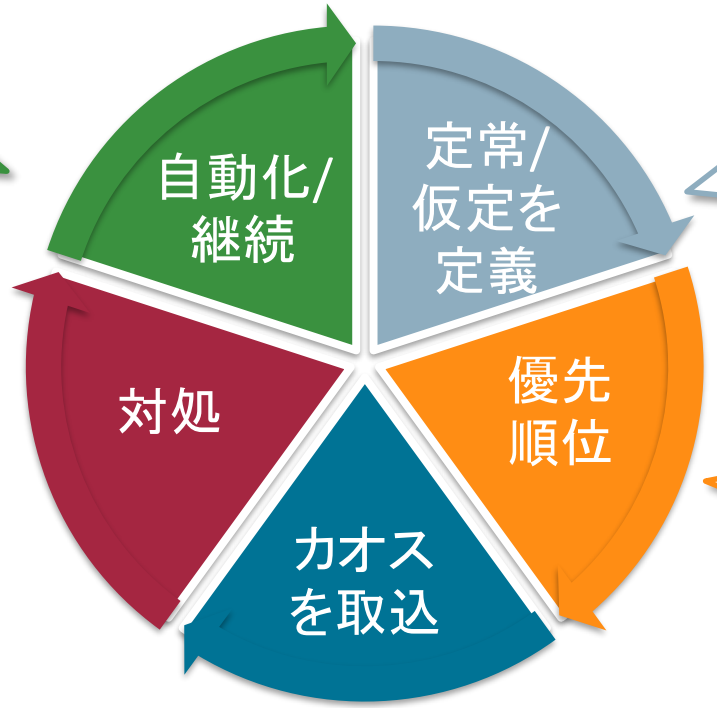
定常状態からの逸脱が小さければ小さいほど、システムの可用性に自信が持てる
もし実験中に問題があれば、起こったことから学んで、適切な措置をとることができる
1～4を繰り返し行う

http://tech.cygames.co.jp/archives/3178/
http://principlesofchaos.org/

カオスエンジニアリングの理想（高度な原則）

本番稼働中のサービスにあえて擬似的な障害を起こすことで、実際の障害にも耐えられるようにする

5.実験を自動化して継続的に実行する



1.定常状態の振る舞いについて仮説を立てる

4. 爆発半径を最小化する

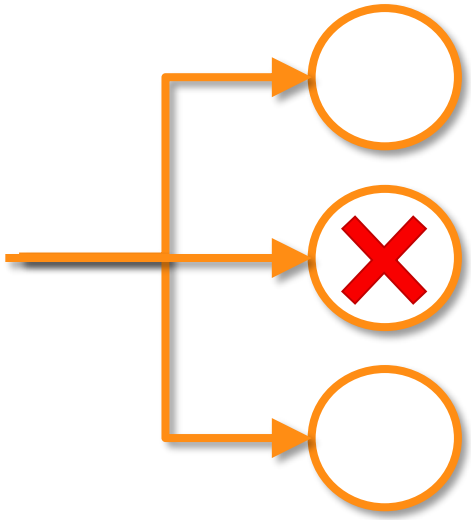
2.影響度や想定される発生頻度を考慮し優先順位をつけて実験をする

3. プロダクション環境で実験を実行する

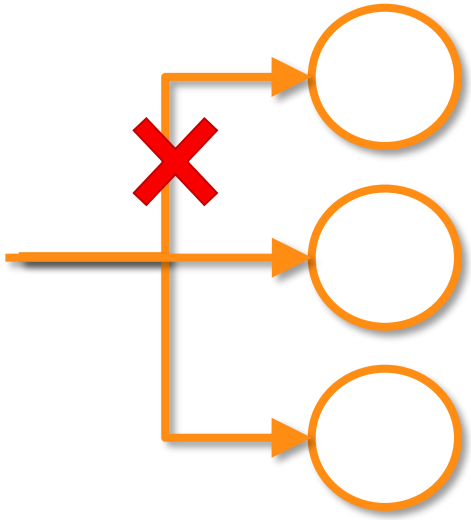
http://principlesofchaos.org/

カオスの例

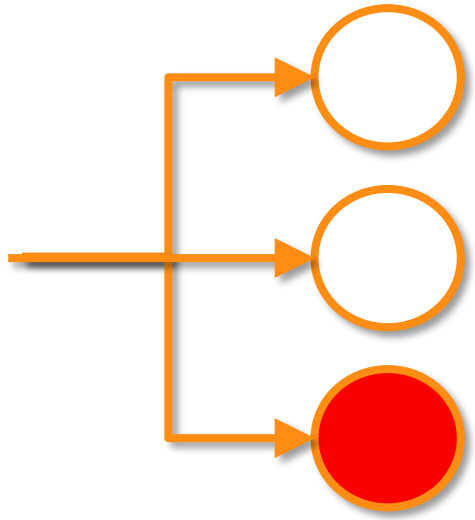
実際に起こりうる障害を取り入れる



インスタンス（Pod）の停止



Networkの遮断、遅延



インスタンス（Pod）の
CPU負荷が100%

http://tech.cygames.co.jp/archives/3178/

カオスエンジニアリングを実現するツール

- Chaos Monkey
  - Netflixが公開している最も有名なカオスエンジニアリングツール
  - クラウドインスタンスやKubernetes上のコンテナを落とすだけでなく、NW、DISK、CPUの負荷を高くしたりと様々な障害を注入できる
    - Spinnaker（継続的デリバリ（CD））ツールの導入が必須
- Pumba
  - Chaos Monkeyのような挙動をDockerのコンテナレベルで行うツール
    - コンテナより下位レイヤーに障害を注入することは出来ないですが、非常に扱いやすく、カオスエンジニアリング対象のスコープが合えば十分に効果のあるものだと思います
- Gremlin
  - Failure as a Serviceと呼ばれる、システムに障害を注入するためのSaaSプラットフォーム
  - システム構成を大きく変更することなく、インフラ、アプリケーションに障害を注入

Istioでカオスを発生させる

- IstioのFault Injection
  - 対象範囲と障害内容を指定し障害を疑似的に起こす(カオスを発生させる)ことが可能
  - 指定された割合のリクエストに対し以下の設定が可能
    - Delay:指定された時間だけ遅延させる
    - Abort:指定されたステータスコードを返す

```
1   spec:
2     hosts:
3     - ratings
4     http:
5     - match:
6       - headers:
7           end-user:
8             exact: jason
9       fault:
10        abort:
11          percent: 100
12          httpStatus: 500
13      route:
14      - destination:
15          host: ratings
16          subset: v1
17    - route:
18      - destination:
19          host: ratings
20          subset: v1
21  ~
```

“リリース”による影響をテストする

リリースと可用性の関係

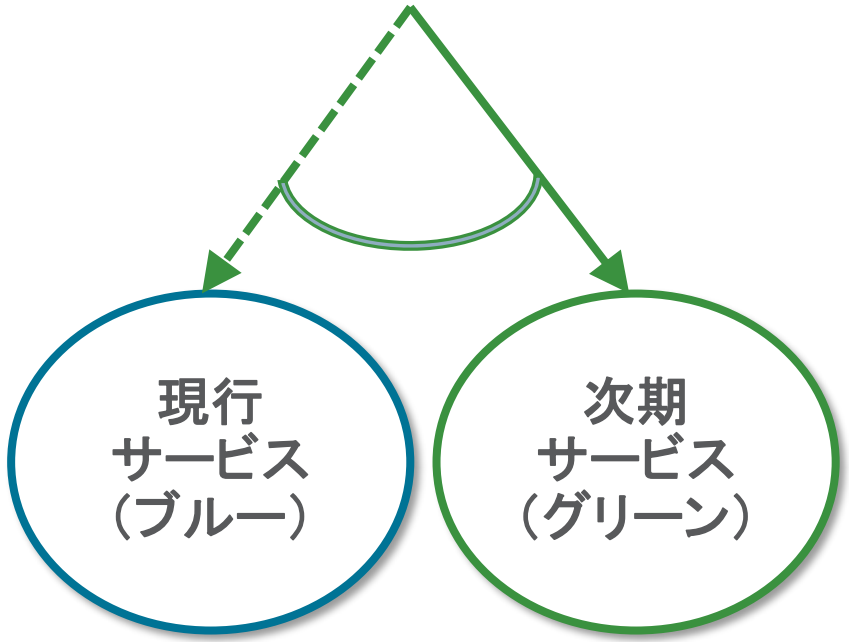
- リリースは本番環境に変更を加えること
  - 変更は障害のリスクとなる
  - マイクロサービスは頻繁にリリースされる

- 安定した開発を行い、適切なステージング環境で十分なテスト
  - バグを含まないようにする
  - 負荷テスト・カオステストを行う
  - 自動化することでテスト・リリース作業でミスをしない

ブルー/グリーンデプロイメントによるカナリア・リリース

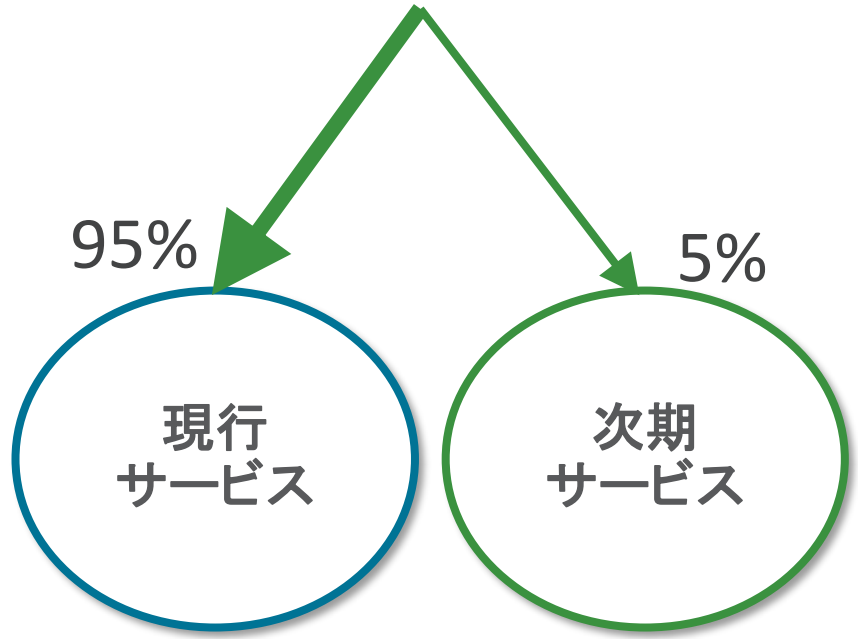
安定・安全なリリースを実現するデプロイ手法

ブルー/グリーンデプロイメント



不具合があった場合には
すぐにブルーに切り替える

カナリア・リリース



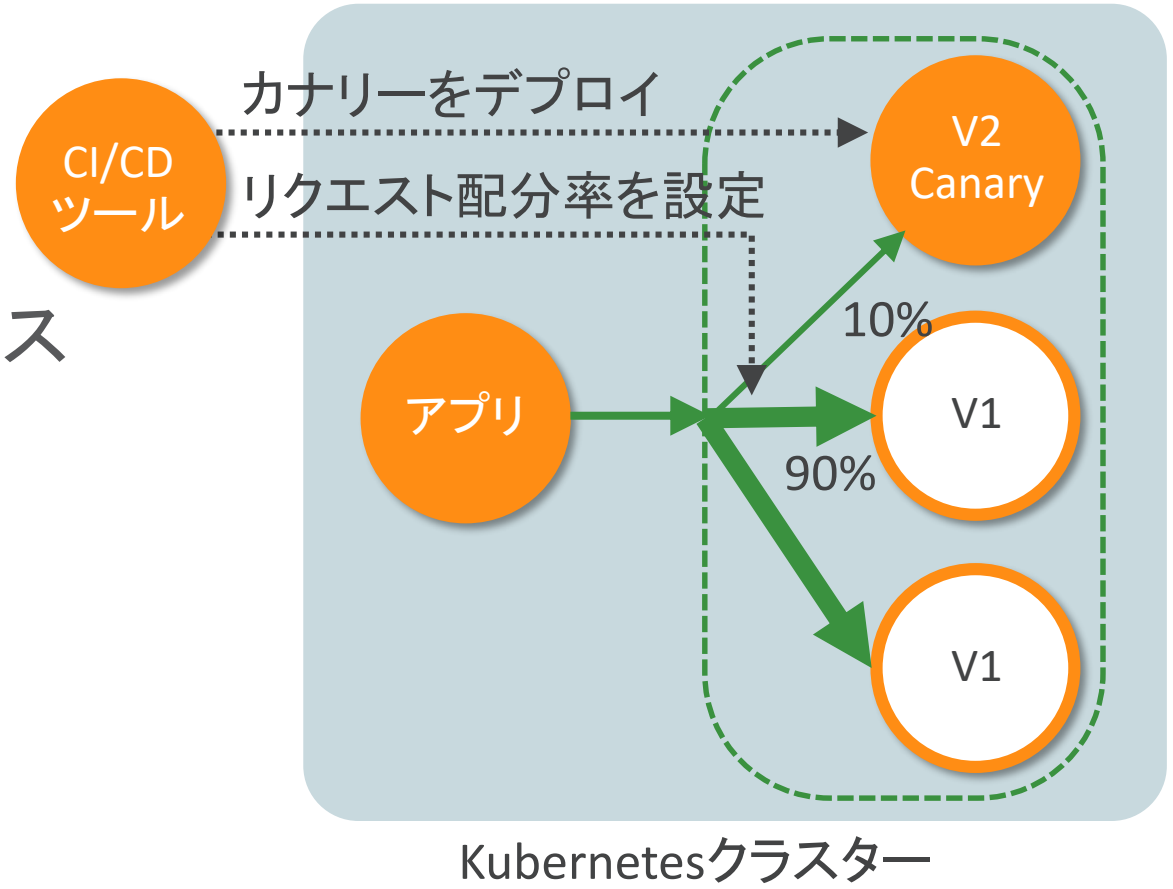
大半のユーザーは
現行バージョンを使用

一部ユーザーに
次期バージョンをリリース

Istioによるカナリア・リリース

カナリア・リリースによって安定・安全なリリースを実現

- Istioにより、リクエストの配分率をきめ細かに設定可能
- CI/CDツールを用いてカナリア・リリースに必要なタスクを自動化可能
  - リクエスト配分率の適用
  - 次期バージョンのデプロイ



Istioによるカナリア・リリース

カナリア・リリースによって安定・安全なリリースを実現

- トラフィックのWeightを指定
- 例の場合v1に90%、v2に10%割り振る設定をしている

```
1   kind: VirtualService
2   metadata:
3     name: helloworld
4   spec:
5     hosts:
6       - helloworld
7     http:
8     - route:
9       - destination:
10          host: helloworld
11          subset: v1
12        weight: 90
13      - destination:
14          host: helloworld
15          subset: v2
16        weight: 10
17
```

https://istio.io/blog/2017/0.1-canary/

サービス境界の問題解決に向けた取り組み

取り組み	手段	活用できるツール （本日紹介するもの）
事前に把握する	カオスエンジニアリング カナリアリリース	Istio
監視する	**ログ・メトリック可視化する 依存関係やデータフローを可視化する**	**Istio / Prometheus / Grafana / Kiali**
分析する	ログ・メトリックを収集する 因果関係を追跡する	Istio / Prometheus / Jaeger / Kiali

マイクロサービスの監視の課題

システム全体の把握が難しい

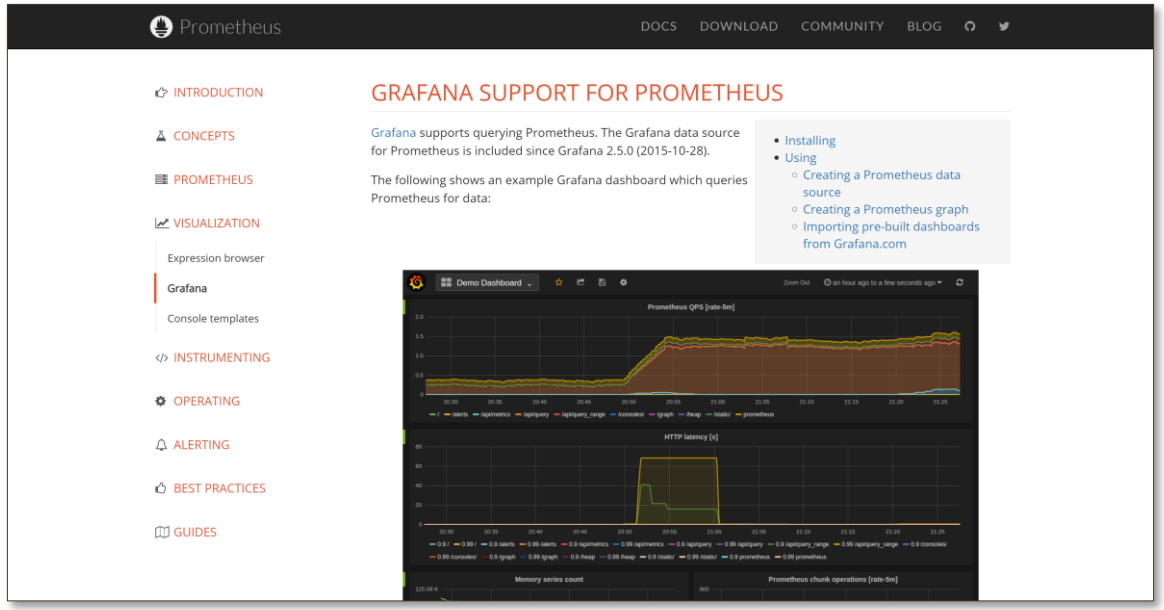
- 新規にサービスが立ち上がった場合に、サービスディスカバリする必要がある
- サービス、アプリケーションの数が大きくなりサービス依存関係やデータフローの管理が困難

ログ・メトリック可視化するツール

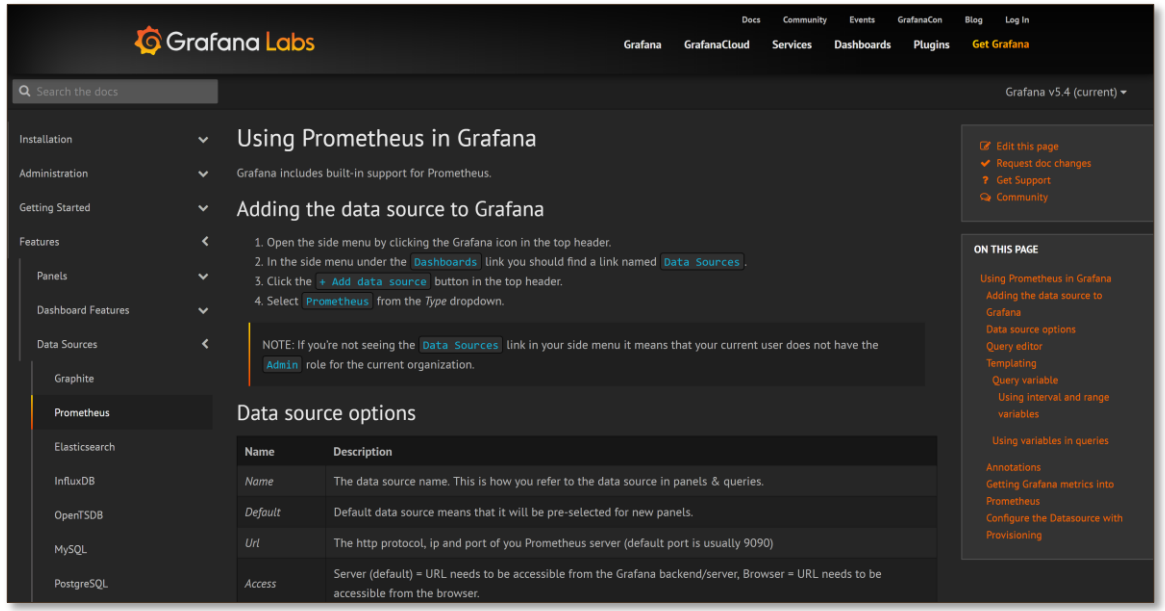
- Prometheus （https://prometheus.io）
  - SoundCloud社によって2012年に開発されたOSSの監視ツール
    - Google の社内監視システム Borgmon の思想をベースに作成
  - 監視対象のサーバーからの情報取得と保管、柔軟なアラート設定に加えて、独自のデータモデルを活用した時系列クエリがお得意

- Grafana （https://grafana.com）
  - Grafana Labs（旧raintank社）が OSS として開発しているメトリック分析・可視化ツール
  - 150,000 アクティブインストール数を誇る、オシャレ監視ダッシュボードコーデの定番
  - 50種類以上のOSS・商用データソースに対応

ドキュメント上でも相思相愛のご様子



Prometheusの公式サイトより



Grafanaの公式サイトより

Prometheusの特徴

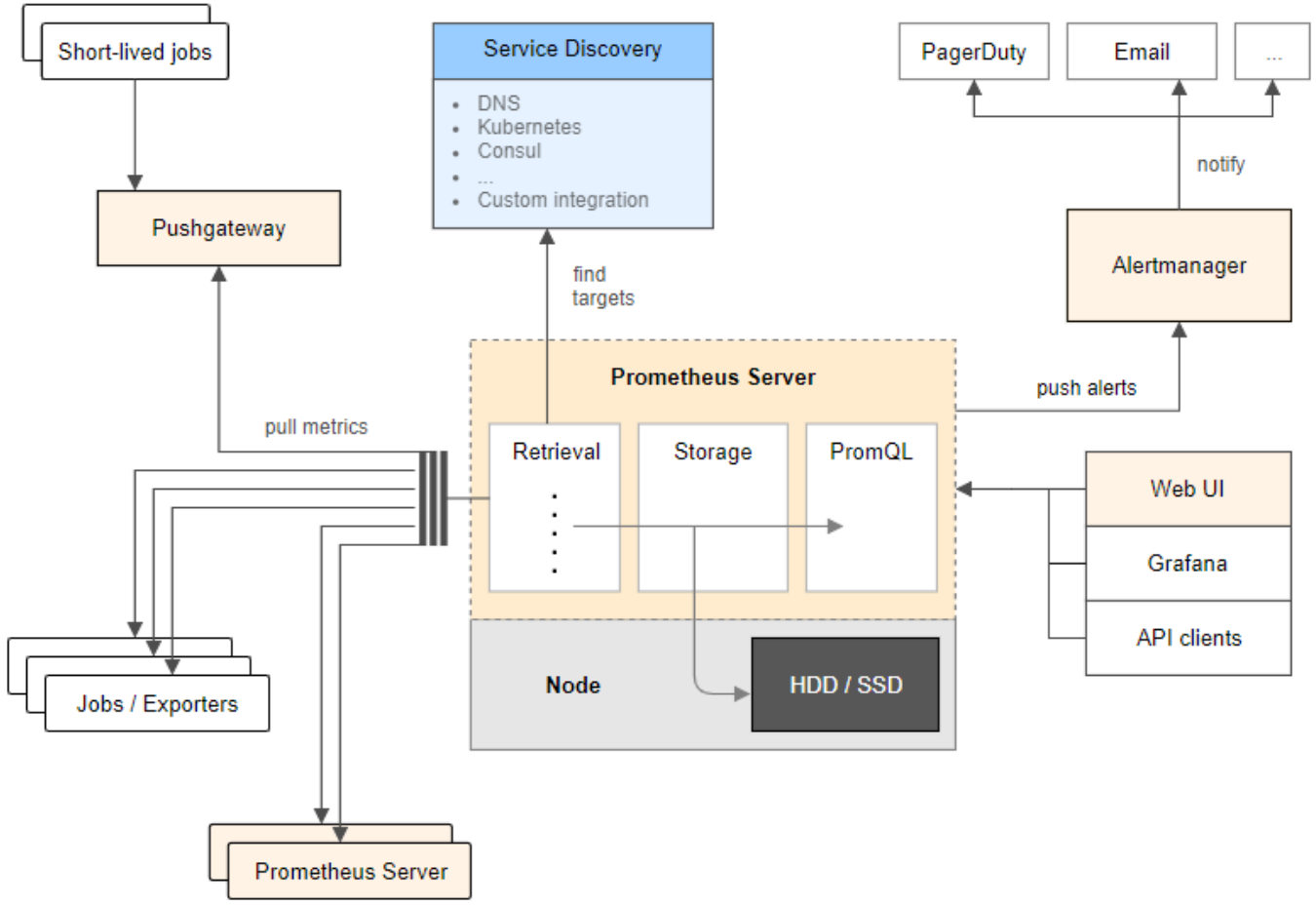
時系列データの記録に特化したPull型監視ツール

- Pull型のメトリクス監視　+ サービスディスカバリ
  - Pull 型は監視対象リストをアップデートする仕組みが必要
  - サービスディスカバリの設定を利用することにより、Podを追加した際にも自動的にそれを検知しメトリクスを取得できるようになる

Prometheus サーバが読み取り可能なデータを生成

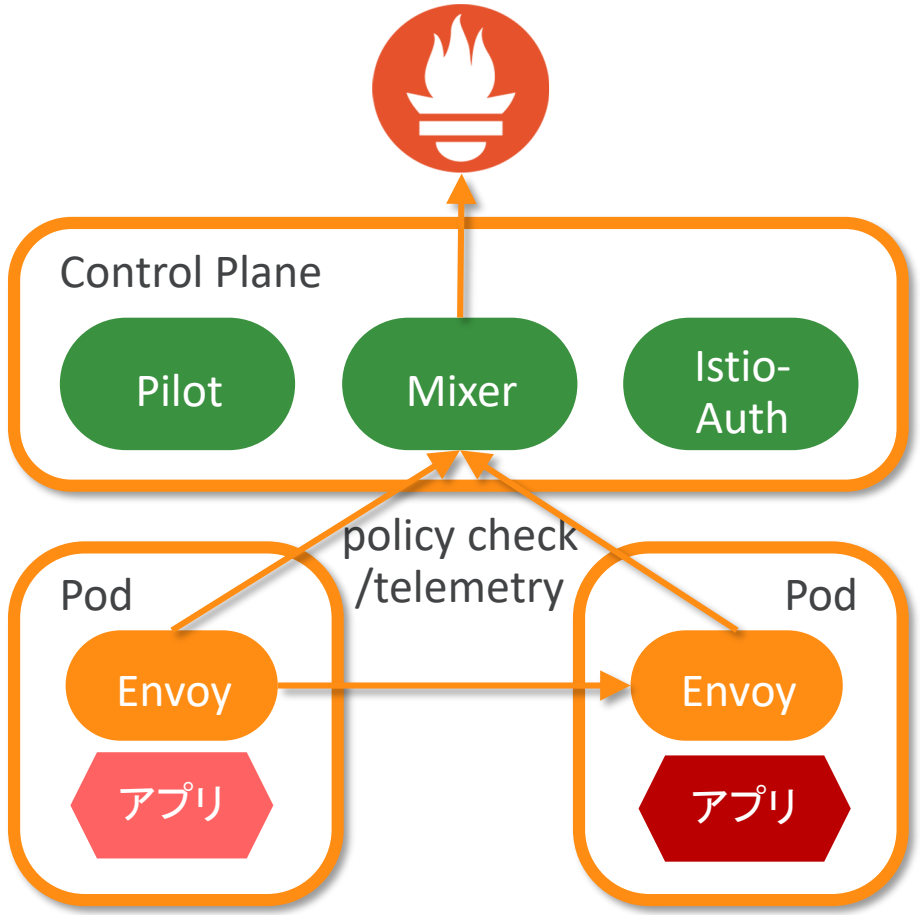
Prometheusの監視例

- パフォーマンスモニタリング
  - サービス（Pod）のリソース監視
    - CPU, RAM, Netowrk ,I/O使用量
  - アプリケーションの監視
    - リクエスト数、レスポンスタイム
      対象ソフト毎の固有の情報
  - ログの監視
    - error等の文字列の有無
- アラート通知
  - 異常検知、障害検知

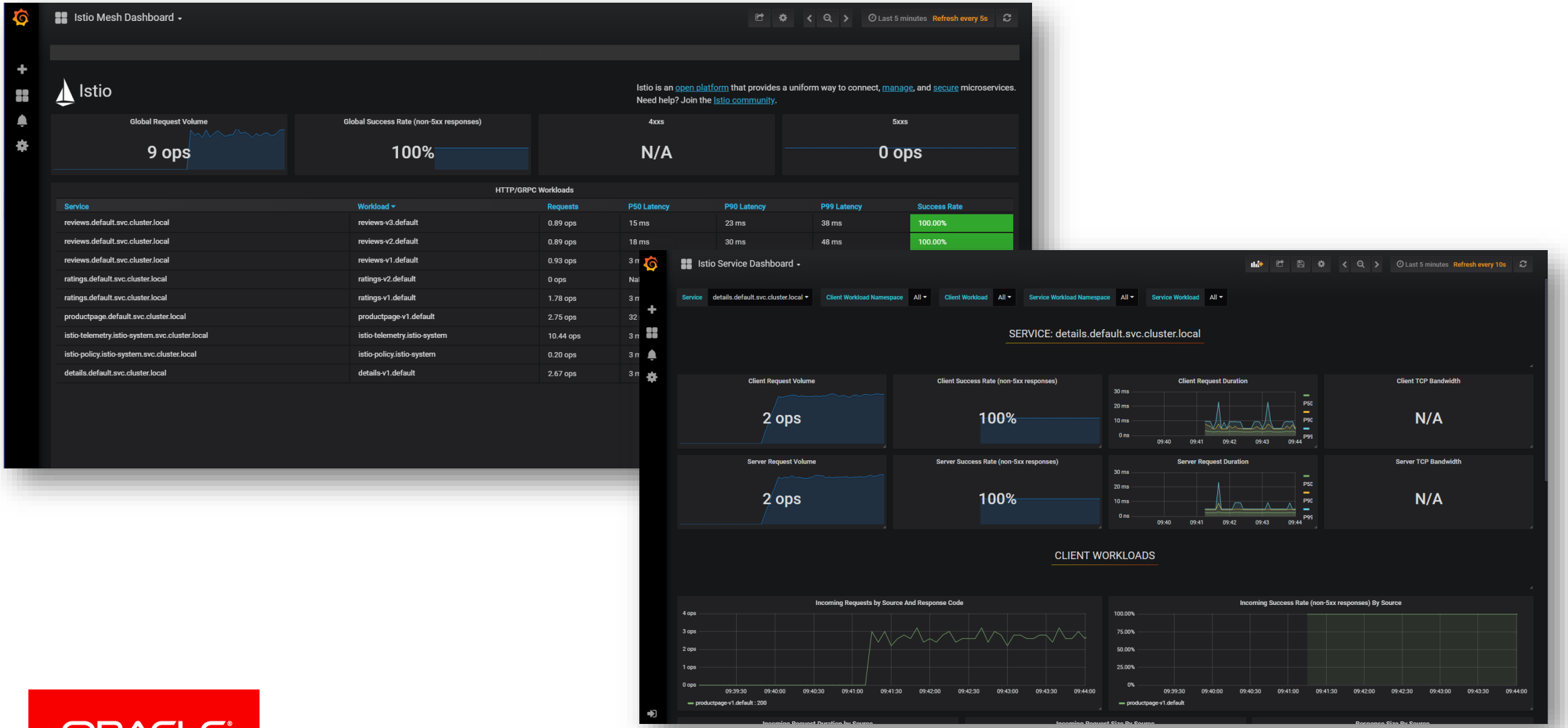


IstioとPrometheusの連携

- PrometheusがMixerのエンドポイントからメトリックを取得
- Mixerにはメトリック値をPrometheusを提供するエンドポイントを公開するアダプタがビルドイン
  - istio-mesh（istio-mixer.istio-system:42422）:
    ミキサーが生成したすべてのメッシュメトリック
  - mixer（istio-mixer.istio-system:9093）:
    すべてのミキサー固有のメトリック(ミキサー自体をモニターするために使用)
  - envoy（istio-mixer.istio-system:9102）:
    Envoyによって生成されたメトリック

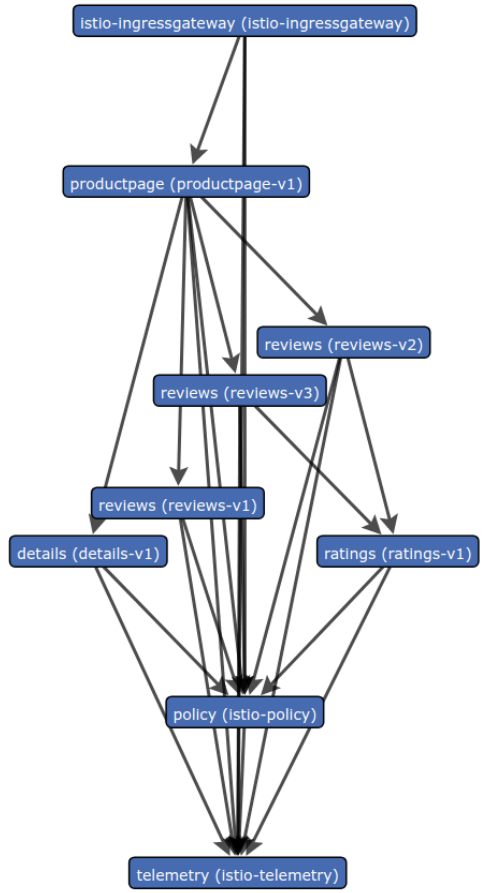


Istio専用のGrafanaダッシュボード



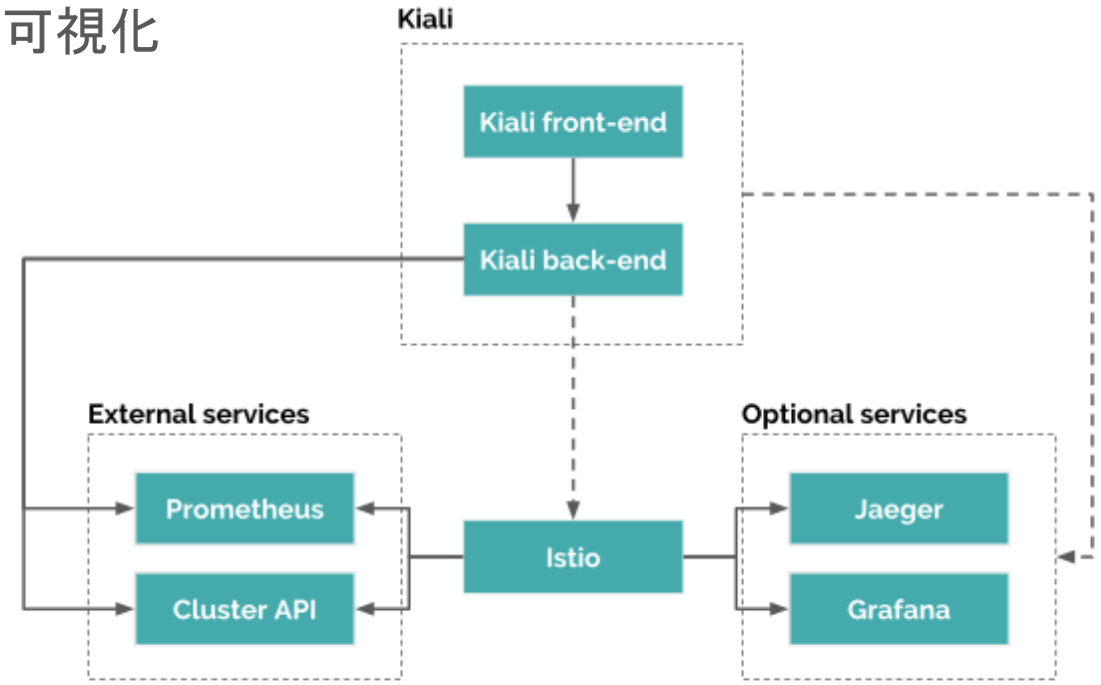
Istioで依存関係やデータフローを可視化する

- 取得したネットワークメトリックから自動的にServiceGraphを生成する
  - Servicegraphアドオンをインストールし、ブラウザからサービス・メッシュのサービスグラフを表示可能



Istioで依存関係やデータフローを可視化する

- Kiali
  - Istioと連携してサービスメッシュトポロジを可視化
    - サーキットブレーカー、要求レートなどの機能を可視化
    - 分散トレーシング
    - ヘルス表示・計算
    - メトリクス収集、グラフ
- 詳しくはDemoで！

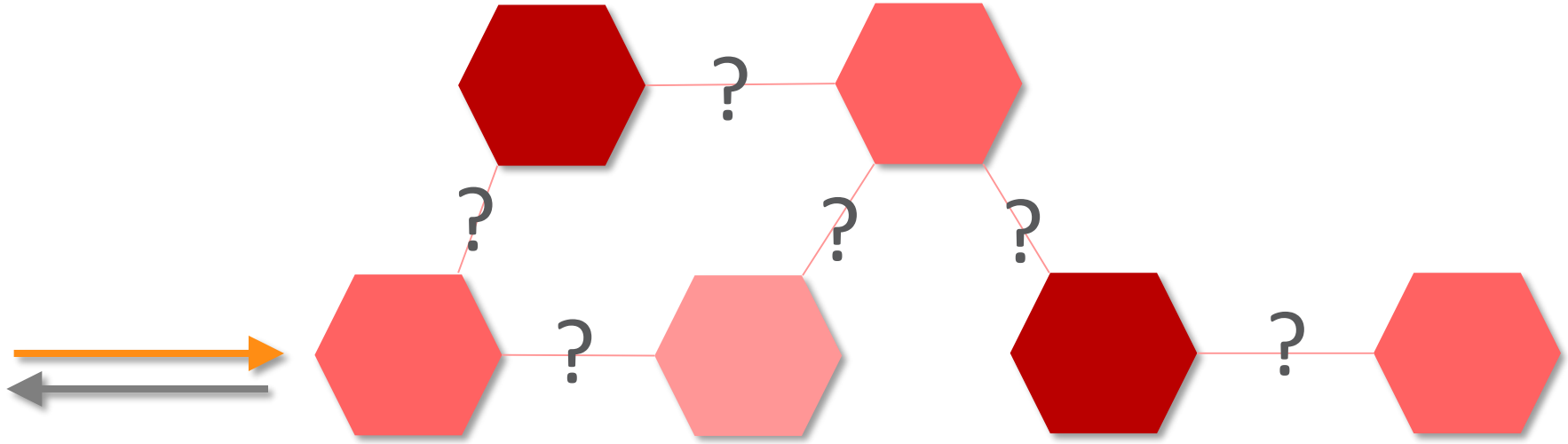


サービス境界の問題解決に向けた取り組み

取り組み	手段	活用できるツール （本日紹介するもの）
事前に把握する	カオスエンジニアリング カナリアリリース	Istio
監視する	依存関係やデータフローを可視化する ログ・メトリック可視化する	Istio / Prometheus / Grafana / Kiali
分析する	**ログ・メトリックを収集する 因果関係を追跡する**	**Istio / Prometheus / Jaeger / Kiali**

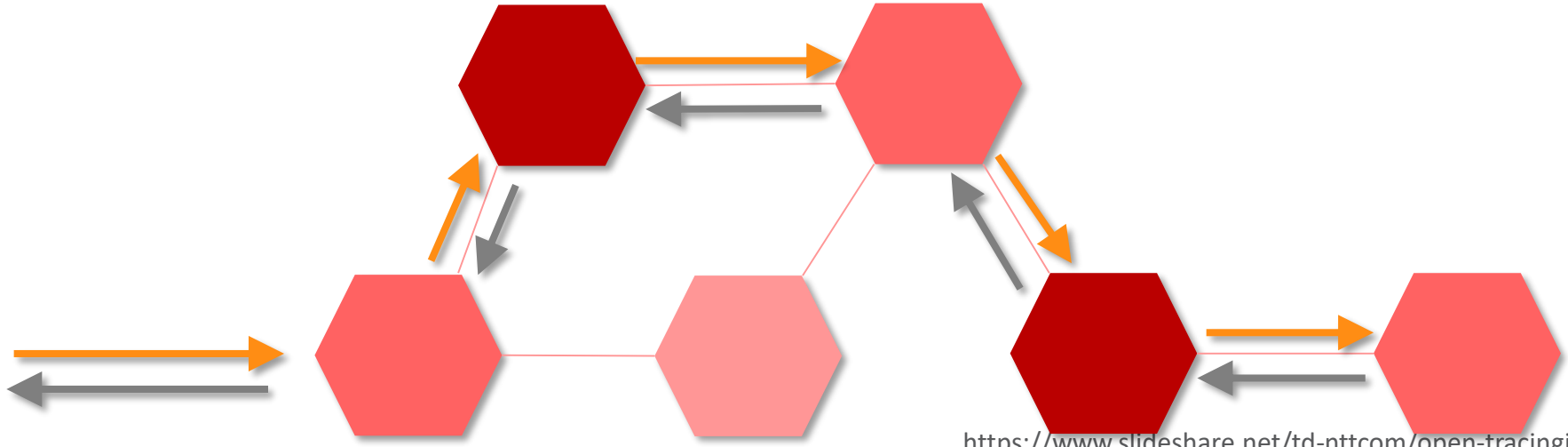
分析の課題

- マイクロサービスのような分散アーキテクチャでは、1つのリクエストに対して、複数のサービスをまたがった処理が行われる
- ネットワークがボトルネックとなるケースが多いが、サービスが増えることでリソースのボトルネック箇所が把握しづらい



分散トレーシング

- 分散システムにおけるアプリケーション間のトランザクションを可視化、モニタリングするための仕組み
  - トレース情報の可視化を行うことによって、特定のリクエストに関連するサービスの特定やデバッグなどに利用



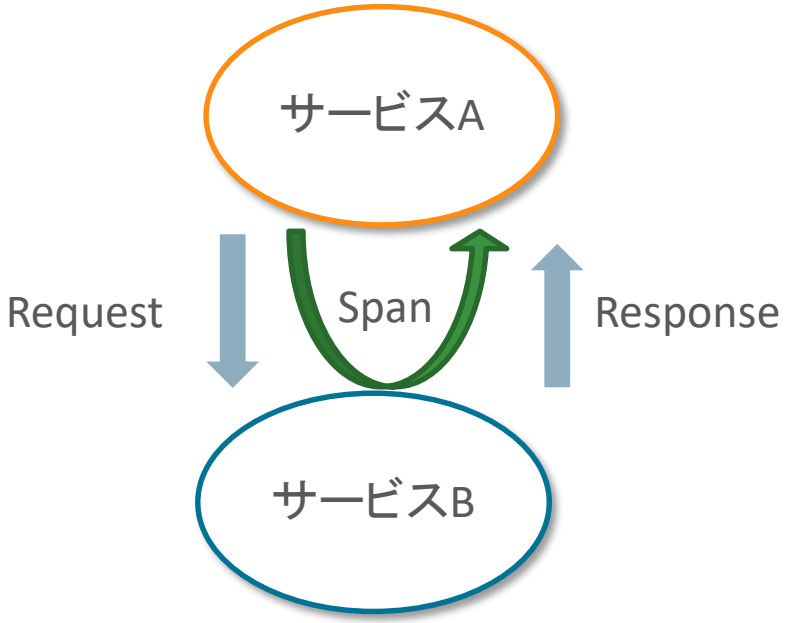
https://www.slideshare.net/td-nttcom/open-tracingjaeger

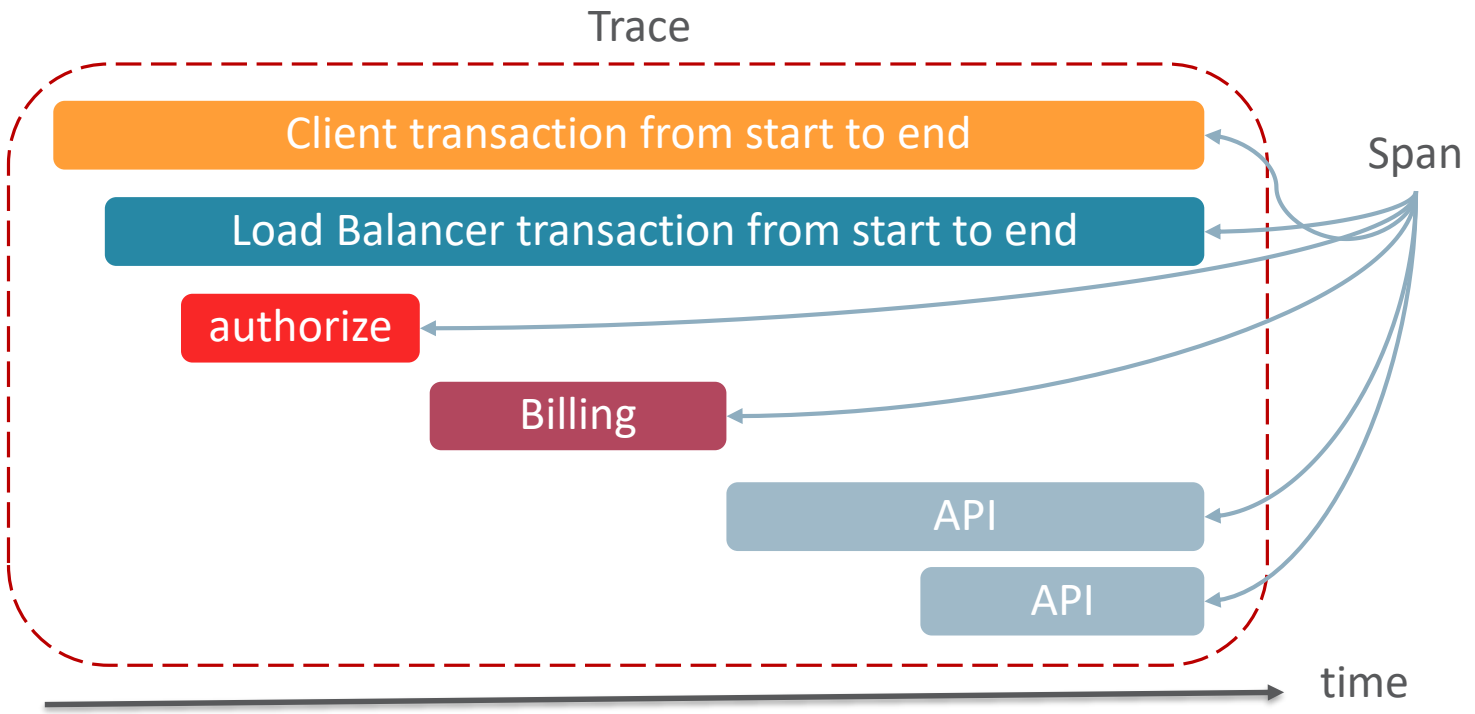
OpenTracing

- 分散トレーシングシステムの標準化を目的とした実装方法を提供
  - ベンダ非依存のAPI仕様とライブラリを提供
  - 開発者はアプリケーションコードにトレーシング機能を追加可能
- OpenTracing プロジェクトは 2015 年に始まり、2016年10月からCloud Native Computing Foundation(CNCF)がホストするプロジェクト

OpenTracing データモデル

- スパン（Span）：一つのサービス処理
- トレース（Trace）：リクエストの開始から終了までを含むSpanの集合体





https://opentracing.io/docs/overview/

代表的な分散トレーシングシステムの実装（例）

- Zipkin (https://zipkin.io)
  - TwitterがGoogle Dapperを参考に開発した分散トレーシングシステム
  - 各サービス間のAPIコールのデータを収集する機能とそのデータを可視化するためのUIを提供

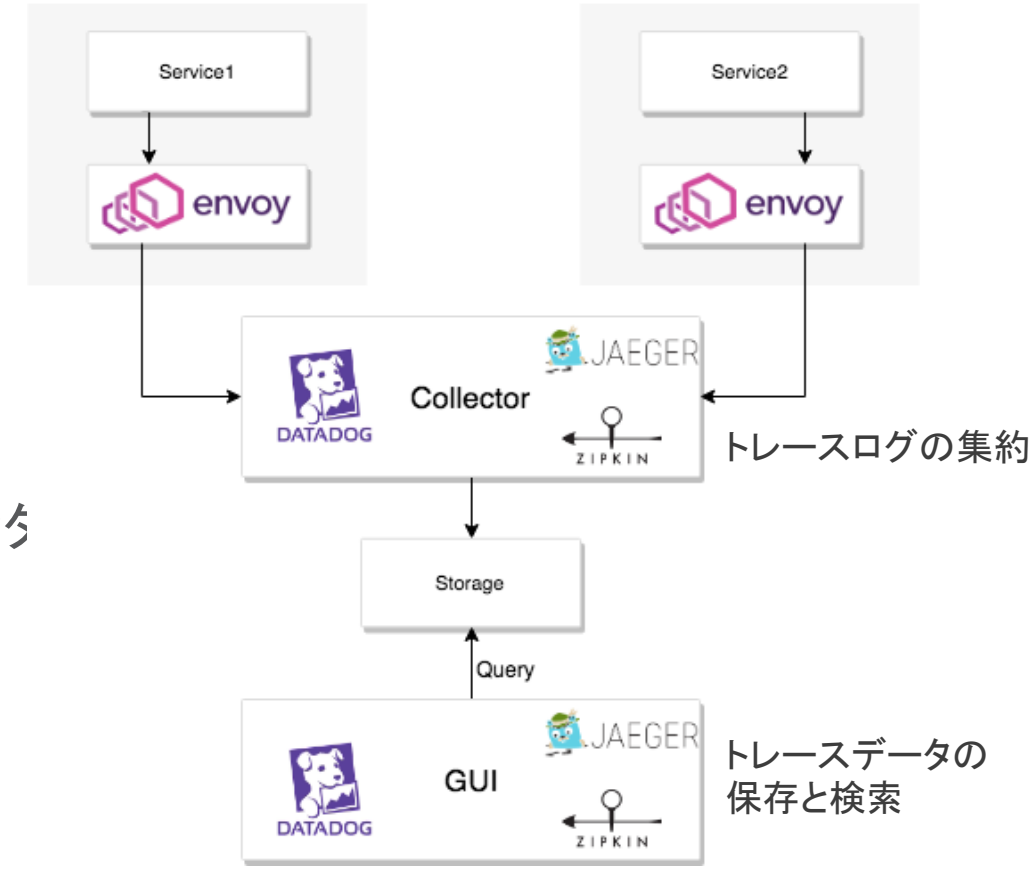
- Jaeger (https://www.jaegertracing.io)
  - Uber TechnologiesがGoogle DapperやOpenZipkinを参考に開発した分散トレーシングシステム
  - 分散型のコンテキスト伝播やトランザクションモニタリング機能、根本原因解析、サービス依存性の分析が可能
    - 代表的な本番環境での利用は、Uber, Symantec, Red Hat等

https://www.slideshare.net/td-nttcom/open-tracingjaeger

Istioと分散トレーシングシステム

Envoyがメッシュ内のサービス間の通信に関するトレース情報を提供

- Envoyが行うこと
  - トレースの生成
    - tracingオブジェクトを設定することで、トレースを生成
    - リクエスト毎に一意の識別子 (x-request-id)を生成
  - トレースコンテキストの伝搬
    - トレース情報を相互に関連Envoyがリクエストのヘッダ
    - 付けるため、トレースコンテキストを伝播
  - コレクターへの送信
    - 自動的にスパンをトレースコレクタに送信



https://www.envoyproxy.io/docs/envoy/latest/intro/arch_overview/tracing#how-to-initiate-a-trace
https://www.jaegertracing.io/docs/1.8/architecture/

Istioと分散トレーシングシステム

- Envoyが自動的にスパンを送信
  - 識別子、コンテキストを伝播しトレーサーはサービス間のトレース情報を相互に関連付けSpan間の親子関係を理解
  - スパン情報を送信したときに、スパンが単一のトレースに正しく関連付けられるようにアプリケーションに適切なヘッダー情報を記述する必要がある

```
1   def getForwardHeaders(request):
2       headers = {}
3
4       if 'user' in session:
5           headers['end-user'] = session['user']
6
7       incoming_headers = [ 'x-request-id',
8                            'x-b3-traceid',
9                            'x-b3-spanid',
10                           'x-b3-parentspanid',
11                           'x-b3-sampled',
12                           'x-b3-flags',
13                           'x-ot-span-context'
14      ]
15
16      for ihdr in incoming_headers:
17          val = request.headers.get(ihdr)
18          if val is not None:
19              headers[ihdr] = val
20              #print "incoming: "+ihdr+":"+val
21
22      return headers
```

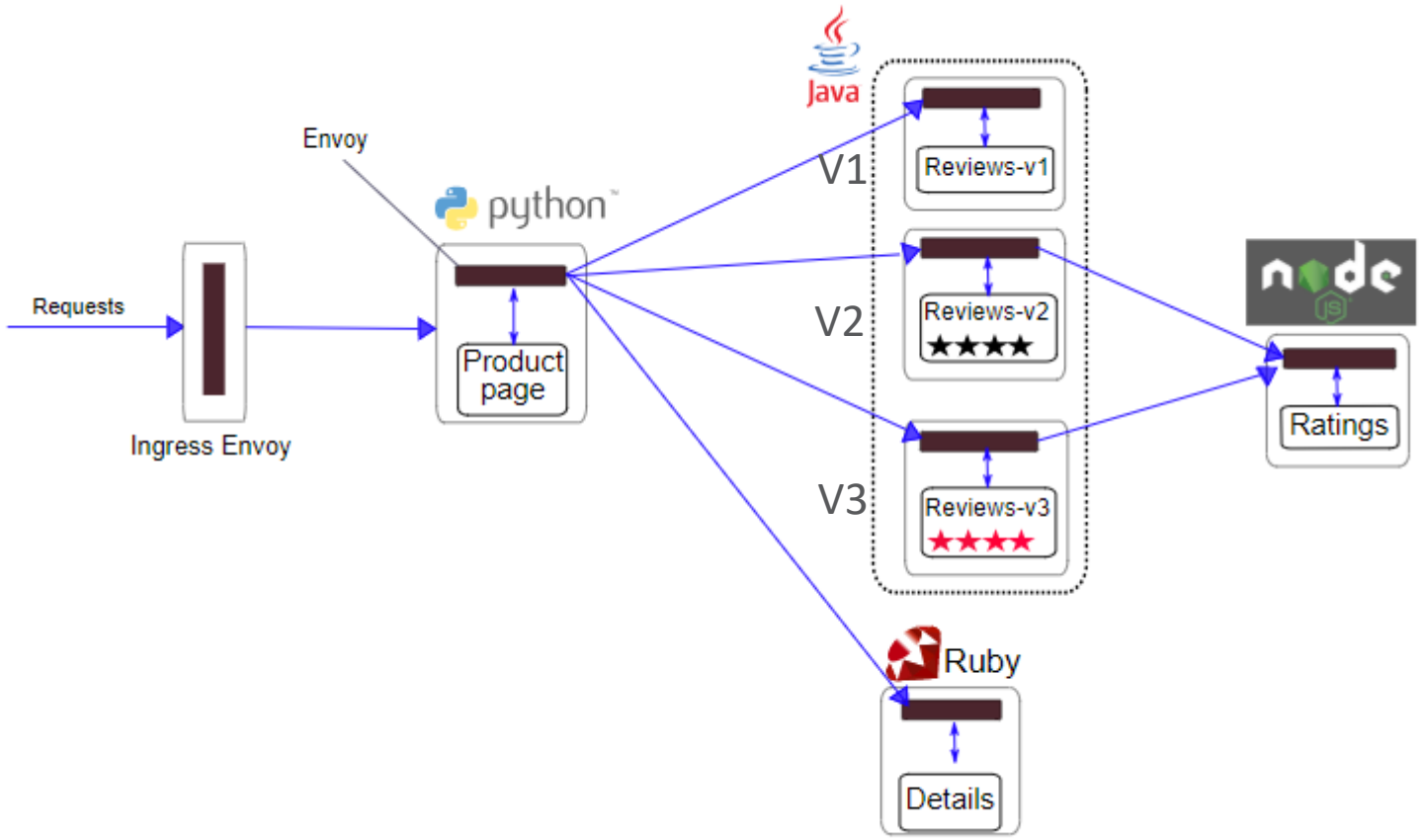
サービス境界の問題解決に向けた取り組み

	取り組み	活用できるツール （本日紹介するもの）
事前に把握する	カオスエンジニアリング カナリアリリース	Istio
監視する	ログ・メトリック可視化する 依存関係やデータフローを可視化する	Istio / Prometheus / Grafana / Kiali
分析する	ログ・メトリックを収集する 因果関係を追跡する	Istio / Prometheus / Jaeger / Kiali

Demo

カオスを起こして、監視・分析してみる

サンプル・アプリケーション：Bookinfo



Bookinfo Application

•Productページ (Python):
取得した本の情報を提供
（フロントの仕組み）
•Reviews (Java):
本のレビュー情報を提供
1~3までのバージョンが存在
•Details (NodeJS):
本の詳細情報を提供する
Productページから呼び出される
•Ratings (Ruby):
本のレートを提供する
レビューのv2、v3から呼び出される

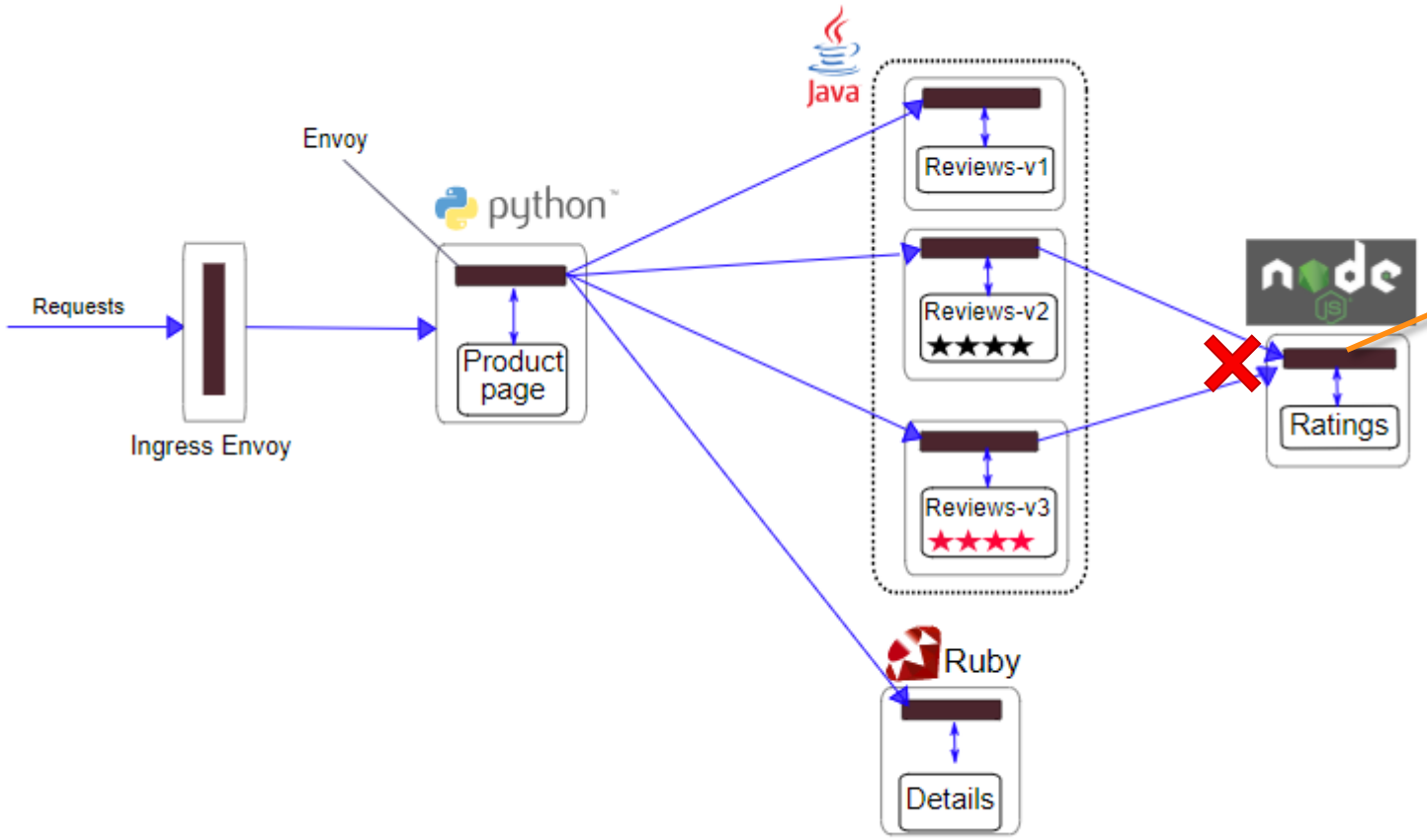
※ Oracle Container Engine for Kubernetes
（OKE）で動いています

やったこと：Istioでカオスを発生させる

- IstioのFault Injection
  - 対象範囲と障害内容を指定し障害を疑似的に起こす（カオスを発生させる）ことが可能
  - 指定された割合のリクエストに対し以下の設定が可能
    - Delay：指定された時間だけ遅延させる
    - Abort：指定されたステータスコードを返す

```
1   apiVersion: networking.istio.io/v1alpha3
2   kind: VirtualService
3   metadata:
4     name: ratings
5     ...
6   spec:
7     hosts:
8     - ratings
9     http:
10    - fault:
11        delay:
12          fixedDelay: 7s
13          percent: 100
14      match:
15      - headers:
16          end-user:
17            exact: jason
```

何が起きていたか



Bookinfo Application

IstioのFault Injectionで
“Jason”ユーザーの処置を
わざと遅延させる

タイムアウトを設定
エラーを返す

※ Oracle Container Engine for Kubernetes
（OKE）で動いています

DemoはContainer Engine for Kubernetesで動いてます

マネージドKubernetesとしての基本をしっかりカバー

- カスタマイズなしのピュアなKubernetes実装
- Certified Kubernetes Program適合[^1]
- GUI / コマンドラインツールによるクラスターのプロビジョニング、アップグレード
- OCIのストレージ、ロードバランサーとの統合[^2]

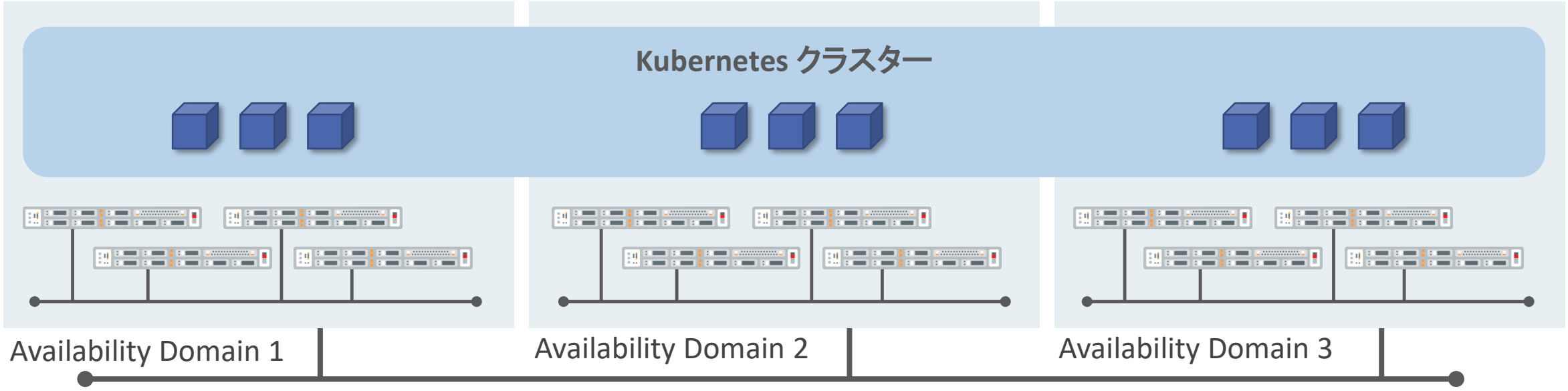
[^1]: Kubernetesクラスターの動作の一貫性とアプリの可搬性を保証する目的で、CNCFが策定した認定プログラム
[^2]: StorageClass と ServiceのLoadBalancerタイプが使用可能

ORACLE

Oracleが提供するKubernetesのアーキテクチャ

OracleのIaaSの能力を生かすKubernetesクラスター構成

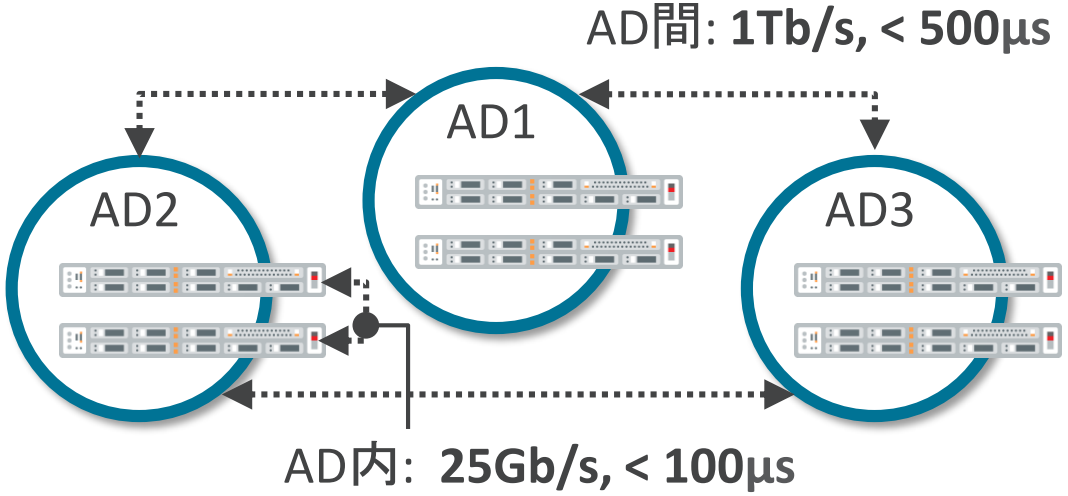
- ハイパフォーマンスな次世代IaaS上にKubernetesクラスターを構築
- Availability Domainを横断してクラスターを構成し、高可用性を実現



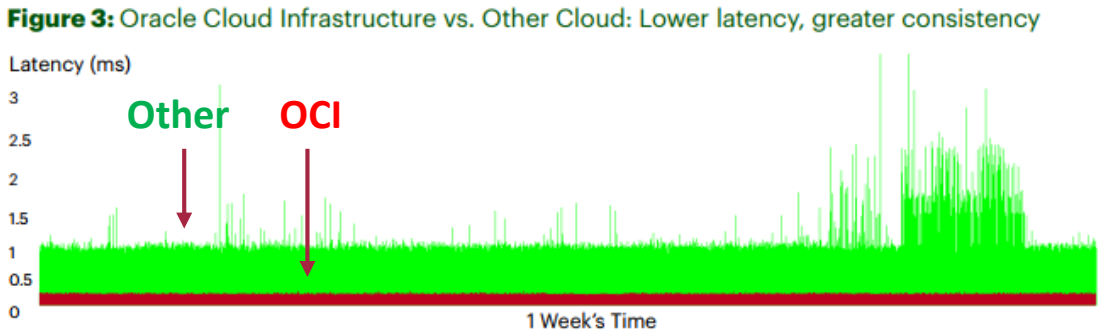
エンタープライズグレードのパフォーマンス

高性能次世代IaaSがもたらすハイパフォーマンス

- ミッションクリティカルなクラスターを運用するのに最適な基盤
  - 高速SSDを搭載したベアメタル（非仮想）サーバー
  - 3つのAvailability Domains (AD) でリージョンを構成。AD間、AD内のホスト間を、低レイテンシー、広帯域のN/Wで接続



第3者機関によるN/W性能検証結果



https://www.accenture.com/t20171003T083750Z__w__/us-en/_acnmedia/PDF-62/Accenture-Enterprise-Workloads-Meet-Cloud.pdf

【参考】パフォーマンスに適用されるSLA

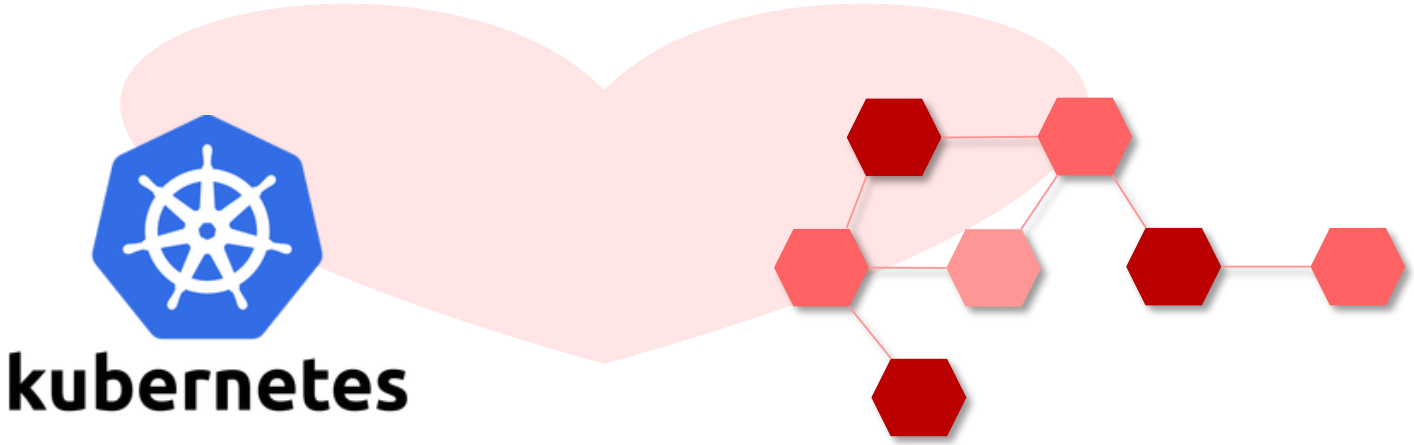
パフォーマンスを含む広範なSLAを、他社に先駆けて提供

Availability, Management, and Performance SLAs

- **Includes** FastConnect availability SLA
- **Includes** compute, storage, database management SLAs
- **Includes** local storage, network block storage, and network performance SLAs

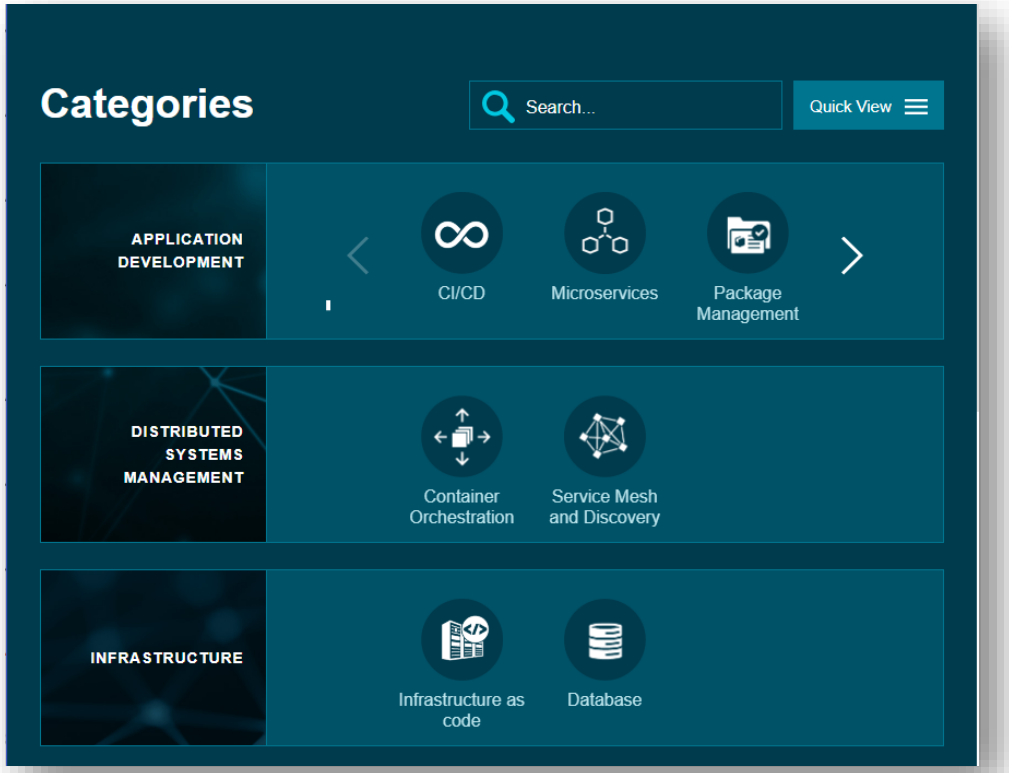
マイクロサービスとコンテナ/Kubernetesの関係

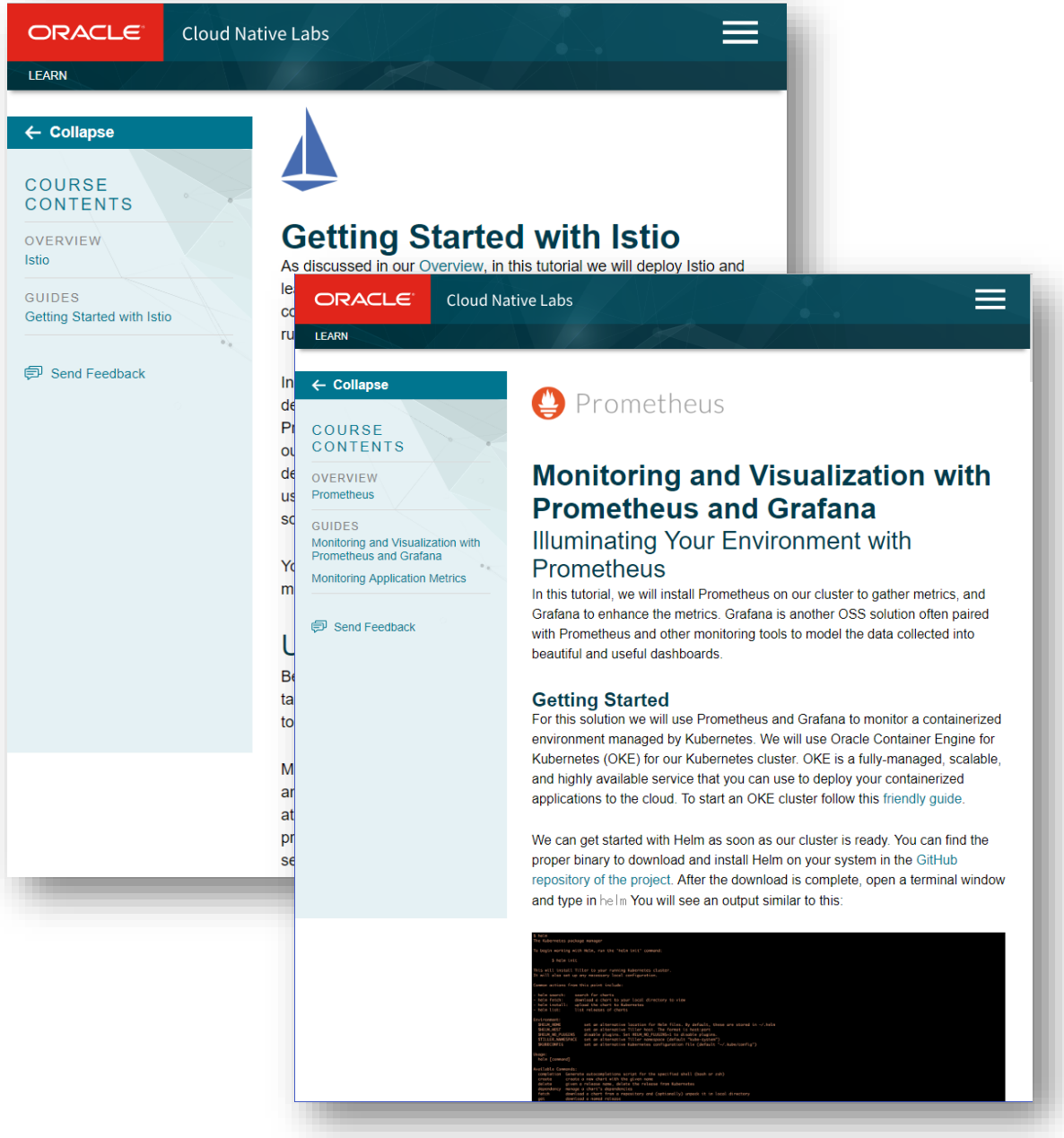
- マイクロサービスはコンテナが持つ独立性と高集約性と相性が良い
  - コンテナが1つのマイクロサービスとして動作することで
    新たなコンテナをたてることでスケールアウトすることができる
  - Kubernetesで複数コンテナのデプロイ、スケーリングを自動管理することが可能



Oracle Cloud Native Labs

https://cloudnative.oracle.com/learn.html





Oracle Cloud
最大 3,500 時間の
無料トライアル

oracle.co.jp/tryit_dev

最新のデータベース、コンピュート、コンテナ、IoT、ビッグ・データ、API管理、統合、チャットボット、その他の各種クラウド・サービスを、無料で体験してみませんか？
無料トライアルのお申込方法の詳細は、QRコード、またはURLにアクセスしてください。


ORACLE®