大規模EKSクラスタにおける
信頼性・セキュリティ向上の取り組み

株式会社ZOZO

技術本部 SRE部 ECプラットフォーム基盤SREブロック

亀井 宏幸・織田 英吾

アジェンダ

- ZOZOTOWNリプレイス
- EKS使用の背景と運用状況
- 大規模EKSクラスタ運用における信頼性・セキュリティ向上の取り組み
  - Design for Failure
  - 耐障害性を高める
  - トラフィック増加に備える
  - Security Groups For Podsによるマルチテナントのアクセス管理

株式会社ZOZO

技術本部 SRE部 ECプラットフォーム基盤SREブロック

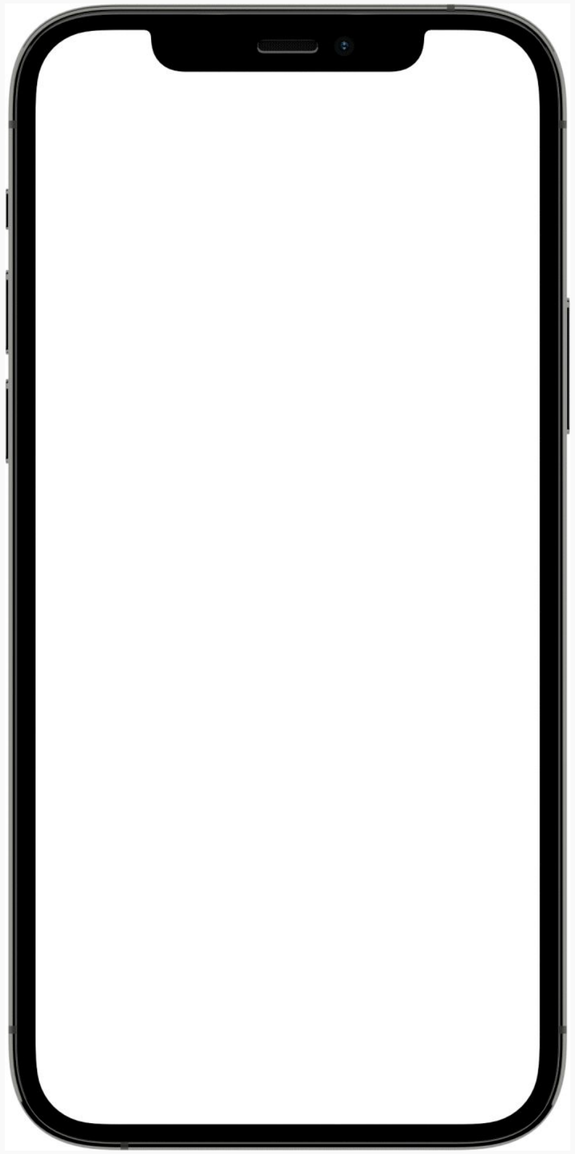
亀井 宏幸

2017年に入社し、オンプレサーバーの運用を経て

2019年にZOZOTOWNリプレイスPJにSREとして参画

現在はZOZOTOWN Microservices PlatformのSREチーム

で、基盤やマイクロサービスの運用を行う



ZOZOTOWN

https://zozo.jp/

- ファッションEC
- 1,500以上のショップ、8,400以上のブランドの取り扱い
- 常時90万点以上の商品アイテム数と毎日平均2,600点以上の新着 商品を掲載（2022年3月末時点）
- ブランド古着のファッションゾーン「ZOZOUSED」やコスメ専門モール「ZOZOCOSME」、靴の専門モール「ZOZOSHOES」、ラグジュアリー＆デザイナーズゾーン「ZOZOVILLA」を展開
- 即日配送サービス
- ギフトラッピングサービス
- ツケ払い など

Download on the App Store　GET IT ON Google Play

アジェンダ

- **ZOZOTOWNリプレイス**
- EKS使用の背景と運用状況
- 大規模EKSクラスタ運用における信頼性・セキュリティ向上の取り組み
  - Design for Failure
  - 耐障害性を高める
  - トラフィック増加に備える
  - Security Groups For Podsによるマルチテナントのアクセス管理

アーキテクチャを変えずに規模を拡大してきた

OPEN　2004年

2005年

2007年

2019年

アーキテクチャを変えずに規模を拡大してきた

OPEN

2004年

2005年

2007年

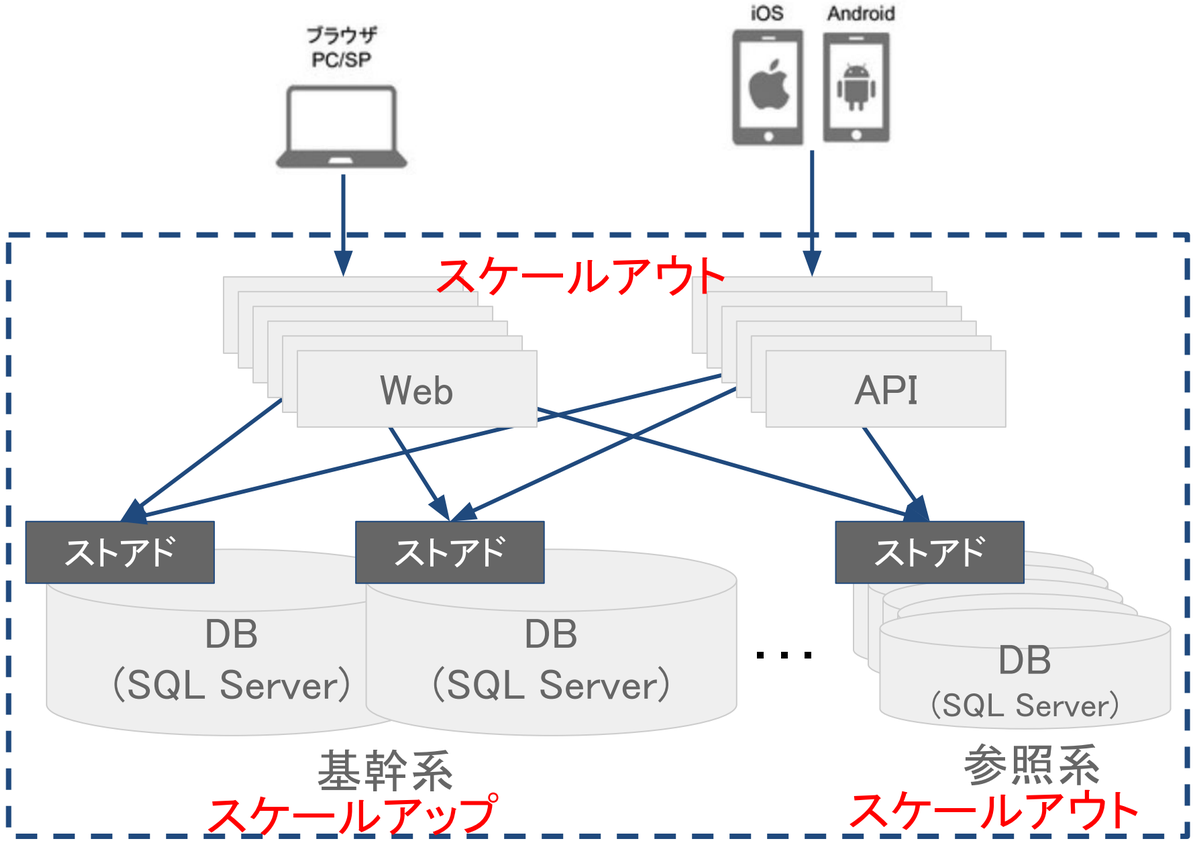
2019年

オンプレミス

Microsoft IIS

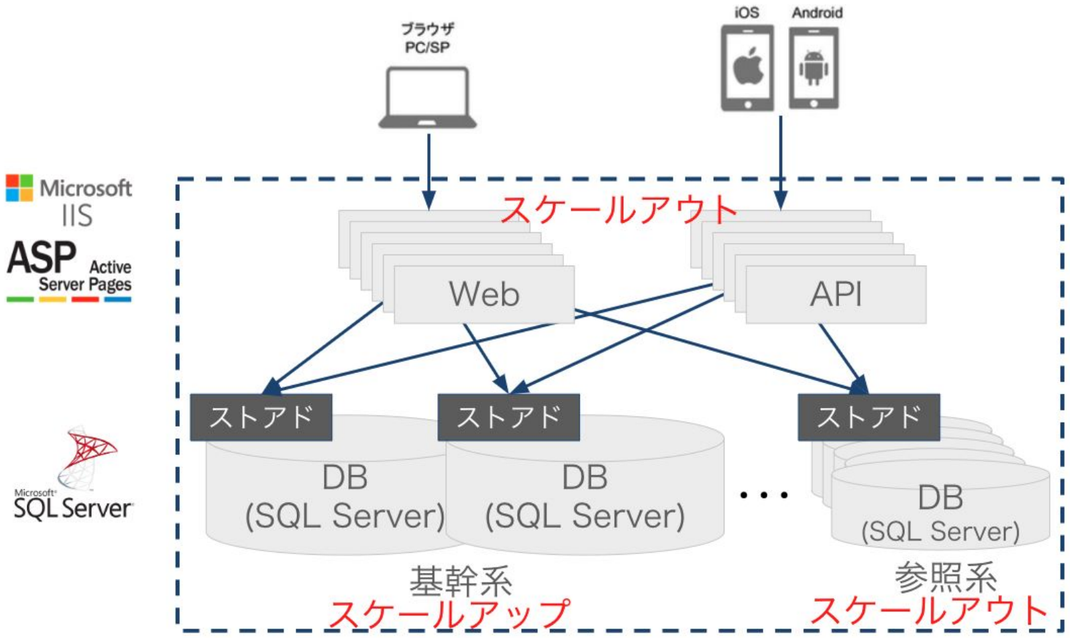
ASP Active Server Pages

Microsoft SQL Server



既存アーキテクチャの問題点

オンプレミス



- Web/APIサーバー
  - プログラムが**モノリシックで巨大**
  - 言語が**VBScript（ClassicASP）でありレガシー**
- DBサーバー
  - ASPだけでなくストアドプロシージャ（ストアド）にも**ロジック**が生まれ**2重管理**
  - DB・ストアドの両方の負荷を考慮したスケールが必要で、**スケールしづらい**
- 全体
  - **Windows Server**運用管理

…など

リプレイスの目的

お客様にいつでも快適に買い物をしていただけるサービスを提供し、ZOZOTOWNを成長させるため

システム観点での継続的に成長させるための4つのポイント

柔軟なシステム

- セールなどの高負荷時にスケール可能

開発生産性

- 並行開発
- リリースの自動化

技術のモダン化

- レガシー技術による独自実装の減少
- 汎用ミドルウェア、SaaSを利用可能

採用強化

- 技術情報のアウトプットによるプレゼンス向上

リプレイスの方針

- **クラウド化**
  - スケーラビリティ（拡張性）を手に入れる

柔軟なシステム

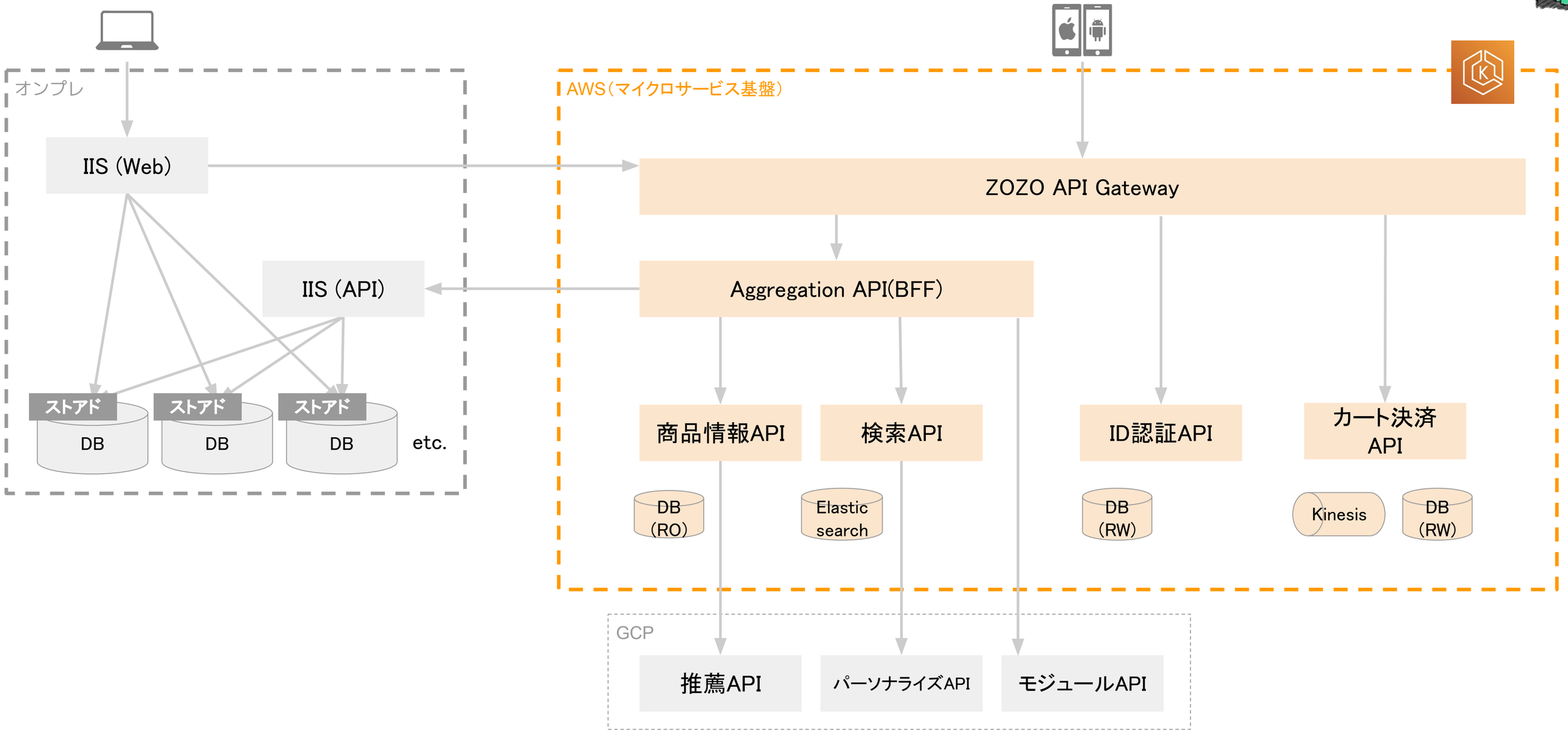
- **マイクロサービス化**
  - アプリケーションをモノリシック→マイクロサービス化することで、
    マイクロサービス（チーム）毎に並行で開発・リリースを可能にする

開発生産性

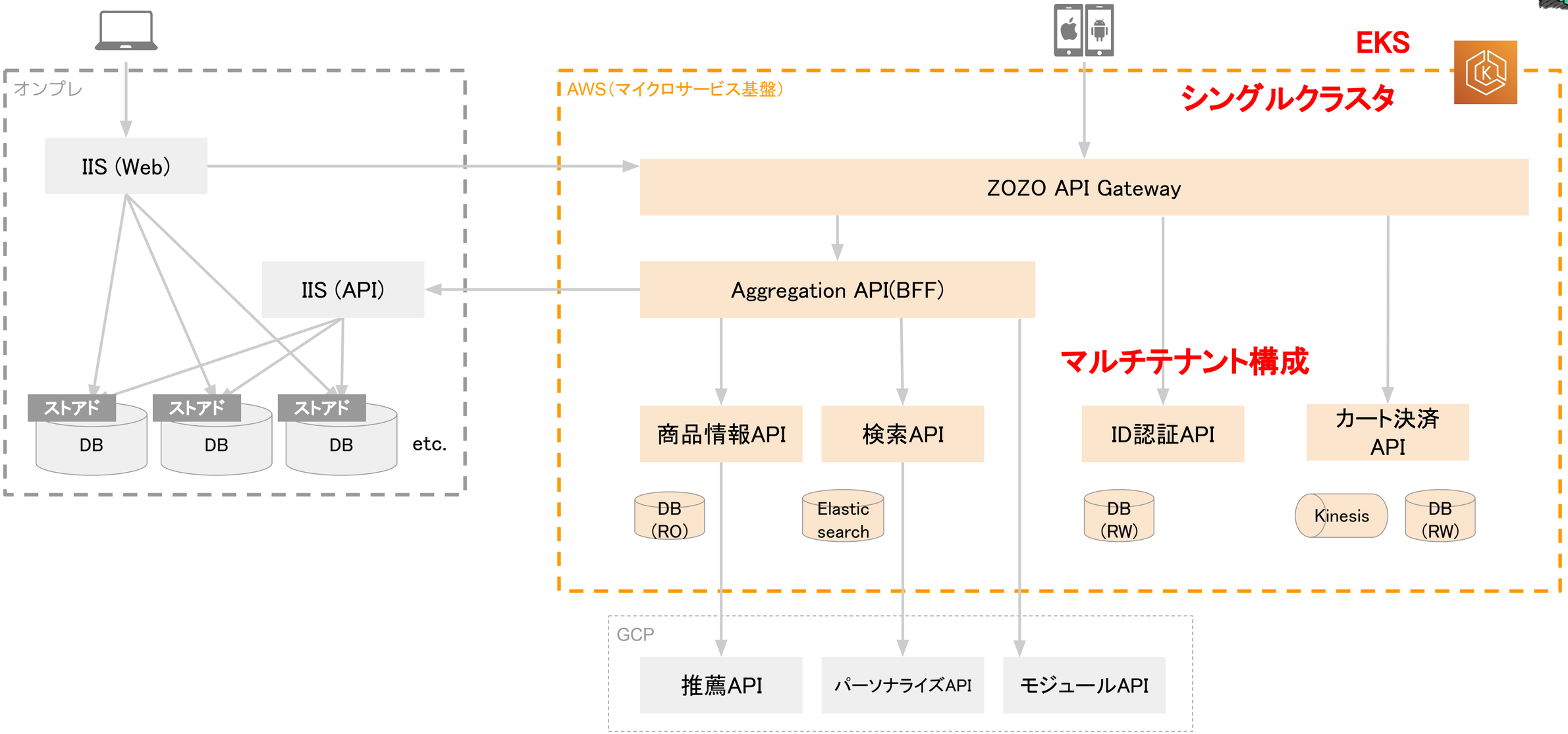
- **脱ClassicASP**
  - 事業継続が出来なくなるリスクヘッジ
  - システムのモダン化に合わせミドルウェアやSaasもモダン化する
    - サーバー：Kubernetes（K8s）
    - 言語：Java, Golang

技術のモダン化

アーキテクチャ（2022）

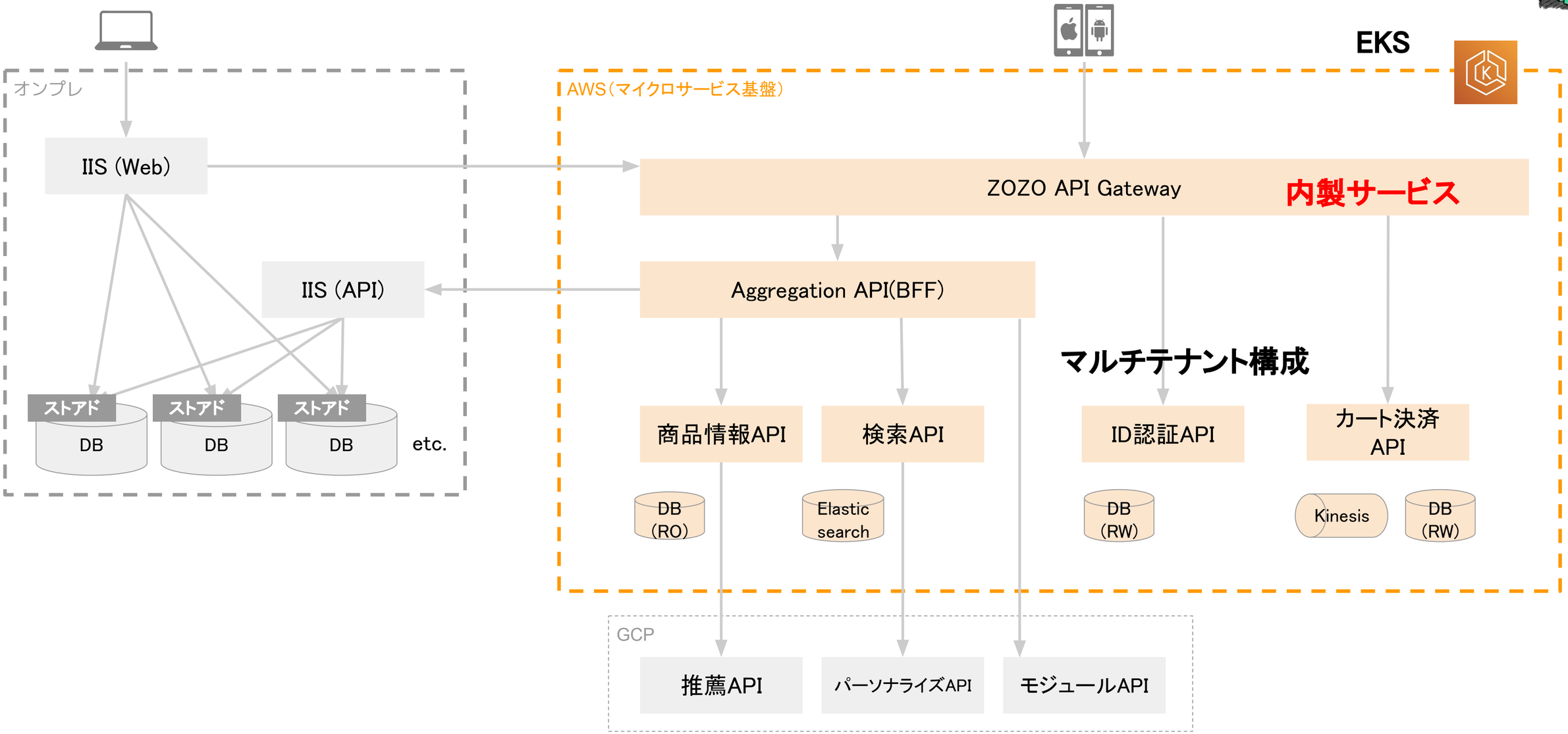


アーキテクチャ（2022）



設計の詳細は ZOZOTOWNにおけるAmazon EKSを中心に据えたマイクロサービスアーキテクチャへの変遷

アーキテクチャ（2022）



ZOZO API Gatewayの詳細は【ZOZOTOWNマイクロサービス化】API Gatewayを自社開発したノウハウ大公開！

アジェンダ

- ZOZOTOWNリプレイス
- **EKS使用の背景と運用状況**
- 大規模EKSクラスタ運用における信頼性・セキュリティ向上の取り組み
  - Design for Failure
  - 耐障害性を高める
  - トラフィック増加に備える
  - Security Groups For Podsによるマルチテナントのアクセス管理

EKSを使用した背景

- **AWSをメインで使う**
  - 社内でAWSをメインで使っている
  - 社内の経験から、Azure/GCPよりも設定が柔軟でエラー率が低い
- **コンテナのメリット**
  - 高い再現性
  - 依存OSも含めたパッケージ化
- **Kubernetesのメリット**
  - コンテナオーケストレーション基盤のデファクトスタンダード
  - 他のコンテナオーケストレーション基盤（具体的にはAmazon ECS）に比べて管理する項目は増えるが、OSSの利用など含めて自由度が高い
  - K8sのナレッジはクラウドに依存しないため社内でノウハウ共有が可能(GKEを利用しているチームも)

開発生産性

柔軟なシステム

EKSを使用した背景

- **AWSをメインで使う**
  - 社内でAWSをメインで使っている
  - 社内の経験から、Azure/GCPよりも設定が柔軟でエラー率が低い
- **コンテナのメリット**
  - 高い再現性
  - 依存OSも含めたパッケージ化
- **Kubernetesのメリット**
  - コンテナオーケストレーション基盤のデファクトスタンダード
  - 他のコンテナオーケストレーション基盤（具体的にはAmazon ECS）に比べて管理する項目は増えるが、OSSの利用など含めて**自由度が高い**
  - K8sのナレッジはクラウドに依存しないため**社内でノウハウ共有が可能**(GKEを利用しているチームも)

開発生産性

柔軟なシステム

AWS EKSを採用

マルチテナント構成の背景

- **シンプルな構成を目指す**
  - KISS(Keep it simple, stupid)
    マルチアカウント・マルチクラスタといった複雑な構成を避ける
- **リプレイス速度向上させる**
  - アカウント・クラスタ管理のオーバーヘッドを極力減らしマイクロサービス化を加速させる

マルチテナント構成の背景

- **シンプルな構成を目指す**
  - KISS(Keep it simple, stupid)の原則
    マルチアカウント・マルチクラスタといった複雑な構成を避ける
- **リプレイス速度向上させる**
  - アカウント・クラスタ管理のオーバーヘッドを極力減らしマイクロサービス化を加速させる

シングルクラスタ、マルチテナント構成を採用

マルチテナント構成の背景

- **シンプルな構成を目指す**
  - KISS(Keep it simple, stupid)の原則
    マルチアカウント・マルチクラスタといった複雑な構成を避ける
- **リプレイス速度向上させる**
  - アカウント・クラスタ管理のオーバーヘッドを極力減らしマイクロサービス化を加速させる
- **横串なSRE組織においてはデメリットも許容できた**
  - マルチテナントではチームをまたぐコミュニケーションコストの増加が懸念された
  - ZOZOTOWNのSREはマイクロサービスを横断する横串な1組織

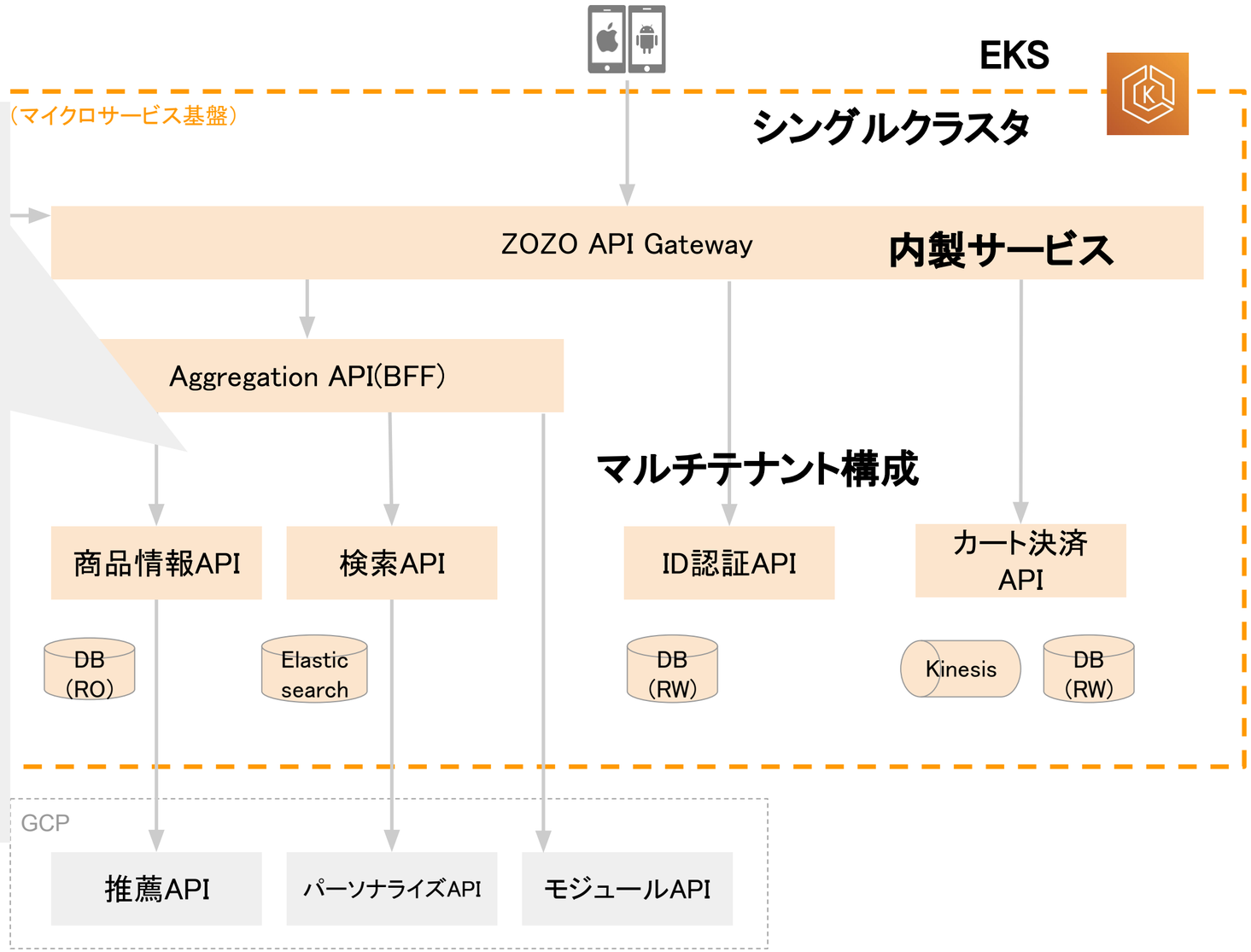
シングルクラスタ、マルチテナント構成を採用

アーキテクチャ規模（2022/7）

- Multi AZ
  - 3AZ
- microservices
  - 10程度
- Nodes
  - 200〜500
- Pods
  - 2000〜5000

※ Node、PodはAuto scaleや、セールに向けた増強によって増える

大規模



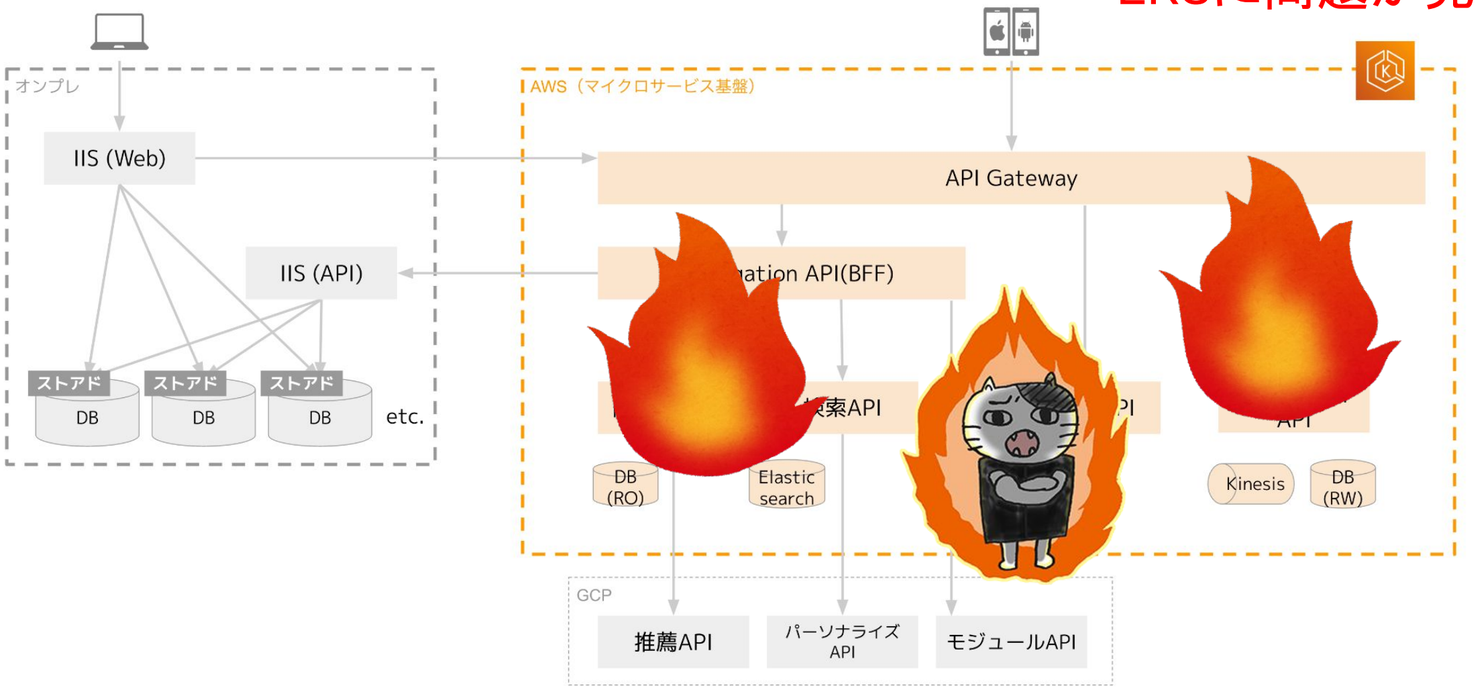
EKSの可用性・信頼性向上が求められる

- マイクロサービスは10を超え、今後も増加する
- 基盤となるEKSはZOZOTOWNの重要なコンポーネント

EKSの可用性・信頼性向上が求められる

- マイクロサービスは10を超え、今後も増加する
- 基盤となるEKSはZOZOTOWNの重要なコンポーネント

EKSに問題が発生



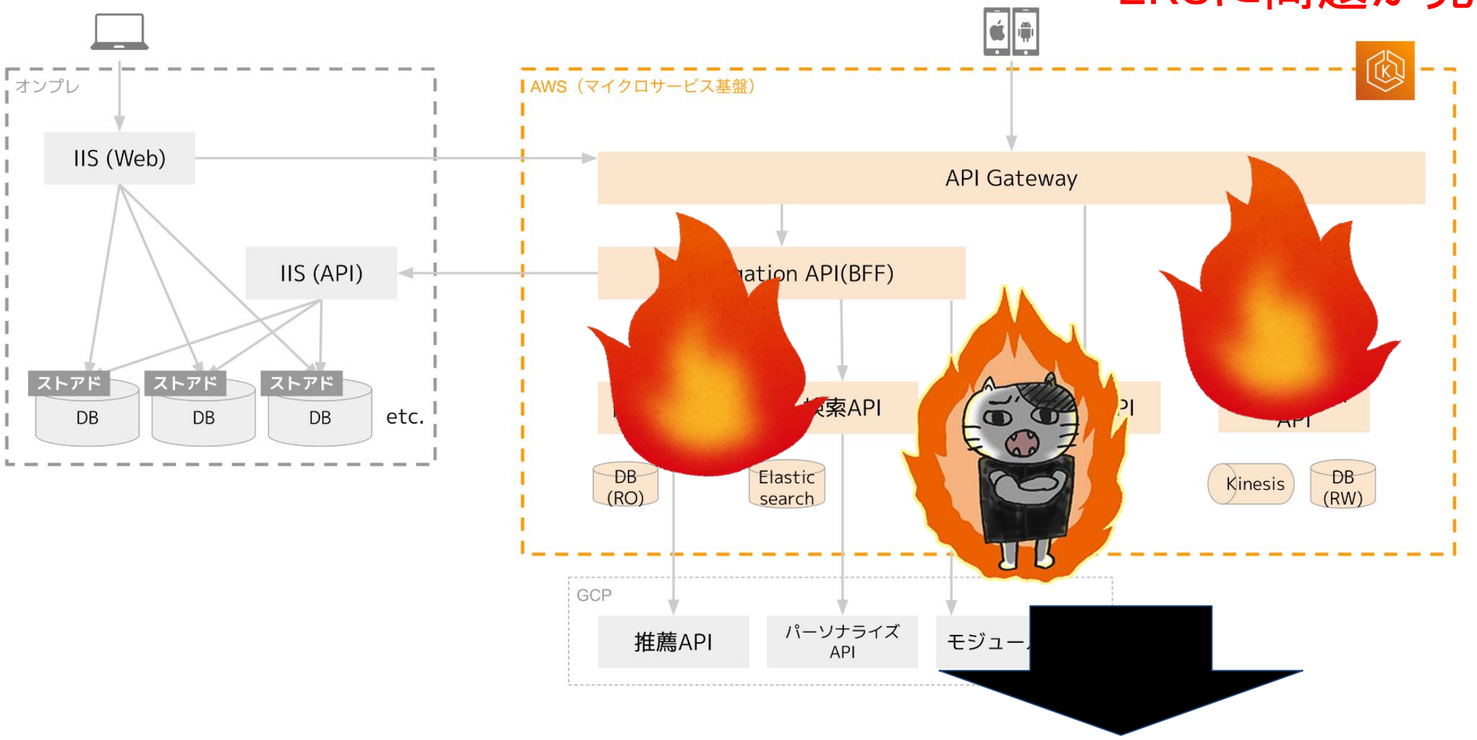
- ZOZOTOWN TOPページ真っ白
- アプリが起動エラー
- ログインが出来ない
- 商品検索が出来ない
- 商品詳細が見られない

・・・など

EKSの可用性・信頼性向上が求められる

- マイクロサービスは10を超え、今後も増加する
- 基盤となるEKSはZOZOTOWNの重要なコンポーネント

EKSに問題が発生



- ZOZOTOWN TOPページ真っ白
- アプリが起動エラー
- ログインが出来ない
- 商品検索が出来ない
- 商品詳細が見られない

・・・など

可用性・信頼性、セキュリティ向上が求められる

ここまでのまとめ

- ZOZOTOWNリプレイス
  - 目的：お客様にいつでも快適に買い物をしていただけるサービスを提供し、ZOZOTOWNを成長させるため
  - 方針3つ：クラウド化 / マイクロサービス化 / 脱ClassicASP
- EKS使用の背景
  - コンテナ、Kubernetesのメリットを享受
  - シンプルかつ管理オーバーヘッド削減のためシングルクラスタ、マルチテナントを採用
- EKSの運用状況
  - 大規模、規模拡大中
  - EKSクラスター障害 = ZOZOTOWN障害
  - 可用性・信頼性、セキュリティ向上が求められる

アジェンダ

- ZOZOTOWNリプレイス
- EKS使用の背景と運用状況
- **大規模EKSクラスタ運用における信頼性・セキュリティ向上の取り組み**
  - Design For Failure
  - 耐障害性を高める
  - トラフィック増加に備える
  - Security Groups For Podsによるマルチテナントのアクセス管理

株式会社ZOZO

技術本部 SRE部 ECプラットフォーム基盤SREブロック

織田 英吾

2022年4月、ZOZOTOWN Microservices PlatformのSREチームに中途入社

主にAWSやKubernetes、Istioを担当し、基盤やサービスの運用、改善を行っている

普段は、WezTerm + Tmux + Neovim で開発してます

アジェンダ

- Design for Failure
- 耐障害性を高める
- トラフィック増加に備える
- Security Groups For Podsによるマルチテナントのアクセス管理

アジェンダ

- **Design for Failure**
- 耐障害性を高める
- トラフィック増加に備える
- Security Groups For Podsによるマルチテナントのアクセス管理

Design for Failure

AWS Well-Architectedの信頼性の柱で定義されている Design for Failure を考慮

- インフラストラクチャまたはサービス障害からの復旧
- 必要に応じた動的なコンピューティングリソースの確保
- 設定ミスや一時的なネットワークの問題などによる障害の軽減

単にシステムを落とさないということでなく

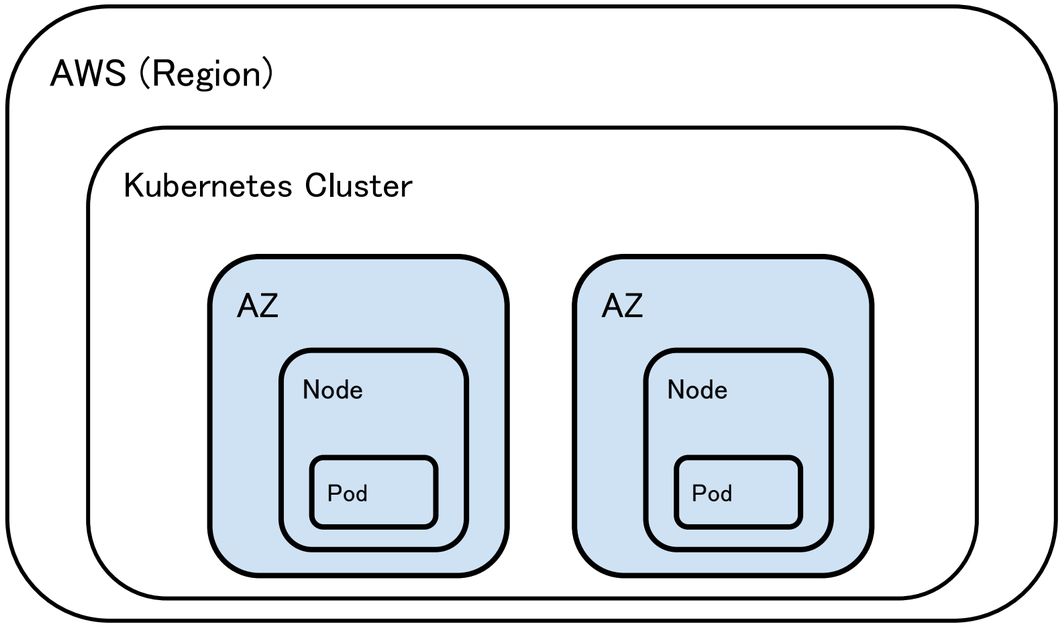
有事の際にどの様にリカバリ出来るのかといったことを予め考えておくことが重要

アジェンダ

- Design for Failure
- **耐障害性を高める**
- トラフィック増加に備える
- Security Groups For Podsによるマルチテナントのアクセス管理

耐障害性を高める – アジェンダ

- Multi AZ(3AZ)構成を取る
- AZ障害に備える
  - Node障害に備える
    - Auto Scaling groups
    - PodDisruptionBudget (PDB)
  - Pod障害に備える
    - Pod Topology Spread Constraints (PTSC)
  - AZ障害からの復旧後に再分散させる
    - Deschueduler



Multi AZ (3AZ) 構成を取る

Design for Failureのインフラストラクチャまたはサービス障害からの復旧の観点で
Multi AZ構成を採用している

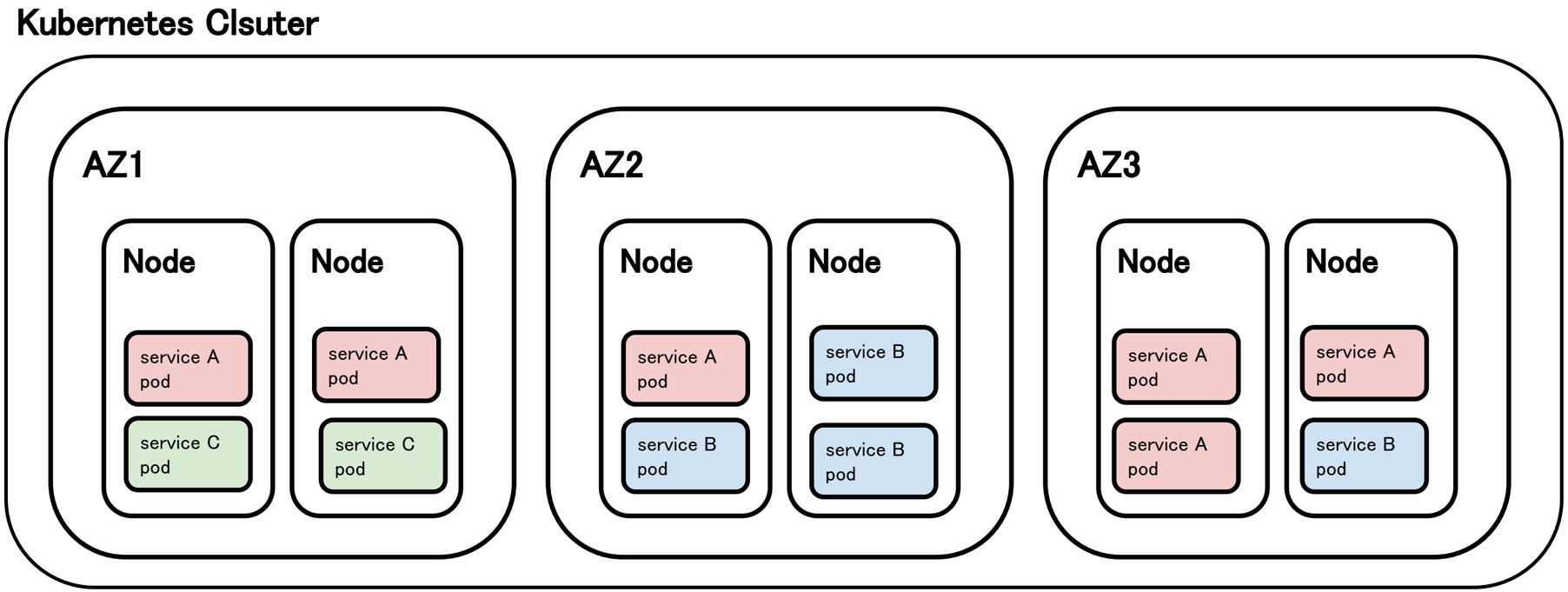
- ALBは最低2AZを指定する必要がある
- 1AZで障害が発生しても、システムへの影響を最小限にするため、3AZ構成としている
  - 障害時にも切り離しなどコントロールできるように、また1AZの障害における影響範囲の観点を含んでいる

AWS障害、“マルチAZ”なら大丈夫だったのか？　インフラエンジニアたちはどう捉えたか、生の声で分かった「実情」
https://www.itmedia.co.jp/news/articles/1908/28/news127_2.html

AZ障害に備える

Multi AZ構成にしていても、下図の様な場合、効果が得られないことが予見された

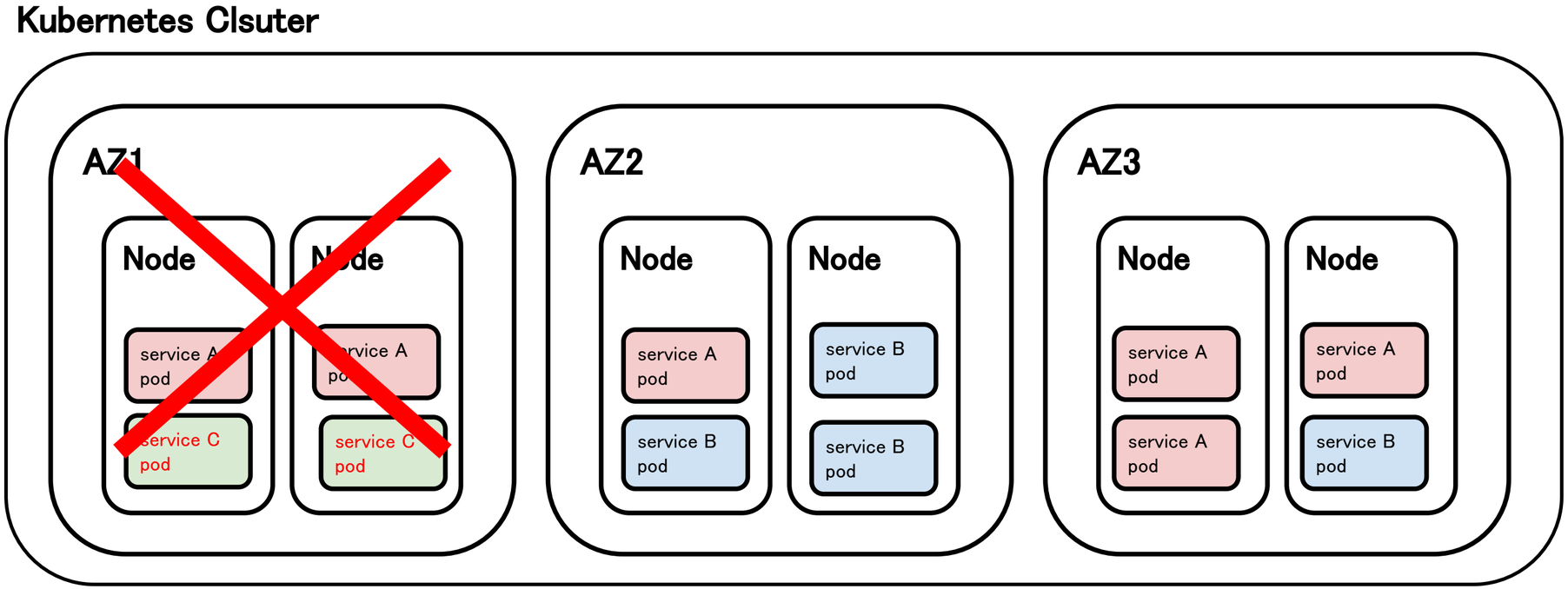
- AZ障害時、ALBのターゲットから当該AZを外す
- サービスの品質が低下する



AZ障害に備える

Multi AZ構成にしていても、下図の様な場合、効果が得られないことが予見された

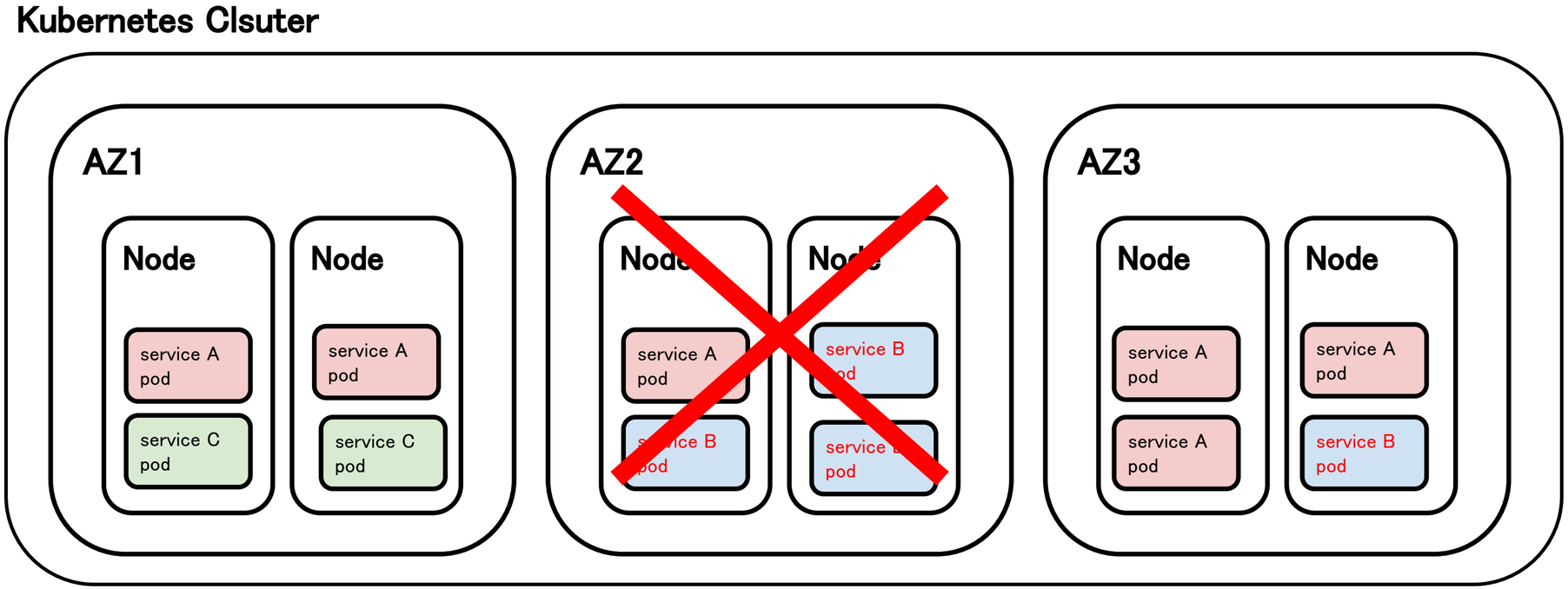
- AZ障害時、ALBのターゲットから当該AZを外す
- サービスの品質が低下する
  - サービスのPodが0
  - サービスのPodが半分より少なくなる



AZ障害に備える

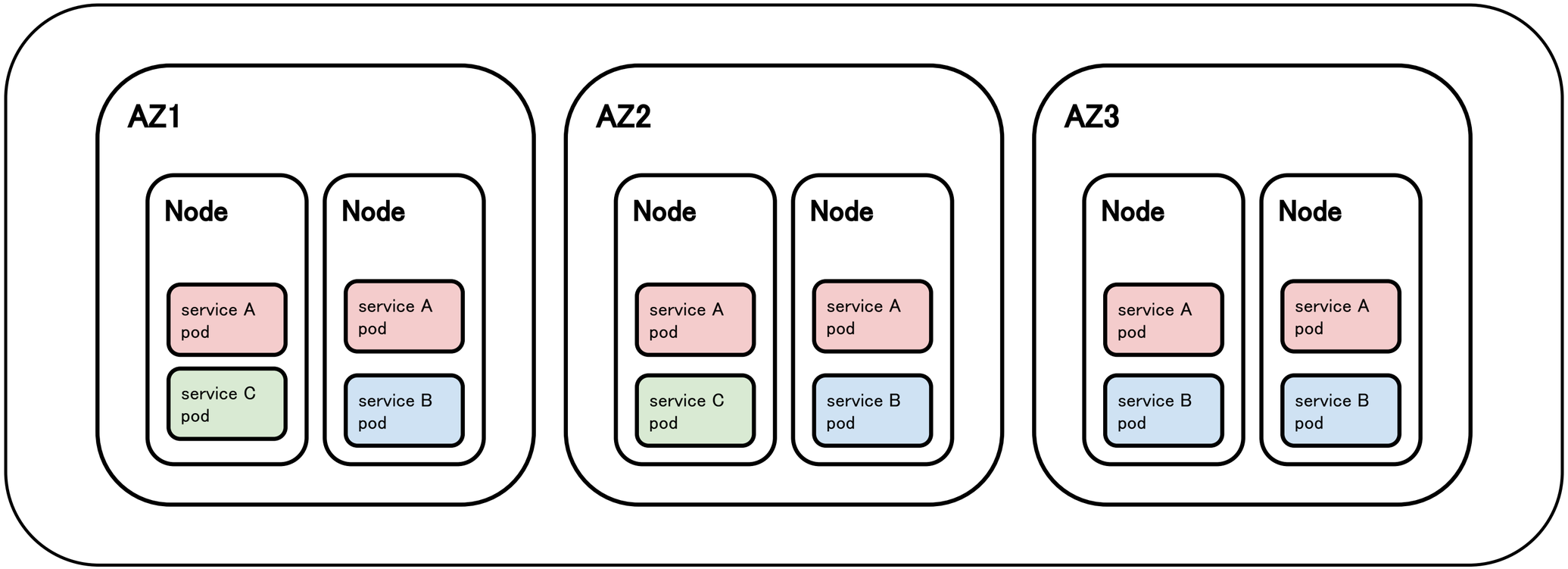
Multi AZ構成にしていても、下図の様な場合、効果が得られないことが予見された

- AZ障害時、ALBのターゲットから当該AZを外す
- サービスの品質が低下する
  - サービスのPodが0
  - サービスのPodが半分より少なくなる



AZ障害に備える

NodeやPodを均等に分散する必要がある



AZ障害に備える

Node障害に備える

- マネージド型 Node groupのAuto Scaling groupsを使用し、Nodeを分散
  - EKS Kubernetes Cluster Node (EC2 Instance)のプロビジョニングとライフサイクル管理を自動化
  - Nodeの終了または更新時にKubernetes APIを使用してNodeをdrainするため、Podを安全に退避
    - 複数のNodeが同時に停止する場合、状況によっては、アプリケーションPodが一つも動作していない状態になる可能性があるため注意が必要
      - PodDisruptionBudget (PDB)を設定し、最低限必要な Pod数を定義することで Nodeのdrainによるアプリケーションへの影響を最小限まで抑えることが可能
  - Cluster AutoscalerによるEKS Nodeのオートスケーリング

Safely Drain a Node

https://kubernetes.io/docs/tasks/administer-cluster/safely-drain-node/

AZ障害に備える

Pod障害に備える

- Pod Topology Spread Constraints (PTSC)を利用し、Podを分散
  - クラスタ全体にPodをどのように分散させるか制御するもの
  - Region・Zone・Nodeなどの単位でPodを分散配置することが可能
    - 弊チームでは、基本的にZone単位で分散させている
  - 可能な限りバランスよくPodを配置
  - 高可用性や効率的なリソース利用を実現することができる

Pod Topology Spread Constraints

https://kubernetes.io/docs/concepts/scheduling-eviction/topology-spread-constraints/

[Kubernetes] PodのAZ分散を実現するPod Topology Spread ConstraintsとDescheduler

https://zenn.dev/tmrekk/articles/07f30b09c26b50

AZ障害に備える

Pod障害に備える

- 使用例）PTSCを用いて、DeploymentをAZ間で分散する

```yaml
apiVersion: aps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  template:
    metadata:
      labels:
        app: nginx
    spec:
      topologySpreadConstraints:
        - topologyKey: topology.kubernetes.io/zone
          maxSkew: 1                                  # AZ間のPod分散の差をどこまで許容するか
          whenUnsatisfiable: DoNotSchedule            # maxSkew を満たせない場合のPodの対処方法
          labelSelector:
            matchLabels:
              app: nginx
```

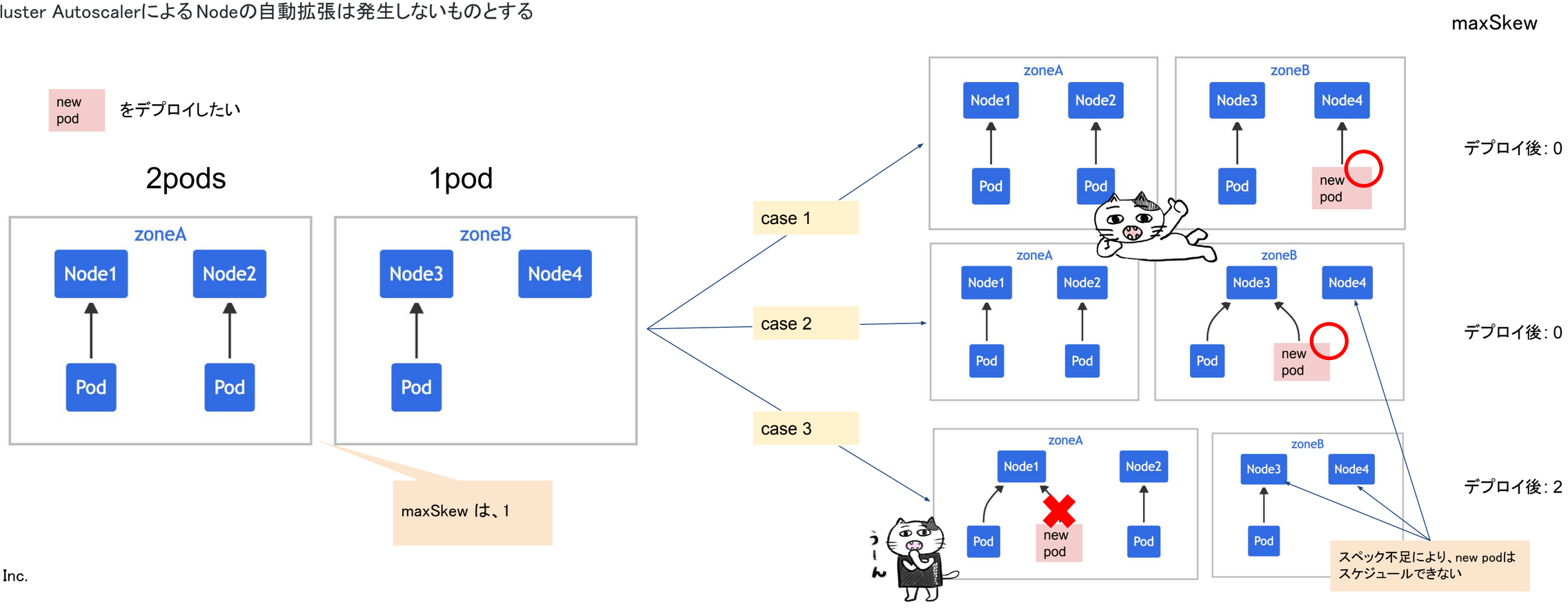
AZ障害に備える

Pod障害に備える

- PTSCでmaxSkewは、各AZ(zone)間で許容できるPod数の差分を設定

maxSkew = 1、whenUnsatisfiable = DoNotSchedule の場合

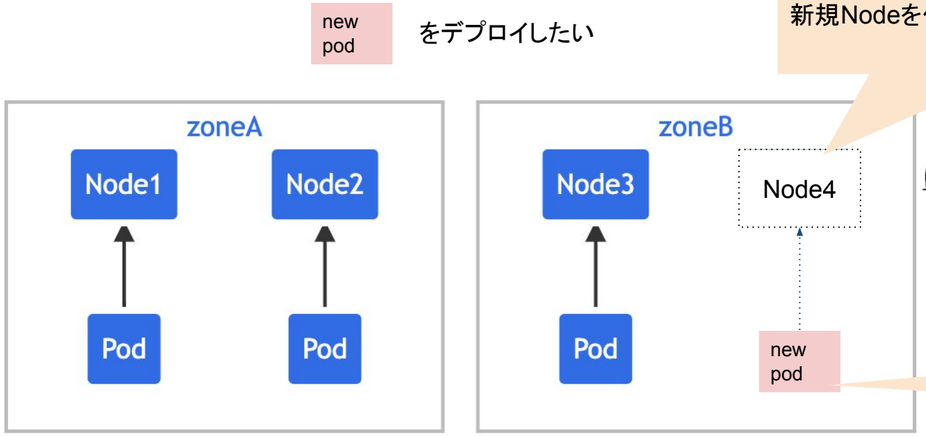
※ Cluster AutoscalerによるNodeの自動拡張は発生しないものとする



AZ障害に備える

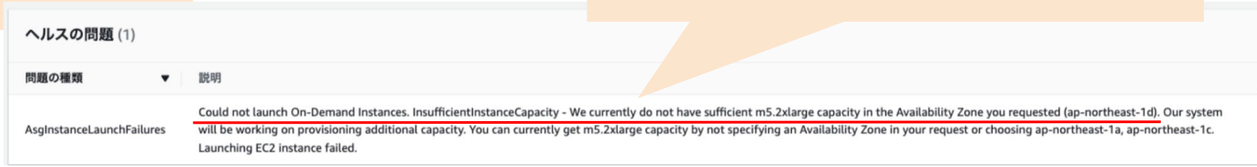
PTSCを設定したが、問題発覚...!

- 遭遇した問題
  - スケジュールされたPodがRunningにならず、Pending状態のまま進まなかった
- 原因
  - 特定AZにおいてEC2のキャパシティ不足で、Node(m5.2xlarge)が作成されない状態となっており Pod がスケジュールされずエラーが発生した
    - m5.2xlargeは約70Node稼働しており、追加で約50Node必要な状態だった



AZのEC2キャパシティ不足で
新規Nodeを作ることができない！

ap-northeast-1d の EC2(m5.2xlarg) がキャパシティ不足により起動できないエラーメッセージ

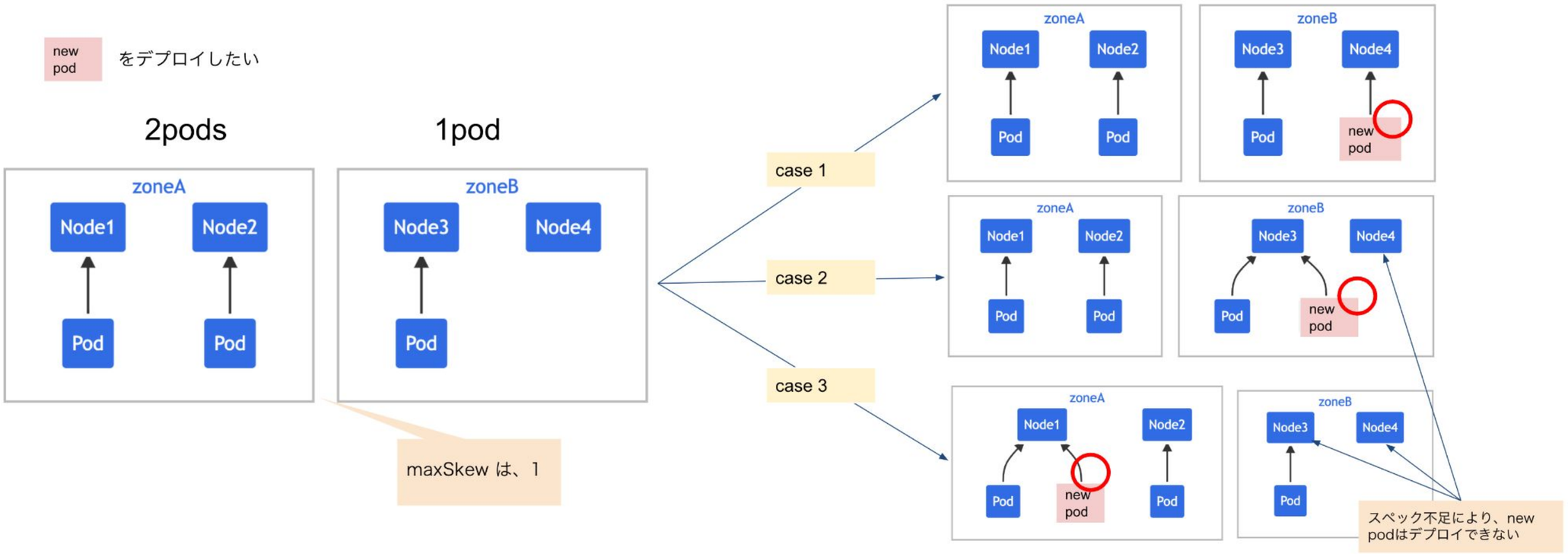


- 負荷分散の観点で、1Node 1Application podとしているため、Application pod がスケーリングするとNodeもスケーリングするようになっている
- whenUnsatisfiabl = DoNotSchedule と設定していて、maxSkew を満たせないため

AZ障害に備える

Pod障害に備える

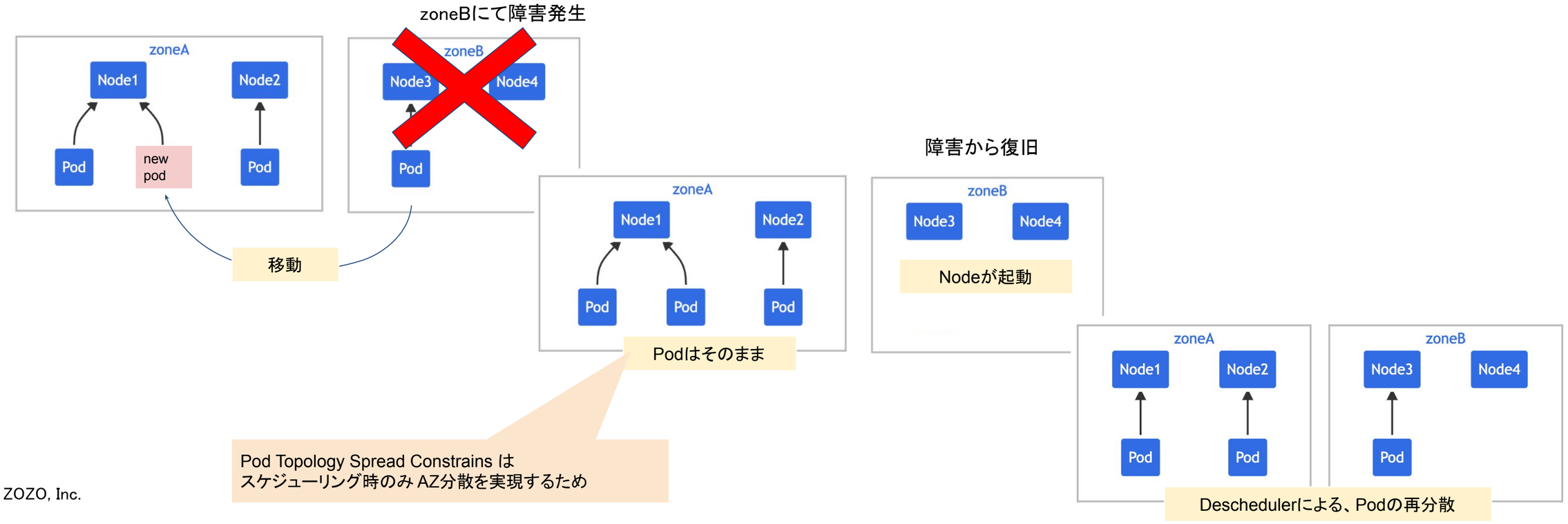
- 改善策
  - Podをスケジューリングさせることを最優先とするため
    whenUnsatisfiableをDoNotSchedule → ScheduleAnywayに変更した



AZ障害に備える

AZ障害からの復旧後に再分散させる

- 現在のPod分散状態に応じて、再配置するコンポーネントの導入
  - e.g.)
    - Descheduler (ZOZOではこれを用いてAZ障害復旧後の再分散を検討中)

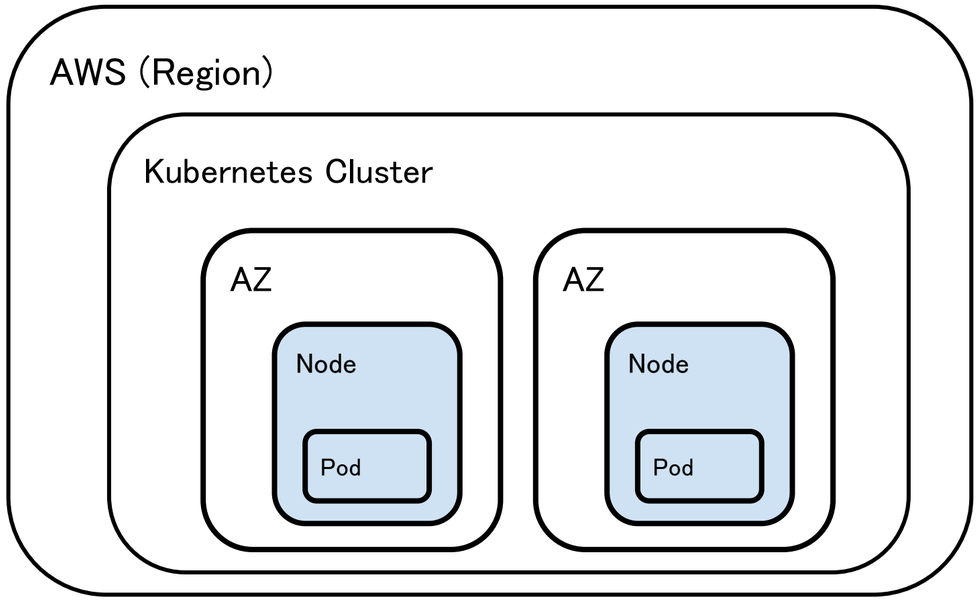


アジェンダ

- Design for Failure
- 耐障害性を高める
- **トラフィック増加に備える**
- Security Groups For Podsによるマルチテナントのアクセス管理

トラフィック増加に備える – アジェンダ

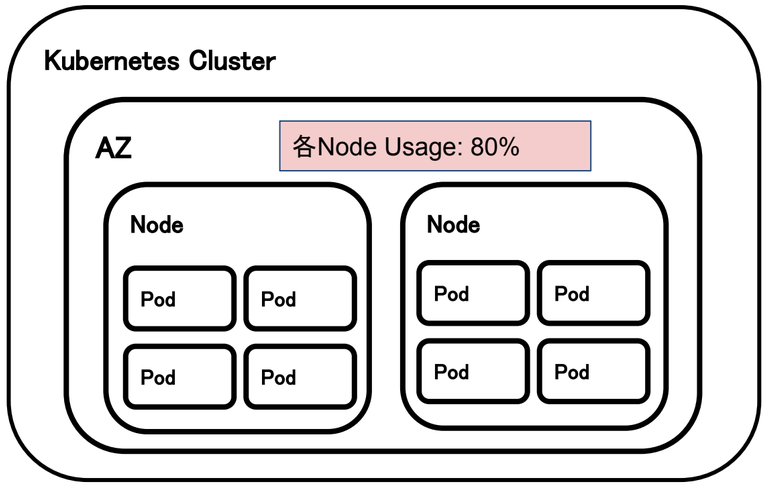
- オートスケールが必要な理由
- オートスケール
  - Nodeのオートスケール
    - Cluster Autoscaler (CA)
    - Karpenter
  - Podのオートスケール
    - Horizontal Pod Autoscaling (HPA)
  - ZOZOTOWNのオートスケーリング戦略
  - IPアドレス枯渇に備える
    - amazon-vpc-cni

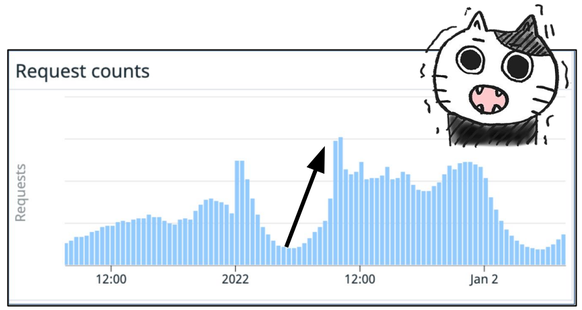


オートスケールが必要な理由

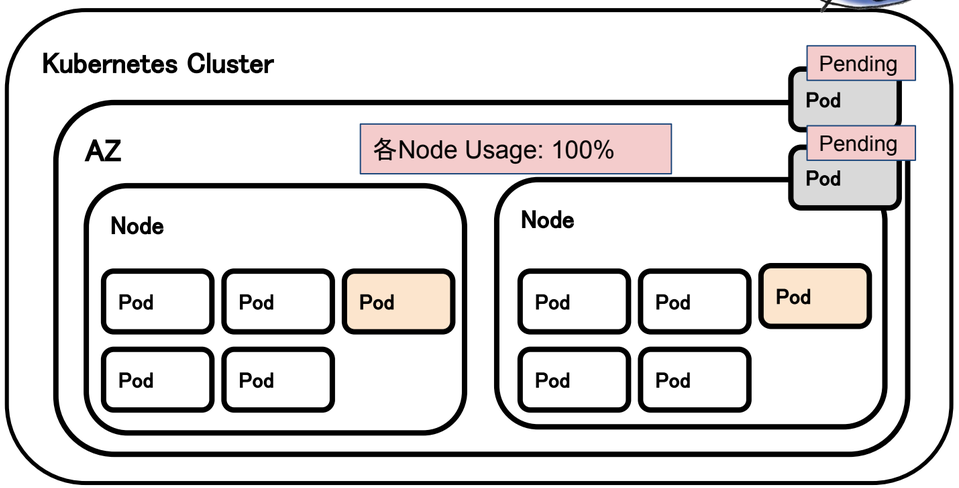
サービスのアクセス傾向、トラフィック傾向に合わせて、十分なサービススケールを用意できていないと、サービスに影響が出てしまう

Nodeがオートスケールしない例





急激なトラフィック



Nodeのオートスケール

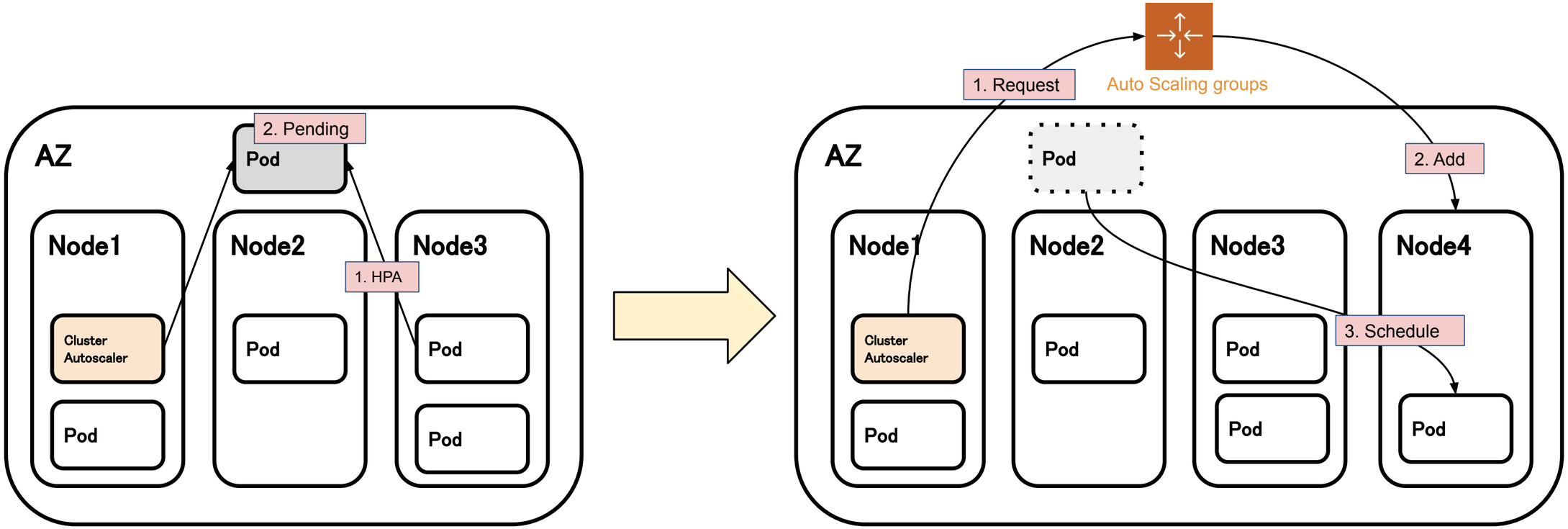
Cluster Autoscaler (CA)

- 必要な場合にNodeを追加し、オーバースペックな場合Nodeの削除を自動的に実施するコンポーネント
  - SIG autoscalingで管理されている
  - Podの要求リソースやNodeのキャパシティを比較し、Nodeの増減を行う
  - Node調整のための仕組みが必要
    - AWSの場合は、Auto Scaling groupsを利用

Cluster Autoscaler

Cluster Autoscalerの挙動

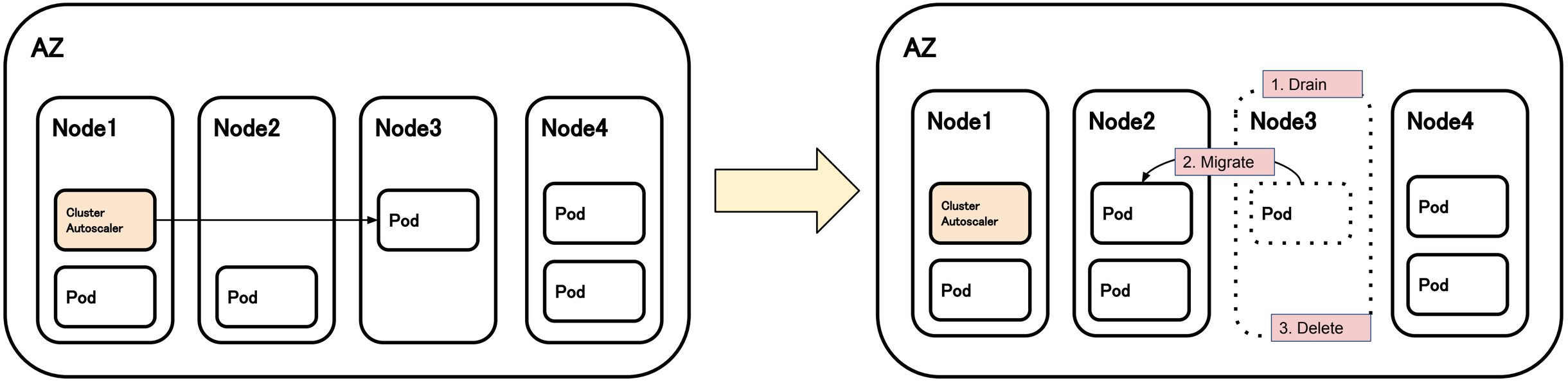
- Nodeが増加する場合
  - PodがPendingになったとき



Cluster Autoscaler

Cluster Autoscalerの挙動

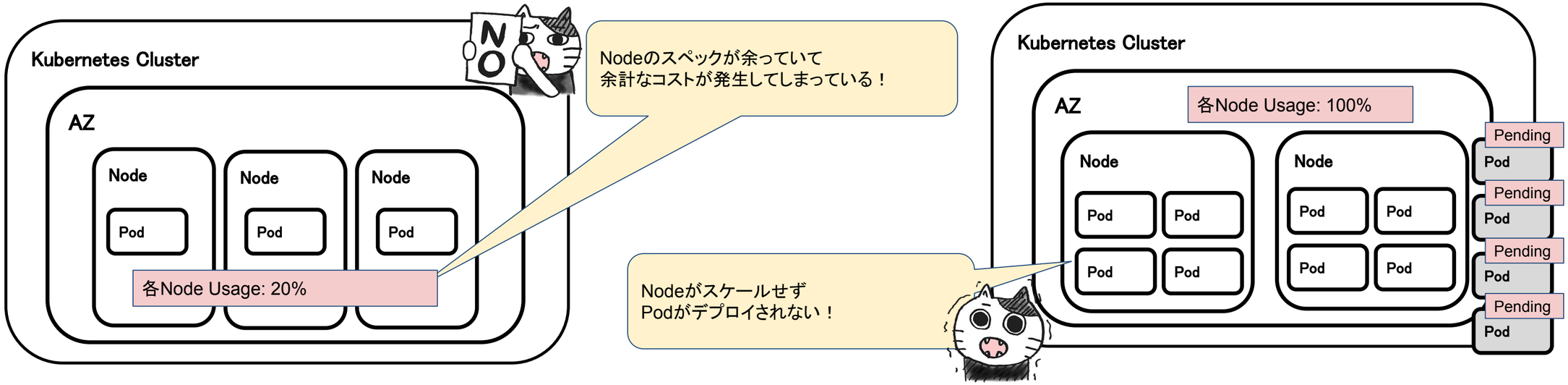
- Nodeが削除される場合
  - PodのRequest値が小さく、他のNodeへ移動できるとき



Cluster Autoscaler

Cluster Autoscaler のユースケース

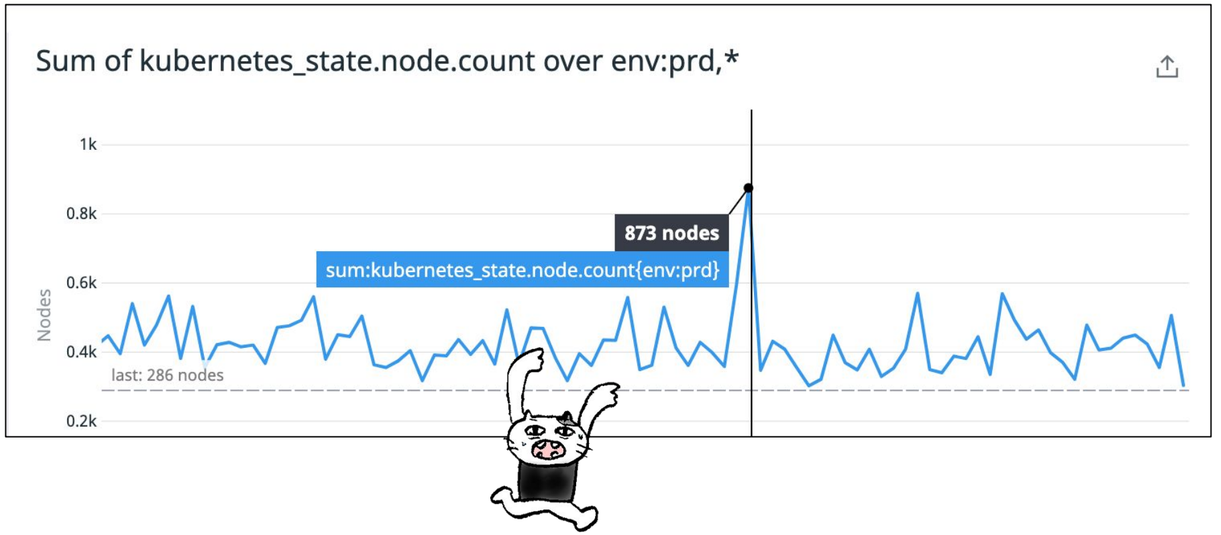
- メリット
  - コスト最適化
    - 必要なときに必要な分のみ
  - 予想外のトラフィック増加への対応
    - 知らぬ間にどこか(TV/YouTube etc.)で紹介されていて、トラフィックが増加しても、それに応じてオートスケールリング
- 注意
  - 1分で数倍になるような急激なトラフィック増加は、対応しきれない場合がある
    - 追加が必要な状態(PodがPending)になってからNodeを追加し始めるため、追加が完了するまで少し時間がかかる



Cluster Autoscaler

ZOZOTOWNでのユースケース

- 弊チームが管理している本番環境のEKS Nodeだけで通常時、約400Node
  セールなどのピーク時は、約900Nodeほどになるため、自動化するよう努めている
- 通常時のスケールに関しては、Cluster Autoscalerで事足りている
- ただし、冬のセールがZOZOTOWNのピークで、土日ピーク時を遥かに超えるアクセスとなり
  急激なアクセス増加で、スパイクに耐えられなかったことがあったため
  予め予測できる負荷に関しては、PodやNodeを事前に増やし、対応している



Nodeのオートスケール

Karpenter

- オープンソースのKubernetesクラスターオートスケーラー
- アプリケーションの負荷の変化に応じて適切なサイズのコンピューティングリソースを迅速に起動
- Cluster Autoscalerと比較して、追加のオーケストレーション(Auto Scaling groups)を必要としない
- 高速なスケーリングができるとされている

ノードスケーリングの高速化

- 課題
  - Nodeのスケール時間が長く、急激なトラフィック増加に対応できていないことがある
  - AWSを利用しているKubernetesのお客様の半数近くが、Kubernetes Cluster Autoscalerを使用してクラスターのオートスケーリングを設定するのは困難で制限的であると報告

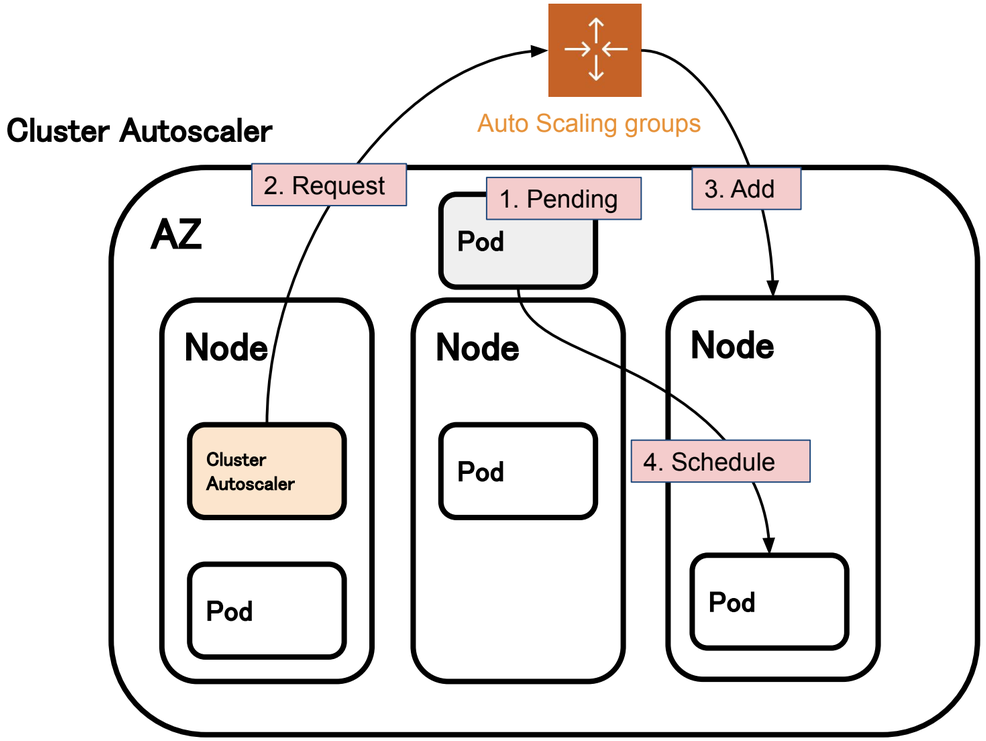
Karpenter のご紹介 – オープンソースの高性能 Kubernetes Cluster Autoscaler

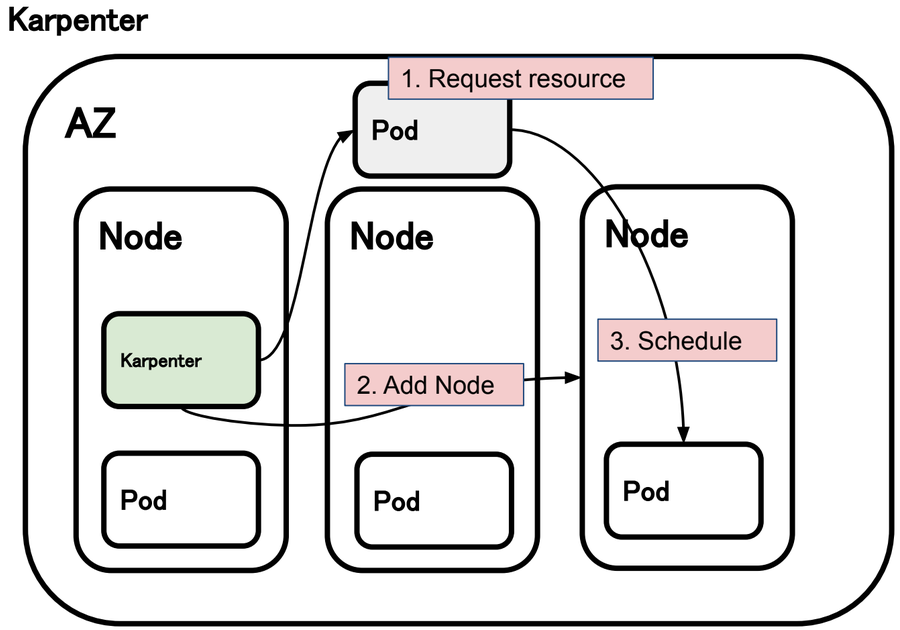
https://aws.amazon.com/jp/blogs/news/introducing-karpenter-an-open-source-high-performance-kubernetes-cluster-autoscaler/

Nodeのオートスケール

少し検証してみた

- m5.xlargeインスタンスで起動時間の比較
  - Cluster Autoscaler: 2分14秒、Karpenter: 63秒





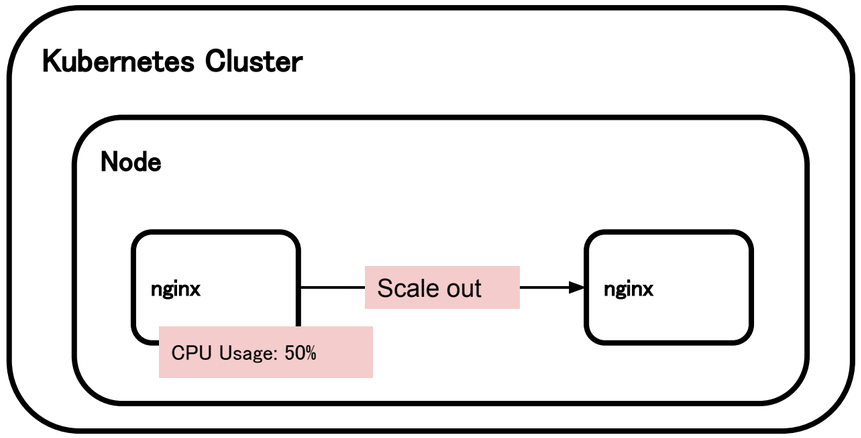
Karpenter – Concepts
https://karpenter.sh/v0.13.2/concepts/#kubernetes-cluster-autoscaler

Podのオートスケール

Horizontal Pod Autoscaling (HPA)

- 負荷に応じてPodを自動的に水平スケーリングさせられる
  - メモリ、CPUの使用率やカスタムメトリクスを基にスケーリング

- 使用例) CPUの使用率を基にDeploymentをスケーリング
  - Deploymentに設定された
    resources.requests.cpu: 1000m(1core)を100%とし
    平均使用率が50%(500m)に到達した場合、Podがスケーリング




```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
...
    spec:
      containers:
      - name: nginx
        image: nginx
        resources:
          requests:
            cpu: 1000m # 1core
```

```yaml
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: nginx-hpa
spec:
  maxReplicas: 10
  minReplicas: 1
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: nginx
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 50
```

Horizontal Pod Autoscaling

https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

ZOZOTOWNのオートスケーリング戦略

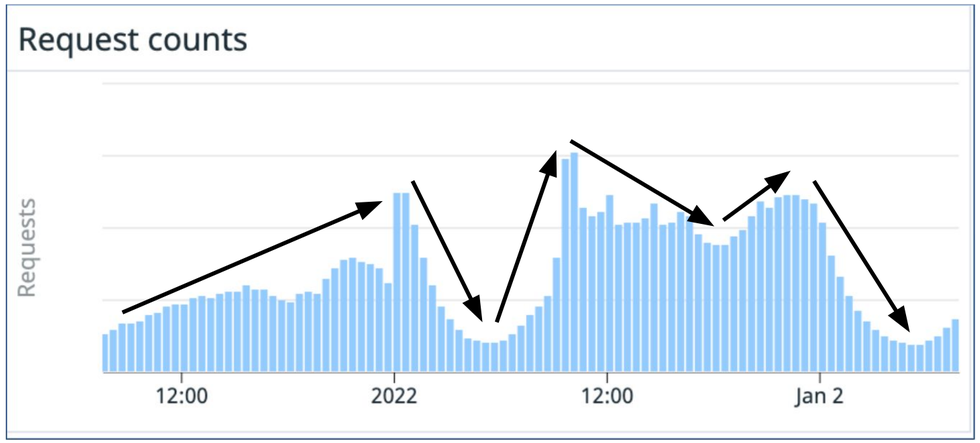
Nodeのオートスケール

- Cluster Autoscaler

Podのオートスケール

- Horizontal Pod Autoscaler

下図の様なトラフィック増加に可能な限り自動で対応できるようにしている



ZOZOTOWNのオートスケーリング戦略

オートスケーリングによって、動的なリクエスト変化への対応は出来るようになったが…

- **問題**
  - 2022年1月1日の冬セールに向けた負荷試験にて、サブネットのIPアドレス不足により、EKS Nodeの起動が失敗し、水平スケーリングで追加される予定のPodがPendingとなってしまった
    - サブネットで利用できるIPが約12,000 あったにも関わらず、発生してしまった

ZOZOTOWNで最大級のイベントである新春セールを乗り越えるための負荷試験とその効果

https://techblog.zozo.com/entry/zozotown-load-test

IPアドレス枯渇に備える

原因

- amazon-vpc-cni（deamonset aws-node）デフォルトの設定では、Nodeごとに **ENIが確保可能なIPv4アドレス数＊2ENI（デフォルト1 + WARM_ENI_TARGETのデフォルト値1）を確保**
  - スケーリングによってNodeが大量に増え、必要の要否にかかわらずNodeがIPを確保しているために追加予定のNodeに付与可能なIPがなく、起動に失敗していた
  - amazon-vpc-cni がデフォルトの場合、各インスタンスごとに最低限確保するIP数

インスタンスタイプ	確保するIP数
m5.large	20
m5.xlarge	30
m5.2xlarge	30
m5.4xlarge	60

負荷分散の観点で
1Node 1Application podとしているため、
Application pod がスケーリングするとNodeもスケーリングするようになっていることも関係している

IPアドレス枯渇に備える

IPアドレス枯渇に備える

- **改善策**
  - amazon-vpc-cniをEKS addonからmanifest管理へ移行
    - amazon-vpc-cniをカスタマイズするため
      - IP関連の環境変数を設定し、インスタンスが最低限確保するIPを必要な分のみに削減
    - EKS addonは環境変数をサポートしていないため（2022/02 時点）

- **結果**
  - IP枯渇が解消し、改善以降、エラーがでなくなった
  - IPが枯渇しそうになる前にアラートを発報し、事前にわかるように監視を設定

アジェンダ

- Design for Failure
- 耐障害性を高める
- トラフィック増加に備える
- **Security Groups For Podsによるマルチテナントのアクセス管理**

Security Groups For Podsによるマルチテナントのアクセス管理

背景

- マイクロサービスへのリプレイスが進む中で
  監査などを視野に入れつつ、マルチテナントでの権限管理の方針を整理し
  セキュリティリスクを抑えるため、細かく権限を制御する必要があった

Security Groups For Podsによるマルチテナントのアクセス管理

取り組んでいること

- 開発者の権限管理
  - Kubernetesリソースの操作
  - AWSリソースの操作
- マイクロサービスの権限管理
  - Pod からAWSリソースへのアクセス
  - Pod からKubernetes APIへのアクセス

Security Groups For Podsによるマルチテナントのアクセス管理

取り組んでいること

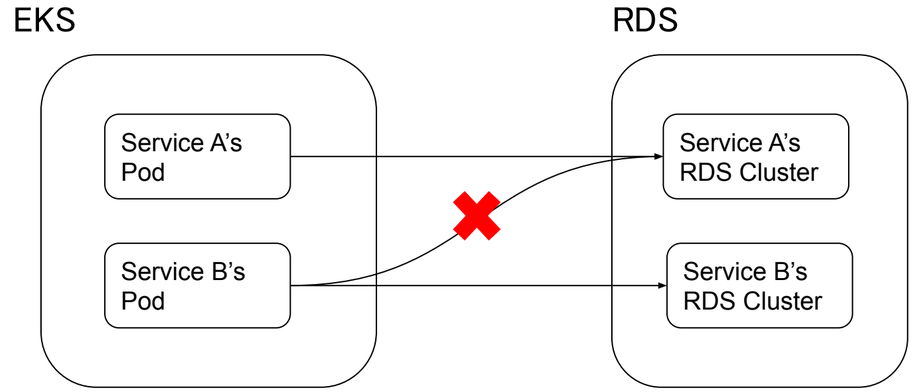
- 開発者の権限管理
  - Kubernetesリソースの操作
  - AWSリソースの操作
- **マイクロサービスの権限管理**
  - Pod からAWSリソースへのアクセス
  - Pod からKubernetes APIへのアクセス

Security Groups For Podsによるマルチテナントのアクセス管理

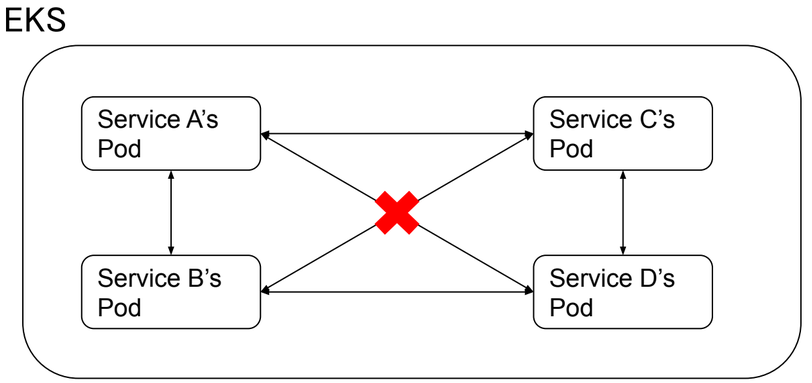
目的

- セキュリティリスクを最小限に抑える
  - 適切な権限のみを付与し、データベースが特定サービスのPodのみから接続を受け付ける
  - マイクロサービス間の通信ルートを絞る
  - など

Pod, RDS間での通信ルート制御



Pod間での通信ルート制御



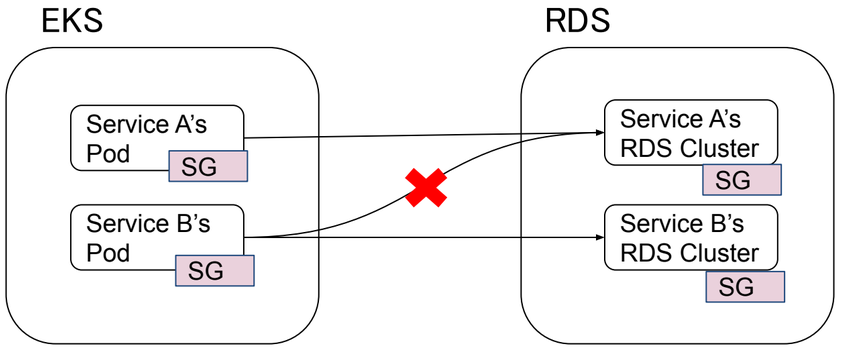
Security Groups For Podsによるマルチテナントのアクセス管理

Security Groups For Pods

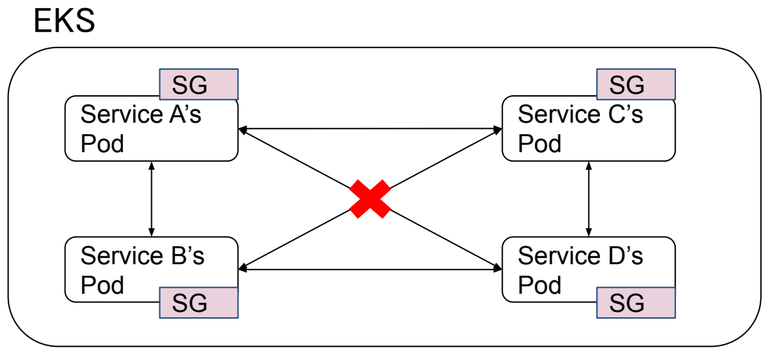
- EKSクラスタ内で実行されているPodにEC2セキュリティグループを割り当てる機能
- セキュリティグループを用いてネットワークトラフィック制御ができる

- 注意事項
  - AWSの機能なので、他のクラウドでは利用できない
  - 利用するにあたってクラスタのバージョン、インスタンスファミリーなど、考慮事項が多いため、事前に確認が必要

Pod, RDS間での通信ルート制御



Pod間での通信ルート制御



Security Groups For Podsによるマルチテナントのアクセス管理

使い方

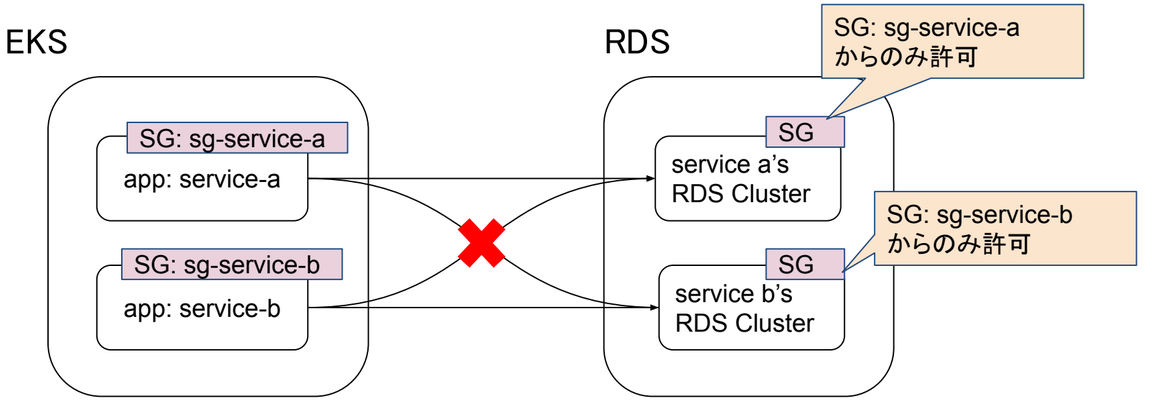
- SecurityGroupPolicy(CR)を作成し、podSelectorでセキュリティグループを割り当てるPodを選択

使用例)

- RDSへ通信できるPodを制限
  - app: service-a label を持つPodに対してセキュリティグループ(id: sg-service-a)を割り当て
  - app: service-b label を持つPodに対してセキュリティグループ(id: sg-service-b)を割り当て

```yaml
apiVersion: vpcresources.k8s.aws/v1beta1
kind: SecurityGroupPolicy
metadata:
  name: service-a-security-group-policy
  namespace: service-a
spec:
  podSelector:
    matchLabels:
      app: service-a
  securityGroups:
    groupIds:
      - sg-service-a
```

```yaml
apiVersion: vpcresources.k8s.aws/v1beta1
kind: SecurityGroupPolicy
metadata:
  name: service-b-security-group-policy
  namespace: service-b
spec:
  podSelector:
    matchLabels:
      app: service-b
  securityGroups:
    groupIds:
      - sg-service-b
```



おわりに

ZOZOTOWNリプレイスから見る大規模マルチテナントEKSクラスタの運用で重要なこと

- **Design for Failure**
  - 有事の際にどの様にリカバリ出来るのかを考え設計
    - ➢ **AZ障害に対するリカバリプランとして、Node/PodのAZ分散を実装**
- **スケーリング戦略による可用性の担保**
  - サービスやユーザ特性を踏まえて、トラフィック変化に備える
    - ➢ **通常のトラフィックは CA、HPA で対処**
    - ➢ **セールイベント時は事前に minReplicas を上げておく戦略**
- **マルチテナントのセキュリティ向上**
  - 各種リソースへの適切な権限付与を行い、安全なクラスタ運用を行う
    - ➢ **Security Groups For Pods を活用し、よりセキュアな基盤を構築**

ZOZO