Amazon EKS アプリケーションの HA と耐障害性のための設計

AWS 規範ガイダンス

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS 規範ガイダンス: Amazon EKS アプリケーションの HA と耐障害性のための設計

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon の商標およびトレードドレスは Amazon 以外の製品およびサービスに使用することはできません。また、お客様に誤解を与える可能性がある形式で、または Amazon の信用を損なう形式で使用することもできません。Amazon が所有していないその他のすべての商標は Amazon との提携、関連、支援関係の有無にかかわらず、それら該当する所有者の資産です。

Table of Contents

序章 1
HA とレジリエンスの設計 2
ワークロードの分散 2
ポッドトポロジのスプレッド制約を使用する 3
ポッドアフィニティとアンチアフィニティ 7
ポッド中断予算 9
プローブとヘルスチェック 9
起動プローブ 10
ライブネスプローブ 10
準備状況プローブ 10
Ingress リソースとロードバランサーのヘルスチェック 11
コンテナライフサイクルフック 11
ゾーン中断中のポッドエビクションを理解する 13
Amazon EKS ゾーンシフトを実装して耐障害性を向上させる 14
ゾーンシフトメカニズムについて 14
ゾーンシフトのアクティベーション方法 15
効果的なゾーンシフトの前提条件 15
ゾーン中断の回復力に関する推奨事項 16
シフトの完了と復旧 16
結論 18
リソース 19
ドキュメント履歴 20
..... xxi

Amazon EKS アプリケーションの高可用性と耐障害性のための設計

Haofei Feng、Frank Fan、Rus Kalakutskiy、Amazon Web Services (AWS)

2025 年 10 月 (ドキュメント履歴)

アプリケーション設計で高可用性 (HA) と回復性を確保することは、目標復旧時点 (RPO) と目標復旧時間 (RTO) をほぼゼロにするには不可欠です。組織がアプリケーションを Kubernetes 環境に移行してモダナイズするにつれて、堅牢でスケーラブルなソリューションに対する需要は増え続けています。Amazon Elastic Kubernetes Service (Amazon EKS) を使用すると、コンテナ化されたアプリケーションを大規模に効率的に管理できます。

このガイドでは、Amazon EKS マイクロサービスアプリケーションを設計および管理するための広く認められている推奨事項とベストプラクティスのセットについて詳しく説明します。豊富な経験と実際のデプロイに基づいて、これらのインサイトはアーキテクトや開発者にとって貴重なガイダンスを提供します。これらの推奨事項を実装して、Kubernetes ベースのアプリケーションのパフォーマンス、信頼性、スケーラビリティを高め、堅牢なオペレーションを実現します。

高可用性と耐障害性の設計上の考慮事項

責任共有モデルは Kubernetes でより複雑になります。Amazon EKS コントロールプレーンの可用性と耐障害性は、Amazon Web Services () によって管理されますAWS。組織はデータプレーンを管理し、マイクロサービスアプリケーションのパフォーマンスと可用性に大きな影響を与える可能性があります。

Amazon EKS で可用性と回復力の高いアプリケーションを設計する場合は、次のコンポーネントを検討してください。

- マイクロサービスアプリケーション: ポッドとコンテナ
- ワークロードデータプレーン: Ingress Controller、ポッド、Amazon Virtual Private Cloud (Amazon VPC) Container Network Interface (CNI)、サービスメッシュサイドカー、kube-proxy などのシステムコンポーネント
- ワークロード管理レイヤー: コントローラー、アドミッションコントローラー、ネットワークポリシーエンジン、およびこれらのコンポーネントの永続的データストレージ
- Kubernetes コントロールプレーン
- インフラストラクチャ: ノード、ネットワーク、ネットワークアプライアンス

このガイドでは、Kubernetes クラスター内で実行されるコンポーネントを参照する最初の 3 つの考慮事項について、以下のトピックについて説明します。

- ノードとアベイラビリティーゾーンにワークロードを分散する
- PDB による重要なワークロードの保護
- プローブとヘルスチェックの設定
- コンテナライフサイクルフックの設定
- ゾーン中断中のポッドエビクションについて

ノードとアベイラビリティーゾーンにワークロードを分散する

アベイラビリティーゾーンやノードなどの障害ドメイン間でワークロードを分散することで、コンポーネントの可用性が向上し、水平方向にスケーラブルなアプリケーションの障害の可能性が軽減されます。以下のセクションでは、ノードとアベイラビリティーゾーンにワークロードを分散する方法について説明します。

ポッドトポロジのスプレッド制約を使用する

Kubernetes ポッドトポロジの分散制約により、Kubernetes スケジューラはStatefulSet、異なる障害ドメイン (アベイラビリティーゾーン、ノード、およびハードウェアのタイプ) によって管理されるポッドReplicaSetを分散するように指示されます。ポッドトポロジの分散制約を使用する場合は、次の操作を実行できます。

- アプリケーションの要件に応じて、さまざまな障害ドメインにポッドを分散または集中させます。たとえば、レジリエンスのためにポッドを配布し、ネットワークパフォーマンスのためにポッドを集中させることができます。
- アベイラビリティーゾーン間での分散やノード間での分散など、さまざまな条件を組み合わせます。
- 条件が満たされない場合は、優先アクションを指定します。
  - maxSkew と の組み合わせwhenUnsatisfiable: DoNotScheduleで を使用してminDomains、スケジューラのハード要件を作成します。
  - whenUnsatisfiable: ScheduleAnyway を使用して を減らしますmaxSkew。

障害ゾーンが使用できなくなった場合、そのゾーンのポッドは異常になります。Kubernetes は、可能であればスプレッド制約に従ってポッドを再スケジュールします。

次のコードは、アベイラビリティーゾーン間またはノード間でポッドトポロジースプレッド制約を使用する例を示しています。

```
...
spec:
  selector:
    matchLabels:
      app: <your-app-label>
    replicas: 3
    template:
      metadata:
        labels: <your-app-label>
      spec:
        serviceAccountName: <ServiceAccountName>
...
        topologySpreadConstraints:
        - labelSelector:
            matchLabels:
              app: <your-app-label>
```

```
          maxSkew: 1
          topologyKey: topology.kubernetes.io/zone # <---spread those pods evenly over
all availability zones
          whenUnsatisfiable: ScheduleAnyway
        - labelSelector:
            matchLabels:
              app: <your-app-label>
          maxSkew: 1
          topologyKey: kubernetes.io/hostname # <---spread those pods evenly over all
nodes
          whenUnsatisfiable: ScheduleAnyway
```

クラスター全体のトポロジのデフォルトの分散制約

デフォルトでは、Kubernetes はノードとアベイラビリティーゾーン間でポッドを分散するためのトポロジースプレッド制約のセットを提供します。

```
defaultConstraints:
  - maxSkew: 3
    topologyKey: "kubernetes.io/hostname"
    whenUnsatisfiable: ScheduleAnyway
  - maxSkew: 5
    topologyKey: "topology.kubernetes.io/zone"
    whenUnsatisfiable: ScheduleAnyway
```

> **Note**
>
> さまざまなタイプのトポロジ制約を必要とするアプリケーションは、クラスターレベルのポリシーを上書きできます。

デフォルトの制約は高い を設定します。これはmaxSkew、少数のポッドを持つデプロイには役に立ちません。現時点では、Amazon EKS では KubeSchedulerConfiguration を変更できません。他のトポロジスプレッド制約のセットを適用する必要がある場合は、以下のセクションのように変更アドミッションコントローラーを使用することを検討してください。代替スケジューラを実行する場合は、デフォルトのトポロジスプレッド制約を制御することもできます。ただし、カスタムスケジューラを管理すると複雑さが増し、クラスターの耐障害性と HA に影響する可能性があります。このような理由から、トポロジの分散制約にのみ代替スケジューラを使用することはお勧めしません。

トポロジの分散制約のゲートキーパーポリシー

トポロジの分散制約を適用するもう 1 つのオプションは、Gatekeeper プロジェクトのポリシーを使用することです。ゲートキーパーポリシーはアプリケーションレベルで定義されます。

次のコード例は、デプロイ用のGatekeeper OPAポリシーの使用を示しています。必要に応じてポリシーを変更できます。たとえば、 というラベルのデプロイにのみポリシーを適用するかHA=true、別のポリシーコントローラーを使用して同様のポリシーを記述します。

この最初の例は、 でConstraintTemplate使用されている を示しています k8stopologyspreadrequired_template.yml。

```
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8stopologyspreadrequired
spec:
  crd:
    spec:
      names:
        kind: K8sTopologySpreadRequired
      validation:
        openAPIV3Schema:
          type: object
          properties:
            message:
              type: string
  targets:
    - target: admission.k8s.gatekeeper.sh
      rego: |
        package k8stopologyspreadrequired

        get_message(parameters, _default) =3D msg {
          not parameters.message
          msg :=_default
        }


        get_message(parameters, _default) =3D msg {
          msg := parameters.message
        }

        violation[{"msg": msg}] {
```

```
          input.review.kind.kind ="Deployment"
          not input.review.object.spec.template.spec.topologySpreadConstraint
          def_msg :"Pod Topology Spread Constraints are required for Deployments"
          msg :get_message(input.parameters, def_msg)

        }
```

次のコードは、YAML `constraints` マニフェスト を示しています`k8stopologyspreadrequired_constraint.yml`。

```
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sTopologySpreadRequired
metadata:
  name: require-topologyspread-for-deployments
spec:
  match:
    kinds:
      - apiGroups: ["apps"]
        kinds: ["Deployment"]
    namespaces:  ## Without theses two lines will apply to the whole cluster
      - "example"
```

トポロジスプレッド制約を使用するタイミング

以下のシナリオでは、トポロジースプレッド制約の使用を検討してください。

- 水平方向にスケーラブルなアプリケーション (ステートレスウェブサービスなど)
- アクティブ/アクティブまたはアクティブ/パッシブレプリカを持つアプリケーション (NoSQL データベースやキャッシュなど)
- スタンバイレプリカを持つアプリケーション (コントローラーなど)

水平方向にスケーラブルなシナリオに使用できるシステムコンポーネントには、次のようなものがあります。

- Cluster Autoscaler と Karpenter (`replicaCount > 1`と を使用`leader-elect = true`)
- AWS Load Balancerコントローラー
- CoreDNS

ポッドアフィニティとアンチアフィニティ

場合によっては、特定のタイプのポッドがノードで実行されないようにすることが有益です。たとえば、同じノードで複数のネットワーク負荷の高いポッドをスケジュールしないようにするには、ラベル `Ingress`または でアンチアフィニティルールを使用できます`Network-heavy`。を使用する場合は`anti-affinity`、次の組み合わせを使用することもできます。

- ネットワーク最適化ノードのテイント
- ネットワーク負荷の高いポッドでの対応する許容範囲
- ネットワーク負荷の高いポッドがネットワーク最適化インスタンスを使用するようにするためのノードアフィニティまたはノードセレクタ

ネットワーク負荷の高いポッドが例として使用されます。GPU、メモリ、ローカルストレージなど、要件が異なる場合があります。その他の使用例と設定オプションについては、Kubernetes ドキュメントを参照してください。

ポッドの再調整

このセクションでは、Kubernetes クラスター内のポッドを再調整するための 2 つのアプローチについて説明します。1 つ目は Descheduler for Kubernetes を使用します。Descheduler は、トポロジの分散制約やアンチアフィニティルールに違反するポッドを削除する戦略を強制することで、ポッドの分散を維持するのに役立ちます。2 番目のアプローチでは、Karpenter 統合とビンパッキング機能を使用します。統合は、ワークロードをより少ない、より効率的にパックされたノードに統合することで、リソースの使用状況を継続的に評価および最適化します。

Karpenter を使用していない場合は、Descheduler を使用することをお勧めします。Karpenter と Cluster Autoscaler を一緒に使用している場合は、ノードグループに Descheduler と Cluster Autoscaler を使用できます。

グループレスノードのデスケジューラ

ポッドが削除されてもトポロジの制約が満たされ続ける保証はありません。たとえば、デプロイをスケールダウンすると、ポッド分散が不均衡になる可能性があります。ただし、Kubernetes はスケジューリング段階でのみポッドトポロジの分散制約を使用するため、ポッドは障害ドメイン間で不均衡なままになります。

このようなシナリオでバランスの取れたポッドディストリビューションを維持するには、Descheduler for Kubernetes を使用できます。デスケジューラは、ポッドの最大有効期間や有効

期限 (TTL) の適用、インフラストラクチャの使用の改善など、複数の目的に便利なツールです。レジリエンスと高可用性 (HA) の観点から、次のデスケジューラ戦略を検討してください。

- RemovePodsViolatingTopologySpreadConstraint
- RemovePodsViolatingInterPodAntiAffinity
- RemoveDuplicates

Karpenter 統合とビンパッキング機能

Karpenter を使用するワークロードでは、統合機能とビンパッキング機能を使用してリソース使用率を最適化し、Kubernetes クラスターのコストを削減できます。Karpenter はポッドの配置とノード使用率を継続的に評価し、可能な場合はワークロードをより少ない、より効率的にパックされたノードに統合しようとします。このプロセスでは、リソース要件を分析し、ポッドアフィニティルールなどの制約を考慮し、ノード間でポッドを移動して全体的なクラスター効率を向上させる可能性があります。次のコードは例を示しています。

```
apiVersion: karpenter.sh/v1beta1
kind: NodePool
metadata:
  name: default
spec:
  disruption:
    consolidationPolicy: WhenUnderutilized
    expireAfter: 720h
```

では`consolidationPolicy`、 `WhenUnderutilized`または を使用できます`WhenEmpty`。

- `consolidationPolicy` が に設定されている場合`WhenUnderutilized`、Karpenter はすべてのノードを統合と見なします。Karpenter が空または十分に使用されていないノードを検出すると、Karpenter はコストを削減するためにノードの削除または交換を試みます。
- `consolidationPolicy` が に設定されている場合`WhenEmpty`、Karpenter はワークロードポッドを含まないノードのみを統合対象と見なします。

Karpenter の統合の決定は、モニタリングツールに表示される CPU またはメモリ使用率のみに基づくものではありません。代わりに、Karpenter はポッドリソースリクエストと潜在的なコスト最適化に基づいて、より複雑なアルゴリズムを使用します。詳細については、Karpenter のドキュメントを参照してください。

PDB を使用して重要なワークロードを保護する

ポッド中断予算 (PDB) は、クラスター内のアプリケーションの高可用性を維持するために不可欠な機能です。PDB は、特定のタイプのポッドの最小可用性であるターゲットサイズを指定します。つまり、特定のポッドタイプのレプリカの最小数が常に実行されている必要があります。実行中のレプリカの数がターゲットサイズを下回ると、Kubernetes はターゲットサイズに達するまで残りのレプリカのさらなる中断を防ぎます。PDBsは、ワークロードがこれらのイベントの影響を受けないようにし、中断なく実行し続けるのに役立ちます。中断が発生すると、Kubernetes は PDB で指定されたレプリカの数を維持しながら、影響を受けるノードからポッドを適切に削除しようとします。

PDB を使用して、 `minAvailable`とレプリカ`maxUnavailable`の数を宣言できます。たとえば、アプリのコピーを 3 つ以上使用できるようにする場合は、次の例のような PDB を作成します。

```
apiVersion: policy/v1beta1
kind: PodDisruptionBudget
metadata:
  name: my-svc-pdb
spec:
  minAvailable: 3
  selector:
    matchLabels:
      app: my-svc
```

アプリケーションに PDBs を正しく設定することで、計画的または計画外のイベント中の中断を最小限に抑えることができます。アンチアフィニティルールを使用して、異なるノードでデプロイのポッドをスケジュールし、ノードのアップグレード中の PDB 遅延を回避できます。

プローブとロードバランサーのヘルスチェックを設定する

Kubernetes には、ロードバランサーのヘルスチェックに加えて、アプリケーションのヘルスチェックを実行する方法がいくつか用意されています。次の Kubernetes 組み込みプローブをロードバランサーのヘルスチェックと一緒にポッドのコンテキストのコマンドとして、または kubelet またはホスト IP アドレスへの HTTP/TCP プローブとして実行できます。

ライブネスプローブと準備状況プローブは、異なる独立した (または少なくとも異なるタイムアウト値を持つ) 必要があります。アプリケーションに一時的な問題がある場合、準備状況プローブは問題が解決されるまでポッドを準備中としてマークします。ライブネスプローブの設定が正しくない場合、ライブネスプローブはポッドを終了する可能性があります。

起動プローブ

スタートアッププローブを使用して、初期化サイクルが長いアプリケーションを保護します。起動プローブが成功するまで、他のプローブは無効になります。

Kubernetes がアプリケーションの起動を待機する最大時間を定義できます。最大設定時間が経過してもポッドが起動プローブに失敗した場合、アプリケーションは終了し、新しいポッドが作成されます。

アプリケーションの起動時間が予測不可能な場合は、起動プローブを使用します。アプリケーションの起動に 10 秒かかることがわかっている場合は、`initialDelaySeconds`代わりに でライブネスプローブまたは準備状況プローブを使用します。

ライブネスプローブ

ライブネスプローブを使用して、アプリケーションの問題や、プロセスが問題なく実行されているかどうかを検出します。ライブネスプローブは、プロセスが実行され続けるがアプリケーションが応答しなくなるデッドロック状態を検出できます。ライブネスプローブを使用する場合は、次の操作を行います。

- を使用して`initialDelaySeconds`最初のプローブを遅延させます。
- ライブネスプローブと準備状況プローブに同じ仕様を設定しないでください。
- ポッドの外部にある要素 (データベースなど) に依存するようにライブネスプローブを設定しないでください。
- 特定の のライブネスプローブを設定します`terminationGracePeriodSeconds`。詳細については[Kubernetes ドキュメント]を参照してください。

準備状況プローブ

準備状況プローブを使用して以下を検出します。

- アプリケーションがトラフィックを受け入れる準備ができているかどうか
- 部分的な可用性。アプリケーションは一時的に使用できない場合がありますが、特定のオペレーションの完了後に再び正常になると予想されます。

準備状況プローブは、アプリケーションがトラフィックを処理できるように、アプリケーション設定と依存関係が問題やエラーなしで実行されていることを確認するのに役立ちます。ただし、準備

状況プローブの設定が不十分な場合、それを防ぐのではなく停止が発生する可能性があります。データベースへの接続などの外部要因に依存する準備プローブは、すべてのポッドがプローブに失敗する可能性があります。このような障害が発生すると停止し、バックエンドサービスから障害が発生したポッドを使用した他のサービスへのカスケード障害が発生する可能性があります。

Ingress リソースとロードバランサーのヘルスチェック

Application Load Balancer と Kubernetes `ingress`にはヘルスチェック機能があります。Application Load Balancer のヘルスチェックでは、ターゲットポートとパスを指定します。

> **Note**
>
> Kubernetes の場合`ingress`、登録解除のレイテンシーが発生します。Application Load Balancer のデフォルトは 300 秒です。準備状況プローブに使用したのと同じ値を使用して、イングレスリソースまたはロードバランサーのヘルスチェックを設定することを検討してください。

NGINX はヘルスチェックも提供します。詳細については、NGINX ドキュメントを参照してください。

Istio イングレスゲートウェイとエグレスゲートウェイには、NGINX からの HTTP ヘルスチェックと同等のヘルスチェックメカニズムはありません。ただし、Istio サーキットブレーカーまたは`DestinationRule`外れ値検出を使用して、同様の機能を実現できます。

詳細については、「Amazon EKS ベストプラクティスガイド」の「可用性とポッドのライフサイクル」を参照してください。

コンテナライフサイクルフックを設定する

正常なコンテナのシャットダウン中、クライアントがダウンタイムを経験しないように、アプリケーションはシャットダウンを開始して`SIGTERM`シグナルに応答する必要があります。アプリケーションは、次のようなクリーンアップ手順を実行する必要があります。

- データの保存
- ファイル記述子を閉じる
- データベース接続を閉じる
- 処理中のリクエストを正常に完了する

- ポッド終了リクエストを満たすためにタイムリーに終了する

クリーンアップが終了するのに十分な猶予期間を設定します。SIGTERM シグナルに応答する方法については、アプリケーションで使用するプログラミング言語のドキュメントを参照してください。

コンテナライフサイクルフックを使用すると、コンテナは管理ライフサイクルのイベントを認識できます。コンテナは、対応するライフサイクルフックが実行されたときにハンドラーに実装されたコードを実行できます。コンテナライフサイクルフックは、Kubernetes とクラウドの非同期的な性質の回避策を提供します。このアプローチにより、進入リソースの前に終了ポッドに転送され、ポッドに新しいトラフィックを送信しないように更新iptablesされる接続が失われるのを防ぐことができます。

コンテナライフサイクル、Endpoint、 EndpointSliceは、さまざまな APIs。これらの APIs をオーケストレーションすることが重要です。ただし、ポッドが終了すると、Kubernetes API は kubelet (コンテナライフサイクル用) とEndpointSliceコントローラーの両方に同時に通知します。図を含む詳細については、「Amazon EKS ベストプラクティスガイド」の「クライアントリクエストを適切に処理する」を参照してください。

kubelet がポッドSIGTERMに送信すると、EndpointSliceコントローラーはEndpointSliceオブジェクトを終了しています。この終了は、Kubernetes API サーバーに通知して、 を更新する各ノードkube-proxyの に通知しますiptables。これらのアクションは同時に行われますが、それらのアクション間に依存関係やシーケンスはありません。kube-proxy 各ノードの がローカルiptablesルールを更新するよりも、コンテナがSIGKILLシグナルを受信する可能性が高くなります。その場合は、次のようなシナリオが考えられます。

- アプリケーションが の受信時に処理中のリクエストと接続をすぐに急に切断するとSIGTERM、クライアントに500エラーが表示されます。
- アプリケーションが の受信時にすべての処理中のリクエストと接続が完全に処理されるようにした場合SIGTERM、猶予期間中、iptablesルールがまだ更新されていない可能性があるため、新しいクライアントリクエストは引き続きアプリケーションコンテナに送信されます。クリーンアップ手順がコンテナのサーバーソケットを閉じるまで、これらの新しいリクエストは新しい接続になります。猶予期間が終了すると、 の送信後に確立された新しい接続SIGTERMは無条件に削除されます。

前述のシナリオに対処するには、アプリ内統合または PreStop ライフサイクルフックを実装できます。図を含む詳細については、「Amazon EKS ベストプラクティスガイド」の「アプリケーションを適切にシャットダウンする」を参照してください。

> **Note**
>
> アプリケーションが正常にシャットダウンされるかどうか、またはpreStopフックの結果にかかわらず、アプリケーションコンテナは最終的に を通じて猶予期間の終了時に終了しますSIGKILL。

コマンドで preStopフックsleepを使用して、 の送信を遅延させますSIGTERM。これにより、イングレスオブジェクトがポッドにルーティングしている間、新しい接続を引き続き受け入れることができます。次の例に示すように、 sleep コマンドの時間値をテストして、Kubernetes やその他のアプリケーションの依存関係のレイテンシーが考慮されていることを確認します。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  containers:
    - name: nginx
      lifecycle:
        # This "sleep" preStop hook delays the Pod shutdown until
        # after the Ingress Controller removes the matching Endpoint or EndpointSlice
        preStop:
          exec:
            command:
              - /bin/sleep
              - "20"
              # This period should be turned to Ingress/Service Mesh update latency
```

詳細については、Kubernetes ドキュメントの「コンテナフック」および「Amazon EKS ベストプラクティスガイド」の「アプリケーションを適切にシャットダウンする」を参照してください。

ゾーン中断中のポッドエビクションを理解する

完全なアベイラビリティーゾーンの中断が発生した場合、つまり、そのアベイラビリティーゾーン内のすべてのノードが Kubernetes コントロールプレーンへの接続を失った場合、Kubernetes のノードライフサイクルコントローラーは状況を検出し、影響を受けるゾーンからポッドを削除します。到達できないノードのポッドは とマークTerminatingされ、新しいポッドは利用可能なアベイラビリティーゾーンの正常なノードでスケジュールされます。この期間中、影響を受けるノード

はNotReadyステータスを表示し、スケジューラは新しいポッドがそれらのノードに配置されないようにし、EndpointSlice コントローラーは接続が復元されるまで、障害のあるアベイラビリティーゾーンに関連付けられたエンドポイントをサービスルーティングから削除します。

ゾーン内で部分的なノード障害が発生し、ノードのサブセットのみが到達不能になるシナリオの場合、ノードライフサイクルコントローラーは異なるエビクション動作を適用します。中断が設定された許容期間 (デフォルトでは 5 分) を超えて持続する場合、切断されたノードのポッドは としてマークTerminatingされ、新しいポッドは使用可能なアベイラビリティーゾーンの正常なノードでスケジュールされます。

Amazon EKS ゾーンシフトを実装して耐障害性を向上させる

Amazon Application Recovery Controller (ARC) と統合される Amazon EKS ゾーンシフトは、アベイラビリティーゾーンの障害発生時にトラフィックをプロアクティブに管理するメカニズムを提供します。この機能を使用すると、異常なアベイラビリティーゾーンから同じゾーン内の正常なゾーンにネットワークトラフィックを一時的にリダイレクト AWS リージョン して、サービスの中断を最小限に抑えることができます。

ゾーンシフトメカニズムについて

Amazon EKS ゾーンシフトは、東西トラフィック (クラスター内のポッド間通信) に対処します。ゾーンシフトが Application Load Balancer または Network Load Balancer で設定されている場合、イングレストラフィックルーティングもサポートされます。このメカニズムは、複数の Kubernetes コンポーネントと AWS コントロールプレーンコンポーネントを調整して、実行中のワークロードを中断することなくトラフィックを安全にリダイレクトすることで動作します。アクティブなゾーンシフト中、Amazon EKS は次の調整されたアクションを自動的に実行します。

- ノードのグルーニング: 障害が発生したアベイラビリティーゾーン内のすべてのノードがグルーニングされます。これにより、Kubernetes スケジューラは既存のワークロードを維持している間、ノードに新しいポッドを配置できなくなります。
- アベイラビリティーゾーンの再調整の停止: マネージド型ノードグループの場合、アベイラビリティーゾーンの再調整オペレーションは中断され、Auto Scaling グループは正常なアベイラビリティーゾーンでのみ新しいデータプレーンノードを起動するように更新されます。これにより、新しい容量が障害ゾーンにプロビジョニングされなくなります。
- エンドポイントの削除: EndpointSlice コントローラーは、障害が発生したアベイラビリティーゾーンのポッドエンドポイントを関連するすべての EndpointSlices から削除します。これにより、サービス検出とロードバランシングのメカニズムは、正常なアベイラビリティーゾーンで実行されているポッドにのみトラフィックをルーティングします。

- ワークロードの保存: Amazon EKS は、影響を受けるアベイラビリティーゾーンでのノードの終了やポッドの削除を控えます。ゾーンシフトの有効期限が切れたりキャンセルされたりすると、追加のスケーリング操作を必要とせずにトラフィックを安全に戻ることができるように、障害ゾーンでフルキャパシティを維持します。

ゾーンシフトのアクティベーション方法

運用モデルに応じて、2 つのアプローチから選択してゾーンシフトを開始できます。

- 手動ゾーンシフトは、モニタリング、アラート、または顧客レポートによって特定のアベイラビリティーゾーンの問題が検出された場合に、オペレーター主導の制御を提供します。このメソッドには、ARC コンソール、 AWS Command Line Interface (AWS CLI)、またはゾーンシフト APIs を介した明示的なアクションが必要です。ここで、オペレーターは障害のあるアベイラビリティーゾーンを指定し、シフトの有効期限を定義します。手動シフトは、チームが専用のモニタリング機能とオンコール機能を持ち、トラフィック管理の決定を直接制御したい場合に適しています。
- ゾーンオートシフトは AWS 、ARC がネットワークメトリクス、Amazon Elastic Compute Cloud (Amazon EC2) AWS のサービス、Elastic Load Balancing など、複数の にわたる内部テレメトリとヘルスシグナルに基づいて潜在的なアベイラビリティーゾーンの障害または障害を検出したときに、 が自動的にシフトを開始することを に許可します。 は、インジケータが問題を解決したことを示すと、オートシフト AWS を自動的に終了します。手動による介入を最小限に抑えて可用性を最大限に高めたい場合は、検出されたアベイラビリティーゾーンの障害に数分で対応できるため、このアプローチをお勧めします。

効果的なゾーンシフトの前提条件

アベイラビリティーゾーンの障害発生時にゾーンシフトがアプリケーションを適切に保護するには、ゾーンシフト機能を有効にする前に、マルチ AZ レジリエンスのためにクラスターを設計する必要があります。

- マルチ AZ ノード分散: 少なくとも 3 つのアベイラビリティーゾーンにワーカーノードをプロビジョニングして、1 つのゾーンが使用できなくなったときに十分な冗長性を確保します。
- キャパシティプランニング: 1 つのアベイラビリティーゾーンがサービスから削除されたときにワークロード全体に対応するために、正常なアベイラビリティーゾーン間で十分なコンピューティングキャパシティを事前プロビジョニングします。これは、アクティブな中断中のスケーリングオペレーションで容量不足が発生する可能性があるためです。

- ポッドの分散と事前スケーリング: すべてのアベイラビリティーゾーンに各アプリケーションの複数のレプリカをデプロイし、すべてのゾーンで CoreDNS などの重要なシステムコンポーネントを事前スケーリングします。これにより、ゾーンを遠ざけた後も十分な容量が維持されます。

ゾーン中断の回復力に関する推奨事項

- クラスターの作成時にゾーンシフトを有効にする: 新しい EKS クラスターの場合、Amazon EKS コンソール、 または などのInfrastructure as Code (IaC) ツールによる初期プロビジョニング中に AWS CLI、ARC とのゾーンシフト統合を有効にします AWS CloudFormation。クイック設定で作成された EKS Auto Mode クラスターでは、デフォルトでゾーンシフトが有効になっています。
- 適切なアクティベーション方法を選択する: 自動応答で最大限の可用性を必要とする本番環境、特にアベイラビリティーゾーンの障害中に数分のダウンタイムがビジネスに大きな影響を与える可能性がある顧客向けアプリケーションには、ゾーンオートシフトを選択します。運用チームがトラフィックのシフト前に明示的な承認を行うことを希望する環境や、アプリケーションのテストと検証がまだ進行中の環境では、手動ゾーンシフトを使用します。
- 本番稼働用デプロイ前のテストレジリエンス: テストゾーンシフトを手動で開始するか、ゾーンオートシフトの練習実行を有効にして、シングル AZ 損失下でクラスターの動作を検証し、アベイラビリティーゾーン数を減らして運用する場合にアプリケーションが可用性を維持し、パフォーマンスが許容範囲内であり、容量が十分であることを確認します。実際のアベイラビリティーゾーンの障害が発生する前に設定ギャップを特定できるように、このテストを強くお勧めします。
- ロードバランサーの設定と調整する: 外部トラフィックを受信するアプリケーションの場合、関連する Application Load Balancer と Network Load Balancer で ARC ゾーンシフトを有効にして、アベイラビリティーゾーンの障害中に進入トラフィックとクラスター内の東西トラフィックの両方が一緒にシフトするようにします。この調整により、外部リクエストが正常なポッドに到達しても、それらのポッドがシフトアウェイゾーンの依存関係と通信できないシナリオを防ぐことができます。
- シフトオペレーションのモニタリング: ゾーンシフトを有効にしたら、オートシフトのアクティベーション、手動シフトの開始、シフトの有効期限などのシフトイベントのモニタリングとアラートを設定して、トラフィック管理アクションとそのアプリケーション動作への影響に関する運用上の可視性を維持します。

シフトの完了と復旧

ゾーンシフトが設定された期間に基づいて期限切れになるか、アベイラビリティーゾーンの障害が解決した後に手動でキャンセルされると、EndpointSlice コントローラーはすべての EndpointSlices を

自動的に更新して、復元されたアベイラビリティーゾーンにエンドポイントを再統合します。トラフィックは、クライアントがエンドポイント情報を更新して新しい接続を確立すると、以前に影響を受けたゾーンに徐々に戻ります。これにより、手動による介入やポッドの再スケジュールを必要とせずに、クラスターの容量を最大限に活用できます。

結論

アプリケーションの可用性と回復性を高めるためにアーキテクチャを設計する場合は、次のコンポーネントを検討してください。

- マイクロサービスアプリケーション (ポッドとコンテナ)
- ワークロードデータプレーン (Ingress Controller、ポッド、Amazon VPC CNI などのシステムコンポーネント、サービスメッシュサイドカー、kube-proxy)
- ワークロード管理レイヤー (コントローラー、アドミッションコントローラー、ネットワークポリシーエンジン、およびこれらのコンポーネントの永続データストレージ)
- Kubernetes コントロールプレーン
- インフラストラクチャ (ノード、ネットワーク、ネットワークアプライアンス)

これらのコンポーネントに関する考慮事項に対処するには、次の主要な戦略を使用します。

- 高可用性と耐障害性を確保するには、ノードとアベイラビリティーゾーンにワークロードを分散します。
- 重要なワークロードを保護するには、ポッド中断予算 (PDBs) を使用して、中断中のアプリケーションの安定性を維持します。
- ポッドが正常に実行され、トラフィックが正しく処理されるようにするには、スタートアッププローブ、ライブネスプローブ、準備状況プローブ、ロードバランサーのヘルスチェックを設定します。
- コンテナの状態遷移を効率的に管理するには、コンテナライフサイクルフックを設定します。
- ノードの障害またはメンテナンス中のエビクションプロセスを制御するには、ポッドのエビクション時間を設定します。

これらのプラクティスを実装することで、Amazon EKS で実行されているアプリケーションの信頼性と耐障害性を大幅に強化し、堅牢なパフォーマンスと高可用性を確保できます。

リソース

- Kubernetes Pod トポロジのスプレッド制約 (Kubernetes ドキュメント)
- Karpenter FAQs (Karpenter ドキュメント)
- Descheduler for Kubernetes (GitHub リポジトリ)
- 可用性とポッドライフサイクルの Amazon EKS ベストプラクティスガイド
- アプリケーションを適切にシャットダウンする Amazon EKS ベストプラクティスガイド
- [EKS] [request]: pod-eviction-timeoutと回避策を設定する機能 (コンテナロードマップリポジトリ)

ドキュメント履歴

以下の表は、本ガイドの重要な変更点について説明したものです。今後の更新に関する通知を受け取る場合は、RSS フィード をサブスクライブできます。

変更	説明	日付
更新	ゾーン中断中のポッドエビクションに関するセクションを改訂しました。	2025 年 10 月 29 日
更新	ポッドトポロジースプレッド制約の使用セクションを改訂しました。	2025 年 1 月 27 日
初版発行	—	2024 年 10 月 23 日

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。