Engineer

高可用性

一部が故障してもサービスを動かし続ける能力

どのようなものか

高可用性は、背後のマシン、ディスク、ネットワーク経路、プロセスの一つが故障しても、サービスが応答し続ける性質です。普通はクラスターを作ります。それぞれがサービスを動かせる複数のホスト、監視するソフトウェア、故障したホストから正常なものへ仕事を移す手順を組み合わせます。オープンソースのクラスター基盤PacemakerとCorosyncを保守するClusterLabsは、複数ホストのサービスを、起動、正常性確認から故障後の復旧まで管理し、「停止時間を最小限にしながら、データの整合性を保つ」ものと説明します。

この文の後半も、前半と同じくらい重要です。誤ったデータですぐ戻るサービスは、高可用になったのではなく、危険になっただけです。

PostgreSQLの文書は、データベースを複数台で動かす理由を分けます。主系の故障時に別のサーバーがすぐ引き継ぐのが、高可用性です。複数のコンピューターから同じデータを提供するのが、負荷分散です。仕組みを共有することは多くても、問いは違います。高可用性は故障を乗り切るため、負荷分散と拡張性は、より多くの仕事を処理するためです。

可用性自体は数値です。従来は、期間のうちサービスを使えた時間の割合で測ります。GoogleのSRE書籍は、世界に分散して、ほぼ常にどこかが動くサービスでは、代わりに、正常に答えたリクエストの割合で測ると説明します。どちらも百分率で書き、9の数で呼びます。

どの高可用性の設計も、同じ少数の部品で組み立てます。

- **冗長化。** 故障するとサービスが止まる部分ごとに、二台目のホスト、待機データベース、第二のネットワーク経路などの予備を置きます。Pacemakerは、稼働系と待機系の組、複数の稼働ノードが予備一つを共有するN+1などに対応します。

- **検知。** 故障を何かが見つける必要があります。PostgreSQLのフェイルオーバーの章は、主系と待機系の間のハートビートで、接続と、主系が仕事を続けているかを常に確認すると説明します。
- **フェイルオーバー。** 仕事を移すことです。待機系の昇格、サービスIPアドレスの移動、別のホストでのプロセス起動などです。
- **フェンシング。** 別の場所で仕事を始める前に、故障ノードが本当に止まったことを確実にします。
- **予備の再構築。** 切り替え後は、新しい予備を用意するまで、冗長性がありません。

高可用性は、バックアップでも災害復旧でもありません。拠点全体が動いている間に起きる、一部の故障に備えます。バックアップは、クラスターが忠実に全ノードへ複製する破損も含め、データの喪失や破損に備えます。災害復旧は、拠点全体を失うときの計画です。

概念を判断に生かす

「高可用性とは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 障害が小さいうちに知らせる監視とアラートを整え、安定した状態をどれだけ維持できているかを数値で示します。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 社内初のオブザーバビリティダッシュボードを構築 — 社内システムの状態を初めて可視化

社内初のオブザーバビリティダッシュボードを構築 お問い合わせ →

なぜ重要なのか

どのサービスにも、故障する部分があります。ハードウェアの破損、カーネルパニック、仮想マシンの誤削除、ネットワーク分断による正常なノードの孤立などです。Kubernetesは、これを非自発的な中断とし、修理や更新のためノードを空けるなど、誰かが計画する自発的な中断と並べます。冗長性がなければ、予定したものも含め、すべて停止になります。一台のサーバーへパッチを当てるには、止めるしかないためです。

計画保守に対応できる点が、チームの最初に気付く利点であることもよくあります。PostgreSQLの章は、主系と待機系を定期的に切り替えると有用とします。順番に止めて保守でき、毎回、必要なときに切り替わるかも試せます。

9の数は時間へ換算できます。一年を365日、525,600分、一か月を30日、43,200分とすると、計算は次のとおりです。

```
可用性      年間の許容停止時間      30日間の許容停止時間
99%         3.65日                  7.2時間
99.9%       8.76時間                43.2分
99.95%      4.38時間                21.6分
99.99%      52.56分                 4.32分
99.999%     約5.26分                約26秒
```

9を一つ増やすごとに許容時間は十分の一になり、費用はそれ以上に増えます。SRE書籍は、次の一段階の信頼性に、その前の百倍かかる場合があり、ある点を超えると、利用者には区別できないと警告します。自身の信頼性が99%の電話なら、背後が99.99%でも99.999%でも、違いは分かりません。そのため、目標を下限でも上限でもあるとします。必要を大きく超えた可用性は、ほかに回せた資金とエンジニアリングの時間を使っています。

目標は、サービスごとに行う事業上の決定です。サービスレベル目標が数値を記し、高可用性は、それを実現可能にするエンジニアリングです。

実際の活用例

架空の予約サービスを考えます。ウェブ、アプリケーション、PostgreSQLの三層を、別々のホストへ置きます。以下の数値は、すべて例のために作ったものです。各層が99.9%の時間動き、全部が必要なら、可用性を掛け合わせます。一層の二重化では、両方が同時に止まる必要があるため、逆の計算です。

```
各99.9%の可用性を持つ三層を直列に接続:
  0.999 × 0.999 × 0.999 ≈ 0.997       年間約26時間停止

二つのコピーが独立して故障する場合、一層を二重化:
  1 − 0.001 × 0.001 = 0.999999        年間約0.5分停止
```

二つ目の計算を支えるのは、「独立して」という一語です。同じホスト、ラック、スイッチ、クラウドゾーンの二つのコピーは、故障を共有するため、計算が成り立ちません。また式は、切り替えが瞬時で、常に成功するとも仮定します。確かめられるのはテストだけです。

Kubernetesでは、トポロジー分散制約が、そのためにあります。文書が導入の理由として挙げるのは、二つのPodまで減る可能性のあるワークロードです。同じノードへ置くと、一つの故障で全体が止まります。Podテンプレートの制約で、レプリカをゾーンへ分けます。

```
topologySpreadConstraints:
    - maxSkew: 1
      topologyKey: topology.kubernetes.io/zone
      whenUnsatisfiable: DoNotSchedule
      labelSelector:
          matchLabels:
              app: booking
```

`DoNotSchedule` なら、ゾーンごとのPod数の差が一つを超える場所へ置く代わりに、Podを未割り当てのままにします。計画保守には、第二の備えが必要です。PodDisruptionBudgetは、自発的な中断で同時に止めてよい、アプリのレプリカ数を制限します。 `kubectl drain` は、Eviction APIへ一つずつ要求するので守りますが、Podを直接削除すると、その制限を回避します。

```
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
    name: booking-pdb
spec:
    maxUnavailable: 1
    selector:
        matchLabels:
            app: booking
```

文書は、レプリカ数の変化へ自動的に追随する `maxUnavailable` を、 `minAvailable` より勧めます。

データベースは状態を持つので、最も難しい層です。PostgreSQLは、読み取り専用クエリを受けるホットスタンバイか、昇格後に初めて接続を受けるウォームスタンバイを保てます。同期と非同期の選択は、切り替えで何を失ってよいかの選択です。同期レプリケーションでは、設定した数の同期スタンバイから、設定した段階までの確認を待ってから、コミットの成功を応答します。この条件は `synchronous_standby_names` と `synchronous_commit` で定めます。フェイルオーバーでは、成功応答済みのデータを保持するスタンバイを選ぶ必要があり、非同期のスタンバイはまだ遅れている場合があります。前者の代償は速さです。文書は、ネットワークが遅いと、完全同期の構成で性能が半分を超えて低下し得ると記します。

PostgreSQL自身は、いつ切り替えるかを決めません。文書は、主系の故障を検出し、待機系へ知らせるシステムソフトウェアは提供しないとします。IPアドレスの移動とともに、外部のクラスター管理ソフトウェアの仕事です。昇格自体

は、`pg_ctl promote`か関数`pg_promote()`の一段階です。計画が忘れるのは、その後です。旧主系がなくなると、新しい待機系を構築するまで一台だけで動きます。文書は、これを縮退状態と呼びます。大きいクラスターでは、`pg_rewind`で再構築を速められます。

このサイトのフェイルオーバーと配信ログを備えるメールの事例は、外部依存先へ同じ仕組みを適用します。一つのインターフェースの後ろに三事業者を置き、切迫した中で設定を変える代わりに、自動で切り替えます。起動検査が実際のメッセージを送り、ステージングと本番は、正常に送信できる事業者がなければ起動を拒否します。事例は、ある午後に事業者が不調でも、停止ではなく機能の低下した経路で済むようになり、代償は三つの連携を維持することと記します。どの程度必要かは、別の決定です。サービスレベルは、夜間のレポートと支払いサービスでは必要な数値が違うため、システムごとに、契約へ可用性を記すと説明します。

問題が起きるところ

同じ故障を共有する冗長化。 同じホスト、電源、スイッチ、クラウドゾーンの二つのコピーは、一緒に故障します。予備ごとに依存先をたどり、偶然でなく、意図して障害領域を分けます。

スプリットブレイン。 互いに連絡が取れなくても動くノードは、相手を故障と判断し、両側でサービスを始め得ます。Pacemakerは、同じIPを両ホストで有効にすると、パケットの行き先が予測不能で、データベースではデータの破損や食い違いにつながると説明します。フェンシング、別名STONITH（Shoot The Other Node In The Head）が備えです。クラスターが制御する電源スイッチなどで、別の場所で仕事を始める前に、故障ノードを切り離します。文書は、隔離装置がなければ応答しないノードから安全に復旧できず、その装置はクラスター自体のネットワークにも依存してはいけないとします。そのネットワークを失うと、隔離手段も失うためです。

誰も試していない切り替え。 昇格したことのない待機系は、計画ではなく希望です。PostgreSQLの助言どおり、定期的に役割を切り替え、手順を書き、カオスエンジニアリングがほかの故障を試すように、予行演習します。

縮退状態を忘れます。 切り替え後は予備がありません。待機系の再構築が遅いか手作業なら、次のどんな小さい故障も、停止になります。冗長性を戻すまでをインシデントの一部とし、後回しにしません。

非同期で何も失わないと思います。 故障直前の、コミット済みトランザクションを失い得ます。データベースごとに許容できるかを決め、データ所有者にも交換条件を見せます。

複製をバックアップと思います。 待機系は、正しい書き込みと同じく、誤った `DELETE` も忠実に複製します。高可用性は動かし続けるもので、元のデータへ戻せるのは、バックアップだけです。

二つのノードだけで、仲裁役がありません。 Pacemakerは、停電でノードを失うのと、通信経路を失うのは、同じに見えると記します。二台なら、両側がそう見ます。Kubernetesは、スタック型の構成で、少なくとも三つの制御プレーンノードを求めます。PostgreSQLも、一部の誤った切り替えを防ぐ第三の監視サーバーに触れますが、増える複雑さに見合うのは、注意深い設定と厳格なテストがある場合だけと警告します。

誰にも見えない9へ支払います。 追加する9は、前より費用がかかり、利用者のネットワークや端末も、目標より信頼性が低いものです。利用者と事業の必要から数値を選び、それに合わせて構築します。

参考資料

- ClusterLabs — オープンソースの高可用性クラスター用ソフトウェア基盤と、管理するもの、フェンシングが必要な理由の定義です。
- 高可用性、負荷分散、複製 — 主系と待機系、同期と非同期の設計、費用についてのPostgreSQLの章です。
- フェイルオーバー — ハートビート、監視サーバー、STONITH、昇格、縮退状態の説明です。
- フェンシング — Pacemaker Explainedによる、スプリットブレイン、隔離装置、フェンシングとクォーラムです。
- リスクを受け入れる — 可用性の測り方、9ごとの費用、目標の選び方についてのGoogle SRE書籍です。
- 中断 — Kubernetesの自発的、非自発的中断と、PodDisruptionBudgetによる制限です。

用語集

実際の制約に合わせて選ぶ

デプロイの流れ、運用上のリスク、緊急対応の担当者をお知らせください。検査、復旧手段、明確な責任の所在を伴う変更の範囲を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「高可用性とは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/high-availability/