LinuC

LinuC-2

LinuC レベル2 Version 10.0
技術解説セミナー

主題2.13 システムアーキテクチャ （前半）

- 2.13.1 高可用システムの実現方式
- 2.13.2 キャパシティプランニングとスケーラビリティの確保
- 2.13.3 クラウドサービス上のシステム構成

2021年12月19日

濵野 賢一朗

LPI-JAPAN

濵野 賢一朗（はまの けんいちろう）

- システム開発会社で、オープンソースソフトウェアや
  基盤系技術を中心とした技術支援や技術開発に従事
  - 並列分散処理（HadoopやSpark等）、データベースミドルウェア、
    JVM/JDK、クラウド基盤、量子コンピューティング、など・・・
  - 先進的なお客様のCTO/CIOとの 技術相談としてのカウンターパート役

- 日本Sambaユーザー会、日本Hadoopユーザー会などの
  オープンソースソフトウェアのユーザーコミュニティ・開発コミュニティの発足・運営にも関わる
- 「日本OSS貢献者賞・奨励賞」実行委員長も務めていた
- 書籍『Linux教科書 LinuC レベル1・2』
  『オープンソースソフトウェアの本当の使い方』
  『Hadoop徹底入門』など監修・執筆

今回のテーマ

主題2.13 システムアーキテクチャ （前半）

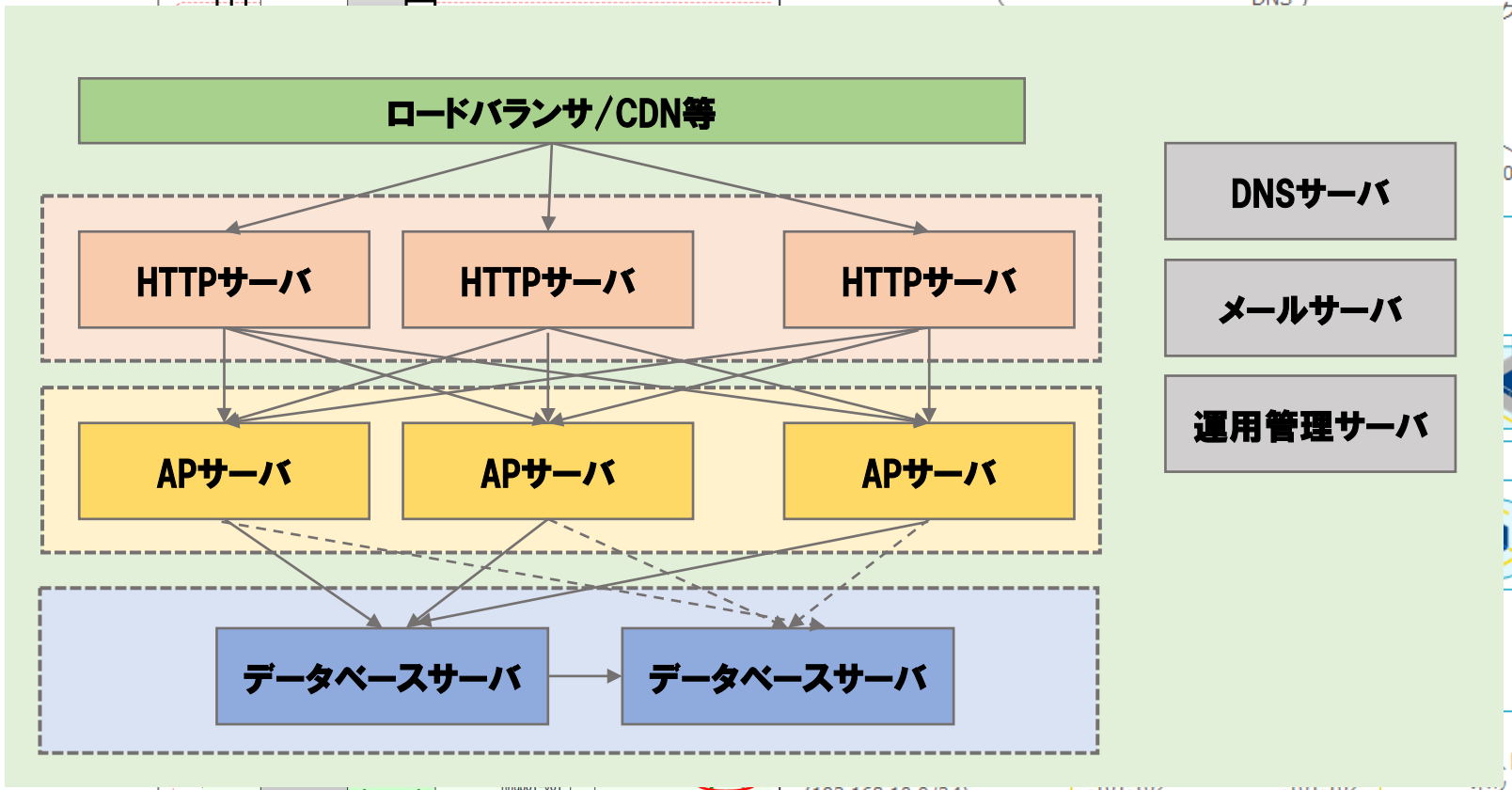
・2.13.1 高可用システムの実現方式
・2.13.2 キャパシティプランニングとスケーラビリティの確保
・2.13.3 クラウドサービス上のシステム構成

- **ここまで、WebサーバやDNSサーバなどサーバ技術について学んできたと思うが・・・**
- **現実のITシステムは、単一のサーバで実現されているわけではない**

- **主題2.13では、実際のITシステムを構成できる基本的な考え方を問われる**
  - **複数のサーバ、ネットワーク機器、サービスなどの組合せをどう組み合わせるか**
  - **機能要件・非機能要件をどう表現するか、満たすための方式 など**

現実のシステム構成

■ システム構成例



NTTコミュニケーションズ Enterprise Cloud QuickStartGuideより抜粋
https://ecl.ntt.com/documents/quick-start-guide/rsts/pattern_4/model_c_cent_configuration.html

NTTデータ 対外発表資料より引用

機能要件・非機能要件

- ITシステムの要件（要求）は、機能と非機能に2側面がある
  - 機能要件：　　機能や挙動
  - 非機能要件：　機能以外に具備しておくべきもの

- 非機能要件は、IPA「非機能要求グレード」では大きく6つに分類
  - 可用性
  - 性能・拡張性
  - 運用・保守性
  - 移行性
  - セキュリティ
  - システム環境・エコロジー

・結果が表示されるまで、どれくらいの時間が許容できるか？
（レイテンシ）
・同時にどの程度のアクセス数に耐えられる必要があるか？
（同時接続数、スループット）
・ハードウェア障害が発生した場合にサービス継続ができるか？
：

※ IPA（独立行政法人 情報処理推進機構）非機能要求グレード 2018
https://www.ipa.go.jp/sec/softwareengineering/std/ent03-b.html

- システムアーキテクチャは、機能要件・非機能要件の両面を実現できるように検討する必要がある

主題2.13 システムアーキテクチャ （前半）

- **2.13.1 高可用システムの実現方式**
- 2.13.2 キャパシティプランニングとスケーラビリティの確保
- 2.13.3 クラウドサービス上のシステム構成

可用性と影響のある事象

■ 可用性（Availability）

- システムを継続して稼働できる能力
- サービスの提供が不可能になる状態に陥らずに安定して利用できると「可用性が高い」

■ ITシステムのサービス提供が継続しない要因の例

要因の類型		例
意図的な要因	内部	メンテナンス停止（計画的、緊急的）等
	外部	サイバー攻撃（不正侵入、データ改竄・破壊）、サービス不能攻撃（DoS）等
非意図的な要因	偶発的な要因	機器故障、人為的ミス（操作ミス・設定ミス）、ソフトウェアの欠陥（バグ）等
	環境的な要因	地震、水害、落雷、火災による設備破壊 等
	他障害からの波及	電源供給や通信の途絶 等

- 物理障害・ハード障害？
  論理障害・ソフト障害？
- 現実には、故障と判別するのは難易度が高い
  （動いたり、動かなかったり）

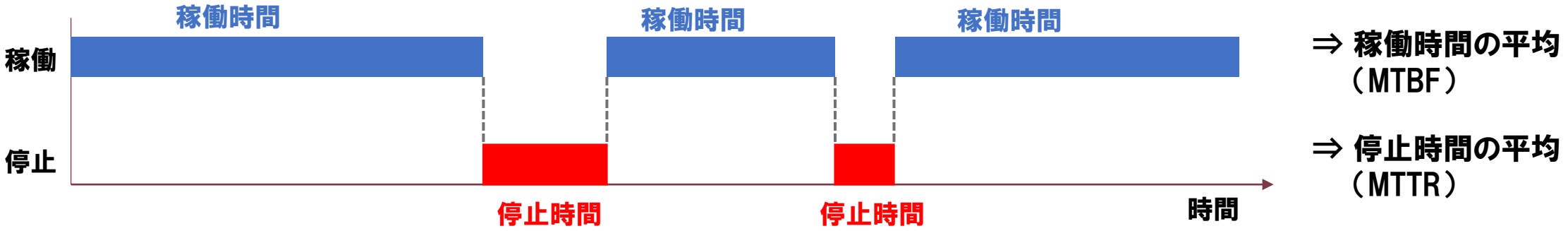
可用性の評価

- 求める可用性によって、システム構成は変わる
  - 妥当な可用性を目指す必要がある
  - 一般的に可用性を高めるには相応のコストを要する

可用性の指標：MTBFとMTTR

- **MTBF** = Mean Time Between Failure
  （平均故障間隔・平均連続稼働時間）
  - 故障と故障の間で、正常に稼働している平均時間

- **MTTR** = Mean Time To Repair
  （平均修理時間・平均停止時間）
  - システムを修理するのに要する平均時間



⇒ 稼働時間の平均
（MTBF）

⇒ 停止時間の平均
（MTTR）

可用性の指標：稼働率

■ 稼働率

- システムが正常に稼働している割合

$$\text{稼働率} = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}}$$

■ 稼働率計算の例

稼働率	1年間の停止時間
99%	87.6時間
99.5%	73.65時間
99.9%	約44分
99.99%	約4分
99.999%	約26秒

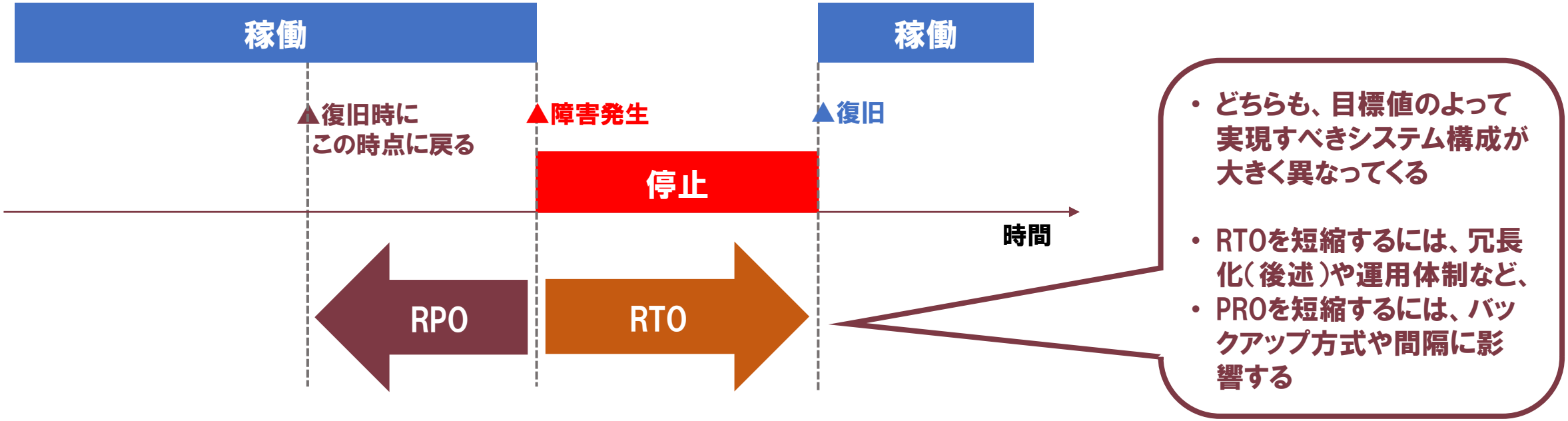
- たとえば、A社クラウドの仮想マシン提供のインスタンスは、稼働率99.5%で、それをを下回った場合、クレジットが返還されるとされている
- 実際に99.5%の稼働が約束されているわけではない

■ SLA = Service Level Agreement

- サービスを提供する事業者などが、サービス範囲と品質を示すために提示する
- 稼働率などで記載されることも多い

可用性の指標：RTOとRPO

- RTO = Recovery Time Objective（目標復旧時間）
  - 障害が発生してから、どのくらいの時間で復旧させるかの指標
- RPO = Recovery Point Objective（目標復旧時点）
  - 復旧時に、障害発生以前のどの段階までデータ復旧できるかの指標



高可用性システムの実現

- 高可用性（High Availability）システム
  - システム障害が発生しにくいシステム
  - システムの一部で故障等が発生しても、システム全体が停止することなくサービス提供し続けられる

- 基本的には、同じ機能や役割に要素をあらかじめ複数用意しておき、異常が発生した場合に肩代わりできる仕組み（冗長化）による実現させる

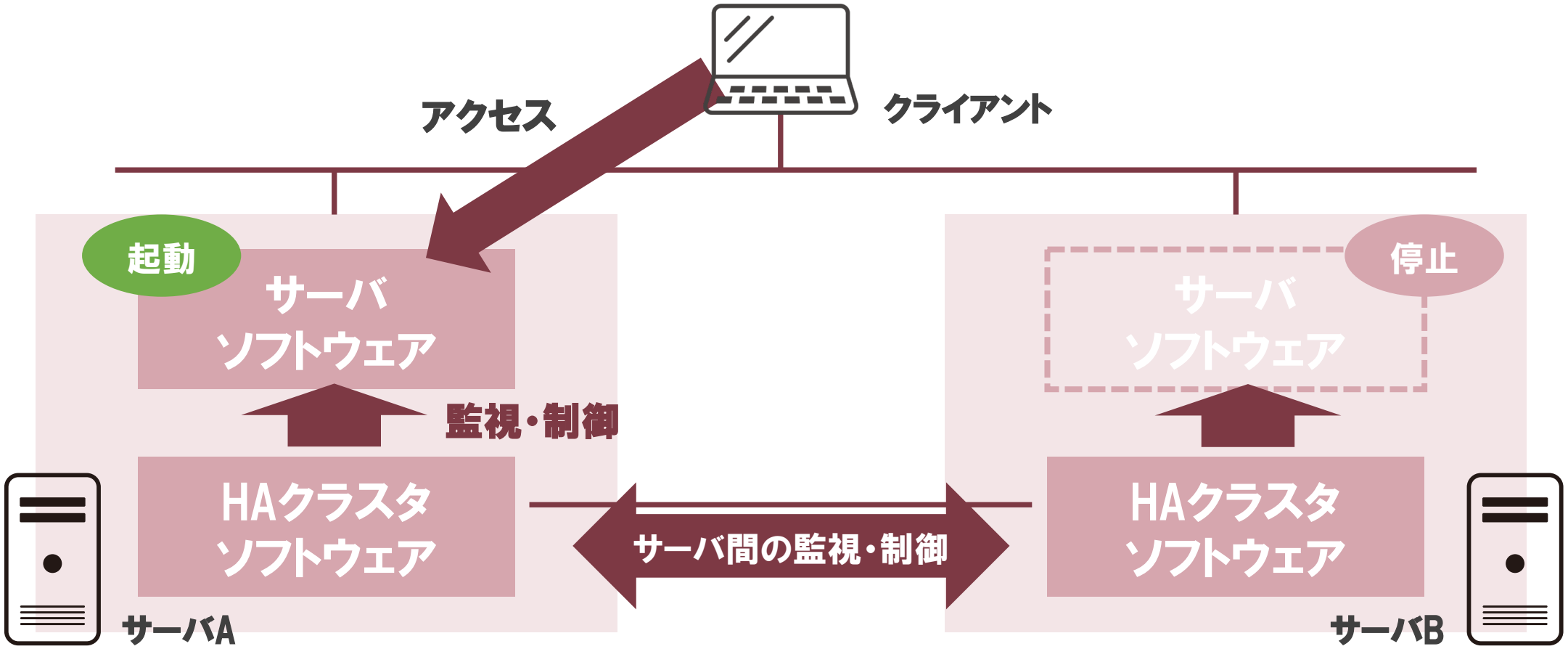
  - 冗長化すると、構成する機器数も増えるため、故障頻度も増す課題もある

- システムにおける SPoF を排除するようにシステムを設計
  - SPoF = Single Point of Failure（単一障害点）

高可用性システムの実現方式 – HAクラスタ

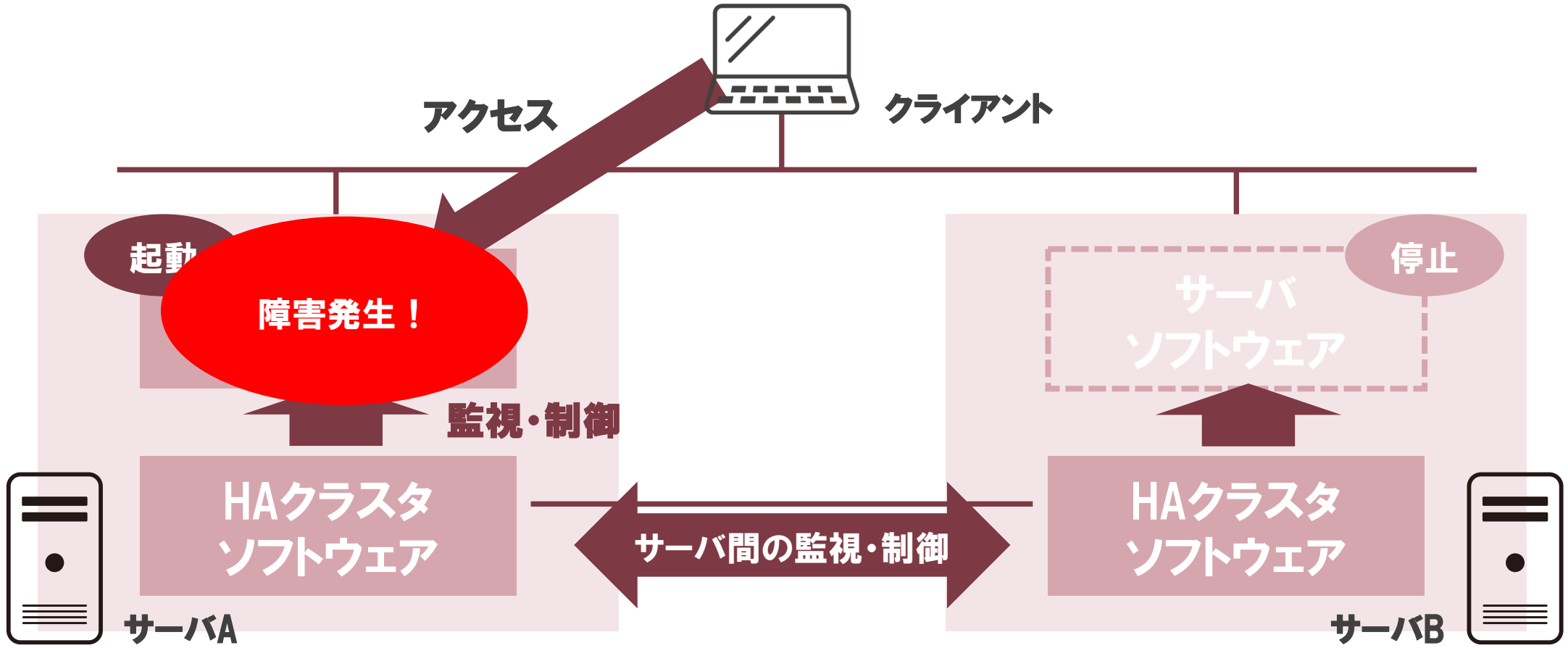
■ HAクラスタソフトウェア

- 複数のサーバで監視して、必要に応じて、サービスを起動・停止等することで、サービス継続を実現



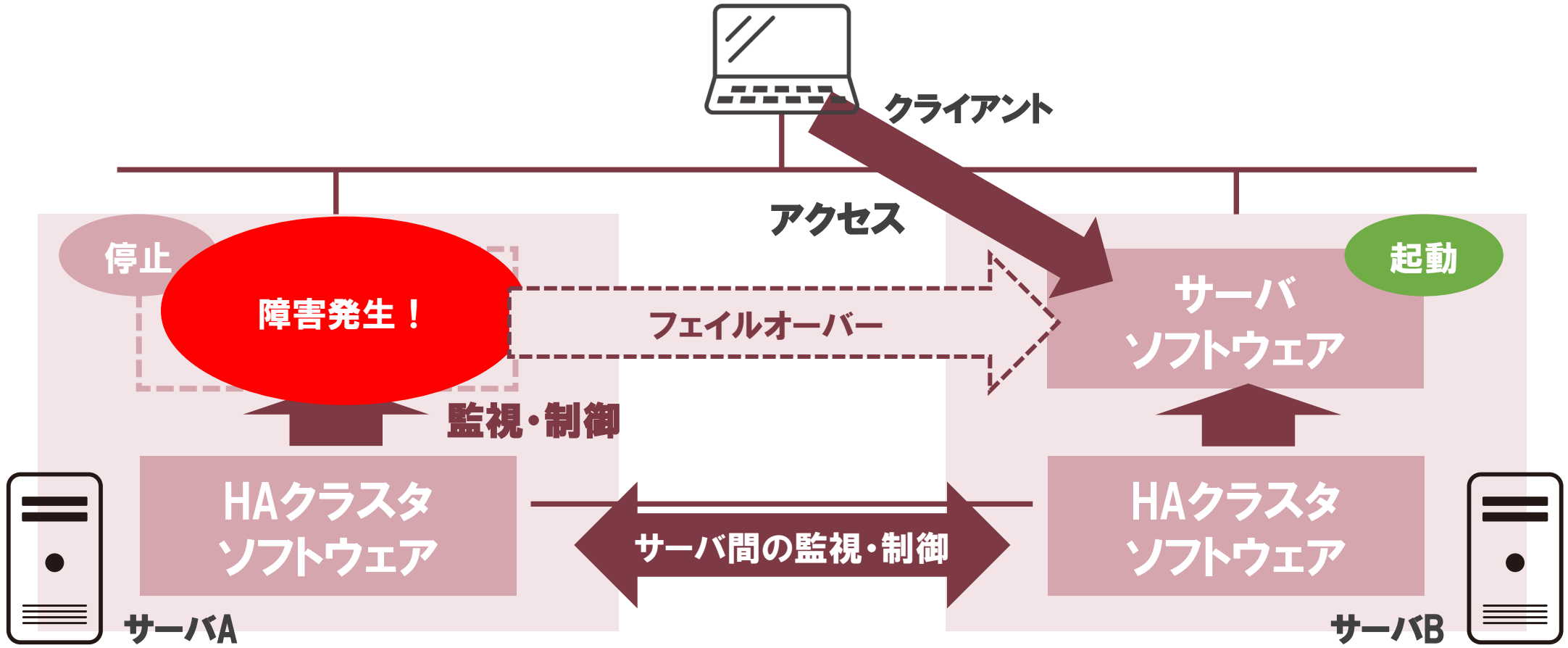
高可用性システムの実現方式 - HAクラスタ

- 稼働中のサーバで障害が発生した場合、サービス継続できていないことを検知し、自動的にフェイルオーバー（切換え）を実施



高可用性システムの実現方式 – HAクラスタ

- サーバAが担っていたサーバ機能が、サーバBに引き継がれ、サービス継続が行われる
  - 障害検出からフェイルオーバー完了までに多少の時間は要する



実際に、SPoFをなくそうとすると・・・

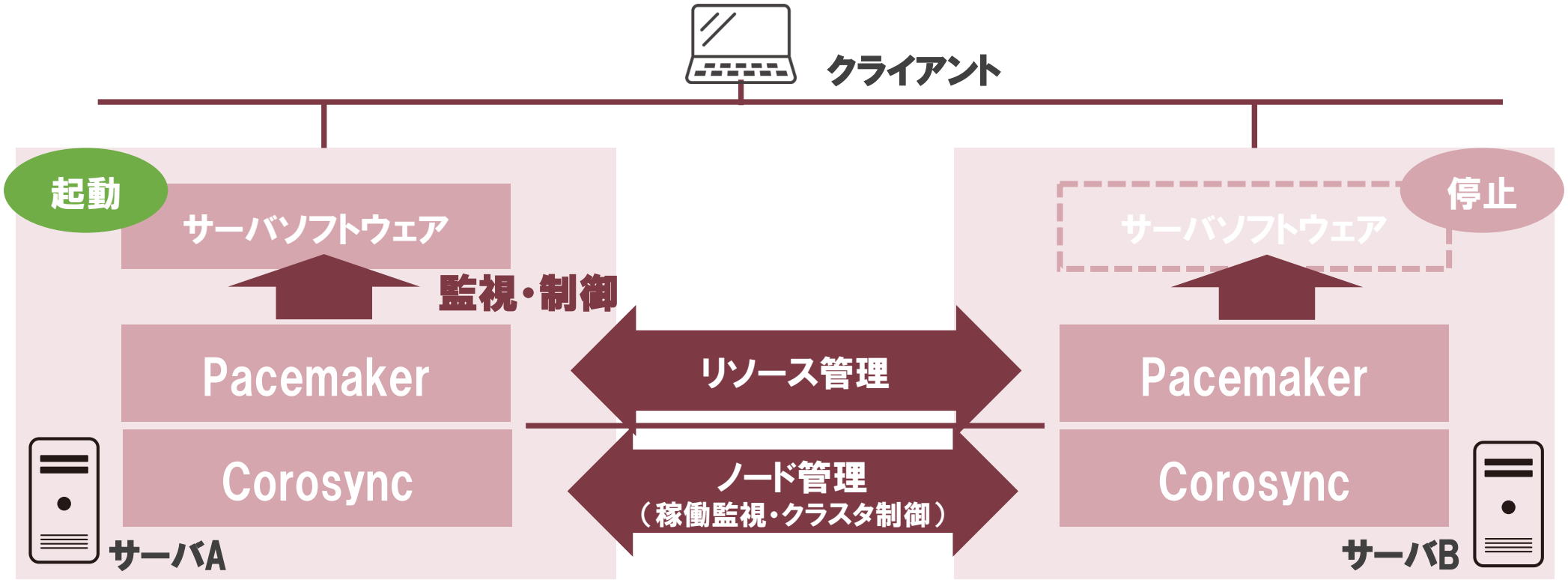
- 前スライドほど簡単ではない

- 対処しないといけない課題（例）
  - サーバ内で持つデータの同期、ストレージの切換対応
  - ハードウェア故障への対応
    - いろんなところが壊れる・・・ネットワーク機器や配線の冗長化
  - 監視対象のサービスが正常に動作しているものの、アクセス増などで高負荷状態、レスポンスが悪化しているため異常と判断して、フェイルオーバ
    - フェイルオーバ先でも同じ症状になるので、また切り替わる（を繰り返す 等）
  - 監視用ネットワークのNIC故障やケーブル断になると、監視ができない
    - 双方から見えなくなるので、どちらのサーバも起動しようとして同一サービスが重複してしまう（スプリットブレイン）
  - 仮想化やコンテナで高収容化できたが、ハードウェア故障の影響が広範囲に及ぶ・・・

- 複雑であるが、奥が深く面白い領域でもある！（→ 304試験）

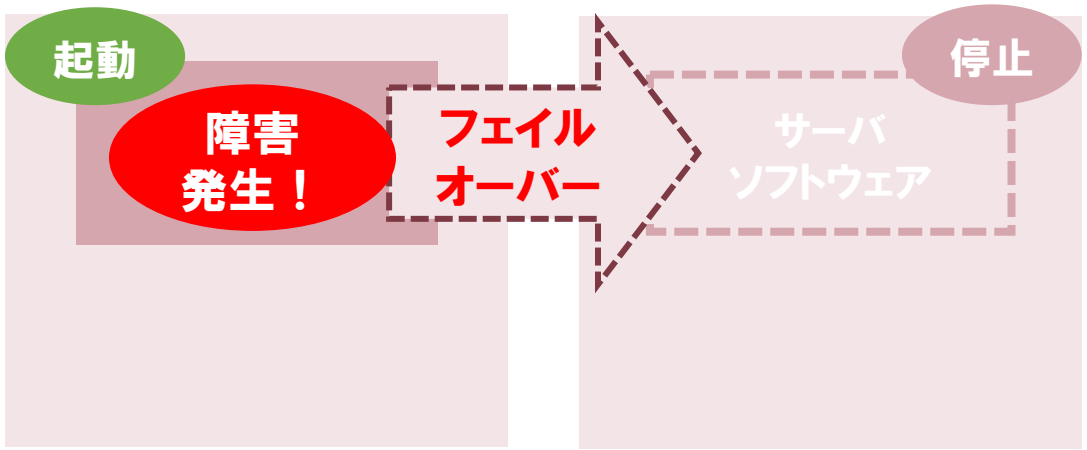
Pacemaker、Corosyncによる構成例

- Pacemaker：サービスの監視や制御
- Corosync：サーバ間のメッセージ交換、ハードウェア制御など



様々なHAクラスタ構成

■ アクティブ・スタンバイ構成

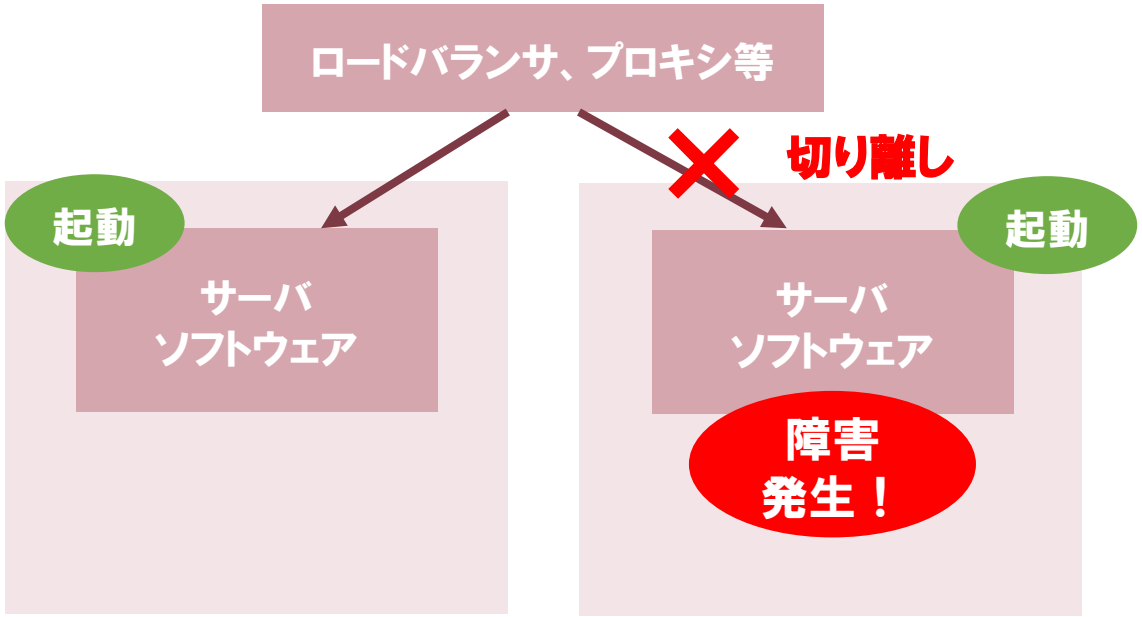


サーバ A
（アクティブ）

サーバ B
（スタンバイ）

アクティブなサーバで障害が発生した場合、
待機するスタンバイサーバに機能を移すことで
サービス継続を図る

■ アクティブ・アクティブ構成



サーバ A
（アクティブ）

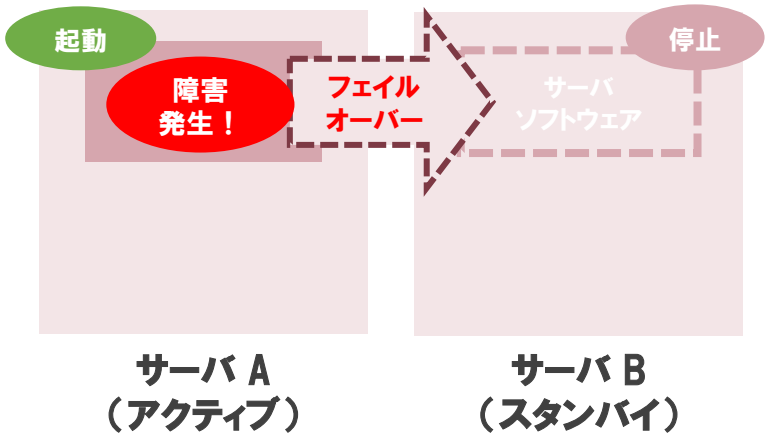
サーバ B
（アクティブ）

通常時も、複数のサーバがアクティブとして稼働して、
リクエストを分担して処理。障害発生時には、切り離
して縮退運転する（場合によっては性能低下）

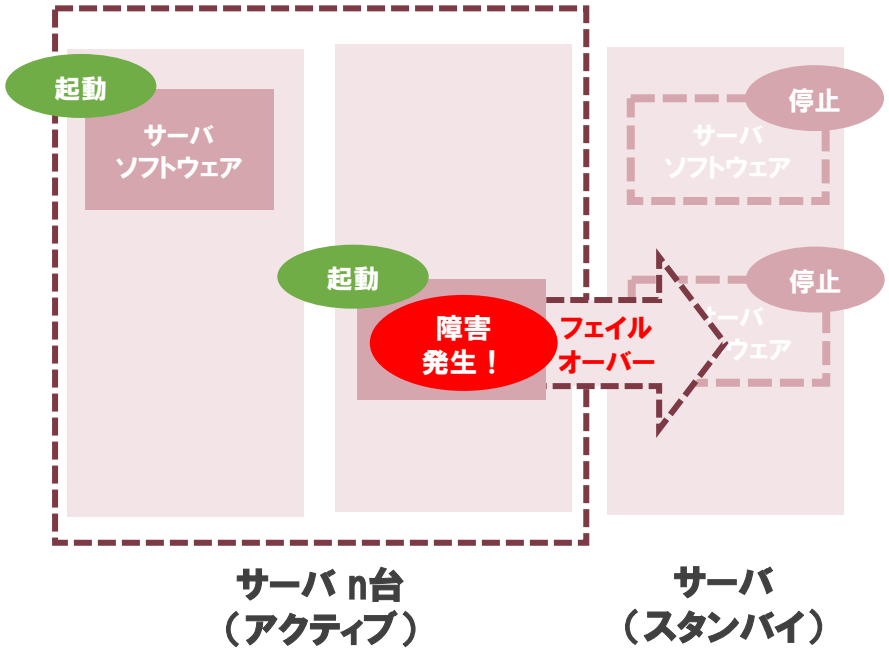
※ オープンソースソフトウェア HAproxyなどを利用することで、
ロードバランサ機能を実現できる

様々なHAクラスタ構成

■ 1+1構成

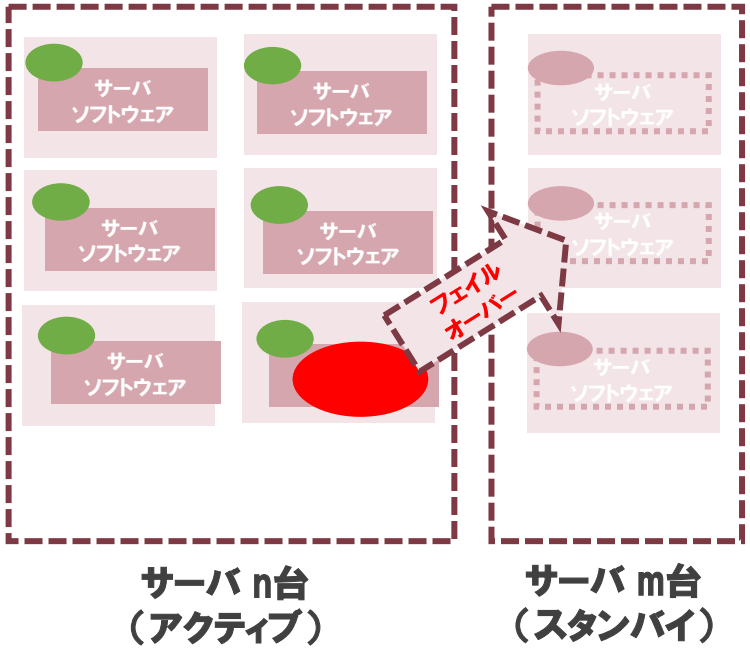


■ n+1構成



n台のアクティブなサーバに対して、
スタンバイサーバ1台を用意。
アクティブなサーバが障害が発生したら、
スタンバイにフェイルオーバー

■ n+m構成

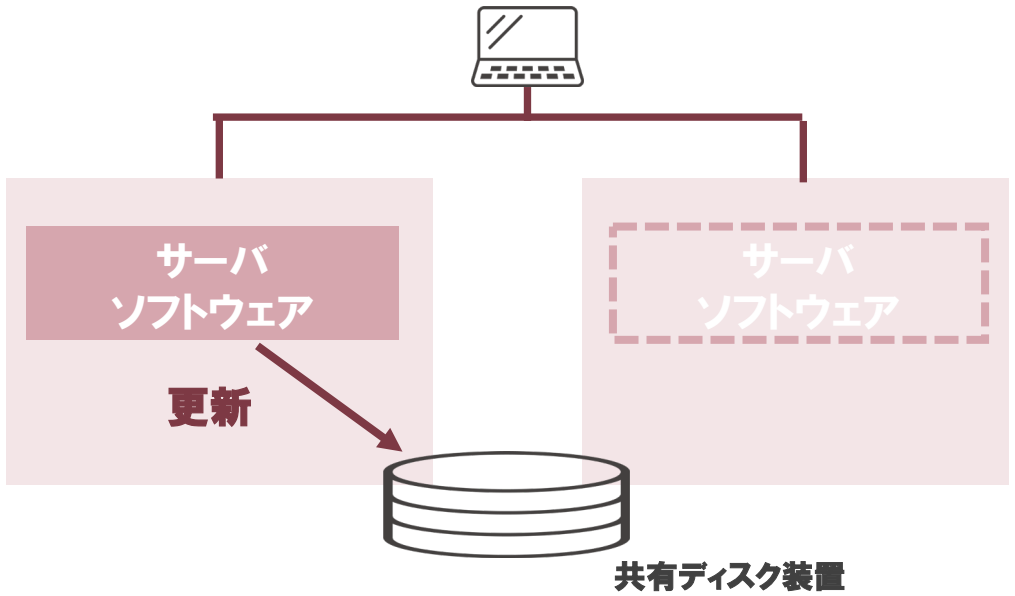


n台のアクティブなサーバに対して、
スタンバイサーバm台を用意。
アクティブなサーバが障害が発生したら、
スタンバイサーバのいずれかにフェイル
オーバー

様々なHAクラスタ構成

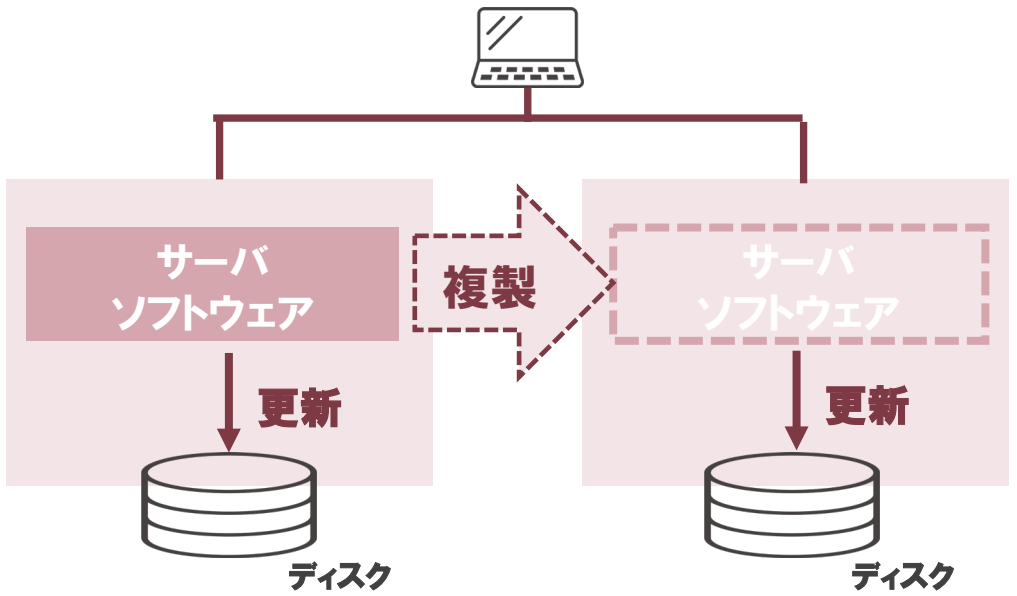
- ファイルオーバーしても、同一データへのアクセスを実現する方式が必要

シェアードディスク構成



共有ディスク装置を、片方のサーバからマウントして利用。
フェイルオーバー時には、故障サーバからアンマウントし、
フェイルオーバー先からマウントしてアクセスする。
2重にマウントしないよう排他制御が必要。
共有ストレージ装置の故障に対する考慮も必要となる。

シェアードナッシング構成



サーバソフトウェアの複製機能を利用して、それぞれの
サーバのディスクに同一の更新を行う。たとえば、
データベースソフト PostgreSQL には、同期レプリケーション
機能が実装されている

主題2.13 システムアーキテクチャ （前半）

- 2.13.1 高可用システムの実現方式
- **2.13.2 キャパシティプランニングとスケーラビリティの確保**
- 2.13.3 クラウドサービス上のシステム構成

性能要件の把握

- 一般的に、システム全体の性能要件は、アプリケーションや業務の観点から示されることが多い
  - 同時接続 xxxxユーザー・yyyyクライアント
  - リクエストからレスポンスまで zz秒以内 など

非機能要求グレードでみる 性能・拡張性（抜粋）

中項目	小項目	メトリクス（指標）
業務処理量	通常の業務処理量	ユーザ数、同時アクセス数、データ量 オンラインリクエスト件数、バッチ処理件数、業務機能数
	業務量増大度	ユーザ数増大率、同時アクセス数増大率、データ量増大率 オンラインリクエスト件数増大率、バッチ処理件数増大率、業務機能数増大率
	保管期間	保管期間、対象範囲
性能目標値	オンラインレスポンス	通常時レスポンス順守率、ピーク時レスポンス順守率、縮退時レスポンス順守率
	バッチレスポンス	通常時レスポンス順守度合い、ピーク時レスポンス順守度合い、縮退時レスポンス順守度合い
	オンラインスループット	通常時処理余裕率、ピーク時処理余裕率、縮退時処理余裕率
	バッチスループット	通常時処理余裕率、ピーク時処理余裕率、縮退時処理余裕率
リソース 拡張性	CPU拡張性	CPU利用率、CPU拡張性
	メモリ拡張性	メモリ利用率、メモリ拡張性
	ディスク拡張性	ディスク利用率、ディスク拡張性
	ネットワーク	ネットワーク機器設置範囲
	サーバ能力増強	スケールアップ、スケールアウト
性能 品質保証	帯域保証機能の有無	帯域保証の設定
	HWリソース専有の有無	HWリソース専有の設定
	性能テスト	測定頻度、確認範囲
	スパイク負荷対応	トランザクション保護

※ IPA（独立行政法人 情報処理推進機構）非機能要求グレード 2018
https://www.ipa.go.jp/sec/softwareengineering/std/ent03-b.html

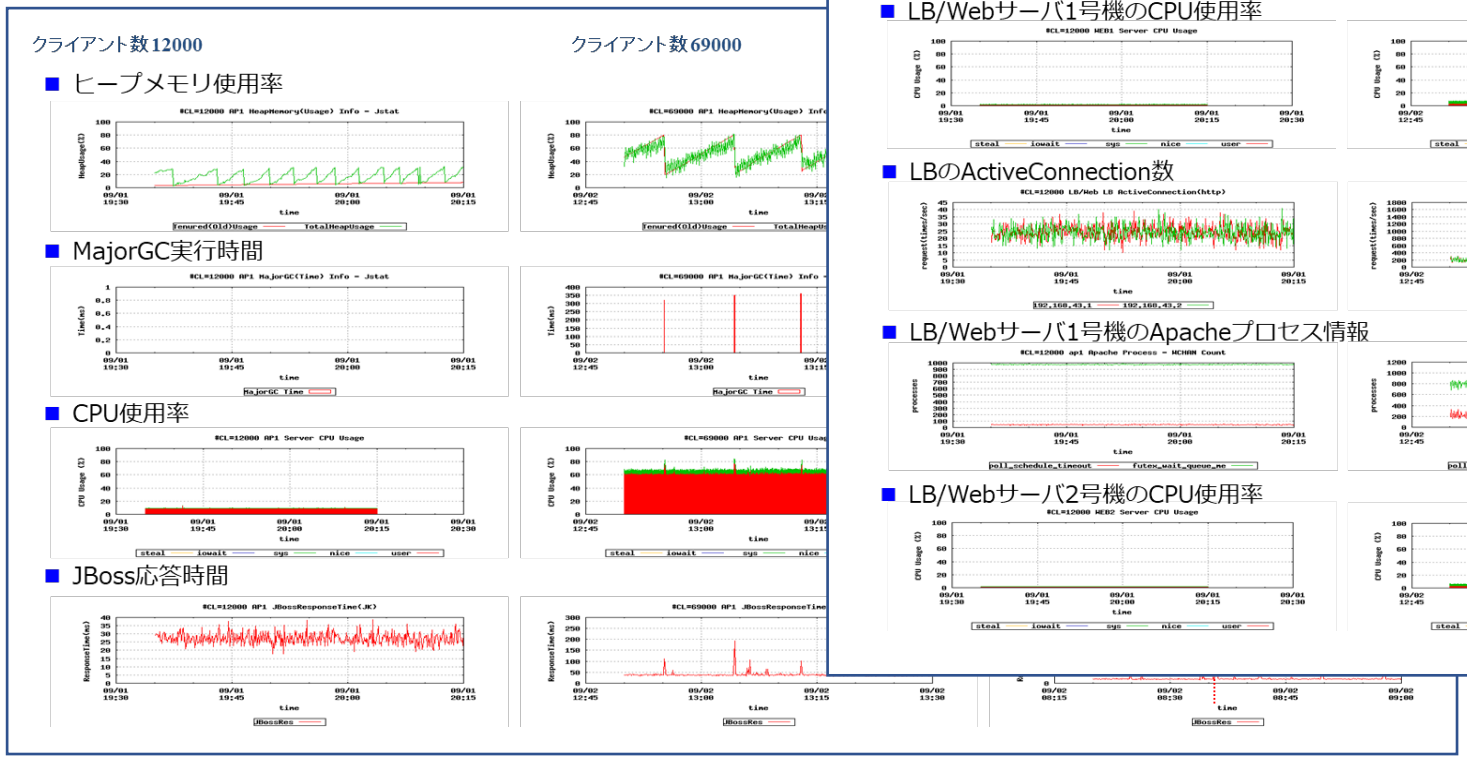
性能要件の把握

- 一般的に、システム全体の性能要件は、アプリケーションや業務の観点から示されることが多い
  - 同時接続 xxxxユーザー・yyyyクライアント
  - リクエストからレスポンスまで zz秒以内 など
- 実際に稼働する観点では、CPU、メモリ、ディスク、ネットワークなどのリソース
  - 足りているか
  - 使い切れているか（ソフトウェアの作りの問題から使い切れていないことも少なくない）
- オンプレ時代は 事前に適切に見積もり切れることが重要であった。
  最近は、ハードウェアの柔軟性向上やクラウドサービス活用の影響により、
  実際の稼働に応じて柔軟に調整できることも重要になっている。

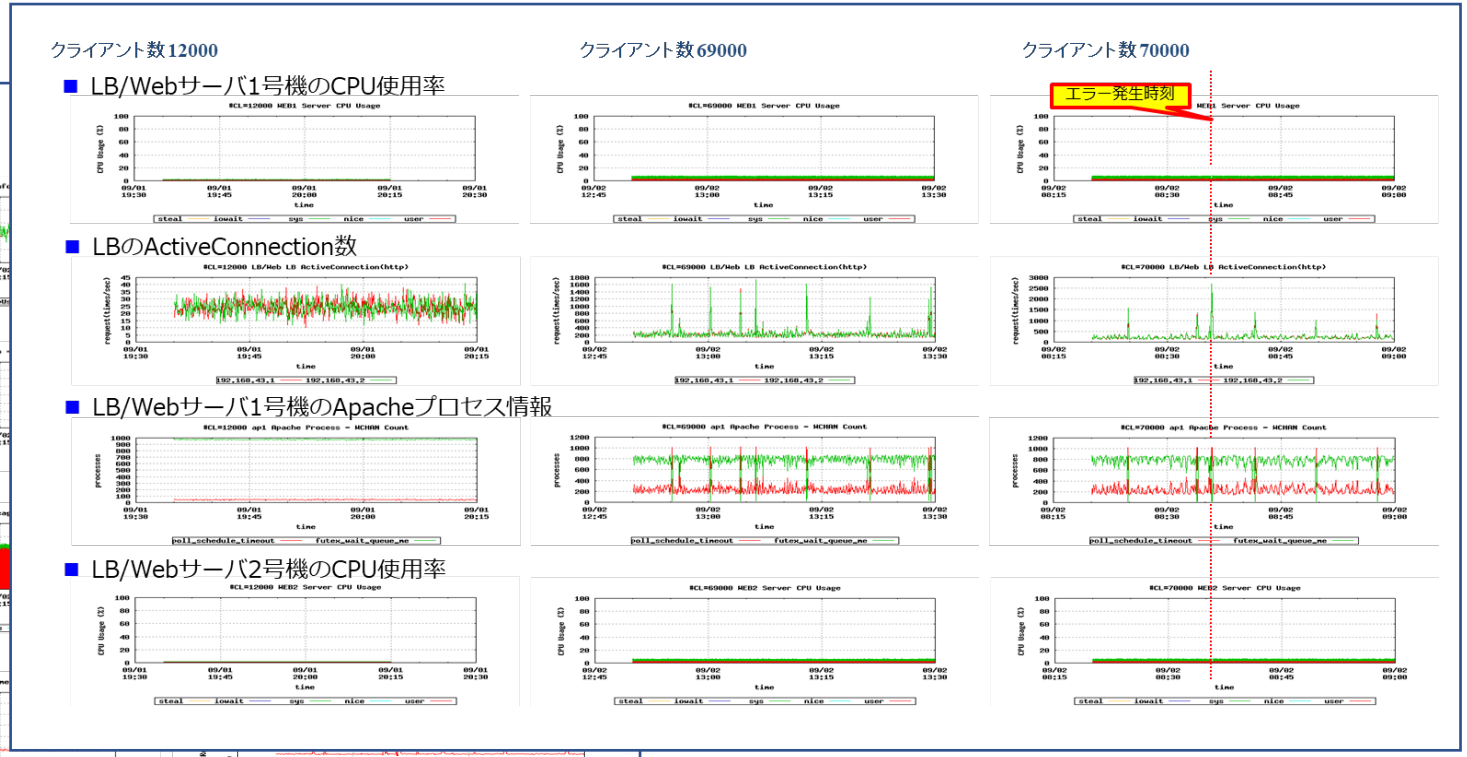
・「2.04.4リソース使用状況の把握」で学んだ内容

実際の性能検証（例）

- 業務アプリケーションを多数の疑似クライアントからアクセスし結果やレイテンシに問題がないかを確認しつつ、リソースのどこがボトルネックになるかを整理する等



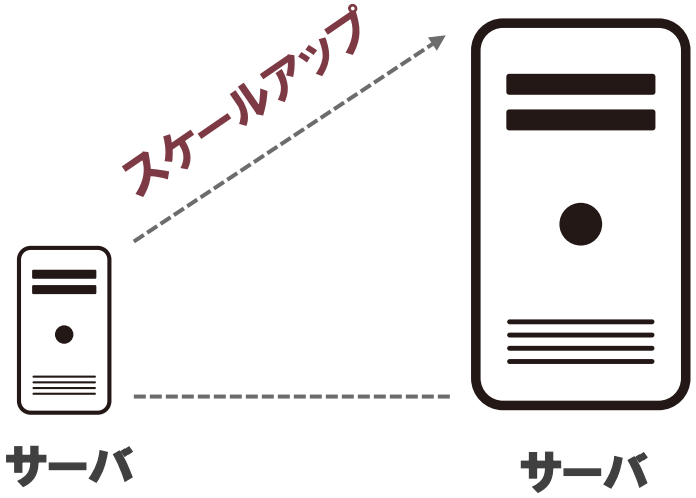
アプリケーションサーバの性能評価例



ロードバランサ/Webサーバの性能評価例

スケールアップ

■ 単一のサーバとして、CPU、メモリ、ディスクなどを
より性能のよいものにして、対応する方法

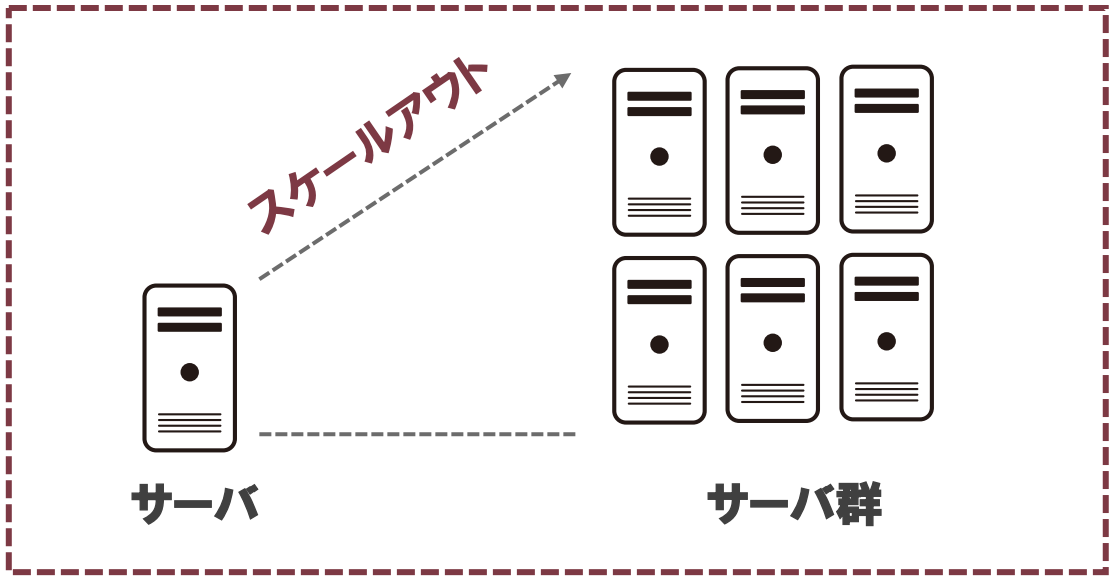


- 通常はサーバの停止を伴う
- クラウドサービスが提供する仮想マシン・IaaSを
利用する場合は、インスタンスタイプを変更する
だけでスケールアップできるケースもある

- アプリケーションの実装や設定によって、サーバの
リソースを増強しても、使い切れないケースがあるので、
注意が必要
  - OSレベル
  （1プロセスあたりのファイルディスクリプタ利用数の
  上限 など）
  - サーバソフトの設定レベル
  （Apache HTTP Serviceにおける最大クライアント数・
  スレッド数、Postfixにおけるプロセス数の上限 など）

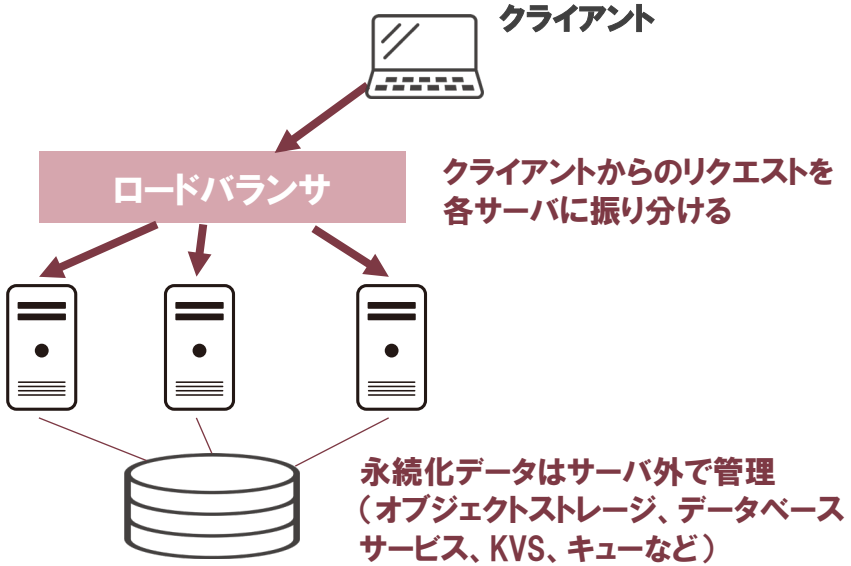
スケールアウト

■ 構成するサーバ台数を増やすことで、対応する方法



- 仮想マシン・コンテナのテンプレートや、構成管理ツールにより同一のサーバを複数構成する

- ロードバランサやDNSラウンドロビンなどを活用して、リクエストが各サーバに振り分けられるようにする
- サーバ内にデータがあると整合性が保てなくなるため、スケールアウト対象のサーバには永続化すべきデータは格納しない（別の仕組みにて管理）



※ DNSラウンドロビンでは、Aレコードなどに複数のサーバを指定し、名前解決のレベルで複数のサーバにリクエストを振り分ける。
障害時の切り離しのためにTTL等の設定にも留意が必要

主題2.13 システムアーキテクチャ （前半）

- 2.13.1 高可用システムの実現方式
- 2.13.2 キャパシティプランニングとスケーラビリティの確保
- **2.13.3 クラウドサービス上のシステム構成**

クラウドサービス – IaaS

- 仮想マシン・インスタンスは、様々なサービスが連携して、提供される

クラウドサービス – ストレージサービス

- 仮想マシン・インスタンス が動作するストレージは
  2種類に大別される

- エフェメラルストレージ（揮発性ストレージ）
  - インスタンス動作中にのみ利用可能
  - インスタント停止・終了すると失われる
  - 永続的に保存すべきデータの格納には向いていない
  - Amazon EC2におけるインスタンスストア

- 永続化ストレージ
  - インスタンス停止・終了しても引き続き使用できるストレージ
  - インスタンスにアタッチ（接続）することで利用
  - Amazon EC2におけるEBS(Elastic Block Store)

クラウドサービス – ネットワーク

■ 固定IPアドレスの割り当て

- 仮想マシン・インスタンスには、プライベートIPアドレスがDHCPなどにより自動で割り当てられるケースが多い
- 固定IPアドレスを割り当てるには、個別設定が必要となる

■ フローティングIPアドレス

- グローバルIPアドレスを、インスタンスに割り当てて利用できる
- 実際には、グローバルIPアドレスとインスタンスに割り当てられているプライベートアドレス（もしくはネットワークインターフェイス）を対応付ける
- 複数のインスタンス（クラスタ）で、アドレスを共有し、クラスタのうちアクティブな1台にIPアドレスが割り当てることもできる。アクティブなインスタンスに障害が発生すれば、別のインスタンスに割り当てられる
- Amazon EC2では、Elastic IPアドレスにより、グローバルIPアドレスを割り当てる

クラウドサービス – ネットワークセキュリティ

- テナントネットワーク
  - パブリッククラウド内で、仮想的なサブネットワーク（テナントネットワーク）が構成できる
  - 他のサービス利用者から隔離されたネットワークとして実現されている
  - AWSでは、Amazon VPCとして提供される

- ファイアウォール・セキュリティグループ
  - 仮想マシン・インスタンス等が、通信できる先を設定する仕組み
    - インバウンド・アウトバンド
    - ソースIPアドレス・ポート番号
    - ディスティネーションIPアドレス・ポート番号 などで設定できるのが通例

まとめ

- **高可用システムの実現方式**
  - 可用性と影響のある事象
  - 指標：MTBF、MTTR、稼働率、RTO、RPO
  - 高可用システムの実現 - HAクラスタ

- **キャパシティプランニングとスケーラビリティの確保**
  - 性能要件の把握と評価
  - スケールアップとスケールアウト

- **クラウドサービス上のシステム構成**
  - ストレージサービス、ネットワーク、ネットワークセキュリティ

- 次回は「2.13.4 典型的なシステムアーキテクチャ」
  - Web 3層システムなど、代表的なシステム構成例をご紹介する予定

Q & A

LinuC

LinuC-2

LinuC レベル2 Version 10.0
技術解説セミナー

主題2.13 システムアーキテクチャ （前半）

- 2.13.1 高可用システムの実現方式
- 2.13.2 キャパシティプランニングとスケーラビリティの確保
- 2.13.3 クラウドサービス上のシステム構成

2021年12月19日

濵野 賢一朗

LPI-JAPAN