UiPath Orchestrator システムの基盤設計・運用ガイド

目次

商標について

免責事項

第 1 章

Orchestrator 概論

UiPath Orchestrator は反復的なビジネスプロセスを実行する UiPath Robot を統合管理するウェブアプリケーションです。Orchestrator を使用することでリソースの作成、監視、デプロイメントといった RPA 環境の運用に必要な管理を行うことができます。

大規模な RPA 環境を実現するためには、以下の 4 要件が特に重要であり、UiPath Orchestrator を活用することで、これらの要件を効率よく実現することが可能となります。

- **スケジュール管理**
  - ロボットの起動、実行等のジョブ管理
    - 都度手動実行を避け、作業ミス、作業漏れ等のヒューマンエラーを防げます
- **統合運用管理**
  - 処理結果集中管理
    - エラーを素早く検知し、対応することが可能になります
  - 開発資源集中管理
    - 開発資源に更新があった場合に各々の端末に一括更新をすることで、作業工数の抑制、作業漏れ等によるロボットの実行エラーを防げます
  - ライセンス集中管理
    - ライセンスの更新工数の抑制や更新漏れを防げます
- **内部統制**
  - ユーザーごとのアクセスコントロール
    - ユーザーごとにアクセスコントロールすることで不正アクセス、不正実行などを防げます
  - 監査証跡の取得
    - 監査で必要な操作ログや処理結果ログを取得することができます
- **セキュリティ対策**

  - パスワードの安全な管理
    * パスワードをセキュアな環境で管理できます
    * エンドユーザが利用するパスワードとシステム管理者が管理するパスワードを分離してメンテナンスが可能

1.1 対象読者

本ドキュメントは UiPath Orchestrator v2018.4 を導入するためのシステム基盤設計・構築・運用に携わる方を対象としています。また、対象読者には以下のような知識があることを前提としています。

- 利用するオペレーティングシステム (Windows Server, Linux) の操作方法についての一般的な知識
- TCP/IP ネットワークについての一般的な知識
- Orchestrator が利用する各種ミドルウェア (SQL Server, Redis, Elasticsearch) の機能および用語に関する一般的な知識

1.2 製品が提供する (標準) 機能

カテゴリ	機能	概要
ダッシュボード	ダッシュボード	ロボット、ジョブ等の状況を一覧で表示
リリース管理	ワークフロー管理	ワークフローパッケージの集中管理
実行管理	ロボット管理	実行ロボットの登録 ロボットのステータス・起動ログ管理
	実行環境管理	ロボットのグループ分け
	ジョブステータス管理	各ジョブの実行状況、結果を一覧で表示 必要に応じてアラートメールの発報
	実行ログ管理	ジョブの実行ログの一元管理

次のページに続く

表 1.1 – 前のページからの続き

カテゴリ	機能	概要
	実行スケジューリング	各ジョブの実行スケジュール管理 週次・日時等の反復実行やロボット指定等
	アセット管理	変数の共有、及び、一元管理
	キュー管理	大量のトランザクションの分散実行
権限管理	ユーザー管理	Orchestrator に接続するユーザーの管理
	ユーザーロール管理	Orchestrator に接続するユーザーの権限管理
	テナント管理	全オブジェクト（パッケージ、ロボット等）をテナントごとに別管理
	組織単位管理	単一テナント内で、ロボット、プロセスなどのオブジェクトを組織単位ごとに管理
その他	ライセンス管理	Orchestrator に接続する Studio や各ロボットのライセンスを集中管理
	監査	Orchestrator のすべてのエンティティによって実行された操作の監査証跡を表示
	ログ分析（Kibana）	実行ログの高度な分析
	REST API	外部システムからのジョブの実行、キューの追加等、管理者が実行できる操作全て

1.3 用語

Orchestrator　UiPath Robot を統合管理するウェブアプリケーション

Studio　UiPath で自動化を行うフローを開発するためのアプリケーション

Robot　フローを実行するアプリケーションロボットには Attended Robot と Unatteneded Robot の二種類があります。

Attended Robot(**以下 AR と略す**)　このタイプのロボットはユーザーイベントによってトリガーされ、同じワークステーション上において有人で動作します。AR は Orchestrator と連携して、プロセス展開と実行ログを集中管理します。

Unattended Robot(**以下 UR と略す**)　ロボットは仮想環境で人手を仲介せずに実行され、任意の数のプロセ

スを自動化することができます。AR の機能に加えて、Orchestrator は、リモート実行、監視、スケジュール実行、およびワークキューのサポートを提供します。

組織単位　割り当てられたユーザー毎にテナント内の Orchestrator コンポーネント (ロボットやキュー等) を分離することができます。

ワークフロー　自動化の定義ファイル

パッケージ　ワークフローをまとめたファイル

ロボットグループ　ロボットをグループ化し、プロセスのデプロイに使用される

プロセス　パッケージを実行するために、ロボットグループとまとめて管理される

ジョブ　プロセスを実行するたびにジョブとして管理される

アセット　ジョブ実行時に参照可能な共有変数または認証情報を定義できます

キュー　トランザクションとして処理したい内容を制御できます

1.4 冗長構成の検討

Orchestrator を構成する各コンポーネントを冗長構成にするかを検討するにあたり、以下の内容を確認します。

1.4.1 Orchestrator がダウンした時の影響

機能	影響
AR	実行中のプロセスは継続して実行されます。 新規実行するためには予め Orchestrator で「切断状態で実行可能な時間 (Run Disconnected Hours)」の設定が必要です。
UR	実行中のプロセスは継続して実行されますが、UR として新規のスケジュール実行はできなくなります。(条件を満たせば AR 手動実行は可能です) キューやアセットを使用するアクティビティは実行にエラーが発生します。 Should Stop アクティビティによる中断ができなくなります。
Studio	キュー、アセットを使った開発ができなくなります。 パッケージを Orchestrator に Publish できなくなります。 Orchestrator の Development タイプを使用している場合、起動中の Studio はそのまま使用できます。新規起動は AR と同様「切断状態で実行可能な時間」を設定している場合は可能になります。

注釈:

「切断状態で実行可能な時間 (Run Disconnected Hours)」

この設定により AR が Orchestrator と切断された状態でもプロセス実行が可能となります。

動作条件は以下になります。

1. AR を Orchestrator でロボットとして登録し、プロセスをデプロイする

2. AR が Orchestrator に接続済で、Orchestrator からライセンスを付与されている

3. AR で該当プロセスを一度でも実行したことがある (実行済プロセスと依存関係のあるアクティビティは端末側にキャッシュされる)

この状態で Orchestrator がダウン、またはネットワーク断により Orchestrator と接続不可の状態になりましても、キャッシュされたプロセスは継続して実行可能となります。

ただし UiPath Robot サービスを再起動、または OS を再起動した場合は実行できなくなります。なお実行ログは端末側のローカルディレクトリにキャッシュされ、Orchestrator と接続可能になった時に送信されます。

1.4.2 各コンポーネントがダウンする可能性、復旧方法

コンポーネント	ダウンする可能性	復旧方法	備考
Orchestrator	HW 障害、プロセス障害、キャパシティを超えるロボット台数による過負荷	Internet Information Services(以下 IIS と略す) 再起動	
SQL Server	HW 障害、プロセス障害、重いクエリによる過負荷、DB 破損	データリストア、SQL Server 再起動	
Redis	HW 障害、プロセス障害	Redis 再起動	Orchestrator 関連のデータをキャッシュ、データリストア不要
Elasticsearch	HW 障害、プロセス障害	データリストア、Elasticsearch 再起動	ロボット実行ログにのみ影響
Kibana	HW 障害、プロセス障害	Kibana 再起動	ロボット実行ログにのみ影響

1.4.3 Orchestrator 構成パターン

ActiveDirectory は以下 AD と略す

シングル構成

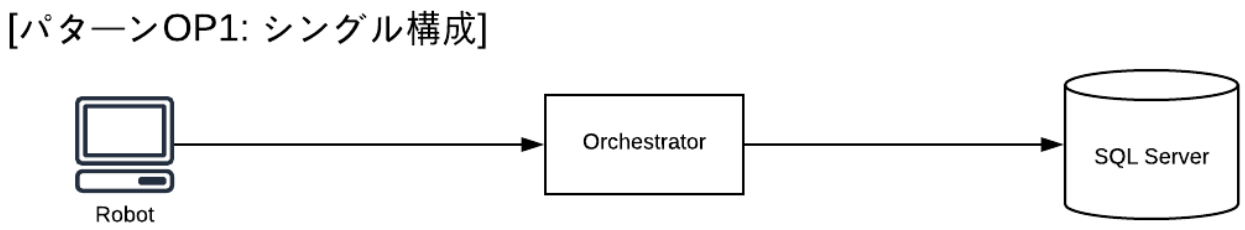


図 1.1　シングル構成

AD	可用性	拡張性	長所	短所
任意	低	低	最もシンプルな構成であり構築が容易	耐障害性がなく、拡張性にも乏しい

シングル構成 + Elasticsearch

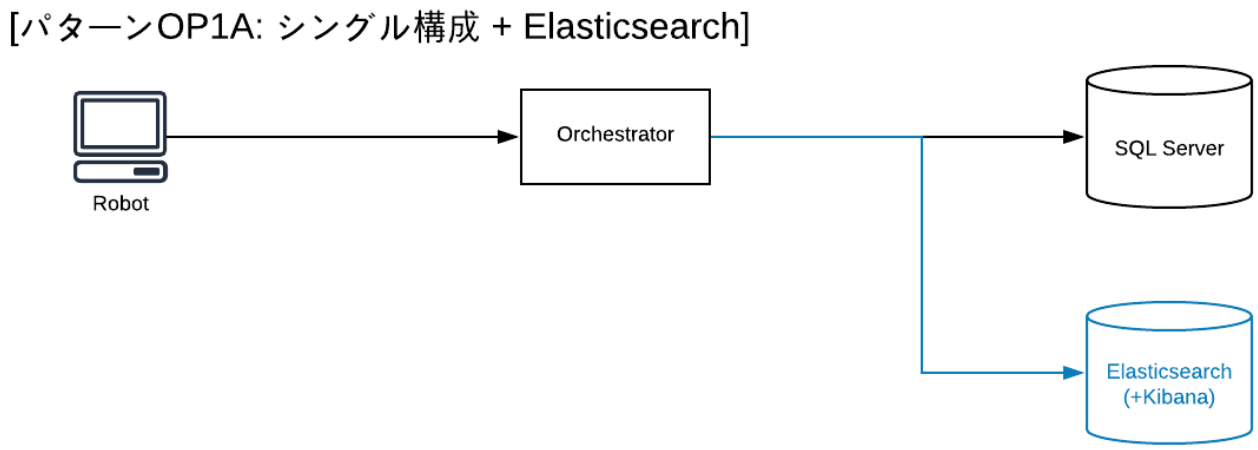


図 1.2　シングル構成 + Elasticsearch

AD	可用性	拡張性	長所	短所
任意	低	低	シンプルな構成であり構築が容易 Kibana によるログ分析が可能	耐障害性がなく、拡張性にも乏しい Elasticsearch の構築・運用コストがかかる

Active-Standby 冗長構成

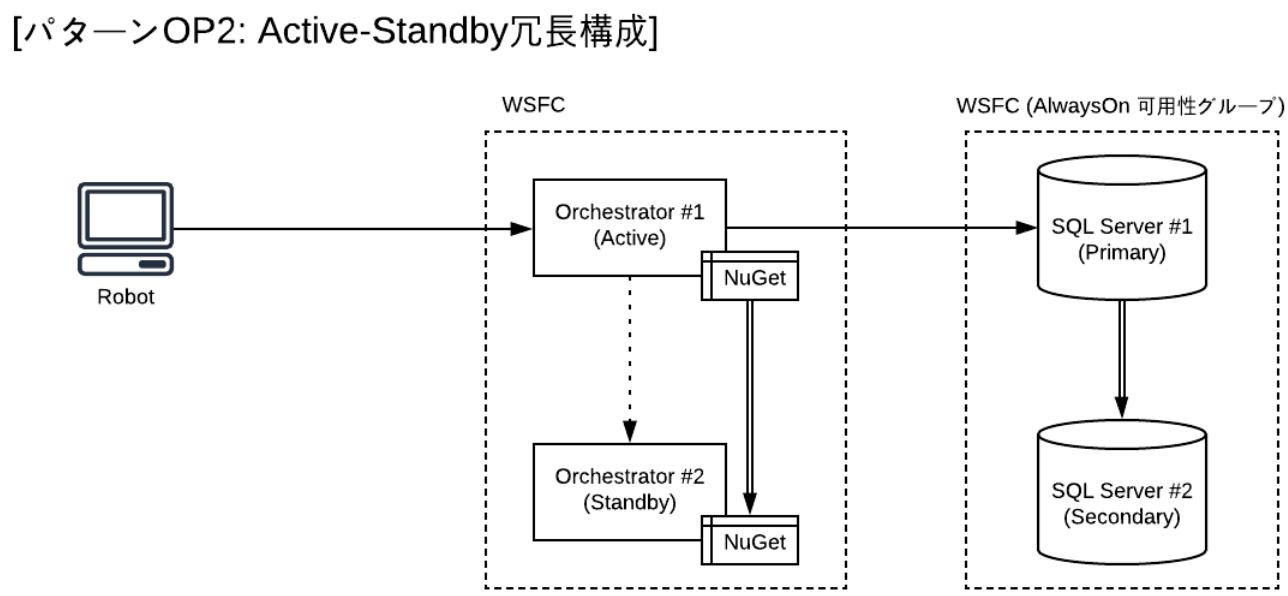


図 1.3　Active-Standby 冗長構成

AD	可用性	拡張性	長所	短所
必須	高	低	サーバー台数を押さえつつ単一障害点が除去されている	Active な Orchestrator は 1 台となるため、拡張性に乏しい また AD 環境が必須となる

Active-Standby 冗長構成 + Elasticsearch

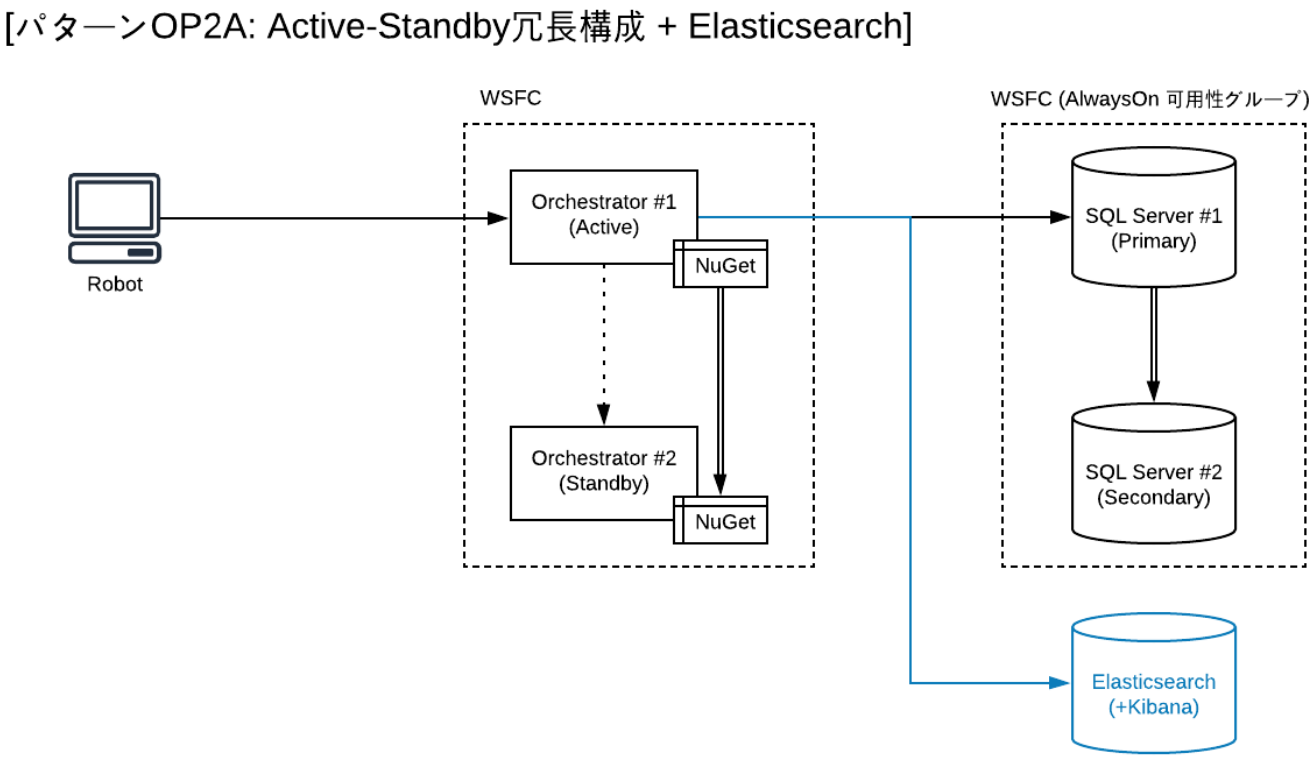


図 1.4　Active-Standby 冗長構成 + Elasticsearch

AD	可用性	拡張性	長所	短所
必須	低	低	サーバー台数を押さえつつ単一障害点が除去されている Kibana によるログ分析が可能	Active な Orchestrator は 1 台となるため、拡張性に乏しい Elasticsearch の構築・運用コストがかかる また AD 環境が必須となる

Active-Active 構成 (一部冗長化)

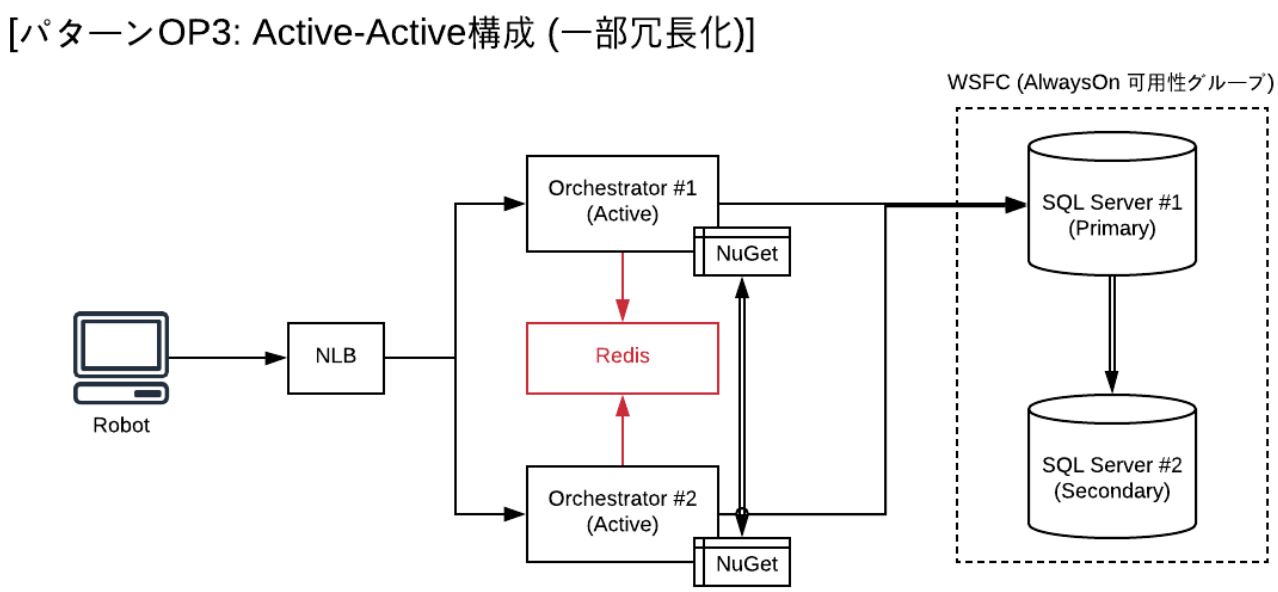


図 1.5　Active-Active 構成 (一部冗長化)

AD	可用性	拡張性	長所	短所
必須	中	高	拡張性とパフォーマンスに優れ、シングル構成に比べて可用性が向上している	NLB と Redis が必須となり、それらの冗長化は別途実装が必要となる Redis の動作環境として Linux が必要となる また AD 環境が必須となる

Active-Active 構成 (一部冗長化) + Elasticsearch

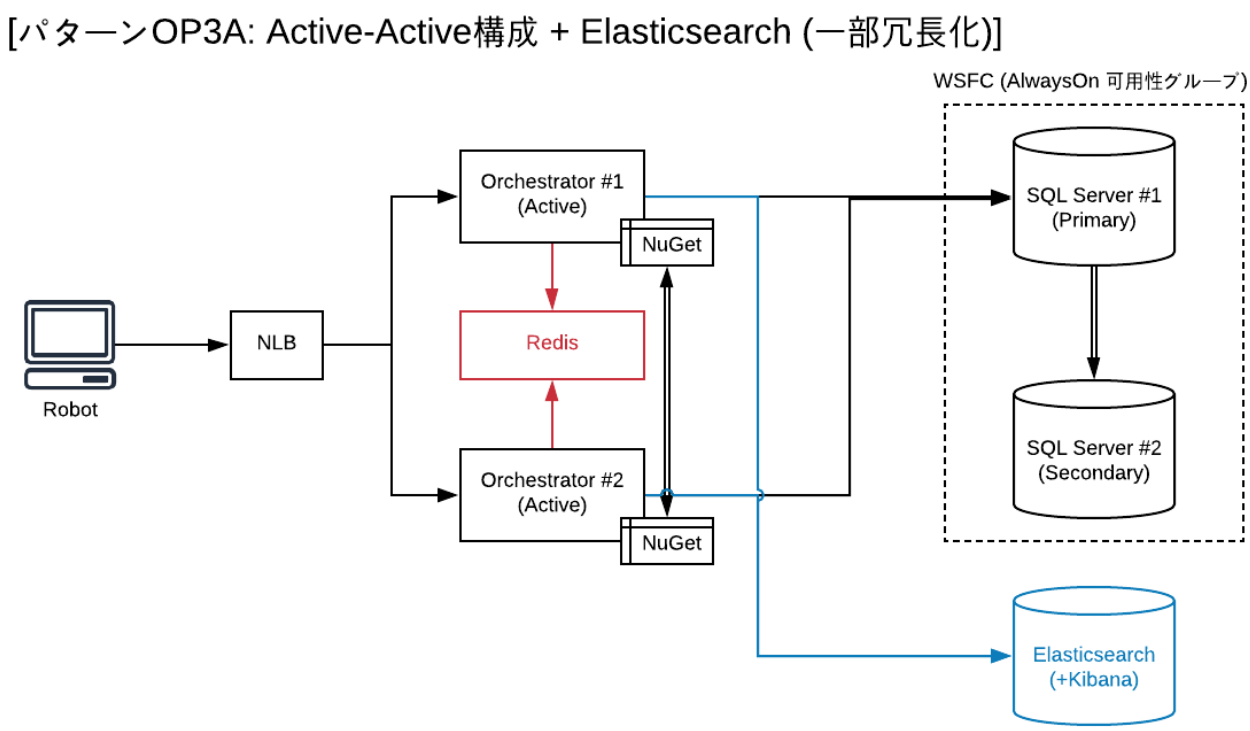


図 1.6　Active-Active 構成 (一部冗長化) + Elasticsearch

AD	可用性	拡張性	長所	短所
必須	中	高	拡張性とパフォーマンスに優れ、シングル構成に比べて可用性が向上している Kibana によるログ分析が可能	NLB と Redis が必須となり、それらの冗長化は別途実装が必要となる Elasticsearch の構築・運用コストがかかる Redis の動作環境として Linux が必要となる また AD 環境が必須となる

Active-Active 構成 (完全冗長化)

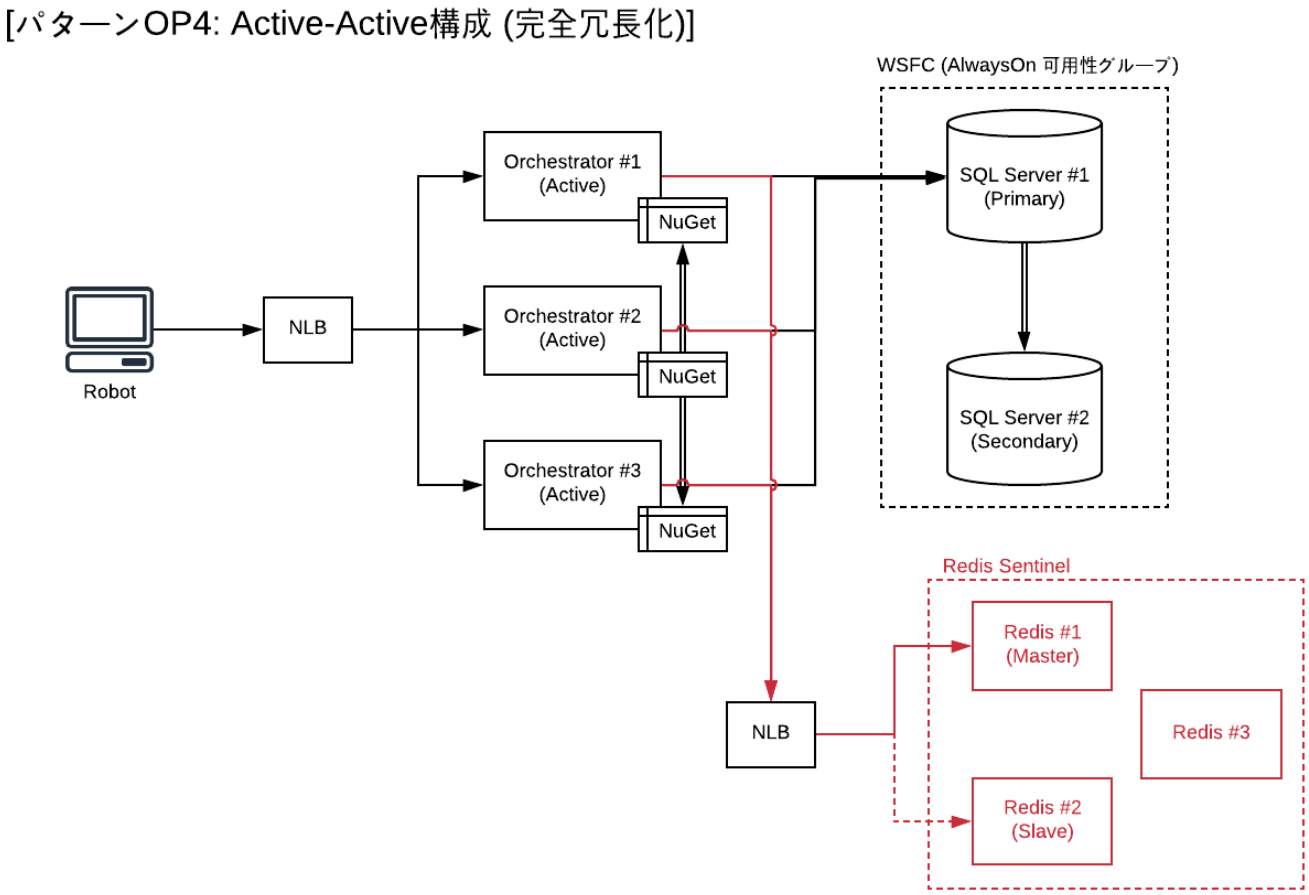


図 1.7　Active-Active 構成 (完全冗長化)

AD	可用性	拡張性	長所	短所
必須	高	高	単一障害点が完全に除去され可用性・拡張性に優れている パフォーマンス向上のために Orchestrator 台数を追加しスケールアウトすることが可能	サーバー台数が多くなるため構築・運用コストがかかる DB に実行ログが大量に蓄積されるため古いログの削除などの定期メンテナンス作業が必須となる Redis の動作環境として Linux が必要となる また AD 環境が必須となる

Active-Active 構成 (完全冗長化) + Elasticsearch

[パターンOP4A: Active-Active構成 + Elasticsearch (完全冗長化)]

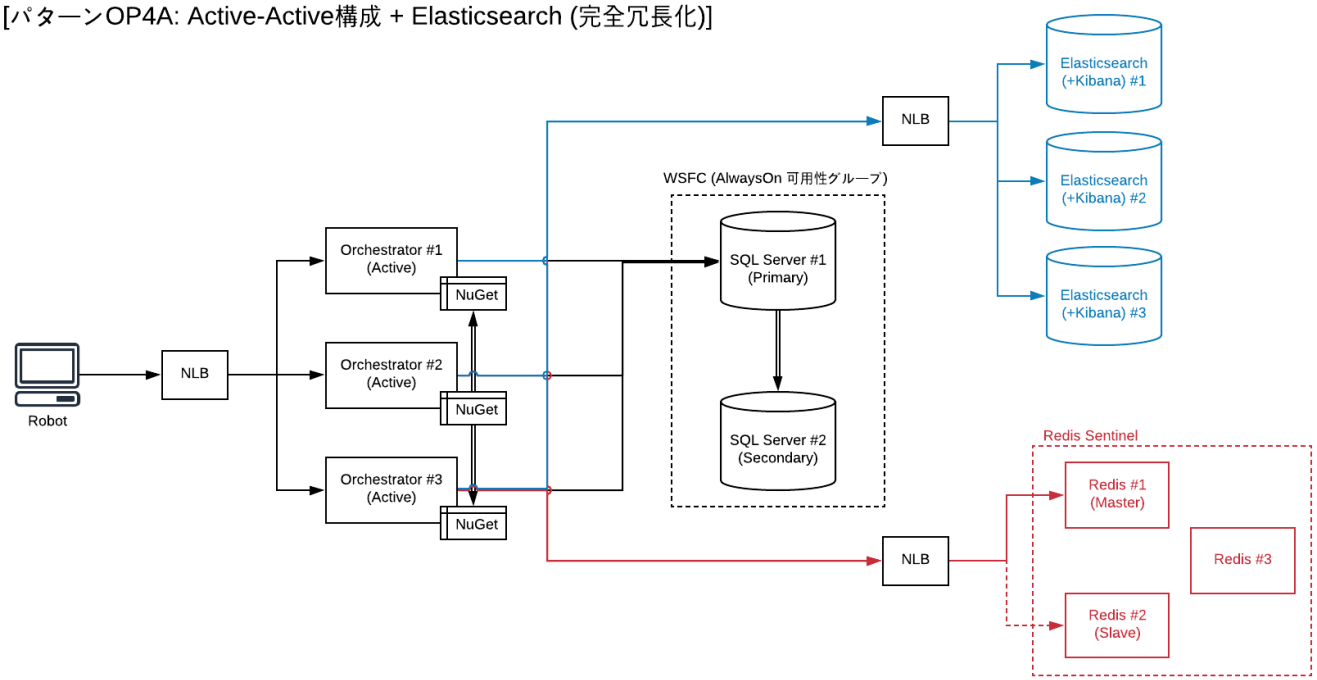


図 1.8　Active-Active 構成 (完全冗長化) + Elasticsearch

AD	可用性	拡張性	長所	短所
必須	高	高	単一障害点が完全に除去され可用性・拡張性に優れている パフォーマンス向上のために Orchestrator 台数を追加しスケールアウトすることが可能 Kibana によるログ分析が可能 DB 内の実行ログ削除などのメンテナンス作業を軽減することができる	サーバー台数が多くなるため構築・運用コストがかかる Elasticsearch の構築・運用コストがかかる Redis の動作環境として Linux が必要となる また AD 環境が必須となる

1.5 構成例

1.5.1 最小構成

Orchestrator を動作させるための最小構成例は 図 1.9 となります。最小構成例では、サーバを 2 台用意し、それぞれに以下の機能をインストールします。

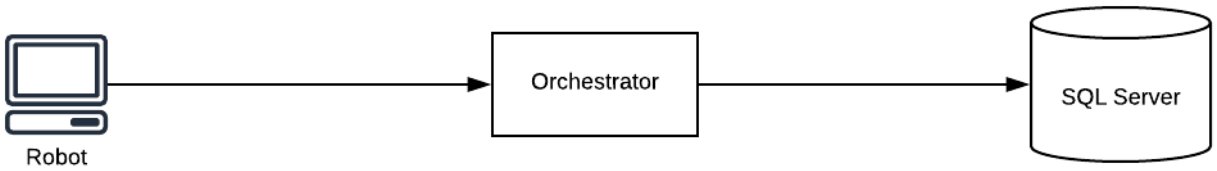


図 1.9　Orchestrator の最小構成

サーバ	OS	役割	インストール機能
Orchestrator サーバ	Windows Server	Orchestrator	IIS, Orchestrator
Database サーバ	Windows Server	Database	SQL Server

この環境は台数も少なく構築も簡単ですが、冗長化されていないため 1 台のサーバが停止すると Orchestrator の機能すべてが利用できません。開発環境に向いた構成例となります。

1.5.2 HA 構成例

Orchestrator を冗長化した構成例は 図 1.10 となります。HA 構成例では、サーバを 8 台用意し、それぞれに以下の機能をインストールします。

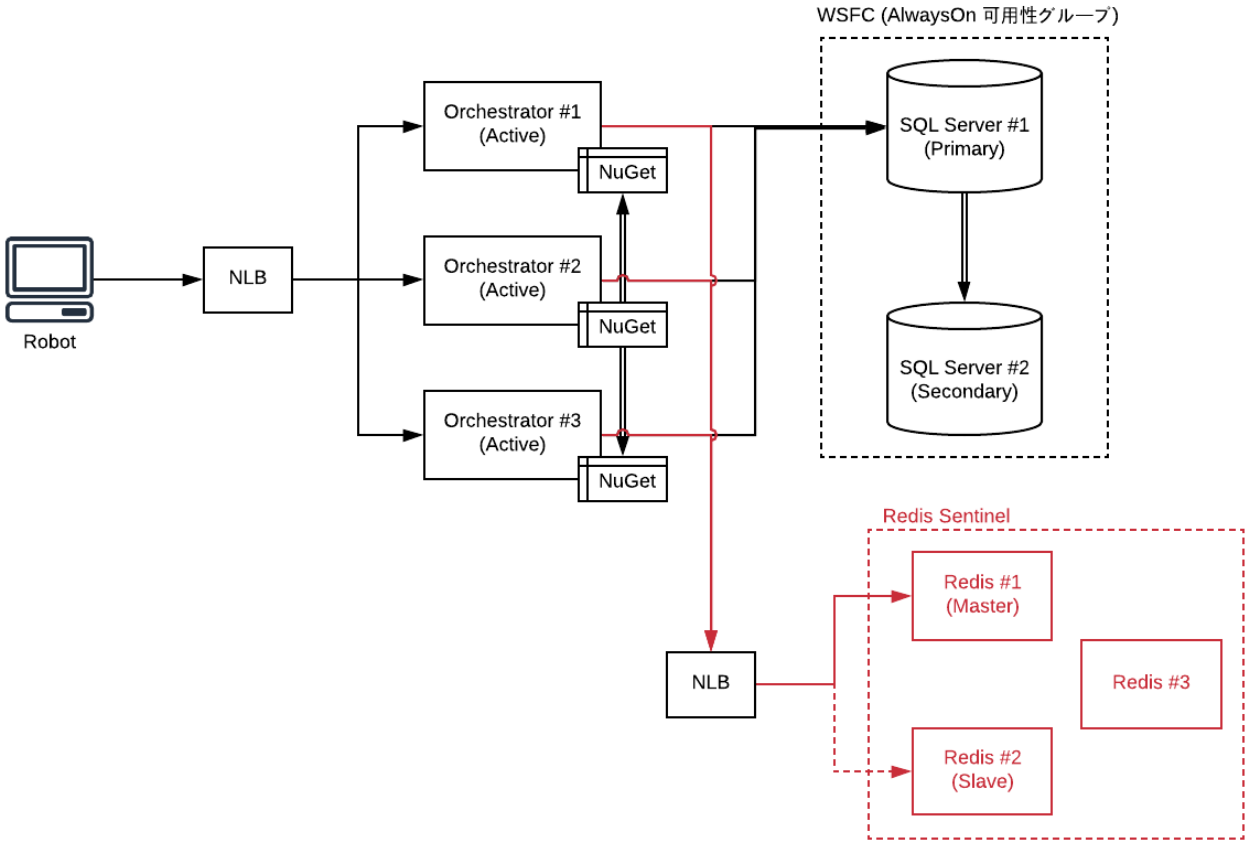


図 1.10　Orchestrator の HA 構成

サーバ	OS	役割	インストール機能
AD サーバ	Windows Server	AD	Distributed File System(以下 DFS と略す)、Windows Server Failover Clustering(以下 WSFC と略す)、AlwaysOn の機能が依存
Orchestrator サーバ 1	Windows Server	Orchestrator	IIS, Orchestrator、DFS
Orchestrator サーバ 2	Windows Server	Orchestrator	IIS, Orchestrator, DFS
Database サーバ 1	Windows Server	Database(稼働系)	SQL Server、WSFC、AlwaysOn
Database サーバ 2	Windows Server	Database(待機系)	SQL Server、WSFC、AlwaysOn
Redis サーバ 1	Linux または Windows	Database(Master), Sentinel	Redis
Redis サーバ 2	Linux または Windows	Database(Slave), Sentinel	Redis
Redis サーバ 3	Linux または Windows	Sentinel	Redis

Orchestrator を 2 台で構成 (Active-Active) します。Active-Active 構成で構築するため、通常時は負荷分散を行いながら運用することができ、障害発生時にも 1 台のサーバダウンであれば、もう一台のサーバで継続して運用することが可能です。また、Database も SQL Server の AlwaysOn 可用性グループ機能[*1] による冗長化を利用することで Database サーバのダウン時にも継続して運用することができます。本番環境では本構成のように Orchestrator とデータベースを冗長化した構成を推奨します。

DFS や Redis は Orchestrator を分散構成で動作させるために必要となります。DFS はロボットを実行させるためのパッケージファイルを複数の Orchestrator 間で同期するために利用します。Redis は複数の Orchestrator 間で情報の更新を通知したり、セッションの管理に利用します。Windows 環境では Redis のサポートが提供されていないため、Linux 環境での仕様を推奨いたします。Redis のバージョン等につきましては以下を参照してください。

https://orchestrator.uipath.com/lang-ja/v2018.4/docs/software-requirements

1.5.3 大規模環境

Orchestrator でロボットを数百台管理するような大規模運用での構成例は 図 1.11 となります。大規模構成ではロボット 250 台ごとに 1 台の Orchestrator を追加します。また大量のジョブのログ格納、分析には Elasticsearch/Kibana を使用します。それぞれのサーバには以下の機能をインストールします。

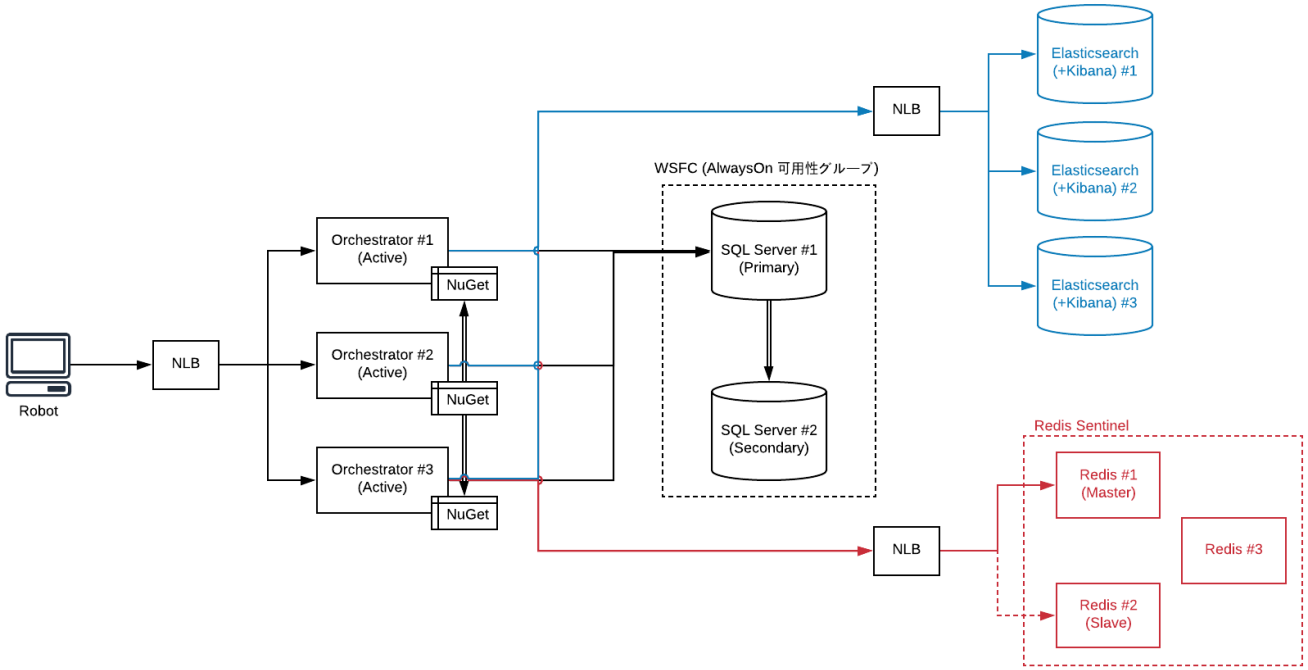


図 1.11　Orchestrator の大規模構成

[*1] https://docs.microsoft.com/ja-jp/sql/database-engine/availability-groups/windows/always-on-availability-groups-sql-server

サーバ	OS	役割	インストール機能
AD サーバ	Windows Server	AD	DFS、WSFC、AlwaysOn の機能が依存
Orchestrator サーバ (複数台)	Windows Server	Orchestrator	IIS, Orchestrator、DFS
Database サーバ 1	Windows Server	Database(稼働系)	SQL Server、WSFC、AlwaysOn
Database サーバ 2	Windows Server	Database(待機系)	SQL Server、WSFC、AlwaysOn
Redis サーバ 1	Linux または Windows	Database(Master), Sentinel	Redis
Redis サーバ 2	Linux または Windows	Database(Slave), Sentinel	Redis
Redis サーバ 3	Linux または Windows	Sentinel	Redis
ログサーバ 1	Linux または Windows	Database(Data, Master)	Elasticsearch, Kibana
ログサーバ 2	Linux または Windows	Database(Data, Master)	Elasticsearch, Kibana
ログサーバ 3	Linux または Windows	Database(Data, Master)	Elasticsearch, Kibana

本構成例はロボット数百台を運用できるように、HA 構成と比較すると以下を構成することでより大規模で運用しやすい構成となります。

- Orchestrator のサーバ台数を追加 (**負荷分散**)
- ロボットの実行ログを SQL Server から Elasticsearch へオフロード (**大容量ログへの対応、運用負荷の低減**)
- Elasticsearch/Kibana を利用したロボットの実行ログ分析 (**実行状況の可視化、障害への迅速な対応**)

Elasticsearch/Kibana のバージョン等につきましては、以下の URL を参照してください。

https://orchestrator.uipath.com/lang-ja/v2018.4/docs/software-requirements

第 2 章

システム設計

2.1 想定システム

2.1.1 本番、テスト、検証、開発

開発、テスト、検証、本番の 4 つ環境を考慮して構成を検討する必要があります。

「開発」→「ユーザー受け入れ評価 / ステージング」 →「本番導入 / 保守」のソフトウェアライフサイクルに合わせた、開発、テスト、本番環境を構築することにより、本番環境に影響を与えることなく、ワークフローのテストを行うことができます。また Orchestrator テスト用の検証環境を用意することにより、UiPath プラットフォームのバージョンアップ評価を行うことができます。UiPath の リリースサイクル にてアナウンスされている通り、お使いのバージョンのサポート期間が終了したり、新しくリリースされたバージョンに実装された新機能やバグフィックス等が必要になった時に備えて、いつでもバージョンアップを検証できる検証環境をご用意いただくことを強くお勧めします。

開発・テスト・検証・本番環境の棲み分け

- 開発環境
  - 開発者が Orchestrator の機能 (アセットやキュー) を使って開発し、試せる
  - 最小構成で十分
- テスト環境
  - 開発されたワークフローを、本番環境に準ずる似た環境で評価し、問題をあらかじめ洗い出すこと（テストされていないワークフローは本番でトラブルを引き起こす）
  - 本番に配置されるアセットやキューなど Orchestrator 設定を模倣すること
  - 接続されている BOR/FOR の端末も本番と同じバージョン、アクティビティが用意されていること
- 検証環境
  - Orchestrator のバージョンアップ、パラメータ変更など、Orchestrator 自身の評価を行う

  - 本番環境と同様に高可用性構成が取られていれば十分ですが、必ずしも本番環境と同じ台数である必要はなし

- 本番環境
  - 少なくとも高可用性構成が取られており、適切なスケーリングがなされていること

2.1.2 環境選定の判断ポイント

初期導入・運用コスト、影響範囲の局所化の考え方等により実際の環境を選定します。

図 2.1　環境選定の判断ポイント

2.2 非機能要件

2.2.1 性能 (ロボット台数依存)

詳細は以下にも記載があります。

https://orchestrator.uipath.com/lang-ja/v2018.4/docs/hardware-requirements-orchestrator

Orchestrator

- 1 台の Orchestrator で管理できる許容量
  - 250 台のロボットによる 100 プロセスの同時実行
  - 1000 台のロボットの接続

ロボット 100 台以上の場合、Active/Active & Orchestrator 3 台以上の構成をお勧めします。

SQL Server

ジョブ実行時に出力するログ

- SQL server はスケールアウトができないため、スケールアップを目指してサイジング要件を上げることをお勧めします。
  - CPU コア数（4+）
  - メモリー（8 GB+）
  - IO（SSD、SAN、RAID 10 など）

上限に当たったら、SQL サーバに送るログの量を減らすため、Elasticsearch を活用すると同時に、SQL 向けの NLog ログレベルを上げる（デフォルト：Info → Warning）

2.2.2 信頼 (冗長構成の必要性)

冗長構成のほか、DR を意識した構成をとることもできます。

DR(災害復旧) 構成

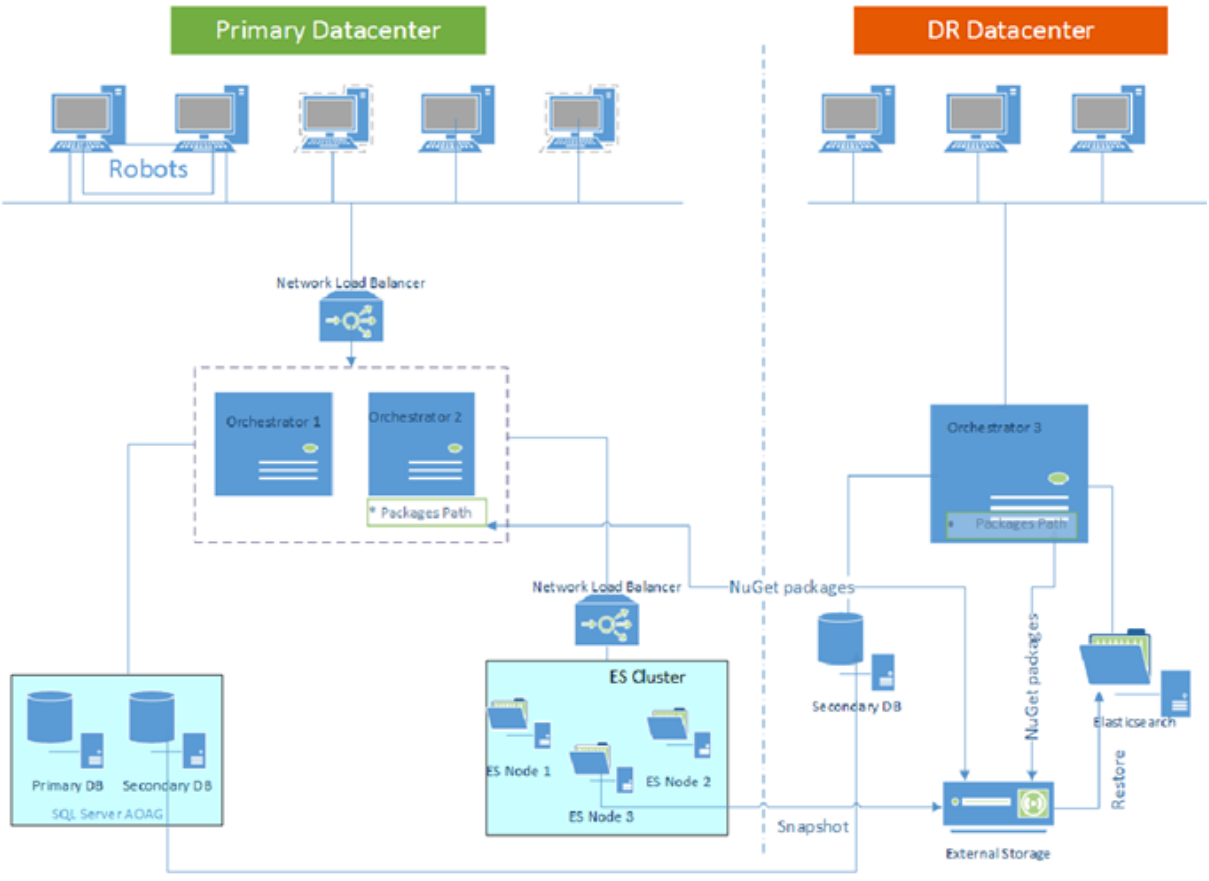


図 2.2　DR(災害復旧) 構成例

- Always On Availability Group を含む SQL Server Farm のマシンを最低 1 台、DR データセンターに物理的に構築する必要がある。

- プライマリ・データセンターと DR データセンター間にはネットワーク接続がある。
- プライマリ・データセンターで「Snapshot」が作成した Elasticsearch スナップショットを保存するために、外部ストレージを DR データセンターに提供、設置する。これらのスナップショットは、「Restore」によって DR データセンターに位置する Elasticsearch インデックスに読み込まれ、適用される。「Snapshot」と「Restore」は、Elasticsearch によって提供されたツールである。
- NuGet 形式を持つ自動化パッケージ (アーティファクト) は、外部ストレージに保存され、各 Orchestrator は " NuGet.Packages.Path " 構成設定を使用してこのストレージを指し示す。
- 任意の (ダイヤグラムで構成されない) 外部ストレージのミラーを作成する。

DR(災害復旧) ＋高可用性 構成

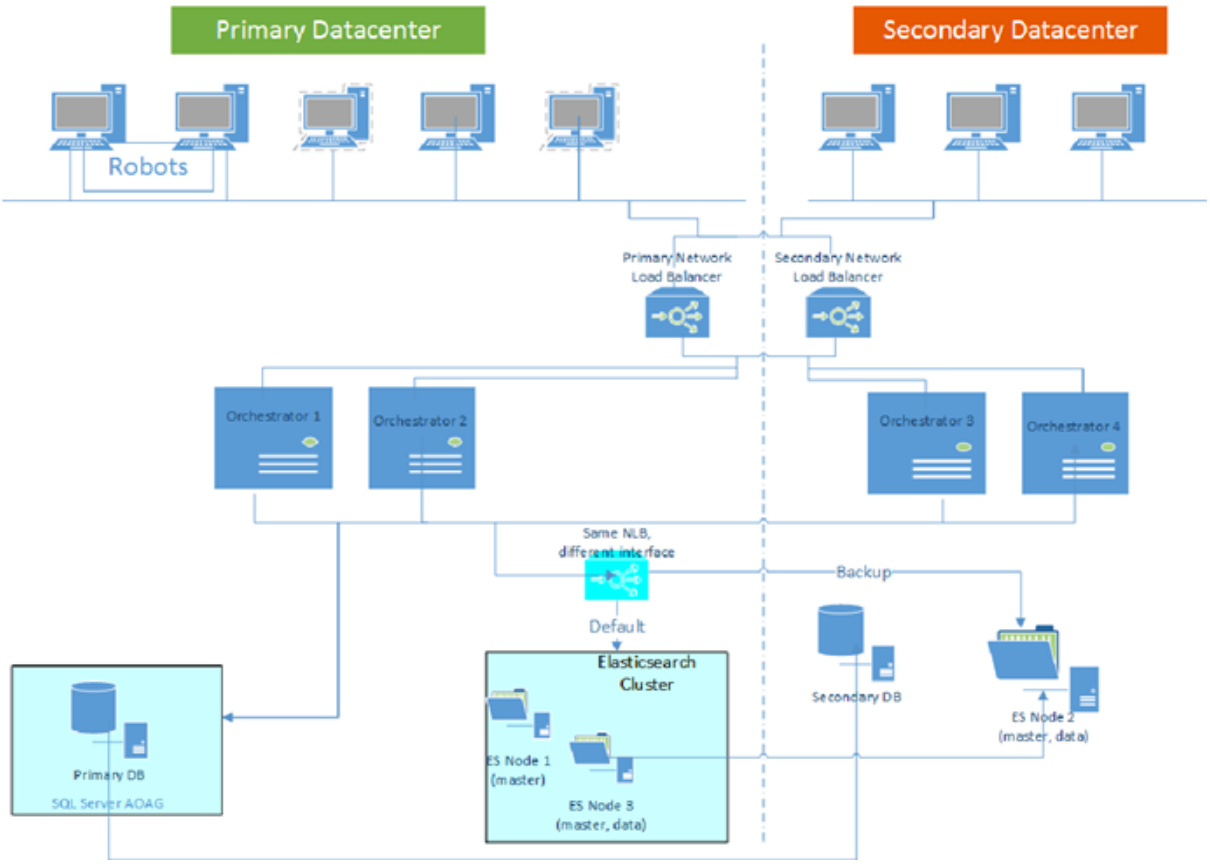


図 2.3　DR(災害復旧) + 高可用性 構成例

この構成例では、前出の DR 構成に加えて、ネットワークロードバランサーを冗長化し、プライマリとセカンダリの両データセンターのロボットと Orchestrator を相互に運用することで、高可用性を実現している。

前提条件としては、前出の DR 構成の場合に加えて、データセンター間のネットワーク帯域が十分に確保されている必要がある。

2.3 調達

2.3.1 HW, OS, ミドルウェア

HW 要件

https://orchestrator.uipath.com/lang-ja/docs/hardware-requirements-orchestrator

SW 要件

https://orchestrator.uipath.com/lang-ja/docs/software-requirements

2.3.2 ドメイン, SSL 証明書

Orchestrator との通信は https で行われるため、適切な SSL 証明書が必要となります。Windows AD CS を利用することも可能です。なお http で動作させることはできませんので、https での運用をお願いします。

2.3.3 ライセンス

https://www.uipath.com/ja/licensing-models

2.4 インフラ設計

2.4.1 構成

基本的な構成要素

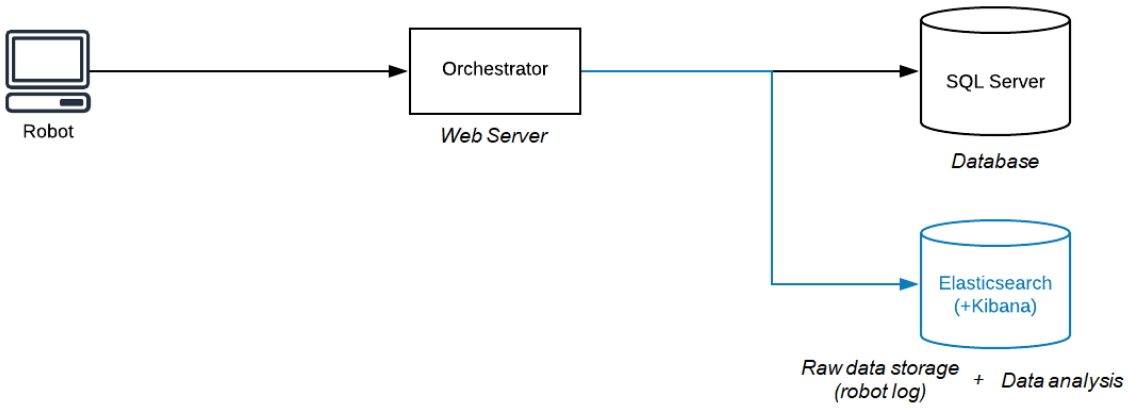


図 2.4　基本的な構成要素

2.4.2 アーキテクチャ

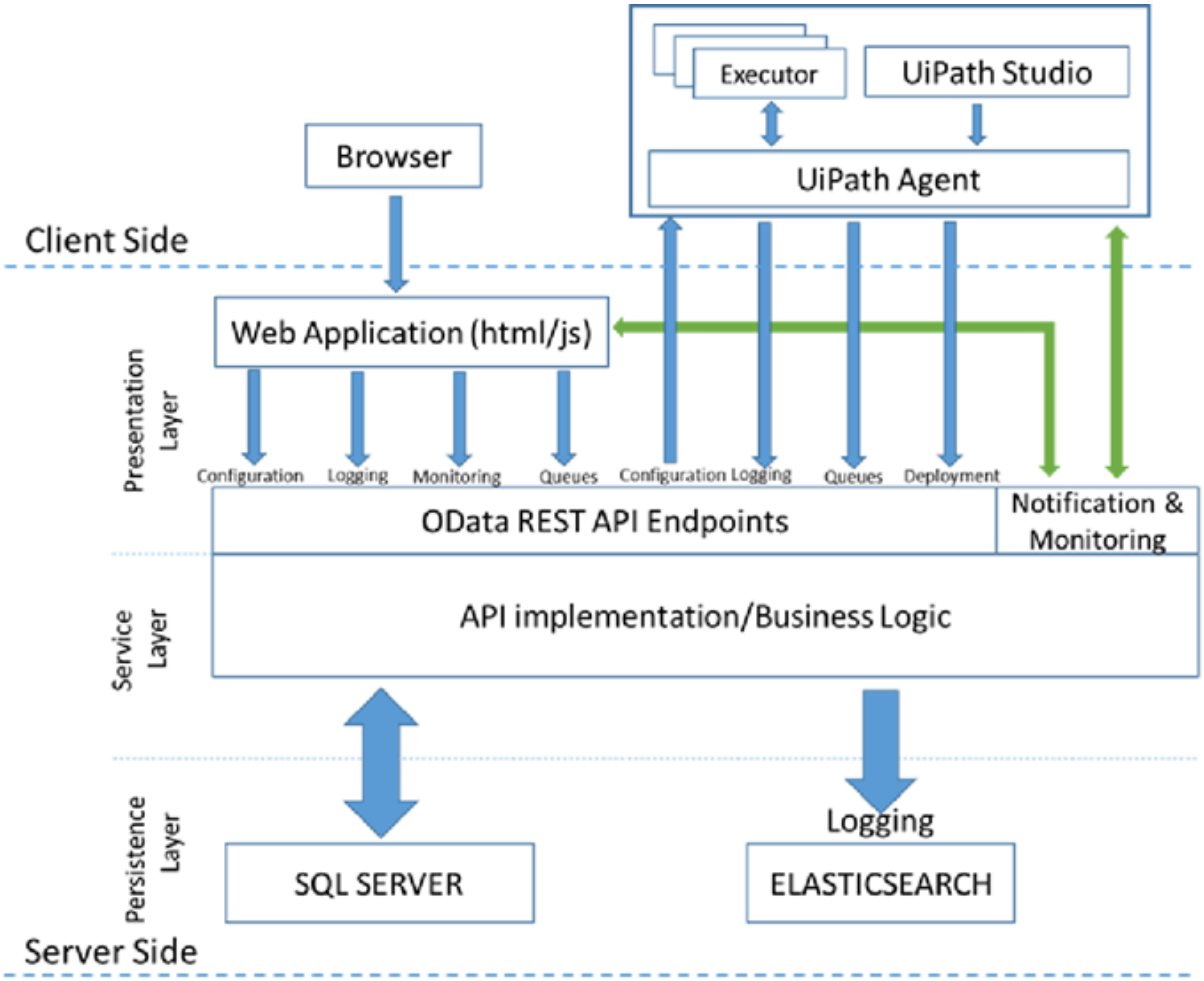


図 2.5　アーキテクチャ (レイヤー)

- プレゼンテーション（ユーザインタフェース）層
  - ウェブ・アプリケーション - ロボットとプロセスを管理するためのウェブ・インタフェース
- ウェブサービス層
  - REST API – エンドポイントは、ウェブ・アプリケーションとエージェントの両方によって使用される。エージェントとは、クライアント・コンピュータ上の 1 台以上のロボティック・スーパーバイザーである
  - 通知・モニタリング - サーバーやエージェントからの通知をハンドリング
  - ビジネスロジック – 各種サービス、API を実装
- 永続層
  - SQL サーバ - キューおよびキュー・アイテムを管理し、ロボット構成を保存する

  - インデックス・サーバ - Elasticsearch を使用してロボットがログ化したインデックス・データを保存する

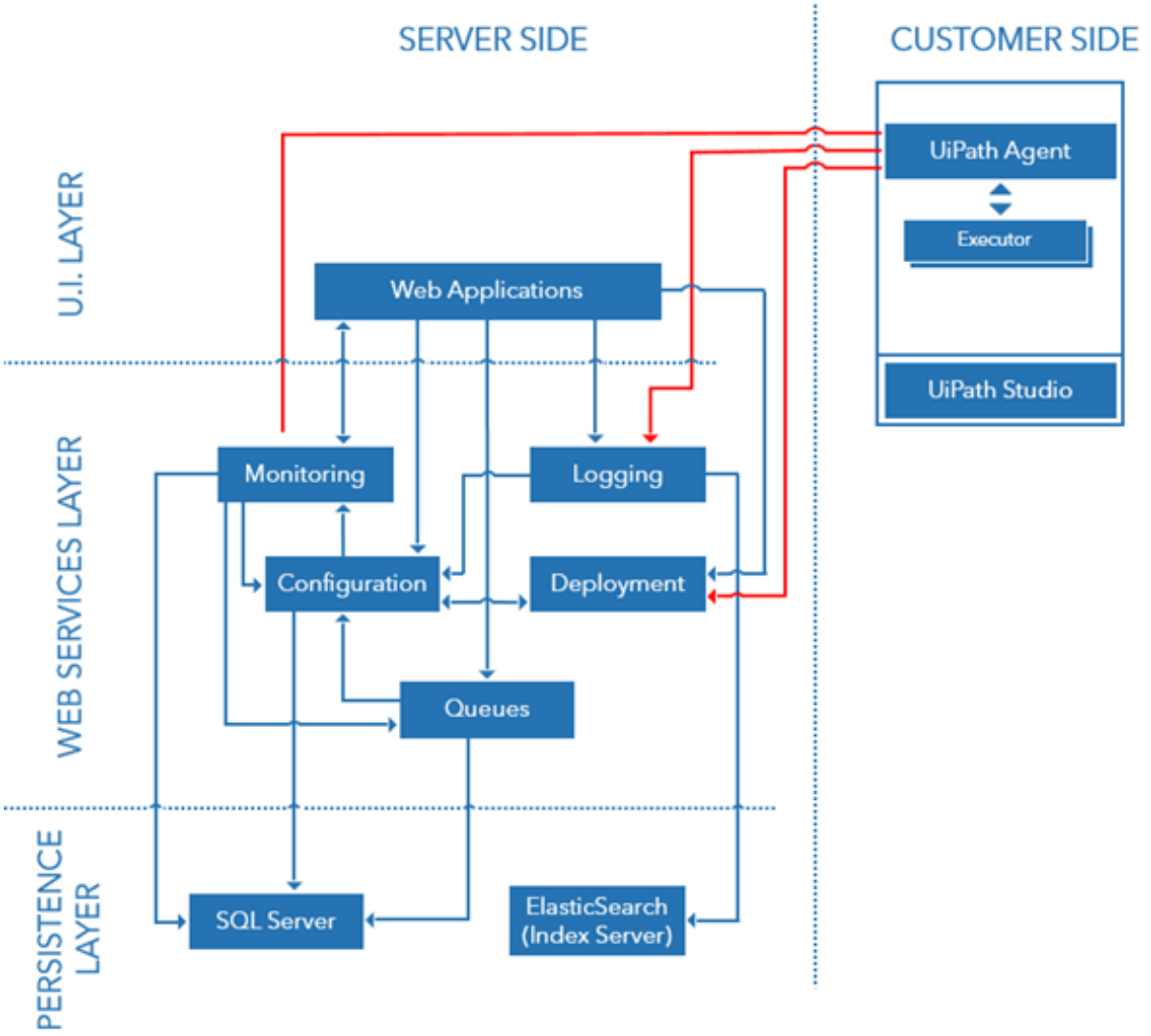


図 2.6　アーキテクチャ (関連)

REST API は、すべての Orchestrator の機能をカバーしている:

- Configuration (構成)]
  - アプリケーション・ユーザ、許可、ロボット、資産、リリース、および環境を定義、構成するのに使用される REST エンドポイント
- Monitoring (モニタリングと通知)
  - エージェントの登録、構成設定の提供、またサーバからやエージェントからの通知の送信/受信に使用される REST エンドポイント。通知 API は WebSocket コミュニケーションを使用する。
- Logging (ロギング)

  - エラー、ロボットによって送られた明白なメッセージ、その他の環境特定の情報などの様々な情報のログ化に使用される REST エンドポイント。

- Deployment (展開)
  - Orchestrator からの「スタート」コマンドの結果、開始する必要があるパッケージ・バージョンにクエリを送信するために、ロボットによって使用される REST エンドポイント。
- Queues (キュー)
  - キューおよびキューアイテム管理を担当する REST エンドポイント。キューへのデータ追加、キューからのトランザクション・アイテムの取得、トランザクション・アイテムのステータス設定など。

2.4.3 AD ドメイン、証明機関

- ドメイン名
- 証明機関
- ユーザ
  - DB で使用するユーザ
- Floating robot
- WSFC(Orchestrator/SQL Server 冗長構成時)
- DFS(NuGet ディレクトリ冗長構成時)

2.4.4 OS(Windows Server, Linux)

- Windows Server
  - Orchestrator
  - SQL Server
  - AD
- Linux
  - Redis
  - Elasticsearch/Kibana

2.4.5 SQL Server

SQL Server の設計ポイント

本番環境での運用では冗長構成での運用を推奨します。冗長構成 (Always On) を利用する時、SQLServer を動かすサービスアカウントはドメインユーザである必要があります。そのため、サービス実行用アカウントを用意する必要があります。また可用性 DB に対して作成されたユーザ権限では Orchestrator インストールができませんので、インストール用に別途ユーザを用意する必要があります。別途用意したユーザに必要な権限等は以下の URL を参照してください。https://orchestrator.uipath.com/lang-ja/docs/prerequisites-for-installation#section-sql-server

よって、作業の流れは WSFC のインストール、SQL Server のインストール、Orchestrator をインストールして DB を作成、Always On の設定、DB をバックアップ、可用性グループの作成、DB アクセス用のユーザ作成、Orchestrator の DB 接続パラメータの変更となります。

自動拡張設定

SQL Server ではトランザクションログが不足したときに自動的に拡張する機能が備わっています。自動拡張にしておくことによって、手動でのメンテナンスを行う必要がなくなります。本拡張サイズは比率 (デフォルトは 10%) で指定することができますが、拡張前のサイズが大きいと自動拡張するサイズが増え、拡張に時間がかかります。自動拡張中はデータベースへのトランザクション処理ができなくなり、Orchestrator での操作ができなくなります。長時間 Orchestrator が応答不能になることを避けるために以下を注意してください。

- 比率指定ではなく MB 単位指定で行う
- 定期的にトランザクションログの切り捨て (トランザクションログのバックアップ) を行い、自動拡張が行われないようにする

SQL サーバーの「自動拡張」および「自動圧縮」の設定に関する考慮事項 も合わせて確認してください。

その他の設定項目

SQL Server のその他の設定項目における考慮事項に関しては「Orchestrator 管理者のためのミドルウェア運用設定ガイド」 をご参照ください。

2.4.6 WSFC(Always on 使用時)

WSFC の設計ポイント SQLServer のリスナー用 IP を管理します。特にポイントはありません。

2.4.7 DFS(Orchestrator 冗長時)

DFS の設計ポイント冗長構成時における各ノード上のパッケージを同期します。Orchestrator のパッケージ情報のキャッシュファイルは同期する必要がありませんので、*.bin を除外するようにします。

2.4.8 IIS

IIS の設計ポイントアプリケーションプールのリサイクルタイミングを考慮する必要があります。デフォルトは 29 時間おきになっています。アセット、キューを使用するロボット実行時にリサイクルが実行されると

ジョブが失敗する可能性があります。そのため定期メンテナンスの時間帯にリサイクルまたは IIS の再起動を行います。

2.4.9 Redis(Orchestrator 冗長時)

Redis 上には必要な情報がキャッシュされています。参考： https://orchestrator.uipath.com/lang-ja/docs/prerequisites-for-installation#section-redis

Redis 上に保存されている情報は永続化する必要はありませんので、rdb、aof は不要となります。

2.4.10 Elasticsearch/Kibana(ロボット実行ログ保存が 2 百万件以上)

Elasticsearch/Kibana の設計ポイント大量のログデータを扱う場合は SQL Server の負荷低減のため、ログデータを Elasticsearch に保存することを推奨します。※ Logs テーブルに保存されるログが 2 百万件以上の場合は SQL Server の性能に影響が出る場合があります。以下の URL を参照してください。

https://orchestrator.uipath.com/lang-ja/docs/logging-configuration

Elasticsearch に保存されたデータは Kibana を使って可視化することができます。データはインデックスとよばれる単位で管理されており、デフォルトは 1 ヶ月単位になっています。蓄積されたログ情報を定期的に削除するため、メンテナンス用のツール (Curator) 利用を推奨します。また要件に応じて必要となるプラグインを検討します。

https://www.elastic.co/guide/en/elasticsearch/plugins/current/index.html

2.4.11 ネットワーク

論理構成

Orchestarator、Redis、Elasticsearch、kibana を冗長構成で利用している場合、振り分けに LoadBalancer を使用します。

Orchestrator、Elasticsearch、kibana については均等にアクセスが振り分けられるようにアルゴリズムを設定します。一般的なラウンドロビンのアルゴリズムで問題ありません。Redis に関しては健全性チェックを Sentinel サーバで行っていますので、どのノードがプライマリサーバか問い合わせを行って振り分けを行うように設計します。また、異常が発生したノードへの振り分けを停止するために、健全性チェックを定期的に行うようにします。

機能	健全性チェック方法
Orchestrator	https で ” GET /api/Status ” を送信し"200 OK ” が返されることを確認
Redis	tcp で ”auth PASSWORD\r\ninfo\r\nquit\r\n(パスワード認証なしの場合は auth 部分を取り除く) ” を送信し"role:master"が返されたノードへ振り分け
Elasticsearch	http で ” GET / ” を送信し"200 OK ” が返されることを確認
Kibana	http で ” GET / ” を送信し"200 OK ” が返されることを確認

IIS の接続タイムアウトについても規定値 120 秒から 20 秒に変更することを推奨します。理由は以下の通りです。

- ロボットから Orchestrator へは既定値で 30 秒ごとにハートビートが送信されます。一方 HTTP キープアライブにより既定値のタイムアウト 120 秒が発生する前に次のハートビートが送信されるため、ロボットからの TCP セッションは張りっぱなしになります。
- メンテナンス時などに複数台ある Orchestrator のうち 1 台停止した時、他の Orchestrator に接続が偏ります。次に停止した Orchestrator を起動しても、TCP セッションが他の Orchestrator に張りっぱなしとなるため、平準化されないという現象が発生します。
- 接続タイムアウトを 20 秒に変更すると、次のハートビートが来る前にセッションが一旦終了し、再度張りなおすため、この偏りを回避することができます。

Orchestrator については、2 つのコネクション (HTTP リクエストおよび WebSocket) を利用しますが、WebSocket に関してはセッションは張りっぱなしとなります。そのため障害発生時などで片系がダウンし WebSocket セッションの偏りが発生した後、片系が復旧してもセッションの偏りは復旧しませんが、WebSocket 通信で発生する負荷は軽微なものでありシステム全体への影響はありません。

ポート

各サーバ間に Firewall が存在する場合は、必要な通信ができるようポートが開いていることを確認します。詳細は以下のドキュメントを参照してください。

https://www.uipath.com/ja/resources/knowledge-base/orchestrator-network-requirements

2.5 Orchestrator 設計

2.5.1 web.config 設定

Orchestrator のカスタマイズを行うことができます。設計可能な項目は以下を参照してください。

https://orchestrator.uipath.com/lang-ja/docs/app-settings

またログの設定については以下を参照してください。

https://orchestrator.uipath.com/lang-ja/docs/logging-configuration

2.5.2 その他設定

Orchestrator インストール後、Web から設定を行う項目があります。詳細は以下を参照してください。

https://orchestrator.uipath.com/lang-ja/v2018.4/docs/about-tenant-settings

2.5.3 ライセンス登録

Orchestrator へのライセンスアップロード

ライセンス登録に必要なライセンスファイル生成には Robot もしくは Studio がインストールされた端末が必要となります。別途環境を用意してください。

https://orchestrator.uipath.com/lang-ja/docs/activating-and-uploading-your-license

高密度ロボットを利用する場合

高密度ロボットを利用する場合は、リモート デスクトップ ライセンスサーバへ必要な数の CAL(クライアントアクセスライセンス) をインストールする必要があります。

高密度ロボット用の Windows Server のセットアップについては以下を参照してください。

https://robot.uipath.com/lang-ja/docs/setting-up-windows-server-for-high-density-robots

リモート デスクトップ ライセンスサーバへの CAL のインストールについては以下を参照してください。

https://docs.microsoft.com/ja-jp/windows-server/remote/remote-desktop-services/rds-install-cals

第 3 章

運用設計

3.1 概要

事前に運用設計を行うことで Orchestrator を安全・効率的に運用することができます。運用で必要となる設計項目は主に以下になります。

- 管理項目: 何を管理するかを決定する
- 運用体制: どのような運用体制かを決定する
- 運用スケジュール: 定期作業や不定期作業を決定し、いつ行うかを決定する
- 監視設計: 監視項目、監視方法を決定する
- バックアップ設計: バックアップ方式、バックアップ対象、スケジュール等を決定する
- 障害設計: 障害発生時の作業フロー、エスカレーションルールを決定する

3.2 管理項目

Orchestrator を運用するにあたって何を管理対象にするかを設計しておきます。Orchestrator が正常に動作するためには以下を管理し、適切に運用する必要があります。

- サーバ
- 各種ミドルウェア
  - IIS
  - SQL Server
  - Redis
  - AD
  - Windows Server Failover Cluster(WSFC)
  - Network Load Balancer

- ネットワーク
- Orchestrator

また、ログ分析を行う場合は以下も対象にする必要があります。

- Elasticsearch
- Kibana

3.3 運用体制

運用を行う運用者及び管理者を決定しておきます。規模が大きい場合は基盤管理と Orchestrator を運用するチームを作って、それぞれのチームが連携しながら運用する体制を推奨します。また、必要によっては業務を行っている (ロボットを動かしている) 部門からの問い合わせを受ける窓口を設けることで作業分担を行いやすくなります。

3.3.1 基盤管理

OS、主要ミドルウェア (DB など)、ネットワークなどのインフラに関わる管理・運用を行うチームです。基盤管理チームではサーバ、各種ミドルウェアが正常に動作しているかを監視、および日々の定型作業を行い、障害発生に備えます。OS のセキュリティパッチ適用やミドルウェアのバックアップ、バージョンアップ等の作業もあります。

3.3.2 Orchestrator 管理

Orchestrator の運用に関わる管理・運用を行うチームです。Orchestrator 管理チームでは Orchestrator が正しく動作しているかを監視、および日々の定型作業を行い、障害発生に備えます。Orchestrator のデータのバックアップやバージョンアップ等の作業もあります。

3.4 運用スケジュール

3.4.1 定期作業

DB メンテナンス、バックアップ作業など定期的にメンテナンスを行うための時間を考慮、いつ行うかを設計しておく必要があります。特に DB メンテナンスに関しては、インデックス再構築を Orchestrator 実行中に行うとデッドロックが発生する場合があります。デッドロックの発生を防ぐためには DB アクセスを止める必要がありますので、メンテナンス中は Orchestrator(IIS) を停止することを推奨します。

以下、日次、週次、または月次など定期的に行うことを推奨する作業です。

- データのバックアップ
- ログの削除

- データベースメンテナンス
- 不要なパッケージの削除

以下、作業項目とスケジュール間隔、Orchestrator の停止有無の設計例になります。

作業項目	実行間隔	Orchestrator の停止有無
データバックアップ (完全)	毎週	不要
データバックアップ (差分)	毎日	不要
ログの削除	毎週	不要
データベースメンテナンス	毎週	必要
不要パッケージの削除	毎月	不要

具体的なメンテナンス項目・方法については **メンテナンス** を参照してください。

3.4.2 不定期作業

DB の拡張やデータのリストアなど、状況に応じてメンテナンスが発生します。事前に作業にどのぐらいの時間がかかるかを検証しておくことで、障害発生時の復旧にかかる時間の見積もりなどを行いやすくなります。

3.5 メンテナンス

3.5.1 概要

障害に対する予防や障害からの復旧を行うためには日ごろから適切なメンテナンスが必要です。予防や復旧には以下のようなものがあります。

- データベース肥大化に伴うパフォーマンス低下や障害の予防
- 不要なログやパッケージによるディスクの圧迫、ディスクフルへの予防
- ディスク障害、データベースの破損における速やかな復旧

このため、メンテナンスに必要な作業項目を洗い出し、作業手順を設計しておく必要があります。メンテナンスで必要な作業としては以下が挙げられます。

- データベースのバックアップ
- データベースのアーカイブと削除
- データベースメンテナンス
- Orchestrator のパッケージと設定ファイルのバックアップ
- Orchestrator の古いパッケージファイルの削除
- リカバリー手順

3.5.2 データベースのバックアップ

前提

データベースのバックアップの設計ではまず RPO(目標復旧地点) を決定することが重要になります。RPO を決定することで、復旧モデル、バックアップの種類 (フル・差分・トランザクションログ) が決定できます。頻繁にバックアップを取ることで、RPO を短くすることが可能です。一方で、運用の高度化、必要なディスクサイズが大きくなることがあります。

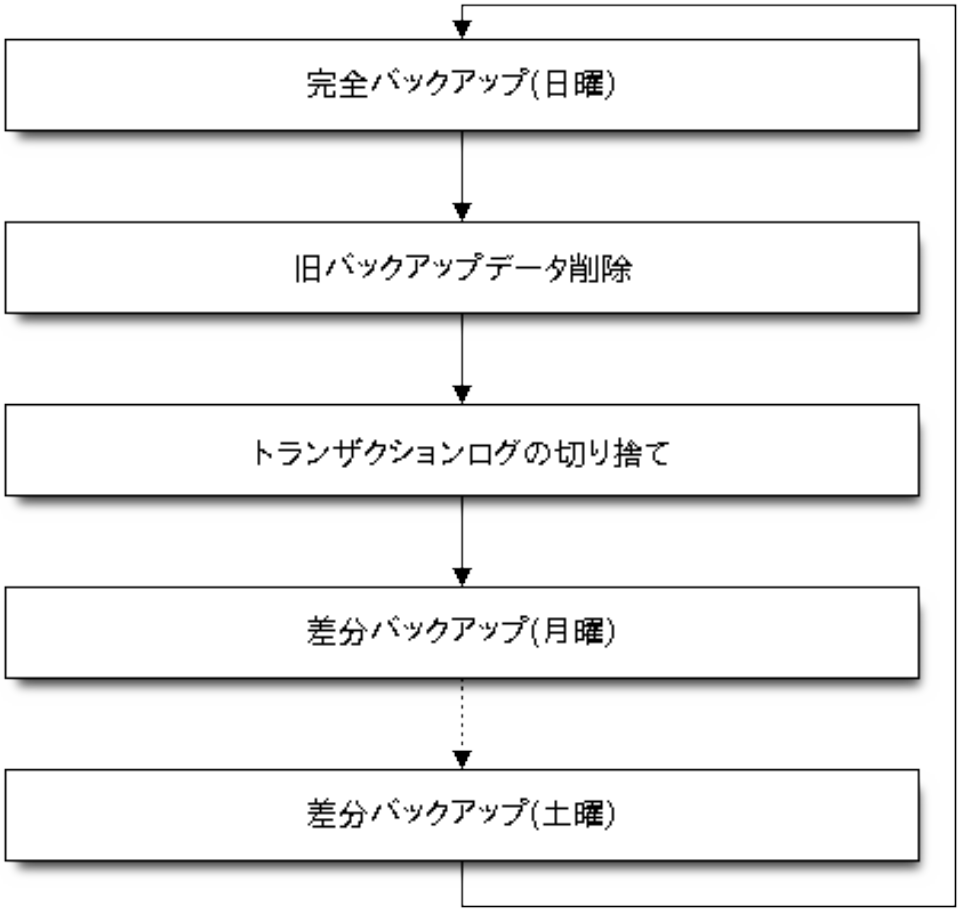
SQL Server

一般的に、特定の時点に復旧するためには

- 完全復旧モデル
- トランザクションログのバックアップ

が必要となります。復旧モデルとバックアップ詳細については https://docs.microsoft.com/ja-jp/sql/relational-databases/backup-restore/back-up-and-restore-of-sql-server-databases?view=sql-server-2017 を参照してください。

週次で完全バックアップを取り、毎日バックアップを取るような場合には以下のような手順になります。



Elasticsearch

> **警告:** Index データファイルを直接コピーしてバックアップした場合、正しくリストアすることはできません。必ず本資料に従って Curator または API 経由でバックアップを行ってください。

Curator のインストール

下記サイトからインストーラをダウンロードしてください。

https://www.elastic.co/guide/en/elasticsearch/client/curator/5.6/installation.html

利用する OS に応じた方法でダウンロード/インストールを行ってください。本ドキュメントでは *curator* コマンドのパスが通っていることを前提にして進めますので、事前にパスを通してください。

Curator の設定

Curator を実行するユーザの *%HOMEPATH%/.curator/curator.yml* を以下の通り作成します。ホスト名やポート番号、SSL 設定については環境に合わせて変更してください。

```
client:
  hosts:
    - "Elasticsearch ホスト名"
  port: "9200"
  use_ssl: False
  ssl_no_validate: False
  timeout: 30

logging:
  loglevel: INFO
```

メンテナンス運用設計

本書が想定している Elasticsearch 運用の例を下図に示します。参照可能なインデックスは直近のもののみとし、一定期間を経過したインデックスは随時削除します。参照できなくなった過去のインデックスは、定期バックアップ時のスナップショットとして保管され、一定期間は復旧可能とします。スナップショットは日次で取得します。当月以外のスナップショットは月初に取得した最終スナップショット（＝月次スナップショット）のみを残して削除します。また、保管期間を経過した月次スナップショットは順次廃棄していきます。

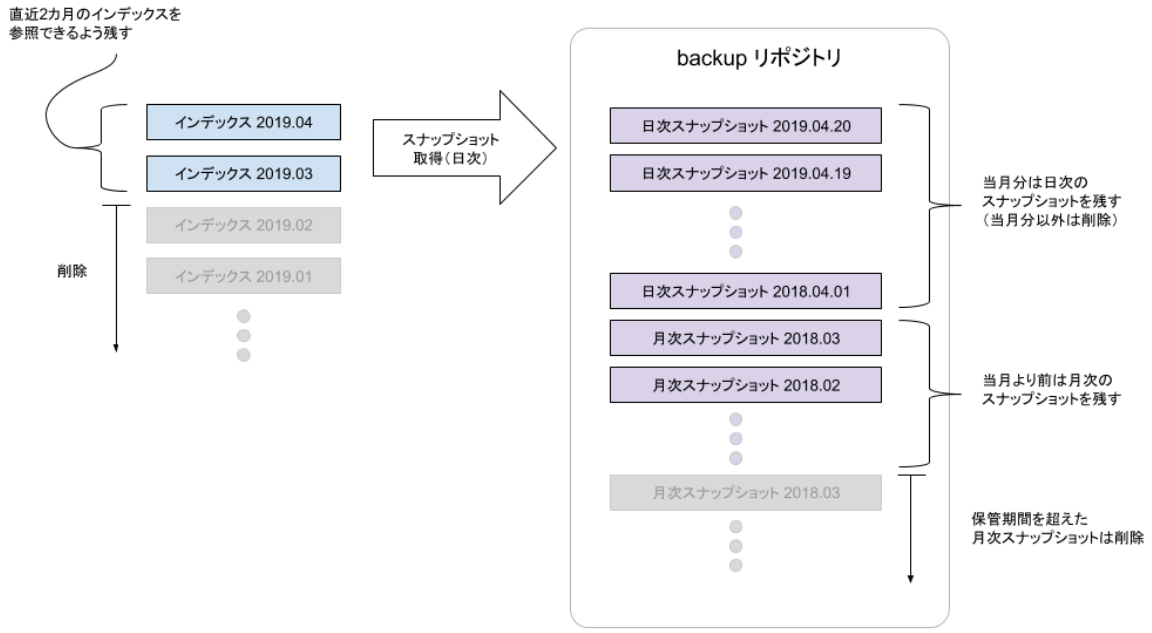


図 3.1　Elasticsearch 運用例

リポジトリの作成

Elsaticsearch では、スナップショットを利用するために事前にリポジトリと呼ばれる領域が必要になります。本作業はスナップショットの運用を開始する前に一度だけ実施してください。

Curator の場合

以下のコマンドを実行してリポジトリを作成します。

```
es_repo_mgr.exe create fs --repository リポジトリ名 --location リポジトリ格納パス
```

例えば backup という名前のリポジトリを作成する場合は以下のように実行します。

```
es_repo_mgr.exe create fs --repository backup --location backup
```

リポジトリ格納パスは絶対パスで指定することもできますが、Elasticearch で設定している *path.repo* を基準に相対パスで指定してください。上記コマンドが実行されると格納パスの配下にリポジトリ名のフォルダが作成されます。

API の場合

Curator を利用しない場合は Kibana の Dev Tools 等で直接 Elasticsearch の API を以下のように呼び出します。

```
PUT _snapshot/リポジトリ名
{
  "type": "fs",
```

(次のページに続く)

(前のページからの続き)

```
  "settings": {
    "location": "リポジトリ格納パス"
  }
}
```

例えば backup という名前のリポジトリを作成する場合は以下のように実行します。

```
PUT _snapshot/backup
{
  "type": "fs",
  "settings": {
    "location": "backup"
  }
}
```

月次スナップショット整理

月初にスナップショットの作成のために一度だけ実施します。

Curator の場合

テキストエディタで以下の内容のファイルを作成します。ファイルの拡張子は .yml としてください。

```
actions:
  1:
    action: snapshot
    description: "Create snapshot of previous month"
    options:
      repository: "リポジトリ名"
      name: '<monthly-{now/M-1M{YYYY.MM}}>'
      continue_if_exception: False
      wait_for_completion: True
    filters:
      - filtertype: pattern
        kind: prefix
        value: "インデックス名のプレフィクス"
      - filtertype: period
        period_type: relative
        source: name
        timestring: "%Y.%m"
        range_from: -1
        range_to: -1
        unit: months
  2:
    action: delete_snapshots
    description: "Delete daily snapshot of previous month"
    options:
      repository: "リポジトリ名"
      ignore_empty_list: True
```

(次のページに続く)

(前のページからの続き)

```
      continue_if_exception: False
    filters:
      - filtertype: pattern
        kind: prefix
        value: "daily-"
      - filtertype: period
        period_type: relative
        source: name
        range_from: -1
        range_to: -1
        timestring: "%Y.%m"
        unit: months
```

例えば以下のように作成します。

```
actions:
  1:
    action: snapshot
    description: "Create snapshot of previous month"
    options:
      repository: "backup"
      name: '<monthly-{now/M-1M{YYYY.MM}}>'
      continue_if_exception: False
      wait_for_completion: True
    filters:
      - filtertype: pattern
        kind: prefix
        value: "default-"
      - filtertype: period
        period_type: relative
        source: name
        timestring: "%Y.%m"
        range_from: -1
        range_to: -1
        unit: months
  2:
    action: delete_snapshots
    description: "Delete daily snapshot of previous month"
    options:
      repository: "backup"
      ignore_empty_list: True
      continue_if_exception: False
    filters:
      - filtertype: pattern
        kind: prefix
        value: "daily-"
      - filtertype: period
        period_type: relative
        source: name
        range_from: -1
        range_to: -1
```

(次のページに続く)

(前のページからの続き)

```
        timestring: "%Y.%m"
        unit: months
```

ファイルを作成したら以下のコマンドを実行します。

```
curator.exe 作成したファイルのパス
```

以上で月次スナップショット整理は終了です。

API の場合

Curator を利用しない場合は Kibana の Dev Tools 等で直接 Elasticsearch の API を利用します。まず、以下の API で先月分の月次スナップショットを作成します。

```
PUT _snapshot/backup/%3Cmonthly-%7Bnow%2FM-1M%7BYYYY.MM%7D%7D%3E?wait_for_
↪completion=true
{
  "indices": "先月分のインデックス名",
  "ignore_unavailable": true,
  "include_global_state": false
}
```

例えば、2019 年 3 月の月初に実施する場合は以下のように指定します。

```
PUT _snapshot/backup/%3Cmonthly-%7Bnow%2FM-1M%7BYYYY.MM%7D%7D%3E?wait_for_
↪completion=true
{
  "indices": "default-2019.02*",
  "ignore_unavailable": true,
  "include_global_state": false
}
```

月次スナップショットを作成した後は不要な日次スナップショットを削除します。まず、削除対象のスナップショットを調べるために以下の API を利用します。赤字の YYYY.MM 部分は先月分のスナップショットに合致するように指定します。

```
GET _snapshot/backup/daily-YYYY.MM.*?filter_path=snapshots.snapshot
```

例えば、2019 年 3 月の月初に実施する場合は以下のように指定すれば 2019 年 2 月の日次スナップショットの一覧が得られます。

```
GET _snapshot/backup/daily-2019.02.*?filter_path=snapshots.snapshot
```

以下は出力例です。

```
{
  "snapshots" : [
    {
```

(次のページに続く)

(前のページからの続き)

```
      "snapshot" : "daily-2019.02.01"
    },
    {
      "snapshot" : "daily-2019.02.02"
    },
    ...(略)...
  ]
}
```

一覧表示されたそれぞれのスナップショットについて以下の API で削除します。複数のスナップショットを一括で削除はできないため、赤字の YYYY.MM.dd を日付を指定して繰り返し実行する必要があります。

```
DELETE _snapshot/backup/daily-YYYY.MM.dd
```

例えば、2019 年 2 月 3 日に取得した日次スナップショットを削除する場合は以下のように指定します。

```
DELETE _snapshot/backup/daily-2019.02.03
```

以上で月次スナップショット整理は終了です。

日次スナップショット取得

日次スナップショットを作成するために毎日実施します。

Curator の場合

テキストエディタで以下の内容のファイルを作成します。ファイルの拡張子は .yml としてください。

```
actions:
  1:
    action: snapshot
    description: "Create daily snapshot of this month"
    options:
      repository: "リポジトリ名"
      name: 'daily-%Y.%m.%d'
      continue_if_exception: False
      wait_for_completion: True
    filters:
      - filtertype: pattern
        kind: prefix
        value: "インデックス名のプレフィクス"
      - filtertype: period
        period_type: relative
        source: name
        timestring: "%Y.%m"
        range_from: 0
        range_to: 0
        unit: months
```

例えば以下のように作成します。

```
actions:
  1:
    action: snapshot
    description: "Create daily snapshot of this month"
    options:
      repository: "backup"
      name: 'daily-%Y.%m.%d'
      continue_if_exception: False
      wait_for_completion: True
    filters:
      - filtertype: pattern
        kind: prefix
        value: "default-"
      - filtertype: period
        period_type: relative
        source: name
        timestring: "%Y.%m"
        range_from: 0
        range_to: 0
        unit: months
```

ファイルを作成したら以下のコマンドを実行します。

```
curator.exe 作成したファイルのパス
```

以上で日次スナップショットの取得は終了です。

API の場合

Curator を利用しない場合は Kibana の Dev Tools 等で直接 Elasticsearch の以下の API を利用します。

```
PUT _snapshot/backup/%3Cdaily-%7Bnow%2Fd%7BYYYY.MM.dd%7D%7D%3E?wait_for_
↪completion=true
{
  "indices": "今月分のインデックス名",
  "ignore_unavailable": true,
  "include_global_state": false
}
```

例えば、2019 年 3 月に実施する場合は以下のように指定します。

```
PUT _snapshot/backup/%3Cdaily-%7Bnow%2Fd%7BYYYY.MM.dd%7D%7D%3E?wait_for_
↪completion=true
{
  "indices": "default-2019.03*",
  "ignore_unavailable": true,
  "include_global_state": false
}
```

以上で日次スナップショットの取得は終了です。

3.5.3 データベースのアーカイブと削除

SQL Server

SQL Server のデータサイズの肥大化を防ぐためには以下の作業を行う必要があります。

- アーカイブ用 DB の作成とデータの移行
- 古いレコードの定期的な削除

データが肥大化しやすく、定期的な確認・削除が必要な項目には以下のようなものがあります。

テーブル名	特徴
Logs	プロセスの実行ごとに開始、終了のレコードが生成されます。また、ワークフロー中に *Log Message* アクティビティを利用してログを出力する度に生成されます。Orchestrator から削除はできないため、過去の不要なログを定期的に削除が必要となります。
QueueItems	*Add Queue* アクティビティによりレコードが生成されます。Orchestrator から Queue を削除しても論理削除 (Deleted=1) のみでレコードが残り続けるため、不要な Queue を定期的に削除が必要となります。
AuditLogs	管理者による設定変更や、ジョブ実行など管理オブジェクトの状態変化によりレコードが生成されます。監査の観点より基本的には削除しないことを推奨します。

アーカイブ用 DB の作成とデータの移行

古いレコードを削除する前に保存したいテーブルのみを持つデータベースを作成することを推奨します。別にデータベースを作成することで、監査などの理由で保存する必要がある項目のアーカイブとして機能します。

例えば、*UiPathArchive* という新しいデータベースをアーカイブ用として運用にするには以下のように行います。

1. *UiPathArchive* という新しいデータベースを作成します。

```
create database UiPathArchive
```

2. *Logs* と同じ構造を持つ *ArchiveLogs* テーブルを作成します。

```
select * into [UiPathArchive].[dbo].[ArchiveLogs] from [UiPath].[dbo].[Logs] where␣
↪1=2
```

3. 同様に *QueueItems* のテーブルをアーカイブ用に作成します。

```
select * into [UiPathArchive].[dbo].[ArchiveQueueItems] from [UiPath].[dbo].
↪[QueueItems] where 1=2
```

4. データを削除する前に対応するアーカイブテーブルにデータをコピーします。

古いレコードの定期的な削除

Logs テーブルの削除

45 日以上経過したログメッセージは、削除することを推奨します。次のクエリ例は、「Info」レベルで 45 日以上経過した古いメッセージを削除します。オプションで、TenantId を含めるか、行 *and level = 2* をコメントアウトすることにより、レベルに関係なく、ログメッセージをすべて削除することができます。

SQL サーバーでクエリを手動で実行するか、スケジュールで実行することができます。

```
DELETE FROM [UiPath].[dbo].[Logs]
/*  level: 0 = Verbose, 1 = Trace, 2 = Info,  3 = Warn, 4 = Error, 5 = Fatal  */
where 1=1
and level = 2
-- and TenantId = 1 -- default tenant
and  DateDiff(day, TimeStamp, GetDate()) > 45
```

ヒント: ログの保存日数は過去ログを参照したい期間とプロセスが出力するログ数により見積もりが必要です。ログの表示性能、データベースの肥大化を防ぐための目安としては保存する件数は 200 万件程度に収めることを推奨します。どれくらい前までの古いアイテムを削除するかを変更する場合には、最後の行の「45」を目的の日数に変更します。

警告: Logs テーブルのレコードを削除するとインデックスが断片化し、ログ表示のパフォーマンスが低下します。レコード削除後は必ずインデックス再構築を行ってください。インデックスの再構築については **データベースメンテナンス** を参照してください。

QueueItems テーブルの削除

処理済みで 45 日より古いキューアイテム (*status* = 3) を削除するには、次のようなクエリを使用します。オプションで、TenantId と ReviewStatus を含めることができます。

SQL サーバーでクエリを手動で実行するか、スケジュールで実行することができます。

```
DELETE FROM [UiPath].[dbo].[QueueItems]
/* 0 = new, 1 = in progress, 2 = failed, 3 = success, 4 = invalid, 5 = retried */
where status = 3
--and ReviewStatus != 0
--and TenantId = 1 -- default tenant
and  DateDiff(day, CreationTime, GetDate()) > 45
```

ヒント: どれくらい前までの古いアイテムを削除するかを変更する場合には、最後の行の「45」を目的の日数に変更します。

Elasticsearch

不要なインデックスのクローズ

インデックスをオープンしていると Elasticsearch はデータをメモリに保持し続けるため、Java ヒープの枯渇につながります。月の切り替わりなどで、参照しなくなった古いインデックスについてはクローズを行ってください。

テキストエディタで以下の内容のファイルを作成します。ファイルの拡張子は .yml としてください。なお日数にはログを保持したい月数× 31 を指定してください。インデックスの単位を日単位にしている場合でも削除の単位が月単位であればそのまま利用できますが、任意の日数単位で削除したい場合は *timestring:* を *'%Y.%m.%d'* に変更する必要があります。

```
actions:
  1:
    action: close
    description: "Close indices"
    options:
      ignore_empty_list: True
      continue_if_exception: False
    filters:
      - filtertype: pattern
        kind: prefix
        value: "インデックス名のプレフィックス"
      - filtertype: age
        source: name
        direction: older
        timestring: '%Y.%m'
        unit: days
        unit_count: 日数
```

例えば 2 カ月より前のインデックスをクローズする場合は以下のように作成します。このファイルで 2019 年 1 月に実行した場合は、 *default-2019.01* と *default-2019.12* のインデックスを残し、それより古いインデックスが削除されます。

```
actions:
  1:
    action: close
    description: "Close indices"
    options:
      ignore_empty_list: True
      continue_if_exception: False
    filters:
      - filtertype: pattern
        kind: prefix
        value: "default-"
      - filtertype: age
        source: name
        direction: older
        timestring: '%Y.%m'
```

(次のページに続く)

(前のページからの続き)

```
        unit: days
        unit_count: 62
```

ファイルを作成したら以下のコマンドを実行します。

```
curator.exe 作成したファイルのパス
```

以上でインデックスのクローズは終了です。

古いインデックスの削除

インデックスをクローズしてもインデックスを残し続けるとディスクの枯渇につながります。不要なインデックスの削除を定期的に行ってください。

Curator の場合

テキストエディタで以下の内容のファイルを作成します。ファイルの拡張子は .yml としてください。なお日数にはログを保持したい月数× 31 を指定してください。インデックスの単位を日単位にしている場合でも削除の単位が月単位であればそのまま利用できますが、任意の日数単位で削除したい場合は *timestring:* を *'%Y.%m.%d'* に変更する必要があります。

```
actions:
  1:
    action: delete_indices
    description: "Delete indices"
    options:
      ignore_empty_list: True
      continue_if_exception: False
    filters:
      - filtertype: pattern
        kind: prefix
        value: "インデックス名のプレフィックス"
      - filtertype: age
        source: name
        direction: older
        timestring: '%Y.%m'
        unit: days
        unit_count: 日数
```

例えば 2 カ月より前のインデックスを削除する場合は以下のように作成します。このファイルで 2019 年 1 月に実行した場合は、 *default-2019.01* と *default-2019.12* のインデックスを残し、それより古いインデックスが削除されます。

```
actions:
  1:
    action: delete_indices
    description: "Delete indices"
    options:
```

(次のページに続く)

(前のページからの続き)

```
      ignore_empty_list: True
      continue_if_exception: False
    filters:
      - filtertype: pattern
        kind: prefix
        value: "default-"
      - filtertype: age
        source: name
        direction: older
        timestring: '%Y.%m'
        unit: days
        unit_count: 62
```

ファイルを作成したら以下のコマンドを実行します。

```
curator.exe 作成したファイルのパス
```

以上でインデックスの削除は終了です。

API の場合

Curator を利用しない場合は Kibana の Dev Tools 等で直接 Elasticsearch の API を利用します。まず、以下の API でインデックスの一覧を取得します。

```
GET _cat/indices/default*?s=index&h=index
```

以下は出力例です。

```
default-2018.12
default-2019.01
default-2019.02
```

不要なインデックスがあれば以下の API でインデックスを削除します。インデックス名はカンマ区切りやワイルドカード (*) で複数指定することができます。

```
DELETE インデックス名
```

例えば default-2018.12 と default-2019.01 の 2 つのインデックスを削除する場合は以下のように指定できます。

```
DELETE default-2018.12,default-2019.01
```

default-2019 で始まるインデックスをすべて削除する場合は以下のように指定できます。

```
DELETE default-2019*
```

インデックスを削除したら再度一覧を表示して、削除したインデックスが表示されなくなっていることを確認してください。

```
GET _cat/indices/default*?s=index&h=index
```

以上でインデックスの削除は終了です。

3.5.4 データベースメンテナンス

SQL Server

SQL Server のメンテナンスプランを作成して、定期的に実行することでデータベースの健全性を保つことに役立ちます。

タスク	期待される効果
データベースの整合性確認	データベース内のすべてのオブジェクトの割り当てと構造上の整合性をチェックし、データ破損を早期に検出
インデックスの再構築	インデックスの断片化を解消することにより、I/O 負荷を軽減し、パフォーマンスを向上
統計の更新	統計情報を最新の状態にすることで、現在のデータ分布に最適な実行プランを選択インデックスの再構築で自動的に統計情報も更新されるため、インデックスの再構築実施時には不要

警告: メンテナンスプランの実行時は Orchestrator を停止してから行ってください。インデックスの再構築中に Orchestrator からデータベースへのアクセスが行われると操作に失敗することがあります。また、CPU が 16 コア以上のサーバで SQL Server を動作させる場合、自動的に Lock Partioning が有効になり、デッドロックによる失敗が発生しやすくなるため必ず Orchestrator を停止してください。

SQL Server Management Studio のウィザードを使ったメンテナンスプランの作成は次の手順になります。画面は SQL Server 2012 のものになります。バージョンによって一部差異がありますので、ご利用されるバージョンのマニュアルも合わせて参照してください。

注釈: メンテナンスプランの作成・実行には [SQL Server エージェント] サービスが実行中である必要があります。実行されていない場合は [SQL 構成マネージャ] からサービスの開始をおこなってください。

---

1. SQL Server Management Studio を開いて UiPath のデータベースが格納されているインスタンスに接続します。

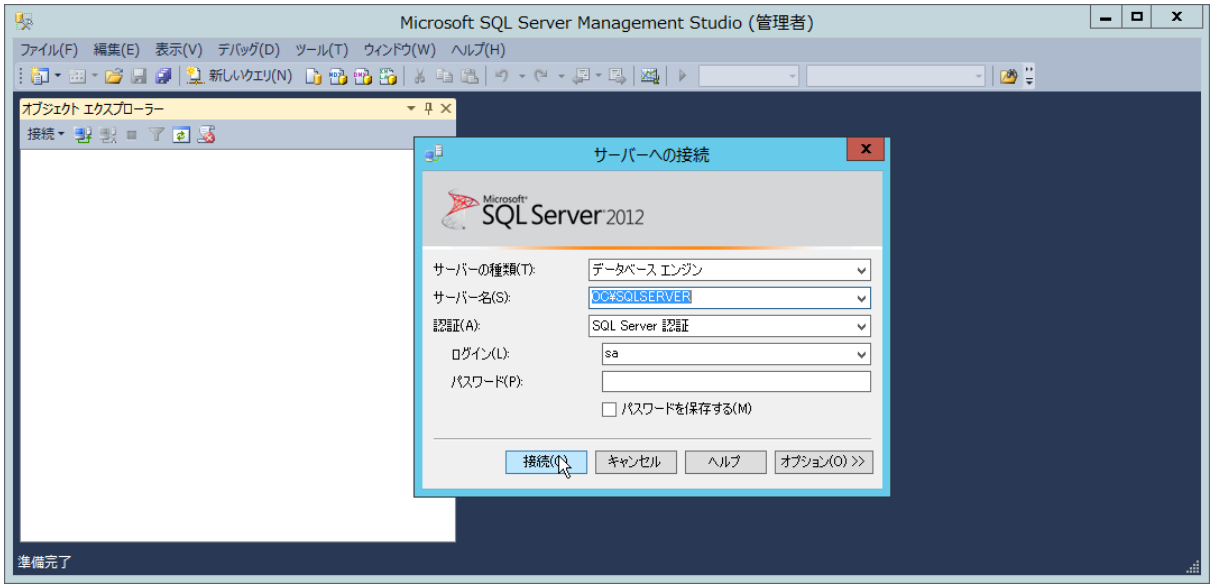


図 3.2　接続画面

2. 左のツリーペインから *[**管理**]* -> *[**メンテナンスプラン**]* を選択して、右クリックメニューから *[**メンテナンス プラン ウィザード**]* を選択します。

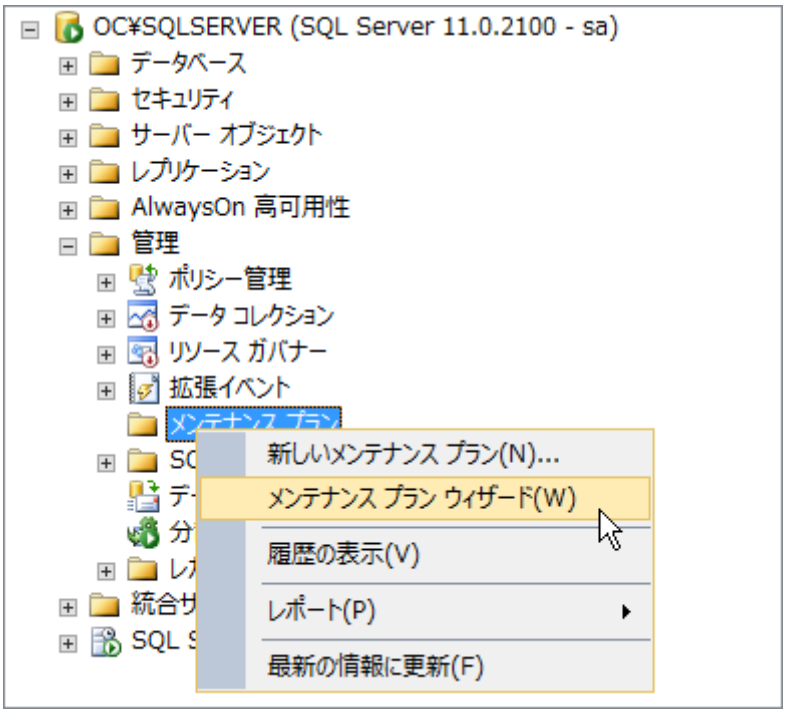


図 3.3　メンテナンス プラン ウィザード選択

3. メンテナンスプランウィザードが開きますので、 *[次へ]* を押下します。

図 3.4　SQL Server メンテナンス プラン ウィザード

4. メンテナンスプランの名前とスケジュールを設定します。

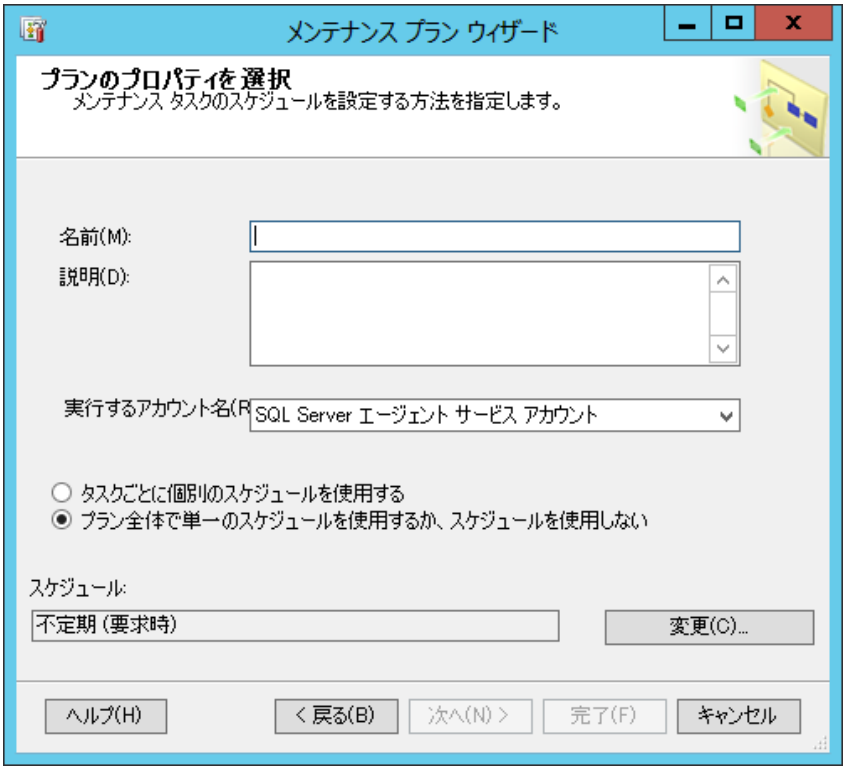


図 3.5　名前設定

5. スケジュールは毎週や毎月などを実施したいタイミングを選択します。

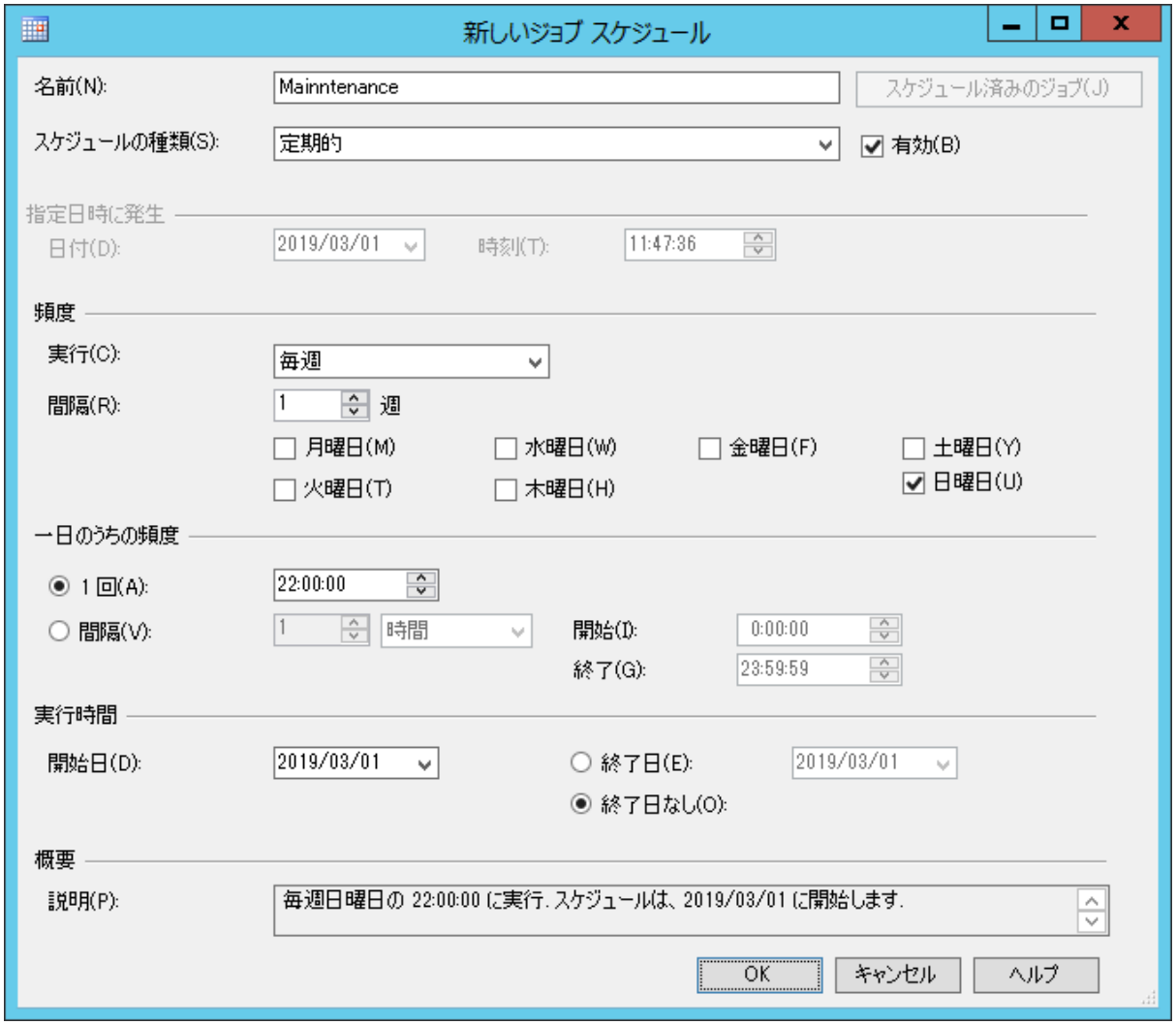


図 3.6　スケジュール設定

警告: スケジュールによる自動実行を行う場合は Windows のタスク機能などを利用して、Orchestrator が稼働しているホスト上で事前に IIS のサービスも停止するようにスケジュールしてください。

6. メンテナンスプランで実行するタスクを選択します。ここでは、[データベースの整合性確認] と [インデックスの再構築] を選択します。

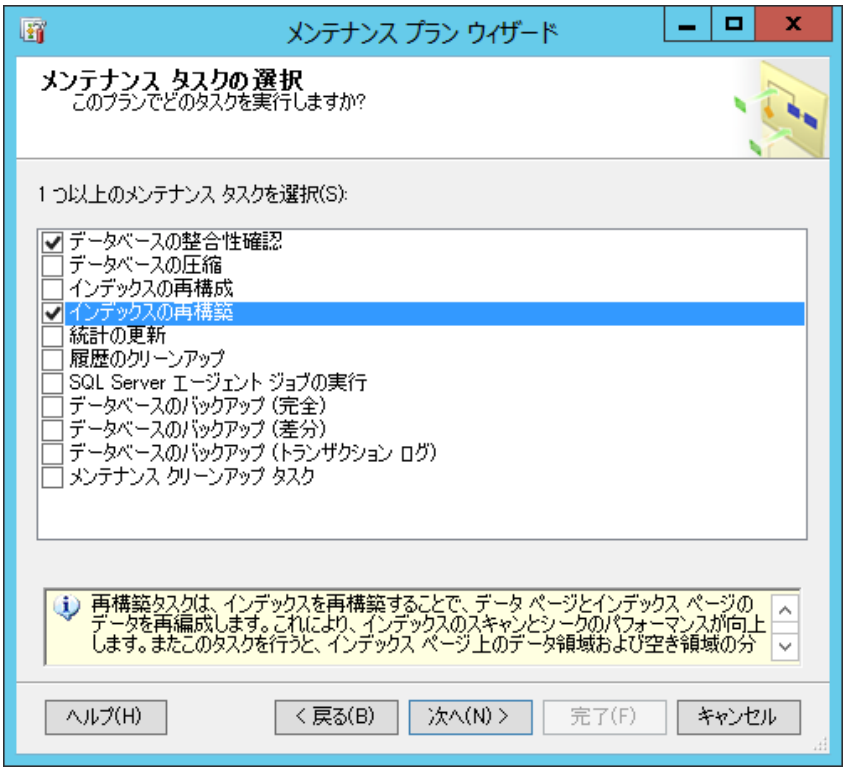


図 3.7　タスク選択

7. メンテナンスプランのタスク実行順を決定します。そのまま ***[次へ]*** を押下します。

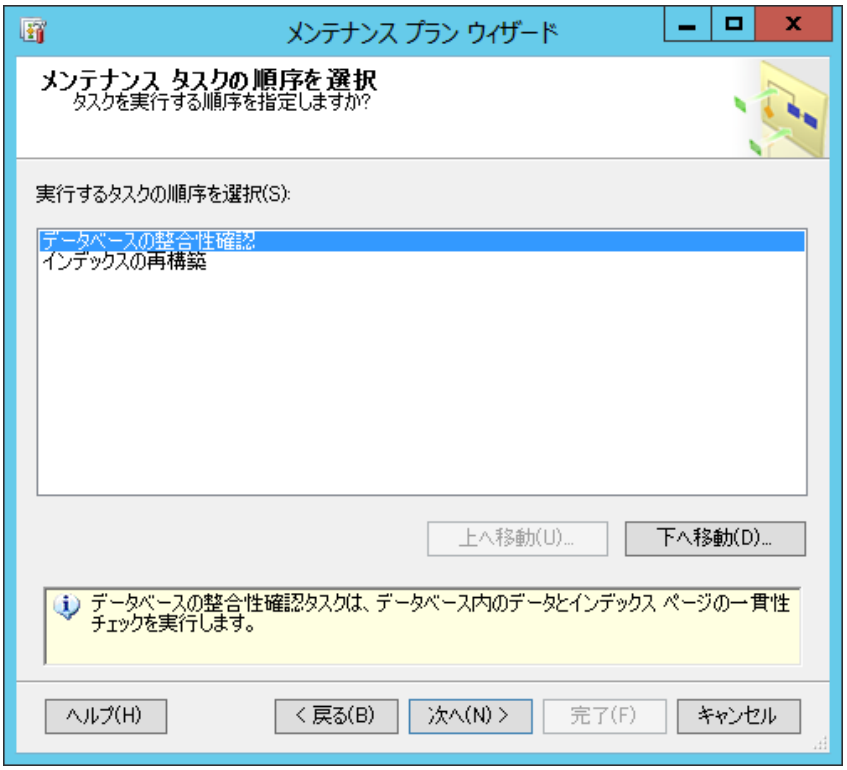


図 3.8　順序選択

8. データベースの整合性確認タスクを実行するデータベースを選択します。

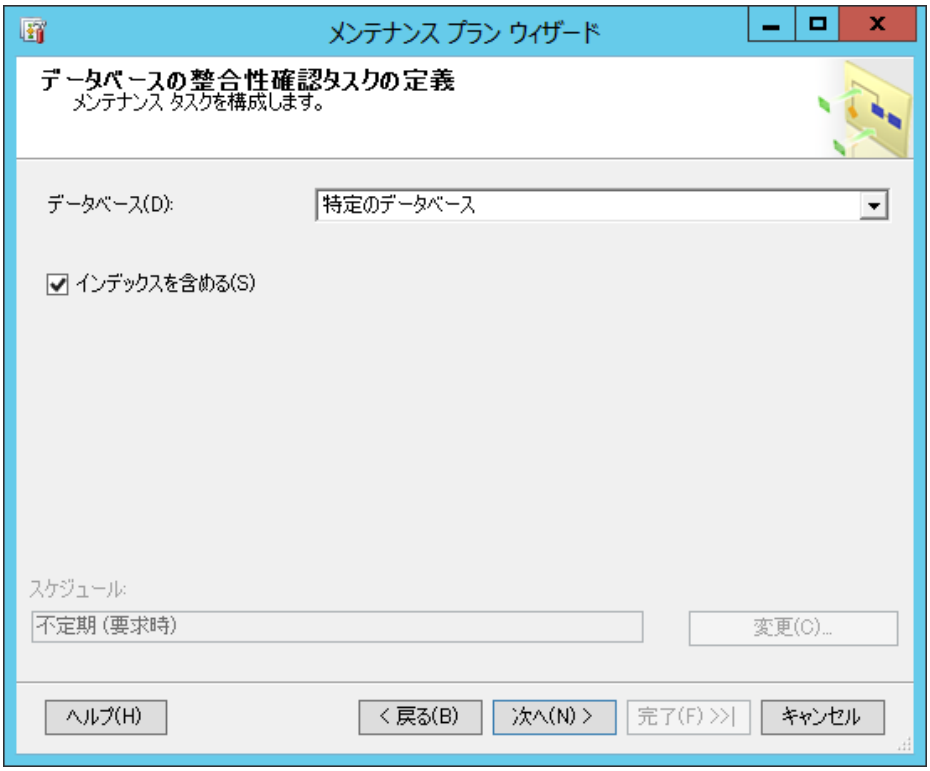


図 3.9　データベース選択

9. ここでは *[UiPath]* を選択します。(Orchestrator で利用するデータベース名を変更している場合は本設定の対象も合わせて変更してください)

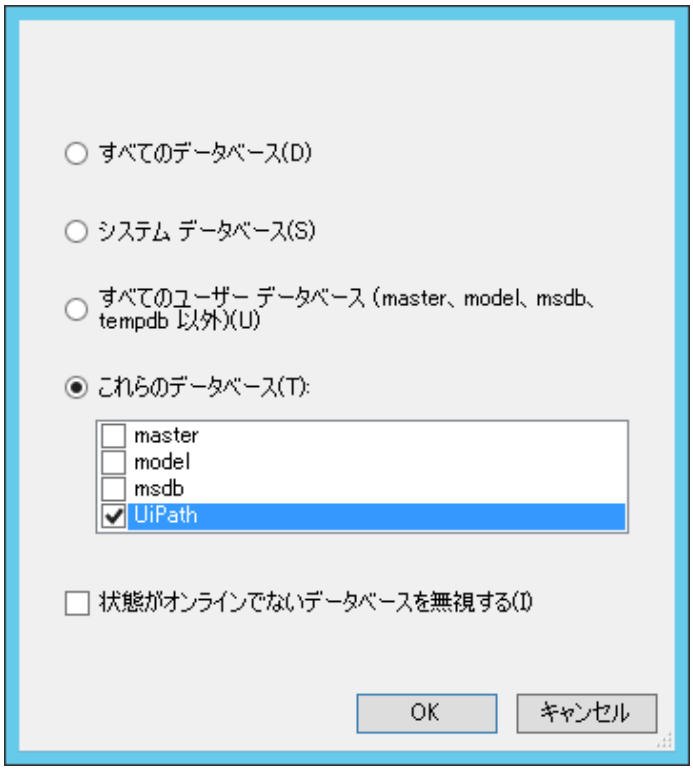


図 3.10　データベース選択

10. インデックスの再構築タスクを実行するデータベースも同様に選択します。

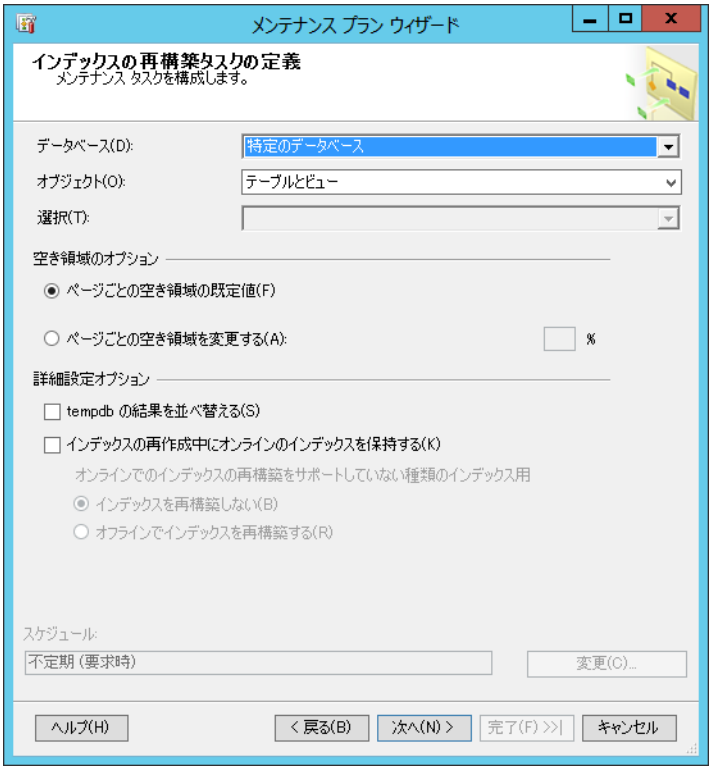


図 3.11　データベース選択

11. レポートの出力先を選択します。

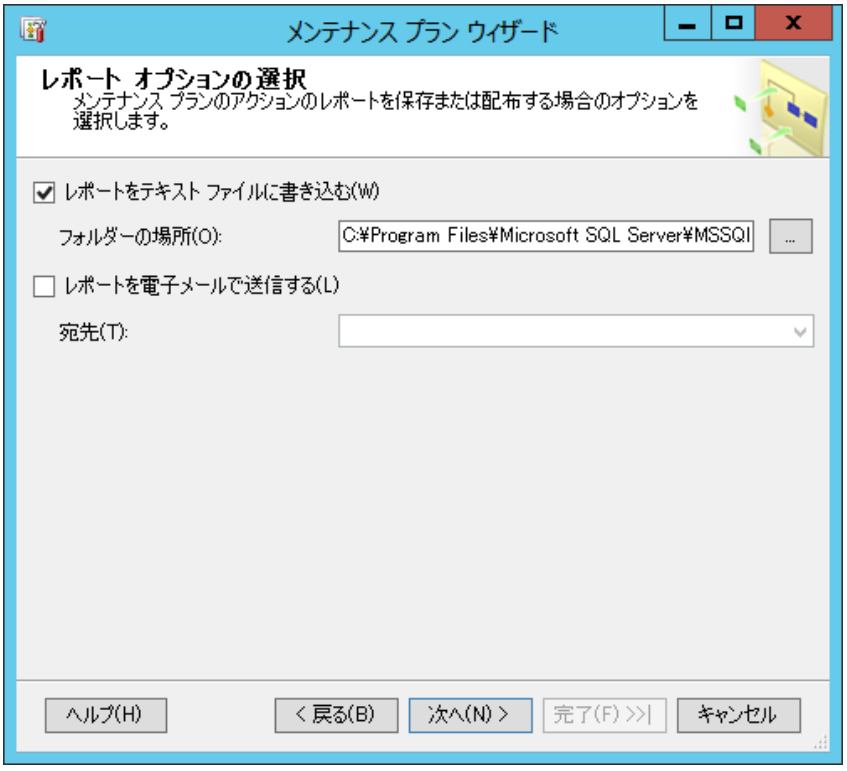


図 3.12　レポート出力先設定

12. 設定内容を確認し、問題なければ *[**完了**]* を押下します。

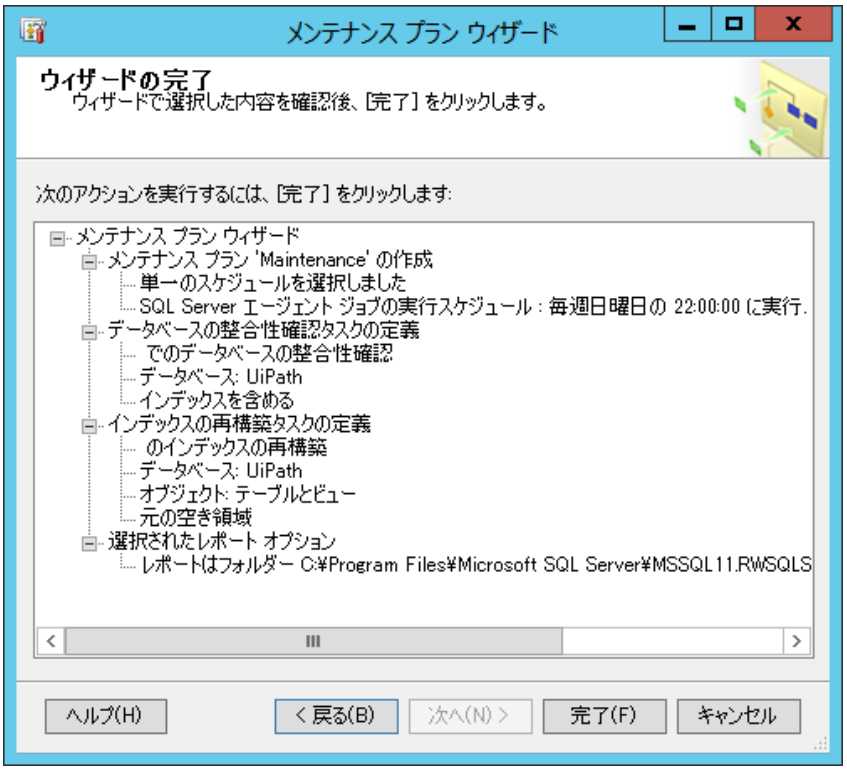


図 3.13　確認画面

13. メンテナンスプランの作成が行われますので、作成が成功したらダイアログを閉じます。

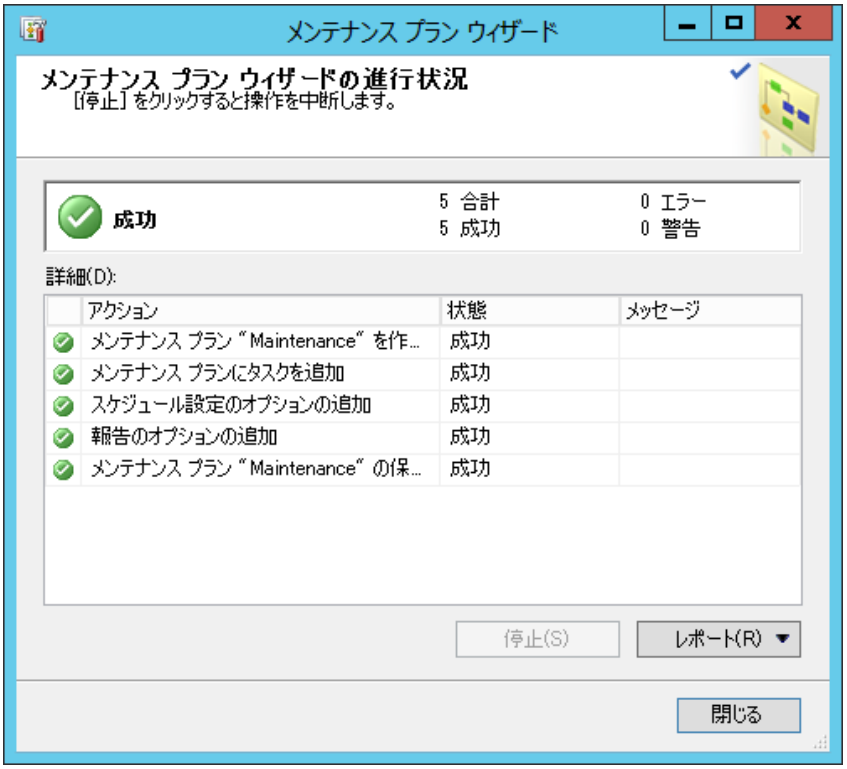


図 3.14　結果画面

14. メンテナンスの作成が完了したら、動作確認のために実行します。手動で実行する場合は、ツリービューでメンテナンスプランを選択して、右クリックメニューから *[**実行**]* を選択します。

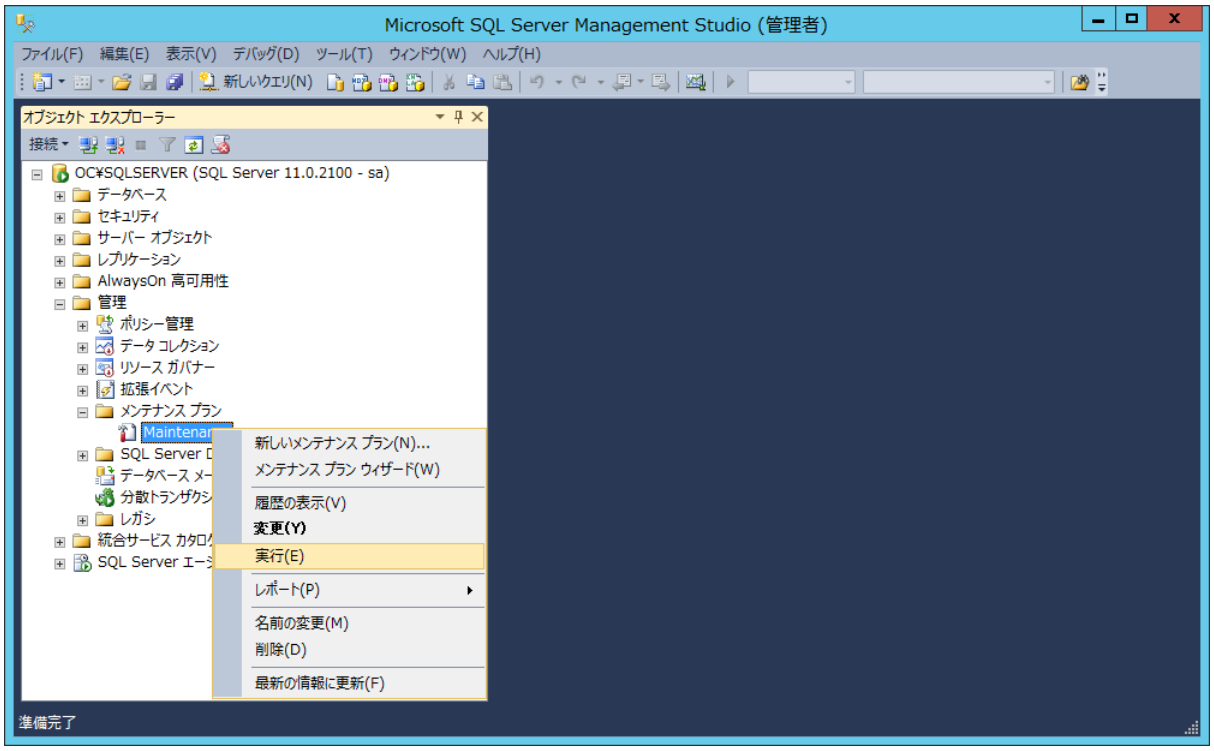


図 3.15　動作確認

注釈: メンテナンスプラン動作確認も事前に Orchestrator の停止を行ってください。

15. メンテナンスプランが問題なく完了することを確認します。

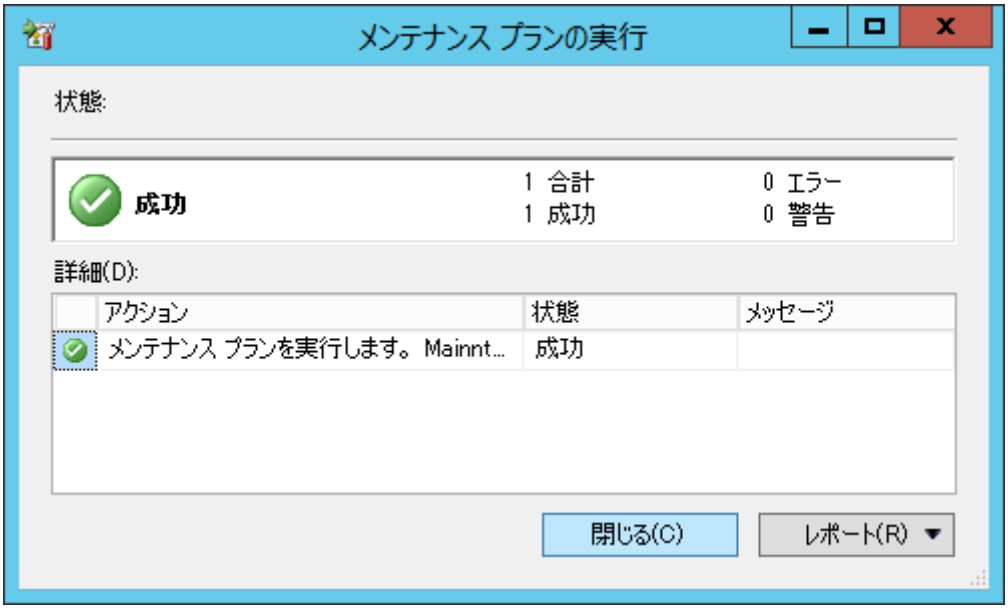


図 3.16　動作確認

Orchestrator のパッケージと設定ファイルのバックアップ

Orchestrator にアップロードされたパッケージを定期的にバックアップすることでディスク障害に備えます。パッケージはデフォルトでは **<インストール先>***NuGetPackages* に格納されます。スクリプトを利用したインストールで *-nugetPackagesPath* 引数を指定している場合は、指定したフォルダがパッケージの格納先になります。

パッケージフォルダについてはフォルダごと、バックアップ用の別ディスクにコピーを行ってください。

また、設定ファイル *Web.config* も変更する際にはバックアップを別ディスクにとってください。

Orchestrator の古いパッケージファイルの削除

パッケージが非アクティブ (Inactive) ステータスの場合にのみ、Orchestrator からパッケージを削除することができます。パッケージがプロセスにデプロイされていない場合のみ、非アクティブステータスとなります。

非アクティブなパッケージは Orchestrator のパッケージ画面で [バージョン表示] ダイアログを表示してから、*[***すべての非アクティブを削除***]* を押下することで削除できます。

図 3.17　非アクティブなパッケージの削除

注釈: 必要に応じてパッケージのダウンロードを行い、削除前にバックアップを取ってください。

リカバリー手順

SQL Server

SQL Server のデータベースのバックアップからのリストアは復旧モデルによって異なります。復旧の仕方は ‘SQL Server データベースのバックアップと復元 <https://docs.microsoft.com/ja-jp/sql/relational-databases/backup-restore/back-up-and-restore-of-sql-server-databases?view=sql-server-2017>‘ を参照してください。

注釈: リストア中に Orchestrator からデータベースにアクセスするのを防ぐために、データベースのリストア時には Orchestrator を停止してください。

Elasticsearch

Curator の場合

以下のコマンドでスナップショットの一覧を取得し、リストアしたいスナップショットの名前を確認します。スナップショットの作成については *Elasticsearch* **のバックアップ** を参照してください。

```
curator_cli.exe show_snapshots --repository リポジトリ名
```

スナップショットに含まれるインデックス等を確認したい場合は Elasticsearch の以下の API を直接利用してください。

```
GET _snapshot/リポジトリ名/スナップショット名
```

上記 API を呼び出すと JSON 形式でスナップショットの情報が返却されます。返却された JSON の indices フィールドにスナップショットに含まれるインデックス名の一覧が記載されています。

リストアするスナップショットを決定したら以下のファイルを作成します。ファイルの拡張子は .yml としてください。

```
actions:
  1:
    action: close
    description: "Close indices before restoring snapshot"
    options:
      continue_if_exception: True
      ignore_empty_list: True
    filters:
      - filtertype: pattern
        kind: prefix
        value: "インデックス名のプレフィクス"
  2:
    action: restore
    description: "Restore snapshot"
    options:
```

(次のページに続く)

(前のページからの続き)

```
      repository: "リポジトリ名"
      name: "スナップショット名"
      wait_for_completion: True
    filters:
      - filtertype: state
        state: SUCCESS
  3:
    action: open
    description: "Open indices after restoring snapshot"
    filters:
      - filtertype: pattern
        kind: prefix
        value: "インデックス名のプレフィクス"
```

ファイルを作成したら以下のコマンドを実行します。

```
curator.exe" 作成したファイルのパス
```

以上でスナップショットのリストアは終了です。

API の場合

Curator を利用しない場合は Kibana の Dev Tools 等で直接 Elasticsearch の API を利用します。まず、以下の API を利用してスナップショットの一覧を取得して、リストアするスナップショットの名前を確認します。

```
GET _snapshot/リポジトリ名/_all?filter_path=snapshots.snapshot
```

レスポンス例を以下に示します。snapshot フィールドの値がスナップショットの名前です。

```
{
  "snapshots" : [
    {
      "snapshot" : "daily-2019.01.30"
    },
    {
      "snapshot" : "daily-2019.01.31"
    },
    {
      "snapshot" : "daily-2019.02.01"
    },
    {
      "snapshot" : "monthly-2019.01"
    },
    {
      "snapshot" : "daily-2019.02.02"
    }
  ]
}
```

スナップショットに含まれるインデックス等を確認したい場合は Elasticsearch の以下の API を利用してくだ

さい。

```
GET _snapshot/リポジトリ名/スナップショット名
```

上記 API を呼び出すとスナップショットの情報が返却されます。返却された JSON の indices フィールドにスナップショットに含まれるインデックス名の一覧が記載されています。次に以下の API でインデックスをクローズ状態にします。

```
POST インデックス名/_close
```

インデックス名は default-* のようにワイルドカードで複数のインデックスを指定することができます。API が成功した場合は以下のような JSON が返却されます。

```
{
  "acknowledged" : true
}
```

インデックスがクローズ状態となっているかは以下の API で確認してください。

```
GET _cat/indices?v&h=health,status,index
```

レスポンス例を以下に示します。中央のカラムがオープン状態かクローズ状態かを示しています。

```
health status index
       close  default-2019.01
green  open   default-2019.02
green  open   .kibana_1
```

インデックスをクローズ状態にしたことを確認したら、以下の API でスナップショットをリストアします。

```
POST _snapshot/リポジトリ名/スナップショット名/_restore
```

リストア操作が受け付けられると以下のような JSON が返却されます。

```
{
  "accepted" : true
}
```

リストアの進捗は以下の API で確認できます。

```
GET _cat/recovery?v&h=index,shard,time,stage,snapshot,bytes_percent
```

レスポンス例を示します。インデックス default-2019.01 をスナップショット daily-2019.01.30 からのリストアが完了した後の状態で実行した例です。

```
index           shard time  stage snapshot         bytes_percent
（略）
default-2019.01 0     74ms  done  daily-2019.01.30 100.0%
（略）
```

リストアが完了するとインデックスは自動的にオープン状態となります。スナップショットに含まれていないリストア対象外のインデックスについては状態は変更されないので、クローズ状態のものでオープンが必要なインデックスについては以下の API でオープン状態にしてください。

```
POST インデックス名/_open
```

以上でスナップショットのリストアは終了です。

3.6 監視設計

Orchestrator を正常に稼働させるために様々な観点での監視が必要となります。本節では監視が必要な対象、および監視項目について記載します。監視するためのツールにおける監視可否や設定については、利用される監視ツールのマニュアルを確認ください。

3.6.1 監視対象

主に監視が必要となるのは以下の箇所です。

- アプリケーション* Orchestrator
- ミドルウェア* IIS * Microsoft SQL Server * Elasticsearch/Kibana * Redis
- サーバ
- ネットワーク* ロードバランサー* NLB と IIS 間のサブネット* IIS と SQL・Elasticsearch NLB 間のサブネット* Elasticsearch 間のサブネット

3.6.2 死活監視

Orchestrator および Orchestrator が利用するアプリケーションが正常に動作しているかを確認するためにヘルスチェック用の URL にアクセスすることで正常確認を行います。

Orchestrator

以下のヘルスチェック用の Rest API を実行することで確認できます。

```
GET <Orchestrator の URL>/api/Status
```

レスポンスのステータスコードが 200 であれば、Orchestrator およびネットワークは正常です。また、あわせて Orchestrator の URL にもアクセスして、レスポンスのステータスコードが 200 かどうかを確認してください。

Elasticsearch

以下のヘルスチェック用の API を実行することで確認できます。

```
GET <Elasticsearch の URL>/_cluster/health
```

3.6.3 アプリケーション監視

Orchestrator が出力するイベントログ (アプリケーション) を監視します。Orchestartor が出力するイベントはソース名が ‘ Orchestrator‘ から始まります。

3.6.4 ミドルウェア監視

IIS

- サービスの死活監視
- 性能監視
  - 応答性能

Microsoft SQL Server

- サービスの死活監視
- パフォーマンスカウンタ
- スロークエリ監視

Elasticsearch/Kibana

Redis

redis-cli コマンドを使って立ち上がっているかを確認できます。

```
redis-cli -h Redis サーバ名 ping
```

3.6.5 サーバ監視

- CPU
- メモリ (スワップアウトの状態)
- HDD 空き容量
- **故障**
  - リソース (CPU, メモリ, ストレージ)
  - ハードウェア

3.6.6 ネットワーク監視

- ネットワーク帯域
- ネットワーク接続 (数)

3.7 障害設計

3.7.1 ツール

Orchestrator で発生したエラー等はイベントログに記録されます。また、ロボットや Orchestrator Web 画面との通信は HTTPS が使われますが、そのログは IIS ログ、HTTPERR ログに記録されます。LogParser 等を用意しておくと、上記イベントログ、IIS ログ、HTTPERR ログ解析に役に立ちますので、予め該当するサーバーにインストールしておくことを推奨します。

第 4 章

システム構築・設定

4.1 AD 構築、証明書サービスの設定

Orchestrator には AD の機能が必要になる場合があります。

- HTTPS 通信のために証明書サービスを利用
- 冗長構成時のディスク同期に DFS を利用
- SQL Server を冗長構成時に利用 (WSFC、AlwaysOn)

すでに存在している AcriveDirectory サーバを利用することもできます。証明書サービスの利用については以下の URL を参照してください。

https://docs.microsoft.com/ja-jp/windows-server/networking/core-network-guide/cncg/server-certs/server-certificate-deployment

4.2 各サーバ共通

4.2.1 ドメイン参加

AD の機能が必要となる各マシンはドメインに参加する必要があります。Orchestrator サーバーの所属ドメインとロボット端末の所属ドメインは異なっていても構いませんが、サーバー証明書はグループポリシーなどでロボット端末に配布する必要があります。手動で個別に証明書をインストールするにはこちらの手順をご参照ください。

https://forum.uipath.com/t/topic/22015

4.3 DB サーバ

まず SQL Server をインストールします。冗長構成の場合はすべてのノードでインストールしてください。その後、WFSFC を設定し、AlwaysOn の設定を行います。

4.3.1 SQL Server インストール

SQL Server をインストールします。インストール後、Orchestrator 動作に必要な設定を行います。冗長構成の場合はユーザに注意してください。

4.3.2 WSFC 構築

冗長構成の場合、WSFC を構築します。

4.3.3 AlwaysOn 構成

冗長構成の場合、AlwaysOn を構築します。高可用性グループ作成時のポイント https://support.microsoft.com/ja-jp/help/2847723/cannot-create-a-high-availability-group-in-microsoft-sql-server-2012

4.4 AP サーバ

4.4.1 役割・機能追加

必要となる役割・機能は以下の URL を参照してください。

https://orchestrator.uipath.com/lang-ja/docs/server-roles-and-features

また、一括でインストールするスクリプトも提供しています。

https://orchestrator.uipath.com/lang-ja/docs/prerequisites-for-installation

IIS に関してはアプリケーションプールのリサイクルタイミングも考慮する必要があります。

4.4.2 必須パッケージのインストール

必須となるパッケージは以下の URL を参照してください。

https://orchestrator.uipath.com/lang-ja/docs/prerequisites-for-installation

4.4.3 Orchestrator インストール

冗長構成の場合は、先に Redis のインストールを行っておいてください。また Elasticsearch/Kibana を利用する場合も先にインストールを行っておいてください。

インストール方法は以下の URL を参照してください。

https://orchestrator.uipath.com/lang-ja/docs/about-installation

4.4.4 web.config 設定

設計時の検討した内容を設定します。冗長構成時は各ノードで同じ設定になっていることを確認します。

4.5 Redis サーバ (Linux)

対応しているバージョンは以下の URL を参照してください。

https://orchestrator.uipath.com/lang-ja/docs/software-requirements

大まかなインストール手順は以下の通りです。

- gcc, make インストール
- Redis のビルド・インストール
- カーネルパラメータ設定
- Redis 実行ユーザ・グループの作成
- master, slave, sentinel 毎の設定
- systemd サービス登録
- redis.conf 設定

注釈: Redis 上のデータは永続化する必要はないため、rdb、aof は無効にすることを推奨します。

4.6 Elasticsearch/Kibana サーバ

対応しているバージョンは以下の URL を参照してください。

https://orchestrator.uipath.com/lang-ja/docs/software-requirements

4.7 Orchestrator 設定

インストール後ライセンスを登録し、必要な設定を行います。
