UiPath Orchestrator

システムの基盤設計・運用ガイド [第 3 版]

UiPath Robotic Process Automation ®

リビジョン履歴

Date	Version	Author	Description
June 24, 2019	1.0	UiPath Infra Japan Team	First edition for v2018.4 release
April 6, 2020	2.0	UiPath Infra Japan Team	Second edition for v2019.10 release
March 18, 2022	2.1	UiPath Infra Japan Team	Correction for 2.2.5
November 22, 2022	3.0	UiPath Infra Japan Team	Third edition for v22.10 release

商標について

- UiPath のソフトウェア、製品、サービス、(これには、UiPath Orchestrator、 UiPath Robot、 UiPath Studio が含まれますが、これらに限りません) はアメリカ合衆国で登録された UiPath Inc.、 および 他の国・地域で登録された UiPath の関係会社の商標または登録商標です。 UiPath のロゴは UiPath Inc., が所有するものであり、UiPath の事前の明示的な許可なく、お客様及びその他の方が使用することはできません。
- Microsoft のソフトウェア、製品、サービス (これには、 Microsoft、 Windows、 Windows Server、 SQL Server 及び Active Directory が含まれますが、これらに限りません) は アメリカ合衆国で登録された Microsoft Corporation 及び他の国・地域で登録されたその関係会社の商標または登録商標です。
- Oracle のソフトウェア、製品、サービス (これには、 Java も含まれますがこれに限りません) は アメリカ合衆国で登録された Oracle 及びその他の国・地域で登録された関係会社の商標または登録商標です。
- Elastic は、 Elastic N.V. 及びその関係会社の商標または登録商標です。
- Redis は、 Redis Labs Ltd の商標です。
- その他、記載されている製品名、会社名およびサービス名はそれぞれの各社の商標または登録商標です。

免責事項

- 本ガイドの内容は 2022 年 11 月現在の情報であり、下記の製品リリースに基づいております。
  - UiPath Orchestrator v2022.10

- 製品の新しいリリース、修正プログラムなどによって、本ガイドの説明と異なる動作・仕様となる可能性がありますので、予めご留意ください。
- 本ガイドに含まれる情報は可能な限り正確を期しておりますが、 UiPath 株式会社の正式なドキュメントではありません。本ガイドに記載された内容に関して UiPath 株式会社は何ら保証していません。従って、本ガイドに含まれる情報の利用はお客様の責任においてなされるものであり、UiPath はガイドの内容によって受けたいかなる被害に関して一切の補償をするものではございません。
- 本ガイドは UiPath を法的に拘束する書類ではありません。UiPath はお客様に通知なくして、本ガイドの内容の一部または全部を修正及びアップデートできます。
- お客様は UiPath および執筆者の書面の承諾なしで本ガイドを複製、修正、頒布できません。

目次

1. Orchestrator 構築・運用概要

1.1. 本ガイドの目的・前提

- 本ガイドでは UiPath Orchestrator (以下 Orchestrator または OC と略す) を設計、構築、運用するにあたり、ミドルウェア・基盤観点からそれぞれのフェーズにおける留意事項や Tips などについて説明しています。これによって Orchestrator に関わる構築・運用担当者がトラブルなく導入・運用することを目的としています。
  - アプリケーション観点での Orchestrator 機能そのものについては基盤・ミドルウェアへの考慮が必要となる点についてのみ言及しております。それぞれの Orchestrator 機能の詳細については UiPath Orchestrator ガイド をご参照ください。
- 本ガイド執筆時点では Orchestrator は下記 3 つの形態で利用可能となっていますが、本ガイドでは 3 番目の Orchestrator の形態を前提としています。

#	形態	特徴
1	Automation Cloud	UiPath が提供する SaaS 型のクラウドサービス。サーバーサイドの製品群はお客様が導入作業を行うことなくすぐに利用することが可能です。お客様環境の既存または新規の UiPath Studio/Robot をインターネット経由で Automation Cloud に接続してお使いいただけます。
2	Automation Suite	Automation Cloud で提供されている殆どの製品群をお客様の環境に展開して利用することが可能です。基盤はオンプレミス・プライベートクラウド・パブリッククラウドに対応しております。
3	Orchestrator ※ 本ガイドの対象	Orchestrator 単体をお客様の環境に展開して利用することが可能です。基盤はオンプレミス・プライベートクラウド・パブリッククラウドに対応しております。

- 本ガイドでは UiPath 製品のバージョン **2021.10** 以降を前提としております。
- 本ガイドでワークフロー開発環境として “Studio” と記述していますが、StudioX・Studio Pro あるいは各種 Developer ライセンスを使用する場合も含んでおります。

1.2. 対象読者・前提知識

- 本ガイドは Orchestrator 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
  - Orchestrator を構成する基盤 (VMware vSphere, AWS, Azure など) についての一般的な知識

1.3. 本ガイドの構成

本ガイドのセクションはそれぞれの利用フェーズごとに構成されております。実施予定の作業項目に応じて該当するセクションをご参照ください。

フェーズ / セクション	作業項目
2. システム設計	● 機能・非機能要件を整理し、ハイレベルな Orchestrator 構成の検討を行います。 ● オンプレミスもしくはクラウドで準備・利用すべきサーバー・サービスについて整理します。 ● 高可用性が求められる環境においてはシングル構成・冗長構成を比較検討します。サーバー構築・運用費用を見積もり、投資対効果が得られるかによって意思決定を行います。
3. システム構築・設定	● それぞれの構成において構築時のポイントについて説明します。 ● 計画段階では構築にかかる作業工数を見積もり、構築・運用のスケジュールを立案します。
4. システム運用	● 長期安定稼働を実現するために、あらかじめ定時・非定時の運用手順・作業について整理をします。 ● 万が一のデータ破損に備えてバックアップ・リストアの手順を確立します。 ● 死活監視やリソース使用率の監視を行い、障害またはその予兆を早期検知します。

1.4. 用語

本ガイドで使用される Orchestrator 固有の主な用語と説明は次の通りです。

用語	説明
Orchestrator	UiPath Studio/Robot を統合管理する Web アプリケーションサーバー
Studio	自動化を行うワークフローを開発するためのアプリケーション
Robot	● Studio で開発された自動化ワークフローを実行するためのアプリケーション ● Robot には Attended Robot と Unattended Robot の二種類があります
Attended Robot (以下 AR と略す)	● ユーザーの操作によって開始され、同じマシン上においてユーザー監視下のもと実行されるロボット ● Orchestrator は AR と連携してプロセスの展開と実行ログを集中管理します
Unattended Robot (以下 UR と略す)	● スケジューリングなどにより人手を仲介せずに開始され、プロセス実行が自動化されたロボット ● Orchestrator は UR と連携してプロセス展開と実行ログの集中管理に加えて、リモート実行、監視、スケジュール実行などを提供します

High Availability Add-on (以下 HAA と略す)	● Orchestrator の Active-Active 冗長構成で必須となる Redis ベースのインメモリ DB ● 接続されている Studio/Robot のセッション情報などを Orchestrator ノード間で共有します
フォルダー	割り当てられたユーザーごとにテナント内の Orchestrator リソース (プロセスやキューなど) を分離するもの
ワークフロー	Studio によって開発される自動化処理を定義したファイル
パッケージ	Studio によってプロジェクト単位でワークフローをまとめたファイル
プロセス	ロボットによってパッケージを実行するために、パッケージをフォルダーに割り当てて管理されたもの
ジョブ	ロボットによりプロセスを実行したもの
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したもの
キュー	● ワークフローによって処理されるデータを収納するもの ● キューに含まれる個々のデータは処理されることによってトランザクションと呼ばれる

1.5. Orchestrator 基本機能

1.5.1. Orchestrator 概要

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 Web 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 運用に必要な管理を行うことができます。

大規模な RPA 環境の安定運用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。

要件	機能	得られる効果
スケジュール管理	ロボットの起動、実行等のジョブ管理	都度の手動実行を避け、作業ミス、作業漏れ等のヒューマンエラーを防ぎます
統合運用管理	実行結果一元管理	エラーを素早く検知し、対応することが可能になります
	開発成果物一元管理	開発成果物 (ワークフロー) 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ぎます

	ライセンス一元管理	ライセンスの更新工数の抑制や更新漏れを防ぎます
内部統制・ガバナンス	ユーザーごとのアクセスコントロール	ユーザーごとに参照・操作可能な Orchestrator リソースをコントロールすることで不正アクセス、不正実行などを防ぎます
	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	ロボットごとに自動化対象の業務システムで利用するユーザー名・パスワードをセキュアな環境で一元管理できます Windows 認証、Azure AD 認証または SAML 認証を利用した Orchestrator 管理コンソールへのシングルサインオンが可能です

1.5.2. Orchestrator 標準機能一覧

Orchestrator が提供する主な標準機能は下記表の通りです。

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
	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、ユーザー等) を権限分離して管理
	フォルダー管理	単一テナント内で、プロセス、アセットなどの一部のオブジェクトを権限分離して管理
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 のすべてのリソース (プロセス、ジョブ、スケジュールなど) に対する操作の監査証跡を表示
	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を API で提供

2. システム設計

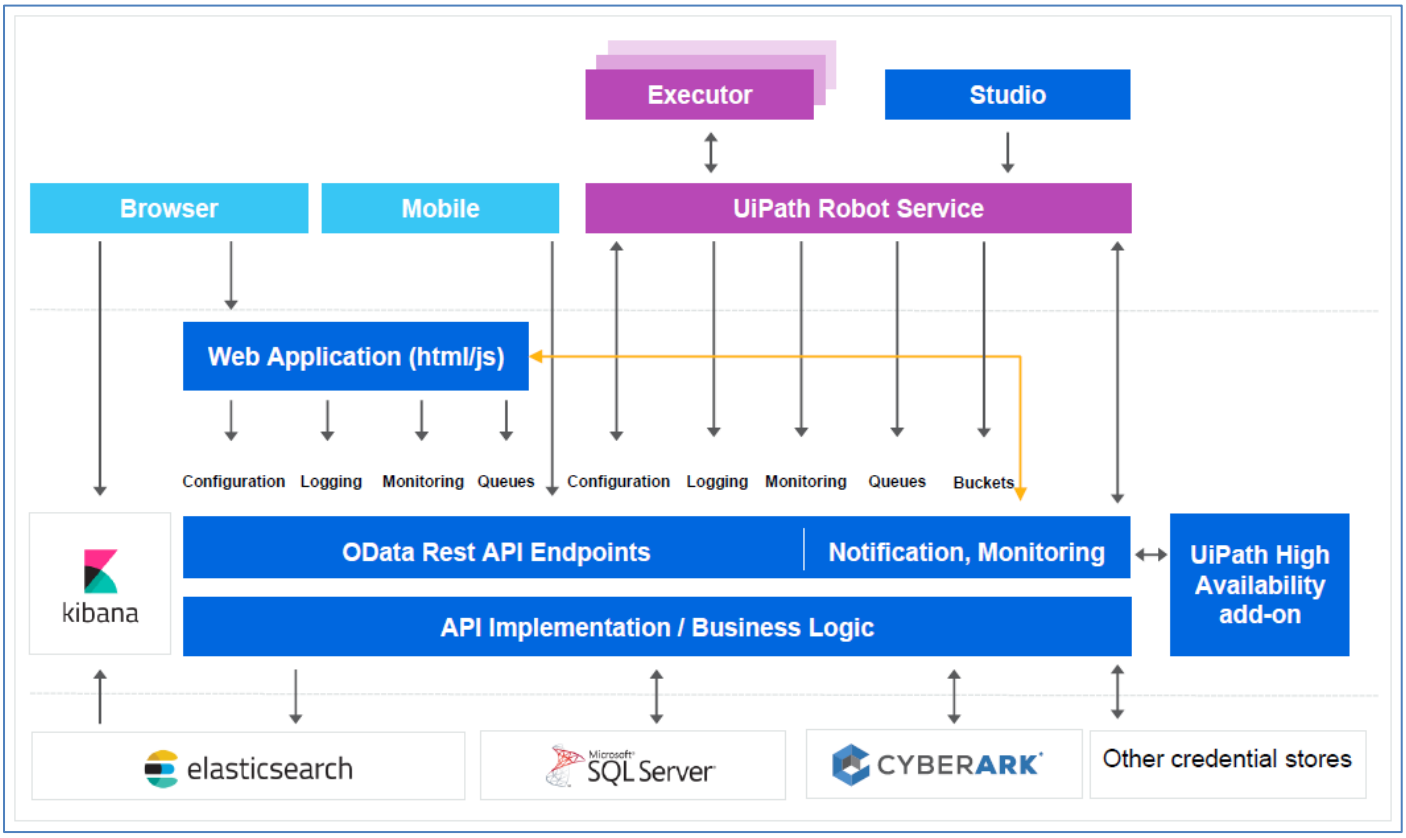
2.1. 概要

- システム設計に先立ち、Orchestrator の機能要件・非機能要件を整理し、ハイレベルな Orchestrator 構成の検討を行います。
- オンプレミスあるいはクラウドで準備・利用すべきサーバー・サービスについて整理します。
- 高可用性が求められる環境においては シングル構成と冗長構成を比較検討します。それぞれのサーバー構築・運用費用を見積もり、投資対効果が得られるかによって意思決定を行います。
- Orchestrator 構成が固まり次第、個々のコンポーネントの設計を開始します。

2.2. 構成検討の考慮点

まず Orchestrator が機能するための内部的なアーキテクチャーとアプリケーションデータの流れと配置場所を説明し、次にこれらを実装するための基盤・環境について説明をします。

2.2.1. Orchestrator 内部アーキテクチャー

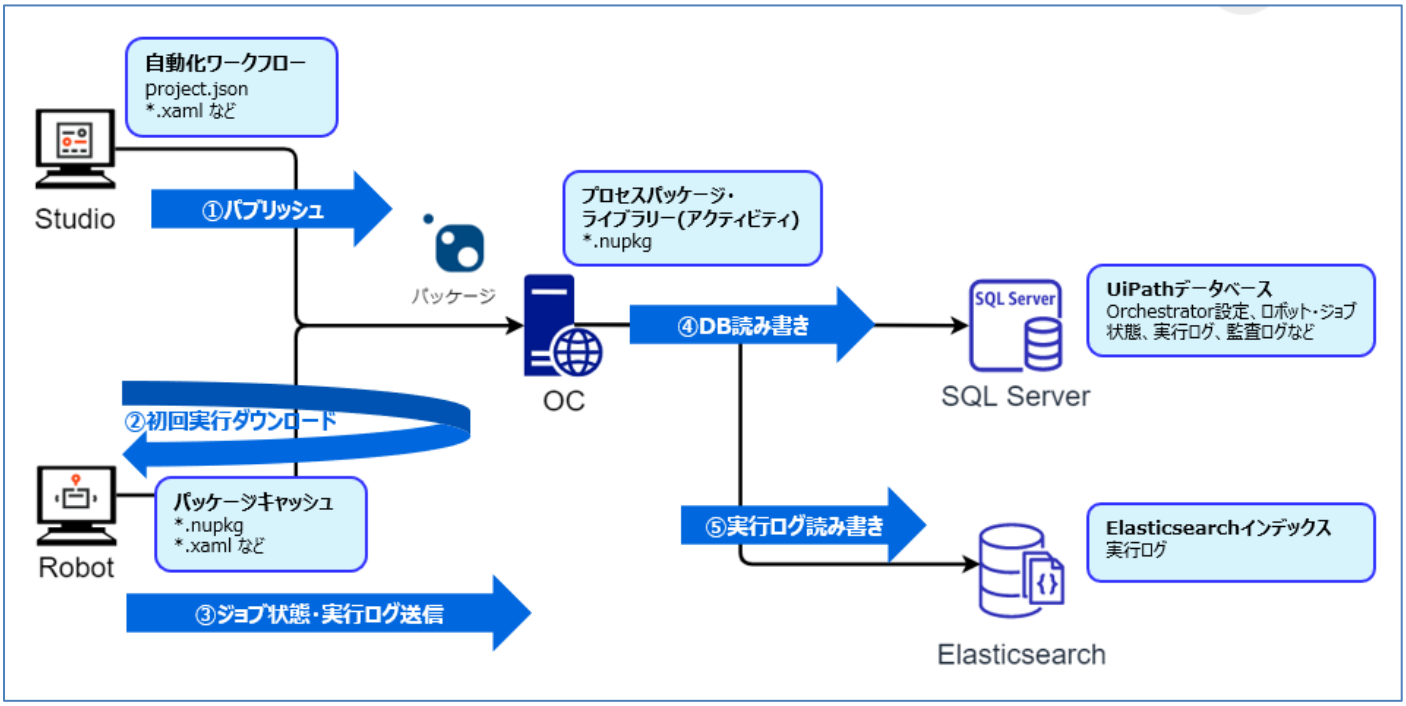


- Orchestrator は 3 階層から構成される Web アプリケーションサーバーです。
  1. プレゼンテーション層
  2. アプリケーション層 (Web サービス層)
  3. パーシステンス層

- このうちプレゼンテーション層・アプリケーション層は IIS サーバー上の Web サイトおよびアプリケーションプールとして動作します。
  - Web/AP サーバーまたは単に Web サーバーとも表現されます。
  - パーシステンス層のうち必須となるコンポーネントは Orchestrator のアプリケーションデータを格納する SQL Server 上のデータベースであり、DB サーバーとも表現されます。
    - 実行ログは既定では SQL Server に格納されますが、オプション設定で Elasticsearch に格納し Kibana で可視化することも可能です。
    - Unattended ロボットのパスワードやクレデンシャル型アセットなどの機密情報は既定では SQL Server に暗号化して格納されますが、オプション設定でユーザー管理の CyberArk や Azure Key Vault などの 資格情報ストア に別途格納して管理することも可能です。
  - クライアントは、ブラウザー・Studio・Robot または Orchestrator API を利用するカスタムのスクリプトなどです。

2.2.2. アプリケーションデータの流れと配置場所

下図は Studio・Robot・Orchestrator で作成・利用される主なアプリケーションデータの流れと配置場所について説明したものです。



主なデータの流れ

① Studio で作成したワークフローをパブリッシュすることによりプロセスパッケージとして Orchestrator に保存されます。

② Robot からのプロセス実行時には初回は Orchestrator からプロセスパッケージと依存するアクティビティをダウンロードし、ローカルディレクトリにキャッシュします。2 回目以降の同一プロセス実行時にはキャッシュが使用されます。

③ Robot からジョブの実行状態や実行ログなどが随時 Orchestrator に送信されます。

④ Orchestrator から SQL Server 上のデータベースへ、Orchestrator 設定、Robot・ジョブ状態、実行ログなどが読み書きされます。

⑤ オプション設定により実行ログを Elasticsearch のインデックス(データベース)に読み書きすることも可能です。

上記は説明のための順序であり、実際のデータのやり取りは並行して行われるものもあります。

主なデータの配置場所

ワークフロー

- Studio によって開発される自動化処理を定義したプロジェクトファイルおよび XAML ファイル
- 既定では Studio マシンのローカルディレクトリ (%UserProfile%\Documents\UiPath) 配下にプロジェクトごとにファイルが作成されます。

パッケージ・ライブラリー

- Studio からワークフローをパブリッシュすることによって作成される NuGet パッケージ形式(*.nupkg)のファイル
- 自動化処理をパッケージ化したプロセスパッケージと、共通処理を部品化したライブラリー(アクティビティ)の 2 種類があります。
- Studio から直接パブリッシュするか Orchestrator 管理画面よりアップロードすることにより、Orchestrator のアプリケーションデータとして保存されます。既定では Orchestrator Web サーバーのローカル Storage ディレクトリ(C:\Program Files (x86)\UiPath\Orchestrator\Storage) 配下にテナントごとに保存されますが、設定変更により別のファイルサーバーもしくはストレージサービスに保存することも可能です。なおパッケージのメタデータ(バージョン情報や説明など)は UiPath データベースに格納されます。
- パッケージ・ライブラリー以外のアプリケーションデータについては Storage ディレクトリ をご参照ください。

パッケージキャッシュ

- Robot がプロセス実行時にローカルディレクトリ (%UserProfile%\.nuget\packages) にキャッシュするプロセスパッケージと依存関係のあるアクティビティ
- 初回プロセス実行時にはプロセスパッケージは Orchestrator から、アクティビティはパッケージキャッシュ・ローカルフィードに存在しないものは Orchestrator または事前設定された NuGet フィードからをダウンロードされます。2 回目以降の同一プロセス実行時にはキャッシュが使用されダウンロードは行われません。

UiPath データベース

- Orchestrator アプリケーション設定(ユーザー・ロール・マシン・ロボット・プロセス・トリガー・アセット・キューなど)やジョブ実行結果・ロボット実行ログなどの永続的なアプリケーションデータが保存される SQL Server 上のデータベース

- Identity Server、更新サーバー、Test Automation など Orchestrator に内包されるサービス・機能のアプリケーションデータも保存されますが、設定によりデータベースを分離することも可能です。
- オプション設定によりロボット実行ログを Elasticsearch などの別の保存先に格納することも可能です。
- 資格情報ストア 機能により機密データを CyberArk や Azure Key Vault など別の Vault サーバーに格納することも可能です。

Elasticsearch インデックス

- オプション設定によりロボット実行ログの保存先として Elasticsearch を指定した場合、Elasticsearch のインデックス(データベース)として実行ログを保存し、Orchestrator から参照することが可能です。
- Elasticsearch に保存された実行ログは Kibana を使用して分析・可視化することが可能です。

2.2.3. 基盤選定

- UiPath Orchestrator/Studio/Robot は、稼働させる基盤としてオンプレミス(VMware vSphere などの仮想基盤または物理サーバー)とパブリッククラウド (AWS/Azure など) の両環境に対応しております。また Orchestrator はパブリッククラウド、Robot はオンプレミスといったハイブリッドな構成を取ることも可能です。既に他システムでの実績があればそれに倣って基盤を決定するのも一案ですが、ゼロベースでどちらの基盤が良いかいくつかの観点から比較検討するケースも考えられます。
- 一般的なオンプレミスとパブリッククラウドの比較に加えて、UiPath 製品を加味したメリット・デメリットを整理した表を下記に示します。これらはあくまでも一例となりますので、お客様環境のご要件に応じて取捨選択していただくことを推奨いたします。

観点	クラウド利用時のメリット	クラウド利用時のデメリット
導入	● 機器調達が不要 ● 短期間での構築が可能 ● サイジングの精緻化が不要 ● 自動デプロイ機能を利用することで設計書のドキュメント化が容易 ● 冗長構成の構築が容易	● 既存システムやオンプレの Robot との接続に考慮が必要 ● 認証付きプロキシ経由でのオンプレ Robot-Orchestrator 間の接続は未対応 ● 割り当てられるリソースに上限がある場合がある

運用・保守	● ある程度の監視機能をデフォルトで利用可能 ● メンテナンス作業の自動化が容易 ● SLA が設定されている ● バックアップやリストアが容易 ● UiPath 製品のバージョンアップ時、インプレース(上書き)方式だけでなく、パラレル(並行稼働)方式でのバージョンアップが容易 ● スケールアウトやスケールアップが容易 ● Orchestrator バージョンアップ検証時などに一時的な検証環境の構築が可能	● マネージドサービスの場合、ログ取得に制限があり、トラブルシューティングが困難となる場合がある ● クラウドサービス事業者による障害やメンテナンスにより停止する場合がある ● 古いバージョンの OS が利用できない場合がある ● マネージドサービスはスケールに制限がある
セキュリティ	● リソース操作の証跡として監査ログを取得可能	● データの保管場所、リスク、適切なセキュリティコントロールについての認識と対策が必要
コスト	● 初期投資が低い	● 傾向として長期利用の場合には割高 ● 正確なコスト算出が困難

パブリッククラウド環境での Orchestrator 構成例と概算コストは パブリッククラウド環境における Orchestrator 設計 をご参照ください。

2.2.4. 非本番環境

- 開発、テスト(ステージング)、検証、本番の 4 つ環境を考慮して構成を検討することを推奨します。「開発」→「ユーザー受け入れテスト / 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。
- 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 プロダクトライフサイクル にてアナウンスされている通り、使用中バージョンのサポート期間の終了または新規バージョンに実装された新機能や不具合修正等が必要な場合に備えて、バージョンアップ検証が可能な環境を別途ご用意いただくことをお勧めします。
- 非本番環境 (開発・テスト・検証) は Orchestrator Non-Production ライセンスを使用することができます。

開発・テスト・検証・本番環境の棲み分け

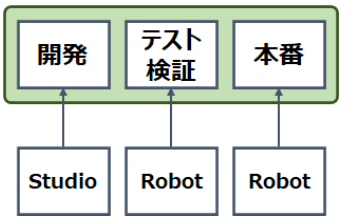
それぞれの環境における用途と留意点は次の通りです。

環境	用途	留意点
開発環境	開発者が Orchestrator の機能 (アセットやキュー) を使用したワークフロー開発およびデバッグを行う	● Orchestrator は最小構成で十分
テスト(ステージング)環境	開発されたワークフローを、本番環境に準ずる環境で受け入れテスト (UAT) を実施し、本番環境での予期しないトラブルを未然に防止する	● 本番環境で構成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 Robot の端末も本番環境と同じ Robot バージョン、アクティビティが準備されていること
検証環境	Orchestrator のバージョンアップ手順の確認、パラメータ変更など、Orchestrator 自身の事前検証を行う	● Unattended Robot が常時稼働している環境など、Orchestrator のダウンタイムを極小化したい場合には検証環境構築が推奨される ● 効果的な検証を行うために本番環境と同様の構成 (シングルまたは冗長構成) が推奨されるが、サーバースペックは同じである必要はない (スケールダウン可)
本番環境	自動化の本番業務を行う。	● 機能要件・非機能要件に応じて適切に構成し、運用状況に応じてスケールを行う。

構成例ごとのメリット・デメリット

初期導入・運用コスト、障害時の影響範囲により物理的な分離環境を構成します。下記にいくつかの構成例を示します。

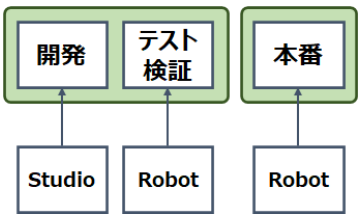
例 1: オールインワン (すべて兼用)



- 特徴: 一つの Orchestrator がすべての役割を兼用し、Studio/Robot もすべて接続する。
- メリット: 初期コスト、運用コストを最小化できる。
- デメリット: ワークフロー開発または受入テスト時の Studio/Robot 稼働が本番環境に影響を与える可能性がある。たとえば誤って無限ループのワークフローを実装してしまった場合、大量のログ送信によって本番環境がスローダウンまたは停止することがある。

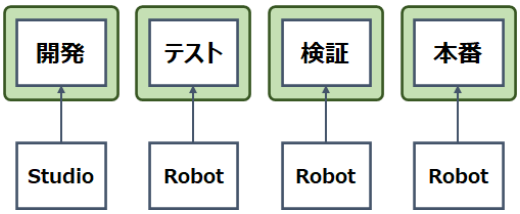
開発・テスト用のキュー・アセットなどのリソースを本番用と混同し、誤操作してしまうリスクがある。

例 2: 本番環境独立 (開発兼テスト兼検証)



- 特徴: 2 つの Orchestrator 環境を構築し、開発・テスト・検証環境を兼用した環境と本番環境を構成する。
- メリット: 本番環境を分離することによって、ワークフロー開発・テスト時の本番環境への負荷や誤操作による影響を無くすことができる。
- デメリット: 開発から本番へのデータ移行の手順を確立する必要がある。Orchestrator バージョンアップ検証時に開発・テスト環境として使用できなくなる。

例 3: 完全独立



- 特徴: それぞれの Orchestrator を独立して構築する。
- メリット: 各環境での処理や作業が他環境へは無影響となり、並行して稼働・作業を行うことができる。
- デメリット: それぞれの環境を構築・運用することになるため高コスト。それぞれの環境からのデータ移行の手順を確立する必要がある。

これらはあくまでも構成例であるため、お客様のご要件に応じて物理環境を構成し、必要に応じてマルチテナントによる論理的なデータ分離や、Non-Production ライセンスの適用などを検討してください。

2.3. 非機能要件

2.3.1. 性能

一般的にロボット台数に応じて Orchestrator (Web サーバー) と SQL Server (DB サーバー) をホストするサーバーのスペックと構成を決定します。

- 詳細は ハードウェア要件 をご参照ください。
- パフォーマンスの考慮事項については パフォーマンスのベスト プラクティス をご参照ください。

Orchestrator (Web サーバー)

- 1 台の Orchestrator で管理できるロボット台数に機能的な制限はありませんが、接続されているロボット台数がハードウェア要件を満たさない場合にはパフォーマンス劣化が生じる可能性があります。またロボット台数の増加に比例して Orchestrator ダウン時の影響範囲も大きくなります。
- Unattended Robot 500 台より多い場合、Active-Active の Orchestrator 冗長構成によりスケールアウトによって負荷分散することをお勧めします。
- Attended Robot は目安として Unattended Robot に比べてログ送信量などが 1/10 程度に収まるケースがありますが、Orchestrator への負荷は利用環境およびジョブ実行頻度に大きく左右されます。一般的には Unattended Robot よりも多くの Attended Robot を 1 台の Orchestrator に接続することが可能ですが、Orchestrator のリソース利用状況を監視しながら、徐々に Attended Robot の接続台数を増加させることを推奨します。

SQL Server (DB サーバー)

- ロボット接続台数に応じてハードウェア要件を満たすようにスペックを選定します。
- ロボットのジョブ実行時に出力される実行ログ量および保持期間に応じて将来に渡って必要となるディスク容量を推定します。実行ログの保持期間はエンドユーザーのコンプライアンス・データ監査のガイドライン等も考慮します。古いログは データベースのアーカイブと古いレコードの削除 を参照して定期的に削除するよう運用設計します。
- SQL Server はスケールアウトによる負荷分散ができないため、性能向上にはスケールアップ(マシンスペックの増強) によって対応します。
- スケールアップの上限に達した場合には、SQL Server に送るロボット実行ログの量を減らすため、Elasticsearch ノードを別途構築してログを退避させるとともに、SQL Server 向けの NLog ログレベル変更を検討します。

2.3.2. 可用性・信頼性

可用性と信頼性を高めるために 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、以下の内容を検討します。

- Orchestrator がダウンした時の影響
- 各コンポーネントがダウンする可能性、ダウンした時の影響、復旧方法

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「ライセンスを検証せずにロボットをオフラインで実行できる合計時間」の設定が必要です。 詳細な動作要件は ライセンスキャッシュ機能 をご参照ください。

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 (停止すべきか確認) 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、アセットなど Orchestrator 機能を利用するワークフローの開発ができなくなります。 パッケージを Orchestrator にパブリッシュできなくなります。 Orchestrator からライセンスを付与されている場合、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「ライセンスを検証せずにロボットをオフラインで実行できる合計時間」を設定している場合は可能になります。

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	ダウンした時の影響	復旧方法
Orchestrator	ハードウェア障害、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	上記「Orchestrator がダウンした時の影響」参照	Internet Information Services (以下 IIS と略す) 再起動
SQL Server	ハードウェア障害、プロセス障害、重いクエリによる過負荷、DB 破損	上記「Orchestrator がダウンした時の影響」参照	データリストア、SQL Server 再起動
HAA (Redis)	ハードウェア障害、プロセス障害	上記「Orchestrator がダウンした時の影響」参照	HAA (Redis) 再起動
Elasticsearch	ハードウェア障害、プロセス障害	ロボット実行ログ保存に影響	データリストア、Elasticsearch 再起動
Kibana	ハードウェア障害、プロセス障害	ロボット実行ログ分析に影響	Kibana 再起動

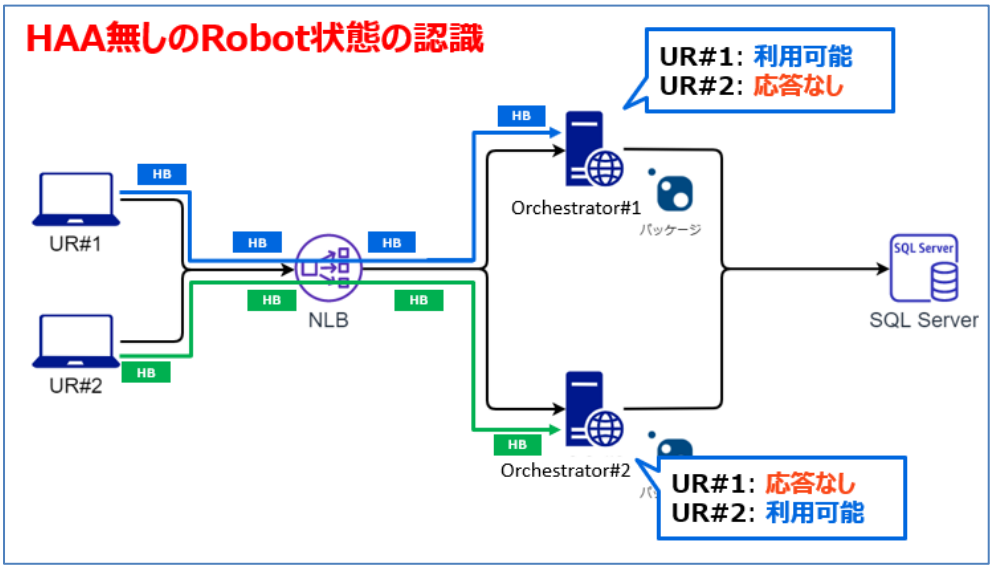
HAA (Redis)

HAA (Redis) の役割

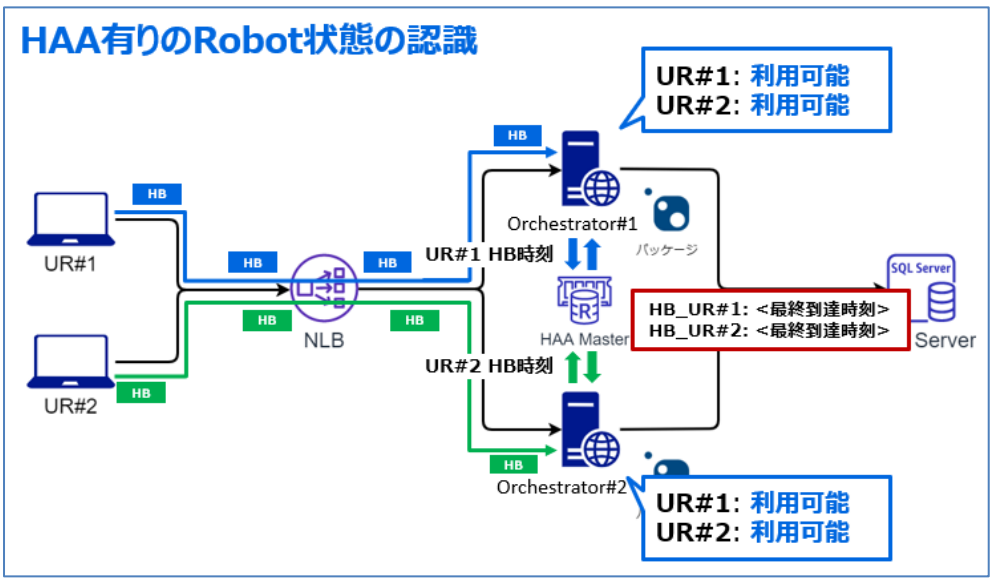
Orchestrator を Active-Active な冗長構成にする場合には、HAA (High Availability Add-on) または Redis のサーバーが別建てで必須となります。HAA (Redis) はインメモリ DB として動作し ①Studio/Robot からのセッション情報の管理 および②DB 情報を一部キャッシュしパフォーマンス向上に寄与する、という 2 つの役割があります。

このうち①の役割が特に重要で Active-Active 構成において HAA(Redis)が存在しない場合、Orchestrator 各ノードに接続されている Robot 状態の認識にずれが生じるため UR で正常にジョブ実行されなくなります。

下記に例を示します。(※ SQL Server は簡略化して 1 台構成で記載)



- 各 Robot は 30 秒に 1 回 HB(ハートビート)を Orchestrator に送信し、受け取った Orchestrator は対象 Robot が「利用可能」と認識します。
- HB が 4 回(つまり 30 秒*4=120 秒間)到達しない場合、Orchestrator は対象 Robot を「応答なし」と認識します。
- Robot の状態は各 Orchestrator がメモリ上に記憶しており、逐一 DB に書き込みは行いません。
- HAA (Redis) が存在しない場合、Orchestrator の各ノードは自分に送信された HB しか分からないため Robot 状態の認識にずれが生じます。
- この状態で Unattended Robot でジョブのトリガー実行が Orchestrator #1 で開始された場合、UR#1 は「利用可能」状態のためジョブ開始することができますが UR#2 は「応答なし」状態のためジョブ開始することはできません。



- HAA (Redis) は存在する場合、各 Orchestrator が受信した各 Robot の HB の情報をそれぞれが HAA メモリ DB に書き込みます。

- 各 Orchestrator が HB 情報を HAA メモリ DB から読み込むことにより、Robot 状態認識の一貫性を保つことができます。
- Orchestrator いずれかのノードに HB が到達していれば、いずれの Orchestrator がトリガー実行を開始した場合でもジョブ実行開始が可能となります。

HAA (Redis) の冗長化

- HAA (Redis) 本体をシングル構成とするとそこが単一障害点となるため、HAA (Redis) も冗長化させることが可能です。冗長構成ではクォーラム投票によって過半数を取得したノードがプライマリとして機能するよう 3 台にて構成する必要があります。
- HAA 冗長構成では DNS サーバーの NS レコードを利用して Orchestrator からの接続において固定のエンドポイント (接続先) を作成することもできます。詳細な手順は DNS エントリの設定をご参照ください。
- Redis オープンソース版でも Orchestrator と連携することは可能ですが、Redis 本体は UiPath ではサポート対象外となります。Redis を含めた包括的なサポートが必要な場合には、有償の商用版である HAA の導入をご検討ください。

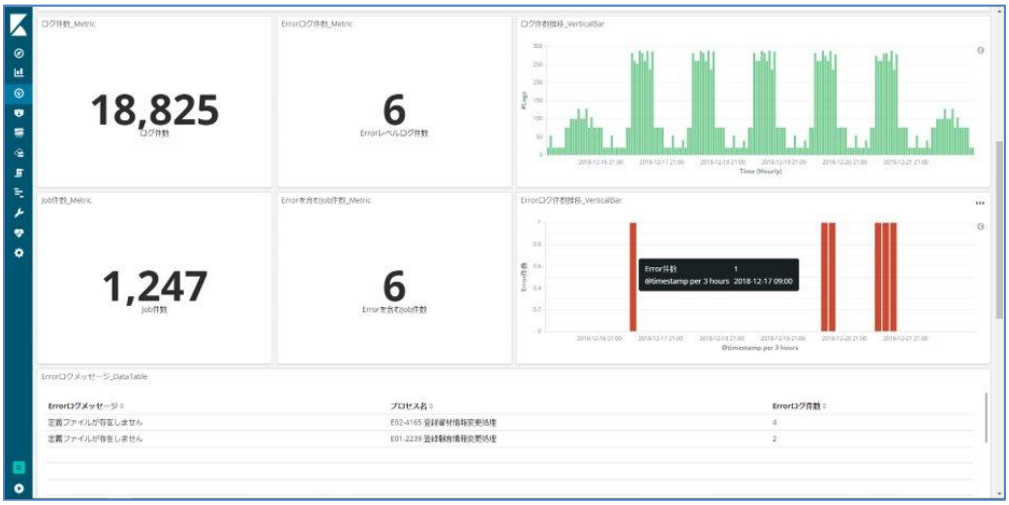
Elasticsearch/Kibana の役割

Elasticsearch は Elastic 社が提供する分散型データベースです。Orchestrator は既定値では実行ログを SQL Server に格納しますが、設定変更により Elasticsearch に格納することも可能です。

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 200 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。

ログ設定方法については ログの構成 を参照してください。

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。Kibana を使用したダッシュボードによりジョブのエラー検知、エラー箇所のリアルタイム分析、ログ量監視、端末の利用状況把握などを実現することができます。



2.3.3. 拡張性

今後ロボット台数の増加が見込まれる場合には、前述のハードウェア要件を超過しないように予め余裕を持ったスペックでサーバー構築することが推奨されますが、利用状況の監視によって運用後リソース不足が明らかとなるケースが考えられます。

そのような状況に備えて、予めサーバーの拡張方針について検討しておくことを推奨します。

- スケールアップ (Web サーバーおよび DB サーバー)
- スケールアウト (Web サーバー)
- Elasticsearch 導入 (DB サーバーの負荷軽減)

2.3.4. セキュリティ

Orchestrator におけるセキュリティ上の考慮点について説明します。最新のセキュリティ考慮事項につきましては セキュリティのベスト プラクティス をご参照ください。

DB 接続文字列

Orchestrator から SQL Server 認証を使用して DB 接続を行っている場合には、接続に使用する SQL Server のユーザーとパスワードが設定ファイル UiPath.Orchestrator.dll.config および appsettings.Production.json に平文として設定されます。

これらの設定ファイルは Orchestrator インストールディレクトリに配置されるため、管理者以外が参照できないように権限を適切に設定することを推奨します。

加えて設定ファイルの内容によって DB 接続ユーザーの資格情報が知られないように下記のいずれかの方法を採ることができます。

- Windows 認証による DB 接続
  - Orchestrator のアプリケーションプール ID としてドメインユーザーを指定し、SQL Server に **db_owner** ロールを持つユーザーとして登録することにより、パスワードを設定ファイルに記述する必要がなくなります。
- Azure AD マネージド ID
  - Orchestrator を Azure VM または Azure App Service にインストールしている環境では SQL Server に対してマネージド **ID** を使用して Azure AD 認証することができます。この方法によってパスワードを設定ファイルに記述する必要がなくなります。
- DB 接続文字列暗号化
  - 下記サイトを参照して、それぞれの設定ファイルの DB 接続文字列を暗号化します。ただし Orchestrator をバージョンアップする際には一時的に復号する必要があります。
    - UiPath.Orchestrator.dll.config セクションを暗号化する
    - AppSettings.Production.json を暗号化する

NuGet パッケージダウンロード

Orchestrator 管理画面においてパッケージ参照権限を持つユーザーはパッケージをダウンロードすることが可能ですが、ワークフローに機微な情報を含む場合などセキュリティの観点から禁止したい場合があります。

また Orchestrator 管理画面でパッケージ内のワークフローを表示する機能があり、この機能を禁止したいケースが考えられます。

Orchestrator ではこれらを無効化する機能が備わっていないため、代替手段として Orchestrator 管理画面からパッケージダウンロードやワークフロー表示を行う API を禁止します。この設定を行っても Robot によるプロセス実行には影響はありません。

Web.config で所定のセクションに該当リクエストを拒否するフィルターを追加します。

```
<configuration>
 <system.webServer>
  <security>
   <requestFiltering>
    <denyUrlSequences>
     <add sequence="/odata/Processes/UiPath.Server.Configuration.OData.DownloadPackage" />
    </denyUrlSequences>
   </requestFiltering>
  </security>
 </system.webServer>
</configuration>
```

ライブラリーのダウンロードも拒否するには同じセクションに次のフィルターを追加します。

```
<add sequence="/odata/Libraries/UiPath.Server.Configuration.OData.DownloadPackage" />
```

安全性の低い SSL/TLS バージョンの無効化

Robot から Orchestrator への HTTPS 通信は TLS1.2 が使用されます。またブラウザーアクセスもブラウザー設定により明示的に無効化しない限り TLS1.2 が使用されます。

SSL3.0/TLS1.0/TLS1.1 は安全性が低いため Orchestrator ホスト側のレジストリ設定によって無効化することが推奨されます。また RC4 などの脆弱性のある暗号化スイートの無効化も検討します。詳細な設定手順は次のサイトをご参照ください。

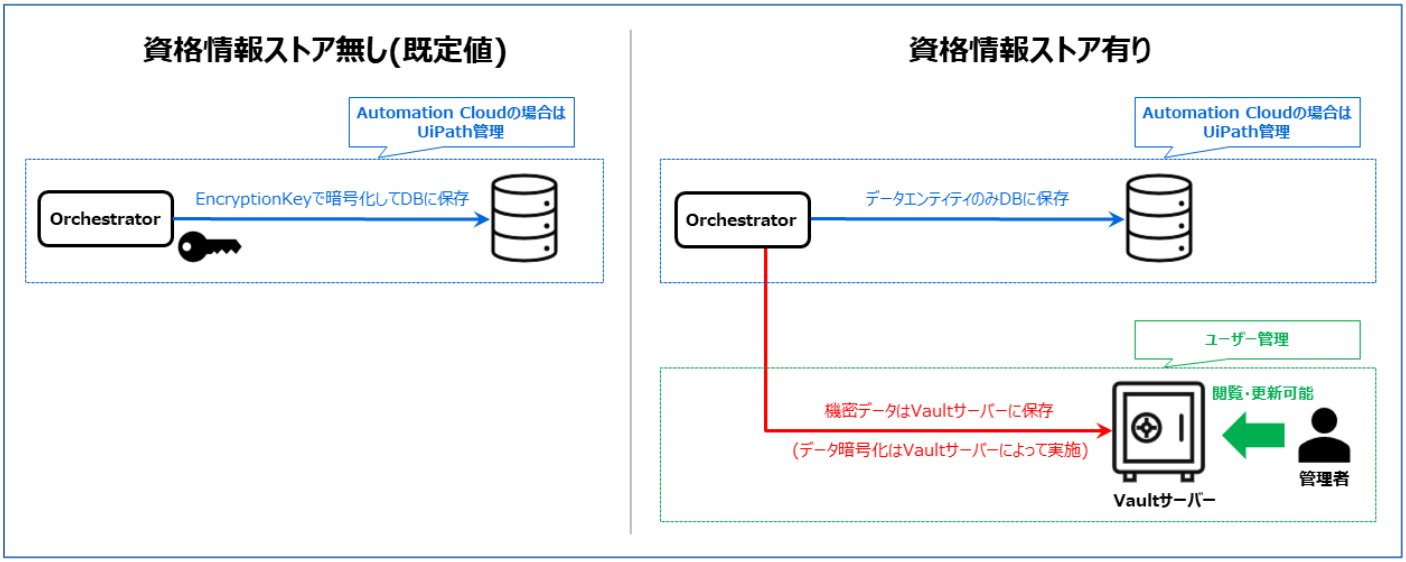
- クライアントの SSL/TLS プロトコルと暗号スイートの AD FS を管理する
- トランスポート層セキュリティ (TLS) のレジストリ設定

ただし Orchestrator API をコールするスクリプトを使用している場合、スクリプトの REST API 処理が TLS1.2 を使用するよう設定が必要となるケースがあります。たとえば PowerShell の Invoke-WebRequest や Invoke-RestMethod は環境によっては既定で TLS1.0 が使用される場合があるため、明示的に TLS1.2 を使用するように次の一文を実行します。

```
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12
```

資格情報ストア

既定の設定では機密データ(Unattended ロボット実行ユーザーのパスワードとクレデンシャル型アセット)はデータベースに暗号化して保存されます。資格情報ストア機能を利用することにより、これらの機密データをデータベースから分離しエンドユーザー管理の Vault サーバーに保管できるようになります。



この機能はオンプレミスでも Automation Cloud でも使用可能で、エンドユーザーのセキュリティポリシーに応じて利用を検討します。特に Automation Cloud 環境ではデータ保管の暗号化キーおよびデータ本体が UiPath 管理となっており、ユーザー側ではコントロールすることができない仕様になっています。エンドユーザーのセキュリティ要件を満たすために、資格情報ストアを利用してこれら機密データを分離してエンドユーザー管理の Vault サーバーに保管することができます。

機能の詳細、サポートされる Vault サーバーの種類および設定手順は 資格情報ストア をご参照ください。

Elasticsearch/Kibana セキュリティ

Elasticsearch/Kibana を導入し、実行ログを格納する場合にはアクセス制御について検討が必要となります。既定値ではアクセス制限がかけられていないため、Elasticsearch/Kibana の URL を知る一般ユーザーがすべての実行ログを参照できることになります。

Elasticsearch/Kibana のセキュリティ機能 (ログイン画面によるアクセス制御など) は以前のバージョンでは X-Pack と呼ばれる有償サブスクリプションとして提供されていましたが、Elasticsearch 6.8.0 および 7.1.0 以降はベーシックライセンスにより一部機能が無償提供されるようになりました。古いバージョンをお使いの場合には 6.8.0/7.1.0 以降へのバージョンアップを行い、セキュリティ機能の有効化を推奨いたします。

なお有償サブスクリプションとの機能比較は Elastic Stack サブスクリプション をご参照ください。

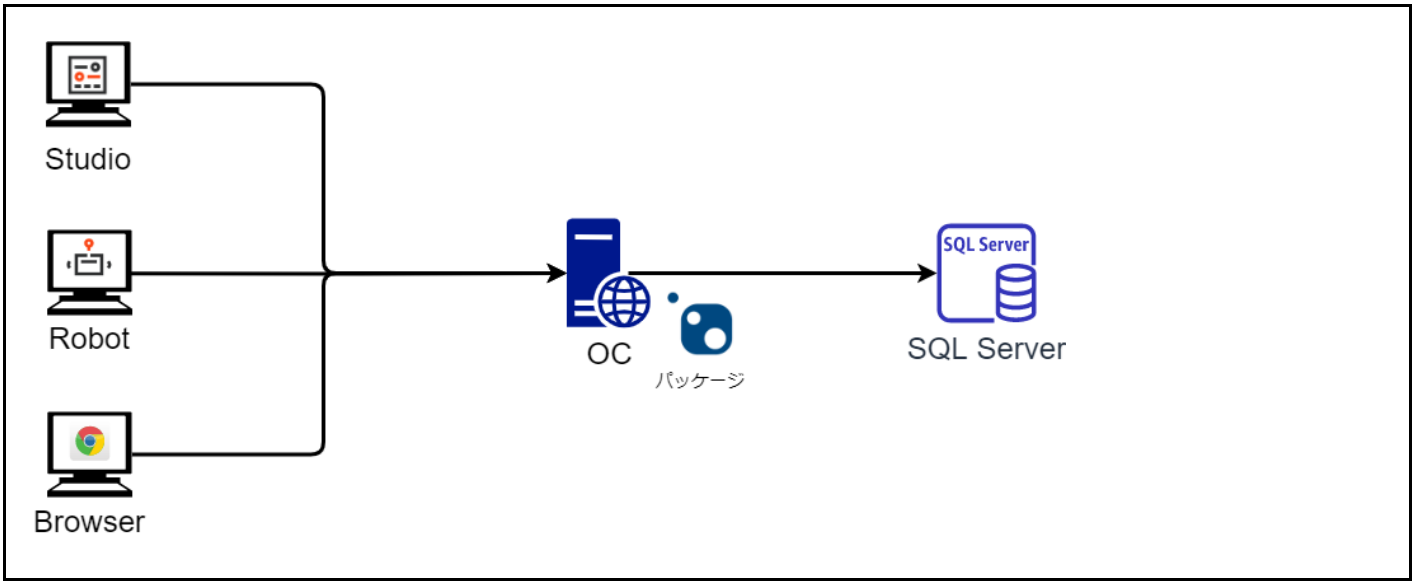
Elasticsearch で認証が有効化されている場合、Orchestrator からのアクセスを許可するため UiPath.Orchestrator.dll.config ファイルでの設定が必要となります。

- Elasticsearch でユーザー名とパスワードによって認証する場合の設定手順は ユーザー名とパスワードによる認証 をご参照ください。
- Elasticsearch の認証方式として OAuth を使用することも可能です。詳細な手順は OAuth 2.0 認証 をご参照ください。

2.4. オンプレミス構成例

- オンプレミス環境での Orchestrator の構成パターンの例を下記に示します。AWS/Azure 環境での構成パターンについては パブリッククラウドにおける Orchestrator 設計 をご参照ください。
- Active Directory は以下 AD と略します。

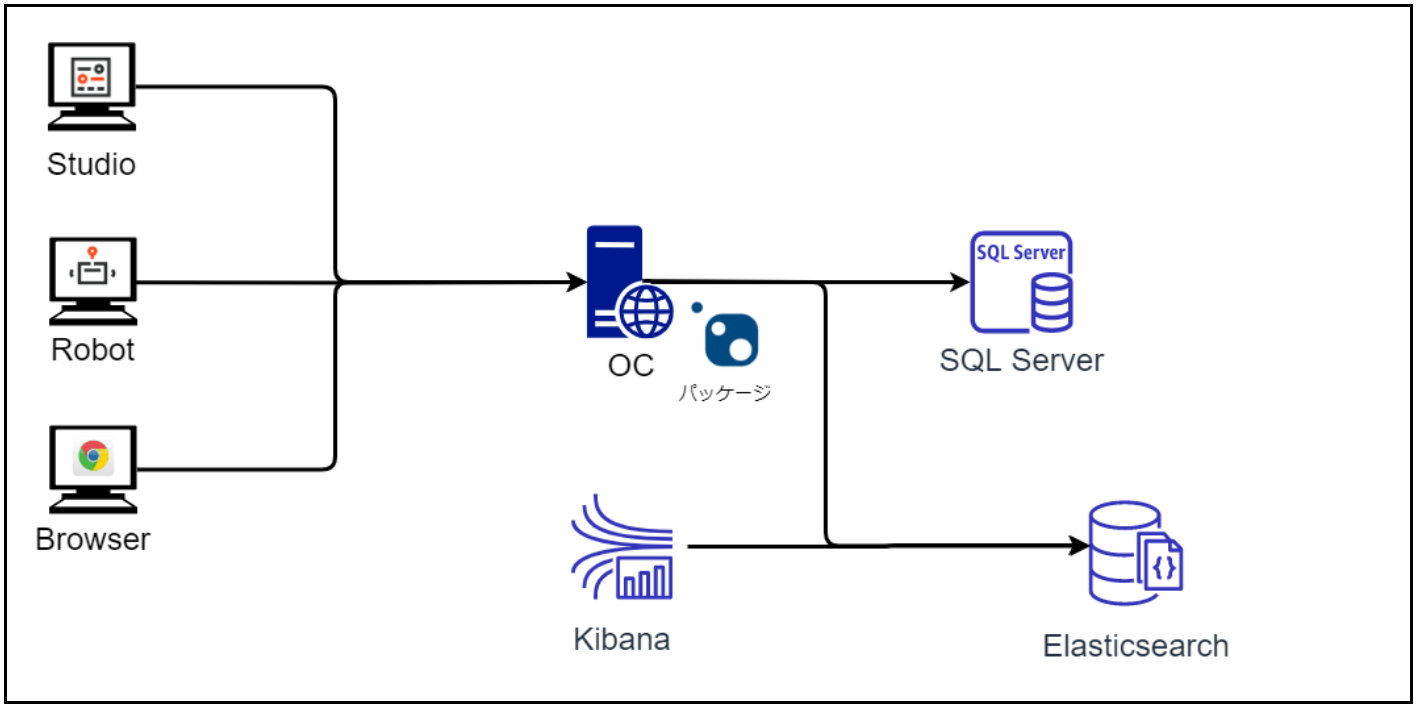
2.4.1. シングル構成



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

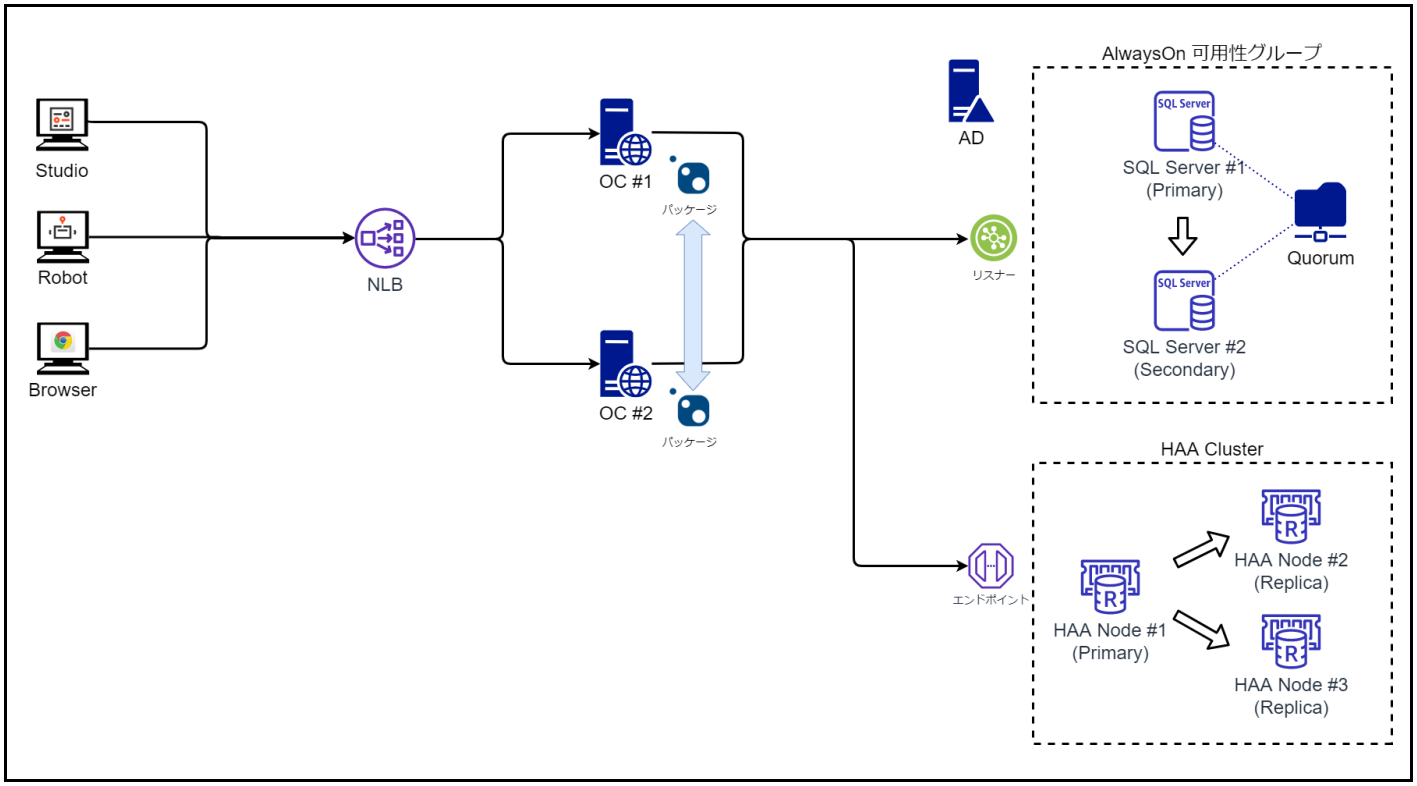
- Orchestrator を動作させるための最小構成 となります。
- Web サーバー・DB サーバーの計 2 台のサーバーを準備します。
- この環境は台数も少なく構築も簡単ですが、冗長化されていないためそれぞれのサーバーが単一障害点となるため 1 台のサーバーが停止すると Orchestrator の機能すべてが利用できなくなります。
- 開発環境や小規模な本番環境に向いた構成となります。
- 本番環境で利用する際には障害に備えてバックアップ・リストア手順を確立することを推奨します。
- マシンリソース不足の場合にはスケールアップ(マシンスペックの増強)で対応します。
- AWS での構成例 および Azure での構成例 はそれぞれのリンクを参照してください。

2.4.2. 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 SQL Server の負荷とメンテナンスコストを下げることができる	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

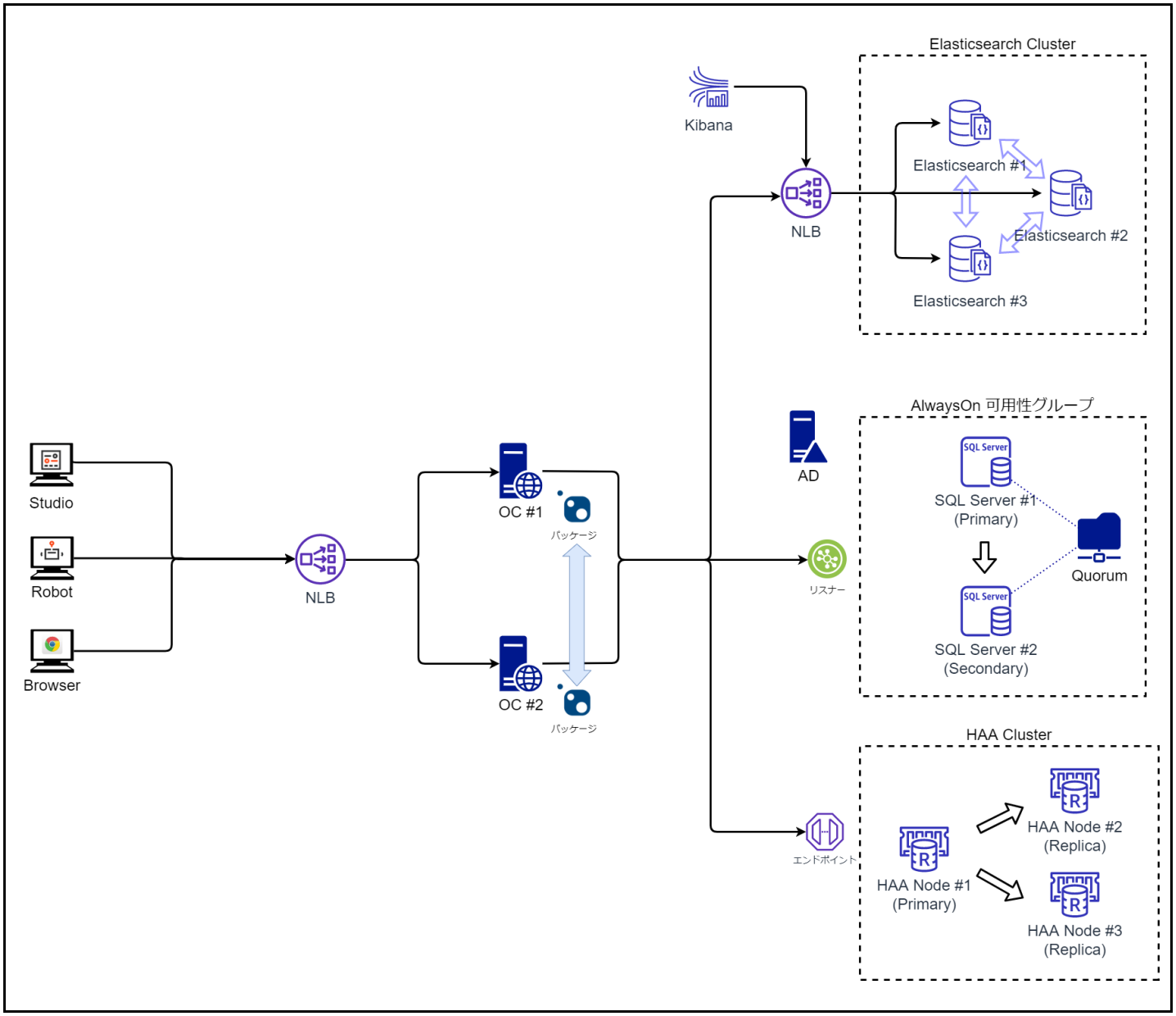
2.4.3. Active-Active 構成



AD	可用性	拡張性	長所	短所
必須	高	高	各サーバー冗長化により、障害時には自動フェイルオーバーが可能 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	構築・運用コストが高くなるため、安定稼働のためには構築・監視・運用の設計が重要となる 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 HAA (Redis) の動作環境として Linux が必要となる

- Active-Active 構成で構築するため、通常時 Web サーバー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 台のサーバーダウンであれば、もう 1 台のサーバーで継続して運用することが可能です。
- データベースは SQL Server の Always On 可用性グループ機能 による冗長化を利用することで、稼働系 SQL Server のダウン時にも待機系に自動切換えして継続運用することができます。
- DFS レプリケーションはロボットにてプロセス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。
- HAA (Redis) は複数の Orchestrator 間で情報更新の通知や、頻繁に参照されるデータベース内の一部データのキャッシュ、セッションの管理などに使用されます。サポートされる HAA (Redis) のバージョン等につきましては ソフトウェア要件 を参照してください。
- AWS での構成例 および Azure での構成例 はそれぞれのリンクを参照してください。

2.4.4. Active-Active 構成 + Elasticsearch



AD	可用性	拡張性	長所	短所
必須	高	高	単一障害点が除去され可用性・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 SQL Server の負荷とメンテナンスコストを下げることができる	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 HAA (Redis) の動作環境として Linux が必要となる

- ロボットを数百台以上管理する大規模運用での Orchestrator 構成例は 上図 となります。 大規模構成では最低でも UR 500 台ごとに 1 台の Orchestrator を追加します。
- 本構成例はロボット数百台を運用できるように、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 のサーバー台数を追加 (負荷分散)
  - ロボット実行ログを 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)
- サポートされる Elasticsearch/Kibana のバージョン等につきましては、ソフトウェア要件 を参照してください。

2.5. 各機能・コンポーネントの設計考慮点

2.5.1. Active Directory 設計

Orchestrator には Active Directory (AD) の機能が必要になる場合があります。それぞれのユースケースにおいて参考となるリンクを記載します。

検討項目	ユースケース	参考リンク
ユーザー認証	Orchestrator 管理画面に Windows 認証でログイン	Active Directory との連携を構成する
ユーザー管理	Orchestrator 管理画面へのログインユーザーまたはロボットの実行ユーザーを AD グループまたは AD ユーザーとして管理	アカウントの種類
サーバー証明書	サーバー証明書発行にドメイン証明機関を利用	サーバー証明書の展開 設定手順は Microsoft ドメイン証明機関(CA)によるサーバー証明書発行 を参照
Windows 認証による DB 接続	Windows 認証を使用した Orchestrator から SQL Server への DB 接続	Windows 統合認証
Storage ディレクトリ同期	冗長構成時の NuGet パッケージなどが保存された Storage ディレクトリ同期に DFS レプリケーションを利用	DFS レプリケーションの概要 設定手順は DFS レプリケーション設定手順 を参照
SQL Server 冗長化	Always On 可用性グループによる SQL Server の冗長化	Always On 可用性グループとは

2.5.2. ネットワーク設計

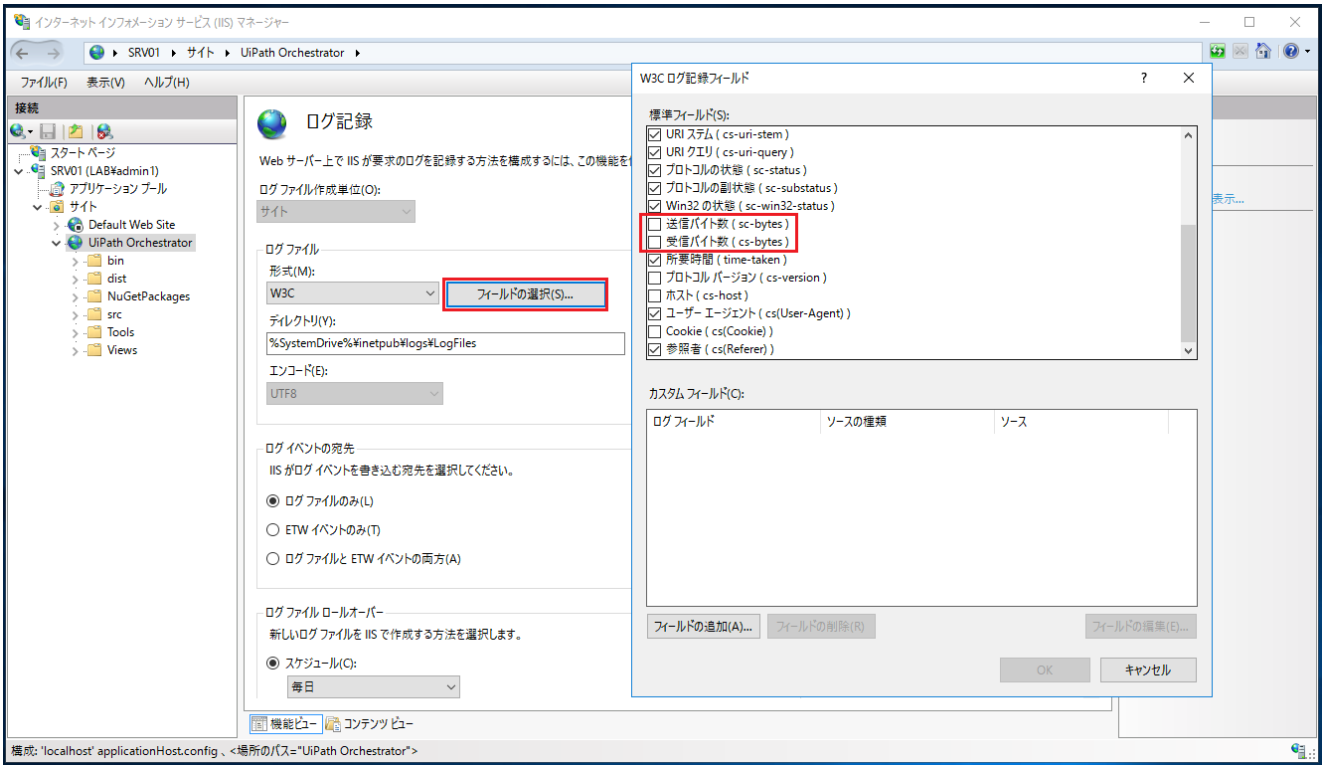
Robot ⇔ Orchestrator 間のトラフィック

必要に応じて Robot と Orchestrator 間のトラフィック量を見積もります。主に下記の種類のトラフィックが Robot マシン(またはブラウザー端末) と Orchestrator 間で発生します。

#	処理内容	送信データ量 (Byte)	受信データ量 (Byte)	頻度	注釈
1	パッケージ発行	ワークフロー実装に依存	600~800	Studio からパッケージをパブリッシュをするタイミング	ワークフローの実装によって大きく異なる。誤ってパッケージ内に不要なファイルやフォルダーが含まれていた場合、それらもパッケージとして含まれて発行されてしまうため注意が必要である。
2	パッケージダウンロード	パッケージ構造に依存	パッケージ構造に依存	新しいバージョンのパッケージが Orchestrator にアップロードされ、ロボットで初回実行されるタイミング	依存するアクティビティがローカルに存在しない場合には、合わせてダウンロードされます。 一度ダウンロードされたプロセスパッケージとアクティビティは ローカルディレクトリ (%UserProfile%\.nuget\Packages) にキャッシュされ、2 回目以降の実行で使用されます。
3	ハートビート	500 ~ 650	800 ~ 1000	30 秒ごと	Orchestrator 接続されている Robot マシンごとにハートビートを Orchestrator に対して送信する。 ロボットとして 1 マシンに 1 ユーザーが紐づいている場合の測定データ。 複数ユーザーが紐づいている場合にはユーザー数分のデータが送受信されるためデータ量が増大する。
4	「メッセージをログ」アクティビティによるログ出力	700 ~ 800	300 ~ 400	プロセス実行時に随時送信	出力メッセージが 50 半角英数字文字列の場合の測定データ。 これ以外にもプロセス開始と終了時にもそれぞれログが送信される。
5	ジョブステータス更新	500 ~ 2000	300 ~ 400	プロセス実行時	ステータスによってデータ量が変化。
6	「トランザクションアイテムの取得」ア	400 ~ 500	1300 ~ 1400	キューを使用しているプロセス実行時	レスポンス結果のキュー(トランザクション)データに 50 半角英数字文字列が含まれる場合の測定データ。

	クティビティの実行				
7	「アセットを取得」アクティビティの実行	400 ~ 500	800 ~ 900	アセットを使用しているプロセス実行時	レスポンス結果のアセットデータが 50 半角英数字テキストデータの場合の測定データ。
8	Web UI 操作	操作内容に依存	操作内容に依存	ユーザーが画面操作をするごと	

実際のトラフィック量はロボット台数、ワークフローの実装、プロセスの実行頻度などによって大きく異なるため、IIS ログの送信/受信バイトの記録を有効化し、実測値に基づいて算出していただくことを推奨します。

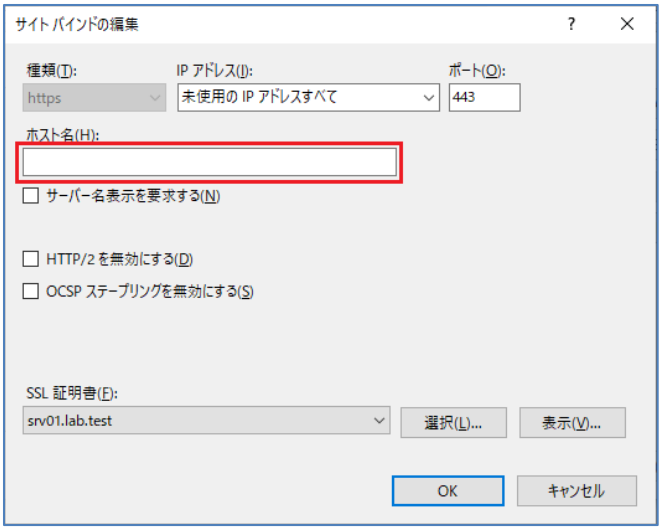


ロードバランサー

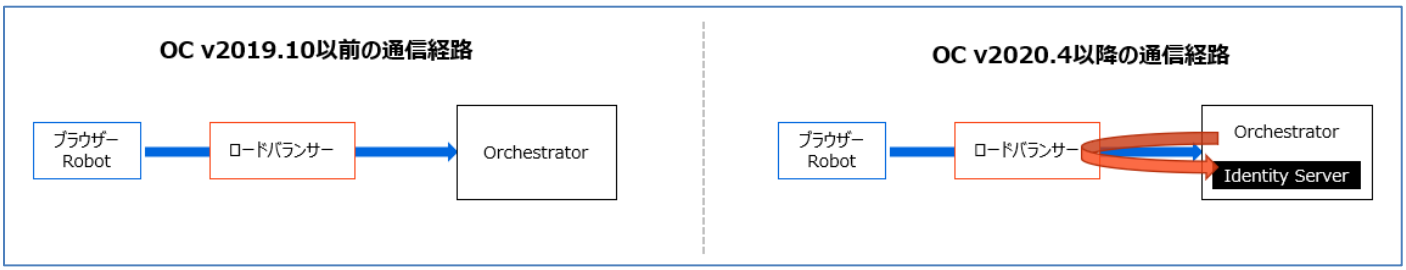
冗長構成の Orchestrator 環境において適切に負荷分散や自動フェイルオーバーするためにネットワークロードバランサーが必要となります。ロードバランサー設定におけるポイントは下記の通りです。

- Orchestrator を 2 台以上の Active-Active 構成で構築する際には、Robot/ブラウザーと Orchestrator 間にロードバランサーが必要となります。また Elasticsearch を冗長化している場合には、ロードバランサーは Orchestrator → Elasticsearch の負荷分散のために必要となります。

- 冗長構成にてロードバランサー経由でアクセスする場合でも、ロードバランサーでは SSL 終端させず IIS で終端 (SSL パススルー) するようサーバー証明書は各 Orchestrator ホストにてインストールします。
  - ロードバランサーで SSL 終端し、IIS との間を HTTP で通信する SSL オフロードはサポートされません。
  - ロードバランサーで SSL 終端し、IIS との間を HTTPS で通信する SSL ブリッジはサポートされますが、ロードバランサーの仕様によっては Windows 認証などの資格情報が正常に Orchestrator まで渡されず Orchestrator 管理画面に正常にログインできなくなる可能性があります。
- HAA は DNS サーバーと連携して一つのエンドポイントを使用して動的な名前解決によりマスター切り替えを行う機能を備えているため、Orchestrator → HAA 接続においてロードバランサーは不要です。
- Orchestrator、Elasticsearch 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で問題ありません。またすべての通信はステートレスであるためパーシステンス (アフィニティまたはスティッキーセッションとも言う) は設定する必要がありません。
- 異常が発生したノードへの振り分けを停止するために、下記のようにヘルスチェックをそれぞれ定期的に行うようにします。各ポート番号を既定値から変更している場合には適宜変更します。

機能	ヘルスチェック方法
Orchestrator	HTTPS でポート 443 に対して **GET /api/health** を送信し **200 OK** が返されることを確認する。 ※ UiPath Orchestrator サイトのサイトバインド設定でホスト名が空白または*であることを確認します。ホスト名が指定されている場合、このホスト名以外を使用する HTTPS リクエストがすべて 404 エラーとなり、ヘルスチェックが失敗します。 
Elasticsearch	HTTP(または HTTPS)でポート 9200 に対して **GET /** を送信し **200 OK** が返されることを確認する

- Orchestrator v2020.4 以降、Orchestrator のユーザー認証・認可には Identity Server が使用されます。Identity Server は Orchestrator に内包されたサービスとして動作しますが、Orchestrator は Identity Server のクライアントとして動作するため内部通信が発生します。
  - ロードバランサー環境では、既存の通信経路に加えてバックエンドプールとしての Orchestrator ホストからロードバランサーを経由で自分自身に接続できることを確認します。(下図参照)



  - Azure Load Balancer ではバックエンドプールとの通信に制約があります。
    - 参考: Azure Load Balancer のバックエンド トラフィック応答に関するトラブルシューティング
  - ネットワーク環境の制約により接続できない場合には Orchestrator Web サーバーの hosts ファイル (C:\Windows\System32\drivers\etc\hosts) に下記エントリを追加し、ロードバランサー経由せずに自分自身に接続します。

```
127.0.0.1 <Orchestrator FQDN(例: rpa.lab.test)>
```

※ IP バインドしている場合には 127.0.0.1 の代わりに IP アドレスを指定します。

IIS 接続タイムアウトの調整

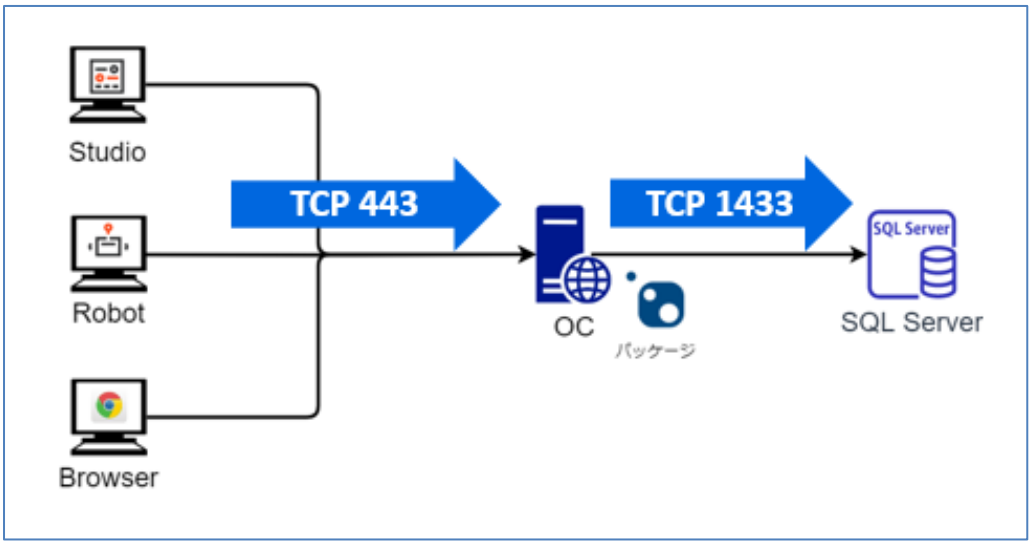
ロードバランサー配下において Orchestrator を 2 台以上の Active-Active 構成で構築する場合、IIS の接続タイムアウトを既定値 120 秒から **20** 秒に変更することを推奨します。理由は以下の通りです。

- 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事象が発生します。
- 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 種類のコネクション (HTTP リクエストおよび SignalR) を利用しますが、SignalR 接続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 SignalR 接続の偏りが発生した後、片系が復旧しても接続の偏りは復旧しませんが、SignalR 接続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。

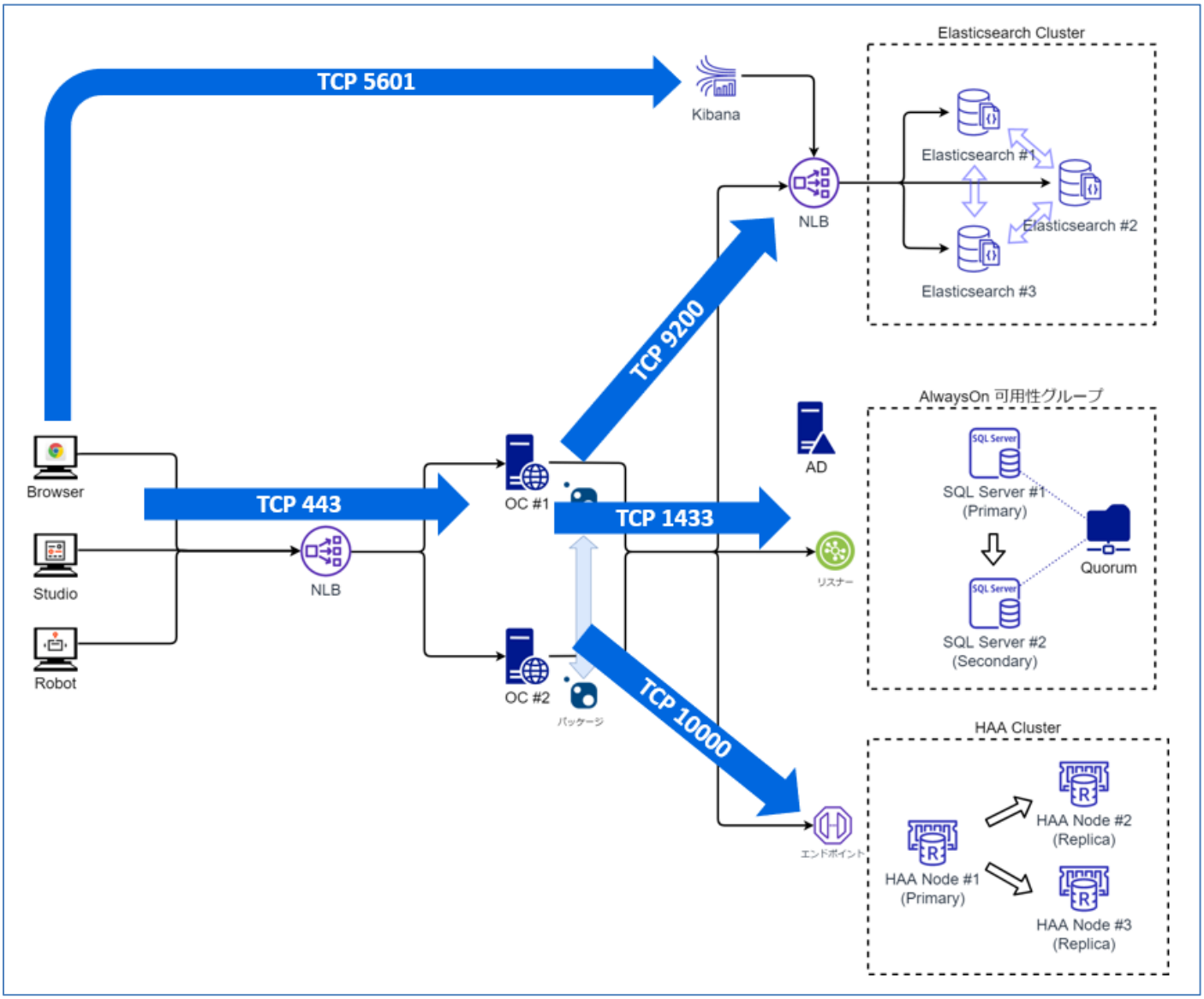
ファイアウォール

Orchestrator を構築・運用するにあたり通信が必要となるポートは次の通りです。

シングル構成



Active-Active 構成 + Elasticsearch



いずれかのコンポーネント間にファイアウォールが存在する場合は、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。 ポート番号を既定値より変更している場合には適宜変更します。

接続元	接続先	プロトコル (既定ポート)	注釈
各 Studio/Robot/ ブラウザー端末	Orchestrator Web サーバー	HTTPS (TCP443)	プロキシサーバー無しを想定 逆向きのファイアウォールポート開放は不要、NAT ルーター越えは問題なし
Orchestrator Web サーバー	SQL リスナーまたは SQL Server ノード	MSSQL (TCP1433)	SQL リスナーは Always On 可用性グループによる冗長構成で割り当てられる仮想 IP
Orchestrator Web サーバー	Elasticsearch	HTTP または HTTPS (TCP9200)	実行ログ保存先として使用している場合 SSL/TLS 有効化されている場合は HTTPS 接続となります
管理者端末	Kibana	HTTP (TCP5601)	実行ログ保存先として Elasticsearch を使用し、ログ分析のために Kibana を使用している場合 SSL/TLS 有効化されている場合は HTTPS 接続となります
Orchestrator Web サーバー	HAA または Redis サービス	RESP (TCP10000)	Active-Active 冗長構成の場合に必要となります HAA 以外の Redis サービスを利用している場合はポート番号を確認します

上記以外にも Orchestrator 構成や利用する機能・サービスに応じて追加設定が必要となるケースがあります。詳細はそれぞれのガイドをご参照ください。

- Orchestrator
- SQL Server Always On 可用性グループ
- Elasticsearch
- HAA

2.5.3. Storage ディレクトリ

- Storage ディレクトリには、主に Studio からパブリッシュされたプロセス用 NuGet パッケージおよびライブラリー用カスタムアクティビティ(共通部品)と、インストール時にコピーされる既定のアクティビティパッケージが配置されます。
- Storage ディレクトリは NuGet パッケージ以外にも Orchestrator 機能であるストレージバケットや実行メディア(記録機能)のデータも格納されます。これらのデータはまずテナントごとに分離され、さらに機能ごとにサブディレクトリで分離されます。実際のディレクトリ構造は使用される機能・設定などの利用状況により異なりますが、一般的な例を下図に示します。

Storageディレクトリ構造

テナント	ディレクトリパス	説明
ホストテナント	Orchestrator-<HostTenantKey> Libraries	➢ デフォルトでインストールされるアクティビティ＋カスタム共通部品
defaultテナント	Orchestrator-<DefaultTenantKey> Processes <Project01> ▪ <project01>.1.0.1.nupkg ▪ <project01>.1.0.2.nupkg … <Project02> … BlobFilePersistence ExecutionMedia	➢ NuGetパッケージ ➢ パッケージ名 (Studioでのプロジェクト名) ▪ <パッケージ名>.<バージョン>.nupkg … ➢ ストレージバケットのデータ ➢ 実行メディア(記録機能)のデータ
(追加テナント01)	Orchestrator-<CustomTenantKey01> Processes BlobFilePersistence ExecutionMedia	追加テナントの構造はdefaultテナントと同様
…		

- Orchestrator がシングル構成の場合には Storage ディレクトリはローカルディレクトリに配置しても問題ありません。Orchestrator が冗長構成の場合には、次のいずれかの方法を検討します。
  - ➢ すべての Orchestrator ノードがアクセス可能なファイルサーバーを UNC パスで指定します。ファイルサーバーにアクセス可能なユーザーを Orchestrator のアプリケーションプール ID として指定し、適切な権限 (変更/読み取りと実行/フォルダーの内容の一覧/読み取り) を付与します。既存ファイルサーバーも利用できるため実装しやすいが、ファイルサーバーそのものが冗長化されていない場合、単一障害点となります。
  - ➢ Amazon S3、Azure Blob ストレージ、S3 互換ストレージも使用可能です。一般的にパブリッククラウドのストレージサービスはそれ自身が高可用性となっているため、ユーザー自身で冗長化の仕組みを実装する必要はありません。クラウドでは前述のストレージサービスに加えて Amazon FSx や Azure Files といった Windows ファイルサーバーのサービスも利用することが可能です。その場合はファイルサーバーに読み書きを行うためのアプリケーションプール ID の権限 (Windows インストーラー → [アプリケーション プール設定]) に注意します。
  - ➢ 各 Orchestrator ノードではそれぞれのローカルディレクトリを Storage ディレクトリとして指定し、DFS レプリケーションを使用して各ノードのディレクトリを同期します。DFS を使用するには

Orchestrator ノードを AD に参加させる必要があります。DFS レプリケーションの実装方法については DFS レプリケーション設定手順 をご参照ください。

- Orchestrator v2020.10 以降、Storage ディレクトリは下記の構成となります。
  - NuGet パッケージ本体は UiPath.Orchestrator.dll.config の Storage.Location 設定に応じてテナントごとに保存されます。(既定値は相対パス: .\Storage, 絶対パス: C:\Program Files (x86)\UiPath\Orchestrator\Storage)
  - NuGet パッケージのメタデータは “UiPath” データベースのテーブル内に保存されます。
  - Orchestrator インストーラーに含まれる既定のアクティビティパッケージは UiPath.Orchestrator.dll.config の Storage.Location 設定に応じてホスト共通で保存されます。設定変更 によってテナントごとに保存することも可能です。
  - Storage ディレクトリを移行する際には、単純なファイルコピーでは移行できません。Orchestrator 管理画面のパッケージまたはライブラリー画面を使用して *.nupkg ファイルを個別にアップロードします。多数のパッケージを一括してアップロードするには、ファイルのアップロードサイズ制限の値を増加 させます。

2.5.4. IIS / Orchestrator

Orchestrator は IIS 上のアプリケーションプールおよび Web サイトとして動作し、必要に応じて IIS 側の設定を調整します。

アプリケーションプールリサイクル

アプリケーションプールのリサイクルのタイミングを考慮する必要があります。 デフォルトは **1740** 分 **(29** 時間**)** ごと になっています。 アセット、キューを使用するプロセス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。 そのためプロセスが実行されていない時間帯にリサイクルまたは IIS の再起動を行います。設定変更手順は アプリケーションプールリサイクル設定 をご参照ください。

SQL Server コネクションプール制限値の変更

Orchestrator から SQL Server にクエリを送信する際にはコネクションプール(接続プール)を使用し、DB 接続を維持・再利用することによりパフォーマンス改善を図っております。

- 参考: SQL Server の接続プール (ADO.NET)

Orchestrator からの DB 接続が発生するケースは様々あり、例えば下記のような場合です。

- Robot サービスおよび Assistant 起動時に Orchestrator からライセンス情報などを取得
- Robot から Orchestrator へのハートビード (既定値では 30 秒に 1 回送信)
- Robot でプロセス実行時に送信されるステータス情報
- Robot でプロセス実行時に送信されるログ
- Robot によるキューやアセットの処理
- ブラウザーによる Orchestrator 管理画面の表示
- Orchestrator API 呼び出し

コネクションプール制限値は既定値では 100 となっており、Robot 台数の増加に伴って上限に達する可能性があります。上限に達した場合には、任意の既存クエリ実行が終了するまで、新規 SQL クエリが待機状態となり、30 秒以上経過するとタイムアウトされ処理が失敗します。

この問題を回避するために UiPath.Orchestrator.dll.config にて上限値を 1000 に変更することを推奨いたします。

```
<connectionStrings>
  <add name="Default" providerName="Microsoft.Data.SqlClient" connectionString="Data Source=***;Initial
Catalog=UiPath;User ID=***;Password=***;Max Pool Size=1000" />
</connectionStrings>
```

上限値の変更によって Orchestrator Web サーバーのメモリ使用量が増加することはありませんが、数多くの SQL クエリの並行処理によるパフォーマンスは SQL Server の性能に依存します。

なお providerName は Orchestrator v2021.10.3 以降は Microsoft.Data.SqlClient、v2021.10.2 以前は System.Data.SqlClient が使用されます。

SignalR 接続

SignalR は ASP.NET 上に構築されたライブラリーであり、双方向リアルタイム通信を実現します。Robot⇔Orchestrator 間の SignalR 接続は、Unattended Robot への手動またはトリガーによるジョブ実行を行う際、Orchestrator は接続済みの SignalR チャネルを使用してリアルタイムで Robot に対してジョブ実行命令を送信します。

Robot が Orchestrator へ接続を試みるとき、次の 3 つの方法を順に試行します:

① WebSocket
② Server-Sent-Events
③ Long Polling

まず①で接続を試み、①が失敗した場合は②で、②が失敗した場合は③で接続を試行します。接続が成功すれば、その後は継続してその接続方法が使用されます。Robot サービスを再起動した場合や Orchestrator 接続を再設定した場合には再度①から接続が試行されます。

③Long Polling は Robot が Orchestrator に接続するための最終手段ですが非効率的で高負荷であるため、禁止することを推奨します。

Windows 7 または Windows Server 2008 R2 上に Robot が存在する場合は OS の制約として WebSocket が利用できません。 Long Polling のみを禁止して WebSocket と Server-Sent-Events に限定します。結果的に Server-Sent-Events のみが利用されるようになります。

SignalR 接続は UiPath.Orchestrator.dll.config または Orchestrator 管理画面上で設定することができます。

- スケーラビリティ
- SignalR (Robot)

Unattended Robot を使用していないケースには SignalR 接続そのものを無効化することも検討します。

SignalR 接続を無効化することによる影響は下記の通りです。

- Unattended Robot に対して手動またはトリガーによってジョブ実行した場合、SignalR の未接続によって最大 30 秒の遅延が発生しますがジョブは開始されます。
- Unattended Robot のスケジュール実行は定期的なハートビートの応答によってプロセス名と実行時刻を Robot が受信するため、SignalR 接続は使用されません。

なお Orchestrator 管理画面でのリアルタイム表示のためにブラウザー⇔Orchestrator 間でも SignalR 接続が使用されますが、この接続を設定によって無効化することはできません。上記設定は Robot⇔Orchestrator 間の SignalR 接続のみに適用されます。

パッケージファイルの上限

パッケージ (ワークフローおよびアクティビティ) の 1 回アップロード当たりの最大ファイルサイズは IIS の Web.config 設定ファイルで定められておりますが既定値は Orchestrator バージョンにより異なります。

- v2020.10 以降 – 314,572,800 バイト (300MB)
- v2020.4 以前 – 30,000,000 バイト (およそ 28.6MB)

この上限を超えるファイルをアップロードする場合は、IIS が許可するアップロードファイルのサイズを増加させます。

UiPath.Orchestrator.dll.config 設定

Orchestrator サイトの様々なアプリケーション設定を行うことができます。 設定可能な項目は UiPath.Orchestrator.dll.config を参照してください。

またログの設定については ログの構成 セクションを参照してください。

テナント設定

Orchestrator インストール後、管理画面から設定を行う項目があります。 詳細は テナント設定 を参照してください。

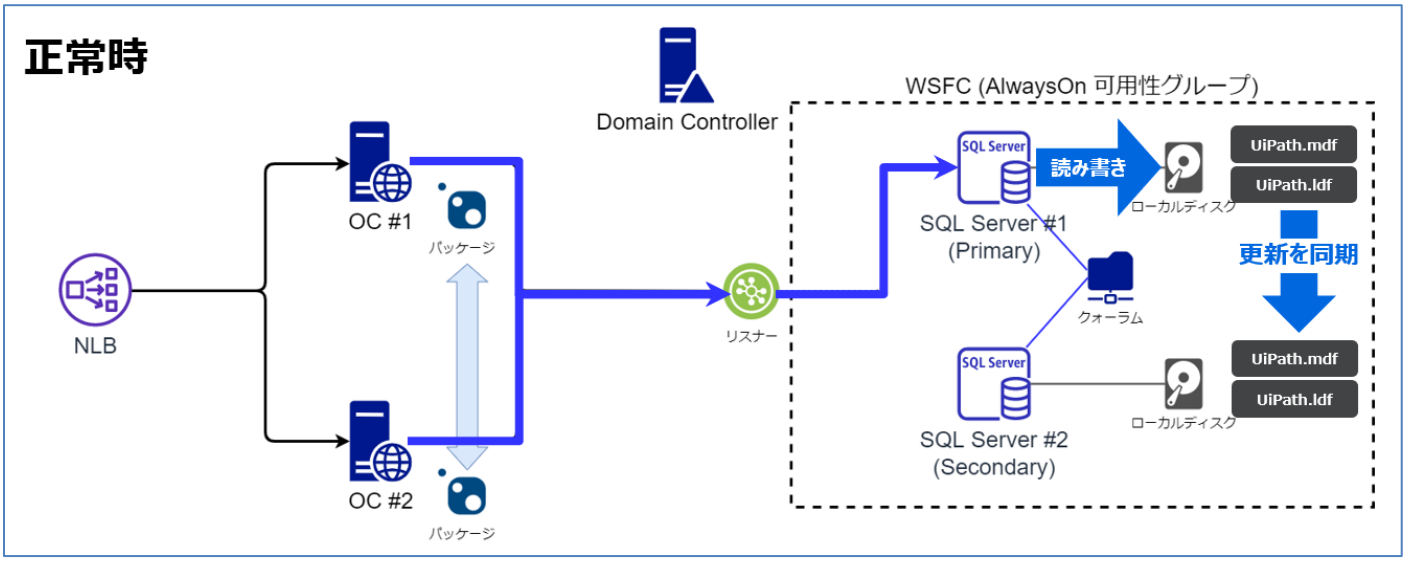
2.5.5. SQL Server

冗長構成

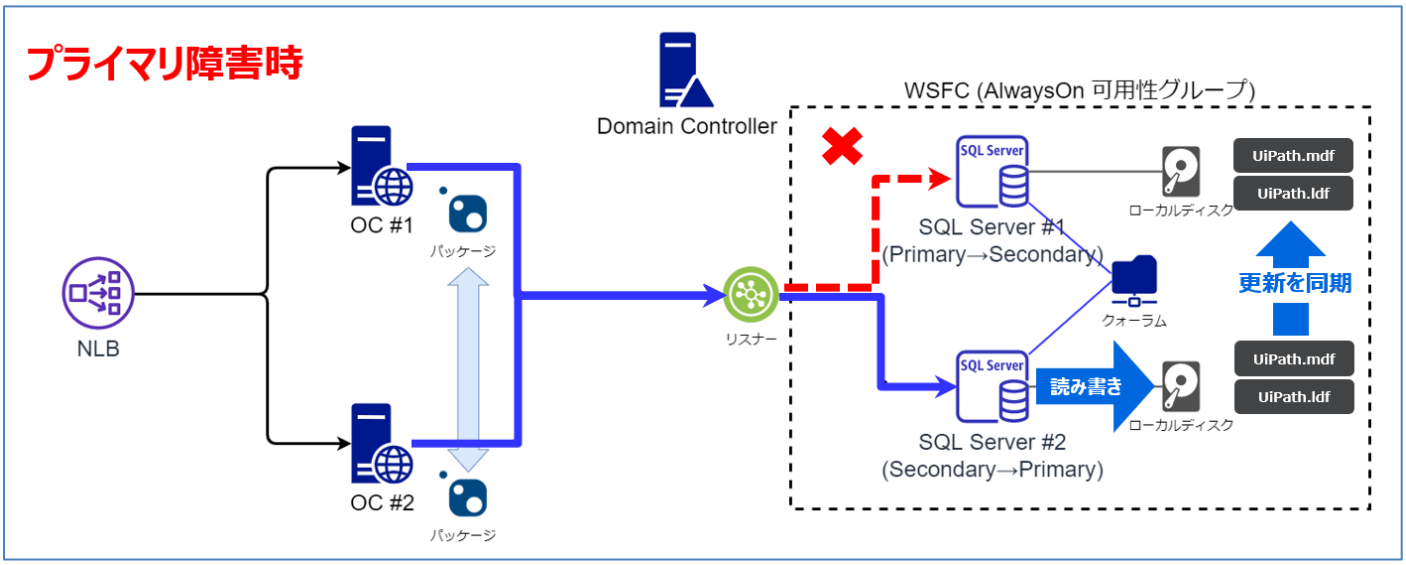
SQL Server を冗長化するには Always On 可用性グループ を構成することを推奨します。

可用性グループの作成により DB 接続先はリスナーと呼ばれる仮想 IP に固定することができます。

正常時に Orchestrator からはリスナー経由でプライマリ DB に読み書きされます。DB への更新はプライマリ DB からセカンダリ DB へ同期されます。



プライマリ障害時には、自動的にセカンダリがプライマリに昇格し、リスナーは新しいプライマリをポイントするため、Orchestrator で DB 接続先を変更する必要はありません。



自動拡張設定

SQL Server ではデータ領域およびトランザクションログ領域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 Server 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。

最大サーバーメモリ設定

SQL Server は稼働時間経過に伴いメモリ使用量が増え続けます。その際、OS や他のプログラム・サービスとメモリ使用の競合や取り合いを起こさせないために SQL Server が使用するメモリ量の上限を設定することが推奨されます。

設定手順については Orchestrator 管理者のための ミドルウェア運用設定ガイド Ⅰ の **4.2.1** メモリの設定 をご参照ください。

その他の設定項目

SQL Server 設定のその他の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 Ⅰ をご参照ください。

2.5.6. Elasticsearch/Kibana

実行ログ格納先

Elasticsearch を導入することにより、プロセス実行ログの格納先を SQL Server から Elasticsearch に変更し、SQL Server の負荷とメンテナンスコストを下げることができます。実行ログは Elasticsearch のみに書き込むことができますが、必要に応じて SQL Server と Elasticsearch の両方に書きこむことも可能です。ログ量と格納先によって、SQL Server および Elasticsearch の必要ディスク容量を見積もります。

インデックス

Elasticsearch のデータはインデックスとよばれる単位で論理的に管理されており、デフォルトは 1 ヶ月単位になっています。 大規模展開において実行ログ量が膨大となる場合には、インデックスを 1 日単位で生成するように UiPath.Orchestrator.dll.config で設定することもできます。

```
<target xsi:type="ElasticSearch" name="robotElastic" uri="http://elasticsearch-ip:9200" requireAuth="false" username="" password="" index="${event-properties:item=indexName}-${date:format=yyyy.MM.dd}" documentType="logEvent" includeAllProperties="true" layout="${message}" excludedProperties="agentSessionId,tenantId,indexName" />
```

ただし日付は UTC の時刻となります。つまり日単位の設定した場合、日本時間の午前 9 時に毎日新しいインデックスに切り替わることに注意してください。

シャード

Elasticsearch の物理データはシャードと呼ばれる単位で管理されており、インデックスサイズに応じて最適なシャード数を設定します。シャード数によってパフォーマンス低下を招く可能性があります。

一般的にはインデックスサイズが 30GB を超える場合には 2 つ以上に分割します。具体的にはインデックスサイズに応じて下記のように変更します。

- 30GB 未満の場合には 1
- 30GB 以上、60GB 未満の場合には 2
- 60GB 以上、90GB 未満の場合には 3
- インデックスサイズは月単位か日単位かでサイズが大きく異なる点に注意してください。
- 冗長構成においてはデータの冗長性を担保するために上記のシャード (プライマリシャード) とは別にレプリカシャードを 1 つ設定します。
- シャード数の変更手順を含めた最適な Elasticsearch のテンプレートについては UiPath ナレッジベースの「Elasticsearch Index テンプレート」をご参照ください。

冗長構成

Elasticsearch を冗長化するには、少なくとも 3 台 (3 ノード) のクラスター構成にします。

各ノードはマスター候補ノードやデータノードなどの役割を持たせることができますが、Orchestrator からの実行ログ処理においてはサーバーリソースを有効活用するためにマスター候補ノード兼データノードを 3 台準備することを推奨します。

障害発生時には各ノードがマスターノード選出を投票し過半数 (3 ノード構成の場合は 2) を獲得したノードが新たなマスターノードに昇格します。

2 台構成の場合にはネットワーク障害時などに 2 台ともマスターノードに昇格し「スプリットブレイン」と呼ばれる状態に陥りクラスターとして正常に機能しなくなる可能性があります。

冗長構成においては Orchestrator と Elasticsearch ノード間の通信を負荷分散するためにロードバランサーが必要となります。振り分けやヘルスチェックの設定については ネットワーク設計 > ロードバランサー をご参照ください。

UiPath.Orchestrator.dll.config に複数の Elasticsearch の URI を指定することも可能ですが、この場合には 1 号機がダウンすれば 2 号機に、2 号機がダウンすれば 3 号機にログが送信される仕組みとなっており、3 ノード均等にログ送信はされません。

Kibana ダッシュボード作成

Elasticsearch に格納された実行ログデータを使用して、Kibana でビジュアライズされたダッシュボードを作成することができます。UiPath ナレッジベースでは、Kibana 活用をすぐにでも始められるように Kibana ダッシュボードのテンプレートを公開しています。Template ファイル (.json 形式) をダウンロードし、Kibana 画面でインポートすることによって標準的な分析が可能となります。

Kibana の操作方法や分析手法等、トレーニングも提供しています。詳細は UiPath 社にお問い合わせください。

その他の設定項目

Elasticsearch/Kibana 設定のその他の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 II をご参照ください。

2.5.7. HAA (Redis)

HAA または Redis は 1 ノードでの運用も可能ですが、単一障害点となるため、HAA または Redis そのものも冗長化することを推奨します。

冗長構成

HAA で冗長構成を構築するには 3 台のサーバーを準備し、HAA クラスターを作成し、メンバーサーバーとして 3 台のノードを追加します。詳細な手順は HAA インストール をご参照ください。

Orchestrator から HAA への接続には下記のいずれかの方法を採ることができます。

- 3 台の HAA サーバーの IP アドレスを UiPath.Orchestrator.dll.config に設定します。

```
<add key="LoadBalancer.Redis.ConnectionString"
value="10.10.20.184:10000,10.10.24.148:10000,10.10.22.114:10000" />
```

- DNS エントリの設定 により DNS サーバーにて固定の接続エンドポイントを作成し UiPath.Orchestrator.dll.config にて接続先を設定します。HAA プライマリノードへの接続は DNS サーバーによる接続エンドポイントの名前解決によって自動的に行われます。

```
<add key="LoadBalancer.Redis.ConnectionString" value="mycluster.mydomain.com" />
```

いずれの方法でも HAA プライマリノードがダウンした際には他のメンバーサーバーが自動的にプライマリに昇格し、Orchestrator は自動的に再接続します。つまり UiPath.Orchestrator.dll.config の再設定は不要です。

HAA (Redis) 接続の最適化

Orchestrator から HAA (Redis) の接続が切断された時に自動再接続するように UiPath.Orchestrator.dll.config の下記箇所を変更することを推奨します。

```
<add key="LoadBalancer.Redis.ConnectionString" value="haa-server:10000,abortConnect=false " />
```

なお Orchestrator の旧バージョンでは接続タイムアウトの既定値は 1 秒となっていたためネットワーク瞬断によってタイムアウトエラーが発生する可能性があり、この問題を回避するためタイムアウトを 5 秒に設定(responseTimeout=5000)することを推奨しておりましたが、最近のバージョンでは既定値が 5 秒に変更されたためこの設定は不要となりました。

その他の設定項目

HAA (Redis) 設定のその他の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 II をご参照ください。

2.6. Orchestrator ユーザー

Orchestrator 管理者として Orchestrator 管理画面にログインを行うユーザー、および Studio/Robot の実行ユーザーはあらかじめ Orchestrator 上で登録・設定が必要となります。

2.6.1. アカウント管理

アカウントの種類 はローカルアカウント・ディレクトリアカウント・ローカルグループ・ディレクトリグループがあり、どの方式を利用するか事前に検討します。

ディレクトリサービスは Active Directory および Azure AD と連携することが可能です。これらを利用することによりユーザー管理とユーザー認証をそれぞれのディレクトリサービスで行うことにより権限管理・運用のコストを削減することが可能になります。

それぞれのディレクトリサービスと Orchestrator との連携には下記の注意点があります。

- **Active Directory** … Orchestrator と連携可能なドメインは、Orchestrator サーバーが所属しているドメインおよびそのドメインと双方向の信頼関係にあるドメインです。Orchestrator テナントごとに連携するドメインを別々に指定することはできません。
- **Azure AD** … Orchestrator のテナント単位で連携する Azure AD テナントを設定できます。つまり Orchestrator をマルチテナントで運用している場合にはそれぞれの Orchestrator テナントで連携する Azure AD テナントを設定することが可能です。

2.6.2. ユーザー認証

Orchestrator 管理画面へのログイン、および Studio/Robot で対話型サインインを利用する際には Orchestrator 設定により下記の方式 (併用可能) でユーザー認証が行われます。

認証方式	特徴
基本認証	● ユーザーID・パスワードでログインする方法。 ● 実装は最も容易ですがパスワード漏洩によって成りすましのリスクがあるためセキュリティレベルは高くありません。 ● 他の認証方式を有効化し、基本認証を制限することもできます。
Windows 認証	● ドメインユーザー・パスワードでログインする方法。設定によっては自動ログインも可能です。 ● アカウントとパスワードを AD 側で一元管理し、既存のセキュリティポリシーを適用することが可能です。
Azure AD 認証	● Azure AD の SSO を設定することにより、Azure AD 認証を利用して Orchestrator にログインすることができます。ただしこの方法では Azure AD ユーザーは同期されません。ユーザー同期も行うには Enterprise SSO を設定 します。 ● Azure AD で多要素認証が有効化されている場合、Orchestrator ログインにも適用できるというメリットがあります
SAML 認証	● SAML の SSO を設定することにより、標準的な SAML2.0 に対応した外部プロバイダーを利用して Orchestrator にログインすることができます。

2.7. パブリッククラウド環境における Orchestrator 設計

前述の通り Orchestrator はパブリッククラウド環境でも構築・運用することが可能です。クラウド環境では展開スクリプトなどを利用してサーバー構築にかかる工数を削減し、マネージドサービスを利用して冗長構成を比較的容易に構築・運用することも可能になります。あるいは運用後のサイジング変更や、検証環境を必要な時に構築し、不要になればクリーンナップするなど、オンプレミスに比べて柔軟性があります。一方、アクセスコントロールを適切に行わなければ知らず知らずのうちに Orchestrator がインターネットに外部公開されるというリスクもあるため、クラウド固有の設計にも留意する必要があります。

2.7.1. クラウド利用の注意点

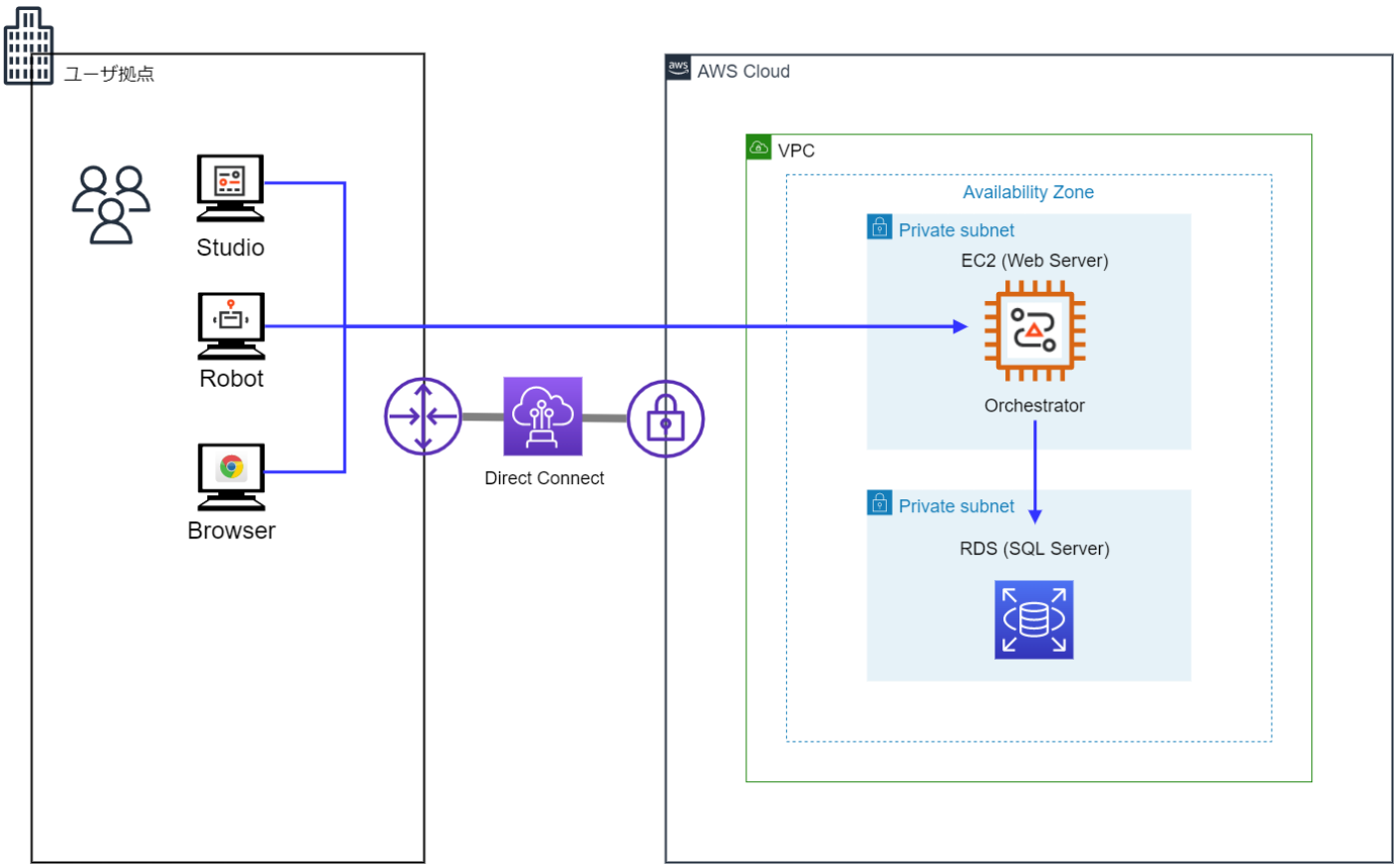
- Orchestrator 環境を構築する際にオンプレミスに構築するかクラウドに構築するかで悩むことがあるかもしれません。一般的なクラウドのメリット・デメリットは Orchestrator 環境についてもそのまま適用されます。UiPath 固有の検討ポイントとしては、「業務システム」「Robot 端末」「Orchestrator」をどのロケーションに配置し、ロケーション間をどのように繋ぐのかの検討が必要な点です。セキュリティ要件によっては、自社ネットワークとクラウド間を専用線(閉域網)で接続する必要があるかもしれません。ロボット端末は自動化対象の対向システム(業務システム)と Orchestrator の双方と通信できる必要があることに留意してください。
- クラウドサービスでは Orchestrator 環境で利用するコンポーネントのいくつかがマネージドサービス(PaaS) として提供されています。マネージドサービスを利用することで構築・運用の手間が削減される一方、OS へのログインができない、詳細なログが取得できない、ミドルウェアのバージョンが固定できないなどのデメリットもあります。要件に合わせてマネージドサービス利用の可否を決定してください。
- クラウドサービス上のインフラの設計・構築・運用はオンプレミスと共通する領域もありますが、異なる考え方やスキルが必要となる部分もあります。もし自社内にクラウドについての十分なノウハウが蓄積されていない場合、クラウドベンダーが認定するパートナー利用の検討をおすすめします。

2.7.2. クラウド構成例

AWS シングル構成例

AWS 環境でのシングル構成例です。

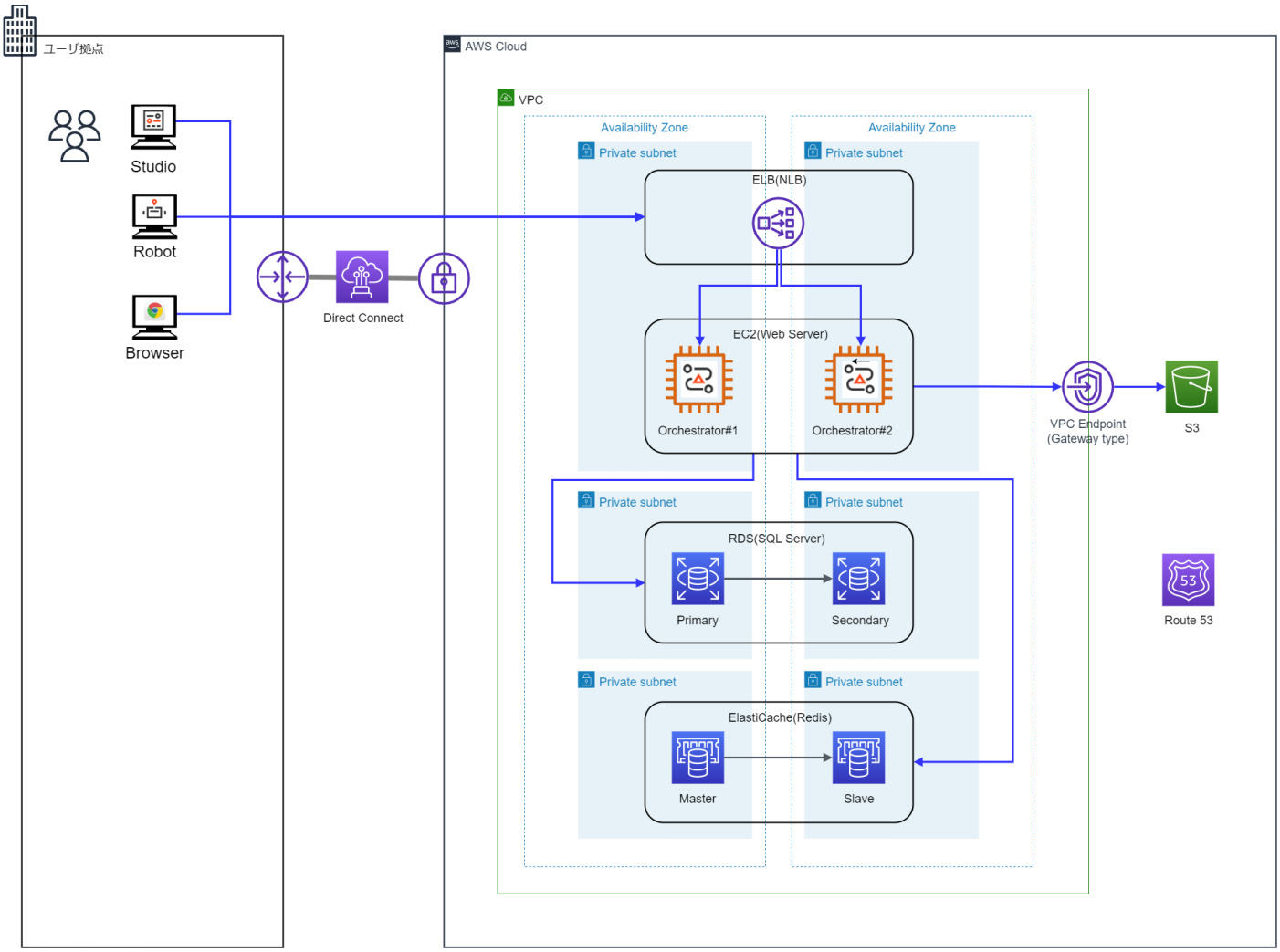
ユーザー拠点から AWS への接続は Direct Connect による閉域網接続を前提としています。



AWS 冗長構成例

AWS 環境での冗長構成例です。

ユーザー拠点から AWS への接続は Direct Connect による閉域網接続を前提としています。



AWS 環境での各コンポーネント・役割

AWS 環境においては下記のサービスをそれぞれ利用することにより Orchestrator を構成することが可能です。

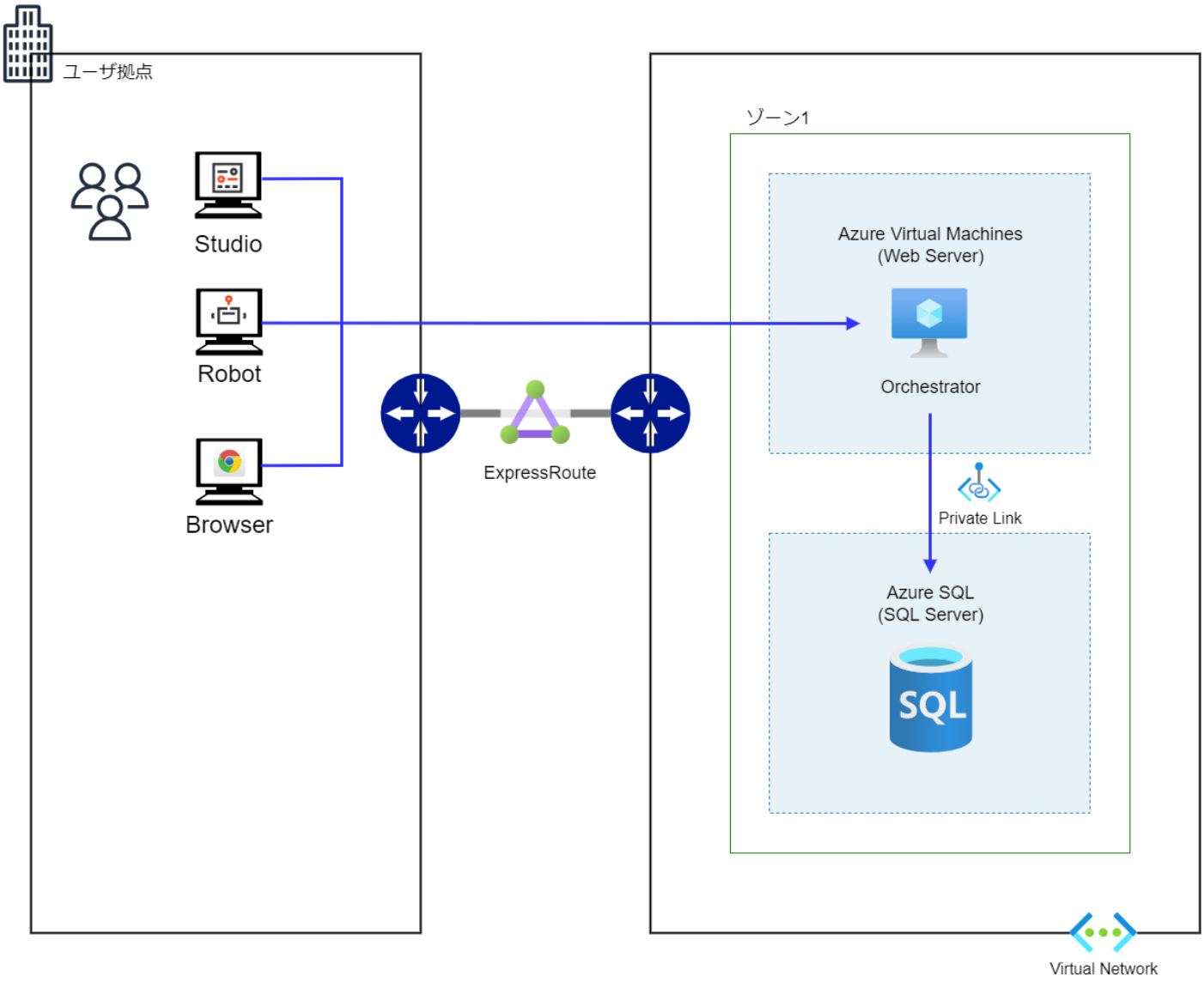
コンポーネント	利用可能な **AWS** サービス
Orchestrator	● **Amazon EC2** (Windows Server) に Orchestrator をインストールして利用可能です。
SQL Server	● **Amazon RDS for SQL Server** が利用可能です。 ● Multi-AZ 構成による高可用構成も構築可能です。
Storage ディレクトリ (NuGet パッケージ)	● **Amazon S3** と **Amazon FSx** に対応しています。
HAA/Redis	● **Amazon ElastiCache for Redis** が利用可能です。 ● Multi-AZ 構成による高可用構成も構築可能です。
ロードバランサー	● **Amazon ELB** を利用可能ですが、**NLB** (Network Load Balancer)を推奨します。 ● Windows 認証によって Orchestrator 管理画面にログインするには接続元ポートを保持する必要があるため、ALB (Application Load Balancer) ではなく NLB を使用します。
Elasticsearch	● バージョン 7.10.2 までの Elasticsearch Service は利用可能ですが、その後フォークされた OpenSearch は現時点では Orchestrator は正式にサポートしていません。 ● Elastic Cloud を AWS 環境にデプロイすることも可能です。Orchestrator のソフトウェア要件を満たす Elasticsearch/Kibana のバージョンを選択します。Elastic Cloud サービスの詳細は Elastic 社にお問い合わせください。 ● EC2 (Windows Server または Linux) に Elasticsearch をインストールして利用することも検討します。
DNS サービス	● **Amazon Route 53** を利用可能です。 ● Orchestrator の FQDN を名前解決する場合などに使用します。

Azure シングル構成例

Azure 環境でのシングル構成例です。

Web サーバーは Azure Virtual Machines (IaaS) を利用しています。

ユーザー拠点から Azure への接続は ExpressRoute による閉域網接続を前提としています。

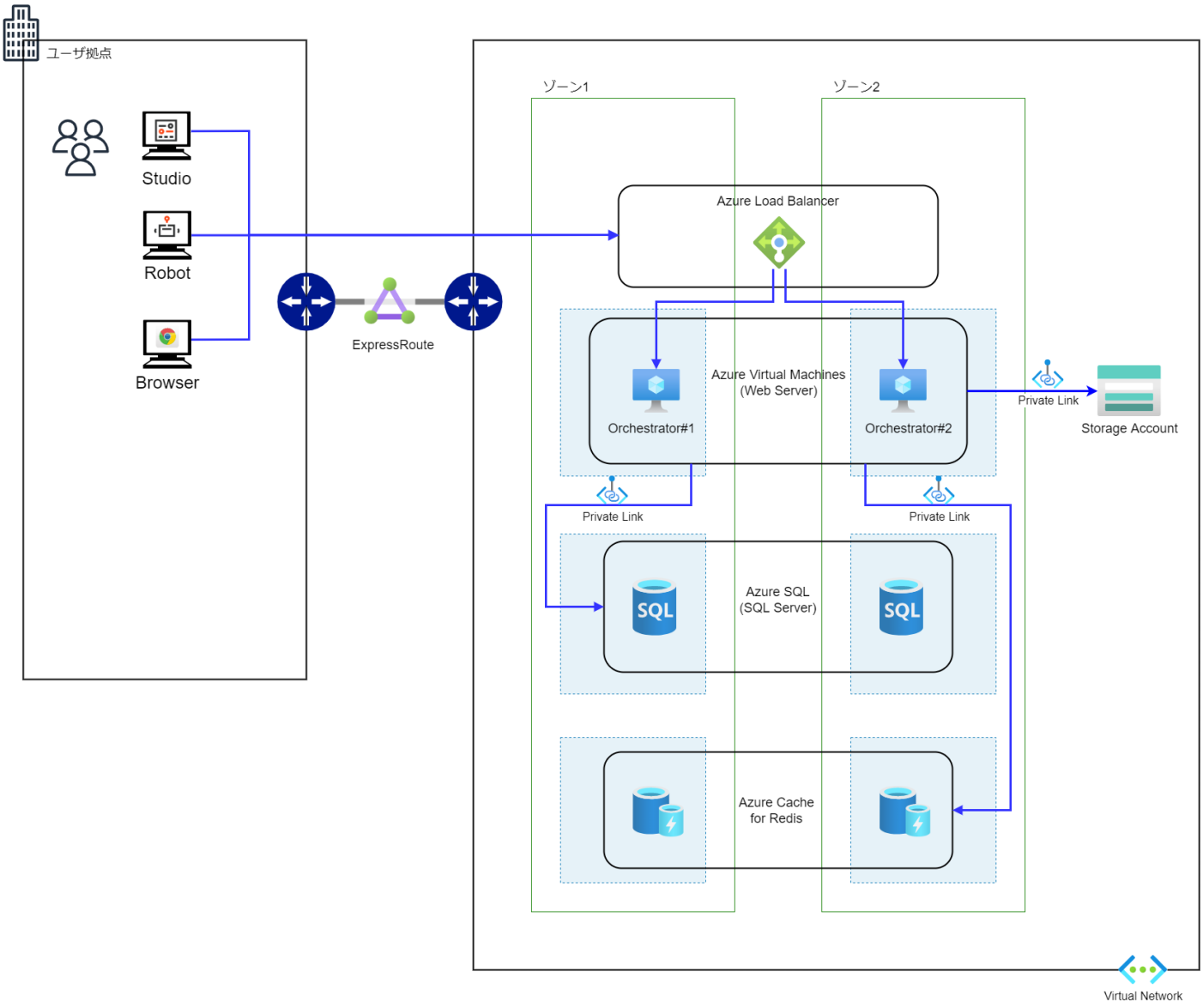


Azure 冗長構成例

Azure 環境での冗長構成例です。

Web サーバーは Azure Virtual Machines (IaaS) を利用しています。

ユーザー拠点から Azure への接続は ExpressRoute による閉域網接続を前提としています。



Azure 環境での各コンポーネント・役割

Azure 環境においては下記のサービスをそれぞれ利用することにより Orchestrator を構成することが可能です。

コンポーネント	利用可能な **Azure** サービス
Orchestrator	● **Azure Virtual Machines** または **Azure App Service** の利用が可能です。 App Service の場合は インストールスクリプトを用いて Orchestrator を構築します。
SQL Server	● **Azure SQL** データベース が利用可能です。
Storage ディレクトリ (NuGet パッケージ)	● **Azure** ストレージアカウント に対応しています。 ● Blob または Files を利用可能です。
HAA/Redis	● **Azure Cache for Redis** が利用可能です。
ロードバランサー	● **Azure Load Balancer** を利用可能です。
Elasticsearch	● Elastic Cloud を Azure 環境にデプロイすることも可能です。Orchestrator のソフトウェア要件を満たす Elasticsearch/Kibana のバージョンを選択します。Elastic Cloud サービスの詳細は Elastic 社にお問い合わせください。 ● Azure Virtual Machines (Windows Server または Linux) に Elasticsearch をインストールして利用可能です。
DNS サービス	● Azure DNS ゾーンまたは プライベート **DNS** ゾーン を利用可能です。 ● Orchestrator の FQDN を名前解決する場合などに使用します。

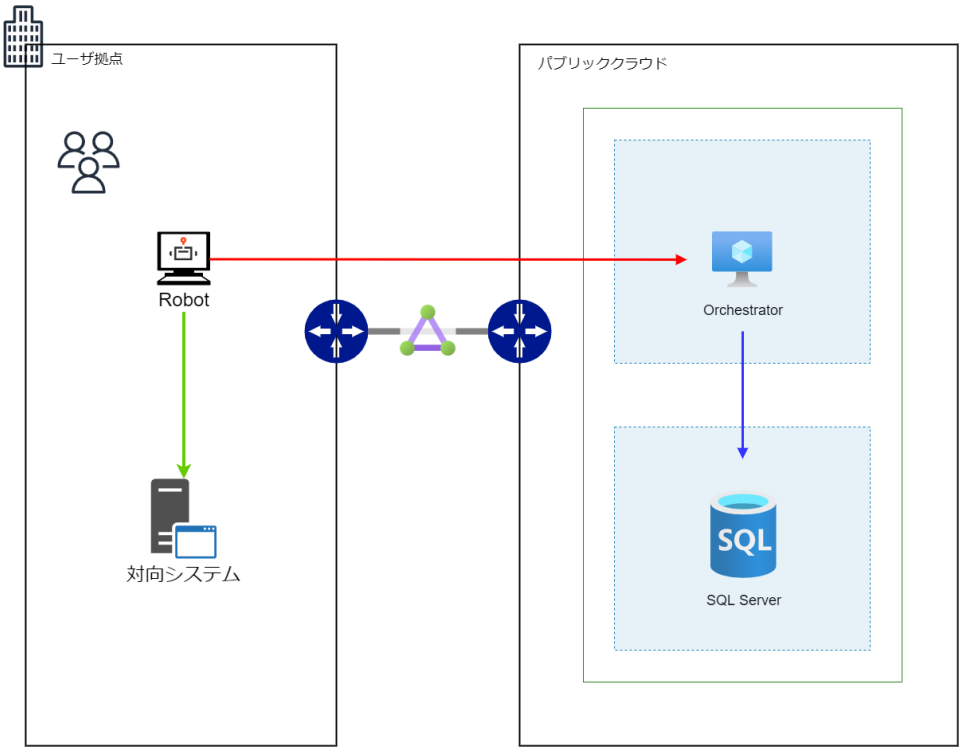
2.7.3. ネットワーク

クラウド環境に Orchestrator を構築する際のネットワークについて留意点は下記の通りです。

- ロボット端末をユーザー拠点またはクラウド環境いずれに置くか？
- ユーザー拠点とクラウド環境をどの回線 (専用線・インターネット VPN・インターネット) で接続するか？

ユーザー拠点にロボット端末を置く場合

ユーザー拠点にロボット端末と自動化対象の対向システムがあり、クラウド上に Orchestrator 環境がある際のネットワーク接続形態ごとの注意点を記載します。接続の方向はロボット端末→Orchestrator への一方向となります。



専用線接続

AWS の Direct Connect または Azure の ExpressRoute サービスを使い、専用線(閉域網)接続を利用する場合、ネットワーク帯域の上限に制限が設けられます。大量のログを出力するようなワークフローを実行する場合、ネットワークの帯域に注意が必要です。

インターネット VPN 接続

Azure の VPN Gateway、AWS の AWS VPN を使いインターネット上に VPN を構築する場合、専用線接続に比べ回線の品質が下がります。VPN の状態を細かく確認することが推奨されます。

インターネット接続

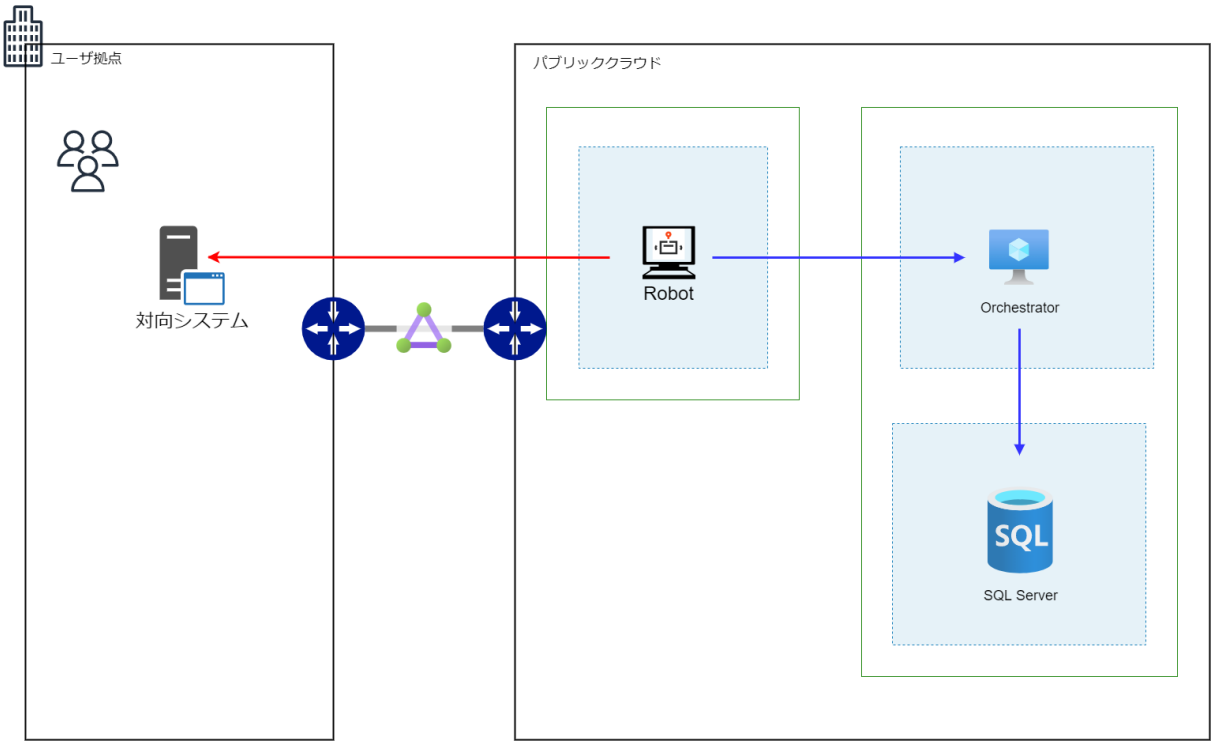
閉域網や VPN を使用せずに、インターネットを介した接続を行います。Orchestrator がインターネットに対し開放されますので接続元の IP アドレスを制限する、ネットワーク経由のアタックを防御するなど、慎重なセキュリティ設定が求められます。

また社内 LAN からインターネットへの接続が社内プロキシサーバーを経由する場合には、ロボット端末側に プロキシ設定 が必要となります。

クラウド環境にロボット端末を置く場合

クラウド環境にロボット端末と Orchestrator 環境を置き、ユーザー拠点に自動化対象の対向システムがある場合、ロボット端末からユーザー拠点への通信が必要となるため専用線またはサイト間 VPN 接続サービスを利用します。

ロボット端末としてクラウドの VM サービスまたは VDI サービス (Amazon WorkSpaces や Azure Virtual Desktop など) を利用します。ワークフロー内で Microsoft Office などライセンスが必須となるアプリケーションを利用する場合はクラウド環境で利用可能かサービスプロバイダーに確認します。



3. システム構築・設定

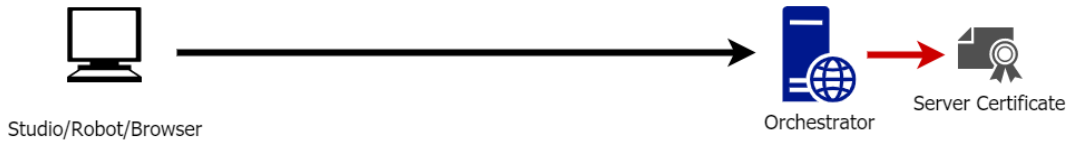
本章では Orchestrator および関連するサーバーの構築および設定手順について説明します。主にオンプレミス環境での手順を説明しますが、パブリッククラウド環境に特化した手順については パブリッククラウド環境での Orchestrator 構築 もご参照ください。

3.1. サーバー証明書

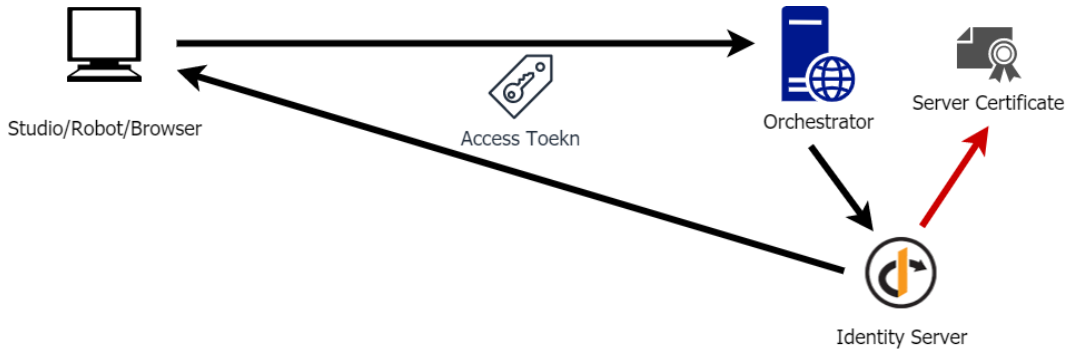
3.1.1. サーバー証明書の役割

Orchestrator が使用するサーバー証明書には 2 つの役割があります。

- 【役割 1】HTTPS 通信
  - クライアント(Studio/Robot/ブラウザー)からの HTTP 通信を SSL/TLS 暗号化(つまり HTTPS 化)するためにサーバー証明書が使用されます。 この役割は Orchestrator の従来のバージョンから最新版において必須となるネットワーク要件です。



- 【役割 2】Identity Server アクセストークン発行
  - Orchestrator v2020.4 以降では HTTPS 化に加えて、Orchestrator に内包される Identity Server と呼ばれる認証・認可を行うサービスがアクセストークンを発行するためにも使用されます。
  - アクセストークンはクライアントが Orchestrator に対して適切な権限を用いて処理要求を送信する際に使用されます。Orchestrator は自身の保護対象リソースへの処理を行う際にはアクセストークンの妥当性を検証し、正当な処理要求のみを受け入れます。



3.1.2. サーバー証明書発行の認証局とクライアント端末への配布方法

- サーバー証明書発行するための認証局(CA)の選定とクライアント(Studio/Robot/Orchestrator 管理画面にアクセスするブラウザー)端末への配布方法を検討します。

認証局	証明書配布方法
パブリック CA	OS にあらかじめ主なパブリック CA ルート証明書が配布されているため、追加作業は不要だが、ローカルドメイン (例: mydomain.local) では使用不可
ドメイン CA	CA のルート証明書がグループポリシーで配布されるため、ドメイン所属の端末であれば追加作業は不要
なし (自己署名)	CA を使用せず自己署名による証明書は、各端末に個別にインストールまたはグループポリシーで配布する必要あり

3.1.3. サーバー証明書発行の注意点

- サーバー証明書を発行する際には有効期限に注意します。原則としてエンドユーザーのセキュリティポリシーに合わせて有効期限を定めますが、期限切れとなった場合には Orchestrator が正常に機能しなくなるため期間に余裕をもって発行するか、期限切れとなる前に サーバー証明書の入れ替え を実施するようにあらかじめ計画します。
- 証明書のサブジェクト代替名 (SAN: Subject Alternative Name)は Orchestrator のアクセス URL における FQDN (https://rpa.lab.test の場合には rpa.lab.test) と一致するように作成します。

3.1.4. 冗長構成におけるサーバー証明書と DNS 設定

- 冗長構成にてロードバランサー経由でアクセスする場合でも、ロードバランサーでは SSL 終端させず IIS で終端するようサーバー証明書は各 Orchestrator ホストにてインストールします。
- サーバー証明書はそれぞれの Orchestrator ホスト名で作成せず “rpa.lab.test” のようにホスト名に依存しない FQDN を使用して共通のサーバー証明書を作成し、同一の証明書をそれぞれの Orchestrator ホストにインポートします。
- FQDN はロードバランサーの仮想 IP に名前解決されるように DNS サーバーのレコードを設定します。

3.2. SQL Server

3.2.1. 概要

Orchestrator ソフトウェア要件 に記載のある SQL Server バージョンおよびエディションを選択します。

- SQL Server エディションに起因する Orchestrator の機能はありません。
- SQL Server のインストール手順は Microsoft 公式ガイド をご参照ください。
- SQL Server エディションによる機能の違いは Microsoft 公式ガイド をご参照ください。

3.2.2. 冗長構成における SQL Server 構築

Always On 可用性グループの構成

本番環境での大規模運用では SQL Server 冗長構成を推奨します。Always On 可用性グループによる冗長構成を利用する時、ドメイン環境下の WSFC (Windows Server フェールオーバークラスタ) が前提となり、SQL

Server を稼働させる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。必要な権限等は SQL Server を参照してください。

作業の流れは WSFC のインストール、クォーラム設定、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ー作成、Orchestrator の DB 接続パラメータの変更となります。

詳細な手順は チュートリアル:可用性グループを手動で構成する (Azure VM 上の SQL Server) をご参照ください。

可用性モード

同期コミットモードと非同期コミットモードがありますが、同一データセンターで運用する際にはデータ欠損が発生しない同期コミットモードを使用します。DR サイトに対してレプリカを作成する場合には非同期コミットモードを使用します。

読み取り可能セカンダリ

AlwaysOn 可能性グループにはセカンダリに読み取り専用でアクセスする機能があります。この機能によって本番のプライマリデータベースに影響を与えずに、バックアップを取得したり、統計情報の取得やパフォーマンスの悪い参照系クエリの分析などトラブルシューティングを行ったりすることができます。ただしこの機能は SQL Server Enterprise でのみ利用可能です。詳細については Microsoft 公式ガイドをご参照ください。

- 読み取り可能セカンダリ
- 基本的な可用性グループの制限事項

3.3. Orchestrator

3.3.1. 概要

Orchestrator インストール手順は各種ガイドをご参照ください。

冗長構成の場合は、先に HAA (Redis) のインストールを行います。

Elasticsearch/Kibana を利用する場合は Elasticsearch/Kibana を先にインストールを行います。

- Orchestrator インストールの前提条件
- Orchestrator インストールについて
- Orchestrator 導入ステップバイステップガイド

3.3.2. UiPath.Orchestrator.dll.config 設定

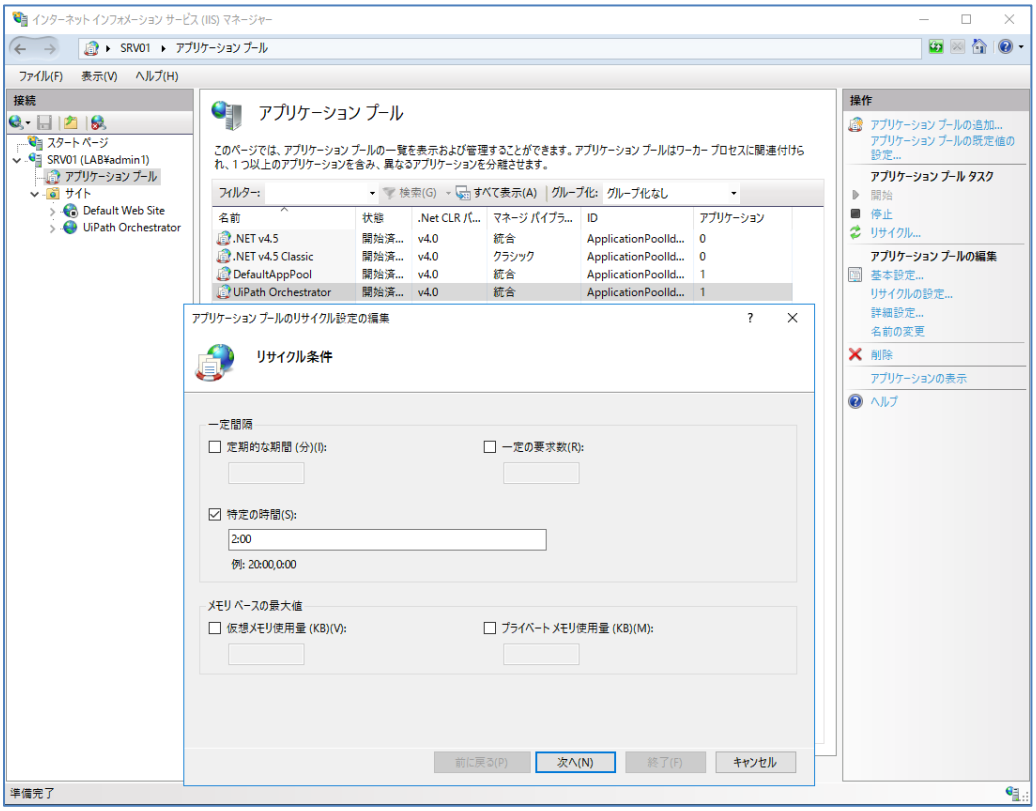
設計時に検討した項目を設定します。

冗長構成時は各 Orchestrator ノードで同一の設定になっていることを確認します。

3.3.3. アプリケーションプールリサイクル設定

既定値では 1740 分 (29 時間) ごとに Orchestrator 関連のアプリケーションプールが新しい Windows プロセスにて定期的にリサイクル(再起動)されます。日中にリサイクルが実行されることによって、一時的に

Orchestrator への接続不可となる可能性があります。この問題を回避するために、夜間の時刻指定 (例: 午前 2 時) に変更することを推奨します。



3.4. HAA (Redis)

対応しているバージョンは Orchestrator ソフトウェア要件 を参照してください。

UiPath のサポートが必要な場合には、HAA (High Availability AddOn) の導入をご検討ください。HAA のシステム要件とインストール手順は下記をご参照ください。

- HAA ハードウェア・ソフトウェア要件
- HAA インストール

3.5. Elasticsearch/Kibana

3.5.1. 対応バージョン

Orchestrator がサポートする Elasticsearch/Kibana バージョンは Orchestrator ソフトウェア要件 を参照してください。

Elasticsearch/Kibana インストールの手順は Elastic 社の公式ガイドをご参照ください。

- Installing Elasticsearch
- Install Kibana

Elasticsearch/Kibana 本体のサポートが必要な場合には、Elastic 社のサブスクリプション の導入をご検討ください。

3.5.2. ログ書き込み設定

それぞれのケースにおける UiPath.Orchestrator.dll.config 設定値は次の通りです。

SQL Server のみ (既定値)	<logger name="Robot.*" final="true" writeTo="database" />
Elasticsearch のみ	<logger name="Robot.*" final="true" writeTo="robotElasticBuffer" />
SQL Server と Elasticsearch の両方	<logger name="Robot.*" final="true" writeTo="database,robotElasticBuffer" />

エラーレベルによって格納先を変更する設定例を下記に示します。この設定により Critical/Error は SQL Server に、Critical/Error/Warn/Info は Elasticsearch に格納されるようになります。

```
<logger name="Robot.*" minlevel="Error" final="false" writeTo="database" />
<logger name="Robot.*" minlevel="Info" final="true" writeTo="robotElasticBuffer" />
```

3.5.3. ログ読み込み設定

SQL Server と Elasticsearch の両方にログ書き込みを設定している場合、Orchestrator 管理画面上のログ表示は Elasticsearch から読み込みが行われます。このケースにおいて SQL Server から読み込みを行うには UiPath.Orchestrator.dll.config にて次の設定を行います。(参照: ログの構成)

<appSettings>セクションに追加

```
<add key="Logs.RobotLogs.ReadTarget" value="database" />
```

Elasticsearch からログを読みこむ場合、一度のクエリで読み込まれるログ件数は既定値では 10000 件に制限されています。10000 件より多くのログを読み込む必要がある場合には、UiPath.Orchestrator.dll.config の次のパラメータで上限値を調整します。

```
<add key="Logs.Elasticsearch.MaxResultWindow" value="10000" />
```

3.5.4. Kibana ダッシュボード

Kibana ダッシュボードのテンプレート を使用することにより、Orchestrator 運用で一般的に使用されるダッシュボードをインポートにて構築することができます。

ダッシュボードの作成・変更前後においては ダッシュボードオブジェクトのエクスポートを行い 、復元できるように備えることを推奨します。

一般的な Kibana の操作方法やダッシュボードの作成方法については、Elasticsearch/Kibana のトレーニングを受講していただくことを推奨しております。トレーニングにつきましては UiPath 社までご相談ください。

3.6. Storage ディレクトリ

- オンプレミス環境では NuGet パッケージなどのアプリケーションデータを保存するストレージとしてローカルディレクトリやファイルサーバーを準備します。

- ファイルサーバーなど既定値から Storage ディレクトリを変更している場合にはアクセス権限に注意します。Orchestrator をアプリケーションプール ID で動作させている場合にはビルトインアカウント **IIS AppPool\UiPath Orchestrator** を変更先ディレクトリに割り当て、変更権限を付与します。ファイルサーバーの場合にはドメインアカウントを準備し、Orchestrator インストール時のアプリケーションプール設定にてカスタムアカウントで Orchestrator を動作させることを検討します。
- 冗長構成の場合には DFS レプリケーションを使用して各 Orchestrator ノードの Storage ディレクトリを同期します。旧バージョンでレポジトリタイプが Legacy の場合には パッケージのメタデータが格納される *.bin ファイルを同期除外する必要がありましたが、v2020.10 以降はレポジトリタイプが Composite となりメタデータはデータベース内に格納されるため同期除外は不要です。設定手順は DFS レプリケーション設定手順 を参照ください。

- AWS 環境では Amazon S3、Azure 環境では Azure Blob Storage を Storage ディレクトリとして利用することも可能です。また AWS 環境での Amazon FSx、Azure 環境での Azure Files も利用可能ですがこれらは前述のファイルサーバーと同様にアプリケーションプール ID のディレクトリへの変更権限に注意します。
- Storage ディレクトリの設定は デプロイ を参照してください。

3.7. パブリッククラウド環境での Orchestrator 構築

AWS/Azure などのパブリッククラウド環境で Orchestrator を展開することによって、環境構築および運用管理コストを削減することが可能になります。またパブリッククラウドならではの拡張性や耐久性を享受できます。パブリッククラウドにおいてもオンプレミスと同様に手動で Orchestrator を構築することも可能ですが、ここでは AWS および Azure 環境それぞれにおいてリソース展開スクリプトを活用する手順のリンクを紹介します。

3.7.1. AWS 環境での Orchestrator 構築

AWS 環境では CloudFormation を活用して Orchestrator 関連のリソースを一括でプロビジョニングすることが可能です。

- AWS クラウドでの UiPath Orchestrator
  - AWS クラウドでの UiPath Orchestrator
  - Deployment Guide (英語)

3.7.2. Azure 環境での Orchestrator 構築

Azure 環境ではマーケットプレイスに公開されているソリューションによって Orchestrator 関連のリソースをシングル構成または冗長構成にて一括でプロビジョニングすることが可能です。

- Azure Marketplace - UiPath Orchestrator
- 構築手順

4. システム運用

4.1. 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。具体的にはサーバー障害を未然に防止し、万が一トラブルが発生した場合にも原因特定にかかる時間を短縮し、元の状態に素早く復旧させることが可能になります。

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
- 障害対応: 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

4.2. 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。

- 各サーバー
- 各ミドルウェア
  - IIS
  - SQL Server
  - HAA/Redis
- ネットワーク
- ロードバランサー

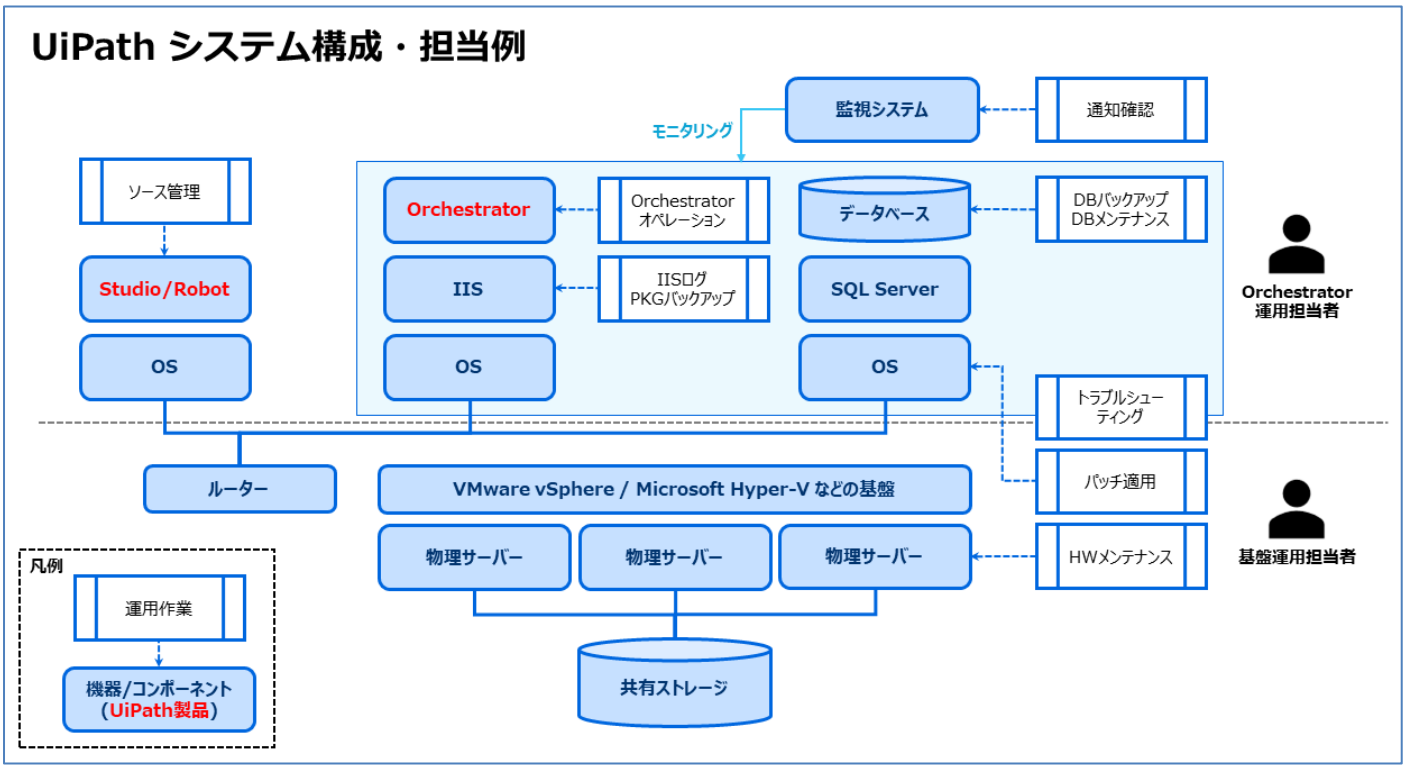
また、ログ分析を行う場合は以下も対象にする必要があります。

- Elasticsearch
- Kibana

4.3. 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、必要に応じて業務を行っている (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します。

運用担当者と作業内容の一例を図示したものを下記に示します。



4.3.1. 基盤運用

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物理的なハードウェアのメンテナンス、物理サーバー、仮想化基盤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。

4.3.2. Orchestrator 運用

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。

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は基盤管理運用担当者と協議して役割分担を行います。

4.4. 運用スケジュール

4.4.1. 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し、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。このためメンテナンス時間帯にはジョブ実行されないように業務担当チームとあらかじめ時間調整を行います。

Unattended ジョブは必要に応じて時間トリガー(スケジュール)設定の [ジョブの実行を終了するスケジュールを設定] を使用して、メンテナンス開始前に実行中のジョブを強制終了します。

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。

- IIS ログの退避
- データベースのバックアップ
- 古い実行ログの削除
- データベースメンテナンス
- 不要なパッケージの削除
- サーバー証明書の入れ替え

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
IIS ログの退避	毎週	不要
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
サーバー証明書の入れ替え	毎年	必要

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・バックアップ を参照してください。

4.4.2. 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が行いやすくなります。

また OS のパッチ適用およびミドルウェアのバージョンアップなど、Orchestrator 停止が必要となる作業を行う場合には、ジョブ停止の時間帯をあらかじめ業務側と調整しておきます。

4.5. メンテナンス・バックアップ

4.5.1. 概要

障害に対する予防や障害からの復旧を迅速に行うためには日ごろから適切なメンテナンス・バックアップが必要です。 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・バックアップ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・バックアップ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 設定ファイルのバックアップ
- NuGet パッケージのバックアップ
- 古い NuGet パッケージファイルの削除
- リストア手順の確立

4.5.2. データベースのバックアップとリストア

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、SQL Server データベースにおいて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 SQL Server データベースのバックアップと復元 を参照してください。

バックアップスケジュールの例として、週次で完全バックアップを取り、日次で差分バックアップを取る場合には以下のような手順になります。

- [日曜日] 完全バックアップ
- [日曜日] トランザクションログの切り捨て
- [日曜日] 旧バックアップデータ削除
- [月曜日～土曜日] 差分バックアップ

保持するバックアップ世代数、保存先と将来のデータ増加を見越したディスク容量の確保、古い実行ログの削除するタイミングなどを検討します。

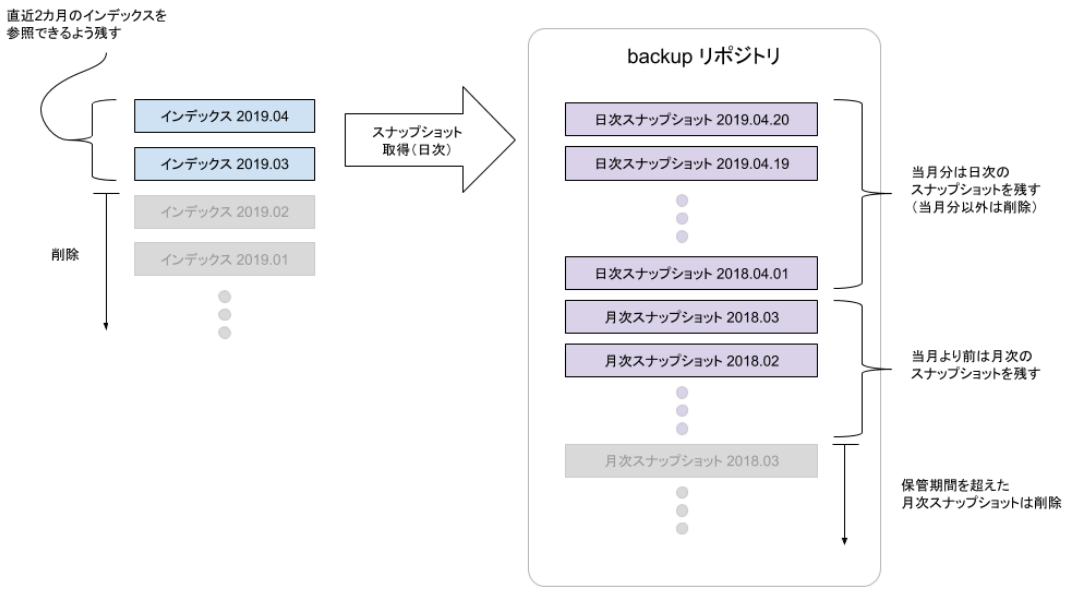
SQL Server のデータベースのバックアップからのリストア手順は復旧モデルによって異なります。 バックアップと復旧の手順は SQL Server データベースのバックアップと復元 を参照してください。

注意: データベースのバックアップはデータファイル (*.mdf) やトランザクションログファイル (*.ldf) のコピーでは正常にリストアできない可能性があります。必ず上記サイトの手順に従ってください。

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

本ガイドが想定している 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。



具体的な Elasticsearch スナップショット取得とリストア手順は Elasticsearch スナップショット取得とリストア手順 をご参照ください。

4.5.3. データベースのアーカイブと古いレコードの削除

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、過去の不要なログの定期的な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 の定期的な削除が必要となります。

注意: 上記以外にもデータが蓄積されるテーブルがありますが、他テーブルとの整合性等の理由により物理削除することはサポート対象外となります。最新情報については メンテナンスに関する考慮事項 をご参照ください。

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. 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。

```
insert into [UiPath].[dbo].[ArchiveLogs] ([OrganizationUnitId], [TenantId], [TimeStamp], [Level], [WindowsIdentity], [ProcessName], [JobKey], [RobotName], [Message], [RawMessage], [MachineId])
select [OrganizationUnitId], [TenantId], [TimeStamp], [Level], [WindowsIdentity], [ProcessName], [JobKey], [RobotName], [Message], [RawMessage], [MachineId]
from [UiPath].[dbo].[Logs]
where [Logs].[Id] BETWEEN 101 and 200 /* 条件に応じて変更します */
```

4. 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
```

5. 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 テーブルのレコード削除

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 = Trace, 2 = Info,  3 = Warn, 4 = Error, 5 = Fatal  */
where level = 2
-- and TenantId = 1 -- default tenant
and  DateDiff (day, TimeStamp, GetDate () )  > 45
```

※ 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る容量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」を実際の日数に変更します。

DELETE 文でレコード削除する際の注意点

1. DELETE 文によって Logs テーブルのレコードを大量に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 SQL Server データベースメンテナンス を参照してください。
2. データベースの復旧モデルが単純以外の場合、DELETE 文実行によってトランザクションログが大量に作成され、ディスク空き容量を逼迫する場合があります。ディスク空き容量に余裕がない場合には次の手順をセットとして複数回に分けて繰り返し実行します。
    a. 次のクエリ文により上限(下記例では 10000 件)を指定して DELETE 文を実行します。

    ```
    DELETE TOP(10000) FROM [UiPath].[dbo].[Logs] where <条件>
    ```

    b. トランザクションログのバックアップを生成し、トランザクションログの切り捨てを行う。
3. DELETE 文でのレコード削除が困難な場合、次のクエリ文によってトランザクションログを生成せずに瞬時にレコード削除を行うことができます。

    ```
    TRUNCATE TABLE [UiPath].[dbo].[Logs]
    ```

    ただし保存日数を指定することはできず全レコードが削除されてしまうため、データベースのバックアップを事前に取得することを推奨します。

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
```

※ 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

Elasticsearch のデータサイズ肥大化を防止するため、古いデータを自動または手動にてアーカイブまたは削除します。これにはいくつかの方法があり、要件と Elasticsearch バージョンなどを考慮して方法を検討します。

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-2018.12 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'
     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-2018.12 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:
   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*
```

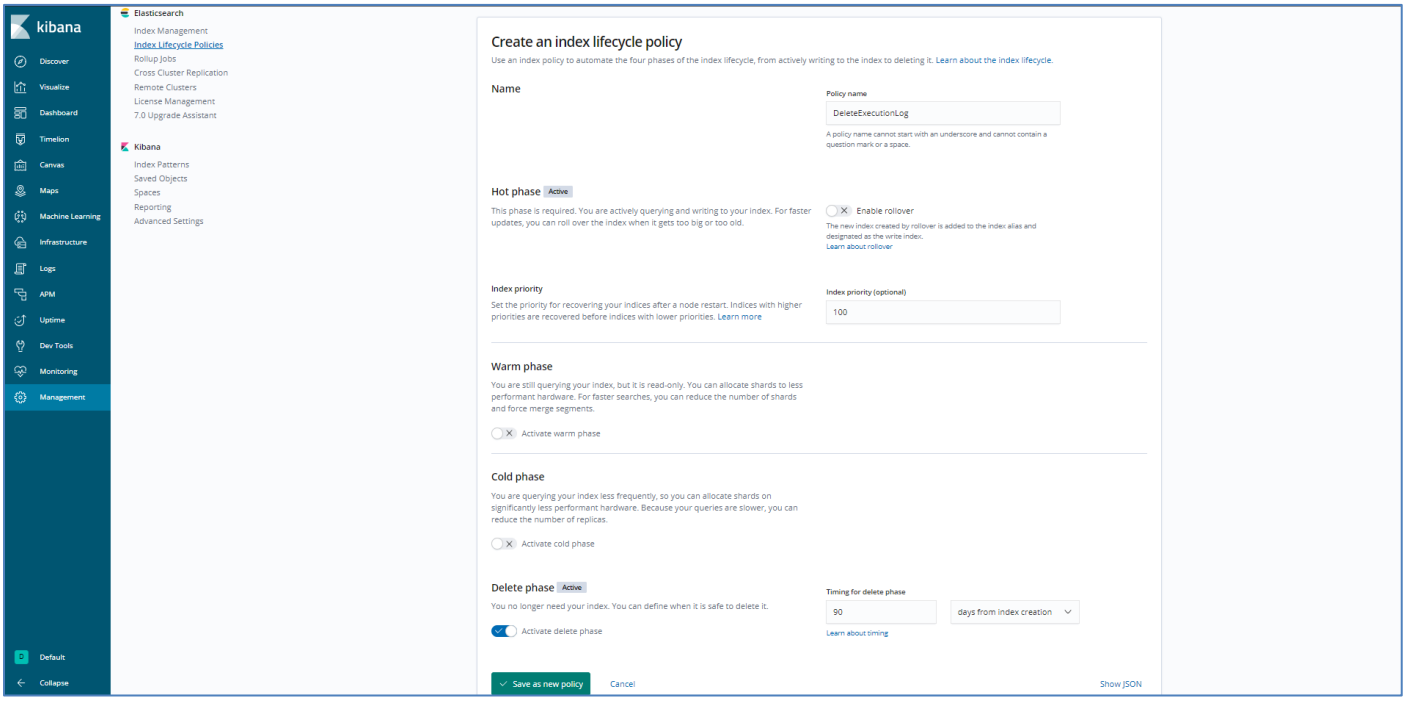
インデックスを削除したら再度一覧を表示して、削除したインデックスが表示されなくなっていることを確認してください。

```
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。

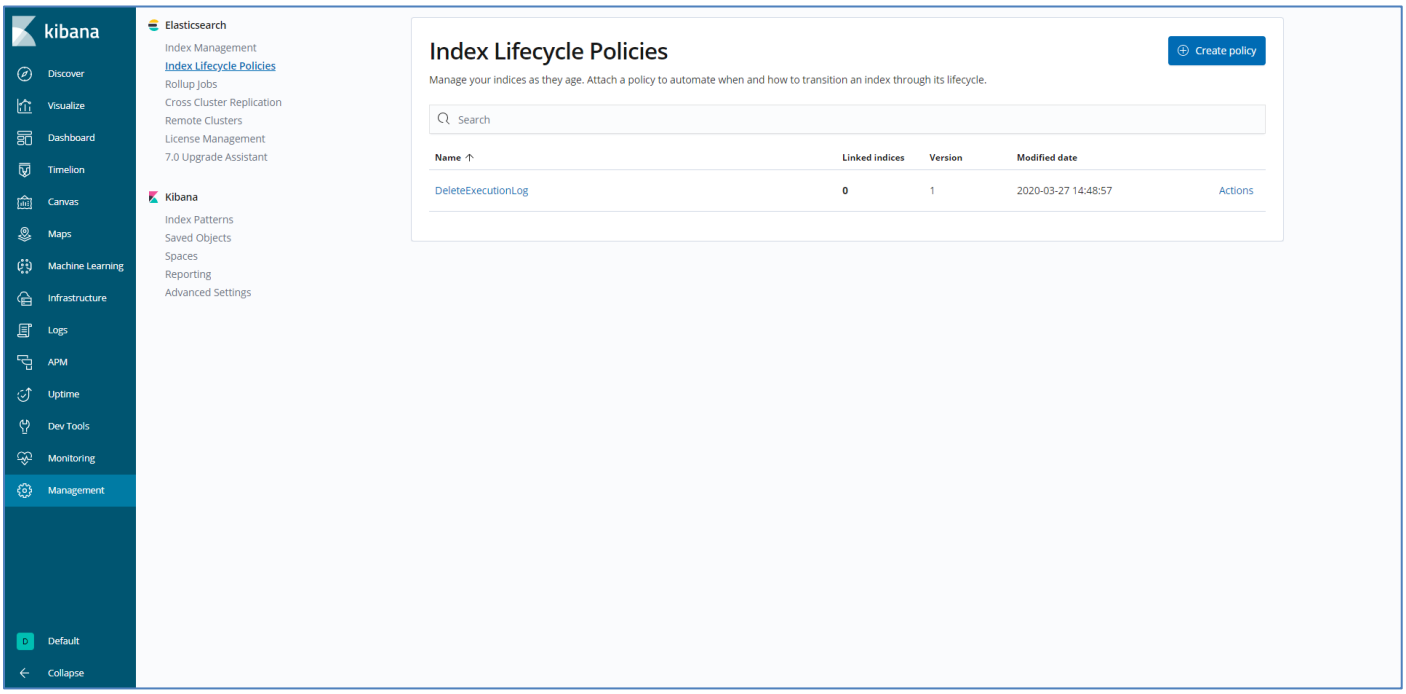
Index Lifecycle Management (ILM)の場合

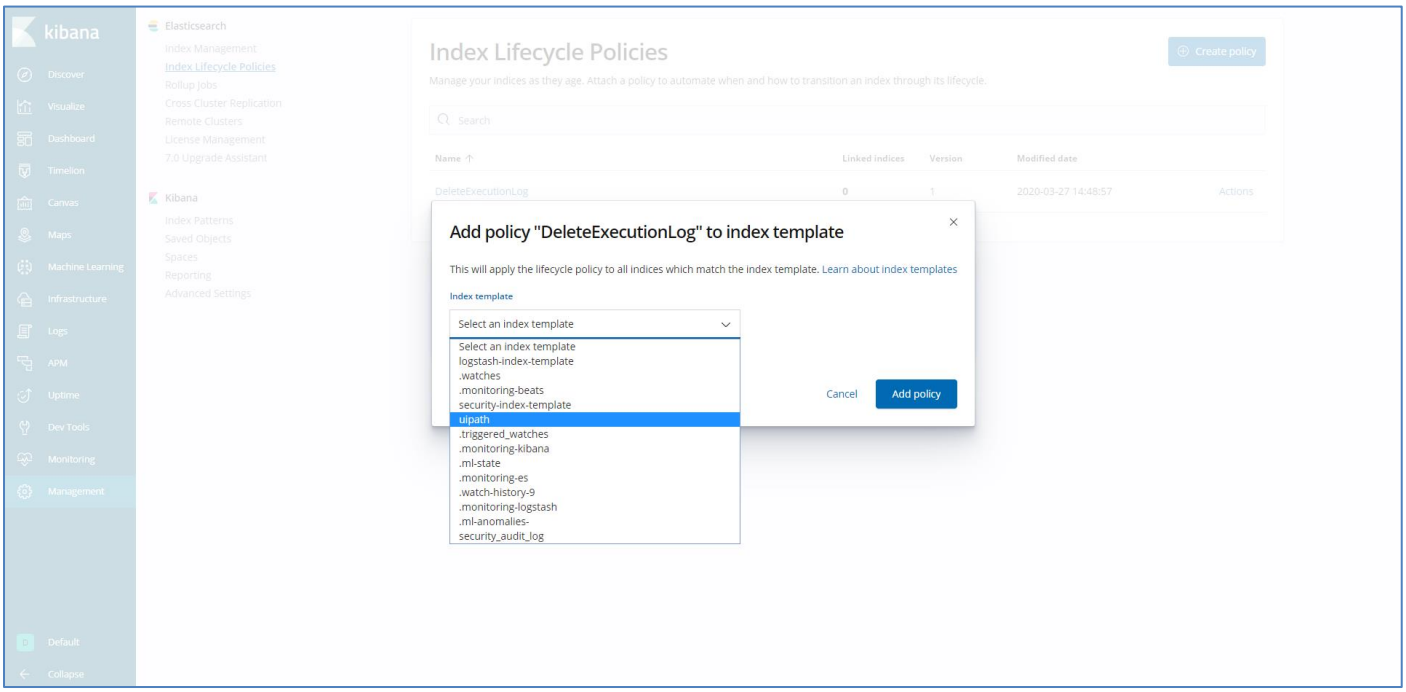
Elasticsearch v6.6 から導入された Index Lifecycle Management (ILM) 機能を使用し、インデックスを自動的に定期削除することが可能です。例えば、インデックスが作成されてから 90 日で自動削除したい場合は以下の通り、Kibana コンソール「Management」>「Index Lifecycle Policies」>「Create policy」画面で「Delete phase」を有効化し、「90 days from index creation」と設定します。



この時点では、作成したポリシーが適用されないことに注意してください。

作成した ILM ポリシーを適用するには、作成した ILM ポリシーの右端にある「Action」ボタン>「Add policy to index template」を押下し、対象となるインデックスに設定されたインデックステンプレートを選択します。以下の例では、インデックステンプレート名「uipath」、適用インデックスパターン 「default-*」について ILM ポリシーを適用しており、「default-2022.12」のようなインデックスに対し自動削除が行われるようになります。インデックステンプレートの設定例は UiPath ナレッジベース の「Elasticsearch Index テンプレート」に記載されています。





設定された ILM ポリシーは、既存のインデックスについては適用されず、次回新規で作成されるインデックスから適用されることに注意してください。

4.5.4. SQL Server データベースメンテナンス

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 コア以上のサーバーで SQL Server を動作させる場合、自動的に ロックのパーティション分割 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。

インデックス断片化率を確認するには次のクエリを実行します。

```
USE UiPath
GO

SELECT 'ALTER INDEX ' + '[' + C.name + ']' + ' ON [' + D.name + '].[' + B.name + '] REBUILD' AS 'rebuild command'
    ,D.name AS schemaname
    ,B.name AS table_name
    ,C.name AS index_name
    ,A.avg_fragmentation_in_percent
    ,A.page_count

FROM sys.dm_db_index_physical_stats (DB_ID(),null,null,null,null) AS A

LEFT OUTER JOIN  sys.objects AS B
ON  A.object_id = B.object_id

LEFT OUTER JOIN  sys.indexes AS C
ON  A.object_id = C.object_id  AND A.index_id = C.index_id

LEFT OUTER JOIN  sys.schemas AS D
ON  B.schema_id = D.schema_id

WHERE B.type = 'U'
AND   C.index_id > 0

ORDER BY table_name
GO
```

参考:

- sys.dm_db_index_physical_stats (Transact-SQL)
- インデックスの断片化率を調べる方法

Logs テーブルなどレコード件数が多いテーブルのインデックス断片化 (30%以上) が発生している場合は、インデックス再構築または再構成のメンテナンス作業を行うことを検討してください。

SQL Server Management Studio を使用したメンテナンスプラン作成手順については SQL Server メンテナンスプラン作成手順 をご参照ください。

4.5.5. Orchestrator 設定ファイルのバックアップ

下記の Orchestrator 設定ファイルに変更を行う際には前後で必ずバックアップを取得します。

- C:\Program Files (x86)\UiPath\Orchestrator\Web.config
- C:\Program Files (x86)\UiPath\Orchestrator\UiPath.Orchestrator.dll.config
- C:\Program Files (x86)\UiPath\Orchestrator\Identity\appsettings.Production.json
- C:\Program Files (x86)\UiPath\Orchestrator\Webhooks\appsettings.Production.json
- 【v2022.4 以降】C:\Program Files (x86)\UiPath\Orchestrator\ResourceCatalog\appsettings.Production.json

万が一の事態に備えて Orchestrator 設定ファイル、NuGet パッケージとデータベースのバックアップを取得していれば、一から Orchestrator 環境を再構築しアプリケーションデータを復旧することが可能です。

4.5.6. Storage ディレクトリのバックアップとメンテナンス

Storage ディレクトリのバックアップ

Storage ディレクトリには Orchestrator のテナントごとにプロセスパッケージ・ライブラリー・ストレージバケット・実行メディア(記録機能)のアプリケーションデータが格納されています。このディレクトリ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。 既定値でのパスは下記の通りとなります。

- 相対パス: .\Storage
- 絶対パス: C:\Program Files (x86)\UiPath\Orchestrator\Storage

Storage ディレクトリを丸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。

古い NuGet 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なバージョンを削除] を押下することで削除できます。

注意: 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。

NuGet パッケージディレクトリから物理的にパッケージファイルを削除することは推奨しておりません。パッケージのメタデータ(バージョンや説明など)は UiPath データベースに格納されているため、物理削除により不整合が生じるためです。

その他の Storage ディレクトリ配下のファイル削除

Storage ディレクトリ には NuGet パッケージ以外にもストレージバケットや実行メディア(記録機能)の機能を利用している場合にはそれらのアプリケーションデータが格納されます。NuGet パッケージ同様にこれらをディレクトリから直接物理削除することは推奨しておりません。

これらの古いデータを削除するには Swagger を利用して関連する API (ストレージバケット→Buckets、実行メディア→ExecutionMedia) によって削除を行います。

4.5.7. IIS ログの退避

IIS ログは既定値では C ドライブ配下 (C:\inetpub\logs\LogFiles\W3SVC2) に日ごとに生成されます。Robot マシンの台数が多い場合には 1 ファイルあたり数百 MB 以上になるケースもあるため、長期保存することによって C ドライブの空き容量が枯渇する恐れがあります。そのため定期的にファイルサーバーなど他のディスクに古い IIS ログを退避させることを推奨します。

バッチファイルとタスクスケジューラによる IIS ログ自動退避の実装例は Orchestrator 管理者のための ミドルウェア運用設定ガイド I の **3.2.1 IIS** の内部ログのパージ をご参照ください。

4.6. 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してください。

4.6.1. 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
  - SQL Server
  - Elasticsearch/Kibana
  - HAA (Redis)
- サーバーリソース
  - CPU 使用率
  - メモリ使用率
  - ディスク空き容量
- ネットワーク
  - ロードバランサー
  - NLB と Orchestrator 間のサブネット
  - Orchestrator と SQL Server 間のサブネット
  - Orchestrator と Elasticsearch 間のサブネット

4.6.2. 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 API を実行することで確認できます。

```
GET <Orchestrator URL>/api/health
```

レスポンスのステータスコードが 200 であれば、Orchestrator、DB 接続およびネットワークは正常です。

Orchestrator v2020.10 以前ではヘルスチェック用の API として **/api/Status** が利用されていましたが、v2021.4 以降は上記の **/api/health** が利用できます。

AWS での設定例は サービスイベント監視 を参照してください。

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 URL>/_cluster/health
```

4.6.3. イベントログ監視

Orchestrator が出力するイベントログ (アプリケーション) には様々なメッセージが出力されますが、原則としてパフォーマンス低下や動作不良の際の原因調査の手段いわばデバッグログとしての位置づけとなります。リアルタイムの障害検知には前述のヘルスチェック API を利用することが推奨されます。

イベントログ監視を障害検知の手段として利用することは推奨されませんが、他に手段がない場合には重大なエラーメッセージのみを監視します。

Orchestartor に関連するイベントログソースは下記の 4 種類あります。

1. "Orchestrator" イベントログソースは、主に Orchestrator の内部処理の情報や例外処理発生時のエラーなどが記録されます。
2. "Orchestrator.BusinessException" イベントログソースは、主に Orchestrator 管理画面で設定を変更した際の妥当性チェックなどにより発生するエラーが記録されます。
3. "IdentityService" イベントログソースは、Identity Server が Orchestrator ログイン認証やアクセストークン発行などの処理に関連するメッセージが記録されます。
4. "WebhookService" イベントログソースは、Webhook と呼ばれる通知メッセージサービスの処理に関連するメッセージが記録されます。

"Orchestrator" イベントログソースのエラーには Orchestrator サービスや基盤に影響を与える可能性があるものも含まれるため、メッセージ監視により障害検知を行う場合には特に下記のエラーメッセージを部分一致により検知するように監視を実装します。

エラーメッセージ	原因	対処
The transaction log for database 'UiPath' is full	DB トランザクションログが一杯	トランザクションログ拡張またはバックアップを取得してトランザクションログを切り捨て
System.Data.Entity.Core.EntityException: The underlying provider failed on Open.	SQL Server への接続不可	Orchestrator <-> SQL Server 間のネットワークおよび SQL Server サービスの起動を確認
No connection is available to service this operation	【冗長構成のみ】HAA (Redis) への接続エラー	Orchestrator <-> HAA (Redis) 間のネットワークおよび HAA (Redis) サービスの起動を確認 ● タイムアウト値を延長
System.CompenentModel.Win32Exception: 待ち操作がタイムアウトになりました。	SQL クエリの実行タイムアウト (30 秒超過)	SQL Server リソース状態確認 ● インデックス断片化確認および SQL Server メンテナンスプランの実行
System.InvalidOperationException: タイムアウトに達しました。プールから接続を取得する前にタイムアウト期間が過ぎました。	コネクションプール (既定値 100) の枯渇	● コネクションプール制限値の変更

プールされた接続がすべて使用中で、プール サイズの制限値に達した可能性があります。		
The associated certificate has expired	サーバー証明書の有効期限切れ	● サーバー証明書の入れ替え

4.6.4. ミドルウェア監視

ミドルウェアの監視対象と方法は下表の通りです。

対象	方法
IIS	● IIS サービスの死活監視 ● Orchestrator サイトの監視 ● アプリケーションプールの監視 ● 性能監視、応答性能
SQL Server	● サービスの死活監視 ● パフォーマンスカウンター ● スロークエリ監視
Elasticsearch	● サービスの死活監視 ● 性能監視
HAA (Redis)	● サービスの死活監視 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。正常な場合には “PONG” が返されます。 `redis-cli -h <HAA または Redis ホスト> -p <ポート番号> -a <パスワード> ping` redis-cli は Windows 版 Redis の ダウンロードサイト から zip ファイルをダウンロードし、ローカルディレクトリに解凍して実行します。

AWS での設定例は サービスイベント監視 を参照してください。

4.6.5. サーバー監視

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

AWS での設定例は リソース監視 を参照してください。

4.6.6. 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

4.7. パブリッククラウドでの運用設計

AWS/Azure 環境における運用例をそれぞれ説明します。

4.7.1. バックアップ設計

AWS 環境

対象	方法
Orchestrator (Web サーバー)	マスターイメージの更新時は AMI を取得します。 稼働中インスタンスのディスクは AWS Backup でスケジュールバックアップを取得します。
SQL Server	RDS for SQL Server の自動バックアップを利用します。
Storage ディレクトリ	Amazon S3 を使用している場合はそれ自身が冗長化されているため可用性の観点ではバックアップ作成は不要ですが、オペレーションミスによるパッケージファイル消去に備えてバックアップを作成するか、Studio から再度ワークフローを Publish します。
Elasticsearch	自動スナップショット機能を利用することも可能です。

AWS での設定例は バックアップ を参照してください。

Azure 環境

対象	方法
Orchestrator (Web サーバー)	IaaS (Azure Virtual Machines)の場合には、Azure Backup などを使用して VM のバックアップを実施します。 PaaS (Azure App Service)の場合は、App Service の バックアップと復元の機能を用いてスケジュールバックアップを実施します。
SQL Server	Azure SQL の自動バックアップを利用します。
Storage ディレクトリ	Azure Storage を使用している場合はそれ自身が冗長化されているため可用性の観点ではバックアップ作成は不要ですが、オペレーションミスによるパッケージファイル消去に備えてバックアップを作成するか、Studio から再度ワークフローを Publish します。

Elasticsearch	Curator 等を利用して Elasticsearch のスナップショット取得をスケジュール化します。

4.7.2. ログメンテナンス

AWS 環境

対象	方法
Orchestrator (Web サーバー)	IIS アクセスログは IIS の機能にてローテーションを実施します。ローテーション済みのファイルは Windows スケジューラーにて削除スクリプトを実行し削除します。 Windows イベントログはログサイズの上限を指定することでローテーションを実施します。
SQL Server	SQL サーバーログは CloudWatch Logs へ連携し CloudWatch Logs で管理します。 ロボット実行ログは SSMS からジョブを作成し削除します。手順は SQL Server メンテナンスプラン作成手順 を参照してください。
Elasticsearch	Elasticsearch ログは CloudWatch Logs に連携し、CloudWatch Logs で管理します。 ロボット実行ログは利用ユーザーの保存ポリシーに応じて古いインデックス削除します。 設定例は ログメンテナンス を参照してください。

Azure 環境

対象	方法
Orchestrator (Web サーバー)	IIS アクセスログは IIS の機能にてローテーションを実施します。ローテーション済みのファイルは Windows スケジューラーにて削除スクリプトを実行し削除します。 Windows イベントログはログサイズの上限を指定することでローテーションを実施します。
SQL Server	ロボット実行ログは SSMS からジョブを作成し削除します。手順は SQL Server メンテナンスプラン作成手順 を参照してください。

4.7.3. システムメンテナンス

AWS 環境

対象	方法
Orchestrator (Web サーバー)	必要に応じて EC2 の再起動などをスケジュールします。
SQL Server	システムメンテナンスウィンドウを適切に設定します。Single-AZ 構成の場合、メンテナンスが完了するまでは SQL サーバーは利用できません。Multi-AZ 構成としている場合、プライマリからセカンダリへの切り替え時に数分間の SQL サーバー利用不可時間が生じます。 ● 参考: Amazon RDS での高可用性 (マルチ AZ) 必要に応じて以下のジョブを SSMS のジョブ機能から登録します。ジョブ実行前には IIS を停止し、不用意な DB アクセスが発生しないように制御することが推奨されます。 ● DB 整合性チェック ● インデックス再構築 ● 統計情報更新

AWS での設定例は システムメンテナンス を参照してください。

Azure 環境

対象	方法
Orchestrator (Web サーバー)	IaaS (Azure Virtual Machines)の場合には、必要に応じて VM の再起動などをスケジュールします。 PaaS (Azure App Service)の場合は、ユーザーによるシステムメンテナンスは不要です。
SQL Server	必要に応じて以下のジョブを SSMS のジョブ機能から登録します。ジョブ実行前には IIS を停止し、不用意な DB アクセスが発生しないように制御することが推奨されます。 ● DB 整合性チェック ● インデックス再構築 ● 統計情報更新

4.8. トラブルシューティング

4.8.1. 障害対応体制・フロー

Orchestrator 障害による業務インパクトを最小限に抑えるため、あらかじめ障害時の対応フローと担当者を定めておくことを推奨いたします。

4.8.2. 障害対応手順

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、その履歴は IIS ログ、HTTPERR ログに記録されます。

発生する障害のタイプによって収集すべき情報が異なります。Orchestrator トラブルシューティングガイドを参照し、障害に備えて情報収集の手順をあらかじめご確認いただくことを推奨いたします。

4.9. Orchestrator バージョンアップ

UiPath Orchestrator は新機能追加、不具合修正のために定期的に新規バージョンがリリースされます。新規リリースに伴い、既存バージョンのサポート終了日が確定されます。詳細については プロダクトライフサイクルのサイト をご参照ください。

新規バージョンの新機能、不具合修正などの変更に関する情報は 各リリースノート をご参照ください。

Orchestrator バージョンアップの具体的な手順については、Orchestrator バージョンアップガイド をご参照ください。

実作業前に作業漏れやミスを未然に防止するためにバージョンアップの手順書を作成し、関係者でレビューを行うことを推奨いたします。

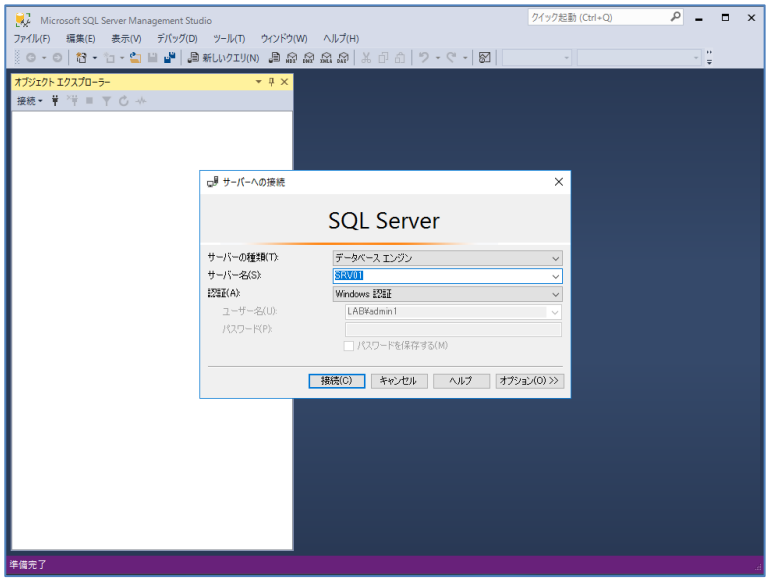
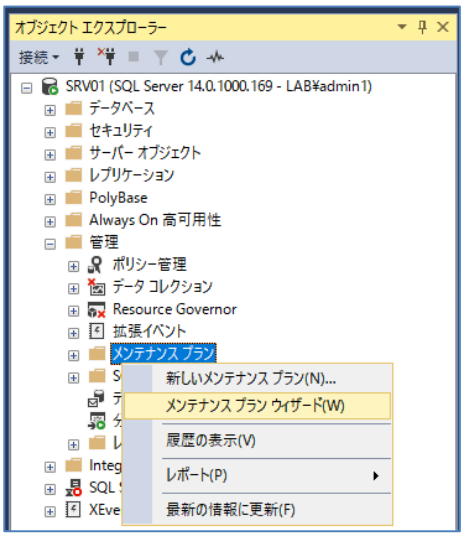
また検証環境が構築されている場合には、事前にリハーサルを実施し手順の精緻化を行うことを推奨いたします。

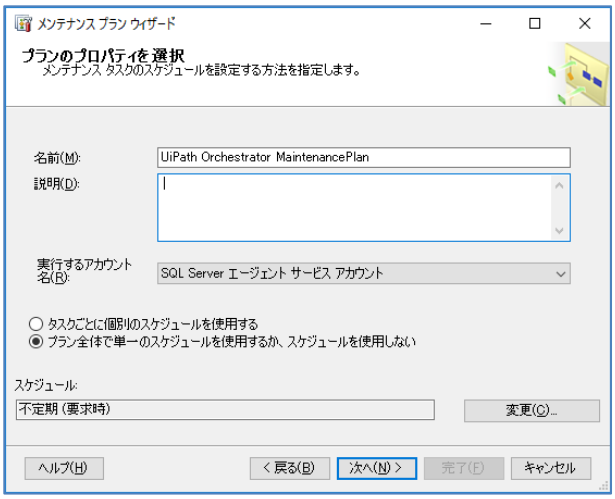
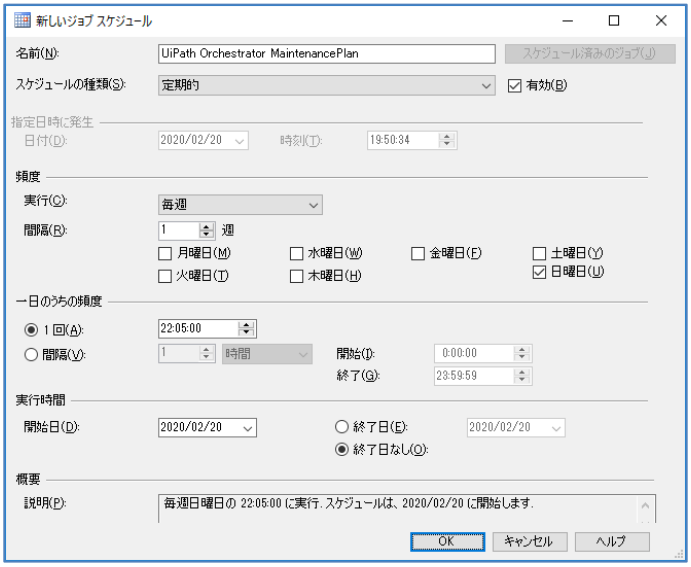
5. Appendix

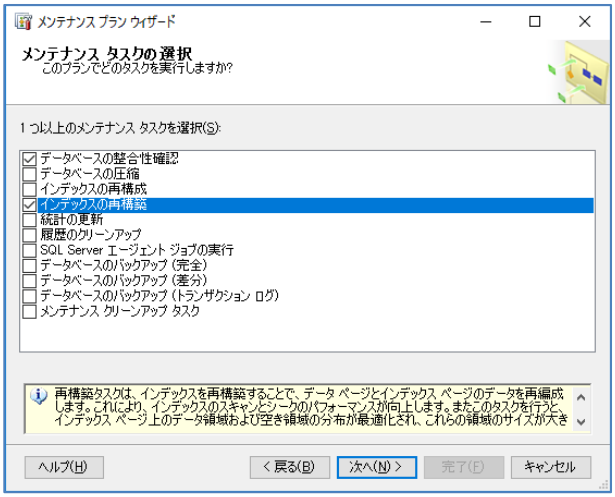
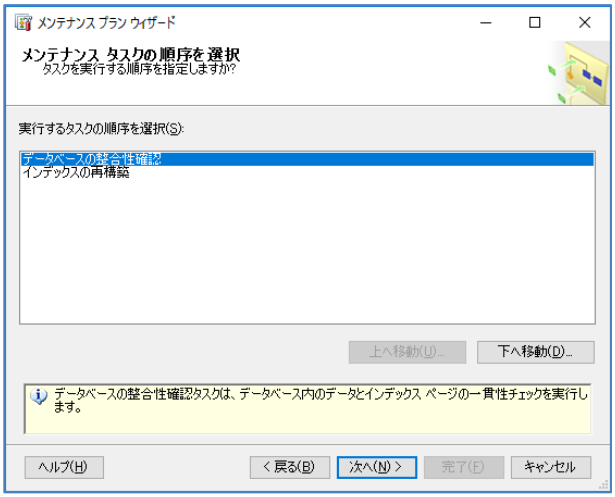
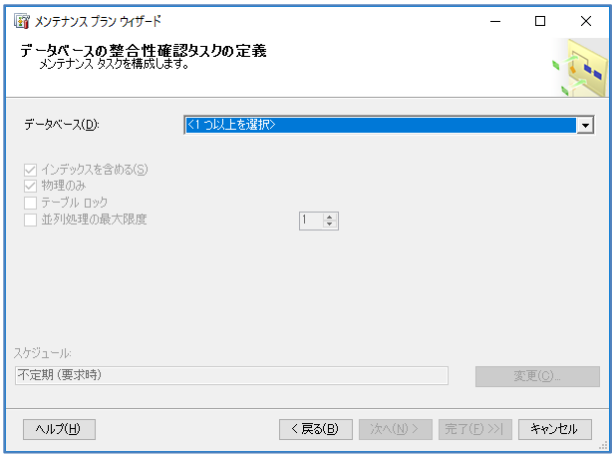
5.1. SQL Server メンテナンスプラン作成手順

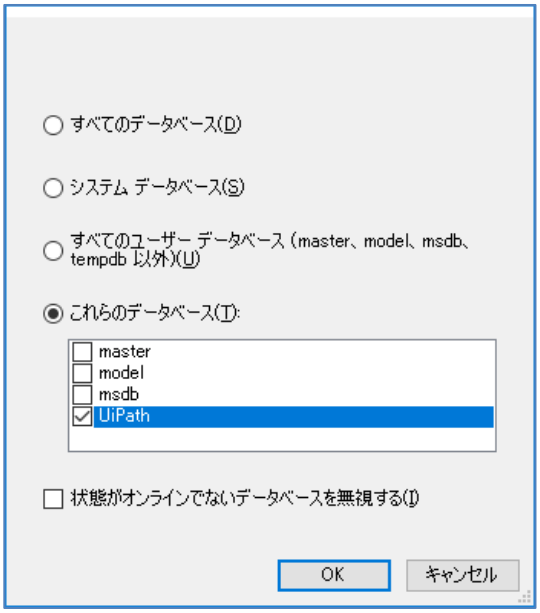
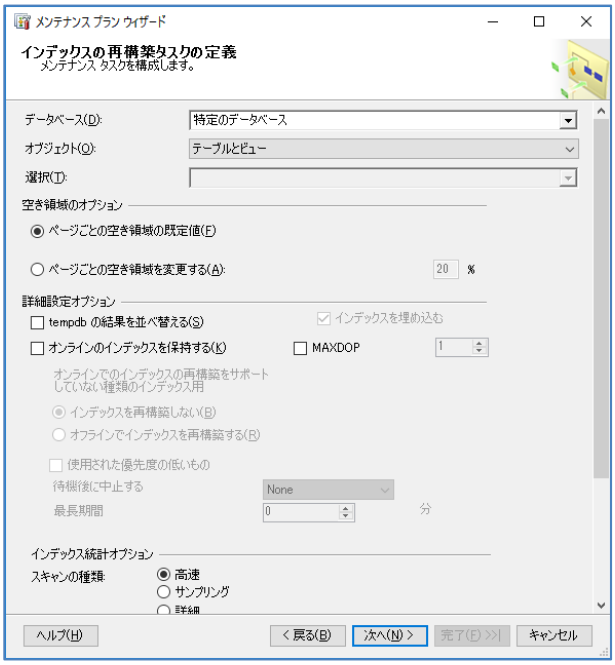
SQL Server Management Studio のウィザードを使ったメンテナンスプランの作成は次の手順になります。 画面は SQL Server 2017 のものになります。バージョンによって一部差異がありますので、ご利用されるバージョンのマニュアルも合わせて参照してください。

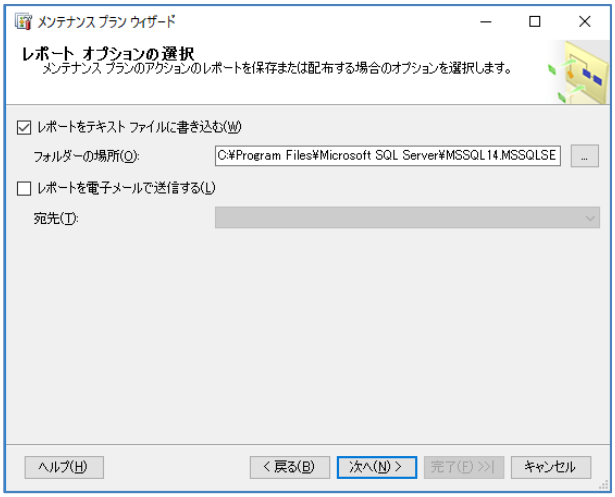
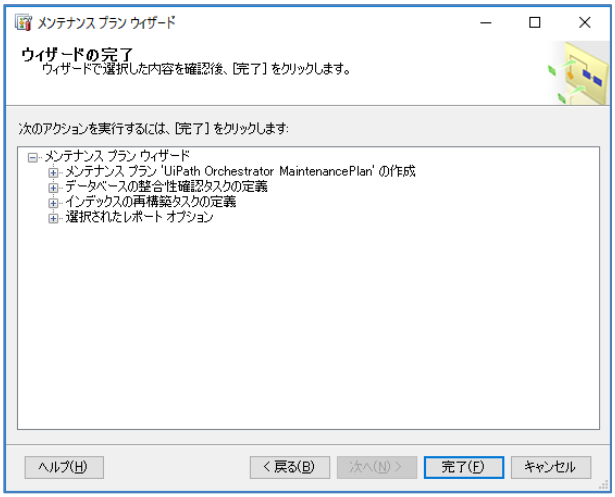
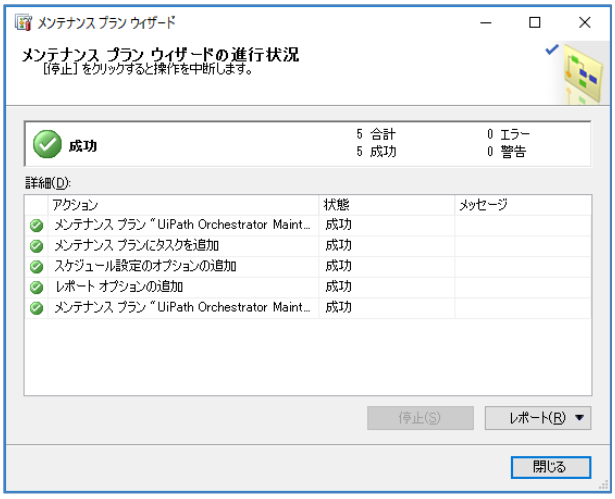
注意: メンテナンスプランの作成・実行には **SQL Server** エージェント サービスが実行中である必要があります。 実行されていない場合は [SQL 構成マネージャー] からサービスの開始をおこなってください。

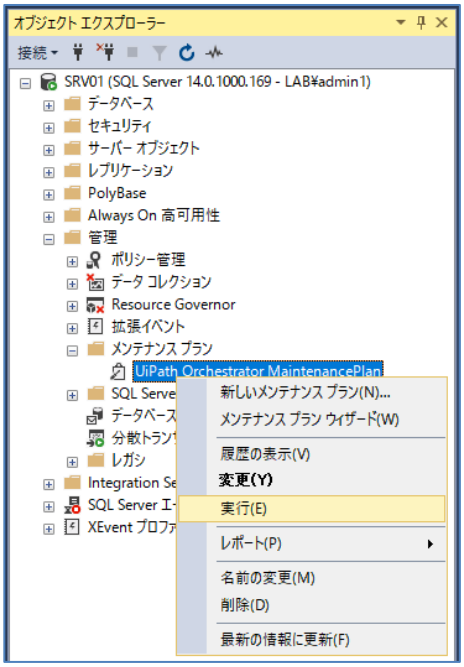
	SQL Server Management Studio を開いて UiPath のデータベースが格納されているインスタンスに接続します。
	左のツリーペインから **[管理] -> [メンテナンスプラン]** を選択して、右クリックメニューから **[メンテナンス プラン ウィザード]** を選択します。

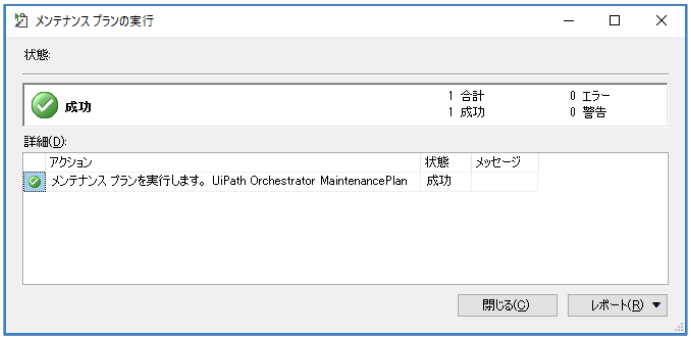
	メンテナンスプランウィザードが開きますので、 [次へ] を押下します。
	メンテナンスプランの名前とスケジュールを設定します。
	スケジュールは毎週や毎月などを実施したいタイミングを選択します。 警告: スケジュールによる自動実行を行う場合は Windows のタスク機能などを利用して、Orchestrator が稼働しているホスト上で事前に IIS のサービスも停止するようにスケジュールしてください。

	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。
	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。
	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。

<table>
<tr>
<td> 
</td>
<td>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ます)</td>
</tr>
<tr>
<td> 
</td>
<td>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。</td>
</tr>
</table>

	レポートの出力先を選択します。
	設定内容を確認し、問題なければ [完了] を押下します。
	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。

<table>
<tr>
<td> 
</td>
<td>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。

注意: メンテナンスプラン動作確認も事前に Orchestrator 停止することを推奨します。</td>
</tr>
<tr>
<td> 
</td>
<td>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。</td>
</tr>
</table>

5.2. Elasticsearch スナップショット取得とリストア手順

本節では Elasticsearch のスナップショット (バックアップ) の取得手順について説明します。スナップショット取得には Elastic 社が提供している Curator と呼ばれるツール、もしくは Elasticsearch API を使用します。

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。

5.2.1. Curator のインストール

Curator の Installation サイト からインストーラーをダウンロードしてください。

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。また curator のインストールディレクトリにあらかじめパスを通してください。たとえば Windows MSI を使用している場合には環境変数 **PATH** に **C:\Program Files\elasticsearch-curator** を追加します。

Curator の設定

Curator を実行するユーザーの %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
```

5.2.2. リポジトリの作成

Elasticsearch 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ーが作成されます。

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",
 "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"
 }
}
```

5.2.3. 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
    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
    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" : [
  {
   "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。

5.2.4. 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。

5.2.5. リストア手順

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 データベースのバックアップとリストア > Elasticsearch 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:
      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)
```

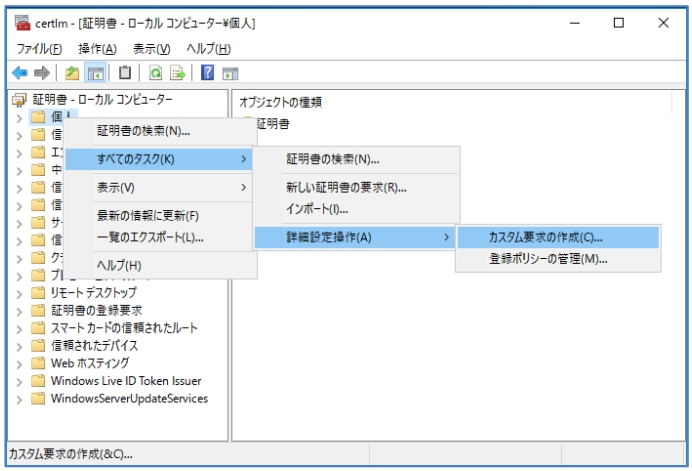
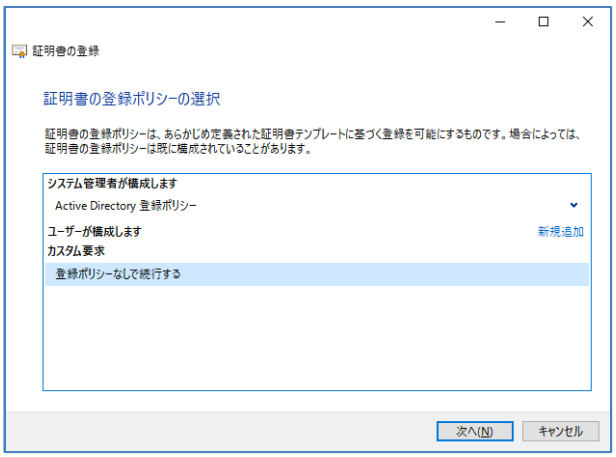
リストアが完了するとインデックスは自動的にオープン状態となります。スナップショットに含まれていないリストア対象外のインデックスの状態は変更されないので、 クローズ状態のものでオープンが必要なインデックスについては以下の API でオープン状態にしてください。

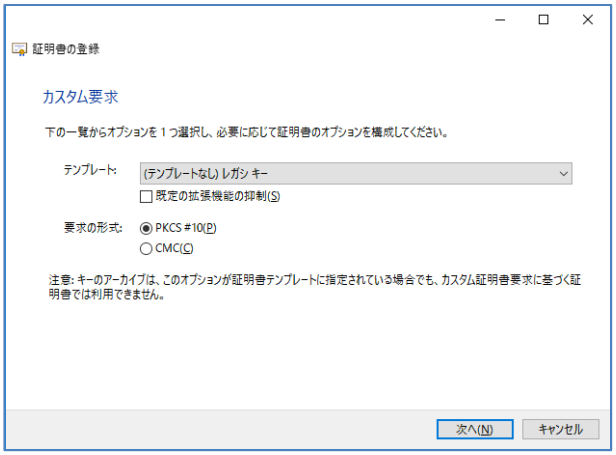
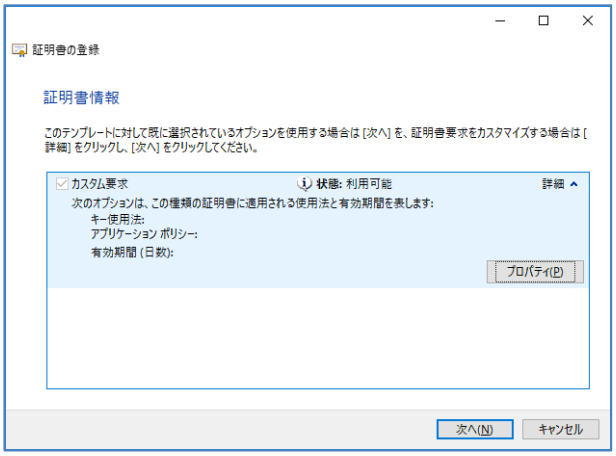
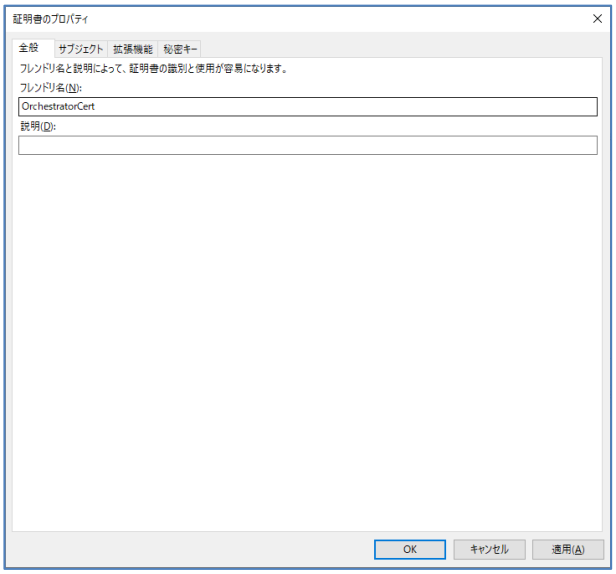
```
POST インデックス名/_open
```

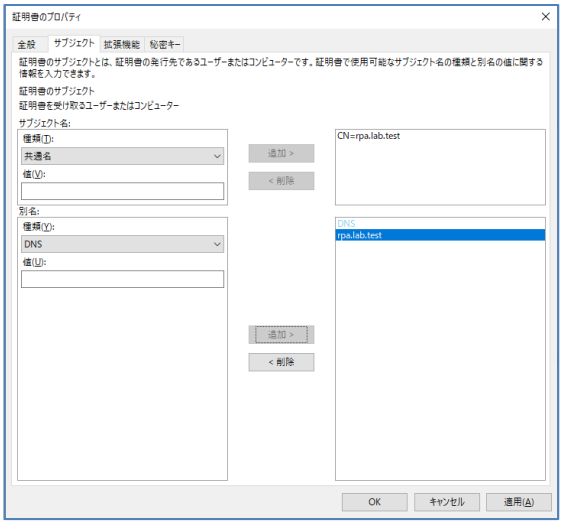
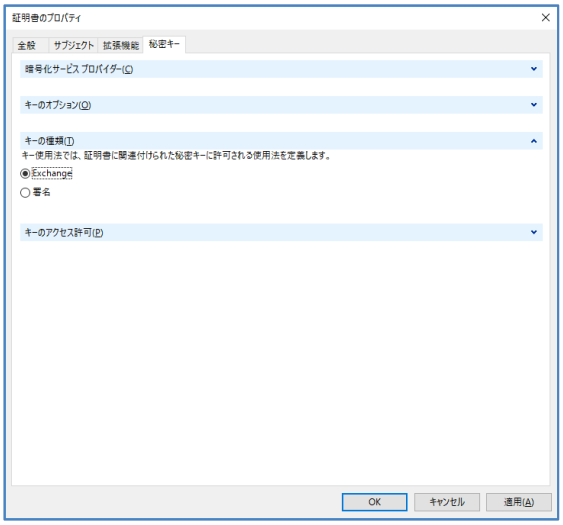
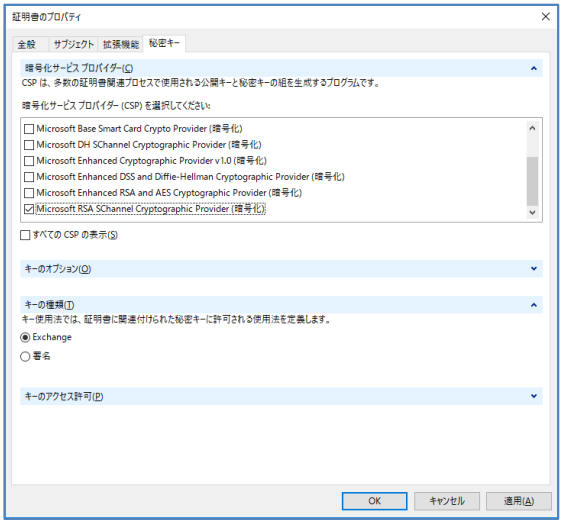
以上でスナップショットのリストアは終了です。

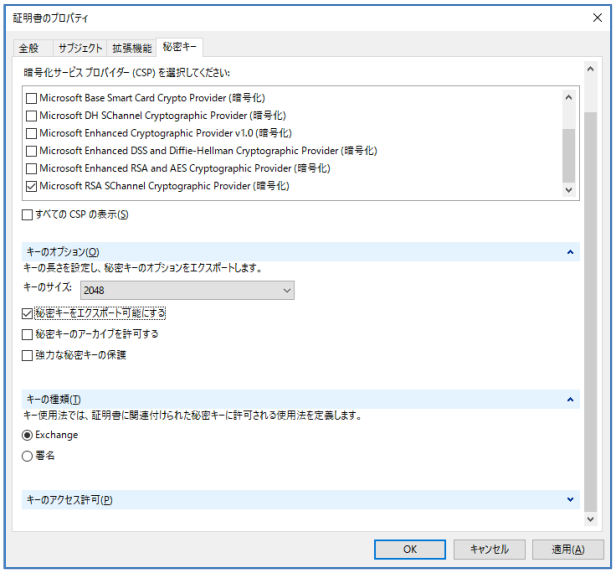
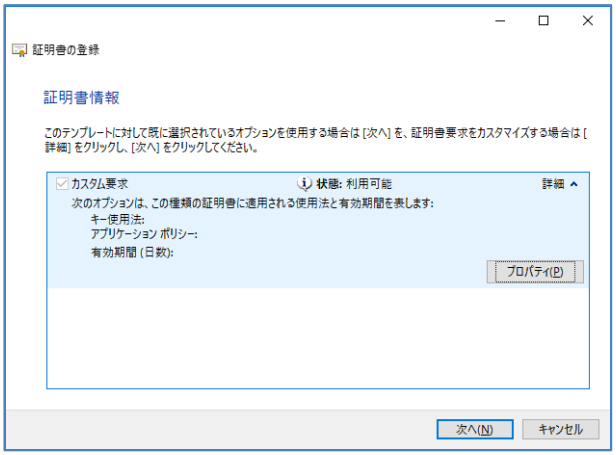
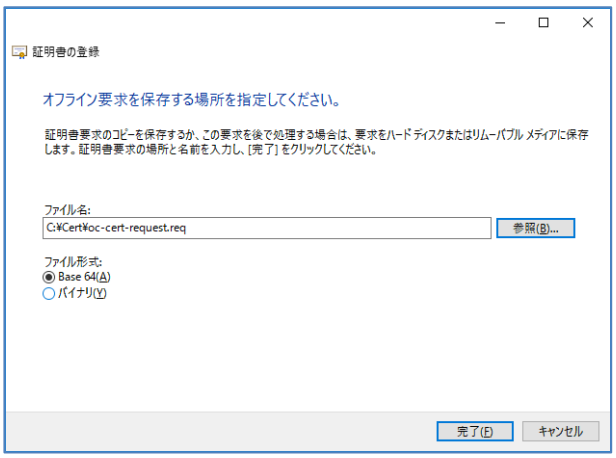
5.3. Microsoft ドメイン証明機関(CA)によるサーバー証明書発行

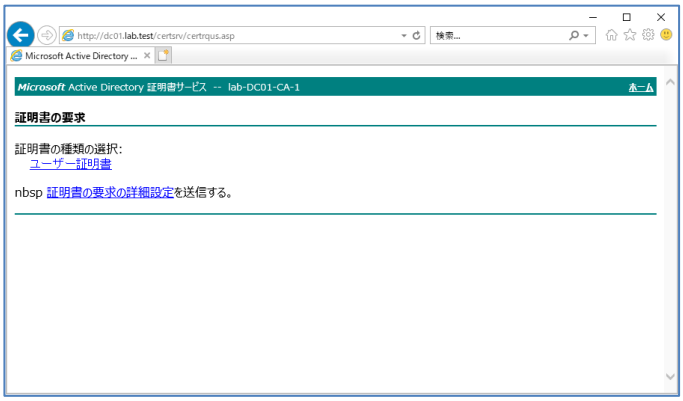
- Microsoft ドメイン CA を使用してサーバー証明書を発行する手順について説明します。
- 前提として Orchestrator 2 台の冗長構成環境を想定しています。シングル構成で省略可能なステップについては注意書きをしています。

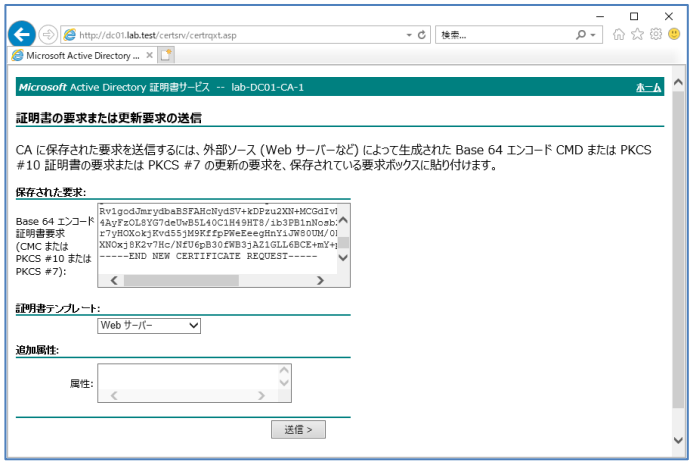
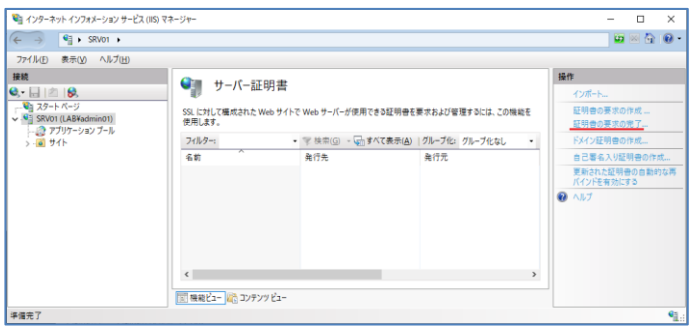
	Orchestrator サーバーにおいて、コマンドプロンプトで **certlm.msc** を実行し、ローカルコンピューターの証明書ストアを表示します。 個人 (右クリック) > すべてのタスク > 詳細設定操作 > カスタム要求の作成 をクリックします。
	証明書の登録ウィザードが開始されます。次へ をクリックします。 カスタム要求 > 登録ポリシーなしで続行する を選択し、次へ をクリックします。

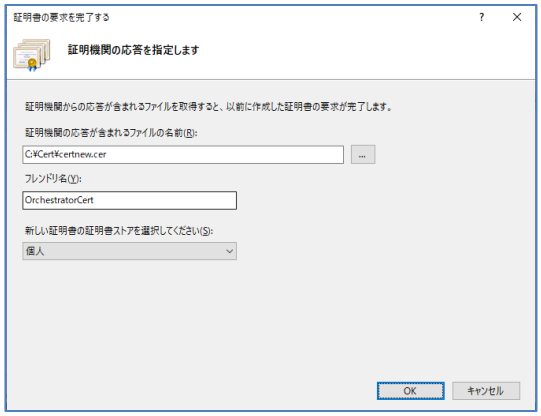
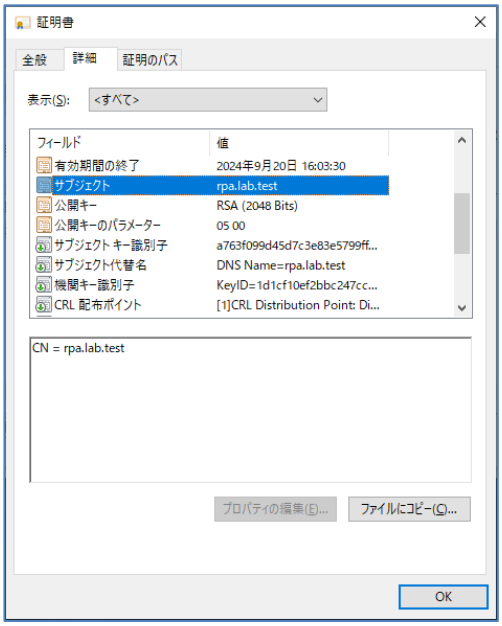
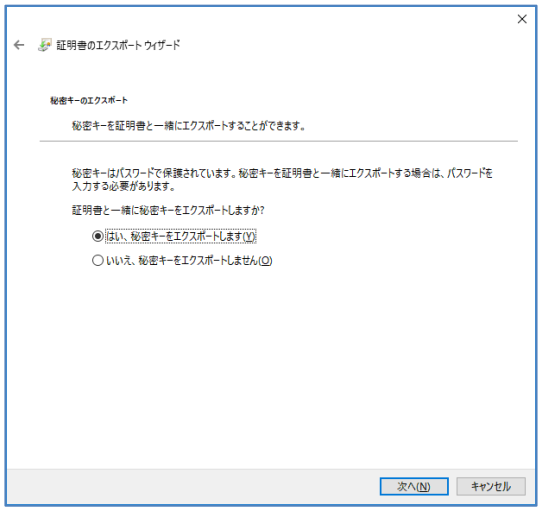
	**(**テンプレートなし**)** レガシ キーを選択し、要求の形式として **PKCS #10** を選択し、次へをクリックします。
	カスタム要求の詳細をクリックし、プロパティ をクリックします。
	[全般] タブにてフレンドリ名として適切な名前を指定します。

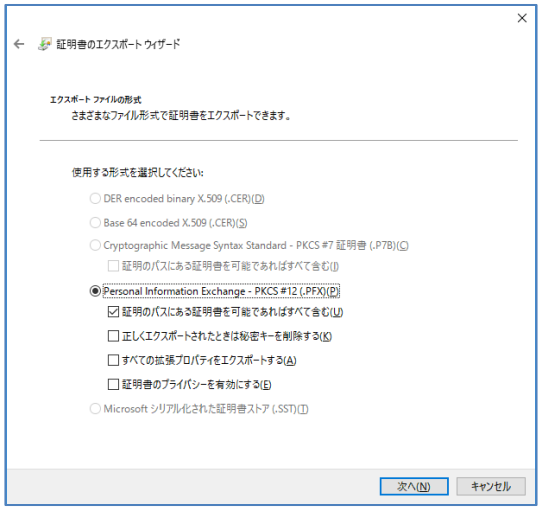
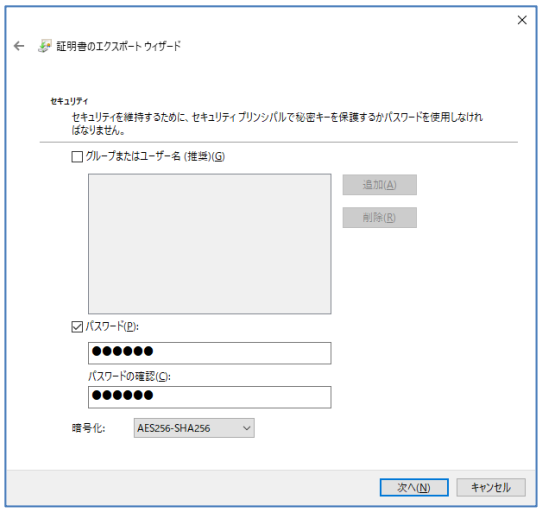
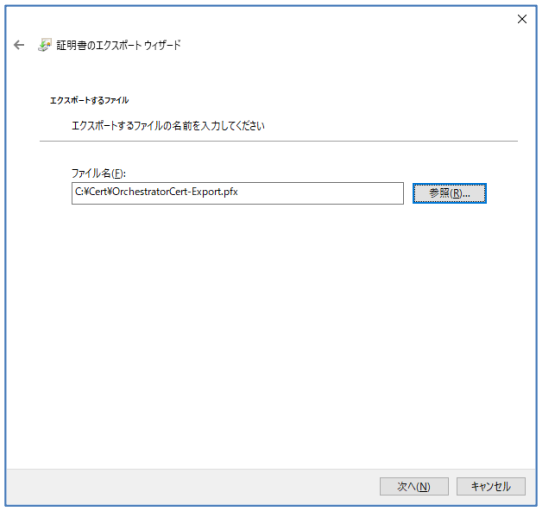
	サブジェクト タブで次の情報を入力し、それぞれ 追加 ボタンをクリックします。*”rpa.lab.test”* は Orchestrator の FQDN を指定します。 サブジェクト名: 種類: 共通名 値: rpa.lab.test 別名: 種類: DNS 値: rpa.lab.test
	[秘密キー] タブにて キーの種類 を展開し、**Exchange** を選択します。
	暗号化サービスプロバイダー を展開し、既定値の Microsoft Strong Cryptographics Provider はオフにし、 **Microsoft RSA Schannel Cryptographics Provider** をオンにします。

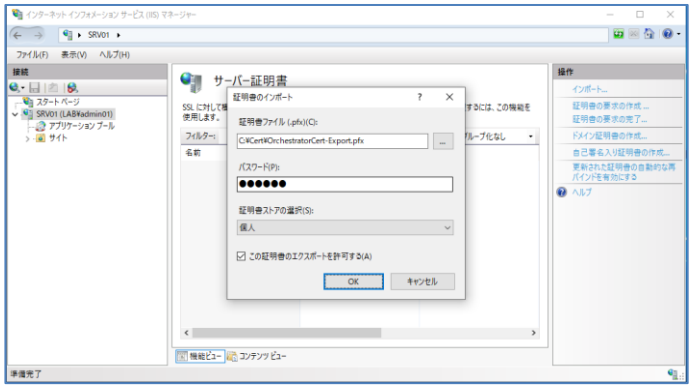
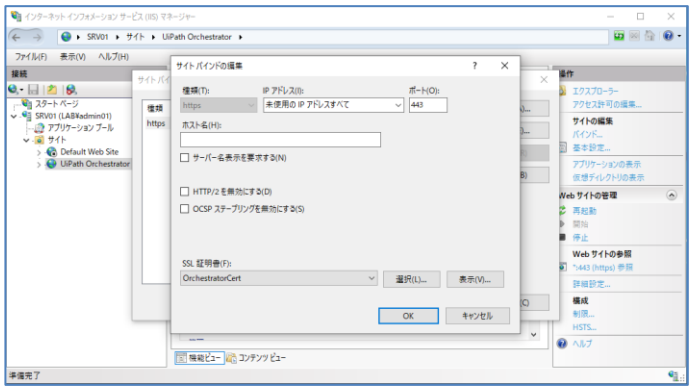
	キーのオプションを展開し、キーのサイズとして **2048** を選択し、秘密キーをエクスポート可能にする をオンにし、**OK** をクリックします。
	次へ をクリックします。
	ファイル形式として **Base64** を選択し、要求ファイルを適切なディレクトリに保存します。

	要求ファイルをドメイン CA 管理者に提供し、証明書発行を依頼します。 証明書 Web サービスを使用して証明書発行依頼を提出するには次の手順を実行します。 ※ この手順の要否はドメイン CA 管理者にご確認ください。この機能を利用するには CA ホストにて「証明機関 Web 登録」の役割を有効化します。 証明書 Web サービスにアクセス可能なマシンにて、ブラウザーで URL にアクセスします。URL とアクセスに必要なアカウントはドメイン CA 管理者にご確認ください。 サイトが表示されましたら信頼済みサイトに登録します。 証明書を要求する をクリックします。
	証明書の要求の詳細設定 をクリックします。
	Base 64 エンコード CMC または PKCS #10 ファイルを使用して証明書の要求を送信するか、または Base 64 エンコード PKCS #7 ファイルを使用して更新の要求を送信する をクリックします。 ※ このページは Chrome/Edge(Chrominum モード)/Firefox ではスキップされます。

<table>
<tr>
<td> 
</td>
<td>保存された要求には、先ほどファイルとして保存した内容をメモ帳で開きコピー＆ペーストします。
証明書テンプレートは Web サーバー を選択し、送信 をクリックします。</td>
</tr>
<tr>
<td> 
</td>
<td>証明書が自動発行される場合には、Base 64 エンコードにて証明書のダウンロード をクリックします。
自動的に証明書が発行されない場合には、ドメイン CA 管理者に証明書を発行するように依頼し、証明書ファイル (*.cer) を入手します。</td>
</tr>
<tr>
<td> 
</td>
<td>発行された証明書を次の手順で Orchestrator Web サーバーに登録します。
● Orchestrator サーバーにドメインユーザーでログインします。
● IIS マネージャー > サーバー証明書 > 証明書の要求の完了 をクリックします。</td>
</tr>
</table>

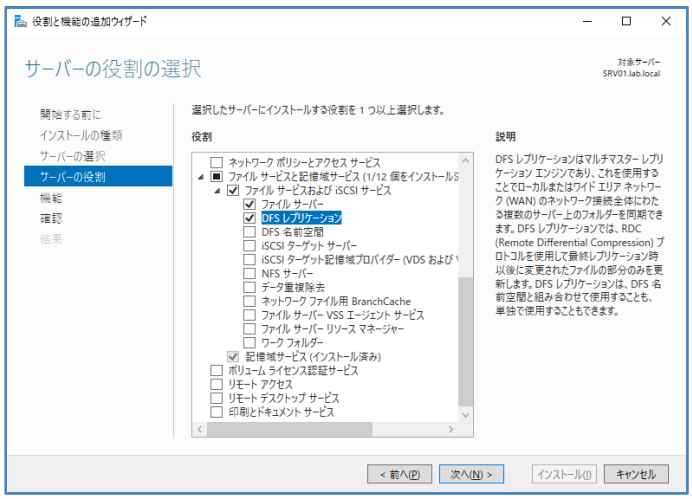
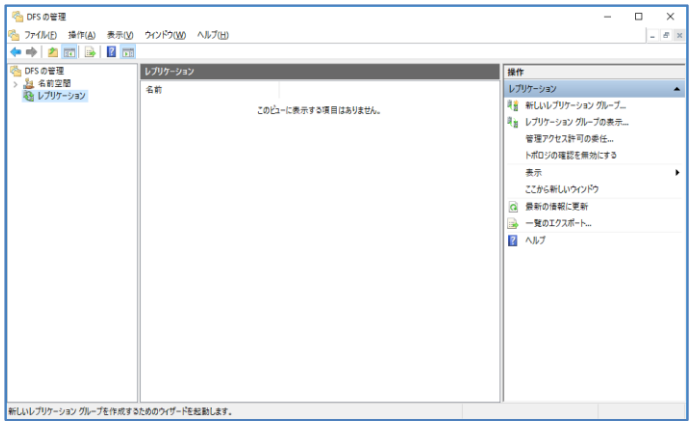
	作成された証明書ファイル (*.cer) を指定し、フレンドリ名を指定します。証明書ストアは「個人」を指定します。OK をクリックします。
	証明書をダブルクリックして表示し、詳細タブにてサブジェクトおよびサブジェクト代替名が *rpa.lab.test* となっていることを確認します。 Orchestrator がシングル構成の場合は下記エクスポート・インポートの手順は不要のため、最後のバインド設定のみ行ってください。 冗長構成の場合にはファイルにコピーをクリックします。
	はい、秘密キーをエクスポートします を選択して、次へ をクリックします。

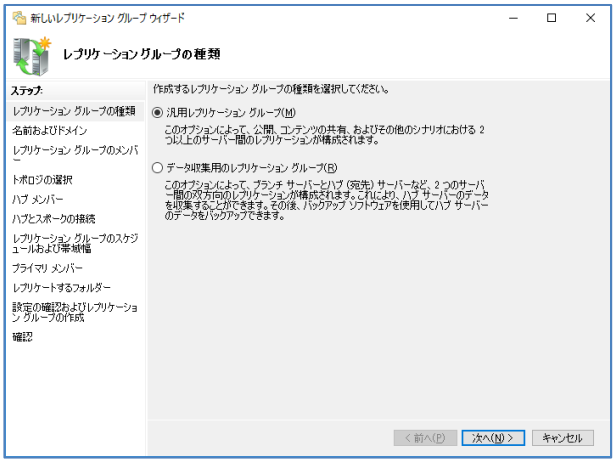
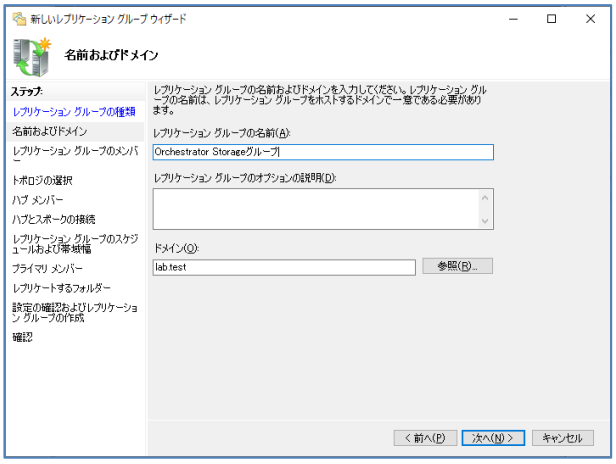
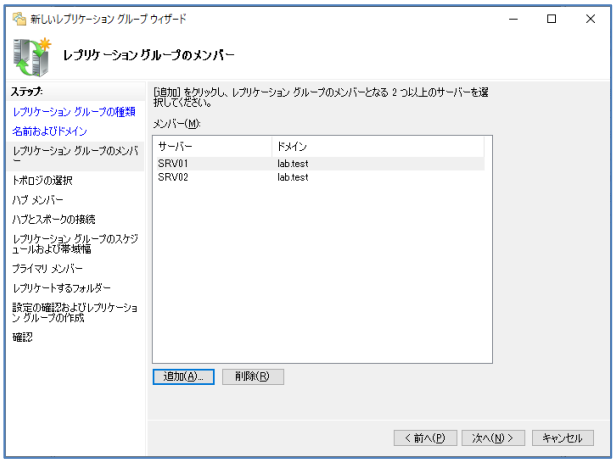
	PKCS#12 が選択されていることを確認して、次へ をクリックします。
	パスワードを設定して、次へ をクリックします。 ※ Windows Server 2019 以降では暗号化オプションとして TripleDES-SHA1 または AES256-SHA256 が選択できます。Windows Server 2016 以前の OS にインポートする場合は TripleDES-SHA1、Windows 2019 以降の OS にインポートする場合は AES256-SHA256 を選択します。
	任意のディレクトリに保存して、次へ をクリックし、証明書のエクスポートウィザードを終了します。 証明書の登録ウィザードで 完了 をクリックします。

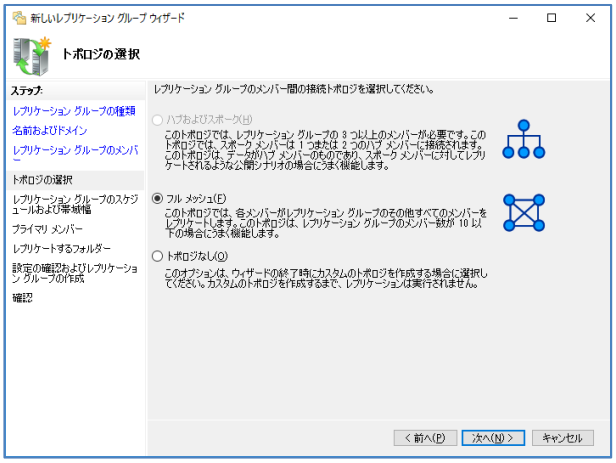
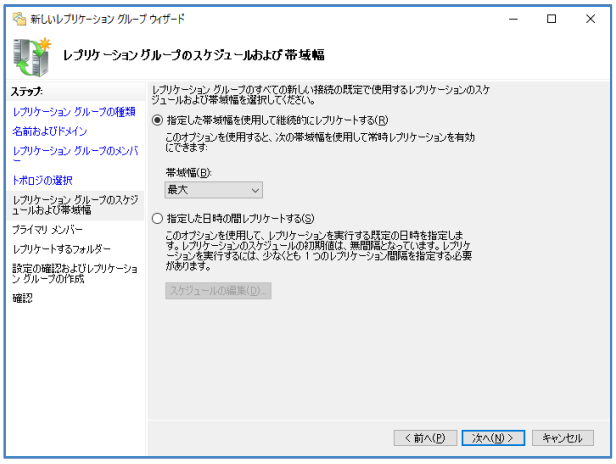
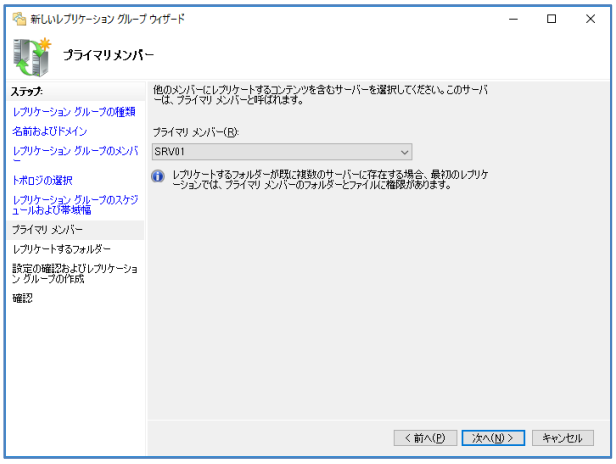
	Orchestrator#1, Orchestrator#2 にてそれぞれエクスポートした秘密キー(*.pfx) を配置します。 IIS マネージャーを起動し、サーバー証明書 > インポートをクリックします。 秘密キーとパスワードを指定し、**OK** をクリックします。
	UiPath Orchestrator サイトを選択し、右の操作ペインからバインドをクリックし、https を編集します。 サイトバインドの編集 > SSL 証明書にてインポートした証明書を選択して、**OK** をクリックします。 Orchestrator#1, Orchestrator#2 にて上記手順をそれぞれ実行します。

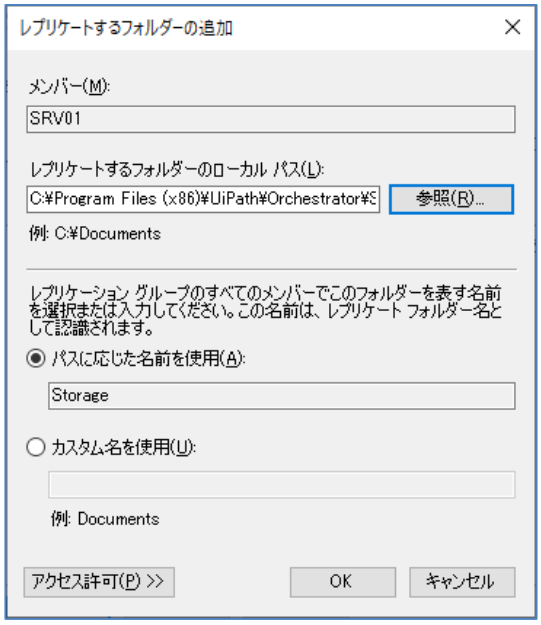
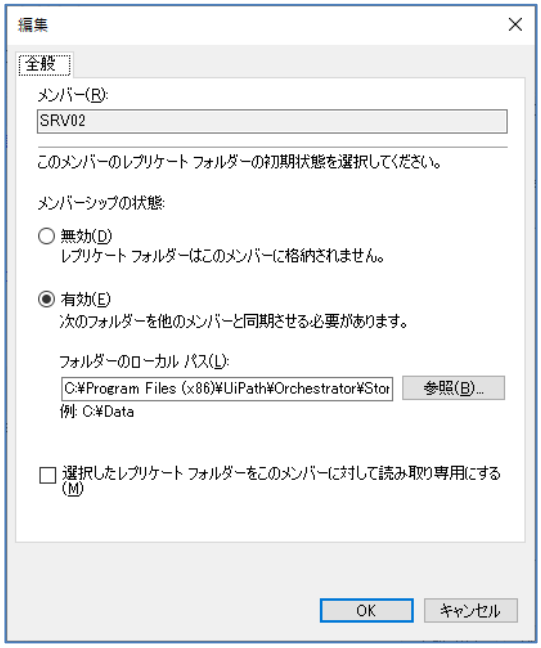
5.4. DFS レプリケーション設定手順

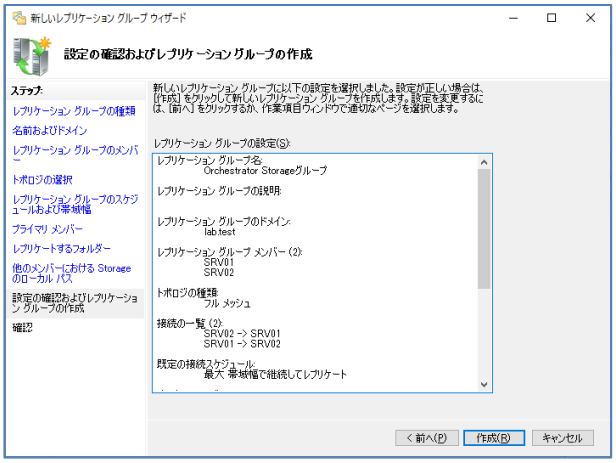
冗長化のため複数の NuGet パッケージディレクトリを DFS レプリケーションによって同期することができます。前提として AD 環境にてすべての Orchestrator サーバーが同一ドメインに参加している必要があります。本節では 2 台の Orchestrator ローカルディレクトリを同期する手順について説明します。

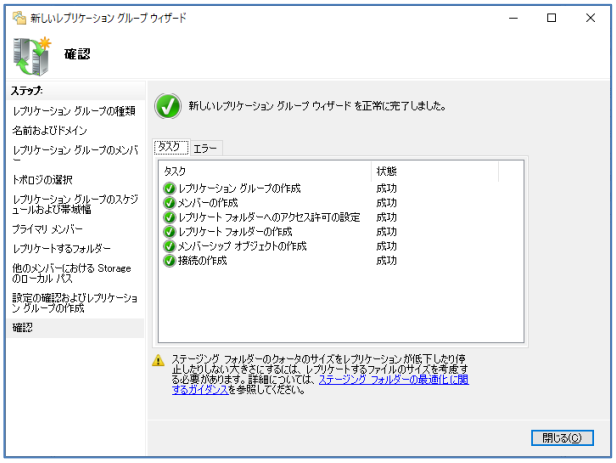
	それぞれの Orchestrator ホストにおいて、サーバーマネージャを使用して **DFS** レプリケーションと関連する役割をインストールします。
	いずれかの Orchestrator ホストにて DFS の管理 を起動し、 新しいレプリケーショングループ をクリックします。

	汎用レプリケーショングループ を選択します。
	適切なレプリケーショングループ名を入力します。
	同期対象となる NuGet パッケージディレクトリを持つ Orchestrator サーバーのコンピューターオブジェクトをすべて追加します。

	フルメッシュを選択します。
	既定値を使用します。
	初期転送のコピー元となるプライマリメンバーを選択します。

<table>
<tr>
<td> 
</td>
<td>同期するフォルダーパスを指定します。必要に応じて変更します。
● 既定値: C:\Program Files (x86)\UiPath\Orchestrator\Storage</td>
</tr>
<tr>
<td> 
</td>
<td>他メンバーのパスを編集し、同様に Storage ディレクトリを指定します。</td>
</tr>
</table>





作成をクリックし、レプリケーショングループ作成のウィザードを終了します。

しばらく経過した後、プライマリメンバーのディレクトリに追加したパッケージファイル (*.nupkg) が他のメンバーのディレクトリにも同期されることを確認します。

5.5. AWS 利用時の運用監視設定例

AWS を利用した際の運用監視の設定例です。

各項目について、運用監視設定のおすすめ度を「優先度」として表中に記載しております。

優先度「High」のものは、システムの安定運用の観点から推奨される項目となります。

5.5.1. サービスイベント監視

監視対象		監視内容	優先度	監視間隔	統計	監視方法	アラート通知条件
AP サーバー (EC2)	Orchestrator のサービス死活監視	Orchestrator プロセスの稼働状況を監視	High	1 分毎	最小	ELB 標準 CloudWatch メトリクス + CloudWatch アラーム	UnHelathyHostCount >=1 (ELB の HealthCheck で Orchestrator の /api/Status/Get を監視している前提)
	EC2 自体の死活監視	ELB 配下の EC2 が正常にサービス提供されていることを監視	Mid	1 分毎	最小	EC2 標準 CloudWatch メトリクス + CloudWatch アラーム	・StatusCheckFailed > 0 もしくは ・INSUFFICIENT(データ不足)
	IIS のレスポンス	ELB の Latency を監視	Mid	1 分毎	最大	ELB 標準 CloudWatch メトリクス + CloudWatch アラーム	Latency を監視
	IIS のエラー	ELB の HTTP エラーを監視	High	1 分毎	最大	ELB 標準 CloudWatch メトリクス + CloudWatch アラーム	HTTP-5XX-error を監視
DB サーバー (RDS)	RDS サービス死活監視	DB インスタンスの稼働状態を監視	Mid	1 分毎	-	RDS イベントサブスクリプション	下記 RDS イベントが発生した場合 ・フェイルオーバー

Elasticsearch	Elasticsearch サービス死活監視	Elasticsearch Service のクラスターステータスを監視	Mid	1 分毎	最小	ES 標準 CloudWatch メトリクス＋ CloudWatch アラーム	下記クラスターステータス(※)になった場合 ・ClusterStatus.red > 0 ・ClusterStatus.yellow > 0
	Kibana サービス死活監視	Elasticsearch Service の KibanaHealthyNodes を監視	Mid	1 分毎	最小	Kibana 標準 CloudWatch メトリクス＋ CloudWatch アラーム	KibanaHealthyNodes < 1
ElastiCache(Redis)	ElastiCache サービス死活監視	死活監視に有効なメトリクスがないため監視しない		-	-	-	-

5.5.2. ログ監視

監視対象	監視内容	優先度	監視間隔	統計	監視方法	アラート通知条件	フィルター内容		
							ログレベル	ログ文言	エラー内容
AP サーバー(EC2)の Windows イベントログ	システムログ、アプリケーションログのエラーの内、OC 稼働に影響するイベントを監視	Mid	1 分	最大	Cloud Watch Logs Agent + Cloud Watch Logs+ Cloud Watch アラーム	右記ログ文言をフィルターするメトリクスを設定し、そのメトリクスが 1 以上(フィルタ	Error	The transaction log for database 'UiPath' is full	DB トランザクションログが一杯
		Mid					Error	Transaction not connected, or was disconnected	SQL Server への接続ができない
		High					Error	License expired! Starting jobs is no longer possible!	ライセンス有効期限切れ
		Mid					Fatal	Error during Orchestrator start-up	Orchestrator の起動中にエラー発生

		Mid				合致)となった場合	Error	StackExchange.Redis.RedisConnectionException: No connection is available to service this operation	Redis 接続で例外発生

検知するイベントログメッセージについては イベントログ監視 もご参照ください。

5.5.3. リソース監視

リソース監視の要否、閾値については環境への依存度が高いためカスタマイズが必要です

監視対象		監視内容	優先度	監視間隔	統計	監視方法	アラート通知条件
CPU 使用率	AP サーバー(EC2)	サービス毎の CPU 全体の使用率を監視	Low	5 分毎	平均	CloudWatch 標準メトリクス＋ CloudWatch アラーム	下記条件が 2 回連続発生した場合 ・[CPUUtilization](CPU 使用率(%)) > 80
	DB サーバー(RDS)		Low	5 分毎	平均	CloudWatch 標準メトリクス＋ CloudWatch アラーム	下記条件が 2 回連続発生した場合 ・[CPUUtilization](CPU 使用率(%)) > 80
	Elasticsearch		Low	5 分毎	平均	CloudWatch 標準メトリクス＋ CloudWatch アラーム	下記条件が 2 回連続発生した場合 ・[CPUUtilization](CPU 使用率(%)) > 80
	ElastiCache (Redis)		Low	5 分毎	平均	CloudWatch 標準メトリクス＋ CloudWatch アラーム	下記条件が 2 回連続発生した場合 ・[CPUUtilization](CPU 使用率(%)) > 80
メモリ使用率	AP サーバー(EC2)	サービス毎の CPU	Low	5 分毎	平均	CloudWatch カスタムメトリクス＋ CloudWatch アラーム	下記条件が 3 回連続発生した場合 ・[Memory % Commited Bytes in Use] (メモリ使用率) > 90％

	DB サーバー(RDS)	全体の使用率を監視	Low	5 分毎	平均	CloudWatch 標準メトリクス＋ CloudWatch アラーム	下記条件が 3 回連続で発生した場合 ・(※残メモリ率 5%未満)
	Elasticsearch		Low	5 分毎	平均	CloudWatch 標準メトリクス＋ CloudWatch アラーム	下記条件が 3 回連続で発生した場合(*1) ・[JVMMemoryPressure](JVM メモリ使用率(%)) > 90
	ElastiCache (Redis)		Low	5 分毎	平均	CloudWatch 標準メトリクス＋ CloudWatch アラーム	下記条件が 3 回連続で発生した場合 ・(※残メモリ率 5%未満)
ディスク使用率	AP サーバー(EC2)	サービス毎のディスク使用率を監視	Mid	5 分毎	最小	CloudWatch カスタムメトリクス＋ CloudWatch アラーム	下記条件が発生した場合 ・[LogicalDisk % Free Space](空きディスク率(%)) < 10
	DB サーバー(RDS)		Mid	5 分毎	最小	CloudWatch 標準メトリクス＋ CloudWatch アラーム	下記条件が発生した場合 (※空きディスク率 10%未満) ・[FreeStorageSpace](空きディスク容量(GB)) < 30
	Elasticsearch		Mid	5 分毎	最小	CloudWatch 標準メトリクス＋ CloudWatch アラーム	下記条件が発生した場合 ・(※空きディスク率 10%未満)

5.5.4. バックアップ

対象		優先度	方法	タイミング	保存期間
AP サーバー(EC2)	システムバックアップ	High	システム変更前に手動で AMI を取得する運用とする	システム変更時	-
	データバックアップ	Low	AWS Backup で EBS のスナップショットを取得する	日次	
DB サーバー(RDS)	データバックアップ	High	自動バックアップ機能を有効化し、バックアップを取得 (※オンラインバックアップ)	日次 2:00-2:30(JST)	7 日間(7 世代)

Elasticsearch	データバックアップ	High	自動スナップショット(クラスター復元用)	毎時	14 日間(※固定変更不可)
S3	データバックアップ	-	バックアップしない。(※もともと最低 3AZ に保存され、パッケージは Orchestrator の機能でバージョン管理されるため)	-	-

5.5.5. ログメンテナンス

対象ログ		メンテナンス内容	メンテナンス方法	ローテーションタイミング	保存期間
ロボプロセス実行端末	実行ログ(Execution.log)	•毎日 0:00 に UiPath Robot の機能で自動的にログローテーションされ、これと同時に、過去 30 日を超えるログがあれば当該ログの削除処理を行っている。(※この保存期間は、各仮想端末上の「<Installation Folder>\NLog.config」で設定している。)	UiPath Robot 標準機能(設定ファイル)	日次	40 日
AP サーバー(EC2)	IIS アクセスログ	•ログファイル肥大化防止のため、IIS アクセスログに日次でローテーションする設定をする。	ローテーション：IIS 標準機能	日次	40 日
		•IIS アクセスログは、週次にて IIS の機能を利用してログの削除を行う。	削除：スクリプトを作成して、Windows タスクスケジューラから定期削除処理を実行		
	Windows イベントログ	•Windows 標準機能で指定したファイルサイズ分を保持。※1ヵ月程度は保持できる見込みのサイズ(512MB)に設定する	Windows 標準機能	-	(512 MB)

DB サーバー	エラーログ(SQLServer ログ) エージェントログ	•当該ログを CloudWatch Logs へ連携し、CloudWatch Logs 側で 30 日保存できるように設定する	AWS RDS 標準機能で CloudWatch Logs に連携	-	40 日
	ロボプロセス実行ログ	•ロボット実行ログが保存されている DB テーブル(Logs テーブル)に対して、保存 40 日を超える DB ログを削除する。	SQL スクリプトを EC2 にインストールした SQL Server Management Studio(SSMS)のジョブで定期実行	毎週月曜日 0：00 開始	40 日
Elasticsearch	検索スローログ インデックススローログ エラーログ ロボプロセス実行ログ	•当該ログを CloudWatch Logs へ連携し、CloudWatch Logs 側で 30 日保存できるように設定する •ロボプロセス実行ログは原則削除しない	Elasticsearch Service 標準機能で CloudWatch Logs に連携	-	40 日

5.5.6. システムメンテナンス

メンテナンス対象		メンテナンス内容	メンテナンス方法	メンテナンス時間
サーバー定期再起動	全サーバー	EC2 以外は AWS がマネージしているため再起動処理は不要 今回、EC2 は定期再起動の対象外とする。	-	-
ロボ実行端末再起動	ロボ実行端末の定期再起動	端末の安定稼働のため定期再起動を実行する	Windows のタスクスケジューラで自身を再起動	毎週月曜日 0：00 開始
DB メンテナンスプラン	DB 整合性チェック	DB 整合性チェックを行う	EC2 にインストールした SQL Server Management Studio(SSMS)のジョブで定期実行	毎週月曜日 ロボプロセス実行ログ削除完了後

	インデックス再構築	インデックスの再構築を行う	(※ ジョブ実行前に IIS の停止を Windows タスクスケジューラから実行、メンテ完了後に起動する設定とする。)	毎週月曜日 DB 整合性チェック完了後
	統計情報更新	統計情報の更新を行う		毎週月曜日 インデックス再構築完了後
メンテナンスウィンドウ	RDS	AWS がパッチ適用等のためにメンテを実行する ※RDS はマルチ AZ 構成のため、メンテナンスウィンドウによって SQL サーバー全体が停止することはない	AWS によるメンテ実行	毎週月曜日 AM3:00-3:30
	ElastiCache(Redis)	AWS がパッチ適用等のためにメンテを実行する ※Redis は Master-Slave 構成のため、メンテナンスウィンドウによって Redis 全体が停止することはない	AWS によるメンテ実行	毎週月曜日 AM2:00-3:00

5.6. ライセンスキャッシュ機能

Orchestrator テナントのセキュリティ設定「ライセンスを検証せずにロボットをオフラインで実行できる合計時間」を利用することにより、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。この機能は旧バージョンでは「切断状態で実行可能な時間 (Run Disconnected Hours) 」というフィールドで設定されていました。

下記の条件がすべて満たされた時にライセンスキャッシュが動作します。

- Orchestrator で AR をロボットとして登録し、プロセスをデプロイする
- AR が Orchestrator に接続済で、Orchestrator からライセンスを付与されている
- 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った場合でも、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。

ただしこの状態から Windows ログアウト、UiPath Robot サービスを再起動または OS を再起動した場合、AR 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。

6. 技術支援のご案内

- UiPath 社では Orchestrator および周辺のテクノロジーに関わる技術支援の有償コンサルティングサービスを提供しております。下記のような課題に対して技術支援が必要なお客様は弊社担当営業までご相談ください。
  - Orchestrator 設計・構築・運用
    - シングル構成または冗長構成での導入支援
    - オンプレミスまたはパブリッククラウド環境への導入支援
    - インストール後のフォルダー・ロール設定など運用ルール策定支援
  - Orchestrator / Studio / Robot のバージョンアップ
    - ベストプラクティスに基づくバージョンアップ作業手順の作成支援
  - Elasticsearch / Kibana 導入・活用
    - ダッシュボード作成によるログ可視化の活用支援

以上