

# Robot Auto Scaling powered by UiPath IT Automation

Documentation: Deployment / Configuration / User-Guide



**Release Version:** 1.5

**Publish Date:** 2020.10.02

**Author:** [andrei.oros@uipath.com](mailto:andrei.oros@uipath.com)

## Table of Contents

Introduction .....	3
What is UiPath IT Automation? .....	4
Components Overview.....	5
Client Tenants .....	5
Database .....	5
Webhook Receiver. Webservice / Function.....	5
Management Orchestrator .....	5
Process Flow .....	6
1. Event Driven .....	6
Step 1. Job events from managed clients .....	6
Step 2. Webhook Events Receiver Service.....	6
Step 3. Trigger a Job to process new events.....	7
Step 4. Process Events .....	7
2. Pooling via Scheduled Job .....	8
Step 1. Trigger for Scheduled Job .....	8
Step 2. Process Client.....	8
Solution Deployment and Configuration .....	9
1. Management Orchestrator Configuration .....	10
Add or Publish RAS Solution Packages.....	10
Queue configuration .....	10
Trigger setup .....	10
Assets configuration.....	12
2. Webhooks Receiver Setup .....	15
3. Database Deployment. Clients Configuration.....	16
Database Setup .....	16
Add managed clients.....	16
Scaling Strategies .....	18
1. High Availability for Pending Jobs .....	18
2. Cost Efficiency .....	18
Solution Customization .....	20
Release Notes .....	21

## Introduction

**UiPath Robots Auto Scaling (RAS)** is a workflow based solution that helps optimize IT Infrastructure costs by making RPA Deployments run more efficiently.

The robot machines are started / stopped automatically as a function of:

**1. Defined Scaling Strategy:**

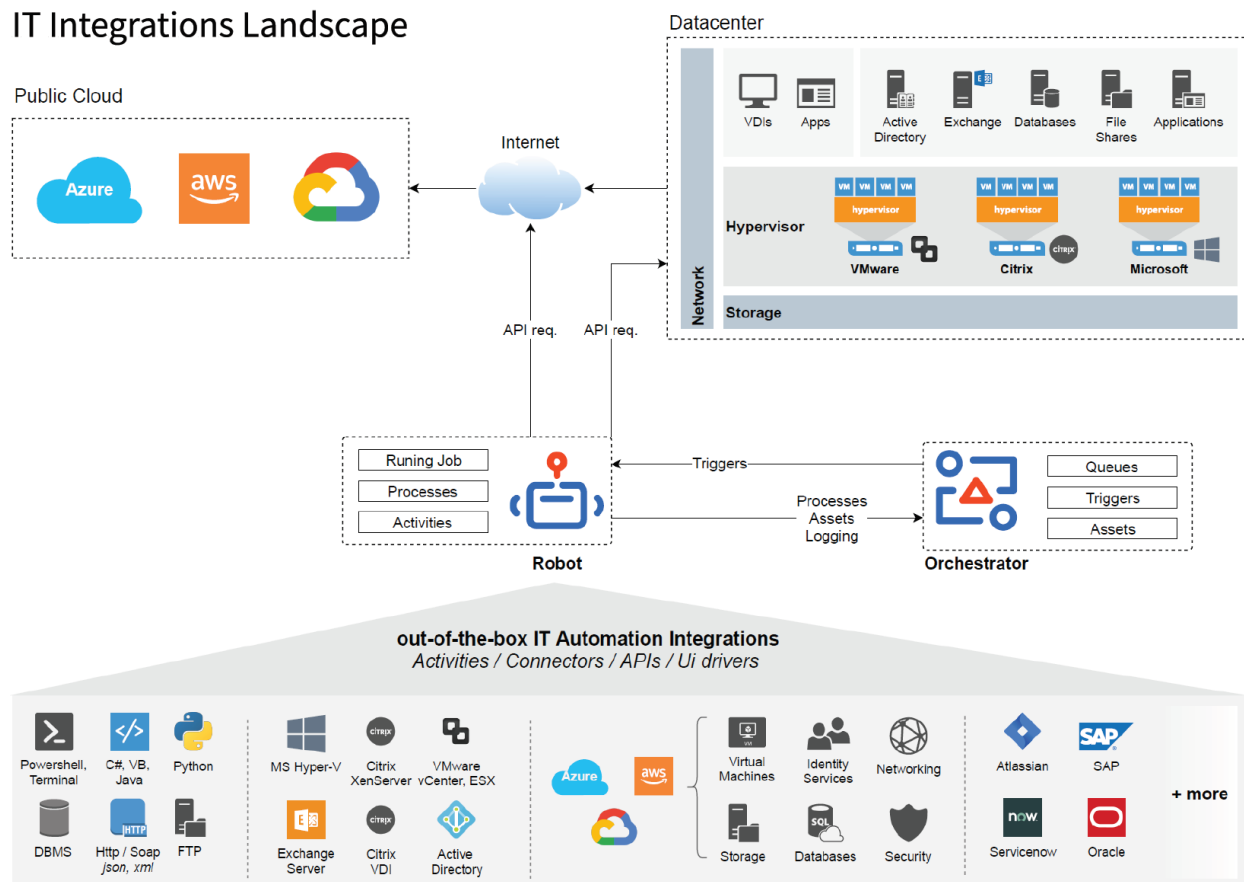
- **hot** Robots count - their machines are always running
- **cold** Robots count - their machines will be allocated/deallocated dynamically

**2. Client's Current State:**

- Jobs (pending, running, completed, stopped, faulted)
- Robots (available, disconnected, busy)

The RAS solution is implemented with the help of the official UiPath IT Automation activities - they empower us to automate use cases from all areas of our IT Ecosystem, in both on-premises datacenters and public clouds:

## IT Integrations Landscape



## What is UiPath IT Automation?

UiPath's platform has builtin capabilities that enable us to automate IT processes. The official IT Automation activities are background running, implemented on top of the official SDKs from vendors such as Microsoft, Amazon, VMware, Citrix and more.

The UiPath IT Automation activities can be included in your projects at no cost from UiPath Studio (any edition) > Manage Packages.

### Official Activities:

Azure, Amazon Web Services (AWS),  
Active Directory, Azure Active  
Directory, VMware vCenter ESXi,  
Citrix Hypervisor, Hyper-V, Exchange  
Server, Microsoft System Center  
.. and more.



Server  
Virtualization



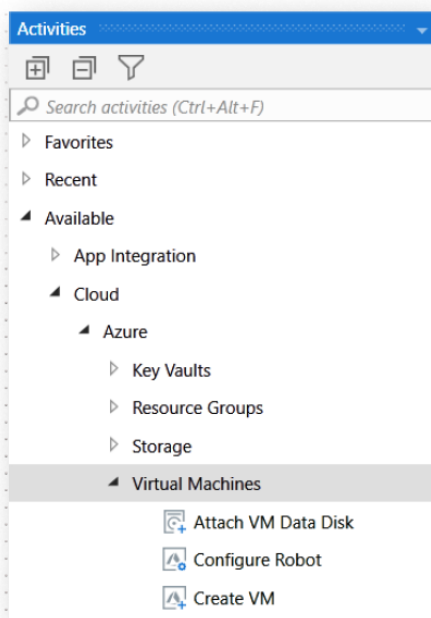
Cloud  
Infrastructure & Services



User Management

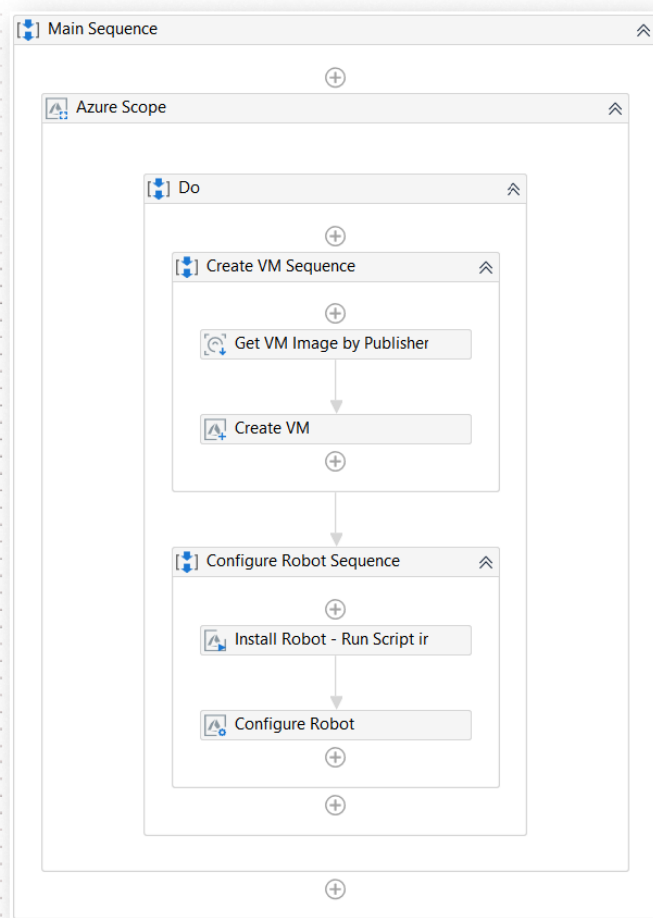


Networking & Security



500+ IT activities

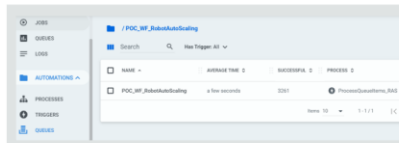
**easy to use** with drag & drop  
**robust** background execution via APIs  
**secure** Veracode certified  
**supported** by UiPath



**Simple. Elegant**  
powerful workflow automation for IT Tasks

# Components Overview

## Management Orchestrator



1+ Robots (for HA)

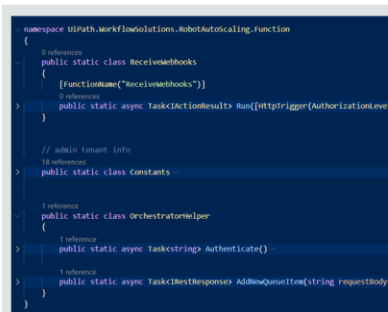
Assets // db conn, infra auth, ..

Queue // webhook events

Autoscale Process Package

Trigger // on new queue item added

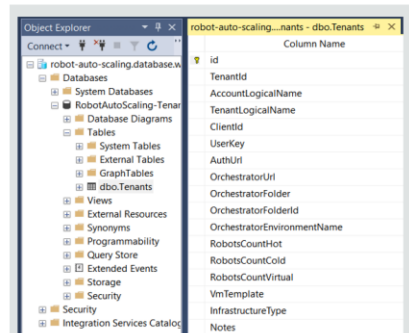
## Webhook Receiver



```
namespace UPath.werkfloodsolutions.RobotAutoScaling.Function
{
    [reference]
    public static class ReceiveWebhooks
    {
        [functionName("ReceiveWebhooks")]
        [reference]
        public static async Task<ActionResult> Run([httpTrigger(AuthorizationLevel.Anonymous, "POST")] HttpRequest req, CancellationToken ct)
        {
            // admin tenant info
            [reference]
            public static class Constants
            {
                [reference]
                public static class OrchestratorHelper
                {
                    [reference]
                    public static async Task<string> Authenticate()
                    {
                        [reference]
                        public static async Task<HttpResponseMessage> AddQueueItem(string requestBody)
                    }
                }
            }
        }
    }
}
```

Events sent via webhooks by the managed clients are added in the *Management OR. Queue*

## Clients Database



Column Name
id
TenantId
AccountLogicalName
TenantLogicalName
ClientId
UserKey
AuthUrl
OrchestratorUrl
OrchestratorFolderId
OrchestratorEnvironmentName
RobotsCountHot
RobotsCountCold
RobotsCountVirtual
VmTemplate
InfrastructureType
Notes

Orchestrator API: key, secret, tid, fid, ..  
Infra type: Azure / AWS / VMware / ..  
Scaling rules: **cold** / **hot** Robot no.

## Client Tenants

Orchestrator API	Folders (classic) & Environments	Robots (unattended)	Webhooks
------------------	----------------------------------	---------------------	----------

## Client Tenants

The managed clients need to have configured in the Webhooks area the url associated with the **Webhook Receiver's** endpoint for the events job. created / completed / faulted / stopped. These events act as triggers for the autoscaling process and they are processed in bulk, to minimize concurrency risk for servers start / stop operations.

## Database

The SQL Server database with the client Orchestrator acces data + AutoScaling Strategy robot counters can be deployed anywhere and it only contains the *Tenants* table (sql create script provided). The connection to the DB is done with the *UiPath.Database* activities – the connection string is retrieved from the *TenantsDbConnectionString* text Orchestrator asset (from the **Management Orchestrator**).

## Webhook Receiver. Webservice / Function

This component consists of minimal code that is designed to receive events, parse their data and forward them to the **Management Orchestrator** as new queue items.

## Management Orchestrator

Most of the RAS solution items are to be configured in an Orchestrator tenant, in a dedicated folder: *Processes* (ProcessClient, ..), *Assets* (DB connection, infrastructure access), *Queue + Trigger* (on new item added, if event driven), *Trigger for Scheduled Job* (if pooling). The RAS solution can be deployed in the same Orchestrator tenant as one of the managed clients, as long as it is configured in a folder that is not associated with a client from the DB.

## Process Flow

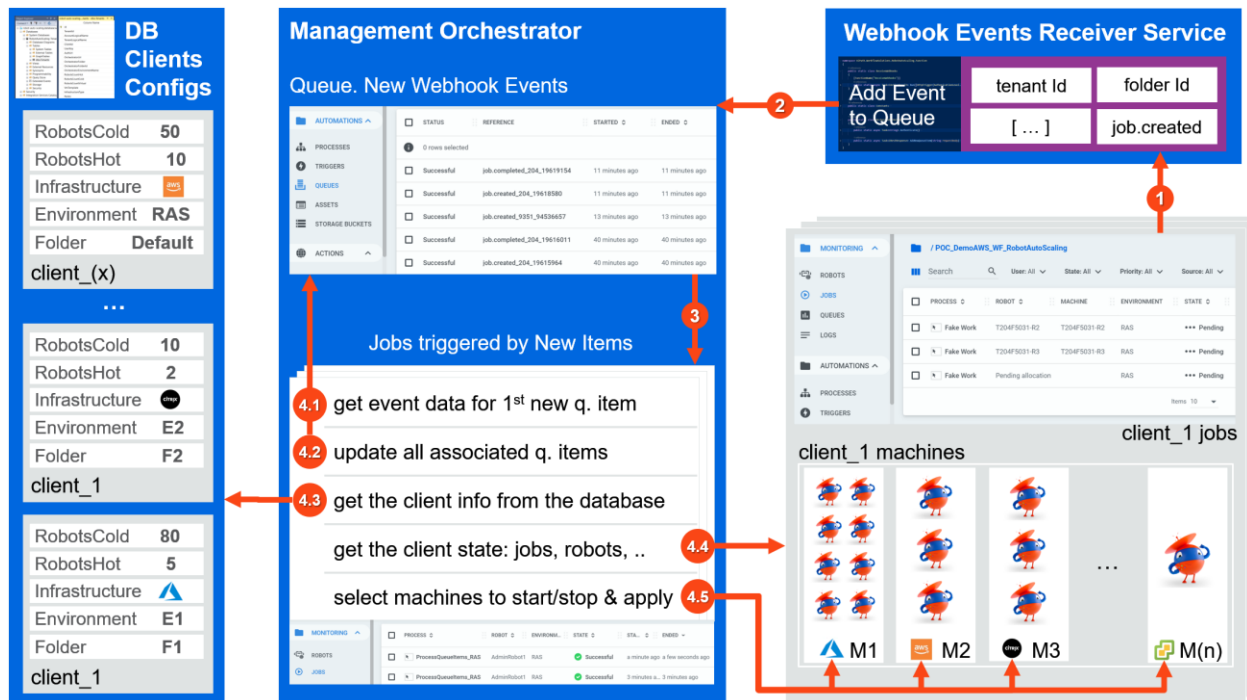
One deployment of the RAS solution can perform autoscaling for multiple clients at the same time

### 1. Event Driven

When configured in this mode, the solution leverages the Orchestrator's capability to emit job events via webhooks: the job events that are incoming from the managed clients are received by a small web-service / function and forwarded to a Queue in the Management Orchestrator for processing.

As part of the processing step, a *ProcessClient* job connects to the Database and tries to identify the client of an event in the Tenants table - if found, the information from the selected row has the client Orchestrator API access data + the scaling strategy for the client's folder or folder+environment.

The processing job connects to the client's Orchestrator and retrieves its current state (jobs running/pending, robots available/disconnected/busy). With all the data available, the job will evaluate the state + scaling strategy robot counters, selects the impacted robot machines and proceeds to apply the necessary actions (start / stop server).



### Step 1. Job events from managed clients

All the managed Clients Orchestrators emit Job events automatically for *Job pending / running / completed / stopped / faulted*, no matter the source of the job request: manual, scheduled / triggered.

### Step 2. Webhook Events Receiver Service

The webhook receiver endpoint accepts incoming Http POST requests from Clients. For each request:

- it parses the incoming job event data (*Type, TenantId, OrganizatnUnitId, EventId, Timestamp, Jobs, Release ProcessKey, ReleaseId*)
- if the event was emitted by a job associated with a process that is part of the RAS solution, it will be ignored:
  - o *TenantId* and *FolderId* match the configured RAS Management Orchestrator
  - o this scenario is possible because you can deploy the solution in a dedicated folder on a managed client, so webhooks will be sent for RAS too
- it authenticates to the RAS Management Orchestrator (works with both UiPath Cloud Tenants and On-Premises deployments)
- performs an API call to the Management Orch. - *UiPathODataSvc.AddQueueItem* - that adds the parsed event data as a *New Item* in the Management Orchestrator *RAS Queue*.

### Step 3. Trigger a Job to process new events

The *RAS Queue* has a trigger associated that starts a new Job instance (from *ProcessClient*) for each new item added.

### Step 4. Process Events

UiPath Orchestrator queue items ordering and retrieval is done according to FIFO, so the *first* new item will be retrieved by the processing job (not the *latest* new added item).

After retrieving the event information from the item, we know its *TenantId*, *FolderId* and *Release* (Process) details. With this information, we can get from the queue all the new items associated with the item's client. If the autoscaling is configured to be performed at *Environment* level (by setting the *EnvironmentLevelCounters* bool asset to *true* in the Management Orchestrator), the events will be retrieved by taking into account the *ReleaseId* and *ReleaseProcessKey* too.

All the new items associated with the Client that were added in the RAS Queue up to the moment of the processing job will be marked as processed – we do this in order to avoid concurrency issues:

- the queue's trigger (on new items) will start processing jobs for the events that we marked as completed, they will only process newer items (if any) that were added to the *RAS Queue* after the other job's execution

Having the *TenantId*, *FolderId* (and if so configured the *Environment*), we proceed to retrieving the client's information from the Database (the DB communication is done with the help of the connection string retrieved from the *TenantsDbConnectionString* asset, defined in the Management Orchestrator).

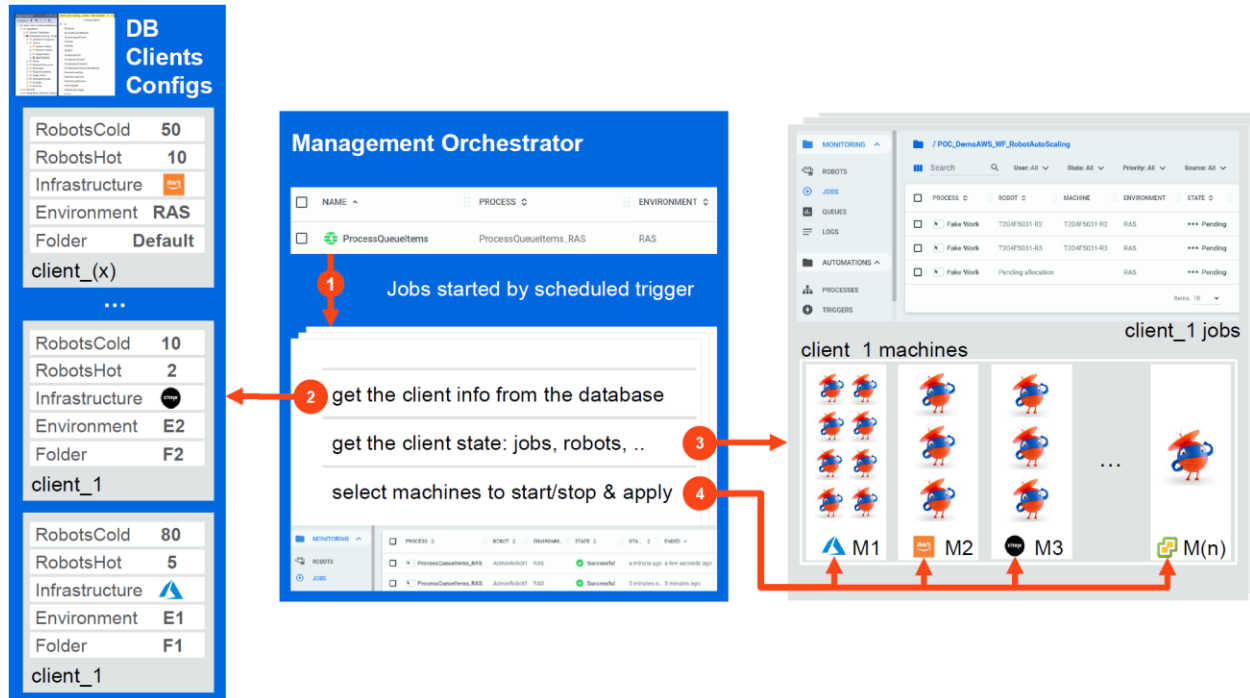
As we now have the client's API access info for its Orchestrator, Infrastructure Type (eg. *azure*, *aws*, *citrix*, *vmware*, ...) and scaling strategy (counters for *hot/cold Robots*), we can perform the autoscaling:

1. retrieve the client's state (jobs running/pending, robots available/disconnected/busy)
2. select Robot Machines to be processed (based on the client's state + hot/cold robot counters)
3. apply the computed operation (start / stop) for each of the selected machines



## 2. Pooling via Scheduled Job

When configured in this mode, the solution doesn't require any of the job events related RAS components (webhooks, events receiver service, queue): the autoscaling process is triggered periodically (scheduled start at each 5+ min), according to a specified scheduled processing job (1 for each client DB entry):



### Step 1. Trigger for Scheduled Job

Add *ProcessClient* job is started by the *Scheduled Time* trigger for each client entry from the DB Tenants table. The client is identified by the *SpecifiedClientId* parameter (it must match the client's *id* column from the DB Tenants table).

### Step 2. Process Client

The processing job doesn't go through any of the queue / events related workflow steps and directly retrieves the client's information from the Database based on the *SpecifiedClientId* workflow in-argument (it matches with the *id* column from the DB Tenants table). The DB communication is done with the help of the connection string retrieved from the *TenantsDbConnectionString* asset, defined in the Management Orchestrator).

As we now have the client's API access info for its Orchestrator, Infrastructure Type (eg. *azure*, *aws*, *citrix*, *vmware*, ..) and scaling strategy (counters for *hot/cold Robots*), we can perform the autoscaling:

1. retrieve the client's state (jobs running/pending, robots available/disconnected/busy)
2. select Robot Machines to be processed (based on the client's state + hot/cold robot counters)
3. apply the computed operation (start / stop) for each of the selected machines



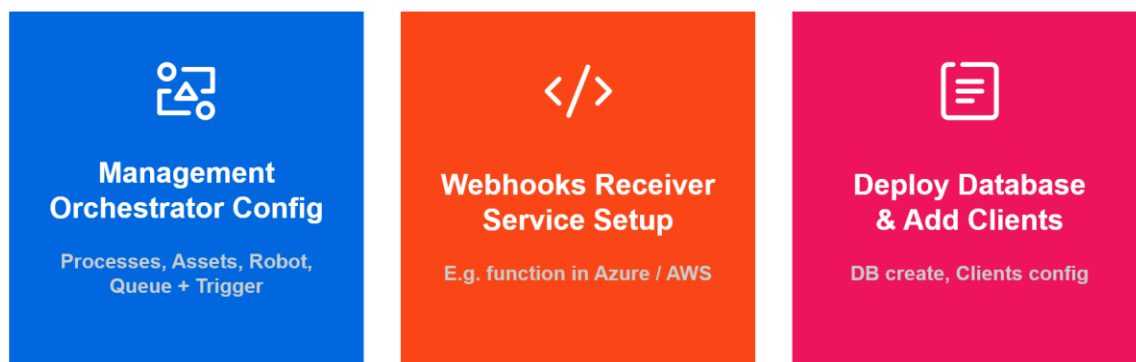
## Solution Deployment and Configuration

The Robots Auto Scaling solution is workflow based, so it runs on top of UiPath's RPA Platform. The implications of this dependency are that:

1. it can be deployed in all environments compatible with UiPath's platform
2. It can manage the autoscaling for clients deployed in on-prem environments, in private & public clouds, in UiPath's RPA Cloud



The deployment of the RAS solution for the *Event Driven* processing mode is done in 3 main steps:



If the desired processing mode is by *Pooling*, the following steps can be skipped:

- Webhooks Receiver Service Setup
- Management Orchestrator: queue + trigger configuration

## 1. Management Orchestrator Configuration

The RAS solution can be deployed in the same Orchestrator tenant as one of the managed clients, as long as it is configured in a folder that is not associated with a client from the DB Tenants table.

### Basic Setup

In the Orchestrator where the RAS solution is to be deployed you will need to have the following:

- create a new Folder for the solution (any name )
- add Robot(s)
  - o the solution's processes are background running and not resource intensive, so 1 unattended robot should be enough to handle the RAS load
  - o it is recommended to have at least 2 robots for High Availability reasons

### Add or Publish RAS Solution Packages

The RAS Solution .zip archive contain both the process .nupkg file(s) as well as the UiPath Studio workflow project files (the xaml sources). There are two options for adding the RAS processes:

1. import the .nupkg package in your Orchestrator
2. you can publish it from

After the packages are in the Management Orchestrator, create the process(es).

### Queue configuration

(skip when desired *Processing Mode = "Pooling"*)

Add a new Queue with no unique reference, without auto-retry:

Update Queue	Schema Definitions	SLA Predictions
Name * POC_WF_RobotAutoScaling	Specific Data JSON Schema <a href="#">BROWSE</a>	<input type="checkbox"/> Enable SLA for this queue
Description	Output Data JSON Schema <a href="#">BROWSE</a>	
Unique Reference <input type="radio"/> Yes <input checked="" type="radio"/> No	Analytics Data JSON Schema <a href="#">BROWSE</a>	
Auto Retry <input type="radio"/> Yes <input checked="" type="radio"/> No		

### Trigger setup

#### For Processing Mode = Event Driven

Add a new trigger for the RAS Queue:

- for each new item added, start 1 Job from the process *ProcessClient*
- trigger another job for each 1 new item(s)
- max number of parallel pending+running jobs depends on your Client Jobs load (a high value is recommended eg. 1000)

Name \*

ProcessQueueItems

Process \*

ProcessClient\_RAS

Job priority \*

→ Inherited

Parameters

Int32

SpecifiedClientId

-1

Queue

POC\_WF\_RobotAutoScaling

The queue trigger runs in the environment associated to the selected process.

Minimum number of items to trigger the first job.

1

item(s)

Maximum number of pending and running jobs allowed simultaneously.

1000

job(s)

Another job is triggered for each

1

new item(s)

Timezone \*

(UTC) Coordinated Universal Time

Non-working days restrictions

No calendar selected.

Disable Trigger at

### For Processing Mode = Pooling

Add a separate *Time trigger* for each client entry from the DB Tenants table (*SpecifiedClientId* should match the client's *id* column from the *DB Tenants* table):

Name \*

ProcessClient

Process \*

ProcessClient\_RAS

Job priority \*

→ Inherited

Execution Target

Parameters

Int32

SpecifiedClientId

66

Timezone \*

(UTC) Coordinated Universal Time

Minutes

Every 5

minute(s)

Hourly

Daily

Weekly

Monthly

Advanced

The process will be scheduled in (UTC) Coordinated Universal Time

Non-working days restrictions

No calendar selected.

Stop Job after

Disable Trigger at

### The Execution Target:

Execution Target

Parameters

All Robots

Specific Robots

Allocate dynamically

Execute the process

1

times

## Assets configuration

Add the following assets to your Management Orchestrator RAS Folder.

⚠ The values are common for all clients defined in the database  
- important to note especially for the assets *RobotMachinesNameFilter* and *RobotMachinesToKeepAlive*

### Core Assets

#### TenantsDbConnectionString (Text)

- used to connect to the RAS Clients Database

#### AdminTenantEventsQueueName (Text)

- name of the RAS client events queue

#### EnvironmentLevelCounters (Bool)

If value is *False*:

- the autoscaling strategy (hot/cold robot counters) will be applied at tenant + folder level

If value is *True*:

- the autoscaling strategy (hot/cold robot counters) will be applied at tenant + folder + environment too
- you can have multiple entries for a client (same TenantId + FolderId and different Environment name) that differ when it comes to the Environment, hot/cold Robot counts, infrastructure type

#### RobotsType (Text)

Value (one of the following): *Unattended*, *NonProduction*, *Attended*, *Unattended*, *Studio*, *Development*, *StudioX*, *Headless*, *StudioPro*, *TestAutomation*

#### RobotMachinesNameFilter (Text)

Only process (apply autoscaling) to the robot machines that contain the asset's value in their name.  
For the default value "-", no name filtering will be done.

#### RobotMachinesToKeepAlive (Text)

CSV (separator char is ",") with machines that will be skipped by the scale-down (stop) part of the auto-scaling process. For the default value "-", all machines are taken into account by the solution.

#### VmRobotCredentials (Credential)

Used by the *AddClient* process:

- when creating the Robots in Orchestrator (setting the robot's *username* and *password* )
- for provisioning the actual robot machine servers the infrastructure (Azure, VMware, ..)
  - o the workflow creates the specified account and adds it to the local Admins group

No *domain* prefix is needed for the *username* – the workflow adds it dynamically based on the robot's associated machine.

### Infrastructure Access. Azure

#### Azure\_VmImages\_RG (Text)

Name of the Azure resource group where the VM image for virtual robots (*upcoming feature*) and clients onboarding workflow (*AddClient process – upcoming feature*) is located

**AzureAuth\_ClientID (Credential)**

Add to the asset's *Value* the *ClientID* of the *AppRegistration* used to connect to Azure.

Documentation: [UiPath Azure Scope activity](#)

**AzureAuth\_ClientSecret (Credential)**

Add to the asset *Value* the *ClientSecret* of the *AppRegistration* used to connect to Azure.

**AzureAuth\_SubscriptionID (Text)**

Azure Subscription ID where the resource group(s) from the *Azure\_VMs\_RG* asset are located.

**AzureAuth\_TenantID (Credential)**

Azure Tenant ID associated with the *AppRegistration*.

**Azure\_VMs\_RG (Text)**

Name of the Azure resource group where the VMs associated with the robot machines are located (for multiple RGs, use a csv value – eg: *RG1, RG2*)

*Infrastructure Access. Amazon Web Services***AmazonAWS\_AccessKeyID (Credential)**

The access key used to connect to Amazon Web Services.

Documentation: [UiPath AWS Scope activity](#)

**AmazonAWS\_SecretAccessKey (Credential)**

The secret key used for connecting to Amazon Web Services.

**AmazonAWS\_Region (Text)**

The AWS Region to connect to.

**AmazonAWS\_InstanceKeyPairName**

The AWS Instance KeyPairName associated with the *VmTemplate* for the onboarding workflow (*AddClient process – upcoming feature*).

*Infrastructure Access. VMware ESXi***VMwareAuth\_Server (Text)**

The address of the VMware vCenter server you want to connect to.

Documentation: [UiPath VMware Scope activity](#)

**VMwareAuth\_Credentials (Credential)**

Username: the username used to connect to the VMware server

Password: the password used to connect to the VMware server

**VMware\_VMs\_Location (Text)**

Folder full path where the robot machines are located. Eg. value: *Datacenter/FolderA/SubFolder1*

**VMware\_Host (Text)**

The address of the VMware vCenter server you want to connect to for provisioning machines for virtual robots or for the client onboarding (*AddClient* process).

**VMware\_VmTemplateAuth (Credential)**

Credentials used for running a script in newly provisioned machines for changing the computer name.

**VMware\_VmTemplateLocation (Text)**

Folder full path where the VM Template for new robot machines is located. Eg. value:

*Datacenter/FolderA/SubFolder1*

**VMware\_Datastore (Text)**

Datastore for the new robot machines.

*Infrastructure Access. Citrix Hypervisor*

**CitrixAuth\_Host (Text)**

The host name or IP address of the Citrix Hypervisor Server.

Documentation: [UiPath Citrix Scope activity](#)

**CitrixAuth\_HostPort (Integer)**

Citrix host port for the connection (default value is 80).

**CitrixAuth\_Credentials (Credential)**

Username: the username used to connect to the Citrix server

Password: the password used to connect to the Citrix server

**Citrix\_VMs\_Folder (Text)**

Folder full path where the robot machines are located. Eg. value: *Folder1*

---

**Notes:**

Not all infrastructure related assets need to be added: the ones that don't apply to your managed clients can be skipped (eg. don't add the Azure, VMware, Citrix assets if all your robot machines are all in AWS).

## 2. Webhooks Receiver Setup

The C# code for the Webhooks Receiver is available in the *ReceiverService.cs* file from the RAS Solution .zip archive. It was designed to be portable and easy to configure:

- it can be easily published in a cloud function (Azure, AWS, ..)
- it could also be hosted in a small web-service that can be published on a dedicated server

We need to assign values for a few constants:

```
public static class Constants
{
    // TRUE - on-prem constants will be used for the Orchestrator requests to add webhook event as new Queue Item
    // FALSE - cloud api constants will be used for the Orchestrator requests to add webhook event as new Queue Item
    internal static readonly bool AdminOrchestratorIsOnPrem = false;

    #region common
    internal static readonly string AdminOrganizationUnitId = "";
    internal static readonly string AdminTenantId = "";
    internal static readonly string AdminEventsQueueName = "";
    #endregion

    #region on-prem
    internal static readonly string AdminOnpremTenantName = "";
    internal static readonly string AdminOnpremTenantUsername = "";
    internal static readonly string AdminOnpremTenantPassword = "";
    internal static readonly string AdminOnpremTenantAuthUrl = "https://YOUR-ORCHESTRATOR-AUTH-URL/api/Account/Authenticate";
    internal static readonly string AdminOnpremTenantUrl = "https://YOUR-ORCHESTRATOR-AUTH-URL";
    #endregion

    #region cloud api
    internal static readonly string AdminAccountLogicalName = "";
    internal static readonly string AdminTenantLogicalName = "";
    internal static readonly string AdminClientId = "";
    internal static readonly string AdminUserKey = "";
    internal static readonly string AdminAuthUrl = "https://account.uipath.com/oauth/token";
    internal static readonly string AdminTenantUrl = $"https://platform.uipath.com/{AdminAccountLogicalName}/{AdminTenantLogicalName}";
    #endregion
}
```

**AdminEventsQueueName** – name of the Management Orchestrator RAS Queue

**AdminOrganizationUnitId** – the Management Orchestrator RAS Folder Id

**AdminOrganizationTenantId** – the Management Orchestrator Tenant Id



**AdminOrchestratorIsOnPrem**

True: auth token is retrieved based on user & password

False: auth token is retrieved via the cloud api, based on clientId & userKey

documentation: [how to get the UiPath Cloud API access info](#)

---

After publishing the W.H. Receiver, save the the url pointing to the request processing function/method (it is needed for the Client's Orchestrator Webhook URL). Example url from publishing the code as an Azure function: <https://robotautoscaling.azurewebsites.net/api/ReceiveWebhooks>



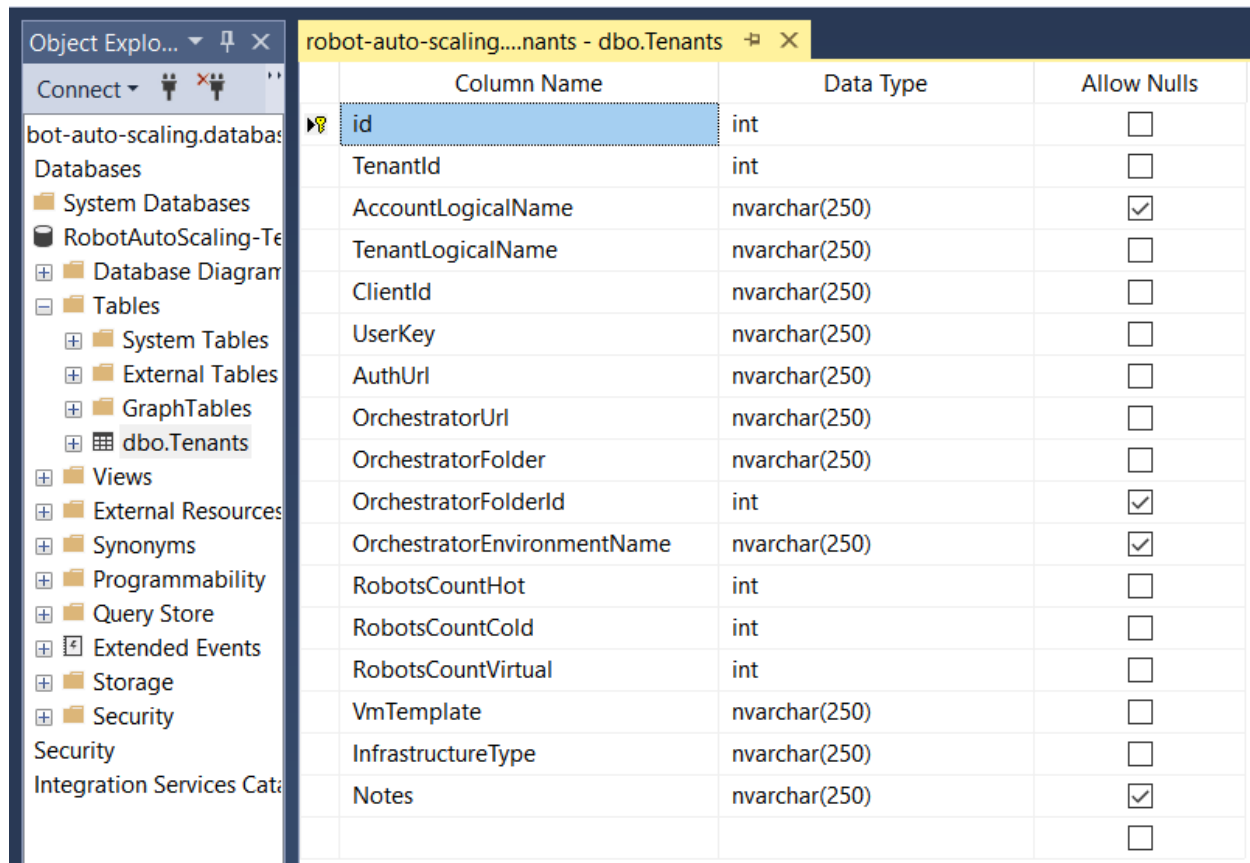
### 3. Database Deployment. Clients Configuration

#### Database Setup

Create a Database for on a SQL Server and run the *DbTenantsTable.sql* script(file provided in the RAS solution .zip archive) to create the *Tenants* table.

Save the DB Connection String in the *TenantsDbConnectionString* asset. Example value:

*Server=tcp: YOUR-SERVER,1433; Initial Catalog=YOUR-DATABASE; Persist Security Info=False; User ID=YOUR-USER; Password=YOUR-PASSWORD; MultipleActiveResultSets=False; Encrypt=True; TrustServerCertificate=False; Connection Timeout=30;*



Column Name	Data Type	Allow Nulls
id	int	<input type="checkbox"/>
TenantId	int	<input type="checkbox"/>
AccountLogicalName	nvarchar(250)	<input checked="" type="checkbox"/>
TenantLogicalName	nvarchar(250)	<input type="checkbox"/>
ClientId	nvarchar(250)	<input type="checkbox"/>
UserKey	nvarchar(250)	<input type="checkbox"/>
AuthUrl	nvarchar(250)	<input type="checkbox"/>
OrchestratorUrl	nvarchar(250)	<input type="checkbox"/>
OrchestratorFolder	nvarchar(250)	<input type="checkbox"/>
OrchestratorFolderId	int	<input checked="" type="checkbox"/>
OrchestratorEnvironmentName	nvarchar(250)	<input checked="" type="checkbox"/>
RobotsCountHot	int	<input type="checkbox"/>
RobotsCountCold	int	<input type="checkbox"/>
RobotsCountVirtual	int	<input type="checkbox"/>
VmTemplate	nvarchar(250)	<input type="checkbox"/>
InfrastructureType	nvarchar(250)	<input type="checkbox"/>
Notes	nvarchar(250)	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

#### Add managed clients

*Eg. On-Prem*

- TenantId: 1
- AccountLogicalName: NULL
- TenantLogicalName: RAS1
- ClientId: MyUsername
- UserKey: MyPassword
- AuthUrl: <https://MyOrchestratorUrl/api/Account/Authenticate>
- OrchestratorUrl: <https://MyOrchestratorUrl/>
- OrchestratorFolder: Default

- OrchestratorFolderId: 1
- OrchestratorEnvironmentName: MyEnvironment
- RobotCountHot: 10
- RobotCountCold: 100
- RobotCountVirtual: 0
- VmTemplate: -
- InfrastructureType: azure
- Notes: example values for on-prem client

*Eg. UiPath Cloud*

- TenantId: 123
- AccountLogicalName: MyService
- TenantLogicalName: ASxaCebhya
- ClientId: 8AeTQ4gWCTasW3y4ljtf62EE4c212PORn5
- UserKey: SF45HY\_e5mWJloe4DDFEREfwe3d343gFEgdfvV
- AuthUrl: <https://account.uipath.com/oauth/token>
- OrchestratorUrl: <https://platform.uipath.com/>
- OrchestratorFolder: Default
- OrchestratorFolderId: 1234
- OrchestratorEnvironmentName: MyEnvironment1
- RobotCountHot: 10
- RobotCountCold: 100
- RobotCountVirtual: 0
- VmTemplate: -
- InfrastructureType: azure
- Notes: example values for uipath cloud rpa client

*Adding multiple entries for the same tenant*

One client tenant can have multiple entries in the Tenants table, but for different folders. If the *EnvironmentLevelCounters* asset value is set to *True*, a client can have multiple entries with the same folder, but for different Environments.

For each client entry you can define both a different infrastructure type (azure, aws, citrix, vmware) and a different scaling strategy (the values for the *RobotsCountHot* and *RobotsCountCold*). Examples:

- *EnvironmentLevelCounters* = False
  - o client\_1, folder\_1, environment\_1, 10 hot robots, 100 cold robots, azure
  - o client\_1, folder\_2, environment\_2, 20 hot robots, 50 cold robots, aws
  - o client\_2, folderA, environmentA, 5 hot robots, 50 cold robots, vmware
- *EnvironmentLevelCounters* = True
  - o client\_1, folder\_1, environment\_1, 10 hot robots, 100 cold robots, azure
  - o client\_1, folder\_2, environment\_2, 20 hot robots, 50 cold robots, aws
  - o client\_2, folderA, environmentA, 5 hot robots, 50 cold robots, vmware

## Scaling Strategies

The hot / cold robot counters empower us to define strategies that can help us balance the efficiency provided by the RAS solution between:

### 1. High Availability for Pending Jobs

#### Pluses:

- a higher number of hot Robots will be able to respond faster to pending job requests
- the time between the job being created and it executing is minimal (seconds)

#### Minuses:

- even when there are no pending jobs, the machines associated with the hot robots will be running and that can increase the operational costs with your RPA deployment




### 2. Cost Efficiency

#### Pluses:

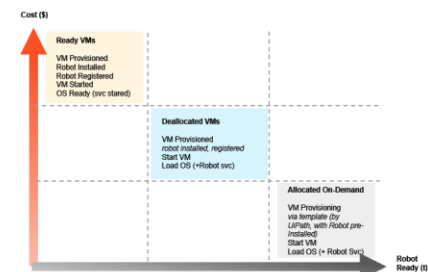
- a higher number of cold Robots will save us costs by allocating machines based on the number of created Jobs and deallocating them when there are 0 pending

#### Minuses:

- the time between the job being created and it executing on a cold Robot is in the range of ~3 minutes (varies based on the VM's configuration)

Robot Type	VM / Server state	T(min) to Ready	Cost (\$) VM / month
 Hot	<b>Ready VMs</b> VM is provisioned and running, robot configured and available for Jobs	0	<b>High \$160+</b> compute allocation
 Cold	<b>Deallocated VMs</b> VM is created / exists + robot configured, but it is deallocated (only storage costs)	~2-3	<b>Low \$1.54</b> HDD standard S4 <b>Medium \$2.4</b> SSD standard E4
 Virtual	<b>Allocated On-Demand VMs</b> VM will be created on-demand from specified image + robot configuration	~10	<b>\$0</b>

\* Azure (pay-as-you-go) ref. VM: D2v3 + 32Gb Standard S4 hdd / E4 ssd



**Scaling Strategy = F(x,y,..)**

Pending Jobs

Robots Available  
Robots Disconnected  
Robots Busy

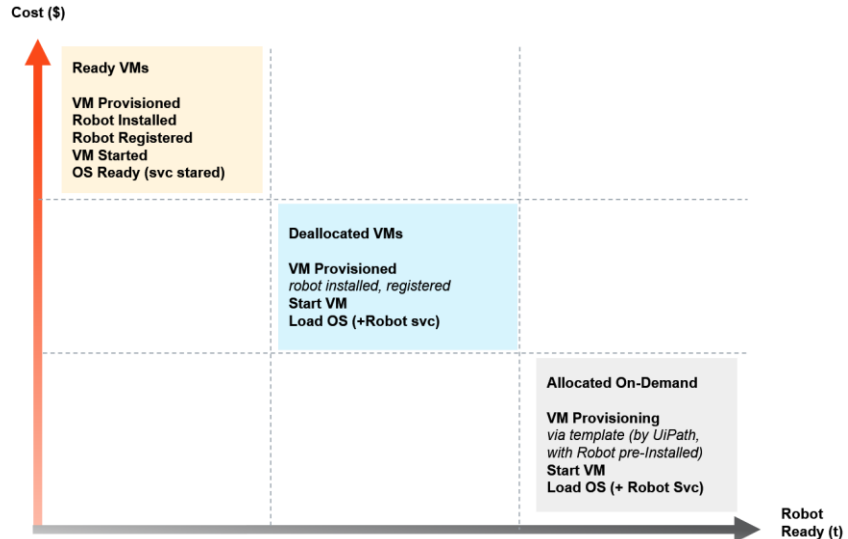
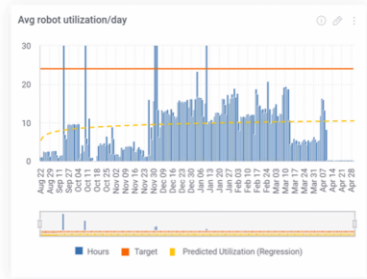
Hot Robots  
Cold Robots  
Virtual Robots

As illustrated in the above table, the efficiency delivered by the RAS solution can contribute significantly to cost savings (especially if the robot machines type/size is large). The solution's impact scales with the size of your operation (number of robots and autoscaled servers).

When deciding the value for the hot Robots counter in RPA deployments with high density robots, it is recommended to analyze the costs of having more smaller machines with 1 robot vs 1 larger machine with 2+ robots.

**Couple with Robots usage**  
eg. Insights / analyze Job requests

Adjust **hot** / **cold** Robots variables for the identified intervals.



The fact that the hot / cold robot counters that define the scaling strategy are retrieved from the Database by the processing job with each execution enables us to very easily and dynamically adjust the autoscaling logic by editing their values.

If there is historical data available that reflects the client's jobs/robots load over time (eg. from UiPath Insights), it can be leveraged to further increase the RAS solution's impact by adjusting the hot / cold Robot counters to anticipate both increases and decreases in requests.

Example:

- client\_1 analyzes the insights and discovers that there are requests spikes at the end of each month: the data suggests that by having 50 hot Robots and 100 cold robots configured in the last week of each month would handle the load, while for the rest of the month a configuration of 10 hot Robots and 140 cold Robots would be optimal
- solution:
  - o a job is scheduled to be trigger monthly (7 days before the end of the month) to edit the DB entry for client\_1: *RobotsCountHot=50, RobotsCountCold=100*
  - o a job is scheduled to be trigger monthly (last day of the month) to edit the DB entry for client\_1: *RobotsCountHot=10, RobotsCountCold=140*

---

## Notes. Performing maintenance on autoscaled robot machines

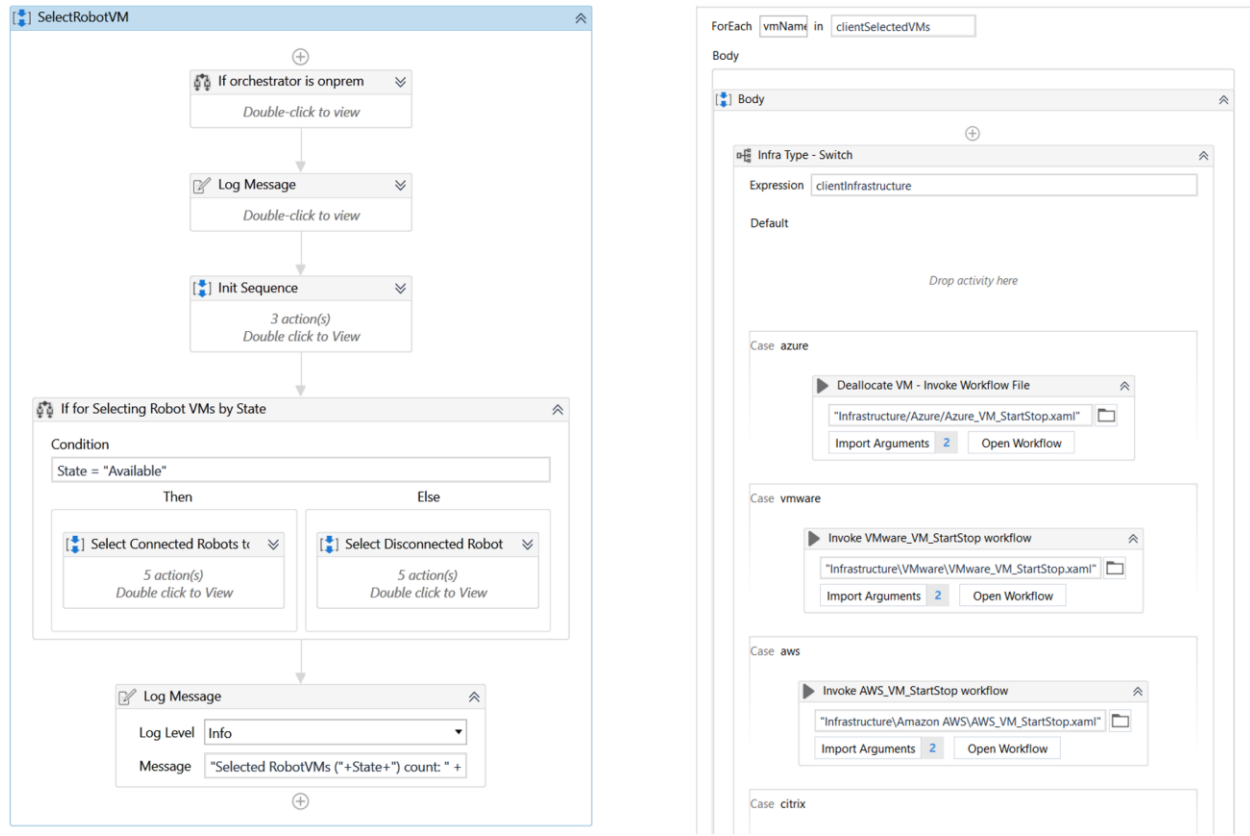
When it is needed to perform maintenance for a machine, we have a few options:

- if the *EnvironmentLevelCounters* asset's value is *True*, we can move its associated autoscaled robots to a separate Orchestrator environment
- move the machine in a different location (eg in Azure to a different RG, in VMware or Citrix to a different folder)
- move the robots to another Orchestrator folder

## Solution Customization

The Robots Auto Scaling solution is workflow based and all the project's sources (workflow xaml files) are provided in the .zip archive.

The solution can be modified to accommodate any customization needs, from different scaling logic based on the available client state related variables to adjustments for the infrastructure workflows that must align with internal IT policies (eg. Citrix VMs to be retrieved by tag, not by folder).



### Notes

There is a known VB Expression validation bug in the current Studio version (a fix is in progress) that can prevent the project from publishing: **'JObject' is ambiguous in the namespace 'Newtonsoft.Json.Linq'**. At runtime there are no problems (the robots run the process jobs without issues). There are 2 workarounds:

1. If you don't use Citrix Hypervisor infrastructures, you can simply exclude the *Infrastructure\Citrix\_VM\_StartStop.xaml* workflow from the project
2. Publish with the following steps:
  - a. Close Studio and delete all the files from *%userprofile%\AppData\Local\UiPath\cache*
  - b. Open the project and click on Publish asap (while the cache is being rebuilt)

# Release Notes

## Version 1.5

- added [Pooling mode](#) (alternative for the Event Driven processing)
  - o renamed *ProcessQueueItems* to *ProcessClient*
- improvements to reduce risk of race conditions
- bugfixes

## Version 1.0

- Autoscaling for [High Density Robots](#)
- Folder types: [classic folders \(+environments\)](#)
- Infrastructure types: [Azure](#), [AWS](#), [VMware](#), [Citrix](#)
- Scaling: [RobotsCountHot](#), [RobotsCountCold](#)
- Processes
  - o [ProcessQueueItems](#) – autoscales existing RPA deployments

## Upcoming Releases

- [AddClient](#) - new process for client on-boarding
  - o automates the RAS onboarding of a client - end to end provisioning of VMs + orchestrator setup (creates machines, robots, environment, adds webhook)
- Folder types: [support for Modern Folders](#)
- Infrastructure types: [Hyper-V](#), [Google Cloud](#)
- Scaling: [RobotsCountVirtual](#)