Robot Auto Scaling
powered by UiPath IT Automation

Documentation: Deployment / Configuration / User-Guide

Release Version: 1.6

Publish Date: 2020.12.23

Author: andrei.oros@uipath.com

Table of Contents

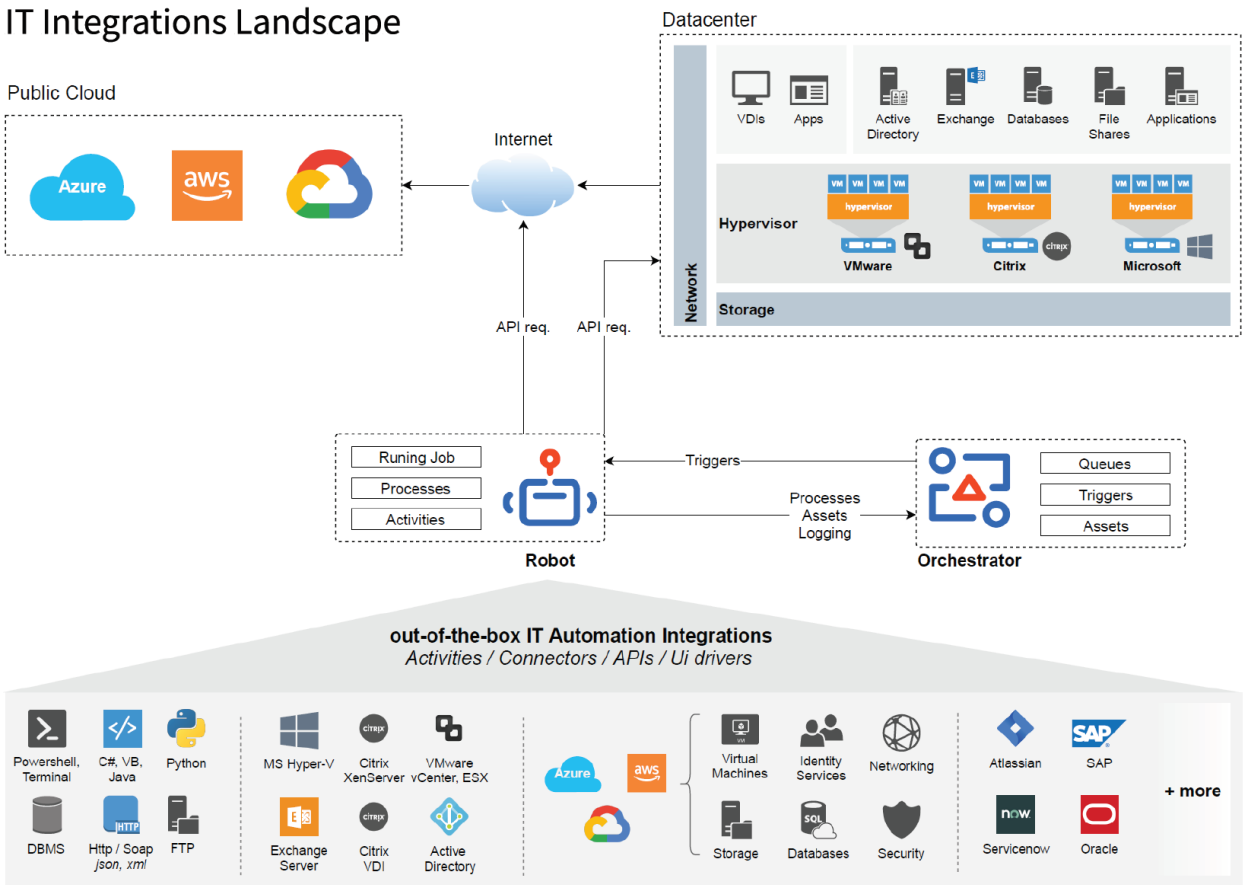
Introduction

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:



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.

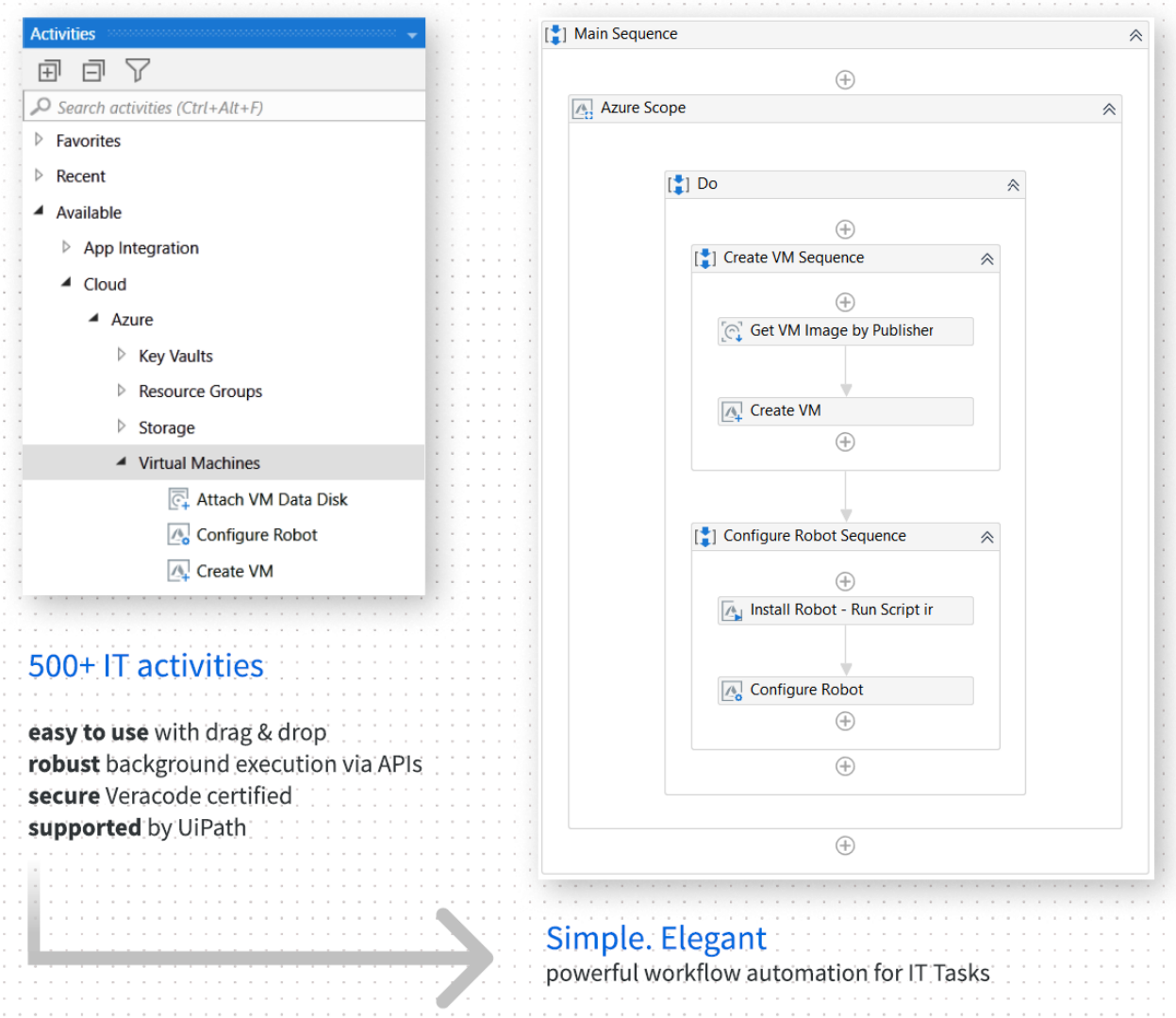
Official Activities:
Azure, Amazon Web Services (AWS), Active Directory, Azure Active Directory, VMware vCenter ESXi, Citrix Hypervisor, Hyper-V, Exchange Server, Microsoft System Center
.. and more.

Server Virtualization

Cloud Infrastructure & Services

User Management

Networking & Security



500+ IT activities

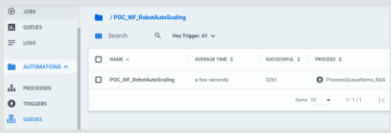
easy to use with drag & drop
robust background execution via APIs
secure Veracode certified
supported by UiPath

Simple. Elegant

powerful workflow automation for IT Tasks

Components Overview

Management Orchestrator



1+ Robots (for HA)

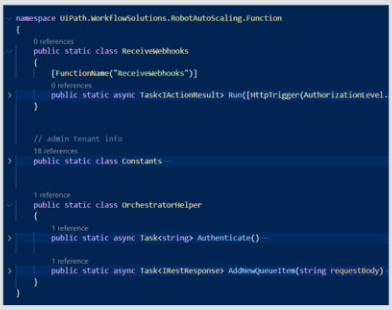
Assets *// db conn, infra auth, ..*

Queue *// webhook events*

Autoscale Process Package

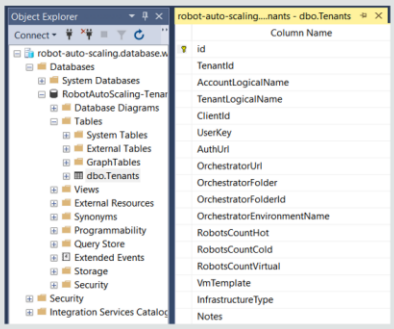
Trigger *// on new queue item added*

Webhook Receiver



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

Clients Database



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 associeted with a client from the DB.

Process Flow

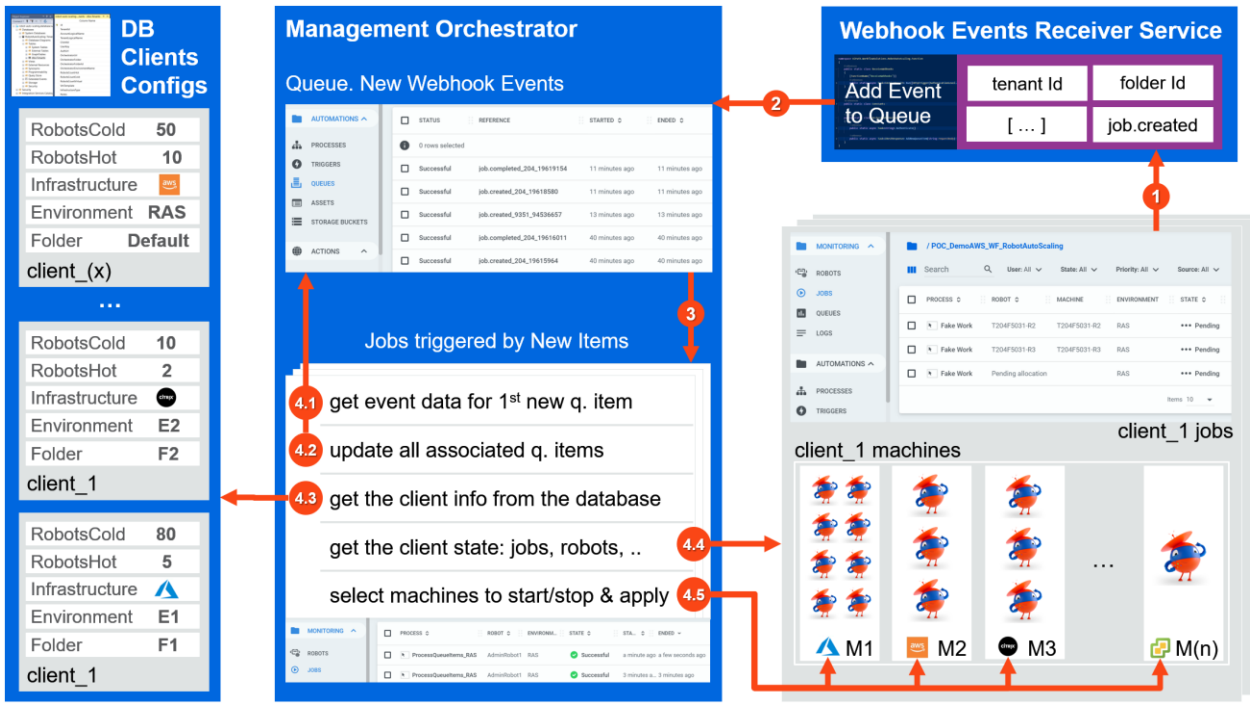
One deployment of the RAS solution can perform autoscaling for multiple clients at the same time

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 identity the client of an event in the Tenants table - if found, the information from the selected row has the client Orchestrator API acces 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 neccesary 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:
    - TenantId and FolderId match the configured RAS Management Orchestrator
    - 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 Queu*e after the other job's execution

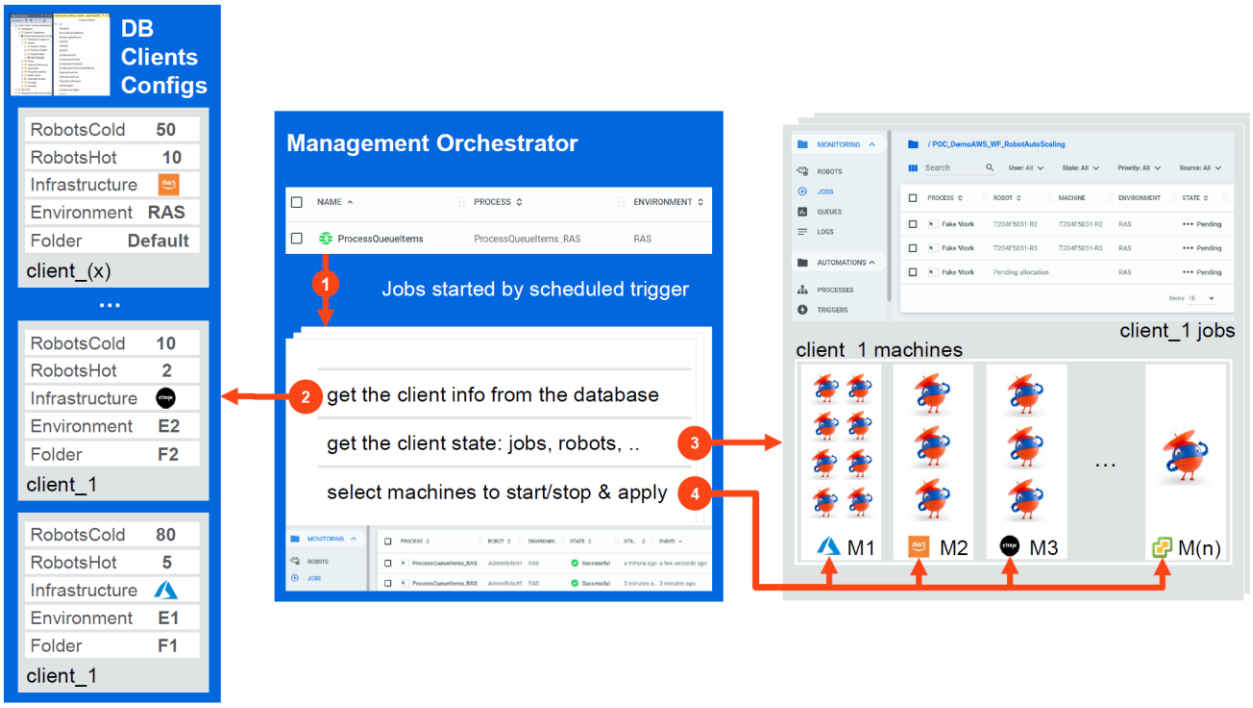
Having the *TenantId, FolderId* (and if so configured the *Environment*), we proceed to retrieving the client's information from the Database (the DB communication is done with the help of the connection string retrieved from the *TenantsDbConnectionString* asset, defined in the Management Orchestrator).

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 associeted 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)
    - the solution's processes are background running and not resource intensive, so 1 unattended robot should be enough to handle the RAS load
    - 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 to options for adding the RAS processes:

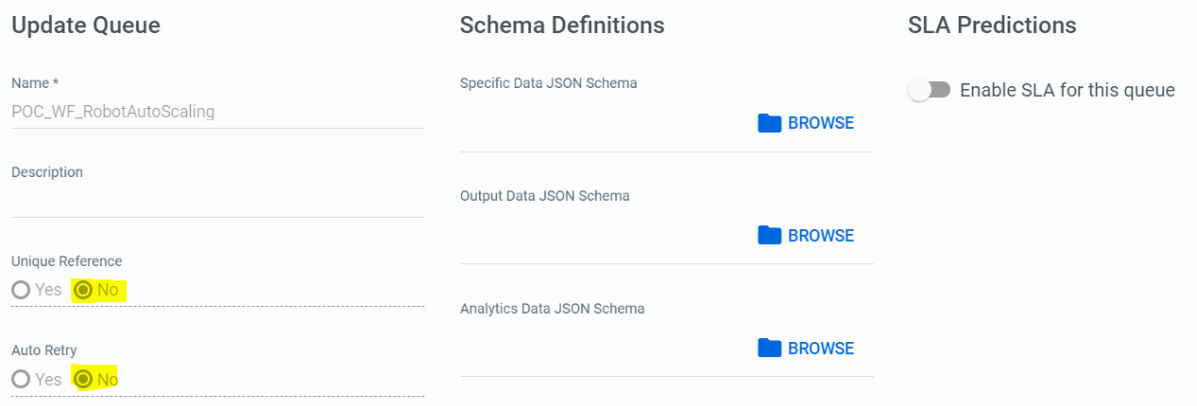
1. import the .nupkg package in your Orchestrator
2. you can publish it from

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:

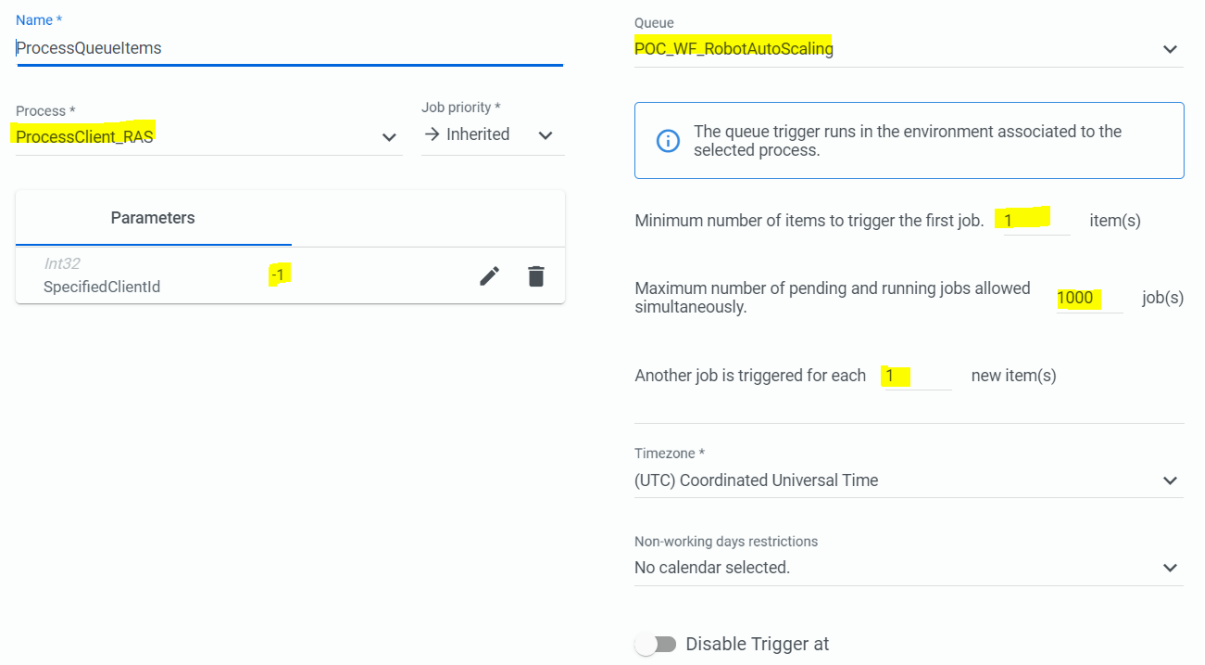


Trigger setup

For Processing Mode = Event Driven

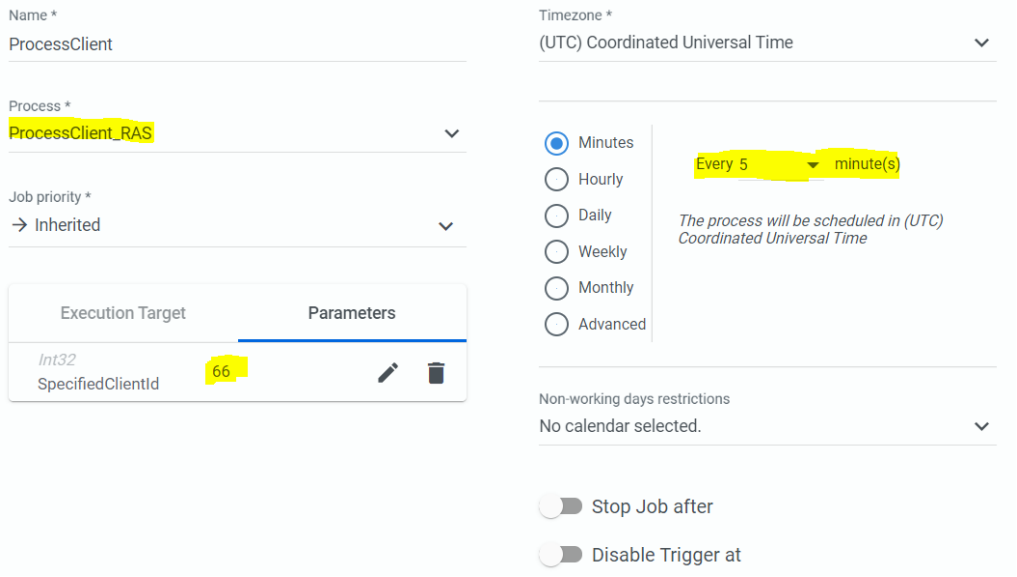
Add a new trigger for the RAS Queue:

- 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)

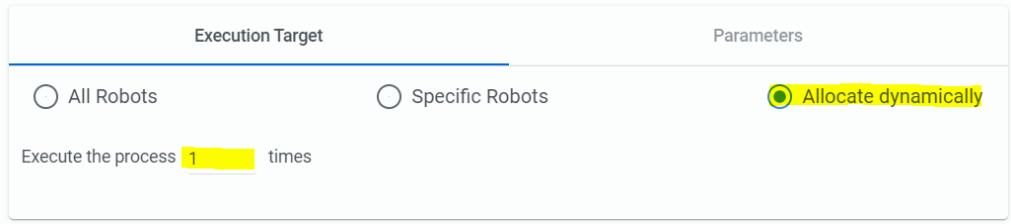


For Processing Mode = Pooling

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):



The Execution Target:



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

Common_RobotStatusToggle (Bool)

(only works with 20.10+) If set to "True", the autoscaling process will disconnect Robots associated with Machines selected for deallocation and Re-enables them back after the Machines shutdown is performed.

Common_TemporaryEnvironmentName (Text)

Moves Robots associated with Machines selected for deallocation to the specified Temporary Environment and restores them to their initial environment after the Machines shutdown is performed. For the default value "-", robots will remain in their current environment through the entire auto-scaling process.

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 + envirnoment 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, ..)
    - 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 Webkooks Receiver is available in the *ReceiverService.cs* file from the RAS Solution .zip archive. It was designed to be portable and easy to configure:

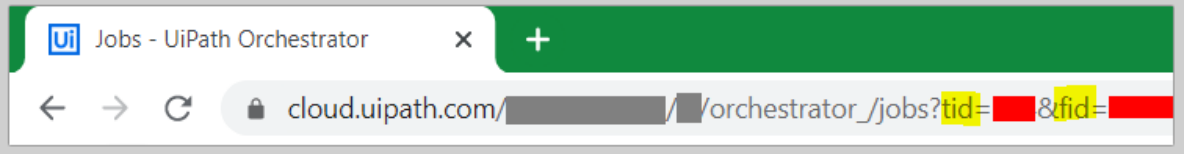
- it can be easily published in a cloud function (Azure, AWS, ..)
- it could also be hosted in a small web-service that can be published on a dedicated server

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]

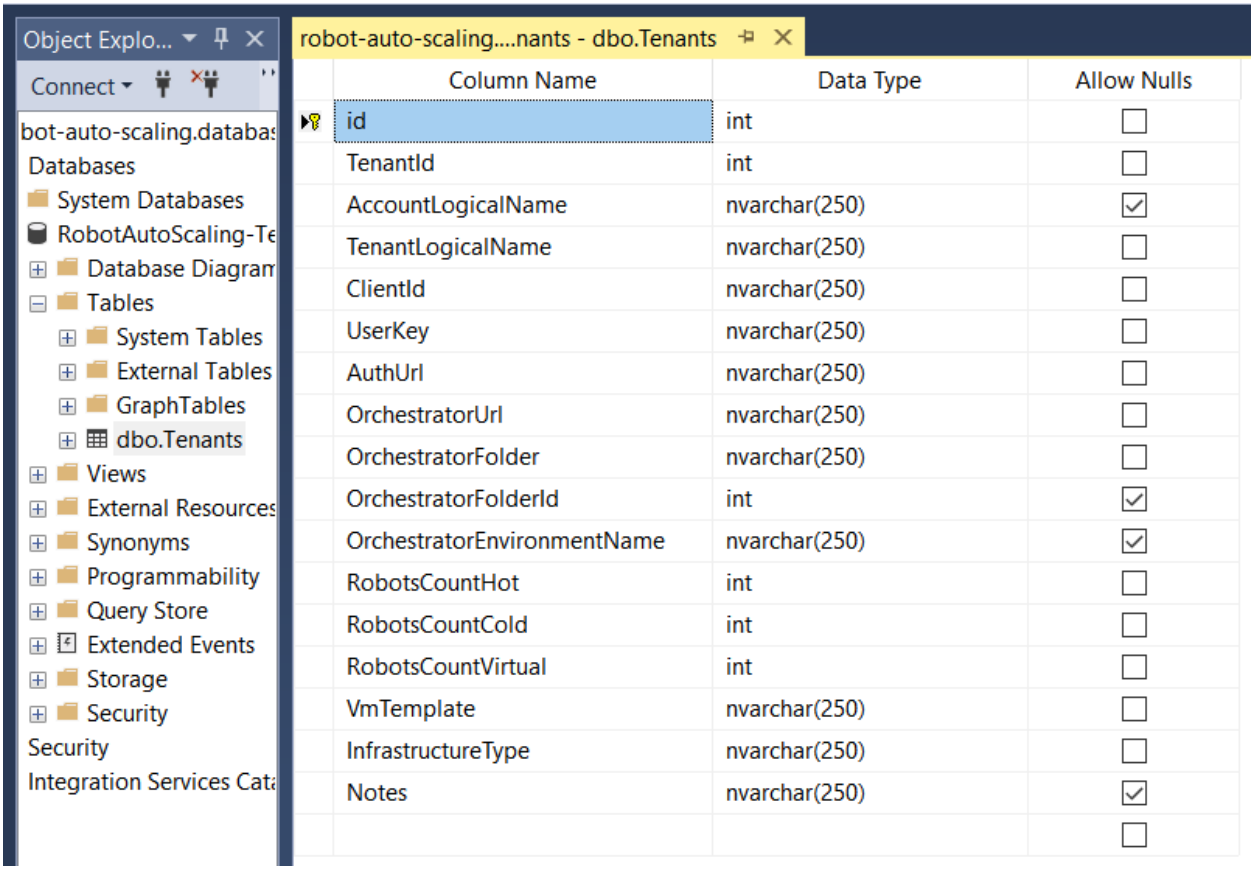
After publishing the W.H. Receiver, save the the url pointing to the request processing function/method (it is needed for the Client's Orchestrator Webhook URL). Example url from publishing the code as an Azure function: *https://robotautoscaling.azurewebsites.net/api/ReceiveWebhooks*

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;*



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_e5mWJloe4DDFEREfwe3d343gFEgdvfV
- 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*
  - client_1, folder_1, environment_1, 10 hot robots, 100 cold robots, azure
  - client_1, folder_2, environment_2, 20 hot robots, 50 cold robots, aws
  - client_2, folderA, environmentA, 5 hot robots, 50 cold robots, vmware
- *EnvironmentLevelCounters = True*
  - client_1, folder_1, environment_1, 10 hot robots, 100 cold robots, azure
  - client_1, folder_2, environment_2, 20 hot robots, 50 cold robots, aws
  - 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:

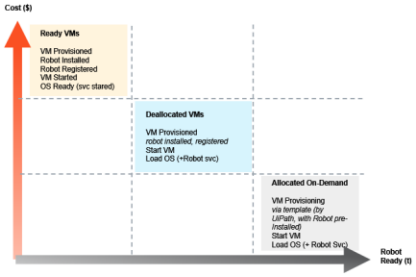
- a higher number of cold Robots will save us costs by allocating machines based on the number of created Jobs and deallocating them when there are 0 pending

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	**Ready VMs** VM is provisioned and running, robot configured and available for Jobs	**0**	**High** $160+ compute allocation
Cold	**Deallocated VMs** VM is created / exists + robot configured, but it is deallocated (only storage costs)	**~2-3**	**Low** $1.54 HDD standard S4 **Medium** $2.4 SSD standard E4
Virtual	**Allocated On-Demand VMs** VM will be created on-demand from specified image + robot configuration	**~10**	**$0**

* 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

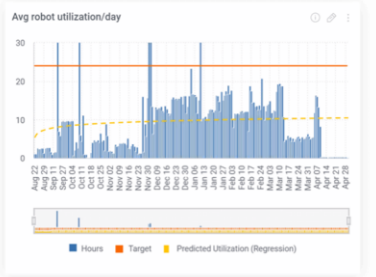
Hot Robots
Cold Robots
Virtual Robots

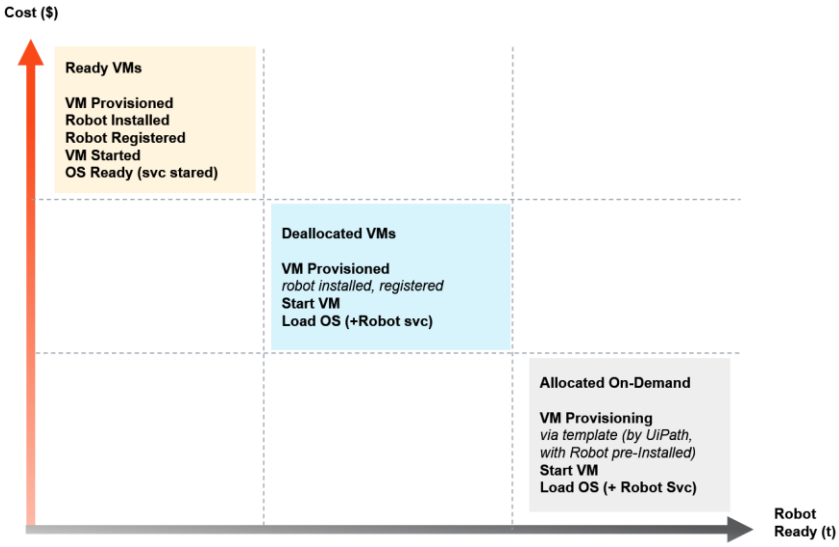
As illustrated in the above table, the efficiency delivered by the RAS solution can contribute significantly to cost savings (especially if the robot machines type/size is large). The solution's impact scales with the size of your operation (number of robots and autoscaled servers).

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:
    - 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*
    - 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

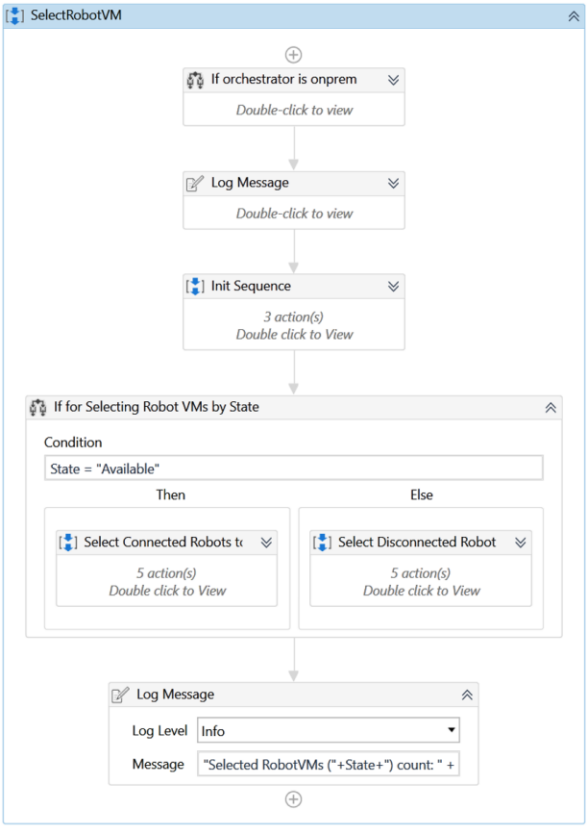
When it is needed to perform maintenance for a machine, we have a few options:

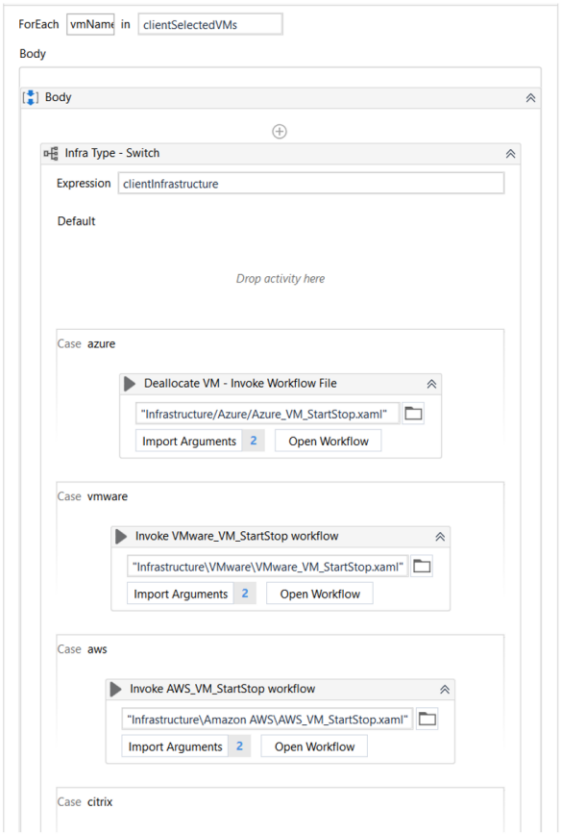
- if the *EnvironmentLevelCounters* asset's value is *True*, we can move its associated autoscaled robots to a separate Orchestrator environment
- move the machine in a different location (eg in Azure to a different RG, in VMware or Citrix to a different folder)
- move the robots to another Orchestrator 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 1.6

- added option to make the robots selected for deallocation instantly unavailable at Orchestrator level; works with 20.10+ (see asset *RobotStatusToggle*) as well as older versions (see asset *Common_TemporaryEnvironmentName*)
- improvements & bugfixes

Version 1.5

- added Pooling mode (alternative for the Event Driven processing)
    - 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
    - ProcessQueueItems – autoscales existing RPA deployments

Upcoming Releases

- AddClient - new process for client on-boarding
    - 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