



The bridge to possible

Intersight Cloud Orchestrator

Deep Dive Deck

ICO TMEs

Cloud and Compute

About This Document

This document aims to provide a detailed insight of Intersight Cloud Orchestrator (ICO) capabilities through examples.

Note: some screenshots may change over time due to the nature of Intersight, please refer to the documentation on <https://intersight.com/help>

Table of Content

[Sample Scenario](#)

[Custom Data Types](#)

[Task Designer - Web API Executor and Outcomes](#)

[Workflow Designer](#)

[Validate and Execute a Workflow](#)

[Use Task Inputs](#)

[Conditional Task](#)

[Parallel Loops](#)

[Workflow Variables](#)

[Serial Loops](#)

[Workflow Versioning](#)

[Transformations](#)

[Rollback Tasks](#)

[Create a Notification Task with Webex](#)

[Powershell Executor](#)

[SSH Executor](#)

[Ansible Executor](#)

[Go Templates Cheat Sheet \(Advanced Mapping\)](#)

[ICO Intersight API](#)

[Execute an ICO Workflow from Terraform](#)

[Execute an ICO Workflow with the Python SDK](#)

[Execute an ICO Workflow with Powershell SDK](#)

Sample Scenario

Example workflow

Goal:

- Fetch current weather in a given city
- Return a structured workflow output

How:

- Openweathermap.org
- Build and map Custom Data Types (CDTs)
- Use the Task Designer and the REST API executor and map CDTs
- Create a workflow that uses the CDT as input and the custom task

Call current weather data for one location

By city name

You can call by city name or city name, state code and country code. Please note that searching by states available only for the USA locations.

API call

```
api.openweathermap.org/data/2.5/weather?q={city name}&appid={API key}
```

```
api.openweathermap.org/data/2.5/weather?q={city name},{state code}&appid={API key}
```

```
api.openweathermap.org/data/2.5/weather?q={city name},{state code},{country code}&appid={API key}
```

Parameters

q required City name, state code and country code divided by comma, Please, refer to [ISO 3166](#) for the state codes or country codes. You can specify the parameter not only in English. In this case, the API response should be returned in the same language as the language of requested location name if the location is in our predefined list of more than 200,000 locations.

appid required Your unique API key (you can always find it on your account page under the "API key" tab)

Call current weather data for one location

- By city name
- By city ID
- By geographic coordinates
- By ZIP code

Call current weather data for several cities

- Cities within a rectangle zone
- Cities in circle

Parameters

q required City name, state code and country code divided by comma, Please, refer to [ISO 3166](#) for the state codes or country codes. You can specify the parameter not only in English. In this case, the API response should be returned in the same language as the language of requested location name if the location is in our predefined list of more than 200,000 locations.

appid required Your unique API key (you can always find it on your account page under the "API key" tab)

mode optional Response format. Possible values are `xml` and `html`. If you don't use the `mode` parameter format is JSON by default. [Learn more](#)

units optional Units of measurement. `standard`, `metric` and `imperial` units are available. If you do not use the `units` parameter, `standard` units will be applied by default. [Learn more](#)

lang optional You can use this parameter to get the output in your language. [Learn more](#)

Input/Outputs

```
➔ ~ > curl 'https://api.openweathermap.org/data/2.5/weather?q=rome,it&
units=metric&appid=a0dddc1224b1a18ef1346d50cc9711f1'
{"coord":{"lon":12.4839,"lat":41.8947},"weather":[{"id":802,"main":"Clo
uds","description":"scattered clouds","icon":"03d"}],"base":"stations",
"main":{"temp":27.57,"feels_like":28.65,"temp_min":24.65,"temp_max":30.
07,"pressure":1014,"humidity":58},"visibility":10000,"wind":{"speed":1.
34,"deg":342,"gust":5.81},"clouds":{"all":40},"dt":1623682231,"sys":{"t
ype":2,"id":2037790,"country":"IT"},➔ ~ >➔ ~ >➔ ~ >➔ ~ >➔ ~ >➔ ~ >
➔ ~ > █
```

User Inputs:

- City
- Country
- Units (metric, imperial)

Desired Outputs:

- Weather
- Temperature

```
1 {
2   "coord": {
3     "lon": 12.4839,
4     "lat": 41.8947
5   },
6   "weather": [
7     {
8       "id": 802,
9       "main": "Clouds",
10      "description": "scattered clouds",
11      "icon": "03d"
12    }
13  ],
14  "base": "stations",
15  "main": {
16    "temp": 27.57,
17    "feels_like": 28.65,
18    "temp_min": 24.65,
19    "temp_max": 30.07,
20    "pressure": 1014,
21    "humidity": 58
22  },
23  "visibility": 10000,
24  "wind": {
25    "speed": 1.34,
26    "deg": 342,
27    "gust": 5.81
```

Claim the Target in Intersight

The screenshot shows the Cisco Intersight Admin console. The left sidebar has a red box around the 'Targets' link under the 'ADMIN' section. The main content area shows a 'Targets' page with a 'Claim Target' button in the top right corner, indicated by a red arrow. Below the button is a summary card for 'Top Targets by Types' showing 11 Claimed, 5 Not Connected, and 22 Connected targets. A table below lists 38 items found, with columns for Name, Status, Type, IP Address, Claimed Time, Claimed By, Connector Version, and Last Update. The table shows various target types such as Standalone M4 Server, UCSM Managed Domain, UCS Director, and VMware vCenter.

Name	Status	Type	IP Address	Claimed Time	Claimed By	Connector Version	Last Update
[Redacted]	Connected	Standalone M4 Server	[Redacted]	Dec 11, 2020 3:18 PM	[Redacted]	1.0.9-1737	2 hours ago
[Redacted]	Connected	Standalone M4 Server	[Redacted]	Jun 20, 2018 12:32 AM	[Redacted]	1.0.9-1737	2 hours ago
[Redacted]	Connected	Standalone M4 Server	[Redacted]	Mar 19, 2019 6:14 PM	[Redacted]	1.0.9-1737	2 hours ago
[Redacted]	Connected	Standalone M4 Server	[Redacted]	May 20, 2020 9:42 AM	[Redacted]	1.0.9-1737	2 hours ago
[Redacted]	Connected	UCSM Managed Domain	[Redacted]	Jan 22, 2020 1:56 PM	[Redacted]	1.0.9-3613	2 hours ago
[Redacted]	Connected	UCS Director	[Redacted]	May 14, 2021 3:29 PM	[Redacted]	1.0.9-1570	2 hours ago
[Redacted]	Connected	UCSM Managed Domain	[Redacted]	Feb 16, 2018 6:56 PM	[Redacted]	1.0.9-3613	2 hours ago
[Redacted]	Connected	Cisco Nexus Dashboard	[Redacted]	Jun 7, 2021 6:25 PM	[Redacted]	1.0.9-763	2 hours ago
[Redacted]	Connected	UCS Director	[Redacted]	Dec 17, 2019 12:35 PM	[Redacted]	1.0.9-1568	2 hours ago
[Redacted]	Connected	VMware vCenter	[Redacted]	May 15, 2021 10:45 PM	[Redacted]	1.0.9-803	2 hours ago

In order to execute automation against any target, you need to claim it.

In our example it will be <https://api.openweathermap.org>

Claim the Target in Intersight

The screenshot shows the 'Select Target Type' interface in Intersight. On the left, there are filters and categories. The main area displays various target types categorized by type: Hyperconverged, Cloud Native, Orchestrator, Network, and Storage. The 'HTTP Endpoint' target type is highlighted with a red box.

Filters: Available for Claiming (checked)

Categories: All (selected), Cloud, Cloud Native, Compute / Fabric, Guest OS Process / APM, Hyperconverged, Hypervisor, Network, Orchestrator, Platform Services, Storage

Target Types:

- Hyperconverged: Nutanix Acropolis, Cisco HyperFlex Cluster
- Cloud Native: Kubernetes
- Orchestrator: Cisco UCS Director, **HTTP Endpoint** (highlighted)
- Network: Cisco APIC, Cisco DCNM, Cisco Nexus Dashboard
- Storage: HPE 3PAR, Dell EMC SC Series, EMC VMAX, EMC ScaleIO, EMC VPLEX, NetApp ONTAP

The screenshot shows the 'HTTP Endpoint' configuration form. The 'Connect through an Intersight Assist' toggle is turned off. The form fields are filled with the following information:

HTTP Endpoint
An external REST API endpoint to be used in Intersight Orchestrator.

Connect through an Intersight Assist

Name *
OpenWeather

Hostname/IP Address *
api.openweathermap.org

Port
443

Authentication Required

Enable HTTPS Protocol

0 - 65535

We don't need to connect through the Assist as the target endpoint is reachable from intersight.com

Once done, click **Claim**

Custom Data Types (CDTs)

Custom Data Types

- Define types and associated constraints and validations
- Needed when no other data type fits (i.e. string, int, etc.)
- Can be used as workflow inputs

The screenshot shows the Cisco Intersight interface. The left sidebar is under the 'CONFIGURE' section, with 'Orchestration' highlighted. The main content area is titled 'Data Types' and shows a table of data types. The table has columns for 'Reference Name', 'Display Name', 'Composite Type', 'System Defined', and 'Description'. There are also summary statistics for 'Composite Type' (Yes: 68, No: 53) and 'System Defined' (Yes: 119, No: 2).

	Reference Name	Display Name	Composite Type	System Defined	Description
	WeatherInputs	WeatherInputs	Yes	No	Inputs for Op
	Outcome	Outcome	Yes	Yes	Intersight Orc
	ResponseParserType	Response Parser	Yes	Yes	Response par
	ResponseParameter...	Response Parameter...	No	Yes	The types of v
	ResponseParameter	Response Parameter	Yes	Yes	This data typ
	OutcomeTypeEnum	Outcome Type Enum	No	Yes	The outcome
	FormattedText	Formatted Text	Yes	Yes	This data typ

Create a Custom Data Type

The screenshot shows the Cisco Intersight 'Data Types' configuration page. The interface includes a sidebar with 'CONFIGURE' selected and 'Orchestration' highlighted. The main area displays a table of data types with columns for Reference Name, Display Name, Composite Type, System Defined, Description, Last Update, and Organization. A 'Create Data Type' button is visible in the top right corner. Red arrows point to the 'Data Types' tab and the 'Create Data Type' button.

	Reference Name	Display Name	Composite Type	System Defined	Description	Last Update	Organization	
<input type="checkbox"/>	WeatherInputs	WeatherInputs	Yes	No	Inputs for OpenWeather...	9 hours ago	default	...
<input type="checkbox"/>	Outcome	Outcome	Yes	Yes	Intersight Orchestrator al...	Jun 11, 2021 3:23 AM	-	...
<input type="checkbox"/>	ResponseParserType	Response Parser	Yes	Yes	Response parser framew...	Jun 11, 2021 3:23 AM	-	...
<input type="checkbox"/>	ResponseParameterTyp...	Response Parameter Type	No	Yes	The types of values that ...	Jun 11, 2021 3:23 AM	-	...
<input type="checkbox"/>	ResponseParameter	Response Parameter	Yes	Yes	This data type takes info...	Jun 11, 2021 3:23 AM	-	...
<input type="checkbox"/>	OutcomeTypeEnum	Outcome Type Enum	No	Yes	The outcome type speci...	Jun 11, 2021 3:23 AM	-	...
<input type="checkbox"/>	FormattedText	Formatted Text	Yes	Yes	This data type takes hold...	Jun 11, 2021 3:23 AM	-	...
<input type="checkbox"/>	ComplexResponsePara...	Complex Response Para...	Yes	Yes	If the given API/device re...	Jun 11, 2021 3:23 AM	-	...
<input type="checkbox"/>	WebUrlType	Web URL	No	Yes	A uniform naming sche...	Jun 11, 2021 3:10 AM	-	...
<input type="checkbox"/>	UuidType	UUID	No	Yes	A Universally Unique Ide...	Jun 11, 2021 3:10 AM	-	...

We create a new data type that can be used in our main workflow input.

In this case, this new custom data type will include the necessary data to built the API call to openweathermap.org:

- City
- Country
- Units

Create a Custom Data Type

Organization *
default

Name *
WeatherInputs

Label *
WeatherInputs

Description
Inputs for OpenWeatherMap API

Set Tags
owner rtorori x Enter a tag in the key:value format. x

Inputs
Simple Composite

We specify a name, a label and a description.

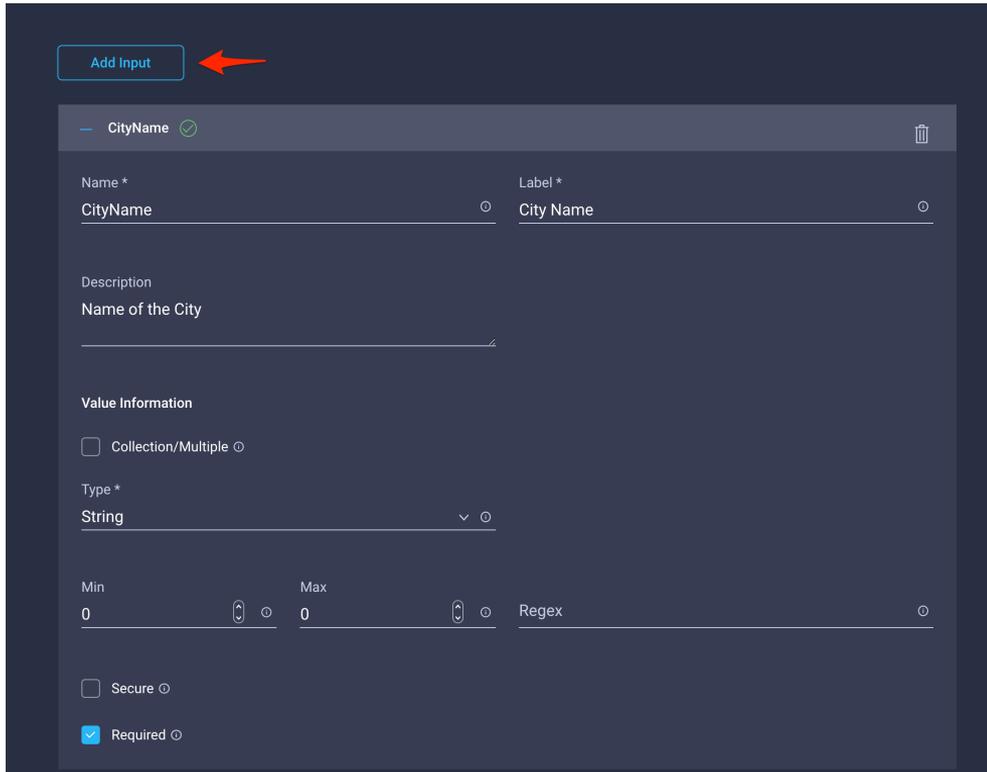
Simple Input:

A simple definition of input. Can be one or a collection of String, Integer, Float, Boolean, JSON or enum

Composite Input:

Multiple combinations of the above

Create a Custom Data Type



The screenshot shows a configuration interface for creating a custom data type. At the top left, there is a button labeled "Add Input" with a red arrow pointing to it. Below this, a configuration card for "CityName" is visible. The card has a header with a checkmark and a trash icon. The main configuration area includes:

- Name ***: CityName
- Label ***: City Name
- Description**: Name of the City
- Value Information**:
 - Collection/Multiple
 - Type ***: String
 - Min**: 0
 - Max**: 0
 - Regex**: (empty)
 - Secure
 - Required

The first input will be the name of the city we want the current weather of (**CityName**)

Type is 'String', we do not enforce any constraints/validations (empty regex)

Marked as 'Required'

Create a Custom Data Type

The screenshot shows a configuration interface for a custom data type. At the top left, there is a button labeled "Add Input" with a red arrow pointing to it. Below this, there are two input fields: "CityName" and "Units". The "Units" field is expanded to show its configuration. It has a "Name" of "Units" and a "Label" of "Units". The "Description" is "Metric or Imperial". Under "Value Information", the "Collection/Multiple" checkbox is unchecked, and the "Type" is set to "String". The "Min" and "Max" values are both set to "0". The "Regex" field contains the pattern "^(metric|imperial)\$" and is highlighted with a red box. The "Secure" checkbox is unchecked, and the "Required" checkbox is checked.

The second input will be the units we want to use for the temperature (**Units**)

Type is 'String'.

As units can be either 'metric' or 'imperial' we enforce validation with a regex. Any other value will be invalid.

Marked as 'Required'

Create a Custom Data Type

The screenshot shows a configuration interface for a custom data type. At the top left, there is a button labeled "Add Input" with a red arrow pointing to it. Below this, there are three input fields: "CityName", "Units", and "Country". The "Country" field is expanded, showing its configuration details. The "Name" and "Label" fields are both set to "Country". The "Description" is "ISO3166 country code (e.g: it, us, uk, ...)". The "Type" is set to "String". The "Regex" field is highlighted with a red box and contains the value `*[A-Za-z][A-Za-z]$`. The "Required" checkbox is checked.

The third input will be the country code of the city we are requesting the weather of (Country)

Type is 'String'.

As we require the country code to be ISO3166 compliant, we enforce validation with a regex. Only two alphabetic characters (case insensitive) will be accepted.

Marked as 'Required'

Ref. (Alpha2):

<https://www.iso.org/obp/ui/#iso:pub:PUB500001:en>

Preview a Custom Data Type

Preview

WeatherInputs

City Name *

Rome

Units *

metrics

Units is not valid. Check info for details

Country *

aaa

Country is not valid. Check info for details

Save

The Preview section allows the user to test their validation regex. Hit 'Save' when done

Task Designer

Web API Executor and Outcomes

Create a Custom Task in the Task Designer

The screenshot shows the Cisco Intersight Task Designer interface. The left sidebar has 'Orchestration' selected. The main area displays a list of tasks with columns for 'Display Name', 'Description', 'Last Update', 'System Defined', and 'Organization'. A 'Create Task' button is highlighted in the top right corner.

Display Name	Description	Last Update	System Defined	Organization
Get Weather	Get Weather from OpenWeather	9 hours ago	No	default
Invoke Web API Request	Invokes the given Web API against t...	Jun 11, 2021 3:23 AM	Yes	-
Dismount Server Virtual Media Device	Dismounts the selected vMedia devi...	Jun 4, 2021 4:20 AM	Yes	-
DismountStandaloneServerVirtualM...	Mounts vMedia device on a standal...	Jun 4, 2021 4:20 AM	Yes	-
Set Server to Server Profile	Sets server to server profile with ser...	Jun 4, 2021 4:20 AM	Yes	-
Remove Server Profile	Deletes server profile given by user. ...	Jun 4, 2021 4:20 AM	Yes	-
Remove Server from Server Profile	Unassigns server from server profile...	Jun 4, 2021 4:20 AM	Yes	-
Remove Server Policies from Profile	Disassociates given list of policies f...	Jun 4, 2021 4:20 AM	Yes	-
MountStandaloneServerVirtualMedi...	Mounts vMedia device on a standal...	Jun 4, 2021 4:20 AM	Yes	-
Mount Server Virtual Media Device	Mounts the given vMedia device on ...	Jun 4, 2021 4:20 AM	Yes	-

This custom task will be responsible to query openweather and fetch the weather details.

It will use the WeatherDetails CDT and can be reused in any workflow

Create a Custom Task using the Web API Executor

1/9

The screenshot shows the Cisco Intersight Designer interface. On the left, a sidebar lists various categories like MONITOR, OPERATE, and CONFIGURE. The 'Tools' pane is open, showing a list of executors under the 'Executors' section. A red arrow points from the 'Invoke Web API Request' executor to a workflow diagram in the center. The workflow consists of three blocks: 'Start', 'Get Weather from OpenWeather' (highlighted with a blue border), and 'Success'. On the right, the 'Task Properties' panel is open, showing the configuration for the selected task. The 'General' tab is active, and the 'Display Name' is set to 'Get Weather...'. Other fields include Organization (default), Reference Name (GetWeather), Description (Get Weather from OpenWeather), Retry Count (3), Retry Delay (60), and Timeout (600).

Drag the web API Request task in the central pane and connect the 'Start' and 'Success' blocks to it.

Select an organization and give the task a name (**Get Weather**).

Create a Custom Task using the Web API Executor

2/9

The screenshot shows a workflow editor with a central task pane. The task is 'Get Weather from OpenWeather' (Executors). It is connected between a 'Start' block and a 'Success' block. A 'Task Properties' panel is open on the right, showing the 'Inputs' tab. An 'Add Input' dialog is in the foreground, showing the following configuration:

- Display Name *: WeatherInputs
- Reference Name *: WeatherInputs
- Description: Inputs to get the weather
- Value Restrictions:
 - Required
 - Collection/Multiple
- Type: WeatherInputs | default (highlighted with a red box)

Drag the web API Request task in the central pane and connect the 'Start' and 'Success' blocks to it.

Select an organization and give the task a name (**Get Weather**).

Click 'Create Task Input'.

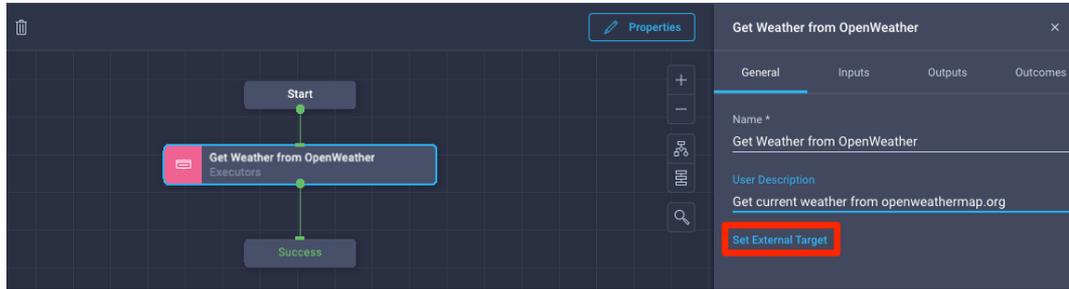
Specify a Display Name as well as a Reference name (**WeatherInputs**).

Mark it as 'Required'.

For the data type, we are going to use our Custom Data type **WeatherInputs**

Create a Custom Task using the Web API Executor

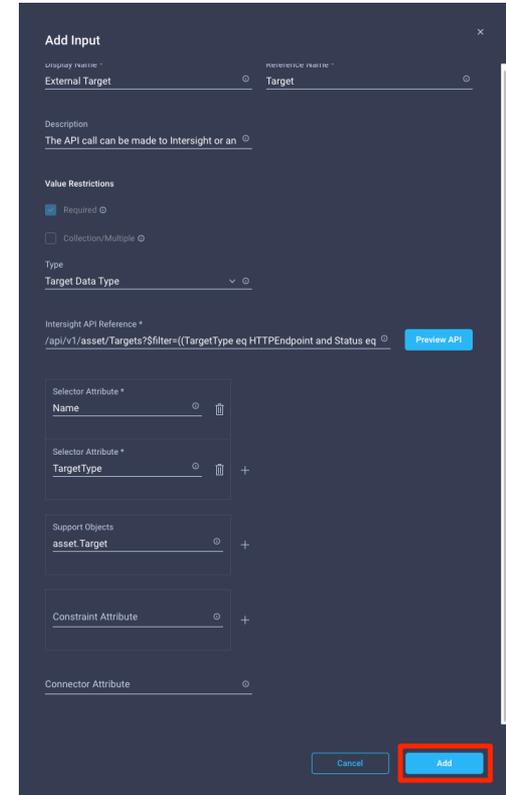
3/9



Click on the executor task, then click on 'Properties'. Give the executor a name (**Get Weather from OpenWeather**) and set the external target. This option will allow to execute API calls against external endpoints as by default those calls will hit the Intersight API.

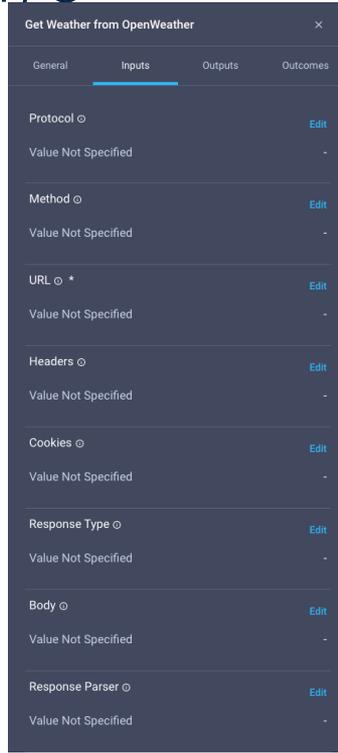
Click **Add** to confirm the default Target Types, this will include all valid targets you can run API calls against.

These are currently: HTTP endpoint, Pure Storage, NetApp, Hitachi, Terraform Cloud, vCenter, MDS, Nexus and Cisco APIC



Create a Custom Task using the Web API Executor

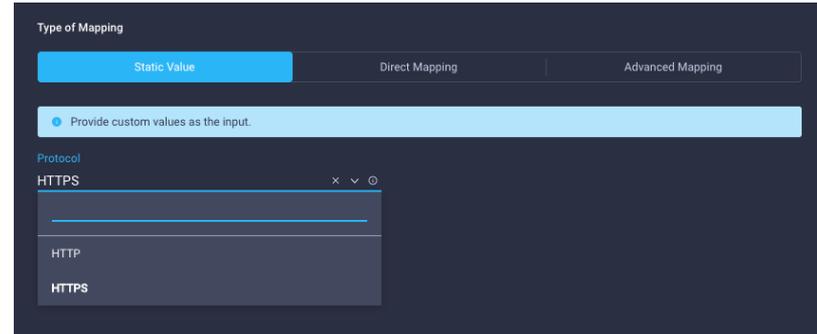
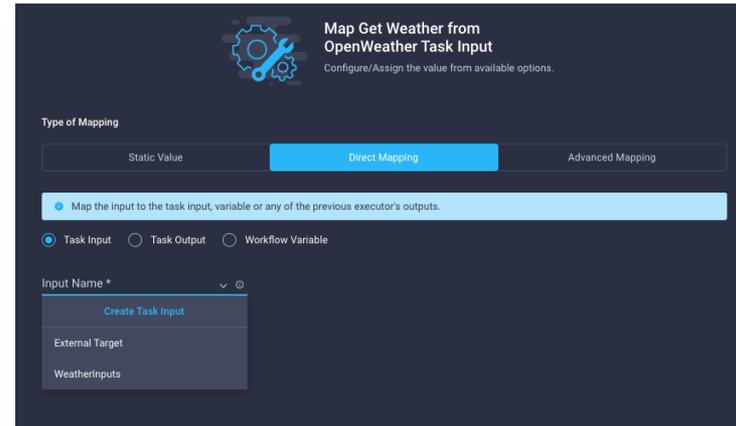
4/9



Inputs Tab

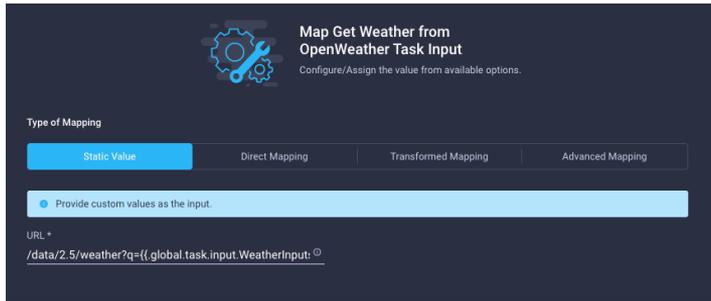
Click on 'edit' to set a value for each input of this task. The 'i' icon will provide detailed information of each input as well as the default value.

When you click on 'edit', you can either map an existing task input, create a new one or set a custom value



Create a Custom Task using the Web API Executor

5/9



Edit the 'URL' input. This is a relative URL that gets appended to the target (in this case openweather), not an FQDN.

We set a **Static Value**.

We want to replicate exactly the same call we did with

`curl:`

```
$ curl 'https://api.openweathermap.org/data/2.5/weather?q=rome,it&units=metric&appid=a0dddc1224b1a18ef1346d50cc9711f1'
```

Fill in with the following (replace with your own OpenWeather API key):

```
/data/2.5/weather?q={{ global.task.input.WeatherInputs.CityName }},{{ global.task.input.WeatherInputs.Country }}&units={{ global.task.input.WeatherInputs.Units }}&appid=a0dddc1224b1a18ef1346d50cc9711f1
```

Everything between `{{ }}` will be replaced dynamically:

`.global.task.input.WeatherInputs.CityName`

One of the custom task input

Name of the Input

Key in the input

Create a Custom Task using the Web API Executor

6/9

Response Type ○	×	Edit
Custom Value		JSON
Response Parser ○		Edit
Value Not Specified		-

Set the 'Response Type' to JSON

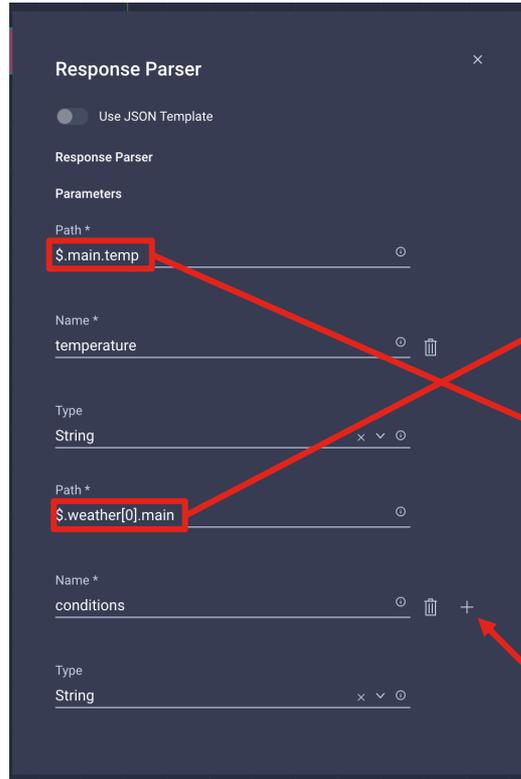
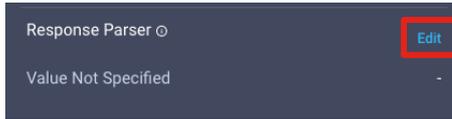
The response parser is going to extract and manipulate the response from the target. In our case we want to extract the **weather conditions** and the **temperature**

```
{
  "coord": {
    "lon": 12.4839,
    "lat": 41.8947
  },
  "weather": [
    {
      "id": 800,
      "main": "Clear",
      "description": "clear sky",
      "icon": "01d"
    }
  ],
  "base": "stations",
  "main": {
    "temp": 25.39,
    "feels_like": 25.19,
    "temp_min": 22.86,
    "temp_max": 27.59,
    "pressure": 1016,
    "humidity": 46
  },
  "visibility": 10000,
  "wind": {
    "speed": 3.09,
    "deg": 260
  },
}
```

Openweathermap response example

Create a Custom Task using the Web API Executor

7/9



```
{
  "coord": {
    "lon": 12.4839,
    "lat": 41.8947
  },
  "weather": [
    {
      "id": 800,
      "main": "Clear",
      "description": "clear sky",
      "icon": "01d"
    }
  ],
  "base": "stations",
  "main": {
    "temp": 25.39,
    "feels_like": 25.19,
    "temp_min": 22.86,
    "temp_max": 27.59,
    "pressure": 1016,
    "humidity": 46
  },
  "visibility": 10000,
  "wind": {
    "speed": 3.09,
    "deg": 260
  }
}
```

Set the jsonpath of the value you want to extract from the response.

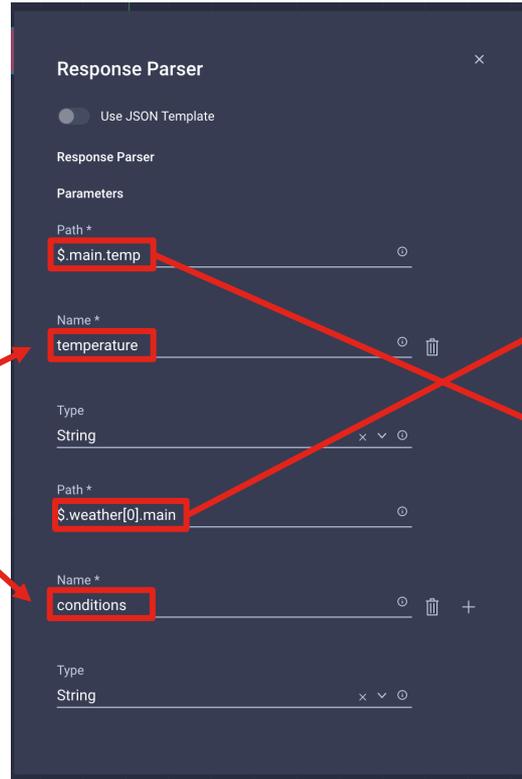
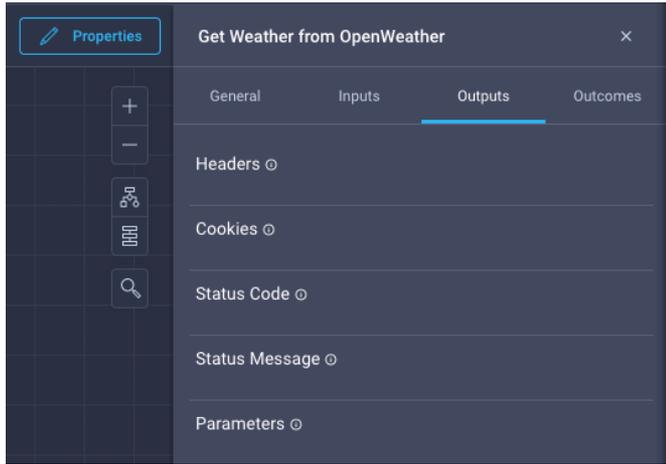
In the example:

1. \$.main.temp will extract 25.39 as a string and set this value to the parameter task output with 'Name' = **temperature**
2. \$.weather[0].main will extract 'Clear' from the 'main' key on the first array element [0] of the 'weather' key. The value will appear in the parameter task output with 'Name' = **conditions**

Click the plus (+) sign to add more parameters

Create a Custom Task using the Web API Executor

8/9

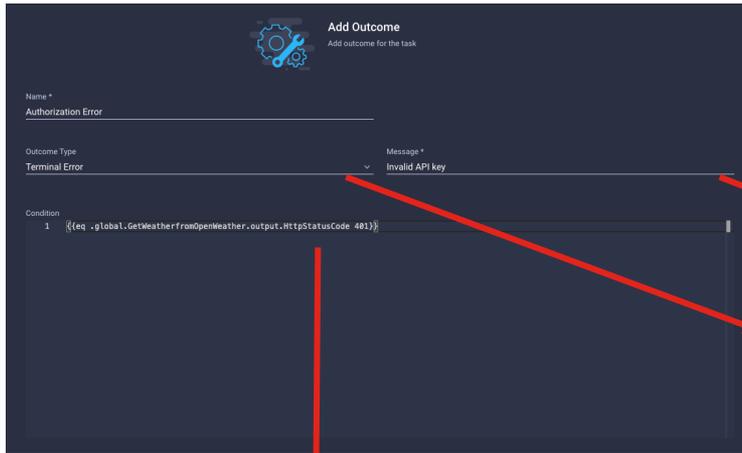
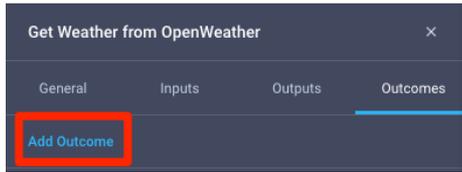


```
{
  "coord": {
    "lon": 12.4839,
    "lat": 41.8947
  },
  "weather" [
    {
      "id": 800,
      "main": "Clear",
      "description": "clear sky",
      "icon": "01d"
    }
  ],
  "base": "stations",
  "main": {
    "temp": 25.39,
    "feels_like": 25.19,
    "temp_min": 22.86,
    "temp_max": 27.59,
    "pressure": 1016,
    "humidity": 46
  },
  "visibility": 10000,
  "wind": {
    "speed": 3.09,
    "deg": 260
  }
},
```

In the task output tab, there is a list of predefined Outputs. The response parser will put the extracted values using the names specified. In our example, temperature and conditions respectively.

These are normal task outputs and can be leveraged elsewhere in ICO

Task Outcomes



`{{eq .global.GetWeatherfromOpenWeather.output.HttpStatusCode 401}}`

A task outcome defines how the task behaves based on configurable conditions.

In the example:

`{{eq .global.GetWeatherfromOpenWeather.output.HttpStatusCode 401}}`

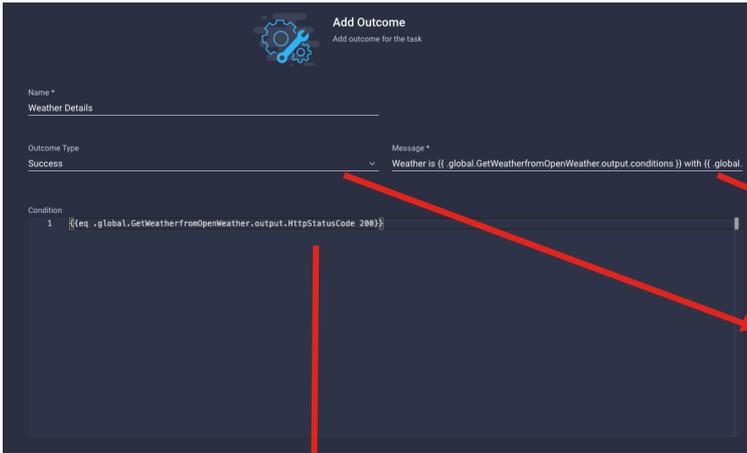
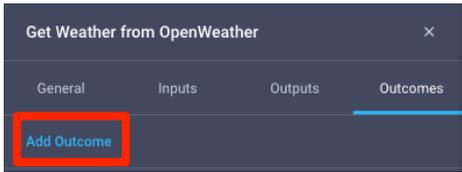
Checks whether the `HttpStatusCode` task out is 401. If that's the case, we return a **Terminal Error** with the custom **Message** "Invalid API key"

Where `GetWeatherfromOpenWeather` is the name of the task with no spaces.



Requests > Get Weather	
Details	Execution Flow
Status	Failed
Name	Get Weather
ID	60ca1cee696f6e2d303fe909

Task Outcomes



`{{eq .global.GetWeatherfromOpenWeather.output.HttpStatusCode 401}}`

We are going to add one more task outcome.

In the example:

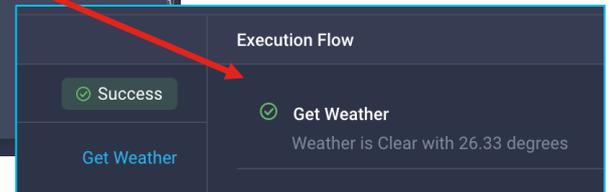
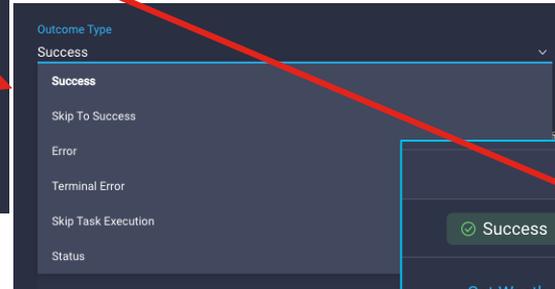
`{{eq .global.GetWeatherfromOpenWeather.output.HttpStatusCode 200}}`

Checks whether the HttpStatusCode task out is 200. If that's the case, we succeed with the custom Message "Weather is {{ .global.GetWeatherfromOpenWeather.output.conditions }} with {{ .global.GetWeatherfromOpenWeather.output.temperature }} degrees"

We can use the task outputs to customise the Message.

Assuming the output **conditions** to be "Clear" and the **temperature** to be "26.33" the message would look like this:

"Weather is Clear with 26.33 degrees"



Additional Outcome Expression Examples

Let's assume the sample scenario where you want to make a task **fail** if the server returns an empty response (example: " [] ")

As of February 2022, ICO doesn't store the full response by default.

You will need to create a parameter from the **Response parser** that stores the full response using '\$' as JSONPATH.

In this example, it will be called **fullResponse**



Case 1 – Full Response as String

Here we want to trigger the outcome if the returned payload string is equal to " [] "

Outcome Condition

Type: Terminal Error

Message: 'Response is empty'

Condition:

```
{{eq .global.InvokeGenericWebApi1.output.fullResponse "[]"}}
```

Case 2 – Full Response as JSON

Here we want to trigger the outcome if the returned payload has zero elements in the array

Outcome Condition

Type: Terminal Error

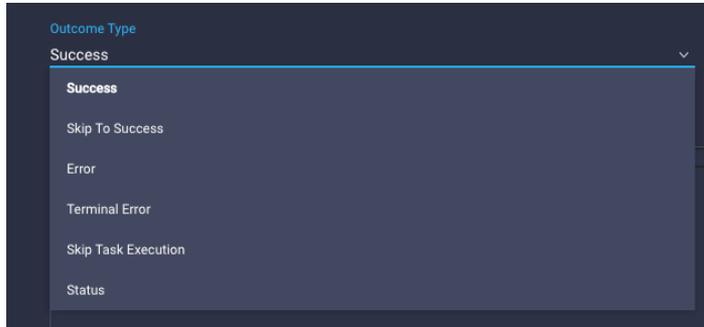
Message: 'Response is empty'

Condition:

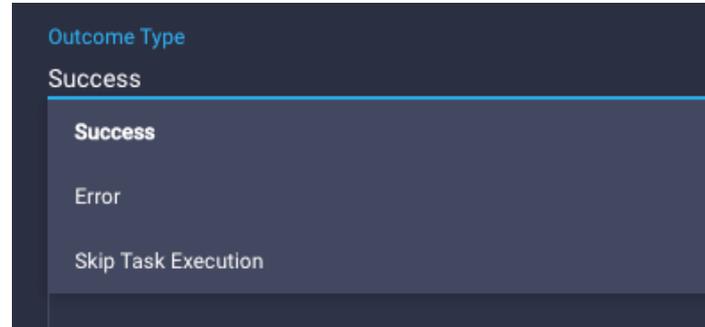
```
{{eq (len .global.InvokeGenericWebApi1.output.fullResponse) 0}}
```

Outcomes Deep Dive

Depending whether you are using an executor as an embedded task or you are building a reusable custom task, you will have different outcome types available.



Custom Task



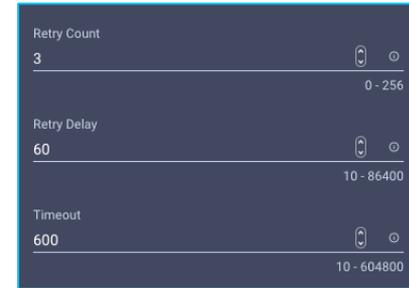
Embedded Task

Retry Parameters

Outcomes use retry parameters to define the behavior of a single executor or the whole custom task.

Retry parameters are available **only** for custom tasks created in the **Task Designer (General Tab)** and are predefined (not configurable) in the **Workflow Designer**.

Note: Retries are *global* to the custom task, this means that the amount of retries are shared among all tasks in a custom task. If a single task consumes all retries, there will be no more retries left for subsequent tasks



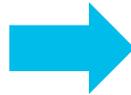
Name	Description	Default	Notes
Retry Count	The number of times a task should be tried before marking as failed	3	Not Configurable in Workflow Designer
Retry Delay	The delay in seconds after which the the task is re-tried	60	Not Configurable in Workflow Designer
Timeout	The timeout value in seconds after which task will be marked as timed out. Max allowed value is 7 days	600 (120 for embedded tasks)	Timeout is set to 120 in the Workflow Designer, not configurable by the user
Retry Policy	Can be Fixed (Retry happens after a specific amount of time) or BackOff	Fixed	Backoff* currently not supported (as of April 2022)

* Backoff will allow delay to increase between retries

Order of Operations

Outcomes are evaluated sequentially,
order matters!

If no outcome will be matched, the task
will follow its default behaviour, that is
timing out or success.



Invoke Web API Request

General Inputs Outputs **Outcomes**

[Add Outcome](#)

Search

First				
Second				
Last				

Outcome Types

Outcome Type	Description	Availability
Success	Task will succeed if condition evaluates to true	TD and WD
Skip To Success	If the condition matches, all subsequent API in a custom task will be skipped and go directly to Success block	Task Designer Only
Error	If the condition matches the whole custom task will be retried assuming there are retries left	TD and WD
Terminal Error	If the condition matches, retry count will be ignore and the whole task will be marked as failed	Task Designer Only
Skip Task Execution	If the condition matches the whole task will be skipped	TD and WD
Status	If the condition matches the whole custom task will be retried (assuming there are retries left), however the user will not be notified about the fail. A user would use the status outcome type when the target is returning a response representing an ongoing activity and they want to retry until the response matches a condition. For instance, retry an API that returns {"Status": "Creating"} until it returns {"Status": "Completed"}	Task Designer Only

Create a Custom Task using the Web API Executor

9/9

The screenshot shows a workflow editor with a 'Start' node, a 'Get Weather from OpenWeather' task, and a 'Success' node. A 'Properties' button is visible above the task. A 'Task Properties' panel is open on the right, showing the 'Outputs' tab. The 'conditions' and 'temperature' outputs are highlighted with red boxes. A 'Get Weather from OpenWeather' task properties dialog is open in the foreground, also showing the 'Outputs' tab. Annotations include a blue arrow pointing to the 'Output name' field, a red arrow pointing to the 'Path: name as configured in the Response Parser' field, and a red arrow pointing to the 'Task outputs' label.

Output name

Path: name as configured in the Response Parser

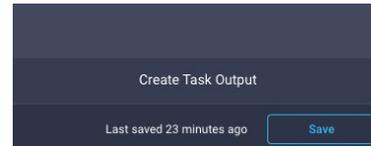
Task outputs

Click on 'Start', then 'Properties' to get the overall task (workflow) properties.

To set the overall task Outputs, click on the **Outputs** tab.

You can map outputs to task outputs. Click on the **Task Output** dropdown menu, these are the task outputs that can be selected. Selecting **Parameters**, we can access the outputs created by the response parser.

Create outputs for both **conditions** and **temperature**. In this example, the name of the outputs matches the keys we used to represent the values extracted by the response parser



When done, click the **Save** button

Workflow Designer

Create a Workflow

The screenshot displays the Cisco Intersight interface for workflow management. The left sidebar shows the navigation menu with 'Orchestration' selected. The main content area includes a 'Workflows' tab, a 'Create Workflow' button (highlighted with a red arrow), and a list of workflows. Summary cards show validation status (7 Invalid, 35 Valid), last execution status (3 Failed, 7 Success), and top 5 workflows by execution count (e.g., Get Weather: 30, GetFirstAvailableIP: 20).

Display Name	Description	Default Version	Executions	Last Execution Status	Last Execution Time	Validation Status	Last Update	Organization	
Get Weather	Get current weather in ...		1	30	Success	a minute ago	Valid	2 minutes ago	default
New Storage Interface	Create a storage IP or ...		1	0			Valid	23 minutes ago	-
Update VMFS Datastore	Expand a datastore on ...		3	0			Valid	Jun 11, 2021 3:10 AM	-
Update Storage Host	Update the storage ho...		3	0			Valid	Jun 11, 2021 3:10 AM	-
Update NAS Datastore	Update NAS datastore ...		1	0			Valid	Jun 11, 2021 3:10 AM	-
Remove VMFS Datast...	Remove VMFS datasto...		5	0			Valid	Jun 11, 2021 3:10 AM	-
Remove Storage Host ...	Remove storage host ...		2	0			Valid	Jun 11, 2021 3:10 AM	-
Remove Storage Host ...	Remove storage host ...		3	0			Valid	Jun 11, 2021 3:10 AM	-
Remove Storage Expor...	Remove the NFS volu...		1	0			Valid	Jun 11, 2021 3:10 AM	-
Remove NAS Datastore	Remove the NAS data...		1	0			Valid	Jun 11, 2021 3:10 AM	-

Create a Workflow

CONFIGURE > Orchestration > Create Workflow

General Designer Mapping Code

Display Name *
Get Current Weather

Reference Name *
GetCurrentWeather

Organization *
default

Version
1

Set Tags
owner: rtortori Enter a tag in the key:value format.

Description
Sample Workflow to get the current weather

Set as Default Version

Retryable

Enable Debug Logs

Workflow Inputs Workflow Outputs

Add Input

No Inputs Defined

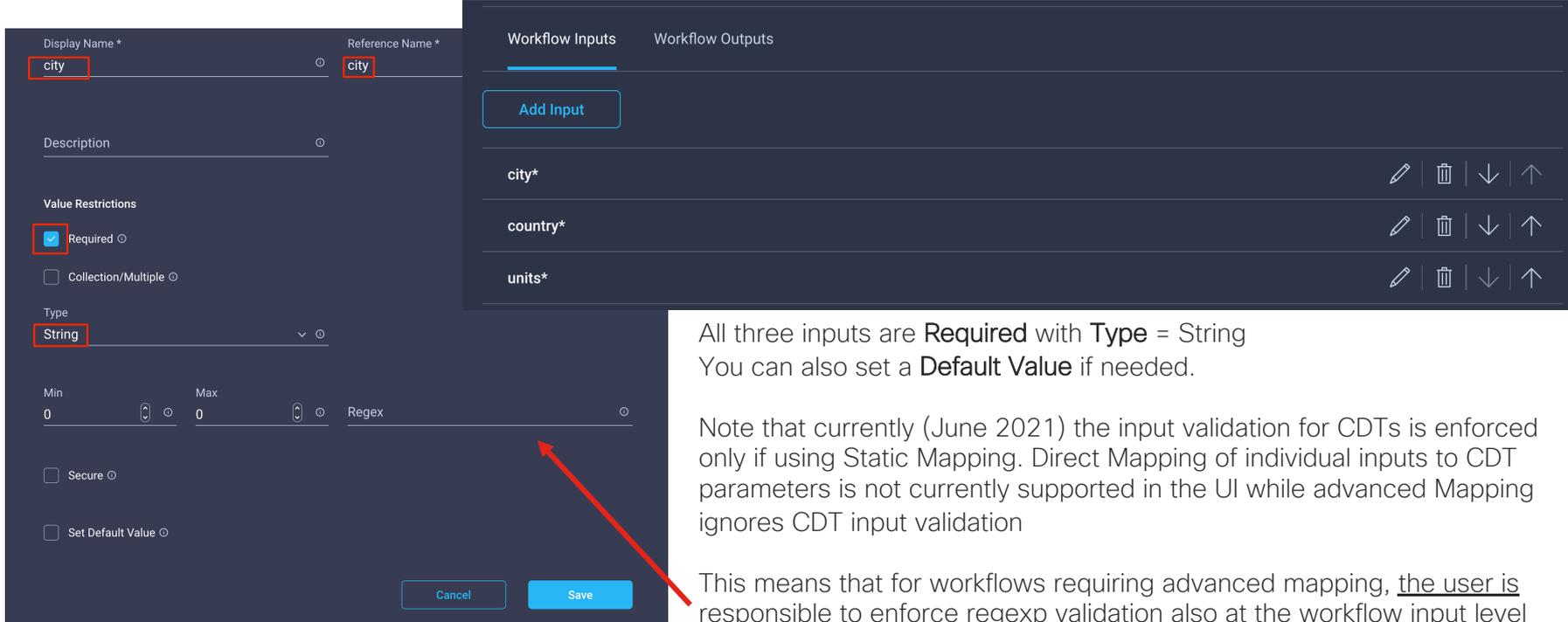
We are going to create a workflow that gets the current weather in a given city and country.

In the **General** tab, we give the workflow a **Display Name** as well as a **Reference Name** which will be used to reference this workflow within ICO.

Clicking on **Add Input**, we want to add 3 inputs:

- City
- Country
- Units

Create a Workflow



The screenshot displays the configuration interface for a workflow. On the left, the 'Display Name' and 'Reference Name' are both set to 'city'. The 'Value Restrictions' section has 'Required' checked. The 'Type' is set to 'String'. Below this, there are fields for 'Min' (0), 'Max' (0), and 'Regex'. At the bottom, there are 'Cancel' and 'Save' buttons. On the right, the 'Workflow Inputs' tab is active, showing a list of three inputs: 'city*', 'country*', and 'units*'. Each input has edit, delete, and sort icons.

All three inputs are **Required** with **Type** = String
You can also set a **Default Value** if needed.

Note that currently (June 2021) the input validation for CDTs is enforced only if using Static Mapping. Direct Mapping of individual inputs to CDT parameters is not currently supported in the UI while advanced Mapping ignores CDT input validation

This means that for workflows requiring advanced mapping, the user is responsible to enforce regexp validation also at the workflow input level

Create a Workflow

The screenshot shows the workflow designer interface with the 'Designer' tab selected. The left-hand 'Tools' palette is open, displaying a list of tasks under the 'General' category. The 'Get Weather' task is highlighted with a red box, and a red arrow points from it to the workflow canvas. On the canvas, a 'Start' block is connected to a 'Get Weather' block (General). The 'Get Weather' block has a green connector on its top edge, which is connected to the 'Start' block. Below the 'Get Weather' block, there are two output blocks: 'Success' (green) and 'Failed' (red). A dashed red line connects the red connector on the right side of the 'Get Weather' block to the 'Failed' block. The 'Success' block is connected to the green connector on the bottom edge of the 'Get Weather' block.

In the **Designer** tab, drag the 'Get Weather' custom task we created in the palette.

Connect the **Start** block to the green connector of the custom task, as well as the **Success** block as shown.

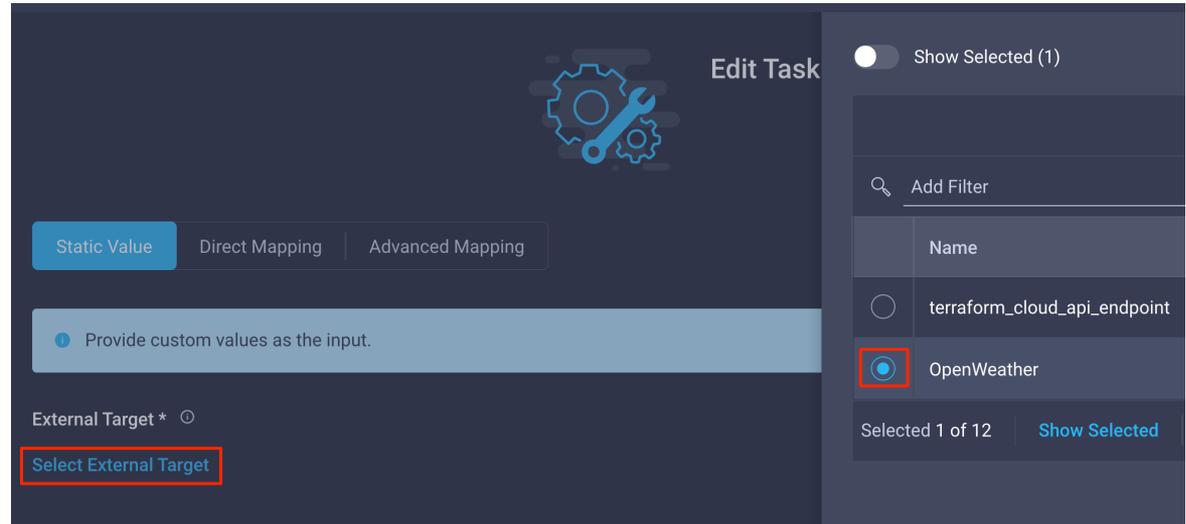
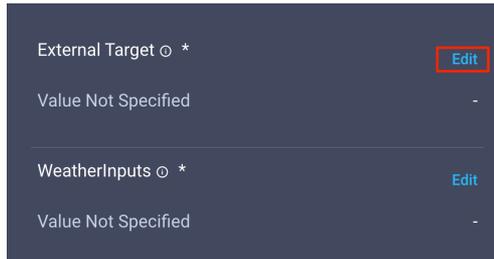
Create a Workflow

The image shows a workflow editor interface. On the left, a workflow diagram is displayed on a dark grid background. It starts with a 'Start' node, followed by a 'Get Weather' task (highlighted with a blue border). From the 'Get Weather' task, two paths emerge: a solid green line leading to a 'Success' node, and a dashed red line leading to a 'Failed' node. On the right, a configuration panel for the 'Get Weather' task is open. The 'Inputs' tab is selected and highlighted with a red box. The panel shows two input fields: 'External Target' and 'WeatherInputs', both marked with an asterisk and an 'Edit' link. The values for both are currently 'Value Not Specified'.

Clicking on the **Inputs** tab on the 'Get Weather' task properties, we can find two inputs are required:

1. The external API target endpoint
2. WeatherInputs, our Custom Data Type

Create a Workflow

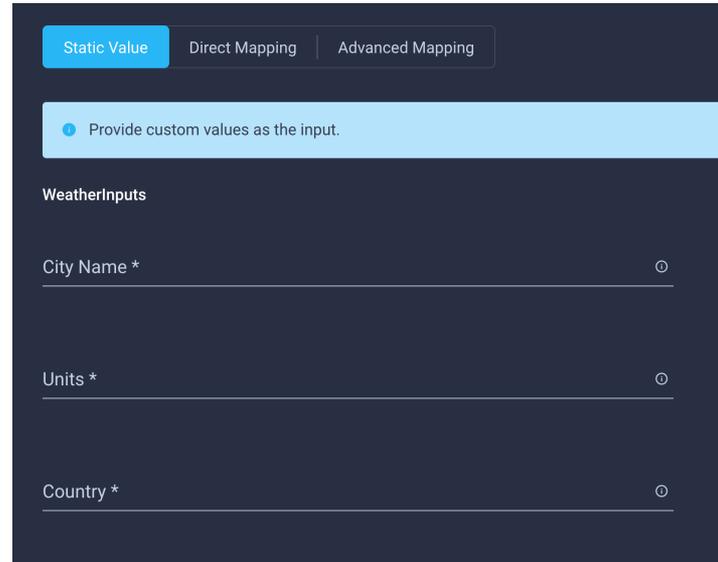
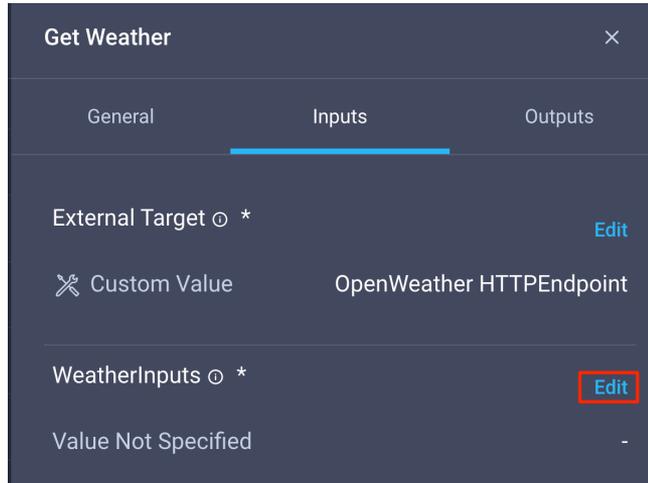


On **External Target**, click on **Edit**.

You can map the external target using a **Static Value** (the workflow will always execute against this target), **Direct Mapping** (map to an existing workflow input or task output) or **Advanced Mapping** (leverage Go templates for advanced mappings/transformations)

Click on **Select External Target** to statically map the input to the 'OpenWeather' target

Create a Workflow



On **WeatherInputs**, click on **Edit**.

The default static value mapping will give you the opportunity to hardcode the three required values of the CDT. However, since we want them to be mapped to the workflow input (in order for the user to execute the workflow for multiple cities) and there is currently no way in the UI to map individual inputs to CDT properties, we are going to select **Advanced Mapping**

Create a Workflow

Custom Data Type

Inputs

Simple Composite

Add Input

+ CityName ✓

+ Units ✓

+ Country ✓

Static Value Direct Mapping Advanced Mapping

Write Go templates to map the input as required

```
1 {
2   "CityName" "{{.global.workflow.input.city}}",
3   "Units" "{{.global.workflow.input.units}}",
4   "Country" "{{.global.workflow.input.country}}",
5 }
```

Workflow Inputs

Workflow Inputs Workflow Outputs

Add Input

city*

country*

units*

The WeatherInputs CDT we created has three properties:

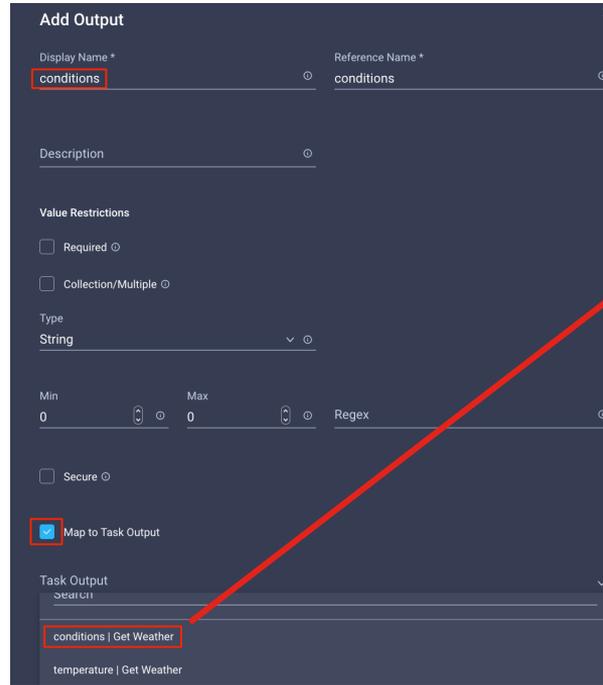
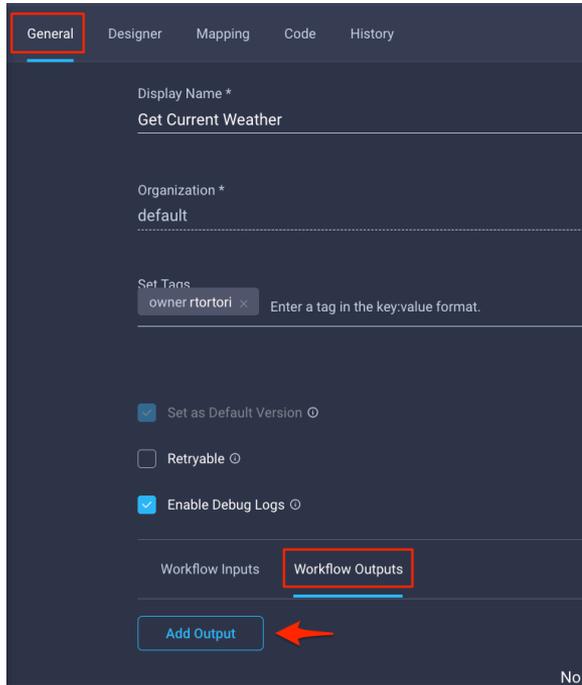
1. CityName
2. Units
3. Country

This Go Template maps each CDT parameter to specific workflow inputs. Refer to the documentation for additional details on Template Parameters:

https://www.intersight.com/help/resources/web_api_request#template_parameters

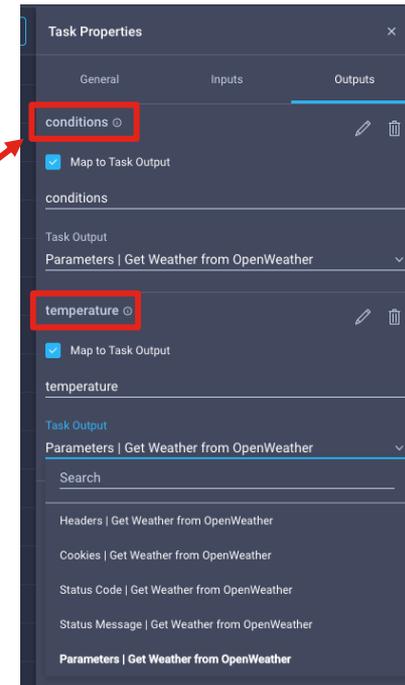
```
{
  "CityName": "{{.global.workflow.input.city}}",
  "Units": "{{.global.workflow.input.units}}",
  "Country": "{{.global.workflow.input.country}}",
}
```

Create a Workflow



Recall: we have set two outputs on the custom tasks:

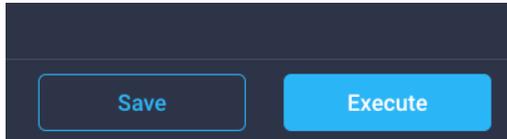
- conditions
- temperature



Add two **Workflow Outputs** named 'conditions' and 'temperature' then map them to appropriate 'Parameters' task outputs

Validate and Execute a Workflow

Validate and Execute the Workflow



Save the workflow. The system will perform validation checks.

Once done, click on **Execute** to run the workflow

Enter Workflow Input - Get Current Weather

Organization *
default

Workflow Instance Name
Get Current Weather

city *
rome

country *
it

units *
metric

Cancel Execute

A dark blue modal form titled 'Enter Workflow Input - Get Current Weather'. It contains several input fields: 'Organization *' with a dropdown menu showing 'default'; 'Workflow Instance Name' with a dropdown menu showing 'Get Current Weather'; 'city *' with a text input containing 'rome'; 'country *' with a text input containing 'it'; and 'units *' with a text input containing 'metric'. At the bottom, there are two buttons: 'Cancel' and 'Execute'.

Select an organization, the required inputs of the workflow should now appear.

Fill in the details with your favourite city, country (ISO3166) as well as the units (metric or imperial).

Click on **Execute**

Validate and Execute the Workflow

The screenshot displays the workflow execution interface. On the left, a workflow diagram shows a 'Start' node leading to a 'Get Weather' task, which then branches into 'Success' and 'Failed' nodes. The 'History' tab is selected at the top. The main panel shows the execution details for 'Get Current Weather - Today at 12:46 PM'. The status is 'Success'. Below this, a list of execution steps is shown, including 'Start' and 'Get Weather'. The 'Get Weather' step is expanded to show 'Debug Logs'. The logs are structured as follows:

```
Object:
  Retry: 0
  TaskInstId: a8469a18-4889-4c15-9a9f-8e37c6b981af
  TaskDebugLogEntries:
    GetWeatherfromOpenWeather:
      ContentType: json
      Headers:
        Content-Type: application/json
      Method: GET
      Outcome:
        Outcome: Weather is Clear with 32.27 degrees
        OutcomeType: Success
```

Right after the execution you'll move to the **History** tab of the workflow.

Here you can check details of all workflow executions

Here is the status of the selected execution

Shows **Debug Logs** (if enabled in the workflow under the **General** tab) for deeper inspection or troubleshooting. Example: TargetURL, Headers, ContentType, Outcomes, etc.

This inset shows a control panel with a checked checkbox labeled 'Enable Debug Logs'. Below it are sections for 'Workflow Inputs' and 'Workflow Outputs', with an 'Add Input' button at the bottom.

Validate and Execute the Workflow

Execution: Get Current Weather - Today at 12:46 PM

Organization: default

Status: Success

Workflow Inputs

Workflow Outputs

Start Jun 21, 2021 12:46:25 PM

1 Get Weather Jun 21, 2021 12:46:30 PM

- Debug Logs
- Logs
- Inputs
 - WeatherInputs: { 3 }
 - CityName: rome
 - Country: it
 - Units: metric
 - External Target: { 2 }
 - Moid: 60ba98ac6f72612d31ddb1d0
 - ObjectType: asset.Target
- Outputs

Success Jun 21, 2021 12:46:30 PM



The **Inputs** fields will display the workflow inputs value used in this specific execution

Start Jun 21, 2021 12:46:25 PM

1 Get Weather Jun 21, 2021 12:46:30 PM

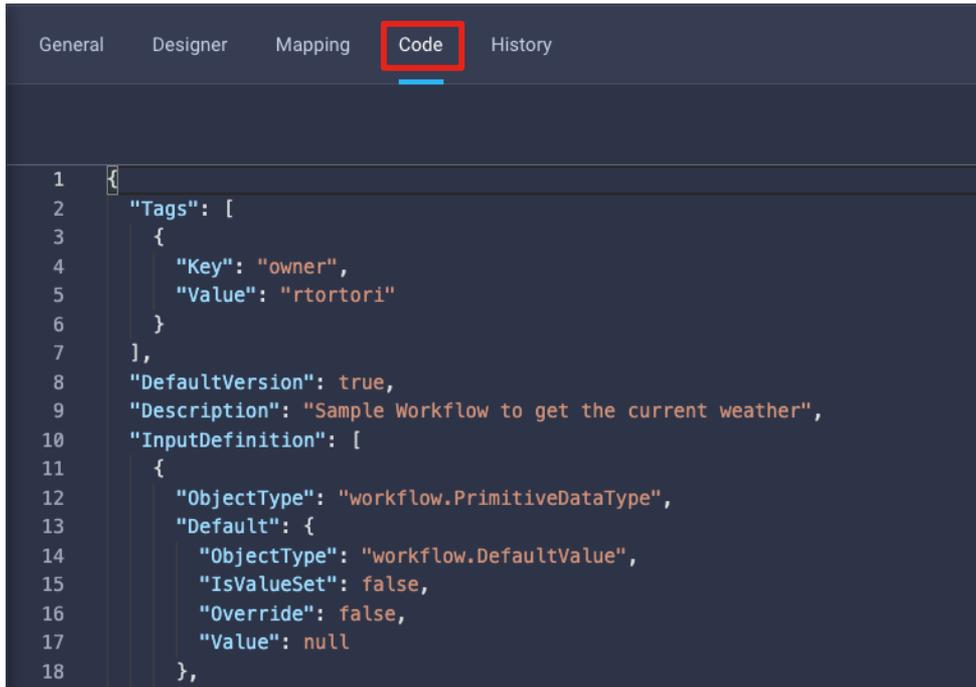
- Debug Logs
- Logs
- Inputs
- Outputs

```
ConfigResults: [ 1 ]
  Object: { 4 }
    ConfigResCtx: { 1 }
      EntityData: { 1 }
        task: workflow.ApiTask
      Message: Weather is Clear with 32.27 degrees
      State: Ok
      Type: Config
    conditions: Clear
    temperature: 32.27
```

The **Outputs** fields will display the workflow outputs.

You can concatenate outputs to other task inputs

Validate and Execute the Workflow

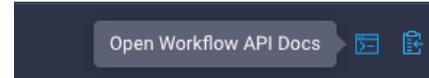


```
1 {
2   "Tags": [
3     {
4       "Key": "owner",
5       "Value": "rtortori"
6     }
7   ],
8   "DefaultVersion": true,
9   "Description": "Sample Workflow to get the current weather",
10  "InputDefinition": [
11    {
12      "ObjectType": "workflow.PrimitiveDataType",
13      "Default": {
14        "ObjectType": "workflow.DefaultValue",
15        "IsValueSet": false,
16        "Override": false,
17        "Value": null
18      },
```

The **Code** tab displays the generated code for this workflow.

There will eventually be the ability to live edit the code.

For now it's there to use as reference for things you can't see in the UI, such as unique names of the task blocks as they're put on the canvas



Clicking on the 'terminal' icon in the upper right corner, will open the REST API docs for the workflows directly, where you can optionally copy the current code and make modifications

Validate and Execute the Workflow

The screenshot shows the 'Requests' interface with tabs for 'All', 'Active', and 'Completed'. The 'Active' tab is selected, and the main area displays 'No Active Requests'. A red arrow points to the 'View All' button at the bottom right.

View all will display all Requests

The screenshot shows the 'Requests' interface with a list of requests. A red arrow points to the 'View All' button at the bottom right.

Name	Status	Initiator	Target Type	Target Name	Start Time	Duration	ID
Get Current Weather	Success	rtortori@cisco.com	-	-	2 hours ago	5 s	60d06e00696f6e2d305...
Get Current Weather	Success	rtortori@cisco.com	-	-	Jun 18, 2021 02:55:14 ...	6 s	60cc97b2696f6e2d30b...
Get Current Weather	Success	rtortori@cisco.com	-	-	Jun 18, 2021 02:54:07 ...	1 s	60cc976f696f6e2d30b...

The screenshot shows the 'Requests' interface with a context menu open over a request. A red arrow points to the context menu.

Name	Status	Initiator
Get Current Weather	Success	rtortori@cisco.com
Get Current Weather	Success	rtortori@cisco.com
Get Current Weather	Success	rtortori@cisco.com
Get Current Weather	Success	rtortori@cisco.com
Get Current Weather	Success	rtortori@cisco.com

- Proceed
- Terminate
- Retry
- Pause
- Retry Failed

You can select one or multiple Requests and execute actions

Validate and Execute the Workflow

The screenshot displays a workflow execution interface. At the top, the breadcrumb 'Requests > Get Current Weather' is visible. The right-hand navigation bar includes notification icons for 29 errors and 27 warnings, a checkmark icon, a filter icon with '34', a search icon, a settings icon, a help icon, and the user profile 'Riccardo Tortorici'.

The interface is divided into two main sections: 'Details' on the left and 'Execution Flow' on the right.

Details Section:

- Status:** Success (indicated by a green checkmark icon)
- Name:** [Get Current Weather](#)
- ID:** 60d06e00696f6e2d3052160c
- Source Type:** Orchestration
- Source Name:** [GetCurrentWeather](#)
- Initiator:** rtortori@cisco.com
- Start Time:** Jun 21, 2021 12:46 PM
- End Time:** Jun 21, 2021 12:46 PM
- Duration:** 5 s
- Organizations:** [default](#)

Execution Flow Section:

- Task:** [Get Weather](#) (indicated by a green checkmark icon)
- Output:** Weather is Clear with 32.27 degrees
- Timestamp:** Jun 21, 2021 12:46 PM

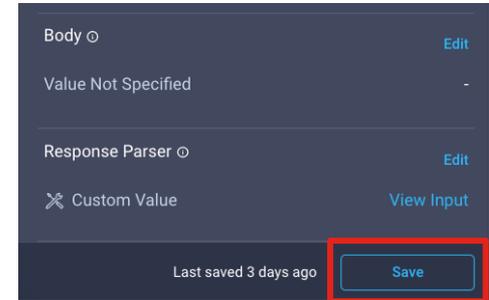
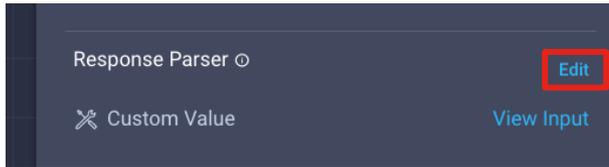
Clicking on one **Request**, you will get its **Execution Flow**, which will display outcomes for each tasks and external workflows invoked in this execution. In this case, we had a single task in our workflow but you can nest multiple workflows and use them as atomic tasks.

Use Task Outputs as Inputs
for other Tasks

Modify Custom Tasks – Response Parser

Scope: extract from API response the 'longitude' and 'latitude'

In the custom task, we edit the **Response Parser** to extract two additional parameters



Modify Custom Tasks – Custom Task Outputs

The screenshot shows a workflow editor with a task named "Get Weather from OpenWeather" (Executors). The task is connected to a "Start" node and a "Success" node. The "Properties" panel is open to the "Outputs" tab, showing a list of outputs: "conditions", "temperature", and "payload". Each output has a "Map to Task Output" checkbox checked. At the bottom of the panel, a "Create Task Output" button is highlighted with a red box.

The screenshot shows the "Create Task Output" dialog for the "longitude" output. The dialog has a "Map to Task Output" checkbox checked. Below the checkbox, the "Task Output" is set to "Parameters | Get Weather from OpenWeather". At the bottom of the dialog, a "Save" button is highlighted with a red box. The text "Last saved 3 days ago" is visible next to the "Save" button.

Check New Task Outputs

Execution: Get Current Weather - Today at 3:50 PM

Organization: default

Status: Success

Workflow Inputs

Workflow Outputs

- conditions: Rain
- temperature: 30.91

This screenshot shows the high-level view of a workflow execution. The 'Workflow Outputs' section is highlighted with a red box, and a red arrow points from this box to the detailed view on the right.

A new execution highlights the new task outputs.

Note: **Workflow Outputs** still shows the original outputs

Get Weather Jun 21, 2021 03:50:22 PM

- Debug Logs
- Logs
- Inputs
- Outputs

```
ConfigResults: [ 1 ]
├── Object: { 4 }
│   ├── ConfigResCtx: { 1 }
│   │   └── EntityData: { 1 }
│   │       └── task: workflow.ApiTask
│   ├── Message: Weather is Rain with 30.91 degrees
│   ├── State: Ok
│   └── Type: Config
└── conditions: Rain
    ├── latitude: 41.8947
    ├── longitude: 12.4839
    └── temperature: 30.91
```

This screenshot shows the detailed configuration and output of the 'Get Weather' task. The 'latitude' and 'longitude' values are highlighted with a red box, and a red arrow points from the text 'Note: Workflow Outputs still shows the original outputs' to this box.

Create a New Task to Get Detailed Info - 1/3

The 'Task Properties' dialog is shown with the 'General' tab selected. It contains the following fields and controls:

- Organization *: default (dropdown)
- Task Name *: Get Forecasts (text input)
- Description: Get Forecasts (text input)
- Retry Count: 3 (spinner, range 0 - 256)
- Retry Delay: 60 (spinner, range 10 - 86400)
- Timeout: 600 (spinner, range 10 - 604800)
- Set Tags: Enter a tag in the key:value format. (text input)
- Enable Rollback Task: (checkbox, currently unchecked)

The 'Task Properties' dialog is shown with the 'Inputs' tab selected. It contains the following inputs:

- External Target ○ *: (input field with up/down arrows, edit, and delete icons)
- latitude *: (input field with up/down arrows, edit, and delete icons)
- longitude *: (input field with up/down arrows, edit, and delete icons)
- units *: (input field with up/down arrows, edit, and delete icons)

Name: 'Get Forecasts'
All inputs as Strings.

The 'Task Properties' dialog is shown with the 'Outputs' tab selected. It contains the following outputs:

- humidity (input field with edit and delete icons)
- Map to Task Output
- humidity (input field)
- Task Output: Parameters | Get Forecasts (dropdown)

Create one output named 'humidity'.
This will be mapped to the humidity
response parser

Create a New Task to Get Detailed Info - 2/3

The screenshot displays a workflow editor interface. On the left, a workflow is shown on a grid background, consisting of three nodes: a 'Start' node at the top, a 'Get Forecast' executor node in the middle, and a 'Success' node at the bottom. The 'Get Forecast' node is highlighted with a blue border. To the right of the workflow is a 'Properties' panel for the selected 'Get Forecast' task. The panel has a title 'Get Forecast' and a close button. It contains four tabs: 'General', 'Inputs', 'Outputs', and 'Outcomes'. The 'General' tab is active. Under the 'General' tab, there are three fields: 'Name *' with the value 'Get Forecast', 'User Description' with the text 'Invokes the given Web API against the given endpoint. The', and a checked checkbox labeled 'Set External Target'. A red arrow points from the 'Set External Target' checkbox to the text below the screenshot.

Executor properties.
Name: 'Get Forecast'
Set External Target checked (calls will run against the same target)

Create a New Task to Get Detailed Info - 2/3

Get Forecast

General Inputs Outputs Outcomes

Protocol Edit

Value Not Specified

Method Edit

Custom Value GET

URL Edit

Custom Value `/data/2.5/onecall?lat={{global.task.input.latitude}}&lon={{global.task.input.longitude}}&units={{global.task.input.units}}&appid=a0dddc1224b1a18ef1346d50cc9711f1`

Headers Edit

Value Not Specified

Cookies Edit

Value Not Specified

Response Type Edit

Custom Value JSON

This call requires latitude and longitude. They will be mapped to the latitude and longitude outputs of the 'Get Weather' task within the main workflow.

```
/data/2.5/onecall?lat={{global.task.input.latitude}}&lon={{global.task.input.longitude}}&units={{global.task.input.units}}&appid=a0dddc1224b1a18ef1346d50cc9711f1
```

Response Parser

Use JSON Template

Response Parser

Parameters

Path * `$.current.humidity`

Name * `humidity`

Type `String`

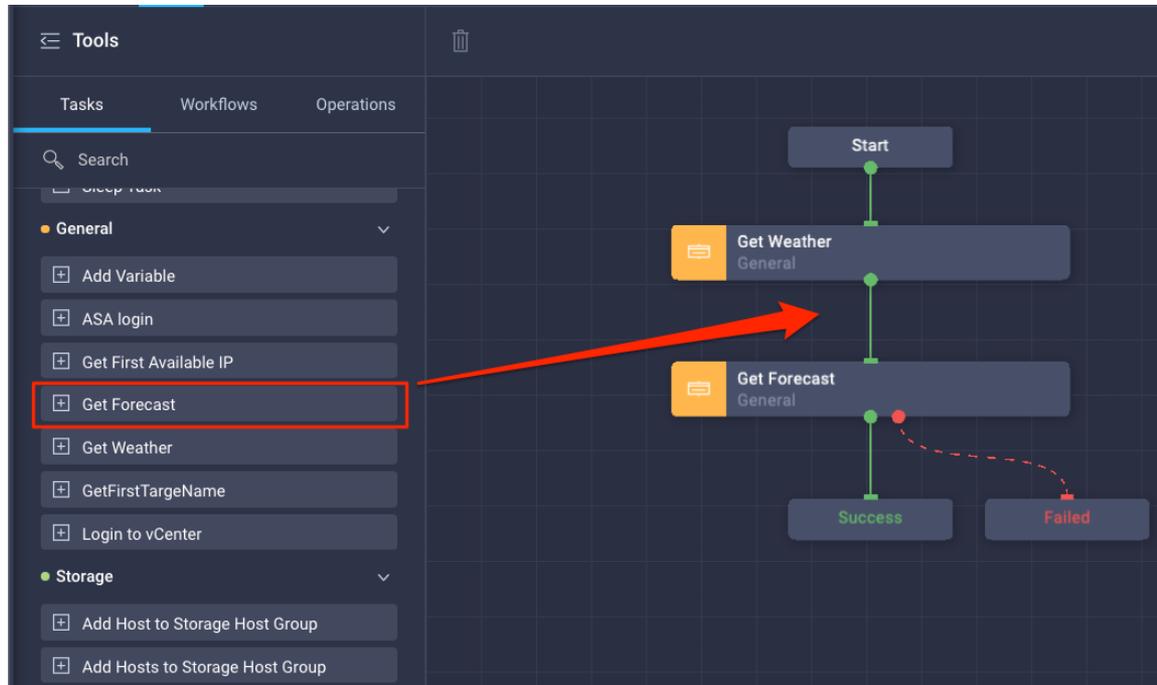
Response parser is extracting the current humidity and store in the 'humidity' parameter as a string

Save once done

Last saved an hour ago

Save

Add New Tasks to an Existing Workflow



In the main workflow, drag the new 'Get Forecast' task in the designer right below the 'Get Weather' task

Add New Tasks to an Existing Workflow

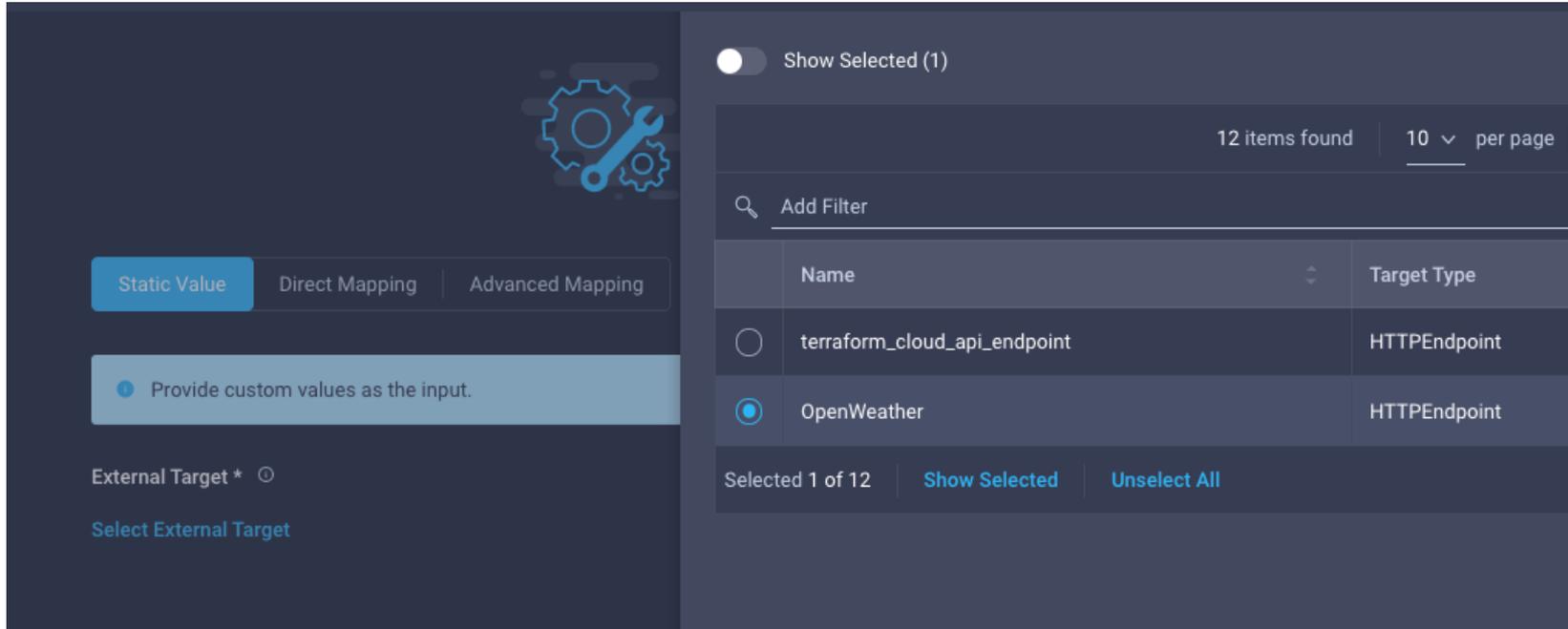
The screenshot shows a configuration window for a task named 'Get Forecast'. The window has three tabs: 'General', 'Inputs', and 'Outputs'. The 'Inputs' tab is selected and highlighted with a blue underline. Below the tabs, there are four input fields, each with a label, a value, and an 'Edit' link. The labels are 'External Target *', 'latitude *', 'longitude *', and 'units *'. The values for all four fields are 'Value Not Specified'. The 'Edit' links are in blue text.

Input Name	Value	Action
External Target *	Value Not Specified	Edit
latitude *	Value Not Specified	Edit
longitude *	Value Not Specified	Edit
units *	Value Not Specified	Edit

The 'Get Forecast' task wants four inputs:

- External Target – it will be the same target we claimed for the weather
- Latitude – we get this value from the previous task output
- Longitude – we get this value from the previous task output
- Units – we get this value from the workflow input

Add New Tasks to an Existing Workflow



The screenshot displays a workflow configuration interface. On the left, there are three tabs: "Static Value" (selected), "Direct Mapping", and "Advanced Mapping". Below the tabs, a blue box contains a bullet point and the text "Provide custom values as the input." At the bottom left, there is a section for "External Target * ⓘ" with a link "Select External Target". On the right, a "Show Selected (1)" toggle is visible. Below it, a summary bar shows "12 items found" and "10 per page". A search bar labeled "Add Filter" is present. A table lists external targets with columns for "Name" and "Target Type". The "OpenWeather" target is selected. At the bottom of the table, there are buttons for "Selected 1 of 12", "Show Selected", and "Unselect All".

	Name	Target Type
<input type="radio"/>	terraform_cloud_api_endpoint	HTTPEndpoint
<input checked="" type="radio"/>	OpenWeather	HTTPEndpoint

External Target will be a static mapping

Add New Tasks to an Existing Workflow

Edit Task Input

Static Value **Direct Mapping** Transformed Mapping Advanced Mapping

Map the input to the workflow input or any of the previous task's outputs.

Workflow Input Task Output

Task Name * Output Name *

For the latitude, we want the value to be the output of the previous task ('Get Weather')

Add New Tasks to an Existing Workflow

Edit Task Input

Static Value **Direct Mapping** Transformed Mapping Advanced Mapping

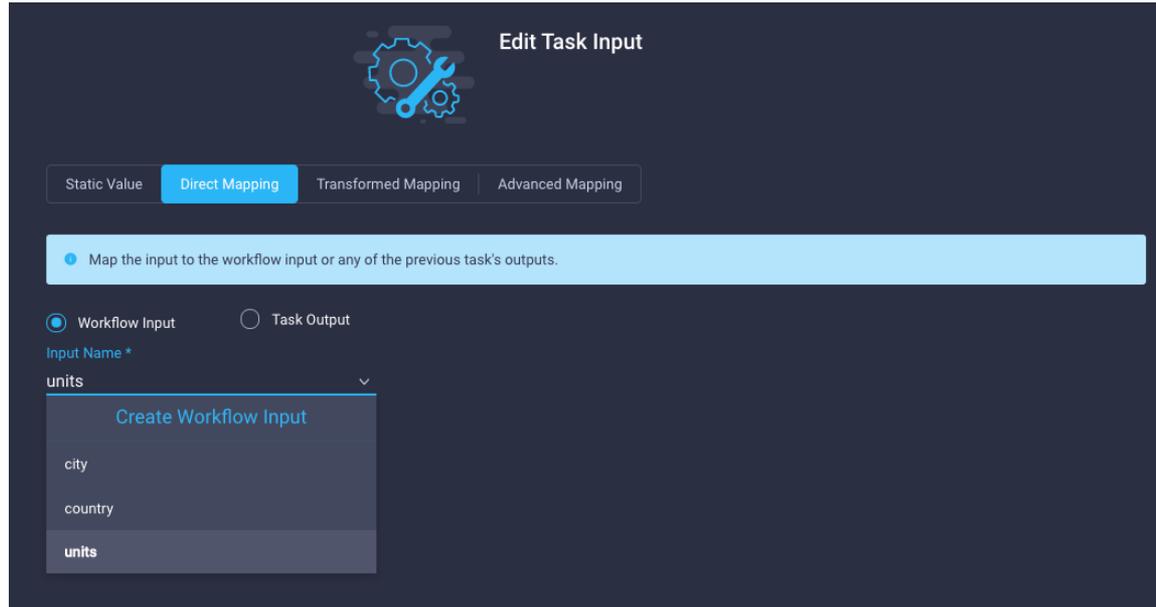
1 Map the input to the workflow input or any of the previous task's outputs.

Workflow Input Task Output

Task Name * Output Name *

For the longitude, we want the value to be the output of the previous task ('Get Weather')

Add New Tasks to an Existing Workflow



Edit Task Input

Static Value **Direct Mapping** Transformed Mapping Advanced Mapping

Map the input to the workflow input or any of the previous task's outputs.

Workflow Input Task Output

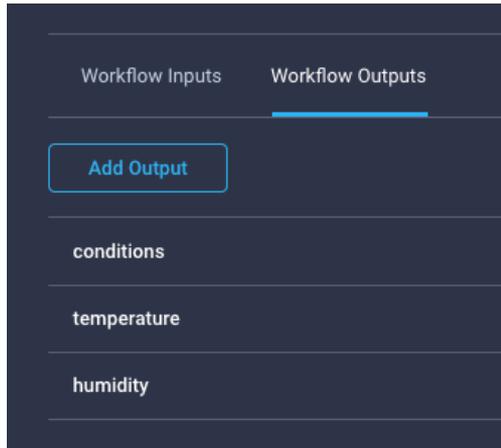
Input Name *

units

- Create Workflow Input
- city
- country
- units

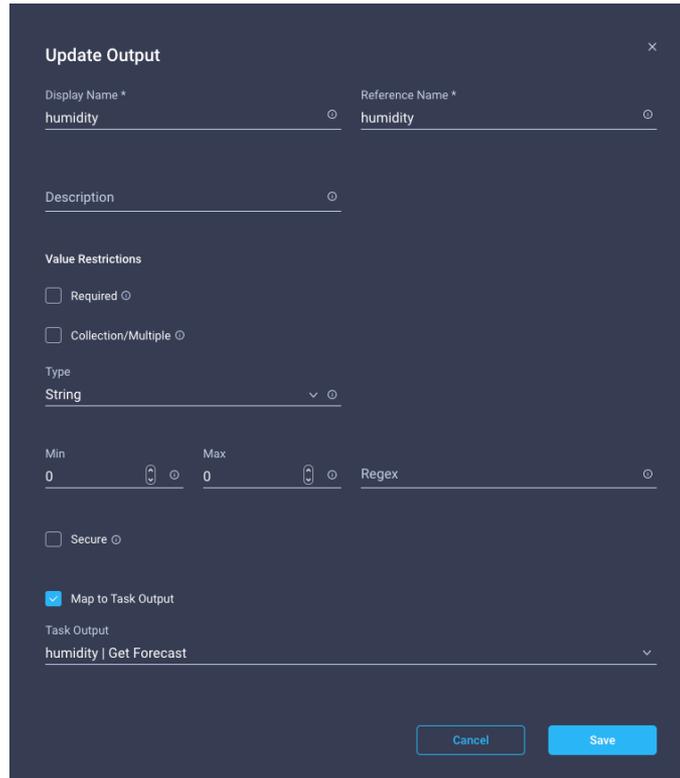
For the units, we take this value from the workflow input

Add Workflow Outputs



We are going to add an additional workflow output called 'humidity'.

We will **Map to Task Output** 'Get Forecast' output called 'humidity'



Save the workflow, then **Execute**



Check Workflow Execution Details

Execution: Get Current Weather - Today at 7:07 PM

Organization: default

Status: Success

Workflow Inputs

Workflow Outputs

```
conditions: Clear
temperature: 29.93
humidity: 34
```

Start Jun 21, 2021 07:07:06 PM

Get Weather Jun 21, 2021 07:07:18 PM

- Debug Logs
- Logs
- Inputs
- Outputs

Get Forecast Jun 21, 2021 07:07:20 PM

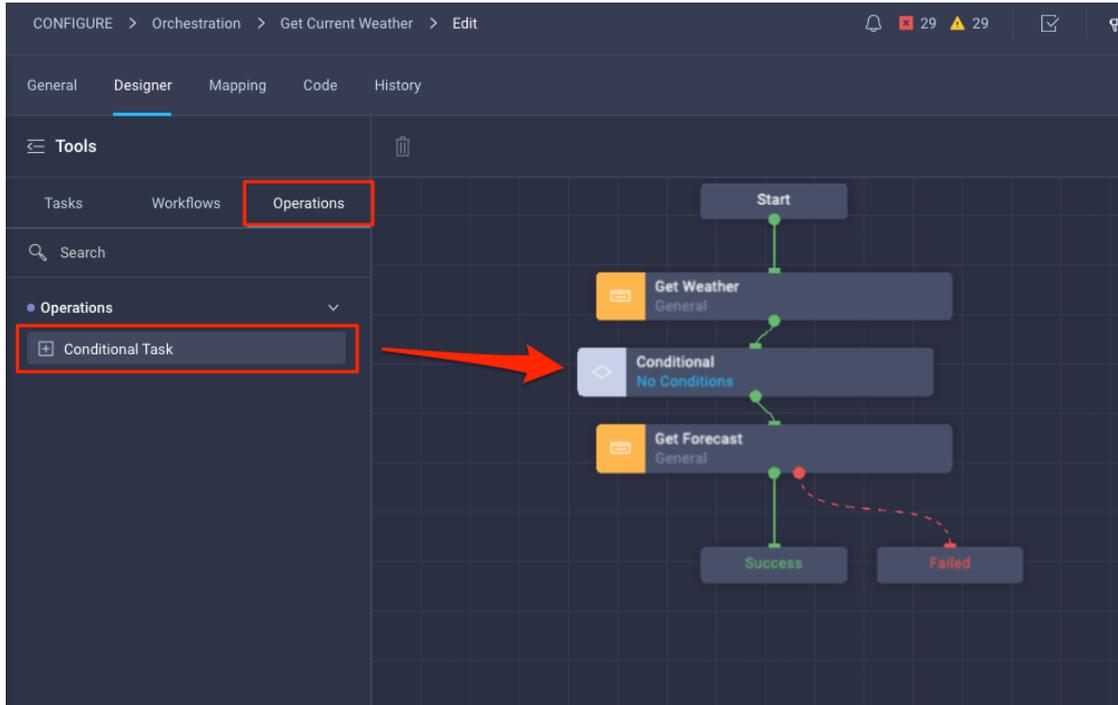
- Debug Logs
- Logs
- Inputs
- Outputs

Success Jun 21, 2021 07:07:21 PM

← Workflow Outputs now shows conditions, temperature as well as the humidity we got with the second task

Conditional Task

Using the Conditional Task



Scenario:

Get humidity value only if the temperature crosses a threshold

Drag the **Conditional Task** in the designer, between the 'Get Weather' and the 'Get Forecast' tasks

Using the Conditional Task

The screenshot shows a configuration window titled "Check Temp" with a close button (X) in the top right corner. It has two tabs: "General" and "Conditions". The "Conditions" tab is active, showing a "Condition *" field with the text: `if (${GetWeather1.output.temperature} > 20) 'high_temp'; els`. Below this is a "Cases" section with a table:

Value *	Description	
<u>high_temp</u>	<u>high temperature</u>	🗑️
Value *	Description	
<u>temp_ok</u>	<u>temp is fine</u>	🗑️ +

The conditional task requires two main inputs:

- The **Condition** we want to check
- The **Cases**, that is the possible outcomes

Conditional expressions support the following operators:

- Comparison operators such as `===` (Equal to), `!=` (Not equal to), `>` (Greater than), `<` (Less than), `>=` (Greater than or equal to), `<=` (Less than or equal to)
- Arithmetic operators such as `=`, `-`, `*` (Multiplication), `/` (division), `%` (Modulo), `**` (Logical AND)
- Logical operators such as `&&` (Logical AND), `||` (Logical OR), `!` (Logical NOT)
- Ternary operator such as `condition ? val1 : val2`

Using the Conditional Task

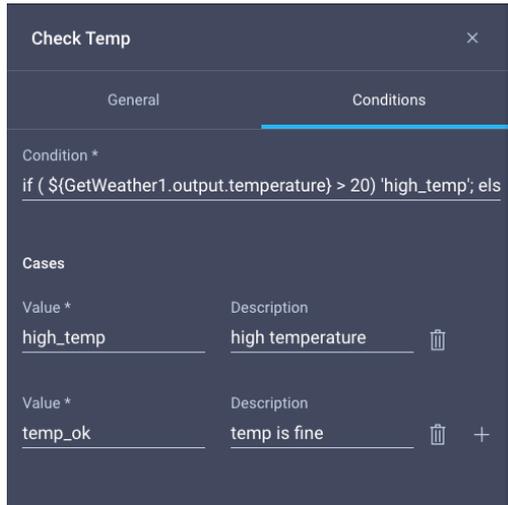
```
225 }
226 "Tasks": [
227 {
228   "Description": "",
229   "Label": "",
230   "Name": "StartTask",
231   "NextTask": "GetWeather1",
232   "ObjectType": "workflow.StartTask"
233 },
234 {
235   "CatalogMoid": "5c81de2c696f6e2d3028226c",
236   "Description": "Get Weather from OpenWeather",
237   "InputParameters": {
238     "Target": {
239       "Moid": "60ba98ac6f72612d31ddb1d0",
240       "ObjectType": "asset.Target"
241     },
242     "WeatherInputs": {
243       "CityName": "{{ .global.workflow.input.city }}",
244       "Country": "{{ .global.workflow.input.country }}",
245       "Units": "{{ .global.workflow.input.units }}"
246     }
247   },
248   "Label": "Get Weather",
249   "Name": "GetWeather1",
250   "ObjectType": "workflow.WorkerTask",
```

```
107 "InputParameterSet": [],
108 "Label": "Get Current Weather",
109 "Name": "GetCurrentWeather",
110 "OutputDefinition": {
111 {
112   "Default": {
113     "IsValueSet": false,
114     "ObjectType": "workflow.DefaultValue",
115     "Override": false,
116     "Value": null
117   },
```

In order to define the **Conditions**, you will need the unique **Name** of the task or workflow you want to run the condition on.

That can be accomplished by clicking on the **Code** tab in the designer

Using the Conditional Task



If you want to check a **Condition** against a workflow input, the object syntax is:

```
${workflow.input.<workflow input  
ReferenceName>}
```

If you want to check a **Condition** against a task output, the object syntax is:

```
${<Task Name>.output.<Task Output  
Name>}
```

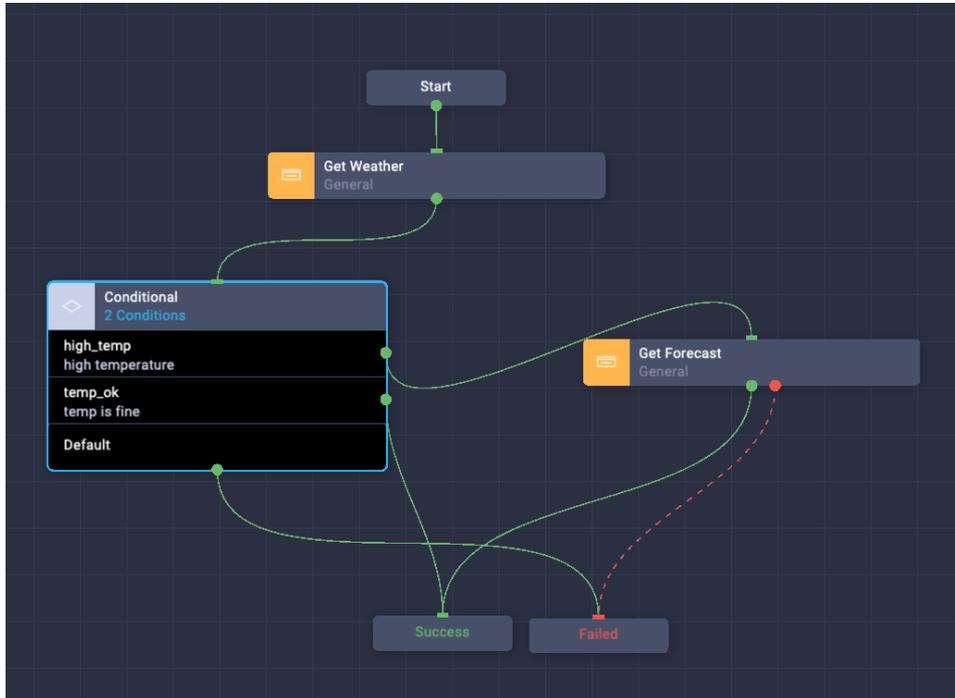
In this example, we want to check that the 'temperature' output of the task 'Get Weather' is *greater* (>) than 20 (degrees). If that is the case, we set a value of 'high_temp', otherwise we set a value of 'temp_ok'.

To set this condition, we use the following syntax:

```
if ( ${GetWeather1.output.temperature} > 20) 'high_temp'; else 'temp_ok'
```

In the **Cases**, we create two entries, one for each value we've set.

Using the Conditional Task



In the designer, the **Cases** we just configured will appear as blocks you can connect to other tasks based on the outcome you want or directly to the **Success** or **Failed** blocks.

In this example, as soon as we start the workflow, we will get the current weather conditions and temperature of the selected location. The conditional task will evaluate whether the temperature is high or ok and if the temperature is high, it will go to the 'Get Forecast' task that will return the relative humidity, otherwise it will exit.

Save the workflow and Execute

Save

Execute

Using the Conditional Task

The screenshot displays a workflow execution environment. On the left, a workflow diagram shows a 'Start' task leading to a 'Get Weather' task, which then branches into a 'Conditional' task. The 'Conditional' task has two paths: one leading to a 'Success' terminal and another leading to a 'Failed' terminal. On the right, the execution details for 'Get Current Weather - Today at 3:47 PM' are shown. The status is 'Success'. The workflow outputs are listed as follows:

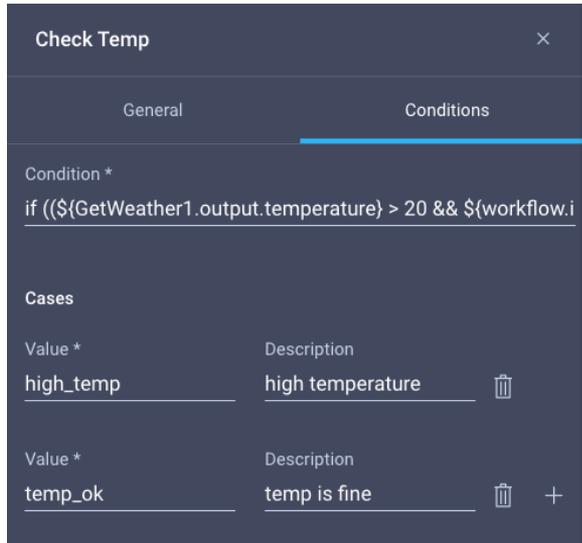
- Workflow Inputs: none
- Workflow Outputs:
 - conditions: Clear
 - temperature: 33.59
 - humidity: 45

The execution history shows the following steps:

- Start (Jun 22, 2021 03:47:59 PM)
- Get Weather (Jun 22, 2021 03:48:03 PM)
- Check Temp (Jun 22, 2021 03:48:04 PM) - This step is highlighted with a blue box and shows an output of 'high_temp'.
- Get Forecast (Jun 22, 2021 03:48:06 PM)
- Success (Jun 22, 2021 03:48:06 PM)

Since the 'temperature' returned from the 'Get Weather' task is greater than 20 (33.59 degrees), the workflow executes the 'Get Forecast' task and gets the relative 'humidity'

Compound Conditional Operations



The current implementation only works for the 'metric' unit as the **Condition** will always look at values greater than 20 degrees even if the 'temperature' is returned as 'imperial'. We want to implement a logic so we can also set a threshold for 'imperial' units.

The following expression will cover both cases, using the OR (||) operator:

```
if (($GetWeather1.output.temperature) > 20 &&
${workflow.input.units} === 'metric') ||
(${GetWeather1.output.temperature} > 68 &&
${workflow.input.units} === 'imperial')) 'high_temp'; else
'temp_ok'
```

In this example we set the threshold for 'imperial' as 68 degrees and 20 degrees for 'metric'.

Additional information and examples on conditional expression can be found here:

https://www.intersight.com/help/resources/Workflow_Designer#operations

Compound Conditional Operations

Workflow Inputs

- city: new york
- country: us
- units: imperial

Workflow Outputs

- conditions: Clouds
- temperature: 69.96
- humidity: 69

Start Jun 22, 2021 06:08:34 PM

1 Get Weather Jun 22, 2021 06:08:39 PM

- Debug Logs
- Logs
- Inputs
- Outputs

2 Check Temp Jun 22, 2021 06:08:39 PM

- Outputs

caseOutput: [1]

- high_temp

3 Get Forecast Jun 22, 2021 06:08:47 PM

- Debug Logs

New York, US, Imperial
Case output: high_temp – ‘Get Forecast’ triggered

Workflow Inputs

- city: san jose
- country: us
- units: imperial

Workflow Outputs

- conditions: Clouds
- temperature: 65.97

Start Jun 22, 2021 06:09:44 PM

1 Get Weather Jun 22, 2021 06:09:46 PM

- Debug Logs
- Logs
- Inputs
- Outputs

2 Check Temp Jun 22, 2021 06:09:46 PM

- Outputs

caseOutput: [1]

- temp_ok

Success Jun 22, 2021 06:09:47 PM

San Jose, US, Imperial
Case output: temp_ok – ‘Get Forecast’ skipped

Parallel Loops

Parallel Loops

Parallel Loops allow a user to execute a specific task multiple times in parallel.

A sample use case would be to provision multiple objects at once (i.e. a given number of Virtual Machines)

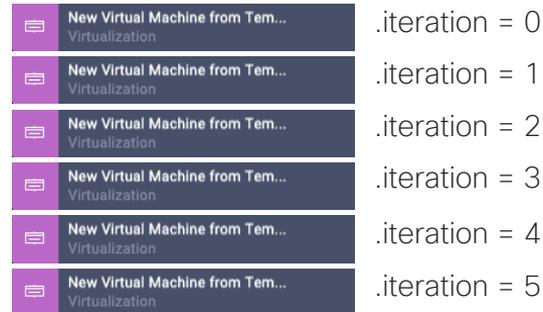
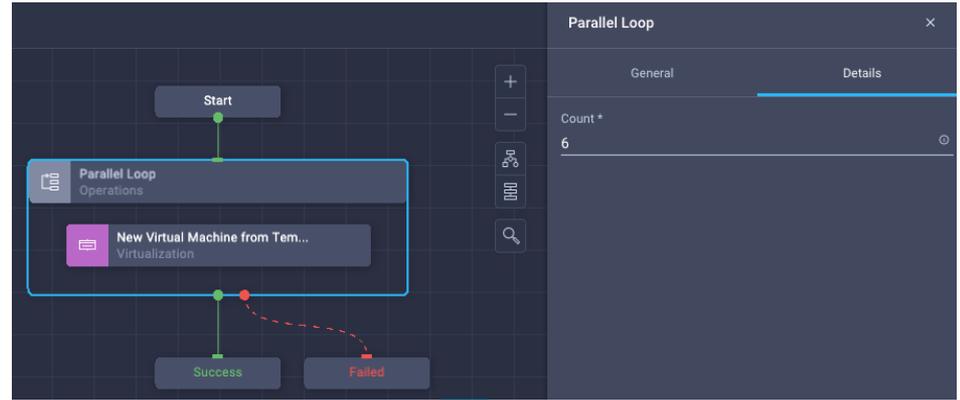
There are two parameters to consider:

1. **Count.** This parameter specifies the number of executions you want to run in parallel for the task you drag into the loop
2. **Iteration.** This parameter has significance only within the individual task execution and represent the current number of the task execution

Example:

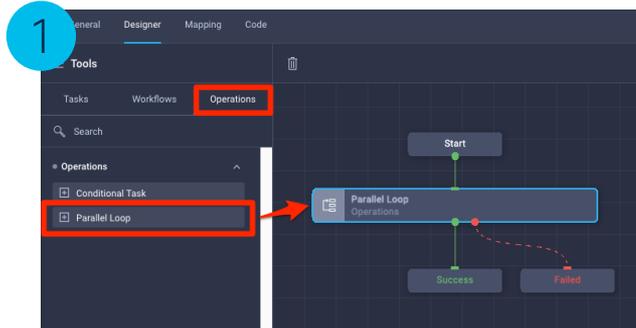
You specify a **Count** value of 10 and drag the 'New Virtual Machine from Template or Clone from Virtual Machine' task into the loop.

The loop will execute the task 10 times in parallel, meaning that 10 Virtual Machine will be created. Inside each task execution, the **.iteration** parameter will represent the task execution number

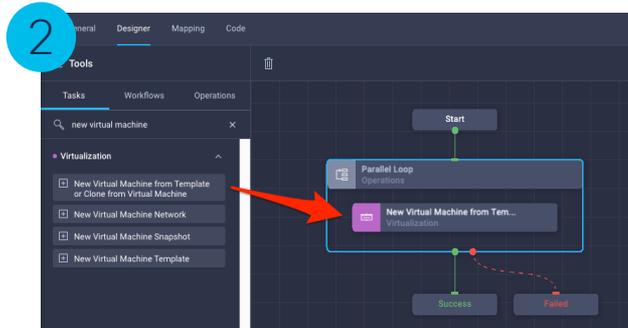


Example:
If you set the **Virtual Machine** name as `myvm_{{.iteration}}`
ICO will create 6 VM named: `myvm_0, myvm_1, myvm_2, ...`

Parallel Loops Example – Create Multiple VMs 1/2

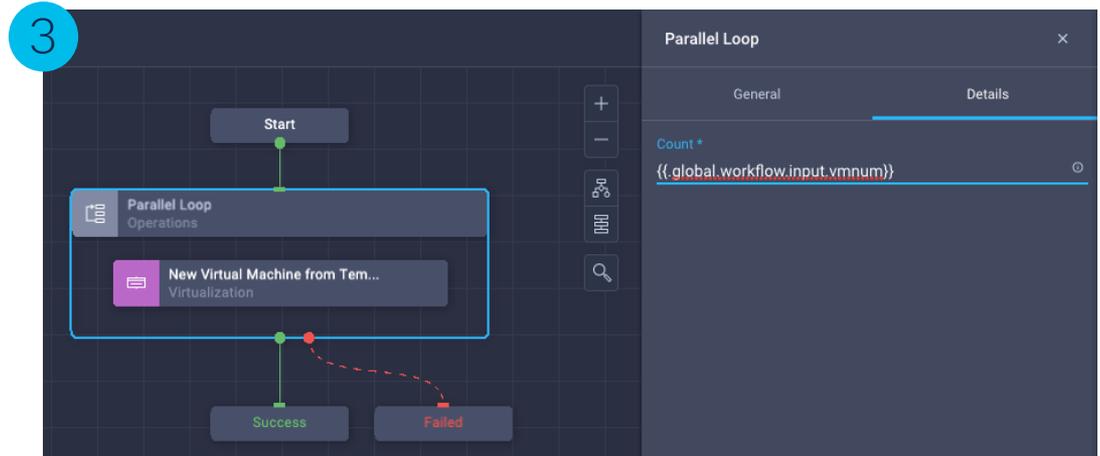


Drag the **Parallel Loop** task into the designer



Drag the task you want to execute multiple times in parallel inside the **Parallel Loop** task

task⁰ Cisco and/or its affiliates. All rights reserved. Cisco Confidential

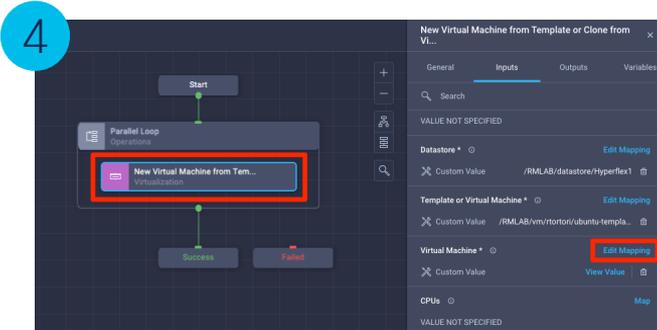


Click on the **Parallel Loop**, **Details** tab and specify a count value. This needs to be an integer, you can set it statically (i.e: 6) or use the templating syntax to set it dynamically or from a workflow input.

In this case, we want the user to specify the number of execution from a workflow input so we use the `{{.global.workflow.input.vmmnum}}` syntax.

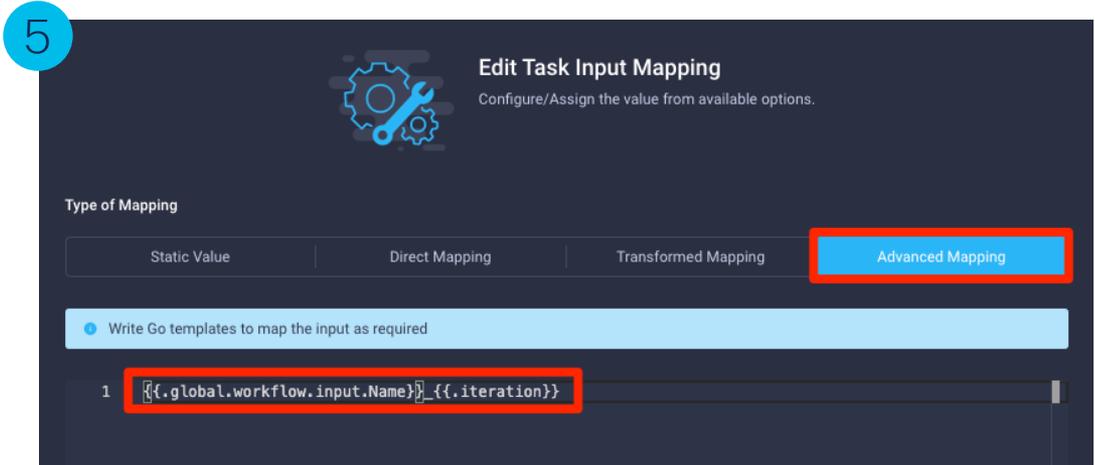
Note: this requires a WF input with reference name **vmname**, type **Integer**

Parallel Loops Example – Create Multiple VMs 2/2



In this example, we want to give a distinguished name for each VM, otherwise they will all have the same name. Click on the **New Virtual Machine from Template or Clone from Virtual Machine** task inside the **Parallel Loop** and map the **Virtual Machine** input

*Note: This example just covers the **Parallel Loop** use case, it won't cover how to set all required inputs to create a Virtual Machine.*



Use **Advanced Mapping** to set the Virtual Machine name:
`{{.global.workflow.input.Name}}``_{{.iteration}}`

Value from a Workflow Input with
reference name **Name**
Example: myvm

Iteration number
Example: 1

Sample Result: myvm_1

Parallel Loops Example – Create Multiple VMs from List

1

Update Workflow Input

Display Name *
Virtual Machine Name

Reference Name *
vmlist

Description

Value Restrictions

Required

Collection/Multiple

Min Size 1 Max Size 5

Type
String

Min 0 Max 0 Regex

Secure

Object Selector

Set Default Value

Cancel Save

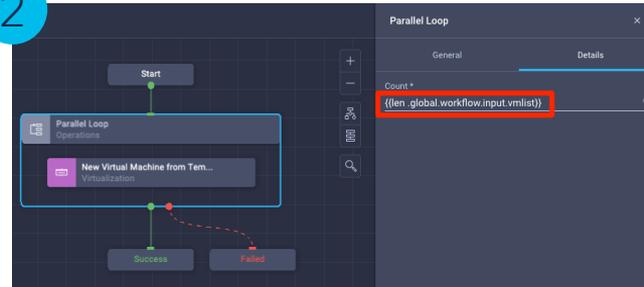
In this case, we have a WF Input with reference name **vmlist** defined as a collection of **String** (basically, an array).

The use case is to create n virtual machines, where n is the number of virtual machine names present in the collection.

As an example, we specified 1 and 5 as minimum and maximum number of items respectively.

This also means that we can only create max 5 VMs per each workflow execution

2



The **Count** value in the **Parallel Loop** can be set as follows:

```
{{len .global.workflow.input.vmlist}}
```

This means ICO will execute n instances of the inner task where n is the length of the list provided by the user

Note: the `len` function measure the length of a string (characters) or array (number of items).

For additional info please refer to the following documentation:

https://www.intersight.com/help/saas/resources/Workflow_Designer#operations_-_parallel_loop_task

Parallel Loops Example – Create Multiple VMs from List

3

Edit Task Input Mapping
Configure/Assign the value from available options.

Type of Mapping

Static Value Direct Mapping Transformed Mapping **Advanced Mapping**

Write Go templates to map the input as required

1 `{{index .global.workflow.input.vmlist .iteration}}`

For the Virtual Machine name input mapping we are now following a different approach:
As the user will specify a list of **Virtual Machine** names, we want to assign those exact names to each VM.

```
{{index .global.workflow.input.vmlist .iteration}}
```

The `index` function will extract the value of the `.iteration` element of the array `.global.workflow.input.vmlist` (Workflow Input)

Enter Workflow Input - Create Multiple Virtual Machines From List

Organization *
default

Workflow Instance Name
Create Multiple Virtual Machines from List

Virtual Machine Name *
production_vm

Virtual Machine Name *
test_vm

Virtual Machine Name *
development_vm

Cancel Execute

Executing the workflow, the user will set, as an example, 3 different **Virtual Machine** names which are creating our collection of elements.

In this case, **production_vm** will be element 0 (`.iteration = 0`), **test_vm** will be element 1 (`.iteration = 1`) and **development_vm** element 2 (`.iteration = 2`)

Example:

When the task with `.iteration = 2` will execute, the `index` function will extract from the array the value of element 2, in this case **development_vm**

Workflow Variables

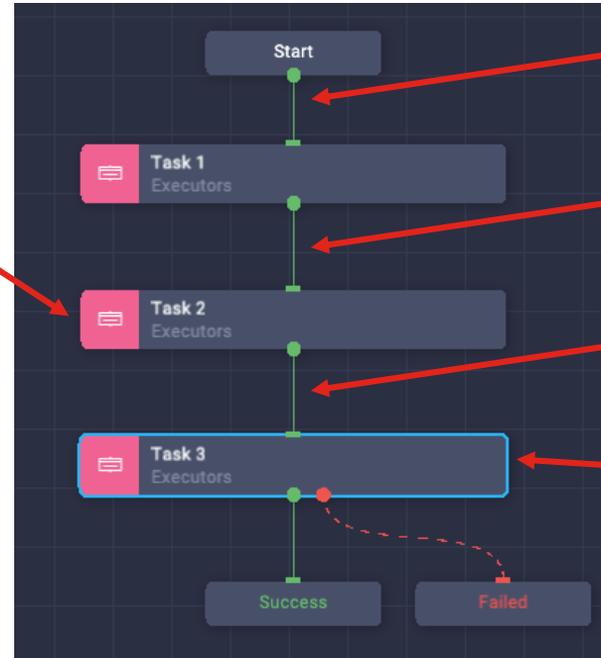
Workflow Variables

Workflow variables are similar to local variables within functions of a programming language. Workflow variables are scoped within a workflow. In a workflow, all tasks are bound to the scope of the workflow and can **read** or **update** the workflow variables that are defined for the workflow.

The update of a variable happens **after the task has been executed**, for instance **Task 2** executes and before it exits it updates the variable **mynumber**

Sample use cases:

- Simplify workflows using conditional branches. Multiple tasks can update (save a value) the same variable, this value can be then used by another task irrespective of the branch from which it was taken
- Transformation. Apply a transformation function and save the result in a variable without having to redo the transformation in every mapping. For instance, you turn a Virtual Machine name into all uppercase and this transformation will be available in all tasks requiring the Virtual Machine Name



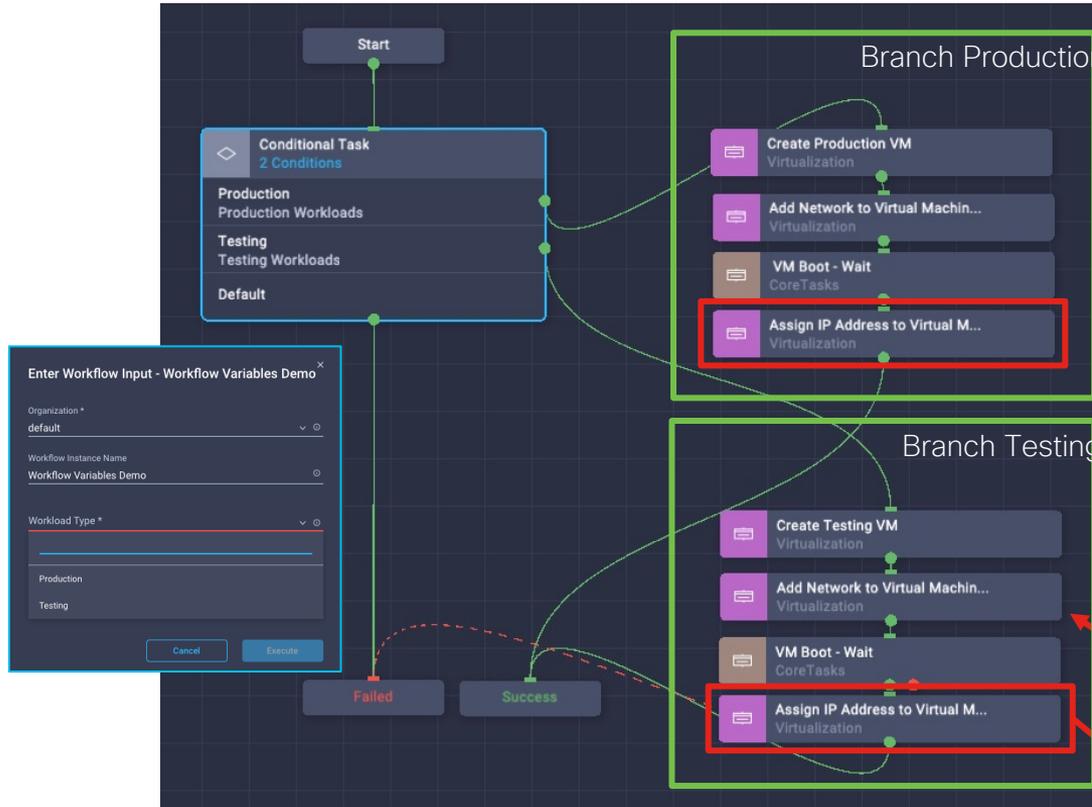
Variable mynumber:
Initial value: 10

Variable mynumber:
Task 1 updates value: 2

Variable mynumber:
Task 2 updates value: 3

Task 3 uses variable
Mynumber. Value: 30

Example of Workflow without Variables

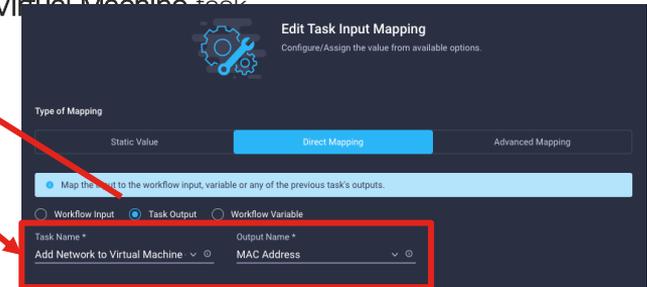


Consider this sample workflow.

There is a workflow input that allows the user to request a 'Production' VM or a 'Testing' VM. Based on this condition we split in two branches to create the VM with the required specs, add the right network to it and eventually assign an IP address.

As the **Assign IP Address to Virtual Machine** task requires the MAC address of the VM, there is no way to know it before the VM has been created.

Since we can't map two task outputs to a single input, we create the same task in the two branches and map the **Virtual NIC - Mac Address** input to the **MAC Address** output of the corresponding **Add Network to Virtual Machine** task.



Using Variables 1/3

The image consists of two side-by-side screenshots from a workflow editor. The left screenshot shows the 'General' tab of a workflow configuration. The 'Workflow Variables' section is highlighted with a red box, and the 'Add Workflow Variable' button at the bottom is also highlighted with a red box. A blue circle with the number '1' is positioned to the left of the screenshot. The right screenshot shows the 'Add Workflow Variable' dialog box. The 'Reference Name' field is set to 'MacAddress' and the 'Type' dropdown is set to 'String', both fields are highlighted with red boxes. A blue circle with the number '2' is positioned to the right of the dialog box.

In the **General** tab, click on **Workflow Variables** and **Add Workflow Variable**. Assign a **Name** and **Type**.

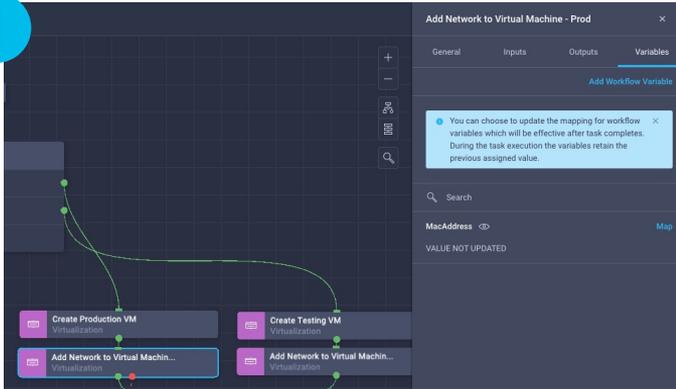
At the time of writing (March 2022) we support the following types: *String*, *Integer*, *Float*, *Boolean*, *Json*, *Enum*, *MoReference* and *Target Data Type*

You can optionally assign a value using the **Initial Mapping To** option. The variable will be created and can be mapped to any compatible input and updated by any task

The image shows a close-up of the 'Initial Mapping To' dialog box. It has a 'Default Value' field with a dropdown arrow on the right. Below it, there is a 'Default Values *' section with a text input field containing the value 'somevalue'.

Using Variables 2/3

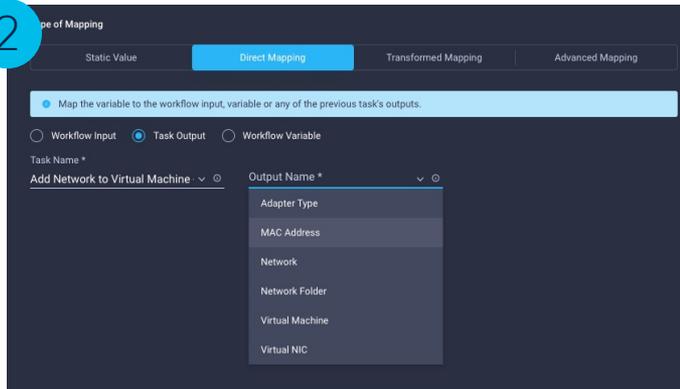
1



Select any task and click on the **Variables** tab. You will see the configured workflow variables. **If you want, you can also create Workflow variables from there.**

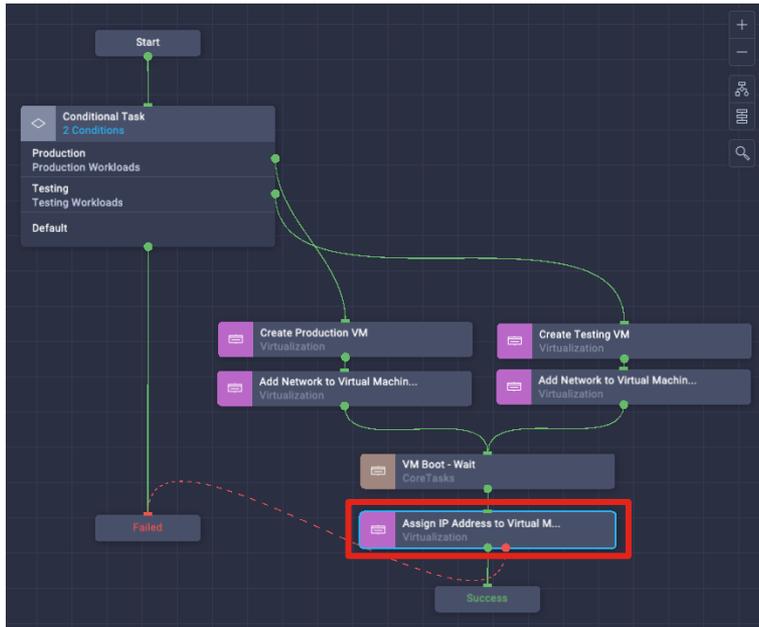
Clicking on **Map** you can assign a value using a **Static Value**, **Direct Mapping**, **Transformed Mapping** (if String) or **Advanced Mapping**. After the execution of the task the variable will have the configured value.

2

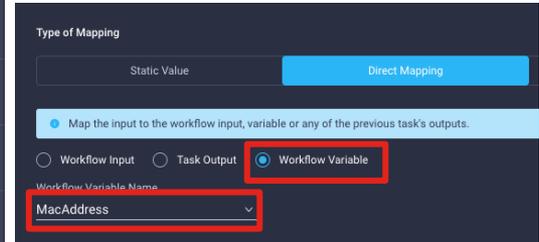
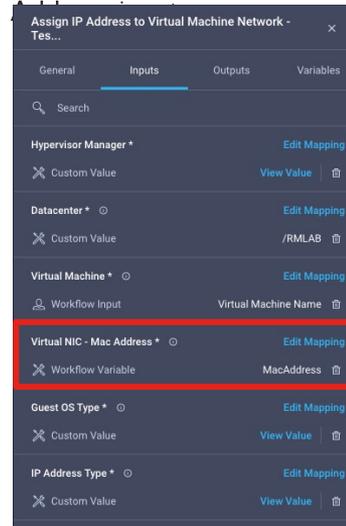


In our example, we will assign as a value the **Output MAC Address** coming from the **Add Network to Virtual Machine** task, which is actually itself. This means that once the task is completed, the variable will contain the MAC address we are looking for. For this specific case, we do the same operation on the other branch **Add Network to Virtual Machine** task.

Using Variables 3/3



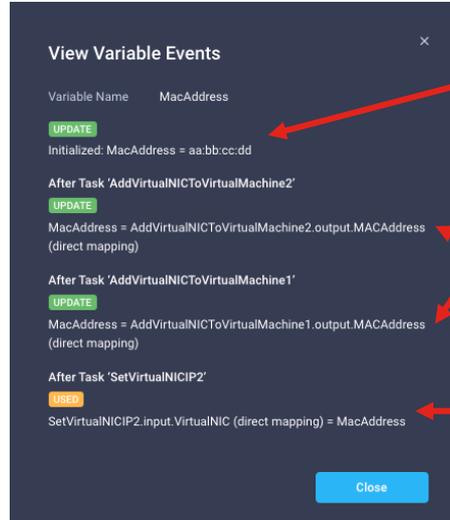
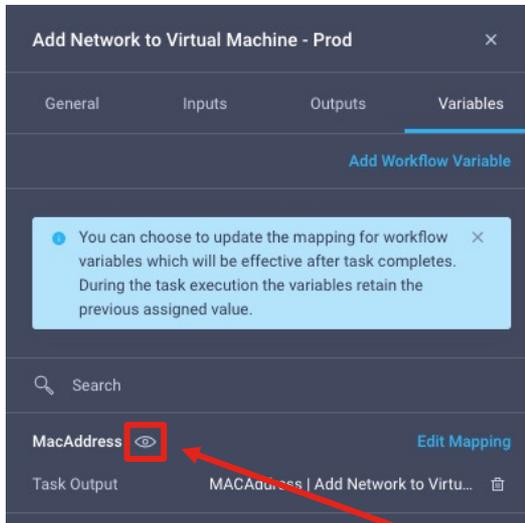
The refactored workflow will look something like this. We now have only one instance of **Assign IP Address to Virtual Machine**, which uses the variable **MacAddress** as the value for **Virtual NIC - Mac**



With this implementation, we simplified the workflow from 9 to 7 tasks, removing some redundancies.

The **Assign IP Address to Virtual Machine** task will get the MAC Address from the variable, which gets updated by whatever task is executed, independently on the conditional branch that was executed

Workflow Variable Events



The variable **MacAddress** has a default value of:
aa:bb:cc:dd

Tasks **AddVirtualNICtoVirtualization 1 and 2** (whatever gets executed based on the conditional task) will update the variable based on the MAC address they return as an output

Task **SetVirtualNICIP2** will use the variable (its value) using direct mapping

In the **Variables** tab of every task, clicking on the 'Eye' button, you can view the variable events, which represent the lifecycle of the variable from the beginning of the workflow, to the end.

You can also access variable events in the **General** tab of the workflow, under **Workflow Variables**

Serial Loops

Serial Loops

Serial Loops allow a user to execute one or more tasks in a loop.

You can't use Serial Loops, Parallel Loops or Conditional Tasks inside a Serial Loop.

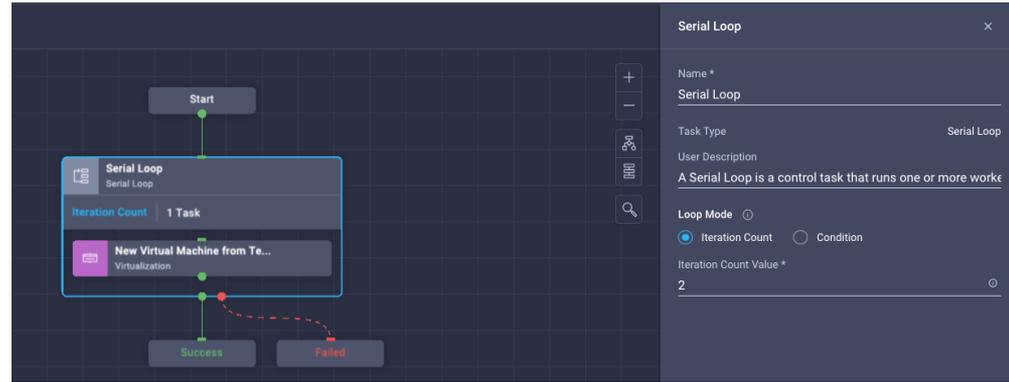
To prevent infinite loops, we enforce the limit of maximum 100 iterations. This number will be gradually relaxed in the future.

Serial Loops can be based on **Iteration Count** or **Condition** (mutually exclusive)

There are three parameters to consider:

1. **Iteration Count.** This parameter specifies the number of times you want to execute the tasks in the loop
2. **Condition.** This parameter specifies a condition for which the loop runs as long as the condition is true *
3. **Iteration.** This parameter has significance only within the individual task execution and represent the current

*The current serial loop implementation is do-while
Parallel Loops
This means that the first iteration will be **always** executed



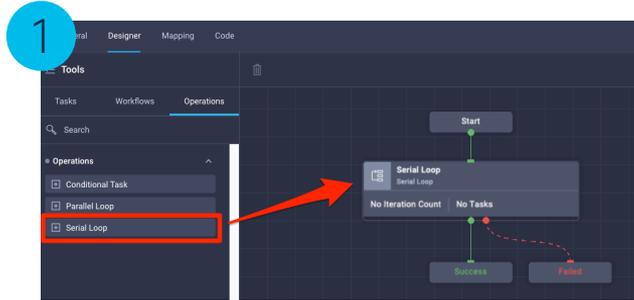
.iteration = 0

.iteration = 1

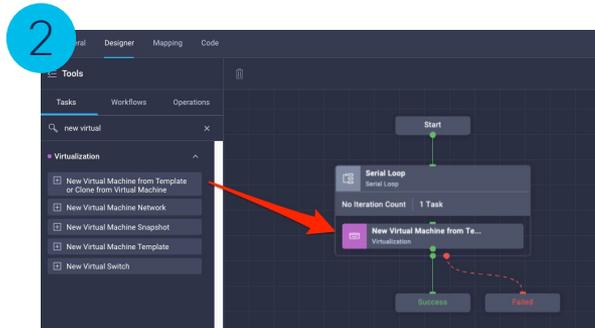
Example:
If you set the **Virtual Machine** name as
`myvm_{{.iteration}}`

ICO will create 2 VM in a serial fashion named:
`myvm_0, myvm_1, myvm2,`
...

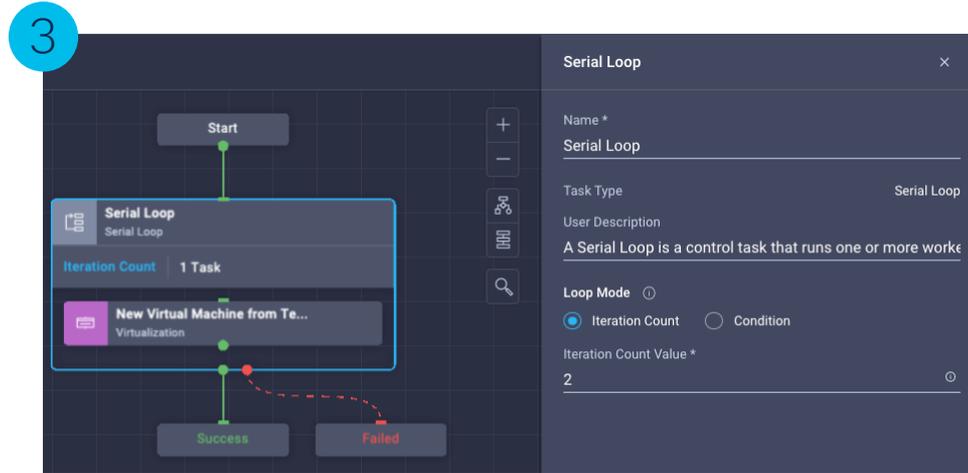
Serial Loops Example - Create Multiple VMs with Count 1/3



Drag the **Serial Loop** task into the designer



Drag one or more tasks **inside** the **Serial Loop** task



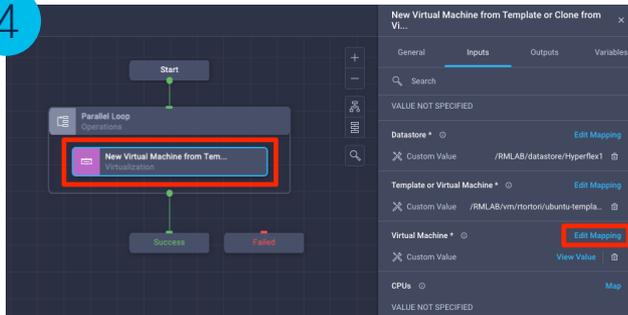
Click on the **Serial Loop** and specify a count value.

This needs to be an integer, you can set it statically (i.e: 2) or use the templating syntax to set it dynamically or from a workflow input as you would do in a **Parallel Loop**

In this case, we want the user to specify the number of execution statically using an integer.

Serial Loops Example – Create Multiple VMs with Count 2/3

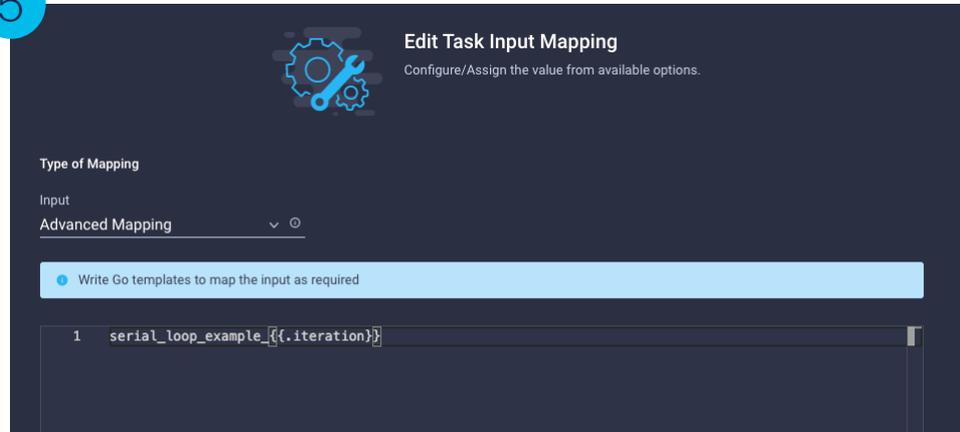
4



In this example, we want to give a distinguished name for each VM, otherwise they will all have the same name. Click on the **New Virtual Machine from Template or Clone from Virtual Machine** task inside the **Serial Loop** and map the **Virtual Machine** input

*Note: This example just covers the **Serial Loop** use case, it won't cover how to set all required inputs to create a Virtual Machine.*

5



Use **Advanced Mapping** to set the Virtual Machine name:
serial_loop_example_{{.iteration}}

Static Naming Prefix

Iteration number

Example: 1

Sample Result: serial_loop_example_1

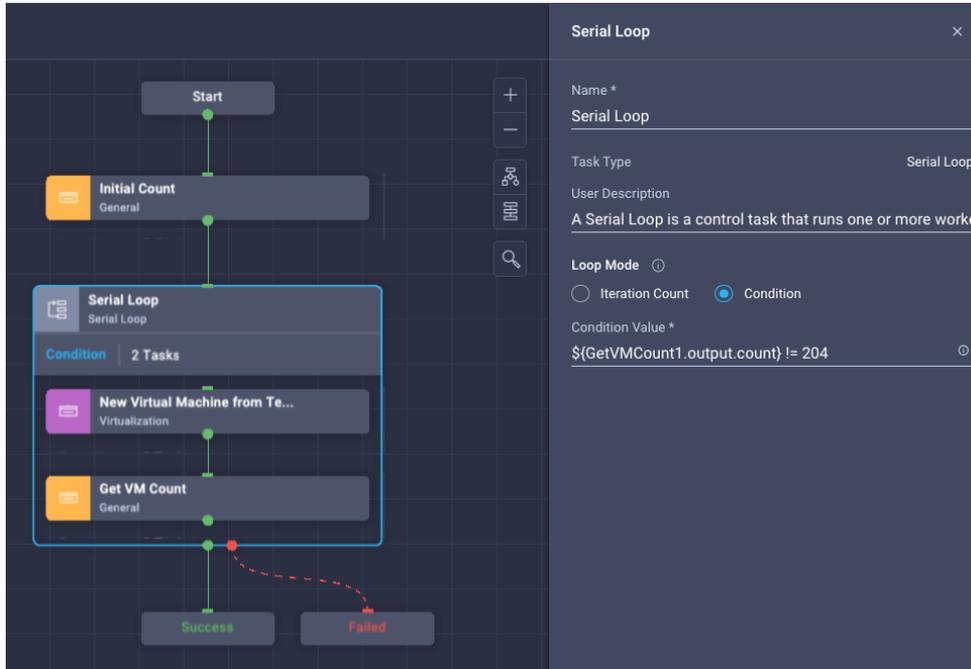
Serial Loops Example – Create Multiple VMs with Count 3/3

The screenshot displays a workflow execution interface. On the left, a workflow diagram shows a 'Start' node leading to a 'Serial Loop' node, which contains a task 'New Virtual Machine from Te...'. Below the diagram are 'Success' and 'Failed' status indicators. The main panel shows the execution details for 'Serial Loop Demo 2 - Apr 21, 2022 1:06 PM'. The status is 'Success'. The execution timeline includes:

- Start: Apr 21, 2022 01:06:09 PM
- Serial Loop: Apr 21, 2022 01:19:46 PM
 - Iteration 1: Apr 21, 2022 01:19:46 PM
 - New Virtual Machine from Template o...: Apr 21, 2022 01:19:46 PM
 - Logs
 - Inputs
 - Outputs
 - Iteration 2: Apr 21, 2022 01:19:46 PM
 - New Virtual Machine from Template o...: Apr 21, 2022 01:19:46 PM
 - Logs
 - Inputs
 - Outputs
- Success: Apr 21, 2022 01:10:14 PM

Iteration Count Value = 2

Serial Loops Example – Create Multiple VMs with Condition



In this example, we don't know in advance how many VMs to create.

We are using a custom task that returns the VM count from the target vCenter. For this specific case, we will start from 202, which is the current number of VMs present.

Note that to prevent infinite loops we currently enforce the limit of max 100 iterations.

We set a **Condition** that creates VMs until the total number of VMs in the target vCenter is 204

For this sample scenario, we drag the count task **within** the serial loop, so the loop will execute the VM creation and the count tasks.

The first iteration will always run, however at the end of each iteration, we check the output of the **Get VM Count** task. If that is different than 204, we keep running, otherwise we exit the loop.

Serial Loops Example – Create Multiple VMs with Condition Execution

The screenshot displays a workflow execution interface. On the left, a workflow diagram shows a sequence of tasks: 'Start', 'Initial Count', 'Serial Loop', 'New Virtual Machine from Te...', and 'Get VM Count'. The 'Initial Count' task is highlighted with a red box. The 'Serial Loop' task is also highlighted and shows a condition of '2 Tasks'. Below the diagram, the execution progress is shown with 'Success' and 'Failed' indicators.

The main panel shows the execution details for 'Serial Loop Demo 2 - Apr 21, 2022 3:07 PM'. The status is 'Success'. The execution timeline shows the following steps:

- Start: Apr 21, 2022 03:07:17 PM
- Initial Count: Apr 21, 2022 03:07:49 PM
- Serial Loop: Apr 21, 2022 05:16:41 PM
- Iteration 1: Apr 21, 2022 05:16:41 PM
- New Virtual Machine from Template o...: Apr 21, 2022 05:16:41 PM

The 'Initial Count' task output is shown in a red box, displaying the following configuration results:

```
ConfigResults: [ 1 ]
  Object: ( 3 )
    ConfigResCtx: ( 1 )
      EntityData: ( 1 )
        task: workflow.PowerShellApiTask
    State: Ok
    Type: Config
  Total VMs: 202
```

For reference, we run an **Initial Count** outside the loop, which returns 'Total VMs = 202'

Serial Loops Example – Create Multiple VMs with Condition Execution

The screenshot displays a workflow execution interface. On the left, a workflow diagram shows a 'Start' task leading to an 'Initial Count' task, followed by a 'Serial Loop' task. The 'Serial Loop' task is highlighted with a red box and contains two sub-tasks: 'New Virtual Machine from Te...' and 'Get VM Count'. Below the diagram are 'Success' and 'Failed' outcome buttons. On the right, the execution details for the 'Serial Loop' are shown, including a tree view of tasks. The 'Iteration 1' task is highlighted with a red box. Below it, the 'Outputs' section shows a 'ConfigResults' object with a 'Total VMs: 203' value highlighted in a red box.

```
graph TD; Start[Start] --> Init[Initial Count]; Init --> Loop[Serial Loop]; subgraph Loop; direction TB; VM[New Virtual Machine from Te...]; Count[Get VM Count]; end; Loop --> Success[Success]; Loop --> Failed[Failed];
```

Execution: Serial Loop Demo 2 - Apr 21, 2022 3:07 PM

Organization: default

Status: Success

Serial Loop (Apr 21, 2022 07:16:35 PM)

- Iteration 1 (Apr 21, 2022 07:16:35 PM)
 - New Virtual Machine from Template o...
 - Logs
 - Inputs
 - Outputs
 - Get VM Count
 - Logs
 - Inputs
 - Outputs

ConfigResults: [1]

- Object: (3)
 - ConfigResCtx: (1)
 - EntityData: (1)
 - task: workflow.PowerShellApiTask
 - State: Ok
 - Type: Config

Total VMs: 203

After the first iteration, we count 203 Total VMs

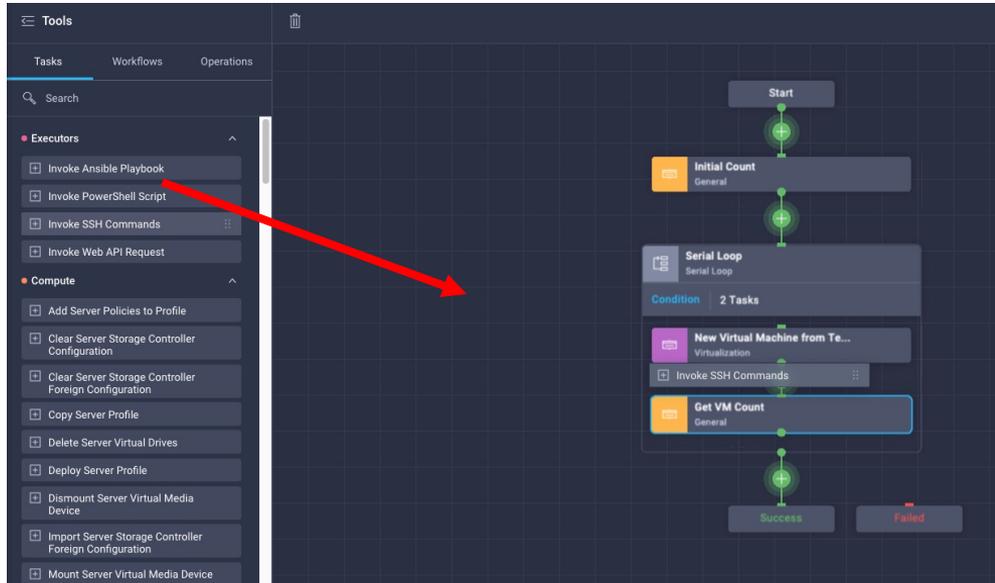
Serial Loops Example – Create Multiple VMs with Condition Execution

The screenshot displays a workflow execution interface. On the left, a task graph shows a 'Serial Loop' containing two tasks: 'New Virtual Machine from Te...' and 'Get VM Count'. The 'Serial Loop' task is highlighted with a red box. The main execution console shows the following details:

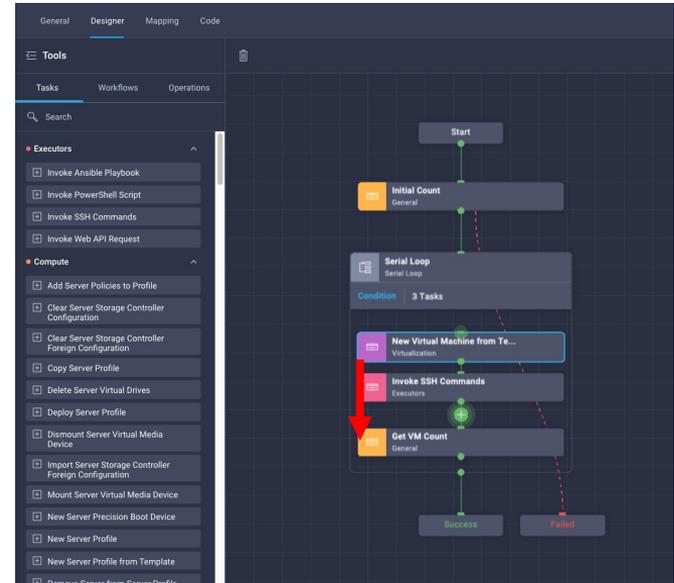
- Execution: Serial Loop Demo 2 - Apr 21, 2022 3:07 PM
- Organization: default
- Status: Success
- Iteration 1: Apr 21, 2022 07:16:35 PM
- Iteration 2: Apr 21, 2022 07:16:35 PM (highlighted with a red box)
- Task 1: New Virtual Machine from Template o... (highlighted with a red box)
- Task 2: Get VM Count
- ConfigResults: [1]
 - Object: (3)
 - ConfigResCtx: (1)
 - EntityData: (1)
 - task: workflow.PowerShellApiTask
 - State: Ok
 - Type: Config
 - Total VMs: 204 (highlighted with a red box)
- Final Status: Success (highlighted with a red box) - Apr 21, 2022 03:13:40 PM

After the second iteration, we count 204 Total VMs, so we stop iterating and exit the loop

Add Multiple Tasks to the Loop and Reordering



Drag a task and drop it in one of the drop zones within the loop box



Drag a task and drop it in one of the drop zones within the loop box to rearrange the task order

Workflow Versions

Managing Workflow Versions

The screenshot shows the Cisco Intersight Orchestration interface. The left sidebar is under the 'CONFIGURE' section, with 'Orchestration' selected. The main area shows a list of workflows under the 'All Workflows' tab. The workflow 'Get Current Weather' is highlighted with a red box. A context menu is open for this workflow, with 'Manage Versions' highlighted by a red box. A red arrow points to the context menu icon (three dots) in the table row.

Display Name	Description	Default Version	Executions	Last Execution Status	Last Execution Time	Validation Status	Last Update	Organization	
Get Current Weather	Sample Workflow to...	1	34	Success	12 minutes ago	Valid	17 minutes ago	default	...
		1	0		-	Valid	23 minutes ago	-	Clone
		1	2	Failed	Jun 17, 2021 9:14 PM	Valid	Jun 17, 2021 9:11 PM	default	Execute
		1	34	Success	Jun 18, 2021 2:43 PM	Valid	Jun 16, 2021 6:18 PM	default	History
		3	0		-	Valid	Jun 11, 2021 3:10 AM	-	Manage Versions
		3	0		-	Valid	Jun 11, 2021 3:10 AM	-	Delete
		1	0		-	Valid	Jun 11, 2021 3:10 AM	-	...
		5	0		-	Valid	Jun 11, 2021 3:10 AM	-	...
		2	0		-	Valid	Jun 11, 2021 3:10 AM	-	...
		3	0		-	Valid	Jun 11, 2021 3:10 AM	-	...

Managing Workflow Versions

Manage Versions
You can create a new workflow version, delete an existing version, and change the default version

Create a New Version

Version	Validation Informat...	Executions	Last Execution Status	Description	Last Update
1 (default)	✓	34	✓	Sample Workflow to get ...	20 minutes ago

Create A New Version

Source Version *
1

Version *
2

Description
New Version of the Workflow

Set as Default Version

Cancel Create

Version	Validation Informat...	Executions	Last Execution Status	Description	Last Update
2 (default)	✓	0	✓	New Version of the Work...	a few seconds ago
1	✓	34	✓	Sample Workflow to get ...	a few seconds ago

Execute
Set as Default
Delete

If **Set as Default Version** is checked, all execution will use version 2 if no version is specified.
You can execute specific version of the workflow or set any existing version as the default

Transformations

Input Transformations

External Target	Custom Value	Task Output	Workflow Input
External Target *	OpenWeather HTTPEndpoint	latitude Get Weather	units
latitude *		latitude Get Weather	
longitude *		longitude Get Weather	
units *			units

Static Value **Direct Mapping** Transformed Mapping Advanced Mapping

Map the input to the workflow input or any of the previous task's outputs.

Workflow Input Task Output

Input Name *
units

Scenario:

The input 'units' for the task 'Get Forecast' is currently set as a **Direct Mapping** to the 'units' workflow input.

We want to manipulate this input before it get processed by the 'Get Forecast' task.

To do so, we need to use **Trasformed Mapping**

Input Transformations

The screenshot shows a configuration interface for 'Transformed Mapping'. At the top, there are four tabs: 'Static Value', 'Direct Mapping', 'Transformed Mapping' (which is selected and highlighted in blue), and 'Advanced Mapping'. Below the tabs is a light blue banner with a blue circle icon and the text: 'Transform the required data from workflow input/s or previous task's outputs, or custom value in to an expected format through'. Underneath is a section titled 'Transformation Stage'. It contains a blue button labeled 'Add Transformation Stage'. Below that is a form for a 'New Stage' with a close icon. The form has two main fields: 'Name *' with the value 'AllUpperCase' and 'Transformation Function *' with a dropdown menu. The dropdown menu is open, showing a search bar and a list of options: 'Atob', 'Atof', 'Atoi', and 'Btoa'. A 'Preview' button is visible on the left side of the dropdown menu.

You can add one or multiple **Transformation Stages.**

Arbitrary Name - less than 64 characters and can only contains letters (a-z,A-Z) or numbers

Select a **Transformation Function**

Input Transformations

The screenshot shows the configuration for the 'AllUpperCase' transformation function. The 'Name' field is set to 'AllUpperCase'. The 'Transformation Function' dropdown is set to 'UpperCase'. The 'StringInput' dropdown is open, showing a list of workflow inputs. The input '.global.workflow.input.units' is highlighted with a red box.

Name *

AllUpperCase

Transformation Function *

UpperCase

StringInput *

- Custom value
- .global.workflow.input.city
- .global.workflow.input.country
- .global.workflow.input.units**
- .global.Get Weather.output.conditions
- .global.Get Weather.output.temperature
- .global.Get Weather.output.payload
- .global.Get Weather.output.latitude
- .global.Get Weather.output.longitude

Select the **UpperCase** transformation function and the **StringInput**. This could be either one of the input/outputs or a **Custom Value**.

The screenshot shows the 'Preview (Read Only)' section with a 'Refresh' button and a Go Template code block. Below it is the 'Test Mode' section with a 'StringInput' field containing 'some string' and a 'Test' button. At the bottom is the 'Data Transformation Result' section showing the output of the transformation.

Preview (Read Only)

Refresh

```
1 {{- /* Auto-generated by transformed mapping section. DO NOT EDIT. */ -}}
2 {{- $AllUpperCase := UpperCase .global.workflow.input.units -}}
3 {{- $AllUpperCase -}}
```

Test Mode

StringInput *

some string

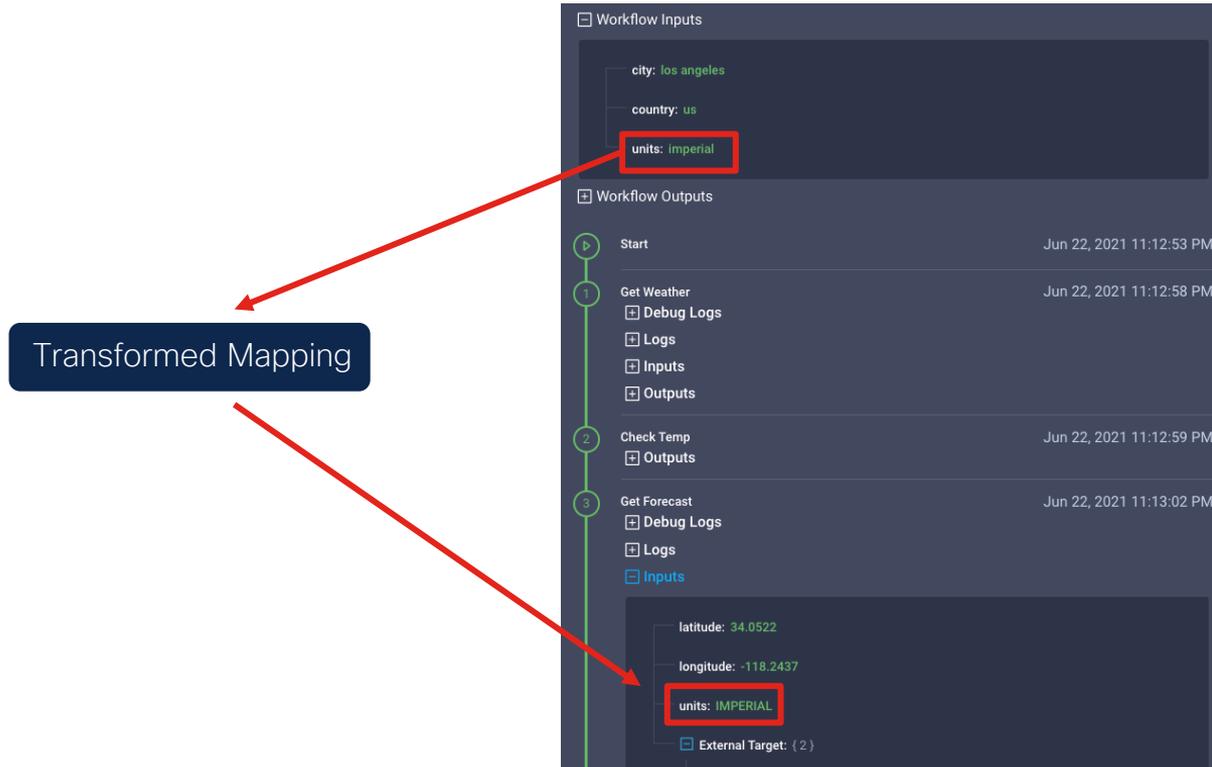
Test

Data Transformation Result

```
1 {"AllUpperCase": "SOME STRING"}
```

In this example, we picked the 'units' workflow input (.global.workflow.input.units) You can **Test** the results in real time and **Preview** the corresponding Go Template

Verify Input Transformation



Transformation Functions 1/2

FindAllString returns a slice of all substrings that match given regular expression in given string.

TrimSpace returns a slice of the string s, with all leading and trailing white space removed, as defined by Unicode.

ContainsString reports whether substr is within s.

Atoi is equivalent to `ParseInt(s, 10, 0)`, converts provided string to `Type int`.

StringReplace returns a copy of the string s with the first n non-overlapping instances of old replaced by new. If old is empty, it matches at the beginning of the string and after each UTF-8 sequence, yielding up to k+1 replacements for a k-rune string. If n < 0, there is no limit on the number of replacements.

UpperCase returns s with all Unicode letters mapped to their upper case.

LowerCase returns s with all Unicode letters mapped to their lower case.

ToJson returns the json format of given value that can be a number, slice, object and any goolang `Type` that can be marshalled into json.

Ftoa returns the given value as string with given precision. By default goolang uses the E-notation for large float64 values. This function allows the float64 numbers to be replaced with proper floating point notation.

Itoa converts the given int value in base 10 as string.

Atof converts the given string to a floating point number.

Transformation Functions 2/2

Ftoi converts the given float to an integer.

Itof converts the integer to a floating point number.

IntToBool converts 0 and 1 to false and true respectively.

BoolToInt converts true and false to 1 and 0 respectively.

Atob converts string values of true and false to boolean.

Btoa Converts boolean value to a string.

Substring returns the substring within the given index bounds of the given string.

IpToCidr retrieve CIDR notation for Ipv4.

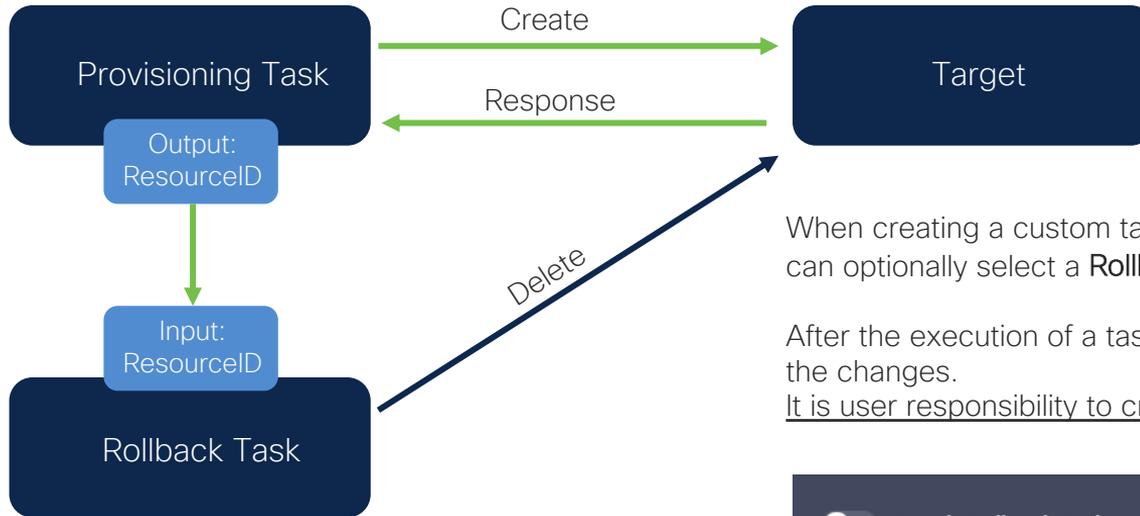
CalculateDate takes a datetime input in string, and returns a new datetime in standard RFC3339 format with an adjustment in seconds based on a time.Duration input. Both positive and negative increments can be handled.

CalculateDateDifference takes two datetime inputs as strings, a base date and a date to be adjusted against, and returns the difference in duration between the two in seconds. It can return a positive or negative value, depending on the relation between the two datetime values.

FormatDate takes two datetime inputs, one being the input date and another containing the reference layout string to which the input date has to be converted. All reference datetimes in Go follow the same format with an error being

Rollback Tasks (Task Designer)

Concept



When creating a custom task using the **Web API Request** executor, you can optionally select a **Rollback Task**.

After the execution of a task, you will be given the opportunity to rollback the changes.

It is user responsibility to create a working logic to rollback a custom task



Example Scenario

Scenario:

Claim an HTTP endpoint in Intersight using the API and rollback execution

Web API Request Target:

Intersight

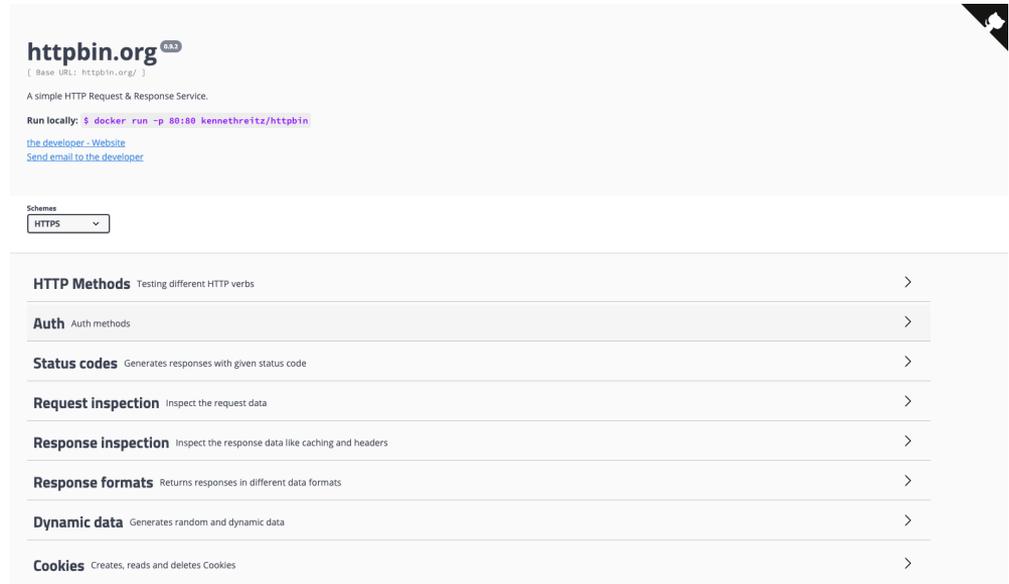
HTTP Target:

Endpoint: <https://httpbin.org>

Name: httpbin

Unauthenticated

Direct connection (no Intersight Assist)



The screenshot shows the httpbin.org website. At the top, it says "httpbin.org 0.0.0" with a subtext "[Base URL: httpbin.org/]". Below that, it describes it as "A simple HTTP Request & Response Service." and provides a command to run locally: "Run locally: \$ docker run -p 80:80 kennethreitz/httpbin". There are also links for "the developer - Website" and "Send email to the developer". A "Schemes" dropdown menu is set to "HTTPS". Below this is a list of services with right-pointing chevrons:

- HTTP Methods** Testing different HTTP verbs >
- Auth** Auth methods >
- Status codes** Generates responses with given status code >
- Request inspection** Inspect the request data >
- Response inspection** Inspect the response data like caching and headers >
- Response formats** Returns responses in different data formats >
- Dynamic data** Generates random and dynamic data >
- Cookies** Creates, reads and deletes Cookies >

REST API Call to Claim a Target in Intersight

Method: POST

Path: /api/v1/asset/Targets

JSON Payload:

```
{
  "Connections": [
    {
      "ManagementAddress": "httpbin.org",
      "Port": 443,
      "IsSecure": true,
      "Credential": {
        "ObjectType":
"asset.NoAuthenticationCredential"
      },
      "ObjectType": "asset.HttpConnection"
    }
  ],
  "TargetType": "HTTPEndpoint",
  "ManagementLocation": "Intersight",
  "Name": "httpbin"
}
```

Sample Response (partial information shown)

```
{
  "Moid": "60d25b3e6f72612d316c836f",
  "ObjectType": "asset.Target",
  "ClassId": "asset.Target",
  [...]
  "Connections": [
    {
      "ClassId": "",
      "ObjectType": "asset.HttpConnection",
      "Credential": {
        "ClassId":
"asset.NoAuthenticationCredential",
        "ObjectType":
"asset.NoAuthenticationCredential"
      },
      [...]
      "Assist": null,
      "RegisteredDevice": null
    }
  ]
}
```

REST API Call to Unclaim a Target in Intersight

Method: DELETE

Path: /api/v1/asset/Targets/<Target MOID>

No Response provided

Creating the HTTP Endpoint Claim Task

Task Properties

Organization *
default

Task Name *
Claim HTTP Target

Description
Claim an HTTP target in Intersight

Retry Count
3

Retry Delay
60

Timeout
600

Set Tags

Task Properties - General
Name: Claim HTTP Target

Task Properties - Inputs
name - String, Required
endpoint - String, Required

General **Inputs** Outputs

name * ↑ ↓ ✎ 🗑

endpoint * ↑ ↓ ✎ 🗑

Task Properties - Outputs
moid - String, Map to Task Output: 'Parameters | Invoke Web API Request'

General Inputs **Outputs**

moid ✎ 🗑

Map to Task Output

moid

Task Output
Parameters | Invoke Web API Request

Creating the HTTP Endpoint Claim Task

Executor Properties - General

Name: Claim HTTP Target

Set External Target: NO (will use Intersight as API target)

Executor Properties - Inputs

Method: POST

URL: Custom Value - /api/v1/asset/Targets

Body (Content Type: JSON):

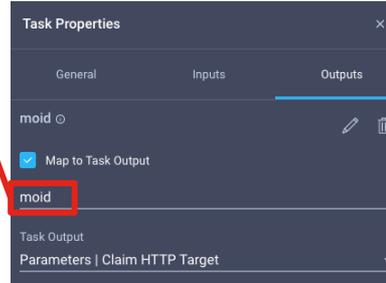
```
{
  "Connections": [
    {
      "ManagementAddress": "{{.global.task.input.endpoint}}",
      "Port": 443,
      "IsSecure": true,
      "Credential": {
        "ObjectType": "asset.NoAuthenticationCredential"
      },
      "ObjectType": "asset.HttpConnection"
    }
  ],
  "TargetType": "HTTPEndpoint",
  "ManagementLocation": "Intersight",
  "Name": "{{.global.task.input.name}}"
}
```

Response Parser Parameters:

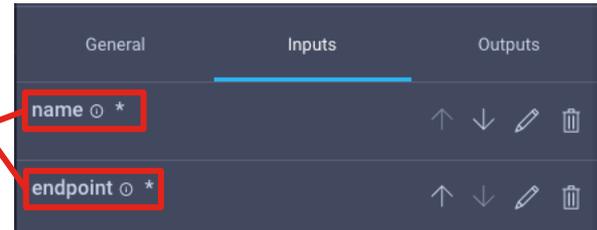
Path: \$.Moid

Name: moid

Type: String

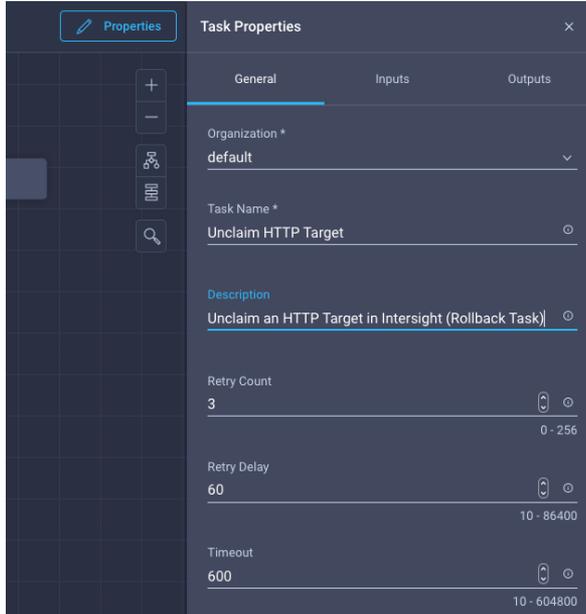


We extract the Moid from the response and output it as this will be used as an input for the rollback task



Using templates in the Body will get those values replaced by the Task Input values

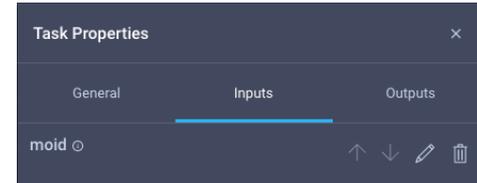
Creating the HTTP Endpoint Rollback Task



Task Properties - General
Name: Unclaim HTTP Target

Task Properties - Inputs
moid - String, Required

Task Properties - Outputs
None



Creating the HTTP Endpoint Rollback Task

Executor Properties - General

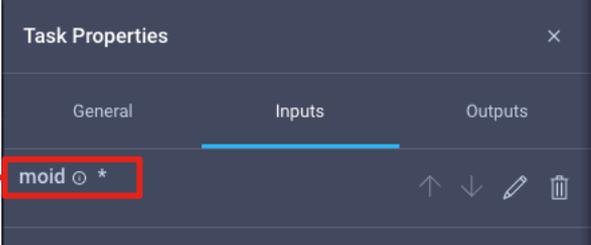
Name: Unclaim HTTP Target

Set External Target: NO (will use Intersight as API target)

Executor Properties - Inputs

Method: DELETE

URL: Custom Value - /api/v1/asset/Targets/{{.global.task.input.moid}}

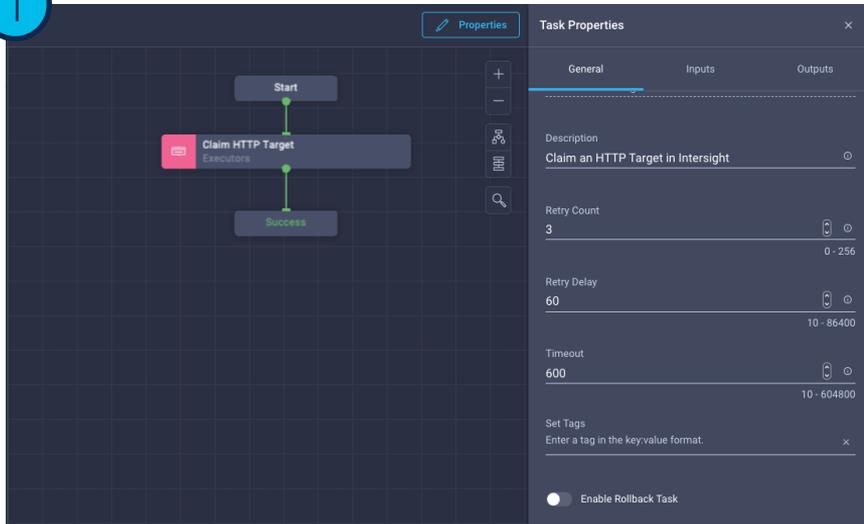


The screenshot shows a 'Task Properties' dialog box with three tabs: 'General', 'Inputs', and 'Outputs'. The 'Inputs' tab is active. A list of inputs is shown, with 'moid' selected and highlighted by a red box. To the right of the 'moid' input are icons for up/down arrows, edit, and delete. A red line connects the 'moid' input to the placeholder in the URL above.

Part of the URL will be replaced with the value of the 'moid' task input

Link the Rollback Task to the Claim Task

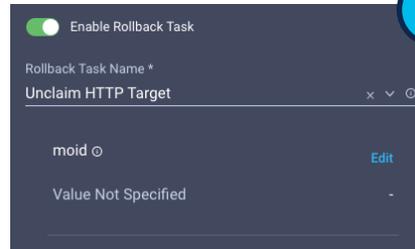
1



The screenshot shows a workflow editor with a 'Claim HTTP Target' task selected. The 'Task Properties' panel is open, showing the 'General' tab. The 'Enable Rollback Task' checkbox is currently unchecked. The task description is 'Claim an HTTP Target in Intersight'. Other settings include a Retry Count of 3, a Retry Delay of 60, and a Timeout of 600. The 'Set Tags' field is empty.

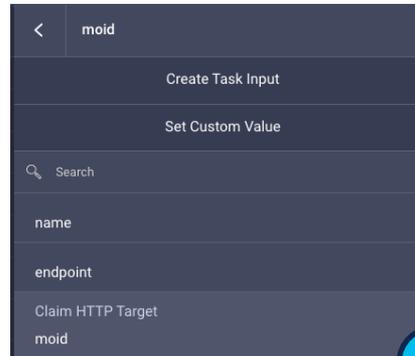
Back in the Claim task, in the **Task Properties**,
Enable Rollback Task

2



The dialog shows the 'Enable Rollback Task' checkbox is checked. The 'Rollback Task Name' is set to 'Unclaim HTTP Target'. The 'moid' field is selected, and the 'Value Not Specified' field is empty. An 'Edit' button is visible next to the 'moid' field.

Select a **Rollback Task Name**
from the list ('Unclaim HTTP
Target')



The dialog shows the 'moid' task input configuration. The 'Create Task Input' and 'Set Custom Value' options are visible. The 'Search' field is empty. The 'name' field is set to 'Claim HTTP Target' and the 'endpoint' field is set to 'moid'.

The rollback task requires an
input (we created it).

Click on **edit** than click on the
'Claim HTTP Target', 'moid'
task output.

3

Create a Workflow to Claim an HTTP Target

The screenshot displays the workflow designer interface. On the left, the configuration panel for the 'Claim HTTP Target' task is visible, with the 'Workflow Inputs' section highlighted by a red box. This section contains an 'Add Input' button and two input fields: 'name*' and 'endpoint*'. The main workspace shows a workflow diagram with a 'Start' node connected to a 'Claim HTTP Target' task node, which is also highlighted by a red box. An orange arrow points from the 'Claim HTTP Target' task in the tools palette to the task node in the workflow diagram. The workflow diagram also shows 'Success' and 'Failed' output nodes connected to the task node.

Configure Inputs for 'name' and 'endpoint' (String, Required)

Create a Workflow to Claim an HTTP Target

Direct Mapping to 'name' and 'endpoint' workflow inputs

Organization *
default

Workflow Instance Name
Claim HTTP Target

name *
httpbin

endpoint *
httpbin.org

Cancel Execute

Name	Status	Type	IP Address	Claimed Time	Claimed By	Connector Ver...	Last Update
httpbin	Claimed	HTTP Endpoint	httpbin.org	a minute ago	rtortori@cisco.com		a minute ago

Rollback Execution

1

The screenshot shows a workflow designer interface with a 'Start' node connected to a 'Claim HTTP Target' task. The task has a green checkmark and a dashed red arrow pointing to a 'Failed' node. A 'Rollback' button is highlighted with a red arrow pointing to it from the right. A 'Clone Execution' button is also visible.

2

The 'Rollback Execution' dialog box is shown. It includes a 'Selected Request' section with details for 'Claim HTTP Target' (Status: Success). Below is a 'Select tasks to Rollback' section with a search bar and a list of tasks. The 'Claim HTTP Target' task is selected and highlighted with a red box. A 'Rollback' button is highlighted with a red arrow pointing to it from the right.

Rollback Execution
The Rollback execution feature reverts the created or modified entities while executing a workflow

Selected Request

Name	Status
Claim HTTP Target	Success

Select tasks to Rollback

Abort rollback, if any rollback task fails

Search

Show Rollback Supported

Collapse All | Select All | Show Selected (1) |

Claim HTTP Target

3

The screenshot shows a table with 304 items found. The table has columns: Name, Status, Initiator, Target Type, Target Name, Start Time, Duration, and ID. The 'Rollback Claim HTTP T...' entry is highlighted.

Name	Status	Initiator	Target Type	Target Name	Start Time	Duration	ID
Rollback Claim HTTP T...	Success	rtortori@cisco.com	-	-	a few seconds ago	1 s	60d34921696f6e2d30...
Claim HTTP Target	Success	rtortori@cisco.com	-	-	8 minutes ago	3 s	60d3472e696f6e2d30...

search httpbin x Add Filter

0 items found | 10 per page | 0 of 0

NO ITEMS AVAILABLE

Create a Notification Task with Webex

Create a BOT in webex.com

<https://developer.webex.com/docs/bots>

Bots

Give Webex users access to outside services right from their Webex spaces. Bots help users automate tasks, bring external content into the discussion, and gain efficiencies.

Create a Bot

Provide the details necessary to create a bot, such as a name, username, icon, description.

Once the BOT is created, it will redirect to a page where details of the BOT are present; a bot access token is also provided

New Bot

Bot name*

Name of your bot as it will appear in Webex.

ICO Demo Bot

Bot username*

The username users will use to add your bot to a space. Cannot be changed later.

rtotori-ico-demo

@webex.bot

rtotori-ico-demo@webex.bot is available

Icon*

Upload your own or select from our defaults. Must be exactly 512x512px in JPEG or PNG format.



Edit

Description*

Provide some details about what your bot does, how it benefits users, and most importantly, how a user can get started using it. The description should be under 1500 characters. You can use bullets, links, and Markdown formatting. If your app is listed on the Webex App Hub, this field will be used as the listing's description.

ICO demo bot

Supported markdown

1488 characters remaining

By creating this app, you accept the [Terms of Service](#) and [Privacy Statement](#).

Cancel

Add Bot

Copy the BOT Access Token

Congratulations! 🎉

ICO Demo Bot is one step closer to becoming a reality.

ICO Demo Bot

👉 **Next Step:** Use your Bot Access Token to set up your webhook and finish building your bot.

Bot access token

Non-expiring (good for 100 years) access token for your bot. Save this token to set up your webhook.

YTM0MTEyOGItYjczNS00NTQ3LWI4NmQtOGVhYTJjYWM4YzI5YzZlI

Copy Token

💡 **Tip:** Save this token!
It won't be shown again (but you can regenerate a new one if needed).

A generated token will be valid for 100 years

Copy the token as this is the only time you will access it

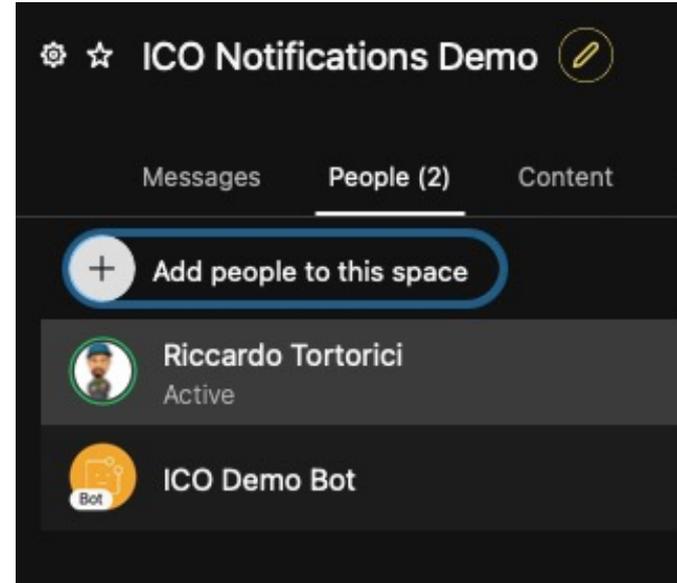
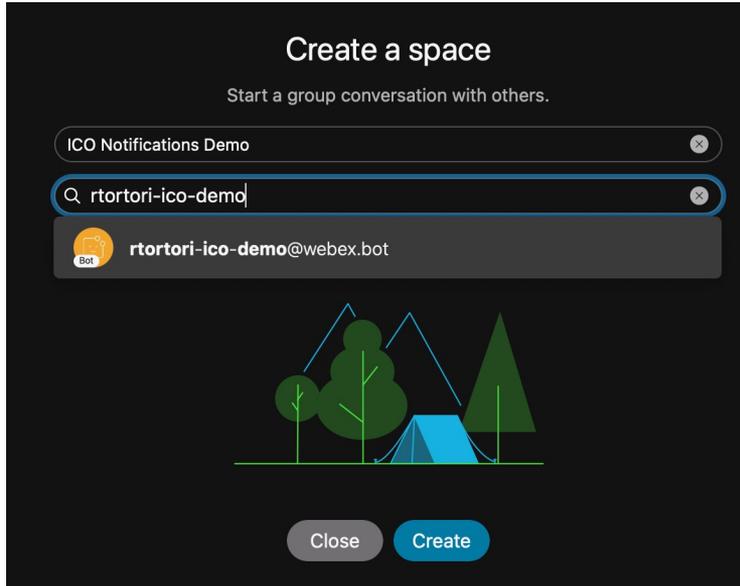
If needed, you can generate a new one.

Bot access token

Non-expiring (good for 100 years) access token for your bot. Save this token to set up your webhook.

[Regenerate Access Token](#)

Create a Webex Space and Invite your BOT



Get the Room ID

<https://developer.webex.com/docs/api/v1/rooms/list-rooms>

The screenshot shows the Webex API client interface. At the top, the method is set to GET and the URL is `/v1/rooms?type=group&sortBy=created&max=1`. Under the 'Header' section, the 'Authorization' toggle is turned on, and the 'Bearer' token field is filled with a redacted token. Below this, a note states: "This limited-duration personal access token is hidden for your security." In the 'Query Parameters' section, the following values are entered: `teamId` is `e.g. Y2lzY29zcGFyazovL3VzL1JPT00`, `type` is `group`, `sortBy` is `created`, and `max` is `1`. A red box highlights the 'Run' button at the bottom right.

This will return the last group created. If you leave the default, you will get all rooms you are subscribed to. Feel free to experiment with **parameters** if needed.

As an alternative, you can use the BOT access token to fetch the list of rooms where the BOT is present

The screenshot shows the 'Response' tab of the Webex API client. The status is `200 / OK`. The response body is a JSON array of room objects. The first object is highlighted with a red box, showing its `id` field: `"id": "Y2lzY29zcGFyazovL3VzL1JPT00vNTBjMmExYzAtZG"`. The full response is as follows:

```
{
  "items": [
    {
      "id": "Y2lzY29zcGFyazovL3VzL1JPT00vNTBjMmExYzAtZG",
      "title": "ICO Notifications Demo",
      "type": "group",
      "isLocked": false,
      "lastActivity": "2021-07-01T19:05:35.452Z",
      "creatorId": "Y2lzY29zcGFyazovL3VzL1BFT1BMRs8xMGJ",
      "created": "2021-07-01T19:05:35.452Z",
      "ownerId": "Y2lzY29zcGFyazovL3VzL09SR0FOSVpBVElPT"
    }
  ]
}
```

Take note of the room **id**
Note: the **id** in this example is truncated on purpose

Claim Webex in Intersight

HTTP Endpoint
An external REST API endpoint to be used in Intersight Orchestrator.

Connect through an Intersight Assist

Name *
Bot RMLAB Notifications

Hostname/IP Address *
webexapis.com Port
0 - 65535

Authentication Required

Authentication Scheme *
Bearer Token

Token *
.....

Enable HTTPS Protocol

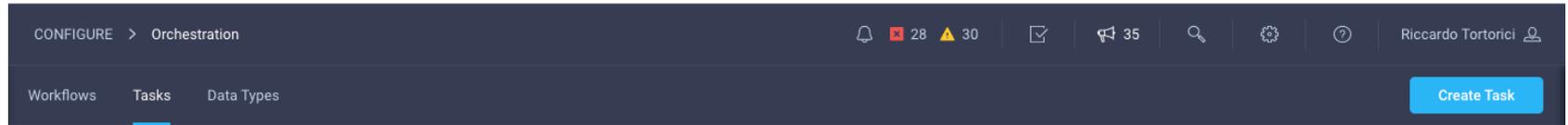
Hostname/IP Address: webexapis.com
Bearer Token Authentication Scheme
Enable HTTPS Protocol

Copy the BOT access token in the **Token** field

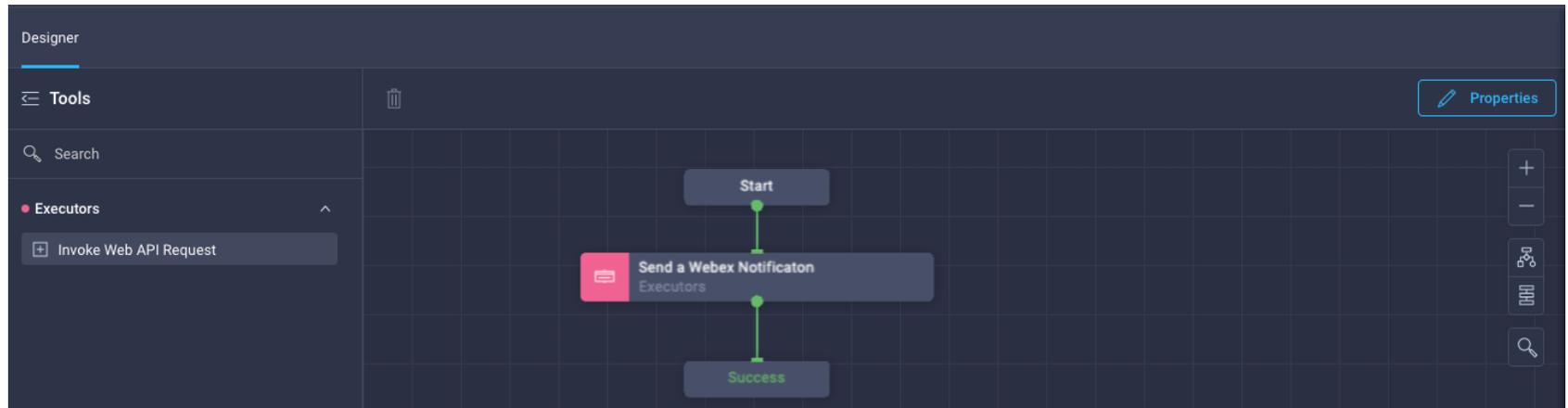
Click **Claim** when done.

Name	Status	Type	IP Address	Claimed Time	Claimed By
Bot RMLAB Notifications	Claimed	HTTP Endpoint	webexapis.com	a few seconds ago	rtortori@cisco.com

Create a Notification Task in the Task Designer



Orchestration -> Tasks -> Create Task



Drag the **Invoke Web API Request** in the designer

Executor Task Properties: General

The screenshot displays a workflow editor interface. On the left, a workflow diagram shows a sequence of three nodes: 'Start', 'Send a Webex Notificaton', and 'Success'. The 'Send a Webex Notificaton' node is highlighted with a blue border. On the right, a properties dialog titled 'Send a Webex Notificaton' is open. The dialog has four tabs: 'General', 'Inputs', 'Outputs', and 'Outcomes'. The 'General' tab is selected. The 'Name *' field contains the text 'Send a Webex Notificaton'. The 'User Description' field contains the text 'Invokes the given Web API against the given endpoint. The'. The 'Set External Target' checkbox is checked and highlighted with a red box.

Under **General**, **Set External Target** and give the task a **Name**

Executor Task Properties: Inputs

Send a Webex Notificaton

General Inputs Outputs Outcomes

Protocol ○ Edit

Value Not Specified -

Method ○ Edit

Custom Value POST

URL ○ * Edit

Custom Value /v1/messages

Headers ○ Edit

Custom Value View Input

Cookies ○ Edit

Value Not Specified -

Response Type ○ Edit

Custom Value JSON

Method: POST

URL: /v1/messages

Headers:

```
{  
  "Content-Type": "application/json"  
}
```

Response Type: JSON

Body:

```
{  
  "roomId": "{{.global.task.input.roomid}}",  
  "markdown": "{{.global.task.input.message}}"  
}
```

`.global.task.input.roomid` and `.global.task.input.message` map to task inputs (yet to be created, see next slides) defining the room id to post a message to (retrieved a few slides back from the Webex API) and the actual message respectively. With this configuration, we can send arbitrary messages base on other task input/outputs in our workflows)

Refer to the documentation to get more details on the many parameters you can use to send messages, attachments, etc : <https://developer.webex.com/docs/api/v1/messages/create-a-message>

Body ○ Edit

Custom Value View Input

Response Parser ○ Edit

Custom Value View Input

Custom Task Properties

The screenshot shows a workflow editor with a 'Send a Webex Notification' task. The task is connected to a 'Start' node and a 'Success' node. The 'Task Properties' panel is open, showing the following configuration:

General	Inputs	Outputs
Organization *	default	
Task Name *	Send a Webex Notification	
Description	Send a Webex Notification	
Retry Count	3	0 - 256
Retry Delay	60	10 - 86400
Timeout	600	10 - 604800
Set Tags	owner:rtortoli	Enter a tag in the key:value format.

The screenshot shows the 'Task Properties' panel with the 'Inputs' tab selected. The inputs are:

Inputs	Outputs
message *	
roomid *	
External Target *	

`.global.task.input.roomid` and `.global.task.input.message` map to task inputs (yet to be created, see next slides) defining the room id to post a message to (retrieved a few slides back from the Webex API) and the actual message respectively.

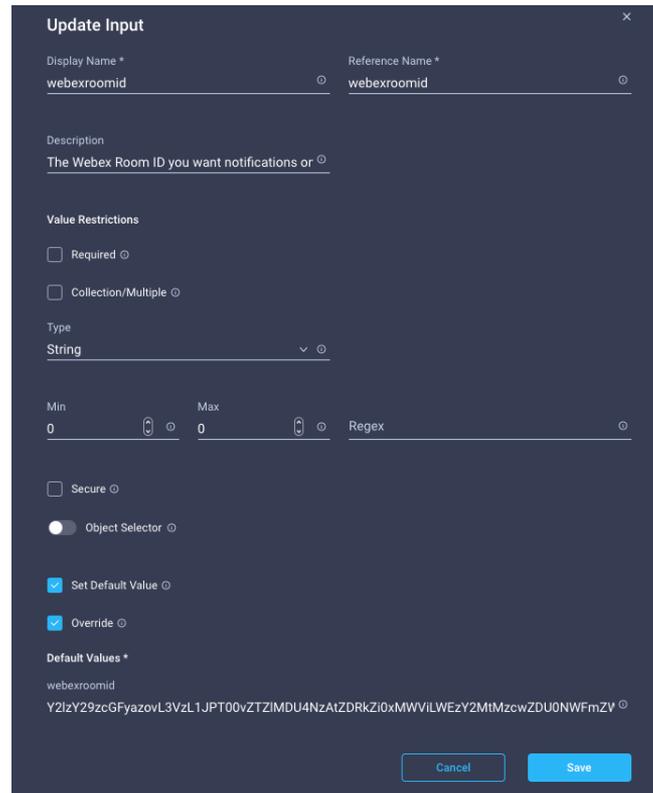
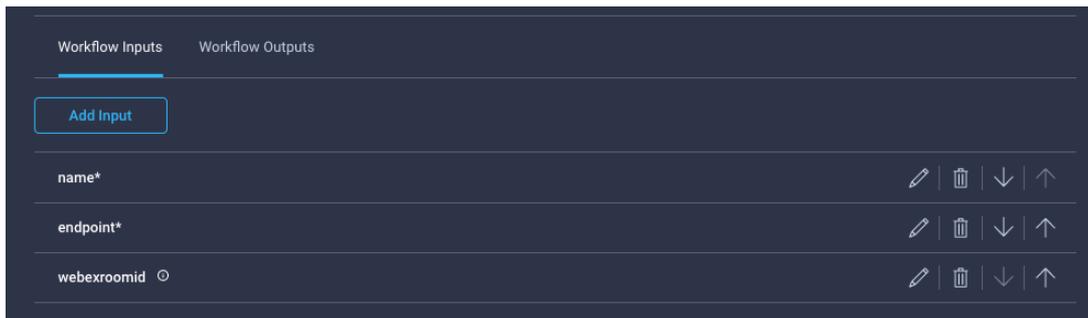
With this configuration, we can send arbitrary messages based on other task input/outputs in our workflows) Refer to the Webex API documentation to get more details on the many parameters you can use to send messages, attachments, etc : <https://developer.webex.com/docs/api/v1/messages/create-a-message>

Use the Notification Task

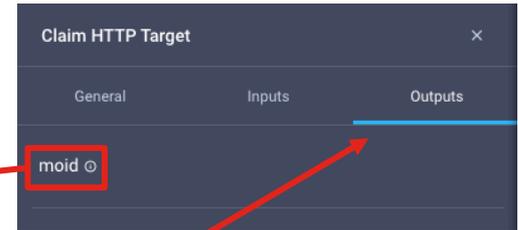
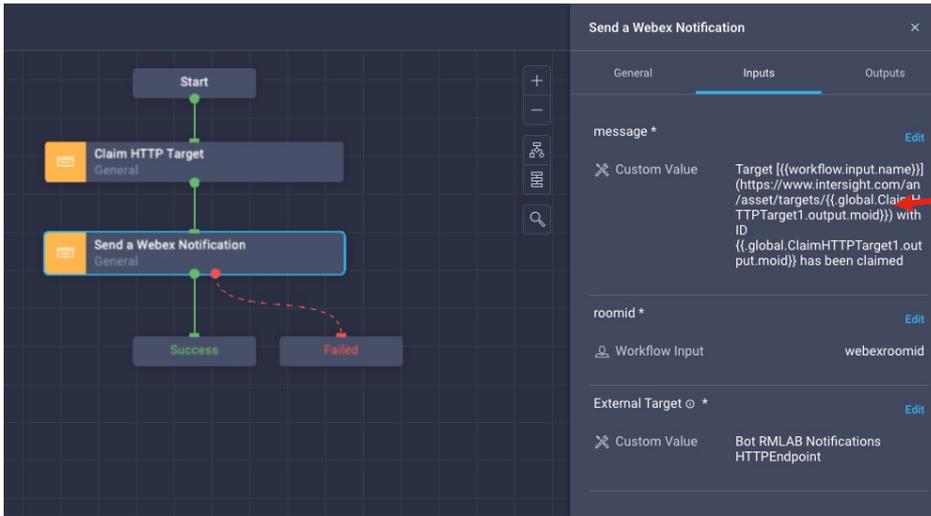
You can create a new workflow or just use an existing one. For this example we will use the HTTP Target Claim task we created a few slides back.

First, we add an additional **Workflow Input** called 'webexroomid' for the Webex Room ID we want our messages notified on. While this can be hardcoded, you may want to leverage different rooms based on the workflow you run.

We set it as a **String**. You can optionally set a default value if you don't want to enter the room ID on each execution.



Use the Notification Task



Recall the **Claim HTTP Target** task returns the target MOID as an output.

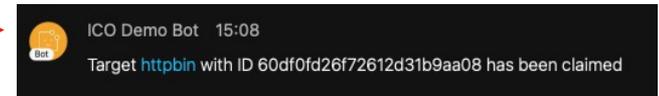
You can leverage this output to build a custom message in your BOT.

For the message input mapping we will leverage the **Advanced Mapping** as we use templates to render a custom message:

```
Target {{{.global.workflow.input.name}}}(https://www.intersight.com/asset/targets/{{{.global.ClaimHTTPTarget1.output.moid}}}) with ID {{{.global.ClaimHTTPTarget1.output.moid}}}) has been claimed
```

As soon as we claim a target, we will have our BOT sending a message like this. **'httpbin'** is the name of the target, but that is actually a hyperlink that will bring you to the Intersight target just claimed. For further info on markdown syntax:

<https://guides.github.com/pdfs/markdown-cheatsheet-online.pdf>



Use the Notification Task

The image shows a workflow configuration interface. On the left, a task configuration card for 'roomid' is shown. It has a 'Workflow Input' type and is mapped to 'webexroomid'. Below it, an 'External Target' dropdown is set to 'Bot RMLAB Notifications HTTPEndpoint'. Two red boxes highlight the 'Edit' buttons for both the task and the external target. On the right, two panels show the configuration options for these elements. The top panel shows the 'Direct Mapping' tab selected, with the instruction 'Map the input to the workflow input or any of the previous task's outputs.' and 'Workflow Input' selected. The bottom panel shows the 'Static Value' tab selected, with the instruction 'Provide custom values as the input.' and a list of external targets where 'Bot RMLAB Notifications' is selected.

roomid will be a **Direct Mapping** to the 'webxroomid' workflow input
External Target a **Static Value** pointing to the Webex endpoint we claimed at the beginning of the process.

Save and **Execute** the workflow

Run the Workflow

The image illustrates the process of running a workflow to claim an HTTP target in Cisco Intersight. It is divided into three main sections:

- Workflow Input Dialog:** A dark-themed dialog box titled "Enter Workflow Input - Claim HTTP Target". It contains the following fields:
 - Organization: default
 - Workflow Instance Name: Claim HTTP Target
 - name: sometarget
 - endpoint: httpbin.org
 - webexroomid: Y2lzY29zcGFyazovL3VzL1JPT00vNTBJMmExYzAtZGE5Zi0xMButtons for "Cancel" and "Execute" are at the bottom.
- Intersight Target Details:** A browser window showing the Intersight interface. The URL is `intersight.com/an/asset/targets/60df24fe6f72612d31bf1ab3`. The "OPERATE" menu is expanded, and the "Details" panel shows the following information:

Details	
Status	Claimed
Name	sometarget
Type	HTTP Endpoint
IP Address	httpbin.org
Port	443
Claimed Time	a minute ago
Claimed By	rtortori@cisco.com
Access Mode	Allow Control
Last Update	a minute ago
- Chat Message:** A message from "ICO Demo Bot" at 16:38 stating: "Target **sometarget** with ID 60df24fe6f72612d31bf1ab3 has been claimed". It is marked as "New messages" and "Seen by" the bot.

Parameter Set – Use Cases

Condition-based workflow input display

The Parameter Set rules control the availability of specific parameters or inputs during the execution. After the first input is specified, the Parameter Set rule controls which subsequent input fields are made available during the workflow execution

Example:

I want to show VMware *Datacenter* workflow input **after** the *Hypervisor Manager* has been selected and **if** the selected manager is in state *Connected*

The screenshot shows a 'Workflow Inputs' panel with a list of parameters. The parameters are: 'Add Input', 'Hypervisor Manager*', 'Datacenter', 'Cluster', 'Host', 'Folder', 'Resource Pool', 'Datastore*', 'Virtual Machine*', 'CPUs', 'Memory', 'Power On', 'Network', and 'Template or Virtual Machine*'. The 'Hypervisor Manager*' and 'Datacenter' parameters are highlighted with red boxes, indicating they are currently visible and selected.

Default behavior:

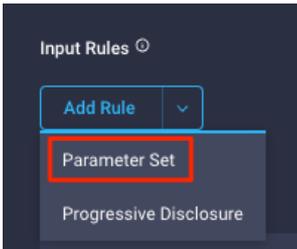
All workflow inputs displayed at once unconditionally

Drawbacks:

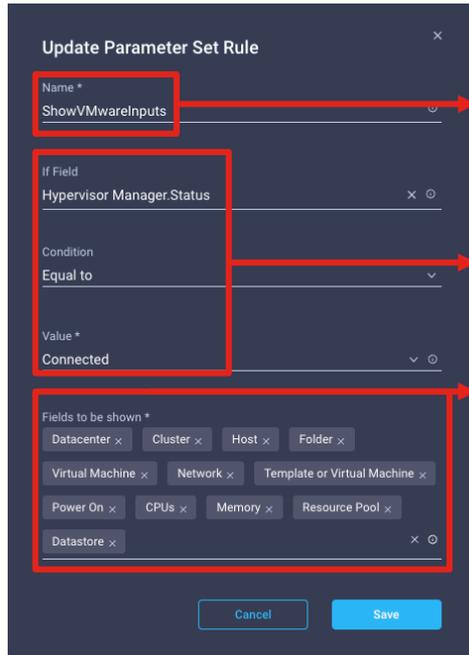
- 1) User may execute a workflow against a disconnected target (which will eventually fail)
- 2) Unstructured inputs, clutter, missing a meaningful flow

The screenshot shows a dialog box titled 'Enter Workflow Input - Create A VM - Master'. The dialog contains a list of parameters: 'Organization', 'Workflow Instance Name', 'Hypervisor Manager', 'Datacenter', 'Cluster', 'Host', 'Folder', 'Resource Pool', 'Datastore', 'Virtual Machine', 'CPUs', 'Memory', and 'Power On'. The 'Hypervisor Manager' and 'Datacenter' parameters are highlighted with red boxes, indicating they are currently visible and selected.

Parameter Set – Implementation Example



Under **General** tab of the workflow in workflow designer, click on **Input Rules, Parameter Set** to add a new parameter set rule

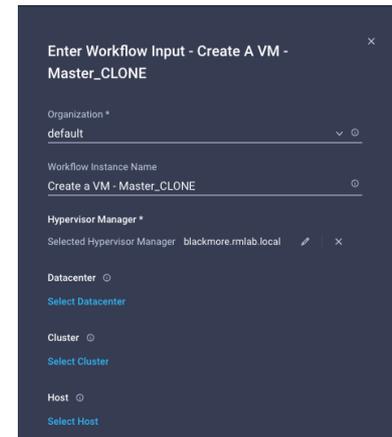
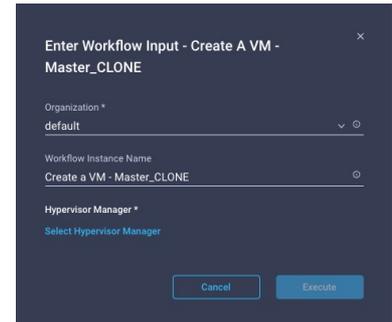


Arbitrary name

Condition

Fields to show if condition is met

Only **Hypervisor Manager** is shown



All other inputs will be disclosed after the condition is met (user has selected a connected hypervisor manager)

Parameter Set – Key Concepts

Documentation:

https://intersight.com/help/saas/resources/Workflow_Designer#workflow_input_parameter_set_and_progressive_disclosure_rules

The supported data types for Parameter Set rules are:

- Boolean
- Enum
- String Object Selector
- MoReference
- Target

If an input is not shown because of a parameter set rule, it will have NO value even if a default has been configured

Example:

Workflow Input **CPU** has a default value of 4

- A parameter set rule prevents **CPU** to be shown to the user. **CPU** value will be *null*
- A parameter set rule shows **CPU** as the condition is met. **CPU** value will be 4

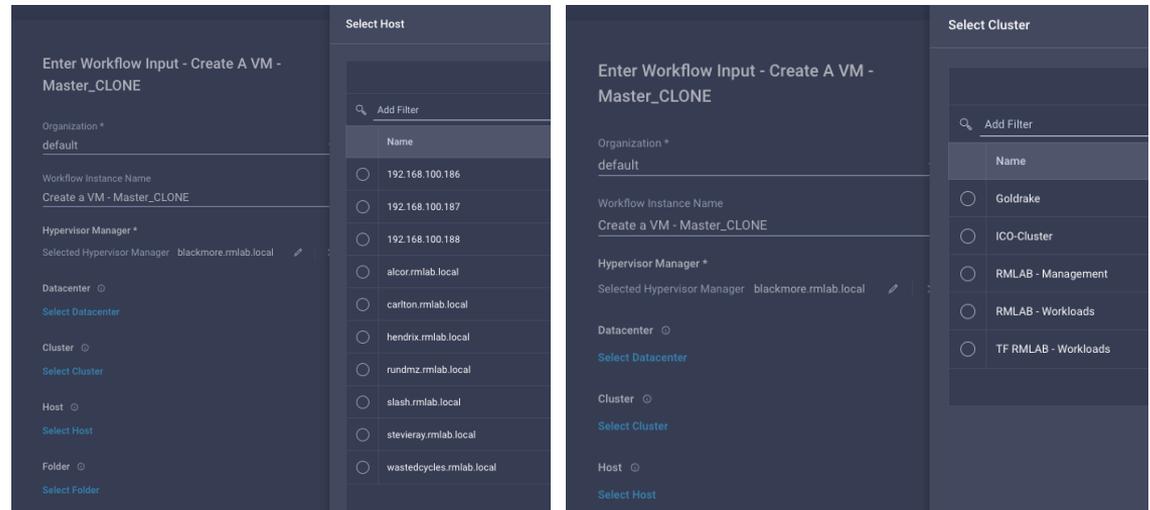
Progressive Disclosure – Use Cases

Filters data available in an input field based on another input selection

The Progressive Disclosure rules filter the data available in an input field based on the preceding selection during a workflow execution. The first input field is populated with broadest options. The subsequent input fields are populated with options based on the previous selection.

Example:

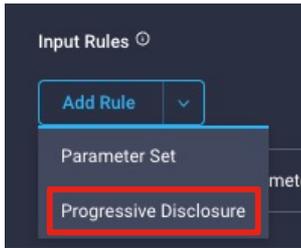
If I select VMware *Cluster* ‘Alpha’ in the *Cluster* input, I want users to select in the *Hosts* input only hosts belonging to that cluster



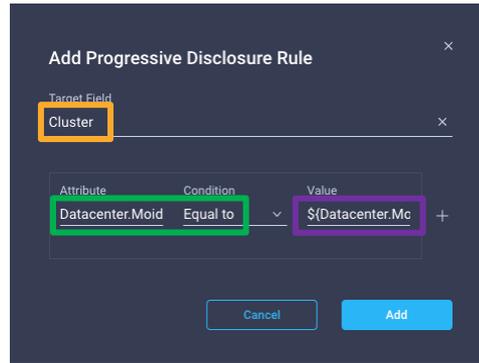
Default behavior:

User can potentially select incompatible data, like hosts not belonging to selected clusters

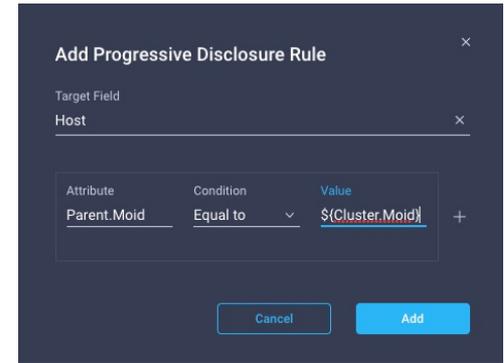
Progressive Disclosure– Implementation Example



Under **General** tab of the workflow in workflow designer, click on **Input Rules, Progressive Disclosure** to add a new rule



We add a second rule for the **Hosts**, in order to only show entries belonging to the selected **Cluster**



From apidocs, you can query the target resource, in the first case 'VmwareClusters', and look at the JSONPATH of the **Attribute** and **Value** you need for your rule. In this example we want the **Cluster** input, to only show entries that have **Datacenter.Moid Equal to** the **Moid** value of the **Datacenter** workflow input (Reference Name)

```
"Datacenter": {  
  "ClassId": "mo.MoRef",  
  "Moid": "610abdf2736c6f2d30f94de8",  
  "ObjectType": "virtualization.VmwareDatacenter",  
  "link": "https://staging.starshipcloud.com/api/v1/virtualization/Vmw  
},
```

Excerpt taken from:
<https://intersight.com/apidocs/apirefs/api/v1/virtualization/VmwareClusters/get/>

Progressive Disclosure– Key Concepts

Documentation:

https://intersight.com/help/saas/resources/Workflow_Designer#workflow_input_parameter_set_and_progressive_disclosure_rules

The supported data types for the Progressive Disclosure rules are:

- MoReference
- Target
- String Object Selector

Executors

Executors

Executors

- Invoke Ansible Playbook

Executes Ansible Playbook against the given endpoint. This task can be executed on targets added as Ansible Endpoint in Intersight.

Inputs

- Ansible Controller*
- Playbook Path*
- Host Inventory*
- Command Timeout
- Command Line Arguments

Outputs

- Exit Code
- Execution Summary
- Execution Log Path

- Invoke PowerShell Script
- Invoke SSH Commands
- Invoke Web API Request

Executors

- Invoke PowerShell Script

Executes the given PowerShell script against the given PowerShell endpoint. The endpoint is typically a windows machine with PowerShell Remoting enabled to allow for scripts to be executed remotely. The task connects to the PowerShell endpoint using the WinRM protocol.

Inputs

- External Target* Script*
- Timeout
- Response Parser
- Outcomes

Outputs

- Exit Code
- Response
- Extracted Parameters

- Invoke SSH Commands
- Invoke Web API Request

Executors

- Invoke SSH Commands

Executes SSH commands against the given endpoint. This task can be executed on targets added as SSH Endpoint or Ansible Endpoint in Intersight.

Inputs

- External Target*
- SSH Command*
- Command Timeout
- Expected Exit Codes
- Show Command Output
- Response Parser

Outputs

- Exit Code
- Command Output
- Command Execution Error
- Extracted Parameters

- Invoke Web API Request

Executors

- Invoke Web API Request

Invokes the given Web API against the given endpoint. The endpoint can be Intersight API or an external endpoint added as Target in Intersight. Please refer Supported Endpoints section in the Web API request end user documentation for the list of Intersight Targets on which Web API task can be invoked.

Inputs

- External Target
- Method
- URL*
- Headers
- Cookies
- Response Type
- Body
- Response Parser
- Outcomes

Outputs

- Headers
- Cookies
- Status Code
- Status Message
- Parameters

Executors are tasks that allow for custom actions in Intersight Cloud Orchestrator (ICO)

They can be used as **Embedded Executors** in the **Workflow Designer** to invoke one-off executions (scoped in the current workflow) or as **Reusable Tasks** in the **Task Designer** to create custom tasks that can be reused across multiple workflows.

When used in the Task Designer you have the option to also create outputs that will extract values from a response parser, the user experience will be similar to what you would have using native tasks.

Executors - Powershell

Powershell Executor

Scope:

Execute Powershell Script in a target Powershell endpoint as embedded task or a custom task

Requirements:

- The endpoint is reachable through Intersight Assist
- The endpoint target is in the **Connected state**
- PowerShell Remoting is enabled on the target endpoint to allow the endpoint to receive PowerShell remote commands
- As of March 2022, we only support WinRM and Windows Powershell Endpoint. SSH remoting is not supported

Documentation:

https://intersight.com/help/saas/resources/Executor_PowerShell#using_the_executor_powershell_script_embedded_task

Claim a Powershell Endpoint

Endpoint Configuration – Create an SSL certificate and WinRM HTTPS Listener

On the target endpoint, open a “Command Prompt” with Administrator privileges.

Create a new self-signed certificate and take note of the result **Thumbprint**.

Replace the placeholder below with your host DNS name or host IP:

```
New-SelfSignedCertificate -DnsName <IP or DNS name> -CertStoreLocation Cert:\LocalMachine\My
```

Create a WinRM HTTPS listener. Fill in with your IP or DNS name and the certificate **Thumbprint**:

```
winrm create winrm/config/Listener?Address=*&Transport=HTTPS @{Hostname="<IP or DNS name>";  
CertificateThumbprint="<Thumbprint>" }
```

```
C:\Users\Administrator>winrm create winrm/config/Listener?Address=*&Transport=HTTPS @{Hostname="192.168.130.61"; CertificateThumbprint="989E42FD13A5E0DF0F33E08D919321CE5104C283"}  
ResourceCreated  
Address = http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous  
ReferenceParameters  
ResourceURI = http://schemas.microsoft.com/wbem/wsman/1/config/listener  
SelectorSet  
Selector: Address = *, Transport = HTTPS
```

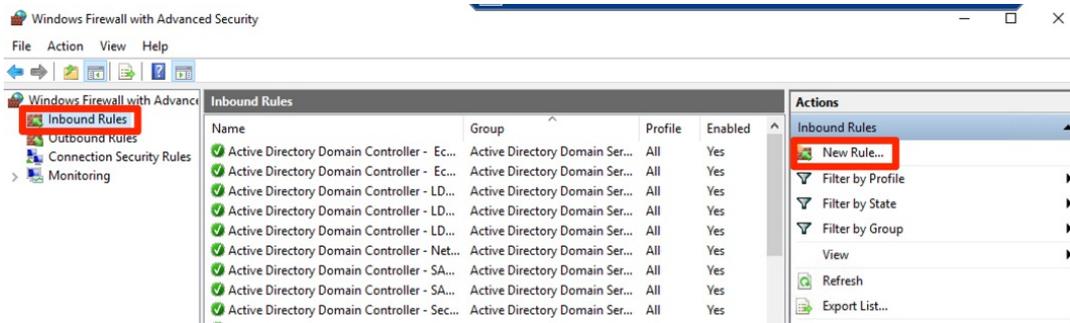
Force Powershell to use TLS 1.2 for web requests

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

Claim a Powershell Endpoint

Endpoint Configuration – Create a firewall rule to allow TCP port 5986

On Windows Firewall, select **Inbound Rules** and create a **New Rule**



Rule Type: Port

Protocol and Ports: TCP, Specific local ports: 5986

Action: Allow the connection

Profile: select all that apply

Name: Allow HTTPS Powershell

Claim a Powershell Endpoint

Endpoint Configuration – Create a firewall rule to allow TCP port 5986

Create a powershell script (example: intersight.ps1), taking the content from:

https://intersight.com/help/saas/resources/Executor_PowerShell

This script will check everything is right and will create the required configuration if something is missing.

Execute Powershell script in the endpoint, using the **Powershell** command line

```
PS C:\Users\Administrator> .\intersight.ps1
Basic authentication is already disabled
Firewall rule already exists for WinRM
PS Remoting has been successfully configured for Intersight.
PS C:\Users\Administrator> _
```

Claim a Powershell Endpoint

Intersight Configuration – Claim the target

The image consists of two screenshots from the Intersight configuration interface. The left screenshot, titled "Select Target Type", shows a sidebar with filters. The "Available for Claiming" checkbox is checked. Under the "Categories" section, "Orchestrator" is selected. In the main area, under the "Orchestrator" category, the "PowerShell Endpoint" option is highlighted with a red box. The right screenshot, titled "Claim PowerShell Endpoint Target", shows a form for configuring the endpoint. The form includes an "Intersight Assist" dropdown set to "intersightassist002.rmlab.local". The "Name" field is "powershell-endpoint-1", the "Hostname/IP Address" is "192.168.130.61", the "Port" is "5986", and the "Username" is "ITFLEET\Administrator". The "Password" field is masked with dots. The entire form area is highlighted with a red box.

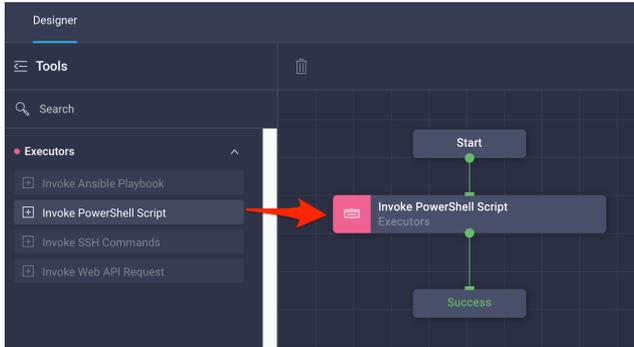
Select one **Intersight Assist** (must be able to reach the target endpoint), specify a **name**, the target **IP** or **Hostname** as well as the credentials.

Hit **Claim**.

The screenshot shows a table of endpoints. The first entry is "powershell-endpoint-1", which is highlighted with a red box. Its status is "Connected", also highlighted with a red box. The endpoint type is "PowerShell Endpoint" and it was last updated "a few seconds ago".

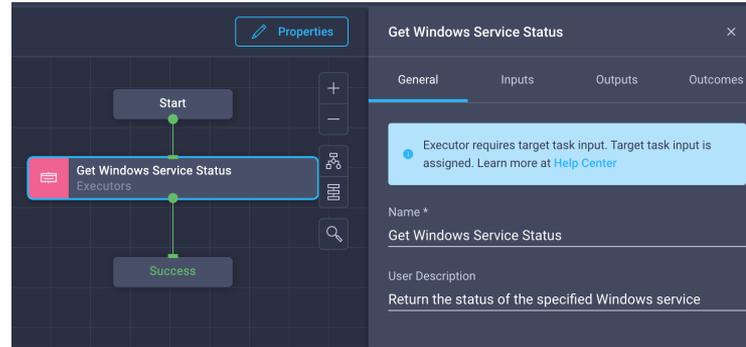
Create a Powershell Custom Task 1/6

Sample Scenario: Get the status of the specified Windows Service



Drag the **Invoke Powershell Script** executor in the designer.

Note: you can't use more than one executor type in a single custom task. However, you can use multiple executors of the same type.



Optionally, assign a name and a description

Create a Powershell Custom Task 2/6

Sample Scenario: Get the status of the specified Windows Service

The screenshot displays a workflow editor with a central task named 'Get Windows Service Status' (Executors). The task is connected to a 'Start' node above and a 'Success' node below. A 'Properties' button is visible in the top right. A 'Task Properties' panel is open on the right, showing the 'General' tab. The following fields are highlighted with red boxes:

- Organization *
default
- Display Name *
Get Windows Service Status
- Reference Name *
GetWindowsServiceStatus

Other visible fields in the 'General' tab include:

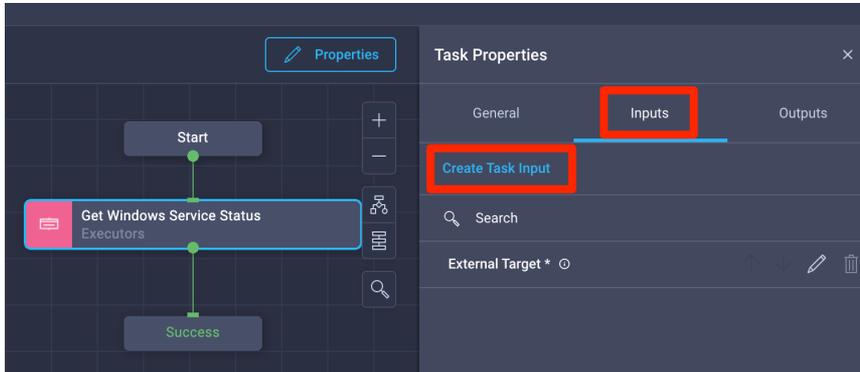
- Description: Return the status of a Windows Service
- Retry Count: 3 (range 0 - 256)
- Retry Delay: 60 (range 10 - 86400)

Click **Properties** to access the **General** custom task properties.

Select an **Organization**, assign a **Display Name**, a **Reference Name** and a **Description** (optional)

Create a Powershell Custom Task 3/6

Sample Scenario: Get the status of the specified Windows Service



Create a **Task Input**.

This will be a required input for the custom task representing the name of the service you want to get the status for.

The **Reference Name** will be used when you reference this input in the Powershell script

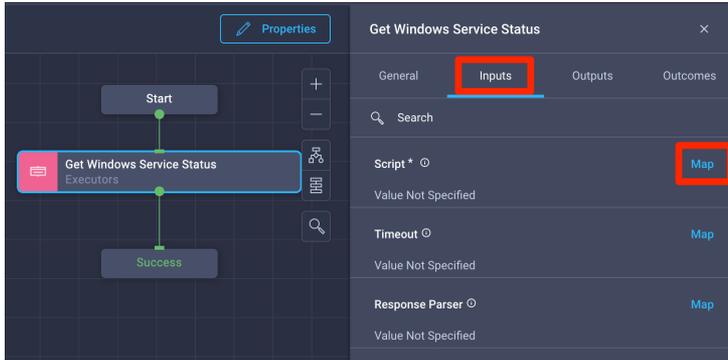
Reference Name: servicename

Type: String

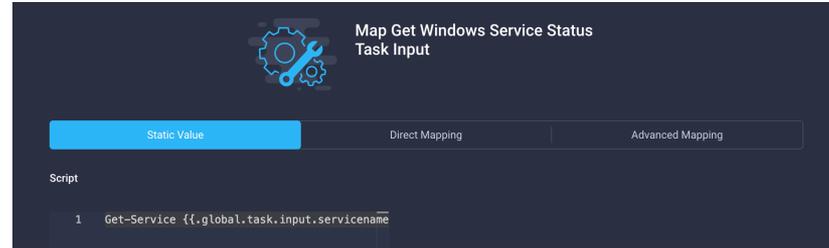
The 'Add Input' dialog box is shown. It has a 'Display Name *' field with the value 'Service Name' and a 'Reference Name *' field with the value 'servicename'. The 'Description' field contains 'Name of the Windows Service'. Under 'Value Restrictions', the 'Required' checkbox is checked. The 'Type' is set to 'String'. There are 'Min' and 'Max' fields, both with the value '0', and a 'Regex' field. There are also 'Secure' and 'Object Selector' checkboxes.

Create a Powershell Custom Task 4/6

Sample Scenario: Get the status of the specified Windows Service



Map a Powershell **Script** under the executor **Inputs**



Write or copy a Powershell Script. In this example, it's just a simple command that requests information on a given Windows Service.

```
Get-Service {{.global.task.input.servicename}}
```

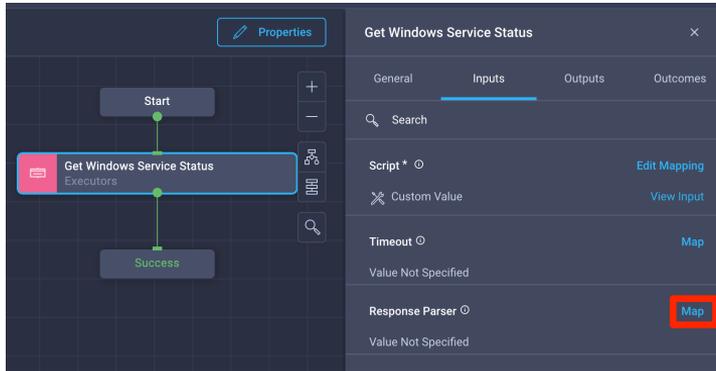
We are using the templating syntax to pass the **Service Name** input (Reference name: **servicename**) as a parameter of the `Get-Service` command

You can also use Direct Mapping to a Workflow Input or a Task Output if necessary. Advanced Mapping may also be used if you want to use gotemplates to apply transformations, etc.

Hit **Map** to confirm.

Create a Powershell Custom Task 5/6

Sample Scenario: Get the status of the specified Windows Service



Map a Response Parser, it will be used to extract data from the server response.

In this example, we will extract the value of key “Status”, we give it a name **ServiceStatus** using JSON Parsing and create a task **Output** with this value.

To do this we use the jsonpath syntax `$.Status`

For additional info on jsonpath syntax:
<https://lzone.de/cheat-sheet/JSONPath>

The screenshot shows the 'Response Parser' configuration panel. The 'Response Type' is set to 'JSON'. The 'Depth' is set to '1'. The 'Enable JSON Parsing' checkbox is checked. The 'JSON Parser Parameters' section shows a 'Path' of '\$.Status' and a 'Name' of 'ServiceStatus'. The 'Type' field is set to 'Text'.

*Note: When you specify the PowerShell script statement and enable the Response Parser Type as JSON, at the time of execution of the workflow, Intersight pipes `convertTo-Json` to the script to get the corresponding responses as JSON output. This means that you should **not** explicitly convert your script output to JSON.*

Create a Powershell Custom Task 6/6

Sample Scenario: Get the status of the specified Windows Service

```
PS C:\Users\Administrator> Get-Service W32Time | ConvertTo-Json
{
  "CanPauseAndContinue": false,
  "CanShutdown": true,
  "CanStop": true,
  "DisplayName": "Windows Time",
  "DependentServices": [
    ],
  "MachineName": ".",
  "ServiceName": "W32Time",
  "ServicesDependedOn": [
    ],
  "ServiceHandle": {
    "IsInvalid": false,
    "IsClosed": false
  },
  "Status": 4,
  "ServiceType": 32,
  "StartType": 2,
  "Site": null,
  "Container": null,
  "Name": "W32Time",
  "RequiredServices": [
    ]
}
```

Sample Response from a Windows Server
2012

Fields

ContinuePending	5	The service continue is pending. This corresponds to the Win32 <code>SERVICE_CONTINUE_PENDING</code> constant, which is defined as 0x00000005.
Paused	7	The service is paused. This corresponds to the Win32 <code>SERVICE_PAUSED</code> constant, which is defined as 0x00000007.
PausePending	6	The service pause is pending. This corresponds to the Win32 <code>SERVICE_PAUSE_PENDING</code> constant, which is defined as 0x00000006.
Running	4	The service is running. This corresponds to the Win32 <code>SERVICE_RUNNING</code> constant, which is defined as 0x00000004.
StartPending	2	The service is starting. This corresponds to the Win32 <code>SERVICE_START_PENDING</code> constant, which is defined as 0x00000002.
Stopped	1	The service is not running. This corresponds to the Win32 <code>SERVICE_STOPPED</code> constant, which is defined as 0x00000001.
StopPending	3	The service is stopping. This corresponds to the Win32 <code>SERVICE_STOP_PENDING</code> constant, which is defined as 0x00000003.

Status code mappings of the ServiceControllerStatus.

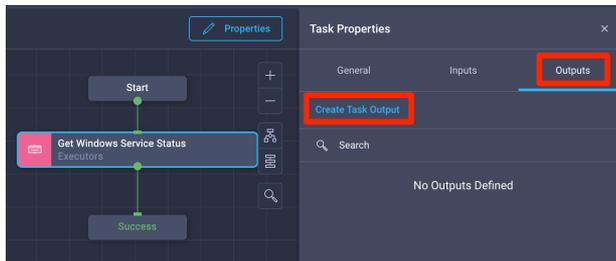
Reference:

<https://docs.microsoft.com/en-us/dotnet/api/system.serviceprocess.servicecontrollerstatus?view=dotnet-plat-ext-6.0>

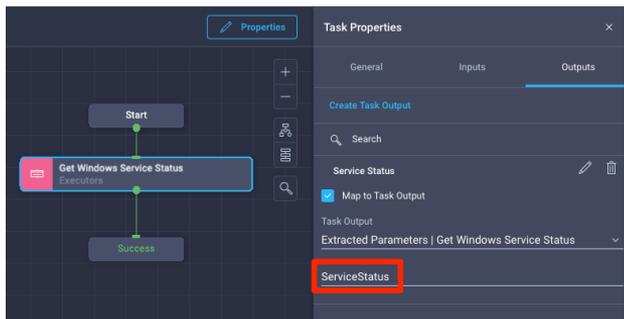
Create a Powershell Custom Task 7a/7

Sample Scenario: Get the status of the specified Windows Service

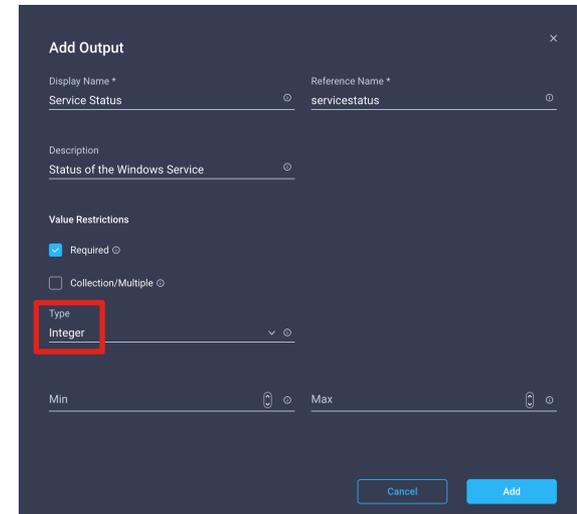
Approach 1: Map the response parser extracted parameter to the task output



Create a **Task Output** from the **Output** tab of the general custom task **Properties**



Map to Task Output specifying the **Extracted Parameters** path of the **Get Windows Service Status** task (in this case **ServiceStatus**)

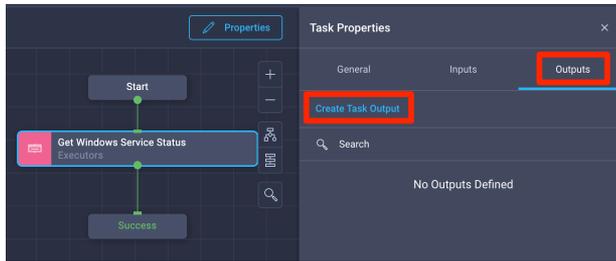


*Note: As of February 14 2022, this approach won't work due to defect **CSCwa37847***

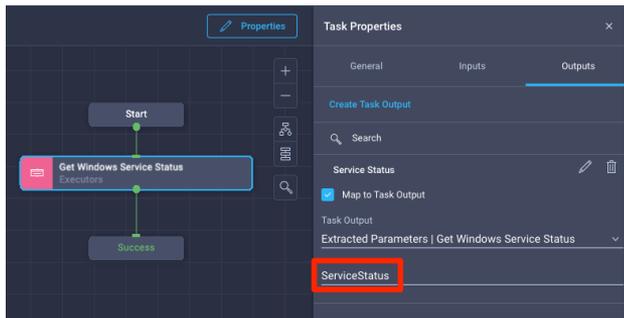
Create a Powershell Custom Task 7b/6

Sample Scenario: Get the status of the specified Windows Service

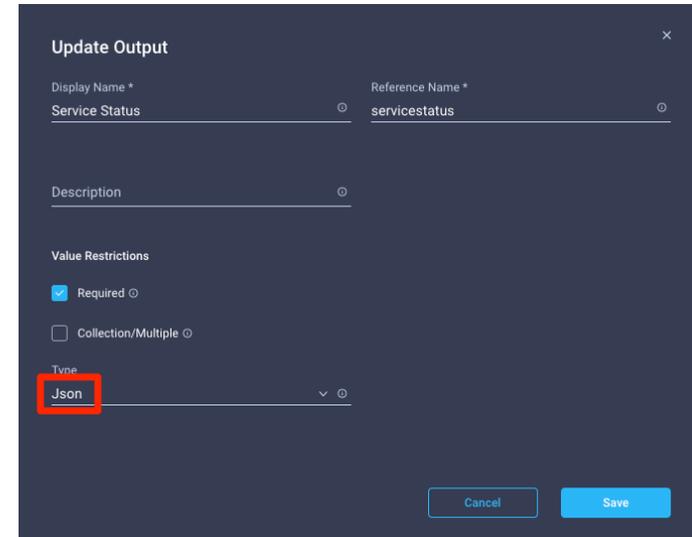
Approach 2: Map the full response to the task output (Response parser will be ignored)



Create a **Task Output** from the **Output** tab of the general custom task **Properties**



Map to Task Output specifying the **Response** of the **Get Windows Service Status** task



Execute a Workflow with the Powershell Custom Task

Sample Scenario: Get the status of the specified Windows Service

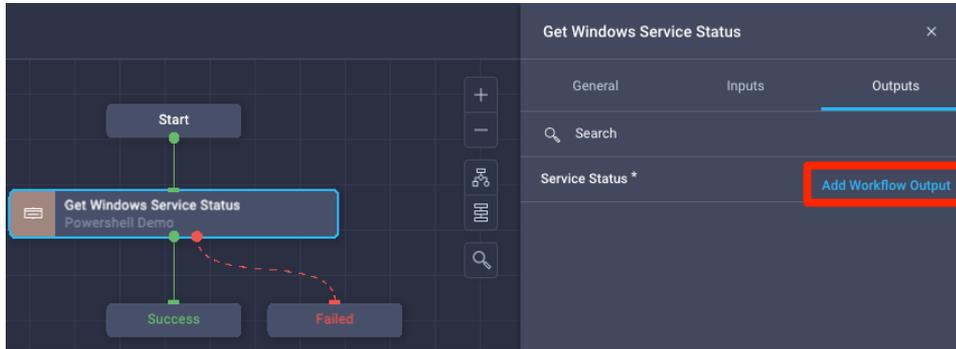
The screenshot displays a workflow editor interface. On the left, a workflow diagram shows a 'Start' node connected to a 'Get Windows Service Status' task, which then branches into 'Success' and 'Failed' nodes. The task is highlighted with a blue border. On the right, the configuration panel for the 'Get Windows Service Status' task is open, showing the 'Inputs' tab. The inputs are:

Input Name	Value	Action
External Target *	Custom Value	Edit Mapping, View Input
Service Name *	Custom Value	Edit Mapping, W32Time

Drag the new **Get Windows Service Status** custom task from the task library.
Map the **External Target** to your Powershell Endpoint.
Map a **Service Name** to a Workflow Input or set it statically for testing purposes.
An example Windows Service for testing may be the **W32Time** service (Windows Time)

Execute a Workflow with the Powershell Custom Task

Sample Scenario: Get the status of the specified Windows Service



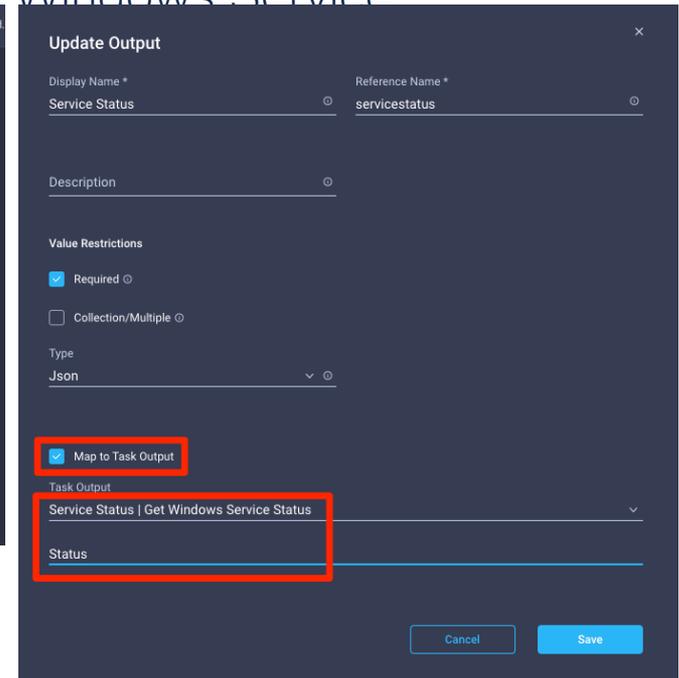
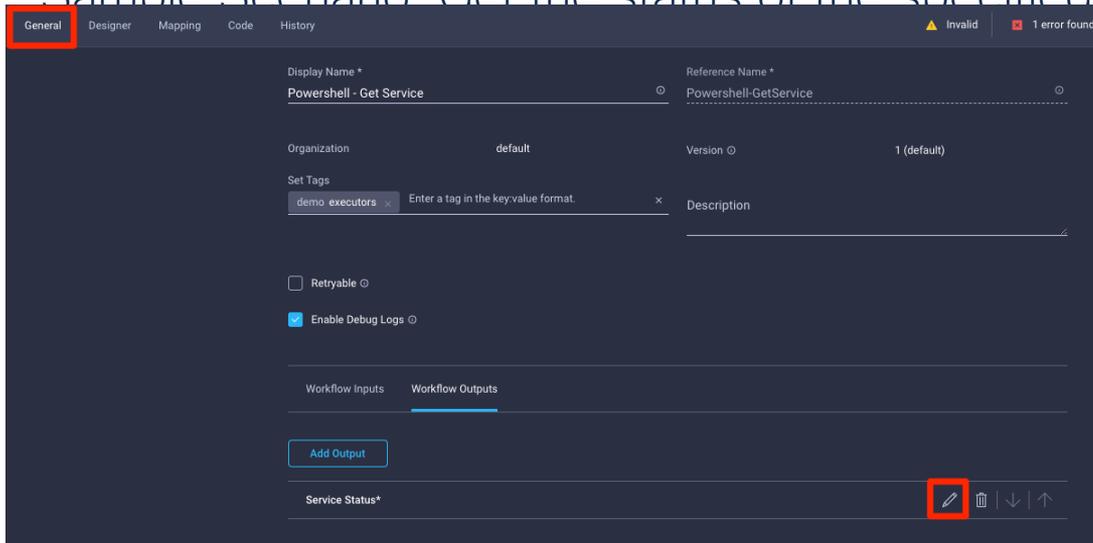
The 'Add Output' dialog box is shown. It has a title bar with a close button. The 'Display Name *' field contains 'Service Status' and the 'Reference Name *' field contains 'servicestatus'. Below these is a 'Description' field. Under the 'Value Restrictions' section, the 'Required' checkbox is checked, and the 'Collection/Multiple' checkbox is unchecked. The 'Type' dropdown menu is open, and 'Json' is selected and highlighted with a red box. At the bottom right, there are 'Cancel' and 'Add' buttons.

On the task output, select **Add Workflow Output** to create a workflow output.

The new Workflow is going to have automatically the same data type of the task output, click **Add**

Execute a Workflow with the Powershell Custom Task

Sample Scenario: Get the status of the specified Windows Service



In the **General** tab, edit the Workflow Output **Service Status** and **Map to Task Output** the **Status** path of the **Service Status** output of **Get Windows Service Status** task. Save and execute the workflow

Execute a Workflow with the Powershell Custom Task

Sample Scenario: Get the status of the specified Windows Service

Verification

The screenshot displays a workflow execution interface. On the left, a workflow diagram shows a 'Start' node leading to a 'Get Windows Service Status' task (highlighted with a green border and a green checkmark), which then branches into 'Success' and 'Failed' nodes. The 'Success' node is highlighted in green. On the right, the execution details for the 'Get Windows Service Status' task are shown, including a 'Workflow Outputs' section with a red border containing the text 'Service Status: 4'. Below this, the task's execution history is visible, showing a successful run on Feb 14, 2022 at 03:15:25 PM. The 'Outputs' section is expanded, showing a tree structure of results: 'ConfigResults: [1]' containing 'Object: (3)', which includes 'ConfigResCtx: { 1 }' with 'EntityData: (1)' (task: workflow.PowerShellApiTask), 'State: Ok', and 'Type: Config'. At the bottom, 'Service Status: (11)' is also listed.

Executors - SSH

SSH Executor

Scope:

Execute CLI commands over SSH in a target endpoint as embedded task or a custom task

Requirements:

- The endpoint is reachable through Intersight Assist
- The endpoint target is in the **Connected state**
- SSH server is enabled and running on the target endpoint
- Username/Password or private key for authentication

Documentation:

https://www.intersight.com/help/saas/resources/Executor_SSH

Supported CLI Command Types

The screenshot shows a configuration interface for CLI commands. At the top, there are three tabs: "Static Value" (selected), "Direct Mapping", and "Advanced Mapping". Below the tabs is a light blue bar with a bullet point and the text "Provide custom values as the input." The main configuration area is titled "SSH Command" and contains several fields: "Command *" (with a dropdown arrow), "Command Type" (with a dropdown arrow), "Expect Prompts" (a container for "Expect *" and "Send *" fields, with a plus sign to the right), and "Shell Prompt *" (with a dropdown arrow). The "Command Type" section has two radio buttons: "Non-Interactive" (unselected) and "Interactive" (selected).

Command Types:

Non-Interactive

Commands that exit as soon as it's executed

Example: `sh /path/to/bin -A -B`

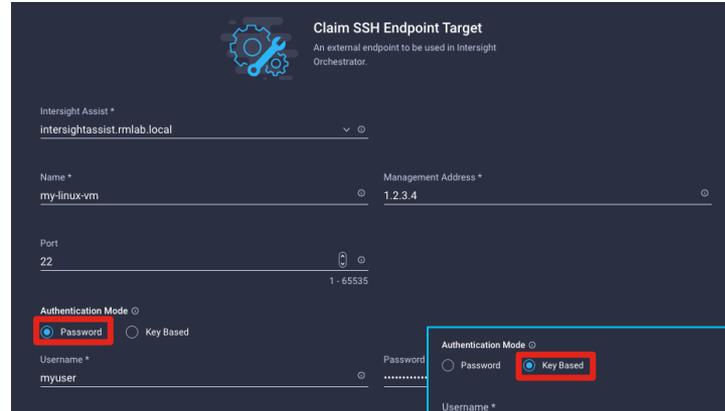
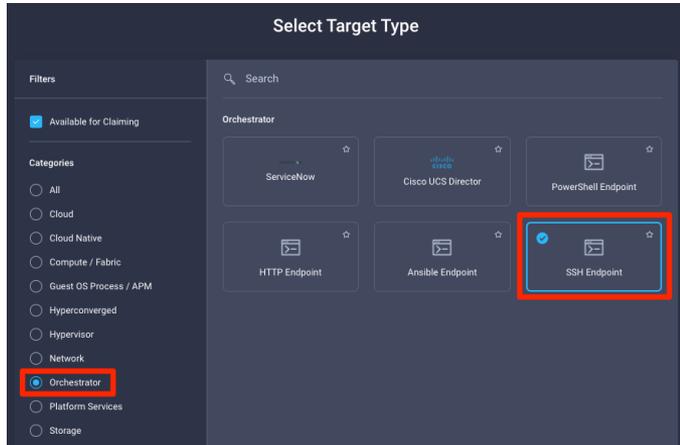
Interactive

Commands that requires user interaction (i.e. asking questions)

Example: `sudo yum install -> y`

Claim a SSH Endpoint

Intersight Configuration – Claim the target

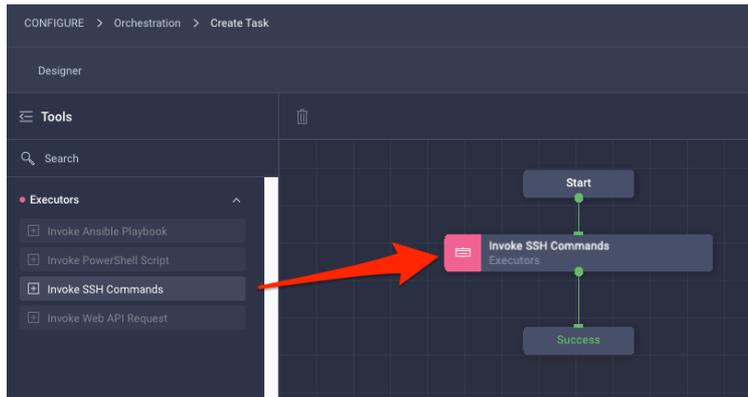


Select one **Intersight Assist** (must be able to reach the target endpoint), specify a **name**, the target **IP** or **Hostname** as well as the credentials. For SSH, you can select between **Password** authentication mode and **Key Based** Hit **Claim**.



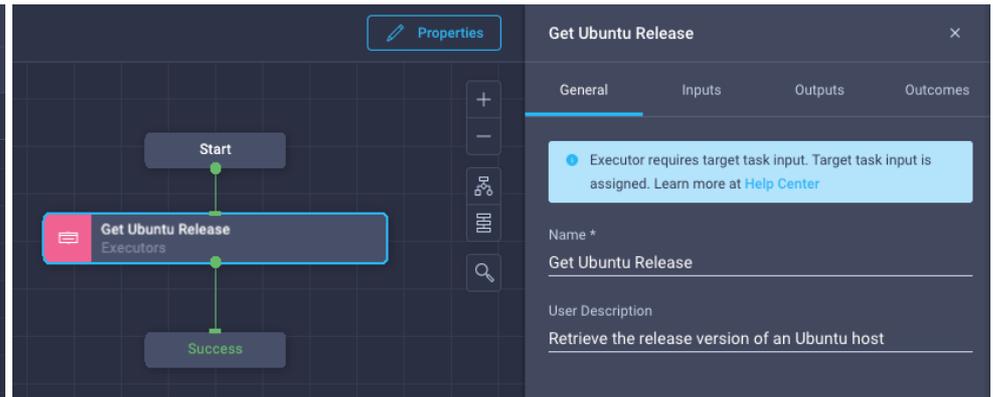
Create a SSH Custom Task - Non-Interactive 1/5

Sample Scenario: Get the release number of an Ubuntu Linux Host



Drag the **Invoke SSH Command** executor in the designer.

Note: you can't use more than one executor type in a single custom task. However, you can use multiple executors of the same type.



Optionally, assign a name and a description

Create a SSH Custom Task - Non-Interactive 2/5

Sample Scenario: Get the release number of an Ubuntu Linux Host

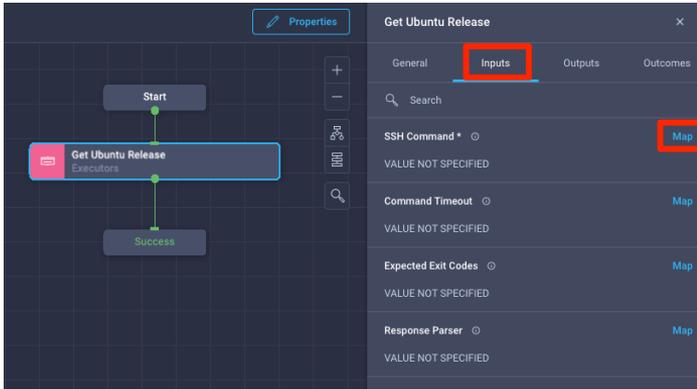
The screenshot shows a workflow editor on the left with three nodes: 'Start', 'Get Ubuntu Release' (highlighted in blue), and 'Success'. The 'Get Ubuntu Release' node is connected to 'Start' and 'Success'. On the right, the 'Task Properties' panel is open, showing the 'General' tab. The 'Organization *' field is set to 'default'. The 'Display Name *' field is 'Get Ubuntu Release Number'. The 'Reference Name *' field is 'GetUbuntuReleaseNumber'. The 'Description' field contains 'Retrieve the release number of an Ubuntu Host'. The 'Retry Count' is set to 3, and the 'Retry Delay' is set to 60. The 'Properties' button is visible at the top left of the workflow editor.

Click **Properties** to access the **General** custom task properties.

Select an **Organization**, assign a **Display Name**, a **Reference Name** and a **Description** (optional)

Create a SSH Custom Task - Non-Interactive 3/5

Sample Scenario: Get the release number of an Ubuntu Linux Host

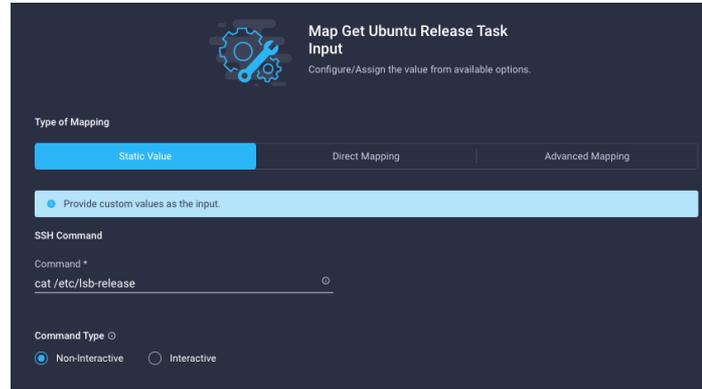


As mentioned, we have two ways to execute SSH commands:

1. Non-interactive Script
2. Interactive Script

In this example, we will use a non-interactive command.

Map a **SSH Command** under the executor **Inputs**



The command `cat /etc/lsb-release` will return the details of the installed Ubuntu release. Here's an example of the output:

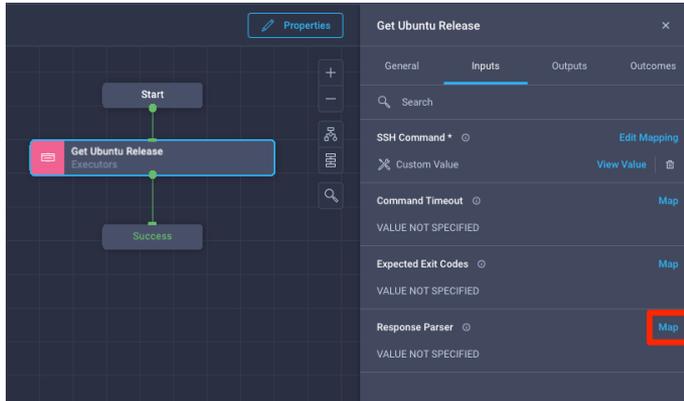
```
rtortori@rtortori-ubuntu-jh:~$ cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=18.04
DISTRIB_CODENAME=bionic
DISTRIB_DESCRIPTION="Ubuntu 18.04.6 LTS"
```

You can also use Direct Mapping to a Workflow Input or a Task Output if necessary. Advanced Mapping may also be used if you want to use gotemplates to apply transformations, etc.

Hit **Map** to confirm

Create a SSH Custom Task – Non-Interactive 4/5

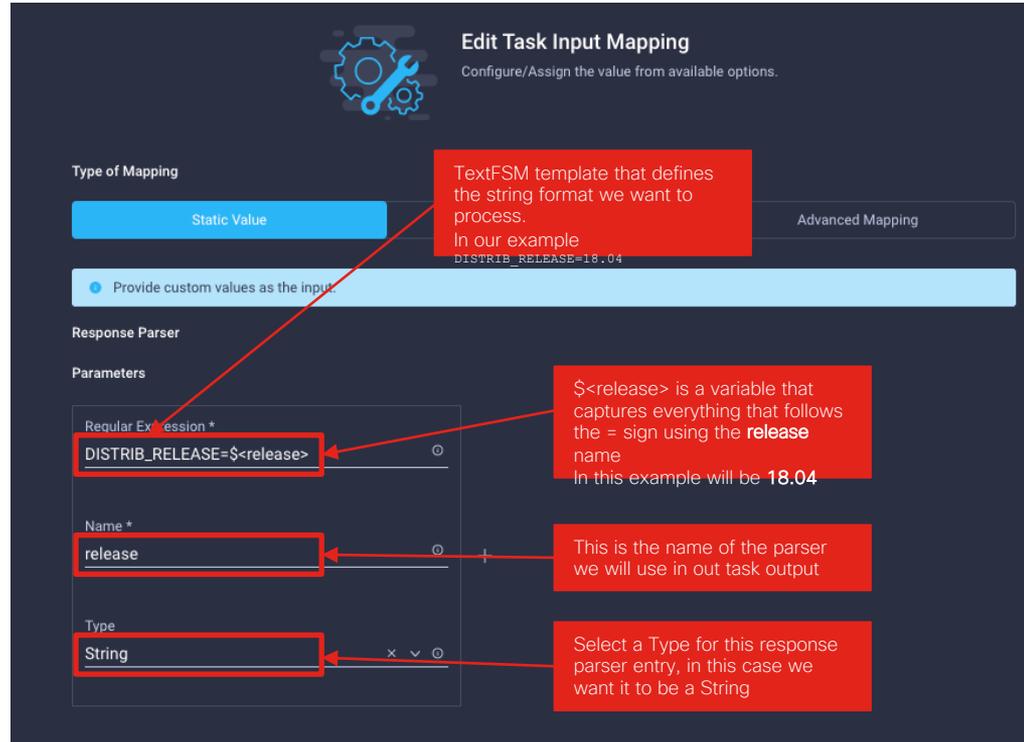
Sample Scenario: Get the release number of an Ubuntu Linux Host



Map a Response Parser, it will be used to extract data from the server response.

The SSH Executor uses the TextFSM which is a parser for semi-formatted text.

In this example, we will extract the **release** output from the output of the `cat /etc/lsb-release` command. Hit **Save** once done to save the task.



TextFSM template that defines the string format we want to process.
In our example
`DISTRIB_RELEASE=18.04`

Static Value | **Advanced Mapping**

Provide custom values as the input

Response Parser

Parameters

Regular Expression *
`DISTRIB_RELEASE=${release}`

Name *
`release`

Type
`String`

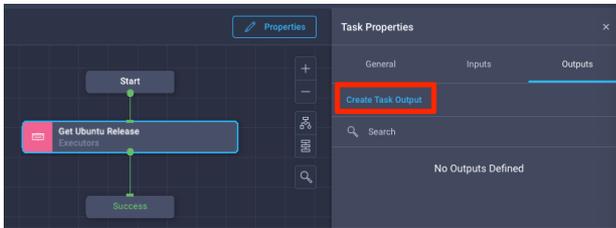
`<release>` is a variable that captures everything that follows the = sign using the **release** name
In this example will be **18.04**

This is the name of the parser we will use in our task output

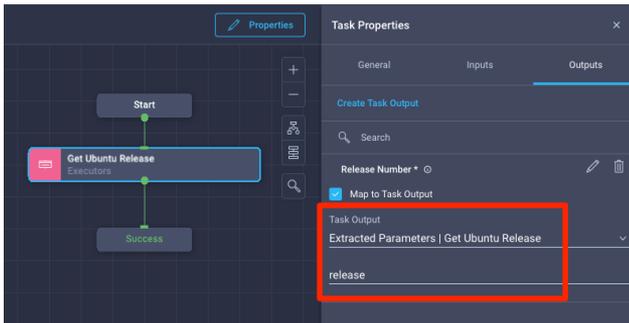
Select a Type for this response parser entry, in this case we want it to be a String

Create a SSH Custom Task – Non-Interactive 5/5

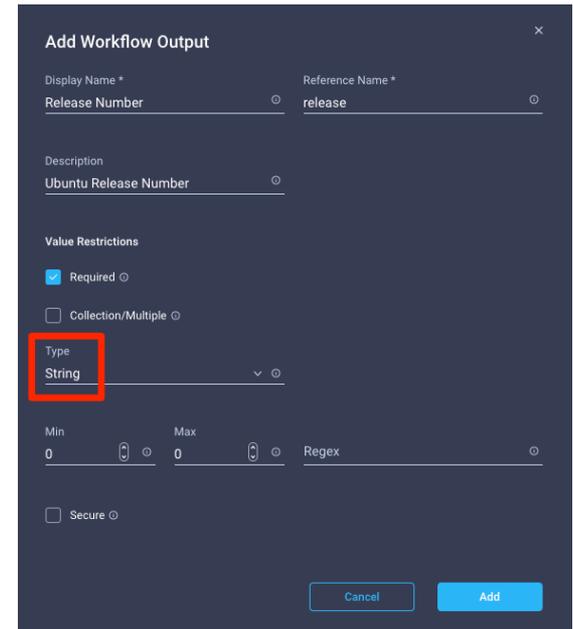
Sample Scenario: Get the release number of an Ubuntu Linux Host



Create a **Task Output** from the **Output** tab of the general custom task **Properties**

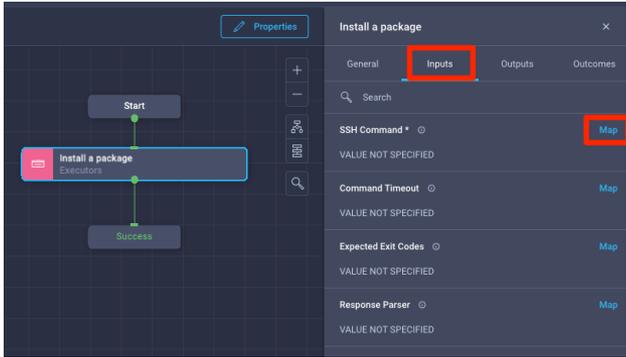


Map to **Task Output** specifying the **Extracted Parameters** path of the **Get Ubuntu Release** task (in this case **release**, the response parser entry we created earlier)



Create a SSH Custom Task - Interactive

Sample Scenario: Install a package using APT



If you want to leverage an interactive command, **Map** a **SSH Command** under the executor **Inputs** and select **Interactive**

Our goal in this example will be to install a package which name will be passed as a workflow input (Ref name: **PackageToInstall**) using the command:

```
sudo apt install {{.global.task.input.PackageToInstall}} -y
```

This is an interactive command as the system will ask for the sudo password as we are connected as the user **admin**. We need to intercept that request and send the password, then continue the execution

Map Install a package Task Input
Configure/Assign the value from available options.

Type of Mapping
Static Value | Direct Mapping | Advanced Mapping

Provide custom values as the input.

SSH Command
Command *
sudo apt install {{.global.task.input.PackageToInstall}} -y

Command Type
 Non-Interactive Interactive

Expect Prompts
Expect *
\[sudo\]\\spassword\\sfor\\sadmin:\\s\$

Send
{{.global.task.input.SudoPassword}}

Shell Prompt *
~:\\s\\s\$

Command to execute

Matches (regex):
[sudo] password for admin:

When this is matched from the output...

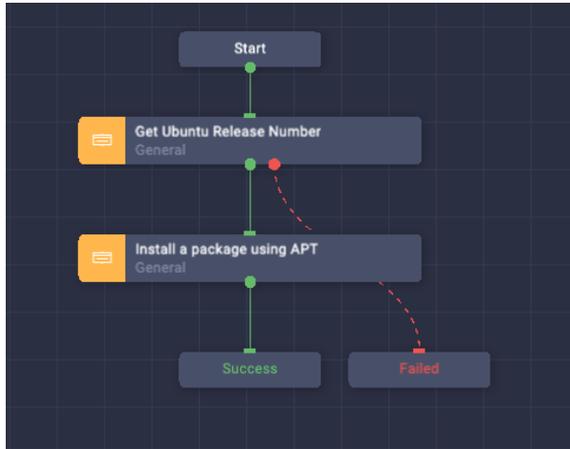
...send this (the password). In this case it's a static mapping to a task input

This is a regex that matches the entry prompt, the command is executed only if there is a match.
In this example we match whatever ends with :~\$ (note the \s that represents an empty space after the \$). The ending \$ is mandatory

Note: all regular expressions **must** end with the \$ sign.
The creation of task inputs is not covered for this example

Execute a Workflow with the SSH Custom Task

Sample Scenario: Get release version and install a package



In this example we will run both SSH tasks (interactive and non-interactive). Drag both tasks from the task library

Map the **External Targets** to the SSH hosts we claimed. All commands will be executed on this host

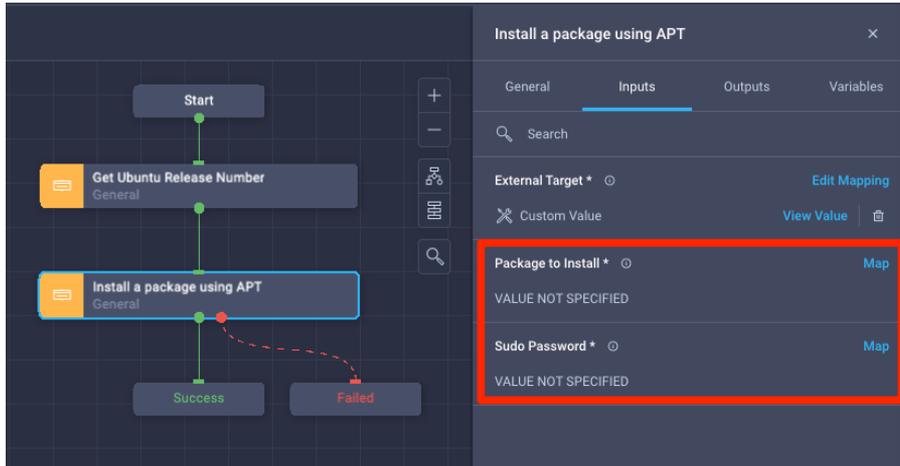
The screenshot shows the 'Map Task Input' configuration interface. On the left, there are options for 'Type of Mapping' (Static Value, Provide custom values as the input) and 'External Target' (Select External Target). On the right, a table lists 6 items found, with 10 items per page. The table has columns for Name and Target Type.

Name	Target Type
<input type="radio"/> ico-ansible-controller	AnsibleEndpoint
<input checked="" type="radio"/> ico-ssh-controller	SSHEndpoint
<input type="radio"/> Ansible Controller SSH Claim	SSHEndpoint
<input type="radio"/> rrtori-ssh-host	SSHEndpoint
<input type="radio"/> tempLinuxVMHost_frankie765	SSHEndpoint
<input type="radio"/> ssh-localhosttest	SSHEndpoint

Selected 1 of 6 Show Selected Unselect All

Execute a Workflow with the SSH Custom Task

Sample Scenario: Get release version and install a package



For the **Install a package using APT** task we configured two inputs, map them accordingly. Either statically or using a workflow input.

In this example the **Package to Install** input will be a direct mapping to a workflow input while the **Sudo Password** will be statically assigned

Execute a Workflow with the SSH Custom Task

Sample Scenario: Get release version and install a package

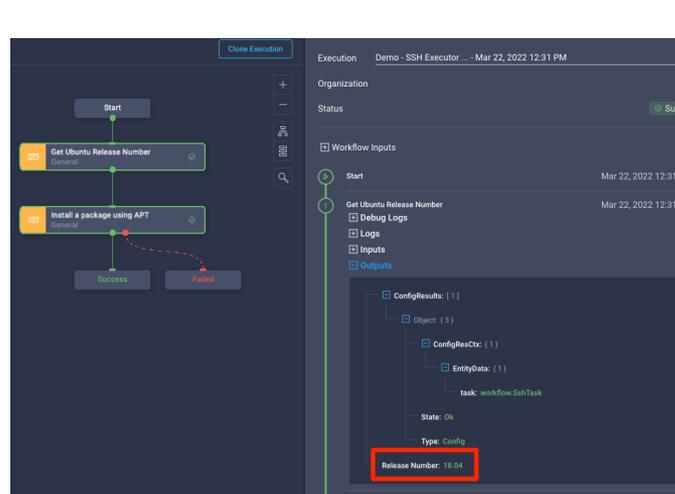
Enter Workflow Input - Demo - SSH Executor - Custom Tasks

Organization *
default

Workflow Instance Name
Demo - SSH Executor - Custom Tasks

Package to Install *
dos2unix

Cancel Execute



```
Execution Demo - SSH Executor ... - Mar 22, 2022 12:31 PM
Organization default
Status Success

Get:1 http://it.archive.ubuntu.com/ubuntu bion
1% [1 dos2unix 3.188 B/351 kB 1%]
9% [1 dos2unix 41.2 kB/351 kB 12%] 6.354 B/s
23% [1 dos2unix 101 kB/351 kB 29%] 6.354 B/s
44% [1 dos2unix 192 kB/351 kB 55%] 6.354 B/s
68% [1 dos2unix 258 kB/351 kB 85%] 6.354 B/s
100% [Working] 6.354 B/s 0s

Fetched 351 kB in 14s (24.5 kB/s)

Selecting previously unselected package dos2u
(Reading database ...
(Reading database ... 5%
(Reading database ... 10%
(Reading database ... 15%
(Reading database ... 20%
(Reading database ... 25%
(Reading database ... 30%
(Reading database ... 35%
(Reading database ... 40%
(Reading database ... 45%
(Reading database ... 50%
(Reading database ... 55%
(Reading database ... 60%
(Reading database ... 65%
(Reading database ... 70%
(Reading database ... 75%
(Reading database ... 80%
(Reading database ... 85%
(Reading database ... 90%
(Reading database ... 95%
(Reading database ... 100%
(Reading database ... 203418 files and director
Preparing to unpack .../dos2unix_7.3.4-3_amd6
Progress: [ 0%] [#####]
Progress: [ 33%] [#####]
Progress: [ 67%] [#####]
```

Execute the workflow and inspect the outputs (Note: Enable Debug Logs is turned on for this workflow)

Executors - Ansible

Ansible Executor

Scope:

Execute Ansible Playbooks in a target control node as embedded task or a custom task

Requirements:

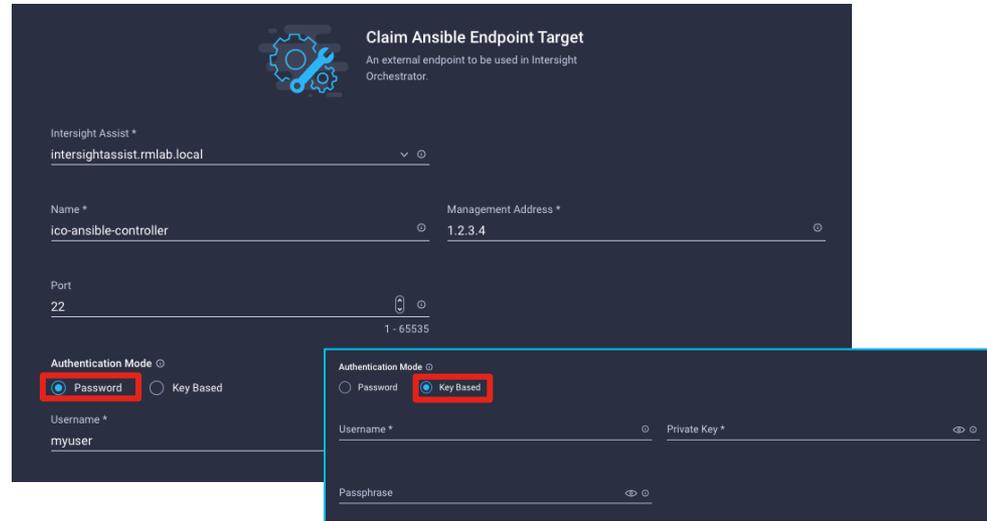
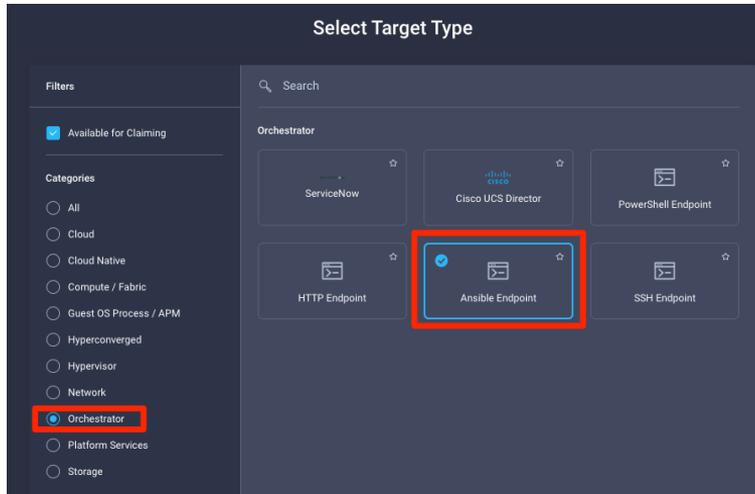
- The Ansible Control node (Intersight Target) is reachable through Intersight Assist
- The Ansible Playbook modules and executables (i.e. `ansible-playbook`) required to run the playbooks are already installed on the Ansible Target
- The Ansible Control Node has password-less SSH access to the hosts (endpoints)
- The Ansible Control Node can resolve endpoints hostname if the Host Inventory file specified the hostnames

Documentation:

https://intersight.com/help/appliance/resources/Executor_Ansible

Claim an Ansible Target

Intersight Configuration – Claim the target



Select one **Intersight Assist** (must be able to reach the target endpoint), specify a **name**, the target **IP** or **Hostname** as well as the credentials. For Ansible target controller, you can select between **Password** authentication mode and **Key Based** Hit **Claim**.



Anatomy of the Ansible Playbook Executor

The screenshot shows a web interface for 'Invoke Ansible Playbook'. It has a search bar and four input fields, each with a 'Map' icon to its right. The fields are: 'Playbook Path *', 'Host Inventory *', 'Command Timeout', and 'Command Line Arguments'. Each field currently contains the text 'VALUE NOT SPECIFIED'.

The full path of the Ansible Playbook in the claimed Ansible Control Node
Example: `/home/ansible/deploy-nginx.yaml`

This can have two formats:

1. The full path of an existing inventory file on the host which will be used by the ansible playbook to retrieve the targets
 2. Comma separated IPs or hostnames of the targets.
- Note: it is required to **always** have a comma at the end, even if there's only one entry.

Example: `root@1.2.3.4
admin@myswitch.company.com`

Timeout of the playbook execution. Default is 600 seconds. If this timer expires the task will fail

The command line arguments for running the Ansible playbook against the given endpoint.
Note: The escape character backslash(\) needs to be used when the command line arguments contain double quotes(") in them.

The following command line options are not supported:

1. `-vvv`
2. `-vvvv`
3. `-k, -K`
4. `-c`
5. `--connection`
6. `--sftp-extra-args`
7. `--scp-extra-args`
8. `--ask-vault-password`
9. `--step`

For more information on the Ansible playbook documentation and the list of other supported command line options, see:

<https://docs.ansible.com/ansible/latest/cli/ansible-playbook.html>

End to end Ansible Scenario

Sample Scenario: Deploy NGINX Webserver

In this sample scenario, we will deploy an NGINX webserver in a target host.
The actual Ansible implementation consists in running two playbooks:

1. Install NGINX
2. Sync Site Content

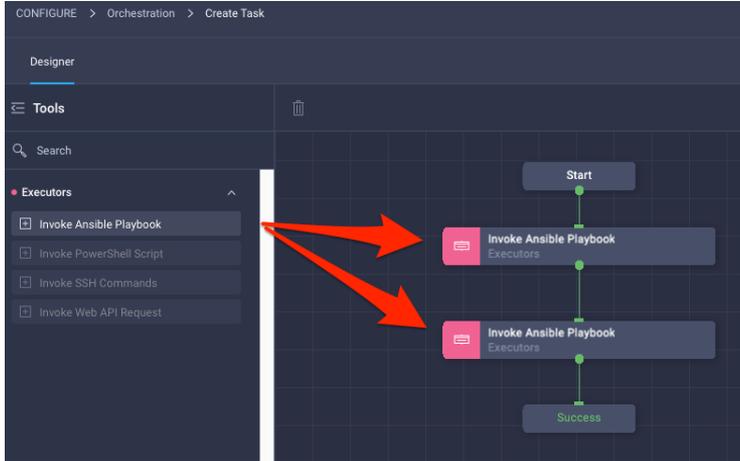
We will create a custom task named `Install NGINX with Ansible` with two Ansible Playbook executors to invoke those two runs.

As the Ansible playbook executor requires password-less SSH access to the endpoints, in the complete workflow we will also create an additional task with the **SSH Executor** to copy the ssh public key of the Ansible controller to the target host.

The Playbooks used can be found here: [<PLACEHOLDER>](#)

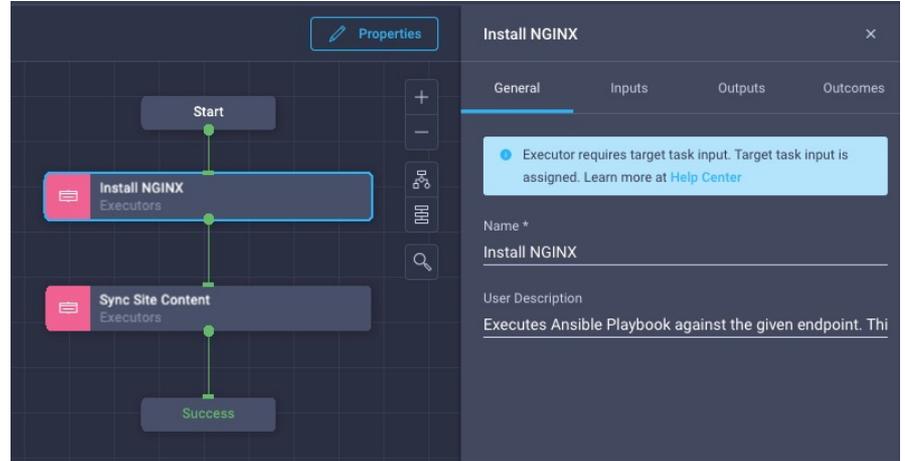
Create an Ansible Custom Task - 1/7

Sample Scenario: Deploy NGINX Webserver



Drag the **Invoke Ansible Playbook** executor in the designer.

Note: you can't use more than one executor type in a single custom task. However, you can use multiple executors of the same type.



Assign a name and a description to both executors

Create an Ansible Custom Task - 2/7

Sample Scenario: Deploy NGINX Webserver. Custom Task Properties, General

The screenshot displays the Ansible Tower interface. On the left, a workflow diagram shows a sequence of tasks: 'Start', 'Install NGINX' (Executors), 'Sync Site content' (Executors), and 'Success'. A 'Properties' button is visible at the top of the workflow. On the right, the 'Task Properties' dialog box is open, showing the 'General' tab. The 'Organization' is set to 'default', the 'Display Name' is 'Install NGINX With Ansible', and the 'Reference Name' is 'InstallNGINXWithAnsible'. The 'Description' field is empty. The 'Retry Count' is set to 3, 'Retry Delay' is 60, and 'Timeout' is 600. The 'Set Tags' section shows a tag 'category: Ansible'.

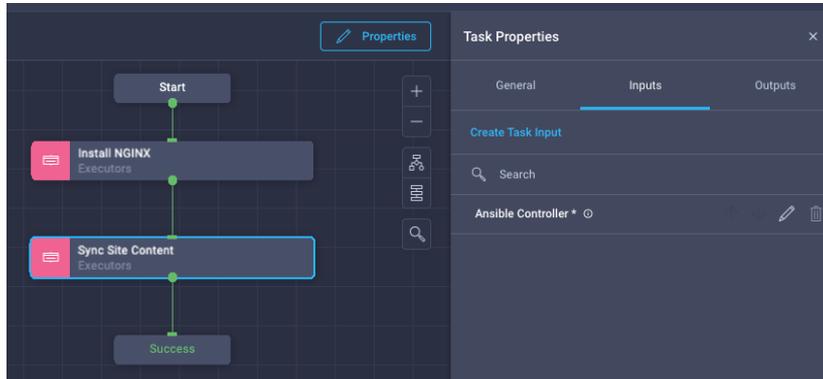
Click on **Properties** to access the custom task properties.

Under the **General** tab, select an **Organization** and specify a **Display Name**.

Optionally, set a category for this custom task using the **Tag** `category:Ansible`

Create an Ansible Custom Task - 3/7

Sample Scenario: Deploy NGINX Webserver. Custom Task Properties, Inputs



Click on **Create Task Input** to create inputs as we want to pass their values in the **Command Line Arguments** inputs of the executors.

The following table recaps the input required, their names and types

Note: these are specific to this example, if you want users to pass the ansible variables you will need to create inputs based on your implementation

Display Name	Reference Name	Type
Server Domain Name	domain	string
Target Endpoint Ansible User	ansuser	string
SSH Private Key File	sshprivkey	string
Target Endpoint Ansible SUDO Password	anssudopwd	string

Create an Ansible Custom Task - 4/7

Sample Scenario: Deploy NGINX Webserver. Executor: Install NGINX

Static Mapping of the playbook path in the claimed Ansible control node:
`/home/admin/ansible/nginx/nginx.yml`

Direct Mapping (create Task Input) as we want to run this playbook to an endpoint or list of endpoint specified in the parent workflow. We set the input type to be a **String** and enforce regex validation with `, $` to be sure we always have the trailing comma (see requirements slide)

Type of Mapping

- Static Value
- Direct Mapping**

Map the input to the task input, variable or any of the previous execut

Task Input Task Output Workflow Variable

Input Name *
Create Task Input

Ansible Controller

Add Workflow Input

Display Name *
Host Inventory

Reference Name *
HostInventory

Description
The path of the host inventory file tha

Value Restrictions

- Required
- Collection/Multiple

Type
String

Min 0 Max 0
Regex `, $`

Create an Ansible Custom Task - 5/7

Sample Scenario: Deploy NGINX Webserver. Executor: Install NGINX

The screenshot shows the Ansible Tower interface. On the left, a workflow diagram consists of four steps: 'Start', 'Install NGINX' (highlighted with a blue border), 'Sync Site Content', and 'Success'. On the right, the 'Install NGINX' task properties are displayed in a dark-themed panel. The 'Inputs' tab is active, showing a search bar and several input fields, all with the value 'VALUE NOT SPECIFIED'. The fields are: 'Playbook Path *', 'Host Inventory *', 'Command Timeout', and 'Command Line Arguments'. Each field has a 'Map' button to its right. A red arrow points from the 'Command Line Arguments' field to a text box on the right.

Static Mapping.

This executor input accepts the command line arguments as you would pass to the ansible-playbook CLI command. In this case the target playbook accepts variables so we will pass them using the **Command Line Arguments**:

```
-e "domain={{.global.task.input.domain}}
ansible_user={{.global.task.input.ansuser}}
ansible_ssh_private_key_file={{.global.task.inp
ut.sshprivkey}}
ansible_sudo_pass={{.global.task.input.anssudop
wd}}
ansible_python_interpreter=/usr/bin/python3"
```

Note that this is specific to the playbook you are using.

Also note that if you are using the Ansible Playbook Executor as an embedded task you are required to escape double quotes with the \ character

Please refer to the following documentation about using variables in Ansible Playbooks:

https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html

Create an Ansible Custom Task - 6/7

Sample Scenario: Deploy NGINX Webserver. Executor: Sync Site Content

The screenshot shows the Ansible workflow editor interface. On the left, a workflow diagram displays a sequence of tasks: 'Start', 'Install NGINX', 'Sync Site content', and 'Success'. The 'Sync Site content' task is highlighted with a blue border. On the right, the 'Properties' panel for this task is open, showing the 'Inputs' tab. The inputs listed are: 'Playbook Path *', 'Host Inventory *', 'Command Timeout', and 'Command Line Arguments'. Each input has a 'Map' button next to it. The values for 'Playbook Path' and 'Host Inventory' are currently 'VALUE NOT SPECIFIED'.

Static Mapping of the playbook path in the claimed Ansible control node:
`/home/admin/ansible/nginx/sync.yml`

Direct Mapping to the created workflow input `Host Inventory`

The screenshot shows a dialog box titled 'Map the input to the task input, variable or workflow input'. It has three radio buttons: 'Task Input' (selected), 'Task Output', and 'Workflow Input'. Below the radio buttons, there is a text input field labeled 'Input Name *' with a dropdown arrow. A list of suggestions is shown below the input field, including 'Create Task Input', 'Ansible Controller', and 'Host Inventory'.

Create an Ansible Custom Task - 7/7

Sample Scenario: Deploy NGINX Webserver. Executor: Sync Site Content

The screenshot shows the Ansible Tower interface. On the left, a workflow is visible with four tasks: 'Start', 'Install NGINX', 'Sync Site content', and 'Success'. The 'Sync Site content' task is highlighted with a blue border. On the right, the 'Properties' sidebar for the 'Sync Site content' task is open, showing the 'Inputs' tab. The inputs are:

Input	Type
Playbook Path *	Map
Host Inventory *	Map
Command Timeout	Map
Command Line Arguments	Map

Static Mapping.

This executor input accepts the command line arguments as you would pass to the ansible-playbook CLI command. In this case the target playbook accepts variables so we will pass them using the **Command Line Arguments**:

```
-e "domain={{.global.task.input.domain}}
ansible_user={{.global.task.input.ansuser}}
ansible_ssh_private_key_file={{.global.task.inp
ut.sshprivkey}}
ansible_sudo_pass={{.global.task.input.anssudop
wd}}
ansible_python_interpreter=/usr/bin/python3"
```

Note that this is specific to the playbook you are using.

Also note that if you are using the Ansible Playbook Executor as an embedded task you are required to escape double quotes with the \ character

Please refer to the following documentation about using variables in Ansible Playbooks:

https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html

Save

Save the task

Create a Custom Task to configure password-less SSH 1/2

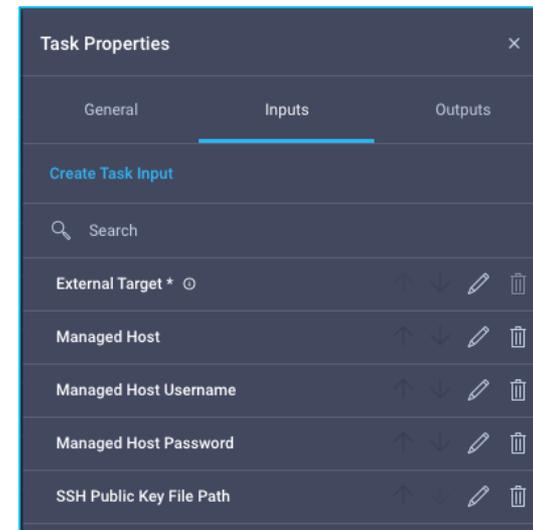
Sample Scenario: Deploy NGINX Webserver

Recall that in order to work, Ansible requires password-less SSH access to the targets it's executing against. If the Ansible controller already has password-less SSH access to the target, no additional steps are needed. However, there are cases where we want to target freshly created hosts that may not have granted access to the Ansible controller yet.

To overcome this issue, we will create a custom task using the SSH executor that injects the Ansible target public key into the endpoint hosts (refer to the SSH executor section to get more info on creating SSH custom tasks)

Inject SSH Key custom task inputs:

Display Name	Reference Name	Type
Managed Host	ManagedHost	string
Managed Host Username	ManagedHostUsername	string
Managed Host Password	ManagedHostPassword	string
SSH Public Key File Path	SSHPublicKeyName	string



Create a Custom Task to configure password-less SSH 2/2

Sample Scenario: Deploy NGINX Webserver

Type of Mapping

Static Value Direct Mapping Advanced Mapping

Provide custom values as the input.

SSH Command

Command *

```
ssh-copy-id -i {{ global.task.input.SSHPublicKeyName }}
```

Command Type

Non-Interactive Interactive

Expect Prompts

Expect *

```
password:\s$
```

Send

```
{{ global.task.input.ManagedHostPassword }}
```

Shell Prompt *

```
\s$
```

Command:

```
ssh-copy-id -i  
{{ global.task.input.SSHPublicKeyName }}  
{{ global.task.input.ManagedHostUsername }}@{{ global.task.input.ManagedHost }} -o  
"StrictHostKeyChecking no"
```

Command-Type: **Interactive**

Expect:

```
password:\s$
```

Send:

```
{{ global.task.input.ManagedHostPassword }}
```

Shell Prompt:

```
\s$
```

This task will simply execute the `ssh-copy-id` command in the Ansible target against another target host. From that point on, it will be able to have password-less SSH access. **Save** the task once done

Sample End to End Workflow

Sample Scenario: Deploy NGINX Webserver

The screenshot shows a workflow editor with a vertical sequence of tasks: Start, New Virtual Machine from Tem..., Sleep Task, Get VMware Virtual Machine G..., Inject SSH Key, and Install NGINX with Ansible. The 'Inject SSH Key' task is highlighted with a red box. To its right, a configuration panel titled 'Inject SSH Key' is open, showing the 'Inputs' tab. The inputs are:

Input Name	Value
External Target *	Custom Value
Managed Host *	Custom Value
Managed Host Username *	admin
Managed Host Password *	C1sco123
SSH Public Key File Path *	/home/admin/.ssh/id_rsa

The screenshot shows the same workflow editor as the first image, but with the 'Install NGINX with Ansible' task highlighted with a red box. To its right, a configuration panel titled 'Install NGINX with Ansible' is open, showing the 'Inputs' tab. The inputs are:

Input Name	Value
Ansible Controller *	Custom Value
Server Domain Name	demo-intersight.com
Target Endpoint Ansible User	admin
SSH Private Key File	/home/admin/.ssh/id_rsa
Target Endpoint Ansible SUDO Password	C1sco123
Host Inventory *	Custom Value

Sample End to End Workflow

Sample Scenario: Deploy NGINX Webserver

Enter Workflow Input - Demo Ansible

Organization *
default

Workflow Instance Name
Demo Ansible

Virtual Machine *
ico-webserver

Cancel Execute

Rollback Clone Execution

Execution Demo Ansible - Apr 14, 2022 3:18 PM

Organization default

Status Success

Start

New Virtual Machine from Tem...
Virtualization

Sleep Task
Code Tasks

Get VMware Virtual Machine G...
Virtualization

Inject SSH Key
General

Install NGINX with Ansible
General

Success Failed

Inject SSH Key
Apr 14, 2022 03:22:10 PM

Install NGINX with Ansible
Apr 14, 2022 03:23:37 PM

State: Ok

Type: Config

Virtual Machine Guest IP: 192.168.130.168

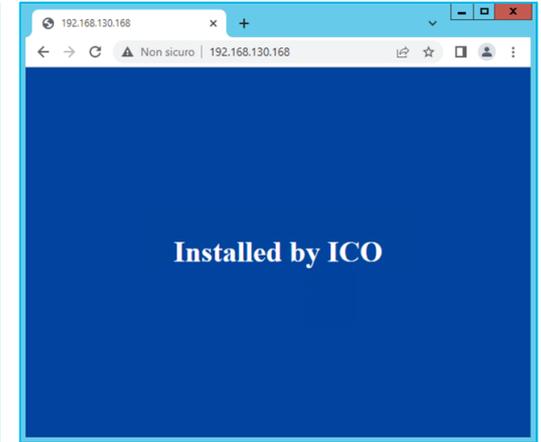
Object: (4)

- ConfigResCtx: (1)
- EntityData: (1)
- task: workflow.AnsibleTask

Message: Ansible playbook execution completed. Please refer the execution logs /home/admin/ansible/nginx/ansible_2022-04-14_15-23-07.log

State: Ok

Type: Config



Go Template Cheat Sheet

Referencing variables in the Workflow Designer

Usage	Description	Code
Workflow Input	You can reference a workflow input by replacing NAME with the the reference name you have given it inside the workflow designer	<code>{{.global.workflow.input.NAME}}</code>
Task Output	You can reference a task output by replacing TASK with the name of the task (from the Code view), and by replacing NAME with the name of the output	<code>{{.global.TASK.output.NAME}}</code>
Org Moid	The Moid of the Intersight org used to execute this workflow	<code>{{.security.OrganizationMoid}}</code>

Referencing variables in the Task Designer

Usage	Description	Code
Task Input	You can reference a task input by replacing NAME with the the reference name you have given it inside the task designer	<code>{{.global.task.input.NAME}}</code>
Sub-Task Output	You can reference a sub-task output by replacing SUBTASK with the reference name of the sub-task, and by replacing NAME with the name of the output	<code>{{.global.SUBTASK.output.NAME}}</code>

Advanced functions 1/3

Usage	Description	Code
Conditional	Will allow you to select what option you want to execute based on a conditional. Replace VALUE with the specific value you want to compare against.	<pre>{{if (eq .global.workflow.input.example VALUE)}} IF TRUE {{else if eq .global.workflow.input.example VALUE}} ELIF TRUE {{else}} ELSE {{end}}</pre>
Optional variable	Only executes a block of the template if the variable was provided in the input. Inside of the with block, you refer to the name of the original variable as \$x (or whatever variable you use)	<pre>{{with \$x := .global.workflow.input.example}} {{\$x}} {{end}}</pre>

Advanced functions 2/3

Usage	Description	Code
Regex	Apply a regex pattern to a string to filter out a sub-string. Replace REGEX with your regex pattern. This can be combined with the index feature below to filter out the n-th sub-string.	<code>{{FindAllString .global.workflow.input.example "REGEX"}}</code>
Get n-th entry	Allows you to get the n-th entry from a list. Replace NUMBER with the position of the list item you want to select.	<code>{{index .global.workflow.input.example NUMBER}}</code>
Sub-parameter	Allows you to select the subparameter of a previous operation. Replace SUBELEMENT with the parameter you are looking for	<code>{{(index .global.workflow.input.example NUMBER).SUBELEMENT}}</code>

Advanced functions 3/3

Usage	Description	Code
Loop	Allows you to loop through a list and execute a template for each entry in the list. If you want to prepend something to your list, you can use if \$index to only apply on the first pass of the loop	<pre>{{range \$i, \$element := .global.workflow.input.example}} {{ \$element }} {{end}}</pre>
List length	Get the length of a list	<pre>{{len .global.workflow.input.example}}</pre>

ICO Intersight API

Intersight APIs

Intersight APIs are documented here:

<https://intersight.com/apidocs/introduction/overview/>

The Intersight API is based on the OpenAPI Specification standard:

<https://github.com/OAI/OpenAPI-Specification/blob/main/versions/3.0.3.md>

Requirements:

- Basic knowledge of REST APIs and JSON
- Basic knowledge of Postman
- Create an API and Secret Key in Intersight
- APIs can be invoked/tested also from the embedded REST client here:
<https://intersight.com/apidocs/apirefs/aaa/AuditRecords/model/>
- Intersight APIs requires all requests to be signed. The following GitHub repo provides a Postman collection that will take care of it automatically
<https://github.com/CiscoDevNet/intersight-postman>
- Examples in this document use the following postman collection (fork from the DevNet repo):
<https://github.com/rtortori/intersight-postman/tree/ico-workflows>

ICO Intersight APIs

ICO APIs are generally under the **workflow** path : /api/v1/workflow

They can be explored and tested from

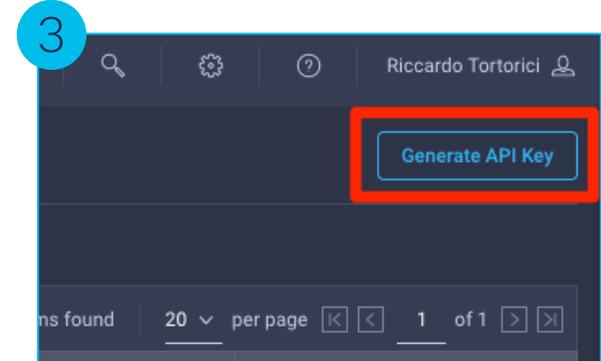
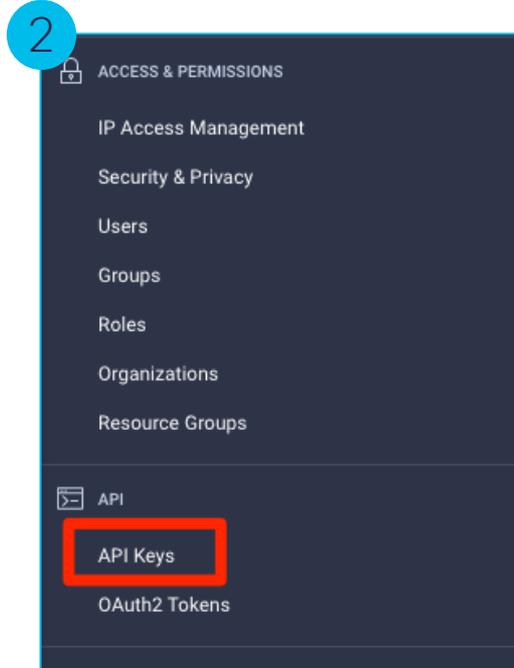
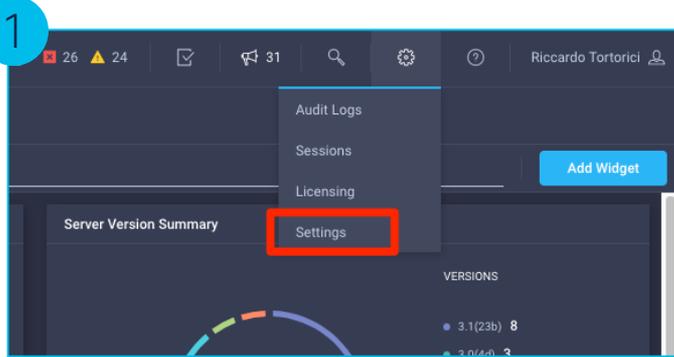
<https://intersight.com/apidocs/apirefs/aaa/AuditRecords/model/>

You will need to be logged in to use the REST Client.

In this document, the following APIs will be covered:

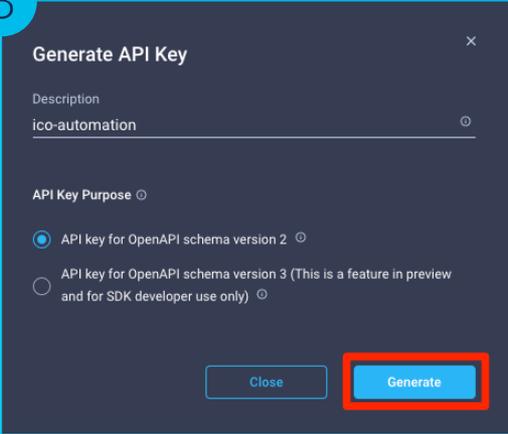
Scope	Description	Path
Workflows	Workflows Definitions	/api/v1/WorkflowDefinitions
Workflows	Workflow Executions (Requests)	/api/v1/WorkflowInfos
Workflows	Executions Rollbacks	/api/v1/RollbackWorkflows
Tasks	Task info on Workflow Executions (i.e. outputs)	/api/v1/TaskInfos
Tasks	Task Definitions	/api/v1/TaskDefinitions

Create an API and Secret Key in Intersight 1/2



Create an API and Secret Key in Intersight 2/2

3



Generate API Key [Close]

Description
ico-automation [Clear]

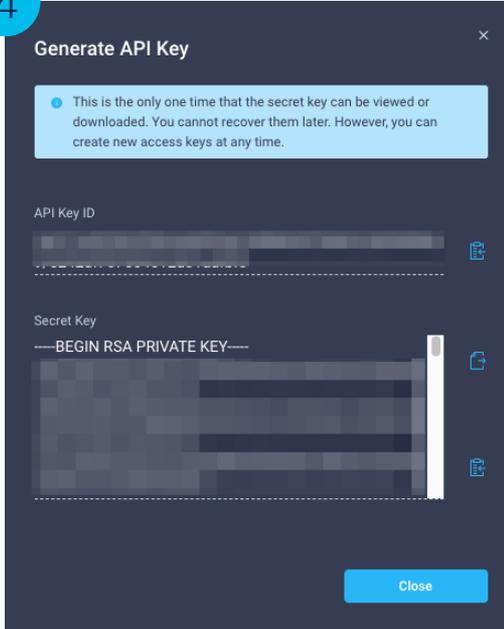
API Key Purpose [Clear]

API key for OpenAPI schema version 2 [Clear]

API key for OpenAPI schema version 3 (This is a feature in preview and for SDK developer use only) [Clear]

[Close] **Generate**

4



Generate API Key [Close]

This is the only one time that the secret key can be viewed or downloaded. You cannot recover them later. However, you can create new access keys at any time.

API Key ID
[Redacted] [Copy]

Secret Key
-----BEGIN RSA PRIVATE KEY-----
[Redacted] [Copy]

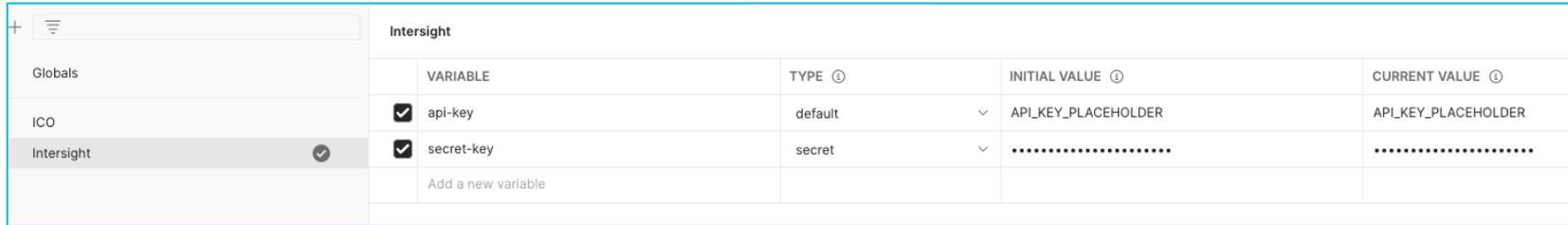
[Close]

Take note of the API Key ID and Secret Key

Note:
This will be the only time the **Secret Key** will be shown. If you lose it, you'll need to generate another API key

Postman Setup

1. Download and Import the collection in Postman
2. Set the Api Key and Secret Key in the **Intersight** Postman Environment and **Save**



The screenshot shows the Postman environment configuration for 'Intersight'. On the left, a sidebar lists 'Globals', 'ICO', and 'Intersight' (selected with a checkmark). The main area displays a table of environment variables for the 'Intersight' environment.

Intersight				
	VARIABLE	TYPE ⓘ	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ
<input checked="" type="checkbox"/>	api-key	default	API_KEY_PLACEHOLDER	API_KEY_PLACEHOLDER
<input checked="" type="checkbox"/>	secret-key	secret
	Add a new variable			

Example: Get ICO Workflow List

The screenshot displays a REST client interface for a GET request. The sidebar on the left shows a tree view with the following structure:

- Intersight - ICO
 - ICO
 - Intersight-Examples
 - Advisories
 - Device Claiming
 - Api Key Management
 - Alarms
 - Search Servers
 - Create and Assign NTP Policy
 - Ntp Cleanup
 - Turn on Server Locator LED
 - Testing
 - Cloud Orchestrator
 - Workflows
 - Get ICO Workflows List
 - Execute ICO Workflow
 - Workflows Executions (Requests)
 - Get ICO Workflow Definition
 - Workflows Executions (Requests)
 - Get ICO Workflow Execution Details
 - Get ICO Workflow Executions List
 - Get ICO Execution Tasks Details
 - Get ICO Execution Tasks Outputs
 - Rollback
 - 1. Create Rollback for Execution
 - 2. Rollback ICO Workflow Execution
 - Tasks
 - Get Tasks List
 - Get Contract Status
 - Get Audit Record

The main interface shows the following details:

- Method:** GET
- URL:** `https://(server)/api/v1/workflow/WorkflowDefinitions?$filter=(Properties.ExternalMeta eq true) and (Name ne 'DeployHyperFlexSDWAN') and (DefaultVersion eq true)`
- Query Params:**

KEY	VALUE	DESCRIPTION
<input type="checkbox"/> \$select	Label	
<input checked="" type="checkbox"/> \$filter	(Properties.ExternalMeta eq true) and (Name ne 'DeployHyperFlexSDWAN	
Key	Value	Description

- Body:** Pretty, Raw, Preview, Visualize, JSON
- Status:** 200 OK, Time: 967 ms, Size: 1.36 MB
- Response (JSON):**

```
1 |
2 | "ObjectType": "workflow.WorkflowDefinition.List",
3 | "Results": [
4 |   {
5 |     "AccountMoid": "5a871b403362396c6a6854ae",
6 |     "Ancestors": [
7 |       {
8 |         "ClassId": "mo.MoRef",
9 |         "Moid": "5c81de2c696f6e2d3028226c",
10 |        "ObjectType": "workflow.Catalog",
11 |        "link": "https://www.intersight.com/api/v1/workflow/Catalogs/5c81de2c696f6e2d3028226c"
12 |      }
13 |    ],
14 |    "Catalog": {
15 |      "ClassId": "mo.MoRef",
16 |      "Moid": "5c81de2c696f6e2d3028226c",
17 |      "ObjectType": "workflow.Catalog",
18 |      "link": "https://www.intersight.com/api/v1/workflow/Catalogs/5c81de2c696f6e2d3028226c"
19 |    },
20 |    "ClassId": "workflow.WorkflowDefinition",
21 |    "ClonedFrom": null,
22 |    "CreateTime": "2020-06-30T17:59:24.08Z",
23 |    "DefaultVersion": true,
24 |    "Description": "a test workflow to experiment with custom integrations",
25 |    "ExternalMeta": true,
26 |    "Name": "Test Workflow",
27 |    "ParentMoid": "5c81de2c696f6e2d3028226c",
28 |    "Properties": {
29 |      "ExternalMeta": true,
30 |      "Name": "Test Workflow",
31 |      "ParentMoid": "5c81de2c696f6e2d3028226c",
32 |      "Version": 1
33 |    },
34 |    "Version": 1
35 |  }
36 | ]
```

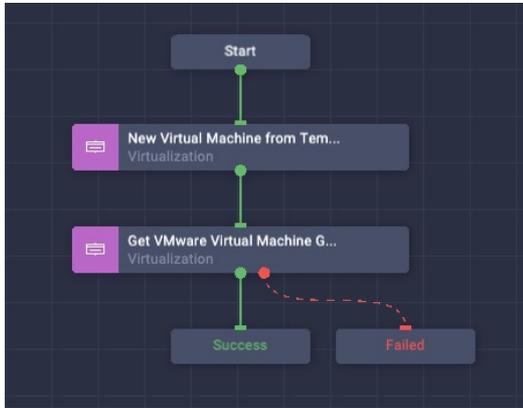
Example: Execute a Workflow 1/4

A popular use case is to execute a workflow using the API from a 3rd party application, orchestrator or service catalog.

While the payload structure is always the same, depending on which workflow you want to execute, you may have different inputs.

A possible strategy would be to invoke your workflow manually and capture the **POST** call done by the browser. This is an action you only have to do once and will help you shaping your call (i.e. a service catalog item creation).

In this case, we are executing a Workflow named **Deploy a Virtual Machine**



This workflow has two tasks:

1. Deploys a Virtual Machine
2. Query the vCenter to extract the IP of the VM

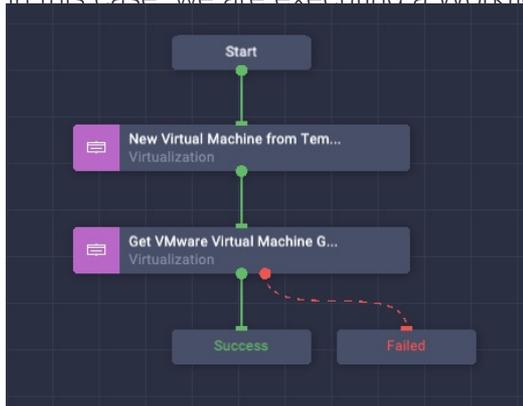
Example: Execute a Workflow 1/5

A popular use case is to execute a workflow using the API from a 3rd party application, orchestrator or service catalog.

While the payload structure is always the same, depending on which workflow you want to execute, you may have different inputs.

A possible strategy would be to invoke your workflow manually and capture the **POST** call done by the browser. This is an action you only have to do once and will help you shaping your call (i.e. a service catalog item creation).

In this case, we are executing a Workflow named **Deploy a Virtual Machine**



This workflow has two tasks:

1. Deploys a Virtual Machine
2. Query the vCenter to extract the IP of the VM

Click on **Execute** and fill in with your inputs as you were deploying the workflow manually.

DO NOT launch the workflow execution once done but keep the window opened

The screenshot shows a form titled 'Enter Workflow Input - Deploy A Virtual Machine'. The form contains the following fields and values:

- Organization *: default
- Workflow Instance Name: Deploy a Virtual Machine
- Hypervisor Manager *: Selected Hypervisor Manager: blackmore.rmlab.local
- Datcenter *: Selected Datcenter: RMLAB
- Host *: Select Host
- Cluster *: Selected Cluster: Goldrake
- Datastore *: Selected Datastore: Hyperflex1
- Folder *: Selected Folder: rtortori
- Template or Virtual Machine *: /RMLAB/vm/rtortori/ubuntu-template-hx
- Virtual Machine *: my-virtual-machine
- Network *: Selected Network: 192.168.130

At the bottom of the form, there are two buttons: 'Cancel' and 'Execute'.

Example: Execute a Workflow 2/5

Open the developer tools of your browser and go to the **Network** tab

The screenshot displays the Cisco Intersight web interface. The main window shows a configuration page for 'Deploy a Virtual Machine' with a modal dialog titled 'Enter Workflow Input - Deploy A Virtual Machine'. The dialog contains the following fields and options:

- Organization: default
- Workflow Instance Name: Deploy a Virtual Machine
- Hypervisor Manager: Selected Hypervisor Manager: blackmore.mlab.local
- Datacenter: Selected Datacenter: RMLAB
- Host: Select Host

Buttons for 'Cancel' and 'Execute' are visible at the bottom of the dialog. The background interface includes a left sidebar with navigation options like 'MONITOR', 'OPERATE', and 'CONFIGURE', and a top navigation bar with 'Orchestration' and 'Deploy a Virtual Machine'.

Below the Intersight interface, the browser's developer tools are open to the 'Network' tab. The 'Filter' bar shows 'invert', 'Hide data URLs', 'FetchXHR', 'JS', 'CSS', 'img', 'Media', 'Font', 'Doc', 'WS', 'Wasm', 'Manifest', and 'Other'. The network activity area is currently empty, displaying the message: 'Recording network activity... Perform a request or hit **⌘ R** to record the reload. [Learn more](#)'.

Example: Execute a Workflow 3/5

The screenshot displays the Cisco Intersight interface. The top navigation bar shows 'CONFIGURE > Orchestration > Deploy a Virtual Machine > Edit'. The left sidebar has 'MONITOR' and 'OPERATE' sections, with 'Orchestration' selected. The main area shows a workflow diagram with steps: 'Start', 'New Virtual Machine from Tem...', and 'Get VMware Virtual Machine G...'. A 'Cancel Execution' button is visible. The bottom browser window shows the 'Network' tab with a 'Workflows' entry selected in the filter. The 'Request Payload' is expanded, showing the following JSON:

```
{
  "Name": "Deploy a Virtual Machine...",
  "AssociatedObject": {
    "ObjectType": "organization.Organization",
    "Moid": "5ddee8b6972652d3183b0af"
  },
  "Input": {
    "vCenter": {
      "Moid": "68a033446f726124369f5d",
      "ObjectType": "asset.DeviceRegistration",
      ...
    }
  },
  "Name": "Deploy a Virtual Machine",
  "WorkflowCic": {
    "InitiatorCic": {
      "InitiatorMoid": "6192e224696f6a2d3124372e",
      "InitiatorName": "Deploy a Virtual Machine",
      ...
    }
  },
  "WorkflowDefinition": {
    "Moid": "6192e224696f6a2d3124372e",
    "ObjectType": "workflow.WorkflowDefinition"
  }
}
```

Open the developer tools of your browser and go to the **Network** tab
Copy the payload

Example: Execute a Workflow 4/5

This is the payload to be used to invoke the workflow:



Replace inputs based on what you want to create. For instance, change the Virtual Machine **Name**

Example: Execute a Workflow 5/5

Copy the payload in the **Body** tab of the request **Intersight-Examples -> Cloud Orchestrator -> Workflows -> 2. Execute ICO Workflow**

Click **Send** to execute the workflow

```
POST https://[server]/api/v1/workflow/WorkflowInfos

{
  "Name": "Deploy a Virtual Machine",
  "AssociatedObject": {
    "ObjectType": "organization.Organization",
    "Moid": "5ddee8bb697262d31830ba1"
  },
  "Action": "Start",
  "Input": {
    "Vcenter": {
      "Moid": "66a833446f72612d33e9f5df",
      "ObjectType": "asset.DeviceRegistration"
    },
    "Datacenter": "/RMLAB",
    "Cluster": "/RMLAB/host/Goldrake",
    "Datastore": "/RMLAB/datastore/Hyperflex1",
    "Folder": "/RMLAB/vm/rtortori",
    "Template": "/RMLAB/vm/rtortori/ubuntu-template-hx",
    "Name": "test-ico-api4",
    "Network": "192.168.130"
  },
  "WorkflowDefinition": {
    "Moid": "6192e224696f6e2d3124372e",
    "ObjectType": "workflow.WorkflowDefinition"
  }
}
```

In this postman collection, we will capture the **Moid** of the execution and save it in a global variable named **execution-moid**

We can then use this Moid to invoke other actions, such as get the workflow tasks output and rollback the request (next slides)

```
POST https://[server]/api/v1/workflow/WorkflowInfos

const response = pm.response.json();
pm.globals.set('execution-moid', response.Moid);
```

Example: Get Workflow Outputs

The screenshot shows a REST client interface with a GET request to `https://(server)/api/v1/workflow/WorkflowInfo/(execution-moid)`. The response body is displayed in JSON format. A red box highlights the `execution-moid` in the URL, and another red box highlights the `vmip` field in the response body.

```
1 |
2 | {
3 |   "Account": null,
4 |   "AccountMoid": "5a871b403362396c5a68545e",
5 |   "Action": "None",
6 |   "Accessors": [],
7 |   "AssociatedObject": {
8 |     "ClassId": "mo.MoRef",
9 |     "Moid": "5d5ee8b6972652d31030ba8",
10 |    "ObjectType": "organization.Organization",
11 |    "link": "https://www.intersight.com/api/v1/organization/Organizations/5d5ee8b6972652d31030ba8"
12 |  },
13 |   "ClassId": "workflow.WorkflowInfo",
14 |   "CleanupTime": "2022-06-29T12:51:37.128Z",
15 |   "CreateTime": "2022-03-29T12:49:28.398Z",
16 |   "DomainGroupMoid": "5b2541987a766274346509e9",
17 |   "Email": "tortorzi@isc.com",
18 |   "EndTime": "2022-03-29T12:51:37.128Z",
19 |   "FailedWorkflowCleanupDuration": 2160,
20 |   "Input": {
21 |     "Cluster": "/RMLAB/host/Goldrake",
22 |     "Datacenter": "/RMLAB",
23 |     "Datastore": "/RMLAB/datastore/Hyperflex1",
24 |     "Follows": "/RMLAB/vm/virt001",
25 |     "Name": "test-ico-vm01",
26 |     "Network": "192.168.130",
27 |     "Template": "/RMLAB/vm/tortorzi/ubuntu-template-hx",
28 |     "vcenter": {
29 |       "Moid": "66a83346f72612d33e9f5d1",
30 |       "ObjectType": "asset.DeviceRegistration"
31 |     }
32 |   },
33 |   "InstId": "8a190eb1-af64-4561-a595-8863bd5de17e",
34 |   "Internal": false,
35 |   "LastAction": "Start",
36 |   "Message": [],
37 |   "MetaVersion": 0,
38 |   "ModTime": "2022-03-29T12:51:37.128Z",
39 |   "Moid": "62a3909896f0e203284305",
40 |   "Name": "Deploy a Virtual Machine",
41 |   "ObjectType": "workflow.WorkflowInfo",
42 |   "Organization": null,
43 |   "Output": {
44 |     "vmip": "192.168.130.147"
45 |   }
46 | }
```

We use the **execution-moid** extracted from the workflow execution call

We fetch the output of the workflow, in this case the Virtual Machine IP

Example: Get the List of the Workflow Executions

The screenshot shows a REST client interface for a GET request. The URL is `https://(server)/api/v1/workflow/WorkflowInfos?$select=Name,Status,Moid,Email,StartTime&orderby=StartTime desc`. The query parameters table is as follows:

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> \$select	Name,Status,Moid,Email,StartTime	
<input checked="" type="checkbox"/> \$orderby	StartTime desc	

The response body is a JSON array of workflow execution objects. The last execution is highlighted with a red box:

```
1 {
2   "ObjectType": "workflow.WorkflowInfo.List",
3   "Results": [
4     {
5       "ClassId": "workflow.WorkflowInfo",
6       "Email": "rtortori@cisco.com",
7       "Moid": "62430058696f6e2d312843b5",
8       "Name": "Deploy a Virtual Machine",
9       "ObjectType": "workflow.WorkflowInfo",
10      "StartTime": "2022-03-29T12:49:28.459Z",
11      "Status": "COMPLETED"
12    },
13    {
14      "ClassId": "workflow.WorkflowInfo",
15      "Email": "rtortori@cisco.com",
16      "Moid": "6242f2f2696f6e2d312834a9",
17      "Name": "Deploy a Virtual Machine",
18      "ObjectType": "workflow.WorkflowInfo",
19      "StartTime": "2022-03-29T12:27:27.796Z",
20      "Status": "COMPLETED"
21    }
22  ]
23 }
```

We only show the Name, Status, Moid, Email and StartTime (\$select)

We sort the results by StartTime in descending (desc) order

Last execution, the virtual machine deployment we triggered earlier

Example: Get Workflow Tasks Outputs

The screenshot shows a REST client interface with the following details:

- URL:** `https://[server]/api/v1/workflow/TaskInfos?select=Label,Output&filter=(WorkflowInfo.Moid eq '(execution-moid))`
- Method:** GET
- Params:** Authorization, Headers (6), Body, Pre-request Script, Tests, Settings
- Query Params:** \$select, \$filter (WorkflowInfo.Moid eq '(execution-moid))
- Status:** 200 OK, Time: 242 ms, Size: 3.9 KB
- Response Body (JSON):**

```
1 {
2   "ObjectType": "workflow.TaskInfo.List",
3   "Results": [
4     {
5       "ClassId": "workflow.TaskInfo",
6       "Label": "New Virtual Machine from Template or Clone from Virtual Machine",
7       "Moid": "62430058696f6e2d312843fe",
8       "ObjectType": "workflow.TaskInfo",
9       "Output": {
10        "ConfigResults": [
11          {
12            "ConfigResCtx": {
13              "Message": "Virtual Machine 'test-ico-api4' created successfully.",
14              "MessageGeneration": "None",
15              "Moid": "",
16              "State": "OK",
17              "Type": "Config"
18            }
19          }
20        ],
21        "Folder": "/RMLAB/vm/rtortori",
22        "Name": "/RMLAB/vm/rtortori/test-ico-api4",
23        "TaskId": "task-398425",
24        "Template": "/RMLAB/vm/rtortori/ubuntu-template-hv"
25      }
26    },
27    {
28      "ClassId": "workflow.TaskInfo",
29      "Label": "Get VMware Virtual Machine Guest IP",
30      "Moid": "62430058696f6e2d31284484",
31      "ObjectType": "workflow.TaskInfo",
32      "Output": {
33        "ConfigResults": [
34          {
35            "vmip": "192.168.130.147"
36          }
37        ]
38      }
39    },
40    {
41      "ClassId": "workflow.TaskInfo",
42      "Label": "",
43      "Moid": "62430058696f6e2d312843dd",
44      "ObjectType": "workflow.TaskInfo",
45      "Output": {}
46    }
47  ]
48 }
```

Example: Rollback an Execution (Request) 1/3

In order to rollback an execution you need to know the **Moid** of the request. You can use the **execution-moid** variable extracted earlier or you can use the **Get ICO Workflow Execution Details** postman request to select the execution you want to rollback.

To rollback, you need to make two API Calls:

1. Create the rollback
2. Execute the rollback

Example: Rollback an Execution (Request) 3/3

The image displays a REST client interface with three main panels:

- Request Editor (Top):** Shows a POST request to `https://{(server)}/api/v1/workflow/RollbackWorkflows`. The body is a JSON object with the following structure:

```
1 {
2   "PrimaryWorkflow":
3   {
4     "ObjectType": "workflow.WorkflowInfo",
5     "Moid": "{{execution-moid}}"
6   }
7   "Action": "Start",
8   "ContinueOnError": true,
9   "SelectedTasks":
10  []
11 }
```
- Request List (Middle):** A table showing the execution status of requests:

Name	Status	Initiator	Target Type	Target Name
Rollback Deploy a Virtual Ma...	In Progress 0%	rtortori@cisico.com	Registered Device	blackmore.rmlab.local
Deploy a Virtual Machine	Success	rtortori@cisico.com	Registered Device	blackmore.rmlab.local
- Response Viewer (Bottom):** Shows the response for the "Rollback Deploy a Virtual Machine" request, which is a success. The execution flow includes a task "Remove Virtual Machine" with a log message: `"Message": "Virtual machine 'test-ico-api4' deleted successfully."`

Execute an ICO Workflow from Terraform

(Provider version 1.0.11)

Get the Target Workflow MOID (The Easiest Way)

The screenshot shows a web browser window with the following details:

- Browser tab: "Orchestration Claim HTTP Targ x +"
- Address bar: `https://intersight.com/an/workflow/workflow-definitions/edit/60d34618696f6e2d30ec43fa`. The MOID `60d34618696f6e2d30ec43fa` is highlighted with a red box.
- Page header: "Intersight" logo and breadcrumb "CONFIGURE > Orchestration > Claim HTTP Target > Edit".
- Left sidebar: "MONITOR" and "OPERATE" sections. Under "OPERATE", there are links for "Servers", "Chassis", "Fabric Interconnects", and "Networking Sites".
- Main content area: Tabs for "General", "Designer", "Mapping", "Code", and "History". The "Designer" tab is active. Below it is a "Tools" section with sub-tabs "Tasks", "Workflows", and "Operations". A search bar is present. Under "Executors", there is a button labeled "Invoke Web API Request".
- Bottom right: A yellow notification icon and a partially visible "Clai Gen" button.

Get the Target Workflow MOID (The API Way)

The screenshot displays the Cisco Intersight API documentation for the endpoint `/api/v1/workflow/WorkflowDefinitions/get/`. The `API Reference` tab is active, showing the `GET` method for reading a `'workflow.WorkflowDefinition'` resource. The left sidebar lists various endpoints, with `workflow/WorkflowDefinitions` selected. The main content area details the endpoint's parameters, including `$filter` (a query parameter for filtering resources), `$orderby` (for sorting), `$top` (for limiting results), `$skip` (for skipping results), and `$select` (for selecting specific properties). A REST client simulation is shown on the right, with the `$select` parameter set to `'Moid'` and the `$filter` parameter set to `Label eq 'Claim HTTP Target'`. The simulation shows a successful `200 Success` response with the following JSON:

```
1 {
2   "ObjectType": "workflow.WorkflowDefinition.List",
3   "Results": [
4     {
5       "classId": "workflow.WorkflowDefinition",
6       "Moid": "60d34618696f6e2d30ec43fa",
7       "ObjectType": "workflow.WorkflowDefinition"
8     }
9   ]
10 }
```

API Docs: <https://www.intersight.com/apidocs>

Terraform Resource

The screenshot shows the 'Edit' page for a 'Claim HTTP Target' resource in the Intersight configuration tool. The 'General' tab is active, displaying various settings like 'Set as Default Version', 'Retryable', and 'Enable Debug Logs'. Below these are sections for 'Workflow Inputs' and 'Workflow Outputs'. An 'Add Input' button is visible. A red box highlights the input fields: 'name*', 'endpoint*', and 'webexroomid'. A red arrow points from this box to the corresponding Terraform code on the right.

```
resource "intersight_workflow_workflow_info" "myworkflow" {  
  name = var.workflowname  
  action = "Start"  
  
  input = {  
    "name" = "from_terraform"  
    "endpoint" = "httpbin.org"  
    "webexroomid" = var.roomid  
  }  
  
  organization {  
    object_type = "organization.Organization"  
    moid = data.intersight_organization_organization.myorg.id  
  }  
  
  workflow_definition {  
    object_type = "workflow.WorkflowDefinition"  
    moid = var.workflow_moid  
  }  
}
```

Specify your input variables as if you were executing the workflow from the UI. Ensure you respect types and constraints.

Provider documentation:

https://registry.terraform.io/providers/CiscoDevNet/intersight/latest/docs/resources/workflow_workflow_info

Sample code and instructions here:
<https://github.com/rtortori/intersight-terraform/tree/main/workflows>

Terraform Apply

```
→ workflows git:(main) > terraform apply --auto-approve
```

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
+ create

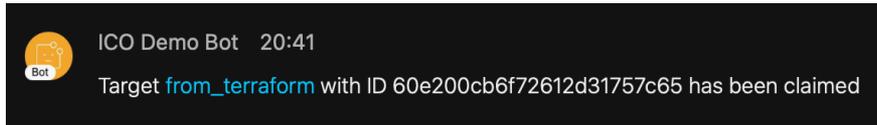
Terraform will perform the following actions:

```
# intersight_workflow_workflow_info.myworkflow will be created
+ resource "intersight_workflow_workflow_info" "myworkflow" {
[...]
```

```
Plan: 1 to add, 0 to change, 0 to destroy.
intersight_workflow_workflow_info.myworkflow: Creating...
intersight_workflow_workflow_info.myworkflow: Creation complete after 1s [id=60e200ca696f6e2d30f9a742]
```

```
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

```
→ workflows git:(main) >
```



	Name	Status	Type	IP Address	Claimed Time
<input type="checkbox"/>	from_terraform	Claimed	HTTP Endpoint	httpbin.org	2 minutes ago

```
terraform.tfstate x
workflows > terraform.tfstate > [ ]resources > [ ]1 > [ ]instances > [ ]0 > [ ]attribut
102     "selector": ""
103   },
104 ],
105   "class_id": "workflow.WorkflowInfo",
106   "cleanup_time": "0001-01-01 00:00:00 +0000 UTC",
107   "create_time": "2021-07-04 18:41:14.254 +0000 UTC",
108   "domain_group_moid": "5b2541987a7662743465d0e9",
109   "email": "rtortori@cisco.com",
110   "end_time": "0001-01-01 00:00:00 +0000 UTC",
111   "failed_workflow_cleanup_duration": 2160,
112   "id": "60e200ca696f6e2d30f9a742",
113   "input": {
114     "endpoint": "httpbin.org",
115     "name": "from_terraform",
116     "webexroomid": "Y2lzY29zcGFyazovL3VzL1JPT00vNTBjMmExYz
117   },
118   "inst_id": "ed59dc7d-1d34-46a5-8e5f-d0939b48d368",
119   "internal": false,
120   "last_action": "Start",
```

Terraform Destroy

Name	Status	Initiator	Target Type	Target N...	Start Time	Duration	ID
Claim HTTP Target	Success	rtortori@cisco.com	-	-	5 minutes ago	3 s	60e200ca696f6e2d30f9a742
Claim HTTP Target	Failed	rtortori@cisco.com	-	-	20 hours ago	3 m 14 s	60e0e283696f6e2d30cf211d

```
→ workflows git:(main) X > terraform destroy -auto-approve
intersight_workflow_workflow_info.myworkflow: Refreshing state... [id: 60e200ca696f6e2d30f9a742]
```

[...]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
- destroy

Terraform will perform the following actions:

```
# intersight_workflow_workflow_info.myworkflow will be destroyed
- resource "intersight_workflow_workflow_info" "myworkflow" {
```

[...]

```
Plan: 0 to add, 0 to change, 1 to destroy.
intersight_workflow_workflow_info.myworkflow: Destroying... [id=60e200ca696f6e2d30f9a742]
intersight_workflow_workflow_info.myworkflow: Destruction complete after 1s
```

```
Destroy complete! Resources: 1 destroyed.
```

```
→ workflows git:(main) X >
```

As of July 2021, terraform destroy will delete the request in Interact but will NOT rollback the workflow execution (in this example the target will be still claimed)

Name	Status	Initiator	Target Type	Target N...	Start Time	Duration	ID
Claim HTTP Target	Failed	rtortori@cisco.com	-	-	21 hours ago	3 m 14 s	60e0e283696f6e2d30cf211d

Execute an ICO Workflow
with the Python SDK

Requirements

- Knowledge of Python and Intersight
- Python Installed
- An Intersight API key and Secret
- Connectivity to intersight.com (or your CVA/PVAPP if you are running on-prem)

Documentation and examples:

<https://github.com/CiscoDevNet/intersight-python>

Python Examples described in this document:

<https://github.com/rtortori/ico-python-examples>

Install the Python SDK

Guides API Reference **Downloads** Support

Downloads

Intersight provides a downloadable Software Development Kit (SDK) for the Python programming language. You can generate SDKs for other programming languages using the open-source OpenAPI tools. The tables below provide you the links to download the SDKs and other Resources.

SDKs/Plugins

Title	Description	Installation	Examples
Intersight Python SDK	Python Software Development Kit based on OpenAPI schema version 3	Pypi	DevNet Python on GitHub
Intersight Ansible Modules (Python)	Ansible Modules for configuration management of Cisco Intersight	Galaxy	DevNet Playbooks on GitHub
Terraform Provider for Intersight	This provider facilitates the use of Terraform for managing infrastructure as code in Intersight	Terraform	Terraform Registry Modules
Intersight Powershell Module	Intersight Powershell Cmdlets based on OpenAPI schema version 3	PSGallery	DevNet PowerShell on GitHub
Intersight ITSM plugin for ServiceNow	This plugin facilitates the use of ServiceNow for all its ITSM requirements for Intersight	ServiceNow	—

<https://intersight.com/apidocs/downloads/>

Working with ICO and the Python SDK

Clone the GitHub repo: <https://github.com/rtortori/ico-python-examples>

This repo includes code that can be used and reworked to address a number of use cases.

This document describes just a few of them, refer to the repository for additional info and instructions.

In order to authenticate to the Intersight API, in the repo is present a file called `credentials.py`

This Python module contains the necessary code to authenticate to Intersight, provided you have generated an API Key and Secret:

https://intersight.com/apidocs/introduction/security/%23generating-api-keys&sa=D&ust=1612024909729000&usg=AOvVaw362rkbFqxhX_Mo8w0xkDJG/

You will need to import that module in order to authenticate to the Intersight API.

The repo also contains a file named `env-linux-mac-example.sh`, which is a sample file you can take as a reference to understand how the `INTERSIGHT_API_KEY_ID` and `INTERSIGHT_API_PRIVATE_KEY` environment variables can be set. The `credentials.py` Python

module will fetch the keys from those variables

Anatomy of an ICO Python Script (Sample)

```
import credentials
from pprint import pprint
from intersight.api import workflow_api
from intersight.rest import ApiException

# Specify the MOID here
moid='6239aab7696f6e2d3118775d'

def get_tasks_definitions(api_client):
    """ Gets the list of tasks """
    api_instance = workflow_api.WorkflowApi(api_client)

    try:
        api_response = api_instance.get_workflow_task_definition_by_moid(moid=moid, check_return_type=False)
        pprint(api_response)
    except ApiException as e:
        print("Exception when calling WorkflowApi->get_workflow_task_definition_list: %s\n" % e)

# Authenticate
api_client = credentials.config_credentials()

# Get the list of Tasks
get_tasks_definitions(api_client)
```

Import credential module to authenticate

Import workflow_api to work with ICO

Create an instance of the workflow_api,
passing the credentials

In this case, we use the
get_workflow_task_definition_by_moid class
as we want to get the task definition for a
specific moid. We pass as arguments the
moid as well as the optional
_check_return_type set to False to get output
in JSON format

We set api_client which will be passed to the
get_tasks_definitions function in order to
authenticate

We call the get_tasks_definitions function
which will return the task definition in JSON
format

Example 1: Execute a Workflow

```
% source source.sh
% env | grep INTERSIGHT

INTERNSIGHT_API_PRIVATE_KEY=/Path/To/secret.txt
INTERNSIGHT_API_KEY_ID=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

% python ico_wf_execute_by_name.py

{'account': None,
 'account_moid': '5f871f403361396c6a3154ae',
 'action': 'None',
 'ancestors': None,
 [...],
 'workflow_task_count': 3,
 'workflow_worker_task_count': 2}

Execution Moid: 6254166b696f6e2d31db1e93
Executed by: rtortori@cisco.com
```

Requests > Deploy a Virtual Machine

Details

Status	In Progress
Name	Deploy a Virtual Machine
ID	62544889696f6e2d31dbb80e
Target Type	Registered Device
Target Name	blackmore.rmlab.local
Source Type	-
Source Name	-
Initiator	rtortori@cisco.com
Start Time	Apr 11, 2022 5:26 PM
End Time	-
Duration	47 s

Execution Flow

Progress

Show Additional Details

New Virtual Machine from Template or Clone from Virtual Machine

- Logs
- Inputs
- Outputs

Example 2: Inspect Workflow Execution Outputs

```
% python ico_wf_get_request_tasks_outputs_by_moid.py

### Task New Virtual Machine from Template or Clone from Virtual
Machine Output ###

{'ConfigResults': [{'ConfigResCtx': {'EntityData': {'task':
'workflow.ApiTask'}, 'EntityMoid': '', 'EntityName': '',
'EntityType': ''}, 'Message': "Virtual Machine 'test-ico-pythonsdk'
created successfully.", 'MessageParams': None, 'OwnerId': '',
'State': 'Ok', 'Type': 'Config'}], 'Folder': '/RMLAB/vm/rtortori',
'Name': '/RMLAB/vm/rtortori/test-ico-pythonsdk', 'TaskId': 'task-
318756', 'Template': '/RMLAB/vm/rtortori/ubuntu-template-hx'}

### Task Get VMware Virtual Machine Guest IP Output ###

{'ConfigResults': [{'ConfigResCtx': {'EntityData': {'task':
'workflow.ApiTask'}, 'EntityMoid': '', 'EntityName': '',
'EntityType': ''}, 'Message': '', 'MessageParams': None, 'OwnerId':
'', 'State': 'Ok', 'Type': 'Config'}], 'vmip': '192.168.130.150'}
```

Execution Flow

Show Additional Details

Get VMware Virtual Machine Guest IP

- Logs
- Inputs
- Outputs

```
4      "ConfigResCtx": {
5        "EntityData": {
6          "task": "workflow.ApiTask"
7        }
8      },
9      "State": "Ok",
10     "Type": "Config"
11   }
12 }
13 "Virtual Machine Guest IP": "192.168.130.150"
14 }
```

Example 3: Rollback a Workflow

```
% python ico_wf_rollback_by_moid.py
Rolling back Workflow with Moid 625448896f6e2d31dbb80e
Rollback execution Moid: is 62544a86696f6e2d31dbba8c
Status: Running
```

The screenshot displays a workflow management interface with a dark theme. The main title is "Requests > Rollback Deploy a Virtual Machine". In the top right corner, there are notification icons: a red square with '26', a yellow triangle with '24', a blue circle with '1', and a speaker icon with '31'. The interface is split into two main sections: "Details" on the left and "Execution Flow" on the right.

Details Section:

Status	In Progress
Name	Rollback Deploy a Virtual Machine
ID	62544a86696f6e2d31dbba8c
Target Type	Registered Device
Target Name	blackmore.rmlab.local
Source Type	Orchestration
Source Name	Rollback Deploy a Virtual Machine
Initiator	rtortori@cisco.com
Start Time	Apr 11, 2022 5:34 PM
End Time	-
Duration	25 s

Execution Flow Section:

Progress: Show Additional Details

Remove Virtual Machine

Inputs

```
1 {
2   "Datacenter": "/RMLAB",
3   "Folder": "/RMLAB/vm/rtortori",
4   "Force Delete": true,
5   "Hypervisor Manager": {
6     "Moid": "60a033446f72612d33e9f5df",
7     "ObjectType": "asset.DeviceRegistration"
8   },
9   "Virtual Machine": "/RMLAB/vm/rtortori/test-ico-pythonsdk"
10 }
```

Example 4: Get ICO Statistics

```
% python ico_statistics_get.py

### Workflow Designer Statistics ###

Total Workflows: 162
System Workflows: 29
User Workflows: 133
Valid/Invalid Workflows: 149/13

### Executions Statistics ###

Total Executions: 695
Success/Failed Executions: 434/261
Last Request: "Rollback Deploy a Virtual Machine", by user
rtortori@cisco.com, with Moid 62544a86696f6e2d31dbba8c

### Task Designer Statistics ###

Total Tasks: 229
System Tasks: 152
Custom Tasks: 77

### Data Types Statistics ###

Total Data Types: 228
System Data Types: 213
Custom Data Types: 15
```

Execute an ICO Workflow with Powershell SDK

Requirements

- Knowledge of Powershell and Intersight
- Powershell 7.1 or later
- Dotnet SDK 3.1 or later
- An Intersight API Key and Secret
- Connectivity to intersight.com (or your CVA/PVAPP if you are running on-prem)

Powershell Examples described in this document:

<https://github.com/rtortori/ico-powershell-examples>

Install and Configure Intersight Powershell Module

```
rtortori@laptop ~ % pwsh
PowerShell 7.2.2
Copyright (c) Microsoft Corporation.
```

```
https://aka.ms/powershell
Type 'help' to get help.
```

```
PS /Users/rtortori> Install-Module -Name Intersight.PowerShell
```

```
Untrusted repository
```

```
You are installing the modules from an untrusted repository. If you trust this repository, change its
InstallationPolicy value by
running the Set-PSRepository cmdlet. Are you sure you want to install the modules from 'PSGallery'?
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "N"): Y
PS /Users/rtortori>
```

Working with ICO and the Powershell SDK

Refer to the repo: <https://github.com/rtortori/ico-powershell-examples>

This repo includes code that can be used and reworked to address a number of use cases.

This document describes just a few of them, refer to the repository for additional info and instructions.

In order to authenticate to the Intersight API, in the repo is present a file called `sample_configuration.ps1`

This powershell script contains the necessary code to authenticate to Intersight, provided you have generated an API Key and Secret:

https://intersight.com/apidocs/introduction/security/%23generating-api-keys&sa=D&ust=1612024909729000&usg=AOvVaw362rkbFxqhX_Mo8w0xkDJG/

You will need to either execute that script in your session or embed it in your powershell scripts in order to authenticate to the Intersight API.

Example 1: Execute and Rollback a Workflow

Invoke Execution

```
ICO > ./ico_wf_execution_by_name.ps1
```

Name	Moid	Email
----	----	-----
Deploy a virtual machine	6254166b696f6e2d31db1e93	user@cisco.com

Rollback

```
ICO > ./ico_wf_rollback_by_moid.ps1 6254166b696f6e2d31db1e93
```

Moid	Status
----	-----
625416ec726f6e2d312e72ea	Created

Moid	Status
----	-----
625416ec726f6e2d312e72ea	Running

Example 2a: Get all Workflows Definitions

```
ICO > $filter="(Properties.ExternalMeta eq true) and (Name ne 'DeployHyperFlexSDWAN') and (DefaultVersion eq true)"
ICO > $wf = Get-IntersightWorkflowWorkflowDefinition -InlineCount allpages -Top 0 -Filter $filter
ICO > $wf
```

Count Results

```
522 {class WorkflowWorkflowDefinition {...
```

```
ICO > $wf.Results.Count
```

```
522
```

```
ICO > $wf.Results
```

```
ClassId           : WorkflowWorkflowDefinition
ObjectType        : WorkflowWorkflowDefinition
LicenseEntitlement : Premier
DefaultVersion    : True
Description       : a test workflow to experiment with custom integrations
InputDefinition   : {class WorkflowBaseDataType {
                    class MoBaseComplexType {
                        ClassId: 0
                        ObjectType: 0
                    }
                }
                }
[...]
```

Intersight Powershell
SDK cmdlets accept
the parameter `-Filter`

This parameter accepts
a filter query that can be
used to reduce the
scope of the search as
shown in the example

Example 2b: Get all Workflows Definitions (Table View)

Sort in descending order by CreateTime

Only display Workflow Label, Moid and CreateTime

```
ICO > $wf.Results | Sort-Object CreateTime -Descending | Format-Table -Property Label,Moid,CreateTime
```

Label	Moid	CreateTime
testwf12	624f59c2696f6e2d316e9464	04/07/2022 21:38:11
testwfe01233	624ed76e696f6e2d316bc7b1	04/07/2022 12:22:06
Test Powershell Inspector	624ec3b8696f6e2d316b95a0	04/07/2022 10:58:00
Powershell - MS Services	624cb5ef696f6e2d31643927	04/05/2022 21:34:39
Configure on-premises FlexPod storage	624b4bbe696f6e2d3161fe0a	04/04/2022 19:49:18
Create Workspace	6244201a696f6e2d314bd343	03/30/2022 09:17:14
Deploy a Baremetal server	623d2245696f6e2d30384732	03/25/2022 02:00:37
Demo - Detach a workspace from a Policy Set	623a0f8c696f6e2d3119fdc4	03/22/2022 18:03:56
Test - Loop in template	6239cb92696f6e2d3119624c	03/22/2022 13:13:54
Demo - Create a Policy Set and add a workspace	6239b7c7696f6e2d3118ec22	03/22/2022 11:49:27
Demo - SSH Executor - Custom Tasks	6239adb696f6e2d311886b0	03/22/2022 11:06:39
Create a new Policy Set	6239abf6696f6e2d3118798f	03/22/2022 10:59:02
Variables Demo - Math	6234a4d3696f6e2d310dabd4	03/18/2022 15:27:15
New Virtual Machine Network on Multiple Hosts	62345857696f6e2d310c47d7	03/18/2022 10:00:55
Workflow Variables Demo - With Vars	62320c11696f6e2d31cb5611	03/16/2022 16:10:57
HTTP Dump	622b4a0d696f6e2d31c0b44d	03/11/2022 13:09:34



The bridge to possible