Yet Another Presentation

Welcome to the YAP preface, a multitab section that you can use to describe the demo flow without skipping between YAP and PowerPoint. Each tab supports a Markdown or HTML formatted text that allows you to be creative with the documentation style. Best of all, you can include this information in the demo PDF report as live documentation for your demo.

This basic Hello Work demo describes the Yet Another Presentation tool (YAP) functionalities by orchestrating a simple project in a safe mockup environment. Following three basic steps, this demo:

- Creates a new project
- Add two connected router objects to the project
- Validate that there is no connectivity between them
- Enable their interface status and confirm the connectivity

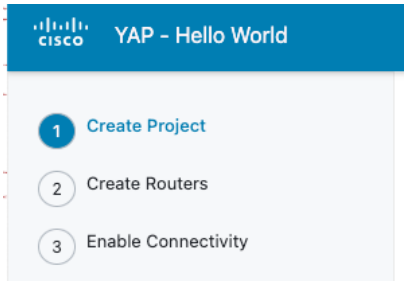
Step 1 - Review the demo content

Click on the rest of the preface tabs for step-by-step information about this demo, and remember that you always come back to this information by clicking on the demo name (on the top left).

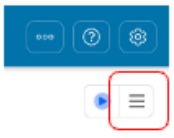
Step 2 - Execute the proposed demo steps

As you finish reviewing the demo material, click on the demo steps list on the left to drive the actual demo using the flow suggested in the demo content.

This list represents a simple demo workflow that will mark a step as completed or failed based on the outcome of the configured API calls.



Note that you can render the step documentation as a floating window by clicking on the burger on the top right.



After experimenting with presentation mode, check the Next Steps section to learn more about the Edit mode. YAP exposes intuitive copy, delete and edit actions to customize the flow.

Create Project

In the first step of our demo, we will create a project container for our network. To perform this activity, we must execute a single POST call toward the demo endpoint, evaluate the return code and payload, and store the UUID allocated for the Project for the following steps.

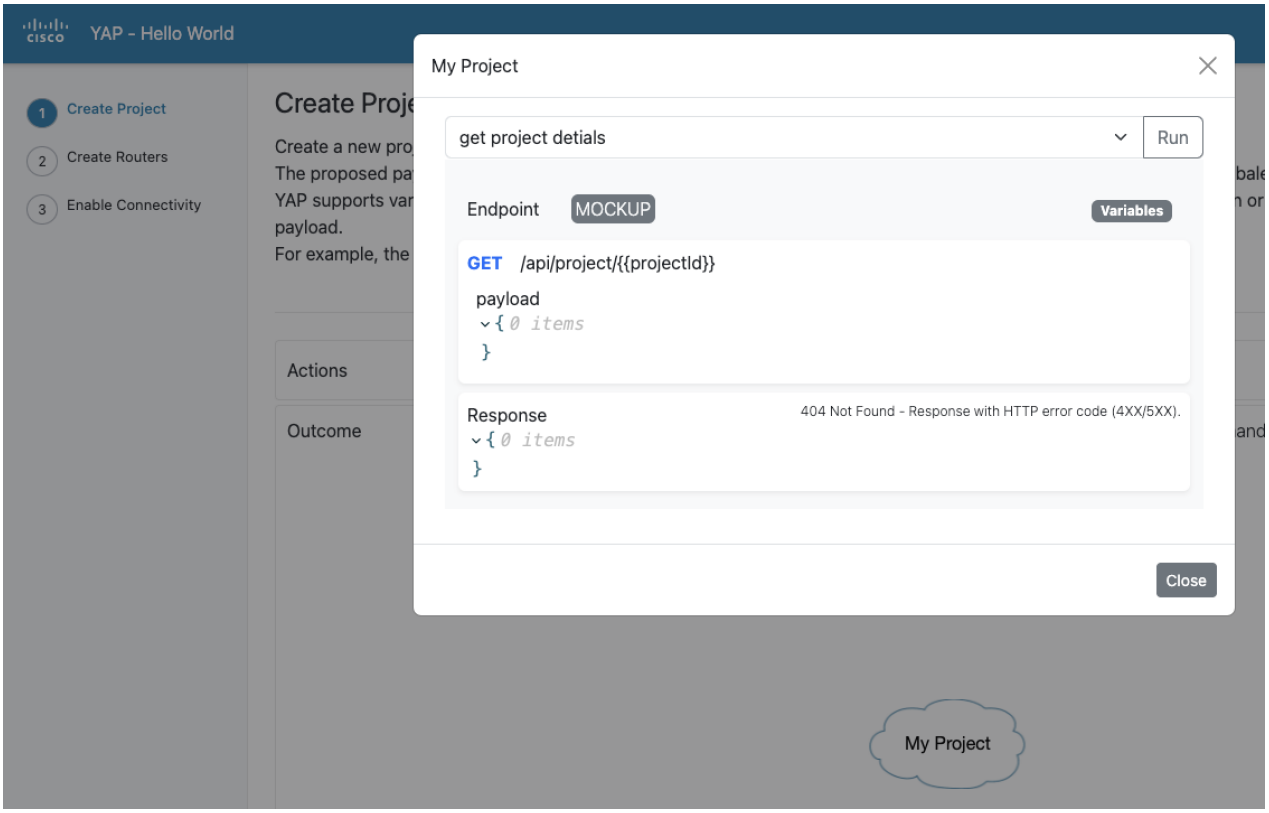
For every step in a demo, YAP exposes an optional interactive contextual diagram that visualizes the achieved goal. You can interact with each object in the graph, executing pre-configured API calls or accessing the target object via an SSH session.

Step 1 - Interact with YAP context diagram

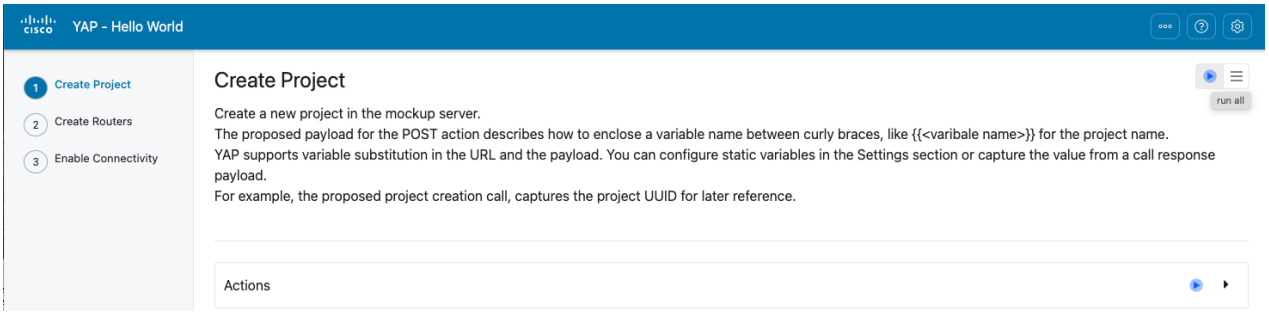
Click on the cloud, select "Get Project Details" from the list of available commands and then Run to execute the call.

Step 2 - Execute the proposed APIs, exposing an appropriate level of information for the demo audience

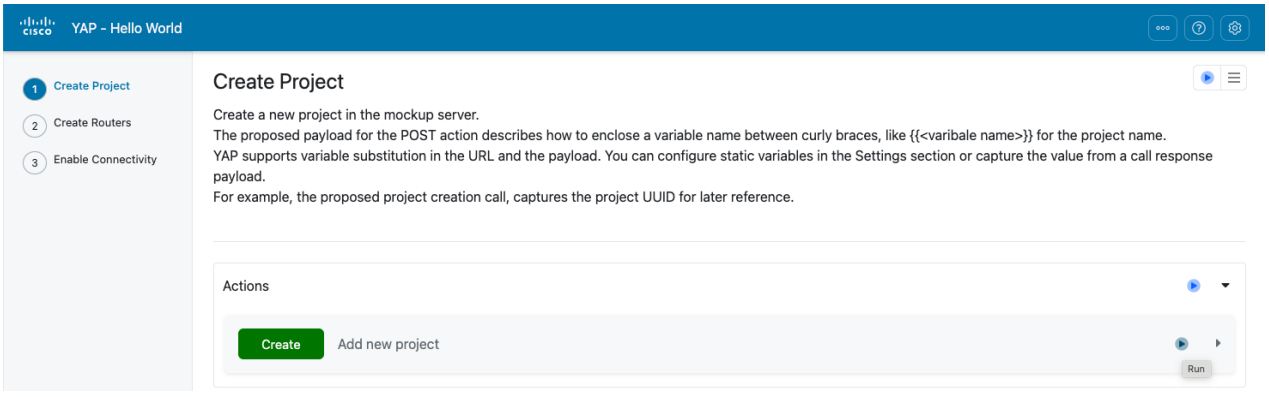
This example executes a GET call for a project that we still need to create and, as expected, returns a 404 error code. YAP interprets strings as variables when enclosed between two curly brackets, like projectId in this example. If you hover over the variables object, you can see no project UUID yet.



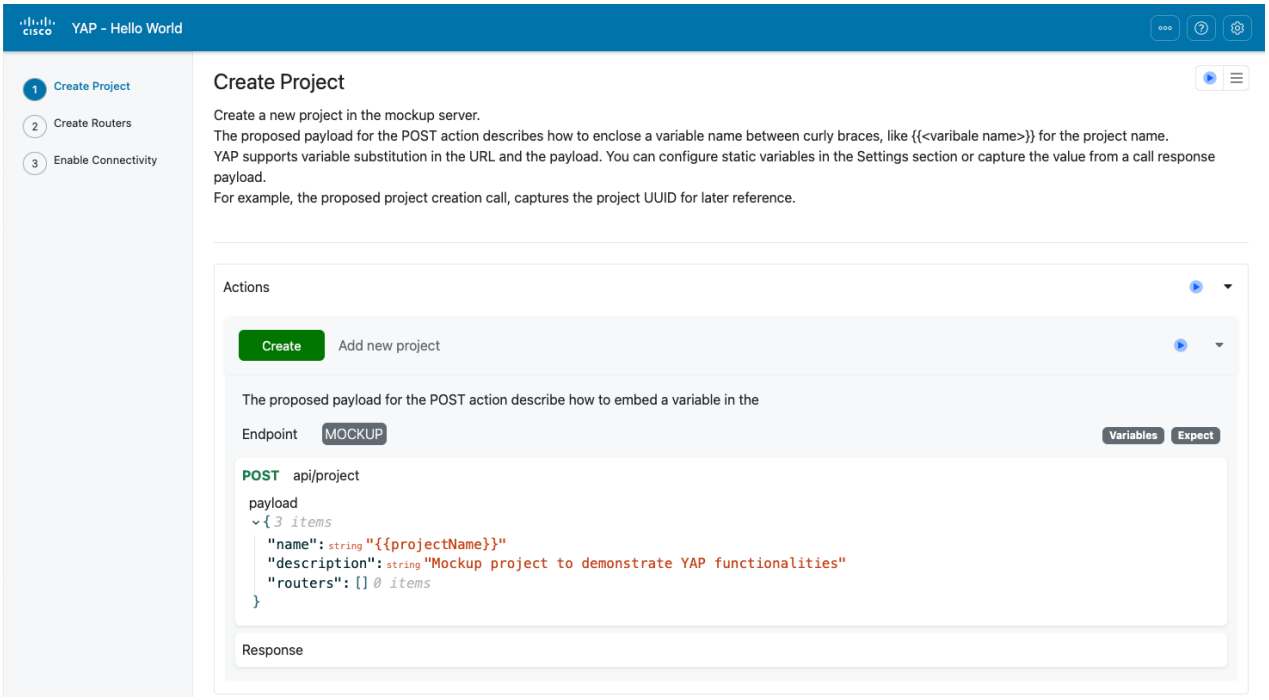
For high-level presentations, you can execute all the pre-checks, actions and post-checks with a single click on the top right icon **run all**, without exposing the API details to the audience.



You can expand the pre-checks, actions, and post-checks for more technical demos. Execute all APIs in a block, or each API call independently as your use case requires. Notice that YAP renders the pre-checks and post-checks sections only when they contain at least one API call.

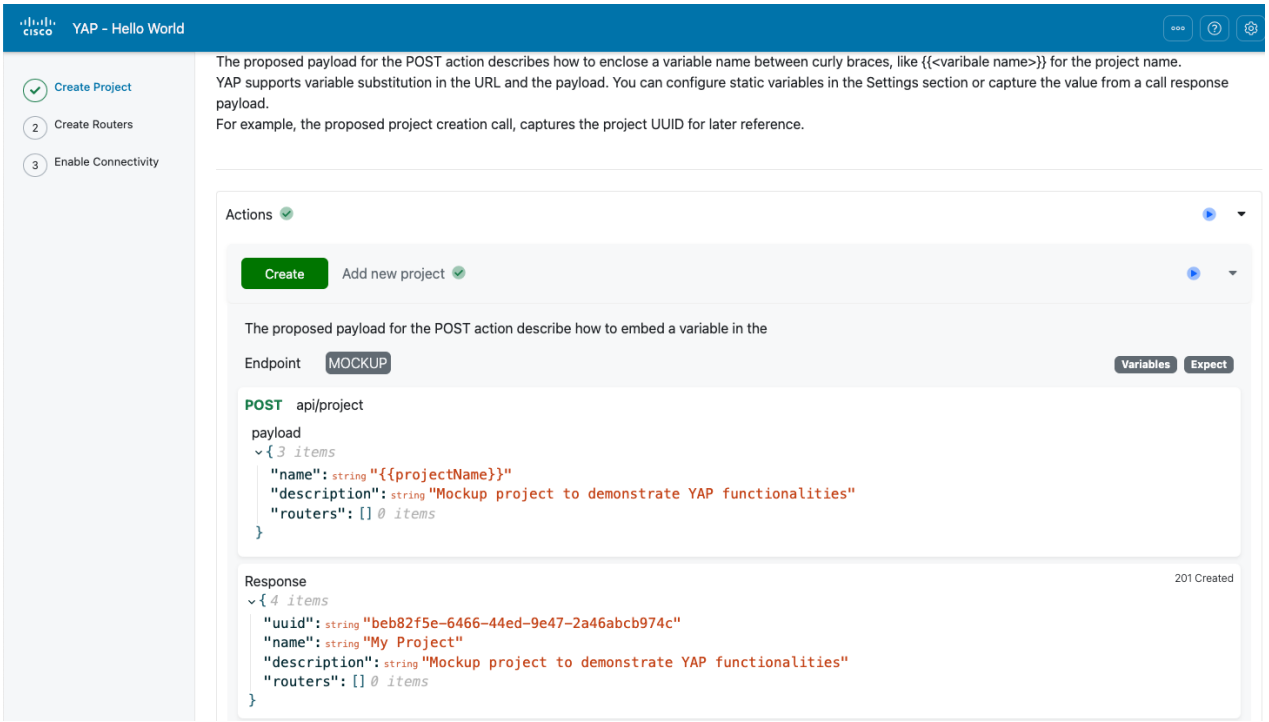


You can expand an API to expose the payloads, endpoint, variables and success criteria.

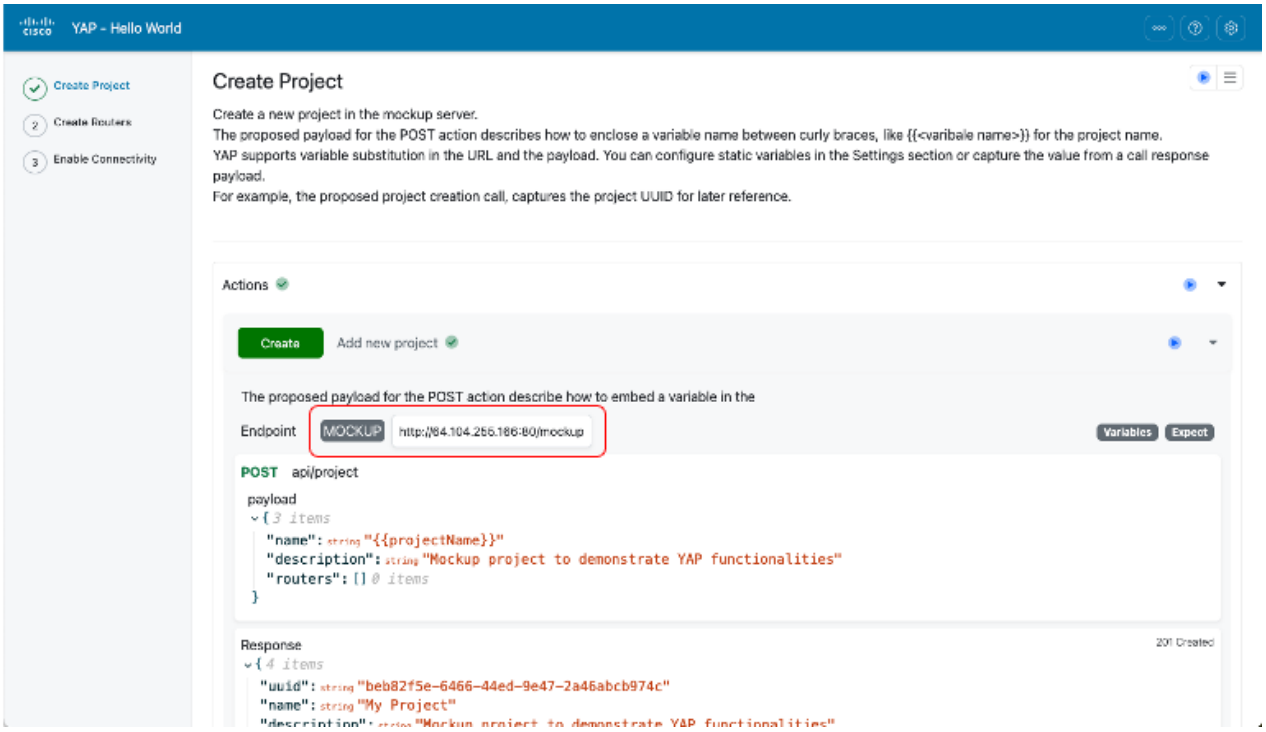


Step 3 - Present the API call response and sucess criterias

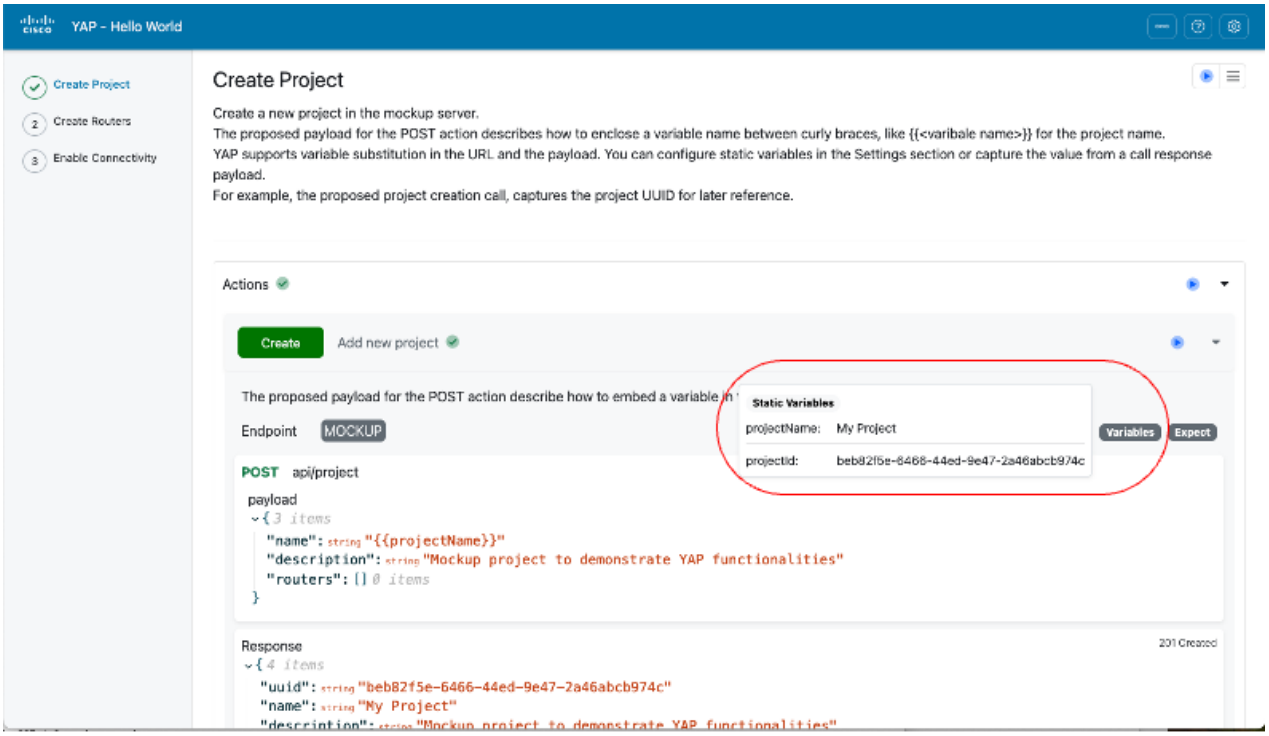
For technical presentation, after executing an API call, YAP renders the response code and payload.



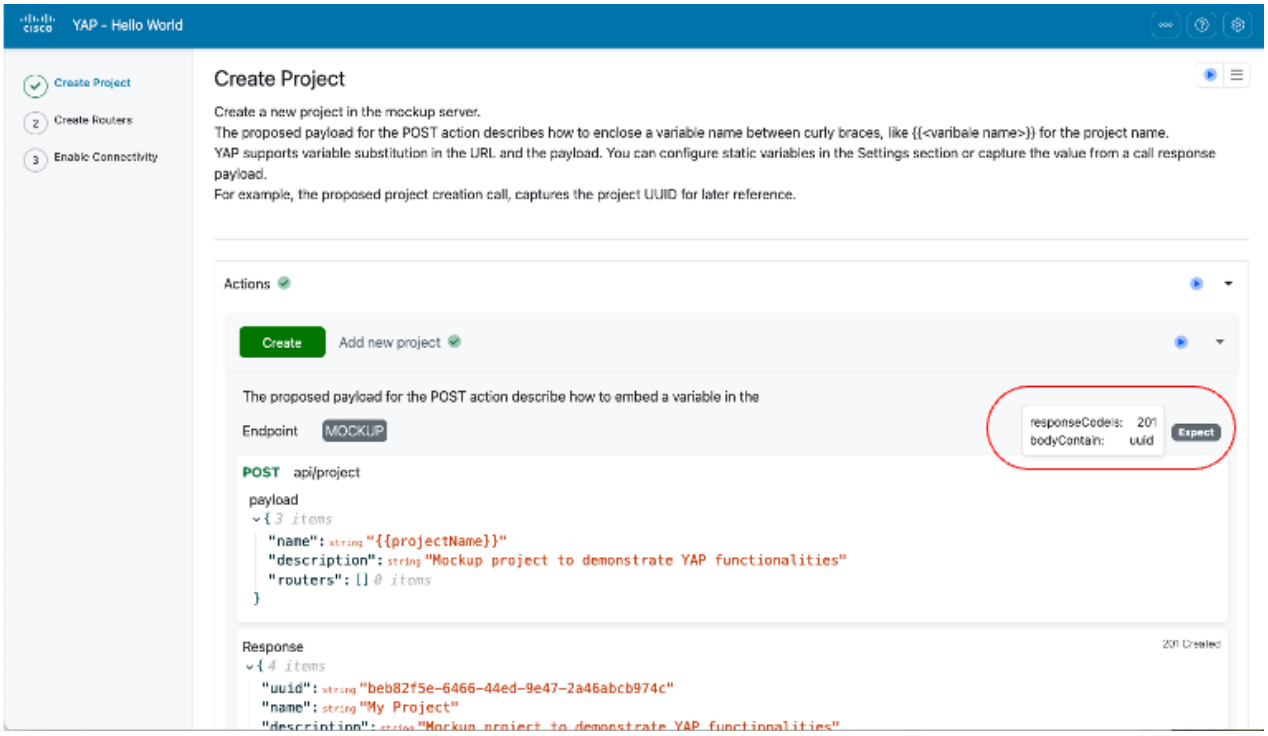
Hover over the Endpoint label to visualize endpoint details.



Hover over the Variabales label to visualize the variables used or captured by the API.

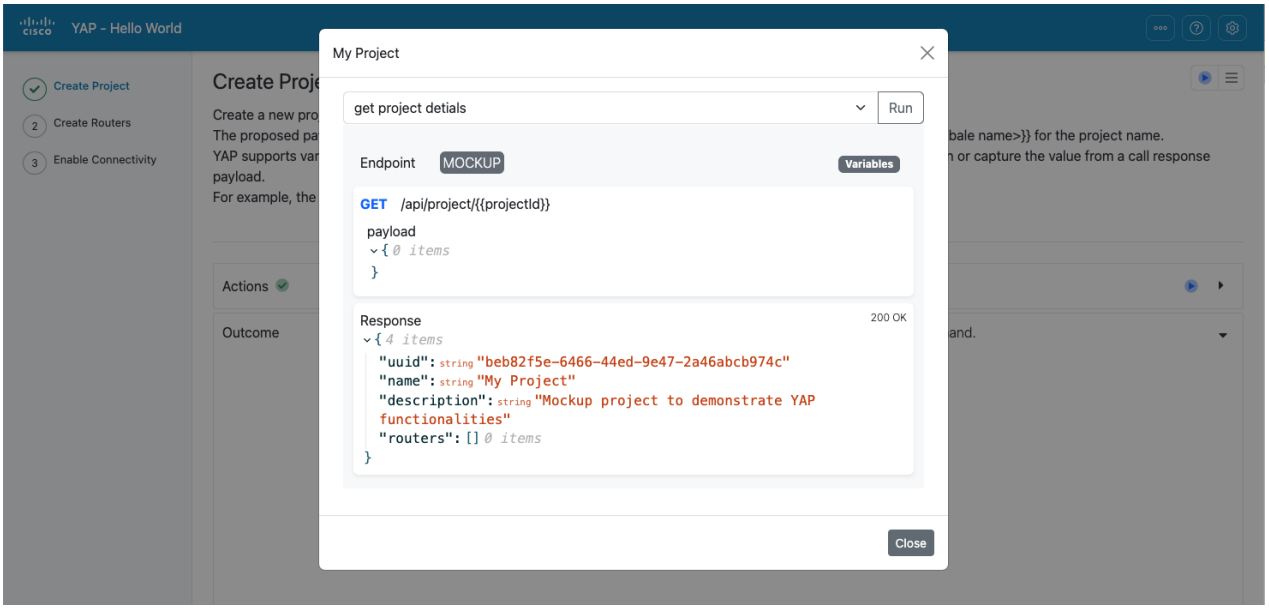


Hover on the Expect label to visualize the expected response code or string to be matched in the response payload before marking this API as completed successfully.



Step 4 - Validate the step objective

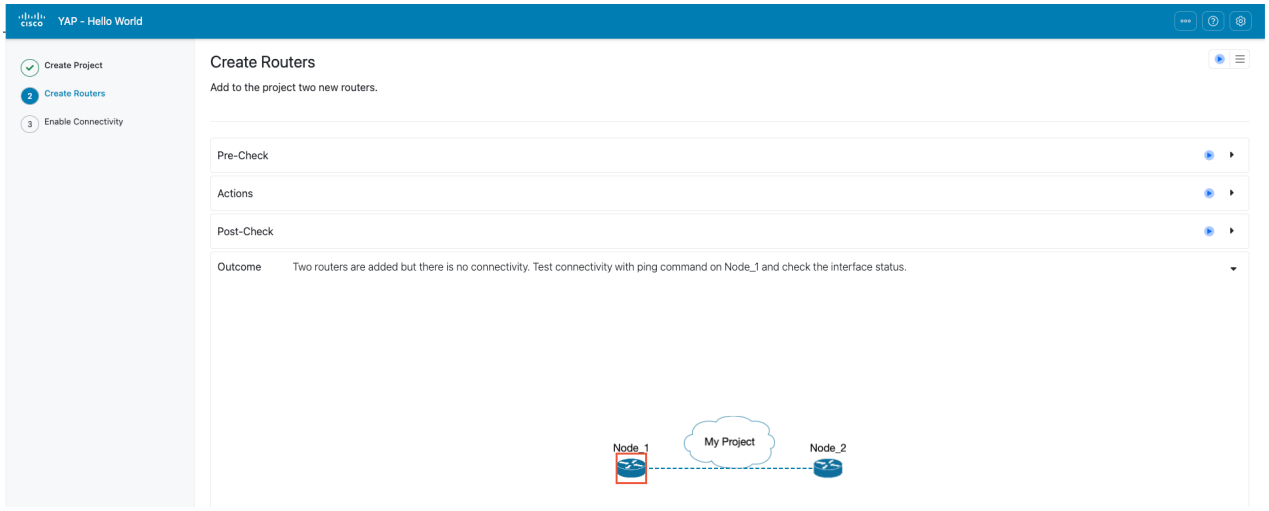
Follow the same process as Step 1, click on the cloud icon and execute "Get Project Details" to confirm that your project has been created.



Also, notice that the demo step and each API show a green check mark to confirm the successful execution. All APIs in a section must execute successfully before marking a phase on the demo work flow. If any API fails, the demo phase shows a failure mark.

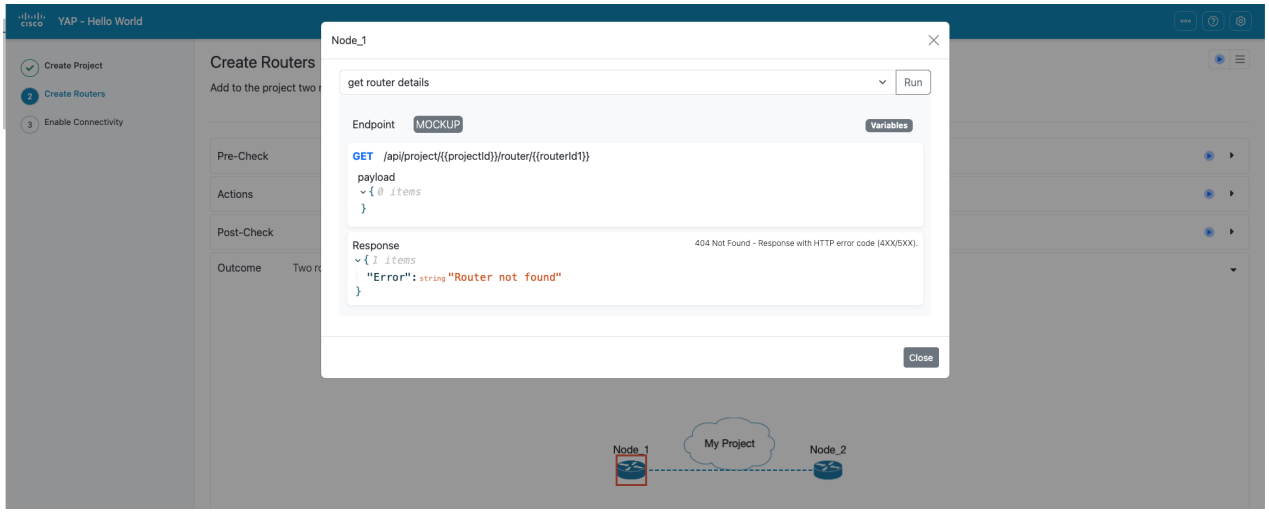
Create Routers

The second step in our Hello World demo adds two routers to the project just created and validates the connectivity between them.



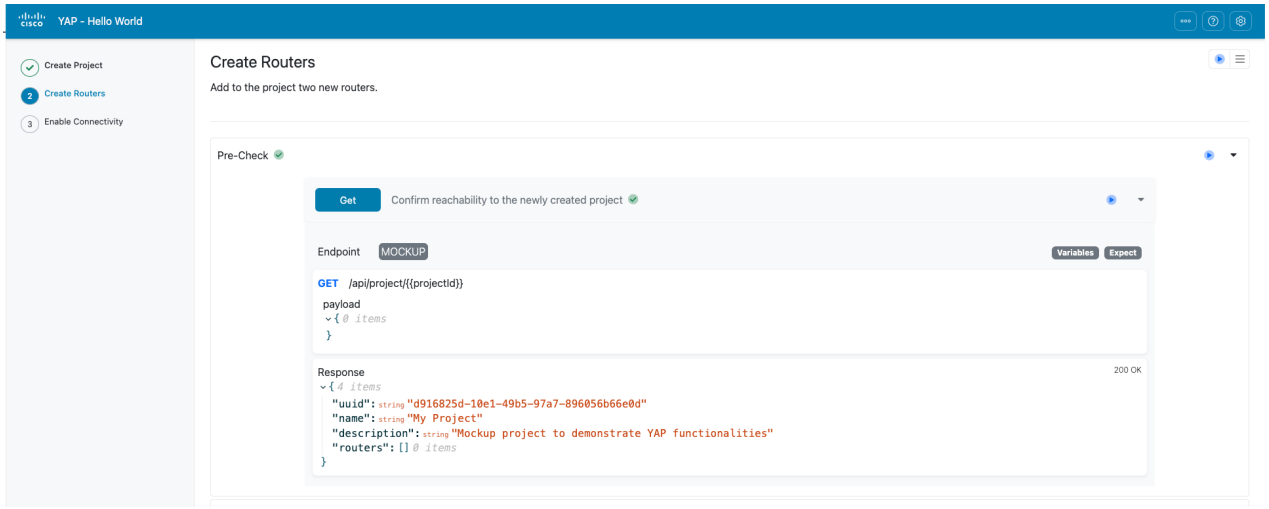
Step 1 - Confirm there is no router in the project

Click on Node_1, select and run "Get router detail" to confirm that your project has no router configuration.



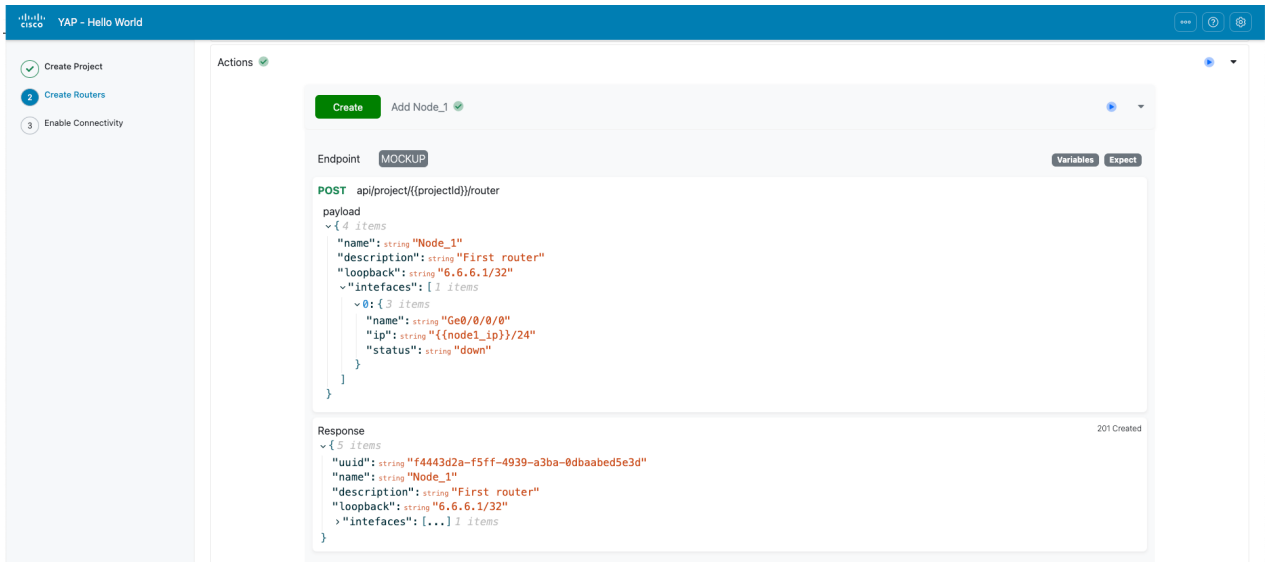
Step 2 - Execute the pre-check section

This simple action on the pre-check explains how you can validate with one or more GET calls your demo state before progressing on the step actions.



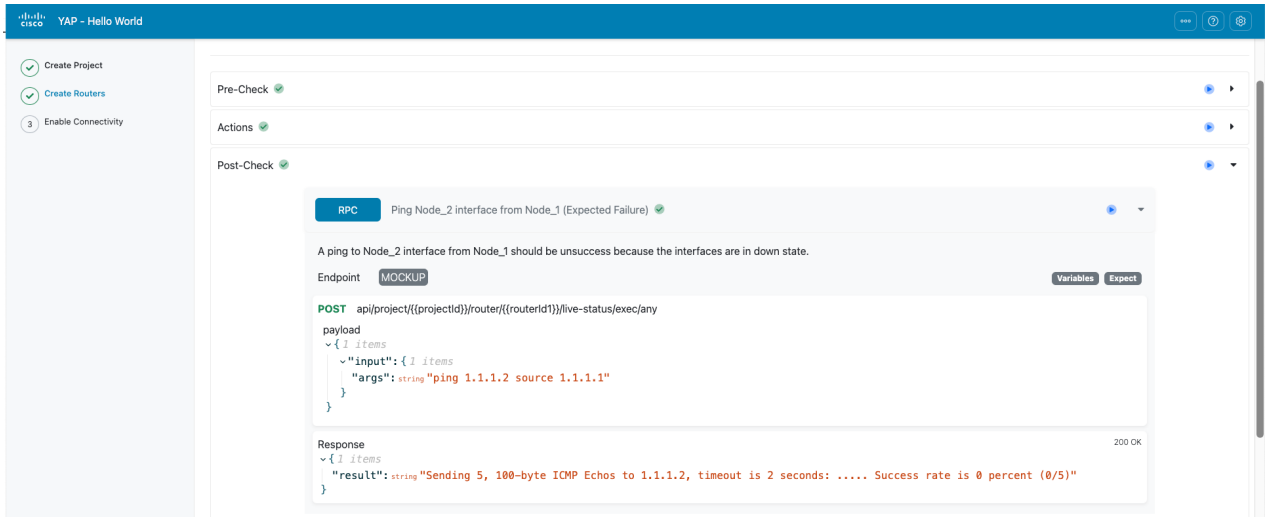
Step 3 - Execute action section

Create two routers in our project by executing two basic POST API calls. You can run the whole API block or run one API at a time. Every demo and audience is different, but YAP provides this flexibility level in the demo delivery. If you run their APIs multiple times, the demo will still work, but multiple routers (with different UUIDs) instances will be added to the project because this is a POST command.



Step 4 - Execute post-check section

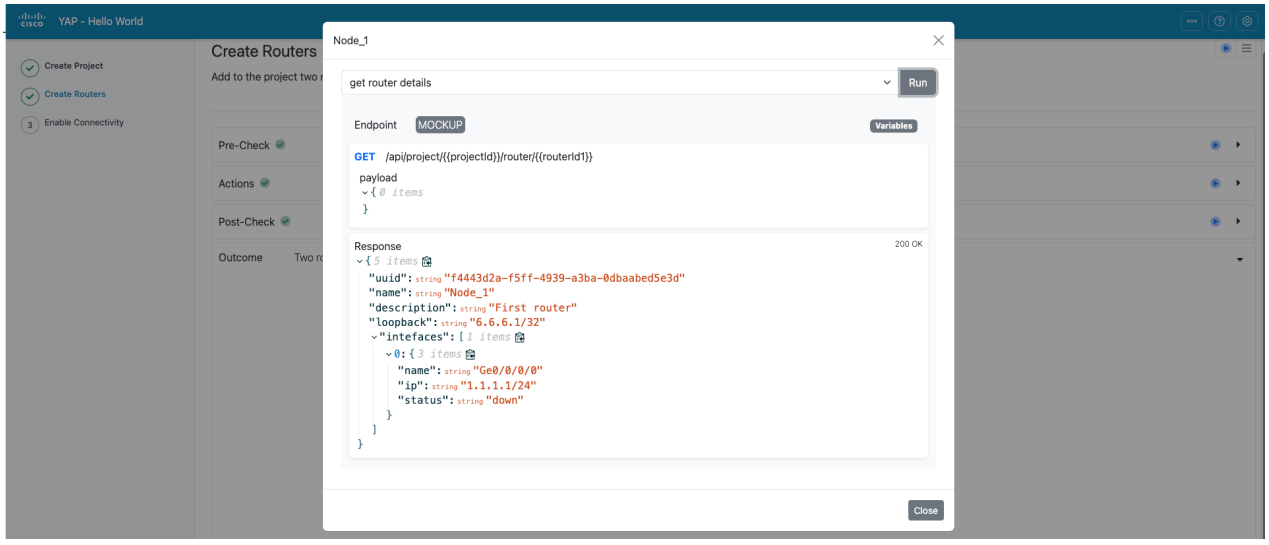
A Post-check is an optional section that behaves like the Actions but, as the Pre-checks, helps separate the purpose of your API calls and focuses on the validation.



Step 5 - Confirm the outcome from contextual diagram

For high-level presentations, instead of exposing the API response, we can run the preconfigured commands or show directly the outcome on the target system UI.

As we learned in the previous section, execute again the commands pre-configured for Node_1 in the Outcome section that confirm the newly created interface is in a down state.



As the outcome complexity increases, you can highlight one or more objects to focus attention and drive the user when presenting.

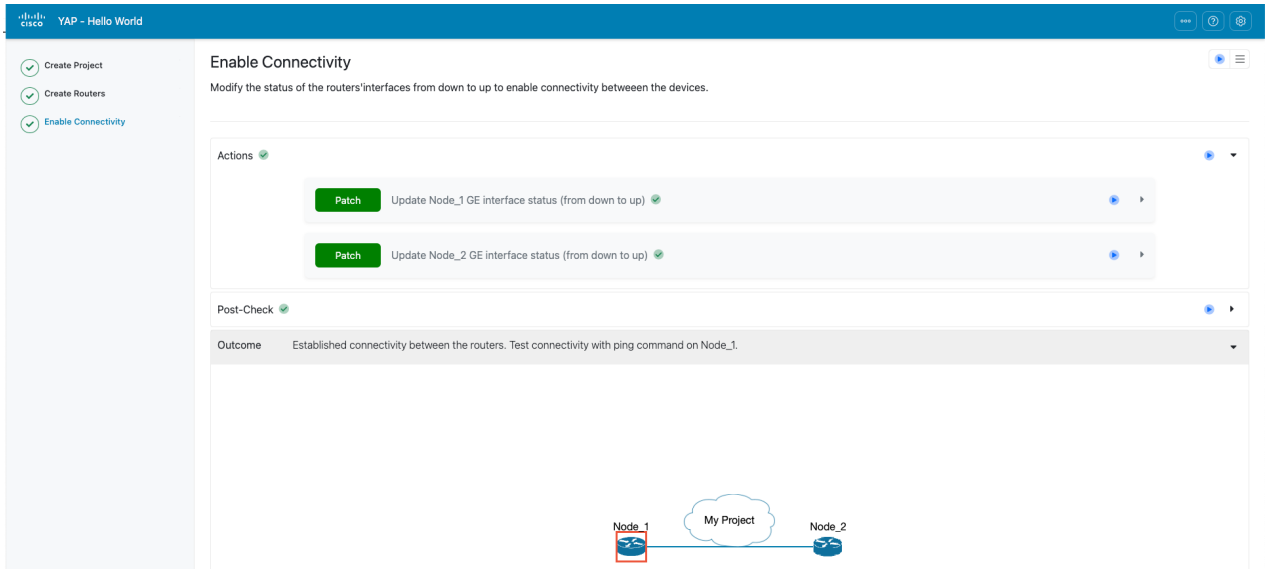
Enable Connectivity

The third step follows the same flow suggested in Step 2, enabling connectivity between the two routers in your project.

Step 1 - Execute all API calls in this step

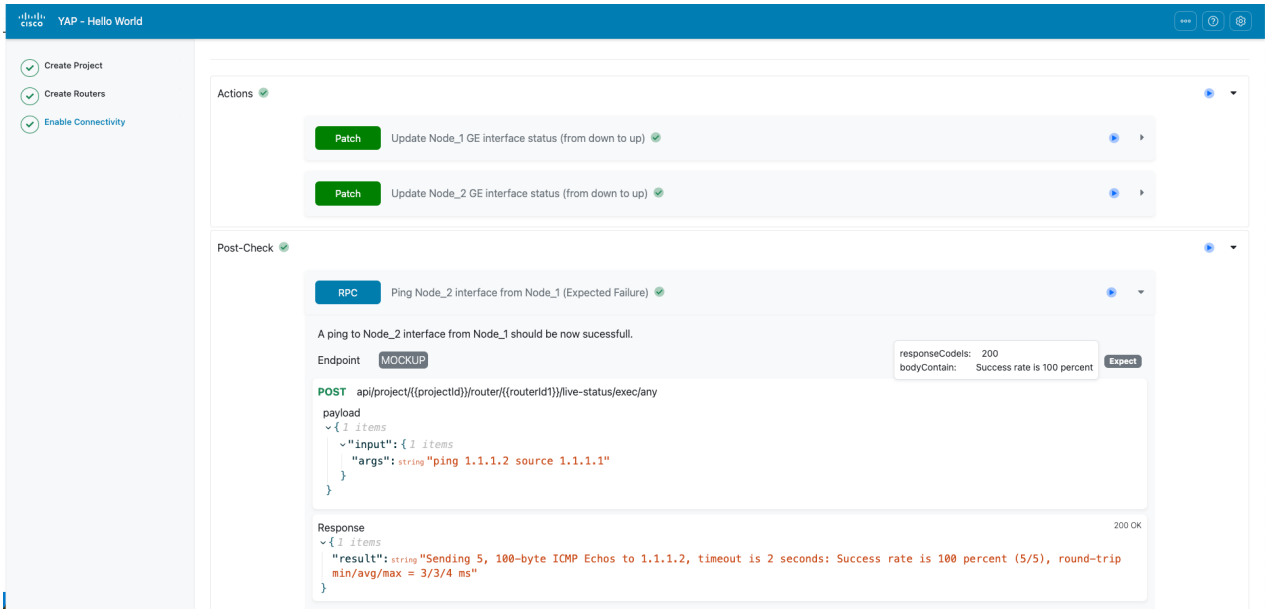
As we learned in previous steps, we can execute all API with a single click on Run All (play icon in the top right). This activity performs two PATCH calls on the resources created in the previous step, changing the status of the newly

created interfaces to up.



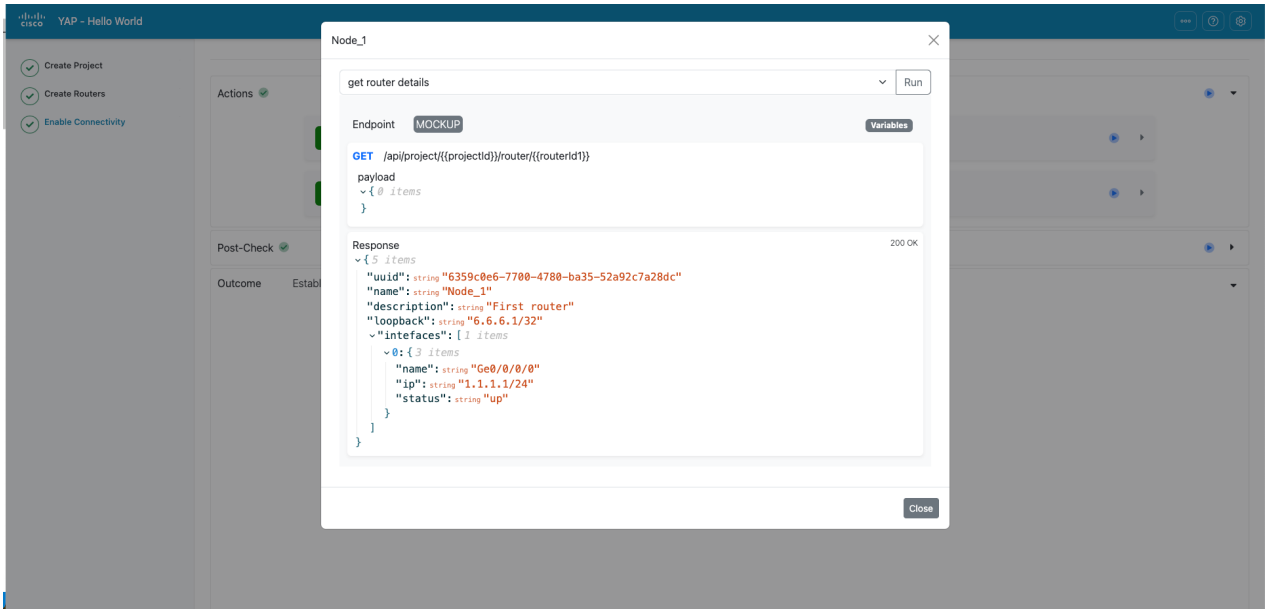
Step 2 - Confirm Outcome in Post_Checks

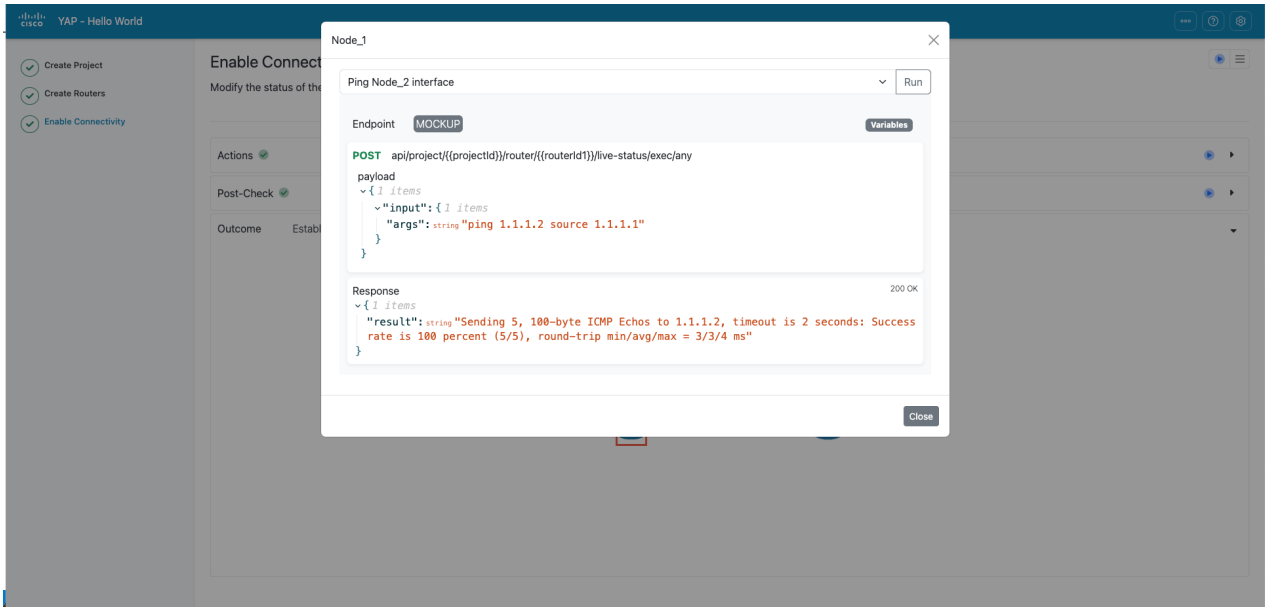
Expand the Post-Check section to see that the connectivity is now successful. If you hover over the expect, you appreciate how YAP validate this state by matching the success rate in the response payload.



Step 3 - Confirm Outcome in the Outcome

Finally, execute the "Get Router Details" on Node_1 in th Outcome section to confirm the interface status and connectivity.

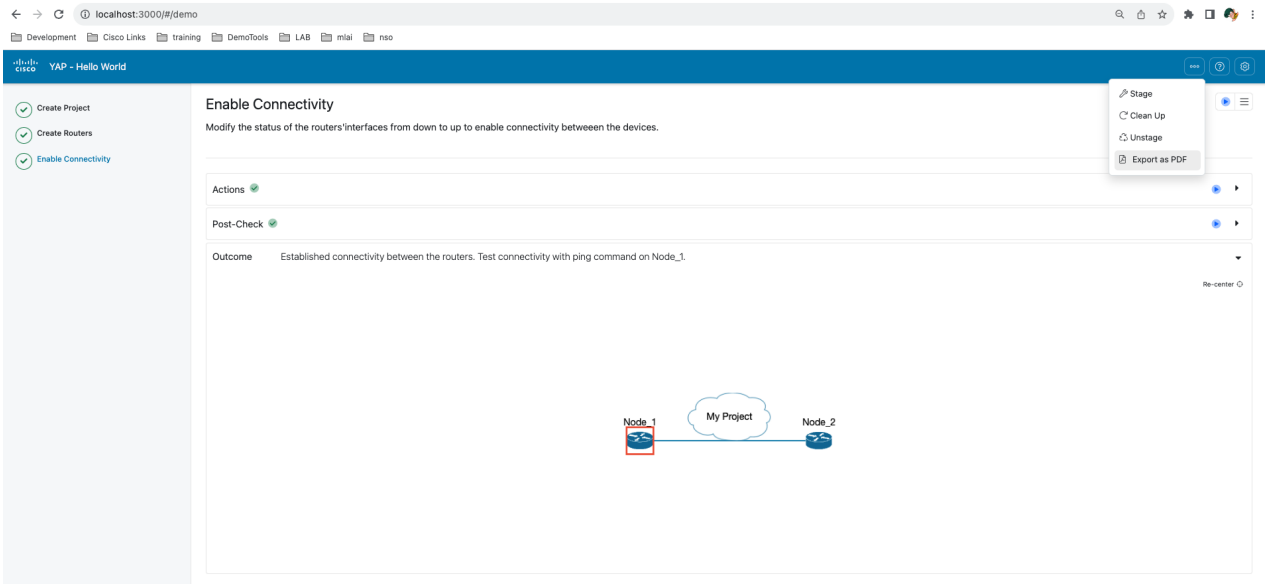




Generate Documentation

YAP can generate the documentation for your demo, summarizing the preface information and a complete API description, including responses for the executed calls.

Click on the three dots (...) on the top right menu and select **Export as PDF** from the proposed menu.



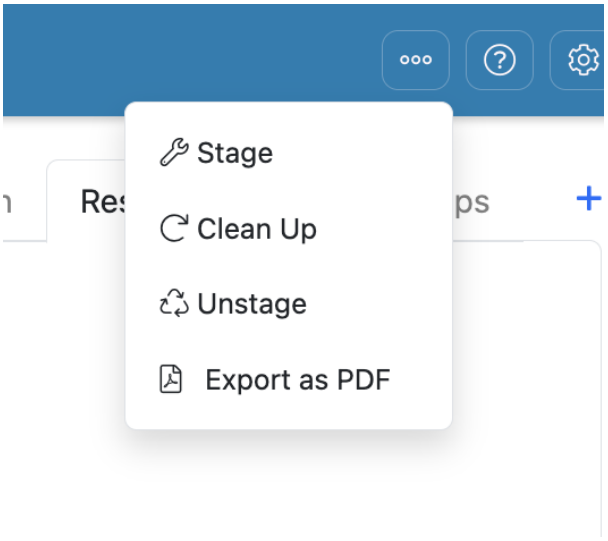
YAP will generate the documentation as a PDF and ask where to save the document. You can now open this document with Word and, if required, modify the format proposed.

Cleanup Demo

YAP supports three operation phases: Stage, Unstage and Cleanup.

You can access these stages by clicking the three dots (...) on the top right menu.

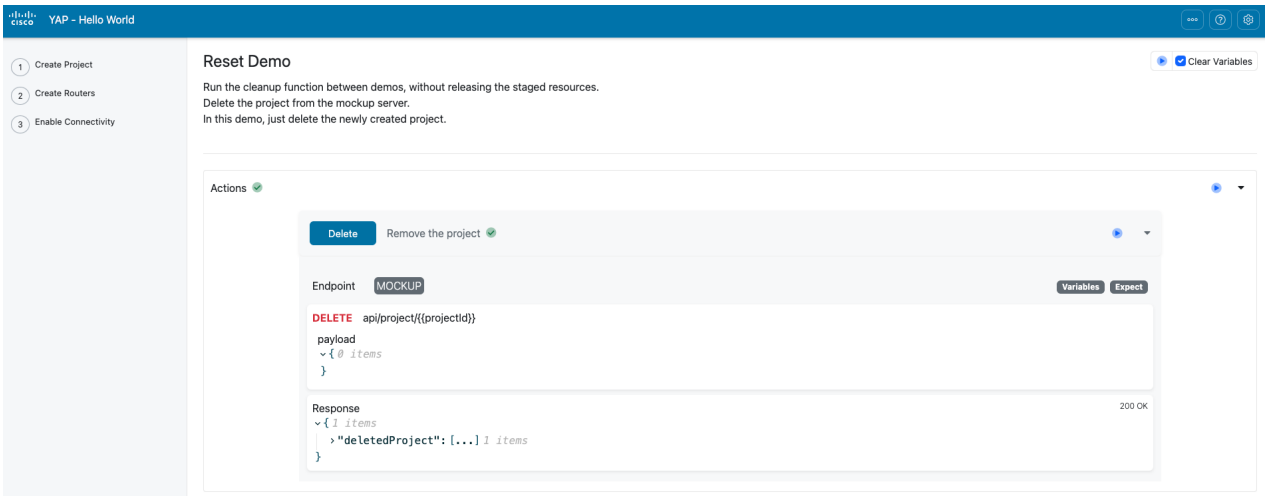
These stages behave like the other demo phases, supporting pre-checks, actions, and post-checks API sections. They are optional; you can decide how to use them.



- For example, you can leverage **Stage** to bring up demo resources or set up a demo baseline.
- You can run **Clean Up** between demos to remove configured resources without losing the state achieved by running Stage.
- Finally, **Unstage** is suitable for decommissioning your demo resources.

Step 1 - Run All for Cleanup step

This basic demo leverages only the Cleanup step. Execute Run All to remove your project and reset the status of the demo workflow, removing the success checks.



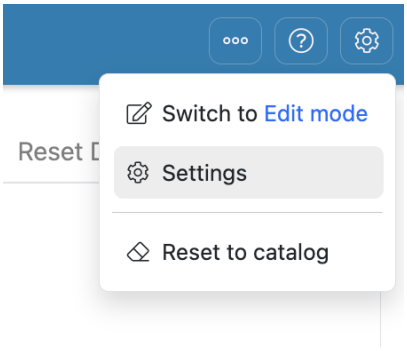
Clean Up is the perfect place for hiding your authentication if you want to avoid exposing it as demo steps.

In this case, remember to uncheck "Clear Variables" to prevent the action of cleaning the local storage with retrieved certificates or tickets.

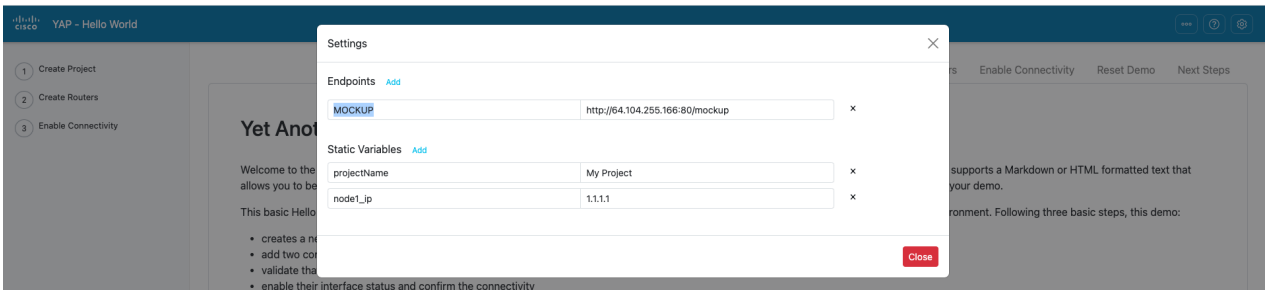
Next Steps

After experimenting with the presentation mode, try a few changes on this demo or add some API calls to your environment before stepping in and creating a new use case.

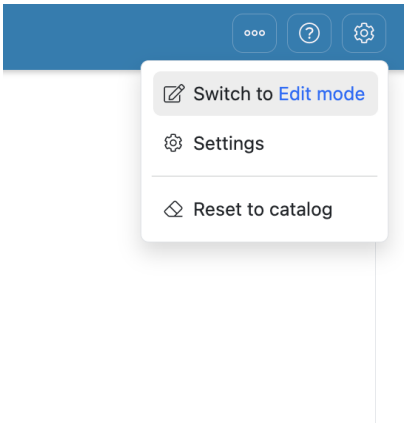
Click the right wheel icon on the top right to access this demo Setting.



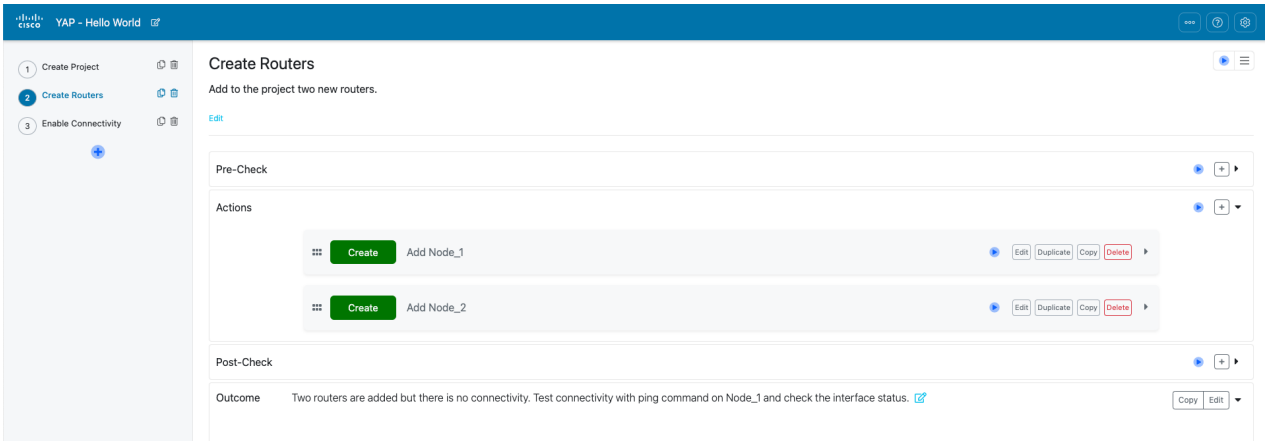
Settings exposes the endpoint. You can have one or multiple endpoints, click on the endpoint names to access its details.



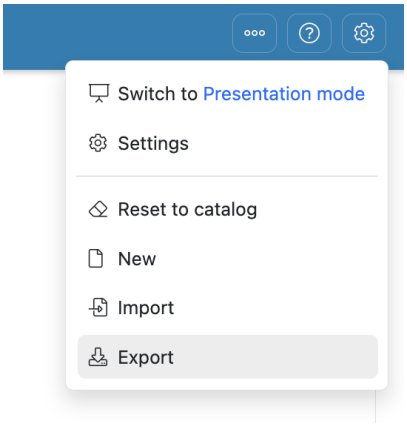
Click the right wheel icon again and select the switch YAP in **Edit mode**. YAP exposes intuitive copy, delete and edit actions to customize the flow.



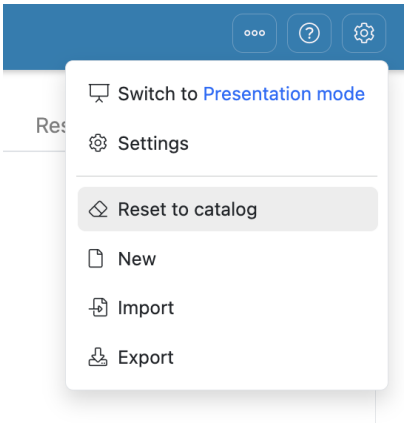
Click around the different sections of the demo and explore the edit, copy, duplicate, and delete options offered by the YAP UI



After updating your demo, you may save your work by exporting your changes as a JSON file to re-import at a later stage, from this same view or from the Catalog.



Finally, select **Reset to catalog** to close this demo and render YAP startup catalog page.



If you have any comments or are interested in collaborating on this project, leave us a message on YAP discussion.

Endpoints

This page summarize all the endpoints used in this document.

MOCKUP

- baseURL : `http://64.104.255.166:80/mockup`

Headers

Key	value
Content-Type	application/json

API Information

1. Create Project

Create a new project in the mockup server.
The proposed payload for the POST action describes how to enclose a variable name between curly braces, like {{<varibale name>}} for the project name.
YAP supports variable substitution in the URL and the payload. You can configure static variables in the Settings section or capture the value from a call response payload.
For example, the proposed project creation call, captures the project UUID for later reference.

Actions

Add new project

API Endpoint : `MOCKUP` **Method** : `POST`
Path : `api/project`
Payload :

```
{
   "name": "{{projectName}}",
   "description": "Mockup project to demonstrate YAP functionalities",
   "routers": []
}
```

Response : `201 Created`

```
{
   "uuid": "1d8c9560-5053-429e-b160-1064b8bef999",
   "name": "My Project",
   "description": "Mockup project to demonstrate YAP functionalities",
   "routers": []
}
```

2. Create Routers

Add to the project two new routers.

Pre-Check

Confirm reachability to the newly created project

API Endpoint : `MOCKUP` **Method** : `GET`
Path : `/api/project/{{projectId}}`
Response : `200 OK`

```
{
   "uuid": "1d8c9560-5053-429e-b160-1064b8bef999",
   "name": "My Project",
   "description": "Mockup project to demonstrate YAP functionalities",
   "routers": []
}
```


Actions

Add Node_1

API Endpoint : `MOCKUP` **Method** : `POST`
Path : `api/project/{{projectId}}/router`
Payload :


```
{
   "name": "Node_1",
   "description": "First router",
   "loopback": "6.6.6.1/32",
   "intefaces": [
      {
         "name": "Ge0/0/0/0",
         "ip": "{{node1_ip}}/24",
         "status": "down"
      }
   ]
}
```


Response : `201 Created`


```
{
   "uuid": "ae055cfa-0c5c-4d85-9b7c-33c380e60e24",
   "name": "Node_1",
   "description": "First router",
   "loopback": "6.6.6.1/32",
   "intefaces": [
      {
         "name": "Ge0/0/0/0",
         "ip": "1.1.1.1/24",
         "status": "down"
      }
   ]
}
```


Add Node_2

API Endpoint : `MOCKUP` **Method** : `POST`
Path : `api/project/{{projectId}}/router`
Payload :

```
{
   "name": "Node_2",
   "description": "Second router",
   "loopback": "6.6.6.2/32",
   "intefaces": [
      {
         "name": "Ge0/0/0/0",
         "ip": "1.1.1.2/24",
         "status": "down"
      }
   ]
}
```


Response : `201 Created`


```
{
   "uuid": "374eb367-df28-4bca-811a-9cc30b4b9dfd",
   "name": "Node_2",
   "description": "Second router",
   "loopback": "6.6.6.2/32",
   "intefaces": [
      {
         "name": "Ge0/0/0/0",
         "ip": "1.1.1.2/24",
         "status": "down"
      }
   ]
}
```


Post-Check

Ping Node_2 interface from Node_1 (Expected Failure)

A ping to Node_2 interface from Node_1 should be unsuccess because the interfaces are in down state.

API Endpoint : `MOCKUP` **Method** : `POST`
Path : `api/project/{{projectId}}/router/{{routerId1}}/live-status/exec/any`
Payload :


```
{
   "input": {
      "args": "ping 1.1.1.2 source 1.1.1.1"
   }
}
```


Response : `200 OK`


```
{
   "result": "Sending 5, 100-byte ICMP Echos to 1.1.1.2, timeout is 2 seconds: ..... Success
```

```
rate is 0 percent (0/5)"
}
```

3. Enable Connectivity

Modify the status of the routers'interfaces from down to up to enable connectivity betweeen the devices.

Actions

Update Node_1 GE interface status (from down to up)

API Endpoint : `MOCKUP` **Method** : `PATCH`
Path : `api/project/{{projectId}}/router/{{routerId1}}`
Payload :

```
{
   "intefaces": [
      {
         "name": "Ge0/0/0/0",
         "ip": "1.1.1.1/24",
         "status": "up"
      }
   ]
}
```

Response : `200 OK`

```
{
   "uuid": "ae055cfa-0c5c-4d85-9b7c-33c380e60e24",
   "name": "Node_1",
   "description": "First router",
   "loopback": "6.6.6.1/32",
   "intefaces": [
      {
         "name": "Ge0/0/0/0",
         "ip": "1.1.1.1/24",
         "status": "up"
      }
   ]
}
```

Update Node_2 GE interface status (from down to up)

API Endpoint : `MOCKUP` **Method** : `PATCH`
Path : `api/project/{{projectId}}/router/{{routerId2}}`
Payload :

```
{
   "intefaces": [
```

```
        {
           "name": "Ge0/0/0/0",
           "ip": "1.1.1.2/24",
           "status": "up"
        }
     ]
  }
```


Response : `200 OK`


```
{
   "uuid": "374eb367-df28-4bca-811a-9cc30b4b9dfd",
   "name": "Node_2",
   "description": "Second router",
   "loopback": "6.6.6.2/32",
   "intefaces": [
      {
         "name": "Ge0/0/0/0",
         "ip": "1.1.1.2/24",
         "status": "up"
      }
   ]
}
```


Post-Check

Ping Node_2 interface from Node_1 (Expected Failure)

A ping to Node_2 interface from Node_1 should be now sucessfull.

API Endpoint : `MOCKUP` **Method** : `POST`
Path : `api/project/{{projectId}}/router/{{routerId1}}/live-status/exec/any`
Payload :


```
{
   "input": {
      "args": "ping 1.1.1.2 source 1.1.1.1"
   }
}
```


Response : `200 OK`


```
{
   "result": "Sending 5, 100-byte ICMP Echos to 1.1.1.2, timeout is 2 seconds: Success rate is
100 percent (5/5), round-trip min/avg/max = 3/3/4 ms"
}
```