# Create a workflow tutorial

This section contains the following topics:

# Create a workflow – tutorial

This chapter shows you how you can structure a workflow based on a simple example that uses the `operation` and `switch` states to create a VPN service in Cisco NSO for some simulated devices. We go through the example workflow definition part by part to give you an idea how you can use different definition components in creating your original workflows.

If you need full information on how workflows can be defined, refer to the Serverless Workflow specification.

## Example workflow overview

Workflows can be written either in JSON or YAML. For the example purpose, we'll pick the JSON formatting.

The purpose of the example workflow is to automatically create a VPN service for Cisco NSO devices.

First we point to the devices in the data input and then try to perform the NSO `check-sync` operation on them. Then, depending on the result:

- if not in sync, we push a device to perform a `sync-from`, and only then try to create a VPN for it;
- if in sync, we don't perform `sync-from` but directly create a VPN for the device.

If all the steps are executed successfully, CWM reports workflow execution completion and diplays the final data input. The results are visible in Cisco NSO too. If the engine encounters errors while performing a step, it uses the specified `retry` policy. In case errors persist beyond the retry limits, the execution engine ends the execution with a **Failed** status.

Go through the sections below to learn the details of how data input, functions, states, actions, and data filters are defined.

## Provide data input

The workflow definition usually includes some input data at beginning of the JSON file. While the provided data is not part of the workflow, it is referred to within the workflow definition and can also be updated between states, if such a data update is defined. For more details, see Workflow data input in the Serverless Workflow Specification.

In the quickstart example, we'll only need to provide two user-defined `deviceName` JSON object keys and values, which are the names of the test devices in the local NSO instance, and the `nsoResource` key, where we specify which CWM resource will be used in the workflow. The workflow data input in JSON should therefore look like this:

```
{
"device0Name": "ce0",
"device1Name": "ce1",
"nsoResource": "NSOLocal"
}
```

## Define top-level parameters and functions

A workflow definition starts with the required workflow `id` key. Among other keys, `specVersion` is also required, defining the Serverless Workflow specification release version. The `start` key defines the name of the workflow starting state, but it is not required. In the `functions` key, you pass in Cisco NSO adapter activity name as function `name`, adapter activity ID as function `operation`, and provide the worker name under `metadata`: `worker` key:

```
{
    "id": "CreateL3VPN",
    "version": "1.0",
    "specVersion": "0.9",
    "name": "Create Layer3 VPN",
    "description": "Create an L3 VPN for MPLS devices",
    "start": "start",
    "functions": [
    {
        "name": "NSO.RestconfPost",
        "operation": "cisco.nso.v1.0.0.restconf.Post",
        "metadata": {
          "worker": "cisco.nso.v1.0.0"
        }
    }
  ],
}
```

**Note** Effectively, what you do under `functions` is you provide the workflow with the IDs of any activities as they are defined in the Cisco NSO adapter and presented in its `main.go` file. Also, under `metadata` you provide the name of the worker that will execute any actions that refer to the defined function.

## Specify retry policy

With the `retries` key, you define the retry policies for state actions in the event that an action fails. Multiple retry policies can be specified under this key and they are reusable across multiple defined state actions.

```
"retries": [
    {
      "name": "Default",
      "maxAttempts": 4,
      "delay": "PT5S",
      "multiplier": 2
    },
        {
          "name": "Custom",
          "maxAttempts": 2,
          "delay": "PT30S",
          "multiplier": 1
        }
      ],
```

**Note** As you can see, the `Default` policy assumes that a failed action will be retried up to 4 times with an increasing delay between attempts: 5, 10, 20, 40 seconds between consecutive retries.

## Define states

Workflow states are the building blocks of a workflow definition. In the present quickstart example, we will be using the `operation` and `switch` states, but others are possible. You can check them in detail in the Workflow states section of the Serverless specification.

### operation state

```
"states": [
        {
        "name": "start",
        "type": "operation",
        "stateDataFilter": {
            "input": "${ . }"
        },
        "actions": [],
        "transition": {
            "nextState": "syncFromOrCreateVPN"
            }
        }
    ]
```

Inside the `operation` state, apart from state `name` and `type`, you define:

- `stateDataFilter` - point to the data input defined at beginning of example JSON file. In the `input` parameter, we state `${ . }`, which is a jq expression that means: use the whole of data input existing at this point of workflow execution.

  **Note** For more information on how jq expressions are used in workflows, see the Workflow expressions chapter in the Serverless Workflow specification.

- `actions` - specify the function to be used by the action, and two basic `arguments`: `input` and `config`. Read more in the subsection below.
- `transition` or `end` - point to the next state to which the workflow should transition after executing the present one. If there are no more steps to be executed, use `end`.

### switch state

```
          {
            "name": "syncFromOrCreateVPN",
        "type": "switch",
        "dataConditions": [
           {
             "name": "shouldSyncFrom",
            "condition": "${ if (.checkSyncResult0) then  .checkSyncResult0 != \"in-sync\"
 else null end }",
             "transition": {
               "nextState": "syncFrom"
             }
           },
           {
             "name": "shouldCreateVPN",
            "condition": "${ if (.checkSyncResult0) then  .checkSyncResult0 == \"in-sync\"
 else null end }",
            "transition": {
               "nextState": "createVPN"
             }
          }
         ]
      }
```

Inside the `switch` state, apart from state `name` and `type`, you define:

- `dataConditions` - define the conditions to be met by a device to be transitioned to a specified next state. You can view the `switch` state as a "gateway" for the workflow which directs the devices to appropriate states based on their status. Using the jq expression `${ if (.checkSyncResult0) then .checkSyncResult0 == \"in-sync\" else null end }` in the `condition` parameter, we create a boolean value that, if it evaluates to `true`, is used to transition the device directly to the `CreateVPN` state.

## Specify actions

Let's analyse `actions` on the basis of the `checkSync` action of the `operation` state for device `ce0`.

```
{
"name": "checkSync",
"retryRef": "Default",
"functionRef": {
  "refName": "NSO.RestconfPost",
  "arguments": {
    "input": {
      "path": "restconf/operations/tailf-ncs:devices/device=${ .device0Name }/check-sync"
      },
    "config": {
      "resourceId": "${ .nsoResource }"
      }
    }
  },
"actionDataFilter": {
  "results": "${ if (.data) then .data | fromjson.\"tailf-ncs:output\".result else null end
 }",
  "toStateData": "${ .checkSyncResult0 }"
  }
}
```

Among the possible parameters, two are especially useful to consider:

- `functionRef` - refer to the function (aka an activity, from the NSO adapter perspective) to be used in action execution. Here, you need to pass in some `arguments`:
    - `input`:
        - `path` - point to a path for the adapter to send the request to.

        - `data` - forward any data to be included in the request (not applicable for the `checkSync` action).

    - `config`:

        - `resourceId` - provide ID of the resource you created for an external service. In the example workflow, the local host and the default port of the Cisco NSO instance is provided. The resource also points to the secret ID, which is used to provide authentication data for an external service: in this case, `username` and `password` to the Cisco NSO instance.

- `actionDataFilter` - define how to process the data passed on in the `checkSync` response from NSO:

    - `results` - use the jq expression `"${ if (.data) then .data | fromjson.\"tailf-ncs:output\".result else null end }"` to handle incoming NSO data. By using `fromjson` you translate the resultant `tailf-ncs:output` into JSON formatting, then with `.result` you cherrypick the `result` key value. In this case (if the device is in the in-sync state), the output of the expression would be `"in-sync"`.

    - `toStateData` - take the output of the expression defined in the `results` parameter above and save it as a key and value pair inside the workflow input data under any name that you pick: in this case, `.checkSyncResult0`.

## Example workflow definition

The following example workflow definition is the end result of the workflow creation process presented in this chapter:

```
{
  "id": "CreateL3VPN-1.0",
  "name": "CreateL3VPN",
  "start": "start",
  "states": [
    {
      "name": "start",
      "type": "operation",
      "actions": [
        {
          "name": "checkSync",
          "retryRef": "Default",
          "functionRef": {
            "refName": "NSO.RestconfPost",
            "arguments": {
              "input": {
                "path": "restconf/operations/tailf-ncs:devices/device=${ .device0Name
}/check-sync"
              },
              "config": {
                "resourceId": "${ .nsoResource }"
              }
            }
          },
          "actionDataFilter": {
            "results": "${ if (.data) then .data | fromjson.\"tailf-ncs:output\".result
else null end }",
            "toStateData": "${ .checkSyncResult0 }"
          }
        },
        {
            "name": "checkSync",
            "retryRef": "Default",
            "functionRef": {
              "refName": "NSO.RestconfPost",
              "arguments": {
                "input": {
                  "path": "restconf/operations/tailf-ncs:devices/device=${ .device1Name
}/check-sync"
                },
                "config": {
                  "resourceId": "${ .nsoResource }"
                }
              }
            },
            "actionDataFilter": {
              "results": "${ if (.data) then .data | fromjson.\"tailf-ncs:output\".result
else null end }",
              "toStateData": "${ .checkSyncResult1 }"
            }
          }
        ],
        "transition": {
          "nextState": "syncFromOrCreateVPN"
        },
        "stateDataFilter": {
          "input": "${ . }"
        }
      },
      {
        "name": "syncFromOrCreateVPN",
        "type": "switch",
        "dataConditions": [
          {
            "name": "shouldSyncFrom",
            "condition": "${ if (.checkSyncResult0) then  .checkSyncResult0 != \"in-sync\"
else null end }",
            "transition": {
              "nextState": "syncFrom"
            }
          },
          {
            "name": "shouldCreateVPN",
            "condition": "${ if (.checkSyncResult0) then  .checkSyncResult0 == \"in-sync\"
else null end }",
            "transition": {
              "nextState": "createVPN"
            }
          },
          {
            "name": "shouldSyncFrom",
            "condition": "${ if (.checkSyncResult1) then  .checkSyncResult1 != \"in-sync\"
else null end }",
            "transition": {
              "nextState": "syncFrom"
            }
          },
          {
            "name": "shouldCreateVPN",
            "condition": "${ if (.checkSyncResult1) then  .checkSyncResult1 == \"in-sync\"
else null end }",
            "transition": {
              "nextState": "createVPN"
            }
          }
      ],
      "defaultCondition": {
        "end": {
          "terminate": true
        }
      }
    },
    {
      "name": "syncFrom",
      "type": "operation",
      "actions": [
        {
          "name": "syncFrom",
          "retryRef": "Default",
          "functionRef": {
            "refName": "NSO.RestconfPost",
            "arguments": {
              "input": {
                "path": "restconf/operations/tailf-ncs:devices/device=${ .device0Name
}/sync-from"
              },
              "config": {
                "resourceId": "${ .nsoResource }"
              }
            }
          },
          "actionDataFilter": {
            "results": "${ if (.data) then .data | fromjson.\"tailf-ncs:output\".result
else null end }",
            "toStateData": "${ .syncFromResult0 }"
          }
        },
        {
          "name": "syncFrom",
          "retryRef": "Default",
          "functionRef": {
            "refName": "NSO.RestconfPost",
            "arguments": {
              "input": {
                "path": "restconf/operations/tailf-ncs:devices/device=${ .device1Name
}/sync-from"
              },
              "config": {
                "resourceId": "${ .nsoResource }"
              }
            }
          },
          "actionDataFilter": {
            "results": "${ if (.data) then .data | fromjson.\"tailf-ncs:output\".result
else null end }",
            "toStateData": "${ .syncFromResult1 }"
          }
        }
      ],
      "transition": {
        "nextState": "createVPN"
      }
    },
    {
      "end": {
        "terminate": true
      },
      "name": "createVPN",
      "type": "operation",
      "actions": [
        {
          "name": "createVPN",
          "retryRef": "Custom",
          "functionRef": {
            "refName": "NSO.RestconfPost",
            "arguments": {
              "input": {
                "data":
"[illegible]
 ",
                "path": "restconf/data/l3vpn:vpn"
              },
              "config": {
                "resourceId": "${ .nsoResource }"
              }
            }
          },
          "actionDataFilter": {
            "results": "${ if (.status) then .status else null end }",
            "toStateData": "${ .createServiceResult }"
          }
        }
      ]
    }
  ],
  "retries": [
    {
      "name": "Default",
      "delay": "PT30S",
      "multiplier": 2,
      "maxAttempts": 4
    },
    {
      "name": "Custom",
      "delay": "PT10S",
      "multiplier": 1,
      "maxAttempts": 2
    }
  ],
  "version": "1.0",
  "functions": [
    {
      "name": "NSO.RestconfPost",
      "metadata": {
        "worker": "cisco.nso.v1.0.0"
      },
      "operation": "cisco.nso.v1.0.0.restconf.Post"
    }
  ],
  "description": "",
  "specVersion": "0.9"
}
```

For a complete procedure on how to execute the example workflow in CWM and Cisco NSO to get tangible results, see the Quick Start guide