



PHYSICS

HOW TO USE THE PHYSICS DESIGN ENVIRONMENT GUIDE

Contents

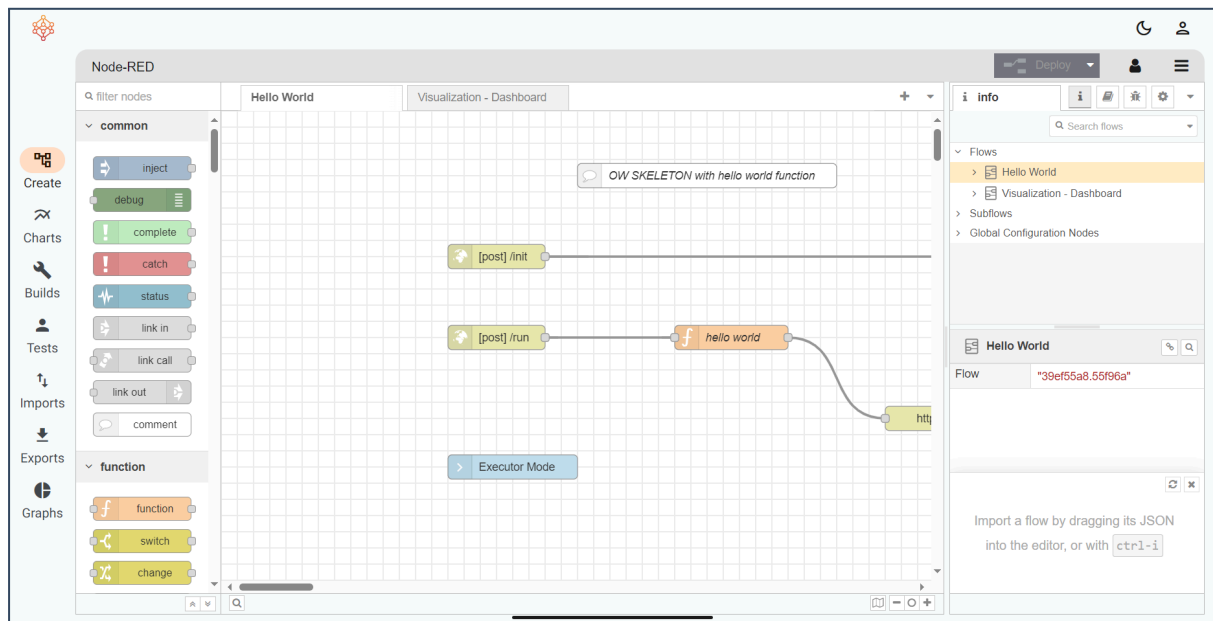
Introduction	3
Authentication	3
First login	4
Create	4
Base flow	5
PHYSICS Provided Palette of Patterns and Helper Flows	6
PHYSICS Semantic Annotator Nodes	7
Extending the palette with external Node-RED nodes	7
Customization of the Dockerfile for External Dependency Inclusion	8
Charts	8
First login	9
Step one	10
Step two	10
Step tree	11
Step four	11
Step five	12
Builds	12
Start a build	13
Tests	14
Request a test	16
Local platform (Openwhisk)	17
Performance	18
View of Performance Pipeline Results in the Dashboard	19
Imports	20
Start the import	21
Exports	23
Preparatory work	24
Final export process to npm and Node-RED repos	25
Application Graphs	27
Create a new Application Graph	28
Inclusion of an imported image to an Application Graph	28
Log	29
Flow Update Process	30
FAQs and Common Errors	33
Step 1: Start the environment locally	33
STEP 2: Work in the environment and try to test deploy on OW	35
OPTIONAL STEP: Customize the baseline environment image	35
Troubleshoot	36
Update images of Design Environment	37
Potential build fail when changing flows	37
Potential Break of the git updates	38

Subflow in Subflow	38
Memory considerations of invoked action	38
Size of output message	39
Update of Base Image	40
Git Dubious Ownership error	40
Subflow id not iterable	41
Multiple /run Endpoints in Node-RED	41
Parallelization Strategies	41

INTRODUCTION

In the following document will be explained how to use the visual interface deployed on cloud that require a internet connection and a Chrome browser v.118 o sup. The interface is available on URL <https://control-ui.apps.ocphub.physics-faas.eu/>.

There's also a downloadable version that needs a local docker installation, you can find the process of the installation in the dedicated chapter “Install local version”; keep in mind that the layout is different between the two versions, but the main features are almost the same.



The Interface, is compose by four parts

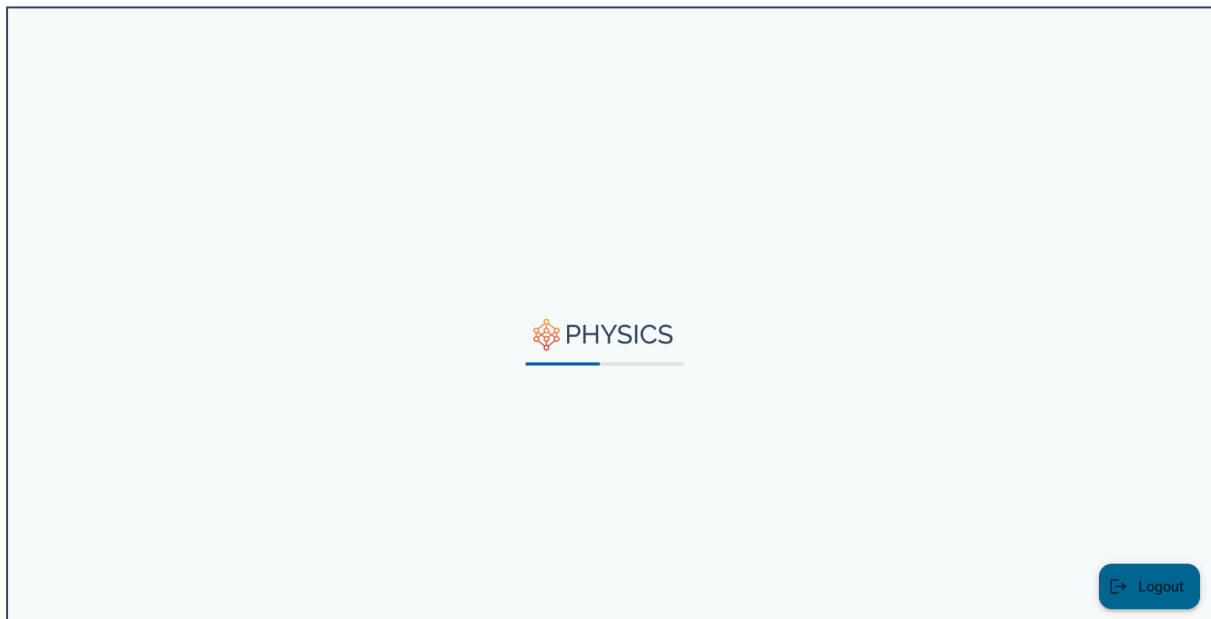
- **Top bar:** where you can switch between the light and dark mode and the user icon, which when pressed shows the logged user name, a link to download the last version of this guide and the logout button .
- **Left menu:** a rail menu for moving in all the available sections of the interface.
- **Bottom line:** if pressed showed a panel with the user backend log.
- **Main:** the center of the page that shows the content of the opened section.

In the following chapter we deep dive into each section and the log.

Authentication

For access, the GUI requires a valid user account on Physics platforms. If you don't have a valid user, you can create yourself by accessing the interface (<https://control-ui.apps.ocphub.physics-faas.eu/>) creations user, please contact the system administrator to get it.

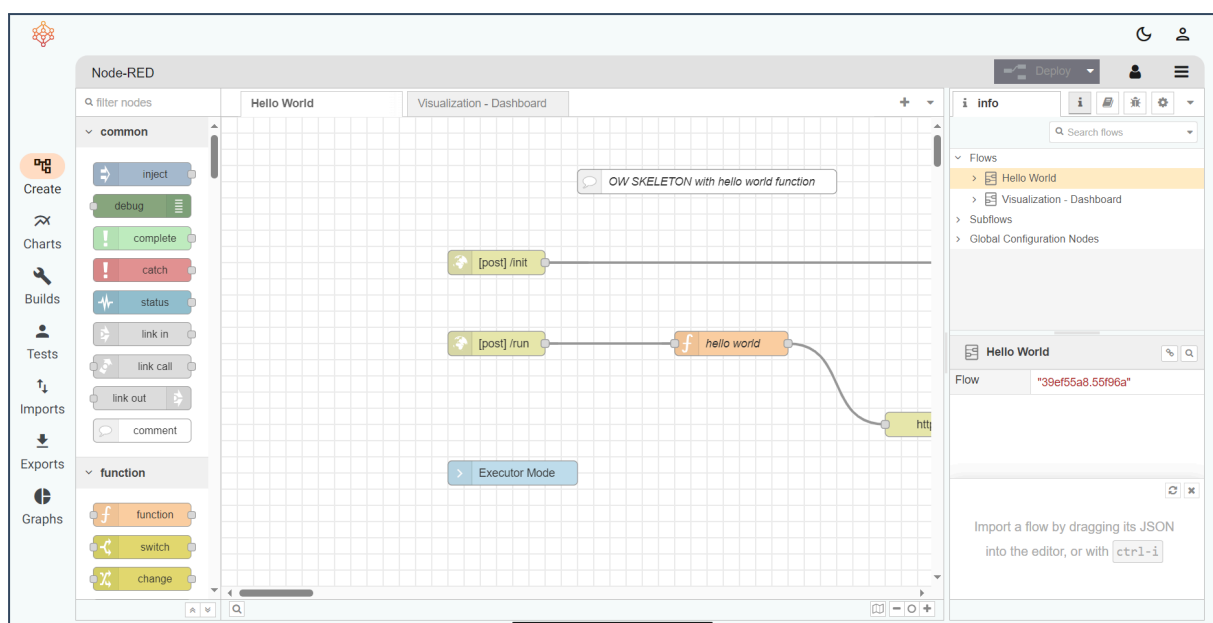
First login



On the first login you will guide through two manual steps to provisioning your personal environment. The first one requires first access to the Physics versioning system with your user account and the second is the requesting of your password to start the creation of your personal environment. The first login takes approximately 4 minutes, the next login requires only the time to start up your environment.

It's warmly recommended to enable the browser notifications, to receive notifications of completed builds so you don't need to keep the page open to check the end of the job.

CREATE

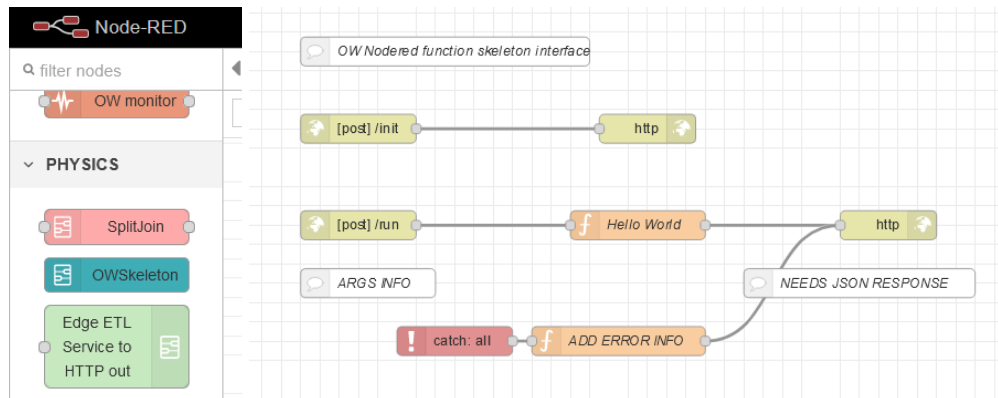


In the create section you can use your personal Node-red that is a programming tool for wiring together hardware devices, APIs and online services in new and interesting ways.¹ This tool permits the creation of a flow in a block code way, each block is declared as a **node**, that it can be an atomic custom function written in javascript or python, a prebuilt node (es. Access a db) and a reference to a subflow or to another flow.

For more information about node-red environments please refer to the official guide on <https://nodered.org/docs/>.

Base flow

Each flow, to be running on physics environments, needs to include two **http in node**, configured in POST method and respectively with URL /init and /run, below is reported a basic physics flow. A helper subflow is available in the PHYSICS patterns palette (OW Skeleton node), the contents of which appear below, including instructions on how the arguments are passed from Openwhisk as well as what type of response is needed. Adding an error node helps in better debugging your flows. Be aware that if you want to add an “error” field in the response, this will be translated by Openwhisk as an erroneous execution and will appear as such in the Openwhisk activation results. Therefore if one needs to add some extra info on partial errors noticed within an execution, they should use some other notation for the error field in the JSON response.



In the flow is present the following node:

- **2 http in:** as described before is needed for the physics environment
- **1 function:** where is present the business logic of this flow
- **2 http response:** for close the session of the flow
- **1 comment:** an example node where you can insert some comment
- **1 custom node:** the *Executor Mode*, is one of the custom node made for the physics environment

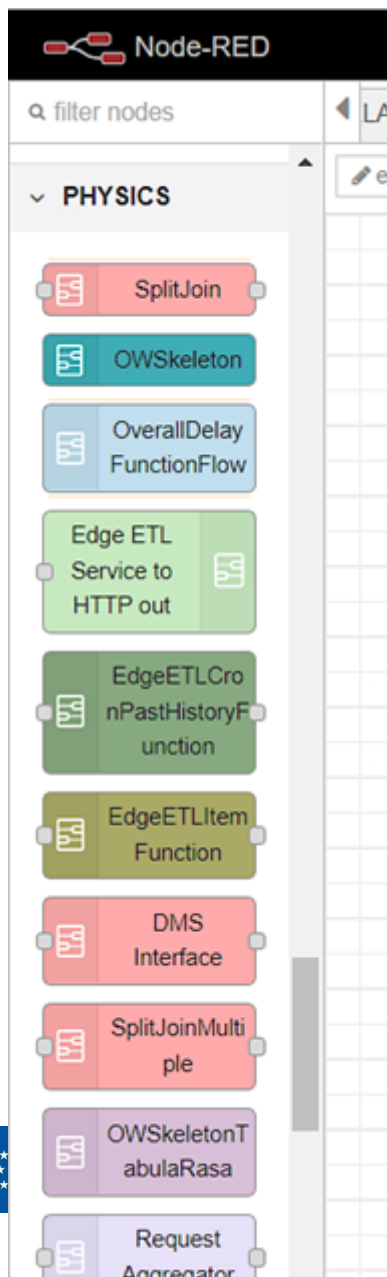
¹ [Node-RED \(nodered.org\)](https://nodered.org/)

For more information about the standard node or the available node downloadable from node-red repository please refer to the official node-red documentation.

Important notice 1: within the PHYSICS platform, the **branch name** of the developer as well as the **flow name** are used to create unique identifiers for a function. Thus if one creates such a flow and then goes through various stages like the performance pipeline, all the results and filters will be based on the flow+branch name attributes. Therefore it is recommended that the flow name is not changed during development or if it is, then the developer should repeat the stages involving usage of these attributes as filters (such as the performance statistics of a function).

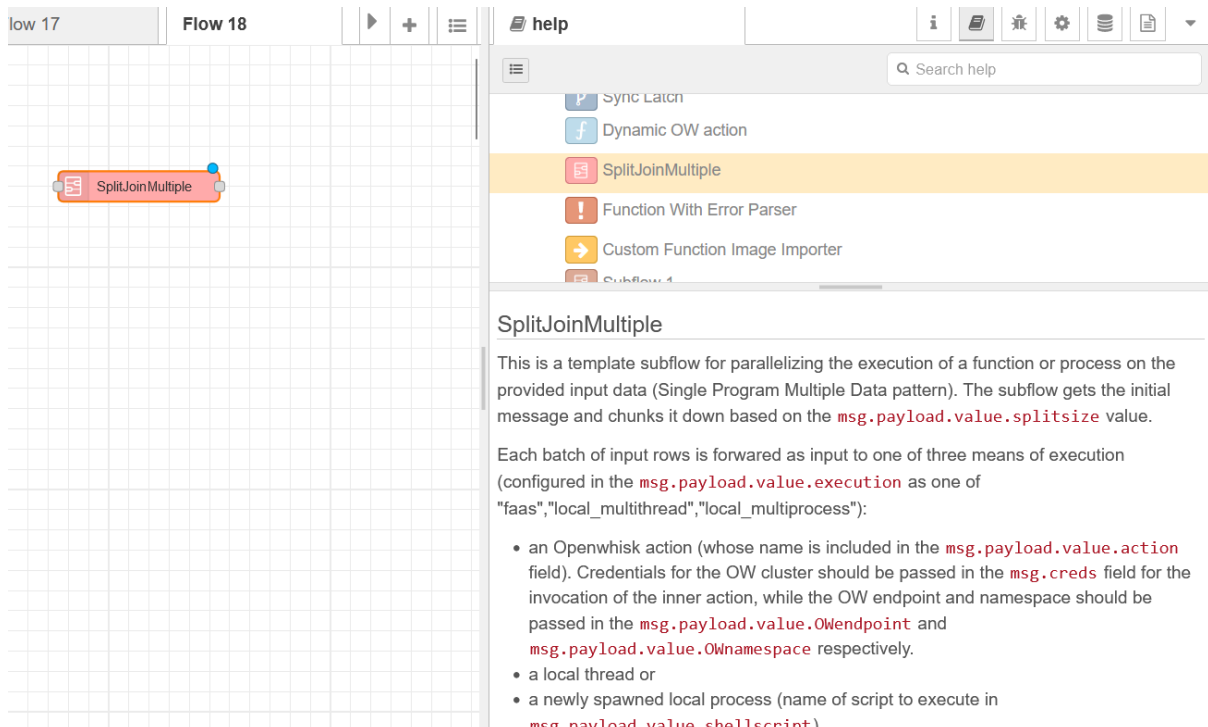
Important notice 2: due to the K8s naming conventions, refrain from using capital letters or special characters like underscores in flow names. K8s will automatically lowercase and replace “_” with “-”.

PHYSICS Provided Palette of Patterns and Helper Flows



The Design Patterns aim at offering reusable and parametric operational capabilities to the developers in order to enable an easier and more abstracted flow creation process. Patterns have been created for workflow enhancement, load distribution, message manipulation etc. In the final version, new patterns have been included in order to automate aspects such as routing between available endpoints, dynamic orchestration of functions etc.

Each pattern or subflow/node comes with its own specification in relation to its usage. The interfaces are through the fields of the incoming message to the subflow. The information is included in each pattern documentation to be directly accessible in the Node-RED environment by the developer. Once dragged inside a flow and selected, the documentation appears by selecting the relevant icon in the right Node-RED panel side. This documentation includes the specification of the incoming messages as well as configuration information. Most of the patterns are configurable also through the UI. The incoming messages values however prevail over the manually set UI values for enabling more dynamic configuration.



The screenshot shows the Node-RED web interface. On the left, a canvas displays a 'SplitJoinMultiple' node. On the right, the 'help' sidebar is open, showing a list of nodes with 'SplitJoinMultiple' selected. Below the list, the documentation for 'SplitJoinMultiple' is displayed, explaining its purpose as a template subflow for parallelizing execution and listing its configuration options.

SplitJoinMultiple

This is a template subflow for parallelizing the execution of a function or process on the provided input data (Single Program Multiple Data pattern). The subflow gets the initial message and chunks it down based on the `msg.payload.value.splitsize` value.

Each batch of input rows is forwarded as input to one of three means of execution (configured in the `msg.payload.value.execution` as one of "faas", "local_multithread", "local_multiprocess"):

- an Openwhisk action (whose name is included in the `msg.payload.value.action` field). Credentials for the OW cluster should be passed in the `msg.creds` field for the invocation of the inner action, while the OW endpoint and namespace should be passed in the `msg.payload.value.OWendpoint` and `msg.payload.value.OWnamespace` respectively.
- a local thread or
- a newly spawned local process (name of script to execute in `msg.payload.value.shellscript`)

This type of distribution has the benefit that the developer can also edit the subflow template, meaning that they can change some part of the pattern implementation to better suit their needs. On the other hand, it means that the developer needs to have pre-installed any external dependency (e.g. a Node-RED node that is not in the default Node-RED environment).

PHYSICS Semantic Annotator Nodes

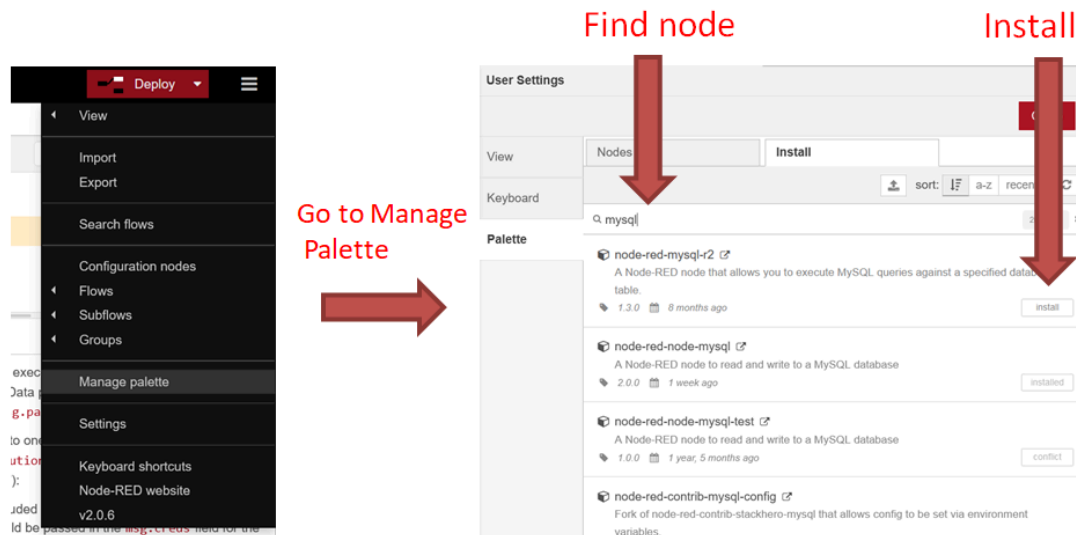
At the flow level, a special set of nodes (semantic annotators) has been created as subflows and included in the Node-RED palette. Node-RED offers the ability to create subflows, in which one can define the required fields (e.g. in a UI format). These fields are included as subflow properties and environment variables. Various nodes have been implemented up to this point in order to address one or more categories of annotations needed at the flow level that give directives to the underlying management layers. Each of the semantic nodes is also accompanied by a relevant README file accessible in the Node-RED environment. Through these nodes the developer can dictate needs for function sizing, locality preferences, optimization preferences etc.

Extending the palette with external Node-RED nodes

Node-RED comes with a wide extension of available open source nodes². In order for the developer to use one, they first need to install them through the typical Node-RED UI

² <https://flows.nodered.org/>

process. Once installed inside the Design Environment, they can use them inside their flows. Upon building of the function image, the imported dependencies will be automatically included in the image so that it is directly operational when executing inside Openwhisk.



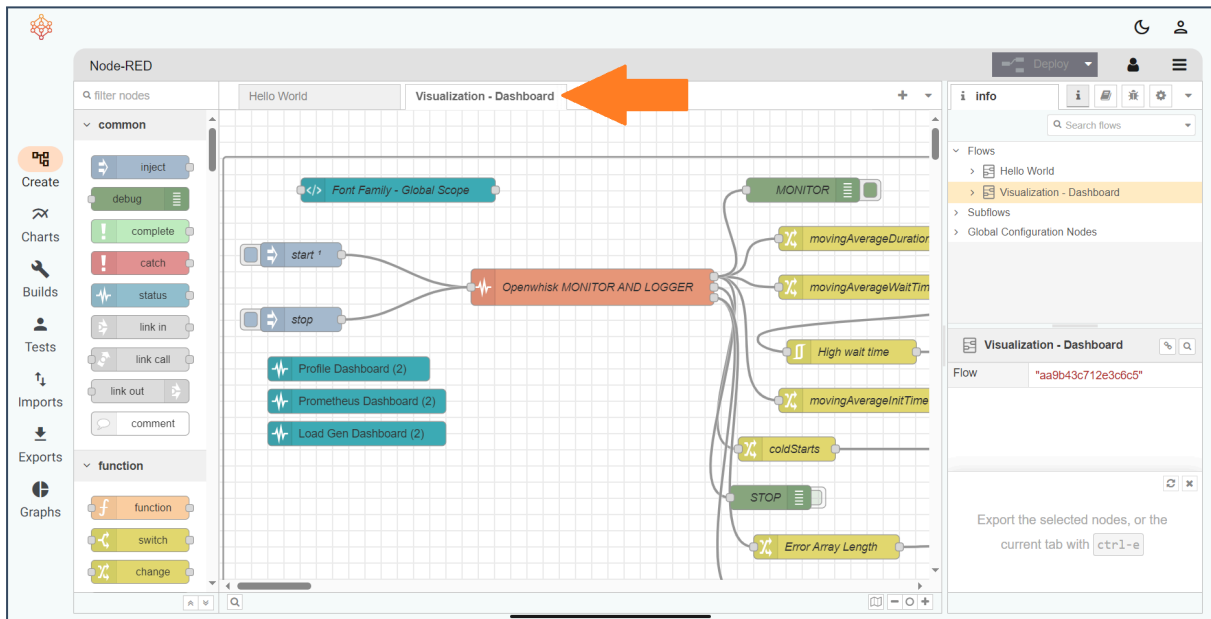
First login

On the first login, for a security purpose, it is required to set the username and the password in some subflow on the “Visualization - Dashboard” flow you can find in Node-RED on section Create.

Keep aware to not delete the flow “Visualization - Dashboard”.

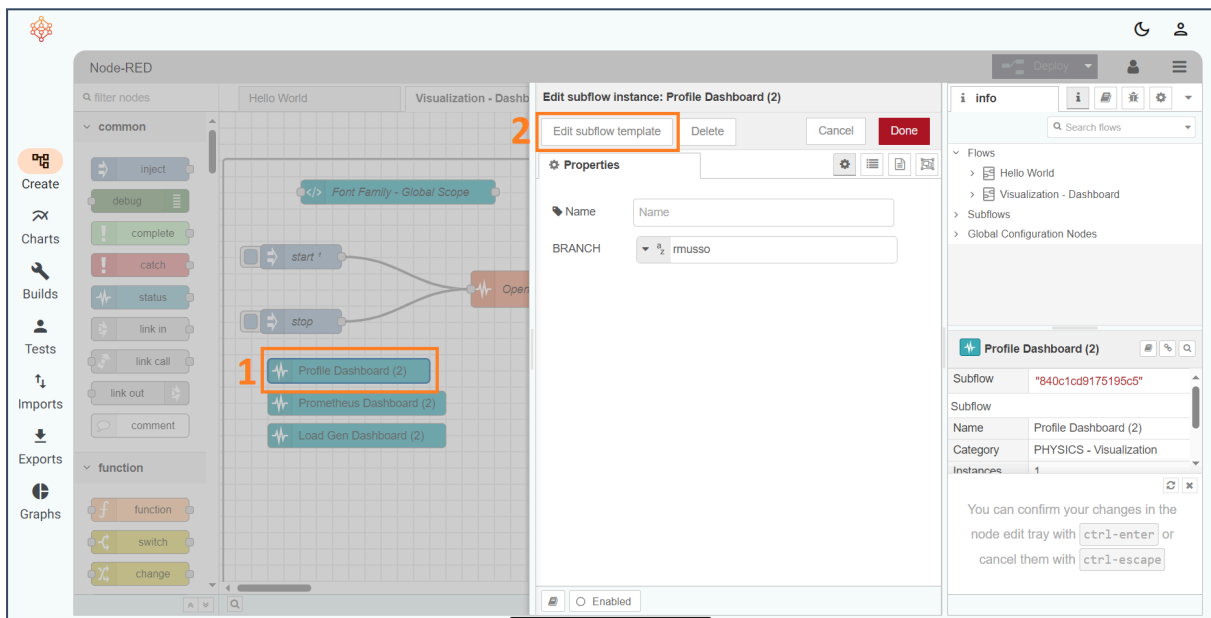
Step one

Open the flow “Visualization - Dashboard” in Create section



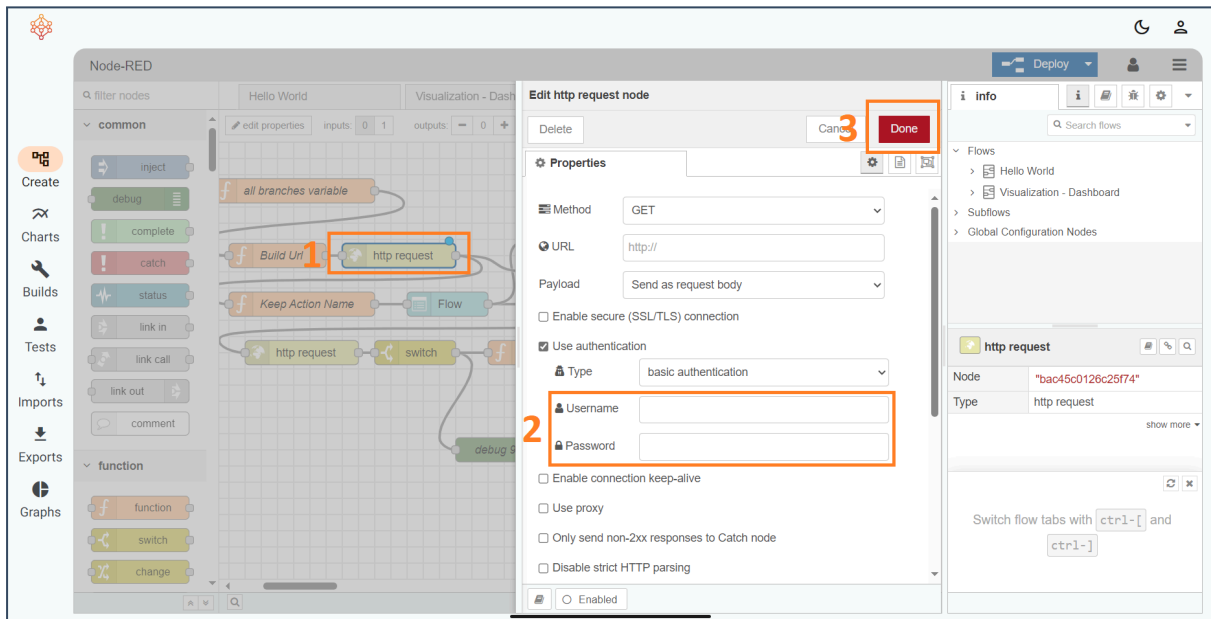
Step two

Click two time on “Pofile Dashboard (2)” and then open “Edit subflow template”



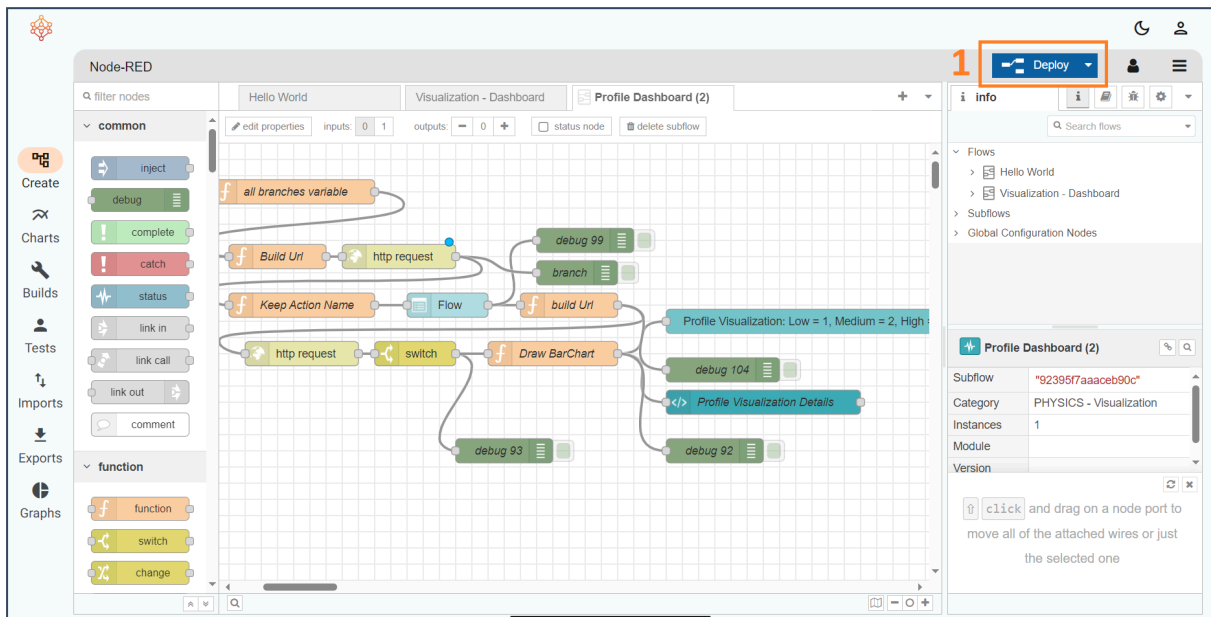
Step tree

Click two times on the http request block. In the panel set **admin** as Username and **bigds** as Password then click on **Done**.



Step four

Click on **Deploy** button and you will see a pop up that confirm the successfully deploy

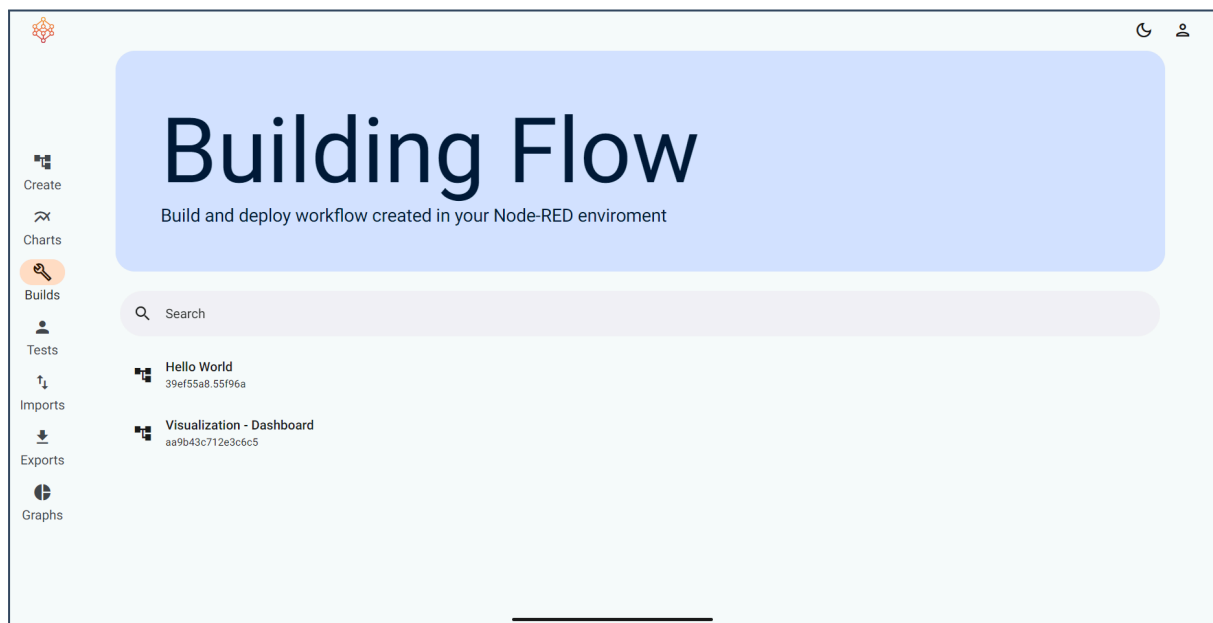


Step five

Repeat the step from two to four for the block “Prometheus Dashboard (2)” and “Load Gen Dashboard (2)”.

BUILDS

In the **builds** section, is shown the list of the flows present on your Node-RED and your deleted flow if you are made at least one build of it, for each flow is also present the related artifacts. It also presents a search bar to filter the flows.



When pressed on a flow, it is showing the related available artifact, with the following information in this order for each artifact:

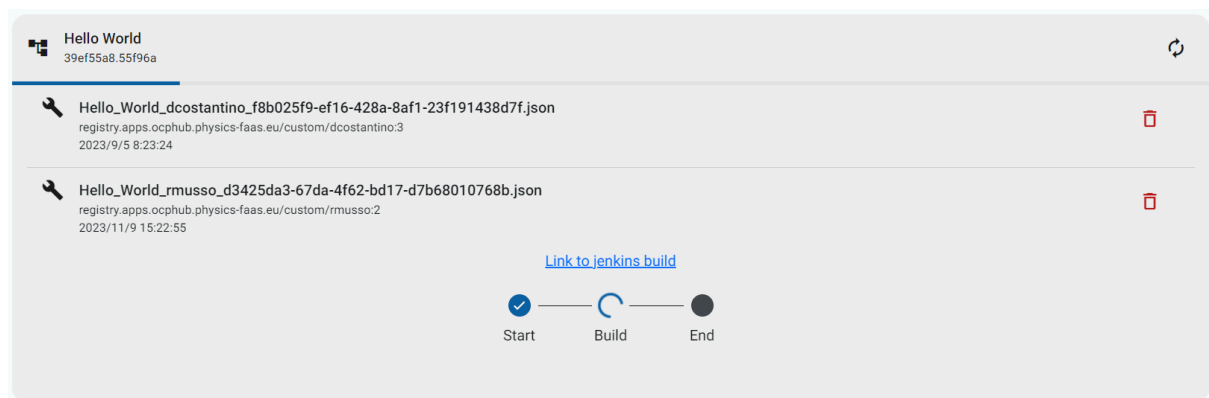
- **Action name:** the name of the Openwhisk action associated to the artifact
- **Image:** the position of the container created in the build process
- **Build date:** the date of the requested build

Start a build

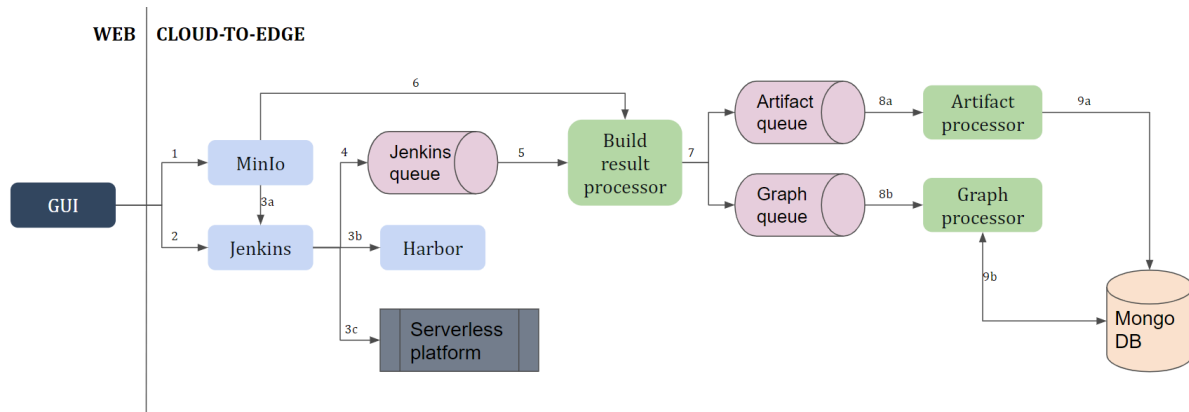
To start the build process, open the flow and press the **build** button (or **rebuild** if is already present an *artifact* of the flow)



After the build starts you can see the state of the process on jenkins pipeline or you can access on jenkins pipeline with the link “Link to jenkins job”.



The deploy consists of the following workflow, where after that it will be created an **action name** to call the function on the serverless platform in the test section.



In the diagram flow is present all the custom components involved in the deployment process with the numbered sequence. The main component that orchestrate all the process is a *jenkins*³ pipeline.

TESTS

In this section you can test the function deployed, to see if it works as expected or to get performance data.



In the main section is reported a filterable list of all your runned tests, if you click on one of them, it is reported the result of the test with the parameter used to run it.

³ Jenkins is an open source automation server. It helps automate the parts of software development related to building, testing, and deploying, facilitating continuous integration and continuous delivery. ([jenkins \(software\) - Wikipedia](https://en.wikipedia.org/wiki/Jenkins_(software)))


Run_calibration_smartAgri_dbfe0f0a-999c-4eb4-bb06-d0497572856c.json
2023-12-11T13:46:38.215Z
✓

Action*

Run_calibration_smartAgri_dbfe0f0a-999c-4eb4-bb06-d0497572856c.json

JSON params

```
{
  "input": [
    {
      "TM": "sum",
      "prec": "sum",
      "net_radiation": "sum",
      "RH": "sum"
    },
    {
      "TM": "sum",
      "prec": "sum",
      "net_radiation": "sum",
      "RH": "min"
    },
    {
      "TM": "sum",
      "prec": "sum",
      "net_radiation": "sum",
      "RH": "max"
    },
    {
      "TM": "sum",
      "prec": "sum",
      "net_radiation": "min",
      "RH": "sum"
    }
  ]
}
```

Openwhisk
Performance

Result

```
{
  "predictions": [
    {
      "RH": "sum",
      "TM": "sum",
      "net_radiation": "sum",
      "prec": "sum",
      "score": 1.227140168463633
    },
    {
      "RH": "min",
      "TM": "sum",
      "net_radiation": "sum",
      "prec": "sum",
      "score": 0.6547932975811974
    }
  ]
}
```

Repeat
Delete

It is also available in the test report, the buttons to **delete** the test or to **request** a new test with the same data in input.

Each test on the list have one of the following status icon:



The test is running



The test is successfully completed



The test is finished with an error

Request a test

To request a new test you can use the **plus** button in rail menu (1) or you can request a new test with the same parameters as one already made with the button **Repeat** (2).

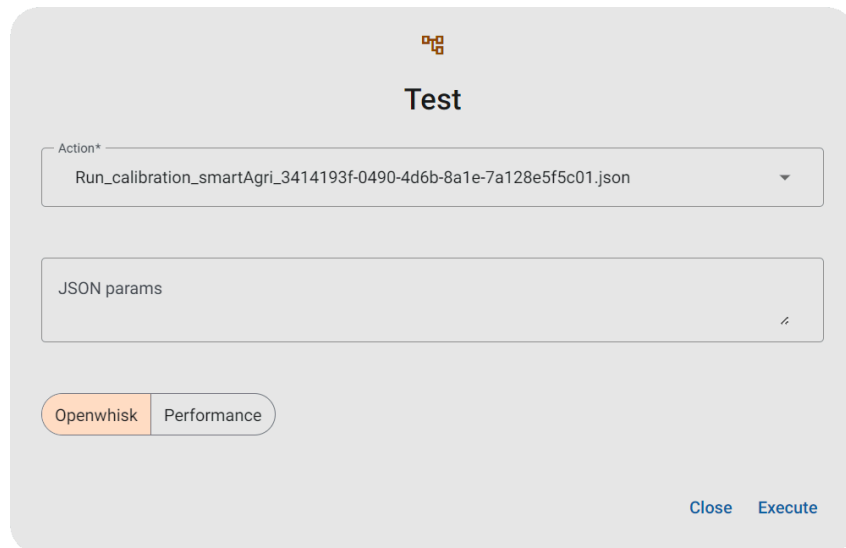


The user can select a function in the list filled with the action present in the local serverless platform. After the selection of the function you can choose to perform the test on the local platform (Openwhisk Test) with a focus on the functionality or to get the performance data.

After the choice of the test and the settings of the parameters (described in the following chapter) you can start the test with the **Execute** button.

After the Execution in the main page you will find a new entry for the running test. At the completion of the test you can see the result.

Local platform (Openwhisk)

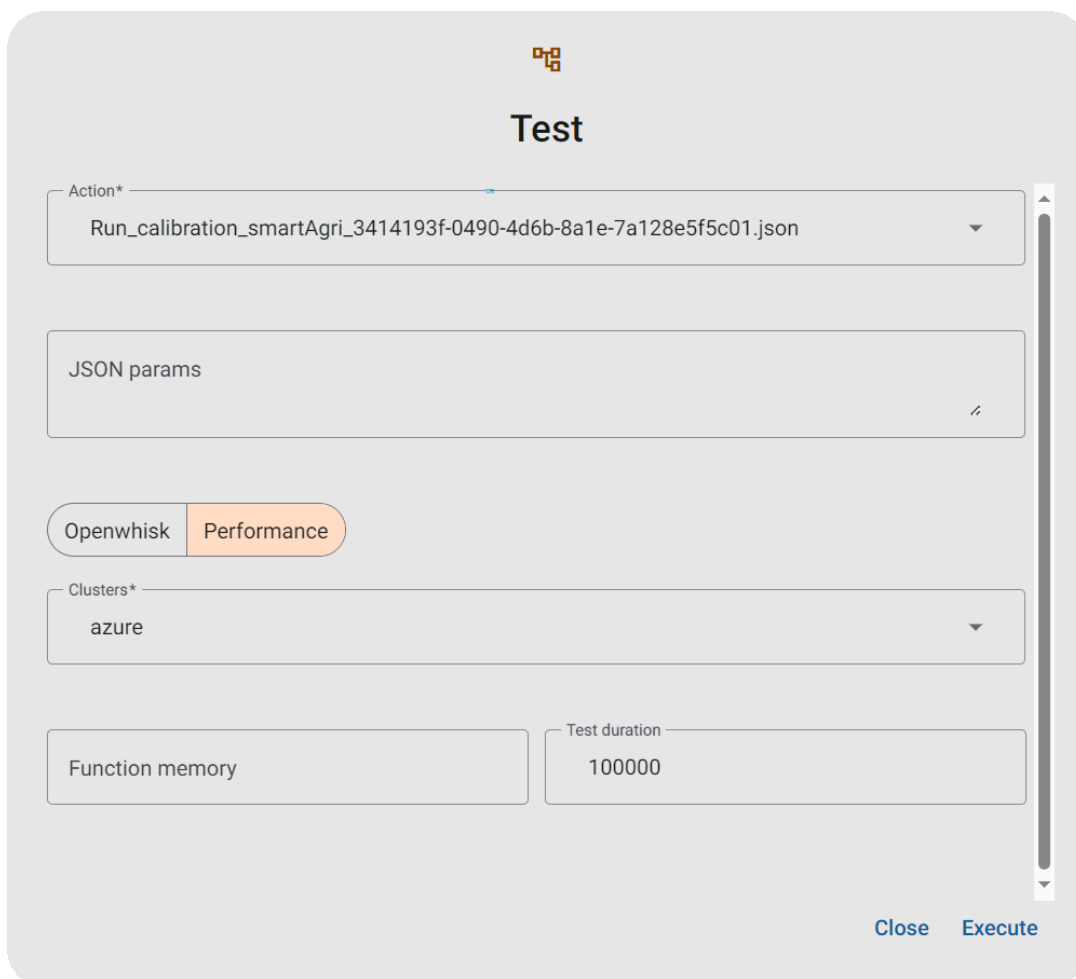


The 'Test' dialog box for the Openwhisk platform shows the following fields and options:

- Action***: A dropdown menu with the selected value 'Run_calibration_smartAgri_3414193f-0490-4d6b-8a1e-7a128e5f5c01.json'.
- JSON params**: A text input field for defining parameters.
- Platform Selection**: Two buttons, 'Openwhisk' (highlighted in orange) and 'Performance'.
- Buttons**: 'Close' and 'Execute' buttons at the bottom right.

In case of selection of Openwhisk in addition to the action you will only have to define the input parameters if necessary.

Performance Test



The 'Test' dialog box for the Performance platform shows the following fields and options:

- Action***: A dropdown menu with the selected value 'Run_calibration_smartAgri_3414193f-0490-4d6b-8a1e-7a128e5f5c01.json'.
- JSON params**: A text input field for defining parameters.
- Platform Selection**: Two buttons, 'Openwhisk' and 'Performance' (highlighted in orange).
- Clusters***: A dropdown menu with the selected value 'azure'.
- Function memory**: A text input field.
- Test duration**: A text input field with the value '100000'.
- Buttons**: 'Close' and 'Execute' buttons at the bottom right.

In case of performance testing you can also select:

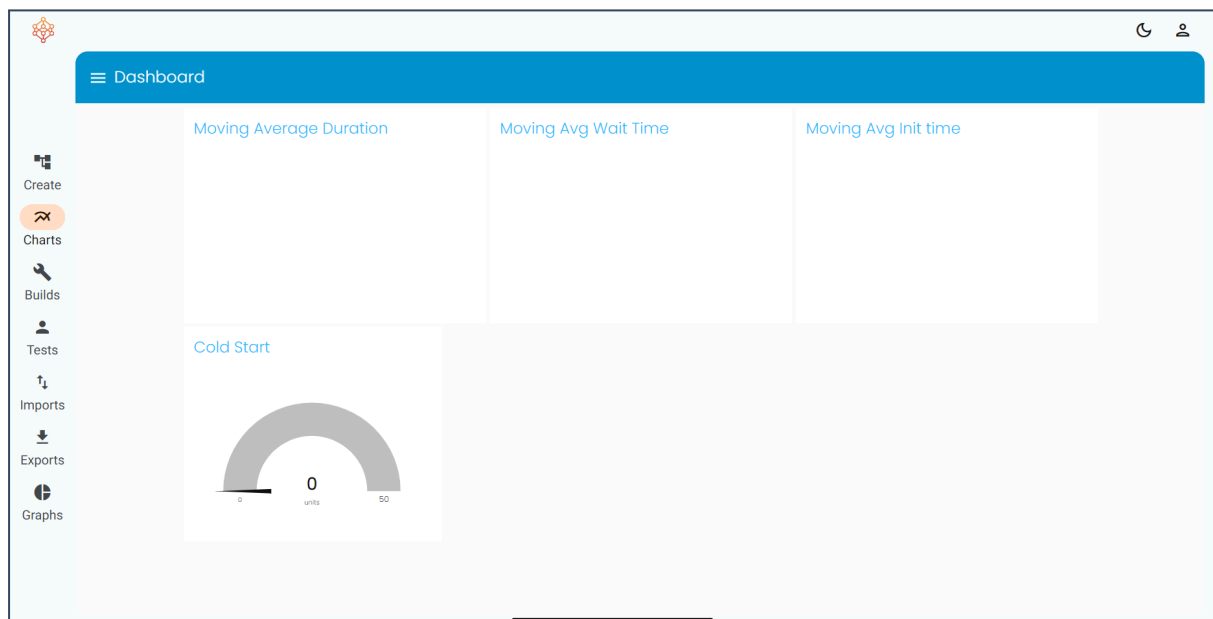
- **Clusters:** drop list with the local and remote platform where you can run your function
- **Function memory** (optional): the number of the maximum memory can be used by the function during the execution
- **Test duration:** the maximum duration of the test

The list in clusters is retrieved in mongoDB on the document **cluster**. In case you want to add a new *remote serverless platform* you can made a request to the administrator with the following information:

- Label to associate to the platform
- The url of the remote serverless platform
- The username and the password to access the remote platform

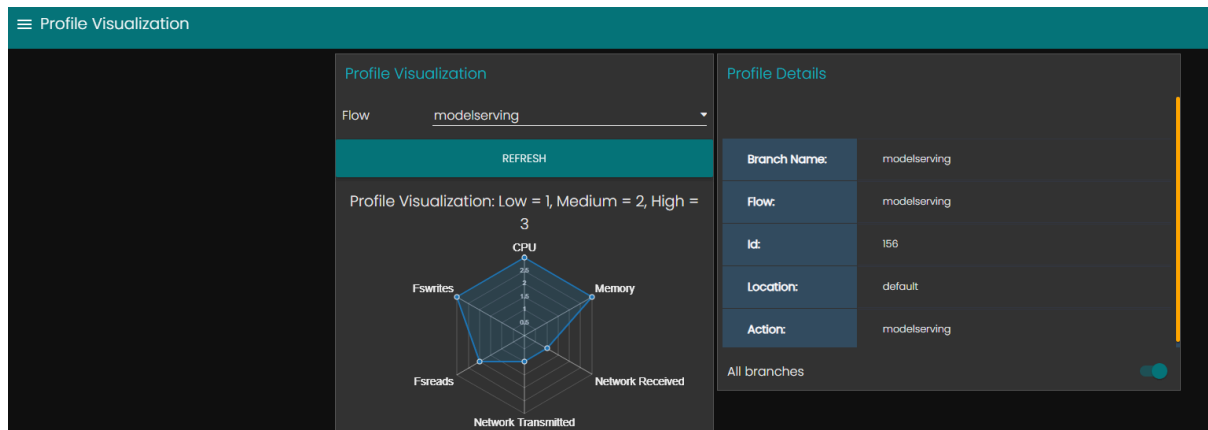
The username and the password will be encrypted in jenkins, where in the database will be stored only the id of the encrypted jenkins key.

View of Performance Pipeline Results in the Charts Dashboard

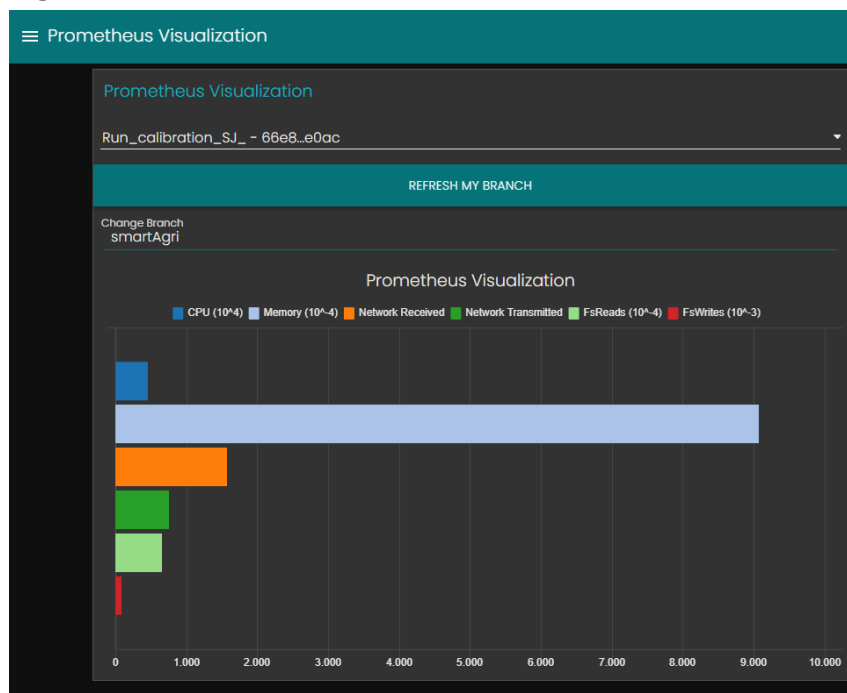


In the Charts section you can retrieve the performance measurement of the Performance test, it is also available in the burger menu the performance for the running flow on the PHYSICS FaaS Platforms under the section “Prometheus Visualization”. Following the execution of the performance pipeline, the user can view the results in their Charts tab of the dashboard.

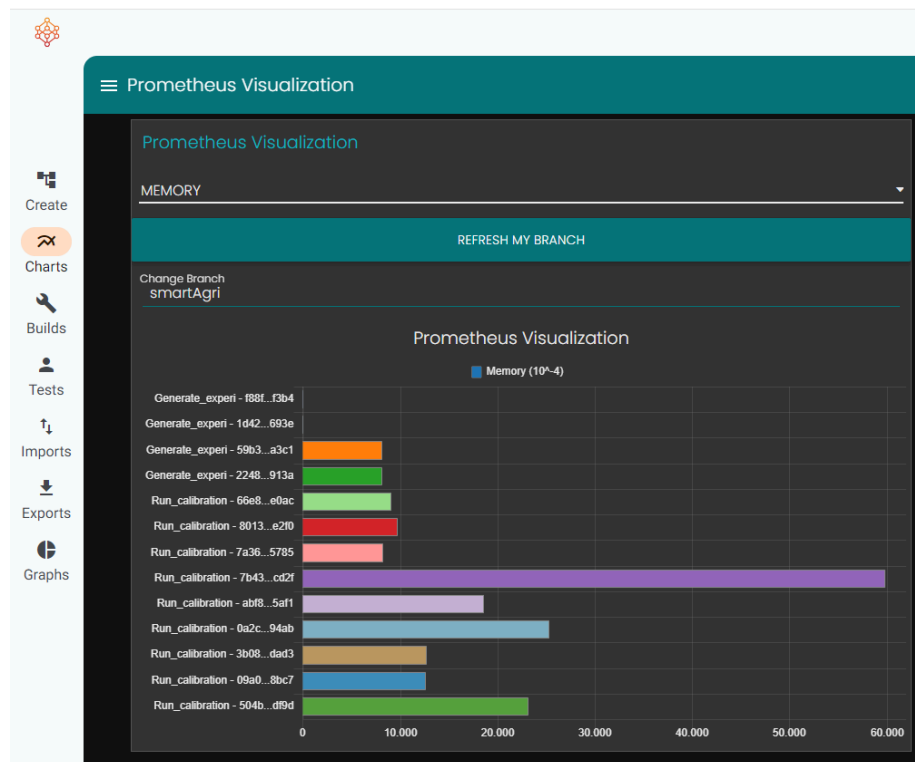
In the Profiles selection they can view the relative classification of a function regarding its usage of CPU, Memory, Network and Filesystem resources.



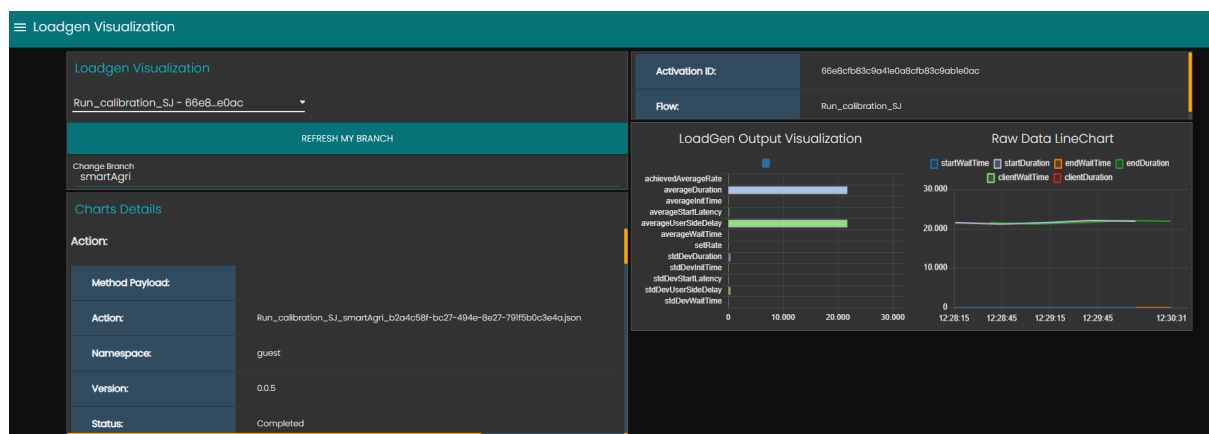
In the Prometheus Visualization, they can investigate the absolute usage of the resources by a given function.



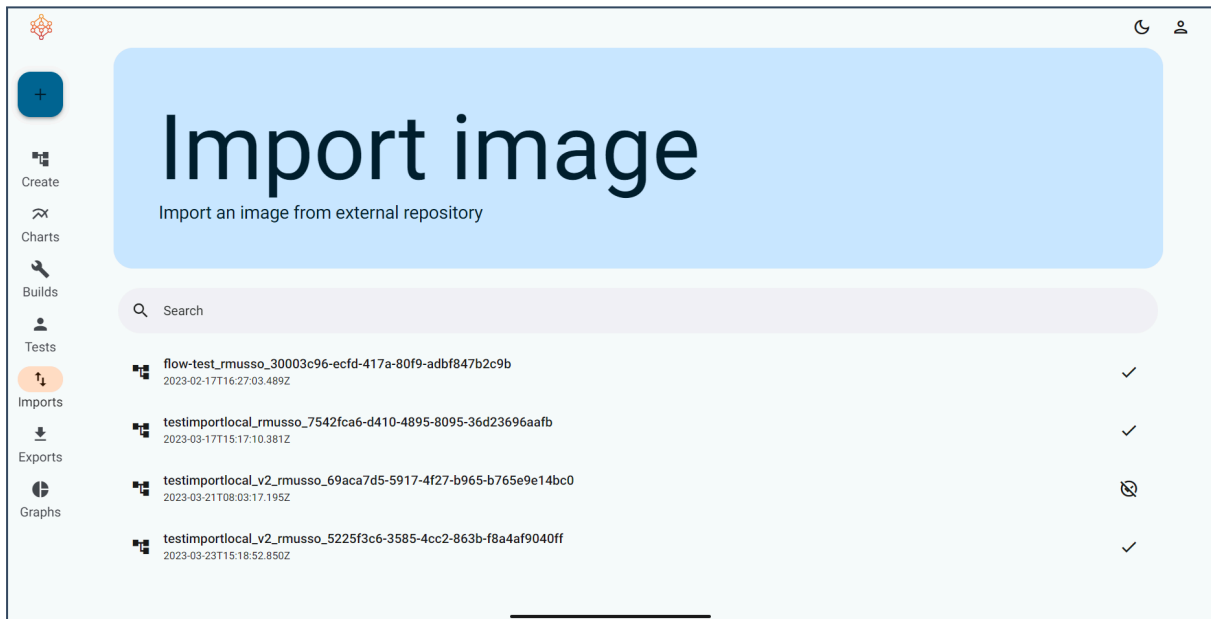
Furthermore, they can compare a single resource used (e.g. memory) from all their functions that have undergone the Performance test.



While in the Load Gen tab, they can view the top level information (average function duration, wait time, total user side delay) of the executions, as well as the experiment progression.



IMPORTS



Section dedicated to importing external images and where you can see the history and status of the imported images, you can import an external image from a remote repository, and define for it an **action name**, which can then be used to run the image in the test section.

The format of the action name following the convention

custom-action-name_<user-name>_uuid

Where custom-action-name is set by the user, user-name is the name of the logged-in user and uuid is generated automatically from the UI.

After the user requests for an import the DE calls the jenkins pipeline with the parameters set by the user. After the request, the user can monitor the state of the import directly from the page of import in the DE, where is report the history of all user imported image, with the action name associated, the date of the import request and the state represented by the following icon



The image is successfully imported and the action is active on openwhisk.



The image was successfully imported but a new version of the docker image with the same version was imported so the action on openwhisk is no longer available.



There was an error during the import, the image is not imported and the action was not created on openwhisk



The import is in queue

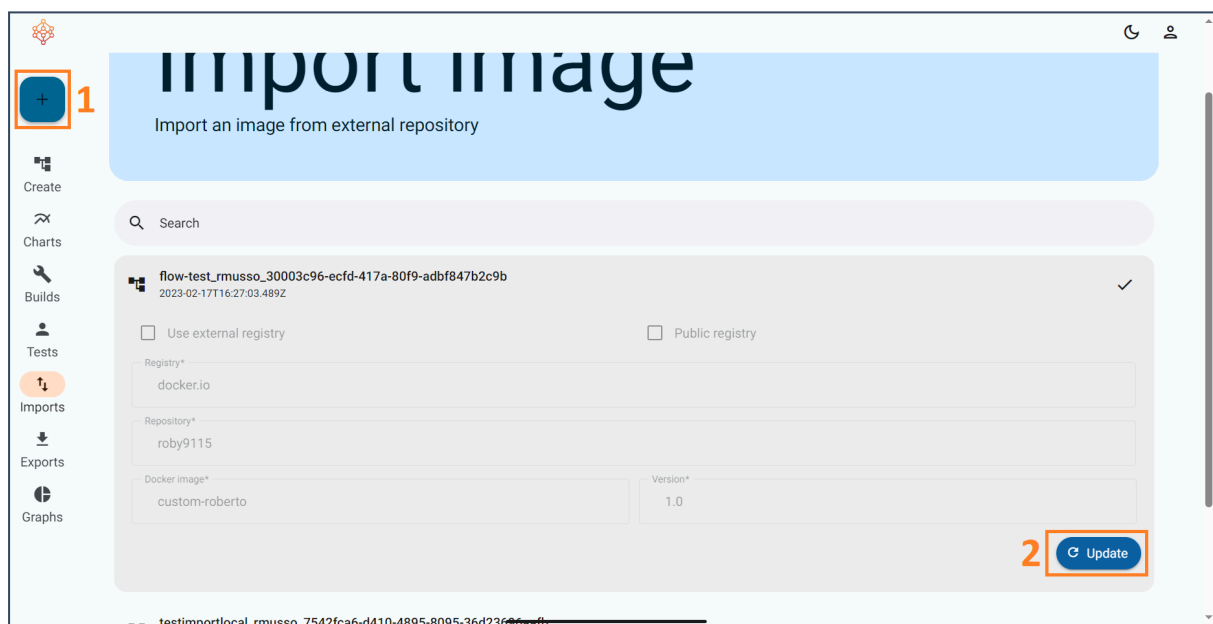


The import image is in progress

For the security purpose, the user credential to access the remote repository is stored encrypted on Jenkins with a user custom label, that it can be visible only from the user who created it and it can be reused for future import.

Start the import

The import can be a new import with the **plus** button (1) or it can be an upload of a previous one with the **Update** button (2).



The screenshot shows the 'Import image' interface. On the left is a sidebar with navigation options: Create, Charts, Builds, Tests, Imports (highlighted), Exports, and Graphs. The main area has a blue header 'Import image' with the subtitle 'Import an image from external repository'. Below this is a search bar. A table lists existing imports, including one with ID 'flow-test_rmusso_30003c96-ecfd-417a-80f9-adbf847b2c9b'. Below the table is a form to create a new import. The form has checkboxes for 'Use external registry' and 'Public registry'. It includes input fields for 'Registry*' (docker.io), 'Repository*' (robby9115), 'Docker image*' (custom-roberto), and 'Version*' (1.0). At the bottom right of the form is an 'Update' button, which is highlighted with an orange box and the number 2. A 'plus' button in the top left corner of the form area is highlighted with an orange box and the number 1.

The main difference between the two types is that in the update, you can only change the imported image data but not the action name associated, so in case you have a new version of the image you don't need to update all the reference to the created action.

Both types open the same form to request an import

Import image

Action name*
_rmusso_ab92a55f-800f-4ce7-9bb3-e88347558b4c

☒ Use external registry

☐ Public registry

Registry*

Repository*

Docker image*

Version*

☐ Use credential

Description*

[Close](#)
[Import](#)

Below is reported the field:

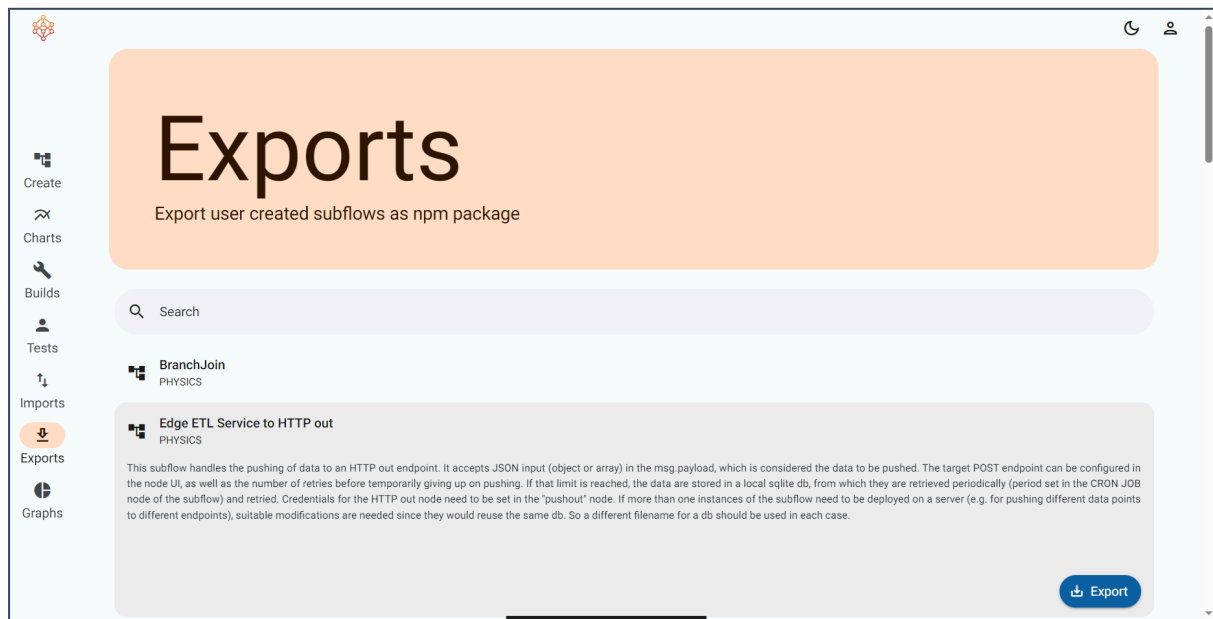
- **Action name:** it will be set as action on openwhisk if the upload successfully.
- **Use external registry:** in case the image to be uploaded is external from physics environments (es. DockerHub)
- **Public registry:** in case the image is stored in a public registry, if it is checked the section Credential will be hidden.
- **Registry:** the registry of the external image.
- **Repository:** the repository of the external image.
- **Docker image:** the name of the image
- **Version:** the version of the image to be imported
- **Use credential:** in case of unchecked the user can set the username and password with a description that jenkins can use to retrieve the image, in this case username and password is stored in jenkins and it was encrypted. After the first creation the credentials can be reused for the next import by checking the box Use credentials and selecting the credentials by the Description.

After the filling of the form the user can press the button Create to start the pipeline. After the start the Design Environment reloads the user imported images where the last one is present with the status of the pipeline.

The imported image is stored by the user, so the same *image:version* will not conflict with the imported by another user, but you cannot have a same *image:version* associated to two your actions; if you try to add a new action that points to an image you are already using, the system will ask you if you want proceed, if you accept, the previous action will be canceled and it will be flagged with the yellow icon.

EXPORTS

In this section you can export in a zip file your Node-RED subflow, to share it to other users or to upload on npm.



In the main section you will see the list of your subflow and if you expand the description are inserted in Node-RED.

With the **Export** button you can request the download in a zip file of the subflow. In the following chapter is described how to publish the subflow to npm.

Preparatory work

Before exporting the subflow we need to include information on the description of the node as shown in Figure 23, since a number of these fields are mandatory for the package creation. Finally, a set of files is created that is downloaded from the DE and contains all the main files of our node (README, etc.) as exported from the subflow definition, as well as the code files, that can be installed through a typical npm install command.

Edit subflow template: node-red-contrib-owmonitor

Cancel
Done

Module Properties

Module

node-red-contrib-owmonitor

Node Type

8262c8e75cfbabdd

Version

0.0.1

Description

This is a subflow node in order to monitor the latest per

License

▼ Apache-2.0

Author

George Kousiours <gkousiou@hua.gr>

Keywords

openwhisk, performance, monitor, sliding window

Final export process to npm and Node-RED repos

In order to publish in the npm repository, from the previous folder generated by the DE we can push the node on npm, following the inclusion of the desired npm account:

```
npm adduser
```

Then we can publish the node folder contents on npm through the:

```
npm publish
```

Search packages

Search

node-red-contrib-owmonitor

0.0.2 • Public • Published 5 hours ago

Readme

Explore

0 Dependencies

0 Dependents

2 Versions

Settings

node-red-contrib-owmonitor

This is a subflow node in order to monitor the latest performance (sliding window) of Openwhisk actions. The flow pings periodically the target Openwhisk installation in order to retrieve the last executed actions and extract statistics from their execution.

Install

> npm i node-red-contrib-owmonitor

Repository

github.com/gkousiouris/node-red-contrib-owmonitor

Homepage

github.com/gkousiouris/node-red-contrib-owmonitor

Weekly Downloads

37

Version

0.0.2

License

Apache-2.0

Unpacked Size

16 kB

Total Files

5

Issues

0

Pull Requests

0

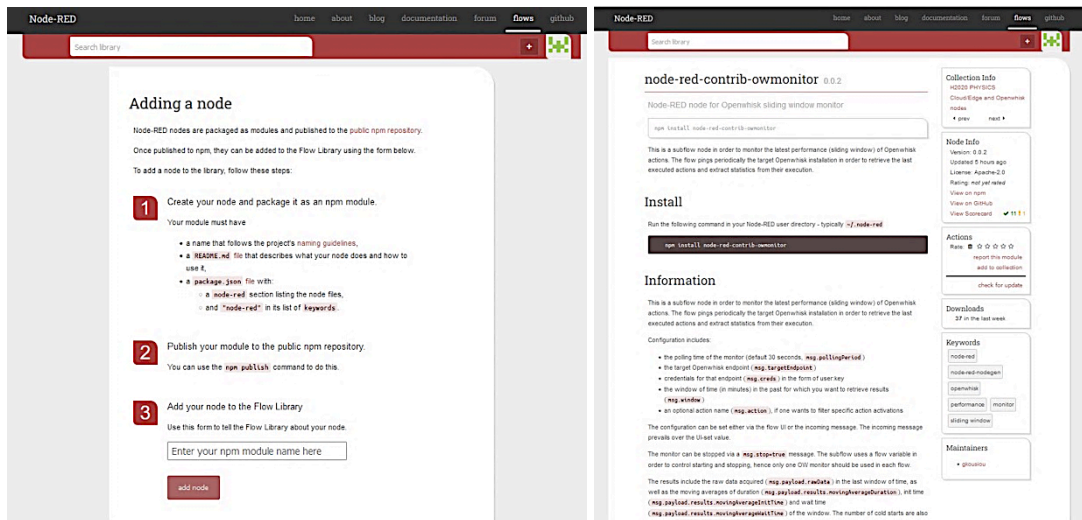
Information

This is a subflow node in order to monitor the latest performance (sliding window) of Openwhisk actions. The flow pings periodically the target Openwhisk installation in order to retrieve the last executed actions and extract statistics from their execution.

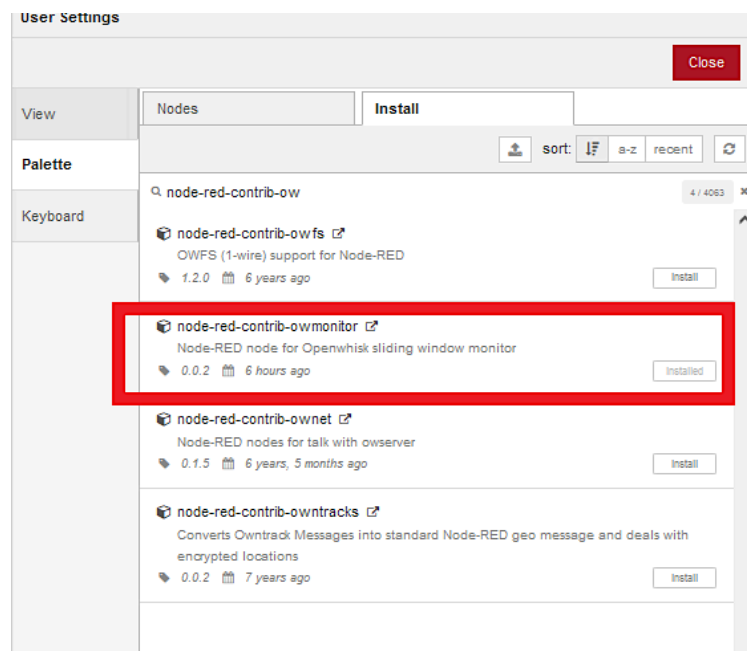
Configuration includes:

- the polling time of the monitor (default 30 seconds, `msg.pollingPeriod`)

For adding in the Node-RED node repository, in order for it to be also available for installation directly (including all its dependencies) from the Node-RED main palette environment, we also need to register it through the Node-RED site⁴ and add the npm module name. This declares the node in the Node-RED community repository.



Following that step, the contribution is now packaged as a Node-RED node and can be found directly from the built-in palette management functionality of Node-RED.

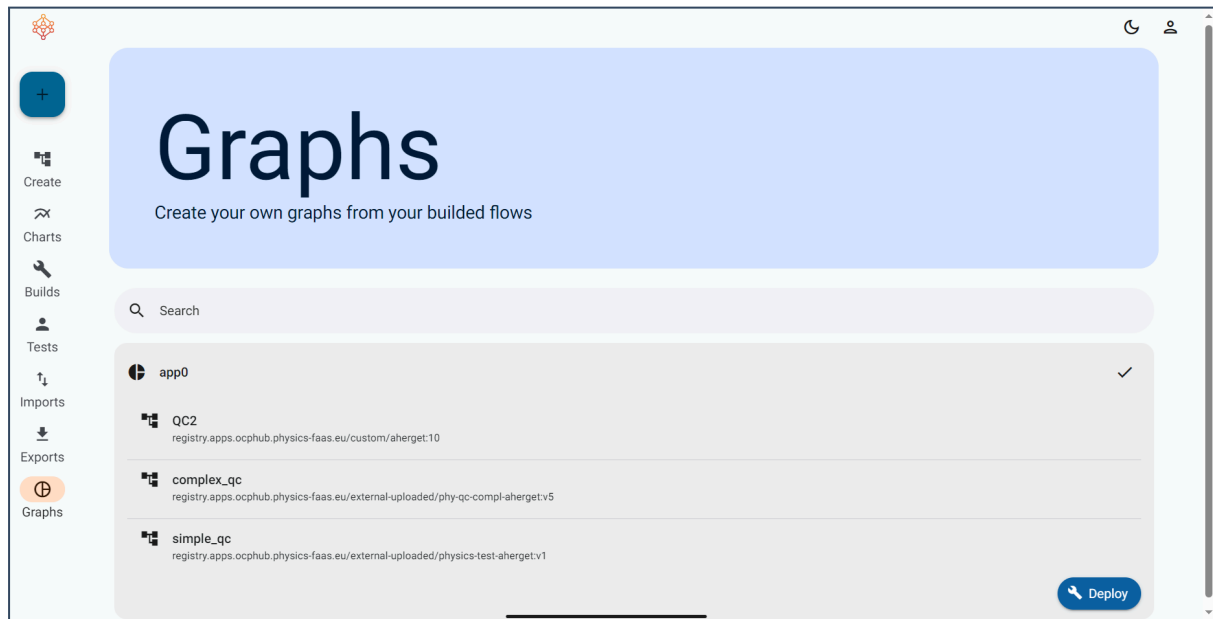


⁴ Node addition in Node-RED repository: <https://flows.nodered.org/add/node>

In our case it was also added in a specific collection that is organized by PHYSICS specifically for nodes⁵ and is different than the one for the subflows⁶.

This way of publishing (NPM and Node-RED node) also allows us to keep track of usages, comments etc. for our node. Thus, it is a very good way for reusability of the results as well as statistics of usage of a node artifact. It can also be linked to a GitHub repository, in which the testers or users can interact in the form of issues etc.

APPLICATION GRAPHS



In the previous sections, the functions created are deployed as test versions for a given flow. Thus the developer may create multiple versions and progressively add the functionality needed. The deployment of the test functions in this case takes place in the default test cluster. However, once the implementation of the functions is finalized and tested, the developer can proceed with grouping them in relevant function groups (application graphs), that are managed together by the PHYSICS platform. By doing so, the developer may exploit the advanced features of the latter, such as optimized placement across multiple clusters, annotation enrichment and management.

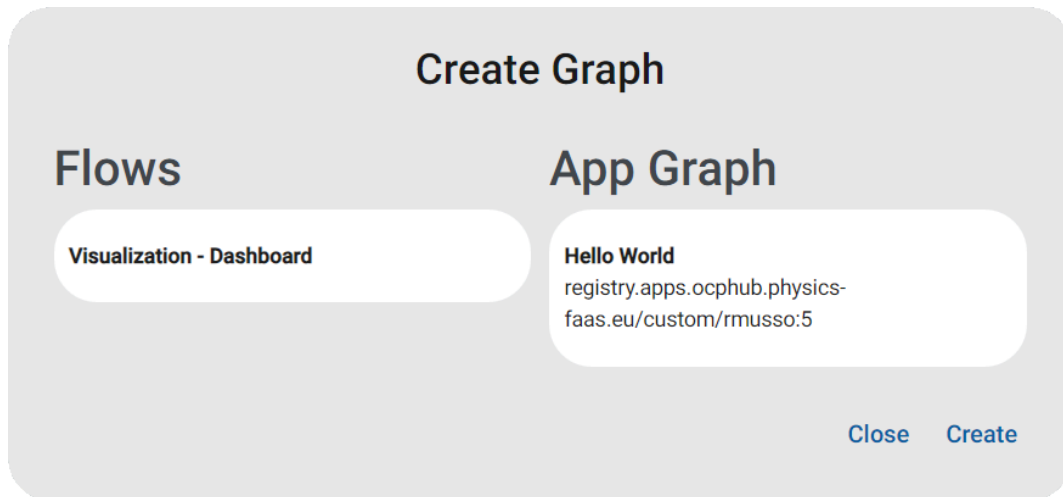
.

⁵ PHYSICS node collection on Node-RED repository, available at: <https://flows.nodered.org/collection/9C3h7Hnru943>

⁶ PHYSICS Flows collection: <https://flows.nodered.org/collection/HXSkA2JJLcGA>

Create a new Application Graph

To create a new graph you can use the **plus** button on the rail menu that will open a modal for the creation of the graph.

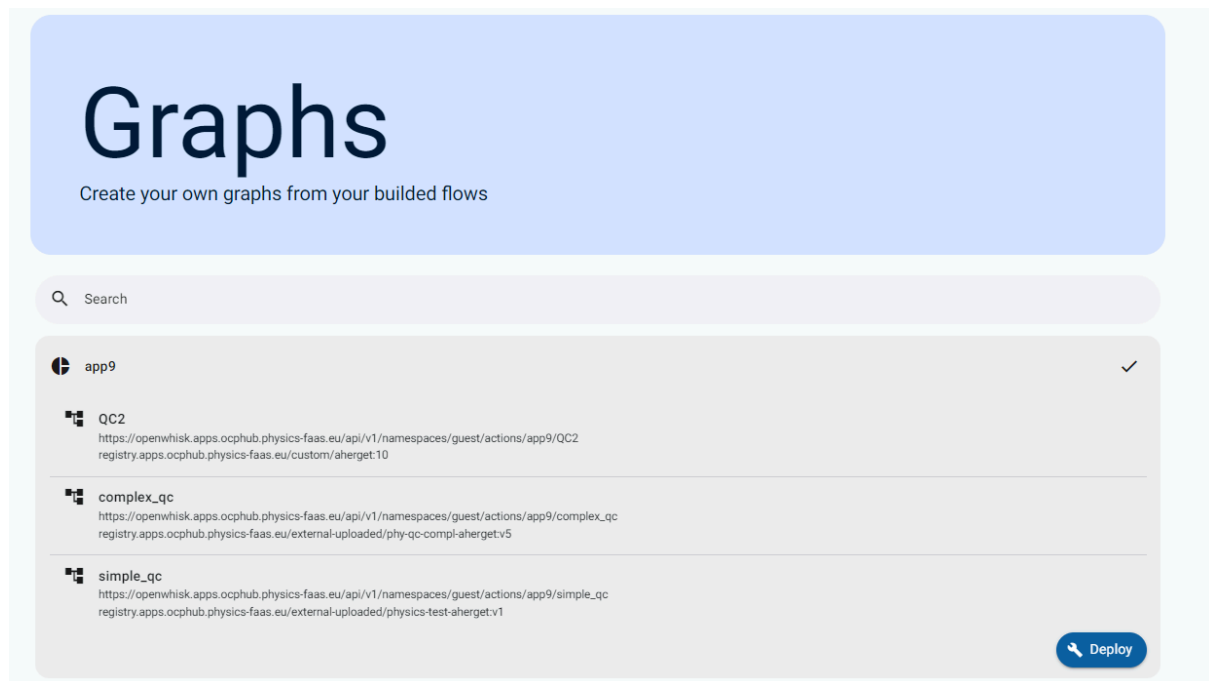


Where you can grab your local flow from the left to the right to create your graph. When you are complete you can request the creation with the button **Create**.

In the creation it will use the the image indicated under the Flow name, if the flow haven't an image, the system before send the request to the semantic extractor, made a build of the flow; in this case you will see, in the main section, the **Graph Draft** button that report all your building flow started for the graph creation.

Once created, the app graph is passed through the semantic processing and submitted to the platform services of PHYSICS for the deployment of the formal version of the functions. Created functions are deployed based on the flow name (without the version id included in a typical test run), compensated by the application id (as package name). In the example below, one of the finalized deployed functions endpoint is as follows:
<https://openwhisk.apps.ocphub.physics-faas.eu/api/v1/namespaces/guest/actions/app9/QC2>

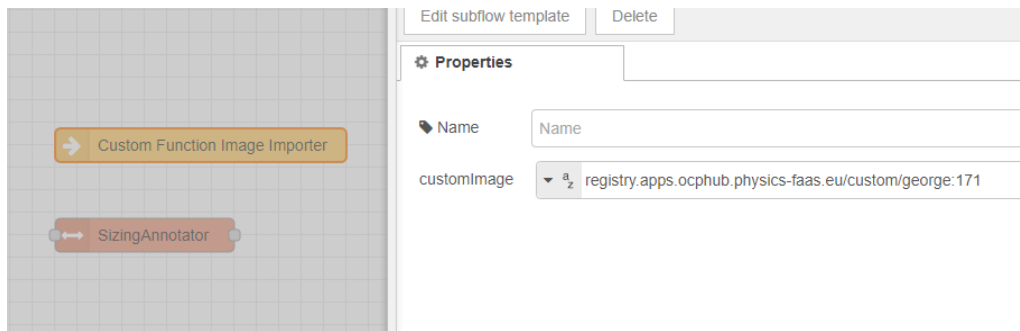
For the final deployment, the user needs also to push the Deploy button in the Graphs panel.



Inclusion of an imported image to an Application Graph

In order to include a manually imported image to an application graph, the developer needs to go through a stage of declaring it to the framework. After importing the image from the DE functionality, the main intention is to use it in the context of a PHYSICS application, thus to include it in a collection of flows and functions to be deployed at the typical PHYSICS production environment.

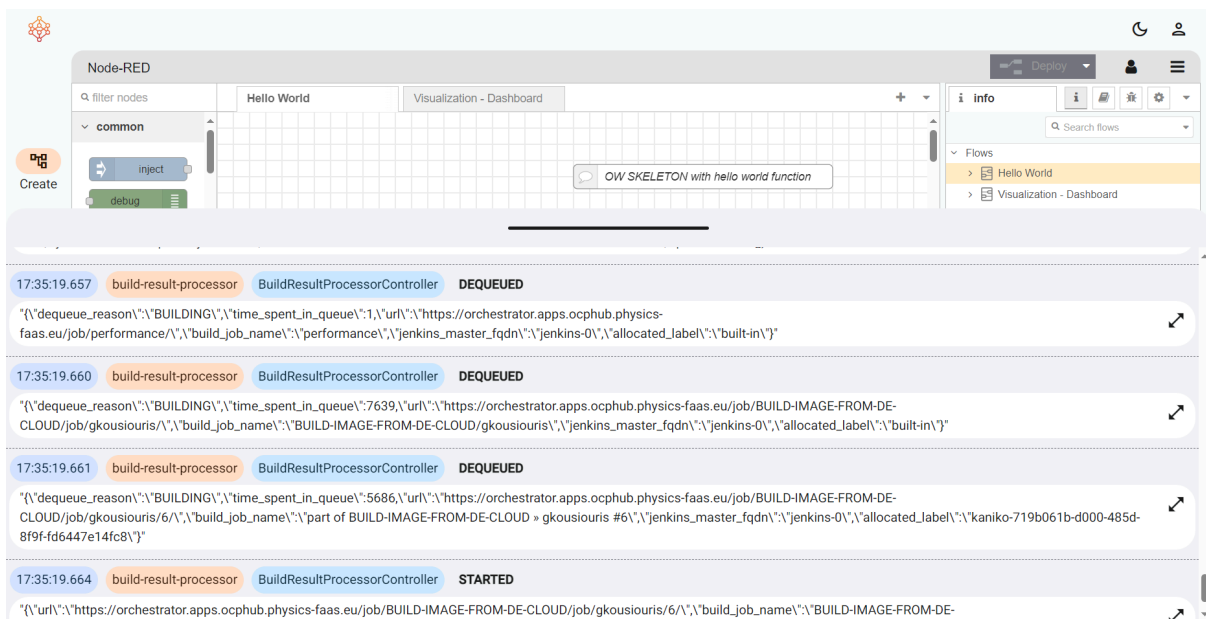
However, given that this function has not followed the typical stages of a PHYSICS function, a relevant declaration process needs to be followed, including creating a semantic annotation for this custom image used. In order to support this, the PHYSICS DE provides a relevant semantic node, the “Custom Function Image Importer”. The user needs to create a new flow in the DE, in which they will drag and drop that node and populate it with the name of the custom image in the relevant field (Figure 45). The name given to this flow will also be the name of the resulting function. In this flow they can also include other needed annotations from the PHYSICS available ones (e.g. sizing, locality etc.).



Finally, although there is no relevant function logic inside this flow, they need to build it through the DE. The reason for this is that only built flows are allowed to be included in an app graph in the next stage. So the system needs to have this build documented. Once the flow is built it can then be added to a PHYSICS App like any other available flow.

Log

With a click on the bar on bottom of the main, you can access your backend log.



With the bar you can resize if you grab it, or close if you click on it.

In the log is reported the time of the creation, the application and the function that made it and a message. If the log have a payload it is reported under the message and it is expandable with the arrow icon and copyable.

Provided pattern flows by the PHYSICS project may be periodically updated or extended⁷. These updates may be a result of newly needed features, specific requests or debugging and improved parameterization. In order to import the updated versions in the environment, if the pattern is packaged as a node, this process is done through the palette management of the environment. However, if the pattern is packaged as a subflow or typical flow, then different variations of import may be performed.

wsk action create dockeraction --docker gkousiou/noderedaction

Copy JSON

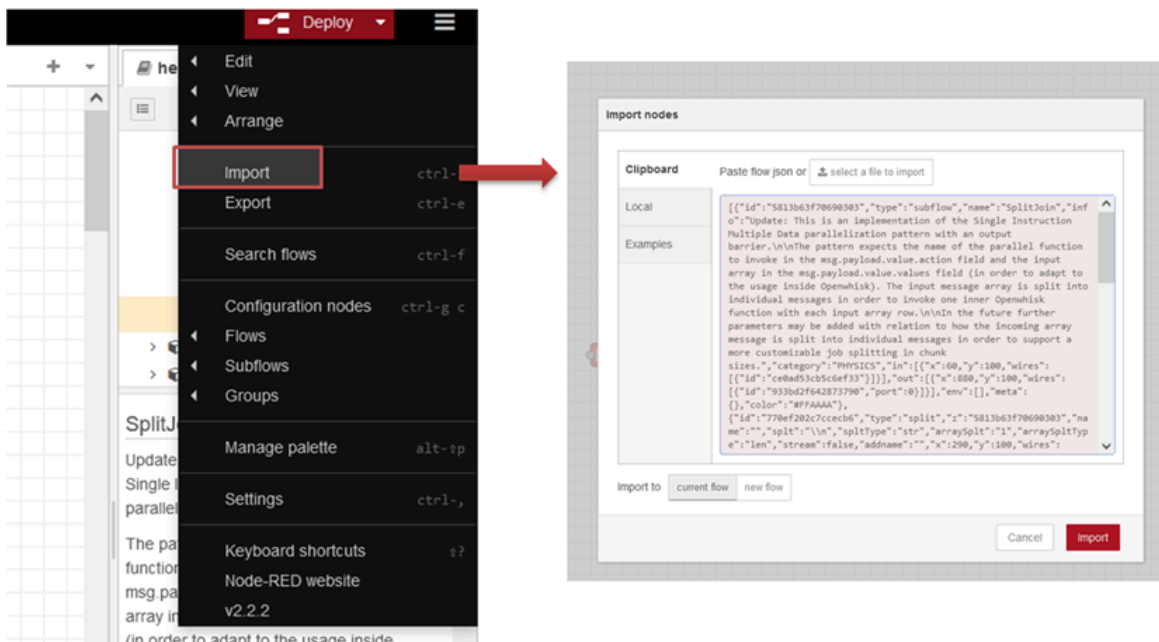
Flow 1

PollTOPush

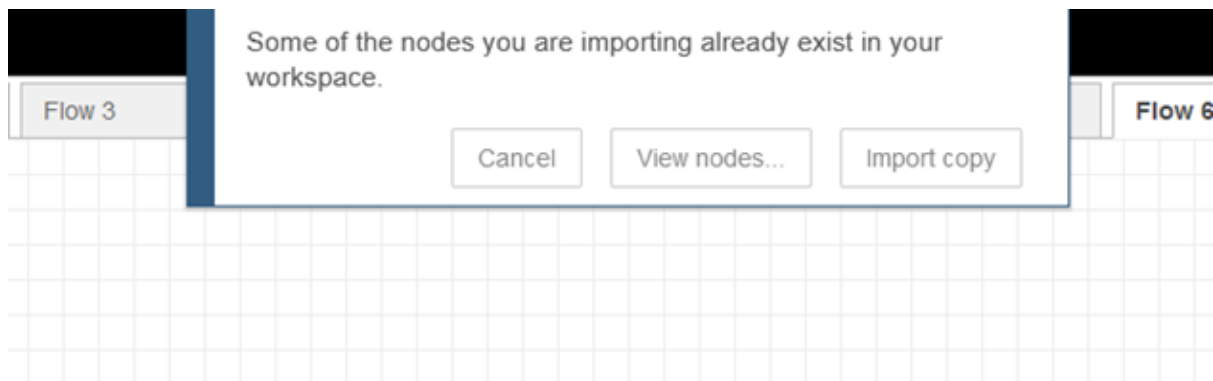
PHYSICS L

Note: some third-party nodes may appear with blank styling, and not as they appear in the Node-RED Editor

⁷ PHYSICS Patterns Collection, Available at: <https://flows.nodered.org/collection/HXSkA2IJLcGA>



If a previous version of the flows exists in the environment (it should), the user will get a warning message for importing existing nodes.



By selecting the “View nodes” option, the user will get a relevant screen about conflicting nodes . If they do not select the grey subflow in the Subflows section of import, only the node reference will get imported, so the imported flow will use the locally existing (previous) version of the subflow.

Import nodes

Some of the nodes you are importing already exist in your workspace.
Select which nodes to import and whether to replace the existing nodes, or to import a copy of them.

Subflows

☐ >

SplitJoin

☐ replace

Nodes

☒

SplitJoin

Configuration nodes

☐

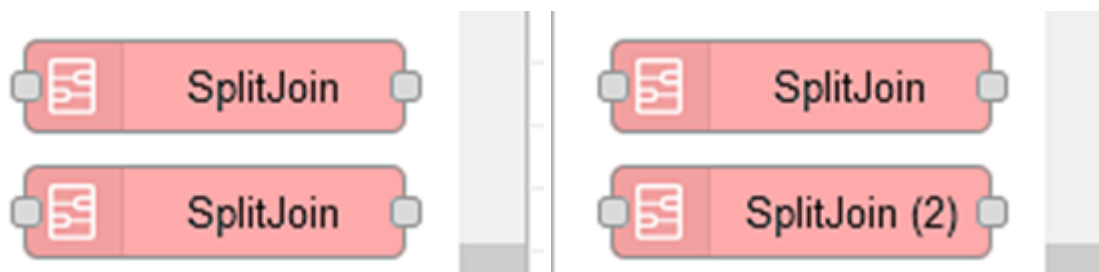
MYOPENWHISK

☐ replace

Cancel

Import selected

If we select the SUBFLOW and REPLACE check box, the subflow is replaced across all flows and the palette has only one instance of the subflow. This is a **complete update** of the subflow across the entire environment (this and all other existing flows that may use the subflow). If we don't check "replace", a new version is included in the palette (but without differentiation in the name) and the subflows local versions are not replaced in the previously existing flows. This is useful in case the developer has created some modifications in the subflow local versions and wants to maintain them, while importing the new version as well. For any new flow we can select whatever of the two versions from the palette, although there is no differentiation in the name appearing in the palette. This is in essence versioning of the subflow.



Given that this creates confusion afterwards, if we want the versioning option, we should use the direct Import Copy option from the initial warning. This would result in maintaining the current version of the subflow in existing flows and the new version in

the imported flow, while the new version will be clearly marked in the palette with a new version number (2).

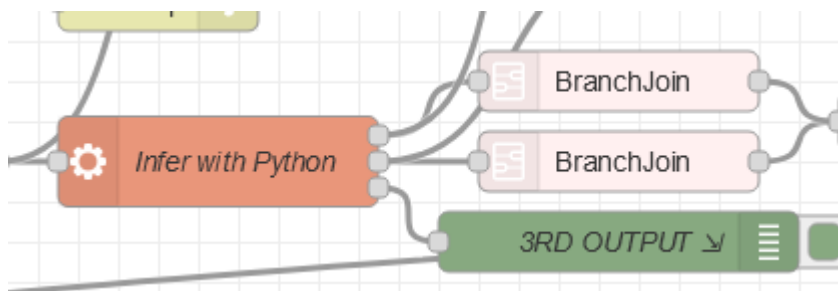
NODE-RED RELATED COMMON ERRORS

Subflow in Subflow

When there is a subflow inside another subflow in an imported flow, we need to have a copy of the inner subflow in the main flow, since otherwise it is not recognized as a type (the inner subflow definition is not exported in the flows file). So we either need to ensure we have all the subflows in the typical node-red environment or the person who shares the subflow needs to have an idle such inner subflow node in the shared flow for definition purposes.

Memory considerations of invoked action

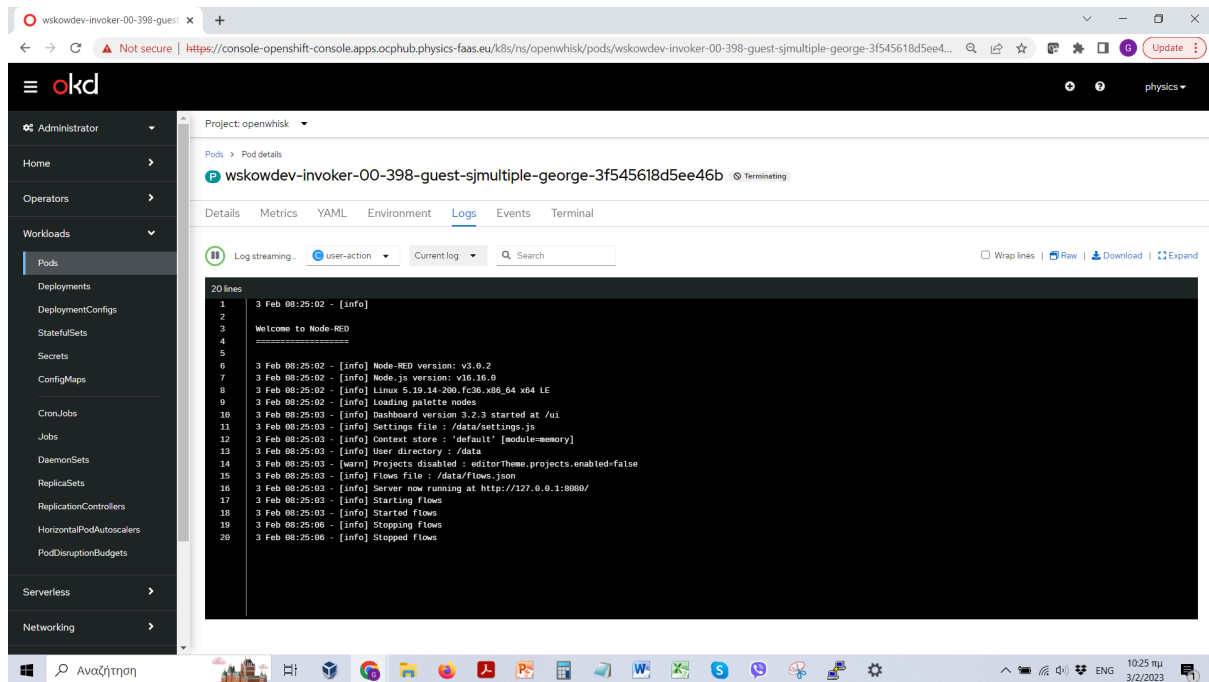
When using the exec node to launch a shell script, make sure to always print the 3rd output of the exec node in Node-RED. In some cases, when the child process launched fails, this is the only place where some logging information will be available. For example, if the process consumes a lot of memory, ending up in being killed by K8S due to memory size requirements, the 3rd output will indicate the reception of a 'SIGKILL' signal. This information is not propagated neither by K8S nor by Openwhisk. If that signal appears in the logs of the executed function, it means that the assigned memory is too low. By increasing the amount of memory assigned to the action, the problem is solved.



Size of output message

If the output to be returned to Openwhisk is significant (e.g. logs from two cascading functions) then the return from the flow fails without obvious reason. No warning or event is issued at the Openwhisk or K8s level. You can detect that from the fact that right before the final response node, and although everything seems ok up to then, your Node-RED function runtime will stop (Stopping nodes and flows in Node-RED function logs as shown below) without apparent reason and the function container will

disappear. In a successful execution the container should remain active (at the Started flows or any follow-up printed message).



Subflow id not iterable

Any subflow needs to have at least one node inside it, even if it does not do anything. If there is an error message such as “TypeError: groupedNodes[subflowId] is not iterable”, this means that there is a subflow with no nodes in it. One can drag and drop one comment node in the relevant subflow in order to resolve this. Such nodes may exist especially in the case of semantic nodes, if the PHYSICS user needs to somehow extend the list of key/value annotations that are passed to the semantic layer but without the need for a function logic inside the subflow.

Multiple /run Endpoints in Node-RED

Given that the main Openwhisk function interface dictates the need to have a POST /run interface, all the created flows in the environment will need to include one. The problem is that there can be only one REST endpoint with the same name in the Node-RED server on which the DE editor is based. Thus if someone tries to test a new flow locally (inside the Node-RED environment), the test call will actually be sent to the first flow that has been created with this endpoint. Hence it is advised to change the name of this endpoint for local testing needs (e.g. /run2) and then switch it back to /run prior to building the flow.

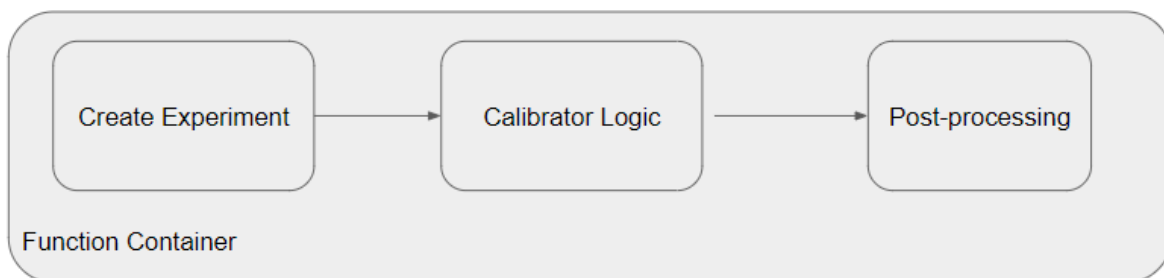
PARALLELIZATION STRATEGIES

The PHYSICS environment provides a number of ways through which the developer can implement parallelization strategies. The selection of a strategy depends on the specifics of a function, taking into account among other things the typical function execution time (to avoid timeouts), the computational characteristics of a function (cpu intensive versus i/o wait intensive) etc.

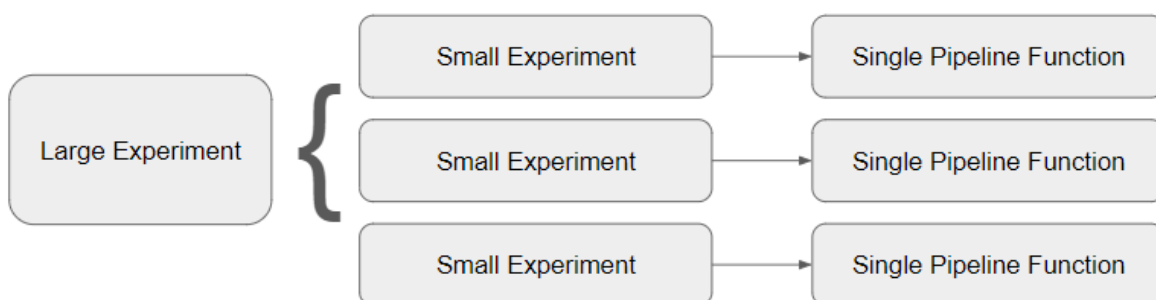
This guide aims to indicate common means through which such a parallelization may be performed, exploiting the FaaS concepts as well as the tools provided by PHYSICS. As an example, the PHYSICS Smart Agriculture use case is presented, that includes 3 different parts. An experiment creation stage (not parallelizable) as well as a (parallelizable) calibration stage that takes as inputs the combinations of the experiment creation in order to simulate the behavior of the agricultural system. Following, a number of possibilities are portrayed.

Single Function Pipeline

In this case, all the stages are included in a single Node-RED flow, executed as a function. The parallelization in this case relates to multiple single functions that can be invoked, i.e. splitting externally one large experiment into multiple smaller ones and triggering one function for each.

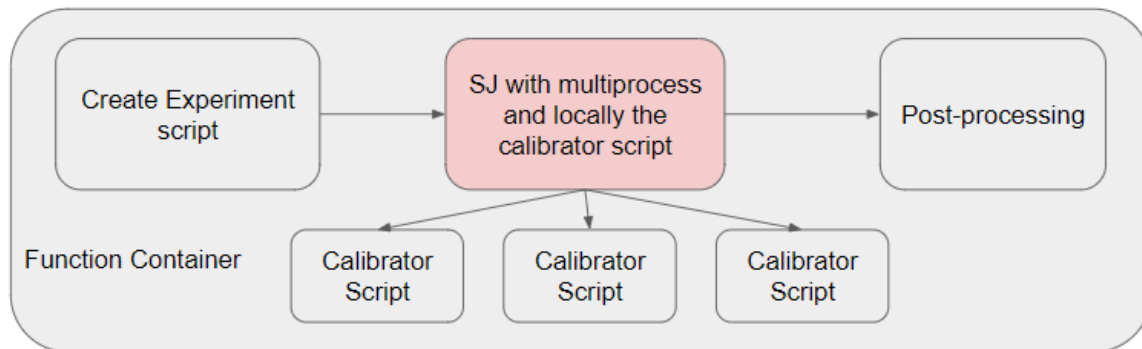


This way exploits the inherent parallelism of FaaS but also introduces a risk of timeouts since all the operations are included in one single function.



Single Function Multi-process Pipeline (Intra-container parallelism)

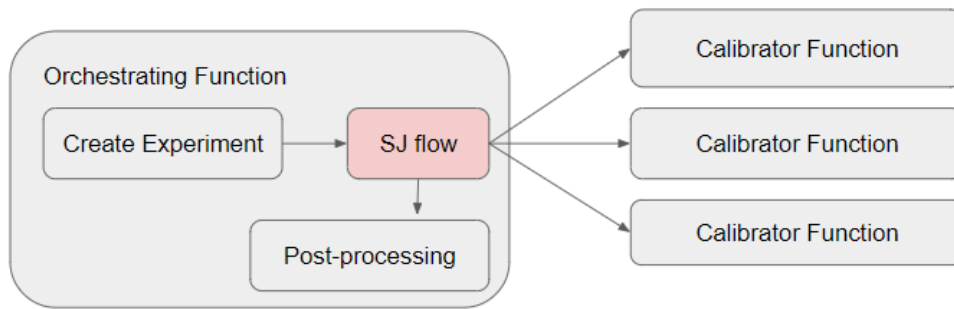
In this case, all the stages are included in a single Node-RED flow, executed as a function but the flow includes the SplitJoin Multiple pattern configured with the local multiprocessing option.



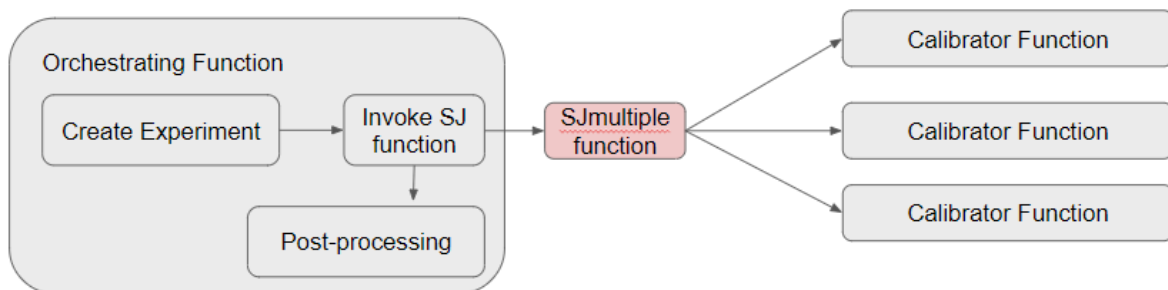
The parallelization in this case extends the previous scope and includes multiple spawned processes within the same container. Thus it aims to replicate the behavior of OpenMP-type of parallelism. Each calibrator script receives a subset of the input experiment and undertakes its completion. The SJmultiple joins the answers on completion of all Calibrators and proceeds to the next stage. This is especially useful when the main calibration logic is not only cpu-intensive but also has blocking intervals. If this logic is only cpu-intensive, then it is possible that this way will not be very beneficial due to task switching and cache miss phenomena.

Orchestrating Function-Worker Function (Inter-container parallelism)

A third option is a combination of an Orchestrating Function that undertakes the main workflow and a Worker Function, multiple instances of which are spawned in order to simulate faster the experiment combinations. This enables an MPI-style parallelism, in which containers can be spawned across the available cluster. In this case the SJ node needs to be configured with the FaaS option.

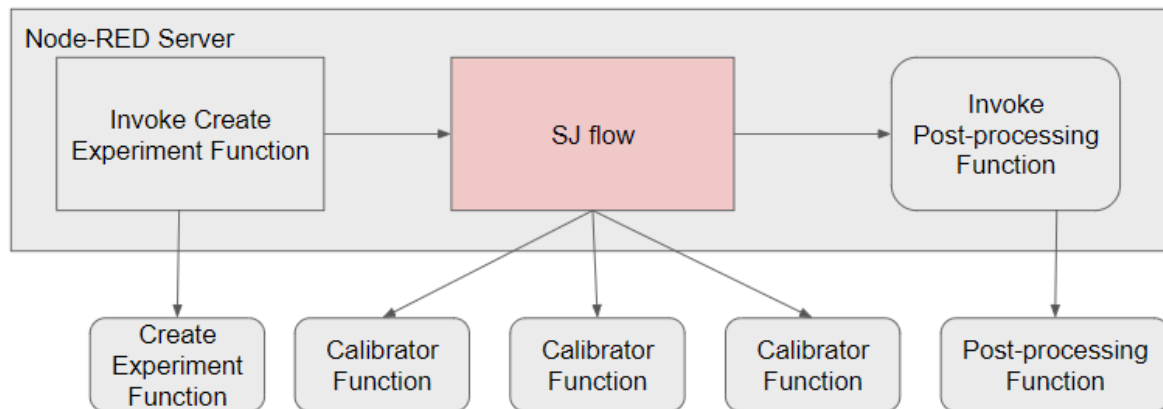


Another similar option in this case is to have a generic SJmultiple function. The parameterization and abstract behaviour of the PHYSICS SJ pattern enables the creation of such an abstract function. This means that this function may be reused across different use cases and thus reduce the complexity of the flows.



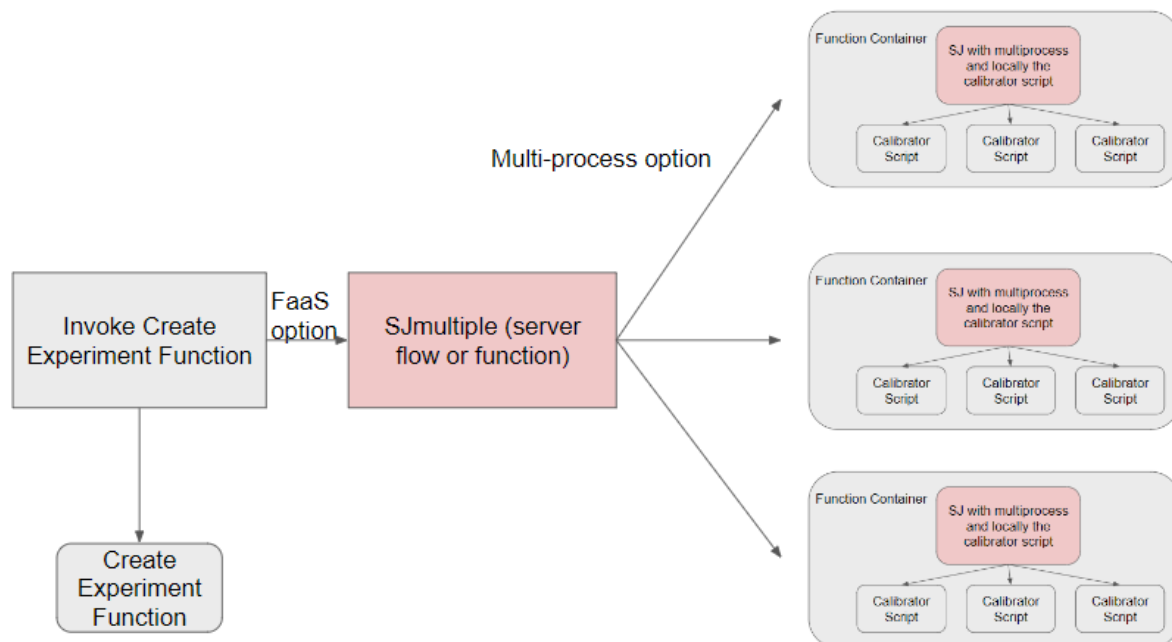
Orchestrating Server-Function Implementations (Inter-container parallelism)

The next strategy involves a combination between a server-based orchestrator and a number of functions that are used to implement the main logic. The main benefit of this approach is that we are not constrained by the SJ function timeouts in case of large experiments. The only limitation from a timeout point of view is the one for the base calibrator function. But this is easily configured through reducing the number of inputs assigned to each worker through the split size parameter of the SJ pattern. On the other hand, one needs to have a server running continuously for the orchestrator. In the case of the PHYSICS Smart Agriculture use case, this server is already running at the edge (Raspberry Pi) in order to collect the data. So it is neutral from a cost point of view to add a lightweight orchestrator flow for managing cloud-based simulations.



Hybrid Parallelism (Inter and Intra-container parallelism)

The last case is a combination of the above strategies. In that sense, one can select which parts are server-based and which function-based, but the main difference is also the combination of parallelisms. So one can use a first level of SJ (with the FaaS option) to parallelize between different containers and then a second level of the SJ (with the multiprocessing option) for multiple processes inside each container.



DEBUGGING STRATEGIES

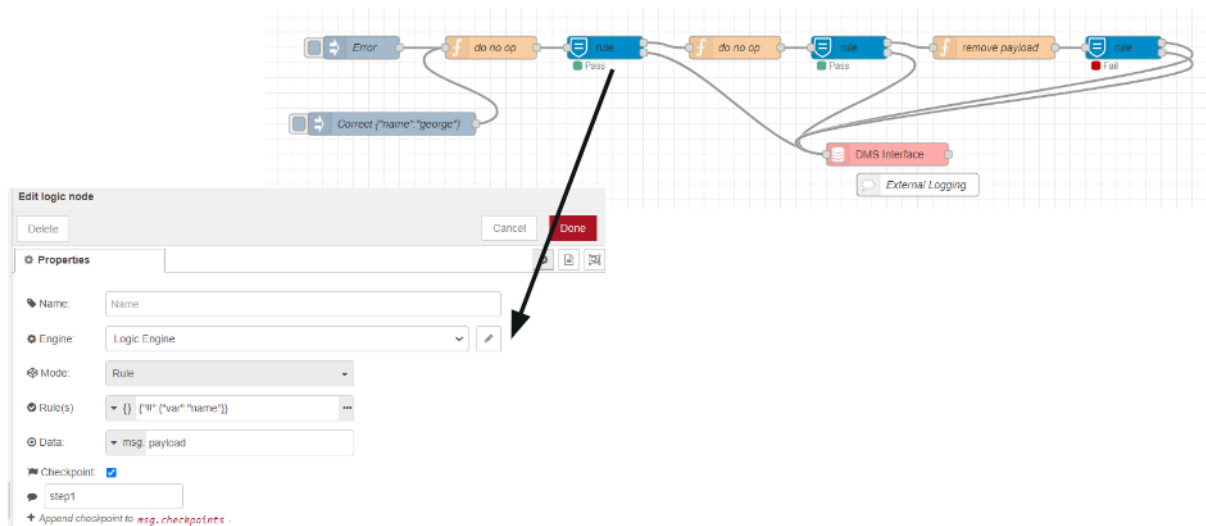


A number of possibilities exist for the developer to debug their flows in the PHYSICS environment. In the previous sections a number of features were shown, including error catching in the main skeleton flow, log and test details in the according tabs etc.

In order to support a more fine grained and detailed analysis of the flow execution, PHYSICS provides two helper flows in order to capture both functional and non functional issues.

Functional Flow Testing

The functional flow testing is supported by the json-logic node, created by the PHYSICS project⁸. Through this, the developer may insert functional checkpoints within a flow and perform a very fine grained root cause analysis for an error. An example appears in the following flow.



In this case we expect that after the first function, the message should include a variable “name” in the message payload. Thus if the message includes this, the test is passed, if not it has failed and the results are forwarded to an external logging service. Similarly to this, the remaining checkpoints are checked if the previous ones are passed and a detailed report is produced.

⁸ <https://flows.nodered.org/node/node-red-contrib-json-logic/in/9C3h7Hnru943>

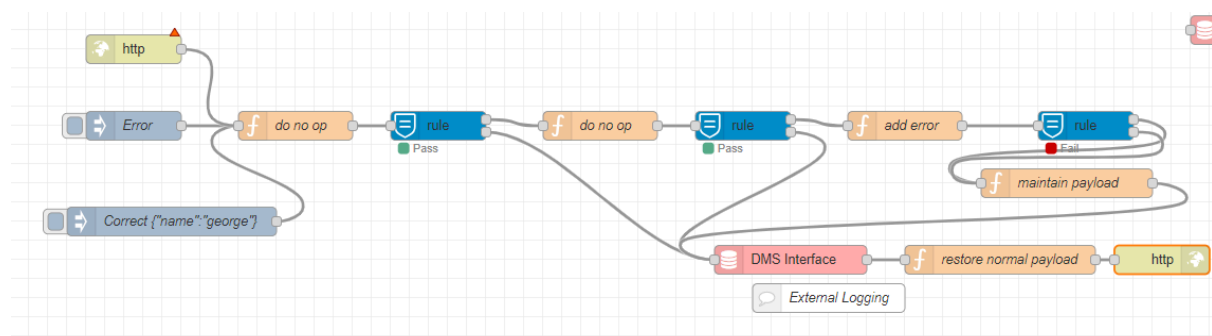


```

▼ checkpoints: array[3]
  ▼ 0: object
    id: "59fd820bf366df46"
    engine: "f2735fd26858fbb5"
    mode: "rule"
    ▶ rule: object
      data: "msg.payload"
      result: true
      timestamp: "Fri Nov 03 2023 11:04:48 GMT+0000 (Coordinated Universal Time)"
      message: "step1"
  ▼ 1: object
    id: "b955a47f1c9efe63"
    engine: "f2735fd26858fbb5"
    mode: "rule"
    ▶ rule: object
      data: "msg.payload"
      result: true
      timestamp: "Fri Nov 03 2023 11:04:48 GMT+0000 (Coordinated Universal Time)"
      message: "step2"
  ▼ 2: object
    id: "312f6ec49b67d61f"
    engine: "f2735fd26858fbb5"
    mode: "rule"
    ▶ rule: object
      data: "msg.payload"
      result: false
      timestamp: "Fri Nov 03 2023 11:04:48 GMT+0000 (Coordinated Universal Time)"
      message: "final"

```

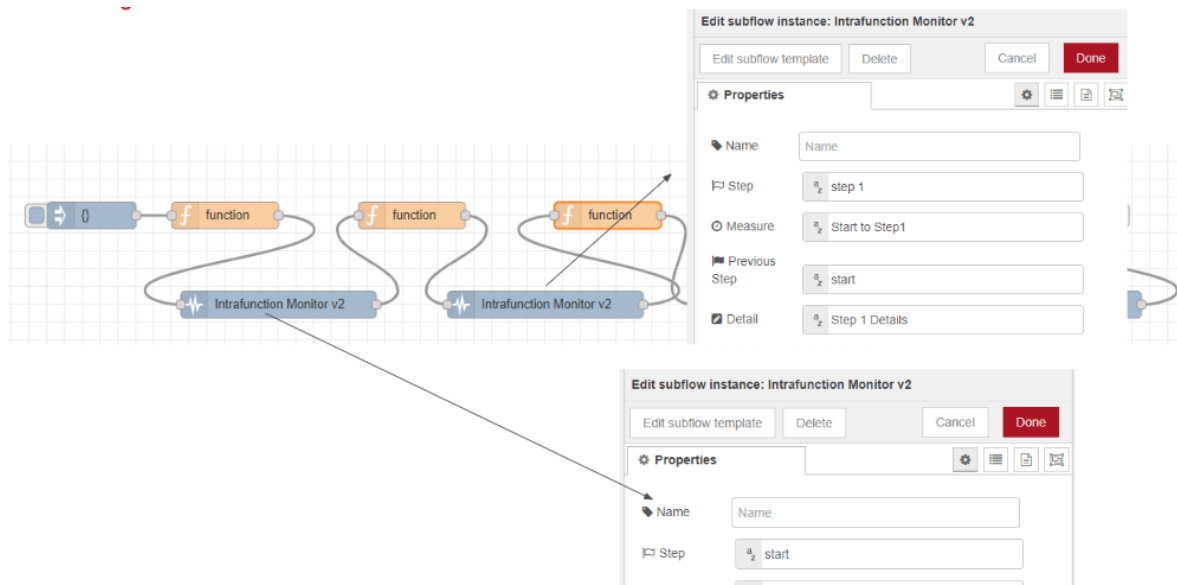
If this needs to be applied in an Openwhisk skeleton function template, we need to wait for the external logging process to finish before returning, in order to ensure proper logging. But in this case we need to preserve whatever `msg.payload` is before the usage of the external logging client, since the latter may overwrite the `msg.payload` that is intended to be delivered to the invoker. Thus the aforementioned flow may be transformed as follows.



Non-Functional Flow Testing

For the non-functional (performance) analysis, the intra-function monitor node can be used in order to insert performance measurement checkpoints within a flow. In this case

the developer can annotate differences between the needed checkpoints and receive in the end a detailed performance report.



Similarly to the Functional Testing, the logging can then be directed to an external logging service and included in the Openwhisk function template.

```
11/3/2023, 3:08:13 PM node: d5f3b2d3260fae59
msg: Object
  object
    _msgid: "b8bdba985f383d18"
    payload: object
      msgid: "b8bdba985f383d18"
      timestamp: 4601875804.88086
      performance: array[3]
        0: object
          name: "Start to Step1"
          entryType: "measure"
          startTime: 4601875796.432357
          duration: 2.859792709350586
          detail: "Step 1 Details"
        1: object
          name: "Step1 to Step2"
          entryType: "measure"
          startTime: 4601875799.29215
          duration: 3.605325698852539
          detail: "Step 2 Details"
        2: object
          name: "Start to End"
          entryType: "measure"
          startTime: 4601875796.432357
          duration: 8.483386993408203
          detail: "Total Performance"
```