evolve

Leading the Big Data Revolution

Cookbook

Evolve Frontend Notepad Cookbook V 0.3

This document presents the Evolve platform Notebook user guide and documentation for the Frontend UI and the Zeppelin interpreter reference.

Authors	Michail Flouris, John Sfakianakis, Stelios Louloudakis (Sunlight.io)
Description	User guide and documentation for the Evolve Notebook Frontend UI and the Zeppelin interpreter reference.
Date	June 2020
Revisions	0.3 (June 2020) - Added guidelines for running stages and sensors 0.2 (January 2020) - Added information for all commands and examples 0.1 (December 2020) - Initial version with workflow command an

Contents

evolve
Leading the Big Data Revolution

1.

Access & Instalation of Zeppelin Evolve Frontend

1. Access & Installation of Zeppelin Evolve Frontend

The EVOLVE dashboard (KARVDASH) is now available for running Zeppelin through a container.

The EVOLVE dashboard (provided by FORTH) is a service management software for Kubernetes that provides the following:

1. a top-level user-facing frontend to coordinate accesses to the platform,
2. service deployment in containers from predefined templates – including a Zeppelin notebook, and
3. secure provisioning of multiple services under one externally-accessible endpoint.

This dashboard is used to manage services or applications that run in Kubernetes to manage container images stored in a private docker registry, as well as to manage collections of data that are automatically attached to service and application containers when launched. It also provides:

1. a method to launch services or applications from templates that support setting variables before launch,
2. an easy and automated way to isolate services in different Kubernetes namespaces, and
3. an integrated solution to securely provision multiple services under one secure network address and port.

The latest dashboard user guide is available in the project's platform, under the "Files" tab, in the "Shared" data domain.

The Dashboard user guide is also available at the EVOLVE consortium area, under the following path:

> Consortium Files > WP3_ Workflow meta-model and specification (OnApp) > Zeppelin_CookBook > EVOLVE_Dashboard_KARVDASH

Alternatively, you can use the following user guide in order to set up a VM in your own infrastructure, containing a Zeppelin frontend, which includes the custom EVOLVE interpreter.

WP3 - Installation & Update Guide for Zeppelin Evolve interpreter

Available at the consortium area web site, under the following path:

> Consortium Files > WP3_ Workflow meta-model and specification (OnApp)> Zeppelin_CookBook

evolve
Leading the Big Data Revolution

2.

Developing Notebooks for the Evolve platform

2. Developing Notebooks for the Evolve platform

2.1. Notes and notebooks

A Zeppelin note consists of one or more “paragraphs” of executable code or scripts (layout shown in Figure below), which can be used in order to define and run snippets of code in a flexible manner. A paragraph contains code to access services, run jobs and display results. A paragraph consists of two main sections: an interactive box of code and the box for displaying the results. Paragraph commands are also displayed on the right-hand side of the interface. Each paragraph code is for one specific interpreter of the Zeppelin platform and can be executed separately (i.e. one paragraph), or the whole note (multiple paragraphs) at once.

At the top of each note there is a toolbar, shown in the following Figure, which contains a number of operations and the most important of those are summarized below.

- Execute all paragraphs in the note sequentially, in the order in which they are displayed in the note.
- Clear the result section in all paragraphs.
- Clone the current note.
- Export the current note to a JSON file.
- Schedule the execution of all paragraphs using CRON syntax.
- Configure interpreters that are bound to the current note.

- Switch display mode:
    - Default: the notebook can be shared with (and edited by) anyone who has access to the notebook.
    - Simple: similar to default, with available options shown only when your cursor is over the cell.
    - Report: only your results are visible and are read-only (no editing).

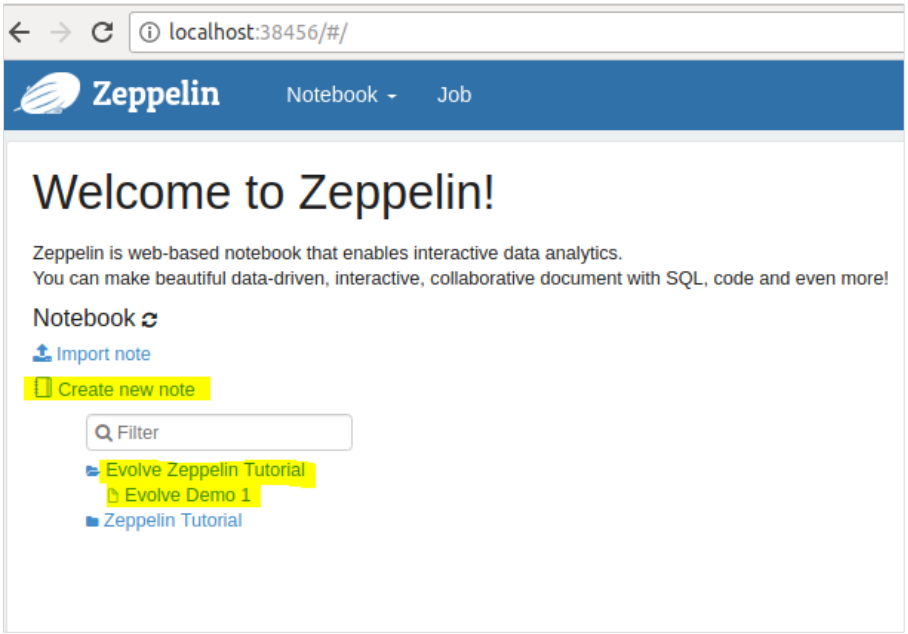
In addition, each paragraph can also be separately configured, and the most important options provided through the Zeppelin interface are listed below (Using Apache Zeppelin):

- display the paragraph id
- move the paragraph 1 level up
- move the paragraph 1 level down
- disable the run button for this paragraph
- export the current paragraph as an iframe and open the iframe in a new window

2.2. Creating and loading notes

To create a new note, select “Create new note” on the main page of Zeppelin. You can also create folders of notes and move your notes there to categorize them. In the following screenshot we see a folder named “Evolve Zeppelin Tutorial” with a note named “Evolve Demo 1”.

To load an existing note such as “Evolve Demo 1”, click on it to open in the browser.



More information on the Zeppelin UI can be found here:
https://zeppelin.apache.org/docs/0.8.1/quickstart/explore_ui.html

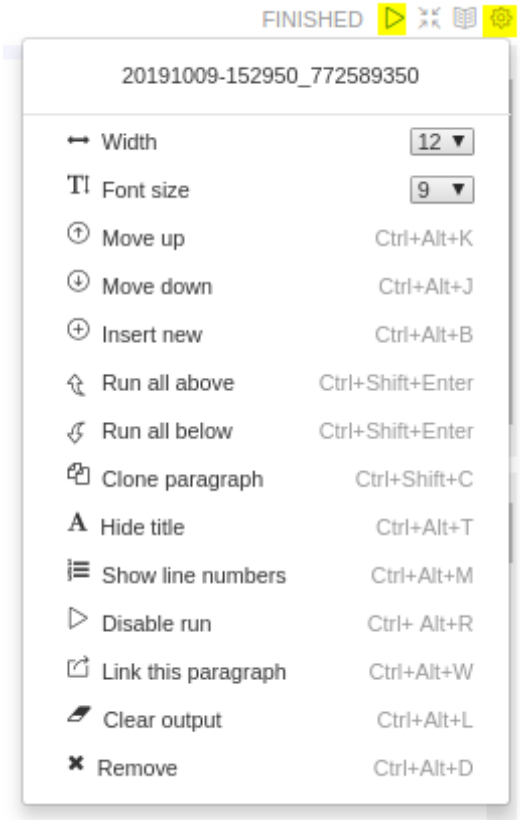
evolve
Leading the Big Data Revolution

3.

Writing new notes and paragraphs

3. Writing new notes and paragraphs

The user can create as many new paragraphs as needed, copy them, delete them if needed and execute all or part of the notebook. The execution and management (e.g. insert, copy, edit, delete paragraphs) is performed from the buttons and right-side menu, as shown in the following screenshot.



There are several Zeppelin interpreters supported in the platform, the most common ones being the Shell interpreter, Markdown, Python, Sql and several others.

For an introduction to Zeppelin notes please refer to the introductory tutorial at https://zeppelin.apache.org/docs/0.8.1/quickstart/tutorial.html

For more detailed information on Zeppelin operation and notebooks, please check the documents at https://zeppelin.apache.org/docs/0.8.1/

Evolve has developed a new custom Zeppelin interpreter based on Python. This interpreter supports the Evolve workflows and programmatic management of the testbed resources, as well as the execution of jobs.

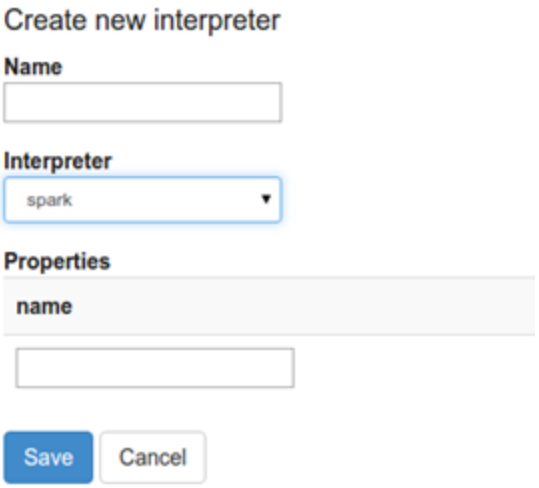
4.

Zeppelin Interpreters

4. Zeppelin Interpreters

The concept of Zeppelin interpreter allows any language/data-processing-backend to be plugged into Zeppelin. Zeppelin supports many interpreters such as Scala (with Apache Spark), Python (with Apache Spark), Spark SQL, JDBC, Markdown, Shell and so on. More interpreters can be plugged; this way, Zeppelin can be customised to support different environments and integrate other external processing systems into dataflow pipelines.

When you click the +Create button in the interpreter page, the interpreter drop-down list box will show all the available interpreters on the server.

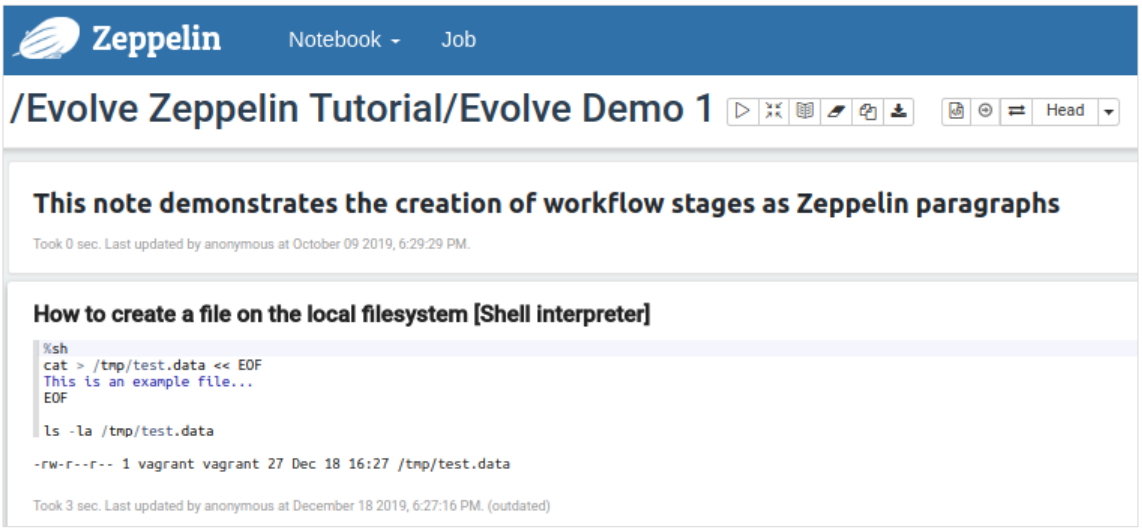


4.1. Using the Shell Interpreter

The shell interpreter is invoked using the **%sh** keyword. As depicted the following screenshot, the shell interpreter executes a local shell command via the frontend notebook. This command can do anything a local shell can do, such as create/delete/update files or run local commands to view files and execute binaries and shell scripts.

The example shell script we used below is:

```
%sh
cat > /tmp/test.data << EOF
This is an example file...
EOF

ls -la /tmp/test.data
```



4.2. The Python Interpreter

In a notebook, to enable the **Python** interpreter, the user must click on the **Gear** icon and select **Python**. In a paragraph, we use ***%python*** keyword to select the **Python** interpreter and then input all commands.

Table 4: Python interpreter property default

Property	Default	Description
python	python	Path of the already installed Python binary (could be python2 or python3). If python is not in the $PATH you can set the absolute directory (example : /usr/bin/python)

The interpreter can only work if there is already python installed (2 or 3), as the interpreter doesn't include its own python binaries. The interpreter can use all modules already installed within the python package modules (with pip, easy_install, or other).

Interpreter Architecture - Current interpreter implementation spawns new system python process through ProcessBuilder and redirects it's stdin\stdout to Zeppelin. When the interpreter is starting it launches a python process inside a Java ProcessBuilder. Python is started with -i (interactive mode) and -u (unbuffered stdin, stdout and stderr) options. Thus the interpreter has a "sleeping" python process.

Interpreter sends commands to python with a Java outputStreamWriter and reads from an InputStreamReader. To know when stop reading stdout, interpreter sends print "*!?flush reader!?*" after each command and reads stdout until he receives back the *!?flush reader!?*.

When the interpreter is starting, it sends some Python code (bootstrap.py and bootstrap_input.py) to initialize default behavior and functions (help(), z.input()...). Code written in bootstrap.py and bootstrap_input.py should always be Python 2 and 3 compliant. Python code should follow the PEP8 convention.

Matplotlib figures are displayed inline with the notebook automatically using a built-in backend (function `z.show()`) for zeppelin in conjunction with a post-execute hook. You need to have matplotlib module installed and a XServer running to use this functionality

evolve
Leading the Big Data Revolution

5.

Using the Evolve Interpreter

5. Using the Evolve Interpreter

One of the most significant features of Zeppelin is that it provides the capability to the end users to build their own interpreters. We have implemented a custom Evolve interpreter that would simplify the definition and deployment of workflows for the pilots of the *EVOLVE* project on the Argo workflow engine running on the Evolve cluster. Furthermore, we simplify the definition of Sensors on the Argo Events engine, which is an event-driven workflow automation framework for Kubernetes. Therefore, this capability to build custom plugin interpreters, is of particular importance for the implementation and interoperability of the *EVOLVE* platform.

The custom Evolve interpreter can be invoked with the %evolve keyword at the beginning of a paragraph, as shown in the screenshot below.

The Evolve interpreter is based on the Python interpreter of Zeppelin and the custom objects and directives are also written in Python, one of the most popular and widely used programming languages. Being based on the Python interpreter, the objects, commands and directives of the Evolve interpreter are Python objects themselves, which allows them to be used as Python scripts too. This is an important feature, since it allows the programmatic handling of the cluster management for images, jobs and workflows through the Evolve interpreter of Zeppelin.

One example for this is the following paragraph script, which allows the user to get and process the list of images found on the cluster, within a Zeppelin paragraph.

```
%evolve
import evolve

# initialize connection to the cluster / testbed
cluster = evolve.cluster()

imglist = cluster.image_list()

for img in imglist:
    if cluster.image_exists(img)
        imginfo = cluster.image_info(img)
        print cluster.image_tags(img)
    else:
        print("Image %s not found" % "img")
```

The commands used here will be explained in detail in the following sections.

Connect to Evolve testbed and check status FINISHED

```
%evolve
import evolve

# initialize connection to the cluster / testbed
cluster = evolve.cluster()

cluster.get_info()
imglist = cluster.image_list()

print "Jobs:", cluster.job_list()       # return list of jobs on server, running or terminated

#for jobdesc in cluster.job_list():     # check list of jobs on server
#   print cluster.job_status(jobdesc['name'])

for img in imglist:
    imginfo = cluster.image_info(img)
    print cluster.image_tags(img)
```

```
Evolve config: OK

=====> Get info from 92.43.249.202:5000
Docker version:
Client: Docker Engine - Community
 Version:           19.03.3
 API version:       1.40
 Go version:        go1.12.10
 Git commit:        a872fc2
 Built:             Tue Oct  8 00:59:54 2019
 OS/Arch:           linux/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:          19.03.3
  API version:      1.40 (minimum version 1.12)
  Go version:       go1.12.10
  Git commit:       a872fc2
  Built:            Tue Oct  8 00:58:28 2019
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.2.6
  GitCommit:        894b81a4b802e4eb2a91d1ce216b8817763c29fb
 runc:
  Version:          1.0.0-rc8
  GitCommit:        425e105d5a03fabd737a126ad93d62a9eeede87f
 docker-init:
  Version:          0.18.0
```

Took 30 sec. Last updated by anonymous at December 09 2019, 6:47:45 PM.

evolve
Leading the Big Data Revolution

6.

Evolve Interpreter Command Reference

6. Evolve Interpreter Command Reference

6.1. Interpreter configuration file

The configuration file of the Evolve interpreter contains several important parameters that are needed to access the argo/kubernetes cluster and the docker repository for images.

In the Evolve VM the file is expected at the path: /usr/local/zeppelin/evolve.config , which in the vagrant installation is symlinked to

/usr/local/zeppelin/evolve.config -> /home/vagrant/argoflow/evolve.config

The config file is a yaml file that looks like the following (password not shown):

```
vagrant@ubuntu-xenial:~$ cat /usr/local/zeppelin/evolve.config
# Evolve docker private registry (remote)
# Nova external IP docker repo:  92.43.249.202:5000
# Nova internal IP docker repo:  172.9.0.240:5000

--- # Evolve config file
  registry: { host: '92.43.249.202', port: '5000', user: 'evolve', pass:
'PASSWORD' }
  paths: { workpath: '/tmp/argo_gen_flow.yaml', workfile:
'/tmp/argo_gen_flow.yaml' }
  host: '92.43.249.202'
  workflow_engine: 'argoflow'
```

As it can be seen, the main parameters required for the registry are the host IP address, the port and credentials. Please note that the whitespace and dashes should be compliant with the yaml file format, in order for the interpreter to parse the config correctly and not result in an exception.

6.2. Evolve main objects

The Evolve Zeppelin interpreter consists of specific Objects that have a set of methods to implement different functionality. The objects are presented in Table 1 below.

Table 1: Custom interpreter objects used for Evolve workflow management.

Objects	Description
Cluster	Represents the cluster / testbed and can be called to manage the cluster. Initialize connection to the cluster / testbed, for example: `cluster = evolve.cluster()`
workflow	An Evolve workflow with multiple stages, can be created, loaded, saved and run on a cluster as a job (results in a Job object). For example to create new workflow with a specific name: `wkflow = evolve.workflow(name="cybele")`
Job	Represents a specific runtime job created from a workflow that is executed and can be managed, i.e. check status, killed, get results, etc.

These objects contain several methods that are described in detail in the table below.

6.3. Cluster object and methods

The Cluster object represents the kubernetes/argo cluster (e.g. Nova cluster used in Evolve) and can be used to manage the cluster and its jobs.

A brief description of all methods of the cluster object are shown in the table below and more detailed information can be found in the following sections.

METHOD	Description
cluster = evolve.cluster()	Initialize connection to the cluster / testbed. Use configuration file parameters for cluster host, port and credentials.
cluster.get_info()	Print cluster information (e.g. Argo cluster info, Kubernetes info, resources, etc.).
cluster.job_list()	Return and/or print the list of jobs in the cluster, running or terminated, returns dict.
cluster.job_status("jobname")	Return and/or print the status of a job named “jobname”. If the job is not found, returns None.
cluster.job_exists(“jobname”)	Returns true if a job named “jobname” exists, else returns None.
cluster.job_kill(“jobname”)	Terminates the job named “jobname” in the argo-k8s cluster. If not found returns None. The job remains in the list of argo jobs.

cluster.job_delete("jobname")	Deletes the job named "jobname" from the list of jobs in the argo / k8s cluster.
job = cluster.job_load("jobname")	Creates a job object with the properties of the job named "jobname" from the list of jobs in the argo / k8s cluster.
cluster.image_list()	Returns and/or prints a list of names of all the docker images (containers) found in the repo of the cluster.
cluster.image_upload("imagename",""tagname")	Upload a container image named "imagename" to the cluster repository with a tag "tagname".
cluster.image_info("imagename")	Return information about a specific image named "imagename" found in the cluster, in Json format.
cluster.image_tags("imagename")	Return the list of tags of a specific image named "imagename" in the cluster.
cluster.image_exists("imagename")	Returns true if an image named "imagename" is found in the cluster, else None.
cluster.image_delete("imagename")	Deletes a specific image named "imagename" from the cluster repository.

6.3.1. Initialize Cluster object

NAME	**cluster = evolve.cluster()**
Description	Initialize a connection to the cluster / testbed. Uses the configuration file parameters for cluster host, port and credentials.
Return Value	A cluster Object that can be called through its methods in a python code.
Arguments	1. "Cfgfile" (optional): Use configuration file with specific path. Default option cfgfile="evolve.config" to use the default configuration file. Example: cluster2 = evolve.cluster(cfgfile='/home/evolve/config.new')
Errors	Raises an exception if the configuration file is not found, or the config is wrong, or the connection to the cluster argo engine fails.

6.3.2. Cluster: Get information

NAME	**cluster.get_info()**
Description	Print cluster information (e.g. Argo cluster info, Kubernetes info, resources, etc.).
Return Value	(bool) True if there is no error.
Arguments	(optional) silent=Boolean: Default is False, to print the information to the output
Errors	Raises an exception if the configuration file is not found or the config is wrong, or the connection to the cluster argo engine fails.

6.3.3. Cluster: List jobs

NAME	**cluster.job_list()**
Description	Return and/or print the list of jobs in the cluster, running or terminated.
Return Value	Returns a python List of jobs found, or empty list if no jobs exist. Each job in the list is a python Dictionary with values {'status': 'X', 'duration': 'X', 'age': 'X', 'name': 'X'}. Example of returned job python List: `[{'status': 'Failed', 'duration': '5s', 'age': '11d', 'name': 'cybele-ng58z'}, {'status': 'Succeeded', 'duration': '11m', 'age': '21d', 'name': 'cybeleclean-bdzcd'},...]`
Arguments	(optional) silent=Boolean: Default is True, not to print the list to the output.
Errors	Raises an exception if the configuration file is not found or the config is wrong, or the connection to the cluster argo engine fails.

6.3.4. Cluster: Check job status

NAME	**cluster.job_status("jobname")**
Description	Return and/or print the status of a job named “jobname”. If a job is not found, returns None.

Return Value	(string) Status value of the job, which can be one of “Failed”, “Succeeded”, “Pending”, “Running”.
Arguments	(string) “jobname”: the name of the job to be loaded from the cluster
Errors	Raises an exception if the configuration file is not found or the config is wrong, or the connection to the cluster argo engine fails.

6.3.5. Cluster: Check job exists

NAME	**cluster.job_exists(“jobname”)**
Description	Check if a job named “jobname” exists in the cluster or not.
Return Value	(bool) Returns true if a job named “jobname” exists, else returns None.
Arguments	(string) “jobname”: the name of the job to be loaded from the cluster
Errors	Raises an exception if the configuration file is not found or the config is wrong, or the connection to the cluster argo engine fails.

6.3.6. Cluster: Terminate / kill job

NAME	**cluster.job_kill(“jobname”)**
Description	Terminates the job named “jobname” in the argo/k8s cluster.The job remains in the list of jobs in the cluster and can be accessed later.
Return Value	(bool) Returns True if the job named “jobname” is found and terminated. Returns False if the job does not exist or the operation failed.
Arguments	(string) “jobname”: the name of the job to be loaded from the cluster
Errors	Raises an exception if the configuration file is not found or the config is wrong, or the connection to the cluster argo engine fails.

6.3.7. Cluster: Remove job

NAME	**cluster.job_delete("jobname")**
Description	Deletes the job named "jobname" from the list of jobs in the argo / k8s cluster.
Return Value	(bool) Returns True if the job named "jobname" has been found and deleted. Returns False if the job does not exist.
Arguments	(string) "jobname": the name of the job to be loaded from the cluster
Errors	Raises an exception if the configuration file is not found or the config is wrong, or the connection to the cluster argo engine fails.

6.3.8. Cluster: Load job

NAME	**job = cluster.job_load("jobname")**
Description	Creates a job object with the properties of the job named "jobname" from the list of jobs in the argo / k8s cluster.
Return Value	Returns a job Object, created from the running cluster, with the provided jobname and the current configuration file. Returns None if the job name is not found.
Arguments	(string) "jobname": the name of the job to be loaded from the cluster
Errors	Raises an exception if the configuration file is not found or the config is wrong, or the connection to the cluster argo engine fails.

6.3.9. Cluster: Image list

NAME	**cluster.image_list()**
Description	Returns and/or prints a list of names of all the docker images (containers) found in the repo of the cluster.
Return Value	Returns a python List with all the image names (strings) found in the cluster. If no image is found the List is empty.

Arguments	(optional) silent=Boolean: Default is False, to print the information to the output
Errors	Raises an exception if the configuration file is not found or the config is wrong, or the connection to the cluster argo engine fails.

6.3.10. Cluster: Image Upload

NAME	**cluster.image_upload(“imagename”,”tagname”)**
Description	Upload a container image named “imagename” to the cluster docker image repository with a tag “tagname”.
Return Value	(bool) Returns True if the image upload was completed successfully, and False if the image was not found or the upload failed.
Arguments	(string) imagename: Name of the image to upload (string) tagname: Name of the image tag to use (optional) silent=Boolean: Default is False, to print the information to the output
Errors	Raises an exception if the configuration file is not found or the config is wrong, or the connection to the cluster docker repo cannot be established, or the upload is interrupted.

6.3.11. Cluster: Image Get Information

NAME	**cluster.image_info(“imagename”)**
Description	Return or print information about a specific image named “imagename” found in the cluster, in Json format.
Return Value	Returns a json string with information for the image, or False if the image is not found on the repository. Example of returned string: `{u'name': u'argoexec', u'tags': [u'v2.3.0']}`
Arguments	(string) imagename: Name of the image to get information about (optional) silent=Boolean: Default is True, not to print information to the output
Errors	Raises an exception if the configuration file is not found or the config is wrong, or the connection to the cluster docker repo failed, or is interrupted.

6.3.12. Cluster: Image Get Tags

NAME	**cluster.image_tags(“imagename”)**
Description	Return the list of tags of a specific image named “imagename” in the cluster. This is equivalent to the “tags” field returned by the image_info() method.
Return Value	Return the list of tags in Json string, or False if the image is not found. Example: `[u'v2.3.0']`
Arguments	(string) imagename: Name of the image to get information about. (optional) silent=Boolean: Default is True, not to print information to the output
Errors	Raises an exception if the configuration file is not found or the config is wrong, or the connection to the cluster docker repo failed, or is interrupted.

6.3.13. Cluster: Image Exists

NAME	**cluster.image_exists(“imagename”)**
Description	Checks if an image named “imagename” is found in the cluster docker repo.
Return Value	Returns True if an image named “imagename” is found in the cluster, else it returns False.
Arguments	(string) imagename: Name of the image to get information about. (optional) silent=Boolean: Default is True, not to print information to the output
Errors	Raises an exception if the configuration file is not found or the config is wrong, or the connection to the cluster docker repo failed, or is interrupted.

6.3.14. Cluster: Image Delete

NAME	**cluster.image_delete("imagename")**
Description	Deletes a specific image named "imagename" from the cluster repository.
Return Value	Returns True if an image named "imagename" has been found in the cluster and the delete operation succeeded, else it returns False.
Arguments	(string) imagename: Name of the image to get information about. (optional) silent=Boolean: Default is True, not to print information to the output
Errors	Raises an exception if the configuration file is not found or the config is wrong, or the connection to the cluster docker repo failed, or is interrupted.

6.3.15. Example cluster management paragraph

As discussed previously, these Evolve interpreter objects and methods are essentially Python objects and they can be used for scripting in Python notes supported in Zeppelin. This allows the programmatic management not only of workloads, but also cluster jobs and images.

An example of this scripting concept is shown in the following paragraph, using the cluster object methods described above.

```
%evolve
import evolve

# initialize connection to the cluster / testbed
cluster = evolve.cluster()

cluster.get_info(silent=False)             # print cluster information
imglist = cluster.image_list()             # get list of images

for img in imglist:
    imginfo = cluster.image_info(img)      # get image information
    print cluster.image_tags(img)

print "Jobs:", cluster.job_list()   # return list of server jobs,
                                      running or terminated

for jobdesc in cluster.job_list():  # check list of jobs on server
    print cluster.job_status(jobdesc['name'])
```

6.4. Workflow object and methods

The Workflow object represents the Evolve workflow with one or multiple stages that will be created, saved in yaml files (compatible with the argo workflow engine) and will be run on the kubernetes/argo cluster (e.g. Nova cluster used in Evolve).

We also support Sensors from Argo Events that are a set of event dependencies (inputs) and triggers (outputs). It listens to events from one or more gateways and acts as an event dependency manager.

A brief description of all methods of the workflow object for Evolve workflows and Evolve Sensors are shown in the table below and more detailed information can be found in the following sections.

METHOD	Description
workflow=evolve.workflow(name="WKName", secrets="regcred", cfgfile="evolve.conf", sensor = False, sensorinit = {})	Initialize an Evolve workflow with the specific name "WKName" based on the configuration "evolve.conf". Returns a new Workflow object.
workflow.addsensortemplate (templatename, inputs, outputs)	Creates a sensor template that will contain steps, scripts, containers or resources
workflow.addsensorstep(templatename, name, template, loopitems, args)	Adds a step to a specific template of a sensor
workflow.addsensorscript(templatename, name, image, command, source, inputs, outputs, volume)	Adds a script to a specific template of a sensor
workflow.addsensorresource (templatename, name, action, manifest, condition)	Adds a resource to a specific template of a sensor
workflow.addsensorcontainer (templatename, name, inputs, image, command, args)	Adds a container to a specific template of a sensor
workflow.addvolume(name, type, attribs)	Add a data volume to the workflow with the specified name, and attributes (hostpath, nfs, h3 map, persistent, etc.)
workflow.addstage(name, image, env, command, args, volumes)	Add a new stage in the workflow, with the specified name, container image, command to run in the container, arguments, and volumes to mount for accessing data.

workflow.save()	Save workflow definition (including volumes & stages) into a yaml file compatible with the Argo workflow engine.
workflow.save_sensor(silent)	Save sensor definition into a yaml file compatible with the Argo events engine.
workflow.load("/pathname.yaml")	Load a workflow definition (including volumes & stages) from a yaml file at the specified path. The yaml definition must be compatible with the Argo workflow engine.
workflow.show()	Print the current workflow definition (including name, volumes & stages) to the output.
job = workflow.run()	Use the workflow yaml file to execute the workflow in the Argo engine of the configured cluster. This includes sending the specification file to the Argo/Kubernetes cluster and starting it in the specified environment with the defined resources. Returns a job object which can be used to manage the job.

6.4.1. Workflow: Initialize

NAME	**evolve.workflow(name, secrets, cfgfile, sensor, sensorinit)**
Description	Initialize an Evolve workflow with the specific name "WKName".
Return Value	Returns a new Workflow object which can be used to manage the workflow.
Arguments	1. (string) name : The name of the workflow to be created. This will also be used as a prefix for the jobs based on this workflow. 2. (optional, string) secrets='secfile': Definition of secrets for the argo workflow. Default value is 'regcred' and should work for most workflow definitions. 3. (optional, string) cfgfile='path': Path of configuration file. The default value is 'evolve.config' and should not need to be changed. 4. (optional, boolean) sensor: We create a sensor object instead of a workflow 5. (optional, array) sensorinit, A json array including information about the sensor, such as volumes, result_event, result_destination, and other dependencies
Errors	Raises an exception if the configuration file is not found or the config is wrong, or the connection to the cluster argo engine fails.

Example call	`# Create a new workflow` `myflow = evolve.workflow(name="calendar-sensor", sensor=True)`

6.4.2. Workflow: Add Sensor Template

NAME	**evolve.addsensortemplate(templatename, inputs, outputs)**
Description	Initialize a Evolve sensor template.
Return Value	Returns a new Template object which can be used to manage the sensor.
Arguments	1. (string) templatename: The name of the template to be created. 2. (optional, string) inputs: The input parameters for the sensor. 3. (optional, string) outputs: The outputs parameters for the sensor.
Errors	Raises an exception if the configuration file is not found or the config is wrong, or the connection to the cluster argo engine fails.
Example call	`# Add sensor template` `myflow.addsensortemplate("nested-getter-and-processor")`

6.4.3. Workflow: Add Sensor Step

NAME	**evolve.addsensorstep(templatename, name, template, loopitems, args)**
Description	Initialize a Evolve sensor template.
Return Value	Returns a new Template object which can be used to manage the sensor.
Arguments	1. (string) templatename: The name of the template that the step belongs to. 2. (string) name: The name of the step. 3. (string) template: The name of the template inside the step. 4. (array) loopitems: The parameters of the step. 5. (array) args: The values for the parameters of the step.
Errors	Raises an exception if the sensor object does not have a template with the name given in the templatename variable.

Example call	`# Add sensor step` `myflow.addsensorstep("nested-getter-and-processor", "getter-and-processor", "getter-and-processor")`

6.4.4. Workflow: Add Sensor Script

NAME	**evolve.addsensorscript(templatename, name, image, command, source, inputs, outputs, volume)**
Description	Initialize a Evolve sensor template.
Return Value	Returns a new Template object which can be used to manage the sensor.
Arguments	1. (string) templatename: The name of the template that the step belongs to. 2. (string) name: The name of the script. 3. (string) image: The docker image used in the script. 4. (string) command: The command of the docker image that the script will execute. 5. (optional, array) inputs: The input parameters used in the script. 6. (optional, array) outputs: The output parameters used in the script. 7. (optional, string) volume: A volume that will be mounted in the container.
Errors	Raises an exception if the sensor object does not have a template with the name given in the templatename variable.
Example call	`# Add sensor step script` `myflow.addsensorscript("getter-and-processor", "getter", "evolve-registry.microlab.ntua.gr:5050/wp6/p4/getter:v0.2", "sh", "ls -al")`

6.4.5. Workflow: Add Sensor Resource

NAME	**evolve.addsensorresource (templatename, name, action, manifest, condition)**
Description	Initialize a Evolve sensor template.
Return Value	Returns a new Template object which can be used to manage the sensor.
Arguments	1. (string) templatename: The name of the template that the resource belongs to. 2. (string) name: The name of the resource. 3. (string) action: The action of the resource. 4. (string) manifest: The manifest of the resource. 5. (optional, boolean) condition: The condition of the resource.
Errors	Raises an exception if the sensor object does not have a template with the name given in the templatename variable.
Example call	`# Add sensor step resource` `myflow.addsensorresource(templatename="spark-processor-cleaner", name="wp6-p4-spark-driver-cleanup-impl", action="delete", manifest="{{workflow.parameters.label}}")`

6.4.6. Workflow: Add Sensor Container

NAME	**evolve.addsensorcontainer (templatename, name, inputs, image, command, args)**
Description	Initialize a Evolve sensor template.
Return Value	Returns a new Template object which can be used to manage the sensor.
Arguments	1. (string) templatename: The name of the template that the container belongs to. 2. (string) name: The name of the container. 3. (optional, array) inputs: The input parameters of the container. 4. (string) image: The image that the container instantiates. 5. (string) command: The command that the container executes. 6. (optional, string) arguments: The arguments that of the command
Errors	Raises an exception if the sensor object does not have a template with the name given in the templatename variable.
Example call	`# Add sensor step container myflow.addsensorcontainer(templatename="spark-processor-cleaner", name="wp6-p4-spark-processor-impl", inputs=["label"], image="evolve-registry.microlab.ntua.gr:5050/wp6/p4/spark:2.4.0", command="touch", args=["/tmp/test"])`

6.4.7. Workflow: Add Volume

NAME	**workflow.addvolume(name, path, server, vtype, mapname, mapitems)**
Description	Add a data volume to the workflow with the specified name, and attributes (hostpath, nfs, h3 map, persistent, etc.)
Return Value	Returns True if the volume was added without error, else returns False.
Arguments	1. (string) name=’string’ : Name of the volume to add 2. (string) path='string' : Storage path of the volume to add 3. (optional, string) server='string' : Storage server name, default value is “localhost” 4. (optional, string) vtype=’string’ : Type of storage. Default value is 'local', but can also be ‘nfs’ or ‘map’ type (see examples below). 5. (optional, string) mapname=’string’ : name of map, used if the storage type is ‘map’. Default value is None. 6. (optional, dictionary) mapitems={ dict } : map items (key,value) in a dictionary, used if the storage type is ‘map’. Default value is None.

Errors	Raises an exception if the provided arguments or argument types are incorrect, or some arguments are missing.

Example call	`myflow.addvolume(name='archive',`
	`vtype='nfs',`
	`server='192.168.122.1',`
	`path='/home_nfs/home_john/my-workflow/files')`
	`myflow.addvolume(name='config',`
	`vtype='map',`
	`mapname='example-h3-config',`
	`mapitems={'key':'h3-config',`
	`'path':"h3config.ini"})`

6.4.8. Workflow: Add Stage

NAME	**workflow.addstage(name, image, command, args, params, inputs, outputs, volumes, resources, env, parallel)**
Description	Add a new stage in the workflow, with the specified name, container image, command to run in the container, arguments, and volumes to mount for accessing data.
Return Value	Returns True if the volume was added without error, else returns False.
Arguments	1. (string) name=’string’ : Name of the workflow stage 2. (string) image='string' : Name of the docker image to use for this stage 3. (string, or list of strings) command=”string” : The command to run in this stage, command=[“cmd1”, “cmd2”]: Creates a loop step with the same container and different commands an arguments (the size of this list must be same with) 4. (list of strings, or list of list of strings+) args = [“arg1”, “arg2”, ...] for the command, args = [[“arg11”, “arg12”, ...], [“arg21”, “arg22”, ...], ...] Creates a loop step with the same container and different commands an arguments 5. (optional, dict of two lists) params=[“name”:{}, “value”:{}]: Arguments defined in the step 6. (optional, list of strings) inputs =[“input1”, “input2”, ...]: Input parameters of the step 7. (optional, list of strings) outputs =[“output1”, “output2”, ...]: Output parameters of the step 8. (optional, list of dicts) volumes=[{ volume1 }, {volume2}...] : Storage volumes to use for this stage, default value is empty list. 9. (optional, list of dicts) resources=[{resource1},...] : List of resources (key-value pairs) that are provided for this stage. This can be for example some requirements for hardware acceleration. 10. (optional, list of dicts) env=[{env1}, {env2}...] : List of environment variables (as key-value pairs) that are provided for this workflow stage (check examples below). 11. (optional, boolean) parallel: Makes this step parallel to the previous one.

Errors	Raises an exception if the provided arguments or argument types are incorrect, or some arguments are missing.

Example call	`myflow.addstage(name='vi', image="172.9.0.240:5000/cybeletech:v2", env=[ {'H3_CONFIGURATION':"/h3conf"} ], command="python3", args=["compute_vegetation_index.py", "/compute_vegetation_index.json"], volumes=[{'name':'config', 'mountPath':"/h3conf"}])`
	
	`myflow.addstage(name='mergeImages', image="172.9.0.240:5000/evolve/merge-images:v1", command="bash", args=["run.sh"], resources=[{"nvidia.com/gpu": 1}], env=[{'H3_CONFIGURATION':"/h3conf/"}, {'IMAGES':"/var/data/DataVin1/"}, {'INPUT':"/var/data/Result/"}, {'OUTPUT':"/var/data/Result/"}, {'CONFIG_FOLDER':"/var/data/DataVin1/"}], volumes=[{'name':'config', 'mountPath':"/h3conf"}])`

6.4.9. Workflow: Save

NAME	**workflow.save()**
Description	Save workflow definition (including volumes & stages) into a yaml file compatible with the Argo workflow engine.
Return Value	Returns True if the workflow has been saved, else False.
Arguments	(optional) silent=Boolean: Default is False, to print information to the output
Errors	Raises an exception if the configuration file is not found or the config is wrong, or the path of the workflow yaml file is inaccessible.

6.4.10. Workflow: Load

NAME	**workflow.load("/pathname.yaml")**
Description	Load a workflow definition (including volumes & stages) from a yaml file at the specified path. The yaml definition must be compatible with the Argo workflow engine.
Return Value	Returns True if the workflow has been loaded, else False.
Arguments	(optional) silent=Boolean: Default is False, to print information to the output
Errors	Raises an exception if the configuration file is not found or the config is wrong, or the path of the workflow yaml file is inaccessible.

6.4.11. Workflow: Show

NAME	**workflow.show()**
Description	Prints the current workflow definition (including name, volumes & stages) to the output.
Return Value	(string) Returns the current workflow name and definition as a string.
Arguments	(optional) silent=Boolean: Default is False, to print information to the output
Errors	None

6.4.12. Workflow: Run

NAME	**job = workflow.run()**
Description	Sends the saved workflow to execute in the Argo workflow engine of the configured cluster. This includes sending the yaml specification file to the Argo/Kubernetes cluster and starting it in the specified environment with the defined resources.
Return Value	Returns a Job object which can be used to manage the specific job.
Arguments	(optional) silent=Boolean: Default is False, to print information to the output
Errors	Raises an exception if the configuration file is not found or the config is wrong, or the connection to the cluster argo engine fails.

6.5. Job object and methods

The Workflow object represents the Evolve workflow with one or multiple stages that will be created, saved in yaml files (compatible with the argo workflow engine) and will be run on the kubernetes/argo cluster (e.g. Nova cluster used in Evolve).

A brief description of all methods of the workflow object are shown in the table below and more detailed information can be found in the following sections.

METHOD	Description
job.status()	Return and/or print the status of the specific job.
job.inprogress() / job.active()	Returns True if the job is still active / in progress.
job.wait_for_completion()	Waits for job completion and return the job exit status.
job.get_results()	Returns the job results in a raw file, if the job is done
job.terminate() / job.kill()	Terminates the running job by force.

6.5.1. Job: Get Status

NAME	**job.status()**
Description	Return and/or print the status of the specific job.
Return Value	(string) Status value of the job, which can be one of “Failed”, “Succeeded”, “Pending”, “Running”.
Arguments	(optional) silent=Boolean: Default is True, not to print information to the output.
Errors	Raises an exception if the configuration file is not found or the config is wrong, or the connection to the cluster argo engine failed.

6.5.2. Job: Active / In progress

NAME	**job.inprogress() / job.active()**
Description	Check if the job is still active / in progress in the cluster argo engine.
Return Value	Returns True if the job is still active / in progress, else returns False.
Arguments	(optional) silent=Boolean: Default is True, not to print information to the output.
Errors	Raises an exception if the configuration file is not found or the config is wrong, or the connection to the cluster argo engine failed.

6.5.3. Job: Wait for completion

NAME	**job.wait_for_completion()**
Description	Waits for job completion and return the job exit status.
Return Value	Returns True is the job execution was successful
Arguments	(optional) checksec=(int) number of seconds to check the job completion status, default value is 20 seconds. (optional) timeoutsec=(int) maximum time to wait for job completion, default value is 600 seconds.

	(optional) silent=Boolean: Default is True, not to print information to the output.
Errors	Raises an exception if the configuration file is not found or the config is wrong, or the connection to the cluster argo engine failed.

6.5.4. Job: Get Results

NAME	**job.get_results("resultpath")**
Description	job.get_results("resultpath")
Return Value	Download the results in a raw file, if the job is done.
Arguments	(string) Path name to raw file of results, if the job is completed.
Errors	(string) resultpath="path" Remote path name of results file. (optional) silent=Boolean: Default is True, not to print information to the output.

6.5.5. Terminate / Kill

NAME	**job.terminate() / job.kill()**
Description	Terminates the running job by force.
Return Value	Returns True if the job termination has been successful, otherwise False.
Arguments	(optional) silent=Boolean: Default is True, not to print information to the output.
Errors	Raises an exception if the configuration file is not found or the config is wrong, or the connection to the cluster argo engine failed.

7.

Examples of workflows and jobs

7. Examples of workflows and jobs

7.1. Cybeletech workflow

The example code shown below is based on the workflow provided by partner “CYBELETECH”. It starts with the definition of the workflow name (“Cybele”) and the creation of the “cyb-folder” volume. This data volume definition contains the name of the volume, and the remote or local access path to that volume, so that containers running in the workflow stages can access data on that volume. Next the interpreter adds two stages in the workflow, each of which describes the container name, the command line and arguments, as well as the data path that the application in the container uses. In this specific example, there are two workflow stages, performed one after the other namely the “vi” and the “segmentation” workflows.

After these steps, when the workflow definition is complete, the save() function saves the Argo workload definition yaml in a temporary local file. Next, when we call the run() function, the loaded yaml file is uploaded to the execution cluster, where it is stored and provided as input to the Argo workflow engine to execute on the configured Kubernetes cluster. The containers in the stages of the workflow have already been uploaded and are accessible in the Nova cluster and are scheduled according to the workflow definition we have written in the notebook.

We are using CYBELETECH’s workflow as an example, but the same rules and configurations apply to all of the Evolve workflows, as shown in the rest of this user guide.

```
%evolve
import evolve

# Create a new workflow
myflow = evolve.workflow(name="cybele")

myflow.addvolume(name='archive',
                vtype='nfs',
                server='192.168.122.1',
                path='/home_nfs/home_kozanitc/workflows/cybele-workflow/files')

myflow.addvolume(name='config',
                vtype='map',
                mapname='example-h3-config',
                mapitems={'key':'h3-config', 'path':"h3config.ini"})

myflow.addstage(name='loadData',
                image="172.9.0.240:5000/cybeleseedh3:v2",
                env=[ {'H3_CONFIGURATION':"/h3conf"} ],
                command="python3",
                args=["/putDataCybele.py", "/archive/files/"],
                volumes=[{'name':'archive', 'mountPath':"/archive"},
{'name':'config', 'mountPath':"/h3conf"}])

myflow.addstage(name='vi',
                image="172.9.0.240:5000/cybeletech:v2",
                env=[ {'H3_CONFIGURATION':"/h3conf"} ],
                command="python3",
                args=["compute_vegetation_index.py",
"/compute_vegetation_index.json"],
                volumes=[{'name':'config', 'mountPath':"/h3conf"}])

myflow.addstage(name='segmentation',
                image="172.9.0.240:5000/cybeletech:v2",
                env=[ {'H3_CONFIGURATION':"/h3conf"} ],
                command="bash",
                args=["segmentation.sh"],
                volumes=[{'name':'config', 'mountPath':"/h3conf"}])

# Parallel Stage
myflow.addstage(name='mergeImages',
        image="172.9.0.240:5000/evolve/merge-images:v1",
        command="bash",
        args=["run.sh"],
        resources=[ {"nvidia.com/gpu": 1} ],
        env=[ {'H3_CONFIGURATION':"/h3conf/"}, {'IMAGES':"/var/data/DataVin1/"},
                    {'INPUT':"/var/data/Result/"}, {'OUTPUT':"/var/data/Result/"},
{'CONFIG_FOLDER':"/var/data/DataVin1/"} ],
        volumes=[{'name':'config', 'mountPath':"/h3conf"}],
        parallel=True) # give parallel=True


# Loop Stage
myflow.addstage(name='splitImages',
        image="172.9.0.240:5000/evolve/split-images:v1",
        command=["bash", "bash"], # runs container split-images:v1 two times with
cmd bash.sh
        args=[["run.sh"],["run.sh"]], # and arguments run.sh and run.sh
        resources=[ {"nvidia.com/gpu": 1} ],
        env=[ {'H3_CONFIGURATION':"/h3conf/"}, {'IMAGES':"/var/data/DataVin1/"},
```

```
                    {'OUTPUT':"/var/data/Result/"}, {'CONFIG_FOLDER':"/var/data"},
{'N_SPLITS':8} ],
        volumes=[{'name':'config', 'mountPath':"/h3conf"}])

myflow.addstage(name='spark',
                image="172.9.0.240:5000/spark:v2.4.0",
                env=[ {'H3_CONFIGURATION':"/h3conf"} ],
                command="/opt/spark/bin/spark-submit",
                args=[ "--master", "k8s://https://172.9.0.240:6443", "--deploy-
mode", "cluster",
                                "--conf",
"spark.kubernetes.container.image=172.9.0.240:5000/spark:v2.4.0",
                                "--conf",
"spark.kubernetes.driverEnv.H3_CONFIGURATION=/h3conf/", "/pyspark-app.py" ],
                volumes=[{'name':'config', 'mountPath':"/h3conf"}])

myflow.addstage(name='cleanData',
                image="172.9.0.240:5000/cybeleseedh3:v2",
                env=[ {'H3_CONFIGURATION':"/h3conf"} ],
                command="python3",
                args=["/cleanBucketCybele.py"],
                volumes=[{'name':'archive', 'mountPath':"/archive"},
{'name':'config', 'mountPath':"/h3conf"}])

myflow.save()

myflow.show()

job = myflow.run()

if not job.active() or not job.wait_for_completion():
       evolve.error("execution failed")

job.status()

#results = job.get_results()

#job.terminate()
```

7.1.1. Argo events sensor example

The example notebook shown below is based on the workflow provided by partner “Neurocom”. It starts with the definition of the sensor with name (“kafka-sensor”). The workflow includes a single template (“spark-processor-cleaner”), three sensorsteps, two sensorresources, and one sensorcontainer.

```
%evolve
import evolve

myflow = evolve.workflow(name="kafka-sensor", sensor=True,
sensorinit={"sensor_params":[{"name":"message", "value":"hello world"},
{"name":"label", "value":"hello world"}], "res_event":"kafka-gateway:wp6-p4-events",
"res_dest":"spec.arguments.parameters.1.value", "ImagePullSecrets":"wp6-p4-regcred",
"dependencies": "kafka-gateway:wp6-p4-events"})

# Add sensor template
#myflow.addsensortemplate("spark-processor-cleaner", ["label", "test"],
{"name":["generated-message", "generated-message2"], "path":["/tmp/save.txt",
"/tmp/save2.txt"]})
myflow.addsensortemplate("spark-processor-cleaner", ["label"])

# Add sensor step
myflow.addsensorstep("spark-processor-cleaner", "wp6-p4-spark-processor-def", "wp6-
p4-spark-processor-impl", args=[{"name":"label",
"value":"{{inputs.parameters.label}}"}])

# Add sensor step container
myflow.addsensorcontainer(templatename="spark-processor-cleaner", name="wp6-p4-
spark-processor-impl", inputs=["label"], image="evolve-
registry.microlab.ntua.gr:5050/wp6/p4/spark:2.4.0", command="/opt/spark/bin/spark-
submit", args=["--master", "k8s://https://147.102.37.161:6443",
                    "--deploy-mode", "cluster",
                    "--conf","spark.kubernetes.namespace=wp6-p4",
                    "--conf",
"spark.kubernetes.authenticate.driver.serviceAccountName=spark",
                    "--conf",
"spark.kubernetes.container.image.pullPolicy=Always",
                    "--conf",
"spark.kubernetes.driver.pod.name={{inputs.parameters.label}}",
                    "--conf",
"spark.kubernetes.driver.volumes.persistentVolumeClaim.wp6-p4-pvc-
nfs.options.claimName=wp6-p4-pvc-nfs",
                    "--conf",
"spark.kubernetes.driver.volumes.persistentVolumeClaim.wp6-p4-pvc-
nfs.mount.readOnly=false",
                    "--conf",
"spark.kubernetes.driver.volumes.persistentVolumeClaim.wp6-p4-pvc-
nfs.mount.path=/data/",
                    "--conf",
"spark.kubernetes.executor.volumes.persistentVolumeClaim.wp6-p4-pvc-
nfs.options.claimName=wp6-p4-pvc-nfs",
                    "--conf",
"spark.kubernetes.executor.volumes.persistentVolumeClaim.wp6-p4-pvc-
nfs.mount.readOnly=false",
                    "--conf",
```

```
"spark.kubernetes.executor.volumes.persistentVolumeClaim.wp6-p4-pvc-
nfs.mount.path=/data/",
                     "--conf",
"spark.kubernetes.driver.volumes.persistentVolumeClaim.wp6-p4-pvc-spark-
config.options.claimName=wp6-p4-pvc-spark-config",
                     "--conf",
"spark.kubernetes.driver.volumes.persistentVolumeClaim.wp6-p4-pvc-spark-
config.mount.path=/opt/spark/etl-config/",
                     "--conf",
"spark.kubernetes.driver.volumes.persistentVolumeClaim.wp6-p4-pvc-spark-
config.mount.readOnly=true",
                     "--conf", "spark.kubernetes.container.image=evolve-
registry.microlab.ntua.gr:5050/wp6/p4/spark:2.4.0",
                     "--conf",
"spark.kubernetes.authenticate.driver.serviceAccountName=spark",
                     "--conf", "spark.dynamicAllocation.enabled=false",
                     "--class", "lu.neurocom.evolve.core.IngestRTBusEvent",
                     "--conf", "spark.evolve.prop=/opt/spark/etl-
config/test.properties",
                     "--conf","spark.executor.memory=4g",
                     "--conf","spark.driver.memory=4g",
                     "--
conf","spark.source.dir=/data/{{inputs.parameters.label}}.json",
                     "local:///opt/spark/wp6-p4-jar/etl-assembly-0.2.jar"])

# Add sensor step
myflow.addsensorstep("spark-processor-cleaner", "wp6-p4-spark-driver-status-def",
"wp6-p4-spark-driver-status-impl")

# Add sensor step resource
myflow.addsensorresource(templatename="spark-processor-cleaner", name="wp6-p4-spark-
driver-status-impl", action="get", manifest="{{workflow.parameters.label}}",
condition=True)

# Add sensor step
myflow.addsensorstep("spark-processor-cleaner", "wp6-p4-spark-driver-cleanup-def",
"wp6-p4-spark-driver-cleanup-impl")

# Add sensor step resource
myflow.addsensorresource(templatename="spark-processor-cleaner", name="wp6-p4-spark-
driver-cleanup-impl", action="delete", manifest="{{workflow.parameters.label}}")

# Save the workflow
myflow.save_sensor()
```

7.1.2. Argo yaml file generated from the Zeppelin note

The yaml file for the argo workflow engine, generated from the above Zepellin paragraph (saved via myflow.save()) is the following:

```
apiVersion: argoproj.io/v1alpha1
```

```
kind: Workflow
metadata:
  generateName: cybele-
spec:
  imagePullSecrets:
  - name: regcred
  entrypoint: cybele
  volumes:
      - name: archive
        nfs:
            server: 192.168.122.1
            path: /home_nfs/home_masourod/cybele-workflow/files
      - name: config
        configMap:
          name: example-h3-config
          items:
          - key: h3-config
            path: h3config.ini
  templates:
  - name: cybele
    steps:
    - - name: loadData
        template: loadData
    - - name: vi
        template: vi
    - - name: segmentation
        template: segmentation
    - - name: spark
        template: spark
    - - name: cleanData
        template: cleanData

  - name: loadData
    container:
      image: 172.9.0.240:5000/cybeleseedh3:v2
      env:
      - name: H3_CONFIGURATION
        value: "/h3conf"
      command: ["python3"]
      args: ["/putDataCybele.py", "/archive/files/"]
      volumeMounts:
      - name: archive
        mountPath: "/archive"
      - name: config
        mountPath: "/h3conf"

  - name: vi
    container:
      image: 172.9.0.240:5000/cybeletech:v2
      env:
      - name: H3_CONFIGURATION
        value: "/h3conf"
      command: ["python3"]
      args: ["compute_vegetation_index.py", "/compute_vegetation_index.json"]
      volumeMounts:
      - name: config
        mountPath: "/h3conf"

  - name: segmentation
    container:
```

```
      image: 172.9.0.240:5000/cybeletech:v2
      env:
      - name: H3_CONFIGURATION
        value: "/h3conf"
      command: ["bash"]
      args: ["segmentation.sh"]
      volumeMounts:
      - name: config
        mountPath: "/h3conf"

  - name: spark
    container:
      image: 172.9.0.240:5000/spark:v2.4.0
      env:
      - name: H3_CONFIGURATION
        value: "/h3conf"
      command: ["/opt/spark/bin/spark-submit"]
      args: ["--master", "k8s://https://172.9.0.240:6443", "--deploy-mode",
"cluster", "--conf",
"spark.kubernetes.container.image=172.9.0.240:5000/spark:v2.4.0", "--conf",
"spark.kubernetes.driverEnv.H3_CONFIGURATION=/h3conf/", "/pyspark-app.py"]
      volumeMounts:
      - name: config
        mountPath: "/h3conf"

  - name: cleanData
    container:
      image: 172.9.0.240:5000/cybeleseedh3:v2
      env:
      - name: H3_CONFIGURATION
        value: "/h3conf"
      command: ["python3"]
      args: ["/cleanBucketCybele.py"]
      volumeMounts:
      - name: archive
        mountPath: "/archive"
      - name: config
        mountPath: "/h3conf"
```

7.2. Loading a workflow from a custom yaml file

The following note script loads a workflow from custom yaml that has been created by the user. Please note that this yaml file must be compatible with the definition of the argo workflow engine yaml files. If there is an error, it will be shown at the run() phase, where the file is parsed and executed.

```
%evolve
import evolve
```

```
# Create a new workflow
myflow = evolve.workflow(name="wp6kafka")

myflow.load("/tmp/wp6-p4-kafka-sensor.yaml")

myflow.save()

myflow.show()

job = myflow.run()

if not job.active() or not job.wait_for_completion():
       evolve.error("execution failed")

#results = job.get_results()

#job.terminate()
```

8.

Appendix I

8. Appendix I

8.1. Cluster management via Zeppelin note and output

```
%evolve
import evolve

# initialize connection to the cluster / testbed
cluster = evolve.cluster()

# ==> cluster cmds:
cluster.get_info()                          # print cluster information

print "Jobs:", cluster.job_list()           # return list of jobs on server, running
or terminated

for jobdesc in cluster.job_list():          # check list of jobs on server
       print cluster.job_status(jobdesc['name'])

===============
  OUTPUT:
===============
Evolve config: OK

==>>> Get info from 92.43.249.202:5000
Docker version:
Client: Docker Engine - Community
 Version:           19.03.3
 API version:       1.40
 Go version:        go1.12.10
 Git commit:        a872fc2
 Built:             Tue Oct  8 00:59:54 2019
 OS/Arch:           linux/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:          19.03.3
  API version:      1.40 (minimum version 1.12)
  Go version:       go1.12.10
  Git commit:       a872fc2
  Built:            Tue Oct  8 00:58:28 2019
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.2.6
  GitCommit:        894b81a4b802e4eb2a91d1ce216b8817763c29fb
 runc:
  Version:          1.0.0-rc8
  GitCommit:        425e105d5a03fabd737a126ad93d62a9eeede87f
 docker-init:
  Version:          0.18.0
  GitCommit:        fec3683

Kubernetes info:
Kubernetes master is running at https://92.43.249.202:6443
CoreDNS is running at https://92.43.249.202:6443/api/v1/namespaces/kube-
system/services/kube-dns:dns/proxy
```

```
To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.


==>>> List images from Registry 92.43.249.202:5000
/usr/local/lib/python2.7/dist-packages/urllib3/connectionpool.py:1004:
InsecureRequestWarning: Unverified HTTPS request is being made. Adding certificate
verification is strongly advised. See:
https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
  InsecureRequestWarning,
0. Image: ais_data_getter
1. Image: argoexec
2. Image: argoproj/gateway-client
3. Image: argoproj/kafka-gateway
4. Image: argoproj/sensor
5. Image: argoui
6. Image: caffe_test
7. Image: cloudsuite/in-memory-analytics
8. Image: cloudsuite/movielens-dataset
9. Image: cybeleseedh3
10. Image: cybeletech
11. Image: evolve/change_detection
12. Image: evolve/merge-images
13. Image: evolve/split-images
14. Image: evolve/vine-controller
15. Image: mlbench/mlbench-master
16. Image: mlbench/mlbench-worker
17. Image: openaccesshub_downloadsarimage
18. Image: openaccesshub_getproductlist
19. Image: postgres
20. Image: redis
21. Image: spark
22. Image: spark-stream
23. Image: sumo
24. Image: thales-seed-h3
25. Image: vine_controller
26. Image: workflow-controller
27. Image: wp6/p4/pusher
28. Image: wp6/p4/spark
29. Image: wp6/p4/spark-with-jar
30 images found.
Jobs: [{'status': 'Failed', 'duration': '3s', 'age': '1d', 'name': 'cybeleclean-
bdzcd'}, {'status': 'Failed', 'duration': '3s', 'age': '1d', 'name': 'cybeleclean-
wv89r'}, {'status': 'Failed', 'duration': '4s', 'age': '1d', 'name': 'thales-
p8pc6'}, {'status': 'Failed', 'duration': '3s', 'age': '2d', 'name': 'cybele-
nsmv5'}, {'status': 'Succeeded', 'duration': '10s', 'age': '6d', 'name': 'open-
access-hub-flow-sph-b7cb5'}, {'status': 'Succeeded', 'duration': '13m', 'age': '6d',
'name': 'open-access-hub-flow-sph-6x2jt'}, {'status': 'Succeeded', 'duration':
'15m', 'age': '7d', 'name': 'open-access-hub-flow-sph-5ggcc'}, {'status':
'Succeeded', 'duration': '3s', 'age': '7d', 'name': 'streaming-d9x6r'}, {'status':
'Succeeded', 'duration': '1m', 'age': '7d', 'name': 'streaming-f8cns'}, {'status':
'Failed', 'duration': '6s', 'age': '7d', 'name': 'cybele-pc6f5'}, {'status':
'Failed', 'duration': '17s', 'age': '8d', 'name': 'thalesclean-kbm2v'}, {'status':
'Failed', 'duration': '4s', 'age':'8d', 'name': 'cybeleclean-b5h5g'}, {'status':
'Succeeded', 'duration': '1m', 'age': '8d', 'name': 'hello-world-w6bx5'}, ... ]

==>>> Check job at 92.43.249.202:5000
Name:                cybeleclean-bdzcd
Namespace:           default
ServiceAccount:      default
```

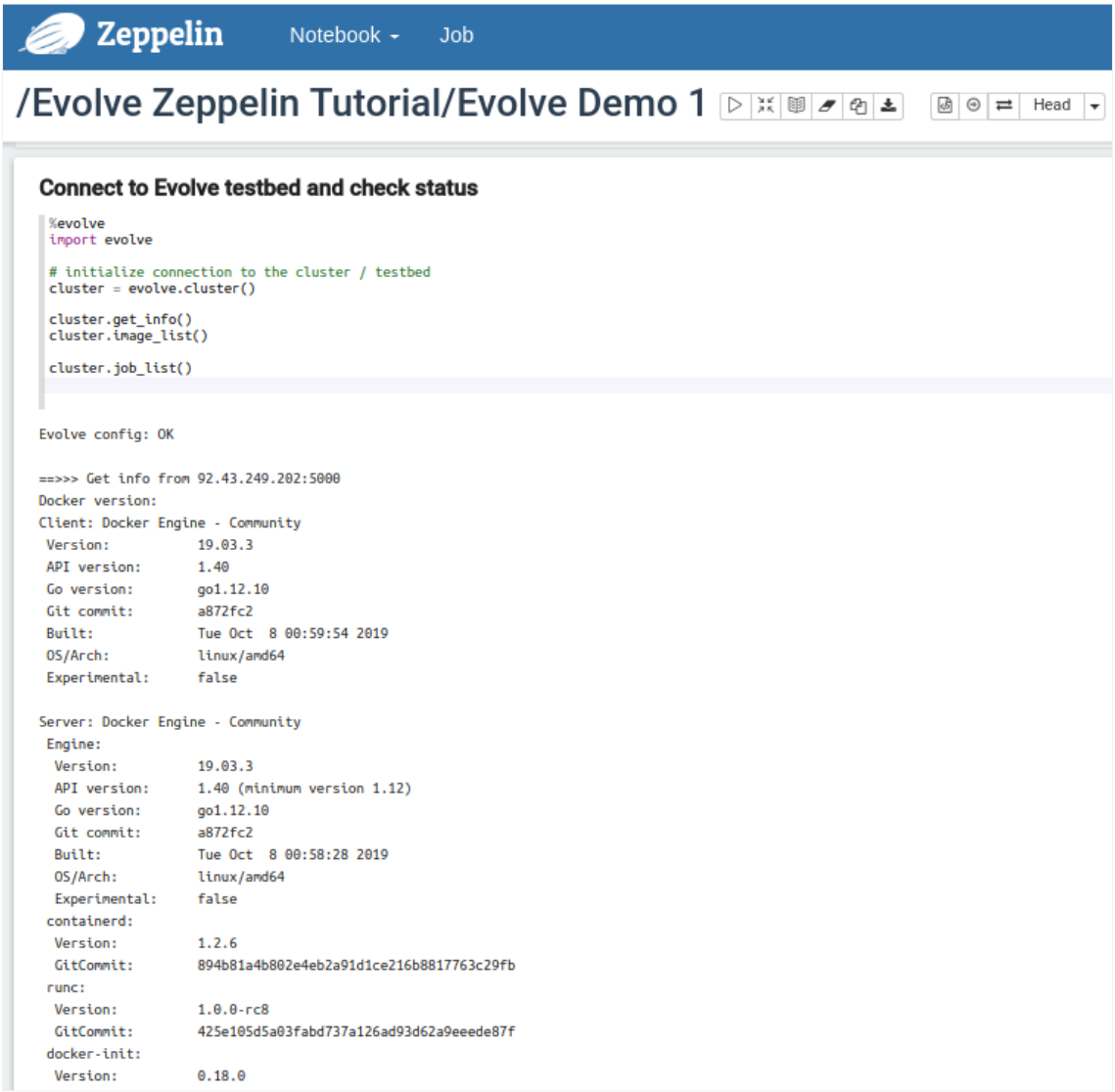
```
Status:             Failed
Message:            child 'cybeleclean-bdzcd-1214827307' failed
Created:            Tue Nov 26 18:44:03 +0000 (1 day ago)
Started:            Tue Nov 26 18:44:03 +0000 (1 day ago)
Finished:           Tue Nov 26 18:44:06 +0000 (1 day ago)
Duration:           3 seconds

STEP                 PODNAME                       DURATION  MESSAGE
 ? cybeleclean-bdzcd                                          child 'cybeleclean-
bdzcd-1214827307' failed
 +---? cleanData      cybeleclean-bdzcd-1214827307  2s        failed with exit code
1

Workflow job status: failed
failed

==>>> Check job at 92.43.249.202:5000
Name:               cybeleclean-wv89r
Namespace:          default
ServiceAccount:     default
Status:             Failed
Message:            child 'cybeleclean-wv89r-3376321090' failed
Created:            Tue Nov 26 18:43:06 +0000 (1 day ago)
Started:            Tue Nov 26 18:43:06 +0000 (1 day ago)
Finished:           Tue Nov 26 18:43:09 +0000 (1 day ago)
Duration:           3 seconds

STEP                 PODNAME                       DURATION  MESSAGE
 ? cybeleclean-wv89r                                          child 'cybeleclean-
wv89r-3376321090' failed
 +---? cleanData      cybeleclean-wv89r-3376321090  2s        failed with exit code
1

Workflow job status: failed
failed

==>>> Check job at 92.43.249.202:5000
Name:               thales-p8pc6
Namespace:          default
ServiceAccount:     default
Status:             Failed
Message:            child 'thales-p8pc6-1939801308' failed
Created:            Tue Nov 26 14:35:08 +0000 (1 day ago)
Started:            Tue Nov 26 14:35:08 +0000 (1 day ago)
Finished:           Tue Nov 26 14:35:12 +0000 (1 day ago)
Duration:           4 seconds

STEP            PODNAME                  DURATION  MESSAGE
 ? thales-p8pc6                                    child 'thales-p8pc6-1939801308'
failed
 +---? loadData  thales-p8pc6-1939801308  3s        failed with exit code 1

Workflow job status: failed
failed

==>>> Check job at 92.43.249.202:5000
Name:               cybele-nsmv5
Namespace:          default
ServiceAccount:     default
Status:             Failed
```

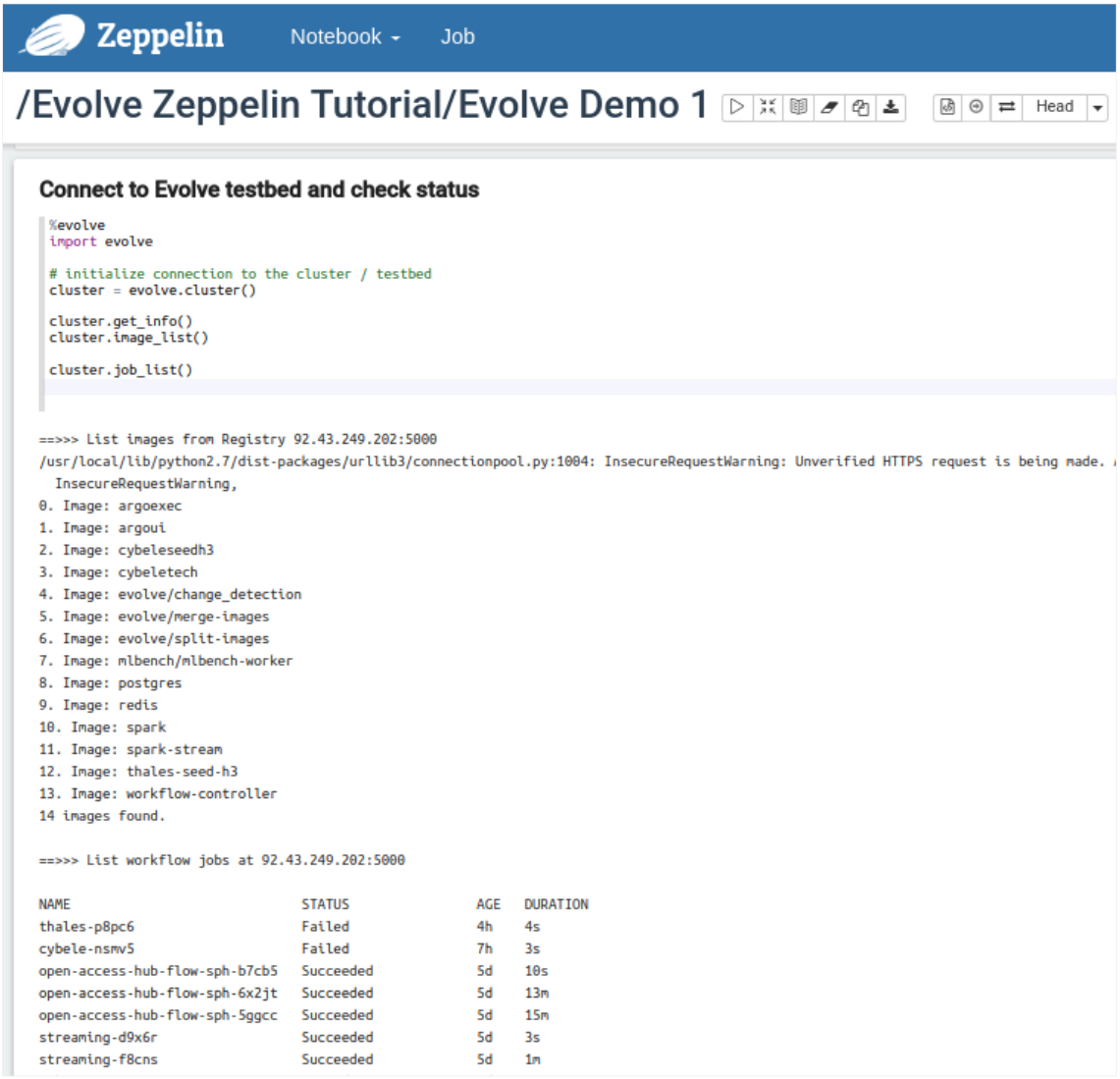
```
Message:            child 'cybele-nsmv5-2211377565' failed
Created:            Tue Nov 26 10:52:35 +0000 (2 days ago)
Started:            Tue Nov 26 10:52:35 +0000 (2 days ago)
Finished:           Tue Nov 26 10:52:38 +0000 (2 days ago)
Duration:           3 seconds

STEP            PODNAME                  DURATION  MESSAGE
 ? cybele-nsmv5                                    child 'cybele-nsmv5-2211377565'
failed
 +---? loadData  cybele-nsmv5-2211377565  2s        failed with exit code 1

Workflow job status: failed
failed

==>>> Check job at 92.43.249.202:5000
Name:               open-access-hub-flow-sph-b7cb5
Namespace:          default
ServiceAccount:     default
Status:             Succeeded
Created:            Thu Nov 21 14:23:52 +0000 (6 days ago)
Started:            Thu Nov 21 14:23:52 +0000 (6 days ago)
Finished:           Thu Nov 21 14:24:02 +0000 (6 days ago)
Duration:           10 seconds

STEP                            PODNAME
DURATION  MESSAGE
 ? open-access-hub-flow-sph-b7cb5
 +---? get                      open-access-hub-flow-sph-b7cb5-3681319521  5s
 +---? read                     open-access-hub-flow-sph-b7cb5-2142939980  3s
 +-----? recursion
when '[] != []' evaluated false

Workflow job status: succeeded
succeeded

==>>> Check job at 92.43.249.202:5000
Name:               open-access-hub-flow-sph-6x2jt
Namespace:          default
ServiceAccount:     default
Status:             Succeeded
Created:            Thu Nov 21 14:08:45 +0000 (6 days ago)
Started:            Thu Nov 21 14:08:45 +0000 (6 days ago)
Finished:           Thu Nov 21 14:21:46 +0000 (6 days ago)
Duration:           13 minutes 1 seconds

STEP
PODNAME                                   DURATION  MESSAGE
 ? open-access-hub-flow-sph-6x2jt
 +---? get
open-access-hub-flow-sph-6x2jt-2619088828  15s
 +---? read
open-access-hub-flow-sph-6x2jt-3602308099  4s
 +-·-? download-step(0:Date:2019-11-
20T16:32:01.088+00:00,Name:S1A_IW_GRDH_1SDV_20191120T163201_20191120T163226_029997_0
36C95_4D28,Uuid:15cad8aa-ad88-4dc1-95f7-bc02d8ebd6cd,Wkt:MULTIPOLYGON (((20.116304
37.792873, 23.08069 38.204361, 22.775383 39.706036, 19.747887 39.29615, 20.116304
37.792873))))    open-access-hub-flow-sph-6x2jt-2169172772  12m
 | +-? download-step(1:Date:2019-11-
20T16:31:36.089+00:00,Name:S1A_IW_GRDH_1SDV_20191120T163136_20191120T163201_029997_0
36C95_2DBD,Uuid:5a73c1ec-4f73-4fab-9a43-0976befc6529,Wkt:MULTIPOLYGON (((20.481131
```

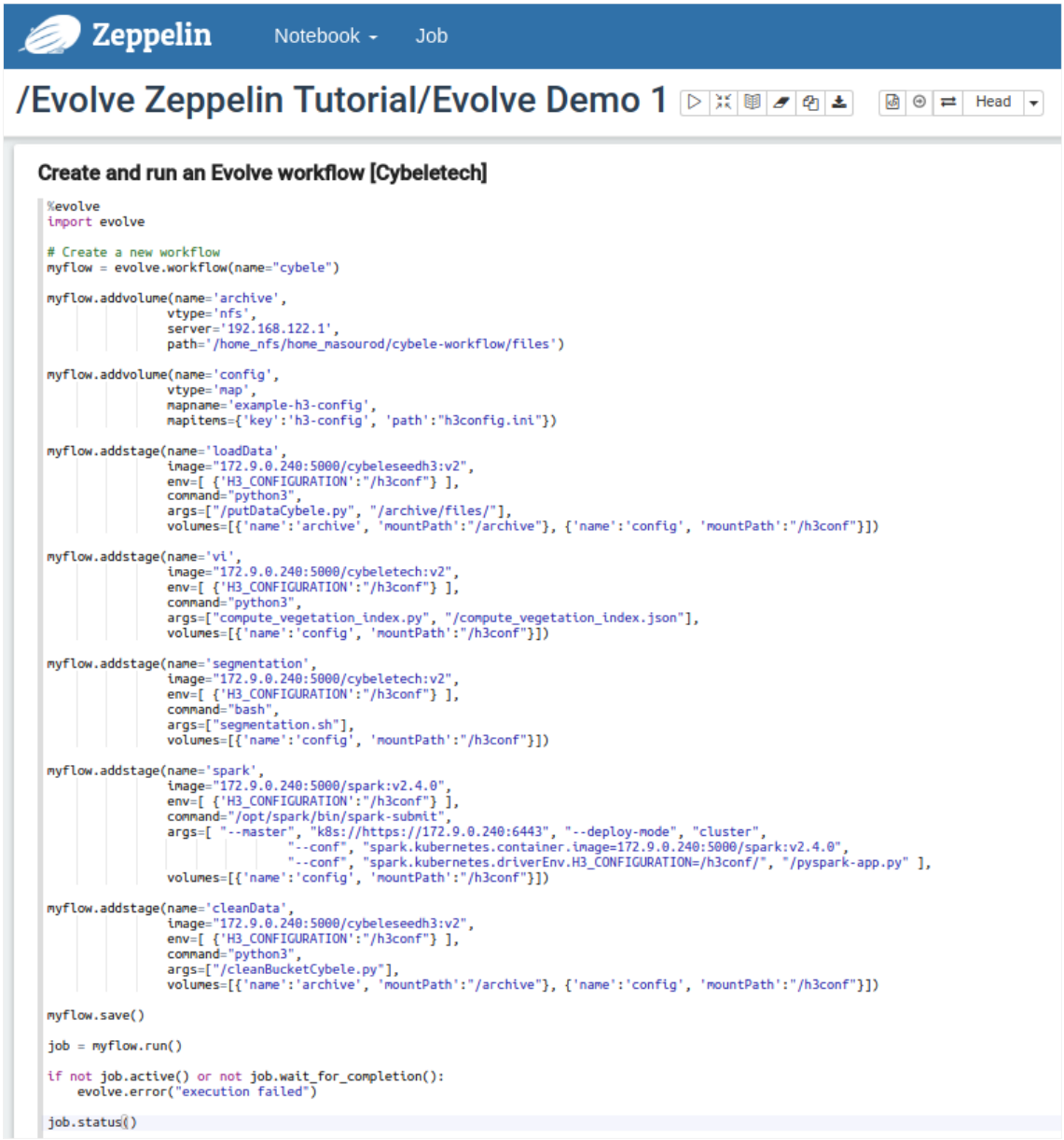
```
36.289394, 23.385889 36.702496, 23.079691 38.204144, 20.116325 37.792782, 20.481131
36.289394))))  open-access-hub-flow-sph-6x2jt-3313064707  7s
 | +-? download-step(2:Date:2019-11-
20T16:31:11.089+00:00,Name:S1A_IW_GRDH_1SDV_20191120T163111_20191120T163136_029997_0
36C95_3595,Uuid:207eeb7b-1e8a-4f10-a722-3646e58a8783,Wkt:MULTIPOLYGON (((20.839029
34.785053, 23.688183 35.199982, 23.384802 36.702263, 20.481152 36.289303, 20.839029
34.785053))))  open-access-hub-flow-sph-6x2jt-992353821   11m
 +---? recursion
     +---? get
open-access-hub-flow-sph-6x2jt-230561589   6s
     +---? read
open-access-hub-flow-sph-6x2jt-247927760   7s
     +-----? recursion
when '[] != []' evaluated false

Workflow job status: succeeded
succeeded

==>>> Check job at 92.43.249.202:5000
Name:                open-access-hub-flow-sph-5ggcc
Namespace:           default
ServiceAccount:      default
Status:              Succeeded
Created:             Thu Nov 21 13:49:36 +0000 (1 week ago)
Started:             Thu Nov 21 13:49:36 +0000 (1 week ago)
Finished:            Thu Nov 21 14:04:49 +0000 (6 days ago)
Duration:            15 minutes 13 seconds

STEP
PODNAME                                   DURATION  MESSAGE
 ? open-access-hub-flow-sph-5ggcc
 +---? get
open-access-hub-flow-sph-5ggcc-3064460371  17s
 +---? read
open-access-hub-flow-sph-5ggcc-4247699570  3s
 +-·-? download-step(0:Date:2019-11-
20T16:32:26.088+00:00,Name:S1A_IW_GRDH_1SDV_20191120T163226_20191120T163251_029997_0
36C95_8DD8,Uuid:4d2e1425-0abe-4bfb-b364-756fa78e8c1e,Wkt:MULTIPOLYGON (((19.747875
39.296242, 22.77664 39.70628, 22.46497 41.207035, 19.368393 40.798275, 19.747875
39.296242))))     open-access-hub-flow-sph-5ggcc-3312151167  14m
 | +-? download-step(1:Date:2019-11-
20T16:32:01.088+00:00,Name:S1A_IW_GRDH_1SDV_20191120T163201_20191120T163226_029997_0
36C95_4D28,Uuid:15cad8aa-ad88-4dc1-95f7-bc02d8ebd6cd,Wkt:MULTIPOLYGON (((20.116304
37.792873, 23.08069 38.204361, 22.775383 39.706036, 19.747887 39.29615, 20.116304
37.792873))))    open-access-hub-flow-sph-5ggcc-655771736   7s
 | +-? download-step(2:Date:2019-11-
20T16:31:36.089+00:00,Name:S1A_IW_GRDH_1SDV_20191120T163136_20191120T163201_029997_0
36C95_2DBD,Uuid:5a73c1ec-4f73-4fab-9a43-0976befc6529,Wkt:MULTIPOLYGON (((20.481131
36.289394, 23.385889 36.702496, 23.079691 38.204144, 20.116325 37.792782, 20.481131
36.289394))))  open-access-hub-flow-sph-5ggcc-2070450389  6s
 | +-? download-step(3:Date:2019-11-
20T16:31:11.089+00:00,Name:S1A_IW_GRDH_1SDV_20191120T163111_20191120T163136_029997_0
36C95_3595,Uuid:207eeb7b-1e8a-4f10-a722-3646e58a8783,Wkt:MULTIPOLYGON (((20.839029
34.785053, 23.688183 35.199982, 23.384802 36.702263, 20.481152 36.289303, 20.839029
34.785053))))  open-access-hub-flow-sph-5ggcc-1290489321  12s
 | +-? download-step(4:Date:2019-11-
20T04:39:11.126+00:00,Name:S1B_IW_GRDH_1SDV_20191120T043911_20191120T043936_019006_0
23DD9_4004,Uuid:57021b46-c67b-4d2e-b4ba-564b73a3af00,Wkt:MULTIPOLYGON (((22.327696
37.668228, 22.699976 39.170807, 19.731701 39.574108, 19.421041 38.072968, 22.327696
37.668228))))  open-access-hub-flow-sph-5ggcc-3041534728  14m
```

```
 +---? recursion
     +---? get
open-access-hub-flow-sph-5ggcc-1543964536  5s
     +---? read
open-access-hub-flow-sph-5ggcc-664259359   3s
     +-----? recursion
when '[] != []' evaluated false

Workflow job status: succeeded
succeeded

==>>> Check job at 92.43.249.202:5000
Name:                streaming-d9x6r
Namespace:           default
ServiceAccount:      default
Status:              Succeeded
Created:             Thu Nov 21 13:18:59 +0000 (1 week ago)
Started:             Thu Nov 21 13:18:59 +0000 (1 week ago)
Finished:            Thu Nov 21 13:19:02 +0000 (1 week ago)
Duration:            3 seconds

STEP                   PODNAME                     DURATION  MESSAGE
 ? streaming-d9x6r
 +---? antonis-stream  streaming-d9x6r-3592915388  2s

Workflow job status: succeeded
succeeded

==>>> Check job at 92.43.249.202:5000
Name:                streaming-f8cns
Namespace:           default
ServiceAccount:      default
Status:              Succeeded
Created:             Thu Nov 21 13:16:40 +0000 (1 week ago)
Started:             Thu Nov 21 13:16:40 +0000 (1 week ago)
Finished:            Thu Nov 21 13:17:43 +0000 (1 week ago)
Duration:            1 minute 3 seconds

STEP                   PODNAME                     DURATION  MESSAGE
 ? streaming-f8cns
 +---? antonis-stream  streaming-f8cns-2993996021  1m

Workflow job status: succeeded
succeeded
```

8.2. Screenshots of Zeppelin notes and output




Zeppelin
Notebook
Job
/Evolve Zeppelin Tutorial/Evolve Demo 1
Head
Connect to Evolve testbed and check status
%evolve
import evolve
# initialize connection to the cluster / testbed
cluster = evolve.cluster()
cluster.get_info()
cluster.image_list()
cluster.job_list()
==>> List images from Registry 92.43.249.202:5000
/usr/local/lib/python2.7/dist-packages/urllib3/connectionpool.py:1004: InsecureRequestWarning: Unverified HTTPS request is being made.
InsecureRequestWarning,
0. Image: argoexec
1. Image: argoui
2. Image: cybeleseedh3
3. Image: cybeletech
4. Image: evolve/change_detection
5. Image: evolve/merge-images
6. Image: evolve/split-images
7. Image: mlbench/mlbench-worker
8. Image: postgres
9. Image: redis
10. Image: spark
11. Image: spark-stream
12. Image: thales-seed-h3
13. Image: workflow-controller
14 images found.
==>> List workflow jobs at 92.43.249.202:5000
NAME STATUS AGE DURATION
thales-p8pc6 Failed 4h 4s
cybele-nsmv5 Failed 7h 3s
open-access-hub-flow-sph-b7cb5 Succeeded 5d 10s
open-access-hub-flow-sph-6x2jt Succeeded 5d 13m
open-access-hub-flow-sph-5ggcc Succeeded 5d 15m
streaming-d9x6r Succeeded 5d 3s
streaming-f8cns Succeeded 5d 1m


Zeppelin
Notebook
Job
/Evolve Zeppelin Tutorial/Evolve Demo 1
Head
Create and run an Evolve workflow [Cybeletech]
%evolve
import evolve
# Create a new workflow
myflow = evolve.workflow(name="cybele")
myflow.addvolume(name='archive',
vtype='nfs',
server='192.168.122.1',
path='/home_nfs/home_masourod/cybele-workflow/files')
myflow.addvolume(name='config',
vtype='map',
mapname='example-h3-config',
mapitems={'key':'h3-config', 'path':"h3config.ini"})
myflow.addstage(name='loadData',
image="172.9.0.240:5000/cybeleseedh3:v2",
env=[{'H3_CONFIGURATION':"/h3conf"}],
command="python3",
args=["/putDataCybele.py", "/archive/files/"],
volumes=[{'name':'archive', 'mountPath':"/archive"}, {'name':'config', 'mountPath':"/h3conf"}])
myflow.addstage(name='vi',
image="172.9.0.240:5000/cybeletech:v2",
env=[{'H3_CONFIGURATION':"/h3conf"}],
command="python3",
args=["compute_vegetation_index.py", "/compute_vegetation_index.json"],
volumes=[{'name':'config', 'mountPath':"/h3conf"}])
myflow.addstage(name='segmentation',
image="172.9.0.240:5000/cybeletech:v2",
env=[{'H3_CONFIGURATION':"/h3conf"}],
command="bash",
args=["segmentation.sh"],
volumes=[{'name':'config', 'mountPath':"/h3conf"}])
myflow.addstage(name='spark',
image="172.9.0.240:5000/spark:v2.4.0",
env=[{'H3_CONFIGURATION':"/h3conf"}],
command="/opt/spark/bin/spark-submit",
args=["--master", "k8s://https://172.9.0.240:6443", "--deploy-mode", "cluster",
"--conf", "spark.kubernetes.container.image=172.9.0.240:5000/spark:v2.4.0",
"--conf", "spark.kubernetes.driverEnv.H3_CONFIGURATION=/h3conf/", "/pyspark-app.py"],
volumes=[{'name':'config', 'mountPath':"/h3conf"}])
myflow.addstage(name='cleanData',
image="172.9.0.240:5000/cybeleseedh3:v2",
env=[{'H3_CONFIGURATION':"/h3conf"}],
command="python3",
args=["/cleanBucketCybele.py"],
volumes=[{'name':'archive', 'mountPath':"/archive"}, {'name':'config', 'mountPath':"/h3conf"}])
myflow.save()
job = myflow.run()
if not job.active() or not job.wait_for_completion():
evolve.error("execution failed")
job.status()

```
Evolve config: OK
Workflow init: OK
Workflow save: OK
Starting execution of workflow 'cybele'
Name:                cybele-7gzjs
Namespace:           default
ServiceAccount:      default
Status:              Pending
Created:             Tue Nov 26 11:45:46 +0000 (5 minutes from now)

Job cybele-7gzjs init: OK
Waiting for completion of job 'cybele-7gzjs'
Name:                cybele-7gzjs
Namespace:           default
ServiceAccount:      default
Status:              Running
Created:             Tue Nov 26 11:45:46 +0000 (5 minutes from now)
Started:             Tue Nov 26 11:45:46 +0000 (5 minutes from now)
Duration:            5 minutes

STEP             PODNAME                  DURATION  MESSAGE
 ● cybele-7gzjs
 └---◷ loadData  cybele-7gzjs-2434091125  5m

[0 sec] Workflow job status: running
Name:                cybele-7gzjs
Namespace:           default
ServiceAccount:      default
Status:              Failed
Message:             child 'cybele-7gzjs-2434091125' failed
Created:             Tue Nov 26 11:45:46 +0000 (4 minutes from now)
Started:             Tue Nov 26 11:45:46 +0000 (4 minutes from now)
Finished:            Tue Nov 26 11:45:49 +0000 (4 minutes from now)
Duration:            3 seconds

STEP             PODNAME                  DURATION  MESSAGE
 ✖ cybele-7gzjs                                    child 'cybele-7gzjs-2434091125' failed
 └---✖ loadData  cybele-7gzjs-2434091125  1s        failed with exit code 1

[20 sec] Workflow job status: failed
ERROR: Workflow job FAILED after 3 seconds with message child 'cybele-7gzjs-2434091125' failed
EVOLVE EXCEPTION: execution failed
```

evolve

Leading the Big Data Revolution

@evolve-h2020

@evolve_h2020

info@evolve-h2020.eu

www.evolve-h2020.eu

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825061