

Title: MODACLOUDS Integration Report – Final version

Authors: Marcos Almeida (Softeam), Antonin Abherve (Softeam), Weikun Wang (Imperial), Pooyan Jamshidi (Imperial), Nicolas Ferry (SINTEF), Gabriel Iuhasz (IeAT), Daniel Pop (IeAT), Michele Ciavotta (Polimi), Marco Miglierina (Polimi), Marco Scavuzzo (Polimi), Giovanni Paolo Gibilisco (Polimi), Jacek Dominiak (CA), Román Sosa González (ATOS)

Editor: Román Sosa González (ATOS)

Reviewers: Marcos Almeida (Softeam), Craig Sheridan (Flexiant)

Identifier: Deliverable # D3.4.2

Nature: Report

Version: 1.2

Date: 28/09/2015

Status: Final

Diss. level: Public

Executive Summary

This deliverable presents the activities performed for the integration of the different MODAClouds software components and describes the procedures followed to obtain integrated platforms for the Design Time environment and the Runtime environment.

Members of the MODAClouds consortium:

Politecnico di Milano	Italy
Stiftelsen Sintef	Norway
Institutul E-Austria Timisoara	Romania
Imperial College of Science, Technology and Medicine	United Kingdom
SOFTEAM	France
Siemens srl	Romania
BOC Information Systems GMBH	Austria
Flexiant Limited	United Kingdom
ATOS Spain S.A.	Spain
CA Technologies Development Spain S.A.	Spain

Published MODAClouds documents

These documents are all available from the project website located at <http://www.modaclouds.eu/>

Contents

1	INTRODUCTION	5
1.1	CONTEXT AND OBJECTIVES.....	5
1.2	STRUCTURE OF THE DOCUMENT.....	5
2	INTERNAL INTEGRATION TASKS	7
2.1	VENUES 4CLOUDS – CREATOR 4CLOUDS	8
2.2	SLA TOOL – CREATOR 4CLOUDS.....	9
2.3	SPACE ^{DEV} 4CLOUDS – FUNCTIONAL MODELLING TOOL.....	9
2.4	FEEDBACK LOOP – CREATOR 4CLOUDS.....	9
2.5	AUTO-SCALING REASONER – CREATOR 4CLOUDS.....	9
2.6	CREATOR 4CLOUDS – CLOUDML.....	10
2.7	CREATOR 4CLOUDS – MODELS@RUNTIME.....	10
2.8	BATCH ENGINE – FEEDBACK LOOP	11
2.9	FEEDBACK LOOP – TOWER 4CLOUDS HISTORY DB.....	11
2.10	TOWER 4CLOUDS HISTORY DB – TOWER 4CLOUDS MANAGER	11
2.11	SLA TOOL – MODELS@RUNTIME	11
2.12	SLA TOOL – RUNTIME GUI	12
2.13	SLA TOOL – TOWER 4CLOUDS	13
2.14	RUNTIME GUI – MODELS@RUNTIME	14
2.15	TOWER 4CLOUDS – MODELS@RUNTIME	15
2.16	MODELS@RUNTIME – AUTO-SCALING REASONER	16
2.17	MODELS@RUNTIME – DATA MIGRATION	16
2.18	MODELS@RUNTIME – LOAD BALANCER CONTROLLER.....	16
2.19	LOAD BALANCER CONTROLLER – ARTIFACT REPOSITORY.....	17
2.20	LOAD BALANCER REASONER – LOAD BALANCER CONTROLLER	17
2.21	LOAD BALANCER REASONER – OBJECT STORE	17
2.22	MODELS@RUNTIME – LOAD BALANCER REASONER	17
3	EXTERNAL INTEGRATION TASKS	19

3.1	VENUES 4CLOUDS.....	19
3.2	CLOUDML.....	20
4	INTEGRATION TESTS	23
5	DESIGN TIME INTEGRATED PLATFORM.....	34
6	RUNTIME INTEGRATED PLATFORM	35
7	CONCLUSION	36
8	REFERENCES	37
	APPENDICES	38
A.1.	MODACLOUDS RPM PACKAGES.....	39
A.2	BUILDING THE PACKAGE.JSON DESCRIPTOR.....	46
A.3	THE WRAPPER SCRIPT	55

1 Introduction

1.1 Context and objectives

This document complements the work done in the deliverable D3.5.3 “MODAClouds integrated solution – Final version” [8], also released at M36. It reports on the progress made in the project integration activities since M24 and is a continuation of the work described in D3.4.1 “MODAClouds integration report – Initial version” [1].

The MODAClouds project has implemented several software components to address the objectives of WP2 to WP5. These components need to communicate between them in a Service Oriented Architecture fashion, so they must agree on interfaces and define a set of tests that ensure the right integration between the components. This integration process is called internal integration in the document. On the other hand, the integration and tests are also needed in the case of components that use services outside of MODAClouds (e.g. cloud service providers). In the document, we refer to this as external integration.

Moreover, there is a need to obtain a distribution of the platform that is easy to install and run. The consortium has worked toward this objective in the last year with the result of two distributions: the Design Time integrated platform and the Runtime integrated platform.

The deliverable continues the new naming scheme of MODAClouds components. The repackaged service offerings of the integrated solution are detailed in D3.5.3 “MODAClouds integrated solution – Final version” [8], and can be summarised as:

- **Creator 4Clouds** consisting of Functional Modelling Tool, Space^{Dev} 4Clouds, LINE, CloudML 4Clouds, Resource Repository.
- **Venues 4Clouds** consisting of Decision Support System.
- **Energizer 4Clouds** is divided into three sub assets:
 - **Tower4Clouds** consisting of Monitoring Manager, DDA, Data Collector Factory, QoS Models, Metrics Observer, Metrics Explorer, Knowledge Base, Data Collectors, Matlab/Weka SDA.
 - **Space^{Ops} 4Clouds** consisting of Self-Adaptation Reasoner, Self-Adaptation Stress Tester, Load-Balancer Reasoner, Cloud-Bursting.
 - **ADDapters 4Clouds** consisting of Data Migration and Synchronization, Load-Balancer Controller, Object Store, Artifact Repository, Batch Engine, mOS Image Builder, mOS Package Builder.

1.2 Structure of the document

This document is structured as follows:

- Section 2 describes the internal integration challenges faced since M24 and how the consortium addressed them, while the Section 3 describes the external integration challenges addressed in the same period.

- Section 4 enumerates the integration tests that have been implemented in order to check the successful integration of the different components.
- Section 5 describes the work carried out in the development of an integrated platform for the Design Time environment.
- Section 6 describes the procedure to obtain RPM packages of the runtime components in order to have an easy to install and update Runtime Integrated Platform.
- Section 7 explains the conclusions of the document.

2 Internal integration tasks

The Section 2 reports on the integration issues tackled by the consortium since M24, and are detailed in the following sub-sections. Here we mainly refer to the internal integration issues due to the communication and exchanging of information between the MODAClouds components.

In order to manage and maintain a list of issues, the consortium agreed the use of the issue tracker present in the MODAClouds Redmine portal¹.

The main objectives of the use of the issue tracker were to report any bugs found by the case study owners in the functionalities offered by the MODAClouds components, and to help in the process of obtaining an integrated platform - described in sections 5 and 6.

At the time of writing this deliverable, the following statistics can be extracted from the current issues:

- Number of issues: 23
- Type of issue:
 - Packaging: 6
 - Installation: 3
 - Bug/New feature in component: 3
 - Interaction between components: 1
 - Wrong component integration with the platform: 10
- Open issues (unsolved or in feedback status): 6
- Unsolved issues: 3 (do not provide any major problem).

The Figure 1 shows an excerpt of the filed issues.

¹ <https://dev.modaclouds.eu/redmine/projects/modaclouds>

#	Project	Tracker	Parent task	Status	Priority	Subject	Assignee	Updated	Category
142	MODAClouds	Bug		Closed	Urgent	Failed to acquire error in FG analyser	Weikun Wang	10/08/2015 04:54 pm	
141	MODAClouds	Bug		Closed	Urgent	Venus4clouds computation of proposed providers doesn't work	Roman Sosa	14/08/2015 03:33 pm	
140	MODAClouds	Bug		In Progress	Normal	load-balancer-reasoner error in wrapper script	Ciprian Craciun	26/08/2015 11:22 am	load-balancer-reasoner
139	MODAClouds	Bug		Feedback	Normal	RDF History DB failed to start	Ciprian Craciun	06/08/2015 07:01 pm	Tower4Clouds
138	MODAClouds	Bug		Rejected	Urgent	Object store doesn't work	Ciprian Craciun	30/07/2015 12:43 pm	object-store
137	MODAClouds	Bug		In Progress	Low	Installation failed	Roman Sosa	30/07/2015 01:00 pm	Installation
136	MODAClouds	Bug		In Progress	Normal	load-balancer-reasoner: wrong packaging	Ciprian Craciun	26/08/2015 11:34 am	packaging
135	MODAClouds	Bug		Resolved	Normal	sda-matlab: wrong packaging	Ciprian Craciun	30/07/2015 12:58 pm	packaging
134	MODAClouds	Bug		Closed	Normal	object-store: service dies	Ciprian Craciun	30/07/2015 12:46 pm	object-store
133	MODAClouds	Bug		Resolved	Normal	fg-analyzer: component does not start	Weikun Wang	28/07/2015 10:18 pm	fg-analyzer
132	MODAClouds	Bug		Closed	Normal	fg-local-db: Component does not start	Weikun Wang	28/07/2015 02:27 pm	fg-local-db
131	MODAClouds	Bug		Closed	Normal	sda-matlab: Component does not start	Weikun Wang	28/07/2015 02:49 pm	monitoring-sda-matlab
130	MODAClouds	Bug		Closed	Normal	load-balancer-reasoner: wrapper script does not start	Weikun Wang	28/07/2015 04:16 pm	load-balancer-reasoner
129	MODAClouds	Bug		Closed	Normal	sda-weka: wrapper script needs full review	Weikun Wang	28/07/2015 03:10 pm	monitoring-sda-weka
128	MODAClouds	Bug		Closed	Normal	m@R: T4C_manager and LB_controller vars not used	Nicolas Ferry	25/08/2015 05:22 pm	CloudML
127	MODAClouds	Bug		Feedback	Normal	Cannot install modaclouds-platform-services	Marco Miglierina	25/08/2015 05:15 pm	packaging
126	MODAClouds	Bug		Closed	Normal	m@R wrong packaging	Nicolas Ferry	28/07/2015 12:02 pm	packaging
125	MODAClouds	Bug		Closed	Normal	Interaction between Modelio and CloudML	Antonin Abhervé	25/08/2015 05:18 pm	

Figure 1: MODAClouds issue tracker

The upcoming subsections report about what we consider the most critical internal integration tasks that were faced so far and how they were solved.

2.1 Venues 4Clouds – Creator 4Clouds

Context: Creator 4Clouds tool allows the user to specify the model of the application on the design and run time. The Venues 4Clouds UI tool is looking for suitable services during the design time, hence the need of sharing the selected components of the architecture.

What was done: Implemented the export mechanism for Creator 4Clouds which allows storing the architecture model in the XML file. Implemented appropriate read model on the Venues 4Clouds tool with validation of the architectural description taken from the Space^{Dev} 4Clouds model definitions.

2.2 SLA Tool – Creator 4Clouds

Context: The Mediator subcomponent of the SLA Tool is updated to support the Application Provider – Cloud Provider layer. This issue required minor changes in SLA Mediator interface: the Functionality2tiers file is not needed any more and a new parameter containing Space^{Dev} 4Clouds rules has been introduced.

What was done: Changes implemented, empty rule set sent as Space^{Dev} 4Clouds rules.

Context: Creator 4Clouds is updated to support SLA Mediator 1.0. The SLA Mediator needed more Space^{Dev} Clouds files as input: the allocation model, the system model, the resource environment, the resource container extension and the functionalities2tiers file.

What was done: Creator 4Clouds now produces these files as output.

2.3 Space^{Dev} 4Clouds – Functional Modelling Tool

Context: Export of functionalities2tiers became necessary for Space^{Dev} 4Clouds.

What was done: Export added

Context: The integration between the Functional Modelling Tool and Space^{Dev} 4Clouds occurs through a process of file generation and exchange. The Functional Modelling Tool converts its own internal model in extended PCM format, which is the Space^{Dev} 4Clouds input format. The latter generates output file that are aspirated within the Functional Modelling Tool.

What was done: The generation of the input files has been validated.

2.4 Feedback Loop – Creator 4Clouds

Context: The configuration file of Feedback Loop component was not supported by Creator 4Clouds

What was done: Format and contents of configuration file defined and implemented.

Context: Communication protocol and lifecycle of both components was not defined

What was done: Protocol and lifecycle defined. We will use the Object store, made available by MODAClouds runtime, as communication channel of configuration and data files; and of updates coming from runtime.

2.5 Auto-scaling Reasoner – Creator 4Clouds

Context: The integration between Creator 4Clouds and the Auto-scaling Reasoner is implemented by means of generating and delivering to the runtime platform a set of input files containing monitoring rules and information about the predicted application behavior. Such pieces of information are generated by Space^{Dev} 4Clouds and conveyed by Creator 4Clouds.

What was done: Recently, the Functional Modelling Tool has been modified in order to export a new file that Space^{Dev} 4Clouds needs to generate the input files required by the Auto-scaling Reasoner. Moreover, the Functional Modelling Tool now retrieves the outcomes of Space^{Dev} 4Clouds and handles the communication with the runtime platform.

2.6 Creator 4Clouds – CloudML

Context: The Creator 4Clouds tool provides the functional, operational and data modelling environments as well as some modules enabling the analysis of non-functional characteristics of a multi-cloud system. Among other things, it provides specific services for modelling the deployment of an application in a cloud-computing environment. Since the previous integration report, the CloudML library has evolved to finalize its integration with the runtime platform.

What was done: The version 2.0 of CloudML library, which supports integration with the runtime platform, has been integrated to Creator 4Clouds. The deployment of the cloud architecture is now performed at runtime, the CloudML library ensuring the transmission of the deployment model between Creator 4Clouds and Models@Runtime.

The Creator 4Clouds generator tools that produce deployment files from models also evolved to support Puppet based deployment process.

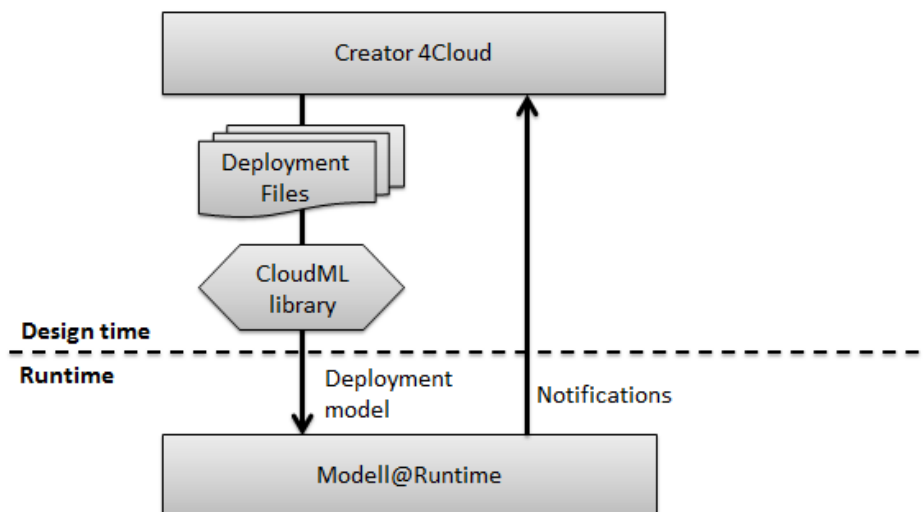


Figure 2: Creator 4Clouds

2.7 Creator 4Clouds – Models@Runtime

Context: In order to integrate the MODAClouds runtime and design-time platforms, the deployment specified using CloudML and generated by Creator 4Clouds have to be enacted by the Models@Runtime engine. The various runtime components can thus in turn interact with the Models@Runtime engine in order to monitor and adapt the deployment and provisioning of an application.

What was done: In order to ease the integration between these two components, the CloudML facade has been extended to seamlessly interact either with CloudML locally or with a remote Models@Runtime engine. As a result, when using CloudML as a library, one can either tune a deployment model or trigger high level commands (e.g., deploy, scale out) programmatically using directly the library or a remote Models@Runtime engine through the exact same interface.

In order to perform a deployment, the Models@Runtime engine requires user specific information such as cloud provider credentials or SSH key to access the provisioned VMs, which are typically stored locally. The facade has been extended so that these data are seamlessly provided to the remote Models@Runtime engine.

In order to retrieve feedback about the status of an ongoing provisioning and deployment process performed by a remote Models@Runtime engine, the facade registers to the notification mechanism offered by the Models@Runtime engine.

2.8 Batch Engine – Feedback Loop

Context: The Feedback Loop may analyse a large amount of data which could be time consuming, therefore requires the ability for parallel executions.

What was done: The Batch Engine exposes Rest API for the Feedback Loop to submit jobs to Condor cluster. The Feedback Loop is now able to execute jobs on the condor cluster with the API and executes locally if the Rest call fails.

2.9 Feedback Loop – Tower 4Clouds History DB

Context: The Feedback Loop requires monitoring data saved at runtime to execute the analysis. The Tower 4Clouds History DB stores all the runtime data. Therefore, the Feedback Loop needs to access the data from the History DB and save them into its own local DB.

What was done: The Feedback Loop now connects the Tower 4Clouds History DB with the REST API exposed by the History DB. The Feedback Loop will periodically call the API and obtain data relevant for its analysis.

2.10 Tower 4Clouds History DB – Tower 4Clouds Manager

Context: Monitoring data and model history is required to be stored for off-line analysis.

What was done: Tower 4Clouds RDF History DB component was implemented as a metric observer and can be attached to observable metrics using either the Tower 4Clouds Manager API or the Tower 4Clouds Manager Webapp. The specified metrics will be stored in an Apache Fuseki RDF datastore together with the history of the Model, whose deltas are recorded by the Manager whenever an update occurs.

2.11 SLA Tool – Models@Runtime

Context: The Models@Runtime engine is responsible for triggering the enforcement of an SLA agreement. In order to perform such operation, the Models@Runtime engine has to be aware of the endpoint of the SLA Tool and of the id of the agreement to be enforced.

What was done: Both the endpoint and the agreement id are included as part of the CloudML deployment model and, this way, are provided to the Models@Runtime engine. In particular, they are modelled using the concept of properties attached to the deployment model. Once the deployment completed, the engine exploits these information to perform the appropriate REST call to enforce the SLA agreement.

```
"properties" : [{  
  
    "eClass" : "net.cloudml.core:Property",
```

```
        "name" : "sla_url",
        "value" : <URL>
    }, {
        "eClass" : "net.cloudml.core:Property",
        "name" : "agreement_id",
        "value" : <UUID>
    }
],
```

Listing 1: SLA related properties in a deployment model

2.12 SLA Tool – Runtime GUI

Context: The Runtime GUI is responsible for displaying information about SLA violations. This information can be obtained from the SLA manager. However, in order to retrieve details about the violations of a specific component of the cloud application, the Runtime GUI has to be aware of the agreement id attached to this component.

What was done: Agreement ids are attached to component instances within a CloudML deployment model and are thus provided to the Runtime GUI. Because the various components that compose the system can have different agreements, an agreement id can be attached to each of them and are thus modeled in CloudML as properties (see Listing 2).

```
"properties" : [{
    "eClass" : "net.cloudml.core:Property",
    "name" : "agreement_id",
    "value" : "agreement-b"
}]
```

Listing 2: Example of SLA properties in a deployment model

On the basis of the component and agreement ids, the Runtime GUI is able to perform the appropriate asynchronous REST calls to retrieve from the SLA Tool:

- The number of SLA violations
- The number of SLA violations that occurred within the day
- The violations details

Within the graphical user interface, these calls can be triggered by the user by clicking on the component of interest and then on the load buttons as depicted in Figure 3.

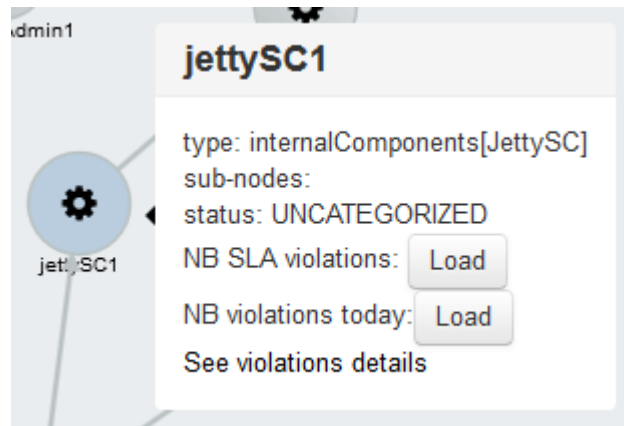


Figure 3: Integration Runtime GUI and SLA Tool

2.13 SLA Tool – Tower 4Clouds

Context: The Tower 4Clouds API for subscribing an observer has slightly changed. Tower 4Clouds now offers several data formats to send the data to the observers, and thus the subscription request is a JSON structure with several options, rather than the plain URL used previously.

What was done: The code in the SLA Tool that handled the subscription to the Tower 4Clouds Manager changed to support the change in the API. The only needed modification was to send a JSON body specifying the endpoint of the SLA Tool as the `callbackUrl` field and the required data format, as shown in Listing 3:

```
{
  "format": "RDF/JSON",
  "callbackUrl": "http://ENDPOINT_IP:ENDPOINT_PORT/metrics/agreement-id"
}
```

Listing 3: SLA subscription to Tower4Clouds

Context: The data sent in RDF/JSON format from the Tower 4Clouds manager to observers has slightly changed, and the namespaces prefixing each field have been renamed from `http://www.modaclouds.eu/rdfs/1.0/monitoringdata` to `http://www.modaclouds.eu/model`, like shown in Listing 4.

```
{
  "_:-1c2601ea:14e24ce91fd:-7c83" : {
    "http://www.modaclouds.eu/model#timestamp" : [ {
      ...
    }
  ] ,
}
```

```
"http://www.modaclouds.eu/model#value" : [ {  
    ...  
  }  
  ] ,  
  ...  
}
```

Listing 4: Tower 4Clouds output metric

What was done: The observer has been modified to interpret the data in the current namespace.

2.14 Runtime GUI – Models@Runtime

Context: The Runtime GUI is responsible for: (i) providing feedback about the status of the deployment, (ii) providing feedback about the status of the running system, (iii) providing access to high level operation commands such as deploy, scale, burst. All the status information can be obtained from the Models@Runtime engine. Similarly, the high level commands are enacted by the Models@Runtime engine.

What was done: The Models@Runtime engine exposes a WebSocket interface which provides a remote access to the high level commands offered by the Models@Runtime engine. In addition, a notification mechanism has been created and provides external client with the ability to register for notifications. A notification is send every time the status of a component in the deployment model changes. The Runtime GUI, once connected to a Models@Runtime engine, registers all notifications. As a result, it can provide users with details on the status of the deployment and of the running system. In addition, it implements a WebSocket client to trigger deployment actions as depicted in Figure 4.

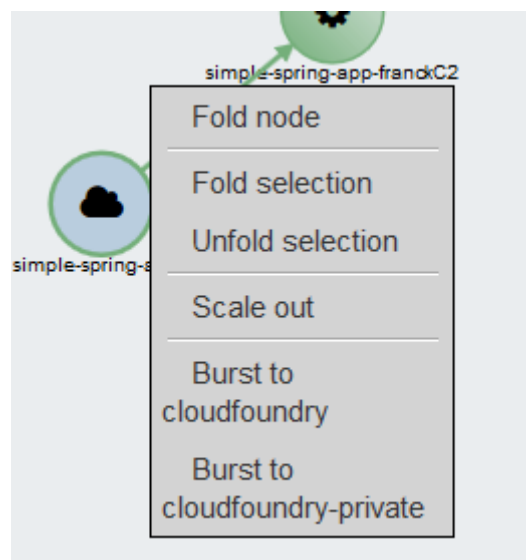


Figure 4: High level commands accessible from the Runtime GUI

Details about the WebSocket interface are available in: <https://github.com/SINTEF-9012/cloudml/tree/master/ui/websocket>

2.15 Tower 4Clouds – Models@Runtime

Context: The Models@Runtime engine is responsible for keeping the Tower 4Clouds platform informed about the status of the deployment. In particular, the Models@Runtime has to provide information about the component to be monitored (e.g., type, id, host).

What was done: On one hand, data collectors are responsible for providing the Tower 4Clouds Manager with details about the cloud resources and software components to be monitored. On the other hand, the Models@Runtime engine is responsible for providing these data to the data collectors as well as the endpoint of the monitoring manager. The data collectors are deployed using the Models@Runtime engine and thus included in the deployment models. The details about the deployment are given to the data collectors using environment variables. The Models@Runtime engine has thus been extended with the ability to export environment variable. This operation is performed every time a VM is provisioned and before the software components are installed. The environment variables to be exported are specified in the deployment models as properties (see Listing 5).

```
"properties" : [{
  "eClass" : "net.cloudml.core:Property",
  "name" : "env:MODACLOUDS_TOWER4CLOUDS_VM_ID",
  "value" : "${this.host.id}"
}, {
  "eClass" : "net.cloudml.core:Property",
  "name" : "env:MODACLOUDS_TOWER4CLOUDS_VM_TYPE",
  "value" : "${this.host.type.name}"
}, {
  "eClass" : "net.cloudml.core:Property",
  "name" : "env:MODACLOUDS_TOWER4CLOUDS_CLOUD_PROVIDER_ID",
  "value" : "${this.provider.id}"
}, {
  "eClass" : "net.cloudml.core:Property",
  "name" : "env:MODACLOUDS_TOWER4CLOUDS_CLOUD_PROVIDER_TYPE",
  "value" : "IaaS"
}],
```

Listing 5: Exporting environment variables to configure data collectors

2.16 Models@Runtime – Auto-scaling Reasoner

Context: The Auto-scaling Reasoner exploits the Models@Runtime engine in order to enact adaptation actions (i.e., scaling, bursting).

What was done: The Models@Runtime engine exposes a set of high level commands that can be exploited to adapt the deployment of a cloud-based application. In particular, the engine offers the following commands:

- Start/stop one to n components in parallel. These commands can be used to improve the performances of the scaling operation since it avoids provisioning new cloud resources.
- Scale an instance of VM n times in parallel.
- Burst a VM to a new cloud provider.

These commands can be triggered remotely by exploiting the Websocket interface exposed by the Models@Runtime engine.

2.17 Models@Runtime – Data migration

Context: The Models@Runtime engine is responsible for triggering the data migration.

What was done: Data Migration tool exposes a set of REST APIs that allows an external application to start the data migration task from a source database to one or more target databases. Once the deployment completed, the Models@Runtime engine exploits this interface to initiate the migration (offline or online).

2.18 Models@Runtime – Load Balancer Controller

Context: The MODAClouds Load Balancer is one of the runtime components that are used and thus considered as part of the running cloud application. As a result it has to be managed by the Models@Runtime engine, which is responsible for the provisioning, deployment and adaptation of multi-cloud applications.

What was done: The Load Balancer has been integrated with the Models@Runtime engine. Within a deployment model, the Load Balancer is modelled as a component. This way, the necessary information is provided to the Models@Runtime engine. The latter can in turn exploit the API exposed by the Load Balancer during the deployment to:

- configure the Load Balancer with the necessary information (i.e., gateway, protocol),
- start the Load Balancer,
- and manage the pool of resources behind the Load Balancer on the basis of the relationships, defined within the deployment model, between Models@Runtime components and the Load Balancer.

In addition, this pool is dynamically managed by the Models@Runtime engine when performing scaling and bursting operations.

2.19 Load Balancer Controller – Artifact Repository

Context: The MODAClouds Load Balancer Controller and Artifact Repository are both supporting services part of the runtime components. In particular the Artifact Repository is used to store artifacts produced or required by any component from the runtime.

What was done: The integration between the Load Balancer Controller and Artifact Repository is important, as it stores the current state of the Load Balancer inside the Artifact Repository. It is also possible to store several state versions. The exposed functionality is as follows:

- LB REST API supports the querying of the current stored states inside the Artifact Repository
- From the LB REST API it is possible to create new LB state versions
- It is also possible to delete old unwanted states

The LB States take the form of an SQLite database, which is exported as is into the Artifact Repository. This database contains current configured endpoints, gateways, generated configuration and stored security keys.

2.20 Load Balancer Reasoner – Load Balancer Controller

Context: The Load Balancer Reasoner needs to access the Load Balancer Controller so that the reasoner can check if the backend VMs are running and change the weights of the VMs of the weighted round robin policy.

What was done: The Load Balancer Reasoner is able to interact with the Load Balancer Controller by the Load Balancer Controller REST API to start the Load Balancer and change the various configurations of it.

2.21 Load Balancer Reasoner – Object Store

Context: The Load Balancer Reasoner needs some preset configuration to start. However, during development the configuration should be passed to the Load Balancer Reasoner.

What was done: The configurations are now stored in the Object Store, exposed by environment variables for the Load Balancer Reasoner to access it with HTTP requests.

2.22 Models@Runtime – Load Balancer Reasoner

Context: Models@Runtime engine must be able to configure Load Balancers running on different cloud providers during the deployment process on the basis of their relationships with other CloudML components.

What was done: The Load Balancer Reasoner has been integrated with the Models@Runtime engine. Within a deployment model, the layered Load Balancer is modelled as a dependency between components. This way, the necessary information is provided to the Models@Runtime engine. The Models@Runtime engine uses the API exposed by the Load Balancer Controller during the deployment to:

- configure the Load Balancer with the necessary information (i.e., gateway, protocol),
- add the appropriate IPs of the backend machines created by the engine to the configuration file of the local Load Balancer,
- add the appropriate IPs of the local Load Balancers to the configuration file of the Global Load Balancer,
- start the Global Load Balancer and Local Load Balancers,
- manage the pool of resources behind the local Load Balancers on the basis of the relationships, defined within the deployment model, between CloudML components and the Load Balancer.

In addition, this pool is dynamically managed by the Models@Runtime engine when performing scaling and bursting operations.

3 External integration tasks

This section reports on the external integration issues faced by the consortium since M24. These external integration issues are due to the communication and exchanging of information between MODAClouds components and services outside of MODAClouds, used by the former ones.

3.1 Venues 4Clouds

Context: Venues 4Clouds needs to read the data from the multiple data sources available as open data in order to enrich the Venues 4Clouds dataset.

What was done: One of the biggest advantages of the Venues 4Clouds is the possibility of data collection, organization and aggregation from multiple source destinations. These destinations can be online data catalogues, API's and plain text files.

For this sore reason, two modules have been designed and implemented. Modular approach to the problem allowed us to split the responsibility and ensure that the data collected should be as close to the source as possible.

Module “dss-data-import” is a module which can read multiple data sources and map the data extracted to the SDOUT. Module is a compiled cross-platform binary which can read online sources, like API's, HTTP based XML files, including HTML; as well as local sources with XML, JSON and Excel structure. Dss-data-import module uses user defined map, described in the deliverable D2.3.2 to map extracted data to the descriptor used within the DSS platform, hence making it readable for the Venues 4Clouds. It has been used successfully to extract the data from sources like:

- <http://crunchbase.com> (HTTP REST API)
- <http://cloudharmony.com> (HTTP REST API)
- <http://wikipedia.com> (HTTP HTML parse)
- Cloud Security Alliance cloud providers submitted excel files

“dss-data-import” module produces human readable JSON output which can be either fed to as a STDIN to the other modules or redirected to a file in order to produce the input for further processing.

Second module designed to fulfil the needs of the data extraction is “dss-data-save”. The responsibility of this module is to form or enrich existing graph of data with the information provided. Deliverable D2.3.2 describes the inputs and outputs of the module as well as details of the implementation.

Context: There is a requisite to integrate a mechanism to collect the data directly from the providers giving them access to add and modify existing data.

What was done: An online survey like mechanism directly bounded to the graph enriching mechanism was designed and implemented. It allows hosting multiple surveys at once, sharing the responsibilities of data collection as well as later modification of the provided provider data. Implementation designed solely to ease the data collection from small European providers which do not provide detailed information needed, or are excluded from other catalogues due to the size, to be included in the Venues 4Clouds cloud services selection.

Context: Integrate mechanisms to collect crowdsourced data to the platform in order to enrich, validate and extend the collected cloud services provided data set.

What was done: An independent platform was designed and implemented which allows the federated ways of extending the existing dataset. Platform is designed to allow the users to search, review and correct the DSS dataset in a collective manner, meaning that the changes need to be validated and reviewed multiple times by other users in order to validate themselves.

3.2 CloudML

Context: CloudML library needs to manage deployments in IaaS cloud offerings.

What was done: In order to refine deployment models with provider-specific and runtime information (e.g., type of VM, IP address) as well as to provision VM and deploy applications on them, CloudML interacts with the cloud provider APIs. Within the CloudML deployment engine, these interactions are under the responsibility of a set of connectors. Currently, for the IaaS level management, the provisioning and deployment engine relies on the jclouds library[2], which provides an abstraction over more than 20 IaaS provider APIs, and an ad-hoc connector to interact with the Flexiant Cloud Orchestrator API[3]. So far, the CloudML IaaS connectors have been tested against the following providers:

- Flexiant
- AWS EC2
- CloudSigma
- Openstack

An IaaSConnector Interface has been created and allows the deployment engine to seamlessly exploit the library and other ad-hoc connectors.

Compared to D3.4.1 the set of operations under the responsibility of these connectors have been extended with operations for the creation and management of images and:

- Create snapshot of a VM
- Create an image of a VM
- Transform a Snapshot into an image
- List existing images

The updated implementation of the connectors is always available at the following address:

<https://github.com/SINTEF-9012/cloudml/tree/master/connectors/src/main/java/org/cloudml/connectors>

Context: CloudML library needs to manage deployments in PaaS cloud offerings.

What was done: For the PaaS level management, the provisioning and deployment engine relies on the unified PaaS layer library initiated during the Cloud4SOA project as well as on ad-hoc connectors.

They provide the ability to access and manage multiple cloud offerings (public and private). So far, the CloudML PaaS connectors have been tested against the following PaaS offerings:

- AWS RDS
- AWS SQS
- AWS Beanstalk
- Cloud Foundry
- Pivotal
- Cloudbees²

Compared to M24, in order to support public and private CloudFoundry PaaS services, an ad-hoc connector that relies on the CloudFoundry Java Client library[4] has been created. In addition, the PaaSConnector Interface that allows the deployment engine to seamlessly exploit the Unified PaaS library and the other ad-hoc PaaS connectors has evolved. As a result, the PaaS connectors are now responsible for the following operations:

- create application
- delete application
- create application's environment
- delete application's environment
- upload application
- start application
- stop application
- create service (e.g., message queues, databases)
- stop service
- bind a service to an application
- export environment variables

As depicted in Figure 5, the PaaS connectors, including the Unified PaaS library, are integrated within CloudML and exploited by the deployment engine in order to interact with the cloud provider APIs and then to trigger typical management actions on a specific cloud.

² As explained in D4.3.1[1], CloudBees decided to change the target of its service (and business) toward being an enterprise 100% Jenkins company. This change means that CloudBees shuttered down the RUN@cloud (which includes application and database hosting) services. CloudML offered support for these services before this change happened.

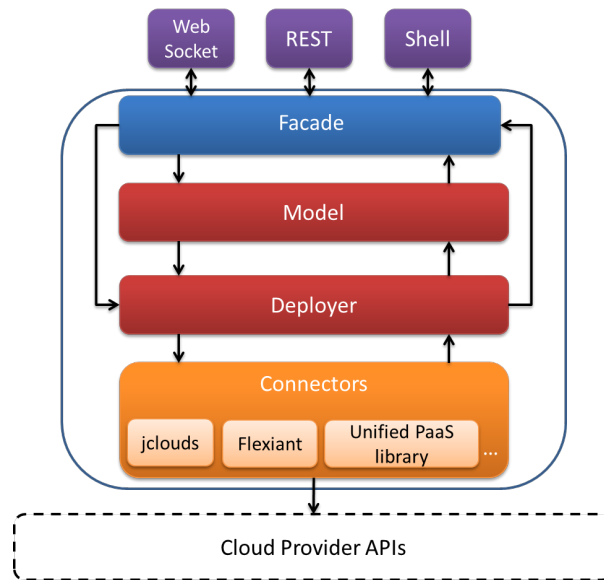


Figure 5 : CloudML architecture

The updated implementation of the connectors is always available at the following address:

<https://github.com/SINTEF-9012/cloudml/tree/master/connectors/src/main/java/org/cloudml/connectors>

4 Integration tests

The following tables describe the integration tests that have been designed for validating the integration between MODAClouds components. Each table contains the tests for a pair of components, and each is described with a brief description of the test, the result of the test, and any additional comment. Most of the components had to be modified in order to pass the tests. This explains why all the tests had succeeded as result.

Models@Runtime – SLA Tool		
Description	Result	Comments
<p>Agreement IDs compatibility</p> <p>The aims of this test is to check that the ID specified within the CloudML deployment model by the Functional modelling environment match the ID specified in the SLA manager</p>	Succeeded	N/a
<p>Agreement enforcement when deploying</p> <p>The objective of this test is to check that the SLA manager URL is properly provided to the Models@Runtime engine together with the agreement ID and that, in turn, the Models@Runtime engine is able to enforce the agreement once the deployment completed.</p>	Succeeded	N/a
<p>Enforcing agreement only when a proper ID is specified.</p> <p>The objective is to check that the absence of an agreement ID or the improper specification of an ID does not prevent the deployment of the cloud application.</p>	Succeeded	N/a

Models@Runtime - Runtime GUI		
Description	Result	Comments

<p>Trigger a deployment from the runtime GUI</p> <p>The objective of this test is to check that a deployment process can be triggered from the Runtime GUI. This includes: connecting to the Models@Runtime engine, pushing a model to the engine, starting a deployment, and receiving notifications describing the status of the deployment.</p>	<p>Succeeded</p>	<p>All credentials, SSH files and scripts to be uploaded with UploadCommands have to be placed on the machine running the Models@Runtime engine.</p>
<p>Trigger a bursting from the Runtime GUI</p> <p>The objective of this test is to trigger the bursting of a VM from the Runtime GUI.</p>	<p>Succeeded</p>	<p>N/a</p>
<p>Trigger a scaling from the Runtime GUI</p> <p>The objective of this test is to trigger an horizontal scaling of a VM from the Runtime GUI</p>	<p>Succeeded</p>	<p>For now, using the Runtime GUI, it is only possible to scale VMs one by one.</p>
<p>Load model into GUI from models@runtime</p> <p>The objective of this test is to check that it is possible to display into the Runtime GUI the status of an ongoing or achieved deployment.</p>	<p>Succeeded</p>	<p>N/a</p>

<p align="center">Models@Runtime - Load Balancer Controller</p>		
<p align="center">Description</p>	<p align="center">Result</p>	<p align="center">Comments</p>
<p>Start load balancer</p> <p>The objective of this test is to validate that when modelling the MODAClouds Load Balancer as a CloudML external component, the Models@Runtime engine is able to start it.</p>	<p>Succeeded</p>	<p>N/a</p>

<p>Configure load balancer</p> <p>The objective of this test is to validate that when modelling the MODAClouds Load Balancer as a CloudML external component, the Models@Runtime engine is able to configure it during the deployment process on the basis of its relationships with other CloudML components.</p>	<p>Succeeded</p>	<p>N/a</p>
--	------------------	------------

<p>Models@Runtime – Tower 4Clouds</p>		
<p>Description</p>	<p>Result</p>	<p>Comments</p>
<p>Test the proper export of the environment variables pointing to Tower 4Clouds</p> <p>In order to communicate with the monitoring manager, the data collectors have to be aware of its IP address and port. These data are provided by exporting environment variables. This test checks that the Models@Runtime engine properly export these variables</p>	<p>Succeeded</p>	<p>N/a</p>
<p>Test the proper export of environment variables describing the components to be monitored.</p> <p>Information related to the components to be monitored has to be provided to the Tower 4Clouds Manager by the data collector. This information is fed into the data collector by the Models@Runtime engine, which export relevant environment variables. This test validate that these environment variables are properly exported</p>	<p>Succeeded</p>	<p>N/a</p>

<p>Test component with environment variables are monitored.</p> <p>Information related to the components to be monitored has to be provided to the Tower 4Clouds Manager by the data collector. This information is fed into the data collector by the Models@Runtime engine, which export relevant environment variables.</p> <p>This test validate that the environment variable are properly retrieved and exploited by the data collectors.</p>	<p>Succeeded</p>	<p>N/a</p>
<p>Added components are dynamically monitored</p> <p>When deploying new components or adapting the deployment of an application, the monitoring manager has to be aware of the new deployment. This test validate that in such case data collectors are properly deployed and that the necessary environment variables are exported.</p>	<p>Succeeded</p>	<p>N/a</p>

Models@Runtime – Creator 4Clouds		
Description	Result	Comments
<p>Multi-cloud deployment</p> <p>The objective of this test is to validate that Creator 4Clouds can seamlessly trigger multi-cloud deployments using either the Models@Runtime engine or an CloudML as a library.</p>	<p>Succeeded</p>	<p>N/a</p>

Runtime GUI – SLA Tool		
Description	Result	Comments
<p>Display SLA violations</p> <p>The objective of this test is to validate that the Runtime GUI can retrieve and display SLA violations for a specific ID</p>	<p>Succeeded</p>	<p>N/a</p>

Display number of violations	Succeeded	N/a
The objective of this test is to validate that the Runtime GUI can display the number of violations for a specific cloud service		

Space^{Dev} 4Clouds – Creator 4Clouds		
Description	Result	Comments
<p>Analysis of an application modelled in Creator 4Clouds with IaaS resources, using Space^{Dev} 4Clouds.</p> <p>This tests validates that an application modelled in Creator 4Clouds by specifying IaaS resources in the deployment diagram can be successfully exported by Creator 4Clouds, imported in Space^{Dev} 4Clouds and analysed to assess QoS</p>	Succeeded	Manual tests performed in the Constellation and BOC case studies
<p>Analysis of an application modelled in Creator 4Clouds with PaaS resources, using Space^{Dev} 4Clouds.</p> <p>This tests validates that an application modelled in Creator 4Clouds by specifying PaaS resources in the deployment diagram can be successfully exported by Creator 4Clouds, imported in Space^{Dev} 4Clouds and analysed to assess QoS</p>	Succeeded	N/a
<p>Analysis of an application modelled in Creator 4Clouds with mixed IaaS and PaaS resources, using Space^{Dev} 4Clouds.</p> <p>This tests validates that an application modelled in Creator 4Clouds by specifying both IaaS and PaaS resources in the same deployment diagram can be successfully exported by Creator 4Clouds, imported in Space^{Dev} 4Clouds and analysed to assess QoS</p>	Succeeded	Manual Test performed on the BOC case study (still while the case study was under development)

<p>Optimization of an application modelled in Creator 4Clouds with IaaS resources, using Space^{Dev} 4Clouds.</p> <p>This tests validates that Space^{Dev} 4Clouds is capable of deriving deployment decisions from applications that specify IaaS resources in their deployment diagram</p>	<p>Succeeded</p>	<p>Manual Tests performed in the Constellation and BOC case studies</p>
<p>Optimization of an application modelled in Creator 4Clouds with PaaS resources, using Space^{Dev} 4Clouds.</p> <p>This tests validates that Space^{Dev} 4Clouds is capable of deriving deployment decisions from applications that specify PaaS resources in their deployment diagram</p>	<p>Succeeded</p>	<p>N/a</p>
<p>Optimization Result feedback in Creator 4Clouds.</p> <p>This test validates that Creator 4Clouds is capable of ingesting the deployment configuration derived by an optimization from Space^{Dev} 4Clouds and update the model show to the user</p>	<p>Succeeded</p>	<p>N/a</p>

Space 4Clouds^{Dev} – Line		
Description	Result	Comments
<p>LQN model solution from Space^{Dev} 4Clouds.</p> <p>This test validates that Space^{Dev} 4Clouds is able to communicate with LINE to provide the LQN model for the analysis and retrieve the results</p>	<p>Succeeded</p>	<p>Manual test, validated by extensive usage of LINE</p>
<p>Assessment of Space^{Dev} 4Clouds solutions using LINE.</p> <p>This test validates the ability of LINE and Space^{Dev} 4Clouds to evaluate a solution composed by 24 LQN model in parallel.</p>	<p>Succeeded</p>	<p>Manual test, validated by extensive usage of LINE</p>

<p>Optimization of Space^{Dev} 4Clouds solutions using LINE.</p> <p>This test validates that LINE is capable of sustaining the high number of model evaluation necessary for the optimization performed by Space^{Dev} 4Clouds</p>	Succeeded	Manual test, validated by extensive usage of LINE
<p>Importing Results from LINE to Space^{Dev} 4Clouds.</p> <p>This tests validates that the QoS results produced by LINE are successfully imported in Space^{Dev} 4Clouds</p>	Succeeded	Manual test
<p>Auto-detection of existing LINE servers.</p> <p>This test validate that Space^{Dev} 4Clouds is capable of detecting existing instances of a LINE server and connecting to them.</p>	Succeeded	Manual test, validated by extensive usage of LINE
<p>Automatic launch of LINE server.</p> <p>This test validates the ability of Space^{Dev} 4Clouds to launch a standalone LINE server and connect to it.</p>	Succeeded	Manual test

Tower 4Clouds – CloudML		
Description	Result	Comments
<p>Connection between data collectors and manager.</p> <p>This test validates that data collectors can connect to the manager after being deployed using CloudML</p>	Succeeded	N/a
<p>Model creation and register by data collectors.</p> <p>This test validates that data collectors can successfully create and register the model according to the environment variables specified in CloudML</p>	Succeeded	N/a

Data Migration and Synchronization – Models@Runtime		
Description	Result	Comments

<p>Execute CRUD operations compatible with Data Migration.</p> <p>This test validates that the CPIM library is able to execute CRUD operations which are compatible with Data Migration and Synchronization which allow Data Migration and Synchronization to perform correct data migration and synchronization</p>	Succeeded	N/a
--	-----------	-----

SLA Mediator – Creator 4Clouds		
Description	Result	Comments
<p>Test SLA generation when application is modelled to be deployed on IaaS.</p> <p>The cloudElement element in the ResourceContainerExtension file has a structure depending of the type of provider, which has been modified several times in the life of the project. This test checks both components are in sync regarding the definition of the cloudElement.</p>	Succeeded	Mediator makes a sanity check to ensure the agreement is at least generated in case of a mismatch.
<p>Test SLA generation when application is modelled to be deployed on PaaS.</p> <p>The cloudElement element in the ResourceContainerExtension file has a structure depending of the type of provider, which has been modified several times in the life of the project. This test checks both components are in sync regarding the definition of the cloudElement.</p>	Succeeded	Mediator makes a sanity check to ensure the agreement is at least generated in case of a mismatch.
<p>Test successful binding with Monitoring Rules.</p> <p>This test checks that the guarantee terms are correctly generated using the names of the output metric of the monitoring rules.</p>	Succeeded	N/a
<p>Test successful generation of Quality of Business (QoB) rules.</p> <p>This test checks that the QoB rules match the penalty schema and that they are bounded to the right rule.</p>	Succeeded	N/a

Creator 4Clouds - Tower 4Clouds		
Description	Result	Comments
Test deployment of monitoring rules in Tower 4Clouds and successfully register rules in Tower 4Clouds webapp.	Succeeded	N/a

Creator 4Clouds – Feedback Loop		
Description	Result	Comments
Test if deployment of Feedback Loop configuration files created the right files on the Object store. Check for configuration file, usage model and resource model.	Succeeded	N/a
Import files generated by filling the gap and check CCIM model is updated accordingly. Verify data imported from configuration file, usage model and resource model.	Succeeded	N/a

Creator 4Clouds – CloudML		
Description	Result	Comments
Generate scripts for IaaS and PaaS models and check (manually) if scripts correspond to the model.	Succeeded	N/a
Deploy IaaS and PaaS models and check if instances are correctly created according to the model.	Succeeded	N/a

Creator 4Clouds - QoS Models		
Description	Result	Comments
Create a QoS constraint with any values for all attributes, connect it to a model element then generate a monitoring rule from it. Data from the constraint should be carried	Succeeded	N/a

to the generated rule.		
------------------------	--	--

Auto-scaling Reasoner - Models@Runtime		
Description	Result	Comments
<p>Send scaling commands to Models@Runtime.</p> <p>This test validates that Auto-scaling Reasoner is able to communicate with Models@Runtime scaling up and down the number of active VMs</p>	Succeeded	Manual test, validated by extensive usage of Models@Runtime. Several bugs have been solved during this process
<p>Send cloud bursting command</p> <p>This test aims at validating the Auto-scaling Reasoner ability to perform cloud bursting</p>	Succeeded	Manual test. In this test a synthetic limit has been set in Models@Runtime to work with Amazon and emulate the case of full private cloud.

Auto-scaling Reasoner - Tower4Clouds		
Description	Result	Comments
<p>Cloud bursting rules</p> <p>This test aims at validating the integration of the Auto-scaling Reasoner with the monitoring platform. In particular we refer here to the cloud bursting mechanism that relies on monitoring rules installed in the monitoring platform</p>	Succeeded	Manual test, validated by extensive usage of Tower4Clouds
<p>Stress tests</p> <p>We performed a stress test on the monitoring platform to verify its capacity to handle data arrival rates required by the SAR</p>	Succeeded	Manual test, validated by extensive usage of Tower4Clouds

Auto-scaling Reasoner – SDA		
Description	Result	Comments
Interface Auto-scaling Reasoner - Statistical Data Analyzer (SDA)	Succeeded	Manual test, validated by extensive usage of

<p>This test aims at validating the integration of the Auto-scaling Reasoner with several SDAs. In particular, we realized and tested an communication interface between SDAs and Auto-scaling Reasoner</p>		<p>Tower4Clouds</p>
<p>Stress test</p> <p>We performed a stress test on the monitoring platform to verify its capacity to handle data arrival rates required by the SAR</p>	<p>Succeeded</p>	<p>Manual test, validated by extensive usage of Tower4Clouds</p>

<p>Venues 4Clouds – Creator 4Clouds</p>		
<p>Description</p>	<p>Result</p>	<p>Comments</p>
<p>Allow to import the Cloud Resource Descriptor design time model into Venues 4Clouds and extract the information about the services needed in order to fulfil the required deployment.</p>	<p>Succeeded</p>	<p>N/a</p>
<p>Enrich the Cloud Resource Descriptor model with the user selected deployment strategy in order to allow specification of a service and provider needed.</p>	<p>Succeeded</p>	<p>N/a</p>
<p>Export the modified Cloud Resource Descriptor model into the format which Creator 4Clouds can read.</p>	<p>Succeeded</p>	<p>N/a</p>

5 Design Time integrated platform

The MODAClouds design time environment is described in detail in D4.3.3 “MODACloudML IDE - Final version” [9], and is composed of the following components:

- Functional Modelling Tool
- Space^{Dev} 4Clouds
- LINE
- CloudML
- Resource Repository

The most important component of the Design Time environment is the Functional Modelling Tool. Considering it is implemented as a module for the Modelio modelling tool, it is therefore an actual program to be installed and used in a client host, and not to be used as an external service.

The initial approach for obtaining a Design Time platform was to prepare a VM with all the aforementioned components already installed. This decision was taken by the fact that there are strong dependencies between Space^{Dev} 4Clouds and Palladio that makes it difficult the creation of installation scripts that automates the task of installing the components on any machine. This VM will be hosted on multicloudsdevops.com. One then can use VNC or any other remote desktop applications to use MODACloudsML IDE.

Further, the instructions for installing a MODAClouds IDE – without Space^{Dev} 4Clouds - are described in detail in [10].

At the time of writing, the consortium is working towards having Docker images as another way of obtaining a Design Time platform, with the intention of facilitating a potential update of the platform.

6 Runtime integrated platform

The MODAClouds runtime environment, used to integrate and deploy all MODAClouds services, is described in detail in D6.5.2 “Run-time environment – Initial release” [5] and D6.5.3 “Run-time environment – Final release” [6] deliverables. The underlying platform for integration of MODAClouds services is based on mOSAIC platform, which has been updated for MODAClouds context as described in D6.5.1 “Run-time environment – proof of concept” [7]. Thus, the mOS platform provides a distributed execution environment and resource discovery services for the MODAClouds runtime components. mOS 2.x, based on OpenSUSE 13.x operating system, is designed to run on any compatible Cloud infrastructure. Hence, the components need to be packed as RPM packages, which is the format supported by OpenSUSE. The process of obtaining RPM packages is described in Appendix A.1.

This procedure allows the deployment of a MODAClouds Runtime Platform not only on mOSAIC, but on any OpenSuse13.1 host. The installation and usage instructions of the integrated platform can be found in D3.5.3 “MODAClouds integrated solution – Final version” [8].

As a summary of the integration process, each MODAClouds component developer needs to provide the following artefacts:

- a distribution bundle, namely `distribution.tar.gz` archive that contains the files needed to run the component
- a descriptor in JSON format, namely `package.json`, used build the RPM i.e. packing the distribution and the wrapper script
- a wrapper script, namely `service-run.bash`, that runs the service.

Each component is installed in `/opt/modaclouds-services-<component-name>` with the wrapper script in `modaclouds-services-<component-name>/bin` sub-folder.

The rest of the details for the integration of the Runtime Platform can be found in the appendices. Appendix A.1 describes the workflow and assets involved in the process of obtaining the RPM package for a MODAClouds component. Appendix A.2 details the service configuration, covering `package.json` descriptor. Guidelines and example of how to build a wrapper script are provided in Appendix A.3.

7 Conclusion

This document describes the work done in the context of the Task 3.3 “MODACLOUDS coordination for the integration”, from M24 to M36. It is part of last series of documents to be delivered in the context of WP3, along with D3.5.3 “MODAClouds integrated solution – Final version” [8] and D3.7.2 “MODACLOUDS evaluation report – Final version” [11].

It reported on the progress made by the consortium in the project integration activities, explaining the different integration issues encountered by the MODAClouds components, how they were addressed, and the battery of integration tests that have been developed to ensure the right integration of the MODAClouds components with other MODAClouds components or external services .

It also describes the work performed to facilitate the creation of instances of the Design Time and Runtime platforms.

The result of this integration work will be visible in the official MODAClouds sites.

8 References

- [1]: MODAClouds deliverable D3.4.1: MODAClouds Integration Report – Initial version
- [2]: JClouds: <http://www.jclouds.org>
- [3]: Flexiant Cloud Orchestrator API: <http://docs.flexiant.com>
- [4]: CloudFoundry Java Client: <https://github.com/cloudfoundry/cf-java-client>
- [5]: MODAClouds deliverable D6.5.2: Run-time environment – Initial release
- [6]: MODAClouds deliverable D6.5.3: Run-time environment – Final release
- [7]: MODAClouds deliverable D6.5.1: Run-time environment – Proof of concept
- [8]: MODAClouds deliverable D3.5.3: MODAClouds integrated solution – Final version
- [9]: MODAClouds deliverable D4.3.3: MODACloudML IDE - Final version
- [10]: Creator 4Clouds User Guide: <http://forge.modelio.org/projects/creator-4clouds/wiki/Wiki>
- [11]: MODAClouds deliverable D3.7.2: MODACLOUDS evaluation report – Final version

Appendices

Appendix A

A.1. MODAClouds RPM packages

Workflow for packing MODAClouds components

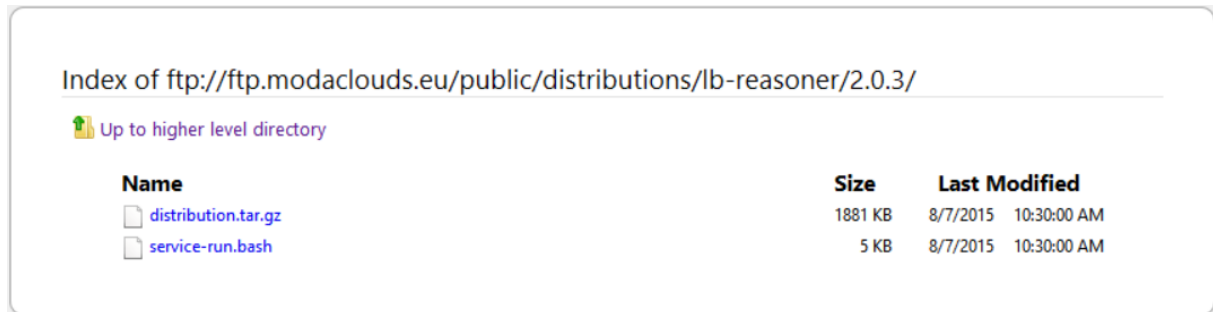
Building a fully deployable RPM package for a MODAClouds software component needs two actors (the DEVELOPER of the component and the PACKAGER), and a number of steps. The PACKAGER role is played by one of IeAT staff members. The detailed workflow for packing MODAClouds components is depicted in Table 1 MODAClouds Packing Workflow.

Table 1 MODAClouds Packing Workflow

#	Task	Actor	RPM status
1	Development of the tool	DEVELOPER	Pending release
2	Build the distribution bundle (distribution.tar.gz)	DEVELOPER	Pending release
3	Create the wrapper script needed to run the service (service-run.bash)	DEVELOPER	Pending release
4	Upload the distribution bundle and wrapper script to MODAClouds FTP server ³ under /public folder and notify the PACKAGER	DEVELOPER	Pending release -> Pending packing
5	Packing, creation of RPM and uploading the RPM on mOS repository	PACKAGER	Pending packing -> Pending validation
6	Test the RPM package	DEVELOPER	If OK then status=Available, else Notify the PACKAGER

The structure of MODAClouds FTP server /public section is illustrated in Figure 6 MODAClouds FTP server structure. Under /distributions sub-folder each component has its own home folder, where the artefacts (distribution.tar.gz and service-run.bash files) are organized per version. The figure illustrates the distribution archive and the wrapper script for Load Balancer Reasoner 2.0.3 component.

³ ftp.modaclouds.eu



Index of ftp://ftp.modacloudeu/public/distributions/lb-reasoner/2.0.3/

[Up to higher level directory](#)

Name	Size	Last Modified
distribution.tar.gz	1881 KB	8/7/2015 10:30:00 AM
service-run.bash	5 KB	8/7/2015 10:30:00 AM

Figure 6 MODAClouds FTP server structure

The `package.json` files are stored on the MODAClouds mOS platform packages repository at:

<https://github.com/modacloudeu/modacloudeu-mos-platform-packages>

In order to follow-up the packing process of all MODAClouds components a spreadsheet has been introduced where components are grouped by MODAClouds Exploitable artefacts (Tower 4Clouds, ADDapters 4Clouds, Space^{Ops} 4Clouds, Filling the Gap and Other tools). Table 2 MODAClouds packages illustrates an excerpt of this table for ADDapters 4Clouds and Space^{Ops} 4Clouds artefacts.

The column “RPM status” holds the status of each component:

- available means that the RPM exists and it was tested by the developer to be fully functional;
- pending validation means that a (new) RPM exists (just packaged), and the developer has to test if the package works in the integration environment;
- pending packaging means that a new distribution archive exists (on MODAClouds FTP) (together with updated wrapper scripts), and the packager has to generate the RPM;
- pending release: the developer is working on a new version to soon be released.

The distribution column points to the MODAClouds FTP server location of the distribution archive and wrapper script of each component.

Table 2 MODAClouds packages

	identifier	type	RPM status	RPM name	RPM version	RPM dependencies	packaged version	released version	distribution (tar.gz, jar, etc.)
ADDapters 4Clouds									
	load-balancer-controller	service	available	modaclouds-services-load-balancer-controller	0.7.0.15	python haproxy	0.2.8	0.2.8	http://ftp.modaclouds.eu/public/distributions/lb-controller/0.2.8/
	artifact-repository	service	pending release	mosaic-artifact-repository	[missing]	python	[missing]	[missing]	[missing]
	object-store	service	available	mosaic-object-store	0.7.0.15	[N/A]	[N/A]	[N/A]	[N/A]
SpaceOps 4Clouds									
	models-at-runtime	service	pending validation	modaclouds-services-models-at-runtime	0.7.0.15	mosaic-rt-jre-7	0.2.0	0.2.0	http://ftp.modaclouds.eu/public/distributions/mrt/0.2.0/
	load-balancer-reasoner	service	pending validation	modaclouds-services-load-balancer-reasoner	0.7.0.15	mosaic-rt-jre-7 modaclouds-rt-matlab-mcr-r2013a	2.0	2.0	http://ftp.modaclouds.eu/public/distributions/lb-reasoner/2.0/
	mdload	tool	pending validation	modaclouds-tools-mdload	0.7.0.15	mosaic-rt-jre-7 modaclouds-rt-matlab-mcr-r2013a	0.1.0	0.1.0	http://ftp.modaclouds.eu/public/distributions/mdload/0.1.0/

Building the distribution.tar.gz archive

For Java applications, one can use Maven to create the needed `distribution.tar.gz`, which should hold both the JAR with dependencies and other required files. Following the steps below will create an archive ending in `tar.gz` that contains a single top folder named `distribution`, and inside it a file `service.jar`:

1. Copy the `maven-assembly-plugin` configuration snippet presented below in your POM file
2. Copy the `distribution-tar-gz.xml` file from

https://github.com/cipriancraciun/modaclouids-mos-platform-packages/blob/development/packages/_modaclouids-services-template/application/src/assembly/distribution-tar-gz.xml

to your local `src/assembly` folder.

3. In order to include additional files into the distribution archive simply place them into the `src/distribution` folder.

```
<plugin>
  <artifactId>maven-assembly-plugin</artifactId>
  <executions>
    <execution>
      <id>jar-with-dependencies</id>
      <phase>package</phase>
      <goals>
        <goal>single</goal>
      </goals>
      <configuration>
        <attach>>true</attach>
        <archive>
          <manifest>
            <!-- NOTE: Replace with the proper main class! -->
            <mainClass>RunService</mainClass>
          </manifest>
        </archive>
      </configuration>
    </execution>
  </executions>
</plugin>
```

```
        </archive>

        <descriptorRefs>
            <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
    </configuration>
</execution>

<execution>
    <id>distribution-tar-gz</id>
    <phase>package</phase>
    <goals>
        <goal>attached</goal>
    </goals>
    <configuration>
        <attach>true</attach>
        <appendAssemblyId>>false</appendAssemblyId>
        <descriptor>src/assembly/distribution-tar-gz.xml</descriptor>
    </configuration>
</execution>
</executions>
</plugin>
```

Listing 6: Maven Assembly plugin configuration

For other languages, one can easily create a similar archive with other methods.

Building the RPM package

To create the RPM, one only needs to execute the following command:

```
env mpb_debugging_enabled=true \  
  
python2.7 /.../mos-package-builder.py /.../modaclouids-service-x-  
package/packaging /tmp/modaclouids-service-x.rpm
```

The `mos-package-builder.py` script is found in the repository <https://github.com/cipriancraciun/mosaic-mos-package-builder>, inside the `sources` sub-folder.

The present package descriptor (see section 0 for details) makes the following assumptions:

- the contents of the `distribution.tar.gz` will be extracted inside the `/opt/modacloudds-service-x/lib/distribution` folder, which will also be expanded into the `service-run.bash` script as `@{definitions:environment:SERVICE_X_DISTRIBUTION};`
- the `service-run.bash` script will be included inside `/opt/modacloudds-service-x/lib/scripts` as `run.bash`;
- the `service-run.bash` script will also be symlinked as `/opt/modacloudds-service-x/bin/modacloudds-service-x--run-service`; the superfluous `modacloudds-service-x--run-service` is needed to easily allow all `bin` folders to be put in `${PATH}` without having colisions, and allowing shell auto-completion;
- the run-time platform will always call `../../bin/modacloudds-service-x--run-service`;

Regarding the dependencies, inside the `requires` field one should not list those services to which the current one needs to connect to or use, but only those packages which provide actual run-time support for the service such as the Java JRE, or library, etc.

Packages versioning

The RPM repository synchronization script has been modified to save only the latest version (and release) of each package. Once build, the latest versions of the RPM packages are available under:

<http://mos.repositories.mosaic-apps.eu/v2/packages/modacloudds/rpm>

<http://mos.repositories.mosaic-apps.eu/v2/packages/mosaic/rpm>

<http://mos.repositories.mosaic-apps.eu/v2/packages/mos/rpm>

<http://mos.repositories.mosaic-apps.eu/v2/packages/external/rpm>

The older versions of the packages have been "archived" under

<http://mos-m1.repositories.mosaic-apps.eu/v2/packages/archive/rpm/modacloudds>

<http://mos-m1.repositories.mosaic-apps.eu/v2/packages/archive/rpm/mosaic>

<http://mos-m1.repositories.mosaic-apps.eu/v2/packages/archive/rpm/mos>

<http://mos-m1.repositories.mosaic-apps.eu/v2/packages/archive/rpm/external>

Both latest and archived mOS repositories can be accessed either as OpenSUSE RPM repositories (via Zypper) or loaded in Internet browsers to get a nice HTML view of what is contained.

Mirrors

Moreover IEAT currently maintains three mirrors which can be accessed by replacing ``http://mos.repositories...`` with ``http://mos-m1.repositories...`` / ``http://mos-m2.repositories...`` / ``http://mos-m3.repositories...``. These are hosted as follow:

- ``mos-m1`` hosted at our university which is pointed to by ``mos.repositories...``, which is thus the default one;
- ``mos-m2`` hosted at DigitalOcean as a backup
- ``mos-m3`` hosted on S3 Ireland as a backup

Remark: Never use these mirrors explicitly unless there is an issue with the main one (i.e. just ``mos``)!!!

Installing MODAClouds packages

There are three "meta-packages" that when installed also install all the packages in that category:

- `modaclouds-platform-core`, depends on the next two meta-packages and installs everything needed to run the services on-top of the mOSAIC run-time;
- `modaclouds-platform-services`, depends on all MODAClouds related services, providing everything needed to run them manually (or integrated in another deployment system), thus without ties to the mOSAIC run-time;
- `modaclouds-platform-tools`, currently depends only on ``modaclouds-tools-mdload``.

Thus one can now just issue the following and have everything installed:

```
zypper install modaclouds-platform-services
```

To install a single RPM, one only needs to execute the following:

```
zypper install /tmp/service-x.rpm
```

To run a MODAClouds installed service, see testing the wrapper script sub-section in section 0.

A.2 Building the package.json descriptor

The `package.json` is a file used by the mOS package builder, which provides all the details needed to generate mOS compliant RPM packages. For more details check the examples available at

<https://github.com/mosaic-cloud/mosaic-mos-package-builder/tree/development/examples>

Table 3 package.json example

```
{
// This part is constant. It identifies the JSON structure.
"_schema": "tag:ieat.ro,2014:mosaic:v2:mos-package-
builder:descriptors:composite-package",
"_schema/version" : 1,

// These are the the main package information tags.
// See: https://docs.fedoraproject.org/en-US/Fedora\_Draft\_Documentation/0.1/html/RPM\_Guide/ch-specfile-syntax.html
// The values are accessible as @{{package:<name>}}, etc.
"package" : {
  "name" : "@{{definitions:package:name}}",
  "version" : "@{{definitions:package:version}}",
  "release" : "@{{definitions:package:release}}",
  "architecture" : "i686",
  "root" : "/opt/@{{package:identifier}}"
},

// These are the package dependency tags. (See the link as above.)
"dependencies" : {
  "provides" : ["@{{definitions:package:name}}"],
  "requires" : ["mosaic-rt-erlang-r15b-32bit", "glibc-32bit"]
},
}
```

```
// These are the other (less important) package information tags.
"miscellaneous" : {
  "license" : "apache-2.0",
  "url" : "http://mosaic.ieat.ro/",
  "summary" : "mOSAIC components: RabbitMQ"
},

// This section describes how the contents of the package is to be created.
"overlays" : [
  { // This overlay states that some folders should be created.
    "generator" : "folders",
    // This `target` states that the folders to be created should be
    prefixed with the specified value.
    "target" : "@{package:root}",
    // This in effect states that `@{package:root}` should be created.
    "folders" : ["/"]
  },
  { // This overlay states that the contents of the resource file should
    be unarchived.
    "generator" : "unarchiver",
    // This states that the contents should be placed under the specified
    path.
    "target" : "@{package:root}",
    // This states the name of the rource to be used. (See the next
    section.)
    "resource" : "mosaic-components-rabbitmq.cpio.gz",
    // This states that the archive is to be extracted with `cpio` after
    `gunzip` has been run.
    "format" : "cpio+gzip"
  }
],
```

```
// This section describes resource files that are needed to create the
package.

"resources" : {

  "mosaic-components-rabbitmq.cpio.gz" : {

    // This states how the resource file is to be obtained, i.e. fetching
it from an URL.

    "generator" : "fetcher",

    // The URL of the resource (see below the URL base)!

    "uri"      :      "@{definitions:resources:url:base}/mosaic-components-
rabbitmq--@{definitions:resources:url:suffix}.cpio.gz",

    // This value is inferred automatically. It should not be usually
specified!

    "cache"          :          "mosaic-components-rabbitmq--
@{definitions:resources:url:suffix}.cpio.gz"

  }

},

// These are "variables" that allow some customization.
// The values are accessible as @{definitions:<name>}, etc.

"definitions" : {

  "package:name" : "mosaic-components-rabbitmq",

  "package:version" : "0.7.0_dev",

  "package:release" : "@{execution:timestamp}",

  "resources:url:base" :
"http://data.volution.ro/ciprian/public/mosaic/packages",

  "resources:url:suffix" : "@{definitions:package:version}"

}

}
```

Table 3 lists a `package.json` example, showing the main sections of the package descriptor:

- package general information, such as name and release

- external dependencies
- miscellaneous details, such as license and url
- overlays, which describes the overlays of the RPM, i.e. how actually the package is created
- resources needed to create the package
- definitions of variables, which allow service customization.

Communication between different services of MODAClouds platform is performed through **environment variables**. Each environment variable has an "owner" service, i.e. the service that specifies the name of the variable, what values should be taken and how it should be used. For example, the variables managed by the service `modaclouds-services-<something>` are named

`MODACLOUDS_<SOMETHING>_...`

To cope with the complexity of service configuration on MODAClouds platform, we introduced a unique sheet (Configuration) where all services HAVE TO declare the environment variables they export or use. This sheet provides the complete view of how the entire system is configured and deployed. This is used to provide the required information to configure the system under different conditions:

- when started on the same VM manually without any special configuration (the values are the default ones in the packaging descriptor)
- when started manually on multiple VM's with some configuration provided (the first column stating the environment variable name)
- when started automatically by the platform (the first column, Resolved by RT and Default columns are used).

Table 4 Configuration sheet example presents an extract from this table, more specifically for Load Balancer Reasoner component. One can see the definition of the environment variables "owned" by this component.

Table 4 Configuration sheet example

load-balancer-reasoner	Configuration variable (for the wrapper)	Conveyed to the code (by the wrapper)	Configuration variable (for the package JSON)	Type	Resolved by RT	Default (provided by the package JSON)
	MOSAIC_OBJECT_STORE_ENDPOINT_IP	environment:MOSAIC_OBJECT_STORE_ENDPOINT_IP	environment:OBJECT_STORE_ENDPOINT_IP	TCP-connect	resolved	0.0.0.0
	MOSAIC_OBJECT_STORE_ENDPOINT_PORT	environment:MOSAIC_OBJECT_STORE_ENDPOINT_PORT	environment:OBJECT_STORE_ENDPOINT_PORT	TCP-connect	resolved	20622
	MODACLOUDS_LB_REASONER_CONFIG_PATH	environment:MOSAIC_OBJECT_STORE_LB_REASONER_PATH	environment:LB_REASONER_PATH	URI-path	(package)	/modaclouds/lb/configuration_LB.xml
	MODACLOUDS_LB_REASONER_MCR_HOME	argument:\$1	environment:MCR_HOME	FS-path-RO	(package)	/opt/modaclouds-rt-matlab-mcr-r2013a/v81
	MODACLOUDS_LB_REASONER_JAVA_HOME (to-be-deleted)	???	environment:JAVA_HOME	FS-path-RO	(package)	/opt/mosaic-rt-jre-7
	MODACLOUDS_LB_REASONER_PATH	environment:PATH	environment:PATH	\$PATH	(package)	@{definitions:environment:JAVA_HOME}/bin:/usr/bin:/bin
	MODACLOUDS_LB_REASONER_TMPDIR	environment:TMPDIR	environment:TMPDIR	FS-path-TMP	(script)	/tmp/modaclouds/@{package:identifier}
	MODACLOUDS_LB_REASONER_DISTRIBUTION	argument:\$0/run_main.sh	environment:LB_REASONER_DISTRIBUTION	FS-path-RO	(package)	@{package:root}/lib/distribution

The Configuration sheet contains one environment variable per row and for each variable the following information is provided:

- The first column holds the service name.
- The second column 'Configuration variable (for the wrapper)' is how the environment variable used by the wrapper script is named. Use '(no)' value if such a variable cannot be actually set by the user (e.g. PATH, or JAVA_HOME, which although are provided by the package, or perhaps computed by the script, cannot be directly overridden by the user.)
- The third column 'Configuration variable (for the package JSON)' lists how the variable should be defined in the 'package.json' inside the "definitions" map. (This name prefixed with '@{definitions:...}' will be expanded by the packager in the 'service-run.bash' script and other files.)
- The fourth column 'Conveyed to the code (by the wrapper)' is how the wrapper script communicates that value to the actual code; most of the time it just re-exports the same environment variable, but in some cases this value is either transmitted to as an argument or expanded in a configuration file template; for example:
 - 'environment:NAME', it re-exports the value as the environment variable 'NAME';
 - 'argument:\$N', it uses the value as the N-th positional argument; (the flags don't count as positional arguments;)
 - 'argument:--arg=\$X', it uses the value after the '--arg' flag; (i.e. not counting as a positional argument;)
 - '???' , it is not conveyed to the code; (i.e. this is a bug, please fix;)
 - 'clone:/etc', it copies that folder to a temporary place, most likely to expand some values; (the '/etc' is just a symbol, it doesn't actually copy in '/etc', but in a temporary folder that we designate logically as '/etc';)
 - '(expand)' it is used in some files (mentioned in an 'expand:/...' item), thus not used directly by the wrapper script
 - '(no)' if it is not actually passed directly to the code, but perhaps it is expanded into other variables and those are passed; (this is the example of SLA-core which takes two variables (for the endpoint) and one for the database name and constructs a JDBC URI; another example is 'JAVA_HOME' and the like which are used by the script, but not exported further to the code;)
 - '(...)', it uses some other technique;
- The fifth column 'Type' means the purpose of the configuration item:
 - 'TCP-listen', it designates the IP and port (in separate variables) where the service should bind and listen for connections;
 - 'TCP-listen-public' which denotes that this should be the Internet-facing endpoint that the service will use to register to other services on the Internet.

- `TCP-connect`, it designates the IP and port (in separate variables) where the service could contact other services it depends on;
 - `FS-path-RO`, it designates a path, most likely from the package, used to store either some static configuration files, or configuration file templates;
 - `FS-path-RW`, it designates a path, where the service can store data for long-term storage (i.e. databases);
 - `FS-path-TMP`, it designates a path, which should only be used to store volatile or cached data; (i.e. it is regularly cleaned)
 - `URI-path`, `(value)`, etc. represent various tokens;
 - `\$PATH` a `PATH`-like concatenation of other paths used for searching files.
- The sixth column, `Resolved by RT` describes how the value is derived when the service is started by the platform:
 - `provided`, usually for `TCP-listen`, means the run-time will set a unique value that is specific only for this instance; (on restart this value would change;)
 - `resolved`, usually for `TCP-connect`, means that the run-time will resolve the dependent service and provide its endpoint or other data;
 - `(package)` is a hard-coded value provided by the package and not computed by the script; (although it can be overridden when started manually via environment variables;) (basically all instances get the same hard-coded value;)
 - `(script)`, is a value computed by the script (perhaps based on some hard-coded value), but each instance of the service will get a different value; (like in the case of `TMPDIR`;) (also when started manually some of these could be overridden by an environment variable;); examples for `FS-path-RW` and `FS-path-TMP`
 - The last column, `Default`, is the hard-coded value (usually in `package.json`) that will be used by the wrapper script, and conveyed to the application, in case the service is started manually;

Although some values can be overridden by the user via environment variables when the service is started by hand, when started by the platform this is not possible, thus it is either taken from the package default, or computed by the script.

The only configuration arguments that can be automatically resolved by the mOS platform are the endpoints to listen to or to connect to. Therefore, for an automatic deployment on the platform, the default values at packaging time will be used; if these are 'dynamic' they should be provided in the object store (or other such store).

Running the scripts before packaging is not possible unless the `@definitions` (e.g. `_FG_DISTRIBUTION=@{definitions:environment:FG_DISTRIBUTION}'`) are fixed beforehand. This can be easily achieved by exporting all the environment variables, and the wrapper scripts should ignore the `@{...}` ones. To do this one can use the 'Integration environment' sheet and run the following command to obtain the proper environment variables values (then one just pastes that contents into the shell before running).

```

curl -s -f --
'https://docs.google.com/spreadsheets/d/1_faCBHf45xpHLb05XuAg-
7i0QMnMQ16ImpguPoc/pub?gid=370961097&single=true&output=tsv'

\

| tr '\r' '\n' \

| tr -s '\n' \

| cut -s -f 2,3,5 \

| grep -v -F -e '@' -e '!' \

| sed -r -e 's/^(([^\t]+)\t([^\t]+)\t([^\t]*)$/\1\t\3\n\2\t\3/g' \

| sort -u \

| sed -r -e 's/^(([^\t]+)\t([^\t]*)$)/export \1='\''\2'\''/g'

```

A coloring protocol has been also introduced to ensure consistency between what is described in the spreadsheet and the code / scripts themselves:

- green – has been verified and everything is ok (either in the scripts for the `Configuration variable (for the wrapper)` and `Conveyed to the code (by the wrapper)`, or in the packaging descriptor for the `Configuration variable (for the package JSON)` and `Default`)
- orange - those variables that should be changed because they mismatch from the service owner and the service user,
- yellow - those that should be updated to match the naming policy.
- red – there is an issue with this variable, e.g. must be changed as they conflict, it was recently introduced and not checked etc.

A special attention need to be paid regarding the configuration of endpoints for each service. Two separate endpoints could be configured for each MODAClouds service:

- `SERVICE_X_ENDPOINT_IP` and `SERVICE_X_ENDPOINT_PORT` -- used to bind and advertise to the other services part of the same cluster;
- `SERVICE_X_ENDPOINT_IP_PUBLIC`, `SERVICE_X_ENDPOINT_PORT_PUBLIC` -- used to advertise to clients connecting outside the cluster.

During the deployment of services in mOS platform, the cluster IP addresses `0.0.0.0` will be replaced with the actual cluster IP of the VM the service is running on. Hence, the usage of `0.0.0.0` as IP address in configuration scripts is a default value that allows both binding-on and connecting-to when all the components run on the same VM. For testing purposes, if it's done on the same VM the usage of `0.0.0.0` for IP's is acceptable as a workaround. If testing on multiple VM's, always export the correct cluster IP of the VM's in each of the `*_ENDPOINT_IP`.

All services connecting to other services part of the same cluster or cloud should always use the cluster IP, via `*_ENDPOINT_IP`; It is not recommended to use the PUBLIC endpoint to connect to services part of the same cluster or cloud (see below).

The PUBLIC endpoints IPs are not available at VM level, thus binding on that IP would yield an error. They are managed by the hosting infrastructure that maps that public IP to the cluster one, usually in the routing layer via DNAT. Thus, **a service must always bind to the cluster IP!** Trying to bind to a public endpoint might even fail to work due to the fact that the packet would need to travel up the network path until it reaches the router that knows about the public IP to cluster IP mapping. Thus, even if a service needs the PUBLIC endpoint (e.g. to send it to other services interested in accessing it over Internet) it must also require the cluster endpoint and use it to bind to.

PUBLIC endpoints are required only by those services that need to register themselves to other services running outside the cluster/cloud (i.e. over the Internet). Only the services that listen on that endpoint should require to know the public endpoint (IP and port). The usage of public endpoints need to be carefully mitigated since it usually increases latency, hence impacting the performance, and incurs additional costs since the data (network packages) need to travel over cluster boundaries.

Below are some rules that need to be followed for a successful service endpoint configuration in MODAClouds distributed environment:

- If service X exports a variable (thus is the "owner"), say `MODACLOUDS_SERVICE_X_ENDPOINT_IP` then only the ``modaclouids-services-X`` service will use it to listen (bind) on it, and all the other services needing to connect to service X will use this name also
- If one service listens on an endpoint and this is configured with the variable ``MODACLOUDS_SERVICE_X_ENDPOINT_IP``, then all the other services needing to connect to it should also use the same variable (not a different one).
- All services (that require listening on sockets) should allow the specification of both ``...ENDPOINT_IP`` and ``...ENDPOINT_PORT``, even though the same service might require multiple endpoints, which presumably share the same IP but different ports; moreover each socket should be bound to its proper IP and port (which might have different IP's). Same rule applies to the public endpoints as well.

In the end of this section, we provide a brief checklist to be verified for your service configuration:

1. Check that the service wrapper use the default values specified in config tab. This should be easy: start the script without parameters, and check that the values of the parameters shown at the end of the script are as they should be.
2. Check that the service communicates with the other components. ``netstat -tnp`` would help, or ``lsof``.
3. Check (if possible) if the service actually got the correct values, e.g. logging to ``stderr`` all the configuration details.
4. Make sure that the service is listening on the correct ports (via ``netstat -tnlp``).

A.3 The wrapper script

The wrapper script is executed when a service is started using the following command:

```
/opt/modaclouids-service-X/bin/modaclouids-service-X--run-service
```

The script is saved in the file `service-run.bash`

How to create the wrapper script (`service-run.bash`)

The `service-run.bash` file provided at

<https://github.com/modaclouids/modaclouids-mos-platform-packages/blob/development/packages/modaclouids-services-template/packaging/service-run.bash>

is based on the other existing services run scripts and:

- it allows one to hard-code some variables during the packaging (see section 5.2) by simply using `@{definitions:...}`;
- it allows one to override at run-time some variables by using the matching `MODACLOUDS_...` environment variables;
- it takes care of creating a temporary workspace for each instance of the service, enabling thus to run multiple instances of the same service without interfering.

To customize it, you only need to touch three sections:

- `_variables_defaults`, which is an array of variables used within the script (usually to configure the service), whose values are either hard-coded or expanded at packaging time; the values such as `@{definitions:...}` are defined inside the `package.json` file; these are specified in the Configuration sheet's last column
- `_variables_overrides`, which usually holds the same variables as the previous section, but it tries to use the overriding `MODACLOUDS_...` environment variables, and if not present falling back to the hard-coded values;
- `_environment`, which is an array of environment variables to be exported to the service.

In a nutshell, if a service needs a variable, it has to do the following:

- add its value to the `package.json` definitions, by using the identifier `environment:SERVICE_X_SOMETHING;`
- add a line for it in the `_variables_default` array, by using the identifier `_SERVICE_X_SOMETHING;`
- add a line for it in the `_variables_overrides` array, by using the internal identifier `_SERVICE_X_SOMETHING`, and the overriding environment variable `MODACLOUDS_SERVICE_X_SOMETHING;`

- add a line for it in the `_environment` array, by using the identifier `MODACLOUDS_SERVICE_X_SOMETHING`;

Note that no other environment variables, except the ones explicitly added to the `_environment` array are exported further to the service, thus enabling a more robust and deterministic execution, not tainted by the testing environment.

How to test the wrapper script

In order to test the script, and because the default values for the various configuration variables are not yet expanded, one has to manually export all the proper values like below:

It is assumed that the contents of the `distribution.tar.gz` is already available at the path `/tmp/modaclouds-service-x-distribution`.

In order to test the wrapper script (i.e. `service-run.bash`) before packing, we propose the following workflow:

1. Start with a fresh VM with OpenSUSE 13.x image (or mOS 2.x)
2. Run `'zypper install modaclouds-services'` command to install all MODAClouds services from MODAClouds repository
3. Export required environment variables for your service using the following command:

```
env \  
  
MODACLOUDS_SERVICE_X_PATH="${PATH}" \  
  
MODACLOUDS_SERVICE_X_TMPDIR="/tmp/modaclouds-service-x" \  
  
MODACLOUDS_SERVICE_X_JAVA_HOME=/opt/java \  
  
MODACLOUDS_SERVICE_X_DISTRIBUTION=/tmp/modaclouds-service-x-  
distribution \  
  
MODACLOUDS_SERVICE_X_ENDPOINT_IP=127.0.0.1 \  
  
MODACLOUDS_SERVICE_X_ENDPOINT_PORT=8081 \  
  
MODACLOUDS_SERVICE_Y_ENDPOINT_IP=127.0.0.1 \  
  
MODACLOUDS_SERVICE_Y_ENDPOINT_PORT=8082 \  
  
./service-run.bash
```

It is assumed that the contents of the `distribution.tar.gz` is already available at the path `/tmp/modaclouds-service-x-distribution`.

Note that the full instructions to obtain a ready to use integrated platform from a fresh OpenSUSE image are detailed in MODAClouds installation section.

Good practices for writing the wrapper script

- Never use `sudo` inside the wrapper script, instead use the proper user.
- Redirecting the stdout is not enough; you should also redirect also the stderr, e.g. `command... >/tmp/some.log 2>&1`