H2020-ICT-2018-2-825040

Rational decomposition and orchestration for serverless computing

Deliverable D5.6

Data pipeline orchestration II

Version: 1.0

Publication Date: 31-Oct-2020

Deliverable Card

Deliverable	D5.6
Title:	Data pipeline orchestration II
Editor(s):	Chinmaya Dehury (UTR)
Contributor(s):	Chinmaya Dehury (UTR), Pelle Jakovits (UTR), Matija Cankar (XLB)
Reviewers:	Matija Cankar (XLB), Giuliano Casale (IMP)
Type:	Report
Version:	1.0
Date:	31-Oct-2020
Status:	Final
Dissemination level:	Public
Download page:	http://radon-h2020.eu/public-deliverables/
Copyright:	RADON consortium

The RADON project partners

IMP	IMPERIAL COLLEGE OF SCIENCE TECHNOLOGY AND MEDICINE
TJD	STICHTING KATHOLIEKE UNIVERSITEIT BRABANT
UTR	TARTU ULIKOOL
XLB	XLAB RAZVOJ PROGRAMSKE OPREME IN SVETOVANJE DOO
ATC	ATHENS TECHNOLOGY CENTER ANONYMI BIOMICHANIKI EMPORIKI KAI TECHNIKI ETAIREIA EFARMOGON YPSILIS TECHNOLOGIAS
ENG	ENGINEERING - INGEGNERIA INFORMATICA SPA
UST	UNIVERSITAET STUTTGART
PRQ	PRAQMA A/S

The RADON project (January 2019 - June 2021) has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825040.

Executive summary

This document presents the final version of the RADON data pipeline orchestration deliverable which extends its initial version D5.5. This deliverable presents the data pipeline related TOSCA models allowing the users to compose data intensive cloud applications using freely deploy, schedule and scale the pipeline tasks (i.e. microservices or serverless functions). In addition to that, to ensure the deployability of the cloud applications, this deliverable presents the details of the developed data pipeline plugin. The major functionalities of the RADON data pipelines such as scheduling the pipelines, encryption of the data while moving across multiple systems, data analytics tasks, etc. are discussed in this deliverable.

Glossary

DP	Data Pipeline
DPP	Data Pipeline Plugin
Pipeline Block	PB
AWS	Amazon Web Services
GCP	Google Cloud Platform
IDE	Integrated Development Environment
CLI	Command Line Interface
API	Application Programming Interface
EMR	Elastic MapReduce
FaaS	Function as a Service
GMT	Graphical Modeling Tool
XML	Extensible Markup Language
TOSCA	Topology and Orchestration Specification for Cloud Applications
YAML	YAML Ain't Markup Language
IDE	Integrated Development Environment
CDL	Constraint Definition Language
CSAR	Cloud Service Archive
DAG	Directed Acyclic Graph

Table of contents

1. Introduction

This deliverable presents the final version of methodology and orchestration of RADON data pipeline and extends its initial version (i.e. deliverable D5.5). RADON data pipeline focuses on development of a solution for the EU software industry enabling the developers to rapidly design, model and orchestrate data intensive cloud applications. The developed solution uses one commercial data management platform and one open-source data management platform. In the following sections, we present the overall data pipeline architecture, different functionalities, and the plugin that ensures the deployability of the data pipeline service blueprint.

1.1. Deliverable objectives

The main objective of this deliverable is to give an in-depth knowledge on the working principle of the RADON data pipeline including the data pipeline related TOSCA templates and the data pipeline plugin. The main objectives of this deliverable can be summarized as follows:

1. Presents the final version of the data pipeline architecture that works atop referenced data pipeline technologies.
2. Describes the integration of Apache NiFi as the open-source data management tool and AWS data pipeline as the commercial data management platform.
3. Presents the developed data pipeline plugin that ensures that the designed TOSCA service template is deployable.
4. Describes the developed data pipeline related TOSCA models.
5. Describes the following functionalities offered by RADON data pipeline.
    a. CRON based and event driven scheduling of pipelines
    b. Support the movement of data across multiple cloud environments
    c. Support the encryption of data while moving through different systems
    d. Provides specific pipelines that initiate data analytics task
6. Overall work progress in Y2

1.2. Overview of main achievements

The main contributions of the consortium can be summarized below:

1. A set of data pipeline related TOSCA node types are created using the underlined commercial and open-source data management platforms.
2. The developed TOSCA models support the schedule and movement of encrypted data across multiple cloud platforms. Further, a set TOSCA pipelines are developed that can be used to initiate the data analytics tasks.

3. To fulfill the requirement from the use case, TOSCA node types are created that allow the movement of data from one cloud platform to another, such as synchronisation of Amazon S3 bucket and Google cloud storage bucket without any dedicated serverless function.
4. As envisioned in D5.5, a plugin is developed that analyzes the service blueprint (the CSAR file), corrects the errors if any, updates the service blueprint and makes sure the deployability of the service blueprint by the RADON orchestrator.
5. With the developed RADON data pipeline, it is now possible to verify and freely compose an application combining independently deployable, schedulable, and scalable pipeline tasks, such as microservices, serverless functions, or self-contained applications.

1.3. Structure of the document

The rest of this deliverable is structured as follows:

- **Section 2** describes the detailed data pipeline architecture including the discussion of different categories of pipeline related TOSCA models.
- **Section 3** presents the RADON supported functionalities and gives an overall knowledge on how RADON data pipeline works atop other data management platforms.
- **Section 4** presents the developed data pipeline plugin along with its working principle and interactions with other RADON components such as RADON orchestrator, Template library, Graphical Modeling Tool (GMT), RADON's TOSCA repository.
- **Section 5** presents the status of each requirement related to the RADON data pipeline.
- **Section 6** gives an overall direction which will be followed even after this final deliverable and will be continued until M27.
- **Section 7** presents the concluding remarks along with a summary of level of compliances.

2. Data pipeline architecture

This section discusses the data pipeline architecture including the extended TOSCA models related to the data pipelines. Before that it is essential to discuss the underlined technologies upon which the RADON data pipeline relies on. As shown in Figure 2.1, RADON data pipeline uses one open-source data management solution, which is Apache NiFi and one commercial data management platform, i.e. AWS data pipeline. We have given the detailed introduction to both open-source and commercial data management platforms and a comparative study between both technologies in the initial version of this deliverable i.e. D5.5[1]. A very short introduction to both Apache NiFi and AWS data pipeline necessary to contextualize the developments that follow is presented below.

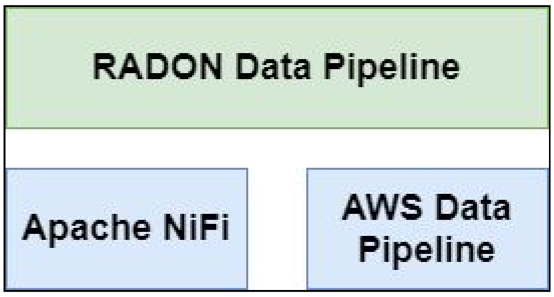


Figure 2.1. RADON data pipeline with underlined technologies

Apache NiFi

Apache NiFi [7] provides a web interface to design the flow and preprocessing of the data while moving from one cloud platform to another. This offers a number of pipeline blocks (PB) to design the data intensive cloud applications. The pipeline blocks, also called as *processors*, can be used for a variety of jobs, such as listening or querying the data from external sources and local directory structures, transforming the data, handling the files, jobs related to AWS, GCP, database related tasks etc.

Each processor receives the data from an *input port* and sends the data through the *output port*. When two NiFi processors are connected, a *queue* is created automatically to store the intermediate data/results. The queue generally follows the first-come-first-serve (FCFS) concept but can be configured to use different queueing strategies. Each data object is represented as a *FlowFile*. A FlowFile contains the path to the data, name of the data object, size of the data, a priority value, etc. When the data moves from one pipeline to another within one system, the FlowFile is actually updated, rather than moving the actual data object. This makes the flow of the data faster.

In addition to above components, NiFi provides a number of features such as the ability to schedule each processor that can be triggered by time or by specific event. A whole data flow design can be made portable by exporting (in XML format) and deploying it on another system.

Amazon Data pipeline

For handling the flow of data and preprocessing within the Amazon cloud premises, AWS data pipeline service is offered to its users [6]. DataNodes and Activity are the two main components of the AWS data pipeline. DataNode specifies the storage unit from the data that needs to be queried or to where the result needs to be written, whereas Activities specifies the operation to be performed on the data. A datanode can be dynamoDB, SqlDatanode, RedshiftDataNode, S3DataNode, etc. On the other hand, an activity can be CopyActivity, EmrActivity, HiveActivity, HiveCopyActivity, PigActivity, RedshiftCopyActivity, ShellCommandActivity, or SqlActivity. Except for ShellCommandActivity, other activities offer very limited capabilities to manipulate the data.

Similar to Apache NiFi, AWS data pipeline provides a way to schedule the pipeline using CRON. Users can specify the precondition to perform an activity. Each pipeline can be scheduled to run on either EC2 instances or EMR clusters that are entirely managed by Amazon. AWS data pipeline also allows the users to be notified upon failure of any pipeline job.

RADON data pipeline provides a unified solution that works atop two data management technologies and eliminates the requirement of in-depth knowledge on Apache NiFi and AWS data pipeline, as shown in Figure 2.2. The research results of this work are also published in [2 - 3]. This solution solely focuses on modelling of data pipeline based cloud applications using TOSCA language. Each pipeline can be independently deployable, schedulable and scalable and can be a microservice, serverless function, or self-contained application.

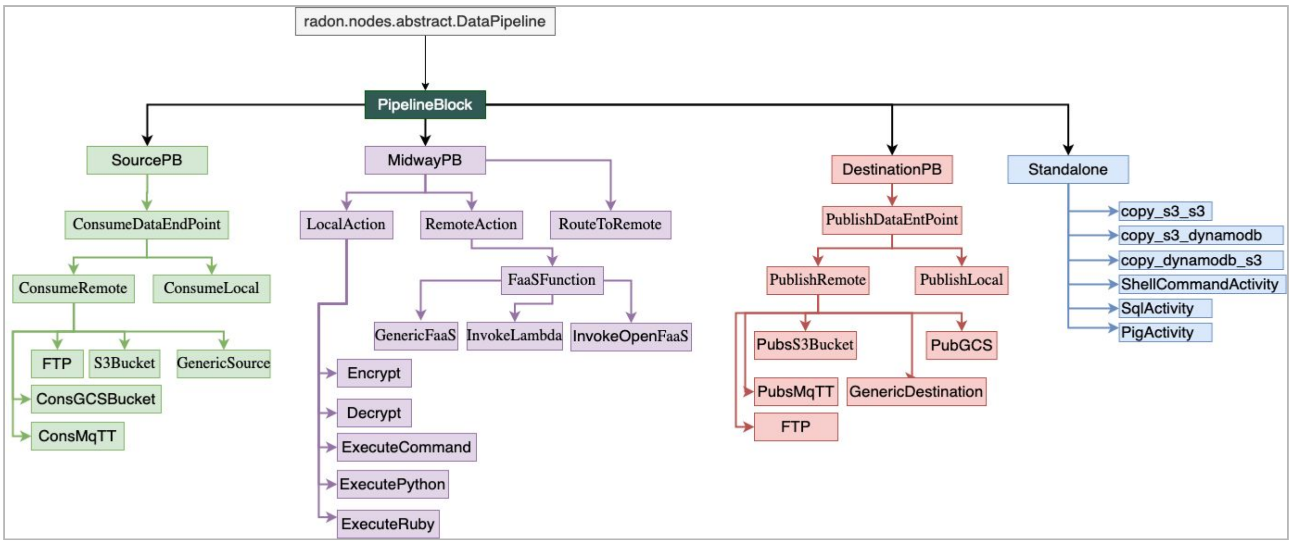


Figure 2.2: TOSCA extension for data pipeline.

The next subsections present the updated TOSCA models for RADON data pipeline. The initial version of this model tree is presented in our initial deliverable D5.5 [1]. The initial version focused on basic pipeline functionalities, such as consuming/publishing data from/to S3 bucket, invoking serverless functions like AWS Lambda, OpenFaaS function, etc. All the TOSCA-based pipeline models are using Apache NiFi as the underlined technology.

However, to fulfill the demand of being able to support the data pipelines in public cloud platforms such as AWS data pipeline, we have updated the TOSCA-based data pipeline model tree. With this updated version, a new category of TOSCA node types is introduced that focuses on performing some specific tasks such as synchronising two AWS S3 buckets, transferring data from S3 bucket to DynamoDB storage, or transferring data from S3 bucket to Google Cloud bucket and vice versa. This allows the user to move and handle the flow of the data within a specific cloud platform. In case of moving the data from one S3 bucket to another DynamoDB, the data will not move out of the AWS cloud.

In this updated TOSCA data pipeline model tree, as shown in Figure 2.2, our focus is to integrate AWS data pipeline with the existing underlined technology. This would allow the user to create a TOSCA based service model to deploy the data pipeline into AWS cloud platform. This infers that no NiFi platform is needed and no additional host machine is needed.

Irrespective of the TOSCA node type, in general the development of data pipeline based TOSCA nodes must go through the following steps:

- The data pipeline is first designed in the NiFi web interface or AWS data pipeline web console, depending upon the node type and its functionality.
- During the design phase, we decide the parameters that should have the corresponding properties name in the TOSCA node.
- We then export the data pipeline template. For NiFi based data pipeline the exported template is in XML format, whereas for AWS data pipeline, the template is in JSON format.
- Following this, the implementation scripts are created using Ansible for each lifecycle command.
- The implementation script for the create lifecycle command creates the pipeline with the given pipeline name. It then returns a unique ID to the respective attribute of the pipeline node.
- The implementation script for configuration command updates the data pipeline template with the input properties values.
- The implementation scripts for a start, stop, and delete lifecycle commands use the unique pipeline ID to start/activate, stop and delete/deactivate the pipeline.
- Once the implementation scripts are ready, we now design the required node type in RADON GMT. At this stage, properties, attributes, parent node type, requirements, and capabilities are defined for the Tosca node.

2.1. SourcePB

As discussed in D5.5[1], the TOSCA node types that are derived from *SourcePB* node type mainly focus on consuming data from local or external storage units or data sources. In the previous deliverable we had developed *ConsumeLocal and ConsS3bucket* node types to consume the data from local and AWS S3 buckets. *ConsumeLocal* node type is derived from *ConsumeDataEndPoint* and is used to consume the data from the local directory structure. The mandatory input while creating this node type is the name of the *directory*. To fetch the object from AWS S3 bucket *ConsS3Bucket* TOSCA node type was created, which requires the essential information such as name of the bucket, region of the S3 bucket and the path to the credential file.
In this deliverable we have developed the following TOSCA node types to consume the data from external storage units.

ConsGCSBucket: This node type is used to consume the data from Google Cloud storage bucket. This requires three inputs as given in Listing 2.1. This node type assumes that the given project and the bucket are already created.

```
tosca_definitions_version: tosca_simple_yaml_1_3
node_types:
  radon.nodes.datapipeline.source.ConsGCSBucket:
   derived_from: radon.nodes.datapipeline.source.ConsumeRemote
   properties:
     bucket:
       type: string
     project_ID:
       type: string
     credential_JSON_file:
       type: string
       description: Path of the credentials in the form of JSON file
```

Listing 2.1. TOSCA node type for consuming data from a Google Cloud storage bucket

ConsMQTT: To fulfill the requirement of TOSCA node types related to messaging transport, we have developed *ConsMqTT*[1] TOSCA nodetype that allows the user to consume the message from a specific URL over TCP/IP protocol. The essential properties are listed below in Listing 2.2. Such node type assumes that the MqTT broker is already running. To install, configure and run MqTT broker we have developed *MosquittoBroker*[1] TOSCA node type.

```
tosca_definitions_version: tosca_simple_yaml_1_3
```

[1] https://github.com/radon-h2020/radon-particles/pull/64

```
node_types:
  radon.nodes.datapipeline.source.ConsMQTT:
    derived_from: radon.nodes.datapipeline.source.ConsumeRemote
    properties:
      Broker_URI:
        type: string
        default: "tcp://localhost:1883"
      Max_Queue_Size:
        type: integer
      Quality_of_Service:
        type: integer
      Client_ID:
        type: string
      TopicName:
        type: string
```

Listing 2.2. TOSCA node type for consuming MqTT messages from remote broker

In addition to the above developed TOSCA node types, we will continue development of other SourcePB node types (e.g to consume data from remote FTP servers, databases, data brokers) as part of our future work to extend the number of reusable data pipeline components in the final RADON solution.

2.2. MidwayPB

MidwayPB mainly focuses on the processing of data while moving through a data pipeline across multiple cloud platforms. Data can be processed either by invoking a local script or a remote serverless function. In this deliverable, we have developed following TOSCA node types:

InvokeOpenFaaS[2]: This node type is used to invoke an OpenFaaS function. The essential properties of this node type are the URL of the function, the invocation method, and the type of the input content. The default function invocation method is “POST”. This assumes that the OpenFaaS function is already deployed and configured.

Encrypt[3] and *Decrypt*[3] : For the purpose of data encryption and decryption, we have developed *Encrypt* and *Decrypt* TOSCA node types that are derived from *radon.nodes.datapipeline.process.LocalAction* node type. In the current form, these two node types accept a password from the user that will be used in the encryption and decryption process. The main concern here is that the user needs to provide the same password for each pair of Encrypt and Decrypt TOSCA nodes. However in case of password mismatch, the developed data pipeline

[2] https://github.com/radon-h2020/radon-particles/pull/63

plugin will generate a random password and assign it to the *Encrypt* and *Decrypt* nodes. These node types are mainly useful when the data moves across cloud platforms. The detailed description with an example is given in Section 3.3.

A variety of other node types are under development to process the data locally using different script engines such as Python, Ruby etc. *ExecuteCommand* TOSCA node type is under development that can be used to issue a general terminal command with a set of arguments.

2.3. DestinationPB

This category of TOSCA node types fulfill the requirements to publish the final or intermediate result to local or external storage units, as shown in Figure 2.2. The data can be published to local directory structure using *PublishLocal* TOSCA node type. Similarly, users can use *PubsS3Bucket* TOSCA node types to push the result to S3 bucket. The developed *PublishLocal* and *PubsS3Bucket* node types were presented in the initial deliverable D5.5. In this deliverable we have developed TOSCA node types to publish the data to Google Cloud storage (*PubGCS*), publish the message using MqTT protocol.

PubGCS: This node type is used to publish the data to Google Cloud storage bucket and act as the counterpart of *ConsGCSBucket* TOSCA node type. Similar to *ConsGCSBucket*, the user needs to give the name of the bucket, project id and the path to the credential file information to *PubGCS* TOSCA nodes. The TOSCA node of type *PubGCS* needs the bucket to be present before pushing any data. However, if no bucket is present, *PubGCS* node type will continue trying to push the data in a regular interval of time as per the scheduling information.

```
tosca_definitions_version: tosca_simple_yaml_1_3
node_types:
  radon.nodes.datapipeline.destination.PubGCS:
        derived_from: radon.nodes.datapipeline.destination.PublishRemote
        properties:
          BucketName:
            type: string
          cred_file_path:
            type: string
          ProjectID:
            type: string
```

Listing 2.3: TOSCA node type for publishing data to Google Cloud storage bucket

PubsMQTT: This node type is the counterpart of the *ConsMQTT* TOSCA node type, which is used to publish the message using MqTT protocol. The properties of the *PubsMQTT* node type are the same with that of the *ConsMQTT* node type. The prerequisite to this node type is that the users need to deploy and run the MqTT broker using *MosquittoBroker* TOSCA node.

To provide more options to publish the data, we are currently developing different TOSCA node types, such as for publishing data to remote FTP server, remote databases etc.

2.4. Standalone pipeline

We have introduced this category to facilitate the users with a set of TOSCA node types related to AWS data pipeline. In the second year, we have investigated and developed the models to deploy the AWS data pipeline into Amazon data pipeline platform. Amazon data pipeline offers a set of operations or activities, as discussed in previous sections. Keeping those set of AWS data pipeline activities in mind, we have planned to develop a set of TOSCA node types under *Standalone* category, as shown in Figure 2.2.

copy_s3_s3[3]*:* We have developed a Copy_s3_s3 TOSCA node type that allows the user to synchronise two AWS S3 buckets without using any dedicated serverless function for the synchronization purpose. The current form of this node type does not need the TOSCA node type for EC2 instance[4] or OpenStack instance[5]. Further, this also does not need a NiFi platform unlike other data pipeline node types. Creating a TOSCA node of this type would require the source and destination S3 bucket. If a user needs a specific file to be copied, the file_name properties can be used. The user also needs to give the bucket name and the directory name to keep all the logs. *schedule* properties is used to provide the scheduling information using CRON language.

```
tosca_definitions_version: tosca_simple_yaml_1_3
node_types:
 radon.nodes.datapipeline.Standalone.copy_s3_s3:
      derived_from: radon.nodes.datapipeline.Standalone
      attributes:
        pipeline_id:
           description: ID of the pipeline
           type: string
      properties:
         log_directory:
           type: string
         schedule:
           type: string
```

[3] https://github.com/radon-h2020/radon-particles/pull/63
[4] https://github.com/radon-h2020/radon-particles/tree/master/nodetypes/radon.nodes.VM/EC2
[5] https://github.com/radon-h2020/radon-particles/tree/master/nodetypes/radon.nodes.VM/OpenStack

```
        configure_file_path:
          type: string
        dp_name:
          type: string
        credential_file_path:
          type: string
        destination_bucket:
          type: string
        file_name:
          type: string
        source_bucket:
          type: string
        log_bucket:
          type: string
```

Listing 2.4: TOSCA node type for deploying AWS data pipeline for synchronisation of two S3 buckets.

configure_file_path is mainly used to provide the region information. So this nodetype assumes that source bucket, destination bucket and the bucket for logs are available in the same region as mentioned in the configuration file present in *configure_file_path* path. A user also needs to provide the path to the credential file to create, deploy and activate a AWS data pipeline. The credential file should contain the access key and secret access key in the following format, as mentioned in Listing 2.5.

```
[default]
accessKey= enter your access key here
secretKey= enter your secret key here
```

Listing 2.5: Credential file format to create, deploy and activate AWS data pipeline.

Upon deployment of *Copy_s3_s3* TOSCA node, AWS will create and configure a new EC2 instance with required instance flavor and the software packages. The major disadvantage of such a node type is that the creation of a new EC2 instance and the cost associated with that instance may not be visible to the user at TOSCA level.

Another major disadvantage of AWS data pipeline is its ability to integrate with other cloud environments. For example, It is difficult for the user to integrate the OpenFaaS serverless function or Google Cloud Function to the databases provided by AWS, such as MariaDB. Further, AWS data pipeline itself provides minimal access to AWS specific databases. For instance, for CopyActivity operation, the input datanode can be only S3 bucket, or Dynamodb, or Redshift datanode. But it is not possible to define a data pipeline that copies the data from an S3 bucket to a MariaDB. AWS Lambda is also not yet integrated into the AWS data pipeline service. Further, it is not possible to bring the data from third-party services and integrate with the existing AWS datanodes.. The only way to integrate with other unsupported external/internal data sources or other serverless functions, is by issuing a shell command that runs on AWS EC2 instance. For this we have developed *AWSShellCommand* TOSCA node type as described below.

AWSShellCommand[6]: This TOSCA node type can be used to create, deploy and activate an AWS data pipeline with *AWSShellCommandActivity* operation. With this, the user can run a shell command via the linux terminal. This will allow the user to trigger/invoke a Lambda function or a remote serverless function, extract data from an external data source etc. The required command needs to be given to the *aws_cli_command* properties of *AWSShellCommand* TOSCA node, as shown in Listing 2.6. In addition to this, the user needs to provide the S3 bucket name (using *log_bucket* and *log_directory* properties) to keep the logs. Similar to *copy_s3_s3* TOSCA node type, this *AWSShellCommand* node type also has *schedule*, *configure_file_path*, and *credential_file_path* properties with the same purposes.

```
tosca_definitions_version: tosca_simple_yaml_1_3
node_types:
  radon.nodes.datapipeline.Standalone.AWSShellCommand:
    derived_from: radon.nodes.datapipeline.Standalone
    properties:
      log_directory:
        type: string
        description: The directory name in the log_bucket.
      schedule:
        type: string
        description: The scheduling info using CRON syntax
      aws_cli_command:
        type: string
        description: command to be executed on AWS CLI
      dp_name:
        type: string
        description: name of the data pipeline
      configure_file_path:
        type: string
        description: AWS configuration file
      credential_file_path:
        type: string
        description: Path to the AWS credential file.
      log_bucket:
```

[6] https://github.com/radon-h2020/radon-particles/pull/63

```
    type: string
    description: The bucket name where logs will be kept
```

Listing 2.6: Essential properties of AWSShellCommand TOSCA node type.

AWS data pipeline deployment process:

Here we will discuss how the *Standalone* TOSCA node types are created, deployed and activated taking *AWSShellCommand* TOSCA node type as an example. The same process will also be followed in future especially while creating the TOSCA node types that fall under *Standalone* category.

The design of AWS data pipeline specific TOSCA node type includes the development of different components. For each such type of TOSCA node, AWS data pipeline is designed, activated and tested using the AWS provided web interface. In this stage, we also decide what parameters should be made user-defined. For example, source S3 bucket, destination S3 bucket, log bucket/directory, scheduling parameter, etc are made user-defined and hence the value of such parameters need to be provided through the corresponding properties of the *copy_s3_s3* TOSCA node. Once tested, the AWS data pipeline is exported in JSON format.

After a JSON file is obtained, the implementation files for the lifecycle commands are created. The implementation files for *create*, *start*, *stop* and *delete* operations of all the AWS data pipeline TOSCA node types are the same. Except for a *create* command, implementation files for other lifecycle commands need only the unique ID of the AWS data pipeline as input. The unique ID is created during the execution of *create* lifecycle command.

The create operation is implemented using a create Ansible script (*create.yml*). Here, an empty AWS data pipeline is created with the name as given in *dp_name* property of the TOSCA node. Once the pipeline is successfully created, an unique ID is returned by AWS and set to the *pipeline_id* attribute of the corresponding TOSCA node.

In configuration Ansible script (*configure.yml*), an AWS CLI command is issued with the previously obtained pipeline id and all the required parameter values given in the TOSCA node to define the previously created pipeline. In case of *AWSShellCommand* TOSCA node following AWS CLI command is issued.

```
aws datapipeline put-pipeline-definition --pipeline-id {{pipeline_id}}
   --parameter-values myAWSCLICmd={{aws_cli_command}}
         myLogBucket={{log_bucket}}/{{log_directory}}
         mySchedule="{{edit_schedule}}"
   --pipeline-definition file://ShellCommandActivity.json
```

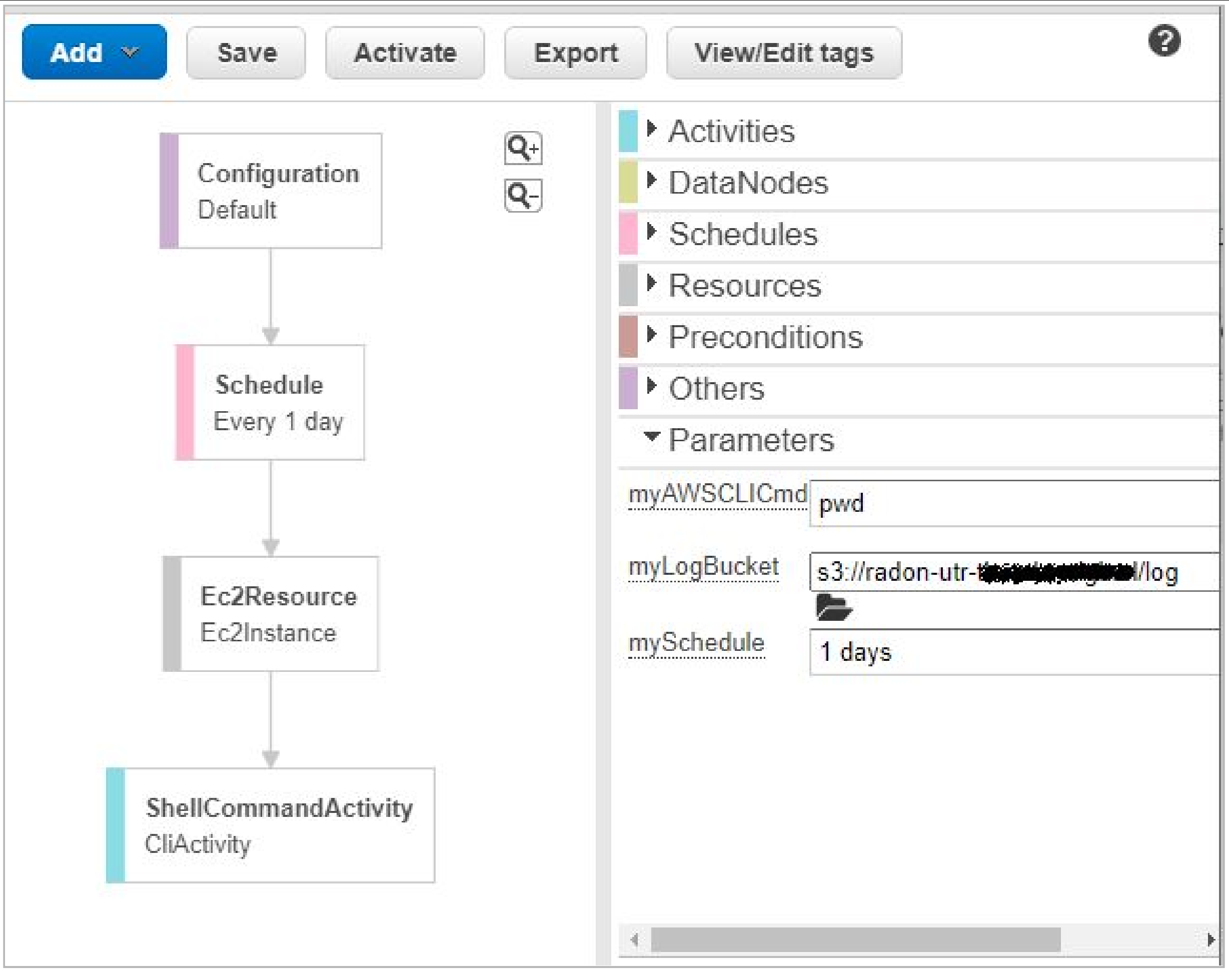
Listing 2.7: A code snippet to edit AWS data pipeline definition

The above code snippet assigns the values to the corresponding parameters. The list of parameters are created in the pipeline definition file, which is in the JSON format and is very specific to the node types. In the *AWSShellCommand* example, following parameters are created in the JSON template.

```
{
"parameters": [
   {
    "id": "myAWSCLICmd",
    "watermark": "aws [options] <command> <subcommand> [parameters]",
    "description": "AWS CLI command",
    "type": "String"
   },
   {
    "id": "myLogBucket",
    "description": "S3 Log Bucket name",
    "type": "AWS::S3::ObjectKey"
   },
   {
    "id": "mySchedule",
    "description": "Scheduling info",
    "type": "String"
   }
 ]
}
```

Listing 2.8: Parameters for *ShellCommandActivity* in *AWSShellCommand* TOSCA node type

Once the AWS pipeline is created and configured, the list of parameters and the resources can be seen in the AWS data pipeline console. Below, we have given a screenshot that shows the list of parameters created for *AWSShellCommand* TOSCA node.



myAWSCLICmd: pwd
myLogBucket: s3://radon-utr-[redacted]/log
mySchedule: 1 days

Figure 2.3: Screenshot of *AWSShellCommand* AWS data pipeline after its creation and configuration.

pipeline_id attribute is further given as input to the respective Ansible script of *start*, *stop* and *delete* operations to activate, deactivate and delete the AWS data pipeline, respectively. It is to be noted that, for all AWS data pipeline related node types, which are under *Standalone* category (Figure 2.2), a dedicated template is created in JSON format.

After designing the AWS data pipeline template and the respective implementation files, it is now time to design the TOSCA node type using RADON GMT with all the implementation files and the template as artifacts. The detailed description on how to design a TOSCA node can be found in Deliverable D4.5 “Graphical modelling tool I” [9] and Deliverable D4.6, which is scheduled to be published.

In the same way, we have also developed *copy_s3_dynamodb* and *copy_dynamodb_s3* TOSCA node types that focus on copying data between S3 bucket and DynamoDB. In future we will also develop TOSCa node type for executing SQL command and Pig command using *SQLActivity* and *PigActivity* TOSCA node type.

3. Data Pipeline Functionalities

In this section, we will discuss the functionalities provided by the RADON data pipeline such as the support for scheduling, encryption, data analytics tasks and cross platform data movement. These functionalities are developed in YR2 fulfilling different requirements as mentioned in next subsections.

3.1. Data pipeline Scheduling

To fulfill the requirement R-T5.4-2 and R-T5.4-3 that allows the users to schedule the data pipelines, either based on the event or based on the CRON, we have developed all the TOSCA node types with such functionality. All the TOSCA nodes will be triggered either on a specified time or by an event. The event could be an arrival of a new object into the database, modification of an existing data object, deletion of record from the database, arrival of a request to update/access the database, etc. Currently, all the TOSCA nodes support event-driven scheduling. This functionality is inherited from the scheduling feature of an underlined Apache NiFi. This applies to the TOSCA node types that are solely based on NiFi.

This functionality is implemented at the top level TOSCA nodetype *PipelineBlock*. For this, two properties are introduced: *schedulingStrategy* and *schedulingPeriodCRON.* The *schedulingStrategy* property is of string type and the possible values are EVENT_DRIVEN and TIMER_DRIVEN. The default scheduling strategy is EVENT_DRIVEN, which indicates that it is optional for the users to give any specific value to this property. However, to schedule a data pipeline on a specific time interval, *schedulingStrategy* should be set to TIMER_DRIVEN. In such scenarios, a user needs to provide the detailed scheduling information to the *schedulingPeriodCRON* property. The scheduling time should be given using CRON syntax. In case of missing the *schedulingPeriodCRON* property value, the default value will be used, which is *"* * * * * ?".* Such scheduling strategy and CRON values will be updated during the configuration state of the TOSCA nodes by the respective *configure.yml* Ansible script.

```
tosca_definitions_version: tosca_simple_yaml_1_3
node_types:
  radon.nodes.datapipeline.PipelineBlock:
    derived_from: radon.nodes.abstract.DataPipeline
    properties:
      schedulingStrategy:
        type: string
        description: Either EVENT_DRIVEN (default) or CRON_DRIVEN.
EVENT_DRIVEN is similar to TIMER_DRIVEN with 0 sec value in NiFi.
        required: false
        default: "EVENT_DRIVEN"
      schedulingPeriodCRON:
        type: string
        description: For CRON_DRIVEN give in CRON syntax.
        required: false
        default: "* * * * * ?"
```

Listing 3.1: *schedulingStrategy* and *schedulingPeriodCRON* properties of TOSCA data pipeline node types

The cron expression comprises 7 fields, out of which one field can be set as “no specific value” using a question mark (“?”) symbol. The fields are seconds, minutes, hours, day of month, month, day of the week, and year. A cron expression “* * * * * ?” means run the job always. Another example can be “0 15 10 15 * ?” for scheduling the job to run at 10:15 on the 15th day of every month. The detailed CRON syntax can be found in the official Cron GNU project [10].

To schedule the AWS data pipeline such as *copy_s3_s3*, *AWSShellCommand*, *copy_s3_dynamodb*, etc. a user needs to give the value to the *schedule* property of respective TOSCA nodes.

3.2. Cross platform data movement

The requirement mentioned in R-T5.4-5, indicates the functionality of movement of the data from one cloud to another cloud using TOSCA data pipeline nodes. The current models allow the users to move the data between Amazon S3 bucket, Google Cloud storage and private OpenStack cloud. Below, we have given two examples of the design of TOSCA service templates using RADON GMT that (1) move the data from Amazon S3 bucket to Google Cloud storage, as given in Figure 3.1, and (2) moves the data from private cloud instance’s directory to Amazon S3 bucket and Google Cloud storage, as given in Figure 3.2. Figure 3.1 and 3.2 show how the design of data pipeline based service templates looks in RADON GMT.

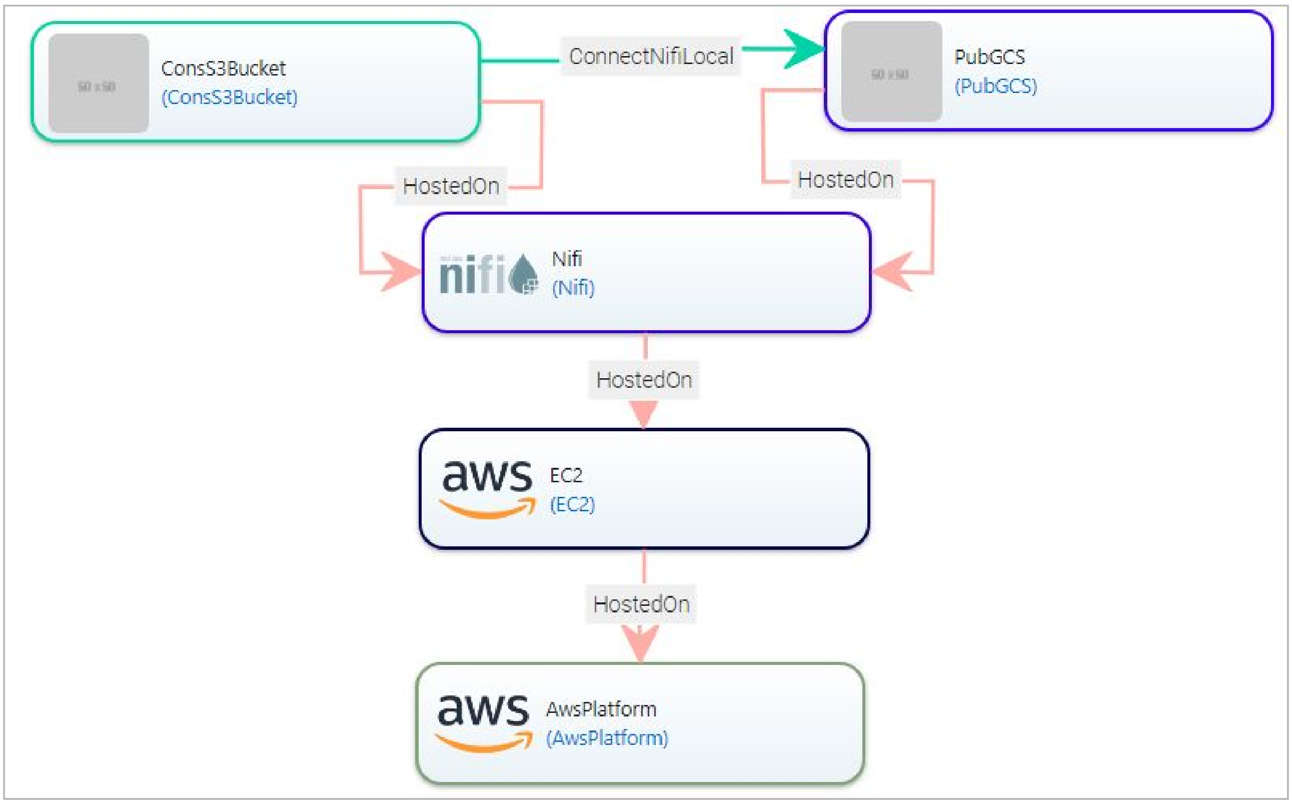


Figure 3.1: Data pipeline based service template that moves data from an Amazon S3bucket to a Google cloud storage.

As the example given in Figure 3.1, the data pipeline service template creates an EC2 instance atop *AWSplatform*. Apache NiFi platform will be installed and configured to execute TOSCA data pipeline nodes. Here, *ConsS3Bucket* TOSCA node type is used to consume the data from Amazon S3 bucket. For this, users need to provide the bucket name, the credential information and the region of the bucket. Similarly, another TOSCA data pipeline node *PubGCS* is used to publish the data to a specified Google Cloud storage bucket. For *PubGCS* node type, users need to provide the bucket name, credential information and the projectID. Using these two node types, users will be able to move the data from Amazon cloud to Google Cloud. *ConsS3Bucket* node type can be scheduled to run only when a new data object is added to the S3 bucket. This can also be scheduled to run once in a day allowing the pipeline to idle at other times.
On execution of the above example in Figure 3.1, *ConsS3Bucket* TOSCA node reads the data from the specified S3 bucket and first writes the data onto a NiFi-specific directory in EC2 instance. The data is then read by the *PubGCS* node type and published to the given Google Cloud storage bucket. Such a service template can be used to take the backup of all the data in the Amazon S3 bucket on a regular interval of time without designing and implementing any dedicated serverless function in between *ConsS3Bucket* and *PubGCS*.

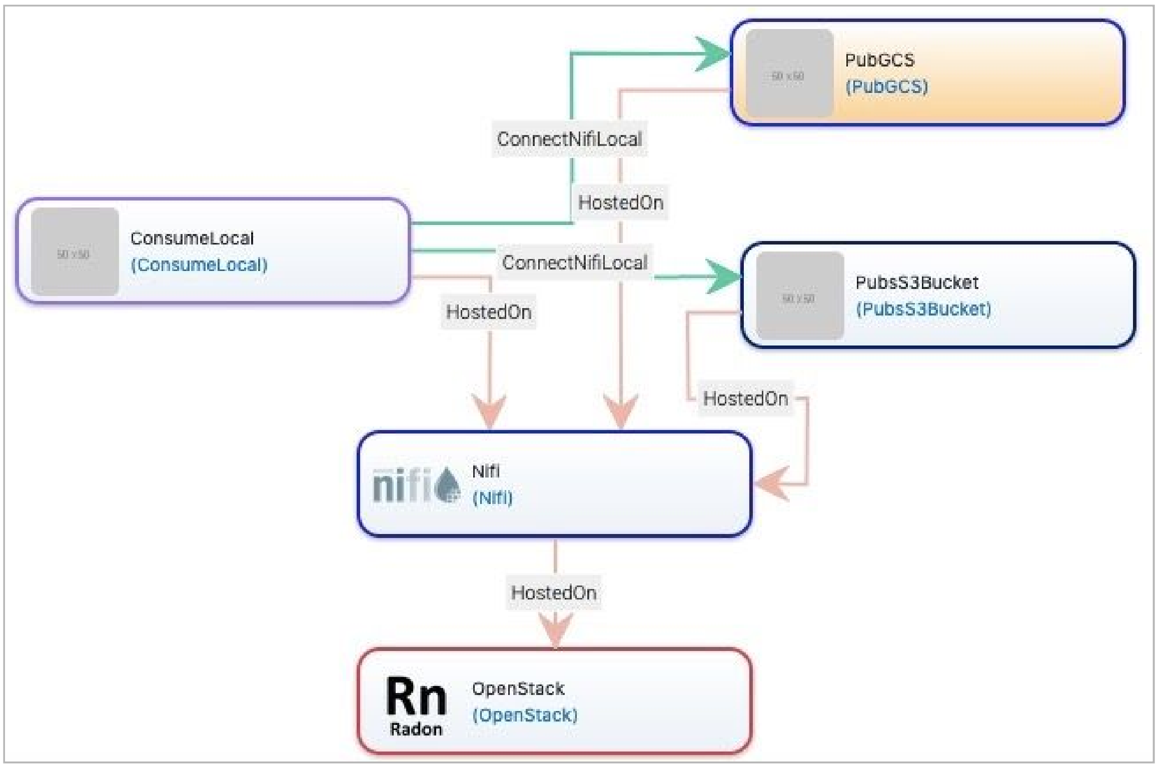


Figure 3.2: Data pipeline based service template that moves data from private cloud instance's directory to Amazon S3 bucket and Google Cloud storage

Another example of cross cloud platforms data movement is given in Figure 3.2. The example demonstrates the movement of data from the private cloud platform, such as OpenStack cloud to the public cloud platforms: Amazon S3 bucket and Google Cloud storage. In the example (Figure 3.2), an OpenStack instance is created and the Apache NiFi platform is installed and configured. Atop the NiFi, three TOSCA data pipeline nodes are created: *ConsumeLocal*, *PubGCS*, and *PubsS3Bucket*. The *ConsumeLocal* TOSCA node with its properties and requirements is given below. This node consumes/reads the data from the given directory "*home/data/*" in the OpenStack instance using the *GetFile* NiFi processor. The consumed data then moved to the NiFi flowfile repository.

```
ConsumeLocal_0:
    type: radon.nodes.datapipeline.source.ConsumeLocal
    properties:
      directory: "/home/data/"
    requirements:
      - connectToPipeline:
          node: PubGCS_0
          relationship: con_ConnectToPipeline_0
          capability: ConnectToPipeline
```

```
  - connectToPipeline:
      node: PubsS3Bucket_0
      relationship: con_ConnectToPipeline_1
      capability: ConnectToPipeline
  - host:
      node: Nifi_0
      relationship: con_HostedOn_0
      capability: host
```

Listing 3.2: ConsumeLocal TOSCA node for the service template given in Figure 3.2

Listing 3.3 and 3.4 gives the *PubsS3Bucket* and PubGCS TOSCA nodes with essential properties. *PubsS3Bucket* TOSCA nodetype creates *PutS3Object* NiFi processor and configures with the values of *BucketName*, *Region* and *cred_file_path* properties. The *ConnectToPipeline* relationship establishes a connection form the *Output port* attached to GetFile NiFi processor to the *Input port* attached to *PutS3Object* NiFi processor. With that connection, the consumed data is automatically forwarded to the *PutS3Object* NiFi processor, which pushes each data to the given *RADON-utr* S3 bucket.

The consumed data are then forwarded simultaneously to the Google Cloud storage (using *PubGCS* data pipeline TOSCA node type) and to Amazon S3 bucket (using *PubsS3Bucket* data pipeline TOSCA node type).

```
PubsS3Bucket_0:
    type: radon.nodes.datapipeline.destination.PubsS3Bucket
    properties:
      cred_file_path: "/tmp/.gcp/credential"
      BucketName: "RADON-utr"
      Region: "eu-central-1"
```

Listing 3.3: PubsS3Bucket TOSCA node for the service template given in Figure 3.2

Similarly, *PubGCS* TOSCA node type, as given in Listing 3.4, creates *PutGCSObject* NiFi processor and configures with the values of *BucketName*, *ProjectID*, and *cred_file_path* properties. As *the ConsumeLocal* TOSCA node is connected to *PubGCS* with the *ConnectToPipeline* relationship type, a connection from the underlined *GetFile* NiFi processor to *the PutGCSObject* NiFi processor will be established. NiFi will automatically forward the consumed data to the *PutGCSObject* NiFi processor, which eventually pushes the data to the given *RADON-utr* bucket available under *Project1* GCP project. This demonstrates the flow of data among one private cloud platform and two public cloud platforms.

```
PubGCS_0:
    type: radon.nodes.datapipeline.destination.PubGCS
```

```
properties:
  cred_file_path: "/tmp/.gcp/credential"
  BucketName: "RADON-utr"
  ProjectID: "Project1"
```

Listing 3.4: PubGCS TOSCA node for the service template given in Figure 3.2

3.3. Data encryption

To fulfill the requirement R-T5.4-7 stating the ability to encrypt data while moving from one system to another, we have created specific TOSCA node types that implement encrypting and decrypting the data at source and destination sides, respectively. Such functionality is needed mainly when the data moves from one cloud platform to another cloud. For example, as given in Figure 3.3, the data is moving from Amazon cloud to Google cloud. In such a scenario, it might be possible to access the data while it's moving through the internet, so it's important to make sure the data is sent over an encrypted channel or data objects themselves are encrypted. To mitigate this issue, *Encrypt* and *Decrypt* data pipeline[7] TOSCA node types are introduced to give users control over data encryption in the data pipeline. Figure 3.1 can be modified with the encryption functionality, as given in Figure 3.3.

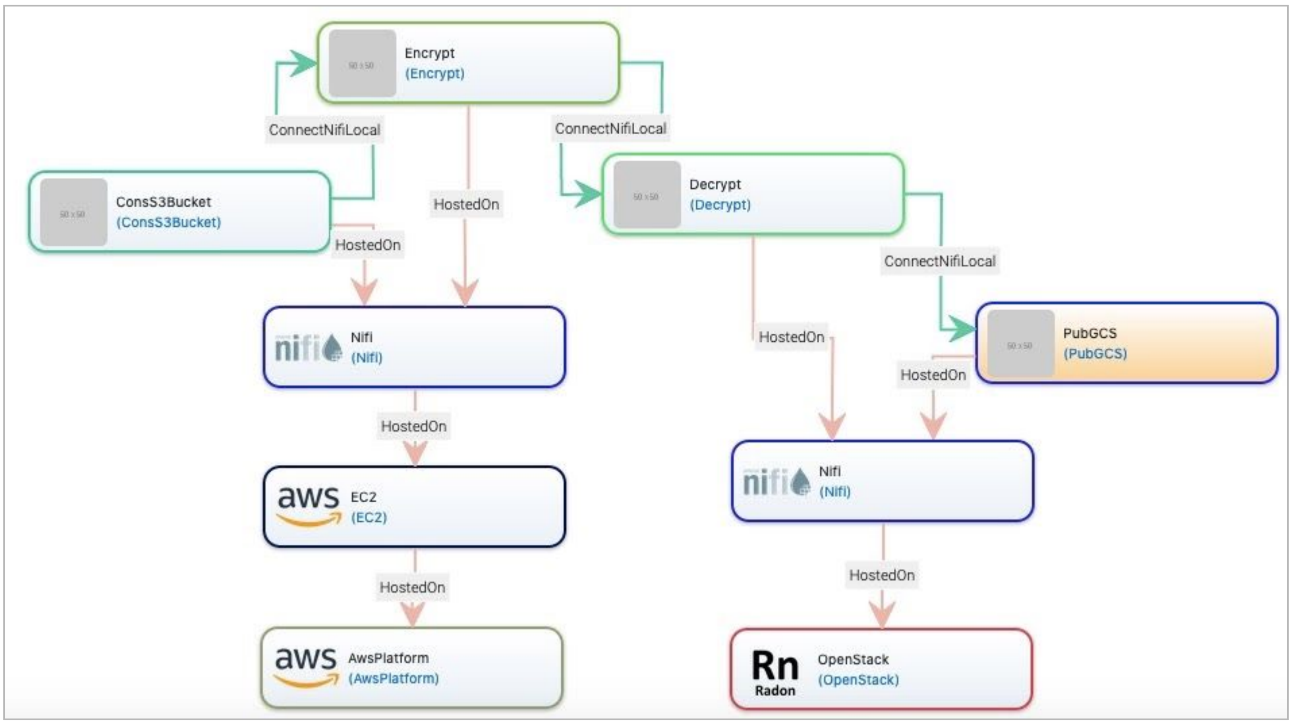


Figure 3.3: Data pipeline based service template that moves the encrypted data from Amazon S3bucket to Google cloud storage.

[7] https://github.com/radon-h2020/radon-particles/pull/63

Encrypt and *Decrypt* TOSCA node types are derived from *LocalAction* data pipeline TOSCA node type. Table 3.1, gives the list of essential configurations associated with *Encrypt* and *Decrypt* TOSCA node type.

Configuration	**Value**
Password	Given by user or Randomly generated by DPP
Key-derivation-function	OPENSSL_EVP_BYTES_TO_KEY
Encryption Algorithm	MD5_128AES

Table 3.1: Essential configurations and their values used in Encrypt and Decrypt TOSCA node

Encrypt TOSCA node type is used at the data consumption side to encrypt the data either using the user given password while designing the service template or using a randomly generated password. Similarly, on the other hand, *Decrypt* data pipeline TOSCA node type is used at the destination side to decrypt the received encrypted data using the same password. The *Decrypt* and the *Encrypt* TOSCA nodes can also be used in between any two *MidwayPB* data pipeline nodes. MD5 (or message-digest version 5) algorithm with digest 128 bits is used in the entire process of encryption and decryption of data. *The EVP_BytesToKey* algorithm of openSSL is used to derive the keying material for the encryption algorithm. With the current version of the *Encrypt* and *Decrypt* TOSCA node types, *Key-derivation-function* and *Encryption Algorithm* configuration values are fixed. However, it is in the part of future development to allow the users to choose from a set of options for encryption algorithms and the Key Derivation functions.

In the current version, we are allowing users to provide the password to the *Encrypt* and *Decrypt* nodes. However, the user needs to provide the same password. DPP is equipped with the feature to ensure that the same password is given to each *Encrypt* and *Decrypt* node pair. In case of password mismatch, the DPP generates a random password and assigns to the password properties of each pair of Encrypt and Decrypt nodes in the service template. The detailed information is given in Section 4..

As a part of our future plan, the password for each *Encrypt* and *Decrypt* node pair will be provided only by DPP. Hence, the potential error in password mismatch can be eliminated.

3.4. Data Analytics tasks

This functionality of the data pipeline enables the users to process the intermediate data by initiating the data analytics tasks. With the current version of the developed data pipeline TOSCA models, data analytics tasks can be implemented as serverless functions andRADON data pipeline provides already-developed AWSLambda and OpenFaaS TOSCA node types to invoke the remote data analytics tasks. For example, performing face recognition on an image or translating Twitter messages.

To extend the current features, additional data pipeline TOSCA node types will be created focusing on extraction, transformation and routing of the data. With the current plan, different data pipeline TOSCA node types will be developed using which specific external data analytics tasks can be invoked and executed using Python (*ExecutePython* TOSCA node type) or Ruby (*ExecuteRuby* TOSCA node type) script engine. Users can also issue a data analytics related command with required arguments in the terminal using *ExecuteCommand* TOSCA node type.

However, to make the application development process faster, specialised data pipeline node types will be developed focusing on the specific tasks. One of such types can be Amazon specific TOSCA node types. Such node types can be used to execute SQL query, Pig query, or Hive query on Amazon S3 buckets. With this users can issue the command/query inside the TOSCA service blueprint and specify the input and output S3 bucket upon which the query will be executed. Another motivation to such Amazon specific TOSCA node types can be to provide functionality to easily define data analytics tasks which consume data from S3, launch an Elastic MapReduce (EMR) cluster automatically and allow users to define SQL or Pig queries to process very large data.

4. Data Pipeline plugin

This section presents the outcome of envisioned data pipeline plugin as discussed in Deliverable D5.5: Data pipeline orchestration I [1]. In the previous sections and the D5.5, we have discussed the methodology and the orchestration of the data pipeline with RADON thumbnail generation example. While improving the methodology, the necessity of DP plugin is observed that can fix the potential common errors present in the data pipeline service template designed by the user in GMT.

It was envisioned in D5.5 [1] that the RADON CDL will be equipped with a set of DP rules which will be checked against the exported CSAR and based on the verification the DP plugin will update the CSAR before it is passed to the orchestrator for deployment. The major responsibility of CDL would be to only check a set of rules, suggest corrections to the RADON model and not to update the CSAR, which is beyond the scope of CDL [4]. To handle this issue, we came up with the data pipeline plugin[5] that would be responsible for both verifying the consistency, updating the CSAR (if required) and making sure that the data pipeline based CSAR is deployable. In this way, we can eliminate the requirement to invoke the CDL tool and develop the pipeline-based service blueprint preparation faster.

4.1. Interaction of DPP with other RADON tools

Figure 4.1 shows the connection of the data pipeline plugin with other RADON tools. User designs the pipeline based cloud service using RADON's graphical modelling tool. Upon export of the service blueprint/template in TOSCA CSAR format, the data pipeline plugin is invoked, taking the CSAR as an input. The CSAR is then parsed and checked if there is any inconsistency or if any error made by the developer still exists.

Two pipelines can be connected and the relationship between two pipelines can either be local or remote. Local relationship refers to the deployment of corresponding pipelines on the same virtual machine. On the other hand, remote relationship infers that two pipelines are on different virtual machines. The virtual machines can be in private or public clouds. During the design phase it is not possible to check if the underlined virtual machines are the same or different and the type of the relationship entirely depends on the users. Hence, it is possible that the user may export a service template with the wrong relationship type, which may not be possible for the orchestrator to orchestrate and establish the relationship between pipeline nodes. To handle this issue, the data pipeline plugin will be responsible for making sure that the correct relationship is used between two connected pipeline-based TOSCA nodes.

Flow of the encrypted data among multiple cloud systems is another situation, where DPP plays a major role. Encryption feature of data pipeline would encrypt the data before sending to another pipeline at the source side and also will decrypt the received encrypted data. The encryption and decryption will be performed using a randomly generated passphrase. In the design phase, users need to enable this feature at both source and destination pipeline or need to add the *Encrypt* and *Decrypt* TOSCA node type. Failing to do so, users may send the data without proper encryption or may receive the encrypted data. To avoid such a situation, the data pipeline plugin needs to make sure that the feature in both the source and destination pipelines is the same.

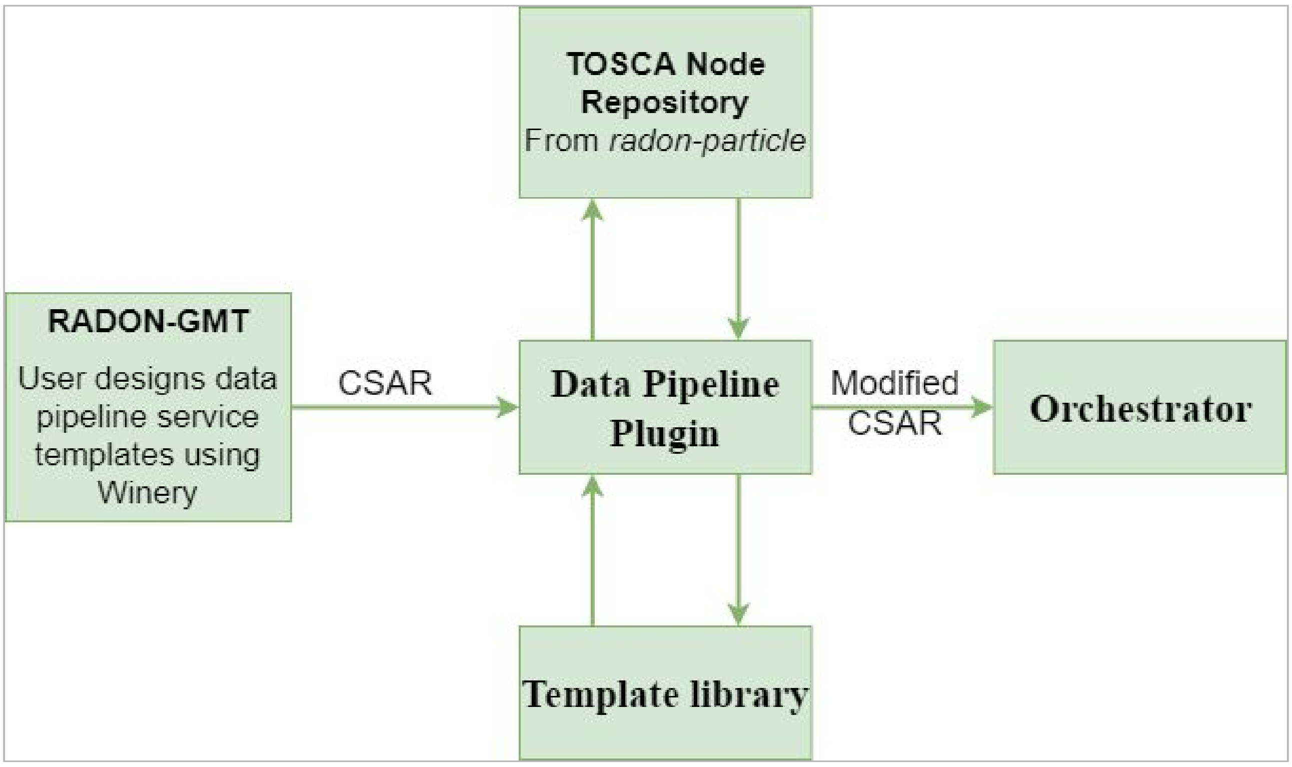


Figure 4.1: Data pipeline plugin with other RADON components/tools

4.2. Execution process

This section presents the internal working procedure of data pipeline plugin and explains the steps to ensure the orchestrate-ability of the CSAR by the orchestrator.

1. For each input CSAR, the data pipeline plugin first unpacks the CSAR file and finds the required YAML file that contains the definition of the TOSCA node topology and relationship topology.
2. We are using the TOSCA parsing module[8] of the RADON orchestrator to analyse the DAG of all the nodes and their relationship types.

Ensuring correct relationship type:

3. After parsing, DPP first traverses through all the TOSCA nodes in the CSAR that have a *host* and a *ConnectToPipeline* requirement.

[8] https://github.com/radon-h2020/xopera-opera/tree/master/src/opera/parser

4. With the *ConnectToPipeline* requirement, DPP extracts all the pairs of TOSCA nodes and checks if the adjacent data pipeline nodes are on different hosts as given in the following code snippet.

```
def get_nodelist_to_edit(self, node_list):
   for connect in range(len(self.connection_names)):
         if self.connection_names[connect] != 'no':
                  connection = self.connection_names[connect]
                  pipeline = node_list.index(connection)
                  connection_host = self.host_names[pipeline]
                  node_host = self.host_names[connect]
                  if connection_host != node_host:
                           self.nodes_to_change.append(node_list[connect])
```

5. If both the node types are on different hosts the relationship type will be updated to *ConnectToRemotePipeline.* However, if the adjacent data pipeline nodes are on the same host, the relationship type remains unchanged. The code snippet for this purpose is given below:

```
def make_changes(self, content):
   for nodes_change in self.nodes_to_change:
         node = content["topology_template"]["node_templates"][nodes_change]
         if "requirements" in node.keys():
                  for req in node["requirements"]:
                    if "ConnectToPipeline" in req:
                       req["ConnectToRemotePipeline"] = req["ConnectToPipeline"]
                       del req["ConnectToPipeline"]
   return content
```

Ensuring correct data encryption configuration:

6. In case of encryption of data while moving through different systems, it is necessary to make sure that the password in both ends is the same. Currently, the pugin is checking if both nodes have the same password.
7. For this purpose, DPP first extracts all the TOSCA nodes that are of type *Encrypt* node followed by the adjacent *Decrypt* nodes. DPP then reads the password values of each pair of Encrypt and Decrypt nodes.
8. If the passwords are not the same the plugin itself will create a random password and assign it to the respective pair of *Encrypt* and *Decrypt* nodes. For this purpose, the code snippet is given below.

```
def update_password(self, content, nodes_to_modify):
    for nodes in nodes_to_modify:
        node_list = nodes.split('*')
        updated_password = self.generate_password(10)
        for n in range(0, len(node_list)-1, 1):
            if
content["topology_template"]["node_templates"][node_list[n]]["properties"]["password"]
!=
content["topology_template"]["node_templates"][node_list[n+1]]["properties"]["password
"]:

content["topology_template"]["node_templates"][node_list[n]]["properties"]["password"]
= updated_password
content["topology_template"]["node_templates"][node_list[n+1]]["properties"]["password
"] = updated_password
            return content
```

9. The TOSCA service template (YAML file) is then modified using `ruamel.yaml`[9] library to carry out all the YAML file related operations such as editing and dumping the YAML file, making sure that the alignment of the content is intact.
10. Upon updating the node topology and the relationship topology, the data pipeline plugin then prepares the CSAR by compressing all the files without compromising the directory structure.
11. The updated CSAR can then be stored in the template library and used for the deployment of all the data pipeline nodes using RADON orchestrator.

The RADON GMT exports the CSAR only with the required reference files, such as relationship types, node types, implementation files, and templates. However, in certain situation DPP needs to update the CSAR with new references such as to update the CSAR with a different relationship type DPP needs to pull the required relationship type definitions and implementation files from the RADON TOSCA node repository (radon-particle GitHub repository). Further, after integrating the DPP with the RADON IDE (which will be done in the coming periods), DPP needs to store the updated CSAR to the template library, which can be used in further development stages.

[9] https://pypi.org/project/ruamel.yaml/

4.3. Installation and setup

DPP can be used in two ways: as a standalone tool or as an API. In case of standalone system and invoking through the terminal, user needs to follow the steps below:

4.3.1. Standalone use:

User needs to install the prerequisites and makes sure the input CSAR is ready:

```
$ mkdir ~/opera && cd ~/opera
$ python3 -m venv .venv && . .venv/bin/activate
(.venv) $ pip install opera
(.venv) $ pip install ruamel.yaml
```

Clone the source code from the GitHub repo and invoke the plugin from the terminal with a python command:

```
(.venv) $ git clone https://github.com/radon-h2020/radon-datapipeline-plugin
(.venv) $ cd radon-datapipeline-plugin
(.venv) $ cd ../src
(.venv) $ python3 DPP.py <path to the CSAR file>
```

The detailed and updated description for the installation and usage example can be found in the GitHub repo [5].

4.3.2. Plugin API:

We have also developed the web service version of the plugin, which would allow the user to install this plugin in a Docker container that can be invoked through an REST API. This is available in the `datapipeline-server` folder in our GitHub repo [5].

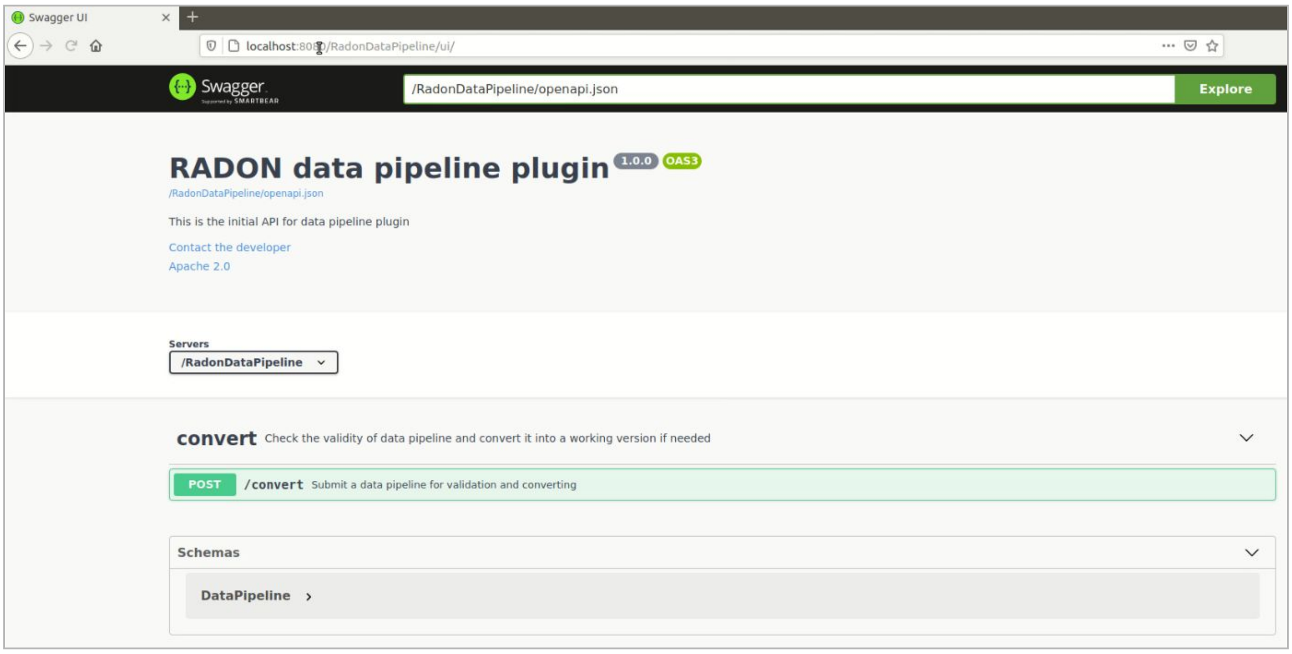
For this user needs to build the Docker image and create the container with the following code snippet:

```
# building the image
docker build -t openapi_server .

# starting up the container
docker run -p 8080:8080 openapi_server
```

Once the Docker container is running, users can use a Swagger UI for invoking the REST API operations. The detailed description can be found in the GitHub repo [5].

`http://localhost:8080/RadonDataPipeline/ui/`


Swagger UI
localhost:8080/RadonDataPipeline/ui/
Swagger
/RadonDataPipeline/openapi.json
Explore

RADON data pipeline plugin 1.0.0 OAS3

/RadonDataPipeline/openapi.json

This is the initial API for data pipeline plugin

Contact the developer

Apache 2.0

Servers

/RadonDataPipeline

convert Check the validity of data pipeline and convert it into a working version if needed

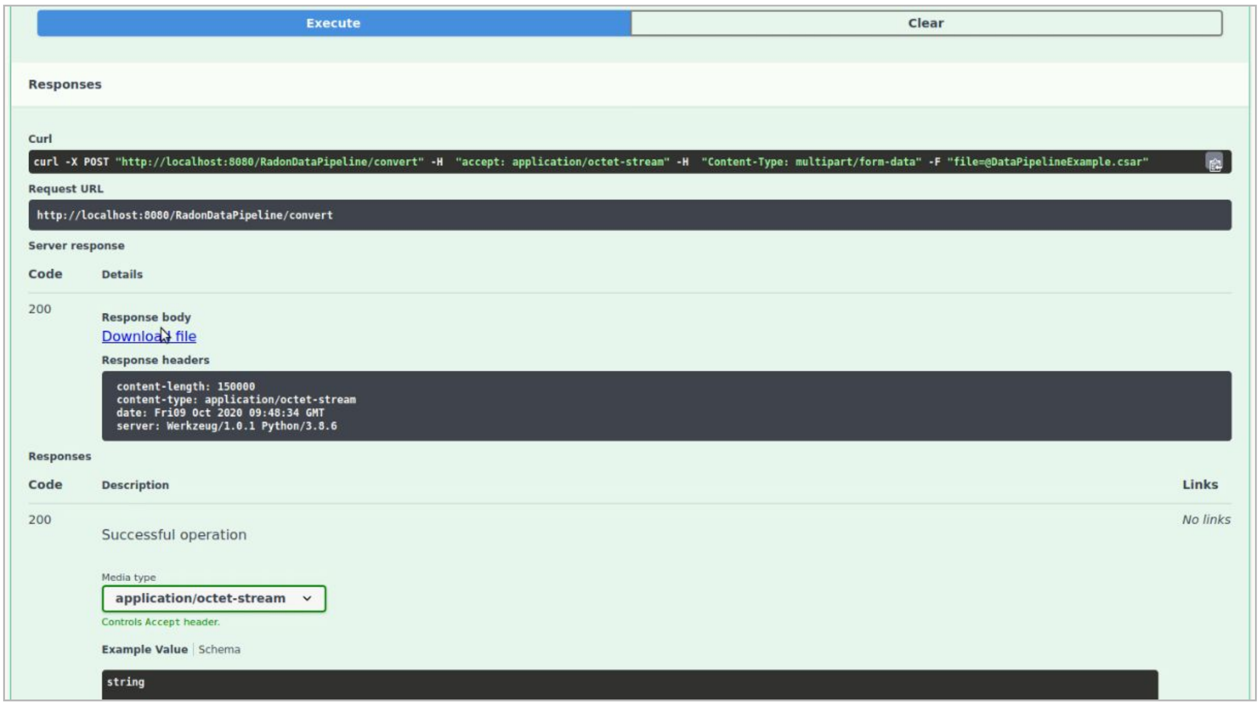
POST /convert Submit a data pipeline for validation and converting

Schemas

DataPipeline

Figure 4.2: A screenshot of RADON data pipeline plugin API

Upon accessing the data pipeline plugin UI, the user needs to click on the *POST* button and then click on the *Browse* button to upload the CSAR file through the *File Upload* dialog box. To verify and update the data pipeline CSAR, the user needs to click on the *Execute* button. On success plugin API will return a 200 response code and the user can then download the file by clicking on the *Download File* link, as given in Figure 4.3. Using the web UI is not required, the same functionality can be accessed through REST POST request (e.g. by the RADON Delivery Toolchain).



Responses

Curl

```
curl -X POST "http://localhost:8080/RadonDataPipeline/convert" -H  "accept: application/octet-stream" -H  "Content-Type: multipart/form-data" -F "file=@DataPipelineExample.csar"
```

Request URL

```
http://localhost:8080/RadonDataPipeline/convert
```

Server response

Code	Details
200	Response body Download file Response headers

```
content-length: 150000
content-type: application/octet-stream
date: Fri09 Oct 2020 09:48:34 GMT
server: Werkzeug/1.0.1 Python/3.8.6
```

Responses

Code	Description	Links
200	Successful operation	*No links*

Media type: application/octet-stream

Controls Accept header.

Example Value | Schema

```
string
```

Figure 4.3: A screenshot of response from RADON data pipeline plugin API after updating pipeline CSAR

With the ability to invoke the DPP operations through the REST API, the plugin can be integrated with the rest of the RADON tools. Delivery Toolchain is capable of running invoking REST operations as part of its CI/CD pipeline. To apply the DPP to convert a CSAR before it is deployed by the orchestrator, CI/CD pipeline is modified to perform a REST operation against the DPP endpoint before the orchestrator is invoked.

5. Requirements

In this section, we discuss the final requirements of the RADON data pipeline along with the progress of each requirement.

5.1. The data pipeline module of the orchestrator must be able to orchestrate data pipelines.

Requirement ID	R-T5.4-1
Title	The data pipeline module of the Orchestrator must be able to orchestrate data pipelines
Priority	Must have
User story	As an Operations Engineer, I want to be able to deploy data pipelines and control their life-cycle using RADON toolchain.
Type	FUNCTIONAL_SUITABILITY
Means of Verification	Evaluate whether the tool is able to successfully deploy data pipelines which have been defined using RADON models.

This requirement mainly focuses on ability to deploy the data-pipeline that is expressed using TOSCA language. For each DP node type created only using TOSCA language, separate implementation files are developed to control the life cycle of the nodes and manage the relationship with other nodes. In addition to this, we have developed the data pipeline plugin that is able to ensure the deployability of the pipeline-based cloud application by analyzing and fixing errors made by the user. The current version is able to orchestrate all the developed TOSCA-based data pipeline nodes mostly fulfilling the requirement R-T5.4-1. However, some typical features such as data routing, data filtering, etc will be equipped and then will be tested. Along with that, we will also continue until the end of WP5 to develop more data pipeline-related TOSCA node types and will make sure of the deployability of those.

5.2. The data pipeline module of the Orchestrator must support cron based scheduled data pipelines.

Requirement ID	R-T5.4-2
Title	The data pipeline module of the Orchestrator must support cron based scheduled data pipelines
Priority	Must have
User story	As an Operations Engineer, I want to be able to schedule data pipelines periodically.
Type	FUNCTIONAL_SUITABILITY
Means of Verification	Test data pipeline execution with different time based periodic schedules and verify that the pipeline is executed every single time.

One of the most important features that allows the user to efficiently handle the flow of data is the ability to provide the scheduling information using CRON. With this feature, users would be able define the scheduling time using CRON syntax and indicate the time at which the pipeline will be executed. This is an optional feature that comes with all data pipeline nodes with default value immediately after the pipeline is started and without going through any delay. All the DP nodes that are developed satisfy this feature. Furthermore, all the upcoming DP node types will also have this feature.

5.3. The data pipeline module of the Orchestrator should support event based scheduled data pipelines.

Requirement ID	R-T5.4-3
Title	The data pipeline module of the Orchestrator should support event based scheduled data pipelines
Priority	Should have
User story	As an Operations Engineer, I want to be able to trigger the data pipeline execution in an Event-driven manner.
Type	FUNCTIONAL_SUITABILITY
Means of Verification	Test data pipeline execution with different event based triggers and verify that pipeline is executed every single time.

Unlike the requirement R-T5.4-2, which talks about the CRON based scheduling, all the pipelines should also support the event based data movement and processing. The event can be creation of a new data object or deletion of a data object from a cloud storage/database. In the current version, all the DP nodes provide this feature and the nodes are enabled with this feature, which indicates that each DP node will listen to the events.

5.4. It would be useful for the data pipeline module of the Orchestrator to support logging and generating alerts on pipeline task failures.

Requirement ID	R-T5.4-4
Title	It would be useful for the data pipeline module of the Orchestrator to support logging and generating alerts on pipeline task failures.
Priority	Should have
User story	As a QoS Engineer, I want to be notified if the data pipeline execution fails.
Type	RELIABILITY
Means of Verification	Test data pipelines that have notifications enabled, force critical situations to happen (e.g. by sending malformed data, shutting down specific services or generating too much data) and verify that alarms are raised.

This requirement mainly focuses on the support of logging and generating alerts on failure of the pipeline tasks. This feature would increase the reliability of the system as the developer/user can take actions to mitigate the pipeline failure. The alert can be generated through email or in the dashboard.

5.5. The data pipeline module of the Orchestrator should support deployment of data pipelines which automate movement of data between two or more clouds.

Requirement ID	R-T5.4-5
Title	The data pipeline module of the Orchestrator should support deployment of data pipelines which automate movement of data between two or more clouds.
Priority	Should have
User story	As an Operations Engineer, I want to be able to deploy data pipelines across multiple (cloud) systems.
Type	FUNCTIONAL_SUITABILITY
Means of Verification	Test data pipeline orchestration where different parts of the pipeline are deployed on different clouds.

To meet the requirement of real life use cases, it is highly essential to provide a platform to deploy the data pipeline across multiple cloud systems along with the connection among those pipelines, so that the data can move across multiple clouds. The current version of data pipeline allows the user to deploy data pipeline onto private cloud, specifically OpenStack environment, and also onto the Amazon cloud. We have also developed TOSCA-based data pipeline nodes to move the data between Google Cloud storage and AWS S3 bucket.

5.6. The data pipeline module must support data pipelines tasks that initiate data analytics tasks for processing data moving through the pipeline.

Requirement ID	R-T5.4-6
Title	The data pipeline module must support data pipelines tasks that initiate data analytics tasks for processing data moving through the pipeline
Priority	Must have
User story	As an Operations Engineer, I want to be able to deploy data pipelines that trigger NLP data analytics jobs.
Type	FUNCTIONAL_SUITABILITY
Means of Verification	Test data pipelines that include initiating data analytics jobs as one or more pipeline tasks and verify that resulting data is correct.

The intermediate data can be processed either by invoking the remote serverless function or by injecting the data to specific pipelines that come with specific data analytics tasks, such as data filtering, data merging, data preprocessing, etc. With the current version, users are able to invoke and execute local analytics tasks using Python and Ruby engines. Users can also invoke remote Lambda and OpenFaaS based serverless functions that can be designed to analyze and process data. We will continue the development of pipelines with very specific data analytics tasks.

5.7. The data pipeline module of the Orchestrator should support configuring encryption between data pipeline tasks when data needs to be moved between systems.

Requirement ID	R-T5.4-7
Title	The data pipeline module of the Orchestrator should support configuring encryption between data pipeline tasks when data needs to be moved between systems.
Priority	Should have
User story	As an Operations Engineer, I want to be able to configure encryption for data moving through the data pipeline.
Type	SECURITY
Means of Verification	Test data pipelines that move data between two systems. Analyze network traffic and verify data is not sent unencrypted.

When the data move across multiple cloud systems, as discussed in requirement R-T5.4-5, a feature needs to be developed that would increase the security of the data. For this, the data pipeline should allow the user to encrypt the data before sending to another cloud system. With the current development, a user can add the encrypt and decrypt TOSCA node types in the source and destination side, respectively.

5.8. The data pipeline module of the Orchestrator should support deploying data pipelines expressed using TOSCA models into the AWS data pipeline service.

Requirement ID	R-T5.4-8
Title	The data pipeline module of the Orchestrator should support deploying data pipelines expressed using TOSCA models into the AWS data pipeline service.
Priority	Must have
User story	As an Operations Engineer, I want to deploy data pipelines to AWS data pipeline service.
Type	PORTABILITY
Means of Verification	Tests to verify that deployment to AWS data pipeline service is successful.

To fulfill the requirement R-T5.4-8, necessary TOSCA node types are created and made available through RADON-particle GitHub repository. Such nodes can be used to create, deploy or activate AWS data pipelines. Users can describe the operation using TOSCA language and the developed internal implementation file will control the entire lifecycle of the data pipeline. In the current version, users can deploy and activate CopyActivity and ShellCommandActivity data pipelines using dedicated TOSCA node types.

5.9. The data pipeline module of the Orchestrator should support deploying data pipelines expressed using TOSCA models into a private OpenStack cloud.

Requirement ID	R-T5.4-9
Title	The data pipeline module of the Orchestrator should support deploying data pipelines expressed using TOSCA models into a private OpenStack cloud.
Priority	Must have
User story	As an Operations Engineer, I want to deploy data pipelines to Private clouds
Type	PORTABILITY
Means of Verification	Tests to verify that deployment to AWS data pipeline service is successful.

Similar to requirement R-T5.4-8, users should also be able to deploy the data pipelines that are described using TOSCA language into the OpenStack environment. For this, Apache NiFi is used as the underlined technology. This means that before deploying the data pipelines into the OpenStack environment, Apache NiFi platform is installed and configured, atop which the different pipelines are deployed. The current version of the data pipeline fulfills this requirement and the developed TOSCA node types can be found in RADON particles GitHub repository.

5.10. Support Google BigQuery data warehouse as a data source when deploying data pipelines.

Requirement ID	R-T5.4-10
Title	Support Google BigQuery data warehouse as a data source when deploying data pipelines.
Priority	Must have
User story	As a user, I want to use Google BigQuery as a cloud-based enterprise data warehouse.
Type	PORTABILITY
Means of Verification	Test the data pipeline to verify that the data can be consumed and published from/to Google storage.

This requirement came from the external users of the RADON framework. They communicated to us that they are interested in using Google Cloud BigQuery data warehouse in the future and it would be good for the RADON data pipelines to support moving data from and into Google Cloud. In the current version, we have created TOSCA node types for consuming data from Google Cloud storage and publishing data into Google Cloud storage service and designed them in a way that they can be used together with all previous data pipeline node types. As Google BigQuery can import data from Google Cloud Storage, it partially fulfills the requirement. An additional node type will be developed which also initiates the import process and configures everything necessary on the Google BigQuery side.

6. Future work

This final deliverable gives an overall picture RADON data pipeline including the set of TOSCA models with different functionalities and the plugin to ensure the deployability of the service template. This section gives the development direction to follow even after this deliverable until the month 27 as per the description of action.

We see RADON data pipeline as a continuous development of the different TOSCA node types taking the use cases into consideration and improvement of the data pipeline plugin. From the development of TOSCA node type point of view, we will be developing a number of standalone TOSCA models that can be used to perform specific tasks. For example, synchronising Amazon S3 bucket and Google Cloud storage, copying the data from Amazon S3 bucket to dynamoDB, synchronising Amazon S3 bucket with the local directory structure, etc. The main goal of such standalone node types would be to fasten the design and modelling of cloud applications. We will continue developing pipelines that will initiate very specific and basic data analytics tasks. We will also continue developing more number of data pipeline related TOSCA node types to consume and publish the data from different storage units, such as from FTP and other remote databases.

On the other hand, the current version of the data pipeline plugin can handle the inconsistency related to the relationship type and the data encryption present in the service template (CSAR file). However, with the development of more number of data pipeline related node types, it is possible that there exist new potential bugs/inconsistencies, which may require the plugin to be updated. In such a scenario, the data pipeline plugin needs to be tested with new node types. The current version of the data pipeline ensures that the passwords of Encrypt and Decrypt data pipeline nodes are the same. In case of different passwords, the plugin will generate a new random password and assign it to the corresponding Encrypt and Decrypt data pipeline nodes. We will improve the plugin so that the users don’t need to worry about adding the Encrypt and Decrypt nodes. In this case, the plugin will automatically add the necessary nodes for encryption/decryption if the data is moving from one system to another.

7. Conclusions

This deliverable extends the initial deliverable D5.5 and presents the final version of the RADON data pipeline methodology and its orchestration. The data pipeline architecture is presented and discussed with a set of related TOSCA models. The developed TOSCA models are available in the RADON particles GitHub repository[10] and can be used to freely compose a data intensive cloud application combining independently deployable, schedulable, and scalable pipeline tasks, such as microservices, serverless functions, or self-contained applications. The supported functionalities are also discussed in-detailed in this deliverable report. In addition to this, we have presented the working principle of the data pipeline plugin that handles the potential errors made by the developer.

Table 7.1 shows the achieved level of compliance to RADON requirements. The values in “Level of compliance” are defined as follows. Some of the “Must have” requirements have less “Level of Compliance” values as compared to that of the requirements with “Should have” priority. The level of compliance for requirement R-T5.4-1 is partially-high. This is due to the fact that some typical features such as data routing, data filtering will be equipped in the coming months and then will be tested. Similarly, with the current development of requirement R-T5.4-6, users can invoke the remote serverless function and local data analytics task. However, it is under development process to develop the data pipeline TOSCA node with very specific data analytics tasks, such as data aggregation, data fusion, and data extraction. For these reasons, requirement R-T5.4-6 has a partially-high level of compliance. Requirements R-T5.4-8 and R-T5.4-10 have a partially-high level of compliance. The current versions will further be developed with more advanced features. For instance, in requirement R-T5.4-8, we will add more TOSCA node types, as discussed in Section 2.4. In case of requirement R-T5.4-10, as Google BigQuery can import data from Google Cloud Storage, it partially fulfills the requirement. An additional node type will be developed, which also initiates the import process and configures everything necessary on the Google BigQuery side.

(i) ✗ (unsupported): the requirement is not fulfilled by the current version

(ii) ✔ (partially-low supported): a few of the aspects of the requirement is fulfilled by the current version

(iii) ✔✔ (partially-high supported): most of the aspects of the requirement is fulfilled by the current version

(iv) ✔✔✔ (fully supported): the requirement is fulfilled by the current version.

Table 7.1: Achieved level of compliance to RADON requirements

[10] https://github.com/radon-h2020/radon-particles

Id	Requirement Title	Priority	Level of compliance
R-T5.4-1	The data pipeline module of the Orchestrator must be able to orchestrate data pipelines	Must have	✔✔
R-T5.4-2	The data pipeline module of the Orchestrator must support cron based scheduled data pipelines	Must have	✔✔✔
R-T5.4-3	The data pipeline module of the Orchestrator should support event based scheduled data pipelines	Should have	✔✔✔
R-T5.4-4	It would be useful for the data pipeline module of the Orchestrator to support logging and generating alerts on pipeline task failures.	Should have	✔
R-T5.4-5	The data pipeline module of the Orchestrator should support deployment of data pipelines which automate movement of data between two or more clouds	Should have	✔✔✔
R-T5.4-6	The data pipeline module must support data pipelines tasks that initiate data analytics tasks for processing data moving through the pipeline	Must have	✔✔
R-T5.4-7	The data pipeline module of the Orchestrator should support configuring encryption between data pipeline tasks when data needs to be moved between systems	Should have	✔✔✔
R-T5.4-8	The data pipeline module of the Orchestrator should support deploying data pipelines expressed using TOSCA models into the AWS data pipeline service.	Must have	✔✔
R-T5.4-9	The data pipeline module of the Orchestrator should support deploying data pipelines expressed using TOSCA models into a private OpenStack cloud.	Must have	✔✔✔
R-T5.4-10	Support Google BigQuery data warehouse as a data source when deploying data pipelines.	Must have	✔✔

References

[1] RADON Consortium, Deliverable D5.5 - Data pipeline orchestration I, 2019, http://radon-h2020.eu/wp-content/uploads/2020/01/D5.5-Data-Pipeline-Orchestration-I.pdf

[2] C. K. Dehury, S. N. Srirama, T. R. Chhetri, "CCoDaMiC: A framework for Coherent Coordination of Data Migration and Computation platforms", Future Generation Computer Systems, Vol 109, pp 1-16, 2020, ISSN 0167-739X

[3] C. Dehury, P. Jakovits, S. N. Srirama, V. Tountopoulos, G. Giotis, "Data Pipeline Architecture for Serverless Platform", European Conference on Software Architecture, 2020, isbn 978-3-030-59155-7,

[4] RADON Consortium, Deliverable D4.1 - Constraint Definition Language I, 2019, http://radon-h2020.eu/wp-content/uploads/2020/01/D4.1-Constraint-definition-language-I.pdf

[5] radon-datapipeline-plugin: A plugin to make the TOSCA-based data pipeline service blueprint workable. https://github.com/radon-h2020/radon-datapipeline-plugin

[6] Amazon Web Services (AWS) - Cloud Computing Services, https://aws.amazon.com/ [Online; accessed 10-Oct-2020] (2020).

[7] Apache NiFi documentation, https://nifi.apache.org/, [Online; accessed 10-Oct-2020] (2020)

[8] ruamel.yaml·PyPI, https://pypi.org/project/ruamel.yaml/, [Online; accessed 21-Oct-2020] (2020)

[9] RADON Consortium, Deliverable D4.5 - Graphical Modeling Tool I, 2019, http://radon-h2020.eu/wp-content/uploads/2020/01/D4.5-Graphical-Modelling-Tools.pdf

[10] Cron - Linux manual page, https://man7.org/linux/man-pages/man8/cron.8.html, [Online; accessed 27-Oct-2020] (2020)