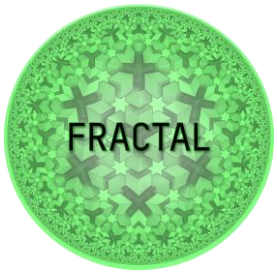


Deliverable

D6.4 FRACTAL engineering framework validation

Deliverable Id:	D6.4
Deliverable Name:	FRACTAL engineering framework validation
Status:	Final
Dissemination Level:	Public
Due date of deliverable:	2023 (M31)
Actual submission date:	2023 (M31)
Work Package:	WP6 CPS Communication Framework
Organization name of lead contractor for this deliverable:	IKERLAN
Author(s):	Ana Patricia Bautista, IKER Adrián Morán, IKER Luca Visconti, AKKODIS Pietro Abbatangelo, AKKODIS Enrico Ferrari, RULEX Nicola Alchera, RULEX
Reviewers:	Roman Obermaisser, SIEG Stefan Krassin, PLC2

Abstract:
D6.4 "FRACTAL engineering framework validation" is a report of the results of the validation tests of the functionalities and components of the FRACTAL Edge Node, designed and implemented in tasks T6.1 and T6.2. The tests were carried out following the validation methodology defined in deliverable D6.3, where the steps to be followed to carry out the validation process were set out.

ECSEL Joint Undertaking
Electronic Components and Systems for European Leadership

This project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 877056

Co-funded by the Horizon 2020 Programme of the European Union under grant agreement No 877056.

Contents

1 History

Version	Date	Modification reason	Modified by
0.0	30/05/2022	Draft	Ana Patricia Bautista
0.1	15/11/2022	Redefinition of sections	Ana Patricia Bautista
0.2	15/03/2023	Deliverable ready for review	Authors
1.0	29/03/2023	Final version	Authors

Table 1 Document history

2 Summary

This deliverable reports the results of the validation tests implemented in the framework of task T6.3. The inputs of this task have been the developments of tasks T6.1 and T6.2, the Edge Node design, and implementation and the Edge Controller design and implementation, respectively.

For the development of this task, deliverables D6.1 and D6.2, other code, video demos, and repositories that were delivered as results of tasks T6.1 and T6.2 were studied and considered. The methodology outlined in D6.3 has been followed to define and develop the validation tests that will ensure that the components function correctly. However, the tests for tasks T6.1 and T6.2 have been approached in different ways and are presented in different chapters. The reason is that the results have been presented differently, and some of the developments that were in the scope of task T6.1 were moved to task T6.2 as is the case for the Low End Node. Therefore, the components of Task T6.2 complement the developments in Task T6.1, so the results presented in this document are aligned with that situation.

It is important to mention that during the definition of the validation tests, developers have been consulted to gain a better understanding of how the components work. In addition, they have also received feedback and have been able to implement some improvements to the components while the validation was being carried out.

2.1 Achievements

Highlights

1) "Coverage" – the microservices related to connectivity presented in D6.1 have been considered in this deliverable.

2) "Coverage" - all components presented in D6.2 have been taken into account in this deliverable.

3) "Coverage" - D6.2 presents all functionalities related to each component, but some of them are out of project scope or for future scope; in this deliverable there is a clarification on what is working at this project step and what is out of scope.

4) "External view" - in many cases partners who led the validation activities and test cases definition were not involved in the development.

5) "Know-how sharing" - as per point 4, partner had the possibility to work together, tester and developer.

6) "Quality improvement" - The validation work has helped to identify some bugs in the components, and these have been fixed in the course of the validation.

In addition, feedback has been sent to the developers with some suggestions for improvement.

Lowlights

1) "Coverage": some functionality presented in D6.1 and D6.2 are of future scope.

2) In some cases, functionalities are presented and tested in simple scenarios. More complex scenarios will be defined during Use Cases implementation.

3 Introduction

The main content of this deliverable is the report of the validation results of the components developed in WP6 (tasks T6.1 and T6.2). The validation tests defined in this deliverable are based on deliverables D6.1, D6.2 and other materials such as video demos, code scripts and repositories that have been delivered by the developers of tasks T6.1 "FRACTAL processing node design and implementation" and T6.2 "FRACTAL Edge Controller design and implementation".

The phases of the validation process defined in section 5.4 of deliverable D6.3 have been followed to carry out the validation tests:

1. Test planification: where the scope of the test is defined, and the identification data is assigned.
2. Test case development: where the steps to carry out the test are defined, and the necessary scripts or configurations are created.
3. Test environment setup: This section refers to or explains how to prepare the test environment for the test (this information is given by the developers).
4. Test execution: the test is carried out and the results are reported according to these guidelines.

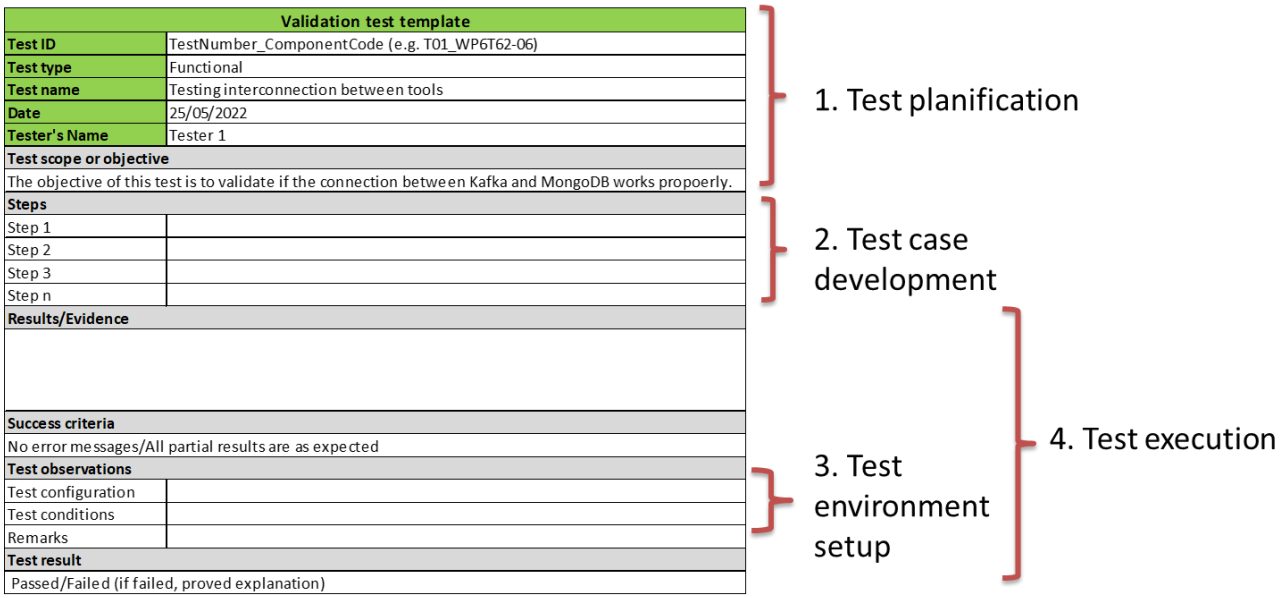
The reporting process was documented through subchapters with the name of the phases of the validation process, and in the case of the tests performed for task T6.2 components, they were documented in a validation template which encloses these steps. An overview of the validation template and usage is given in chapter 4.

First, chapter 5 provides an overview of the architecture of the Edge Node and the input received by task T6.1, and in chapter 6 the implementation of the validation tests of the microservices to test connectivity functionalities are documented.

Then, a brief description of the FRACTAL Edge Controller is presented in chapter 7, and in chapter 8, the implementation of the validation tests of the components developed in task T6.2 are documented.

4 Validation test template overview

As it is mentioned in the introduction, in this chapter, the validation test template is briefly explained. It was designed as a supporting document to report the phases of the validation process based on the validation methodology approach defined in deliverable D6.3, section 5.4.



Validation test template	
Test ID	TestNumber_ComponentCode (e.g. T01_WP6T62-06)
Test type	Functional
Test name	Testing interconnection between tools
Date	25/05/2022
Tester's Name	Tester 1
Test scope or objective	
The objective of this test is to validate if the connection between Kafka and MongoDB works propoerly.	
Steps	
Step 1	
Step 2	
Step 3	
Step n	
Results/Evidence	
Success criteria	
No error messages/All partial results are as expected	
Test observations	
Test configuration	
Test conditions	
Remarks	
Test result	
Passed/Failed (if failed, proved explanation)	

Figure 1: Validation test template

The template is self-explanatory. However, here's how to fill in some important fields:

TestID: consist of the test number + component code e.g., the first test for component WP6T62-06 would be:

T01_ WP6T62-06

Test type: it depends on the kind of test to be implemented, it could be functional or nonfunctional.

- Functional requirements describe the function of the system and its components, defining the behaviour between inputs and outputs of the system.

- Nonfunctional requirements establish the standardized criteria to assess the quality of the product developed. This kind of requirements should be tested on the actual HW platform. Since in a simulated environment the data might

not be accurate, and the results will be meaningless for the implementation in real use cases.

Test scope or objective: it defines what is to be validated and why.

Steps: the steps for carrying out the test are defined.

Results/Evidence: this space shows evidence that the test was carried out (screenshot of the execution, message from the application, etc.).

Success criteria: describe the criteria to consider the test successful.

Test observations:

- **Test configuration:** provide the set-up configuration and requirements to carry out the test. It could be a link to the FRACTAL repository or the reference to a document.
- **Test conditions:** briefly describe the conditions of the test, e.g., the test was done remotely, three servers were used to perform the test, etc.
- **Remarks:** report the issues detected during the testing process.

Test result: passed or failed/Not passed.

5 Edge Node architecture review

This chapter briefly describes the Edge Node architecture design implemented in task T6.1, which developed and deployed the necessary Edge Computing infrastructure. For more information and detail on the Edge Node software design and implementation, refer to deliverable D6.1.

The Edge Node architecture was designed using mostly open-source software. It includes core functionalities, core microservices and appropriate mechanisms to support remote monitoring, resource management and dynamic reconfiguration. It also provides connection interfaces to different IoT devices and cloud platforms.

According to D6.1, the reference architecture for the development of the Edge Node was Kubernetes with Docker, which suits the interoperability approach and integrations with other systems. In addition, it is open to extensions, which provides more openness and fits into fractality design principles. The implementation was based on Kubernetes family Microk8S and k3S as it brings lightweight, fully-featured, conformant Kubernetes for IoT devices.

5.1 FRACTAL Edge Node processing architecture

The FRACTAL architecture is hierarchical, which means that upon layers are built upon, and consume data and services provided by lower layers. Each layer performs an important function throughout the architecture. The application layer, for example, contains modules to implement core functions for visualization, control, analytics, data fusion, filtering, and storage database. On the other hand, the communication and connectivity layer provide the intermediate elements required in terms of hardware and software to exchange data between the data center and the network devices. For a detailed explanation of the different layers, refer to D6.1.

The following image shows the FRACTAL Edge Node processing architecture designed in task T6.1:

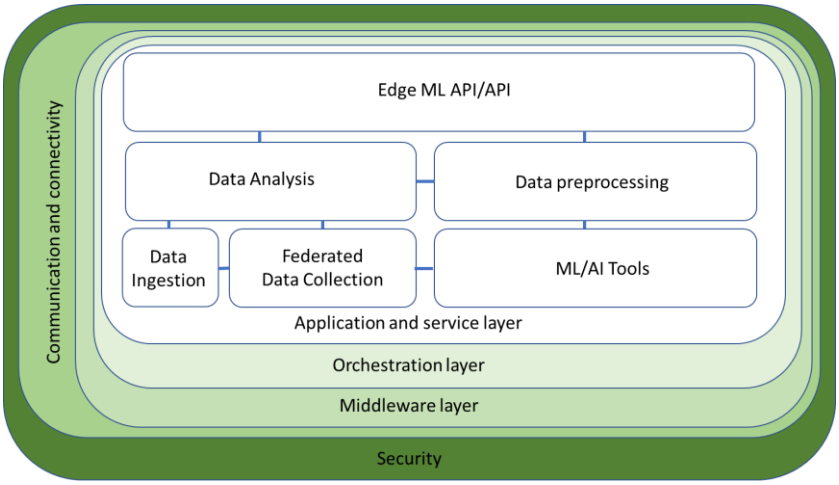


Figure 2 FRACTAL Edge Node processing architecture designed in task T6.1

5.2 FRACTAL Edge Node processing architecture implementation

The FRACTAL Edge Node processing architecture is the main input for this deliverable. The validation test plan will be designed based on it to validate the correct functionality of the Edge Node.

As it was mentioned before, the FRACTAL processing architecture was implemented using open-source applications and following an architecture based on microservices, which allows the Edge Node to have flexibility and scalability, as well as interoperability between heterogeneous devices and applications. Figure 3 below shows the FRACTAL Edge Node processing architecture implementation developed in task T6.1. For further detail, refer to deliverable D6.1.

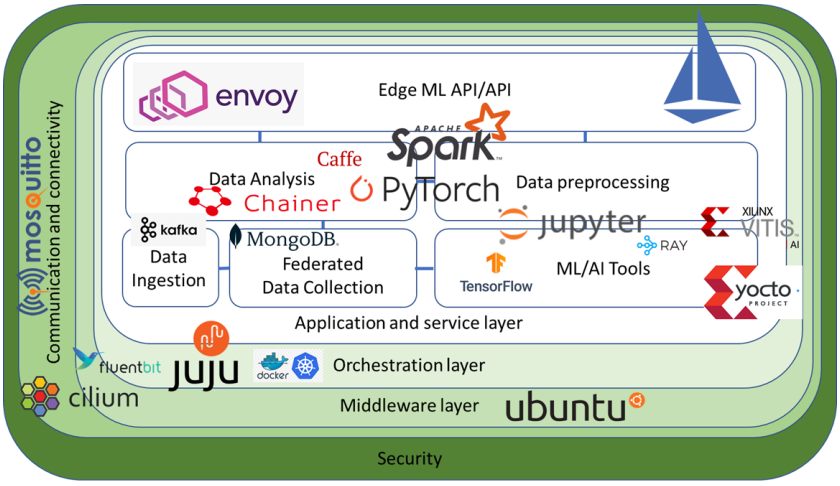


Figure 3 FRACTAL Edge Node processing architecture implementation developed in task T6.1

6 Validation tests implementation of the Edge Node microservices to test connectivity functionalities.

This chapter presents the results of the implementation of the tests for connectivity functionalities of the Edge Node. The first step, "Test planification" was done in the previous deliverable D6.3. It is important to mention that not all tests foreseen in the deliverable D6.3 were feasible to perform because the developments were not in the scope of task T6.1.

The code provided for testing can be founded in this link: Code

6.1 Test case development

In this test case, the interoperability between different data technologies has been evaluated. In order to understand the validations carried out, the whole test bed prepared must be explained. The testbed architecture is shown in Figure 4.

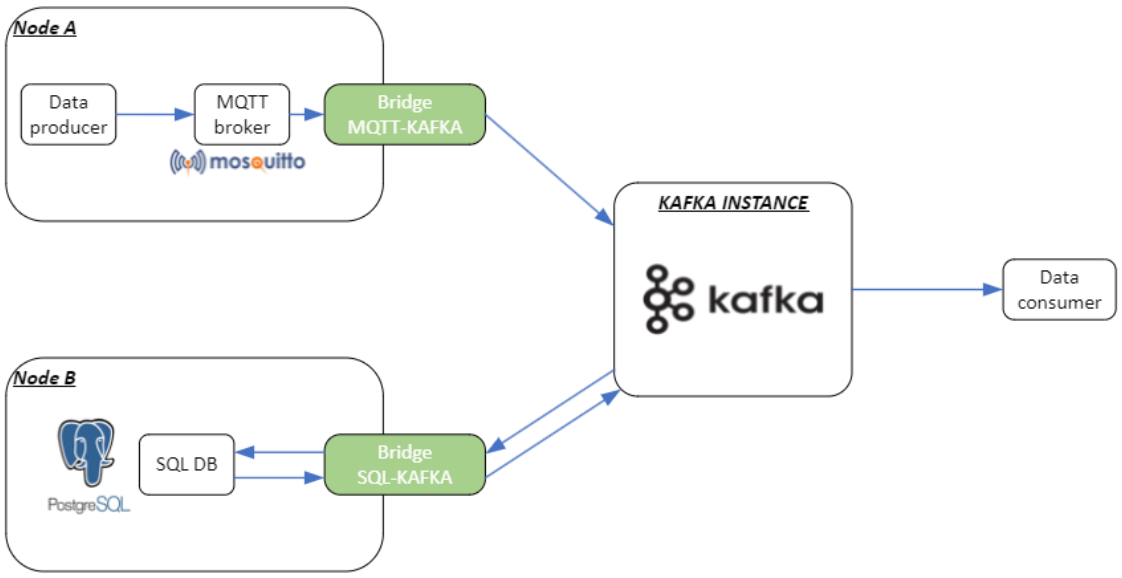


Figure 4 FRACTAL Edge Node testbed architecture

In the proposed testbed, there are three main entities that is worth to be described independently.

The so-called **Node A** has a data producer generating, in this case, the temperature of a city every 3 seconds. This data is produced randomly (with values between 1 and 30) and sent to the MQTT Broker in the same node. This node has a bridge to translate MQTT data to KAFKA. This will be one of the main entities under test.

On the other side, there is a so-called **Node B** that also has a bridge but, in this case, to translate information between KAFKA and a SQL database. This bridge is

bidirectional and will be the second main entity under test. This bridge has been configured to retrieve temperature data from KAFKA to store it in the SQL database and to periodically export the stored temperature value to another KAFKA topic.

Finally, there is the **KAFKA INSTANCE** that works as a central node to exchange data between nodes. Connected to this KAFKA broker, there is a data consumer in charge of reading information exchanged over the KAFKA broker to verify that the testbed is working correctly.

As aforementioned, this testbed was designed to validate that intercommunication between nodes is possible regardless of their differences in terms of technology.

The code of each entity is quite simple and is provided below for a better understanding:

```
import paho.mqtt.client as mqtt
from random import randint
import time

# MQTT Address
HOST = "localhost"
TOPIC = "temp_measured"

mqtt_client = mqtt.Client("Random_Generator")
mqtt_client.connect(HOST, 1883)

while True:
    randNumber = randint(1, 30)
    mqtt_client.publish(TOPIC, randNumber)
    print("Send a message to MQTT: " + str(randNumber) + " to topic " + TOPIC)
    time.sleep(3)
```

Figure 5 - MQTT data producer code

As can be seen in Figure 5, the data producer entity generates, every 3 seconds, a random number between 1 and 30 and publish that value on the MQTT broker in the topic *temp_measured*.

```python
from kafka import KafkaProducer
import paho.mqtt.client as mqtt
import time

# The Topic Name
MQTT_TOPIC = "temp_measured"
KAFKA_TOPIC = "temp_on_kafka"

# The address of Kafka server
KAFKA_HOST = "127.0.0.1:29092"

# Mqtt Address
MQTT_HOST = "localhost"

# MQTT Settings
mqtt_client = mqtt.Client("BridgeMQTT2Kafka")
mqtt_client.connect(MQTT_HOST, 1883)

# Kafka Settings
kafka_producer = KafkaProducer(bootstrap_servers=KAFKA_HOST)

def on_message(client, userdata, message):
    msg_payload = message.payload
    msg_payload = msg_payload.decode()
    print("Received MQTT message: ", msg_payload)
    kafka_producer.send(KAFKA_TOPIC, message.payload)
    print("Send the message: " + msg_payload + f" to Kafka with topic {KAFKA_TOPIC}!")

mqtt_client.loop_start()
mqtt_client.subscribe(MQTT_TOPIC)
mqtt_client.on_message = on_message
time.sleep(30000)
mqtt_client.loop_end()
```

Figure 6 - Bridge MQTT-KAFKA code

As can be seen in Figure 6, the MQTT-KAFKA bridge performs a connection to Node A's MQTT broker, in which it subscribes to the topic *temp_measured* (the one in which the data producer sends the generated data). On the other side, it also performs a connection with the KAFKA broker. Each time the bridge receives new data on the subscribed MQTT topic, it produces the same data of the KAFKA topic called *temp_on_data*. This way, the data produced by the data producer is available over MQTT within Node A (in MQTT's topic *temp_measured*) and also over KAFKA from other nodes (in KAFKA's *temp_on_kafka* topic).

```
import psycopg2
from kafka import KafkaConsumer
from kafka import KafkaProducer
import time
import threading

# KAFKA Address
KAFKA_HOST = "127.0.0.1:29092"

# PostgresSQL Settings
PS_DB_NAME  = "ikerlan"
PS_USERNAME = "ikerlan"
PS_PASSWORD = "ikerlan"
PS_HOST = "localhost"
PS_PORT = 5432

# The Topic Name
TOPIC_WT = "postgresql"
TOPIC_RD = "temp_on_kafka"

def send_to_kafka():
    # Kafka Settings
    kafka_producer = KafkaProducer(bootstrap_servers=KAFKA_HOST)

    # POSTGRESQL
    connection = psycopg2.connect(f"dbname={PS_DB_NAME} user={PS_USERNAME}
password={PS_PASSWORD} host={PS_HOST} port={PS_PORT}")
    cursor = connection.cursor()

    while True:
        cursor.execute("SELECT * FROM weather WHERE city = 'Oulu'")
        data = cursor.fetchall()
        for d in data:
            kafka_producer.send(TOPIC_WT, str(d[1]).encode())
            print("Sent the " + d[0] + " temperature " + str(d[1])  + f" to topic {TOPIC_WT}")
        time.sleep(1)

def fetch_from_kafka():
    # Kafka Settings
    kafka_consumer = KafkaConsumer(bootstrap_servers=KAFKA_HOST)
    kafka_consumer.subscribe(TOPIC_RD)

    # POSTGRESQL
    connection = psycopg2.connect(f"dbname={PS_DB_NAME} user={PS_USERNAME}
password={PS_PASSWORD} host={PS_HOST} port={PS_PORT}")
```

```
    cursor = connection.cursor()

    while True:
        for msg in kafka_consumer:
            data = msg.value.decode()
            cursor.execute("UPDATE weather SET temp = " + data + " WHERE city = 'Oulu'")
            connection.commit()
            print("Received city temperature: " + data)

if __name__ == "__main__":
    threading.Thread(target=send_to_kafka).start()
    fetch_from_kafka()
```

Figure 7 - Bridge SQL-KAFKA code

As aforementioned, SQL-KAFKA bridge was designed as a bidirectional bridge. In this sense, it can be seen in the code (Figure 7) hat there are two threads, one in charge of moving data from the KAFKA broker to the SQL database and a second thread in charge of extracting data from SQL database and publishing it on KAFKA topic. The thread in charge of getting data from KAFKA reads information from KAFKA's topic temp_on_kafka, and each time it receives data, it stores the received data in the database. On the other side, the thread in charge of sending data to KAFKA reads information from the database and sends it to KAFKA's topic postgresql periodically.

At this point of the testbed's explanation, it is possible to see that the data generated by the data generator is available at these points:

- At MQTT level in Node A's scope.
- At KAFKA level in KAFKA instance's scope.
- At SQL level in Node B's scope.

Moreover, the value stored in the database is also available for reading at KAFKA's scope.

```python
from kafka import KafkaConsumer

# Address of the Kafka
HOST = "127.0.0.1:29092"

# LIST OF THE TOPICS
topics_list = ["temp_on_kafka", "postgresql"]

consumer = KafkaConsumer(bootstrap_servers=HOST)
consumer.subscribe(topics_list)

for i in consumer:
    print("Message from topic: ", i.topic, "VALUE: ", i.value)
```

Figure 8 - Data consumer code

Finally, as can be seen in Figure 8, the data consumer subscribes itself on KAFKA broker to the topics *temp_on_kafka* and *postgresql*. It is a simple way to check that data exported from Node A is correct and that data imported to Node B is also successfully stored in the database and exported then to KAFKA again.

6.2 Test environment setup

For the simplicity of the test, all the testbed has been developed over a single node using Docker. The node used for this purpose is described in Table 6:

OS	Ubuntu 20.04 LTS
CPUs	2
RAM	4GB
Docker engine version	20.10.21
Python version	3.8

Table 2 – Testing node specifications

To run common entities of the testbed in an autonomous way, docker compose has been used, Figure 9 shows the configuration for the automatic launching of MQTT, KAFKA and SQL instances:

```yaml
version: "3"
services:
   mosquitto:
      image: eclipse-mosquitto:1.6.12
      ports:
         - 1883:1883
   zookeeper:
      image: confluentinc/cp-zookeeper:latest
      environment:
         ZOOKEEPER_CLIENT_PORT: 2181
         ZOOKEEPER_TICK_TIME: 2000
      ports:
         - 22181:2181
   kafka:
      image: confluentinc/cp-kafka:latest
      depends_on:
         - zookeeper
      environment:
         KAFKA_BROKER_ID: 1
         KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
         KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:9092,PLAINTEXT_HOST://localhost:29092
         KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
         KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
         KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      ports:
         - 29092:29092
         - 9092:9092
   postgres:
      image: postgres
      environment:
         POSTGRES_PASSWORD: ikerlan
         POSTGRES_USER: ikerlan
         POSTGRES_DB: ikerlan
      ports:
         - 5432:5432
```

Figure 9 - Automatic launching of MQTT, KAFKA and SQL instances

This docker compose configuration launches:

- MQTT broker based on Mosquitto.
- KAFKA broker (which also requires Zookeeper).
- SQL database based on PostgreSQL.

6.3 Test execution

The first step to execute the test, is running the docker compose configuration with basic services as it is shown in Figure 10:

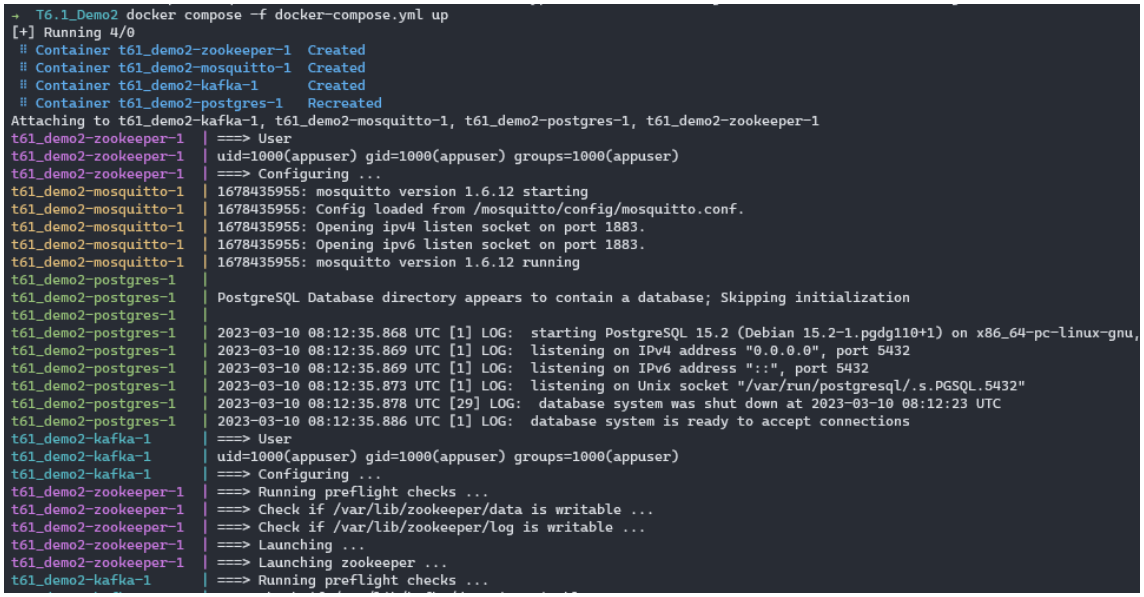


Figure 10 - Run common services

Now that we have MQTT, KAFKA and PostgreSQL running, the next step is to run the data producer (Figure 11):

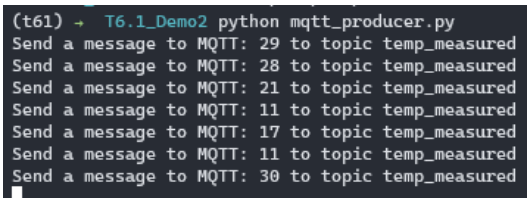


Figure 11 - Run data producer

Now that we have data in Node A's MQTT broker, it is time to run the bridge that exports this data to KAFKA's scope (Figure 12):

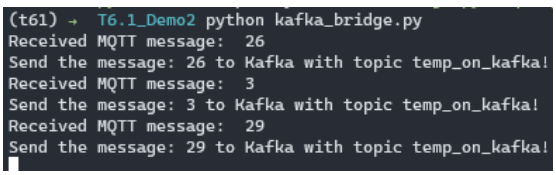


Figure 12 - Run MQTT-KAFKA bridge

With the data in KAFKA's scope, it is time to run the bridge (Figure 13) that imports such data into SQL scope and also exports databased stored information into KAFKA's scope:

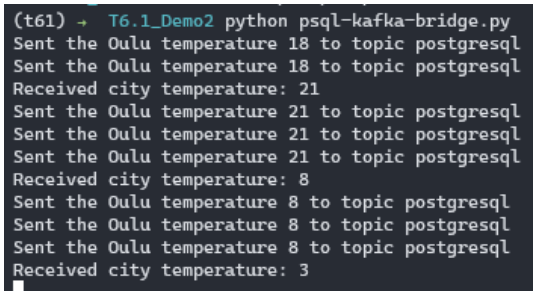


Figure 13 - Run SQL-KAFKA bridge

At this point, the only missing entity to run is the data consumer (Figure 14), which is, in turn, the entity that allows us to validate that the data pipeline is working correctly:

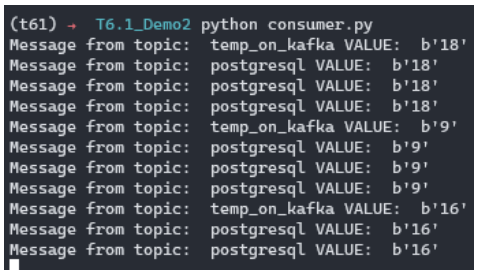


Figure 14 - Run data consumer

Now that we have all the entities involved in the test up and running, it is time to validate that the data workflow is the expected one. For this task, capture of all entities generating/moving data is presented in order to analyse it:

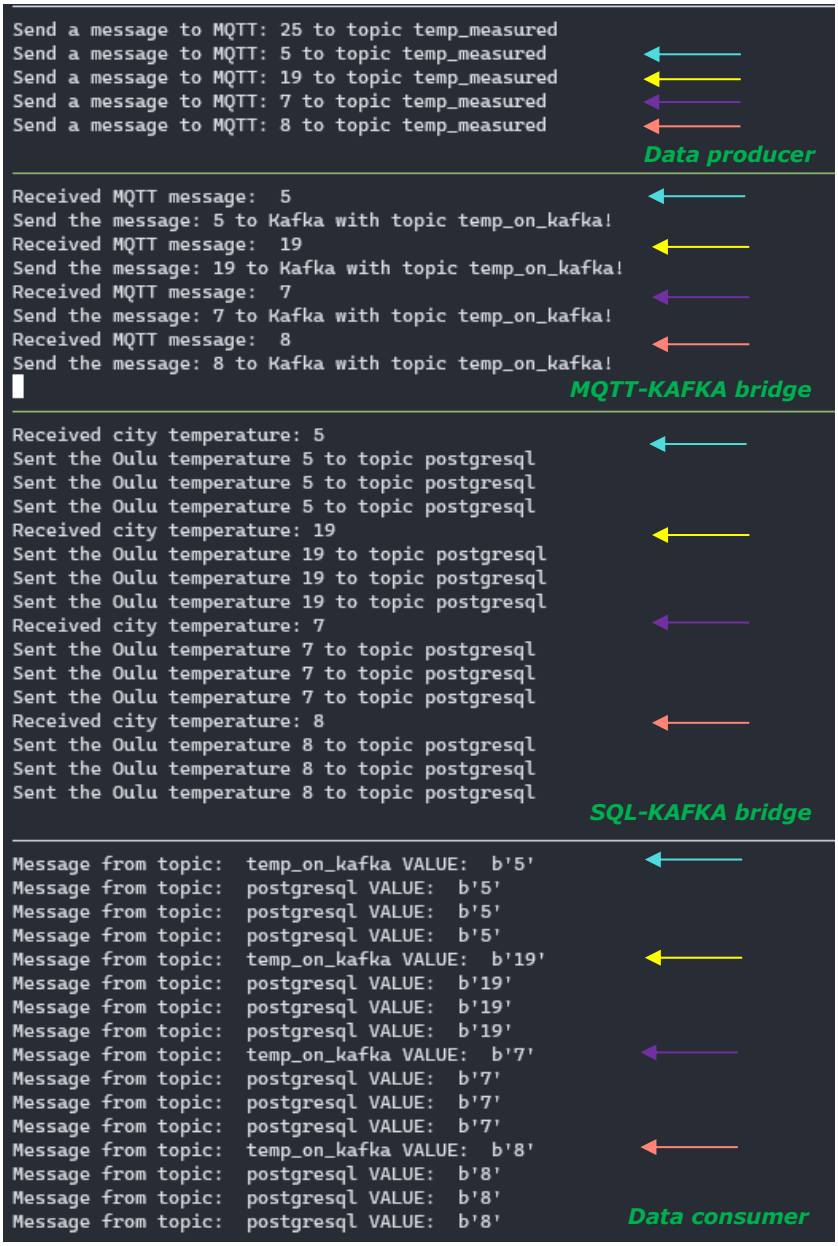


Figure 15 - Testbed data analysis

As can be seen in the results above (Figure 15), each data generated by the data producer is received by MQTT-KAFKA bridge and forwarded to KAFKA. This forwarded data is received by the SQL-KAFKA bridge and stored into the SQL database. Finally, the value stored in the database is retrieved every second and sent to the KAFKA broker. At the consumer level, it is possible to validate that the value retrieved from MQTT and the one retrieve from SQL are ok.

The scripts used to perform these validation tests are not exactly the same as the received ones (from task T6.1). Some adaptations were necessary, but the fundamental idea of the tests can be carried out to validate that data exchange

mechanism designed in the project is viable and usable. Therefore, the tests are considered passed.

7 FRACTAL Edge Controller review

This chapter briefly describes the FRACTAL Edge Controller architecture implemented in task T6.2, which focuses on the development of a communication and system monitoring component to optimize the overall system resources of a group of FRACTAL nodes. Therefore, it is an open-source software component to provide the Edge platform with self-orchestration and independence mechanisms at various levels. Figure 5 shows a diagram with the architecture of the Edge Controller designed in Task T6.2. For more detailed information on the Edge Controller design and implementation, refer to the deliverable D6.2.

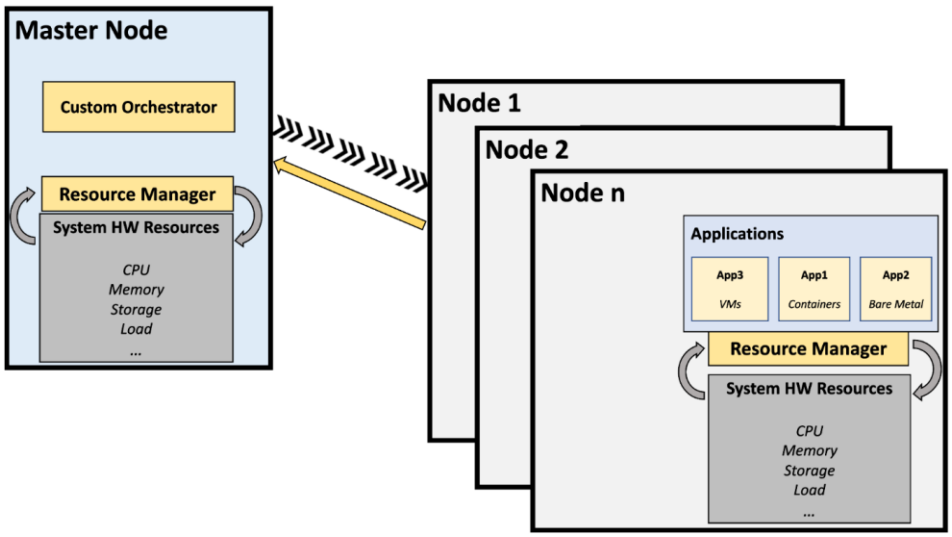


Figure 16: Multi-node Edge Controller architectural design (designed in task T6.2)

The main inputs to Task T6.2 are the following eigth components:

1. WP6T62-06 Orchestration (Edge Controller)
2. WP6T62-06 Orchestration (Agent Nodes Controller)
3. WP6T62-03 Run time Manager
4. WP6T62-01 Data Ingestion
5. WP6T62-0W MQTT Cloud comm. System --> Merged into Data Ingestion
6. WP6T62-02 Federated Data Collection
7. WP6T62-06 Low-end node orchestrator
8. WP6T62-0X Hardware Edge Controller

Through chapter 8, the validation report of each component will be presented. Here is a summary of each component:

WP6T62-06 Orchestration (Edge Controller)

The Edge Controller is an autonomous orchestrator for containers to support K8S, Docker, or orchestrator-less Fractal nodes.

WP6T62-06 Orchestration (Agent Nodes Controller)

The Agent Nodes Controller is part of the orchestration and is an add-on to the Edge Controller that provides Edge Nodes with the ability to orchestrate tasks and assign them to available nodes.

WP6T62-03 Run time Manager

The Runtime Manager is a component developed to coordinate and manage task scheduling and load balancing operations between modules in one or more fractal nodes at runtime. The purpose of the Runtime Manager is to enable communication and data dispatch among the various components installed on the node, and to manage the load balancing operations, when needed, by assigning the execution of the activities to a different instance of the Runtime Manager module installed on another node.

WP6T62-01 Data Ingestion

The Data Ingestion component provides data ingestion and data streaming processing tools for the High-End and Mid-End Fractal nodes.

WP6T62-0W MQTT Cloud comm. System

This component is part of the WP6T62-01 Data Ingestion component.

WP6T62-02 Federated Data Collection

The federated Data Collection component provides data storage capabilities for the Fractal High-End and Mid-End nodes.

WP6T62-06 Low-end node orchestrator

As a result of WP3, Nuttx RTOS was ported to the PULP low-end systems to offer a Posix completable application environment and reported in D3.6. In WP6 an IoT Hub was integrated into the Nuttx. In this way, nodes connect to the cloud, where they are orchestrated based on their identity.

WP6T62-0X Hardware Edge Controller

The hardware Edge Controller based on a network-on-chip (NoC) multicore architecture was developed in WP4 and reported in D4.4. It supports heterogeneous cores connected via NoC. In order to allow multiple nodes communication within WP6, one of the NoC cores was devoted to function as a hardware gateway controller. It allows communication between on-chip and off-chip network, as described in D6.2.

8 Validation test plan and implementation of the Edge Controller

This section will be dedicated to the definition of the validation tests and their implementation for the validation of the Edge Controller and the components implemented in task T6.2. For further information on each component, refer to D6.2 and components from task T6.2 in the FRACTAL GitHub repository. In addition, to facilitate the reading of this section a complete overview of each test can be found in the Annexes (chapter 14).

8.1 Orchestration (Edge Controller)

According to D6.2, the main functionality of this component is to monitor (separately) the status of "n" number of nodes where it is deployed. It is in charge of collecting all the resources information inside each of the Fractal nodes and was designed following a modular design. This software is composed of two main modules: the metrics exporter and the resource manager. These modules are managed by the custom Edge Orchestrator, which is designed to modify and take actions on both Kubernetes and Docker nodes, as well as user-defined orchestrators, by using the information from the metrics-exporter.

It is based on the architecture illustrated in Figure 5.

8.1.1 Test planification

8.1.1.1 Define the testing scope and identify the functionality that needs to be tested

Since this component and the next one (WP6T62-06-mid-range-orchestration "Agent Nodes Controller") has the same code but were delivered as individual components in different repositories, we have differentiated their tests by adding EC (e.g. T01_WP6T62-06_EC) when it comes to Edge Controller and ANC (e.g. T01_WP6T62-06_ANC) when it comes to Agent Nodes Controller.

After careful study of this component, 4 test cases have been identified, which can be carried out.

1. Installation.
2. Validate if the master node can monitor several (2) workers' nodes.
3. Validate through the REST API if the "metrics exporter" is working properly.
4. Test how the resource manager behaves if the nodes are stressed.

a) T01_WP6T62-06_EC - Testing the installation of the component

Validation test	
Test ID	T01_WP6T62-06_EC
Test type	Functional-Installation
Test name	Testing the installation of the component
Date	15/11/2022
Tester's Name	Ana Bautista
Test scope or objective	
The objective of this test is to validate if the Edge Controller Orchestrator can be installed without any issues.	

Table 3 - Validation Test T01_WP6T62-06_EC

b) T02_WP6T62-06_EC - Testing if the master node can monitor several (2) workers' nodes

Validation test	
Test ID	T02_WP6T62-06_EC
Test type	Functional
Test name	Testing if the master node can monitor several (2) workers' nodes
Date	20/01/2023
Tester's Name	Ana Bautista
Test scope or objective	
The objective of this test is to validate if the master node can monitor two workers' nodes.	

Table 4 - Validation Test T02_WP6T62-06_EC

c) T03_WP6T62-06_EC - Testing through the REST API if the metrics exporter is working properly

Validation test	
Test ID	T03_WP6T62-06_EC
Test type	Functional
Test name	Testing through the REST API if the metrics exporter is working properly
Date	15/11/2022
Tester's Name	Ana Bautista
Test scope or objective	
The objective of this test is to validate through the REST API if the metrics exporter is working properly.	

Table 5 - Validation Test T03_WP6T62-06_EC

d) T04_WP6T62-06_EC - Testing how the resource manager behaves if the nodes are stressed

Validation test	
Test ID	T04_WP6T62-06_EC
Test type	Functional
Test name	Testing how the resource manager behaves if the nodes are stressed
Date	15/11/2022
Tester's Name	Ana Bautista
Test scope or objective	
The objective of this test is to observe how the resource manager behaves if the nodes are stressed.	

Table 6 - Validation Test T04_WP6T62-06_EC

8.1.2 Test case development

In this section, the steps to be followed to carry out the validation tests for the Edge Controller were identified.

8.1.2.1 T01_WP6T62-06_EC - Testing the installation of the component

Steps	
Step 1	Prepare a node with Ubuntu and all needed dependencies (Python). This will be the master node.
Step 2	Deploy the metrics exporter container.
Step 3	Deploy the resource manager container.
Step 4	Deploy the custom orchestrator container.

Table 7 - Steps for Validation Test T01_WP6T62-06_EC

8.1.2.2 T02_WP6T62-06_EC - Testing if the master node can monitor several (2) workers' nodes

Steps	
Step 1	Prepare a node with Ubuntu and all needed dependencies (Python). This will be the master node.
Step 2	Deploy the metrics exporter container.
Step 3	Deploy the resource manager container.
Step 4	Deploy the custom orchestrator container.
Step 5	Prepare tow nodes with Ubuntu and all needed dependencies (Python). These will be the worker's nodes.
Step 6	Deploy the metrics exporter container on each of the worker nodes.
Step 7	Review the logs from the resource manager (deployed on the master node).

Table 8 - Steps for Validation Test T02_WP6T62-06_EC

8.1.2.3 T03_WP6T62-06_EC - Testing through the REST API if the metrics exporter is working properly

Steps	
Step 1	Prepare a node with Ubuntu and all needed dependencies (Python). This will be the master node.
Step 2	Deploy the metrics exporter container.
Step 3	Deploy the resource manager container.
Step 4	Deploy the custom orchestrator container.
Step 5	Prepare tow nodes with Ubuntu and all needed dependencies (Python). These will be the worker's nodes.
Step 6	Deploy the metrics exporter container on each of the worker nodes.
Step 7	Go to: http://<NODE_IP>:61208/api/3/cpu

Table 9 - Steps for Validation Test T03_WP6T62-06_EC

8.1.2.4 T04_WP6T62-06_EC - Testing how the resource manager behaves if the nodes are stressed

Steps	
Step 1	Prepare a node with Ubuntu and all needed dependencies (Python). This will be the master node.
Step 2	Deploy the metrics exporter container.
Step 3	Deploy the resource manager container.
Step 4	Deploy the custom orchestrator container.
Step 5	Prepare tow nodes with Ubuntu and all needed dependencies (Python). These will be the worker's nodes.
Step 6	Deploy the metrics exporter container on each of the worker nodes.
Step 7	Install **stress-ng o**n one of the worker nodes.
Step 8	Execute the command **stress-ng --cpu 8 --timeout 60s** which will stress the node for 60 seconds.
Step 9	Review the logs from the resource manager (deployed on the master node).
Step 10	Check the alerts and the metrics in the logs of the resource manager.

Table 10 - Steps for Validation Test T04_WP6T62-06_EC

8.1.3 Test environment setup

The steps to install and configure the component can be found in the FRACTAL project GitHub repository: https://github.com/project-fractal/WP6T62-06-edge-controller-orchestrator

8.1.4 Test execution

The following tables showthe results of the execution of each of the tests.

Some issues that were detected during the testing process are reported in remarks and some of them were already solved by developers.

8.1.4.1 T01_WP6T62-06_EC - Testing the installation of the component

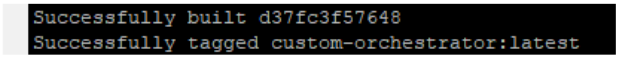
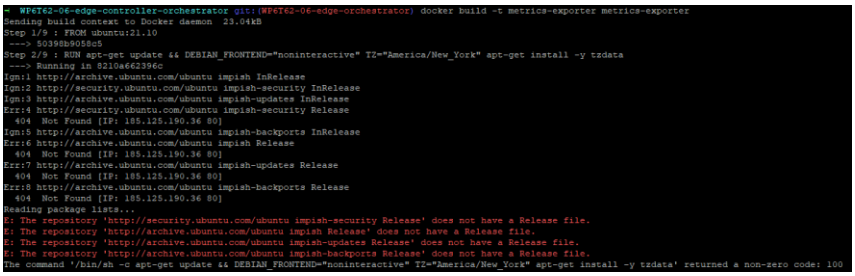
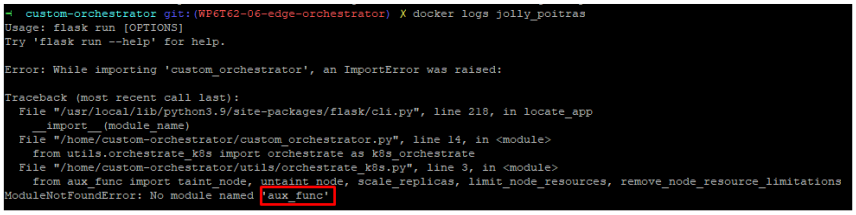
Results/Evidence	
	
Success criteria	
No error messages/All partial results are as expected	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-06-edge-controller-orchestrator
Test conditions	The metrics exporter, resource manager and custom orchestrator containers runs in the same node (master node).
Remarks	Two bugs were found during installation that have been reported and corrected by the developers. 1. apt-get update no longer works on containers with Ubuntu21.10 so we need to use Ubuntu22.04.  2. The import 'aux_func' was corrected. 
Test result	
Passed	

Table 11 - Results of the test T01_WP6T62-06_EC

8.1.4.2 T02_WP6T62-06_EC - Testing if the master node can monitor several (2) workers' nodes

Results/Evidence

Master node:

```
WP6T62-06-edge-controller-orchestrator git:(WP6T62-06-edge-orchestrator) docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED        STATUS        PORTS     NAMES
d110afe4abf6   resource-manager      "/bin/sh -c 'python3…"   20 hours ago   Up 20 hours             zen_feistel
27a41d81626b   metrics-exporter      "/bin/sh -c 'glances…"   20 hours ago   Up 20 hours             quirky_curran
02edb3c53b93   custom-orchestrator   "/bin/sh -c 'flask r…"   20 hours ago   Up 20 hours             sleepy_jones
```

Worker node 1:

```
WP6T62-06-edge-controller-orchestrator git:(WP6T62-06-edge-orchestrator) docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED        STATUS        PORTS     NAMES
b13d2b0b9af9   metrics-exporter   "/bin/sh -c 'glances…"   22 hours ago   Up 22 hours             peaceful_gauss
```

Worker node 2:

```
WP6T62-06-edge-controller-orchestrator git:(WP6T62-06-edge-orchestrator) docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED        STATUS        PORTS     NAMES
4570cb97cacc   metrics-exporter   "/bin/sh -c 'glances…"   22 hours ago   Up 22 hours             pedantic_antonelli
```

Review logs from the resorce manager:

```
2023-02-16 14:56:12,187 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total CPU usage: 0.5 %
2023-02-16 14:56:12,187 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total memory: 16.783896576 Gb
2023-02-16 14:56:12,187 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Available memory: 15.629955072000001 Gb
2023-02-16 14:56:12,187 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total memory usage: 6.9 %
2023-02-16 14:56:12,187 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Load avg 1 min: 0.19775390625 %
2023-02-16 14:56:12,187 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Alerts: []
2023-02-16 14:56:12,188 - logger - INFO - Tainted nodes: {}
2023-02-16 14:56:12,278 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total CPU usage: 0.6 %
2023-02-16 14:56:12,278 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total memory: 16.783896576 Gb
2023-02-16 14:56:12,278 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Available memory: 15.620268032 Gb
2023-02-16 14:56:12,278 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total memory usage: 6.9 %
2023-02-16 14:56:12,278 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Load avg 1 min: 0.0126953125 %
2023-02-16 14:56:12,278 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Alerts: []
2023-02-16 14:56:12,278 - logger - INFO - Tainted nodes: {}
2023-02-16 14:56:27,369 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total CPU usage: 0.5 %
2023-02-16 14:56:27,369 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total memory: 16.783896576 Gb
2023-02-16 14:56:27,369 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Available memory: 15.630966784000002 Gb
2023-02-16 14:56:27,369 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total memory usage: 6.9 %
2023-02-16 14:56:27,369 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Load avg 1 min: 0.1533203125 %
2023-02-16 14:56:27,369 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Alerts: []
2023-02-16 14:56:27,369 - logger - INFO - Tainted nodes: {}
2023-02-16 14:56:27,455 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total CPU usage: 0.5 %
2023-02-16 14:56:27,455 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total memory: 16.783896576 Gb
2023-02-16 14:56:27,455 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Available memory: 15.621595136000002 Gb
2023-02-16 14:56:27,455 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total memory usage: 6.9 %
2023-02-16 14:56:27,455 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Load avg 1 min: 0.00927734375 %
2023-02-16 14:56:27,456 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Alerts: []
2023-02-16 14:56:27,456 - logger - INFO - Tainted nodes: {}
2023-02-16 14:56:42,572 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total CPU usage: 0.5 %
2023-02-16 14:56:42,572 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total memory: 16.783896576 Gb
2023-02-16 14:56:42,572 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Available memory: 15.630843904 Gb
2023-02-16 14:56:42,572 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total memory usage: 6.9 %
2023-02-16 14:56:42,572 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Load avg 1 min: 0.34716796875 %
2023-02-16 14:56:42,572 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Alerts: []
2023-02-16 14:56:42,573 - logger - INFO - Tainted nodes: {}
2023-02-16 14:56:42,652 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total CPU usage: 0.5 %
2023-02-16 14:56:42,653 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total memory: 16.783896576 Gb
2023-02-16 14:56:42,653 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Available memory: 15.62216448 Gb
2023-02-16 14:56:42,653 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total memory usage: 6.9 %
2023-02-16 14:56:42,653 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Load avg 1 min: 0.00634765625 %
2023-02-16 14:56:42,653 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Alerts: []
2023-02-16 14:56:42,653 - logger - INFO - Tainted nodes: {}
```

Success criteria

No error messages/All partial results are as expected

Test observations

Test configuration	https://github.com/project-fractal/WP6T62-06-edge-controller-orchestrator
Test conditions	The metrics exporter, resource manager and custom orchestrator containers run in the master node. The metrics exporter runs in the two worker's nodes.
Remarks	Reviewing the logs from the resource manager it can be observed that the information from the worker's nodes is given in the right way (as expected).

Test result

Passed

Table 12 - Results of the test T02_WP6T62-06_EC

8.1.4.3 T03_WP6T62-06_EC - Testing through the REST API if the metrics exporter is working properly

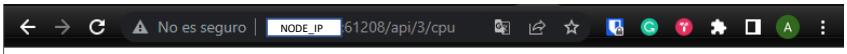
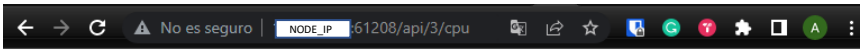
Results/Evidence	
Master node:  {"total": 1.1, "user": 0.7, "nice": 0.0, "system": 0.4, "idle": 98.8, "iowait": 0.1, "irq": 0.0, "softirq": 0.0, "steal": 0.0, "guest": 0.0, "guest_nice": 0.0, "time_since_update": 1128.0130908489227, "cpucore": 8, "ctx_switches": 7384901, "interrupts": 3861801, "soft_interrupts": 838841, "syscalls": 0} **Worker node 1:**  {"total": 0.8, "user": 0.7, "nice": 0.0, "system": 0.2, "idle": 99.1, "iowait": 0.0, "irq": 0.0, "softirq": 0.0, "steal": 0.0, "guest": 0.0, "guest_nice": 0.0, "time_since_update": 3.140522003173828, "cpucore": 8, "ctx_switches": 5630, "interrupts": 3113, "soft_interrupts": 1195, "syscalls": 0} **Worker node 2:**  {"total": 0.9, "user": 0.4, "nice": 0.0, "system": 0.3, "idle": 99.3, "iowait": 0.0, "irq": 0.0, "softirq": 0.0, "steal": 0.0, "guest": 0.0, "guest_nice": 0.0, "time_since_update": 3.3217294216156006, "cpucore": 8, "ctx_switches": 5666, "interrupts": 3047, "soft_interrupts": 1100, "syscalls": 0}	
Success criteria	
No error messages/All partial results are as expected	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-06-edge-controller-orchestrator
Test conditions	The metrics exporter, resource manager and custom orchestrator containers run in the master node. The metrics exporter runs in the two worker's nodes.
Remarks	The REST API exposed by the custom orchestrator is reached by the resource manager and provides information about the nodes previously configured (as expected).
Test result	
Passed	

Table 13 - Results of the test T03_WP6T62-06_EC

8.1.4.4 T04_WP6T62-06_EC - Testing how the resource manager behaves if the nodes are stressed

Results/Evidence

Master node:

```
WP6T62-06-edge-controller-orchestrator git:(WP6T62-06-edge-orchestrator) docker logs zen_feistel
```

Stressed worker node: fractal-k8s0.ipd.ikerlan.es

```
WP6T62-06-edge-controller-orchestrator git:(WP6T62-06-edge-orchestrator) stress-ng --cpu 8 --timeout 60s
stress-ng: info:  [1051804] dispatching hogs: 8 cpu
stress-ng: info:  [1051804] successful run completed in 60.07s (1 min, 0.07 secs)
```

Results 1:

Stressed worker node: fractal-k8s0.ipd.ikerlan.es

Worker node: fractal-k8s1.ipd.ikerlan.es

```
2023-02-17 12:40:44,687 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total CPU usage: 33.1 %
2023-02-17 12:40:44,687 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total memory: 16.783896576 Gb
2023-02-17 12:40:44,687 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Available memory: 15.542321152000001 Gb
2023-02-17 12:40:44,687 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total memory usage: 7.4 %
2023-02-17 12:40:44,687 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Load avg 1 min: 0.6962890625 %
2023-02-17 12:40:44,687 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Alerts: []
2023-02-17 12:40:44,688 - logger - INFO - Tainted nodes: {}
2023-02-17 12:40:44,792 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total CPU usage: 0.5 %
2023-02-17 12:40:44,792 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total memory: 16.783896576 Gb
2023-02-17 12:40:44,792 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Available memory: 15.575318528 Gb
2023-02-17 12:40:44,792 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total memory usage: 7.2 %
2023-02-17 12:40:44,792 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Load avg 1 min: 0.0 %
2023-02-17 12:40:44,792 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Alerts: []
2023-02-17 12:40:44,792 - logger - INFO - Tainted nodes: {}
2023-02-17 12:40:59,933 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total CPU usage: 100.0 %
2023-02-17 12:40:59,933 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total memory: 16.783896576 Gb
2023-02-17 12:40:59,933 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Available memory: 15.541850112 Gb
2023-02-17 12:40:59,933 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total memory usage: 7.4 %
2023-02-17 12:40:59,933 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Load avg 1 min: 2.314453125 %
2023-02-17 12:40:59,933 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Alerts: [[1676637659.0, -1, 'CRITICAL', 'CPU_TOTAL', 100.0, 100.0, 100.0, 100.0, 1, [], '', 'cpu_percent']]
2023-02-17 12:40:59,934 - logger - WARNING - CPU in node fractal-k8s0.ipd.ikerlan.es is over 80%: CPU = 100.0
2023-02-17 12:40:59,934 - logger - INFO - Tainted node fractal-k8s0.ipd.ikerlan.es to avoid scheduling of any other containers
2023-02-17 12:40:59,934 - logger - INFO - Tainted nodes: {'fractal-k8s0.ipd.ikerlan.es': 'NoSchedule'}
2023-02-17 12:41:00,064 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total CPU usage: 0.6 %
2023-02-17 12:41:00,064 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total memory: 16.783896576 Gb
2023-02-17 12:41:00,064 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Available memory: 15.574118400000001 Gb
2023-02-17 12:41:00,065 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total memory usage: 7.2 %
2023-02-17 12:41:00,065 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Load avg 1 min: 0.0 %
2023-02-17 12:41:00,065 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Alerts: []
2023-02-17 12:41:00,065 - logger - INFO - Tainted nodes: {'fractal-k8s0.ipd.ikerlan.es': 'NoSchedule'}
```

Results 2:

```
2023-02-17 12:41:30,417 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total CPU usage: 100.0 %
2023-02-17 12:41:30,417 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total memory: 16.783896576 Gb
2023-02-17 12:41:30,417 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Available memory: 15.539376128 Gb
2023-02-17 12:41:30,417 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total memory usage: 7.4 %
2023-02-17 12:41:30,417 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Load avg 1 min: 4.5556640625 %
2023-02-17 12:41:30,417 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Alerts: [[1676637659.0, -1, 'CRITICAL', 'CPU_TOTAL', 100.0, 100.0, 100.0, 300.0, 3,
.04, 0.0, 0.0, 0.0], 'pid': 1051806, 'num_threads': 1, 'memory_info': [7311360, 55607296, 3723264, 1728512, 0, 27832320, 0], 'nice': 0, 'ppid': 1051804, '
 1, 2, 3, 4, 5, 6, 7], 'num_ctx_switches': [1, 9550], 'memory_swap': None, 'tcp': None, 'udp': None, 'extended_stats': True, 'key': 'pid', 'time_since_upd
, '', ''], 'username': 'metricsexporter'}, {'cpu_percent': 99.8, 'io_counters': [0, 0, 0, 0, 0], 'name': 'stress-ng-cpu', 'status': 'R', 'cpu_times': [50.
7832320, 0], 'nice': 0, 'ppid': 1051804, 'memory_percent': 0.04356175556071301, 'gids': [1000, 1000, 1000], 'key': 'pid', 'time_since_update': 15.21255326
ame': 'metricsexporter'}, {'cpu_percent': 99.6, 'io_counters': [0, 0, 0, 0, 0], 'name': 'stress-ng-cpu', 'status': 'R', 'cpu_times': [50.6, 0.0, 0.0, 0.0,
: 0, 'ppid': 1051804, 'memory_percent': 0.04392782073349211, 'gids': [1000, 1000, 1000], 'key': 'pid', 'time_since_update': 15.212553262710571, 'cmdline':
ter']}], '', 'cpu_percent']]
2023-02-17 12:41:30,418 - logger - WARNING - CPU in node fractal-k8s0.ipd.ikerlan.es is over 80%: CPU = 100.0
2023-02-17 12:41:30,418 - logger - INFO - Node fractal-k8s0.ipd.ikerlan.es is already tainted as NoSchedule
2023-02-17 12:41:30,418 - logger - INFO - Tainted nodes: {'fractal-k8s0.ipd.ikerlan.es': 'NoSchedule'}
2023-02-17 12:41:30,500 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total CPU usage: 0.6 %
2023-02-17 12:41:30,500 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total memory: 16.783896576 Gb
2023-02-17 12:41:30,500 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Available memory: 15.574810624000001 Gb
2023-02-17 12:41:30,501 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total memory usage: 7.2 %
2023-02-17 12:41:30,501 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Load avg 1 min: 0.0 %
2023-02-17 12:41:30,501 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Alerts: []
2023-02-17 12:41:30,501 - logger - INFO - Tainted nodes: {'fractal-k8s0.ipd.ikerlan.es': 'NoSchedule'}
2023-02-17 12:41:45,597 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total CPU usage: 61.5 %
2023-02-17 12:41:45,598 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total memory: 16.783896576 Gb
2023-02-17 12:41:45,598 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Available memory: 15.564402688000001 Gb
2023-02-17 12:41:45,598 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total memory usage: 7.3 %
2023-02-17 12:41:45,598 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Load avg 1 min: 4.677734375 %
2023-02-17 12:41:45,598 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Alerts: [[1676637659.0, 1676637705.0, 'CRITICAL', 'CPU_TOTAL', 100.0, 100.0, 100.0,
 [50.63, 0.04, 0.0, 0.0, 0.0], 'pid': 1051806, 'num_threads': 1, 'memory_info': [7311360, 55607296, 3723264, 1728512, 0, 27832320, 0], 'nice': 0, 'ppid':
nity': [0, 1, 2, 3, 4, 5, 6, 7], 'num_ctx_switches': [1, 9550], 'memory_swap': None, 'tcp': None, 'udp': None, 'extended_stats': True, 'key': 'pid', 'time
'', '', '', ''], 'username': 'metricsexporter'}, {'cpu_percent': 99.8, 'io_counters': [0, 0, 0, 0, 0], 'name': 'stress-ng-cpu', 'status': 'R', 'cpu_ti
8512, 0, 27832320, 0], 'nice': 0, 'ppid': 1051804, 'memory_percent': 0.04356175556071301, 'gids': [1000, 1000, 1000], 'key': 'pid', 'time_since_update': 1
'], 'username': 'metricsexporter'}, {'cpu_percent': 99.6, 'io_counters': [0, 0, 0, 0, 0], 'name': 'stress-ng-cpu', 'status': 'R', 'cpu_times': [50.6, 0.0,
0], 'nice': 0, 'ppid': 1051804, 'memory_percent': 0.04392782073349211, 'gids': [1000, 1000, 1000], 'key': 'pid', 'time_since_update': 15.212553262710571,
tricsexporter']}], '', 'cpu_percent']]
2023-02-17 12:41:45,598 - logger - INFO - Node fractal-k8s0.ipd.ikerlan.es untainted. All parameters are below critical tresholds.
```

Success criteria	
No error messages/All partial results are as expected	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-06-edge-controller-orchestrator
Test conditions	The metrics exporter, resource manager and custom orchestrator containers run in the master node. The metrics exporter runs in the two worker's nodes. The node called **fractal-k8s0.ipd.ikerlan.es** is the node that was stressed.
Remarks	As it can be observed in the "Results 1" screenshot: when the CPU usage of a node is over 80% it is considered as tainted (low on resources and restricted) as NoSchedule. According to the component documentation, if any of the monitored resources are above some fixed thresholds, that node is no longer able to perform any new container deployments until the resource limitation is lifted.
Test result	
Passed	

Table 14 - Results of the test T04_WP6T62-06_EC

8.2 Orchestration (Agent Nodes Controller)

This component is part of the Edge Controller, so it has to do with orchestration, in this case of "tasks". The name of this component is **WP6T62-06-mid-range-orchestration** and according to the developers' documentation, this software consists of three main components based on the architecture illustrated in Figure 176:

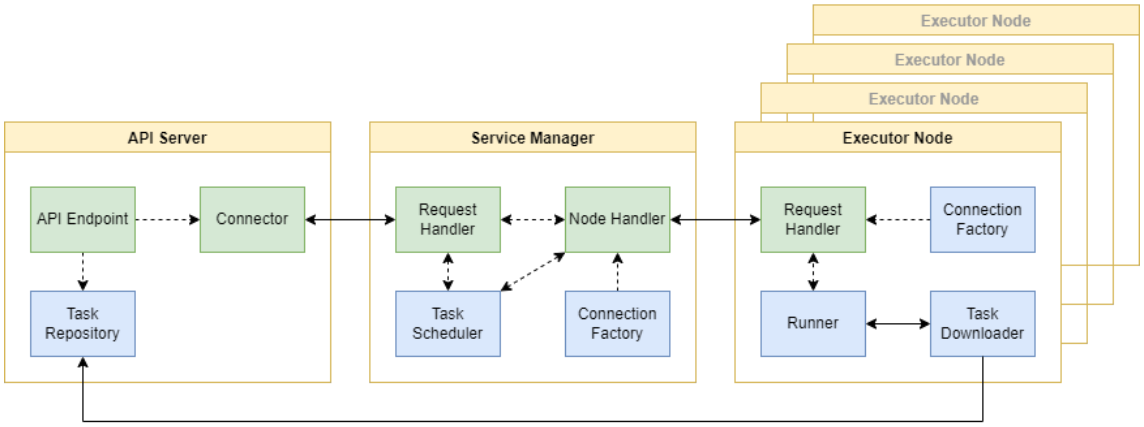


Figure 17: WP6T62-06-mid-range-orchestration architecture (designed in task T6.2)

The main functionality of this component is to orchestrate tasks execution on the available "Executor Nodes". For more details on how this component works, see the D6.2 deliverable and the GitHub repository https://github.com/project-fractal/WP6T62-06-mid-range-orchestration.

8.2.1 Test planification

8.2.1.1 Define the testing scope and identify the functionality that needs to be tested

After careful study of this component, 6 test cases have been identified, which can be carried out.

1. Installation.
2. Basic orchestration functionality.
3. Validate that a running task can be deleted.
4. Validate that a running task can be stopped and started again.
5. Validate that running multiple tasks is possible and list their state.
6. Validate the behaviour of the orchestrator with multiple Executor Nodes.

a) T01_WP6T62-06_ANC - Testing that the Agent Nodes Controller can be installed without any issues

Validation test	
Test ID	T01_WP6T62-06_ANC
Test type	Functional
Test name	Testing that the Agent nodes controller can be installed without any issues
Date	20/01/2023
Tester's Name	Ana Bautista, Adrian Moran
Test scope or objective	
The objective of the test is to validate if the Agent nodes controller can be installed without any issues.	

Table 15 - Validation Test T01_WP6T62-06_ANC

b) T02_WP6T62-06_ANC - Testing basic orchestration functionality.

Validation test	
Test ID	T02_WP6T62-06_ANC
Test type	Functional
Test name	Testing WP6T62-06-mid-range-orchestration component
Date	09/02/2023
Tester's Name	Ana Bautista, Adrian Moran
Test scope or objective	
The objective of the test is validate basic orchestration functionality.	

Table 16 - Validation Test T02_WP6T62-06_ANC

c) T03_WP6T62-06_ANC - Testing that a running task can be deleted

Validation test	
Test ID	T03_WP6T62-06_ANC
Test type	Functional
Test name	Testing WP6T62-06-mid-range-orchestration component
Date	09/02/2023
Tester's Name	Ana Bautista, Adrian Moran
Test scope or objective	
The objective of the test is validate that a running task can be deleted.	

Table 17 - Validation Test T03_WP6T62-06_ANC

d) T04_WP6T62-06_ANC - Testing that a running task can be stopped and started again

Validation test	
Test ID	T04_WP6T62-06_ANC
Test type	Functional
Test name	Testing WP6T62-06-mid-range-orchestration component
Date	09/02/2023
Tester's Name	Ana Bautista, Adrian Moran
Test scope or objective	
The objective of the test is validate that a running task can be stopped and started again.	

Table 18 - Validation Test T04_WP6T62-06_ANC

e) T05_WP6T62-06_ANC - Testing that a running multiple tasks is possible and list their state

Validation test	
Test ID	T05_WP6T62-06_ANC
Test type	Functional
Test name	Testing WP6T62-06-mid-range-orchestration component
Date	09/02/2023
Tester's Name	Ana Bautista, Adrian Moran
Test scope or objective	
The objective of the test is validate that a running multiple tasks is possible and list their state.	

Table 19 - Validation Test T05_WP6T62-06_ANC

f) T06_WP6T62-06_ANC - Testing the behaviour of the orchestrator with multiple Executor Nodes

Validation test	
Test ID	T06_WP6T62-06_ANC
Test type	Functional
Test name	Testing WP6T62-06-mid-range-orchestration component
Date	09/02/2023
Tester's Name	Ana Bautista, Adrian Moran
Test scope or objective	
The objective of the test is validate the behaviour of the orchestrator with multiple Executor Nodes.	

Table 20 - Validation Test T06_WP6T62-06_ANC

8.2.2 Test case development

In this section, the steps to be followed to carry out the validation tests for the Agent Nodes Controller were identified.

8.2.2.1 T01_WP6T62-06_ANC - Testing that the Agent Nodes Controller can be installed without any issues

Steps	
Step 1	Prepare a node with Ubuntu and all needed dependencies (Python).
Step 2	Launch backend-service-manager (Service Manager).
Step 3	Launch API Server.
Step 4	Launch frontend-service-manager (Executor Node).

Table 21 - Steps for Validation Test T01_WP6T62-06_ANC

8.2.2.2 T02_WP6T62-06_ANC - Testing basic orchestration functionality

Steps	
Step 1	Prepare a node with Ubuntu and all needed dependencies (Python).
Step 2	Launch backend-service-manager (Service Manager).
Step 3	Launch API Server.
Step 4	Launch frontend-service-manager (Executor Node).
Step 5	From a different device, use REST API to create a new task.

Table 22 - Steps for Validation Test T02_WP6T62-06_ANC

8.2.2.3 T03_WP6T62-06_ANC - Testing that a running task can be deleted

Steps	
Step 1	Prepare a node with Ubuntu and all needed dependencies (Python).
Step 2	Launch backend-service-manager (Service Manager).
Step 3	Launch API Server.
Step 4	Launch frontend-service-manager (Executor Node).
Step 5	From a different device, use REST API to create a new task and delete it.

Table 23 - Steps for Validation Test T03_WP6T62-06_ANC

8.2.2.4 T04_WP6T62-06_ANC - Testing that a running task can be stopped and started again

Steps	
Step 1	Prepare a node with Ubuntu and all needed dependencies (Python).
Step 2	Launch backend-service-manager (Service Manager).
Step 3	Launch API Server.
Step 4	Launch frontend-service-manager (Executor Node).
Step 5	From a different device, use REST API to create a new task, stop and then start it.

Table 24 - Steps for Validation Test T04_WP6T62-06_ANC

8.2.2.5 T05_WP6T62-06_ANC - Testing that a running multiple tasks is possible and list their state

Steps	
Step 1	Prepare a node with Ubuntu and all needed dependencies (Python).
Step 2	Launch backend-service-manager (Service Manager).
Step 3	Launch API Server.
Step 4	Launch frontend-service-manager (Executor Node).
Step 5	From a different device, use REST API to create a new tasks.

Table 25 - Steps for Validation Test T05_WP6T62-06_ANC

8.2.2.6 T06_WP6T62-06_ANC - Testing the behaviour of the orchestrator with multiple Executor Nodes

Steps	
Step 1	Prepare a node with Ubuntu and all needed dependencies (Python).
Step 2	Launch backend-service-manager (Service Manager).
Step 3	Launch API Server.
Step 4	Launch two instances of frontend-service-manager (Executor Node).
Step 5	From a different device, use REST API to create two simultaneous tasks.

Table 26 - Steps for Validation Test T06_WP6T62-06_ANC

8.2.3 Test environment setup

The steps to install and configurate the component can be found in the FRACTAL project repository: https://github.com/project-fractal/WP6T62-06-mid-range-orchestration

8.2.4 Test execution

The following tables show the results of the execution of each of the tests.

Some issues that were detected during the testing process are reported in remarks and were also reported as feedback to developers.

8.2.4.1 T01_WP6T62-06_ANC - Testing that the Agent Nodes Controller can be installed without any issues

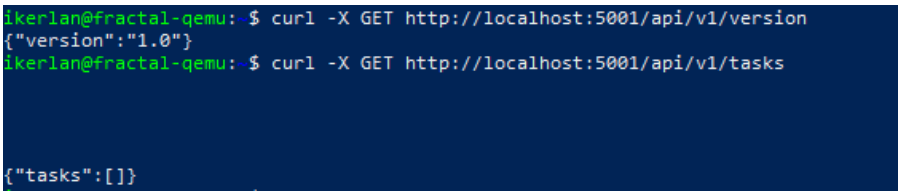
Results/Evidence	
	
Success criteria	
No error messages/All partial results are as expected	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-06-mid-range-orchestration
Test conditions	API Server, Service Manager and Executor Node runs in the same node.
Remarks	All the subcomponents are up and running.
Test result	
Passed	

Table 27 - Results of the test T01_WP6T62-06_ANC

8.2.4.2 T02_WP6T62-06_ANC - Testing basic orchestration functionality

Results/Evidence	

Client:
`→ task curl -X POST http://172.16.58.4:5001/api/v1/tasks/test -F file=@test_task.py -F "cmd=test_task.py" -F "rt="`
`{"status":"task: test created successfuly."}`
 	
API:
`172.16.105.43 - - [09/Feb/2023 11:13:59] "DELETE /api/v1/tasks/test HTTP/1.1" 200 -`
`172.16.105.43 - - [09/Feb/2023 11:14:58] "POST /api/v1/tasks/test HTTP/1.1" 201 -`
 	
Executor Node:
`2023-02-09 11:14:58,501 INFO heartbeat received`
`('task', 71, 'json', {'task_name': 'test', 'args_to_run': 'test_task.py', 'return_type': ''})`
`2023-02-09 11:14:58,514 INFO running task: test`
`2023-02-09 11:15:00,504 INFO heartbeat received`
`2023-02-09 11:15:02,507 INFO heartbeat received`
`2023-02-09 11:15:03,535 INFO task: test completed with return code: 0`
`2023-02-09 11:15:04,508 INFO heartbeat received`	

Success criteria	
No error messages/All partial results are as expected	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-06-mid-range-orchestration
Test conditions	API Server, Service Manager and Executor Node runs in the same node.
Remarks	Basic workflow is completed successfully.
Test result	
Passed	

Table 28 - Results of the test T02_WP6T62-06_ANC

8.2.4.3 T03_WP6T62-06_ANC - Testing that a running task can be deleted

Results/Evidence	

Client:
`→ task curl -X DELETE http://172.16.58.4:5001/api/v1/tasks/test`
`{"backend-status":{"status":"ok"},"status":"task: test deleted successfuly from API host."}`
 	
API:
`172.16.105.43 - - [09/Feb/2023 11:17:23] "DELETE /api/v1/tasks/test HTTP/1.1" 200 -`	

Success criteria	
No error messages/All partial results are as expected	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-06-mid-range-orchestration
Test conditions	API Server, Service Manager and Executor Node runs in the same node.
Remarks	Task is deleted from API Server repository succesfully.
	Deleted task is not deleted from Executor node .tasks folder, which can lead to problems like lack of storage or DoS attacks.
Test result	
Passed	

Table 29 - Results of the test T03_WP6T62-06_ANC

8.2.4.4 T04_WP6T62-06_ANC - Testing that a running task can be stopped and started again

Results/Evidence

Client:

```
→  task curl -X POST http://172.16.58.4:5001/api/v1/tasks/test -F file=@test_task.py -F "cmd=test_task.py" -F "rt="
{"status":"task: test created successfuly."}
→  task curl -X POST http://172.16.58.4:5001/api/v1/tasks/test/stop
{"status":"ok"}
→  task curl -X POST http://172.16.58.4:5001/api/v1/tasks/test/start
{"status":"ok"}
→  task curl -X DELETE http://172.16.58.4:5001/api/v1/tasks/test
{"backend-status":{"status":"ok"},"status":"task: test deleted successfuly from API host."}
```

API:

```
172.16.105.43 - - [09/Feb/2023 11:18:35] "POST /api/v1/tasks/test HTTP/1.1" 201 -
127.0.0.1 - - [09/Feb/2023 11:18:35] "GET /api/v1/tasks/test/download HTTP/1.1" 200 -
172.16.105.43 - - [09/Feb/2023 11:18:45] "POST /api/v1/tasks/test/stop HTTP/1.1" 200 -
172.16.105.43 - - [09/Feb/2023 11:18:48] "POST /api/v1/tasks/test/start HTTP/1.1" 200 -
127.0.0.1 - - [09/Feb/2023 11:18:48] "GET /api/v1/tasks/test/download HTTP/1.1" 200 -
from server {'status': 'ok'}
172.16.105.43 - - [09/Feb/2023 11:18:58] "DELETE /api/v1/tasks/test HTTP/1.1" 200 -
```

Executor Node:

```
2023-02-09 11:18:34,783 INFO heartbeat received
('task', 71, 'json', {'task_name': 'test', 'args_to_run': 'test_task.py', 'return_type': ''})
2023-02-09 11:18:35,796 INFO running task: test
2023-02-09 11:18:36,785 INFO heartbeat received
2023-02-09 11:18:38,788 INFO heartbeat received
2023-02-09 11:18:40,791 INFO heartbeat received
2023-02-09 11:18:42,794 INFO heartbeat received
2023-02-09 11:18:44,797 INFO heartbeat received
2023-02-09 11:18:45,786 INFO task stopped
2023-02-09 11:18:45,786 INFO task: test terminated with return code: -15
2023-02-09 11:18:46,800 INFO heartbeat received
2023-02-09 11:18:48,803 INFO heartbeat received
('task', 71, 'json', {'task_name': 'test', 'args_to_run': 'test_task.py', 'return_type': ''})
2023-02-09 11:18:48,852 INFO running task: test
2023-02-09 11:18:50,806 INFO heartbeat received
2023-02-09 11:18:52,809 INFO heartbeat received
2023-02-09 11:18:54,812 INFO heartbeat received
2023-02-09 11:18:56,814 INFO heartbeat received
2023-02-09 11:18:58,512 INFO task stopped
2023-02-09 11:18:58,513 INFO task: test terminated with return code: -15
```

Success criteria	
No error messages/All partial results are as expected	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-06-mid-range-orchestration
Test conditions	API Server, Service Manager and Executor Node runs in the same node.
Remarks	Task lifecycle in correctly handled.
Test result	
Passed	

Table 30 - Results of the test T04_WP6T62-06_ANC

8.2.4.5 T05_WP6T62-06_ANC - Testing that a running multiple tasks is possible and list their state

Results/Evidence

Client:

```
→  task curl -X GET http://172.16.58.4:5001/api/v1/tasks
{"tasks":[]}
→  task curl -X POST http://172.16.58.4:5001/api/v1/tasks/test -F file=@test_task.py -F "cmd=test_task.py" -F "rt="
{"status":"task: test created successfuly."}
→  task curl -X GET http://172.16.58.4:5001/api/v1/tasks
{"tasks":["test"]}
→  task curl -X POST http://172.16.58.4:5001/api/v1/tasks/test2 -F file=@test_task.py -F "cmd=test_task.py" -F "rt="
{"status":"task: test2 created successfuly."}
→  task curl -X GET http://172.16.58.4:5001/api/v1/tasks
{"tasks":["test","test2"]}
→  task curl -X GET http://172.16.58.4:5001/api/v1/tasks/test/status
{"status":"ok","task_status":"running"}
→  task curl -X GET http://172.16.58.4:5001/api/v1/tasks/test2/status
{"status":"ok","task_status":"created"}
→  task
```

API:

```
172.16.105.43 - - [09/Feb/2023 11:21:41] "GET /api/v1/tasks HTTP/1.1" 404 -
172.16.105.43 - - [09/Feb/2023 11:21:49] "POST /api/v1/tasks/test HTTP/1.1" 201 -
127.0.0.1 - - [09/Feb/2023 11:21:50] "GET /api/v1/tasks/test/download HTTP/1.1" 200 -
from server {'status': 'ok', 'tasks': ['test']}
172.16.105.43 - - [09/Feb/2023 11:21:51] "GET /api/v1/tasks HTTP/1.1" 200 -
172.16.105.43 - - [09/Feb/2023 11:21:55] "POST /api/v1/tasks/test2 HTTP/1.1" 201 -
from server {'status': 'ok', 'tasks': ['test', 'test2']}
172.16.105.43 - - [09/Feb/2023 11:21:59] "GET /api/v1/tasks HTTP/1.1" 200 -
172.16.105.43 - - [09/Feb/2023 11:22:12] "GET /api/v1/tasks/test/status HTTP/1.1" 200 -
172.16.105.43 - - [09/Feb/2023 11:22:16] "GET /api/v1/tasks/test2/status HTTP/1.1" 200 -
```

Success criteria

No error messages/All partial results are as expected

Test observations

Test configuration	https://github.com/project-fractal/WP6T62-06-mid-range-orchestration
Test conditions	API Server, Service Manager and Executor Node runs in the same node.
Remarks	It is possible to list the tasks. It is possible to add multiple tasks and get their status. A bug has occured generating two task, one named "test" and other one named "test2". Executor Node computes the execution path from name, and since it does not deletes the old tasks, this leads to failure:

```
57
58      logging.info(f"running task: {task_name}")
59      await socket.send("task-running", {"task_name": task_name})
60      task_dir = [dir
61          for dir in os.listdir(os.getcwd() + "/.tasks/") if (task_name in dir and "venv" != dir)
62          ]
63      task_dir.sort()
64      task_dir = os.getcwd() + "/.tasks/" + task_dir[-1]
65      python_dir = os.getcwd() + "/.tasks/" + "venv/bin"
66      task_args = task_args.split(" ")
67      try:
68          PROCESS = await asyncio.subprocess.create_subprocess_exec(
```

This way of computing task_dir is bugged. If the same node has executed in their lifetime a tasks called "test" and "test2", there won't be any chance of running "test" task, since this piece of code will always select last "test2-timestamp" folder:

```
drwxrwxr-x 2 ikerlan ikerlan 4096 Feb  9 11:18 test-1675941515
drwxrwxr-x 2 ikerlan ikerlan 4096 Feb  9 11:18 test-1675941528
drwxrwxr-x 2 ikerlan ikerlan 4096 Feb  9 11:21 test-1675941710
drwxrwxr-x 2 ikerlan ikerlan 4096 Feb  9 11:24 test-1675941879
drwxrwxr-x 2 ikerlan ikerlan 4096 Feb  9 11:27 test-1675942024
drwxrwxr-x 2 ikerlan ikerlan 4096 Feb  9 11:30 test-1675942222
drwxrwxr-x 2 ikerlan ikerlan 4096 Feb  9 11:23 test2-1675941833
drwxrwxr-x 6 ikerlan ikerlan 4096 Feb  9 11:30 venv
→  .tasks git:(main) ✗
```

Test result

Not passed

Table 31 - Results of the test T05_WP6T62-06_ANC

8.2.4.6 T06_WP6T62-06_ANC - Testing the behaviour of the orchestrator with multiple Executor Nodes

Results/Evidence

Client:

```
→  task curl -X POST http://172.16.58.4:5001/api/v1/tasks/test -F file=@test_task.py -F "cmd=test_task.py" -F "rt="
{"status":"task: test created successfuly."}
→  task curl -X POST http://172.16.58.4:5001/api/v1/tasks/ikerlan -F file=@test_task.py -F "cmd=test_task.py" -F "rt="
{"status":"task: ikerlan created successfuly."}
```

API:

```
172.16.105.43 - - [09/Feb/2023 13:22:36] "POST /api/v1/tasks/test HTTP/1.1" 201 -
127.0.0.1 - - [09/Feb/2023 13:22:36] "GET /api/v1/tasks/test/download HTTP/1.1" 200 -
172.16.105.43 - - [09/Feb/2023 13:22:42] "POST /api/v1/tasks/ikerlan HTTP/1.1" 201 -
127.0.0.1 - - [09/Feb/2023 13:22:42] "GET /api/v1/tasks/ikerlan/download HTTP/1.1" 200 -
```

Executor Node 1:

```
2023-02-09 13:22:34,827 INFO heartbeat received
('task', 71, 'json', {'task_name': 'test', 'args_to_run': 'test_task.py', 'return_type': ''})
2023-02-09 13:22:36,821 INFO running task: test
2023-02-09 13:22:36,828 INFO heartbeat received
```

Executor Node 2:

```
2023-02-09 13:22:42,737 INFO heartbeat received
('task', 74, 'json', {'task_name': 'ikerlan', 'args_to_run': 'test_task.py', 'return_type': ''})
2023-02-09 13:22:42,823 INFO running task: ikerlan
2023-02-09 13:22:44,739 INFO heartbeat received
```

Success criteria	
No error messages/All partial results are as expected	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-06-mid-range-orchestration
Test conditions	API Server, Service Manager and Executor Node runs in the same node.
Remarks	Multiple nodes works fine.
Test result	
Passed	

Table 32 - Results of the test T06_WP6T62-06_ANC

8.3 Runtime Manager

The Runtime Manager coordinates and manages task scheduling and load balancing operation between modules in one or more FRACTAL nodes at runtime. It performs the scheduling of various operations which are entirely configurable. In addition, it provides load balancing capabilities using the interface with the Load Balancer component, sending the task execution to a different node.

8.3.1 Test planification

In this section Runtime Manager functionalities are defined as per D6.2 and, for each function, test cases are defined. We will have three test cases for the first function and two test cases for the second one.

8.3.1.1 Define the testing scope and identify the functionality that needs to be tested

The functionality that was identified for the Runtime Manager Component that needs to be tested are related to how to distribute the computational load and the task scheduling, they were defined as follows:

1. The Runtime Manager mustbe able to distribute the computational load;
2. Execution of configured task related to the task scheduling.

For the first functioningthree test cases are defined and shown below:

a) T01_WP6T62-03 - Testing interaction between nodes with local node overloaded

Validation test	
Test ID	T01_WP6T62-03
Test type	Functional
Test Name	Testing interaction between nodes with local node overloded
Date	05/12/2022
Tester's Name	Luca Visconti (Modis Consulting SRL)
Test scope or objective	
The objective of this test is to validate the correct interaction and data exchange between nodes	

Table 33 - Validation Test T01_WP6T62-03

b) T02_WP6T62-03 - Testing interaction in the local node when the local node can perform the computation

Validation test	
Test ID	T02_WP6T62-03
Test type	Functional
Test Name	Testing interaction in the local node when the local node can perform the computation
Date	05/12/2022
Tester's Name	Luca Visconti (Modis Consulting SRL)
Test scope or objective	
The objective of this test is to validate the correct interaction and data exchange in the local node	

Table 34 - Validation Test T02_WP6T62-03

c) T03_WP6T62-03 - Testing interaction between nodes with Node 1 and Node 2 overloaded

Validation test	
Test ID	T03_WP6T62-03
Test type	Functional
Test Name	Testing interaction between nodes with Node 1 and Node 2 overloaded
Date	05/12/2022
Tester's Name	Luca Visconti (Modis Consulting SRL)
Test scope or objective	
The objective of this test is to validate the correct interaction and data exchange between nodes	

Table 35 - Validation Test T03_WP6T62-03

For the second functioning there are defined two test cases that are shown below:

d) T04_WP6T62-03 - Task Scheduling on the local node

Validation test	
Test ID	T04_WP6T62-03
Test type	Functional
Test Name	Task Scheduling on the local node
Date	05/12/2022
Tester's Name	Luca Visconti (Modis Consulting SRL)
Test scope or objective	
The objective of this test is to validate the correct execution of the Task in the local node	

Table 36 - Validation Test T04_WP6T62-03

e) T05_WP6T62-03 - Task Scheduling on the remote node

Validation test	
Test ID	T05_WP6T62-03
Test type	Functional
Test Name	Task Scheduling on the remote node
Date	05/12/2022
Tester's Name	Luca Visconti (Modis Consulting SRL)
Test scope or objective	
The objective of this test is to validate the correct execution of the tasks in the remote node	

Table 37 - Validation Test T05_WP6T62-03

8.3.2 Test case development

All test cases are based on the same architecture. The test environment is presented in detail in section 8.3.3.

There are three nodes and a Runtime Manager on each node ("RMx" on node "Nx").

N1, the local node, is on a Xilinx board, N2 and N3, the remote nodes are virtual machines.

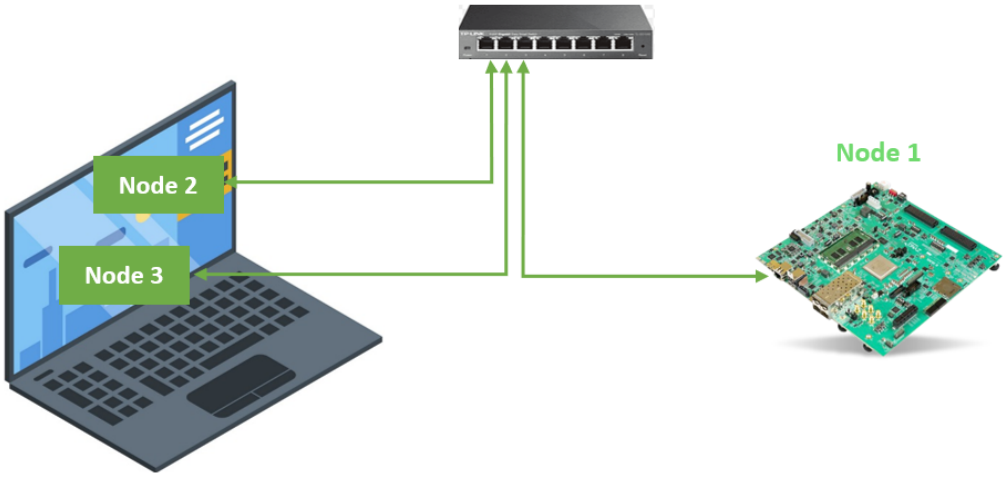


Figure 18 - Node interconnection

8.3.2.1 T01_WP6T62-03 - Testing interaction between nodes with local node overloaded

Step 1: Create the condition that overload the Node 1

For the overload condition of the N1 node, a simple "While True" cycle was executed on the python interpreter. This occupies resources at the node's processor. This procedure needs to be repeated until the processor is not overloaded (>85%). The procedure is described as follows:

- Open the command prompt of the N1 nodes and execute the following line:
  - python3 #to open the python interpreter
  - while True: #infinite cycle
    print(1)

As it is possible to notice in the table (using the htop command) the overload condition is verified and the node is in an overloaded state.

Step 2: Send the command of the execution flow to RM1 running "test_mqtt_published.py"

Expected Results: having generated the overload condition for the N1 node, RM1 will ask at Load Balancer instances which of the nodes can perform the workflow. The load balancer will respond with {"id_node":2}. RM1 will direct the workflows to the N2 node as shown in the figures below.

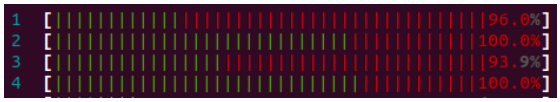
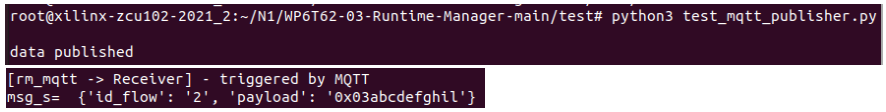
Steps	
Step 1	Create the condition that overload the Node 1 
Step 2	Send the command of the execution flow to RM1 running "test_mqtt_published.py" 

Table 38 - Steps for Validation Test T01_WP6T62-03

8.3.2.2 T02_WP6T62-03 - Testing interaction in the local node when the local node can perform the computation

Step 1: Send the command of the execution flow to RM1 running "test_mqtt_published.py"

Expected Result: Having not generated the overload condition for N1 nodes, it will be to execute the computational load locally. RM1 will ask at load balancer instance what is the node that can execute the computational load, the load balancer respond with {"id_node": none}. RM1 will execute the flows on the node N1 as shown below in the figure.

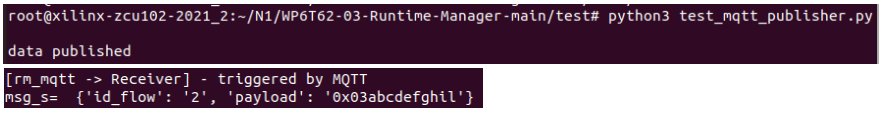
Steps	
Step 1	Send the command of the execution flow to RM1 running "test_mqtt_published.py" 

Table 39 – Steps for Validation Test T02_WP6T62-03

8.3.2.3 T03_WP6T62-03 - Testing interaction between nodes with Node 1 and Node 2 overloaded

Step 1: Create the condition that overload the Node 1

For the overload condition of the N1 node, it was executed some simple "while True" cycles on the python interpreter. This occupies resources at the node's processor. This procedure needs to be repeated until the processor is not overloaded. The procedure is described as follows:

- Open the command prompt of the N1 nodes and execute the following line:
  - python3 #to open the python interpreter
  - while True: #infinite cycle
    print(1)

As is possible to notice in the table (using the htop command), the overload condition (>85%) is verified and the node is to be in an overloaded state.

Step 2: Create the condition that overload the Node 2

In the same way as Node N1, - has been performed the overload condition on Node N2.

Step 3: Send the command of the execution flow to RM1 running "test_mqtt_published.py"

Expected Results: having generated the overload condition for the N1 and N2 nodes, RM1 will ask at Load Balancer instances which of the nodes can perform the workflow. The load balancer will respond with {"id_node":3}. RM1 will direct the workflows to the N3 node as shown in the table.

Steps	
Step 1	Create the condition that overload the Node 1 1 [||||||||||||||||||||||||||||||||||96.1%] 2 [||||||||||||||||||||||||||||||||||100.0%] 3 [||||||||||||||||||||||||||||||||||100.0%] 4 [||||||||||||||||||||||||||||||||||93.8%]
step 2	Create the condition that overload the Node 2 1 [||||||||||||||||||||||||||||||||||95.3%] 2 [||||||||||||||||||||||||||||||||||92.3%] 3 [||||||||||||||||||||||||||||||||||89.3%]
Step 3	Send the command of the execution flow to RM1 running "test_mqtt_published.py" root@xilinx-zcu102-2021_2:~/N1/WP6T62-03-Runtime-Manager-main/test# python3 test_mqtt_publisher.py data published [rm_mqtt -> Receiver] - triggered by MQTT msg_s= {'id_flow': '2', 'payload': '0x03abcdefghil'}

Table 40 – Steps for Validation Test T03_WP6T62-03

8.3.2.4 T04_WP6T62-03 - Task Scheduling on the local node

Step 1: Send the command of the execution flow "1" to RM1 running "test_mqtt_published.py" with "id_flow=1".

Flow 1, as described in the configuration file "flows.conf", provides the following information:

```
1
TO component1 1
POST payload
TO component2 1
POST result1
TO component3 1
POST result1 result2
```

Figure 19 - Runtime Manager Flow 1

Expected Results: N1 executes correctly the flow 1 compared to configuration files. N1 receives the json message with "id_flow" and "payload" information, in particular "id_flow=1", it takes and executes the flow with the same id within the configuration file "flows.conf".

Step 2: Send the command of the execution flow "2" to RM1 running "test_mqtt_published.py" with "id_flow=2".

Flow 2, as described in the configuration file "flows.conf", provides the following information:

```
2
TO component1 2
GET
TO component2 1
POST result1 payload
```

Figure 20 - Runtime Manager Flow 2

Expected Results: N1 executes correctly the flow 2 compared to configuration files. N1 receives the json message with "id_flow" and "payload" information, in particular "id_flow=2", it takes and executes the flow with the same id within the configuration file "flows.conf".

Step 3: Send the command of the execution flow "3" to RM1 running "test_mqtt_published.py" with "id_flow=3".

Flow 3, as described in the configuration file "flows.conf", provides the following information:

```
3
TO component1 2
GET
TO component2 2
GET
```

Figure 21 - Runtime Manager Flow 3

Expected Results: N1 executes correctly the flow 3 compared to configuration files. N1 receives the json message with "id_flow" and "payload" information, in particular "id_flow=3", it takes and executes the flow with the same id within the configuration file "flows.conf".

Steps	

Step 1	Send the command of the execution flow "1" to RM1 running "test_mqtt_published.py" with "id_flow=1".
	`root@xilinx-zcu102-2021_2:~/N1/WP6T62-03-Runtime-Manager-main/test# python3 test_mqtt_publisher.py`
	`data published`
	`[rm_mqtt -> Receiver] - triggered by MQTT`
	`msg_s= {'id_flow': '1', 'payload': '0x03abcdefghil'}`

Step 2	Send the command of the execution flow "2" to RM1 running "test_mqtt_published.py" with "id_flow=2".
	`root@xilinx-zcu102-2021_2:~/N1/WP6T62-03-Runtime-Manager-main/test# python3 test_mqtt_publisher.py`
	`data published`
	`[rm_mqtt -> Receiver] - triggered by MQTT`
	`msg_s= {'id_flow': '2', 'payload': '0x03abcdefghil'}`

Step 3	Send the command of the execution flow "3" to RM1 running "test_mqtt_published.py" with "id_flow=3".
	`root@xilinx-zcu102-2021_2:~/N1/WP6T62-03-Runtime-Manager-main/test# python3 test_mqtt_publisher.py`
	`data published`
	`[rm_mqtt -> Receiver] - triggered by MQTT`
	`msg_s= {'id_flow': '3', 'payload': '0x03abcdefghil'}`

Table 41 – Steps for Validation Test T04_WP6T62-03

8.3.2.5 T05_WP6T62-03 - Task Scheduling on the remote node

Step 1: Create the condition that overload the Node 1

For the overload condition of the N1 node, it was executed some simple "while True" cycles on the python interpreter. This occupies resources at the node's processor. This procedure needs to be repeated until the processor is not overloaded (>85%). The procedure is described as follows:

- Open the command prompt of the N1 nodes and execute the following line:
  - python3 #to open the python interpreter
  - while True: #infinite cycle
    print(1)

As is possible to notice in the table (using the htop command), the overload condition is verified and the node is to be in an overloaded state.

Step 1: Send the command of the execution flow "1" to RM1 running "test_mqtt_published.py" with "id_flow=1".

Flow 1, as described in the configuration file "flows.conf", provides the following information:

```
1
TO component1 1
POST payload
TO component2 1
POST result1
TO component3 1
POST result1 result2
```

Figure 22 - Runtime Manager Flow 1

Expected Results: having generated the overload condition for the N1, RM1 will ask at Load Balancer instances which of the nodes can perform the workflow. The load balancer will respond with {"id_node": 2}. RM1 will direct the workflows to the N2 node as shown in the table. N2 executes correctly the flow 1 compared to configuration files. N2 receives the json message with "id_flow" and "payload" information, in particular "id_flow=1", it takes and executes the flow with the same id within the configuration file "flows.conf".

Step 2: Send the command of the execution flow "2" to RM1 running "test_mqtt_published.py" with "id_flow=2".

Flow 2, as described in the configuration file "flows.conf", provides the following information:

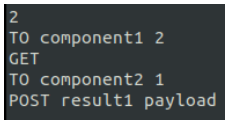


Figure 23- Runtime Manager Flow 2

Expected Results: having generated the overload condition for the N1, RM1 will ask at Load Balancer instances which of the nodes can perform the workflow. The load balancer will respond with {"id_node": 2}. RM1 will direct the workflows to the N2 node as shown in the table. N2 executes correctly the flow 2 compared to configuration files. N2 receives the json message with "id_flow" and "payload" information, in particular "id_flow=2", it takes and executes the flow with the same id within the configuration file "flows.conf".

Step 3: Send the command of the execution flow "3" to RM1 running "test_mqtt_published.py" with "id_flow=3".

Flow 3, as described in the configuration file "flows.conf", provides the following information:

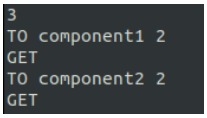


Figure 24 - Runtime Manager Flow 3

Expected Results: having generated the overload condition for the N1, RM1 will ask at Load Balancer instances which of the nodes can perform the workflow. The load balancer will respond with {"id_node": 2}. RM1 will direct the workflows to the N2 node as shown in the table. N2 executes correctly the flow 3 compared to configuration files. N2 receives the json message with "id_flow" and "payload" information, in particular "id_flow=3", it takes and executes the flow with the same id within the configuration file "flows.conf".

Steps	

Step 1	Create the condition that overload the Node 1
	`1 [||||||||||||||||||||||||||||||||||||95.9%]`
	`2 [||||||||||||||||||||||||||||||||||||100.0%]`
	`3 [||||||||||||||||||||||||||||||||||||96.7%]`
	`4 [||||||||||||||||||||||||||||||||||||100.0%]`

Step 2	Send the command of the execution flow “1” to RM1 running “test_mqtt_published.py” with “id_flow=1”.
	`root@xilinx-zcu102-2021_2:~/N1/WP6T62-03-Runtime-Manager-main/test# python3 test_mqtt_publisher.py`
	`data published`
	`[rm_mqtt -> Receiver] - triggered by MQTT`
	`msg_s= {'id_flow': '1', 'payload': '0x03abcdefghil'}`

Step 3	Send the command of the execution flow “2” to RM1 running “test_mqtt_published.py” with “id_flow=2”.
	`root@xilinx-zcu102-2021_2:~/N1/WP6T62-03-Runtime-Manager-main/test# python3 test_mqtt_publisher.py`
	`data published`
	`[rm_mqtt -> Receiver] - triggered by MQTT`
	`msg_s= {'id_flow': '2', 'payload': '0x03abcdefghil'}`

Step 4	Send the command of the execution flow “3” to RM1 running “test_mqtt_published.py” with “id_flow=3”.
	`root@xilinx-zcu102-2021_2:~/N1/WP6T62-03-Runtime-Manager-main/test# python3 test_mqtt_publisher.py`
	`data published`
	`[rm_mqtt -> Receiver] - triggered by MQTT`
	`msg_s= {'id_flow': '3', 'payload': '0x03abcdefghil'}`

Table 42 - Steps for Validation Test T05_WP6T62-03

8.3.3 Test environment setup

The Test environment in the Runtime Manager Validation Test is configured using three different nodes:

- Local Node with id = 1 is the Zynq UltraScale+ ZCU102 board having Quad-Core Arm Cortex-A53 processor, CPU frequency up to 1.5GHz and 4GB of RAM Memory;
- Remote Node with id = 2 is the Virtual Machine Computer having 2 Core Intel i7-8650U processor, CPU Frequency up to 1.90 GHz and 2 GB RAM Memory;
- Remote Node with id = 3 is the Virtual Machine Computer having the same characteristics of Remote Node with id = 2.

The nodes have been interconnected using an ethernet connection. A schematic is reported in Figure 18 just presented in previous section.

To better understand the nomenclature, we have defined the following legend:

- RM {1, 2, 3} = Runtime Manager on the node with id = 1, 2, 3;
- LB {1, 2, 3} = Load Balancer on the node with id = 1, 2, 3 related to RM {1, 2, 3}
- N {1, 2, 3} = Node with id = 1, 2, 3.

On each node were installed all the requirements specified on the GitHub repository of the Runtime Manager Component [https://github.com/project-fractal/WP6T62-03-Runtime-Manager.git].

After that, the configurations files of the Runtime Manager Component were modified as follows. In this way, a test environment was created within a local network.

The nodes N1, N2, N3 have been configured in the following way:

-N1:

RM1:

ip = "192.168.0.1"
ip api = "192.168.0.1"
port api = 7777
entrypoint api = "/startrm"
loadbalancer ip = "127.0.0.1"
loadbalancer port = 7776
loadbalancer endpoint = "LB/id_node"

LB1:

id = 1
ip = 192.168.0.1
ip api = 127.0.0.1
port_api = 7776
ip_broker = 192.168.0.2

-N2:

RM2:

ip = 192.168.0.2
ip api = "192.168.0.2"
port api = 8888
entrypoint = "/nodo2"
loadbalancer ip = "127.0.0.1"
loadbalancer port = 8886
loadbalancer endpoint = "LB/id_node"

LB2:

id = 2
ip = 192.168.0.2
ip_api = 127.0.0.1
port_api = 8887
ip_broker = "localhost"

-N3:

RM3:

ip = 192.168.0.2
ip api = "192.168.0.2"
port api = 9999
entrypoint = "/nodo3"
loadbalancer ip = "127.0.0.1"
loadbalancer port = 9998
loadbalancer endpoint = "LB/id_node"

LB3:

id = 3
ip = 192.168.0.2
ip_api = 127.0.0.1
port_api = 9998
ip_broker = "localhost"

As described in the github repository, to execute Runtime Manager has been used "rm_apy.py" and "rm_mqtt.py" scripts. For example, on the node N1:

- Runtime Manager listens on API REST

```
root@xilinx-zcu102-2021_2:~/N1/WP6T62-03-Runtime-Manager-main/runtime_manager# python3 rm_api.py
```

Figure 25 - Run "rm_api.py" script

- Runtime Manager listens on MQTT

```
root@xilinx-zcu102-2021_2:~/N1/WP6T62-03-Runtime-Manager-main/runtime_manager# python3 rm_mqtt.py
```

Figure 26 - Run "rm_mqtt.py" script

Furthermore, the "test_mqtt_publisher.py" script has been used to send a JSON message with the following field:

- "id_flow", describes the execution flow
- "payload", describes the data that needs to be exchanged between nodes.

Modifying the "id_flow" field with {1, 2, 3} is possible to execute different workflows.

To create the condition that overloads the nodes it is possible to execute a blocking function (ex: a few instances of a while True cycle) The results of this action is a processor overloaded.

8.3.4 Test execution

8.3.4.1 T01_WP6T62-03 - Testing interaction between nodes with local node overloaded

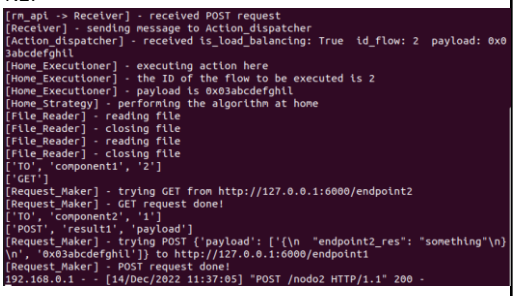
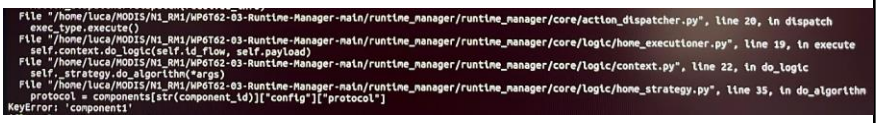
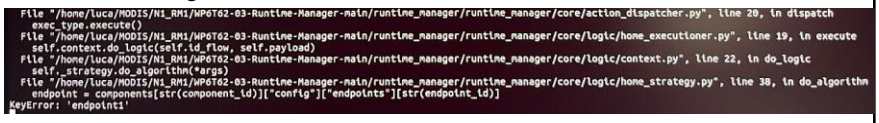
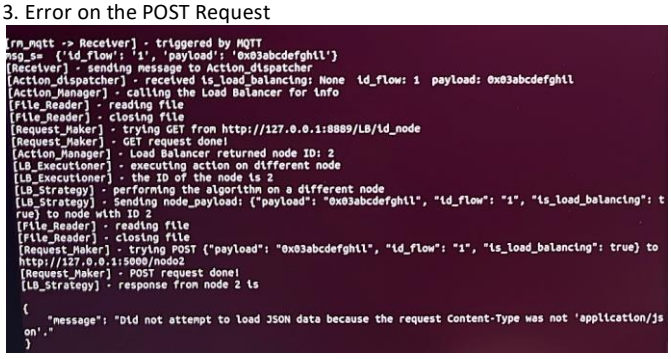
Results/Evidence	
N1:  	N2: 
Success criteria	
The interaction and data exchange between nodes have to be executed successfully	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-03-Runtime-Manager RM{1, 2, 3} = Runtime Manager on the node with id = 1, 2, 3 N{1, 2, 3} = Node with id = 1, 2, 3 N1 is defined by Zynq UltraScale+ ZCU102 board with IP=192.168.0.1 and PORT=7777 N2 is defined by a Virtual Machine on Computer with IP=192.168.0.2 and PORT=8888 N3 is defined by a Virtual Machine on Computer with IP=192.168.0.2 and PORT=9999
Test conditions	N1 is overloaded and it cannot perform other computation
Remarks	Three bugs were foud during validation test that have been reported and corrected by the developers. 1. Error on the configuration file "component.conf"  2. Error on the configuration file "comm.conf"  3. Error on the POST Request 
Test Result	
Passed	

Table 43 - Results of the test T01_WP6T62-03

8.3.4.2 T02_WP6T62-03 - Testing interaction in the local node when the local node can perform the computation

Results/Evidence	
N1:	
Success criteria	
The interaction and data exchange in the local nodes have to be executed successfully	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-03-Runtime-Manager
	RM{1, 2, 3} = Runtime Manager on the node with id = 1, 2, 3
	N{1, 2, 3} = Node with id = 1, 2, 3
	N1 is defined by Zynq UltraScale+ ZCU102 board with IP=192.168.0.1 and PORT=7777
	N2 is defined by a Virtual Machine on Computer with IP=192.168.0.2 and PORT=8888
	N3 is defined by a Virtual Machine on Computer with IP=192.168.0.2 and PORT=9999
Test conditions	N1 is not overloaded and it can perform any computation
Remarks	
Test Result	
Passed	

Table 44 - Results of the test T02_WP6T62-03

8.3.4.3 T03_WP6T62-03 - Testing interaction between nodes with Node 1 and Node 2 overloaded

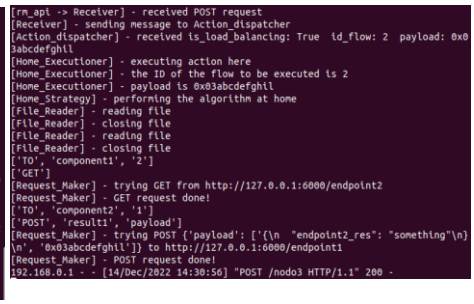
Results/Evidence	
N1:    N3: 
Success criteria	
The interaction and data exchange between nodes have to be executed successfully	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-03-Runtime-Manager
	RM{1, 2, 3} = Runtime Manager on the node with id = 1, 2, 3
	N{1, 2, 3} = Node with id = 1, 2, 3
	N1 is defined by Zynq UltraScale+ ZCU102 board with IP=192.168.0.1 and PORT=7777
	N2 is defined by a Virtual Machine on Computer with IP=192.168.0.2 and PORT=8888
	N3 is defined by a Virtual Machine on Computer with IP=192.168.0.2 and PORT=9999
Test conditions	N1 and N2 is overloaded and it cannot perform other computation
Remarks	
Test Result	
Passed	

Table 45 - Results of the test T03_WP6T62-03

8.3.4.4 T04_WP6T62-03 - Task Scheduling on the local node

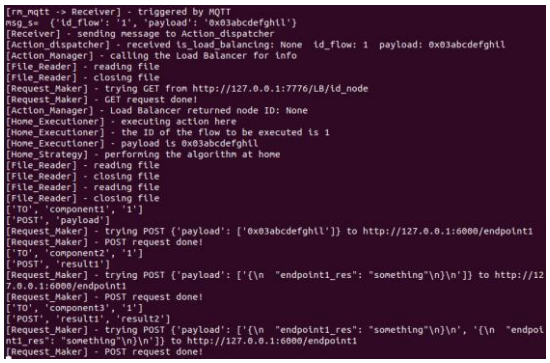
Results/Evidence

Result Step 1:

Flow1:

```
1
TO component1 1
POST payload
TO component2 1
POST result1
TO component3 1
POST result1 result2
```

N1:

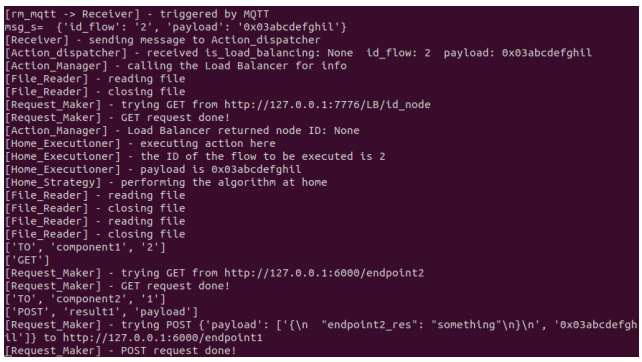


Result Step 2:

Flow2:

```
2
TO component1 2
GET
TO component2 1
POST result1 payload
```

N1:



Result Step 3:

Flow3:

```
3
TO component1 2
GET
TO component2 2
GET
```

N1:

Success criteria	
The execution of the task in local node have to be executed successfully	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-03-Runtime-Manager
	RM{1, 2, 3} = Runtime Manager on the node with id = 1, 2, 3
	N{1, 2, 3} = Node with id = 1, 2, 3
	N1 is defined by Zynq UltraScale+ ZCU102 board with IP=192.168.0.1 and PORT=7777
	N2 is defined by a Virtual Machine on Computer with IP=192.168.0.2 and PORT=8888
	N3 is defined by a Virtual Machine on Computer with IP=192.168.0.2 and PORT=9999
Test conditions	N1 is not overloaded and it can perform any computation
Remarks	
Test Result	
Passed	

Table 46 - Results of the test T04_WP6T62-03

8.3.4.5 T05_WP6T62-03 - Task Scheduling on the remote node

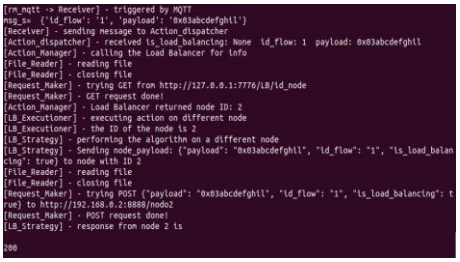
Results/Evidence

Result step 2:

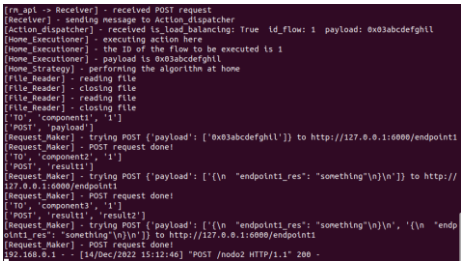
Flow1:

```
1
TO component1 1
POST payload
TO component2 1
POST result1
TO component3 1
POST result1 result2
```

N1:



N2:

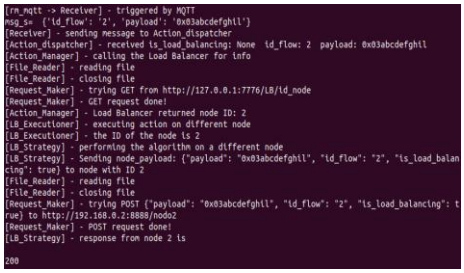


Result step 3:

Flow2:

```
2
TO component1 2
GET
TO component2 1
POST result1 payload
```

N1:



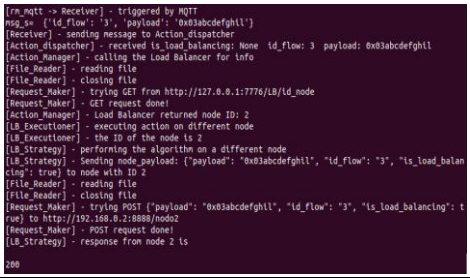
N2:

Result step 4:

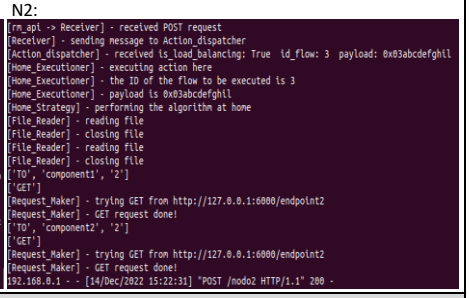
Flow2:

```
3
TO component1 2
GET
TO component2 2
GET
```

N1:



N2:



Success criteria	
The execution of the task and the exchange data with remote node have to be executed successfully	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-03-Runtime-Manager
	RM{1, 2, 3} = Runtime Manager on the node with id = 1, 2, 3
	N{1, 2, 3} = Node with id = 1, 2, 3
	N1 is defined by Zynq UltraScale+ ZCU102 board with IP=192.168.0.1 and PORT=7777
	N2 is defined by a Virtual Machine on Computer with IP=192.168.0.2 and PORT=8888
	N3 is defined by a Virtual Machine on Computer with IP=192.168.0.2 and PORT=9999
Test conditions	N1 is overloaded and it cannot perform other computation
Remarks	
Test Result	
Passed	

Table 47 - Results of the test T05_WP6T62-03

8.4 Data Ingestion

Data may come into the Fractal Edge Node in various forms, formats, file sizes, and from various sources. For this reason, a Fractal component able to deal with all these heterogeneous types of data is needed. Data ingestion is the process of collecting, importing, and processing raw data from various sources into a data storage or analysis system. This process is a crucial step in data management and is essential to prepare data for analysis and knowledge extraction. Data ingestion involves several steps, including data collection, transformation, and loading. In D6.2 different open-source robust and reliable tools have been identified to support the data ingestion task. Each of them can be used according to the specific needs, the requirements, and the operating system where the component is installed. The tools have been selected to support the different environment and architectures used in the FRACTAL node. In particular, tools have been selected for use in ARM64, RISCV64 and RISCV32 architectures. Since they are well consolidated and open-source tools, a broad documentation can be found online for each of them.

8.4.1 Test planification

8.4.1.1 Define the testing scope and identify the functionality that needs to be tested

In this section the tools proposed in D6.2 are reviewed to check that they can be installed and used according to the guidelines that can be found here https://github.com/project-fractal/WP6T62-01-data-ingestion.

The tests that have been identified consist of installing the components and validating that they are up and running and able to perform basic tasks. More detailed validation can be found in the documentation of the different tools.

T01_WP6T62-01_DI – Testing that Apache NiFi can be installed without any issue.

Validation test	
Test ID	T01_WP6T62-01_DI - Testing Apache NiFi
Test type	Functional-Installation
Test name	Installation of Apache NiFi
Date	13/01/2023
Tester's Name	Nicola Alchera (Rulex)
Test scope or objective	
The objective of this test is to validate that Apache Nifi can be installed without any issue.	

Table 48 - Validation Test T01_WPT62-01_DI

T02_WP6T62-01_DI – Testing that PySpark can be installed and configured without any issue.

Validation test	
Test ID	T02_WP6T62-01_DI
Test type	Functional
Test name	Installation and configuration of PySpark
Date	16/02/2023
Tester's Name	Nicola Alchera (Rulex)
Test scope or objective	
The objective of this test is to ensure that PySPark can be installed and configured without any issue.	

Table 49 - Validation Test T02_WPT62-01_DI

T03_WP6T62-01_DI – Testing that Faust can be installed and configured without any issue.

Validation test	
Test ID	T03_WP6T62-01_DI
Test type	Functional-Installation
Test name	Installation and configuration of Faust
Date	16/02/2023
Tester's Name	Nicola Alchera (Rulex)
Test scope or objective	
The objective of this test is to validate the Faust installation guidelines.	

Table 50 - Validation Test T03_WPT62-01_DI

T04_WP6T62-01_DI – Testing that RedNote can be installed and configured without any issue.

Validation test	
Test ID	T04_WP6T62-01_DI
Test type	Functional-Installation
Test name	Installation of RedNote
Date	16/02/2023
Tester's Name	Nicola Alchera (Rulex)
Test scope or objective	
The objective of this test is to validate the RedNote installation guidelines.	

Table 51 - Validation Test T04_WPT62-01_DI

T05_WP6T62-01_DI – Testing that the MQTT is working properly.

Validation test	
Test ID	T05_WP6T62-01_DI
Test type	Functional-Installation
Test name	test of MQTT Cloud Communications Component
Date	03/03/23
Tester's Name	Nicola Alchera (Rulex)
Test scope or objective	
The objective of this test is to validate the MQTT Cloud Communication guidelines.	

Table 52 - Validation Test T05_WPT62-01_DI

8.4.2 Test case development

8.4.2.1 T01_WP6T62-01_DI - Testing Apache NiFi

Steps	
Step 1	Prepare a node with Ubuntu and required dependencies (Python)
Step 2	Download files and checking prerequisites
Step 3	Configure and run ApacheNifi
Step 4	Login and Open Apache Nifi in browser

Table 53 - Steps for Validation Test T01_WP6T62-01_DI

8.4.2.2 T02_WP6T62-01_DI – Testing PySpark

Steps	
Step 1	Prepare a node with Ubuntu and the required dependencies (Python 3)
Step 2	Install pySpark and open it on python3

Table 54 - Steps for Validation Test T02_WP6T62-01_DI

8.4.2.3 T03_WP6T62-01_DI – Testing Faust

Steps	
Step 1	Prepare a node with Ubuntu and required dependencies (python 3)
Step 2	install Faust and run python3
Results/Evidence	

Table 55 - Steps for Validation Test T03_WP6T62-01_DI

8.4.2.4 T04_WP6T62-01_DI – Testing RedNote

Steps	
Step 1	Prepare a node with Ubuntu and required dependencies (python 3)
Step 2	Install Docker
Step 3	Run the docker
Step 4	Open the docker in browser

Table 56 - Steps for Validation Test T04_WP6T62-01_DI

8.4.2.5 T05_WP6T62-01_DI – Testing MQTT Broker

Steps	
Step 1	Prepare a node with Ubuntu and required dependencies (python 3)
Step 2	Set up a container with an MQTT broker
Step 3	Test the publish functionalities
Step 4	Test the subscribe functionalities

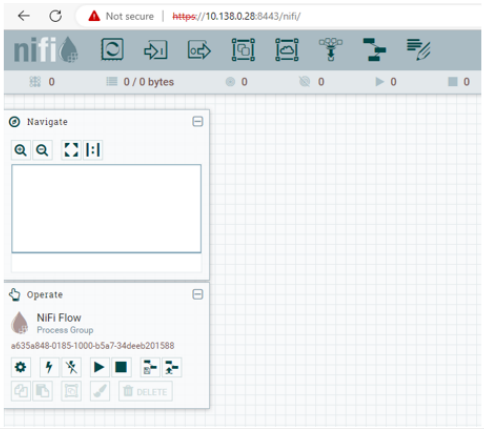
Table 57 - Steps for Validation Test T05_WP6T62-01_DI

8.4.3 Test environment setup

The steps to install and configure the component can be found in the FRACTAL project repository https://github.com/project-fractal/WP6T62-01-data-ingestion. According to the guidelines reported there, the tests have been performed on an Ubuntu 22.04 machine.

8.4.4 Test execution

8.4.4.1 T01_WP6T62-01_DI - Testing Apache NiFi

Results/Evidence
The Nifi software has been successfully opened and it is ready to use 
Success criteria
No error messages/All partial results are as expected

Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-01-data-ingestion
Test conditions	The tests have been performed on a node with Ubuntu 22.04.
Remarks	Running *./nifi.sh start* you might be warned that the Java home path is empty. This seems to be a warning. It can be fixed using the following command *"export JAVA_HOME="your_java_folder_path"* 
	You cannot download the tarball file from *https://nifi.apache.org/download.html* because the 1.18 and 1.19 release has the .zip binary files only. To download the .tar file you need to consider the archive *https://archive.apache.org/dist/nifi/* . The latest version with the .tar file available is the 1.16.3 and this is the one that has been used.
Test result	
Passed	

Table 58 - Results for test T01_WP6T62-01_DI

8.4.4.2 T02_WP6T62-01_DI - Testing PySpark

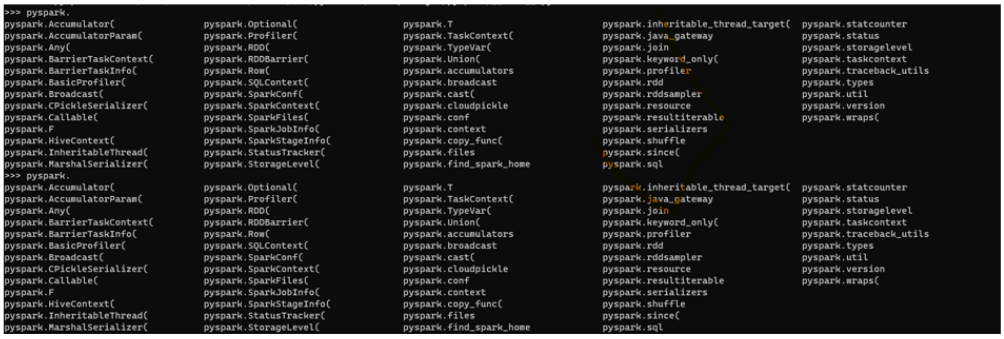
Results/Evidence	
pySpark has been correctly installed and it can be used as a Python3 package	
	
Success criteria	
No error messages	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-01-data-ingestion
Test conditions	The tests have been performed on a node with Ubuntu 22.04.
Remarks	No issues have been higlighted during the test
Test result	
Passed	

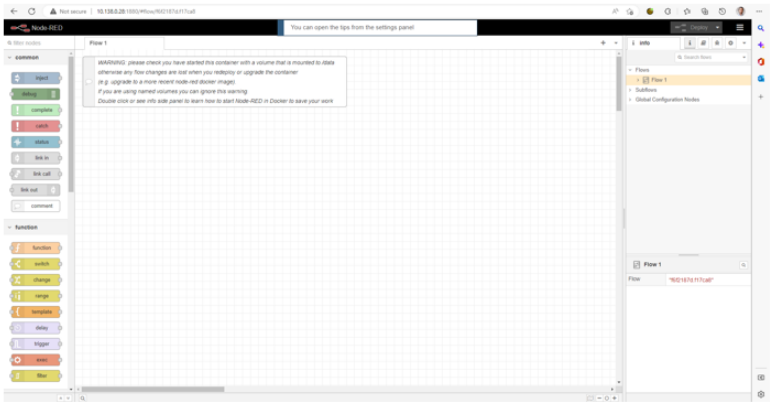
Table 59 - Results for test T02_WP6T62-01_DI

8.4.4.3 T03_WP6T62-01_DI - Testing Faust

Results/Evidence	
Faust has been correctly installed and it can be used as python3 library	
Success criteria	
No error messages/All partial results are as expected	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-01-data-ingestion
Test conditions	The tests have been performed on a node with Ubuntu 22.04.
Remarks	No issues have been highlighted during the test
Test result	
Passed	

Table 60 - Results for test T03_WP6T62-01_DI

8.4.4.4 T04_WP6T62-01_DI - Testing RedNote

Results/Evidence
Rednote has been successfully opened and it is ready to use 
Success criteria
RedNote has been installed without major errors

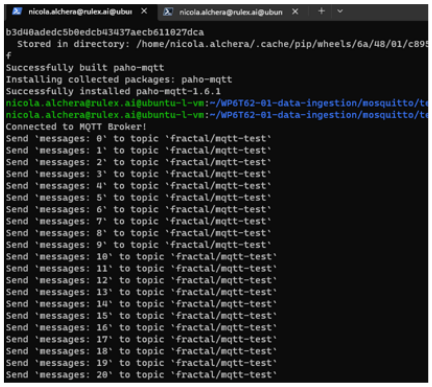
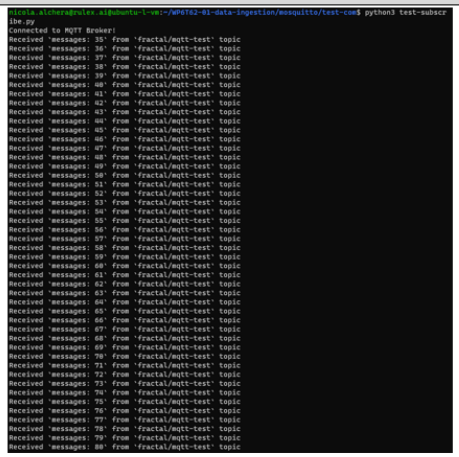
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-01-data-ingestion
Test conditions	The tests have been performed on a node with Ubuntu 22.04.

Remarks	Check you have started this container with a volume mounted on /data. I checked, and it seems the container is monted on /data, as you can see in the image below.
	Despite this, the workflow is not saved .
	"Mounts": [
	{
	"Type": "volume",
	"Name": "node_red_data",
	"Source": "/var/lib/docker/volumes/node_red_data/_data",
	"Destination": "/data",
	"Driver": "local",
	"Mode": "z",
	"RW": true,
	"Propagation": ""
	}
	],

Test result	
Passed	

Table 61 - Results for test T04_WP6T62-01_DI

8.4.4.5 T05_WP6T62-01_DI - Testing MQTT Broker

Results/Evidence
Both the publisher and subscriber script works correctly.    
Success criteria
No error messages/All partial results are as expected

Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-01-data-ingestion
Test conditions	The tests have been performed on a node with Ubuntu 22.04.
Remarks	The clone command "git clone https://github.com/project-fractal/WP6T62-01-data-ingestion.git" could not work properly. The user password authentication is no longer supported: a valid token is required for access. If you haven't any token, you have create a new one following the instructions you can fine here https://docs.github.com/en/get-started/getting-started-with-git/about-remote-repositories#cloning-with-https-urls The command "source run.sh" could give you an error. To avoid the error you shold give the permission through the following command "sudo chmod 777 /var/run/docker.sock" There is an error in guideline: guidlines says to test two times the same python script (test-publish.py) and not the "test-publish.py" and the "test-subscribe.py"
Test result	
Passed	

Table 62 - Results for test T05_WP6T62-01_DI

8.5 Federated Data Collection

Federated data collection is a process that enables organizations to collect and analyse data from multiple sources in a decentralized manner. Unlike traditional data collection methods, which involve bringing all data into a single central location for analysis, federated data collection allows organizations to analyze data in place, without moving it to a central location. In D6.2 different solutions, based on open-source tools, are proposed to handle the federated collection of the data coming from the Edge Nodes.

8.5.1 Test planification

In this section the tools proposed in D6.2 are reviewed to check that they can be installed and used according to the guidelines that can be found here https://github.com/project-fractal/WP6T62-02-federated_data_collection.

The tests that have been identified consist of installing the components and validating that they are up and running and able to perform basic tasks. More detailed validation can be found in the documentation of the different tools.

T01_WP6T62-02_FDC – Testing that CrateDB can be installed and used without any issue.

Validation test	
Test ID	T01_WP6T62-02_FDC
Test type	Functional-Installation
Test name	Installation of CrateDB
Date	16/01/2023
Tester's Name	Nicola Alchera (Rulex)
Test scope or objective	
The objective of this test is to validate the CrateDB installation guidelines.	

Table 63 - Validation Test T01_WPT62-02_FDC

T01_WP6T62-02_FDC – Testing that MongoDB can be installed and used without any issue.

Validation test	
Test ID	T02_WP6T62-02_FDC
Test type	Functional-Installation
Test name	Installation of MongoDB
Date	24/01/2023
Tester's Name	Nicola Alchera (Rulex)
Test scope or objective	
The objective of this test is to validate the MongoDB installation guidelines.	

Table 64 - Validation Test T02_WPT62-02_FDC

8.5.2 Test case development

8.5.2.1 T01_WP6T62-02_FDC - Testing CrateDB

Steps	
Step 1	Prepare a node with Ubuntu and required dependencies
Step 2	Install CrateDB on Ubuntu
Step 3	Run the docker and open CrateDB in a web browser
Step 4	Create and view a table using SQL into Crate DB

Table 65 - Steps for Validation Test T01_WP6T62-02_FDC

8.5.2.2 T02_WP6T62-02_FDC - Testing MongoDB

Steps	
Step 1	Prepare a node with Ubuntu and required dependencies
Step 2	Install Mongo on Ubuntu
Step 3	Run the docker

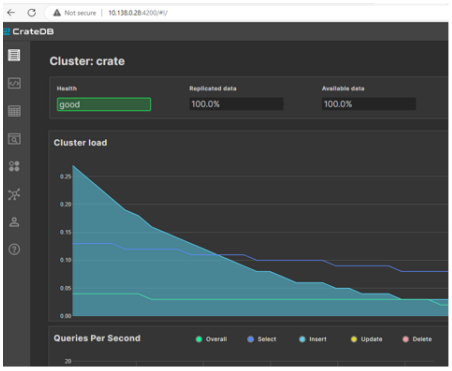
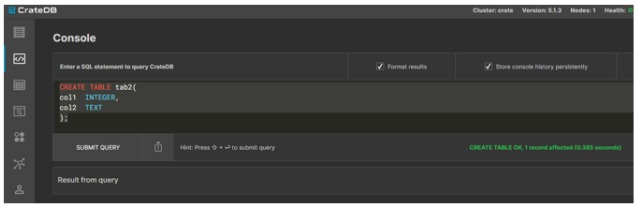
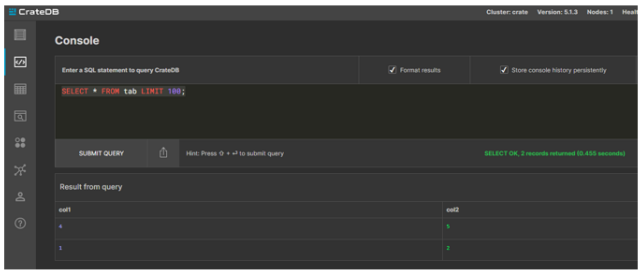
Table 66 - Steps for Validation Test T02_WP6T62-02_FDC

8.5.3 Test environment setup

The steps to install and configure the component can be found in the FRACTAL project repository https://github.com/project-fractal/WP6T62-02-Federated_Data_Collection. According to the guidelines reported there, the tests have been performed on a Ubuntu 22.04 machine following all the instruction reported in the guidelines repository.

8.5.4 Test execution

8.5.4.1 T01_WP6T62-02_FDC - Testing CrateDB

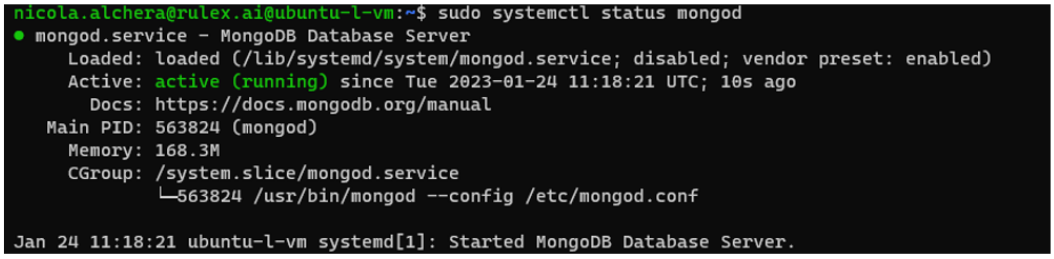
Results/Evidence
CrateDB has been successfully opened in browser and it is ready to use
  
Success criteria
Data can be created and viewed.

Success criteria	
Data can be created and viewed.	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-02-federated_data_collection
Test conditions	The tests have been performed on a node with Ubuntu 22.04.
Remarks	No issues were highlighted during the test
Test result	
Passed	

Table 67 - Results for Validation Test T01_WP6T62-02_FDC

8.5.4.2 T02_WP6T62-02_FDC - Testing MongoDB

Results/Evidence	

MongoDB has been successfully installed
 
`nicola.alchera@rulex.ai@ubuntu-l-vm:~$ sudo systemctl status mongod`
`● mongod.service - MongoDB Database Server`
`Loaded: loaded (/lib/systemd/system/mongod.service; disabled; vendor preset: enabled)`
`Active: active (running) since Tue 2023-01-24 11:18:21 UTC; 10s ago`
`Docs: https://docs.mongodb.org/manual`
`Main PID: 563824 (mongod)`
`Memory: 168.3M`
`CGroup: /system.slice/mongod.service`
`└─563824 /usr/bin/mongod --config /etc/mongod.conf`
`Jan 24 11:18:21 ubuntu-l-vm systemd[1]: Started MongoDB Database Server.`	

Success criteria	
Data can be created and viewed via MongoDB	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-02-federated_data_collection
Test conditions	The tests have been performed on a node with Ubuntu 22.04.
Remarks	No issues were highlighted during the test
Test result	
Passed	

Table 68 - Results for Validation Test T02_WP6T62-02_FDC

8.6 Low End Node

This component is based on the open-source RISC-V based PULP platform. The main features of the Low End Node are:

- Bridges between the IoT Hub and Kubernetes Cluster
- Shows the Nodes in CRD format in the Kubernetes Cluster
- Support any other MQTT broker-based connection

8.6.1 Test planification

8.6.1.1 Define the testing scope and identify the functionality that needs to be tested

The functionalities identified for the Low End Node component that need to be tested are related to how to connect and communicate with the cloud platform and the task scheduling.They were defined as follows:

1. The Low End Node has to be able to connect and communicate with the Cloud Platform;
2. The Low End Node has to be able to execute task scheduling;
3. The Low End Node has to be able to manage Ingestion and Storage.

For the first functioning three test cases are defined as shown below.

a) T01_WP6T62-06 – Testing the connection between the Device and the Cloud Platform.

Validation Test	
Test ID	T01_WP6T62-06
Test type	Functional
Test name	Testing the connection between the Device and the Cloud Platform
Date	19/01/2022
Tester's Name	Luca Visconti (Akkodis)
Test scope or objective	
The objective of this test is to validate the connection between Low End Node Device and Cloud Platform.	

Table 69 - Validation Test T01_WP6T62-06

b) T02_WP6T62-06 – Testing the communication from the Device to the Cloud.

Validation Test	
Test ID	T02_WP6T62-06
Test type	Functional
Test name	Testing the communication from Device to the Cloud
Date	19/01/2022
Tester's Name	Luca Visconti (Akkodis)
Test scope or objective	
The objective of this test is to validate communication between Low End Node Device and Cloud Platform.	

Table 70 - Validation Test T02_WP6T62-06

c) T03_WP6T62-06 – Testing the communication from the Cloud Platform to the Device.

Validation Test	
Test ID	T03_WP6T62-06
Test type	Functional
Test name	Testing the communication from the Cloud Platform to the Device
Date	19/01/2022
Tester's Name	Luca Visconti (Akkodis)
Test scope or objective	
The objective of this test is to validate the communication between Cloud Platform and Low End Node Device.	

Table 71 - Validation Test T03_WP6T62-06

For the second functioning, one test case is defined and shown below.

d) T04_WP6T62-06 – Testing the task scheduling running Nuttx on the Device.

Validation Test	
Test ID	T04_WP6T62-06
Test type	Functional
Test name	Testing the Tasks Scheduling running Nuttx on the Device
Date	19/01/2022
Tester's Name	Luca Visconti (Akkodis)
Test scope or objective	
The objective of this test is to validate the task scheduling on the Low End Node	

Table 72 - Validation Test T04_WP6T62-06

Nuttx supports normal posix socket and posix file systems, where data flow control can be handled with adequate drivers. In this way, various communication and storage mediums may be added. However, driver development is out of scope of validation activities.

8.6.2 Test case development

8.6.2.1 T01_WP6T62-06 - Testing the connection between the Device and the Cloud Platform

Step 1: Power up the device.

In this step, the device is connected to a power source with 5V to start it correctly.

Expected Results: having booted correctly the device is possible to notice that the red LED switch on as expected.

Step 2: Connect the device to the Internet.

In this step, the device is connected to an Access Point using SSID and password parameters. The figures below show that the device is connected to the network.

```
ubuntu@master:~$ kubectl get lowends.fractal-cluster.eu -n low-end-ctrl
NAME           AGE
low-end-2v2wf   3m24s
low-end-5r2gf   3m18s
low-end-llq4k   3m17s
low-end-lzjrf   3m21s
low-end-mm6pw   3m21s
low-end-w9p27   3m14s
low-end-z8srs   3m15s
ubuntu@master:~$
```

Figure 27 - The list of the provisioned devices as CRD in Kubernetes cluster with their ids

In the figure below is shown the list of IoT Devices on the Cloud side that are Connected or Disconnected. The connected device id is "fractal-node-8063XXX".

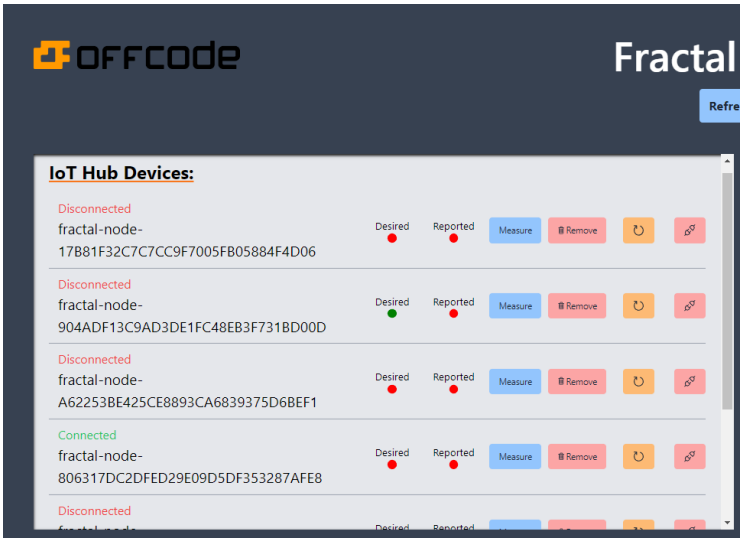


Figure 28 - The list of devices in IoT hub

The following figure shows through "kubectl" command the description of the Low End node in the Kubernetes cluster. The K8s generated id is low-end-5r2gf and the device id is "fractal-node-8063XX".

```
ubuntu@master:~$ kubectl describe lowends.fractal-cluster.eu -n low-end-ctrl low-end-5r2gf
Name:         low-end-5r2gf
Namespace:    low-end-ctrl
Labels:       <none>
Annotations:  kopf.zalando.org/last-handled-configuration:
                {"spec":{"connectionState":"Connected","desiredState":{"lastUpdated":"2023-02-20T07:33:00.9524174Z","state":0},"deviceI
d":"fractal-node-80...
API Version:  fractal-cluster.eu/v1
Kind:         LowEnd
Metadata:
  Creation Timestamp:  2023-02-20T07:39:24Z
  Finalizers:
    kopf.zalando.org/KopfFinalizerMarker
  Generate Name:  low-end-
  Generation:     2
  Managed Fields:
    API Version:  fractal-cluster.eu/v1
    Fields Type:  FieldsV1
    fieldsV1:
      f:metadata:
        f:annotations:
          .:
          f:kopf.zalando.org/last-handled-configuration:
        f:finalizers:
          .:
          v:"kopf.zalando.org/KopfFinalizerMarker":
    Manager:      kopf
    Operation:    Update
    Time:         2023-02-20T07:39:31Z
    API Version:  fractal-cluster.eu/v1
    Fields Type:  FieldsV1
    fieldsV1:
      f:metadata:
        f:generateName:
      f:spec:
        .:
        f:connectionState:
        f:desiredState:
          .:
          f:lastUpdated:
          f:state:
        f:deviceId:
        f:reportedState:
          .:
          f:lastUpdated:
          f:state:
    Manager:         pykube-ng
    Operation:       Update
    Time:            2023-02-20T07:39:31Z
  Resource Version:  555854366
  UID:               dc9e051f-8362-4065-8f4b-4d013a4c08d6
Spec:
  Connection State:  Connected
  Desired State:
    Last Updated:  2023-02-20T07:33:00.9524174Z
    State:         0
  Device Id:       fractal-node-806317DC2DFED29E09D5DF353287AFE8
  Reported State:
    Last Updated:  2023-02-20T07:39:29.6870527Z
    State:         0
```

Figure 29 - The description of the connected to the Kubernetes device

The figures below show the connected device and its green LED for the connection status.

Figure 30 - The connected device with its serial number: 8063

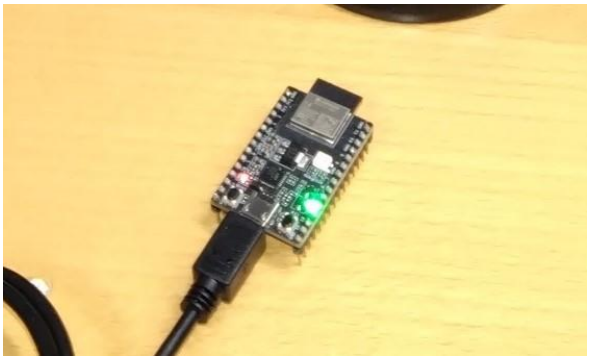


Figure 31 - Device connected with the green LED

Expected Results: during the connection period green LED is dim. When the device is connected the green LED is turned on. In this way, having connected the device to the internet is possible to notice that it is accepted and connected to the Cloud Platform.

Steps	
Step 1	Power up the device
Step 2	Connect the device to internet

Table 73 - Steps for Validation Test T01_WP6T62-06

8.6.2.2 T02_WP6T62-06 – Testing the communication from the Device to the Cloud

Step 1: Power up the device.

In this step, the device is connected to a power source with 5V to start it correctly.

Expected Results: having booted correctly the device is possible to notice that the red LED switch on as expected.

Step 2: Connect the device to the Internet.

In this step, the device is connected to an Access Point using SSID and password parameters. The figure below shows that the device is connected to the network.

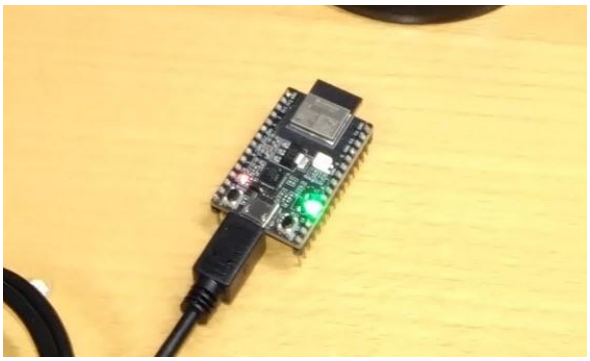


Figure 32 - Device connected with the green LED

Expected Results: During the connection period green LED is dim. When the device is connected the green LED is turned on. In this way, having connected the device to the Internet is possible to notice that it is accepted and connected to the Cloud Platform.

Step 3: Change the device status

There is a button on the device that changes its desired state. To perform this step, we simply press the button on the device. The figures below show that the device status is changed, also the status LEDs are changed from red to green.

Figure 33 - The desired state is OFF before pressing the button

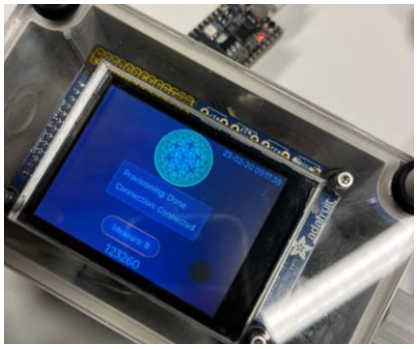


Figure 34 - The device before pressing button

The following figures are shown that the desired state and reported flags of the "fractal-node-7AEDXXXX" are changed, and its related LEDs green is switched on.

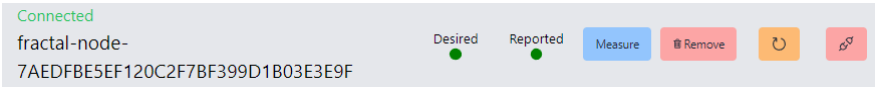


Figure 35 - The status of the device is changed in IoT hub after pressing button

Figure 36 - The device LED is green after pressing button

Expected Results: having connected the device on the Internet and having changed the device status is possible to notice that is reported status on the Cloud Platform side is as expected. In this way, it was tested the communication from the Device to the Cloud Platform.

Steps	
Step 1	Power up the device
Step 2	Connect the device to internet
Step 3	Change device status

Table 74 - Steps for Validation Test T02_WP6T62-06

8.6.2.3 T03_WP6T62-06 - Testing the communication from the Device to the Cloud

Step 1: Power up the device.

In this step, the device is connected to a power source with 5V to start it correctly.

Expected Results: having booted correctly the device is possible to notice that the red LED switch on as expected.

Step 2: Connect the device to the Internet.

In this step, the device is connected to an Access Point using SSID and password parameters. The figure below shows that the device is connected to the network.

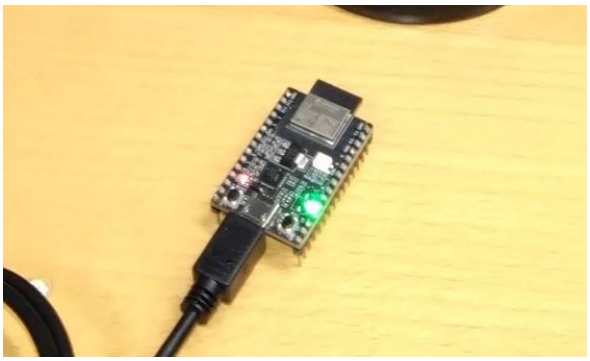


Figure 37 - Device connected with the green LED

Expected Results: During the connection period green LED is dim. When the device is connected the green LED is turned on. In this way, having connected the device to the Internet is possible to notice that it is accepted and connected to the Cloud Platform.

Step 3: Change the status in Kubernetes by the "patch" method.

In this step, there is a Kubernetes method called "kubectl patch" that updates the Kubernetes object or updates the running configuration. It uses "json merge" patch type passing json message with "desired state: 1" or "desired state: 0". The structure of the command is shown below.

- kubectl patch -n low-end-ctrl lowends.fractal-cluster.eu low-end-5r2gf --type merge --patch '{ "spec": { "desiredState": { "state": 1} }}'

```
ubuntu@master:~$ kubectl patch -n low-end-ctrl lowends.fractal-cluster.eu low-end-5r2gf --type merge --patch
 '{ "spec": { "desiredState": { "state": 1} }}'
lowend.fractal-cluster.eu/low-end-5r2gf patched
```

Figure 38 - Running the patch command using K8s CLI (Updating the time is ignored here)

In the following figures is shown that the desired state of the Low End node is changed from 0 to 1 and its related LED green is switched on.

```
Spec:
  Connection State:  Connected
  Desired State:
    Last Updated:  2023-02-20T07:33:00.9524174Z
    State:         1
  Device Id:       fractal-node-806317DC2DFED29E09D5DF353287AFE8
  Reported State:
    Last Updated:  2023-02-20T07:39:29.6870527Z
    State:         0
```

Figure 39 - Desired state of the device is changed to 1 in Kubernetes CRD (ON)

```
Normal  Logging  27s    kopf  Updating is processed: 1 succeeded; 0 failed.
Normal  Logging  27s    kopf  Handler 'update_fn' succeeded.
```

Figure 40 - Kubernetes log showing the invocation of update function

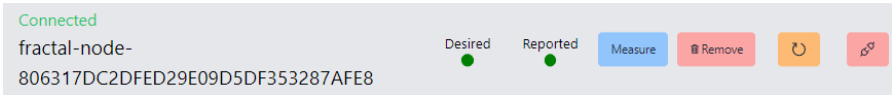


Figure 41 - The state in IoT hub has been switched to green

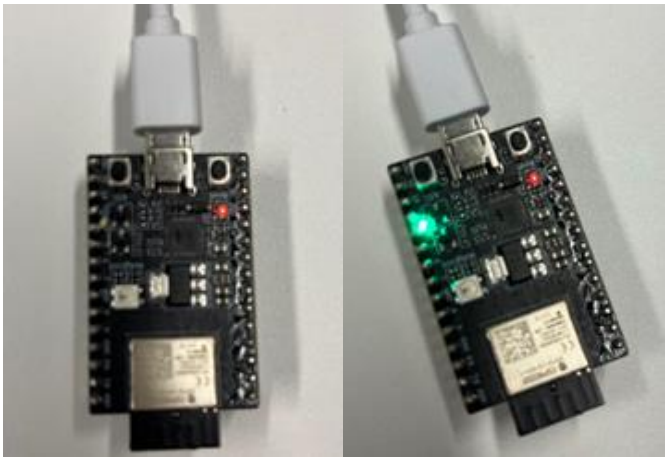


Figure 42 - The device before the patch command on the left, and after the patch command on the right

Expected Results: having connected the device on the Internet and having changed the device status on the Cloud Platform side is possible to notice that its reported status on the Device side is as expected. In this way, it was tested the communication from the Cloud Platform and the Device.

Steps	
Step 1	Power up the device
Step 2	Connect the device to internet

Step 3	Change status in Kubernetes by "patch" method
	`ubuntu@master:~$ kubectl patch -n low-end-ctrl lowends.fractal-cluster.eu low-end-5r2gf --type merge --patch '{ "spec": { "desiredState": { "state": 1} }}'`
	`lowend.fractal-cluster.eu/low-end-5r2gf patched`

Table 75 - Steps for Validation Test T03_WP6T62-06

8.6.2.4 T04_WP6T62-06 - Testing the tasks scheduling running Nuttx

Step 1: Power up the device.

In this step, the device is connected to a power source with 5V to start it correctly.

Expected Results: having booted correctly the device is possible to notice that the red LED switch on as expected.

Step 2: Connect the device by USB to the PC.

In this step, the device is connected to the PC via USB (that can also power the device).

At Linux PC with "dmesg" command is possible to show all the USB peripheral connected to the device:

```
$dmesg
[598658.253187] usb 3-2.1.4: new full-speed USB device number 16 using xhci_hcd
[598658.338481] usb 3-2.1.4: New USB device found, idVendor=10c4, idProduct=ea60,
bcdDevice=                                                              1.00
[598658.338492]  usb  3-2.1.4:  New  USB  device  strings:  Mfr=1,  Product=2,
SerialNumber=3
[598658.338497]  usb  3-2.1.4:  Product:  CP2102N  USB  to  UART  Bridge  Controller
[598658.338501]      usb      3-2.1.4:      Manufacturer:      Silicon      Labs
[598658.338504]  usb  3-2.1.4:  SerialNumber:  363aaf4f44a0eb11a2a8cbacdf749906
[598658.340065]     cp210x     3-2.1.4:1.0:     cp210x     converter     detected
[598658.345156] usb 3-2.1.4: cp210x converter now attached to ttyUSB1
```

Figure 43 - dmesg command

The device is at ttyUSB1 port.

Step 3: Open the terminal connection and run the command "ps".

After connecting the device to the PC we run the process status (ps) command. It is used to get information about currently running processes and their PIDs in your system.

Using the command below is possible to access at the serial console of the device.

```
$ minicom -D /dev/ttyUSB1

offcode>
```

Figure 44 - Command to open the terminal of device

The "ps" command and its related information are shown below:

```
offcode>
offcode> ps
 PID GROUP PRI POLICY   TYPE    NPX STATE    EVENT     SIGMASK   STACK COMMAND
   0     0   0 FIFO     Kthread N-- Ready              00000000 004048 Idle Task
   1     1 224 RR       Kthread --- Waiting  Semaphore 00000000 004016 hpwork 0x3fc844d8
   2     2 224 RR       Kthread --- Waiting  Semaphore 00000000 004016 hpwork 0x3fc844d8
   3     3 100 RR       Kthread --- Waiting  Semaphore 00000000 004016 lpwork 0x3fc844ec
   4     4 100 RR       Kthread --- Waiting  Semaphore 00000000 004016 lpwork 0x3fc844ec
   5     5 100 RR       Task    --- Running            00000000 001968 nsh_main
   6     6 223 RR       Kthread --- Waiting  Semaphore 00000000 001968 rt_timer
   7     7 253 RR       Kthread --- Waiting  MQ empty  00000000 006592 wifi
   8     8 100 RR       Task    --- Waiting  Signal    00000000 002944 NTP daemon
0.pool.ntp.org;1.pool.ntp.org;2.pool.ntp.org
  10    10 100 RR       Task    --- Waiting  Signal    00000000 016320 fractal
  11    10 100 RR       pthread --- Waiting  Signal    00000000 003024 pt-0x420347ee
0x3fcb0bd0
  12    10 100 RR       pthread --- Waiting  Signal    00000000 003024 pt-0x420347ee
0x3fcb6800
offcode>
```

Figure 45 - ps command

Expected Results: having connected the device to the PC and after running the "ps" command on the device terminal, the result is as expected. This command prints all the parallel tasks that are running on the device.

Steps	
Step 1	Power up the device
Step 2	Connect the device by usb to a pc
Step 3	Open terminal connection and run command "ps" `$ minicom -D /dev/ttyUSB1` offcode> offcode> ps

Table 76 - Steps for Validation Test T04_WP6T62-06

8.6.3 Test environment setup

First of all, we put in the config.yaml file the IoT hub credential that should be encoded to base64. After that, we defined a Wi-Fi configuration with parameters SSID and password for the local network.

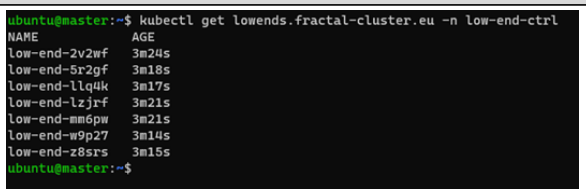
Other configuration and information of the Low End Node Device at the GitHub Repository, in the following link https://github.com/project-fractal/WP6T62-06-low-end-node-orchestrator.

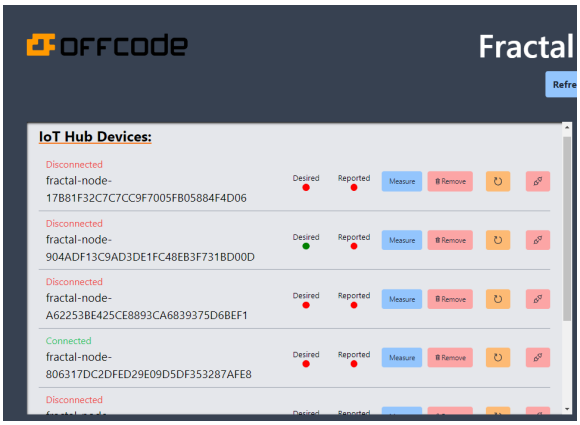
8.6.4 Test execution

8.6.4.1 T01_WP6T62-06 - Testing the connection between the Device and the Cloud Platform

Results/Evidence

Step 2:





Success criteria	
Device accepted and device status "connected"	
Test observations	
Test configuration	Device access point config: SSID and PW
Test conditions	Device connected to local wifi
Remarks	
Test result	
Passed	

Table 77 - Results of the test T01_WP6T62-06

8.6.4.2 T02_WP6T62-06 – Testing the communication from the Device to the Cloud Platform

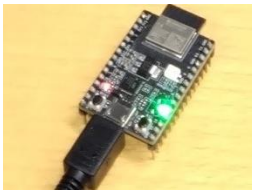
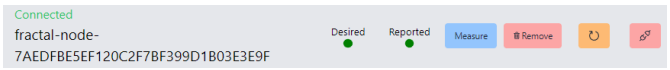
Results/Evidence	
Step 2:   Step 3:  	
Success criteria	
Reported status as expected	
Test observations	
Test configuration	Device access point config: SSID and PW
Test conditions	Device connected to local wifi
Remarks	
Test result	
Passed	

Table 78 - Results of the test T02_WP6T62-06

8.6.4.3 T03_WP6T62-06 - Testing the communication from the Cloud Platform to the Device

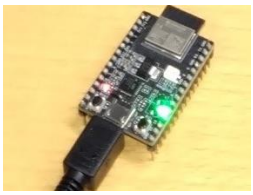
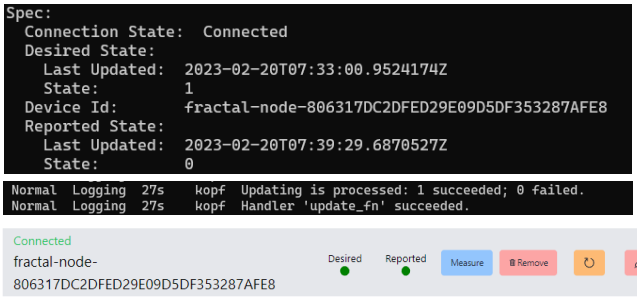
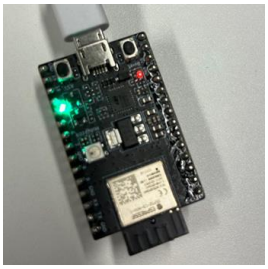
Results/Evidence	
Step 2:   Step 3:  	
Success criteria	
Device accepted and device status "connected"	
Test observations	
Test configuration	Device access point config: SSID and PW
Test conditions	Device connected to local wifi
Remarks	
Test result	
Passed	

Table 79 - Results of the test T03_WP6T62-06

8.6.4.4 T04_WP6T62-06 - Testing the tasks scheduling running Nuttx on the device

Results/Evidence	

Step 2:	`$dmesg`
	`[598658.253187] usb 3-2.1.4: new full-speed USB device number 16 using xhci_hcd`
	`[598658.338481] usb 3-2.1.4: New USB device found, idVendor=10c4, idProduct=ea60, bcdDevice= 1.00`
	`[598658.338492] usb 3-2.1.4: New USB device strings: Mfr=1, Product=2, SerialNumber=3`
	`[598658.338497] usb 3-2.1.4: Product: CP2102N USB to UART Bridge Controller`
	`[598658.338501] usb 3-2.1.4: Manufacturer: Silicon Labs`
	`[598658.338504] usb 3-2.1.4: SerialNumber: 363aaf4f44a0eb11a2a8cbacdf749906`
	`[598658.340065] cp210x 3-2.1.4:1.0: cp210x converter detected`
	`[598658.345156] usb 3-2.1.4: cp210x converter now attached to ttyUSB1`

Step 3:	`offcode>`
	`offcode> ps`
	`PID GROUP PRI POLICY TYPE NPX STATE EVENT SIGMASK STACK COMMAND`
	`0 0 0 FIFO Kthread N-- Ready 00000000 004048 Idle Task`
	`1 1 224 RR Kthread --- Waiting Semaphore 00000000 004016 hpwork 0x3fc844d8`
	`2 2 224 RR Kthread --- Waiting Semaphore 00000000 004016 hpwork 0x3fc844d8`
	`3 3 100 RR Kthread --- Waiting Semaphore 00000000 004016 lpwork 0x3fc844ec`
	`4 4 100 RR Kthread --- Waiting Semaphore 00000000 004016 lpwork 0x3fc844ec`
	`5 5 100 RR Task --- Running 00000000 001968 nsh_main`
	`6 6 223 RR Kthread --- Waiting Semaphore 00000000 001968 rt_timer`
	`7 7 253 RR Kthread --- Waiting MQ empty 00000000 006592 wifi`
	`8 8 100 RR Task --- Waiting Signal 00000000 002944 NTP daemon 0.pool.ntp.org;1.pool.ntp.org;2.pool.ntp.org`
	`10 10 100 RR Task --- Waiting Signal 00000000 016320 fractal`
	`11 10 100 RR pthread --- Waiting Signal 00000000 003024 pt-0x420347ee 0x3fcb0bd0`
	`12 10 100 RR pthread --- Waiting Signal 00000000 003024 pt-0x420347ee 0x3fcb6800`
	`offcode>`

Success criteria	
Print all parallel running task as expected	
Test observations	
Test configuration	GitHub repository: https://github.com/project-fractal/WP6T62-06-low-end-node-orchestrator Device access point config: SSID and PW
Test conditions	Device connected to local wifi
Remarks	
Test result	
Passed	

Table 80 - Results of the test T04_WP6T62-06

8.7 Hardware-level Edge Controller

The hardware Edge Controller controls the underlying hardware, such as communications and computations. The underlying hardware is based on a network-on-chip (NoC) multicore architecture that supports heterogeneous cores connected via NoC. The description of the underlying NoC-based multicore architecture is reported in contribution D4.4 (WP4). It is a time-triggered extension layer used in the VERSAL NoC to establish the temporal partitioning over the Chip.

In order to allow multiple nodes communication, within WP6, one of the NoC cores was devoted to function as a hardware gateway controller. It allows communication between on-chip and off-chip networks.

About the validation work in T6.3, we focused on the Network Gateway Interface that connect both on-chip and off-chip.

8.7.1 Test planification

8.7.1.1 Define the testing scope and identify the functionality that needs to be tested

In D6.2, several services are presented for this network hardware gateway architecture. Some of them are for future extensions, and they are not in this project scope.

This section will focus on following functionalities:

1. Message-Classification and Message-Scheduling Service;
2. Egress-Queuing and Ingress-Queuing Service;
3. Serialization Service.

For the first functioning, three test cases were defined, and they are shown below. Message scheduling refers to timing and port definition according to configuration, so we check messages from sending port to destination, seeing timing and message content correctness.

Message classification refers to the ability of the NGW to manage different types of messages. They are TT (Time Triggered), RATE (Rate Constraint), BE (Best Effort), but in the project scope, only TT messages are used.

a) T01_WP6T62-0X – Testing the Message-Classification and Message-Scheduling Services at different port according to scheduling configuration

Validation Test	
Test ID	T01_WP6T62-0X
Test type	Functional
Test name	Testing the Message-Classification and Message-Scheduling Services at different port according to scheduling configuration
Date	03/02/2022
Tester's Name	Luca Visconti (Akkodis)
Test scope or objective	
The objective of this test is to validate the message-classification and message-scheduling services sending 3 message at different port and according to the scheduling configuration.	

Figure 46 - Validation Test T02_WP6T62-0X

b) T02_WP6T62-0X – Testing the Message-Classification and Message-Scheduling Services at same port according to scheduling configuration

Validation Test	
Test ID	T02_WP6T62-0X
Test type	Functional
Test name	Testing the Message-Classification and Message-Scheduling Services at same port according to scheduling configuration
Date	03/02/2022
Tester's Name	Luca Visconti (Akkodis)
Test scope or objective	
The objective of this test is to validate the message-classification and message-scheduling services sending message at same port according to scheduling configuration	

Figure 47 - Validation Test T02_WP6T62-0X

c) T03_WP6T62-0X – Testing the Message-Classification and Message-Scheduling Services at same NI

Validation Test	
Test ID	T03_WP6T62-0X
Test type	Functional
Test name	Testing the Message-Classification and Message-Scheduling Services at same NI
Date	03/02/2022
Tester's Name	Luca Visconti (Akkodis)
Test scope or objective	
The objective of this test is to validate the message-classification and message-scheduling services sending message at same NI when have 2 port configured	

Figure 48 - Validation Test T03_WP6T62-0X

For the second functioning, one test case was defined, and it is shown below. Each NI has a queue for each port, and it can be configured as ingress or egress. Here we check the capacity of these queues using different lengths.

d) T04_WP6T62-0X – Testing the Ingress and Egress-queuing Services

Validation Test	
Test ID	T04_WP6T62-0X
Test type	Functional
Test name	Testing the Ingress and Egress-queuing Services
Date	03/02/2022
Tester's Name	Luca Visconti (Akkodis)
Test scope or objective	
The objective of this test is to validate the ingress-queuing and egress-queuing services using the max lenght of queue.	

Figure 49 - Validation Test T04_WP6T62-0X

For the third functioning one test case was defined and it is shown below:

e) T05_WP6T62-0X – Testing the serialization services

Serializer component is a subcomponent of NGW. As, it is not possible to see what happens inside the NGW, we defined according to the developers to run a simulation in order to proof the correct behaviors.

The Scope of the serializer is to take a message from the off-chip network and to provide a protocol conversion. It also works from on-chip to off-chip network in the same way.

Validation Test	
Test ID	T05_WP6T62-0X
Test type	Functional
Test name	Testing the serialization services
Date	03/02/2022
Tester's Name	Luca Visconti (Akkodis)
Test scope or objective	
The objective of this test is to validate the serialization service on NGW out port	

Figure 50 - Validation Test T05_WP6T62-0X

8.7.2 Test case development

8.7.2.1 T01_WP6T62-0X - *Testing the Message-Classification and Message-Scheduling Services at different port according to scheduling configuration*

Step 1: Define a set of messages to send (3 messages at 3 different ports according to the scheduling configuration TTCommSched.cfg of each NI).

- One message to NI0 on port 2

Port ID	Instant	Next	Instant (microsec)
2 [NI0-NI2]	956 (7)	0	28,483

Figure 51 - Information described in "ttcommsched.cfg" on NI0

- Two messages to NI1 respectively on port 2 and 3

Port ID	Instant	Next	Instant (microsec)
2 [NI1-NI0]	1092 (8)	1	32,552
3 [NI1-NI3]	2594 (19)	0	77,311

Figure 52 - Information described in "ttcommsched.cfg" on NI1

In the "ttcommsched.cfg" on the NI0 we have defined the following row:

- 00000000000000000000000003BC020

In the "ttcommsched.cfg" on the NI1 we have defined the following rows:

- 00000000000000000000000001444020
- 0000000000000000000000000A22030

The "hw.cfg" file is the same for all NIs, so it has the following rows:

- 04
  1
  1601010C0000000000000001
  1601010A0000000000000001
  1006
  1006
  1006
  1006

Step 2: Send messages to NI0 and to NI1.

In the architecture, we have four processing elements, and one processing element is working as a network gateway, and the rest of the processing elements are working as cores.

We have created this file in a C language that simply creates and send the message on a specific NI and Port. The first message is sent to NI0 on port 2, the second message is sent to NI1 on port 2 and the third message is sent to NI1 on port 3.

```
// send message from NI0 on port 2
for(int i=0;i<mesg_size;i++)
{
    Xil_Out32(XPAR_SKINOC_0_S00_AXI_BASEADDR+131072,i); // Write Message on NI0, port2
}
Xil_Out32(XPAR_SKINOC_0_S00_AXI_BASEADDR+163840,7);  // Terminate the Message

// send messages from NI1 on port 2

for(int i=0;i<mesg_size;i++)
{
    Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+131072,i); // Write Message on NI0, port2
}
Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+163840,7);  // Terminate the Message

// send messages from NI1 on port 3

for(int i=0;i<mesg_size;i++)
{
    Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+196608,i); // Write Message on NI0, port2
}
Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+229376,7);  // Terminate the Message
```

Figure 53 - Send messages scripts

Expected Results: In this case, having defined and sent messages to different NI, we expect to receive at the defined port the complete message. As depicted in the figures below is possible to see that the messages are arrived at the destination port and period according to the scheduling. The message content is also not corrupted.

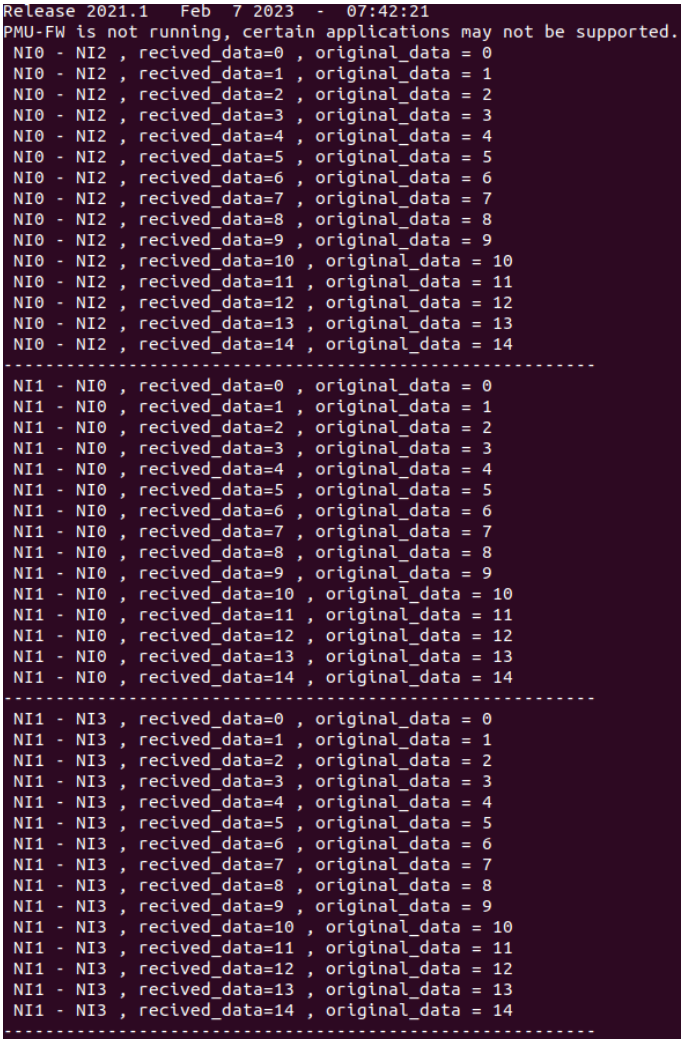


Figure 54 - Messages received

Furthermore, as shown in the figure below, the injection time and the different time between messages are correct according to configuration files.

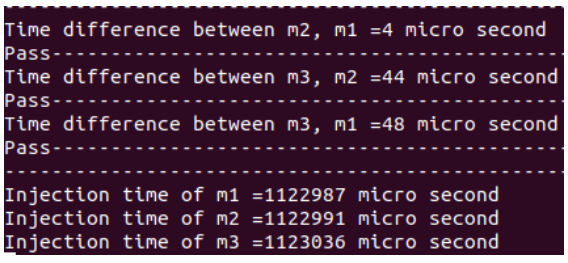


Figure 55 - Time different and Injection time

<table>
<tr><th colspan="2">Steps</th></tr>
<tr><td>Step 1</td><td>Define a set of messages to send (3 messages at 3 different port according to the scheduling configuration TTCommScheld.cfg of each NI)
- 1 message to NI0 on the port 2
<table>
<tr><th>Port ID</th><th>Instant</th><th>Next</th><th>Instant (microsec)</th></tr>
<tr><td>2 [NI0-NI2]</td><td>956 (7)</td><td>0</td><td>28,483</td></tr>
</table>
- 2 messages to NI1 on the port 2 and 3
<table>
<tr><th>Port ID</th><th>Instant</th><th>Next</th><th>Instant (microsec)</th></tr>
<tr><td>2 [NI1-NI0]</td><td>1092 (8)</td><td>1</td><td>32,552</td></tr>
<tr><td>3 [NI1-NI3]</td><td>2594 (19)</td><td>0</td><td>77,311</td></tr>
</table>
</td></tr>
<tr><td>Step 2</td><td>Send a message to NI0:
<pre>
// send message from NI0 on port 2
for(int i=0;i<mesg_size;i++)
{
    Xil_Out32(XPAR_SKINOC_0_S00_AXI_BASEADDR+131072,i); // Write Message on NI0, port2
}
Xil_Out32(XPAR_SKINOC_0_S00_AXI_BASEADDR+163840,7); // Terminate the Message
</pre>
Send a messages to NI1:
<pre>
// send messages from NI1 on port 2

for(int i=0;i<mesg_size;i++)
{
    Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+131072,i); // Write Message on NI0, port2
}
Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+163840,7); // Terminate the Message

// send messages from NI1 on port 3

for(int i=0;i<mesg_size;i++)
{
    Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+196608,i); // Write Message on NI0, port2
}
Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+229376,7); // Terminate the Message
</pre>
</td></tr>
</table>

Table 81 - Steps for Validation Test T01_WP6T62-0X

8.7.2.2 T02_WP6T62-0X - Testing the Message-Classification and Message-Scheduling Services at the same port according to scheduling configuration

Step 1: Define a set of messages to send (3 messages from the same port according to the scheduling configuration TTCommScheld.cfg of NI1)

- Three messages to NI1 on the port 2:

Port ID	Instant	Next	Instant (microsec)
2 [NI1-NI0]	1092 (8)	1	32,552
3 [NI1-NI3]	2594 (19)	0	77,311

Figure 56 - Information described in "ttcommsched.cfg" on NI1

In the "ttcommsched.cfg" on the NI1 we have defined the following rows:

- 00000000000000000000000001444020
- 0000000000000000000000000A22030

The "hw.cfg" file is the same for all NIs, so it has the following rows:

- 04
  1
  1601010C0000000000000001
  1601010A0000000000000001
  1006
  1006
  1006
  1006

Step 2: Send messages to NI1

We have created this file in a C language that simply create and send the message on a specific NI and Ports.

```
// send messages from NI1-NI0 using port2
for(int i=0;i<mesg_size;i++) // Message size = 5
{
    Xil_Out32(XPAR_SKINOC_0_S00_AXI_BASEADDR+131072,i); // Decimals (02) 000(
}
Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+163840,7); // terminate Message  00(

/*-------------------------------------------------------------------------*/

// send message from NI1-NI3
for(int i=0;i<mesg_size;i++) // Message size = 5
{
    Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+196608,i);
}
Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+229376,7); // terminate Message

/*-------------------------------------------------------------------------*/

// send messages from NI1-NI0.
for(int i=0;i<mesg_size;i++) // Message size = 5
{
    Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+131072,i);
}
Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+163840,7); // terminate Message

/*-------------------------------------------------------------------------*/

  // send message from NI1-NI3
  for(int i=0;i<mesg_size;i++) // Message size = 5
  {
    Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+196608,i);
  }
  Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+229376,7); // terminate Message

  /*-------------------------------------------------------------------------*/

   // send messages from NI1-NI0.
   for(int i=0;i<mesg_size;i++) // Message size = 5
   {
     Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+131072,i);
   }
     Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+163840,7); // terminate Message

  /*-------------------------------------------------------------------------*/

  // send message from NI1-NI3
  for(int i=0;i<mesg_size;i++) // Message size = 5
  {
    Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+196608,i);
  }
  Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+229376,7); // terminate Message
```

Figure 57 - Send messages scripts

Expected Results: In this case, having defined and sending messages to the same NI, we expect to receive at the defined port the complete message. In the figures below is possible to see that the messages arrived at the destination port and period according to the scheduling. The message content is also not corrupted.

```
--------------------------------------------------
NI1 - NI3 , recived_data=0 , original_data = 0
NI1 - NI3 , recived_data=1 , original_data = 1
NI1 - NI3 , recived_data=2 , original_data = 2
NI1 - NI3 , recived_data=3 , original_data = 3
NI1 - NI3 , recived_data=4 , original_data = 4
NI1 - NI3 , recived_data=5 , original_data = 5
NI1 - NI3 , recived_data=6 , original_data = 6
NI1 - NI3 , recived_data=7 , original_data = 7
NI1 - NI3 , recived_data=8 , original_data = 8
NI1 - NI3 , recived_data=9 , original_data = 9
NI1 - NI3 , recived_data=10 , original_data = 10
NI1 - NI3 , recived_data=11 , original_data = 11
NI1 - NI3 , recived_data=12 , original_data = 12
NI1 - NI3 , recived_data=13 , original_data = 13
NI1 - NI3 , recived_data=14 , original_data = 14
--------------------------------------------------
--------------------------------------------------
NI1 - NI3 , recived_data=0 , original_data = 0
NI1 - NI3 , recived_data=1 , original_data = 1
NI1 - NI3 , recived_data=2 , original_data = 2
NI1 - NI3 , recived_data=3 , original_data = 3
NI1 - NI3 , recived_data=4 , original_data = 4
NI1 - NI3 , recived_data=5 , original_data = 5
NI1 - NI3 , recived_data=6 , original_data = 6
NI1 - NI3 , recived_data=7 , original_data = 7
NI1 - NI3 , recived_data=8 , original_data = 8
NI1 - NI3 , recived_data=9 , original_data = 9
NI1 - NI3 , recived_data=10 , original_data = 10
NI1 - NI3 , recived_data=11 , original_data = 11
NI1 - NI3 , recived_data=12 , original_data = 12
NI1 - NI3 , recived_data=13 , original_data = 13
NI1 - NI3 , recived_data=14 , original_data = 14
--------------------------------------------------
--------------------------------------------------
NI1 - NI3 , recived_data=0 , original_data = 0
NI1 - NI3 , recived_data=1 , original_data = 1
NI1 - NI3 , recived_data=2 , original_data = 2
NI1 - NI3 , recived_data=3 , original_data = 3
NI1 - NI3 , recived_data=4 , original_data = 4
NI1 - NI3 , recived_data=5 , original_data = 5
NI1 - NI3 , recived_data=6 , original_data = 6
NI1 - NI3 , recived_data=7 , original_data = 7
NI1 - NI3 , recived_data=8 , original_data = 8
NI1 - NI3 , recived_data=9 , original_data = 9
NI1 - NI3 , recived_data=10 , original_data = 10
NI1 - NI3 , recived_data=11 , original_data = 11
NI1 - NI3 , recived_data=12 , original_data = 12
NI1 - NI3 , recived_data=13 , original_data = 13
NI1 - NI3 , recived_data=14 , original_data = 14
```

Figure 58 - Messages received

Furthermore, as shown in the figure below, the injection time and the period between messages are correct according to configuration files. Each message is received in a different period.

```
Time difference between two consecutive messages m2 and m1 = 122
Time difference between two consecutive messages m3 and m2 = 122
--------------------------------------------------
```

Figure 59 - Time different between messages

Steps	
Step 1	Define a set of messages to send (3 messages at 3 from the same port according to the scheduling configuration TTCommScheld.cfg of NI0) - 3 messages to NI0 on the port 2: Port ID | Instant | Next | Instant (microsec) 2 [NI0-NI2] | 956 (7) | 0 | 28,483
Step 2	Send messages to NI1:

```
// send messages from NI1-NI0 using port2
for(int i=0;i<mesg_size;i++) // Message size = 5
{
    Xil_Out32(XPAR_SKINOC_0_S00_AXI_BASEADDR+131072,i); // Decimals (02) 000(
}
Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+163840,7); // terminate Message  00(

/*-------------------------------------------------------------------------*/

// send message from NI1-NI3
for(int i=0;i<mesg_size;i++) // Message size = 5
{
    Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+196608,i);
}
Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+229376,7); // terminate Message

/*-------------------------------------------------------------------------*/

// send messages from NI1-NI0.
for(int i=0;i<mesg_size;i++) // Message size = 5
{
    Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+131072,i);
}
Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+163840,7); // terminate Message

/*-------------------------------------------------------------------------*/

  // send message from NI1-NI3
  for(int i=0;i<mesg_size;i++) // Message size = 5
  {
    Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+196608,i);
  }
  Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+229376,7); // terminate Message

  /*-------------------------------------------------------------------------*/

   // send messages from NI1-NI0.
   for(int i=0;i<mesg_size;i++) // Message size = 5
   {
     Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+131072,i);
   }
     Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+163840,7); // terminate Message

  /*-------------------------------------------------------------------------*/

  // send message from NI1-NI3
  for(int i=0;i<mesg_size;i++) // Message size = 5
  {
    Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+196608,i);
  }
  Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+229376,7); // terminate Message
```

Table 82 - Steps for Validation Test T02_WP6T62-0X

8.7.2.3 T03_WP6T62-0X - *Testing the Message-Classification and Message-Scheduling Services at the same NI*

Step 1: Define a set of messages to send (3 messages at 2 ports according to the scheduling configuration TTCommScheld.cfg of NI2)

- Two messages to NI2 on the port 1
- One message to NI2 on the port 2

Port ID	Instant	Next	Instant (microsec)
2 [NI2-NI0]	2048 (15)	1	61,035
3 [NI2-NI3]	2185 (16)	0	65,104

Figure 60 - Information described in "ttcommsched.cfg" on NI2

In the "ttcommsched.cfg" on the NI2 we have defined the following rows:

- 00000000000000000000000001800020
- 00000000000000000000000000889030

The "hw.cfg" file is the same for all NIs, so it has the following rows:

- 04
  1
  1601010C0000000000000001
  1601010A0000000000000001
  1006
  1006
  1006
  100

Step 2: Send messages to NI2:

We have created this file in a C language that simply creates and sends the message on a specific NI and Port. The first message is sent to NI2 on port 2, the second message is sent to NI2 on port 3 and the third message is sent to NI2 on port 2.

```
// send messages from NI2-NI0 using port2
for(int i=0;i<mesg_size;i++) // Message size = 5
{
    Xil_Out32(XPAR_SKINOC_0_S02_AXI_BASEADDR+131072,i);
}
Xil_Out32(XPAR_SKINOC_0_S02_AXI_BASEADDR+163840,7);

/*-------------------------------------------------------------------------*/

// send message from NI2-NI3
for(int i=0;i<mesg_size;i++) // Message size = 5
{
    Xil_Out32(XPAR_SKINOC_0_S02_AXI_BASEADDR+196608,i);
}
Xil_Out32(XPAR_SKINOC_0_S02_AXI_BASEADDR+229376,7); // terminate Message

/*-------------------------------------------------------------------------*/
// send messages from NI2-NI0 using port2
   for(int i=0;i<mesg_size;i++) // Message size = 5
   {
    Xil_Out32(XPAR_SKINOC_0_S02_AXI_BASEADDR+131072,i);
   }
   Xil_Out32(XPAR_SKINOC_0_S02_AXI_BASEADDR+163840,7);
/*-------------------------------------------------------------------------*/
```

Figure 61 - Send messages scripts

Expected Results: In this case, having defined and sending messages to the same NI we expect to receive at the defined port the complete message. In the figures below is possible to see that the messages arrived at destination port and period according to the scheduling. The message content is also not corrupted.

```
NI1 - NI0 , recived_data=0 , original_data = 0
NI1 - NI0 , recived_data=1 , original_data = 1
NI1 - NI0 , recived_data=2 , original_data = 2
NI1 - NI0 , recived_data=3 , original_data = 3
NI1 - NI0 , recived_data=4 , original_data = 4
NI1 - NI0 , recived_data=5 , original_data = 5
NI1 - NI0 , recived_data=6 , original_data = 6
NI1 - NI0 , recived_data=7 , original_data = 7
NI1 - NI0 , recived_data=8 , original_data = 8
NI1 - NI0 , recived_data=9 , original_data = 9
NI1 - NI0 , recived_data=10 , original_data = 10
NI1 - NI0 , recived_data=11 , original_data = 11
NI1 - NI0 , recived_data=12 , original_data = 12
NI1 - NI0 , recived_data=13 , original_data = 13
NI1 - NI0 , recived_data=14 , original_data = 14
------------------------------------------------
NI1 - NI3 , recived_data=0 , original_data = 0
NI1 - NI3 , recived_data=1 , original_data = 1
NI1 - NI3 , recived_data=2 , original_data = 2
NI1 - NI3 , recived_data=3 , original_data = 3
NI1 - NI3 , recived_data=4 , original_data = 4
NI1 - NI3 , recived_data=5 , original_data = 5
NI1 - NI3 , recived_data=6 , original_data = 6
NI1 - NI3 , recived_data=7 , original_data = 7
NI1 - NI3 , recived_data=8 , original_data = 8
NI1 - NI3 , recived_data=9 , original_data = 9
NI1 - NI3 , recived_data=10 , original_data = 10
NI1 - NI3 , recived_data=11 , original_data = 11
NI1 - NI3 , recived_data=12 , original_data = 12
NI1 - NI3 , recived_data=13 , original_data = 13
NI1 - NI3 , recived_data=14 , original_data = 14
------------------------------------------------
NI1 - NI0 , recived_data=0 , original_data = 0
NI1 - NI0 , recived_data=1 , original_data = 1
NI1 - NI0 , recived_data=2 , original_data = 2
NI1 - NI0 , recived_data=3 , original_data = 3
NI1 - NI0 , recived_data=4 , original_data = 4
NI1 - NI0 , recived_data=5 , original_data = 5
NI1 - NI0 , recived_data=6 , original_data = 6
NI1 - NI0 , recived_data=7 , original_data = 7
NI1 - NI0 , recived_data=8 , original_data = 8
NI1 - NI0 , recived_data=9 , original_data = 9
NI1 - NI0 , recived_data=10 , original_data = 10
NI1 - NI0 , recived_data=11 , original_data = 11
NI1 - NI0 , recived_data=12 , original_data = 12
NI1 - NI0 , recived_data=13 , original_data = 13
NI1 - NI0 , recived_data=14 , original_data = 14
------------------------------------------------
```

Figure 62 - Messages received

Furthermore, as shown in the figure below, the period between messages is correct according to the configuration files. Two messages are received in the same period and one message is received in the next period.

```
Time difference between m2, m1 = 4
Time difference between m3, m1 = 122
```

Figure 63 - Time different between messages

<table>
<tr><th colspan="2">Steps</th></tr>
<tr><td>Step 1</td><td>Define a set of messages to send (3 messages at 2 ports according to the scheduling configuration TTCommScheld.cfg of NI2)
- 2 messages to NI2 on the port 1
- 1 message to NI2 on the port 2
<table>
<tr><th>Port ID</th><th>Instant</th><th>Next</th><th>Instant (microsec)</th></tr>
<tr><td>2 [NI2-NI0]</td><td>2048 (15)</td><td>1</td><td>61,035</td></tr>
<tr><td>3 [NI2-NI3]</td><td>2185 (16)</td><td>0</td><td>65,104</td></tr>
</table></td></tr>
<tr><td>Step 2</td><td>Send messages to NI2:
<pre>
// send messages from NI2-NI0 using port2
for(int i=0;i<mesg_size;i++) // Message size = 5
{
    Xil_Out32(XPAR_SKINOC_0_S02_AXI_BASEADDR+131072,i);
}
Xil_Out32(XPAR_SKINOC_0_S02_AXI_BASEADDR+163840,7);

/*---*/

// send message from NI2-NI3
for(int i=0;i<mesg_size;i++) // Message size = 5
{
    Xil_Out32(XPAR_SKINOC_0_S02_AXI_BASEADDR+196608,i);
}
Xil_Out32(XPAR_SKINOC_0_S02_AXI_BASEADDR+229376,7); // terminate Message

/*---*/
// send messages from NI2-NI0 using port2
   for(int i=0;i<mesg_size;i++) // Message size = 5
   {
    Xil_Out32(XPAR_SKINOC_0_S02_AXI_BASEADDR+131072,i);
   }
   Xil_Out32(XPAR_SKINOC_0_S02_AXI_BASEADDR+163840,7);
/*---*/
</pre></td></tr>
</table>

Table 83 - Steps for Validation Test T03_WP6T62-0X

8.7.2.4 T04_WP6T62-0X - *Testing the Ingress and Egress-Queuing Services*

We have defined the same configuration of Validation Test T01. In the "ttcommsched.cfg" on the NI0 we have defined the following row:

- 00000000000000000000000003BC020

And in the "hw.cfg" file is the same for all NIs, so it has the following rows:

- 04
  1
  1601010C0000000000000001
  1601010A0000000000000001
  1006
  1006
  1006
  1006

We defined a set of messages to send, in particular, three messages from the same port according to the scheduling configuration TTCommScheld.cfg of NI0 with a queue length equal to 16:

- Message 1 size: 8-bit, length 10;

- Message 2 size: 16-bit, length 16;
- Message 3 size: 32-bit, length 50.

Step 1: Send a message with length less than queue length.

We have created this file in a C language that simply creates and sends the message to NI0 on port 2. The message has a length of 10. In the configuration file "port.cfg" is defined a queue length equal to 16.

```
for(int i=0;i<mesg_size1;i++)
{
    Xil_Out32(XPAR_SKINOC_0_S00_AXI_BASEADDR+131072,message1+i);
}
Xil_Out32(XPAR_SKINOC_0_S00_AXI_BASEADDR+163840,7);
```

Figure 64 - Send message

Expected Results: In this case, having defined and sent a message with length 10 we expect to receive at the defined port the complete message. In the figures below is possible to see that the message has arrived at the destination port. The message content is also not corrupted.

```
----------------------------------------
Message id = 0, recived_data=3 , original_data = 3
Message id = 1, recived_data=4 , original_data = 4
Message id = 2, recived_data=5 , original_data = 5
Message id = 3, recived_data=6 , original_data = 6
Message id = 4, recived_data=7 , original_data = 7
Message id = 5, recived_data=8 , original_data = 8
Message id = 6, recived_data=9 , original_data = 9
Message id = 7, recived_data=10 , original_data = 10
Message id = 8, recived_data=11 , original_data = 11
Message id = 9, recived_data=12 , original_data = 12
Time stamp=37196732
----------------------------------------**----------
```

Figure 65 - Message received

Step 2: Send a message with length equal than queue length

We have created this file in a C language that simply creates and sends the message to NI0 on port 2. The message has a length of 16. In the configuration file "ort.cfg" is defined a queue length equal to 16.

```
for(int i=0;i<mesg_size2;i++)
{
 Xil_Out32(XPAR_SKINOC_0_S00_AXI_BASEADDR+131072,message2+i);
}
Xil_Out32(XPAR_SKINOC_0_S00_AXI_BASEADDR+163840,7);
```

Figure 66 - Send message

Expected Results: In this case, having defined and sent a message with length 16 we expect to receive at the defined port the complete message. In the figures below is possible to see that the message has arrived at the destination port. The message content is also not corrupted.

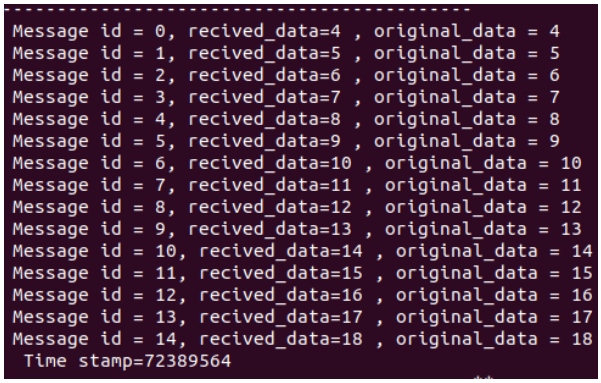


Figure 67 - Message received

Step 3: Send a message with length longer than queue length

We have created this file in a C language that simply creates and sends the message to NI0 on port 2. The message has a length of 32. In the configuration file "port.cfg" is defined a queue length equal to 16.

```
for(int i=0;i<mesg_size3;i++)
{
  Xil_Out32(XPAR_SKINOC_0_S00_AXI_BASEADDR+131072,message3+i);
}
Xil_Out32(XPAR_SKINOC_0_S00_AXI_BASEADDR+163840,7);
```

Figure 68 - Send message

Expected Results: In this case, having defined and sent a message with length 32, we expect to receive at the defined port the only first 16 rows and to have a queue overflow. In the figures below is possible to see that the message has arrived at the destination port, but the message content is corrupted after the 16 rows.

```
----------------------------------------
Message id = 0, recived_data=5 , original_data = 5
Message id = 1, recived_data=6 , original_data = 6
Message id = 2, recived_data=7 , original_data = 7
Message id = 3, recived_data=8 , original_data = 8
Message id = 4, recived_data=9 , original_data = 9
Message id = 5, recived_data=10 , original_data = 10
Message id = 6, recived_data=11 , original_data = 11
Message id = 7, recived_data=12 , original_data = 12
Message id = 8, recived_data=13 , original_data = 13
Message id = 9, recived_data=14 , original_data = 14
Message id = 10, recived_data=15 , original_data = 15
Message id = 11, recived_data=16 , original_data = 16
Message id = 12, recived_data=17 , original_data = 17
Message id = 13, recived_data=18 , original_data = 18
Message id = 14, recived_data=19 , original_data = 19
Message id = 15, recived_data=20 , original_data = 20
Message id = 16, recived_data=20 , original_data = 21
Message id = 17, recived_data=20 , original_data = 22
Message id = 18, recived_data=20 , original_data = 23
Message id = 19, recived_data=20 , original_data = 24
Message id = 20, recived_data=20 , original_data = 25
Message id = 21, recived_data=20 , original_data = 26
Message id = 22, recived_data=20 , original_data = 27
Message id = 23, recived_data=20 , original_data = 28
Message id = 24, recived_data=20 , original_data = 29
Message id = 25, recived_data=20 , original_data = 30
Message id = 26, recived_data=20 , original_data = 31
Message id = 27, recived_data=20 , original_data = 32
Message id = 28, recived_data=20 , original_data = 33
Message id = 29, recived_data=20 , original_data = 34
Message id = 30, recived_data=20 , original_data = 35
Message id = 31, recived_data=20 , original_data = 36
Message id = 32, recived_data=20 , original_data = 37
Message id = 33, recived_data=20 , original_data = 38
Message id = 34, recived_data=20 , original_data = 39
Message id = 35, recived_data=20 , original_data = 40
Message id = 36, recived_data=20 , original_data = 41
Message id = 37, recived_data=20 , original_data = 42
Message id = 38, recived_data=20 , original_data = 43
Message id = 39, recived_data=20 , original_data = 44
Message id = 40, recived_data=20 , original_data = 45
Message id = 41, recived_data=20 , original_data = 46
Message id = 42, recived_data=20 , original_data = 47
Message id = 43, recived_data=20 , original_data = 48
Message id = 44, recived_data=20 , original_data = 49
Message id = 45, recived_data=20 , original_data = 50
Message id = 46, recived_data=20 , original_data = 51
Message id = 47, recived_data=20 , original_data = 52
Message id = 48, recived_data=20 , original_data = 53
Message id = 49, recived_data=20 , original_data = 54
Time stamp=20
```

Figure 69 - Messages received

Steps	

Step 1	Send a message to NI0 on port 1 with a length less than the queue length
	`for(int i=0;i<mesg_size1;i++)`
	`{`
	`Xil_Out32(XPAR_SKINOC_0_S00_AXI_BASEADDR+131072,message1+i);`
	`}`
	`Xil_Out32(XPAR_SKINOC_0_S00_AXI_BASEADDR+163840,7);`

Step 2	Send a message to NI0 on port 1 with a length equal to the queue length
	`for(int i=0;i<mesg_size2;i++)`
	`{`
	`Xil_Out32(XPAR_SKINOC_0_S00_AXI_BASEADDR+131072,message2+i);`
	`}`
	`Xil_Out32(XPAR_SKINOC_0_S00_AXI_BASEADDR+163840,7);`

Step 3	Send a message to NI0 on port 1 with a length bigger than the queue length
	`for(int i=0;i<mesg_size3;i++)`
	`{`
	`Xil_Out32(XPAR_SKINOC_0_S00_AXI_BASEADDR+131072,message3+i);`
	`}`
	`Xil_Out32(XPAR_SKINOC_0_S00_AXI_BASEADDR+163840,7);`

Table 84 - Steps for Validation Test T04_WP6T62-0X

8.7.2.5 T05_WP6T62-0X - *Testing the Serialization service*

Step 1: Define a set of messages to send (1 message at 1 port according to the scheduling configuration TTCommScheld.cfg of NI3)

- One message to NI3 on port 2

Port ID	Instant	Next	Instant (microsec)
2[NI3-NI2]	1502 (11)	0	44,759

Figure 70 - Information described in "ttcommsched.cfg" on NI3

In the "ttcommsched.cfg" on the NI3 we have defined the following rows:

- 00000000000000000000000005DE020

The "hw.cfg" file is the same for all NIs, so it has the following rows:

- 04
  1
  1601010C0000000000000001
  1601010A0000000000000001
  1006
  1006
  1006
  1006

Step 2: Send a message to NI3 on port 2.

We have created this file in a C language that simply creates and sends the message to NI3 on port 2.

```
// send message from NI3 , port2
 INIT_AXI_TXN=1;

 PORT_ID_WR =2;
 INPUT_WDATA=0;
   #48
    INIT_AXI_TXN=0;
for(integer j=0;j<16;j=j+1)//f
    begin
    #8;
    INPUT_WDATA =j;
    end
```

Figure 71 - Send message

Expected Results: In this case, we expect to see the serialized messages in the NI sender, after messages pass through the serialize component.

In this way, we check the message from the NI sender on port 2. We can see the message on the "buffer_inst" field, it is a queue with a 005 "msglen".

Name	Value
clk	1
reset_n	1
enq	0
din[31:0]	0000001c
full	0
deq	0
dout[31:0]	UUUUUUUU
msglen[9:0]	005
empty	0
buffer_inst[0:511][31:0]	00000000,00000001,00000002,00000003,00000004,UUUUUUUU,UUUUUUUU,UUU
wrpntr[9:0]	005
rdpntr[9:0]	000
full_loc	0
empty_loc	0
nqd[9:0]	005
AddrWidth	9
WordWidth	32
C_MAX_NUM_ELEMENTS[9:0]	200

Figure 72 - Message on NI sender

In order to see the message in a waveform we wrote it in memory. We can see the message before serialization and refer to the clock.

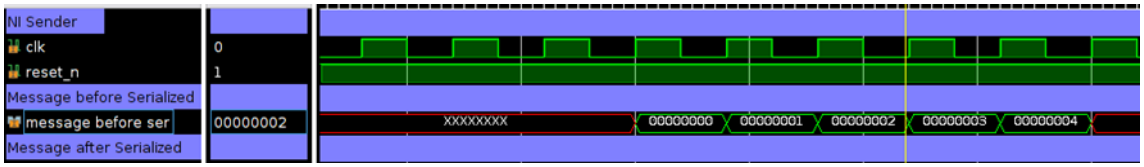


Figure 73 - Message before serialization

Here we see the serialize packetized message and added all information needed to forward the message to the NoC (head filt that contains the message path, timestamp, destination port, etc.)

Figure 74 - Message after serialization

Steps	
Step 1	Define a set of messages to send (1 message at 1 port according to the scheduling configuration TTCommScheld.cfg of NI3) - One message to NI3 on port 2 Port ID | Instant | Next | Instant (microsec) 2[NI3-NI2] | 1502 (11) | 0 | 44,759

Step 2	Send message at one port
	`// send message from NI3 , port2`
	`INIT_AXI_TXN=1;`
	`PORT_ID_WR =2;`
	`INPUT_WDATA=0;`
	`#48`
	`INIT_AXI_TXN=0;`
	`for(integer j=0;j<16;j=j+1)//f`
	`begin`
	`#8;`
	`INPUT_WDATA =j;`
	`end`

Table 85 - Steps for Validation Test T05_WP6T62-0X

8.7.3 Test environment setup

The Hardware Edge Controller component is mainly configured through the following configuration and scheduling files:

- "hw.cfg", contains the overall information about the NoC;
- "port.cfg" contains the port configuration for each NI;
- "ttcommsched.mem", contains the time-triggered communication schedule.

The "hw.cfg" represents the configuration of the whole NoC and serves as the basis for the instantiation of the NIs and their internal building blocks. The file is composed as described in the following figure:

Name	Range	Description
NR_TILES	1 (“01”) to 255 (“FF”)	Number of tiles on the chip
NR_PORTS	0 (“00”) to 255 (“FF”)	Number of ports at each NI
TIMELY_BLOCK	0 or 1	0 if timely block is deactivated and 1 if activated
PHASE_WIDTH	1 (“01”) to 63 (“3F”)	Width of the phase slice in the time format
NR_PERIODS	1 (“01”) to 63 (“3F”)	Number of supported periods
PERIOD_DELTA	1 (“01”) to 63 (“3F”)	Bit distance between period bits of two consecutive periods
MSB_PERIODBIT	1 (“01”) to 63 (“3F”)	Positional index of the bit of the greatest period in the global time base (greater than the MACROTICK_BIT
PERIOD_ENABLE	“0000000000000000” “FFFFFFFFFFFFFFFFFF” to to 255 (“FF”)	Hex representation of bit-wise period enable; 1 for activation.

Figure 75 – “hw.cfg” file

The “port.cfg” contains the configuration parameters of the ports for each NI. All configuration parameters are represented by hexadecimal digits, which are generated based on the parameters. The file is composed as described in the following figure:

Name	bits	location	Hex digits	Values	Description
Reserved	3 bits	127-125	1	----	
Enable	1 bit	124		Enable: "1" Disable: "0"	Shows whether the port is activated or not
Type	2 bits	123-122	1	TT: "00" RC1: "01" RC2: "10" BE: "11"	Represents the type of the port
Direction	1 bit	121		OUT: "0" IN: "1"	Represents the direction of the port
Semantics	1 bit	120		State: "0" Event: "1"	Represents the semantics of the port
Buffer Size	12 bits	119-108	3	0 to 4095	The size of the buffer in words
Queue length	12 bits	107-96	3	0 to 4095	This fields is valid for event ports. in case of state ports, this field should be 1
Cluster ID	8 bits	95-88	2	0 to 255	
Node ID	8 bits	87-80	2	0 to 255	Represent the Physical address of destination
Tile ID	8 bits	79-72	2	0 to 255	
Port ID	8 bits	71-64	2	0 to 255	
MINT	64 bits	63-0	8	Global Time format	In case of an RC port, this field represent the value for MINT

Figure 76 – “hw.cfg” file

The “ttcommsched.cfg” is conceptually made up of a circular linked list. Each list is associated with a period. The information included in this file is shown below:

- “next”, this field is the next pointer. It points to the next entry of the circular linked list, and it has a width of 5 bits.
- “instant”, this field denotes the phase of the injection of the message which is located at the given port (in PortId). It has a width of 8 bits.
- “PortId”, this field represents the ID of the port, whose message is injected into the NoC at the time given by Instant. It has a width of 4 bits.

In particular, in those configuration files we have defined:

- period of the clock (122us)
- Clock granularity (28,483ns)
- Injection time for messages (Instant)

We also installed ATTNoC on the FPGA board from Xilinx. The configuration has been applied with the Vitis Unified Development Environment. The simulation and visualization of the off-chip network was driven with the Vivado Design Suite.

All the configuration and information of the Hardware Edge Control at the GitHub Repository, in the following link https://github.com/project-fractal/WP6T62-HW-Edge-Controller.

8.7.4 Test execution

8.7.4.1 T01_WP6T62-0X - *Testing the Message-Classification and Message-Scheduling Services at different port*

Results/Evidence

Step 2:

```
Release 2021.1   Feb  7 2023  -  07:42:21
PMU-FW is not running, certain applications may not be supported.
 NI0 - NI2 , recived_data=0 , original_data = 0
 NI0 - NI2 , recived_data=1 , original_data = 1
 NI0 - NI2 , recived_data=2 , original_data = 2
 NI0 - NI2 , recived_data=3 , original_data = 3
 NI0 - NI2 , recived_data=4 , original_data = 4
 NI0 - NI2 , recived_data=5 , original_data = 5
 NI0 - NI2 , recived_data=6 , original_data = 6
 NI0 - NI2 , recived_data=7 , original_data = 7
 NI0 - NI2 , recived_data=8 , original_data = 8
 NI0 - NI2 , recived_data=9 , original_data = 9
 NI0 - NI2 , recived_data=10 , original_data = 10
 NI0 - NI2 , recived_data=11 , original_data = 11
 NI0 - NI2 , recived_data=12 , original_data = 12
 NI0 - NI2 , recived_data=13 , original_data = 13
 NI0 - NI2 , recived_data=14 , original_data = 14
-------------------------------------------------------
 NI1 - NI0 , recived_data=0 , original_data = 0
 NI1 - NI0 , recived_data=1 , original_data = 1
 NI1 - NI0 , recived_data=2 , original_data = 2
 NI1 - NI0 , recived_data=3 , original_data = 3
 NI1 - NI0 , recived_data=4 , original_data = 4
 NI1 - NI0 , recived_data=5 , original_data = 5
 NI1 - NI0 , recived_data=6 , original_data = 6
 NI1 - NI0 , recived_data=7 , original_data = 7
 NI1 - NI0 , recived_data=8 , original_data = 8
 NI1 - NI0 , recived_data=9 , original_data = 9
 NI1 - NI0 , recived_data=10 , original_data = 10
 NI1 - NI0 , recived_data=11 , original_data = 11
 NI1 - NI0 , recived_data=12 , original_data = 12
 NI1 - NI0 , recived_data=13 , original_data = 13
 NI1 - NI0 , recived_data=14 , original_data = 14
-------------------------------------------------------
 NI1 - NI3 , recived_data=0 , original_data = 0
 NI1 - NI3 , recived_data=1 , original_data = 1
 NI1 - NI3 , recived_data=2 , original_data = 2
 NI1 - NI3 , recived_data=3 , original_data = 3
 NI1 - NI3 , recived_data=4 , original_data = 4
 NI1 - NI3 , recived_data=5 , original_data = 5
 NI1 - NI3 , recived_data=6 , original_data = 6
 NI1 - NI3 , recived_data=7 , original_data = 7
 NI1 - NI3 , recived_data=8 , original_data = 8
 NI1 - NI3 , recived_data=9 , original_data = 9
 NI1 - NI3 , recived_data=10 , original_data = 10
 NI1 - NI3 , recived_data=11 , original_data = 11
 NI1 - NI3 , recived_data=12 , original_data = 12
 NI1 - NI3 , recived_data=13 , original_data = 13
 NI1 - NI3 , recived_data=14 , original_data = 14
-------------------------------------------------------
```

```
-------------------------------------------------
Time difference between m2, m1 =4 micro second
Pass-------------------------------------------
Time difference between m3, m2 =44 micro second
Pass-------------------------------------------
Time difference between m3, m1 =48 micro second
Pass-------------------------------------------
-------------------------------------------------
Injection time of m1 =1122987 micro second
Injection time of m2 =1122991 micro second
Injection time of m3 =1123036 micro second
```

Success criteria

Check:
- destination port as per configuration "port.cfg"
- message content (not corrupted)
- period as per configuration "hw.cfg"
- Is it possible to check the “instant” (Time Unit)?

Test observations

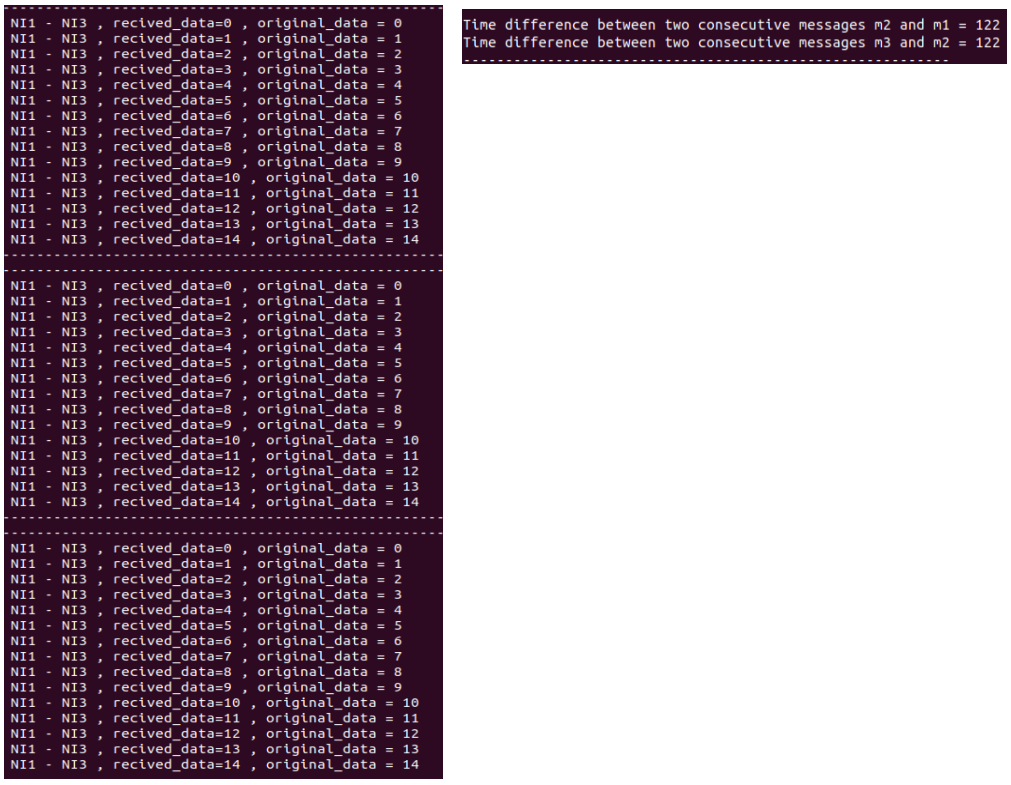
Test configuration	Config: "ttcommsched.cfg", "port.cfg", "hw.cfg"
Test conditions	Vitis for FPGA configuration, ATTNoC running on the board, all configuration files at github link https://github.com/project-fractal/WP6T62-HW-Edge-Controller
Remarks	

Test result

Passed

Table 86 - Results of the test T01_WP6T62-0X

8.7.4.2 T02_WP6T62-0X - *Testing the Message-Classification and Message-Scheduling Services at the same port*

Results/Evidence
Step 2: 
Success criteria
Check: - destination port (and NI) as "port.cfg" - the message content (not corrupted) - period (we expect three different period)
Test observations

Test configuration	Config: "ttcommsched.cfg", "port.cfg", "hw.cfg"
Test conditions	Vitis for FPGA configuration, ATTNoC running on the board, all configuration files at github link https://github.com/project-fractal/WP6T62-HW-Edge-Controller
Remarks	

Test result
Passed

Table 87 - Results of the test T02_WP6T62-0X

8.7.4.3 T03_WP6T62-0X - *Testing the Message-Classification and Message-Scheduling Services at the same NI*

Results/Evidence

Step 2:

```
NI1 - NI0 , recived_data=0 , original_data = 0
NI1 - NI0 , recived_data=1 , original_data = 1
NI1 - NI0 , recived_data=2 , original_data = 2
NI1 - NI0 , recived_data=3 , original_data = 3
NI1 - NI0 , recived_data=4 , original_data = 4
NI1 - NI0 , recived_data=5 , original_data = 5
NI1 - NI0 , recived_data=6 , original_data = 6
NI1 - NI0 , recived_data=7 , original_data = 7
NI1 - NI0 , recived_data=8 , original_data = 8
NI1 - NI0 , recived_data=9 , original_data = 9
NI1 - NI0 , recived_data=10 , original_data = 10
NI1 - NI0 , recived_data=11 , original_data = 11
NI1 - NI0 , recived_data=12 , original_data = 12
NI1 - NI0 , recived_data=13 , original_data = 13
NI1 - NI0 , recived_data=14 , original_data = 14
------------------------------------------------
NI1 - NI3 , recived_data=0 , original_data = 0
NI1 - NI3 , recived_data=1 , original_data = 1
NI1 - NI3 , recived_data=2 , original_data = 2
NI1 - NI3 , recived_data=3 , original_data = 3
NI1 - NI3 , recived_data=4 , original_data = 4
NI1 - NI3 , recived_data=5 , original_data = 5
NI1 - NI3 , recived_data=6 , original_data = 6
NI1 - NI3 , recived_data=7 , original_data = 7
NI1 - NI3 , recived_data=8 , original_data = 8
NI1 - NI3 , recived_data=9 , original_data = 9
NI1 - NI3 , recived_data=10 , original_data = 10
NI1 - NI3 , recived_data=11 , original_data = 11
NI1 - NI3 , recived_data=12 , original_data = 12
NI1 - NI3 , recived_data=13 , original_data = 13
NI1 - NI3 , recived_data=14 , original_data = 14
------------------------------------------------
NI1 - NI0 , recived_data=0 , original_data = 0
NI1 - NI0 , recived_data=1 , original_data = 1
NI1 - NI0 , recived_data=2 , original_data = 2
NI1 - NI0 , recived_data=3 , original_data = 3
NI1 - NI0 , recived_data=4 , original_data = 4
NI1 - NI0 , recived_data=5 , original_data = 5
NI1 - NI0 , recived_data=6 , original_data = 6
NI1 - NI0 , recived_data=7 , original_data = 7
NI1 - NI0 , recived_data=8 , original_data = 8
NI1 - NI0 , recived_data=9 , original_data = 9
NI1 - NI0 , recived_data=10 , original_data = 10
NI1 - NI0 , recived_data=11 , original_data = 11
NI1 - NI0 , recived_data=12 , original_data = 12
NI1 - NI0 , recived_data=13 , original_data = 13
NI1 - NI0 , recived_data=14 , original_data = 14
------------------------------------------------
```

```
Time difference between m2, m1 = 4
Time difference between m3, m1 = 122
```

Success criteria

Check:
- destination port (and NI) as "port.cfg"
- the message content (not corrupted)
- period ?? (Two different periods)

Test observations

Test configuration	Config: "ttcommsched.cfg", "port.cfg", "hw.cfg"
Test conditions	Vitis for FPGA configuration, ATTNoC running on the board, all configuration files at github link https://github.com/project-fractal/WP6T62-HW-Edge-Controller
Remarks	

Test result

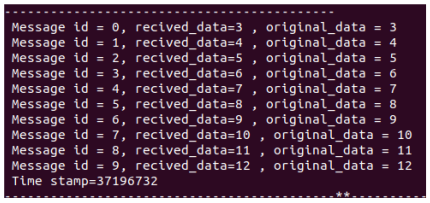
Passed

Table 88 - Results of the test T03_WP6T62-0X

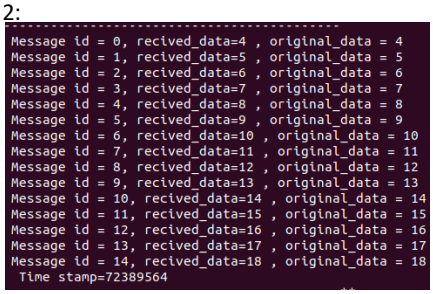
8.7.4.4 T04_WP6T62-0X - *Testing the Ingress and Egress-Queuing Services*

Results/Evidence

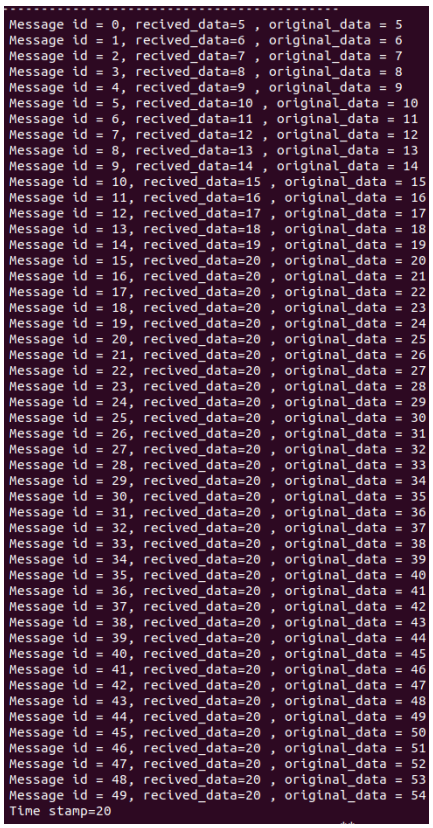
Step 1:



Step 2:



Step 3:



Success criteria

Check:
- to receive all messages on destination NI
- all messages on sending NI

Test observations

Test configuration	Config: "ttcommsched.cfg", "port.cfg", "hw.cfg"
Test conditions	Vitis for FPGA configuration, ATTNoC running on the board, all configuration files at github link https://github.com/project-fractal/WP6T62-HW-Edge-Controller, max queue length for a port = 16
Remarks	

Test result

Passed

Table 89 - Results of the test T04_WP6T62-0X

8.7.4.5 T05_WP6T62-0X - *Testing the Serialization service*

In this case, as introduced in test planification session we defined to perform a simulation. We used Vivado to see the waveform in subcomponents referred to clock signal, so we can observe the behaviour with a time reference.

The scope is to compare the message before and after the serializer in order to validate the serializer subcomponent that is in charge of the serialization service as described in D6.2.

Serializer subcomponent is very important in order to have at the end, a complete message packet to send through the network, from on-chip to off-chip network and vice versa.

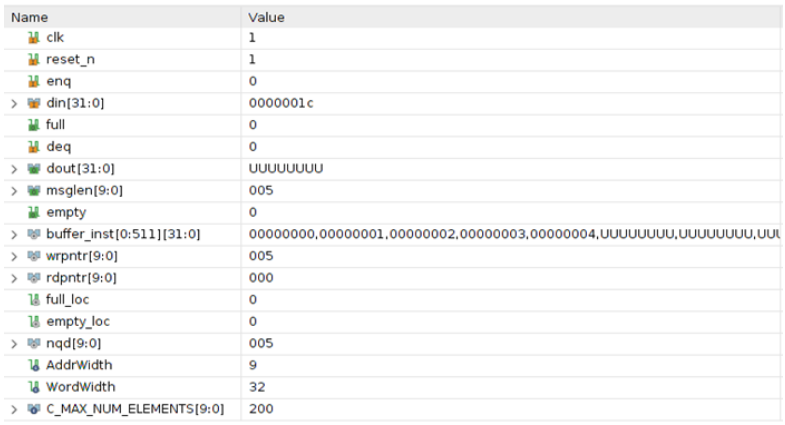
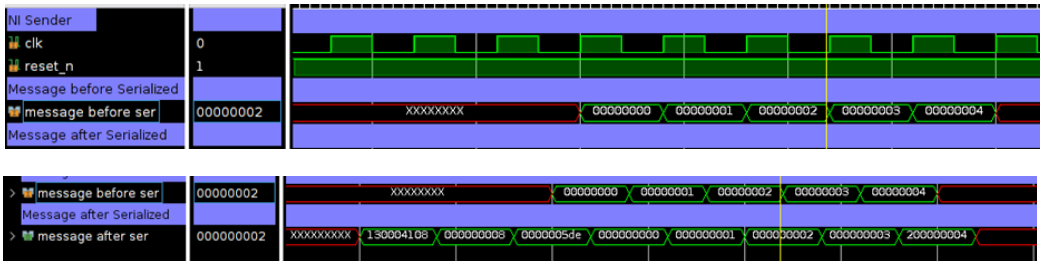
Results/Evidence	
Step 2:  	
Success criteria	
Check: - the message from the NI sender on the port 2 - the serialized message (flits in ordered)	
Test observations	
Test configuration	Config: "ttcommsched.cfg", "port.cfg", "hw.cfg"
Test conditions	Vitis for FPGA configuration, ATTNoC running on the board, all configuration files at github link https://github.com/project-fractal/WP6T62-HW-Edge-Controller
Remarks	
Test result	
Passed	

Table 90 - Results of the test T05_WP6T62-0X

9 Conclusions

In conclusion, this deliverable is focused on identifying the functionalities of the FRACTAL Edge Node Architecture implemented in task T6.1 and task T6.2, and on defining, implementing, and documenting the results of the validation tests that have been performed to validate the FRACTAL Edge Node proper operation at a component level.

Validation was carried out starting with the microservices regarding to connectivity, then the orchestration components, the runtime manager component, the Data ingestion component and Federated data collection component for the High-end nodes and Mid-range nodes. For the Low End Node there were a series of tests to validate the following functionalities:

- connection and communication with the Cloud Platform
- execution of task scheduling
- management of ingestion and storage

Finally, the Hardware-level Edge Controller was also validated, focused on the Network Gateway Interface that connects both on chip and off chip networks.

Some of the tests were performed on real HW, that is to say, on the HW platforms that are part of the project, and others in a virtual environment. In the case of some components, the functionalities could be tested on any HW/virtual environment, since by design they were valid for both High-end nodes and Mid-range nodes. In the case of the Low End Node, it was crucial to test the functionalities on the real HW.

Almost all tests passed validation. Except for one: T05_WP6T62-06_ANC. This failed test was communicated to the developers in order to improve the quality of the component.

10 Bibliography

Richardson, C. (2021). Microservices Pattern: Microservice Architecture pattern. Retrieved 12 April 2022, from https://microservices.io/patterns/microservices.html

Vaid, A., Maria, M., & Udupa, N. (2020). A Framework-driven Approach for Verification and Validation (V&V) of IoT Systems. Retrieved 5 April 2022, from https://www.wipro.com/content/dam/nexus/en/service-lines/product-engineering/latest-thinking/a-framework-driven-approach-for-verification-and-validation-v-and-v-of-iot-systems.pdf

11 List of figures

12 List of tables

13 List of abbreviations

API	Application Programming Interface
ATTNoC	Adaptable Time Triggered Network on Chip
CPU	Central Processing Unit
CRD	Custom Resource Definition
DB	Database
FPGA	Field Programmable Gate Array
HW	Hardware
ID	Identifier
IoT	Internet of Things
LED	Light Emitting Diode
MQTT	Message Queue Telemetry Transport
NGW	Network Gateway
NI	Network Interface
NoC	Network on Chip
PC	Personal Computer
PW	Password
RAM	Random Access Memory
REST	Representational State Transfer
SQL	Structured Query Language
SSID	Service Set Identifier
USB	Universal Serial Bus

14 Annexes

14.1 Orchestration (Edge Controller) component complete templates

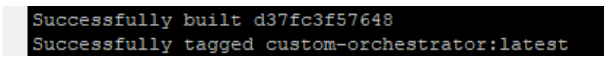
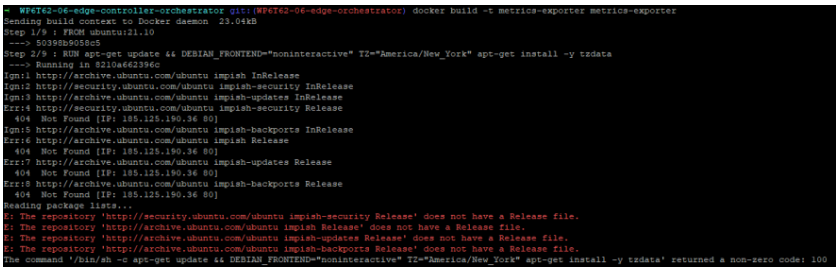
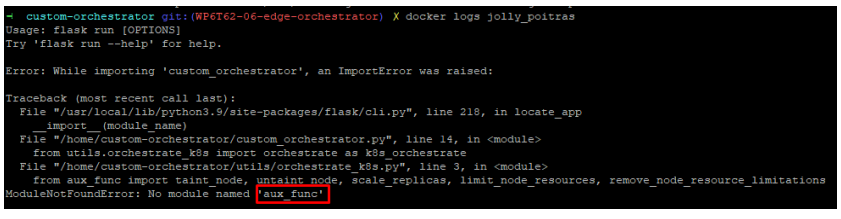
Validation test	
Test ID	T01_WP6T62-06_EC
Test type	Functional-Installation
Test name	Testing the installation of the component
Date	15/11/2022
Tester's Name	Ana Bautista
Test scope or objective	
The objective of this test is to validate if the Edge Controller Orchestrator can be installed without any issues.	
Steps	
Step 1	Prepare a node with Ubuntu and all needed dependencies (Python). This will be the master node.
Step 2	Deploy the metrics exporter container.
Step 3	Deploy the resource manager container.
Step 4	Deploy the custom orchestrator container.
Results/Evidence	
	
Success criteria	
No error messages/All partial results are as expected	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-06-edge-controller-orchestrator
Test conditions	The metrics exporter, resource manager and custom orchestrator containers runs in the same node (master node).
Remarks	Two bugs were found during installation that have been reported and corrected by the developers. 1. apt-get update no longer works on containers with Ubuntu21.10 so we need to use Ubuntu22.04.  2. The import 'aux_func' was corrected. 
Test result	
Passed	

Table 91 - Validation Test T01_WP6T62-06_EC (overview)

Validation test	
Test ID	T02_WP6T62-06_EC
Test type	Functional
Test name	Testing if the master node can monitor several (2) workers' nodes
Date	20/01/2023
Tester's Name	Ana Bautista
Test scope or objective	
The objective of this test is to validate if the master node can monitor two workers' nodes.	
Steps	
Step 1	Prepare a node with Ubuntu and all needed dependencies (Python). This will be the master node.
Step 2	Deploy the metrics exporter container.
Step 3	Deploy the resource manager container.
Step 4	Deploy the custom orchestrator container.
Step 5	Prepare tow nodes with Ubuntu and all needed dependencies (Python). These will be the worker's nodes.
Step 6	Deploy the metrics exporter container on each of the worker nodes.
Step 7	Review the logs from the resource manager (deployed on the master node).
Results/Evidence	

Master node:

```
WP6T62-06-edge-controller-orchestrator git:(WP6T62-06-edge-orchestrator) docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED        STATUS        PORTS     NAMES
d110afe4abf6   resource-manager     "/bin/sh -c 'python3…"   20 hours ago   Up 20 hours             zen_feistel
27a41d81626b   metrics-exporter     "/bin/sh -c 'glances…"   20 hours ago   Up 20 hours             quirky_curran
02edb3c53b93   custom-orchestrator  "/bin/sh -c 'flask r…"   20 hours ago   Up 20 hours             sleepy_jones
```

Worker node 1:

```
WP6T62-06-edge-controller-orchestrator git:(WP6T62-06-edge-orchestrator) docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED        STATUS        PORTS     NAMES
b13d2b0b9af9   metrics-exporter   "/bin/sh -c 'glances…"   22 hours ago   Up 22 hours             peaceful_gauss
```

Worker node 2:

```
WP6T62-06-edge-controller-orchestrator git:(WP6T62-06-edge-orchestrator) docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED        STATUS        PORTS     NAMES
4570cb97cacc   metrics-exporter   "/bin/sh -c 'glances…"   22 hours ago   Up 22 hours             pedantic_antonelli
```

Review logs from the resorce manager:

```
2023-02-16 14:56:12,187 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total CPU usage: 0.5 %
2023-02-16 14:56:12,187 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total memory: 16.783896576 Gb
2023-02-16 14:56:12,187 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Available memory: 15.629955072000001 Gb
2023-02-16 14:56:12,187 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total memory usage: 6.9 %
2023-02-16 14:56:12,187 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Load avg 1 min: 0.19775390625 %
2023-02-16 14:56:12,187 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Alerts: []
2023-02-16 14:56:12,188 - logger - INFO - Tainted nodes: {}
2023-02-16 14:56:12,278 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total CPU usage: 0.6 %
2023-02-16 14:56:12,278 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total memory: 16.783896576 Gb
2023-02-16 14:56:12,278 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Available memory: 15.620268032 Gb
2023-02-16 14:56:12,278 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total memory usage: 6.9 %
2023-02-16 14:56:12,278 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Load avg 1 min: 0.0126953125 %
2023-02-16 14:56:12,278 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Alerts: []
2023-02-16 14:56:12,278 - logger - INFO - Tainted nodes: {}
2023-02-16 14:56:27,369 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total CPU usage: 0.5 %
2023-02-16 14:56:27,369 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total memory: 16.783896576 Gb
2023-02-16 14:56:27,369 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Available memory: 15.630966784000002 Gb
2023-02-16 14:56:27,369 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total memory usage: 6.9 %
2023-02-16 14:56:27,369 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Load avg 1 min: 0.1533203125 %
2023-02-16 14:56:27,369 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Alerts: []
2023-02-16 14:56:27,369 - logger - INFO - Tainted nodes: {}
2023-02-16 14:56:27,455 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total CPU usage: 0.5 %
2023-02-16 14:56:27,455 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total memory: 16.783896576 Gb
2023-02-16 14:56:27,455 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Available memory: 15.621595136000002 Gb
2023-02-16 14:56:27,455 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total memory usage: 6.9 %
2023-02-16 14:56:27,455 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Load avg 1 min: 0.00927734375 %
2023-02-16 14:56:27,456 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Alerts: []
2023-02-16 14:56:27,456 - logger - INFO - Tainted nodes: {}
2023-02-16 14:56:42,572 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total CPU usage: 0.5 %
2023-02-16 14:56:42,572 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total memory: 16.783896576 Gb
2023-02-16 14:56:42,572 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Available memory: 15.630843904 Gb
2023-02-16 14:56:42,572 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total memory usage: 6.9 %
2023-02-16 14:56:42,572 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Load avg 1 min: 0.34716796875 %
2023-02-16 14:56:42,572 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Alerts: []
2023-02-16 14:56:42,573 - logger - INFO - Tainted nodes: {}
2023-02-16 14:56:42,652 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total CPU usage: 0.5 %
2023-02-16 14:56:42,653 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total memory: 16.783896576 Gb
2023-02-16 14:56:42,653 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Available memory: 15.62216448 Gb
2023-02-16 14:56:42,653 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total memory usage: 6.9 %
2023-02-16 14:56:42,653 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Load avg 1 min: 0.00634765625 %
2023-02-16 14:56:42,653 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Alerts: []
2023-02-16 14:56:42,653 - logger - INFO - Tainted nodes: {}
```

Success criteria	
No error messages/All partial results are as expected	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-06-edge-controller-orchestrator
Test conditions	The metrics exporter, resource manager and custom orchestrator containers run in the master node. The metrics exporter runs in the two worker's nodes.
Remarks	Reviewing the logs from the resource manager it can be observed that the information from the worker's nodes is given in the right way (as expected).
Test result	
Passed	

Table 92 - Validation Test T02_WP6T62-06_EC (overview)

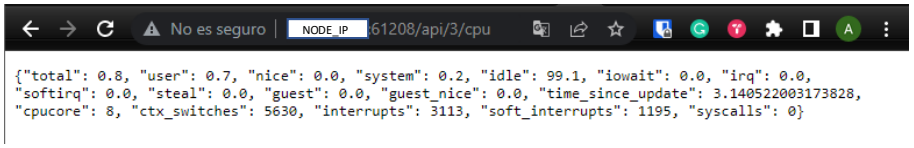
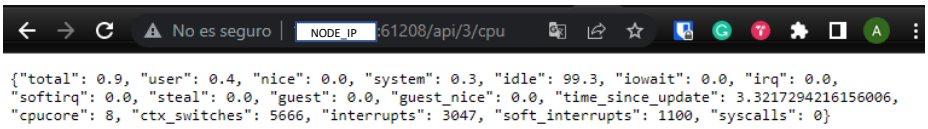
Validation test	
Test ID	T03_WP6T62-06_EC
Test type	Functional
Test name	Testing through the REST API if the metrics exporter is working properly
Date	15/11/2022
Tester's Name	Ana Bautista
Test scope or objective	
The objective of this test is to validate through the REST API if the metrics exporter is working properly.	
Steps	
Step 1	Prepare a node with Ubuntu and all needed dependencies (Python). This will be the master node.
Step 2	Deploy the metrics exporter container.
Step 3	Deploy the resource manager container.
Step 4	Deploy the custom orchestrator container.
Step 5	Prepare tow nodes with Ubuntu and all needed dependencies (Python). These will be the worker's nodes.
Step 6	Deploy the metrics exporter container on each of the worker nodes.
Step 7	Go to: http://<NODE_IP>:61208/api/3/cpu
Results/Evidence	
Master node:  {"total": 1.1, "user": 0.7, "nice": 0.0, "system": 0.4, "idle": 98.8, "iowait": 0.1, "irq": 0.0, "softirq": 0.0, "steal": 0.0, "guest": 0.0, "guest_nice": 0.0, "time_since_update": 1128.0130908489227, "cpucore": 8, "ctx_switches": 7384901, "interrupts": 3861801, "soft_interrupts": 838841, "syscalls": 0} **Worker node 1:**  {"total": 0.8, "user": 0.7, "nice": 0.0, "system": 0.2, "idle": 99.1, "iowait": 0.0, "irq": 0.0, "softirq": 0.0, "steal": 0.0, "guest": 0.0, "guest_nice": 0.0, "time_since_update": 3.140522003173828, "cpucore": 8, "ctx_switches": 5630, "interrupts": 3113, "soft_interrupts": 1195, "syscalls": 0} **Worker node 2:**  {"total": 0.9, "user": 0.4, "nice": 0.0, "system": 0.3, "idle": 99.3, "iowait": 0.0, "irq": 0.0, "softirq": 0.0, "steal": 0.0, "guest": 0.0, "guest_nice": 0.0, "time_since_update": 3.3217294216156006, "cpucore": 8, "ctx_switches": 5666, "interrupts": 3047, "soft_interrupts": 1100, "syscalls": 0}	
Success criteria	
No error messages/All partial results are as expected	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-06-edge-controller-orchestrator
Test conditions	The metrics exporter, resource manager and custom orchestrator containers run in the master node. The metrics exporter runs in the two worker's nodes.
Remarks	The REST API exposed by the custom orchestrator is reached by the resource manager and provides information about the nodes previously configured (as expected).
Test result	
Passed	

Table 93 - Validation Test T03_WP6T62-06_EC (overview)

Validation test	
Test ID	T04_WP6T62-06_EC
Test type	Functional
Test name	Testing how the resource manager behaves if the nodes are stressed
Date	15/11/2022
Tester's Name	Ana Bautista
Test scope or objective	
The objective of this test is to observe how the resource manager behaves if the nodes are stressed.	
Steps	
Step 1	Prepare a node with Ubuntu and all needed dependencies (Python). This will be the master node.
Step 2	Deploy the metrics exporter container.
Step 3	Deploy the resource manager container.
Step 4	Deploy the custom orchestrator container.
Step 5	Prepare tow nodes with Ubuntu and all needed dependencies (Python). These will be the worker's nodes.
Step 6	Deploy the metrics exporter container on each of the worker nodes.
Step 7	Install **stress-ng o**n one of the worker nodes.
Step 8	Execute the command **stress-ng --cpu 8 --timeout 60s** which will stress the node for 60 seconds.
Step 9	Review the logs from the resource manager (deployed on the master node).
Step 10	Check the alerts and the metrics in the logs of the resource manager.
Results/Evidence	

Master node:

```
WP6T62-06-edge-controller-orchestrator git:(WP6T62-06-edge-orchestrator) docker logs zen_feistel
```

Stressed worker node: fractal-k8s0.ipd.ikerlan.es

```
WP6T62-06-edge-controller-orchestrator git:(WP6T62-06-edge-orchestrator) stress-ng --cpu 8 --timeout 60s
stress-ng: info:  [1051804] dispatching hogs: 8 cpu
stress-ng: info:  [1051804] successful run completed in 60.07s (1 min, 0.07 secs)
```

Results 1:

Stressed worker node: fractal-k8s0.ipd.ikerlan.es

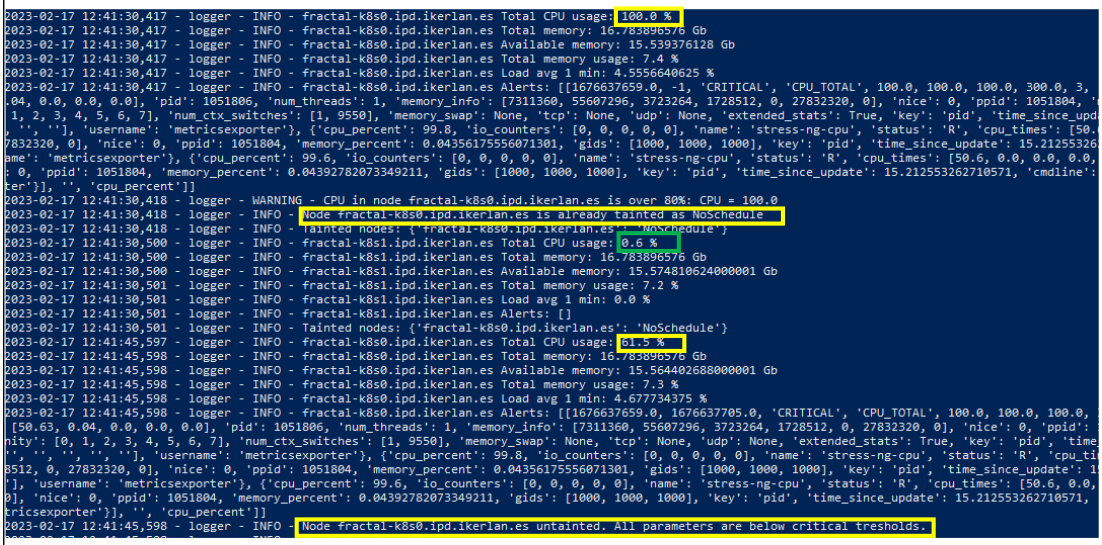
Worker node: fractal-k8s1.ipd.ikerlan.es

```
2023-02-17 12:40:44,687 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total CPU usage: 33.1 %
2023-02-17 12:40:44,687 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total memory: 16.783896576 Gb
2023-02-17 12:40:44,687 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Available memory: 15.542321152000001 Gb
2023-02-17 12:40:44,687 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total memory usage: 7.4 %
2023-02-17 12:40:44,687 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Load avg 1 min: 0.6962890625 %
2023-02-17 12:40:44,687 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Alerts: []
2023-02-17 12:40:44,688 - logger - INFO - Tainted nodes: {}
2023-02-17 12:40:44,792 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total CPU usage: 0.5 %
2023-02-17 12:40:44,792 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total memory: 16.783896576 Gb
2023-02-17 12:40:44,792 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Available memory: 15.575318528 Gb
2023-02-17 12:40:44,792 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total memory usage: 7.2 %
2023-02-17 12:40:44,792 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Load avg 1 min: 0.0 %
2023-02-17 12:40:44,792 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Alerts: []
2023-02-17 12:40:44,792 - logger - INFO - Tainted nodes: {}
2023-02-17 12:40:59,933 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total CPU usage: 100.0 %
2023-02-17 12:40:59,933 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total memory: 16.783896576 Gb
2023-02-17 12:40:59,933 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Available memory: 15.541850112 Gb
2023-02-17 12:40:59,933 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Total memory usage: 7.4 %
2023-02-17 12:40:59,933 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Load avg 1 min: 2.314453125 %
2023-02-17 12:40:59,933 - logger - INFO - fractal-k8s0.ipd.ikerlan.es Alerts: [[1676637659.0, -1, 'CRITICAL', 'CPU_TOTAL', 100.0, 100.0, 100.0, 100.0, 1, [], '', 'cpu_percent']]
2023-02-17 12:40:59,934 - logger - WARNING - CPU in node fractal-k8s0.ipd.ikerlan.es is over 80%: CPU = 100.0
2023-02-17 12:40:59,934 - logger - INFO - Tainting node fractal-k8s0.ipd.ikerlan.es to avoid scheduling of any other containers
2023-02-17 12:40:59,934 - logger - INFO - Tainted nodes: {'fractal-k8s0.ipd.ikerlan.es': 'NoSchedule'}
2023-02-17 12:41:00,064 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total CPU usage: 0.6 %
2023-02-17 12:41:00,064 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total memory: 16.783896576 Gb
2023-02-17 12:41:00,064 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Available memory: 15.574118400000001 Gb
2023-02-17 12:41:00,065 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Total memory usage: 7.2 %
2023-02-17 12:41:00,065 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Load avg 1 min: 0.0 %
2023-02-17 12:41:00,065 - logger - INFO - fractal-k8s1.ipd.ikerlan.es Alerts: []
2023-02-17 12:41:00,065 - logger - INFO - Tainted nodes: {'fractal-k8s0.ipd.ikerlan.es': 'NoSchedule'}
```

(continues)

(continued)

Results 2:



Success criteria	
No error messages/All partial results are as expected	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-06-edge-controller-orchestrator
Test conditions	The metrics exporter, resource manager and custom orchestrator containers run in the master node. The metrics exporter runs in the two worker's nodes. The node called **fractal-k8s0.ipd.ikerlan.es** is the node that was stressed.
Remarks	As it can be observed in the "Results 1" screenshot: when the CPU usage of a node is over 80% it is considered as tainted (low on resources and restricted) as NoSchedule. According to the component documentation, if any of the monitored resources are above some fixed thresholds, that node is no longer able to perform any new container deployments until the resource limitation is lifted.
Test result	
Passed	

Table 94 - Validation Test T04_WP6T62-06_EC (overview)

14.2 Orchestration (Agent Nodes Controller) component complete templates

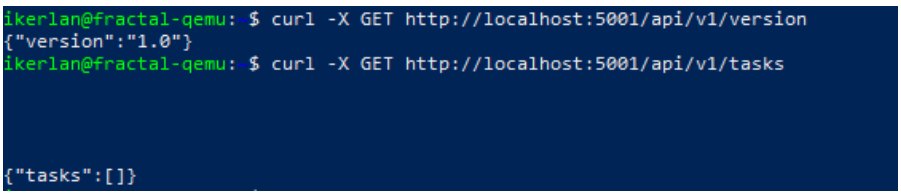
Validation test	
Test ID	T01_WP6T62-06_ANC
Test type	Functional
Test name	Testing that the Agent nodes controller can be installed without any issues
Date	20/01/2023
Tester's Name	Ana Bautista, Adrian Moran
Test scope or objective	
The objective of the test is to validate if the Agent nodes controller can be installed without any issues.	
Steps	
Step 1	Prepare a node with Ubuntu and all needed dependencies (Python).
Step 2	Launch backend-service-manager (Service Manager).
Step 3	Launch API Server.
Step 4	Launch frontend-service-manager (Executor Node).
Results/Evidence	
	
Success criteria	
No error messages/All partial results are as expected	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-06-mid-range-orchestration
Test conditions	API Server, Service Manager and Executor Node runs in the same node.
Remarks	All the subcomponents are up and running.
Test result	
Passed	

Table 95 - Validation Test T01_WP6T62-06_ANC (overview)

Validation test	
Test ID	T02_WP6T62-06_ANC
Test type	Functional
Test name	Testing basic orchestration functionality
Date	09/02/2023
Tester's Name	Ana Bautista, Adrian Moran
Test scope or objective	
The objective of the test is to validate basic orchestration functionality.	
Steps	
Step 1	Prepare a node with Ubuntu and all needed dependencies (Python).
Step 2	Launch backend-service-manager (Service Manager).
Step 3	Launch API Server.
Step 4	Launch frontend-service-manager (Executor Node).
Step 5	From a different device, use REST API to create a new task.
Results/Evidence	

Client:
`→ task curl -X POST http://172.16.58.4:5001/api/v1/tasks/test -F file=@test_task.py -F "cmd=test_task.py" -F "rt="`
`{"status":"task: test created successfuly."}`
API:
`172.16.105.43 - - [09/Feb/2023 11:13:59] "DELETE /api/v1/tasks/test HTTP/1.1" 200 -`
`172.16.105.43 - - [09/Feb/2023 11:14:58] "POST /api/v1/tasks/test HTTP/1.1" 201 -`
Executor Node:
`2023-02-09 11:14:58,501 INFO heartbeat received`
`('task', 71, 'json', {'task_name': 'test', 'args_to_run': 'test_task.py', 'return_type': ''})`
`2023-02-09 11:14:58,514 INFO running task: test`
`2023-02-09 11:15:00,504 INFO heartbeat received`
`2023-02-09 11:15:02,507 INFO heartbeat received`
`2023-02-09 11:15:03,535 INFO task: test completed with return code: 0`
`2023-02-09 11:15:04,508 INFO heartbeat received`	

Success criteria	
No error messages/All partial results are as expected	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-06-mid-range-orchestration
Test conditions	API Server, Service Manager and Executor Node runs in the same node.
Remarks	Basic workflow is completed successfully.
Test result	
Passed	

Table 96 - Validation Test T02_WP6T62-06_ANC (overview)

Validation test	
Test ID	T03_WP6T62-06_ANC
Test type	Functional
Test name	Testing that a running task can be deleted
Date	09/02/2023
Tester's Name	Ana Bautista, Adrian Moran
Test scope or objective	
The objective of the test is to validate that a running task can be deleted.	
Steps	
Step 1	Prepare a node with Ubuntu and all needed dependencies (Python).
Step 2	Launch backend-service-manager (Service Manager).
Step 3	Launch API Server.
Step 4	Launch frontend-service-manager (Executor Node).
Step 5	From a different device, use REST API to create a new task and delete it.
Results/Evidence	

Client:
`→ task curl -X DELETE http://172.16.58.4:5001/api/v1/tasks/test`
`{"backend-status":{"status":"ok"},"status":"task: test deleted successfuly from API host."}`
 	
API:
`172.16.105.43 - - [09/Feb/2023 11:17:23] "DELETE /api/v1/tasks/test HTTP/1.1" 200 -`	

Success criteria	
No error messages/All partial results are as expected	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-06-mid-range-orchestration
Test conditions	API Server, Service Manager and Executor Node runs in the same node.
Remarks	Task is deleted from API Server repository succesfully. Deleted task is not deleted from Executor node .tasks folder, which can lead to problems like lack of storage or DoS attacks.
Test result	
Passed	

Table 97 - Validation Test T03_WP6T62-06_ANC (overview)

Test ID	T04_WP6T62-06_ANC
Test type	Functional
Test name	Testing that a running task can be stopped and started again
Date	09/02/2023
Tester's Name	Ana Bautista, Adrian Moran
Test scope or objective	
The objective of the test is to validate that a running task can be stopped and started again.	
Steps	
Step 1	Prepare a node with Ubuntu and all needed dependencies (Python).
Step 2	Launch backend-service-manager (Service Manager).
Step 3	Launch API Server.
Step 4	Launch frontend-service-manager (Executor Node).
Step 5	From a different device, use REST API to create a new task, stop and then start it.
Results/Evidence	

Client:

```
→  task curl -X POST http://172.16.58.4:5001/api/v1/tasks/test -F file=@test_task.py -F "cmd=test_task.py" -F "rt="
{"status":"task: test created successfuly."}
→  task curl -X POST http://172.16.58.4:5001/api/v1/tasks/test/stop
{"status":"ok"}
→  task curl -X POST http://172.16.58.4:5001/api/v1/tasks/test/start
{"status":"ok"}
→  task curl -X DELETE http://172.16.58.4:5001/api/v1/tasks/test
{"backend-status":{"status":"ok"},"status":"task: test deleted successfuly from API host."}
```

API:

```
172.16.105.43 - - [09/Feb/2023 11:18:35] "POST /api/v1/tasks/test HTTP/1.1" 201 -
127.0.0.1 - - [09/Feb/2023 11:18:35] "GET /api/v1/tasks/test/download HTTP/1.1" 200 -
172.16.105.43 - - [09/Feb/2023 11:18:45] "POST /api/v1/tasks/test/stop HTTP/1.1" 200 -
172.16.105.43 - - [09/Feb/2023 11:18:48] "POST /api/v1/tasks/test/start HTTP/1.1" 200 -
127.0.0.1 - - [09/Feb/2023 11:18:48] "GET /api/v1/tasks/test/download HTTP/1.1" 200 -
from server {'status': 'ok'}
172.16.105.43 - - [09/Feb/2023 11:18:58] "DELETE /api/v1/tasks/test HTTP/1.1" 200 -
```

Executor Node:

```
2023-02-09 11:18:34,783 INFO heartbeat received
('task', 71, 'json', {'task_name': 'test', 'args_to_run': 'test_task.py', 'return_type': ''})
2023-02-09 11:18:35,796 INFO running task: test
2023-02-09 11:18:36,785 INFO heartbeat received
2023-02-09 11:18:38,788 INFO heartbeat received
2023-02-09 11:18:40,791 INFO heartbeat received
2023-02-09 11:18:42,794 INFO heartbeat received
2023-02-09 11:18:44,797 INFO heartbeat received
2023-02-09 11:18:45,786 INFO task stopped
2023-02-09 11:18:45,786 INFO task: test terminated with return code: -15
2023-02-09 11:18:46,800 INFO heartbeat received
2023-02-09 11:18:48,803 INFO heartbeat received
('task', 71, 'json', {'task_name': 'test', 'args_to_run': 'test_task.py', 'return_type': ''})
2023-02-09 11:18:48,852 INFO running task: test
2023-02-09 11:18:50,806 INFO heartbeat received
2023-02-09 11:18:52,809 INFO heartbeat received
2023-02-09 11:18:54,812 INFO heartbeat received
2023-02-09 11:18:56,814 INFO heartbeat received
2023-02-09 11:18:58,512 INFO task stopped
2023-02-09 11:18:58,513 INFO task: test terminated with return code: -15
```

Success criteria	
No error messages/All partial results are as expected	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-06-mid-range-orchestration
Test conditions	API Server, Service Manager and Executor Node runs in the same node.
Remarks	Task lifecycle in correctly handled.
Test result	
Passed	

Table 98 - Validation Test T04_WP6T62-06_ANC (overview)

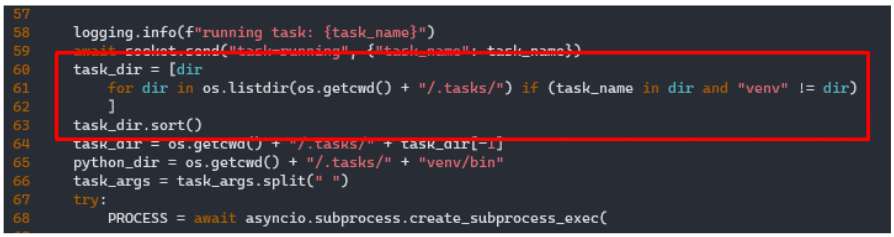
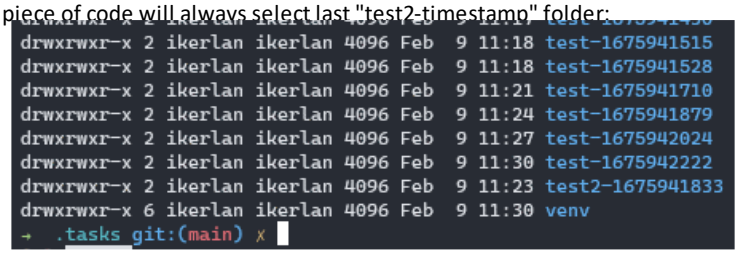
Validation test	
Test ID	T05_WP6T62-06_ANC
Test type	Functional
Test name	Testing that a running multiple tasks is possible and list their state
Date	09/02/2023
Tester's Name	Ana Bautista, Adrian Moran
Test scope or objective	
The objective of the test is to validate that a running multiple tasks is possible and list their state.	
Steps	
Step 1	Prepare a node with Ubuntu and all needed dependencies (Python).
Step 2	Launch backend-service-manager (Service Manager).
Step 3	Launch API Server.
Step 4	Launch frontend-service-manager (Executor Node).
Step 5	From a different device, use REST API to create a new tasks.
Results/Evidence	

```
→  task curl -X GET http://172.16.58.4:5001/api/v1/tasks
{"tasks":[]}
→  task curl -X POST http://172.16.58.4:5001/api/v1/tasks/test -F file=@test_task.py -F "cmd=test_task.py" -F "rt="
{"status":"task: test created successfuly."}
→  task curl -X GET http://172.16.58.4:5001/api/v1/tasks
{"tasks":["test"]}
→  task curl -X POST http://172.16.58.4:5001/api/v1/tasks/test2 -F file=@test_task.py -F "cmd=test_task.py" -F "rt="
{"status":"task: test2 created successfuly."}
→  task curl -X GET http://172.16.58.4:5001/api/v1/tasks
{"tasks":["test","test2"]}
→  task curl -X GET http://172.16.58.4:5001/api/v1/tasks/test/status
{"status":"ok","task_status":"running"}
→  task curl -X GET http://172.16.58.4:5001/api/v1/tasks/test2/status
{"status":"ok","task_status":"created"}
→  task
```

```
172.16.105.43 - - [09/Feb/2023 11:21:41] "GET /api/v1/tasks HTTP/1.1" 404 -
172.16.105.43 - - [09/Feb/2023 11:21:49] "POST /api/v1/tasks/test HTTP/1.1" 201 -
127.0.0.1 - - [09/Feb/2023 11:21:50] "GET /api/v1/tasks/test/download HTTP/1.1" 200 -
from server {'status': 'ok', 'tasks': ['test']}
172.16.105.43 - - [09/Feb/2023 11:21:51] "GET /api/v1/tasks HTTP/1.1" 200 -
172.16.105.43 - - [09/Feb/2023 11:21:55] "POST /api/v1/tasks/test2 HTTP/1.1" 201 -
from server {'status': 'ok', 'tasks': ['test', 'test2']}
172.16.105.43 - - [09/Feb/2023 11:21:59] "GET /api/v1/tasks HTTP/1.1" 200 -
172.16.105.43 - - [09/Feb/2023 11:22:12] "GET /api/v1/tasks/test/status HTTP/1.1" 200 -
172.16.105.43 - - [09/Feb/2023 11:22:16] "GET /api/v1/tasks/test2/status HTTP/1.1" 200 -
```

(continues)

(continued)

Success criteria	
No error messages/All partial results are as expected	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-06-mid-range-orchestration
Test conditions	API Server, Service Manager and Executor Node runs in the same node.
Remarks	It is possible to list the tasks. It is possible to add multiple tasks and get their status. A bug has occured generating two task, one named "test" and other one named "test2". Executor Node computes the execution path from name, and since it does not deletes the old tasks, this leads to failure:  This way of computing task_dir is bugged. If the same node has executed in their lifetime a tasks called "test" and "test2", there won't be any chance of running "test" task, since this piece of code will always select last "test2-timestamp" folder: 
Test result	
Not passed	

```
    logging.info(f"running task: {task_name}")
    await socket.send("task-running", {"task_name": task_name})
    task_dir = [dir
        for dir in os.listdir(os.getcwd() + "/.tasks/") if (task_name in dir and "venv" != dir)
        ]
    task_dir.sort()
    task_dir = os.getcwd() + "/.tasks/" + task_dir[-1]
    python_dir = os.getcwd() + "/.tasks/" + "venv/bin"
    task_args = task_args.split(" ")
    try:
        PROCESS = await asyncio.subprocess.create_subprocess_exec(
```

```
drwxrwxr-x 2 ikerlan ikerlan 4096 Feb  9 11:18 test-1675941515
drwxrwxr-x 2 ikerlan ikerlan 4096 Feb  9 11:18 test-1675941528
drwxrwxr-x 2 ikerlan ikerlan 4096 Feb  9 11:21 test-1675941710
drwxrwxr-x 2 ikerlan ikerlan 4096 Feb  9 11:24 test-1675941879
drwxrwxr-x 2 ikerlan ikerlan 4096 Feb  9 11:27 test-1675942024
drwxrwxr-x 2 ikerlan ikerlan 4096 Feb  9 11:30 test-1675942222
drwxrwxr-x 2 ikerlan ikerlan 4096 Feb  9 11:23 test2-1675941833
drwxrwxr-x 6 ikerlan ikerlan 4096 Feb  9 11:30 venv
→  .tasks git:(main) ✗
```

Table 99 - Validation Test T05_WP6T62-06_ANC (overview)

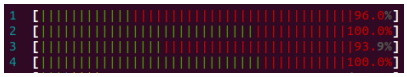
Validation test	
Test ID	T06_WP6T62-06_ANC
Test type	Functional
Test name	Testing the behavior of the orchestrator with multiple Executor Nodes
Date	09/02/2023
Tester's Name	Ana Bautista, Adrian Moran
Test scope or objective	
The objective of the test is to validate the behaviour of the orchestrator with multiple Executor Nodes.	
Steps	
Step 1	Prepare a node with Ubuntu and all needed dependencies (Python).
Step 2	Launch backend-service-manager (Service Manager).
Step 3	Launch API Server.
Step 4	Launch two instances of frontend-service-manager (Executor Node).
Step 5	From a different device, use REST API to create two simultaneous tasks.
Results/Evidence	

Client:
`→ task curl -X POST http://172.16.58.4:5001/api/v1/tasks/test -F file=@test_task.py -F "cmd=test_task.py" -F "rt="`
`{"status":"task: test created successfuly."}`
`→ task curl -X POST http://172.16.58.4:5001/api/v1/tasks/ikerlan -F file=@test_task.py -F "cmd=test_task.py" -F "rt="`
`{"status":"task: ikerlan created successfuly."}`
API:
`172.16.105.43 - - [09/Feb/2023 13:22:36] "POST /api/v1/tasks/test HTTP/1.1" 201 -`
`127.0.0.1 - - [09/Feb/2023 13:22:36] "GET /api/v1/tasks/test/download HTTP/1.1" 200 -`
`172.16.105.43 - - [09/Feb/2023 13:22:42] "POST /api/v1/tasks/ikerlan HTTP/1.1" 201 -`
`127.0.0.1 - - [09/Feb/2023 13:22:42] "GET /api/v1/tasks/ikerlan/download HTTP/1.1" 200 -`
Executor Node 1:
`2023-02-09 13:22:34,827 INFO heartbeat received`
`('task', 71, 'json', {'task_name': 'test', 'args_to_run': 'test_task.py', 'return_type': ''})`
`2023-02-09 13:22:36,821 INFO running task: test`
`2023-02-09 13:22:36,828 INFO heartbeat received`
Executor Node 2:
`2023-02-09 13:22:42,737 INFO heartbeat received`
`('task', 74, 'json', {'task_name': 'ikerlan', 'args_to_run': 'test_task.py', 'return_type': ''})`
`2023-02-09 13:22:42,823 INFO running task: ikerlan`
`2023-02-09 13:22:44,739 INFO heartbeat received`	

Success criteria	
No error messages/All partial results are as expected	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-06-mid-range-orchestration
Test conditions	API Server, Service Manager and Executor Node runs in the same node.
Remarks	Multiple nodes works fine.
Test result	
Passed	

Table 100 - Validation Test T06_WP6T62-06_ANC (overview)

14.3 Runtime Manager component complete templates

Validation test	
Test ID	T01_WP6T62-03
Test type	Functional
Test Name	Testing interaction between nodes with local node overloaded
Date	05/12/2022
Tester's Name	Luca Visconti (Modis Consulting SRL)
Test scope or objective	
The objective of this test is to validate the correct interaction and data exchange between nodes	
Steps	
Step 1	Create the condition that overload the Node 1 

Step 2	Send the command of the execution flow to RM1 running "test_mqtt_published.py"
	`root@xilinx-zcu102-2021_2:~/N1/WP6T62-03-Runtime-Manager-main/test# python3 test_mqtt_publisher.py`
	`data published`
	`[rm_mqtt -> Receiver] - triggered by MQTT`
	`msg_s= {'id_flow': '2', 'payload': '0x03abcdefghil'}`

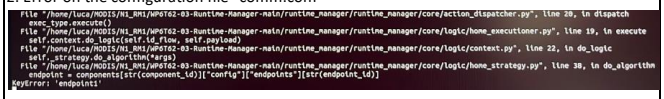
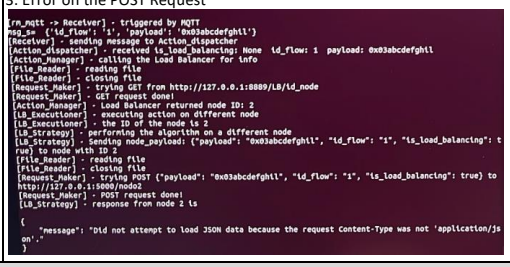
Results/Evidence	
N1:  N2: 	
Success criteria	
The interaction and data exchange between nodes have to be executed successfully	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-03-Runtime-Manager RM{1, 2, 3} = Runtime Manager on the node with id = 1, 2, 3 N{1, 2, 3} = Node with id = 1, 2, 3 N1 is defined by Zynq UltraScale+ ZCU102 board with IP=192.168.0.1 and PORT=7777 N2 is defined by a Virtual Machine on Computer with IP=192.168.0.2 and PORT=8888 N3 is defined by a Virtual Machine on Computer with IP=192.168.0.2 and PORT=9999
Test conditions	N1 is overloaded and it cannot perform other computation
Remarks	Three bugs were foud during validation test that have been reported and corrected by the developers. 1. Error on the configuration file "component.conf"  2. Error on the configuration file "comm.conf"  3. Error on the POST Request 
Test Result	
Passed	

Table 101 - Validation Test T01_WP6T62-03 (overview)

Validation test	
Test ID	T02_WP6T62-03
Test type	Functional
Test Name	Testing interaction in the local node when the local node can perform the computation
Date	05/12/2022
Tester's Name	Luca Visconti (Modis Consulting SRL)
Test scope or objective	
The objective of this test is to validate the correct interaction and data exchange in the local node	
Steps	

Step 1	Send the command of the execution flow to RM1 running "test_mqtt_published.py"
	`root@xilinx-zcu102-2021_2:~/N1/WP6T62-03-Runtime-Manager-main/test# python3 test_mqtt_publisher.py`
	`data published`
	`[rm_mqtt -> Receiver] - triggered by MQTT`
	`msg_s= {'id_flow': '2', 'payload': '0x03abcdefghil'}`

Results/Evidence	

N1:	`[rm_mqtt -> Receiver] - triggered by MQTT`
	`msg_s= {'id_flow': '2', 'payload': '0x03abcdefghil'}`
	`[Receiver] - sending message to Action_dispatcher`
	`[Action_dispatcher] - received is_load_balancing: None id_flow: 2 payload: 0x03abcdefghil`
	`[Action_Manager] - calling the Load Balancer for info`
	`[File_Reader] - reading file`
	`[File_Reader] - closing file`
	`[Request_Maker] - trying GET from http://127.0.0.1:7776/LB/id_node`
	`[Request_Maker] - GET request done!`
	`[Action_Manager] - Load Balancer returned node ID: None`
	`[Home_Executioner] - executing action here`
	`[Home_Executioner] - the ID of the flow to be executed is 2`
	`[Home_Executioner] - payload is 0x03abcdefghil`
	`[Home_Strategy] - performing the algorithm at home`
	`[File_Reader] - reading file`
	`[File_Reader] - closing file`
	`[File_Reader] - reading file`
	`[File_Reader] - closing file`
	`['TO', 'component1', '2']`
	`['GET']`
	`[Request_Maker] - trying GET from http://127.0.0.1:6000/endpoint2`
	`[Request_Maker] - GET request done!`
	`['TO', 'component2', '1']`
	`['POST', 'result1', 'payload']`
	`[Request_Maker] - trying POST {'payload': ['{\n "endpoint2_res": "something"\n}\n', '0x03abcdefghil']} to http://127.0.0.1:6000/endpoint1`
	`[Request_Maker] - POST request done!`

Success criteria	
The interaction and data exchange in the local nodes have to be executed successfully	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-03-Runtime-Manager
	RM{1, 2, 3} = Runtime Manager on the node with id = 1, 2, 3
	N{1, 2, 3} = Node with id = 1, 2, 3
	N1 is defined by Zynq UltraScale+ ZCU102 board with IP=192.168.0.1 and PORT=7777
	N2 is defined by a Virtual Machine on Computer with IP=192.168.0.2 and PORT=8888
	N3 is defined by a Virtual Machine on Computer with IP=192.168.0.2 and PORT=9999
Test conditions	N1 is not overloaded and it can perform any computation
Remarks	
Test Result	
Passed	

Table 102 - Validation Test T02_WP6T62-03 (overview)

Validation test	
Test ID	T03_WP6T62-03
Test type	Functional
Test Name	Testing interaction between nodes with Node 1 and Node 2 overloaded
Date	05/12/2022
Tester's Name	Luca Visconti (Modis Consulting SRL)
Test scope or objective	
The objective of this test is to validate the correct interaction and data exchange between nodes	
Steps	
Step 1	Create the condition that overload the Node 1 1 [||||||||||||||||||||||||||||||||||||||96.1%] 2 [||||||||||||||||||||||||||||||||||||||100.0%] 3 [||||||||||||||||||||||||||||||||||||||100.0%] 4 [||||||||||||||||||||||||||||||||||||||93.8%]
step 2	Create the condition that overload the Node 2 1 [||||||||||||||||||||||||||||||||||||||95.3%] 2 [||||||||||||||||||||||||||||||||||||||92.3%] 3 [||||||||||||||||||||||||||||||||||||||89.3%]

Step 3	Send the command of the execution flow to RM1 running "test_mqtt_published.py"
	`root@xilinx-zcu102-2021_2:~/N1/WP6T62-03-Runtime-Manager-main/test# python3 test_mqtt_publisher.py`
	`data published`
	`[rm_mqtt -> Receiver] - triggered by MQTT`
	`msg_s= {'id_flow': '2', 'payload': '0x03abcdefghil'}`

Results/Evidence	

N1:
`[rm_mqtt -> Receiver] - triggered by MQTT`
`msg_s= {'id_flow': '2', 'payload': '0x03abcdefghil'}`
`[Receiver] - sending message to Action_dispatcher`
`[Action_dispatcher] - received is_load_balancing: None id_flow: 2 payload: 0x03abcdefghil`
`[Action_Manager] - calling the Load Balancer for info`
`[File_Reader] - reading file`
`[File_Reader] - closing file`
`[Request_Maker] - trying GET from http://127.0.0.1:7776/LB/id_node`
`[Request_Maker] - GET request done!`
`[Action_Manager] - Load Balancer returned node ID: 3`
`[LB_Executioner] - executing action on different node`
`[LB_Executioner] - the ID of the node is 3`
`[LB_Strategy] - performing the algorithm on a different node`
`[LB_Strategy] - Sending node_payload: {"payload": "0x03abcdefghil", "id_flow": "2", "is_load_balancing": true} to node with ID 3`
`[File_Reader] - reading file`
`[File_Reader] - closing file`
`[Request_Maker] - trying POST {"payload": "0x03abcdefghil", "id_flow": "2", "is_load_balancing": true} to http://192.168.0.2:9999/nodo3`
`[Request_Maker] - POST request done!`
`[LB_Strategy] - response from node 3 is`
`200`
	N3:
	`[rm_api -> Receiver] - received POST request`
	`[Receiver] - sending message to Action_dispatcher`
	`[Action_dispatcher] - received is_load_balancing: True id_flow: 2 payload: 0x03abcdefghil`
	`[Home_Executioner] - executing action here`
	`[Home_Executioner] - the ID of the flow to be executed is 2`
	`[Home_Executioner] - payload is 0x03abcdefghil`
	`[Home_Strategy] - performing the algorithm at home`
	`[File_Reader] - reading file`
	`[File_Reader] - closing file`
	`[File_Reader] - reading file`
	`[File_Reader] - closing file`
	`['TO', 'component1', '2']`
	`['GET']`
	`[Request_Maker] - trying GET from http://127.0.0.1:6000/endpoint2`
	`[Request_Maker] - GET request done!`
	`['TO', 'component2', '1']`
	`['POST', 'result1', 'payload']`
	`[Request_Maker] - trying POST {'payload': ['{\n "endpoint2_res": "something"\n}\n', '0x03abcdefghil']} to http://127.0.0.1:6000/endpoint1`
	`[Request_Maker] - POST request done!`
	`192.168.0.1 - - [14/Dec/2022 14:30:56] "POST /nodo3 HTTP/1.1" 200 -`	

Success criteria	
The interaction and data exchange between nodes have to be executed successfully	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-03-Runtime-Manager
	RM{1, 2, 3} = Runtime Manager on the node with id = 1, 2, 3
	N{1, 2, 3} = Node with id = 1, 2, 3
	N1 is defined by Zynq UltraScale+ ZCU102 board with IP=192.168.0.1 and PORT=7777
	N2 is defined by a Virtual Machine on Computer with IP=192.168.0.2 and PORT=8888
	N3 is defined by a Virtual Machine on Computer with IP=192.168.0.2 and PORT=9999
Test conditions	N1 and N2 is overloaded and it cannot perform other computation
Remarks	
Test Result	
Passed	

Table 103 - Validation Test T03_WP6T62-03 (overview)

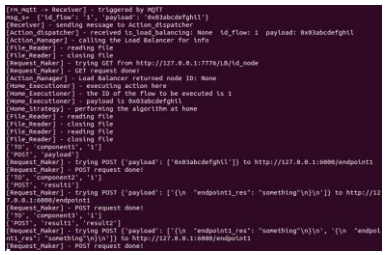
Validation test	
Test ID	T04_WP6T62-03
Test type	Functional
Test Name	Task Scheduling on the local node
Date	05/12/2022
Tester's Name	Luca Visconti (Modis Consulting SRL)
Test scope or objective	
The objective of this test is to validate the correct execution of the Task in the local node	
Steps	

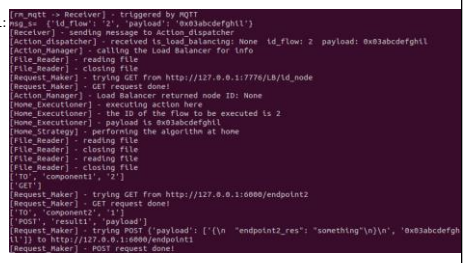
Step 1	Send the command of the execution flow "1" to RM1 running "test_mqtt_published.py" with "id_flow=1".
	`root@xilinx-zcu102-2021_2:~/N1/WP6T62-03-Runtime-Manager-main/test# python3 test_mqtt_publisher.py`
	`data published`
	`[rm_mqtt -> Receiver] - triggered by MQTT`
	`msg_s= {'id_flow': '1', 'payload': '0x03abcdefghil'}`

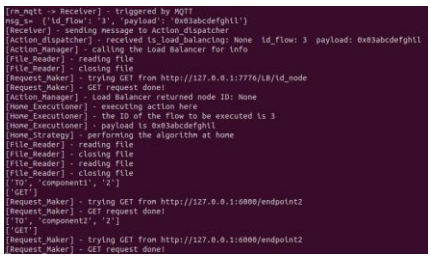
Step 2	Send the command of the execution flow "2" to RM1 running "test_mqtt_published.py" with "id_flow=2".
	`root@xilinx-zcu102-2021_2:~/N1/WP6T62-03-Runtime-Manager-main/test# python3 test_mqtt_publisher.py`
	`data published`
	`[rm_mqtt -> Receiver] - triggered by MQTT`
	`msg_s= {'id_flow': '2', 'payload': '0x03abcdefghil'}`

Step 3	Send the command of the execution flow "3" to RM1 running "test_mqtt_published.py" with "id_flow=3".
	`root@xilinx-zcu102-2021_2:~/N1/WP6T62-03-Runtime-Manager-main/test# python3 test_mqtt_publisher.py`
	`data published`
	`[rm_mqtt -> Receiver] - triggered by MQTT`
	`msg_s= {'id_flow': '3', 'payload': '0x03abcdefghil'}`

Results/Evidence	

Result Step 1:
Flow1:
`1`
`TO component1 1`
`POST payload`
`TO component2 1`
`POST result1`
`TO component3 1`
`POST result1 result2`
N1: 	

Result Step 2:
Flow2:
`2`
`TO component1 2`
`GET`
`TO component2 1`
`POST result1 payload`
N1: 	

Result Step 3:
Flow3:
`3`
`TO component1 2`
`GET`
`TO component2 2`
`GET`
N1: 	

Success criteria	
The execution of the task in local node have to be executed successfully	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-03-Runtime-Manager RM{1, 2, 3} = Runtime Manager on the node with id = 1, 2, 3 N{1, 2, 3} = Node with id = 1, 2, 3 N1 is defined by Zynq UltraScale+ ZCU102 board with IP=192.168.0.1 and PORT=7777 N2 is defined by a Virtual Machine on Computer with IP=192.168.0.2 and PORT=8888 N3 is defined by a Virtual Machine on Computer with IP=192.168.0.2 and PORT=9999
Test conditions	N1 is not overloaded and it can perform any computation
Remarks	
Test Result	
Passed	

Table 104 - Validation Test T04_WP6T62-03 (overview)

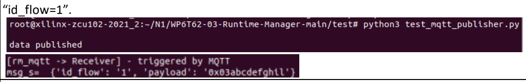
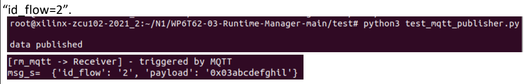
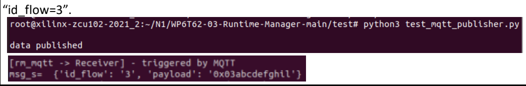
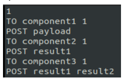
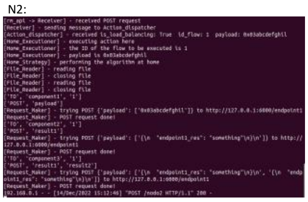
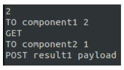
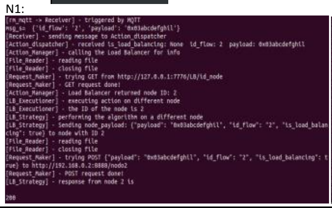
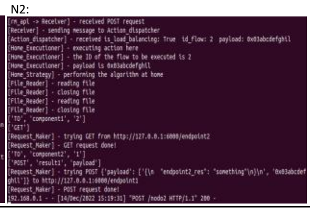
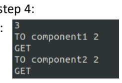
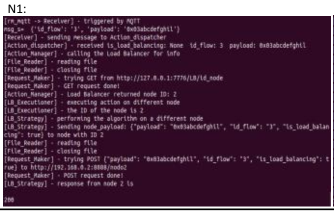
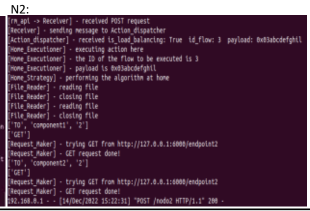
Validation test	
Test ID	T05_WP6T62-03
Test type	Functional
Test Name	Task Scheduling on the remote node
Date	05/12/2022
Tester's Name	Luca Visconti (Modis Consulting SRL)
Test scope or objective	
The objective of this test is to validate the correct execution of the tasks in the remote node	
Steps	
Step 1	Create the condition that overload the Node 1 
Step 2	Send the command of the execution flow "1" to RM1 running "test_mqtt_published.py" with "id_flow=1". 
Step 3	Send the command of the execution flow "2" to RM1 running "test_mqtt_published.py" with "id_flow=2". 
Step 4	Send the command of the execution flow "3" to RM1 running "test_mqtt_published.py" with "id_flow=3". 
Results/Evidence	
Result step 2: Flow1:  N1:  N2: 	
Result step 3: Flow2:  N1:  N2: 	
Result step 4: Flow2:  N1:  N2: 	
Success criteria	
The execution of the task and the exchange data with remote node have to be executed successfully	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-03-Runtime-Manager
	RM{1, 2, 3} = Runtime Manager on the node with id = 1, 2, 3
	N{1, 2, 3} = Node with id = 1, 2, 3
	N1 is defined by Zynq UltraScale+ ZCU102 board with IP=192.168.0.1 and PORT=7777
	N2 is defined by a Virtual Machine on Computer with IP=192.168.0.2 and PORT=8888
	N3 is defined by a Virtual Machine on Computer with IP=192.168.0.2 and PORT=9999
Test conditions	N1 is overloaded and it cannot perform other computation
Remarks	
Test Result	
Passed	

Table 105 - Validation Test T05_WP6T62-03 (overview)

14.4 Data Ingestion component complete templates

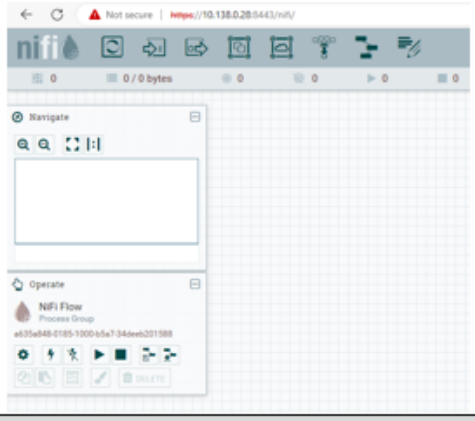
Validation test	
Test ID	T01_WP6T62-01_DI - Testing Apache NiFi
Test type	Functional-Installation
Test name	Installation of Apache NiFi
Date	13/01/2023
Tester's Name	Nicola Alchera (Rulex)
Test scope or objective	
The objective of this test is to validate that Apache Nifi can be installed without any issue.	
Steps	
Step 1	Prepare a node with Ubuntu and required dependencies (Python)
Step 2	Download files and checking prerequisites
Step 3	Configure and run ApacheNifi
Step 4	Login and Open Apache Nifi in browser
Results/Evidence	
The Nifi software has been successfully opened and it is ready to use 	
Success criteria	
No error messages/All partial results are as expected	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-01-data-ingestion
Test conditions	The tests have been performed on a node with Ubuntu 22.04.
Remarks	Running *./nifi.sh start* you might be warned that the Java home path is empty. This seems to be a warning. It can be fixed using the following command "*export JAVA_HOME="your_java_folder_path"* 
	You cannot download the tarball file from *https://nifi.apache.org/download.html* because the 1.18 and 1.19 release has the .zip binary files only. To download the .tar file you need to consider the archive *https://archive.apache.org/dist/nifi/* . The latest version with the .tar file available is the 1.16.3 and this is the one that has been used.
Test result	
Passed	

Table 106: Validation Test T01_WP6T62-01_DI (overview)

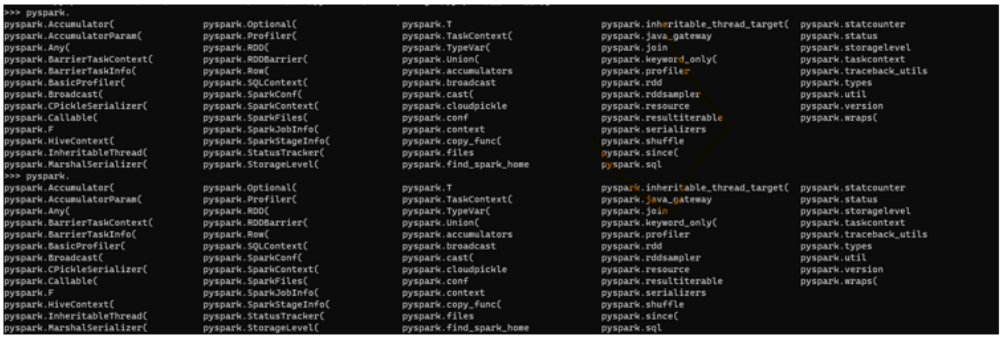
Validation test	
Test ID	T02_WP6T62-01_DI
Test type	Functional
Test name	Installation and configuration of PySpark
Date	16/02/2023
Tester's Name	Nicola Alchera (Rulex)
Test scope or objective	
The objective of this test is to ensure that PySPark can be installed and configured without any issue.	
Steps	
Step 1	Prepare a node with Ubuntu and the required dependencies (Python 3)
Step 2	Install pySpark and open it on python3
Results/Evidence	
pySpark has been correctly installed and it can be used as a Python3 package	
	
Success criteria	
No error messages	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-01-data-ingestion
Test conditions	The tests have been performed on a node with Ubuntu 22.04.
Remarks	No issues have been higlighted during the test
Test result	
Passed	

Table 107: Validation Test T02_WP6T62-01_DI (overview)

Validation test	
Test ID	T03_WP6T62-01_DI
Test type	Functional-Installation
Test name	Installation and configuration of Faust
Date	16/02/2023
Tester's Name	Nicola Alchera (Rulex)
Test scope or objective	
The objective of this test is to validate the Faust installation guidelines.	
Steps	
Step 1	Prepare a node with Ubuntu and required dependencies (python 3)
Step 2	install Faust and run python3
Results/Evidence	
Faust has been correctly installed and it can be used as python3 library	
Success criteria	
No error messages/All partial results are as expected	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-01-data-ingestion
Test conditions	The tests have been performed on a node with Ubuntu 22.04.
Remarks	No issues have been highlighted during the test
Test result	
Passed	

Table 108: Validation Test T03 WP6T62-01_DI (overview)

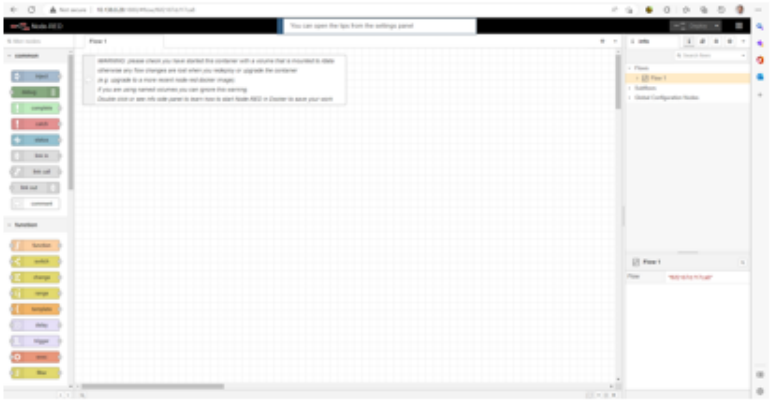
Validation test	
Test ID	T04_WP6T62-01_DI
Test type	Functional-Installation
Test name	Installation of RedNote
Date	16/02/2023
Tester's Name	Nicola Alchera (Rulex)
Test scope or objective	
The objective of this test is to validate the RedNote installation guidelines.	
Steps	
Step 1	Prepare a node with Ubuntu and required dependencies (python 3)
Step 2	Install Docker
Step 3	Run the docker
Step 4	Open the docker in browser
Results/Evidence	
Rednote has been successfully opened and it is ready to use 	
Success criteria	
RedNote has been installed without major errors	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-01-data-ingestion
Test conditions	The tests have been performed on a node with Ubuntu 22.04.
Remarks	Check you have started this container with a volume mounted on /data. I checked, and it seems the container is monted on /data, as you can see in the image below. Despite this, the workflow is not saved . 
Test result	
Passed	

Table 109: Validation Test T04_WP6T62-01_DI (overview)

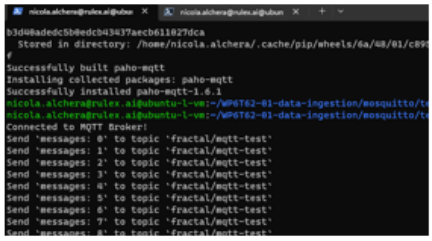
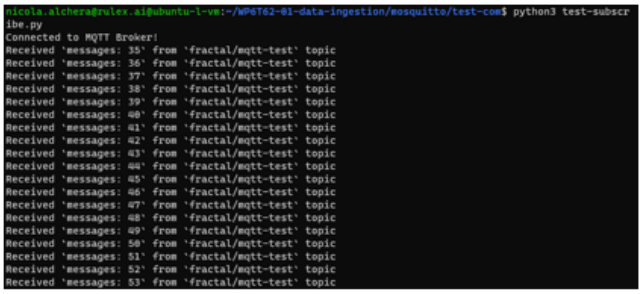
Validation test	
Test ID	T05_WP6T62-01_DI
Test type	Functional-Installation
Test name	test of MQTT Cloud Communications Component
Date	03/03/23
Tester's Name	Nicola Alchera (Rulex)
Test scope or objective	
The objective of this test is to validate the MQTT Cloud Communication guidelines.	
Steps	
Step 1	Prepare a node with Ubuntu and required dependencies (python 3)
Step 2	Set up a container with an MQTT broker
Step 3	Test the publish functionalities
Step 4	Test the subscribe functionalities
Results/Evidence	
Both the publisher and subscriber script works correctly.  	
Success criteria	
No error messages/All partial results are as expected	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-01-data-ingestion
Test conditions	The tests have been performed on a node with Ubuntu 22.04.
Remarks	The clone command "git clone https://github.com/project-fractal/WP6T62-01-data-ingestion.git" could not work properly. The user password authentication is no longer supported: a valid token is required for access. If you haven't any token, you have create a new one following the instructions you can fine here https://docs.github.com/en/get-started/getting-started-with-git/about-remote-repositories#cloning-with-https-urls The command "source run.sh" could give you an error. To avoid the error you shold give the permission through the following command "sudo chmod 777 /var/run/docker.sock" There is an error in guideline: guidlines says to test two times the same python script (test-publish.py) and not the "test-publish.py" and the "test-subscribe.py"
Test result	
Passed	

Table 110: Validation Test T05_WP6T62-01_DI (overview)

14.5 Federated Data Collection component complete templates

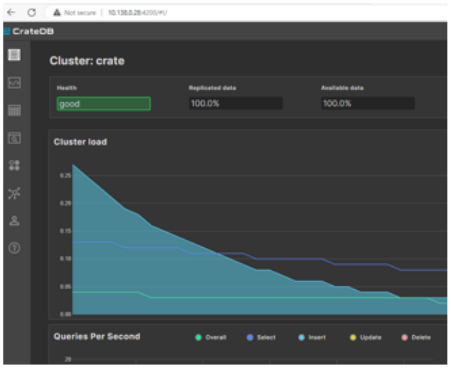
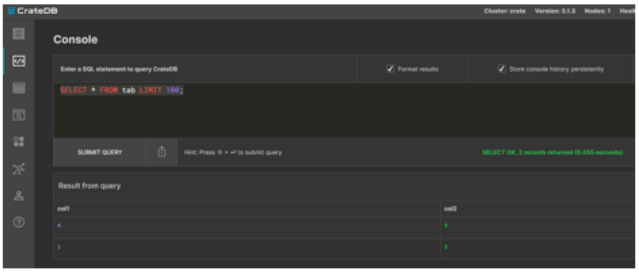
Validation test	
Test ID	T01_WP6T62-02_FDC
Test type	Functional-Installation
Test name	Installation of CrateDB
Date	16/01/2023
Tester's Name	Nicola Alchera (Rulex)
Test scope or objective	
The objective of this test is to validate the CrateDB installation guidelines.	
Steps	
Step 1	Prepare a node with Ubuntu and required dependencies
Step 2	Install CrateDB on Ubuntu
Step 3	Run the docker and open CrateDB in a web browser
Step 4	Create and view a table using SQL into Crate DB
Results/Evidence	
CrateDB has been successfully opened in browser and it is ready to use	
  	
Success criteria	
Data can be created and viewed.	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-02-federated_data_collection
Test conditions	The tests have been performed on a node with Ubuntu 22.04.
Remarks	No issues were highlighted during the test
Test result	
Passed	

Table 111: Validation Test T01_WP6T62-02_FDC (overview)

Validation test	
Test ID	T02_WP6T62-02_FDC
Test type	Functional-Installation
Test name	Installation of MongoDB
Date	24/01/2023
Tester's Name	Nicola Alchera (Rulex)
Test scope or objective	
The objective of this test is to validate the MongoDB installation guidelines.	
Steps	
Step 1	Prepare a node with Ubuntu and required dependencies
Step 2	Install Mongo on Ubuntu
Step 3	Run the docker
Results/Evidence	

MongoDB has been successfully installed
`nicola.alchera@rulex.ai@ubuntu-l-vm:~$ sudo systemctl status mongod`
`● mongod.service - MongoDB Database Server`
`Loaded: loaded (/lib/systemd/system/mongod.service; disabled; vendor preset: enabled)`
`Active: active (running) since Tue 2023-01-24 11:18:21 UTC; 10s ago`
`Docs: https://docs.mongodb.org/manual`
`Main PID: 563824 (mongod)`
`Memory: 168.3M`
`CGroup: /system.slice/mongod.service`
`└─563824 /usr/bin/mongod --config /etc/mongod.conf`
`Jan 24 11:18:21 ubuntu-l-vm systemd[1]: Started MongoDB Database Server.`	

Success criteria	
Data can be created and viewed via MongoDB	
Test observations	
Test configuration	https://github.com/project-fractal/WP6T62-02-federated_data_collection
Test conditions	The tests have been performed on a node with Ubuntu 22.04.
Remarks	No issues were highlighted during the test
Test result	
Passed	

Table 112: Validation Test T02_WP6T62-02_FDC (overview)

14.6 Low End Node component complete templates

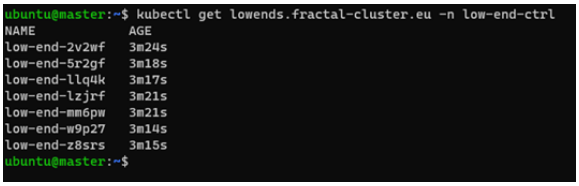
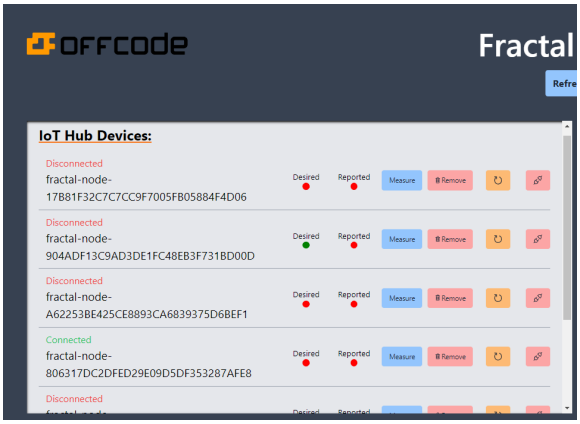
Validation Test	
Test ID	T01_WP6T62-06
Test type	Functional
Test name	Testing the connection between the Device and the Cloud Platform
Date	19/01/2022
Tester's Name	Luca Visconti (Akkodis)
Test scope or objective	
The objective of this test is to validate the connection between Low End Node Device and Cloud Platform.	
Steps	
Step 1	Power up the device
Step 2	Connect the device to internet
Results/Evidence	
Step 2:    	
Success criteria	
Device accepted and device status "connected"	
Test observations	
Test configuration	Device access point config: SSID and PW
Test conditions	Device connected to local wifi
Remarks	
Test result	
Passed	

Table 113 - Validation Test T01_WP6T62-06 (overview)

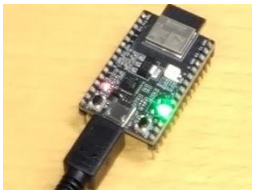
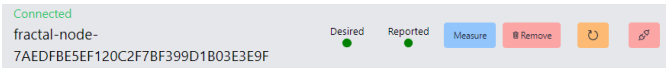
Validation Test	
Test ID	T02_WP6T62-06
Test type	Functional
Test name	Testing the communication from Device to the Cloud
Date	19/01/2022
Tester's Name	Luca Visconti (Akkodis)
Test scope or objective	
The objective of this test is to validate communication between Low End Node Device and Cloud Platform.	
Steps	
Step 1	Power up the device
Step 2	Connect the device to internet
Step 3	Change device status by pressing button
Results/Evidence	
Step 2:   Step 3:  	
Success criteria	
Reported status as expected	
Test observations	
Test configuration	Device access point config: SSID and PW
Test conditions	Device connected to local wifi
Remarks	
Test result	
Passed	

Table 114 - Validation Test T02_WP6T62-06 (overview)

Validation Test	
Test ID	T03_WP6T62-06
Test type	Functional
Test name	Testing the communication from the Cloud Platform to the Device
Date	19/01/2022
Tester's Name	Luca Visconti (Akkodis)
Test scope or objective	
The objective of this test is to validate the communication between Cloud Platform and Low End Node Device.	
Steps	
Step 1	Power up the device
Step 2	Connect the device to internet

Step 3	Change status in Kubernetes by "patch" method
	`ubuntu@master:~$ kubectl patch -n low-end-ctrl lowends.fractal-cluster.eu low-end-5r2gf --type merge --patch '{ "spec": { "desiredState": { "state": 1} }}'`
	`lowend.fractal-cluster.eu/low-end-5r2gf patched`

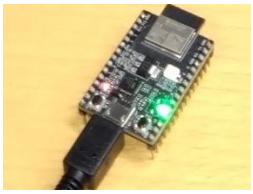
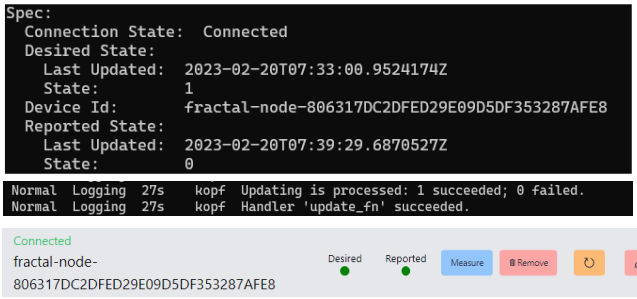
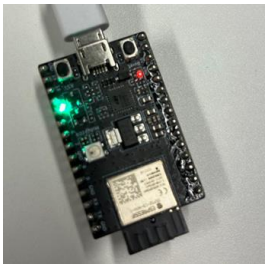
Results/Evidence	
Step 2:  	
Step 3:  	
Success criteria	
Device accepted and device status "connected"	
Test observations	
Test configuration	Device access point config: SSID and PW
Test conditions	Device connected to local wifi
Remarks	
Test result	
Passed	

Table 115 - Validation Test T03_WP6T62-06 (overview)

Validation Test	
Test ID	T04_WP6T62-06
Test type	Functional
Test name	Testing the Tasks Scheduling running Nuttx on the Device
Date	19/01/2022
Tester's Name	Luca Visconti (Akkodis)
Test scope or objective	
The objective of this test is to validate the task scheduling on the Low End Node	
Steps	
Step 1	Power up the device
Step 2	Connect the device by usb to a pc
Step 3	Open terminal connection and run command "ps" `$ minicom -D /dev/ttyUSB1` offcode> offcode> ps
Results/Evidence	

Step 2:

```
$dmesg
[598658.253187] usb 3-2.1.4: new full-speed USB device number 16 using xhci_hcd
[598658.338481] usb 3-2.1.4: New USB device found, idVendor=10c4, idProduct=ea60,
bcdDevice=                                                              1.00
[598658.338492]  usb  3-2.1.4:  New  USB  device  strings:  Mfr=1,  Product=2,
SerialNumber=3
[598658.338497]  usb  3-2.1.4:  Product:  CP2102N  USB  to  UART  Bridge  Controller
[598658.338501]      usb      3-2.1.4:      Manufacturer:      Silicon      Labs
[598658.338504]  usb  3-2.1.4:  SerialNumber:  363aaf4f44a0eb11a2a8cbacdf749906
[598658.340065]     cp210x     3-2.1.4:1.0:     cp210x     converter     detected
[598658.345156] usb 3-2.1.4: cp210x converter now attached to ttyUSB1
```

Step 3:

```
offcode>
offcode> ps
 PID GROUP PRI POLICY   TYPE    NPX STATE    EVENT     SIGMASK   STACK COMMAND
   0    0   0 FIFO     Kthread N-- Ready              00000000 004048 Idle Task
   1    1 224 RR       Kthread --- Waiting  Semaphore 00000000 004016 hpwork 0x3fc844d8
   2    2 224 RR       Kthread --- Waiting  Semaphore 00000000 004016 hpwork 0x3fc844d8
   3    3 100 RR       Kthread --- Waiting  Semaphore 00000000 004016 lpwork 0x3fc844ec
   4    4 100 RR       Kthread --- Waiting  Semaphore 00000000 004016 lpwork 0x3fc844ec
   5    5 100 RR       Task    --- Running            00000000 001968 nsh_main
   6    6 223 RR       Kthread --- Waiting  Semaphore 00000000 001968 rt_timer
   7    7 253 RR       Kthread --- Waiting  MQ empty  00000000 006592 wifi
   8    8 100 RR       Task    --- Waiting  Signal    00000000 002944 NTP daemon
0.pool.ntp.org;1.pool.ntp.org;2.pool.ntp.org
  10   10 100 RR       Task    --- Waiting  Signal    00000000 016320 fractal
  11   10 100 RR       pthread --- Waiting  Signal    00000000 003024 pt-0x420347ee
0x3fcb0bd0
  12   10 100 RR       pthread --- Waiting  Signal    00000000 003024 pt-0x420347ee
0x3fcb6800
offcode>
```

Success criteria	
Print all parallel running task as expected	
Test observations	
Test configuration	GitHub repository: https://github.com/project-fractal/WP6T62-06-low-end-node-orchestrator Device access point config: SSID and PW
Test conditions	Device connected to local wifi
Remarks	
Test result	
Passed	

Table 116 - Validation Test T04_WP6T62-06 (overview)

14.7 Hardware-level Edge Controller component complete templates

Validation Test	
Test ID	T01_WP6T62-0X
Test type	Functional
Test name	Testing the Message-Classification and Message-Scheduling Services at different port according to scheduling configuration
Date	03/02/2022
Tester's Name	Luca Visconti (Akkodis)
Test scope or objective	
The objective of this test is to validate the message-classification and message-scheduling services sending 3 message at a different port and according to the scheduling configuration.	
Steps	
Step 1	Define a set of messages to send (3 messages at 3 different port according to the scheduling configuration TTCommScheld.cfg of each NI) - 1 message to NI0 on the port 2 Port ID | Instant | Next | Instant (microsec) 2 [NI0-NI2] | 956 (7) | 0 | 28,483 - 2 messages to NI1 on the port 2 and 3 Port ID | Instant | Next | Instant (microsec) 2 [NI1-NI0] | 1092 (8) | 1 | 32,552 3 [NI1-NI3] | 2594 (19) | 0 | 77,311

Step 2	Send a message to NI0:
	`// send message from NI0 on port 2`
	`for(int i=0;i<mesg_size;i++)`
	`{`
	`Xil_Out32(XPAR_SKINOC_0_S00_AXI_BASEADDR+131072,i); // Write Message on NI0, port2`
	`}`
	`Xil_Out32(XPAR_SKINOC_0_S00_AXI_BASEADDR+163840,7); // Terminate the Message`
	Send messages to NI1:
	`// send messages from NI1 on port 2`
	`for(int i=0;i<mesg_size;i++)`
	`{`
	`Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+131072,i); // Write Message on NI0, port2`
	`}`
	`Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+163840,7); // Terminate the Message`
	`// send messages from NI1 on port 3`
	`for(int i=0;i<mesg_size;i++)`
	`{`
	`Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+196608,i); // Write Message on NI0, port2`
	`}`
	`Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+229376,7); // Terminate the Message`

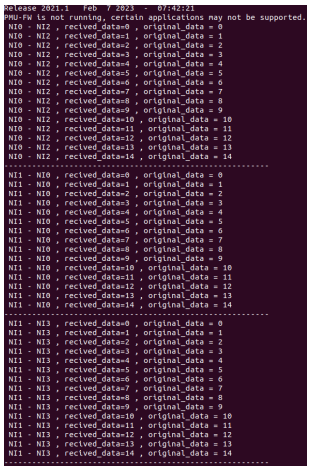
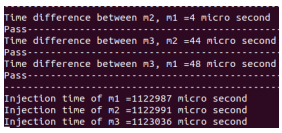
Results/Evidence	
Step 2:  	
Success criteria	
Check: - destination port as per configuration "port.cfg" - message content (not corrupted) - period as per configuration "hw.cfg" - Is it possible to check the "instant" (Time Unit)?	
Test observations	
Test configuration	Config: "ttcommsched.cfg", "port.cfg", "hw.cfg"
Test conditions	Vitis for FPGA configuration, ATTNoC running on the board, all configuration files at github link https://github.com/project-fractal/WP6T62-HW-Edge-Controller
Remarks	
Test result	
Passed	

Table 117 - Validation Test T01_WP6T62-0X (overview)

Validation Test	
Test ID	T02_WP6T62-0X
Test type	Functional
Test name	Testing the Message-Classification and Message-Scheduling Services at same port according to scheduling configuration
Date	03/02/2022
Tester's Name	Luca Visconti (Akkodis)
Test scope or objective	
The objective of this test is to validate the message-classification and message-scheduling services sending message at same port according to scheduling configuration	
Steps	
Step 1	Define a set of messages to send (3 messages at 3 from the same port according to the scheduling configuration TTCommScheld.cfg of NI0) - 3 messages to NI0 on the port 2:

Port ID	Instant	Next	Instant (microsec)
2 [NI0-NI2]	956 (7)	0	28,483

Step 2: Send messages to NI1:

```
// send messages from NI1-NI0 using port2
for(int i=0;i<mesg_size;i++) // Message size = 5
{
    Xil_Out32(XPAR_SKINOC_0_S00_AXI_BASEADDR+131072,i); // Decimale (02) 000(
}
Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+163840,7); // terminate Message  00(

/*-------------------------------------------------------------------*/

// send message from NI1-NI3
for(int i=0;i<mesg_size;i++) // Message size = 5
{
    Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+196608,i);
}
Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+229376,7); // terminate Message

/*-------------------------------------------------------------------*/

// send messages from NI1-NI0.
for(int i=0;i<mesg_size;i++) // Message size = 5
{
    Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+131072,i);
}
Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+163840,7); // terminate Message

/*-------------------------------------------------------------------*/

  // send message from NI1-NI3
  for(int i=0;i<mesg_size;i++) // Message size = 5
  {
    Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+196608,i);
  }
  Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+229376,7); // terminate Message

  /*-------------------------------------------------------------------*/

   // send messages from NI1-NI0.
   for(int i=0;i<mesg_size;i++) // Message size = 5
   {
     Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+131072,i);
   }
    Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+163840,7); // terminate Message

  /*-------------------------------------------------------------------*/

  // send message from NI1-NI3
  for(int i=0;i<mesg_size;i++) // Message size = 5
  {
    Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+196608,i);
  }
  Xil_Out32(XPAR_SKINOC_0_S01_AXI_BASEADDR+229376,7); // terminate Message
```

Results/Evidence

Step 2:

```
NI1 - NI3 , recived_data=0 , original_data = 0
NI1 - NI3 , recived_data=1 , original_data = 1
NI1 - NI3 , recived_data=2 , original_data = 2
NI1 - NI3 , recived_data=3 , original_data = 3
NI1 - NI3 , recived_data=4 , original_data = 4
NI1 - NI3 , recived_data=5 , original_data = 5
NI1 - NI3 , recived_data=6 , original_data = 6
NI1 - NI3 , recived_data=7 , original_data = 7
NI1 - NI3 , recived_data=8 , original_data = 8
NI1 - NI3 , recived_data=9 , original_data = 9
NI1 - NI3 , recived_data=10 , original_data = 10
NI1 - NI3 , recived_data=11 , original_data = 11
NI1 - NI3 , recived_data=12 , original_data = 12
NI1 - NI3 , recived_data=13 , original_data = 13
NI1 - NI3 , recived_data=14 , original_data = 14
-------------------------------------------------
NI1 - NI3 , recived_data=0 , original_data = 0
NI1 - NI3 , recived_data=1 , original_data = 1
NI1 - NI3 , recived_data=2 , original_data = 2
NI1 - NI3 , recived_data=3 , original_data = 3
NI1 - NI3 , recived_data=4 , original_data = 4
NI1 - NI3 , recived_data=5 , original_data = 5
NI1 - NI3 , recived_data=6 , original_data = 6
NI1 - NI3 , recived_data=7 , original_data = 7
NI1 - NI3 , recived_data=8 , original_data = 8
NI1 - NI3 , recived_data=9 , original_data = 9
NI1 - NI3 , recived_data=10 , original_data = 10
NI1 - NI3 , recived_data=11 , original_data = 11
NI1 - NI3 , recived_data=12 , original_data = 12
NI1 - NI3 , recived_data=13 , original_data = 13
NI1 - NI3 , recived_data=14 , original_data = 14
-------------------------------------------------
NI1 - NI3 , recived_data=0 , original_data = 0
NI1 - NI3 , recived_data=1 , original_data = 1
NI1 - NI3 , recived_data=2 , original_data = 2
NI1 - NI3 , recived_data=3 , original_data = 3
NI1 - NI3 , recived_data=4 , original_data = 4
NI1 - NI3 , recived_data=5 , original_data = 5
NI1 - NI3 , recived_data=6 , original_data = 6
NI1 - NI3 , recived_data=7 , original_data = 7
NI1 - NI3 , recived_data=8 , original_data = 8
NI1 - NI3 , recived_data=9 , original_data = 9
NI1 - NI3 , recived_data=10 , original_data = 10
NI1 - NI3 , recived_data=11 , original_data = 11
NI1 - NI3 , recived_data=12 , original_data = 12
NI1 - NI3 , recived_data=13 , original_data = 13
NI1 - NI3 , recived_data=14 , original_data = 14
```

```
Time difference between two consecutive messages m2 and m1 = 122
Time difference between two consecutive messages m3 and m2 = 122
-----------------------------------------------------------------
```

Success criteria	
Check: - destination port (and NI) as "port.cfg" - the message content (not corrupted) - period (we expect three different period)	
Test observations	
Test configuration	Config: "ttcommsched.cfg", "port.cfg", "hw.cfg"
Test conditions	Vitis for FPGA configuration, ATTNoC running on the board, all configuration files at github link https://github.com/project-fractal/WP6T62-HW-Edge-Controller
Remarks	
Test result	
Passed	

Table 118 - Validation Test T02_WP6T62-0X (overview)

Validation Test	
Test ID	T03_WP6T62-0X
Test type	Functional
Test name	Testing the Message-Classification and Message-Scheduling Services at same NI
Date	03/02/2022
Tester's Name	Luca Visconti (Akkodis)
Test scope or objective	
The objective of this test is to validate the message-classification and message-scheduling services sending 3 message at same Port when 2 port configured	
Steps	
Step 1	Define a set of messages to send (3 messages at 2 ports according to the scheduling configuration TTCommScheld.cfg of NI2) - 2 messages to NI2 on the port 1 - 1 message to NI2 on the port 2

Port ID	Instant	Next	Instant (microsec)
2 [NI2-NI0]	2048 (15)	1	61,035
3 [NI2-NI3]	2185 (16)	0	65,104

Step 2 — Send messages to NI2:

```
// send messages from NI2-NI0 using port2
for(int i=0;i<mesg_size;i++) // Message size = 5
{
    Xil_Out32(XPAR_SKINOC_0_S02_AXI_BASEADDR+131072,i);
}
Xil_Out32(XPAR_SKINOC_0_S02_AXI_BASEADDR+163840,7);

/*-------------------------------------------------------------------*/

// send message from NI2-NI3
for(int i=0;i<mesg_size;i++) // Message size = 5
{
    Xil_Out32(XPAR_SKINOC_0_S02_AXI_BASEADDR+196608,i);
}
Xil_Out32(XPAR_SKINOC_0_S02_AXI_BASEADDR+229376,7); // terminate Message

/*-------------------------------------------------------------------*/
// send messages from NI2-NI0 using port2
   for(int i=0;i<mesg_size;i++) // Message size = 5
   {
    Xil_Out32(XPAR_SKINOC_0_S02_AXI_BASEADDR+131072,i);
   }
   Xil_Out32(XPAR_SKINOC_0_S02_AXI_BASEADDR+163840,7);
/*-------------------------------------------------------------------*/
```

Results/Evidence

Step 2:

```
NI1 - NI0 , recived_data=0 , original_data = 0
NI1 - NI0 , recived_data=1 , original_data = 1
NI1 - NI0 , recived_data=2 , original_data = 2
NI1 - NI0 , recived_data=3 , original_data = 3
NI1 - NI0 , recived_data=4 , original_data = 4
NI1 - NI0 , recived_data=5 , original_data = 5
NI1 - NI0 , recived_data=6 , original_data = 6
NI1 - NI0 , recived_data=7 , original_data = 7
NI1 - NI0 , recived_data=8 , original_data = 8
NI1 - NI0 , recived_data=9 , original_data = 9
NI1 - NI0 , recived_data=10 , original_data = 10
NI1 - NI0 , recived_data=11 , original_data = 11
NI1 - NI0 , recived_data=12 , original_data = 12
NI1 - NI0 , recived_data=13 , original_data = 13
NI1 - NI0 , recived_data=14 , original_data = 14
-----------------------------------------------------
NI1 - NI3 , recived_data=0 , original_data = 0
NI1 - NI3 , recived_data=1 , original_data = 1
NI1 - NI3 , recived_data=2 , original_data = 2
NI1 - NI3 , recived_data=3 , original_data = 3
NI1 - NI3 , recived_data=4 , original_data = 4
NI1 - NI3 , recived_data=5 , original_data = 5
NI1 - NI3 , recived_data=6 , original_data = 6
NI1 - NI3 , recived_data=7 , original_data = 7
NI1 - NI3 , recived_data=8 , original_data = 8
NI1 - NI3 , recived_data=9 , original_data = 9
NI1 - NI3 , recived_data=10 , original_data = 10
NI1 - NI3 , recived_data=11 , original_data = 11
NI1 - NI3 , recived_data=12 , original_data = 12
NI1 - NI3 , recived_data=13 , original_data = 13
NI1 - NI3 , recived_data=14 , original_data = 14
-----------------------------------------------------
NI1 - NI0 , recived_data=0 , original_data = 0
NI1 - NI0 , recived_data=1 , original_data = 1
NI1 - NI0 , recived_data=2 , original_data = 2
NI1 - NI0 , recived_data=3 , original_data = 3
NI1 - NI0 , recived_data=4 , original_data = 4
NI1 - NI0 , recived_data=5 , original_data = 5
NI1 - NI0 , recived_data=6 , original_data = 6
NI1 - NI0 , recived_data=7 , original_data = 7
NI1 - NI0 , recived_data=8 , original_data = 8
NI1 - NI0 , recived_data=9 , original_data = 9
NI1 - NI0 , recived_data=10 , original_data = 10
NI1 - NI0 , recived_data=11 , original_data = 11
NI1 - NI0 , recived_data=12 , original_data = 12
NI1 - NI0 , recived_data=13 , original_data = 13
NI1 - NI0 , recived_data=14 , original_data = 14
```

```
Time difference between m2, m1 = 4
Time difference between m3, m1 = 122
```

Success criteria	
Check: - destination port (and NI) as "port.cfg" - the message content (not corrupted) - period ?? (Two different periods)	
Test observations	
Test configuration	Config: "ttcommsched.cfg", "port.cfg", "hw.cfg"
Test conditions	Vitis for FPGA configuration, ATTNoC running on the board, all configuration files at github link https://github.com/project-fractal/WP6T62-HW-Edge-Controller
Remarks	
Test result	
Passed	

Table 119 - Validation Test T03_WP6T62-0X (overview)

Validation Test	
Test ID	T04_WP6T62-0X
Test type	Functional
Test name	Testing the Ingress and Egress-queuing Services
Date	03/02/2022
Tester's Name	Luca Visconti (Akkodis)
Test scope or objective	
The objective of this test is to validate the ingress-queuing and egress-queuing services using the max lenght of queue.	
Steps	

Step 1	Send a message to NI0 on port 1 with a length less than the queue length
	`for(int i=0;i<mesg_size1;i++)`
	`{`
	`Xil_Out32(XPAR_SKINOC_0_S00_AXI_BASEADDR+131072,message1+i);`
	`}`
	`Xil_Out32(XPAR_SKINOC_0_S00_AXI_BASEADDR+163840,7);`

Step 2	Send a message to NI0 on port 1 with a length equal to the queue length
	`for(int i=0;i<mesg_size2;i++)`
	`{`
	`Xil_Out32(XPAR_SKINOC_0_S00_AXI_BASEADDR+131072,message2+i);`
	`}`
	`Xil_Out32(XPAR_SKINOC_0_S00_AXI_BASEADDR+163840,7);`

Step 3	Send a message to NI0 on port 1 with a length bigger than the queue length
	`for(int i=0;i<mesg_size3;i++)`
	`{`
	`Xil_Out32(XPAR_SKINOC_0_S00_AXI_BASEADDR+131072,message3+i);`
	`}`
	`Xil_Out32(XPAR_SKINOC_0_S00_AXI_BASEADDR+163840,7);`

Results/Evidence	

Step 1:
`Message id = 0, recived_data=3 , original_data = 3`
`Message id = 1, recived_data=4 , original_data = 4`
`Message id = 2, recived_data=5 , original_data = 5`
`Message id = 3, recived_data=6 , original_data = 6`
`Message id = 4, recived_data=7 , original_data = 7`
`Message id = 5, recived_data=8 , original_data = 8`
`Message id = 6, recived_data=9 , original_data = 9`
`Message id = 7, recived_data=10 , original_data = 10`
`Message id = 8, recived_data=11 , original_data = 11`
`Message id = 9, recived_data=12 , original_data = 12`
`Time stamp=37196732`
 	
Step 2:
`Message id = 0, recived_data=4 , original_data = 4`
`Message id = 1, recived_data=5 , original_data = 5`
`Message id = 2, recived_data=6 , original_data = 6`
`Message id = 3, recived_data=7 , original_data = 7`
`Message id = 4, recived_data=8 , original_data = 8`
`Message id = 5, recived_data=9 , original_data = 9`
`Message id = 6, recived_data=10 , original_data = 10`
`Message id = 7, recived_data=11 , original_data = 11`
`Message id = 8, recived_data=12 , original_data = 12`
`Message id = 9, recived_data=13 , original_data = 13`
`Message id = 10, recived_data=14 , original_data = 14`
`Message id = 11, recived_data=15 , original_data = 15`
`Message id = 12, recived_data=16 , original_data = 16`
`Message id = 13, recived_data=17 , original_data = 17`
`Message id = 14, recived_data=18 , original_data = 18`
`Time stamp=72389564`
	Step 3:
	`Message id = 0, recived_data=5 , original_data = 5`
	`Message id = 1, recived_data=6 , original_data = 6`
	`Message id = 2, recived_data=7 , original_data = 7`
	`Message id = 3, recived_data=8 , original_data = 8`
	`Message id = 4, recived_data=9 , original_data = 9`
	`Message id = 5, recived_data=10 , original_data = 10`
	`Message id = 6, recived_data=11 , original_data = 11`
	`Message id = 7, recived_data=12 , original_data = 12`
	`Message id = 8, recived_data=13 , original_data = 13`
	`Message id = 9, recived_data=14 , original_data = 14`
	`Message id = 10, recived_data=15 , original_data = 15`
	`Message id = 11, recived_data=16 , original_data = 16`
	`Message id = 12, recived_data=17 , original_data = 17`
	`Message id = 13, recived_data=18 , original_data = 18`
	`Message id = 14, recived_data=19 , original_data = 19`
	`Message id = 15, recived_data=20 , original_data = 20`
	`Message id = 16, recived_data=20 , original_data = 21`
	`Message id = 17, recived_data=20 , original_data = 22`
	`Message id = 18, recived_data=20 , original_data = 23`
	`Message id = 19, recived_data=20 , original_data = 24`
	`Message id = 20, recived_data=20 , original_data = 25`
	`Message id = 21, recived_data=20 , original_data = 26`
	`Message id = 22, recived_data=20 , original_data = 27`
	`Message id = 23, recived_data=20 , original_data = 28`
	`Message id = 24, recived_data=20 , original_data = 29`
	`Message id = 25, recived_data=20 , original_data = 30`
	`Message id = 26, recived_data=20 , original_data = 31`
	`Message id = 27, recived_data=20 , original_data = 32`
	`Message id = 28, recived_data=20 , original_data = 33`
	`Message id = 29, recived_data=20 , original_data = 34`
	`Message id = 30, recived_data=20 , original_data = 35`
	`Message id = 31, recived_data=20 , original_data = 36`
	`Message id = 32, recived_data=20 , original_data = 37`
	`Message id = 33, recived_data=20 , original_data = 38`
	`Message id = 34, recived_data=20 , original_data = 39`
	`Message id = 35, recived_data=20 , original_data = 40`
	`Message id = 36, recived_data=20 , original_data = 41`
	`Message id = 37, recived_data=20 , original_data = 42`
	`Message id = 38, recived_data=20 , original_data = 43`
	`Message id = 39, recived_data=20 , original_data = 44`
	`Message id = 40, recived_data=20 , original_data = 45`
	`Message id = 41, recived_data=20 , original_data = 46`
	`Message id = 42, recived_data=20 , original_data = 47`
	`Message id = 43, recived_data=20 , original_data = 48`
	`Message id = 44, recived_data=20 , original_data = 49`
	`Message id = 45, recived_data=20 , original_data = 50`
	`Message id = 46, recived_data=20 , original_data = 51`
	`Message id = 47, recived_data=20 , original_data = 52`
	`Message id = 48, recived_data=20 , original_data = 53`
	`Message id = 49, recived_data=20 , original_data = 54`
	`Time stamp=20`

Success criteria	
Check: - to receive all messages on destination NI - all messages on sending NI	
Test observations	
Test configuration	Config: "ttcommsched.cfg", "port.cfg", "hw.cfg"
Test conditions	Vitis for FPGA configuration, ATTNoC running on the board, all configuration files at github link https://github.com/project-fractal/WP6T62-HW-Edge-Controller, max queue length for a port = 16
Remarks	
Test result	
Passed	

Table 120 - Validation Test T04_WP6T62-0X (overview)

Validation Test	
Test ID	T05_WP6T62-0X
Test type	Functional
Test name	Testing the serialization services
Date	03/02/2022
Tester's Name	Luca Visconti (Akkodis)
Test scope or objective	
The objective of this test is to validate the serialization service on NGW out port	
Steps	
Step 1	Define a set of messages to send (1 message at 1 port according to the scheduling configuration TTCommScheld.cfg of NI3) - One message to NI3 on port 2

Port ID	Instant	Next	Instant (microsec)
2[NI3-NI2]	1502 (11)	0	44,759

Step 2: Send message at one port

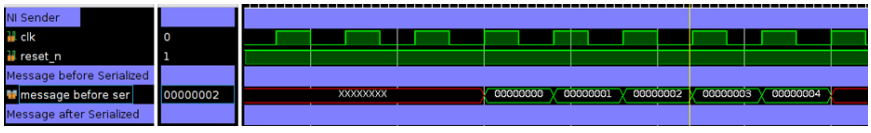
```
// send message from NI3 , port2
 INIT_AXI_TXN=1;

 PORT_ID_WR =2;
 INPUT_WDATA=0;
   #48
    INIT_AXI_TXN=0;
for(integer j=0;j<16;j=j+1)//f
   begin
   #8;
   INPUT_WDATA =j;
   end
```

Results/Evidence

Step 2:

Name	Value
clk	1
reset_n	1
enq	0
din[31:0]	0000001c
full	0
deq	0
dout[31:0]	UUUUUUUU
msglen[9:0]	005
empty	0
buffer_inst[0:511][31:0]	00000000,00000001,00000002,00000003,00000004,UUUUUUUU,UUUUUUUU,UU
wrpntr[9:0]	005
rdpntr[9:0]	000
full_loc	0
empty_loc	0
nqd[9:0]	005
AddrWidth	9
WordWidth	32
C_MAX_NUM_ELEMENTS[9:0]	200



Success criteria	
Check: - the message from the NI sender on the port 2 - the serialized message (flits in ordered)	
Test observations	
Test configuration	Config: "ttcommsched.cfg", "port.cfg", "hw.cfg"
Test conditions	Vitis for FPGA configuration, ATTNoC running on the board, all configuration files at github link https://github.com/project-fractal/WP6T62-HW-Edge-Controller
Remarks	
Test result	
Passed	

Table 121 - Validation Test T05_WP6T62-0X (overview)