A Q M O

Connecting Europe Facility 2014-2020

AQMO

Air Quality and MObility

Grant Agreement Number: INEA/CEF/ICT/A2017/1566962

2017-FR-IA-0176

D5.1

Workflow Management System

FINAL

Version:	2.0, 27/09/2019
Author(s):	Yiannis Georgiou, RYAX
Date:	27/09/2019

Project and Deliverable Information Sheet

AQMO Project	**Project Ref. №:** INEA/CEF/ICT/A2017/1566962	
	Project Title: Air Quality and mobility	
	Project Web Site: http://aqmo.irisa.fr/	
	Deliverable ID: D5.1	
	Dissemination Level: CO	**Contractual Date of Delivery:** 31 / 08 / 2019
		Actual Date of Delivery: 27 / 09 / 2019
	EC Project Officer: Mark VELLA MUSKAT	
Authorship	**Written by:**	Yiannis Georgiou (RYAX)
	Contributors:	
	Reviewed by:	François Bodin (UR1) Benjamin Depardon (UCit)
	Approved by:	Technical and Management boards

* - The dissemination level are indicated as follows: **PU** – Public, **CO** – Confidential, only for members of the consortium (including the Commission Services) **CL** – Classified, as referred to in Commission Decision 2991/844/EC.

Document Status Sheet

Version	Date	Status	Comments
0.8	12/08/2019	Draft V1	BD: document plan
1.0	22/09/2019	V1.0	Final
2.0	27/09/2019	V2.0	Approved

Tables

Table of Contents

Table of Figures

Executive Summary

Workflow Management Systems are responsible for automating the orchestration of groups of tasks upon computational resources. They usually rely upon Resource Management and Orchestration Systems which perform the actual job of resource allocation, task deployment and task life-cycle control.

AQMO platform has specific needs in terms of workflow management since it implicates execution of data analytics on different computational domains such as Edge, Fog and Cloud/HPC. This report analyzes the architecture, configuration and usage of the workflow management system Ryax which is used in the context of AQMO.

The report describes the different challenges that Ryax has to deal with related to hybrid Edge/HPC executions and provides different examples of workflows needing automation in AQMO. It analyzes Ryax internals and architecture while providing solutions to address the various AQMO issues.

1 Introduction

The general objective of the AQMO platform is to provide a service for measuring and simulating air quality in the Rennes Metropolis in France. In particular, it will serve citizens and policy makers with tools to provide air quality monitoring and insights for decision-making with reliable information and rational forecasts.

On the technical side AQMO is composed of a hybrid infrastructure consisting of edge, fog and High Performance Computing (HPC)/cloud domains of computational resources: Initially the air quality data are collected by sensors upon different buses and drones (edge) circulating in the city, they are aggregated on centralized servers (fog) before being sent for processing on HPC centers (on-premises or Cloud based) where the heavy lifting simulations take place to extract adapted forecasts and insights. On each different domain where compute power is available (edge/fog/cloud), processing may take place to perform various light or heavy data manipulations. Furthermore, data movement and aggregations can take place in different phases among computational domains. Automating the orchestration of data analytics tasks like the previous ones, demands the involvement of a specialized software stack combining processes such as resource allocation, load balancing, fault tolerance and controlled network communications.

The Workflow Management System (WMS) is responsible for automating the orchestration of task collections upon computational resources. The WMS usually relies upon a Resource Management and Orchestration System (RMOS) which will handle the difficult job of actually executing the tasks on the resources. The specificities of AQMO platform such as the usage of a hybrid distributed computing infrastructure, the need for lightweight and low-power systems software stack for the edge layer computational resources (within buses and drones), the need for stream processing and that of secure network transfers widens the scope of traditional workflow management and orchestration systems.

In AQMO, the combined jobs of WMS and RMOS are performed by the proprietary solution Ryax whose functionalities and architecture are the subject of the present report and will be described in the remainder of this document.

2 State of the art

This section provides the state of the art on software systems and research similar to workflow management, resource management and orchestration upon distributed environments.

2.1 Workflow Management Systems

The Workflow Management system is responsible for the automation of orchestration and execution of task collections upon computational resources. A common pattern in scientific and cloud computing involves the execution of many computational and data manipulation tasks which are usually coupled i.e., output of one task used as input on another. Hence coordination is required to satisfy data dependencies. The workload of task execution is divided among the available distributed computational resources. Consequently this introduces further complexity related to processes such as load balancing, data storage, data transfer, tasks monitoring and fault-tolerance Automation of the aforementioned aspects of the orchestration process has led in the creation of workflow management systems.

A recent study by Deelman et al. [i] related to scientific workflow management analyses the current state of the art on these systems and provides future research challenges. On the HPC side there are some particular tools such as Taverna[ii], Pegasus[iii] and Makeflow[iv] that have been used in production since years with different maturity levels. Those systems generally share many similarities in their concepts. They all have one or more principal languages to program workflows and they provide connections towards specific resource management systems for the deployment part. They have been designed with scalability and fault tolerance in mind and their multiple years in production has allowed them to make a lot of progress on the interoperability part. Nevertheless their main focus is basically on the HPC and scientific workflows with none or minimum support of dynamic data analytics. On the other side systems such as Airflow [v], Luigi [vi] and Argo [vii] have been designed for Cloud applications and allow the usage of more flexible abstractions such as containerization and micro-services which makes them more flexible and more adapted for data analytics. Nevertheless those systems can principally address batch processing operations with no facilitations for streaming operations.

Dealing with Streaming data is a need of new applications and use cases requiring reactivity and flexibility. Systems using Internet of Things (IoT) and Edge Computing will need to support stream processing to better cope with the challenges of new technologies and applications. Software solutions such as Beam[viii] and Google Cloud Dataflow [ix] have been designed to provide unified model for batch and stream data processing with facilitation in the expression of Big Data workflows using higher level abstraction. They integrate seamlessly with specialized data processing frameworks such as Spark[x], Flink[xi] and others on which they delegate the difficult task of runtime.

Those systems are ideal for Cloud Computing infrastructures with powerful computing characteristics but their design choices such as the choice of programming language does not make them adaptable for use cases implicating Edge Computing and IoT. Indeed, their choice of Java and Scala as base language makes them heavyweight for constrained computing power infrastructures and their design does not support network intermittence which is an important challenge for various edge computing use cases.

The workflow management system used within AQMO project is Ryax which considers the current state of the art and goes beyond since it is specifically designed to deal with both batch and streaming operations along with the challenges of Edge Computing and IoT use cases. In addition it provides high-level abstractions which specializes in the programming of data analytics workflows based on intuitive human logic and not complex computational algorithms.

2.2 Resource Management and Orchestration Systems

Resource management and Orchestration holds a very important place in the software stack of distributed systems since it is responsible for providing the necessary compute power to user jobs based on their needs and the resources availabilities.

The advent of Cloud and Big Data systems along with the usage of microservices and containerization brought the needs of environment provisioning and auto-scaling. Hence, the management of applications' lifecycle orchestration became an integrated part of resource managers which are also known as orchestrators.

Older state-of-the-art HPC resource managers such as Slurm and PBSPro do not provide integrated support for environment provisioning and hence no orchestration is feasible. However, newer resource managers such as Mesos, Yarn and Kubernetes enable the deployment of containers and allow the applications' lifecycle management.

This study[xii] upon orchestrators discusses the various advances that have been made regarding scheduling. Kubernetes[xiii][xiv] and Mesos[xv] are two of the most advanced open-source orchestrators. Kubernetes is probably the one that has been more widely adopted, it has a rapidly growing community and ecosystem[xvi] with plenty of platforms being developed upon it.

Kubernetes simplifies the deployment and management of containerized applications. It is based on a highly modular architecture which abstracts the underlying infrastructure and allows internal customizations such as deployment of different software defined networking or storage solutions. It supports various Big Data frameworks such as Hadoop MapReduce, Spark and Kafka and has a powerful set of tools to express the application lifecycle considering parameterized redeployment in case of failures, auto-scaling, state management, etc. Furthermore, it provides advanced scheduling capabilities and the possibility to express different schedulers per job.

However, our requirements in terms of orchestration have grown larger and as this recent study[xvii] presents we are still early in reaching our objectives. Orchestrators are just starting to integrate the edge and serverless functions but still effort is needed for supporting resources disaggregation at the edge, simplified data management and abstraction mechanisms for administrators and developers.

Even if Kubernetes is today production-level for typical cloud data centres it is not adapted for the constrained edge capabilities nor for multi-cluster deployments, such as integrating different layers of compute resources (edge, fog and cloud).

Efforts are currently ongoing to better adapt Kubernetes for the edge. Open-source solutions such as k3s[xviii] where Kubernetes heavyweight internal procedures have been stripped down are more adapted to our needs.

Concerning multi-clustering, the multi-cluster special interest group (SIG) community of Kubernetes works on the federation v2 project[xix] which focuses on integrating multiple clusters under a federation while providing a generic scheduling engine that, based on policies, is able to make decisions on how to place arbitrary Kubernetes Application

Co-financed by the Connecting Europe Facility of the European Union

Programming Interface (API) objects. In the same lines other efforts such as specific networking service meshes[xx] and multi-cluster schedulers[xxi] for Kubernetes provide promising solutions but their adaptation in our contexts needs to be confirmed.

In AQMO the workflow management system Ryax adopts Kubernetes orchestrator and resource management. In particular the lightweight K3S distribution for optimal usage on the Edge computing nature of the project. In addition the software defined networking solution of ZerotierOne[xxii] is used to cover the needs of security, authentication, flexibility and simplicity for the layer of network management. Different solutions for multi-clustering are currently under evaluation.

3 AQMO platform: hybrid infrastructure & workflow automations

This section describes the hybrid distributed computing infrastructure upon which AQMO platform is built along with the specificities and complexities that this brings in terms of resource management. In addition it presents some typical workflow automations needed in the context of AQMO.

3.1 Hybrid Distributed Computing Infrastructure

AQMO computational infrastructure is composed of different domains where the data analytics tasks of workflows can be executed. The list that follows provides each of the 3 domains along with their characteristics, requirements and usage:

i) The Edge Central Unit composed of low power and constrained capabilities compute platform is situated within each bus or drone and is physically in proximity with the sensors. The edge central unit is typically used for data pre-treatment and temporary storage. Besides the sensors collecting air quality data, other sensors may be connected to the compute platform such as noise sensors, cameras, etc which raises the need of providing secure ways to add new endpoints in a controlled plug&play manner. The compute platform on each bus or drone needs to function independently in terms of orchestration regardless the stability of the network connection with the rest of the fleet or the internet. Nevertheless, the fleet of edge central units has to be controlled and managed from a centralized point facilitating their configuration, monitoring and updates. The fleet of edge central units represents the edge domain. AQMO administrators and Central Unit operators have the rights to configure and update edge central units.

ii) The Fog Primary Server(s) composed of a typical computational platform is (are) situated somewhere within the same city (university, city hall, etc) where the buses circulate. The fog primary server(s) is (are) responsible for tasks such as permanent storage, raw data collection from buses, simulation dispatching to HPC centers, insights and reports creation based on the simulation results. They also play the role of centralized points where administrators can automate the configuration, monitoring and updates programmed for the fleet of edge central units. Methods to support the secure registration of edge central units in the fleet need to be proposed in a controlled plug&play way. The network connection from the primary server towards the HPC platforms and the internet is considered stable. AQMO administrators and Central Unit operators have the rights to configure and update edge central units. AQMO data scientists have the rights to submit simulations on HPC platforms and create reports using insights and forecasts from the simulations.

iii) The HPC platforms deployed on-premises (on an HPC center) or on Cloud (Amazon Web Service (AWS) or other) which are remote but powerful computing infrastructures destined for long batch simulations. The HPC platforms either on-premises or on Cloud are controlled by specific local resource management systems. AQMO platform passes through an HPC-as-a-Service mode to execute simulations on these domain. The on-premises HPC platfom grant no administrative control what-so-ever to AQMO administrators, operators or data-scientists. They only allow simulations submission, execution monitoring and results collection. On the other hand, in case of Cloud deployed HPC platforms AQMO stakeholders have more rights. In particular, besides submitting and monitoring executions, AQMO administrators, operators and data-scientists can configure the HPC cluster based on their needs prior to launching their simulation.

3.2 Workflow Automations

There are basically 3 different processing areas where data analytics can take place. The edge within the buses or drones directly where data are collected, the fog layer or primary server where data from all buses and drones are aggregated along with other tasks such as reporting, visualizations or group reconfiguration of buses and finally the Cloud or the HPC centers. Workflows of data analytics need to be created, modified, automated and deployed upon the edge, the primary server and the HPC platform or even on all of them, seamlessly. Here are some of the principal workflows that are needed to be deployed on each compute domain individually or combined with other domains:

3.2.1 *Edge layer workflows*

The workflows deployed at the edge within the buses or the drones are the ones related with the initial processing of the raw data collected from the different sensors integrated upon the AQMO connected vehicles. Readers can refer to deliverable D2.1 for more information on the hardware and sensors characteristics. Here are some examples for the edge related workflows:

1. One or multiple cameras may be connected to the edge compute processing module (Intel Next Unit of Computing (NUC)) within the AQMO vehicles. Images captured are transferred with Message Queuing Telemetry Transport (MQTT) protocol from the camera to the Intel NUC. When transferred each image needs to pass from an on-the-fly data real-time object detection analysis (like YOLO[xxiii])

Co-financed by the Connecting Europe Facility of the European Union

to contextualize the air quality measurements (realize if a bus follows another car or bus) and perform automated blurring to preserve data privacy.

2. Air quality and Global Positioning System (GPS) measurements captured from specialized sensors are transferred with MQTT protocol to the Intel NUC every second. Besides, other sensors for environmental measures may be added in the future (such as noise). In order to better manage the limited storage facilities of the Intel NUC a need to periodically compress data has appeared. Hence another workflow automation is: Every X minutes or Y hours compress collected data from sensors and remove raw data.

3. AQMO vehicles are not permanently connected to the Internet. There is at least one moment per day that the buses may connect to the Internet through WiFi without intermittence and that is when they are out of duty parked for the night. Other moments during the day 3G/4G or even WiFi may function depending on their localization but the connection is intermittent. A workflow automation need is: Whenever Internet connection is established send data that have not yet been sent to the primary server and remove data from the Intel NUC when transfer has been acknowledged.

3.2.2 Fog (primary server) layer workflows

The workflows deployed at the primary server (fog layer) are related to tasks around grouping of data or controlling workflows on multiple edge modules or programming the deployment of workflows on the Cloud and HPC. Here are some examples for the fog related workflows:

1. Data transferred from buses on a specific directory need to be cleaned and stored within a database (such as MongoDB) for easy retrieval and querying. Hence, the workflow automation need is: Periodically access the directory where measurement data are gathered, clean data using specific custom data manipulation techniques and store data within the MongoDB deployed on server.

2. The data stored within the database will be used with decreasing frequency considering their collection date hence particular attention should be given on how and where older data should be stored. A workflow automation may do the following: Every month move 1 month old data from hot DB (MongoDB) on warm Cloud storage (Amazon S3) and move 1 month old data from Cloud storage (Amazon S3) on cold Cloud storage (Amazon Glacier).

3. Perform group configuration (or reconfiguration) of the workflows deployed on the edge modules. Instead of waiting the connection of each bus on the network program or update a specific workflow for example the one that periodically compresses collected data and deploy it on each bus. In this case the workflow automation will be: When a bus is connected to the network deploy the new workflow and remove the older one if it exists.

Co-financed by the Connecting Europe Facility of the European Union

4. Submit an HPC simulation using specific data on an on-premises HPC platform. The workflow automation may be: Upon request, extract data related to a selected geographical and temporal region and upload them to a selected HPC Platform. Then launch simulation using these data, wait for the end of the simulation collect results, compress them and collect them on the Primary Server.

5. Submit an HPC simulation on a Cloud deployed HPC platform. The workflow automation may be: Based on the needs for execution deploy the right size of an HPC platform on the Cloud and once this is ready transfer data and launch the simulation. Then wait for the end of the simulation collect data back to Primary Server and destroy the deployed HPC platform.

3.2.3 HPC - Cloud layer workflows

The workflows deployed on HPC platforms are not directly controlled by Ryax - the AQMO workflow management system, like the workflows in the fog and edge domains. Hence only actions related to the ones allowed by the intermediate HPC-as-a-Service software can be performed. Here are some examples for the HPC-Cloud related workflows:

1. In the context of urgent-computing launch multiple simulations with the same data but with variations in their configuration. When each simulation is finished transfer data to Primary Server.

2. Combine simulations between them for a complete view. Wait for the end of one simulation before launching the next one. When all of the simulations are finished transfer all data to Primary Server.

3.2.4 Urgent-computing workflows

The workflows related with urgent-computing have the particularity to be high priority hence they are more likely to combine all 3 computational domains (edge-fog-cloud). Here is an example for the urgent-computing workflows

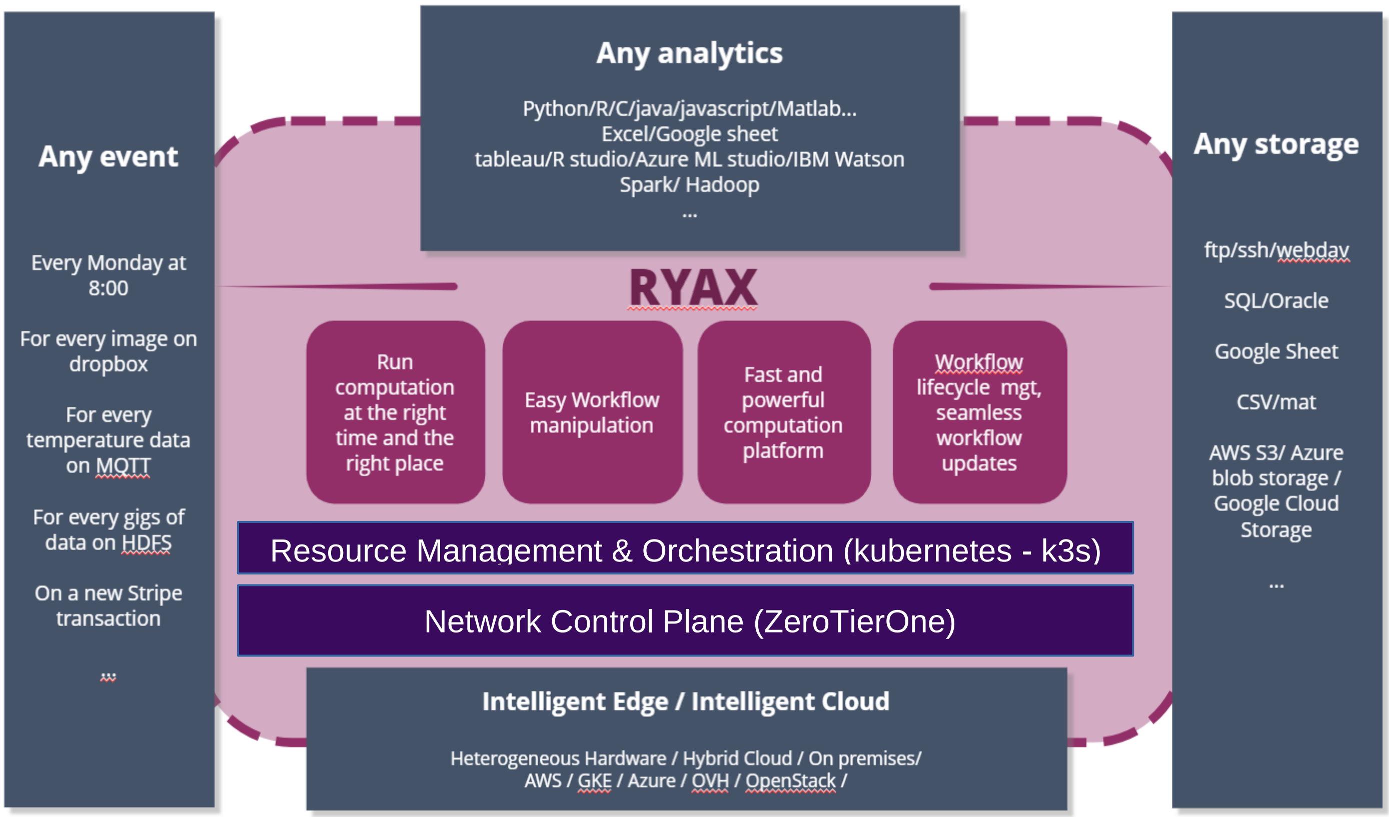
1. Workflow that combines all 3 edge-fog-cloud domains. In the context of urgent-computing when data are collected on drones they are transferred to the Primary Server which are then forwarded directly to the HPC platform where multiple simulations are triggered, to have a result fast. Then results are collected on the Primary server and reports are created featuring insights and forecasts calculated with the simulations. Since more collected data arrive continuously, new simulations are launched automatically and eventually new reports are created on-the-fly based on the updated forecasts.

4 AQMO Workflow Management Design

Based on the description of the underlying hybrid distributed computing infrastructure of AQMO platform, its characteristics and requirements along with the needs in terms of workflow automations, presented in the previous section; this section presents the high-level view of the AQMO Workflow Management as designed to address the aforementioned specificities of AQMO platform. We initially provide a high-level overview of Ryax Workflow Management System and then we describe how Ryax will help addressing the different AQMO's requirements.

4.1 High level view of Ryax Workflow Management System

Ryax solution developed by Ryax Technologies provides the lower-level resource management and high-level workflow automation building blocks of AQMO. Ryax data engineering platform provides the means to create, deploy, update and monitor workflows of data analytics while setting up the secure and efficient connection of the computation domains in a way that data analytics workflows can be deployed on different levels of the hybrid distributed infrastructure.



Any event
Every Monday at 8:00
For every image on dropbox
For every temperature data on MQTT
For every gigs of data on HDFS
On a new Stripe transaction
...
Any analytics
Python/R/C/java/javascript/Matlab...
Excel/Google sheet
tableau/R studio/Azure ML studio/IBM Watson
Spark/ Hadoop
...
RYAX
Run computation at the right time and the right place
Easy Workflow manipulation
Fast and powerful computation platform
Workflow lifecycle mgt, seamless workflow updates
Resource Management & Orchestration (kubernetes - k3s)
Network Control Plane (ZeroTierOne)
Intelligent Edge / Intelligent Cloud
Heterogeneous Hardware / Hybrid Cloud / On premises/
AWS / GKE / Azure / OVH / OpenStack /
Any storage
ftp/ssh/webdav
SQL/Oracle
Google Sheet
CSV/mat
AWS S3/ Azure blob storage / Google Cloud Storage
...

Figure 1: General Overview of Ryax Workflow Management System

As Figure 1 depicts Ryax can handle different types of events as input, various types of storage as output while providing any type of integrations for the analytics part.

Ryax high-level architecture is composed of 3 layers, as represented in Figure 2, the Network, the Resource and the Workflow management layers.

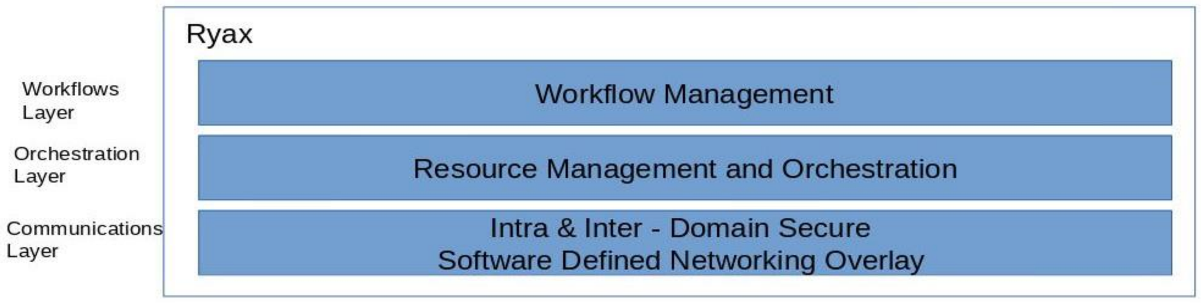

Ryax
Workflows Layer
Workflow Management
Orchestration Layer
Resource Management and Orchestration
Communications Layer
Intra & Inter - Domain Secure
Software Defined Networking Overlay

Figure 2: Principal layers of Ryax solution

Network Control Plane

The Network Management layer in Ryax is implemented based on ZeroTierOne open-source software. In particular, Ryax implements Software Defined Networking mechanisms through a ZerotierOne integration regardless the type of connection being used (3G, 4G, Wifi or Ethernet). Ryax allows secure authentication and authorization mechanisms of different compute resources along with the support of same virtual IP usage even when the connection passes through a different network. In addition, ZeroTier and consequently Ryax performs Network Address Translation and firewall traversal transparently while providing encryption and compression of data by default.

Resource Management & Orchestration

Based on that low-level networking mechanisms, Ryax provides the secure networking overlay necessary for the resource management tasks. For Resource Management, Ryax is based on Kubernetes resource manager and container orchestration platform and in particular upon k3s lightweight open-source distribution of Kubernetes. Kubernetes is used for its microservices architecture and modular design to manage a distributed computational system along with its abstraction capabilities and declarative way to describe resources.

The choice of k3s distribution from Kubernetes has been made in order to have a more lightweight middleware and to be able to deal with the autonomy of edge clusters. In k3s the Kubernetes heavyweight internal procedures (such as the use of Docker Container Runtime Interface (CRI) which is replaced by the more lightweight containerd CRI) have been stripped down and are more adapted to constrained edge hardware. In our architecture each bus will be equipped with specialized processing unit (Intel NUC) which will be used for the pre-treatment of some collected data.

Data Analytics Workflow Management

Ryax provides the necessary tools to create workflows with high level abstractions and descriptive approaches, while offering the adequate methods to distribute the processing of data analytics across the resources of the computational platform along with the support of data treatment in various forms (batch or stream).

The goal is to provide the means to enable the development of data analytics workflows following the principles of human thinking, rather than complex algorithms. Ryax enables a workflow to be expressed by a group of data manipulation functions which will act upon a flow of data defined by at least one input and one output.

Ryax processing engine is based on a serverless architecture that allows applications to be composed of a group of interdependent functions whose execution can be distributed on different resources in a flexible and dynamic way. In parallel it decorrelates the complex task of functions deployment and execution lifecycle with the actual development of their logic and allows functions to be entirely managed by the platform offering features such as autoscaling and fault-tolerance.

Ryax provides a unified batch and stream processing engine offering the users the capabilities to take into account different forms of data: either group of cold data such as a directory of images (batch) or flow of hot data such as real-time sensor data (stream).

4.2 Addressing AQMO's Requirements with Ryax Workflow Management System functionalities

We enumerate some of the most important particularities of AQMO platform and we present how the Workflow Management architecture deals with them:

AQMO Requirement 1. The multiple independent autonomous processing points (Edge Central Units within buses and drones) are composed of low processing power hardware (Intel NUC) and can collect data directly from sensors or through intermediate gateways (Rasberry Pi connected to a camera). Each Edge Central Unit needs to have the ability to orchestrate its own workflows and there is no possibility to manage the Edge Central Unit from the Cloud because the internet connection is intermittent.

WMS solution 1. Each Edge Central Unit will execute its own workload management system Ryax in order to manage and orchestrate its local workflows autonomously. Ryax provides the right abstractions to orchestrate workflows and manage the computational resources at the edge relying upon Kubernetes orchestrator. Ryax is specifically designed to function on constrained processing power edge computational units so it makes use of a lightweight distribution of Kubernetes, the open-source software k3s. K3s, and consequently Ryax, can provide the resource management even on single node clusters (one master with no agents) which is the typical case of the Edge Central Unit (running on Intel NUC). Nevertheless, the flexibility of the Edge Central Unit to allow the connection of any types of sensors including those that need a gateway with processing power (intelligent camera connected with Raspberry Pi) raises the need of managing clusters with processing power on multiple nodes. K3s and by extension Ryax can enable the connection of additional computing nodes on-the-fly as long as the new nodes are connected in the same flat TCP/IP[1] network with the master. Then it can distribute the workload on the various nodes that participate in the cluster.

AQMO Requirement 2. The Fog Primary Server will have permanent Wifi connection, it will be situated somewhere centrally in the city and it will be composed of a single powerful computer (with possibility to have some more similar nodes to form a cluster if needed). The Fog Primary Server will be used as the central location to work with AQMO platform. A user connected to the Fog Primary Server will have the capabilities to submit executions in the form of workflows to be deployed on one or more computational domains (Edge, Fog or HPC/Cloud), submit configurations for the Edge Central Units, view reports and visualizations.

WMS Solution 2. The Fog Primary Server will execute its own workflow and resource management system, in order to enable the execution of workflows autonomously. For this the Fog Primary Server will make use of Ryax based on K3s Kubernetes distribution, just like the Edge Central Units,. The offloading of workflows to the Edge and the configuration of Edge Central Units from a centralized point will make use of the capabilities of Ryax to support multi-clustering where the k3s master of the Fog Primary Server will have the ability to submit workflows on one or multiple k3s masters in a federated way. The offloading of workflows and executions of simulations destined for the HPC resources will pass through the support of specific API part of the HPC as a Service software described in detail in the deliverable D4.1.

AQMO Requirement 3. The multiple Edge Central Units occasionally (at least once per day) need to transfer the raw or pre-treated data coming from the different sensors to the higher-level computational domain, the Fog Primary Server. For this each Edge Central Unit needs to be able to communicate with the Fog Primary Server participating in the same flat and secure TCP/IP network. Furthermore, different modes have to be supported such as Wifi & 3G/4G and the intermittence of the connection needs to be handled as efficiently as possible. In case the Edge Central Unit is composed internally of a cluster of multiple nodes the same rules of flat and secure TCP/IP network apply.

WMS solution 3. Ryax, AQMO's workflow management system is equipped with a Secure Software Defined Networking overlay based on the ZeroTierOne software. This open-source networking solution enables the communication of different computational resources on the same flat and secure TCP/IP network bypassing firewalls automatically and providing encryption of data by default. ZeroTierOne allows the authorization of each device demanding permission to participate in the network through integrated centralized techniques. On activation, it attributes a virtual network interface (besides eth0) and a unique external IP address which both will remain intact even if the device changes from Wifi to 3G/4G or vice versa. Intermittence of networks is handled with ZeroTierOne and once the network connection is returned the same network interface and IP address will be used. In the context of AQMO we will define one virtual network overlay for the inter-clusters communications between Edge Central Units and the Fog Primary Server. Furthermore in case an Edge Central Unit or Fog Primary Server makes use of multiple nodes per cluster then each new networked

[1] Transmission Control Protocol/Internet Protocol

Co-financed by the Connecting Europe Facility of the European Union

computational device will join the same virtual network which will be used for both intra and inter clusters communications.

AQMO Requirement 4. AQMO project needs to have a way to create workflows that can be executed seamlessly on all different computational domains (Edge, Fog, HPC/Cloud). Workflows need to be expressed with a high level of abstraction and using the same tools and methods regardless of where they are going to be executed. A workflow should be expressed based on microservices architecture style which is easier to manage on a highly distributed computing infrastructure such as AQMO. The workflow expression should facilitate the usage of streaming data and the support of event processing besides the classical batch executions. For example, a periodic arrival of data coming from a sensor on the Edge Central Unit should trigger the continuity of the workflow which may take place on the Edge Central Unit or on other computational domains. A command line utility to create workflows is primarily needed. A web interface to create workflows to facilitate users with simple to use interface will be interesting to have.

WMS solution 4. Ryax workflow management system supports workflow description through a programming model with high level abstractions based on the high-level language YAML. The programming model is based on serverless architecture which allows to decompose an application into small functions that can connect among each other with dependencies and can be executed autonomously on different computational resources, if needed. Furthermore, functions may be written in various general-purpose programming languages. Ryax implements a unified batch and stream processing engine which means that both streaming data and batch executions are supported. The creation of workflows can be done either using YAML files programmatically along with a command line interface to manage workflows and functions or through a web interface which allows a design based on intuitive tools using data-flow view.

5 Ryax Workflow Management System Architecture, Internals and Usage

This section provides a detailed description of Ryax Workflow Management System analyzing its layers in more depth.

5.1 Software Defined Networking

This section describes the communications layer of Ryax Workflow Management System. Ryax offers a secure software defined networking (SDN) overlay which allows different nodes on different physical networks to participate on the same virtual network and exchange data in a secure way.

For this Ryax is based on the ZeroTierOne open-source software which is a distributed network hypervisor built on top of a cryptographically secure global peer to peer

Co-financed by the Connecting Europe Facility of the European Union

network. It provides advanced network virtualization and management capabilities in addition to an enterprise SDN switch that functions across both local and wide area networks and enabling the communication of almost all networked devices, VMs, containers, and applications as if they all reside in the same physical data center or cloud region.

Eventually there will be a tight integration between Ryax and ZeroTierOne but for the current version the SDN Overlay layer needs to be deployed independently in advance. Once this is done we continue with the deployment of the Resource Management and Orchestration layer.

To build a Zerotier Virtual Private Network (VPN) we need to select at least one controller of the network which will hold all the needed information regarding the network details, memberships authorizations, IP address ranges, etc. In the context of AQMO there will be one virtual network for the communication of all Edge Central Units of all buses and drones with the Fog Primary Server hence the network controller needs to be kept within the Fog Primary Server since the Edge Central Units will be rarely online. The ZeroTierOne controller does not provide a single point of failure, at least not for communications, meaning that 2 Edge Central Units participating in the same virtual network will have the ability to communicate even when the controller is not available. However, without the controller no new authorizations will be possible until the controller is back.

Here is the procedure to follow in order to build the ZeroTierOne VPN to be used among the Fog Primary Server and the Edge Central Units.

1) Each node participating in the same network needs to have ZeroTierOne installed either by downloading the right binaries or by building the sources:

```
$git clone https://github.com/zerotier/ZeroTierOne.git
$cd ZeroTierOne
$git checkout dev
$make
```

2)Once ZeroTierOne is built it needs to be started in background as root, on each node:

```
$sudo ./zerotier-one -d
```

3)Then we need to select one of the nodes to become the controller of the ZerotierOne network. In our case this is the Fog Primary Server. We configure the network using the “ryax-zero-conf” script developed and offered with Ryax packages to cover the networking configuration needs. Hence we connect on the Fog Primary Server and we initially create a new network with the following command:

```
$sudo ./ryax-zero-conf --net-add
```

This command will return a json output with the different details of the new VPN. In the end it will show something like:

```
"id": "9d32babae01bc203"
```

This is the "id" of the newly created VPN.

To get all the information related to a particular network with id=9d32babae01bc203 we can use the following command:

```
"$sudo ./ryax-zero-conf --net-info -n 9d32babae01bc203
```

The following command can give the information about all the networks on which this node is a controller.

```
"$sudo ./ryax-zero-conf --net-list
```

4)Then we create the IP address ranges for this network using the following command:

```
$sudo ./ryax-zero-conf --net-ipadd 10.147.15.0/24 -n 9d32babae01bc203
```

5)Now that the network is created we have to connect each Edge Central Unit on the previously created VPN. To do that we connect on an Edge Central Unit and use the following command from ZeroTierOne software to join the previously created network:

```
$sudo ./zerotier-cli join 9d32babae01bc203
```

The following command will show us the list of networks that we are connected on the Edge Central Unit and the state of the connection

```
$sudo ./zerotier-cli listnetworks
```

Initially we will get something like:

```
200 listnetworks <nwid> <name> <mac> <status> <type> <dev> <ZT assigned ips>
200 listnetworks 9d32babae01bc203  02:81:c1:f6:1d:7e ACCESS_DENIED PRIVATE zt2 -
```

Which is normal because we need to authorize this Edge Central Unit node to participate on the VPN

6)To authorize a node to participate on a VPN we need to go on the controller and use the ryax-zero-conf script.

The following command shows us the list of nodes that have requested authorization for the VPN

```
$sudo ./ryax-zero-conf --member-list -n 9d32babae01bc203
```

We use the ID of each node as returned from the above command in order to authorize them using the following command:

```
$sudo ./ryax-zero-conf --member-auth -z 43da16a7c4 -n 9d32babae01bc203
```

and we assign a particular IP address from the initially defined network range using the following command:

```
$sudo ./ryax-zero-conf --member-ipadd 10.147.15.1 -z 43da16a7c4 -n 9d32babae01bc203
```

7)Then we go back on the Edge Central Unit node on which we authorized access and we check the status of the connection:

```
$sudo ./zerotier-cli listnetworks
```

and we should get something like

```
200 listnetworks <nwid> <name> <mac> <status> <type> <dev> <ZT assigned ips>
200 listnetworks 9d32babae01bc203  02:81:c1:f6:1d:7e OK PRIVATE zt2
10.147.15.204/24,10.147.15.1/24
```

while ifconfig should have a new ethernet interface with the above IP assigned

```
zt2      Link encap:Ethernet  HWaddr 02:81:c1:f6:1d:7e inet addr:10.147.15.1  Bcast:10.147.15.255
Mask:255.255.255.0
```

8)We do the following for all the Edge Central Units that are going to be connected in the Fog Primary Server VPN. Once that is finished we can use the newly created ZeroTier virtual interface for the Resource Management and Orchestration layer.

Based on the previous procedure we are sure that the authorized networked devices and only those are eligible to communicate in the VPN which guarantees security. Furthermore, whenever the Edge Central Unit will lose connection and come back again it will keep the same IP address (even if that is through another physical network interface, ie. Wifi instead of 4G) which guarantees flexibility and efficiency in communications.

In case an Edge Central Unit or the Fog Primary Server is a multi-node cluster (at least one additional computational device is connected to the principal master node) then each computational networked device that participates in the multi-node cluster will get a virtual network interface and an IP address by demanding authorization to join the virtual network. This ensures the security of the internal communications and the safe authorization of additional devices.

5.2 Resource Management and Orchestration

This section describes the resource management and orchestration layers of Ryax Workflow Management System. Ryax is based on the resource manager and container orchestration platform Kubernetes which is based on a microservices style architecture which is ideal for our distributed and highly dynamic computational environment.

In particular Ryax adopts the K3S distribution of Kubernetes which is a lightweight distribution specialized for edge and low power computational units. K3S offers the same services as Kubernetes which is resource allocation, load balancing, autoscaling and orchestration to site a few of its functionalities, with the difference that it does this with a lightweight environment. Of course this means that there is support for less features in comparison with the mainstream Kubernetes but the selection of which ones to keep was optimal for edge and minimal environments so we can find more or less all the most important ones. The basic changes with Kubernetes are that some Cloud, storage and alpha or non-default modules have been removed , in addition etcd has been replaced by a lightweight version sqlite. Docker has been removed from K3s and it is substituted by Containerd. Containerd is an Open Containers Initiative (OCI) standard compliant container runtime, which is part of Docker stack and responsible for the deployment tasks of Docker. So Containerd can deploy typical Docker images without the heavyweight tools for pushing and building images that Docker provides. Furthermore the whole platform has been wrapped in a simple launcher binary file that handles big part of the Transport Layer Security (TLS) complexity and various options. The result is that the binary is around 40Mega Bytes (MB) and it needs no more than 512MB of Random Access Memory (RAM) to be deployed. Additionally, it can be used to orchestrate clusters of both multiple nodes and simple mono-node machine (similarly to minikube but with production-level features). K3S can orchestrate functions by using either an online or a local offline registry. There is also the possibility to predownload needed images and deploy them when time comes facilitating cases where internet connection is not always available.

All the above features make K3S an ideal resource manager and orchestrator for edge-fog-cloud distributed environments and consequently for the AQMO project architecture. In the context of AQMO, as it has been previously mentioned, each Edge Central Unit will be equipped by one autonomous lightweight cluster managed by K3S. By default the cluster will be mono-node but in case more computing nodes need to be connected then a multi-node cluster will be deployed. The same will be applied for the Fog Primary Server regardless the fact that this computational unit will be more powerful and permanently connected to internet.

In the case we build a mono-node cluster for either the Edge Central Unit or the Fog Primary Server then the installation procedure of the orchestration layer is composed of simply downloading the latest version of k3s and deploying it with the following parameters:

```
$curl -sfL https://get.k3s.io
$k3s server –disable-agent --bind-address 10.147.17.126 --node-ip 10.147.17.126 --tls-san
10.147.17.126
```

The disable-agent flag will just deploy a master, with no workers, which will have the ability to execute and orchestrate tasks (or pods, as they are called in Kubernetes terms, which are groups of containers deployed on the same node) autonomously. The bind-address, the node-ip and the tls-san flags take as parameter the virtual IP address attributed to the master by ZeroTierOne and the procedure described in the previous section. In order to deploy a multi-node cluster we need to launch an agent on each computational unit with the following command, after downloading the same executable on each node.

```
$curl -sfL https://get.k3s.io
$k3s agent --server https://10.147.17.126:6443 --token ${NODE_TOKEN}
```

Where the server flag defines the virtual IP address of the k3s master as used by the k3s server command previously. When the server starts it creates a file /var/lib/rancher/k3s/server/node-token. Using the contents of that file as NODE_TOKEN will allow us to join each node with the master server of K3S.u

The above procedure repeated for each Edge Central Unit and the Fog Primary Server will enable us to obtain multiple mono or multi-node clusters with the advantage of having all nodes of all clusters on the same virtual, flat and private IP network that handles network intermittence.

K3s and in particular the command line kubectl has the ability to connect on different clusters by taking advantage of the “kubectl config use context” option. Following the examples here[xxiv] we can connect on the Fog Primary Server and by changing contexts we can execute, monitor or reconfigure different clusters remotely just by merging the contents of the kubeconfig file of all the Edge Central Units with that of the Fog Primary Server and then executing the following command

```
$kubectl use-context edge-central-unit-1
$kubectl use-context fog-central-unit
$kubectl use-context edge-central-unit-2
```

The first version of AQMO project release will provide the means to create an automated workflow that will wait for the internet connection to be established between an Edge Central Unit and the Fog Primary Server and then after changing the context it will automatically apply a configuration or launch a group of functions to be deployed directly on the remote cluster. In a later version of AQMO project release we will provide advanced multi-cluster features where the K3s of the Fog Primary Server will have the ability to federate executions on one or multiple clusters without the need to changing contexts. The distribution of tasks will take place dynamically. For this we are following project such as Cilium Container Networking Interface (CNI), Admiralty multi-clustering and Kubernetes Federation v2.

5.3 Workflows Management

This section describes the Workflows Management internals along with the usage of the workflow management layer.

Ryax is designed in such a way so that it can perform processing of both batch and streaming operations. This is done in order to optimally address use cases related to IoT and Edge Computing. In Ryax terms, a workflow can be expressed as a flow of data with at least one input, one output and intermediate functions performing data manipulations on the data-flow. Viewing and managing a workflow in such a way facilitates the handling of streaming data which is by default more complex than batch and enables the system to decompose the workflow into smaller interdependent functions which is more adapted for a distributed processing runtime.

A graphical representation of the Workflow Management Layer as developed within Ryax is provided by the figure 3. As represented in the figure 3, Ryax implements its workflow management around an API which is playing the role of communication means amongst different parts. In particular all the user interaction with the system which begins on the command line or web user interfaces pass through the API which are then communicated with the main system.

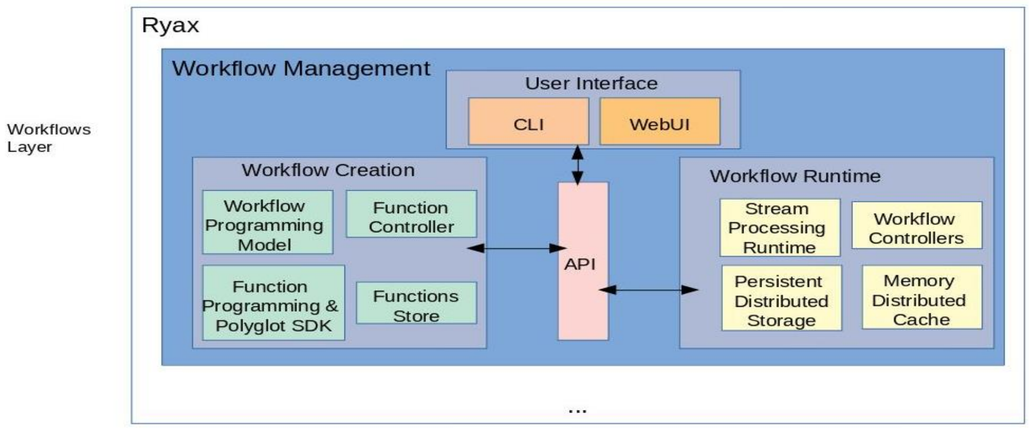


Figure 3: Ryax Workflow Management internal architecture

Ryax provides the workflow management based on a serverless architecture both on the workflow creation and the workflow runtime. Serverless architecture technology does not mean that there are not any servers for the application execution; but that the application developer will act as if there are not. This means that the underlying system handles all the issues related to application packaging and runtime relieving developers from the complexity to deal with them in their development.

Hence concerning workflow creation, Ryax programming model allows the expression of a workflow by implementing stand-alone functions, connected amongst them to convey a certain logic, whose code has no complexity whatsoever related to the distributed infrastructure to be executed. The only requirement is to decompose the logic into multiple steps (functions) which can be performed intuitively when one considers the dataflow view of the workflow. The workflow creation representation is described on the left of the figure. The Workflow programming model is based on YAML high-level language which is used to enable users to express the workflow logic taking advantage of the flexibility, simplicity and high level abstractions of YAML. The user can develop the intermediate functions in different programming languages leveraging on the poly-glot SDK offered by Ryax. Various general-purpose functions are already implemented and stored within the function store. All new implementations of functions can be stored there. This is the place where various external services integrations can be found and used directly for the creation of new workflows. Finally the Function controller is re-sponsible for packaging the function in a container and keep it in a registry so that it can be downloaded for execution when time comes.

As far as the workflow runtime is concerned (figure 3 at the right), Ryax is responsible to cope with all the complexities of executing on a hybrid edge-fog-cloud distributed computational environment and hide them from the users making execution transpar-ent. Ryax is developed in such a way that each packaged function is then orchestrated by the underlying K3S software. The workflow controller is responsible to spawn the execution and to manage the interaction among functions. The stream processing runtime is the means that handles the execution of the analytics upon the data by communicating with the underlying orchestrator K3S. The exchanges of functions metadata are controlled by a specialized in memory distributed cache and the ex-change of data are controlled by particular techniques similar with a persistent distrib-uted storage medium but with further optimizations.

Figure 4 shows the same Ryax internal architecture but when it is deployed on multiple computational resources. A part of the workflow runtime remains in the master such as the workflow controller which will handle the spawning of workflows and functions along with a memory and persistent storage facility to keep system and workflow re-lated information on the master node. On the worker node only the runtime is available which makes the system more lightweight. Of course in the case of mono-node cluster the master and worker share the same node.

```
$cat testmqttgw.yaml
apiVersion: "ryax.tech/v1alpha4"
kind: Workflows
spec:
  id: testmqttgw
  humanName: Test the MQTT Gateway.
  functions:
    - id: mqttgw
      type: mqttgw
      version: "1.0"
      inputs_values:
        mqtt_server: "mosqito.default"
        mqtt_topics: ['$SYS/broker/uptime', '$SYS/broker/load/#']
      streams_to: ["pushtoredis"]
    - id: pushtoredis
      type: pushtoredis
      version: "1.0"
      inputs_values:
        host: "redis.ryaxns"
        port: "6379"
        key: "testmqttgw"
        data: "=MQTTgw.test_input1"
      streams_to: []
```

The installation and usage of Ryax workflow management system are provided within the documentation of the software. However we provide here a high level view concerning the creation of workflow and usage of the platform. First of all we need to create our workflow by writing a yaml file with the main logic of it. The example here mentions a simple workflow that collects data from Mqtt and pushes them to Redis Key/Value store.

In the yaml we see that the workflow is decomposed in 2 functions one capturing data from an Mqtt server on a specific topic and the other one which stores data to redis when it receives something from Mqtt. The connection between the 2 functions is done with the parameter streams_to on the Mqtt function side and with the data parameter on the pushtoredis function to show which data to finally story on Redis.

Now that the general logic is created we need to create the logic of each individual function. To create a function we need to create a directory containing a yaml file (ryax_metadata.yaml) describing the function's inputs and outputs and some specific metadata; the source code of the function (handler.py) and its dependencies (requirements.txt) if there are. Depending on the programming language "*funclang*" of the function, some specificities apply.

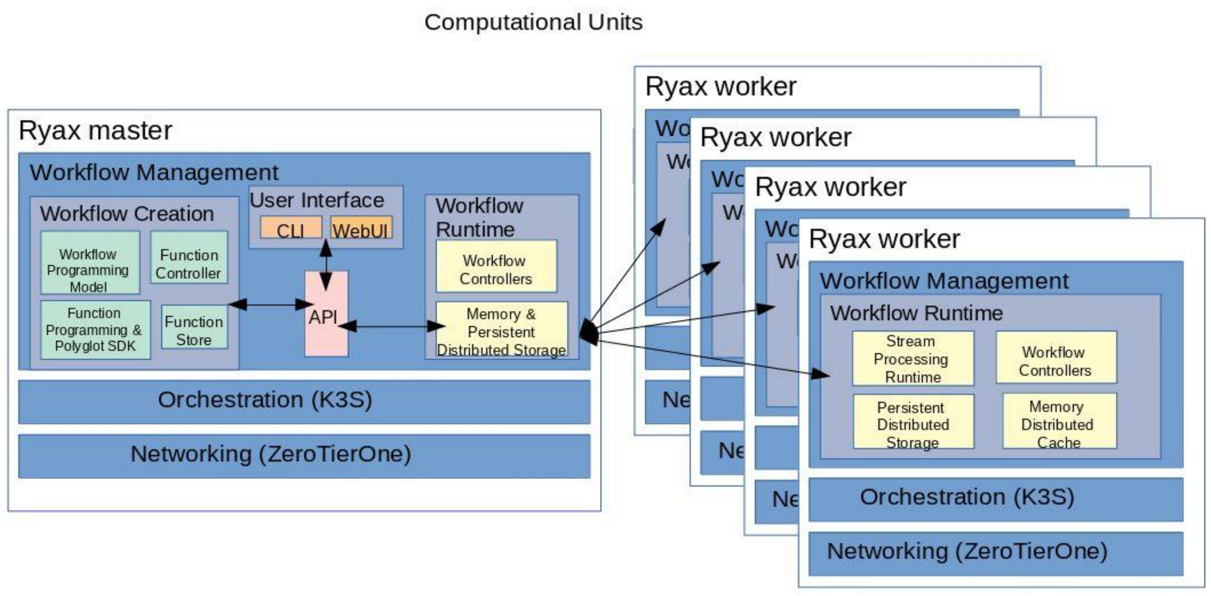


Figure 4: Ryax Workflow Management Internal Architecture view when deployed on a distributed computing infrastructure

The following code is related to the function Mqtt:

```
$ls mqttgw
requirements.txt  handler.py  ryax_metadata.yaml
$cat requirements.txt
hbmqtt
```

```
$cat ryax_metadata.yaml
apiVersion: "ryax.tech/v1alpha4"
kind: Gateways
spec:
  type: mqttgw
  humanName: MQTT Gateway.
  funclang: gateway
  version: "1.0"
  summary_template: "Get message from MQTT {topics} from the MQTT {server}"
  inputs:
  - help: MQTT server
    humanName: MQTT server to subscribe on
    name: mqtt_server
    type: string
  - help: MQTT topics
    humanName: MQTT topics to subscribe on
    name: mqtt_topics
    type: list[string]
  outputs:
  - help: MQTT message
    humanName: MQTT message data
    name: mqtt_message
    type: string
```

The code of handler.py for Mqtt gateway is provided in the annexes along with some more details related to the yaml parameters. Once the directory contains all the necessary files the following command needs to be executed on the terminal using the command line facility in order to create the function.

```
$ryax-cli function create path/to/your/function/directory
```

Once this is done for both functions declared in the workflow then we can start the workflow which can be deployed with the following command:

```
$ryax-cli workflow start testmqttgw
```

The workflow will be distributed on the underlying computational environment using the K3S orchestrator.

5.4 Ryax Workflow Management Architecture in the context of AQMO

The graphical representation of Ryax architecture as it is being used in the context of AQMO is provided by the following figure 5.

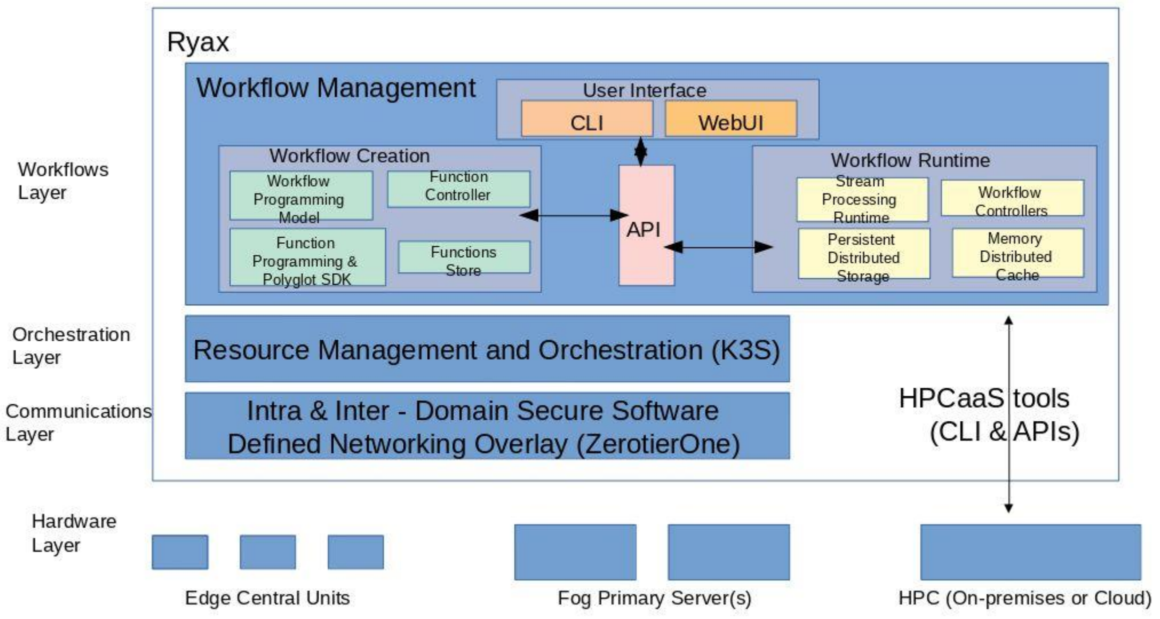

Ryax
Workflow Management
User Interface
CLI
WebUI
Workflows Layer
Workflow Creation
Workflow Programming Model
Function Controller
Function Programming & Polyglot SDK
Functions Store
API
Workflow Runtime
Stream Processing Runtime
Workflow Controllers
Persistent Distributed Storage
Memory Distributed Cache
Orchestration Layer
Resource Management and Orchestration (K3S)
Communications Layer
Intra & Inter - Domain Secure Software Defined Networking Overlay (ZerotierOne)
HPCaaS tools (CLI & APIs)
Hardware Layer
Edge Central Units
Fog Primary Server(s)
HPC (On-premises or Cloud)

Figure 5: Ryax Workflow, Orchestration and Communication layers as adapted in the context of AQMO

One of the principal things that is shown in this figure is the fact that the orchestration and networking management of Ryax are not used to control executions upon the HPC center. This is because the HPC center is controlled by a specific HPC resource manager which keeps the control of the computational units. However the workflow management system can command the execution of workflows through the HPC as a Service tools and in particular the EnginFrame API which can be used to trigger HPC simulations. For this to be possible workflows will have the capability to include EnginFrame calls from within their functions, hence special function type that integrates with EnginFrame will be provided within the Ryax Function store in order to facilitate the creation of workflows needing HPC simulations.

Most of the workflows will be distributed within single computational domains following the examples presented in section 3. However, in the case of urgent computing workflows will need to span all 3 computational domains at once meaning that functions part of the workflow will need to execute on the edge central units, others on the fog primary server and finally HPC simulations may need to be started. Of course since there is no actual control of the HPC resources, metadata or data can not be streamed on the fly, as it happens from Edge to Fog domains, but we can at least make use of the most recent updates of data and submit new simulations in high frequency.

6 Conclusion

The report presented the requirements and the challenges of Workflow Management for AQMO project along with the software solution that has been adopted to cover the specificities of the hybrid Edge/Fog/Cloud-HPC infrastructures and the automations needed in terms of workflows. The report provides an introduction to the new workflow management system Ryax which focuses on streaming operations and specializes on Edge Computing and seamless deployments on multi-computational domain environments. It analyzes its internals and provides ways to solve the different issues that have been raised in the context of AQMO.

7 Annexes

7.1 Detailed explanations related to the yaml specifications for the function creation

- apiVersion: gives the API version of the file. A given CLI will only support a given API version.
- kind: What kind of Ryax object this file is describing, in this guide we want to create a function.
- spec: a dictionary with all the fields specifying the function.
    - type: the unique identifier of the function.
    - detail: A human description of the function.
    - funclang: the "language" of the function. This can be a programming language (ex: python3), or something else (ex: docker)
    - version: The version of the function. Ideal to manage updates.
    - inputs: an array of objects describing inputs.
    - outputs: an array of objects describing outputs.

7.2 Detailed explanations regarding the function components for the particular case of funclang python3:

- the requirements.txt file that lists the dependencies will be run against a standard pip command.

- a handler.py file that run the code. This code MUST have a "handle" function (non-asynchronous), that takes a dict as input and returns a dict. The input (resp. output) dict will (resp. must) contain a key for each input (resp. output) with their values as value.

7.3 Code of handler.py for section 5.4

```
$cat handler.py
#!/usr/bin/env python3
# Copyright (c) Ryax Technologies

"""
Ryax MGTT gateway.

To test push to the gateway queue:

{"mqtt_server": "192.168.99.100:30883", "mqtt_topics": ["$SYS/broker/uptime", "$SYS/broker/load/#"] }

"""
import asyncio

import hbmqtt
from hbmqtt.client import MQTTClient
from hbmqtt.mqtt.constants import QOS_1
from ryax_common import get_logger

from ryax_gateway import RyaxGateway, ryax_gateway_main

logger = get_logger("GATEWAY_MQTT")


async def init_mqtt(mqtt_server):
    mqtt_client = MQTTClient()
    logger.info(f"MQTT server endpoint: {mqtt_server}")
    await mqtt_client.connect(f"mqtt://{mqtt_server}")
    return mqtt_client


class MQTTGateway(RyaxGateway):
    def validate_request_format(self, req_data):
        if "mqtt_server" not in req_data or "mqtt_topics" not in req_data:
            raise Exception(
                f"Malformed request, 'mqtt_topics' and 'mqtt_server' fields are required. Request {req_data}"
            )
    async def register_request(self, req_id):
        req_data = self.request_store[req_id]["request_data"]
        try:
            mqtt_client = await init_mqtt(req_data["mqtt_server"])
            # FIXME arbitrarly choose QOS_1 because... why not!
            mqtt_client.subscribe([(topic, QOS_1) for topic in req_data["mqtt_topics"]])
        except hbmqtt.client.ConnectException as err:
            raise Exception(
                f"Request {req_id} because the MQTT server "
    async def start_gateway_handler(self):
        while True:
            if len(self.request_store) == 0:
                await asyncio.sleep(10)
            for req_id, req_data in self.request_store.items():
                mqtt_client = req_data["mqtt_client"]
                message = await mqtt_client.deliver_message()
                packet = message.publish_packet
                data = str(packet.payload.data)
                logger.debug(f"Form post data: {data}")

                datatosend = {packet.variable_header.topic_name: data}
                await self.send_new_execution_event(req_id, datatosend)

async def cleanup(self):
        for req_id, req_data in self.request_store.items():
            mqtt_client = req_data["mqtt_client"]
            mqtt_topics = req_data["mqtt_topics"]
            await mqtt_client.unsubscribe(mqtt_topics)
            await mqtt_client.disconnect()

if __name__ == "__main__":
    ryax_gateway_main(MQTTGateway)
```

References

i Ewa Deelman, Tom Peterka, Ilkay Altintas, Christopher D. Carothers, Kerstin Kleesevan Dam, Kenneth Moreland, Manish Parashar, Lavanya Ramakrishnan, Michela Taufer, Jef frey S. Vetter: The future of scientific workflows. IJHPCA 32(1): 159-175 (2018)

ii Duncan Hull, Katy Wolstencroft, Robert Stevens, Carole A. Goble, Matthew R. Pocock, Peter Li, Tom Oinn: Taverna: a tool for building and running workflows of services. Nu cleic Acids Research 34(Web-Server-Issue): 729-732 (2006)

iii Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira da Silva, Miron Livny, R. Kent Wenger: Pegasus, a workflow management system for science automation. Future Generation Comp. Syst. 46: 17-35 (2015)

iv Casey Robinson, Douglas Thain: Automated packaging of bioinformatics workflows for portability and durability using makeflow. WORKS@SC 2013: 98-105

v https://airflow.apache.org/

vi https://github.com/spotify/luigi

vii https://github.com/argoproj/argo

viii https://beam.apache.org/

ix https://cloud.google.com/dataflow/

x https://spark.apache.org/

xi https://flink.apache.org/

xii Malte Schwarzkopf: Cluster Scheduling for Data Centers. ACM Queue 15(5): 70 (2017)

xiii Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, John Wilkes: Borg, Omega, and Ku bernetes. Commun. ACM 59(5): 50-57 (2016)

xiv https://kubernetes.io/

xv B. Hindman et al, “Mesos: A platform for fine-grained resource sharing in the data cen ter,” in NSDI'11 Proceedings of the 8th USENIX conference on Networked systems des sign and implementation, 2011.

xvi https://landscape.cncf.io/

xvii Luis M. Vaquero, Félix Cuadrado, Yehia Elkhatib, Jorge Bernal Bernabé, Satish Naraya na Srirama, Mohamed Faten Zhani: Research challenges in nextgen service orchestra tion. Future Generation Comp. Syst. 90: 20-38 (2019)

xviii https://k3s.io/

xix https://github.com/kubernetes-sigs/federation-v2

xx https://cilium.io/blog/2019/03/12/clustermesh/

xxi https://github.com/admiraltyio/multicluster-scheduler

xxii https://www.zerotier.com/manual/

xxiii https://pjreddie.com/darknet/yolo/

xxiv https://kubernetes.io/docs/tasks/access-application-cluster/configure-access-multiple-clusters/