Ref. Ares(2020)8018055 - 31/12/2020

ACCORDION

Adaptive edge/cloud compute and network continuum over a heterogeneous sparse edge infrastructure to support nextgen applications

DRAFT

Deliverable D2.3

Architecture design (I)

ACCORDION receives funding from the European Union’s Horizon 2020 research and innovation programme under Grant Agreement No. 871793

DOCUMENT INFORMATION

PROJECT	
PROJECT ACRONYM	ACCORDION
PROJECT FULL NAME	Adaptive edge/cloud compute and network continuum over a heterogeneous sparse edge infrastructure to support nextgen applications
STARTING DATE	01/01/2020 (36 months)
ENDING DATE	31/12/2022
PROJECT WEBSITE	http://www.accordion-project.eu/
TOPIC	ICT-15-2019-2020 Cloud Computing
GRANT AGREEMENT N.	871793
COORDINATOR	CNR
DELIVERABLE INFORMATION	
WORKPACKAGE N. | TITLE	WP2: Requirements & System Design
WORKPACKAGE LEADER	HUA
DELIVERABLE N. | TITLE	D2.3: Architecture design (I)
EDITOR	K. Tserpes (HUA)
CONTRIBUTOR(S)	K. Tserpes (HUA), G. Kousiouris (HUA), S. Xydis (HUA). T. Kafatari (HUA), Ioannis Korodanis (HUA), Patrizio Dazzi (CNR), Emanuele Carlini (CNR), Hanna Kavalionak (CNR), Luca Ferrucci (CNR), Felipe Huici (NEC), Eduard Marin Fabregas (TID), Nicolas Kourtellis (TID), Diego Andilla Ferran (TID), Ioannis Violos (ICCS), Evangelos Psomakelis (ICCS), Mateusz Kamiński (BSOFT), Bartłomiej Lipa (BSOFT), Tom Loven (PLEX), Yago Gonzalez Rozas (PLEX), Andrea Toro (HPE), Marco Russo (HPE), Lorenzo Blasi (HPE), Alain Vailati (HPE), Marco Di Girolamo (HPE), Nadir Zinelaabidine (AALTO), Zbyszek Ledwoń (ORBK), Saman Zadtootaghaj (TUB), Maria Pateraki (OVR), Spiros Fotis (AEGIS), Leonidas Kallipolitis (AEGIS), Spyros Vantolas (AEGIS)
REVIEWER	Patrizio Dazzi (CNR)

DRAFT

CONTRACTUAL DELIVERY DATE	12/2020
ACTUAL DELIVERY DATE	31/12/2020
VERSION	1.0
TYPE	Report
DISSEMINATION LEVEL	Public
TOTAL N. PAGES	68
KEYWORDS	Architecture, Cloud, Edge, Federation

DRAFT

EXECUTIVE SUMMARY

This deliverable provides an account of the work done for the specification of the ACCORDION ecosystem architecture, which is mainly comprised of two artefacts: the infrastructure to support the cloud-edge continuum and the platform to manage the resources and the applications hosted in that infrastructure. The main outcomes of the work described in this deliverable are: a) the definition of components that comprise the architecture as well as their interactions, and; b) the assignment of the implementation of those components to specific project Tasks and partners.

The deliverable describes the process for reaching to those outcomes, starting with the delineation of the scope of the described artefacts and working down to their specifics. The main approach used is the analysis of the platform use case scenarios. In practice, we analyze the scenarios and extract the required functionality. Then we cluster and map this functionality to components. Finally, we schematically present the interactions of those components using block and sequence diagrams. These outcomes are then validated against the use case requirements, especially those linked to the applications operation itself. This is meant to provide a proof-of-concept that the ACCORDION architecture is able to support the ACCORDION applications.

DISCLAIMER

ACCORDION (871793) is a H2020 ICT project funded by the European Commission.

ACCORDION establishes an opportunistic approach in bringing together edge resource/infrastructures (public clouds, on-premise infrastructures, telco resources, even end-devices) in pools defined in terms of latency, that can support NextGen application requirements. To mitigate the expectation that these pools will be “sparse”, providing low availability guarantees, ACCORDION will intelligently orchestrate the compute & network continuum formed between edge and public clouds, using the latter as a capacitor. Deployment decisions will be taken also based on privacy, security, cost, time and resource type criteria.

This document contains information on ACCORDION core activities. Any reference to content in this document should clearly indicate the authors, source, organisation and publication date.

The document has been produced with the funding of the European Commission. The content of this publication is the sole responsibility of the ACCORDION Consortium and its experts, and it cannot be considered to reflect the views of the European Commission. The authors of this document have taken any available measure in order for its content to be accurate, consistent and lawful. However, neither the project consortium as a whole nor the individual partners that implicitly or explicitly participated the creation and publication of this document hold any sort of responsibility that might occur as a result of using its content.

The European Union (EU) was established in accordance with the Treaty on the European Union (Maastricht). There are currently 27 members states of the European Union. It is based on the European Communities and the member states’ cooperation in the fields of Common Foreign and Security Policy and Justice and Home Affairs. The five main institutions of the European Union are the European Parliament, the Council of Ministers, the European Commission, the Court of Justice, and the Court of Auditors (http://europa.eu.int/).

Copyright © The ACCORDION Consortium 2020. See https://www.accordion-project.eu/ for details on the copyright holders.

You are permitted to copy and distribute verbatim copies of this document containing this copyright notice, but modifying this document is not allowed. You are permitted to copy this document in whole or in part into other documents if you attach the following reference to the copied elements: “Copyright © ACCORDION Consortium 2020.”

The information contained in this document represents the views of the ACCORDION Consortium as of the date they are published. The ACCORDION Consortium does not guarantee that any information contained herein is error-free, or up to date. THE ACCORDION CONSORTIUM MAKES NO WARRANTIES, EXPRESS, IMPLIED, OR STATUTORY, BY PUBLISHING THIS DOCUMENT.

DRAFT

REVISION HISTORY LOG

VERSION No.	DATE	AUTHOR(S)	SUMMARY OF CHANGES
0.1	10/11/2020	K. Tserpes (HUA)	ToC, Initial content
0.2	01/12/2020	K. Tserpes (HUA)	Assignments
0.3	15/12/2020	All	Contributions' deadline
0.4	21/12/2020	K. Tserpes (HUA)	Review version
0.5	28/12/2020	P. Dazzi (CNR)	Review report
1.0	30/12/2020	K. Tserpes (HUA)	Final version

DRAFT

GLOSSARY

EU	European Union
EC	European Commission
H2020	Horizon 2020 EU Framework Programme for Research and Innovation
DoA	Description of Action
VNF	Virtualized Network Function
QoE	Quality of Experience
SLA	Service Level Agreement
SOA	Service-oriented Architecture
VIM	Virtualized Infrastructure Manager
CSI	Container Storage Interface
CRUD	Create Read Update Delete
DevOps	Development and Operations
CI/CD	Continuous Integration/Continuous Delivery
SCM	Source code management
ACR	Absolute Category Rating

DRAFT

TABLE OF CONTENTS

DRAFT

DRAFT

DRAFT

1 Relevance to ACCORDION

1.1 Purpose of this document

This document presents the first version of the ACCORDION architecture as the main outcome of work in *Task 2.3: Frameworks architecture & specifications* (T2.3). It is a living document that will be updated in later iterations of the project. The first release of the report describes the architecture of the ACCORDION framework, presenting it in an organic way along with the description of its components and their APIs.

The work in T2.3 is active throughout the period M07-M35 and is led by HUA. The contributing partners are: CNR, TID, HPE, ICCS, NEC, AALTO, TUB, BSOFT and AEGIS. The goals of this work are:

- To map the requirements to blocks of functionality and from those to standalone software components
- To assign partners with the responsibility to implement those components
- To provide instructions to both individual component developers and integrators about component implementation

The starting point for the analysis to lead to these objectives was the project vision, i.e., to implement an **infrastructure** comprised of mixed cloud and edge resources and implement a **framework** to manage it and allow the application development and deployment on top of it. As stated in the DoA, *ACCORDION aspires to:*

- *provide an open, low-latency, privacy-preserving, secure and robust virtualized infrastructure*
- *provide an application management framework, tailored to the needs of the European SMEs skillset, that will enable the deployment of NextGen applications on top of this infrastructure, reaping the benefits of edge computing.*

These statements set the goal of the system(s) that T2.3 needed to design. This was enhanced with the analysis provided in *D2.1*: *User requirements (I)*. Specifically, the technical/scientific requirements together with the above-mentioned goal, set the scope for the design of the infrastructure and the management framework.

With those in mind, T2.3 set to define the characteristics of the infrastructure and management framework through a use case scenario analysis. The scenarios were meant to answer questions about the use of the system. Such questions of interest to ACCORDION were:

- how resources participate to the cloud/edge continuum
- how an application can be developed and deployed in an environment like ACCORDION
- how an application can be launched within ACCORDION
- how the lifecycle of the application can be managed in an automatic way

For each of those scenarios, we identified the functionality that their realization entails and allocated components to bring about this functionality. We continued with defining the specification of those components, a work largely guided by the work in *D2.2: State of the art report (I)*. This process gradually led us to the definition of a conceptual architecture which is the main outcome of this report. To validate sch outcome, we assessed the extent to which the use case requirements defined in report *D2.1: User requirements (I)* can be met by the current version of the architecture.

1.2 Relevance to project objectives

This deliverable is particular in the sense that it is a precursor to the technical work and as such it is link to all technical project objectives. In fact, the objectives of the architecture are identical to the project technical objectives, namely:

- Obj1: Maximize edge resource pool
- Obj2: Maximize robustness of cloud/edge compute continuum
- Obj3: Minimize overheads in migrating applications to cloud/edge federations
- Obj4: Realize NextGen applications

1.3 Relation to other work packages

Task T2.3 generates an output that will be consumed by all the research tasks (since they all contain a development part) as well as from the integration task. The system architecture relies on SOA principles allowing for each task from WP3-5 to act independently selecting their preferred underlying technologies and generate loosely coupled components under well-defined interfaces.

Furthermore, the architecture will be consumed by WP6, employing the concepts defined here in order to design the application pilot implementations. In particular, sequence diagrams will be created for the particular use cases showing the interaction and flow of logic between the components defined here.

Finally, this work is also influenced by the work reported in *D7.5: Technoeconomic analysis* in which high-level concepts have been specified and are used as such in D2.3.

1.4 Structure of the document

To achieve T2.3 goals, during the period M07-M12, the partners worked on the development of the main concept and the relevant definitions (Sections 2 & 3), including that of the **ACCORDION applications** and the **ACCORDION platform**. With those artifacts conceptually defined, the partners worked on the identification of the **platform operation scenarios**, i.e., scenarios that describe the main operations of the ACCORDION platform (Section 4). Those scenarios provided the guidelines for singling out the required functionality and mapping it into software components (Section 5). With those components specified, T2.3 was able to draft the first conceptual architecture diagrams followed by sequence diagrams to further assist the potential readers of this deliverable (Section 6). Finally, working backwards, we checked how this work meets the use case requirements defined in D2.1 (Section 7).

DRAFT

2 Introductory concepts

What is an **ACCORDION application**? An ACCORDION application is a set of interacting, independent application components, each one running in its own execution environment (e.g. containers) with the following characteristic requirements: extremely low latency demands in their pairwise communication with the end-user devices and high end-to-end QoE.

In a traditional setting such as Kubernetes[1], those components would reside within a single pod with their desired state being described in a single deployment. Unfortunately, the demands for low latency with the end devices, dictate the need to deploy the components to resources that are close, in terms of latency, to the end devices. We refer to these resources as **edge nodes**.

It is not safe to assume that edge nodes are available on demand because they can't be virtually infinite and everywhere. And when they become available, it might be the case that they are of limited capacity or availability. Apart from technical reasons, one needs to consider that the reduced supply and unique positioning of the edge nodes will render them expensive and they need to be wisely utilized. Those edge nodes may become available based on an opportunistic scenario. For example, they may become available if there is a large profit margin or underutilization in the core business of the infrastructure owner and unavailable otherwise. It turns out that the pool of edge nodes is dynamic for two reasons: a) edge nodes may come and go unpredictably, and b) their capacity may change at runtime.

In order to mitigate the infused uncertainty, the application components need to be distributed across edge and cloud nodes. Such a distributed execution environment needs to be properly managed in a way that is agnostic to the application. It also needs to provide services and operations that will accommodate the need for high QoE and simplify the developer's work. Such services may be auto-scalers, load balancers, message brokers, event processors or combinations of the latter two (e.g., Apache Pulsar[2]), streaming data processors and higher-order functions (e.g., map, reduce, filter) but also general-purpose virtualized network functions (VNF) that intervene in the data path compressing or interpolating data. Those VNFs are also called "**data services"**.

DRAFT

[1] https://kubernetes.io/
[2] https://pulsar.apache.org/

3 High-level approach

ACCORDION considers that each edge or cloud node in a computing continuum is an execution environment for application components. We refer to this environment as "**minicloud**". The minicloud is managed in the same way disregarding the underlying resources, through the ACCORDION **VIM (Virtualized Infrastructure Manager)**. As such, the ACCORDION infrastructure is consisted of a **federation of miniclouds**. The overarching management layer that manages the federation (continuum management framework) and provides tools for the deployment as well as development (application management framework) of ACCORDION applications is called **ACCORDION platform**.

Figure 1 depicts the high-level organization of the ACCORDION architecture. The infrastructure layer (bottom layer) is consisted of miniclouds that can be hosted in public or private clouds, or even dedicated bare-metal infrastructures. On top of that layer resides the **ACCORDION platform** which in turn can be hosted in any ACCORDION minicloud. The ACCORDION platform bears all the services required to manage the federation, collectively referred to as Federation Management Services (FMS). The platform can be either distributed or centralized in nature, i.e., spread across multiple cooperating miniclouds or hosted in a single one. Obviously, the distributed version entails bigger challenges than the centralized, yet it bears benefits related to the scalability and fault tolerance of the platform.

DRAFT

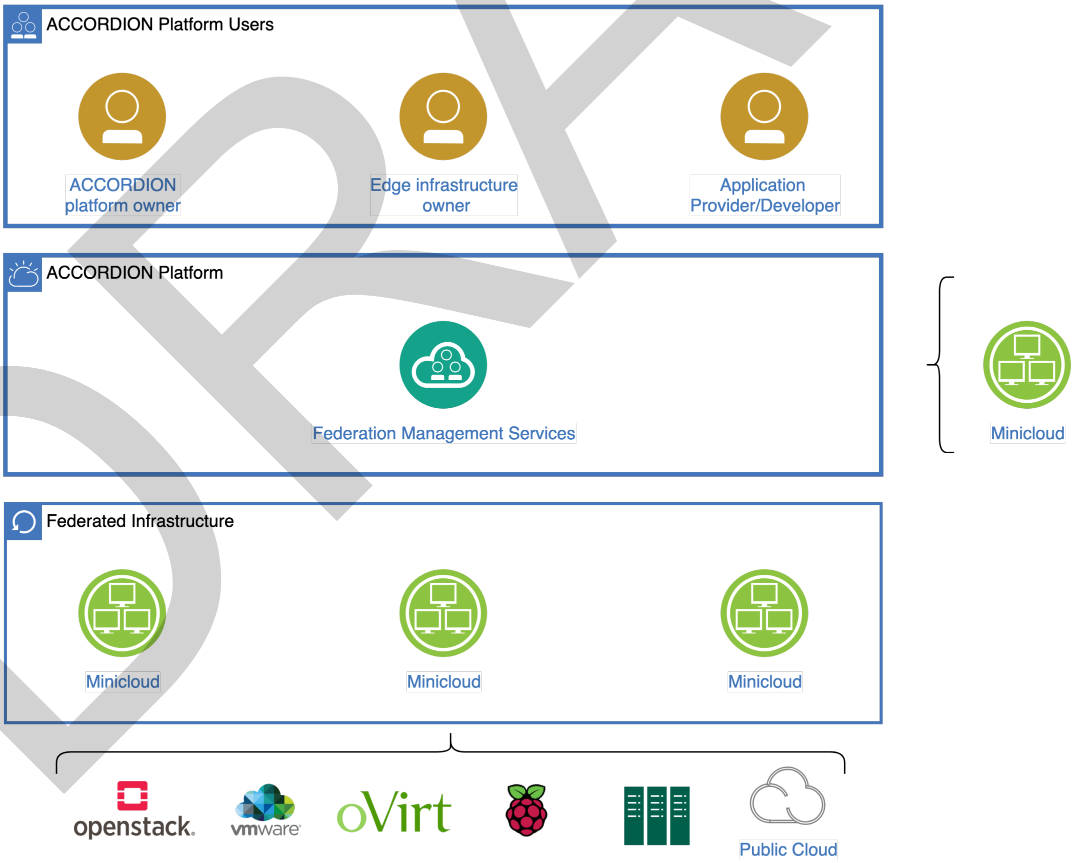


Figure 1: High level view of the ACCORDION architecture

3.1 Actors

The top layer of the ACCORDION architecture depicts the ACCORDION platform end-users. Each actor depicted refers to a business entity. For each of such entity there is an agent, a human actor that acts on behalf of the business entity. Based on that the analysis is the following:

- **Infrastructure owner**: An entity that owns one or more edge nodes and makes them available in the federation. It makes a profit by charging the utilization of its edge nodes, for a premium that depends on its unique position in terms of latency with the application end devices.
  - Agent: **Infrastructure administrator**
- **Application provider**: An entity that owns an application and knows the details of the application operation. The application provider uses the ACCORDION resources and services to host his application, taking advantage of the proximity (in terms of latency) of the ACCORDION-controlled edge nodes to the end devices.
  - Agent: **Application developer**
- **ACCORDION Platform Owner**: It is the entity that benefits from brokering between application providers and infrastructure owners. It finds resources meeting the special requirements for the first party and applications to utilize the resources for the second party. In order to facilitate the above, the entity deploys and manages the ACCORDION Platform.
  - Agent: **Platform administrator**

More information about the business aspects pertaining those entities may be found in the ACCORDION Report series: Technoeconomic analysis (version I, D7.5, December 2020).

DRAFT

4 Platform operation scenarios

DRAFT

We distinguish a number of high-level scenarios that the ACCORDION platform must support:

- **Join/Leave Federation**: An infrastructure owner decides to commit his resources to the ACCORDION federation or decommission from when he decides to leave the ACCORDION federation.
- **Deploy application**: An application provider decides to deploy his application to the ACCORDION platform. This step potential includes the building of the application components from which the application is composed of.
- **Start application**: The application provider decides to start/launch the application.
- **Runtime adaptation**: The ACCORDION platform reacts to its changing environment with the objective to mitigating possible QoE degradation, like when miniclouds are entering or leaving the federation, or recovering from failures.

Through these scenarios we provide a narrative about what the application is supposed to be doing for its actors. Analyzing the narratives, we gradually identify the required functionality that the ACCORDION platform must implement.

4.1 Join/Leave Federation

In this use case, an infrastructure owner decides to commit his resources to ACCORDION (Figure 2).

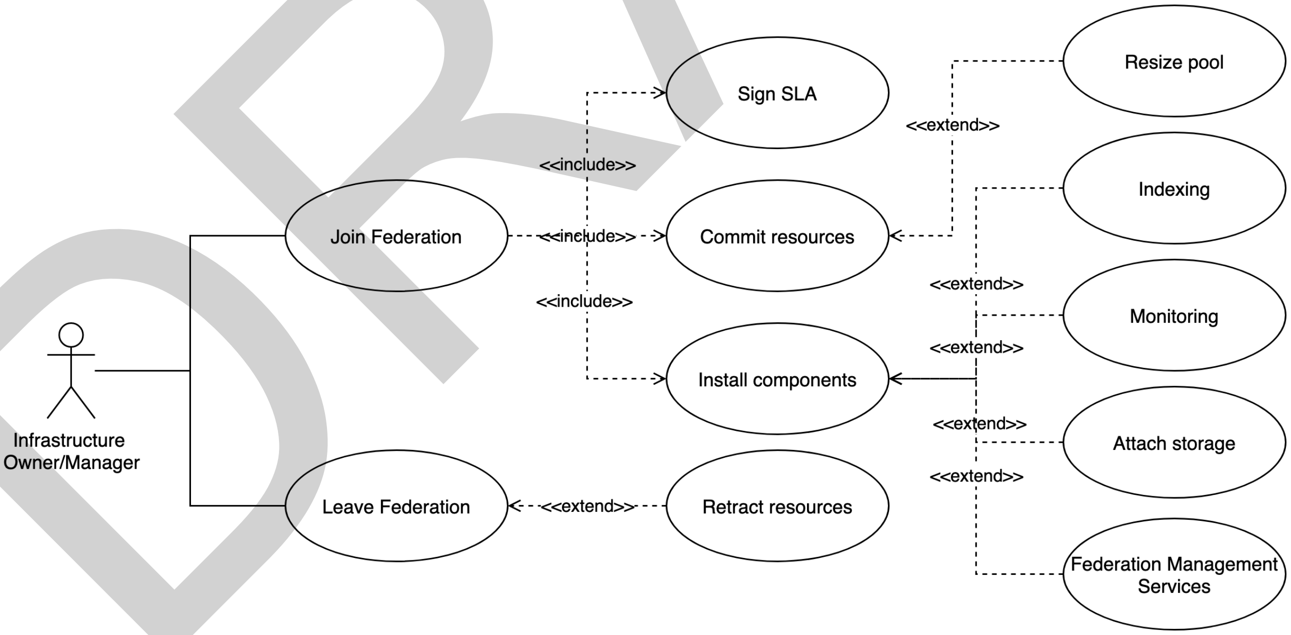


Figure 2: Use case diagram for joining/leaving the ACCORDION federation

Assumptions	● The resources within the minicloud are used to provide either containers (CaaS) or VMs (IaaS) as a service to its client ● The minicloud enables control over its services through an API. Control comes through a set of processes such as: to upload, organize, start, stop, scale and otherwise manage VMs, containers, applications. Furthermore, the minicloud also offers an API with monitoring information of the underlying infrastructure. ● The infrastructure owner may retract the resources being available to the federation at any time. ● The business-related part of the federation is currently out of the scope of this analysis and more information can be sought at the ACCORDION Report series: Technoeconomic analysis (version I, D7.5, December 2020).

The steps for joining the federation include the following sub use cases:

- **Sign SLA**[3]: The infrastructure manager needs to agree on the **Federation SLA** so as to provide some sort of guarantees regarding a predefined time period for the migration of the hosted components and/or a general period for which they may become available.
- **Commit resources**: The infrastructure owner carves out a part of his infrastructure and provides it to ACCORDION platform as a minicloud or to be added to an existing minicloud. This means that the minicloud resources are available to the ACCORDION platform to deal them as if they belonged to it.
- **Resize resource pool**: The infrastructure owner needs to increase/decrease the number of resources committed to the ACCORDION resource pool
- **Install ACCORDION components**: The Infrastructure manager installs all necessary components or commits his resources to an appropriate existing minicloud. The functionality that these components implement include: accessing and managing the edge node through the VIM; monitoring and indexing the edge node; linking the node with a storage unit, and; Federation Management Services in cases where the ACCODION platform is installed in a distributed fashion.

The respective steps for leaving the federation include the following sub use cases:

- **Retract resources**: The Infrastructure owner can revoke the ACCORDION access token validity at any time. Of course, a process needs to be defined for that that will include the definition of the time period to migrate application components and informing the Federation Management Services.

4.1.1 Required functionality

[3] https://it.wikipedia.org/wiki/Service_level_agreement

DRAFT

In what follows we list the functionality that we must implement in order to validate the use case scenario, together with the task that will define and implement it.

Federation SLA Lifecycle Management (Task 7.5) (only definition)

There is the need to define the SLAs that will govern the business relationship between the ACCORDION actors and for a component that will perform all the actions to manage the complete lifecycle of the SLAs (monitoring, billing, penalizing, etc.). The implementation of this component is out of the scope of ACCORDION, yet its definition remains part of the work in Task 7.5.

Virtualized Infrastructure Management (Task 3.5)

There is the need for a Virtual Infrastructure Manager (VIM), i.e., a component that allows the management of the miniclouds. It represents an entity that has access to the underlying resources that also exposes an API that allows the implementation of operations for cloud- and container-based environments. It also extends a typical cloud API by adding monitoring data etc.

Monitoring and Characterization (Task 3.1)

A component must exist that will be able to collect monitoring information about the resources and even the applications that are controlled by each minicloud. It must provide its data in a push and/or pull way and it must allow for the registration of listeners, i.e., sort of alerts that are fired when certain metrics (or combinations) are evaluated. Assuming a vastly heterogeneous environment, the component must be able to characterize the resources in terms of their capacity and capabilities, e.g., if they bear a GPU, whether they are battery powered, etc.

Indexing and Discovery (Task 3.2)

A component must be able to keep an up-to-date status of computational resources among the various miniclouds and provide a service that allows to run queries on these resources. The component should be able to respond to queries about the appropriateness of available resources to meet certain computational criteria.

Storage Provision (Task 3.3)

For each minicloud there must be a component that will be able to abstract the storage units that are made available to the edge nodes be it in the minicloud itself or in another. This component must be able to provide the necessary storage resources even at runtime.

Minicloud Membership Management (Task 3.5)

There must be a component to maintain the list of edge nodes that are available at any time, together with information about their characteristics. We need to be able to know what and how many resources are provided by each minicloud at any time and we also need to have the endpoints of their APIs to access and manage them.

DRAFT

4.2 Deploy/un-deploy application

In this use case, the Application Provider wants to (un-)deploy the application components and configure the infrastructure so as to support their operation (Figure 3). In this case, the term “deploy” implies the building of the application components and their storage close to where the appropriate resources may be. The resources can either be found or created, yet the actions of the platform must result in the minimum possible cost for the application provider. Example scenario: a container is found in an edge node of interest. The application component image will be stored in the local storage but will be not loaded in the container. In this case, we assume that the application component is deployed.

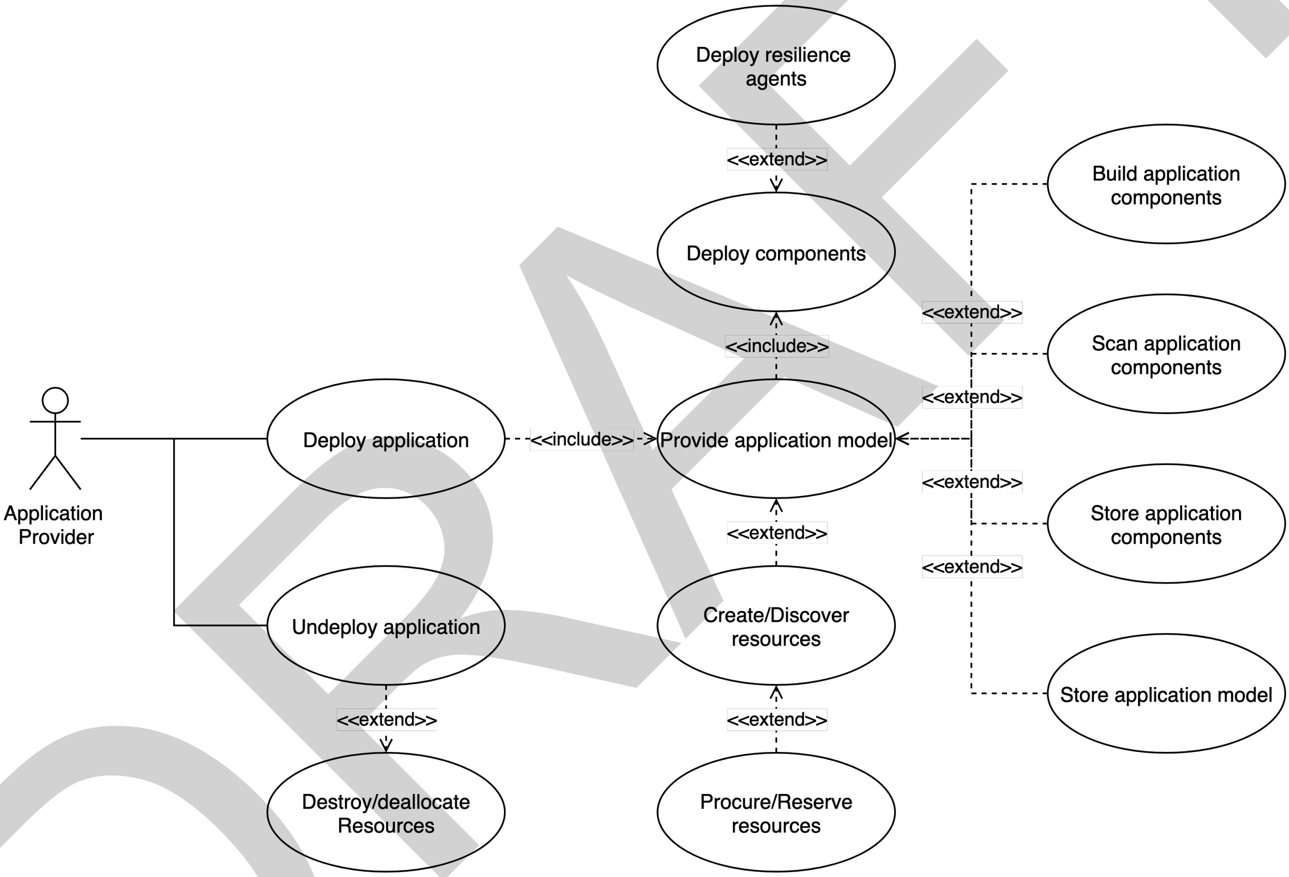


Figure 3: Use case diagram for deploying an application in ACCORDION

DRAFT

Assumptions	- The application provider authenticates before deployment. It might have to be registered and obtain some credentials

The steps for deploying the application are the following:

- **Provide application model**: The application provider submits a document that primarily aims at ensuring resource provisioning but secondarily it may define the deployment, the application QoS/QoE requirements, the application workflows and the application lifecycle management.
- **Store application model**: The model is stored in an application registry, under an application account for monitoring its status and resource utilization.
- **Build application components**: The application developer provides the links to the source code and the build instructions to build the application component in a DevOps pipeline. Such component may be a container or VM image.
- **Scan application components**: The application components need to be checked whether they comply with security and privacy preserving standards.
- **Create/Discover resources**: The application model is consumed primarily by the Federation Management Services. Compute and network resource orchestrators query the indexing service, to find or create resources that match the QoE/QoS requirements of the application.
- **Procure/Reserve resources**: In case of "eager" provision (procurement), the orchestrators will hold and mark the discovered resources as "owned" by the application. In case of "lazy" provision (reservation), the resources are reserved, and procurement happens at. In ACCORDION "lazy" provision is the most probable scenario, because it won't be possible to know where the server application components will have to be executed with clients appearing at virtually anywhere in the world.
- **Deploy components**: Once all the above-mentioned processes have finished, the orchestrator deploys the actual application components as per instructions from the application model.
- **Deploy resilience agents**: Together with the application components, the platform installs the resilience scanning agents, i.e., security, privacy preserving and fault-tolerance. Those agents continuously scan, report and mitigate fault and challenges to normal operation either based on the application and its QoE model or based to an established knowledge of "normal" operation.

The steps for un-deploying the application are the following:

- **Un-deploy application**: The application provider may decide to un-deploy the application and thus destroy or de-allocate the assigned resources.
- **Destroy/de-allocate resources**: If the resources are created, then they have to be destroyed. If they were found, they need to be freed. This also entails the updating of the application account.

DRAFT

4.2.1 Required functionality

In what follows we list the functional components that derive from the use case scenario together with the task that will define and implement it.

Application Model (Task 5.1)

The application model (more accurately referred to "application description") must define, perhaps at distinct times and documents, the following concepts about the application components:

- **definitions**: names and other metadata related to the application components
- **building instructions**: repository links and scripts to build the docker images for each application component
- **resource requirements**: number of needed vCPUs, memory, storage, etc.
- **deployment instructions**: scripts to configure and "install" the application components, e.g., define replica sets and ingress (ingress controller or load balancer)
- **workflow definitions**: possible pipelines that refer to different application-related procedures
- **QoE requirements**: definition of metrics to be monitored and thresholds that define normal/not normal operation
- **lifecycle management instructions**: definition of events that may trigger the enactment of cloud/edge ACCORDION services or operations (e.g., scaling operations)
- **application status**: the application model needs to reflect the status of the application at runtime. I.e., with regards to the application component instances, we need to know how many, where and for how long they are running. We also need endpoints to those instances to be able to manage them.

Application Registration (Task 6.4)

We need a registry to store the applications currently running in ACCORDION and maintain the application status. The relevant component must create the notion of "application account". It must store information about the owner of the application and the application itself. It particularly must keep information about the application workflows and the events that trigger them. It should also maintain a list of the utilized resources for this application account.

Identity and Access Management (Task 5.5)

A front-end to the ACCORDION end users must exist. It must provide the starting point for the user providing authentication, authorization and accounting mechanisms.

Build Application Components (Task 5.4)

ACCORDION must allow for the application VM or container images to be built through appropriate instructions (scripts and source code links).

Define Application Workflows with ACCORDION Services (Task 5.1)

The application provider should be able to define workflows including not only application components but also ACCORDION services. Based on the application use cases discussed in Section 0, ACCORDION services may be data services such as higher-order functions, standard offerings that may facilitate the use case workflows (e.g., message brokers) as well as standard cloud offerings such as auto-scalers and load balancers.

Create Network Paths for Optimized Application Deployment (Task 5.3)

DRAFT

ACCORDION must efficiently spot problems that could disrupt the operation and performance of the system (e.g., congested links, node capacity excess, etc.), and accordingly plan operations to fix these issues (e.g., migrate virtual machines, remove congested links, etc.). The DevOps approach will permit to automate these operations within a reusable workflow that contains automated processes, optimized to ensure the optimal planning and management of network paths at the edge, and including automated recovery actions.

Lightweight virtualization support (Task 3.4)

ACCORDION must be able to support the creation of Unikernels[4], reaping the benefits they bring in terms of performance and security. Also, the project may benefit from the capabilities of Unikraft[5] to construct Unikernels.

Submission or Creation of Application Model (Task 5.5)

There must be a usable way to provide or even construct the deployment instructions, the application component requirements, the QoS requirements and the application lifecycle management instructions. This needs to be stored and indexed with appropriate endpoints to access it.

Application Component Registration (Task 5.5)

A container or VM image registry is required. This is where the application components that succeed the scanning tests are stored and indexed. It must be possible to deploy those components easily with low latency.

Compute Resource Orchestration (Task 4.1)

An orchestrator must exist that must be able to:

- Find or create appropriate (in terms of meeting the QoE requirements) resources for the application components
- Deploy application components to *appropriate* resources
- Perform necessary activities for the application lifecycle management including the invocation of data and cloud services and operations
- Inform the application status at all times

Network Resource Orchestration (Task 4.2)

An orchestrator must exist that must be able to:

- Find or create *appropriate* network resources to satisfy the application needs
- Perform necessary activities for the application lifecycle management including the invocation of data and cloud services and operations and the network reconfiguration
- Inform the application status at all times

[4] http://unikernel.org/
[5] http://www.unikraft.org

DRAFT

DRAFT

Deployment and Runtime Security Conformance Evaluation (Task 4.4)

A component must exist that will check whether a built application component conforms with the ACCORDION security standards and provide recommendations for fixing them in case they exist. The same must be done at runtime.

Deployment and Runtime Privacy Preserving Conformance Evaluation (Task 4.5)

A component must exist that will check whether a built application component conforms with the ACCORDION privacy preservation standards and provide recommendations for fixing them in case they exist. The same must be done at runtime.

Failure detection and mitigation (Task 4.3)

A component must exist that will be responsible for monitoring and mitigating potential faults in the system. Mitigation may include hot (or live) migration commands or replication.

QoE Model (Task 5.2)

A model must be able to map the application QoE requirements to QoS requirements and to define what values are acceptable and not.

4.3 Start/Stop Application

The application provider decides to launch the application, after it has been deployed and perhaps stop it later (Figure 4). The term "launch" or "start" implies that the application components start being executed and more costs are incurring. In the example scenario described in the "deploy" section, the start of the application would imply the creation of the container, if it didn't exist, and the loading of the image in it.

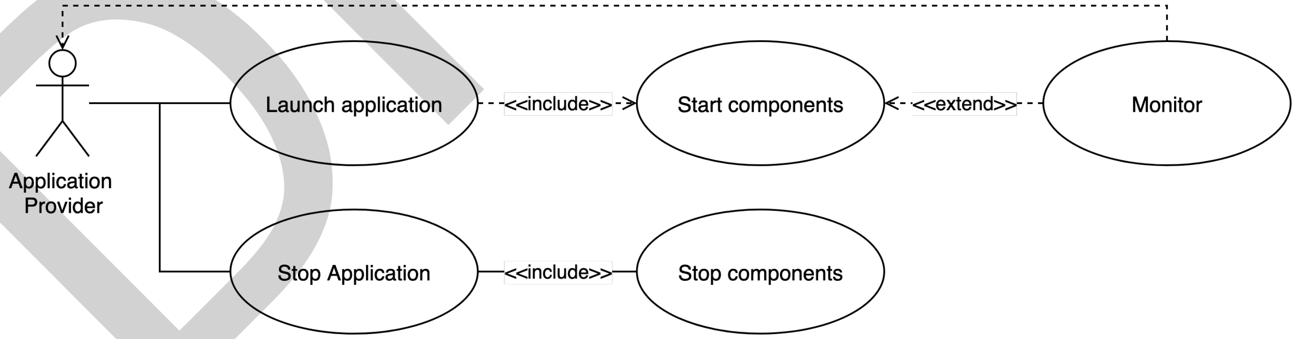


Figure 4: Use case diagram for starting/stopping an application in ACCORDION

Assumptions	● The system is running on an event-driven logic ● All system related messages are flowing in an event bus

The steps for starting/stopping the application are the following:

Start components: Once the developer issues the explicit "launch" intent, the workflow declared as "main" is initiated.

Monitor: The relevant queries are issued to monitor the application's resources status and report it back to the Application provider.

Stop components: Once the developer issues the explicit "stop" intent, the components are stopped. Alternatively, the components stop, when an event that has been declared as terminal for the application happen.

4.3.1 Required functionality

Application Component Registration (Task 6.4)

As defined in 4.2.1.

Event Bus (Task 6.4)

A component is needed in order to consume several input streams and detect events. Such streams may be logs or reported output from various components. The component must check in the message flows attempting to find combinations of messages that compose an event that may trigger an action. It may do so based on systems like Apache Druid[6] or Apache Pulsar[7]. The actions are references to cloud computing operations such as those found in any IaaS cloud API. E.g., start, stop, delete, move VM/container, scale in/out, scale up/down, deploy subnetwork, assign IP, etc. They can also be references to software engineering patterns, such as balance load, replicate, enable secure channel, microservice or serverless design patterns, for instance higher-order functions. Finally, they can be references to complex cloud API operations that can be provided as "black box" scripts.

Application Status Reporting (Task 5.5)

There must be a front end that will allow the application provider/developer to monitor the status of the application, including whether it is running or not, where the application components instances and

[6] https://druid.apache.org
[7] https://pulsar.apache.org

DRAFT

ACCORDION services are deployed, for how long, how much has it costed yet. The end user must be able to start and stop the application from the same front-end.

Monitoring Application Status (Task 3.1)

The application status must be maintained through appropriate queries at the monitoring mechanism and by appropriate configuration at the level of execution environment. For example, in Kubernetes, we can reserve a namespace for the pods to which the application runs.

4.4 Runtime adaptation

This use case is related to the self-adaptive behavior of the ACCORDION platform with the primary objective to optimize its QoS provision (mitigate QoS degradation).

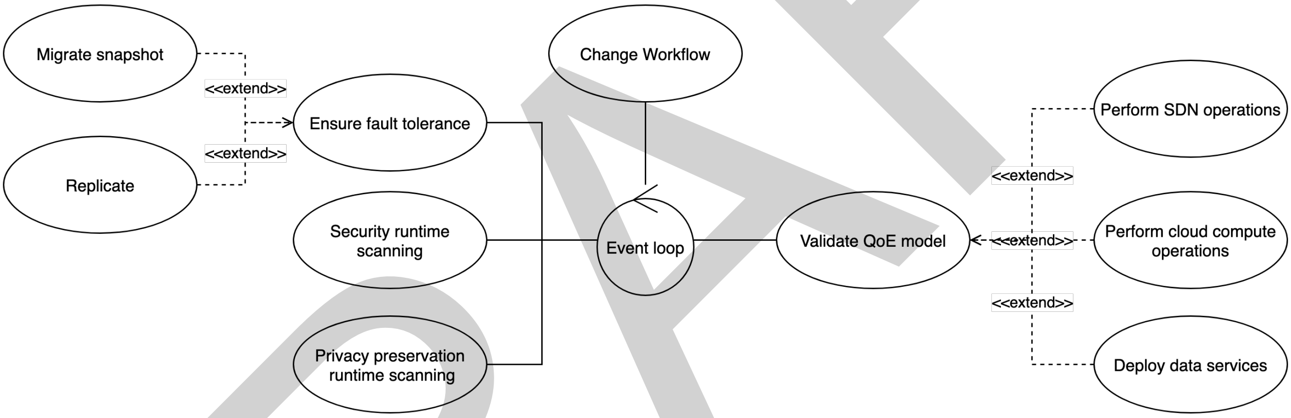


Figure 5: Use case diagram for the ACCORDION platform runtime adaptation

DRAFT

Assumptions	• All necessary information for determining the events that trigger runtime adaptation activities is provided in the form of an event loop

The steps for managing the lifecycle of the application are the following:

Validate QoE model: The steps involve the constant monitoring of the state of the application, and its evaluation against the application model and particularly the QoE part. The QoE model includes monitoring parameters and sets the conditions that indicate the degradation of the QoE. A handler for QoE-related events fires the necessary orchestrating activities.

Perform cloud operations: In case that the QoE levels are compromised, the orchestrators may issue commands for the scaling of the application component or the deployment of a load balancer or the restarting of application components and the complete workflow.

Perform SDN operations: This use case refers to activities such as network reconfiguration from the network resource orchestrator, should this be deemed necessary.

Deploy data services: For similar reasons as above, the orchestrators may demand to deploy a data service.

Ensure resilience: A component detects potential off-normal behavior of resources and in case of high possibility for failure it triggers mitigation plans. Such plans include the migrations of the running snapshot to more appropriate resources or the replications of the components for redundancy.

Security runtime scanning: An agent regularly scans the resources where the application is running and checks whether they are OK in terms of security

Privacy preservation runtime scanning: An agent regularly scans the resources where the application is running and checks whether there is any privacy leakage

Change workflow: Certain events defined in the application model may trigger the changing of the application workflow.

4.4.1 Required functionality

Event Bus (Task 6.4)

Same as 4.3.1.

QoE Model Validation (Task 5.2)

A mechanism must exist for mapping current QoS parameters, as measured by the monitoring component, to the desired application QoE.

Compute Resource Orchestrator (Task 4.1)

Same as 4.2.1.

Network Resource Orchestrator (Task 4.2)

Same as 4.2.1.

ACCORDION Data Services (Task 5.1)

There must be a mechanism to define and deploy general-purpose and even programmable components that facilitate the needs of the applications by intervening in the data pathway. These can be higher-order functions or message queues.

Failure Detection and Mitigation (Task 4.3)

Same as in Section 4.2.1.

DRAFT

Runtime Security Conformance Evaluation (Task 4.4)

Same as in Section 4.2.1.

Runtime Privacy Preserving Conformance Evaluation (Task 4.5)

Same as in Section 4.2.1.

DRAFT

5 Architecture

In what follows, we use the requirements as they were defined in the previous Section and start identifying the components and the functionality that could meet them. Figure 6 presents the ACCORDION stack, a conceptual view of the ACCORDION architecture in which the components are placed in layers based on whom they are providing services to. We start with the ACCORDION platform users/actors and move on to the ACCORDION Platform layer that is containing components that are meant to manage the applications and the underlying infrastructure. The bottom layer is the Federated Infrastructure Layer, in which we position the Miniclouds, the basic computation unit of ACCORDION. The figure also depicts the possible resource composition that could support the Minicloud. These are defined in the Description of Action and they are meant to act as a proof of concept. In particular, the edge Miniclouds may be hosted in private or public clouds and clusters of mainstream or single-board computers. The ACCORDION Platform itself, will be hosts in a Minicloud.

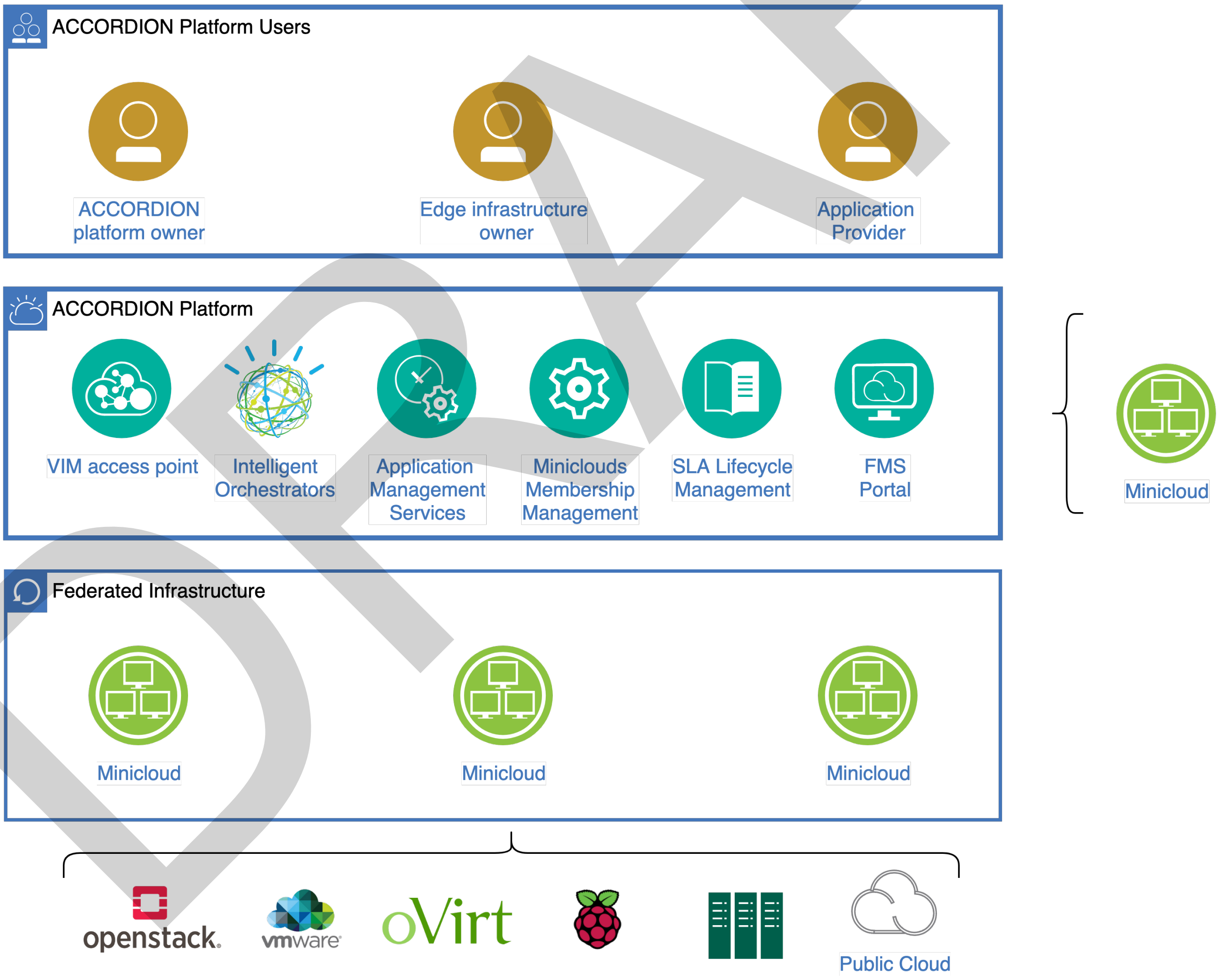


Figure 6: ACCORDION Stack

Before we delve into the details of the functional components' specification as they are presented in the fugure, we would like to draw the attention of the reader in two notes:

DRAFT

- Figure 6 is preceding the analysis in a possible counter-intuitive way, however it is meant to assist the reader have an overview of the findings before those are presented.
- Not all the components are functional and therefore not present in Figure 6. In fact, in the analysis that follows, each layer is singled out and both the functional and non-functional components are explained.

The structure of the analysis that follows is this: for each of the identified components we first present the relevant functionality that is collected and grouped from Section 4 and then we provide the description of the component and its interfaces. For the time being, the developers of the components are free to define whatever interfaces they feel right, however in the coming versions of the architecture document these will be specified based on OpenAPI and will be presented as such.

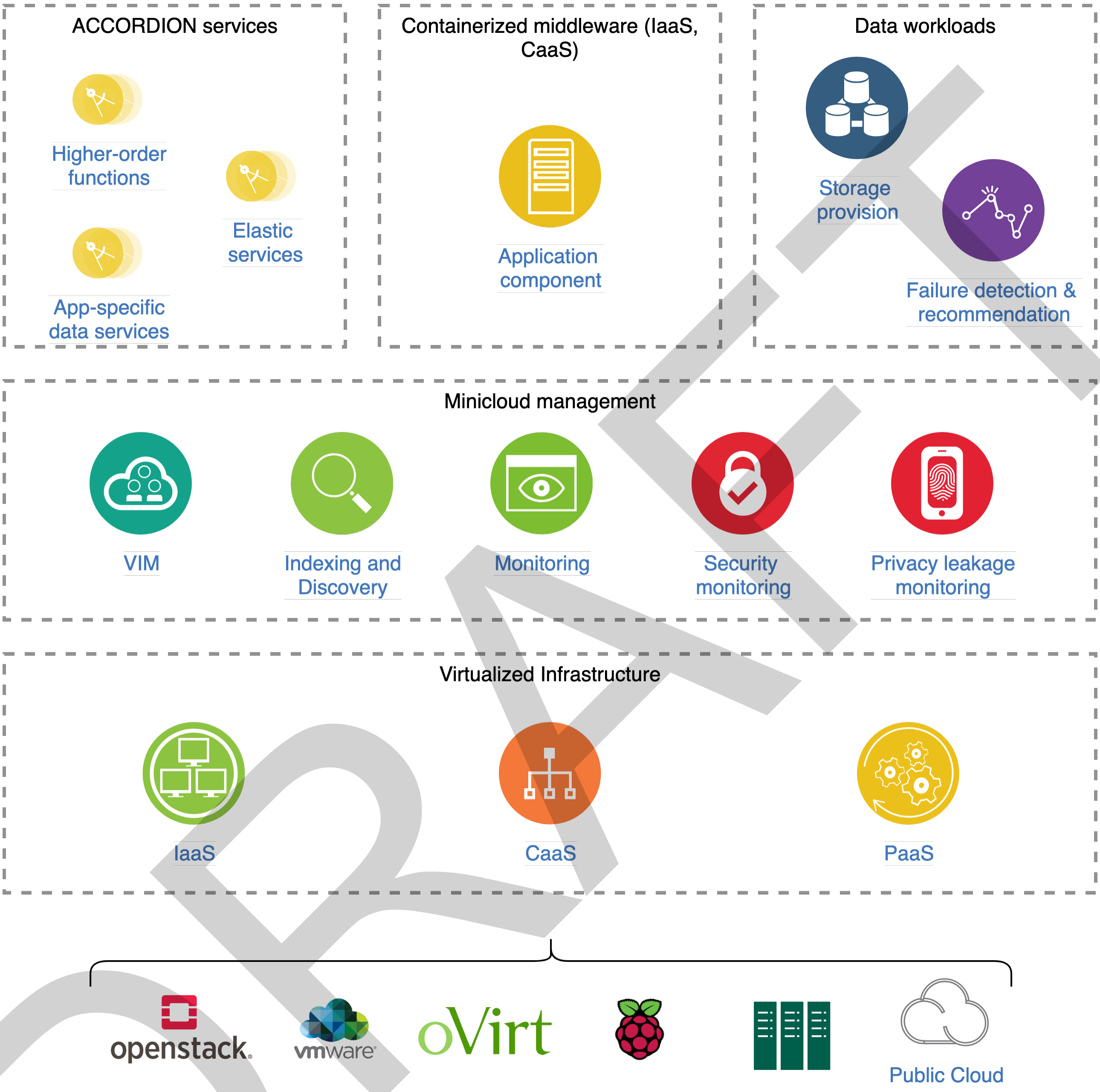
5.1 Federated infrastructure layer

This layer deals mainly with the Miniclouds, i.e., the computing unit of ACCORDION. They are composed of several internal components, presented in Figure 7. Again, the approach is layered, i.e., the components are grouped based on the consumer of the service that they are providing. At the top layer, we have all the components that the application needs in order to be executed: the run-time, the persistence system functions and the ACCORDION data services that intelligently “mend the rips” of the cloud/edge fabric.

One layer below is the Minicloud management layer, that performs all the operations to abstract and manage the underlying resources.

The Virtualized Infrastructure layer intervenes between the actual resources and the management layer. The role of this layer is to highlight that ACCORDION builds on top of existing infrastructures that support at the very least the runtime environment for executing code, VMs and/or container images as well as management services such as allocation of resources, authorization and accounting, deployment, etc.

DRAFT



DRAFT

Figure 7: Minicloud Architecture View

The following of this Section is focusing on the presentation of only the ACCORDION components of the Minicloud management layer and the layers above. As such, the application component is not presented here.

5.1.1 VIM

Relevant functionality

- Virtualized Infrastructure Management (Task 3.5)

Description

The functionality of VIM is to manage a virtual infrastructure allowing you to modify its configuration, monitor and manage the operation of workloads in an environment where resources can vary from node to node, can be low or not standard like an ARM architecture instead of x86_64. Thus, it provides monitoring services for the processes running on the cluster nodes, maintains a configuration database and runtime status and interacts closely with virtualization systems, but also can support different CPU architectures and has small footprints in term of memory ram and disk.

As VIM implementation, we choose K3S[8], a light weighted Kubernetes distribution. The choice was driven by several criteria like, but not only: licencing, developer community, security feature, project maturity, extensibility, hardware requisite. K3S has all the features of K8S[9] and a series of improvements designed to better adapt to run on clusters that can be made up of hosts with few resources, as often happens in the IoT. So, it is distributed with a single small binary file, obtained by eliminating old code such as deprecated API or non-essential API as alpha versions that are not enabled by default. Moreover, are catted out all non-default admission controllers, in-tree cloud providers and storage drivers which can be added by users as they needs.

In order to reduce RAM usage, the K3S processes have been reduced and merged, has been introduced compatibility with the SQLite DB[10] which also simplifies maintenance. For same reason also the Docker[11] runtime has been replaced with Containerd[12] which is a much more efficient runtime, losing some features such as libnetwork, swarm, Docker storage drivers and other plugins.

As Kubernetes distribution, K3S support execution of Docker container trough Containerd, to add virtual machine execution features, in ACCORDION installation will be used also KubeVirt framework. KubeVirt is a set of API and processes running in a K8S cluster (and thus K3S cluster also), both on master and worker nodes, that enable a new set of K8S managed object: the VM and VMI, virtual machine and virtual machine instance: It enable to create, delete, start, stop, monitor, migrate to different node virtual machines. Also, for choosing this software component has been done a screen of different available project using similar criteria used for VIM comparison, the choice was KuberVirt also for its project maturity and developer community, this project has started by RedHat and it's now a CNCF Sandbox project[13]. KubeVirt is compatible with Kubernetes (>=1.10) and derivative.

Interfaces

DRAFT

[8] https://k3s.io
[9] https://kubernetes.io
[10] https://www.sqlite.org
[11] https://www.docker.com
[12] https://containerd.io
[13] https://www.cncf.io/sandbox-projects/

K3S is a Kubernetes distribution, so it inherits the same K8S interfaces, which has been lightened cutting out deprecated or alpha version API. The programmatic interfaces are composed of different RESTful APIs enabling the user to manipulate Kubernetes object for example to create, delete or update. As RESTful API uses standard HTTP method: GET, POST, PUT, PATCH, DELETE. Three different object representations are allowed supported: JSON, Protobuf, Table. Also, a change notification mechanism is available.

We can group those API into main arguments: configuration and storage, workloads, scheduler.

Configuration is a set of APIs that enable user to inject data to an application via "ConfigMaps", store and provide password or token necessary to applications with "Secrets", and with "Volumes" provide a persistent external filesystem to containers. It is possible to write and deploy plugin exposing new storage systems: for doing so it is necessary to implement Container Storage Interface (CSI).

Workloads resources are managed by controllers that create Pods wrapping containers and manage their dependencies, there are several types of controllers the most common are: deployments, statefulsets, jobs.

In order to distribute workload, to choose best node to run a pod into, there is the component "Scheduler" that working with scheduling policies and scheduling profile enable a better control of workload scheduling of available resources.

The list of API described before is not complete but a meaningful set, plus RestfulAPI isn't the only interface available, actually also a CLI utility is available: “kubectl”, it enables every operation on a K3S cluster with the right authorization.

KubeVirt Interfaces

KubeVirt itself has its own interfaces: a CLI interface which is “virtctl” and a Restful API, this interfaces enables client to perform all operations to work on KubeVirt objects in similar way Kubernetes API work.

All above interfaces above are secured with https protocol and client authentication and permission enforcement.

5.1.2 Indexing and Discovery

Relevant functionality

- Indexing and Discovery (Task 3.2)
- Minicloud Membership Management (Task 3.5)

Description

The component “Resource Indexing and Discovery” (RID) provides an up-to-date view of the status of computational resources, possibly described by static and dynamic features, that are available among the various miniclouds. The component also provides functionality for retrieving such status effectively. Ultimately, any ACCORDION component that wants to find resources with specific computational features can rely on this functionality.

DRAFT

We envision the RID as a distributed component, in practice implementing a distributed data structure. This means that there is an instance of the RID running in every minicloud of the ACCORDION federation. Each RID instance connects to the local monitoring component and injects status data into the data structure; likewise, each instance can accept and reply to queries. The instances communicate with each other via internet connection.

Internally, the RID is composed of the following components:

- An interface toward the local monitoring. It is essentially a client that will pull from the local (i.e., in the same minicloud) monitoring instance (Resource monitoring & characterization component) to get resources status data.
- The local storage component keeps the data received from the monitoring interface component. Distributed indexing techniques are used to distribute data among all their instances, in order to speed up retrieval.
- The Query mapping component. The component is aimed at transforming the incoming query request into the necessary software artifacts in order to process it in the system.
- The Topology manager component. The topology manager is managing the execution of the query on top of the distributed topology of the system.

Interfaces

Each RID instance exposes a HTTP interface that allows other ACCORDION components to submit queries on computational resources. The current version of the RID interface accepts 'POST' requests with the queries in the JSON format. Likewise, the results of the query are returned in JSON format. At the current status, the following types of queries are supported: (i) multi attribute range queries on numerical values, e.g., available RAM; (ii) exact queries for string values, e.g., specific GPU chipset names; (iii) Boolean query for specific feature availability, e.g., availability of a GPU. The available interfaces of the component will be subject to changes with the progress of the project and changing requirements of other ACCORDION system components.

5.1.3 Monitoring

Relevant functionality

- Monitoring and Characterization (Task 3.1)

Description

The resource monitoring & characterization component has to monitor and characterize the resources as other components will query for information to perform actions based on metrics. Monitoring and characterization won’t have a central component instead it will be done on the Edge. Monitoring is going to be repetitive whereas characterization is static, as it will have information for the hardware of the Edge devices. Monitoring should be able to monitor both physical (devices) and virtual layer (VMs, containers, pods). In addition, it should be able to characterize and monitor heterogenous devices with different computational power.

DRAFT

DRAFT

For the monitoring part we use Prometheus[14] and Grafana[15]. Prometheus is a popular open-source system that performs monitoring. Prometheus uses a time series database to store metrics in a key-value pair format. It has its own query language named PromQL which Grafana uses to present realtime graphs. Prometheus has a pull model which basically pulls metrics from exporters. Exporters can fetch statistics from non-Prometheus systems and convert them into Prometheus metrics. To pull metrics from the exporters Prometheus must know the targets through service discovery or static configuration. To deploy Prometheus and Grafana on K3s we used a monitoring stack that we found on Github[16]. This project uses Prometheus Operator to manage and configure Prometheus instances on Kubernetes / K3s. Prometheus Operator can automatically generate monitoring targets configuration, so each node of the cluster will have exporters to expose their metrics to Prometheus which will be installed to the master node of Kubernetes. For bare metal or VM monitoring the node exporter pods expose the required metrics. Kube-state-metrics expose critical metrics about the condition of a Kubernetes cluster, it generates them from the Kubernetes API server. This project also uses Prometheus adapter which is an implementation of the Kubernetes resource metrics, custom metrics, and external metrics APIs.

To be able to characterize resources every node of a cluster has to host a char-agent container which identifies the characteristics of the device and exposes them via an API in JSON format. The master node of a K3s cluster is the one who collects the information from the characterization-agents and stores them in a MongoDB database. Char-agents give information about device name, UUID, CPU (arch, bits, cores), the region (continent, country, city), GPU model (model name, type: dedicated/integrated, video memory, unified memory), battery (details or None), RAM (in bytes), K3s node role (name of role), Disk (disk device, fstype, mountpoint) and OS (name, version) for each node.

Interfaces

The monitoring API will provide to other ACCORDION components monitoring and characterization information. As it is a REST API the format of the results is JSON. Depending on the call that monitoring API will receive, it will query Prometheus for monitoring results of MongoDB[17] for characterization results. For monitoring information, the path for the calls is /monitoring and the HTTP parameter is metric. In the metrics/parameters that are now supported for the physical layer are bare metal CPU usage, memory usage, filesystem usage, disk write latency, disk read latency, time spent for disk IO operations, disk size and disk free space. There is also the physical_metrics parameter which returns all the above. In case of virtual layer monitoring the parameters that are currently supported are pods CPU usage, pods memory usage, pods status phase and pods info. The equivalent parameter to physical_metrics is the virtual_metrics which returns all the virtual metrics in one JSON response.

In case of characterization results the path is /characterization and the supported parameter is the format. The result can be returned in JSON with REST calls or as a TOSCA YAML[18] downloadable file. In both cases the

[14] https://prometheus.io
[15] https://grafana.com
[16] https://github.com
[17] https://www.mongodb.com/
[18] http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.1/TOSCA-Simple-Profile-YAML-v1.1.html

response would be a description that would present a K3s and its devices. The results are related to the K3s cluster that monitoring component is installed to.

5.1.4 Security monitoring

Relevant functionality

- Deployment and Runtime Security Conformance Evaluation (Task 4.4)
- Runtime Security Conformance Evaluation (Task 4.4)

Description

See Section 5.2.3

Interfaces

5.1.5 Privacy leakage monitoring

Relevant functionality

- Deployment and Runtime Privacy Preserving Conformance Evaluation (Task 4.5)
- Runtime Privacy Preserving Conformance Evaluation (Task 4.5)

Description

The privacy-preserving component (PPC) is responsible for guaranteeing users' privacy at various levels within the Accordion infrastructure. This component will carry out (at least) three main functions. First, PPC will provide the necessary mechanisms to ensure that containers can be correctly executed atop a network infrastructure without the network administrators being able to infer any information about them (e.g., which type of application runs inside them). Second, PPC will enable the generation of machine learning (ML) models that provide high accuracy while at the same time being resistant to attacks that aim to infer private information from the ML model. This includes privacy-preserving mechanisms suitable for federated learning. Finally, PPC will allow the detection of user data leakage to unauthorised third parties by passively or actively monitoring users on devices and user components (e.g., browsers or containers). All these privacy features can either be enabled by default within the Accordion infrastructure or be offered as a service, e.g., to customers who run sensitive workloads.

Interfaces

We envision a scenario where the ACCORDION platform owner - possibly in collaboration with the Edge infrastructure provider - can give its customers the possibility to running their containerised applications in a confidential and isolated manner protected by a trusted execution environment (TEE). However, the use of a TEE alone does not provide strong privacy guarantees to containerised applications. This is because TEEs do not conceal the interactions between the container and the host kernel, which, based on our hypothesis, could be used to uniquely identify the applications running inside containers. To prevent this, PPC will

DRAFT

examine the syscall patterns of containerised applications. It will take a container image as input and will output a privacy score that determines how unique the syscall pattern of the container is (compared to syscall patterns of other containers). In case a containerised application can be accurately fingerprinted, PPC will additionally provide a set of recommendations to make the syscall pattern of the containerised application less unique, e.g., by adding a small amount of noise in the form of dummy syscalls.

We envisage the design and implementation of generic techniques to create privacy-preserving ML models that provide high accuracy without incurring a high overhead. The techniques employed and the way these privacy-preserving ML models will be generated will vary depending on the use case, the desired level of privacy and the considered threat model. Yet, in all cases, PPC will take as input training data (or locally generated ML models) from different entities and will produce a privacy-preserving yet accurate ML model as output. When doing so, PPC will apply a combination of privacy mechanisms and technologies, such as differential privacy and TEEs, such that the ML model is resistant to privacy attacks and the accuracy of the ML model is (at worst) only reduced minimally.

Finally, in order to detect user data leakage, PPC will collect data exchanged between different entities and will analyse the obtained information at distinct levels (e.g., in the application or the network layers). As before, depending on the use case, the considered threat model and the desired level of privacy, PPC will take as input various plaintext and/or encrypted data collected while users browse the Web or while containers communicate with each other (among others). PPC will then output a set of indicators and privacy metrics that will be used to estimate the privacy leakage. Based on the existing privacy leakage, PPC will then apply various types of obfuscation techniques at different levels in order to enhance the users' privacy. Optionally, PPC will also notify the appropriate entities about potential privacy violations that can occur.

5.1.6 Storage provision

Relevant functionality

- Storage Provision (Task 3.3)

Description

The Edge storage component has the goal of providing an edge storage framework that can support the QoE needs of the users, optimizing resource usage in the edge devices and networks. We have two possible base technologies for this; the MinIO[19] and OpenStack[20] platforms that enable us to create highly distributed, lightweight and scalable storage clusters, using Kubernetes as an orchestrator. The final choice of the tool will be made after running a number of experiments, testing their effectiveness and optimizing their configuration for scenarios close to the real use cases that the ACCORDION will be called to handle. The tool will then be modified in order to optimize its deployment and functionality in order to achieve the best QoE possible, taking into consideration the requirements set by the ACCORDION use cases.

DRAFT

[19] https://min.io
[20] https://www.openstack.org

The component will use the Kubernetes ecosystem by using the Kubernetes master as the storage controller, storage UI access point and Prometheus master for the specific cluster. Each node that is connected to the Kubernetes cluster has the potential of becoming a storage worker for this cluster or/and for the ACCORDION ecosystem. This is enabled by defining a custom label for the node, enrolling it as a storage worker. As a node we define a Kubernetes node, which can be a PC, laptop, IoT device or any other compatible device. In order to be eligible for the role of storage worker a node must have sufficient hard disk space available. The amount of space is highly dependent on the use case, so it is not pre-configured. The following figure (Figure 8) depicts a high-level architecture of the component.

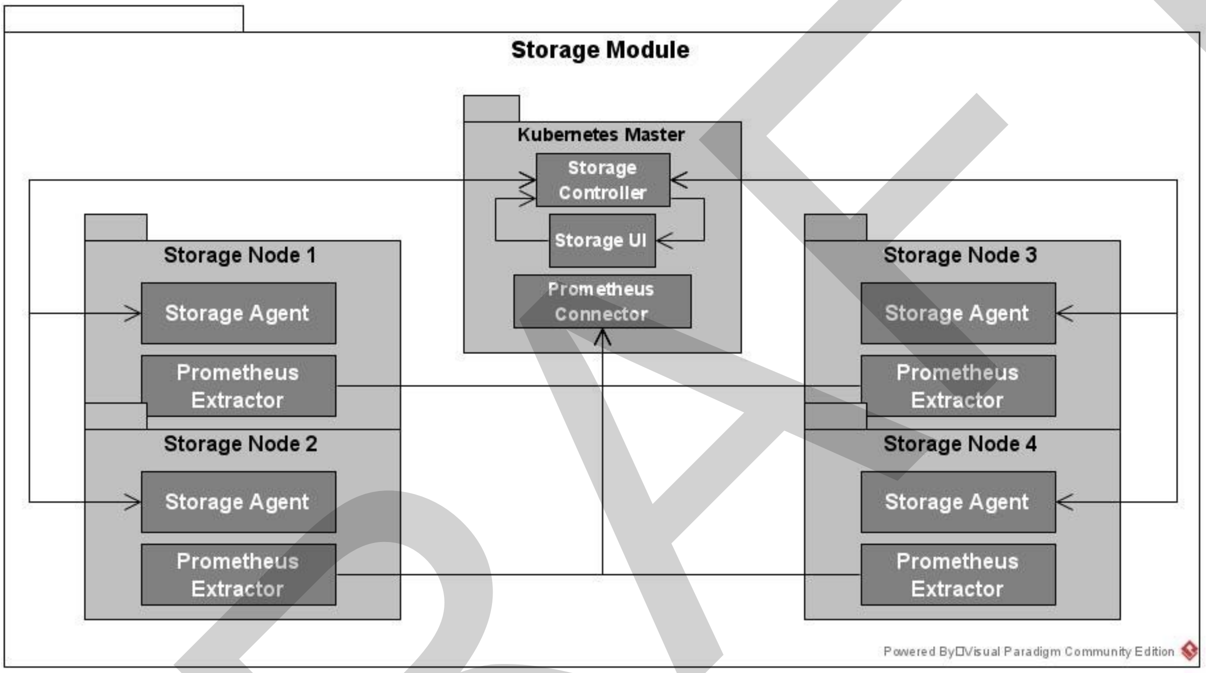


Figure 8: Edge Storage high level architecture.

After the choice of the appropriate technology and configuration a middleware layer will be added between the VIM and storage components in order to expose specific, role-based APIs that ensure security of the data, integrity of the system, optimized QoE for the users, fault proofing, fault tolerance, intelligent caching and other relevant functionalities.

Interfaces

We have isolated four actors that are using the services of the Storage module; the VIM, the Prometheus Aggregator, the Infrastructure Administrator and the ACCORDION Platform Administrator. VIM will be using the APIs exposed by the component in order to perform automated or semi-automated processes or even expose the functionalities in other interfaces or components. A draft of the APIs that will be exposed by the component is included in Table 1 at the end of this chapter. The Prometheus Aggregator will access the endpoint provided by the cluster Prometheus master in order to scrape the data and collect them in an aggregated database that collects information from all the ACCORDION miniclouds. The Min-cloud Administrator will be using the storage UI in order to manage the storage cluster and the data in it for administrative purposes. The ACCORDION Administrator will also be using the storage UI in order to manage

DRAFT

and monitor the data and the cluster in accordance with the general ACCORDION needs. In the following UML we can see a visual representation of these relations.

The following table lists the indented endpoints for Edge Storage, providing interfaces to the aforementioned actors. The standard AWS S3[21] APIs will also be available as they are supported in both proposed base technologies.

Table 1: Edge Storage indented endpoints.

Endpoint	Method	Inputs	Outputs	Description
data/{{filename}}	GET	None	Binary File	Request a file using the ACCORDION edge storage system
data/{{filename}}	POST	Binary File, Path	Confirmation	Send a file for storage in the ACCORDION edge storage system
data/{{filename}}	DELETE	Path	Confirmation	Delete a file from the ACCORDION edge storage system
cluster	GET	Cluster ID	Cluster Status	Gets the status and some monitoring data about the cluster
cluster	POST	JSON Configuration	Access Details	Creates a new edge cluster with the configuration provided or updates an existing cluster with new configuration
cluster	DELETE	Cluster ID	Confirmation	Deletes an edge cluster with the ID provided
scale	POST	JSON Configuration	Confirmation	Scales out or up an existing cluster based on the configuration provided
scale	DELETE	JSON Configuration	Confirmation	Scales in or down an existing cluster based on the configuration provided

5.1.7 Failure detection and recommendation

Relevant functionality

- Failure detection and mitigation (Task 4.3)

Description

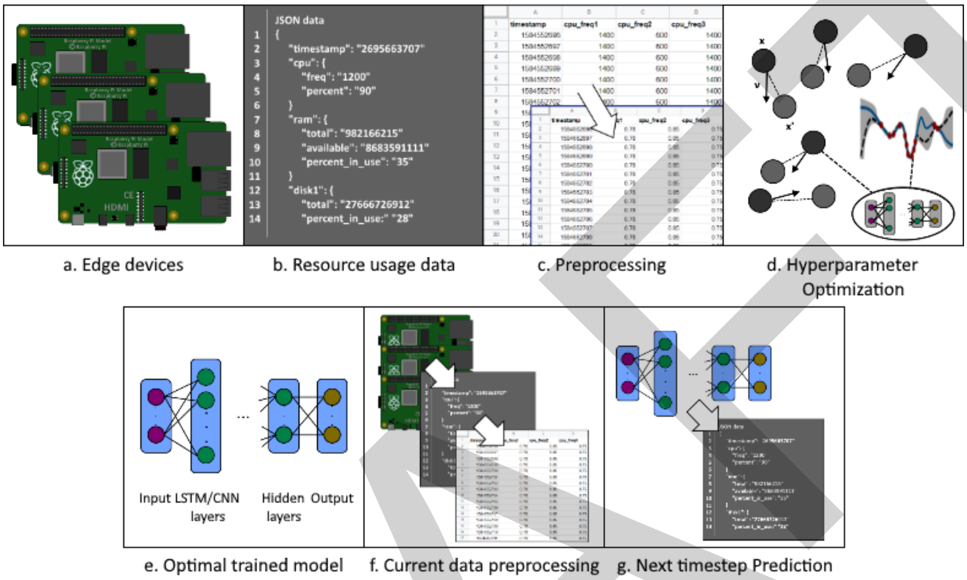
The Failure detection & recommendation component provides a proactive fault tolerance mechanism leveraging resource monitoring data. The processing edge nodes will be assessed periodically in order to predict the QoE degradation. If potential QoS deterioration is identified, the component proactively will inform the intelligent orchestrator to trigger hot and cold migration policies.

The Failure detection and recommendation component receives input from the Resource Monitoring & Characterization component and makes real time process with the resource utilization model. The resource utilization model involves a training and inference workflow as depicted in Figure 9. In the beginning, the training process receives historic data and builds a multi-output regression model as we can see in the steps *a* to *e* of Figure 9. Specifically, the resource usage data of the historic edge nodes contain parameters like

DRAFT

[21] https://aws.amazon.com/s3/

CPU, ram disk and network. The component makes pre-processing with a normalization method and pipes them to the hyper-parameter optimization process which trains and evaluates multiple neural networks in order to identify an optimal topology.



```
JSON data
{
    "timestamp": "2695663707"
    "cpu": {
        "freq": "1200"
        "percent": "90"
    }
    "ram": {
        "total": "982166215"
        "available": "8683591111"
        "percent_in_use": "35"
    }
    "disk1": {
        "total": "27666726912"
        "percent_in_use:" "28"
```

a. Edge devices b. Resource usage data c. Preprocessing d. Hyperparameter Optimization

e. Optimal trained model f. Current data preprocessing g. Next timestep Prediction

Figure 9: Resource usage prediction and QoS deterioration

In the next steps *f* and *g* is illustrated the inference process which takes the current status of resource usage, makes the same pre-processing with the training stage and provides resource utilization predictions for the next timestep. The predictions involve the resource usage bottlenecks and the potential QoE deteriorations for each processing edge node. The prediction of the QoS deterioration is related to the decision whether an edge node is reliable to process workload or migration actions should take place.

Interfaces

The Failure detection and recommendation component consumes periodically the response of the monitoring API regarding the resource usage for each node. This includes physical layer parameters such as bare metal CPU usage, memory usage, filesystem usage, disk write latency, disk read latency, time spent for disk IO operations, disk size and disk free space. The Failure detection and recommendation component publish messages for each node that contain the probability of QoS deterioration. These messages will be in json format and can be consumed from the intelligent orchestrator.

5.1.8 ACCORDION Services

Relevant functionality

DRAFT

- Define Application Workflows with ACCORDION Services (Task 5.1)
- ACCORDION Data Services (Task 5.1)
- Application Model (Task 5.1)

Description

ACCORDION services or data services, are small components that can intervene the data pathway to do filtering, interpolation, compression etc. They can be implemented as Virtualized Network Functions (VNFs) through the network orchestrators, when they deem this operation meaningful for mitigating QoS/QoE issues. They can also be explicitly requested by the application developer to be integrated in the application workflow through the application development/deployment process. For the time being, we consider operations such as higher-order functions, message queues or even interpolation and general compression functions to be provided by ACCORDION as a service. The ACCORDION services will be exposed in a microservice setting, allowing their easy and low-cost replication for meeting the needs of various applications and workflows.

Interfaces

Each of the ACCORDION Services exposes its own RESTful API while it also includes a client for the ACCORDION Event Bus.

5.2 ACCORDION Platform

The ACCORDION Platform layer deals with the functionality that supports the ACCORDION applications and the infrastructure. On its own, this layer can be broken down to more layers. Figure 10 depicts such an internal structure, focusing on the ACCORDION user of interest, i.e., the application provider. The latter uses the ACCORDION Platform Portal to allow the development and deployment of the application. To facilitate some level of automation and simplicity in the migration of applications to the ACCORDION Platform, the portal consumes a description of the application, provided by the application provider. This description is then consumed by a number of components for building the application and from another set of components for the deployment and runtime management of the application.

DRAFT

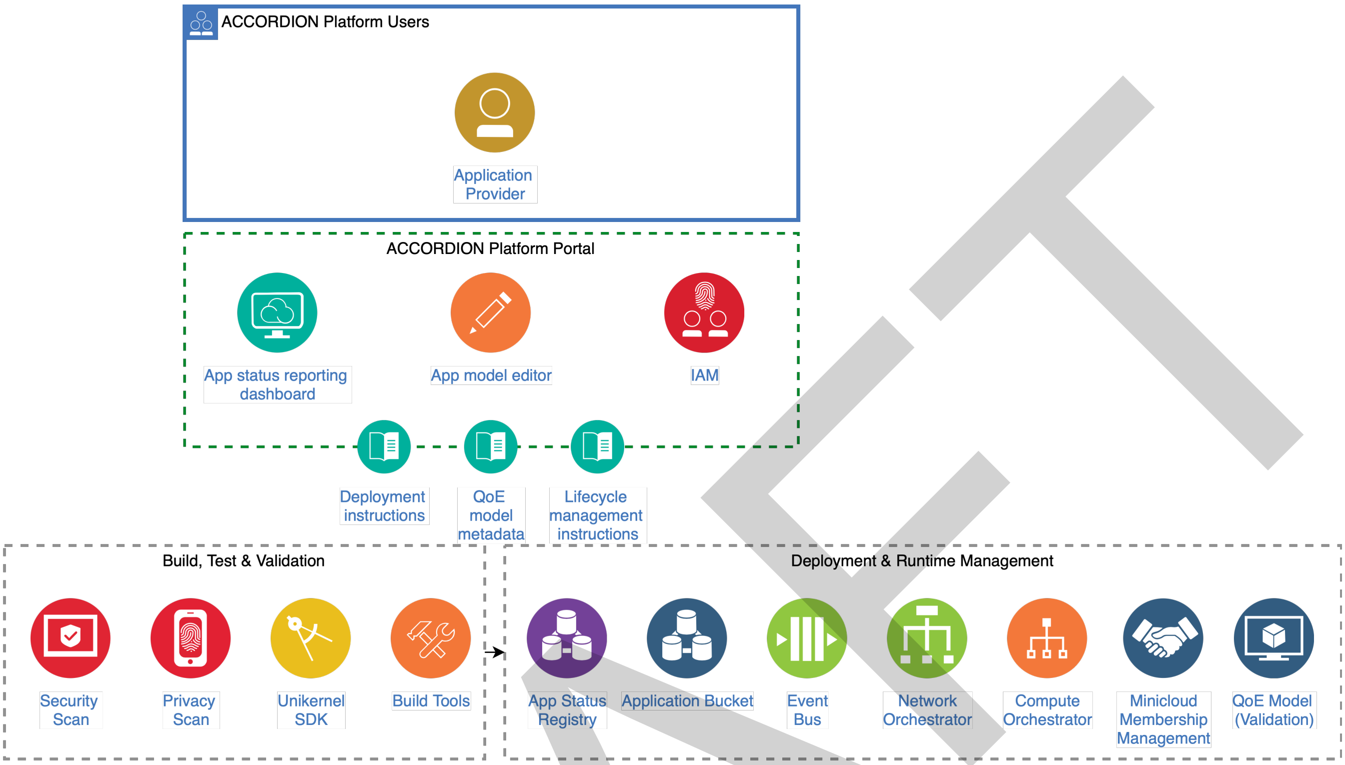


Figure 10: ACCORDION Platform Architecture View

DRAFT

Our analysis starts with the details of the initial description of the application that the application provider must provide. It then moves on with the specification of the functional components as depicted in Figure 10.

5.2.1 Application description

The initial configuration of the ACCORDION platform is necessary so as to enable the development, deployment and runtime management of the application to the highest possible level of automation. For that to happen, the application provider must provide some information about how to build the application components, how to deploy them and which workflows must be supported as the application is running.

Relevant functionality

- Application Model (Task 5.1)
- QoE Model (Task 5.2)
- ACCORDION Data Services (Task 5.1)

Application model

In the application model the main components that can be described are containers, end devices, edge devices, ACCORDION framework and the relationships between them. The goal of the application model is to automate Cloud operations. Technologies like Compose[22] and Kubernetes Resources[23] can perfectly describe containers and pods, but they cannot describe if a container/pod must be deployed on Edge or Cloud like our application model does. For each scenario of the application there would be one application model file to describe it. The application model describes both design time properties like ports or hardware requirements for the containers, and runtime properties like IP of containers and runtime application related properties like the number of players in a container. The application model may support an event driven application flow. As such, it allows the definition of events and responses to the events, in the form of the triple {target, event, handler}. In detail, the application model may contain the following.

- **Build instructions**

Docker can build images automatically by reading the instructions from a Dockerfile[24]. A Dockerfile is a text document that contains all the commands a user would execute on the command line to assemble an image. The build process starts from fetching the source code from the code repository, then using bash script a compiled package is produced. Depends on the executable file and the compile process itself, the image building instructions may vary significantly and there is no way to make one, reusable Dockerfile. Nevertheless, to make DevOps process as efficient as possible, Dockerfile should not contain any environmental properties and every image built from Dockerfile should be generic and treated as a blueprint. All environmental variables should be injected directly to specific instances built out of the docker image.

- **QoE model metadata**

The QoE model allows the estimation of the achieved application QoE levels based on some monitored metrics. At this point, the application provider needs to define thresholds for the QoE levels that the ACCORDION platform must respect, applying countermeasures. The "QoE model metadata" must possible present suggestions about these countermeasures, allowing the application provider to select from a set of predefined operations (e.g., scale out/up, etc.).

- **Lifecycle management model**

As the application model will have an event driven approach, it describes the workflow of the actions in such a way to automate the deployment of use case owners. The application model provides instructions about the order of the actions and the conditions which will trigger them. There is also a need to have a lifecycle management model to provide instructions about the events that trigger software engineering patterns or the deployment of data services. Data services are software components with a minimal resource fingerprint that can be deployed along the application workflow to improve the QoS offering.

5.2.2 ACCORDION Platform portal

Relevant functionality

[22] https://docs.docker.com/compose/
[23] https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
[24] https://docs.docker.com/engine/reference/builder/

DRAFT

- Identity and Access Management (Task 5.5)
- Submission or Creation of Application Model (Task 5.5)
- Application Status Reporting (Task 5.5)

Description

The main is goal to provide an interface to support the DevOps process in ACCORDION. The component will also visualise all the essential information about the deployments, their statuses and the properties of each deployment across the whole platform.

The Platform component is going to provide meaningful insight into the Accordion Platform and via its CRUD interface, users will be able to manage the applications. The component is going to be integrated with GitLab[25] repository, thus allowing for fetching the source code directly from there and building an instance of an application, or from docker image. Besides deploying applications, the component will be capable of running tests which has been specified in a special file provided by the user.

Interfaces

After analytics of business requirements of the platform, it was determined that Platform component should be integrated with the code repository, it should provide an interface for deployment and building process of application and finally, an interface for testing. The integration with GitLab was done by implementing OAuth2.0[26] code grant flow and using GitLab as an OAuth provider, therefore there was no need for creating the dedicated interface for authentication. The component is only going to have interfaces for deployment, building process and for application testing, though the exact details of these interfaces have not been yet specified.

5.2.3 Security scan

Relevant functionality

- Deployment and Runtime Security Conformance Evaluation (Task 4.4)

Description

The Security component aims at improving the level of security in the application development and deployment lifecycle of Accordion Applications by providing guidance and tools in the form of DevSecOps techniques, focusing in the areas of automated static code analysis (SAST), dynamic application analysis (DAST) and automated container images security inspection.

The functionalities of the Security component described in the previous paragraph can be implemented through a myriad of solutions available in the market. An evaluation of suitable software security tools and

DRAFT

[25] https://about.gitlab.com
[26] https://oauth.net/2/

methods has been conducted in the context of Task 4.4 using specific metrics considered relevant for the ACCORDION project, including license type, development activity and additional capabilities including extendibility and possibility of adoption in automated CI/CD contexts. This evaluation produced following results:

- For SAST: SonarQube[27]
- For DAST software: OWASP Zap[28]
- For Container image security: Harbor[29] (integrated image scanning tool *Clair 2*[30])

Considering a generic DevOps process, the following figure (Figure 9) shows the phases where the selected tools should be integrated.

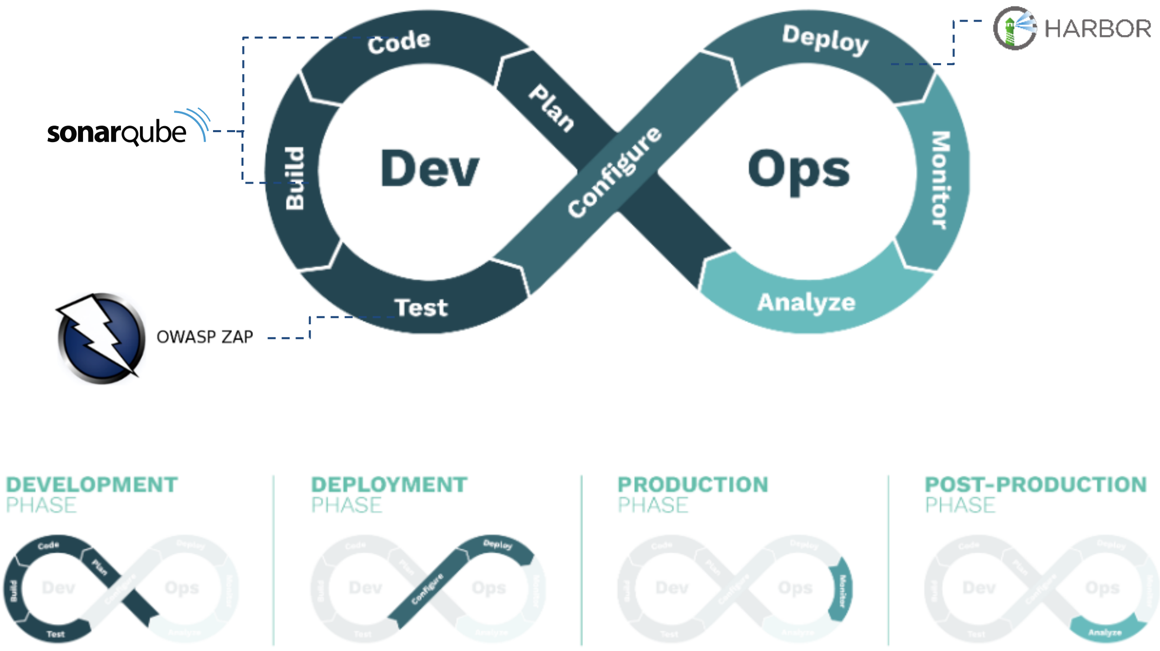


Figure 9: Generic DevOps process

At the moment, it is not yet completely decided if the development phase of ACCORDION applications will be performed in an ACCORDION environment or on premises. The latter hypothesis seems the more probable: in this case, application authors will run their development pipelines on premise (development phase) uploading artifacts into Accordion for the remaining deployment, production and post-production phases. In this case, SAST and DAST analysis tools must be integrated and operated directly on premise.

SonarQube

[27] https://www.sonarqube.org
[28] https://owasp.org/www-project-zap/
[29] https://goharbor.io
[30] https://coreos.com/clair/docs/latest/

SonarQube is a tool for automatic scanning of source code to detect bugs and vulnerabilities. It can be integrated with existing workflows to enable continuous and automated code inspection. One of the SonarQube key features is its capability of scanning more than 25 programming languages source types is particularly useful for ACCORDION where all the applications, which might greatly differ in terms of programming language and development models, can be analyzed using a single solution. SonarQube works by analyzing the source code of a given project and summarizing the results in a *snapshot*¸ which contains a set of metrics and issues for that project at a given time according to a *Quality Profile*¸ that is a configurable set of rules to be applied for an analysis. Found issues relate to potential Bugs, Vulnerabilities, Code Smells and Security Hotspots. Estimations on time required to fix Vulnerability (*Remediation Cost*) and to fix all Maintainability Issues / Code smells (*Technical Debt*) are also provided. Additional details on SonarQube basic concepts are available at *[Reference: https://docs.sonarqube.org/latest/user-guide/concepts/]*.

OWASP ZAP

OWASP ZAP (Zed Attack Proxy) is an open-source tool for web applications penetration testing. ZAP acts as a proxy between browsers and applications and can be instructed to perform attacks on application URLs both manually and automatically by crawling and scanning all the available pages with its spider. ZAP can perform both passive and active scanning, searching several known security issues like SQL injection, Cross Site Scripting and the other vulnerabilities included in the OWASP Top 10 list *[Ref. https://owasp.org/www-project-top-ten/]*. Passive scanning does not change the requests nor the responses in any way and is therefore safe to use, while active scanning attacks selected targets searching potential vulnerabilities by using known attack techniques and could potentially break the target application. The specific rules applied in the scanning tests are defined in one or more configurable *Scan Policies*. ZAP creates statistics of the attacks which can possibly be used to generate reports.

Harbor

The open-source Harbor project (https://goharbor.io/) is a trusted cloud native repository for Kubernetes that offers many interesting features like:

- Private docker image registry
- Integrated control on image signing process
- Integrated control on image vulnerabilities

We had chosen this platform as docker registry in ACCORDION because the three features above are all integrated in a single tool and no other external components are necessary to perform image signing and image security scanning.

With Harbor a docker image can be signed at “push time” using the Client DCT Notary feature; in this way Harbor is able to check image signatures automatically and independently of any Docker client environment.

In addition, Harbor integrates Clair 2 docker image vulnerability scanning tool, when images are pushed to Harbor, a Clair scan is performed, and the image is “rated” based on the vulnerabilities found.

Another strength of Harbor is that both image signature verification and vulnerability scores can be configured individually for each project. It is possible to configure setting like “allow only verified images to be deployed” (i.e., signed images) and “prevent images with vulnerability severity of Medium and above to be deployed" (images with medium-high score). This are just two examples to understand the flexibility of the harbor configuration. All this configuration is performed at Harbor server side and is agnostic respect to the docker client used.

Interfaces

SonarQube

SonarQube consists of four main architectural components: one or more SonarScanners, installed on an existing project's CI/Build Servers and performing the code analysis, a central server (providing WEB interface) for configuration and administration of the SonarQube platform and the processing of analyses and reports produced by the SonarScanners, a database to store all the data related to configurations, views, quality snapshots etc. and, finally, a set of plugins installed on the Server for managing governance, authentication, Source Code Management integration etc. An optional component, SonarLint, is also available as an IDE extension to be installed on developers' workstations to perform code analysis in real time and warning developers before code is pushed in Source code management (SCM) repository. SonarQube platform can be used with several Application Lifecycle Management and CI/CD tools and also provides web API to access and possibly extends its functionalities, making it a good choice for integration in ACCORDION development and operation pipelines.

OWASP ZAP

ZAP provides several interfaces. It is mainly used as an application running on the penetration tester machine (in the form of a standalone application or as a browser plugin), but it also provides a CLI interface and an API for all its functionalities allowing the integration of the tool in existing CI/CD pipelines. The most common way to perform automated penetration testing in CI/CD contexts using ZAP is through its Jenkins[31] plugin.

Harbor

Harbor is composed by several components which can be easily installed on Kubernetes using Helm, the main ones are:

- Postgresql[32]
- Redis[33]
- Clair
- Beego[34]
- Chartmuseum[35]
- Docker/distribution
- Docker/notary

[31] https://www.jenkins.io
[32] https://www.postgresql.org
[33] https://redis.io
[34] https://beego.me
[35] https://chartmuseum.com

- Helm
- Swagger-ui[36]

Harbor can be managed by an extensive Web UI, provide a Command Line Interface and a set of openAPI compatible APIs, it Is able to communicate with other tools (like jenkins) using secure webhooks.

API reference can be found at https://goharbor.io/docs/1.10/build-customize-contribute/configure-swagger/

5.2.4 Unikernels SDK

Relevant functionality

- Lightweight virtualization support (Task 3.4)

Description

ACCORDION will natively support Unikernels, i.e., specialised, single address space machine images constructed by using library operating systems. The developers can select, from a modular stack, the minimal set of libraries which correspond to the OS constructs required for their application to run. These libraries are then compiled with the application and configuration code to build sealed, fixed-purpose images (Unikernels) which run directly on a hypervisor or hardware without an intervening OS such as Linux or Windows. By tailoring the operating system, libraries and tools to the specific needs of the application, it vastly reduces virtual machine and container image sizes to a few KBs, drastically cutting down your software stack's attack surface.

Given the specialized nature of the Unikernels, ACCORDION will assist the application developers to migrate their software. Such assistance comes in the form of an SDK. Such an SDK will be largely based on the Unikraft build system[37]. In particular, ACCORDION will leverage *kraft*, a command line companion tool used for defining, configuring, building, and running Unikraft applications. With kraft you can seamlessly create a build environment for your Unikernel and painlessly manage dependencies for its build.

Interfaces

As stated, ACCORDION will leverage the tool “kraft” for allowing the build of Unikernels.

5.2.5 Build tools

Relevant functionality

[36] https://swagger.io/tools/swagger-ui/
[37] https://github.com/unikraft/kraft

The most core feature of this component is automating software development process by providing a set of useful tools for building, testing, deploying.

Build Application Components (Task 5.4)

Description

Since the requirements of each use case are vast and different, the ACCORDION application component building tools will be based on Jenkins. Jenkins helps automate the parts of software development related to building, testing, and deploying, facilitating continuous integration and continuous delivery. Jenkins Pipeline is a suite of plugins which supports implementing and integrating continuous delivery pipelines into Jenkins. This process involves building the software in a reliable and repeatable manner, as well as progressing the built software (called a "build") through multiple stages of testing and deployment. Pipelines provide an extensible set of tools for modelling simple-to-complex delivery pipelines "as code" via the Pipeline domain-specific language (DSL) syntax. The definition of a Jenkins Pipeline is written into a text file (a Jenkinsfile) which in turn can be committed to a project's source control repository. In Jenkins' pipeline, a bash script to build an executable package out of source code will be executed.

Interfaces

The component will use Jenkins' API, especially for the visualisation of DevOps process and for executing pipelines.

5.2.6 Application status registry

Relevant functionality

- Application Registration (Task 6.4)

Description

This main role of this component is to store the information about the lifecycle of the application. Each application will have an instance of the application registry. Beside the information about the lifecycle, the registry also keeps track about the resource in which the services of the application are running. The application registry is updated by the orchestrators (in case of the startup or change of the application deployment) and via the dashboard (for example when the application provider forces a manual stop of the application).

An important distinction must be made between the Application Registry in ACCORDION and the Container Registry technologies that are common to store containerized images (e.g., Harbor). As written above, the former contains the metadata referred to the application to support the work of the orchestrator; The latter allow users to manage containers throughout their applications and networks, also providing fine-grained access control to individual containers. A similar functionality in ACCORDION is provided by the Application Bucket (see 5.2.7).

Interfaces

The component will be listening on the ACCOORDION event bus for information about possible changes in the status of the application. In addition, the component exposes REST API for a seamless integration with the web dashboard of ACCORDION.

5.2.7 Application Bucket

Relevant functionality

- Submission or Creation of Application Model (Task 5.5)
- Application Component Registration (Task 6.4)

Description

After an application is built and end-user wants to deploy it, the ACCORDION platform should provide storage from which the image might be fetched from. The component will consist of two services, one of them is a storage of image registry, the other one is storage from application model properties. Docker images will be kept in a Docker registry, which can be provided e.g., by GitLab. Application model properties will be stored in a database.

Interfaces

The component will have interface for performing CRUD actions on applications like e.g., saving new application or getting one.

5.2.8 Event bus

Relevant functionality

- Event Bus (Task 6.4)

Description

This component provides the communication fabric that interconnects many components of the ACCORDION ecosystem. In particular, the component will support:

- platform-to-platform communications, i.e., those communication that happens between two services of the accordion platform;
- app-to-platform communication, i.e., a specific channel thought which the application can communicate with the accordion platform, for instance to signal possible QoE degradation.

To support these communications, several event bus instances might be present at the same time. Beside the single instance that is needed to support the platform-to-platform communication, there is an instance of the event bus for each application to support the app-to-platform communication.

The even bus is implemented according to a publish subscribe model. In this model the producers send messages to the event bus connected to a specific topic. The messages are then stored inside the event bus, waiting to be delivered to the consumers. The consumers subscribe to their topics of interests, and then receives messages according to their preferences. A well-known implementation of this model is provided by Apache Kafka[38], which is currently considered as the tool of choice for ACCORDION.

Interfaces

The event bus interfaces strongly depend on the specific tool used for its implementation, in this case Apache Kafka. There are different packages available for most programming languages that allow producers and consumers to interact with the system, including Java, Python, C#.

5.2.9 Compute resource orchestrator

Relevant functionality

The ACCORDION federation is composed of a "continuum" of (physical or virtualized) resources belonging to Edge Miniclouds (EM) or public Central Clouds (CC). The main functionalities of the Compute resource orchestrator (CRO) are:

- allowing the allocation of resources to the various application components, considering the applications' QoS/QoE requirements and their internal topologies
- composing a Service Allocation Plan (SAP) for all the required application components
- modifying or composing a new SAP when a QoS/QoE violation or another type of disrupting event is detected and signaled by the VIM Monitoring Agents or other ACCORDION services and executing the suitable recovery actions (migration, scale up/down, etc.)

Description

To maintain a global vision of the resources of the federation and of the placement of all the application components, the CRO is logically viewed as a **centralized** component. Although the high level logical abstract architecture of the CRO is viewed as centralized (single instance), its implementation can be instead **decentralized** (multiple instances), to increase fault tolerance and scalability respect to the size of the federation. An instance of the CRO is deployed on top of the VIM of each EM or CC and manage in an exclusive way the local group of resources which are called the **associated resources**.

The CRO instance, when received an application deployment request, must be able to solve a **Multi-resource Multi-objective service allocation problem (MRMO)** over the associated resources. To do so, the CRO must build an internal representation model of the problem: the most common models used in literature are the **Mixed Integer Linear or Non-Linear Programming model (MILP or MINLP)** and various **Graph Models,** which have been investigated. Independently from the chosen model, inputs for it are the application model of the application to be deployed, retrieved by the associated Application Bucket, and the resources availability

[38] https://kafka.apache.org/

model, which is obtained by querying the Resource Indexing and Discovery (RID). The MRMO problem is then solved exploiting algorithms which use approximated techniques**:** these algorithms produce a near-optimal solution with a low level of computational effort, exploiting heuristic, random or other types of techniques to reduce the solution space. Examples are genetic algorithms, iterative search algorithms or vote and consensus algorithms, which will be investigated. Such class of algorithms is well suitable for constrained resources environments such as Edge Clouds.

One or more valid SAPs (Service Allocation Plan) are computed and then executed, one at a time, by the Allocator component of the CRO: in case a QoS/QoE violation is detected, or a disrupting event occurred, i.e., a security violation, a failure of a hardware component or a risk of a privacy data leakage, detected and signalled to the CRO by the properly ACCORDION services, the Allocator could execute and/or compute another SAP. When an SAP failed, the Allocator could execute a set of one or more recovery actions. Such QoS/QoE violation or disrupting event could happened also during the lifecycle of the application, causing a modification to the executed SAP.

If no SAPs are computed or all the SAPs failed to be executed by the Allocator, the CRO could try to find additional resources to satisfy the application deployment request from other group of remote resources, belonging to other EMs or CCs, managed by a different CRO instance: in this case, a possible solution technique is to contact a selected group of other CRO instances, chosen following certain selection criteria, creating an **Aggregation.** An MRMO service allocation problem is then solved over the union of all the associated resources of the CRO instances of the Aggregation: distributed algorithms are well suitable to solve this problem efficiently, or a leading CRO instance could be elected or created from scrap, forming a possible hierarchical architecture of CRO instances. All the critical issues relative to form, maintain and manage the lifecycle of such Aggregations are investigated.

Interfaces

The main input interface for the CRO provides an entry point for the application deployment requests, both for its entirety or for one or more of its components and it will be developed as a REST-like API. Its main input parameters are constituted by the information to identify a single application or application component, which had been built and deployed by the Application Provider. In the same API are also specified all other typical IaaS functions of an orchestrator, such as migration, scale up/down, stop, etc. which are not internally generated.

Internal platform-to-platform interfaces are defined with other components: an interface with the internal ACCORDION event bus is needed to receive asynchronous events such a privacy leakage, a failure of a hardware component, the QoS/QoE deterioration for an application component or a security violation which are sent by respectively the Privacy preserving application component, the Failure detection and recommendation component and the Security application monitoring component. The characteristics of this interface depend on the technology used for the implementation of the ACCORDION event bus. There are also internal interfaces with the Resource Indexing and Discovery component to submit queries to retrieve the availability of remote resources, and with the VIM and its Monitoring and Network agents to know the availability of the associated resources and the estimation parameters about traffic congestion, latency or bandwidth of the network connections.

Finally, another internal interface with the Application Bucket is needed to retrieve the application model of an application to be deployed and to update its status each time the orchestrator needs to change it. These interfaces have not been defined yet, and they are currently under development.

5.2.10 Network resource orchestrator

Relevant functionality

- Network Resource Orchestration (Task 4.2)
- AI-based automated orchestration framework of network resources

Description

Alongside the Compute Resource Orchestrator (CRO), the Network Resource orchestrator (NCO) also takes part in the deployment and runtime of ACCORDION applications. This component assures that the deployment and runtime of applications do not violate the network requirements of these applications. The role of this component is to provide a multi-domain AI-based orchestration framework of the network elements by ensuring reliability and latency. This component provides automated orchestration and intelligent management operations and facilitates the life cycle management of the network slices with the aim of rapid slice creation and activation, enabling application developers, Use Case owners (In the case of ACCORDION: ORAMA VR, ORBK and PLEXUS) to define blueprints for their VR/AR ready slices. This component relies on monitoring the network resources at the edge/public cloud for any potential QoS degradation (e.g., congested links, node capacity excess, etc.) and accordingly plan operations to fix these issues (e.g., service migration, remove congested links, scaling, etc.) and guarantee the network requirements of these applications. To enable self-configuration and self-optimization capabilities of network resources, this component considers the exploitation of machine learning techniques and their integration in the ACCORDION framework.

Interfaces

This component comprises two interfaces with (i) ACCORDION internal services, and (ii) application developer. The former consists of data signalled by other internal services which could periodic such as monitoring data or unexpected events such QoE deterioration. The application developer inputs comprise the application itself and its blueprint. The application developer provides the application as a container (ex: Docker container) and the blueprint in the form of a YAML file. This component should also provide an output interface, a set of functions, that gives an app developer more flexibility to decide how and where applications should be deployed. Communication between internal components and between NCO and the application developer could be achieved via a Rest-API interface and event bus. The available interfaces are subject to change with the progress of the project.

5.2.11 Minicloud membership management

Relevant functionality

- Minicloud Membership Management (Task 3.5)

Description

While adding a node is almost automatic, you just need to install the K3S agent with server configuration, deletion needs explicit commands to remove running workload containers, and then delete node.

Interfaces

To modify cluster configuration, the entry points are master nodes via CLI or Rest API hosted on master nodes. There is one main CLI command “kubectl” and several rest API.

To remove nodes there are two options:

1. By command line:
    a. kubectl drain <node name>
    b. kubectl delete node <node-name>
2. Via Rest api
    a. Eviction API (restfull): /api/v1/namespaces/default/pods/<node name>/eviction
    b. Node api: DELETE /api/v1/nodes/{name}

To add a node:

Node registration is automatic at K3S agent startup, to successfully complete registration the process must know master IP address and cluster token, parameters the must be configured as environment variable: K3S_TOKEN, K3S_URL or as service start arguments.

5.2.12 QoE model (validation)

Relevant functionality

- QoE Model Validation (Task 5.2)

Description

Within the ACCORDION project, three QoE models will be developed to continuously measure the quality of ACCORDION use cases by monitoring the network, compression, and client parameters. The model predicts the QoE of each use case on a 5-point Absolute Category Rating (ACR) scale based on the network parameters as well as compression parameters.

The models are decided to be a parametric model that gets network, compression, and client parameters as input for quality prediction. Since each use case has its own characteristics, the model will be developed and employed separately and independently from other use cases. However, all models follow the same structure in the development phase, which is described in D5.1. The development of model will be done in different iterations where in each iteration, a draft version of the model will be developed with limited scope and range of parameters, and then the developed model will be evaluated. Based on the result of the evaluation, more data will be collected to improve the model but also to ensure that the model can be integrated into the ACCORDION platform in the early stage of the project. So far, a draft model is developed for the ORBK use case, which is described in WP5.

The models are taking the network, compression, and client parameters as input parameters to predict the quality of ACCORDION use cases. The input parameters must be provided by use case owners as described below:

Network Parameters: The parameters collected from the network are delay, jitter, packet loss rate, and burst rate.

Encoding Parameters: In addition to network parameters, multiple encoding parameters will be collected for the use case with video streaming components (OVR use case). Parameters such as encoding bitrate, encoding resolution, encoding frame rate can be considered as required parameters for encoding. It must be noted that in more advanced monitoring services, bitstream information could also be collected, such as packet header information (e.g., I-frame and P-frame size) or even payload bitstream information (e.g., quantization parameters). For the first draft model, only encoding parameters will be considered, while, for other iteration, bitstream information will also be considered.

Client Parameters: The client information includes information about the device or application that is running on the client-side. For example, the size of the display or input device plays an essential role in quality assessment. In the ORBK use case, the information about sensitivity of the game towards delay is considered as an input of the model in the first draft of the model.

The QoE model will be used to monitor the quality of the ACCORDION use cases in which the quality will be predicted in R-scale range from 0 to 100 (where 0 is the worst quality and 100 the best quality) or transformed R-scale to 5-point ACR scale (where 1 is the worst quality and 5 is the best quality). In order to get more insight into the root of low-quality prediction, diagnostic scores will be provided for each use case. The diagnostic predictions are described in D5.1, which could be beneficial to track the cause of errors or low quality of a specific use case, consequently, decide on the strategies that need to be implemented in response to the low quality. As an example of such diagnostic information, the transmission impairment will be predicted how the interaction of the user with a use case is affected by packet loss. The model predicts the diagnostic information in R-scale where 0 is the worst quality and 100 is the best quality.

Interfaces

The QoE model validator will provide a RESTful API for the provision of the input parameters. In fact, we need a single endpoint accepting a POST HTTP request with the parameters in its body (application/json). The HTTP response will bear the QoE metrics' values in a JSON array.

6 Sequence Diagrams

In this Section we create a set of sequence diagrams with the intention to explain the interplay of the ACCORDION components (Section 5) when implementing the platform operation scenarios (Section 4).

6.1 Join Federation

The sequence diagram in Figure 11 depicts the high-level interaction between the involved entities when an infrastructure owner decides to join the minicloud federation. This flow is not implemented in the context of ACCORDION however it is specified in D7.5 and to an introductory extent in Section 4.1. ACCORDION assumes that the procedures depicted in the sequence diagram have already been performed and that the minicloud federation already contains an adequate number of miniclouds.

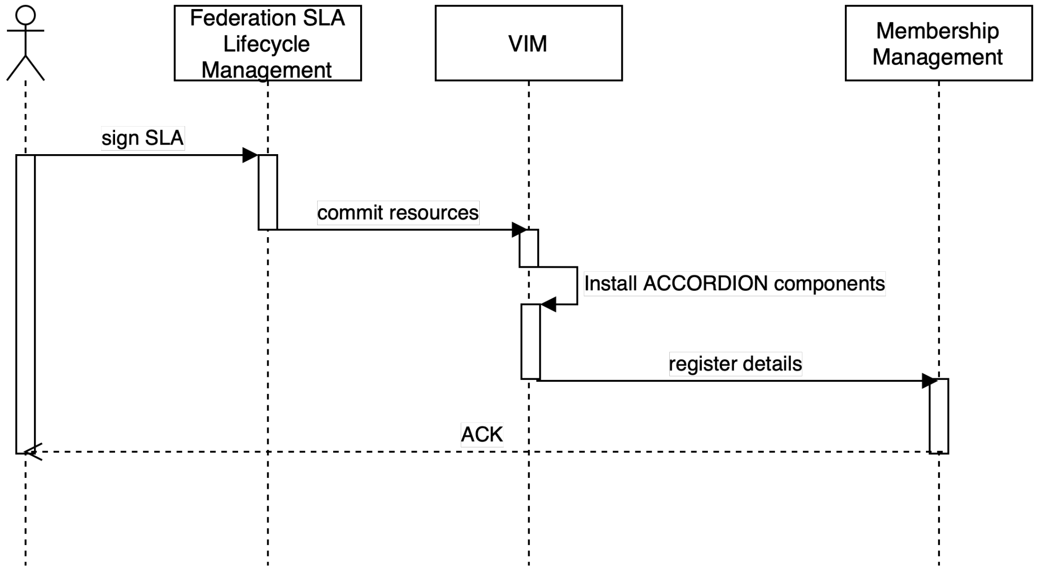


Figure 11: Flow of logic for joining the minicloud federation

6.2 Deploy Application

The sequence diagram in Figure 12 shows which components are activated and interact when the application developer uses the ACCORDION Platform to build and test the application components. In this scenario we use the term “test” in a slightly different context than in a typical DevOps scenario. Apart from the traditional unit testing for the code, ACCORDION also applies security and privacy evaluations and provides a report on it results to the application developer together with recommendations. Note that the application description is focusing on the part of the application model and the build instructions.

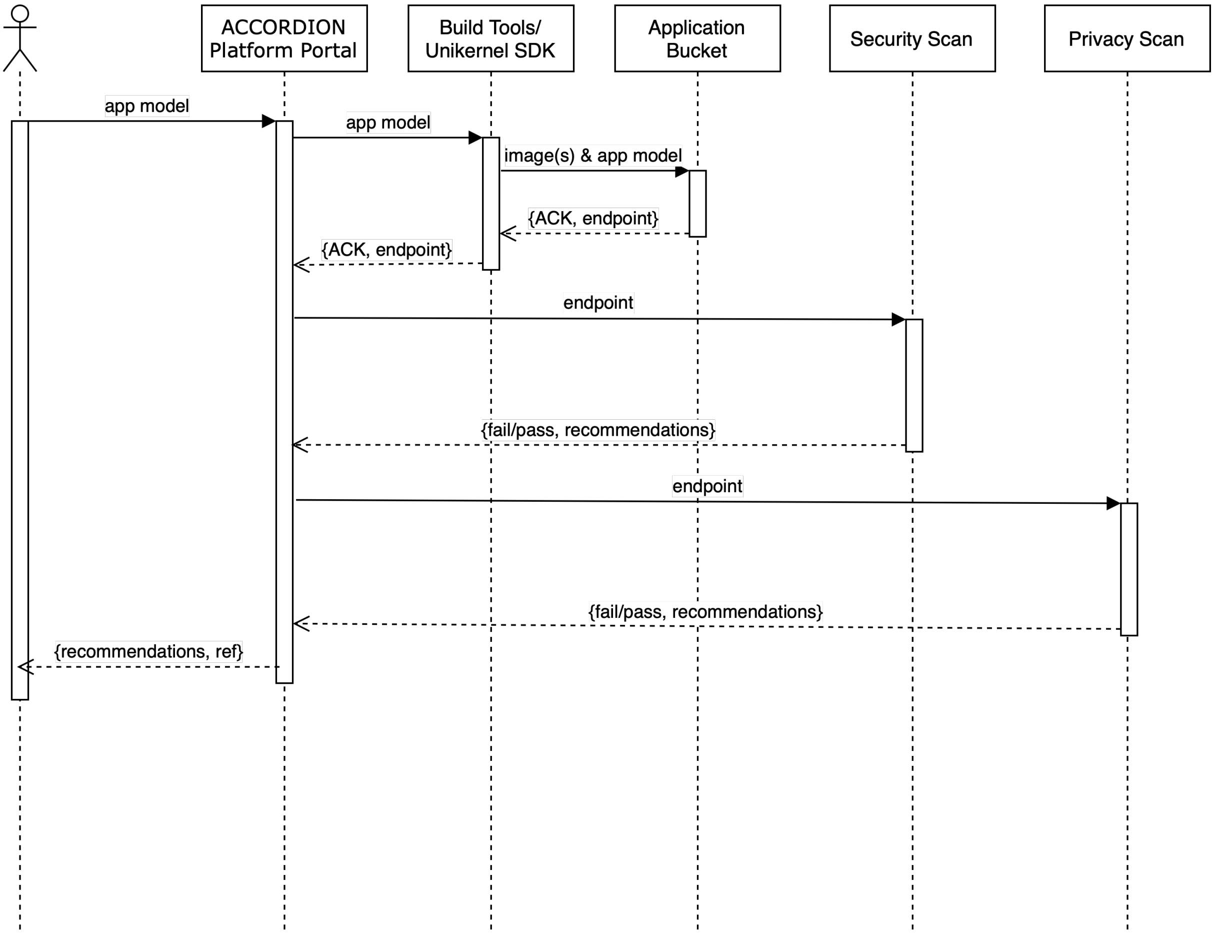


Figure 12: Flow of logic for Build and Test

6.3 Start application

The following sequence diagram (Figure 13) shows how the application components are being deployed. This operation requires more than just the application model; it also requires the QoE model metadata in order to estimate the achieved QoE levels based on the selection of resources. The figure shows how the Application Platform components are interacting with the Minicloud components in order to find, create and reserve the appropriate resources for the application components.

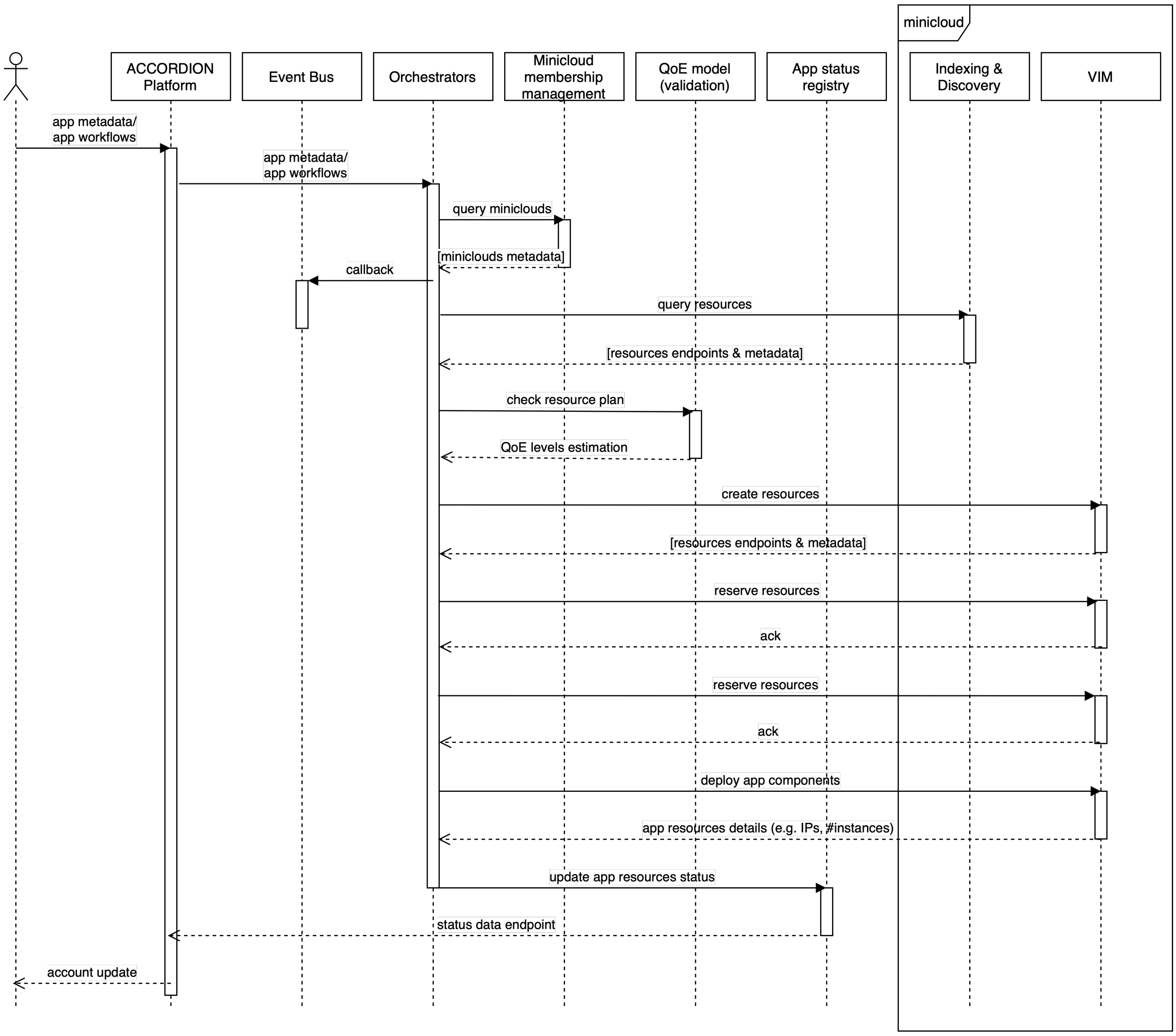


Figure 13: Flow of logic for starting the application

6.4 Runtime adaptation

The following sequence diagram (Figure 14) depicts the activation sequence of the components involved in the runtime adaptation of the platform to accommodate application-related workflows. The diagram highlights the use of the event bus to facilitate the communication between application and ACCORDION components and generally, to accommodate the automatic implementation of the various application workflows. Especially during the start of the application, a number of Minicloud components are attaching to the resources that hosts the application component to be able to monitor its performance and resilience levels.

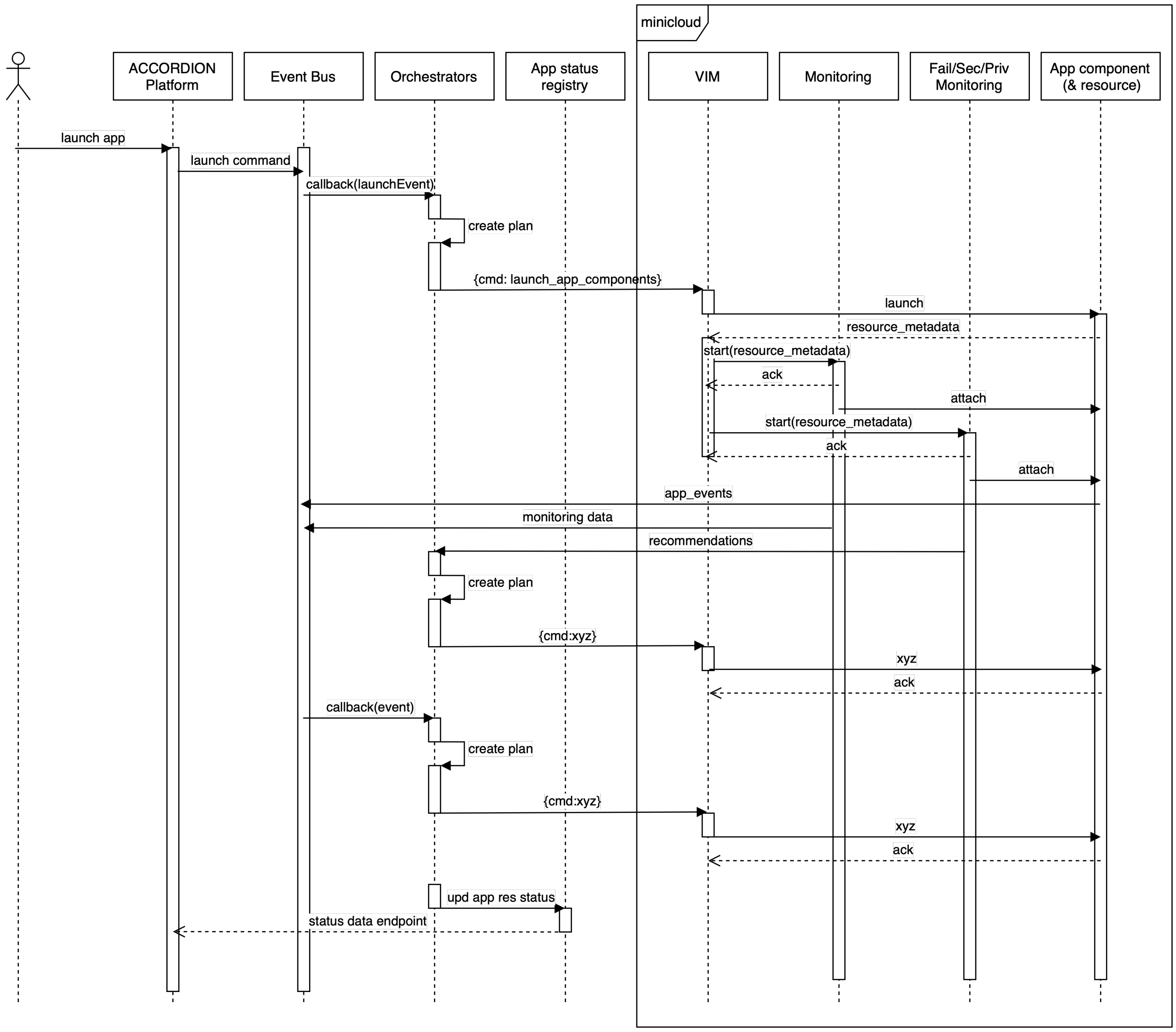


Figure 14: Flow of logic for application lifecycle management

7 Validating use case requirements

In what follows we provide the descriptions of the three validating use cases of ACCORDION. We validate the proposed architecture through the use case requirements provided in the first version of the series of reports on User requirements (D2.1). For each of those requirements we mark the components of our architecture that provide the functionality to support them. This exercise proves the coverage of the architecture in terms of supporting applications. If a requirement is marked with at least one 'x', this implies that the architecture can support it. Note that the descriptions of the components in Section 5 are already considering the use case requirements. For readability purposes the compliance table is provided in the Appendix: Requirements Compliance Table. We adopt the notation used in D2.1, with "FR" standing for "Functional Requirement", "NFR" for "non-Functional Requirement" and "UC#" stands for "Use Case #". A final remark is that there is a number of use case requirements that are excluded because they are application-oriented, i.e., they refer to the way the applications are implemented, and not to the hosting platform.

Finally, in order to ensure that this report is self-contained, we provide a brief description of the use cases. For more details the reader can refer to the ACCORDION report: *D2.1: User requirements (I)*.

7.1 Use case #1: Collaborative VR

Solutions are sought to enable immersive collaborative VR training tools and applications for a large number of remote users while overcoming the necessity to rely on heavy, local technology assets. To scale current solution designs, edge-cloud resources can be exploited for untethered mobile collaborative VR training experiences supporting advanced processing capabilities with low-latency and thus reduce constraints in resource limits, GPU, battery and mobility on untethered HMDs. Moreover, adopt a networking functionality beyond the standard client-server model and with decrease dependence on local service with one of the users to act as the server responsible for handling game logic and broadcasting messages to remote clients.

7.2 Use case #2: Multiplayer Mobile Gaming

ORBK Use Case is a multiplayer mobile game. Game servers will be deployed on top of the ACCORDION system so as to meet the requirements of NextGen mobile gaming, which aims to lower latency between servers and clients and highly improve user experience. It will also take advantage of AI-based network orchestration to dynamically and automatically deploy new servers based on performance metrics and player's geographical localization.

7.3 Use case #3: Content delivery for cloud gaming engines

PLEX approach towards ACCORDION raises the development of a brand-new complete platform, based on a PaaS model integrating edge computing and derived services.

A multi-level, microservices-based scenario platform that calculates varied information from different locations and creates real-time analytics. Analytics and derived knowledge elements that provide advantages to all parts of a new value chain based on information processing.

The continuous processing of information will be managed from different leveled nodes. The integration of all the information will lead to an expert system whose parts are mutually feeding each other. As a result, content delivery to different devices and the measurement of their response will be done in a consistent way across peripheral devices and connections.

8 Summary and future work

This document summarizes the efforts of the partners involved in T2.3 and WP2 in general to design the ACCORDION environment architecture. It is mainly composed of two large-scale systems: the infrastructure of federated edge and cloud nodes and the ACCORDION platform. The former is meant to solve issues of interoperability among heterogeneous edge/cloud nodes that are put together in a dynamic way forming a federation. The latter is meant to solve management issues, orchestrating resources and application components across miniclouds and providing an environment for the development and deployment of applications.

ACCORDION is emphasizing on the needs of the application providers/developers trying to accommodate their needs for smooth transition to the ACCORDION ecosystem from their traditional ones. This high-level requirement has a deep impact on the design of the system with a lot of effort being put in the specification of components and processes that will automate procedures and especially the application lifecycle management.

The result of the work is a layered architecture and the definition of the encompassed functional and non-functional components. These components were delineated in D2.2 and their functionality was – to an extent -defined in D2.1. This report (D2.3) solidifies the specification of the functionality of those components so as to allow for the release of a first set of components and a complete first version of the ACCORDION artifacts (infrastructure and platform). The document (D2.3) will be updated on M22 and M36 incorporating changes dictated by the evaluation of the pilots and the updating of the requirements. It will also be updated with a technical presentation of the API details that each of the components will implement.

Appendix: Requirements Compliance Table

ID	Description	Orchestrators	Event Bus	Application description	Platform Portal	Application status registry	Application Bucket	Build & Testing	QoE model	VIM	Monitoring (Perf/Sec/Priv/Fail)	Indexing and Discovery	Storage	Membership management	ACCORDION Services
F_UC1_01	APPLICATION DEVELOPER: Use the mirror networking service or similar for matchmaking, creation of session and selection of an already existing session (ip, location, userid master)		x							x	x				x
F_UC1_02	USER: Able to create session and/or select an existing session from the application on the HMD based on my credentials		x			x									
F_UC1_03	APPLICATION DEVELOPER: Session management through a relay server or message broker in the cloud	x		x		x									x
F_UC1_04	USER: Application updates through cloud-based repository, including apk installation on HMD					x	x								
F_UC1_05	APPLICATION DEVELOPER: Communication of the application to Azure Cloud to store and retrieve user analytics		x	x											x
F_UC1_06	USER: Able to visualize my performance analytics on the HMD	x									x				
F_UC1_07	APPLICATION DEVELOPER: The application component running on the HMD should be aware of the connected resources where part of the application has been offloaded.		x			x				x		x			
F_UC1_08	APPLICATION DEVELOPER: The application running on the HMD should be able to connect via standardised protocols to the resources where part of the application has been offloaded.		x			x				x		x			
F_UC1_09	APPLICATION DEVELOPER: Able to send output from the controllers and HMD to the resource where part of the application has been offloaded		x			x				x		x			
F_UC1_10	APPLICATION DEVELOPER: Able to receive encoded image data to the HMD for display from the resource where part of the application has been offloaded		x			x				x		x			
F_UC1_11	APPLICATION DEVELOPER: Support continuous streaming of two images (one per eye) per user from the edge resource node to the HMD.		x						x	x	x			x	x

F_UC1_12	APPLICATION DEVELOPER: Application component on the untethered HMD needs to be deployed and run on ARM based architecture (to support SoC)	x		x			x			x					
F_UC1_13	APPLICATION DEVELOPER: The resource discovery mechanism of ACCORDION should offload part of the application functionality from the HMD to nearby edge resource considering lowest average latency	x							x	x	x	x		x	x
F_UC1_14	APPLICATION DEVELOPER: Efficient sharing of the same resource among different users	x				x			x	x	x			x	
F_UC1_15	APPLICATION DEVELOPER: The send rate of rotations/translations from the HMD should be dynamically set considering the network characteristics		x	x					x		x				x
F_UC1_17	APPLICATION DEVELOPER: Communication of the HMD to the Remote Service (RS) in the cloud when launching the app on the HMD		x	x		x									x
F_UC1_18	APPLICATION DEVELOPER: Establish communication and initial data transfer from edge node for the user to enter the virtual OR		x											x	x
F_UC1_19	APPLICATION DEVELOPER: VR scene, data assets and avatars are locally stored on the edge node	x		x						x		x	x		
F_UC1_20	APPLICATION DEVELOPER: Different instances of the application can run on the same edge node supporting different users via their HMD devices	x		x			x							x	
F_UC1_21	APPLICATION DEVELOPER: The HMD should be able to receive data from different modules running on different edge nodes (e.g. the part of the application, the QoE model)		x			x			x	x					x
NF_UC1_01	USER, APPLICATION DEVELOPER: Round trip time (RTT) latency <15ms	x							x	x	x				x
NF_UC1_02	USER, APPLICATION DEVELOPER: Data rate >50 Mbps supported by at least 5Ghz wifi or 5G			x					x		x	x			x
NF_UC1_03	USER, APPLICATION DEVELOPER: Minimum connectivity requirements of 5GHz wifi or 5G			x					x		x				x
NF_UC1_04	USER, APPLICATION DEVELOPER: Connectivity from user HMD device <10 ms			x					x		x				x
NF_UC1_07	ADMINISTRATOR: Support at least 15 CCUs in the same OR			x					x						x
NF_UC1_09	ADMINISTRATOR: The ACCORDION framework to ensure security of the software components on the end devices and computing resources and transferred data across the network		x	x			x				x				x
NF_UC1_10	ADMINISTRATOR, APPLICATION DEVELOPER: To be able to use existing networks and available infrastructures	x		x						x				x	x
NF_UC1_11	APPLICATION DEVELOPER: Receive error messages on potential problems with existing resources, continue the VR app by on a newly discovered communicating to another resource (discovery and placement)		x						x		x				x
NF_UC1_13	ADMINISTRATOR: Maintain application integrity and user’s security			x		x									

NF_UC1_14	APPLICATION DEVELOPER, USER: Proximity of the relay server based on the users' footprints	x	x						x		x	x			x
NF_UC1_15	APPLICATION DEVELOPER: GPU and CUDA acceleration capabilities available at edge nodes, where part of the application is instantiated.			x			x	x	x			x			
NF_UC1_19	USER, APPLICATION DEVELOPER: Performed actions from all users should be synchronized to the output rendered image to each individual user on his HMD with lowest average latency		x						x		x				x
F_UC2_01	INFRASTRUCTURE: As APPLICATION DEVELOPER I want the selected version of the Lobby Server to be up and running all the time or whenever a Player starts the Game on her/his device	x		x		x				x	x			x	
F_UC2_02	INFRASTRUCTURE: As APPLICATION DEVELOPER I want the selected version of the Game Server to be up and running whenever a Player starts the Game on her/his device	x		x		x				x	x			x	
F_UC2_03	INFRASTRUCTURE: As ADMINISTRATOR I want to have access to the web-based administration after authorization by entering user and password				x										
F_UC2_04	INFRASTRUCTURE: As ADMINISTRATOR I want to be able to see a list of all Administrator User Accounts				x										
F_UC2_05	INFRASTRUCTURE: As ADMINISTRATOR I want to be able to select any of the Administrator User Account in order to perform one of actions: remove, reset password, block / unblock				x										
F_UC2_06	INFRASTRUCTURE: As ADMINISTRATOR I want to be able to add new Administrator User Account by entering new user's email address and sending the invitation				x										
F_UC2_07	INFRASTRUCTURE: As ADMINISTRATOR I want to be able to use web-based administration panel in order to perform all necessary configuration operations				x										
F_UC2_08	INFRASTRUCTURE: As ADMINISTRATOR I want to be able to see a list of all uploaded versions of the Lobby Server (Docker image file) with upload date and time and version number				x		x								
F_UC2_09	INFRASTRUCTURE: As ADMINISTRATOR I want to be able to see a list of all uploaded versions of the Game Server (Docker image file) with upload date and time and version number				x		x								
F_UC2_10	INFRASTRUCTURE: As ADMINISTRATOR I want to be able to select any of the Lobby Server (Docker image file) displayed on the list of Lobby Servers in order to perform one of actions: replace, remove, edit information (version, requirements, description)				x										
F_UC2_11	INFRASTRUCTURE: As ADMINISTRATOR I want to be able to select any of the Game Server (Docker image file) displayed on the list of Game Servers in order to perform one of actions: replace, remove, edit information (version, requirements, description)				x										
F_UC2_12	INFRASTRUCTURE: As ADMINISTRATOR I want to be able to upload new Lobby Server (Docker image file) with automatic date and time information and manually entered version and description				x		x								
F_UC2_13	INFRASTRUCTURE: As ADMINISTRATOR I want to be able to upload new Game Server (Docker image file) with automatic date and time information and manually entered version, requirements and description				x		x								

F_UC2_14	INFRASTRUCTURE: As ADMINISTRATOR I want to be able to select what version of Lobby Server should currently be used by the system				x		x								
F_UC2_15	INFRASTRUCTURE: As ADMINISTRATOR I want to be able to select what version of Game Server should run in relation to the Game version reported by the Player's device during the Game launch process		x		x	x	x				x				
F_UC2_16	INFRASTRUCTURE: As ADMINISTRATOR I want to be able to select what version of Lobby Server should currently be used by the system for all defined regions (i.e. EU, US, AS)				x		x								
F_UC2_17	INFRASTRUCTURE: As ADMINISTRATOR I want to be able to select what version of Game Server should currently be used by the system for all defined Regions (i.e. EU, US, AS)				x		x								
F_UC2_18	INFRASTRUCTURE: As ADMINISTRATOR I want to be able to see a list of all defined Regions				x										
F_UC2_19	INFRASTRUCTURE: As ADMINISTRATOR I want to be able to select any of the Regions in order to perform one of actions: remove, edit				x										
F_UC2_20	INFRASTRUCTURE: As ADMINISTRATOR I want to be able to add new Region by entering new Region's name				x										
F_UC2_21	SERVICES: As APPLICATION DEVELOPER I want to have a service that deploys Lobby Server	x	x		x		x				x	x			
F_UC2_22	SERVICES: As APPLICATION DEVELOPER I want to have a service that deploys Game Server	x	x		x		x				x	x			
F_UC2_23	SERVICES: As APPLICATION DEVELOPER I want to have a service which returns Lobby Server's IP address based on my Player's IP addresses		x		x	x					x				x
F_UC2_24	SERVICES: As APPLICATION DEVELOPER I want to have a service which returns Game Server's IP address based on my Player's IP addresses		x		x	x					x				x
F_UC2_25	SERVICES: As APPLICATION DEVELOPER I want to have a service that returns a list of all currently deployed Lobby Servers along with all the details: versions, requirements and description		x		x	x					x				
F_UC2_26	SERVICES: As APPLICATION DEVELOPER I want to have a service that returns a list of all currently deployed Game Servers along with all the details: versions, requirements and description		x		x	x					x				
NF_UC2_01	INFRASTRUCTURE: As APPLICATION DEVELOPER I want all the versions of uploaded Lobby Servers (Docker image files) to be stored securely and to be well protected against unauthorized access				x		x	x							
NF_UC2_02	INFRASTRUCTURE: As APPLICATION DEVELOPER I want all the versions of uploaded Game Servers (Docker image files) to be stored securely and to be well protected against unauthorized access				x		x	x							
NF_UC2_03	INFRASTRUCTURE: As ADMINISTRATOR I want to the web-based administration the panel to be well protected against unauthorized access				x			x							

NF_UC2_04	INFRASTRUCTURE: As ADMINISTRATOR I want to the web-based administration the panel to enable comfortable work on PC and Mac computers but also on mobile devices				x			x							
NF_UC2_05	INFRASTRUCTURE: As ADMINISTRATOR I want to the web-based administration the panel to enable comfortable work on newest versions of popular web browsers: Chrome, Safari, Firefox including mobile versions				x			x							
NF_UC2_06	INFRASTRUCTURE: As ADMINISTRATOR I want to the web-based administration the panel to enable very fast upload times of the Docker image files				x			x							
NF_UC2_07	SERVICES: As APPLICATION DEVELOPER I want to the service that deploys Lobby Server to select optimal localization for the deployment of Lobby Server in terms of lowest average latency between Game Server and all connected Players	x	x	x							x	x			
NF_UC2_08	SERVICES: As APPLICATION DEVELOPER I want to the service that deploys Game Server to select optimal localization for the deployment of Game Server in terms of lowest average latency between Game Server and all connected Players	x	x	x							x	x			
NF_UC2_09	SERVICES: As APPLICATION DEVELOPER I want to the service that deploys Game Server to select optimal resource for the deployment of Game Server in terms of requirements attached to the given Game Server version	x	x						x		x				
NF_UC2_10	GAME: As an APPLICATION DEVELOPER I want to have as low as possible latency in communication between Player's device and Game Server (< 100 ms)	x	x						x		x	x			
F_UC3_11	DEVICE IDENTIFICATION, ACCORDEON AI: As APPLICATION DEVELOPER I want to Receive updated information in real time so that I can keep the service updated		x	x		x					x				
F_UC3_12	DEVICE IDENTIFICATION: As APPLICATION DEVELOPER I want to Get device data so that I can develop a platform				x	x	x				x	x			x
F_UC3_16	CLIENT IDENTIFICATION: As USER I want to Broadcasted information to be useful and personal related so that I stay better informed		X								X				
F_UC3_17	CLIENT IDENTIFICATION: As USER I want to Data processed does not threaten my privacy so that I feel I am in a legal protected environment										X				
F_UC3_19	CLIENT IDENTIFICATION: As ADMINISTRATOR I want to Develop services for authenticated clients so that I offer added value to the use of the service			X											
F_UC3_20	CLIENT IDENTIFICATION: As ADMINISTRATOR I want to Be able to consult generated data, both historical and real time so that I am informed of crowd characteristics				X										
F_UC3_21	CLIENT IDENTIFICATION: As ADMINISTRATOR I want to that I can define concrete areas and trigger events so that security managing and employees' control is included	X		X											
F_UC3_23	CLIENT IDENTIFICATION: As MARKETER I want to Collect as much information as possible so that I can make deep analysis												X		X

F_UC3_24	CLIENT IDENTIFICATION, ACCORDEON AI: As APPLICATION DEVELOPER I want to Interact autonomously with customers so that We can test and improve Ai, BI and Big Data tools												X		
F_UC3_26	CONTENT DECISION MAKING: As USER I want to Decision making rules to be fair and honest so that content is not biased and becomes interesting and relevant	X		X								X			
F_UC3_27	CONTENT DECISION MAKING: As USER I want to Customer response influences content decision making so that Service will be interactive	X	X						X			X			
F_UC3_28	CONTENT DECISION MAKING: As USER I want to Having absolute clarity on how the algorithm works so that Confidence in the process is high			X											
F_UC3_30	CONTENT DECISION MAKING: As USER I want to identify when content derived from a decision making so that I maintain my sovereignty as a consumer	X		X											
F_UC3_33	CONTENT DECISION MAKING: As ADMINISTRATOR I want to have a platform to manage the decision making so that I can make decisions easily				X										
F_UC3_34	CONTENT DECISION MAKING: As ADMINISTRATOR I want to Broadcast and performance they have their own statistics crowd related so that information analysis is complete		X												
F_UC3_35	CONTENT DECISION MAKING: As ADMINISTRATOR I want to Platform itself had processed tested and defined about content decision making so that I can manage it easily				X										
F_UC3_37	CONTENT DECISION MAKING: As MARKETER I want to interact with a system of rules to define behaviors so that I'll use the platform intensively for my profession				X										
F_UC3_42	CONTENT DECISION MAKING: As MARKETER I want to Accomplish all the safety and privacy standards so that administrative offenses are avoided							X			X				
F_UC3_43	CONTENT DECISION MAKING: As MARKETER I want to Import/export info in compatible format so that it could be handled in other areas														X
F_UC3_45	CONTENT DECISION MAKING, ACCORDEON AI: As APPLICATION DEVELOPER I want to Use all processed data in machine learning systems so that new patterns and solutions are found			X											
F_UC3_46	CONTENT DECISION MAKING, ACCORDEON AI: As APPLICATION DEVELOPER I want to Check to have adequate content or requests to cloud so that Optimal content is available	X	X	X					X						
F_UC3_48	CONTENT DECISION MAKING: As APPLICATION DEVELOPER I want to Define simple and friendly tools to incorporate content so that content and rules can be added by inexperienced staff			X											
F_UC3_49	ADAPTED CONTENT: As USER I want to not publicly reveals my habits so that so that no compromising information is shown										X				
F_UC3_53	ADAPTED CONTENT: As ADMINISTRATOR I want to Content can be rated by customer so that I receive direct feedback, in addition to the derivative of behavioral analysis								X						

F_UC3_54	ADAPTED CONTENT: As MARKETER I want to Coordinate content with other edges via cloud so that advantage of global trends can be taken	X	X	X						X					
F_UC3_55	ADAPTED CONTENT: As MARKETER I want to Receive statistics on the relationship between adaptation and customer behavior so that report quality improves								X						
F_UC3_56	ADAPTED CONTENT, ACCORDEON AI: As APPLICATION DEVELOPER I want to Receive inputs about the relationship between adaptation and customer behavior so that report quality improves								X						
F_UC3_58	CONTENT STREAMING, ACCORDEON AI: As APPLICATION DEVELOPER I want to Keep transparent for other actors interactions with the cloud so that they are allowed to focus on other areas	X	X	X		X		X							
F_UC3_60	CONTENT STREAMING: As APPLICATION DEVELOPER I want to that Edges and Cloud are able to interchange info without human actions and solving possible problems so that Service integrity is kept	X	X	X				X							
F_UC3_61	ADVANCED ANALYTICS: As USER I want to Interact by asking questions or making proposals so that receiving data relates to my interest		X						X						
F_UC3_65	ADVANCED ANALYTICS: As ADMINISTRATOR I want to Summarized and visual access to result of advanced analytics so that as much information as possible is concentrated in the simplest visual model				X										
F_UC3_66	ADVANCED ANALYTICS: As ADMINISTRATOR I want to Access in general terms to autonomous or semi-autonomous data processing so that I can understand data processing				X										
F_UC3_71	AUGMENTED REALITY GAME: As USER I want to play anonymously so that data privacy is protected								X		X				
F_UC3_77	AUGMENTED REALITY GAME: As ADMINISTRATOR I want to Relate it to AR games from other malls so that the potential between multiple locations can be shared	X	X	X		X				X		X			
F_UC3_78	AUGMENTED REALITY GAME: As MARKETER I want to Use some api to add content so that I can participate in design without modeling knowledge				X										
F_UC3_81	MOBILE APP: As USER I want to Increase data accuracy so that accordion decisions are improved								X						
F_UC3_82	MOBILE APP: As USER I want to Facilitate personal data control so that personal data management is made easy										X				
F_UC3_83	MOBILE APP: As USER I want to Exploit device capabilities so that a whole new personal communication channel is created		X												X
F_UC3_84	MOBILE APP: As USER I want to Receive push notification updates so that there is a better communication stream		X												X

F_UC3_85	MOBILE APP: As USER I want to Use a local chat geolocated or walkie-talkie like so that there is local centered communication														X
F_UC3_88	MOBILE APP: As ADMINISTRATOR I want to that the app provides data even far from mall so that the whole system receives more data	X		X		X						X			
F_UC3_89	MOBILE APP: As ADMINISTRATOR I want to Add personal devices as accordion edges devices so that edge number increase based on my audience	X	X	X					X	X		X			
F_UC3_91	MOBILE APP, ACCORDEON AI: As APPLICATION DEVELOPER I want to Find more ways for interact so that AI improves			X	X										
NF_UC3_02	INFRASTRUCTURE & NETWORK: As USER I want to have access throughout the mall so that there is continuous interactivity, no areas limited	X		X					X			X			
NF_UC3_03	INFRASTRUCTURE & NETWORK: As ADMINISTRATOR I want to Use existing networks and infrastructures without the need for new network installation, except Accordion infrastructure so that I can deploy fast with minimal added costs						X			X					
NF_UC3_04	INFRASTRUCTURE & NETWORK: As ADMINISTRATOR I want to Accordion prevent falls infrastructure and network falls so that accordion services work autonomously, even without network	X				X					X	X			
NF_UC3_05	INFRASTRUCTURE & NETWORK: As ADMINISTRATOR I want to Accordion prevent and security breaches so that these can be a complement for other security tools										X				
NF_UC3_07	INFRASTRUCTURE & NETWORK, ACCORDEON AI: As APPLICATION DEVELOPER I want to Collect updated info about infrastructure and network status so that I can identify and prevent possible infrastructure derivate failures				X						X				
NF_UC3_08	INFRASTRUCTURE & NETWORK: As APPLICATION DEVELOPER I want to work in agnostic to infrastructure and network paradigm so that the platform runs in any network or infrastructure	X		X						X					
NF_UC3_09	TECHNOLOGY: As USER I want to Assure that accordion does not break my device so that Full confidence to the system										X				
NF_UC3_10	TECHNOLOGY: As USER I want to Accordion do not involve expenses or updates so that I keep my trust										X				
NF_UC3_12	TECHNOLOGY: As USER I'm familiar with accordion tech environment and I don’t need training to use it so that Accordion use is instantly			X	X										
NF_UC3_13	TECHNOLOGY: As ADMINISTRATOR I want to that everything is updated, open architecture, free and open code so that there are only platform costs, without dependencies			X			X								
NF_UC3_17	TECHNOLOGY: As APPLICATION DEVELOPER I want to Develop over a universal and portable development platform so that team integration is eased		X	X	X			X							
R_TS_01	APPLICATION DEVELOPER: Minimum changes in the application components when migrating them to ACCORDION			X			X	X							

R_TS_02	APPLICATION PROVIDER: Automated application lifecycle management	X				X					X				
R_TS_03	APPLICATION PROVIDER: Definition of accepted QoE levels								X		X				
R_TS_04	APPLICATION PROVIDER: Proactive fault tolerance	X	X			X					X	X			
R_TS_05	APPLICATION PROVIDER:Minimization of security and privacy vulnerabilities							X			X				
R_TS_06	APPLICATION DEVELOPER: Simplified definition of application lifecycle flows and blueprint generation			X	X										
R_TS_07	APPLICATION DEVELOPER: Provision of automations for lifecycle management	X				X									
R_TS_09	INFRASTRUCTURE OWNER: Minimum changes required to join ACCORDION federation of edge infrastructures.						X	X		X					
R_TS_10	APPLICATION PROVIDER: Application components being provided as Docker Images						X	X							
R_TS_11	APPLICATION PROVIDER: File storage functionality at the edge									X			X		
R_TS_12	APPLICATION PROVIDER: Docker images starting at edge nodes on demand	X	X	X			X	X		X	X	X			
R_TS_13	APPLICATION PROVIDER: Validation pipeline ensuring application and deployment process correctness						X	X							
R_TS_14	APPLICATION PROVIDER: Notification system in case of deployment failures or errors		X								X				
R_TS_15	ADMINISTRATOR: A system showing particular deployment steps that failed				X										
R_TS_16	APPLICATION PROVIDER: Support for heterogeneous hardware at the edge									X		X			
R_TS_17	APPLICATION PROVIDER: Ability to execute applications on geographically sparse edge resources											X			
R_TS_18	APPLICATION PROVIDER: Ability to migrate applications between edge resources						X			X					
R_TS_19	Definition of a common application model for accordion			X											
R_TS_20	INFRASTRUCTURE OWNER: Blueprints for VR/AR slice generation, and rapid slice creation and activation														X
R_TS_21	INFRASTRUCTURE OWNER: Efficiently spotting problems and accordingly plan operations to fix the issues			X	X										