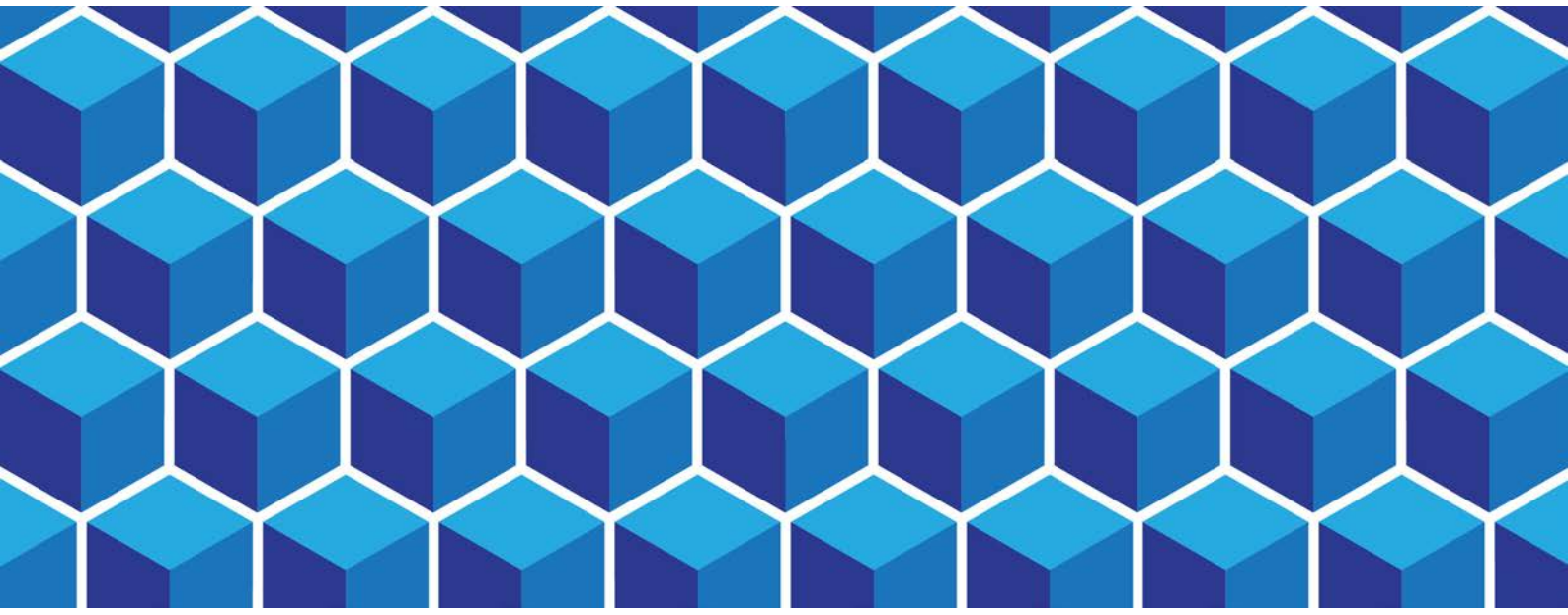
ARTICONF

D7.1: ARTICONF Testing Procedure, Infrastructure, Software Management, Deployment and Validation Plan

30/09/2019



Deliverable Responsible: Carlos Rubia, AGI

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825134, the ARTICONF Project.

D7.1: ARTICONF Testing Procedures, Infrastructure, Software Management, Deployment and Validation Plan

smART socIal media eCOsytstem in a blockchaiN Federated environment

Due date	30 September 2019
Work package	WP7
Lead partner	AGI
Dissemination level	PU – Public
Type	R – Document, report (excluding the periodic and final reports)
Authors	Carlos Rubia, AGI Cristian Martin, AGI Nishant Saurabh, UNI-KLU Long Cui, BY Aleksander Karadimce, UIST Spiros Koulouzis, UvA Alexandre Ulisses, MOG David Sarlos, VG
Reviewers	Nishant, Saurabh UNI-KLU Spiros, Koulouzis UvA Pedro Jacobetty, UEDIN Radu Prodan, UNI-KLU

Document History

Version	Date	Description
v0.1	01 June 2019	Table of contents distributed to the partners
v0.2	02 July 2019	First version distributed to the partners
v0.3	16 July 2019	Integration of ARTICONF tools and use case descriptions
v0.4	30 July 2019	Refined and homogenised sections with procedures timelines
v0.5	13 August 2019	Timeline clarification and new annexes
v0.6	14 August 2019	Version ready for contents review
v0.7	11 September 2019	Review comments addressed and ready for quality review
V0.8	23 September 2019	Major revision from Quality Manager
V0.9	24 September 2019	Major revision from Project Coordinator

Citation

Carlos Rubia, Cristian Martín, et al. (2019). ARTICONF testing procedure, infrastructure, software management, deployment and validation plan. ARTICONF Consortium, http://articonf.eu.

Disclaimer

Keyword list

Social Media | Software-Defined Networking | Cloud | Blockchain | Trust | Quality of Service | Quality of Experience

Table of Contents

Executive Summary

D7.1 is the first deliverable of the workpackage WP7: Integration, testing and use-cases of the ARTICONF project, responsible for integration, deployment, testing and evaluation of the tools developed in the project along with the use cases. Specifically, D7.1 relates to task T7.1: ARTICONF infrastructure, testbed deployment and management. This task aims to establish an experimental environment and procedures for all members of the consortium to deploy, run and test the ARTICONF tools. Moreover, T7.1 explores solutions to integrate these tools with use cases, allowing the consortium members to evaluate the objectives of the project.

This deliverable makes an initial effort to gather procedures for the software management, deployment and testing of tools (i.e. TIC, CONF, SMART, TAC) described in deliverable D2.2: ARTICONF Architecture and Interfaces Definition. The consortium will present a more descriptive testing, deployment and validation planning measures and methodologies in the revised version D7.3 due at month M18, encompassing any changes to the ARTICONF architecture and interface requirements.

Figures

Tables

1 Introduction

This deliverable explains the infrastructure that the ARTICONF consortium will use as a testbed for its ecosystem. More precisely, the document comprises of essential details including resource availability and allocation for the deployment of the tools and pilot use cases developed during the project lifecycle. We describe the ARTICONF testbed with respect to the heterogeneity and the diverse characteristic features associated to each of them. The current testbed details outlined in the deliverable reflects the basic testbed structure in accordance to the requirements of Milestones MS1 and MS2 of the project. We will describe the final version of the testbed in the subsequent updated version of this deliverable (D7.3 due at month M18), which will integrate future changes in the ARTICONF architecture, interface design and use cases requirements.

Additionally, we present the ARTICONF software management and testing procedures. We also outline the procedure for using a central and secure software development repository for the code of different entities, which allows continuous integration of functionalities. In general, we follow the software development and integration practices mentioned in the deliverable D1.2: Software Quality Assurance, where code inspection is the first step to ensure code quality, followed by the unit testing and functional testing of the different tools developed during the project. We explain the testing procedure as a part of the process to manage the project testbed and appoint a testing team comprising several testing automation experts from each consortium partner. Every expert holds the responsibility to supervise the testing, deployment and integration methodology for the developed set of tools and use cases.

We follow the standard procedure to ensure coherence and continuous integration of the ARTICONF features for the deployment of the developed software tools onto the testbed, by establishing a concurrent interaction between the unit testing procedures and the software deployment. This deliverable describes the initial testing and integration procedure, along with the associated tool description and deployment details. However, the testing and deployment procedure of the use cases differ depending on the specific policies and regulations of the companies. To evade any discrepancy, testing and deployment of every use case requires using the company's specific procedural guidelines and exploiting the ARTICONF testbed described in this deliverable to validate the Key Performance Indicators (KPIs) of the project. We also outline the associated validation procedures with their respective timelines, while porting the use cases to the testbed and the specific process for each use case evaluation will follow in future deliverables.

This deliverable has nine sections:

- Section 2 describes the architecture of the tools and the use cases;
- Section 3 explains the software development management;
- Section 4 explains the deployment procedures for the tools;

- Section 5 describes the testing procedures, the validation plan and defines the Quality Assurance (QA) Team to execute and control them;
- Section 6 depicts the tools needed for the testing process, software management and deployment of services and features;
- Section 7 reports the conclusion gathered in this deliverable;
- Section 8 lists the references found in the deliverable;
- Section 9 introduces the annexes used in the project.

2 Testbed Architectural Overview

ARTICONF is composed of several tools utilised by its use cases to accomplish their objectives. To enable members to prototype and test their solutions and to allow use cases evaluation, the consortium provides a testbed for the ARTICONF ecosystem and associated software tools deployment and testing management. This testbed is composed of specific resources provided by members of the consortium and it encompasses private resources and public resources from commercial cloud providers.

2.1 ARTICONF Testbed Specification

ARTICONF members provide several servers and machines to create an initial testbed, as summarised in Table 1. ARTICONF members will use this testbed to deploy, test, improve and add functionalities to the platform by following a continuous integration approach. This testbed also enables the deployment and testing of the use cases, including their integration with the ARTICONF ecosystem and associated tools for subsequent evaluation of the project objectives.

Table 1: Testbed infrastructure.

Member	CPU	RAM	Hard Disk	SSD	Operating System	Provider
UNI-KLU	2x Intel Xeon Gold 6148	12x 32GB		3.5TB	Ubuntu 16.04	UNI-KLU (Austria)
UVA	7xvCPU	4x 4GB 1x 8GB	4x 20GB 1x 220GB		Debian 9	UvA (Netherlands)
	(UvA ExoGENI) 9x Intel Xeon E5-2670	64 GB	9x 300GB 1x 7TB		Debian 9	UvA (Netherlands)
	(Typical ExoGENI) 10x Intel X5650	1x 12GB 10x 4GB	6x 1TB 10x 146GB			
UIST	4x 3.3GHz	8GB	1TB		Windows 7 Pro	UIST (North Macedonia)
BY	Intel Xeon	32GB	640GB		Ubuntu 18.04.2 x64	DigitalOcean[1]
	Intel Xeon	32GB	640GB		Ubuntu 18.10 x64	DigitalOcean
	Intel Xeon	8GB	160GB		Ubuntu 16.04.6 x64	DigitalOcean
	Intel Xeon	16GB	320GB		Ubuntu 16.04.4 x64	DigitalOcean
	Intel Xeon	1 GB	25GB		Ubuntu 16.04.4 x64	DigitalOcean
	Intel Xeon	16 GB	100GB		Ubuntu 16.04.4 x64	DigitalOcean
	Intel Xeon	32 GB	640GB		Ubuntu 18.04.2 x64	DigitalOcean
MOG	8x 2.4GHz	16GB	1TB	512GB	Ubuntu 18	MOG (Portugal)
AGI	2x 2.5GHz	8 GB	–	100GB	Debian 9	AWS Ireland
	i7-8700	8 GB	1TB	240GB	Ubuntu 18.04	AGI (Spain)
VG	2x 2.5GHz	8 GB	–	100GB	Debian 9	AWS Ireland

[1] The location of DigitalOcean VMs is in Frankfurt (Germany), London (UK) and Amsterdam (Netherlands).

ExoGENI, provided by UvA, is a multi-domain cloud structure based on an extended infrastructure-as-a-service cloud model with coordinated provisioning across multiple sites. Currently, ExoGENI consists of 21 sites, each of them maintaining its own rack. Other sites also maintain a similar rack; however, their hardware specification is constantly changing. Table 1 also provides a typical hardware specification for an ExoGENI rack, while Table 2 provides a list of currently available sites, including an estimate of the available Virtual Machines (VMs). For the ARTICONF project, the testbed composition, code deployment and data management will only use the resources provisioned in Amsterdam (Netherlands). Table 3 shows the types of VMs ExoGENI may provision.

Table 2: ExoGENi sites and VM resources.

Exogeni Rack Name	VM Resources	Bare Resources	Location
ExoGENI WVN	52		USA, WVN XO
ExoGENI WSU	50		USA, WSU XO
ExoGENI UvA NL	15		Netherlands, UVA XO
ExoGENI UNF	51		USA, UNF
ExoGENI UMass			USA,UMASS XO
ExoGENI UH	50		USA, UH XO
ExoGENI UFL	44		USA, UFL XO
ExoGENI UAF	51		USA, UAF XO
ExoGENI TAMU	46		USA, TAMU XO
ExoGENI SL	51		USA, SL XO
ExoGENI RCI	63		
ExoGENI PSC	52		USA, PSC XO
ExoGENI OSF	52		USA, OSF XA
ExoGENI NICTA			Australia, NICTA
ExoGENI LAT	51		USA, LAT
ExoGENI FIU			USA, FIU XO
ExoGENI ExoSM	565	15	
ExoGENI Duke			USA, Duke
ExoGENI CIENA HQ	25		Canada, Ciena XO
ExoGENI CIENA	25		USA, Ciena XO
ExoGENI BBN	33		USA, BBN XO

This testbed hosts the ARTICONF diverse toolset (i.e. SMART, TIC, CONF, TAC), the use cases and software services needed for the management of the ecosystem. Moreover, it hosts the platform's integration environment, including the ARTICONF's testing platform for the use cases. The testbed allows unit tests execution with an efficient software management system and the continuous integration principles intact, as described in in the next sections. The testbed will also facilitate ARTICONF platform performance evaluation testing by internal and external teams.

Table 3: ExoGENi resource specifications.

Resource Type	Name	Cores	RAM	Disk
Bare-metal node	ExoGENI-M4	16	48GB	146GB / 600GB
VM	XOSmall	1	1GB	10GB
VM	XOMedium	1	3GB	25GB
VM	XOLarge	2	6GB	50GB
VM	XOXLarge	4	12GB	75GB

2.2 ARTICONF Testbed Infrastructure

The initial testbed infrastructure composed of these resources interconnects through the public Internet by means of application programming interfaces (APIs) developed during the project. In this section, we provide an overview of the ARTICONF tools (i.e. TIC, CONF, SMART, TAC) and their deployment scenarios, following the microservices architectural paradigm. At the present stage of the project, however, we only cover the microservices that we intend to develop in the first 18 months of the project associated to each specific tool of the ARTICONF ecosystem. The updated version of this deliverable (D7.3) due at month M18 will cover the entire set of microservices representing the final platform. This section also presents specific resource descriptions for the microservices deployment and the associated APIs to facilitate intra- and inter-communication between tools. Currently, most microservices within the different tools are accessible through a REST API.

In addition to the ARTICONF tools, some services will also use the testbed to provide a suitable environment to manage the software, the deployment of microservices and the continuous testing of the code. We will explain the software management functional scenarios in Section 6. We plan to deploy software management tools in Amazon Web Services (AWS) instances owned by AGI in Ireland (see row AGI in Table 1).

2.2.1 Trust and Integration Controller (TIC)

Figure 1 depicts the architectural workflow of TIC consisting of four microservices:

- Blockchain Consortium;
- Cloud-based Big Data Storage;
- Relationship System;
- Personal Certificate Authority (PCA).

Each social media application integrates a personal certificate authority (PCA) bundle, which enables local content encryption before further processing and preserves user privacy by design principles through public key sharing. Therefore, encryption and decryption take place on the user side only (i.e. end-to-end encryption) and the blockchain consortium connector only receives anonymous encrypted data. Upon a new transaction request, the consortium connector invokes the endorser peer in the hyperledger fabric blockchain that verifies the transaction based on provided certificate information, returning the approval

or rejection feedback. The returned data also contains the simulated result of the associated smart contract execution. If approved, the connector sends the authenticated transaction to the ordering peer that, in turn, delivers the transaction information to all anchor peers, allowing them to update their ledger. By design, the blockchain transaction only keeps the fingerprint of the actual data, while a cloud-based big data storage stores all encrypted data, indexed by the fingerprint for fast queries during the retrieval operation.

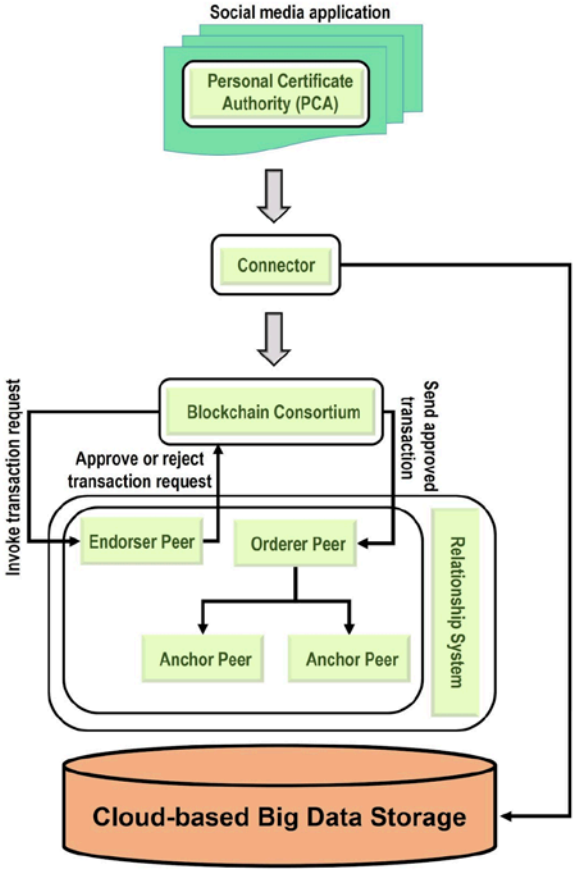


Figure 1: TIC architectural workflow.

Blockchain Consortium is the core part of TIC that offers a consistent and traceable mechanism to maintain trust. One can integrate TIC with third party authentication providers as a verification layer, alongside different consensus algorithms. TIC consumes requests dispatched from the Connectors that act as entry points and gateways. The Blockchain Consortium is a decoupled bundle, which one can flexibly integrate and teste with other bundles.

Cloud-based Big Data Storage is a distributed storage system that collects large shared data. As the blockchain consortium keeps track of the fingerprints of the cloud stored data, the cloud-based Big Data Storage offers a fast and persistent query service based on the returned indexing result from blockchain.

Relationship System is a Turing-complete component of the blockchain that handles logic processing. As different configured permissions map to different relationships, the relationship system verifies the

relationship logic when generating new transactions and meeting a byzantine fault tolerance consensus according to a verification algorithm.

Personal Certificate Authority (PCA) is a client-side software development kit that issues a certificate for each end-user based on the configuration from the central certificate authority, which controls the permission and authorization. The PCA uses its public key to convert user sensitive data into anonymous secured data that persists in the blockchain and cloud-based Big Data Storage. Since the private key always resides locally, the chances of decrypting anonymous data are minimal.

TIC deployment testbed will host the microservices on a Docker swarm cluster across the host machines available on the DigitalOcean cloud provided by BY (see Table 1), using five VMs with one core, 2GB of RAM, and 25GB of SSD. This cluster is available as a sample environment for the first 18 months. We will do a new estimation for every use case provider to define the final testbed of the project. At a later stage, each use case provider cluster will connect to the TIC cluster to form the Blockchain Consortium.

2.2.1 Co-located and Orchestrated Network Fabric (CONF)

CONF provides adaptive infrastructure provisioning for social media applications over an orchestrated network. It seamlessly integrates with the cloud edge infrastructure, able to intelligently provision services based on abstract application service requirements, operational conditions at the infrastructure level, and time-critical event triggering. The distribution of the networked infrastructure provisioned by CONF receives information from the intelligent community analytics of SMART and TAC services and supports them for a smooth and optimised operation. CONF will adopt a microservice architecture composed of several microservices, described in the following.

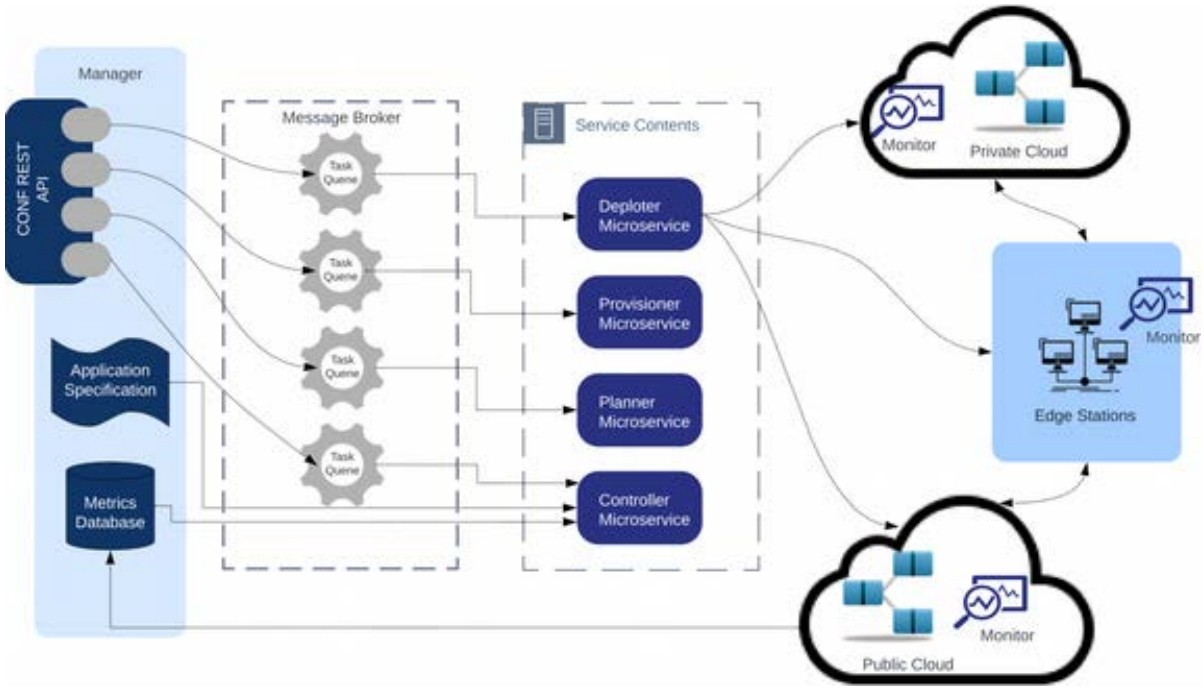


Figure 2: CONF architectural workflow.

Orchestration Manager provides a REST web service that allows CONF functions invoked by external clients. The Orchestration Manager forwards each request to the appropriate component and is responsible for coordinating all the individual components.

Message Broker implements message validation, transformation, routing, and facilitates communications between the orchestration manager and the different microservices. It can help compose asynchronous, loosely coupled applications by providing transparent communications to independent microservices.

Metrics Database enables application and infrastructure agents to store predefined metrics. We plan to use time series databases (e.g. Cassandra and InfluxDB) that can collect large amounts of data and seamlessly provide them to the monitoring services.

Application Specifications Repository is a service used by social media applications to store and modify their specification, including their Quality of Service (QoS) and Quality of Experience (QoE) attributes. Using this repository, the Planner checks if QoS/QoE attributes for a given scenario and if applications have potential bottlenecks.

Controller controls and changes the infrastructure solutions based on the QoS metrics of social network applications and their infrastructures throughout the entire process of planning, provisioning, and deployment to ensure system self-adaptability.

Planner encapsulates all infrastructure planning functions based on several state-of-the-art scheduling and planning algorithms. Its goal is to produce efficient infrastructure topologies based on application retirements and constraints and to select cost-effective VMs.

Provisioner automates the provisioning of infrastructure plans provided by the Planner onto underlying infrastructure services. More specifically, the provisioner decomposes the infrastructure description and provisions it across multiple clouds, edge or fog infrastructure with transparent network configuration.

Deployer deploys application microservices onto the provisioned cloud and edge infrastructures. The Deployer provides a scheduling mechanism based on network bottlenecks and maximises the satisfaction of deployment deadlines. It is also responsible for deploying a system for autonomous monitoring of the application and its underlying infrastructure.

Metrics Collector deployed on the provisioned infrastructure collects system and application-level metrics that assist in the adaptive control of the infrastructure and in maintaining QoS constrains.

CONF deployment testbed will host all these microservices in a Docker swarm cluster at UvA (aggregating 7 vCPU with 24GB of RAM and 300GB of disk), except for the Metrics Collector. For testing the planning, provisioning, deployment and monitoring, CONF will use the ExoGENI cloud that provides around 1200 VMs with exclusive use. This number, however, depends on each site’s availability and multi-user load.

2.2.2 Semantic Model with Self-Adaptive and Autonomous Relevant Technology (SMART)

SMART's objective is to improve trust and eliminate malicious actors in participatory exchanges and collaborative decision making. Therefore, it processes anonymised data from interactions saved in the blockchain to provide intelligence about consensus decisions, reduce costs and latency by analysing previous voting outcomes and preferences, and targeting appropriate users for the different use cases. Additionally, it will give insights about physical events to help resource provisioning in CONF. To fulfil this goal, SMART adopts a role-stage model integrating various facets of social media to define and classify users based on social information and content attributes. As the blockchain maintains user privacy by design, the SMART tool cannot identify individual users or weaken their privacy during this process. SMART will deploy several microservices during the first 18 months, described in the following.

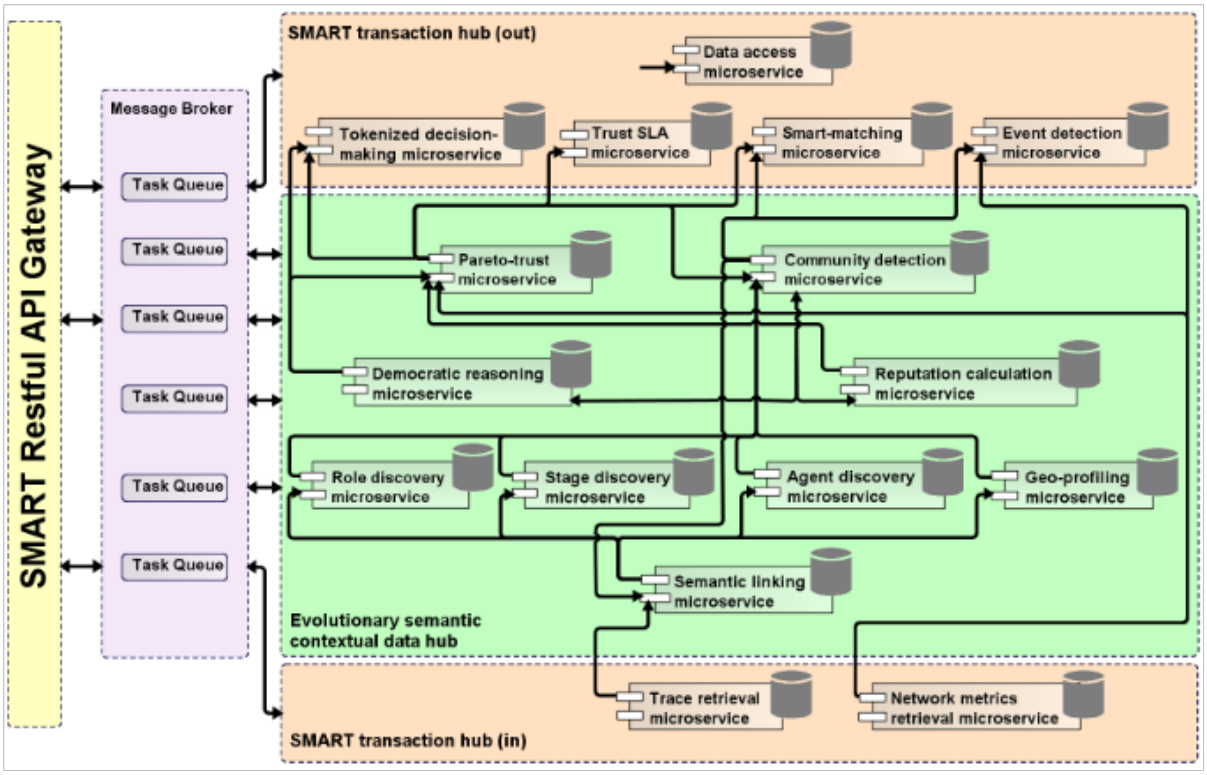


Figure 3: SMART architectural workflow.

Trace retrieval microservice provides an interface to the blockchain. The blockchain component pushes all openly available traces to SMART via this interface.

Semantic linking microservice helps to explicitly state implicit connections between users by analysing extensive and repeated interactions. The explicit information about links will provide the basis for all further reasoning, classification and intelligence.

Agent discovery microservice labels anonymous logins with individual agents, e.g. a human or an automated device.

Role discovery microservice labels anonymous users with individual roles which are important for classification (e.g. family role).

Geo-profiling microservice clusters users physically close to each other. Creating such physical communities will be one of the time-critical triggers for the event detection microservice.

Stage discovery microservice classifies the roles of individual users (e.g. family role has a stage father), again important for classification.

Democratic reasoning microservice serves as a central knowledge-based component providing all facts and rules for other microservices.

Reputation calculation microservice estimates users' trustworthiness considering various conflicting parameters, such as accuracy, timeliness, latency, and high anonymity preservation.

Event detection microservice publishes information about events to its subscribers. Upon detecting new communities (e.g. the geo-profiling algorithm assigns a group of users to a physical location), it broadcasts the physical group discovery event to the consuming services.

Data access microservice offers heuristics and data from SMART to enable other ARTICONF components to apply evaluation and cognition for different use cases.

RESTful gateway provides public access to all interfaces of the SMART tool's microservices required by other components.

Message broker enables loose coupling between all microservices as it abstracts communication.

SMART deployment testbed will host these loosely-coupled microservices with self-contained principles inside Docker containers within a Kubernetes cluster at UNI-KLU (see Table 1), which enables faster deployment, better versioning and clearer documentation. The message broker allows integration of any self-contained microservice and their interfaces without affecting other microservices functionality.

2.2.3 Tools for Analytics and Cognition (TAC)

Like the other tools, TAC also adopts a microservice architecture described in the following.

Guided Analytics Dashboard is responsible for aggregating and exploiting the content diffusion supply chain across different providers, communities, groups and users. It seeks to provide information that supports user engagement in collaborative economies with monetary inclusion using two microservices:

- **Guided analytics** provisions media mining, machine learning and statistics, and knowledge extraction from selected data. The aggregated summarised information flows to the dashboard and the relevant microservices.
- **Use-case rating** microservice provides results for the use case providers to be aware of the users' activity on the platform and helps them track the rating and functioning of their application.

Augmented Cognition Data Model provides analytics and extract meaningful insights about users' changing behaviour, cultural and social requirements. These tools collect a large amount of geospatial and temporal data about user activity within social media applications, as follows:

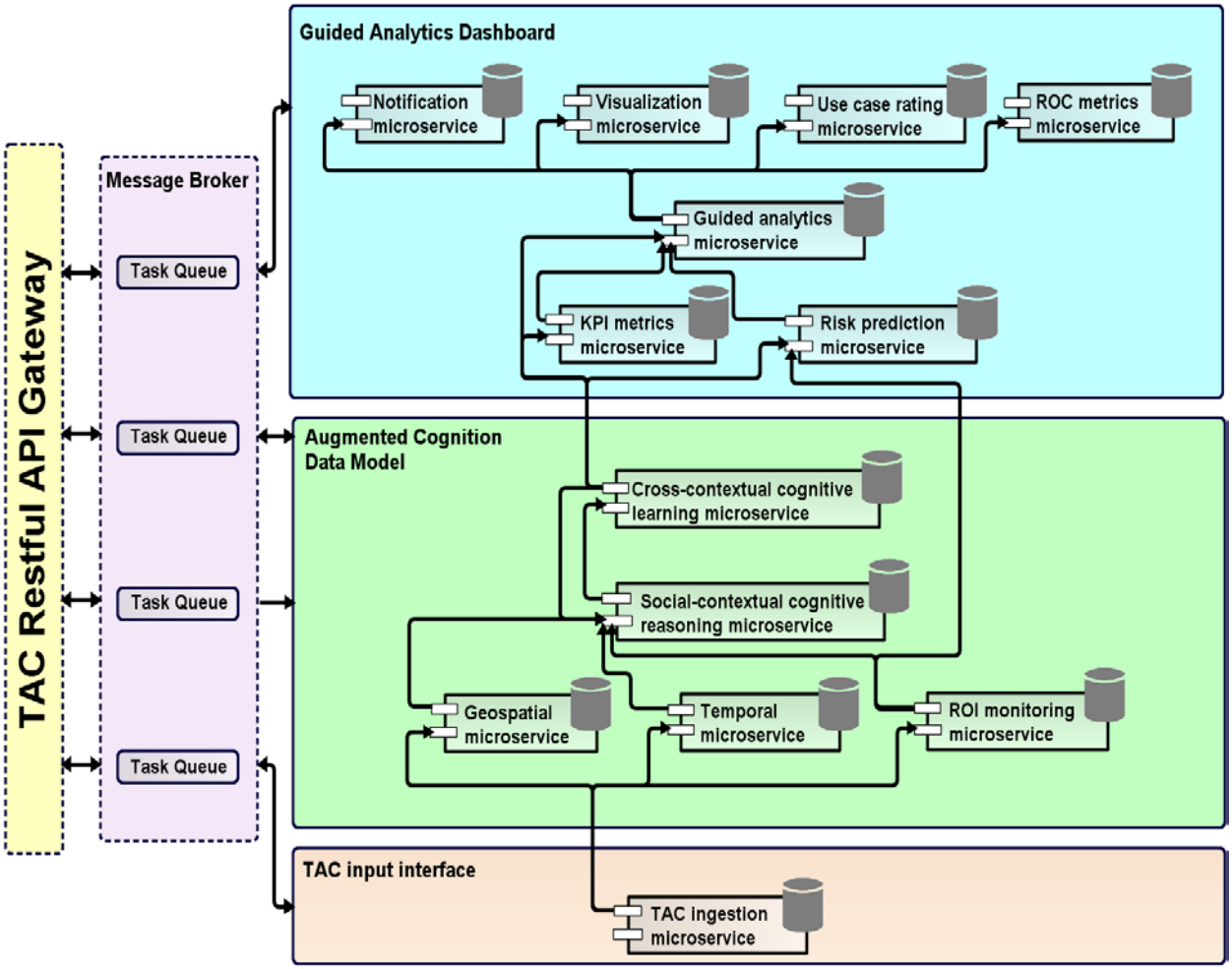


Figure 4: TAC architectural workflow.

- **Geospatial** microservice handles the gathering, display, and manipulation of Global Positioning System (GPS) data, satellite photography, geo-tagging and historical data based on geographic coordinates, or implicitly in terms of a street address, postal code, or forest stand identifier.
- **Temporal** microservice offers support for analysing complex social networks and for providing users with actionable insights, even for a large amount of data produced over a short time, coupled with visualisation to uncover influential actors, finding helpful bridging people and identifying destructive spammers.
- **Return on investment (ROI) monitoring** microservice helps to explicitly provide business insights and measurable Return on collaboration (ROC) metrics, quantified in terms of improvement relative to the capital invested in a given functional area. TAC also provides users with real-time cost per engagement information to analyse and control their ROC and the success of their specific use case.
- **Social-contextual cognitive reasoning** and **cross-contextual cognitive learning** microservices reduces the uncertainty by double-checking the validity of information and their sources in a hostile environment. These microservices will improve the collaboration amongst intelligently defined

communities elaborating over the shared knowledge acquisition and learning, reducing biases and increasing the benefits.

RESTful API gateway provides public access to all interfaces of the TAC tool's microservices required by other components.

Message Broker is an architectural pattern for message validation, transformation, and routing, which helps in composing asynchronous, loosely coupled applications by providing transparent communication to independent microservices. The Message Broker allows adding and changing single services and their interfaces without affecting other microservices.

TAC input interface integrates the **TAC ingestion** microservice that serves as an input interface for the TIC, SMART, and CONF tools, responsible for pushing all openly available and anonymised semantic data.

TAC deployment testbed scenario. We will deploy all microservices describe above on a VM container within a TAC server hosted at UIST (see Table 1), which enables faster deployment, better versioning and clearer documentation. The message broker allows integration of any self-contained microservice and their interfaces without affecting other microservices functionality. TAC and its microservices provide easy integration with the ARTICONF Guided Analytics Dashboard accessible on the Web and as a mobile app.

2.3 Use Cases Testbed Infrastructure

The testbed infrastructure described in Section 2.1 includes the use cases too, which require integration with the ARTICONF platform tools and their microservices. As the use cases are still in the design stage, this section presents services involved in every use case at a conceptual level. The final version of the use cases will describe them in a greater detail at a microservice level as part of the deliverable D7.2 (and its subsequent updated versions).

2.3.1 Crowd Journalism

The crowd journalism use case intends to stimulate cooperation between ordinary citizens and professional journalists. This use case allows the production of media content through a cooperative process that will support both ordinary citizens and journalists.

Citizens will use a mobile phone application which allows them capture and share live video feeds. In addition to the mobile application, this use case includes other components, such as the media engine, the central viewing platform, the storage and the marketplace, as defined in the high-level architecture as presented in the deliverable D2.1: ARTICONF Platform Requirements and Use Cases. The goal is to properly deploy all modules in a connected fashion, checking the functionalities and validating the different types of associated workflow tasks, namely videos production, viewing, purchase of video or other contents. ARTICONF tools provide several mechanisms that will aid the testing and help reducing the necessity of a full scenario deployment during the tests of this use case. Figure 5 shows the original conceptual use case architecture, before the deployment of ARTICONF on the testbed. MOG uses a server

located in its own facilities to deploy the crowd journalism use case services not managed by ARTICONF, as presented in Table 1.

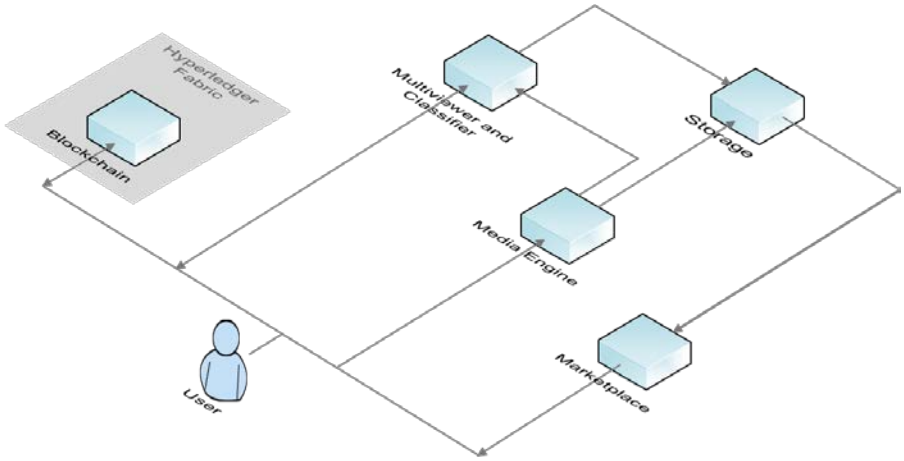


Figure 5: Conceptual crowd journalism use case architecture (before ARTICONF).

Figure 6 represents the crowd journalism use case deployment on the testbed integrated with the ARTICONF toolset. Upon ARTICONF testbed deployment, the use case uses the TIC tool to customise and manage the blockchain network. The SMART tool improves the rating system in the social network. The TAC tool receives data from SMART and provides insights about the transactions handled by TIC and the user interactions in the ecosystem. The CONF tool handles the server deployment to host the social network backend, media engine and the storage components. Moreover, it allows dynamic vertical and horizontal scaling (i.e. up or down) of the resources according to the requirements.

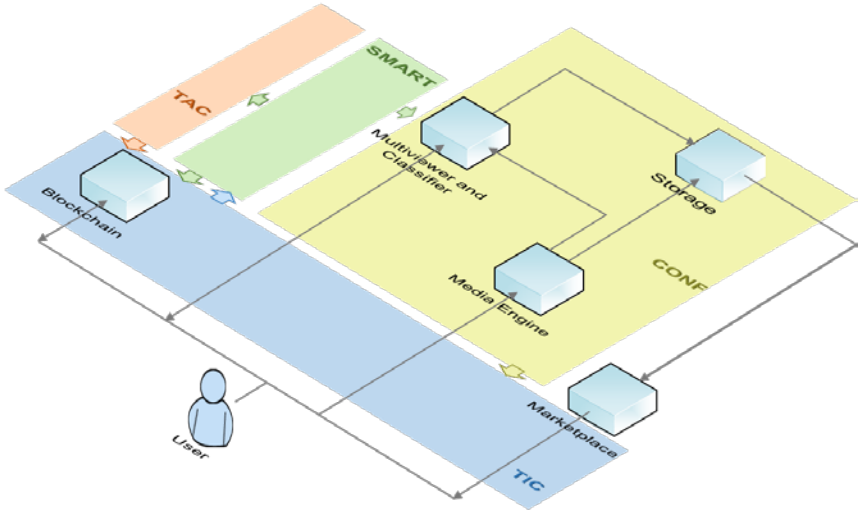


Figure 6: Conceptual crowd journalism use case architecture in ARTICONF.

2.3.2 Car Sharing

The car sharing use case has four different components: blockchain network, mobile application, social network and artificial intelligence (AI). As the ARTICONF tools handle several deployments in these

components, the testbed needed for this use case is minimal. Figure 7 shows the original use case architecture at a conceptual level, representing the use case scenario before its integration in ARTICONF. Prior to its integration in the ARTICONF testbed, the car sharing use case uses internal resources in AGI which are not part of ARTICONF testbed resources for development purpose. Figure 7 shows a conceptual version of car sharing use case. To make the development easier, we will deploy one VM or container in the AGI premises for each service. AGI provides a server for hosting the services needed by its car sharing social network and the database not managed by ARTICONF, as outlined in Table 1. Like other tools developed by ARTICONF project, AGI will provide a mobile testing environment for its car sharing use case based on a git repository, enabling a continuous integration and delivery of the services to mobile phones.

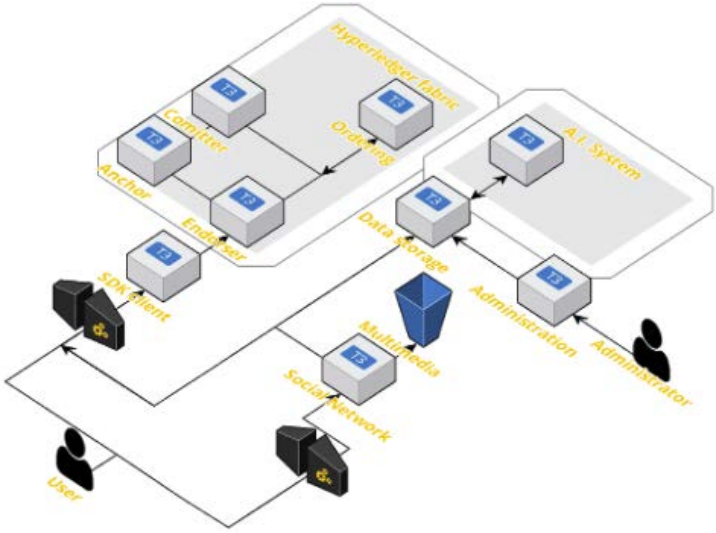


Figure 7: Conceptual car sharing use case architecture (before ARTICONF).

Figure 8 shows car sharing use case exploiting the ARTICONF stand-alone tools by means of well-defined APIs. The TIC tool handles the deployment of the blockchain network on the ARTICONF testbed, represented as a cloud in the figure. Consequently, the definition of TIC architecture and functionality is transparent and beyond the scope of car sharing use case testbed definition. Once the car sharing use case runs in the ARTICONF ecosystem using TIC, the SMART tool improves the rewarding system in its car sharing social network. Moreover, some resources dedicated to computationally-intensive AI components reside outside the ARTICONF testbed, exploited by AGI depending on the use case requirements. The CONF tool handles the deployment of the servers to host the social network backend and database and provides functionality for scaling up or down of servers, depending on the dynamics of the system.

Figure 9 shows the deployment scenario of the car sharing use case testbed and its integration with the ARTICONF testbed. We will discuss the communication interfaces and complete integration scenarios between use case and ARTICONF tools in the upcoming deliverables. The figure also shows the physical geographical deployment locations. The communication between the different tools and the use case itself takes place through RESTful APIs.

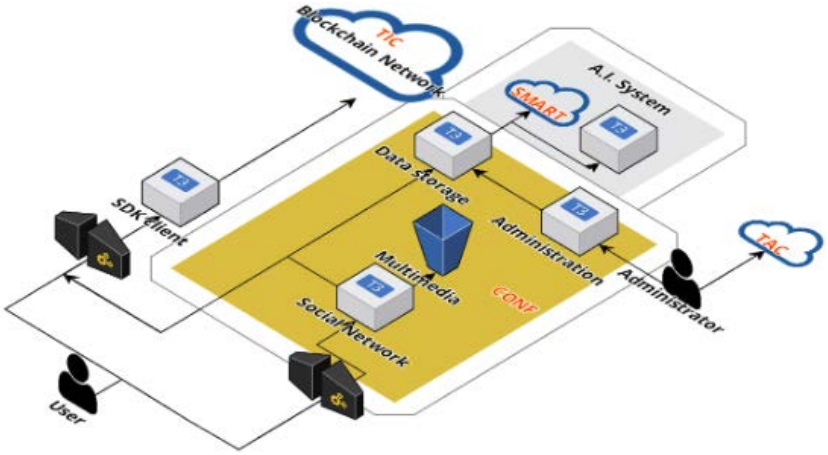


Figure 8: Conceptual car sharing use case architecture in ARTICONF.

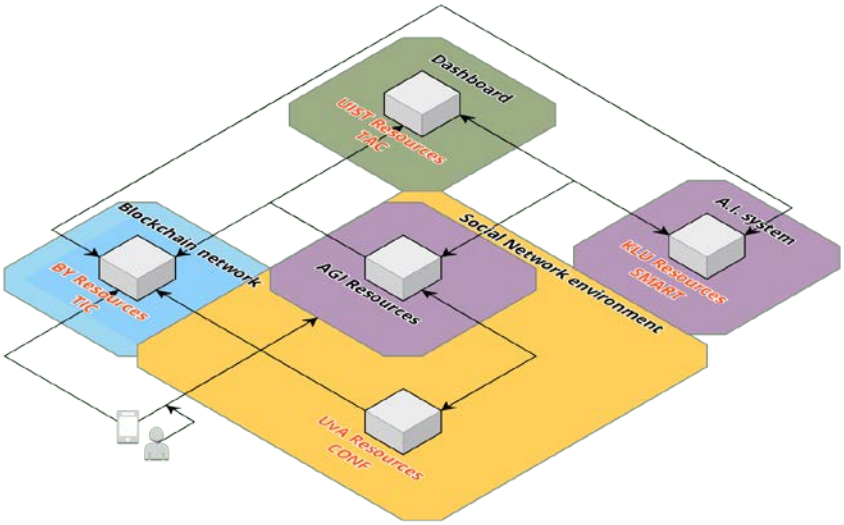


Figure 9: ARTICONF tools deployment and communication in the car sharing use case.

2.3.3 Crowd-Cooperative Video Creation

Before its deployment on ARTICONF, the Vialog crowd-cooperative video creation use case consisted of a centralised reputation system depicted in Figure 10, which neither makes use of Decentralised Identifiers (DIDs) that allows the users keep control of their own identity independently of any centralised registry, nor of a blockchain-based reward system. In general, the application uses a classic frontend/backend architecture (deployed on AWS), leveraging centralised user accounts with a reputation system akin to reviewer scores on mainstream platforms today.

After ARTICONF deployment, the TIC tool handles the deployment of the blockchain network to implement operations related to DIDs, including sign-in, transfer and acquisition of financial rewards (i.e. tokens stored on ARTICONF's underlying ledger), and storage/management of reputation points of the created videos (i.e. linked to their DIDs). In general, the Vialog backend increasingly relies on ARTICONF components as they are available, while deploying the front-end services within the relevant applications

leaving the DID management to the end users. Furthermore, VG takes the advantage of the CONF component for deployment and scaling of its backend code and database and uses the SMART tool to improve the reward system for video reviewers, as showed in Figure 11.

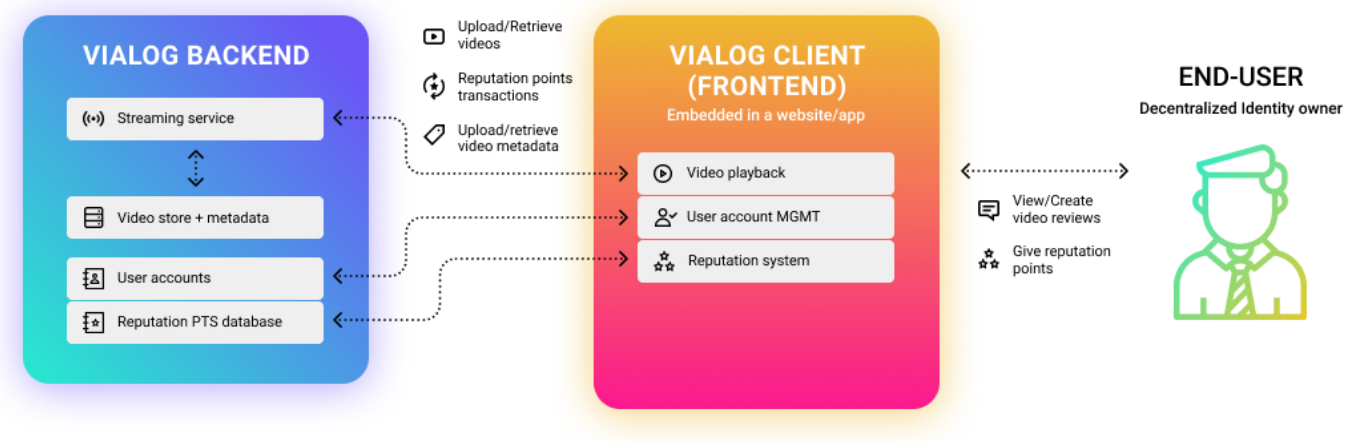


Figure 10: Conceptual crowd-cooperative use case architecture (before ARTICONF).

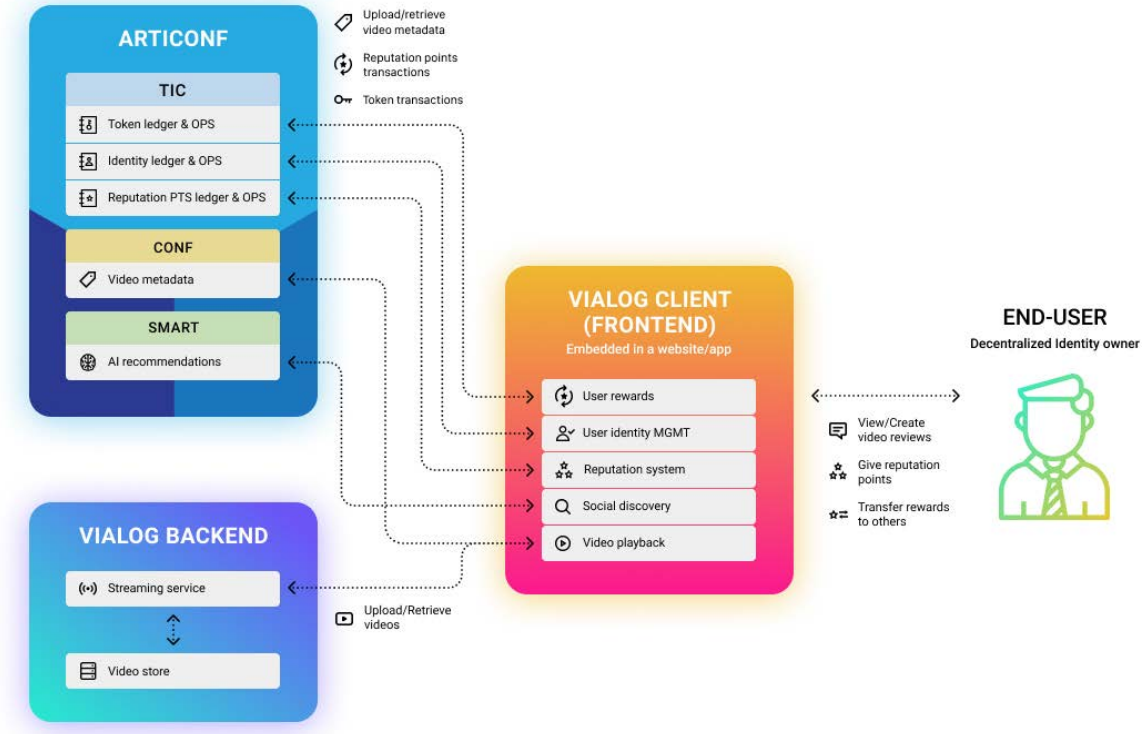


Figure 11: Conceptual crowd-cooperative use case architecture in ARTICONF.

The code development pertaining to use case is versioned and controlled using Git on Github or any other Git repository providers. Partners or other third parties use this facility to execute their own frontend (or an open source reference implementation) leveraging the Vialog video reviews and DIDs, reputation points and reward system, implemented via ARTICONF in any testing or real-world environment.

2.3.4 Smart Energy

The Smart energy use case uses the bitYoga blockchain smart energy platform depicted in Figure 12, which allows utility companies incentivise customers to log their consumption and production data. The use case offers a blockchain platform for recording user transactions concerning energy consumption, production and trading, which stores all the transaction data as references pointing to a big data cloud storage of locally encrypted user-related sensitive data. Utility companies benefit from an analytics system for performing granular predictions on in-home and community-level energy production and consumption. Additionally, a mobile application is available to users and utility companies. Customers use their mobile application to register themselves with the bitYoga platform and store their energy production and consumption data in the locally encrypted platform (i.e. user DB). They also choose to share their data with utility companies (i.e. Grid Sys) by providing them access to their data, rewarded in form of tokens. They can trade their excess energy with their neighbourhood and the grid repaid in tokens and compensate their energy bills using the tokens generated from sharing their data and trading energy. Utility companies that participate in the sharing economy by rewarding users for sharing their data can then use those data for predictive analytics, planning and trading.

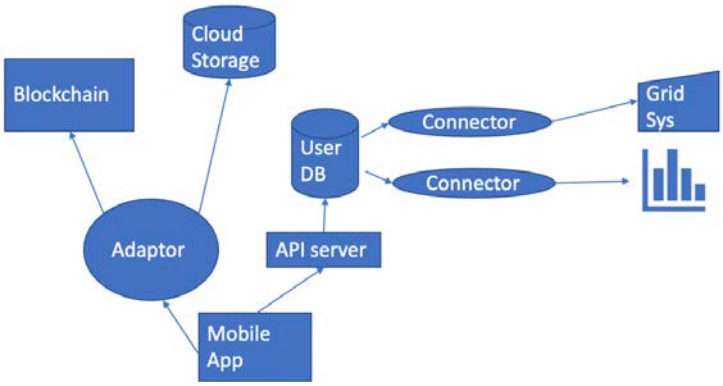


Figure 12: Conceptual smart energy use case architecture (before ARTICONF).

With ARTICONF in place, the smart energy use case utilises the TIC tool as the blockchain and cloud storage service underneath, SMART and TAC as intelligent analysis and evaluation tools, and CONF as the resource auto-provisioning tool. As Figure 13 illustrates, most of the use case components integrate with the ARTICONF tools and testbed, except for the mobile app and the user sensitive database.

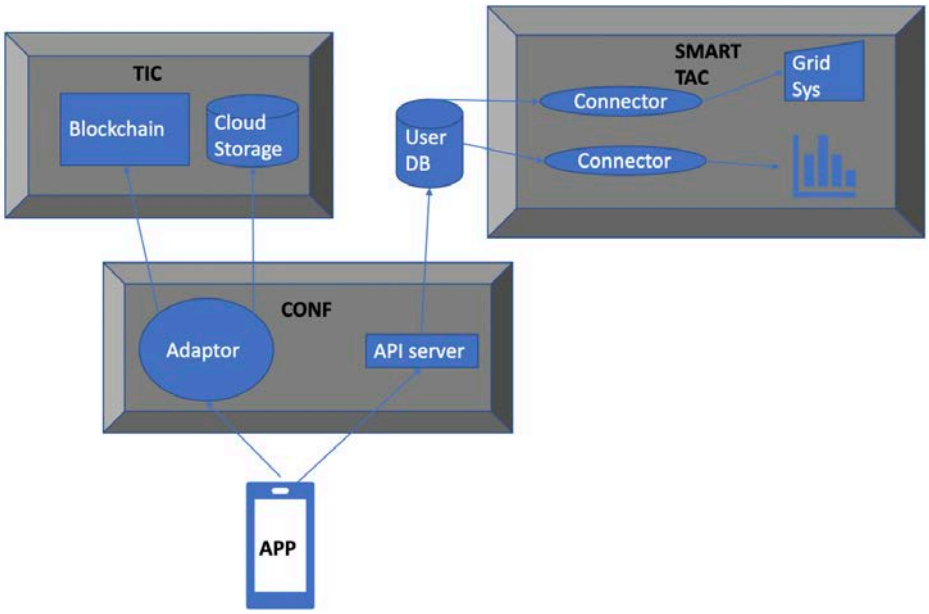


Figure 13: Conceptual emart energy use case architecture in ARTICONF.

3 Software Management

Software management is a foundation to create a robust and stable project. Big projects, such as ARTICONF, involve a high number of partners and developers who need protocols and tools to work together. Nowadays, Git repositories are one of the best tools to create and to collaborate on the same source code within a project. A Git repository allows easy sharing of code and working with new features without interruptions thanks to its branches approach that allows creating new feature or updates in a separated branch. Git branches are easy to merge while defining an understandable workflow. The decentralised branches also provide an isolated environment for every change in the code on the developer side. New features contained in new branches allow a continuous delivery, while the main branch contains a stable code version ready for deployment in the staging environment. In addition, commits in the Git repository show the complete changes in the code and allow to return to a previous version, if necessary. A Git repository working with a code inspector and a proper work protocol is a good recipe to create high quality code. A code inspector tool is also important in the development stage, prohibiting multiple developers working on the same project introduce unexpected bugs that become important security risks. A good code inspector also informs about suspicious code and about test coverage. Moreover, some code inspectors like SonarQube inform the developers about the time required to fix suspicious code or bugs, further providing an improved schedule planning.

ARTICONF uses two main services for the software management plan: Gitlab as the remote Git code repository and SonarQube as code inspector tool (see Figure 14). Once a partner pushes the code to the development branch, the continuous integration tool calls the SonarQube service that analyses it and provides a dashboard with feedback about duplicate code, bugs and other information. SonarQube also sets a quality gate representing a minimum standard to pass the analysis. In case of major bugs, the code analysis fails, and the developers need to fix it before merging it into the staging branch, which launches the deployment on the testbed. The SonarQube code inspector service runs on the same server as the continuous integration tool and the GitLab services. This service only works in the development environment and provides quality analysis of the code. The programmers analyse results using the dashboard and utilise it to improve the quality of the code, reduce duplicate code, eliminate bugs, and enhance test coverage. To work with GitLab, we will use the Gitflow workflow that allows the developers to program every feature in a new branch, merged back into the main development branch upon its completion. This way, the development environment is always ready and contains the latest features for deployment. Once the merging completes, the software deployment tool launches the SonarQube analysis that shows the results on its own dashboard.

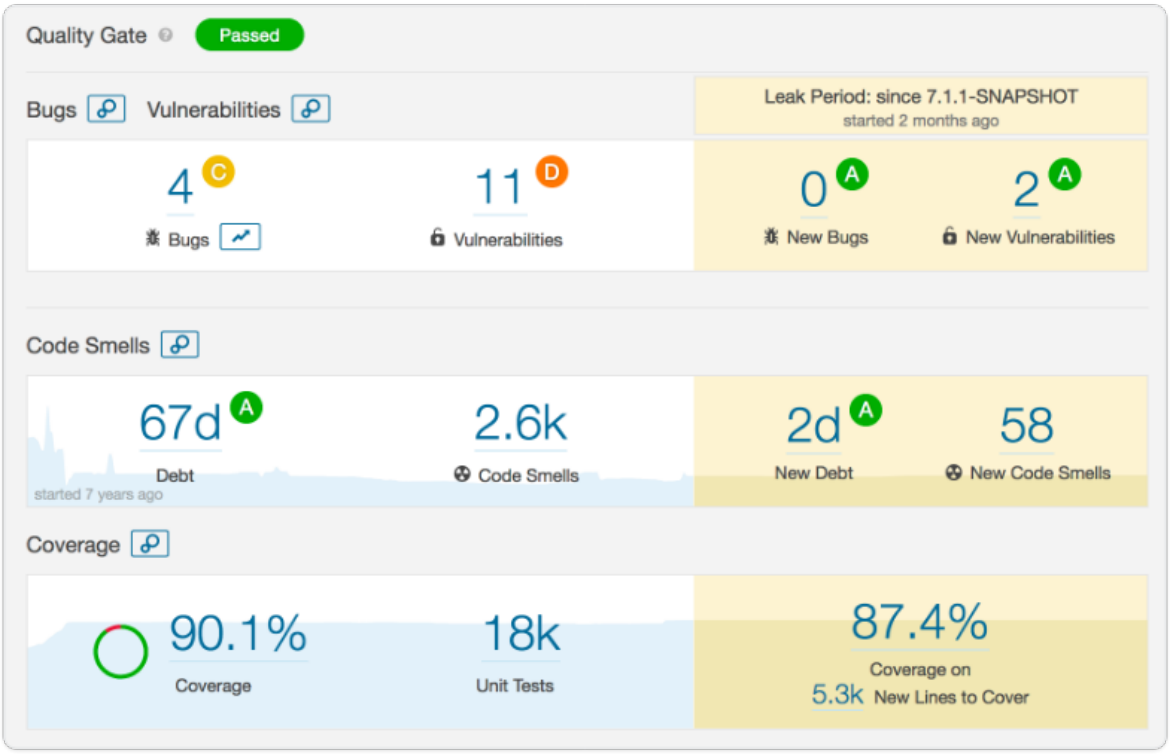


Figure 14: SonarQube dashboard example.

4 Deployment Procedure

A software deployment procedure encompasses all processes required to prepare a software to run and become available for usage. The deployment of the software uses the testbed resources described Section 2.1. This testbed represents a staging environment for the deployment where the consortium tests the code and evaluate the tools during the project. To achieve this, we designed an automatic deployment system based on the Jenkins service that facilitates the continuous integration process as a standard practice for the project development, which allows the developers integrate new code in the GitLab repository, while making them aware of integration errors during code merging and automated execution of tests. This approach enables developers prevent integration problems and allows them to add new features, while easily detecting and locating errors. Moreover, Jenkins allows the deployment of the entire code developed by different partners in the testbed after completing a new functionality and passing the unit tests. To provide a stable deployment process, each partner is responsible for provisioning its own resources in a stable environment and enabling the technical staff automatically deploy the code every time a new functionality is operational. After the provisioning process, Jenkins configured with the access profile will deploy the code whenever a merge request comes. The service updates take place for each technical partner whenever necessary, while providing information about scheduled downtimes to the rest of the partners.

The established deployment procedure is as follows (see Figure 15):

- Whenever a developer pushes a new code to the development branch, Jenkins callas the code inspector service to analyse it and provide quality metrics. The service integrated with the deployment process is SonarQube that provides feedback to developers for maintaining a high code quality without blocking the deployment process.
- Whenever a developer deploys a new code in the testbed, he/she requests for a merge in the staging branch. If AGI, as responsible entity for the software management, accepts this merge, an attempt of the new code deployment will take place over the testbed. This merging triggers the unit tests integrated with Jenkins in the deployment process. These tests can block the deployment if detecting errors or inconsistencies in the code. If the deployment process stops because of unit tests, an email informs the responsible person for the faulty code and the error received.
- Once the unit tests pass, the system deploys the code in the testbed or the staging environment available for testing and evaluation. If resources are not available or Jenkins cannot access them, the code becomes unavailable to the users, while informing the responsible person for the error received.
- If the automatic deployment correctly completes, the Jenkins service deploys the new developed code in the testbed and makes it available in the staging environment.

This integration process will deploy the software tools developed by the ARTICONF project: TIC, SMART, CONF and TAC. The deployment of the use cases will use a combination of resources available in the ARTICONF testbed and on partners' own premises. For the latter case, the companies will apply

methodologies and processes specific to their use cases development code that does not conflict with their security and resource accesses policies from external services. Therefore, even though the use cases will finally use the same deployment testbed, the industrial partners will manage them according to their own internal policies for deployment.

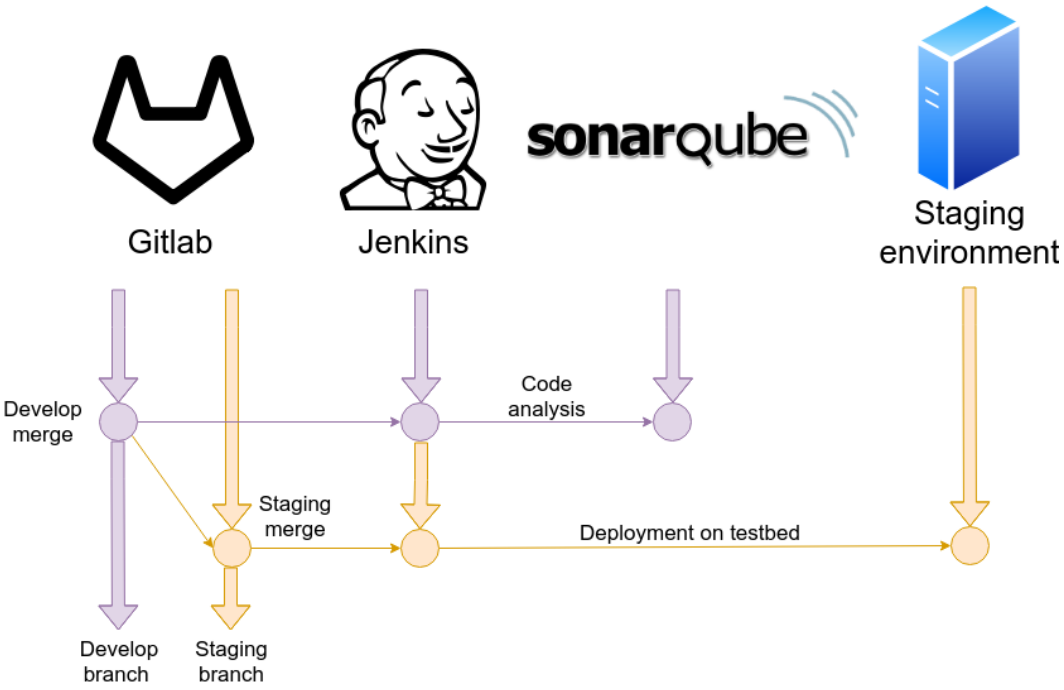


Figure 15: Continuous integration and deployment workflow.

5 Testing and Validation Procedures

Testing is the process of verifying the code developed in the project at several levels, avoiding errors, vulnerabilities and inconsistencies in the applications or tools. This process takes place automatically or manually, but always pursues the objective of guaranteeing the quality of the software. The evaluation of the platform is the process of testing the ARTICONF platform through its use cases and evaluating its KPIs through specific functional and non-functional testing of use cases. Since these use cases are still within the design phase, we will present the final plan for these tests in the upcoming deliverables within WP7.

5.1 Test Plan

The procedure to test the tools developed by the ARTICONF project (i.e. TIC, SMART, CONF, TAC) consists of several processes that aims to test the code from different viewpoints. The procedure follows the standard IEEE 29119 adapted to the project for simplifying the process of testing the software.

5.1.1 Unit and Integration Testing

A unit is the smallest part of any software, usually with a few inputs and one output. The deployment system managed by the Jenkins service automatically carries out these tests designed by code unit developers. Individual unit tests take place in isolation from other units and aim to verify the correct implementation of the detailed design. The unit tests have many benefits:

1. It discovers defects early so that they do not propagate into subsequent phases where detection becomes more difficult;
2. It simplifies the integration and testing and produces that are easier to maintain;
3. It ensures that individual modules or units correctly operate, independently of other system components. This is usually the task of each coding team responsible for testing its own units.

In this stage, the integration tests also verify the interfaces as a part of the unit testing. Testing other units linked with the tested ones is important for verifying the behaviour of more complex subsystems, as many errors arise in later stages because of problems with the interfaces between units. In this context, integration testing aims to verify the integration at a low level, while the entire system integration takes place in a second stage using functional tests.

The developers typically design and engineer the tests before implementing them. In this case, the QA Team performs and supervises the unit and integration tests, specifically appointed for this task (see Section 5.3). The report of these tests will follow the document presented in Annex I, which will serve as a guide for developers to test the integrity and integration of the code.

5.1.2 Functional Testing

The second stage performs functional tests to check complete functionalities and a higher level of interface testing using real data communication and separately covering each functionality. This stage

verifies that the functionalities of the tools work comply with original requirements. Testers or individuals not necessarily involved in the code development cycle carry out these tests and the QA Team specifically appointed for this task supervises them (see Section 5.3). Annex II shows the template for the functional testing report used by the QA Team to report results and descriptions of the process.

The multi-stage testing procedure described in this document cannot further proceed until having the result of functional tests, which holds a stable version of the code after its completion. The QA Team records and documents the testing results and their execution procedures. Upon detecting anomalies, the tests restart from the first stage (unit testing) after correcting the malfunction. Every new release repeats these functional tests with extended or new functionalities.

5.1.3 Acceptance Testing

Acceptance or compliance testing ensures that the functionalities developed in the ARTICONF project comply with the market viewpoint. External users, close to the ARTICONF target market, perform these tests in two steps:

1. automated testing based on testing scripts;
2. external user group testing defined at a later stage.

This external user group composed of beta testers relates to the ARTICONF market and uses the received feedback to modify or add new functionalities to the project. Furthermore, the same QA Team defined Section 5.3 will create a specific report to gather these feedbacks or market insights.

5.1.4 Testing Workflow and Schedule

The process of different testing stages takes place according to the the schema presented in Figure 16. In this workflow, the unit testing is a continuous process performed during the entire development cycle, while the other tests have specific time slots during the project. Figure 17 shows the timeline of the testing process aligned with the milestones of the project.

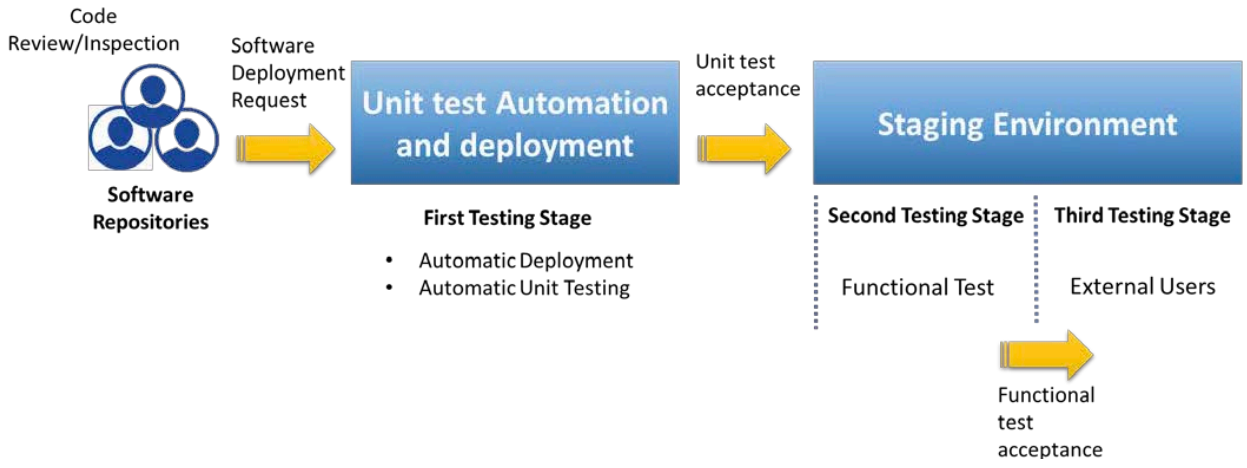


Figure 16: ARTICONF testing workflow.

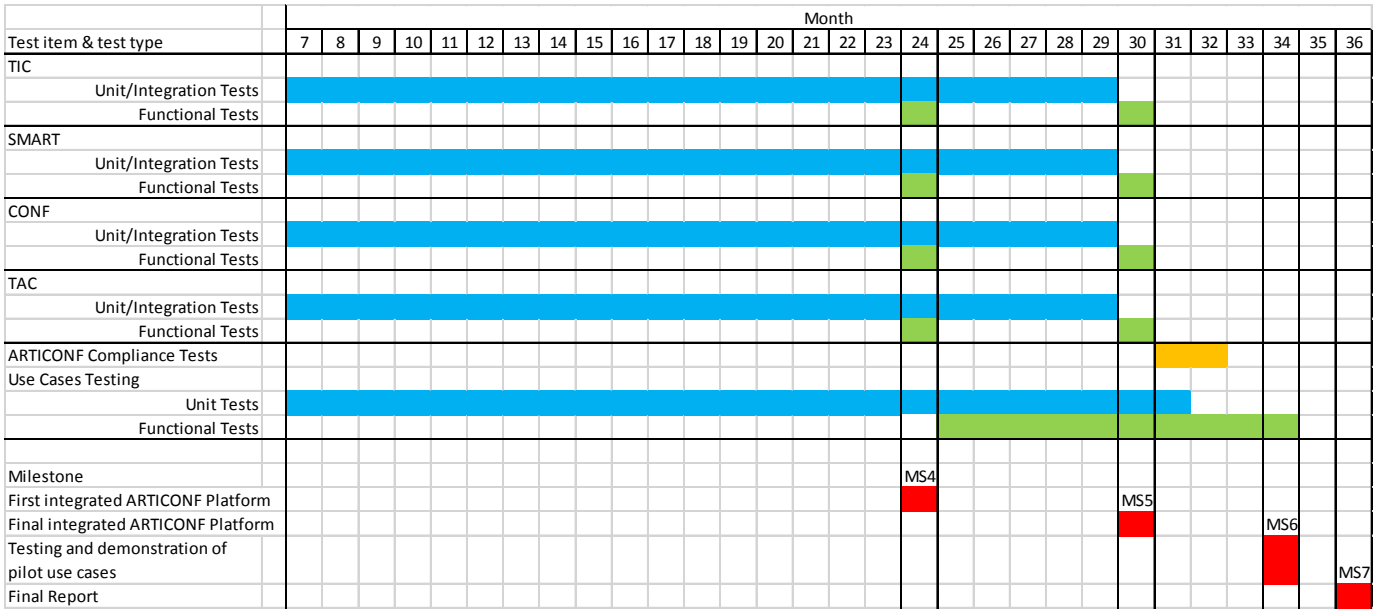


Figure 17: Timeline of the testing process in ARTICONF.

Regarding the use cases, the unit and functional testing follow the specific company policies, which are out of the scope of this procedure. This approach aims not to enter conflict with these testing processes and the policies regarding documentation and internal schedules. The functional testing, according to the functional description of use cases provided in the deliverable D2.1, aims to test the final integration of the ARTICONF tools in the testbed after releasing and deploying both prototypes of the ARTICONF platform at months M24 and M30 (as part of Task T7.4 of WP7).

5.2 Validation Plan

The validation plan encompasses only functional and non-functional systems tests, which takes place after testing the individual ARTICONF tools and their integration with the use cases. The objective of the validation plan is to check that the platform fulfils the requirements of the use cases and to check the final KPIs of the project. In this context:

- Functional tests check the functionality and integration of the use cases with the ARTICONF platform and tools;
- Non-functional tests check non-functional aspects of a software application, usually related to performance and usability under different thresholds of stress.

Table 4, Table 5, Table 6 and Table 7 gather the initial requirements of the four use cases under evaluation, as described in the deliverable D2.1.

To test all these requirements, we will define a specific set of experiments (systems tests) after the integration of the use cases with the prototype of ARTICONF platform in the testbed. We will explain the system tests to carry out the evaluation of the platform in upcoming deliverables within WP7 in accordance to appropriate use case.

Table 4: Crowd journalism use case requirements.

Requirement	Description	Success Criterion
ART.REQ.MOG.1	Guarantee full quality raw input data bandwidth from mobile cameras to feed transcoders; Guarantee normalised quality resolution connections between transcoders and switcher.	100 new contracts in 5 minutes
ART.REQ.MOG.2	Horizontally scale transcoders (based on the number of cameras) and replicating components.	Dynamic scale between 1 and 50 users
ART.REQ.MOG.3	Vertically scale switcher to accept all incoming channels in high resolution.	On demand provisioning of resources
ART.REQ.MOG.4	Choreograph deployment and scale interconnectivity of all distributed components; Specify and monitor QoS/QoE for each component.	Automated deployment, scaling and orchestration of application modules
ART.REQ.MOG.5	Provide a decision mechanism that determines where to deploy the application based on geographical location of crowd location.	Edge based deployment (when applicable)
ART.REQ.MOG.6	Maintain compatibility with the operating systems required by the use-case applications; Build functional blocks for x86_64 architectures.	x86_64 support
ART.REQ.MOG.7	Map use case functional blocks to virtual containers	Virtual containers support
ART.REQ.MOG.8	Offer a high degree of reliability.	99,99% reliability
ART.REQ.MOG.9	Provide high availability in every functional block and in the global system.	99,99% service availability
ART.REQ.MOG.10	Ensure operational, productivity, and strategic ROI for efficiency of investment and revenue growth.	30% better investment and revenue; 80% improved engagement consumer rate
ART.REQ.MOG.11	Provide efficient and configurable blockchain deployment for smart transactions between content providers and content consumers.	Automated deployment of blockchain framework
ART.REQ.MOG.12	Provide mechanisms to define, create and execute smart contracts between users associated with different crowd journalism use case deployments.	10 new smart contracts per minute
ART.REQ.MOG.13	Provide tools to increase the speed of adding members in the blockchain consortium.	10 new users per minute to blockchain consortium
ART.REQ.MOG.14	Define and implement configurable security measures between blockchain members.	Blockchain security configurable toolkit
ART.REQ.MOG.15	Provide tools to store user and content identity in blockchain and video data in separate storage; Make necessary links among different elements.	Federated use case data
ART.REQ.MOG.16	Provide mechanisms to lower fake news in crowd journalism and malicious actors.	80% improved information accuracy
ART.REQ.MOG.17	Provide democratic and decentralised content classification by studying user behavioural patterns.	80% improved information accuracy

Table 5: Car sharing use case requirements.

Requirement	Description	Success Criterion
ART.REQ.AGI.1	Guarantee deployment of a high number of smart contracts in short time.	100 new contracts in 5 minutes
ART.REQ.AGI.2	Provide a continuous service at any time.	99,99% service availability
ART.REQ.AGI.3	Horizontal scaling according to demand; Easily deploy use cases keeping connectivity and interoperability among components.	On demand resources provisioning
ART.REQ.AGI.4	Provide federated blockchain network with correct communication of user information.	Same user data across different networks
ART.REQ.AGI.5	Store data streams from different sources; Provide elastic resource provisioning; Provide fast response and store throughput.	Real-time data gathering from social network and geolocation data sources
ART.REQ.AGI.6	Choreograph deployment, scaling and interconnectivity distributed components; Specify and monitor QoS/QoE for components.	Automated deployment, scaling and orchestration of components; 32% better reaction time
ART.REQ.AGI.7	Provide decision mechanism to classify and remove malicious content in real-time; Score and trace malicious users.	90% reliable use
ART.REQ.AGI.8	Analyse user data, classify users and set strategies to target them; Increase ROI for car sharing companies and private vehicle owners.	80% improved engagement consumer rate
ART.REQ.AGI.9	Allow exchanges among platform participants to reduce car sharing cost.	40% cost reduction
ART.REQ.AGI.10	Easily deploy a new blockchain network; Define parameters for composing network.	Automated parameterised blockchain network deployment

Table 6: Crowd-cooperative video creation use case requirements.

Requirement	Description	Success Criterion
ART.REQ.VG.1	Reputation point ledger: provide a public persistent reputation point ledger attached to DIDs.	100 new contracts in 5 minutes
ART.REQ.VG.2	Token balance ledger: provide a public persistent token ledger attached to DIDs.	Dynamic scale between 1 and 50 users
ART.REQ.VG.3	Immutable transactions: prohibit rollback once reaching consensus about transactions validity and storing it on underlying ledger.	On demand resource provisioning
ART.REQ.VG.4	Low transaction confirmation time: confirm video metadata write and reputation point/token transactions on underlying ledger in real-time.	Automated deployment, scaling and orchestration of application components
ART.REQ.VG.5	Low query time: retrieve metadata, balances and transaction data from underlying ledger in real-time regardless the query initial location.	Edge based deployment (when applicable)

ART.REQ.VG.6	Standard (public) interface: store and retrieve valid data and transactions from ledger through standard HTTP endpoints with publicly accessible queries.	x86_64 support
ART.REQ.VG.7	Scalability: scale to thousands of concurrent users	Virtual container support
ART.REQ.VG.8	Uptime guarantee: high accessibility and reliability to reduce use case maintenance costs.	99,99% reliability

Table 7: Smart energy use case requirements.

Requirement	Description	Success Criterion
ART.REQ.BY.1	Allow individuals integrate, operate and interact with smart energy platform using their social identity management system (tied with Norwegian services sector); Allow use case owners use PCA from TIC to register themselves.	Integration with public identity system
ART.REQ.BY.2	Allow individuals register energy profile and utilities with the system, integrate existing smart meter APIs, solar panel meters and other IoT-enabled utilities, register and encrypt data streams, and store data fingerprint on TIC blockchain.	Integration with third party energy system
ART.REQ.BY.3	Allow individuals advertise and trade energy within neighbourhood and grid; Allow grid providers bid on their energy; Use TIC relationship management package to create smart contracts for this purpose, hosted on blockchain.	99,99% service availability; 1000 transactions per second
ART.REQ.BY.4	Allow deployment of reward mechanisms for individuals to earn incentives for trading and redeem value, facilitated by the grid operation as a minor beneficiary; Use relationship management to create smart contracts that allow deployment of reward mechanisms and tracks fulfilment of such schemes, hosted by blockchain package.	Reward system that allows configuration of custom rewards and redeem scheme
ART.REQ.BY.5	Allow deployment of gamification tasks issued by grid operator that specify weekly goals to drive user behaviour towards optimal energy use; Allow individuals earn rewards by adhering to such tasks; Use relationship management to create smart contracts hosted by blockchain package.	Incentivation system for custom reward schemes and monitoring success criteria
ART.REQ.BY.6	Offer secure and rewarding mechanisms to share in-house energy production and consumption to grid operators for better planning and scheduling without violating the household privacy using PCA, relationship management, data storage and blockchain packages of TIC.	Analytics system that provides insights into micro grid production and consumption
ART.REQ.BY.7	Facilitate in-house energy consumption change leading to peak shaving and reduction of greenhouse emission.	20% less peak hours; consumption; 5% less in-house emissions

5.2.1 Validation Schedule

Figure 18 illustrates the timeline of the evaluation together with the milestones of the project covering two expected prototypes. The first prototype is due at month M24 and the second prototype at month M30. Following this schedule, the first prototype will represent the first attempt to evaluate the platform, while the final evaluation starts at month M34 (representing milestone 6 of the project) after finishing the final integration of the use cases and lasts until the project completion. The QA Team (see Section 5.3) manages, supervises and performs the evaluation of the platform. We will document the results of these tests and evaluations following the template in Annex III (see Section 9.3).

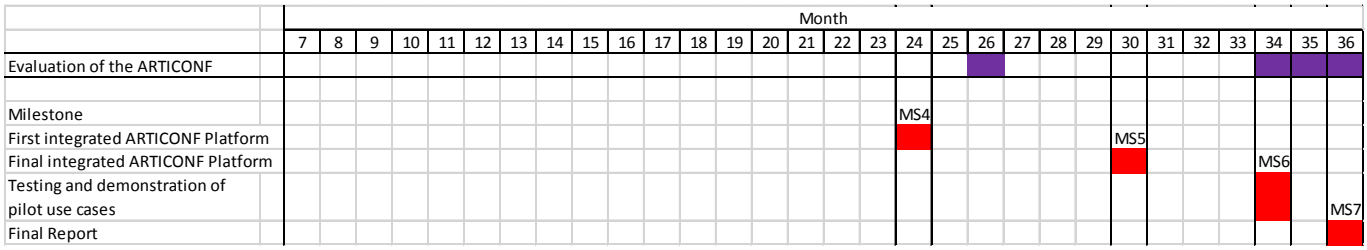


Figure 18: Timeline of the ARTICONF evaluation process.

5.3 Quality Assurance Team

A QA Team controls the testing procedure along with the evaluation of the platform. The objective of this team is to supervise the completeness of the testing process along with documentation associated with it. At this stage of the project, the roles and people appointed in the QA Team are as follows:

- **Testing Manager** is Cristian Martín from AGI whose responsibilities are to:
    - Schedule of testing activities, creating a timeline that fits in the software release planning of the project.
    - Ensure that the Testing Team has the necessary resources to execute the testing activities;
    - Prepare the report of testing activities and make them available to the testing team;
    - Regularly update the scientific coordinator about the progress of testing activities.
- **Testing Team** consists of seven groups, one per technical partner in the consortium, responsible for their own parts of the testing activities. A Test Team Leader conducts each group ensures the its communication with the Testing Manager, as follows: Alexander Lercher (UNI-KLU), Rusell Wolff (BY), Spiros Koulouzis (UvA), Pavel Taskov (UIST), Pedro Santos (MOG), Jorge Pérez Cerpa (AGI), and Jeremiah Smith (VG). The tasks of the Testing Team are to:
    - Design and perform unit tests and functional tests over the use cases;
    - Provide information to the Test Team Leader on how to perform system tests over the testbed to evaluate final KPIs;
    - Inform to the Test Team Leader about the resource and requirements for software testing;

- Prioritise testing activities to meet schedule requirements;
- Document the test cases and report defects and solutions for the anomalies.

The annexes in Section 9 include the documentation templates for the testing and evaluation reports.

6 Software and Tools

During the project and particularly for the testbed used, the ARTICONF partners will work with several software services and tools, which we explain in this section.

GitLab[2] is the main software management tool of ARTICONF. GitLab is an application designed for all stages of the project lifecycle allowing the different teams work together and concurrently on the same project. GitLab enables teams to collaborate and work in a single tool, instead of managing multiple application and channels of communication. Moreover, GitLab provides a single data store to keep the code safe, one friendly user interface, and one permission model, allowing teams to collaborate and focus on building quality software. Lastly, GitLab includes tools for continuous integration and delivery such as Jenkins.

Jenkins[3] is an open source automation tool used to build and test the software, allowing continuous integration and delivery. The concept of continuous integration is one of the basic pillars of a quality project. Jenkins facilitates a continuous integration approach allowing early detection of integration failures. It also provides automated testing, and enables the automated building and deployment of the project on the testbed with a stable code. It has a high impact on the speed of development and on the effort to accomplish a good integration. We chose the Jenkins tool owing to its capability of adapting to any technology or language through open source plugins.

SonarQube[4] is a web-based open source platform to measure and analyse the source code quality. SonarQube receives source code files as input, analyses them, calculates important metrics, and displays them on a dashboard. This analysis automatically detects bugs and alerts developers to fix them, warns about code smells, informs about duplicate code, and provides rating with respect to the quality of the code. This allows the developers fix and improve the code, keeping the quality intact with improved sustainability, while preventing the occurrence of bugs and security risks in an automated fashion.

[2] https://about.gitlab.com/
[3] https://jenkins.io/
[4] https://www.sonarqube.org/

7 Conclusions

This deliverable compiled the information related to the initial ARTICONF infrastructure to deploy and test its tools, APIs, software services, testing and deployment management, and use cases. This infrastructure encompasses different resources provided by the project partners as a combination of consortium private and public resources and commercial cloud providers. The deployment of the code uses this testbed following a continuous integration approach. This methodology allows developers continuously add new functionalities to the platform from a common software repository, enabled and provisioned to the consortium. To facilitate this procedure, we selected several open source services expected to provide a stable technical development environment.

This deliverable also established a formal testing process for the project, covering several layers of testing to ensure a healthy and high quality code, as well as integration and functional integrity. The process follows a timeline aligned with the project milestones and expected software releases of the final platform. Although the specific tests are still under design, we proactively defined an initial platform evaluation plan that measures the different platform metrics through its use cases. The work schedule defined for this evaluation provides a balanced view in accordance to the testing and timeline of the ARTICONF platform prototypes.

This document serves as a foundation for the next steps in WP7 and includes relevant information about porting the use cases, as well as platform testing and evaluation reports. We will cover such documentation in the upcoming deliverables and according to the project developments.

8 Abbreviations

Abbreviation	Description
AI	Artificial Intelligence
API	Application Programming Interface
ARTICONF	smART soclal media eCOsystem in a blockchaiN Federated environment
AWS	Amazon Web Services
CONF	Collocated and Orchestrated Network Fabric
DID	Digital Identifier
GPS	Global Positioning System
KPI	Key Performance Indicator
MS	Milestone
PCA	Personal Certificate Authority
QA	Quality Assurance
QoE	Quality of Experience
QoS	Quality of Service
REST	Representational State Transfer
ROC	Return on collaboration
ROI	Return on investment
SMART	Semantic Model with self-adaptive and Autonomous Relevant Technology
TAC	Tools for Analytics and Cognition
TIC	Trust and Integration Controller
VM	Virtual Machine
WP	Workpackage

9 References

Alexandre Ulisses, Ivone Amorim, et al. (2019). ARTICONF Platform Requirements and Use Cases. ARTICONF Consortium, http://articonf.eu.

Radu Prodan, Nishant Saurabh, et al. (2019). ARTICONF Architecture and Interfaces Definition. ARTICONF Consortium, http://articonf.eu.

10 Annexes

10.1 Annex I: Template for Unit and Integration Tests

ARTICONF Project: Unit / Integration Tests								
Tool or Use Case under Test								
Test ID	**Test Type**	**Test Description**	**Inputs**	**Outputs**	**Test Date**	**Associated Tool / Use Case**	**Test Status**	**Comment**
#	*Unit / Integration*	*Describe the function, interaction or module to test*	*Inputs for the unit test*	*Expected outputs*		*Tool: TIC / TAC/ SMART / CONF* *Use case: AGI / MOG / BY/ VG*	*Passed / failed*	*If failed, explain why*
#								
#								
#								
#								

10.2 Annex II: Template for Functional Tests

ARTICONF Project: Functional Tests						
Tool or Use Case under Test						
Test ID	**Test Description**	**Preconditions / Dependencies**	**Expected Result**	**Test Date**	**Actual Result**	**Test Status**
#	*Description of the test case*	*E.g. previously registered user for a login*	*Expected outputs*		*Only if different to expected result*	*Passed / failed*
#						
#						
#						
#						

10.3 Annex III: Template for Platform Evaluation

ARTICONF Project: Evaluation Tests							
Test ID	**Test Type**	**Test Description**	**Precondition / Dependencies**	**Expected Result**	**Test Date**	**Actual result**	**Test Status**
Same id than in other deliverables	*Functional / Non-functional*	*Description of the test*	*E.g. previously registered user for a login*	*Expected outputs*		*Result of the test*	*Passed / Failed compared to initial requirements or KPIs*
#							
#							
#							
#							