Marie Sklodowska Curie, Research and Innovation Staff Exchange (RISE)

IdeNtity verifiCatiOn with privacy-preservinG credeNtIals for anonymous access To Online services

WP6 – Platform realization and integration

Deliverable D6.1 “Initial system design and prototyping”

Editor(s):	CSGN
Author(s):	Anca Zugravu (ESR, CSGN – secondee), Petru Scurtu (ER, CSGN – outside secondment), Vaios Bolgouras (ESR, UPRC – outside secondment), Markos Charalambous (ESR, CUT - secondee), Ioana Stroinea (ESR, CSGN – secondee), Nikos Salamanos (ER, CUT – outside secondment), Konstantinos Papadamou (ER, CUT – outside secondment), Evangelos Kotsifakos (ER, LST – outside secondment), George Kalatzantonakis (ESR, LST – outside secondment), Michael Sirivianos (ER, CUT – outside secondment), Christos Xenakis (ER, UPRC – outside secondment), George Gugulea (ER, CSGN – outside secondment)
Dissemination Level:	Public
Nature:	Report
Version:	1.0

INCOGNITO is funded by the European Commission”s Horizon 2020 Research and Innovation Framework program under the Marie Skłodowska-Curie Research and Innovation Staff Exchanges Action, Grant Agreement no 824015. The content of this deliverable reflects only the views of the project owner. The European Agency / Commision is not responsible for any use that may be made of the information it contains.

Project Profile	
Contract Number	824015
Acronym	INCOGNITO
Title	IdeNtity verifiCatiOn with privacy-preservinG credeNtIals for anonymous access To Online services
Start Date	Jan 1st, 2019
Duration	48 Months

Partners

Τεχνολογικό Πανεπιστήμιο Κύπρου	TECHNOLOGIKO PANEPISTIMIO KYPROU (BEN2, CUT)	Cyprus
University of Piraeus	University of Piraeus research center (BEN1, UPRC)	Greece
certSIGN	Certsign SA (BEN3, CSGN)	Romania
Telefónica Telefónica Investigación y Desarrollo	Telefonica Investigacion Y Desarrollo SA (BEN6, TID)	Spain
LS TECH	LSTech Espana SL (BEN4, LST)	Spain
FOGUS INNOVATIONS & SERVICES	FOGUS INNOVATIONS & SERVICES P.C. (BEN7, FOG)	Greece

Document History

VERSIONS

Version	Date	Author	Remarks
0.1	21/01/2021	Anca Zugravu, Ioana Stroinea	Table of Contents
0.2	16/02/2021	Konstantinos Papadamou, Vaios Bolgouras, Markos Charalambous, Evangelos Kotsifakos	TOC revision
0.5	20/03/2021	Konstantinos Papadamou, Vaios Bolgouras, Markos Charalambous, Evangelos Kotsifakos, George Kalatzantonakis, Petru Scurtu	Architectural Components
0.7	20/04/2021	Konstantinos Papadamou, Vaios Bolgouras, Markos Charalambous, Evangelos Kotsifakos, George Kalatzantonakis, Petru Scurtu, Anca Zugravu	Software Components
0.8	22/05/2021	Anca Zugravu, Petru Scurtu, Vaios Bolgouras, Markos Charalambous, Evangelos Kotsifakos	Integration Platform
0.9	25/06/2021	Evangelos Kotsifakos, Michael Sirivianos, Christos Xenakis, Nikos Salamanos, George Gugulea	Document Review
1.0	30/06/2021	Nikos Passas	Final Version

Contract delivery due date: 30/06/2021 (M30)
Actual delivery date of version n.1: 30/06/2021 (M30)

Fellow ID	Name/Surname	Researcher category	Declaration No.	PM
6	Markos Charalambous	ESR	3	3
11	Ioana Stroinea	ESR	18	0.13
23	Anca Zugravu	ESR	21	5.87

Executive Summary

The purpose of this deliverable D6.1 "Initial system design and prototyping" is to present an overview of the initial integrated INCOGNITO platform. It is part of Work Package 6 and it aims to describe the software and hardware components of the system. Moreover, this document covers information about the techniques, tools, technologies and methodologies used to integrate the sub-modules into one system. Having in mind the already defined architectural and software components of the system, we follow the Agile development methodology and we employ CI/CD tools like Jenkins, Docker and Kubernetes, and collaboration tools such as Phabricator, Git and OneDrive to define the initial system design and integration.
The intent of the deliverable is to have a well-defined foundation of the platform upon which more modules will be added as the project continues its progress.

Table of Contents

List of Figures

Table of Abbreviations

ABAC	Attribute-based Access Control
AI	Artificial Intelligence
API	Application Programming Interface
BAC	Basic Access Control
FIDO2	Fast IDentity Online version 2
HTTP	Hypertext Transfer Protocol
ID	Identity
IDC	Identity Consolidator
IdP	Identity Provider
IIM	Identity Integration Module
IoT	Internet of Things
JSON	JavaScript Object Notation
NFC	Near-Field Communication
OAuth	Open Authentication
OCR	Optical Character Recognition
OIDC	OpenID Connect
QA	Qualified Anonymity
REST	Representation State Transfer
RFID	Radio Frequency Identification
SSL	Secure Sockets Layer
SP	Service Provider
TLS	Transport Layer Security
UD	User Device
UI	User Interface
UMA	User-Managed Access
UX	User Experience
XML	eXtensible Markup Language
CI/CD	Continuous Integration / Continuous Deployment
SSH	Secure Shell

1 Introduction

The objective of this deliverable is to provide an overview of the first version of the application and how the software modules will be integrated together.

Three main topics are addressed: the software components that are going to be integrated, the hardware components and the technologies and methodologies used for the integration. We provide a detailed description of the deployment process and software development practices. We also cover the CI/CD (Continuous Integration and Continuous Deployment) pipelines that help with the process automation. Moreover, we have a dedicated part for the software modules that have been already deployed on the integration server.

The aim of this document is to offer enough insight on what technologies and steps are needed in order to recreate the same system from physical infrastructure to software modules. That is why we delivered a comprehensive description of the tools accompanied by their minimum requirements and configuration specifications.

2 Integration methodology

In this chapter we describe the optimal practices the team will be using to link together and integrate all the sub-systems developed in INCOGNITO.
Considering the fact that the researchers involved in this project are spread across Europe, consisting an international team of developers, a series of software engineering practices to collaborate as efficiently as possible are needed.
Moreover, the software modules were not developed sequentially; therefore, it is necessary to diminish the possible errors when integrating them.

Another challenge the developers have to face is finding the optimal work style in the ongoing COVID19 pandemic by making sure that in some cases, not working in the organization premises does not impede collaboration. Maintaining the quality of the cooperation at a high level in order to increase efficiency is of utmost importance.

Agile Development Methodology

Agile software development refers to a set of practices applied to iterative development, where the software is delivered incrementally instead of all at once [1]. This methodology embraces teamwork, self-organization and continuous monitoring and adaptation when new challenges occur.

The Agile methodology was designed to change and improve the Waterfall development model. The Waterfall paradigm follows a linear and sequential method, contrary to the iterative approach of the Agile methodology. This means that tasks and activities in the waterfall model move to the next phase only after the ones in the current phase have been completed.

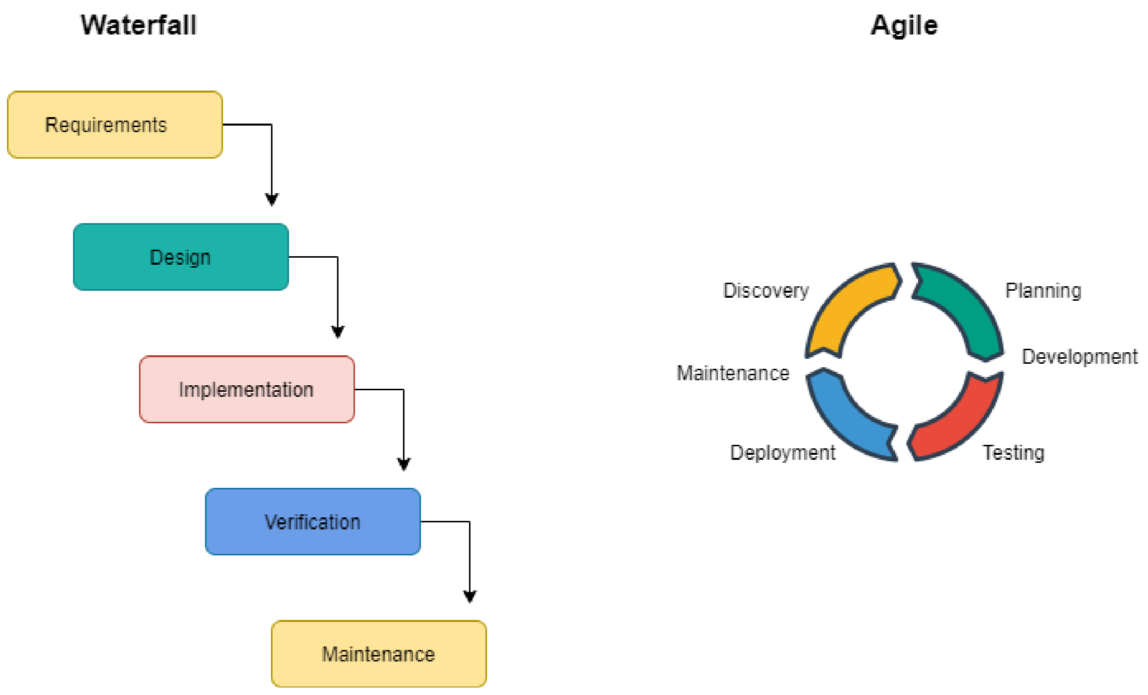


Figure 1. Waterfall Methodology vs Agile Methodology

As seen in Figure 1, in a Waterfall model, the project relies on the initial requirements to a great extent. In Agile, these initial requirements are checked and confirmed throughout the entire development life-cycle. This type of approach is essential not only for the software modules we are developing, but also for the deliverable submission phase when there is a request to review the submitted document. Having flexibility in case of potential changes results in a higher quality end product, be it in the form of deployed software or well documented deliverables.

Adopting the Agile methodology offers us the perfect balance between responding to changes and following a predefined workflow. These changes are not only on a technical level regarding the software, but also on the team level, as a result of a constant flow of new arriving secondees in the project.

Moreover, considering that this deliverable serves as a foundation for the final integrated version of the system, new software modules will be delivered and will require integration until the final months of this project. This is why it is important to have consistency in the integration methodology.

Borrowing some of the Agile methodology principles, there are a few key practices the team is following during the software development process:

1. Organize and keep track of the tasks using Phabricator[1]

[1] https://www.phacility.com/phabricator/

2. Have a centralized version of the code in multiple repositories using GitLab
3. Keep each other updated by hosting online meetings whenever needed
4. Collaborate on tasks of high priority and participate in knowledge sharing activities to speed up the deployment process and gather new skills
5. Share ideas and address technical issues in a quick manner on platforms like Slack, created for software development teams

CI/CD Pipelines in INCOGNITO

A CI/CD pipeline is a series of steps configured with the purpose of delivering new versions of an application and it is devised into two major sub-processes: Continuous Integration and Continuous Delivery. Continuous Integration (CI) means new code changes are regularly built and added to a central shared repository. When developers practice CI, the software and database components are packaged together and the automation executes unit tests. The tests are essential for providing feedback to developers about how the new code they have committed integrated successfully with the already existing one. If the tests fail, a rollback will trigger with the purpose of bringing the application back to a stable functional version. Continuous Delivery (CD) means that every new feature of the application that proves itself to be functional after the automated tests is integrated and deployed with the application running in a production environment. Depending on the tool, the developers usually have the option to choose if the builds are triggered by code commits or on a defined schedule. In more complex systems, the code can be deployed simultaneously to different environments, like production, staging and testing.

The CI/CD concept is substantial because some of the pipeline's steps can be automated, resulting in speeding up the process and minimizing the number of errors.

From task creation to deployment, the workflow of software development in INCOGNITO should follow these steps:

1. The developer creates a task in Phabricator for the feature they are working on or picks a task that has been already created

2. The developer writes the code necessary for the development of the selected feature

3. The developer tracks changes for the written code in a local repository using Git, then synchronizes them with the corresponding remote repository on GitLab

4. Jenkins pulls the code from the central repository on GitLab, runs a suite of tests and builds the deployment artifacts

5. Jenkins creates a docker image with the built artefacts

6. Docker sends the docker image to the Docker Registry

7. Kubernetes pulls the image from the registry and deploys the image inside pods deployed on the integration platform

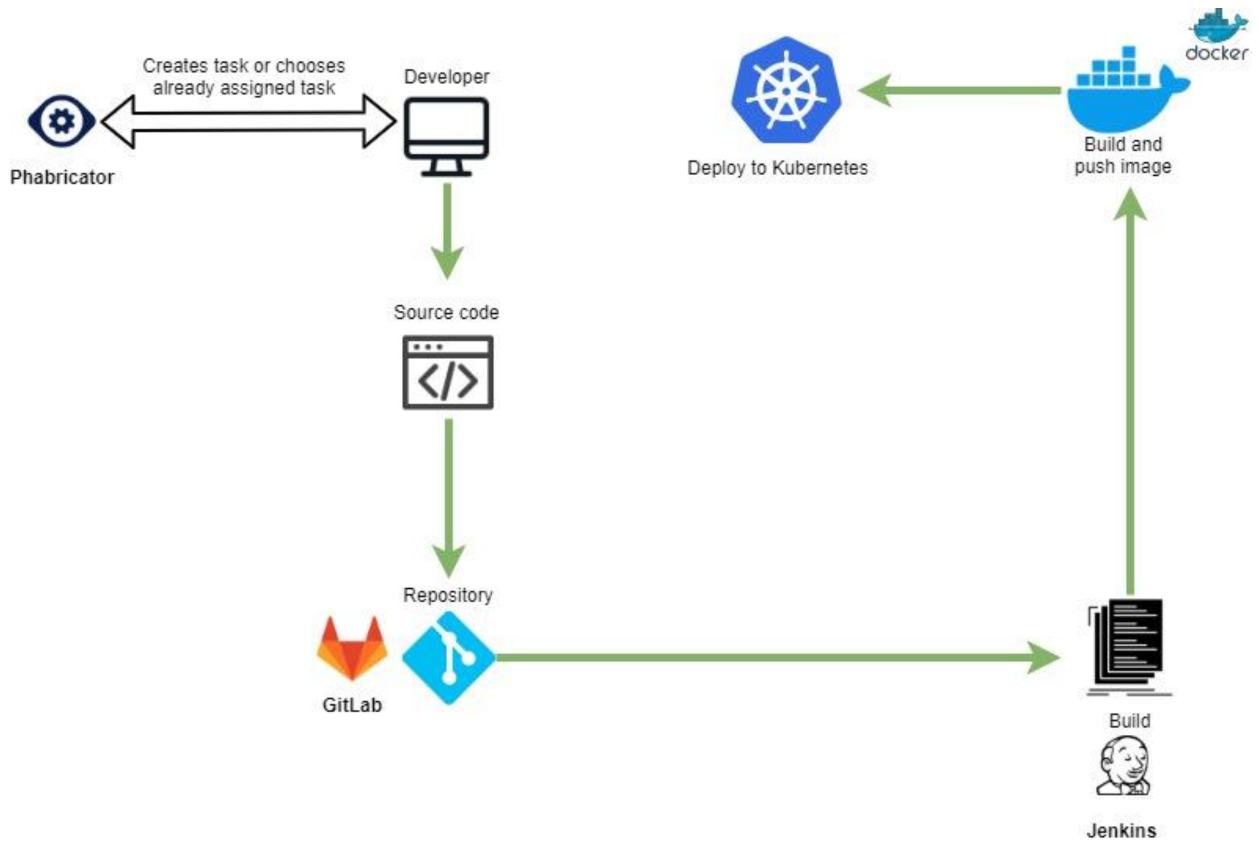


Figure 2. CI/CD pipeline in INCOGNITO

3 The System-level prototype

The INCOGNITO platform consists of a set of software components that are deployed on the hardware infrastructure provided for this project. The software consists of developed modules in the working packages. The hardware is where these modules run and will be integrated. The client-side component of the hardware is the user device, where the user accesses the application. The server-side component of the hardware is where the components accessed by the user device will be kept.

Next, we describe in detail the different components.

3.1 Architectural components

3.1.1 Identity Consolidator

INCOGNITO’s central node is the Identity Consolidator which integrates multiple modules/platforms that offer the user a unique identity acquisition, verification, and management solution:

The Identity Consolidator is the place where the online identity of the user lies. It has modules that gather real-world and online information about a user and makes one central trusted identity.

After transposing this user information into identity attributes at the Identity Consolidator, the latter acts as a connection point between the user and Identity or Service Providers. It mediates and makes trustworthy the communication needed for a user to access a Service Provider's resources by generating cryptographic credentials for the users to authenticate at different Service Providers or by hosting different servers such as FIDO2[2] for secure authentication or User-Managed Access (UMA) for user-based content management.

It also enables users to manage their own identity through modules such as Account Management Module or Identity Management Module. Therefore, the user can enhance the security of their account and restrict the Identity Consolidator's control over their identity by making use of the UMA server. The user can set policies that have a higher priority over the identity attributes than the level of assurance calculated by the Identity Consolidator when integrating the information gathered from different Identity Providers about the user's identity.

The Identity Consolidator also enables the user to use the services of an AI-based assistant that will assist the user in the management and disclosure of their identity. The AI-based assistant will have the expertise to prevent the user from potential risks and allow them to take actions regarding this matter and take steps to protect the user's privacy and security.

[2] https://fidoalliance.org/fido2/

Two different layers of security are added by using a Tor[3] Network to mediate the communication between the user and the Service Providers and blockchain to log user's interactions with the online services such as Identity Consolidator or Service Providers.

3.1.2 User-Device

INCOGNITO uses the mobile device to realize the authentication of the user to a service. In the first step, the user authenticates to the mobile device using a biometric factor. INCOGNITO employs the FIDO2 authentication process that takes part between the local device and the online service. In addition, it extends FIDO2 by: a) adding additional user-to-device biometric authentication factors such as gait and face recognition b) integrating an OpenID Connect authenticator with the FIDO2 authentication server to provide device-centric federated authentication c) introducing cryptographic privacy-preserving attribute-based authentication to the FIDO2 protocol, thus allowing users to cryptographically prove distinct ID attributes instead of account ownership. Furthermore, INCOGNITO supports Attribute-Based Access Control (ABAC) [2] cryptographic credentials to Identity Providers.

The user's cryptographic credentials, which are received from the Identity Consolidator and the multiple IdPs, are stored in the cryptographic credentials storage that is allocated in the user's device. Also, to enhance the security of the user's credentials, the cryptographic credentials storage takes advantage of the user device's Trusted Execution Environment (TEE) capabilities. A cryptographic interface is also included in the user device that communicates with the Idemix protocol stack installed on the user's device. The Idemix credentials that are saved in the Cryptographic Credential Storage can be released from these stacks.

The user device runs federated login protocols for authentication and authorization purposes between the IdP and SPs (OpenID Connect[4]/OAuth 2.0[5]). In addition, we integrate FIDO2 for authentication between the user device and the Identity Provider, which supersede the standard password paradigm.

The user can control his identity information with the identity and access control management application, which is also included on the user device and communicates with the IDC. The application enables the user to manage his identity attributes exposed to each Service Provider. The user can also issue cryptographic credentials from his identity attributes directly to his device and use them for ABAC. The consent management is also included in the application and ID privacy functionality that enables the users to have knowledge and control over which Service Provider knows which aspects of his identity.

There is also an AI-based Assistant on the user device, which communicates with the Identity Consolidator to inform the user and guide him into properly managing their identity. The AI-based Assistant notifies the user, for example, about the minimum identity attributes required to be

[3] https://www.torproject.org
[4] https://openid.net/connect/
[5] https://oauth.net/2/

revealed to the Service Providers in order to get access to its resources. Also, the AI-based Assistant has the ability to inform the user about the risk of revealing specific identity attributes to Service Providers, which in turn may enable the SPs to infer the complete identity of the user.

The device has an identity acquisition module that will allow the user to quickly and securely acquire identity attributes from online identities (Facebook account, etc.), as well as his/her physical ID Documents (e-Passports, eID, etc.) with the Near-Field Communication[6] (NFC) protocol. The user will have the option to store the acquired and verified identity attributes to the IDC.

3.1.3 Identity Provider

In INCOGNITO, the Identity Provider is a trusted system that authenticates users on behalf of another web resource, such as Service Providers. More specifically, Identity Providers are responsible for transferring and securely maintaining user’s identity attributes. They incorporate robust authentication mechanisms so that they can regulate user access. In addition, they are responsible for issuing and verifying the user’s cryptographic credentials.

The Identity Providers rely on two key servers. The first one, FIDO 2.0 server that represents how users register and authenticate at the Identity Providers. The second one is a QR[7] authentication server that allows the user to access a service from another device or browser if it is already using that service on their user device. A QR Client existent on the user’s device will enable the user to scan a QR code and send it to the QR Authentication server for verification. In this way, the Identity Provider will be able to authenticate the user and give access to resources from a different device than the one he scans the QR Code from.

3.1.4 Service Provider

The Service Provider is an entity that is responsible only for authorizing users to their service. All the other crucial operations like authentication and verification of credentials are performed by delegating them to the Identity Providers, which are trusted entities via Federated solutions like OpenID Connect.

The Service Providers use the XACML-based Access Control Policy Reasoning Tool to define access control policies regarding the users’ identity attributes required to gain access to their services. These attributes are acquired from the Identity Consolidator, as users do not directly share information regarding their identity with the Service Providers. The OpenID Connect protocol combined with a cryptographic credentials stack, such as Idemix, is utilized to deliver the attribute(s) needed by the Service provider to grant access to the user.

[6] https://en.wikipedia.org/wiki/Near-field_communication
[7] https://en.wikipedia.org/wiki/QR_code

In addition, INCOGNITO will also incorporate a blockchain solution to boost its privacy and security. The Service Providers will be part of that network, running an endpoint and participating in the submission of transactions on the blockchain, while they will be held accountable for their actions throughout the network’s lifecycle.

3.2 Software Components

The software modules developed in the project are hosted on GitLab and they can be accessed on: https://incognitogit.ds.unipi.gr/incognito.

3.2.1 Identity Acquisition and Management

The Identity Acquisition and Integration platform are among the significant components of the INCOGNITO architecture. This platform plays a significant role in most piloting activities and uses cases of the INCOGNITO platform since it facilitates the seamless integration of the multiple soft proofs of identities (both physical and online) of a user. More specifically, this platform is responsible for horizontally binding the online identities of a user and vertically binding the physical identities of a user to independently verifiable identity attributes.

The platform contains the Physical Identity Acquisition and Verification module, which is responsible for securely acquiring and verifying all the user's identity information from their real-world identity documents. The functionality of this module is exposed to the users through web and mobile application interfaces, which allow us to leverage innovative, trusted-computing enabled devices (e.g., mobile phones) to securely acquire all the physical characteristics and the information included in the real-world identities of the users.

In addition, this module uses trusted software paths of commodity devices to securely capture, through the device's camera device, pictures of the user and their physical identity documents. Identity information of the users (i.e., location) used for verification of his identity attributes is captured through the sensors available on commodity mobile devices. Upon acquiring all the users' identity information, we use various verification algorithms to validate all the collected identity information.

The identity verification process uses existing techniques (such as OCR) and peer-to-peer (crowdsourcing) techniques. Automated verification is established on the acquired photos of the users using face detection, face recognition, and Optical Character Recognition. Peer-to-peer verification using crowdsourcing techniques takes place to verify that the information on the acquired photos matches the users' declared identity information and physical characteristics. All the verified identity information is then stored in the Identity Attributes Storage as independently verifiable identity attributes. The Identity Acquisition and Verification module support additional verification of the collected identity information through remote identity verification by trained professional auditors leveraging the WebRTC protocol.

We note that having enough information about a user, the Identity Acquisition and Verification module is in place to infer other identity attributes by aggregating the collected identity information of a user.

The horizontal binding of the various online identities of a user (e.g., Facebook, Google+, etc.) is performed by the Online Identity Acquisition module. For each online account, an online authentication and authorization process using federation authentication protocols (i.e., OpenID Connect) is required so that the user gives explicit authorization to the Online Identity Acquisition module to access and retrieve his account's personal information. Following the authentication process, the acquisition of the attributes takes place, and the collected attributes are stored in the Identity Attributes Storage.

At the same time, the Identity Integration and Normalization module, which runs as a background service, is responsible for the normalization of all the collected identity information of the user into independent verifiable distinct identity attributes and for assigning confidence to each normalized attribute. In other words, it is responsible for aggregating and connecting the acquired online and physical identity attributes of the user and inferring the veracity of the claimed identity attributes via means of statistical data analysis techniques. The Identity Integration and Normalization module is also responsible for assigning confidence scores for the integrity of the attributes for labeling identity attributes based on their origin. In the end, all the normalized attributes are stored in the Identity Attributes Storage of the IDC.

All the aforementioned modules interact with the Identity Attribute Storage using the Identity Attributes Storage REST API. This API enables all the previously described modules to interact with the Identity Attributes Storage for READ, WRITE, and UPDATE the user's identity information. On the other hand, external entities outside the IDC and third parties can communicate with the IDC and its Identity Attributes Storage.

Last, users will be able to authenticate and interact with the IDC and the web and mobile applications of the Identity Acquisition and Integration platform various diverse authentication mechanisms like a FIDO-enhanced OpenID Connect mechanism described in other deliverables of INCOGNITO.

The Figure below depicts the architecture of the first of the initial version of the IDC which includes all the modules that are part of the Identity Acquisition and Integration platform.

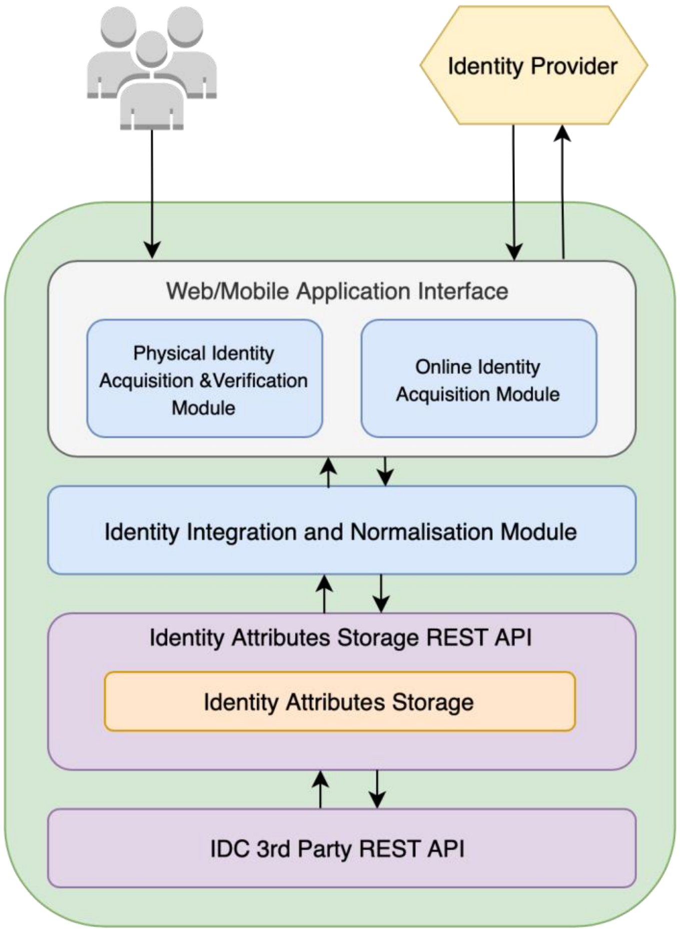

Identity Provider
Web/Mobile Application Interface
Physical Identity Acquisition &Verification Module
Online Identity Acquisition Module
Identity Integration and Normalisation Module
Identity Attributes Storage REST API
Identity Attributes Storage
IDC 3rd Party REST API

Figure 3. Identity Acquisition and Integration Platform

3.2.2 Qualified Anonymity software components

- Idemix

Idemix is an attribute-based credential scheme which allows users to be authenticated in an anonymous manner. The INCOGNITO project leverages the advantages of anonymous credentials by using the IRMA implementation of Idemix.
IRMA implements the Idemix attribute-based credential scheme, allowing users to safely and securely authenticate themselves as privacy-preserving as the situation permits. The main advantages of using this scheme are:

- Unlinkability: consecutive verifications of the issued credentials for an attribute cannot be traced back to the disclosing user
- Unforgeability: by using a blind signature scheme the authenticity of the issued credentials is guaranteed

- Immunity to replay attacks: nonces are used in the disclosure process to prevent consecutive disclosure sessions to produce the same output; verifiers cannot deduce the attribute from the proof of knowledge.

IRMA uses three main components:

- an Issuer of anonymous credentials, a corresponding Verifier component and the end entity which is usually a User with the corresponding User Device. The Issuer component is tasked with issuing anonymous credentials upon user requests. The issuance process will be handled in a secure manner after the user is authenticated and the validity of the provided credentials is proven.
- The issuance and verification of the anonymous credentials is based on a blind signature scheme. As a result, each Issuer is in possession of a secret key which is not disclosed and is used only in the issuance process. Each secret key has a corresponding public key which can be used by the Verifiers to attest that the credential corresponding to the disclosed attributes is valid and authentic. Due to the nature of the cryptographic scheme employed it is impossible for the Verified to derive the attributes from the received proof of knowledge.
- The User Device is responsible for securely storing the issued credentials. To achieve this it can make use of secure encrypted storages and use secure cryptographic mechanisms such as Trusted Execution Environments. It is also responsible for authenticating the user prior to disclosure sessions and guarantying that the credentials are used only in disclosure sessions.

IRMA support multiple session types:

- Issuance session: an end user may request the issuance of a new credentials which corresponds to one or multiple user attributes. The Issuer uses the private key to sign the issued credentials
- Disclosure session: an end user uses the received credential is order to generate a proof of knowledge. This proof does not disclose the original attributes and can be used by a Verifier in conjunction with the Issuer public key to prove the authenticity of the proof.
- Attribute-based signature session: similar to disclosure sessions they imply generating a digital signature using the credentials. The signature can be validated at a later time to prove the authenticity of the message and that the credentials were valid at signature time.

- OpenID Connect - FIDO 2.0

The federated identities used on the INCOGNITO platform for authentication purposes, heavily rely on the OpenID Connect (OIDC) and FIDO technologies, which are deeply intertwined. The FIDO2 Authentication and OIDC protocols complement each other, even though they have different objectives. FIDO2 protocol aims in achieving authentication without requiring the employment of traditional password methods or the transfer of attributes. FIDO2 does not require the definition of user attributes, but rather provides the confirmation that a user’s claim is true, without revealing the entirety of an identity. OIDC on the other hand, addresses the issue of transmitting user attributes among network participants in a secure manner.

For the prototyping process, we have followed the design described in detail in D3.1. More specifically, OIDC makes it possible for the SPs to delegate the authentication of end-users to the IdCs, which also act as the OIDC Providers. Keycloak, the open-source Identity and Access Management solution we utilize at the INCOGNITO platform, lies on the Identity Consolidator and acts as the OIDC provider. The FIDO2 technology on the other hand, gives the ability to end-users to authenticate to their IdPs by making use of strong authenticators and cryptographic protocols. The FIDO2 Client is utilized by the users on their devices, sends JSON messages with the corresponding signature to the FIDO2 Server for authentication purposes. In turn, the FIDO2 Server residing on the IdC, validates the authenticity of a user. The federated authentication flow followed in the prototype at this stage is very well summarized in the following diagram:

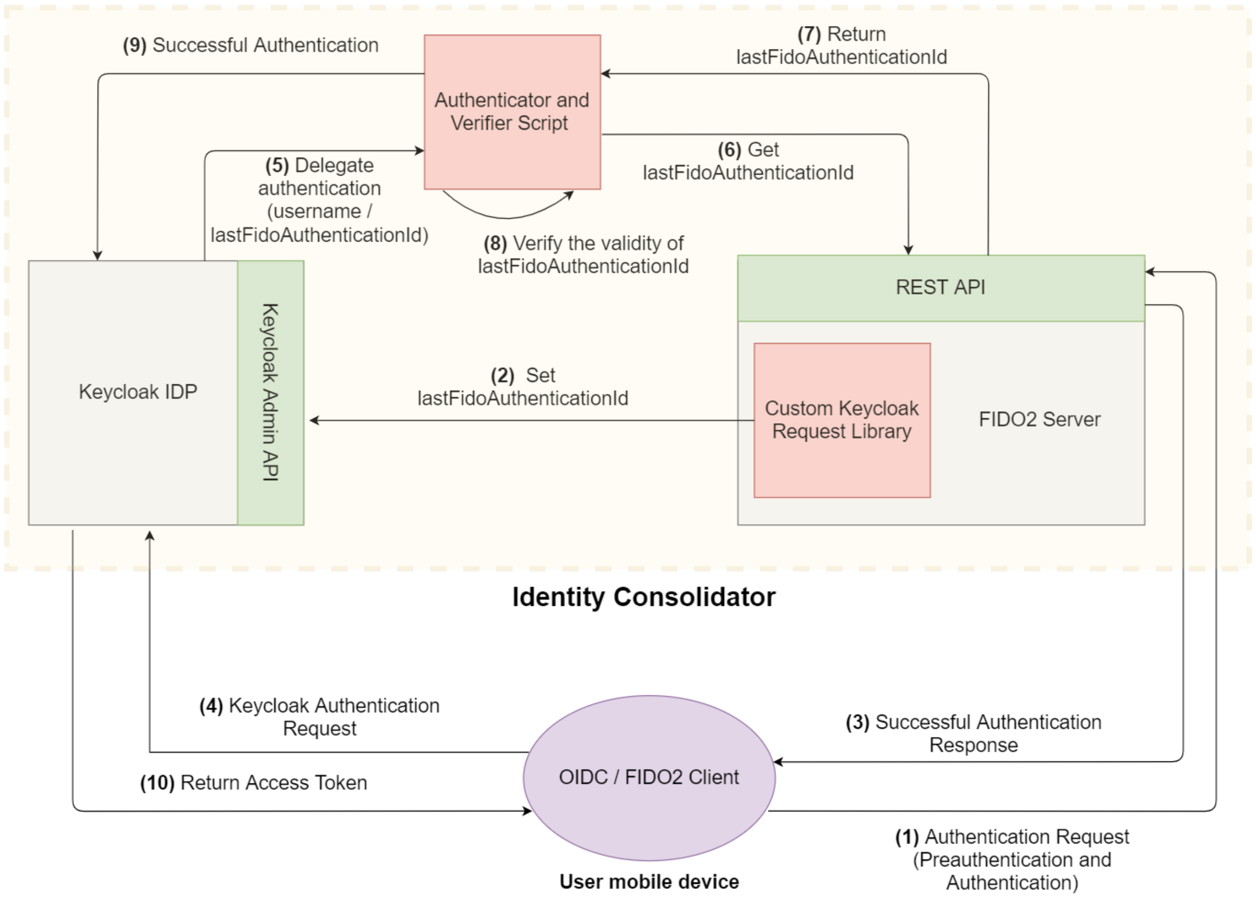

(9) Successful Authentication
(7) Return lastFidoAuthenticationId
Authenticator and Verifier Script
(6) Get lastFidoAuthenticationId
(5) Delegate authentication (username / lastFidoAuthenticationId)
(8) Verify the validity of lastFidoAuthenticationId
REST API
Keycloak IDP
Keycloak Admin API
(2) Set lastFidoAuthenticationId
Custom Keycloak Request Library
FIDO2 Server
Identity Consolidator
(4) Keycloak Authentication Request
(3) Successful Authentication Response
OIDC / FIDO2 Client
(10) Return Access Token
User mobile device
(1) Authentication Request (Preauthentication and Authentication)

Figure 4. Federated authentication flow

- Trusted Execution Environment (TEE)

The incorporation of a trusted execution environment on a device allows for the deployment of trusted application without making use of the normal computational resources. Moreover, trusted applications executed on TEE, along with their resources, cannot be accessed by other non-trusted entities. Thus, the trusted execution environment manages to achieve the following functionalities when deployed in an information and communications technology system:

  - Trusted applications can be executed in a confined manner, limiting the connection among the processes.
  - Application which are not inherently required to be executed in such a secure environment, can perform their tasks in the same computing setting.

In the context of the INCOGNITO platform, Open-TEE is currently the solution that’s ahead of its other competitors and is being implemented. This trusted execution environment is closely related to the anonymous credentials produced by Idemix. These credentials require the secure cryptographic primitives offered by the TEE in order to issue the corresponding credentials, and storage locally on the client. The Open-TEE solution will be integrated on the devices/components that require the use of cryptographic processes and secure storage.

- TOR

TOR is a software used to provide online anonymity. It is based on the concept of Onion Routing, a technique that encapsulates the content of a message in multiple layers of encryption. In onion routing, instead of making socket connections directly to a responding machine, initiating applications make connections through a sequence of machines called onion routers. The onion routing network allows the connection between the initiator and responder to remain anonymous.

In INCOGNITO we use TOR to route the HTTP requests sent from the User Device to the Identity Provider and Service Provider through the Tor Network. This way we are hiding the user's IP Address, so no third-party application can link the user's IP to any personal information, like location. In order to integrate TOR in our architecture, we are using a Tor Library for Android and deploying it together with the rest of the mobile application.

3.2.3 Decentralized Identity Management

The Decentralized Identity Management module is being currently implemented. The Hyperledger Fabric has been chosen among others as the development framework due to the advantages it provides, which can be found in D2.3. The blockchain technology will give the ability to the INCOGNITO platform, and more specifically the IdC/IdP, to manage identity attributes in a distributed and privacy-preserving manner. The INCOGNITO blockchain structure serves as a secure and immutable storage of “Identity Associations”, that is not susceptible to the “single point of failure” threat and can be accessed only by entities that have been granted access. Along with the user attributes, policies set from the user through Keycloak regarding which identity attributes the user will share with specific service providers, will be logged.

The entities that may be required to access the blockchain ledger, will have to incorporate in their implementation the Fabric Gateway. The Fabric Gateway is a core component of the Hyperledger Fabric-based blockchain network and coordinates the actions required to submit transactions and query ledger state on behalf of client applications. By using the Gateway, client applications only need to connect to a single endpoint in the Fabric network.[8] Using that connection, functions created in the smart contract of the blockchain can be called in order to execute certain actions, e.g. query an identity attribute from a user. The aforementioned are depicted in the following figure, where we have included the IdC and an SP indicatively.

[8] https://hyperledger.github.io/fabric-gateway/

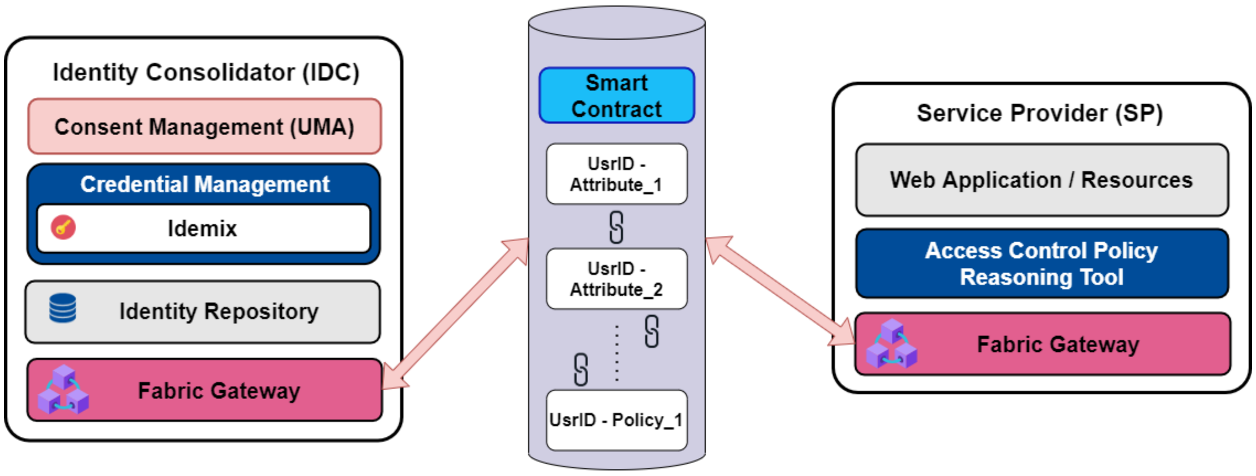


Figure 5. Decentralized Identity Management

3.2.4 Consent Management/User-Managed Access

INCOGNITO leverages a consent management module which allows users to obtain control over the information they share. In order to authenticate in an anonymous manner, the user will disclose relevant information in the form of proof of knowledge of attributes. These proofs can be verified by the Service Providers at the corresponding Authorization Providers. The consent management will help users identify and manage which attributes they wish to disclose.

To achieve the goals of the consent management module, UMA protocol will be used. UMA 2.0 is a federated authorization standard protocol built on top of OAuth2.0. It allows users to manage their own resources and decide with which entities of the ecosystem they wish to share them. In addition to that, the UMA 2.0 standards enable users to always feel in control of their attributes, offers them the ability to make an independent decision and the choice to establish when it’s the right time to share their attributes, making security the most important aspect to take into consideration.

Keycloak acts as an Authorization Server and provides the necessary functionality for using the UMA protocol for consent management. It consists of a set of administrative UIs and a RESTful API designed to provide the necessary means for managing the protected resources of clients and users. User-Managed Access specifications and leading standards such as OAuth2 are at the base of Keycloak Authorization Services. The UMA REST API is part of the authorization services and will be used by the consent management module. The Protection API of Keycloak, which is part of the UMA REST APIs, exposes endpoints for the required actions needed in order to satisfy the project’s needs.

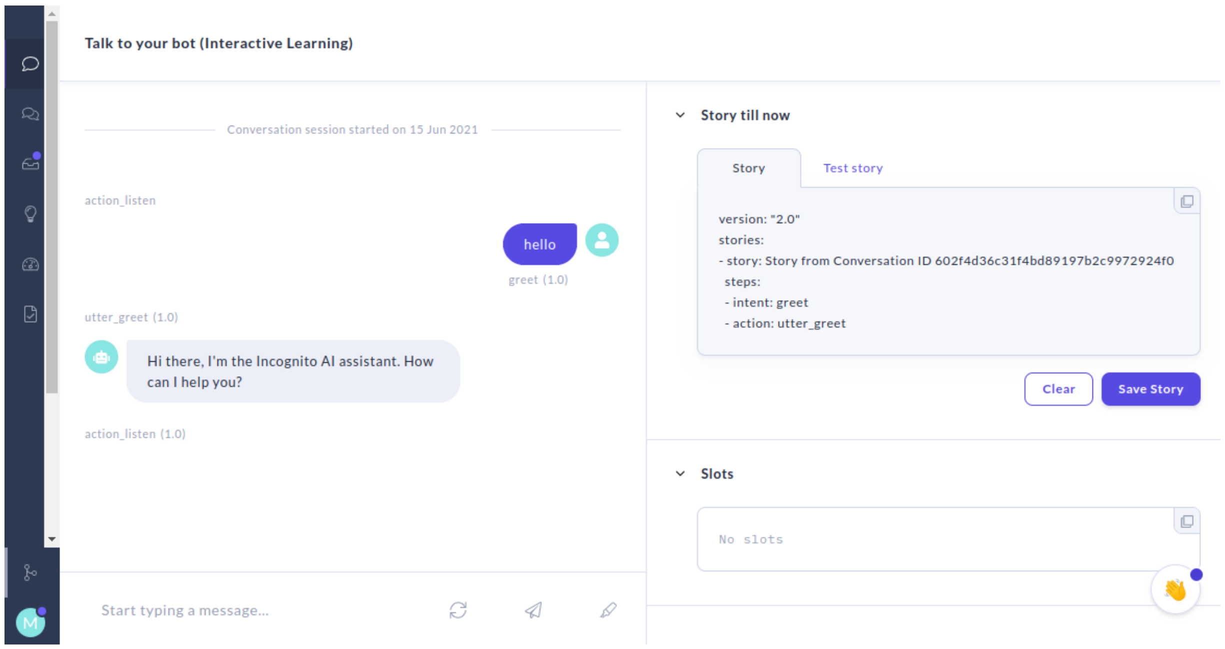
In Keycloak, the term client refers to applications or services that are registered with the Keycloak server in order to acquire security related functionalities. The term user refers to any individual registered with Keycloak who can log into the system. The object being protected is referred to as a resource in authorization policy terminology. A resource can belong to a client or to a user, in which case the corresponding entity is denoted as the resource owner. In INCOGNITO, the user is the owner of his resources and can define access policies for accessing them which are referred to as “permissions” in Keycloak terminology. A permission links the protected object to the policies that must be evaluated to determine if access should be granted. Those permissions can then be edited, removed or queried by using the Protection API of Keycloak.

3.2.5 UI/UX AI-based Assistant

The UI/UX AI-based assistant has two main parts, the front end and the back-end. The front end lies on the user device, being the mobile or the PC through the web-browser. This front end is getting user commands/questions and transfers them to the back-end that processes them and in connection with the IDC module, it formulates the response to be sent back to the front end and the user.
For the implementation of the UI/UX AI-based assistant back-end we have chosen the RASA framework.
Rasa[9] is a leading AI platform that allows personalizing user experience through the building of virtual assistants (chatbots). Rasa provides infrastructure & tools necessary for building such chatbots and comes with three main components:

- Rasa Open Source: it is the machine learning framework for automated text and voice-based conversations. This is the core component and includes the Rasa server that handles the conversations. Rasa Server has functionality to understand messages, hold conversations, and connect to messaging channels and APIs. It can interact with many different applications such as Facebook messenger, Slack, Telegram, Twilio, or with any custom application via a REST channel.
- Rasa Action Server: it runs custom actions for a Rasa conversational assistant. Implementing custom actions gives the chatbot the ability to perform complex tasks such as transforming user input, querying external services via REST calls and returning any kind of payload back to user.
- Rasa X is a graphical tool that helps us develop chatbots able to handle as many story paths as possible. This is done using Conversation-Driven Development (CDD), the process of listening to actual users and using those insights to improve the AI assistant. Rasa X allows to share a trained assistant to many users-testers and collects the interactions so the developers get valuable feedback that helps to debug and improve the assistant.
-

[9] https://rasa.com/


Talk to your bot (Interactive Learning)

Conversation session started on 15 Jun 2021

action_listen

hello

greet (1.0)

utter_greet (1.0)

Hi there, I'm the Incognito AI assistant. How can I help you?

action_listen (1.0)

Story till now

```
version: "2.0"
stories:
- story: Story from Conversation ID 602f4d36c31f4bd89197b2c9972924f0
  steps:
  - intent: greet
  - action: utter_greet
```

Slots

```
No slots
```

Figure 6. Rasa conversational assistant

For the implementation of the frontend, a simple, dialog/chat-like interface is being developed with Kotlin[10]. We are considering continuing to use Kotlin for the rest of the AI-based assistant features that are compatible with Java. In terms of integration, the connection between the front end and the back-end and the connection to the IDC will be realized through REST APIs to ensure security and extensibility.

The details of the UI/UX AI-based assistant module, its design and technology can be found in the deliverable D5.1.

[10] https://developer.android.com/kotlin

4 The Integration Platform

This chapter gives an overview of the tools and technologies we are using in INCOGNITO, from collaborative tools to software integration and deployment tools.

4.1 Collaborative tools

In order to add traceability and visibility to the entire software development process, we will use dedicated tools to collaborate on task creation, source code management, and document sharing. Good organization and transparency are vital in INCOGNITO, due to the international nature of the development team and the continuously new arriving secondees who join the team.

4.1.1 Phabricator

During the software development process, we decided to use Phabricator to collaborate on tasks. Phabricator is an open-source platform that offers project management features, allowing team members to host Git repositories, share documents and other information like wiki pages or details about events.
Phabricator allows us to organize tasks in a hierarchical way, set deadlines for them, add commentaries and set priorities. Moreover, we are tracking progress in two ways: by using Kanban boards and by adding comments on the task specifying its status. Kanban is a software development method to manage work across systems. The main goal of Kanban boards is to provide a visual representation of the progress, helping the team to prioritize activities in a more efficient way. A classic Kanban board is usually divided in three columns: To Do, In Progress and Done like shown in Figure 7.

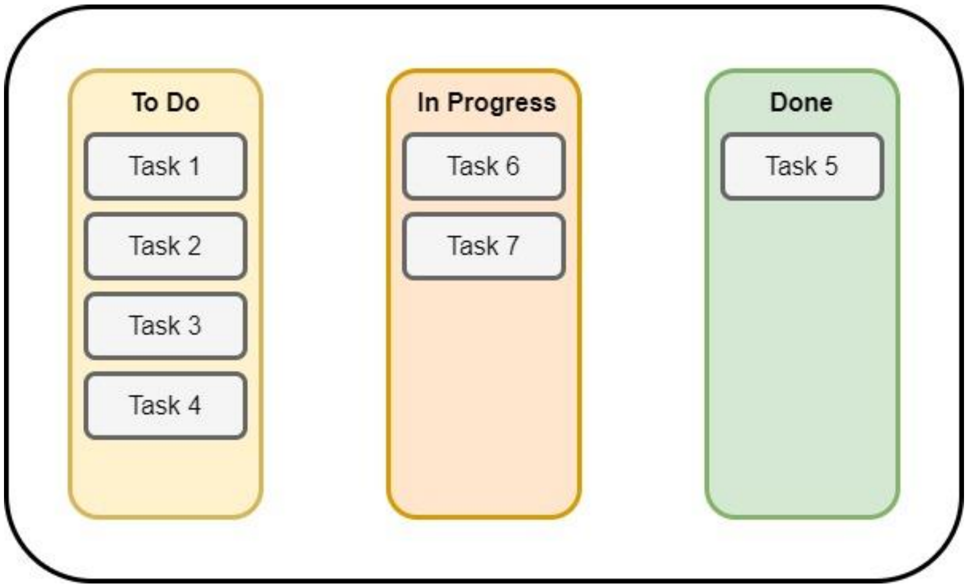


Figure 7. Example of organizing task on the board

4.1.2 GitLab

One of the critical components in a software development project is a version control software with a friendly User Interface. As multiple developers across different countries will contribute to the codebase, it’s essential to keep everything updated in a centralized repository. A platform like GitLab offers transparency and traceability, contributes to knowledge sharing and real-time collaboration.
In INCOGNITO we have multiple repositories for different modules of the application.

GitLab has key functionalities for uploading new code and updating the existing codebase that are easily accessible in the interface for all developers. Each repository has a main branch called Master. When working on a sub-task, the developer creates a new branch and applies and then publishes the changes on it. The next step is a request to add the code to the main branch which implies an operation called "creating a pull request". The pull request must be approved by the assigned developer/s. Once approved, the two branches are merged together and the newly added code can be found in the Master branch.

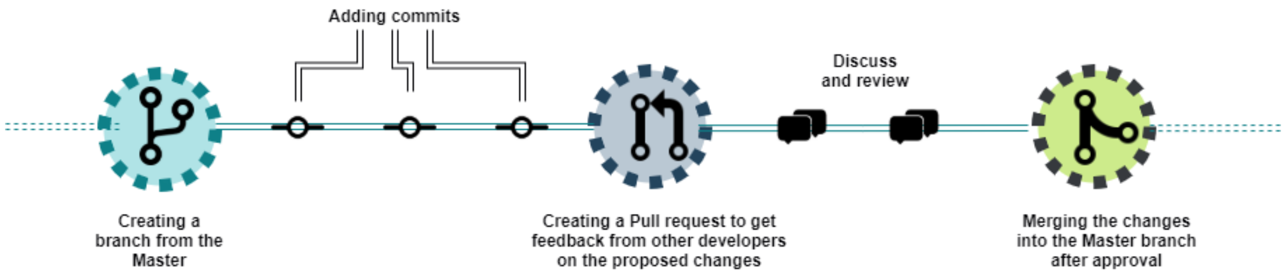

Adding commits
Discuss and review
Creating a branch from the Master
Creating a Pull request to get feedback from other developers on the proposed changes
Merging the changes into the Master branch after approval

Figure 8. Creating a new branch from the Master branch

4.1.3 Git

Git[11] is an open-source version control system for code management. It is designed to make and track local file changes and share them with a remote repository where everyone can access a stable version of the code.
With Git, the developer can create an environment for the local files on their computer called "repository". Within this repository, files can be added, modified and deleted while maintaining evidence of every change. This is an important feature because the developer can see a history of changes that have been made and revert them to a previous version if needed. Moreover, the developer can sync their local repository with a remote one (in our case GitLab) where the team keeps a centralized stable version of the code.

[11] Git, https://ro.wikipedia.org/wiki/Git

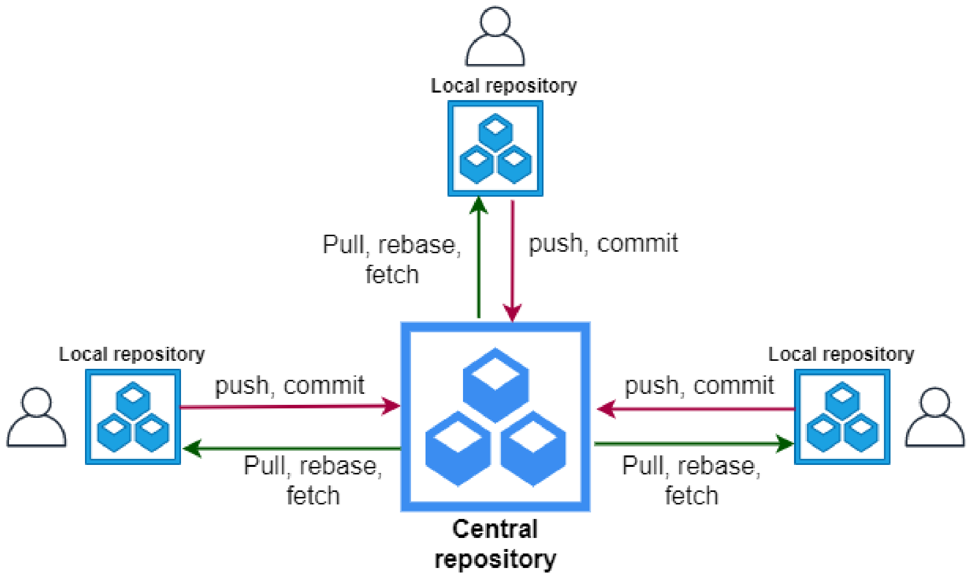


Figure 9. Git Local and Remote Repositories workflow

4.1.4 Office365 – One Drive

One of the significant parts of this project revolves around documents. From deliverables to reports and documents for knowledge sharing, the team of researchers needs a software to create and edit documents. We chose Office365 provided by Microsoft. This is a cloud-based solution where team members can easily collaborate on writing documents (Word, Excel, etc.), track changes and edit in real time, as long as they have an internet connection. These key functionalities of Office365 are valuable and helpful for our team of researchers based in multiple locations.

4.2 Continuous Integration and Continuous Deployment tools

Although INCOGNITO does not strictly follow the CI/CD philosophy because it does not use all the usual environments (development, testing, staging and production) it uses a replica of the production environment where every new functionality has to prove to be stable in order to be deployed. As a result, a workflow designed to integrate and deploy new features continuously is needed. There are a few notable tools that became a standard for every efficient CI/CD process over the years which will be addressed in this chapter.

4.2.1 Jenkins

Jenkins[12] is a service used to perform continuous integration and build automation by configuring it to execute a predefined list of steps.

[12] Jenkins, https://www.jenkins.io/

This open-source tool may help the team discover errors in the early stages of the development by continuously testing the builds.

Among the advantages of using Jenkins are:

- Developers don't have to synchronize the time of committing their code because it is tested and built every time a developer applies a new change. Therefore, new features can be proposed multiple times during a day.

- It is easier to detect whose change in the code generated any potential error since the code is built after each commit of a single developer.

- More time is saved, and new features are delivered faster as a result of the build and test process is automated and no longer manual

On our integration server, Jenkins runs on port 8081 and can be accessed at: http://incognito-dev.certsign.ro:8081

When a repository is ready on GitLab, it can be incorporated into a Continuous Integration pipeline. For that, a connection between the desired repository and Jenkins must be established. These are the steps to establish a connection between Jenkins and a repository hosted on GitLab:

- Generate an API Token from the GitLab interface with your user's credentials
- Add the API Token on Jenkins credentials section and test connection (It should return a Successful message)
- Generate a SSH key pair, add the public key on GitLab and the private key on Jenkins
- Create a project on Jenkins, providing the corresponding GitLab repository link, in order to trigger a build every time a change is pushed to GitLab

4.2.2 Docker

Docker[13] is a containerization platform that packages an application and all its dependencies together inside a container. This open-source tool is used to deploy applications inside software containers and solves the "works on my machine" problem when collaborating with a team of developers by allowing everyone to run the code in an identical environment. This issue is of interest and great importance, considering that the group of researchers in INCOGNITO is geographically distributed across Europe. Therefore, all have different computers and working stations.

A few notable Docker concepts are:

[13] Docker, https://www.docker.com/

DOCKERFILE - A Dockerfile is a text document that contains a series of commands to be run by the user in order to assemble an image

IMAGE – A Docker image is a template file that contains the source code, libraries, dependencies required to run an application.

CONTAINER – A container is a running image. They package applications with all the necessary dependencies and configurations. Containers are portable, which means they can be easily shared between developers. They are kept in a repository that can be private or public (Example: Docker Hub is the public repository for Docker).

DOCKER COMPOSE – Docker Compose[14] is used to deploy multi-container applications. The configuration is written in a Yaml file that contains details about the containers and services used to run the application

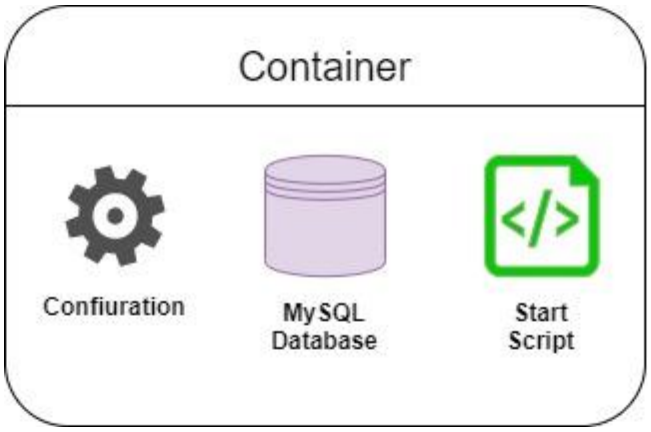


Figure 10. Docker Container components

A typical Docker workflow to create an image when having the application ready and running it as a container:

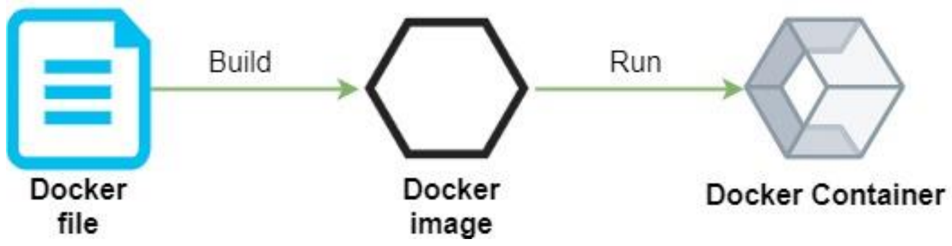


Figure 11. Docker workflow

[14] https://www.educative.io/blog/docker-compose-tutorial

1. Create the Dockerfile
2. Build the file:

   ```
   docker build <file_name>
   ```

3. Check the successful creation of the image from the previous step by listing all the available images:

   ```
   docker images
   ```

4. Create a container based on the image:

   ```
   docker run <image_id>
   ```

5. Check the created containers available:

   ```
   docker ps
   ```

4.2.3 Kubernetes

Kubernetes is a container management tool used to automate the deployment and management of containers. In INCOGNITO, Kubernetes is used to orchestrate the Docker containers in a lightweight manner. Kubernetes proves itself to be especially useful when many containers are constituting a modular application that requires to be connected, integrated, and updated. One of the major advantages of Kubernetes over Docker is the self-healing capability of the containers (deployed on Pods) in case of failure.

In a classic Kubernetes setup, there are multiple master and worker nodes on separate machines. Since INCOGNITO does not use multiple servers for deployment (only one integration server described in chapter 4.3), a Kubernetes implementation named Minikube[15] is used. Minikube is a one node cluster, where the master and worker processes are on the same machine.

There are a few key concepts in a Kubernetes architecture[16]:

POD – A pod is the smallest unit in Kubernetes. It is a group of one or more application containers (such as Docker) running a module of the application: the database, the web application etc. Pods run on the nodes and have their own IP Address. The IP Address of a pod is non-persistent.

[15] Minikube, https://minikube.sigs.k8s.io/docs/

[16] Kubernetes objects, https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/

DEPLOYMENT – A deployment is a set of multiple, identical pods that run multiple replicas of the application. A Deployment configuration declares the expected state of a Pod. It works in conjunction with Replica Sets to ensure that the expected number of instances of a given pod is available at any time in the cluster, including when scaling up or down and during updates. Deployment is a resource to deploy stateless applications.

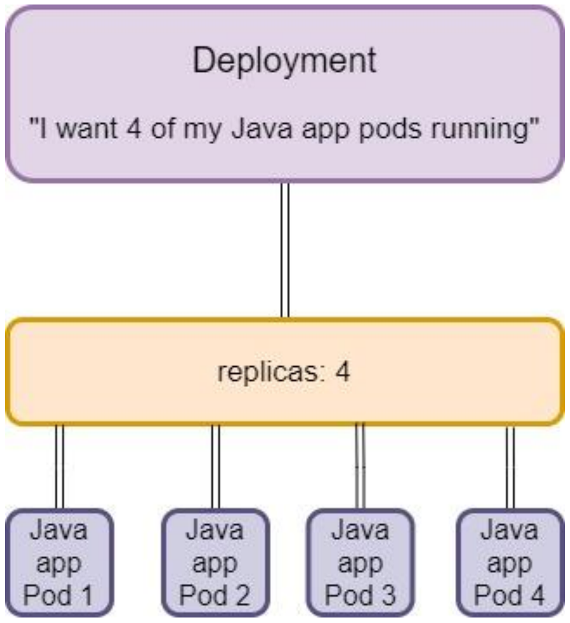


Figure 12. Kubernetes Deployment functionality

Deployments ensure that at least 75% of pods are UP while updating an image of a container. It does not kill the pods until enough new ones have been created.
Deployments can be scaled:
Example: `kubectl scale <name-of-deployment> --replicas = 10`

STATEFUL SET - StatefulSet is a resource used to manage stateful applications. Stateful applications are usually databases and apps that need to store data. In a StatefulSet configuration each replica of a pod will have its own state and will be using its own volume. This means that pods are not interchangeable, which is a key feature when working with data in order to avoid data inconsistency. It is mandatory to have a service declared in the configuration of the StatefulSet. The service creates the necessary endpoints to expose the pods with DNS names inside the cluster. StatefulSets can be scaled:
Example: `kubectl scale <name-of-statefulset> --replicas = 5`

REPLICA SET - A ReplicaSet is a resource that runs multiple instances of a Pod and assures that the desired number of pods are running at any time. When a ReplicaSet needs to create new pods, it uses the configuration of the pod that has been specified as a template. Compared to the Deployment resource, ReplicaSets don’t allow declarative updates to pods. According to the official documentation, depending on the scenario, Deployments are usually preferred since they operate on a higher-level[17].

SERVICE - A Service exposes a set of pods to other pods within the cluster, or to the outside world. It acts as a permanent IP Address that can be assigned to a pod, because the lifecycle of a pod and a service are not connected.

VOLUMES – A volume is a way to store data between pod restarts. The storage has to follow two general rules: it must be available on all nodes and it must survive even if the cluster crushes. Volumes in Kubernetes can be local (on the physical server) or remote (in the cloud).

Steps to deploy a module of the application with Minikube on the server:

Start the Minikube cluster with one node:

```
minikube start –driver=Docker
```

It is also the possible to start minikube with multiple nodes:

```
minikube start --nodes <number-of-nodes>
 Eg: minikube start --nodes 2
```

Check the status of minikube cluster:

```
minikube status
```

Create the deployment by specifying the Yaml file containing the configuration:

```
kubectl create -f <file-name.yaml>
```

Check the status of the Deployments or StatefulSets:

```
kubectl get deployments
Kubectl get statefulsets
```

[17] https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/

See details of the deployment:

```
    kubectl describe deployment <name-of-deployment>
```

The number of pods will be dictated by the 'replicas' field in the Yaml file. To see the pods and their status:

```
    kubectl get pods
```

For troubleshooting, see logs about each container in the deployment:

```
    kubectl logs <name-of-pod> <name-of-container>
```

The desired output after a successful deployment is presenting the status "running" when checking the state of a pod:

```
kubectl get pods

NAME                                READY   STATUS    RESTARTS
keycloak-mysql-69cdb8bbff-9tcxq     1/1     Running   0
keycloak-mysql-69cdb8bbff-bwpsm     1/1     Running   0
```

It is useful to mention that there are ways to migrate from a docker-compose configuration to a Kubernetes configuration. One of these tools is an open-source software called "kompose"[18]. This is a solution to help developers who are more familiar with Docker concepts to migrate to Kubernetes deployments. The tool is available on all operating systems and it translates the "Docker-compose.yaml" file to one or multiple Yaml files corresponding to the Kubernetes objects necessary for that specific application.

Compared to ReCRED (which uses only Docker to deploy the software modules), this new way of deploying the application with Kubernetes assures availability of the containers by the "self-healing" capacity of the Pods. If a component of the application deployed on a Pod goes down, Kubernetes will automatically re-deploy it. There are three states a Container can have in a deployment: Waiting, Running and Terminated. The Waiting state corresponds to when the container is created or the image is being pulled. The Running state indicates that the container inside that pod is working accordingly with no issues. The Terminated state is for containers that fail or have completed their execution. On the other hand, pods have phases based on which

18

Kubernetes will perform liveliness and readiness probes in order to "heal" them if necessary. These phases are:

- Pending – the pod was created but it is not running
- Running – the pods is running the container successfully
- Succeeded – the pod has successfully completed the container lifecycle
- Failed – minimum one container failed or all containers terminated
- Unknown

Based on these concepts, Kubernetes makes sure that the defined state of a cluster and the actual state are in-sync.

On this initial version of the integration platform, we have the following components deployed: Keycloak with FIDO2 and Rasa.

The pods in the Keycloak deployment can be listed and accessed by the following command:
`kubectl get pods`

There are 3 pods for the Mysql database component:

```
NAME                                READY   STATUS    RESTARTS   AGE

keycloak-mysql-0                    1/1     Running   0          8d
keycloak-mysql-1                    1/1     Running   0          8d
keycloak-mysql-2                    1/1     Running   0          8d
```

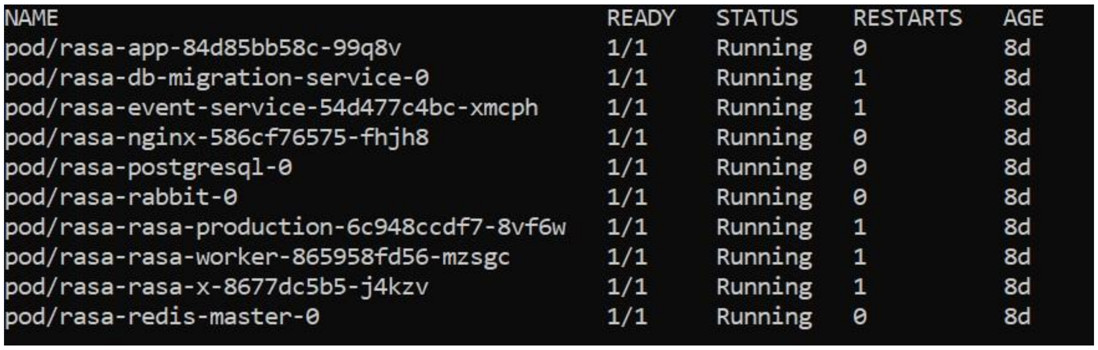
There are 2 pods for the Keycloak containers:

```
NAME                                READY   STATUS    RESTARTS   AGE
incognito-keycloak-646dd8f458-lwh5n 1/1     Running   0          9s
incognito-keycloak-646dd8f458-xbvh6 1/1     Running   0          8s
```

The MySQL component is deployed as a StatefulSet, therefore we have a headless service associated with it. The service can be listed with the command "`kubectl get services`" and it displays the following service:

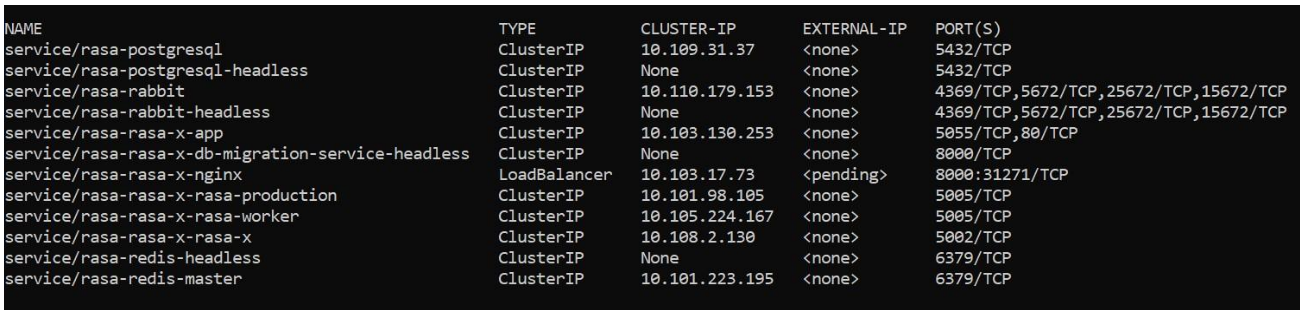
```
NAME             TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
keycloak-mysql   ClusterIP   None         <none>        3306/TCP   8d
```

All the components of the Rasa deployment are organized in a namespace called "rasa" and can be listed by "`kubectl get all –n rasa`" and the output presents all the resources shown in Figures 13, 14, 15, 16 and 17.



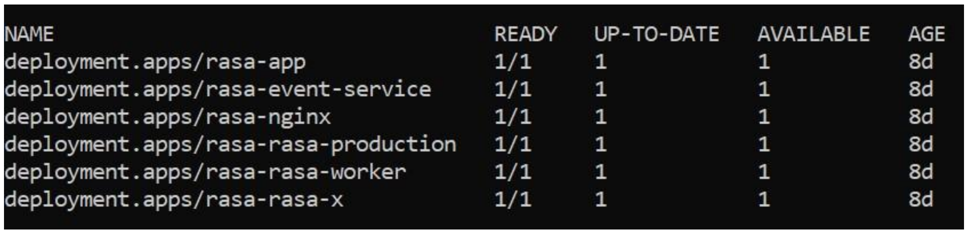
```
NAME                                      READY   STATUS    RESTARTS   AGE
pod/rasa-app-84d85bb58c-99q8v             1/1     Running   0          8d
pod/rasa-db-migration-service-0           1/1     Running   1          8d
pod/rasa-event-service-54d477c4bc-xmcph   1/1     Running   1          8d
pod/rasa-nginx-586cf76575-fhjh8           1/1     Running   0          8d
pod/rasa-postgresql-0                     1/1     Running   0          8d
pod/rasa-rabbit-0                         1/1     Running   0          8d
pod/rasa-rasa-production-6c948ccdf7-8vf6w 1/1     Running   1          8d
pod/rasa-rasa-worker-865958fd56-mzsgc     1/1     Running   1          8d
pod/rasa-rasa-x-8677dc5b5-j4kzv           1/1     Running   1          8d
pod/rasa-redis-master-0                   1/1     Running   0          8d
```

Figure 13. The pods in Rasa Deployment



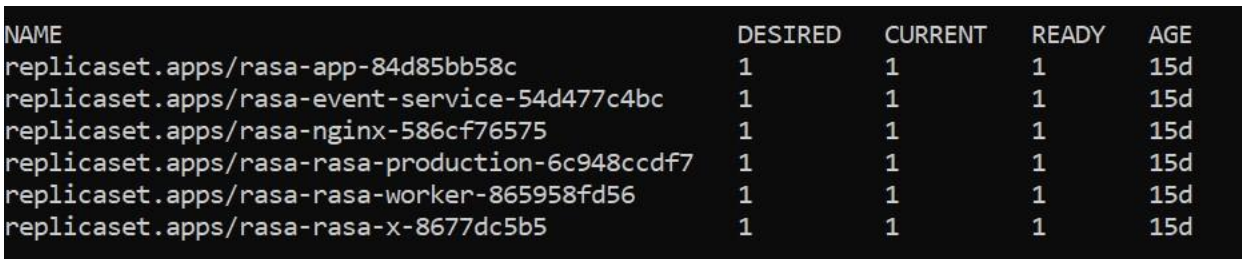
```
NAME                                               TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)
service/rasa-postgresql                            ClusterIP      10.109.31.37     <none>        5432/TCP
service/rasa-postgresql-headless                   ClusterIP      None             <none>        5432/TCP
service/rasa-rabbit                                ClusterIP      10.110.179.153   <none>        4369/TCP,5672/TCP,25672/TCP,15672/TCP
service/rasa-rabbit-headless                       ClusterIP      None             <none>        4369/TCP,5672/TCP,25672/TCP,15672/TCP
service/rasa-rasa-x-app                            ClusterIP      10.103.130.253   <none>        5055/TCP,80/TCP
service/rasa-rasa-x-db-migration-service-headless  ClusterIP      None             <none>        8000/TCP
service/rasa-rasa-x-nginx                          LoadBalancer   10.103.17.73     <pending>     8000:31271/TCP
service/rasa-rasa-x-rasa-production                ClusterIP      10.101.98.105    <none>        5005/TCP
service/rasa-rasa-x-rasa-worker                    ClusterIP      10.105.224.167   <none>        5005/TCP
service/rasa-rasa-x-rasa-x                         ClusterIP      10.108.2.130     <none>        5002/TCP
service/rasa-redis-headless                        ClusterIP      None             <none>        6379/TCP
service/rasa-redis-master                          ClusterIP      10.101.223.195   <none>        6379/TCP
```

Figure 14. The services in Rasa Deployment



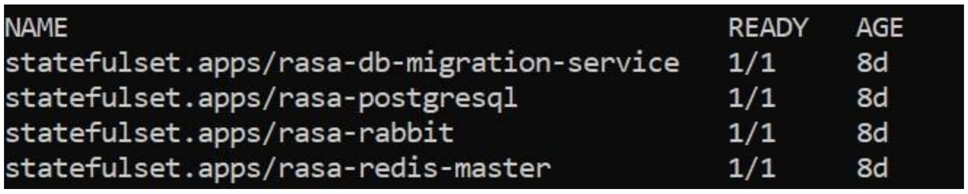
```
NAME                                 READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/rasa-app             1/1     1            1           8d
deployment.apps/rasa-event-service   1/1     1            1           8d
deployment.apps/rasa-nginx           1/1     1            1           8d
deployment.apps/rasa-rasa-production 1/1     1            1           8d
deployment.apps/rasa-rasa-worker     1/1     1            1           8d
deployment.apps/rasa-rasa-x          1/1     1            1           8d
```

Figure 15. The deployments in Rasa Deployment



```
NAME                                        DESIRED   CURRENT   READY   AGE
replicaset.apps/rasa-app-84d85bb58c         1         1         1       15d
replicaset.apps/rasa-event-service-54d477c4bc 1       1         1       15d
replicaset.apps/rasa-nginx-586cf76575       1         1         1       15d
replicaset.apps/rasa-rasa-production-6c948ccdf7 1     1         1       15d
replicaset.apps/rasa-rasa-worker-865958fd56 1         1         1       15d
replicaset.apps/rasa-rasa-x-8677dc5b5       1         1         1       15d
```

Figure 16. ReplicaSets in Rasa Deployment



```
NAME                                      READY   AGE
statefulset.apps/rasa-db-migration-service   1/1     8d
statefulset.apps/rasa-postgresql             1/1     8d
statefulset.apps/rasa-rabbit                 1/1     8d
statefulset.apps/rasa-redis-master           1/1     8d
```

Figure 17. The StatefulSets in Rasa Deployment

4.3 Hardware platform

Each partner has their own infrastructure (personal computers or servers) for developing software and testing different technologies. However, a common server was needed in order to integrate all components and sub-modules together.
The hardware component is where the application will run, be deployed and hosted. It is also the place where some of the CI/CD tools will run.

The server is based on Linux with a CentOS 8 distribution and it is owned by CertSIGN.

There are some minimum specifications for this server in order to set up a similar environment dedicated for deploying and hosting the application. The server has 8 CPUs, Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz and a total usable RAM memory of 16 GB.

Each developer in INCOGNITO has an individual account on the server with a corresponding username and they access it through SSH either by having a SSH key pair or by password.

On the integration server we have the following tools installed and configured:

Docker (version 20.10.6) – the software used to create, deploy and run applications by using containers described in Chapter 4.2.2
Jenkins (version 2.288) – the continuous integration and build automation tool described in Chapter 4.2.1
Minikube (version v1.18.1) – the one node cluster described in Chapter 4.2.3
Kubectl (version v1.20.5) - a command line tool used to interact with any type of Kubernetes cluster setup: Minikube or Cloud.

5 Conclusions

Deliverable D6.1 "Initial System Design and Prototyping" is the first deliverable of Work Package 6. In this deliverable we covered information about the software modules, the hardware infrastructure, the CI/CD tools and the software development methodology and practices we used in order to collaborate successfully as a team.

The work of D6.1 serves as a foundation for the next deliverable, D6.2 "Final Integrated System". The integration process is continuous as more software modules will be ready for deployment, but the initial set up of the platform together with all the tools installed and configured on it represent the core for any future activity in Work Package 6.

6 References

[1] S. Sharma, D. Sarkar, D. Gupta , “Agile Methodology and Principles: A conceptual study”, available at: https://www.researchgate.net/publication/267706023_Agile_Processes_and_Methodologies_A_Conceptual_Study

[2] E. Yuan and J. Tong, “Attributed based access control (ABAC) for Web services,” in *IEEE International Conference on Web Services (ICWS’05)*, Jul. 2005, p. 569, doi: 10.1109/ICWS.2005.25.