intellect EU

DAML-on-Fabric - Technical document

June, 2020

Table of Contents

1. Introduction

This document describes the current architecture of the DAML-on-Fabric connector as well as the technical status of the system according to Digital Asset's plans on this project.

DAML-on-Fabric allows you to build your DAML smart-contracts and deploy them to a local DAML Ledger using Hyperledger Fabric network as the infrastructure layer.

Hyperledger Fabric is an open sourced blockchain technology, under the guidance of the Linux Foundation. The way this blockchain technology was designed, allowing a lot of flexibility and configurability, where it is possible to easily scale up or down the infrastructure resources.

To trial the solution it is both possible to launch a local network or an already existing Hyperledger Fabric network to connect DAML-on-Fabric. To help onboard users to the first option, a set of scripts have been provided to bootstrap an Hyperledger Fabric network locally. These execute tasks relating to crypto config, constructing a channel between two parties, and deploy a default chaincode through multiple organizations (by default 2 organizations).

Then, as the DAML-on-Fabric connector is itself bootstrapped and connected to a network, it is possible to test and run DAML applications where all business logic, workflows and data is processed through the Ledger API and stored under the Hyperledger Fabric network.

2. Requirements Specification

In this section, the main stakeholders foreseen the system are listed together with the requirements gathered to guide this implementation.

2.1. Stakeholders

For the purpose of this system the following stakeholders have been considered and included as relevant for the solution specified by this document:

- **DevOps/Developer** – Technical subject matter responsible for the operational and management support of the DAML-on-Ledger instance;
- **DAML Party** – Any entity that will interact with DAML's Ledger API or Navigator to read, create, exercise and archive ledger states/contracts. This stakeholder is foreseen to use the system through an application.

2.2. Functional Requirements

In this section, the main functional requirements specified for the system are described. To simplify the analysis the features have been merged into functional groups. Below, it is presented the macro-requirements for the system.

Code	Functional group
R1	Ledger Bootstrapper
R2	Ledger DAML Server
R3	Ledger API Authorization

Along the next subsections, the functional requirements for each module are listed. To describe each functionality the following columns are used:

- **ID**: identifies the functional requirement univocally;
- **Name**: name of the functional requirement;
- **Stakeholders**: lists the stakeholders that can access the functional requirement;

2.2.1. Ledger Bootstrapper

This module is composed of features for DAML ledger configuration and any other setup services required under the scope of this implementation.

The current project allows the user to bootstrap a Hyperledger Fabric network by using the Ledger Bootstrapper. To better understand the main actions executed by the Ledger bootstrapper, a brief summary of the main components and concepts of Hyperledger Fabric are presented below.

- **Peer**: is a network component that runs the chanicode and stores the data (blocks) and maintains the network consistency.
- **Orderer**: an orderer is the network component that orders the blocks and sends them out to all the peers, guaranteeing that the block order is the same for everyone. A network requires the existence of at least one orderer.

- **Certificate Authority:** to participate in the blockchain network both users and nodes require digital identities. To provide such identities one or multiple trusted certificate authorities (CAs) may be used. In this project it was not necessary to use a Certificate Authority. Instead of using this component these identities have been generated using the configtxgen tool. These identities take the form of cryptographically validated digital certificates compliant with the X.509 standard.

The main concepts of the platform used in this project were:

- **Chaincode**: the chaincode is an application that implements the smart-contracts of the project. It is flexible and can be adapted to a great variety of topics. You can develop chaincode in three different programming languages, Golang, Java, and NodeJS. The chaincode runs in each peer in the scope of a channel.
  The Daml-on-Fabric connector depends on a chaincode developed in Golang and located in the folder chaincode. This chaincode defines the rules that append or query states to or from the ledger.
- **Ledger**: the Ledger is an immutable, auditable, and secure key-value storage that persists the data needed to implement the business logic developed in DAML.
- **Channel:** a channel connects multiple parties and allows data segregation between participants of a network. Each channel has multiple chaincodes and ledgers that only are accessible to participants. Fabric supports private channels (data collections)l inside one channel if there is any requirement to not show all data of an organization - DAML-on-Fabric does not make use of this feature.
  A channel can have a policy.
- **Channel Policy**: The channel policy is a set or rules that implements the transaction agreements agreed by the participants (e.g., all organizations should validate the transaction or at least one participant in the channel should validate the transaction). In this project, by default, the policy configured states that any solo member within the channel can accept a transaction for it to be committed.

Regarding the users, Fabric supports multiple users, meaning different identities. The default implementation of DAML-on-Fabric uses only one Fabric user per organization, that is then used by the multiple DAML parties. Every call to Fabric is done through the user defined in the configuration file.

ID	NAME	STAKEHOLDERS
R1.1	Launch Orderer Service	Devops/Developer
R1.2	Launch Peers	Devops/Developer
R1.3	Construct a Channel	Devops/Developer
R1.4	Add orderer to channel	Devops/Developer
R1.5	Add peers to channel	Devops/Developer
R1.6	Install chaincode on channel	Devops/Developer
R1.7	Query chaincode	DAML Party and Devops/Developer
R1.8	Invoke chaincode	DAML Party and Devops/Developer

A description of the functionalities is available below:

- **Launch Orderer Service**: launches an orderer service using the script restart-fabrict.sh described in the docker-compose.yaml file.
- **Launch Peers:** launches a peer per organization using the script fabric-restart.sh described in the docker-compose.yaml file, where it can be defined for the peer to use a LevelDB or CouchDB amongst other configurations. By default the peers are configured to use a LevelDB.
- **Construct a channel:** creates a channel in the network between two organizations. It uses the name included in the config-local.yaml file.
- **Add orderer to a channel:** adds the created orderer to the channel. This operation is executed when the user operates the default network. When connecting to an existing network, this operation may not execute if the orderer(s) are in the channel.
- **Add peers to a channel:** adds the created peers to the default channel. This operation is executed when the user operates the default network. When connecting to an existing network, this operation may not execute if the peers are in the channel.
- **Install chaincode on a channel:** installs the default chaincode to the channel. his operation is executed when the user operates the default network. When connecting to an existing network, this operation is held before the connector starts the communication.
- **Query chaincode:** operation that retrieves data (transactions) from the ledger.
- **Invoke chaincode:** operation that inserts data (transactions) into the ledger through a chaincode function.

Except the two last requirements (query and invoke chaincode), all others can be executed by a DevOps/Developer in a cloud solution. It means that the set up of the network is not handled by this project. To connect to an existing network, the user should retrieve the right certificates, create a new config file based on the “config-default.yaml” file, and fill in the attributes correctly.

2.2.2. Ledger DAML Server

This module is composed of the operations that interact with the chaincode. These operations are sent by the Ledger DAML server to execute data retrieval or data input.

ID	NAME	STAKEHOLDERS
R2.1	Batch write into ledger	DAML Party
R2.2	Read from ledger	DAML Party
R2.3	Write the commit log	DAML Party
R2.4	Read the commit log	DAML Party
R2.5	Read the commit height	DAML Party
R2.6	Read package	DAML Party
R2.7	Write package	DAML Party
R2.8	Read ledger ID	DAML Party
R2.9	Write ledger ID	DAML Party

Below, each requirement is briefly described:

- **Write into ledger (RawBatchWrite):** issues a set of new states into the ledger of the shared channel. The submission of a create command via the DAML Ledger API invokes the "RawBatchWrite" method on the chaincode.
- **Read from ledger (RawRead):** retrieves a state from the ledger of the shared channel. The submission of a get command via the DAML Ledger API invokes the "RawRead" method on the chaincode.
- **Write the commit log (WriteCommitLog):** writes the latest state change after the submission of a create command.
- **Read the commit log (ReadCommitLog):** reads the current state of the ledger. This command synchronizes the DAML client with the current state of the ledger.
- **Read the commit height (ReadCommitHeight):** retrieves the height of blocks in the ledger. This command synchronizes the DAML client with the current state based on the height of blocks.
- **Read package (PackageRead):** retrieves a DAML package from the ledger into the shared channel. This command triggers the reading of the packages in the channel.
- **Write package (PackageWrite):** writes a package into the ledger of the shared channel. It stores the DAML package submitted through DAML upload-dar command.
- **Read ledger ID (LedgerIDRead):** retrieves the identifier of the ledger to let DAML clients synchronize data available in the ledger of the shared channel..
- **Write ledger ID (LedgerIDWrite):** creates a new state in the shared ledger containing the identifier of the ledger. This operation executes during project startup if no ledgerID is found.

2.2.3. Ledger API Authorization

In this module, the authorization methods implemented during the scope of this project are presented.

ID	NAME	STAKEHOLDERS
R3.1	JWT-RS256	DevOps
R3.2	JWT-ES256	DevOps
R3.3	JWT-ES512	DevOps
R3.4	JWT-RS256-JWKS	DevOps
R3.5	JWT-HS256-UNSAFE	DevOps

The Ledger API implements the following authorization mechanisms to allow secure the connections between any client and the Ledger API. For more information on how authorization works with the Ledger API please visit DAML Authentication page[1]. The JWT algorithms supported by the connector are:

- --auth-jwt-rs256-crt - JWT auth token signed with RS256 (RSA Signature with SHA-256);
- --auth-jwt-es256-crt - JWT auth token signed with ES256 (ECDSA using P-256 and SHA-256);
- --auth-jwt-es512-crt - JWT auth token signed with ES512 (ECDSA using P-521 and SHA-512);
- --auth-jwt-rs256-jwks - JWT token to be signed with RS256 (RSA Signature with SHA-256) with the public key loaded from the given JWKS URL;
- --auth-jwt-hs256-unsafe - JWT token to be signed with HMAC256 with the given plaintext secret.

[1] https://docs.daml.com/app-dev/authentication.html

2.3. Non-Functional Requirements

The non-functional requirements specify the set of criteria used to judge the quality of the solution. These state the improvements expected under the scope of this project. Due to the short time frame of the project, it has been decided to focus on the following non-functional requirements based on the business and quality requirements identified by Digital Asset.

2.3.1. Performance/Efficiency Requirements

The performance/efficiency requirements focused on how the application uses the resources allocated to it to achieve the intended results.

2.3.1.1. Transaction Sizing Capacity

In the scope of the project, this solution shall support at a bare minimum transaction with the sizing of 100 kb. However, a larger capacity will be pursued for transactions of up to 1 Mb.

Performance Definitions	Minimum Level	Intended Level
Supports transactions of size <= X [kb]	100	1000

During the project we concluded that the size of transactions did not have impact in the performance. We were able to execute transactions with 25 MB size without noticing any impact in the performance.

This result led the team to reach the Alpha stage of the product.

2.3.1.2. Transaction Hit Rate

Regarding the capacity for the system to handle a certain amount of transaction requests the hit rate levels must be 5 transactions requests per second or higher. The higher mark to be sought on this requirement is at 20 transactions per second.

Performance Definitions	Minimum Level	Intended Level
Supports at least X ping transactions per second [tx/s]	5	20

To test this requirement the team used a tool provided by DA (ledger-api-test-tool.jar) and a DAML package (PerformanceTests.dar). These tools check the number of transactions that the network can achieve in a specific timeframe.

To use this test package, we must first upload the package using the command and run test suite.

$ *daml ledger upload-dar PerformanceTests.dar --host localhost --port 6865*

$ *java -jar ledger-api-test-tool.jar --perf-tests=PerformanceEnvelope.Alpha.Throughput localhost:6865 --timeout-scale-factor=4*

2.3.1.3. Transaction Throughput

As for the throughput of the system the basis for the system is to guarantee that over 90% of transactions execute in under 3 seconds, but seek to reach throughputs under a second.

Performance Definitions	Minimum Level	Intended Level
90% quantile of above ping transactions commits in time <= X [s]	3	1

To test this requirement the team used a tool provided by DA (ledger-api-test-tool.jar) and a DAML package (PerformanceTests.dar). These tools check the number of transactions that the network can achieve in a specific timeframe.

To use this test package, please run the commands described in the previous section.

2.3.2. Reliability requirements

These are the reliability requirements that were achieved within this project.

2.3.2.1. Recoverability

In the event of an interruption or a failure, the application must recover the data affected directly and potentially re-establish the system without support intervention. For automation of this process it is important for a health check service to be created as mentioned in the previous requirement.

ID	Interruption/Failure Type	Recover Action
NFR1.1	Ledger API fails or stops	Retrieve unprocessed blocks for execution of Block Listener tasks
NFR1.2	Ledger Client fails or stops	Reconnecting to the ledger node

When the Ledger API fails and disconnects itself from the network, there are still blocks or data that will be added to the network while the Ledger API is offline. When the Ledger API restarts, it checks the height of the blocks in the ledger. If it has a lower height, then it synchronizes its data. If the ledger client unexpectedly stops only the DAML parties dependent on that client will not be able to add or see any data. The stored data will still be maintained in the network and no desynchronization occurs, since no data is stored on the client side.

2.3.2.2. Maturity

The maturity level for the application should go from the current alpha stage, where features are not complete and serious errors may still occur. Under the current status any resulting instability may cause crashes or data loss. Along the path towards the beta stage, although it is expected for the application to still contain a number of known or unknown bugs, the applicable issues of an alpha release must be solved.

2.3.3. Security Requirements

Regarding security, the project focused on addressing authentication features and ensuring a secure connection between connector and Fabric under TLS/SSL. By default client authentication is not enforced, however it can be configured by accessing official documentation of Hyperledger Fabric. The configuration file allows you to activate client authentication and required certificates (client key and certificate).

2.3.4. Usability Requirements

The usability requirements focused on easiness of configuring and bootstrapping the application.

2.3.4.1. Learnability

As part of the requirements for this system clear and concise guidelines shall be provided on how to achieve a number of user stories by the stakeholders with focus on the following:

ID	Name	Stakeholders
NFR2.1	Deploy a fully operational DAML-on-Ledger instance	Devops
NFR2.2	How to run a DAML application on top of the previously mentioned instance	Devops
NFR2.3	Upgrade an existing DAML-on-Ledger instance	Devops

2.3.4.2. Operability

The system should be operated as much as possible in compliance with the instanced DAML Ledger own operational functionalities.

ID	Name	Stakeholders
NFR3.1	Network configuration through configuration files	Devops
NFR3.2	Upgrade an existing DAML-on-Ledger instance	Devops

2.3.4.3. User error protection

Provide clear error handling and protection mechanisms for the multiple functionalities delivered. Design informative errors and warnings while enlightening the stakeholder on what has been wrongly executed.

2.3.5. Maintainability Requirements

These requirements set the stage for how the system shall be maintained and other constraints that will guide the architectural design to achieve these goals.

2.3.5.1. Modularity

It is vital for the application to be built in such a way where the impacts of changing a component will not severely affect other components with which it interacts.

2.3.5.2. Reusability

It should be taken into account how different components may be reused for different purposes and make them easily decoupled to achieve the goal of building different assets when expanding the system capabilities.

2.3.5.3. Analyzability

The system should enable activities happening within the different existing components to be easily audited, traced and debugged.

2.3.5.4. Modifiability

The developed application should take advantage of design patterns.

2.3.5.5. Testability

The application will be coupled with a battery of tests leveraging the Ledger API test tool both to test functional and non-functional parameters of the overall system put in place.

2.3.6. Portability Requirements

These requirements focus on the ease to run the application in multiple environments.

2.3.6.1. Adaptability

The current system is designed in a way that multiple DAML APIs can be run simultaneously and that they can run against an existing network or to a newly created or default local network..

2.3.6.2. Installability

Continuous deployment and integration was created and configured. This ensures the code quality and that it continues to work after any change.

2.3.6. Documentation requirements

The documentation requirements for this project focused on good code documentation of all functions and the description of technical configuration in a README file.

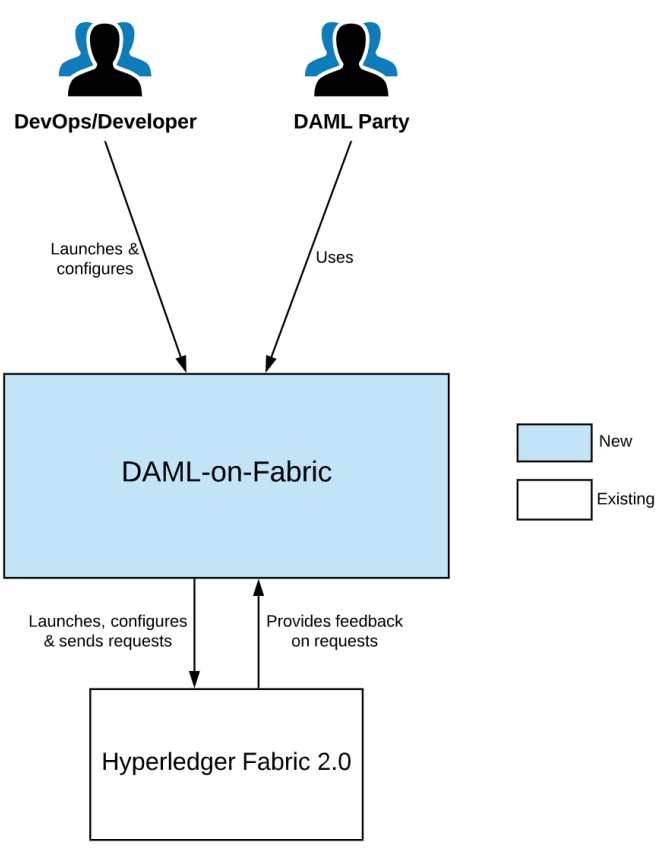
This document is part of documentation requirements.

3. Architecture

The following pages present a detailed architecture of the solution. It describes the understanding of the main participants, components and modules of the solution.
The next sections make use of the C4 model for visualising software architecture[2] approach to present the architecture of the system.

3.1. System scope and context view

The system and scope view represents an helicopter view of the system and it aims to describe the stakeholders and external systems that interact with the connector. The next diagram provides that vision.



First and foremost the following major stakeholders have been identified:

- DevOps/Developer User: is responsible to create, maintain and update the network and all its components as well as deploy DAML code, configure and launch the DAML-on-Fabric connector to interact with the network. The DAML party will interact with this project using DAML related tools, namely, the DAML Navigator

[2] https://c4model.com/

- DAML party: interacts with the system using Navigator or any Backend operation that communicates with Ledger API.

This project is ready to integrate with an Hyperledger Fabric network after configuration. It is possible to run it against two type of networks:

1. Bootstrap and configure a default local network: this project has the capability to create and run a Hyperledger Fabric network. It will bootstrap the network, configure the channel, join the orderer and peers to that channel and install the DAML-on-Fabric chaincode.
2. Connect to an existing network: the project allows the connection to an existing network running on cloud. The user only needs to configure correctly the attributes available in the config-local.yaml file. This running process assumes that the network and its components are up and running as well as channel configuration and chaincode installation.

3.2. System containers view

The system containers view present the runnable applications part of the project. The DAML-on-Fabric runs a docker container in a straightforward and cross-platform manner.

The main containers of the solution are described in the following sections.

3.2.1. Navigator & Application

These two containers act as clients to the DAML-on-Fabric system. The Navigator is an application provided by Digital Asset that provides a simple way to interact with DAML contracts. The Application service represents any Backend system that acts as a generic client that accesses the Ledger API.

3.2.2. DAML-on-Fabric

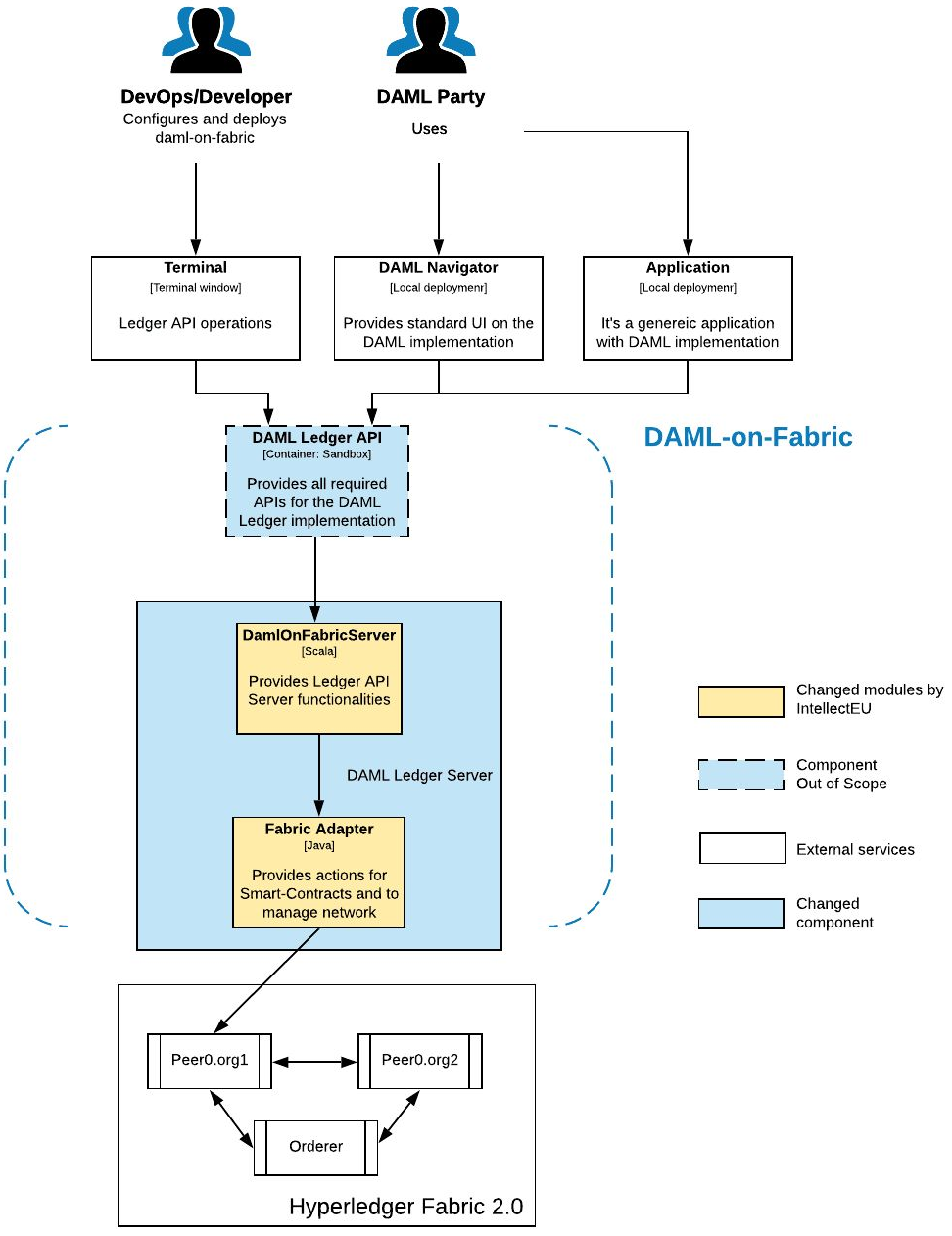
This container is the core application target of this project. It is able to establish a channel between two existing organizations, add peers and orderer to it and install the chaincode. It also has the ability to only initialize the channel objects if the network already exists. The connector delivers the requirements described in section 2.

3.2.3. Hyperledger Fabric network

The containers from Hyperledger Fabric have different purposes. The peers are responsible for storing the data and query & submit transactions. The orderer container is responsible for ordering the blocks and sends them to all the peers of that channel for storage.

3.3. System components view

The system components view aims to zoom in and decompose each container to identify the major structural building blocks of the system. The current DAML-on-Fabric is composed of a few components. They all act in conjunction to provide a simple usage of the system.



In the next subsections describe the main building blocks of the DAML-on-Fabric connector.

3.3.1. DAML Ledger API

This component is responsible for providing an API interface to DAML applications. This component interfaces with the DAML Ledger Server to retrieve or send data from the ledger. The API access is also controlled by this component.

3.3.2. DAML Ledger Server

The DAML Ledger Server is a component which can be overall divided into two major components: the Fabric server and the Fabric adapter.

3.3.2.1. DamlOnFabricServer

This component acts as a buffer for all incoming requests. It is here that the Ledger API request is parsed, prepared and put in a queue to be sent to Hyperledger Fabric SDK, namely, FabricContext.

3.3.2.2. Fabric Adapter

This component acts as a client for the Hyperledger Fabric network. This component uses Fabric's Java SDK instance to execute the business logic to interact with the network (e.g., channel creation, chaincode installation and chaincode querying & invocation).
This component is also configurable through the configuration file.

4. Conclusions

The developed solution has made it possible to deploy and run DAML applications on top of a 2.0 Hyperledger Fabric network.
The efforts to bring the current solution to an Alpha stage have been established successfully based on a requirements perspective (functional and non-functional). However, it is clear that further space for improvement exists in order to take this solution to a production readiness state.

4.1. Results

As the major goal for the project was to deliver the current DAML-on-Fabric solution to an Alpha fidelity stage, this goal was achieved successfully.
Regarding the functional requirements, the team were able to do a comprehensive refactor to the project while keeping the previously identified features working accordingly. On the other hand, from the non-functional requirements the team was able to achieve the following results:

- Performance requirements:
  - Transaction Sizing Capacity: the size of the transactions has not had any impact in the performance. The size of transactions have been increased to 25 MB without any significant impact on the performance.
  - Transaction Hit Rate section: a throughput of 7.5 transactions per second has been reached.
  - Transaction Throughput: the time taken by the system sits on the 20 ms for queries and 150 ms for invoke operations.
- Security
  - Ledger API Authorization was added to support the mechanisms identified by Digital Asset.
  - TLS communication between Fabric SDK client and Fabric components was added to the project.
- Portability requirements:
  - Adaptability: the configuration file has become readable and accepts multiple types of network topologies, while also centralizing the application configuration.
  - Installability: a few issues with paths and requests to, from. containers were detected and solved, when configuring the GitLab CI environment.

4.2. Recommendations

As previously mentioned DAML-on-Fabric has achieved the Alpha Fidelity levels. Given that, more work is required to reach the Beta level and production-grade levels. A few recommendations on what could be improved in addition to existing roadmap follows:

- Code optimization regarding the modularity of the application. Analysis of the separation of concerns between Hyperledger Fabric network launch responsibility and configuration/integration logic of the DAML server.
- Add support multi-threading, enabling execution of multiple requests at the same time. The aim is to allow a better parallelization of write operations into the ledger.
- Research incorporation of Fabric Certificate Authority: multiple fabric users to multiple DAML parties and using Fabric CA to store all the necessary certificates.