BalticLSC Software Architectural Vision

Initial considerations regarding structure and dynamics of the BalticLSC Software

Version 1.00

Priority 1: Innovation

Warsaw University of Technology, Poland
RISE Research Institutes of Sweden AB, Sweden
Institute of Mathematics and Computer Science, University of Latvia, Latvia
EurA AG, Germany
Municipality of Vejle, Denmark
Lithuanian Innovation Center, Lithuania
Machine Technology Center Turku Ltd., Finland
Tartu Science Park Foundation, Estonia

BalticLSC Software Architectural Vision

Initial considerations regarding structure and dynamics of the BalticLSC Software

Work package	WP5
Task id	A5.1
Document number	O5.1
Document type	Vision Document
Title	BalticLSC Software Architectural Vision
Subtitle	Initial considerations regarding structure and dynamics of the BalticLSC Software
Author(s)	Kamil Rybiński, Michał Śmiałek, Agris Šostaks, Henri Hanson, Radosław Roszczyk, Krzysztof Marek, Sergejs Rikačovs
Reviewer(s)	Daniel Olsson, Magnus Nilsson-Mäki
Accepting	Michał Śmiałek
Version	1.00
Status	**Final version**

Interreg Baltic Sea Region
EUROPEAN UNION
EUROPEAN REGIONAL DEVELOPMENT FUND

History of changes

Date	Ver.	Author(s)	Change description
01.06.2019	0.01a	Krzysztof Marek (WUT)	Setup of the document structure
11.06.2019	0.01b	Michał Śmiałek, Kamil Rybiński (WUT)	Input regarding functional requirements, conceptual model and component model.
19.06.2019	0.02	Agris Šostaks (IMCS)	Input regarding high-level architectural vision. Input to Introduction.
21.06.2019	0.03	Henri Hanson (TSP)	Input to Executive Summary and Introduction
21.06.2019	0.04	Agris Šostaks, Sergejs Rikačovs (IMCS)	Input to vision of technologies used.

Executive summary

The overall aim for the Baltic LSC activities is developing and providing a platform for truly affordable and easy to access LSC Environment for end-users to significantly increase capacities to create new innovative data-intense and computation-intense products and services by a vast array of smaller actors in the region.

Current Software Architectural Vision Document is an important output of the Baltic LSC project, based on technical workshops held during Q1-Q2 2019 with participation of external experts and led mainly by the project's technology partners from:

- Warsaw University of Technology (WUT)
- RISE Research Institutes of Sweden AB (RISE)
- IMCS University of Latvia (IMCS)

The Software Architectural Vision Document includes also input obtained during Baltic LSC project's international and local workshops as well as individual meetings with experts, potential cooperation partners and end-users from March – June 2019.

It is supporting the Environment Vision Document (Baltic LSC Output 3.1), which is providing an overall vision of the Baltic LSC Environment and is complemented by the Baltic LSC Platform Architectural Vision (Baltic LSC Output 4.1) describing in detail the hardware visions for the platform.

Table of contents

List of figures

1. Introduction

1.1 Objectives and Scope

To describe the vision of BalticLSC Software in relation to BalticLSC Environment Vision and BalticLSC Platform Vision. The main objectives: describe the main concepts, components and software development technologies of the system.

Document contains vision for functional requirements (as UML use-case model), vocabulary (conceptual model as UML class diagrams) architecture (as UML component model), and technologies.

1.2 Relations to Other Documents

The current Software Architectural Vision Document is supporting the Environment Vision (Baltic LSC Output 3.1), which is providing an overall vision of the Baltic LSC Environment and is complemented by Baltic LSC Platform Architectural Vision (Baltic LSC Output 4.1) describing in detail the hardware visions for the platform.

1.3 Intended Audience and Usage Guidelines

This Vision Document is the Output 5.1 as described in the Baltic LSC Application Form and intend-ed for internal use within BalticLSC consortium as basis for future software development activities as well as for reporting purposes for local First Level Control and Baltic Sea Region Program.

1.4 Notation Convention

Simplified versions of UML diagrams (Use-case, Class, and Component). Free-form diagrams.

2. Functional Requirements

2.1 Actors

End-user

A person performing calculations using the BalticLSC Network.

Developer

A person able to create applications for the BalticLSC Network.

Supplier

A representative of a company that supplies computation power to the BalticLSC Network.

Administrator

A person supervising BalticLSC Network.

Computation Resource

An external machine with the BalticLSC Client that performs computations.

2.2 Computation Application Management

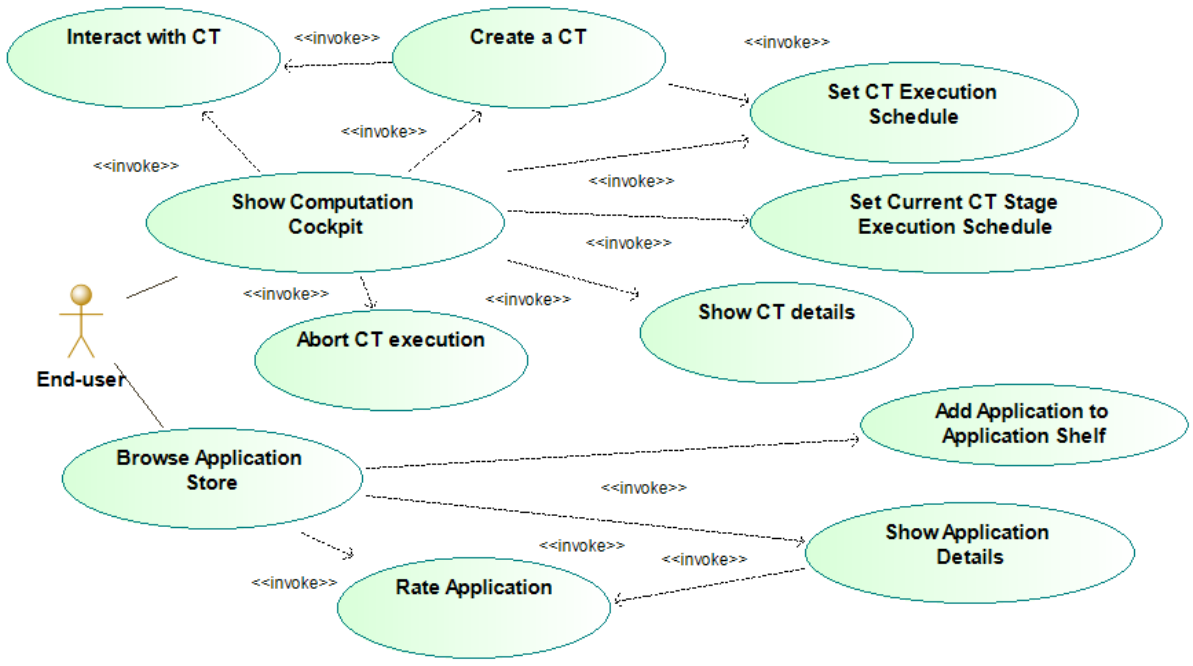

Figure 1. Use Case Diagram for Computation Application Management

Name	Show Computation Cockpit
Description	Show all the Computation Applications that the user can run – retrieved from the app store or created by him/her. Also shows all the Computation Tasks created by the user based on those applications.

Name	Create a CT

Description	Prepare a new Computation Task using a selected Computation Application. Includes specifying the Computation Valuation.

Name	Interact with CT
Description	Performs required interactions with a selected Computation Task. Available only for the Computation Tasks that needs such interaction.

Name	Set CT Execution Schedule
Description	Allows to define in advance results of interactions with the Computation Application and occurrences of pauses during its execution.

Name	Set Current CT Stage Execution Schedule
Description	Allows resume execution of the paused Computation Task or postpone that resumption to the certain point in time.

Name	Show CT details
Description	Show details of a Computation Task including its valuation and configuration. If task execution was started, shows also the task status and statuses of its subtasks.

Name	Abort CT Execution
Description	Stops execution of a running Computation Task.

Name	Browse App Store
Description	Allows for browsing of verified Computation Applications.

Name	Retrieve App from App Store
Description	Add a selected Computation Application to the user's App Repository, which make it available in the Computation Cockpit.

Name	Show App Details
Description	Show details about a Computation Application.

Name	Rate App
Description	Allows the user to review a selected Computation Application.

2.3 Computation Application Development

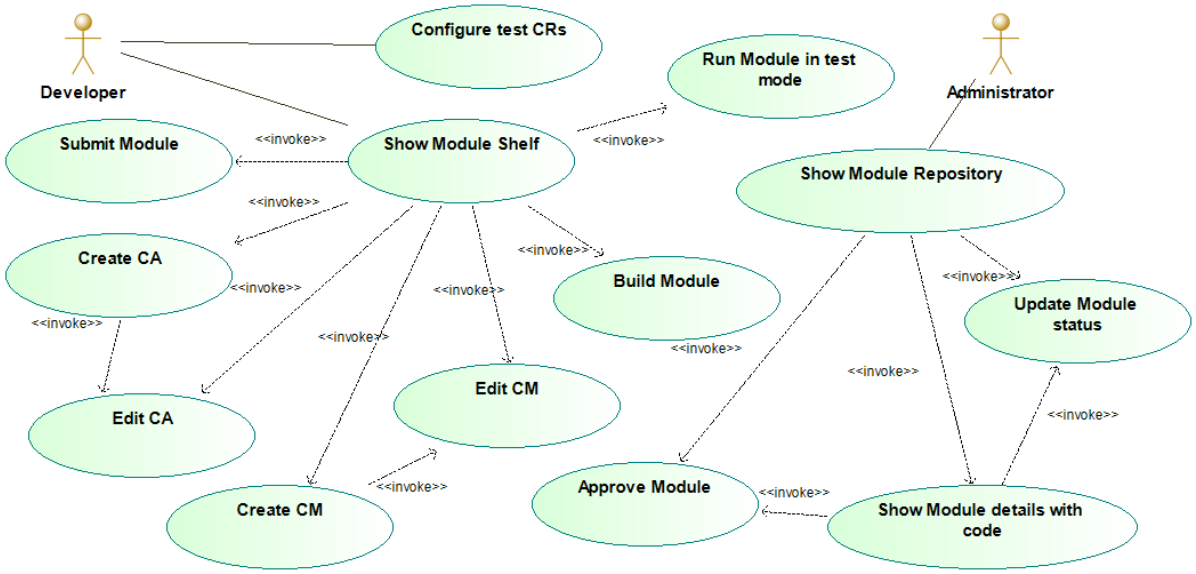

Figure 2. Use Case Diagram for Computation Application Development

Name	Show Module Shelf
Description	Allows to browse all the modules created by the current user.

Name	Submit Module
Description	Allows submitting a module to be verified and made available to other users.

Name	Create CA
Description	Create a new Computation Application.

Name	Edit CA
Description	Specify content of an Computation Application using a editor of CAL language.

Name	Create CM
Description	Create a new Computation Module.

Name	Edit CM
Description	Specify content of a Computation Module.

Name	Build Module
Description	Prepare the executable form of a module.

Name	Configure test CRs
Description	Configure connection to local Computation Resources that can be used for testing purposes.

Name	Run Module in test mode
Description	Allows for testing of created Modules, either on test CRs, or in restricted mode on the regular BalticLSC Network.

Name	Show Module Repository
Description	Allows to browse all the modules that have been ever submitted by the user.

Name	Approve Module
Description	Mark a module as available to all the users. Contains procedures related to module verification.

Name	Show module details with code
Description	Show advanced details about a module, including its implementation.

Name	Update Module Status
Description	Allows for status change of the already approved module.

2.4 Computation Resource Management

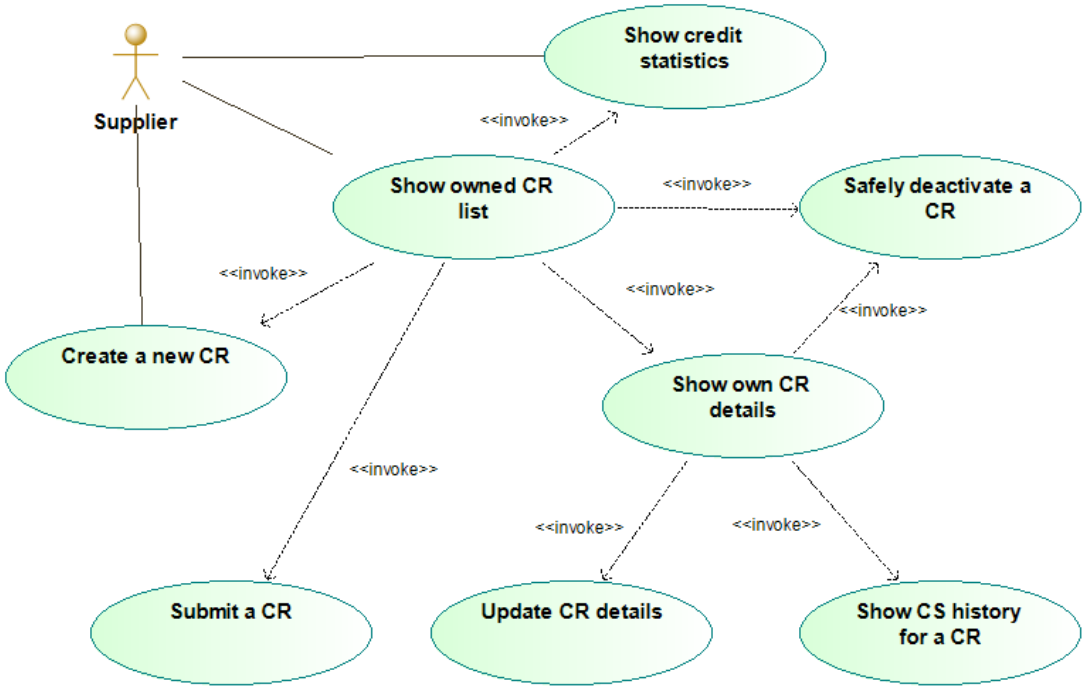

Figure 3. Use Case Diagram for Computation Resource Management

Name	Create a new CR
Description	Define a new Computation Resource.

Name	Show owned CR list
Description	Show all Computation Resources defined by the current user.

Name	Submit a CR
Description	Submit a Computation Resource for verification, after which it will be made available for performing computations of other users.

Name	Show credit statistics
Description	Show statistic about credits earned by the user's Computation Resources.

Name	Show own CR details
Description	Show details about a selected Computation Resource.

Name	Show CS history for a CR
Description	Show information about all Computation Steps that have been ever executed on selected Computation Resources. Information includes earned credits, execution time, execution results.

Name	Safely deactivate a CR
Description	Stop receiving new Computation Tasks by a Computation Resource and deactivate it after all already assigned Computation Tasks are completed.

Name	Update CR details
Description	Allows for modifying information that describes a selected Computation Resource.

2.5 Computation Resource Supervision

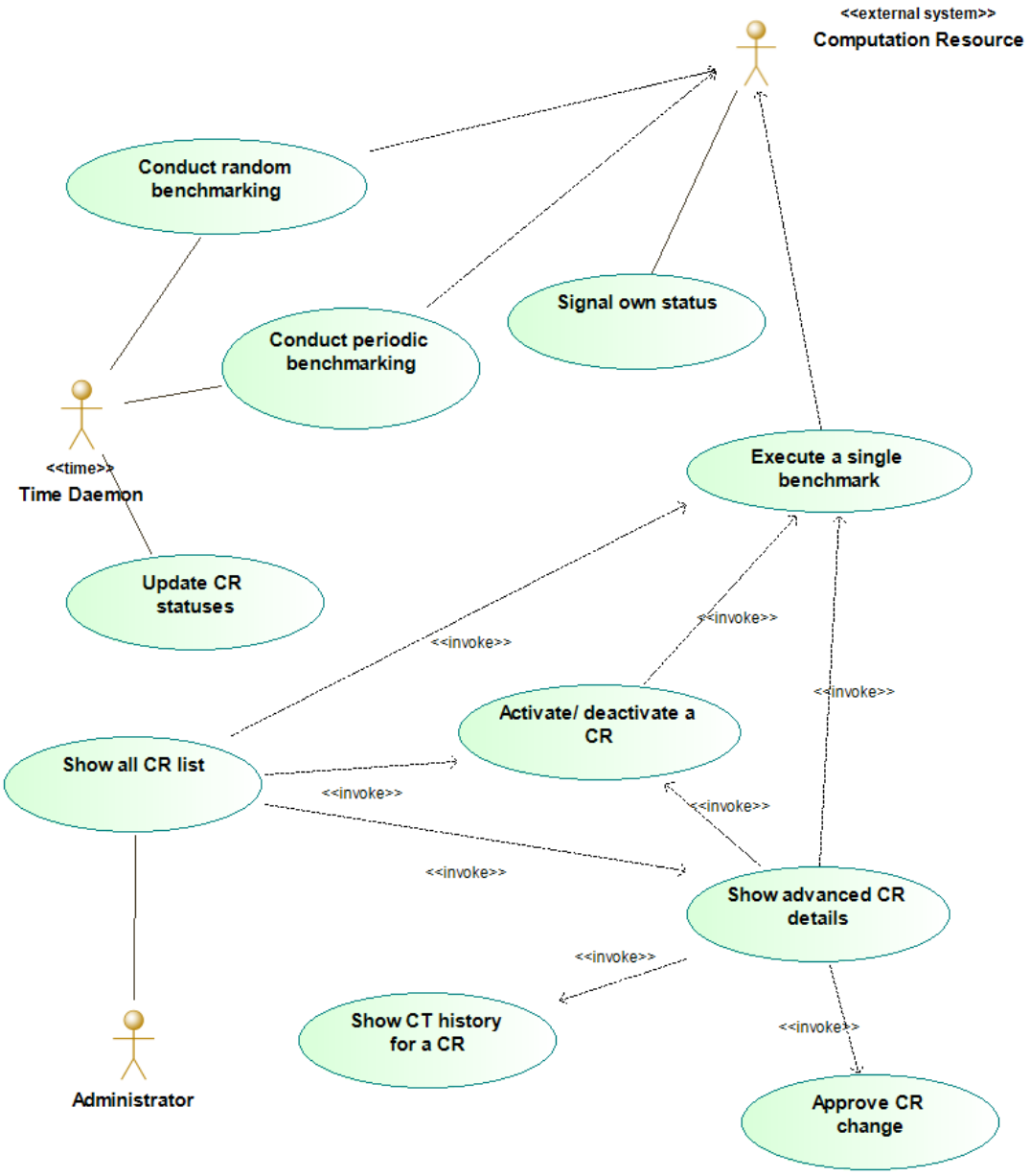

Figure 4. Use Case Diagram for Computation Resource Supervision

Name	Show All CR list
Description	Shows a list of all submitted Computation Resources.

Name	Show advanced CR details
Description	Show details about a Computation Resource, including benchmark results.

Name	Activate / deactivate a CR
Description	Allows to make a selected Computation Resource available or unavailable for performing computations of users other than its owner.

Name	Approve CR change
Description	Accept changes to information about a selected Computation Resource that require approval.

Name	Execute single benchmark
Description	Choose and run a benchmark on a selected Computation Resource.

Name	Conduct random benchmarking
Description	Run benchmarks on randomly selected set of Computation Resources.

Name	Conduct periodic benchmarking
Description	Periodically run benchmarks on all Computation Resources.

Name	Signal own status
Description	Allows a Computation Resource to inform the BalticLSC Network of its status and send information about progress of its assigned Computation Tasks.

Name	Update CR statuses
Description	Mark Computation Resources that didn't inform about its statuses as inactive.

2.6 Computation Application Requests

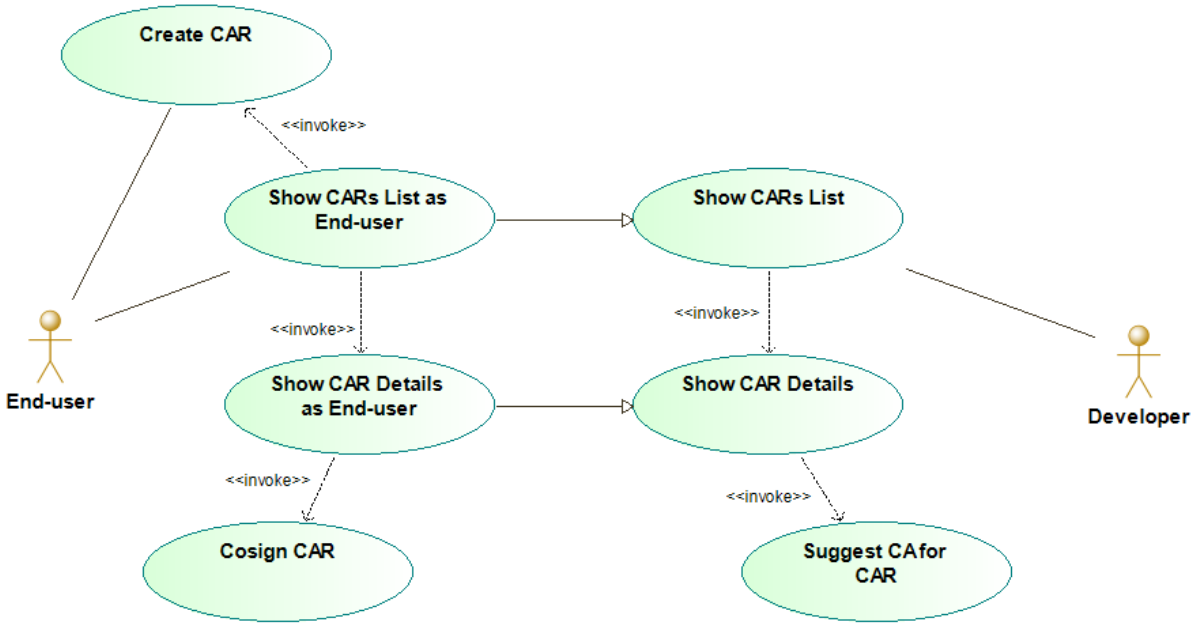

Figure 5. Use Case Diagram for Computation Application Requests

Name	Create CAR
Description	Allows to create a Computation Application Request specifying details of a Computation Application user needs.

Name	Show CARs List/Show CARs List as End-user
Description	Show a list of all created Computation Application Requests.

Name	Show CARs Details/Show CARs Details as End-user
Description	Show the definition of a selected Computation Application Request.

Name	Cosign CAR
Description	Mark the current user as being interested in a Computation Application Request created by another user.

Name	Suggest CA for CAR
Description	Allows to assign applications as fulfilling a selected Computation Application Request.

3. Domain Vocabulary

3.1 Computation Tasks

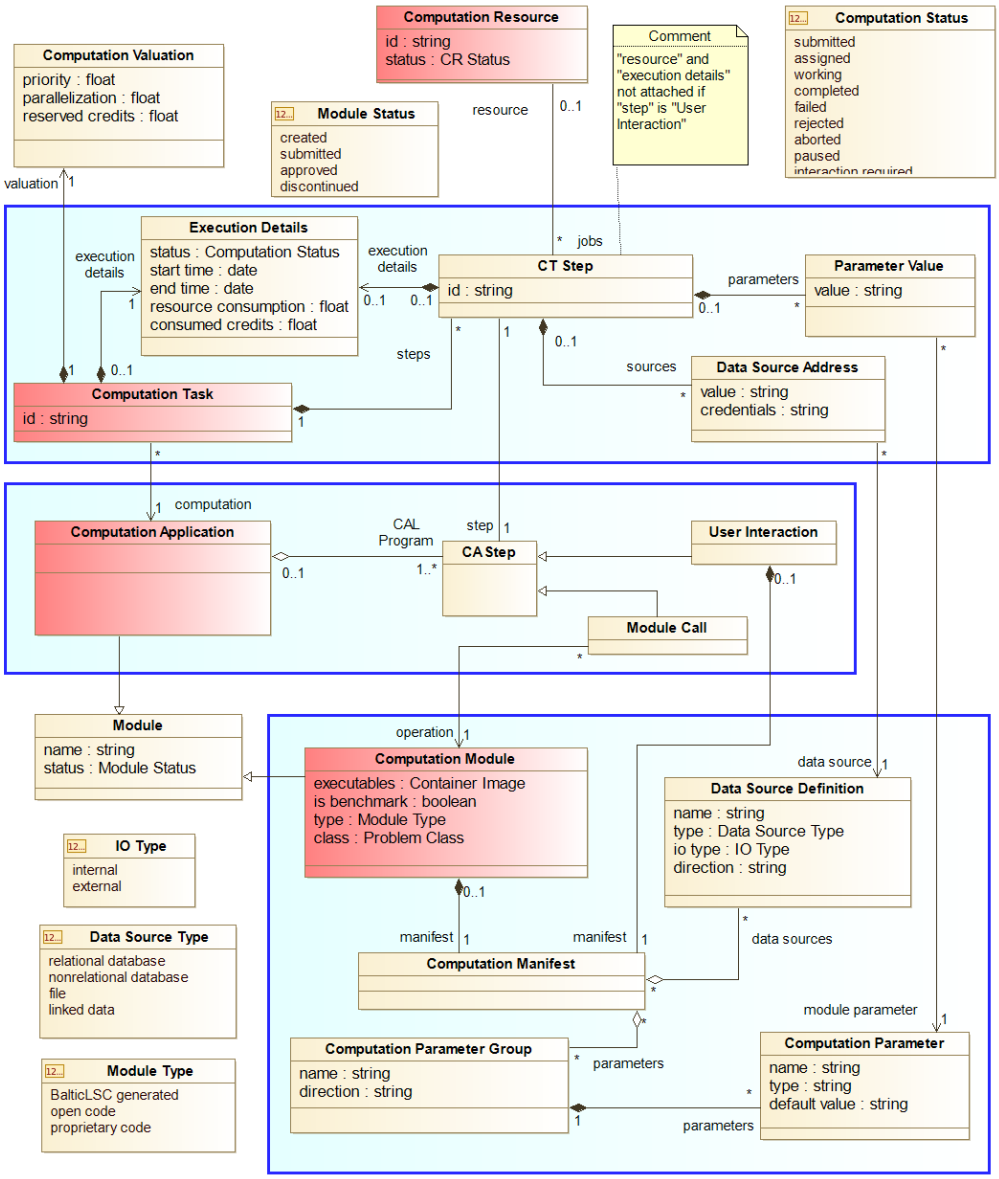

Figure 6. Class Diagram for Computation Tasks

Computation Module

Definition of computation part that can be run independently by enclosing it in separate container.

Computation Application

Program in CAL language that can be run by the user. Define control flow and data flow between given computation modules.

Computation Task

A computation application with valuation regarding its execution. Contains parameters that describe that execution.

Computation Manifest

Defines parameters and data sources required for module execution.

CA Step

Single instruction of CAL program. Indicates either execution of given computation module or occurrence of user interaction needed to provide parameters values and data sources addresses.

CT Step

Execution of single Computation Application Step on given Computation Resource. Store used parameters values and data sources addresses. Contains parameters that describe its execution.

3.2 Computation Resources

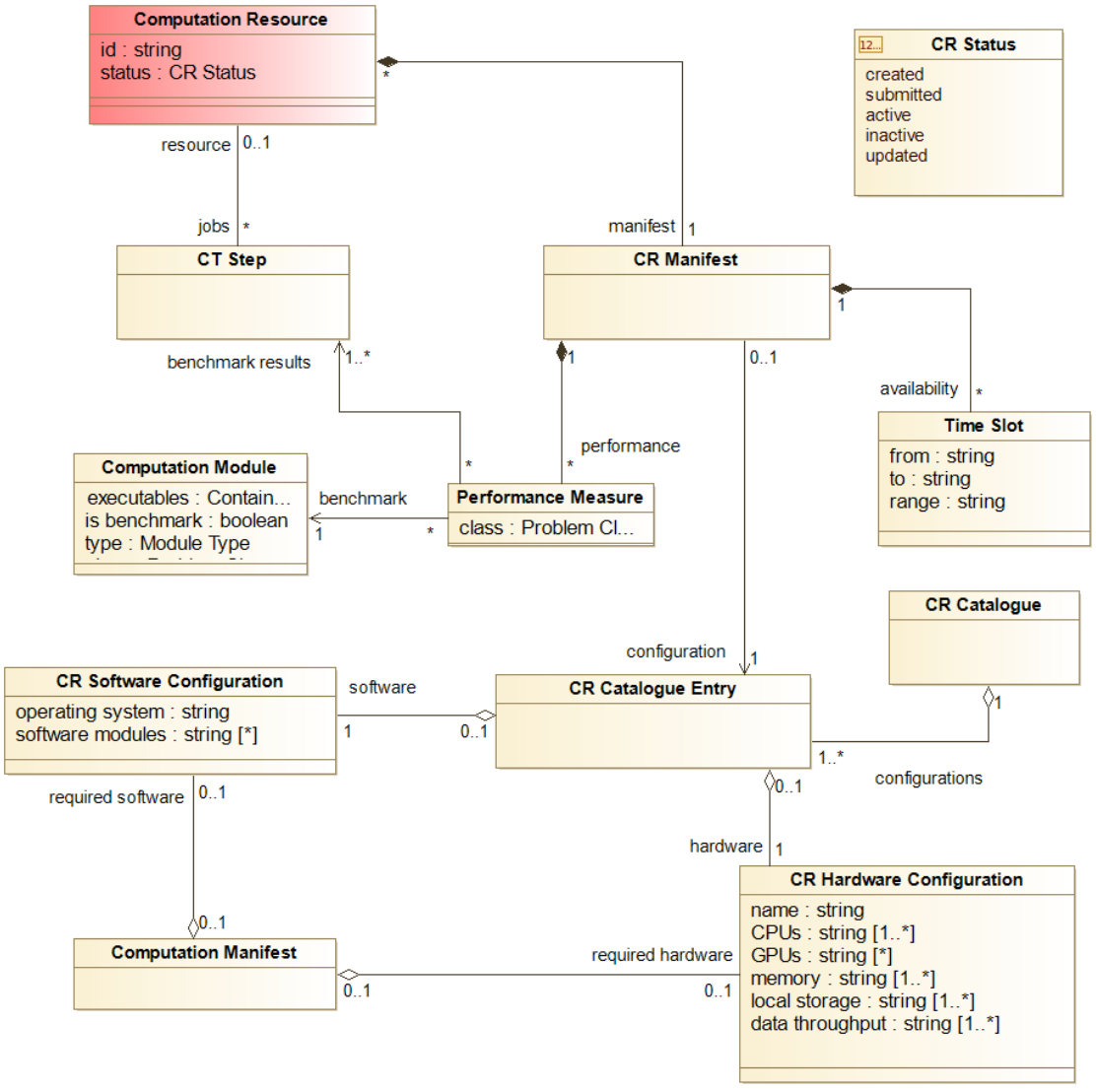

Figure 7. Class Diagram for Computation Resources

Computation Resource

Representation of an external machine with the BalticLSC Client installed on it that enables execution of Computation Tasks.

CR Manifest

Description of a Computation Resource that defines its software and hardware configuration, availability and performance.

CR Catalogue Entry

A catalogue entry describing one of the software-hardware configurations that is admissible for use in the BalticLSC Network.

Performance Measure

A summary of performance measure for a given type of tasks based on the results obtained by using a specific Application Module as a benchmark.

Problem Class

Classification of Computation Modules allowing to match them to proper Performance Measures.

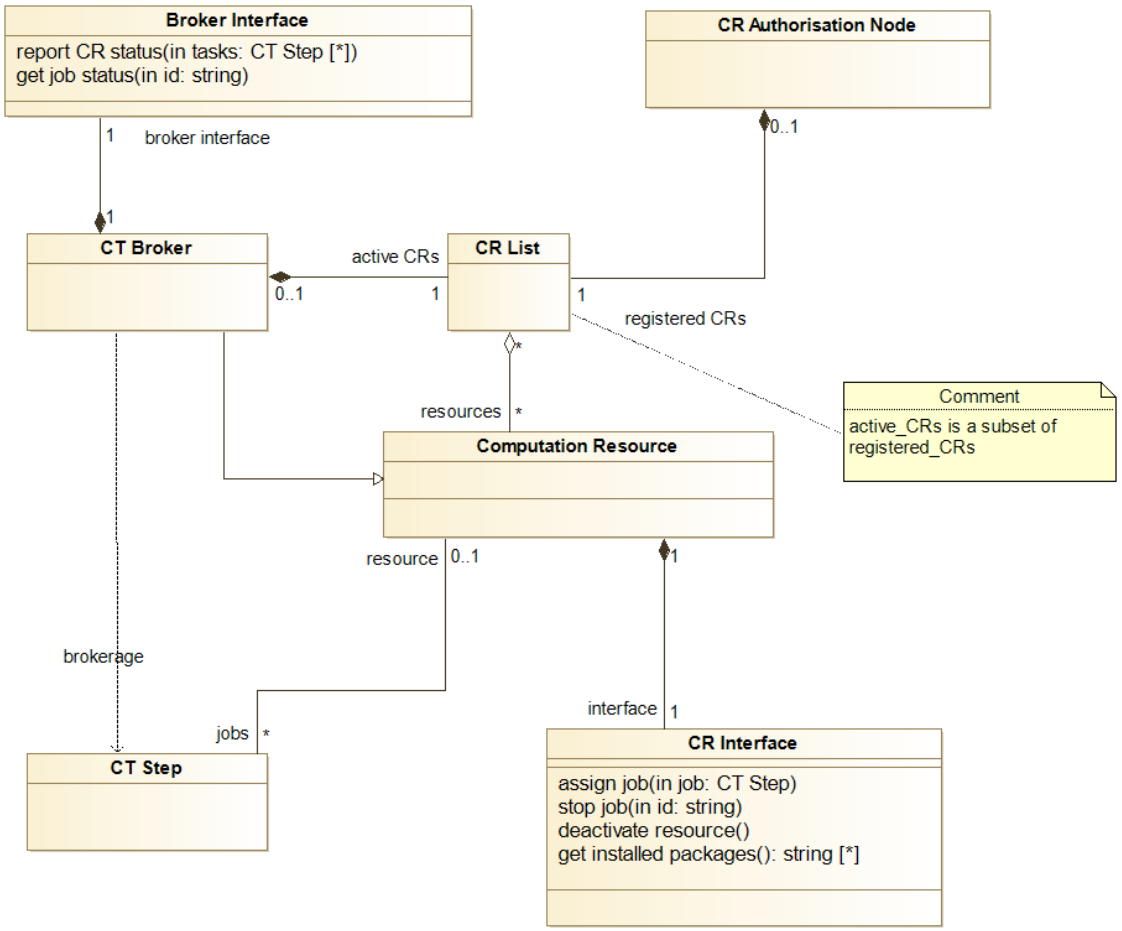

Figure 8. Class Diagram for Computation Brokerage

CT Broker

A network node responsible for distributing Computation Task Steps to Computation Resources. Contains a list of Computation Resources that signal it their status in a defined period of time. It also contains a list of statuses of the tasks assigned to these Computation Resources.

CR Authorization Node

A network node that stores a list of all Computation Resources certified to connect to the BalticLSC Network.

3.3 Computation Network

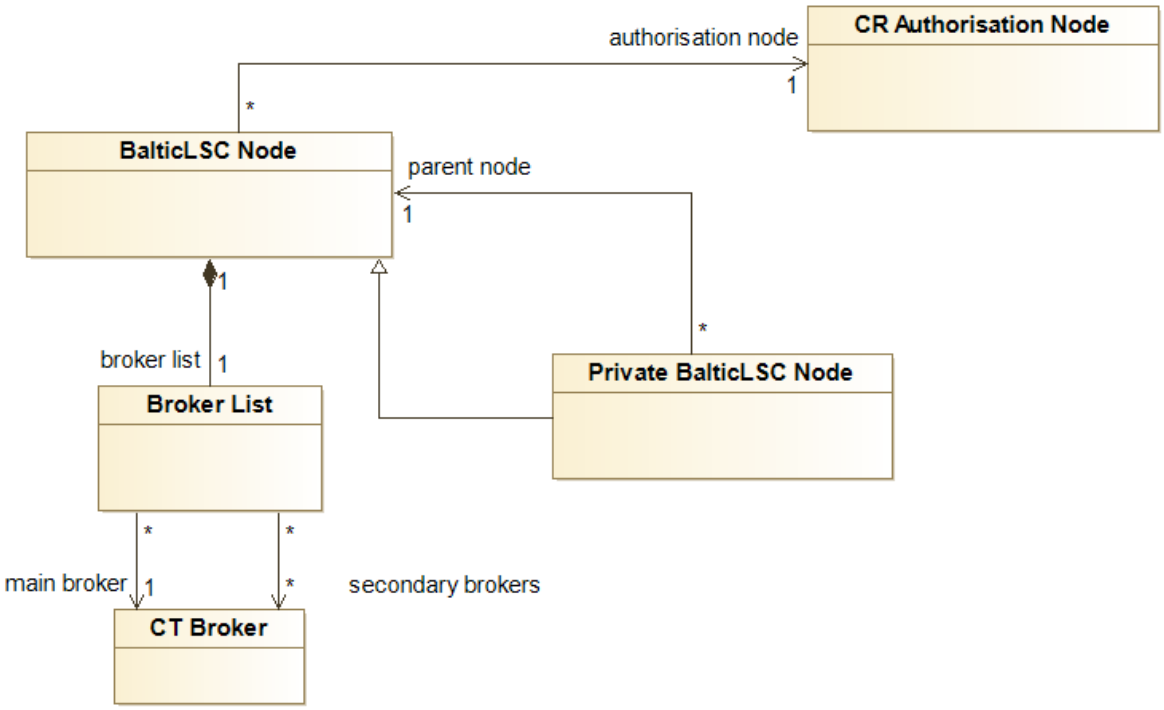

Figure 9. Class Diagram for Computation Network

BalticLSC Node

A network node containing the front-end for the BalticLSC Network. Contains a list of CT Brokers with a distinguished main broker, to which it directs Computation Tasks.

Private BalticLSC Node

A private front-end for a private part of the BalticLSC Network. It uses mainly its own Computational Resources. Contains a reference to a regular BalticLSC Node, in case it needs additional Computational Resources.

3.4 Application Store

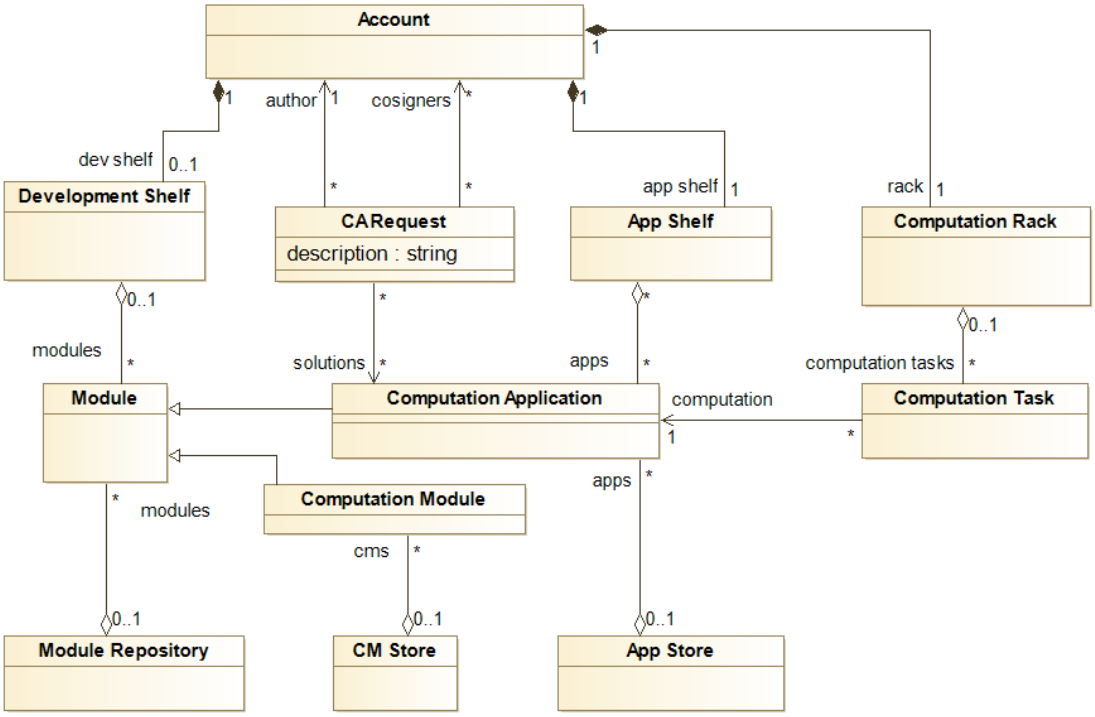

Figure 10. Class Diagram for Application Store

Module Shelf

A repository of all the modules created by the given user.

App Shelf

A local repository containing Computation Applications created by the user or retrieved by him from the App Store.

Module Repository

A central (but possibly distributed) repository containing all the modules that have been submitted by the users.

App Store

A central (but possibly distributed) store that contains verified Computation Applications that can be retrieved by the users to their App Repositories.

Module Store

A central (but possibly distributed) store that contains verified Computation Modules that can be used by the users to write their Computation Applications.

Computation Rack

A list of Computation Tasks created by a given user based on Computation Applications.

CA Request

A request describing some new Computational Application that is needed by some user. Can be cosigned by some users other than its author, who also need the described application.

3.5 CAL Programs

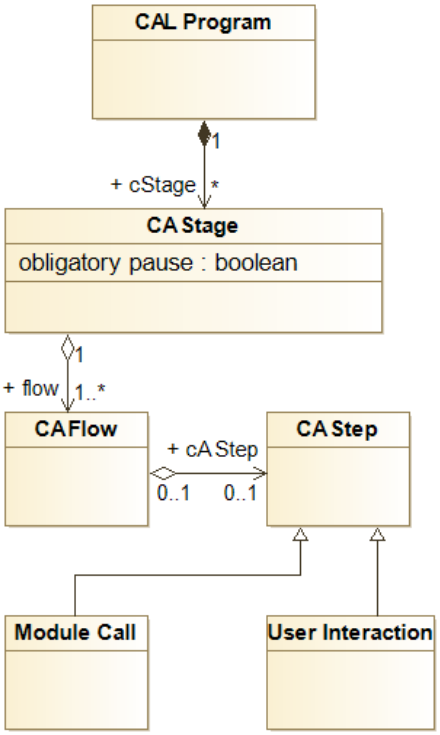

Figure 11. Class Diagram for CAL Programs

CA Stage

Set of CA Steps after which synchronization occur and global pause may occur.

CA Flow

One of CA Steps sequences that can be executed in parallel with others.

4. Architecture

BalticLSC Network is governed by one or more Master Nodes (see Figure 12 BalticLSC Network - High-Level Architecture). Master Node is a BalticLSC Software instance which stores information on assets (users, resources, applications) in the Network and provides the functionality to create, debug and execute BalticLSC applications. All software is web-based, i.e., it is accessible through a browser or through web-services (e.g., REST API). The storage where assets are stored is called BalticLSC Registry. If there is more than one Master Node in the network, then the BalticLSC Registry is synchronized between all Master Nodes. Thus, each Master Node is aware of every resource, user, and application available in the network and may access them even if the particular Master Node where resources have been registered is no more available. The purpose of having multiple Master Nodes is the load balancing and efficiency of data transfers through the network.

Resource Clusters are Kubernetes clusters with BalticLSC Platform installed and registered in a BalticLSC Master Node. Resource Cluster enables access to the hardware nodes provided by a resource supplier.

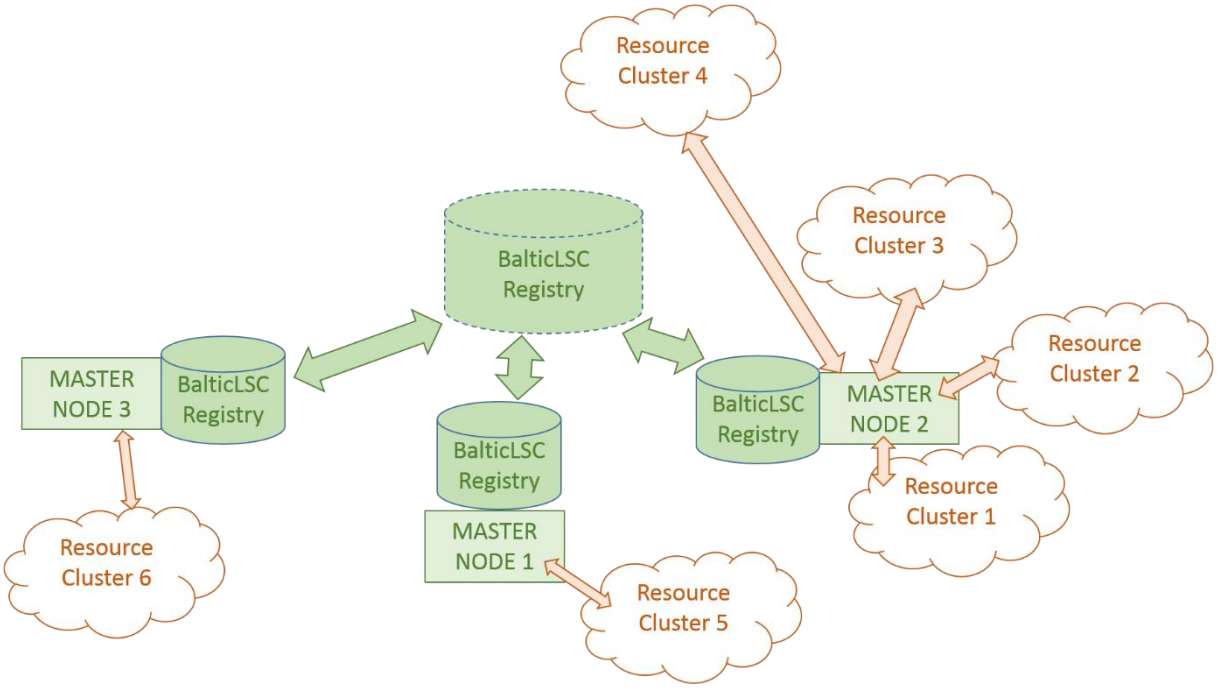

Figure 12 BalticLSC Network – High-Level Architecture

The main components of the Master Node are (Figure 13 BalticLSC Network Master Node and Resource Cluster):

1. Registry
2. Admin Tool
3. App Store
4. Computation Tool

The registry is a logical datastore where information on user accounts, computation resources, computation modules, applications, tasks, task execution logs, billing is persisted. Datastore does not need to be homogenous and different parts may be implemented using different database technologies (e.g., relational databases, document databases, etc. ...)

Admin Tool is software which is used to manage the assets of the Baltic Network. It is used to manage user accounts, computation resources, computation modules, and applications by BalticLSC Network's

Administrators. Managing computation resources involves also monitoring the availability and performance of the hardware nodes in the Resource Clusters. This includes active communication to the BalticLSC Platform through Resource Cluster Proxy interface and receiving information from the Cluster Agent – a piece of software continuously running within the Resource Cluster and monitoring the hardware nodes. Resource Clusters can be managed through the Admin Tool by BalticLSC Network's Suppliers. It should be noted that Cluster Agent would be able to detect changes in the Resource Cluster automatically.

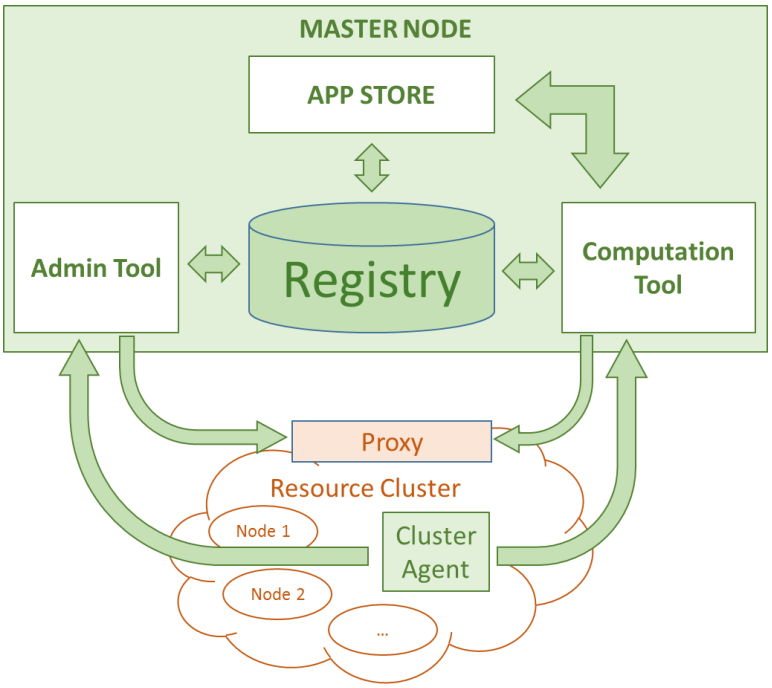

Figure 13 BalticLSC Network Master Node and Resource Cluster

APP STORE is a software component used by BalticLSC Network's Developers to create Computation Applications. It includes Computation Application Language (CAL) IDE – the Development Environment which provides convenient means for the specification of CAL programs. App Store is used to manage Computation Modules. BalticLSC Network's End-users explore the available Computation Applications, make application requests and provides feedback to the Developers.

Computation Tool is used by the End-Users to actually perform the computations. It allows for managing Computation Tasks. Computation Task is based on a Computation Application. It can be executed by Computation Tool. Execution involves finding the appropriate Computation Resources within the BalticLSC Network, running and orchestrating the appropriate Computation Modules, logging the usage of resources for billing purposes and monitoring the status of the execution process.

In the following subsections for all software components (the Admin Tool, APP STORE and Computation Tool) a more detailed design vision is provided using UML Component diagrams.

4.1 Admin Tool

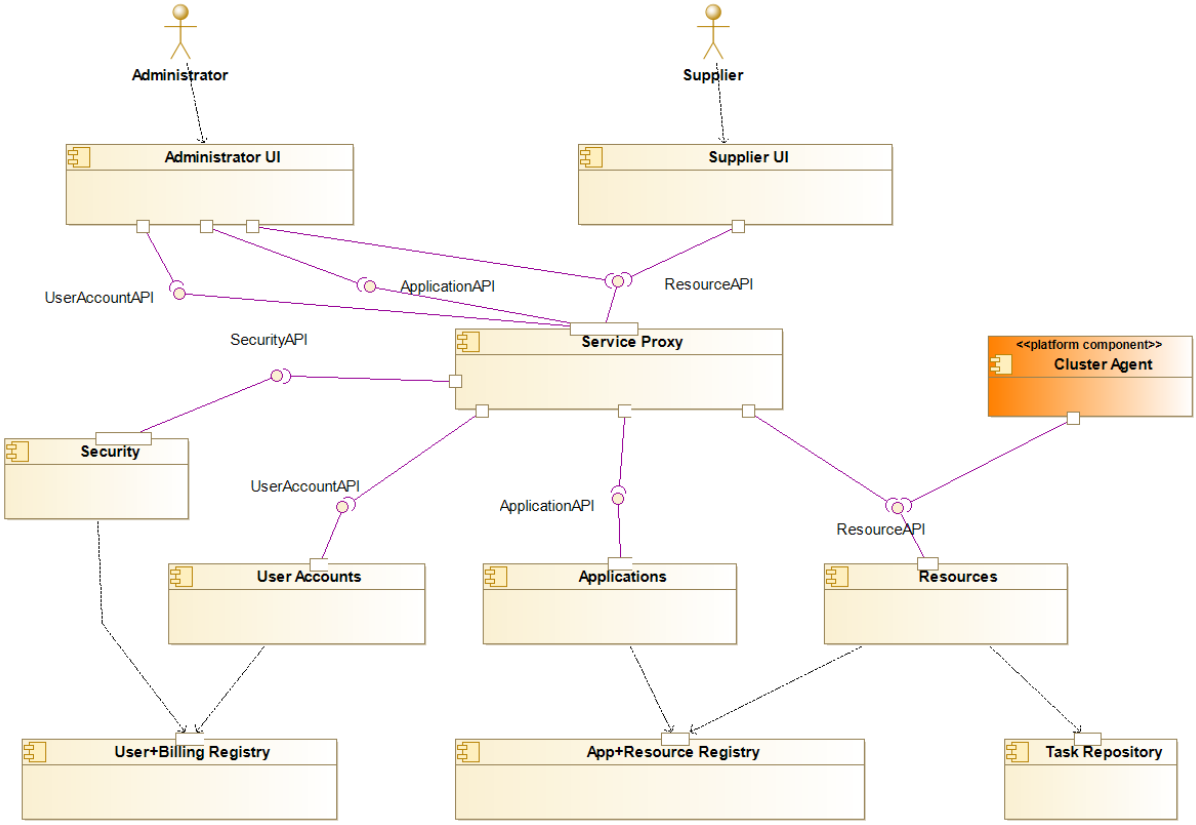

Figure 14. Component Diagram for Admin Tool

Cluster Agent

Agent monitoring computation resources in a given cluster.

Service Proxy

Proxy service intermediary in access to other components.

Security

Module responsible for authorization to system services.

Administrator UI, Supplier UI

Components implementing user interfaces associated with given actors.

User Accounts, Applications, Resources

Modules implementing operations on given subdomains.

User+Biling Registry

Part of the database responsible for storing sensitive information about users and payments.

App+Resource Registry

Part of the database responsible for storing definitions of computation applications and computation resources.

Task Repository

Part of the database responsible for storing information about running and historical computation tasks.

4.2 App Store

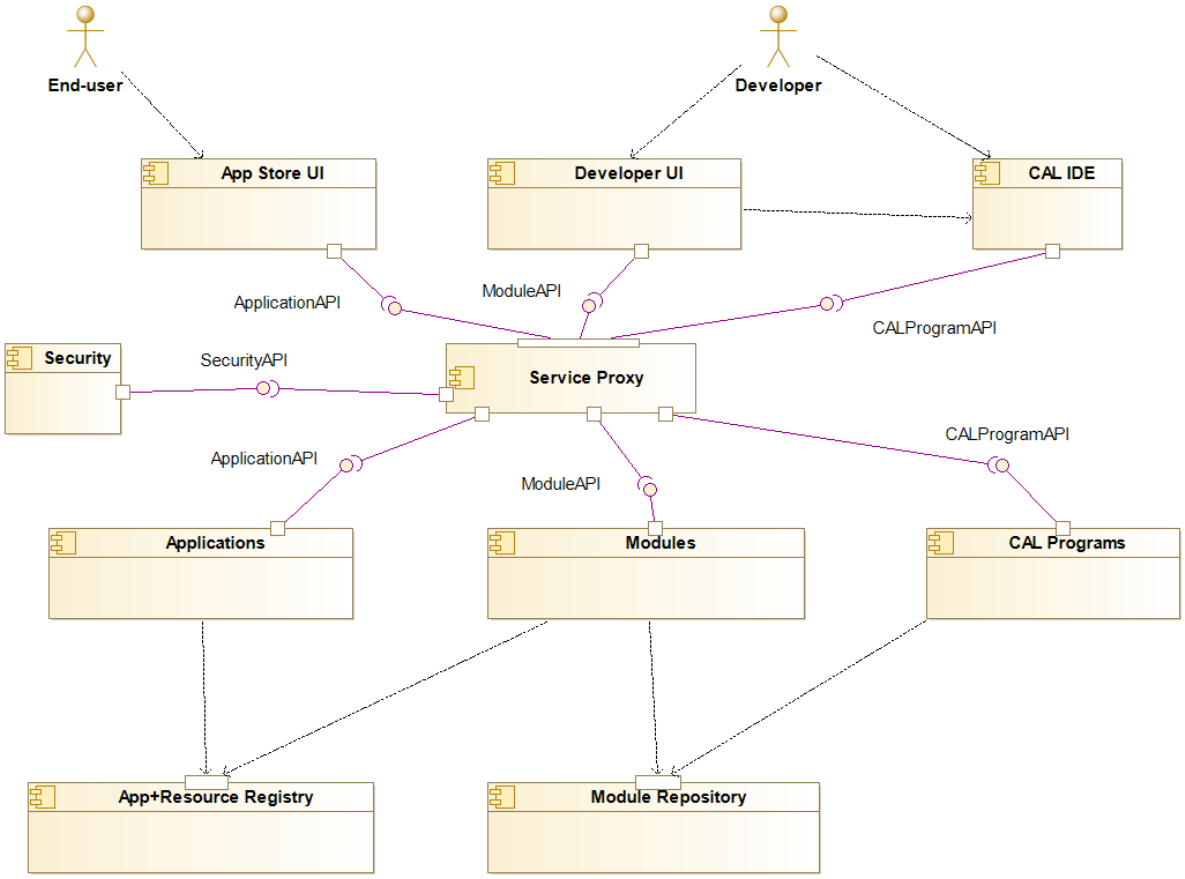

Figure 15. Component Diagram for App Store

CAL IDE

Component implementing the editor for CAL language.

Developer UI

Components implementing user interface associated with the developer.

Modules, CAL Programs

Modules implementing operations on the given subdomains.

4.3 Computation Tool

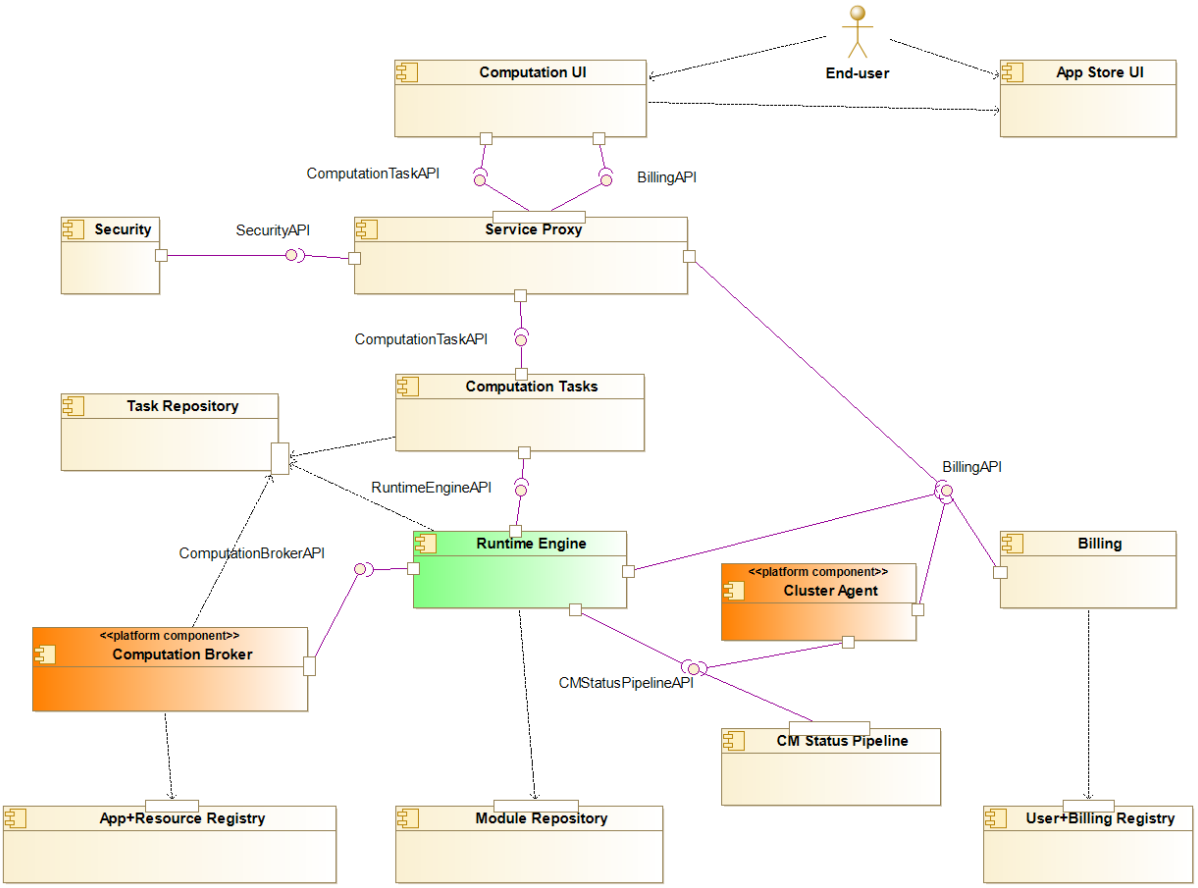

Figure 16. Component Diagram for Computation Tool

Computation Broker

Module responsible for distribution of computation tasks to proper computation resources.

Runtime Engine

Module implementing interpreter of CAL language.

CM Status Pipeline

Module implementing pipeline for Computation Module Statuses.

Computation UI, App Sore UI

Components implementing user interfaces associated with given functionalities.

Computation Task

Modules implementing operations associated with Computation Tasks.

Module Repository

Repository storing code of computation applications and computation modules.

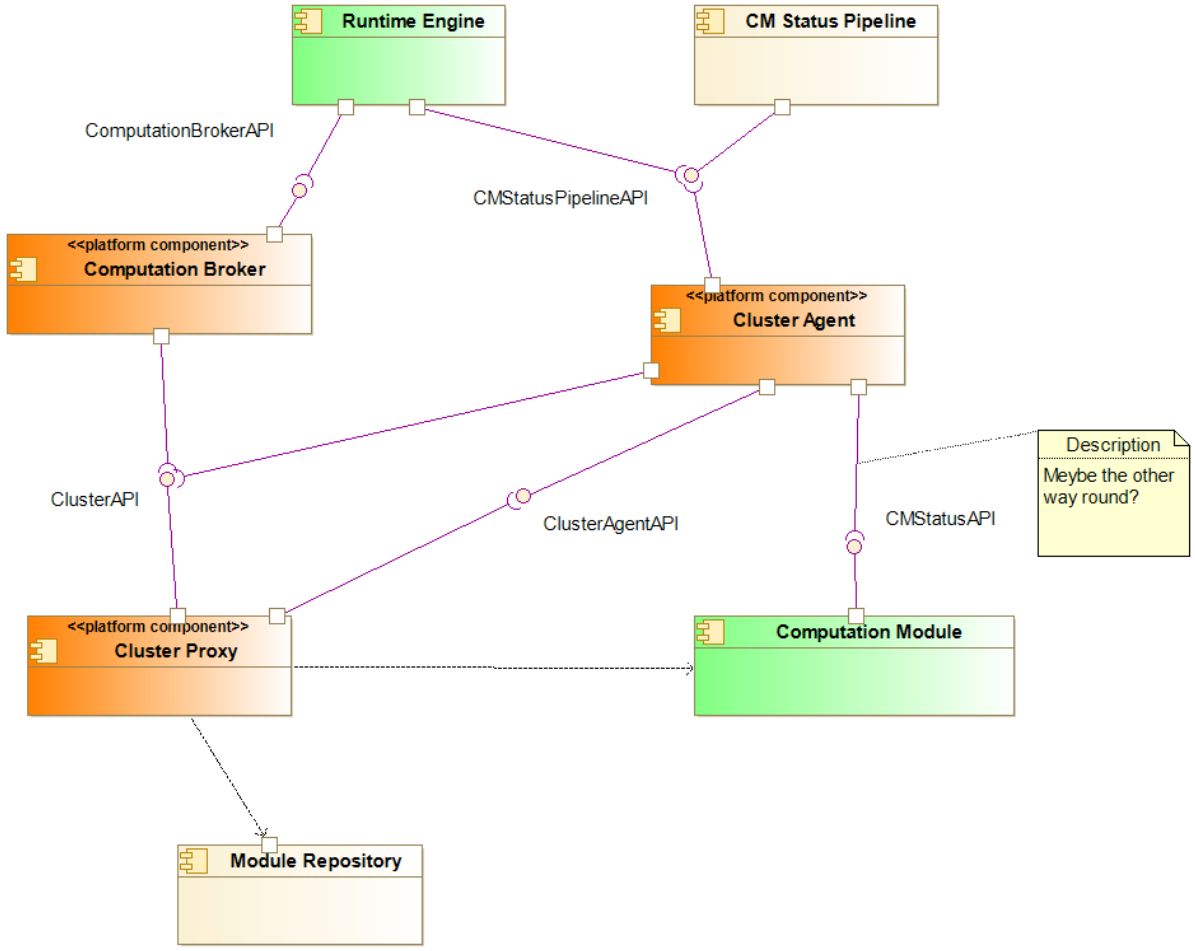

Figure 17. Component Diagram for Computation Modules

Cluster Proxy

A proxy which provides information about computation resources in a given cluster and allows commissioning task to them.

Computation Module

Enclose in container separate computation part.

5. Technologies

Microservices is an approach to software architecture that builds a large, complex application from multiple small components that each perform a single function, such as authentication, notification, or payment processing. Each microservice is a distinct unit within the software development project, with its own codebase, infrastructure, and database. The microservices work together, communicating through web APIs or messaging queues to respond to incoming events.

5.1 Microservices - Basic Architecture

Microservice architecture has been selected as the main approach to software construction in BalticLSC project. The microservice architecture allows to mitigate several problems that otherwise would have to be faced:

1) geographically spread development teams - thanks to the use of microservices, it is possible to divide the system to loosely coupled parts and delegate development of each part to a separate development group
2) scalability of services - beforehand it is not clear exactly which services and how scalable should be, so it is necessary to divide the system into several services and scale each of these services according to current needs

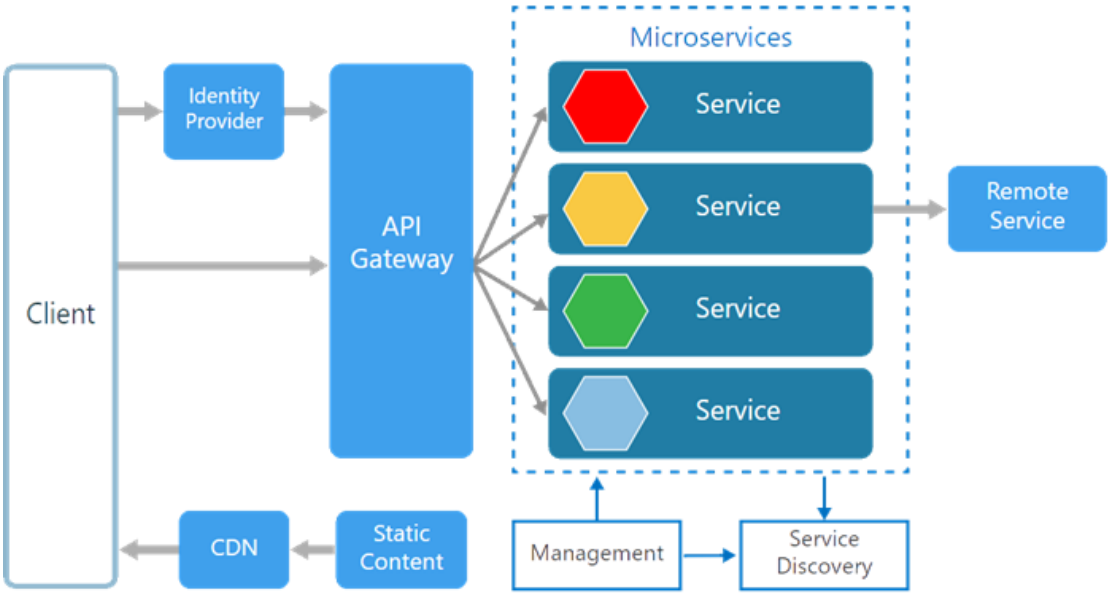

Figure 18 Microservices - Basic Architecture[1]

Figure 18 Microservices - Basic Architecture shows the basic Microservice architecture. Each component of the BalticLSC can be implemented as separate a service within Microservice architecture. Besides these services which provide the functionality of BalticLSC, Microservice architecture contains services which provide the functionality of Microservice architecture itself. These services are:

1. Management - maintains the nodes for the service.
2. Identity Provider - manages the identity information and provides authentication services within a distributed network.
3. Service Discovery - keeps track of services and provides information about service addresses and endpoints.

[1] https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices

EUROPEAN REGIONAL DEVELOPMENT FUND EUROPEAN UNION

4. API Gateway - serves as a client's entry point. It is the single point of contact from the client which, in turn, returns responses from underlying microservices and sometimes an aggregated response from multiple underlying microservices.
5. CDN - a content delivery network to serve static resources, for example, pages and web content in a distributed network.

5.2 Technology review

Microservices architecture allows the use of different technologies (frameworks, database management systems, etc. …) for implementation of microservices. In this subsection, we will provide a description of technologies which cover the needs of the Microservices architecture. Although it is based on .NET technology, we do not ensure the use of it in the project. It is used mainly for illustration purposes. We provide also other alternatives.

Following technologies covers the Microservices architecture:

1) **Service orchestration** – Kubernetes[1] and Docker[2] will be used to orchestrate microservices. The same technologies that will be used by BalticLSC Platform to perform calculations.

2) .Net Core – as **the main software platform**. NET Core is a free and open-source managed computer software framework for the Windows, Linux, and macOS operating systems. It is an open source, the cross-platform successor of .NET Framework. Although there is no need for a single software platform for Microservices architecture, it might ease the distributed development, deployment and maintenance. Other technologies under consideration are Python (Flask[3]), Java.

3) Entity Framework (EF) Core – for **data source access**. Entity Framework (EF) Core is a lightweight, extensible, open source and cross-platform version of the popular Entity Framework data access technology. EF Core can serve as an object-relational mapper (O/RM), enabling .NET developers to work with a database using .NET objects, and eliminating the need for most of the data-access code they usually need to write. EF Core supports many database engines. It should be noted that O/RM-s are available also in other technologies, like, SQLAlchemy[4] for Python, Hibernate[5] for Java. Generally, the selection of O/RM technology does not affect the selection of a database management system. We plan to use relational database management systems, like, PostgreSQL or MySQL. The document-oriented databases, like, Elasticsearch or MongoDB are also under consideration (e.g., for logging).

4) **Communication between microservices** will be organized with one of the following approaches:
    a. **Synchronous** with direct HTTP REST calls
    b. **Internal synchronous** with direct gRPC calls. gRPC (gRPC Remote Procedure Calls) is an open source remote procedure call (RPC) system initially developed at Google. It uses HTTP/2 for transport, Protocol Buffers as the interface description language, and provides features such as authentication, bidirectional streaming and flow control, blocking or nonblocking bindings, and cancellation and timeouts. It generates cross-platform client and server bindings for many languages. Most common usage scenarios include connecting services in microservices style architecture.
    c. **Asynchronous with a message queue**, e.g., RabbitMQ[6]. RabbitMQ is lightweight and easy to deploy on-premises and in the cloud. It supports multiple messaging protocols. RabbitMQ can be deployed in distributed and federated configurations to meet high-scale, high-availability requirements. It runs on many operating systems and cloud environments and provides a wide range of developer tools for most popular languages.

5) An API gateway – takes all API calls from clients, then routes them to the appropriate microservice with request routing, composition, and protocol translation. Typically it handles a request by invoking multiple microservices and aggregating the results, to determine the best path. Ocelot [7]may be used to implement the API gateway. Ocelot is a .NET API Gateway. It provides a unified point of entry into a system running a micro services/service oriented architecture. Other alternatives are Kong API Gateway[8] (Nginx + Lua), Express Gateway[9] (NodeJS), Tyk API Gateway[10] (Go lang), APIMan[11] (Java/JBOSS).

6) Service discovery – the automatic detection of the network location of a microservice by name. Eureka[12] is a tool provided by Netflix to monitor and maintain the registry of all the microservices in the ecosystem. It consists of the Eureka Server and Eureka clients. Eureka Server is in itself a microservice to which all other microservices registers. Eureka Clients are independent microservices. Eureka service discovery is supported by Ocelot.

7) Resilience – the ability to recover after software failures. There is a number of patterns to be implemented by microservices and there are frameworks, that support them. Polly[13] is a .NET resilience and transient-fault-handling library that allows developers to express policies such as Retry, Circuit Breaker, Timeout, Bulkhead Isolation, and Fallback in a fluent and thread-safe manner.

8) Logging – ability to monitor distributed services is challenging at scale. There are several good practices that should be followed, e.g., correlating requests and responses with unique ID within the whole system, sending logs also to the centralized location, etc. … The Serilog[14] is a diagnostic logging library for .NET applications. Serilog provides diagnostic logging to files, the console, and many other outputs. Serilog is built from the ground up to record structured event data. Other alternatives are Log4J[15] (Java), NLog[16] (.NET), Node-Loggly[17] (Node), standard logging module in Python.

9) Security – allowing the usage of services by trusted parties only and protecting data. Basically, it means that the system should provide access control and encryption. JSON Web Token[18] (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA. The OAuth 2.0[19] Framework describes patterns for granting authorization. OpenID Connect 1.0[20] is an identity layer on top of the OAuth 2.0 protocol. It allows to verify the identity based on the authentication performed by an Authorization Server, as well as to obtain basic profile information about the user in a REST-like manner. Another essential element is to encrypt all the communication between client and server with transport layer security (TLS) protocols.

10) Frontend – software that interacts with the user through the browser using HTML, CSS and JavaScript technologies. React[21], Vue.js[22] and Angular[23] are among alternatives.

[1] https://kubernetes.io/
[2] https://www.docker.com/
[3] http://flask.pocoo.org/
[4] https://www.sqlalchemy.org/
[5] https://hibernate.org/
[6] https://www.rabbitmq.com/
[7] https://threemammals.com/ocelot/
[8] https://konghq.com/kong/
[9] https://www.express-gateway.io/

[10] https://tyk.io/
[11] http://www.apiman.io
[12] https://github.com/Netflix/eureka
[13] http://www.thepollyproject.org/
[14] https://serilog.net/
[15] https://logging.apache.org/log4j/2.x/
[16] https://nlog-project.org/
[17] https://www.loggly.com/docs/node-js-logs-2/
[18] https://jwt.io/
[19] https://oauth.net/2/
[20] https://openid.net/connect/
[21] https://reactjs.org/
[22] https://vuejs.org/
[23] https://angular.io/