Architecture & Technology Specification
version 1.2
Sep 2019

Accelerating Scientific Publishing

Connecting researchers and publishers with trusted reviewers to accelerate publishing along with incentivized peer review.
**Introduction**

- Overview ................................................................. 4
- Components .................................................................. 5
  - Components Classification ........................................ 5
  - Functional Components .............................................. 5
  - Operational Components ............................................. 6
  - Observational components .......................................... 8
- Linked Data and Semantic Web ........................................... 9
- Platform ......................................................................... 10
  - Web App .................................................................... 10
  - REST API ................................................................... 10
  - Storage ...................................................................... 11
  - Cloud Functions .......................................................... 11
  - Big Data Analytics ....................................................... 11
  - Big Data Architecture ................................................ 12
  - Event Driven Architecture .......................................... 13
Introduction

This document will cover all aspects of the Orvium ecosystem from a pure technological perspective. After reading it, you will gain enough knowledge to understand what are the solutions the team designing and the technologies we have chosen to achieve our goals.

This file is separated in four main sections:

• Overview,
• Components,
• Linked data and semantic web,
• Platform.

This is a living document that will continue evolving during the development of the project. All the details found in this document are merely informative and they can change in any moment to adapt new requirements.
Overview

Due to the extensive experience of Orvium team in developing complex software solutions and driving technology innovation, we knew that the development of the platform would be an “in-house” job and will be driven using Agile practices. All aspects of the software development life cycle are fully covered as the team consists of developers, DBA engineers, devops engineers, security experts, blockchain developers, cloud architects, etc.

The code base for our solutions will be hosted in GitLab to use all its built-in services like CI/CD, issues, merge requests, registry, code analysis, etc. Our platform will be running using cloud leader providers as this is key for a fast evolving software solution. Orvium has established a partner relationship with these providers as part of our strategic plan.

The main programming language for the interaction with services like APIs, blockchain or databases is JavaScript. Solidity will take part during the development of the smart contracts. Frameworks like Terraform, which uses HCL (Hasicorp Configuration Language) will be used for the definition of our architecture which will be executed following CI/CD best practices. Other frameworks like Angular, Webpack or languages like HTML and CSS will be used for the frontend. For the ”data storage“ we will use a combination of NoSql and Sql databases for document and relational data. For ”binary storage“ we are assessing different decentralized solutions like: IPFS, but this point will be explained deeply further in this document as it is highly coupled to the technology level of maturity.

The technology stack is currently the following:

- AWS Gateway + Lambda functions
- CDN
- Flask
- Execution Queues
- Blob storages
- MongoDB
- GitLab CI/CD
- Angular
Components

Components Classification

We can classify the components in three categories: **Functional, Operational and Observational**. Functional components are those that support the platform functionality, like Storage, API, etc. Secondly, the Operational components are those used to automate operations on the infrastructure itself, like updates, deployments and so on. Finally, the Observational components are responsible of monitoring, alerting, logging and system data aggregation.

Functional Components

The diagram shows a microservice architecture. We are developing Orvium using AWS cloud technologies. The different elements in the diagram are explained below:
• **Route53**: This service provides Domain and DNS management as well as other inherent functionalities to the service such as HA and Healthchecks. In the particular case of this initial design, Route53 will resolve for two main resources: The static content on one side and the platform API (API Gateway) in the other side.

• **CloudFront**: This service is the CDN. It will be used to distribute our static content globally. Hint: Static content is mainly the client code (SPA), document and images.

• **API Gateway**: This services allows us to integrate all microservices behind a single API, where we have full control of the HTTP traffic.

• **Cognito**: In combination with the API Gateway, this services give us the possibility to integrate Authentication and Authorization, custom and/or 3rd party providers.

• **Lambda**: Serverless computing service. It supports NodeJS, Python and serveral other languages to run our application in a scalable way without the overhead of managing the servers.

• **S3 buckets**: There are 2 buckets in the diagram: Input and Public. The Public bucket contains all the content that will be distributed via CloudFront. On the other hand, the Input bucket is a storage used by the application temporary. The idea is to leave room for future asynchronous services that will process the incoming data before making it public. One possible architecture for asynchronous processes could be composed by SQS queues, where the incoming events are sent by the applications to a queue. Such events contain information about the files stored in the input bucket and will be consumed by other services, such as Comprehend, for Natural Language Processing, validations, etc., before to be placed in the public bucket and even registered in the metadata database.

• **Blockchain**: Integrated using decoupled modules from our core infrastructure. This helps us to plug and play in a very easy way, with the provider better fits our requirements. Orvium is integrated with the Ethereum blockchain.

• **MongoDB Atlas**: Fast, secure, and highly available MongoDB service for any scale. Built and run by the team behind the database.

**Operational Components**

Following cloud best practices recommendations as well as general best practices, we are implementing the infrastructure to support our components in code, and automating not only the application deployments but also the infrastructure provisioning. This guarantees that all changes in our platform either for components or either for infrastructure will be recorded.

Therefore, in order to be consistent with the principle described above, we only grant write permissions for developers and operators to access the OCs and only review code. Thus, all operations have to be performed through a revision control system and triggered after changes in the repository.
Developers and operators will push changes to the FCs through a revision control system. We have chosen to store our git repositories in GitLab and use its own CI/CD pipeline to execute calls to Cloud APIs after changes in code. Each of these calls belong to one of the following three categories (represented by their corresponding arrows in the diagram):

- **Infrastructure deployment or update**: Using CloudFormation and/or Terraform, an Infrastructure deployment or update will be triggered after an update in our repository.

- **Static Content deployment**: Some of the solution assets are static content, like GUIs components. After changes in these elements, GitLab CI/CD will deploy them into the public S3 bucket. This will help CloudFront to replicate them globally to reduce the load time of these assets.

- **Microservices deployments**: To deploy our services in Lambda, we are using the serverless framework (https://serverless.com), as it provides a seamless mechanism to run and test locally any Lambda function and deploy them.
Observational components

These components establish mechanisms to guarantee that the platform is up and running and any outage is reduced to a minimum time. For that reason, this phase will cover the details to create a proper "Alert, Logging & Monitoring" (ALM) system.

ALM is a combination of three moving parts which work together and are closely interconnected. ELK stack (Elasticsearch, Logstash, and Kibana) will cover the logging requirements. "Elasticsearch Service" is a fully managed service that delivers Elasticsearch's easy-to-use APIs and real-time analytics capabilities alongside the availability, scalability, and security that production workloads require. The service offers built-in integrations with Kibana, Logstash, and cloud services so we can transform raw data to actionable insights quickly and securely.
Linked Data and Semantic Web

The Academic Knowledge Network is one in a few sectors where structured data is a must to publish content on the web. A number of initiatives, some of them at governmental level, have promoted linked data as the standard way to publish journals. One of the most important criteria to assess quality on public journals is based on whether or not the data exposed in their APIs follows the standards defined by DCMI (http://dublincore.org). These allow crawlers to index data and establish relationships across academic data on the World Wide Web.

Of course, Orvium cannot disrupt the academic sector without following the rules of the semantic web. However, the current ontologies, agreed by the W3C and other entities as standards for scholarship publications, do not cover all the information Orvium’s is aimed to provide. Part of Orvium’s mission is to enrich the Academic Knowledge Network with more granularity and traceability of each piece of science shared on the Web.

To meet this challenge, we need to extend the current ontologies to add more information of our entities. Note that we are now making public, not only the scientific research itself, but also peer reviews, drafts and all versions between the first submission and the final article. Furthermore, we need to include these fields that contain information about the blockchain transactions.
Platform

Web App

Orvium Web App is the frontend and entry point to the Orvium platform. This component is a web application developed using Angular Framework.

Web App interacts with the APIs to authenticate users, write and retrieve data, show notifications, etc. When the user performs an action that requires the use of the blockchain, it uses to interact with the blockchain and the smart contracts.

- **Dashboard**: Orvium Dashboard provides a friendly interface to gain quick insight on the status of your articles and reviews. At a glance, users can take actions, retrieve statistics or interact with other users.

- **Explorer**: Orvium Explorer enables users to find articles from other researchers by querying based on specific keywords.

- **Editor**: Orvium Editor allows users to write their articles in a standardised way by the research community. It also helps Orvium to identify the different sections in an article, provide better discoverability and categorization.

- **Papers**: User will gain control of their articles and the status of their reviews via Orvium Articles. It will also provide information about the impact of the article in the Orvium community, comments from other researchers, views, bookmarking and more.

- **Reviews**: The Reviews component is the mechanism for researchers to request other peers for validation of their work. Once an article is in review phase, users can access these reviews and accept or decline them. Based on these decisions the impact of the paper might be affected hence the researcher reputation.

- **Citations**: This component is highly relevant as it connects the article findings with the work in other reviewed articles. Citations also might interfere in the impact of other papers as it represents a bridge between the work of many researchers throughout history.

REST API

In order to provide a seamless and auto scalable communication mechanism with third parties, we will provide a REST API hosted in Cloud API Gateway service. This service gives us the ability to abstract multiple tasks involved in the development process of a regular API and provides auto scaling functionalities out of the box.
Besides, it integrates with other services which will give us great insights on the performance of the API. Monitoring and alerting will be key for this component as we expect many third parties integrating with us.

Orvium REST API provides the data and the business logic to the Orvium UI. The API uses an internal database to store all the metadata concerning the research article lifecycle. This metadata is used to give the user details about his articles, peer-reviews, pending actions, impact factor of his research, notifications, etc.

The Orvium REST API is also responsible to store the raw data from the articles. The Storage Layer section explains more details about it.

Orvium REST also interacts with the Ethereum blockchain to verify important details for the lifecycle of research article publications. To do this, it sends RPC-JSON calls to the blockchain using the Infura services. This verifications can be done in the client side as well in the same way the REST API does it. Infura provides secure, reliable, and scalable access to Ethereum and IPFS.

**Storage**

Orvium platform is agnostic in terms of specific storage technology and it will be possible to work with multiple storage providers. This also allows to remove or add new storage providers in the future if it is necessary.

MongoDB is used to store persistent user data like: accounts, comments, likes etc. Regarding raw data, we are currently using AWS S3 and we are evaluating the use of decentralized storage solutions like IPFS.

**Cloud Functions**

Cloud functions are small pieces of logic triggered by a particular event, like submitting a paper or review in the platform. Cloud functions have auto scaling capabilities out of the box, which means, no matter how many number of request we get that the platform will grow automatically based on that demand. If the number of requests go down, Cloud functions will react accordingly making cost efficient.

**Big Data Analytics**

We will integrate heterogeneous data sources such as transactions in the blockchain, existing research literature, connections between researchers and publications, observe user behaviors, etc. to provide meaningful insights to the users in the platform.

Initially will cover different use cases:

- Calculate the impact factor in an optimal way using public information and data
• Identify citation and peer review rings
• Suggest the right peer reviewers to increase the quality of the work
• Facilitate peer review accuracy
• Identify and suggest keywords for the manuscript
• Find and identify indirect relations between publications
• Automatically classify papers by field based on their content
• Help to identify the journal that best fits a paper

**Big Data Architecture**

• **Hadoop & Spark**: Spark is a powerful tool for processing large amounts of data in parallel. We will use it for our ETL, but also for designing, training and running machine learning models with Spark MLlib.
- **Apache Kafka**: This component will be the backbone for our analytics infrastructure and events in the platform will be sent to be processed by multiple services. For example, when a user submit a new paper, this event will be sent to a topic where different consumers (or microservices) will classify the content of the paper, analyse citation rings, identify plagiarism, suggest missing keywords or tags, etc.

**Event Driven Architecture**

OP is meant to process multiple long running requests per second, i.e., the process of metadata in a blob file. It’s a high availability system and the performance is key to engage with our users. We cannot allow process to hang forever and the users have to be informed of what is happening under the scenes: when the paper is being processed, when it’s finished and ready to make public, etc.

For these reasons we introduce Event Driven Architecture which is a common pattern in Software Development and it’s probed that can serve a very demanding ecosystem like Orvium. CQRS (Command Query Responsibility Segregation) is a pattern that fits well with Event-Driven Architectures. It’s common to see CQRS system split into separate services communicating with Event Collaboration. This allows these services to easily take advantage of Event Sourcing.