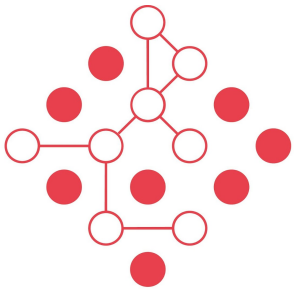


DATA MARKET

AUSTRIA

www.datamarket.at

D4.1: First Version of the DMA Federated Cloud

Deliverable number	*D4.1*
Dissemination level	*Internal*
Delivery date	*31.12.2017*
Status	*Final*
Author(s)	*Martin Heuschober, Michael Kofler-Häusler, Axel Quitt, Elisabeth Rössner, Michael Schmollngruber*

FFG

The Data Market Austria Project has received funding from the programme “ICT of the Future” of the Austrian Research Promotion Agency (FFG) and the Austrian Ministry for Transport, Innovation and Technology (Project 855404)

bmvit

Executive Summary

The present document is a detailed description of the design decisions made to implement the technical and infrastructural basis of a Data Economy Platform using a federated cloud approach. In particular, this platform will serve as the foundation of the Data Market Austria (DMA).

As this is a research and development project our (meaning the members of Work Package 4, WP4) emphasis in the first month of the project was to develop a solid vision of the structure and components which are needed to produce a functional result and fits the operational model of the Data Market Austria.

In the beginning we started with a monolithic approach, but soon it became clear, that this working hypothesis needed to be changed to a more flexible and decentralized approach. Otherwise the ideas of becoming an “OPEN” platform could not have been realized.

Following the demand of FFG, that various options of computing infrastructure needed to be technically and economically evaluated, we proposed either the use of a bare metal infrastructure or the Open Telekom Cloud (OTC), which is quite similar to the offering of other Public Cloud Providers (e.g. Microsoft Azure). After consulting our findings within the consortium, we suggested to opt for an OTC-based approach for the duration of this project.

OTC comes along with a technology stack, which is quite common in the consortium, including Docker and Kubernetes.

To enforce a certain set of best practices, which builds the basis for smooth inter service communication some functionality needs to be run and administered centrally. We call this entity the “Central Node”. This Central Node runs on the OTC and distributes requests to either the same or other Clouds.

The Central Node also provides the Data Market Austria Portal, which is the landing zone for all applicants, which are interested in consuming DMA Services or Data.

The new GDPR legislation also led to a rethinking of some prior concepts to ensure a strict separation of processing metadata and detailed data, which might include personal information. To be compliant with this legislation we have redesigned the Central Node in a way, so that in fact only metadata will be processed within the infrastructure associated with the DMA legal entity. Detailed data will be processed in separate entities which can be legally defined by Data Processing Agreements (DPA, in Austrian legislation: Auftragsdatenverarbeitung (ADV)).

In this document you will find the current status of the first version of the federated cloud and examples showcasing how the Central Node, based on the OTC, interacts with services in other cloud environments. To show the feasibility of the federated cloud we connected the Central Node with the CAS authentication and authorization service hosted by Catalysts in a separate cloud.

In accordance with the European legislation, who changed the environment for data processing and data economy projects (like DMA), the integration of real world datasets was not possible.

We, T-Systems, as leader of WP4 and our consortium members working in this package are confident to gain speed, now that common directions and the architecture is established. And we will present a more mature second version of the federated cloud in D4.3 and the final version in D4.5.

Our next steps focus on integrating services like blockchain, metadata search and others into the Central Node and presenting these services on the Portal landing page.

Table of Contents

List of Figures

List of Abbreviations

CAS	Central Authentication and Authorization Service
CCE	Cloud Container Engine
DMA	Data Market Austria
DPA	Data Processing Agreement
GDPR	General Data Protection Regulation
HA	High Availability
OTC	Open Telekom Cloud
PaaS	Platform as a Service
PoC	Proof of Concept
TLS	Transport Layer Security
WP	Work Package

1 Introduction

1.1 Different Approaches to implementing a federated cloud

Building an orchestration layer for any software project using multiple components and external services is a complex endeavour and has been subject to constant re-design in the last few years.

The **classical approach** is to create a single monolithic application that performs tasks like:

- monitoring
- logging
- alerting
- configuration
- registration and coordination of services
- scheduling
- (re)starting services on demand

Typically an application like this would run on a dedicated custom hardware to provide fault tolerance and stability, and external components would communicate via SOAP or other (proprietary) protocols.

The **modern approach** is to organize it with microservices, i.e. each application is minimal in the sense that it only performs one of the above tasks, and all of those microservices communicate over REST-interfaces or over a message broker. Each microservice would run on a virtual machine or commodity hardware. Fault tolerance in a microservice architecture is usually obtained by running multiple versions of each service at the same time or – assuming the start-up time is short enough - starting services on demand. Packaging and bundling each microservice is usually done via a container technology like docker, which provides a running environment (i.e. all necessary libraries) and the (microservice) application.

Another microservice technology worth mentioning is the use of unikernels, which are a bundled operating system/application and run virtualized directly on top of hypervisors like XEN or kvm. Without the need of a container, the use of unikernels is usually resulting in a faster start-up (milliseconds vs. minutes), and a smaller attack surface, since it only contains a minimal set of libraries.

1.1.1 Benefits of Microservices

- **Real fault tolerance**
  Docker and Kubernetes (Orchestration and Cluster Management) or OpenShift (PaaS) provide enough abstractions for realizing failover to other services and restarting services/service groups which have died or become unavailable, tasks that are well understood and though complex by nature, very much manageable. Compared to a monolithic application, a microservice architecture should never have a single point of failure.

- **Upgrading and replacing services**
  Having multiple versions of a service, protocol or interface allow to easily upgrade a service and dependent components, while other services can still use the parallel running old versions. Upgrading one library within one microservice is not affecting any other microservice, whereas in a monolithic approach one task might force you to use an older library in the whole application where every other “component” would be fine with upgrading that library. Especially for components that are accessible from the internet not being able to upgrade or patching a component is a critical security issue.

- **Scalability**
  Starting 5 or 500 microservices is just a matter of configuration, and adjusting to service usage is a fully automatized process, and easily done with multiple providers like Open Telekom Cloud (OTC), AWS, Azure and the like. In comparison, the classical monolithic approach would need better hardware, which might result in significantly higher expenses, whereas the higher performance is only needed at utilization peaks. Microservices, on the other hand, are scaled up or driven down by the demand.

1.1.2 Downsides of Microservices

- **Complexity**
  Usually a setup of microservices involves multiple instances of one service, which possibly need to work (over a network) to achieve a consensus, to allow data integrity. Moreover, microservices are not constrained to one (virtual) network, but can exist on different (virtual) machines in different computing centres in different time zones/continents, which lead to another class of possible errors.

- **Resource usage**
  Although the term microservice suggests a small impact on resources, the average need of a Spring Boot microservice is about 1 GB RAM. Here a monolithic application profits a lot from sharing resources and libraries.

- **Bug tracking across the system**
  Debugging or finding bugs/root causes is inherently difficult in a distributed system.

A more in depth analysis of the pros and cons of microservices can be found in (Newman, 2015).

1.1.3 Reasons for choosing the microservice based approach

One of the most fundamental results of Work Package 4 is the evaluation of the economic and technical aspects of a monolithic approach versus a microservice based.

The classical approach is reflected in the early infrastructure architecture draft shown in Figure 1.

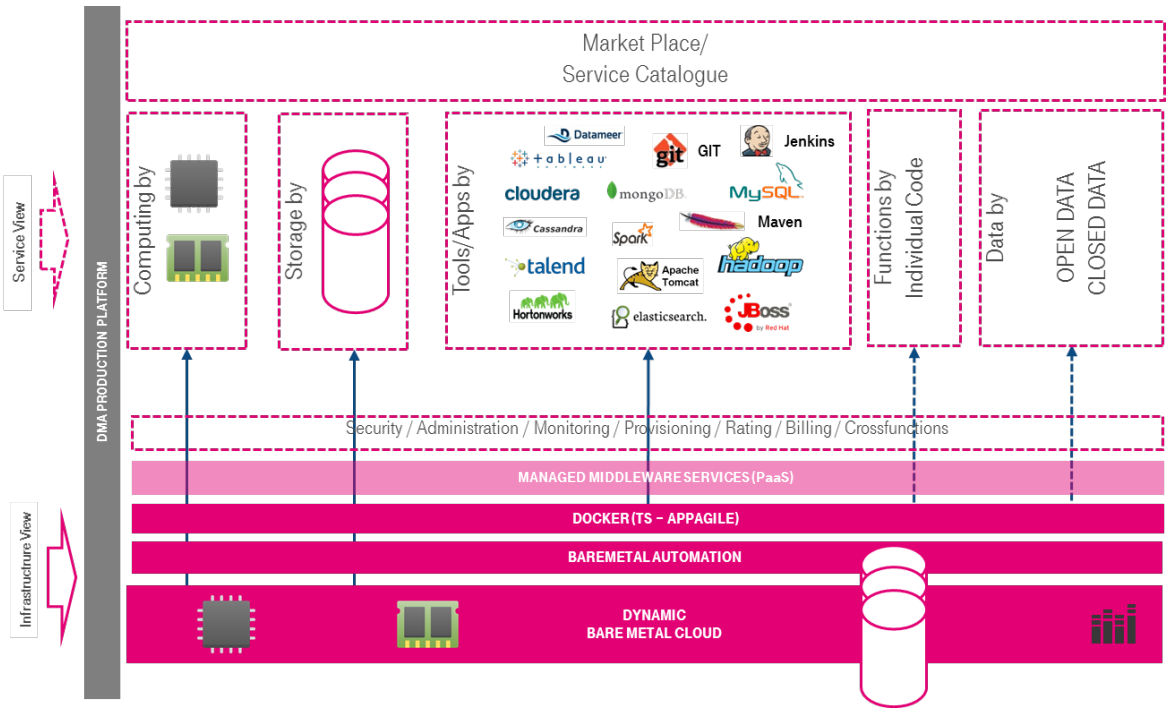


Figure 1: Initial architecture concept for DMA

Historically the computing power has been the expensive part of development, whereas nowadays the time of developers, analysts and designers is becoming the bottleneck, when at the same time hardware is getting cheaper. Thus the downside of resource usage is negligible. But maintainability and short development cycles are more important reasons to consider when deciding an architecture.

The above requirements of fast and reliable development combined with high availability and fault tolerance are met in a cloud based infrastructure. It was the consensual decision of FFG and the consortium to leverage OTC as infrastructure, as the OTC already provides Kubernetes and other software to manage dockerized services. As the OTC is the predestined environment for using a microservice architecture, the next logical step was the choice of using microservices and therefore gaining the benefits of distributed computing.

2 Architecture and technology overview

2.1 Conceptual Design of the DMA federated cloud

DMA Cloud Federation refers to the unionization of software, infrastructure and platform services from disparate networks that can be accessed by a client via the internet. The schematic is shown in Figure 2.

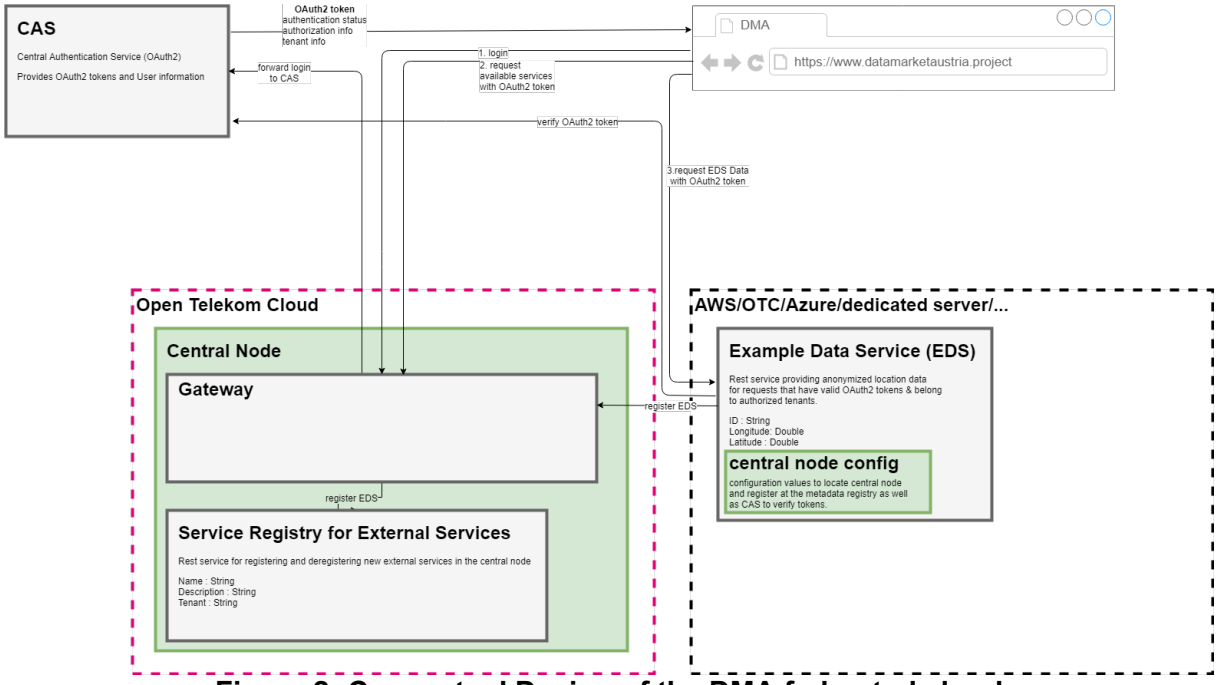


Figure 2: Conceptual Design of the DMA federated cloud

The federation of cloud resources in DMA is facilitated through the Central Node's gateway that connects public or external clouds, private or internal clouds (owned by a DMA infrastructure provider) and community clouds (owned by several cooperating DMA infrastructure providers); creating a hybrid cloud computing environment.

It is important to note that federated cloud computing services still rely on the existence of physical data centres.

2.1.1 Central Authentication Service (CAS)

The Central Authentication Service (CAS) is a software package providing single sign-on. Its purpose is to permit a user to access multiple applications, while providing their credentials (such as user ID and password) only once. It also allows web applications to authenticate users without gaining access to a user's security credentials, such as a password, by using an OAuth2 token in our case.

2.1.2 Central Node

The Central Node consists of two main system parts:

- **Gateway**: The entry point into the Central Node and dispatcher of synchronous and asynchronous requests from the internet to all services registered at the registry for external services.

- **Registry for external services**: Overview of existing services with name, description and assigned tenant.

2.1.3 Example Data Service

Example REST service including a GUI component, which provides anonymized location data for requests that have a valid OAuth2 token and belong to an authorized tenant.

2.2 Technologies and infrastructure used in the Central Node

Designing such an infrastructure from scratch is a complex endeavour and quite error prone with a steep learning curve. Therefore we are leveraging existing, open-source technologies that have stood the test of time and are used by many companies, which report errors and contribute to make the existing libraries/components better.

The Central Node facilitates the following technologies:

- Open Telekom Cloud: Infrastructure provider based on OpenStack
- Docker: container for each microservice
- Kubernetes: Production-Grade Container Orchestration
- JHipster: generator for Spring Boot based microservices + best practices/configurations
    - Netflix-Open-Source Stack: providing gateway, registry and configuration service
    - Kafka: Message broker for asynchronous communication between services (alternative to HTTP-REST calls)
    - ELK-Stack (Elasticsearch/Logstash/Kibana): provides logging for distributed systems including a powerful dashboard/search
    - Prometheus: metrics aggregator for distributed systems
    - Grafana: visualization of metrics
    - Swagger: API documentation/unified interface description
    - Angular + Bootstrap: front end and CSS framework/technology
- CAS: Central Authentication and Authorization Service (CAS) provided by Catalysts
- Nginx: acting as a reverse proxy and handles TLS termination

2.3 Architecture of the Central Node

The current development version of the Central Node consists of the components shown in Figure 3.

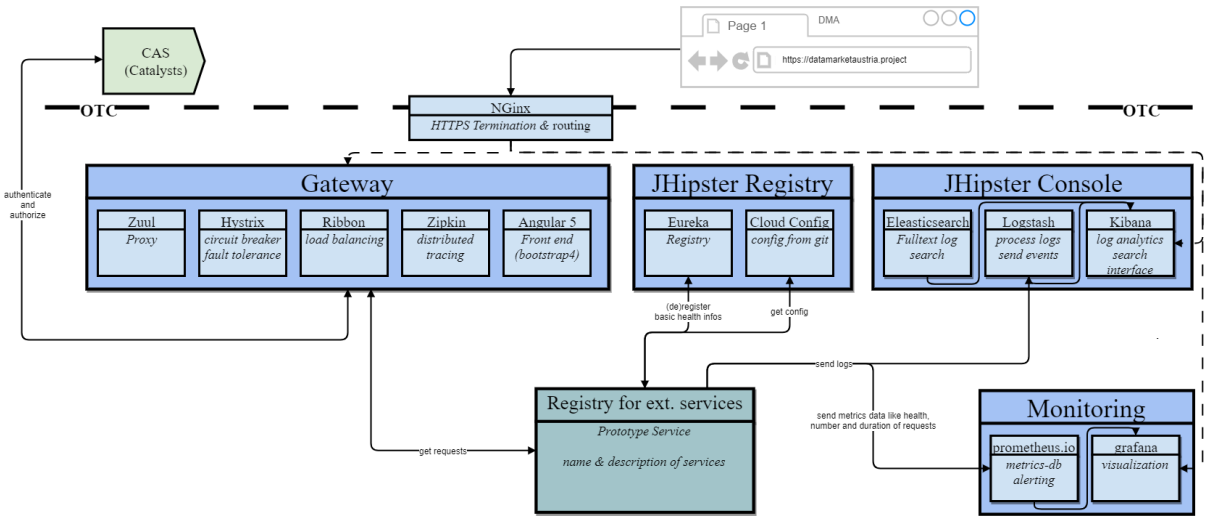


Figure 3: Architecture of the Central Node

The universal entry point for microservices is the **gateway**, a docker container, which provides a website to login, navigate, and access the metadata registry service.

The tasks of user authentication and authorization via OAuth2 are delegated to the external service **CAS** (provided by Catalysts).

Every microservice within the Central Node will consume its configuration on start-up from an **eureka registry** based on a folder structure or git repository. Furthermore, every service will register basic data on start-up and de-register on shut-down.

Part of the configuration is the location of the **console**, i.e. the three docker containers for elasticsearch, logstash, and kibana, so every microservice can send its **logging** data to a centralized location, where they are indexed for full-text search and stored.

Another part of the configuration is the location of the **monitoring** component - two dockerized services, prometheus and grafana, used for data aggregation and visualization, respectively.

The **registry for external services** is a prototype microservice, which stores meta-information like name, and description of each external data- and service provider. All of this information is secured by OAuth2 such that no information can be accessed without registration at the CAS server.

All of the infrastructure in the Central Node (except for the external component CAS) is protected and secured by a reverse proxy – **nginx**, such that ports and services are not exposed directly to the internet.

2.4 Overview of the existing components in the Central Node

2.4.1 Screenshot: Landing Page

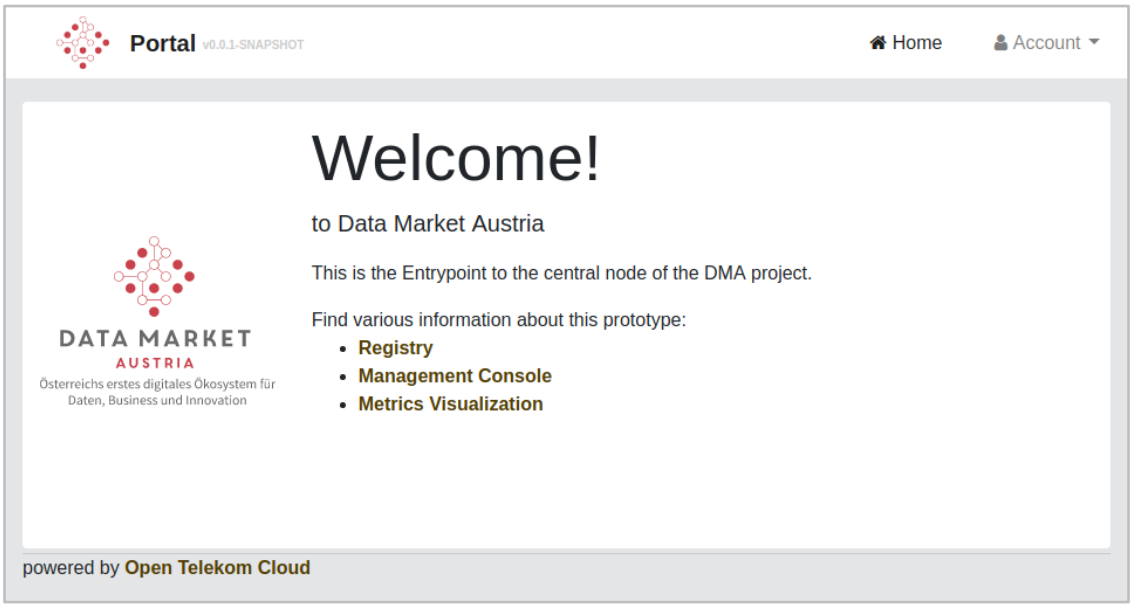
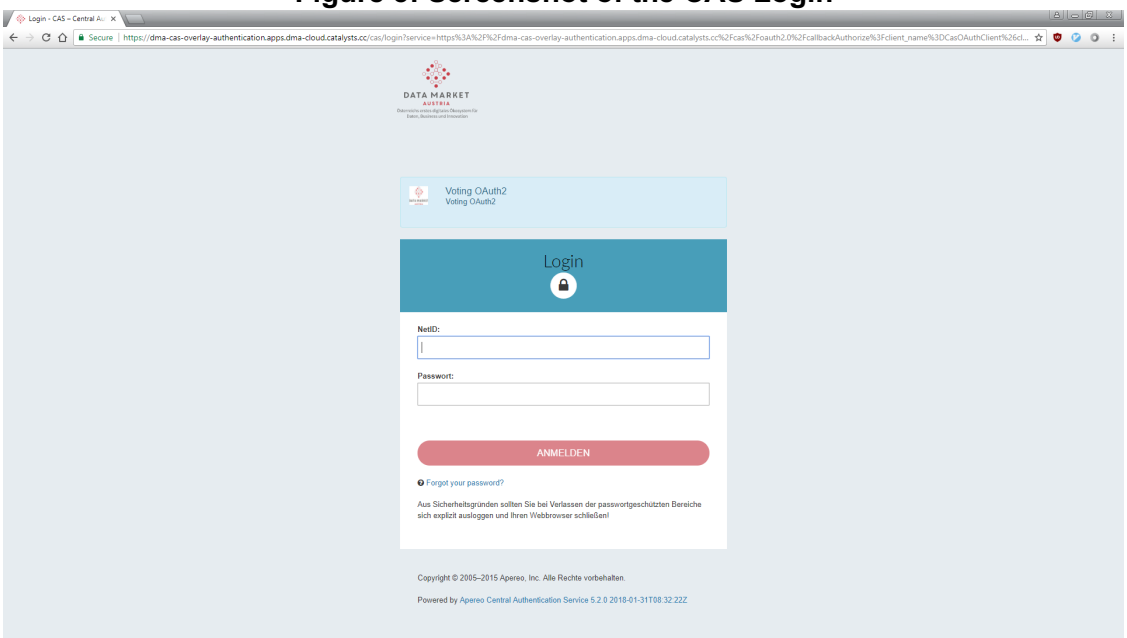


Figure 4: Screenshot of the Portal Landing Page

2.4.2 Screenshot: CAS Login

Figure 5: Screenshot of the CAS Login



2.4.3 Screenshot: Internal Service Registry

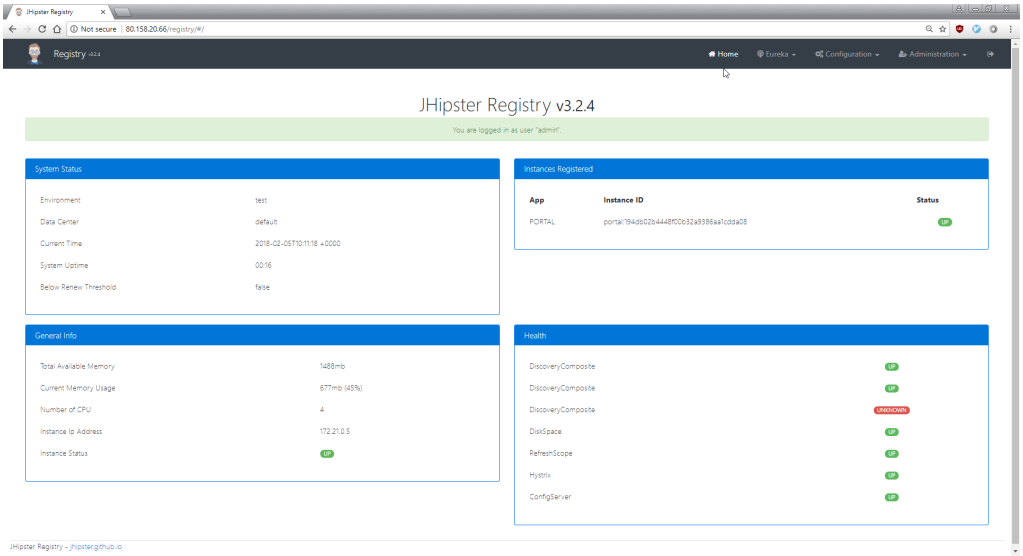


Figure 6: Screenshot of the internal registry for services inside the Central Node

2.4.4 Screenshot: Console/Kibana

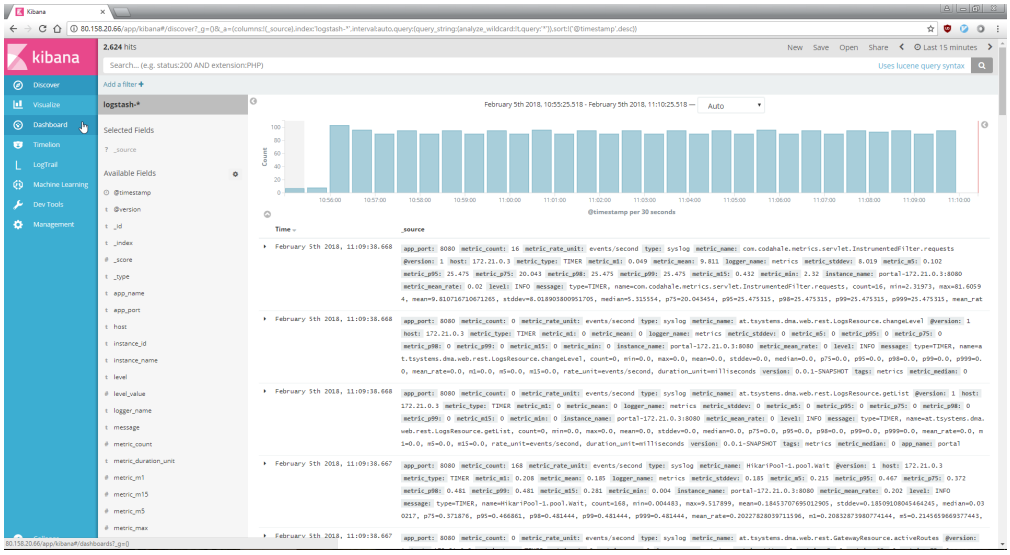


Figure 7: Screenshot of the Console/Kibana

2.4.5 Screenshot: Grafana

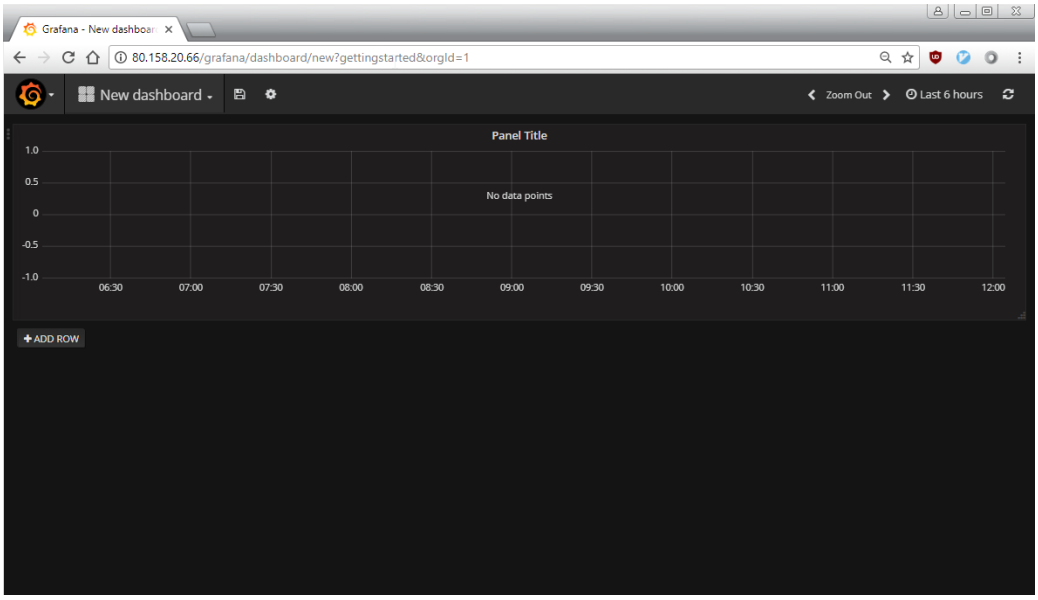


Figure 8: Screenshot of Grafana (no data in the system available yet)

3 Next steps in developing the Central Node

The first Proof of Concept (PoC) version of a microservice based Central Node is running in OTC and it is subject to ongoing further development. The following list highlights some of the improvements that are currently being implemented:

- Securing the communication from the client to nginx with HTTPS
- Providing a more fine grained security concept for authentication of services within the Central Node (modularization, centralization)
- Setting up fault tolerance and high availability (HA), by making components like the service registry redundant
- Integrating full traceability with audit logs and augmenting the security with tenants
- Improving the compatibility between CAS and Spring security
- Setting up advanced metrics and visualization with prometheus and grafana
- Deploying to Cloud Container Engine (CCE) and setting up a modern staging environment with a redundant reverse proxy/load balancer for production.
- Securing the internal service communication by means of Transport Layer Security (TLS)

3.1 Services for the Central Node

Goal of this project is to provide an infrastructure to access data-services as well as services providing business logic. Here we want to list a few and explain how they will integrate with the Central Node.

3.1.1 Service registry for external services

This registry will serve the purpose of giving an overview of existing services as well as providing a platform for administration purposes. It also serves as a monitoring tool for currently deployed services.

- Overview
  - name and description of each external service and its provider
- Administration of containerized infrastructure
  - registration and deregistration of services
  - configuration information for services (e.g. endpoints of CAS, gateway, etc.)
  - managing dependencies between external services
- Monitoring
  - health check
  - number of instances of each external service
  - location of each external service instance

One thing to note is that the registry for external services is just a technical registry. Where the metadata registry manages business information about datasets and services as well as descriptions value to the end user, this registry collects only information about location of services to be able link to, name, a minimal health status, as other technical information, which are to be defined within WP4.

3.1.2 Billing engine

Another central part is the billing engine, where invoices to DMA Customers will be generated based upon the individual consumption of data, infrastructure, or services.

In the beginning, our idea was that billing or at least the measurements of consumptions shall be a central service. Due to the fact that European GDPR legislation sets special requirements to process personal data, our point of view is that billing itself shall be a service outside the Central Node. In our discussions we often came around the topic that in an “OPEN” Data Economy you cannot prevent a Billing Service Provider to provide his services also on the Data Market.

The Billing of Data Market Services need to follow a “Multilevel”-Billing approach. This means that various tariff and business models have to be defined. Furthermore consumption of infrastructure, services, and data usage need to be bundled, priced, redirected to “Service Consumer” data, and presented to the DMA participant.

T-Systems can provide such a billing system, originally designed for Multilevel-Utility Providers. This system is also built in its major parts on open source Software and might not increase licence fee obligations for the DMA, requiring further evaluation.

The architectural sketch shown in Figure 9 provides an overview on the components which are necessary for a Multilevel Billing approach:

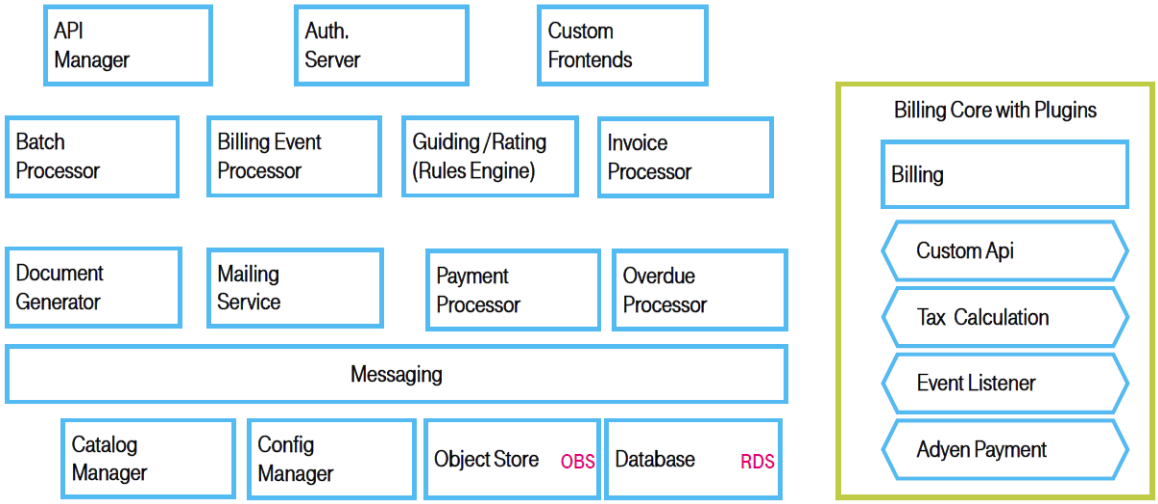


Figure 9: Components of a Multilevel Billing Engine

3.1.3 Data and functional services

External services which provide either data or functionality can be developed in any fashion or language as long as they provide a minimal core to interface with the Central Node, i.e. comply with **registration** at the service registry for external services, use **OAuth2** as authentication/authorization protocol, and communicate via **HTTPS**.

3.1.4 Preview: Developer assistance for a distributed cloud

The main customers of DMA are software developers, data scientists, and researchers. To minimize the entry barrier, we provide tooling for those target groups. In a typical use case, a customer needs to have a first look into a (demo-)dataset before purchasing it. The same is the case for data-centered services.

To allow for such an evaluation with minimal effort, we have created a sandbox environment within the OpenShift platform. In this environment, customers can start a data-science workbench. We have developed respective project templates that set up a self-contained working environment including code editors, interpreter, execution kernel, a best-practice set of relevant libraries, etc. with minimal effort.

As a next step, we will connect these workbenches to the DMA authentication service as well as the data and service catalogue. This will allow us to give customers access to resources they are entitled to in an even more effortless way.

3.2 Security

3.2.1 Securing external-facing services

Both users and services will only use HTTPS (HTTP over TLS1.2) to communicate with the Central Node and external services, if used, to provide a secure privacy and data integrity.

3.2.2 Authorization with OAuth 2.0

OAuth2 is an authorization framework, which enables applications to obtain limited access to user accounts of DMA services. It works by delegating the user authentication to the CAS

service and allows third-party services to authorize the user via a token, which then can be verified with the CAS server at any time. Moreover the CAS server also provides authorization with roles, which can be requested for each user with the same token.

3.3 The orchestration layer as part of the federated cloud

3.3.1 Kubernetes

Kubernetes is an orchestration platform for applications, which run as containers. It handles issues like configuration, replication, load balancing, scheduling of all the nodes or logical groups in the cluster. The basic architecture of a setup containing 3 nodes (1 master node, 2 worker nodes), is shown in Figure 10.

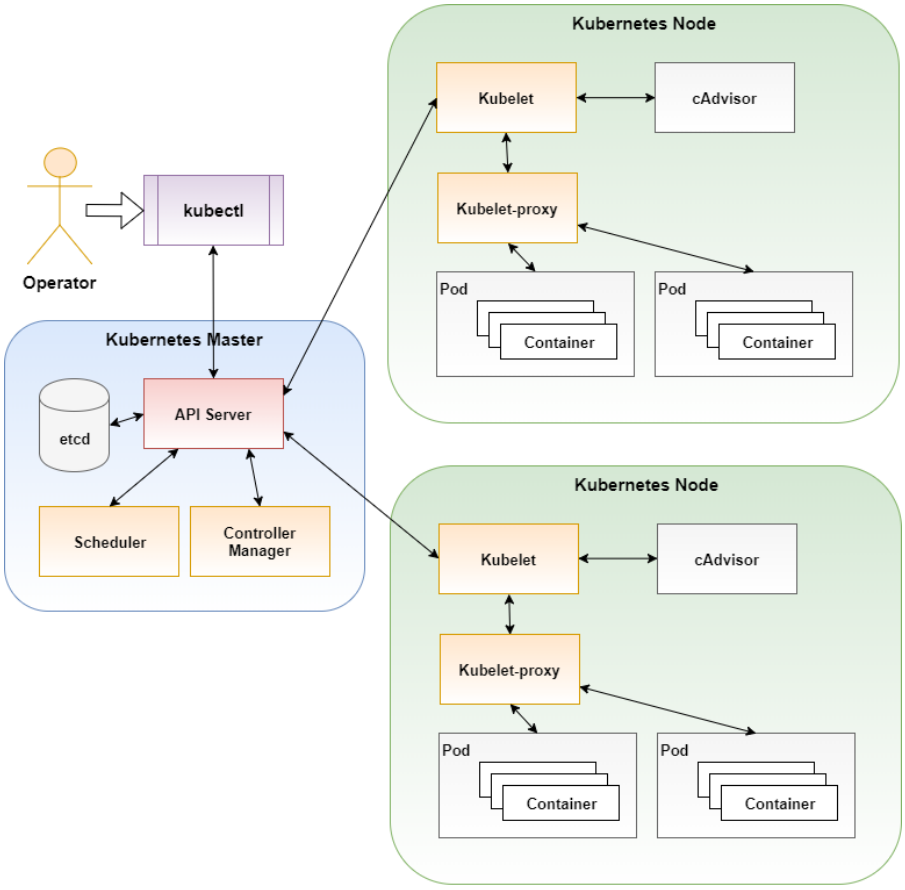


Figure 10: Basic architecture of a minimal Kubernetes cluster

3.3.2 OTC Cloud Container Engine (CCE)

The Cloud Container Engine (CCE) is the management interface for orchestrating services in the OTC based on Kubernetes and provides users with maximum platform independence. As it is based on standard container technology the CCE also offers to access resources from the Docker hub.

The CCE manages clusters, images, templates and container-capable applications as well as operation of the applications. When working with Kubernetes it helps to set up and manage containerized apps.

In the current status of work, the focus was to build a prototype of the infrastructure architecture used for the central node in the OTC. The next step is to design an environment according to the technical requirements to meet the specification of a federated cloud. To integrate the

external services and other cloud infrastructures, it is necessary to evaluate the impact and the consequences of the application of OpenShift versus the application of CCE.

4 References

Newman, S. (2015). *Building Microservices.* O'Reilly Media.

URL: https://kubernetes.io