Bielefeld University

Master Thesis

Concept, design and initial implementation of the de.NBI Cloud Portal

Author:

Maximilian Wiens

Supervisors:

Dr. Alexander Sczyrba

Dipl-Inform. Björn Fischer

A thesis submitted in fulfillment of the requirements

for the degree of Master of Science

in the Informatics in the Natural Sciences (NWI)

Faculty of Technology,

AG Computational Metagenomics

and

Center for Biotechnology (CeBiTec),

Bioinformatics Resource Facility (BRF)

October 30, 2017

Declaration of Authorship / Erklärung

Hiermit erkläre ich, dass ich die vorliegende Masterarbeit selbständig verfasst und gelieferte Datensätze, Zeichnungen, Skizzen und graphische Darstellungen selbständig erstellt habe. Ich habe keine anderen Quellen als die angegebenen benutzt und habe die Stellen der Arbeit, die anderen Werken entnommen sind - einschl. verwendeter Tabellen und Abbildungen - in jedem einzelnen Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht.

Bielefeld, den

Unterschrift:

Printed on non-aging, wood-free, and acid-free paper.

“It is unworthy of excellent men to lose hours like slaves in the labor of calculation which could be relegated to anyone else if machines were used.”

Gottfried Wilhelm von Leibniz

BIELEFELD UNIVERSITY

Abstract

Faculty of Technology,

AG Computational Metagenomics

and

Center for Biotechnology (CeBiTec),

Bioinformatics Resource Facility (BRF)

Master of Science

Concept, design and initial implementation of the de.NBI Cloud Portal

by Maximilian Wiens

The amount of data produced in life sciences is continuously rising and is impossible to analyze on local computers. For that reason the German network for bioinformatics de.NBI is establishing a cloud computing environment called de.NBI Cloud with the prospect to be integrated into the European life sciences network Elixir. For that process and for the interconnection of compute centers a novel cloud platform "de.NBI Cloud Portal" was developed. It utilizes Elixir's authentication and authorization infrastructure and connects five OpenStack-driven compute centers together in an abstract manner. This thesis deals with requirements, design and initial implementation of the de.NBI Cloud Portal.

Acknowledgements

I would like to thank...

Dr. Alexander Sczyrba and **Björn Fischer** for the suggestion to write this thesis and for showing me the right direction, **Peter Belmann** for providing me with useful information, discussion and helpful advice, navigating me through the challenging parts of this work, **Dr. Michal Procházka** for explaining how Perun works, and providing bugfixes, **Tatyana Polle** for proofreading this thesis, **Xenia Wiens** for being patient with me and **Theodor** for sweetening my everyday life.

Contents

List of Figures

Abbreviations

ACL	Access Control List
API	Application Programming Interface
AWS	Amazon Web Services
BMBF	German Federal Ministry of Education and Research
BPMN	Business Process Modelling and Notation
CLI	Command Line Interface
CMS	Content Management System
CSRF	Cross Site Request Forgery
DFN	German Research Network
EC2	Elasic Compute Cloud
GUI	Graphic User Interface
GPU	Graphic Processing Unit
HRZ	University Computer Center
HTTP	Hypertext Transfer Protocol
IAAS	Infrastructure As A Service
IdP	Identity Provider
IP	Internet Protocol
NIST	National Institute of Standards and Technology
OMG	Object Modeling Group
OS	Operating System
PAAS	Platform As A Service
REST	Representational State Transfer
RIPE NCC	Réseaux IP Européens Network Coordination Centre
SAAS	Software As A Service
SQL	Structured Query Language

SSH	Secure Shell
SSL	Secure Sockets Layer
SSO	Single Sign-On
VM	Virtual Machine
XAAS	X (anything) as a Service

Chapter 1

Introduction

1.1 Motivation

The amount of data produced in life sciences is constantly rising. Scientists produce terabytes of sequences, spectrometric images, and similar data. For example, the “Illumina NovaSeq 6000” high throughput sequencing platform generates over 6 terabyte (TB) of sequence data in just two days[1]. The analysis of enormous amount of data on scientists' local computers take a long time or is impossible. However, only a small part of workgroups in Germany have access to large compute clusters for data processing. For that reason, the Federal Ministry of Education and Research (BMBF) started a project to tackle the resource problem in life sciences.

The project named de.NBI (German network for bioinformatics) was founded in 2013. de.NBI should provide training courses and conferences, initiate collaborations and establish the computational cloud for scientists. In 2016 additional funding for the establishment of a cloud computing environment (de.NBI cloud) was approved.

Five data centers (Figure 1.1) will provide compute power for life science workgroups in Germany [11][3]. The compute power should be aggregated to the de.NBI cloud. Scientists will be able to upload, store and process their data. Nevertheless, to efficiently utilize the cloud resources, advanced knowledge in informatics and experience in cloud computing is necessary. For easier access, a central portal with an easy to use interface should be developed.

[1]See `https://www.illumina.com/systems/sequencing-platforms.html`

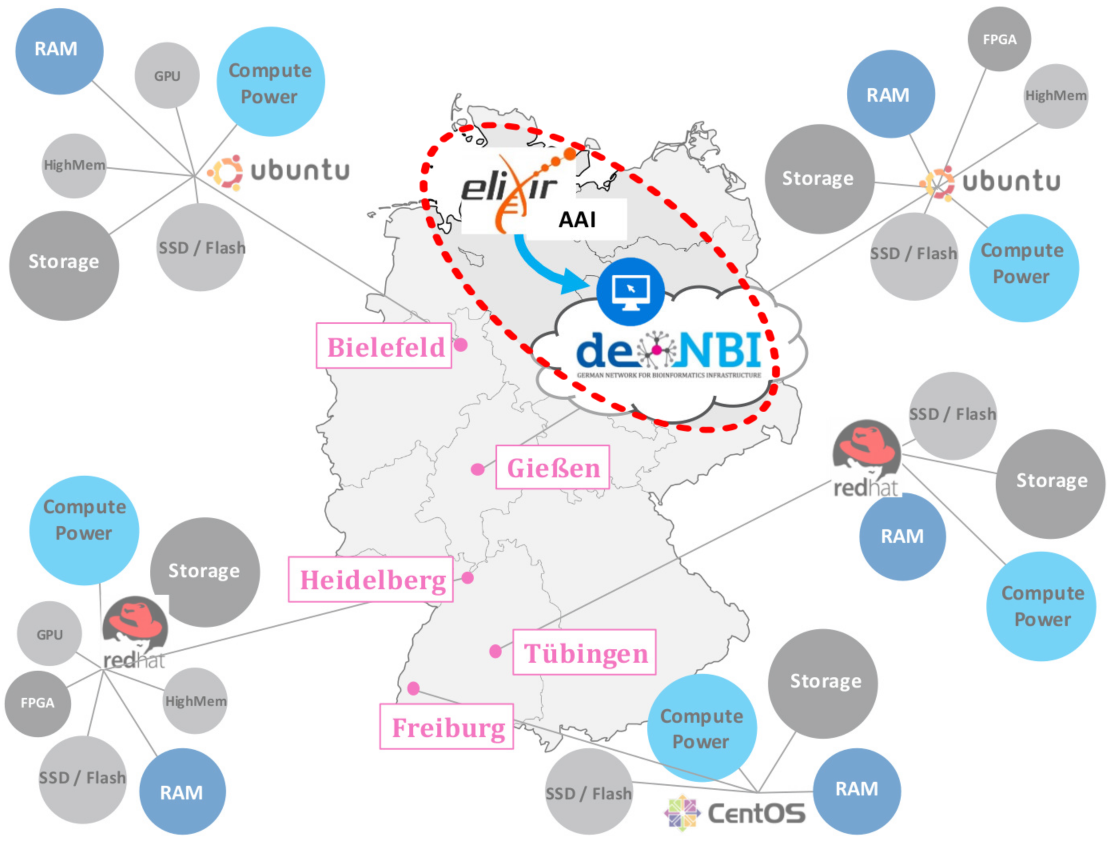


FIGURE 1.1: The data centers in the de.NBI Cloud landscape. The scope of this master thesis is shown with the red oval[2].

The de.NBI Portal will allow scientists with less knowledge in informatics to use resources in an easy way. Furthermore, the portal will offer the possibility for easy cloud administration as well. Additionally, the portal should be integrated into the European life sciences network "Elixir" to support collaborations between scientists across the Europe.

1.2 Structure of this work

This work is divided into seven chapters. Besides this motivational chapter (1), next chapters (2-3) provide an overview over cloud computing, authentication mechanisms, and related work. Chapters 4-5 describe the analysis, organization, and design of the de.NBI Portal. The 6th chapter provides a brief overview of the implementation of this project. The last chapter (7), represents a short discussion section and outlook.

Chapter 2

Cloud computing and IAAS fundamentals

2.1 Definition of the "cloud" term

There are a lot of different definitions of the "cloud" term regarding the specific context. Within the media, the "cloud" is widely provided as a form of service or an internet site/portal which enables the user to solve everyday problems. Those definitions are not always correct and the meaning of the "cloud" is often misused as a buzzword for marketing purposes. The National Institute of Standards and Technology (NIST) defines essential characteristics of the "cloud computing", which comply with the definition of the "cloud" as well [10]:

- The cloud is an *on-demand* and *self-service* construct. The user of the cloud can decide for himself when to use the service and how many compute resources to use. There is no human interaction from the cloud provider needed.
- The cloud has a *broad network access* so it is able to access the services over standard mechanisms such as a computer or a smartphone.
- The cloud is using *resource pooling*, so the service is delivered by multiple physical and virtual resources (e.g., machines, data centers), which are dynamically assigned to the user. The user, on this occasion, has no control or knowledge about the underlying hardware is used.

- The cloud is *elastic and scalable* on demand. The computational capabilities can be dynamically increased or decreased by adding or removing additional resources.
- The cloud *service is measurable*. Cloud resources can be separated into different parts (e.g., storage, compute-power and per-user usage), reported and accounted transparently for the user and the provider.

Moreover, the cloud can imply different deployment models and provide different services [10]: one of the common known deployment models is a *Public Cloud*, which is open for everyone to use and is operated by a business or academic organization. In the business sector, the *Private Cloud*, which can be used exclusively by a single organization, is prevalent. The *Community Cloud* is similar to the private cloud except that it is used by a community of consumers that share similar interests. The combination of two or more cloud deployment models is called *Hybrid Cloud*, so the cloud infrastructure has its own, self-operated units which are bound together and can provide, e.g., load balancing between services.

2.1.1 Services provided by the cloud

In addition to the cloud computing definition and different deployment models, NIST distinguishes between the following services that can be provided by the cloud [10]:

- **SAAS** – Software-as-a-service is an application which runs on the cloud infrastructure. The application is accessible through the web browser or a separate client. The cloud infrastructure itself is not visible to the user and can not be controlled by the user.
- **PAAS** – Is a platform-as-a-service which allows the user to deploy his own application which uses the interfaces, frameworks, and tools given by the provider. The management of used software and interfaces is neither visible nor controllable by the user. Nonetheless, the user is able to manage and control his own application and the deployment configuration.
- **IAAS** – Infrastructure-as-a-Service provides fundamental cloud components such as storage, compute resources and networks. The user has no control over the

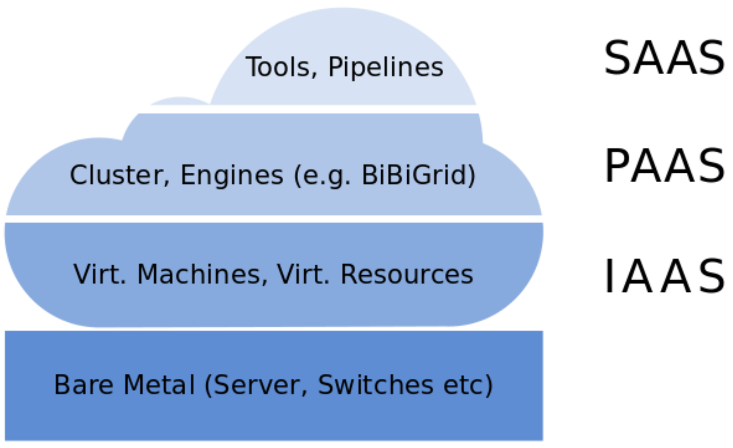

Tools, Pipelines
Cluster, Engines (e.g. BiBiGrid)
Virt. Machines, Virt. Resources
Bare Metal (Server, Switches etc)
SAAS
PAAS
IAAS

Figure 2.1: The abstraction layers of the cloud.

underlying infrastructure such as hardware components but builds upon provided resources his own platform and is able to control all applications, networks, and storage used.

The three service types provide different degrees of cloud abstraction (Figure 2.1) from the bare-metal server to the user. The most common variant in the consumer world is SAAS for private data storage such as “Dropbox” or “OneDrive”.

In the scientific area, many different applications and tools exist which are not able to serve as SAAS or be deployed upon PAAS infrastructure, since they provide mostly a command line interface (CLI). So there is a need for the possibility to integrate tools into the cloud infrastructure with as least effort as possible. Since the scientific tools mostly require a complicated installation process and have many dependencies [2], one way to bring that software into the cloud is to containerize the software and deploy it directly on the IAAS platform. From the user’s point of view, there are some differences starting the software on bare-metal machines or in the cloud.

2.1.2 Difference between IAAS and bare-metal deployment

One of the significant differences between IAAS and bare-metal is the abstraction of computing resources. The user of the IAAS is using virtual resources such as virtual machines, virtual networks, and virtual storage. The actual physical provision of resources is not evident for the user. It may happen, that in case of server maintenance the cloud administrator migrates all virtual machines from the server A to a server B. Since resources are virtual, the user will not notice the migration to another server. From a user's perspective, the virtual machine will run without any interruption. However, the case looks entirely different if the user uses a bare-metal server. Using bare-metal server implies the tight coupling to the resources of that specific server. For example, a memory upgrade is only possible with hardware changes and implies the shutdown of the server. The migration of the operating system and the software to another bare-metal server is a non-trivial task. Finally, by server maintenance, the downtime is unavoidable.

2.2 Authentication with single sign-on

In any application a sign-on is used to allow access only for specific members, to manage access rights and to protect contents of users. With the growing number of new software, there is a growing number of accounts and consequently a high number of credentials per user. To provide fast and easy access to the application without complex registration and additional accounts, the idea of single sign-on (SSO) was developed. SSO accelerates the user access to resources across different providers and outsources the user management to one central identity provider (IdP).

2.2.1 SSO authentication mechanisms

There are several different SSO technologies available which take different technological approaches.

- **Microsoft Passport** – It was developed and launched in 1999 and serves as the SSO for Microsoft services. Therefore Microsoft Passport is used as a login for MSN Hotmail, X-box Live or Zune. Microsoft Passport system is centralized and the authentication is done by the passport-server which provides the so-called PUID (passport user ID) of the user to the accessed service. Nevertheless, this system is rarely used in non-Microsoft services [12].
- **Shibboleth** – Is a technology which provides SSO on different websites across different Identity Providers (IdPs) and represents a distributed authentication system. To authenticate, the user has to login into the home-IdP which then gains user-data access for the service, that the user tries to access. Its infrastructure

consists of different services like IdP, service provider and the helper service (to identify users home organization) called "where-are-you-from" (WAYF) or discovery service (DS). [12].

- **OpenID** – Similar to Shibboleth it allows cross-site authentication. It generates like Shibboleth a digital identity which is provided to the user's requested service. It is entirely open source and reuses open source methods and tools like hypertext transfer protocol (HTTP), secure sockets layer (SSL), also known as transport layer security (TLS), and Diffie-Hellman key exchange [12].

2.2.2 Benefits and risks of single sign-on

The benefits and risks of single sign-on are more or less evenly balanced. The main advantage is that the user can log in once per session and reuses the session for all kind of services he accesses. On the other hand, it can be a dangerous if the account gets compromised and the attacker gains access to a significant amount of resources and information. Therefore, the login server is a bottleneck since the possibility to login depends directly on the accessibility of the identity service, which is out of control of the service provider [12].

Nevertheless, the chance of the password leakage decreases as the password is transferred only once per session. In addition, single password transfer decreases the probability of the phishing-attack because the user logs in on the different site than the service is located.

2.3 Accounting and supervision

The cloud allows the usage of many resources, which scale on demand. As described in 2.2.2, if the account gets compromised the offender would be able to access a significant amount of compute resources. The offender could use the exposed resources for instance to send spam messages, to perform distributed denial-of-service attacks (DDoS) or for Bitcoin mining among others. To prevent adverse exploitation, there is a high need for supervision and accounting mechanisms.

2.3.1 The purpose of accounting and resource allocation

In addition to detecting unfair resource usage, accounting allows to control the shared resources in the scope of the time course. The user or project of the cloud is holding an account and can buy or request credits. The credits can be used for any resources, and the user is billed in short time frames according to his resource consumption. The user alone decides which resources to use and when to use them. With the exceeding use of resources the user gets a negative balance. This can be prevented quickly after billing (e.g., by suspending user's VMs) or can be tolerated according to an automated procedure which follows a defined policy created by the cloud owner. An additional feature of accounting is the possibility of detailed reports about resource consumption, which can help a lot to improve the cloud service by adapting the provided service to the needs of the user. For example, by adding additional high memory resources to the cloud, if a lot of users need them.

2.3.2 Calculation of resource costs

Since the credits are closely linked to the resources and resource types, it is necessary to define the unit price per resource type for proper accounting. There are different models possible to calculate the unit price of a specific resource type. The price can be either static (same price all the time) or dynamic and can depend on the following factors:

- **service** – Amount of provided resources, e.g., 16 cores or 64 cores.
- **location** – Physical location of the hardware, e.g., Europe or North America.
- **personal** – Special prices and discounts, e.g., student price or educational discount.
- **time** – Time period of the actual consumption, e.g., night prices or day prices.
- **quantitative** – Set-up fee, e.g., for ready-to-go "Mesos" cluster.
- **price bundling** – Combined prices for special instances e.g., "Extra High-Memory 32xlarge" or "High I/O Extra Large" machines.

The very basic price calculation for compute instances in the cloud could be the price for one core provided; omitting the random access memory (RAM), disk space, network bandwidth and other factors. Exemplary for the 24 hours (H_{total}) of cloud usage with 32 cores (P_{total}) and (exemplary) 2 credits for one core per hour (F_{unit}) the total used amount of credits (C_{total}) results to:

$$C_{total} = H_{total} \cdot P_{total} \cdot F_{unit} = 24 \cdot 32 \cdot 2 = 1536 \text{ credits}$$

Since the cost is only tied to the number of the central processing units (CPU), it is likely that the user would claim all available RAM and additional resources which would probably lead to the occupancy of all resources.

2.3.3 Amazon Elastic Compute Cloud Accounting example

Amazon is alongside Google, Microsoft, and IBM one of the world's leading cloud providers. One outstanding product is the Amazon Web Services Elastic Compute Cloud (AWS EC2). The configuration of custom instance types is not possible in EC2 but there are approx. 80 different instance types [1] available that define different virtual machine (VM) configurations, in particular, the number of cores, size of RAM, storage, and network bandwidth. Each instance type has its on-demand bundle price which varies from the region (15 regions available[2]). The user is charged hourly depending on the used resources. Additionally, to the on-demand prices there is an option to reserve the instances for the long term and to pay for the resources upfront. The prices for reserved instances are lower than on-demand and vary depending on the reservation time: longer reservations result in lower prices. Besides the on-demand and reserved instances AWS offers a cheaper on-demand alternative called "spot instance". Spot instances use the reserve capacity of EC2 and get immediately shut down when they are needed elsewhere. Therefore, spot instances are only available for specific instance types and are suitable for applications with flexible start and end times.

[1]See `https://aws.amazon.com/de/ec2/instance-types/`

[2]See `http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html`

2.4 Cloud provider overview

There are various cloud providers on the market, as described in 2.3.3. All engines provide simple compute instances as well as storage. Each engine provides its own specific components like data analysis pipelines, neuronal networks, and graphics processing unit (GPU) instances. While the most cloud engines are a closed source and the engine holders act as cloud providers as well there is software available which is open source and enables the opportunity to create its own cloud based on its own hardware.

2.5 OpenStack - an IAAS provider

OpenStack is an open source software project consisting of different subsystems which allow to set up and to maintain an IAAS cloud infrastructure on its own hardware. The installations of OpenStack can vary and e.g., may be extended on demand using available components [9].

2.5.1 Structure of OpenStack

OpenStack consists of different subsystems (Figure 2.2) that can be connected together or used in another software. Communication between each subsystem is managed inter alia over representational state transfer (REST) application programming interface (API), but there are command line interface (CLI) commands and Python libraries available as well. From the beginning of OpenStack project, the number of components has increased and existing subsystems sometimes are split into two separate subsystems in future releases. The following list is a brief overview of some OpenStack components [9]:

- **Nova** – is one of the fundamental services in OpenStack and must be installed on all cluster compute nodes. It abstracts from the hardware components (CPU, RAM, storage, etc.) and provides the control over the virtual machines.
- **Neutron** – represents a networking subsystem of OpenStack, handles the networking between VMs. Neutron allows users to add their own virtual networks, attach floating internet protocol (IP) addresses and offers similar functionality.

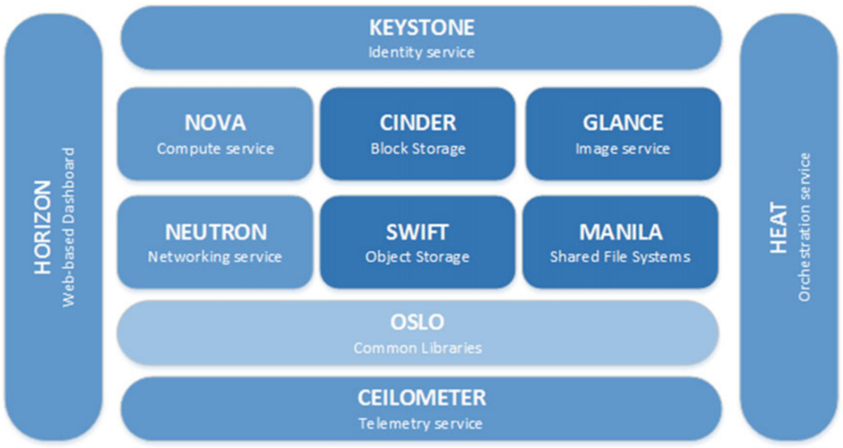


FIGURE 2.2: The components of OpenStack [9].

- **Keystone** – acts as an identification service, represents a centralized user directory and handles user authentication tokens and access control lists (ACL).
- **Glance** – operates as an image storage and runs a catalog of operating system (OS) images, VM snapshots and delivers backup functionality.
- **Cinder and Swift** – manage different storage types and allow users to create their own volumes, objects etc.
- **Ceilometer** – allows to collect, store, and monitor the cloud metrics. Additionally, Ceilometer has an option for billing the user.
- **Heat** – enables provisioning of the application lifecycle and automatically creates the cloud infrastructure based on templates.
- **Horizon** – provides a web graphical user interface (GUI) for user interaction with other OpenStack components and allows comprehensive administration as well.
- **Manila** – provides functionality for shared file systems.
- **Oslo** – delivers common libraries to the OpenStack components.

2.5.2 Sample Horizon use case

If the user intends to build his own PAAS or SAAS it is obligatory for him to use the OpenStack APIs.

The alternative way, which is especially used by inexperienced users to start the VMs over the Horizon web interface. Consider the following use case:

> The user aims to start a VM with a Linux OS and to access it through the secure shell (SSH). The user needs 16 cores, 32 gigabytes (GB) of RAM and 100 GB of disk space. First of all the user has to login into Horizon, navigate to the "Instances" menu and select "Start Instance". In the appearing pop-up the user configures the details (like RAM size etc.) of the instance. Usually it is enough to fill out the necessary information and click "Start Instance" (Appendix, Figure A.2). After the instance is scheduled, it is necessary to attach a "floating IP" to the running VM. Floating IP is an IP address from the open, public IP pool, that can be attached to VMs for e.g. external SSH access.

Besides the basic settings, an inexperienced user could accidentally change advanced settings of the VM configuration and the settings of Horizon Project as well. That could lead to the broken Horizon project environment and may affect VMs deployed by other project participants. Since users are able to start as many VMs in Horizon as the project allows, and projects are not monitored, there could be the tendency for unused, idling VMs over the time. Resources of the cloud would stay reserved and inaccessible for other projects. Such allocation of unused resources and other possible unintentional mistakes in Horizon lead to the need for additional user roles and implicit the need of proper user access control in the cloud environment.

2.5.3 Keystone in detail

As described in 2.5.1, OpenStack includes a subsystem for the user identity management called Keystone. Almost every action in the OpenStack ecosystem requires authentication (Figure 2.3).

Keystone is usually the first component which is installed when creating OpenStack cloud. It allows users to authenticate with several authentication types such as username and password or token-based authentication types [9]. There are various components in the Keystone database which can be configured to regulate access [9]:

- **Service** – OpenStack component like Nova or Glance

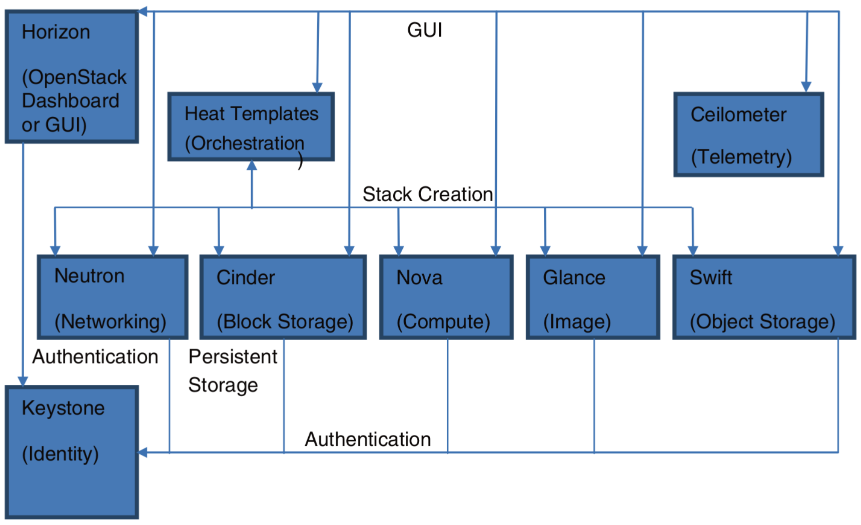


FIGURE 2.3: The dataflow in OpenStack. The most components require Keystone authentication[13].

- **Endpoint** – Access URL for the API e.g., `https://cloud.example.com:9292`.
- **Project** – The base unit of resource ownership, it contains VMs, networks, users, etc.
- **Domain** – Collection of projects which subdivide administrative permissions between domain administrators across the OpenStack.
- **Region** – Different OpenStack environments that use one Keystone service.
- **Token** – Authentication string for e.g., API calls.
- **User** – The user of the cloud / API. A user has a role and affiliates in one or more projects.
- **Role** – A set of allowed operations that can be assigned to a particular user.

On the technical side, Keystone uses a “MariaDB” / “MySQL” database [9]. Technically it is possible to mechanize the supervision of the Keystone by using the Keystone API components or simple database calls for data changing and thus automate and outsource the user configuration elsewhere.

2.5.4 OpenStack API usage example with Python

A user, who is familiar with the OpenStack API could set up his own cluster or just a single VM with a few lines of code. For that purpose, OpenStack provides language bindings and CLI tools that use OpenStack API in the backend. The commonly used bindings are Python bindings.

The first step is the import of the OpenStack library and adding the login information:

```
from openstack import connection


keystone_url =  "https://openstack.example.org:5000/v3/"
user_domain = "Default"
project_domain = "ProjectDomain"
project_name = "Testproject"
username = "sample-user"
password = "sample-password"
```

Listing 2.1: Setting up Python library to use OpenStack API.

The next step is the establishment of the connection via the `connection` object of the `openstack` library. The returned object is the actual OpenStack connection which allows access to further services like `compute`:

```
try:
    os = connection.Connection( keystone_url, user_domain,
                                project_domain, project_name,
                                username, password)
except Exception as e:
    print("Error:␣", str(e))
    exit(1)
```

Listing 2.2: Establishing connection to the OpenStack API

After establishing the connection, it is possible to access all OpenStack services through it. Consider the user wants to start a small VM; the flavor (machine configuration regarding CPU, RAM and disk space) and image are already known. The user generates a new key pair and adds metadata as key-value pairs to the VM object as well as the additional information for himself or other users.

```
vm_name = "Test␣VM"
image = c.compute.find_image("Ubuntu␣16.04")
flavor = c.compute.find_flavor("general.small")
key_pair = c.compute.create_keypair(name="MyNewKey")


metadata = {"purpose": "Litte␣VM␣for␣the␣\ostack{}␣API␣test",
             "contact_email": "john.doe@example.org"}


try:
    vm = os.compute.create_server(image_id=image, flavor_id=flavor,
        name=vm_name, key_name=key_pair.name metadata=metadata)
except Exception as e:
    print("Error:␣", str(e))
    exit(1)
```

Listing 2.3: Scheduling a small VM to OpenStack Nova

By calling `os.compute.create_server` the connection object requests from the Keystone the endpoint of the compute subsystem of OpenStack and schedules the VM. The response object is a `server` object of the OpenStack library, that provides a set of methods for further configuration (e.g., pause the VM) or attaching a floating IP. After the VM is created, it is possible to determine the IP address of the VM:

```
    if (not server):
        print("Error:␣", "VM␣not␣created!")
        exit(1)
    else:
        print("VM␣id:␣", str(vm.id))
        print("VM␣IP:␣", str(vm.access_ipv4))
```

Listing 2.4: Retrieving VM IP address for SSH connection

Knowledge of the IP address allows to establish a SSH connection with the new key pair for the (default) user (here: “ubuntu”). Furthermore, OpenStack Python bindings contain a lot of different methods that facilitate the detailed configuration of the VM, different networks, and storage to build custom cluster environment from scratch.

Chapter 3

Related work

3.1 Federated OpenStack

If all compute centers use OpenStack there is a the possibility to use federated keystone authentication [1] between the compute centers. Consequently one keystone instance would store and manage endpoints of all compute centers and hold all user accounts. The main benefit of this approach is the centralized management of all users. Each compute center would appear as a region [1], and the user would be able to decide to which region the VM should be scheduled to. This approach involves many disadvantages as well; the main disadvantages are:

- **Own IdP** – Prerequisite for using federated Keystone is the need of its own IdP. Since de.NBI Cloud should be integrated into Elixir and be accessible for scientific users, the usage of (inter)federated educational identity providers like eduGAIN, which is university-related, should be aimed for.
- **No Abstraction** – Using only the centralized authentication mechanism is not enough to simplify the usage of the cloud, because the user faces with a complex front-end, which is unsuitable for inexperienced users. Furthermore, the project management is either done by the domain administrators or OpenStack administrators.

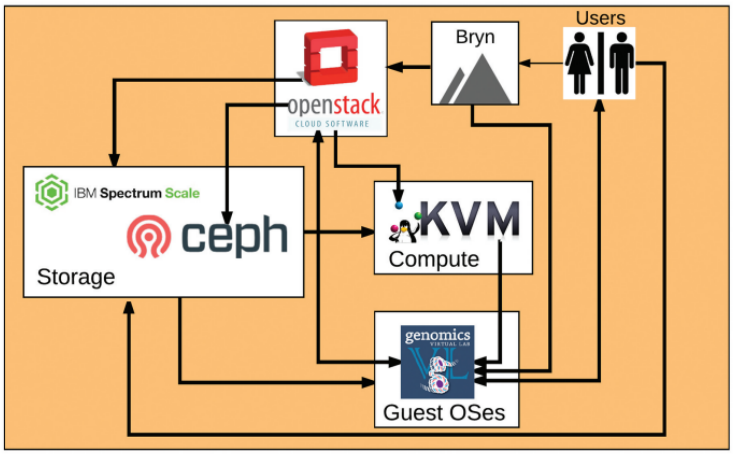


FIGURE 3.1: Bryn is the layer which provides abstract OpenStack access to the user [7].

Moreover, the federated OpenStack plugin is maintained by the community and can implicate lack of proper maintenance and bug fixes. Integration of Elixir IdP is problematic and would present difficulties. Additionally, tight coupling to OpenStack and maintaining similar version and configuration is impossible for autonomous compute centers. Therefore a platform which abstracts from OpenStack is of high interest.

3.2 CLIMB – Bryn

Four UK universities founded a joint project called "CLIMB" (Cloud Infrastructure for Microbial Bioinformatics) in 2014 [5] in order to provide computational resources to the microbiologists. CLIMB utilizes OpenStack as a cloud engine. Three of four universities run the cloud hardware (Birmingham, Warwick, Cardiff)[1] and maintain the same hardware and software configuration [5]. Currently, the CLIMB cloud contains 7680 vCPU Cores and 78 TB total RAM.

To access the web panel and configure VMs, the user must register using UK academic credentials [6]. After that, the user can start a generic pre-configured VM or access "a dashboard, similar to that provided by Amazon Web Services"[7]. The latter option is reserved to expert bioinformaticians [6]. The layer between OpenStack and the user is called "Bryn" (Figure 3.1). For fair usage of the resources, to every user a quota limit is assigned. Otherwise, the compute cloud is free of charge for academic UK users [6]. This approach for providing cloud computing resources to the biologist is entirely new

[1]See the source code of Bryn `https://github.com/MRC-CLIMB/bryn/tree/master/brynweb`

(CLIMB launched 2016 [5]) but using Bryn for the de.NBI cloud brings some constraints and limitations.

- **No external IdP** – The user registration in Bryn is tied to the local user database[2]. Bryn uses Django framework as a back-end, and the registration is restricted to the principal investigators of the UK academia. That obviously excludes the application of Bryn besides the UK. Even if Bryn were adopted for de.NBI , the handling, and updating of software would take additional time. For example members which left the institution must be removed by the CLIMB administrators.
- **OpenStack is a "must"** – Bryn works only with OpenStack. The codebase of Bryn[3] utilizes its own tenant model and OpenStack-client in a very interwoven manner. To use Bryn with another cloud provider besides OpenStack would result in massive code changes.
- **Hardcoded regions** – the compute centers are hardcoded in the Bryn source code. Adding new regions would entail code changes.
- **Storing of unencrypted passords** – the passwords for the compute centers are stored without any hashing or encryption.

However, the most significant limitation of Bryn is the fact that the whole system is tied to the OpenStack API v2, which is partially deprecated. CLIMB itself uses OpenStack release *Kilo* which is five releases behind the current version (Pike). In the case of CLIMB, where the count of the compute centers and compute resources is quite manageable, Bryn may be the right choice; but for the de.NBI infrastructure with currently five compute centers (with the prospect to serve in the next future over 15.000 cores), the use of Bryn is inappropriate. For that purpose a new modular and abstract platform should be developed.

[2]See `https://raw.githubusercontent.com/MRC-CLIMB/bryn/master/brynweb/brynweb/settings.py` for Bryn configuration

[3]See `https://github.com/MRC-CLIMB/bryn/tree/master/brynweb` for complete source code

Chapter 4

Analysis of the de.NBI project

4.1 General purpose and core functionality

Every compute center in the de.NBI Cloud runs its own OpenStack installation. Separated OpenStack installations allow registered members to use the cloud resources only of a specific compute center. Due to the separation there should be a possibility to create global compute projects and schedule them to one or multiple compute centers. Furthermore, there is a need for different roles with different access and administrative rights. Every user in the de.NBI network should be able to start VMs, perform computations, and be a member of different compute projects. There is a necessity for a compute project manager who has the authority over the group and its members. On the other hand, the administrator of the compute center should have control over the project and compute resources which are assigned to the specific compute center, to prevent misuse or to solve resource conflicts. The administration unit or cloud government is mandatory as well since it has the decision-making powers for project application acceptance and compute center assignment for existing projects.

4.2 Use cases and studies

4.2.1 Basic role categories

We distinguish between four basic roles that are present in the entire federated de.NBI cloud:

- **Administration Office (AO)** – The administration office is based in Bielefeld and consists of a few persons. AO internally has different coordinators such as the cloud governance. Administration office is responsible for managing the resources and credit allocation to the de.NBI projects. Members of the AO have information about every compute center in the cloud and over distinctive features of each compute center.
- **Compute Center Responsible (CCR)** – Each compute center has its own support team which monitors and administrates the local cloud installation and moreover maintains the cloud infrastructure of the respective compute center. CCRs are familiar with the cloud technology and especially with the unique features of their compute center. Additionally, the CCRs may use compute resources exclusively for the workgroup of the institution the compute center is located in. Another task, is the compute center supervision and controlling to prevent unfair resource usage.
- **Principal Investigator (PI)** – A PI is the manager of the compute project in de.NBI . New projects and additional credits are handled by the principal investigator. PI is responsible for assignment of new users and for utilizing resources of the projects as well.
- **User** – It is the most common person type in the de.NBI cloud. In most basic cases the user would like to start and stop VMs, configure and execute software that the user is familiar with and experiment with new software. The user relies on the knowledge of the tools he is using and additionally on advises of colleagues; possibly on rd party tutorials and How-To's from elsewhere. The user will access the site mostly form a PC and is familiar with the internet and typical control elements of the web page. Besides the normal user there exist special user groups which differ in knowledge and in computational tasks they want to perform.

- **Course Participant** – A course participant is a person who is not registered in de.NBI and whose usage of the cloud is reduced to the scope of the course. The technical knowledge of the participants may vary and not every user is familiar with the command line or Linux systems in general. To use the de.NBI cloud the user needs an Elixir account and the registration in the de.NBI . For registration, a web browser access is required. After the contact with the de.NBI , the course participant, may use the cloud for further projects besides the course.
- **Experienced User** – This user type has advanced knowledge in bioinformatics and has experience with Linux systems and with command line operation as well. Besides the calculations with existing tools, an experienced user tries to write his own tools or combine tools into a pipeline. Furthermore, this type of user is actively experimenting with new software, tools, and packages. Using the de.NBI cloud the user could configure its own VMs and environments and use powerful and long run VMs as well. The browser of the user can vary widely and contain such browsers as *Vivaldi* or *Midori*.
- **Power user** – This type of user is most commonly a biologist or informatician which uses a massive amount of bioinformatic tools and produces own software. This user is experienced and very familiar with Linux, networking and cloud computing. The general task of the user is to try out new software or to perform calculations. Furthermore, the user may want to automate the data analysis and would like to create a custom cluster over API with custom scripts or build upon existing PAAS. Since the user is experimenting with new software, he is massively using tutorials, How-To's and may need specialized hardware or technology for calculations. This user will use OpenStack Horizon or OpenStack API most of the time.

However not every user can be associated with a single user type. Sometimes the user is a combination of one or more types and pursues different computational targets. For example for the tool A for genomics the user would act as a course participant, but while developing tool B for metabolomics the user acts as an experienced user.

4.2.2 User stories

Since user is someone who consumes cloud resources, the primary task of the user is the reservation and usage of compute resources. In addition, the user should be able to manage own data storage, create and deploy his own compute cluster and manage his own project memberships.

The following user stories capture the functionality that needs to be implemented. User stories can also help to develop a process and answer questions like "Which type of actions should be a user able to do?", "What kind of permissions does the person have?", "What is the default *workflow*?", and "Are there any special *workflows*?".

Administration Office user stories

User stories of the administration office (AO) primarily relate to tasks of the cloud governance and management. The AO

- determines who is allowed to use the cluster.
- determines who is allowed to add further users to the cloud.
- must have an overview over all centers, used and free resources.
- should know which centers are operational.
- should know who is using the de.NBI cloud and how.
- should be able to remove users and also restrict resources.
- is able to approve, assign projects to a PI, remove, change them (in case of assigned compute center), add additional credits and users that belong to a project.

Compute Center Responsible user stories

The user stories of the compute center responsible (CCR) describe in particular the management and controlling of the local compute center and using his resources for his own projects as well. The CCR

- should be able to monitor the resource consumption of any machine in the entire de.NBI cloud allocated by a person related to the compute center.
- should be able to allow other persons to use the de.NBI cloud and allocate resources and determine new users.
- should be able to upgrade the system independent of the administration office.
- should be able to restrict the part of the cloud which used as part of the de.NBI cloud at any time.
- should be able to add and remove users.
- should be able to start and stop all machines in the compute center.
- wants as much as possible introspection of his own system. (e.g. "Who is consuming the resources?", "How much is used?")

Principal Investigator user stories

User stories of the principal investigator (PI) are basically about the user management of projects, application submissions, and project management. Furthermore PI can consume the resources as well. The PI

- maintains multiple projects that need a different kind of resources.
- should be able to add users to his projects.
- should not use or delegate more resources that are assigned to his project.
- should be able to ask for more resources that can be increased.
- should be able to start and stop all machines administered by the members of his project.

- should be able to monitor the current resource consumption.
- should be notified if no more free resources or credits are available.

Chapter 5

Project setup and design

5.1 Development workflow of de.NBI Portal

The project organization is significant in every stage of the project lifecycle since it helps to keep the quality level high and avoid mistakes before the production release. Moreover, the proper setup helps to find the source of mistakes and eliminate them quickly. Additionally, with the right documentation and deployment process, the novice contributors become acquainted with the project faster. The basic organizational mechanisms are project versioning, project documentation and process modeling of software interactions.

5.1.1 Versioning

While writing code it is necessary to make changes and test out different techniques or possibilities until a feature of a software gets done. When collaborating with multiple contributors, there is a need for code organization, which help to split the software into parts so that each developer works independently.
There are different versioning systems available. This project utilizes `git`[1] as the versioning system. Each part of the project is developed independently as a feature branch. The branching structure should consist of the following branches (inspired by the "Successful git branching model[2]"):

[1]See `https://github.com/deNBI/` for repositories
[2]See `http://nvie.com/posts/a-successful-git-branching-model/` for more information.

- **`master`** – the master branch represents the current production-ready branch which instantly can be deployed.
- **`dev`** – the development branch consists of the development code which is not ready for release. Ready feature branches are merged into it.
- **`feature/`*`(feature name)`*** – the branch in which new features are developed. After the feature is ready and tested, it should be integrated after code review via the merge request.
- **`hotfix`** – this branch allows fixing bugs found after the feature branch is already merged.

Each commit (saving of the current state as a version) must have a message which describes changes. There are different styles and possibilities how to write down messages. The easiest way is to write every message as a small text describing the changes made. However, this approach can get chaotic and complicated over the project lifetime, so it gets hard to find a specific change. A more clear and structured approach is to use a convention of commit messages. Structured commit messages make it is easy to find changes or impacted software parts in the future. Another advantage of this approach is the ability to automatically generate a summary of the changes made. The development workflow of de.NBI cloud portal uses commit message convention similar to the one of the AngularJS convention.[3]. The convention contains different commit types which can be written in the same message. The commit message format has the following structure:

```
commit-type(impacted-software-part) changes made
```

Let's consider the following example: the user branched a new feature branch, added a new logo in the HTML template and fixed an issue with the database model. The resulting commit message would be:

```
feat(template) added new logo to the template.html
fix(model) fixed name issue #42 in the user.model.py
```

[3]See `https://github.com/angular/angular.js/blob/master/CONTRIBUTING.md#commit` for more information.

Additionally, the issue number of the issue tracker can be provided as well as in the example above (`#42`).

Each release and patch must have its own version number. For that purpose there is a semantic versioning model[4] used. The information that the version number provides is which kind of changes are made with every new release. The composition of the version number is `MAJOR.MINOR.PATCH`. Besides the specification of each component there are three uncomplicated, but essential change types, which are stored in the version number:

- **`MAJOR`** – the change in this number points at changes which are not compatible with fewer releases.
- **`MINOR`** – the change in this number points at new functionality which is backward compatible.
- **`PATCH`** – the change in this number points at backward-compatible bug fixes in the software.

For example when the API software gets an additional model, which does not affect other functions of the software, it would be the change in the minor number. In opposition, the switch from one technology to another, which breaks all previous functionality in API calls, would be a major change.

5.1.2 Documentation

The project has to be documented in order to allow the reproducibility of code deployment and for understanding the code. The benefit of the current git repository hoster – GitHub[5] – is the possibility to show markdown (a lightweight markup language) rendered directly in the web browser. So the documentation for the deployment is stored in the code repository as markdown files as well. This kind of documentation provides the benefit of the versioned documentation in every release. No matter major or minor release, it will include the proper supplying documentation. The other features of markdown are the possibility to add code listings and automatic generation of a website containing project documentation using special conversion software.

[4]See `http://semver.org/` for more information.
[5]See `https://github.com/deNBI/`

5.1.3 Process mapping

The process is derived from the use cases and possible communication events between user-user, user-software, and software-software (machine-to machine (M2M) communication). While planning the software architecture it is important to keep the procedures abstract without implementation characteristics as otherwise the overview of the designed processes can get lost. For implementation-specific process modeling there is for example unified modeling language (UML) available which can help to specify, construct, visualize, and document models of object-oriented software systems [4]. The UML specification is developed and maintained by the Object Modeling Group (OMG). This group developed other formats for modeling as well, like the BPMN (Business Process Modeling and Notation). BPMN is a standard for graphical (flowchart) process representation which is defined by the collection of rules and specific components. This standard has different types of diagrams, one of the most common is the process and collaboration diagram, where the process is built on activities, interrupts, and parallel flows between the process-participants. The main benefit of using BPMN in this project is for easier communication between all stakeholders of the project including non-developers. The core processes are modeled in BPMN and the process modeling should be used for new components in the future as well.

5.2 Component design

5.2.1 Components architecture overview

Analysis of the related work (Section 3) suggests that the components, as well as the code structure should be modular. A major advantage of this that the development of each module can proceed independently. As for the deployment, many components have to be installed and set up. During the lifecycle of the project, updates are necessary. In that case, all dependencies must be reviewed before the production software gets replaced by the newer one. To satisfy that requirement all components should be split up into separate modules (cf. Section 2.5.1 - OpenStack subsystems) and run independently as far as possible to provide sufficient options for better development.

5.3 Shibboleth and Elixir AAI

5.3.1 Single Sign-On with Shibboleth

Shibboleth is among other systems (see Section 2.2) a software solution which can provide single sign-on (SSO) and allows (besides the federated authentication of the user) the implementation of authorization constrains as well[6]. Shibboleth is based on the security assertion markup language (SAML) and consists of multiple, loosely coupled components. The basic components of Shibboleth are:

- **Service provider (SP)** – that software runs on the site of the server operator, which wants to offer SSO to the users of the service.
- **Discovery service (DS)** – allows the user to choose the federation for the authentication. This can be a home institution for example.
- **Identity Provider (IdP)** – is a service which runs in each federation and maintains user lists and attributes. IdP can use for example "LDAP", "ActiveDirectory" or similar technology as a kind of user list.

The SP must be configured to use the proper DS and IdP. The configuration exchange happens through metadata files which are unrestricted and available. When the user tries to access any resource, which requires authentication (Figure 5.1 - (1)), the SP detects missing *SHIBSESSION* cookie and redirects the user to the discovery service (2) of the federation. This service allows the user to choose his home institution where the user holds an account. After the user selects the proper institution the DS redirects the user back to the proper IdP login page (3). On the institution login page the user provides the correct credentials and the IdP authenticates the user (4) utilizing the user database (e.g., LDAP or AD) and redirects to the site of the service the user tried to access with an additional token as a parameter (5). This token allows the SP to request the attributes of a specific user directly by the IdP (6). After the attributes are provided to the SP (7), and the user is authenticated, the service can check the authorization of the user optionally to access the content (commonly with attributes from the IdP) and then deliver the requested content to the user (8).

[6]See `https://wiki.shibboleth.net/confluence/display/SHIB2/Home`

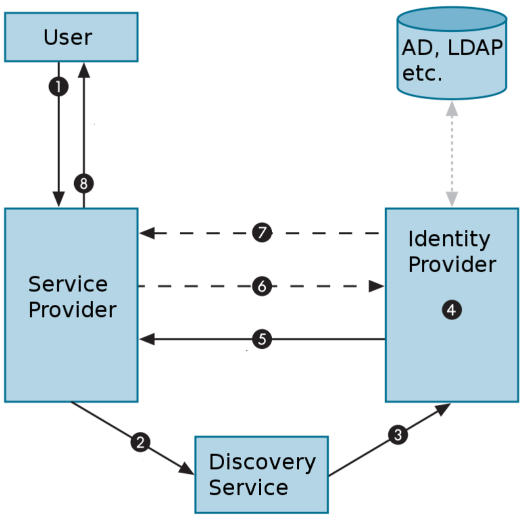


FIGURE 5.1: Shibboleth authentication flow. (1) Unauthenticated content request, (2) Redirect to the DS and IdP choise, (3) redirect to the IdP, (4) Authentication in IdP, (5) Redirect with the token, (6) attribute request with the token, (7) attributes provision, (8) content delivery to the authenticated user. [8], (*adopted by the author*)

This kind of authentication allows also detailed permission handling since the IdP provides different attributes to the user like member status, user role, affiliation. Academic universities and research institutions can be aggregated into a federation and thus Shibboleth offers appropriate authentication mechanism for that purpose.

Excursus – eduGAIN

eduGAIN [7] is an interfederation service for research and education, which consists of many institutions and countries across the world. To take advantage of the eduGAIN interfederation service, university must provide a SAML based authentication mechanism and sign a policy which is universal for all eduGAIN members. The current membership status is shown in Appendix A, Figure A.1

5.3.2 Elixir AAI

Elixir provides an infrastructure for the authentication and authorization of users in the life science context. This infrastructure (Elixir AAI) is using eduGAIN IdPs as an authentication service but also offers the login with Google, Linked In and ORCID (Organization, that provides a persistent digital identifier for every researcher[8]).

[7]See `https://www.geant.org/Services/Trust_identity_and_security/eduGAIN`
[8]See `https://orcid.org/` for details

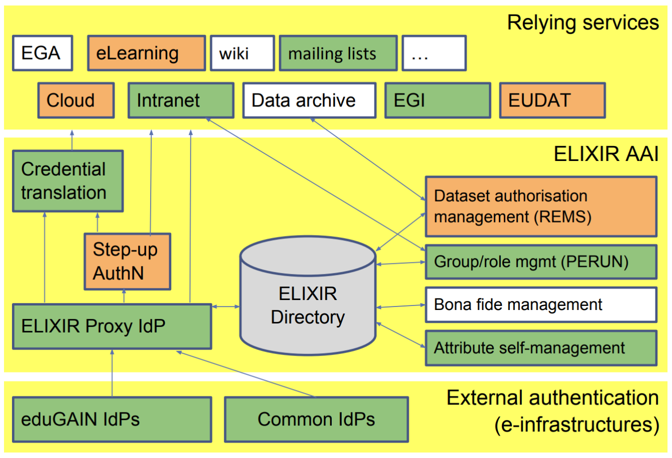


FIGURE 5.2: Elixir AAI structure. Components, which are already in production are shown in green. Borrowed from `https://indico.cern.ch/event/605369/contributions/2440443/attachments/1414958/2165898/ELIXIR_AAI_udpate_-_FIM4R_2017.pdf`

To combine all authentication methods into one and extend the functionality, Elixir AAI uses a so-called *IdP proxy* (Figure 5.4). Internally Elixir contains additional components for user account database (Elixir Directory) and Users Role management (Perun). For the service providers, Elixir AAI acts as a single IdP service.

5.3.3 Elixir Authentication

The authentication process of Elixir is slightly different from the usual federation login with Shibboleth, shown in Section 5.3.1, in the way Elixir acts as a proxy and still holds its own user information in the Elixir directory together with a role and group management system. The steps of the login with Elixir are the following (Figure 5.3):

- **1** – The user requests the site content
- **2** – SP detects missing SHIBSESSION cookie and redirects the user to the where-are-you-from (WAYF) discovery service. In that case, Elixir offers options for eduGAIN, Google, Linked In and ORCID.
- **3** – The user chooses the IdP to log in
- **4** – After the user made a choice, he is redirected to the IdP of the organization through the IdP proxy to log in for the Elixir AAI.
- **5** – The IdP requests login credentials from the user.

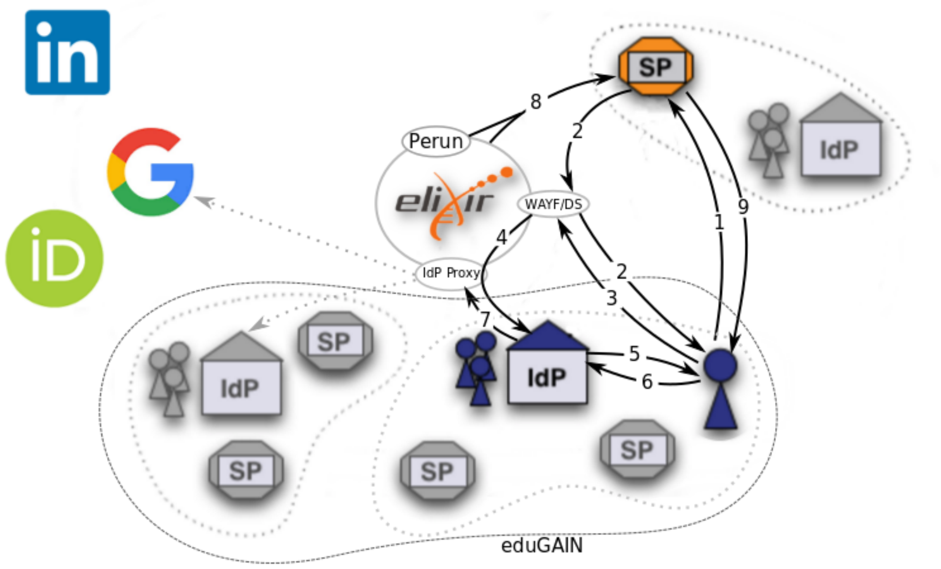


FIGURE 5.3: Elixir AAI authentication flow. The user authenticates first against Elixir (1-7). Elixir authenticates user for the service and delivers additional attributes (8). See Section 5.3.3 for detailed explanation. [9]

- **6** – User logs in using the credentials of his account
- **7** – IdP authenticates the user against the Elixir AAI and redirects him back to the Elixir AAI proxy.
- **8** – After successful authentication, Elixir authenticates the user for the service that the user tries to access. Additionally, Elixir gathers attributes like affiliation or group membership from its own Perun database and provides the access token to the SP.
- **9** – After SP received the attributes, authenticated and authorizes the user, it delivers the desired content to the user.

5.4 Elixir's Perun

Perun is the management software for user roles and groups inside the Elixir AAI. Perun distinguishes between two roles: *manager* and *member* that can be applied to different organizational units:

- **Virtual Organization (VO)** – the primary organizational unit that holds groups and members. A member can participate in different group memberships or remain without a group but then the user is still a member of the VO and the default group *members*. The manager of the VO can add new VO members, create, modify and delete groups, appoint group managers and configure VO settings and attributes.
- **Group** – this is an organizational unit of the VO. Group members can only see the members of the certain group but not all members of the VO. Group manager is able to add/remove members of the group and edit group memberships as well. Finally, a user can be a group manager in different groups at the same time.
- **Facility** – this unit is on the same level as the group, but does not belong to the VO. The facility is a stand-alone organizational unit, which manages resources and services that can be used by different VO groups. For that purpose facility contains multiple VO-related *resources* which, in turn, contain a set of assigned groups and a set of assigned services. Each resource is able to propagate resource attributes by using so called *propagation endpoints* to which changes of the Perun database according to the services can be pushed. The facility is able to have managers, but no regular members.

Furthermore, the service mentioned above has a set of attributes (key-value pairs) that either depend on the VO / Group / Member or remain independent. Attributes can be declared as numbers, strings, arrays etc. These keys and values are distributed to the propagation endpoints of Perun.

In order for de.NBI users to be able to use the authentication mechanism and for the portal to be able to use Elixir components or Perun, users must be part of the de.NBI VO. Manager roles of the VO are reserved for the administration office. Each compute project is represented as a VO group and the group manager represents the principal investigator of the workgroup. Every facility should represent a compute center and the provided services are compute services of each center. The manager of the facility is the CCR of the compute center and can manage groups which are assigned to them.

For storing restrictions, quotas and future credits, we created additional attributes. These attributes relate to VO / Groups and Facilities. Some of them are:

- **denbiCreditsCurrent** – (Integer) represents current credit amount of the Project. This attribute is group related, facility independent and is propagated to all endpoints (compute centers).
- **denbiDirectAccess** – (Boolean) shows if the group has access to Horizon and the OpenStack API of each compute center. This attribute is group-related, is facility dependent and is propagated to each compute center individually.
- **denbiCreditsHistory** – (Map) stores credit usage history of each user for documentation purposes. This attribute is user related, facility independent and not propagated since it is accessed over Perun API (see Section 5.4.2).

5.4.1 Propagation service

The propagation service of Perun allows adding subscribers to the Perun system, which get notified when the subscribed services have changed. Currently, data transport is realized by an SSH login from the Perun system to the endpoint with public key authentication (PKA). This service transfers a JSON object of the actual state in the "System Cross-domain Identity Management" (SCIM[10]) format to the target system. After that, the endpoint client triggers a set of Shell scripts, which represent the updating mechanism of the local database or service. In the future, propagation over HTTPS and Email is planned. The propagation service, which can be hooked to group memberships and attributes should be used to update the information regarding projects, users, and permissions inside the de.NBI compute centers. The propagation endpoint at the site of the compute center should be represented as a *Portal client.*

[10]See `http://www.simplecloud.info/`

5.4.2 Web API and Service User

Although Perun provides a web GUI, it is complex and provides many possibilities for configuration. On the back-end side, Perun GUI uses a JSON RPC API which can be reused for the implementation of the de.NBI Portal. For that purpose, the JSON RPC endpoint allows CORS requests from the specific de.NBI domains only.

In some cases it is necessary to perform automatic requests with the rights of a VO manager role. For example to update the credit balance of a project.Because of this use case a service user in the Perun system was created. This user is affiliated with managers of the de.NBI VO, but does not need a federated login and can access the API without a browser (e.g., from a script) using the "BasicAuth" authentication mechanism.

5.5 Portal core

5.5.1 Purpose

The primary task of the Portal core is to tie all components together by providing its own API and to build an abstraction layer of all components. Moreover, the core component is managing incoming project applications and user authentication using SSO.

5.5.2 Basic components

The main component is the API engine which handles and stores requests regarding project applications. Additionally, it sends a request to Perun when the project has been approved.

5.5.3 API design

The API for creating and updating Project applications should be able to manage the following objects:

- **project applications** – a suitable model for projects submissions should be designed. The model should be user-related and contain at least values for requested

RAM size, number of cores, disk space and status of application (approved or declined).

- **application status** – This small model should store different statuses of project applications (e.g., approved, declined etc.) and be dynamically adaptive regarding the status name or status ID if the application process changes.
- **special hardware** – This model should store names of the particular hardware types (e.g., GPUs) which can be additionally requested. The name or ID should be dynamically adoptable when the hardware of compute center changes.

5.5.4 Session handling with Shibboleth

To manage users and permissions, the Shibboleth service provider, that works with Elixir AAI should be configured. The minimum requirement for authorization in portal is the unique user ID. This user ID should be automatically provided by the IdP when the user logs in. Furthermore, access to the content of the portal should only be possible with a valid Shibboleth user session.

5.5.5 Perun communication

Since Perun offers a set of attributes and an API (see section Section 5.4), a connector for the proper Perun communication should be developed. This connector should be able to perform JSON RPC requests and process the answers accordingly. A further task of the connector is to authorize certain users for requests, which otherwise only stand for VO or group managers.

For example requesting the current credit balance of a group's account is possible by group managers and VO managers only. If the request is called by the group member, connector must first check the group affiliation of the user. When affiliation check is passed, balance is requested from Perun by the connector (with the help of the machine user which acts as a VO manager). Afterwards the credit balance of the group is received it is forwarded to the user.

5.5.6 Compute Center Connection

Besides the communication with Perun, the Portal core handles user requests addressed at the compute centers. This is the case when the user wants to start a virtual machine or to attach a volume. The primary task of the Portal core is to verify the request, to forward it to the corresponding compute center and to notify the user about the result. Additionally, Portal core should be able to generate SSH key pairs for users, which do not provide a public key when requesting a new VM.

5.6 Portal webapplication

5.6.1 Purpose

The user interaction with Portal components should be as easy and intuitive as possible and run on all platforms. To keep it as simple as possible a web browser can be used to render the view. Web browser specific technologies like HTML5 and CSS3 allow a flexible and adaptive design, which run on mobile devices and tablets.

5.6.2 Schematic overview

Besides the Portal code, the web application should provide additional interactions directly with Perun and Portal core (Figure 5.4).

5.6.3 Interactions with Perun and Portal core

Another essential task of the web application is the communication with Perun using the web API. To handle both, Portal core and Perun, there is a need for custom connectors. Portal webapplication will reuse already established SSO session and the calls to the API and Perun are always authenticated with the current user. This reduces the risk of unauthorized access. Furthermore, the web application should use the CSRF -token (cross-site request forgery) mechanism for the all requests in order to prevent possible cross-site attacks.

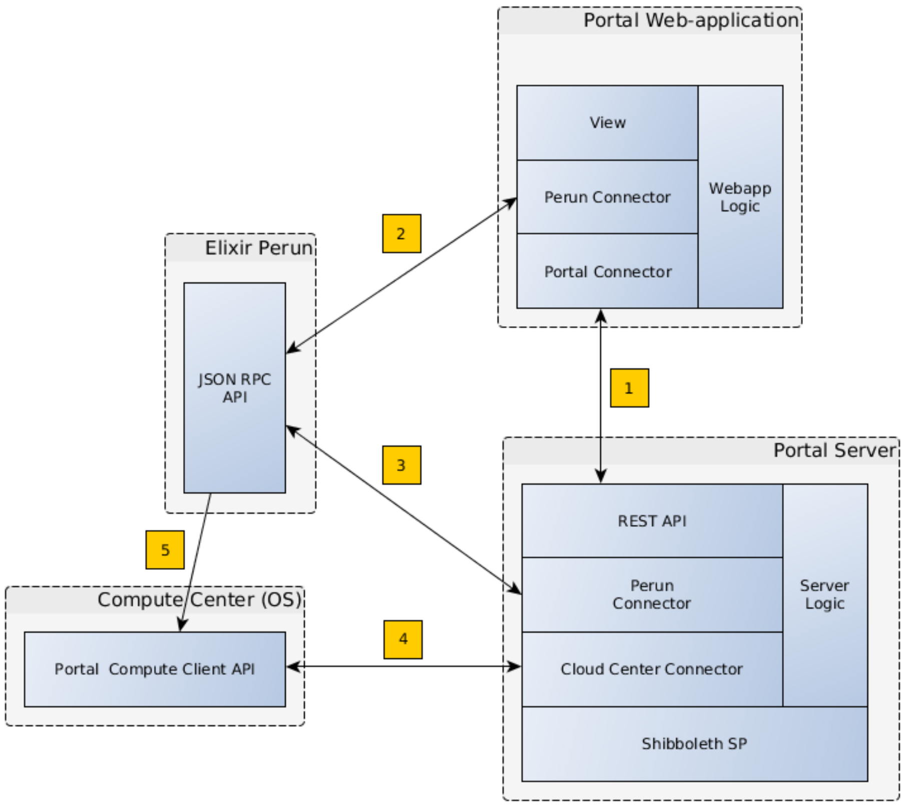


FIGURE 5.4: The Portal web application should be able to communicate to Portal (1) and to Perun (2) as well. The Portal core (server) should be able to communicate to the Portal clients (4) and to Perun (3). Additionally Portal client is acting as an endpoint of the Perun propagation service (5)

5.7 Portal client

5.7.1 Purpose

On the server side of compute centers, there is a need for a connector which can handle virtual machine scheduling, volume handling and offers an API for the M2M communication with the Portal. For now, all compute centers use OpenStack (see Section 2.5) as the cloud software. Unfortunately, this type of software tends to experience some changes over time, whereas an additional abstraction layer to the Portal core over the Portal client would ensure that API calls are stable and functional. An additional advantage of the abstraction is the specification of the API. The defined API interface does not determine the concrete implementation of the Portal client. For that reason the implementation can vary from compute center to compute center. This approach makes different VM handling techniques on the back-end possible. For example, a ticket system notifies the compute center administrator to start the VM manually. As long as the API is implemented correctly, the rest of the Portal will be able to communicate with the compute center, no matter how the back-end of the Portal client is implemented.

There is also a need for an API that abstracts from the cloud provider and provides an interface for the Portal core. The interface should be implemented by each compute center, to fulfill every compute center's special features and VM scheduling flows. For that purpose, there is a need for a proper specification (endpoints, incoming and outgoing data types and structures) of the API design and a reference implementation of the interface.

5.7.2 Perun endpoint

Another important task of the Portal client is the provision of the Perun propagation endpoint. The Perun system is able to aggregate pre-defined data sets as soon as their values change and to automatically propagate them to a list of endpoints. In that case, it is possible to propagate any relevant changes in Perun (which are made via Portal client or Portal core) to the according endpoints. In this way, the member lists of all groups can be kept up to date at all compute centers. Therefore, when a user becomes a member of the project, which has permission to direct access to the compute center "A", Perun will automatically notify this compute center and send the new user list for that project. The Portal client receives the information and updates the internal database of the allowed direct-access users (or notifies the administrator - depending on the implementation). After that, the user will be able to access the compute center "A" directly. Currently, the Perun propagation service implements the push mechanism utilizing remote command execution via SSH, but in later releases, an update over HTTPS and Email should be possible.

5.8 Cloud center VM provisioning

5.8.1 Overview provisioning techniques

When the compute center starts a Virtual Machine for the user, the user should have access to the VM over SSH. The provisioning of the VM to the user can be done in different ways. Some of the main techniques for doing this are:

- **Public IP** – The assignment of public IP addresses allowing a simple and easy way of providing access to the VM for users. However, when hundreds of users start multiple independent (non-cluster) machines with public IPs, the public IP pool of the compute center can get exhausted very quickly. To get more IPv4 addresses each compute center has to request a new block by the corresponding authority (e.g., HRZ or DFN). compute centers which are registered as LIRs (local internet registry) can request new blocks by RIRs (regional internet registry), such as RIPE NCC in Europe, directly. In both cases, the new IP block assignment request is quite a time-consuming task.

- **Jump host** – a jump host represents a server or a set of servers that connect an internal network, which uses RFC 1918 addresses, with the internet. This technique is usually used when the server in the internal network should not have access to the internet, but the user should have the possibility to access the network from the internet. The benefit of a jump host is that only one public IP address is needed to give the user the possibility to access all machines in the network. For that case, the user logs onto the jump host machine and from there connects via SSH to the machine of interest. One main problem occurs when data transfer is needed or some machines should provide an open port to the internet. Nevertheless, those problems can be solved using SSH tunnels.

- **Port forwarding / NAT** – NAT is similar to the jump host in the way that an additional server with connections to the internal network and internet is needed. However, the server acts as a D-NAT (destination network address translation) server which forwards its own port to a port of a specific machine in the internal network. The benefits of using NAT are the IP address savings and the possibility of the direct access to the machine. The drawback is that the ports the user has access to, must be defined. Consequently the user is not able to access any port on the machine until there is a NAT rule for it. In addition, no standardized "well-known" ports can be taken and the user must be notified which port number to use.

5.8.2 Port Mapping

According to the defined user roles (Section 4.2.1), the main use of the single virtual machines is to test new software or to perform small to intermediate computational tasks. Therefore the main requirement for the VM will be the possibility to upload data directly to the VM and to access the virtual machine over SSH or via web browser. The requirement points out that the user will only need a small number of ports and thus provision over NAT would be the most appropriate type.

The private internet address ranges are specified in the RFC1918 [14] and can contain up to 2^{24} addresses. The most common type is the `192.168.0.0/16` address range, which contains 2^{16} addresses. The address range can be reduced using subnetting. In the case of the `/24` network it would allow up to 254 hosts, and the `/22` network can contain up to 1022 hosts. The NAT host would use either the dynamic port range (49152–65535) or, additionally, the registered ports range (1024-49151) for mapping ports. In theory up to 64512 ports would be available for mapping purposes. Even when the number of hosts grows over the number of free ports, there is a possibility to create an additional NAT host which will expand the port mapping even further. In both cases, this way is more suitable for the provision of Portal related VMs. To make it easier to map hosts a type of "port-coding" scheme should be used. For example, in the `/24` network the ports of the NAT host `nat.example.com` would map like `nat.example.com:50211` ⇒ `192.168.0.211:22` and `nat.example.com:51211` ⇒ `192.168.0.211:80`. So the first two digits determine the destination port (here: 50 is SSH, 51 is HTTP) and the last three digits determine the last value of the IPv4 address (here 211 maps to 192.168.0.211).

5.9 Portal infopages

5.9.1 Purpose

Portal infopages represent the landing page of the Portal domain. On this page, which should build upon a content management system (CMS), the general information about the Portal should be presented. Moreover, this site should contain actual information from compute centers like maintenance, new features, or similar notifications. This

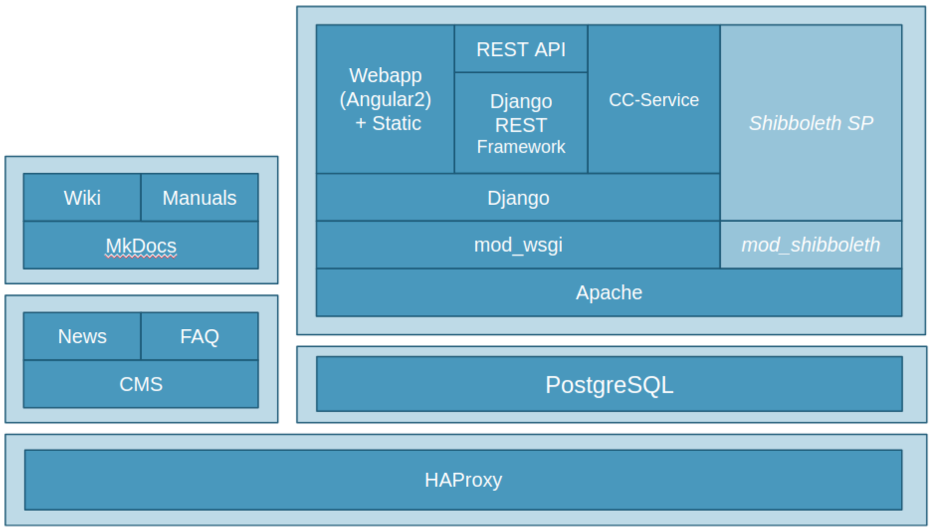


FIGURE 5.5: Microservices of the de.NBI Portal. Every Service is accesible over HAProxy with URL mapping.

service should be accessible without any authentication required. The design of the landing page should conform to the de.NBI corporate design. The maintenance of the content and news should be made over extra accounts for that purpose and is reserved strictly to the administration office and eventually CCRs.

5.10 Isolation and component communication

5.10.1 Microservice architecture

To solve the requirement of independent deployment, as mentioned in Section 5.2.1 and make the deployment process as easy as possible, each component should become its own individual service, which runs isolated and independently (Figure 5.5). To solve the installation issues, all microservices should get containerized, run in isolated environments, and communicate over the network. For the containerization and environment setup, the docker engine is used. Docker automatically allows the user to generate image with installed and configured software and manage ports and volumes mounted in the container. Since the containers communicate with each other via HTTP, a private network must be set up. Automatic network establishment, set-up and configuration of docker containers can be achieved with the tool docker-compose. This tool needs a configuration file with settings and guidelines (e.g. auto-relaunch when a container crashes) for each docker container.

5.10.2 URL mapping and SSL offloading

The services must be available under the same domain, so it is necessary to map the paths of the Portal URL to the corresponding service. For example when the user navigates to the URL `https://portal.example.com/wiki` the documentation service should be accessed. For this purpose, an extra service should exist, which acts as a proxy for the internal network. This kind of a proxy gateway connects the internal service-network with the outer world. Neither the Portal core nor any other service can be accessed directly from outside of the private network.

Furthermore using central "gateway" for all services makes it possible to make use of SSL offloading as an additional benefit. If SSL offloading is used, the proxy-service handles the SSL handshake and HTTPS communication with the clients. Communication between containers in the internal, isolated Network uses unencrypted HTTP and thus brings an internal speedup. An additional advantage of SSL offloading is reduced certificate maintenance overhead, because proxy-service is the only place to configure the certificates.

5.10.3 Developement environment

When the project is developed by several developers, it is advantageous to keep the software modular. It is also important to keep the test environment as simple as possible for the developers. The microservice architecture allows to start only those components that are needed for the current development and thus recreate a part of the production environment. Additionally, the docker based containers have the possibility to integrate local (development) directories into the container. This feature allows the software changes to be tested immediately without rebuilding and restarting the container. For the development workflow of this project there is a need for a set of services which mount the directory and allow a fast reload of the developed instance.

Another possibility for creating a development environment is a pre-configured virtual machine, which runs all services without isolation and contains the correct environment in a single image. That approach would allow accessing all services without switching containers.

5.10.4 Production environment and deployment

Disposing the software for public access requires an production environment. The production environment only needs to contain the components needed to run the software. In this project the production environment contains all microservices which run as production docker containers. The difference to the development containers is that they configured to contain only production-related components (e.g. lightweight web server), and all development-related components (e.g. debugging tools) are omitted. This configuration also allows automatic deployment of the components, so that automatic tests and updates will be possible in the future.

Chapter 6

Basic implementation of the Project

6.1 Basic implementation of the de.NBI Portal

In this thesis, a part of the requirements was also implemented. The implemented functionality concerns project management and group management. In addition, pre-tests for the implementation of the portal client were done. Since authentication is an important component for the further implementation of the project, integration in Elixir AAI with Shibboleth was made as the first implementation step.

6.2 Shibboleth configuration

As already described in Section 5.3.1, Shibboleth authentication mechanism consists of different components. The component that has to be installed and set up by the de.NBI cloud portal operator is the SP. The installation of the Shibboleth software on debian linux is done via command line. For the configuration of the SP there are several files in the directory `/etc/shibboleth/`. Among other files there are two files which are significant in the configuration of the SP: `Metadata.xml` and `shibboleth2.xml`.

The latter file contains the IdP configuration section:

```
<SSO entityID="https://login.elixir-czech.org/idp/">
 SAML2
</SSO>
```

LISTING 6.1: Shibboleth IdP configuration: The URL of the provider is used here as `entityID` (unique identifier) of the IdP, the value between the tags is the used protocol.

The rest of the configuration file consists of the attribute extractor, which points to an XML file, containing all configuration for attribute mapping. Each attribute is a key-value pair and contains attributes which should be queried from the IdP. Therefore, additional attribute configuration and a pointer to the SSL certificates are present in `Metadata.xml` as well.

```
<AttributeExtractor type="XML" validate="true" reloadChanges="false"
    path="attribute-map.xml"/>
<AttributeResolver type="Query" subjectMatch="true"/>

<!-- Default filtering policy for recognized attributes, lets other
    data pass. -->
<AttributeFilter type="XML" validate="true" path="attribute-policy.
    xml"/>

<!-- Simple file-based resolver for using a single keypair. (the cert
     and keys that are have generated)-->
<CredentialResolver type="File" key="sp-key.pem" certificate="sp-cert
    .pem"/>
```

LISTING 6.2: Shibboleth IdP configuration: The attribute and certificate configuration

One of the essential configuration parts is the definition of the metadata provider of the IdP. The metadata URL returns an entity descriptor of the IdP which contains all information about the configuration of this IdP:

```
<MetadataProvider type="XML" uri="https://login.elixir-czech.org/
    proxy/saml2/idp/metadata.php"
backingFilePath="elixir-idp-metadata.xml" reloadInterval="1800" >
</MetadataProvider>
```

LISTING 6.3: Shibboleth IdP configuration: IdP Metadata configuration

Another important file for the configuration is the `Metadata.xml` which provides the configuration details of the service provider for the IdP. The XML file contains an object which defines available signing algorithms, encryption methods, shibboleth endpoints, and the descriptor of the service.

The complete configuration assumes a registration of the SP by the IdP as well. For this purpose, the SP operator sends the public key used in the SP configuration (and possibly additional information about the service) to the IdP operator. After the identity provider verified the SP and added it to the allowed service providers on his own site, the Shibboleth service on the SP site can be started. If the service has started successfully, it can accept login requests. To allow Shibboleth to protect specific content, there is an additional configuration of the web service needed. The most basic case is to protect a web directory using `.htaccess` file and `mod-shibboleth` in the Apache web server:

```
<Location /portal/ >
AuthType shibboleth
Require shibboleth
ShibRequestSetting requireSession 1
require shib-session
</Location>
```

Listing 6.4: Directory protection with shibboleth: In order to access the directory `/portal/` a valid shibboleth session is required.

Additionally, there is a possibility to filter the user by providing Shibboleth attributes, which are automatically submitted to the service provider after user login:

```
<Location /portal/ >
AuthType shibboleth
ShibRequestSetting requireSession true
Require shib-attr unscoped-affiliation staff
</Location>
```

Listing 6.5: Directory protection with shibboleth: In order to access the directory `/portal/` a valid shibboleth session **and** staff affilitation in SAML attributes is required.

Besides the directory protection with Apache web server, Shibboleth is also able to communicate directly with the web application via the "`mod_shibboleth`" module. Thus the web application authenticates the user, it also allows software-side authorization of the user and enables a more detailed access control of the web application. For example, the web application could use additional complex access policies depending on SAML attributes.

6.3 Portal core

The Portal core is the central node for communication between Portal web application, Perun and Portal clients. It processes requests from the web application, submits requests to Perun and communicates with the compute centers via portal client instances. Portal provides an API interface and can only be accessed after authentication through the web application.

6.3.1 REST API framework comparison

There are different platforms for the REST interface implementation available. The main criteria for comparison are:

- **Shibboleth compatibility** – the framework should be able to integrate the Shibboleth SP as the authentication method as easy as possible.
- **Modularity** – the framework should be modular and easy to extend.
- **Open Source** – the framework project should be an open source software and be free of charge

In addition, the programming language should be widely used and have a good community support. The good community support is especially important for inexperienced developers, because they would quickly familiarize themselves with the software by reading programming language tutorials and How-Tos. Additionally, the programming language should be able to execute system calls in a simple way.

There are some REST API frameworks with different programming languages available. The four most often used languages of the web application development are JavaScript, Python, Java and PHP[1]. Since the de.NBI Portal core should be able to make system calls besides serving the web application, JavaScript is not applicable.

[1]See `https://spectrum.ieee.org/computing/software/ the-2017-top-programming-languages`

For development in Java, the Spring framework would be one of the choices for the de.NBI Portal. And for development in PHP a Yii framework would be a good choice. To develop in Python a Django framework, which is used to create RESTful services, could be used. All three frameworks provide similar functions and are well-known. The syntax of Python is clearer than PHP and deployment of Python is less resources intensive than Java. In addition Portal client will be developed in Python, since OpenStack provides excellent Python bindings. For resource-sparing deployment, clearly arranged source code and to keep the variety of used languages at the minimum, the Django framework was chosen.

6.3.2 Django components and Django REST Framework

The Django Framework consists of different apps, modules, and classes which can be used while building an application. To use the apps and modules, they must be listed in the framework settings file called `settings.py`. This file is an essential part of every Django project and is located in the project folder. Newly developed software components must also appear in the list in order to become active:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'main',
    'rest_framework',
    'api_v0',
    'projects',
    'shibboleth',
 ]
```

LISTING 6.6: Used apps of the project.

The Listing 6.7 shows all apps which are installed/used in the Django project. Along with the Django apps for administration, sessions, and static files provider, Portal apps are listed as well, such as the `main` application and `api_v0` – the API provider for

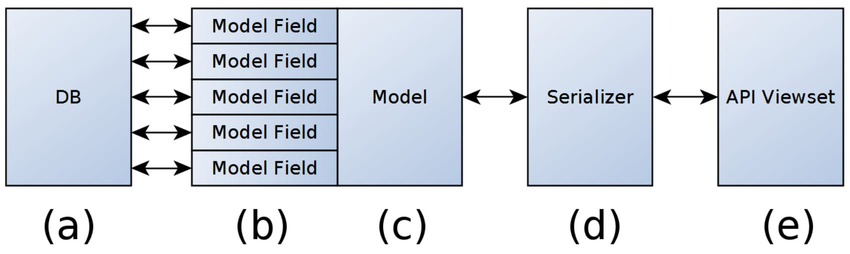


FIGURE 6.1: Django information flow: The table fields of the database (a) are mapped with Django's field models (b) and are aggregated to user-defined models (c) (represented as tables in the database). On any interaction the requested model is mapped according to user-defined rules over the serializer (d) into JSON objects and then displayed using viewset (e). The serialization or update of objects works in the reverse direction.

project applications. Additional components like `rest_framework` or `shibboleth` must appear in the list as well, in order to get activated by the framework.

Django Rest Framework (DRF) is an additional module which implements a lot of very useful classes for creating REST API endpoints. For example, different serializers to convert Python Models from and to JSON, to deal with field mapping of models to the corresponding JSON keys and verifying the data using constraints. The serializer can get extended and modified by adding overwrite methods. The other important structure type of the DRF is the ViewSet, a class for correct API query handling. It can handle different HTML methods and references different serializers to render JSON output (Figure 6.1).

6.3.3 Shibboleth authentication method

For the realization of the Shibboleth authentication, an external Django plugin is used which is developed and maintained by the library of Brown university[2]. This plugin allows to integrate Shibboleth seamlessly into the Django environment, acts as an authentication provider for the Django back-end, and utilizes the Django user model with the local database. The configuration implies an installed and configured Shibboleth service provider, Apache web server, and enabled mod_shibboleth.

[2]See `https://github.com/Brown-University-Library/django-shibboleth-remoteuser` for more information

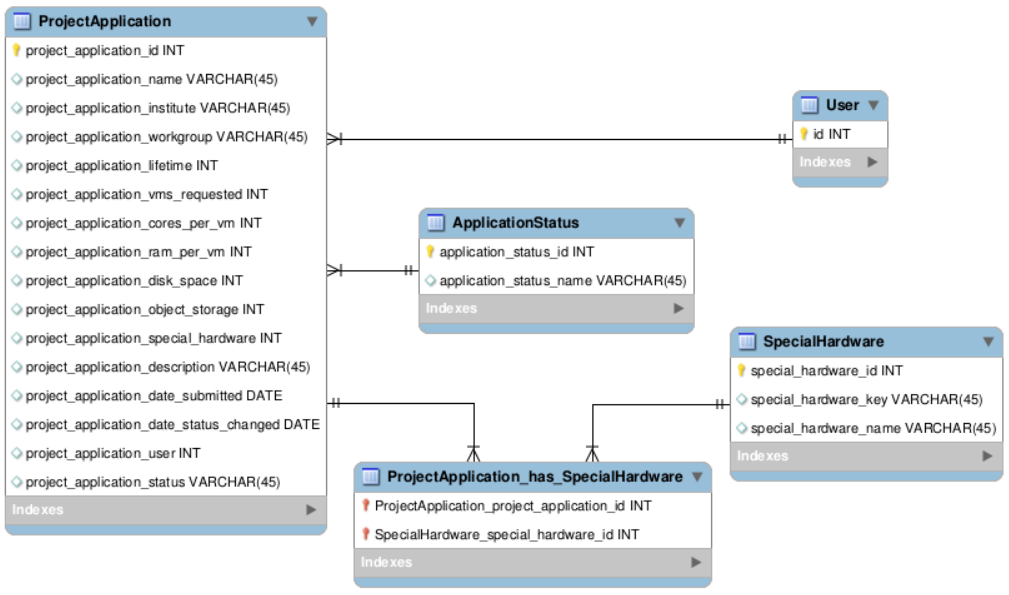

ProjectApplication
project_application_id INT
project_application_name VARCHAR(45)
project_application_institute VARCHAR(45)
project_application_workgroup VARCHAR(45)
project_application_lifetime INT
project_application_vms_requested INT
project_application_cores_per_vm INT
project_application_ram_per_vm INT
project_application_disk_space INT
project_application_object_storage INT
project_application_special_hardware INT
project_application_description VARCHAR(45)
project_application_date_submitted DATE
project_application_date_status_changed DATE
project_application_user INT
project_application_status VARCHAR(45)
Indexes
User
id INT
Indexes
ApplicationStatus
application_status_id INT
application_status_name VARCHAR(45)
Indexes
SpecialHardware
special_hardware_id INT
special_hardware_key VARCHAR(45)
special_hardware_name VARCHAR(45)
Indexes
ProjectApplication_has_SpecialHardware
ProjectApplication_project_application_id INT
SpecialHardware_special_hardware_id INT
Indexes

FIGURE 6.2: Additional models "SpecialHardware" and "ApplicationStatus" allow to extend the application process without model update.

6.3.4 Data modelling

The Django rest framework brings additional tools for the serializing objects for building API endpoints. Since the local DB is only used for the handling of new project applications, the model is relatively small (Figure 6.2). A project application has significant relationship to the special hardware model, which describes additional non-default hardware requirements for the project and two foreign key fields (one for the user and one for application status, which determines the actual step in the application process).

The primary model of the Portal is outsourced by Perun using additional attributes, and the project application is the only model which is not covered by the Perun system and is hosted locally. Only accepted applications become actual projects in the Perun environment, so the state of Perun represents the current working state with projects that have been approved.

6.3.5 Deployment

The deployment of the Django application can be done in different ways. Django provides a small web server out of the box, making it suitable for development purposes not to be used in production. For production deployment, the Apache server is used since the Shibboleth plugin of Django requires mod_shibboleth to communicate with the Shibboleth service provider daemon *shibd*.

6.4 Portal Webapplication

While Portal core offers an API, there is a need for a web GUI which connects to the Perun system and the Portal API as well. The web application confronts with many challenges on the client side. Modern web browsers feature different technologies; the application should support different web browser types, operating systems and browser configurations. The application should have a clear design for mobile devices and desktops as well. Furthermore, it should be easy to maintain the application, as to modify, add and exchange functionality of the application. Writing a modular application from scratch which fulfills these requirements is a time-intensive task. For that purpose, a browser-based Angular4 framework was chosen, which compiles TypeScript to standardized JavaScript (ISO/IEC 16262:2011 – ECMAScript) and fulfills requirements of most web browsers. The Angular4 framework consists of different modules and follows the modular architecture pattern. Modularization enables easy extension and maintenance of the application.

6.4.1 Angular components

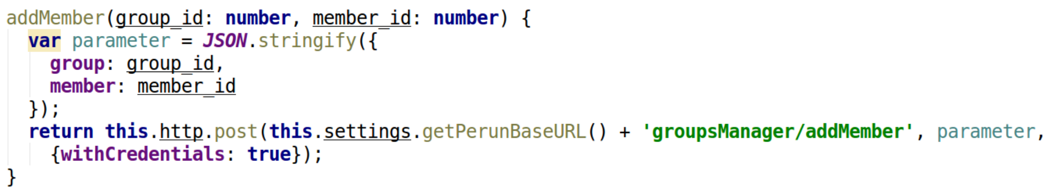
Angular consists of different modules, ranging from services, components, functions or just values. The services can be used in multiple components over dependency injection and thus allow to produce reusable software parts. Furthermore Angular also provides tags for data/property binding and event binding in HTML templates. Angular utilizes TypeScript such as a programming language for logic, HTML for templating and SASS (Syntactically Awesome Stylesheets) for style markup. For production, all components are translated into JavaSctipt (e.g., ECMAScript 6), CSS, and corresponding templates. All components get compressed and minimized by the Angular command line build tool.

6.4.2 Data modeling

For easy handling of the APIs and internal markup logic, all types of API responses are mapped to the internal models in a precise and accurate manner, with getters and setters written in TypeScript.

6.4.3 Perun connection service

One of the core parts of the web application is the Perun connector. This connector, which is added over dependency injection to the Angular component can perform Perun API calls to different endpoints of Perun's JSON RPC endpoints. The API calls have a simple structure:



```
addMember(group_id: number, member_id: number) {
  var parameter = JSON.stringify({
    group: group_id,
    member: member_id
  });
  return this.http.post(this.settings.getPerunBaseURL() + 'groupsManager/addMember', parameter,
    {withCredentials: true});
}
```

Listing 6.7: Example call of a function in the Perun system using JSON RPC.

6.4.4 Portal core service

As well as the Perun connection service consumes Perun's JSON RPC API, there is also a service for the Portal API. This service works in a similar way and performs calls which deal with new project applications. On the Django-side a valid Shibboleth session is needed as well. Since the Django application already requires authorization, no additional authentication tokens are sent to the portal core when submitting requests. The only token that is sent (together with write requests) is CSRF token. The CSRF token is not used for authentication and provides additional protection of the application against cross-site request attacks.

6.4.5 Frontend components

To display HTML components in a browser, the additional template "Core UI" was chosen. This template is built with Angular components and utilizes the Twitter's Bootstrap HTML5 framework, which provides components with mobile-adapted functionality and wide spectrum of classes and patterns for GUI design.

6.5 Portal client

After the user has sent a request to start a virtual machine, Portal core forwards the request to the portal client (See Section 5.7.1). Since the processes for starting a virtual machine differ between the locations, Portal client should represent an abstraction layer for these tasks. Abstraction will be made possible by an API that unifies all requests to manage VMs for all compute centers. As part of this work, an API testimplementation for REST and for JSON RPC has been done. It has been shown that JSON RPC API, compared to REST, meets the requirements of the interface definition of Portal client. This knowledge gained from the test implementation should be used in the future for the reference implementation of the Portal client.

6.5.1 JSON-RPC benefits over REST

For the test implementation of the REST based Portal client, the API was specified using the OpenAPI design rules. After that Swagger, an online OpenAPI specification editor generated the complete python REST server, which utilizes the API specification file (called `swagger.yml`) directly via the *connexion* library and provides an REST API interface with request verification out of the box. Unfortunately, the REST interface comes to its limits, when more *complex* functionality over the interface is required such as attaching floating IP's to VM. Those cases can hardly fit into the set of Create Remove Update Delete (CRUD) actions (those map to the HTTP Methods[3]) of REST and therefore another technology for the API was tested – JSON RPC. It takes another approach and offers only one endpoint for all requests while REST has multiple endpoints, at least one for each object.

[3]HTTP Methods are e.g. `GET`, `POST`, `PATCH`

To perform the correct JSON RPC request, the application must know the name of the method to call and which data/parameters to send. For example, a correct JSON RPC call according to the specification[4] would look like:

```
{"jsonrpc": "2.0",
 "method": "vm.create",
 "params": {"name": "test",
            "number": 42},
 "id": 3}
```

LISTING 6.8: Example of the JSON RPC request object.

In this case, the method `vm.create` would be called with parameters `name = test` and `number = 42`. It is also possible to call the methods without any parameter (omit `"params"`) or send batch requests. The required parameters are `"jsonrpc"` which provides the version of the protocol, `"method"` the called method on the server side. An optional, but highly useful parameter is the `"id"`, which usually contains a unique number, and ensures that answers, especially those from the batch queries, can be distinguished from each other and correctly assigned to the requests.

6.5.2 Flask framework

As mentioned in Section 2.5, OpenStack consists of tools which are written in Python and provides python bindings as well. Since the test application should remain lightweight, a web framework called *Flask* is used. With Flask it is possible to set up a simple JSON RPC server using the `flask_jsonrpc` module. Since function names from JSON objects may differ from actual function names in Flask, they can be mapped via "@"-annotation:

```
@jsonrpc.method('vm.create(vm_definition=dict)␣->␣object', validate=
    True)
def vm_create(vm_definition):
    #do stuff
    return "Response"
```

LISTING 6.9: Sample JSON RPC method with Flask

Unfortunately, JSON RPC for Flask can not deeply verify the incoming data.

[4]`http://www.jsonrpc.org/specification`

The only form of verification that can be done is one of the parameter types of the request (e.g. if it is a `dict` or `string`). For explicit semantic verification of the `dict`, the developer must provide his own code. This issue is automatically solved in the REST API through providing the correct API definition file. Nonetheless, JSON RPC was chosen as the communication technology between Portal client and Portal core.

6.6 Portal infopages

Portal infopages provide current news and general information about the de.NBI cloud portal to the user. For that purpose, a small and lightweight CMS – WordPress (WP) is used. It is the standard, well-known engine for blogging and creating small websites and is available as an official Docker container as well. For the correct container set up the following steps have to be completed:

- **Connect the database** – the DB is a standalone container which runs the official MariaDB container. Database content is stored permanently outside of the container and is mounted as an external volume. WP connects to this DB using the internal network.
- **Adding template** – to design the Portal infopages similar to the de.NBI homepage[5] a small template was used and adopted in the way to resemble the de.NBI page.
- **Content and Menus** – finally, the top menu structure was added and some basic content about the de.NBI cloud was added.

The main advantage of this container configuration is a simple and effortless update of the WordPress engine. In order to upgrade Portal infopages to a newer WordPress version, the new WP container version needs to be pulled, the old container stopped, removed and finally the new container should be deployed.

[5]See `https://denbi.de`

Since all information regarding the content and the template is stored outside of the container, there is no further configuration required:

```
$> docker pull wordpress:latest
$> docker-compose stop portal-infopages
$> docker-compose rm portal-infopages
$> docker-compose start portal-infopages
```

Listing 6.10: WordPress update in Portal infopages

6.7 Deployment

6.7.1 Automatic container build

Currently the containers are built and deployed manually, but since the source code of the software and container configurations (Dockerfiles) are stored in the repositories, it is possible to create an automatic environment for building and testing of containers for production deployment.

6.7.2 reverse Proxy

The main component of the deployment of the microservices is the reverse proxy, whose functional requirements are described in Section 5.10.2. HAproxy runs in its own container, which consists of the configuration file and SSL certificates for the production domain. The configuration is divided into the `frontend` and the `backend` section[6]. In the sections it is defined which incoming requests are accepted, and to which containers the content should be forwarded. The rules for the accepted URL patterns are defined in the `frondend` section and the corresponding servers are listed in the `backend` section. Furthermore, the `frontend` section contains settings for SSL encryption.

[6]See `https://www.digitalocean.com/community/tutorials/an-introduction-to-haproxy-and-load-balancing-concepts`

In the following example the frontend leads to the SSL port, distinguishes requests by the incoming URL pattern and transfers the request to the corresponding server in the `backend`:

```
frontend www-https
    mode http
        reqadd X-Forwarded-Proto:\ https
        bind *:443 ssl crt /etc/haproxy/ssl/portal-dev.denbi.de.pem
        redirect scheme https if !{ ssl_fc }

        acl is_media path_beg -i /media
        use_backend portal if is_media

[...]

backend portal
    server p1 portal:443 check ssl verify
```

LISTING 6.11: Sample HAProxy configuration

6.7.3 Docker encapsulating with LXC containers

In addition to the deployment via Docker on a bare-metal server or in a virtual machine, the docker containers can be deployed in the LXC containers [7]. LXC container lets the developer create a close environment similar to the standalone Linux installation, but without a need for a separate kernel. LXC allows the operation of unprivileged containers with other user identifiers (UID) aside of root. The production containers therefore run in an LXC *meta*container. This type of deployment provides additional security (unprivilidged LXC container) while maintaining the flexibility (Docker based microservices).

6.7.4 Deplyoment without containers.

Another deployment approach is to install software directly on one bare metal host or to set up multiple VMs instead of containers. Automatic deployment can also be achieved with this approach using special software “Puppet” with scripts for example,

[7]See `https://linuxcontainers.org/lxc/introduction/`

or “Ansible” with playbooks. However, among all deployment techniques, the most lightweight and low-maintenance solution is still provided by the use of Docker containers, since they combine a good level of abstraction with the simplicity of automation using docker-compose.

Chapter 7

Discussion

7.1 Look back, conclusion

Good project development flow requires a lot of organizational and planning tasks. A good development of the software also requires a well structured software design. In view of the project's longevity, the project was kept as modular as possible so that the individual software components could be replaced by new ones in the course of time. Furthermore an important part of the project development was the periodic exchange of knowledge between the project participants. This project started with the the international Elixir AAI kick-off workshop in Freiburg. The aim of the workshop was to get to know Elixir and in particular the infrastructure for authentication and authorization. The further exchange of knowledge regarding project development and integration into the Elixir AAI took place in over 10 online conferences, among others with Elixir AAI representatives. Additionally partial results of the design and planning of de.NBI cloud portal were discussed in the offline conferences of so-called "special interest group for the de.NBI cloud (SIG6)" and other de.NBI meetings.

The newly collected ideas were discussed, tested, presented and documented. Finally, the technologies that met the requirements were selected and tested. Despite various bugs in some systems and partly undocumented software, the basic implementation of the deNBI portal could be created. An important success is the extensive use of the Elixir AAI for the authentication and storage of user and group information.

7.2 Outlook

The provision of computing resources for life scientists is an honourable and demanding task for the de.NBI project. The further task is to extend the portal's functionality. For example, alternative SSO technologies should be integrated, since Elixir offers openID registration in addition to Shibboleth.An important feature is the reference implementation of the portal client and the corresponding module in the portal core, as well as the integration of the functionality to start virtual machines in the portal web application. During this work new Perun attributes were defined and already implemented in Perun. These can store information about the compute center and also the credits of the computing projects.

The further tasks to allow users deploy own VMs and mount volumes will be not less challenging than amount of this work. The enhancement of the Portal and extending with new features will provide an excellent platform for German scientist, support them with the compute power and help them to get better results faster, without loosing hours in the labor of calculation.

Appendix A

Appendix A

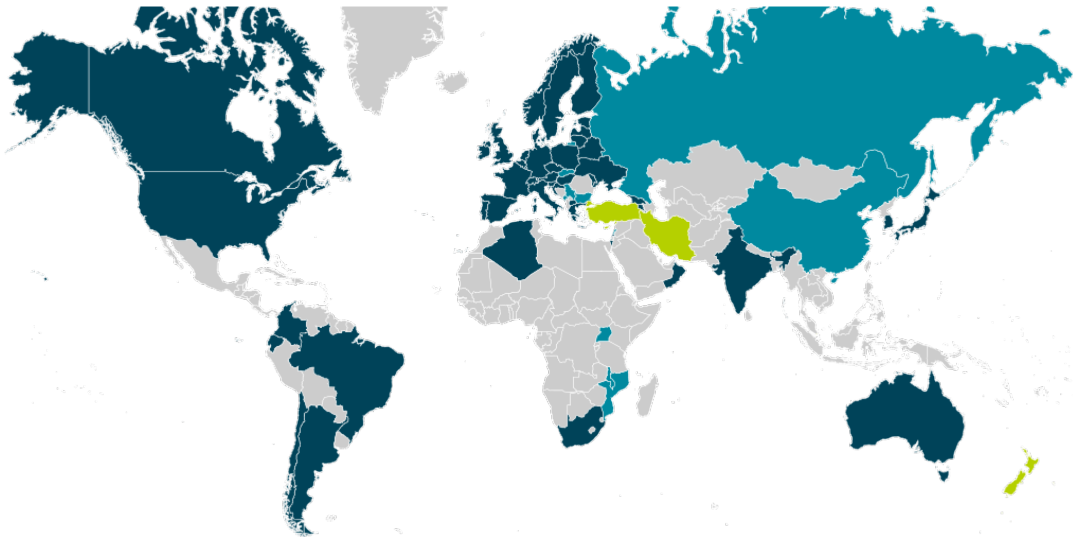


Figure A.1: eduGAIN memberships. Full members of eduGAIN are shown in dark blue. Borrowed from `https://technical.edugain.org/status.php`

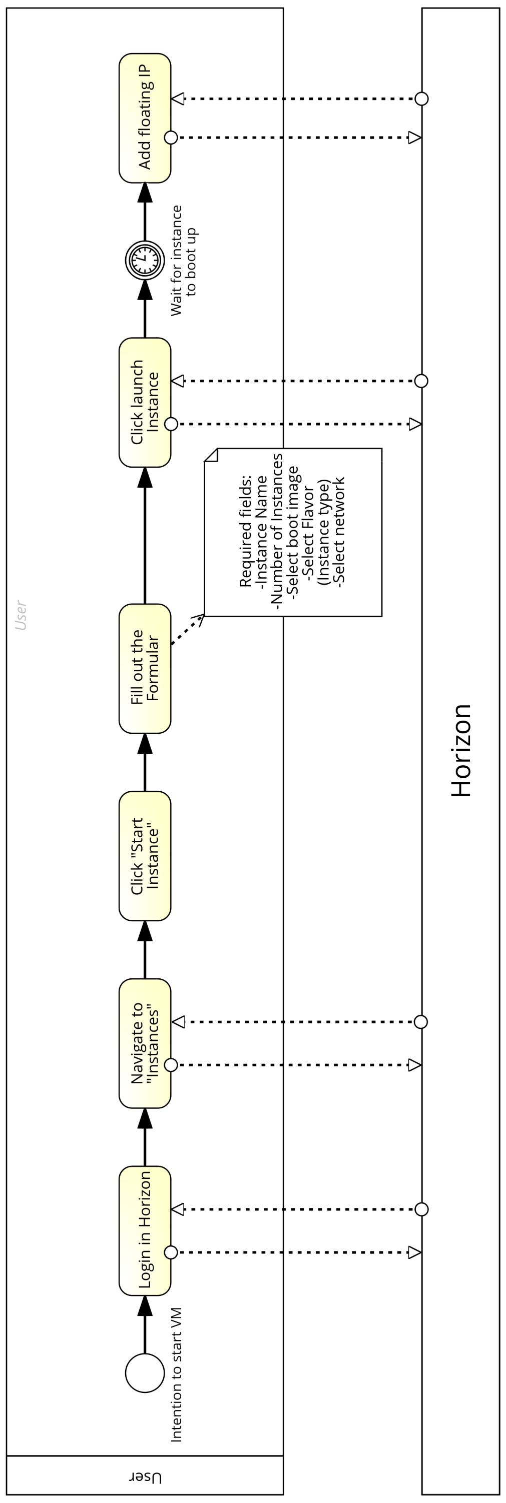
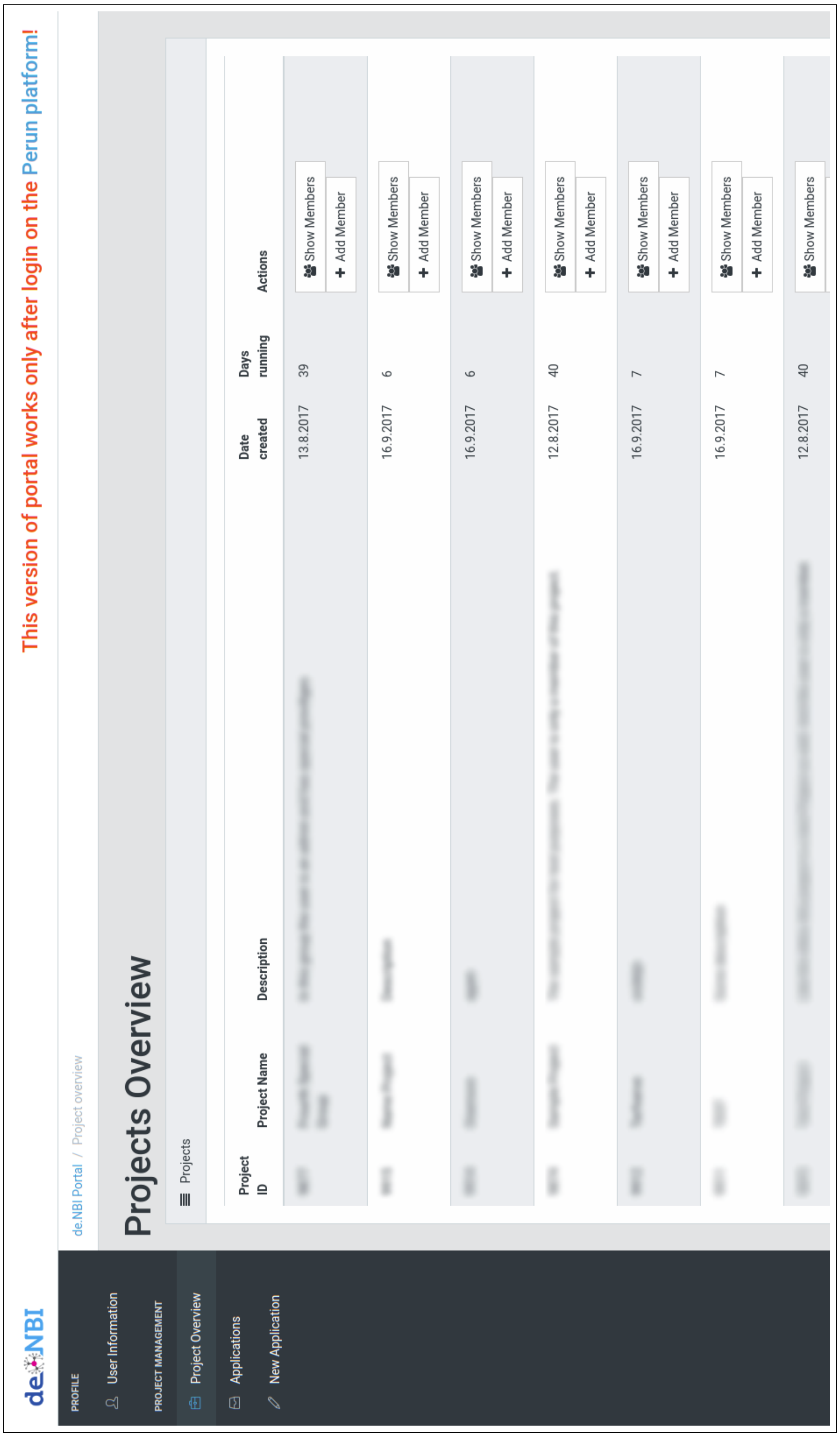


FIGURE A.2: The process of starting a single VM with floating IP in horizon



This version of portal works only after login on the Perun platform!

Projects Overview

Projects

Project ID	Project Name	Description	Date created	Days running	Actions
[illegible]	[illegible]	[illegible]	13.8.2017	39	Show Members + Add Member
[illegible]	[illegible]	[illegible]	16.9.2017	6	Show Members + Add Member
[illegible]	[illegible]	[illegible]	16.9.2017	6	Show Members + Add Member
[illegible]	[illegible]	[illegible]	12.8.2017	40	Show Members + Add Member
[illegible]	[illegible]	[illegible]	16.9.2017	7	Show Members + Add Member
[illegible]	[illegible]	[illegible]	16.9.2017	7	Show Members + Add Member
[illegible]	[illegible]	[illegible]	12.8.2017	40	Show Members

Figure A.3: The page, where user can projects he is involved in.

Bibliography

[1] (2017). *Federated keystone (OpenStack Documentation).* OpenStack. https://docs.openstack.org/security-guide/identity/federated-keystone.html.

[2] Belmann, P., Dröge, J., Bremges, A., McHardy, A. C., Sczyrba, A., and Barton, M. D. (2015). Bioboxes: standardised containers for interchangeable bioinformatics software. *Gigascience*, 4(1):47.

[3] BMBF-Internetredaktion (2015). Bekanntmachung des bundesministeriums für bildung und forschung zu förderrichtlinien für ein "deutsches netzwerk für bioinformatik-infrastruktur".

[4] Chonoles, M. J. (2017). *OCUP 2 Certification Guide: Preparing for the OMG Certified UML 2.5 Professional 2 Foundation Exam.* Morgan Kaufmann.

[5] CLIMB (2017). Climb - the project; http://www.climb.ac.uk/overview/project/.

[6] Connor, T. R., Loman, N. J., Thompson, S., Smith, A., Southgate, J., Poplawski, R., Bull, M. J., Richardson, E., Ismail, M., Elwood-Thompson, S., et al. (2016a). Climb (the cloud infrastructure for microbial bioinformatics): an online resource for the medical microbiology community. *Microbial Genomics*, 2(9).

[7] Connor, T. R., Loman, N. J., Thompson, S., Smith, A., Southgate, J., Poplawski, R., Bull, M. J., Richardson, E., Ismail, M., Elwood-Thompson, S., et al. (2016b). Climb (the cloud infrastructure for microbial bioinformatics): an online resource for the medical microbiology community. *Microbial Genomics*, 2(9).

[8] Hetze, S. (2012). Single sign-on für webanwendungen mit shibboleth. *Heise ix*, (5):119–123.

[9] Markelov, A. (2016). *Certified OpenStack Administrator Study Guide.* Springer.

[10] Mell, P., Grance, T., et al. (2011). The nist definition of cloud computing.

[11] Pühler, A. (2017). de.nbi – ein deutsches netzwerk für bioinformatik-infrastruktur. *BIOspektrum*, (1):103–103.

[12] Tsolkas, A. and Schmidt, K. (2017). Single sign on. In *Rollen und Berechtigungskonzepte*, pages 189–224. Springer.

[13] Vyas, U. (2016). *Applied OpenStack Design Patterns: Design solutions for production-ready infrastructure with OpenStack components*. Apress.

[14] Yakov Rekhter, Robert G Moskowitz, D. K. G. J. d. G. E. L. (1996). Address Allocation for Private Internets. RFC 1918, RFC Editor.