D5.3 First HIDALGO Portal Release and System Operation Report

Document Identification			
Status	Final	**Due Date**	30/11/2019
Version	1.0	**Submission Date**	12/12/2019

Related WP	WP5	**Document Reference**	D5.3
Related Deliverable(s)	D5.1, D5.2, D5.6, D5.7	**Dissemination Level (*)**	PU
Lead Participant	ATOS	**Lead Author**	F. Javier Nieto (ATOS)
Contributors	KNOW, ICCS, USTUTT, BUL, PSNC	**Reviewers**	Gregor Bankhamer (PLUS)
			Miroslaw Kupczyk (PSNC)

Keywords:
HPCaaS, HIDALGO, CoE, Portal, Web, Entrypoint

Document Information

List of Contributors	
Name	Partner
Aleix Sanchis	ATOS
Marcin Lawenda	PSNC
Dineshkumar Rajagopal	USTUTT

Document History			
Version	Date	Change editors	Changes
0.1	15/11/2019	F. Javier Nieto	TOC, Section 1.
0.2	21/11/2019	Dineshkumar Rajagopal	Sections 3, 6 and 8
0.3	28/11/2019	F. Javier Nieto	Section 2
0.4	29/11/2019	Marcin Lawenda	Section 7
0.5	02/12/2019	Aleix Sanchis	Section 4
0.6	05/12/2019	Aleix Sanchis, F. Javier Nieto	Sections 5, 9 and 10
1.0	11/122019	F. Javier Nieto	Incorporate comments from internal review. Final version.

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	F. Javier Nieto (ATOS)	12/12/2019
Quality manager	Marcin Lawenda (PSNC)	12/12/2019
Project Coordinator	Francisco Javier Nieto de Santos (ATOS)	12/12/2019

Table of Contents

List of Tables

List of Figures

List of Acronyms

Abbreviation / acronym	Description
API	Advanced Programming Interface
A&A	Authentication and Authorization
CI/CD	Continuous Integration / Continuous Deployment
DC	Data Catalogue
DMS	Data Management System
DRF	Django-Rest-Framework
Dx.y	Deliverable number y belonging to WP x
EC	European Commission
FAQ	Frequently Asked Questions
GDPR	General Data Protection Regulation
GUI	Graphical User Interface
IdAM	Intelligent Digital Asset Management
MooCs	Massive Open Online Courses
MSX	Project Milestone X
MVP	Minimum viable product
OIDC	OpenID Connect
Q&A	Questions and Answers
REST	Representational State Transfer
SAML	Security Assertion Markup Language
SCM	Source Control Management
SPA	Single-Page Applications
SSO	Single Sign On
VM	Virtual Machine
WP	Work Package

Table 1 List of acronyms

Executive Summary

This document presents the implementation done for the first release of the HiDALGO Portal. The implemented features have been focused on the execution of applications, the community support, the data management and the code maintenance.

The document presents the features available, comparing the current implementation with the original plans. Then, for each main feature, the document presents how the feature was implemented, it describes the available APIs for each component (both graphical and REST APIs) and provides information about how to use the features. In the case of REST APIs, the document provides examples of calls to the services. In the case of GUIs, providing screenshots and some guidance about the options.

1 Introduction

1.1 Purpose of the document

This report presents the current implementation of the first release of the HiDALGO Portal, according to the list of features and design done in D5.2[1]. It includes information about the deployment of components and the operation of the Portal during the first months of the project. This document will be updated in future releases, according to the progress done in development activities.

1.2 Relation to other project work

This document is directly related to D5.2[1], since it describes the features and designs to be followed in the Portal implementation, according to the requirements defined in D6.1[2]. It is also related to the workflows defined in D6.2[3] and WP4 in general (to be supported by the Portal). It is the first release of the portal development in T5.3, that will be updated in D5.6 and D5.7.

1.3 Structure of the document

This document is structured in 8 major chapters:

Chapter 2. talks about the features that have been implemented in the context of the first release of the Portal, in line with the designs done in D5.2[1].

Chapter 3. to **9.** describe how the features were implemented, the supported functionality, the components used, the APIs available and how these features can be used in the context of HiDALGO.

Chapter 10. just provides a summary and a set of conclusions obtained after the current implementation.

2 Implemented Features

2.1 Current Features Available

The report D5.2[1] (in its section 2) already sketched the list of features that the project was planning to provide through the HiDALGO Portal. It was identifying eight main features:

- *Centralized user management:* authentication and authorization related feature, enabling single-sign-on and being compliant with GDPR for users' personal data;
- *User discovery:* allow users to publish their profiles and to find other users with similar interests or concrete skills;
- *User community support:* set of solutions for supporting HiDALGO stakeholders, like training material, FAQ, adequate usage documentation and concrete tools for support (ticketing system and public forum);
- *Data visualization:* visualization of users' data, including some analysis algorithms which support the detection of patterns/outliers, as well as certain search functions;
- *Data management:* set of tools for exploring, transferring and storing datasets in an easy way;
- *Interactive code:* enable the prototyping and testing of code and data analytic tasks through notebooks and similar web-based tools;
- *Application visualization:* show progress indicators and other information about the application status (monitoring information, also about job queues);
- *Application execution:* abstract HPC resources, so users can configure and execute easily their applications through the Portal;
- *Portal maintenance:* other features like code testing, continuous integration, deployment support and monitoring.

From this list, according to the initial plans, feasibility criteria and potential utility of the features, the consortium has focused on the features related to components development & maintenance, easy execution of applications, the data catalogue and the first tools for community support.

As a summary, we can say that the following features are currently available:

- A CI/CD infrastructure, based on Jenkins[4] and a Git repository, for code testing and maintenance;
- A Moodle[5] instance for users' training and a matchmaking tool, in order to enable user community support;

- An instance of Keycloak[7] in order to enable a centralized user management and single-sign-on mechanisms;
- An instance of CKAN[6], as the solution for data management, as it acts as data catalogue and it can be also used to enable local storage of data and its movement;
- An instance of the Cloudify Orchestrator[8], with the Croupier plugin [13] (which enables the connection to HPC systems), and a customized frontend (with its corresponding backend) for launching the applications in the HPC systems in an easy way, enabling application execution and visualization features.

2.2 Integration and Deployment Infrastructure

PSNC and HLRS cloud services offer infrastructure for the HiDALGO portal integration and deployment to satisfy the CI/CD infrastructure requirements. Deployment infrastructure is interconnected with Jenkins as mentioned in Figure-1 to coordinate all the activities in a central location by portal developers and administrators. Similar integration and deployment infrastructure configurations are provided to resolve the conflicts between those environments. This is a similar concept to the web hosting services to provide staging and production infrastructure to ensure a smooth transition of integrated code to the deployment infrastructure without any changes. HLRS infrastructure uses KVM to enable the cloud capability on the physical server and provides VMs for each service as mentioned in Table 2 for both integration and deployment infrastructure.

Production infrastructure is used to deploy the portal for public access, so it should be fast enough and highly secure to provide interactive and trustworthy service. All the user access will be redirected to the corresponding service VM to distribute workload properly to meet the high service quality and avoid a single point of failure in the portal. If some service is slow due to heavy traffic, then more computing power will be provided. Password authentication is disabled in the VMs, so brute force attack is not possible to the infrastructure provided for the portal. A firewall is set up between the HLRS portal deployment network and the outside world to define security rules. The firewall rules are defined for administrative (SSH port: 22) and end-user access (HTTP port: 80 and HTTPS port: 443) to enhance network security. Physical infrastructure is accessible only from the HLRS network and provides access only for the HLRS administrators to prevent any damage from the outside world.

Services	HLRS Deployment VM Configuration	PSNC Integration VM Configuration
Jenkins	2 cores, 7 GB Memory, 30 GB Disk, Ubuntu 18.04 Server OS	4 cores, 8 GB Memory, 40 GB Disk, Ubuntu 18.04 Server OS

Moodle	2 cores, 7 GB Memory, 30 GB Disk, Ubuntu 18.04 Server OS	4 cores, 8 GB Memory, 40 GB Disk, Ubuntu 18.04 Server OS
Askbot	2 cores, 7 GB Memory, 30 GB Disk, Ubuntu 18.04 Server OS	4 cores, 8 GB Memory, 40 GB Disk, Ubuntu 18.04 Server OS
Matchmaking	2 cores, 7 GB Memory, 30 GB Disk, Ubuntu 18.04 Server OS	4 cores, 8 GB Memory, 40 GB Disk, Ubuntu 18.04 Server OS
CKAN	2 cores, 7 GB Memory, 30 GB Disk, Ubuntu 18.04 Server OS	4 cores, 8 GB Memory, 40 GB Disk, Ubuntu 18.04 Server OS
Portal	2 cores, 7 GB Memory, 30 GB Disk, Ubuntu 18.04 Server OS	4 cores, 8 GB Memory, 40 GB Disk, Ubuntu 18.04 Server OS
Keycloak IDM	2 cores, 7 GB Memory, 30 GB Disk, Ubuntu 18.04 Server OS	4 cores, 8 GB Memory, 40 GB Disk, Ubuntu 18.04 Server OS
Cloudify	2 cores, 7 GB Memory, 30 GB Disk, Centos7	4 cores, 8 GB Memory, 40 GB Disk, Centos 7

Table 2 Integration and Deployment VMs configuration for each service

2.3 Portal Roadmap Implementation

Deliverable D5.2[1] was proposing a roadmap related to the potential user needs. The focus was on enabling the execution of applications, as well as features related to users' management and data management (the catalogue, basically). This has been the main focus during the implementation tasks, although there were some changes with respect to the original plans, progressing more on the part of community management tools, so it will be possible to support them as soon as stakeholders are involved.

Main Component	Original Plan for MS4	Current Implementation
Single Sign On	Login once and access all the services	Ongoing
	All services connected to one account	Ongoing
	Users to sign up themselves	Done
	Manage users' roles and permissions	Done
	Group permissions and assign users to certain groups	Done

Portal Maintenance	Have a sandbox environment, in order to test changes quickly without risk	Done
	Test changes automatically, so new features do not break other parts of the code	Done
Application Execution	Execute a pilot and retrieve the results that are interesting to me	Done
	Abstract users from the complexity of the underlying infrastructure	Done
Application Status Visualization	Know the status of a running pilot, so users may access partial results if they want to	Done
	Check the logs of a pilot execution in order to know what happened	Postponed
Explore and Manage Data	Allow users to explore the data available, so they can see data that can be used	Done
	Create a public dataset and share it with other users, if the user wants to	Done
	Create private datasets, so only concrete users (or groups of users) may know about it	Done
Internal Storage	Store small input datasets in the platform, so they can be used for running pilots	Done
External Storage	Use datasets stored outside HiDALGO in order to execute pilots	Done
Visualization	Visualize datasets, so it is possible to show demonstrations	Partially Done
	Visualize datasets with temporal information, so it is possible to understand them	Partially Done
Book	Have all documentation organized for users	Postponed
	Developers can treat documentation as code, so it is easier to keep it updated	Partially Done
	Make available information for running pilots and for using the provided UIs	Partially Done
Support	Enable the possibility to keep on discussions through email for general support information	Ongoing
Non-Functional	Make the UI compliant with GDPR, so there will not be any legal issues	Partially Done

Table 3 Original features proposed for MS4 vs current implementation

Main Component	Plan for other release	Current Implementation
Application Execution	Do not require users to know a lot of technical details in order to run pilots	Done
Application Status Visualization	Visualize the pilot workflow execution in real-time, so it is possible to know its stage.	Partially Done
Explore and Manage Data	Retrieve information about the datasets, such as format, quality and quantity	Partially Done
Visualization	Visualize datasets with geospatial information, so it is possible to understand it better	Partially Done
Support	Allow users to have open discussions in a forum, so other users may benefit from such discussions	Ongoing
Training	Facilitate pilots' usage training material to users, so they can get the full potential of the applications	Partially Done
	Provide a catalogue of training courses, so users may select those they are interested in.	Done

Table 4 Features not planned for MS4 but included in the current implementation

Additionally, the matchmaking tool is already available, for allowing users to find peers with similar interests. This is a feature which was not planned neither for MS4 nor for MS5 release.

3 CI/CD Infrastructure

3.1 Implemented Solution

HiDALGO portal integration would be automatic and follow the CI/CD principles. The best open-source tools from the market are identified and installed for supporting automatic integration, testing and deployment. Jenkins is the best candidate to coordinate all the activities centrally. The latest version V2.176.1 is installed in the HLRS infrastructure with the configurations and plugins to support both freestyle and pipeline workflows. Jenkins web server provides seamless interconnection to the integration and deployment infrastructure for frequent integration and deployment by developers and administrators to realize the CI/CD benefits. Jenkins is installed in a standalone mode to have a single master handling all the functionalities (GUI web service, Jenkins pipeline or Jenkins project running, etc). Administrative privilege is disabled for all Jenkins users. Jenkins and its interconnection with the different infrastructures, actors are depicted in Figure 1. Developers have administrative privilege in the Virtual Machines (VMs) to manage all the activities in their respective VMs to install and test their components. Administrators have the privileges required to reset the VMs to the initial Operating System (OS) state manually when the developers request it by Email. OS resetting will be very helpful for the developers when there is an OS crash or unwanted issues during the development and integration phase.

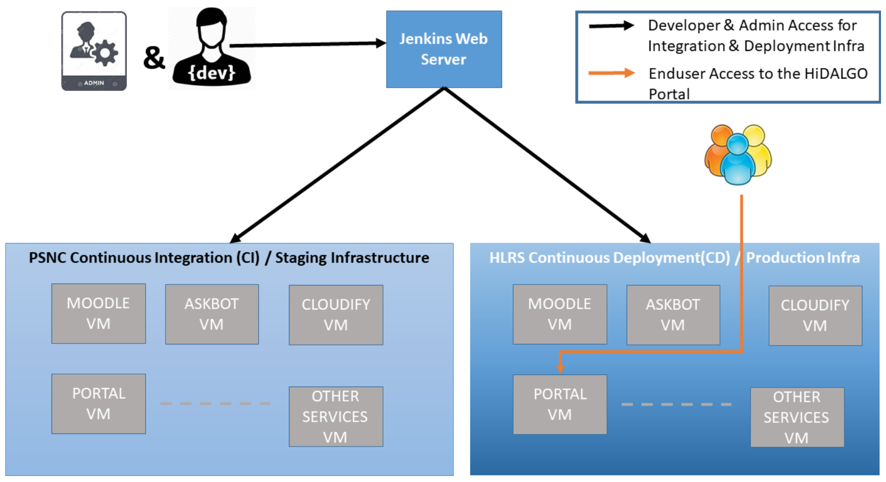

ADMIN
&
{dev}
Jenkins Web Server
Developer & Admin Access for Integration & Deployment Infra
Enduser Access to the HiDALGO Portal
PSNC Continuous Integration (CI) / Staging Infrastructure
MOODLE VM
ASKBOT VM
CLOUDIFY VM
PORTAL VM
OTHER SERVICES VM
HLRS Continuous Deployment(CD) / Production Infra
MOODLE VM
ASKBOT VM
CLOUDIFY VM
PORTAL VM
OTHER SERVICES VM

Figure 1 – Jenkins web server and its interconnection with the portal integration (PSNC), deployment (HLRS) infrastructure and different actors' roles and accessibility

Jenkins pipeline workflow is adapted inside our project, so each service is required to provide a pipeline to automate the manual software development tasks (E.g., building source code). Ansible is the perfect combination with Jenkins[4] to automate the complete software developments from building the components to deploy the integrated portal in the production infrastructure, so AnsibleV2.8.1 is installed in the Jenkins server to support those activities. Ansible will access remote infrastructure from the Jenkins server by SSH (Secure Shell) and perform the activities as defined in the Ansible scripts and Jenkins pipeline. Jenkins pipelines and Ansible scripts are managed inside the HLRS git repository infrastructure as a way to follow an adequate code practice. There is a possibility to release all the code in GitHub as an open-source repository, so sensitive details (credentials, etc) are encrypted and stored with the help of the ansible-vault command in order to avoid storing in the plain text format. More details regarding the Jenkins pipeline, integration and deployment infrastructure for the portal follow in later sub-sections.

3.2 Software Development Pipeline (Jenkins Pipeline)

An automatic software development pipeline (or Jenkins pipeline) consists of multiple stages and each stage runs specific functionalities with multiple tools to automate the software development activities (i.e. compile, run automated tests, etc). Jenkins pipelines are written in the Groovy language and follow a declarative style pipeline definition. Six stages are defined in the Jenkins pipeline, which is detailed below.

1. Checkout SCM - Clone the Jenkins pipeline and Ansible script to Jenkins workspace
2. Install in the integration infrastructure - Build or install in the integration environment
3. Integration test – Test the component in the integration environment
4. Manual approval for deployment – Wait for manual approval
5. Install in the deployment infrastructure - Build or install in the deployment environment
6. Deployment test - Test the component in the deployment environment

In the first stage, the Jenkins Git SCM plugin is used to connect with the repository and clone the Jenkins pipeline code and Ansible script to the Jenkins working space for further operations. Sensitive details are encrypted with ansible-vault, so those details cannot be decrypted without the vault password. Vault password is stored in the Jenkins server to decrypt the Ansible script in the next steps. Jenkins server is having an only admin account, and disabled password authentication, so it is highly impossible to attack the system, obtain the vault password and reveal the secret information at any cost.

In the second stage, respective service will be installed in the corresponding VMs in the integration infrastructure by following the ansible script definitions. The remote VM is connected through a secure shell and performs each step to complete the installation in the PSNC integration environment provided for the specific service.

In the third stage, a basic smoke test is done in the integration environment to ensure the installation is correct. Most of the components are open-source and rolled out the product for many years, so checking accessibility by curl command is enough to guarantee the component functionality and quality. Cloudify[8] and Moodle[5] are tested with a basic smoke test with ping options to check the specific functionality after successful installation in the integration environment.

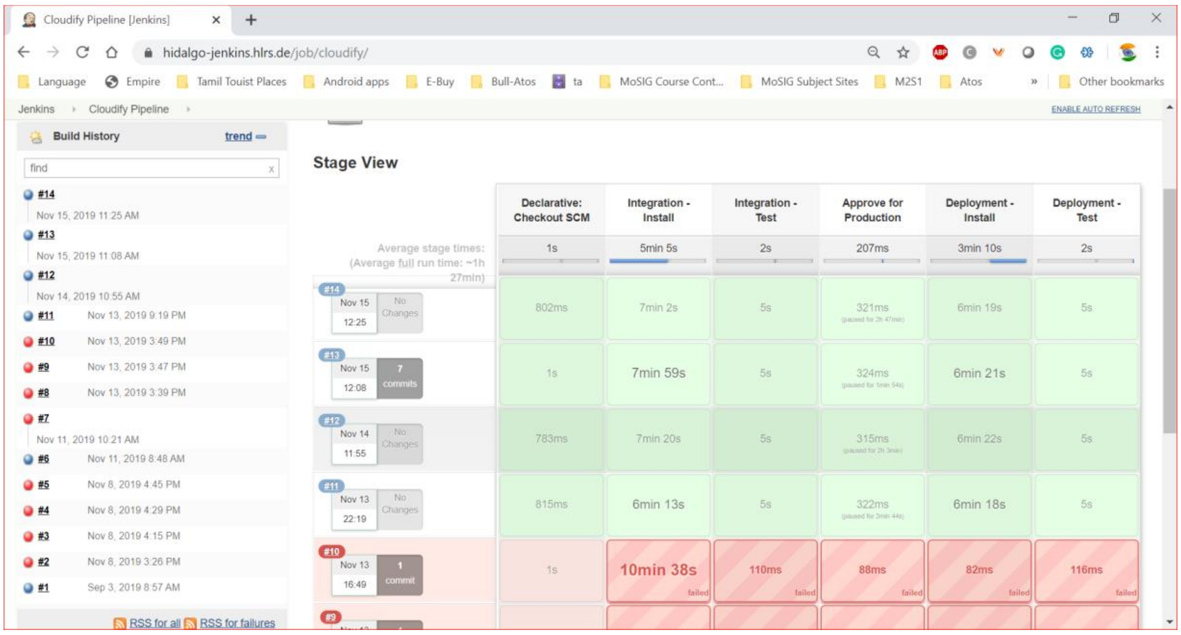
In the fourth stage, the pipeline waits for manual approval from the product owner or admin to proceed for the component deployment in the production infrastructure. This stage will be waiting for one day to get the confirmation from the owner otherwise the pipeline would be failed automatically and not deployed in the production infrastructure.

In the fifth stage, integrated and tested source code will be installed in the corresponding production VMs after getting manual confirmation from the product owner. Ansible script will be used to deploy in the deployment VM through a secure shell. All the components and Jenkins server are installed by following best-practices to enable SSL certificates, etc to ensure a secure portal access for the users. After the deployment, the product is ready for user access and supports real-time operations.

In the sixth stage, a basic smoke test is done in the deployment environment to ensure the installation is correct. Cloudify and Moodle are tested with a basic smoke test with ping to check the specific functionality after successful installation in the deployment environment.

If there is a failure in one of the stages, then the whole pipeline fails, and the complete process is aborted safely. Template of the basic pipeline is given in Appendix 1: Jenkins Pipeline Definition, which is written in the Groovy language. Jenkins graphical view of the Cloudify pipeline and its six stages is shown in Figure 2, and it is the same for other services (Matchmaking and Moodle) pipeline definition.

Jenkins pipeline and Ansible script are easy to reproduce in similar environments, so same Ansible script is used between integration and deployment infrastructure to install the components, as well as to maintain integrity and reproducibility. Jenkins pipeline and Ansible scripts are provided for Cloudify, Matchmaking, and Moodle with six stages as mentioned in Figure 3 and will be provided for other modules later in the project lifetime. Matchmaking manual test is done for different APIs by using the curl commands inside the Ansible scripts, so all sort of integration and testing are possible within the Jenkins pipeline. Cloudify croupier plugin unit test is done in the local system and its outcome (code coverage) will not change in the different infrastructure so that it is manually tested with ‘tux’ and ‘python 2.7’. Cloudify croupier plugin unit test and its code coverage is more than 60%, and it is detailed in Appendix 2: Cloudify Unit Test & Code Coverage. Cloudify croupier plugin unit test will be included in the Jenkins pipeline to automate the process.



Build	Date
#14	Nov 15, 2019 11:25 AM
#13	Nov 15, 2019 11:08 AM
#12	Nov 14, 2019 10:55 AM
#11	Nov 13, 2019 9:19 PM
#10	Nov 13, 2019 3:49 PM
#9	Nov 13, 2019 3:47 PM
#8	Nov 13, 2019 3:39 PM
#7	Nov 11, 2019 10:21 AM
#6	Nov 11, 2019 8:48 AM
#5	Nov 8, 2019 4:45 PM
#4	Nov 8, 2019 4:29 PM
#3	Nov 8, 2019 4:15 PM
#2	Nov 8, 2019 3:26 PM
#1	Sep 3, 2019 8:57 AM

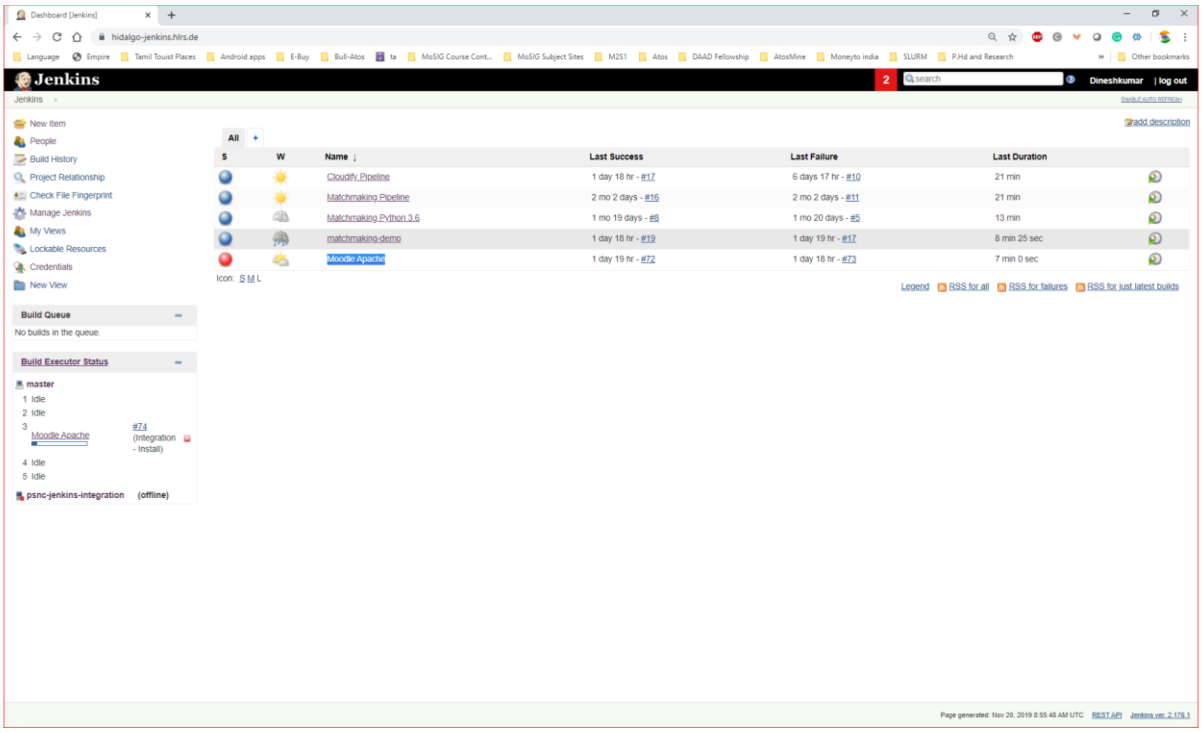
Stage View

	Declarative: Checkout SCM	Integration - Install	Integration - Test	Approve for Production	Deployment - Install	Deployment - Test
Average stage times: (Average full run time: ~1h 27min)	1s	5min 5s	2s	207ms	3min 10s	2s
#14 Nov 15 12:25 No Changes	802ms	7min 2s	5s	321ms (paused for 2h 47min)	6min 19s	5s
#13 Nov 15 12:08 7 commits	1s	7min 59s	5s	324ms (paused for 1min 54s)	6min 21s	5s
#12 Nov 14 11:55 No Changes	783ms	7min 20s	5s	315ms (paused for 2h 3min)	6min 22s	5s
#11 Nov 13 22:19 No Changes	815ms	6min 13s	5s	322ms (paused for 2min 44s)	6min 18s	5s
#10 Nov 13 16:49 1 commit	1s	10min 38s failed	110ms failed	88ms failed	82ms failed	116ms failed
#9						

Figure 2 – Graphical view of six stages Jenkins pipeline for Cloudify

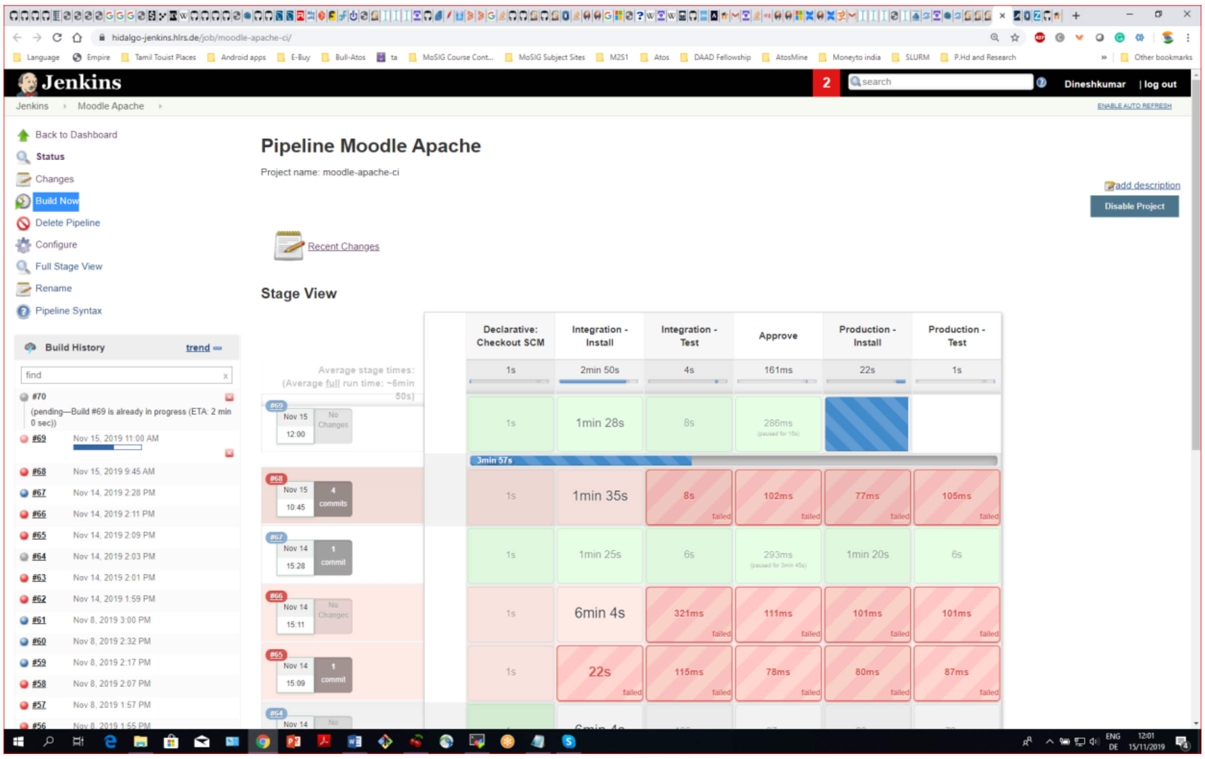
3.3 Jenkins Web Navigation & Usage

Jenkins provides graphical web support, so the developers can build their components easily after finalizing features for specific components or pushing minor changes to the source repository in the HLRS Git. Users can find multiple Jenkins projects after login to the Jenkins web server in the dashboard as shown in Figure 3, and view each project in detail by clicking the corresponding project link. The users can start the Jenkins project by clicking the “Build Now” button in the specific project, which is shown in Figure 4 for Moodle service. After submitting the Jenkins project, developers can track the status of the submission at every stage and analyse failures logs to understand the reasons behind the failure as shown in Figure 6. If the installation and test are successful in the Integration infrastructure, then it would wait for manual approval as mentioned in Figure 5 before proceeding to deploy in the production infrastructure. After getting the manual approval from the respective admin or developer to roll out the new feature in the production infrastructure. If there are any failures in any stages, then the remaining stages will be skipped, the pipeline job is aborted safely and the stages are displayed in red colour as shown in Figure 5 and Figure 6 for build number 65, 66 and 68; otherwise, all the stages would be in green colour and successfully rolled out the product to deployment infrastructure as shown in Figure 2 for build number from 11 to 14.



S	W	Name ↓	Last Success	Last Failure	Last Duration
●	☀	Cloudify Pipeline	1 day 18 hr - #17	6 days 17 hr - #10	21 min
●	☀	Matchmaking Pipeline	2 mo 2 days - #16	2 mo 2 days - #11	21 min
●	☁	Matchmaking Python 3.6	1 mo 19 days - #8	1 mo 20 days - #5	13 min
●	☁	matchmaking-demo	1 day 18 hr - #19	1 day 19 hr - #17	8 min 25 sec
●	⛅	Moodle Apache	1 day 19 hr - #72	1 day 18 hr - #73	7 min 0 sec

Figure 3 – List of Jenkins project in the dashboard after login



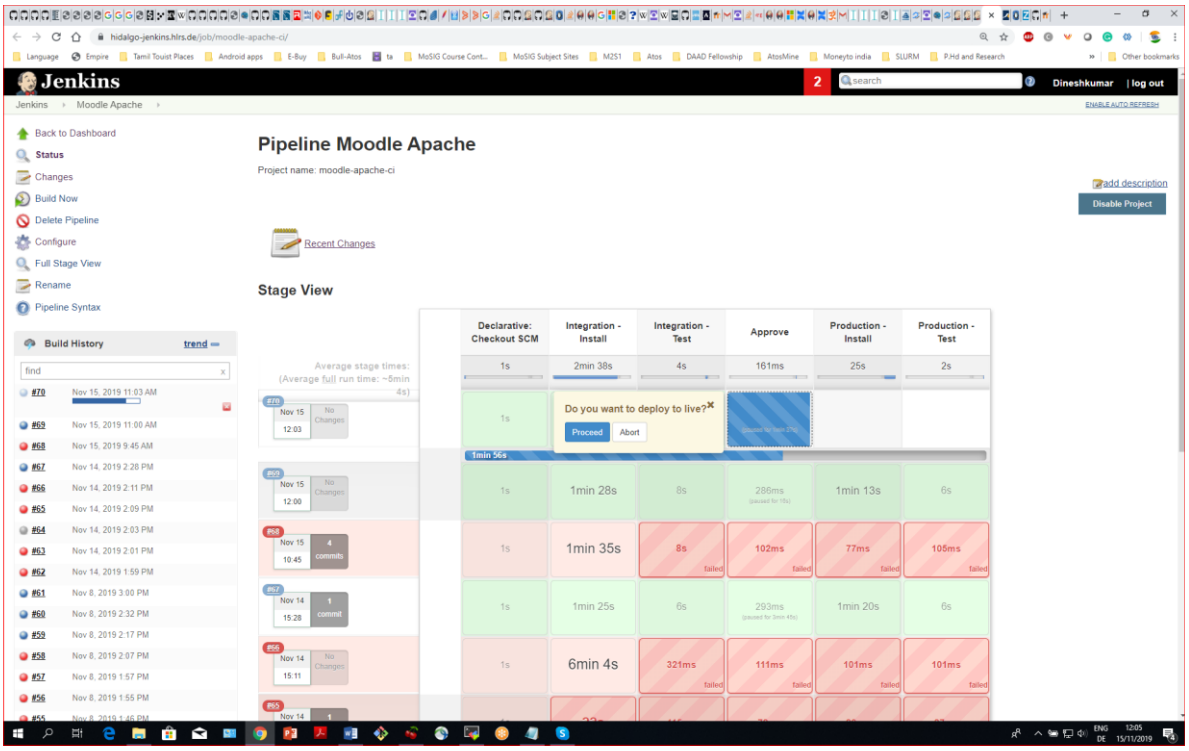
Pipeline Moodle Apache

Project name: moodle-apache-ci

Stage View

	Declarative: Checkout SCM	Integration - Install	Integration - Test	Approve	Production - Install	Production - Test
Average stage times: (Average full run time: ~6min 50s)	1s	2min 50s	4s	161ms	22s	1s
#69 Nov 15 12:00 No Changes	1s	1min 28s	8s	286ms		
#68 Nov 15 10:45 4 commits	1s	1min 35s	8s failed	102ms failed	77ms failed	105ms failed
#67 Nov 14 15:28 1 commit	1s	1min 25s	6s	293ms	1min 20s	6s
#66 Nov 14 15:11 No Changes	1s	6min 4s	321ms failed	111ms failed	101ms failed	101ms failed
#65 Nov 14 15:09 1 commit	1s	22s failed	115ms failed	78ms failed	80ms failed	87ms failed

Figure 4 – 'Build Now' button to submit the Jenkins job, and track the Jenkins pipeline progress.



Jenkins | 2 | search | Dineshkumar | log out

Jenkins › Moodle Apache › ENABLE AUTO REFRESH

- Back to Dashboard
- Status
- Changes
- Build Now
- Delete Pipeline
- Configure
- Full Stage View
- Rename
- Pipeline Syntax

Build History trend

#70	Nov 15, 2019 11:03 AM
#69	Nov 15, 2019 11:00 AM
#68	Nov 15, 2019 9:45 AM
#67	Nov 14, 2019 2:28 PM
#66	Nov 14, 2019 2:11 PM
#65	Nov 14, 2019 2:09 PM
#64	Nov 14, 2019 2:03 PM
#63	Nov 14, 2019 2:01 PM
#62	Nov 14, 2019 1:59 PM
#61	Nov 8, 2019 3:00 PM
#60	Nov 8, 2019 2:32 PM
#59	Nov 8, 2019 2:17 PM
#58	Nov 8, 2019 2:07 PM
#57	Nov 8, 2019 1:57 PM
#56	Nov 8, 2019 1:55 PM

Pipeline Moodle Apache

Project name: moodle-apache-ci

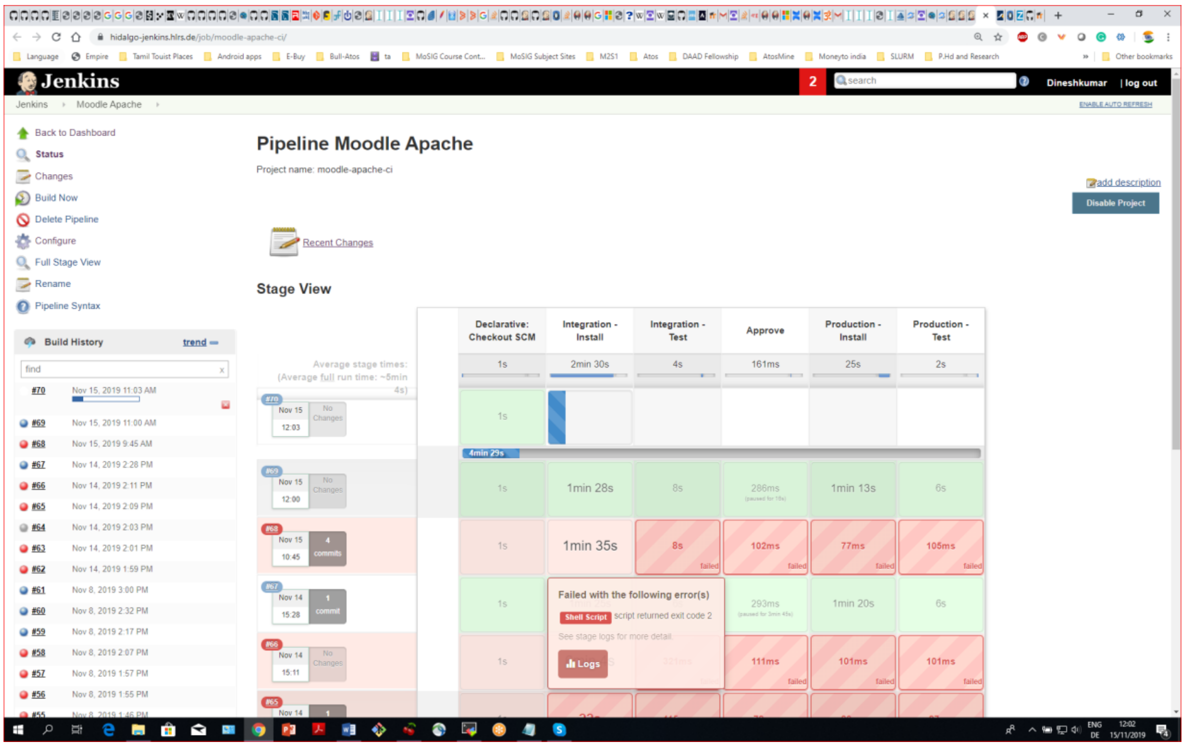
add description | Disable Project

Recent Changes

Stage View

	Declarative: Checkout SCM	Integration - Install	Integration - Test	Approve	Production - Install	Production - Test
Average stage times: (Average full run time: ~6min 4s)	1s	2min 38s	4s	161ms	25s	2s
#70 Nov 15 12:03 No Changes	1s	Do you want to deploy to live? Proceed Abort				
#69 Nov 15 12:00 No Changes	1s	1min 28s	8s	286ms	1min 13s	6s
#68 Nov 15 10:45 4 commits	1s	1min 35s	8s failed	102ms failed	77ms failed	105ms failed
#67 Nov 14 15:28 1 commit	1s	1min 25s	6s	293ms	1min 20s	6s
#66 Nov 14 15:11 No Changes	1s	6min 4s	321ms failed	111ms failed	101ms failed	101ms failed

Figure 5 – Jenkins Pipeline will wait for manual approval after the successful installation and test in the Integration infrastructure.



Jenkins | 2 | search | Dineshkumar | log out

Jenkins › Moodle Apache › ENABLE AUTO REFRESH

Pipeline Moodle Apache

Project name: moodle-apache-ci

Stage View

	Declarative: Checkout SCM	Integration - Install	Integration - Test	Approve	Production - Install	Production - Test
Average stage times: (Average full run time: ~6min 4s)	1s	2min 30s	4s	161ms	25s	2s
#70 Nov 15 12:03 No Changes	1s					
#69 Nov 15 12:00 No Changes	1s	1min 28s	8s	286ms	1min 13s	6s
#68 Nov 15 10:45 4 commits	1s	1min 35s	8s failed	102ms failed	77ms failed	105ms failed
#67 Nov 14 15:28 1 commit	1s			293ms	1min 20s	6s
#66 Nov 14 15:11 No Changes	1s			111ms failed	101ms failed	101ms failed

Failed with the following error(s)

Shell Script script returned exit code 2

See stage logs for more detail

Logs

Figure 6 – Jenkins pipeline failure and its reason can be analysed with logs when hovering over the failure stage.

4 Single Sign On

4.1 Implemented Solution

In order to enable Single Sign On (SSO) in the HiDALGO portal, the open source solution Keycloak has been used. Keycloak[7] is an IdAM backed by Red Hat and allows to easily add authentication and authorization to services and applications using standard protocols: OAuth, OpenID Connect (OIDC)[10], which runs on top of OAuth 2.0, and SAML[9]. Keycloak also offers additional features such as Social Login or LDAP support. In this project, the authentication process will be built on top of OIDC instead of SAML, since OIDC was designed with web use in mind. A future release of the Portal may include the possibility of using Social Login.

Keycloak uses realms to manage users and resources. From the keycloak documentation: "A realm manages a set of users, credentials, roles, and groups. A user belongs to and logs into a realm. Realms are isolated from one another and can only manage and authenticate the users that they control."[7]. A HiDALGO realm has been created, with different clients for each of the different services offered; currently this features the frontend and the backend but will be extended in the future to other services such as the Moodle or the MatchMaker.

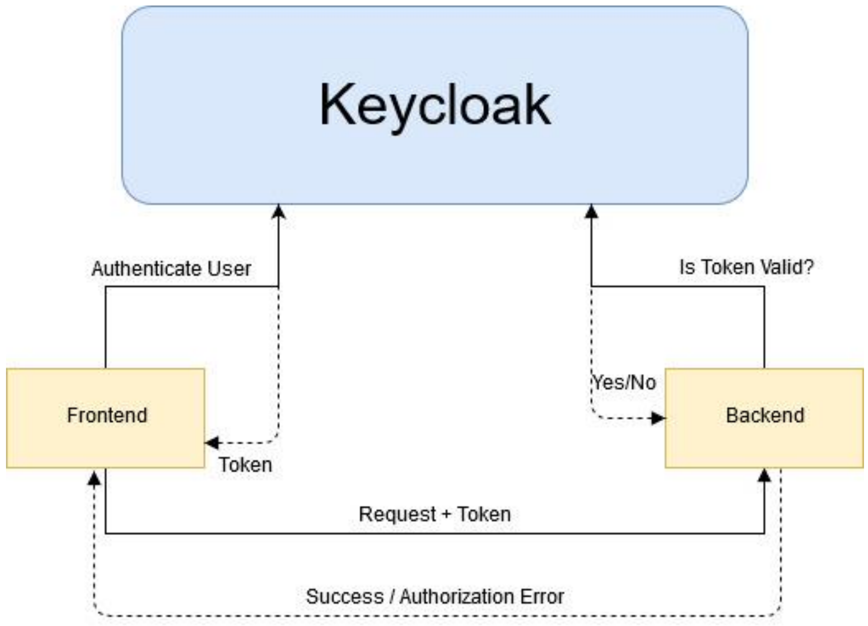


Figure 7 – Action Flow using Keycloak as the IdAM alongside several services

Basically, it provides some interfaces, so other components can invoke them in order to check the credentials of the user who wants to use a functionality. If the user has the access granted, it releases a security token that is used, later, by the component implementing the functionality to check whether the token is valid and the user can execute the functionality (no matter whether it is a web service or a full application like Moodle or Askbot).

4.2 Available APIs

Keycloak offers a web console to configure your realms, create and manage users, establish roles, permissions and resources, etc.

Keycloak exposes its functionality via an OIDC[10] endpoint; as such, connections are performed using an HTTP REST API, using JSON as the data format. Since Keycloak is an OIDC compliant IdAM, generic OIDC libraries are supported, while keycloak-specific adapters for popular platforms such as client-side Javascript are also offered. The following helper libraries where used:

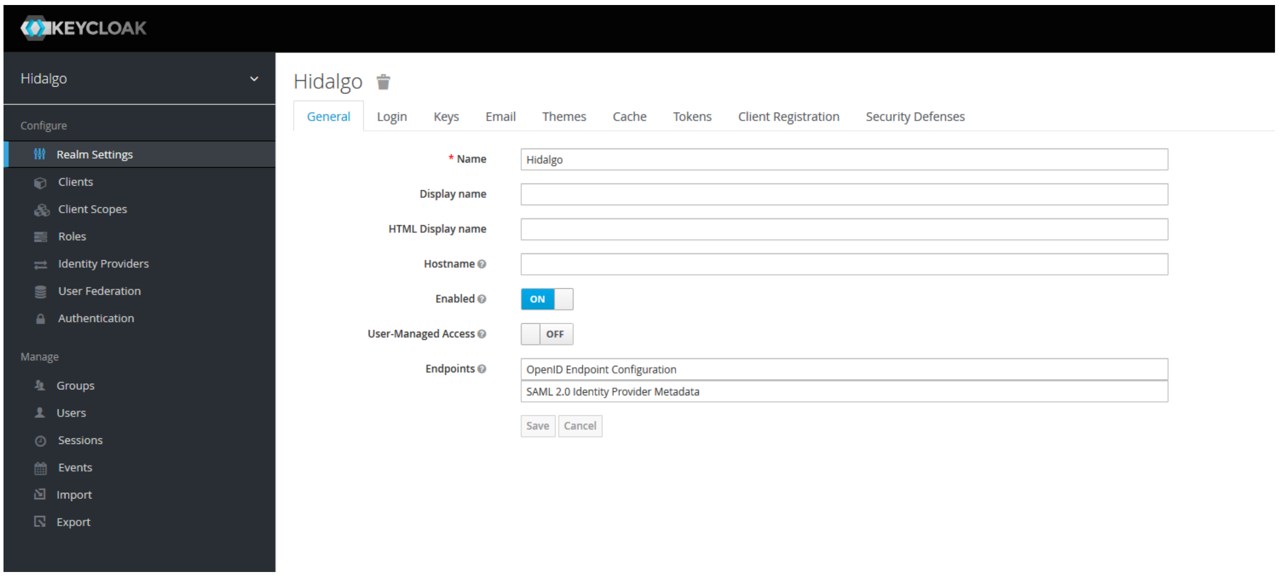
- *Keycloak-angular:* In order to use SSO in the portal frontend, this library was used, which allows an easy keycloak use in Angular applications
- *Mozilla-Django-OIDC:* This library, made by Mozilla, allows developers to connect to an OIDC endpoint using Django. It has been used in the Portal Backend to authenticate HiDALGO services.

In this project, no raw HTTP requests were performed to access Keycloak functionality. Instead, adapters were used to allow for better programmability.

4.3 How to Use and Examples

In Figure 8, the Keycloak Administration Console is shown. All aspects of a realm can be configured from this console, as well as administration settings itself.

There are two options to sign in to a Keycloak Realm. First, Keycloak provides a customizable login form shown in Figure 9 (the customization has not taken place yet), to which applications can redirect in order to log in. This is recommended because the login happens inside the Keycloak server and thus the applications have no access to the user credentials.


KEYCLOAK

Hidalgo

Configure: Realm Settings, Clients, Client Scopes, Roles, Identity Providers, User Federation, Authentication

Manage: Groups, Users, Sessions, Events, Import, Export

Hidalgo

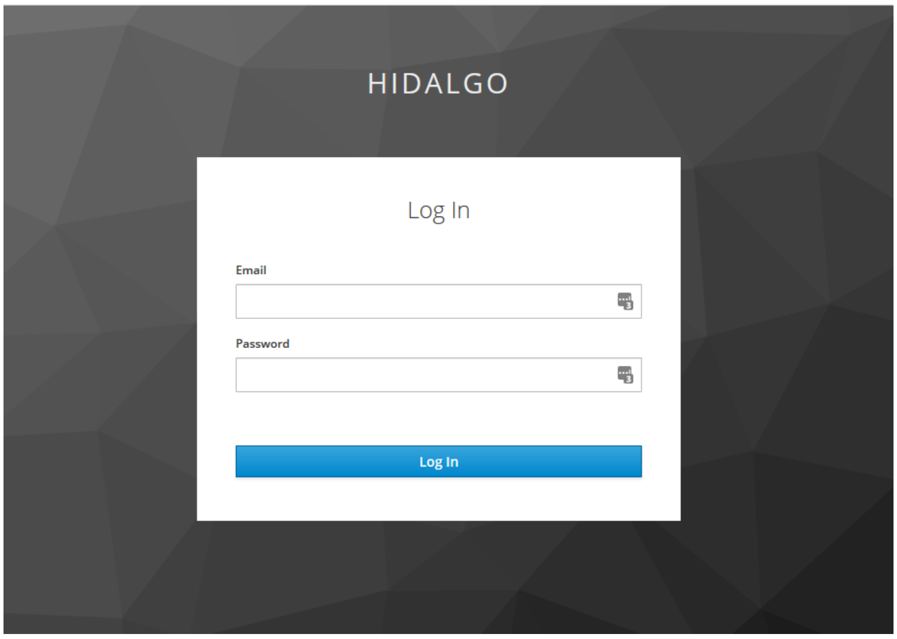
General | Login | Keys | Email | Themes | Cache | Tokens | Client Registration | Security Defenses

* Name	Hidalgo
Display name	
HTML Display name	
Hostname	
Enabled	ON
User-Managed Access	OFF
Endpoints	OpenID Endpoint Configuration SAML 2.0 Identity Provider Metadata

Save | Cancel

Figure 8 – Keycloak Administration Console

The other option is to send a POST HTTP request using the “Direct Access” Flow with the user credentials. The first option will be used in the HiDALGO portal.


HIDALGO

Log In

Email

Password

Log In

Figure 9 – Login form provided by Keycloak

5 Workflows Orchestration

5.1 Implemented Solution

In order to orchestrate the Workflows designed for each use case, the Cloudify[8] framework has been used. Per the Cloudify documentation: "Cloudify is an open source cloud orchestration framework. Cloudify enables you to model applications and services and automate their entire life cycle, including deployment on any cloud or data centre environment, monitoring all aspects of a deployed application, detecting issues and failure, manually or automatically remediating such issues, and performing ongoing maintenance tasks."[8].

Cloudify uses a Domain-Specific Language (DSL) based on TOSCA[11], a standard language to describe a topology of cloud-based web services, their components, relationships and its management during their lifetime [12]. Cloudify manages workflows using Blueprints, which consist each of one or more TOSCA files.

In order to orchestrate workflows in HPC environments, ATOS developed a Cloudify plugin called Croupier. Croupier extends the Cloudify functionality by providing custom node types used to describe HPC infrastructure interfaces or jobs that will run on the infrastructure. Additional functionality includes describing jobs that will be executed inside a Singularity container or expressing dependencies between Croupier jobs. An example of a Cloudify Blueprint using the Croupier extensions can be found in the Annex 3. This Blueprint describes four jobs that should run in an HPC node, either directly in hardware or using Singularity. Also, the user can specify dependencies between jobs, the resources needed, and scripts required to initialize the jobs, if required. More example Blueprints can be found at the repository https://github.com/ari-apc-lab/croupier-resources

Using Cloudify will allow scientists to easily run their applications across the HiDALGO environment without needing knowledge about deployments or DevOps.

5.2 Available APIs

A Django backend has been developed using the Django-Rest-Framework (DRF) that, along with the croupier plugin, enables cloudify to orchestrate HPC and batch jobs. This backend has been secured with the Keycloak IdAM to ensure only authorized users have access to the computation resources.

Cloudify can be accessed either via a web console, or using the cloudify CLI which, in turn, calls an HTTP REST API that exposes the Cloudify functionality. In this project, the Python cloudify-rest-client has been used to integrate Cloudify in the backend.

In the HiDALGO Portal, there are two important concepts: Applications and Instances. Applications are created based on a Cloudify Blueprint, and Instances are individual executions of an Application given some inputs and resources; hence, multiple Instances of an Application can coexist.

The backend API that will be accessed by the Portal Frontend in order to upload Blueprints and execute Instances consists of the following REST endpoints:

REST API endpoints	API Description
`apps`	Accepts several REST Verbs. • Get all the applications - Http GET • Create a new application - Http POST • Update an existing application – Http PUT
`apps/{id}`	Used to manipulate or obtain a specific question, given its identifier. Accepts several REST Verbs. • Get a specific application - Http GET • Delete the specific application - Http DELETE
`apps/{term}`	Used to obtain applications whose name contains the term searched. • Get all the matching applications - Http GET
`instances/{appid}`	Accepts several REST Verbs. • Get all the instances from an Application - Http GET • Create a new instance of an Application - Http POST • Update an existing instance of an Application – Http PUT
`instances/{id}`	Used to manipulate or obtain a specific instance, given its identifier. Accepts several REST Verbs. • Get a specific instance - Http GET • Delete the specific instance - Http DELETE
`instances/{term}`	Used to obtain instances whose name contains the term searched. Get all the matching instances - Http GET
`instances/{id}/events`	Obtains all the events related to an instance. List the events - Http GET

Table 5 API REST of the Orchestrator

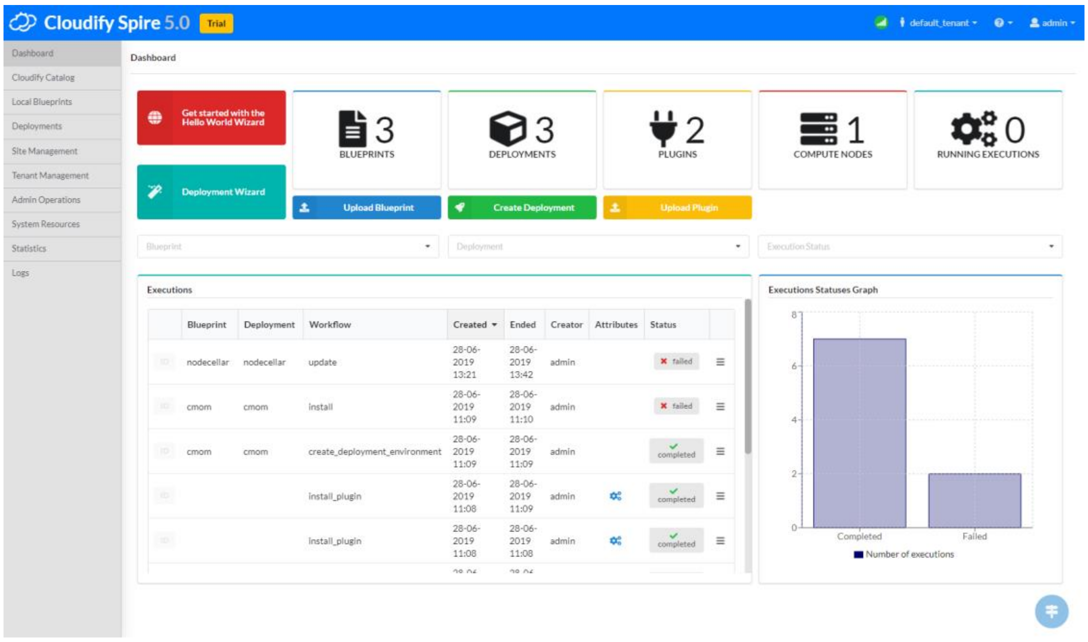
An example Cloudify Blueprint using the *croupier* plugin is at Annex 3, as mentioned before. More example Blueprints can be found at the repository https://github.com/ari-apc-lab/croupier-resources

5.3 How to Use and Examples

As mentioned before, the Cloudify server can be accessed via a web-based console. This console is shown in the Figure 10. This graphical interface allows an operator to, according to the Cloudify Documentation:

- Upload blueprints to the Cloudify Manager
- Deploy and install and remove new services
- Monitor and inspect the status of services
- View logs and events

It is important to note that all functions that can be performed using the Cloudify Console can be done using either the CLI/Python libraries, since they both use the Cloudify REST API.



Executions

	Blueprint	Deployment	Workflow	Created	Ended	Creator	Attributes	Status
	nodecellar	nodecellar	update	28-06-2019 13:21	28-06-2019 13:42	admin		✗ failed
	cmom	cmom	install	28-06-2019 11:09	28-06-2019 11:10	admin		✗ failed
	cmom	cmom	create_deployment_environment	28-06-2019 11:09	28-06-2019 11:09	admin		✓ completed
			install_plugin	28-06-2019 11:08	28-06-2019 11:09	admin	⚙	✓ completed
			install_plugin	28-06-2019 11:08	28-06-2019 11:08	admin	⚙	✓ completed

Figure 10 – Cloudify Web Console main dashboard

In the left menu, the user can upload new Blueprints, either from their local computer or from a catalog. Once a Blueprint is uploaded, a new Deployment can be created, which generates a physical representation from the logical Blueprint. Note that creating a deployment does not create resources, for the resources to be reserved, the install workflow must be executed. More information about Deployments and executing Workflows can be found in the Cloudify documentation [8].

In the following table more detailed examples appear on how to use the backend API, and the expected responses.

HTTP Requests	Response
`curl $API/apps/`	[{"id": 0, "name": "Cholesky", "description": "Performs a Cholesky decomposition", "owner": 0}, {"id": 1, "name": "Heat Transfer", "description": "Performs a heat transfer simulation using the Gauss-Seidel method", "owner": 0}]
`curl $API/apps/0`	{"id": 0, "name": "Cholesky", "description": "Performs a Cholesky decomposition", "owner": 0}
`curl -d '{"name": "Word Count", "description": "Performs a Word Count on an input", "owner": 0}'-X POST $API/apps/` Note: The Blueprint file would be encoded in the body of the request	{"id": 2, "name": "Word Count", "description": "Performs a Word Count on an input", "owner": 0}

Table 6 Usage examples for the Orchestrator REST API

6 Training

6.1 Implemented Solution

Training is one of the services in the HiDALGO portal. The purpose is to disseminate the training materials and build a strong community around the generated knowledge, so we identified and evaluated some open-source tools to achieve the expected results. Moodle[5] is a well-known open-source eLearning management system developed for conducting courses online, so it is the perfect platform to provide training for the HiDALGO end-users. Moodle's latest version V3.7 is installed in the production infrastructure with the rudimentary functionalities to integrate with the portal. Following features are supported in the installed Moodle version.

1. Dashboard to summarise a list of courses and course notifications after Login.
2. Teacher, Student and Admin roles to define functionalities and privileges.
3. Urban Air Pollution (UAP) use cases training course template is provided.
4. The course instructor can upload the course files (maximum 2MB) to the course topic wise and link another web page as a hyperlink. The course instructor can add forum, feedback, assignment, questions and survey as shown in Figure 12 and Figure 13.
5. Student can enrol the course, download the materials and track the course progress.

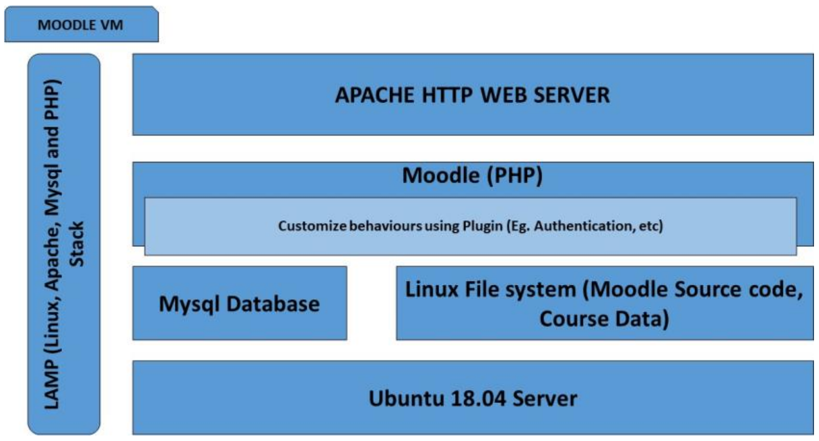

MOODLE VM
LAMP (Linux, Apache, Mysql and PHP) Stack
APACHE HTTP WEB SERVER
Moodle (PHP)
Customize behaviours using Plugin (Eg. Authentication, etc)
Mysql Database
Linux File system (Moodle Source code, Course Data)
Ubuntu 18.04 Server

Figure 11 – Moodle Software Stack

A Jenkins pipeline and an Ansible script are defined to deploy the components automatically and they activate the main functionalities to ensure the expected behaviour of our training system. Moodle is installed in a dedicated environment with the official sub-domain with LAMP (Linux, Apache, Moodle and PHP) software stack as shown in Figure 11.

Moodle has various plugins and configurations to enhance its default behaviour by providing more advanced features. The OAuth2 authorization configuration will be enabled in the Moodle to achieve expected Single Sign On (SSO) behaviour with Keycloak IDM. Different roles and responsibilities will be assigned automatically during the user registration by following the OAuth2 protocol. Course report and student feedback will be collected regularly and will used to improve the services throughout the project's lifetime.

Moodle source code is available on https://github.com/moodle/moodle

6.2 Available APIs

Moodle is a standalone application with its own GUI, so it is installed as an individual component with the separate domain name (https://moodle.hidalgo-project.eu/). The admin account has the capabilities to manage all the users, roles and courses details. The student account can be used to enrol in a course and to view course materials. The courses are organised with multiple topics and has different functionalities to include Quiz, Forum, Feedback, etc as shown in Figure 12.

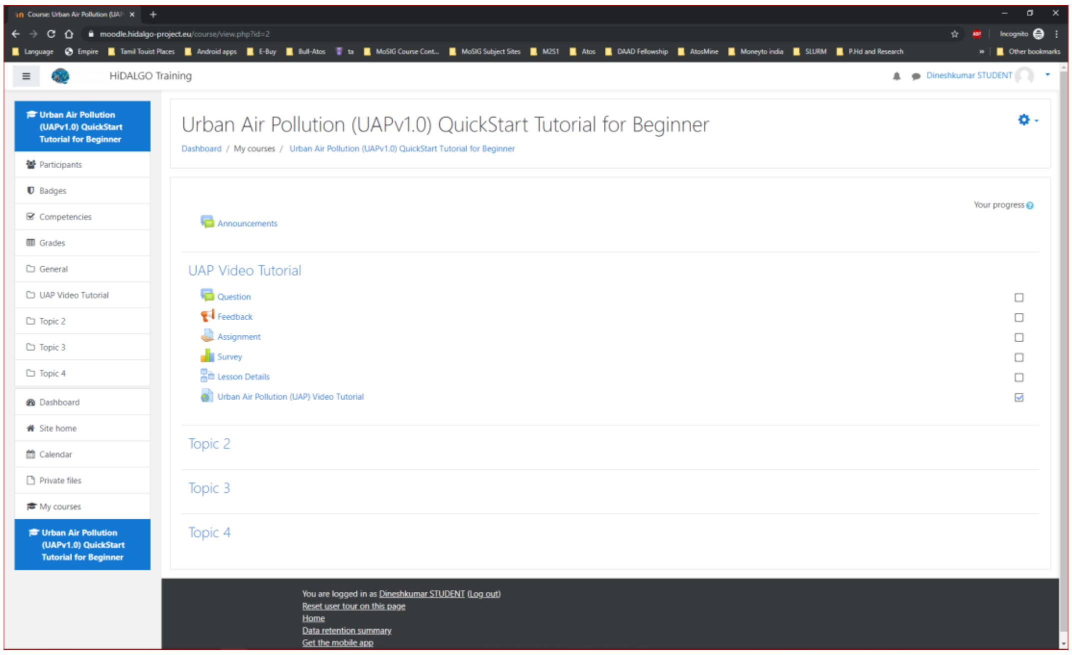

HiDALGO Training
Dineshkumar STUDENT
Urban Air Pollution (UAPv1.0) QuickStart Tutorial for Beginner
Participants
Badges
Competencies
Grades
General
UAP Video Tutorial
Topic 2
Topic 3
Topic 4
Dashboard
Site home
Calendar
Private files
My courses
Urban Air Pollution (UAPv1.0) QuickStart Tutorial for Beginner
Dashboard / My courses / Urban Air Pollution (UAPv1.0) QuickStart Tutorial for Beginner
Your progress
Announcements
UAP Video Tutorial
Question
Feedback
Assignment
Survey
Lesson Details
Urban Air Pollution (UAP) Video Tutorial
Topic 2
Topic 3
Topic 4
You are logged in as Dineshkumar STUDENT (Log out)
Reset user tour on this page
Home
Data retention summary
Get the mobile app

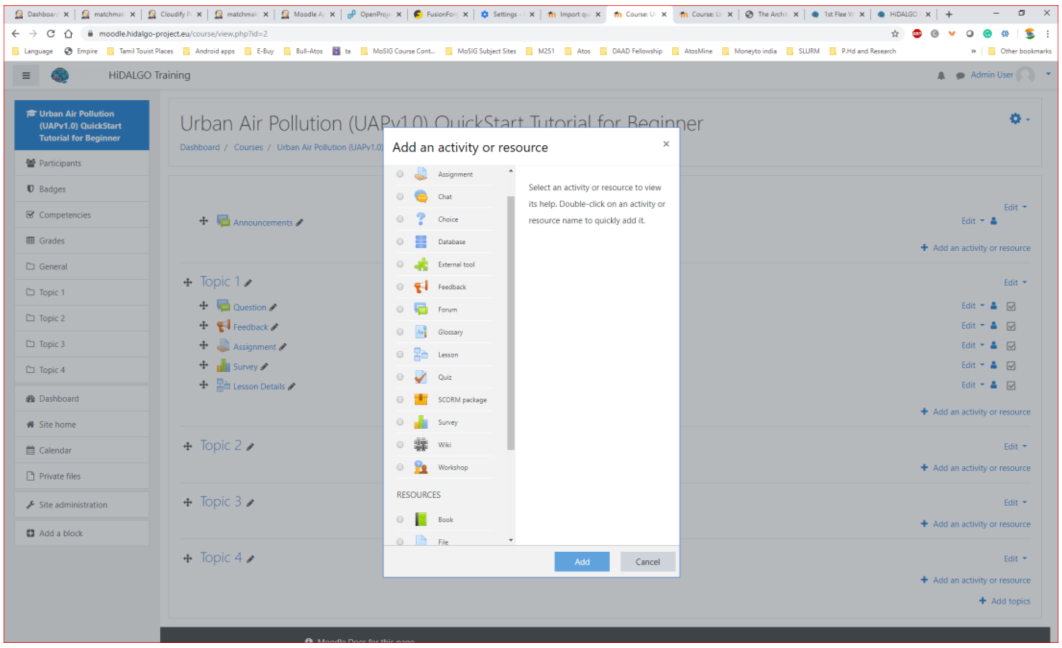
Figure 12 – Moodle course with multiple topics. Each topic with Course content, Question, Feedback, Assignment, and Survey, etc.

6.3 Moodle Usage

Moodle is a standalone and interactive web application, which is accessible on the Internet with a web browser. Moodle has its own login page with HiDALGO logo and has multiple roles (manager, course instructor, and student) to restrict the user access during the registration. The manager role has complete administrative access to manage the whole site (create a course, manage users and roles, etc), which is detailed in subsection 6.3.1. The student role has limited privileges to self-enrol the course and access the course materials, which is detailed in subsection 6.3.2. Moodle has multiple features and configurations, which is beyond the scope of this deliverable, so the main features are covered to get the overview of usage.

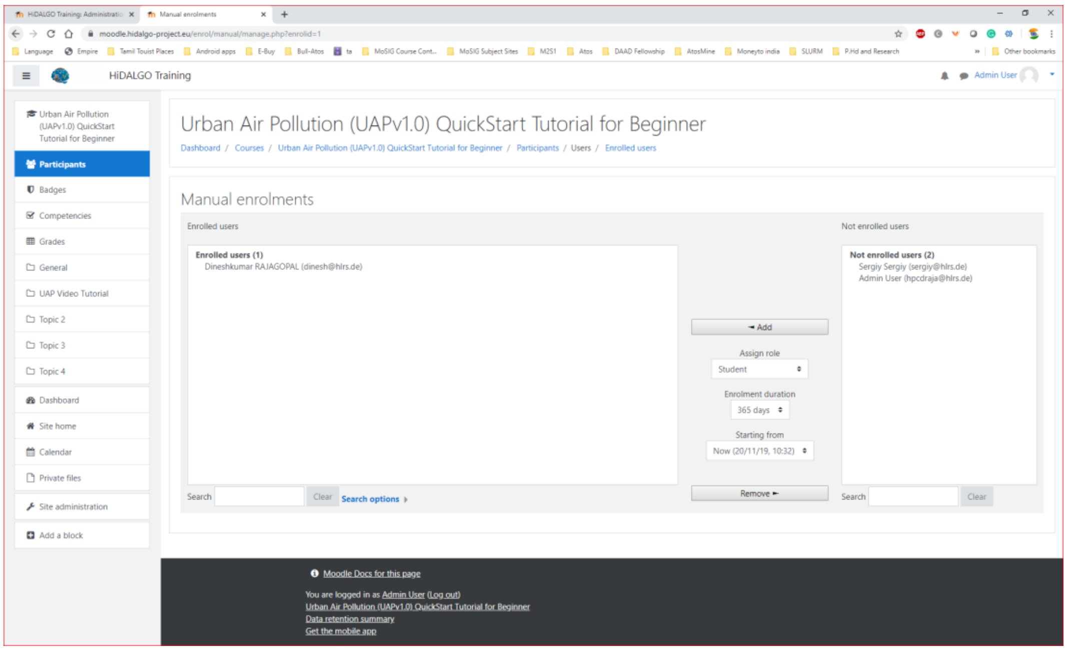
6.3.1 Manager and Course Instructor Functionalities

The manager has the complete admin access of the system, so they have a privilege to manage the users, change the configuration to enable new functionality, etc. The course instructor is the admin for a specific course, so they can customize the course content as mentioned in Figure 13 and manage student enrolment as mentioned in Figure 14.



Select an activity or resource to view its help. Double-click on an activity or resource name to quickly add it.

Figure 13 – Manager and Course instructor has the capabilities to edit the course content.



moodle.hidalgo-project.eu/enrol/manual/manage.php?enrolid=1
HiDALGO Training
Admin User
Urban Air Pollution (UAPv1.0) QuickStart Tutorial for Beginner
Participants
Badges
Competencies
Grades
General
UAP Video Tutorial
Topic 2
Topic 3
Topic 4
Dashboard
Site home
Calendar
Private files
Site administration
Add a block
Dashboard / Courses / Urban Air Pollution (UAPv1.0) QuickStart Tutorial for Beginner / Participants / Users / Enrolled users
Manual enrolments
Search
Clear
Search options
Remove
Moodle Docs for this page

Enrolled users

Enrolled users (1)
Dineshkumar RAJAGOPAL (dinesh@hlrs.de)

Assign role: Student
Enrolment duration: 365 days
Starting from: Now (20/11/19, 10:32)

Not enrolled users

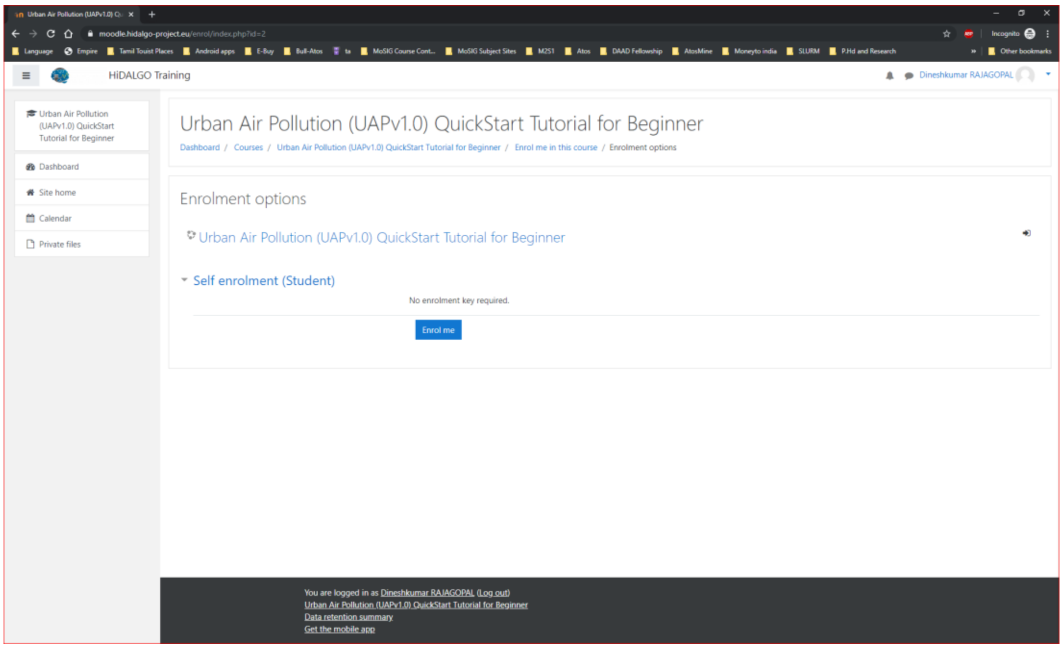
Not enrolled users (2)
Sergiy Sergiy (sergiy@hlrs.de)
Admin User (hpcdraja@hlrs.de)

You are logged in as Admin User (Log out)
Urban Air Pollution (UAPv1.0) QuickStart Tutorial for Beginner
Data retention summary
Get the mobile app

Figure 14 – Manage student enrolment manually by admin and Course Instructor

6.3.2 Student Functionalities

Students can self-enrol as mentioned in Figure 15 and can track the progress of course completion as shown in Figure 16.



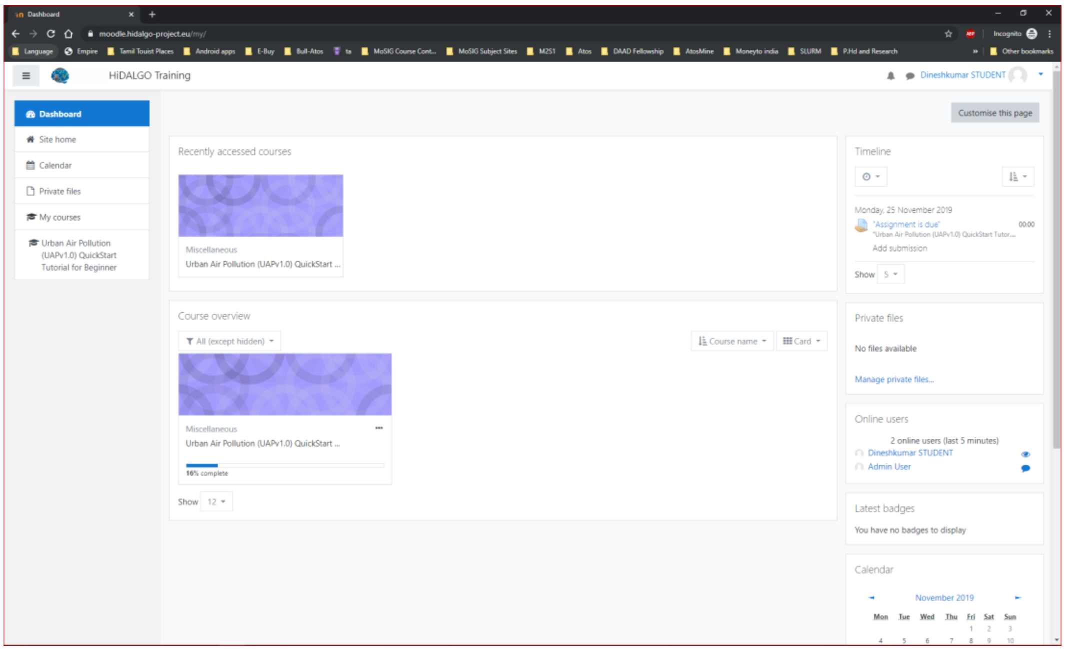
HiDALGO Training
Dineshkumar RAJAGOPAL
Urban Air Pollution (UAPv1.0) QuickStart Tutorial for Beginner
Dashboard
Site home
Calendar
Private files
Dashboard / Courses / Urban Air Pollution (UAPv1.0) QuickStart Tutorial for Beginner / Enrol me in this course / Enrolment options
Enrolment options
Self enrolment (Student)
Enrol me

Urban Air Pollution (UAPv1.0) QuickStart Tutorial for Beginner

No enrolment key required.

You are logged in as Dineshkumar RAJAGOPAL (Log out)
Urban Air Pollution (UAPv1.0) QuickStart Tutorial for Beginner
Data retention summary
Get the mobile app

Figure 15 – Students can self-enrol by clicking "Enrol me" button.



Monday, 25 November 2019
"Assignment is due" 00:00
"Urban Air Pollution (UAPv1.0) QuickStart Tutor...
Add submission

No files available

2 online users (last 5 minutes)
Dineshkumar STUDENT
Admin User

You have no badges to display

Figure 16 – Students can track the course progress, calendar and other notification in the dashboard

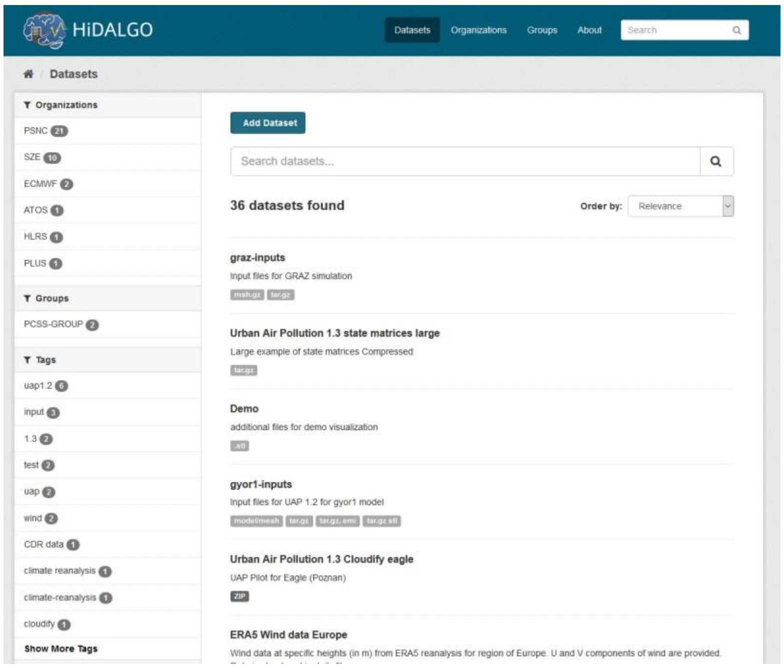
7 Data Catalogue

7.1 Implemented Solution

Contemporary IT systems have to execute applications based upon workflows, supporting complex scenarios conducted in order to get some experimental results. Since many of them are data-oriented processing environments, one of the fundamental components in this case is the data catalogue. Data Catalogue (DC) is considered as Data Management System (DMS) which simplifies data discovery by users, offering a complete set of services. These services provide users with information what datasets are available, complementary description in the form of metadata, search and browsing functionality, organizing datasets along with organizations and defining different access levels to data. Moreover, they implement different access methods either as web interfaces for human interaction or Application programming Interfaces to enable automation by service-to-service interaction.

In the HiDALGO project the role of DC is played by the Comprehensive Knowledge Archive Network (CKAN)[6], which is integrated with the HiDALGO Portal.

The current version of the Data Manager component is running on a set of virtual machines with Ubuntu 18.04 hosted at the Poznań Supercomputing and Networking Center and can be accessed via the address https://hidalgo1.man.poznan.pl/. The CDS harvester functionality can be accessed through this URL: https://hidalgo1.man.poznan.pl/cds/



PSNC 21
SZE 10
ECMWF 2
ATOS 1
HLRS 1
PLUS 1

PCSS-GROUP 2

uap1.2 6
input 3
1.3 2
test 2
uap 2
wind 2
CDR data 1
climate reanalysis 1
climate-reanalysis 1
cloudify 1

36 datasets found

graz-inputs
Input files for GRAZ simulation
mesh.gz tar.gz

Urban Air Pollution 1.3 state matrices large
Large example of state matrices Compressed
tar.gz

Demo
additional files for demo visualization
.stl

gyor1-inputs
Input files for UAP 1.2 for gyor1 model
model/mesh tar.gz tar.gz, emi tar.gz stl

Urban Air Pollution 1.3 Cloudify eagle
UAP Pilot for Eagle (Poznan)
ZIP

ERA5 Wind data Europe
Wind data at specific heights (in m) from ERA5 reanalysis for region of Europe. U and V components of wind are provided.

Figure 17 – The CKAN system – datasets summary page

In order to increase the reliability of the data catalogue system, the standard CKAN[6] environment configuration was enhanced by a number of virtual machines and services to provide mirrored services which can complement each other in case of failure. Currently, it is composed of two HA Proxies, two CKAN instances including databases and files systems.

It is configured to deliver Bi-Directional Replication for PostgreSQL (Postgres-BDR, or BDR), which is the open source multi-master replication system for these databases to reach full production status. BDR is specifically designed for geographically distributed clusters, using highly efficient asynchronous logical replication, supporting anything from 2 to more than 48 nodes in a distributed database. Discussed configuration is presented on the picture below.

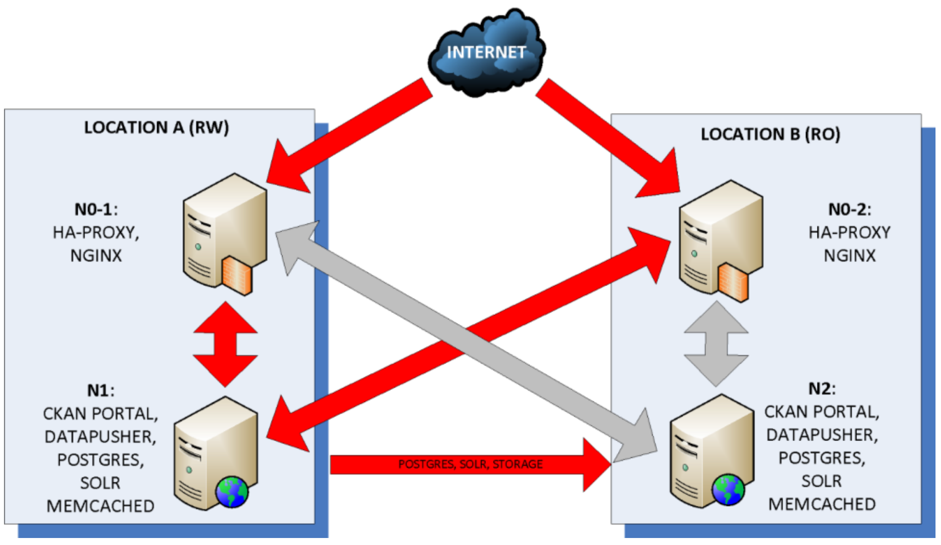


Figure 18 – The configuration of the enhanced CKAN environment

7.2 Available APIs

The data catalogue service in the HiDALGO project is integrated with other services in order to provide a complete bunch of functionalities. In this respect, a quite vital aspect of delivered functionality is the existing Application Programming Interface (API). It facilitates coupling of services into one collaborative system.

The CKAN system offers API capabilities in Remote Procedure Call (RPC) style (https://ckan.org/portfolio/api/). The API offered functionality is very wide and covers all functions which correspond to actions, which can be done using web the interface.

Security is guaranteed by authorization operation. Calling functions requires providing API key in an HTTP request in the form of either an Authorization or X-CKAN-API-Key header.

Document name:	D5.3 First HIDALGO Portal Release and System Operation Report					**Page:**	33 of 55
Reference:	D5.3	**Dissemination:**	PU	**Version:**	1.0	**Status:**	Final

The CKAN API is called as HTTP POST request to a relevant URL using JSON language. Both parameters and responses are provided in a JSON dictionary. Calling the API functions can be made either from command-line interface or supportive libraries available for various bunch of programming languages.

The response is a JSON dictionary with three keys:

- **success**: true or false, in case or “false” status details are provided in the "error" key,
- **result:** result of the called function, type and value depend on called function,
- **help**: string to the documentation for corresponding function.

Consistency with previous versions ensures API versioning. If not specified, the latest API version is used. Currently version 3 is the only available API version.

FileStore API

CKAN’s FileStore extension has its own API. In order to manipulate datasets and files the following API functions are used:

- resource_create() – creates a new resource in the CKAN and upload file
- resource_update() – updates already existing resource

Data and pairs (key, value) can be uploaded (JSON object) through the API. In order to post the binary data, the extra key upload must be used.

DataStore API

The DataStore in the CKAN is understood as database for collections of tables with unknown relationships. To create a new DataStore resource a corresponding CKAN resource (file) must exist first. DataStores can be previewed at the CKAN web site (recline extension) which facilitates user’s work. CKAN’s DataStore extension has its own API for data management without the need to download the entire file first.

Data can be written incrementally to the database table using the API. The following functions are available:

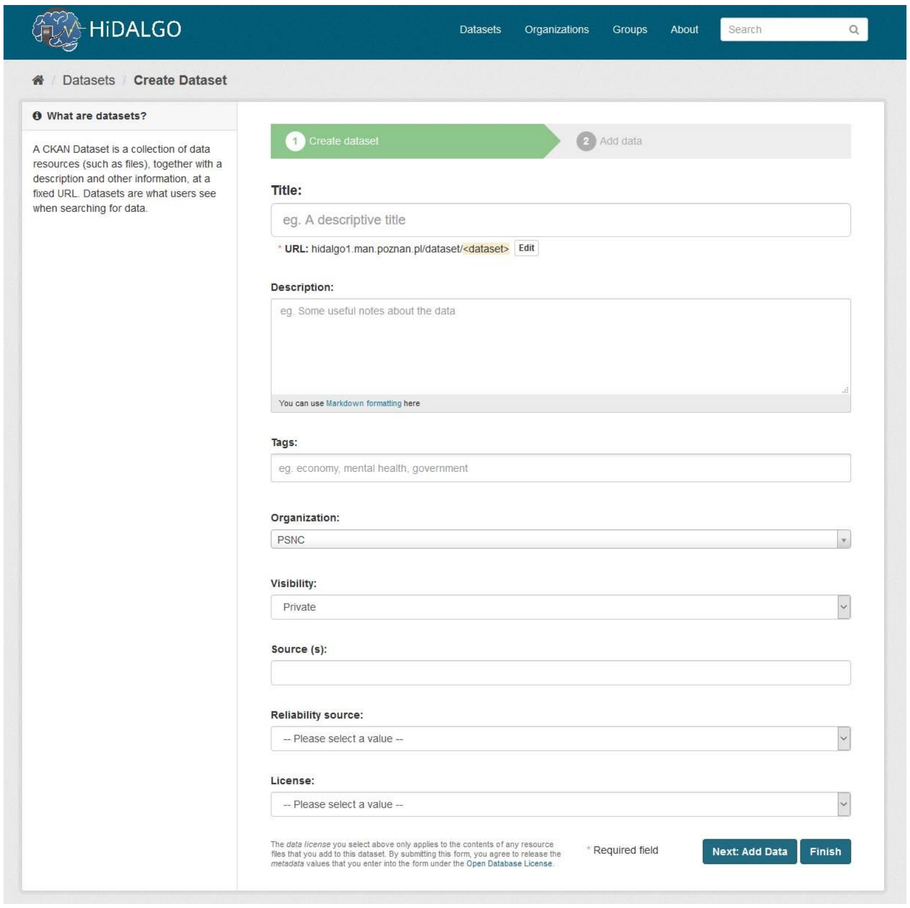
- insert - data a new row is added
- update – data in specific row are altered
- deleted – a specified row is deleted.
- add column – a new column is added to an existing table

An exemplary usage of DataStore API (adding a new table) is presented below:

```
ckanext.datastore.logic.action.datastore_create(context, data_dict)
```

7.3 How to Use and Examples

The CKAN system allows to organize stored files around the dataset concept, which is used to group one or more files with the corresponding metadata for describing the content (see Figure 19). Metadata provides relevant information like: author info, update releases, licensing scheme, etc. Thanks to the CKAN flexibility, the metadata scheme can be adjusted by developers to reflect current project needs. Properly described metadata datasets facilitate the exploration process. When a user performs a search on CKAN resources about a specific set of data, the results are shown as a list of datasets.



What are datasets?

A CKAN Dataset is a collection of data resources (such as files), together with a description and other information, at a fixed URL. Datasets are what users see when searching for data.

Title:	eg. A descriptive title
* URL:	hidalgo1.man.poznan.pl/dataset/<dataset> Edit
Description:	eg. Some useful notes about the data
	You can use Markdown formatting here
Tags:	eg. economy, mental health, government
Organization:	PSNC
Visibility:	Private
Source (s):	
Reliability source:	-- Please select a value --
License:	-- Please select a value --

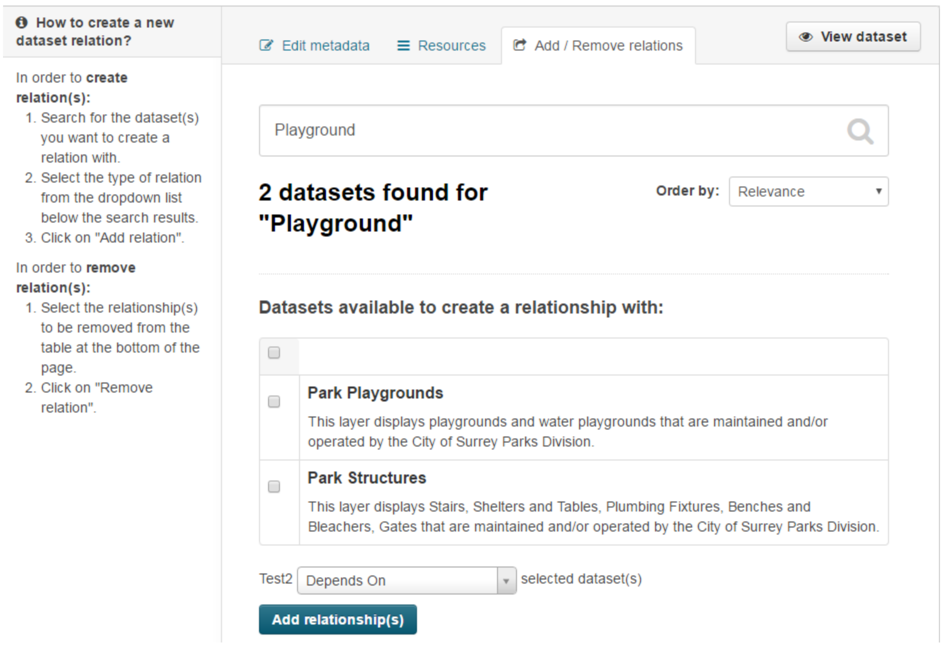
The *data license* you select above only applies to the contents of any resource files that you add to this dataset. By submitting this form, you agree to release the *metadata* values that you enter into the form under the Open Database License.

* Required field

Figure 19 – A new dataset creation in the CKAN system.

Thanks to the CKAN extension capabilities, the metadata information about the datasets can be improved and increased. One of them is `drel` which allows the creation and management of dataset relationships. Users are able to create the following types of relationships between the datasets: *derives from, has derivation, depends on, has dependency, linked from, links to, child of* and *parent of.* These relationships can be created during the dataset creation or from the management page of the dataset itself. The existing relationships can be viewed on the

Dataset Relations tab on the dataset's page, the related datasets can also be navigated from here.


How to create a new dataset relation?

In order to **create relation(s)**:
1. Search for the dataset(s) you want to create a relation with.
2. Select the type of relation from the dropdown list below the search results.
3. Click on "Add relation".

In order to **remove relation(s)**:
1. Select the relationship(s) to be removed from the table at the bottom of the page.
2. Click on "Remove relation".

Playground

2 datasets found for "Playground"

Order by: Relevance

Datasets available to create a relationship with:

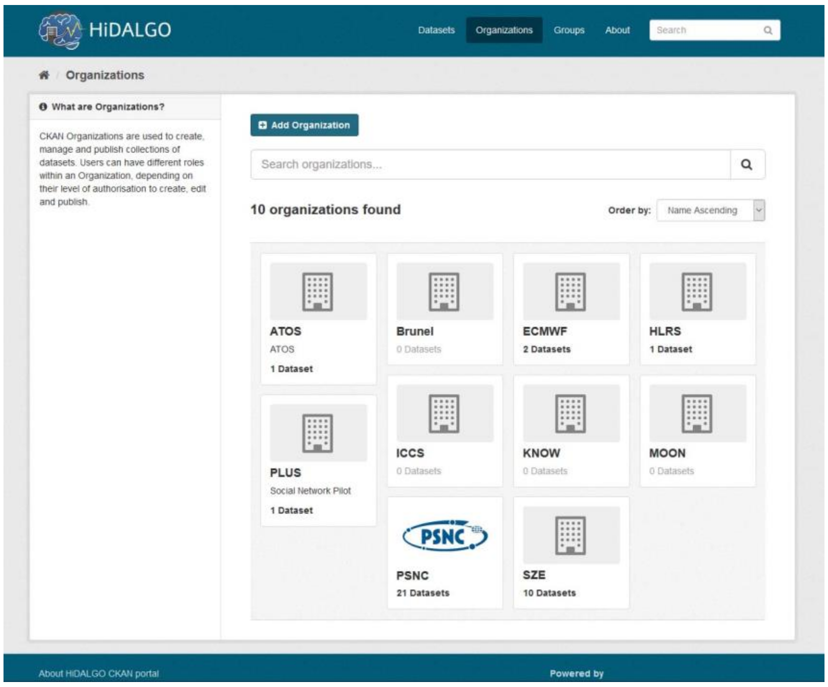
Park Playgrounds
This layer displays playgrounds and water playgrounds that are maintained and/or operated by the City of Surrey Parks Division.

Park Structures
This layer displays Stairs, Shelters and Tables, Plumbing Fixtures, Benches and Bleachers, Gates that are maintained and/or operated by the City of Surrey Parks Division.

Test2 Depends On selected dataset(s)

Figure 20 – Dataset relationship management in the CKAN.

CKAN grants creation of organisations which are collections of users with different levels of authorisation and a collection of datasets (private or public). For a better illustration please see Figure 21. Groups in the CKAN allow the creation of catalogues which could be focused on a certain subject(s) or may be a collection of datasets, which belong to a project or a team. Datasets can be managed (added, removed) only the group members.



What are Organizations?

CKAN Organizations are used to create, manage and publish collections of datasets. Users can have different roles within an Organization, depending on their level of authorisation to create, edit and publish.

10 organizations found

ATOS
ATOS
1 Dataset

Brunel
0 Datasets

ECMWF
2 Datasets

HLRS
1 Dataset

PLUS
Social Network Pilot
1 Dataset

ICCS
0 Datasets

KNOW
0 Datasets

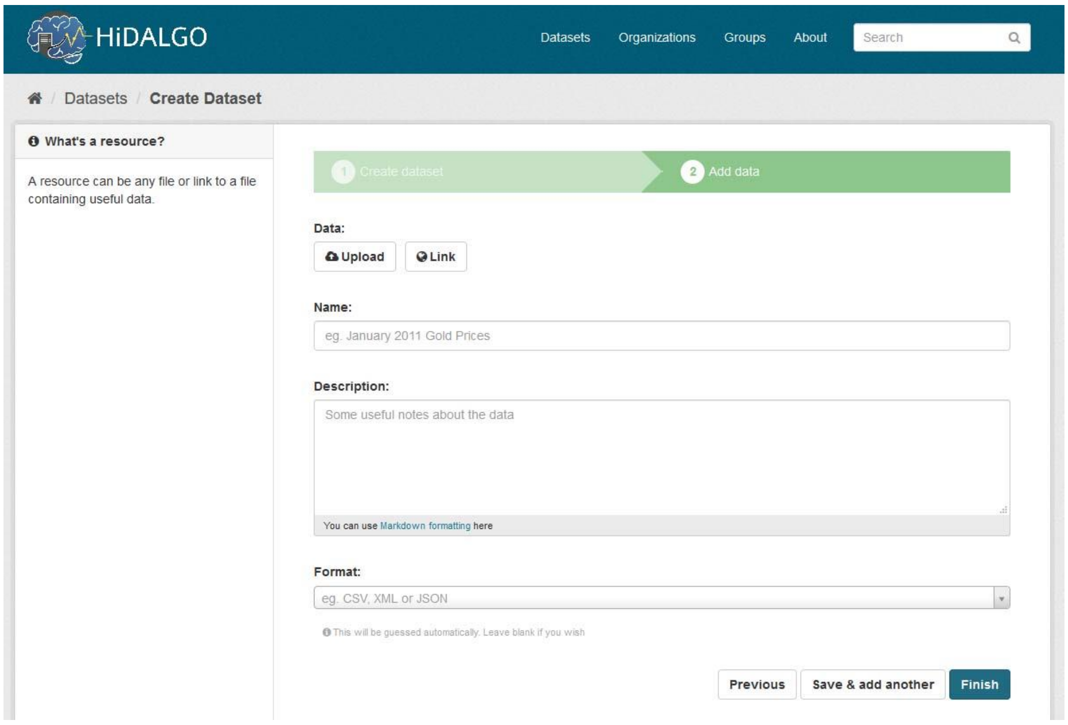
MOON
0 Datasets

PSNC
21 Datasets

SZE
10 Datasets

Figure 21 – Defining organizations in the CKAN system.

In order to feed structures with data, CKAN provides a harvesting mechanism that can be used to consume data either from user's local machine or from a remote system (Figure 22). Moreover, registering of other CKAN instance(s) as a source allows feeding metadata on defined basis (manual, daily, weekly, biweekly and monthly). Once the registration process is complete, CKAN starts pulling the metadata along with the resources to the local instance in the background.



What's a resource?

A resource can be any file or link to a file containing useful data.

Data:

Name:

eg. January 2011 Gold Prices

Description:

Some useful notes about the data

You can use Markdown formatting here

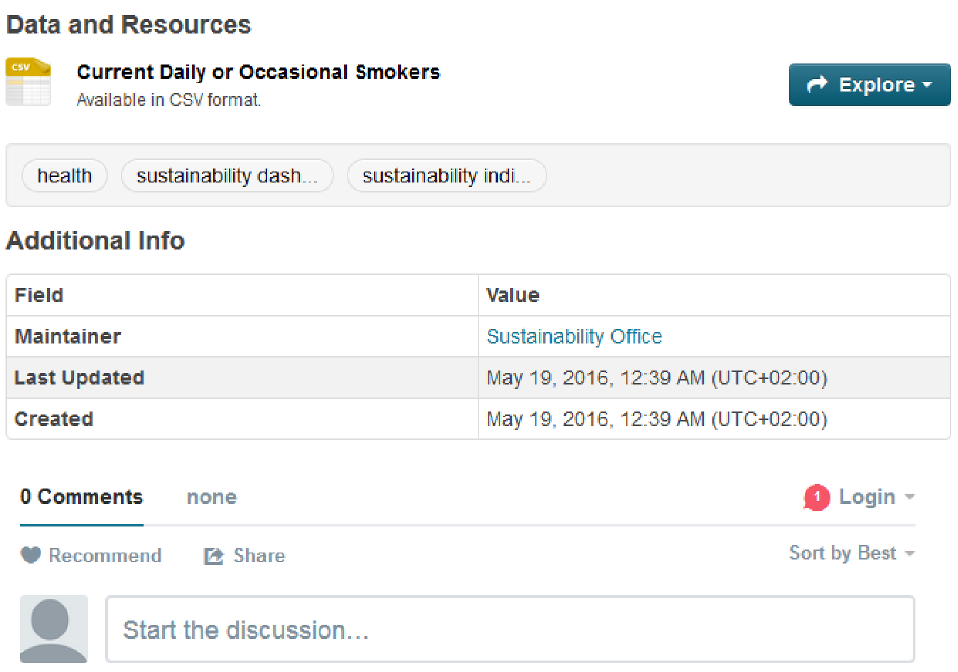
Format:

eg. CSV, XML or JSON

This will be guessed automatically. Leave blank if you wish

Figure 22 – CKAN data harvestation process.

The extension *disqus* adds the functionality that allows commenting on the existing datasets and provides a voting system for these comments (see Figure 23). It is also possible to leave comments as a guest by providing a name and an email address. Users are not required to be registered to Disqus itself in order to comment, since CKAN handles that.


Data and Resources

Current Daily or Occasional Smokers
Available in CSV format.

Additional Info

Field	Value
Maintainer	Sustainability Office
Last Updated	May 19, 2016, 12:39 AM (UTC+02:00)
Created	May 19, 2016, 12:39 AM (UTC+02:00)

Figure 23 – The disqus extension to the CKAN enables commenting feature.

Thanks to the openness and availability of source codes the CKAN enables implementation of customized modules (extensions). An exemplary development (CDS data harvester) is discussed and developed in the context of WP3.

8 Users Matchmaking

8.1 Implemented Solution

Matchmaking is a social networking service to strengthen the HiDALGO community by suggesting similarly interested peers and following them according to their interests. This service is developed and running as an individual web service by following the REST architecture. High-level architecture of the matchmaking and its corresponding REST APIs endpoints are detailed in Figure 24. Users profiles are managed with the details of liked users list and answers to the multiple-choice questions, which is saved in the PostgreSQL database. Matchmaking algorithm calculates the compatibility between two users based on their profiles' information. The algorithm pseudocode is detailed step by step below, which is based on the users' profile data and follows the geometric mean algorithm to provide a match percentage for the specific pair of users. If they answer more questions with similar answers, then their compatibility will increase, so the user has to answer all the questions to minimize error in the calculation.

```
1. Get answers and importance level from the users for all the
   multiple-choice questions
     a. Collect their own answer, importance level for each
        question
     b. Collect expected answers, importance level from others
        for each question
2. user_1_points, user_2_points <- 0
3. user_1_total_points, user_2_total_points <- 0,0000001
4. Convert importance level to importance level points for
   further numerical analysis
5. questions_in_common <- Get the common list of questions
   answered by user_1 and user_2
6. while question_c in questions_in_common:
     a. if user_1's answer for question_c is matched with
        user_2's expected answer for question_c
          i. user_1_points <- user_1_points + expected
             importance level points of user_2
     b. if user_2's answer for question_c is matched with
        user_1's expected answer for question_c
          i. user_2_points <- user_2_points + expected
             importance level points of user_1
     c. user_1_total_points <- user_1_total_points + expected
        importance level points of user_2
        d. user_2_total_points <- user_2_total_points + expected
           importance level points of user_1
  7. user_1_match_percent <- user_1_points/user_1_total_points
  8. user_2_match_percent <- user_2_points/user_2_total_points
  9. match_percentatge <-
     (1/questions_in_common) √(user_1_match_percent * user_2_match_percent)
  10.    return match_percentatge * 100
```

Multiple-choice questions and its answers are collected from users in line number 1 and then abstract answers are converted into numerical points by using the question's importance level in line number 4. Commonly answered questions are filtered in line number 5 based on the user_1 and user_2 profile data. Individual users' match percentage is calculated against another user's expected answer and their importance level as described from line number 6 to 8. Common match between partner is calculated by using the geometric mean of individual users' match percentage on line number 9.

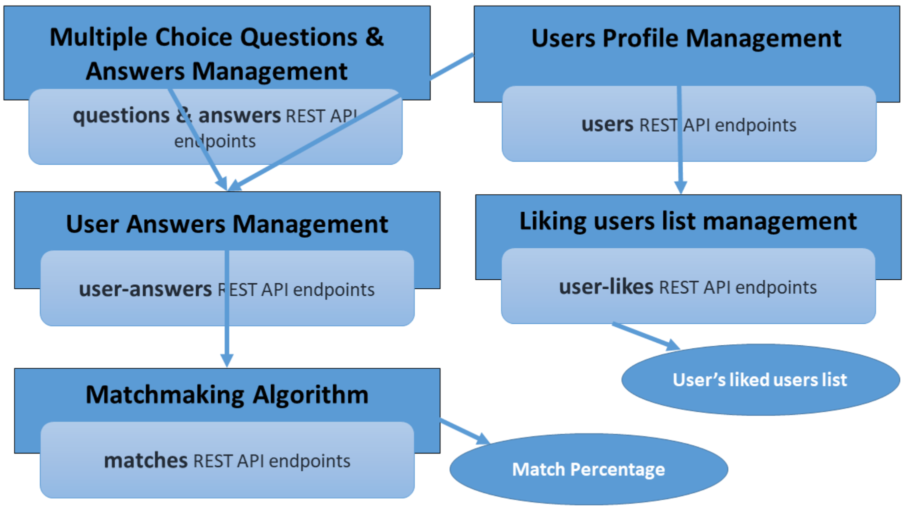

Multiple Choice Questions & Answers Management
questions & answers REST API endpoints
Users Profile Management
users REST API endpoints
User Answers Management
user-answers REST API endpoints
Liking users list management
user-likes REST API endpoints
User's liked users list
Matchmaking Algorithm
matches REST API endpoints
Match Percentage

Figure 24 – High-level architecture of matchmaking service

Python Django rest framework is used for developing the REST service, and each rest resource is defined as a Django application to improve maintainability. All the details in the REST APIs (Questions, User answers, Match percentage, etc) are stored in the PostgreSQL database for leveraging the benefits of database security and storing in long-term storage. JSON (JavaScript Object Notation) data is exchanged between the portal frontend and service to manage the data effectively and efficiently by following the industry standards. REST API and its behaviour are well defined and documented in the RAML (REST API Modelling Language) during the design phase to share with the portal developer. Matchmaking static web document is generated from the raml file automatically by using raml2html tool, which is giving interactive access to all the REST APIs and its usage for different HTTP requests as shown in F.

This service is deployed in a local virtual machine with the accessibility only to web portal backend for reducing vulnerabilities from the Internet. Jenkins pipeline with Ansible script is defined to deploy the service automatically upon the Ubuntu 18.04 server and launch the service with multiple process Gunicorn HTTP server to improve the responsiveness of each Http request. Postgresql database is used as the backend database to store all the matchmaking data, which is the default database to integrate easily with the Django rest framework.

The portal frontend will consume the service and render the JSON data in a graphical manner for end-user access. The Matchmaking tool will provide SSO authentication and manage user details in the Keycloak IDM.

Matchmaking Source code is available on https://github.com/hlrs-121991-germany/matchmaking_rest_user.

8.2 Available APIs

All the matchmaking APIs and its purpose are detailed in Table 7, which is related to the high-level architecture in Figure 24. User answers are collected by Questions, Answers and User-answers endpoints. User details are managed in the users' endpoint. Users relationship calculation is done by Matches API to measure the percentage of compatibility between users. Liked users' details are managed inside the User-likes endpoints.

Serial No	REST API endpoints	API Description
1	`questions`	Questions endpoint is used to do the following actions. • Get all the list of questions - Http GET • Create a new question - Http POST
2	`questions/{id}`	Questions/id endpoint is used to do the following actions on a specific question. Question is identified by question's identity number (id). • Get the specific question - Http GET • Delete the specific question – Http DELETE • Update the specific question – Http PUT
3	`answers`	Answers endpoint is used to do the following actions. • Get all the list of answers - Http GET • Create a new answer – Http POST

4	`answers/{id}`	Answer endpoint is used to do the following actions on a specific answer. Answer is identified by answer's identity number (id). • Get the specific answer - Http GET • Update the specific answer - Http PUT • Delete the specific answer – Http DELETE
5	`user-answers`	The user-answers endpoint is used to do the following actions. • Get all the list of user's answer - Http GET • Create a new user's answer - Http POST
6	`user-answers/{id}`	The user-answers endpoint is used to do the following actions on a specific user's answer. UserAnswers is identified by user answers' identity number (id). • Get the specific user's answer - Http GET • Update the specific user's answer - Http PUT • Delete the specific user's answer - Http DELETE
7	`matches`	Matches endpoint is used to do the following actions. • Get all the matches - Http GET • Get specific user details by filtering with "user" attribute
8	`user-likes`	The user-likes endpoint is used to do the following actions. • Get all the list of liked user - Http GET • Get specific user details by filtering with "user" attribute • Create a new user-likes – Http POST
9	`user-likes/{user_name}`	User-likes/{id} endpoint is used to do the following actions on a specific user. UserAnswers is identified by username or user identity number (id). • Get the specific user's list of liked users - Http GET • Update the specific user details – Http PUT • Delete the specific user – Http DELETE
10	`users`	Users endpoint is used to do the following actions. • Get all the list of users - Http GET • Get specific user details by filtering with "user" attribute

		• Create a new user – Http POST
11	`users/{user_name}`	Users/{user_name} endpoint is used to do the following actions on a specific user. UserAnswers is identified by username or user identity number (id). • Get the specific user - Http GET • DELETE the specific user – Http DELETE

Table 7 All the REST APIs and its description details.

8.3 How to Use and Examples

Table 8 gives the detailed usage scenarios for Answers API, which is same for other APIs mentioned in Table 9 except for matches API. Matches API is responsible for the main matchmaking calculations, so it is allowed to support read-only HTTP GET requests for secure operations. The curl command is used for raising the Http request and its outcome is provided in a short form to get the crux of the output.

Sr. No	HTTP Request to /answers API	HTTP Response from /answers API
1	1) curl -d '{"text" : "Natural Science (Eg. Physics, Chemistry, Biology, etc)..."}' -X POST http://172.18.18.8/match-api/v0/answers 2) curl -d '{"text" : "Natural Science (Eg. Physics, Chemistry, Biology, etc)..."}' -X POST http://172.18.18.8/match-api/v0/answers	1) {"id":31,"text":"Natural Science (Eg. Physics, Chemistry, Biology, etc)..."} 2) {"text":["answer with this text already exists."]} Same answer cannot be created multiple times, so the error is generated for it.
2	curl http://172.18.18.8/match-api/v0/answers	[{"id":1,"text":"Mathematician"},{"id":2,"text":"Computer Programmer"}, ………. {"id":31,"text":"Natural Science (Eg. Physics, Chemistry, Biology, etc)..."}]
3	curl http://172.18.18.8/match-api/v0/answers/31	{"id":31,"text":"Natural Science (Eg. Physics, Chemistry, Biology, etc)..."}

4	1) curl http://172.18.18.8/match-api/v0/answers/31 2) curl -X PUT http://172.18.18.8/match-api/v0/answers/31 -d '{"text":"V V Good"}' 3) curl http://172.18.18.8/match-api/v0/answers/31	1) {"id":31,"text":"Natural Science (Eg. Physics, Chemistry, Biology, etc)..."} 2) {"id":31,"text":"V V Good"} 3) {"id":31,"text":"V V Good"}
5	1) curl http://172.18.18.8/match-api/v0/answers/31 2) curl -X DELETE http://172.18.18.8/match-api/v0/answers/31 3) curl http://172.18.18.8/match-api/v0/answers/31	1) {"id":31,"text":"V V Good"} 2) No RESPONSE. HTTP 204 status code returned for successful deletion. 3) {"error":{"code":404,"message":"Answer is not found"}} Answer object is already deleted, so resource not found error raised.

Table 8 Answers API with detailed usage scenarios with Http GET, POST and PUT requests

Sr. No	HTTP Request	HTTP Response
1	1) curl http://172.18.18.8/match-api/v0/questions 2) curl http://172.18.18.8/match-api/v0/questions/1	[{"id":1,"text":"What is your background?","answers":[{"id":1,"text":"Mathematician"},{"id":2,"text":"Computer Programmer"},{"id":3,"text":"Data Scientist"},{"id":4,"text":"Business & Sales"},{"id":5,"text":"Internet"}]}, ………. {"id":12,"text":"Which HiDALGO use case is interesting for UUU?","answers":[{"id":27,"text":"Social Networks (SN) Pilot"},{"id":28,"text":"Migraton Pilot"},{"id":29,"text":"Generic Usecase"},{"id":30,"text":"Yess"}]}]
2	1) curl http://172.18.18.8/match-api/v0/users 2) curl http://172.18.18.8/match-api/v0/users?	[{"id":1,"username":"test1"}, ………. {"id":5,"username":"test"}]

	user=dinesh1219 91	
3	1) curl http://172.18.18.8/match-api/v0/user-answers 2) curl http://172.18.18.8/match-api/v0/user-answers?user=dinesh1219 91	[{"id":1,"user":{"id":1,"username":"test1"},"question":{"id":1,"text":"What is your background?"},"my_answer":{"id":1,"text":"Mathematician"},"my_answer_importance":"Mandatory","their_answer":{"id":1,"text":"Mathematician"},"their_importance":"Mandatory"}, ………. user":{"id":4,"username":"test4"},"question":{"id":10,"text":"What is your HPC experience?"},"my_answer":{"id":24,"text":"HPC Visualization tools to export results"},"my_answer_importance":"Mandatory","their_answer":{"id":22,"text":"HPC programming to develop scientific models"},"their_importance":"Mandatory"}]
	1) curl http://172.18.18.8/match-api/v0/user-likes 2) curl http://172.18.18.8/match-api/v0/user-likes?user=dinesh1219 91	[{"user":{"id":1,"username":"test1"},"liked_users":[{"id":2,"username":"test2"},{"id":3,"username":"test3"}]}, ……… {"user":{"id":4,"username":"test4"},"liked_users":[{"id":2,"username":"test2"},{"id":3,"username":"test3"}]}]
	1) curl http://172.18.18.8/match-api/v0/matches 2) curl http://172.18.18.8/match-api/v0/matches?user=dinesh1219 91	[{"id":1,"user_a":{"id":1,"username":"test1"},"user_b":{"id":2,"username":"test2"},"match_decimal":"0.98192260","questions_answered":11}, ………. "user_a":{"id":4,"username":"test4"},"user_b":{"id":5,"username":"test"},"match_decimal":"0.00000000","questions_answered":0}]

Table 9 All the APIs supports GET, PUT and POST request except for matches API.

9 Frontend and Applications GUI

9.1 Implemented Solution

The Frontend of the HiDALGO project has been developed as an Angular Application. Angular is a frontend framework used for developing Single-Page Applications (SPA). SPAs work by rendering the websites in the client and rewriting them instead of loading them from the server, which avoids disruption between different pages.

The Frontend allows users to execute, stop and monitor their applications, using the Backend described in section 5. In the future more functionality will be available, and will connect with different HiDALGO services, such as the Moodle Training section, and will allow for Data Visualization and embedded Notebooks for manipulating and performing tests on the datasets generated by the HiDALGO applications.

9.2 Available APIs

The Frontend is a web application accessible via a web browser. In the next section, more details will be provided as to how to operate the website.

9.3 How to Use and Examples

The Portal Frontend is still under development and in prototype phase, so the styles and/or position of the different elements may, and probably will, differ in the future.

When the user lands in the portal, they will be prompted to log in, as shown in the Figure 9. This will authenticate the user in the whole HiDALGO ecosystem, as explained in the section 3. Once Authenticated, they be presented with the dashboard. The user will have the ability to submit an Application Cloudify Blueprint. This screen is shown in the Figure 25. The blueprint will be sent to the Backend, who will check its fields and, if everything is correct, will send it to the Cloudify server to deploy and execute in the HiDALGO infrastructure.

The user will have the ability to monitor running applications as well. The logs will be displayed under each Application in order to inspect possible problems and analyse jobs performance.

First of all, the user must install the blueprint, so the application will be available to be deployed through the Orchestrator. Once the blueprint is installed, it is possible to create as many instances as required, by providing the adequate input files. In case the application workflow is already installed, it is listed by the GUI.

In order to install an application, it is necessary to provide the name and its description. After that, the workflow must be provided with the 'Choose File' button (or doing drag and drop).

Once it is done, we must provide the inputs of the workflow (according to the definitions of the blueprint) and the application will be submitted to the corresponding HPC centre.

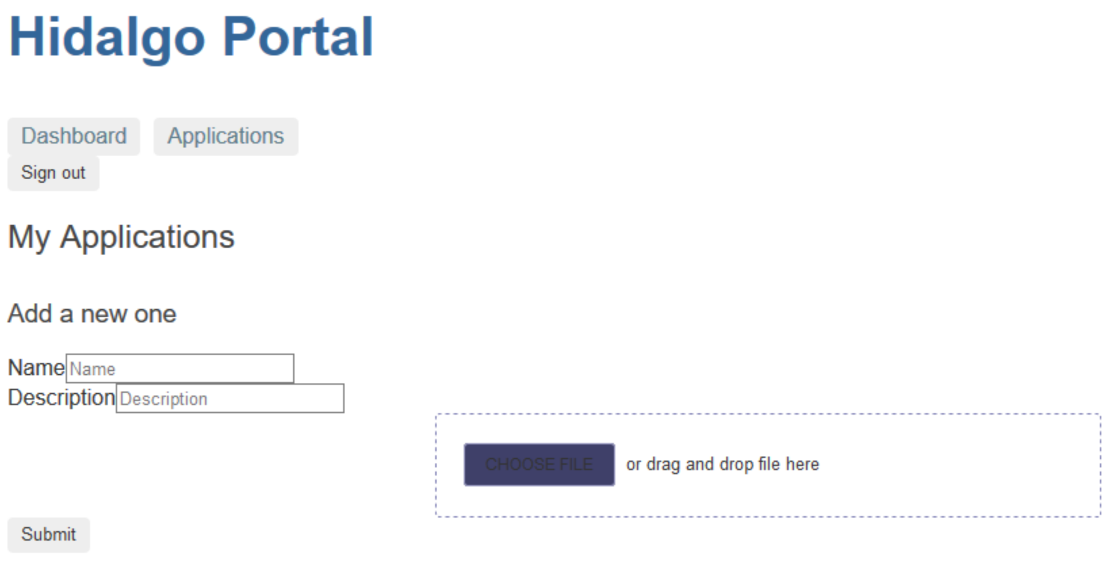


Figure 25 – Form to submit new Cloudify Blueprints

10 Conclusions

The consortium has progressed towards the implementation of the first release of the HiDALGO Portal, following the design already presented in D5.2[1]. We have focused the implementation on three main features: the execution of applications abstracting the complexity of HPC systems, the community support tools and the data management aspects. Additionally, the consortium has been working in setting up a solution for Continuous Integration and Deployment, as a way to support the development and maintenance of HiDALGO components.

The current implementation differs a bit with the original plan, since some features (like the documentation part) have been postponed in order to boost the community building part, also because of the feasibility to get enough progress.

In the end, the selected tools and applications for the Portal are open source and well-proved (CKAN, Moodle, Cloudify with Croupier, etc.), so the Portal will be robust enough and the consortium will have access to the code, in order to carry out any required adaptation.

References

[1] HiDALGO, D5.2 – Portal Architecture and Roadmap. Carnero, Javier. 2019.

[2] HiDALGO, D6.1 – Requirements Process and Results Definition. Maritsch, Martin. 2019.

[3] HiDALGO, D6.2 – Workflow and Services Definition. Maritsch, Martin. 2019.

[4] Jenkins. URL: https://jenkins.io/. Last visited in December 2019.

[5] Moodle Training Service. *URL: https://www.moodle.org*. Last visited in December 2019.

[6] CKAN: https://ckan.org/, last visited in December 2019

[7] Keycloak Server Administration Guide. URL: https://www.keycloak.org/docs/latest/server_admin/, last visited in December 2019

[8] About Cloudify. URL: https://docs.cloudify.co/5.0.0/about/, last visited December 2019

[9] OASIS. Security Assertion Markup Language (SAML) 2.0 Technical Overview. 22nd July 2004. http://xml.coverpages.org/SAML-TechOverviewV20-Draft7874.pdf. Last visited in December 2019.

[10] OpenID. OpenID Connect Core 1.0. 8th November 2014. https://openid.net/specs/openid-connect-core-1_0.html. Last visited December 2019.

[11] OASIS. TOSCA Simple Profile in YAML Version1.3. 18th September 2019. https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/cs01/TOSCA-Simple-Profile-YAML-v1.3-cs01.pdf. Last visited December 2019.

[12] Topology and Orchestration Specification for Cloud Applications Version 1.0. 25 November 2013. Oasis Standard. URL: http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html, last visited December 2019

[13] Croupier GitLab Repository. URL: https://github.com/ari-apc-lab/croupier

Annexes

Appendix 1: Jenkins Pipeline Definition

```
#!/usr/bin/groovy

pipeline {
    agent {label 'master'}

    options {
        disableConcurrentBuilds()
    }

    environment {
        PYTHONPATH = "${WORKSPACE}/"
    }

    stages {
        stage("Integration - Install") {
                steps { integration() }
        }
        stage("Integration - Test") {
                steps { runUAT("sophora-103.man.poznan.pl", "https") }
        }
        stage("Approve for Production") {
                steps { approve() }
        }
        stage("Deployment - Install") {
                steps { deploy() }
        }
        stage("Deployment - Test") {
                steps { runUAT("cloudify.hidalgo-project.eu", "https") }
        }
    }
}

def integration() {
        sh "ansible-playbook -i Jenkins/Inventory/cloudify_integration.INI
./cloudify-integration.yml --vault-password-
file=~/HiDALGO/VaultPassword/cloudify_vault.txt"
}

def deploy() {
        sh "ansible-playbook -i Jenkins/Inventory/cloudify.INI ./cloudify-
deployment.yml --vault-password-
file=~/HiDALGO/VaultPassword/cloudify_vault.txt"
}

def approve() {
        try {
                timeout(time:1, unit:'DAYS') {
                        input('Do you want to deploy to live?')
                }
        } catch(err) {
                def user = err.getCauses()[0].getUser()
                if('SYSTEM' == user.toString()) { // SYSTEM means timeout.
                        didTimeout = true
                } else {
                        userInput = false
                        echo "Aborted by: [${user}]"
                }
        }

}

def runUAT(hostname, protocol) {
        sh "Tests/ping_cloudify.sh ${hostname} ${protocol}"
}
```

Appendix 2: Cloudify Unit Test & Code Coverage

```
-------- coverage: platform linux2, python 2.7.15-candidate-1 ------
--
Name                                                           Stmts
Miss  Cover   Missing
--------------------------------------------------------------------
---------------------
croupier_plugin/__init__                                           0
0   100%
croupier_plugin/external_repositories/__init__                     0
0   100%
croupier_plugin/external_repositories/ckan                        15
15     0%   28-56
croupier_plugin/external_repositories/external_repository         20
20     0%   28-76
croupier_plugin/job_requester                                     49
49     0%   28-126
croupier_plugin/ssh                                              173
142    18%   43-44, 62-125, 129, 133, 137-140, 155-173, 184-267,
272-275, 282-294, 297, 300, 314-361, 366, 371-373
croupier_plugin/tasks                                            299
299     0%   28-653
croupier_plugin/tests/__init__                                     0
0   100%
croupier_plugin/tests/slurm_tests                                 70
1    99%   258
croupier_plugin/tests/spark_tests                                108
1    99%   224
croupier_plugin/tests/torque_tests                                89
9    90%   176-185, 193-200, 215, 269
croupier_plugin/tests/workflow_tests                              91
67    26%    46, 55-56, 74-87, 105-118, 136-149, 167-180, 198-211,
230-243, 275-288, 320-333, 416-429, 433
croupier_plugin/utilities                                          4
0   100%
croupier_plugin/workflows                                        254
254     0%   27-452
```

```
croupier_plugin/workload_managers/__init__                           0
0   100%
croupier_plugin/workload_managers/bash                              54
54      0%   28-119
croupier_plugin/workload_managers/slurm                            118
43    64%   39-79, 118-135, 153, 225-242, 252
croupier_plugin/workload_managers/spark                            151
43    72%   78-104, 121, 145, 182, 190-227, 272, 279, 281
croupier_plugin/workload_managers/torque                           170
97    43%   38-99, 137, 150, 153, 157, 162, 166, 214, 237-271, 277-
280, 284-298, 302-342
croupier_plugin/workload_managers/workload_manager                 145
82    43%   118, 138, 152-153, 158, 188-281, 304-311, 334-348, 353-
367, 384, 404, 422, 437, 449-468, 476, 485-492
--------------------------------------------------------------------
---------------------
TOTAL                                                             1810
1176    35%
--------------------------------------------------------------------
--
Ran 49 tests in 10.752s
```

Appendix 3: Blueprint Example

This Blueprint describes four jobs that should run in an HPC node, either directly in hardware or using Singularity. It also shows that there may be dependencies between jobs, so the Orchestrator will understand how to carry out the execution.

```
first_job:
    type: croupier.nodes.Job
    properties:
        job_options:
            partition: { get_input: partition_name }
            commands: ["touch fourth_example_1.test"]
            nodes: 1
            tasks: 1
            tasks_per_node: 1
            max_time: "00:01:00"
        skip_cleanup: True
    relationships:
        - type: job_managed_by_interface
          target: hpc_interface

second_parallel_job:
    type: croupier.nodes.Job
    properties:
        job_options:
            partition: { get_input: partition_name }
            commands: ["touch fourth_example_2.test"]
```

```
                nodes: 1
                tasks: 1
                tasks_per_node: 1
                max_time: "00:01:00"
            skip_cleanup: True
        relationships:
            - type: job_managed_by_interface
              target: hpc_interface
            - type: job_depends_on
              target: first_job

    third_parallel_job:
        type: croupier.nodes.Job
        properties:
            job_options:
                script: "touch.script"
                arguments:
                    - "fourth_example_3.test"
                nodes: 1
                tasks: 1
                tasks_per_node: 1
                max_time: "00:01:00"
                partition: { get_input: partition_name }
            deployment:
                bootstrap: "scripts/create_script.sh"
                revert: "scripts/delete_script.sh"
                inputs:
                    - "script_"
            skip_cleanup: True
        relationships:
            - type: job_managed_by_interface
              target: hpc_interface
            - type: job_depends_on
              target: first_job

    fourth_job:
        type: croupier.nodes.Job
        properties:
            job_options:
                script: "touch.script"
                arguments:
                    - "fourth_example_4.test"
                nodes: 1
                tasks: 1
                tasks_per_node: 1
                max_time: "00:01:00"
                partition: { get_input: partition_name }
            deployment:
                bootstrap: "scripts/create_script.sh"
                revert: "scripts/delete_script.sh"
                inputs:
                    - "script_"
            skip_cleanup: True
```

```
relationships:
    - type: job_managed_by_interface
      target: hpc_interface
    - type: job_depends_on
      target: second_parallel_job
    - type: job_depends_on
      target: third_parallel_job
```