DITAS

Data-intensive applications Improvement by moving daTA and computation in mixed cloud/fog environmentS

D5.3 Integration of DITAS and Cloud Testbed Final report

Project Acronym	**DITAS**
Project Title	Data-intensive applications Improvement by moving daTA and computation in mixed cloud/fog environmentS
Project Number	731945
Instrument	Collaborative Project
Start Date	01/01/2017
Duration	36 months
Thematic Priority	ICT-06-2016 Cloud Computing
Website:	http://www.ditas-project.eu

Dissemination level: Public

Work Package	**WP5 Real world case studies and integration**
Due Date:	M32
Submission Date:	17/09/2019
Version:	1.0
Status	Final for submission
Author(s):	Grigor Pavlov, Peter Gray (CS), Alexandros Psychas, Achilleas Marinakis, George Chatzikyriakos (ICCS), David García Pérez, Enric Pages, Jose Antonio Sanchez (ATOS), Frank Pallas, Sebastian Werner, Richard Girke (TUB), Maya Anderson (IBM), Mattia Salnitri, Giovanni Meroni (POLIMI)
Reviewer(s)	Pierluigi Plebani (POLIMI), Jose Antonio Sanchez (ATOS)

This project has received funding by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 731945

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
0.1	08/07/2019	Table of contents	Grigor Pavlov (CS)
0.2	11/07/2019	DITAS Integration section added	Borja Tornos (IDEKO)
0.3	24/07/2019	Testbed section integrated	Grigor Pavlov (CS), Peter Gray (CS)
0.4	28/08/2019	Document ready for internal review	Grigor Pavlov (CS)
0.5	04/09/2019	Review remarks addressed	Grigor Pavlov (CS), Pierluigi Plebani (POLIMI, Jose Antonio Sanchez (ATOS)
0.6	05/09/2019	Quality Assessment	Maria Teresa Garcia (ATOS)
0.7	16/09/2019	Second internal review remarks addressed	Pierluigi Plebani (POLIMI, Jose Antonio Sanchez (ATOS)
1.0	17/09/2019	Final quality Assessment. Document ready for submission	Grigor Pavlov (CS), Maria Teresa Garcia (ATOS)

Contents

List of Figures

List of tables

Executive Summary

The DITAS project aims to develop an innovative framework to simplify the development and execution of data intensive applications. To facilitate this, WP5 is responsible for the provisioning of a cloud testbed for integration of the DITAS platform components and the validation of the framework. At month 11 we submitted D5.1, which defined the integration strategy and cloud testbed for building the DITAS platform in support of the two use cases - IDEKO and OSR use cases. This was followed by D5.2, submitted in month 18, which included an integration and validation report for the first iteration of the DITAS framework. Again, the two use case supplied by IDEKO and OSR were used for validation purposes and the outcomes were fed back into Task 5.1 and Task 5.2 for input into the second iteration.

This deliverable outlines the provisioning of the development and production cloud testbed required for the integration of all the DITAS components after their second iteration. The goal was to provide the two use-cases (provided by IDEKO and OSR) with the necessary cloud resources and test environment to effectively demonstrate and validate the DITAS framework.

The DITAS integration involves the creation and use of a continuous integration system and the deployment and integration of the DITAS components on a cloud testbed. The continuous integration (CI) system consists of six VMs, including one Jenkins Master Server and two slave machines, a staging machine and two production machines where the components are finally deployed for the IDEKO and OSR use cases, respectively. To facilitate this, the underlying commercial cloud infrastructure is provided by CloudSigma, making two locations available (Frankfurt and Miami). The staging server and production servers are deployed in CloudSigma datacenters and computing resources are allocated per use case. The use cases have documented the deployment process here so that potential users can gain a better understanding of the typical configurations and the cloud resources needed. This report verifies that the main objectives of the WP have been met. It describes the deployment and integration of the components developed in previous WPs and precisely how the two use-cases are served by the solution for testing and evaluation.

1 Introduction

This deliverable presents the activities undertaken by Work Package 5 relating to the second iteration of the DITAS integration and the provisioning of the cloud testing environment. It reports on the work done in Tasks 5.1 and Task 5.2. It follows on from D5.2, which reported the results of the first iteration and provided the necessary feedback for the second iteration. Deliverable 5.4 will follow in month 36 to present the final case studies validation report.

In section 2 of this deliverable we describe the DITAS integration, the continuous integration system architecture, and the Jenkins Pipeline, while Section 3 outlines the deployment and integration of the DITAS components developed in other Work Packages. In Section 4, we include a description of the cloud infrastructure used and its configuration according to the resource requirement of the two use cases. Section 5 describes the setup for both use cases and provides a summary of the cloud resource usage. Finally, we provide a conclusion to the work carried out.

1.1 Glossary of Acronyms

Acronym	**Definition**
API	Application Program Interface
CI	Continuous Integration
CPU	Central Processing Unit
DAL	Data Access Layer
EE	Execution Engine
FRA	Frankfurt
HDD	Hard Disk Drive (Magnetic)
HTTP	Hypertext Transfer Protocol
MIA	Miami
RAM	Random Access Memory
SDK	Software Development Kit
SSD	Solid State Drive
VDC	Virtual Data Container
VDM	Virtual Data Manager
UC	Use Case
ZRH	Zurich

Table 1. Acronyms

2 DITAS integration

The following section explains the integration within the DITAS project divided in two primary sections, the continuous integration system section which explains the last changes performed to the DITAS CI system, and the components deployment and integration section where the development and integration of the components is described.

2.1 Continuous integration system

The continuous integration system used for the DITAS project has been already explained deeply on D5.1 (see [1]) and D5.2 (see [2]). As stated on those documents, in the case of the DITAS project, we set up a CI system which is triggered when a commit of any repository is done. After passing different tests and building the components, they were originally deployed on a single production machine.

During these months of project, some updates on the architecture, artifact naming and deployment scripts were introduced on the CI system. All these changes can be found in the following sections, which also gives a good insight of how the CI system currently works in DITAS.

2.1.1 Architecture

Initially and as it was stated on D5.2 (see [D5.2]), the DITAS Continuous integration (CI) system architecture consisted in five different machines deployed over the CloudSigma's cloud infrastructure. Due to focusing the demo on the Business Scenario 2: "Data As A Service" (see [3]), we decided to split the original production machine into two different machines, as we need two different entities to fit this scope; one for the IDEKO use case and another for the OSR use case. The following image shows the total of six different machines used for the final DITAS continuous integration system architecture.

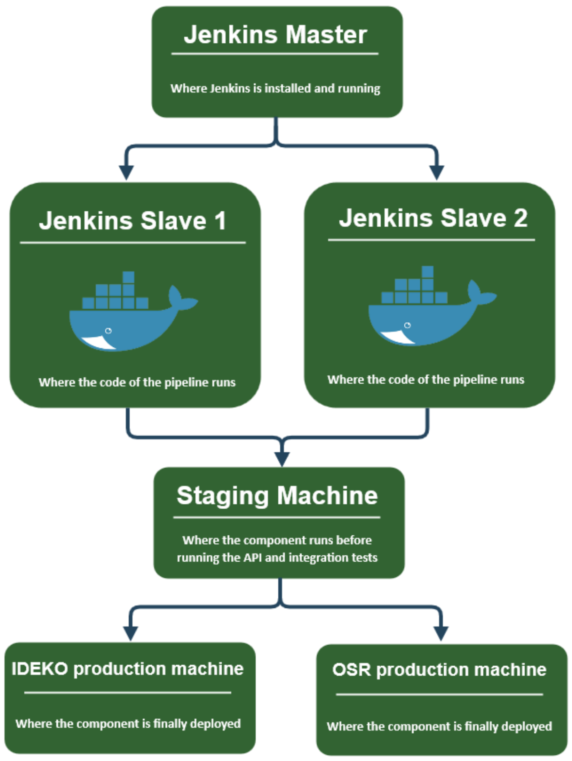

Jenkins Master
Where Jenkins is installed and running
Jenkins Slave 1
Where the code of the pipeline runs
Jenkins Slave 2
Where the code of the pipeline runs
Staging Machine
Where the component runs before running the API and integration tests
IDEKO production machine
Where the component is finally deployed
OSR production machine
Where the component is finally deployed

Figure 1: Final continuous integration system architecture

2.1.2 Artifacts

The artifacts for the DITAS components are the Docker images that are hosted on the public DITAS Docker Hub repository to maintain the open source philosophy of the project. Initially, we were only using the :latest Docker tag, but after realizing that each component has two different status, one before passing the API validation tests (staging), and another one after passing all the tests (production), each container pushed to the DITAS Docker Hub has two different tags, staging and production, a tagging convention acquired for the project:

- staging: Indicates that the container has passed the building and unit testing phases, but still needs to be deployed in the staging server to pass the API validation test.
- production: Indicates that the container has passed all the tests and it's ready to be deployed in production.

It's important to point out that the Deployment Engine, the components which is in charge of deploying the Execution Environment (EE) components, will always deploy the containers with the production tag.

2.1.3 Jenkins pipeline

The Jenkins pipeline used by the DITAS components varies depending on the component type; SDK or the Execution Environment (runtime component).
On the one hand the SDK components pipeline consists of six stages, which are already explained in D5.2 (see [2]). This pipeline is in charge of creating the building and testing environment, validating the API and deploying the component in production.

```
pipeline {
   agent none
   stages {
          stage('Build - test') {
             [...]
          }
          stage('Image creation') {
             [...]
          }
          stage('Image deploy - Staging') {
             [...]
          }
          stage('API validation') {
             [...]
          }
          stage('Integration tests') {
             [...]
          }
          stage('Image deploy - Production') {
             [...]
          }
   }
}
```

The last stage, "*Image deploy - Production*" consist of launching a shell script that runs the components in both use cases server, IDEKO UC server and OSR UC server, as it was explained in the previous *Architecture* section. The following is the shell script used by the vdc-blueprint-repository-engine to deploy the component in both servers.

```
# SSH to the IDEKO server and deploy SDK component there
ssh -i /opt/keypairs/ideko-sdk-key.pem cloudsigma@$IDEKO_UC_IP << 'ENDSSH'
sudo docker stop --time 20 vdc-blueprint-repository-engine || true
sudo docker rm --force vdc-blueprint-repository-engine || true
sudo docker pull ditas/vdc-blueprint-repository-engine:production
HOST_IP="$(ip route get 8.8.8.8 | awk '{print $NF; exit}')"
sudo docker run -p 50009:8080 -e DOCKER_HOST_IP=$HOST_IP --restart unless-stopped -d -
-name vdc-blueprint-repository-engine ditas/vdc-blueprint-repository-engine:production
ENDSSH

# SSH to the OSR server and deploy SDK component there
ssh -i /opt/keypairs/osr-sdk-key.pem cloudsigma@$OSR_UC_IP << 'ENDSSH'
sudo docker stop --time 20 vdc-blueprint-repository-engine || true
sudo docker rm --force vdc-blueprint-repository-engine || true
sudo docker pull ditas/vdc-blueprint-repository-engine:production
HOST_IP="$(ip route get 8.8.8.8 | awk '{print $NF; exit}')"
sudo docker run -p 50009:8080 -e DOCKER_HOST_IP=$HOST_IP --restart unless-stopped -d -
-name vdc-blueprint-repository-engine ditas/vdc-blueprint-repository-engine:production
ENDSSH
```

On the other hand, the EE components pipeline must not deploy the component in production, as the Deployment Engine will be in charge of doing the job.

```
pipeline {
   agent none
   stages {
            stage('Build - test') {
               [...]
            }
            stage('Image creation') {
               [...]
            }
            stage('Image deploy - Staging') {
               [...]
            }
            stage('API validation') {
               [...]
            }
            stage('Integration tests') {
               [...]
            }
            stage('Production image creation and push') {
               [...]
            }
   }
}
```

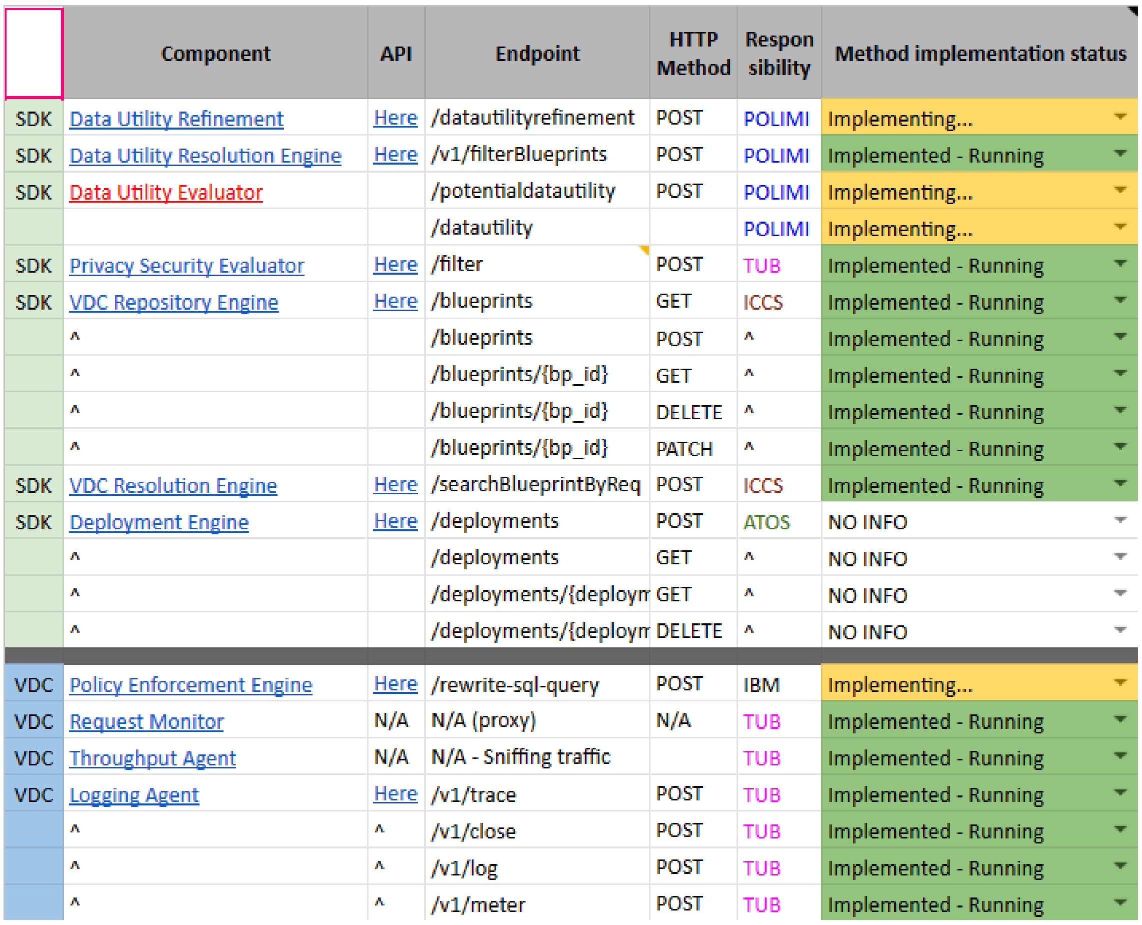
The job of the last stage "Production image creation and push", is to push the image with the tag: production to the DITAS Docker Hub.

3 Components deployment and integration

In order to track the deployment and integration of the DITAS components, a document was created where every component owner where every component owner update it manually. This document gives a fast insight of the status of the development of the components and it is widely used between the developers. It has three different sections that are explained below.

3.1 Components API implementation status

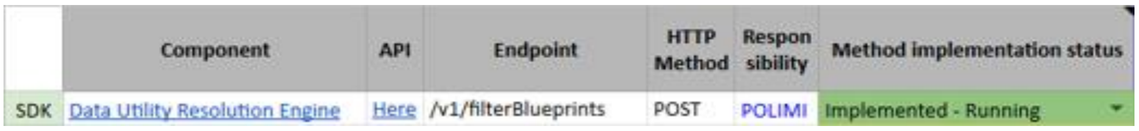
This section describes the status of the implementation of each method of every DITAS component, divided by component type (SDK/VDC/VDM). Figure 2 shows an extract of the table of the Components API implementation status section.



	Component	API	Endpoint	HTTP Method	Respon sibility	Method implementation status
SDK	Data Utility Refinement	Here	/datautilityrefinement	POST	POLIMI	Implementing...
SDK	Data Utility Resolution Engine	Here	/v1/filterBlueprints	POST	POLIMI	Implemented - Running
SDK	Data Utility Evaluator		/potentialdatautility	POST	POLIMI	Implementing...
			/datautility		POLIMI	Implementing...
SDK	Privacy Security Evaluator	Here	/filter	POST	TUB	Implemented - Running
SDK	VDC Repository Engine	Here	/blueprints	GET	ICCS	Implemented - Running
	^		/blueprints	POST	^	Implemented - Running
	^		/blueprints/{bp_id}	GET	^	Implemented - Running
	^		/blueprints/{bp_id}	DELETE	^	Implemented - Running
	^		/blueprints/{bp_id}	PATCH	^	Implemented - Running
SDK	VDC Resolution Engine	Here	/searchBlueprintByReq	POST	ICCS	Implemented - Running
SDK	Deployment Engine	Here	/deployments	POST	ATOS	NO INFO
	^		/deployments	GET	^	NO INFO
	^		/deployments/{deploym	GET	^	NO INFO
	^		/deployments/{deploym	DELETE	^	NO INFO
VDC	Policy Enforcement Engine	Here	/rewrite-sql-query	POST	IBM	Implementing...
VDC	Request Monitor	N/A	N/A (proxy)	N/A	TUB	Implemented - Running
VDC	Throughput Agent	N/A	N/A - Sniffing traffic		TUB	Implemented - Running
VDC	Logging Agent	Here	/v1/trace	POST	TUB	Implemented - Running
	^	^	/v1/close	POST	TUB	Implemented - Running
	^	^	/v1/log	POST	TUB	Implemented - Running
	^	^	/v1/meter	POST	TUB	Implemented - Running

Figure 2: Extract of the Component API implementation status section

In order to ease the understanding of the table let's focus on a single row. Figure 3 describes that the SDK component Data Utility Resolution Engine offers the endpoint /v1/filterBlueprints, which is implemented and running, at least, on the staging machine. In this way, a developer of another component who needs to call the /v1/filterBlueprints method of the Data Utility Resolution Engine knows that the method is full operative and working.
The row also contains a link to the API definition file (swagger file) which is hosted on the GitHub repository, the HTTP method, and the partners who is responsible of the development of the component to ease the interaction of work between all the project members.

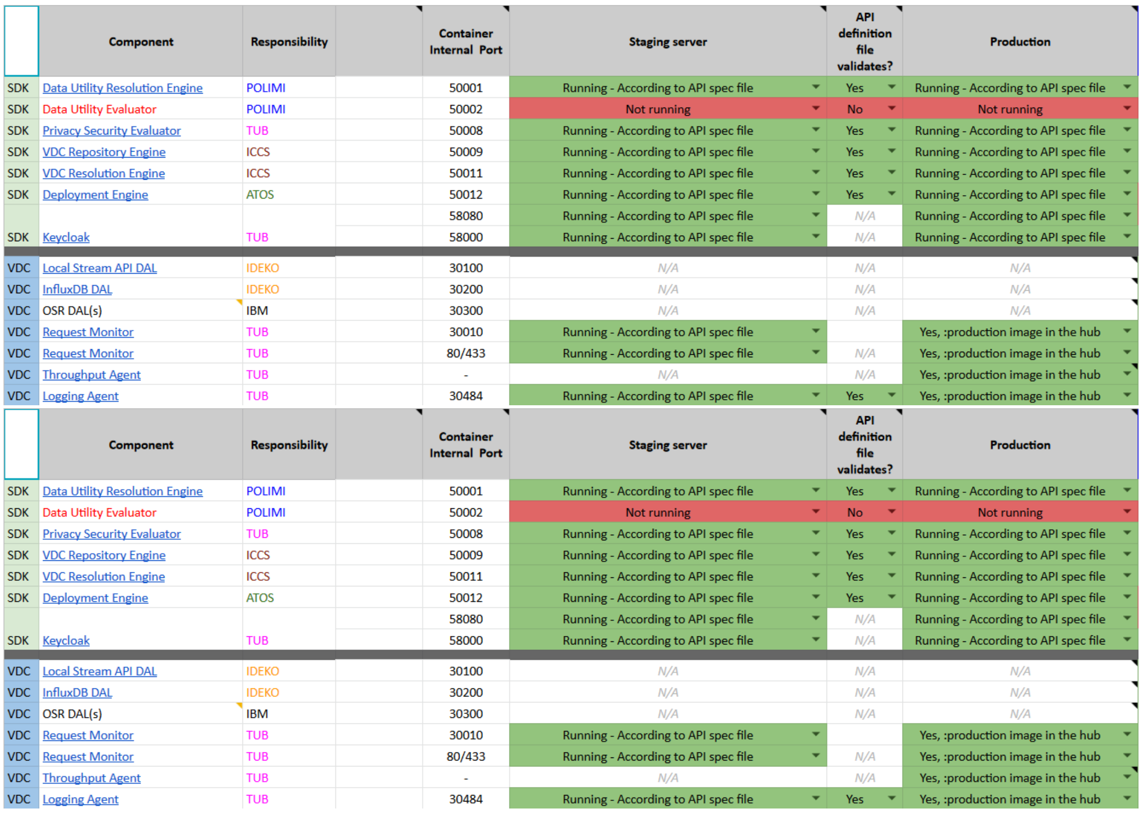


	Component	API	Endpoint	HTTP Method	Responsibility	Method implementation status
SDK	Data Utility Resolution Engine	Here	/v1/filterBlueprints	POST	POLIMI	Implemented - Running

Figure 3: Data Utility Resolution Engine method implementation status

3.2 Deployment status

The section Deployment status shows information of the deployment of every component divided also by component type.
Figure 4 shows an extract of the table of the Deployment status section at the time that these lines where written:

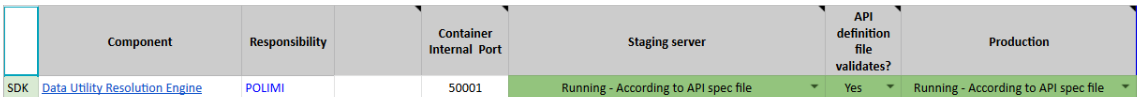


	Component	Responsibility		Container Internal Port	Staging server	API definition file validates?	Production
SDK	Data Utility Resolution Engine	POLIMI		50001	Running - According to API spec file	Yes	Running - According to API spec file
SDK	Data Utility Evaluator	POLIMI		50002	Not running	No	Not running
SDK	Privacy Security Evaluator	TUB		50008	Running - According to API spec file	Yes	Running - According to API spec file
SDK	VDC Repository Engine	ICCS		50009	Running - According to API spec file	Yes	Running - According to API spec file
SDK	VDC Resolution Engine	ICCS		50011	Running - According to API spec file	Yes	Running - According to API spec file
SDK	Deployment Engine	ATOS		50012	Running - According to API spec file	Yes	Running - According to API spec file
				58080	Running - According to API spec file	N/A	Running - According to API spec file
SDK	Keycloak	TUB		58000	Running - According to API spec file	N/A	Running - According to API spec file
VDC	Local Stream API DAL	IDEKO		30100	N/A	N/A	N/A
VDC	InfluxDB DAL	IDEKO		30200	N/A	N/A	N/A
VDC	OSR DAL(s)	IBM		30300	N/A	N/A	N/A
VDC	Request Monitor	TUB		30010	Running - According to API spec file		Yes, :production image in the hub
VDC	Request Monitor	TUB		80/433	Running - According to API spec file	N/A	Yes, :production image in the hub
VDC	Throughput Agent	TUB		-	N/A	N/A	Yes, :production image in the hub
VDC	Logging Agent	TUB		30484	Running - According to API spec file	Yes	Yes, :production image in the hub

	Component	Responsibility		Container Internal Port	Staging server	API definition file validates?	Production
SDK	Data Utility Resolution Engine	POLIMI		50001	Running - According to API spec file	Yes	Running - According to API spec file
SDK	Data Utility Evaluator	POLIMI		50002	Not running	No	Not running
SDK	Privacy Security Evaluator	TUB		50008	Running - According to API spec file	Yes	Running - According to API spec file
SDK	VDC Repository Engine	ICCS		50009	Running - According to API spec file	Yes	Running - According to API spec file
SDK	VDC Resolution Engine	ICCS		50011	Running - According to API spec file	Yes	Running - According to API spec file
SDK	Deployment Engine	ATOS		50012	Running - According to API spec file	Yes	Running - According to API spec file
				58080	Running - According to API spec file	N/A	Running - According to API spec file
SDK	Keycloak	TUB		58000	Running - According to API spec file	N/A	Running - According to API spec file
VDC	Local Stream API DAL	IDEKO		30100	N/A	N/A	N/A
VDC	InfluxDB DAL	IDEKO		30200	N/A	N/A	N/A
VDC	OSR DAL(s)	IBM		30300	N/A	N/A	N/A
VDC	Request Monitor	TUB		30010	Running - According to API spec file		Yes, :production image in the hub
VDC	Request Monitor	TUB		80/433	Running - According to API spec file	N/A	Yes, :production image in the hub
VDC	Throughput Agent	TUB		-	N/A	N/A	Yes, :production image in the hub
VDC	Logging Agent	TUB		30484	Running - According to API spec file	Yes	Yes, :production image in the hub

Figure 4: Extract of the Deployment status section

Let's focus again on a single row. Figure 5 describes that the SDK component Data Utility Resolution Engine is running on the staging server on port 50001, that it validates the API definition file (this means that passes the API validation test) and it's running on both production machines on port 50001.

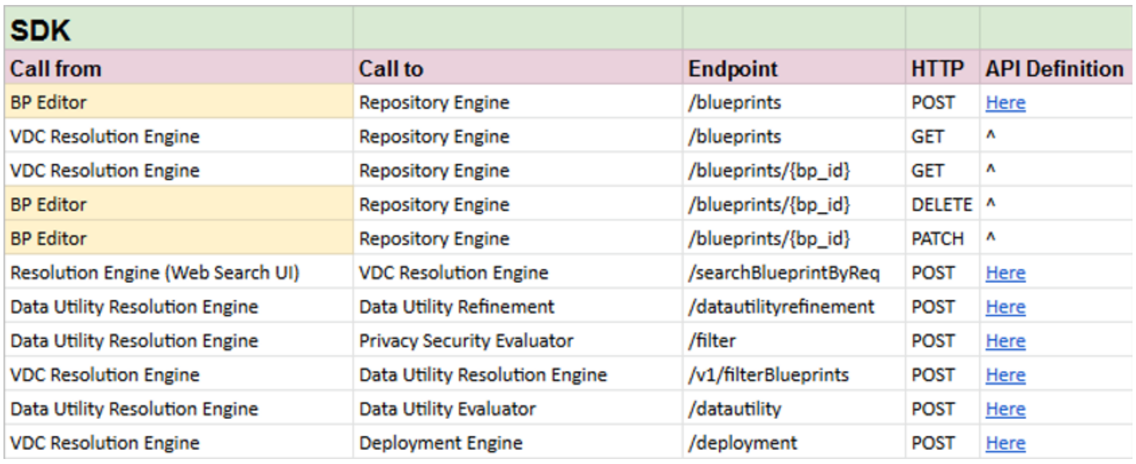


	Component	Responsibility		Container Internal Port	Staging server	API definition file validates?	Production
SDK	Data Utility Resolution Engine	POLIMI		50001	Running - According to API spec file	Yes	Running - According to API spec file

Figure 5: Data Utility Resolution Engine deployment status

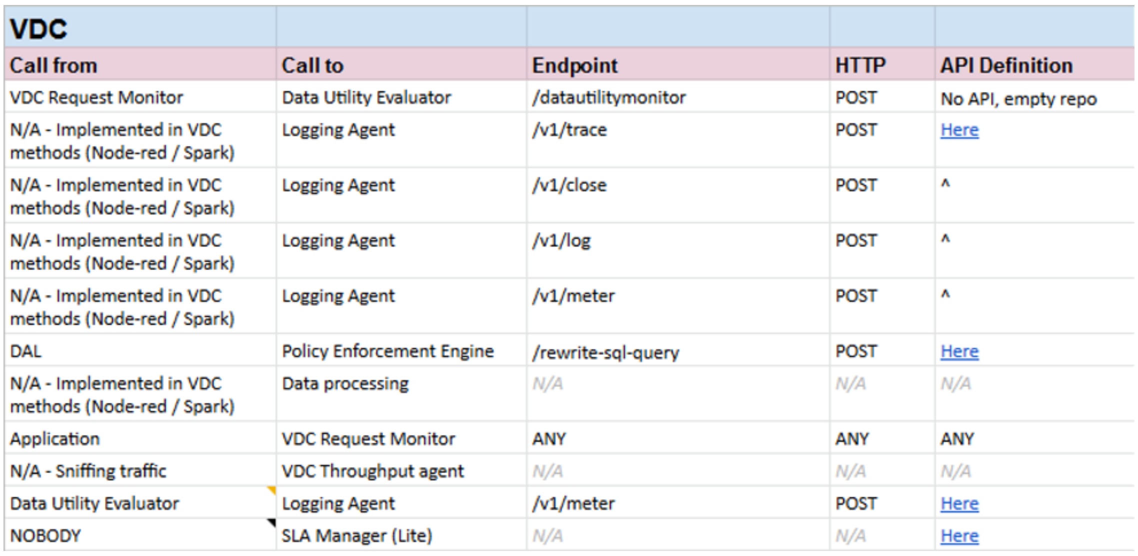
3.2.1 SDK, VDC and VDM call maps

This section describes the relationship among the DITAS components by stating the origin and destination for all the calls between them. This is a critical document as it is used to validate the integration among components. Figure 6 shows the call map for the SDK, Figure 7 for the VDC and Figure 8 for the VDM.



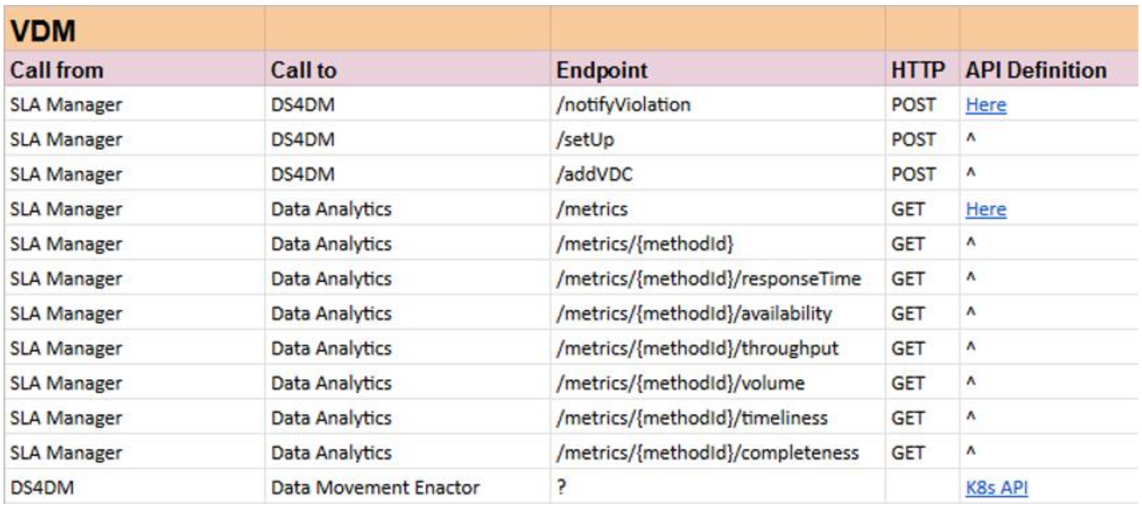
SDK				
Call from	**Call to**	**Endpoint**	**HTTP**	**API Definition**
BP Editor	Repository Engine	/blueprints	POST	Here
VDC Resolution Engine	Repository Engine	/blueprints	GET	^
VDC Resolution Engine	Repository Engine	/blueprints/{bp_id}	GET	^
BP Editor	Repository Engine	/blueprints/{bp_id}	DELETE	^
BP Editor	Repository Engine	/blueprints/{bp_id}	PATCH	^
Resolution Engine (Web Search UI)	VDC Resolution Engine	/searchBlueprintByReq	POST	Here
Data Utility Resolution Engine	Data Utility Refinement	/datautilityrefinement	POST	Here
Data Utility Resolution Engine	Privacy Security Evaluator	/filter	POST	Here
VDC Resolution Engine	Data Utility Resolution Engine	/v1/filterBlueprints	POST	Here
Data Utility Resolution Engine	Data Utility Evaluator	/datautility	POST	Here
VDC Resolution Engine	Deployment Engine	/deployment	POST	Here

Figure 6: SDK call map



VDC				
Call from	**Call to**	**Endpoint**	**HTTP**	**API Definition**
VDC Request Monitor	Data Utility Evaluator	/datautilitymonitor	POST	No API, empty repo
N/A - Implemented in VDC methods (Node-red / Spark)	Logging Agent	/v1/trace	POST	Here
N/A - Implemented in VDC methods (Node-red / Spark)	Logging Agent	/v1/close	POST	^
N/A - Implemented in VDC methods (Node-red / Spark)	Logging Agent	/v1/log	POST	^
N/A - Implemented in VDC methods (Node-red / Spark)	Logging Agent	/v1/meter	POST	^
DAL	Policy Enforcement Engine	/rewrite-sql-query	POST	Here
N/A - Implemented in VDC methods (Node-red / Spark)	Data processing	N/A	N/A	N/A
Application	VDC Request Monitor	ANY	ANY	ANY
N/A - Sniffing traffic	VDC Throughput agent	N/A	N/A	N/A
Data Utility Evaluator	Logging Agent	/v1/meter	POST	Here
NOBODY	SLA Manager (Lite)	N/A	N/A	Here

Figure 7: VDC call map



VDM				
Call from	**Call to**	**Endpoint**	**HTTP**	**API Definition**
SLA Manager	DS4DM	/notifyViolation	POST	Here
SLA Manager	DS4DM	/setUp	POST	^
SLA Manager	DS4DM	/addVDC	POST	^
SLA Manager	Data Analytics	/metrics	GET	Here
SLA Manager	Data Analytics	/metrics/{methodId}	GET	^
SLA Manager	Data Analytics	/metrics/{methodId}/responseTime	GET	^
SLA Manager	Data Analytics	/metrics/{methodId}/availability	GET	^
SLA Manager	Data Analytics	/metrics/{methodId}/throughput	GET	^
SLA Manager	Data Analytics	/metrics/{methodId}/volume	GET	^
SLA Manager	Data Analytics	/metrics/{methodId}/timeliness	GET	^
SLA Manager	Data Analytics	/metrics/{methodId}/completeness	GET	^
DS4DM	Data Movement Enactor	?		K8s API

Figure 8: VDM call map

4 Testbed

4.1 Description

The DITAS testbed is built on top of CloudSigma's commercial cloud infrastructure. CloudSigma offers one of the most customizable cloud solutions on the market with a focus on open design and flexibility. The platform is designed to provide an environment with the same level of freedom as a private on-premise environment. All functionality is available via the API and the WebApp.
As in real-world scenario, project partners are able to provision processing, storage, networking and other fundamental computing resources in an unbundled way. In other words, CPU, RAM, storage and bandwidth can be purchased without the limitation of fixed sizes. Essentially, this means any combination of CPU and RAM can be realized along with multiple mounted drives and network interfaces. The same level of flexibility is provided to the project partners, within the resource limit mentioned in table 2. In a commercial situation, each resource is billed separately and transparently as either a subscription or as "burst" in 5 minute billing intervals. This enables customers to track precisely the amount their cloud servers are costing in near real-time.

Resource type	CPU	RAM	SSD	HDD
Max. allocated	100GHz	100GB	1500GB	3000GB
Actual usage*	168	190	3135	700

Table 2. Actual usage is presented here as indicative usage. The actual usage has varied over time.

As shown in Figure 9, one DITAS account was setup for the Staging server, and some more for each Use Case (UC). Each UC account has an SDK Production server and the possibility to deploy the resources available for the runtime components in different regions (e.g., FRA, MIA). Other instances for the specific UC are also available, on an "as needed" basis.

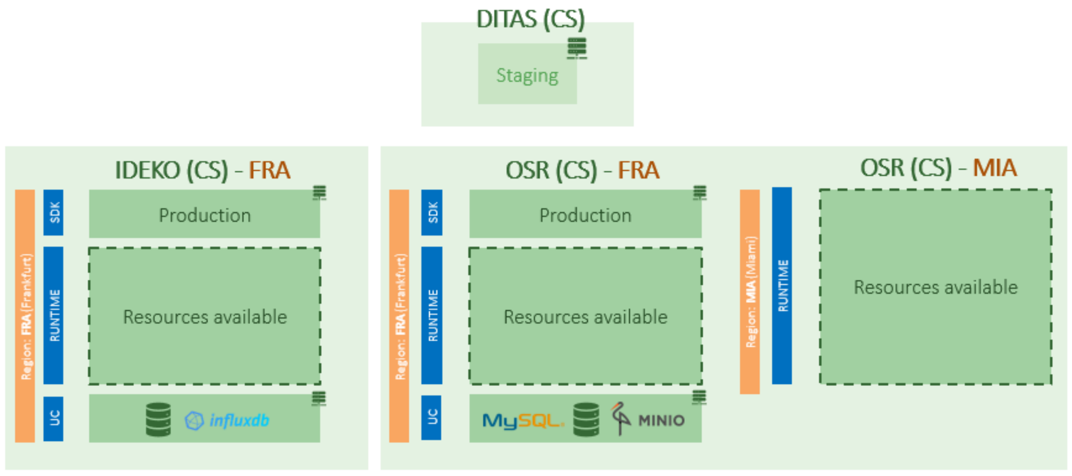


Figure 9: Testbed environment

4.1.1 Advantages of CloudSigma

Any x86 based operating system and software can be used with complete admin/root control. This includes all variants of BSD, Linux and Windows. Users can also upload their own raw ISO images, attach CPU and RAM to it and boot it. This enables full backward compatibility.

As previously mentioned, the cloud servers and drives are persistent and modelled on the same methodology as physical dedicated server equivalents. VLANs and IP addresses are also controlled using standard behavior and support all types of traffic including multicast and broadcast traffic, which is critical for highly available infrastructure in failover.

CloudSigma is also able to provide a number of power tools allowing users to achieve much higher levels of performance. These features include the ability to define a virtual core size, to expose NUMA topology and to tweak the hypervisor timer setting for maximum performance. CloudSigma also exposes the full CPU instruction set to the VM, allowing for faster processing in some situations.

At account-level, administrators are able to assign specific access and control rights over certain account related operations using access control lists (ACLs). For example, the account administrator can use the “labeling” feature to grant access to one or more users.

VMs can be provisioned in a matter of seconds with a high degree of flexibility and control. The average provisioning time for a new cloud server/VM on CloudSigma is just 15 seconds. This is critical for services that require flexible provisioning of resources in accordance with fluctuating user demand.

Users have the option of using either API or WebApp. API access features all account actions available (100% API coverage), allowing complete automation and remote infrastructure monitoring. The WebApp offers users a feature-rich and intuitive browser-based GUI. VM provisioning can be achieved with a “wizard” or custom server creation tool. Users can choose from a wide selection of ready system images including a number of operating systems or upload their own ready ISO image.

4.1.2 Resource allocation and account creation

CloudSigma provides to the project a fully featured production cloud as described above. A number of CloudSigma accounts have been provided for the various ongoing tasks within the project. Three CloudSigma locations have been used, namely Zurich (ZRH), Frankfurt (FRA), and Miami (MIA). Resource usage has been monitored by CloudSigma to ensure optimal resource allocation and to track spending against the planned allocated resources defined in the DoW [4] (CPU: 100 GHz, RAM: 100GB, SSD: 15000GB, HDD: 3000GB). As shown in the following table, the actual usage has exceeded the allocated resources in most cases. However, CloudSigma has allowed some flexibility (over allocation) to ensure the resource requirements are met and the project can run without impediment.

CloudSigma cloud infrastructure summary	
Testbed infrastructure description	Commercial IaaS platform in Zurich (ZRH), Frankfurt (FRA) and Miami (MIA). The CloudSigma platform combines a proprietary cloud stack with open-source technologies to provide a utility approach to IaaS provisioning. The platform offers a high level of control and flexibility to customers wishing to provision compute, storage and networking.
General testbed configuration	
Hypervisor	KVM
IaaS stack / version	Proprietary CloudSigma cloud stack
VM provisioning	Intra-VM testing tools at owners discretion
Access methods	API via https
Connectivity	Internet, VPN, Secure Remote User Access, Direct private patch to local switch
Cloud Interface	
Provisioning	API, API middleware, WebApp, Python library (Pycloudsigma). API documentation found here. https://cloudsigma-docs.readthedocs.io/en/latest/
Cloud integration / drivers	OpenStack HEAT, CloudInit, Apache Libcloud, JClouds, Fog, Ansible, Abiquo Hybrid Cloud, pycloudsigma Library
Networking	API, WebApp
Compute capacity	
CPU (GHz/core)	2.3GHz
CPU (Total available to the project)	100GHz
RAM (GB / VM)	128GB
RAM (Total available to the project)	100GB

CloudSigma cloud infrastructure summary	
Storage	
Available storage interface	Volume storage
Image format	RAW
SSD capacity (GB)	8TB per image subject to availability.
HDD capacity (GB)	-
Total storage available to the project	1500 GB SSD, 3000 GB HDD
Networking	
Max internal network bandwidth per VM (Gb)	20
Max external network bandwidth per VM (Gb)	10
Max inter-VM latency (ms)	1
Total cloud external network bandwidth (Gb)	10+

CloudSigma has also been providing continued technical support as well as scheduling regular maintenance to the cloud. As a result, the infrastructure has met the expectations of the project.

4.2 Integrated parts

4.2.1 Staging server

The Staging server runs under a DITAS account. Pipelines from repositories in GitHub use the Staging server to run unit and integration tests prior to the upload of components to Docker Hub and then proceed to the deployments in Production.

4.2.2 Production server

The Production testbed consists of two servers, one per UC environment. The purpose of this server is to host the DITAS installation. Each Use Case has its own DITAS installation (Business Model 2). When a component gets moved from Staging to Production it gets installed in both Production servers. Therefore, the production servers, host the whole DITAS SDK, along with a Private Docker Registry for the Data Administrator to publish its images: VDC and Data Access Layer (DAL).

4.2.3 Resources available

We refer here to the resources listed in the section description allocated per use case and deployed by the DITAS EE. This is where DITAS can deploy machines to host computation or data. Each UC defines some resources at machine level. For example: VM with 4 GHz CPU, 8 GB RAM, 100GB SSD Storage/Magnetic storage.

4.2.4 Cloud Data sources per UC

The last block in the diagram correspond the particular use-cases cloud data sources. This data sources are not managed by DITAS, but are original data sources the UC's have beforehand. IDEKO deploys here an InfluxDB database that will host historical data for the three machines participating in the demonstrator, while OSR hosts a MinIO database with blood tests and a MySQL database with patient data.

4.2.5 Definition of Resources available

The resources each use case has defined to be deployable by the DITAS platform in the cloud side of each use case.

Kubernetes has some minimal requirements: the master node cannot run in a node with less than 2GB RAM and 2 cores. This is taken into account when defining resources.

5 Usecases

5.1 IDEKO

The following are the cloud resources defined for the IDEKO Use Case. The infrastructure provider is Cloudsigma and the location is Frankfurt.

Resource group	Resource	Image name	O.S. DRIVE (GB)	CPU (GHz)	RAM (GB)	DATA DRIVE (GB)
1	IDEKO-C1	Debian 9.7 Server	40	4	4	10SSD
1	IDEKO-C2	Debian 9.7 Server	40	8	4	70 SSD
1	IDEKO-C3	Debian 9.7 Server	40	8	12	70SSD

Table 3. IDEKO´s resources available for Frankfurt.

IDEKO will demonstrate two applications that run over the same blueprint, so only one resource group is needed.

5.2 OSR

The following are the cloud resources defined for the OSR Use Case. The infrastructure provider is again CloudSigma and in this case two locations are used, namely Frankfurt and Miami.

Resource group	Resource	Image name	O.S. DRIVE (GB)	CPU (GHz)	RAM (GB)	DATA DRIVE (GB)
1	OSR-C1	Debian 9.7 Server	40	4	4	10SSD
1	OSR-C2	Debian 9.7 Server	40	8	4	70 SSD
1	OSR-C3	Debian 9.7 Server	40	8	12	70SSD

Table 4. OSR´s Resources available for Frankfurt.

Resource group	Resource	Image name	O.S. DRIVE (GB)	CPU (GHz)	RAM (GB)	DATA DRIVE (GB)
1	OSR-C1	Debian 9.7 Server	40	4	6	10SSD
1	OSR-C2	Debian 9.7 Server	40	8	4	70 SSD

Table 5. OSR´s Resources available for Miami.

6 Summary of the resources

The following are the resources reserved in order to run the complete testbed for the two use cases.

IDEKO – app@ideko.es				
	CPU	**RAM**	**Storage**	**IP**
Resources Available (Dep. Engine)	4	4	40	1
Resources Available (Dep. Engine)	8	4	100	1
Resources Available (Dep. Engine)	8	12	100	1
SDK Production server	8	8	50	1
Cloud Data Source	8	8	200	1
TOTAL	**36**	**36**	**490**	**5**

Table 6. IDEKO resources received

OSR – catallo.ilio@hsr.it				
	CPU	**RAM**	**Storage**	**IP**
Resources Available (Dep. Engine)	4	4	40	1
Resources Available (Dep. Engine)	8	4	100	1
Resources Available (Dep. Engine)	8	12	100	1
SDK Production server	8	8	50	1
Cloud Data Source				
TOTAL	**28**	**28**	**290**	**4**

Table 7. OSR resources received

6.1 Created resources details

The following are the details of the created resources.

IDEKO					
	CPU	**RAM**	**DISK**	**OS**	**Auth**
SDK Production server	8	8	50	Ubuntu 16.04	keypair (cloudsigma)
Cloud Data Source	8	8	200	Ubuntu 16.04	keypair (cloudsigma)

Table 8. Created resources IDEKO

OSR					
	CPU	**RAM**	**DISK**	**OS**	**Auth**
SDK Production server	8	8	50	Ubuntu 16.04	keypair (cloudsigma)
Cloud Data Source	8	8	80	Ubuntu 16.04	keypair (cloudsigma)

Table 9. Created resources OSR

7 Conclusions

In this deliverable we have outlined the continuous integration system (Jenkins) and documented the deployment and integration of the DITAS components on the Cloud testbed. Being a geographically distributed project, the DITAS integration using the continuous integration system has helped us to make the development of the project much easier. For example, failures are detected faster and as such, can be repaired faster, or builds and tests can be run in an automatic way, enabling the development teams to easily identify bugs in their code without doing it manually. The developer has no time to lose context, as any issue detected by the CI build will be fixed much faster. To sum up, the CI system has led us to reduce overhead, improve consistency, inspire confidence and mitigate risk.

Regarding the tracking of the deployment, having an easy-to-update centralized online document has helped us to give a fast and deep insight of the status of the project. In addition, the document was well received from the partners, all the component developers made an excellent job updating the document and it was used on every weekly technical conferences to track the deployment status.

Further information has been provided about the IDEKO and OSR use cases with regard to their computing resource requirements. We have also presented information regarding the configuration of the commercial cloud infrastructure used for the staging and production servers and how resources have been allocated to the two use cases.

8 References

[1] Deliverable D5.1 of DITAS project: "D5.1–Specification of integration and Validation Testbed and Cloud Platform for DITAS". © DITAS Consortium. November 2017.

[2] Deliverable D5.2 of DITAS project: "D5.2–Integration of DITAS and case studies validation report". © DITAS Consortium. June 2018.

[3] Deliverable D6.3 of DITAS project: "Impact Report". © DITAS Consortium. June 2018.

[4] DITAS Description of Work (DoA). Annex I to the EC Contract.