



***Facultad  
de  
Ciencias***

**Implementación de un sistema de WaaS  
(Workflow as a Service) sobre la  
infraestructura FedCloud de EGI**  
**Implementation of a WaaS (Workflow as a  
Service) system on the EGI infrastructure  
FedCloud**

Trabajo de Fin de Grado  
para acceder al

**GRADO EN INGENIERÍA INFORMÁTICA**

**Autor: Alberto Otero Márquez**

**Director: Jesús Marco De Lucas**

**Co-Director: Fernando Aguilar Gomez**

**Junio - 2015**

# Abstract

The objective of this project is to implement an example of a research application workflow on a Cloud infrastructure as a first step towards the deployment of a Workflow as a Service (WaaS) system.

In order to do so, relevant generic workflow solutions to support research applications, like Kepler and Taverna, have been considered.

After reviewing different examples of research applications requiring the use of workflows, two of them with similar objectives, TRUFA and Galaxy, have been analyzed in more detail, because of their relevance and similar objectives.

The case for TRUFA, developed by researchers at the Natural Science Museum in Madrid in collaboration with the team at IFCA, and currently supported on high performance computing resources, has been selected to study in detail its adaptation to a Workflow as a Service solution.

We propose a solution over the existing FedCloud IaaS (Infrastructure as a Service) European platform of EGI.eu, using a PaaS (Platform as a Service) open framework (OpenShift) using cartridge technology and scripts to provide developers with a simple but flexible approach.

A first implementation test including the installation in FedCloud resources of the OpenShift framework and involving two TRUFA modules connected as an example of basic workflow has been completed, showing the initial feasibility of the solution proposed. .

**Keywords:** TRUFA, Workflow, Cloud, PaaS, OpenShift

# Resumen

El objetivo de este proyecto es implementar un ejemplo de un workflow de una aplicación de investigación en una infraestructura Cloud, esto es un primer paso hacia el despliegue de un sistema de Workflow as a Service (WaaS).

Para realizarlo, se han considerado las soluciones más relevantes para aplicaciones de investigación, como Kepler y Taverna.

Después de analizar distintos ejemplos de estas aplicaciones que hacen uso de workflows, dos de ellas con objetivos similares, TRUFA y Galaxy, han sido estudiadas en mayor profundidad, dada su relevancia y similar contexto.

El caso de TRUFA, desarrollado por investigadores en el Museo Nacional de Ciencias Naturales en Madrid en colaboración con el equipo del IFCA, y actualmente soportada en recursos de computación de alto rendimiento, ha sido seleccionado para estudiar en detalle su adaptación a una solución de Workflow as a Service.

Proponemos una solución sobre la plataforma existente IaaS (Infraestructura como servicio) FedCloud de EGI.eu, usando un framework abierto de PaaS (Plataforma como servicio) OpenShift, utilizando tecnología de cartuchos y scripts para proporcionar a los desarrolladores una simple y a la vez flexible herramienta.

Se ha realizado una primera implementación, incluyendo la instalación en recursos de FedCloud del framework de Openshift e involucrando dos módulos de TRUFA conectados como un ejemplo de un workflow básico, mostrando la factibilidad de la solución propuesta.

Palabras clave: **TRUFA, Workflow, Cloud, PaaS, OpenShift**

# Table of Contents

<b>Abstract</b>	ii
<b>Resumen</b>	ii
<b>Table of Contents</b>	iii
<b>List of Figures</b>	v
<b>Acknowledgements</b>	vii
<b>1 Introduction</b>	1
<b>2 State-of-the-art Workflow Solutions</b>	2
2.1 Kepler	2
2.2 Taverna	3
2.3 myExperiment	5
2.4 Galaxy	8
2.5 TRUFA	10
2.6 Lifewatch Marine VRE	11
2.7 Chipster	12
2.8 Summary and comparison of the different solutions	13
<b>3 e-Infrastructure Context</b>	15
3.1 What is an e-Infrastructure?	15
3.2 European Grid Infrastructure - EGI	15
3.3 INDIGO-Data Cloud	20
3.4 OpenShift	21
<b>4 Application to the LifeWatch project: Galaxy &amp; TRUFA</b>	26
4.1 What is LifeWatch?	26

## *Table of Contents*

---

4.2	Galaxy . . . . .	26
4.3	TRUFA . . . . .	29
<b>5</b>	<b>Implementation of the solutions and comparative Analysis . . . . .</b>	<b>35</b>
5.1	Problem Description . . . . .	35
5.2	Solution proposed . . . . .	35
<b>6</b>	<b>Conclusions . . . . .</b>	<b>49</b>
 <b>Appendices</b>		
<b>Appendix 1: List of acronyms . . . . .</b>		<b>51</b>
<b>Appendix 2: BIND DNS Server Installation . . . . .</b>		<b>52</b>
<b>Appendix 3: Creation of the CUTADAPT cartridge . . . . .</b>		<b>57</b>
<b>Appendix 4: Workflow script . . . . .</b>		<b>62</b>
<b>Appendix 5: Execution of the script . . . . .</b>		<b>64</b>
<b>Appendix 6: Errors . . . . .</b>		<b>69</b>
<b>Bibliography . . . . .</b>		<b>71</b>

# List of Figures

2.1	Taverna Infrastructure. . . . .	4
2.2	myExperiment popular interfaces. . . . .	6
2.3	myExperiment architecture. . . . .	7
2.4	Elastic Computer Cloud Cluster platform. . . . .	9
3.1	Scientific applications which run on EGI. . . . .	16
3.2	EGI Core Infrastructure Platform. . . . .	16
3.3	EGI Cloud Infrastructure. . . . .	17
3.4	INDIGO Global architecture. . . . .	21
3.5	OpenShift Architecture. . . . .	23
3.6	Creating an application on Openshift. . . . .	24
3.7	Creation and deployment of a PHP app on Openshift. . . . .	25
4.1	Galaxy application architecture. . . . .	27
4.2	Overview of the TRUFA pipeline. . . . .	32
4.3	List of software available at TRUFA. . . . .	33
4.4	TRUFA web interface. . . . .	34
5.1	Overall representation of the infrastructure designed for TRUFA Cloud. . . . .	37
5.2	Keys used to acces the virtual machines via ssh. . . . .	38
5.3	Specifications of the TRUFA Server Host. . . . .	39
5.4	Client perspective of Openshift. . . . .	40
5.5	OpenShift configuration on the server. . . . .	41
5.6	Overview of the node . . . . .	42
5.7	Details of the python application created in the node. . . . .	43
5.8	Pinging the node from the server. . . . .	43
5.9	Pinging the server from the node. . . . .	44
5.10	Detailed OpenShift Layer Components . . . . .	44
5.11	OpenShift Layer of a Complete Deployment of TRUFA Cloud. . . . .	48

## *List of Figures*

---

1	CUTADAPT Cartridge folder . . . . .	57
2	Contents of the bin folder . . . . .	57
3	Content of the metadata, version and cutadapt-1.3 folders, respectively. . . . .	59
4	Web interface for uploading customized cartridges on OpenShift. .	60
5	Output after the deployment of the CUTADAPT cartridge on a node.	60
6	List of cartridges after the installation of the CUTADAPT cartridge.	61
7	Error encountered during the OpenShift installation . . . . .	70
8	Hard drive configuration on the server . . . . .	70

# Acknowledgements

I would like to thank my Director Jesus De Marco, for his work conducting this project and for making time to work and supervise the whole project, and also for providing me with the computational resources that I needed at IFCA. I know I have not been easy to manage, but we made it.

I would also like to express my gratitude to my Co-Director, Fernando Aguilar, for helping me through my constants questions, for his technical help in all the work and for his helpful comments on my work.

I would not forget to thanks my friend Guzman, for his inestimable help during the coding part of this project, this could not have been done without your help. You even put my project before your exams. Thank you buddy.

In addition, I want to thank my brother, for helping me building this final document and for listening and guiding me through the project, I know I can be tiresome sometimes but you know you are too.



# 1. Introduction

Scientific workflows are quickly becoming recognized as an important unifying mechanism to combine scientific data management, analysis, simulation, and visualization tasks [52]. They are used to help understanding and modeling complex scientific experiments and applications, providing a high-level declarative way to define the goals of a particular experiment modelled by a workflow, rather than focusing on how it will be executed.

A more general definition of Workflow is the one given by the Workflow Management Coalition [53] regarding the business domain in 1996, which defines workflow as “*The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.*”

*“The variety of the tasks that can be performed in a workflow can be implemented using local services, web services, scripts and sub-workflows. Each of the components is in charge for a small functionality, so many components need to be put together in a pipeline in order to obtain a fully functional workflow that can perform the task desired.”* [3]

The process of linking components is known as *workflow composition* [4], the production of a conceptual model for the described scientific analysis. Representations of all required heterogeneous resources are integrated into this single workflow, thus abstracting superfluous detail and concentrating on the goal of the experiment.

The use of workflows enables the delegation of data processing to remote infrastructure and makes it possible to define and launch larger and more complex workflows from personal desktops. [3]

Web services [5] is the technology usually chosen to delegate most of the core computation in workflows, while local services and scripts are used to perform data format conversion procedures and other auxiliary tasks. Another advantage of workflows is the automatization of repetitive processing stages, which can stimulate the pace of research and overall productivity of experimentation. It also simplifies the reproducibility and preservation of the analysis performed.

## 2. State-of-the-art Workflow Solutions

In the following subsections some of the most interesting workflow solutions will be described and analyzed: Kepler [6], Taverna [7] and Galaxy [8].

### 2.1 Kepler

#### 2.1.1 Overview

Kepler [9] is oriented to help scientists, analysts and computer programmers to design, execute and share models analyses in scientific and engineering fields. Kepler offers a graphical user interface that allows users to create their own scientific workflows [10]. The service also helps user to share, and reuse the workflows created by other members of the scientific community, improving the usability and the time needed to create certain workflows. Kepler is a java-based application that is supported under Windows, OSX, and Linux operating systems.

#### 2.1.2 Implementation

Kepler inherits modeling and design capabilities from Ptolemy II [11], a framework developed at the University of California, Berkeley, including the Ptolemy graphic user interface, workflow scheduling and execution patterns. Kepler also inherits from Ptolemy the actor-oriented modeling paradigm, which separates workflow components from the overall workflow orchestration, offering reusability. Kepler also includes Ptolemy components aimed at scientific applications: remote data and metadata access, data transformations, data analysis, interfacing with legacy applications, Web service invocation and deployment and provenance tracking.

Kepler also offers web-based execution solutions, like Hydrant and SciencePipes: Hydrant provides the means necessary for users to deploy their workflows

on the web and SciencePipes allows users to connect to real biodiversity data and create visualizations.

### 2.1.3 Scope of Applications

Kepler is being used in a wide range of fields, including chemistry, ecology, geology, molecular biology, oceanography and phylogeny. Kepler (and its suites including Serpens) are used by several application from the field of Nuclear Fusion and Astrophysics to manage the workflows runs on Grid. More information about EGI-InSPIRE can be found at <https://www.egi.eu/about/egi-inspire/>. Behind the scenes, Science Pipes is based on the Kepler scientific workflow software which is used by professional researchers for analysis and modeling. More information about Science Pipes can be found at <http://sciencepipes.org>.

### 2.1.4 Summary

Kepler is an open source workflow framework that has been adopted by scientists that work in the fields of biology, ecoinformatics and geoinformatics. Kepler offers a limited set of templates to easily create new workflows for this scientific fields, however its level of complexity hinders when build new ones, and is also hard to support the whole framework efficiently in the Cloud.

## 2.2 Taverna

### 2.2.1 Overview

Taverna [12] allows to automatize multi-step analysis that use several web services. Taverna enables their users to create their models defining their final goals without the need to detail how the services or the processes will be executed: Taverna will automate the solution providing a pipeline model adequate to the user demand.

Taverna also offers data conversion between services which are not entirely data compatible. Another feature of Taverna is the quick incorporation of new services without the necessity of coding: Taverna currently offers more than 3500 services, including access to local and remote resources and different analysis tools.

Taverna also covers thoroughly the result section of the experiments providing detailed information about the execution, including the list of services that were executed and when, which inputs were used by each service, and the outputs produced. Finally it also offers the possibility of sharing the workflows created in Taverna on the myExperiment platform.

### 2.2.2 Implementation

The Taverna suite is written in Java and includes the Taverna Engine that powers both Taverna Workbench (the desktop client application) and Taverna Server (which executes remotely the workflows). Taverna is also available as a Command Line Tool enabling the direct execution of workflows from a terminal without requiring a GUI.

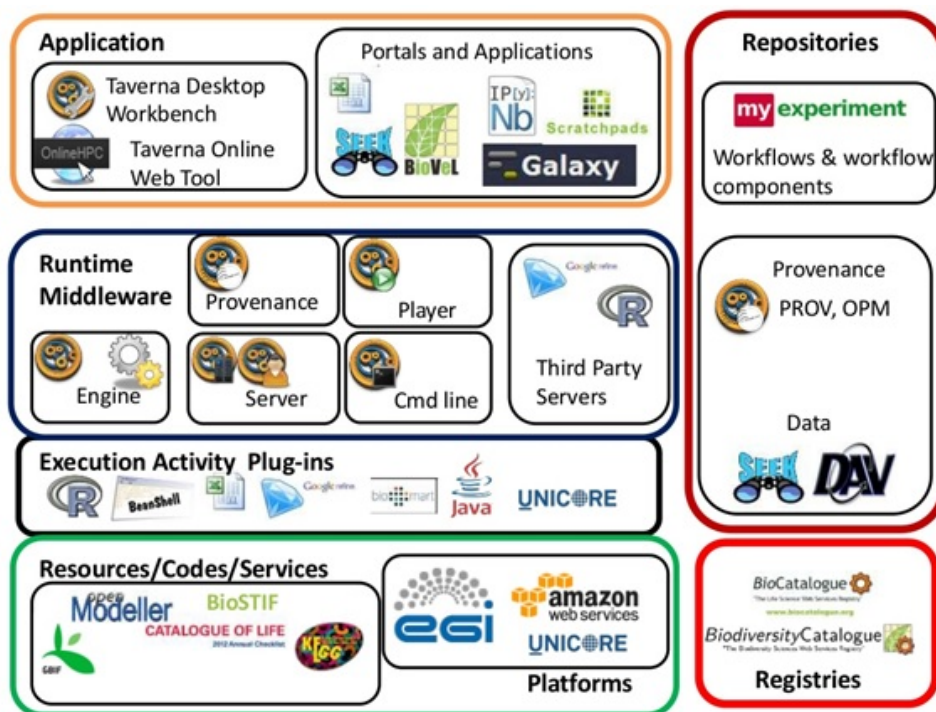


Figure 2.1: Taverna Infrastructure. Taken from [13]

### 2.2.3 Applications

Taverna has been used in the fields of bioinformatics, astronomy, chemistry, health informatics and others. Many examples of Taverna use can be found in the bioinformatics field, although Taverna is actually domain independent. This means that Taverna can be applied to a wide range of fields, for example it has been used for the composition of music using Web services for synthesis.

### 2.2.4 Summary

Taverna enables a scientist who has a limited background in computing, limited technical resources and support, to construct highly complex analyses over data and computational resources that can be both public and private.

Taverna allows to perform the automatization of experimental methods and the use of a number of different services from a diverse set of domains, biology, chemistry and medicine including music, meteorology and social sciences.

Taverna, as Kepler, is a complex platform, that requires a non negligible support effort for an efficient exploitation.

## 2.3 myExperiment

### 2.3.1 Overview

myExperiment [14] is a platform oriented to the use of workflows and based in the principle of reuse, that can be effective in multiple cases:

- a) The reutilization of a workflow with different parameters and/or input data, or the possibility of minor modifications to the workflow for specific purposes.
- b) Sharing of workflows among different scientists that work in similar projects, so each one can help each other in matters of coding and experience, sharing and spreading the practice on workflow design.
- c) Workflows, including their components and workflow patterns, can be re-used in fields that were not considered as the original target of their design[14].

In summary, the main objective of myExperiment is to provide a platform for scientists to share and work collaboratively re-using workflows, taking care of the social and technical challenges.

### 2.3. myExperiment

---

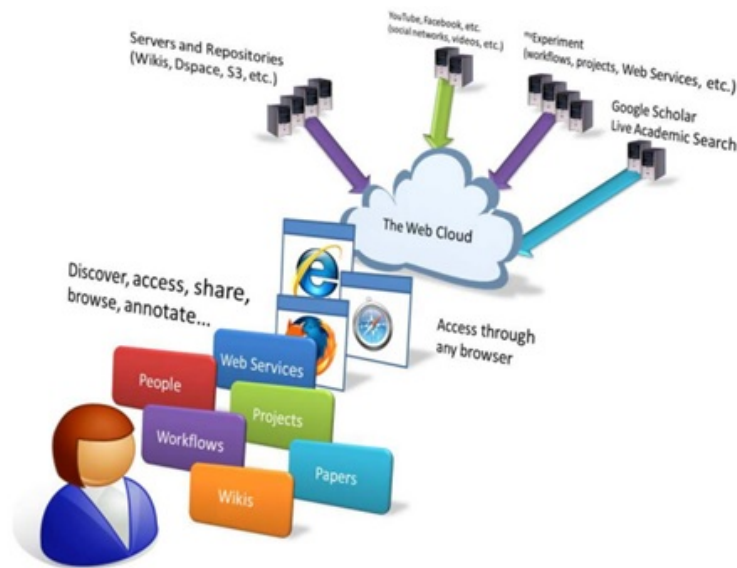


Figure 2.2: myExperiment brings functionality to the user through friendly interfaces. Taken from [14].

#### 2.3.2 Implementation

myExperiment was designed according to an interpretation of the Web 2.0 design principles in the context of the so called Virtual Research Environments. The architecture of an instance of myExperiment is represented in Figure 2. All the interfaces to myExperiment are accessed via HTTP protocol. The user access the myExperiment platform via an HTML based Web interface, while external applications can be accessed through other interfaces. The HTML interface uses JavaScript and AJAX to improve the interactive experience, while the RESTful API makes possible the construction of Rich Web Applications and mashups [14].

### 2.3. myExperiment

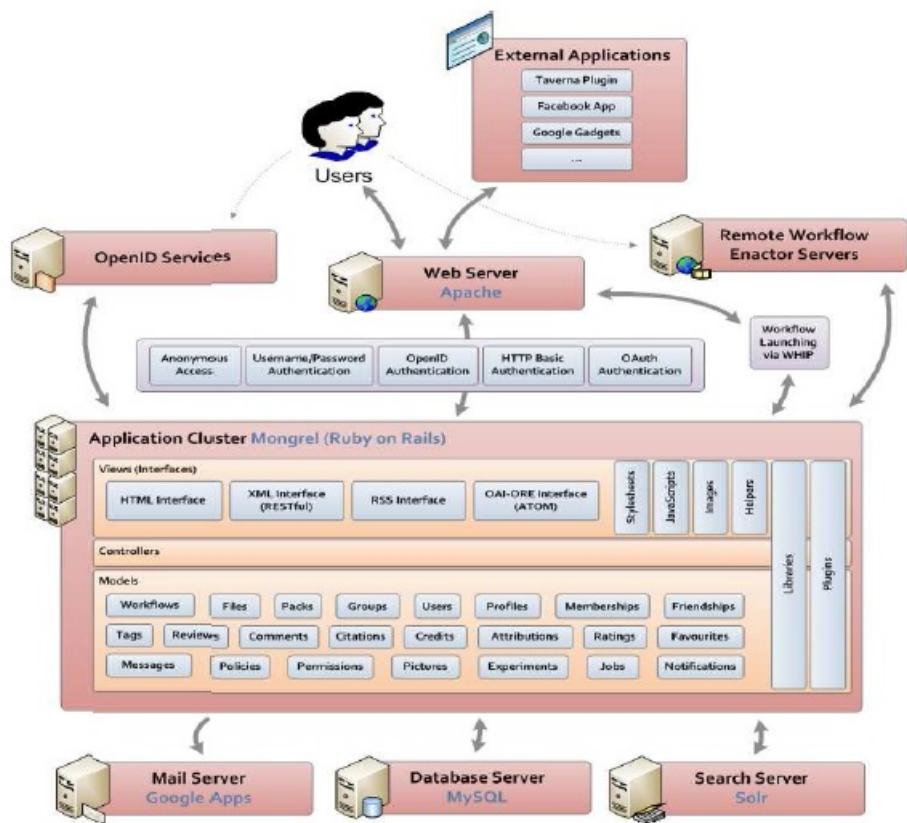


Figure 2.3: Implementation architecture of a myExperiment server instance. Taken from [15]

#### 2.3.3 Applications

myExperiment can be used to share any kind of workflow, therefore, its field of application is not limited to a singular area, although it seems that myExperiment is mostly used within the scientific community.

myExperiment hosts different types of workflows, including scientific workflows using Taverna, Kepler and Galaxy.

### 2.3.4 Summary

myExperiment is the largest public repository of scientific workflows, it offers a service that none of the other workflows applications offer, the possibility to use an existing workflow to adapt it for another purpose, changing the parameters and the data used.

In the context of myExperiment, sharing and collaborating in the use of workflows the way how the user interacts has a lot of impact on the user experience, and for that, the search engine, including a better way to show the results and providing good filters for searches, as upgrading the interface of the web are necessary goods for the success of myExperiment.

## 2.4 Galaxy

### 2.4.1 Overview

Galaxy [16] [17] is an open source, web-based platform for data intensive biomedical research, it allows users to organize and manipulate data from existing resources in different ways. One of the import features of Galaxy is memory, every action of the user is recorded and stored in the history system so any user can repeat and understand a complete computational analysis.

Galaxy allows users to conduct independent queries on genomic data from different sources and then combine or refine them, perform calculations, or extract and visualize corresponding sequences or alignments.

Galaxy differs from existing systems in its specificity for access to, and comparative analysis of, genomic sequences and alignments.

### 2.4.2 Implementation

Galaxy consists in several independent software components that work together to perform tasks. The central core component orchestrates the action, executing the queries and keeping track of user histories, while the user interface and the operation, tool or output libraries are implemented separately. The communication with other sites is handled by the core component. [18]

The user interface communicates with the core via HTTP requests, using the GET or POST methods. The core provides an API consisting of the requests it is prepared to handle, for example using a tool or retrieving a user's query history for a particular assembly of a genome.

The Galaxy core component and operation libraries are written in C, the initial UI (called HUI for History User Interface) is written in Perl for convenient text



manipulation and CGI access, but one could use any language that can generate an HTTP request.

The use of an HTTP API is justified by the great compatibility of user interfaces that can be used, which do not have to be running on the same server. This allows any site on the Web to be able to create its own user interface for Galaxy by crafting the appropriate HTTP requests, and individual researches can use the API directly for programmatic access to Galaxy's features.

The benefits of this design are extensibility (easy of adding new tools and interfaces) and convenient division of labor and expertise among programmers. The design also gives Galaxy flexibility which makes possible different kinds of deployments of Galaxy. In addition to using the public Galaxy server, a user can use his own instance of Galaxy, or he can create a cloud-based instance of Galaxy. Another option is to use one of the public Galaxies hosted by other organizations.

A good example of the flexibility mentioned before is the Elastic Compute Cluster instance for Galaxy which has been designed by the Universitat Politècnica de Valencia (UPV) and Institut national de la recherche agronomique (INRA) which combines Galaxy, Infrastructure Management and Elastic Compute Cloud Clusters, providing automatic elasticity based on the batch queue workload [19].

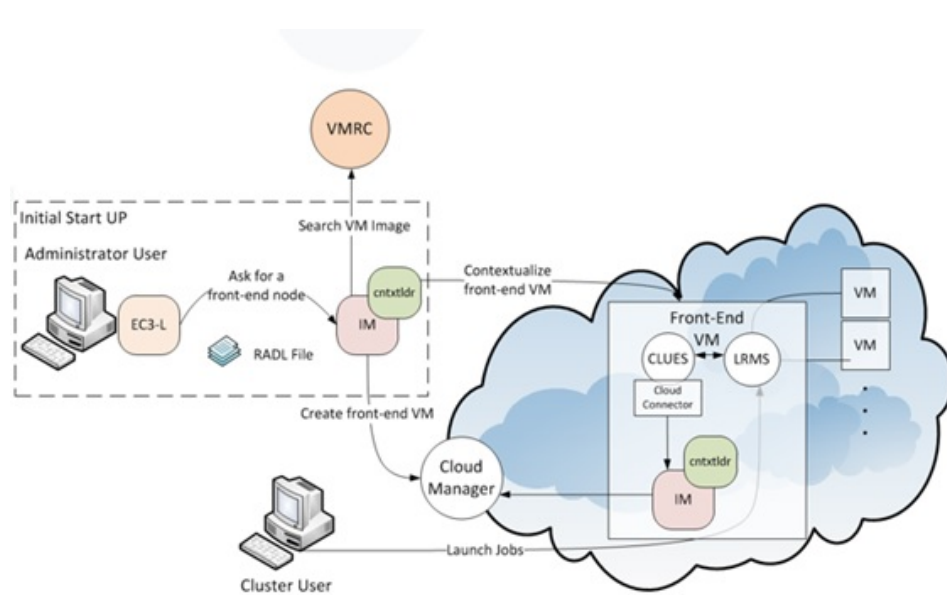


Figure 2.4: Elastic Computer Cloud Cluster platform. Taken from [19]

### 2.4.3 Applications

Galaxy offers a new set of interactive tools for large-scale genome analysis. The application field of Galaxy is restricted to bioinformatics.

### 2.4.4 Summary

Galaxy allows large-scale analyses that previously required users to have some programming experience and database management skills. The Galaxy history page is simple to use, and is able to handle large genome annotation data sets. Users have the ability to perform multiple types of analyses (e.g., query intersections, subtractions, and proximity searches) and then display the results using existing browsers (e.g., the UCSC Genome Browser or Ensembl). The only thing Galaxy's missing is compatibility with different areas not related to bioinformatics.

## 2.5 TRUFA

### 2.5.1 Overview

TRUFA [20] stands for TRanscriptome User-Friendly Analysis, an informatics platform based on a web interface that generates the outputs commonly used in de novo RNA-seq analysis and comparative transcriptomics.

TRUFA offers the next services, raw read cleaning executed, transcript assembly and annotation, and expression quantification. TRUFA is highly parallelized and benefits from the use of high performance computing resources. TRUFA gives the user an easy, fast and valid analysis on RNA-seq data.

### 2.5.2 Implementation

TRUFA was developed at the Instituto de Fisica de Cantabria (IFCA, Spain), the platform is written using JavaScript, Python and Bash and it is currently installed in the ALTAMIRA supercomputer at IFCA. The platform is highly parallelized both at the pipeline and program level, and can access up to 256 cores per execution instance for certain components of the pipeline.

### 2.5.3 Applications

TRUFA is designed to exclusively perform tasks related to de novo RNA-seq analysis, for this reason, the most important fields of application are evolutionary biology, ecology, biomedicine and computational biology.

### 2.5.4 Summary

TRUFA provides a set of the most common tasks to perform a whole de novo RNA-seq analysis. It allows scientists which does not have bioinformatics skills or access to fast computing services. TRUFA works in an efficient, consistent and user-friendly manner, based on a pipeline approach. TRUFA integrates some widely used quality control programs in order to obtain optimization of the assembly process in the RNA-seq analysis.

## 2.6 Lifewatch Marine VRE

### 2.6.1 Overview

The LifeWatch Marine Virtual Research Environment (VRE) [21] assembles several marine resources, data bases, data systems, web services, tools, etc. into one marine virtual research environment. The Marine VRE allows researchers to retrieve and access data resources holding marine biodiversity and ecosystem data, a range of data systems on species names, traits, distribution and genes.

A set of online tools is available to facilitate data analysis of marine biodiversity and ecosystem data, and analysis can be performed on data from known data resources and/or data uploaded by the users themselves. Should a researcher need a specifically adapted service, the Marine VRE gives the possibility to build his/her own marine virtual lab, making use of the web services that access and process data.

Service catalogues and 'how to' manuals will guide the users during the development of their own system. The Marine VRE is already looking to the future, working to further increase the integration and interaction between its components.

### 2.6.2 Implementation

The Lifewatch Marine VRE is a web portal that contains in an organized way a set of web services, applications and scientific workflows, but it does not execute or perform any operation by itself, it merely hosts the references to BioVel (Biodiversity Virtual e-Laboratory) or Taverna in the matter of workflows.

### 2.6.3 Applications

The Lifewatch Marine VRE brings together relevant resources for Web-based marine research: data systems, Web services, workflows, online tools, etc. in one environment in the context of LifeWatch.

### 2.6.4 Summary

LifeWatch marine VRE supports marine environmental research and enables scientists to access resources and conduct analysis without having to install or configure any additional software. With everything accessible in one place, scientists can access data resources and tools and collaborate together. They can analyze their data in conjunction with data from other sources.

## 2.7 Chipster

### 2.7.1 Overview

Chipster [22] is a versatile data analysis platform with interactive visualizations and workflows. It offers a comprehensive collection of analysis tools for next generation sequencing (NGS), microarray and proteomics data. The NGS functionality applies to analysis from quality control and alignment to downstream applications such as pathway analysis and motif detection and more analysis tools are added all the time. The built-in Chipster [23] genome browser allows users to visualize reads and results in their genomic context. The microarray functionality covers expression and allows users to integrate expression data with different data.

### 2.7.2 Implementation

Chipster's platform is technically based on a desktop application user interface, a flexible distributed architecture, and the ability to integrate many types of analysis tools.

The Chipster client software is a full graphical Java desktop application, offering an intuitive user interface with highly interactive visualizations and an overall smooth user experience. To make the client installation and updates as easy and automatic as possible, Chipster uses the Java Web Start technology.

Chipster offers high compatibility that makes possible the integration of almost any kind of tool, regardless of their implementation. As R/Bioconductor provides a rich collection of analysis functionality for microarray and NGS data, Chipster offers a strong support for R integration: Wrappers manage communication with R processes and pool them for rapid responsiveness, and several R versions can be run side-by-side. Integration of command line tools is also supported and can be accomplished even automatically. The tool selection offered by the local server can be augmented by external Web services (SOAP).

Chipster is a client-server system. Server architecture allows tasks to be performed in optimal places: for example, interactive visualizations happen in the

## 2.8. Summary and comparison of the different solutions

---

client, whereas the actual analysis tasks are processed by computing services, which can be run on server machines with ample CPU and memory resources. This way the user can run several analysis tasks simultaneously without burdening his computer power. In addition, there is no need to install any analysis tools or libraries to the user's computer as they are installed and maintained centrally in the computing servers. To avoid transferring data multiple times between the client and server, a caching mechanism is used. The caching extends to multi-user scenarios thanks to Chipster's cryptographically strong data identifiers: When a previously saved analysis session is opened from a different computer, possibly by a different user, the system still uses the original cached copy of the data and does not transfer it again to the server side.

### 2.7.3 Applications

Chipster enables biologists to access a powerful collection of data analysis and integration tools, and to visualize data interactively. Consequently we can conclude that the most important fields of application for Chipster are evolutionary biology, ecology, biomedicine and computational biology.

### 2.7.4 Summary

Taken together, Chipster is a user-friendly open source analysis software for microarray and other high throughput data. Its intuitive user interface brings a comprehensive collection of analysis methods within the reach of experimental biologists, enabling them to analyze and integrate different data types such as gene expression, miRNA and aCGH. The analysis tool arsenal is complemented with powerful interactive visualizations, allowing users to select datapoints and create new gene lists based on these selections. Importantly, users can save the performed analysis steps as reusable, automatic workflows. Chipster promotes collaboration at several levels: While biologists can collaborate by sharing workflows and analysis sessions, bioinformatics core facilities can also easily share their expertise with research groups by providing ready-made workflows and new analysis tool scripts. Finally, Chipster integration is an easy way for analysis method developers to provide their tool with a graphical user interface, thereby making it available for a wider group of users.

## 2.8 Summary and comparison of the different solutions

After reviewing some of the most known workflow solutions we can state that there is no final tool to suit everyone's purposes.

## 2.8. Summary and comparison of the different solutions

---

In order to compare the different solutions we are going to label them into different categories according to their application field.

First, Kepler and Taverna are both workflow solutions that target the execution of general purposes jobs, such as those related to chemistry, ecology and bioinformatics. In one hand, Kepler aims to be the best open source scientific workflow system available, by enabling multiple groups from different scientific disciplines to easily create, support and make available domain-specific Kepler extensions and by developing new core features that will transform Kepler into a more comprehensive scientific workflow system offering full support for data, workflow, service and project management. On the other hand, Taverna excels at the automatization of pipelines, data conversion between services and a pretty solid analysis result system.

In the next group are included Galaxy, TRUFA and Chipster, in this category are included tools related to genome experiments, Next Generation Sequencing e.g DNA and RNA-sequencing. Galaxy is especially powerful when coming to data intensive biomedical research, and it includes a special feature, that records every action made by the users in order to allow any user to repeat and understand a complete computational analysis. TRUFA is a very specific solution that performs RNA-sequence analysis in a pipeline, it has a complete set of tools and it's presented in a very attractive and user-friendly way. Hipster includes a genome browser that allows users to visualize reads and results in their genomic context and a microarray functionality that covers expression and the data expression.

The last of the solutions studied is Lifewatch Marine VRE, that is not actually a workflow solution rather than a web portal that stores and displays in a proper way different resources to oceanography studies.

myExperiment is a tool that allows users to share and work collaboratively in scientific workflows, myExperiment has compatibility with workflows created on Taverna, Kepler or Galaxy, and therefore has a good impact on the relevance of these tools as it extends the functionality of these systems.

## 3. e-Infrastructure Context

In this section some of the existing and upcoming e-infrastructures to deploy workflow applications will be studied, such as the European initiative known as Fed-Cloud(EGI Federated Cloud) and the INDIGO-Data Cloud solution.

### 3.1 What is an e-Infrastructure?

*“e-Infrastructure refers to a combination and interworking of digitally-based technology (hardware and software), resources (data, services, digital libraries), communications (protocols, access rights and networks), and the people and organizational structures needed to support modern, internationally leading collaborative research be it in the arts and humanities or the sciences. “ [24].*

### 3.2 European Grid Infrastructure - EGI

#### 3.2.1 What is EGI?

The European Grid Infrastructure [25] is an e-infrastructure created to allow scientists and researchers to help in their work by providing not only the computing tools but the opportunity to share information securely, analyze data efficiently and collaborate with other researchers worldwide.

Many of the researches conducted nowadays are too complex to be resolved by a single scientist or a single research team, modern challenges rely on large projects, cross-country collaborations and computing power to analyze huge amounts of data, EGI was founded to make possible this kind of works and to help any scientific group no matter they size.

### 3.2. European Grid Infrastructure - EGI

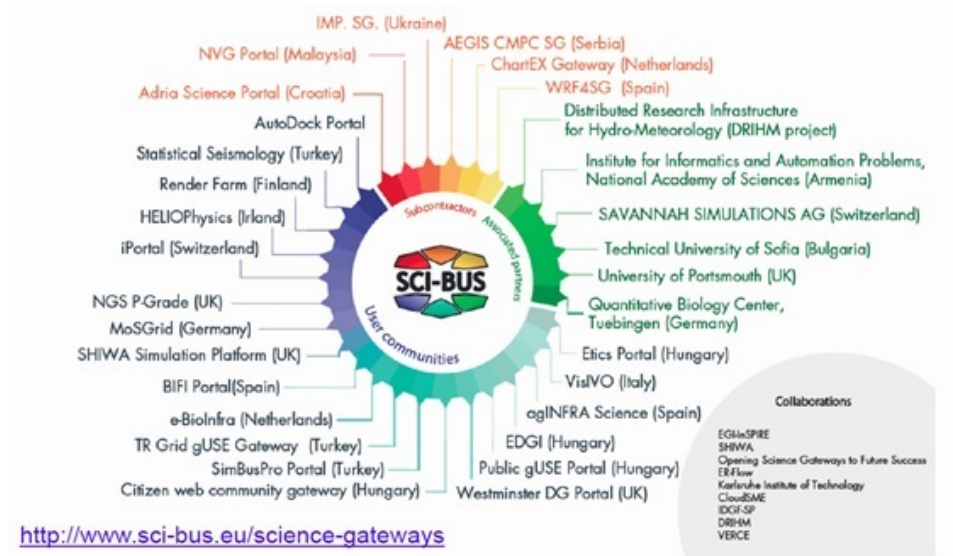


Figure 3.1: Some Scientific applications that run on EGI. Taken from [26]

#### 3.2.2 EGI Infrastructure

##### EGI's Core Infrastructure

The services that federate and integrate the functional services deployed in the production infrastructure can be seen on the following Figure

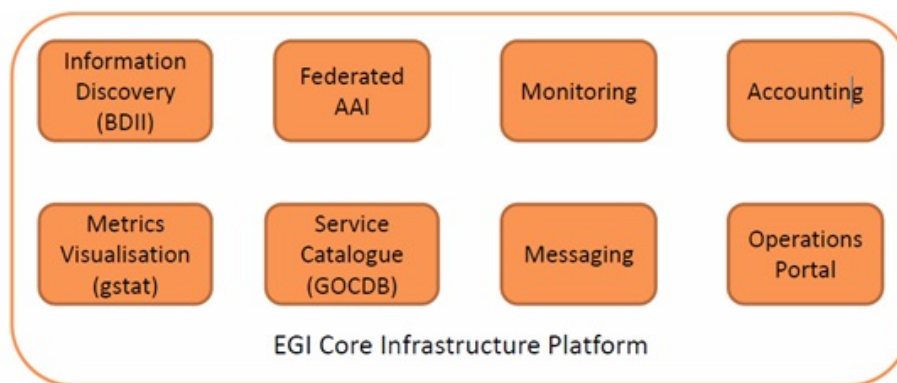


Figure 3.2: EGI Core Infrastructure Platform. Taken from [30]



#### EGI's Cloud Infrastructure

The cloud infrastructure is composed by the EGI Collaboration Platform, that handles the Image Metadata Marketplace and the Image repository, the EGI Core Infrastructure Platform and the EGI Cloud Infrastructure Platform, that contains the Virtual Machine Management, Storage Management and the Information, and supports OpenStack and OpenNebula as Infrastructure as a Service (IaaS).

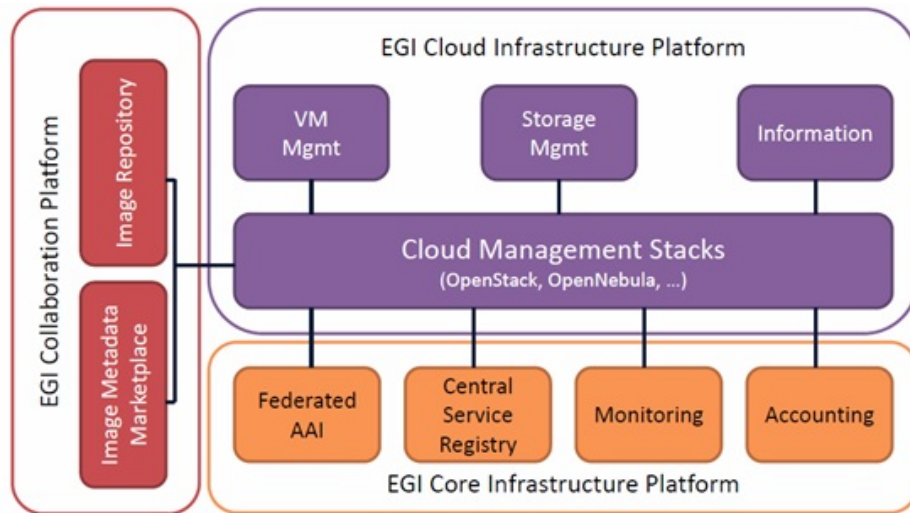


Figure 3.3: EGI Cloud Infrastructure. Taken from [30]

#### 3.2.3 EGI Services

##### What EGI offers to a scientist?

EGI differentiates from any other resource provider by providing scientist with the exact tools that make much easier their computational tasks and their collaboration work. Some of the unique features that EGI offers are:

- Total control over deployed applications
- Elastic resource consumption based on real needs

- Immediately processed workloads , no more waiting time
- An extended e-Infrastructure across resource providers in Europe
- Service performance scaled with elastic resource consumption
- Single sign-on at multiple, independent providers

#### **The EGI Federated cloud: cloud compute and cloud storage services [27]**

The EGI Federated Cloud is a seamless grid of academic private clouds and virtualised resources, built around open standards and focusing on the requirements of the scientific community. Its core elements are:

- Operations Coordination is a set of management and coordinating activities ensuring that operational activities across the federated infrastructure work seamlessly, without fragmentation. The coordination binds the infrastructure so that the services are delivered at the agreed service level.
- Technology Coordination ensures continuous technological innovation through sourcing of software components from technology providers to meet the current and emerging needs of both researchers and resource centers.
- Security Coordination ensures a secure and stable infrastructure to mitigate threats, enhance services, and give users the protection and confidence they demand from a service. A secure infrastructure federation is naturally a top priority.
- Federated Operation Services brings together the operational tools, processes and people necessary to guarantee standard operation of heterogeneous infrastructures from multiple independent providers, with lightweight central coordination (monitoring, accounting, configuration and so on).
- Helpdesk Support provides professional, reliable and efficient technical support to guarantee a well-run infrastructure with improved productivity and usability for the customers. It requires certification so it is only provided to Resource Centers already federated within EGI.
- Specialized Consultancy offers tailored technical and management advice to help partners and clients make the most out of e-Infrastructure technologies.

#### **High-throughput Data Analysis infrastructure: analyze your research data [28]**

EGI offers the “High-throughput Data Analysis” solution to enable users to:

- Access transparently distributed resources with uniform interfaces
- Be authenticated in a uniform way in different sites.
- Autonomously manage their communities structure and to regulate access to services and data throughout the infrastructure.
- Access resources assigned through a central allocation process.

Users are able to access distributed resources through common standard interfaces uniformly available in the different resource centers. The user can manage his data and execute and control the computational tasks using common services and APIs.

Users’ identity is uniformly recognized in the whole infrastructure. This reduces workload, for example, by making it possible for a computational task running in one resource center to access data stored by the user in another center.

#### **Community Driven: Innovation and Support [29]**

EGI offers the Community-driven Innovation and Support solution for potential and existing users to:

- Facilitate access to EGI technical resources by informing them of the administrative and technical requirements.
- Enable data- and computation-intensive research, by providing consulting support to structure research datasets, enabling a user-friendly interface or application.
- Increase the efficiency and effectiveness of the research process.
- Innovate the research approach and methodology, by providing expertise about the digital research procedures from computation experts and like-minded researchers already aware of the new high performance computation research paradigm.
- The ultimate purpose is to create an environment where scientists, regardless of their discipline, can make the most of the European Grid Infrastructure for their research without having to become experts in the infrastructure itself.

### **3.3 INDIGO-Data Cloud**

INDIGO stands for INtegrating Distributed data Infrastructures for Global Exploitation, the INDIGO-Data Cloud project [31] targets the development of a framework, usable on top of existing and evolving e-Infrastructures, capable of supporting the growing needs of scientists to be able to store, manage, share and process research data.

This objective will be achieved by implementing a PaaS component, oriented to the connection of heterogeneous e-infrastructure resources and to build and support the applications of the researchers.

#### **3.3.1 PaaS architecture and implementation**

The design and implementation of a PaaS layer will allow scientific communities to exploit, in a powerful and high-level way, several heterogeneous computing and data e-infrastructure such as: IaaS Cloud, Helix Nebula, EGI Grid, EGI Federated Cloud, PRACE, HPC, EUDAT, etc.

It will be possible to process large amounts of data and to exploit the efficient storage and preservation technologies and infrastructure already available in the European e-infrastructure, with the appropriate mechanisms to ensure security and privacy.

The PaaS Layer will provide the following features:

- **Transparency:** description of services or applications deployment must be independent from execution infrastructures, so the request can eventually be split into sub-requests and run on diverser resources.
- **Error management:** failure or errors of service/application execution should be caught in order to decide if it is possible to deploy it exploiting different solutions.
- **Elasticity management:** description of high-level auto scaling rules will be considered.

It will implement a solution, Geo-deployment Service, to deploy in a transparent and powerful way both services and applications in a distributed and heterogeneous environment made by several different infrastructures (EGI Grid, EGI Fed Cloud, Helix Nebula, etc).

The developed solution will provide access optimization and an abstraction layer that will allow users and higher layers to access storage as they were interacting with a unified, (while distributed and heterogeneous) federated data storage.

### 3.4. OpenShift

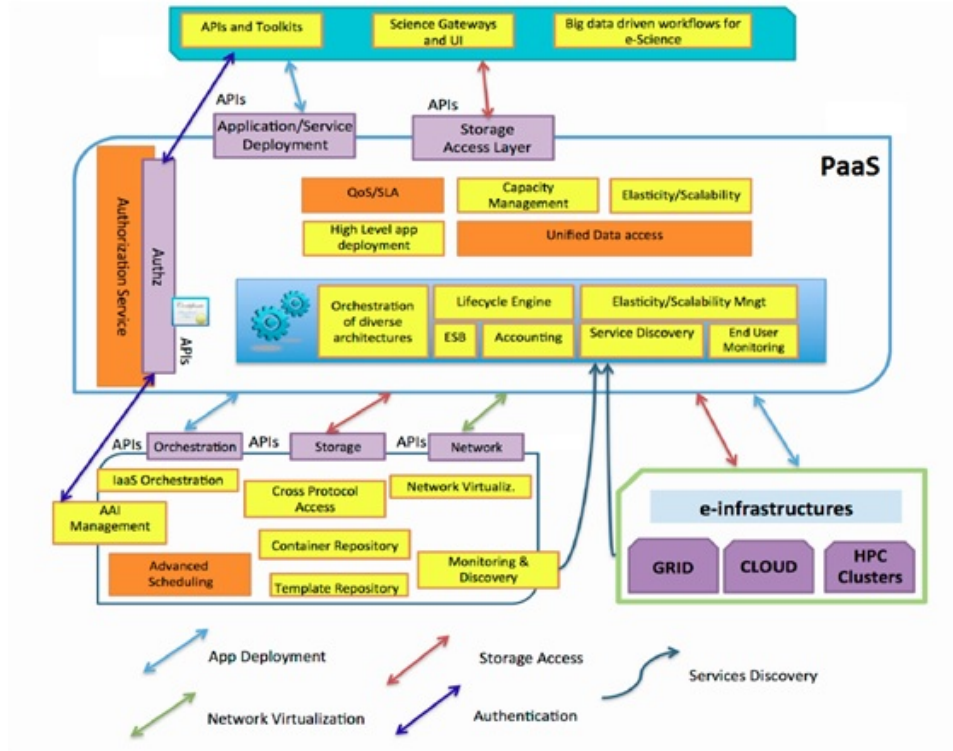


Figure 3.4: INDIGO Global architecture. Yellow nodes represent implementation based on already available solutions; Orange newly implemented services. Taken from [31]

## 3.4 OpenShift

OpenShift [32] is a Platform as a Service (PaaS) solution that can be deployed right in top of the most existing Infrastructure as a Service (IaaS). OpenShift offers a Multilanguage, auto scaling and elastic platform that allows deploying applications using frameworks and its deployment is based on a Software as a Service (SaaS) definition model.

OpenShift makes possible to administrators to serve in a fast way to the needs of the developers by deploying a PaaS that simplifies the process of deploying an application. OpenShift offers a multitenant cloud architecture and granular that allows to deploy in a simple way applications under demand, according to the

choice of programming language, development environment and auxiliary tools. OpenShift supports Java, Ruby, PHP, Python and Perl, and data bases like MySQL and PostgreSQL, the continuous integration service Jenkins and many other.

OpenShift is based on the concept of “cartridge” that will be used to provide services to the platform. The cartridges are extensible, which allow the user the possibility of extending the functionality of their applications by adding his own personalized services. The developers can access to OpenShift using a command line interface (CLI), a web console or an API RESTful. OpenShift also offers middleware systems for applications based on standards like JBoss, Tomcat and Apache.

#### 3.4.1 OpenShift Physical Model

OpenShift is composed by 2 main elements:

- Broker: is the orchestrate mechanism for all the activities of the application management platform. The broken is charged with the following tasks:
  - Manage the login of the users.
  - To update the DNS dynamically.
  - Provide information about the status of the application.
  - Orchestrate the application.
  - Services and operations.

OpenShift is composed by different components which help on performing the task described before, MongoDB and ActiveMQ are worth mentioning. MongoDB keeps information about the status of the applications for the environment. ActiveMQ is the message broker that provides communication between the broker and the nodes. The messages are sent between the broker and the nodes using MCollective. Each broker executes one client and each node hosts a server that sends status messages and listen to incoming messages.

- Nodes: a node in OpenShift is a host that executes applications. There can be many nodes managed by a same or different brokers. A node contains gears that contain applications. One of the features of OpenShift is the separation in multitenant environments, which provides resources and data separated in every application.

### 3.4. OpenShift

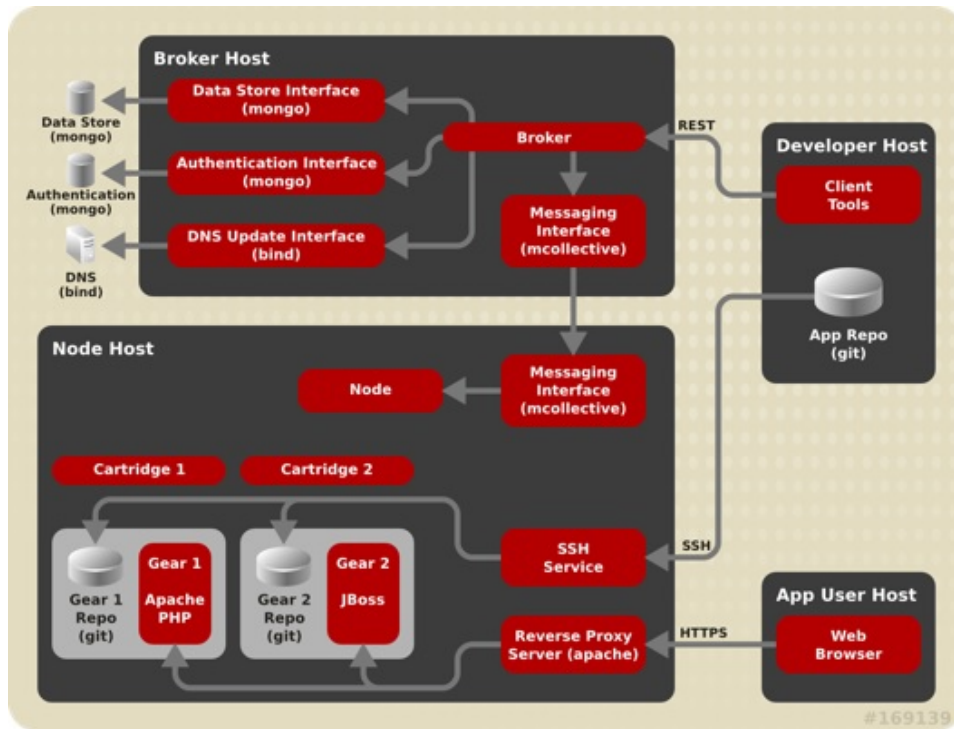


Figure 3.5: OpenShift Architecture. Taken from [32]

Each node is defined according to a series of gears, which represent the portions of CPU, memory RAM and storage space available for a single application. An application can't override these resources, except for the storage space, which quota is extendable for the administrator.

The technologies used to isolate the gears and manage the resource quotas are SELinux and cgroups. Additionally Docker can be used as application container.

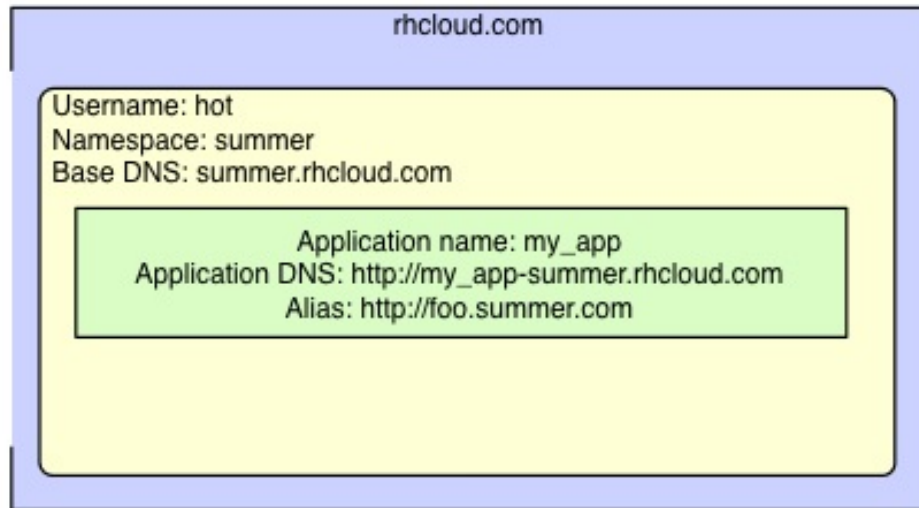


Figure 3.6: Information required to create an application. Taken from [33]

- **Domain:** The domain is not directly related to DNS; instead it provides a unique namespace for all the applications of a specific user. The domain name is appended to the application name to form the final application URL.
- **Application Name:** Identifies the name of the application. The final URL to access the application is of the form: `https://[APPNAME]-[DOMAIN].rhcloud.com`
- **Aliases:** Users can provide their own DNS names for the application by registering an alias with the platform.
- **Dependencies:** Users specify the cartridges required to run their applications.
- **git repository:** Each application gets a git repository. Users can modify code in the repository and then perform a git push to deploy their code.

The next flow describes the case of creating and deploying a simple PHP application.



### 3.4. OpenShift

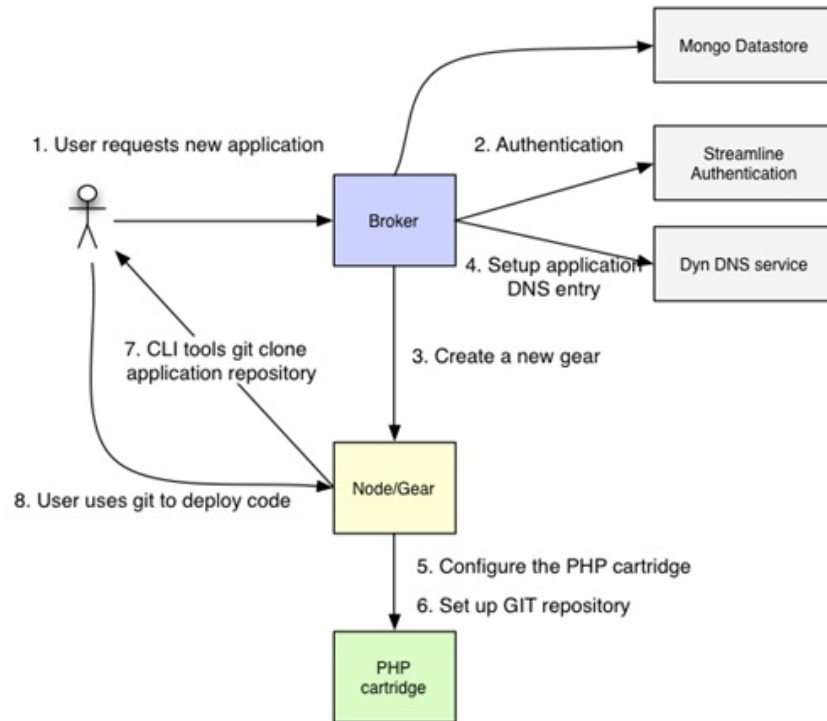


Figure 3.7: Flow describing the creation and deployment of a PHP application. Taken from [33]

## 4. Application to the LifeWatch project: Galaxy & TRUFA

### 4.1 What is LifeWatch?

LifeWatch is an European initiative that was included in the Roadmap of the European Strategy Forum on Research Infrastructures (ESFRI), the body that identifies the new research infrastructures (RIs) of pan-European interest with the goal of promoting the long-term competitiveness of European Research and Innovation.

*“The mission of LifeWatch is to advance biodiversity research and provide knowledge-based solutions to environmental managers for its preservation. This mission is achieved by providing access through a single infrastructure to a set of data, services and tools enabling the construction and operation of Virtual Research Environments (VREs) linked to LifeWatch, and where specific issues related with biodiversity research and preservation are addressed.” [34]*

### 4.2 Galaxy

#### 4.2.1 Implementation

The Galaxy Framework [35] is a set of reusable software components that can be used to:

- Integrate tools into applications.
- Encapsulate functionality for describing generic interfaces to computational tools.
- Build concrete interfaces for users to interact with tools.
- Invoke tools in various execution environments.
- Dealing with general and tool specific dataset formats and conversions.

- Work with metadata describing datasets, tools, and their relationships.

The GALAXY Application is an application built using this framework that provides access to tools through an interface (such as, a web-based interface). A GALAXY Instance is a deployment of this application with a specific set of tools.

The core components of the GALAXY Framework are the *toolbox*, the *job manager*, the *model*, and the *web interface*.

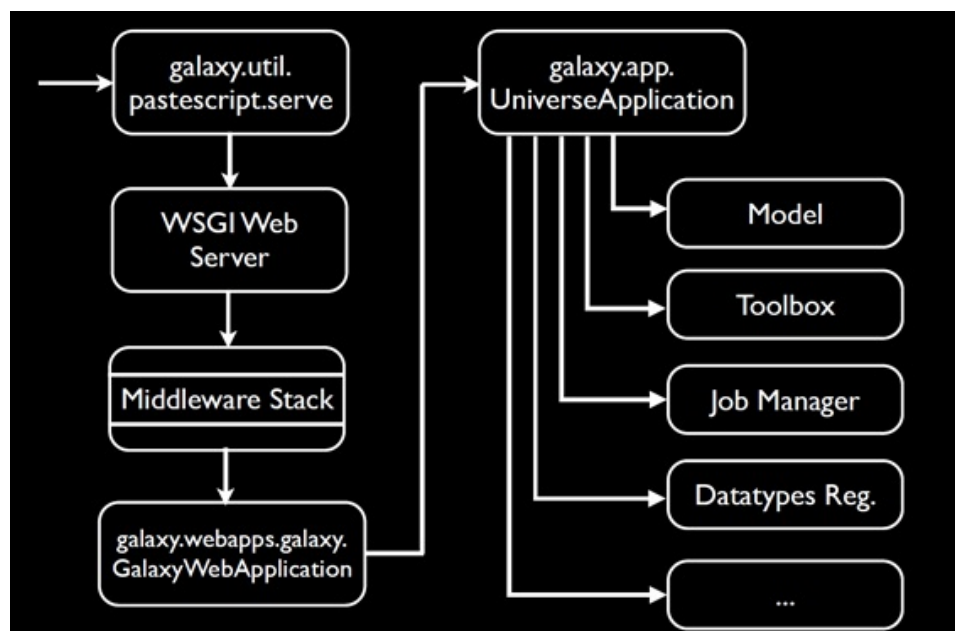


Figure 4.1: Galaxy application architecture. Taken from [36]

### The toolbox

The toolbox manages all of the details of working with command-line and web-based computational tools. It parses GALAXY tool configuration files, including the parameters and input data it can take, their types and restrictions, to a tool and the outputs it produces, in an abstract way that is not specific to any particular user interface. This abstraction is critically important since it allows for changing how tools are displayed without needing to change their configuration (for example, to leverage new accessibility features as web browsers improve, or to provide interfaces that are not web-based).

The toolbox provides support for validating inputs to a tool, and for transforming a valid set of inputs into the commands necessary to invoke that tool. Additionally, the toolbox allows tool authors to provide tests for their tools (inputs and corresponding outputs) and provides support for running those tests in the context of a particular GALAXY instance.

### **The job manager**

The job manager is in charge of dealing with the details of executing tools. It manages dependencies between jobs to ensure that required datasets have been produced without errors before a job is run. It provides support for job queuing, to allow multiple users to each submit multiple jobs to a GALAXY instance and receive a fair execution order. Currently jobs can be executed on the same machine where the GALAXY instance is running, or dispatched to a computational cluster using a standard queue manager.

### **The model**

GALAXY Model provides an abstract interface for working with datasets. It provides an object-oriented interface for working with dataset content; stored as files on disk and metadata; data about datasets, tools, and their relationships, stored in a relational database. Beyond providing access to the data, this component deals with support for different datatypes, datatype specific metadata, and type conversions.

### **The web interface**

The web interface provides support for interacting with a GALAXY instance through a web browser. It generates web-based interfaces to the toolbox, for browsing and choosing tools, individual tools by building forms to accept and validate user input to a tool, and the model, allowing the user to work with all of the datasets they have produced.

The web interface is currently the primary way to interact with a GALAXY instance, but the other underlying components do not need to know anything about the web, all web specific aspects of GALAXY are encapsulated by the web interface.

### **Implementation Details**

The GALAXY framework is implemented in Python. Python has several advantages that justify its choice. First, it is a lightweight dynamic language that allows to rapidly implement new GALAXY features. However, while Python is concise

and easy to write, it is also a highly structured language that is generally easy to read and understand. This is important since it makes customizing and extending the GALAXY framework much easier for users. Additionally, Python has a very powerful standard library, as well as an amazing variety of third party open source components.

However, an important aspect of the GALAXY architecture is the abstraction between the framework and the underlying tools. Because the GALAXY toolbox interacts with tools through command-line and web-based interfaces, there is no requirement that a tool author must use Python. While Python is a powerful language for scientific computing, and many of the tools we provide for comparative genomic analysis are implemented in Python, frequently another language may suit a particular problem better, or simply be preferred by a tool author.

GALAXY includes its own web server and embedded relational database, using SQLite, and a GALAXY download includes all dependencies: a user needs to just edit the configuration file and run one command to start interacting with and customizing their own GALAXY instance.

However, if a user's GALAXY instance needs to support higher throughput, they can customize the web server, the underlying relational database, and the job execution mechanism.

For instance, the public GALAXY instance maintained by the GALAXY team at Penn State (<https://usegalaxy.org>) is integrated with Apache as the web-server, uses the enterprise class relational database PostgreSQL, and executes jobs on a computational cluster with a queue managed by the Slurm scheduler.

## 4.3 TRUFA

### 4.3.1 Description

TRUFA [20] is an informatics platform based on a web interface that generates the outputs commonly used in *de novo* RNA-seq analysis and comparative transcriptomics. TRUFA relies on a pipeline to orchestrate the jobs. TRUFA gives the user an easy, fast and valid analysis on RNA-seq data.

The first step of a *de novo* RNA-seq analysis consists in assessing data quality and cleaning raw reads. The output of a next-generation sequencing (NGS) reaction contains traces of polymerase chain reaction (PCR) primers and sequencing adapters as well as poor-quality bases/reads. Hence, it is advised to perform read trimming, which has been shown to have a positive effect on the rest of the RNA-seq analysis, although parameter values for such trimming have to be optimized.

Once reads have been cleaned, they are assembled into transcripts, which are subsequently categorized into functional classes in order to understand their bio-

logical meaning. Finally, it is possible to perform expression quantification analyses by estimating the amount of reads sequenced per assembled transcript and taking into account that the number of reads sequenced theoretically correlates with the number of copies of the corresponding mRNA in vivo. All the above-mentioned steps in the RNA-seq analysis pipeline are included in TRUFA and correspond to distinct sections in the web-based user interface (see Figures 4.3.2 and 4.3.2).

The platform is mainly written in Javascript, Python, and Bash. The source code is available at Github (<https://github.com/TRUFA-rnaseq>). The long-term availability of the TRUFA web server (and further developed versions) is ensured given that it is currently installed in the ALTAMIRA supercomputer, a facility integrated in the Spanish Supercomputing Network (RES).

#### 4.3.2 Implementation

The overall workflow of TRUFA is shown in Figure 4.3.2. The input, output, and different components of the pipeline are the following:

##### Input

Currently, the input data accepted by TRUFA includes Illumina read files and/or reads already assembled into contigs. Read files should be in FASTQ format and can be uploaded as gzip compressed files (reducing uploading times). Reads from the NCBI SRA databases can be used but should be first formatted into FASTQ format using, e. g., the SRA toolkit. Already assembled contigs should be uploaded as FASTA files. Other FASTA files and HMM profiles can be uploaded as well for custom blast-like and protein profile-based transcript annotation steps, respectively. Thus far, no data size limitation is set.

##### Pipeline

Several programs can be called during the cleaning step (see Figure 4.3.2 ). The program FASTQC (<http://www.bioinformatics.bbsrc.ac.uk/projects/fastqc>) has been implemented to assess the quality of raw reads and give the statistics necessary to tune cleaning parameters (Fig. 4.3.2). After the quality of the data is determined, CUTADAPT [37] and PRINSEQ [38] allow, among other functionalities, the removal of adapters as well as low quality bases/reads. In particular, PRINSEQ has been chosen for its ability to treat both single and paired-end reads and to perform read quality trimming as well as duplicate removal. Using the BLAT fast similarity search tool, reads can be compared against databases of potential contaminants such as, eg, UniVec (which allows identifying sequences of

### 4.3. TRUFA

---

vector origin; (<http://www.ncbi.nlm.nih.gov/VecScreen/UniVec.html>) or user-specified databases. TRUFA's scripts will automatically remove those reads, giving hits with such queried databases.

Cleaned reads, after passing an optional second quality control with FASTQC to verify the overall efficiency of the first cleaning step, are ready for assembly. TRUFA implements the software Trinity [39] which is an extensively used de novo assembler and has been shown to perform better than other single k-mer assemblers [40]. After the assembly, an in-house script provides basic statistics describing transcripts lengths distribution, total bases incorporated in the assembly, N50, and GC content. In addition, to evaluate the complete-ness of the assembly, a Blast+ [41] similarity search is performed against the UniProtKB/Swiss-Prot database, and a Trinity script evaluates whether those assembled transcripts with hits are full-length or nearly full-length. Both the number of recovered genes from the total of 248 and their completeness have been used for de novo assembly quality assessments.

The newly assembled transcripts can be used as query for similarity searches with BLAT [42] and Blast+ against gene databases such as NCBI nr and UniRef90 HMMER [43] searches applying hidden Markov models (HMMs) against the PFAM-A database; other databases with user-specified models can also be used. Further annotation and assignation of gene ontology (GO) terms can be obtained with BLAST2GO28 for the transcripts with blast hits against the nr database.

For expression quantification, Bowtie2 [44] is used to produce alignments of the reads against the assembled transcripts. Alignments are then properly formatted using SAMtools43 and Picard (<http://picard.sourceforge.net>)[45]. Using these alignments, eXpress [46] can be used to quantify the expression of all isoforms. Additionally, the script "run\_RSEM\_align\_n\_estimate" of the Trinity package implemented in TRUFA uses Bowtie [47] and RSEM [48] to provide an alternative procedure for expression quantification of both genes and isoforms. Moreover, the percentage of reads mapping back to the assembled transcripts (obtained with Bowtie and Bowtie2) can be used as another indication of the assembly quality.

### 4.3. TRUFA

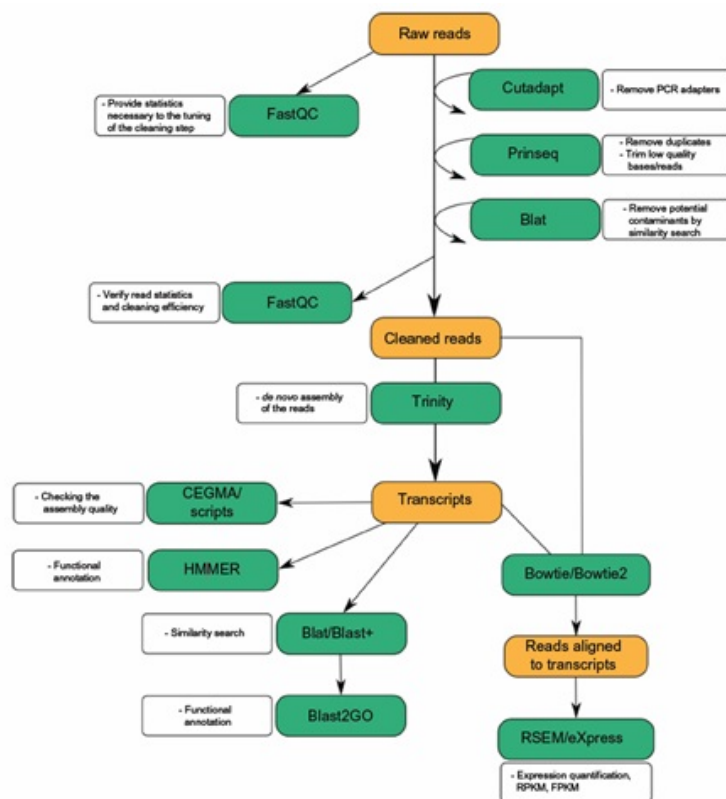


Figure 4.2: Overview of the TRUFA pipeline. Taken from [20]

### Output

TRUFA generates a large amount of output information from the different programs used in the customized pipeline. Briefly, a user should be able to download FastQC html reports, FASTQ files with cleaned reads (without duplicated reads and/or trimmed), Trinity-assembled transcripts (FASTA), read alignments against the transcripts (BAM files), GO annotations (.txt and.dat files which can be imported into the Blast2GO java application), and read counts (text files providing read counts and TPM). Various statistics are computed at each step and are reported in text files, such as the percentage of duplicated/trimmed reads, CEGMA completeness report, assembly sequence composition, percentage of mapped reads, and read count distributions.



#### 4.3. TRUFA

RNA-SEQ STEPS	AVAILABLE PROGRAMS	VERSIONS
Read cleaning	Prinseq	0.20.3
	Cutadapt	1.3
	BLAT	v.35
Assembly and mapping	Trinity	r2012-06-08
	CD-HIT	4.5.4
	CEGMA	2.4
	Bowtie	0.12.8
	Bowtie2	2.0.2
Annotation	BLAT	v.35
	HMMER	3.0
	Blast+	2.2.28
	Blast2GO	2.5.0
Expression quantification	RSEM	1.2.8
	eXpress	1.5.1

Figure 4.3: List of software available at TRUFA. Taken from [20].

## 4.3. TRUFA

TRUFA 0.12.0

HomeStart a JobFile ManagerHow toFAQAboutBug

adminLogout

Type of input:

Depending on the input you will specify you can perform various steps:

- with reads only, you can produce an assembly, then identify the contigs and quantify them
- with an assembly, you can go directly to the identification steps
- with both assembly and reads you can directly identify the transcripts and quantify them.

☒ Single reads (1 fastq file)

☐ Paired end reads (2 fastq files)

☐ Already assembled contigs (1 fasta file)

☐ Already assembled contigs and single reads (1 fastq file and 1 fasta file)

☐ Already assembled contigs and paired reads (2 fastq file and 1 fasta files)

Single reads file:

RNA-seq steps:

You can perform RNA-seq steps independently or sequentially depending on the boxes you check in each step tabs:

1. Cleaning step:

Pre-cleaning quality control:

☐ FastQC

Removing adapters:

☐ Cutadapt

Prinseq:

☐ Duplicated reads

☐ Quality Trimming

BLAT against potential contaminants:

☐ Univec hits

☐ E. coli hits

☐ S. cerevisiae hits

Nucleotide db

Post-cleaning quality control:

☐ FastQC

Options:

Duplication options

Trimming options

Cutadapt options

3. Identification step:

Blat searches:

☐ Uniref

☐ nr

Add custom nucleotides or protein sequences databases for the blat search (uploaded in fasta format):

Nucleotides db

Proteins db

HMMER searches:

☐ PfamA

Add custom profiles for the hmmer search:

Databases available for HMMER

Blast2GO searches:

☐ Blast+ against nr

☐ Blast2GO

2. Assembly and Mapping step:

☒ Assemble with Trinity

☐ Cluster similar sequences with CD-HIT-EST

☐ Assembly quality checks

☐ Align reads against contigs with Bowtie2

Options:

Trinity options

CD-HIT-EST options

Bowtie2 options

4. Expression quantification step:

☐ eXpress

☐ RSEM

Launching the analysis

Start

Figure 4.4: TRUFA web interface for user input. Taken from [20].

## 5. Implementation of the solutions and comparative Analysis

### 5.1 Problem Description

TRUFA is currently installed in the ALTAMIRA supercomputer at IFCA, it offers a good performance and short wait times with a few users. Although, the numbers of users who will rely on TRUFA to perform their RNA-seq jobs is expected to increase, in order to be prepared for this event, the deployment of TRUFA in a cloud environment could be very advantageous. Some of the benefits of TRUFA Cloud could be overall better user experience, better exploitation of the computational resources, scalability and elasticity, an increment in efficiency and extensibility.

### 5.2 Solution proposed

Some of the advantages of migrating TRUFA to a cloud environment would be:

**First** of all, the user experience would be better, because of shorter queue times to deploy their jobs and fastest execution of these.

**Secondly**, if TRUFA were deployed on a cloud environment, there could be a much better usage of the resources which are designated in a static way to run TRUFA at the ALTAMIRA supercomputer.

**Thirdly**, another reason to consider in a cloud environment is scalability, which can be defined as the ability of a system to handle a growing amount of work in a capable manner. TRUFA would be scalable if it was deployed in a cloud environment as new resources could be added on the fly to handle the increasing workload and relocate those free resources when they are not required.

## 5.2. Solution proposed

---

**Fourthly** , another advantage would be the extensibility, as we saw in the section 3.3, one of the achievements of the NDIGO-Data Cloud will be the possibility of running EGI Federated cloud solutions in heterogeneous infrastructures, using different cloud providers and resources providers.

**Lastly**, there would be a huge increment in efficiency. In its current situation, there are a lot of computer resources allocated to run TRUFA, that are constantly powered on and locked so they can't be used to anything else. This increment in efficiency will be obtained deploying TRUFA in a cloud environment, in that case, it will use only the exact machines required for the jobs received by the users, and those machines will be shut down upon job completion and could be relocated to perform any other task.

One of the main reasons to choose OpenShift as the Platform as a Service (PaaS) used in the Cloud infrastructure is its great ability to scale. OpenShift allows a single application to use up to 16 gears, depending on traffic demand. It is able to monitor the incoming web traffic to the application and automatically adds or removes copies of the application to serve requests as needed. Another great feature is its ability to create cartridges with the different software needed to run jobs as part of the TRUFA pipeline. This “containers” are called cartridges, which are the way Openshift handles the deployment of applications.

Therefore, our choice of PaaS would be OpenShift, as it provides a wide range of languages and services availables, which are deployed in applications through a cartridge. Cartridges can contain several different components, ranging from web frameworks, databases, monitoring services up to connectors to external backends.

The initial idea will be to create a cartridge for each single software required by TRUFA. This will allow us to configure the instances needed to perform the jobs based in the user input. Besides, another instance will be needed to act as a server, and will be in charge of handling the Graphic User Interface (GUI) using a web interface as well as the management of the users.

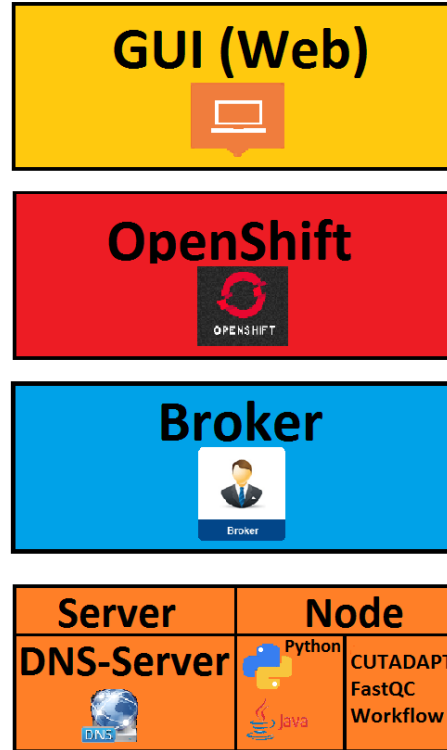


Figure 5.1: Overall representation of the infrastructure designed for TRUFA Cloud.

OpenShift is the PaaS used for the Cloud infrastructure. On top of it there is the Graphic User Interface that will allow the user through a very attractive way to interact with the system via web. On the lower layers are the cloud hosted machines, one hosting the server and other hosting nodes, that will have the necessary dependencies installed (e.g. Python, JRE, Perl) and will contain the cartridges with the methods of TRUFA (e.g. CUTADAPT, FastQC) and the workflow script, the one in charge of orchestrating the whole workflow)

We have focused on the work related to the OpenShift implementation as it is the main concern for this project.

OpenShift has been deployed using virtual machines available at IFCA. These machines were the same as the ones offered by the European Grid Infrastructure (EGI) and they used OpenStack [49] as cloud computing software platform, so using them was the closest realistic approach to implement the project.

## 5.2. Solution proposed

We have installed the free and open version of OpenShift, called “OpenShift Origin”. The operative systems supported by it are both Red Hat Enterprise Linux (RHEL) and CentOS. We have used CentOS 6.6 as it is a well-supported OS, free and was also available at EGI machines.

### 5.2.1 Setting up the virtual machines

We have followed the following steps to create the virtual machines:

First of all the machines in order to access to the virtual machines used ssh and the following .pem – (Privacy-enhanced Electronic Mail) Base64 encoded DER certificate keys. trufa.pem and nodo.pem, which we generated using OpenStack [49].



Nombre de par de claves	Fingerprint
nodo	ea:71:7c:81:b5:a1:45:c7:04:1d:21:7e:70:4d:f3:94
trufa	27:2a:cf:fb:e5:ed:06:53:10:5e:78:f2:b6:f8:7d:4f

Figure 5.2: Keys used to acces the virtual machines via ssh.

After that, we specified the flavor of the machine, and finally we assigned a public IP address to the machine. Then to access to the virtual machine we used the following command:

```
1 ssh -i key.pem centos@PUBLIC_IP_ADDRESS
```

The specifications of the main virtual machine (VM), which acted as the main server can be found on the following Figure 5.3. There were also other MVs which were in charge of hosting nodes that allowed us to perform different tasks:

## 5.2. Solution proposed

---

Specifications	IP Addresses
<b>Flavor</b>	<b>Private</b>
m1.large	172.16.7.13, 193.146.75.149
<b>RAM</b>	
6GB	
<b>VCPU</b>	
4 VCPU	
<b>Disk</b>	
20GB	
<b>Ephemeral Disk</b>	
80GB	

Figure 5.3: Specifications of the TRUFA Server Host.

In order to be able to deploy OpenShift Origin, a wide knowledge of the different components must be acquired, so that a technical remind of them are listed below (see section 3.4.1 for more details).

The installation has been done following a community provided comprehensive guide [50].

### Broker

The Broker role consists of the OpenShift Broker RPMs and a MCollective client. The Broker serves as a central hub of the OpenShift deployment, and provides a web interface where users can manage their hosted applications.

### DBServer

This role consists of the MongoDB database that the Broker uses to track users and applications.

### MsgServer

The MsgServer role includes the ActiveMQ server plus an MCollective client.

### Node

The Node role is assigned to any host that will actually be used to store and serve OpenShift-hosted applications.

## 5.2. Solution proposed

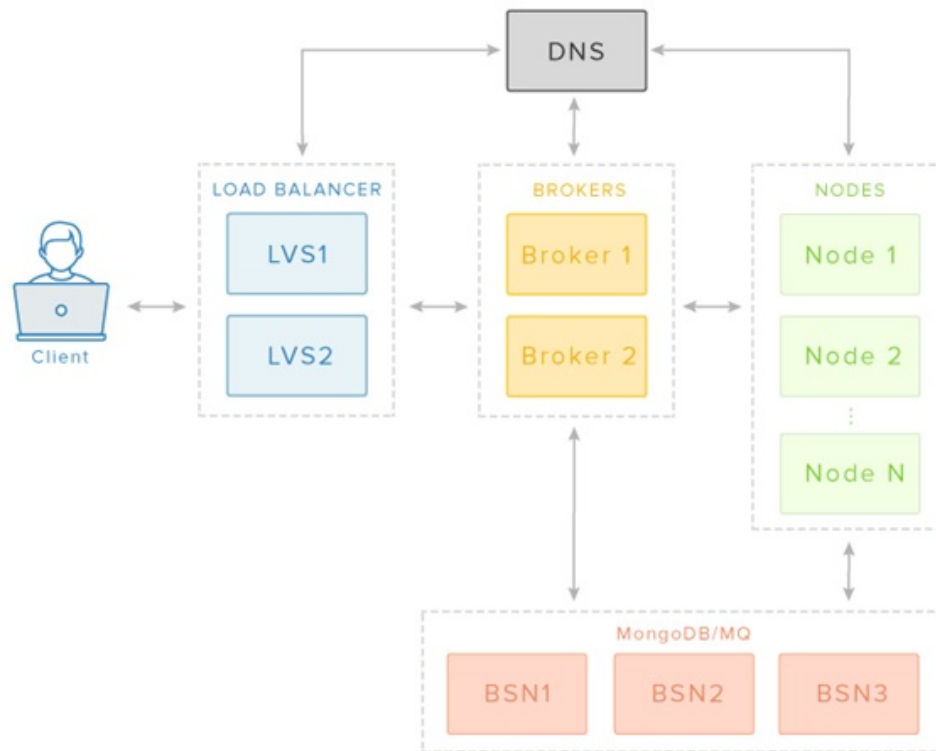


Figure 5.4: Client perspective of OpenShift. Image taken from [50].

### 5.2.2 Installing OpenShift in the server

The steps followed to get OpenShift up and running in our CentOS instance has been the following:

```
1 -Yum update
2 -Yum install ruby
3 -Change hostname on /etc/sysconfig/network
4 -Redirect to the new hostname on /etc/hosts
5 -Sh <(curl s https://install.openshift.com)
6 -Install OpenShift Origin
7 -Install a new DNS server for openshift-hosted
   applications named apps.example.com
```



## 5.2. Solution proposed

- 8 -Register DNS entries **for** openshift hosts with the OpenShift DNS service named openshift.example.com
- 9 -Establish the FQDN of the hostname : master.  
openshift.example.com
- 10 -Establish the IP address **for** SSH access , localhost
- 11 -Confirm the IP address of the detected network interface .
- 12 -Bind the DNS server installed on the host with the IP address
- 13 -Set the broker role
- 14 -Set the msgserver role
- 15 -Set the dbserver role
- 16 -Set the node role
- 17 -Confirm the installation of the packages BIND and PUPPET.

This is the configuration that the deployment generates:

DNS Settings		MCollective User   mcollective	
* Installer will deploy DNS		MCollective Password   58q0yqu5km5XT4	
* Application Domain: apps.example.com		MongoDB Admin User   admin	
* Register OpenShift hosts with DNS? Yes		MongoDB Admin Password   sXIsjD2DMzi5g0	
* Component Domain: openshift.example.com		MongoDB Broker User   openshift	
		MongoDB Broker Password   PmnhPcl55EEITT	
Global Gear Settings		Node Districts	
Valid Gear Sizes	small	District   Gear Size   Nodes	
User Default Gear Sizes	small	Default   small	master.openshift.example.com
Default Gear Size	small		
Account Settings		Role Assignments	
OpenShift Console User	demo	Broker	master.openshift.example.com
OpenShift Console Password	Q0	NameServer	master.openshift.example.com
Broker Session Secret	oZ	MsgServer	master.openshift.example.com
Console Session Secret	tS	DBServer	master.openshift.example.com
Broker Auth Salt	TS	Node	master.openshift.example.com
		Host Information	
		Hostname	Roles
		master.openshift.example.com	Broker DBServer MsgServer NameServer Node

Figure 5.5: OpenShift configuration on the server.

## 5.2. Solution proposed

### 5.2.3 Installing a BIND DNS Server

After installing OpenShift in the server node, we needed a DNS Server in order to enable the communication between the nodes and the server. The detailed process of installing the BIND DNS server is shown on Appendix 2.

After setting up the DNS Bind service, we created the node and assigned it to a district.

```
1 # oo-admin-ctl-district -c add-node -n  
    small_district -i node.example.com
```

Then we added the node to the DNS Bind by executing the following command:

```
1 # oo-register-dns -h node -d apps.example.com -n  
    172.16.5.9 -k ${keyfile}
```

This is the overview of the node through the web interface, and the details of the python application that was created:



Figure 5.6: Overview of the node

## 5.2. Solution proposed

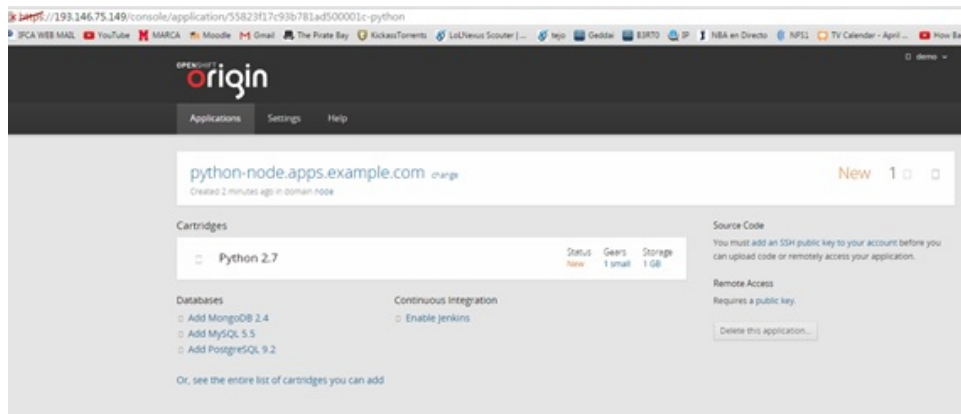


Figure 5.7: Details of the python application created in the node.

In order to verify that we have successfully added our node to our DNS server we executed a ping from the server to the node and vice versa.

```
[centos@master ~]$ ping node.apps.example.com
PING node.apps.example.com (172.16.7.9) 56(84) bytes of data.
64 bytes from 172.16.7.9: icmp_seq=1 ttl=64 time=0.528 ms
64 bytes from 172.16.7.9: icmp_seq=2 ttl=64 time=0.561 ms
64 bytes from 172.16.7.9: icmp_seq=3 ttl=64 time=0.431 ms
64 bytes from 172.16.7.9: icmp_seq=4 ttl=64 time=0.570 ms
♥
--- node.apps.example.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3458ms
rtt min/avg/max/mdev = 0.431/0.522/0.570/0.059 ms
```

Figure 5.8: Pinging the node from the server.

## 5.2. Solution proposed

```
[Centos@node ~]$ ping master.openshift.example.com
PING master.openshift.example.com (172.16.7.13) 56(84) bytes of data.
64 bytes from master.openshift.example.com (172.16.7.13): icmp_seq=1 ttl=64 time
=0.567 ms
64 bytes from master.openshift.example.com (172.16.7.13): icmp_seq=2 ttl=64 time
=0.530 ms
64 bytes from master.openshift.example.com (172.16.7.13): icmp_seq=3 ttl=64 time
=0.534 ms
64 bytes from master.openshift.example.com (172.16.7.13): icmp_seq=4 ttl=64 time
=1.68 ms
--- master.openshift.example.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3035ms
rtt min/avg/max/mdev = 0.530/0.827/1.680/0.493 ms
```

Figure 5.9: Pinging the server from the node.

This is the design for deployment that has taken place in the OpenShift layer:

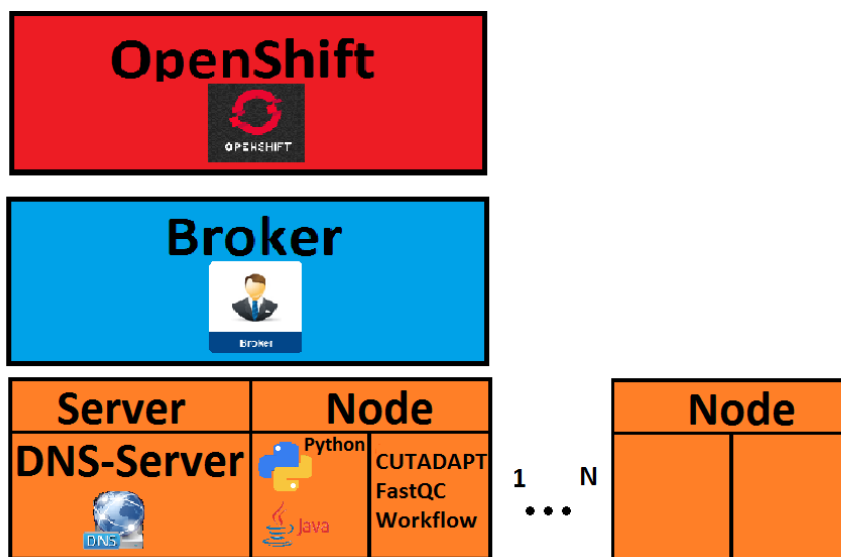


Figure 5.10: Detailed OpenShift Layer Components

The server is the main actor that contains the OpenShift instance which manages the different OpenShift nodes. In the nodes the different cartridges containing the operations ran by the RNA-seq have been installed. The life cycle of these nodes will be dynamically modified depending on the status of the user's job by the broker.

### 5.2.4 Creating the first cartridge (CUTADAPT)

The next step has been the creation of a first cartridge, which contains one of the many operations of the RNA-seq analysis offered by TRUFA. Due to simplicity, as it has a few dependencies and relatively small complexity, we have chosen to implement CUTADAPT.

CUTADAPT removes adapter sequences from high-throughput sequencing data. It can run using only Python, making it a light weight option so that it can fits our resources. Therefore, we have created an OpenShift node with Python (specifically, 2.7.6) underneath. In this application we have implemented and deployed a cartridge with CUTADAPT.

In the Appendix 3 are the step by step instructions to create the CUTADAPT cartridge.

Now we proceeded to test CUTADAPT with some real data: reads\_left.fq [52] and reads\_right.fq [53].

```
1 $ cutadapt -a AACCGGTT reads_left.fq > output_left.fq
2 cutadapt version 1.3
3 Command line parameters: -a AACCGGTT reads_left.fq
4 Maximum error rate: 10.00%
5   No. of adapters: 1
6   Processed reads:      30575
7   Processed bases:      2323700 bp (2.3 Mbp)
8   Trimmed reads:        554 (1.8%)
9   Trimmed bases:        1801 bp (0.0 Mbp)
   (0.08% of total)
10  Too short reads:      0 (0.0% of processed
   reads)
11  Too long reads:       0 (0.0% of processed
   reads)
12    Total time:         0.72 s
13    Time per read:      0.023 ms
14
15 === Adapter 1 ===
16
17 Adapter 'AACCGGTT', length 8, was trimmed 554 times
18 .
```

## 5.2. Solution proposed

```
19 No. of allowed errors :
20 0-8 bp: 0
21
22 Overview of removed sequences
23 length    count    expect    max.err    error counts
24 3          437      477.7     0          437
25 4          104      119.4     0          104
26 5           9       29.9     0           9
27 6           3        7.5     0           3
28 11          1         0.5     0           1
```

So we confirmed that it worked as intended, it deployed and successfully installed the CUTADAPT 1.3 functionality.

After that we created the FastQC cartridge following the previous steps mentioned before. We also tested FastQC with some real data.

```
1 $ fastqc reads_right.fq
2 Started analysis of reads_right.fq
3 Approx 5% complete for reads_right.fq
4 ...
5 Approx 95% complete for reads_right.fq
6 Analysis complete for reads_right.fq
```

We confirmed that the FastQC cartridge also was functional.

### 5.2.5 Creating the workflow script

The next step we took was creating a script that allowed us to simulate the effect of a workflow execution environment. We wanted the script to automatize the creation of an application, the launch of the cartridges, the execution flow and after that deleting the created application. We created this script using shell as it was the most suitable to the tasks at hand.

This is the execution sequence:

1. Create an application on a node.
2. Launch FastQC cartridge.
3. Execute FastQC with the input of the user.

4. Launch CUTADAPT cartridge.
5. Execute CUTADAPT with the input of the user.
6. Execute FastQC with the output from the execution named on the step
7. Delete the application.

The mentioned script can be found at the Appendix 4.

### 5.2.6 Results

In this section we discussed the results obtained by executing the workflow script. We have executed the script for both `reads_left.qc` and `reads_right.qc` taking into account the time the execution has taken.

Appendix 5 contains the execution of the script.

The first execution took 11.35 s to execute and the second took 11.36 s, we can appreciate that both the jobs are light processes and not excessively resource consuming. So the use of the cartridges does not affect in a huge way in the time needed to execute the whole sequence.

We can state that the script works as intended and that both FastQC and CUTADAPT are properly installed and working as desired. We can also affirm that the complete implementation of TRUFA Cloud can be done by using cartridges and workflow scripts.

### 5.2.7 Future Work: Final vision for TRUFA Cloud

We have seen that OpenShift works as expected, using cartridges as an efficient way to deploy our operations. The work we explained before is merely proof of concept to what could be the creation of TRUFA Cloud. The final TRUFA deployment would be as follows:

## 5.2. Solution proposed

---

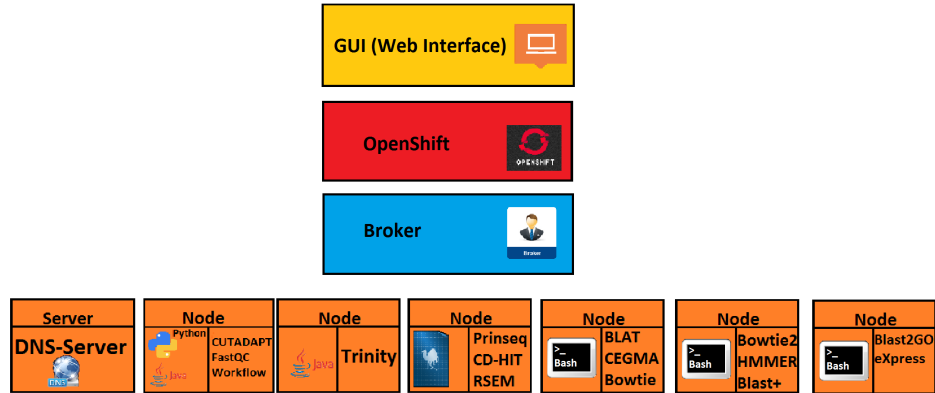


Figure 5.11: OpenShift Layer of a Complete Deployment of TRUFA Cloud.

The final deployment for TRUFA Cloud using the solution that we proposed, would be using different nodes hosting the dependencies (Python, Java, Linux) needed for each program (CUTADAPT, FastQC, Trinity, ..) , and the programs would be deployed into these nodes using cartridges.



## 6. Conclusions

There are a number of different applications and tools that provides non IT experts users with different features that allow them to perform very complex combined processing steps. We have dedicated a certain amount of work in analyzing the existing workflow solutions, which has allowed us to distinguish the good features that we want to implement, and to discern those characteristics we believe are superfluous and unnecessary. Some of them could only be applied to very specific fields, whereas others may be too complicated to modify or to add new features. Also, a few of them are not user-friendly.

We have also analyzed the existing Cloud and Platform as a Service solutions and we have chosen one option, having in mind the resources that we could access to. We have studied Galaxy and TRUFA thoroughly as examples of research applications implementing workflows, and we decided that TRUFA would be the best option to implement for this project, as it is currently running and supported at the Instituto de Fisica de Cantabria, has a high user demand (more than 60 users from ten different countries), is Python-based, which is a popular programming language with a huge community, easily used by OpenShift, it is designed in a modular way, which allows layer substitution without comprommising the general performance, and it is also open source.

A realistic and useful solution based on a PaaS open solution, OpenShift, using cartridges as the basic unit in the workflow implementation, combined with scripting, has been proposed and a first relevant test has been implemented and deployed on real infrastructure.

We have replaced the computational layer in the ALTAMIRA computer for OpenShift cartridges. The batch system role has been substituted by OpenShift and its broker, who acts as the orchestrator. In a final solution, both would manage the deployment of the cartridges and, attending to the user input, would handle the different resources required to deploy the whole infrastructure.

This solution takes advantage of all the benefits inherent to running on a Cloud environment, which can be summarized as a better user experience and an efficiency increase. Our solution can be used as a proof of concept to perform an

integral TRUFA migration to a cloud environment, which can be useful for different ongoing projects, such as INDIGO and the european initiative LifeWatch.

We have checked that the whole implementation of TRUFA Cloud is perfectly doable. That is why we have given such a detailed implementation of the key components to implement the solution. Besides, we have provided the whole design to the complete TRUFA Cloud implementation in order to make the whole implementation as best as possible.

This is only the first example of a large number of workflow applications that will be implemented along the next two years under the LifeWatch European initiative, and deployed in the EGI FedCloud.

This work has been a great challenge, where we have applied all the knowledge acquired during our studies. Is it worth to mention the competences obtained in the next courses : Operative Systems, Networks I and II and Informatic Systems. Also the infrastructure used in the Fed Cloud environment with IFCA resources is real and its designed to be used as real in future projects.

This project can be considered as the first realistic test of a Workflow as a Service solution, that will be useful in the future in many other application areas.

# Appendix 1: List of acronyms

WaaS - Workflow as a service  
PaaS - Platform as a Service  
IaaS-Infrastructure as a Service  
SaaS-Software as a Service  
EGI. European Grid Initiative  
TRUFA-TRanscriptome User-Friendly Analysis  
JRE- Java Runtime Environment  
RI-Research Infrastructure  
VRE-Virtual Research Environment  
GUI-Graphic User Interface  
IFCA-Instituto de Fisica de Cantabria  
UPV-Universitat Politecnica de Valencia  
INRA-Institut National de la Recherche Agronomique  
NGS-Next Generation Sequencing  
API-Application Programming Interface  
SSH-Secure Shell  
OS- Operative System  
VM- Virtual Machine

## Appendix 2: BIND DNS Server Installation

Here it is shown the detailed process to set up the DNS server, both in server and node hosts. We installed the bind-utils package and set the domain to our actual domain by

```
1 yum install -y bind bind-utils
2
3 domain=apps.example.com
```

DNSSEC, which stands for DNS Security Extensions, is a method by which DNS servers can verify that DNS data is coming from the correct place. We created a private/public key pair to determine the authenticity of the source domain name server. In order to implement DNSSEC on OpenShift, we needed to create a key file, which will be stored in /var/named. For convenience, we set the "\$keyfile" variable to the location of this key file:

```
1 keyfile=/var/named/${domain}.key
```

Then we created a DNSSEC key pair and stored the private key in a variable named "\$KEY" by using the following commands:

```
1 pushd /var/named
2 rm K${domain}*
3 dnssec-keygen -a HMAC-MD5 -b 512 -n USER -r /dev/
  urandom ${domain}
4 KEY="$(grep Key: K${domain}*.private | cut -d ' ' -f
  2)"
5 popd
```

We also created an rndc key, which will be used by the init script to query the status of BIND when you run service named status:

```
1 rndc-confgen -a -r /dev/urandom
```

We also configured the ownership, permissions, and SELinux contexts for the keys that we have created:

```
1 restorecon -v /etc/rndc.* /etc/named.*
2 chown -v root:named /etc/rndc.key
3 chmod -v 640 /etc/rndc.key
4 echo "forwarders _[ 8.8.8.8; 8.8.4.4; ];" >> /var/
   named/forwarders.conf
5 restorecon -v /var/named/forwarders.conf
6 chmod -v 640 /var/named/forwarders.conf
```

To ensure that we were using a clean /var/named/dynamic directory, we removed this directory:

```
1 rm -rvf /var/named/dynamic
2 mkdir -vp /var/named/dynamic
```

We used the following command to create the \${domain}.db file

```
1 cat <<EOF > /var/named/dynamic/${domain}.db
2 \${ORIGIN} .
3 \${TTL} 1 ; 1 seconds (for testing only)
4 ${domain} IN SOA ns1.${domain}. hostmaster.${
   {domain}}. (
5     2011112904 ; serial
6     60         ; refresh (1 minute)
7     15         ; retry (15 seconds)
8     1800       ; expire (30 minutes)
9     10         ; minimum (10 seconds)
10    )
11    NS ns1.${domain}.
12    MX 10 mail.${domain}.
13 \${ORIGIN} ${domain}.
```

```
14 ns1          A    127.0.0.1
15 EOF
16 We also needed to create the named.conf file.
17 cat <<EOF > /etc/named.conf
18 options {
19     listen-on port 53 { any; };
20     directory    "/var/named";
21     dump-file     "/var/named/data/cache_dump.db";
22     statistics-file "/var/named/data/
23         named_stats.txt";
24     memstatistics-file "/var/named/data/
25         named_mem_stats.txt";
26     allow-query   { any; };
27     recursion yes;
28     /* Path to ISC DLV key */
29     bindkeys-file "/etc/named.iscdlv.key";
30
31     // set forwarding to the next nearest server (
32     from DHCP response
33     forward only;
34     include "forwarders.conf";
35 };
36 logging {
37     channel default_debug {
38         file "data/named.run";
39         severity dynamic;
40     };
41 };
42 // use the default rndc key
43 include "/etc/rndc.key";
44
45 controls {
46     inet 127.0.0.1 port 953
47     allow { 127.0.0.1; } keys { "rndc-key"; };
48 };
49 include "/etc/named.rfc1912.zones";
50 include "${domain}.key";
51 zone "${domain}" IN {
```

```
49     type master;
50     file "dynamic/${domain}.db";
51     allow-update { key ${domain} ; } ;
52 };
53 EOF
```

Finally, we set the permissions for the new configuration file that we just created:

```
1 chown -v root:named /etc/named.conf restorecon /etc/
  named.conf
```

Now we were ready to start up our new DNS server and add some updates.

```
1 service named start
```

Now we needed to update the resolv.conf file to use the local named service that we installed and configured. We Added in /etc/resolv.conf file the following entry as the first nameserver entry in the file:

```
1 nameserver 127.0.0.1
```

We also need to make sure that named starts on boot and that the firewall is configured to pass through DNS traffic:

```
1 lokkit --service=dns
2 chkconfig named on
```

We also modified the /etc/dhcp/dhclient-eth0.conf with the following information:

```
1 prepend domain-name-servers 172.16.7.13;
2 supersede host-name master.openshift.example.com;
3 supersede domain name apps.example.com;
```

We also modified the /etc/resolv.conf file in both the server and later in the node.

In the server with the following configuration:

## Appendix 2: BIND DNS Server Installation

---

```
1 search apps.example.com
2 nameserver 127.0.0.1
3 nameserver 8.8.8.8
```

And in the node:

```
1 search apps.example.com
2 nameserver 172.16.7.9
3 nameserver 8.8.8.8
```

We also modified the `/etc/sysconfig/network` to set the BIND DNS Server as the gateway in the node:

```
1 GATEWAY=172.16.7.13
```

We also modified the `/etc/hosts` file in the node.

```
1 172.16.7.13
```

We added a record for our broker node to BIND's database:

```
1 # oo-register-dns -h broker -d example.com -n
    172.16.7.13 -k ${keyfile}
```



## Appendix 3: Creation of the CUTADAPT cartridge

In order to create an OpenShift cartridge there's a mandatory set of directories and files that should be present in the cartridge. More information can be consulted in the OpenShift Origin Cartridge Developer's Guide [51].

This is the actual CUTADAPT cartridge folder. Bin contains install and setup, two scripts that handle the status of the application and its installation. Hooks contains actions to be run during lifecycle changes. Metadata contains the manifest.yml, a descriptor of the cartridge. Versions contains a folder called cutadapt1.3 which contains the source code of the application compressed in a .tar.gz.



Figure 1: CUTADAPT Cartridge folder



Figure 2: Contents of the bin folder

Control:

```
1 #!/bin/bash
2 version=cutadapt
3 function status() {
4     client_result $pwd
5     ps -C cutadapt > /dev/null
6     if [ "$?" -eq "0" ];
7     then
```

```

8      client_result "cutadapt_is_running";
9      else
10     client_result "cutadapt_is_not_running";
11     fi
12 }
13 function start() {
14     client_result "starting_cutadapt...."
15     ps -C cutadapt > /dev/null
16     if [ "$?" -eq "0" ]; then
17         client_result "cutadapt_is_running_
18             already...run_stop_first";
19     else
20         client_result "cutadapt_is_
21             not_running...starting_
22             it_now"
23         #Execute cutadapt in background mode
24         #nohup cutadapt/bin/cutadapt -a
25             AACCGGTT input.fastq > output.fastq
26             &
27     fi
28 }
29 # Ensure arguments.
30 if ! [ $# -gt 0 ]; then
31     echo "Usage: _$0_[start|status]"
32     exit 1
33 fi
34 # Source utility functions.
35 source $OPENSIFT_CARTRIDGE_SDK_BASH
36 # Handle commands.
37 case "$1" in
38     start) start ;;
39     status) status ;;
40     *)
41
42 esac

```

### Appendix 3: Creation of the CUTADAPT cartridge

Install:

```
1 #!/bin/bash
2
3 source $OPENSIFT_CARTRIDGE_SDK_BASH
4 client_result "installing_cutadapt.... 'date '"
5 #Unzipping aqui
6 tar -xvzf versions/cutadapt-*/cutadapt-*
7 mv cutadapt-* cutadapt-install
8 #Compiling cutadapt
9 #Dependencies needed python-devel y libxslt-devel
10 python cutadapt-install/setup.py build > log.txt
11 client_result "install_complete.... 'date '"
```

The metadata folder contains the manifest.yml file, which acts as the descriptor of the cartridge.

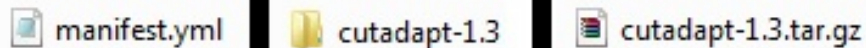


Figure 3: Content of the metadata, version and cutadapt-1.3 folders, respectively.

Manifest.yml:

```
1 Name: cutadapt
2 Cartridge-Short-Name: CUTADAPT
3 Display-Name: cutadapt-1.3
4 Description: CUTADAPT 1.3 (https://pypi.python.org/pypi/cutadapt/1.3).
5 Version: "1.3"
6 Cartridge-Version: 0.1.0
7 Cartridge-Vendor: lbertillo
8 Categories:
9   - service
10 Provides:
```

### Appendix 3: Creation of the CUTADAPT cartridge

```
11   - cutadapt -1.3
12   - cutadapt
13   Scaling :
14     Min: 1
15     Max: -1
16
17   Source-Url: https://github.com/lbertillo/cutadapt.git
```

The versions folder contains the cutadapt-1.3 version folder, where inside is the CUTADAPT source code. After creating these folders and files, the cartridge was ready to be tested. To test it, we created a Python 2.7 application and deployed the cartridge.

This is the interface for submitting customized cartridges to an application.

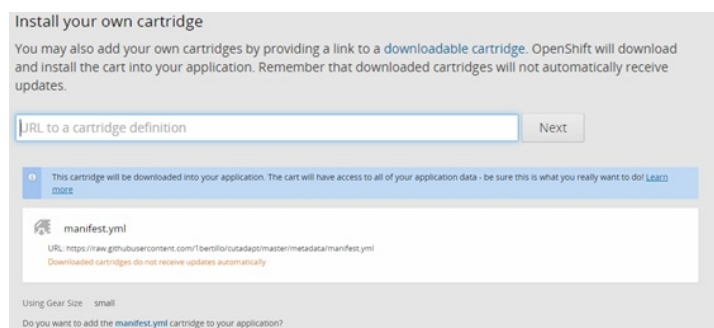


Figure 4: Web interface for uploading customized cartridges on OpenShift.

After it succeeded in the deployment:

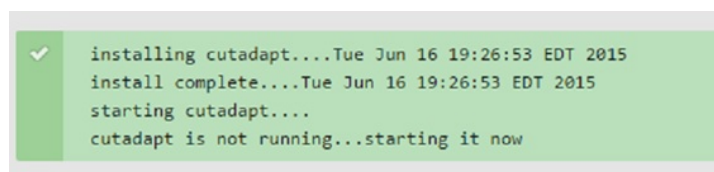


Figure 5: Output after the deployment of the CUTADAPT cartridge on a node.

### Appendix 3: Creation of the CUTADAPT cartridge

---

This is the cartridges that the application holds after the deployment:



Figure 6: List of cartridges after the installation of the CUTADAPT cartridge.

## Appendix 4: Workflow script

```
1  #!/bin/bash
2
3  function exefastqc() {
4      #Executing fastqc
5      echo "INITIATING_FASTQC:"
6      fastqc -o . $1
7      #Decompressing file , copying and removing
8          results
9      echo "PROCESSING_STATISTICS:"
10     #We send the output to dev/null because we
11     don't want it to show in the shell
12     unzip $1_fastqc.zip > /dev/null
13     #fastqc_data.txt is generated by the
14     execution of FastQC and it contains the
15     statistics of the analysis performed.
16     cp $1_fastqc/fastqc_data.txt .
17     rm -R $1_fastqc
18     rm $1_fastqc.html $1_fastqc.zip
19     #We show the result of the FastQC execution
20     by grepping the results
21     cat fastqc_data.txt | grep pass
22     cat fastqc_data.txt | grep fail
23     cat fastqc_data.txt | grep warn
24     echo "_____
```

```

23  #####Executing_cutadapt
24  #####echo_"INITIATING CUTADAPT:"
25  #####cutadapt_-a_$2_$1_>_$1.output.fq
26  #####echo_"#####"
27  }
28
29  #Main
30  if_![_#_-eq_1_]; then
31  #####echo_"Usage: $0 [input.fq]"
32  #####exit_1
33  fi
34  chain="AACCGGTT"
35  app-name="python-node.apps.example.com"
36  repocutadapt="https://raw.githubusercontent.com/1
    bertillo/cutadapt/master/metadata/manifest.yml"
37  repofastqc="https://raw.githubusercontent.com/1
    bertillo/fastqc/master/metadata/manifest.yml"
38
39  #Creating_an_application_with_python-2.7
40  rhc_app-create_$app-name_python-2.7
41  #Launching_FastQC_cartridge
42  rhc_cartridge-add_$repofastqc_-a_$app-name
43  exefastqc_$1
44
45  #Launching_CUTADAPT_cartridge
46  rhc_cartridge-add_$repocutadapt_-a_$app-name
47  executadapt_$1_$chain
48
49  #Executing_FastQC_with_input_as_the_output_of_the_
    cutadapt_execution
50  exefastqc_$1.output.fq
51
52  #Removing_the_application_after_the_execution
53  rhc_app-delete_$app-name

```

## Appendix 5: Execution of the script

```
1 $ ./script reads_left.fq
2 INITIATING FASTQC:
3 Started analysis of reads_left.fq
4 Approx 5% complete for reads_left.fq
5 ...
6 Approx 95% complete for reads_left.fq
7 Analysis complete for reads_left.fq
8 PROCESSING STATISTICS ESTADÍSTICAS:
9 >>Basic Statistics      pass
10 >>Per base sequence quality      pass
11 >>Per sequence quality scores    pass
12 >>Per base N content      pass
13 >>Sequence Length Distribution  pass
14 >>Adapter Content         pass
15 >>Per tile sequence quality      fail
16 >>Per base sequence content      fail
17 >>Per sequence GC content      warn
18 >>Sequence Duplication Levels   warn
19 >>Overrepresented sequences     warn
20 >>Kmer Content      warn
21 _____
22 INITIATING CUTADAPT:
23 cutadapt version 1.3
24 Command line parameters: -a AACCGGTT reads_left.fq
25 Maximum error rate: 10.00%
```



## Appendix 5: Execution of the script

```

26   No. of adapters: 1
27   Processed reads:      30575
28   Processed bases:      2323700 bp (2.3 Mbp)
29   Trimmed reads:        554 (1.8%)
30   Trimmed bases:        1801 bp (0.0 Mbp)
      (0.08% of total)
31   Too short reads:      0 (0.0% of processed
      reads)
32   Too long reads:       0 (0.0% of processed
      reads)
33   Total time:           0.73 s
34   Time per read:        0.024 ms
35
36   === Adapter 1 ===
37
38   Adapter 'AACCGGTT', length 8, was trimmed 554 times
39   .
40   No. of allowed errors:
41   0–8 bp: 0
42
43   Overview of removed sequences
44   length  count   expect  max.err  error counts
45   3         437     477.7    0        437
46   4         104     119.4    0        104
47   5          9       29.9     0         9
48   6          3        7.5     0         3
49   11         1        0.5     0         1
50
51   _____
52   INITIATING FASTQC:
53   Started analysis of reads_left.fq.output.fq
54   Approx 5% complete for reads_left.fq.output.fq
55   ...
56   Approx 95% complete for reads_left.fq.output.fq
57   Analysis complete for reads_left.fq.output.fq
58   PROCESSING STATISTICS:
59   >>Basic Statistics      pass

```

## Appendix 5: Execution of the script

---

```
60 >>Per base sequence quality      pass
61 >>Per sequence quality scores    pass
62 >>Per base N content             pass
63 >>Adapter Content                pass
64 >>Per tile sequence quality      fail
65 >>Per base sequence content      fail
66 >>Per sequence GC content        warn
67 >>Sequence Length Distribution    warn
68 >>Sequence Duplication Levels    warn
69 >>Overrepresented sequences      warn
70 >>Kmer Content                   warn
71 _____
72
73 real      0:11.35 s
74 user      13.13 s
75 sys       0.73 s
76 memory    158428KB
77
78
79 $ ./script reads_right.fq
80 INITIATING FASTQC:
81 Started analysis of reads_right.fq
82 Approx 5% complete for reads_right.fq
83 ...
84 Approx 95% complete for reads_right.fq
85 Analysis complete for reads_right.fq
86 PROCESSING STATISTICS:
87 >>Basic Statistics               pass
88 >>Per base sequence quality      pass
89 >>Per sequence quality scores    pass
90 >>Per base N content             pass
91 >>Sequence Length Distribution    pass
92 >>Adapter Content                pass
93 >>Kmer Content                   pass
94 >>Per tile sequence quality      fail
95 >>Per base sequence content      fail
96 >>Per sequence GC content        warn
97 >>Sequence Duplication Levels    warn
```

## Appendix 5: Execution of the script

```

98 >>Overrepresented sequences      warn
99
100 INITIATING CUTADAPT:
101 cutadapt version 1.3
102 Command line parameters: -a AACCGGTT reads_right.fq
103 Maximum error rate: 10.00%
104   No. of adapters: 1
105   Processed reads:      30575
106   Processed bases:      2323700 bp (2.3 Mbp)
107   Trimmed reads:        492 (1.6%)
108   Trimmed bases:        1660 bp (0.0 Mbp)
109                          (0.07% of total)
110   Too short reads:      0 (0.0% of processed
111                          reads)
112   Too long reads:       0 (0.0% of processed
113                          reads)
114   Total time:           0.74 s
115   Time per read:        0.024 ms
116
117 === Adapter 1 ===
118 Adapter 'AACCGGTT', length 8, was trimmed 492 times
119
120
121 No. of allowed errors:
122 0-8 bp: 0
123
124 Overview of removed sequences
125
126 length  count  expect  max.err  error counts
127 3        381    477.7   0        381
128 4        101    119.4   0        101
129 5         8     29.9    0         8
130 6         1     7.5     0         1
131 67        1     0.5     0         1
132
133
134 INITIATING FASTQC:
135 Started analysis of reads_right.fq.output.fq

```

```
132 Approx 5% complete for reads_right.fq.output.fq
133 ...
134 Approx 95% complete for reads_right.fq.output.fq
135 Analysis complete for reads_right.fq.output.fq
136 PROCESSING STATISTICS:
137 >>Basic Statistics          pass
138 >>Per base sequence quality    pass
139 >>Per sequence quality scores  pass
140 >>Per base N content          pass
141 >>Adapter Content            pass
142 >>Kmer Content               pass
143 >>Per tile sequence quality    fail
144 >>Per base sequence content    fail
145 >>Per sequence GC content      warn
146 >>Sequence Length Distribution warn
147 >>Sequence Duplication Levels  warn
148 >>Overrepresented sequences    warn
149 _____
150
151 real      0:11.36s
152 user      12.46s
153 sys       0.76s
154 memory    155116KB
```

## Appendix 6: Errors

We encountered an error when trying to deploy the CUTADAPT cartridge in the application, it said:

```
1 "/bin/sh: /var/lib/openshift/524d8719e0b8cd61ef0001d4/cutadapt/bin/setup: Permission denied"
```

It was because the scripts at bin didn't have executable permissions, I was trying to fix this by executing

```
1 chmod +x /bin/*
```

The problem was still there because I was doing it using Windows, there were 2 alternative solutions: Using a Linux system for the same chmod command or using instead the next command in Windows

```
1 git update-index --chmod=+x /bin/*
```

We also had problems by installing OpenShift on the server host. This is the information that showed during the installation. At the beginning we thought it was because of wrong DNS configuration, but it was because the MongoDB service couldn't be installed because of not enough free space in disk.

```
Attempting to register available cartridge types with Broker(s).
/opt/rh/ruby193/root/usr/share/gems/gems/moped-1.5.1/lib/moped/cluster.rb:225:in
`with_primary': Could not connect to a primary node for replica set <Moped::Clu
ster nodes=[<Moped::Node resolved_address="127.0.0.1:27017">]> (Moped::Errors::C
onnectionFailure)
    from /opt/rh/ruby193/root/usr/share/gems/gems/moped-1.5.1/lib/moped/clus
ter.rb:222:in `with_primary'
    from /opt/rh/ruby193/root/usr/share/gems/gems/moped-1.5.1/lib/moped/clus
ter.rb:222:in `with_primary'
    from /opt/rh/ruby193/root/usr/share/gems/gems/moped-1.5.1/lib/moped/clus
ter.rb:222:in `with_primary'
    from /opt/rh/ruby193/root/usr/share/gems/gems/moped-1.5.1/lib/moped/clus
ter.rb:222:in `with_primary'
    from /opt/rh/ruby193/root/usr/share/gems/gems/moped-1.5.1/lib/moped/clus
ter.rb:222:in `with_primary'
```

Figure 7: Error encountered during the OpenShift installation

After reviewing logs we realized it was a problem of hard drive bad configuration. It seemed like we were using a bad CentOS image that didn't attend to the flavor of the virtual machine where it was hosted. It had a partition of 4 gb even when the flavor assigned it 40gb.

```
[root@prueb cloud-user]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/xvda1      3.9G  980M  2.8G  27% /
tmpfs           1.9G   0    1.9G   0% /dev/shm
/dev/xvdb       40G   177M   38G   1% /mnt
```

Figure 8: Hard drive configuration on the server

The /dev/xvda1 filesystem had a pre-assigned size of 3,9GB so the only solution was using a clean image of CentOS.

# Bibliography

- [1] Bowers, Shawn et al. (2005). Actor Oriented Design for Scientific Workflows, Lecture in Computer Science (Vol. 3716, pp. 369-384).
- [2] Workflow Management Coalition: Terminology & Glossary. (1996). Workflow Management Coalition.
- [3] Goble, C., & De Roure, D et al. (2009). The impact of workflow tools on data-centric research. In *The fourth paradigm: Data-intensive scientific discovery* (pp. 137-145). Microsoft Research.
- [4] Curbera, Francisco et al. (2007). Workflow Composition for the Web (pp. 94-106). Service-Oriented Computing-ICSOC 2007.
- [5] Perera, S., & Gannon, D et al. . Enabling Web Service Extensions of Scientific Workflows. Computer Science department, Indiana University.
- [6] The Kepler Project. . Retrieved April 4, 2015, from <https://kepler-project.org/>
- [7] Taverna - open source and domain independent Workflow Management System. . Retrieved April 7, 2015, from <http://www.taverna.org.uk/>
- [8] The Galaxy Project: Online bioinformatics analysis for everyone. . Retrieved April 12, 2015, from <http://galaxyproject.org/>
- [9] Bertram, L et al. (2005). Scientific Workflow Management and the Kepler System. *Concurrency and Computation: Practice & Experience*.
- [10] Mandal, N. et al. . Integrating Existing Scientific Workflow Systems. USC Information Sciences, Institute Marina Del Rey,CA.
- [11] Goderis, A et al. (2007). Composing Different Models of Computation in Kepler and Ptolemy (pp. 182-190). *International Conference on Computation Science (ICCS)*.
- [12] Wolstencroft, K et al. (2013). The Taverna workflow suite: Designing and executing workflows of Web Services on the desktop,web or in the cloud.

- 72



## *Bibliography*

---

- [25] European Grid Infrastructure. . Retrieved May 17, 2015, from <http://www.egi.eu/>
- [26] Ferrari, T. 11 years of support to the European Research Area.
- [27] EGI SOLUTIONS - FEDERATED - CLOUD. EGI.EU.
- [28] EGI SOLUTIONS - HIGH-THROUGHPUT -DATA ANALYSIS. EGI.EU.
- [29] EGI SOLUTIONS COMMUNITY DRIVEN -INNOVATION & SUPPORT. EGI.EU.
- [30] Wallom,D. EGI FedCloud; Connecting Researchers to Clouds.
- [31] Donvito,G. (2015) INDIGO DataCloud.
- [32] OpenShift Develop, Host, and Scale Your Apps in the Cloud. Retrieved May 17, 2015, from <https://www.openshift.com/>
- [33] OpenShift Origin System Architecture Guide. . Retrieved May 17, 2015, from [http://docs.openshift.org/origin-m4/oo\\_system\\_architecture\\_guide.html](http://docs.openshift.org/origin-m4/oo_system_architecture_guide.html)
- [34] Scientific and Technical description of LifeWatch ESFRI. (2015). Lifewatch.
- [35] GALAXY Architecture. Retrieved May 10, 2015, from <https://wiki.galaxyproject.org/Develop/Architecture>
- [36] Taylor, J., Coraor, N. (2013). Galaxy Code and Storage Architecture. Galaxy Community Conference 2013.
- [37] Martin, M. (2011). Cutadapt removes adapter sequences from high-throughput sequencing reads. EMBnet. journal, 17(1), pp-10.
- [38] Schmieder, R., & Edwards, R. (2011). Quality control and preprocessing of metagenomic datasets. Bioinformatics, 27(6), 863-864.
- [39] Grabherr, M. G., Haas, B. J., Yassour, M., Levin, J. Z., Thompson, D. A., Amit, I., ... & Regev, A. (2011). Full-length transcriptome assembly from RNA-Seq data without a reference genome. Nature biotechnology, 29(7), 644-652.
- [40] Zhao, Q. Y., Wang, Y., Kong, Y. M., Luo, D., Li, X., & Hao, P. (2011). Optimizing de novo transcriptome assembly from short-read RNA-Seq data: a comparative study. BMC bioinformatics, 12(Suppl 14), S2.

- [41] Camacho, C., Coulouris, G., Avagyan, V., Ma, N., Papadopoulos, J., Bealer, K., & Madden, T. L. (2009). BLAST+: architecture and applications. *BMC bioinformatics*, 10(1), 421.
- [42] Kent, W. J. (2002). BLAT—the BLAST-like alignment tool. *Genome research*, 12(4), 656-664.
- [43] Finn, R. D., Clements, J., & Eddy, S. R. (2011). HMMER web server: interactive sequence similarity searching. *Nucleic acids research*, gkr367.
- [44] Langmead, B., & Salzberg, S. L. (2012). Fast gapped-read alignment with Bowtie 2. *Nature methods*, 9(4), 357-359.
- [45] Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., ... & Durbin, R. (2009). The sequence alignment/map format and SAMtools. *Bioinformatics*, 25(16), 2078-2079.
- [46] Roberts, A., & Pachter, L. (2013). Streaming fragment assignment for real-time analysis of sequencing experiments. *Nature methods*, 10(1), 71-73.
- [47] Langmead, B., Trapnell, C., Pop, M., & Salzberg, S. L. (2009). Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome biology*, 10(3), R25.
- [48] Li, B., & Dewey, C. N. (2011). RSEM: accurate transcript quantification from RNA-Seq data with or without a reference genome. *BMC bioinformatics*, 12(1), 323.
- [49] OpenStack. Retrieved May 17, 2015, from <https://www.openstack.org/>
- [50] How To Install and Configure OpenShift Origin on CentOS 6.5 | DigitalOcean. Retrieved May 17, 2015, from <https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-openshift-origin-on-centos-6-5>
- [51] OpenShift Origin Cartridge Developer's Guide. . Retrieved June 17, 2015, from [http://docs.openshift.org/origin-m4/oo\\_cartridge\\_developers\\_guide.html](http://docs.openshift.org/origin-m4/oo_cartridge_developers_guide.html)
- [52] reads\_left.fq .Retrieved June 17,2015 from [https://trufa.ifca.es/web/static/demo\\_files/reads\\_left.fq.tar.gz](https://trufa.ifca.es/web/static/demo_files/reads_left.fq.tar.gz)
- [53] reads\_right.fq .Retrieved June 17,2015 from [https://trufa.ifca.es/web/static/demo\\_files/reads\\_right.fq.tar.gz](https://trufa.ifca.es/web/static/demo_files/reads_right.fq.tar.gz)