Deliverable No. 5.1.2

Deployment models of the CHIC technical architecture and its private cloud

Grant Agreement No.:	600841
Deliverable No.:	D5.1.2
Deliverable Name:	Deployment models of the CHIC technical architecture and its private cloud
Contractual Submission Date:	31/07/2016
Actual Submission Date:	04/08/2016

Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

COVER AND CONTROL PAGE OF DOCUMENT	
Project Acronym:	**CHIC**
Project Full Name:	Computational Horizons In Cancer (CHIC): Developing Meta- and Hyper-Multiscale Models and Repositories for In Silico Oncology
Deliverable No.:	D5.1.2
Document name:	Deployment models of the CHIC technical architecture and its private cloud
Nature (R, P, D, O)[1]	R
Dissemination Level (PU, PP, RE, CO)[2]	PU
Version:	3.0
Actual Submission Date:	04/08/2016
Editor: Institution: E-Mail:	Giorgos Zacharioudakis FORTH gzaxar@ics.forth.gr

ABSTRACT:

This deliverable reports on the available deployment models of the CHIC architecture and gives an update on its current deployment view. In addition, the document lists the cloud deployment models that suit specialized cloud architectures and provides an overview of the current cloud deployment as well as a discussion on future prospects.

KEYWORD LIST:

Architecture, deployment, deployment models, cloud.

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n^{o} 600841.

[1] **R**=Report, **P**=Prototype, **D**=Demonstrator, **O**=Other

[2] **PU**=Public, **PP**=Restricted to other programme participants (including the Commission Services), **RE**=Restricted to a group specified by the consortium (including the Commission Services), **CO**=Confidential, only for members of the consortium (including the Commission Services)

MODIFICATION CONTROL			
Version	**Date**	**Status**	**Author**
0.1	1/05/16	ToC & Initial Draft	Giorgos Zacharioudakis, Stelios Sfakianakis, Manolis Tsiknakis
0.9	30/07/16	Draft	Giorgos Zacharioudakis
1.0	2/8/2016	Internal review	Stelios Sfakianakis, Manolis Tsiknakis, Ioannis Karatzanis
2.0	4/8/2016	Pre-final	Giorgos Zacharioudakis
3.0	4/8/2016	Final version, submitted to EC	Giorgos Zacharioudakis

List of contributors

- Giorgos Zacharioudakis, FORTH
- Stelios Sfakianakis, FORTH
- Ioannis Karatzanis, FORTH
- Manolis Tsiknakis, TEI-C
- Georgios Stamatakos, ICCS
- Dimitra Dionysiou, ICCS
- Nikolaos Tousert, ICCS
- Elias Neri, CUSTODIX
- Daniele Tartarini, USFD
- Philippe Buechler, UBERN
- Roman Niklaus, UBERN
- Nigel McFarlane, BED
- Shaopeng Wu, BED
- Pierre Grenon, UCL
- Holger Stenzhorn, USAAR

Contents

1 Executive Summary

This deliverable reports on the available deployment models of the CHIC architecture and gives an update on its current deployment view. In specific, the operational environment of the CHIC platform, the dependencies and details of the technical implementation are presented and the advantages and the drawbacks of alternative deployment models that could be used for the realization of the CHIC architecture on different use cases, are discussed.

We also present the current deployment status of the CHIC private cloud and outline exemplar cases of alternative deployment models of cloud infrastructures, which Openstack documents with detailed architectural analysis and discuss their potential use as target models on a future exploitation plan that demands a productive system of high scale and quality.

2 Introduction

2.1 *Purpose of this document*

In this document we provide a status update on the deployment view of the CHIC architecture. We describe the latest updates on the environment and the dependencies of the CHIC components, different deployment models that could be used in the deployment of the CHIC architecture depending on different use cases or restrictions and finally some examples of deployment models and considerations on designing the cloud infrastructure.

2.2 *CHIC architecture*

In the CHIC deliverable D5.1.1 we documented the initial version of the CHIC technical architecture based on the IEEE 1471 - ISO/IEC 42010:2007 standard. By using this standard we can describe the architecture of a complex software system based on a set of Views and the corresponding Viewpoints –different facets of the system as highlighted from corresponding perspectives.

The standard itself does not specify the Views that are necessary to describe the architecture of a system. The Views may vary depending on the system, the stakeholders, where the architect wants to focus most and to which extend of details. In D5.1.1 we selected the following subset out of the views proposed in the Rozanski and Woods model[3] for architecture description:

- The **Functional view** that documents the system's functional elements, their responsibilities, interfaces, and primary interactions.
- The **Information view** that documents the way that the architecture stores, manipulates, manages, and distributes information.
- The **Deployment view** that describes the environment into which the system is deployed, including the dependencies the system has on its runtime environment.
- The **Security view** that describes the security framework employed in CHIC and that was included to the architecture due to the importance of the security to the CHIC platform, because of the nature of the data and the stakeholder's principles and constraints.

The final CHIC architecture will be documented in the deliverable D5.1.3, where an update of the information on all the above Views will be compiled. In this document we provide an update on the Deployment View as an interim report on the definition of the CHIC architecture and the interconnection with the integration of all functional components to a usable platform, in collaboration of WP5 with WP10. In addition, we provide some insights on alternative deployment models that could be used either in the current CHIC platform or future deployments of its architecture.

2.3 *Deployment models of the CHIC architecture*

The CHIC architecture is independent from the underlying technical infrastructure, the CHIC private cloud. The deployment on the cloud was a technical decision since the initial design of the architecture due to the great number of advantages that it offers compared to different approaches,

[3] Rozanski, Nick, and Woods, Eoin. Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives, 2nd Edition. Addison Wesley, 2011.

however, the CHIC architecture combines a set of modular components that do not assume or require the use of the cloud. The CHIC architecture could as well be deployed to one server computer, following a centralized approach with a tight integration of components, or to several distributed computers. The current deployment model, where the CHIC components are located to the same physical location but on independent Virtual Machines (VMs), combines both the centralized and the distributed approaches to a hybrid model that takes advantage of both worlds. Depending on the use case, the availability of computation resources, the agility to the development process etc. the current deployment model could be easily adapted. Although this flexibility is usually easier to be applied to the backend services that are hidden from the end-user, different deployment models are offered also to the front-end services and user interfaces. In chapter 3 we elaborate on these alternatives.

2.4 *Deployment models of the CHIC cloud*

The CHIC private cloud could be characterized as a general purpose cloud, providing usual functionality on the various aspects of such an infrastructure, on the computation, storage and network modules. However, when designing a highly scalable system there are considerations that should be examined early in the design phases. The CHIC platform aims to provide a system that could cope with high demands on data storage, computation load for the execution of models, and strict security considerations that result from the legal framework in which CHIC operates. We list some examples of typical cloud deployments that could give a rough estimate on where CHIC lies in terms of future needs. Identification of the CHIC operational demands would give a guide on how the CHIC platform should evolve to serve these needs if extended to be a production platform with an exploitation plan.

2.5 *Structure of the deliverable*

The rest of this document is organized as following: In chapter 3 we elaborate on the drawbacks and advantages of different deployments of the CHIC architecture, independently of the underlying cloud infrastructure and in chapter 4 we provide an update on the current deployment view of the architecture, in order to clearly illustrate the technical details and restrictions of the platform. In chapter 5 we examine alternative cloud deployment architectures as proposed by Openstack developers, to adapt to different operational and functional requirements. In chapter 6 we provide a view on the current CHIC cloud deployment and some remarks on alternative future deployments. Finally in chapter 7 we discuss some conclusions and remarks on possible future deployments of CHIC based on exploitation and operational scenarios.

3 Deployment Models of the CHIC architecture

The deployment of the CHIC architecture can follow different models such as a centralized, a distributed or a hybrid model for the deployment of its components. Alternative deployment models are easier to be applied to the backend services that are mostly used by developers or through programmatic interfaces. There are, however, alternative deployment models available also for the front-end services and the user interfaces of the platform.

Below we discuss the advantages and drawbacks of each one of them.

3.1 Deployment models for backend services

As described already in D5.1.1 the deployment model that has currently been employed relies on most of the backend CHIC components deployed on Virtual Machines on the CHIC private cloud infrastructure. Although the components reside to the same physical location this technical configuration could be seen as a hybrid rather than a centralized deployment model. A typical centralized deployment would have all components installed on a single server or integrated into one backend system. However, such a centralized system could only serve as a prototype; it could not scale to meet the operational demands of a production system with large data sets and many executions of different models and hypermodels.

Even in the current phase of the CHIC platform deployment where it is only used by the CHIC consortium and not by external users and organizations, a centralized approach could not scale up to meet the operational requirements. This is the reason why the nearest equivalent of a centralized CHIC deployment is a private cloud based deployment.

3.1.1 Centralized, private cloud based

3.1.1.1 Development

Although the various components are all deployed on a centrally located site, in the CHIC private cloud infrastructure at FORTH premises, the majority of components are installed on separate VMs. Having the components deployed on different VMs, even if all reside to the same physical location, affects the assumptions made during development and the degree of integration or separation between them in a number of parameters such as security, programming interfaces, data storage etc. This configuration allows us to maintain a clear functional separation of the developed modules. The developers have flexibility in terms of the environment, libraries and resources that they use and this separation is nearly equivalent of the distributed model where developers have full control and flexibility on their local sites and configurations. In this centralized model developers can have access as needed by having a common access policy, such as a shared VPN server or a common list of permitted remote IP addresses. Network issues might arise now and then, in network connectivity or speed, but the experience has shown that these issues are rare. Network issues arise also on the need to upload all data to a central location; however, this is a one-off problem that pays out later by having all data and computations located on the same physical premises.

In addition, by maintaining a functional separation of the components, if needed on a usage or future exploitation scenario we have the ability to easily install the components on remote sites and transform the deployment to a distributed model. In addition, the cloud infrastructure gives the ability to download an image of a complete VM instance, thus cloning and re-producing locally all components; a use case however that would need further configurations for its deployment.

However, we note that in a centralized deployment model there is a tight integration of the components. When bugs have to be fixed, new functionality has to be added, or plain maintenance tasks to take place these are better handled in a distributed model where the developers have direct hands-on access to make the necessary interventions, implementations or configurations. There is also the case of data, where having it stored on the premises of data providers ensures that the data sets are more up to date and better curated.

3.1.1.2 Maintenance and administration

Maintenance and administration is easier in the centralized cloud model, due to the functionality provided by the cloud infrastructure. New VMs can be created, cloned, transferred or expanded in a few minutes. The cloud infrastructure makes easy the provision of images of tested and pre-configured VMs, thus ensuring a common ground for developers, modellers or end-users. Administration on many day-to-day tasks can be performed by the cloud administrators, thus minimising the need for additional technical stuff on the premises of each partner. Due to the nature of the cloud infrastructure in terms of hardware virtualization, fault tolerance, data replication etc., many of the usual administration concerns on physical hardware such as server crashes or hard disk failures do not exist. Data backups are easily performed and snapshots of VMs can be stored and re-instantiated at will. This deployment scenario also makes easier the future exploitation of the platform since it minimizes the needs on technical personnel to maintain the production status of the platform. However, in future exploitation scenarios this cost of maintenance needs to be accounted. In the distributed model that we outline later below, this cost can be equally distributed to all partners by maintaining their own tools.

3.1.1.3 Security

One of the major advantages of the private cloud infrastructure, if not the biggest, is the tools that it offers to ensure the enforcement of a security policy. Communication can be strictly controlled both inside the CHIC platform components as well as between CHIC platform and the public network. Access policies can be globally enforced and monitored to all services, users and data. Besides the remote access that developers have for the needs of implementation, all components and data are isolated from the public network and any unauthorized access. The CHIC platform can expose to the public only the necessary interfaces, either programmatic (API) or user interface. Access to developers, usually through Secure Shell (SSH), is controlled separately than access for functional needs of communication between components.

3.1.1.4 Resource usage

Another major advantage of the centralized deployment on a cloud infrastructure is the usage of a common pool of resources, allowing both for elasticity and on-demand provision, thus better resource utilization, as well as easy expansion and scaling of the platform by the flexible and user-transparent ability to add more physical resources. Although the need to use local resources for development still exists on a few cases, the majority of resources needed for the CHIC platform to be up and running and providing services and data to its users are hosted on a central location, making easier the future exploitation of the platform as noted also in the Maintenance and administration section.

3.1.2 Distributed

The alternative deployment model that could be employed is the totally distributed model, in which the developers host their services or data in their own premises, providing hardware resources and technical personnel. Although this model was not used in CHIC, the current hybrid design and the cloud functionality gives us the ability to transform it to a totally distributed deployment model on a future exploitation plan. Below we outline the advantages and disadvantages of such a model.

3.1.2.1 Development

The development tasks in this model are performed easier in this task since the developers have direct access and intervention to their tools when any issue arises. During the course of the project, a distributed deployment model makes the development very agile and flexible to respond to changing requirements and adopting research innovation. However, this model adds additional load to the integration activities due to the network dependencies and the environment settings that need to be properly aligned.

In addition, the distributed deployment model makes easier the upload and data curation of the data sets, not only because it is performed on their local premises but also can avoid the curation of duplicate copies which exist on the centralized model.

3.1.2.2 Maintenance and administration

The distributed model assumes that each partner has to maintain their own tools, employ technical personnel and have appropriate budget for such tasks. This model could redirect a significant portion of labour to second-priority tasks that would distract developers from their primary tasks.

The distributed model makes easier tasks such as bug-fixes and upgrades of the tools after the project finishes, provided however that each partner keeps providing personnel and hardware resources for the project. This task however seems difficult for the maintenance of the integrated platform as a whole without securing appropriate budget, because many partners could not sustain those tools or data curation without a specific future exploitation roadmap.

3.1.2.3 Security

One of the major drawbacks of the distributed model in the case of CHIC platform is the security aspects of the project. The nature of the CHIC data and the strict legal requirements call for security policies that are not easy to install, monitor and enforce in a distributed deployment model. Access to data has to be strictly monitored and isolated. The data are used for computations and it would not be easy to be transferred over the network combining both the performance and security considerations.

3.1.2.4 Resource usage

In the distributed model, a major bottleneck is the network since all data have to be transferred many times. For big datasets or for many executions of the same computation, this would impose a huge limitation. Considerable effort would have to be placed on the design of the system in order to be scalable.

3.2 Deployment models for front-end services and user interfaces

3.2.1 Desktop applications

The most classic deployment model for the front-end services is to develop them as desktop applications. This provides the developers with total flexibility as to what programming language and configuration to use for each type of operating system or environment, gives them access to local resources (such as local files) and minimizes the security restrictions. It provides better responsiveness, since the application takes full advantage of the local computation resources. In addition, the desktop application can provide to the end-user a better user experience by integrating all functionality with a common look and feel. This model also permits the utilization of legacy applications that cannot be easily adapted to be used in different environments.

This model however has its drawbacks. The upgrade, bug-fixing or troubleshooting is not easy. For different operating systems or environments there need to be different versions or releases of the software, complicating the development as well as the maintenance. Each user takes care of the installation or configuration or the software, a task that could be very difficult or error-prone for many users especially for complex systems with many dependencies.

In CHIC, this model was used with various tools, such as the standalone (desktop) version of CRAF/Data upload tool and the DrEye application.

3.2.2 Web based

Another typical development and deployment model for front-end services and user interface is the web application model. This model provides a unified view for all users, can be updated/upgraded instantly as soon as new functionality is developed and does not require from the end-users any administration or configuration tasks. It allows to the developers to provide almost global user coverage by focusing on 2-3 major browsers, rather than maintaining the system for all operating systems. However, this model has some disadvantages such as security restrictions on accessing local resources and making it difficult to integrate diverse software that usually comprises a complex system. Many times, the web based applications could be very unresponsive due to the network and resource limitations, thus providing a negative user experience. Also, it is difficult to adapt legacy or complex applications to web-based ones.

In CHIC we have employed this model on a number of components, such as the user interface of the various repositories (Clinical Data Repository, Model Repository, InSilico Trial Repository) and also the web version of CRAF (Clinical Research Application Framework).

3.2.3 Desktop as a Service

A deployment model that is gaining momentum lately with the cloud technology expansion is the Desktop as a Service (DaaS). In this model, the user connects to a remote environment that resembles a typical desktop environment, hosted on a cloud infrastructure. With this hybrid model, the user can take benefit of both models; the desktop and the web based deployment. The DaaS provides pre-configured images of an environment with all the necessary installations and configurations, thus eliminating the need for a user to take care of those tasks. The update/upgrade of the installed applications is as easy as updating a web based application, thus instantly providing the latest version to all users. The user is using resources of the cloud, so the developers can take advantage of its resources and provide better virtual appliances as necessary. In addition, the DaaS can provide the familiar desktop environment of a classic operating system, thus providing the developers the ability to use legacy or complex systems, without adapting them to web applications.

This model has been partially used in CHIC for evaluation purposes, during an evaluation workshop. Various CHIC tools had been installed to a cloud VM which was then cloned to many instances. Users could remotely connect to those VMs to use and evaluate the installed applications, without the mundane task to install them locally to their PC, without the need to provide the necessary environment, libraries and configuration. In addition, this model presents some unique advantages in terms of the security, since the user is given access directly inside the cloud infrastructure to use tools or data but inside a secure sandbox and not accessing from its own local environment. The only resources that each user need was an ordinary computer such as a laptop and a network connection. This model could be also used in the exploitation of the CHIC platform by providing DaaS to end-users either for demonstration purposes or as an alternative to classic deployments.

3.2.4 All-in-one downloadable VM

Another deployment model that is often used, usually for demonstration purposes, is a pre-built VM that contains all necessary tools and configurations for a user to use and evaluate a system. With this model, similar to the DaaS, the user can use and evaluate the tools without installing and configuring anything. By having a VM running locally the user does not need network connection and the latency that this might impose. However, this model has drawbacks such as performance issues with running on a virtual environment and difficulty in importing and exporting data. It does not scale up and it could be difficult to use even for small data sets or computation load. At the same time, this model needs from the end-user to have at least some basic knowledge of using a virtual environment and can be frustrating and have a rather negative user experience for productive usage.

This deployment model has not been used in CHIC, but if needed on a future use case we could easily introduce it by taking advantage of the cloud functionality. We can build VM images with the DaaS model and export them as downloadable VMs for the end-users.

4 Current Deployment View

4.1 *Introduction*

In this section we document the current Deployment view of the architecture, which describes the environment where the system is deployed including the dependencies that the system has on its runtime environment. This view captures the hardware and software environment needs of the system, the technical environment requirements for each element, the mapping of the software elements to the runtime environment that executes them and any other dependency to technical resources, such as hardware or software that is required.

The technical details of the implementation of a system are not usually the main concern of its architecture specification. These details, however, often govern and restrict many of the design decisions of the architecture while trying to provide a feasible and functional system. Thus, for a complex software system such as the CHIC platform it is helpful, if not necessary, to describe in detail its Deployment View and the running environment of the system, in order to better illustrate the technical implementation of the system and the design choices that have been made.

4.2 *Network topology*

The CHIC components have been installed on Virtual Machines (VMs) on the CHIC private cloud. Some components are installed on dedicated instances of VMs while others share the same instance, depending on the resource needs or functional needs of each component. All CHIC VMs are placed in a dedicated subnet (virtual LAN) inside the private cloud where only CHIC instances are running, both for development flexibility as well as security separation with the external (public) network. In Figure 1 we present the network topology of the CHIC subnet.

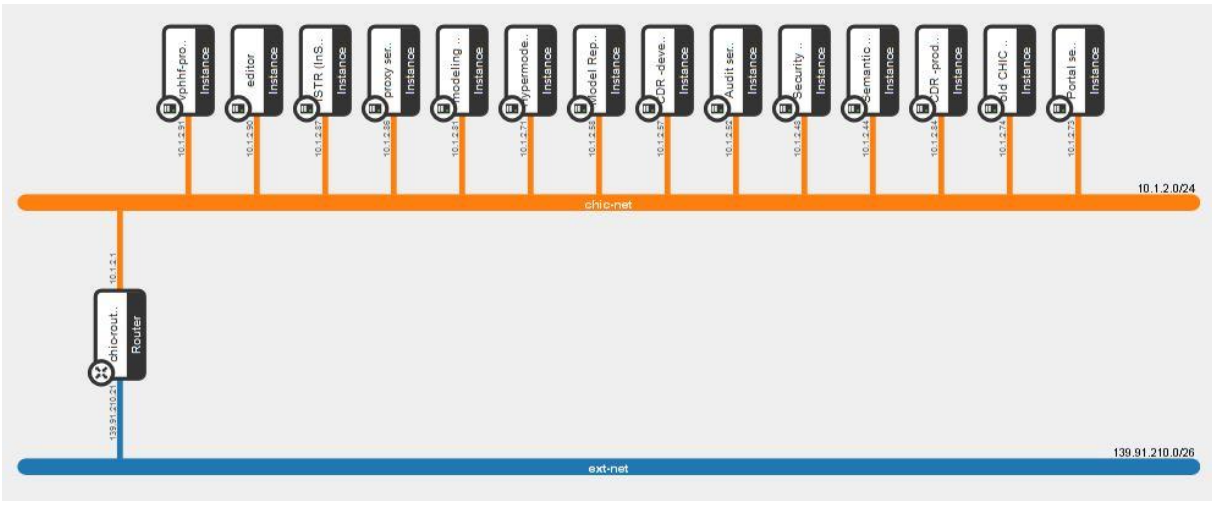
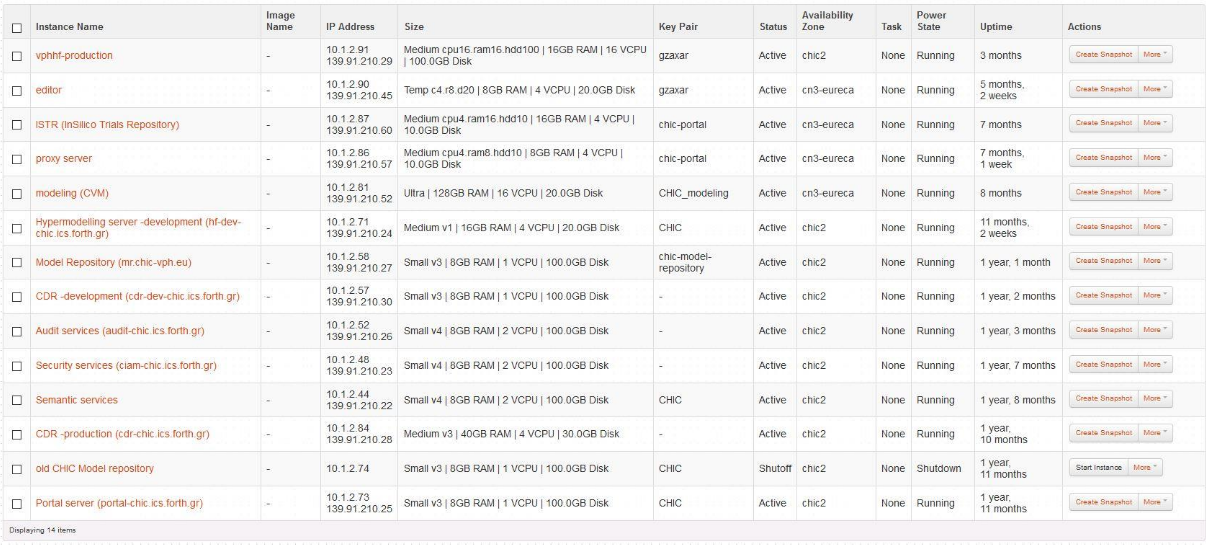


Figure 1 Network topology of the CHIC Virtual Machine instances (screenshot taken from the Openstack Dashboard)

In this figure we see a list of the CHIC VMs that communicate through the chic-net (CHIC dedicated subnet) and connect with the ext-net network (public network) of the cloud, through a virtual router that connects both interfaces.

All VMs are assigned a local Internet Protocol (IP) address inside the subnet for the intra-cloud communication. There is also the ability to assign a public IP address to each VM that forwards all traffic to the local IP address, in order to allow direct communication with the public network, usually for testing purposes and developer's remote access. All communication with the public network passes through a cloud firewall where incoming and outgoing traffic is restricted based on a set of security rules, such as allowed communication ports and allowed remote IP addresses. Security rules may be applied also to the local level (chic-net) if needed inside each VM instance, by configuring a local firewall on each instance.

4.3 Cloud resources used for CHIC VMs

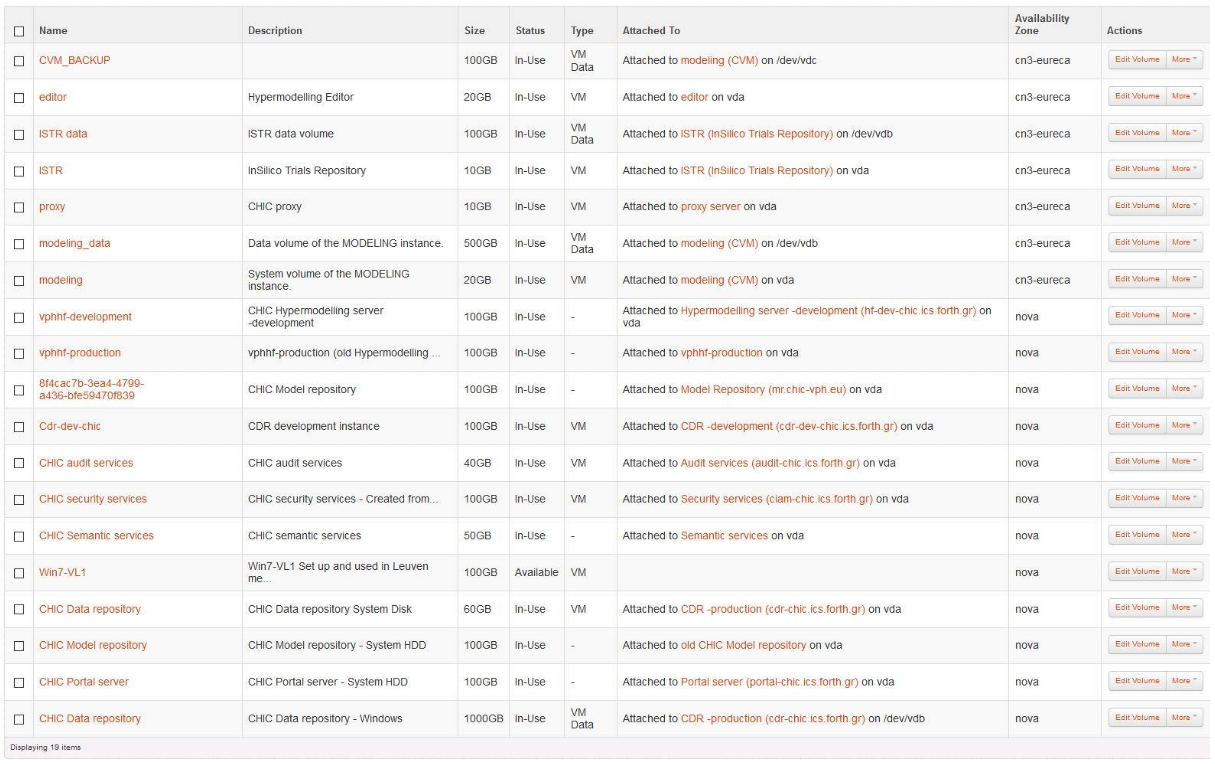


☐	Instance Name	Image Name	IP Address	Size	Key Pair	Status	Availability Zone	Task	Power State	Uptime	Actions
☐	vphhf-production	-	10.1.2.91 139.91.210.29	Medium cpu16.ram16.hdd100 | 16GB RAM | 16 VCPU | 100.0GB Disk	gzaxar	Active	chic2	None	Running	3 months	Create Snapshot More
☐	editor	-	10.1.2.90 139.91.210.45	Temp c4.r8.d20 | 8GB RAM | 4 VCPU | 20.0GB Disk	gzaxar	Active	cn3-eureca	None	Running	5 months, 2 weeks	Create Snapshot More
☐	ISTR (InSilico Trials Repository)	-	10.1.2.87 139.91.210.60	Medium cpu4.ram16.hdd10 | 16GB RAM | 4 VCPU | 10.0GB Disk	chic-portal	Active	cn3-eureca	None	Running	7 months	Create Snapshot More
☐	proxy server	-	10.1.2.86 139.91.210.57	Medium cpu4.ram8.hdd10 | 8GB RAM | 4 VCPU | 10.0GB Disk	chic-portal	Active	cn3-eureca	None	Running	7 months, 1 week	Create Snapshot More
☐	modeling (CVM)	-	10.1.2.81 139.91.210.52	Ultra | 128GB RAM | 16 VCPU | 20.0GB Disk	CHIC_modeling	Active	cn3-eureca	None	Running	8 months	Create Snapshot More
☐	Hypermodelling server -development (hf-dev-chic.ics.forth.gr)	-	10.1.2.71 139.91.210.24	Medium v1 | 16GB RAM | 4 VCPU | 20.0GB Disk	CHIC	Active	chic2	None	Running	11 months, 2 weeks	Create Snapshot More
☐	Model Repository (mr.chic-vph.eu)	-	10.1.2.58 139.91.210.27	Small v3 | 8GB RAM | 1 VCPU | 100.0GB Disk	chic-model-repository	Active	chic2	None	Running	1 year, 1 month	Create Snapshot More
☐	CDR -development (cdr-dev-chic.ics.forth.gr)	-	10.1.2.57 139.91.210.30	Small v3 | 8GB RAM | 1 VCPU | 100.0GB Disk	-	Active	chic2	None	Running	1 year, 2 months	Create Snapshot More
☐	Audit services (audit-chic.ics.forth.gr)	-	10.1.2.52 139.91.210.26	Small v4 | 8GB RAM | 2 VCPU | 100.0GB Disk	-	Active	chic2	None	Running	1 year, 3 months	Create Snapshot More
☐	Security services (ciam-chic.ics.forth.gr)	-	10.1.2.48 139.91.210.23	Small v4 | 8GB RAM | 2 VCPU | 100.0GB Disk	-	Active	chic2	None	Running	1 year, 7 months	Create Snapshot More
☐	Semantic services	-	10.1.2.44 139.91.210.22	Small v4 | 8GB RAM | 2 VCPU | 100.0GB Disk	CHIC	Active	chic2	None	Running	1 year, 8 months	Create Snapshot More
☐	CDR -production (cdr-chic.ics.forth.gr)	-	10.1.2.84 139.91.210.28	Medium v3 | 40GB RAM | 4 VCPU | 30.0GB Disk	-	Active	chic2	None	Running	1 year, 10 months	Create Snapshot More
☐	old CHIC Model repository	-	10.1.2.74	Small v3 | 8GB RAM | 1 VCPU | 100.0GB Disk	CHIC	Shutoff	chic2	None	Shutdown	1 year, 11 months	Start Instance More
☐	Portal server (portal-chic.ics.forth.gr)	-	10.1.2.73 139.91.210.25	Small v3 | 8GB RAM | 1 VCPU | 100.0GB Disk	CHIC	Active	chic2	None	Running	1 year, 11 months	Create Snapshot More

Displaying 14 items

Figure 2 CHIC VM instances and their resources (screenshot from Openstack Dashboard)

In Figure 2 we take a closer look on the CHIC VM instances running in the CHIC private cloud and some details on their assigned resources. This is not a complete list of their attached volumes (virtual storage drives) as some VMs are assigned extra storage space by attaching additional volumes. Below, in Figure 3, we see the complete list of volumes attached to the instances mentioned above.



☐	Name	Description	Size	Status	Type	Attached To	Availability Zone	Actions
☐	CVM_BACKUP		100GB	In-Use	VM Data	Attached to modeling (CVM) on /dev/vdc	cn3-eureca	Edit Volume More
☐	editor	Hypermodelling Editor	20GB	In-Use	VM	Attached to editor on vda	cn3-eureca	Edit Volume More
☐	ISTR data	ISTR data volume	100GB	In-Use	VM Data	Attached to ISTR (InSilico Trials Repository) on /dev/vdb	cn3-eureca	Edit Volume More
☐	ISTR	InSilico Trials Repository	10GB	In-Use	VM	Attached to ISTR (InSilico Trials Repository) on vda	cn3-eureca	Edit Volume More
☐	proxy	CHIC proxy	10GB	In-Use	VM	Attached to proxy server on vda	cn3-eureca	Edit Volume More
☐	modeling_data	Data volume of the MODELING instance.	500GB	In-Use	VM Data	Attached to modeling (CVM) on /dev/vdb	cn3-eureca	Edit Volume More
☐	modeling	System volume of the MODELING instance.	20GB	In-Use	VM	Attached to modeling (CVM) on vda	cn3-eureca	Edit Volume More
☐	vphhf-development	CHIC Hypermodelling server -development	100GB	In-Use	-	Attached to Hypermodelling server -development (hf-dev-chic.ics.forth.gr) on vda	nova	Edit Volume More
☐	vphhf-production	vphhf-production (old Hypermodelling ...	100GB	In-Use	-	Attached to vphhf-production on vda	nova	Edit Volume More
☐	8f4cac7b-3ea4-4799-a436-bfe59470f839	CHIC Model repository	100GB	In-Use	-	Attached to Model Repository (mr.chic-vph.eu) on vda	nova	Edit Volume More
☐	Cdr-dev-chic	CDR development instance	100GB	In-Use	VM	Attached to CDR -development (cdr-dev-chic.ics.forth.gr) on vda	nova	Edit Volume More
☐	CHIC audit services	CHIC audit services	40GB	In-Use	VM	Attached to Audit services (audit-chic.ics.forth.gr) on vda	nova	Edit Volume More
☐	CHIC security services	CHIC security services - Created from...	100GB	In-Use	VM	Attached to Security services (ciam-chic.ics.forth.gr) on vda	nova	Edit Volume More
☐	CHIC Semantic services	CHIC semantic services	50GB	In-Use	-	Attached to Semantic services on vda	nova	Edit Volume More
☐	Win7-VL1	Win7-VL1 Set up and used in Leuven me...	100GB	Available	VM		nova	Edit Volume More
☐	CHIC Data repository	CHIC Data repository System Disk	60GB	In-Use	VM	Attached to CDR -production (cdr-chic.ics.forth.gr) on vda	nova	Edit Volume More
☐	CHIC Model repository	CHIC Model repository - System HDD	100GB	In-Use	-	Attached to old CHIC Model repository on vda	nova	Edit Volume More
☐	CHIC Portal server	CHIC Portal server - System HDD	100GB	In-Use	-	Attached to Portal server (portal-chic.ics.forth.gr) on vda	nova	Edit Volume More
☐	CHIC Data repository	CHIC Data repository - Windows	1000GB	In-Use	VM Data	Attached to CDR -production (cdr-chic.ics.forth.gr) on /dev/vdb	nova	Edit Volume More

Displaying 19 items

Figure 3 List of volumes attached to CHIC VMs (screenshot from Openstack Dashboard)

In the following section we outline the technical details for the deployment of each component of the architecture. We follow the layered categorization of D5.1.1 for practical reasons of mapping the components with their functionality as presented to that deliverable, although on the technical level such distinguishing does not really matter.

4.4 User Interface layer

4.4.1 User Portal

Resource	**Mapping**
Developer	FORTH
Software type (Web application, standalone application, service, library, framework etc.)	Web application/framework
Operating system	Windows/Linux (currently installed on Linux, but there’s no strict dependency or requirement)
Programming environment	Java programming language, Tomcat application server
Communication technologies or protocols with	HTTP/XML/JSON/REST services

other CHIC components	
Data types that it operates on	N/A
Technical dependencies or requirements (libraries, tools)	Liferay/Tomcat/MySQL
Computational needs (estimation of hardware needs)	Server machine running a web server. Depends on the expected load of the portal.
Storage needs (estimation of hardware needs)	1 GB of storage space
URL (if applicable)	https://portal.chic-vph.eu/

4.5 Business Logic layer

4.5.1 Hypermodelling editor

Resource	**Mapping**
Developer	FORTH
Software type (Web application, standalone application, service, library, framework etc.)	Web based , two-tier architecture
Operating system	Cross platform, but Linux is the preferred host Operating System for the server side.
Programming environment	Clojure over the Java Virtual Machine in the server, Javascript on the client (browser)
Communication technologies or protocols with other CHIC components	HTTP/REST based API with JSON payload, and message oriented communication over the RabbitMQ message broker
Data types that it operates on	Hypermodel descriptions coming from the Model Repository, and their RDF annotations from the CHIC Semantic infrastructure
Technical dependencies or requirements (libraries, tools)	Clojure/Jetty application server, PostgreSQL DBMS for persistence, Redis as a caching layer, Nginx web server as a web reverse proxy
Computational needs (estimation of hardware	Moderate requirements, a server machine with

needs)	plenty of RAM is always preferred.
Storage needs (estimation of hardware needs)	5 GB of storage but depending on usage
URL (if applicable)	N/A

4.5.2 Clinical Research Application Framework (CRAF)

Resource	**Mapping**
Developer	FORTH
Software type (Web application, standalone application, service, library, framework etc.)	Web application (two-tier, i.e. client side in Browser talking to the server side over HTTP APIs) but also a standalone application in Java exists
Operating system	Cross platform, but Linux is the preferred host Operating System for the server side.
Programming environment	Java 8 on the server side and the standalone application, Javascript on the client side of the web application
Communication technologies or protocols with other CHIC components	HTTP/REST based API with JSON payload, and message oriented communication over the RabbitMQ message broker
Data types that it operates on	Hypermodels as computational artifacts, patient data for invoking the hypermodels, results and visualization files for the outputs of the executions
Technical dependencies or requirements (libraries, tools)	Server side: Java/Jetty application server, PostgreSQL DBMS for persistence, Redis as a caching layer, Nginx web server as a web reverse proxy Client side: Javascript, Angular 2 Standalone Application: Java, Apache PDFBox

Computational needs (estimation of hardware needs)	Server side: 8 GB of RAM Client side/ application: 4 GB RAM
Storage needs (estimation of hardware needs)	Server side: 5 GB of storage but depending on usage
URL (if applicable)	N/A

4.5.3 Hypermodelling execution framework (VPHHF)

Resource	**Mapping**
Developer	USFD - CINECA
Software type (Web application, standalone application, service, library, framework etc.)	Software framework
Operating system	Cross-platform but the Unix/Linux release is the most tested one.
Programming environment	C++, python Taverna workflow server is based on Java: it is used but not developed in CHIC.
Communication technologies or protocols with other CHIC components	REST/HTTP APIs to access the hypermodelling framework components functionalities. Supported protocol is HTTP/SOAP is being considered for inclusion. SSH protocol is used for data transfer and launch some remote execution between the execution machines in the private cloud.
Data types that it operates on	Taverna Workflow description file and xMML coupled model description file. The hypermodelling framework will have also to access models, data, and any other information necessary to the workflow execution.
Technical dependencies or requirements	The execution framework is mainly constituted by VPHHF which is manly developed in Django webservice

(libraries, tools)	framework with MySql database and few Python components belonging to Django ecosystem. RabbitMQ/Celery are used for event and queue based interactions. Taverna server orchestrates the execution of the models and a series of Bash and Python scripts wrap every model to make it possible to launch via a standardised interface. Muscle library is necessary to execute models developed via message passing paradigm.
Computational needs (estimation of hardware needs)	The VPH-HF is an orchestration layer. Itself it needs a couple of cores, and maybe 1GB of RAM; to this it has to be added all the cores and memory that the hypomodels to be orchestrated require to execute. Taverna server (based on Tomcat server) requires few more GiB of memory (1GiB min). The computational power required grow less than linearly w.r.t the number of requests per second of workflow execution.
Storage needs (estimation of hardware needs)	The VPH-HF orchestrates multiple hypo/hypermodels execution that is performed in a sandbox in the local file system. The needs in term of storage will be the same as the hypomodels to be executed and is model dependent.
URL (if applicable)	https://sourceforge.net/projects/vphhf/ https://github.com/INSIGNEO/VPH-HF

4.5.4 Visualization toolkit

Resource	**Mapping**
Developer	BED
Software type (Web application, standalone application, service, library, framework etc.)	Standalone
Operating system	Windows
Programming environment	C++
Communication technologies or protocols with	Command line arguments allow tasks to be

other CHIC components	executed and data exchanged via files on local disk.
Data types that it operates on	Clinical data and data from model simulation
Technical dependencies or requirements (libraries, tools)	Open source libraries: VTK, ITK, Qt and Qwt
Computational needs (estimation of hardware needs)	Min screen size: small laptop (about 28x18cm) Graphics and CPU: Low to high depending on demands.
Storage needs (estimation of hardware needs)	Low
URL (if applicable)	---

4.5.5 DrEye Image processing toolkit

Resource	**Mapping**
Developer	FORTH
Software type (Web application, standalone application, service, library, framework etc.)	Standalone application
Operating system	Windows (version 8.1 or higher)
Programming environment	C# / C++ / Visual Basic .NET
Communication technologies or protocols with other CHIC components	It interacts indirectly via user interaction with the data repository. Currently, there is no direct communication with the hypermodeling framework planned.
Data types that it operates on	Medical images (.dcm or .mha format)
Technical dependencies or requirements (libraries, tools)	Dr. Eye and its plugin-environment serve as a basis for the toolkit. Moreover, it requires the Insight Registration and Segmentation ToolKit (ITK) library and .NET framework (3.5 or higher).
Computational needs (estimation of hardware needs)	No special requirements. A standard personal computer is sufficient to run the application.

Storage needs (estimation of hardware needs)	N/A
URL (if applicable)	http://biomodeling.ics.forth.gr/dreye http://biomodeling.ics.forth.gr/?page_id=8

4.5.6 Biomechanical Simulator

Resource	Mapping
Developer	UBERN
Software type (Web application, standalone application, service, library, framework etc.)	Muscle-coupled component for simulating biomechanical aspects of tumour growth (BMS)
Operating system	Linux
Programming environment	Programming languages: C++, XML
Communication technologies or protocols with other CHIC components	Multiscale coupling library & environment (MUSCLE)
Data types that it operates on	Segmentation images (mhd, mha, nii, vti), MUSCLE-provided information
Technical dependencies or requirements (libraries, tools)	**Tools:** - FeBIO **Libraries:** - Visualisation Toolkit (VTK) [v 6.2] - Computational Geometry Algorithms Library (CGAL) [v 4.6, with VTK support] - Xerces C++ XML parser [v 3.1.2] - Multiscale coupling library & environment (MUSCLE) [v 2] All dependencies included in packaged simulator
Computational needs (estimation of hardware needs)	> 8GB RAM
Storage needs (estimation of hardware needs)	--
URL (if applicable)	--

4.6 Data Integration layer

4.6.1 Model repository

Resource	**Mapping**
Developer	ICCS
Software type (Web application, standalone application, service, library, framework etc.)	Web based (the User Interface part)
Operating system	Cross-platform
Programming environment	Programming language: Python, XML, Javascript, HTML, CSS Database management system: MySQL community edition Web application framework: Django Web server: Apache HTTP server
Communication technologies or protocols with other CHIC components	• REST services through HTTP protocol and JSON • Passing of messages through AMQP protocol
Data types that it operates on	Models and tools (structured information and files)
Technical dependencies or requirements (libraries, tools)	1) MySQL community edition (GPL license link: http://www.gnu.org/licenses/old-licenses/gpl-2.0.html) 2) Django Rest Framework (Copyright (c) 2011-2016, Tom Christie All rights reserved. Link: http://www.django-rest-framework.org/#license) 3) djangosaml2 (Apache 2 license, link: https://pypi.python.org/pypi/djangosaml2/) 4) dm.xmlsec.binding (BSD license, link: https://pypi.python.org/pypi/dm.xmlsec.binding/1.3.2) 5) Python 2.7 (Open Source, link: https://www.python.org/download/releases/2.7/license/) 6) XML security library (MIT license, link: https://www.aleksey.com/xmlsec/) 7) Django (BSD license, link: https://www.djangoproject.com/foundation/faq/) 8) jQuery library (MIT license, link:

	https://jquery.org/license/) 9) Bootstrap Framework (MIT license, link: http://getbootstrap.com/getting-started/)
Computational needs (estimation of hardware needs)	Intel Xeon E312xx (1 vCPU)
Storage needs (estimation of hardware needs)	60 GB
URL (if applicable)	https://mr.chic-vph.eu

4.6.2 Clinical data repository

Resource	**Mapping**
Developer	UBERN
Software type (Web application, standalone application, service, library, framework etc.)	Web based
Operating system	The Web application is running on a Microsoft Windows Server 2012 R2 64bit.
Programming environment	Programming languages: C#, Javascript, HTML, CSS Database management system: Microsoft SQL Server 2012 R2. Web application framework: Microsoft ASP.NET Framework, ASP.NET MVC and ASP.NET Web API Web server: Microsoft Internet Information Services (IIS)
Communication technologies or protocols with other CHIC components	REST API, JSON
Data types that it operates on	Clinical data (CDISC ODM) Imaging data (DICOM, MetaImage, Analyze, Nifti) Genetic / molecular data (MINiML) Histopathology data (JPEG, CSV, XML)

Technical dependencies or requirements (libraries, tools)	- ASP.NET (http://www.asp.net/) - Entity Framework (https://github.com/aspnet/EntityFramework) - SimpleITK (http://www.simpleitk.org/) - ReCaptcha (https://www.google.com/recaptcha) - Fuseki (http://jena.apache.org/) - Fellow Oak DICOM for .NET (https://github.com/fo-dicom/fo-dicom) - dotNetRDF (http://dotnetrdf.org/) - Newtonsoft Json (http://www.newtonsoft.com/json) - VDS.Common (https://www.nuget.org/packages/VDS.Common/) - jQuery (https://jquery.org/) - Bootstrap (http://www.getbootstrap.com) - HDF5DotNet (http://hdf5.net/) - Statismo (https://github.com/statismo/statismo) - FontAwesome (http://fortawesome.github.io/Font-Awesome) - Google Web Fonts (https://www.google.com/fonts) - ANTLR (http://www.antlr.org/) - Helix Toolkit (https://github.com/helix-toolkit) - Apache log4net (https://logging.apache.org/log4net/) - Modernizr (https://modernizr.com/) - OWIN (http://owin.org/) - RestSharp (http://restsharp.org/) - Web Grease (https://www.nuget.org/packages/WebGrease/) - AngularJS (https://angularjs.org/) - Elasticsearch, Logstash, Filebeat, Kibana (https://www.elastic.co/de/products) - Rdfstore, LOLS (http://open-physiology.org)

Computational needs (estimation of hardware needs)	Production VM: 40 GB RAM, 2.9 GHz (4 sockets and 4 virtual processors) Development VM: 8 GB RAM, 2.9 GHz (1 socket and 1 virtual processor)
Storage needs (estimation of hardware needs)	100 GB (depending on the number of patients)
URL (if applicable)	https://cdr.chic-vph.eu https://cdr-dev-chic.ics.forth.gr

4.6.3 In silico trial repository

Resource	**Mapping**
Developer	ICCS
Software type (Web application, standalone application, service, library, framework etc.)	Web based (the User Interface part)
Operating system	Cross-platform
Programming environment	Programming language: Python, XML, Javascript, HTML, CSS Database management system: MySQL community edition Web application framework: Django Web server: Apache HTTP server
Communication technologies or protocols with other CHIC components	• REST services through HTTP protocol and JSON
Data types that it operates on	Input data, links to hypermodels and results of model execution (structured information and files)
Technical dependencies or requirements (libraries, tools)	1) MySQL community edition (GPL license link: http://www.gnu.org/licenses/old-licenses/gpl-2.0.html) 2) Django Rest Framework (Copyright (c) 2011-2016, Tom Christie All rights reserved. Link: http://www.django-rest-framework.org/#license)

	3) djangosaml2 (Apache 2 license, link: https://pypi.python.org/pypi/djangosaml2/) 4) dm.xmlsec.binding (BSD license, link: https://pypi.python.org/pypi/dm.xmlsec.binding/1.3.2) 5) Python 2.7 (Open Source, link: https://www.python.org/download/releases/2.7/license/) 6) XML security library(MIT license, link: https://www.aleksey.com/xmlsec/) 7) Django (BSD license, link: https://www.djangoproject.com/foundation/faq/) 8) jQuery library (MIT license, link: https://jquery.org/license/) 9) Bootstrap Framework (MIT license, link: http://getbootstrap.com/getting-started/)
Computational needs (estimation of hardware needs)	Intel Xeon E312xx (4 vCPU)
Storage needs (estimation of hardware needs)	100 GB
URL (if applicable)	https://istr.chic-vph.eu

4.6.4 Metadata repository

Resource	**Mapping**
Developer	UCL
Software type (Web application, standalone application, service, library, framework etc.)	Web accessible database
Operating system	Cross-platform
Programming environment	N/A
Communication technologies or protocols with other CHIC components	SPARQL endpoint or Ontology-based services (REST)
Data types that it operates on	Metadata serialized in RDF

Technical dependencies or requirements (libraries, tools)	TBD
Computational needs (estimation of hardware needs)	TBD
Storage needs (estimation of hardware needs)	Web server application
URL (if applicable)	Third parties backend and https://github.com/open-physiology/rdfstore

4.6.5 Data upload tool

Resource	**Mapping**
Developer	FORTH
Software type (Web application, standalone application, service, library, framework etc.)	Web application, incorporated into CRAF
Operating system	Any
Programming environment	Javascript
Communication technologies or protocols with other CHIC components	HTTP/REST based API with JSON payload
Data types that it operates on	Patient data: CRFs, DICOM images, MIAME compatible files of genomic data
Technical dependencies or requirements (libraries, tools)	It's strongly coupled with CRAF, uses DICOM Toolkit (DCMTK, http://dicom.offis.de/dcmtk.php.en) for the DICOM file management
Computational needs (estimation of hardware needs)	4 GB RAM
Storage needs (estimation of hardware needs)	Nothing special, depends on the patient-related files to be managed
URL (if applicable)	NA

4.7 *Security layer*

The security framework is comprised of many different components, each one with its one technical details and configuration, so, we describe each component independently of the others.

4.7.1.1 IAM (Identity and Access Management site)

Resource	**Mapping**
Developer	CUSTODIX
Software type (Web application, standalone application, service, library, framework etc.)	Web application
Operating system	OS with Java available
Programming environment	Java
Communication technologies or protocols with other CHIC components	REST and SOAP Web Services
Data types that it operates on	CHIC User Model
Technical dependencies or requirements (libraries, tools)	
Computational needs (estimation of hardware needs)	1 core
Storage needs (estimation of hardware needs)	25 GB (for application code and log files)
URL (if applicable)	Development: https://ciam-dev-chic.custodix.com/idm Live: https://ciam.chic-vph.eu/idm

4.7.1.2 Idp (Identity Provider)

Resource	**Mapping**

Developer	CUSTODIX
Software type (Web application, standalone application, service, library, framework etc.)	Web application
Operating system	OS with Java available
Programming environment	Java
Communication technologies or protocols with other CHIC components	SAML v2 protocol
Data types that it operates on	
Technical dependencies or requirements (libraries, tools)	
Computational needs (estimation of hardware needs)	1 core
Storage needs (estimation of hardware needs)	25 GB (for application code and log files)
URL (if applicable)	Development: https://ciam-dev-chic.custodix.com/idp Live: https://ciam.chic-vph.eu/idp

4.7.1.3 STS (Secure Token Service)

Resource	**Mapping**
Developer	CUSTODIX
Software type (Web application, standalone application, service, library, framework etc.)	Web service
Operating system	OS with Java available
Programming environment	Java

Communication technologies or protocols with other CHIC components	REST and SOAP Web Services
Data types that it operates on	
Technical dependencies or requirements (libraries, tools)	
Computational needs (estimation of hardware needs)	1 core
Storage needs (estimation of hardware needs)	25 GB (for application code and log files)
URL (if applicable)	Development: https://ciam-dev-chic.custodix.com/sts Live: https://ciam.chic-vph.eu/sts

4.7.1.4 PDP (Policy Decision Point)

Resource	**Mapping**
Developer	CUSTODIX
Software type (Web application, standalone application, service, library, framework etc.)	Web service
Operating system	OS with Java available
Programming environment	Java
Communication technologies or protocols with other CHIC components	Web Service
Data types that it operates on	
Technical dependencies or requirements (libraries, tools)	
Computational needs (estimation of hardware needs)	1 core

Storage needs (estimation of hardware needs)	25 GB (for application code and log files)
URL (if applicable)	

4.7.1.5 PAP (Policy Administration Point)

Resource	**Mapping**
Developer	CUSTODIX
Software type (Web application, standalone application, service, library, framework etc.)	Web application
Operating system	OS with Java available
Programming environment	Java
Communication technologies or protocols with other CHIC components	SOAP and REST Web Service
Data types that it operates on	
Technical dependencies or requirements (libraries, tools)	
Computational needs (estimation of hardware needs)	1 core
Storage needs (estimation of hardware needs)	25 GB (for application code and log files)
URL (if applicable)	

4.7.1.6 PIP (Policy Information Point)

Resource	**Mapping**
Developer	CUSTODIX
Software type (Web application, standalone	Web service

application, service, library, framework etc.)	
Operating system	OS with Java available
Programming environment	Java
Communication technologies or protocols with other CHIC components	SOAP and REST Web Service
Data types that it operates on	
Technical dependencies or requirements (libraries, tools)	
Computational needs (estimation of hardware needs)	1 core
Storage needs (estimation of hardware needs)	25 GB (for application code and log files)
URL (if applicable)	

4.7.1.7 PEP (Policy Enforcement Point)

Resource	**Mapping**
Developer	CUSTODIX
Software type (Web application, standalone application, service, library, framework etc.)	Library
Operating system	
Programming environment	Java
Communication technologies or protocols with other CHIC components	Web Service
Data types that it operates on	
Technical dependencies or requirements (libraries, tools)	

Computational needs (estimation of hardware needs)	
Storage needs (estimation of hardware needs)	
URL (if applicable)	

4.7.1.8 Audit Services

Resource	**Mapping**
Developer	CUSTODIX
Software type (Web application, standalone application, service, library, framework etc.)	Web application
Operating system	OS with Java available
Programming environment	Java
Communication technologies or protocols with other CHIC components	Web Service
Data types that it operates on	
Technical dependencies or requirements (libraries, tools)	
Computational needs (estimation of hardware needs)	1 core
Storage needs (estimation of hardware needs)	50 GB (for application code and log files)
URL (if applicable)	Development: https://audit-dev-chic.custodix.com/audit-viewer https://audit-dev-chic.custodix.com/audit-parser Live: https://audit.chic-vph.eu/audit-viewer

	https://audit.chic-vph.eu/audit-parser

4.7.1.9 Security Gateway

Resource	**Mapping**
Developer	CUSTODIX
Software type (Web application, standalone application, service, library, framework etc.)	Web service
Operating system	OS with Java available
Programming environment	Java
Communication technologies or protocols with other CHIC components	Web Service
Data types that it operates on	N/A
Technical dependencies or requirements (libraries, tools)	N/A
Computational needs (estimation of hardware needs)	1 core
Storage needs (estimation of hardware needs)	25 GB (for application code and log files)
URL (if applicable)	

4.7.1.10 De-identification

Resource	**Mapping**
Developer	CUSTODIX
Software type (Web application, standalone application, service, library, framework etc.)	Web application
Operating system	OS with Java available

Programming environment	Java
Communication technologies or protocols with other CHIC components	SOAP and REST Web Service
Data types that it operates on	N/A
Technical dependencies or requirements (libraries, tools)	N/A
Computational needs (estimation of hardware needs)	2 core
Storage needs (estimation of hardware needs)	50 GB (for application code and log files)
URL (if applicable)	Development: https://ttp-dev-chic.custodix.com Live: https://ttp-chic.custodix.com

4.8 Semantics layer

4.8.1 Ontology-based semantic services

Resource	**Mapping**
Developer	UCL
Software type (Web application, standalone application, service, library, framework etc.)	Web services
Operating system	Cross-platform
Programming environment	JAVA
Communication technologies or protocols with other CHIC components	REST
Data types that it operates on	N/A

Technical dependencies or requirements (libraries, tools)	Metadata repository and Knowledge Base
Computational needs (estimation of hardware needs)	N/A
Storage needs (estimation of hardware needs)	Web server application
URL (if applicable)	https://github.com/open-physiology/chic https://github.com/open-physiology/owlkb https://github.com/open-physiology/rdfstore

4.8.2 Folksonomy semantic services

Resource	**Mapping**
Developer	BED
Software type (Web application, standalone application, service, library, framework etc.)	Web application, RESTful APIs
Operating system	Cross platform
Programming environment	Java, MongoDB, JavaScript, JQuery, Tomcat
Communication technologies or protocols with other CHIC components	RESTful APIs JSON
Data types that it operates on	Resource URI
Technical dependencies or requirements (libraries, tools)	Spring framework
Computational needs (estimation of hardware needs)	2 x CPU @ 2.6GHz, RAM 8 GB
Storage needs (estimation of hardware needs)	50GB for application, data, and logs

URL (if applicable)	Web app: http://api.ccgv.org.uk/taggingapp/ RESTful API: http://api.ccgv.org.uk/taggingservice/tags

4.8.3 Knowledge base

Resource	**Mapping**
Developer	UCL
Software type (Web application, standalone application, service, library, framework etc.)	Web accessible database
Operating system	Cross-platform
Programming environment	Java
Communication technologies or protocols with other CHIC components	Ontology-based services (REST) and Java API
Data types that it operates on	Baseline option: OWL (Ontology Web Language)
Technical dependencies or requirements (libraries, tools)	Baseline option: OWLAPI, Reasoners
Computational needs (estimation of hardware needs)	Intensive (Prototype needs dedicated server and 64Gb RAM)
Storage needs (estimation of hardware needs)	Web server application
URL (if applicable)	https://github.com/open-physiology/owlkb

4.9 Infrastructure layer

4.9.1 Private Cloud

Resource	**Mapping**
Developer	FORTH

Software type (Web application, standalone application, service, library, framework etc.)	OpenStack cloud technology platform. (Framework of tools.)
Operating system	Linux
Programming environment	N/A (The software is not being developed in CHIC. It is provided as free software under the Apache 2.0 license)
Communication technologies or protocols with other CHIC components	Does not communicate directly with other CHIC components. However it does provide many different technologies for communication and management: REST services: JSON/XML format, Command Line Interface (CLI) tools, Java SDK
Data types that it operates on	N/A
Technical dependencies or requirements (libraries, tools)	Operating System, network configuration, hardware resources.
Computational needs (estimation of hardware needs)	Many computer nodes (>5)
Storage needs (estimation of hardware needs)	Many TBs of storage (>5)
URL (if applicable)	https://www.openstack.org/

5 Cloud Deployment Architectures

The cloud infrastructure is the next generation of the classic data centres. It is a complex set of tools that provides functionality to manage hardware resources in a large scale and, what's more, to virtualize these resources and provide it as a service to the applications and the users.

This approach provides better utilization, automation, scalability, elasticity and on-demand provision. The cloud provides a large set of tools that manage and virtualize all different aspects of a data centre, such as storage, computation, network, computer instances and configuration, databases, billing, clustering, benchmarking as well as usual administrative tasks such as network configuration, data backup, security management, identity management, quotas and many others. These tasks can be automated or be handed over to the end-users rather than require trained administrative personnel.

The cloud infrastructure models are often categorized as *public*, *private* or *hybrid*, but this categorization focuses rather on the commercial exploitation and the security isolation of the cloud installation, rather than the actual deployment models, the different architectural choices that a cloud architect has to consider when designing and deploying such a complex system.

In D5.1.1 we performed an elaborate evaluation of the most prominent cloud infrastructure technologies and based on this analysis we chose Openstack to be our technological layer, focusing on solving the issues mentioned above. Openstack offers a wide array of deployment models that focus and adapt to the specific needs of different architectures and below we refer some examples that Openstack lists[4] as exemplar cases to be used as models when designing and deploying a cloud infrastructure. A reader that is interested in more details on the specific deployment architectures, operational and technical considerations and proposed deployment models is referred to study this Openstack reference guide for specific technical details and model configurations.

These deployment models can be used as guides when designing a cloud based architecture or when exploring the transition of a system from the small to the large scale. Thus, in principle these models can be used as guides also for the transition of CHIC platform from a research prototype to a large scale production system.

5.1 *General purpose*

An OpenStack general purpose cloud is often considered a starting point for building a cloud deployment. They are designed to balance the components and do not emphasize any particular aspect of the overall computing environment. Cloud design must give equal weight to the compute, network, and storage components. General purpose clouds are found in private, public, and hybrid environments, lending themselves to many different use cases.

Common uses of a general purpose cloud include:

- Providing a simple database
- A web application runtime environment
- A shared application development platform
- Lab test bed

A general purpose cloud is designed to have a range of potential uses or functions; not specialized for specific use cases. General purpose architecture is designed to address 80% of potential use cases available. The infrastructure, in itself, is a specific use case, enabling it to be used as a base model for the design process. General purpose clouds are designed to be platforms that are suited for general

[4] http://docs.openstack.org/arch-design/

purpose applications. General purpose clouds are limited to the most basic components, but they can include additional resources such as:

- Virtual-machine disk image library
- Raw block storage
- File or object storage
- Firewalls
- Load balancers
- IP addresses
- Network overlays or virtual local area networks (VLANs)
- Software bundles

5.2 *Compute focused*

Compute-focused clouds are a specialized subset of the general purpose OpenStack cloud architecture. A compute-focused cloud specifically supports compute intensive workloads. Compute intensive workloads may be CPU intensive, RAM intensive, or both; they are not typically storage or network intensive.

Compute-focused workloads may include the following use cases:

- High performance computing (HPC)
- Big data analytics using Hadoop or other distributed data stores
- Continuous integration/continuous deployment (CI/CD)
- Platform-as-a-Service (PaaS)
- Signal processing for network function virtualization (NFV)

A compute-focused OpenStack cloud does not typically use raw block storage services as it does not host applications that require persistent block storage.

5.3 *Storage focused*

Cloud storage is a model of data storage that stores digital data in logical pools and physical storage that spans across multiple servers and locations. Cloud storage commonly refers to a hosted object storage service, however the term also includes other types of data storage that are available as a service, for example block storage.

At large scale, management of data operations is a resource intensive process for an organization. Hierarchical storage management (HSM) systems and data grids help annotate and report a baseline data valuation to make intelligent decisions and automate data decisions. HSM enables automated tiering and movement, as well as orchestration of data operations.

Example applications deployed with cloud storage characteristics:

- Active archive, backups and hierarchical storage management.
- General content storage and synchronization. An example of this is private dropbox.
- Data analytics with parallel file systems.
- Unstructured data store for services. For example, social media back-end storage.
- Persistent block storage.
- Operating system and application image store.
- Media streaming.
- Databases.

- Content distribution.
- Cloud storage peering.

5.4 Network focused

All OpenStack deployments depend on network communication in order to function properly due to its service-based nature. In some cases, however, the network elevates beyond simple infrastructure. In this section we discuss architectures that are more reliant or focused on network services. These architectures depend on the network infrastructure and require network services that perform reliably in order to satisfy user and application requirements.

Some possible use cases include:

Content delivery network
This includes streaming video, viewing photographs, or accessing any other cloud-based data repository distributed to a large number of end users. Network configuration affects latency, bandwidth, and the distribution of instances.

Network management functions
Use this cloud to provide network service functions built to support the delivery of back-end network services such as DNS, NTP, or SNMP.

Network service offerings
Examples include VPNs, MPLS private networks, and GRE tunnels.

Web portals or web services
Web servers are a common application for cloud services. The network requires scaling out to meet user demand and deliver web pages with a minimum latency.

High speed and high volume transactional systems
These types of applications are sensitive to network configurations. Examples include financial systems, credit card transaction applications, and trading and other extremely high volume systems. These systems are sensitive to network jitter and latency. Many of these systems must access large, high performance database back ends.

High availability
These types of use cases are dependent on the proper sizing of the network to maintain replication of data between sites for high availability. If one site becomes unavailable, the extra sites can serve the displaced load until the original site returns to service.

Big data
Clouds used for the management and collection of big data (data ingest) have a significant demand on network resources. Big data often uses partial replicas of the data to maintain integrity over large distributed clouds. Other big data applications that require a large amount of network resources are Hadoop, Cassandra, NuoDB, Riak, and other NoSQL and distributed databases.

Virtual desktop infrastructure (VDI)
This use case is sensitive to network congestion, latency, jitter, and other network characteristics. Like video streaming, the user experience is important. However, unlike video streaming, caching is not an option to offset the network issues. VDI requires both upstream and downstream traffic and cannot rely on caching for the delivery of the application to the end user.

Voice over IP (VoIP)
This is sensitive to network congestion, latency, jitter, and other network characteristics. VoIP has a symmetrical traffic pattern and it requires network quality of service (QoS) for best performance. Users are sensitive to latency and jitter fluctuations and can detect them at very low levels.

Video Conference or web conference

This is sensitive to network congestion, latency, jitter, and other network characteristics. Similar to VoIP, users are sensitive to network performance issues even at low levels.

High performance computing (HPC)

This is a complex use case that requires careful consideration of the traffic flows and usage patterns to address the needs of cloud clusters.

5.5 *Multi-site*

OpenStack is capable of running in a multi-region configuration. This enables some parts of OpenStack to effectively manage a group of sites as a single cloud. Some use cases that might indicate a need for a multi-site deployment of OpenStack include:

- An organization with a diverse geographic footprint.
- Geo-location sensitive data.
- Data locality, in which specific data or functionality should be close to users.

5.6 *Hybrid*

A hybrid cloud design is one that uses more than one cloud. For example, designs that use both an OpenStack-based private cloud and an OpenStack-based public cloud, or that use an OpenStack cloud and a non-OpenStack cloud, are hybrid clouds.

Bursting describes the practice of creating new instances in an external cloud to alleviate capacity issues in a private cloud.

Example scenarios suited to hybrid clouds:

- Bursting from a private cloud to a public cloud
- Disaster recovery
- Development and testing
- Federated cloud, enabling users to choose resources from multiple providers
- Supporting legacy systems as they transition to the cloud

Hybrid clouds interact with systems that are outside the control of the private cloud administrator, and require careful architecture to prevent conflicts with hardware, software, and APIs under external control.

5.7 *Massively scalable*

A massively scalable architecture is a cloud implementation that is either a very large deployment, such as a commercial service provider might build, or one that has the capability to support user requests for large amounts of cloud resources.

An example of this is an infrastructure in which 500 or more requests to service instances at any given time is common. A massively scalable infrastructure fulfills such a request without exhausting the available cloud infrastructure resources. While the high capital cost of implementing such a cloud architecture means that it is currently in limited use, many organizations are planning for massive scalability in the future.

Services provided by a massively scalable OpenStack cloud include:

- Virtual-machine disk image library
- Raw block storage
- File or object storage
- Firewall functionality
- Load balancing functionality
- Private (non-routable) and public (floating) IP addresses
- Virtualized network topologies
- Software bundles
- Virtual compute resources

Like a general purpose cloud, the instances deployed in a massively scalable OpenStack cloud do not necessarily use any specific aspect of the cloud offering (compute, network, or storage). As the cloud grows in scale, the number of workloads can cause stress on all the cloud components.

5.8 Special cases

Although most OpenStack architecture designs fall into one of the seven major scenarios outlined in other sections (compute focused, network focused, storage focused, general purpose, multi-site, hybrid cloud, and massively scalable), there are a few use cases that do not fit into these categories. This section discusses these specialized cases and provide some additional details and design considerations for each use case:

- **Specialized networking**: describes running networking-oriented software that may involve reading packets directly from the wire or participating in routing protocols.
- **Software-defined networking (SDN)**: describes both running an SDN controller from within OpenStack as well as participating in a software-defined network.
- **Desktop-as-a-Service**: describes running a virtualized desktop environment in a cloud (Desktop-as-a-Service). This applies to private and public clouds.
- **OpenStack on OpenStack**: describes building a multi-tiered cloud by running OpenStack on top of an OpenStack installation.
- **Specialized hardware**: describes the use of specialized hardware devices from within the OpenStack environment.

5.9 Legal and security requirements

Many jurisdictions have legislative and regulatory requirements governing the storage and management of data in cloud environments. Common areas of regulation include:

- Data retention policies ensuring storage of persistent data and records management to meet data archival requirements.
- Data ownership policies governing the possession and responsibility for data.
- Data sovereignty policies governing the storage of data in foreign countries or otherwise separate jurisdictions.
- Data compliance policies governing certain types of information needing to reside in certain locations due to regulatory issues - and more importantly, cannot reside in other locations for the same reason.

An example of such legal frameworks is the data protection framework[5] of the European Union that affects the CHIC platform and the legal compliance requirements. In D5.3 we compiled a security analysis of the operational requirements that CHIC imposes to its cloud infrastructure. The most critical aspects of security that affect the architecture design and its technical implementation are:

- Identity management
- Authorization and Access control
- Data Placement in object storage
- Data encryption

Openstack provides tools for the necessary functionality regarding Identity Management, Authorization and Access control and Data placement. It does not provide direct Data Encryption, however this functionality can be achieved by using back end solutions, such as Logical Volume Manager (LVM) encrypted volumes.

[5] http://ec.europa.eu/justice/data-protection/

6 The CHIC private cloud

6.1 Introduction

The deployment of the CHIC cloud infrastructure has been described in detail in D5.3. In this chapter we give an update regarding the current status of the private cloud deployment and we examine alternative cloud deployment models based on those that Openstack proposes as exemplar cases of various architectures.

6.2 Current deployment configuration

In the initial –evaluation– deployment of the CHIC private cloud, for the purpose of internal hands-on evaluation of its applicability and appropriateness, three (3) nodes were used; One node for the cloud controller (Cloud controller, Identity services, Network services), one node for the image controller (Image services) and one node as compute and storage node.

During the course of the project, taking into account the additional computation and storage requirements the initial infrastructure was extended with two additional compute and storage nodes. By examining the usual computation load of the compute nodes we have verified that two (2) out of three (3) compute nodes are adequate in sustaining all the load that the project has demanded up to the present point of time. Nevertheless, the additional node was necessary, in order to achieve better load balancing, fault tolerance, and elastic capacity to unexpected load and computation demands.

The current hardware resources are summarized in the following table.

Server	Type	Configuration	Services
CHIC1	Dell PowerEdge SC1425	Intel Xeon 3.0 GHz, 16 GB RAM	Cloud controller, Network controller, Dashboard
CHIC2	Dell PowerEdge SC1425	Intel Xeon 3.0 GHz, 8 GB RAM	Image controller
CHIC3	Dell PowerEdge R720xd	2* Intel Xeon E5-2690 2.90 GHz, 32 cores, 256 GB RAM	Compute node, Storage node
CHIC4	Dell PowerEdge R730xd	2* Intel Xeon E5-2690 2.90 GHz v3, 48 cores, 256 GB RAM	Compute node, Storage node
CHIC5	Dell PowerEdge R730xd	2* Intel Xeon E5-2690 2.90 GHz v3, 48 cores, 512 GB RAM	Compute node, Storage node

Table 1 The hardware resources of the CHIC private cloud

In total our clouds provides an accumulated computation power of 128 cores and 1 TB of RAM and accumulated storage space of 20 TB both for VM images, VM instances and raw block storage. It formulates a small scale private cloud that provides the basis, given the hardware resources, to scale up to as many nodes and resources as needed.

6.3 Transition of the CHIC private cloud to large scale

The CHIC private cloud in its current state is not a large scale installation that could be compared with a commercial cloud in terms of hardware resources or number of users. However, depending on the exploitation plans of CHIC we could envision the CHIC platform on a broader scale, hosting many hypermodels, storing large data sets and serving a great number of users and organizations.

In such a large scale deployment it would be necessary to accordingly scale up the CHIC cloud, forming up a Platform-as-a-Service version of CHIC. However, a transition to large scale deployment would require appropriate cloud design based on the guidelines and best practices as suggested by the cloud experts.

Following the categorization and terminology of chapter 5, the CHIC cloud in its current architecture forms a **general purpose** cloud. It provides simple databases, web applications and services and a shared application development platform. It is not specialized for a specific set of problems and provides the usual basic cloud components such as virtual machines, raw block storage, IP addresses, virtual LANs, firewalls.

A CHIC private cloud that would aim to support many organizations, many users and connect a large number of hardware resources would have additional characteristics of some of the exemplar cloud models mentioned in chapter 5. It would require functionality of a **Storage** cloud, in order to store large volume data sets such as clinical data, imaging data, processed data and results of In Silico trials. It would also require functionality of a **Compute** cloud, in order to run massive parallel executions of models and hypermodels, as is the case in high performance computing. It would have a broad geographic coverage, connecting organizations in different regions or countries and it would also require the ability of geo-location of sensitive data for legal and security reasons. Thus, the cloud would resemble a federated **Multi-site** cloud in which some sites would mainly serve as Storage nodes, others would serve as Compute nodes and others as general purpose nodes hosting identity services, metadata, model repositories etc.

7 Conclusions

In the previous chapters we have explored various alternative models for the deployment of CHIC architecture as well as different deployment models of the underlying technical infrastructure. It is evident that the deployment of the architecture and its technical infrastructure have interdependent design parameters; a complex system that aims to be cloud ready and have the ability to scale up cannot have a monolithic, single-tiered architecture. Cloud utilization can be gained when the load can be spread over several instances and failures in parts of the system can be mitigated without affecting other parts.

Based on the fact that CHIC is a complex system requiring substantial hardware and software resources, that are more easily managed by a cloud layer and back up support by its developers, modelers and administrative staff, we have built the architecture on the assumption of "CHIC platform-as-a-Service", as opposed to the Platform-as-a-Service (PaaS) cloud model. A dimension though that has not been discussed in detail is the operational aspects of having "local" installations of the CHIC platform, providing "CHIC platform-as-an-Application". What would be the requirements or the implications if an organization, e.g. a hospital, wanted to make an in-house deployment of CHIC in its premises? Putting aside of course for the moment the fact that CHIC still needs a clinical evaluation and validation before used in the clinical and research.

There would be obvious implications on the architecture, concerning mainly the security configuration and identity management services but we argue that this use case would not differ very much compared with the deployment models we analyzed in the previous sections. It would rather differ to the operational context and the configuration of the deployment. In essence, the organization would still need a deployment such as

- A local private cloud, aiming to recreate the current CHIC environment.
- An all-in-one centralized installation, either to a single server or to a Virtual Machine. This approach, as we argued before in this document, could serve mostly for evaluation and demonstration purposes and not productive usage on a large scale.
- A distributed deployment that would share parts of the deployment with a remote CHIC platform. Security implications would rather make difficult the deployment of such a model.
- Hiring services of a public cloud. This operational context is outside of the scope of the current CHIC architecture due to legal restrictions and would require major modifications to the security architecture.

An organization might not require specific parts of CHIC, such as the Hypermodelling editor, but major components such as the Data Repository, the Hypermodelling Execution Framework and the Identity management services (authorization, authentication etc) would be required in most cases. An installation locally to a hospital would minimize legal and security concerns due to data locality, but it would impose paying a cost of having double copies of the data, both on their Hospital Information Systems and also the CHIC Clinical Data Repository. These scenarios must be examined in conjunction with concrete exploitation scenarios in order to draw useful conclusions.

7.1 Deployment model vs. Exploitation plan

In D12.4 we have compiled a list of possible exploitation plans of CHIC and besides exploitation of autonomous modules of CHIC, we explore the prospects of CHIC as a whole. These plans include:

- **Clinical exploitation**. CHIC platform could serve as a clinical support system in order to individualize the treatment scheme and schedule for each given patient, based on their own multiscale data.

- **Research exploitation**. The CHIC platform could be used as a platform for in silico experimentation in the generic biological and clinical research context (basic science exploitation). In this exploitation track, numerous in vitro and animal testing experiments are expected to be replaced with a less laboratory demanding and life-friendlier in silico experiments.
- **Educational exploitation**. The CHIC platform could be used as educational tool in the context of academic education (basic science, technology and medical education), general public education (patient's and/or parents' education, citizen's education and health literacy), industry education (education from an industrial perspective), politician education, epistemological, philosophical and social sciences education.

In some of these scenarios such as clinical and research exploitation, major organizations could afford having local CHIC installations, in a local private cloud, since there would be concrete benefits out of it.

In other scenarios, such as educational exploitation or occasional usage by users, we assume that a sustainability plan would be in place, such as StaRC, so that all partners take care to sustain and update the CHIC platform beyond the end of CHIC lifetime in order to provide access to the broader community. In such a plan, a combined deployment-operational model would be that StarRC provides DaaS services (Desktop as a Service) to users of the community so that they freely evaluate, e.g. with public data, or productively utilize with their own data the functionality of CHIC. This model could be sustained through a subscription based billing model, in addition to providing support to individual organizations.

Finally, a deployment model that could be used for educational or research exploitation is usage of a public cloud when no sensitive or private data are used. The public cloud hiring could be covered by subscriptions or donations.

Appendix 1 – Abbreviations and acronyms

AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
CD	Continuous Delivery
CI	Continuous Integration
CRAF	Clinical Research Application Framework
CRF	Case Report Form
CSS	Cascading StyleSheets
DCMTK	DICOM Toolkit
DICOM	Digital Imaging and Communications in Medicine
DLL	Dynamically Linked Library
GiB	Gibibyte, 2^{30} bytes
GB	Gigabyte, 10^9 bytes
HPC	High Performance Computing
HTTP	Hypertext Transfer Protocol
HTML	Hypertext Markup Language
IAM	Identity Access Management
IdP	Identity Provider
IP	Internet Protocol
ITK	Insight Registration and Segmentation Toolkit
JSON	Javascript Object Notation
LAN	Local Area Network
MIAME	Minimum information about a microarray experiment
NFW	Network Function Virtualization
MUSCLE	Multiscale Coupling Library and Environment
OWL	Web Ontology Language
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
SPARQL	SPARQL Query Language
SSH	Secure Shell

TB	Terabyte, 10^{12} bytes
URI	Uniform Resource Identifier
VLAN	Virtual Local Area Network
VM	Virtual Machine
VTK	Visualization Toolkit
XML	Extensible Markup Language