Grant Agreement Number: 257528

KHRESMOI

www.khresmoi.eu

State of the art, concepts and specification for the "Early Cloud infrastructure"

Deliverable number	*D6.4.1*
Dissemination level	*Public*
Delivery date	*14 November 2011*
Status	*Final*
Author(s)	*Ivan Martinez (AtoS), Miguel Angel Tinte (AtoS), Ana Juan Ferrer (AtoS), Francesco D'Andria (AtoS)*

This project is supported by the European Commission under the Information and Communication Technologies (ICT) Theme of the 7th Framework Programme for Research and Technological Development.

Executive summary

Deliverable D6.4.1 of the KHRESMOI project describes the approach that will be followed for the definition and deployment of the Early Cloud Prototype giving solutions satisfying the system scaling requirement.

The goal of the cloud prototype architecture is to provide a high-level overview of the necessary system components and their interactions in terms of composites, to provide understanding of the internal processes of KHRESMOI, and to support developers in providing components that can interoperate with the KHRESMOI system and its components.

Scaling the system depends on the components, processes involved, but primarily on the architecture design principles and their application in practice through the system integration. The success of the project depends heavily on the scale the resulting system is capable of covering. The task of scaling up the system will focus on iterative scale-up cycles involving evaluation and improvement of the key characteristics of the system with formal progress criteria.

The SOA-based software architecture described in deliverable D6.3.1 will be implemented for scaling up the system.

The Cloud approach proposed for KHRESMOI is based in two phase procedure: firstly we will define and deploy a cloud focus in hosting the SCA Composites and SCA Components definition (called Services Orchestration Cloud), and secondly we will update the Cloud deployed in the previous phase to host in the final Cloud (called Full Services Cloud) all the component defined in the project which are: adaptive User Interface, Orchestrated Services and Business Services.

Table of Contents

1 List of abbreviations

KIF	KHRESMOI Integration Framework
SOA	Service Oriented Architecture
SCA	Service Component Architecture
EAI	Enterprise Application Integration
ESB	Enterprise Service Bus
UP	Unified Process
SOMA	Service-Oriented Modeling Architecture
PaaS	Platform as a Service
SaaS	Softweare as a Service
IaaS	Infrastructure as a Service
OVF	Open Virtualization Format
VM	Virtual Machine
SLA	Service Level Agreements
GRAAP	Grid Resource Allocation Agreement Protocol
Guest OS	Guest Operative System
CC	Cloud Computing
REST	Representational State Transfer
OCCI	Open Cloud Computing Interface

Table 1: Abbreviations and acronyms

2 List of figures

3 Introduction

3.1 Introductory Explanation of the Deliverable

As the KHRESMOI system will manage a large amount of data it will have to be specified as a real Cloud Infrastructure. For the scaling problem, the physical resources and the platform configuration need to be taken into account. The Cloud Computing (CC) domain has these aspects under scrutiny to specify the optimized architecture that can dispatch the software tasks towards the available hardware resources.

Traditionally, the Cloud Computing approach consists of considering each part of the whole system as a service to facilitate full monitoring. Three architectural layers are described to decompose each resource as a service that needs to be monitored within a complex environment:

- Software as a Service (SaaS): this layer describes the software structure (components, interfaces, workflows), and is based on the SOA principles.
- Platform as a Service (PaaS): this layer defines the environment configuration required to enable the execution the software services. For example, Semantic Web Services (SWS) need a specific environment to be executed.
- Infrastructure as a Service (IaaS): this layer describes all the physical resources, their characteristics, and the conditions to be used for task processing.

Different virtualization solutions can be chosen to provide a platform abstraction that can be distributed according to the service requirements and the availability of hardware resources.

The service interoperability between these different levels has been formalized by the Web Service Level Agreement project that addresses service level management issues and challenges in a Web Services environment on SLA (Service Level Agreements) specification, creation and monitoring *(http://researchweb.watson.ibm.com/wsla/)*. Also, the Grid Resource Allocation Agreement Protocol (GRAAP) defines a formalism to permit agreements between services to enable large scale interoperability called Web Services Agreement Specification *(http://www.ogf.org/documents/GFD.107.pdf)*. This is a Web Services protocol for establishing agreement between two parties, such as between a service provider and consumer, using an extensible XML language for specifying the nature of the agreement, and agreement templates to facilitate discovery of compatible agreement parties.

3.2 Purpose and Audience

3.2.1 Purpose

The purpose of this deliverable is to provide a first version of the Early Cloud Prototype reference architecture. This document reports on the state of the art in cloud computing architecture and distributed processing techniques, leading to the first specification of the "Early Cloud Infrastructure". The main concepts to deploy the first cloud infrastructure are also described. The deliverable provides a comprehensive architectural overview of the Cloud platform, using various architectural views to depict different aspects of relevance. Following a pragmatic approach to cloud infrastructure specification, the cloud prototype architecture is described from the conceptual and technological point of view. The goal of the Cloud prototype architecture is to provide a high-level overview of the necessary system components and their interactions in terms of composites, to provide understanding

of the internal processes of KHRESMOI, and to support developers in providing components that can interoperate with the KHRESMOI system and its components.

3.2.2 Audience

This deliverable is relevant to all technical work packages in KHRESMOI (WP1-WP9). The target audience includes: component providers, users, and any person inside or outside of the KHRESMOI project interested in learning about the internal processing of the KHRESMOI Cloud platform. As such this deliverable presents the technological fundament, guidelines and technological details for the implementation of the KHRESMOI Cloud Runtime, the various components and the integration of the KHRESMOI project infrastructure as an integrated whole.

3.3 Structure of the Document

This deliverable is organized as follows: Section 4 describes the State of the Art on cloud computing architectures and distributed processing techniques. Section 5 provides a requirement analysis in order to collect the cloud requirements for the KHRESMOI system which are classified into hardware, infrastructure and software stack requirements. Section 6 provides the main concepts description for a better understanding of the KHRESMOI Cloud infrastructure. Section 7 describes the approach proposed for the specification and deployment of the first early cloud prototype and finally in Section 8, the conclusions of the deliverable are presented.

4 State of the Art on cloud computing architectures and distributed processing techniques

Cloud computing first emerged as Infrastructure as a Service, having the Amazon EC Services (EC2 and EC3) as its de facto figureheads. The evolution of the term is moving on to a more generic approach becoming an alternative delivery and acquisition model in which anything, and everything, can be offered as a service.

Cloud computing is the convergence of several trends in the past years, it joins a set of technologies and concepts that have been emerging over time: Software as a Service (SaaS), Grid computing, Virtualization, Utility computing and Hosting. Moreover, it is a change in IT users' behaviour; a user's centre of attention is on what the service offers rather that in how it is implemented or hosted, changing the focus from buying tools to enable a functionality, towards the contracting of a third-party to deliver this functionality in an elastic, on-demand, pay-per-use model. Of course, it is not new, Grid, SaaS and Utility models were already doing it, but it is a clearly a different approach than the classical on-premises, license based models.

Cloud Computing and its underlying "everything as a service" terminology refers to elastic Internet provision of resources or capabilities. The US National Institute of Standards and Technology[1] has provided the following definitions for the different elements on the Cloud Stack:

- *Cloud Software as a Service.* The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through a thin client interface such as a web browser (e.g., web-based email). The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.
- *Cloud Platform as a Service*. The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly application hosting environment configurations.
- *Cloud Infrastructure as a Service.* The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly limited control of select networking components (e.g., host firewalls).

Depending on the type of development of Clouds, they are also distinguished between:

- *Private cloud.* The cloud infrastructure is operated exclusively for an organization.
- *Community cloud.* The cloud infrastructure is shared by several organizations.
- *Public cloud.* The cloud infrastructure is made available to the general public commercially or not.

[1] http://csrc.nist.gov/groups/SNS/cloud-computing/

- *Hybrid cloud*. The cloud infrastructure is a composition of two or more previous cloud types (private, community, or public).

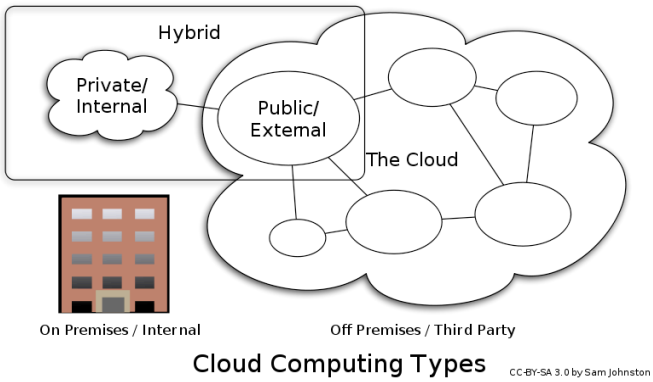


Figure 1. Cloud Computing Deployment Types1

In addition to these layered visions based on the type of products offered a more-business oriented classification can be provided by analysing the different flows of services and payments in the form of a generic value network for Cloud computing [14] (Figure 2):

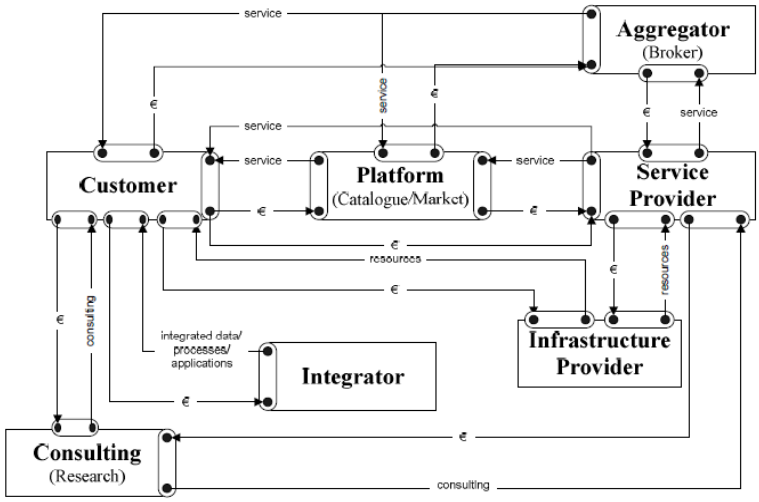


Figure 2. Generic Value Network for Cloud Computing

The following have been identified as potential benefits of Cloud computing:

- *Reduction of capital investment:* Given its outsourcing nature, Cloud converts IT into an operational expense, paying per use. Also, in the case of infrastructure provisioning, the risk of over provisioning or under-provisioning in the data center is reduced from a customer perspective.

[1] http://blog.nus.edu.sg/cloudcomputing/page/2/

- *Scalability on-demand:* The elastic capacity provided by Cloud Computing avoids forecasting on compute capacity or compute demand; it can be swiftly and on-demand adapted to business needs, with no need of over- or under-provisioning.
- *Lower Operating costs:* Cloud providers achieve economies of scale in their shared infrastructure management due to greater resource sharing, greater levels of architectural standardization and operation as well as a better consolidation. These benefits are passed along to the customers of these services that obtain significantly reduced prices in comparison to traditional offerings. In addition it has to be considered that the on-demand nature of the Cloud offering results in a linearly priced business model – as the use increases, costs scale directly, but without any increase in management complexity or any additional overhead.
- *Metered usage and billing:* Different to many outsourcing models based on a fee or a flat rate, Cloud computing has a transparent pay-per-use pricing model. It is not a recurring bill; it is based on the real consumption of the service, allowing a fine granular IT costs assessment.

Forrester in its report "Is Cloud computing ready for the enterprise?" [3] has identified currently the following typologies of users for the Cloud:

- *Start-Ups:* Given the Cloud's cheap infrastructure and low investment required. Used in: Web-based business, SaaS, collaboration services, widget providers, mobile services, social networking.
- *Entertainment industry, mainly gaming and entertainment providers:* Due to their need of highly scalable and temporary systems.
- *Small businesses:* For online businesses, online presence, collaboration and enterprise integration
- *Enterprises:* Used as a quick and cheap experimentation facility by R&D projects, quick promotions, widgets, online collaboration, partner integration, social networking and new business ventures.

The wider adoption of the Cloud model in Enterprise Environments, despite all previously identified benefits, still faces various challenges with no sufficient mature solutions:

- *Security:* Data security is the principal concern in the adoption of Cloud services. Many users only trust systems they have physical control over; systems with corporate firewalls, with known processes and audits; to outsource to any other model is not perceived as a secure model.
- *Regulations:* Some regulations require tracking, logging and auditing of enterprise data that for the moment are not offered by Cloud providers.
- *Reliability:* Nearly all public Cloud providers have suffered episodes of service level failure or unavailability. This severely concerns in enterprise environments, preferring not to outsource services where they lose control.
- *SLA Limitations:* Service Level Agreements provided by current public Cloud providers have very limited and inadequate SLAs for enterprise environments.
- *Existing investments:* Already made investments mean that many companies are not prepared to abandon current systems and outsource to Cloud providers.

According to Gartner [4], during the past 15 years, a continuing trend toward IT industrialization has grown in popularity. IT services delivered via hardware, software and people have become repeatable and usable by a wide range of customers and service providers. This is due to the standardization and commoditization of technologies, virtualization and the rise of service-oriented software architectures, and most importantly to the dramatic growth in popularity of the use of the Internet. These things, all together, constitute a new opportunity to shape the relationship between IT service sellers and vendors

– moving from license-based, on-premises models, long dominant in the IT industry, to "elastic" models represented by Cloud computing. Although this is not a completely new phenomenon; utility computing, Software-as-a-Service and application service providers (ASPs) had their momentum; it is settling the basis for the movement from an Internet seen as a "communications channel" to an approach based on "the deliberate delivery of services" over the Internet. Focussing in outsourcing strategies, again, the shift to cloud services can be considered more evolutionary than revolutionary, being part of an externalisation tendency over last decades. Customers started with fully-customized internal owned and operated services, followed by external provision of IT-Services through outsourcing activities, and finally embracing the emergence of Cloud-based services whereby they buy specific horizontal IT functionality (computing, storage and applications) as a commodity-like service, or fully functional standardized services (payment services, sales, CRM, …).

Other visions, such as the one provided by the Economist in their special report on Corporate IT [5], go further, suggesting that the emergence of Cloud computing could change businesses and the economy. If IT services really allow companies to become more modular and flexible, this should foster further specialization. It could become even easier to outsource business processes, or at least, those parts that do not constitute a competitive advantage, distinguishing between the core processes and the context. This would also mean that companies will rely more on services provided by others, forming increasingly "process" networks, a term for loosely connected groupings of specialized firms. Both trends could result in "huge clouds" that provide basic services for a particular sector and, on top of these systems, many specialized and interconnected firms could construct their services.

4.1 Infrastructure Cloud Middleware and on-going initiatives

The previous section provided a general overview of Cloud Computing. From now on, the analysis will focus exclusively on the IaaS layer of Cloud computing, as it the closest topic to the project developments. In order to do this, Section 4.1.1 will introduce Server Virtualization Technologies as groundwork for IaaS, Sections 4.1.2 and 4.1.3 provide examples of server virtualization technologies next to other initiatives and finally Section 4.1.4 will present mechanisms and tools for Infrastructure Cloud Management Platforms.

4.1.1 Server Virtualization Technologies

Server virtualisation is a software technique that allows perceiving a single physical server as multiple virtual machines (VMs). A VM is then a software partition created within a physical server such that the guest operating system and application run almost exactly as if they were running on a physical machine itself. The piece of software that performs this software abstraction is the hypervisor, or VM Monitor, the software in between the physical server and the VMs.

Server Virtualisation is not in any way a modern technique; it has been extensively used during 60's and 70's in order to share mainframe computers among several user groups that could run different operating systems on unique, and usually very expensive, hardware. In late 90's this technique gained the attention of the industry by the appearance of VMware with x86 virtualisation, and its potential for server consolidation, fault containment, security and resource management. This lead to a significant change in Data Center management, by breaking the 1:1 association between physical servers and applications and becoming the technology foundation for Cloud computing (IaaS) developments.

Usually three different types of Server Virtualization techniques, or hypervisor types, are considered:

- *Full virtualization:* Software that aims to provide the virtual machine with an interface that is identical to that of the underlying hardware.

- *Para-virtualization:* Software that aims to provide the virtual machine with an interface that is similar to that of the underlying hardware, so that modifications to the guest operating system are required in order to gain improved performance.

  Hardware-assisted virtualization offers new instructions to support direct calls by a para-virtualized guest OS into the hypervisor.

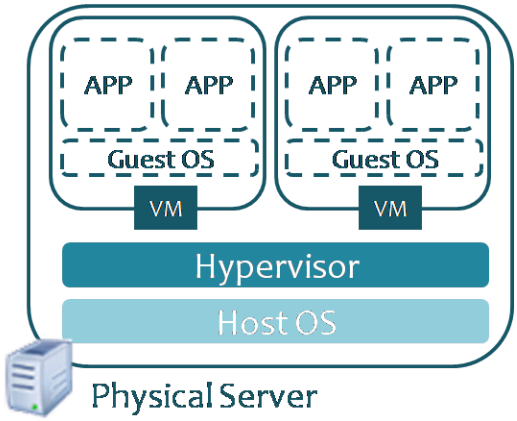


Figure 3. Full Virtualization and Para Virtualization schema

- Native or Bare-metal virtualization: Software that does not require of the presence of an underlying operating system, but it is instead directly controlling the CPU.

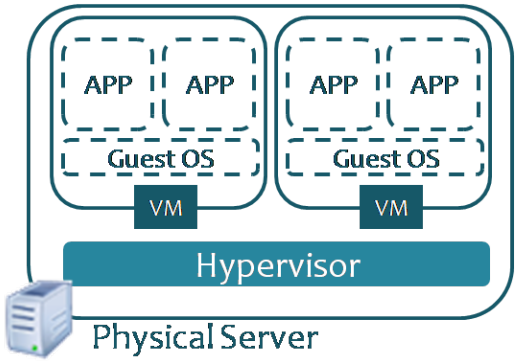


Figure 4. Native Virtualization schema

4.1.2 Examples of Server Virtualization Technologies

4.1.2.1 Xen

Hypervisor	Xen
Description	Xen [6] is an open-source hypervisor based on para-virtualization techniques. Xen originated as a research project at the University of Cambridge in 2003, and was acquired by Citrix systems in 2007. Xen is released under the GNU General Public License (GPL2).
Host OS	NetBSD, Linux, Solaris
Supported OS	Xen is based in para-virtualization, this means the guest OS has to be modified in order to "collaborate" with the hypervisor. It is supported by specific systems on windows and other OSs such as NetBSD, FreeBSD, and OpenSolaris.
VM Format	Own. OVF Supported.

4.1.2.2 KVM

Hypervisor	KVM
Description	Kernel-based Virtual Machine (KVM) [7] is based on a Linux kernel virtualization infrastructure. It is based on native virtualization. It is released under the GPL license.
Host OS	Linux, Windows, FreeBSD, Solaris: Detailed list available in [http://www.linux-kvm.org/page/Guest_Support_Status]
Supported OS	Same as host
VM Format	Own. OVF not supported for KVM. But **Red Hat Enterprise Virtualization Hypervisor** based on KVM in version 5 and later supports OVF format.

4.1.2.3 VirtualBox

Hypervisor	VirtualBox
Description	VirtualBox [8] is based on full virtualization. VirtualBox Open Source Edition is released under the GPL 2 license.
Host OS	Windows, Linux, Mac OS X (Intel), Solaris, FreeBSD, eComStation
Supported OS	DOS, Linux, Mac OS X Server, FreeBSD, Haiku, OS/2, Solaris, Syllable, Windows
VM Format	Own. OVF Supported.

4.1.2.4 Vmware

Hypervisor	VMware
Description	VMware [9] is a pioneer in x86 server virtualization. It is based on full virtualization. Currently it covers a wide range of products that goes from pure virtualization to cloud management. Although it offers free versions of its products, nowadays it is the most used commercial solution for virtualization.
Host OS	VMware ESX Server, VMware ESXi: No host OS VMware Fusion: Mac OS X Server VMware Server, VMware Workstation, VM Ware Player: Linux, Windows
Supported OS	DOS, Linux, Mac OS X Server, FreeBSD, Haiku, OS/2, Solaris, Syllable, Windows
VM Format	Own. OVF Format supported.

4.1.3 Other related initiatives

4.1.3.1 Libvirt

Libvirt [10], [11] is hypervisor-agnostic API that is able to manage multiple guest OSs running on a physical server. It provides a common interface to capabilities that all hypervisors implement, simplifying the task of managing heterogeneous hypervisors in a computing cluster such as a private cloud.

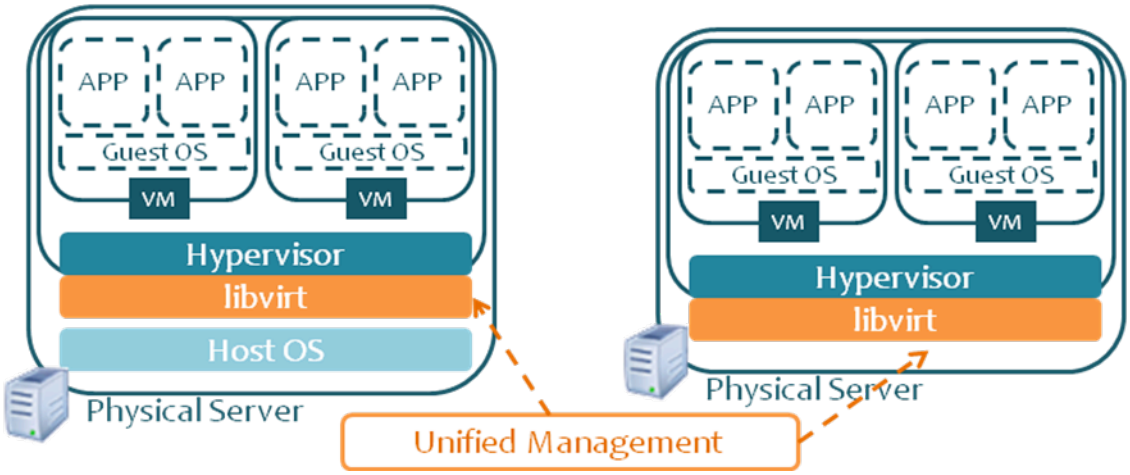


Figure 5. Libvirt Virtualization Schema

Libvirt relies on the hypervisor in order to manage VMs in a physical node by means of the use of specific drivers. Currently there are available drivers, among others, for the following hypervisors:

- Xen
- KVM Linux
- VirtualBox
- VMware ESX

Libvirt provides APIs to monitor and manage resources available on the physical server for each of its created domains, guest OSs, including CPUs, memory, storage and networking.

It enables Virtual Machine Monitors, such as Open Nebula and Eucalyptus, to manage multiple hypervisors transparently, given that it provides a unified interface for these common hypervisor capabilities.

4.1.3.2 OVF

Open Virtualization Format (OVF) [12] is defined as a 'hypervisor neutral standard for describing, packaging and distributing virtual appliances'. A virtual appliance [13] is defined as 'a pre-configured software stack comprising one or more virtual machines'.

OVF is a DMTF standard that is currently in version 1.1.0, released in January 2010. OVF was developed by the System Virtualization, Partitioning, and Clustering Working Group with participation of many industry representatives such as VMWare, XenSource , IBM, HP, Dell and HP.

OVF Specification 1.1.0 defines an OVF package as:

- one OVF descriptor with extension .ovf
- zero or one OVF manifest with extension .mf
- zero or one OVF certificate with extension .cert
- zero or more disk image files
- zero or more additional resource files, such as ISO images

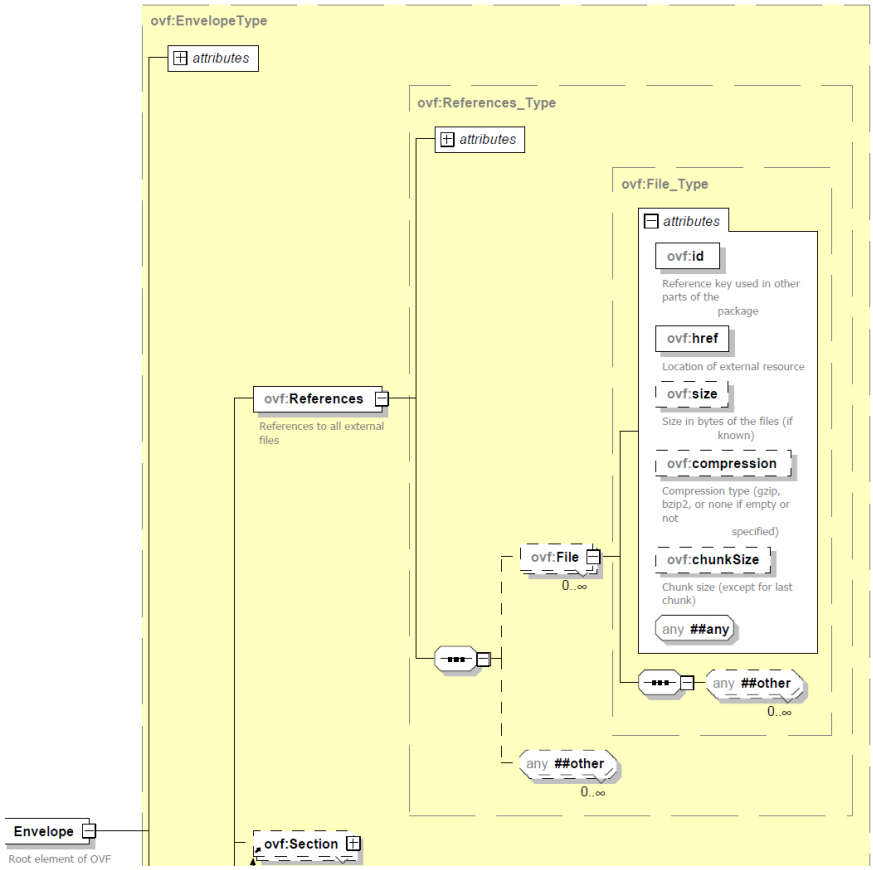
An OVF package can be stored as this set of files or as a single tar file stored with .ova extension.

OVF Descriptor

The OVF descriptor is an XML file that follows the dsp8023_1.1.0.xsd[1] XML schema. It defines the content and requirements of the packaged virtual appliance.

The main element of the schema is the Envelope. The main elements of the envelope are:

- *List of References:* The list of the rest of files that are part of the OVF package. Typically they are virtual disk files, ISO images, and internationalization resources.

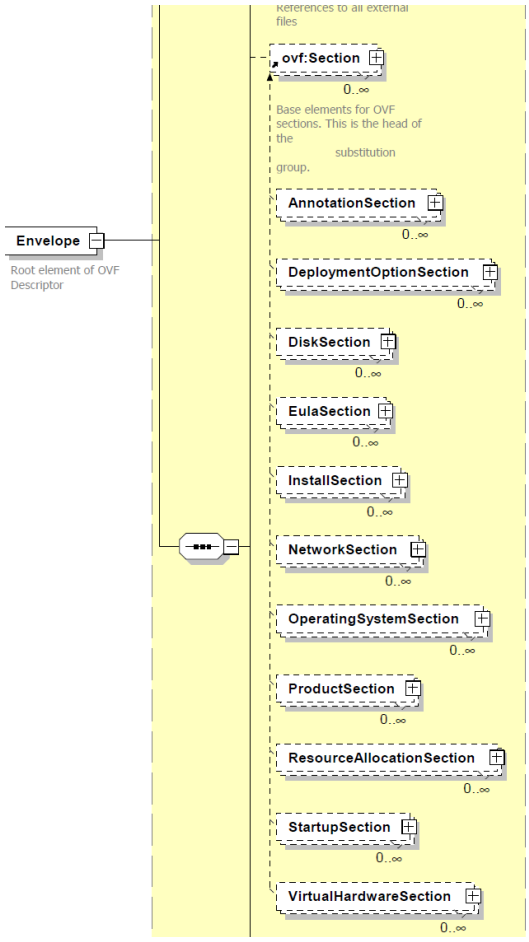


- *Section Elements* provide meta-data information for the virtual appliance components in the OVF package. They include information such as:
  - DiskSection, describing meta-information about all virtual disks in the OVF package (e.g., capacity).
  - NetworkSection, describing meta-information about all logical network used in the OVF package. Only names and descriptions can be specified.
  - ResourceAllocationSection, specifying resource reservations (e.g., CPU, memory, virtualization technology, etc.) to perform at deployment time for virtual machines.
  - ProductSection, specifying product information for software packages in virtual machines. Apart from information regarding the software package itself (e.g., vendor,

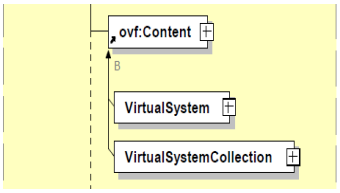
[1] http://schemas.dmtf.org/ovf/envelope/1/dsp8023_1.1.0.xsd

product name, version, etc.) it also allows specifying customization parameters to help configuring the application automatically at deployment time (e.g., DNS servers, gateways, etc.).

- StartupSection, allowing specifying the powering on (off) sequence order for virtual machines that compose the application.
- DeploymentOptionSection, allowing the specification of different configuration options, each one corresponding to different sets of virtual hardware.
- OperatingSystemSection, specifying the virtual machine operating system.
- InstallSection, specifying that the virtual machine has to be booted as part of the deployment process to install and configure software



- *Content:* A description of the content. It can be either a single virtual machine (VirtualSystem element) or a collection of multiple virtual machines (VirtualSystemCollection element).



OVF allows having a portable Virtual Appliance format among multiple hypervisors. In order to execute the virtual appliance defined in a package for each hypervisor it is required to convert this portable format to the specific hypervisor's file format. Currently there are available conversion tools for the following hypervisors: VMWare, XenServer and VirtualBox.

4.1.4 Infrastructure Cloud Management Platforms

Generally speaking, an Infrastructure Cloud Management platform is a software platform with the main characteristic that it provides tools and mechanisms for managing virtual machines in a pool of physical resources, allowing the creation of private Infrastructure Clouds. Several examples of these types of software can be found on the market, both as proprietary solutions, such as VMWare vSphere [15], Ovirt [16] or Platform VM Orchestrator [17], and Open source, such as Open Nebula [18], Globus Nimbus [19] or Eucalyptus [20], although they differ substantially in the rest of capabilities they offer: support for hybrid clouds, types of interfaces offered, server consolidation policies, …

In this analysis we will focus on the capabilities offered by two open source alternatives: Eucalyptus and Open Nebula.

Eucalyptus:

Elastic Utility Computing Architecture Linking Your Programs to Useful Systems (Eucalyptus) [21], [22], [23] was initially developed by University of California. It is an open source tool for developing an IaaS Cloud.

The Architecture of Eucalyptus is the following:

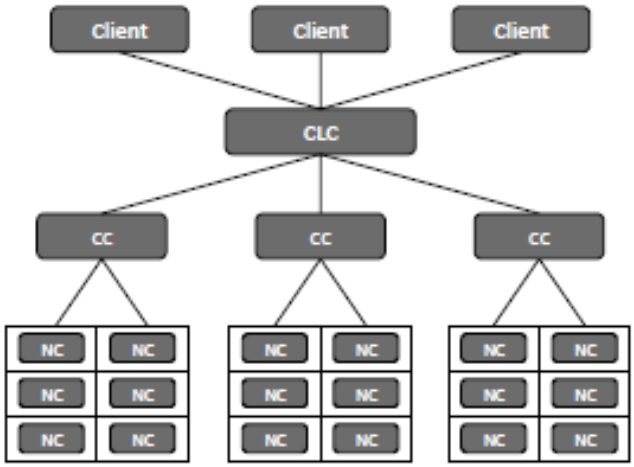


Figure 6 Eucalyptus Architecture

- *NC: Node Controller:* Runs on a physical host. It is in charge of controlling the execution, inspection and termination of VMs in the physical host though interfacing with the Node's operating system or hypervisor instance. Currently it supports Xen and KVM as underlying virtualization platforms.
- *CC: Cluster Controller:* It aggregates at cluster level the information provided by NCs in the cluster. In addition it manages the different Network configurations among different VMs that constitute a virtual appliance.

- *CLC: Cloud Controller:* It is the interface provided by the private/community cloud. It provides three types of services:
    - Resource Services: Managing resource and network allocation.
    - Data Services: Implemented through Walrus, it provides S3 interfaces for managing data sources between the private cloud and the user and among VMs in the Cloud.
    - Interface Services: User authentication and REST Cloud interfaces.

A summary of Eucalyptus is given below:

Cloud Management Platform	Eucalyptus
Deployment scenarios	Private + Community Cloud
Hypervisors	Xen, KVM
Interfaces	EC2 WS and S3 API
Storage management	Yes
Network Management	Yes
Configurable placement policies	No
OVF Support	Depending on the hypervisor
OCCI Support	No

Open Nebula

Open Nebula [24], [25] (ONE) is developed by Universidad Complutense de Madrid. It is defined as 'a virtual infrastructure manager that can be used to deploy and manage virtual machines, either individually or in groups that must be co-scheduled, on local resources or on external public clouds, automating the setup of the virtual machines (preparing disk images, setting up networking, etc.) regardless of the underlying virtualization layer (Xen, KVM, or VMWare are currently supported) or external cloud (EC2and ElasticHosts are currently supported)'.

The Open Nebula architecture is depicted in the following picture:

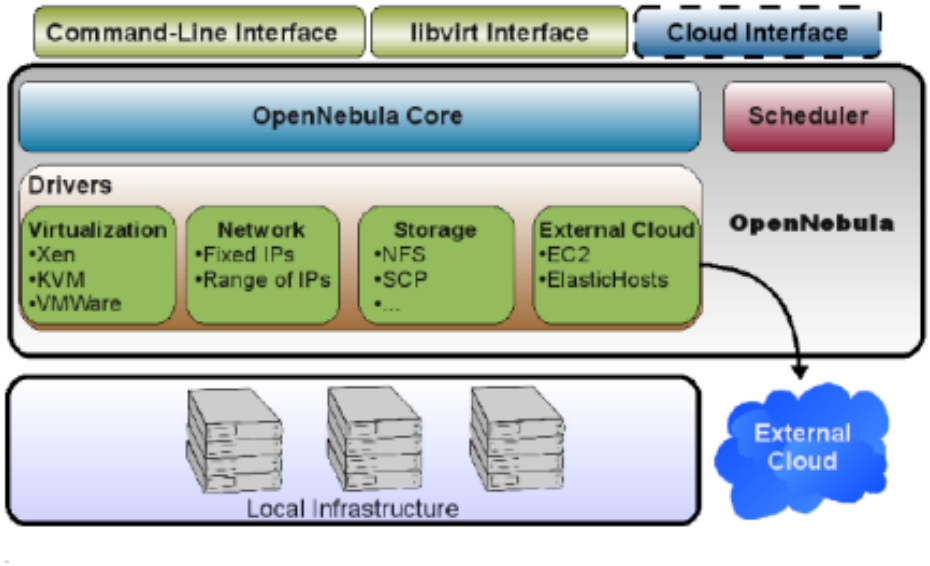


Figure 7 Open Nebula Architecture

A ONE cluster is typically formed by an administrator node, which hosts the scheduler and core functionalities, and a set of worker nodes, that can simultaneously have diverse hypervisor technologies.

The default Open Nebula Scheduler is based on a ranking algorithm highly configurable by cloud administrators, it uses monitoring information from running VMs and physical resources in order to decide on VM placement. Decisions are communicated to Core components that implement them. Additionally the scheduler is able to manage a higher abstraction level, called service, virtual appliance, in order to place together all VMs that constitute it.

The Core manages different drivers enabling three types of capabilities:

- Image and storage capabilities: managing hypervisors to create and control VMs, as well as node storage through distributed file systems.
- Network fabric: Managing network configurations among VMs.
- External Cloud drivers: Act as link to public Cloud offerings, moving to them part of the load in peak demand situations.

A summary of Open Nebula is given in the following table:

Cloud Management Platform	Open Nebula
Deployment scenarios	Private + Hybrid Cloud
Hypervisors	Xen, KVM, VMware
Interfaces	OCCI, CLI(own), libvirt
Storage management	Yes (not S3 interface)
Network Management	Yes
Configurable placement policies	Yes
OVF Support	Depending on the hypervisor
OCCI Support	Yes

4.1.4.1 Standardization Activities for Infrastructure Cloud Management

Open Cloud Computing Interface OGF group aims to provide an API specification for remote management of cloud computing infrastructure. It intends to provide a common interface covering high level functionality for life-cycle management of virtual machines.

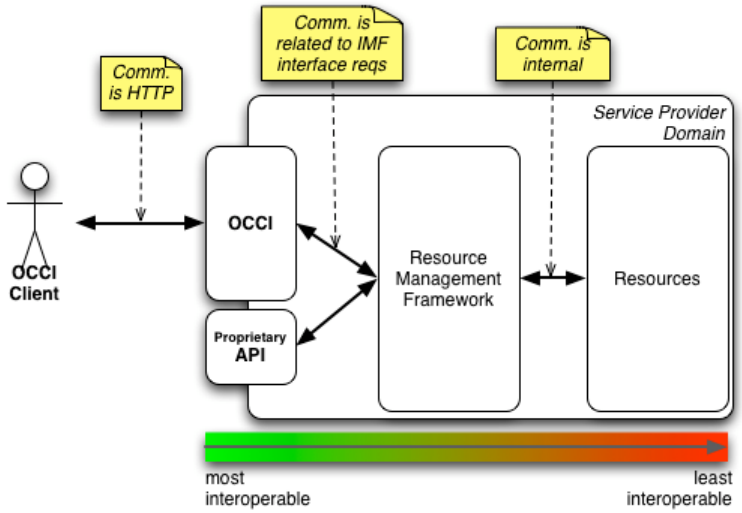


Figure 8. OCCI Interoperability

The specification is not yet an official standard. It is a draft that was released for public consultation in September 2009 by the OGF (Open Grid Forum).

It proposes a REST (Representational State Transfer) interface over HTTP. It currently considers the following operations [26]:

Operation	Description
POST (Create)	"The POST method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line."RFC2616
GET (Retrieve – Metadata and Entity)	"The GET method means retrieving whatever information (in the form of an entity) is identified by the Request-URI."RFC2616
PUT (Create or Update)	"The PUT method requests that the enclosed entity be stored under the supplied Request-URI."RFC2616
DELETE (Delete)	"The DELETE method requests that the origin server delete the resource identified by the Request-URI."RFC2616
COPY (Duplicate)	"The COPY method creates a duplicate of the source resource identified by the Request-URI, in the destination resource identified by the URI in the Destination header."RFC4918
HEAD (Retrieve – Metadata Only)	"The HEAD method is identical to GET except that the server MUST NOT return a message-body in the response."RFC2616
MOVE (Relocate)	"The MOVE operation on a non-collection resource is the logical equivalent of a copy (COPY), followed by consistency maintenance processing, followed by a delete of the source, where all three actions are performed in a single operation."RFC4918
OPTIONS	"The OPTIONS method represents a request for information about the communication options available on the request/response chain identified by the Request-URI."RFC2616

5 KHRESMOI Cloud Requirements

This section reports a short survey that tries to collect data from the partners that need to deploy some virtual machines in the scope of the project to determine which architecture and components (i.e: cloud manager, hypervisor and monitoring) could fit best in the KHRESMOI cloud approach.

5.1 Hardware requirements

- Will you need to define some specific hardware requirements?

UDE: YES
HES-SO: Yes, we will most likely rely on one or two servers with much RAM for testing configurations and optimize results
ONTO: YES (semantic database requires lots of RAM memory, 30GB of RAM for each 5 Billion RDF statements)
CUNI: NO
MUW: NO but they make intensive use of CPU and RAM (SAN file system?)
HON: NO but they need 4xCPUs above 3.0 GHz 2x2HDD 32 GB RAM. Not sure if they can host their own components
USFD: NO

- If YES: Are these special requirements different from hard disk size, number of CPUs, RAM amount and/or network interfaces?

UDE: NO
HES-SO: NO (but they mentioned bandwidth)
ONTO: YES

  - If YES: Can you describe these special requirements please?

ONTO: OWLIM SE (former BigOWLIM) requires disks with low seek latency. Could be used with SATA disks but SSD disk would be the best option.

- Do you own the appropriate hardware to deploy your required VMs?

UDE: NO
HES-SO: They have the hardware but some software is not virtualized.
ONTO: It seems so and no specific requirements to join the project infrastructure (to be confirmed)
CUNI: They can provide a server with the software. They prefer this to share a VM
MUW: They can host their own testing environment but they want to host services on our infrastructure
USFD: YES

- If YES: Would you allow KHRESMOI to include your hardware on the general cloud architecture or do you prefer to keep it apart from the project facility?

HES-SO: Only a part
USFD: NO
CUNI: NO

  - If YES: Which security limitations have your infrastructure to allow third parties to access it? Are there any other limitations to use your hardware?

HES-**SO**: Ports blocked, some servers are not in the DMZ (external access not allowed)
CUNI: University firewall

5.2 Infrastructure requirements

- Would you need VM images to customize?

UDE: NO
ONTO: They prefer to work on already built VMs
USFD: No, Mimir is Java and they can easily install on any standard Linux image
MUW: They are waiting for our decision to start making VMs

- If YES: Would you need to store these customized templates?

HES-SO: They need to install custom libraries
ONTO: Depending on the base image. They mentioned Java a servlet container and the KHRESMOI integration system (Tuscany SCA).

- Would you need to upload your own VM images?

USFD: NO

- If YES: How do you prefer to upload your templates (FTP, SCP, NFS...)?

HES-SO: FTP (opened to open standards)
ONTO: They prefer to work on already installed VMs.

- Have you already chosen the hypervisor (virtualization manager) where your VMs are going to be created?

UDE: NO, but they use Xen
HES-SO: VMWare but this may change if necessary
ONTO: They use VMware ESXI 3.5/4 but they are changing to LXC but they prefer to customize a VM template
CUNI: They prefer to use KVM or Virtualbox

- Does your software depend on special network configurations? Please, explain if it's the case.

UDE: NO
HES-SO: Seems like they didn't but they will check
ONTO: NO
USFD: NO
CUNI: NO (They use only TCP connections)
MUW: NO

- Which is the average hard disk size that your VMs will require?

UDE: 2-10GB and more space for the repository
HES-SO: Depends on the task. 1TB for large DB but if they use the Vienna Db they will need 5TB or more
ONTO: 600-700GB seems reasonable
USFD: They cannot calculate it maybe 1.5 TB is fine
CUNI: Tens of GB. They offered to host their own services
MUW: SAN file system 1TB+

- Is it possible that your software components require extra storage capacity once the VMs are up and running?

UDE: YES
HES-SO: YES (based on the supposition that we are going to add articles everyday)
ONTO: Highly probable
USFD: YES
CUNI: This can happens in a 20-30% of cases
MUW: Covered by the SAN filesystem

- How many VMs do you think that you will need to be running at the same time?

UDE: More than 14 (dccp, user, log, query history, search, repository, term info, directory and broker, 3+ message transfer agent, 3+ wrappers –two last ones scales with users).
HES-SO: Min 2. One for the production environment and another one for testing and optimization.
ONTO: Two VMs (production/development) but it seems like they are going to host their own development environment
USFD: Don't know
CUNI: One or two
MUW: Multi-street solution

- Are all VMs going to be replicas of a base image or are they going to be completely different from each other?

UDE: One base image.
HES-SO: There could be different images
USFD: Seems like they want to install their sw all the time
CUNI: One base image (Debian preferred)

- Can you provide some extra information about these VMs? For example: how many CPUs or which amount of RAM will them need?

UDE: More than 4GB of RAM
HES-SO: Their current server for parameter testing is not a VM and contains 96 GB RAM, 2 TB disk, 12 processor cores; their virtual machines rather contain 4GB RAM, 1 TB disk and 4 processor cores at the moment. Over the time of the project this should increase
CUNI: 8GB RAM minimum and 2CPU min also

- Do you need to monitor infrastructure parameters (not internal VMs parameters)?

HES-SO: RAM usage and free disk space (MUST)
USFD: NO
CUNI: NO requirements
MUW: They said that they are going to write standard log file but it must be done inside VMs

5.3 Software stack requirements

- Do you need to use a different OS from Linux? Do you need to use some MS OSes? Please, think carefully about it because it affects in deep the basic infrastructure design.

UDE: They would like to use Debian based images
HES-SO: LINUX is ok
ONTO: Linux is OK (Ubuntu is preferred)
USFD: NO (Linux is fine)

CUNI: Debian (MUST)
MUW: NO, Linux is fine
HON: They need a SOLARIS VM

- Which software would you like to find pre-installed on the VM templates provided by the KHRESMOI cloud?

UDE: SUN Java 1.7+ and MYSQL 5.x (server/client)
HES-SO: GCC, GIFT, PERL (modules), itk/vtk. Matlab per concrete processes.
ONTO: X Windows installed (GUI)
USFD: GNU text tools, Java, emacs
CUNI: Python, Perl, GCC
MUW: WebApp server and Matlab components
HON: Java, PERL

- Do you need to monitor internal VM parameters?

UDE: YES, CPU load, free HD space, thread count, DB connections, IO load, network load
HES-SO: Probably
ONTO: Web server
USFD: NO
CUNI: NO
MUW: NO (see infrastructure answer)

- If YES: Do you think that it is important to have the same monitoring software on the infrastructure and on the VMs?

UDE: YES
HES-SO: YES
ONTO: NO

- Which monitoring software do you prefer?

UDE: Nagios (but they are opened to other options)
HES-SO: They don't know any

- Does your application or some required libraries have special licensing issues that must be taken into account?

UDE: NO
HES-SO: They are worried about the Open Source restrictive licences and the Matlab professional license.
ONTO: NO
USFD: NO
CUNI: GPLed software
MUW: They need to investigate this further. It is probable

6 Concepts of the KHRESMOI Cloud Infrastructure

Scaling the KHRESMOI system depends on the components, processes involved, but primarily on the architecture design principles and their application in practice through the system integration. The success of the project depends heavily on the scale the resulting system is capable of covering. The task of scaling up the KHRESMOI system will focus on iterative scale-up cycles involving evaluation and improvement of the key characteristics of the system with formal progress criteria.

The SOA-based software architecture described in deliverable D6.3.1 will be implemented for scaling up the system. The cloud computing (CC) environment will provide the following:

- description of the Hardware resources (CPU resources, IP, storage, software environment, etc.),
- API to interact with the CC environment (ask for resources, trigger services), and
- execution environment to dispatch and trigger services.

This work will be divided into three tasks related to the main services layers:

- IaaS (Infrastructure as a Service): virtualization implementation, service specification,
- PaaS (Platform as a Service): different platform configurations (linguistic, semantic, etc.),
- SaaS (Software as a Service): integration of the software prototype implemented in T6.2.

The CC environment will provide:

- scalability: the capability to trigger many tasks together, and dispatch them on the cloud,
- elasticity: the capability to increase the task resources according user needs,
- availability on demand: the capability to always provide resources to the users.

The SCA approach chosen as integration solution permits to add the Service description around the existing components without imposing a component refactoring. The benefit of this approach consists of easily deploying the services and testing the possible composition between them. For example, the four integrated prototypes described in D6.3.1 will be deployed and tested with this approach. Moreover, the new functionalities can be specified quickly to be in concordance with the basic architecture. Finally, the result of the combination of the different integrated prototypes will be the full cloud prototype.

As it was decided in the previous deliverable D6.3.1, Apache Tuscany is the most relevant SCA runtime that could deploy easily the SCA implementation. In particular, it provides the Tuscany Domain manager that permits to configure server nodes where several composites will be deployed.

Then, different configurations of composites can be deployed and tested to evaluate their efficacy. The system can be deployed locally (all composites are deployed on one local server), but also can be deployed in a distributable way (several composites deployed on several servers). The second one will be the way chosen to deploy the KHRESMOI Cloud Prototype, the main objective during year 2.

SCA simplifies the composite deployment and is completely compliant with the requirements of any cloud infrastructure.

6.1 Tuscany runtime environment

In this section we will explain what the domain and node are, what information needs to be configured for a node, and how the hosting environment controls the node and interacts with other nodes in the same SCA domain.

6.1.1 The SCA domain and Tuscany nodes

An SCA domain manages a set of composite applications that can be connected using SCA wires and constrained by the same policy definitions. An SCA domain consists of the following:

- A virtual domain-level composite (that is, a domain composite) whose components are deployed and running
- A set of installed contributions that contain composites, implementations, interfaces, and other artifacts

An SCA domain represents a complete configuration of the SCA assembly. The components within an SCA domain can be hosted by one or more runtime environments that provide the necessary technology infrastructure. Tuscany introduces the node concept to represent a group of components (typically declared in the same composite) that can be started, executed, and stopped together on a Tuscany runtime. Figure 9 illustrates a KHRESMOI definition of such mapping. The composites are part of the KHRESMOI scenarios based on the use cases defined in WP8 and WP9.

In this cloud infrastructure proposed for the KHRESMOI system, three Tuscany nodes are used to run the applications. Node1 is running on one JVM (JVM1) and hosts the components from the *SCATextualSearch* composite. Node2 is started on a different JVM (JVM2) to run components from the *SCAImageSearch* composite. Finally Node3 is started in JVM3 to run components from the *AdvancedSearch* composite. Tuscany is quite flexible in how it maps the domain composite to different node topologies. For example, you can run all three composites on one single node or run each of the composites on a different node. These mappings can be described in the Tuscany node configuration.

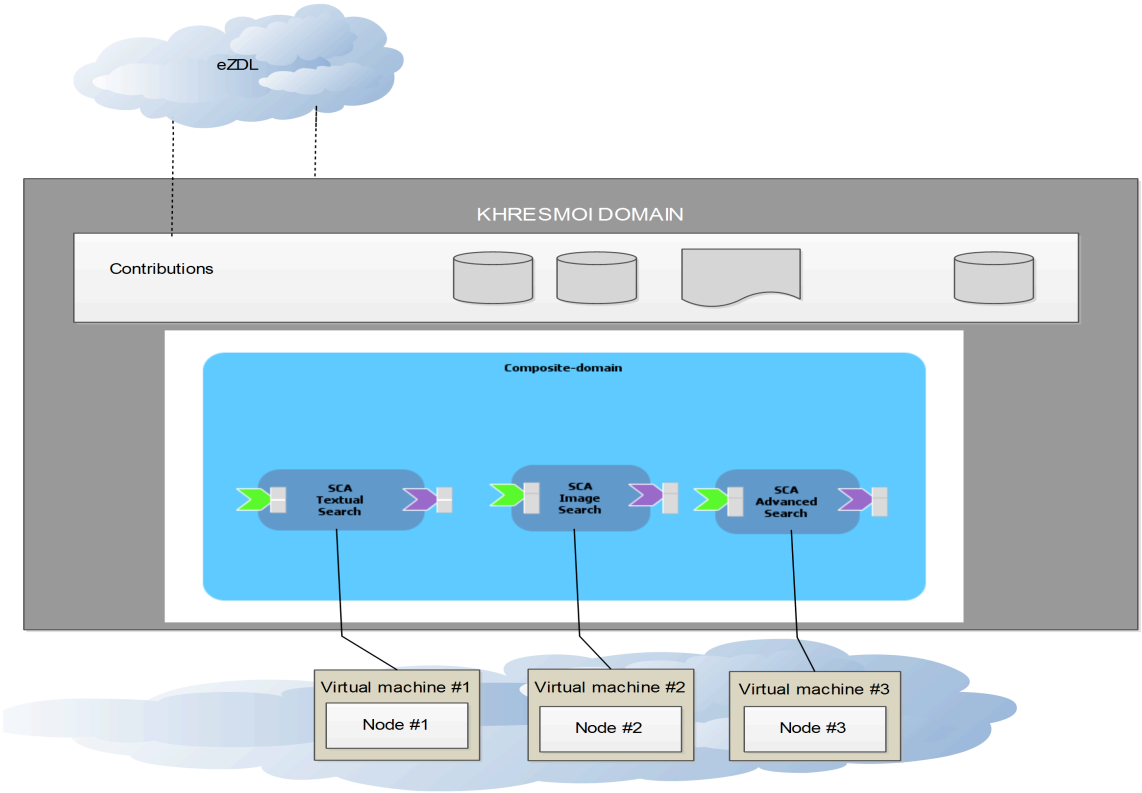


Figure 9. KHRESMOI SCA Composite applications managed by the KHRESMOI SCA Domain

6.1.2 Tuscany node configuration

A Tuscany node is configured to run a composite application, which could be the entire SCA domain composite or a subset of this composite. The configuration captures the following information:

- A collection of contributions that are required to run the components on the node
- One or more deployable composites that describe the components
- A list of binding base URIs that tells Tuscany what addresses are used to publish the service endpoints

To map an SCA domain to Tuscany nodes, the domain composite can be partitioned into one or more groups of components (denoted by composites) that are suitable to be run together. Each group of composites is configured as a Tuscany node.

SCA components typically depend on artifacts from SCA contributions such as the interface, Java implementation class, or XML schema. A list of installed contributions will be selected from the SCA domain for a node.

The default configuration can be also specified for the communication infrastructure that a node provides. For example, a service with *binding.rest* can be published on one node as *http://node1:8080/component1/service1/binding1* and on another node as *http://node2:8085/component1/service1/binding1*. The node configuration defines which HTTP ports to use by default.

The node configuration can be manually defined, automatically discovered, or provided by the domain manager online or offline, depending on how the node is run.

6.1.3 Hosting options for a Tuscany node

The hosting environment for Tuscany controls how to configure, create, start, and stop a node. Each environment uses various approaches to handle the following aspects:

- How to create a node configuration
  - From a configuration file
  - From the domain manager
  - From a list of command-line arguments
  - From what's in the environment (classpath, repository, and so on)
- How to control the lifecycle of a node
  - Start/stop the node using a launcher
  - Start/stop the node explicitly within the embedding application
  - Start/stop the node from a lifecycle listener of the host environment, such as the *ServletFilter* or *ServletContextListener* or the *OSGi Bundle Activator*, *Service Listener*, or *Bundle Listener*
- Are there SCA wires that go across components that are hosted on different nodes?
  - Components in the node access services only in the same node
  - Components that are wired by SCA run on different nodes
- How to find endpoint descriptions of other SCA services in the same domain but running on a different node

- From a domain registry
- From a local file

- How to initiate component interactions
  - Access the SCA services from a remote client
  - Access the SCA services locally
  - From a component that's early initialized
  - From a launcher that knows the client component

These options are illustrated in Figure 10.

As shown in Figure 10, the SCA assembly represented by the domain composite (A) can be partitioned into groups of components. This is typically done by using one or more composites that define the components. You can also use a deployment composite to regroup the components. Each group will be hosted by a Tuscany node (F). You'll define the domain/node mapping using the node configurations (B). You can bootstrap the Tuscany runtime in different environments (D) to provide the infrastructure (E) for one or more nodes. Depending on the hosting environment, you can provide the configuration for a node using different mechanisms (C) and then create and start nodes to run the components. Tuscany provides the *SCANodeFactory* and *SCANode* APIs so that different hosting environments can use them to create a *SCANode* from a node configuration and start or stop the *SCANode* to run the composite application. Each form of hosting option for Tuscany ultimately calls the *SCANodeFactory* and *SCANode* APIs.

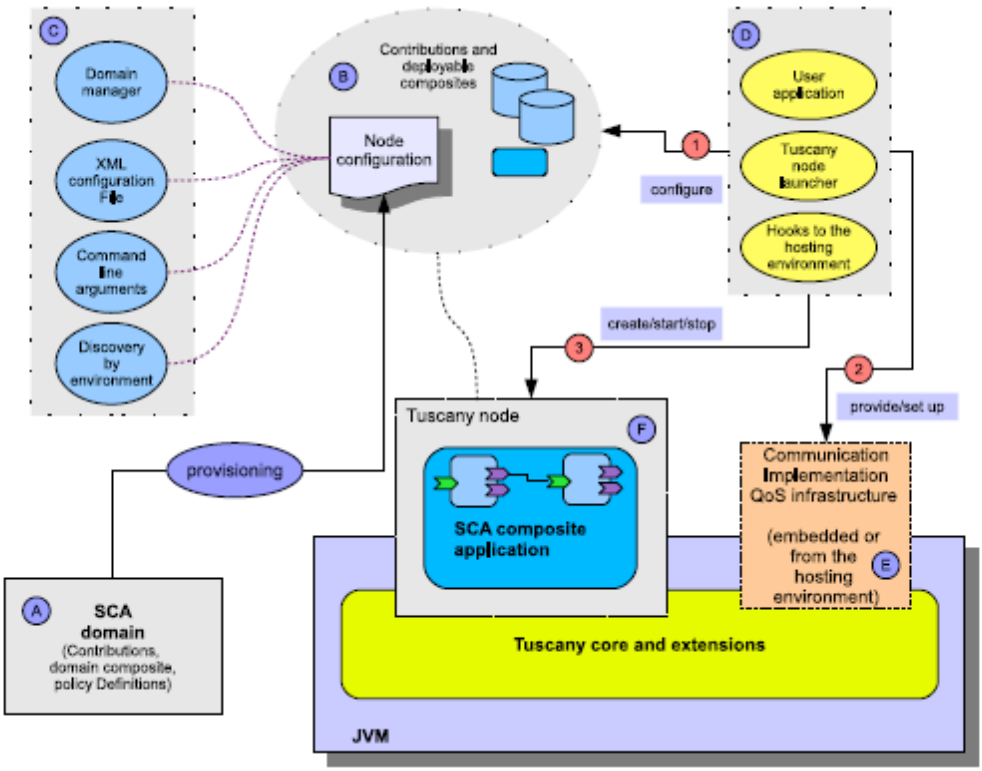


Figure 10. Hosting options for a Tuscany node

6.2 Running Tuscany in the cloud

Cloud computing is an internet-based development model where users rely on the cloud platform, or cloud software provider, to run their business applications. Cloud computing typically means the provisioning of dynamically scalable and often virtualized resources as a service over the internet. Tuscany and SCA's loosely coupled service architecture makes it a good programming model for developing applications in the cloud.

SCA separates all the infrastructure concerns from business logic and makes infrastructure choices declarative. Such declarations make it possible to replace standard infrastructure with cloud infrastructure without changing the business logic.

The cloud is a naturally distributed environment. SCA describes how services can be composed or assembled to create composite applications. Such assembly describes the various natural groups of functions in the application. The cloud infrastructure can take advantage of this information and provision separate groups of functions to different parts of the cloud.

Pulling this together, we can use SCA to remove infrastructure concerns from the application itself. The SCA assembly model can hide the details of the particular cloud infrastructure that's chosen. SCA also provides the opportunity to unify any infrastructure services as reusable SCA components. In this way the differences between different cloud providers can potentially be hidden.

Figure 11 illustrates some ideas about how the Tuscany domain manager can be enhanced to support provisioning of an SCA domain in the cloud.

The SCA domain can be partitioned into one more nodes that together run a group of components. The infrastructure requirements can be calculated from the composite, and a customized Tuscany runtime can be composed based on the technology extensions needed. The nodes can be provisioned to the cloud based on the infrastructure requirements calculated from the composites.

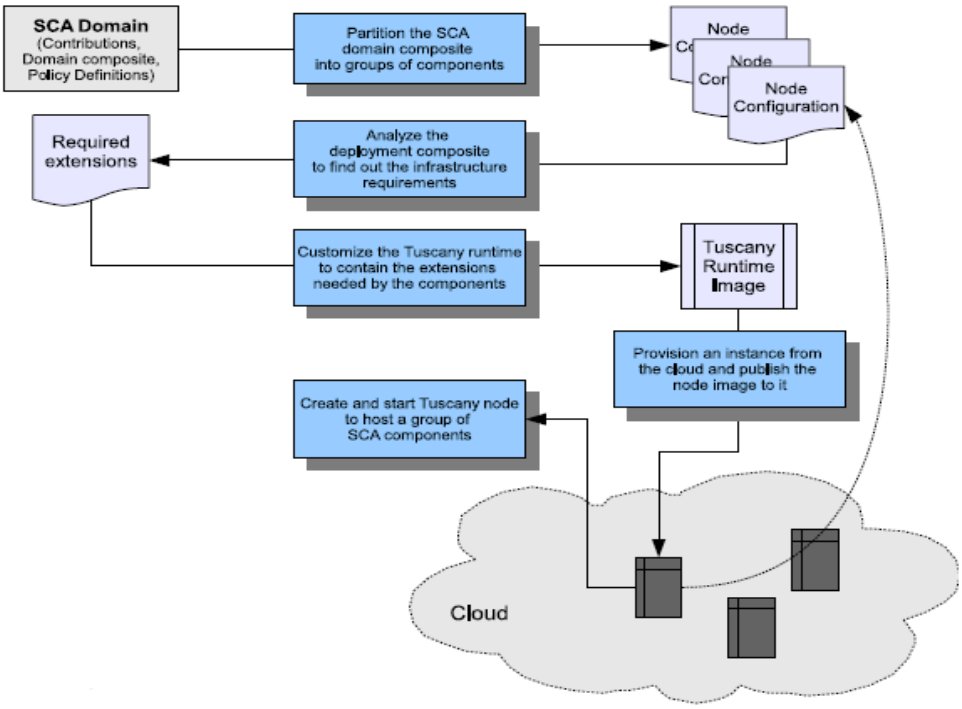


Figure 11. Provisioning an SCA Domain to the Cloud

7 KHRESMOI Cloud Infrastructure Approach

After analyzing the requirements specified by each of the partners in terms of hardware, software and infrastructure, we will define the cloud infrastructure for KHRESMOI (i) taking into account the SCA-based KHRESMOI architecture defined in deliverable D6.3.1 and (ii) using one of the cloud infrastructure management solutions discussed in Section 3, called OpenNebula.

If we analyze in detail the cloud requirements provided by each of the partners, we realize that in some cases the hypervisor or the number of virtual machines is not specified or quantified. Therefore, as we do not have a final specification in terms of real infrastructure to support each of the tools that the KHRESMOI system is composed, we have thought in carrying out a cloud infrastructure definition and deployment in two phases:

1) Phase 1. Services Orchestration Cloud (SOC): in this phase we will focus on just including the service orchestration (represented in the KHRESMOI system by the SCA Composites and the SCA Components) in the Cloud.
2) Phase 2. Full Services Cloud (FSC): in this phase the cloud deployed in Phase 1 will be updated in order to include on it all the components, exposed as REST Services in most cases and hosted by each partner. It is worth mentioning that these components defined in the KHRESMOI project, which cover all the functional needs exposed in the user interface (UI), were outside the Cloud deployed in Phase 1.

7.1 KHRESMOI Cloud Phases

In this section we describe in detail each of the phases defined for the deployment of the KHRESMOI Cloud infrastructure.

7.1.1 Phase 1: Services Orchestration Cloud

The objective of this phase is to define and deploy the cloud with the integration layer. This layer is composed of the SCA Composites as well as the SCA Components defined in the design of KHRESMOI architecture. The SCA Components are combined in the Composites to implement workflows or business processes that are aligned with the functionalities available in the UI by means of the presentation layer.

Figure 12 shows the Services Orchestration Cloud infrastructure in which we can observe three different parts:

1) *UI layer*: it is represented by the UI framework called ezDL. ezDL will work in an adaptive way in order to provide different views of the KHRESMOI system to the potential users identified, which are: general public, medical doctors, and radiologists.
2) *Service Orchestration layer*: it is represented by the Services Orchestration Cloud, and in which are included the SCA Composites that implement the workflows aligned with UI functionalities as well as the SCA Components that add a formal SCA definition to the KHRESMOI components.
3) *Service back end and Persistence layer*: it is represented by each of the components defined in the project and exposed in most cases using a REST API. The components are currently hosted in the USFD Cloud in the case of Mimir and Query Mapping Service (QMS), in the Ontotext Cloud in the case of BigOwlim and Disambiguator Service, in the HON Server in the case of Wrapin, in the CUNI Server in the case of the Multilingual Translator (MT), in the

HES Server in the case of GIFT and finally in the MUW Server in the case of the MUW Image System.

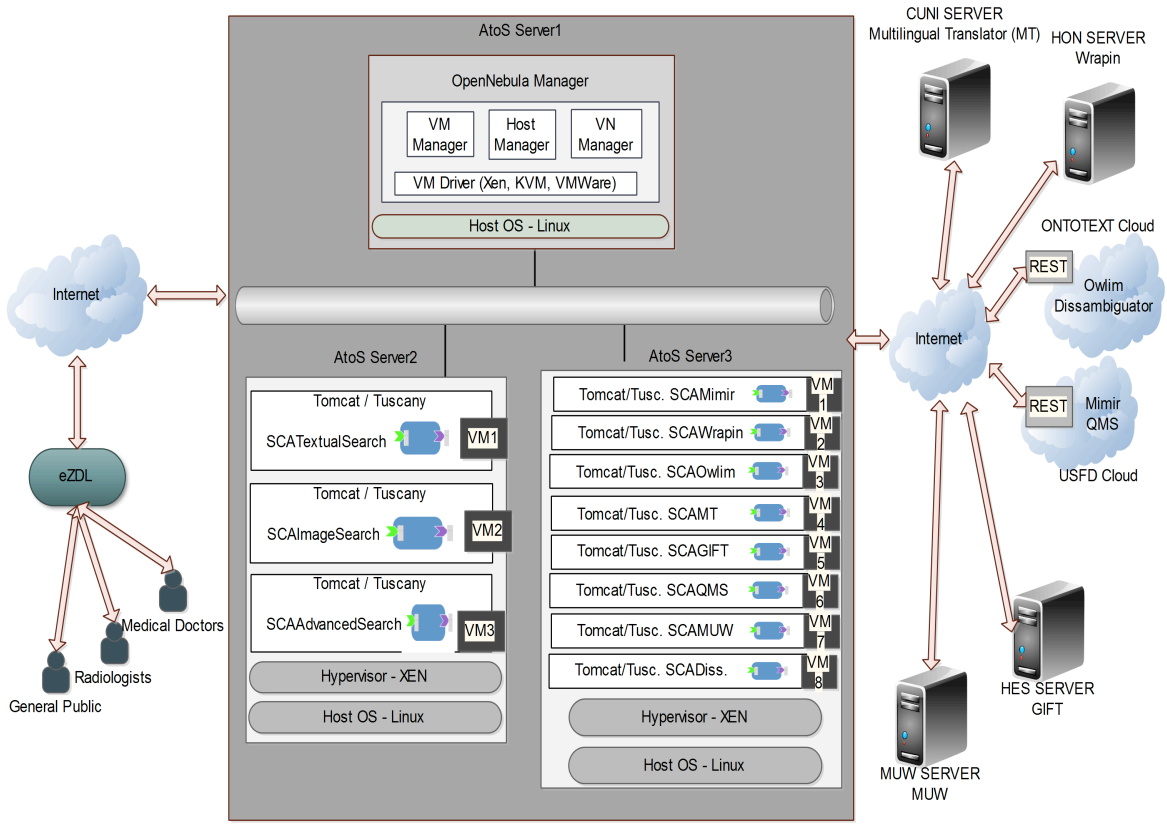


Figure 12. KHRESMOI Services Orchestration Cloud

Focusing on the configuration of the Services Orchestration Cloud hosted by AtoS we can see that in the cloud have been included three Servers.

In Server 1 the OpenNebula installation as well as all the components included in the OpenNebula 3.0 distribution for managing the cloud infrastructure are hosted.

On the other hand, in Server 2 there has been defined three virtual machines (VM) which host each one of the Main Composites and sub composites defined by the three main functionality blocks specified in the project: textual search, image search and combined search. These three main composites called *SCATextualSearch*, *SCAImageSearch* and *SCAAdvancedSearch* have been hosted in VM1, VM2 and VM3 respectively. As execution environment has been included in each of these VMs the Web Server Application Tomcat and the SCA execution framework Apache Tuscany. In this way the SCA composites are exposed with a binding Rest in order for the ezDL to provide the UI functionalities calling to the Composites Rest API exposed in the Cloud. In Section 7.3 the SCA composites are described in detail.

Finally, Server 3 hosts a set of VMs each associated with one KHRESMOi component in order to provide the list of SCA components to be combined in the SCA composites hosted in Server 2 VMs. The SCA components are exposed by means of a binding Rest and have as backend support the components deployed and hosted in the partner's servers or clouds. These components are Mimir and QMS (USFD Cloud), Wrapin (HON Server), GIFT (HES-SO Server), MUW Image System (MUW Server), Multilingual Translator (CUNI Server) and BigOwlim and Disambiguator (Ontotext Cloud).

7.1.2 Phase 2: Full Services Cloud

In this phase it is defined more concretely the deployment of all the components in clusters associated with the cloud environment proposed in Phase 1. These clusters are in charge of exposing each component with its specific software tools in order to be accessible in the full cloud system. In terms of clusters we can divide the cloud in two main parts:

- Service Provider Clusters, which contain the set of components defined in the project. In Phase 1, these components were out of the cloud; however, in this phase such components will be incorporated into the KHRESMOI Cloud infrastructure to create a complete virtualized and scalable environment. In addition, the UI framework represented by ezDL component will be included in the Full Services Cloud to provide all the functionalities identified for the three types of user defined in the Use Cases work packages: General public, Radiologists and Medical doctors.
- AtoS Cluster, which contains the SCA Composites for each workflow required for the Use Cases and the SCA Components which implement the functionalities required. These components will be exposed as REST services in this new Cloud environment, in the same way as explained in Section 7.1.1.

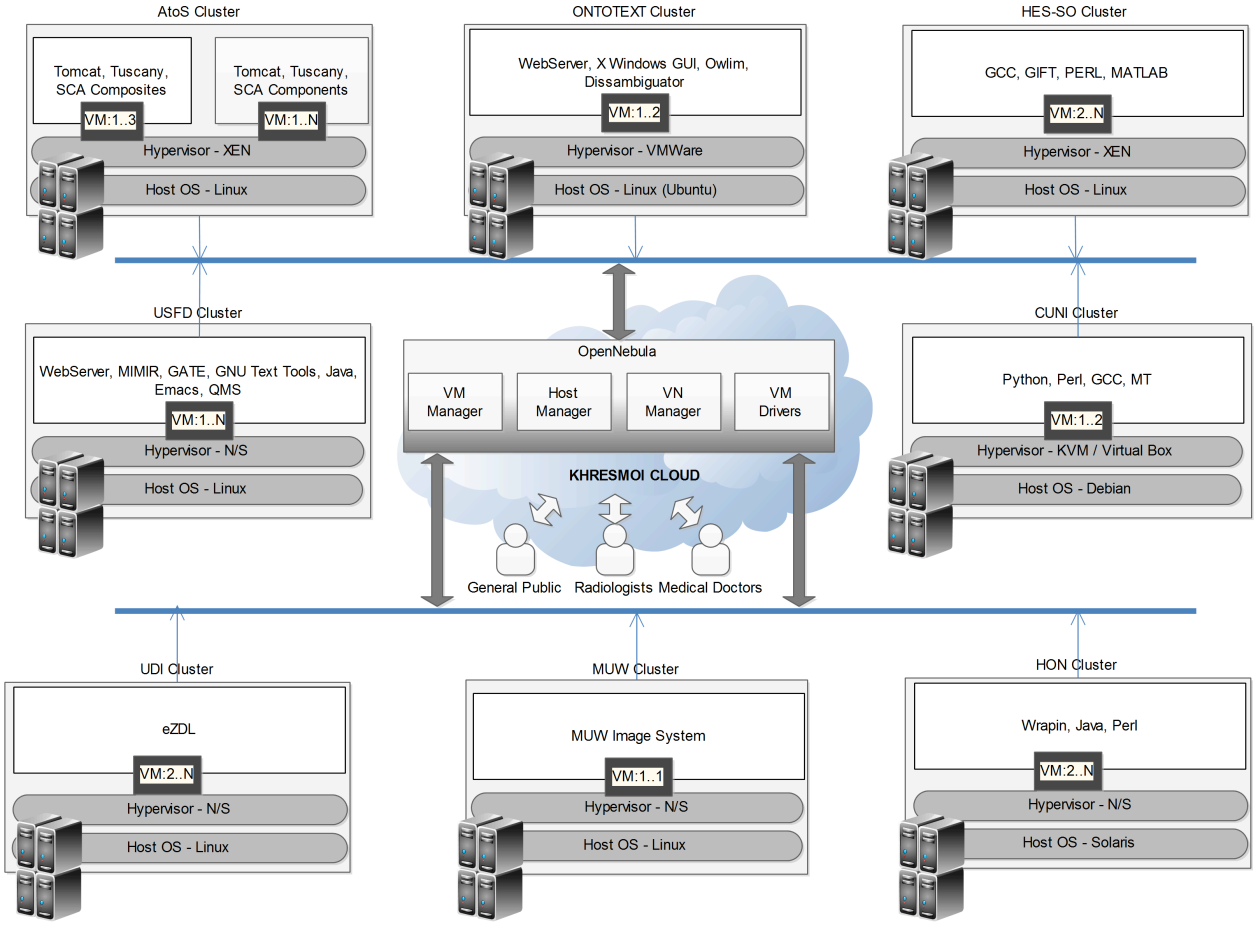


Figure 13. KHRESMOI Full Services Cloud

The Full Services Cloud system will be implemented with OpenNebula 3.0. In Figure 13 OpenNebula box includes the four main sub-components:

- VM Manager: It is responsible of the management and monitoring of VMs.
- Host Manager: manages and monitors the physical hosts.
- VN Manager: enables the mapping of virtual networks onto physical ones.
- VM Drivers: enables the use of the Virtual Machine hosts.

On the other hand in the same figure is shown the Components Clusters Cloud infrastructure in which we can observe the same structure for each Component Cluster associated with the cloud:

1) Host OS: It is the OS required for each component. In this structure proposed as "one cluster per component" it is possible to set up different OS in each cluster depending of the component requirements defined for the KHRESMOI partners.
2) Hypervisor: This is the software that allows the hardware virtualization in each cluster. It is also defined for each partner the hypervisor required. Some of them are Not Specified (N/S).
3) Software requirements: This means all the software needed for each component that must be installed in the clusters. In this case, this is defined for each partner too.

7.2 Planning the Private Cloud Installation

Taking as reference what OpenNebula assumes with respect to the physical infrastructure, the cloud proposed in Phases 1 and 2, which is showed in Figure 14, adopts a classical cluster-like architecture with a front-end, AtoS Server 1 and two cluster nodes (AtoS Server 2 and AtoS Server 3) where Virtual Machines will be executed. There is at least one physical network joining all the cluster nodes with the front-end.

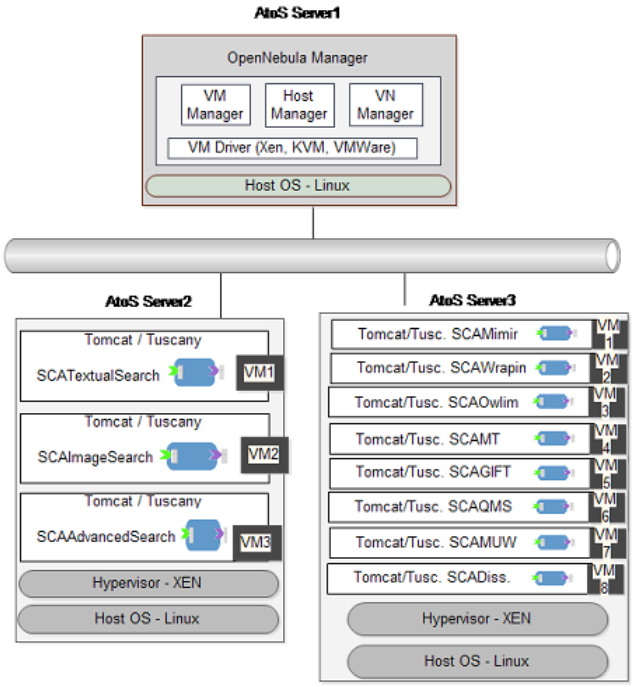


Figure 14. KHRESMOI Cloud high level architecture

The basic components of an OpenNebula system are:

- **Front-end**, which executes the OpenNebula and cluster services (Server 1).
- **Nodes**, hypervisor-enabled hosts that provide the resources needed by the Virtual Machines (Servers 2 and 3).
- **Image repository**, any storage medium that holds the base images of the VMs.
- **OpenNebula daemon**, is the core service of the system. It manages the life-cycle of the VMs and orchestrates the cluster subsystems (network, storage and hypervisors).
- **Drivers**, programs used by the core to interface with a specific cluster subsystem, e.g. a given hypervisor or storage file system.

7.2.1 Cluster Front-End

The front-end will access the image repository that should be big enough to store the VM images for KHRESMOI private cloud. Usually, these are master images that are cloned (copied) when we start the VM. Thus, we need to plan our storage requirements depending on the number of VMs we will run in our virtual infrastructure. Particularly, in the KHRESMOI AtoS Cluster we will have three VM in Server 2 for hosting SCA Composites and eight VMs in Server 3 for hosting the SCA Components identified in the project.

OpenNebula services include:

- Management daemon and scheduler
- Monitoring and accounting daemon
- Web interface server (sunstone)
- Cloud API servers (ec2-query and/or occi)

7.2.2 Preparing the Cluster

OpenNebula features an Image Repository to handle the VM Image files, the image repository has to be accessible through the front-end (AtoS Server 1) using any suitable technology NAS, SAN or direct attached storage. Figure 15 presents the storage model defined by KHRESMOI.

Images are *transferred* to the hosts to use them in the VMs. OpenNebula can handle multiple storage scenarios, in the future we assume that the front-end and hosts share a common storage area by means of a Shared FS of any kind. By doing this, we can take full advantage of the hypervisor capabilities (i.e. live migration) and typically better VM deployment times.

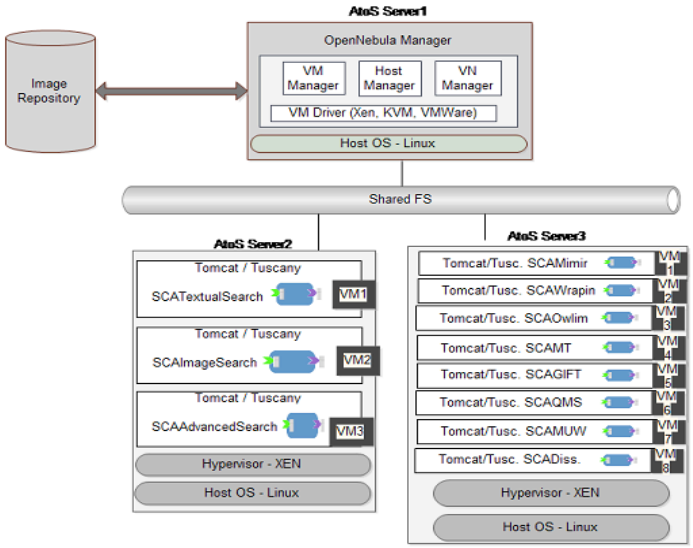


Figure 15. KHRESMOI Storage Model

7.3 Building KHRESMOI SCA Composites

Components are the most basic structures in SCA applications. At their simplest level, components may be classes exposed as an interface. A group of components can be combined to perform a more complex functionality; such structures are called "composites". A set of composites will be encapsulated into a domain which will provide the SCA runtime.

Our KHRESMOI distributed application consists of one domain (*KHRESMOI Cloud Domain*) hosted on separate machines (AtoS Cluster Nodes). Such a domain is composed of several composites which encompass its own set of components. *KHRESMOI cloud domain* consists of a set of components interacting with each other within a complex composite. A set of functionalities will be exposed as services from this domain. The *KHRESMOI domain* will be hosted on an instance running on the Cloud.

Figure 16 illustrates the design layout of part of the *KHRESMOI cloud domain*. The composite *TextualSearchService* consists of several components but we will describe those which have been integrated so far: *SCAWrapin, SCAOwlim* and *SCAMimir*. These components consists of a class *WrapinApplication, OwlimApplication* and *MimirApplication* which use the *JarsWrapinService, JarsOwlimService* and *JarsMimirService* interfaces, thereby exposing them as services. The methods presented in these classes return data in xml format.

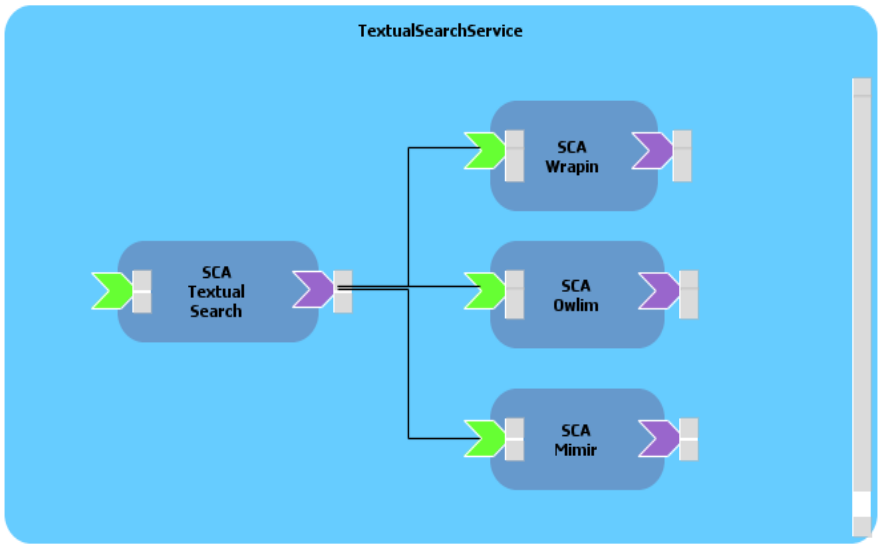


Figure 16. The Service: TextualSearchService

For example, taking as reference the Wrapin component in charge of provide spelling correction functionality we need to define all the following SCA pieces:

Interface service:

The interface is in charge of describing the methods provided by the component in order to make them accessible for other components

```
@Remotable
public interface JaxrsWrapinService {
	@GET
	@Path("/SPELLER/speller.pl")
	@Produces("application/rdf+xml")
	public String getSpellingCorrection(
		@QueryParam("xml") final String query);
```

Application service:

The application service is in charge of implementing the methods described for the interface

```
@Path("wrapin")
public class WrapinApplication {

    @Reference
    protected JaxrsWrapinService wrapinService;

    @GET
    @Path("speller")
    @Produces("application/rdf+xml")
    public String getStatements(@QueryParam("query") String queryText) {
        …
    }
```

Wrapin composite: web.composite

The composite WrapinSCA is composed by WrapinWebComponent, which represents the component hosted in the KHRESMOI cloud domain. This composite declares the reference to the service from an external WebService (hosted in an external domain where the partners must put their tools).

```
<composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
           xmlns:tuscany="http://tuscany.apache.org/xmlns/sca/1.1"
           targetNamespace="http://www.khresmoi.eu"
           name="WrapinSCA">

    <component name="WrapinWebComponent">
        <implementation.web web-uri=""/>
        <reference name="wrapinService">
         <interface.java
interface="eu.khresmoi.sca.wrapin.references.JaxrsWrapinService"/>
         <tuscany:binding.rest uri="http://debussy.hon.ch/cgi-bin"/>
        </reference>
    </component>
</composite>
```

Wrapin is included in the textual search composite that we describe next in terms of SCA pieces:

TextualSearchSCA Service

This service is hosted in the AtoS domain and uses the components in the KHRESMOI cloud domain in order to implement the Textual Search use case workflow. To complete this goal, the TextualSearchApplication class must invoke the component interfaces which reference each component.

TextualSearchApplication

This class implements the workflow by calling to the different component references. Its method *getTextualSearch()* is in charge of controlling all the functionalities required:

```
@Path("prototype1")
public class TextualSearchApplication {

    @Reference
    protected JaxrsWrapinService wrapinService;
    @Reference
    protected JaxrsMimirService mimirService;
    @Reference
    protected JaxrsOwlimService owlimService;
    @GET
    @Produces(MediaType.APPLICATION_XML)
    public List<Doc> getTextualSearch(
            @QueryParam("query") String queryText) {
      // Step1: spellingCorrection
      String textchecked = checkSpelling(queryText);
      // Step2 : dissambiguation
      // Step3 : translation
      // Step4 : query expansion
      // Step5 : search
      …
      return docList;
  }
}
```

TextualSearchSCA Composite: web.composite

The composite TextualSearchSCA is composed by TextualSearchWebComponent, which represents the component hosted in the AtoS domain. This composite declares the reference to the components from an external WebService (hosted in the KHRESMOI cloud domain). Thus, a web service binding will be established between the AtoS domain and the KHRESMOI cloud domain. These bindings are implemented with Tuscany and they will be REST binding type to an URI where the services are exposed:

```
<composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
           xmlns:tuscany="http://tuscany.apache.org/xmlns/sca/1.1"
           targetNamespace="http://www.khresmoi.eu"
           name="TextualSearchSCA">

    <component name="TextualSearchWebComponent">
        <implementation.web web-uri=""/>
        <reference name="wrapinService">
          <interface.java
interface="eu.khresmoi.sca.textualSearch.references.JaxrsWrapinService"/>
          <tuscany:binding.rest
          uri="http://khresmoi.AtoSorigin.es:8080/khresmoi-wrapin"/>
        </reference>
        <reference name="owlimService">
          <interface.java
interface="eu.khresmoi.sca.textualSearch.references.JaxrsOwlimService"/>
          <tuscany:binding.rest
          uri="http://khresmoi.AtoSorigin.es:8080/khresmoi-owlim"/>
        </reference>
        <reference name="mimirService">
          <interface.java
interface="eu.khresmoi.sca.textualSearch.references.JaxrsMimirService"/>
          <tuscany:binding.rest
          uri="http://khresmoi.AtoSorigin.es:8080/khresmoi-mimir"/>
        </reference>

    </component>
```

8 Conclusions

In this deliverable, we present a summary of the efforts provided in task T6.3 related to "System scaling". This document is an introduction to the cloud infrastructure aspects of the KHRESMOI system. Examining the state of the art permits to ensure the advanced positioning of KHRESMOI and to select a cloud specification approach that fits with the requirements of the projected KHRESMOI system.

KHRESMOI Cloud requirements in terms of hardware, infrastructure needs and software stack have been identified and analyzed. Taking as reference these requirements and the SCA integration approach defined in task T6.2 we have provided a high level description of the KHRESMOI Cloud platform by showing the relationship between the three pillars of the architecture: Open Nebula Apache Tuscany and KHRESMOI SCA Composites.

The SCA integration approach provides the most generic specification of software components and the mechanism to compose services. Thus, SCA provides a useful software infrastructure to integrate and deploy various software components as loosely coupled services.

Since SCA promotes all the best practices used in service-oriented architectures, building composite applications using SCA is one of the best guidelines for creating cloud-based composite applications. Applications created using several different runtimes running on the cloud can be leveraged to create a new component, as well as hybrid composite applications which scale on-demand with private/public cloud models can also be built using secure transport data channels.

Next the main concepts to deploy the first cloud infrastructure are described. Finally we describe the approach proposed for the definition and deployment of the Early Cloud Prototype which has been scheduled in two phases: the Services Orchestration Cloud phase and the Full Services Cloud phase. Apache Tuscany and OpenNebula will be the two main pillars for the implementation of both clouds.

9 References

[1] Lenzerini, M. (2002): Data integration: a theoretical perspective. In: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. ACM, New York, 233–246.

[2] Sheth A 1998, Changing Focus on Interoperability in Information Systems: From System, Syntax, Structure to Semantics, in Interoperating Geographic Information Systems, M. F. Goodchild, M. J. Egenhofer, R. Fegeas, and C. A. Kottman (eds) Kluwer Publishers.

[3] Forrester, Cloud Computing fir the Enterprise Webminar, 2009, http://www.forrester.com/imagesV2/uplmisc/CloudComputingWebinarSlideDeck.pdf

[4] Gartner, Key Issues for Cloud Computing 2009, 2009, ID:G00158484

[5] Gartner, Cloud Computing: Defining and Describing an Emerging phenomenon, ID: G00156220

[6] http://www.xen.org/

[7] http://www.linux-kvm.org/page/Main_Page

[8] http://www.virtualbox.org/

[9] http://www.vmware.com/es/

[10] "Intel® Virtualization Technology: Hardware Support for Efficient Processor Virtualization". Intel.com. 2006-08-10. http://www.intel.com/technology/itj/2006/v10i3/1-hardware/3-software.htm. Retrieved 2010-05-02

[11] Libvirt Virtualization API, http://libvirt.org

[12] OVF Whitepaper, http://www.vmware.com/pdf/ovf_whitepaper_specification.pdf

[13] Matthews, J., Garfinkel, T., Hoff, C. and Wheeler, J. 2009. Virtual machine contracts for datacenter and cloud computing environments. In ACDC '09: Proceedings of the 1st workshop on Automated control for datacenters and clouds, 25-30. http://people.clarkson.edu/~jmatthew/publications/acdc09-matthews.pdf

[14] Markus Böhm, Galina Koleva, Stefanie Leimeister, Christoph Riedl and Helmut Krcmar. 2010. Towards a Generic Value Network for Cloud Computing. http://vmkrcmar23.informatik.tumuenchen.de/1724/1/GECON_2010_Towards_a_Generic_Value_Network_for_Cloud_Computing.pdf

[15] http://www.vmware.com/es/

[16] https://fedorahosted.org/ovirt/

[17] http://www.vmware.com/

[18] http://www.opennebula.org/

[19] http://www.nimbusproject.org/

[20] http://www.eucalyptus.com/

[21] Eucalyptus Whitepaper, Eucalyptus Overview, http://www.eucalyptus.com/whitepapers

[22] Eucalyptus Whitepaper, The Eucalyptus Open-source Cloud Computing System, http://www.eucalyptus.com/whitepapers.

[23] Eucalyptus Whitepaper, Building a private Cloud with Eucalyptus, http://www.eucalyptus.com/whitepapers.

[24] Rafael Moreno-Vozmediano, Ruben S. Montero, Ignacio M. Llorente, Elastic Management of Cluster-based Services in the Cloud.

[25] An Open Source Solution for Virtual infrastructure Management in Private and Hybrid Clouds.

[26] Open Cloud Computing Interface WG (OCCI-WG) WIKI, http://forge.ogf.org/sf/go/projects.occi-wg/wiki