PUC-Rio - Certificação Digital Nº 0921330/CA

2 Background Work

This chapter describes the fundamentals of Cloud Computing and related works, organized as follows: Section 2.1 describes a recent definition and taxonomy of different concepts and technologies related to Cloud Computing. Section 2.2 describes related work for the deployment of distributed applications on grid environments and early adoptions of Cloud Computing.

2.1 Cloud Computing Fundamentals

Cloud Computing delivers computing resources as a service like electrical grids. Cloud Computing allows users to eliminate infrastructure annoyances, that is, administrators do not need to be responsible any more for their own hardware and software because these resources are managed by the cloud providers. Recently there has been a great deal of interest on standardizing Cloud Computing definitions, terminology, extensions, related technologies and so on. In order to combine efforts of cloud providers several organizations (software companies, governments, etc.) have been working together and as result they have begun to create initiatives to work on the standardization of a Cloud Computing taxonomy. In the next subsections we describe the main aspects of Cloud Computing.

2.1.1 What is Cloud Computing?

At the beginning of the adoption of Cloud Computing, cloud providers started to offer many types of Cloud Computing models and services. For example, Amazon Web Services began offering IT infrastructure services to businesses in the form of web services. From Amazon's perspective, computing infrastructure is shared like a utility and paid based on the amount of resources used, it can be easily upgraded and scaled up or down as needed. Google App Engine provides a platform to build and deploy web applications. This uses the reliability, performance, security, privacy, and data protection policies of Google's infrastructure. SalesForce started offering software as a service mainly its CRM (Customer Relationship Management) software solution. Therefore,

PUC-Rio - Certificação Digital Nº 0921330/CA

cloud providers proposed their own meanings about Cloud Computing. With an increasing adoption of Cloud Computing more companies have begun to offer private deployment models, therefore creating the beginning of a competitive Cloud Computing market.

Many projects and research groups have been testing available cloud platforms, but few of them have tried to define what exactly Cloud Computing is. Some papers highlight and discuss the key features and characteristics of Cloud Computing [NIST01], [Wang08], [Armbrust09], [Dong10], [Grandison10], [Liu10], [Voorsluys11]. However, there is not a standard definition of Cloud Computing, but one of the most referenced definitions of Cloud Computing has been proposed by NIST (National Institute of Standards and Technology):

Cloud Computing is a model for enabling ubiquitous, on-demand network access to a shared pool of configurable computing resource(e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essentials characteristics, three services, and four deployment models.

a) **Essentials Characteristics**

- *On-demand self-service*: Cloud consumers can provision computing capabilities such as compute time, storage, and network resources with minimal efforts.
- *Broad network access*: Cloud based resources are provisioned over the network, enabling consumers to access them from any device to fulfill the standard mechanisms.
- *Resource pooling*: The computing resources are pooled and shared across multiple consumers using a multi-tenant model, these resources are dynamically assigned and reassigned on-demand.
- *Rapid elasticity*: Provision and realization of computing resources can be applied quickly and can change dynamically to meet a variable workload.
- *Measured service*: Utilization is automatically controlled for metering and resource billing.

b) **Service Models**

- *Infrastructure as Service(IaaS)*: Cloud providers offer the provision of processing, storage, networks, and other fundamental computing resources where the cloud consumer is able to deploy and run arbitrary software.

- *Platform as a Service(PaaS)*: Cloud providers deliver a cloud infrastructure to deploy applications created using programming languages, libraries, services and tools.
- *Software as a Service(SaaS)*: Cloud providers make applications available in the cloud and cloud consumers can access them from any client device.

c) **Deployment Models**

- *Private Cloud*: The cloud infrastructure is provisioned only for a single organization. It can be managed by the organization or a third party, and also hosted internally or outside of the organization.
- *Public Cloud*: The cloud infrastructure is provisioned for the general public by a cloud provider.
- *Hybrid Cloud*: Composition of two or more clouds (private, public, community) working together through the use of compatible technologies.
- *Community Cloud*: The cloud infrastructure is provisioned only for a community of consumers, who share the same infrastructure.

All cloud infrastructures need to enable the five essential characteristics of Cloud Computing.

2.1.2 Cloud Taxonomy

In short time Cloud Computing has raised the interest of many types of users. However, a larger number of potential users did not find any guidance from cloud providers in terms of security, service level agreement, portability, etc. People with expertise in Cloud Computing started to propose the first taxonomies, classifying it as a series of interconnected layers. These layers are usually a combination of main cloud topics, such as infrastructure, storage, network, platform, application, services, providers, and consumers. Subsequentely, several publications started to propose different taxonomies [Hoefer10], [Prodan09], [RimalChLu09], [RimalCh09], [Teckelmann11], and [Strauch11], but at the moment there is still not a recognized standard taxonomy.

To reduce the ambiguous and confused taxonomy of cloud computing, the NIST Cloud Computing Program published a set of documents [NIST01], [NIST02] [NIST03], and [NIST04]. These documents offer a mature classification of cloud computing terminology. For example, the NIST Cloud Computing Reference Architecture proposes a generic high-level conceptual model

PUC-Rio - Certificação Digital Nº 0921330/CA

with the basic purpose of discussing the requirements, structures, and operations of cloud computing. Below we briefly describe this taxonomy and its main components:

a) **The Conceptual Reference Model**: The Conceptual Reference Model is displayed in figure 2.1, it shows a generic high-level architecture to provide a simple framework to understand the requirements, uses, characteristics and standards of cloud computing. This is composed of five major actors divided into *entities*, which represent a person or an organization, each is responsible for performing tasks in cloud computing. The entities are cloud consumers, cloud providers, cloud carriers, cloud auditors, and cloud brokers.

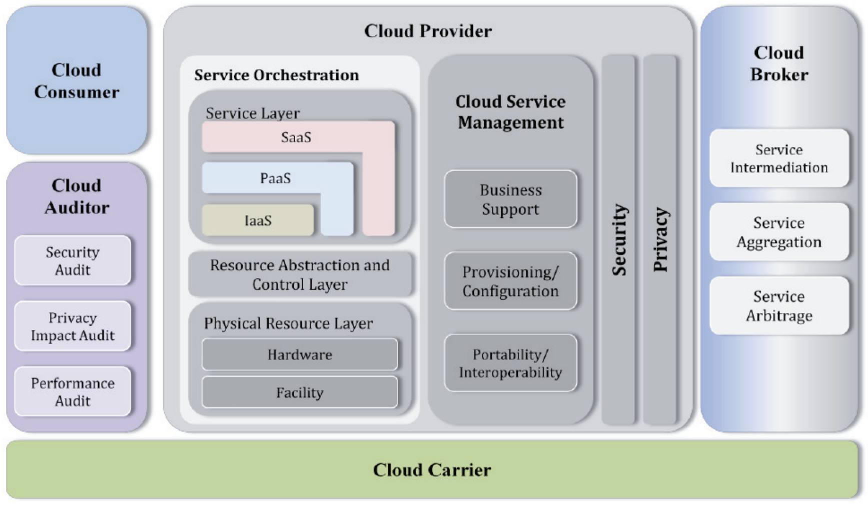

Figure 2.1: Conceptual Reference Model

The *Cloud Consumer* is an entity, who uses services obtained from a cloud services list offered by cloud providers. Once the appropriate services are chosen, an agreement is signed between the cloud consumer and the cloud provider. In most cases, to consume the cloud services involves a cost that is billed according to a pricing policy. The *Cloud Provider* is an entity, it acquires and maintains the operation of cloud infrastructure, and manages software applications. It comprises the management of computing resources, a cloud software to orchestrate its infrastructure, and creates a plan to make the cloud services available. The *Cloud Auditor* is an entity wich has the capabilities to deal with the independent assessments of cloud services, with the objective of evaluating the level of QoS, performance, portability, security, etc. The *Cloud Broker* is an entity responsible for the management of cloud services and negotiates relationships between cloud providers and

PUC-Rio - Certificação Digital Nº 0921330/CA

cloud consumers, because often the integration of cloud services can be too complex for cloud consumers. The *Cloud Carrier* can act as an intermediary that provides connectivity and a transportation network of cloud services from cloud providers to cloud consumers.

b) **Architectural Components**:

- *Service Orchestration*: Combines software components to support the management of computing resources. These components are organized in a three-layered model: *Service Layer*, *Resource Abstraction and Control Layer*, and *Physical Resource Layer*. In Service Layer, cloud providers create interfaces for access to cloud services. Resource Abstraction and Control Layer comprise a set of software components to manage the physical computing resources: hypervisors, virtual machines, virtual data storage, etc. This layer also enables resource pooling, dynamic allocation, and monitoring, therefore allowing the management of physical resources using software abstractions. Physical Resource Layer groups together all the physical computing resources.
- *Cloud Services Management*: Cloud Service Management groups together all the services offered by cloud providers to enable an efficient management of cloud services for cloud consumers. The services related to cloud services management are classified into three groups: *Business Support*, *Provisioning and Configuration*, and *Portability and Interoperability*. Business Support involves the provision of business-related services to cloud consumers, for example, customer management, account and billing services. Provisioning and Configuration services are middle services that share usability between cloud consumers and cloud providers. For example, services related with the SLAs, metering, monitoring, reporting, etc. Portability and Interoperability services are characteristics that cloud consumers seek to achieve autonomy in the migration process between different cloud providers, or when they need to combine the communication layer of several cloud providers.

2.1.3 Cloud Platforms

There are two standpoints of the term *platform* in cloud computing, it can be used to refer either a Cloud Platform or the Platform as a Service(PaaS). In the context of cloud computing we use the first idea of platform. Hence, cloud platforms allow us to build cloud infrastructures, and permit developers to host cloud applications using cloud infrastructures as their target environment.

PUC-Rio - Certificação Digital Nº 0921330/CA

PUC-Rio - Certificação Digital Nº 0921330/CA

A typical cloud platform includes a cloud application hosting server and a cloud storage device such as a database. Application developers can run their software applications on cloud platforms without worrying about higher cost and the complexity to buy, build, and maintain a hardware infrastructure. Cloud providers also offer to cloud consumers a set of integrated services such as embedding frameworks, libraries, applications, etc.

Today we can find several open-source cloud platforms as well as private cloud platforms offering a large number of services. Below we describe two open-source platforms - Eucalyptus and OpenStack. Additionally, we review Amazon Web Service (AWS), which although it holds a private cloud platform it offers a full documentation and mature cloud interfaces.

a) **Eucalyptus**: Eucalyptus is an open-source framework for cloud computing, which implements the IaaS model and enables the creation of on-premise IaaS clouds [Nurmi09]. Eucalyptus is compatible with Amazon Web Services(AWS) and conforms with EC2, S3, and EBS specifications. Eucalyptus accomplishes this with both the syntax and semantic definitions of Amazon API and tool suites. Internally, Eucalyptus is highly modular with a hierarchical design and language-agnostic communication mechanisms. Eucalyptus allows users to manage virtual machine instances (i.e. start, stop, reboot, terminate) using REST and SOAP interfaces. Eucalyptus is composed of five types of components: Cloud Controller(CLC), Walrus, Cluster Controller(CC), Storage Controller(SC), and Node Controller(NC). Cloud Controller offers a EC2-compatible SOAP and Query interfaces, while also performing high-level resource scheduling and system accounting. Walrus implements a bucket-based storage system available through S3-compatible and REST interfaces. Both, Cloud Controller and Walrus are called top-level components, and add resources from multiple clusters.

b) **OpenStack**: OpenStack is an open source cloud initiative launched by NASA and Rackspace in July 2010. OpenStack provides a software platform to build massively scalable applications [OpenStack11], and is composed of a collection of open source technologies to build public and private clouds. It integrates the Nebula platform from NASA and the Cloud Files platform from Rackspace. OpenStack has three projects: OpenStack Compute (Nova), OpenStack Storage (Swift), and OpenStack Image Service (Glance). Nova is a cloud fabric controller used to provide on-demand virtual machine instances and volume services like EBS. Swift is a system to store and retrieve objects with scalability, redundancy and failover features.

Glance is a lookup and retrieval system for virtual machine disk images, and acts as a catalog for disk images.

At the moment of writing this dissertation two new incubating projects are being added to OpenStack: Dashboard (Horizon) and Identity (Keystone). The first, OpenStack Dashboard is a modular web-based interface for all OpenStack Services. The second, OpenStack Identity provides authentication and authorization for all of its services. OpenStack has APIs compatible with Amazon EC2 and Amazon S3, and therefore client applications for Amazon Web Services can be used with OpenStack using minimal porting effort. Finally, we found a rapid evolution of OpenStack, thus a recent list (august-2012) includes the new proposed modules: OpenStack Networking (Quantum), OpenStack Block Storage(Cinder), and OpenStack Identity(Keystone).

c) **Amazon Web Services**: Amazon Web Services(AWS) is the leading cloud provider offering a highly reliable, scalable, and low-cost infrastructure platform, with datacenters located in five countries: U.S., Ireland, Singapore, Japan, and Brazil [AmazonWS06]. AWS began offering IT infrastructure services to businesses in the form of web services, initially focused on computer power and storage, denominated as Elastic Cloud Computing (EC2) and Simple Storage Service (S3), respectively. AWS organizes its services into three groups [AmazonWSDoc]: *Foundation Services*; *Application Platform Services*; *Management and Administration*.

Amazon EC2 is a web service offering a resizable compute capacity, that is, it allows us to launch virtual machine instances from a pre-configured or customized virtual machine image (Amazon Machine Image) and a type of instance. The type of instance depends of the computing requirements. AWS offers command-line tools for the management of virtual machine instances: ec2-api-tools and ec2-api-tools. Amazon S3 is also a web service, which enables to storage and retrieval of any kind of objects, from anywhere at, any time.

2.1.4 Cloud APIs

To access all cloud infrastructures a cloud interface is necessary. Each cloud provider offers its own cloud API, however it involves restrictions of use to the cloud provider. Faced with these restrictions, people have started to implement generic APIs to support several cloud platforms. In this subsection we describe three generic cloud APIs.

PUC-Rio - Certificação Digital Nº 0921330/CA

a) **JClouds**: JClouds is a multi-cloud library, which allows us to write abstract codes to control cloud resources from many cloud providers [JClouds11]. JClouds features a simple interface, runtime portability, unit testability, location-aware, and performance. Currently, according to JClouds' documentation, it offers two java and closure API implementations: Blobstore and Compute Service. Blobstore deals with key-value providers such as Amazon S3, and ComputeService manages virtual machine instances such as Amazon EC2. JClouds supports the main cloud providers, specifically JClouds has been testing over 30 cloud providers and cloud platforms, including OpenStack, Amazon Web Services, Eucalyptus, GoGrid, and Microsoft Azure.

b) **LibCloud**: The Apache LibCloud team has worked on its own multi-cloud API [LibCloud10]. LibCloud has denominated Cloud Server, Cloud Storage, Load Balancers as a Service(LBaaS), and DNS as a Service(DNSaaS) to cloud services focused on managing compute, storage, load balancers, and DNS resources, respectively. Apache LibCloud is a standard python library designed to support multiples of cloud providers. It has developed its own cloud terminology such as Node, NodeSize, NodeImage, NodeLocation, NodeState, etc. When a virtual machine starts the Cloud Server module allows users to insert and execute shell scripts inside them, faciliting the customization of new instances. Cloud Storage has been developed to manage cloud storages such as Amazon S3, Rackspaces CloudFiles, and Google Storage.

c) **DeltaCloud**: DeltaCloud abstracts different cloud providers and platforms [DeltaCloud11]. This API supports Compute and Storage services. DeltaCloud API works as a wrapper and uses a particular driver to communicate with each cloud provider. DeltaCloud users allow different HTTP clients to communicate with the server using the DeltaCloud REST API. DeltaCloud also provides a web application to manage the cloud infrastructure supporting mobile or tablet devices.

2.1.5 Cloud Software Stack

Today the installation and configuration of cloud infrastructures are possible, mainly thanks to open-source initiatives. Fortunately, cloud providers and cloud platform developers are focusing on offering cloud consumers a more compatible set of libraries, tools and APIs. In this subsection, we review common aspects of recent cloud platforms, virtualization and libvirt. We will also describe a popular command-line tool.

PUC-Rio - Certificação Digital Nº 0921330/CA

PUC-Rio - Certificação Digital Nº 0921330/CA

a) **Virtualization**: Virtualization is the creation of a virtual version of computing resources, such as networking, storage devices, operating systems, or hardware resources, and involves a logical separation of the requests for these services from the underlying physical resources. David Chisnall compares virtualization with emulation [Chisnall07]. Emulation means that a system pretends to be another system, whereas in virtualization a system pretends to be two or more of the same system. William von Hagen gives a practical definition of virtualization as the ability to run applications, operating systems, or system services in a logically distinct system environment independent of a specific physical computer [vonHagen08].

   There are many types of virtualization that are found at software or machine level. We have studied the machine virtualization, also known as *server virtualization*, platform virtualization, or simply as virtualization. With platform virtualization, we can execute many virtual machines into the same hardware, without sharing the same kernel. Also, platform virtualization is commonly defined as the technology that introduces a software abstraction layer between the hardware and the operating system. This abstraction layer is represented by the *virtual machine monitor* or *hypervisor* and allows instantiation of virtual machines. The host machine is the actual machine on which the virtualization takes place, and guest machines are virtual machines running on hosts. There are different platform virtualization approaches to implement the platform virtualization, main virtualization techniques are full-virtualization, para-virtualization, and hardware-assisted virtualization.

b) **Libvirt**: Libvirt is an open-source virtualization API becoming the underlying base library on which most of virtualization management software are being built, and widely used to manage the virtualization layer [Libvirt05]. Libvirt provides a set of language bindings focusing on interoperability. It is used as the abstract layer to develop cloud platforms because it allows the user to implement once and gain support from several virtualization platforms. Libvirt has support from Xen and KVM platform virtualization, in addition, after many efforts since 2005 libvirt works with VirtualBox, VMWare, OpenVZ, LXC, and User Mode Linux.

c) **Euca2ools**: Euca2ools is a python-based package of command-line tools for the management of Eucalyptus-based cloud infrastructures, inspired by Amazon command-line tools (api-tools, ami-tools) [Euca2ools08]. Euca2ools allows users to manage instances, images, volumes, snapshots, IP address, security groups, SSH key pairs, and availability zones. Euca2ools uses Boto and M2Crypto libraries, and internally interacts with web services that export a REST/Query-based API. Euca2ools is compatible with Amazon Web Services(EC2 and S3) and OpenStack. Most Linux distributions have added Euca2ools to their repositories making its installation an easy task. To start using Euca2ools users need to export their security credentials.

2.2 Related Work

In the last decade, software deployment in large-scale infrastructures such as grids has raised many challenges and has been focused on several research areas such as climate modeling, earthquake simulation, protein folding, etc. Recently, another type of large-scale infrastructure has facilitated the provision of on-demand computing resources, known as *Cloud Computing* [RimalChLu09]. Several computing paradigms have taken advantageof the adoption of Cloud Computing, specially the *Infrastructure as a Software(IaaS)* usage as the main provider of computing resources. We have found limited related work proposing new approaches and tools for the deployment of distributed component-based applications on large-scale computing infrastructures such as Grid Computing and Cloud Computing. In the following paragraphs we give a short introduction to the deployment process and related work for the deployment on grid environments and cloud environments.

2.2.1 Deployment Process

The figure 2.2 depicts a state diagram of a generic deployment process of distributed component-based applications. It is composed of ten interrelated activities linked with black arrows. The ten enumerated activities are packaging, publishing, planning, installing, configuring, activating, updating, deactivating, uninstalling, and retiring [Heydarnoori08]. Also, this figure shows the deployment activities interacting with a *packager service*, a *repository service*, and a *target environment*. Solid arrows connect the deployment activities with the packager and repository services, and the dashed arrows depict the activities interacting with the target environment.

We explain this generic deployment process as follows. First, the packaging activity is responsible for storing the components in their respective packages, it is supported by a packager service(a). The packager service deals with multi-platform, multi-language, and versioning challenges, as well as resolving static dependencies. Hence, we obtain the application release as a set of packaged components. Second, the publishing activity brings these packaged

PUC-Rio - Certificação Digital Nº 0921330/CA

components to a repository. This step is performed by the repository service(b), its main function is to keep a list of components published. Third, the planning activity builds a high level plan, it is responsible for managing all necessary settings before installing the components in a pre-defined target environment. Fourth, the installing activity executes the installation according to the previous plan. It describes which components will be obtained from the repository(c) and carried out to the target environment(d). Fifth, the configuring activity allows changes to be made to default configuration creating a customized configuration(e). Sixth, the activating activity makes the application available and finally it is ready to use(f). Seventh, the updating activity permits performing changes to a current configuration, it can involve another request to repository(g) and updates the components located in the target environment(h). Eighth, the deactivating activity stops the application(i). Ninth, uninstalling means removing the application's components(j). Finally, the tenth activity is called retiring, it deletes components from the repository(k). All these deployment activities are managed by an orchestrator, thus the application's components are installed into its target environment previously specified.

PUC-Rio - Certificação Digital Nº 0921330/CA

PUC-Rio - Certificação Digital Nº 0921330/CA

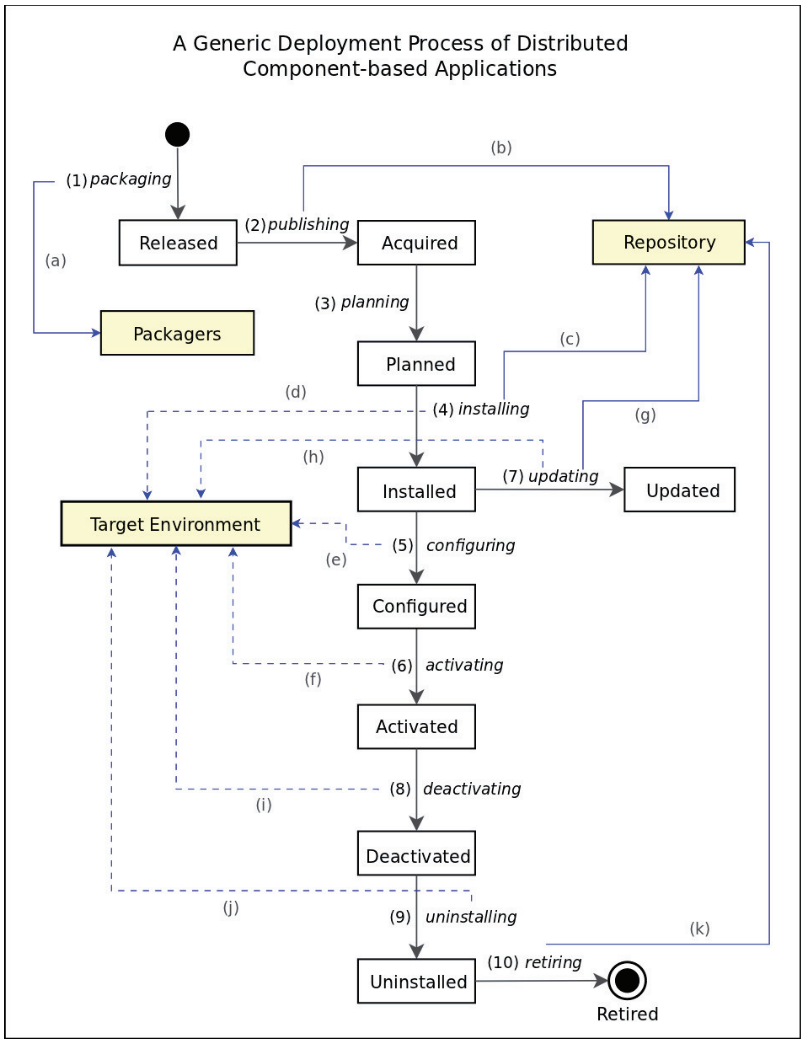

Figure 2.2: A Generic Deployment Process of distributed Component-based Applications

2.2.2 Deployment on Grid Environments

In this subsection we describe three works focusing on the deployment of applications on grid environments. They have different features and were built to run in a specific target environment, however we review aspects of the specification of their target environments.

Fractal and DeployWare

Fractal is a hierarchical and reflective component model, designed to implement, deploy, and manage complex software systems, e.g. middlewares [Bruneton06]. Fractal supports composite components, shared components, introspection, and re-configurable capabilities. A fractal component is an execution entity, which has two parts: controller(membrane) and content.

The controller groups together functional and control interfaces. Functional interfaces are the server interfaces, providing methods to other components like facets and client interfaces like receptacles used to invoke methods. Control interfaces manage non-functional aspects, it is the content that implements the business functionality.

Fractal components can be deployed by the *DeployWare Framework* [Flissi08]. DeployWare was built using fractal components and allows the deployment of distributed and heterogeneous software applications on large scale infrastructures such as grids. It was created to deal with complexity, heterogeneity, validation, and scalability challenges. DeployWare provides (i) a Domain Specific Modeling Language(DSML) based on a metamodel representing distributed deployment concepts and masks software heterogeneity from the user's point of view, (ii) a virtual machine(FDF) executes the deployment process, it contains a library that disguises the heterogeneity of physical machines, and (iii) a graphical console for monitoring and management. Deployment process with DeployWare involves three actors: a *system administrator* describes the nodes, network options, protocols, etc., a *software expert* details the deployment activities, and *end users* declare both the software used to deploy and its target environment.

The Fractal Deployment Framework (FDF) is a component-based framework, which automates the deployment of distributed applications. Deployments are described using a high-level FDF deployment description language or simply FDF language. It is based on a subset of Fractal ADL (Architecture Description Language) and allows end-users to describe deployment configurations. FDF language is also utilized to define a library of deployment components. Additionally, FDF provides a graphical user interface allowing end-users to load, execute, and manage their deployment configurations

DeployWare groups all hosts to define a *network description*, where each host is declared using INTERNET.NETWORK variable, with the following elements: hostname, user, transfer protocol, access protocol, shell, and compute(optional). The hostname parameter supports static and dynamic options: INTERNET.HOSTNAME(NameOfHost) and INTERNET.DYNAMICHOSTNAME. The first option is only necessary to set a hostname or an IP. The second option allows users to define the host in runtime. Code 2.1 shows an example of a static definition of one host where its hostname is defined statically. Code 2.2 declares a dynamic definition of a host on Grid5000 network. The name is computed at runtime after a reservation on the grid called Grid5000 [Grid5000Fr].

DeployWare also deals with large deployments, for example Code 2.3

PUC-Rio - Certificação Digital Nº 0921330/CA

Listing 2.1: Declaration a Static Node

```
GRID5000.STATICNODE = INTERNET.HOST {
  hostname = INTERNET.HOSTNAME(NameOfTheHost);
  user     = INTERNET.USER(doe, password,/home/doe/.ssh/id_rsa);
  transfer = TRANSFER.SCP;
  protocol = PROTOCOL.OpenSSH;
  shell    = SHELL.SH;
}
```

Listing 2.2: Defining dynamic Grid5000 hosts

```
GRID5000.DYNAMICNODE = INTERNET.HOST,
 org.objectweb.fdf.adl.Binding(hostname.ch,compute.ch)
{
  compute  = INTERNET.COMPUTEHOSTNAME(UNDEFINED);
  hostname = INTERNET.DYNAMICHOSTNAME;
  user     = GRID5000.OARUSER(OUTPUT_FILE);
  transfer = TRANSFER.OARCP;
  protocol = PROTOCOL.OarSH;
  shell    = SHELL.SH;
}
```

declares 50 nodes on a Grid5000 using an ADL element called *apply* [Dubus08]. OAR, the resource manager of Grid5000, launches a reservation request of 50 nodes and stores the hosts into the list /tmp/nodes. The compute element returns a hostname from a previous generated list of nodes. Hosts are specified using the iteration *apply* Fractal ADL, which allows system administrators to set an available host from nodes reserved and not previously assigned to another DynamicHost.

Listing 2.3: Extract of DeployWare definition which declares 50 nodes

```
GRID.TEST {
  oar = GRID5000.OARGRID(oar_args n=50, /tmp/nodes);
  compute = INTERNET.COMPUTEHOSTNAME(/tmp/nodes);

  /* The declaration of nodes */
  Hosts = GRID5000.G5K_NETWORK {

    g5k-nodes {
      apply ForEachInIntegerRange(i,1,50) {
        node-%{i} = GRID5000.DYNAMICNODE {
          user = INTERNET.USER(jdubus, ,~/.ssh/id_rsa);
          compute = /;
        }
      }
    }
  }
}
```

ADAGE and CoRDAGe

A Generic Application Description(GADe) model was proposed by Lacour et al. [LacourPePri05] to represent specific application descriptions. The GADe consists of three interconnected computing entities: *system entities*, *processes*, and *codes to load*. A system entity includes one or more process,

PUC-Rio - Certificação Digital Nº 0921330/CA

PUC-Rio - Certificação Digital Nº 0921330/CA

a process is made up of a running instance of a program and the codes to load it, sharing a common space in memory. Also, there are connections between the system entities. A system entity is deployed on distributed hosts, and all processes of the same entity run on the same host. GADe supports the specification of a list of operating system and/or computer architecture as its target environment. Based on this model ADAGe was built, a tool for automatic deployment of distributed and parallel applications. Deployment planning has been developed to accept a generic application description, this was possible because the application context assumes the execution of threads and processes of parallel and distributed paradigms. Deployment planning layer selects compute nodes and installs the components automatically. A translator makes the conversion from a specific application description to a generic description format. Finally, the plan is executed making the necessary configurations.

CoRDAGe is a third-party tool based on ADAGe, for grid applications focusing on *re-deployment* and *co-deployment* of components, enabling a dynamic management of different application types [CudennecAnBo08]. Expansion and retraction approaches enable the *re-deployment* feature, that is, new entities could be added or removed from a previously deployed application, respectively. *Co-deployment* allows users to deploy several applications, grouping sub-applications within their own constraints. CoRDAGe defines an application as a set of types of entities, where each entity represents a program to be executed on a physical resource, therefore the entity is the CoRDAGe unit element designed to be deployed in a single host. These entities are managed by building logical groups, posteriorly generalized into logical trees. They introduce the notion of the virtual node which is located at the root of the tree, then logical trees are mapped for physical resources. Also, physical resources are grouped in hierarchical physical groups in a tree format. The building of CoRDAGe was possible due to the high-level model proposed to describe both the applications and physical resources. Figure. 2.3 displays these representations.

The CoRDAGe model designed its target environment based on grid scenarios. To join the re-deployment of components and the reservation of nodes CorDAGe does not work migrating deployed entities to other hosts, instead it aims to deploy new entities (expand) or remove deployed entities (retract).

DAnCE

DAnCE(Deployment and Configuration Engine) [Deng05] is a QoS-enabled middleware framework, it aims to deploy Distributed Real-time and

PUC-Rio - Certificação Digital Nº 0921330/CA

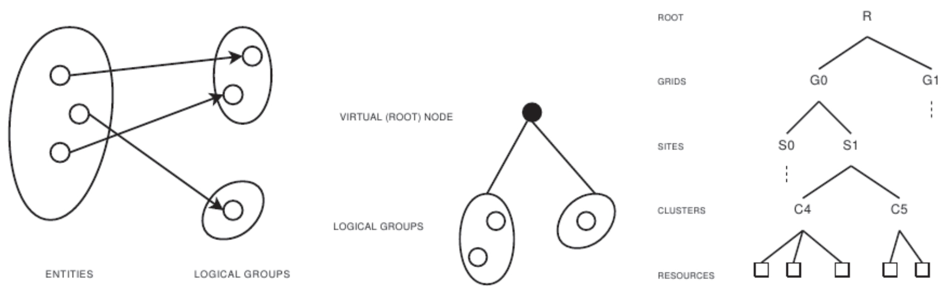

Figure 2.3: Representing physical resources using a physical tree

Embedded(DRE) systems dealing with collocation, memory constraints, and processor loading challenges. DAnCE proposes a combination of a standard runtime environment and metadata to address challenges of DRE systems. DAnCE maps known variations in the application requirements space to known variations in the software solution space. Deng et al. have developed an Inventory Tracking System (ITS) as study case for DAnCE. ITS faces major challenges of efficient storage and retrieval components (multi-platform) from a repository, management and configuration of component's life-cycle, and integration with common middleware services. DAnCE implements the OMG Deployment and Configuration Specification(OMG D&C) [OMGDC06]. OMG D&C standardizes the deployment process and configuration focusing on features such as component configuration, component assembly, component packaging, package configuration, package deployment, and target domains. OMG D&C defines *domain* as the target environment composed of nodes, interconnections, and bridges. DAnCE uses a Data Model and Runtime Model to represent a sequence of steps that components need to operate. In turn the data model uses XML schemas to store metadata for managing the deployment of component assemblies. Meanwhile, the runtime model specifies a set of managers to process the previous metadata. Figure 2.4 shows the design of DAnCE, its architecture is managed by ExecutionManager, DomainApplicationManager, NodeManager, NodeApplicationManager, NodeApplication, and RepositoyManager. We have paid attention to ExecutionManager and NodeManager, because they work directly with the deployment plan and its target domain. ExecutionManager is responsible for overseeing the deployment procedure for one or more domains, it uses the factory and finder design patterns to manage DomainApplicationManagers. This then in turn carries out the deployment over one or more domains. NodeManager allows each node to manage the deployment of itself. RepositoryManager uses a zip compression to package the components.

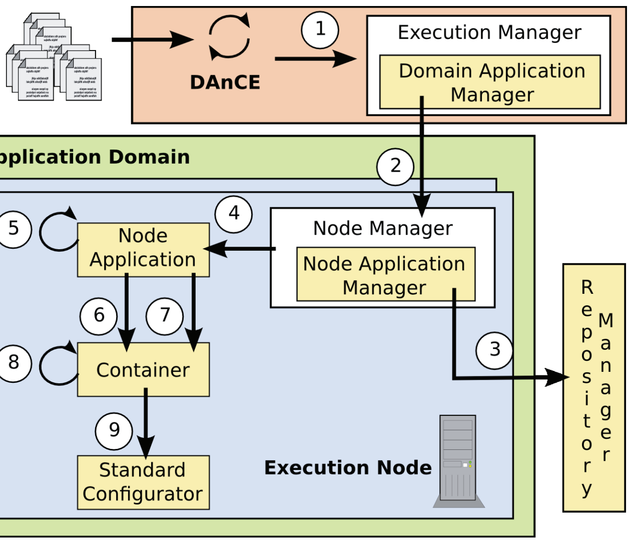

Figure 2.4: Internal Architecture and Deployment Stages of DAnCE

Therefore, we can consider that ExecutionManager, RepositoryManager, and NodeManager services will be run on separated physical machines. Additionally, RepositoryManager needs to be instantiated for each platform i.e. Windows, Linux, etc. DAnCE was tested over an ITS software, which includes 200 components deployed on 26 physical nodes in a warehouse, thus the target environment is previously known. DAnCE describes the QoS requirements and target environment using XML descriptor files.

2.2.3 Deployment on Cloud Infrastructures

We review two new approaches, FraSCAti and OSGi, which aim to solve recent cloud computing challenges and support the development of PaaS.

FraSCAti

FraSCAti is an open-source implementation of Service Component Architecture (SCA) standard, which enables the development and deployment of distributed SCA-based applications. A recent publication [Merle11] suggests the SCA standard as a framework for designing and developing cloud software. Merle et al. have worked trying to address three main developer questions: (i) How to deploy applications for several cloud providers? (ii) How to migrate applications between different cloud providers? and (iii) How to manage applications deployed on multiple cloud providers?. These questions have emerged as a consequence of an increasing concurrence of cloud providers. Challenges such as migration, interoperability, brokering, and geo-diversity

PUC-Rio - Certificação Digital Nº 0921330/CA

PUC-Rio - Certificação Digital Nº 0921330/CA

need to be tackled. The proposal was to develop a reflective middleware-based solution with two objectives: adapting applications to different cloud providers and managing applications deployed on different cloud providers (multi-Cloud systems). This middleware is denominated FraSCAti platform [Seinturier09]. FraSCAti was built using a specialization of the Fractal component model, therefore it supports introspection and reconfiguration capabilities. They deployed the FraSCAti platform and six SCA-based applications over eleven IaaS/PaaS providers. The experiments show that FraSCAti and SCA-based applications can address static migration, inter-operability, and geo-diversity. Merle et al. have contributed enumerable lessons learnt from the experiments. We consider their major contribution to be related to the use of FraSCAti as an extension layer between applications and PaaS/IaaS, because it permits the combination of SCA heterogeneity and FraSCAti fine-grained reflectivity with Cloud Computing scalability, and allows building of large-scale heterogeneous multi-Cloud systems. Additionally, they argue that middlewares and applications need to be more flexible and propose the use of use adaptability in design and runtime levels. The adaptability of design time is accomplished with the use of plugin-like SCA-based architecture and features diagrams. FraSCAti has a plugin-based architecture and allows applications to be adapted in different environments. Feature diagrams are proposed to get a Software Product Line(SPL) to control the complex forms of plugin combinations. For the adaptability of execution time FraSCAti offers two features: reflection and dynamic deployment of reconfiguration scripts.

Paraiso et al. in [Paraiso12] complements the previous work detailing their federated PaaS infrastructure for multi-cloud applications. Federated Clouds mean the union of internal and external clouds, also called interclouds. This infrastructure was built based on (i) an open service model; (ii) a configurable architecture; and (iii) infrastructure services. The open service model is defined by the FraSCAti platform. It allows the handling of portability, interoperability, and heterogeneity challenges. The federated multi-cloud PaaS infrastructure has been built over a generic configurable kernel, inspired by the FraSCAti platform and the design of a Software Product Line (SPL). Hence, the SPL is used to develop both common characteristics and the points of variability, between cloud providers (plugins) using the FraSCAti platform to build this SPL. Finally, four common services were build to support the use of the multi-cloud infrastructure: (i) Cloud Node Provisioning enables the allocation of resources before the deployment phase; (ii) PaaS Deployment Service supports the deployment of the configurable kernel together with a SaaS; (iii) SaaS Deployment Service to deploy/undeploy the SaaS applications; (iv) Federation Management Service is responsible for supervision and reconfiguration of both PaaS and SaaS. In addition to a P2P(per-to-per) monitoring network application, they developed another two applications: a Distributed Complex Event Processing (DiCEPE) SaaS, this application integrates different complex event engines, also interacts with several remote communication protocols, and AntDroid that delivers it to scientists for sensing the activities of mobile users. Finally, these applications were deployed on thirteen cloud environments, validating both the service model and the generic kernel infrastructure.

OSGi

The Open Services Gateway initiative(OSGi) framework is a dynamic module system and service platform for java, defined by OSGi Alliance. OSGi components are called *bundles* containing java classes and configuration files. They can be dynamically installed, started, stopped, updated, and uninstalled on runtime.

Due to a dynamic and seamless deployment of OSGi applications, recent works [HangCan10], [Schmid09] have proposed the use of OSGi as the platform to build complex java applications for cloud infrastructures. In addition, OSGi Cloud Working Group has been studying ways to extend current OSGi standards to provide support for cloud computing. These efforts have been formalized in a new requirement document called RFP (Request for Proposal) 133 [Kriens11]. This document summarizes an introduction to cloud computing, relevance of OSGI to cloud, a problem description, and some usage cases. The OSGi key that supports cloud software is its modularity, because it contributes with its compositional approach based on modular software. Therefore, OSGi allows users to build elastic, scalable, and dependable software applications. RFP 133 details generic properties of software modules, such as its composable feature and its localized behaviour by separating both tightly-coupled and loosely-coupled interaction between modules into package dependencies and services concepts. RFP 133 describes as challenges the properties related to the benefits of components OSGi instead of virtual machine images. For example, re-deployment and upgrading of running systems are faster than application upgrades using virtual machines. Finally, RFP 133 details a series of problems to be addressed in future discussions, some of these relevant challenges are service discovery, monitoring, logging, security, accessibility, provisioning, discovery, and provision of resources.

OSGi and cloud computing have been discussed since 2010 on OSGi conferences. A particular point discussed was the replication of nodes containing services. When nodes are replicated, they need to have facilities to specify mul-

PUC-Rio - Certificação Digital Nº 0921330/CA

tiple destination hosts, then as to avoid all replicas being deployed. Therefore, there are many opportunities proposing OSGi as key technology to address current cloud challenges.

2.3 Final Remarks

In the first part of this chapter we reviewed the fundamentals of cloud computing. We studied its definition, taxonomy, platforms, APIs, and a partial software stack used in cloud computing. To create a cloud-based target environment we consider the following issues. (1) We use the IaaS to create cloud-based target environments because we require the provisioning of computational resources such as processing, storage, and networks. (2) To deploy the applications it is necessary to choose a cloud deployment model. The open-source cloud platforms allow us to set up our own private cloud, and then we can have full control over the cloud infrastructure. Public clouds usually have their own cloud platform and are ready to provide cloud resources, but we have limited access to the private cloud platform. Thus, we use both a private cloud and a public cloud. (3) In order to access the cloud infrastructure, we need a *cloud API*, libraries, and tools. We reviewed three generic cloud APIs(see sub-subsection 2.1.4), however we did not find a cloud API that fits our needs. We decided to develop our own cloud API to be integrated with our *deployment infrastructure* for two reasons. The first, we did not find a ready Lua-based cloud API implementation that supports an easy integration with our deployment infrastructure prototype. The second, although the APIs studied support the use of several cloud platforms, they do not allow specification of the resource requirements such as user-defined requirements and software installed on the virtual machine instanced. (4) The installation and configuration of a cloud platform is not an easy task, because it comprises a set of software layers. To support this set up we installed and configured our own private cloud.

In the second part, we studied a generic deployment process to describe deployment activities and their interactions with the deployment services. Also, we studied the deployment of applications on grid environments. We highlighted the FDF language of *DeployWare* because it allows users to define dynamic hosts.

Finally, we found a limited number of documents combining the deployment process and cloud infrastructures. FraSCAti is trying to solve main cloud computing challenges by proposing to develop multi-Cloud systems.

PUC-Rio - Certificação Digital Nº 0921330/CA