

## A novel reconfigurable by design highly distributed applications development paradigm over programmable infrastructure












### D2.1 - Description of Highly Distributed Applications and Programmable Infrastructure Requirements

<b>Editors:</b>	M. Repetto (CNIT), C. Vassilakis (UBITECH)
<b>Contributors:</b>	P. Gouvas, E. Fotopoulou, A. Zafeiropoulos (UBITECH), L. Tomasini (CNIT), K. Tsagkaris, N. Koutsouris (WINGS), S. Kovaci, G. Carella, T. Quang (TUB), A. Rossini (SINTEF), J. Sterle (UL), S. Siravo (MAGGIOLI), G. Kioumourtzis, L. Charalampous (ADITESS), L. Porwol (NUIG)
<b>Date:</b>	27/07/2015
<b>Version:</b>	1.00
<b>Status:</b>	Final
<b>Workpackage:</b>	WP2 – ARCADIA Framework Specifications
<b>Classification:</b>	Public

## ARCADIA Profile

<b>Grant Agreement No.:</b>	645372
<b>Acronym:</b>	ARCADIA
<b>Title:</b>	A NOVEL RECONFIGURABLE BY DESIGN HIGHLY DISTRIBUTED APPLICATIONS DEVELOPMENT PARADIGM OVER PROGRAMMABLE INFRASTRUCTURE
<b>URL:</b>	<a href="http://www.arcadia-framework.eu/">http://www.arcadia-framework.eu/</a>
<b>Start Date:</b>	01/01/2015
<b>Duration:</b>	36 months

## Partners

	Insight Centre for Data Analytics, National University of Ireland, Galway	Ireland
	Stiftelsen SINTEF	Norway
	Technische Universität Berlin	Germany
	Consorzio Nazionale Interuniversitario per le Telecomunicazioni	Italy
	Univerza v Ljubljani	Slovenia
	UBITECH	Greece
	WINGS ICT Solutions Information & Communication Technologies EPE	Greece
	MAGGIOLI SPA	Italy
	ADITESS Advanced Integrated Technology Solutions and Services Ltd	Cyprus

## Document History

Version	Date	Author (Partner)	Remarks
0.10	15/04/2015	M. Repetto (CNIT), C. Vassilakis (UBITECH)	<b>Preparation of Table of Contents (ToC)</b>
0.20	10/05/2015	P. Gouvas, E. Fotopoulou, (UBITECH), L. Tomasini (CNIT), N. Koutsouris (WINGS), G. Carella (TUB)	<b>Definition of ARCADIA Operational Environment - Section 2</b>
0.30	30/05/2015	M. Repetto (CNIT), C. Vassilakis, A. Zafeiropoulos (UBITECH), L. Tomasini (CNIT), N. Koutsouris (WINGS), G. Carella, T. Quang (TUB), J. Sterle (UL), S. Siravo (MAGGIOLI), G. Kioumourtzis, L. Charalampous (ADITESS), L. Porwol (NUIG)	<b>Primary description of Use Cases and SotA - Sections 3 and 4</b>
0.40	10/06/2015	M. Repetto (CNIT), C. Vassilakis, A. Zafeiropoulos (UBITECH), L. Tomasini (CNIT), N. Koutsouris (WINGS), G. Carella, T. Quang (TUB), J. Sterle (UL), S. Siravo (MAGGIOLI), G. Kioumourtzis, L. Charalampous (ADITESS), L. Porwol (NUIG)	<b>Final description of Use Cases and SotA - Sections 3 and 4</b>
0.50	30/06/2015	M. Repetto (CNIT), C. Vassilakis, P. Gouvas, E. Fotopoulou (UBITECH), L. Tomasini (CNIT), N. Koutsouris (WINGS), G. Carella (TUB), A. Rossini (SINTEF), J. Sterle (UL), S. Siravo (MAGGIOLI), G. Kioumourtzis (ADITESS)	<b>Primary version of requirements - Section 5</b>
0.50	15/07/2015	M. Repetto (CNIT), C. Vassilakis, P. Gouvas, E. Fotopoulou (UBITECH), L. Tomasini (CNIT), N. Koutsouris (WINGS), G. Carella (TUB), A. Rossini (SINTEF), J. Sterle (UL), S. Siravo (MAGGIOLI), G. Kioumourtzis (ADITESS)	<b>Final version of requirements - Section 5, Editing of Sections 1 and 6, Version for Internal review</b>
1.00	27/07/2015	M. Repetto (CNIT), C. Vassilakis, P. Gouvas, E. Fotopoulou, A. Zafeiropoulos (UBITECH), L. Tomasini (CNIT), K. Tsagkaris, N. Koutsouris (WINGS), S. Kovaci, G. Carella, T. Quang (TUB), A. Rossini (SINTEF), J. Sterle (UL), S. Siravo (MAGGIOLI), G. Kioumourtzis, L. Charalampous (ADITESS), L. Porwol (NUIG)	<b>Update based on comments from internal review - Final version</b>

## Executive Summary

This deliverable poses the foundation to start the technical activities in the ARCADIA project. In this respect, it sets the bases for a *common and shared vision of the problem* and provides *the basic set of design and implementation guidelines*. The purpose for a common vision is to agree on the problem definition, to share basic knowledge among partners from different fields (mainly software development, computing and networking), and to identify the main research challenges to be addressed. The guidelines to drive the project activities are expressed in terms of requirements. Requirements are meant to drive the design and development process; they are the constraints that will help the final framework to best match the initial vision and to satisfy the technological challenges.

The *vision* of ARCADIA is to provide a novel reconfigurable-by-design Highly Distributed Applications (HDAs) development paradigm over programmable infrastructure. Given the inability of Highly-Distributed-Application-Developers to foresee the changes as well as the heterogeneity on the underlying infrastructure, it is considerable crucial the design and development of novel software paradigms that facilitate application developers to take advantage of the emerging programmability of the underlying infrastructure and therefore develop Reconfigurable-by-Design applications. In parallel, it is crucial to design solutions that are scalable, support high performance, are resilient-to-failure and take into account the conditions of their runtime environment.

This technical deliverable depicts the general context, identifies relevant actors and technologies, and sets the main group of requirements that will drive the design of the ARCADIA framework.

The context is defined in terms of *operational environment*, which includes the relevant technologies and the roles that are involved in the software development and deployment process. This builds a reference scenario and enables to set a common terminology to be used by all partners. More in details, we define the basic terms that characterize the working environment, i.e., Highly Distributed Applications and Programmable Infrastructure. We state technological assumptions made by the project, which are used to set industry, backward-compatibility and standard compliance requirements. We also describe how the ARCADIA paradigm will affect deployment and execution of both legacy and ARCADIA-compliant applications. This description help understand what are the critical issues in the whole process, which is the preliminary step to identify system (high-level) requirements for the ARCADIA framework. Finally, we distinguish the main roles that are involved in the application lifecycles, starting from its design and implementation, to its deployment and operation. This description points out what users expect from the ARCADIA framework, thus allowing the identification of user requirements.

The *state of the art* briefly reviews solutions that are relevant to ARCADIA design and implementation, covering the full application provisioning chain, from development to execution. The analysis includes the software-development process (model-driven paradigms, context-awareness and in-code annotations), the execution environment (programmable infrastructure like cloud systems and software-defined networks), deployment and orchestration frameworks (configuration, replication, horizontal scaling). The state of the art provides a quick reference about current technological trends; this knowledge help identify what is already available to implement the ARCADIA framework and what is currently missing to achieve the project objectives. The state of the art settles implementation, assumption and constraint requirements, imposed by current and upcoming technology.

The *visionary scenario* is made of several use cases, which together explain how things should be improved with respect to current practice and what challenges need to be addressed to go beyond the

state of the art. Each use case focuses on one or more specific aspects; some use cases address issues for some ARCADIA roles, whilst other use cases point out the impact on the development process or the underlying infrastructure. The analysis of the relevant features highlighted by each use case enables to derive system (high-level) and functional requirements. To this aim, the layout used to describe each use case is conceived to draw attention to the main implication of each scenario about research challenges, advances with respect to the current state of the art, and relationships with the ARCADIA framework. It is worth noting that the indicative use cases described in this document are only meant to depict the project vision, and will not be implemented.

Different types of *requirements* are derived by considering the above aspects. Since the purpose of this document is to drive the following design and implementation stages, the whole set of requirements is clustered into homogeneous groups targeting different activities strands envisaged by the ARCADIA framework: distributed software development, programmable infrastructure management, distributed applications profiling, deployment and orchestration, optimization. The description of each requirement follows a template that includes a short title, the role it concerns, the full description, any type of constraint imposed, its priority, the architectural part of the ARCADIA framework it affects, and possible notes. We use three levels of priority (top, medium, low); the highest priority denotes requirements that must be satisfied by the framework to meet the original expectations and objectives, whereas the bottom priority denotes requirements that are desirable to be present but do not preclude the correct functioning of the system.

The structure of this deliverable is as follows. *Section one* introduces the main context of the project and describes the purpose, motivation and role of this document in the ARCADIA framework. *Section two* outlines the ARCADIA operational environment by defining the concepts of Highly Distributed Application and Programmable Infrastructure, by providing an initial view on the main ARCADIA architectural components, by identifying the main actors and their roles, and by sketching how the framework works with legacy and novel applications. *Section three* reviews the current state of the art; it is organized in three subsections, covering development of distributed applications, programmable infrastructure and applications profiling, deployment and orchestration. *Section four* provides a better understanding of the general problem and the benefits that the ARCADIA framework will bring, by listing a set of use cases that show how the process of developing and deploying applications could change and what opportunities will be available through the novel paradigms. *Section five* lists the requirements identified so far, to design, develop and operate the ARCADIA framework; it is roughly organized to reflect the main activities that will be carried out by the consortium, and it distinguishes among requirements for distributed software development, programmable infrastructure management and application profiling, deployment, orchestration, optimization, and management. Finally, *section six* remarks on the main outcomes from this deliverable and gives recommendations for the following activities.

## Table of Contents

<b>1</b>	<b>Introduction.....</b>	<b>10</b>
1.1	Purpose and Scope.....	11
1.2	Methodology .....	11
1.3	Relation with other WPs.....	12
<b>2</b>	<b>ARCADIA Operational Environment.....</b>	<b>13</b>
2.1	Highly Distributed Applications Definition.....	13
2.2	Programmable Infrastructure and Applications Hosting Environment Definition ....	14
2.3	Highly Distributed Applications Deployment and Operation.....	15
2.4	ARCADIA Architectural Components Vision.....	15
2.5	ARCADIA Ecosystem Roles .....	17
<b>3</b>	<b>State of the Art Analysis and Beyond .....</b>	<b>18</b>
3.1	Distributed Software Development Paradigms .....	18
3.1.1	akka.....	20
3.1.2	Quasar .....	21
3.2	Programmable Infrastructure.....	21
3.2.1	Programmable Infrastructure Building Blocks.....	22
3.2.1.1	Application Execution Environment.....	22
3.2.1.2	Application Networking Environment .....	27
3.2.2	Cloud Infrastructure Platforms and Orchestration Frameworks.....	33
3.2.2.1	Cloud Computing Platforms .....	34
3.2.2.2	SDN/NFV Platforms .....	37
3.2.2.3	Other Orchestration Frameworks .....	39
3.3	Applications Profiling, Deployment and Orchestration.....	40
3.3.1	Application Profiling .....	40
3.3.1.1	Application Profiling, Load testing, Stress testing.....	41
3.3.1.2	Virtualization overhead .....	41
3.3.1.3	Auto-scaling techniques.....	42
3.3.2	Application deployment and orchestration .....	43
3.3.2.1	Application Scalability.....	44
3.3.2.2	Programmable infrastructure.....	45
3.3.2.3	Highly Distributed Application Embedding (HDAE) problem.....	46
3.3.2.4	Related Work to the HDA embedding problem .....	47
3.3.2.5	Deployment Scheduling Approaches in Platforms.....	50
<b>4</b>	<b>ARCADIA Use Cases .....</b>	<b>51</b>
4.1	Energy Efficient Cloud Management .....	51
4.2	Annotated Networking.....	52

4.3	Remote Surveillance .....	53
4.4	Enterprise Networking.....	54
4.5	IoT/Smart Home.....	56
5	Highly Distributed Applications and Programmable Infrastructure Requirements .....	58
5.1	Distributed Software Development Paradigm Requirements.....	58
5.2	Programmable Infrastructure Management Requirements .....	61
5.3	Distributed Applications Deployment and Orchestration Requirements.....	69
5.3.1	Distributed Applications Profiling and Optimization Requirements .....	74
6	Conclusions .....	78
	Annex I: References .....	79

## List of Figures

Figure 1-1: Relationship of D2.1/Task 2.1 with other Tasks in ARCADIA .....	13
Figure 2-1: Highly Distributed Application Indicative Breakdown.....	14
Figure 2-2: ARCADIA Architectural Components Vision .....	16
Figure 3-1: Monolithic vs Microservices Architecture.....	18
Figure 3-1: Characteristics according the application running environment.....	23
Figure 3-2: Source [16] LXC versus Docker.....	27
Figure 3-3: SDN architecture.....	28
Figure 3-4: NFF framework architecture .....	30
Figure 3-5: NFV framework break down .....	30
Figure 3-6: Source [34] Floodlight Openflow controller .....	31
Figure 3-7: Source [5] Openstack.....	34
Figure 3-8: Source [38] OpenDaylight architecture.....	37
Figure 3-9: Source [41] OPNFV Arno Overview Diagram.....	38
Figure 3-10: Source [73] Virtual Network Embedding (VNE) example .....	47
Figure 3-11: Source [73] Virtual Network Embedding (VNE) example with energy efficiency as objective.....	47
Figure 3-12: Source [73] An example of reconfiguration in Virtual Network Embedding (VNE) .....	48
Figure 3-13: Source [84] Virtual Data Centre Embedding (VDCE) example .....	49
Figure 3-14: Source [85] Cloud Application Embedding (CAE) example .....	50
Figure 5-1: Properties of Reactive Systems based on the Reactive Manifesto .....	59

## List of Tables

Table 3-1: Source [10] Comparison of popular Hypervisors .....	24
Table 3-2: List of available SDN software switches. ....	32
Table 3-3: List of commercial switches compliant with the OpenFlow protocol .....	32
Table 3-4: List of controllers compliant with the OpenFlow standard.....	32
Table 3-5: Comparison of Cloud Platforms.....	36



## Acronyms

<b>API</b>	Application Programming Interface
<b>CAE</b>	Cloud Application Embedding
<b>DoW</b>	Description of Work
<b>HDA</b>	Highly Distributed Application
<b>IaaS</b>	Infrastructure as a Service
<b>JVM</b>	Java Virtual Machine
<b>LXC</b>	Linux Container
<b>NFV</b>	Network Function Virtualization
<b>NFVI</b>	Network Functions Virtualization Infrastructure
<b>NV</b>	Network Virtualization
<b>OS</b>	Operating System
<b>PM</b>	Physical Machine
<b>PoP</b>	Point of Presence
<b>QoS</b>	Quality of Service
<b>SDN</b>	Software Defined Networking
<b>VDCE</b>	Virtual Data Centre Embedding
<b>VLAN</b>	Virtual Local Area Network
<b>VNE</b>	Virtual Network Embedding
<b>VNF</b>	Virtual Network Function
<b>VPN</b>	Virtual Private Network
<b>WP</b>	Work Package

## 1 Introduction

The generalized trend towards massive “softwarization” of processes, devices, services and infrastructures has pointed out many limitations of the current practice to develop, deploy and run software applications. In particular, human resources are often overworked with an intrinsic infrastructural heterogeneity, namely a diversity of the programming frameworks and the execution environments that are utilized in the application lifecycle. A transition from the ‘intelligent design’ approach, which currently rules software engineering, to meta-design approaches as well as self-combining software systems has to be realized. To this aim, focus should be given on the design of software components that have the ability to collaborate in an autonomous and decentralized fashion [86]. In particular, a set of considerations about quick development times, software re-utilization, data locality and so forth have brought the concept of *Highly Distributed Applications* (HDA), which run on a global heterogeneous infrastructure built on top of the “Future Internet”.

Key drivers that boost this transition are emerging paradigms like virtualization and the availability of programmable infrastructures. However, the plethora of different solutions and approaches has led to a thicket of execution environments and their relative configuration artifacts, exacerbating the difficulty of software engineers to quickly adapting their systems to different run-time contexts.

Following the basic principles under the DevOps approach, an increasing interest has been devoted to include more “context-awareness” into the very same applications and services, by making them able to adapt and to adjust their behavior to different run-time environments, relying on the autonomic provisioning capability allowed by the large availability of programmable infrastructure. However, this must not turn into an overwhelming configuration burden for developers, rather it should be an opportunity to exploit the peculiarities of each execution environment and hence to optimize performance, availability, dependability, security, and cost of the applications.

Under this perspective, the vision of ARCADIA is to provide a novel reconfigurable-by-design Highly Distributed Applications’ development paradigm over programmable infrastructure. The approach relies on an extensible Context Model that will assist programmers to take into account the heterogeneity and peculiarity of the underlying infrastructure, and a Smart Controller that will undertake the tasks of optimal and dynamic deployment of applications over multiple domains starting from the instantiation of the Context Model.

This document is the first technical deliverable of the project and is committed to depict the general context, to identify relevant actors and technologies, and to set the main group of requirements that will drive the design of the ARCADIA framework. Section one introduces the main context of the project and describes the purpose, motivation and role of this document in the ARCADIA framework. Section two outlines the ARCADIA operational environment by defining the concepts of Highly Distributed Application and Programmable Infrastructure, by providing an initial view on the main ARCADIA architectural components, by identifying the main actors and their roles, and by sketching how the framework works with legacy and novel applications. Section three reviews the current state of the art; it is organized in three subsections, covering development of distributed applications, programmable infrastructure and applications profiling, deployment and orchestration. Section four provides a better understanding of the general problem and the benefits that the ARCADIA framework will bring, by listing a set of use cases that show how the process of developing and deploying applications could change and what opportunities will be available through the novel paradigms. Section five lists the requirements identified so far, to design, develop and operate the ARCADIA framework; it is roughly organized to reflect the main activities that will be carried out by the consortium, and it distinguishes among requirements for distributed software development, programmable infrastructure management and application profiling, deployment, orchestration,

optimization, and management. Finally, section six remarks on the main outcomes from this deliverable and gives recommendations for the following activities.

## 1.1 Purpose and Scope

This document poses the foundation to start the technical activities in the ARCADIA project. In this respect, it sets the bases for a *common and shared vision of the problem* and provides *the basic set of design and implementation guidelines*.

The purpose for a common vision is to agree on the problem definition, to share basic knowledge among partners from different fields (mainly software development, computing and networking), and to identify the main research challenges to be addressed.

The guidelines to drive the project activities are expressed in terms of requirements. Requirements are meant to drive the design and development process; they are the constraints that will help the final framework to best match the initial vision and to satisfy the technological challenges. Requirements show the functional and the non-functional aspects for the particular project and are an important input to the verification process, since tests should trace back to specific requirements.

## 1.2 Methodology

The definition of requirements is a critical aspect for every project, since they affect the final outcomes of the project and its adherence with the initial purposes and specification. To be sure to identify the right set of requirements, the methodology adopted by ARCADIA has put a great effort to agree on a common understanding of the main problem addressed by the project. To this aim, the relevant aspects are the *specific context* targeted by ARCADIA, the *project vision*, and the current *state of the art*. The content of this document has emerged from various sources:

- the *Description of the Action* (annex to the Grant Agreement), which describes the main purpose of the project, the technological context and current practice, open issues and limitations of current solutions, and the use cases originally proposed to demonstrate the project;
- the *Project Meetings* held during the first semester:
  - the kick-off meeting in Athens, on January 26<sup>th</sup>-27<sup>th</sup>, 2015;
  - the 1<sup>st</sup> plenary meeting in Berlin, on May 19<sup>th</sup>-20<sup>th</sup>, 2015;
  - periodic virtual meetings, held every two-three weeks on average.

The context is defined in terms of **operational environment**, which includes the relevant technologies and the roles that are involved in the software development and deployment process. This builds a reference scenario and enables to set a common terminology to be used by all partners. More particularly, we define the basic terms that characterize the working environment, i.e., Highly Distributed Applications and Programmable Infrastructure. We state technological assumptions made by the project, which are used to set industry, backward-compatibility and standard compliance requirements. We also describe how the ARCADIA paradigm will affect deployment and execution of both legacy and ARCADIA-compliant applications. This description help understand what are the critical issues in the whole process, which is the preliminary step to identify system (high-level) requirements for the ARCADIA framework. Finally, we distinguish the main roles that are involved in the application lifecycles, starting from its design and implementation, to its deployment and operation. This description points out what users expect from the ARCADIA framework, thus allowing the identification of user requirements.

The **state of the art** briefly reviews solutions that are relevant to ARCADIA design and implementation, covering the full application provisioning chain, from development to execution. The analysis includes the software-development process (model-driven paradigms, context-awareness and in-code annotations), the execution environment (programmable infrastructure like cloud systems and software-defined networks), deployment and orchestration frameworks (configuration, replication, horizontal scaling). The state of the art provides a quick reference about current technological trends; this knowledge help identify what is already available to implement the ARCADIA framework and what is currently missing to achieve the project objectives. The state of the art settles implementation, assumption and constraint requirements, imposed by current and upcoming technology.

The **visionary scenario** is made of several use cases, which together explain how things should be improved with respect to current practice and what challenges need to be addressed to go beyond the state of the art. Each use case focuses on one or more specific aspects; some use cases address issues for some ARCADIA roles, whilst other use cases point out the impact on the development process or the underlying infrastructure. The analysis of the relevant features highlighted by each use case enables to derive system (high-level) and functional requirements. To this aim, the layout used to describe each use case is conceived to draw attention to the main implication of each scenario about research challenges, advances with respect to the current state of the art, and relationships with the ARCADIA framework. It is worth noting that the indicative use cases described in this document are only meant to depict the project vision, and will not be implemented.

Different types of **requirements** are derived by considering the above aspects. Since the purpose of this document is to drive the following design and implementation stages, the whole set of requirements is clustered into homogeneous groups targeting different activities strands envisaged by the ARCADIA framework: distributed software development, programmable infrastructure management, distributed applications profiling, deployment and orchestration, optimization. The description of each requirement follows a template that includes a short title, the role it concerns, the full description, any type of constraint imposed, its priority, the architectural part of the ARCADIA framework it affects, and possible notes. We use three levels of priority (top, medium, low); the highest priority denotes requirements that must be satisfied by the framework to meet the original expectations and objectives, whereas the bottom priority denotes requirements that are desirable to be present but do not preclude the correct functioning of the system.

### 1.3 Relation with other WPs

This deliverable is the outcome from Task 2.1 – Highly Distributed Application and Programmable Infrastructure Requirements, and represents the preliminary step towards the ARCADIA framework specifications (WP2). It builds the ground of knowledge to understand the basic problem statement and the main guidelines to start the other technical tasks. The vision depicted by the use cases described in this document is shared with Task 2.3 – Smart Controller Requirements and Functionalities, to extend the general set of requirements already derived in Task 2.1. Task 2.3 specifically looks at the Smart Controller, because the latter is a cornerstone in the ARCADIA framework; instead, the scope of this report is more general and takes into consideration the whole system. This deliverable directly feeds Task 2.2 – Definition of ARCADIA Context Model, and Task 2.4 – Design of ARCADIA framework. Figure 1-1 positions D2.1 and Task 2.1 in the global ARCADIA framework, with explicit indication of what tasks are fed by their outputs.

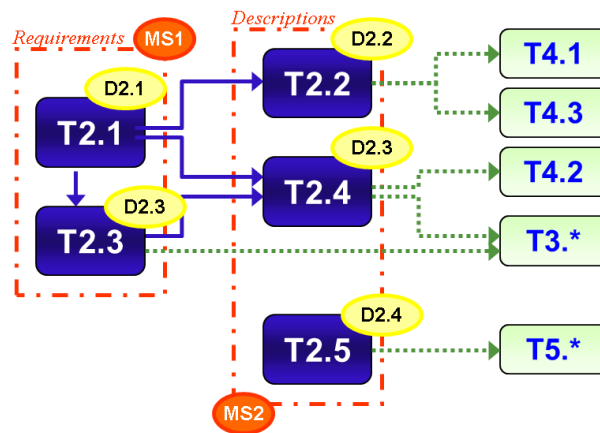


Figure 1-1: Relationship of D2.1/Task 2.1 with other Tasks in ARCADIA

## 2 ARCADIA Operational Environment

### 2.1 Highly Distributed Applications Definition

A Highly Distributed Application (HDA) is defined as a distributed scalable structured system of software entities constructed to illustrate a network service when implemented to run over a cloud infrastructure. An HDA is a multi-tier cloud application consisting of application's tiers chained with other software entities illustrating network functions applied to the network traffic towards/from and between application's tiers. Each software entity provides the ability to horizontally scale (in and out) during runtime in order to handle the workload using needed resources.

An indicative HDA is depicted in Figure 2-1 that corresponds to a graph that contains a set of tiers along with a set of functions implemented in the form of Virtual Network Functions (VNFs). It should be noted that in ARCADIA we are going to adopt the term Virtual Functions (VFs) instead of the term VNF that is denoted in ETSI Network Function Virtualization (NFV) [26] since we do not only refer to networking functions but to generic functions. Each element in the graph is accompanied with a set of characteristics and constraints (e.g. resource capacity constraints, dependencies).

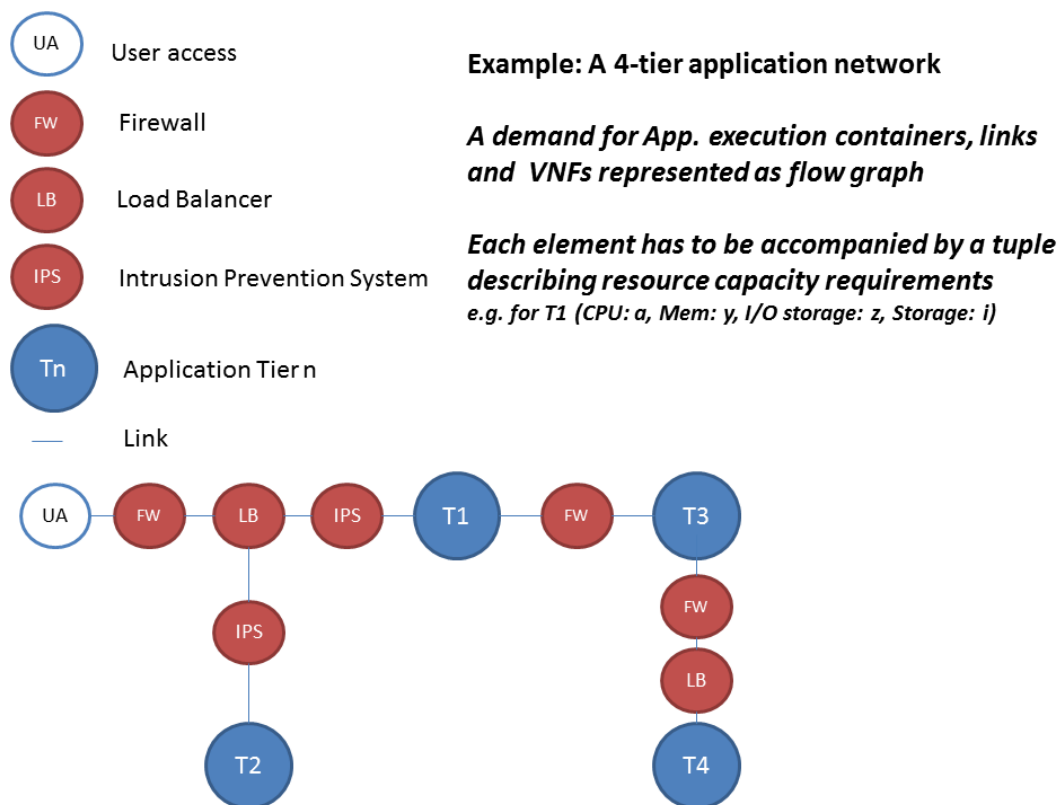


Figure 2-1: Highly Distributed Application Indicative Breakdown

## 2.2 Programmable Infrastructure and Applications Hosting Environment Definition

With the term programmable infrastructure we refer to the networking, computational and storage infrastructure with programmable characteristics that is being used for the deployment and operation of the ARCADIA applications. It should be noted that deployment of the applications will be realized – in most of the cases- on virtual resources (e.g. virtual machines, Linux containers), while management of resources will be supported across multiple IaaS (Infrastructure as a Service) environments. In specific cases, where direct access to hardware is required for management of programmable aspects of devices, specific bare metal services are going to be supported.

With regards to the deployment/execution environment of the considered applications, we are going to refer to the following cases:

- applications running on native Operating System (OS) in case of applications running on an OS of a Physical Machine (PM);
- applications running on a Container in case of applications running on a hosted Container in an OS of a PM;
- applications running on a Virtual Machine (VM) in case of applications running on the OS of the VM that is hosted by a hypervisor, and
- applications running in further nested environments (mostly for testing purposes, e.g. applications running in a Container in an OS of a VM hosted by a hypervisor of a PM).

Illustration of a virtual network among executing environments and middleboxes will be facilitated by technologies developed in the highly convoluted areas [20] of Network Virtualization (NV), Software Defined Networking (SDN) and Network Functions Virtualization (NFV).



## 2.3 Highly Distributed Applications Deployment and Operation

In ARCADIA, we support the deployment and operation of:

- ARCADIA applications; applications that are going to be developed following the proposed software development paradigm,
- Legacy applications; existing applications already available in an executable form, and
- Hybrid applications; applications consisting of application tiers from both the afore-mentioned cases.

In all the cases, a service chaining graph has to be produced and used for the preparation of the deployment scripts. The service chaining graph regards a graph denoting the workflow that has to be realised towards the execution of an application, as already shown in Figure 2-1. Based on the service chaining graph, each application is broken down into a set of micro-apps or micro-services with dependencies among each other.

In case of applications that are developed based on the ARCADIA software development paradigm, the service chaining graph will be partially or fully denoted within the software. Furthermore, a set of monitoring hooks that are going to be used during the deployment as well as the execution time of the application for optimisation purposes will be provided. In case of existing applications, the deployment script has to be created by a DevOps user. In case of hybrid applications, indications for the service chaining graph may be provided within the software; however the final deployment script has to be provided by the DevOps user.

Following, the produced deployment script (in automated or manual way) is provided to the ARCADIA Smart Controller that is responsible for realising the initial deployment and management of the application during the execution time over the available programmable infrastructure. The application's software components –as denoted in the corresponding service chain- are instantiated on demand. The defined monitoring hooks initiate a set of monitoring functionalities for specific performance metrics. The status of these metrics trigger re-configurations in the deployed application based on optimisation objectives (as denoted by application developers, service providers, infrastructure owners) along with a set of constraints that are considered during the application deployment and runtime. Resources reservation and release is realized on demand over the programmable infrastructure.

## 2.4 ARCADIA Architectural Components Vision

The vision of ARCADIA is to provide a novel reconfigurable-by-design Highly Distributed Applications (HDAs) development paradigm over programmable infrastructure. An initial version of the ARCADIA architectural components vision is provided in

Figure 2-2. Based on the architectural approach that is going to be designed, the developer shall be able to develop infrastructural agnostic applications by following proper development architectural patterns.

Considering that a developer complies with HDA development patterns, according to the ARCADIA flow, he/she will use a specific ARCADIA-IDE Plugin in order to provide specific annotations at the source-level. These annotations will be instantiations of the ARCADIA Context Model. The ARCADIA Context Model will conceptualize application, configuration and infrastructural aspects that should be taken under consideration in order for an optimal infrastructural configuration to be defined. According to the conceptual flow, this optimal configuration should be created and instantiated by the component which is called Smart Controller.

The Smart Controller is (among others) the HDA's on-boarding utility which will undertake the tasks of i) translating annotations to optimal infrastructural configuration (through the Configuration Manager) ii) initialize the optimal configuration to the registered programmable resources (through the Deployment Manager) and iii) react pro-actively to the configuration plan based on the infrastructural state and the HDA state (through the Autonomic Prediction and Reconfiguration Engine).

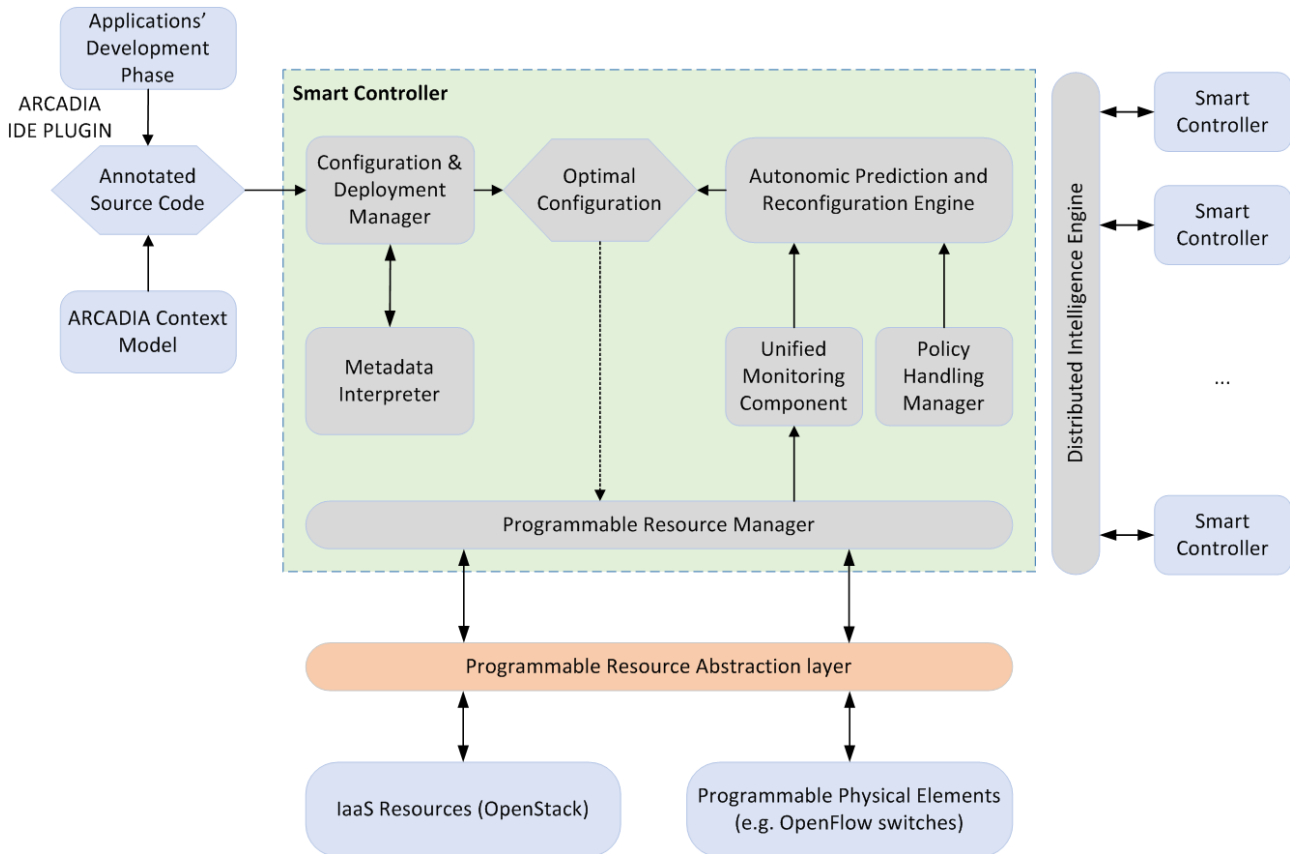


Figure 2-2: ARCADIA Architectural Components Vision

As already stated, the ARCADIA framework aims to radically change the way HDAs are developed, on-boarded, deployed and managed. After the compilation of an HDA component, the executable will be on-boarded to the Smart Controller which is the cornerstone of the ARCADIA framework. Architecturally, its main sub-modules include:

- a) an **ARCADIA-metadata interpreter** which is able to interpret metadata that accompany compiled executables and produce deployment scripts. The interpretation is crucial because it will affect the deployment and management lifecycle. The interpretation will guide the Smart Controller's behaviour during the entire lifecycle of the HDA.
- b) a **Programmable Resource-Manager** which exposes a specific interface where programmable resources are registered and managed. Programmable resources can span from configured IaaS frameworks, programmable physical switching/routing equipment, programmable firewalls, application servers, modularized software entities (databases, HTTP proxies etc).



- c) a **Configuration and Deployment Manager** which will undertake the complex task of mapping source-code metadata to configuration and infrastructural requirements. These requirements have to be 'translated' in optimal configuration taking under consideration i) the registered resources, ii) their programmability capabilities and iii) the existing policies.
- d) a **Unified Monitoring Manager** which will rely on proper probes that will be configured during the deployment phase. Probing is related to active monitoring techniques. Such techniques can be used to monitor in a near-real-time fashion metrics in multiple levels e.g. OS-level (memory, cores etc), application-server-level (connection pools, queues, etc) or application-level (heap, stack etc). However, the Unified Monitoring Manager will also employ passive techniques in order to aggregate measurements from Resources that cannot be probed; yet they can be interfaced through their custom API. Indicative examples are switching and routing devices, firewalls etc. Metrics that are measured using both techniques will be aggregated and used from the "Autonomic Prediction and Reconfiguration Engine" (analysed below).
- e) an **Autonomic Prediction and Reconfiguration Engine** which will be responsible for pro-active adjustment of the running configuration based on measurements that derive from the Unified Monitoring Manager. The ultimate goals of this component are two: i) zero-service disruption ii) re-assure optimal configuration across time. In order to achieve both of these goals predictive algorithms will be employed, which will eliminate, as much as possible, false-positives, regarding triggering of re-configuration.
- f) a **Distributed Intelligence Engine** that undertakes the task of communicating in a peer-to-peer manner with other Smart Controllers in order to cope with limitations, barriers or run-time problems that are caused by many reasons such as limited availability of physical Resources, run-time performance issues etc. Distributed Intelligence is accompanied by many challenges that have to be confronted. Indicative ones include Security and Trust among Smart-Controllers, Distributed Inference and Decision making, Lack of Centralized Control etc.
- g) a **Policy Handling Manager** which will be responsible for defining the high level policies on behalf of the services provider. These policies are by definition multidisciplinary; since they can be affected by many requirements.

## 2.5 ARCADIA Ecosystem Roles

Within the ARCADIA ecosystem, the following roles are identified:

- **Software Developer:** he develops applications based on the ARCADIA software development paradigm or adapts existing applications in order to incorporate, partially or fully, concepts based on the ARCADIA software development paradigm.
- **DevOps User:** he prepares the deployment scripts of existing applications that are not developed based on the ARCADIA software development paradigm but include set of monitoring hooks as well as the notion of service chaining. Based on the produced deployment script, optimal deployment and management of such applications can be supported. For applications developed based on the ARCADIA software development paradigm, seamless integration of different kinds of DevOps artifacts is supported, thus the role of DevOps user is limited (in cases where customizations are required).
- **Smart Controller:** it deploys the applications over the available programmable infrastructure, manages the application during the execution time triggering re-configurations where required based on the defined optimization objectives on behalf of the application developer and the services provider. It provides a set of monitoring functionalities and mechanisms for monitoring in real time the parameters that are provided as input to the optimization

framework. It also manages the available programmable infrastructure, including the management of resources that belong to multi-IaaS environments.

- **Services Provider:** it defines policies and optimization objectives on behalf of the services provider (e.g. communication services provider, cloud computing services provider). These policies along with their prioritization are handled by the Smart Controller towards the deployment/management of applications.
- **IaaS Provider:** it provides interfaces to the Smart Controller for management of large pools of compute, storage, and networking resources. Registration of resources can be realized through a multi-IaaS environment.
- **Arcadia Administrator:** he develops and maintains software libraries of the ARCADIA software development paradigm as well as facets of the ARCADIA context model.
- **Risk Manager:** he identifies emerging risks based on the adoption of ARCADIA-based solutions and proposes a risk management framework for handling unforeseen events (e.g. physical disaster events).

### 3 State of the Art Analysis and Beyond

#### 3.1 Distributed Software Development Paradigms

The evolvement of new software development paradigms is following the need for development of applications that highly include the notion of modularity, distribution, scalability, elasticity and fault tolerance. Actually, we refer to an evolution from applications that are based on **monolithic architectures** to applications that can be represented as **re-active systems** composed by **micro-services**. Micro-services can be considered as the resulting set of services that arise from the process of decomposing an application into smaller pieces (Figure 3-1). Furthermore, we refer to applications that have to be deployed and executed over **heterogeneous environments** –in terms of underlying infrastructure and end users devices- as well as applications that have to take into account **strict constraints** in terms of performance (e.g. millisecond response time, 100% uptime etc., as also specified in the 5G networks evolution requirements).

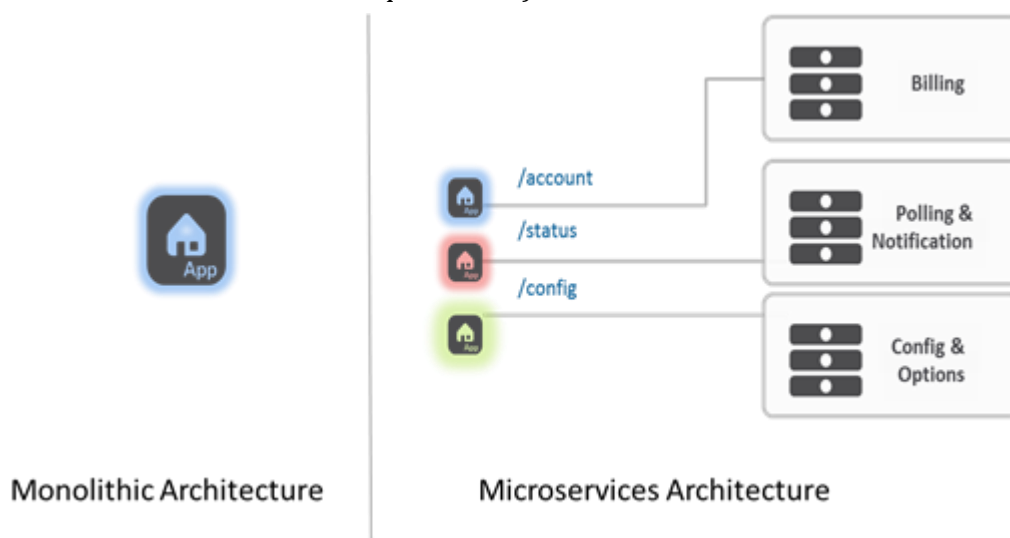


Figure 3-1: Monolithic vs Microservices Architecture<sup>1</sup>

<sup>1</sup> <https://devcentral.f5.com/articles/microservices-versus-microsegmentation>

Such an evolution is accompanied with the current trend in the design nature of applications that are consisted –following an increasing tendency- of distributed components that have to interact among each other and have to react based on events that are initiated in their environment. Such applications cannot be easily and effectively developed based on sequential programming paradigms, since the execution flow of the components, as well as the dynamicity in their instantiation and operation, cannot be predicted or represented on a sequential flow. Furthermore, it is really hard to support stateful mechanisms in such cases, since in case of a state change all the associated components have to be informed at real time, without negative impact in the overall application performance [94].

In case of using traditional programming solutions (such as design patterns and event-driven programming), interactive applications are typically constructed around the notion of asynchronous callbacks (event handlers) [93]. However, in this case, a set of problems may arise. For instance, numerous isolated code fragments can be manipulating the same data and their order of execution is unpredictable, thus causing non-desirable effects at execution time. Furthermore, since callbacks usually do not have a return value, they must perform side effects in order to affect the application state [95][96].

In order to be able to develop efficient **distributed applications** with high notion of reactivity, the **reactive programming paradigm** has been recently proposed as a solution that is well-suited for developing **event-driven applications**. Reactive programming tackles issues posed by event-driven applications by providing abstractions to express programs as reactions to external events and having the language automatically manage the flow of time (by conceptually supporting simultaneity), and data and computation dependencies. Thus, programmers do not need to worry about the order of events and computation dependencies.

Reactive programming supports the development of reactive applications through dedicated language abstractions. It is based on concepts like time-varying values (a.k.a. signals or behaviors), events streams to model discrete updates, automatic tracking of dependencies, and automated propagation of change [90][93]. Reactive programming means changes can be made without worrying about logic ordering. When implemented correctly, reactive programming lets software components take care of dependency management rather than leaving that work to the programmers.

As already stated, the momentum for the adoption of reactive programming approaches has also been fortified by the need to transit from stateful to **stateless approaches** in order to increase the **scalability** of the provided services and applications. Nodes have to be able to be added or removed during runtime, independently if they are related with the same process or not, the same physical machine or not, or even if they are in a completely different point of presence (e.g. data center). Failures are also handled in an automated way, since they can be encapsulated as messages and sent off to another part of the system that has the ability to deal with it properly.

Regarding the evolvement of reactive programming approaches, it should be noted that functional reactive programming was firstly introduced in Haskell to support interactive animations, while its popularity has been increased through its adoption in Scheme, Javascript and Scala. Concepts inspired by reactive programming have been also applied to Microsoft Reactive Extensions and stimulated a significant number of novel popular front-end libraries (e.g. React.js,) [87-92].

Given the transition to reactive programming approaches and the development of highly reactive distributed applications, it should be noted that -in addition to the development of novel applications decomposed in a set of microservices- a set of DevOps operations have to be also planned for supporting the real-time monitoring and management of such microservices. In ARCADIA, such operations are going to be supported in an autonomic or semi-autonomic way depending on the type

of the considered application (e.g. autonomic preparation of deployment script in case of adoption of the ARCADIA software development paradigm for a new application).

*In the following, two representative reactive programming toolkits and libraries; akka and quasar, are presented in brief.*

### 3.1.1 akka

**akka** [97] is a toolkit and runtime for building highly concurrent, distributed, resilient, message-driven applications on the Java Virtual Machine (JVM) [98]. The core characteristics supported are:

- Simple Concurrency & Distribution
  - Asynchronous and Distributed by Design
  - High-level abstractions like Actors<sup>2</sup>, Streams and Futures
- Resilient by Design
  - Write systems that self-heal
  - Remote and local supervisor hierarchies
- High Performance
  - 50 million msg/sec on a single machine
  - Small memory footprint; ~2.5 million actors per GB of heap
- Elastic & Decentralized
  - Adaptive cluster management
  - load balancing
  - routing
  - partitioning and sharding
- Extensible
  - Support of Akka Extensions to adapt Akka to fit any needs

Akka decouples business logic from low-level mechanisms such as threads, locks and non-blocking I/O, and liberates developers from the hardest challenges of managing the state and location of services. It is an Actor-based runtime for managing concurrency, elasticity and resilience on the JVM with support for both Java and Scala.

Akka can be used in two different ways; as a library: used by a web app and as a microkernel: stand-alone kernel to drop your application into.

---

<sup>2</sup> “Actors are objects which encapsulate state and behavior, they communicate exclusively by exchanging messages which are placed into the recipient’s mailbox. In a sense, actors are the most stringent form of object-oriented programming, but it serves better to view them as persons: while modeling a solution with actors, envision a group of people and assign sub-tasks to them, arrange their functions into an organizational structure and think about how to escalate failure (all with the benefit of not actually dealing with people, which means that we need not concern ourselves with their emotional state or moral issues). The result can then serve as mental scaffolding for building the software implementation” [97].

### 3.1.2 Quasar

**Quasar** [99] is an open source JVM library that simplifies the creation of highly concurrent software that is easy to write and reason about, performant, and fault tolerant.

Quasar's core implements true lightweight threads on the JVM called fibers [100]. Fibers can be instantiated and run just like regular threads, but rather than a few thousand of threads, a single JVM can easily run hundreds of thousands or even millions of fibers. Fibers enjoy the scalability and performance benefits of asynchronous (callback-based) programming while still maintaining the simplicity, intuitiveness and familiarity of threaded code. Fibers use channels (CSP), data-flow variables, or other forms of coordination, to communicate with one another efficiently. On top of fibers, Quasar provides an actor framework that strongly resembles Erlang's. Actors are a simple and natural way to implement scalable and fault-tolerant business logic. Quasar's main features supported are:

- True, preemptively scheduled, lightweight threads (fibers)
- Scalability and performance without complex, hard-to-maintain code
- Non-obtrusive integration – use just what you need
- High-performance channels for a CSP (Communicating Sequential Processes) model
- An Erlang-like actor framework
- Selective receive in actors
- Actor supervisors
- Hot code swapping
- Runtime monitoring: exposes metrics about fiber and actor performance and health

Quasar library provides high-performance lightweight threads, Go-like channels, Erlang-like actors, and other asynchronous programming tools for Java and Kotlin.

## 3.2 Programmable Infrastructure

There are two complementary aspects that we must consider, stemming from different perspectives, needs and roles of the relevant actors. On the one hand, for developers programmability is the mean to create the proper execution environment independently of the underlying physical resources. They need both overarching resource abstractions at the design/development stage and convenient APIs at run-time, in order to implement their application in an environment-agnostic way and to dynamically tailor them to the actual (and usually changing) context. To this aim, the **Programmable Infrastructure** provides developers with a common and single point of access to all resources, hiding physical issues like resource nature, faults, maintenance operations, and so on. On the other hand, resource owners are mostly concerned with operation and maintenance of (usually) large pools of resources. They need handy tools to deal with typical management tasks like insertion, replacement, removal, upgrade, restoration and configuration with minimal service disruption and downtimes. To this aim, a high degree of automation is desirable, through programmatic recourse to self-\* capabilities (self-tuning, self-configuration, self-diagnosis, self-healing).

**Cloud computing** implements a Programmable Infrastructure by providing users with (virtual) resources on demand, according to their need, and by metaphorically blurring the real physical infrastructure inside an opaque “cloud”. The kind of resources exposed by clouds depends upon the specific service model; they are infrastructural elements like (virtual) hosts, storage space, network devices (Infrastructure-as-a-Service model, IaaS), computing platforms including the Operating

System and a running environment (Platform-as-a-Service model, PaaS), or application software like databases, web servers, mail servers (Software-as-a-Service model). In ARCADIA, we will mainly target the IaaS model, since it gives developers the broadest control on the execution environment for their applications.

**Software Defined Networking (SDN)** is a new paradigm based on **network programmability** that has been proposed to overcome the many limitations and shortcomings of the legacy model based on configuration of network devices: reduced set of instructions, difficulty in adapting to varying conditions and in responding to a wide range of network events and applications, vertically-integrated network solutions, scarce interoperability and homogeneity among configuration interfaces, uncertainly results due to errors or different human skills.

**Network function virtualization (NFV)** is about all those network services that have been delivered till now as hardware appliances (firewalls, load-balancers, application delivery controllers, etc.) to become virtualized and run on standard hardware. Virtualizing network functions offers flexibility and several benefits similar to those of platform virtualization while all these virtualized network functions (VNF) will take advantage of programmability features to enable service-chain automation.

Though the aforementioned are different architectures, frameworks and implementations, these approaches are interrelated and their synergy towards a fully programmable infrastructure is more and more evident in today's platforms.

In the following, the building blocks of a programmable infrastructure are described and categorised while representative existing solutions are presented. Then, representative platforms are presented which gather most of the attention, efforts and large scale deployments along with some newest platforms expected to gain significant attention.

### 3.2.1 Programmable Infrastructure Building Blocks

In the following, the various alternative execution environments for an HDA are analyzed and compared along with technologies facilitating illustration of its network environment. These technologies are the candidate building blocks of a programmable infrastructure platform capable of supporting the execution of an HDA as a service chain while meeting desired objectives.

#### 3.2.1.1 Application Execution Environment

An application may run **natively on an Operation System (OS) of a physical machine (PM)**, or on a **hosted Container in an OS of a PM** (*Container virtualization*) or on an OS or a hosted container in an OS of a **virtual machine (VM)** hosted by a hypervisor of a PM (*bare metal virtualization*) or of an OS of a PM (*hosted virtualization*). Application running **in further nested running environments** is also possible, although considered mostly valid only for testing purposes e.g. running in a Container in an OS of a VM hosted by a hypervisor of a VM hosted by a hypervisor of a PM.

We will refer as **running on Native OS** the case of running on an OS of a PM, as **running on a Container** the case of running on a hosted Container in an OS of a PM and as **running on a Virtual Machine** the case of running on a VM hosted by a hypervisor. All other valid cases meaningful (like running on a container in a VM) or less meaningful are considered as *nested cases*.



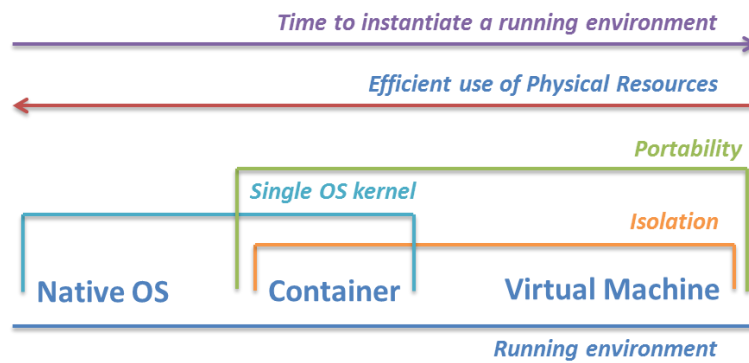


Figure 3-2: Characteristics according the application running environment

Roughly, bare metal or hosted virtualization are based on the hardware level virtualizing hardware resources while container virtualization is done at the operating system level rather at the hardware level. Both hypervisors and containers offer an **isolated environment** for application running in terms of a virtual machine or a container respectively. This is not the case when applications run in the same Native OS (Figure 3-2). *Sandboxing was developed as a technique to isolate for security purposes an application environment in Native OS, but more or less it falls in the category of containerization.* In Container virtualization each container shares the same kernel of the base native OS, thus **hosting a different OS** in a container which will be the running environment of applications is not possible as it is in the hypervisor case. However, container virtualization is considered “light” since unlike in hypervisors, which access physical resources through a virtualization layer, it induces less overhead when resources are accessed. It can be claimed that the most **efficient use of physical resources** is accomplished when an application is running at the Native OS, following when it runs in a Container and lastly when it runs in a Virtual machine. This fact indicates that **the capacity of a physical machine in terms of accommodating resource hungry applications** is higher when these run in native OS than when these run in containers which in turn is higher when these run in virtual machines [1]. Regarding storage, hypervisors consume **storage space** for each virtual machine while containers use a single storage space plus smaller deltas for each layer and thus are much more efficient. Furthermore, considering **instantiating the running environment for a new application**, there is zero latency when native OS is considered, given that it is already up and running, while there is latency for a container or a VM to boot and be application-ready. However, **latency is considerably less in container virtualization** since containers typically can boot in less than 500ms while a VM in a hypervisor boots according to the OS typically in about 20 seconds, depending on storage speed [2] [1]. When **horizontal scaling of an application** is considered it seems that containers offer more opportunities for **rapid scaling**. In the case of Native OS horizontal scaling the application will require booting up a new available physical machine like in Metal as a Service (MaaS) setups is done, thus this will require a noticeable latency typically higher than bringing up a VM. Both containers and virtual machines are **portable**, thus running instances may migrate to a different host; for example when vertical scaling is required that it is not possible on the originating host or consolidation policies impose it in order to operate minimum physical machines in an infrastructure.

The most common required characteristics regarding an application’s running environment which partially led development in the virtualization area are: **isolation of the application environment**, **resource isolation**, **low to zero performance loss** compared to native OS environment, **easy sharing** between virtualized hosts, **easy management** of application running environments, portability.

Furthermore, **resources management** is considered quite crucial in order to efficiently handle capacity and meet applications requirements. Tools to control resources are available in all types of running environments (Native OS, Containers, Virtual machines). In example, Cgroups in Linux Kernel which was originated from Google Engineers who had to limit resource utilization (CPU, memory etc.) for different process groups, serves as a valuable tool in resources management [3] [4]. However, a still not resolved aspect of container resource management is the fact that *processes running inside a container are not aware of their resource limits* [1].

Container virtualization although has not reached yet the maturity of hypervisors most probably will be the future trend regarding application running environments. A late differentiation is between OS containers and **Application containers** [5]. The idea behind application containers is that you create different containers for each of the components in your application which is ideal to deploy a distributed, multi-component system using the microservices architecture, able to scale both horizontally and vertically the different applications.

In the following, representative illustrations of each alternative execution environment are presented and compared.

### 3.2.1.1.1 Hypervisors

Main representatives of the open source hypervisors are KVM [6] and XEN [7] while from the proprietary/commercial ones are VMware Vsphere [8] and Microsoft Hyper-V [9]. The critical mass of cloud solutions for data centers tend to be built around KVM which is a linux hosted virtualization hypervisor and Vsphere which is a bare metal hypervisor. In the following (Table 3-1) a brief side by side comparison on selected characteristics between the aforementioned hypervisors is presented. The open source KVM is compared in its commercial bundling from Redhat and Xen in its commercial bundling from Citrix. For a thorough comparison of these products the reader may refer to [10]. The selected core characteristics presented in this comparison, roughly, refer to the host configuration and capabilities, to the VM configuration, interoperability and to supported functions regarding resources management, security and VM mobility.

**Table 3-1: Source [10] Comparison of popular Hypervisors**

Hypervisor		VMware vSphere 5.5	Redhat RHEV 3.5	Microsoft HyperV 2012R2	Citrix XenServer 6.5
General	<b>Hypervisor Details/Size</b>	Virtual Hardware 'version 10', VMware ESXi 5.5: Build 1331820, vCenter Server 5.5: Build 1312298, vCenter Server Appliance 5.5	<b>KVM</b> with RHEV-H or RHEL	Hyper-V '3'	XenServer 6.5: <b>Xen 4.4 -based</b>
Host Config	<b>Max Consolidation Ratio</b>	512vm, 4096 vCPU , max 32 vCPU / core	No limit stated	1024 vims/host, 2048 vCPUs/host	500 vm (Win) or 650 (Linux) per host
	<b>Max CPU – Host</b>	320 (Logical)	160 (logical)	320 Logical CPUs	160 (logical)
	<b>Max Cores per CPU</b>	unlimited	unlimited	unlimited	unlimited
	<b>Max Memory – Host</b>	6 TB (New with Update 2)	4TB	4TB	1TB
VM Config	<b>Max vCPU per VM</b>	64	160 vCPU per VM	up to 64 vCPU (Win) / 64 vCPU (Linux)	16 (Win) / 32(Linux)



	<b>Max RAM per VM</b>	1TB	4TB	1TB	192GB
	<b>Serial Ports</b>	Yes max 4 (incl. vSPC)	No (serial console via hooks possible)	yes (named pipe)	No
	<b>Hot Add/Plug</b>	Yes (CPU, Mem, Disk, NIC), PCIe SSD	Yes (disk, NIC, CPU)	disks and memory (dynamic) only	Yes (disk, NIC)
Memory	<b>Dynamic / Over-Commit</b>	Yes (Memory Ballooning)	Yes (virtio), Mem Balloon optimization and error messages	Yes - Dynamic Memory (Linux guest support)	Yes (DMC)
	<b>Memory Page Sharing</b>	Yes (Transparent Page Sharing)	Yes (KSM)	No	No
	<b>HW Memory Translation</b>	Yes	Yes	Yes (SLAT)	Yes
Interoperability	<b>OVF Support</b>	Yes	Yes	Yes (OVF Import/Export)	Yes, incl. vApp
	<b>HW Compatibility</b>	Very Comprehensive (see link)	Comprehensive	Strong Windows Ecosystem	Improving
	<b>Guest OS Support</b>	Very Comprehensive (see link)	Limited	Closing the gap	Good
	<b>Scripting / APIs</b>	Web Services API/SDK, CIM, Perl, .NET, Java SDKs, Client Plug-In API, vSphere Clip, vMA	REST API, Python CLI, Hooks, SDK	Yes (WMI API, PowerShell 4 - NEW)	Yes (SDK, API, PowerShell)
	<b>Cloud API</b>	vCloud API	REST API	Service Provider Foundation API, Azure Service Management API	CloudStack APIs, support for AWS API
Other	<b>Resource Pools</b>	Yes	Yes (Quota, Devices SLA, CPU Shares)	Yes (Host Groups)	No
	<b>Security</b>	Free: ESXi Firewall, vShield Endpoint; Advanced (with Vendor Add-On: NSX / vCloud Networking and Security)	SELinux, iptables, VLANs, Port Mirroring	Windows Security, Hyper-V Extensible Switch (DNSSEC, PVLANS, port ACLs, BitLocker etc.)	Basic (NetScaler - Fee-Based Add-On)
VM Mobility	<b>Live Migration of VMs</b>	Yes vMotion, Metro vMotion and 'shared nothing' vMotion (4-8 concurrent)	Yes (Live Migration - unlimited concurrent migrations, 3 by default)	Yes ('Unlimited' Concurrent, 'Shared Nothing'; new compression & SMB3 options)	Yes XenMotion
	<b>Migration Compatibility</b>	Yes (EVC)	Yes (except with CPU pass-through)	Yes (Processor Compatibility)	Yes (Heterogeneous Pools)
	<b>Maintenance Mode</b>	Yes	Yes	Yes	Yes
	<b>Automated Live Migration</b>	Yes (DRS) - CPU, Mem, Storage (new	Yes (LB) - Built-in (CPU) and	Yes - Dynamic Optimization (CPU,	Yes Workload Balancing

		affinity rules, migrate replicated vm)	Scheduler for 'custom'	mem, disk I/O, Net I/O)	
	<b>Power Management</b>	Yes (DPM), Enhanced Host Power Management (C-States)	Yes (Power Saving)	Yes - Power Optimization	Yes Workload Balancing
	<b>Storage Migration</b>	Yes (Live Storage vMotion); including replicated vm	Yes	Yes (Live and 'Shared Nothing')	Yes (Storage XenMotion)

### 3.2.1.1.2 Containers

Lightweight process virtualization is not new; it is met in the past in Solaris Zones, BSD jails, AIX WPARs (Workload Partitions) and Linux-based containers projects [11].

Most focus now is on Linux Containers (LXC) [12] and other solutions which are based or at least initially were based on them, like Docker [13]. The building blocks of LXC are namespaces and cgroups supported at the kernel level [11]. According to the namespaces man page [14]: "A namespace wraps a global system resource in an abstraction that makes it appear to the processes within the namespace that they have their own isolated instance of the global resource" while cgroups (control groups) is a linux kernel feature that limits, accounts for and isolates the resource usage (CPU, memory, disk I/O, network, etc.) of a collection of processes [15].

Containers aim to offer an environment close to the one of a VM but without the overhead that comes from running a separate kernel and simulating all the hardware. LXD [12] is "hypervisor" for containers aiming to combine the speed and density of containers with the security of traditional virtual machines. It is made of three components: A system-wide daemon (lxd), a command line client (lxc) and an OpenStack Nova plugin (nova-compute-lxd) while the daemon exports a REST API.

Docker is in fact an orchestration solution built on top of the linux kernel. Docker originally used LXC as the "engine" but recently developed their solution called "*libcontainer*". Docker containers focus to a single application by design, a container that can only run a single app. Furthermore, layered containers are supported. Thus, it has a different design approach from lxc, it is application oriented while lxc is more like a VM. In Figure 3-3 the key differences between LXC and Docker are shown.

Regarding the Docker's architecture, a set of Docker platform services orchestrate HDAs operating exclusively on Docker containers. These three services, Docker Compose, Docker Swarm, and Docker Machine, cover both deployment and post-deployment functionalities. The deployment of the multi-container distributed applications can be done with Docker Compose and is based on YAML. Failover and resource scaling management is offered through Docker Swarm which creates and manages a cluster of Docker containers. Finally, Docker Machine allows simple, CLI-based provisioning of Docker containers in variety of infrastructures such as laptops, datacenter VMs, and cloud nodes. This can greatly simplify the up-scaling of Docker container clusters that Docker Swarm makes possible.

## Key differences between LXC and Docker

flockport

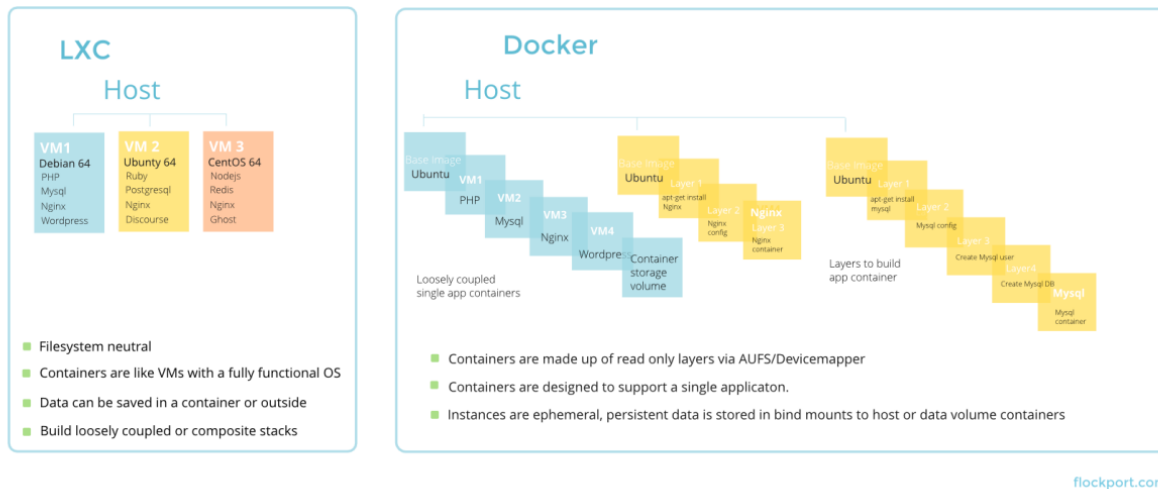


Figure 3-3: Source [16] LXC versus Docker

Microsoft as well is about to present its own solution to support Windows OS containers; Hyper-V Containers, a new container deployment option with enhanced isolation powered by Hyper-V virtualization and Nano Server, a minimal footprint installation of Windows Server that is highly optimized for the cloud, and ideal for containers [17].

### 3.2.1.1.3 Native OS

In a cloud environment, a new type of service attempts to handle a physical machine as easy as a virtual instance. Metal As A Service (MAAS) [18] provides a toolset to power on/off a physical server or more (in case of a cluster) utilizing one out of several remote management technologies (e.g. Wake up on Lan, IPMI, AMT), commission with an OS through PXE boot and assign applications and workloads for execution in Native OS or even containers or virtual machines. Roughly, in a similar philosophy is the Ironi service [19] as part of the Openstack cloud software.

### 3.2.1.1.2 Application Networking Environment

Illustration of a virtual network among executing environments and middleboxes as required by an HDA, is facilitated by **technologies developed in the highly convoluted areas** [20] of **Network Virtualization (NV)**, **Software Defined Networking (SDN)** and **Network Functions Virtualization (NFV)**.

**Network virtualization** is defined as the abstraction of a network that is decoupled from the underlying physical equipment. Any technology that facilitates hosting a virtual network on an underlying physical network infrastructure may be considered to fall within the scope of network virtualization [20]. Multiple *isolated* virtual networks are allowed to run over a shared infrastructure.

Historically, network equipment has supported for many years the creation of virtual networks, in the form of Virtual Local Area Networks (VLANs) and Virtual Private Networks (VPNs). A **Virtual Local Area Network (VLAN)** provides the illusion of a single LAN although it may span over multiple

physical subnets while multiple VLANs may be illustrated over the same collection of switches and routers. A **virtual private network (VPN)** extends a private network across a public network, enabling a network device to send and receive data across shared or public networks as if it were directly connected to the private network by establishing a virtual point-to-point connection through the use of dedicated connections, virtual tunneling protocols, or traffic encryption. The idea of an **overlay network** is met in its expanded version in Peer to Peer (P2P) networking where end-hosts who run a special P2P application form the overlay network by being its nodes, without any kind of demand from the networking equipment.

**Networking equipment virtualization** support; routers and switches able to provide isolated virtual instances of their resources (*partition one physical device and make it appear as multiple or make multiple physical devices appear as one*), further provided the ability of building virtual infrastructures over physical ones and segmenting the physical network into many logical ones (slices) [21] [22]. *Providing several virtual instances of a networking device requires the separation of the control plane and the forwarding plane (management and packet transmission) within the device.* Towards this direction **Software Defined Networking (SDN)** became an **enabling technology** for network virtualization.

**Software Defined Networking (SDN)** provides a different perspective in designing and managing networks. SDN has two defining characteristics. First, an SDN **decouples/separates the system that decides how to handle the traffic (Control Plane) from the underlying systems that forward the traffic to destinations (Data Plane)** according to decisions that the control plane makes. Second, an SDN **consolidates the control plane**, so that a single software control program controls multiple data-plane elements (Figure 3-4). The SDN **control plane directly controls the state in the network's data-plane elements (i.e., routers, switches, middleboxes) via an Application Programming Interface (API).** OpenFlow is an example of such an API.

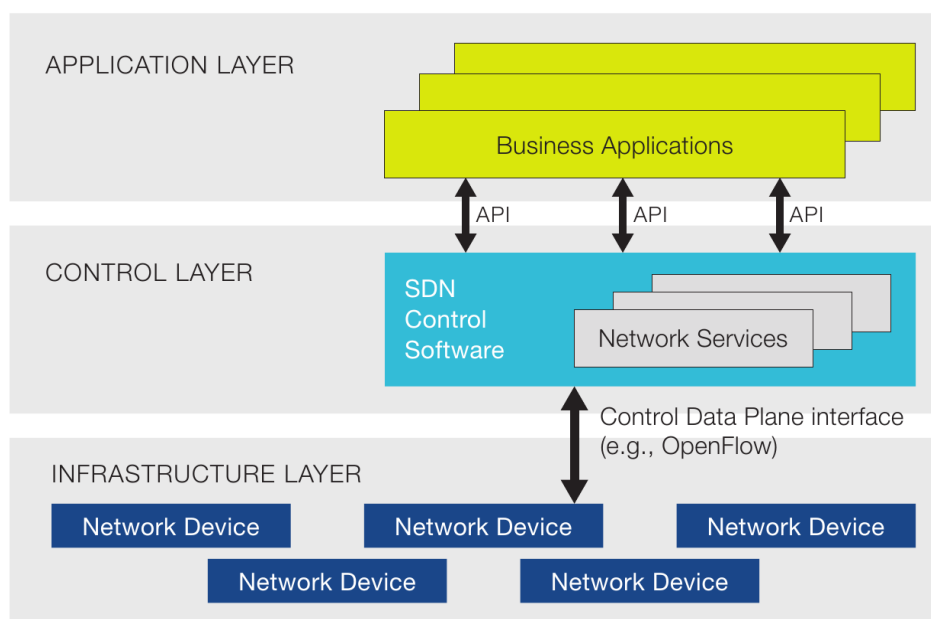


Figure 3-4: SDN architecture

*Network virtualization and SDN does not require or imply one another, however there is a twofold enabling synergy.*

The OpenFlow protocol [23] standardized a data-plane model and a control-plane API by building on technology that switches already supported. OpenFlow enabled more functions than earlier route controllers while building on existing switch hardware. Vendors did not have to upgrade the hardware in order to make their switches OpenFlow-capable but only upgrade their firmware. Relying on existing switch hardware may be considered that limits flexibility, however facilitated OpenFlow to be almost immediately deployable.

An SDN Controller in SDN is the core of an SDN network, relaying information to switches/routers via southbound APIs and the applications via northbound APIs. In an OpenFlow environment, any device that communicates to an SDN/OpenFlow Controller supports the standard OpenFlow protocol. The SDN Controller pushes down changes to the switch/router flow-table allowing network administrators to partition traffic, control flows for optimal performance, and perform testing of new configurations and applications.

Furthermore, SDN's separation between the controller and the data-plane state has facilitated the **live migration of a complete network as an ensemble**—the VMs, the network, and the management system—to a different set of physical resources [24, 25]. In [24, 25] the authors introduce a method for migrating a network, that transparently to the application running on the controller clones the data-plane state to a new set of switches, and then incrementally migrates the traffic sources (e.g., the VMs). During this transition, both networks deliver traffic and a synchronized state is maintained.

**Network Functions Virtualization (NFV)** [26] suggests that any service to be delivered on proprietary, application specific hardware should be able to be delivered on virtual machines. Thus, routers, firewalls, load balancers and other network devices should be able to be illustrated on virtual machines hosted on commodity hardware [27]. Such an approach has been **facilitated by network focused advancements in PC hardware**. The fact that required packet processing at a network device requires proprietary hardware when other than small scale deployments are considered, tends to not hold any more due to moved focus and several advancements such as in packet handling within Intel's processors [28] [29], allowing processor cores to be re-programmed into network processors and PC-based network devices to be able to push 10's or even 100's of Gbp/s.

The NFV framework (Figure 3-5, Figure 3-6) consists of three main components:

1. Virtualized network functions (VNF) are software implementations of network functions that can be deployed on a Network Function Virtualization Infrastructure (NFVI).
2. Network function virtualization infrastructure (NFVI) is the totality of all hardware and software components which build up the environment in which VNFs are deployed. The NFV-Infrastructure can span across several locations. The network providing connectivity between these locations is regarded to be part of the NFV-Infrastructure.
3. Network functions virtualization management and orchestration architectural framework (NFV-MANO Architectural Framework) is the collection of all functional blocks, data repositories used by these functional blocks, and reference points and interfaces through which these functional blocks exchange information for the purpose of managing and orchestrating NFVI and VNFs.

The building block for both the NFVI and the NFV-MANO is the NFV platform. In the NFVI role, it consists of both virtual and physical processing and storage resources, and virtualization software. In its NFV-MANO role it consists of VNF and NFVI managers and virtualization software operating on a hardware controller. The NFV platform implements carrier-grade features used to manage and monitor the platform components, recover from failures and provide effective security - all required for the public carrier network.

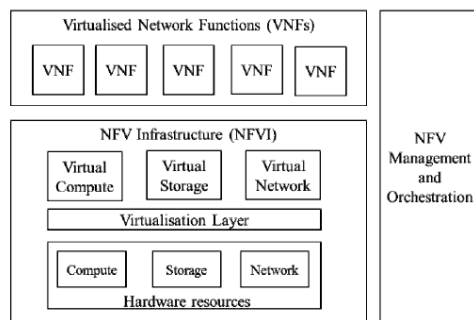


Figure 3-5: NFF framework architecture

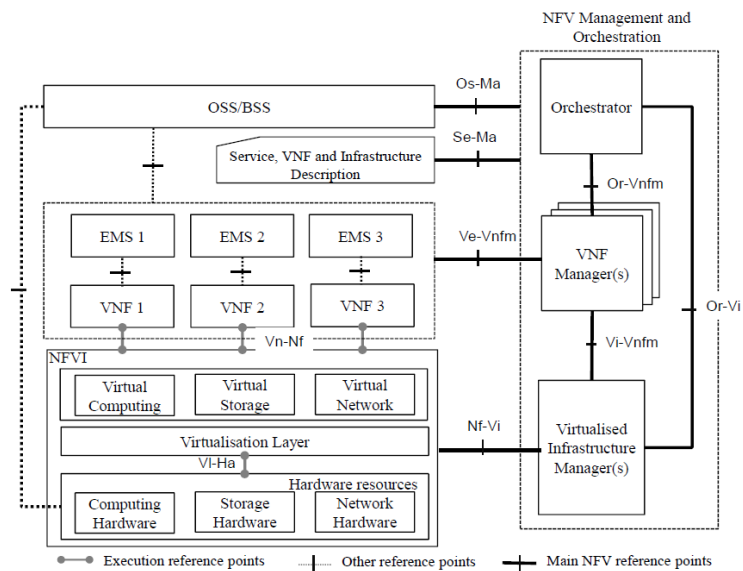


Figure 3-6: NFV framework break down

The benefits of the NFV approach follow the benefits of platform virtualization such as reduced CAPEX and OPEX, reduces complexity, flexibility, easier management, remote instantiation, resources' use efficiency.

*Network Functions Virtualization can be implemented without requiring SDN, however both concepts and solutions can be complementary producing higher value.*

Although the idea of moving every network function as a virtual machine to a central hypervisor is very attractive, however we have to consider that some network functions are tied to a physical location e.g. a firewall is required at the connection point of an internal network. SDN and NFV along may provide effective solutions in such cases because a virtualized network is far less restricted by location e.g. in the firewall example once the firewall function has been allocated to a specific virtual machine, then a software defined network could place it at the network edge regardless of its actual physical location by providing a direct, quarantined link from the Internet to the virtual firewall before traffic entered the internal network. [29]. Admittedly, such a routing to the Virtual Network Function (VNF) the firewall in this example could be manually configured without the need of SDN, however SDN offers a great potential of reforming the network e.g. in the case of network problems reconfiguring immediately the network and redirecting traffic to a different VNF.

A combination of SDN and NFV seems ideal as it ensures not only initial deployment but also required flexibility in terms of reconfigurations during its run time, triggered by scaling requirements,

requirements related to achieving optimization objectives and sustaining continuity of operation under system's dynamics.

In the following, various illustrations of SDN components are presented. Since the focus is on software components illustrating network functionality which may execute in a virtualized execution environment, such components maybe considered as NFV components as well.

### 3.2.1.2.1 SDN/NFV components

A popular open source implementation of a multilayer virtual switch supporting Openflow is Open vSwitch [31]. Open vSwitch is able to operate either as a soft switch running within the hypervisor or as the control stack for switching silicon. It has been ported to multiple virtualization platforms and switching chipsets. It is the default switch in XenServer 6.0, the Xen Cloud Platform and also supports Xen, KVM, Proxmox VE and VirtualBox. It has also been integrated into many virtual management systems including OpenStack, openQRM, OpenNebula and oVirt. Open vSwitch project also includes a trivial Openflow reference controller, OVS.

There are several opensource Openflow controllers like POX [32]; which has a high-level SDN API including a querable topology graph and support for virtualization, IRIS [33]; which is a Resursive SDN Openflow Controller with the following features : (a) Horizontal Scalability for carrier-grade network (b) High Availability with transparent failover from failure (c) Multi-domain support with recursive network abstraction based on Openflow, and Floodlight [34]; which is a java-based Openflow controller designed to work with the growing number of switches, routers, virtual switches, and access points that support the OpenFlow standard (Figure 3-7).

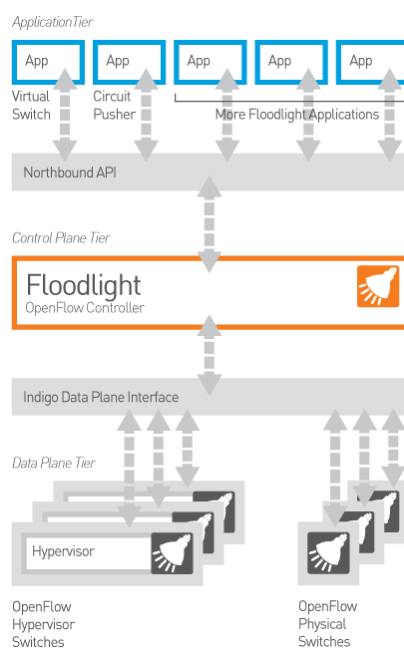


Figure 3-7: Source [34] Floodlight Openflow controller

In the following, Table 3-2 presents a list of current software switch implementations with a brief description including implementation language and the OpenFlow standard version that the current implementation supports.



Table 3-2: List of available SDN software switches.

Name	Implementation	Overview	Version
Open vSwitch [31]	C/Python	Open source software switch that aims to implement a switch platform in virtualized server environments. Supports standard management interfaces and enables programmatic extension and control of the forwarding functions. Can be ported into ASIC switches.	v1.0
Pantou/ OpenWRT [44]	C	Turns a commercial wireless router or Access Point into an OpenFlow-enabled switch.	v1.0
ofsoftswitch13 [45]	C/C++	OpenFlow 1.3 compatible user-space software switch implementation.	v1.3
Indigo [46]	C	Open source OpenFlow implementation that runs on physical switches and uses the hardware features of Ethernet switch ASICs to run OpenFlow.	v1.0
OpenFaucet [47]	Python	As a pure Python implementation of OpenFlow protocol, OpenFaucet can implement both switches and controllers.	v1.0

Table 3-3 provides a list of native SDN switches currently available in the market, with indication of the OpenFlow version they implement.

Table 3-3: List of commercial switches compliant with the OpenFlow protocol

Vendor	Model	Version
Arista	7050 series	v1.0
Hewlett-Packard	8200zl, 6600, 6200zl, 5400zl, 3500/3500yl, 3800	v1.0
Brocade	MLX Series, NetIron CES 2000 Series, NetIron XMR series, ICX 7750 switch	v1.0
Dell Force10	Z9000, S-Series S4810	v1.0
Extreme networks	Summit X440, X460, X480, and X670	v1.0
IBM	RackSwitch G8264	v1.0
Larch networks	Linux-based OpenFlow switch	v1.0
NEC	PF5240, PF5248, PF5820, and PF1000 virtual switch	v1.0
NoviFlow	NoviSwitch 1248 and NoviSwitch 1132	v1.3
Pronto	3290 and 3780	v1.0
Juniper	Junos MX-Series	v1.0
Pica8	P-3290, P-3295, P-3780 and P-3920	v1.3

Table 3-4 shows a snapshot of current controller implementations; all the controllers support the OpenFlow protocol version 1.0, unless stated otherwise.

Table 3-4: List of controllers compliant with the OpenFlow standard.

Controller	Implementation	Open Source	Developer	Overview
POX [32]	Python	Yes	Nicira	General, open-source SDN controller written in Python
NOX [48]	Python/C++	Yes	Nicira	The first OpenFlow controller written in Python and C++.
MUL [49]	C	Yes	Kulcloud	OpenFlow controller that has a C-based multi-threaded infrastructure at its core. It supports a multi-level north-bound interface for application development.
Maestro [50]	Java	Yes	Rice University	A network operating system based on Java; it provides interfaces for implementing modular network control applications and for them to access and modify network state
Trema [51]	Ruby/C	Yes	NEC	A framework for developing OpenFlow controllers written in Ruby and C.
Beacon [52]	Java	Yes	Stanford	A cross-platform, modular, Java-based OpenFlow controller that supports event-based and threaded



				operations.
Helios [54]	C	No	NEC	An extensible C-based OpenFlow controller that provides a programmatic shell for performing integrated experiments.
Floodlight [34]	Java	Yes	BigSwitch	A Java-based OpenFlow controller (supports v1.3), based on the Beacon implementation, that works with physical- and virtual- OpenFlow switches.
SNAC [55]	C++	No	Nicira	An OpenFlow controller based on NOX-0.4, which uses a web-based, user-friendly policy manager to manage the network, configure devices, and monitor events.
Ryu [56]	Python	Yes	NTT, OSRG group	An SDN operating system that aims to provide logically centralized control and APIs to create new network management and control applications. Ryu fully supports OpenFlow v1.0, v1.2, v1.3, and the Nicira Extensions.
IRIS [33]	Java	Yes	ETRI	IRIS is a recursive OpenFlow controller that aims to support scalability, high availability, and multi-domain support.
OESS [57]	Perl	Yes	NDDI	OESS is a set of softwares to configure and control dynamic VLAN networks using OpenFlow-enabled switches.
Jaxon [53]	Java	Yes	Independent Developer	Jaxon is a NOX-dependent OpenFlow controller; it's primarily intended to be used as part of a bigger project, which is working towards using Scala to manage datacenters by representing datacenter entities (like virtual machines) as normal Scala objects.
NodeFlow [58]	JavaScript	Yes	Independent Developers	An OpenFlow controller written in JavaScript for NodeJS.
ovs-controller [31]	C	Yes	Independent Developers	A simple OpenFlow controller reference implementation with Open vSwitch for managing any number of remote switches through the OpenFlow protocol; as a result the switches function as L2 MAC-learning switches or hubs.
OpenDayLight [38]	Java	Yes	OpenDayLight community	OpenDaylight is an open platform for network programmability to enable SDN and NFV for networks at any size and scale. It also features a deeper integration with OpenStack,
Flowvisor [59]	C	Yes	Stanford/Nicira	Special purpose controller implementation. Flowvisor acts as a transparent proxy between OpenFlow switches and multiple OpenFlow controllers, to create network slices and to delegate control of each slice to a different controller.
RouteFlow [60]	C++	Yes	CPqD	Special purpose controller implementation. RouteFlow provides virtualized IP routing over OpenFlow capable hardware.

### 3.2.2 Cloud Infrastructure Platforms and Orchestration Frameworks

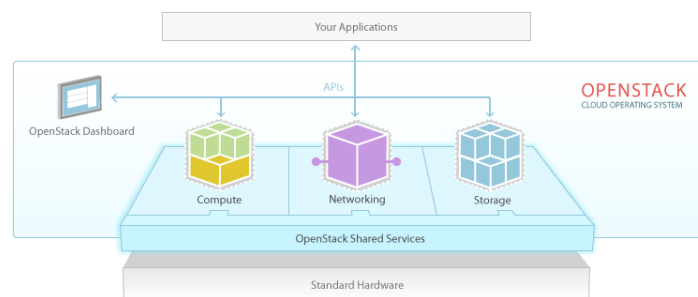
In the following Cloud Infrastructure Platforms and Orchestration Frameworks are presented which bring together several of the previously presented building blocks to provide a complete solution of a programmable infrastructure. Although highly interrelated and synergetic, platforms are presented within a rough categorization in Cloud Computing Platforms, SDN/NFV Platforms and Other Orchestration Frameworks. The latter are mostly focused to provide application oriented solution by providing the orchestration means.

### 3.2.2.1 Cloud Computing Platforms

Openstack, Cloudstack and OpenNebula are presented as the most representative cloud platforms while then compared with each other and other private and public cloud computing infrastructure platforms.

#### 3.2.2.1.1 Openstack

OpenStack software [35] controls large pools of compute, storage, and networking resources throughout a datacenter, managed through a dashboard or via the OpenStack API (Figure 3-8). Users primarily deploy it as an infrastructure as a service (IaaS) solution.



**Figure 3-8: Source [5] Openstack**

OpenStack's architecture is modular. The components of the latest Openstack Kilo version are [36]:

- (1) Compute (Nova); a cloud computing fabric controller designed to manage and automate pools of computer resources which can work with widely available virtualization technologies (such as KVM, XEN, VMWARE, Hyper-V, Linux Containers), as well as bare metal and high-performance computing (HPC) configurations. Compute's architecture is designed to scale horizontally on standard hardware.
- (2) Image Service (Glance); provides discovery, registration, and delivery services for disk and server images.
- (3) Object Storage (Swift); it is a scalable redundant storage system.
- (4) Dashboard (Horizon); provides administrators and users a graphical interface to access, provision, and automate cloud-based resources.
- (5) Identity Service (Keystone); provides a central directory of users mapped to the OpenStack services they can access. It acts as a common authentication system across the cloud operating system and can integrate with existing backend directory services like LDAP.
- (6) Networking (Neutron); it is a system for managing networks and IP addresses. OpenStack Networking provides networking models for different applications or user groups.
- (7) Block Storage (Cinder); provides persistent block-level storage devices for use with OpenStack compute instances.
- (8) Orchestration (Heat); it is a service to orchestrate multiple composite cloud applications using templates, through both an OpenStack-native REST API and a CloudFormation-compatible Query API.
- (9) Telemetry (Ceilometer); provides a Single Point Of Contact for billing systems, providing all the counters they need to establish customer billing, across all current and future OpenStack components.
- (10) Database (Trove); it is a database-as-a-service provisioning relational and non-relational database engines.

(11) Elastic Map Reduce (Sahara); provides users with simple means to provision Hadoop clusters by specifying several parameters like Hadoop version, cluster topology, nodes hardware details and a few more.

(12) Bare Metal Provisioning (Ironi); it is an incubated OpenStack project that aims to provision bare metal machines instead of virtual machines.

(13) Multiple Tenant Cloud Messaging (Zaqar); it is a multi-tenant cloud messaging service for Web developers

(14) Shared File System Service (Manila); provides an open API to manage shares in a vendor agnostic framework.

(15) DNSaaS (Designate); DNS as a Service

(16) Security API (Barbican); it is a REST API designed for the secure storage, provisioning and management of secrets.

#### 3.2.2.1.2 Cloudstack

Apache CloudStack provides an open and flexible cloud orchestration platform to deliver reliable and scalable private and public clouds [37]. It provides a management server and agents for hypervisor hosts so that illustrating an IaaS cloud is possible. Indicatively, it works with hosts running XenServer/XCP, KVM, Hyper-V, and/or VMware ESXi with vSphere, manages storage for instances and orchestrates network services from the data link layer (L2) to some application layer (L7) services, such as DHCP, NAT, firewall, VPN, and so on.

#### 3.2.2.1.3 OpenNebula

OpenNebula [61] is an open-source cloud computing platform for managing heterogeneous distributed data center infrastructures. It manages a data center's virtual infrastructure to build private, public and hybrid implementations of IaaS. OpenNebula orchestrates storage, network, virtualization, monitoring, and security technologies to deploy multi-tier services as virtual machines on distributed infrastructures, combining both data center resources and remote cloud resources, according to allocation policies. It provides support for several cloud interfaces such as Amazon EC2 Query, OGF Open Cloud Computing Interface and vCloud. It works with several hypervisors such as Xen, KVM and VMware while it can accommodate multiple hardware and software combinations in a data center.

#### 3.2.2.1.4 Comparison

There are several software solutions available for cloud computing, both commercial and open-source. We provide a very quick and concise review of the most popular ones, some of them presented before, by comparing the following elements:

- *Resources*: the main virtual resources exposed to users (e.g., computing, networking, storage);
- *Monitoring*: what kinds of statistics can be collected by users;
- *User interfaces (UI)*: what kinds of interfaces are available to users (Web GUI, Command Line Interface, Application Programming Interface);
- *Management interfaces (MI)*: what kind of management interfaces are available for management purposes;
- *Relevant features*: some features that are relevant for ARCADIA, like live migration, scheduling, power management, horizontal and vertical scalability.

Our analysis does not include all software solutions available to build clouds; we only take into account most used commercial services for public clouds and software for building private clouds (Table 3-5).

**Table 3-5: Comparison of Cloud Platforms**

	<i>Resources</i>	<i>Monitoring</i>	<i>UI</i>	<i>MI</i>	<i>Relevant features</i>
<i>Amazon E2C</i>	Computing, Networking, Storage, Network functions (Firewall), Dockers, Interfaces to AWS cloud, Relational Databases	CPU, Memory, Network, Disks, Swap, Page file	GUI (Web), CLI	-	Horizontal and vertical scalability.
<i>Microsoft Azure</i>	Computing, Networking, Storage, Databases	CPU, Disks, Memory, Network, Packets, Pages, Swap	GUI (Web), CLI	-	Horizontal and vertical scalability.
<i>Google Cloud Engine</i>	Computing, Networking, Storage, Network functions (Firewall, VPN), Data locality (regions & zones)	Requests, Latency, Loading Latency, Error Details, Traffic, Utilization, Instances, Memory Usage, Memcache	GUI (Web), CLI	-	Live migration; horizontal and vertical scalability.
<i>VMware vCloud Air</i>	Computing, Networking, Storage	Time, Customer, Host DNS Name, Host RAM, Capacity Remaining, Alerts, Network Statistics, Memory, CPU	GUI (Web), CLI	GUI (Web), CLI	Live migration and storage migration; horizontal and vertical scalability; bandwidth reservation for traffic pools (management, migration, virtual machine traffic).
<i>Open-Stack</i>	Computing, Networking, Storage, Network functions (Firewall, Routers, Load Balancer, VPN)	Sum of Virtual CPUs; Memory allocated; Disk sizes; CPUs, memory and disk sizes of the host machine; CPU, memory, I/O and network statistics for an instance; Diagnostic statistics (r/w, packets, errors), Tenant statistics	GUI	CLI, AWS, API	Live migration; horizontal and vertical scalability; three scheduling disciplines ( <i>simple scheduler</i> , <i>chance scheduler</i> and <i>availability zone scheduler</i> ); traffic shaping and rate limiting.
<i>Eucalyptus (Used by HP Enterprise)</i>	Computing, Networking, Storage	Performance and alarms on CPU, Memory, Storage, I/O	GUI, REST, SOAP (XML)	GUI (Web)	Horizontal and vertical scalability; several scheduler disciplines (same as OpenStack).
<i>CloudStack</i>	Computing, Networking, Storage, Network functions (Firewall, VPN)	Virtual Machine Usage; Network statistics; Disk Volume Usage; Template, ISO, Snapshot Usage; Load Balancer Usage; Network Offering Usage; VPN User Usage	GUI (Web)	GUI (Web)	Live migration and storage migration; horizontal and vertical scalability; <i>FirstFit</i> scheduler discipline; network throttling; static assignment of VM and storage to physical resources.
<i>OpenNebula</i>	Computing, Networking, Storage, Images	VM Start Time, End Time; Assigned Memory; Number of CPUs; Data received from the network; Data sent to the network	GUI (Web)	GUI (Web)	Live and cold migration; horizontal and vertical scalability; several scheduler policies ( <i>Packing</i> , <i>Striping</i> , <i>Load-aware</i> , <i>Fixed</i> ); traffic control (bandwidth).

Regarding migration, it is roughly distinguished among:

- live migration of VM instances, i.e., only computing is moved, while the storage is shared;
- storage migration, when only the storage is moved to a different location;
- cold migration, that means the VM is saved and VM files are transferred to the new resource.

Regarding scheduling policies we will refer later on when deployment related issues are addressed.

### 3.2.2.2 SDN/NFV Platforms

In the following, representative SDN and NFV platforms are presented. Although some of them just released or about to be released, they are presented due to their innovative and promising characteristics.

#### 3.2.2.2.1 OpenDaylight

OpenDaylight [38] is an open platform for network programmability to enable SDN and NFV for networks at any size and scale. OpenDaylight software is a combination of components including a fully pluggable controller, interfaces, protocol plug-ins and applications. There is integration with OpenStack cloud software, including improvements in the Open vSwitch Database Integration project, and new OpenStack features such as Security Groups, Distributed Virtual Router and Load Balancing-as-a-Service.

OpenDaylight layered structure is shown in Figure 3-9. The controller exposes open northbound APIs which are used by applications. OpenDaylight supports the OSGi framework and bidirectional REST for the northbound API. The business logic and algorithms reside in the applications. These applications use the controller to gather network intelligence, run algorithms to perform analytics, and then use the controller to orchestrate the new rules, if any, throughout the network. The controller platform itself contains a collection of dynamically pluggable modules to perform needed network tasks. Platform oriented services and other extensions can also be inserted into the controller platform for enhanced SDN functionality. The southbound interface is capable of supporting multiple protocols (as separate plugins), e.g. OpenFlow 1.0, OpenFlow 1.3, BGP-LS, etc. These modules are dynamically linked into a Service Abstraction Layer (SAL).

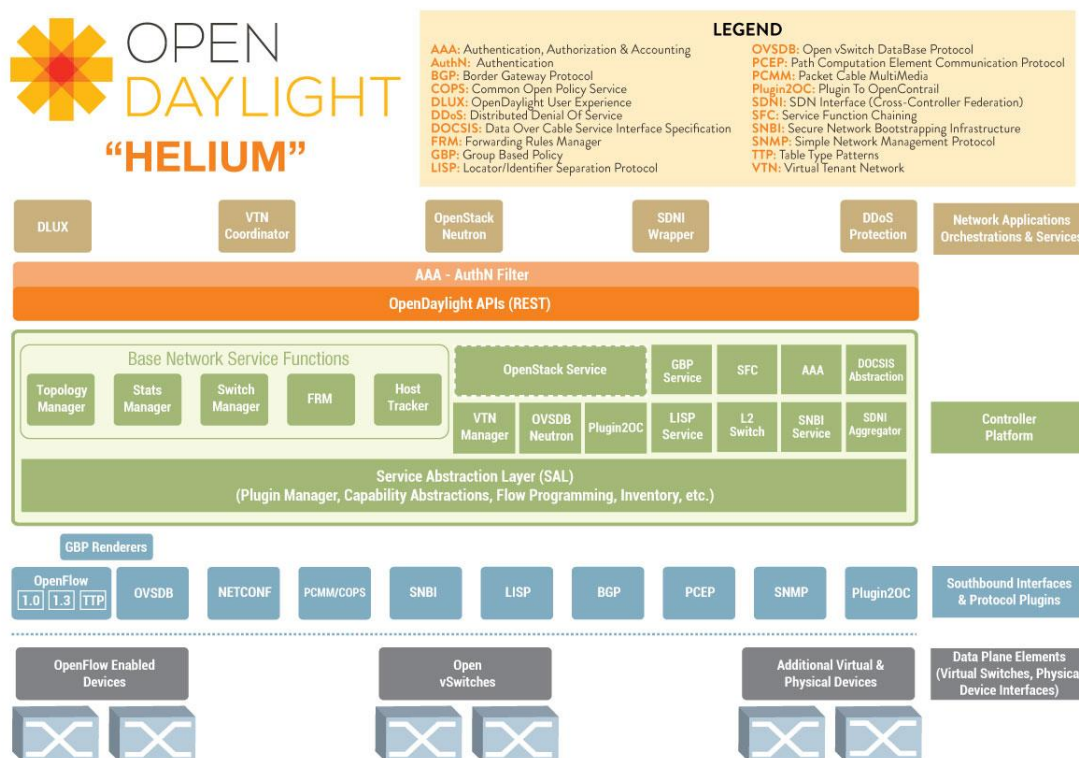


Figure 3-9: Source [38] OpenDaylight architecture



### 3.2.2.2.2 OpenContrail

OpenContrail [39] is an open source platform that provides all the necessary components for network virtualization—SDN controller, virtual router, analytics engine, and published northbound APIs. It has an extensive REST API to configure and gather operational and analytics data from the system. OpenContrail can act as a fundamental network platform for cloud infrastructure. The key aspects of the system are: (1) Network Virtualization; Virtual networks are the basic building block of the OpenContrail approach, (2) Network Programmability and Automation; OpenContrail uses a well-defined data model to describe the desired state of the network. It then translates that information into configuration needed by each control node and virtual router, (3) Big Data for Infrastructure: The analytics engine is designed for very large scale ingestion and querying of structured and unstructured data. OpenContrail interoperates directly with any network platform that supports the existing BGP/MPLS L3VPN standard for network virtualization. Furthermore, it is modular and integrates into open cloud orchestration platforms such as OpenStack, Cloudstack, and is currently supported across multiple Linux distributions and hypervisors.

### 3.2.2.2.3 Atrium

Open Networking Foundation [40] is about to release Atrium 2015/A, an open source SDN distribution that integrates previously standalone open source components. Atrium 2015/A, incorporates the Border Gateway Protocol (BGP) (including Quagga BGP stack), the Open Network Operating System (ONOS) and Open Compute Project (OCP) components. The software elements run in either controllers or switches, communicating using OpenFlow protocol.

### 3.2.2.2.4 OPNFV

The OPNFV community [41] is collaborating on a carrier-grade, integrated, open source platform to accelerate the introduction of new NFV products and services. The scope of OPNFV's initial release (OPNFV Arno) is focused on building NFV Infrastructure (NFVI) and Virtualized Infrastructure Management (VIM) by integrating components from upstream projects such as OpenDaylight, OpenStack, Ceph Storage, KVM, Open vSwitch, and Linux (Figure 3-10). These components, along with application programmable interfaces (APIs) to other NFV elements form the basic infrastructure required for Virtualized Network Functions (VNF) and Management and Network Orchestration (MANO) components. OPNFV's goal is to increase performance and power efficiency; improve reliability, availability, and serviceability; and deliver comprehensive platform instrumentation.

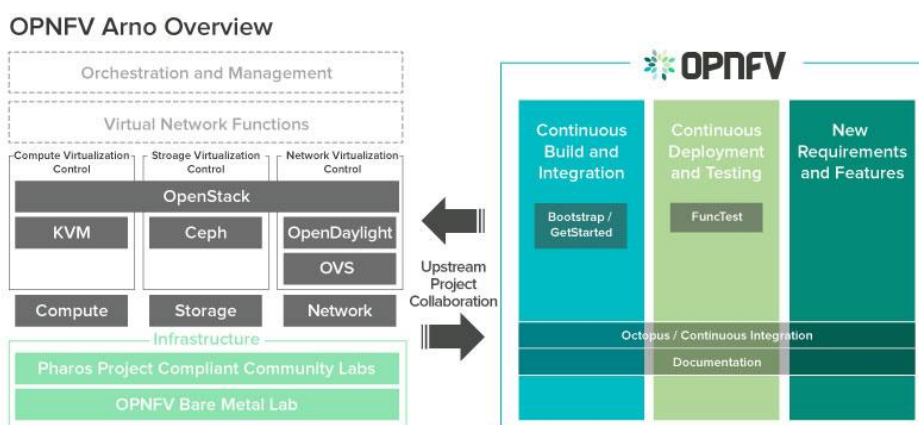


Figure 3-10: Source [41] OPNFV Arno Overview Diagram

### 3.2.2.3 Other Orchestration Frameworks

In the following, representative orchestration frameworks are presented which provide a higher level application specific management, orchestration and automation.

#### 3.2.2.3.1 Cloudify

Cloudify [42] is an open-source framework which allows automation of operational flows from both orchestration and maintenance perspectives. An application in its entirety (Infrastructure, Middleware, Application Code, Scripts, Tool Configuration, Metrics and Logs) may be described in what is called a blueprint. Written in YAML format, a blueprint defines the complete lifecycle of each part of an application. Cloudify will launch the compute instances, and configure network and security while it will execute scripts or configuration management tools to configure the servers and deploy middleware and code.

The suite has a strong focus on deployment modeling and management, which is based on the TOSCA standard. Starting with a concrete topology, workflows can be defined for deployment, and undeployment of the topology components. It is also possible to update a deployed topology without the need for a full uninstall-install cycle.

Orchestration is based on logging and monitoring of the deployment, and deployment environment state. Failover and Resource scaling is possible through custom workflow scripts that are triggered by particular monitoring events.

Configuration and provisioning of cloud resources is performed through Python-based custom workflow scripts or by targeting specialized tools such Chef, Puppet, and SaltStack, which can be integrated to the suite through plugins.

Cloudify interoperates with Openstack, Cloudstack, Docker and other cloud technologies. Furthermore, provides the full end-to-end lifecycle of NFV orchestration through TOSCA-based [43] YAML blueprint.

#### 3.2.2.3.2 ConductR

ConductR [62] is an application management solution for microservice-based applications running on an Akka cluster. An Akka cluster consists of nodes hosted in multiple and varying system configurations, with the single requirement that these configurations are equipped with a JVM. Cluster nodes can be seamlessly added and removed, through a cluster membership system based on Amazon's Dynamo.

ConductR provides resource provisioning for the cluster nodes by targeting third party tools such as Amazon EC2, Docker, Puppet, Chef, and Ansible through a RESTful API.

Application deployment is done with application/service bundles. ConductR optimally distributes the application components to the cluster, in nodes that have the free resources that the components require. ConductR supports deployment of loosely coupled components providing automatic service discovery.

Failover management includes automatic replication of unavailable microservices to healthy nodes and support for redundant application instances.

### 3.3 Applications Profiling, Deployment and Orchestration

**Application deployment** is the last set of actions before having the software up and running. In the case of Highly Distributed Applications, this can be a complex and hard task, which becomes even more complicated due to the need for coordination and management of the various service components of such an application.

An initial deployment of an application requires an initial estimation of the required resources to be acquired from the infrastructure along with several constraints to be met when the application is deployed. Embedding an HDA to a programmable infrastructure while satisfying constraints either coming from the application side or the provider's side is not a trivial task; efficient embedding algorithms have to be devised. Furthermore, initially acquired resources may not be enough to serve workload during runtime and the need of more resources to be allocated to drive a reconfiguration process vertically up or horizontally out scaling the application in order to sustain application's uninterrupted execution and performance may arise. On the other hand, underutilized resources already allocated for an application may also drive a reconfiguration process to release them in favor of efficient usage of available resources; scaling the application vertically down or horizontally in. Reconfigurations may be several as well during the runtime of an application in order to meet either application's or provider's objectives. **Application profiling** as a process to discover in detail an application's resource needs at different types and volumes of workload as well as the process of predicting workloads during an application's runtime, are considered to provide necessary knowledge for driving initial application deployment and several required reconfigurations towards meeting objectives. In such an environment all decisions and directed actions towards meeting goals and satisfying objectives while having a thorough view of each running application, applications about to be deployed and the infrastructure resources and state fall into the duties of the **Orchestration** process, a responsibility of the ARCADIA smart controller.

In the following, approaches in application profiling as well as related matters are presented while next application deployment and orchestration is analyzed in respect to already presented theoretical and illustrated approaches.

#### 3.3.1 Application Profiling

During runtime an application *utilizes compute, storage and network resources*. Most commonly in the majority of cases, **(1) the mapping of the workload mix and volume to the required resources is not known** and **(2) workload to be served and resources needed during application's lifetime are not known** as well.

An application is required to be able, during runtime to **cope with the demand while satisfying QoS constraints and not unnecessarily reserving valuable resources**. Scaling an application either vertically or horizontally to utilize more resources when needed, not only it is not possible at all times but also even if there is adequate resources' availability it is not possible to happen on the fly and **a non-negligible time interval is required until satisfied**. This fact may lead to a serious compromise of the offered QoS or even lead to a service disruption **unless predicted**.

An **initial estimation of the required resources** is necessary when an application is about to be deployed to the cloud and run for the first time. **During runtime and according to the demand the application scales accordingly**; new resources may have to be assigned to the application or resources to be released. The whole process should be automated; often referred as **auto-scaling** [66].

*ARCADIA framework considers the lifetime of an application, starting from its development to cloud deployment, running and termination. Considering and leading the development of an application for sure it is an advantage to be exploited in the direction to meet the objectives aforementioned.*



### 3.3.1.1 Application Profiling, Load testing, Stress testing

Traditionally **software profiling** methods aided developers towards program optimization by revealing, through dynamic program analysis, information about memory and time complexity of programs, frequency and duration of function calls, use of instruction sets etc. Information gathered through program profiling could help developers optimize code towards debugging and better performance as well as towards testing and adapting accordingly program execution for different hardware architectures. Profiling is achieved by instrumenting either the program source code or the program binary executable by the profiler tool which records/measures several information/metrics during program's runtime.

Profiling becomes more complex when it comes to cloud applications since most commonly several software entities form a structured system to deliver a service over variable virtualized architectures. Roughly, it may be distinguished to **application-level profiling** and **function-level profiling** [69]. *Profiling information may prove quite handy when deploying an application for execution over cloud architectures* [70]. For example, deploying an application component with frequent storage I/O (such as a database) into a system where storage I/O is the bottleneck is something that could be avoided when profiling information is available.

**Load testing** is the procedure of analyzing software applications made for multi-user access by subjecting the application to different number of virtual and live users generating different types of loads while measuring performance metrics. Such a procedure **reveals performance and resources utilization related information versus different workload mix and volume produced by different behavioral patterns**.

**Stress testing** is a procedure related to load testing but with the scope of testing an application beyond what is considered "normal operation". It considers heavy load while **focuses on user access behavioral patterns that don't fall into the area of "correct behavior" under "normal circumstances"**.

Load testing and stress testing are procedures usually performed before a service goes live with considerations regarding user access and the system where the service is deployed that attempt to be quite close to the real ones. Although such information is rarely available when a cloud application is deployed, *it is obvious that it could greatly provide a good insight, yet not thorough, on how much of resource is required for an application to cope with a specific workload while maintaining required QoS characteristics* [71].

### 3.3.1.2 Virtualization overhead

Resource requirements of applications differ whether they are running on physical machines or to a virtualized environment. This is due to **overheads caused by the virtualization layer**. Furthermore these overheads differ and they are specific to the type and implementation of the virtualization solution [63] [64]. For example, often, the "amount" of CPU overhead is directly proportional to the "amount" of performed I/O processing. When an application is transferred to a virtualized environment it would be useful to have an estimate of the mapping between the native (when running to a physical machine) resources usage profile and the virtualized one. However in most cases not only the mapping is not available but as well the native resources usage profile.

In [63] the authors develop a set of micro-benchmarks to profile the different types of virtualization overhead on a given platform, and a regression-based model that maps the native system usage profile into a virtualized one. Their approach focuses on estimating the CPU requirements of applications when they are transferred to a virtual environment.

### 3.3.1.3 Auto-scaling techniques

It is desirable the application to scale accordingly to the workload it has to serve, maintaining required QoS guarantees at all times. In other words, at all times the application should ideally have available for use the resources it needs to serve the demand, in a way that does not interrupts its operation while at the same time it does not waste/reserves unneeded valuable resources. **The scaling system should be automatic** (auto-scaling) regarding the application part as well as the provided resources part [66].

*Horizontal scaling is the scaling method of choice for many cloud systems since it provides a way of scaling the application to meet its demands in an uninterruptible way.* Horizontal scaling requires from the application to support a way of cloning itself in a way, in order to be deployed in another virtual container to support part of the demand. *Although vertical scaling seems simpler since it only requires increasing resources of the virtual container hosting the application, in fact it is not appropriate to support application's uninterruptible operation since most of the operating systems does not support on-the-fly changes (without rebooting) on the available resources (e.g. CPU or memory) of a running instance.* Thus, horizontal scaling is mostly preferred in cloud systems.

**Auto-scaling techniques** are distinguished to reactive and proactive (or predictive). **Reactive techniques** refer to those methods that react to the current system and/or application state which states are decided from the latest values of monitored variables. **Proactive (or predictive)** techniques attempt to scale resources in advance of demand by predicting the latter. Reactive techniques may prove inefficient to support uninterruptible at all times operation of the application especially when there is a sudden demand burst (flash crowd or Slashdot effect). This is due to the fact that acquiring new resources and instantiating a new execution environment (virtual container) requires a non-negligible time interval. *On the other hand proactive techniques are more promising; in the worst case they may miss to predict demand and act as a reactive technique.*

A bad performing auto-scaling technique may lead to **problems** such as **under-provisioning**; the application does not have enough resources, **over-provisioning**; the application reserves more resources than the ones really needed, and **oscillation**; scaling actions are carried out too quickly, for the application to see the impact of the scaling action.

**Auto-scaling decisions heavily rely on monitoring several performance metrics.** Their performance will depend on the quality of the available metrics, the sampling granularity, and the overheads (and costs) of obtaining the metrics [66].

Most auto-scaling techniques presented fall into one or more; considering also hybrid techniques, of the following categories:

1. Threshold-based rules (or policies)
2. Reinforcement learning
3. Queuing theory
4. Control theory
5. Time series analysis

**Threshold-based rules (or policies)** belong to the reactive category. Scaling decisions are triggered based on selected performance metrics and predefined thresholds. These reactive techniques are most commonly used by the majority of commercial cloud providers.

**Time-series analysis** is a purely proactive approach. Historical data for a monitored variable are kept as a series of data points and utilized to detect patterns and predict (forecast) future values.

Reinforcement learning, queuing theory and control theory can be used with both reactive and proactive approaches. Queuing theory and control theory rely on modeling the system. **Queuing theory** attempts to catch the relationship between jobs arriving and leaving the system [65] while **control theory** attempts to define a controller to automatically adjust the required resources to the application demands. **Reinforcement learning** attempts to learn the most suitable action for each particular state with a trial-and-error approach.

*Predicting resources needed by an application for the next period in fact requires: predicting **workload (type and volume)** [72] and **estimating resources needed** by the application to serve this kind of demand.*

Since realizing auto-scaling requires the synergy of several components providing and processing data during an application's lifetime, it is clear that **special care is required regarding representation, data handling and coordination**. In [68] the authors extend the Topology and Orchestration Specification for Cloud Applications (TOSCA); an emerging framework capturing the description of cloud application and infrastructure services, the relationships between parts of the services, and the operational behavior of these services (e.g., deploy, patch, shutdown), towards managing the dynamic scaling of cloud applications at run-time (monitor running applications, describe QoS and depict plans to scale up or down applications [67]) while describing how this extension (called Elastic-TOSCA) can be used to support a variety of analytical model-based approaches for elasticity management in complex cloud applications.

*ARCADIA framework considers the lifetime of an application, starting from its development to cloud deployment, running and termination. The fact that the application is not as usual considered from the time it is a binary executable to be deployed, surely provides an advantage. It is considered that **source code profiling information and proper annotations in conjunction to load testing if possible, prior to application deployment**, may drive hybrid auto-scaling techniques that will exhibit high performance. **Representation, data handling and coordination of auto-scaling is considered a crucial task** for the success of the developed methods.*

### 3.3.2 Application deployment and orchestration

The lifecycle of an HDA starts with **HDA development** which produces an executable which is annotated and enriched with processes capable of feeding/getting information to/from the infrastructure. Then follows **HDA onboarding** at which phase resources are initially allocated to the HDA application in order to execute under the considerations of optimally meeting the objectives and pass to the **HDA operation** phase. The next phase in HDA's lifecycle is the **HDA optimization management** at which phase undisrupted guaranteed operation of the HDA is ensured along with objectives regarding HDA operation and infrastructure usage. Finally at some time **HDA terminates** and resources allocated are released.

In order the HDA to be onboarded / embedded / deployed to the infrastructure, at least some initial resources' requirements and performance constraints should accompany the applications. Annotations on the binary/package level will provide that information to that component of the smart controller which is responsible to allocate resources so that the HDA starts its execution/operation. During operation of an HDA more resources may be required to be allocated to support its operation meeting required performance guarantees while the HDA may be required to adapt its operation to the dynamic infrastructure environment so that objectives set regarding infrastructure usage are also met.

Applications will be appropriately **annotated** at the **execution-package level** in order to facilitate **initial configuration of the infrastructure** in terms of **resources reservation and constraints**

**satisfaction** for application's proper envisaged execution; as well at the **source-code level** in order to facilitate **maintaining application's performance** during runtime and smooth **infrastructure reconfigurations towards satisfying complex objectives**.

The use of **annotations** either at the package level or at the code level is a way for the application to pass valid information to the component handling the configuration and use of the infrastructure which in our case is a part of the Smart controller. To complete the **interaction between the application and the infrastructure**, the latter through the smart controller as well, **provides information about the runtime environment** e.g. the values of various performance monitors, **exploitable by the application towards two directions**; to adapt accordingly in order **preserve its performance characteristics** and to **contribute towards the objectives set** regarding the operation of the infrastructure e.g. maximize its capacity or guaranteed performance or energy efficiency etc..

Thus, on one hand **an application requires to acquire resources and guarantees from the infrastructure to meet its objectives** regarding its execution through its lifetime while **the infrastructure provider pursues to facilitate applications' execution while satisfying its own objectives** regarding infrastructure's usage. Objectives to be met for both sides are not necessarily contradicting; however both sides should be taken into account towards mutual benefit.

Furthermore, an application apart from providing annotations and adapting to the conditions of the runtime environment should also be **developed in a form to facilitate its embedding to the infrastructure**. For example, an application requiring extra resources during execution would have more changes getting them if it is able not only to scale vertically (scale up) but to scale horizontally (scale out) as well. While an application that scales up requires more resources (e.g. CPU, memory) from its hosting environment (e.g. a Virtual machine), when it scales out it is able to initiate one or more application instances to new hosting environments (e.g. one or more Virtual machines) in order to acquire needed extra resources to serve demand. Thus, the ability to scale out increases the probability for its needs to be satisfied by the infrastructure.

On the other hand, **the infrastructure should be flexible to facilitate applications embedding**. Software defined programmable infrastructure gives the potential to illustrate such kind of flexibility. For example, an application (service) requiring secure access to it (as a policy enforcement requirement), may require access to go through a firewall or/and a packet filter. The ability to on demand instantiate a firewall and/or a packet filter as well increases the probability for application's needs to be satisfied by the infrastructure versus the case where a constant number of legacy middleboxes were operating at certain locations of the infrastructure.

*The problem we are addressing and will become clear later on when all aspects are analyzed is how to assign resources and satisfy constraints for applications requiring operating (executing) over a given infrastructure so that objectives set are met. We will refer to this problem as the **Highly Distributed Application (HDA) Embedding problem**.*

### 3.3.2.1 Application Scalability

If resources initially reserved prove to be inadequate to support an HDA's operation (e.g. an increased demand for an illustrated service) then extra resources will be requested so that the HDA to scale up (vertical scaling) or scale out (horizontal scaling). An HDA requesting to scale up will request more resources to be assigned to the already allocated execution environments (e.g. VMs and/or containers) either for all application tiers in its service chain or for a subset of them. An HDA requesting to scale out will request to initiate instances in new hosting environments (e.g. new VMs and/or containers) for either each application tier in its service or for a subset of them.

Thus, **scaling the HDA application does not imply that all of the application tiers in the HDA service chain are required to scale.** In other words **the workload of each application in the HDA network may be different during runtime.** Furthermore, **the amount of resources required by an application as it is obvious is not necessarily proportional to its workload;** e.g. doubled workload does not necessarily imply doubled or even constantly proportional in some way utilization of resources. *Not only computational complexity of the algorithms illustrated by an application but as well implementation specifics or even bugs, governs the relation between utilization of resources and workload.* At the same time, **utilization of each resource type (e.g. memory, CPU etc.) does not necessarily increase the same way as workload increases;** e.g. memory utilized may increase slowly or not at all as workload increases, while CPU demand to be increasing proportionally to the workload. Although scaling out an application may provide more choices to allocate required resources, *it seems to depend on the application if in terms of absolute portion of resources and for a specific workload, scaling up will require less or more resources than scaling out.*

**The way an HDA scales may also be dependent to the type of the workload or workload mix** requiring during runtime different applications in the HDA network to scale as needed. When **resources allocated to an HDA application prove to be excess of the needed ones then it scales down or scales in** releasing and making available resources. Releasing resources faces the same considerations as acquiring resources.

*A thorough profiling of the HDA application revealing the relation between workload and resources utilization would be highly desirable as describes in previous section; however this is not possible in many of the cases.*

*Furthermore executing an application in a virtualization environment imposes a virtualization overhead meaning that resources needed in for an application to run in a physical environment are less than those needed to run in virtual machine due to the virtualization layer.*

The **amount of resources needed is rarely static**, varying according to changes in the workload, the workload mix, and internal application phases and changes. Deciding how much resource to allocate is non-trivial since resource needs often change with time and characterizing runtime application behavior is difficult. To maintain application's proper execution, **application resource needs must be predicted in advance** in order to adjust resource allocations ahead of the needs. Reactive allocation of resources would probably lead to interruptions in applications continuous operation violating performance requirements.

If for each application the relation between workload volume and mix with the utilization of resources was known then allocating in advance of need resources would only require predicting workload volume and mix for the next period. Since this relation is most commonly not known either should be estimated or resources needs simply predicted for the next period probably according to known past patterns of usage.

### 3.3.2.2 Programmable infrastructure

Resources from a heterogeneous-diverse dynamic underlying infrastructure; a mixture of physical and virtual resources, have to be assigned, initially and during runtime, to an HDA in order to execute. Several infrastructure domains are considered, with each one controlled by a smart controller. An HDA embedding may illustrate within the same domain (intra domain) or across several domains (inter domain).

Each infrastructure makes available several Programmable Resources which span from configured IaaS frameworks, programmable physical switching/routing equipment, programmable firewalls, Application Servers, modularized Software Entities (Databases, HTTP Proxies etc.).



An infrastructure maybe thought as a network of geographically dispersed **Resources Repositories** where one or more **Resources Pools of a node type** exist. Recursion may stand either for resources repositories or pools; e.g. a resource pool within a resource pool. For example a repository maybe a data center where several hypervisors as resource pools for Virtual machines (VMs) exist, several VMs as resource pools for containers (e.g. Linux Containers), several Middleboxes of several types illustrated in VMs (e.g. firewall elements, packet filters etc.), physical machines, physical or virtual switching/routing equipment.

An HDAs service chain maybe **illustrated as an overlay** either within a repository or a resource pool or across various repositories of a domain or across various repositories several domains.

*It becomes obvious that **an ideal environment** would be a physical infrastructure able to have the flexibility of **illustrating every possible node type or link as a virtual one upon request and on the fly while not depending on specialized hardware** existing in specific locations. In such an environment an HDA network of apps would be possible to map to virtual nodes and links in order to operate within a single resource pool of VMs.*

### 3.3.2.3 Highly Distributed Application Embedding (HDAE) problem

An HDA as a service chain requires executing over a given infrastructure. Required resources should be allocated in order each application tier and network function in the HDA service chain to execute while data links among them should be illustrated as needed. Furthermore, required by the HDA performance constraints should be met and required policies realized (e.g. secure access to the HDA, redundancy and fault tolerance etc.).

At the same time realization of the HDA service chain over the infrastructure should be done in a way that objectives set regarding infrastructure operation and usage should be met. For example, such objectives may be minimizing the energy consumed from the infrastructure's operation or maximizing the service capacity of the infrastructure (number of HDA applications that operate over the infrastructure) or maximizing performance guarantees to operating HDAs.

Several HDAs requests for embedding and operating over the infrastructure come at arbitrary moments in time. HDAs are embedded/mapped to the infrastructure and operate. During operation an HDA may require more resources than the ones allocated to support its operation or may release resources allocated but not needed. At some time an HDA terminates its operation and releases/makes available all the resources allocated for its operation.

We call the problem of embedding an HDA to a given infrastructure under the considerations and restrictions previously mentioned as the **HDA embedding (HDAE) problem**. In the **offline version** of the HDAE problem all requests are apriori known while in the realistic **online version** a request is known only as it arrives in time.

*The problem maybe **expressed as an optimization problem** according to which, given a specific request with specific resources demands and a resources constrained infrastructure (substrate), we seek the positions in the infrastructure where applications will execute at allocated resources so that constraints are satisfied and a cost/revenue function reflecting the objectives is minimized/maximized.*

*Obtaining a **solution to the problem** may be quite computational intensive and time consuming, especially as the input to the problem gets large. This is not desired in our case where decisions should be taken on the fly especially when **an HDA is in operation mode** and continuous operation with quality characteristics should be maintained. This strict restriction relaxes somehow when **initial HDA embedding** (HDA is not yet in operation phase) is considered.*

**Several factors should be avoided and expressed as constraints** when placing and assigning resources to HDAs in an optimal way. Such indicative factors to be avoided are **Resource contention**;

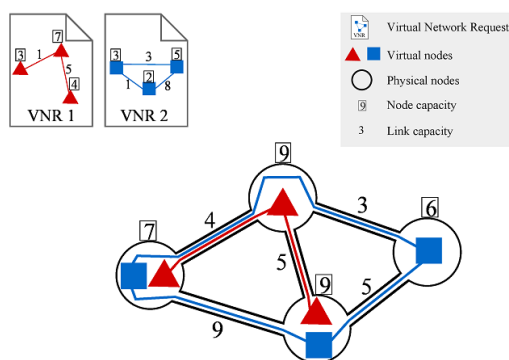
a single resource or utility is being accessed by two different applications at the same time, **Scarcity of resources**; there are not enough resources to deal with the workload, **Resource fragmentation**; valuable resources lie around in a highly disorganized manner, **Over-provisioning**; more resources are being assigned than required, and **Under-provisioning**; not adequate resources assigned.

The HDA embedding problem may be considered as an extension of already studied in the literature problems such as the **Virtual Network Embedding (VNE) problem**, the **Virtual Data Center Embedding (VDCE) problem** and the **Cloud Application Embedding (CAE) problem**, to which we will refer to the related work that follows.

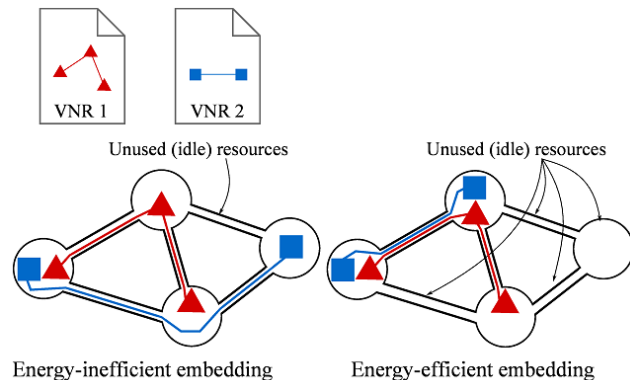
### 3.3.2.4 Related Work to the HDA embedding problem

The **Virtual Network Embedding (VNE) problem** [74] is the problem of assigning physical resources from a substrate network to a requested virtual network, with constraints to virtual nodes and virtual links. Given a substrate network with capacity constrained substrate resources and a Virtual Network request with capacity constraints on Virtual nodes and links the task is to assign virtual nodes and links to substrate nodes and links and allocate resources so that an objective function (reflecting e.g. VN acceptance ratio or overall energy consumption or else) is maximized or minimized accordingly.

The VNE problem in the majority of works has been studied for assigning nodes and links to a VN request by considering respectively only CPU and bandwidth as the critical resources (see Figure 3-11). This optimality has been computed with regard to several different single or multi objectives, such as service capacity, QoS, economical profit, survivability, energy-efficiency (see Figure 3-12), security of the networks.



**Figure 3-11: Source [73] Virtual Network Embedding (VNE) example**



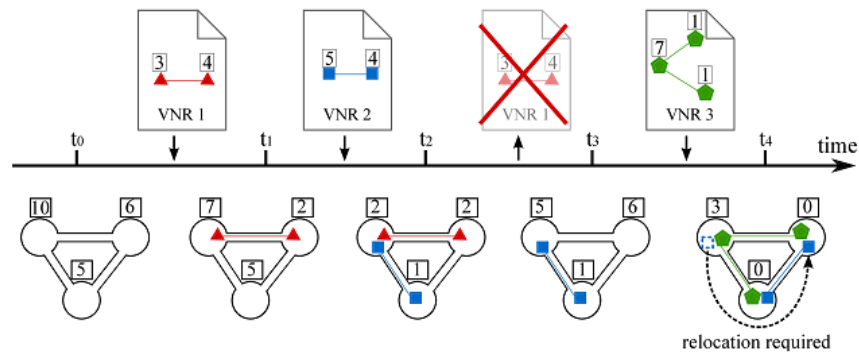
**Figure 3-12: Source [73] Virtual Network Embedding (VNE) example with energy efficiency as objective**

Finding optimal solution to the VN embedding problem is computationally intractable and time consuming. Even in the offline case, the problem is NP-hard [75, 76, 77, 78, 79]; there is no polynomial time algorithm to give the optimal solution, as it is related to the multi-way separator problem. A *polynomial time algorithm is an algorithm which its running time is upper - bounded by a polynomial in the size of the input for the algorithm*. For large problem sizes (i.e. large substrate networks and virtual network requests size) the time to find the optimal solution becomes unaffordable. The online problem becomes even more difficult to solve. *Exact solutions resulting to a global optimum are considered appropriate only for small instances of the problem*. Efficient heuristic-based solutions not fixed to obtain a global optimum and resulting with good solutions in short execution time suit this



problem case. Heuristic solutions may phase the problem that they can get stuck in a local optimum that can be far away from the real optimum. Metaheuristic solutions improve the quality of the result by escaping from local optima in reasonable time.

VNE is an online problem. VN requests (VNRs) will not be known in advance. Instead, they arrive to the system dynamically and can stay in the network for an arbitrary amount of time. VNE algorithm has to handle the VNRs as they arrive, rather than attending a set of VNRs at once (offline VNE). Remapping of whole VNs or partial VN relocation is necessary to ensure performance and objectives; see Figure 3-13 where a needed relocation is exhibited in order a new virtual network request to be accepted and embedded to the substrate.



**Figure 3-13: Source [73] An example of reconfiguration in Virtual Network Embedding (VNE)**

VN assignment shares similarities with older problems such as the embedding of Virtual Private Networks [80], with the difference that in that case there are only bandwidth constraints, as well as with the network testbed mapping problem [79, 81]. In the VN embedding problem there are capacity and placement requirements on both virtual nodes and virtual links while a node and a link is shared by multiple VN requests. This makes the problem hard. There are a few works attempting to address the problem more generally, while the majority restricts the problem by studying it in specific VN topologies or considering only the offline case or assuming infinity capacities or even ignoring one part of the requirements (either for nodes or the links).

In [75] the authors recognizing that all of previous research focused on designing heuristic-based algorithms with clear separation between the node mapping and the link mapping phases provide online VN embedding algorithms with a better coordination between the two phases in order to expand the solution space. They consider CPU as the critical resource of a node and bandwidth as the critical resource of a link. A VN request consists of virtual nodes with CPU constraints and virtual links with bandwidth constraints while at the same time location constraints for each virtual node are also considered. The authors produce an augmented graph (an extension of the physical network graph) including meta-nodes (which are the virtual nodes requested), with meta-links (with unlimited capacity) to those physical nodes that satisfy the constraints. This mapping facilitates their proposed solution. They treat each virtual link with bandwidth constraints as a commodity consisting of a pair of meta-nodes. As a result, finding an optimal flow for the commodity is equivalent to mapping the virtual link in an optimal way.

In [77] the authors focus and provide heuristics on two versions of the VN assignment problem, one without reconfiguration and the other with selective reconfiguration of the most critical VNs. In [78] the authors study the design of the substrate network to enable simpler embedding algorithms and more efficient use of resources, without restricting the problem space. They simplify virtual link embedding by allowing the substrate network to split a virtual link over multiple substrate paths and by employing path migration to periodically re-optimize the utilization of the substrate network.

The **Virtual Data Center Embedding (VDCE) problem** [82] is the problem of mapping Virtual Data Center (VDC) components (e.g., virtual machines, virtual switches and links) onto physical nodes and links. VN embedding models differ from VDC embedding in that they only consider CPU and network resources while in VDC embedding other resources such as memory and disk also need to be considered. Furthermore, in the VNE problem, the substrate network usually comprises tens to hundreds of servers and a virtual network request usually includes tens of VMs. In contrast, in the VDCA problem, the data centers can comprise thousands of servers and a VDC request can include hundreds of VMs. These aspects of the VDCE problem impose significant scalability requirements on the scheduling algorithm. VDCE can be seen as a combination of the bin packing problem and the multi-commodity flow problem, which are both known to be NP-hard [84]. Presented works in the literature propose heuristics to provide a solution considering specific data center design architectures in order to restrict and relax the problem [82, 83] while it has been studied for deployments in distributed infrastructures (see Figure 3-14) as well [84].

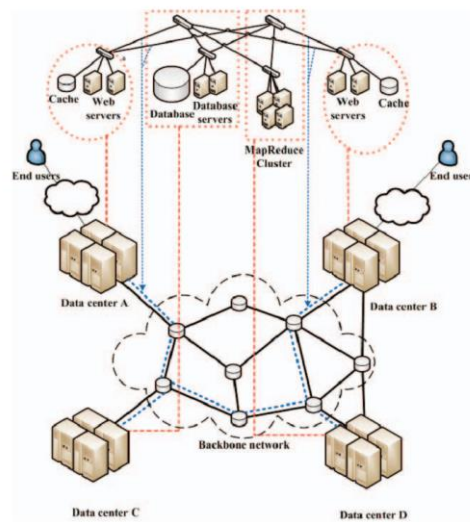


Figure 3-14: Source [84] Virtual Data Centre Embedding (VDCE) example

The **Cloud Application Embedding (CAE) problem** [85] is the problem of mapping distributed applications as structured systems that include not only computational and storage entities, but also policy entities (e.g. load balancers, firewalls, and intrusion prevention boxes) onto cloud data centers.

In Figure 3-15 an example of a two-tier application presented in [85] is shown. The two-tier application is represented as a flow security graph. Tier 1 and 2 are represented as nodes  $u_1$  and  $u_2$  respectively. Middleboxes are firewall  $F_i$ , load balancer  $LB_i$ , intrusion prevention system  $IPS_i$ ,  $i = 1, 2$ . The tuple  $(50, 1)$  means that it needs 50 virtual machines and one fault domain. The Internet clients are denoted as  $u_e$  with zero demands on virtual machines and fault domains. The application needs one copy of each middlebox. Each middlebox requires one fault domain. There are no bandwidth demands on links since a best effort service is considered.

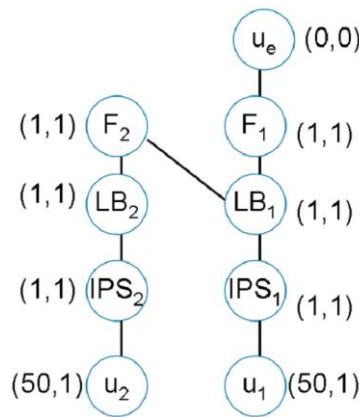


Figure 3-15: Source [85] Cloud Application Embedding (CAE) example

Given a cloud data center with servers (hosts several VMs) and middleboxes (hardware based or software based) with fixed placements the authors of [85] study the problem of embedding an application encoded as a Flow Security Graph to the infrastructure and propose an online algorithm. Various specific data center designs are considered as well as middleboxes placements.

*Their study motivates the **need of flexible policy enforcement mechanisms** such as Openflow and pswitch, and the design of policy-aware data center network architecture.*

*HDAE problem is an extension of VNE, VDCE and CAE problems. First impression suggests it is a much harder problem. However taking advantage of the flexibility of the infrastructure considered as well as the ability to lead appropriately the development of applications to be embedded, initial intuition suggest that the problem may be severely relaxed.*

### 3.3.2.5 Deployment Scheduling Approaches in Platforms

Briefly, the available scheduling policies in three popular cloud computing platforms (Openstack, Cloudstack, OpenNebula) are presented. The purpose of these deployment scheduling policies is to place VMs in available hosts according to selected criteria. As it will become obvious these policies offered with these platforms are simple and generic; however they provide a base for more complicate deployment policies to be devised.

The OpenStack's schedulers filter hosts according to the following criteria:

- Have not been attempted for scheduling purposes (*RetryFilter*).
- Are in the requested availability zone (*AvailabilityZoneFilter*).
- Have sufficient RAM available (*RamFilter*).
- Can service the request (*ComputeFilter*).
- Satisfy the extra specs associated with the instance type (*ComputeCapabilitiesFilter*).
- Satisfy any architecture, hypervisor type, or virtual machine mode properties specified on the instance's image properties (*ImagePropertiesFilter*).
- Are on a different host than other instances of a group (if requested) (*ServerGroupAntiAffinityFilter*).
- Are in a set of group hosts (if requested) (*ServerGroupAffinityFilter*).

CloudStack offers the following Scheduler discipline:

- *FirstFit*: This policy selects the first node with enough free resources to host the newly created virtual machine.

Scheduler policies available in OpenNebula's Match-making scheduler are:

- *Packing*: This policy minimizes the number of cluster nodes in use by packing the VMs in the cluster nodes to reduce VM fragmentation (it uses those nodes with more VMs running first).
- *Striping*: This policy maximizes the resources available to VMs in a node by spreading the VMs in the cluster nodes (it uses those nodes with less VMs running first).
- *Load-aware*: This policy maximizes the resources available to VMs in a node by using those nodes with less load (with more free CPUs).
- *Fixed*: This policy sorts the hosts manually, by using a priority attribute assigned to each node.

## 4 ARCADIA Use Cases

---

### 4.1 Energy Efficient Cloud Management

**Name.** Energy-Efficient Cloud Management

**Objectives.** A Cloud Service Provider exploits the ARCADIA framework to manage its infrastructure in the most energy efficient manner, while providing its users the required service level.

**Description.** ACME Inc. is a Cloud Service Provider whose business is based on the Infrastructure-as-a-Service model. ACME Inc. has recently increased its turnover by cutting down operational costs, especially the electricity bill for running an installation comprising thousands of computing servers and networking devices, and by promoting its image as a green and environmental-friendly company. The key technology for ACME Inc. is the Smart Controller, a software tool that supports the management staff in taking decisions about the optimal placement of workload according to advanced criteria of energy-efficiency. The Smart Controller also parses metadata information incorporated into the software bundles deployed by its clients, in order to know their specific Quality of Service requirements (communication bandwidth and latency, service downtime, resilience to faults, availability and dependability). The Smart Controller exploits the cloud management software (e.g., OpenStack) to depict a full picture of the current run-time environment, consisting in the current assignment of services and applications to servers, the current electricity consumption, and the energy profiles of servers and networking devices. Based on this information, the Smart Controller finds the optimal allocation of computing and networking resources that minimizes the overall energy consumption, while providing enough capacity to cope with the short-time fluctuations in the workload, so to meet all QoS requirements. The Smart Controller automatically applies the new configuration by migrating applications and services among the installed servers and by updating network devices accordingly.

**Involved roles.** Service Providers benefit of an automated tool for applying their policies and optimization objectives on the underlying infrastructure, in a coherent manner with requirements of their users. The Smart Controller takes care of combining information from thousands of physical resources and to find optimal allocation strategies compliant with thousands or millions of application's profiles; this task would be unfeasible for human staff. Software Developers should include specific performance and QoS constraints in their code, together with profile information about the amount of traffic and workload expected.

**Innovation.** Current cloud management software (e.g., OpenStack) provides administrators with the knowledge of the current service allocation in the underlying physical infrastructure and the ability to migrate services among the servers for management purposes (e.g., hardware maintenance, software upgrade); there are both graphical dashboards and command line interfaces to accomplish these tasks. However, information about power consumption and energy profiles of the underlying infrastructure is not available. ARCADIA will enhance current cloud management software with energy-aware

aspects, by integrating the Green Abstraction Layer and its possible extensions. In addition, the Smart Controller will shift the responsibility of seeking for optimal configurations and of applying them from human staff to an automated tool, bringing clear benefits in terms of scaling, optimality, response time, configuration errors. Finally, the characterization of application profiles and the incorporation of requirements at code level will represent an additional facet in the decision-making process, still not considered in energy optimization frameworks.

**Relationships to the ARCADIA framework.** This use case identifies a specific function for the Smart Controller, when it is deployed by the Cloud Service Provider. This use case also exploits context-aware model information available as metadata in the source code.

**Challenges.** i) To integrate power consumption and energy profiles in cloud management software. ii) To define suitable allocation strategies for energy-efficient cloud operation under performance constraints coming from application profiling and user requirements. iii) To enhance cloud management software with suitable APIs for service allocation and infrastructure configuration. iv) To include profiling information and QoS requirements in the Context-aware programming model. v) To exploit multipath routing, rate adaptation, standby and other energy saving techniques in the cloud network.

## 4.2 Annotated Networking

**Name.** Annotated networking

**Objectives.** Software developers are provided the ability to set up their networking functions by annotated metadata in the source code.

**Description.** Bill is a software developer at JustSoft.com, who is creating an application made of several components. He designs the service-chain model for his application (using tools like TOSCA or similar) and implements all software modules. Bill exploits a context model that allows him to include specific requirements for deployment and execution of its applications, through the use of source-code annotations. Among other aspects (e.g., software dependencies, configurations), the context model allows Bill to define network-related functions that should be set up for his application, like horizontal scalability of some components, privacy and confidential issues in data exchange, Quality of Service requirements, resilience to faults of software components, data locality constraints, secure interconnection to the Internet. Annotations are embedded in the object code as metadata and the final package is delivered to his colleague John, who is responsible for deployment. JustSoft.com has its own data center, but John also uses cloud services borrowed from other providers. For most part of his job, John relies on the Smart Controller deployed at JustSoft.com. The Smart Controller parses the code metadata, considers the different kinds of requirements (data locality, security, QoS) and decides the optimal placements of the application's components inside the local and remote clouds. Taking into consideration the service chain, the placement plan and the annotated requirements, the Smart Controller derives the network topology and infers all the network functions needed to interconnect all the application's components: virtual switches/routers, load balancers, NAT/firewalls, packet shapers, traffic queues, encryption/decryption boxes, and so on. The Smart Controller at JustSoft.com sets up most of these functions as Virtual Network Functions in the local and remote clouds, by interacting with its peers deployed by other service providers.

**Involved roles.** Software Developers exploit enhanced context models to embed the description of network functions required to build their service chains. Their requirement should include performance constraints, data locality, load balancing, hot-standby for resilience, security and privacy aspects. DevOps users mostly benefit from the presence of the Smart Controller, which takes the



burden of managing many deployment requirements, of finding optimal placement strategies and of automatically configuring cloud management software. DevOps basically remain in charge of supervising the whole process and to establish relationships with cloud providers.

**Innovation.** Today, deployment of applications and services in the cloud is a tedious and error-prone process, carried out by DevOps through a number of scripting languages. DevOps are responsible to find the optimal placement of the application modules in different clouds, if data locality and bandwidth constraints make this solution convenient. The ARCADIA Smart Controller will take over most of this work, will enable to take into consideration more variables in the decision process, and will find optimal solutions in less time. Moreover, software developers will be able to demand networking functions directly, through the context model.

**Relationships to the ARCADIA framework.** This use case relates to the main components of the ARCADIA framework: the context model and the Smart Controller. It defines how software developers should set up networking functions in the application's lifecycle, starting from the context model, to the embedding of metadata in the code and to the parsing and interpretation by the Smart Controller. Moreover, this use cases also envisages the need for interaction and federation among Smart Controller deployed in different cloud domains.

**Challenges.** i) To integrate networking aspects in the ARCADIA context model. ii) To build complex network topologies over multiple clouds, including switching devices, security middleboxes, QoS functions. iii) To exploit the concept of Virtual Network Functions, in order to avoid the need for specific hardware. iv) To account for inter-cloud issues, like federation of Smart Controllers.

## 4.3 Remote Surveillance

**Name.** Remote surveillance

**Objectives.** Highly-Distributed Applications exploit data locality and In-Network Programmability envisaged for Telecoms' access networks.

**Description.** Eye@Home Co. is a dynamic and enterprising company providing home security solutions. It is undertaking an emerging business model, based on "softwarization" of remote surveillance systems. According to this model, the company directly gathers data from several heterogeneous surveillance hardware installed at clients' homes (cameras, motions sensors, intrusion detectors, fire and gas alarms, baby monitors), processes them, and provides different kinds of alarms, depending on the user wishes. This approach replaces expensive management hardware at home with more flexible and manageable software in the cloud, enabling the company to provide very advanced monitoring and surveillance services based on facial and motion recognition, context-awareness and on-line dynamic reprogrammability. The service provided by Eye@Home notifies the clients by several media (text messages, voice calls, emails) and gives them the opportunity to react by accessing cameras, by checking the status of sensors, by activating specific devices (turning on lights, closing windows), by calling emergency numbers, and so on. Given the need to continuously monitor video from several cameras, Eye@Home has built a Highly-Distributed Application, so data processing modules can be placed close to the clients' homes, by exploiting cloud services offered by the same Network Operators that already provide network access. To implement the Eye@Home application, software developers have exploited a rich context model, in order to specify data locality principles (place an instance of the data processing module close to each client's home), security requirements (use data encryption, authentication and integrity while exchanging messages), dependability and resilience (load balancing, hot-standby), and Quality of Service constraints (transcode the video from cameras according to the current user's device and access network, send video only when required,



etc.). Eye@Home has deployed its service in the cloud infrastructure provided by ACME Inc., where a Smart Controller is present and takes the responsibility to set up replicas in Network Operators' clouds, to scale them horizontally for redundancy and for managing an increasing load, and to set up all virtual network functions in the different clouds to interconnect the distributed modules in a secure and dependable way.

**Involved roles.** Software Developers exploit i) software development paradigms to build Highly-Distributed Applications and ii) context models to denote sets of relationships, dependencies, service chains, constraints that drive and automate the deployment process. DevOps provide minor inputs during the deployment phase, and are mainly responsible to include all cloud systems that enable to place part of the application as close as possible to each client. Network Operators provide computing services following the IaaS model in addition to basic network connectivity, enabling third-party software providers to exploit locality principles in placing their applications and novel business models based on virtualization of home devices. Network Operators also deploy Smart Controllers in their clouds, in order to allow federation with other domains and optimal placement of HDAs.

**Innovation.** Today, many software applications run on dedicated commodity hardware at home (for instance, set-top boxes, home automation systems, media stations, game consoles, home surveillance systems). There are several drawbacks behind this approach: the environmental impact (for production and disposal of many short-lived devices), large energy waste (due to the need to keep most devices always on to be ready-to-use and to be controlled remotely for maintenance and updates), and heavy capability limitations (due to limited computing and storage resources available). The software development paradigms and the context models in ARCADIA, together with In-Network Programmability tackled by other projects (like the Horizon 2020 INPUT project), will enable to create virtual instances of many appliances and to place them very close to the home, with very similar service levels but unprecedented opportunities in terms of energy-saving, software customization, service composition and business models.

**Relationships to the ARCADIA framework.** This use case relates to most components of the ARCADIA framework. It needs software development paradigms to build Highly-Distributed Applications in a fast and error-prone manner, context-aware models to drive the deployment of the application according to locality principles and QoS constraints, Smart Controllers to take over the deployment process and to deal with the complexity of setting up Highly-Distributed Applications that interacts with many external heterogeneous hardware components, cloud management strategies that combine efficient execution of a huge number of software components that may be idle most of the time with strong guarantees on response times, availability, dependability and security.

**Challenges.** i) To provide suitable software development paradigms for building Highly-Distributed Applications. ii) To include locality principles and QoS requirements in the ARCADIA context model. iii) To tackle the need for deploying multiple instances of the same software module with different configurations, to deal with heterogeneous monitoring hardware deployed at home. iii) To guarantee strong QoS constraints, in order to have the application working as it were in the house LAN. iv) To ensure proper isolation between data from different clients and strong security measures.

## 4.4 Enterprise Networking

**Title:** Enterprise Networking

**Objectives:** Develop an application for the efficient management of heterogeneous network infrastructure provided to an enterprise customer with branches distributed throughout large geographical areas.

## Description

In this use case, we consider the case of an Enterprise customer, which has an extensive network with customized settings, including multiple Branch Offices (BOs) and Headquarters (HQ) distributed throughout large geographical areas. We also assume that the Enterprise may use Cloud services as well as have its own data center facilities. In this context, we consider that the Enterprise acquires resources from a Network “Infrastructure Provider”.

It is worth highlighting that, in state-of-the-art Enterprise networks, the administrators need to deal with multiple types of middleboxes such as CPEs, Firewalls, Wan Optimizers, DPIs, Load Balancers (LBs) etc. This approach leads to rigid and heterogeneous ecosystems composed of standalone appliances supplied by many different vendors. The typical consequences derived from this practice are basically the following:

- 1) high CAPEX since corporate networks are based on middlebox cascades that usually demand substantial resource replication at BOs and HQ (e.g., with 20 sites and 5 boxes per site on average, this means 100 physical boxes that need to be maintained);
- 2) high OPEX, since maintaining the different boxes throughout the corporate network is cumbersome, time consuming, and requires considerable skills;
- 3) slow innovation cycles (e.g., it is complex to add new appliances in corporate networks and/or novel features to existing ones);
- 4) lock-in issues, i.e., hard to move from one vendor/solution to another without experiencing operational pain;
- 5) considerable limitations for more advanced and ambitious approaches, since it is hard to adapt and dynamically change the pre-configured processing sequences based on the context or specific outcomes (e.g., shortcut some middleboxes and redirect to others right after deploying a new B2B service).

In addition, almost all Enterprise networks deployed today are multi-connected, i.e., each site has at least a primary and a backup link, and they are connected to their providers by means of different forms of broadband access technologies, such as optical links, MPLS, VPLSs, Ethernet (VLAN, VxLAN, etc.), xDSL/2, etc. At present, there is a clear tendency toward cheaper and less reliable connectivity such as DSL, though, as a countermeasure, the Enterprises are increasing their outdegree and using WAN optimizers in order to keep their availability at 99,999%.

**Involved roles:** Network Infrastructure Providers, Network Administrators, Enterprises with increased networking needs.

## Innovation

In this use case, our research will be particularly focused on the following innovative subjects.

Outsourced middleboxes in the form of “Appliance as a Service” (AaaS) or in other words “Provider Provisioned Virtual Appliances” (PPVAs). These virtual appliances could be even packed and seamlessly combine multiple solutions in a single virtual bundle that can be exposed to enterprise customers. We plan to enable an open selection, combination, and utilization of virtual appliances, including appliance consolidation. Also observe that traditional services such as DNS, SMTP, Web, Intranets and the typical IaaS, PaaS and SaaS will also be offered as components through the ARCADIA platform. In this framework, customization is the king, but simplicity is a key as well, so we plan to work on models where providers could offer packs with predefined bundles along with a consolidated management interface through the Enterprise Application Management GUI. The latter will offer a repository, mainly composed of services and SDN-based Apps, which will basically control and manage the corresponding VNF-FGs transparently to Enterprise Consumers.

Customizable traffic optimization for the BOs and HQ by smartly exploiting the multi-connectivity in place. We will give special attention to cheap L2 connectivity, including heterogeneous broadband access technologies: xDSL2, optical/Ethernet, radio, etc., and the strengths of extending the VNFFGs so as to reach the enterprise premises too. In other words, our research will be mainly focused on a specific type of virtual appliances, namely, WAN optimizers. In the model that we conceive, the optimization functions (i.e., the VNFs) will be split between the provider and the Enterprise premises, and these functions will work in concert and in a customizable way according to the Enterprise policies. These policies will cover aspects such as traffic distribution, reliability criteria, security considerations, etc. It has to be highlighted that with ARCADIA the CPEs at corporate premises can be saved or at least they can be completely virtualized.

Private Internets on the fly, i.e., interconnected corporate networks on demand. It has to be pointed out that NFVbased networks are much easier to deploy and to adapt to logical connectivity requirements. As long as a chain of programmable equipment can offer physical connectivity and e2e transmission, enterprise networks can be dynamically connected to (and latterly disconnected from) their own customers and/or corporate partners in sorts of private Internets on the fly, e.g., for product demonstrations, for support tasks on specific products, for creating virtual room for meetings, etc. In summary, our research will explore how Enterprises can privately connect to their customers and partners on demand for specific events through an ARCADIA based application, according to a well-defined set of service requirements.

All in all, this is a very rich use case, offering scenarios far more advanced than state-of-the-art Enterprise settings. Our research and expected outcomes offer significant exploitable results, leading to advantageous scenarios for all the stakeholders involved, that is, for enterprises, for providers, for vendors fostering openness and programmability, and for external software developers.

#### **Relationships to the ARCADIA framework**

Based on the facts mentioned above, a Highly Distributed Application developed on the ARCADIA platform, will set the basis and deliver the essential components that will be required in this use case. With ARCADIA, Enterprise networks can be substantially simplified, so CAPEX and OPEX can be dramatically reduced. And more importantly, since the advantages and footprint of SDN and NFV will be extended and become readily available for corporate customers in the form of components that can be chained using the ARCADIA platform, the innovation pace will be much faster.

#### **Challenges**

- i) Facilitate network administrators in configuring a large set of devices
- ii) Reduce CAPEX and OPEX
- iii) Develop, chain and deploy numerous SDN and NFV functions, using the Smart Controller, so as to create powerful and customized solutions according to customer needs.

## **4.5 IoT/Smart Home**

**Title:** IoT / Smart Home

**Objectives:** Exploit modern technology, based on ARCADIA platform, in order to facilitate the everyday lives of the members of a family, while endorsing the efficient use of valuable resources (like e.g. electricity, water, food etc) and enhancing the social interactions between them.

**Description:** The Arcadians are a family that knows how to take advantage of the modern technology goods. Due to the recent financial crisis and because, in addition, they have a keen interest on ecological issues and healthy living, they are using several applications that facilitate them in achieving their personal and family targets in energy saving, healthy eating, food waste constraint etc. In parallel,

they have the chance to discuss all these issues and share moments and relative material (photos, videos etc) through an extended home infotainment system. In their personal devices (smart phone, tablet, smart watch etc) they are running a smart home Highly Distributed Application featuring three main functions as presented in the following. i) Energy: this function, allows the user to specify a plan for controlling electronic equipment in the house such as the TV, printers, routers etc. in order to save energy. E.g. it is possible to turn them off completely, instead of staying at standby mode. It can also be applied to electrical equipment, e.g. when leaving a room and nobody is there any more, the lights can be turned off automatically. ii) Healthy living: the user selects a suitable diet and the application provides the necessary guidelines for following this diet, based on input from the user on what food has been consumed. Moreover, the application exploits information on shopping, cooking and eating in order to maintain a view on food supplies, predict future needs and propose a shopping list. iii) Infotainment: the third function is related to the entertainment of the user, as well as the provision of information related to the other two functions. It includes features of a social networking application, with enhanced security and content management, adjusted to the needs of a family or a close circle of friends etc. And of course it provides access to a variety of multimedia material, from movies to family videos and photos etc.

**Involved roles:** Energy suppliers: are interested in convincing their customers to behave more responsibly, as well as to participate in marketing actions such as flexible electricity cost etc. Consumers: are interested in saving money and helping the environment. Of course they are also interested for their health and what their dear family and friends do. Application developers: are looking for new attractive applications that can become indispensable for their users. Home electronics equipment suppliers: they are searching for services that make their equipment to function in a compensatory way, from both financial and user experience perspectives.

**Innovation:** Energy efficiency: there will be a clear focus of the system towards energy saving, rather than just offering an automated control of the home “functions” (heat, air conditioning, lighting, windows, etc). Humidity, temperature, human presence and various other sensors will enable the creation of customized plans according to needs of the home residents. Healthy living and nutrition: through the easy to use GUI the user will be able to register in a few steps every action related to food, from meals and quantities, to what has been stored in a tupperware in the fridge. It will also provide aid in shopping necessary food based on the desired diet and what has left in the kitchen self and several other facilities. Infotainment: there is a home network and content server, able to manage and stream content in a secure and trusted way. Family members can exchange views, experiences, synchronize, communicate and feel together while away.

**Relationships to the ARCADIA framework:** The various service modules will be chained in order to form an application with the help of the ARCADIA platform and the necessary Smart Controllers. The provided application will be tailored to the needs of the specific family and it will be possible to launch it in a wide range of devices, using again the ARCADIA software deployment paradigm. Moreover, the service modules will be able to adapt to their execution environment, based on the incorporated annotations according to the ARCADIA development paradigm. Finally, the smart home application described in the previous will be surely a highly distributed application, since it will involve numerous input sources and user equipment.

**Challenges:** i) To reduce energy consumption in the home ii) To reduce wasting of food iii) To help the family as a whole and each person individually to eat more healthy iv) To stimulate family

members in taking the desired actions so as to achieve the aforementioned challenges v) To inform and entertain family members and not only (core family, broader family, friends etc.)

## 5 Highly Distributed Applications and Programmable Infrastructure Requirements

---

### 5.1 Distributed Software Development Paradigm Requirements

The ARCADIA software development paradigm is targeting at the specification and development of a suitable **development toolkit** along with the appropriate level of abstractions that are going to facilitate software developers to design and develop **re-configurable highly distributed applications**.

The development paradigm has to provide simple and high-level abstractions for **concurrency** and **parallelism**. Given that each application will be represented in the form of a **service chaining graph of software components** with a set of relationships and dependencies among them, the inclusion of the notion of concurrency and parallelism within each component (e.g. component specific functions that can be executed in parallel) as well as among the components (e.g. software components in the service chaining graph with no blocking behavior among each other) is required.

The developed software has to support high levels of **responsiveness**, **resiliency**, **elasticity** and **asynchronous mode of communication**. A set of autonomic (self-\* functionalities, e.g. self-healing) and fault-tolerance characteristics have to be inherently supported within the software programming paradigm. Components developed based on the support of the afore-mentioned aspects are considered as **reactive components** (Figure 5-1), able to react and dynamically adapt to the conditions on their operational environment.

With regards to the communication mode, **message-driven communication models** have to be adopted. Actually, **non-blocking behavior** among the instantiated functions per component has to be ensured through asynchronous and event-driven programming models. The development of message-driven and highly responsive applications based on the interaction among the individual software components ensures **loose coupling**, **isolation** and **location transparency**, making it possible for applications to be deployed on demand and being able to handle failures without the need for centralized orchestration.

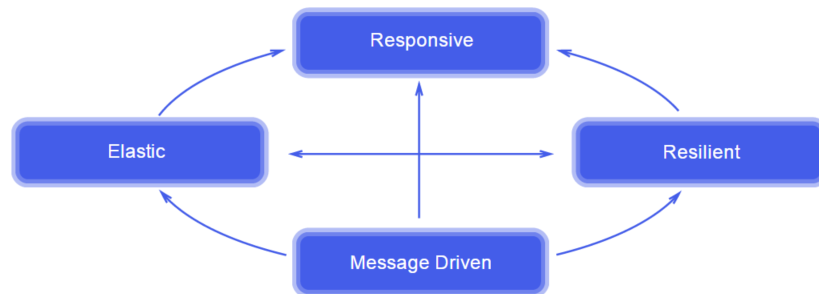
With regards to **resiliency**, reactiveness in cases of failure has to be supported. Resilience can be achieved through replication of the supported functionalities, isolation of the provided functionalities per component and appropriate delegation of roles among components. Isolation of software components is going to ensure that parts of the system can fail and recover without compromising the overall application while high-availability can be ensured through replication in critical components. **“Let-it-crash” models** have to be adopted towards this direction. Based on such models, when a process crashes, it neatly exits and sends a message to the controlling process which can take action. Actually, an assumption is made that some other process (linked process) in the system will observe the death of the process and take appropriate corrective actions. The adoption of such models can provide guarantees for the **design and development of highly fault-tolerant systems that self-heal and never stop providing the envisaged functionality**.

With regards to **responsiveness**, given the message-driven communication models among the software components, provision of rapid and consistent response times are required along with the establishment of reliable upper bounds in order to be able to achieve a desired QoS level. Detection of



faults has to be identified shortly and the undertaking of corrective actions has to be triggered immediately.

With regards to **elasticity**, the developed components have to be able to scale, in an horizontal and vertical way. Responsiveness under various workloads has to be supported in parallel with optimal use of the available resources.



**Figure 5-1: Properties of Reactive Systems based on the Reactive Manifesto**

<b>ID</b>	SOFT.1
<b>Title</b>	Provide simple and high-level abstractions for concurrency and parallelism
<b>Role</b>	Software Developer
<b>Description</b>	Adopt models (e.g. Akka Actor Model) that raise the abstraction level for concurrency and parallelism and provide a better platform to build scalable, resilient and responsive applications.
<b>Constraints</b>	-
<b>Priority</b>	Top
<b>Architectural Part</b>	Software Development Paradigm

<b>ID</b>	SOFT.2
<b>Title</b>	Be based on message-driven communication models
<b>Role</b>	Software Developer
<b>Description</b>	Software development based on processing of messages asynchronously using event-driven models. Process events and generate responses (or more requests) in an event-driven manner. Focus should be given on the application's workflow—how the messages flow in the system—instead of low level primitives like threads, locks and socket IO.
<b>Constraints</b>	Adoption and appropriate extension of software development paradigm that is based on message-driven communication models.
<b>Priority</b>	Top
<b>Architectural Part</b>	Software Development Paradigm

<b>ID</b>	SOFT.3
<b>Title</b>	Support stateless or state-migration mechanisms
<b>Role</b>	Software Developer



<b>Description</b>	Non-blocking behaviors have to be supported within the software components given the dynamic nature of their deployment and re-configuration as well as the adoption of let-it-crash models. Stateless or state-migration mechanisms have to be supported for this purpose. State-migration has to be based on specific components within the ARCADIA architecture.
<b>Constraints</b>	Stateless or state-migration mechanisms developed have to take into account the dependencies in the provided service chaining graph.
<b>Priority</b>	Top
<b>Architectural Part</b>	Software Development Paradigm

<b>ID</b>	SOFT.4
<b>Title</b>	Support vertical and horizontal scalability matters
<b>Role</b>	Software Developer, Smart Controller
<b>Description</b>	Use of remoting technologies for interconnection software components in a peer-to-peer fashion. Horizontal scaling can be realized through the dynamic instantiation and interconnection of similar components. Vertical scaling is associated with usage of more resources in the same node.
<b>Constraints</b>	Adoption of message-driven communication models.
<b>Priority</b>	Top
<b>Architectural Part</b>	Software Development Paradigm

<b>ID</b>	SOFT.5
<b>Title</b>	Location transparency support
<b>Role</b>	Software Developer
<b>Description</b>	Applications deployed in a distributed environment: all interactions of software components have to use pure message passing and everything has to be asynchronous.
<b>Constraints</b>	
<b>Priority</b>	Top
<b>Architectural Part</b>	Software Development Paradigm

<b>ID</b>	SOFT.6
<b>Title</b>	Provision of non-blocking behavior and non-blocking guarantees
<b>Role</b>	Software Developer
<b>Description</b>	No programming thread is able to indefinitely delay others.
<b>Constraints</b>	
<b>Priority</b>	Top
<b>Architectural Part</b>	Software Development Paradigm

<b>ID</b>	SOFT.7
<b>Title</b>	Support supervision strategies and formulation of hierarchies
<b>Role</b>	Software Developer
<b>Description</b>	Supervision describes a dependency relationship between software entities: the supervisor delegates tasks to subordinates and therefore must respond to their failures. Supervision schemes in the form of clustering, dynamic assignment of roles and fault management/escalation schemes have to be supported. Clustering provides a fault-tolerant decentralized peer-to-peer based cluster membership service with no single point of failure or single point of bottleneck.
<b>Constraints</b>	No centralized control.
<b>Priority</b>	Top
<b>Architectural Part</b>	Software Development Paradigm

<b>ID</b>	SOFT.8
<b>Title</b>	Symmetric communication among the interconnected software components
<b>Role</b>	Software Developer
<b>Description</b>	Support of bidirectional communication between involved systems. There is no system that only accepts connections, and there is no system that only initiates connections.
<b>Constraints</b>	
<b>Priority</b>	Top
<b>Architectural Part</b>	Software Development Paradigm

<b>ID</b>	SOFT.9
<b>Title</b>	Representation of the application in the form of service chaining graph
<b>Role</b>	Software Developer, DevOps user
<b>Description</b>	Each application has to be developed in a way that it can be broken down into set of components/services interconnected among them (based on the definition of dependencies and workflow characteristics).
<b>Constraints</b>	Such an activity has to be realized prior to the interpretation of an application from the Smart Controller. In case of newly developed applications based on ARCADIA software development paradigm, such functionality has to be supported in an automated way.
<b>Priority</b>	Top
<b>Architectural Part</b>	Software Development Paradigm

## 5.2 Programmable Infrastructure Management Requirements

Management of the **programmable infrastructure** in ARCADIA has to be realized by the **Smart Controller**. Such management includes the capacity for **registration of resources** (in an IaaS and multi-IaaS environment), **allocation and de-allocation of resources** in a dynamic and effective manner as well as illustrating **QoS policies** in accessing resources. The objective is to hide all the

infrastructural configuration details from the application developer and facilitate him to develop **infrastructural agnostic** applications.

Placement of applications has to be realized in the vast majority of cases over virtual resources, while in some cases reservation of physical resources may be also required for the support of specific functionalities (e.g. in case of need to access to the energy-aware capabilities of a device). Allocation of further resources when required, release of non-used resources as well as migration/consolidation of the provided applications services over the existing resources constitute basic characteristics that have to be facilitated by the flexibility provided by the infrastructure.

*A break down to requirements regarding the programmable infrastructure follows.*

<b>ID</b>	INFRA.1
<b>Title</b>	Programmability of the infrastructure
<b>Role</b>	IaaS provider
<b>Description</b>	The computing, storage and networking infrastructure in ARCADIA should provide management capabilities well beyond the mere configuration of static protocols and paradigms. Programmability means a great flexibility in specifying what the infrastructure is expected to do. There are two complementary aspects in programmability: for developers, programmability is the mean to create the proper execution environment (provide preferences for the infrastructure that the code runs on, including both the virtual hardware – like servers, storage and network devices – and its configuration); for IaaS providers, programmability means a high degree of flexibility in resource management, enabling the implementation of complex and coordinated policies (like energy efficiency).
<b>Constraints</b>	Devices implement programmability paradigms and supports related protocols (e.g., OpenFlow).
<b>Priority</b>	Top
<b>Architectural Part</b>	Software development paradigm, Smart Controller, Programmable Resource Manager
<b>Notes</b>	The programmability of network infrastructures is an essential requirement for ARCADIA. Indeed, the entire framework relies on the possibility to directly control the network, in order to assure a dynamic, responsive and customized behavior, which satisfies the many needs from applications.

<b>ID</b>	INFRA.2
<b>Title</b>	Resources abstraction
<b>Role</b>	Smart Controller, IaaS provider
<b>Description</b>	ARCADIA implements a <i>Programmable Resource Abstraction</i> layer to simplify the operation of the Smart Controller. This abstraction layer should provide a consistent yet simplified view of the underlying communication infrastructure by means of standard information and data models, in order to set up the computational, storage and networking environment for every distributed application in a scalable, dependable, secure and effective way. One of the main concerns in this field is the level of granularity for the abstraction. On one hand, not all the details and characteristics of the resources are necessary to the Smart Controller. On the other hand, excessive abstraction prevents applications from utilizing resources to the maximum because it hides the details of the resources.
<b>Constraints</b>	
<b>Priority</b>	Top

<b>Architectural Part</b>	Smart Controller, Programmable Resource Manager
---------------------------	---

<b>ID</b>	INFRA.3
<b>Title</b>	Horizontal scalability
<b>Role</b>	Software developer, Smart Controller, IaaS provider
<b>Description</b>	The ARCADIA infrastructure should support horizontal scalability of applications, both for load balancing and redundancy, by providing specific functions to deliver packets to multiple application instances. Applications should be able to increase the number of instances and to specify whether they are meant for load balancing, for redundancy or both. Allocation of the required resources has to be realized in a reactive way based on the requests for horizontally scaling.
<b>Constraints</b>	Network devices support load balancing and multicast delivery.
<b>Priority</b>	Top
<b>Architectural Part</b>	Smart Controller, Programmable Resource Manager

<b>ID</b>	INFRA.4
<b>Title</b>	Heterogeneous networking
<b>Role</b>	Smart Controller
<b>Description</b>	The ARCADIA infrastructure should provide for seamless connectivity for application instances running in different administrative domains, by creating virtual topologies spanning heterogeneous infrastructures. Inter-domain connectivity must ensure consistent policies are applied even in presence of heterogeneous infrastructures, especially for what concerns Quality of Service and security (e.g., dynamic cross-network admission control).
<b>Constraints</b>	
<b>Priority</b>	Top
<b>Architectural Part</b>	Smart Controller, Programmable Resource Manager

<b>ID</b>	INFRA.5
<b>Title</b>	Virtualization of network elements
<b>Role</b>	IaaS provider
<b>Description</b>	Network elements are shared among multiple distributed applications. Each application configures network elements according to its own requirements. Hence, it is necessary that these networks and network elements are mutually isolated. This means that the programmable infrastructure slices network resources into multiple logical partitions, and implements suitable isolation mechanisms to avoid packets from jumping between them. Partitioning the network infrastructure into multiple virtual networks concerns both physical and software resources. In addition, applications may require network resources, e.g., bandwidth and packet processing, whereas the applications' requirements cannot be satisfied with a single network element. In that case, multiple network elements can be virtually combined to appear as a single resource able to satisfy the constraints.
<b>Constraints</b>	Networking devices must be logically split into multiple independent partitions and clustered together to appear as one single logical unit.

<b>Priority</b>	Top
<b>Architectural Part</b>	Smart Controller, Programmable Resource Manager

<b>ID</b>	INFRA.6
<b>Title</b>	Improved network programmability
<b>Role</b>	IaaS provider
<b>Description</b>	To better deal with specific needs from different applications, it would be desirable to extend on demand the functions of the resource layer in a programmable manner. This ability can dynamically add or remove additional functions for data transport and processing (such as packet caching, header/payload compression, regular expression matching, data transcoding, or even handling newly developed protocols), thereby avoiding addition or replacement of specific hardware.
<b>Constraints</b>	Programmability of the data plane.
<b>Priority</b>	Low
<b>Architectural Part</b>	Smart Controller, Programmable Resource Manager, Context Model.
<b>Notes</b>	Data plane programmability affects devices in the infrastructure; however, both the Smart Controller and the Context Model should be aware of this capability to enable developers to take full advantage of it.

<b>ID</b>	INFRA.7
<b>Title</b>	User security and privacy
<b>Role</b>	IaaS provider
<b>Description</b>	The ARCADIA framework should guarantee trustworthy networking by traffic encryption and anonymization. Encryption and anonymization are required in order to prevent sensitive data from eavesdropping, alteration, and all other passive and active security threats.
<b>Constraints</b>	Devices should have enough processing power to encrypt/decrypt large amount of data.
<b>Priority</b>	Top
<b>Architectural Part</b>	Smart Controller, Programmable Resource Manager

<b>ID</b>	INFRA.8
<b>Title</b>	Secure networking
<b>Role</b>	IaaS provider
<b>Description</b>	The ARCADIA networking infrastructure should take any relevant countermeasure to eliminate any security threat coming from weak architectural design, and erroneous or faulty configuration. Access to network resources by applications must be granted in order to protect the networking infrastructure and its components from a denial of service attack that may jeopardize the overall robustness, quality and reliability of the entire infrastructure. The affected resources may be easily and quickly isolated, malicious traffic may be safely terminated, sensitive flows can be identified and separately transferred in a more secure manner. In particular, ARCADIA should also take into account how networks are attached to hypervisors. If a network must be separated from other systems at all costs, it may be

	necessary to schedule instances for that network onto dedicated compute nodes. This may also be done to mitigate against exploiting a hypervisor breakout allowing the attacker access to networks from a compromised instance.
<b>Constraints</b>	Access control, Intrusion Prevention and Detection Systems, Denial-of-Service mitigation.
<b>Priority</b>	Medium
<b>Architectural Part</b>	Smart Controller, Programmable Resource Manager
<b>Notes</b>	This requirement addresses security constraints of the whole physical infrastructure owned by a specific IaaS provider.

<b>ID</b>	INFRA.9
<b>Title</b>	Security middleboxes
<b>Role</b>	IaaS provider, Software developer
<b>Description</b>	The network implements security functions like NAT (Network Address Translation), Firewall, NAC (Network Access Control), IBNS (Identity-Based Networking Services), mitigation of DoS (Deny of Service), etc. Applications are provided with suitable abstractions to configure and set up these services, which are run by the ARCADIA framework in each virtual network. These functions are needed to protect distributed applications against malicious attacks and unauthorized use.
<b>Constraints</b>	Physical or virtual resources must have the capability to be split into multiple isolated logical partitions.
<b>Priority</b>	Medium
<b>Architectural Part</b>	Smart Controller, Programmable Resource Manager, Context Model.
<b>Notes</b>	This requirement addresses security providers for applications and services deployed by users.

<b>ID</b>	INFRA.10
<b>Title</b>	Service chaining
<b>Role</b>	IaaS provider, Smart Controller, Software developer
<b>Description</b>	The ARCADIA infrastructure should enable applications dynamic service composition by specifying a chain of intermediate processing functions that packets should traverse. The chain may concern security features (e.g., the sequence of security middleboxes to be traversed by incoming and outgoing packets), or application-specific processing (e.g., multiple transcoding units to deliver the same video with different formats and codecs).
<b>Constraints</b>	
<b>Priority</b>	Low
<b>Architectural Part</b>	Smart Controller, Programmable Resource Manager, Context Model.

<b>ID</b>	INFRA.11
<b>Title</b>	Consistency
<b>Role</b>	IaaS provider



<b>Description</b>	The network applies configuration changes to physical and virtual resources in a synchronous, reliable and consistent way, in order to avoid undesirable effects like routing loops, packet losses and delays, violation of security policies, packet hopping between different virtual slices.
<b>Constraints</b>	
<b>Priority</b>	Top
<b>Architectural Part</b>	Smart Controller, Programmable Resource Manager

<b>ID</b>	INFRA.12
<b>Title</b>	Application mobility (scalability aspects)
<b>Role</b>	IaaS provider
<b>Description</b>	The ARCADIA framework is expected to dynamically adjust the execution environment to meet the application requirements at best, in order to accomplish the QoS requirements, to assure high-availability, to save energy. This entails the possibility of horizontally scaling the application out and in, moving instances among different nodes and cloud infrastructures, and suspending or delegating functions to save energy, depending on service status and end user demands. The network supports this function by providing dynamic, quick and consistent reconfiguration of the communication infrastructures, both in terms of packet forwarding and logical functions.
<b>Constraints</b>	The IaaS infrastructure must support seamless migration of virtual containers.
<b>Priority</b>	Medium
<b>Architectural Part</b>	Smart Controller, Programmable Resource Manager
<b>Notes</b>	The critical factor in this context is service continuity, which means to provide the services without service downtime regardless of whether the services are running in a single domain or are moved to another domain. The usage of resource partitioning techniques (like VLANs, IP subnets) often create a large context state in the system, which hinders service migration.

<b>ID</b>	INFRA.13
<b>Title</b>	Autonomic operation
<b>Role</b>	IaaS provider
<b>Description</b>	The usage of programmable infrastructures may aggravate the damage of security breaches, misconfiguration, privacy infringement and other incidents due to human errors. Properties that were traditionally implemented in hardware and impossible to change can now be modified, misconfigured or can function improperly. It is therefore necessary to enhance monitoring capability and automated operations.
<b>Constraints</b>	All physical and virtual resources implement suitable configuration and management protocols than enable machine-to-machine (M2M) interaction without human intervention.
<b>Priority</b>	Medium
<b>Architectural Part</b>	Smart Controller, Programmable Resource Manager

<b>ID</b>	INFRA.14
-----------	----------

<b>Title</b>	Power consumption modulation
<b>Role</b>	IaaS provider
<b>Description</b>	Energy efficiency is considered as one of the core objectives. To this goal, physical resources should implement power management features that allow them to modulate their power consumption (e.g., CPU voltage/frequency scaling), and to be partially or fully shut down (including low-power states such as standby and hibernation).
<b>Constraints</b>	
<b>Priority</b>	Medium
<b>Architectural Part</b>	Smart Controller, Programmable Resource Manager

<b>ID</b>	INFRA.15
<b>Title</b>	Efficient and effective availability
<b>Role</b>	IaaS provider, Smart Controller
<b>Description</b>	The ARCADIA framework should provide network service availability while balancing power consumption as energy efficiency is considered as one of the core objectives. High-availability of network services implies redundancy at both hardware and virtual level; in addition, proper configuration is needed to quickly and seamlessly resume operation in case of hardware/software failure. However, employing more resources than strictly needed entails larger power consumption. Higher availability must consider the provisioning of redundant resources at different geographical sites.
<b>Constraints</b>	
<b>Priority</b>	Medium
<b>Architectural Part</b>	Management of resources, orchestration.
<b>Notes</b>	Redundancy typically implies low resource utilization; for example, the Spanning Tree protocol for Ethernet networks often leave many unutilized links. The shift towards programmable infrastructure should provide better opportunities to balance resource utilization with redundancy and availability (see also the “Programmability” requirement).

<b>ID</b>	INFRA.16
<b>Title</b>	QoS capabilities
<b>Role</b>	IaaS provider
<b>Description</b>	Applications execution environment hosts (Hypervisors, Containers or native OS) provide QoS capabilities regarding access to compute (CPU, memory), storage (I/O) and network resources. Network equipment provides QoS capabilities, including priority queues, traffic shaping, access control, resource reservation.
<b>Constraints</b>	
<b>Priority</b>	Top
<b>Architectural Part</b>	Smart Controller, Programmable Resource Manager
<b>Notes</b>	QoS capabilities also entail the oversubscription ratio used in many datacenter, by using narrower links at the root of tree topologies.

<b>ID</b>	INFRA.17
<b>Title</b>	QoS management
<b>Role</b>	IaaS provider, Software developer, Context Model
<b>Description</b>	<p>The ARCADIA framework should enable to implement QoS policies according to the applications' requirements, in order to allow guaranteed service of workloads, near-real-time routing and information driven network prioritization. This includes reserving compute and storage resources, reserving bandwidth and memory buffers, labeling packets, setting priority and scheduling disciplines, shaping traffic according the predefined Service Level Agreement, managing Service Level Agreement among different administrative domains, service preemption.</p> <p>It is important that available service parameters are exchanged among different administrative domains for automating the control and/or management of the network services.</p>
<b>Constraints</b>	Programmable infrastructure components support QoS capabilities (see the "QoS capabilities" requirement)
<b>Priority</b>	
<b>Architectural Part</b>	Management of resources, software development paradigm, application profiling.
<b>Notes</b>	The availability of QoS capabilities in the IaaS infrastructure brings software developers the possibility of requiring the proper service level for their applications, but this feature should be available in the Context Model. Further, the specification of QoS capabilities enables to take this information into the application profile, in order to reserve the right amount of spare resources in the optimization process. On the resource management side, the placement of applications in the physical resources pool must consider any topological constraints (e.g., oversubscription ratio in the communication tree).

<b>ID</b>	INFRA.18
<b>Title</b>	Network performance
<b>Role</b>	IaaS provider
<b>Description</b>	The network operates in a reliable and effective way, in order to assure the availability of the communication infrastructure. Network control and management should not introduce any penalties for applications. This means that, in addition to the proper number and size of network devices, control protocols and logic must guarantee very short response time, must support high-volumes of data, must not become the bottleneck of the system, and must provide quick resilience to failure.
<b>Constraints</b>	
<b>Priority</b>	Medium
<b>Architectural Part</b>	Management of resources.
<b>Notes</b>	For example, for an SDN infrastructure, the proper number of controllers should be deployed, and proactive control should be preferred over the reactive one.

<b>ID</b>	INFRA.19
<b>Title</b>	Unified abstraction of resources and capabilities
<b>Role</b>	IaaS provider

<b>Description</b>	It is important that a common resource abstraction model be applied to similar network resources regardless of the underlying technology. Effective and energy efficient management of network devices require the appropriate abstraction of various capabilities, in terms of functional states and operating conditions. These should include at list the different power states available (including power modulation and low-power states), the power consumption and the QoS constraints for each state (delay, throughput, jitter, packet loss). In addition, a standard interface is used to gather data from all network equipment.
<b>Constraints</b>	Programmable resources support power consumption modulation features (see the "Power consumption modulation" requirement).
<b>Priority</b>	Top
<b>Architectural Part</b>	Management of resources.
<b>Notes</b>	The usage of the Green Abstraction Layer will be considered for (at least) networking devices.

### 5.3 Distributed Applications Deployment and Orchestration Requirements

As stated in Section 2.3, in ARCADIA we are going to support applications developed based on the ARCADIA software development paradigm, as well as existing (legacy) and hybrid applications. Deployment of such applications has to be based on the creation (in an automated or manual manner) of a **deployment script** that is going to be handled by the Smart Controller for the application's deployment phase.

The deployment script has to take into account the denoted **service-chaining scheme** of the applications components as well as the imposed **constraints**. The deployment script has to be provided based on a widely used scripting language that can be interpreted later on by the Smart Controller. The script has to include **a set of hooks**, used in order to support actions –such as install software, start/stop a service, manage relationships among components-, as well as real-time monitoring hooks and hooks related with concepts denoted in the ARCADIA context model.

Upon the creation of the deployment script, the Smart Controller is responsible for **instantiating** the appropriate components, **placing** them over the available infrastructure and **validating** the appropriate deployment of the application. In order to provide such functionalities, **registration of the programmable resources** has to be supported, as well as a set of **real time monitoring** and **policies management** functionalities.

Placement has to take into account the software level annotations denoted by the applications developers, the active policies defined by the Services Providers as well as the available resources in the considered multi-PoPs infrastructure.

Instantiation of the application components has to be done on a distributed way based on the principles of the considered software development paradigm. Given that each application is broken down in a set of components and their dependencies, instantiation of components has to be realized in an ad-hoc manner. Towards this direction, the existence of a **components repository** based on the active services in the considered infrastructure can be proven helpful. **De-provision** of the reserved resources upon the end of the execution time of each application has also to be supported.

With regards to the **orchestration** of the provided applications in real-time, this is going to be based on on-the-fly configuration taking into account real-time monitoring probes as well as the existing policies. A set of **real-time monitoring probes** has to be initiated per application component, based on the monitoring hooks provided within the corresponding software. Real-time as well as archived

information can be used for **decision making** purposes for **optimal execution/re-configuration** of an application. Prioritization of the existing policies as well as conflict resolution mechanisms has also to be supported.

*In the following the aforementioned requirements are further analyzed while next, applications profiling and optimization requirements are addressed in separate as important parts of the applications deployment and orchestration.*

<b>ID</b>	DEP.ORCH.1
<b>Title</b>	Creation of the deployment script
<b>Role</b>	Software Developer, DevOps user
<b>Description</b>	Based on the breakdown of each application into a set of components/services which are interconnected among them (based on the definition of dependencies and workflow characteristics) as well as the denoted constraints and monitoring hooks, an application's deployment script has to be created and feed the ARCADIA Smart Controller for deployment purposes.
<b>Constraints</b>	In ARCADIA based applications, this functionality has to be automated. In existing applications or hybrid application, the deployment script has to be prepared by a DevOps user. A scripting language has to be adopted for the description of the deployment scripts.
<b>Priority</b>	Top
<b>Architectural Part</b>	Software Development Paradigm, Smart Controller

<b>ID</b>	DEP.ORCH.2
<b>Title</b>	Registration of programmable resources
<b>Role</b>	Smart Controller, IaaS Provider
<b>Description</b>	The available compute, storage and network resources per Point of Presence (e.g. data center) have to be registered to the Smart Controller which advertises them and makes them accessible through a resource management API.
<b>Constraints</b>	Use/extension of open-source resource management software is highly desired (e.g. OpenStack).
<b>Priority</b>	Top
<b>Architectural Part</b>	Smart Controller, Programmable Resource Manager

<b>ID</b>	DEP.ORCH.3
<b>Title</b>	Registration and management of programmable resources in a Multi-IaaS environment
<b>Role</b>	Smart Controller, IaaS Provider
<b>Description</b>	Registration and management of resources through multiple PoPs has to be supported. Communication and interaction between various smart controllers each one being responsible for a single PoP, has to be supported while resources advertisement and synchronization issues among them to be addressed. Application's deployment to a multi-IaaS environment has to be realized.
<b>Constraints</b>	Establishment of network connectivity among the multiple IaaS infrastructure. Unified

	management functionalities per set of resources across the multiple IaaS infrastructure.
<b>Priority</b>	Top
<b>Architectural Part</b>	Smart Controller, Programmable Resource Manager

<b>ID</b>	DEP.ORCH.4
<b>Title</b>	Application's deployment based on the deployment script
<b>Role</b>	Smart Controller, DevOps User
<b>Description</b>	Given the existence of the application's deployment script, the Smart Controller is responsible for the placement of the application over the available infrastructure and the instantiation of the application's components.
<b>Constraints</b>	Capacity for interpretation of the deployment script. Specification of a notation that can be handled by the Smart Controller. Compatibility with existing deployment scripts.
<b>Priority</b>	Top
<b>Architectural Part</b>	Smart Controller

<b>ID</b>	DEP.ORCH.5
<b>Title</b>	Monitoring and validation of application's deployment process
<b>Role</b>	Smart Controller
<b>Description</b>	Need for monitoring of the deployment process and the validation of the proper instantiation of the required components, as denoted in the deployment script.
<b>Constraints</b>	Need for monitoring probes to the Smart Controller with regards to the status of each component. Need for existence of a component repository.
<b>Priority</b>	Top
<b>Architectural Part</b>	Smart Controller

<b>ID</b>	DEP.ORCH.6
<b>Title</b>	Resources reservation and configuration
<b>Role</b>	Smart Controller
<b>Description</b>	Need for reservation of the appropriate resources in the available programmable infrastructure for the deployment of the considered application. Deployment of the applications components in a virtualized (virtual machine, container) or physical environment.
<b>Constraints</b>	Use of resource management software and the corresponding API.
<b>Priority</b>	Top
<b>Architectural Part</b>	Smart Controller, Programmable Resource Manager

<b>ID</b>	DEP.ORCH.7
-----------	------------



<b>Title</b>	Monitoring hooks deployment and validation
<b>Role</b>	Smart Controller
<b>Description</b>	Based on the monitoring hooks denoted in the deployment script, a set of monitoring probes have to be established and maintained during the application's execution time. Such monitoring probes will be used for collection of information with regards to the status of each component, QoS metrics etc. and real-time decision making from the Smart Controller with regards to optimal deployment/configuration/re-configuration aspects.
<b>Constraints</b>	Access to monitoring mechanisms and data provided by the software.
<b>Priority</b>	Top
<b>Architectural Part</b>	Smart Controller

<b>ID</b>	DEP.ORCH.8
<b>Title</b>	Service locator/Component repository
<b>Role</b>	Smart Controller, ARCADIA Administrator
<b>Description</b>	A repository with the instantiated services/components have to be accessible to the Smart Controller for instantiation of similar services/components in a dynamic manner –upon the receipt of a deployment script. This repository has to support automated registration of components while it has also to be periodically maintained by the ARCADIA Administrator.
<b>Constraints</b>	Need for definition of the ARCADIA administrative domain where such a repository will be active (e.g. services provided over the resources registered to the Programmable Resources Manager of the Smart Controller)
<b>Priority</b>	Top
<b>Architectural Part</b>	Smart Controller

<b>ID</b>	DEP.ORCH.9
<b>Title</b>	Context model repository
<b>Role</b>	Smart Controller, Application Developer, Services Provider, IaaS Provider, Arcadia Administrator
<b>Description</b>	A context model repository has to be available, facilitating the extension/update of the concepts represented in the ARCADIA context model. Such concepts can be used by all the ARCADIA users depending on their preferences and perspective (e.g. for the description of policies on behalf of the Services Provider).
<b>Constraints</b>	Maintenance of the context model in a periodical time period.
<b>Priority</b>	Medium
<b>Architectural Part</b>	Smart Controller, Context Model

<b>ID</b>	DEP.ORCH.10
<b>Title</b>	Service graph repository
<b>Role</b>	Smart Controller, DevOps User

<b>Description</b>	A service graph repository with existing service graphs instantiated and deployed in the ARCADIA operational environment has to be available.
<b>Constraints</b>	Access and privacy issues have to be examined.
<b>Priority</b>	Medium
<b>Architectural Part</b>	Smart Controller

<b>ID</b>	DEP.ORCH.11
<b>Title</b>	Rapid introduction of new services
<b>Role</b>	Software Developer, DevOps User
<b>Description</b>	New services has to be easily designed and deployed over the available infrastructure by exploiting specific architectural components, such as the service locator/component repository as well as the service graph repository.
<b>Constraints</b>	Integration of existing components/graphs in new deployment scripts.
<b>Priority</b>	Medium
<b>Architectural Part</b>	Smart Controller

<b>ID</b>	DEP.ORCH.12
<b>Title</b>	Applications operation QoS guarantees
<b>Role</b>	Smart Controller
<b>Description</b>	Establish a set of monitoring probes based on the existing monitoring hooks and metrics. Provide guarantees on the services provision based on the existing policies in the ARCADIA operational environment.
<b>Constraints</b>	Need for conflict resolution policies. Consider Let-it fail programming model.
<b>Priority</b>	Medium
<b>Architectural Part</b>	Smart Controller

<b>ID</b>	DEP.ORCH.13
<b>Title</b>	Autonomic management of applications
<b>Role</b>	Smart Controller
<b>Description</b>	Re-configure applications based on conditions and objectives/constraints (e.g. support vertical and horizontal auto-scaling characteristics). A set of distributed intelligence mechanisms have to be designed and developed in order to support such functionalities.
<b>Constraints</b>	
<b>Priority</b>	Medium
<b>Architectural Part</b>	Smart Controller

<b>ID</b>	DEP.ORCH.14
<b>Title</b>	De-provisioning of reserved resources
<b>Role</b>	Smart Controller
<b>Description</b>	Upon the end of the running time of the application, the allocated resources have to be released.
<b>Constraints</b>	Existence of API for resource management.
<b>Priority</b>	Medium
<b>Architectural Part</b>	Smart Controller, Programmable Resources Manager

### 5.3.1 Distributed Applications Profiling and Optimization Requirements

One of the main functionalities of the ARCADIA Smart Controller regards the design and deployment of an **optimization framework** that has to support the **optimal placement and execution** of applications and services over the available infrastructure. The optimization framework has to be able to handle **multiple objectives** denoted by different users and provide configuration suggestions taking into account the existing policies in the ARCADIA operational environment along with their hierarchy. Optimization aspects regard **objectives of the services provider**, such as the need for low energy consumption, the need for the provision of QoS guarantees and meeting provided Service Level Agreements (SLAs), cost related or infrastructure-usage related objectives, as well as **objectives of the application**, such as performance related or resources usage and cost related objectives. These objectives have to be described by using a specific notation and feed the Smart Controller in order to be able to proceed to decision making based on them.

The achievement of the objectives has to be clearly stated through the use of a set of metrics. Monitoring of the current status of a set of service level objectives has to be realized based on the monitoring and management mechanisms of the Smart Controller and the monitoring hooks and preferences (in the form of annotations) provided by the software developer as well as the DevOps user. The denoted constraints in various levels have also to be taken into account and be part of the optimal solving problem.

Optimization has to be pursued during the entire applications' management lifecycle, starting from their initial deployment over the programmable infrastructure until the finalization of their execution and the de-provision of the allocated resources. Thus, the optimization framework has to be able to provide suggestions in short time (taking into account time-constraints of each application) and support effective communication paths with the monitoring and policies management components of the ARCADIA Smart Controller. Given that optimization objectives usually refer to bundle of services, the Smart Controller has to be able to interplay with the provision of the individual applications focusing on the fulfillment of the overall objectives.

In addition to real-time solving of optimization problems, the capacity to be able to support **forecasting** regarding the status of the ARCADIA operational environment in the future based on the provided applications/services at each point of time is also considered very helpful. This can be achieved through **application profiling** in terms of the application needs with regards to the usage of resources, horizontal and vertical scaling characteristics as well as sensitivity to specific metrics (e.g. low jitter in case of real-time video streaming functionalities). For instance, information regarding the computational, storage and network traffic creation intensity of an application can provide valuable information to the deployment component of the Smart Controller. Based on the status of the available

resources, the Smart Controller has to decide whether a new deployment can be supported without violating any of the existing constraints. **Prediction** for the necessity to use specific resources in the upcoming time periods may be also realized leading to optimal allocation of resources and new requests admission control. **Identification of misbehavior** of specific applications may be also identified based on their profiling. The Smart Controller has to be also able to react pro-actively in order to adapt to infrastructure and application state changes and select always the optimal configuration. For instance, **consolidation** of envisaged workloads may be realized causing little or no impact to application performance.

*Below, the aforementioned requirements are further analyzed.*

<b>ID</b>	PROF.OPT.1
<b>Title</b>	Optimal placement of applications
<b>Role</b>	Smart Controller, Services Provider, Software Developer
<b>Description</b>	The placement of applications over the available programmable infrastructure has to be realized in an optimal manner with regards to the objectives denoted by the Services Provider and the Software Developer. High level objectives have to be represented in a set of lower level objectives that can be realized through appropriate handling of the operational status of the application's components.
<b>Constraints</b>	Objectives, preferences and constraints denoted at software level by the software developer. Policies imposed by the Services Provider.
<b>Priority</b>	Top
<b>Architectural Part</b>	Smart Controller, Optimization Framework

<b>ID</b>	PROF.OPT.2
<b>Title</b>	Description of high level policies in a custom format
<b>Role</b>	Services Provider
<b>Description</b>	The Services Provider should be able to describe high level policies based on its objectives (e.g. cost reduction policies, high quality of services policies, energy efficiency policies etc.). A custom format has to be used for the description of such policies. Furthermore, a mapping mechanism of high level policies to orchestration of the operational status of the underlying software components has to be available.
<b>Constraints</b>	Capacity to map high level policies to lower level configuration.
<b>Priority</b>	Top
<b>Architectural Part</b>	Smart Controller

<b>ID</b>	PROF.OPT.3
<b>Title</b>	Multi-objectives handling - Conflict Resolution
<b>Role</b>	Services Provider, Software Developer, Smart Controller
<b>Description</b>	Given that in many cases, multi-criteria optimization may be required, there is a need for defining priorities over the denoted service level objectives as well as specific conflict resolution policies. The optimal solution with regards to the multi-criteria objectives has to be

	implemented taking into account the definition of specific utility functions per objective.
<b>Constraints</b>	Support of distributed conflict resolution policies. Definition of hierarchy schemes for the management of conflicts.
<b>Priority</b>	Top
<b>Architectural Part</b>	Smart Controller

<b>ID</b>	PROF.OPT.4
<b>Title</b>	Applications execution real-time adaptation
<b>Role</b>	Smart Controller, Services Provider, Software Developer
<b>Description</b>	Upon the initial placement of the applications, real-time adaptation and re-configuration of the execution part has to be supported, based on the current status of the ARCADIA operational environments. Re-configuration can be triggered towards the fulfillment of the optimization objectives and may regard horizontal or vertical scaling aspects, adaptation of the quality of the provided services (e.g. lower quality ratio in a video on demand service) etc.
<b>Constraints</b>	Constraints and preferences denoted by the software developer.
<b>Priority</b>	Top
<b>Architectural Part</b>	Smart Controller

<b>ID</b>	PROF.OPT.5
<b>Title</b>	Event Triggers based on real-time monitoring probes
<b>Role</b>	Smart Controller, Software Developer, DevOps User
<b>Description</b>	Based on the monitoring hooks denoted in the applications software components as well as the existing policies, a set of monitoring probes have to be established. Real time monitoring information has to be collected and processed. Based on the available information, a set of re-configuration actions may be triggered.
<b>Constraints</b>	Capacity to establish real-time monitoring probes among distributed components.
<b>Priority</b>	Top
<b>Architectural Part</b>	Smart Controller

<b>ID</b>	PROF.OPT.6
<b>Title</b>	Applications profiling support
<b>Role</b>	Software Developer, Smart Controller
<b>Description</b>	It is desirable that the developed application components are accompanied with profiling information in terms of needs for computing, memory, storage and networking resources as well as with peculiarities such as data locality, data volatility, time criticality etc. Such profiling can be available prior to the placement of an application through annotations or through processing of historical data collected from previous executions of the specific components. The overall profiling can then be used for optimal placement as well as for forecasting purposes.
<b>Constraints</b>	Awareness with regards to the behavior of the application.

<b>Priority</b>	Medium
<b>Architectural Part</b>	Smart Controller

<b>ID</b>	PROF.OPT.7
<b>Title</b>	Decision making based on applications profiling information and forecasting mechanisms
<b>Role</b>	Smart Controller
<b>Description</b>	Given the existence of profiling information per application, admission control for the deployment of new applications can be realized taking into account the existing conditions in the ARCADIA operational environment and the predicted impact of the deployment of a new application. Furthermore, forecasting regarding performance oriented issues in the ARCADIA operational environment as well as regarding applications future demands can be realized.
<b>Constraints</b>	
<b>Priority</b>	Medium
<b>Architectural Part</b>	Smart Controller

<b>ID</b>	PROF.OPT.8
<b>Title</b>	Support of distributed intelligence mechanisms
<b>Role</b>	Smart Controller
<b>Description</b>	Distributed intelligence mechanisms have to be supported by the Smart Controller in some cases for being able to react in real-time in changes in the ARCADIA operational environment. Decision making with regards to optimization issues may be realized in a local or network wide perspective, taking into account the peculiarities of each software component.
<b>Constraints</b>	Time constraints in decision making process.
<b>Priority</b>	Medium
<b>Architectural Part</b>	Smart Controller

<b>ID</b>	PROF.OPT.9
<b>Title</b>	Real-time migration/consolidation mechanisms
<b>Role</b>	Smart Controller
<b>Description</b>	Based on the deployed applications at each point of time and the current workload and resource usage conditions, a set of migration/consolidation mechanisms has to be supported for optimal use of the available resources without negative impact on the overall performance.
<b>Constraints</b>	
<b>Priority</b>	Medium
<b>Architectural Part</b>	Smart Controller



## 6 Conclusions

---

The requirements specification plays an important role for the project, since it will provide the input to define the ARCADIA Context Model and framework. The requirements definition phase has involved contribution by all partners, and has followed a logical process including the identification of the main roles, the definition of the ARCADIA vision and context, and the analysis of the current state of the art.

Current best practice recommends a close cooperation between software developers and operations staff (DevOps philosophy); however, the use cases have clearly pointed out that more automation is necessary and new programming paradigms have to be adopted in order to make the deployment process repeatable, resilient, error-prone and adaptable to the actual execution environment. The analysis of the state of the art has revealed that programmable infrastructure is already available both for computing and networking; however, the deployment process still relies on scripts and other artifacts that require strong human intervention.

According to the ARCADIA objectives and use cases, a rich set of requirements has been derived, which cover for system, functional and compliance aspects. The use cases have already been shared with Task 2.3, in order to extend the current set with more specific requirements on the Smart Controller. The requirements are now available to Task 2.2 and Task 2.4 to carry out the definition of the ARCADIA Context Model and framework.

In order to facilitate the identification and selection of requirements for each group of activities, we have already clustered them into five categories: i) distributed software development paradigm (requirements concerning the definition of the Context Model and its usage by software developers during implementation); ii) programmable infrastructure management (requirements about features to be available in the virtualized infrastructure); iii) distributed application profiling (requirements concerning the identification of applications' special and temporal characteristics, to be used for dynamic deployment and orchestration); iv) deployment and orchestration (requirements concerning the placement of software components among programmable infrastructure available in multiple domains); v) optimization (requirements concerning the optimal allocation of resources to meet policies by infrastructure providers and constraints/profiles by applications).

## Annex I: References

---

- [1] Wes Felter, Alexandre Ferreira, Ram Rajamony, Juan Rubio. **An Updated Performance Comparison of Virtual Machines and Linux Containers**. IBM Research Report, RC25482 (AUS1407-001) July 21, 2014.
- [2] **Difference between Hypervisor Virtualization and Container Virtualization**. URL: <http://www.slashroot.in/difference-between-hypervisor-virtualization-and-container-virtualization> , 21 Oct 2014.
- [3] **Red Hat Enterprise Linux 6, Resource Management Guide, Managing system resources on Red Hat Enterprise Linux 6, Edition 6**. URL: [https://access.redhat.com/documentation/en-US/Red\\_Hat\\_Enterprise\\_Linux/6/html/Resource\\_Management\\_Guide/index.html](https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Resource_Management_Guide/index.html)
- [4] **Resource management in Docker**, URL: <https://goldmann.pl/blog/2014/09/11/resource-management-in-docker/>
- [5] **Operating System Containers vs. Application Containers**, URL: <http://blog.risingstack.com/operating-system-containers-vs-application-containers/>
- [6] **Kernel Virtual Machine**, URL: <http://www.linux-kvm.org/>
- [7] **The XEN project**, URL: <http://www.xenproject.org/>
- [8] **VMware Vsphere**, URL: <http://www.vmware.com/products/esxi-and-esx/overview>
- [9] **Microsoft HyperV**, URL: <http://www.microsoft.com/en-us/server-cloud/solutions/virtualization.aspx>
- [10] **Virtualization Matrix - Virtualization comparison**, URL: [http://www.virtualizationmatrix.com/matrix.php?category\\_search=all&free\\_based=1](http://www.virtualizationmatrix.com/matrix.php?category_search=all&free_based=1)
- [11] Rami Rosen. **Linux Containers and the Future Cloud**, URL: [http://media.wix.com/ugd/295986\\_d5059f95a78e451db5de3d54f711e45d.pdf](http://media.wix.com/ugd/295986_d5059f95a78e451db5de3d54f711e45d.pdf)
- [12] **LXC Containers**, URL: <http://linuxcontainers.org/>
- [13] **Docker**, URL: <https://www.docker.io/>
- [14] **Namespaces**, URL: <http://man7.org/linux/man-pages/man7/namespaces.7.html>
- [15] **Cgroups**, URL: <https://en.wikipedia.org/wiki/Cgroups>
- [16] Bobby Banerjee. **Understanding the key differences between LXC and Docker**, URL: <https://www.flockport.com/lxc-vs-docker/>
- [17] **Microsoft Unveils New Container Technologies for the Next Generation Cloud**, URL: <http://azure.microsoft.com/blog/2015/04/08/microsoft-unveils-new-container-technologies-for-the-next-generation-cloud/>
- [18] **Canonical MaaS**, URL: <https://maas.ubuntu.com/>
- [19] **Openstack Ironic**, URL: <http://docs.openstack.org/developer/ironic/deploy/user-guide.html>
- [20] Nick Feamster, Jennifer Rexford, Ellen Zegura. **The Road to SDN: An Intellectual History of Programmable Networks**. ACM SIGCOMM Computer Communication Review, Volume 44 Issue 2, Pages 87-98, April 2014.
- [21] **Cisco Logical Routers**, URL:

[http://www.cisco.com/en/US/docs/ios\\_xr\\_sw/iosxr\\_r3.2/interfaces/command/reference/hr32lr.html](http://www.cisco.com/en/US/docs/ios_xr_sw/iosxr_r3.2/interfaces/command/reference/hr32lr.html)

- [22] **Juniper Logical Routers**, URL: <http://www.juniper.net/techpubs/software/junos/junos85/feature-guide-85/id-11139212.html>
- [23] **Open Networking Foundation**, URL: <https://www.opennetworking.org>
- [24] Eric Keller, Soudeh Ghorbani, Matt Caesar, Jennifer Rexford. **Live Migration of an Entire Network (and its Hosts)**. 11th ACM Workshop on Hot Topics in Networks (HotNets-XI), 2012.
- [25] Soudeh Ghorbani, Cole Schlesinger, Matthew Monaco, Eric Keller, Matthew Caesar, Jennifer Rexford, and David Walker. **Transparent, Live Migration of a Software-Defined Network**. ACM Symposium on Cloud Computing (SOCC '14), 2014.
- [26] ETSI. **Network Function Virtualization**, URL: <http://www.etsi.org/technologies-clusters/technologies/nfv>
- [27] Prayson Pate. **NFV and SDN: What's the Difference?**, URL: <https://www.sdxcentral.com/articles/contributed/nfv-and-sdn-whats-the-difference/2013/03/>
- [28] **Packet Processing on Intel® Architecture**, URL: <http://www.intel.com/content/www/us/en/intelligent-systems/intel-technology/packet-processing-is-enhanced-with-software-from-intel-dpdk.html>
- [29] **Network Functions Virtualisation — fit for purpose?**, URL: [http://www.microwave-eetimes.com/en/network-functions-virtualisation-fit-for-purpose-63.html?cmp\\_id=71&news\\_id=222904860&page=2](http://www.microwave-eetimes.com/en/network-functions-virtualisation-fit-for-purpose-63.html?cmp_id=71&news_id=222904860&page=2)
- [30] **Juniper vMX**, URL: <http://www.juniper.net/us/en/products-services/routing/mx-series/vmx/>
- [31] **Openvswitch**, URL: <http://openvswitch.org/>
- [32] **POX**, URL: <http://www.noxrepo.org/pox/about-pox/>
- [33] **IRIS**, URL: <http://openiris.etri.re.kr/>
- [34] **Floodlight**, URL: <http://www.projectfloodlight.org/floodlight/>
- [35] **Openstack**, URL: <http://www.openstack.org/>
- [36] **Openstack components**, URL: <https://en.wikipedia.org/wiki/OpenStack>
- [37] **Apache Cloudstack**, URL: <https://cloudstack.apache.org/about.html>
- [38] **Opendaylight**, URL: <http://www.opendaylight.org>
- [39] **OpenContrail**, URL: <http://www.opencontrail.org>
- [40] **Open Networking Foundation**, URL: <https://www.opennetworking.org>
- [41] **Open Platform for NFV (OPNFV)**, URL: <https://www.opnfv.org/>
- [42] **Cloudify**, URL: <http://getcloudify.org>
- [43] **TOSCA**, URL: [https://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=tosca](https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca)
- [44] **Pantou: Openflow 1.0 for openwrt**, URL: <http://www.openflow.org/wk/index.php>
- [45] **ofsoftswitch13 - cpqd**, URL: <https://github.com/CPqD/ofsoftswitch13>
- [46] **Indigo: Open source openflow switches**, URL: <http://www.openflowhub.org/display/Indigo/>
- [47] **OpenFaucet**, URL: <http://rlenglet.github.io/openfaucet/index.html/>
- [48] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. **Nox: towards an operating system for networks**. ACM SIGCOMM Computer Communication Review, 38(3):105–110, 2008.
- [49] **Mul**, URL: <http://sourceforge.net/p/mul/wiki/Home/>
- [50] Z. Cai, AL Cox, and TSE Ng. **Maestro: A system for scalable OpenFlow control**. Technical Report TR10-08, Rice University, December 2010.
- [51] **Trema openflow controller framework**, URL: <https://github.com/trema/trema>
- [52] **Beacon**, URL: <https://openflow.stanford.edu/display/Beacon/Home>

- [53] **Jaxon:java-based openflow controller**, URL: <http://jaxon.onuos.org/>
- [54] **Helios by NEC**, URL: <http://www.nec.com/>
- [55] **Simple Network Access Control (SNAC)**, URL: <http://www.openflow.org/wp/snac/>
- [56] **Ryu**, URL: <http://osrg.github.com/ryu/>
- [57] **OESS**, URL: <https://code.google.com/p/nddi/>
- [58] **The nodeflow openflow controller**, URL: <http://garyberger.net/?p=537>
- [59] R. Sherwood, M. Chan, A. Covington, G. Gibb, M. Flajslik, N. Handigol, T.Y. Huang, P. Kazemian, M. Kobayashi, J. Naous, et al. **Carving research slices out of your production networks with openflow**. ACM SIGCOMM Computer Communication Review, 40(1):129–130, 2010.
- [60] Marcelo R. Nascimento, Christian E. Rothenberg, Marcos R. Salvador, Carlos N. A. Correa, Sidney C. de Lucena, and Maurício F. Magalhães. **Virtual routers as a service: the routeflow approach leveraging software-defined networks**. In Proceedings of the 6th International Conference on Future Internet Technologies, CFI '11, pages 34–37, New York, NY, USA, 2011, ACM.
- [61] **OpenNebula**, URL: <http://opennebula.org/>
- [62] **Conduct R**, URL: <http://www.typesafe.com/products/conductr>
- [63] Timothy Wood, Ludmila Cherkasova, Kivanc Ozonat, and Prashant Shenoy. **Profiling and Modeling Resource Usage of Virtualized Applications**. Middleware 2008, 9<sup>th</sup> ACM / IFIP / USENIX International Conference on Middleware (2008).
- [64] Jinho Hwang, Sai Zeng, Frederick y Wu, Timothy Wood. **A Component-Based Performance Comparison of Four Hypervisors**. IEEE IM 2013.
- [65] J.F. Perez, G. Casale, S. Pacheco-Sanchez. **Estimating Computational Requirements in Multi-Threaded Applications**. IEEE Transactions on Software Engineering, December 2014, Volume 41, Issue 3, pp. 264-278.
- [66] Tania Lorido-Botran, Jose Miguel-Alonso, Jose A. Lozano. **A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments**. Journal of Grid Computing, December 2014, Volume 12, Issue 4, pp. 559-592.
- [67] Rafael Weingärtner, Gabriel Beims Bräscher , Carlos Becker Westphall. **Cloud resource management: A survey on forecasting and profiling models**. Journal of Network and Computer Applications, Volume 47, January 2015, pp. 99–106.
- [68] R Han, Moustafa Ghanem, and Yike Guo. **Elastic-TOSCA: Supporting Elasticity of Cloud Application in TOSCA**. CLOUD COMPUTING 2013, The Fourth International Conference on Cloud Computing (2013).
- [69] Gang Ren, Eric Tune, Tipp Moseley, Yixin Shi, Silvius Rus, Robert Hundt. **Google-Wide Profiling: A Continuous Profiling Infrastructure for Data Centers**. IEEE Micro (2010), pp. 65-79.
- [70] Kuai Xu, Feng Wang, Lin Gu. **Profiling-as-a-Service in Multi-Tenant Cloud Computing Environments**. The International Workshop on Security and Privacy in Cloud Computing, Macau, China, June 18-21 2012.
- [71] Anh Vu Do, Junliang Chen, Chen Wang, Young Choon Lee, Albert Y. Zomaya, and Bing Bing Zhou. **Profiling Applications for Virtual Machine Placement in Clouds**. 2011 IEEE 4th International Conference on Cloud Computing.
- [72] Nilabja Roy, Abhishek Dubey and Aniruddha Gokhale. **Efficient Autoscaling in the Cloud using Predictive Models for Workload Forecasting**. 2011 IEEE 4th International Conference on Cloud Computing.
- [73] Andreas Fischer, Juan Felipe Botero, Michael Till Beck, Hermann de Meer, Xavier Hesselbach. **Virtual Network Embedding: A Survey**. Communications Surveys & Tutorials, IEEE (Volume: 15, Issue: 4), Feb. 2013.

- [74] N. M. Mosharaf Kabir Chowdhury, Raouf Boutaba. **Network Virtualization: State of the Art and Research Challenges**. IEEE Communications Magazine, Jul. 2009.
- [75] N. M. Mosharaf Kabir Chowdhury, Muntasir Raihan Rahman, Raouf Boutaba. **Virtual Network Embedding with Coordinated Node and Link Mapping**. In Proceedings of the 28th Conference on Computer Communications (IEEE INFOCOM), Rio de Janeiro, Brazil, Apr. 2009.
- [76] Albrecht, J., Oppenheimer, D., Vahdat, A., and Patterson. **Design and implementation trade-offs for wide-area resource discovery**. ACM Transactions on Internet Technology Vol. 8, No. 4 (Sep. 2008), pp 1-44.
- [77] Y. Zhu and M. Ammar. **Algorithms for assigning substrate network resources to virtual network components**. In Proceedings of IEEE INFOCOM, 2006.
- [78] M. Yu, Y. Yi, J. Rexford, and M. Chiang. **Rethinking virtual network embedding: Substrate support for path splitting and migration**. ACM SIGCOMM Computer Communication Review, vol. 38, no. 2, pp. 17– 29, April 2008.
- [79] D. Andersen. **Theoretical approaches to node assignment**. Unpublished Manuscript, <http://www.cs.cmu.edu/~dga/papers/andersen-assign.ps>, 2002.
- [80] A. Gupta, J. M. Kleinberg, A. Kumar, R. Rastogi, and B. Yener. **Provisioning a virtual private network: A network design problem for multicommodity flow**. In Proceedings of ACM STOC, 2001, pp. 389–398.
- [81] R. Ricci, C. Alfeld, and J. Lepreau. **A solver for the network testbed mapping problem**. ACM SIGCOMM Computer Communication Review, vol. 33, no. 2, pp. 65–81, April 2003.
- [82] Mohamed Faten Zhani, Qi Zhang, Gwendal Simon and Raouf Boutaba. **VDC Planner: Dynamic Migration-Aware Virtual Data Center Embedding for Clouds**. IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), May 2013.
- [83] Qi Zhang, Mohamed Faten Zhani, Maissa Jabri, Raouf Boutaba. **Venice: Reliable Virtual Data Center Embedding in Clouds**. IEEE INFOCOM 2014.
- [84] Ahmed Amokrane, Mohamed Faten Zhani, Rami Langar, Raouf Boutaba and Guy Pujolle. **Greenhead: Virtual Data Center Embedding Across Distributed Infrastructures**. IEEE Transactions on Cloud Computing, Vol.1, Issue 1, Sep. 2013.
- [85] Li Erran Li, Vahid Liaghat, Hongze Zhao, MohammadTaghi Hajiaghayi, Dan Li, Gordon Wilfong, Y. Richard Yang and Chuanxiong Guo. **PACE: Policy-Aware Application Cloud Embedding**. IEEE INFOCOM 2013.
- [86] **Toward a Strategic Agenda for Software Technologies in Europe**, Information Society Technologies Advisory Group (ISTAG), July 2012, Available Online: <http://cordis.europa.eu/fp7/ict/docs/istag-soft-tech-wgreport2012.pdf>
- [87] Guido Salvaneschi, Alessandro Margara, Giordano Tamburrelli. **Reactive Programming: a Walkthrough**. ICSE 2015.
- [88] L. A. Meyerovich, A. Guha, J. Baskin, G. H. Cooper, M. Greenberg, A. Bromfield, and S. Krishnamurthi. **Flapjax: a programming language for Ajax applications**. ser. OOPSLA '09. New York, NY, USA: ACM, 2009, pp. 1–20.
- [89] C. Elliott and P. Hudak. **Functional reactive animation**. ser. ICFP '97. New York, NY, USA: ACM, 1997, pp. 263–273.
- [90] G. H. Cooper and S. Krishnamurthi. **Embedding dynamic dataflow in a call-by-value language**. ser. ESOP'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 294–308.
- [91] G. Salvaneschi, G. Hintz, and M. Mezini. **Rescala: Bridging between object-oriented and functional style in reactive applications**. ser. MODULARITY '14. New York, NY, USA: ACM, 2014, pp. 25–36.

- [92] I. Maier and M. Odersky. **Higher-order reactive programming with incremental lists**. ser. ECOOP'13. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 707–731.
- [93] J. Liberty and P. Betts. **Programming Reactive Extensions and LINQ**. 1st ed. Berkely, CA, USA: Apress, 2011.
- [94] E. Bainomugisha, A. L. Carreton, T. Van Cutsem, S. Mostinckx, and W. De Meuter. **A survey on reactive programming**. ACM Computing Surveys, 2012.
- [95] Cooper, G. H. and Krishnamurthi, S. 2006. **Embedding dynamic dataflow in a call-by-value language**. In ESOP'06: Proceedings of the 15th European conference on Programming Languages and Systems. Springer-Verlag, Berlin, Heidelberg, pp. 294–308.
- [96] Cooper, G. H. 2008. **Integrating dataflow evaluation into a practical higher-order call-by-value language**. Ph.D. thesis, Providence, RI, USA.
- [97] akka, <http://akka.io/>
- [98] akka, java documentation, <http://doc.akka.io/docs/akka/2.3.12/java.html>
- [99] Quasar, <http://www.paralleluniverse.co/quasar/>
- [100] Quasar, documentation, <http://docs.paralleluniverse.co/quasar/>