Complete Dynamic Multi-cloud Application Management

Project no. 644925

Innovation Action

Co-funded by the Horizon 2020 Framework Programme of the European Union

Call identifier: H2020-ICT-2014-1

Topic: ICT-07-2014 – Advanced Cloud Infrastructures and Services

Start date of project: January 1st, 2015 (36 months duration)

Deliverable D6.1

Complex Application Description Specification

Due date:	30/06/2015
Submission date:	20/07/2015
Deliverable leader:	SixSq
Editors list:	C. Loomis (SixSq)

Dissemination Level

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

List of Contributors

Participant	Short Name	Contributor
Interoute S.P.A.	IRT	Domenico Gallico, Matteo Biancani
SixSq Sàrl	SIXSQ	C. Loomis
QSC AG	QSC	
Technische Universitaet Berlin	TUB	M. Slawik
Fundacio Privada I2CAT, Internet I Innovacio Digital A Catalunya	I2CAT	Isart Canyameres, , José Aznar
Universiteit Van Amsterdam	UVA	Y. Demchenko
Centre National De La Recherche Scientifique	CNRS	

Change history

Version	Date	Partners	Description/Comments
0.5	15/07/2015	SixSq	First complete draft of document.
1.0-RC	16/07/2015	SixSq	Respond to comments. Final release.
1.0	17/07/15	IRT	Final review and fixes before submission
1.0	20/07/2015	IRT	Fixed issue on page 13 about Slipstream model assessment

Table of Contents

List of Tables

Executive Summary

CYCLONE facilitates the management of the full lifecycle of multi-cloud applications by providing complementary tools that support the configuration of the cloud application, its automated deployment onto cloud infrastructures, and control of the allocated cloud resources.

Using a single, complete application description ensures a consistent view of the application's requirements across the CYCLONE tools and streamlines the interactions between them. Given that CYCLONE will manage the full lifecycle of an application, its description should contain information relevant for all lifecycle stages:

1. **Analysis**: Identifying appropriate cloud infrastructures for placing the application's components.
2. **Deployment**: Provisioning and configuration of cloud resources to start the application.
3. **Optimization**: Monitoring of the application and adapting its resource allocation or distribution, to deal with changes in load, locations of clients, etc.
4. **Termination**: Shutting down the application and freeing its resources.

The deployment and termination of the application are minimal features that must be supported from the initial CYCLONE prototype. Features related to the placement and optimization can be phased in over the course of the project, provided that their support does not overly complicate the application model and description.

The initial set of 29 requirements (see Table 1) was collected through discussions with the CYCLONE partners, representing both the cloud application developers (through the defined use cases) and the developers of the CYCLONE components.

Five existing relevant application models and APIs – SlipStream, CIMI, TOSCA, OCCI, and Docker Compose – were evaluated against these 29 requirements. All (except OCCI) have similar coverage of the requirements and would need similar extensions. OCCI was eliminated because it lacks a practical serialization format and a native mechanism for describing complex applications.

To allow the project to advance as quickly as possible, a practical strategy has been adopted:

1. The project will start with the SlipStream model and description to allow the CYCLONE use cases to be ported to and deployed via CYCLONE immediately.
2. SlipStream will migrate incrementally to the CIMI model. The ported CYCLONE applications will follow this incremental migration.
3. CIMI will be extended to allow CYCLONE to handle the advanced placement and metrics-based scaling foreseen within the project's work plan.
4. A further evaluation of TOSCA will be done to see if using its bundled serialization format or other features brings any additional benefits over those of the extended CIMI model.

This strategy will allow the project to start porting and deploying the pilot applications immediately while providing a solid, standardized platform to support the advanced features to be treated later in the project.

1. Introduction

CYCLONE facilitates the management of the full lifecycle of multi-cloud applications by providing complementary tools that support the configuration of the cloud application, its automated deployment onto cloud infrastructures, and control of the allocated cloud resources. To manage the cloud application effectively, these tools, and particularly the deployment engine SlipStream, must understand the application in detail. Using a single, complete application description ensures a consistent view of the application's requirements across the CYCLONE tools and streamlines the interactions between them.

The objectives of this document are to:

1. Enumerate the requirements for a common application description based on the CYCLONE use cases and from the CYCLONE tool developers,
2. Identify relevant, existing application models, including both *ad hoc* models and standardized ones,
3. Evaluate those models against the stated requirements, and
4. Determine the application model that will be used within CYCLONE.

Following this introduction, the document addresses each of the above objectives in order. The document concludes by describing the strategy that CYCLONE has adopted regarding the cloud application model.

2. Identified Requirements

2.1. Scope

Given that CYCLONE will manage the full lifecycle of an application, the application description should contain information relevant for all lifecycle stages:

5. **Analysis**: Identifying appropriate cloud infrastructures for placing the application's components.
6. **Deployment**: Provisioning and configuration of cloud resources to start the application.
7. **Optimization**: Monitoring of the application and adapting its resource allocation or distribution, to deal with changes in load, locations of clients, etc.
8. **Termination**: Shutting down the application and freeing its resources.

Deployment and termination of applications are minimal mandatory features that must be supported from the initial CYCLONE prototype. Features related to the placement and optimization can be phased in over the course of the project, provided that their support does not overly complicate the application model and description.

Despite our desire for a comprehensive description of the cloud applications, we also want to minimize intrusion into the application itself, avoiding unnecessary constraints. For example, one area where CYCLONE and its application description should remain agnostic is configuration management. Many developers already use a configuration management system, such as Chef[1] or Puppet[2], to configure and update their applications. To minimize barriers in adopting CYCLONE tools, we want developers to be able to reuse directly the application knowledge and expertise they have in their chosen configuration management tool. The CYCLONE application model and description must not be prescriptive here. Similarly, the application model and description must not impose constraints on the application's execution environment (e.g. operating system) or force the adoption of a particular programming language to read or create the application description or to access the CYCLONE APIs.

2.2. Requirements

The initial set of requirements was collected through discussions with the CYCLONE partners, representing both the cloud application developers (through the defined use cases) and the developers of the CYCLONE components.

Table 1 lists all of these requirements, providing an ID number, short title, level, and description for each. The keywords from RFC 2119 (IETF, 1997) are used to specify the requirement level and thus, to define a rough priority.

[1] https://www.chef.io
[2] https://puppetlabs.com

Table 1: Identified Requirements

ID	Title	Level	Description
1	Machine readability	MUST	The application descriptions will be processed primarily by the CYCLONE components. The chosen format MUST be easily readable by machine from a wide variety of different programming languages.
2	Human readability	SHOULD	Application developers may need to read or to create the descriptions directly for debugging, testing, or development of tooling. The chosen format SHOULD be easily readable by humans.
3	Easily understandable	SHOULD	The application descriptions SHOULD be easily understandable by humans, meaning that the schema should have limited complexity and be well documented (e.g. with tutorials, examples, APIs, specifications, etc.).
4	Extensible	MUST	The application deployment format MUST be extensible to allow for characteristics or features specific to CYCLONE components.
5	Maturity	SHOULD	The application deployment format SHOULD have production adoption within the wider IT community, demonstrating that it is appropriate for general adoption.
6	Application summary	MUST	The description format MUST allow the application developer to provide a detailed, human-readable summary of the application's features, characteristics, and limitations.
7	Single machine apps.	MUST	Many applications and services can be contained within a single virtual machine. The description format MUST be able to describe simple, single-VM applications.
8	Multiple machine apps.	MUST	Many applications require a number of different services deployed on different virtual machines. The description format MUST be able to describe applications composed of multiple machines.
9	Hierarchical composition	SHOULD	Complex applications are often built from reusable components that are themselves complex, multi-machine services. The description format SHOULD allow a hierarchical composition of components to foster reuse of descriptions, minimizing developer effort.
10	Virtual machines	MUST	The description format MUST be able to describe the all of the parameters for the provisioning of virtual machines on a cloud infrastructure.
11	Containers	SHOULD	Use of (Linux) containers is becoming more widespread as an alternative to full virtual machines. The description format SHOULD be able to describe all of the parameters for the provisioning of a container.
12	CPU specification	MUST	The description format MUST allow the application developer (or user) to define the characteristics of the CPU(s) required.
13	RAM specification	MUST	The description format MUST allow the application developer (or user) to define the amount of RAM required.
14	Disk specification	MUST	The description format MUST allow the application developer (or user) to define the amount of disk space available on the application's virtual machines.
15	Persistent storage	MAY	The description format MAY allow the application developer (or user) to specify the characteristics of persistent storage (type, amount, location, etc.).
16	Multi-cloud	SHOULD	The description format SHOULD allow the application developer to indicate which parts of an application can be deployed on different cloud infrastructures.
17	Placement policies	MAY	The description format MAY allow the application developer (or user)

			to specify policies for the placement of application components.
18	Network connectivity	MUST	The description format MUST allow the application developer to define the network connectivity between application components and between the user and the application. This concerns the accessibility of ports on the application's VMs.
19	Network isolation	SHOULD	The description format SHOULD allow the application developer to specify the characteristics of isolated network(s) to be created for a given application. This network isolation MAY be extended to machines outside of the application, such as a user's workstation.
20	Parameterization	MAY	The description format MAY allow an application or application component to be parameterized, allowing information to be passed into or out of an application at deployment or runtime.
21	Metrics	SHOULD	The description format SHOULD allow the application developer (or user) to specify what standard metrics (e.g. CPU load or RAM utilization) should be collected along with their frequency.
22	Dynamic metrics	MAY	The description format MAY allow the application developer (or user) to modify the collected metrics while the application is running.
23	External services	SHOULD	The description format SHOULD allow the application developer (or user) specify outside services to be used in conjunction with a deployed application (for example, external authentication databases for single sign-on).
24	Actions	MAY	The description format MAY allow actions to be taken in response to the values of collected metrics or provided key performance indicators (KPIs) to be defined.
25	SLA	MAY	The description format MAY allow service-level agreements to be defined as constraints on service placement and/or triggers to defined actions.
26	Credentials	MUST	The description format MUST be able to provide credentials for configuring access to deployed application services (e.g. SSH public keys for remote shell access).
27	Tooling	WOULD	Tooling (e.g. IDE) that facilitates the creation and use of the application descriptions WOULD be advantageous.
28	Lifecycle actions	MUST	The description format MUST allow the cloud application developer to define actions associated with state transitions in the application lifecycle.
29	Execution environment	MUST	The application model and description MUST not impose constraints on the execution environment of the application, for example, by excluding the use of certain operating systems (e.g. Windows).

3. Existing Application Models

Because of the large number of organizations (commercial and academic) involved, the cloud ecosystem contains a rich diversity of models, APIs, and standards, not all of which are directly relevant for CYCLONE. To select a subset of relevant models from this ecosystem, the models must be:

- **Able to describe (or be extended to describe) multi-machine applications.** Those that operate strictly at the Infrastructure-as-a-Service (IaaS) level with single machines are not interesting for the project.
- **Used actively in production by a significant community.** For CYCLONE, we want to ensure that the model is sufficiently mature to describe real-world applications and that it appeals to a significant number of application developers.
- **Independent of the specific features of a single tool.** Many models are strongly tied to a single tool preventing their use by other tools and greatly limiting interoperability between tools.

Based on these criteria, CYCLONE has chosen to analyze five application models: SlipStream, CIMI, TOSCA, OCCI and Docker Compose, which are currently available in state of the art and commonly recognized as the most relevant for this application area.

Each of the following subsections briefly describes the aforementioned application models along with advantages and disadvantages. The summary view of the matching of the selected models with respect to the CYCLONE requirements is shown in Table 2. This comparison is done with the most recent production version of the model, even though most of these models are actively evolving in response to feedback from both users and developers. Strategic conclusions deriving from this analysis are provided in the subsequent chapter 4.

Table 2: Evaluation of Application Models

ID	Title	Level	SlipStream	CIMI	TOSCA	OCCI	Docker
1	Machine readability	MUST	✓	✓	✓		✓
2	Human readability	SHOULD		✓			✓
3	Easily understandable	SHOULD		✓			✓
4	Extensible	MUST	✓	✓	✓	✓	
5	Maturity	SHOULD	✓	✓	✓	✓	✓
6	Application summary	MUST	✓	✓	✓		✓
7	Single machine apps.	MUST	✓	✓	✓	✓	✓
8	Multiple machine apps.	MUST	✓	✓	✓		✓
9	Hierarchical composition	SHOULD		✓	✓		✓
10	Virtual machines	MUST	✓	✓	✓	✓	
11	Containers	SHOULD					✓
12	CPU specification	MUST	✓	✓	✓	✓	✓
13	RAM specification	MUST	✓	✓	✓	✓	✓
14	Disk specification	MUST	✓	✓	✓	✓	
15	Persistent storage	MAY		✓		✓	
16	Multi-cloud	SHOULD	✓				✓
17	Placement policies	MAY					

18	Network connectivity	MUST		✓	✓	✓	✓
19	Network isolation	SHOULD		✓			✓
20	Parameterization	MAY	✓		✓		
21	Metrics	SHOULD		✓			
22	Dynamic metrics	MAY		✓			
23	External services	SHOULD					✓
24	Actions	MAY					
25	SLA	MAY					
26	Credentials	MUST	✓	✓	✓	✓	✓
27	Tooling	WOULD			✓		✓
28	Lifecycle actions	MUST	✓		✓		✓
29	Execution environment	MUST	✓	✓	✓	✓	

3.1. SlipStream (SixSq)

3.1.1. Overview

SlipStream is a multi-cloud application management platform that will be used by CYCLONE to handle the deployment and runtime optimization of (multi-)cloud applications. SlipStream provides a general cloud application model and an associated abstract API to hide differences between cloud service providers from the cloud application developers and operators, making cloud portability and multi-cloud applications a reality.

The SlipStream application model (SixSq Sàrl) consists of “Images” and “Deployments”:

- An **image** describes a single virtual machine containing information about its operating system, installed packages, configuration, and resource requirements.
- A **deployment** brings together machine images to create a full cloud application, for example binding a database, business logic, and web frontend into an n-tier web application.

Images (and thus deployments) can be parameterized, making them more generic and allowing them to be customized by the cloud application operator at deployment time. This application model is exposed to the SlipStream users through a web interface and through a REST API.

Although simple, the model has proven flexible enough to handle the advanced, large-scale scientific applications within the Helix Nebula[3] initiative, a partnership between big science and big business to create a federated cloud for European science. SlipStream is the primary interface to the Helix Nebula platform and is also used in a number of commercial deployments. Although specific to SlipStream, this application model has been included because SlipStream will be used by the project and it is important to understand its current capabilities.

3.1.2. Evaluation

The detailed evaluation against the requirements for the current SlipStream application model (v2.12) is shown in Table 2.

[3] http://helix-nebula.eu

In general, the SlipStream model and API cover the requirements well with the notable exceptions of those related to networking specifications and application monitoring. Those areas are expected to be covered as part of the CYCLONE work plan, most likely by an incremental evolution towards the CIMI model and API (described below).

Advantages:

- Allows immediate use of a proven and mature application model and API
- Already capable of full multi-cloud application deployments
- Permits user-defined triggers on state transitions for the application

Disadvantages:

- Does not currently include networking or monitoring features
- XML format is not user friendly
- Specificity to SlipStream will limit interest by people outside CYCLONE

3.2. CIMI (DMTF)

3.2.1. Overview

Defined by DMTF, the Cloud Infrastructure Management Interface (CIMI) (DMTF, 2013) is a standard that focuses on the deployment of applications into IaaS cloud infrastructures. It defines a set of low-level resources (networking, CPU, etc.) that are then aggregated into "machines". These machines can then be aggregated into "systems". Either a "machine" or "system" can be deployed onto a cloud infrastructure. The standard specifies both the CIMI model and API.

Although the vocabulary is somewhat different, the model itself is similar to SlipStream's one, allowing aggregation and reuse of definitions to create full, (multi-)cloud applications. The CIMI specification has a wider range of low-level resources, including networking and monitoring resources. However, parameterization, deployment and coordination of multiple machines, and user-defined triggers for states in the application lifecycle, are useful SlipStream features that are missing in CIMI model.

CIMI is one of the primary interfaces for δ-Cloud[4], an Apache project that abstracts the differences between clouds, which is strongly supported by RedHat. In addition, major IT vendors, such as IBM, Microsoft, RedHat, and VMware, participate in the definition of the standard and are showing interest in supporting the standard in their software and services. The endorsement of the CIMI API by ISO/IEC also provides an additional incentive for supporting it.

3.2.2. Evaluation

The detailed evaluation against the requirements for the most recent CIMI specification (v1.1.0) is shown in Table 2.

The specification continues to evolve with an advanced draft of a 2.0.0 version (DMTF, 2015) available. One notable change in the draft version is the addition of "network services" such as firewalls, VPNs, and

[4] https://deltacloud.apache.org

DNS. These additions fit in well with CYCLONE's aim to integrate advanced networking services into the cloud management platform.

Advantages:

- Detailed, clear specification of a comprehensive set of cloud resources, including monitoring
- Application model can be serialized into XML or JSON and it is well-documented
- Standard defined by DMTF and endorsed by ISO/IEC promises some level of interoperability

Disadvantages:

- Must be extended to include triggers for changes in the lifecycle state
- Must be extended to allow parameterization of applications

3.3. TOSCA (OASIS)

3.3.1. Overview

The Topology and Orchestration Specification for Cloud Applications (TOSCA) v1.0 (OASIS, 2013) is a standard defined by OASIS that allows Cloud Application Developers (CAD) to describe the deployment and operation of their applications such that a Cloud Service Provider (CSP) can deploy the application on its cloud infrastructure.

Within TOSCA, a full application description is built from one or more "services" that are deployed on "servers". The TOSCA model fully supports inheritance allowing the modular definition of services and servers. The application description is bundled together with the application's artifacts into a Cloud Service Archive (CSAR). A "TOSCA Container" then translates the CSAR into a set of deployment actions for a particular cloud to instantiate the application.

By design, TOSCA concentrates on the application definition and has no associated API – cloud service providers are expected to provide a TOSCA Container to deploy CSARs. This makes the specification independent of the concrete cloud API. The TOSCA Primer (OASIS, 2013) for example shows how CIMI can be used as the deployment API. SlipStream also has been effectively used as a TOSCA Container by CELAR[5], another European project, by translating the CSAR into SlipStream's native application model.

TOSCA is often used in academic projects dealing with the deployment of cloud applications. However, when talking with people from other projects using TOSCA, the feedback has been that it is very complex and awkward to use. Moreover, nearly every project has extended the specification without feeding those changes back into the standard, making interoperability of the TOSCA CSARs impossible. Some of the complexity comes from the choice of XML. The support of YAML (OASIS, 2015), which has recently been proposed, may simplify somewhat the application description.

3.3.2. Evaluation

The detailed evaluation against the requirements for TOSCA is shown in Table 2.

[5] http://www.celarcloud.eu

A strong point for this specification is the ability to bundle artifacts with the application description (such as deployment scripts) and to tie actions to specific triggers to transitions in the application lifecycle. Like other specifications, there is little associated with monitoring an application or advanced placement of application components.

Advantages:

- Cloud Service Archive (CSAR) allows artifacts to be bundled with the application description
- Triggers can be associated to lifecycle transitions
- Rather complete coverage of cloud resources, excepting monitoring and placement

Disadvantages:

- Known to be complex and awkward to work with
- Primary XML format is not user friendly
- Extensions of standard limit interoperability with other tools
- Requires separate TOSCA container to realize the deployment of the application

3.4. OCCI (OGF)

3.4.1. Overview

Open Cloud Computing Interface (OCCI) is a recommendation from the Open Grid Forum (OGF). At its core, OCCI (OGF, 2011) is a generic protocol and API for management tasks. It was originally used to define a remote management API for IaaS cloud services (OGF, 2011) with a specific binding for HTTP (OGF, 2011). At a later stage it has been extended to cover other types of cloud services, such as PaaS and SaaS.

The OCCI model for cloud infrastructures focuses strongly on the specification and deployment of virtual machines, identified as a "resource" within the model. The resource can then have networking, computing, and storage associated with it. Connectivity and access to persistent storage can be specified via network links and storage links. As a generic model and API, OCCI can be extended to model an application as a group of "resources"; however, this is not explicitly part of the main specifications.

The recommendation is supported by a large number of cloud distributions (e.g. OpenStack[6]) and tools (e.g. CompatibleOne[7]) although it is not usually the primary interface. It has also been adopted as the standard API for the EGI.eu[8] computing infrastructure in Europe.

3.4.2. Evaluation

The detailed evaluation against the requirements for OCCI is shown in Table 2.

The most notable difference with the other models is the lack of explicit, standard support for multi-machine cloud applications. Moreover, because of the OCCI's reliance on HTTP headers to pass resource information, eschewing a formal representation in the HTTP body, there is no practical way to serialize the

[6] https://www.openstack.org
[7] http://www.compatibleone.com/community/
[8] http://egi.eu

application model into an application description. Both of these are serious deficiencies when considering OCCI for CYCLONE.

Advantages:

- Reasonable support and penetration in the academic sector
- Good coverage of basic cloud resources including persistent storage

Disadvantages:

- Lack of explicit mechanism for describing multi-machine applications
- Lack of a serialized form for the application description

3.5. Compose YML (Docker)

3.5.1. Overview

Docker is a collection of services and tools which allow users to define, share and deploy applications. Docker uses "containers" rather than the virtual machines usually found on cloud infrastructures. Containers are isolated, standard processes running on a (Linux) operating system. As a consequence, they can be started and stopped very rapidly (typically in seconds), but offer less isolation than virtual machines running within a hypervisor. Docker, as an ecosystem, has become extremely popular with developers because of the ease of defining applications and the rapid deployment.

One tool within this ecosystem is "Docker Compose". This tool allows you to define all of an application's components (containers, configuration, volumes, etc.) in a single YAML file (Docker); you can then deploy and manage the full application as a unit. The underlying application model is similar to those for SlipStream and CIMI where machines (in this case, containers) are defined and then grouped into complex applications. Use of YAML format makes description of application and container terse and readable.

3.5.2. Evaluation

The detailed evaluation against the requirements for Compose YML is shown in Table 2.

As Docker is container-based, there are more limited options for specifying the machine resources than other systems but, on the other hand, is competitive with the other application models. Like other models, it does not have much functionality for monitoring and automated actions tied to metrics.

Advantages:

- Simple description format that can be understood by both machines and humans
- Can easily create modular, hierarchical descriptions of application components

Disadvantages:

- Very container oriented, lacking some control over the resource allocation for machines
- No formal specification of the schema, although extensive online documentation exists

4. Strategy

The analysis of the relevant application models in state of the art shows unsurprisingly that there is no existing model that currently covers all of the CYCLONE requirements. All of these models would need to be extended to cover the advanced placement and metric-oriented scaling features foreseen in the CYCLONE work plan. Except for that common deficiency, all of the models have similar coverage of the remaining requirements, except for OCCI. OCCI's lack of a practical serialization format and its lack of a native mechanism for describing complex applications, eliminate it from consideration.

For the remaining application models, the primary benefits of each are:

- **SlipStream**: The model covers well the basic requirements needed for the initial CYCLONE prototypes and can be used immediately.
- **CIMI**: The model and underlying API are well-defined by the standard and also provide resources to support advanced networking and monitoring features.
- **TOSCA**: Provides a serialization format (CSAR) that includes also artifacts associated with the application to be used, for example, for defining triggers on changes on the lifecycle state.
- **Docker**: Very simple and human-readable format along with the support of containers.

Each of these would need to be extended to cover the full set of requirements for CYCLONE.

To allow the project to advance as quickly as possible, a practical strategy has been adopted:

1. CYCLONE will start with the SlipStream model and description to allow use cases to be ported to and deployed immediately.
2. SlipStream will migrate incrementally to the CIMI model. The ported CYCLONE applications will follow this incremental migration.
3. CIMI will be extended to allow CYCLONE to handle the advanced placement and metrics-based scaling foreseen within the project's work plan.
4. A further evaluation of TOSCA will be done to see if using its bundled serialization format or other features could bring any additional benefits over those of the extended CIMI model.

This strategy will allow the project to start porting and deploying the pilot applications immediately while providing a solid, standardized platform to support the advanced features that will be treated later in the project.

References

DMTF. (2013, October 22). *Cloud Infrastructure Management Interface (CIMI) Model and RESTful HTTP-based Protocol.* Retrieved June 30, 2015, from Distributed Management Task Force, Inc.: http://dmtf.org/sites/default/files/standards/documents/DSP0263_1.1.0.pdf

DMTF. (2013, December 12). *Open Virtualization Format Specification.* Retrieved June 30, 2015, from Distributed Management Task Force, Inc.: http://www.dmtf.org/sites/default/files/standards/documents/DSP0243_2.1.0.pdf

DMTF. (2015, March 20). *Cloud Infrastructure Management Interface (CIMI) Model and RESTful HTTP-based Protocol.* Retrieved June 30, 2015, from Distributed Management Task Force, Inc.: http://dmtf.org/sites/default/files/standards/documents/DSP0263_2.0.0c.pdf

Docker. (n.d.). *docker-compose.yml reference*. Retrieved June 30, 2015, from https://docs.docker.com/compose/yml/

IETF. (1997, March 1). *Key words for use in RFCs to Indicate Requirement Levels.* Retrieved June 30, 2015, from Internet Engineering Task Force: https://www.ietf.org/rfc/rfc2119.txt

OASIS. (2013, January 13). *Topology and Orchestration Specification for Cloud Applications (TOSCA) Primer Version 1.0.* Retrieved June 30, 2015, from OASIS: http://docs.oasis-open.org/tosca/tosca-primer/v1.0/tosca-primer-v1.0.pdf

OASIS. (2013, November 25). *Topology and Orchestration Specification for Cloud Applications Version 1.0.* Retrieved June 30, 2015, from OASIS: http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.pdf

OASIS. (2015, April 14). *TOSCA Simple Profile in YAML Version 1.0.* Retrieved June 30, 2015, from OASIS: http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/csd03/TOSCA-Simple-Profile-YAML-v1.0-csd03.pdf

OGF. (2011, April 7). *Open Cloud Computing Interface - Core.* Retrieved June 30, 2015, from Open Grid Forum: https://www.ogf.org/documents/GFD.183.pdf

OGF. (2011, April 7). *Open Cloud Computing Interface - Infrastructure.* Retrieved June 30, 2015, from Open Grid Forum: https://www.ogf.org/documents/GFD.184.pdf

OGF. (2011, June 21). *Open Cloud Computing Interface - RESTful HTTP Rendering.* Retrieved June 30, 2015, from Open Grid Forum: https://www.ogf.org/documents/GFD.185.pdf

SixSq Sàrl. (n.d.). *SlipStream Documentation*. Retrieved June 30, 2015, from http://ssdocs.sixsq.com

[1]

Acronyms

API	Application Programming Interface
CIMI	Cloud Infrastructure Management Interface
CAD	Cloud Application Developer
CAU	Cloud Application User
CSP	Cloud Service Provider
DMTF	Distributed Management Task Force
DNS	Domain Name Server
IaaS	Infrastructure-as-a-Service
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
IT	Information Technology
KPI	Key Performance Indicator
OVF	Open Virtualization Format
PaaS	Platform-as-a-Service
REST	Representational State Transfer
RFC	Request for Comments
SaaS	Software-as-a-Service
SSH	Secure Shell
TOSCA	Topology and Orchestration Specification for Cloud Applications
VM	Virtual Machine
VPN	Virtual Private Network
YAML (YML)	YAML Ain't Markup Language

<END OF DOCUMENT>