Horizon 2020
H2020-ICT-2014-1

CloudLightning

Self-Organizing & Self-Managing Heterogeneous Cloud

Work Package 4: Self-organisation and self-management

Deliverable 4.1.1 Protocol Specification and API

THIS PROJECT HAS RECEIVED FUNDING FROM THE EUROPEAN UNION's HORIZON 2020 RESEARCH AND INNOVATION PROGRAMME UNDER GRANT AGREEMENT NUMBER 643946

Copyright ©2016 CloudLightning Project Consortium

Title: # D4.1.1 Protocol Specification and API

Version: 1.7

Author(s):

- Marian Neagul (IeAT)
- Ioan Drăgan (IeAT)
- Ciprian Dorin Crăciun (IeAT)

Contributing Author(s):

- John Morrison (UCC)
- Huanhuan Xiong (UCC)

Editor(s):

- Marian Neagul (IeAT)
- Ioan Drăgan (IeAT)

Reviewed By:

- Konstantinos Giannoutakis (CERTH)
- Perumal Kuppuudaiyar (INTEL)

Document Nature: Deliverable
Date: 1 February 2016
Dissemination Level: Public
Status: Final

THIS PROJECT HAS RECEIVED FUNDING FROM THE EUROPEAN UNION's HORIZON 2020 RESEARCH AND INNOVATION PROGRAMME UNDER GRANT AGREEMENT NUMBER 643946

Copyright ©2016 CloudLightning Project Consortium

Revision History

Date	Version	Author(s)	Change(s)
1 December 2015	0.1	Marian Neagul	Initial version imported
7 December 2015	0.2	Marian Neagul	Adopted initial ToC
9 December 2015	0.3	Marian Neagul	Initial descriptions
18 December 2015	0.4	Ioan Drăgan	Gateway Documentation
2 January 2016	0.5	HuanHuan Xiong, Ioan Drăgan	Imported Resource Manager documentation from architecture
2 January 2016	0.6	Marian Neagul	Initial Swagger Definitions
4 January 2016	0.7	Marian Neagul	Imported Catalog Services documentation
5 January 2016	0.8	Marian Neagul, Ioan Drăgan	API updates
11 January 2016	0.9	Ioan Drăgan	Various content update
13 January 2016	0.10	Ioan Drăgan	API messages/documents update
13 January 2016	0.11	Ioan Drăgan	Imported deliverable relationship messages from the architecture document
13 January 2016	0.12	Ioan Drăgan	Various message updates according to architecture and various other changes
16 January 2016	0.13	Marian Neagul	Added message relationship diagram and various other changes
18 January 2016	0.14	Marian Neagul, Ioan Drăgan	Added flow diagram, updated sequence diagrams and various other changes
19 January 2016	0.15	Ioan Drăgan, Marian Neagul	Various layout changes and new content
22 January 2016	0.16	Ioan Drăgan, Marian Neagul, Ciprian Dorin Crăciun	Various name changes as agreed in the TB meeting
23 January 2016	0.17	Marian Neagul, Ioan Drăgan	Content update
24 January 2016	0.18	Marian Neagul	Change layout and content to adhere to reviewer recommended structure

25 January 2016	0.5	Marian Neagul, Ciprian Dorin Crăciun, Ioan Drăgan	Include latest API changes contributed by Ciprian
26 January 2016	0.6	Ioan Drăgan, Marian Neagul	More content work
26 January 2016	1.0	Ioan Drăgan, Marian Neagul	Initial Review Version
28 January 2016	1.1	Ioan Drăgan, Marian Neagul	Various changes according to reviews
29 January 2016	1.2	Ioan Drăgan, Marian Neagul, John Morrison	Changes according to review
30 January 2016	1.3	Marian Neagul, Ioan Drăgan	Incorporating Entity-relationship diagram; Adopted D3.1 front-page; Unified term definitions with D3.1; Other changes;
31 January 2016	1.4	Marian Neagul	Final Review Version
31 January 2016	1.6	Marian Neagul, Ioan Drăgan	Preparing for final release: adopted new look, small fixes, changes according to reviewers requests
31 January 2016	1.7	Marian Neagul	Final Release

Definition of Terms

Terms	Description
Blueprint	A Blueprint is a workflow in which each node represents either a service or another Blueprint. The Blueprint is composed to automate a particular set of tasks or business processes.
Blueprint Catalogue	A collection of Blueprint descriptors.
Blueprint Descriptor	Blueprint Name, Parameters, Constraints and Metrics.
Blueprint Name	This is a name used to identify a Blueprint.
Service	A software service (or simply, a service) is a coherent, ready-to-use piece of executable code that is of value to the user.
Service Catalogue	A collection of service descriptors.
Service Descriptor	Service Name, Parameters, Constraints, Metrics and Service Implementation Option(s).
Service Name	This is common name used to identify all Service implementations of a specific Service (each of which is distinguished by its ServiceID).
ServiceID	This is a unique identifier for services.
Service Implementation Option	Options for each Service Implementation ServiceID, CL-Resource
CL-Resource	A CL-Resource is capable of supporting the execution of one or more types of Service. It is perceived, and manipulated, by the CL system as an indivisible entity.
Parameters	Atomic values affecting implementation selection (cluster size, hardware architecture).
Constraint	Logical expressions referring to parameters or metrics
Metric	Quantitatively or qualitatively characterizing value (cost, throughput)
Implementation Library	A collection of Service executables and their associated deployment Manifests.
Manifest	Execution Environment, Code Dependencies, etc.
Coalition	A Coalition is a collection of CL-Resources of the same type.

Cell	A Cell is a collection of Compute, Network and Storage resources.
vRack	A vRack is a partition of the Compute resources within a Cell.
vRack Manager	A vRack Manager is a software component used to maintain information about its associated vRack and to make local management decisions on how its resources are optimally configured to deliver on specific service requests.
vRack Manager Group	A vRack Manager Group is a group of vRacks Managers capable of self-organizing to meet specific objectives.
Cell Manager	A Cell Manager is a software component associated with a Cell. It maintains information about vRack Manager Groups in order to determine possible solutions to match Blueprint requirements.
Enterprise Application Developer	An Enterprise Application Developer (EAD) creates software aimed at helping End Users complete a task or job. They may be employed by Independent Software Vendors, Cloud Service Providers, Telecommunications and Network Service Providers, Systems Integrators, Technology Channel Partners, Hardware Manufacturers, and End User Organizations. EADs may also act as End Users.
Enterprise Application Operator	Enterprise Application Operators submit Blueprints to the CloudLightning system for execution on behalf of End Users.
Cloud Service Provider	A Cloud Service Provider (CSP) is an organization that offers some component of cloud computing – typically Infrastructure-as-a-Service (IaaS), Software-as-a-Service (SaaS) or Platform-as-a-Service (PaaS) – to other organizations or individuals.
End-user	An End User is the person for whom a software product is designed. The End User typically interacts with, or consumes the results from, a Blueprint running in the CloudLightning system.

Relationship with other Tasks

Protocol and API specification	Impact and Impacted by
Protocol and API	T3.1 CloudLightning Architecture definition
Resource Deployment API	T4.4 Service Deployment
Protocol specification	T5.1 Service description format
Catalog Service API, Gateway Service API	T5.2 Gateway Service
Gateway Service API	T5.3 Gateway Service API

Task relationships from the Protocol and API specification point of view

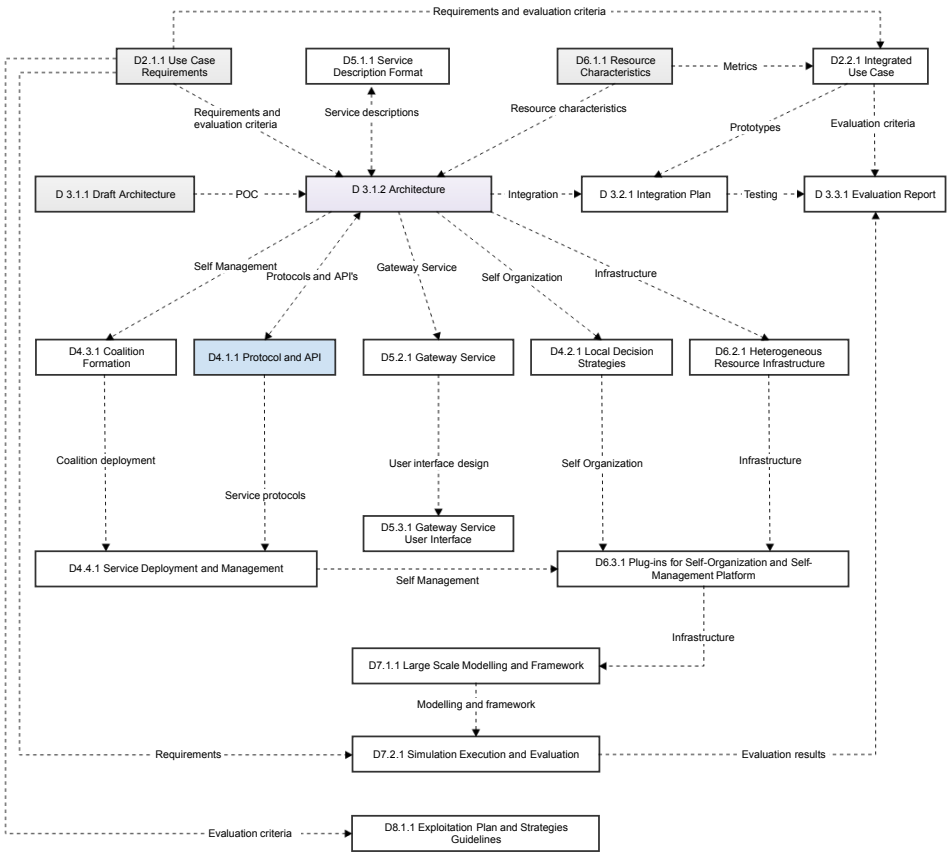

Figure 1: Deliverable Context

Executive Summary

In the context of creating a self-organizing and self-managing cloud infrastructure we propose a set of implementation-independent APIs aimed at establishing a common ground for the various CloudLightning components, i.e., between the cloud fabric and the service gateway, the various catalogues, the deployment infrastructure and, ultimately, an APIs consumable by the final user.

Contents

List of Figures

Abbreviations

API	Application Programming Interface
AWS	Amazon Web Service
CL	CloudLightning
CPU	Central Processing Unit
CSP	Cloud Service Provider
EAD	Enterprise Application Developer
EAO	Enterprise Application Operator
GUI	Graphical User Interface
HPC	High Performance Computing
HTC	High Throughput Computing
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
JSON	JavaScript Object Notation
OS	Operating System
PaaS	Platform as a Service
QoS	Quality of Service
RaaS	Resource as a Service
REST	Representational State Transfer
RPC	Remote Procedure Call
SaaS	Software as a Service
SOA	Service Oriented Architecture
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
UI	User Interface
VM	Virtual Machine
WP	Work Package
WSC	Warehouse Scale Computer

Chapter 1

Introduction

The main objective of this deliverable is to define the APIs offered by the various CloudLightning (CL) components, discussing their semantics and possible implementations. The information provided by this document is expected to serve as a primary reference for the APIs developed as part of the other WP4 tasks, as well as for the activities in the other work packages, particularly the WP5 Service Gateway and the various WP6 components.

In close cooperation with the partners working on task T3.1, the T4.1 task extended the existing self-organizing to include a self-managing feedback loop withing the self-organization process.

This deliverable should not be considered as a standalone reference but rather as an addendum to the D3.1.1 architectural document and D5.2.1 document.

Throughout the document we discuss each of the relevant core component groups identified as part of the D3.1.1 deliverable, present their expected APIs and functional characteristics. The document describes a modular implementation that captures a feedback loop incorporating requests for solution, the acquisition of those resources and the ultimate deployment of the application on those identified resources.

Figure 1.1 presents an overview of how the CloudLightning system is logically organized. From this figure we can distinguish several components that span on three major categories, that are:

- **Gateway Service** components. They include both the effective Gateway Service component and the corresponding user interface (Gateway Service UI) available to external users of the CloudLightning system;
- **Cell Management** components: they represent a collection of entities that are responsible for various resource management tasks. The critical tasks that have to be covered are *resource discovery, resource selection* and *resource acquisition*;
- **Deployment** component, which is responsible for deploying the selected configuration on the CL infrastructure;
- **Catalogues** components: this includes the Blueprint Catalogue, Service Catalogue

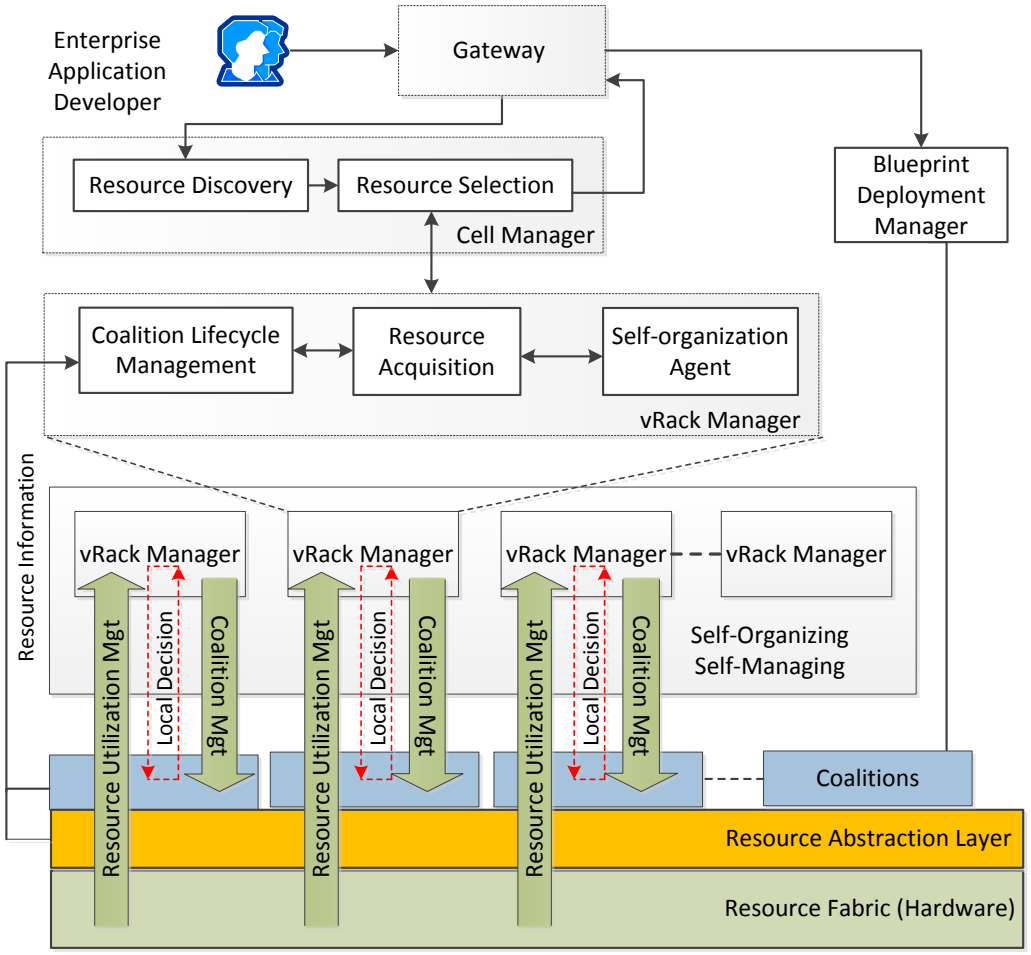

Figure 1.1: D3.1.1 CloudLightning System Architecture

and the Implementation Library. Where each of the catalogues are responsible for storing information about either services, service implementation or blueprint description.

1.1 Actors

An extensive list of the CloudLightning stakeholders is given in Deliverable D8.2.1, Dissemination Plan. From the architecture perspective, the following stakeholder interactions with the CloudLightning system are of interest:

- *(Enterprise) Application Developers:* An Enterprise Application Developer (EAD) creates software aimed at helping End Users to complete a task or job. They may be employed by Independent Software Vendors, Cloud Service Providers, Telecommunications and Network Service Providers, Systems Integrators, Technology Channel

Partners, Hardware Manufacturers, and End User Organizations. EADs may also act as End Users.

EADs may have many roles in the CloudLightning System. The protocols and APIs used by the EADs in these roles are also described in Deliverable D4.1.1. While acting as specialist service creators, EADs may create specialized services (these are recorded in the Service Catalogue and the Implementation Library); while acting as Blueprint Developers, EADs may compose services into Blueprints (these are saved in the Blueprint Catalogue); finally, while acting as Blueprint Operators, EADs may submit Blueprints to the CloudLightning for execution on behalf of End Users.

- *Cloud Service Providers:* A Cloud Service Provider (CSP) is an organization that offers some component of cloud computing – typically Infrastructure-as-a-Service (IaaS), Software-as-a-Service (SaaS) or Platform-as-a-Service (PaaS) – to other organizations or individuals.

- *End Users:* An End User is the person for whom a software product is designed. The End User typically interacts with, or consumes the results from, a Blueprint running in the CloudLightning system.

As mentioned above, service creation is, and will remain in the CloudLightning approach, a highly specialized task to be undertaken by experts in both the application domain and the underlying technologies. These experts are the Enterprise Application Developers. EADs will create service solutions for a specific hardware platform, will profile them and will register them, together with relevant meta information, with the *Services Catalogue* and the *Implementation Library* of the CloudLightning system. The Service Catalogue contains several attributes such as the name of the service, a description, a list of annotations (for limiting the choice of service implementations, parameters, constraints, and metrics). The same service may be implemented on many different hardware types and executable code, for each implementation, should be included in the Implementation Library.

It needs to be noted that the API's described in this document should not be considered final as they represent only the current state of the CloudLightning ecosystem and it is possible to suffer changes as the architecture changes.

1.2 Structure

This document starts by describing in Chapter how this deliverable relates to other activities and deliverables, particularly the CloudLightning architecture related activities. Chapter 2 discusses each major component group by describing the operations and methods for each of them while Chapter 3 summarizes the results presented in this deliverable. In Appendix A we present the Swagger API documentation corresponding to these components.

1.3 Conventions

The API mimics the semantics of a document store (or key-value store) thus the following actions SHOULD be obeyed:

- any URL that supports **PUT** should also support **GET** (obviously returning the same content that was just PUT);
- **PUT** should be called once (and only once) by the creator of the document; (i.e. the resource designated by the URL should be considered immutable;)
- for those resources that could be updated, **PUT** should be used (instead of POST or PATH), and the entire content should be resubmitted;
- **DELETE** could be used to destroy the resource designated by the URL, but should be used sparingly; (i.e. one could delete the attachments, but keep the descriptor of an object);
- in case an object is expected to be updated it should include at least the HTTP header 'Max-Age' with a proper value;
- if a **GET** response doesn't provide the HTTP header 'Max-Age' the client is free to assume that the resource is immutable, thus cacheable;
- URL's should never be reused; (i.e. identifiers should be globally unique;)

When using JSON payload, **POST** should be used very sparingly to trigger actions (like invoking the service matcher). The request body should be as small as possible, usually URL's for both where the input data should be obtained, and where the output data should be submitted.

In the case of asynchronous operations instead of **POST** one could model the invocation by using a pair of PUT. One PUT will be used in order to trigger the action and designating a URL where a second PUT will store the outcome.

Chapter 2

Core Components

This chapter outlines each of the CloudLightning (CL) system components, how they interact with each other and the APIs they expose. From the general architecture of the CL system we can distinguish four major logical groups of components. That is, *Gateway*, *Catalogue*, *Cell Management* and *Deployment*.

In the following we describe each of the components in more details and how they interact with each other. In Figure 2.1 we visually present the interactions between all the core components of the CL system as well as interactions with various types of users. The rest of this chapter is structured as follows. In Section 2.1 we describe the Gateway service component. Section 2.2 presents another core component of the CL system, that is the Catalogue service, followed by the description of the Cell Management services in Section 2.3.

Each of the sections describes the corresponding component operations, how they relate with other components and what are the involved messages. For each of the components we describe the REST[3, 7] API and further document it in Section A.

Figure 2.2 depicts the relationship between the various messages[1] involved in the operation of the afore mentioned components. Each of the messages and its relationship to other messages and operations is discussed in its own subsections. In the rest of this document, we denote *user* either the *developer* or the *operator* unless otherwise specifically referenced by their role.

2.1 Gateway

The *Gateway* service represents the gateway between the *self-organizing and self-managing* cloud and the outside world. It allows consumers (normally solution developers) to submit service descriptions and trigger the self organizing mechanisms.

The Gateway service gathers solutions (represented by computational hardware coalitions) generated by the self-organizing mechanisms and presents them to the enterprise

[1]Note that the term *message* is used interchangeably with the term *document*

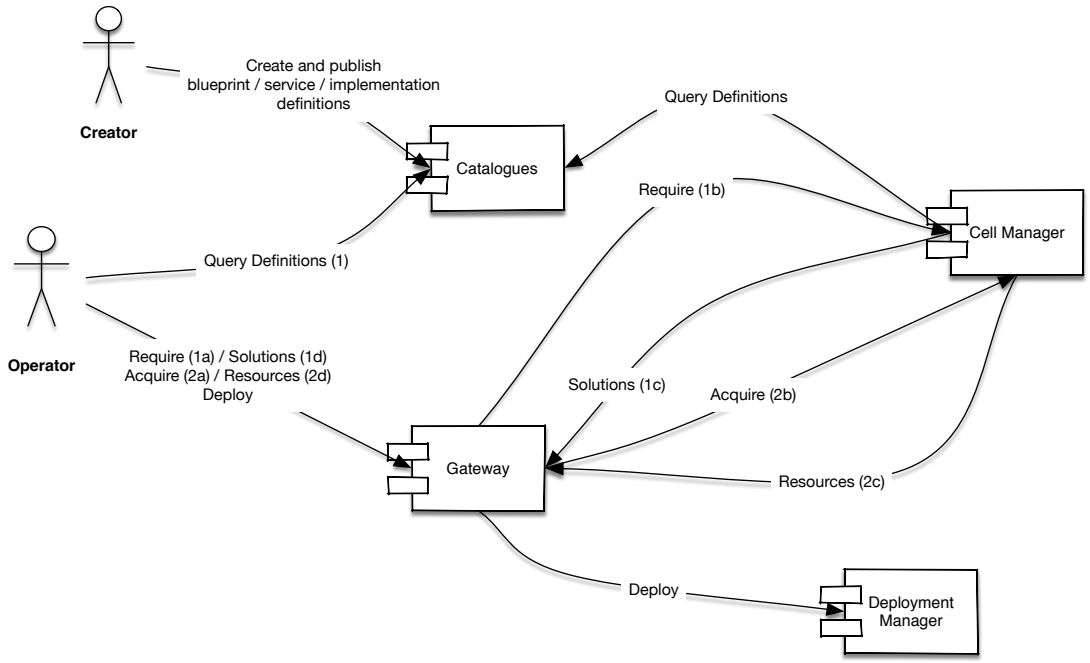

Figure 2.1: CloudLightning API Flow Diagram

user. Based on these solutions the user selects the desired one and asks the Gateway service to trigger deployment of the corresponding services. Service deployment and management is further handled by the components developed in other tasks of this project, particularly tasks T4.4 and T6.3 handling service deployment and management, respectively the creation of Self-organization and Self-management Plugins.

Gateway service represents the main way consumers can interact with the self-organizing and self-managing subsystem, abstracting out CL's inner workings and providing proxy functionality for other components that require consumer interaction. On the other hand, the Gateway service consumes a series of user provided and computer generated documents that provide the required information needed for its proper operation. Each of the consumed or generated documents is discussed in detail in Subsection 2.1.3.

It is important to note that the *gateway service* is designed in such a manner to not produce functional changes to the documents traveling between the consumers and the *Cell Manager*.

Gateway UI In a nutshell the *Gateway service UI* works as an interface between the CloudLightning system and all types of users. It interacts with the API exposed by the Gateway service so that it exposes a user friendly interface to CloudLightning.

By doing so the user is empowered to use all the major functionalities of the CloudLightning system through this interface. That is, it facilitates the interaction with different components of CloudLightning. For example the Gateway service UI is essential in the process of submission of a new application blueprint to the CloudLightning in order to obtain execution solutions for it. The interaction between the user and CloudLightning can be achieved using either in a programmatic manner via the API's documented in this

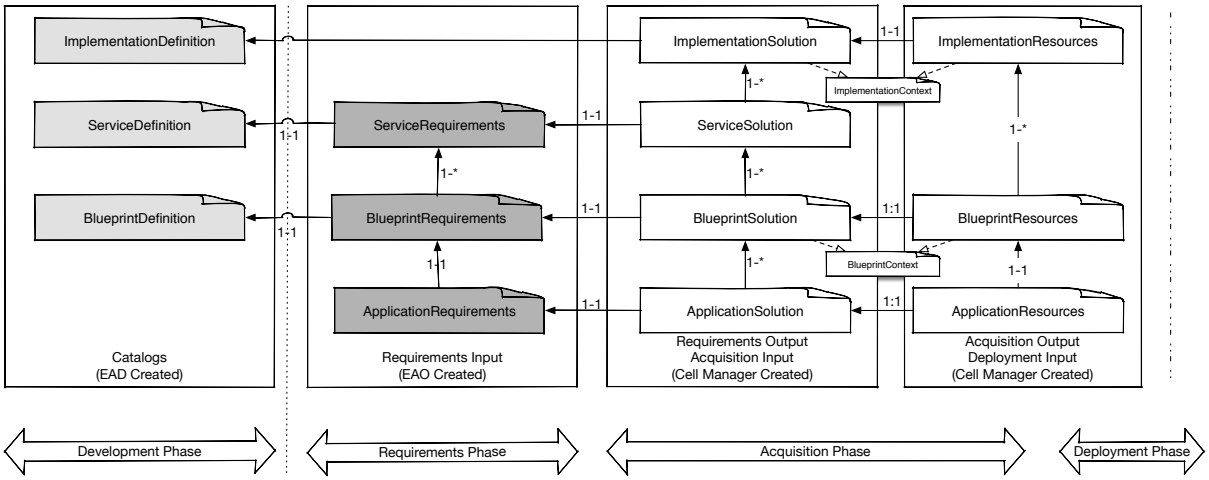

Figure 2.2: CloudLightning Message Relationships

deliverable or the Gateway service UI, for visual representation. As soon as the Cell Management system manages to find suitable services for the current application blueprint, they are presented to the user via the UI. Next the user will be able to select a suitable execution solution and finally deploy the application.

2.1.1 Interactions

From the interactions between components perspective, we can outline the following main actors:

- **Enterprise Application Operator**: as depicted in Figure 2.3 the operator is responsible for triggering the operations at Gateway service level and its underlying components. It creates and submits the required documents, like: *ServiceRequirements* (see 2.1.3.1), *BlueprintRequirements* (see 2.1.3.2) and *ApplicationRequirements* (see 2.1.3.3). Through the rest of the document we use interchangeably the term *operator* to denote Enterprise Application Operator;

- **Cell Management** components: are those components responsible for handling the effective resource management requests. They are designed in order to trigger the self-organizing and self-managing behavior. These components represent black-boxes for the *Gateway Service* as their inner workings are not relevant to the gateway service. In Section 2.3 we will provide a broader overview for each of the components.

  The Cell Management subsystem is contacted by the *Gateway Service* on behalf of the User with resource management requests like: discovering services according to the *ServiceRequirements* and *ApplicationRequirements* documents.

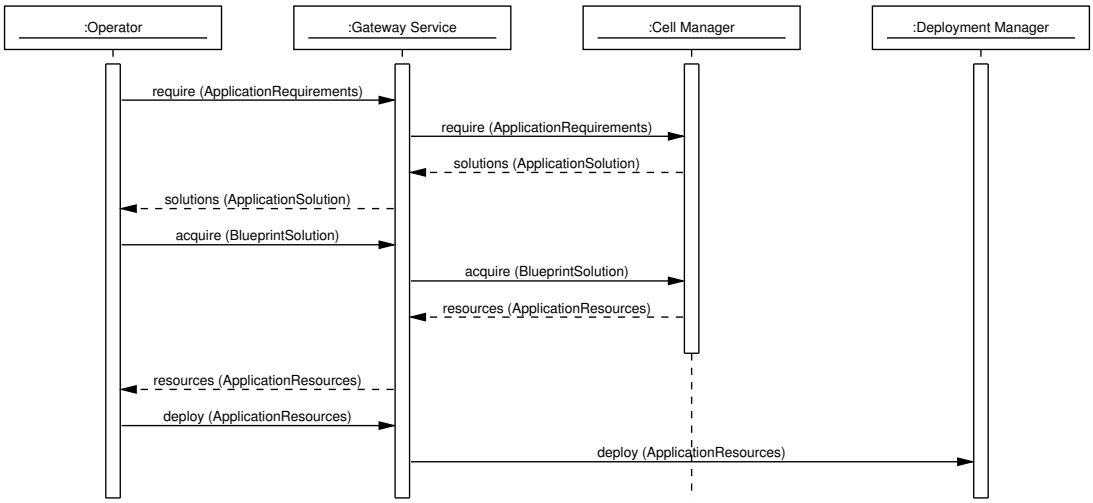

Figure 2.3: Gateway Service Sequence Diagram

2.1.2 Operations

Due to the fact that the *Gateway* service's purpose is to mediate the access to the self-organizing and self-managing infrastructure, it's API is designed to provide the required operations. The operations *should* be as independent as possible from underlying resource manager implementation.

As briefly depicted in Figure 2.3 the main operations of the *Gateway Service* that are exposed to the users are:

- Application requirements handling operations: these operations allow the submission of *ApplicationRequirements*, *BlueprintRequirements* and *ServiceRequirements* documents potentially triggering the resource lookup mechanisms by means of the *Require* operation. These operations are discussed in the sections 2.1.2.1, 2.1.2.2 and 2.1.2.3;
- Presentation of the application execution solutions: allowing the presentation of the resources available for application executions, discussed in Section 2.1.2.4;
- Resource allocation: allowing the consumer to acquire one of the solutions provided by the *Cell Manager*. This operation type is discussed in Section 2.1.2.5;
- Deployment brokering: this operation allows the triggering of application deployment inside the acquired resources. This operation is presented in Section 2.1.2.6;

The interfaces of the operations exposed are described in Appendix A but their concrete implementation is open to change as the CloudLightning project evolves.

2.1.2.1 Service Requirements Registration

This operation allows the user to register *ServiceRequirements* (specifying functional constraints as described in Section 2.1.3.1). This operation does not have any side effects, it

only allows the document to be registered in order to allow it to be referenced in other documents and operations. The main operation referencing the submitted document is the *Requirements Submission* operation discusses in Subsection 2.1.2.3.

Similarly to other *Gateway Service* operations, the *Service Requirements Registration* is accessible to consumers as a HTTP REST service. This service consumes *ServiceRequirements* submitted as HTTP POST body payloads.

2.1.2.2 Blueprint Requirements Registration

The blueprint requirement registration operation enhances the users with the possibility of registering predefined blueprints in the CloudLightning system. The blueprint requirements are specified in a document called *BlueprintRequirements* document. As for the case of Service Requirements registration, the operation has no side-effects. That is, it only allows the document to be registered in order to allow it to be referenced in other documents and operations.

Similarly to other *Gateway Service* operations, the *Blueprint Requirements Registration* is accessible to consumers as a HTTP REST service. The service consumes the *BlueprintRequirements* schema submitted as HTTP POST body payloads.

2.1.2.3 Application Requirements Submission

The application requirements submission operation represents the main tool for the user to trigger the internal CloudLightning self organization of resources. In a nutshell the *ApplicationRequirements* document references the blueprint for which resources are needed. The submission of such a document triggers internally the self organization part of the CL system. Unlike the previously described operations, application requirements submission has side-effects. That is, by triggering the self organization part of the system it receives as response a document containing potentially multiple solutions that meet the described resources. In Subsection 2.1.2.4, we present in more details how these solutions are presented back to the user.

In a similar manner though, the Application Requirements Submission is available to consumers through a HTTP REST service. In this case, the service consumes the *ApplicationRequirements* schema that was submitted by as HTTP POST body.

2.1.2.4 Solution Presentation

The *Solution Presentation* operation is the actual response that the self organization mechanisms returns to the users. Through this operation the system is able to present different offers that satisfy the *ApplicationRequirements* submitted by the user. Although this operation is the main communication channel between the self management components it is not the final step in the process of resource acquisition.

Unlike the Application Requirements Submission this operation has no side-effects, meaning it does not alter the state of the CloudLightning system. The document that is

present to the users is nothing else than the *ApplicationSolution* document.

Similar to other operations, the Solution Presentation is available to the users through a HTTP REST service. More precisely the ApplicationSolution is submitted to the user through a HTTP POST, with the appropriate body, from the Gateway to the User.

2.1.2.5 Resource Acquisition

This operation is responsible for the effective "*acquisition*" of the resources identified by the CloudLightning fabric represented by the *Cell Manager*. The resource allocation operation resumes to the efective allocation of the resources, making them available for subsequent application deployment. This operation is aimed to notify the *Cell Manager* about the decision taken by the *operator* regarding the resources proposed.

From the *Gateway Service* perspective this operation only acts as a message exchange service between the consumer (represented by the operator) and the cloud fabric. The *Gateway Service* does not alter by any means the handled *documents*, forwarding them straight to the *Cell Manager*. Resource acquisition operation accepts as input *ApplicationRequirements* documents. The document content and structure is discussed in Subsection 2.1.3.3.

As results this operation generates a series of *ApplicationResources* documents (documented in Section 2.3.3.5).

The exact operations handled by the Resource Acquisition is subject to deliverables D4.2.1, D4.3.1, D4.4.1 and D5.2.1.

2.1.2.6 Resource Deployment

The resource deployment operation is responsible for mediating the resource deployment requests to the service deployment component. The operation involves the *ApplicationResources* document (as defined in Section 2.3.3.5) and does not have an explicit result.

2.1.3 Documents

In the following we will briefly describe each of the documents that are used by the API in order to communicate between the different core components.

2.1.3.1 ServiceRequirements

The *ServiceRequirements* document is aimed to allow the user to define functional constraints for a specific service interface. The defined constraints are evaluated at resource matching time and are used for selecting the optimal resources satisfying the user requirements. The document is consumed by the underlying resource management system and it's goal is to express the per service requirements for executing the service implementing the referenced interface.

The *ServiceRequirements* document references the *ServiceDefinition* (see Section 2.2.3.1 and, optionally, the *BlueprintDefinition* (see Section 2.2.3.3) documents.

```
{
    "type": "object",
    "properties": {
        "links": {
            "type": "object",
            "properties": {
                "session": {
                    "type": "string",
                    "format": "url"
                }
            }
        },
        "parameters": {
            "$ref": "#/definitions/ParameterRequirementsObject"
        },
        "constraints": {
            "$ref": "#/definitions/ConstraintRequirementsObject"
        }
    },
    "x-TBD": true
}
```

Figure 2.4: ServiceRequirements Example Schema

An example *ServiceRequirements* is presented in Figure 2.4.

2.1.3.2 BlueprintRequirements

The *BlueprintRequirements* document is aimed in mapping user provided requirements to a *blueprint* already registered in the catalog.

This document maps together the predefined blueprint and a series of service requirements defined by means of *ServiceRequirements* documents (discussed in more detail in Section 2.1.3.1).

In Figure 2.5 we present an example *BlueprintRequirements* document referencing a predefined blueprint and one service requirement.

2.1.3.3 ApplicationRequirements

The *ApplicationRequirements* document serves as the primary user input for triggering the self organization process, it references the *BlueprintRequirements* document (as de-

```
{
    "type": "object",
    "properties": {
        "links": {
            "type": "object",
            "properties": {
                "services": {
                    "type": "array",
                    "items": {
                        "type": "string",
                        "format": "url"
                    }
                },
                "session": {
                    "type": "string",
                    "format": "url"
                }
            }
        },
        "parameters": {
            "$ref": "#/definitions/ParameterRequirementsObject"
        },
        "constraints": {
            "$ref": "#/definitions/ConstraintRequirementsObject"
        }
    },
    "x-TBD": true
}
```

Figure 2.5: BlueprintRequirements Example Schema

scribed in the Section 2.1.3.2) and potentially provides various non-functional constraints[2] (various Key Performance Indicator (KPI)).

This document provides/references all the information required by the underlying Self-Organization and Self-Management components. This information includes (indirectly):

- *BlueprintRequirements*: exactly one document reference;
- *ServiceRequirements*: one for each service referenced inside the *BlueprintRequirements* document.

[2]Support for specifying non-functional constraints is expected in a later project stage

```
{
    "type": "object",
    "properties": {
        "links": {
            "type": "object",
            "properties": {
                "blueprint": {
                    "type": "string",
                    "format": "url"
                },
                "session": {
                    "type": "string",
                    "format": "url"
                }
            }
        }
    },
    "x-TBD": true
}
```

Figure 2.6: ApplicationRequirements Example Schema

2.2 Catalogue

The *Catalogue* service is one of the essential components of the CL infrastructure. It is responsible of keeping track of all the services that are registered in the system. Alongside it also keeps track of all the blueprint descriptions that where added to CL. All the information stored in the Catalogue service is critical for the communication between users and the CL system. Although we view the Catalogue as a single component in the actual implementation we have multiple components that store the appropriate information.

In a nutshell the Catalogue service provides ways for the users (either Creator or Developer) to add new services to the list of already existing ones. Besides that, it also empowers the users with ways of searching for different services based on interfaces and also navigate through the set of already deployed (and made public) blueprint descriptions. All these operations are motivated by the way CL system is designed. That is, in order for users to define requirements for their applications atomic services must be defined. If a service is already defined in the catalog it can be further used by users in order to describe their applications. An application defined by a user is just a composition of different services that follow a specific logical flow. The relationship between the user provided definitions is outlined in Figure 2.7.

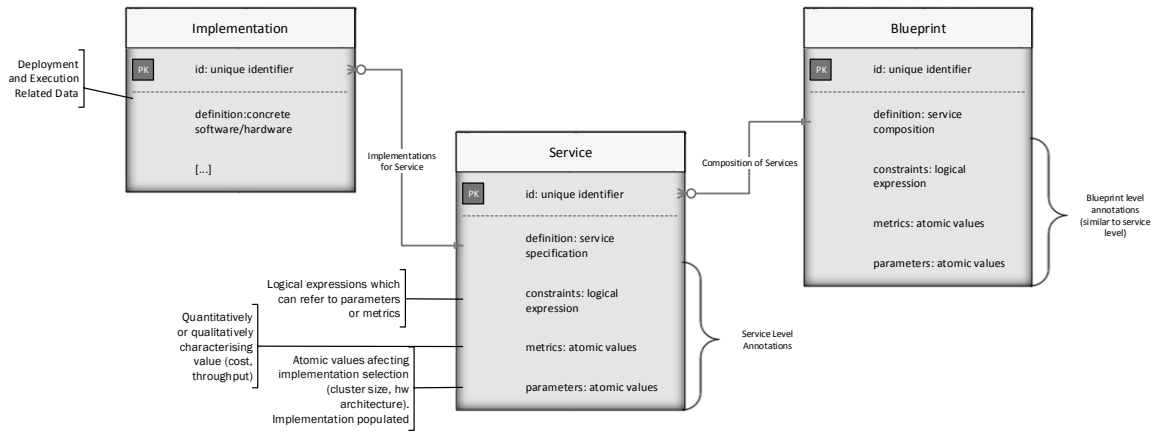

Figure 2.7: Catalog Entity-relationship Diagram

Being one of the core components of the CloudLightning system it basically interacts through the exposed API's with all other core components of the system.

The catalog is composed out of several subcomponents:

- *Blueprint catalogue*: contains blueprint components definition. Each blueprint definition is characterized by several attributes like: name, description, list of annotations (used by the users for limiting the choice of service fingerprints used for forming coalitions: parameters, constraints, metrics);
- *Service catalog*: contains service implementation, one service implementation uniquely describes one of the available service description and contains implementation specific details;

- *Implementation library*: containing implementation files (eg. binaries, recipes, etc)

It is important to note that the provided document definition are intended to be as generic as possible, offering the opportunity to later extension. Of particular interest are the *Blueprint* documents that are expected to be extended as part of WP5, eventually providing support for existing specification formats (like TOSCA[11], ClassAd[10], etc).

2.2.1 Interactions

From the interactions between components perspective, we can outline the following main actors:

- **Creator/Operator**: this actor is responsible for defining different services that have to run on the CL system. Definitions of individual services are stored in the Catalogue component by calling specialized API's for each of the operations. As soon as the user identifies the *Definitions* for the desired services it adds them to the catalogue, see 2.2.3.1. After the individual service interfaces are defined and stored in the catalogue the Creator is responsible for creating and registering their implementations, see 2.2.3.2. Once all the previous steps are completed, a blueprint can be created and registered in the CL system, see 2.2.3.3.
- **Cell Management**. As discussed in Section 2.3 the cell management components are responsible for handling the effective resource management requests including self-organizing and self-managing behavior. When it comes to interactions with the Catalogue, the cell management components query the service catalogue for service interfaces and blueprint descriptions, for more details about the documents that are communicated see 2.2.3.3. More details about interactions with the different catalogue components can be found also in Section 2.3.

Figure 2.8 represents the Catalogue Service sequence diagram that depicts the interactions with other core components of the CL system.

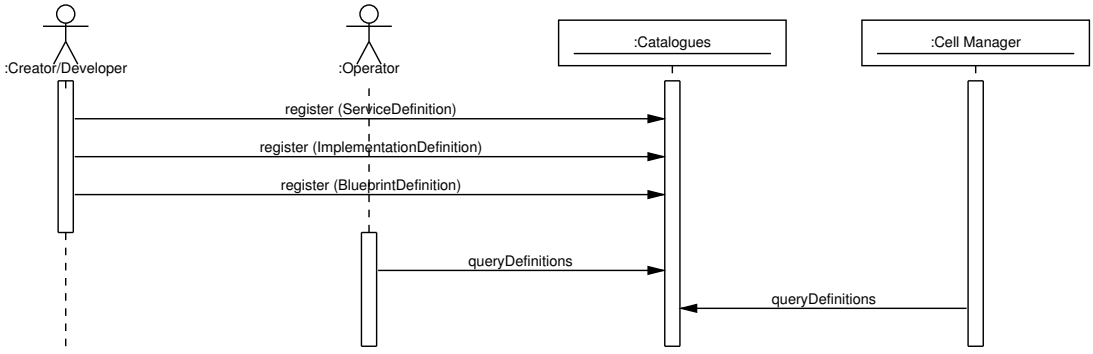

Figure 2.8: Catalogue services sequence diagram

2.2.2 Operations

Being one of the core components in the CL system, the Catalogue service is interacting with most of the other components. The operations supported should be independent and allow different types of users (Creator/Operator) and the CL system to closely interact.

As depicted also in Figure 2.8 the main operation groups exposed are:

- **Service registration**: the operations exposed in the service registration Section *should* present ways for adding, updating and deleting services from the CL Service Catalogue.
- **Blueprint registration**: this group of services is responsible for the tight communication between the CL system and the user requirements for different applications that could be deployed on the system. It **must** provide ways to add, update and delete blueprints.
- **Catalogue query**: this operation group **must** provide mechanisms for the enterprise users to query the Catalogue service but also provides procedures for other components to interact with the catalogues.

The actual operations exposed by the Catalogue service are presented in Appendix A. The operations presented in the appendix are just a hint aimed at providing the direction for what will be implemented. The actual implementation for each of the operations will evolve throughout the duration of the project.

2.2.3 Documents

In order to better understand how communication between the different API's is done one needs to better understand the documents that are communicated. In the following we will describe each of the documents that are related to the Catalog Service.

2.2.3.1 ServiceDefinition

Whenever a new service has to be registered in the CL system the user starts by defining a *ServiceDefinition* document. This document is responsible for describing the interfaces that are implemented by a specific service. In a nutshell this document can be viewed as the conceptual description of the operations that are implemented by a service.

The document contains all the required information needed to characterize a service. The exact content of this document is implementation specific. The only required attributes representing parameters (list of enumeration), constraints (list of logical expressions) and metrics (qualitative and quantitative characteristics of solutions that will fulfill the service).

Figure 2.9 presents a minimalistic document that can represent a *ServiceDefinition*.

```
{
    "type": "object",
    "properties": {
        "parameters": {
            "$ref": "#/definitions/ParameterDefinitionsObject"
        },
        "metrics": {
            "$ref": "#/definitions/MetricDefinitionsObject"
        },
        "constraints": {
            "$ref": "#/definitions/ConstraintDefinitionsObject"
        }
    },
    "x-TBD": true
}

```

Figure 2.9: ServiceDefinition Example Schema

2.2.3.2 ImplementationDefinition

In the *ImplementationDefinition* document the user has to define how different services are implemented.

The document contains all the required information needed to characterize a service implementation. The exact content of this document is implementation specific. The only required attributes representing parameters (list of enumeration), constraints (list of logical expressions) and metrics (qualitative and quantitative characteristics of implementations that will fulfill the service).

Keep in mind that before one defines the implementation document one or more definition documents must be created. Each of the implementation definition documents must contain at least one reference to a definition document. That is, each implementation should materialize at least one, but potentially could materialize more than one definitions.

In Figure 2.10 we present a minimalistic example of an *ImplementationDefinition*. As mentioned also in the description we need to specify the service definitions that are implemented.

2.2.3.3 BlueprintDefinition

The *BlueprintDefinition* represents one of the core documents that has to be defined by the user. In this document the user has to define the interactions between different services so that in the end it will describe the intended application. Basically the document could present both the topology of the application, constraints and interactions between the different components of the intended application.

The document contains all the required information needed to characterize a blueprint.

```
{
    "type": "object",
    "properties": {
        "links": {
            "type": "object",
            "properties": {
                "services": {
                    "type": "array",
                    "items": {
                        "type": "string",
                        "format": "url"
                    }
                }
            }
        },
        "parameters": {
            "$ref": "#/definitions/ParameterDefinitionsObject"
        },
        "metrics": {
            "$ref": "#/definitions/MetricDefinitionsObject"
        },
        "constraints": {
            "$ref": "#/definitions/ConstraintDefinitionsObject"
        }
    },
    "x-TBD": true
}
```

Figure 2.10: ImplementationDefinition Example Schema

The exact content of this document is implementation specific. The only required attributes representing parameters (list of enumeration), constraints (list of logical expressions) and metrics (qualitative and quantitative characteristics applying to the whole blueprint).

Figure 2.11 presents an example *BlueprintDefinition* document.

To summarize, the *BlueprintDefinition* document represents in a sense the requirements from the user with regard to the application that he wants to deploy. All the services that are used in order to describe the current blueprint definition must be already added to CL system.

```
{
    "type": "object",
    "properties": {
        "links": {
            "type": "object",
            "properties": {
                "services": {
                    "type": "array",
                    "items": {
                        "type": "string",
                        "format": "url"
                    }
                }
            }
        },
        "parameters": {
            "$ref": "#/definitions/ParameterDefinitionsObject"
        },
        "metrics": {
            "$ref": "#/definitions/MetricDefinitionsObject"
        },
        "constraints": {
            "$ref": "#/definitions/ConstraintDefinitionsObject"
        }
    },
    "x-TBD": true
}
```

Figure 2.11: BlueprintDefinition Example Schema

2.3 Cell Management Service

In a nutshell the *Cell Management Service* is one of the most essential logical components of the CL architecture. That is, the cell management service is in charge of finding matching resources for each of the service requirements that are stored in the system. Also it keeps track of all the coalitions that are formed and updates them in order to fulfill the current needs of the developers. Besides keeping track of coalitions and resources it is also in charge of service deployment for each individual blueprint.

Although in the API architecture it is reflected as a single component, in reality this logical component is composed from multiple subcomponents that are critical in order to keep the CL system running. In the following we will briefly discuss the interactions with other core components, the API's it exposes and also the documents that make possible communication between CL's components.

2.3.1 Interactions

As previously presented in Sections 2.2 and 2.1, the Cell Management Service is connected with all the components of the CL infrastructure. From the *Cell Manager's* perspective one can observe the interactions with the following components:

- **Catalogue service**: consumes documents stored within the *Catalogue service*. Information retrieved includes both documents received from the Gateway service and documents generated by the *Cell Manager* that have been stored for later reuse. Another important set of documents stored within the *Catalogue service* is represented by solution documents containing resources matched by the *Cell Manager*. These documents are further detailed in Section 2.3.3.
- **Gateway service**: all interactions are based upon requests initiated by the *Gateway service* and are *request-reply* like interactions. All interactions involve referencing documents handled by the *Catalogue service*. The operations involved in the interactions with the *Catalogue service* are further discussed in Section 2.3.2.

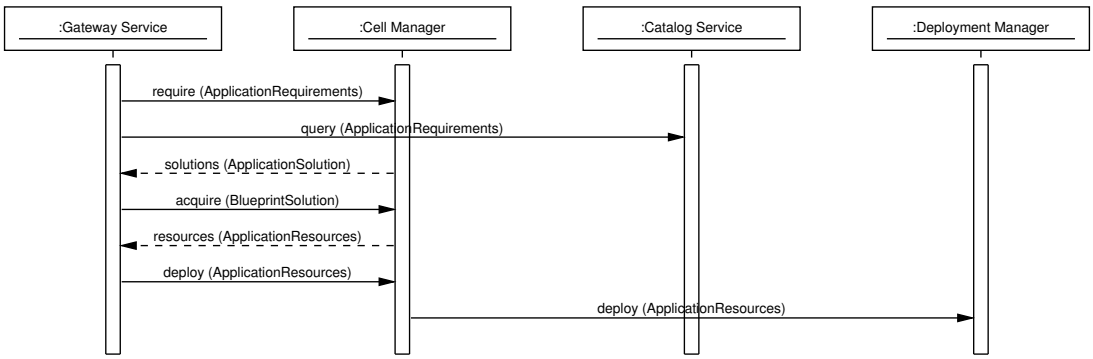

Figure 2.12: Resource Management Service

In Figure 2.12 one can observe the interactions from the perspective of the Cell Management service.

2.3.2 Operations

2.3.2.1 Application *Require*

The *Gateway service* initiates the *Application Requirements Tendering* operation by submitting a reference to an *ApplicationRequirements* document. The requirements document was published before by the *operator* as described in Section 2.1.3.3. The *ApplicationRequirements* document references in turn *BlueprintRequirements* and *ServiceRequirements*.

This operation triggers the internal self-organization and self-management processes aimed at finding resources satisfying the *operator* provided requirements. By itself this operation does not commit the identified resources, but rather potentially reserves the resources[3]. As a result it generates and publishes the corresponding solutions wrapped inside an *ApplicationSolution* document (see Section 2.3.3.1) to the *Catalogue service*.

The generated solution documents include (directly or inherited through references):

- blueprint solutions: at least one *BlueprintSolution* (as described in Section 2.3.3.2). Might contain one blueprint solution for each alternative solution;
- service solutions (for each blueprint solution): at least one *ServiceSolution* (see Section 2.3.3.3) for each service inside a blueprint solution;
- implementation solutions (for each service solution): provide at least one *ImplementationSolution* (see Section 2.3.3.4) document corresponding for a service implementation;
- context information (both blueprint and implementation levels): contains *Cell Manager* specific information usable in later stages;

This API document discusses only the API paradigms exposed by the *Cell Manager*, it's inner workings are discussed in the following deliverables:

- D4.2.1 Local Decision Strategies, as part of WP4 (Self-organization and Self-management) work-package;
- D4.3.1 Coalition Formation part of WP4;
- D4.4.1 Service Deployment and Management part of WP4;

2.3.2.2 Application *Acquire*

The *Cell Manager*'s acquire operation is responsible for effectively committing the resources to the *operator* requesting them. The operation receives a reference to one of the *blueprint solutions* returned by the previous "application require" operation.

As a result of this operation the *operator* obtains the resource documents, particularly: *ApplicationResources* (Section 2.3.3.5), *BlueprintResources* and *ImplementationResources*.

[3]The reservation behavior is subject to the self-organization algorithm design

This documents provide the *operator* with the means of effectively deploying the application on top of the acquired resources.

It is important to note that the *acquire* operation does not trigger an automatic deployment of the application on the acquired resources. This is motivated on the fact that deployment process can be long lasting and, depending on the deployment method, nondeterministic.

Besides facilitating the deployment, the returned resource documents allows the *operator* to later destroy the acquired/deployed application.

2.3.3 Documents

2.3.3.1 ApplicationSolution

The *ApplicationSolution* document represents the solution description for one application blueprint received as part of a submitted *ApplicationRequirements* document (as defined in Section 2.1.3.3. As it's name suggests this document is aimed at storing and reporting the information generated by the Cell Management service for a specific application blueprint. The solution documents are stored as results for blueprint requirements in the Catalogue Service.

In the ApplicationSolution document the link between the blueprint and the solution is made via the blueprint document, that is the *BlueprintDefinition*. Also it presents all the services that are used in order to fulfill the blueprint requirements. For achieving this references to the individual service definition and implementation documents are made, that is *ServiceDefinition* and *ImplementationDefinition*.

Figure 2.13 represents a simplified version of such a document. The information contained in this document can be changed throughout the course of the project to match all the requirements.

Together with the other solution documents, this document serves as input for the solution selection process performed by the *operator*. When provided with multiple service implementations the *operator* bases its service implementation selection decision on it.

2.3.3.2 BlueprintSolution

The *BlueprintSolution* is used by the *Cell Manager* to present a solution to a previously sent, by the *operator*, *BlueprintRequirements* document (as discussed in Section 2.1.3.2).

The document does not directly contain the solutions but it references a *ServiceSolution* for each of the requested services. The services for which solutions are provided are nothing else than the *ServiceRequirements* in the original *BlueprintRequirements* document.

Listing 2.14 outlines a simple *BlueprintSolution* document that might be returned by the *Cell Manager*.

```
{
    "type": "object",
    "properties": {
        "links": {
            "type": "object",
            "properties": {
                "blueprints": {
                    "type": "array",
                    "items": {
                        "type": "string",
                        "format": "url"
                    }
                },
                "session": {
                    "type": "string",
                    "format": "url"
                }
            }
        }
    },
    "x-TBD": true
}
```

Figure 2.13: ApplicationSolution Example Schema

2.3.3.3 ServiceSolution

The *ServiceSolution* document is used by the *Cell Manager* to represent a solution corresponding to a *ServiceRequirements* document received as part of an *BlueprintRequirements* document.

Listing 2.15 presents an example *ServiceSolution* document schema.

It is important to note that the *ServiceSolution* document does not effectively contain suitable services but it might reference multiple implementations of a specific service (represented as *ImplementationSolution* documents). These implementations must satisfy the submitted service requirements. In case no suitable implementation is available the list of referenced service implementations will be empty.

2.3.3.4 ImplementationSolution

The *ImplementationSolution* is used to represent a possible implementation of a service requested by the *operator*. This document is the final solution document forwarded to the *operator* through the gateway service. It references an *ImplementationResources* document containing information relevant to the effective service deployment.

The *ImplementationSolution* shares with the *ImplementationResources* a common *Im-*

```
{
    "type": "object",
    "properties": {
        "links": {
            "type": "object",
            "properties": {
                "services": {
                    "type": "array",
                    "items": {
                        "type": "string",
                        "format": "url"
                    }
                },
                "session": {
                    "type": "string",
                    "format": "url"
                }
            }
        }
    },
    "x-TBD": true
}
```

Figure 2.14: BlueprintSolution Example Schema

plementationContext entity. This entity provides to the *Cell Manager* relevant information that is shared with the *Gateway Service* but not forwarded to the *operator*. The content of this *ImplementationContext* is implementation depended and is outside the scope of this document.

Figure 2.16 contains a sketch of an *ImplementationSolution* document that might be exchanged with the *operator* through the gateway service.

2.3.3.5 ApplicationResources

The *ApplicationResources* document is generated by the *Cell Manager* as a result of the ACQUIRE operation (discussed in Section 2.3.2.2). It references one *BlueprintResources* document.

The *ApplicationResources* document serves as input for the effective deployment operation handled by the CloudLightning Deployment component residing besides the resource management subsystem (i.e: *Cell Manager*). Besides a reference to the *BlueprintResources* document the content of the *ApplicationResources* is opaque to the *operator*.

In Figure 2.17 we present an example *ApplicationResources* document schema.

```
{
    "type": "object",
    "properties": {
        "links": {
            "type": "object",
            "properties": {
                "implementations": {
                    "type": "array",
                    "items": {
                        "type": "string",
                        "format": "url"
                    }
                },
                "session": {
                    "type": "string",
                    "format": "url"
                }
            }
        }
    },
    "x-TBD": true
}
```

Figure 2.15: ServiceSolution Example Schema

```
{
    "type": "object",
    "properties": {
        "links": {
            "type": "object",
            "properties": {
                "session": {
                    "type": "string",
                    "format": "url"
                }
            }
        }
    },
    "x-TBD": true
}
```

Figure 2.16: ImplementationSolution Example Schema

2.3.3.6 BlueprintResources

This document is generated by the *Cell Manager* as a result of the ACQUIRE operation (discussed in Section 2.3.2.2). It references one or more *ImplementationResources*

```
{
    "type": "object",
    "properties": {
        "links": {
            "type": "object",
            "properties": {
                "blueprint": {
                    "type": "string",
                    "format": "url"
                },
                "session": {
                    "type": "string",
                    "format": "url"
                }
            }
        }
    },
    "x-TBD": true
}
```

Figure 2.17: ApplicationResources Example Schema

documents.

```
{
    "type": "object",
    "x-TBD": true,
    "payload": {},
    "comment": "Aditional Blueprint level data wrapped as specified by
↪ the SDL deliverable"
}
```

Figure 2.18: BlueprintResources Example Schema

In Figure 2.18 we present an example *BlueprintResources* document.

As previously mentioned for *ApplicationResources*, the *BlueprintResources* document serves as input for the *CloudLightning* deployment component and it is used for managing the resources at the blueprint level. It shares the same *BlueprintContext*, containing opaque shared data, with the *BlueprintSolution*.

2.3.3.7 ImplementationResources

Similarly to the *ApplicationResources* and *BlueprintResources* document, the *ImplementationResources* document is generated by the *Cell Manager* as part of the *acquire* operation.

```
{
    "type": "object",
    "properties": {
        "links": {
            "type": "object",
            "properties": {
                "implementations": {
                    "type": "array",
                    "items": {
                        "type": "string",
                        "format": "url"
                    }
                },
                "session": {
                    "type": "string",
                    "format": "url"
                }
            }
        }
    },
    "x-TBD": true
}
```

Figure 2.19: ImplementationResources Example Schema

In Figure 2.19 we present an example *ImplementationResources* document.

ImplementationResources document serves as input for various subcomponents of the *CloudLightning* deployment component. It can be used as a handle for controlling deployment and subsequent management of the deployed resources. Depending on the implementation, the management operations can include: resource restart, re-deployment, profiling, etc.

As mentioned for the *ImplementationSolution* (discussed in Section 2.3.3.4) it shares with it a common *ImplementationContext* document containing opaque shared data.

Chapter 3

Conclusions

To summarize, in this document we presented a general overview of the CloudLightning architecture from the perspective of the messages that are exchanged between the various components. Alongside we also present a potential API that provides an abstraction over the capabilities offered by the underlying CloudLightning system. The proposed API's are designed in such a way that they can be easily extended as the project evolves in time, particularly in the remaining WP4 tasks and also in WP2, WP5 and WP6. Of particular importance ar the future contributions of WP5, contributions that will fill the abstract definitions provided in this deliverable for the documents discussed in Section 2.2.

Alongside with the proposed API's we also documented the major interactions between different logical components of the CloudLightning system. For each of the interactions between the core components the documents that are transmitted in order to facilitate communications that are described and exemplified.

This document is meant to be the foundation work for the framework that is going to be developed in order to successfully support the entire CloudLightning infrastructure. Although the details for each of the messages are not finalized in this document, it still represents a good starting point for fixing them and think more deeply about how and through which means different components communicate with each other.

Bibliography

[1] L. A. Barroso, J. Clidaras, and U. Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis lectures on computer architecture*, 8(3):1–154, 2013.

[2] P. F. Brown and R. M. B. A. Hamilton. Reference model for service oriented architecture 1.0, 2006.

[3] R. Fielding and R. Taylor. Principled design of the modern Web architecture. *Proceedings of the 2000 International Conference on Software Engineering. ICSE 2000 the New Millennium*, 2(2):115–150, 2000.

[4] J. Frey. Condor dagman: Handling inter-job dependencies. *University of Wisconsin, Dept. of Computer Science, Tech. Rep*, 2002.

[5] J. L. Hennessy and D. A. Patterson. *Computer Architecture, Fifth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5th edition, 2011.

[6] T. B. Lee. Cool URIs don't change. 1998.

[7] C. Pautasso. Rest vs. soap: Making the right architectural decision. In *SOA Symposium*, pages 2009–01.

[8] D. Petcu. On autonomic hpc clouds. *Proceedings of 2nd International Workshop on Sustainable Ultrascale Computing Systems*, pages 29–40, 2015.

[9] L. Richardson and S. Ruby. *Restful Web Services*. O'Reilly, first edition, 2007.

[10] M. Solomon. The classad language reference manual, version 2.1. *Computer Sciences Department, University of Wisconsin, Madison, WI, USA*, 2003.

[11] O. TOSCA. Topology and orchestration specification for cloud applications (tosca) primer version 1.0, 2013.

Appendix A

Swagger API Definitions

This appendix contains the Swagger schema and documentation for the proposed APIs. As previously mentioned this APIs are expected to change together with the project evolution.

The printed documentation contained in this appendix is complemented by an up-to-date online documentation available on: `https://cloudlightning-docs-gw-api.herokuapp.com`

CloudLightning Gateway API

Default request content-types: application/json
Default response content-types: application/json
Schemes: http, https

Summary

Path	Operation	Description
/definitions/blueprints	GET	
/definitions/blueprints/{identifier}	GET	
	PUT	
/definitions/implementations	GET	
/definitions/implementations/{identifier}	GET	
	PUT	
/definitions/services	GET	
/definitions/services/{identifier}	GET	
	PUT	
/operations/acquire	POST	
/operations/deploy	POST	
/operations/destroy	POST	
/operations/require	POST	
/requirements/applications/{identifier}	GET	
	PUT	
/requirements/blueprints/{identifier}	GET	
	PUT	
/requirements/services/{identifier}	GET	
	PUT	
/resources/applications/{identifier}	GET	
	PUT	
/resources/blueprints/{identifier}	GET	
	PUT	
/resources/implementation/{identifier}	GET	
	PUT	
/sessions	POST	
/sessions/{session}	GET	
/solutions/applications/{identifier}	GET	
	PUT	
/solutions/blueprints/{identifier}	GET	
	PUT	
/solutions/implementations/{identifier}	GET	
	PUT	

/solutions/services/{identifier}	GET	
	PUT	

Paths

GET /definitions/blueprints

DESCRIPTION

RESPONSES

Uses default content-types: **application/json**

200 OK

...

DocumentsList

GET /definitions/blueprints/{identifier}

DESCRIPTION

REQUEST PARAMETERS

Name	Description	Type	Data type	
identifier		path	*string* (identifier)	**required**

RESPONSES

Uses default content-types: **application/json**

200 OK

...

BlueprintDefinitionDocument

PUT /definitions/blueprints/{identifier}

DESCRIPTION

REQUEST BODY

Uses default content-types: **application/json**

BlueprintDefinitionDocument

REQUEST PARAMETERS

Name	Description	Type	Data type

identifier	path	*string* (identifier)	**required**

RESPONSES

Uses default content-types: **application/json**

201 Created
Succeeded

GET /definitions/implementations

DESCRIPTION

RESPONSES

Uses default content-types: **application/json**

200 OK
...

DocumentsList

GET /definitions/implementations/{identifier}

DESCRIPTION

REQUEST PARAMETERS

Name	Description	Type	Data type	
identifier		path	*string* (identifier)	**required**

RESPONSES

Uses default content-types: **application/json**

200 OK
...

ImplementationDefinitionDocument

PUT /definitions/implementations/{identifier}

DESCRIPTION

REQUEST BODY

Uses default content-types: **application/json**

ImplementationDefinitionDocument

REQUEST PARAMETERS

Name	Description	Type	Data type	
identifier		path	*string* (identifier)	**required**

RESPONSES

Uses default content-types: **application/json**

201 Created
Succeeded

GET /definitions/services

DESCRIPTION

RESPONSES

Uses default content-types: **application/json**

200 OK
...

DocumentsList

GET /definitions/services/{identifier}

DESCRIPTION

REQUEST PARAMETERS

Name	Description	Type	Data type	
identifier		path	*string* (identifier)	**required**

RESPONSES

Uses default content-types: **application/json**

200 OK
...

ServiceDefinitionDocument

PUT /definitions/services/{identifier}

DESCRIPTION

REQUEST BODY

Uses default content-types: **application/json**

ServiceDefinitionDocument

REQUEST PARAMETERS

Name	Description	Type	Data type	
identifier		path	*string* (identifier)	**required**

RESPONSES

Uses default content-types: **application/json**

201 Created
Succeeded

POST /operations/acquire

DESCRIPTION

REQUEST BODY

Uses default content-types: **application/json**

AcquireOperationInput

RESPONSES

Uses default content-types: **application/json**

200 OK
...

AcquireOperationOutput

POST /operations/deploy

DESCRIPTION

REQUEST BODY

Uses default content-types: **application/json**

DeployOperationInput

RESPONSES

Uses default content-types: **application/json**

200 OK

...

DeployOperationOutput

POST /operations/destroy

DESCRIPTION

REQUEST BODY

Uses default content-types: **application/json**

DestroyOperationInput

RESPONSES

Uses default content-types: **application/json**

200 OK

...

DestroyOperationOutput

POST /operations/require

DESCRIPTION

REQUEST BODY

Uses default content-types: **application/json**

RequireOperationInput

RESPONSES

Uses default content-types: **application/json**

200 OK

...

RequireOperationOutput

GET /requirements/applications/{identifier}

DESCRIPTION

REQUEST PARAMETERS

Name	Description	Type	Data type	
identifier		path	*string* (identifier)	**required**

RESPONSES

Uses default content-types: **application/json**

200 OK

...

ApplicationRequirementDocument

PUT /requirements/applications/{identifier}

DESCRIPTION

REQUEST BODY

Uses default content-types: **application/json**

ApplicationRequirementDocument

REQUEST PARAMETERS

Name	Description	Type	Data type	
identifier		path	*string* (identifier)	**required**

RESPONSES

Uses default content-types: **application/json**

201 Created

Succeeded

GET /requirements/blueprints/{identifier}

DESCRIPTION

REQUEST PARAMETERS

Name	Description	Type	Data type	
identifier		path	*string* (identifier)	**required**

RESPONSES

Uses default content-types: **application/json**

200 OK

...

BlueprintRequirementDocument

PUT /requirements/blueprints/{identifier}

DESCRIPTION

REQUEST BODY

Uses default content-types: **application/json**

BlueprintRequirementDocument

REQUEST PARAMETERS

Name	Description	Type	Data type	
identifier		path	*string* (identifier)	**required**

RESPONSES

Uses default content-types: **application/json**

201 Created
Succeeded

GET /requirements/services/{identifier}

DESCRIPTION

REQUEST PARAMETERS

Name	Description	Type	Data type	
identifier		path	*string* (identifier)	**required**

RESPONSES

Uses default content-types: **application/json**

200 OK
...

ServiceRequirementDocument

PUT /requirements/services/{identifier}

DESCRIPTION

REQUEST BODY

Uses default content-types: **application/json**

ServiceRequirementDocument

REQUEST PARAMETERS

Name	Description	Type	Data type	
identifier		path	*string* (identifier)	**required**

RESPONSES

Uses default content-types: **application/json**

201 Created
Succeeded

GET /resources/applications/{identifier}

DESCRIPTION

REQUEST PARAMETERS

Name	Description	Type	Data type	
identifier		path	*string* (identifier)	**required**

RESPONSES

Uses default content-types: **application/json**

200 OK
...

ApplicationResourceDocument

PUT /resources/applications/{identifier}

DESCRIPTION

REQUEST BODY

Uses default content-types: **application/json**

ApplicationResourceDocument

REQUEST PARAMETERS

Name	Description	Type	Data type	
identifier		path	*string* (identifier)	**required**

RESPONSES

Uses default content-types: **application/json**

201 Created
Succeeded

GET /resources/blueprints/{identifier}

DESCRIPTION

REQUEST PARAMETERS

Name	Description	Type	Data type	
identifier		path	*string* (identifier)	**required**

RESPONSES

Uses default content-types: **application/json**

200 OK
...

BlueprintResourceDocument

PUT /resources/blueprints/{identifier}

DESCRIPTION

REQUEST BODY

Uses default content-types: **application/json**

BlueprintResourceDocument

REQUEST PARAMETERS

Name	Description	Type	Data type	
identifier		path	*string* (identifier)	**required**

RESPONSES

Uses default content-types: **application/json**

201 Created
Succeeded

GET /resources/implementation/{identifier}

DESCRIPTION

REQUEST PARAMETERS

Name	Description	Type	Data type	
identifier		path	*string* (identifier)	**required**

RESPONSES

Uses default content-types: **application/json**

200 OK

...

ImplementationResourceDocument

PUT /resources/implementation/{identifier}

DESCRIPTION

REQUEST BODY

Uses default content-types: **application/json**

ImplementationResourceDocument

REQUEST PARAMETERS

Name	Description	Type	Data type	
identifier		path	*string* (identifier)	**required**

RESPONSES

Uses default content-types: **application/json**

201 Created

Succeeded

POST /sessions

DESCRIPTION

REQUEST BODY

Uses default content-types: **application/json**

SessionCreateInput

RESPONSES

Uses default content-types: **application/json**

201 Created

Redirect toward newly created document

Header	Description	Data type
Location		*string* (url)

GET /sessions/{session}

DESCRIPTION

REQUEST PARAMETERS

Name	Description	Type	Data type	
session		path	*string* (identifier)	**required**

RESPONSES

Uses default content-types: **application/json**

200 OK

...

SessionDocument

GET /solutions/applications/{identifier}

DESCRIPTION

REQUEST PARAMETERS

Name	Description	Type	Data type	
identifier		path	*string* (identifier)	**required**

RESPONSES

Uses default content-types: **application/json**

200 OK

...

ApplicationSolutionDocument

PUT /solutions/applications/{identifier}

DESCRIPTION

REQUEST BODY

Uses default content-types: **application/json**

ApplicationSolutionDocument

REQUEST PARAMETERS

Name	Description	Type	Data type	
identifier		path	*string* (identifier)	**required**

RESPONSES

Uses default content-types: **application/json**

201 Created
Succeeded

GET /solutions/blueprints/{identifier}

DESCRIPTION

REQUEST PARAMETERS

Name	Description	Type	Data type	
identifier		path	*string* (identifier)	**required**

RESPONSES

Uses default content-types: **application/json**

200 OK
...

BlueprintSolutionDocument

PUT /solutions/blueprints/{identifier}

DESCRIPTION

REQUEST BODY

Uses default content-types: **application/json**

BlueprintSolutionDocument

REQUEST PARAMETERS

Name	Description	Type	Data type	
identifier		path	*string* (identifier)	**required**

RESPONSES

Uses default content-types: **application/json**

201 Created
Succeeded

GET /solutions/implementations/{identifier}

DESCRIPTION

REQUEST PARAMETERS

Name	Description	Type	Data type	
identifier		path	*string* (identifier)	**required**

RESPONSES

Uses default content-types: **application/json**

200 OK
...

ImplementationSolutionDocument

PUT /solutions/implementations/{identifier}

DESCRIPTION

REQUEST BODY

Uses default content-types: **application/json**

ImplementationSolutionDocument

REQUEST PARAMETERS

Name	Description	Type	Data type	
identifier		path	*string* (identifier)	**required**

RESPONSES

Uses default content-types: `application/json`

201 Created
Succeeded

GET /solutions/services/{identifier}

DESCRIPTION

REQUEST PARAMETERS

Name	Description	Type	Data type	
identifier		path	*string* (identifier)	**required**

RESPONSES

Uses default content-types: `application/json`

200 OK
...

ServiceSolutionDocument

PUT /solutions/services/{identifier}

DESCRIPTION

REQUEST BODY

Uses default content-types: `application/json`

ServiceSolutionDocument

REQUEST PARAMETERS

Name	Description	Type	Data type	
identifier		path	*string* (identifier)	**required**

RESPONSES

Uses default content-types: `application/json`

201 Created
Succeeded

Parameter definitions

Key	Name	Description	Type	Data type	
BlueprintDefinitionPutRequest	request		body	*object*	**required**
ServiceDefinitionPutRequest	request		body	*object*	**required**
ImplementationDefinitionPutRequest	request		body	*object*	**required**
ApplicationRequirementPutRequest	request		body	*object*	**required**
BlueprintRequirementPutRequest	request		body	*object*	**required**
ServiceRequirementPutRequest	request		body	*object*	**required**
ApplicationSolutionPutRequest	request		body	*object*	**required**
BlueprintSolutionPutRequest	request		body	*object*	**required**
ServiceSolutionPutRequest	request		body	*object*	**required**
ImplementationSolutionPutRequest	request		body	*object*	**required**
ApplicationResourcePutRequest	request		body	*object*	**required**
BlueprintResourcePutRequest	request		body	*object*	**required**
ImplementationResourcePutRequest	request		body	*object*	**required**
RequireOperationPostRequest	request		body	*object*	**required**
AcquireOperationPostRequest	request		body	*object*	**required**
DeployOperationPostRequest	request		body	*object*	**required**
DestroyOperationPostRequest	request		body	*object*	**required**
DocumentIdentifierPathElement	identifier		path	*string* (identifier)	**required**
SessionIdentifierPathElement	session		path	*string* (identifier)	**required**
SessionsPostRequest	request		body	*object*	**required**

Response definitions

AcquireOperationPostResponse

...

AcquireOperationOutput

ApplicationRequirementGetResponse

...

ApplicationRequirementDocument

ApplicationResourceGetResponse

...

ApplicationResourceDocument

ApplicationSolutionGetResponse

...

ApplicationSolutionDocument

BlueprintDefinitionGetResponse

...

BlueprintDefinitionDocument

BlueprintRequirementGetResponse

...

BlueprintRequirementDocument

BlueprintResourceGetResponse

...

BlueprintResourceDocument

BlueprintSolutionGetResponse

...

BlueprintSolutionDocument

DeployOperationPostResponse

...

DeployOperationOutput

DestroyOperationPostResponse

...

DestroyOperationOutput

DocumentsGetResponse

...

DocumentsList

GenericPost201Response

Redirect toward newly created document

Header	Description	Data type
Location		*string* (url)

GenericPut201Response

Succeeded

ImplementationDefinitionGetResponse

...

ImplementationDefinitionDocument

ImplementationResourceGetResponse

...

ImplementationResourceDocument

ImplementationSolutionGetResponse

...

ImplementationSolutionDocument

RequireOperationPostResponse

...

RequireOperationOutput

ServiceDefinitionGetResponse

...

ServiceDefinitionDocument

ServiceRequirementGetResponse

...

ServiceRequirementDocument

ServiceSolutionGetResponse

...

ServiceSolutionDocument

SessionGetResponse

...

SessionDocument

Schema definitions

AcquireOperationInput: *object*

AcquireOperationOutput: *object*

ApplicationRequirementDocument: *object*

ApplicationResourceDocument: *object*

ApplicationSolutionDocument: *object*

BlueprintDefinitionDocument: *object*

BlueprintRequirementDocument: *object*

BlueprintResourceDocument: *object*

BlueprintSolutionDocument: *object*

DeployOperationInput: *object*

DeployOperationOutput: *object*

DestroyOperationInput: *object*

DestroyOperationOutput: *object*

DocumentsList: *object*

ImplementationDefinitionDocument: *object*

ImplementationResourceDocument: *object*

ImplementationSolutionDocument: *object*

RequireOperationInput: *object*

RequireOperationOutput: *object*

ServiceDefinitionDocument: *object*

ServiceRequirementDocument: *object*

ServiceSolutionDocument: *object*

SessionCreateInput: *object*

SessionDocument: *object*