HORIZON2020 FRAMEWORK PROGRAMME

TOPIC EUK-03-2016

"Federated Cloud resource brokerage for mobile cloud services"

D3.5
BASMATI Server- and Client-side Applications Adaptation and Reconfiguration: Design and Specification

Project acronym: BASMATI

Project full title: *Cloud Brokerage Across Borders for Mobile Users and Applications*

Contract no.: 723131

Workpackage:	WP3	Users and Applications Modelling and Analysis
Editor:	Emanuele Carlini	CNR
Author(s):	Patrizio Dazzi, Emanuele Carlini, Vinicius Monterio de Lira, Cristina Munteanu	CNR
Authorized by	K. Tserpes	ICCS
Doc Ref:	D3.5	
Reviewer	K. Tserpes	ICCS
Dissemination Level	Public	

© All Rights Reserved

Document History

Version	Date	Changes	Author/Affiliation
v.0.1	05-10-2016	Initial ToC	Patrizio Dazzi, CNR
v.0.2	06-12-2016	Revised ToC	Patrizio Dazzi, CNR
v.0.3	28-01-2018	BEAM description	Jamie Marshall, AMEN
v.0.4	29-01-2018	Template agreement	Enric Pages, ATOS
v.0.5	30-01-2018	Application Repository and Decision Maker	Vinicius Monteiro de Lira, Cristina Munteanu, CNR
v 1.0	05-02-2018	Added the remaining content and finalized the document	Emanuele Carlini, CNR

© All Rights Reserved

BASMATI Glossary

Term/Acronym	Definition
Mobile cloud services	Online services offered by cloud resources to support mobile apps. The backend of the mobile apps.
CP	Cloud Provider. The actor that provides the cloud infrastructure/resources, such as VMs
CSP	Cloud Service Provider. The actor that provides cloud services on top of a rent infrastructure from a CP
Cloudlet	Limited capacity infrastructures with virtualization capabilities, often used to support a limited amount of users or perform a limited set of operations on behalf of the central cloud infrastructure that hosts the complete application
Edge resources	Resources aimed to operate specialized functionality, located at the "edge" of the network infrastructure, thus, closer to the end users. Examples are (clusters of) RaspberryPis or cloudlets
KE	Knowledge Extractor
DM	Decision Maker
RB	Resource Broker
MVD	Mobile Virtual Desktop
DASFEST	An 3-day long music festival taking place in Karlsruhe, Germany every July
Tripbuilder	The Tripbuilder application, providing personalized sightseeing tours given the available time for the visit
ACE	Amenesik Cloud Engine. The cloud service deployment tool through which actual federation is achieved
BEAM	BASMATI Enhanced Application Model. An extension of the TOSCA specification
ASP	Application Service Provider. A Federation user that rents resource services in order to provide an Application services to End-users
Brokering	The matchmaking support provided by BASMATI platform to decide about the best cloud resources to exploit for the execution of the back-end of BASMATI applications. This activity regards the placement of the services or data on computational resources and storages belonging to the cloud data centre and the cloudlets within the federation.
End user	A user who benefits the various application and infrastructure services provided by the Cloud. Within BASMATI, the most typical example is exploiting the Cloud federation via a mobile device (possibly a laptop) using specialized apps or a web browser.
Offloading	The ability of BASMATI platform supporting the runtime placement of the components composing the front-end of BASMATI applications on edge resources available nearby the end user. This activity takes place both when edge and mobiles exchange one each other their own workload or when such devices transfer some workload to the clouds or cloudlets. In BASMATI we often distinguish Front-end offloading, related to the mobile part of application, from Back-end offloading, concerning the server side of applications. The latter

© All Rights Reserved

	roughly translates to the known concept of Cloudbursting.
QoE	Quality of experience. It is a measure of a customer's experiences with a service. It may be related to some aspects of the QoS and QoP, but can also take into account other metrics.
Service handover	Service handover refers to the activity of transferring an active service between two computational resources (e.g. Cloudlets) with minimal or no disruption on the availability of the service. Ideally, service handover is transparent with respect to the user.
Situational Awareness	The ability of the BASMATI platform to recognise the “situation” characterising the actual combined status of users, applications and resources, aimed at achieving an effective and efficient management of applications and resources.

© All Rights Reserved

Executive Summary

This report provides a description of the mechanisms, tools, and algorithms used to support application adaptation and reconfiguration in the BASMATI brokerage platform. At the core of this support lies the BASMATI Enriched Application Model (BEAM), which is the xml-based language in which an application is modelled and represented in BASMATI. The design principles behind the BEAM (namel: compatibility, extensibility, decomposability) are the prerequisites to provide efficient and effective geo-placement of services and applications on top of federated Cloud resources. The BEAM is made available to all the components of the platform by the Application Repository, which works as a centralization point for the BEAMs of all the applications. The decomposability of BEAM is exploited by the Decision Maker that has the task to proactively and reactively adapt the application according to the behaviour of users and resources, by means of advanced placement algorithms.

© All Rights Reserved

Table of Contents

Executive Summary v

1 Introduction and Background 1

1.1 Relationship with other Deliverables 1

1.2 Deliverable Outline 2

2 BASMATI Application Modelling 2

2.1 Applications decomposition modelling 3

2.2 BASMATI Enriched Application Model (B.E.A.M.) 4

2.2.1 Application Topology 5

2.2.2 Functional Requirements 12

2.2.3 Template Agreement 12

2.2.4 Decompostion, Selection and Deployment documents 14

2.3 Application Repository 15

3 Adaptivity in BASMATI 17

3.1 Decison Maker 18

4 Further Developments for BASMATI 21

References 21

Appendix A 22

Topology Template - DASFEST APPLICATION 22

Topology Template - TRIPBUILDER APPLICATION 26

© All Rights Reserved

1 Introduction and Background

The BASMATI project aims at the realization of an innovative brokerage platform targeting a scalable management of heterogeneous distributed and federated resources. The motivation behind is to support mobile cloud applications in demanding scenarios. Specifically, the project focuses on the so called "nomadic" users, which comprises the travelling users that interact with cloud applications across different geoposition around the world. Therefore, the BASMATI brokerage platform faces the challenge of coping with the inherent unpredictability of human mobility and the corresponded demand behaviour of the cloud application.

BASMATI tackles the above challenge by applying a wide research and technological improvements strategies, which includes user demands prediction and application orchestration in Cloud Federation. In this deliverable, we describe the advancement and innovation approach proposed by BASMATI in two specific requirement contexts: (1) application composition representation; (2) and adaptive mechanisms for application placement. To accomplish the first requirement, a flexible and expandable definition of the application is a fundamental prerequisite for any complex brokering platform and BASMATI is no exception. BASMATI defines the BASMATI Enriched Application Model (BEAM), a TOSCA-based application model that combines expressivity and portability of multi-services application. Secondly, when dealing with application placement, especial mechanisms and solutions for adaptation of applications at runtime are needed. Such support is a key enabler in a widely distributed environment to enable dynamic migration for cloud applications aiming to keep their proper operation performance. This is particularly evident when the conditions of the platform changes and when users move around.

This report describes the structure, the deployment schemas and the management methodologies of the applications executed on the BASMATI brokerage platform, focusing on their support to adaptation and reconfiguration.

1.1 Relationship with other Deliverables

This deliverable provides the design and specification for apdativity and reconfiguration mechanisms, as well as an overview of the application modelling in BASMATI. Therefore, it has connection with the following other deliverables:

- Deliverable 2.3, which is related to the overall BASMATI Architecture. In particular, D2.3 poses the basis for the application modelling, identifying the application as collection of services. This deliverable provide a full description of the BEAM, which builds upon those initial definitions.

© All Rights Reserved

- Deliverable 4.3, which presents the structure of the BASMATI service placement including the decision maker component, which practically implements the adaptivity concepts described in this deliverable.

1.2 Deliverable Outline

The outline of the deliverable summarized the concepts that drive the innovation steps of BASMATI in terms of application adaptivity and modelling. The deliverable is organized as the following. In Section 2 we present the design principles and the specification of the BASMATI Enhanced Application Model, as well as a description of the Application Respository component. Section 3 focuses the adaptivity in BASMATI and presents an overview of the Decision Maker. Finally Section 4 discusses further and ongoing development in terms of adaptivity and reconfiguration that are planned before the end of the project.

2 BASMATI Application Modelling

One of the most important challenges for the realization of the BASMATI brokerage platform is the realization of an application modelling that allows an effective answer toward the restructuring of the application backend to support user mobility. The design of the BASMATI's application model adheres with the following considerations:

- simple and compatible for the application provider: the definition of a proper application model comes with the necessity to be compliant with application not written specifically for BASMATI. In fact, the application provider should not be forced to specifically rewrite pieces of its own application code to be run on the BASMATI platform. In other words, the idea behind the BASMATI platform is to support what other cloud platforms support with minimal effort.

- expandable: in order to cope with the mobility of the users, and able to apply specific optimization for the application placement, the "static" definition of application has to be enhanced with addition description that describe the dynamic behaviour of the application.

- decomposable: application as a combination of services, rather than a monolithic block. This would allow to develop placement models and mechanisms that optimize the placement of applications in terms of services, allowing for service decoupling and replication.

These considerations, are the basis for the concept of BASMATI Enriched Application Model (BEAM). The BEAM specification is a derivative work that is based on the international standard known as Topology and Orchestration Specification for Cloud Applications (TOSCA) (Binz, Breitenbücher, Kopp, & Leymann, 2013). TOSCA is an open-source XML-based language that

© All Rights Reserved

describe the relationships and dependencies between services and applications running on cloud computing platforms, in a way which is independent of the specific cloud platform considered. BEAM provides a simplification of TOSCA suitable for the management and deployment of mobile applications in a Federated Cloud Scenario. The standard specification of TOSCA may be retrieved and consulted via the following link:

http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.pdf

2.1 Applications decomposition modelling

From an high level perspective, in BASMATI the application is seen as a collection of *N* components, or services, that collaborate toward a unified results. In other words, a component represents what can be defined as a service, which realizes a functionality within the workflow of the application. The deployment of the application is organized at the component level, meaning that the components belonging to the same application can in principle be allocated on different federated cloud resources, unless specific customer constraints.

However, in order to efficiently orchestrate the deployment, we have perform an additional step and organize the components into *partitions*. The relationship between components and partition is the following: each component belongs to only one partition, and each partition can have $1 \leq K \leq N$ components, where *N* is the total number of components of an application. A visualization of these above relationships is depicted in Figure 1.

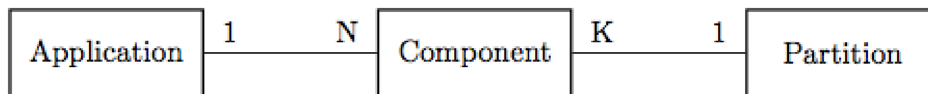

Figure 1 Composition of an application

In order to clarify the above concept let us consider an example. The application in Figure 2 is composed by N = 3 components: (A) a web server interface, used to connect to the application; (B) a business process that elaborates the requests coming from the interface; (C) a database that stores data needed to the business process. The components are organized in two partitions: partition (1) that encapsulates the component A (K = 1); partition (2) that encapsulates components B and C (K = 2).

© All Rights Reserved

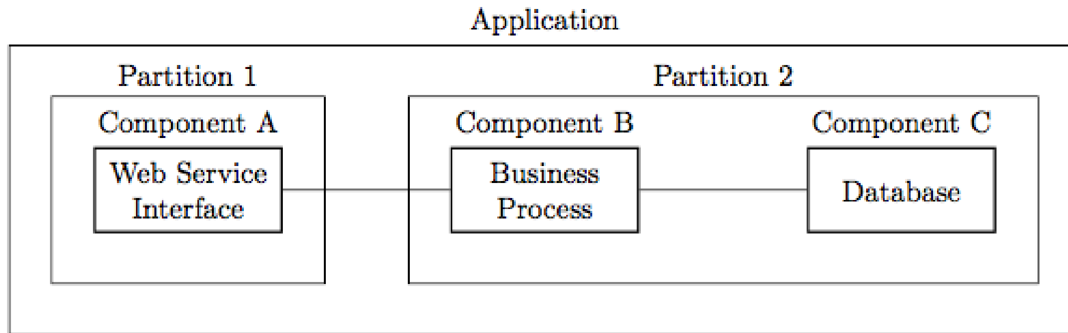

Figure 2 Example of an application composition

Such decomposition permits to consider the single services as the unit of placement. This allows advanced placement models to consider specific features:

- the ability to optimize the placement for the single service, while keeping valid the constraints on the whole application;
- minimize the disruption in case of a partial migrations, as only the problematic services can moved;
- application of optimization in the workflow of the application (e.g. services replication)

An example of such advanced placement models is the one implemented in the Decision Maker, whose general description is provided in this documents, while a full description of its placement algorithms is presented in Deliverable 4.3.

2.2 BASMATI Enriched Application Model (B.E.A.M.)

The previous representation provides a model of the application in terms of functional features of the application. In addition, BASMATI extends such model with other pieces of information that drives the placement of the application in cloud resources. We call this extended model the Basmati Enhance Application Model (BEAM).

Therefore, this section provides a technical reference describing the BEAM. From a practical point of view, a BEAM is a collection of information, materialized as documents, that describes the application as a collection of services and defines hints on the resources its deployment.

Additionally, the BEAM specification defines non-disruptive extensions to the simplified TOSCA subset, for the management of the Service Life Cycle, in line with the semantics defined by the international standard, known as WS Agreement. The standard specification of WS Agreement may be retrieved and consulted via the following link:

```
https://www.ogf.org/documents/GFD.107.pdf
```

© All Rights Reserved

Reference will be made, during the description of BEAM in this document, to the international standard, OCCI, the specification of which may be retrieved and consulted via the following link:

http://occi-wg.org/about/specification/

From a practical point of view, a BEAM is a collection of information, materialized as documents, which describe the application as a collection of services and defines hints on the resources its placement and deployment. The complete list of the documents that compose the BEAM are summarized in the Table below.

Name	***Format***	***Purpose***
ID	128-bit UUID	Identifies the instance of an application within the BASMATI platform
Application Topology	TOSCA	Describes the application in terms of components and relationships between them
Functional Requirements	TOSCA	Describes the functional requirements of each component of the application
Template Agreement	ws-agreement	An agreement between a service consumer and service provider specifies one or more service level objectives
Decomposition Document	XML	Defines the partitions of the applications and the degree of replication for each partition
Selection Document	XML	Provides a ranking of cloud resources for each partition identified in the decomposition document
Deployment Document	XML	Identifies the mapping between application partition and resources of the federated environment

2.2.1 Application Topology

The topology of the application is defined by a document written in TOSCA that describes the application in terms of services and connection among them. TOSCA describes the structure of composite applications as topologies describing their **components** and their **relationships**.

The connotation of such components and relations is defined by typing the aforementioned Node Templates and Relationship Templates by means of **Node Types** and **Relationship Types**, respectively. Relations may be, for example, one component is “hosted on”, “depends on”, or “communicates with” another component.

© All Rights Reserved

The Application Topology comprises two distinct sections:

- The importation of standard and document specific definitions
  This section is used for the inclusion or import of the documents containing the precise Node Type and Node Implementation definition on which the Service Template is based.
- The Service Template and its sub descriptions.
  This section describes the details of the named service making use of the Node Type and Implementations included in the preceding section.

Figure 4 illustrates the structure of the Toplogy Templace in TOSCA.

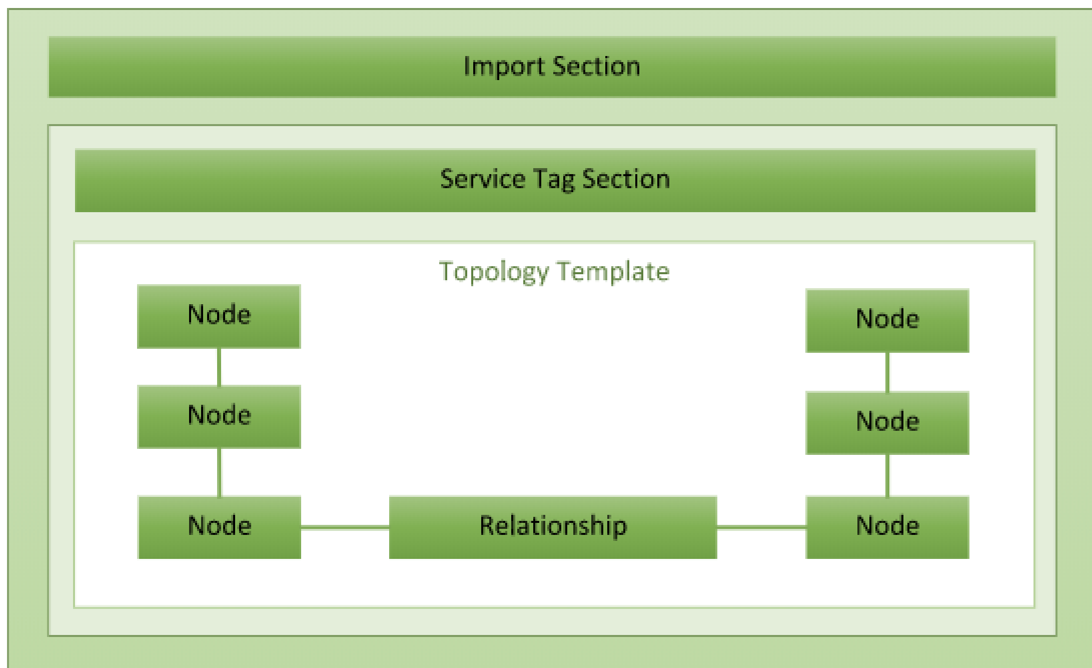

Figure 4. Topology Template in TOSCA format

The **Topology Template** of the Service Template describes the collections of virtual machines required for the composition of the service to be delivered to the customer. Each virtual machine can be described by several layered node templates with associated **Node Types** and **Node Implementations**. At the bottom of the stack we find the standardised COMPUTE Node Templates with their typical IAAS profiles. Layered on top of this we will find MIDDLEWARE Node Templates describing standard software packages, their installation and usage. Finally, on top we will have the CONFIGURATION **Node Templates** providing application usage specific configuration of the middle ware and hardware layers. This hierarchy will be repeated for each of the virtual machines required for the service. Finally, a collection of **Relationship Templates** will dictate the topology of the interconnections between the various compute nodes. For the

© All Rights Reserved

scope of the BEAM, the following TOSCA elements are used to define the Topology Template. Further description can be find through the following link at the TOSCA specification: https://www.oasis-open.org/committees/tosca/

a) **Node Template.** The Node Template allows description of hardware, middleware and software configuration nodes through the following attribute and child elements. As depicted in the diagram showing the overall architecture of the BEAM Document, Node Templates will be both vertically layered through internode requirement and capability matching and horizontally linked through internode Relationship Templates. The attributes and elements will first be presented individually and then the examples that follow will explain the precise semantics and usage.

b) **Node Types**. These elements are used to define the interface characteristics of both the nodes defined by the TOSCA standardisation and the nodes required for the description of the applications requirements. Many different, alternative, interface definitions may be provided for individual NodeType elements, corresponding to different realisation and deployment management tools. The specifications here correspond to the utilisation of the AMENESIK Cloud Engine, known as ACE, responsible for the deployment and management of federated cloud deployment for the BASMATI Platform.

c) **Node Type Implementation.** These elements are used to define the precise deployment characteristics of the operations of the interfaces of the node types defined by the TOSCA standardisation and the nodes required for the description of the applications requirements. Many different, alternative, node type implementations may be defined for a set of NodeType elements, corresponding to different realisation and deployment management tools. The following attributes and child elements are defined for NodeTypeImplementation elements.

d) **Relationship Template.** The Relationship Template element of the Topology Template element is to be used to specify the horizontal relationship links between Node Template elements in the same Topology Template.

Application Topology Definition Steps

In this section we walk through the steps of creating a simple Topology Template. The idea is to facilitate the understanding of the Topology Template Creating for the BASMATI use cases.

Public
© All Rights Reserved

Step 1: The Service Template Definition

The first step requires the creation of the BEAM/TOSCA document and its basic sections and especially **Import** element and the **ServiceTemplate** element with its **Tags** and **TopologyTemplate** child elements. The **Import** element will fetch the default definitions for the example and the **ServiceTemplate** will be named to represent the Application and it's intended purpose.

```
<Definitions>
        <Import location="example-tosca-defaults.xml"/>
        <ServiceTemplate name="LoadBalancedWebServerExample">
                <Tags/>
                <TopologyTemplate/>
        </ServiceTemplate>
</Definitions>
```

Step 2: The Service Context

We can now expand the **Tags** element to add information about the application subject and the application author and date.

```
        <Tags>
                <Tag name="Title" value="BASMATI BEAM EXAMPLE 4"/>
                <Tag name="SubTitle" value="Load Balanced Web Server with Database"/>
                <Tag name="Version" value="1.0a.01"/>
                <Tag name="Author" value="Iain James Marshall"/>
                <Tag name="Date" value="25th November 2017"/>
        </Tags>
```

Step 3: The Node Templates and Relationships

In this step we will expand the **TopologyTemplate** by adding the **NodeTemplate** elements for the four compute nodes and the four **RelationshipTemplate** elements describing the **"hostname"** relationships between them. The names of the NodeTemplate elements and RelationshipTemplate elements may be chosen freely to represent your system as you wish.

```
                <NodeTemplate name="db" type="tosca.nodes.Compute"/>
                <NodeTemplate name="ws1" type="tosca.nodes.Compute"/>
                <NodeTemplate name="wb2" type="tosca.nodes.Compute"/>
                <NodeTemplate name="lb" typ="tosca.nodes.Compute"/>
                <RelationshipTemplate name=“db2ws1” type=“hostname”/>
                <RelationshipTemplate name=“db2ws2” type=“hostname”/>
                <RelationshipTemplate name=“ws12lb” type=“hostname”/>
                <RelationshipTemplate name=“ws22lb” type=“hostname”/>
```

© All Rights Reserved

The following diagram depicts the situation resulting from step 3 with the naming convention for the **RelationshipTemplate** elements representing the semantic “from the source node to the target node”:

Step 4: The Hardware Characteristics

In this step we shall expand the four NodeTemplate elements to provide the required hardware characteristics in terms of compute cores, RAM and disk storage.

The requirement number 4, above, states that both memory and disk is important for the database host. This can be seen expressed below. The values provided are relative and may be adapted as required for your own use case.

```
<NodeTemplate name="db" type="tosca.nodes.Compute">
    <Capabilities>
        <Capability name="host”>
            <Properties>
                <num_cpus>1</num_cpus>
                <mem_size>8G</mem_size>
                <disk_size>100G</disk_size>
            </Properties>
        </Capability>
    </Capabilities>
</NodeTemplate>
```

This can now be repeated for the NodeTemplate elements of the two web servers and for the load balancer NodeTemplate element.

The requirement number 5, above, states that both compute cores and memory are important for the web server hosts. This can be seen expressed below. Here again, the values provided are relative and may be adapted as required for your own use case.

```
<NodeTemplate name="ws1" type="tosca.nodes.Compute">
    <Capabilities>
        <Capability name="host”>
            <Properties>
                <num_cpus>8</num_cpus>
                <mem_size>16G</mem_size>
                <disk_size>8G</disk_size>
            </Properties>
        </Capability>
    </Capabilities>
</NodeTemplate>
<NodeTemplate name="ws2" type="tosca.nodes.Compute">
    <Capabilities>
        <Capability name="host”>
            <Properties>
```

© All Rights Reserved


```
                <num_cpus>8</num_cpus>
                <mem_size>16G</mem_size>
                <disk_size>8G</disk_size>
            </Properties>
        </Capability>
    </Capabilities>
</NodeTemplate>
```

The requirement number 6, above, states that the number of compute cores is important for the load balancer. This can be seen expressed below. The values provided are relative and may be adapted as required for your own use case.

```
<NodeTemplate name="lb" type="tosca.nodes.Compute">
    <Capabilities>
        <Capability name="host”>
            <Properties>
                <num_cpus>8</num_cpus>
                <mem_size>2G</mem_size>
                <disk_size>8G</disk_size>
            </Properties>
        </Capability>
    </Capabilities>
</NodeTemplate>
```

Step 5: System Software Characteristics

In this step we shall further expand the four NodeTemplate elements to define the required system software characteristics as defined by the requirement number 4, above, to be 64 bit Ubuntu 16.04.

The following Capability element can be inserted into each of the NodeTemplate elements just after the “host” Capability, but the actual order is not important so long as the structure of the Capabilities element is respected.

```
        <Capability name="os”>
            <Properties>
                <architecture>x86_64</architecture>
                <type>linux</type>
                <distribution>Ubuntu</distribution>
                <version>16.04</version>
            </Properties>
        </Capability>
```

Step 6: Connecting it all Up

In this step we shall expand the **RelationshipTemplate** elements of the **TopologyTemplate** and add the **SourceElement** and **TargetElement** elements with the corresponding reference details as can be seen below.

```
                <RelationshipTemplate name="db2ws1” type="hostname”>
                    <SourceElement ref="db”/>
                    <TargetElement ref="ws1”/>
                </RelationshipTemplate>
```

© All Rights Reserved

```
                <RelationshipTemplate name="db2ws2" type="hostname">
                    <SourceElement ref="db"/>
                    <TargetElement ref="ws2"/>
                </RelationshipTemplate>
                <RelationshipTemplate name="ws12lb" type="hostname">
                    <SourceElement ref="ws1"/>
                    <TargetElement ref="lb"/>
                </RelationshipTemplate>
                <RelationshipTemplate name="ws22lb" type="hostname">
                    <SourceElement ref="ws2"/>
                    <TargetElement ref="lb"/>
                </RelationshipTemplate>
```

The following diagram depicts the situation resulting from step 6 with the **RelationshipTemplate** elements describing the interconnections between the hardware nodes.

Step 7: The Definitions File

In this step we shall define the characteristics of the standard TOSCA Compute via a **NodeType** definition in the definitions file referenced in the **Import** statement.

```
<NodeType name="tosca.nodes.Compute">
    <Interfaces>
        <Interface name="Standard">
            <Operation name="create"/>
            <Operation name="start"/>
            <Operation name="stop"/>
            <Operation name="save"/>
            <Operation name="snapshot"/>
            <Operation name="delete"/>
        </Interface>
    </Interfaces>
</NodeType>
```

Step 8: Service Conditions

In this step we shall add further details to the Tags element of the ServiceTemplate to specify the customer account and the deployment requirements. The following Tag elements shows standard QUOTA controlled deployment using AMAZON AWS EC2 provisioning in Dublin, Ireland for the customer account named “EXAMPLE”.

```
                        <Tag name="Account" value="example"/>
                        <Tag name="Algorithm" value="quota:default"/>
                        <Tag name="Provider" value="amazonec2"/>
                        <Tag name="Zone" value="eu-west-1"/>
```

The resulting BEAM/TOSCA document could now be processed by the BASMATI Platform and would result in the deployment at AMAZON AWS in Dublin, of the four virtual machines as

© All Rights Reserved

described by the different host capabilities, each running the Ubuntu operating system and configured, as required, with the IP Address at which their correspondents have been deployed.Please refer to APPENDIX A for an example of the BEAM topology in two use cases of BASMATI.

2.2.2 Functional Requirements

Using TOSCA, it is possible to define only the software components of an application in a template and just express constrained requirements against the hosting infrastructure. At deployment time, the provider can then do a late binding and dynamically allocate or assign the required hosting infrastructure and place software components on top.

Below we illustrate the functional requirements specification for the database storage of the Yellow Map application in TOSCA format.

Functional Requirements: tosca.nodes.YellowMap.Storage

```
app_id: XXXX     # application id

host_capabilities:
    disk_size: 10 GB
    num_cpus: [ 1, 2, 4, 8 ]
    mem_size: 4096 MB

os_capabilities: &os_capabilities
    architecture: x86_64
    type: Linux
    distribution: Ubuntu
    version: 14.04

inputs:
   server_mysql_rootpw = abc123
   server_mysql_port = 8091
   database_name =  my_db
   database_user = my_user
   database_port =  8091

app_db:
   db_content:
   file: files/my_db_content.txt
   requirements:
   Create:
     implementation: db_create.sh
```

2.2.3 Template Agreement

The objective of the Webservice-Agreement specification is to define a language and a protocol for advertising the capabilities of service providers and creating agreements based on templates, and for monitoring agreement compliance at runtime. An agreement between a service consumer and a service provider specifies one or more service level objectives both as expressions of requirements of the service consumer and assurances by the service provider on

© All Rights Reserved

the availability of resources and/or on service qualities. The WS-Agreement specification relies on a set of well established standards like XML, SOAP, WSDL and WSRF.

WS-Agreement extends the classical service discovery and usage model since it allows service consumers not only to discover and use services, but also to dynamically negotiate the quality with which the service is provided. Once the service consumer and the service provider achieved a common understanding of the service provisioning, an agreement or SLA is created that serves as a formal contract between the two parties and describes the rights and obligations of each party in the context of the service provisioning process. An agreement life cycle includes the creation, termination and monitoring of agreement states.

Therefore, customer Service Level Agreements are to be created as the result of an interactive service selection and refinement process performed by a customer of an operator of a Basmati platform. Then, the agreement defines the level and nature of the service expected by the customer and may also define and impose the terms under which the customer is authorised to use the service. Additionally , it may define commercial aspects of the service such as price.

As example, we demonstrate bellow part of the content of a **WS Agreement** stored as part of the BEAM in the BASMATI Platform.


```
<wsag:Agreement xmlns:wsag="http://www.ggf.org/namespaces/ws-agreement"
xmlns:sla="http://sla.atos.eu" wsag:AgreementId="agreement-a">    <wsag:Name> Agreement
Yellow Map</wsag:Name>
    <wsag:Context>
        <wsag:AgreementInitiator>client-prueba</wsag:AgreementInitiator>
        <wsag:AgreementResponder>provider-a</wsag:AgreementResponder>
        <wsag:ServiceProvider>AgreementResponder</wsag:ServiceProvider>
        <wsag:ExpirationTime>2017-03-07T12:00:00Z</wsag:ExpirationTime>
        <wsag:TemplateId>template-a</wsag:TemplateId>
        <sla:Service>service-a</sla:Service>
    </wsag:Context>
    <wsag:Terms>
        <wsag:All>
            <wsag:ServiceProperties wsag:Name="ServiceProperties"
wsag:ServiceName="ServiceName">
                <wsag:VariableSet>
                    <wsag:Variable wsag:Name="ResponseTime" wsag:Metric="xs:double">
                        <wsag:Location>service-prueba/ResponseTime</wsag:Location>
                    </wsag:Variable>
                    <wsag:Variable wsag:Name="Performance" wsag:Metric="xs:double">
                        <wsag:Location>service-prueba/Performance</wsag:Location>
                    </wsag:Variable>
                </wsag:VariableSet>
            </wsag:ServiceProperties>
            <wsag:GuaranteeTerm wsag:Name="GT_ResponseTime">
                <wsag:ServiceScope wsag:ServiceName="ServiceName"/>
                <wsag:ServiceLevelObjective>
                    <wsag:KPITarget>
                        <wsag:KPIName>ResponseTime</wsag:KPIName>
                        <wsag:CustomServiceLevel>
                            <sla:Slo>
```

© All Rights Reserved


```
                                <sla:Constraint>ResponseTime LT 0.5</sla:Constraint>
                            </sla:Slo>
                        </wsag:CustomServiceLevel>
                    </wsag:KPITarget>
                </wsag:ServiceLevelObjective>
            </wsag:GuaranteeTerm>
            <wsag:GuaranteeTerm wsag:Name="GT_Performance">
                <wsag:ServiceScope wsag:ServiceName="ServiceName"/>
                <wsag:ServiceLevelObjective>
                    <wsag:KPITarget>
                        <wsag:KPIName>Performance</wsag:KPIName>
                        <wsag:CustomServiceLevel>
                            <sla:Slo>
                                <sla:Constraint>Performance BETWEEN
(0.3,1)</sla:Constraint>
                            </sla:Slo>
                        </wsag:CustomServiceLevel>
                    </wsag:KPITarget>
                </wsag:ServiceLevelObjective>
                <wsag:BusinessValueList>
                    <wsag:CustomBusinessValue>
                        <sla:Penalty type="discount" expression="35" unit="%"
validity="P1D"/>
                    </wsag:CustomBusinessValue>
                    <wsag:CustomBusinessValue count="5" duration="P1D">
                        <sla:Penalty type="service" expression="sms" validity="P1M"/>
                    </wsag:CustomBusinessValue>
                </wsag:BusinessValueList>
            </wsag:GuaranteeTerm>
        </wsag:All>
    </wsag:Terms>
</wsag:Agreement>
```

2.2.4 Decompostion, Selection and Deployment documents

These documents are part of the BEAM but created by the BASMATi platform during its runtime operations. The Decision Maker components coordinates the creation of these documents, which is performed in collaboration with other BASMATI components. For the detailed process on how these documents are created, please refer to the Deliverable D2.3, Section \ref{} that contains the sequence diagram of the interactions between component.

The Decomposition Document (DD) is a XML-based document that describes the decomposition of the application and defines the partition of the applications, in accordance with the model provided in Section 2.1. In addition, it can provide an estimate on the degree of replication for each partition, according to the information received by the Knowledge Extractor component.

The Selection Document (SD) is a ranking of the available federated resources able to execute the partitions identified by the DD, while meeting the SLAs provided with the application. Such ranking is an XML-based document and is provided by the Resource Broker component.

The Deployment Document Identifies the mapping between application partition and resources of the federated environment, and it is created by the Decision Maker by joining the information

© All Rights Reserved

in the DD and SD. The Deployment Document is created by resolving an multi-objective optimization problem, as described in Section 3.1 of this document. Once created, the Deployment Document is delivered to the Amenesik Cloud Engine to trigger the deployment of the application.

2.3 Application Repository

As we have seen, the BEAM has a crucial importance for proper operation of a given application in BASMATI. In this context, the Application Repository is the component in charge of storing the BEAM documents inside the Platform. Each application "basmatized" has it own BEAM documents, no documents are shared between the applications. The Application Repository (AP) is a passive component, serving the request of the other components. For this purpose, the AP provides a RESTFUL API for the retrieval and manipulation of the documents inside the BEAM. Therefore, among the components inside the BASMATI Platform, the AP is seen as common repository of the BEAM documents.

2.3.1 Implementation Design

The Application Repository has been developed using Django REST framework. Django REST framework is a powerful and flexible toolkit for building Web APIs. More details about this framework can be found here: http://www.django-rest-framework.org/

The design motivation on why we have adopted this framework are the following:

(a) Authentication policies including packages for OAuth1a and OAuth2;
(b) Serialization that supports both ORM and non-ORM data sources;
(c) Extensive documentation, and wide-spread community support;
(d) Used and trusted by internationally recognised companies including Mozilla, Red Hat, Heroku, and Eventbrite.

As database engine, we have adopted the SQLite. SQLite is a relational database management system contained in a C programming library. In contrast to many other database management systems, SQLite is not a client–server database engine. Rather, it is embedded into the end program.

More details can be found though the this link: https://sqlite.org

2.3.2 Application repository internal structure

© All Rights Reserved

The AP stores all the information regard the BEAM in a database. The database store the data of the following entities: Application, Application Topologies, Template Agreements, Functional Requirements, Selection Document and Decomposition Document.

a) Application entity. Represents the high level description of application managed by the BASMATI Platform. Internally, the AP salves the following fields:

- uuid: uuid of the application.
- name: the name of the application.
- description: a simple description of the application.

Example of an Application entry:


```
"id": "0a24af65-38ab-48c7-9293-62ac4c7bfc34",
"name": "yellow_map",
"description": "Description of yellow_map application"
```


b) Remaining entities (Applications Topologies, Template Agreements, Functional Requirements, Selection Document and Decomposition Document). The remain entities composes the structure of the BEAM. Internally, in the AP each entity has its own table. Fields saved:

- id: uuid of the register
- content: the textual content of the entity. Notice that here we store XML, Tosca definitions, strings and any other kind of content not binary as text.
- current_application_setting: this flag indicates among the registers of an entity those considered as a current valid for the setup of a basmati application
- application_id: the uuid of a basmati application.
- created_datetime: it informs the date and time that the register has been created.
- udpate_datetime: it informs the date and time that the register was modified.

Example of a register stored as an Application Topology of Yellow_Map.


```
"id": "49d4307e-7aa5-438a-ac0e-d4e34df7d4e8",
"content": "<Definitions>\r\n\r\n\t<Import
location=\"http://www.amenesik.com/tosca/yellow-map-tosca-
defaults.xml\"/>.../Definitions>",
"current_application_setting": true,
"created_datetime": "2017-12-27T11:46:46.884239Z",
"last_updated_datetime": "2018-01-11T09:46:31.275770Z",
```

© All Rights Reserved

"application_id": "0a24af65-38ab-48c7-9293-62ac4c7bfc34"

3 Adaptivity in BASMATI

The BASMATI Platform is aimed at providing a complete ecosystem able to efficiently deploy and manage mobile application on Federated Cloud resources. The demand of these application can be eventually non-uniformly distributed in time and space, creating new challenges for deploying, scaling and *adapting* to such application. Adaptation to the dynamic environment changes such as the resources availability, network conditions, and users behaviour is one of the main aspect that should be considered to design a efficient adaptive mechanisms for placement of applications. In BASMATI, we employ two kinds of adaptively mechanisms, namely proactive and reactive.

Proactive mechanisms are based on the idea to propose alternatives plans for the geographical deployment of the application before an actual change in the environment happens. These plans are created by the Decision Maker component exploiting two resources:

- a proper modeling of the resources, coming from the BEAM topology (as defined above in the Deliverable)
- a timely prediction of the resource usage of the application, and the behaviour of the users from the Knowledge Extractor component (see deliverable [ref]).

In fact, the proactive mechanisms can be as good as the information they receive, and therefore an accurate analysis of the past data is of paramount importance in this case. The Decision Maker employs an evolutionary heuristics to provide the main plans and the alternative plans, considering various optimization objectives such as the coverage of the user and economical convenience (more details in the Deliverable 4.3). In this context the alternative plans are those in the Pareto front of the resulting optimization problem. Please note that this feature is still to be implemented and will come with the next version of the Decision Maker

Reactive mechanisms require the notification of a change in the environment before to be actually executed. The BASMATI Brokerage Platform applies an automatic reactive scaling mechanism. The scaling mechanism needs pre-defined rules to trigger scaling. In BASMATI, the reactive mechanisms relies on online monitors to detect the changes in workload (e.g. request arrival rate) and performs the corresponding scaling. Within this context, the reactive mechanism is applied in BASMATI to perform application scale-up and scale-down whenever the user demand changes or to handle failure of application.

Public
© All Rights Reserved

3.1 Decison Maker

In the context mentioned above, the Decision Maker (DM) module is entitled of taking proactive and reactive decisions for optimal application placement. It is also in charge of making major adaptive offloading decisions at runtime. For this purpose, the DM communicates specially with two other components of the BASMATI platform, the Knowledge Extractor (KE) and the Resource Broker (RB). The idea is that combining predicted information provided by the KE and a set of available resource provided by the RB, the DM generates optimal solutions for deployment of an application. To do so, for given a set of cloud provider instances, the decision maker minimizes the deployment costs and besides, it maximizes the coverage of the application. To improve the coverage, the DM explores placement schemas considering providers closer to the application demand, such that the user's experience is satisfied.

The generated deployment plans will also integrate the BEAM and, therefore, be stored at the Application Repository. The deployment plans guide the Application Controller Component for the deployment of the application in the cloud Federation. As aforementioned, the amount of data effectively stored and managed by the Decision Maker is very limited. Basically, it consists in the information associated to the application it managed and currently in execution into the cloud federation, possibly annotated with user preferences and requirements. Keeping such information is fundamental to be able to properly react to performance issues and violation, either for providing alternative deployment scenarios or for restructuring the application.

Decision Maker Algorithm

The Decision Maker addresses a multi-objective optimization problem. A Multi-objective optimization (also known as multicriteria optimization or Pareto optimization) is an area of multiple criteria decision making, that is concerned with mathematical optimization problems involving more than one objective function to be optimized simultaneously. In this context, the Decision Maker Component addresses a multi-objective problem aimed at minimizing cost while maximizing application demand coverage when allocating resource in a cloud provider federation. Moreover, it is required that the allocated resources meet the application's functional requirements.

Here, the cost is measure based on the fare for hiring cloud provider instances in the Cloud Federation. Cloud providers such as Amazon and Azure apply different fares according to the resource setting (e.g. CPU, Memory, Storage) of their instances. Furthermore, such provider instances are worldwide located, thus covering optimally different geospatial areas. In general, the closer to the demand, the better is the coverage for an spatial area. Thus, from this perspective, the objective of the Decision Maker is to find a set of instance provides, maximizing

© All Rights Reserved

the demand coverage, minimizing the cost up to a given budget and respecting a list of functional requirements.

The Decision Maker receive as input for the optimization process two sparse matrix:

a) Spatio-temporal Demand Predictions Matrix (SDPM). This input is provided by the knowledge Extractor, which uses Machine Learning techniques over past data for predicting spatial demand for an specific application over the time. Therefore, this sparse matrix has the following dimensions: temporal and spatial. The temporal dimension consists into fixed time intervals, defined according to monitoring needs of an application. In turn, the spatial dimension consists in tiles/cell of a Geospatial grid. The geospatial grid comprises the spatial areas of interest for monitoring of user's demands. Then for each pair composed by these two dimensions, we have the predicted demand of the application for a given time interval (in the future) and a specific geospatial area.
b) Provider Instances Resource Cost Matrix. This input is provided by the Broker, which contains the current list of provider instances inside the Federation. This sparse matrix consists of two dimensions: providers instances and resource capabilities. The provider instance dimension correspond to a tuple defined by the instance id and the city where the instance is located. Notice that, each instance (e.g. Amazon EC2 m3.xlarge) belongs to unique cloud provider (e.g. Amazon) and a cloud provider has many instances (e.g. EC2 m3.small, EC2 m3.medium, EC2 m3.xlarge). In turn, the dimension with the resource capabilities represents a dictionary describing a setting of capabilities (e.g. CPU: 2ghz, Memory: 8 Gbs, Hard Disk: 120 Gbs, OS: Ubuntu). It is also important to highlight that the Broker returns only provider instances that meets the function requirements of an application, for example: for an service application running on ubuntu, it would not return Windows VMs as possible resources. By crossing the dimensions, the cost of each provider instance, with specifics resource capabilities can be found.

To solve this multi-objective optimization problem, the Decision Maker uses a genetic algorithm (GA) to find a suitable optimal solution for the trade-off between coverage and cost. In computer science and operations research, a genetic algorithm is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on bio-inspired operators such as mutation, crossover and selection. Furthermore, for the Decision Maker, we have adapted the algorithm to consider a limit budget pruning solution that overcomes it.

© All Rights Reserved

In a genetic algorithm, a population of candidate solutions (called individuals) to an optimization problem is evolved toward better solutions. Each candidate solution has a set of properties (its chromosomes or genotype) which can be mutated and altered; traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible.[2]

The Figure 6 helps us to interpret the genetic representation of the solution domain for the Decision Maker. Here, an application can be seen as a set of services able to run independently in different provider instances. We recall that In the BASMATI Platform, such representation of the application services is defined in the topology template inside the BEAM. In this context, a individual (chromosome) represents a complete solution for deployment of an application in cloud, i.e. the number of instances hired for each service that composes a given application.

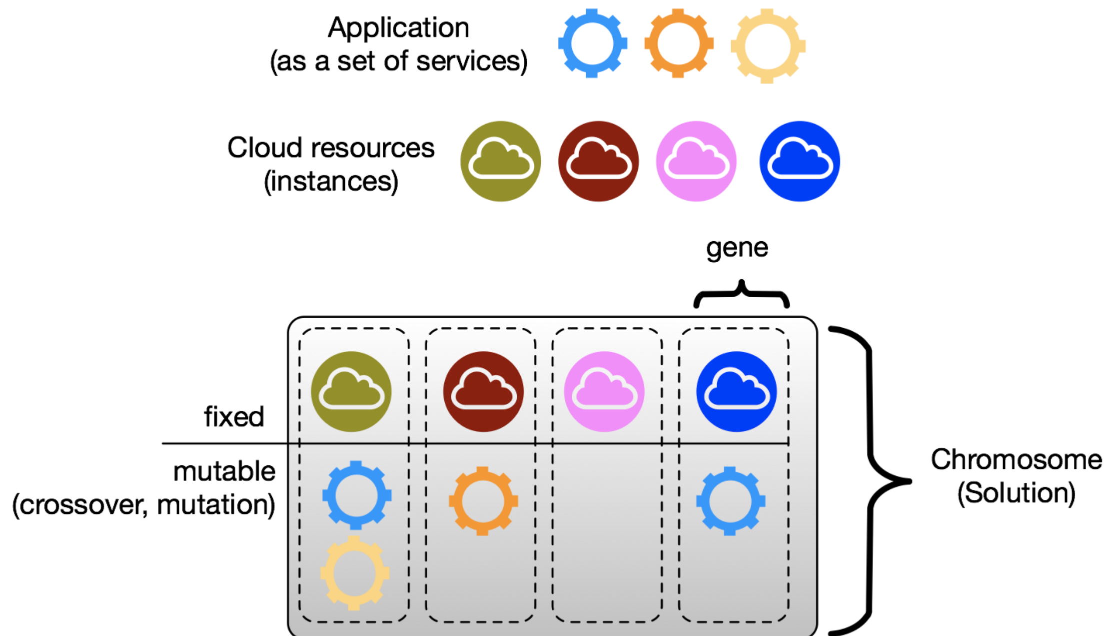

Figure 6. Representation of a Individual (Chromosome) solution in the Decision Maker

In a GA, a fitness function is used to evaluate an individual solution. Thus, in the Decision Maker, the different combinations generated by the AG operators (e.g. Mutation and CrossOver) for allocating the set of services of a given application to the list of available provider instances are evaluated according to our fitness function. This fitness function takes into account the total cost for hiring the provider instances of an individual solution and the coverage achieved by chosen these instances. For a more detailed description of the genetic

Public
© All Rights Reserved

algorithm implemented by the Decision Maker, including the specific fitness functions, please refer to the Deliverable 4.3.

4 Further Developments for BASMATI

Further development are expected before the end of the projects at M26, especially in relation with the components. First, the plan is for the Application Repository to implement an easy way to express interdependencies between TOSCA documents, in fact easing the navigation when dealing with the *import* keyword. Second, the Decision Maker's plan is to implement and optimize the computation and retrieval of alternative solutions, specifically by computing the Pareto front of the optimization problem defined above.

References

Binz, T., Breitenbücher, U., Kopp, O., & Leymann, F. (2013). TOSCA: Portable Automated Deployment and Management of Cloud Applications. In *Advanced Web Services* (pp. 527–549).

© All Rights Reserved

Appendix A

Topology Template - DASFEST APPLICATION

This example shows the first version of the BEAM document defined by AMENESIK and YELLOWMAP for the DAS FEST Use Case Application of BASMATI. This BEAM describes a system comprising five hardware nodes and their corresponding software requirements and the subsequent internode relationships between them.

The initial Import element is used to retrieve the collection of definitions for the NodeType and NodeType Implementation elements used within the TopologyTemplate of this ServiceDescription.


```
<Definitions>
	<Import location="http://www.amenesik.com/tosca/yellow-map-tosca-defaults.xml"/>
	<ServiceTemplate name="basmati-yellow-map-application-world">
		<Tags>
			<Tag name="Title" value="Yellow Map Application Control"/>
			<Tag name="SubTitle" value="Basmati Use Case"/>
			<Tag name="Version" value="1.0a.03-world"/>
			<Tag name="Author" value="Iain James Marshall, Thorsten Zylowski"/>
			<Tag name="Date" value="28th June 2017"/>
			<Tag name="Account" value="basmati"/>
			<Tag name="Algorithm" value="quota:federation"/>
			<Tag name="Provider" value="amazonec2"/>
			<Tag name="Zone" value="[eu-west-1,eu-central-1,ap-northeast-2]"/>
		</Tags>
		<TopologyTemplate>
		<NodeTemplate name="storage" type="tosca.nodes.Compute">
				<Capabilities>
					<Capability name="host">
						<Properties>
						<num_cpus>1</num_cpus>
						<disk_size>40G</disk_size>
						<mem_size>1G</mem_size>
						<tcp_port>27017</tcp_port>
						</Properties>
					</Capability>
					<Capability name="os">
						<Properties>
						<architecture>x86</architecture>
						<type>linux</type>
						<distribution>Ubuntu</distribution>
						<version>14.04</version>
						</Properties>
					</Capability>
				</Capabilities>
			</NodeTemplate>
			<NodeTemplate name="storage-engine"
type="tosca.nodes.YellowMap.Storage">
				<Capabilities>
					<Capability name="engine">
						<Properties>
						<password>password</password>
						<username>someone</username>
						</Properties>
					</Capability>
				</Capabilities>
				<Requirements>
```

© All Rights Reserved

```
                                        <Requirement name="storage" type="host"/>
                                </Requirements>
                        </NodeTemplate>
                        <NodeTemplate name="collector" type="tosca.nodes.Compute">
                              <Capabilities>
                                     <Capability name="host">
                                            <Properties>
                                            <num_cpus>1</num_cpus>
                                            <disk_size>20G</disk_size>
                                            <mem_size>4G</mem_size>
                                            </Properties>
                                     </Capability>
                                     <Capability name="os">
                                            <Properties>
                                            <architecture>x86</architecture>
                                            <type>linux</type>
                                            <distribution>Ubuntu</distribution>
                                            <version>14.04</version>
                                            </Properties>
                                     </Capability>
                              </Capabilities>
                       </NodeTemplate>
                        <NodeTemplate name="userdata_collector"
type="tosca.nodes.YellowMap.Collector">
                                <Capabilities>
                                        <Capability name="engine">
                                                <Properties>
                                                <password>password</password>
                                                <username>someone</username>
                                                </Properties>
                                        </Capability>
                                </Capabilities>
                                <Requirements>
                                        <Requirement name="collector" type="host"/>
                                </Requirements>
                        </NodeTemplate>
                       <NodeTemplate name="miner" type="tosca.nodes.Compute">
                              <Capabilities>
                                     <Capability name="host">
                                            <Properties>
                                            <num_cpus>2</num_cpus>
                                            <disk_size>10G</disk_size>
                                            <mem_size>8G</mem_size>
                                            </Properties>
                                     </Capability>
                                     <Capability name="os">
                                            <Properties>
                                            <architecture>x86</architecture>
                                            <type>linux</type>
                                            <distribution>Ubuntu</distribution>
                                            <version>14.04</version>
                                            </Properties>
                                     </Capability>
                              </Capabilities>
                       </NodeTemplate>
                        <NodeTemplate name="userdata_miner"
type="tosca.nodes.YellowMap.Miner">
                                <Capabilities>
                                        <Capability name="engine">
                                                <Properties>
                                                <password>password</password>
                                                <username>someone</username>
                                                </Properties>
                                        </Capability>
                                </Capabilities>
                                <Requirements>
```

```
                        <Requirement name="miner" type="host"/>
                    </Requirements>
                </NodeTemplate>
                <NodeTemplate name="search_engine" type="tosca.nodes.Compute">
                    <Capabilities>
                        <Capability name="host">
                            <Properties>
                            <num_cpus>2</num_cpus>
                            <disk_size>10G</disk_size>
                            <mem_size>8G</mem_size>
                            <tcp_port>8080</tcp_port>
                            <tcp_port>4848</tcp_port>
                            </Properties>
                        </Capability>
                        <Capability name="os">
                            <Properties>
                            <architecture>x86</architecture>
                            <type>linux</type>
                            <distribution>Ubuntu</distribution>
                            <version>14.04</version>
                            </Properties>
                        </Capability>
                    </Capabilities>
                </NodeTemplate>
                <NodeTemplate name="userdata_search"
type="tosca.nodes.YellowMap.SearchEngine">
                    <Capabilities>
                        <Capability name="engine">
                            <Properties>
                            <password>password</password>
                            <username>someone</username>
                            </Properties>
                        </Capability>
                    </Capabilities>
                    <Requirements>
                        <Requirement name="search_engine" type="host"/>
                    </Requirements>
                </NodeTemplate>
                <NodeTemplate name="operator" type="tosca.nodes.Compute">
                    <Capabilities>
                        <Capability name="host">
                            <Properties>
                            <num_cpus>2</num_cpus>
                            <disk_size>10G</disk_size>
                            <mem_size>8G</mem_size>
                            </Properties>
                        </Capability>
                        <Capability name="os">
                            <Properties>
                            <architecture>x86</architecture>
                            <type>linux</type>
                            <distribution>Ubuntu</distribution>
                            <version>14.04</version>
                            </Properties>
                        </Capability>
                    </Capabilities>
                </NodeTemplate>
                <NodeTemplate name="operator_interface"
type="tosca.nodes.YellowMap.Operator">
                    <Capabilities>
                        <Capability name="engine">
                            <Properties>
                            <password>password</password>
                            <username>someone</username>
                            </Properties>
                        </Capability>
```

© All Rights Reserved

```
                                </Capabilities>
                                <Requirements>
                                        <Requirement name="operator" type="host"/>
                                </Requirements>
                        </NodeTemplate>
                        <RelationshipTemplate name="collector2storage" type="hostname" >
                                <SourceElement ref="storage"/>
                                <TargetElement ref="collector"/>
                        </RelationshipTemplate>
                        <RelationshipTemplate name="miner2storage" type="hostname" >
                                <SourceElement ref="storage"/>
                                <TargetElement ref="miner"/>
                        </RelationshipTemplate>
                <RelationshipTemplate name="search_engine2storage" type="hostname" >
                                <SourceElement ref="storage"/>
                                <TargetElement ref="search_engine"/>
                        </RelationshipTemplate>
                        <RelationshipTemplate name="operator2storage" type="hostname" >
                                <SourceElement ref="storage"/>
                                <TargetElement ref="operator"/>
                        </RelationshipTemplate>
                </TopologyTemplate>
        </ServiceTemplate>
</Definitions>
```

© All Rights Reserved

Topology Template - TRIPBUILDER APPLICATION

This example shows the first version of the BEAM document defined by AMENESIK and CNR for the TRIP BUILDER Use Case Application of BASMATI. This document describes a system comprising five hardware nodes and their corresponding software requirements and the subsequent internode relationships between them.

The initial Import element is used to retrieve the collection of definitions for the NodeType and NodeType Implementation elements used within the TopologyTemplate of this ServiceDescription.

```
<Definitions>
        <Import location="http://www.amenesik.com/tosca/trip-builder-tosca-defaults.xml"/>
        <ServiceTemplate name="basmati-trip-builder-application-world">
                <Tags>
                        <Tag name="Title" value="Trip Builder Application Control"/>
                        <Tag name="SubTitle" value="Basmati Use Case"/>
                        <Tag name="Version" value="1.0a.03-world"/>
                        <Tag name="Author" value="Iain James Marshall, Vinicius Monteiro"/>
                        <Tag name="Date" value="17th May 2017"/>
                        <Tag name="Account" value="basmati"/>
                        <Tag name="Algorithm" value="quota:federation"/>
                        <Tag name="Provider" value="amazonec2"/>
                        <Tag name="Zone" value="[eu-west-1,eu-central-1,ap-northeast-2]"/>
                </Tags>
                <TopologyTemplate>
                        <NodeTemplate name="storage" type="tosca.nodes.Compute">
                                <Capabilities>
                                        <Capability name="host">
                                                <Properties>
                                                <num_cpus>1</num_cpus>
                                                <disk_size>40G</disk_size>
                                                <mem_size>1G</mem_size>
                                                </Properties>
                                        </Capability>
                                        <Capability name="os">
                                                <Properties>
                                                <architecture>x86</architecture>
                                                <type>linux</type>
                                                <distribution>Ubuntu</distribution>
                                                <version>14.04</version>
                                                </Properties>
                                        </Capability>
                                </Capabilities>
                        </NodeTemplate>
                       <NodeTemplate name="storage-engine"
type="tosca.nodes.TripBuilder.Storage">
                                <Capabilities>
                                        <Capability name="engine">
                                                <Properties>
                                                <password>password</password>
                                                <username>someone</username>
                                                </Properties>
                                        </Capability>
                                </Capabilities>
                                <Requirements>
                                        <Requirement name="storage" type="host"/>
                                </Requirements>
                        </NodeTemplate>
```

© All Rights Reserved

```
<NodeTemplate name="crawler" type="tosca.nodes.Compute">
        <Capabilities>
                <Capability name="host">
                        <Properties>
                        <num_cpus>1</num_cpus>
                        <disk_size>20G</disk_size>
                        <mem_size>4G</mem_size>
                        </Properties>
                </Capability>
                <Capability name="os">
                        <Properties>
                        <architecture>x86</architecture>
                        <type>linux</type>
                        <distribution>Ubuntu</distribution>
                        <version>14.04</version>
                        </Properties>
                </Capability>
        </Capabilities>
</NodeTemplate>

 <NodeTemplate name="crawler-engine"
type="tosca.nodes.TripBuilder.Crawler">
         <Capabilities>
                 <Capability name="engine">
                         <Properties>
                         <password>password</password>
                         <username>someone</username>
                         </Properties>
                 </Capability>
         </Capabilities>
         <Requirements>
                 <Requirement name="crawler" type="host"/>
         </Requirements>
 </NodeTemplate>
<NodeTemplate name="processor" type="tosca.nodes.Compute">
        <Capabilities>
                <Capability name="host">
                        <Properties>
                        <num_cpus>2</num_cpus>
                        <disk_size>10G</disk_size>
                        <mem_size>8G</mem_size>
                        </Properties>
                </Capability>
                <Capability name="os">
                        <Properties>
                        <architecture>x86</architecture>
                        <type>linux</type>
                        <distribution>Ubuntu</distribution>
                        <version>14.04</version>
                        </Properties>
                </Capability>
        </Capabilities>
</NodeTemplate>
<NodeTemplate name="processor-engine"
type="tosca.nodes.TripBuilder.Processor">
         <Capabilities>
                 <Capability name="engine">
                         <Properties>
                         <password>password</password>
                         <username>someone</username>
                         </Properties>
                 </Capability>
         </Capabilities>
         <Requirements>
                 <Requirement name="processor" type="host"/>
         </Requirements>
```

© All Rights Reserved

```
                         </NodeTemplate>
                        <RelationshipTemplate name="crawler2storage" type="hostname" >
                                <SourceElement ref="storage"/>
                                <TargetElement ref="crawler"/>
                        </RelationshipTemplate>

                        <RelationshipTemplate name="processor2storage" type="hostname" >
                                <SourceElement ref="storage"/>
                                <TargetElement ref="processor"/>
                        </RelationshipTemplate>
                </TopologyTemplate>
        </ServiceTemplate>
</Definitions>
```