Architecture for Managing Clouds

A White Paper from the Open Cloud Standards Incubator

Version: 1.0.0
Status: DMTF Informational
Publication Date: 2010-06-18
Document Number: DSP-IS0102

Abstract

This white paper is one of two Phase 2 deliverables from the DMTF Cloud Incubator and describes the reference architecture as it relates to the interfaces between a cloud service provider and a cloud service consumer. The goal of the Incubator is to define a set of architectural semantics that unify the interoperable management of enterprise and cloud computing.

Acknowledgments

The Open Cloud Standards Incubator wishes to acknowledge contributions from the following people:

- Kane, Valerie – Advanced Micro Devices
- Vancil, Paul – Advanced Micro Devices
- Kowalski, Vincent – BMC Software
- Lipton, Paul – CA, Inc.
- Moscovich, Efraim – CA, Inc.
- Waschke, Marvin – CA, Inc.
- Armstrong, Joseph – Cisco
- Sankar, Krishna – Cisco
- Serr, Frederick – Cisco
- Siefer, Mike – Cisco
- Tomsu, Peter – Cisco
- Wheeler, Jeff – Cisco
- Pardikar, Shishir – Citrix Systems Inc.
- Sirjani, Abolfazl – Citrix Systems Inc.
- Landau, Richard – Dell
- Ericson, George – EMC
- Linn, John – EMC
- Durand, Jacques – Fujitsu
- Song, Zhexuan – Fujitsu
- Aguiar, Glaucimar – Hewlett-Packard Company
- Cook, Nigel – Hewlett-Packard Company
- Maciel, Fred – Hitachi, Ltd.
- Baskey, Michael – IBM
- Johnson, Mark – IBM
- Pendarakis, Dimitrios – IBM
- Cox, Billy – Intel Corporation
- Li, Hong – Intel Corporation

- Raghu, Rekha – Intel Corporation
- Parchem, John – Microsoft Corporation
- Carter, Steve – Novell
- Jose, Jaimon – Novell
- Carlson, Mark – Oracle
- Vambenepe, William – Oracle
- Yu, Jack – Oracle
- Owens, Ken – Savvis
- Qualls, Richard – Sun Microsystems, Inc.
- Norbeck, Don – SunGard Availability Services
- Ronco, Enrico – Telecom Italia
- Galán Márquez, Fermín – Telefónica
- Bumpus, Winston – VMware Inc.
- Lamers, Lawrence – VMware Inc.
- Matthews, Jeanna – VMware Inc.

CONTENTS

Figures

Tables

1 Scope

This document is one of two documents that together describe how standardized interfaces and data formats can be used to manage clouds. This document focuses on the overall architecture; the other document focuses on interactions and data formats.

The scope of this architecture document includes:

- **Reference architecture for managing clouds**. This reference architecture was introduced in the DMTF *Interoperable Clouds* white paper (DSP-IS0101). The concepts are further explored and described in this document.
- **Requirements**. These requirements are for the architected interfaces in general, including requirements for the protocols, resource model, and security mechanisms.
- **Role of policies and constraints**. Given the focus on the interfaces for managing clouds rather than on the internal details of a cloud implementation, a useful abstraction is to define the desired capabilities of the cloud using policies and constraints. These are interpreted to be *what* the cloud service provider should offer rather than *how* it offers it.
- **Patterns for interactions**. These patterns of consumer/provider interactions repeat across many use cases with different operations and data payloads, depending on the use case.
- **An example cloud ecosystem**. To aid comprehension, an example of how a cloud service provider might design the management interfaces is provided, with a discussion of design considerations. It should be emphasized that this section is not prescriptive; rather, it is illustrative.

A companion white paper to this document, *Use Cases and Interactions for Managing Clouds* (DSP-IS0103), describes other aspects of managing clouds, namely:

- management use cases across the entire lifecycle of a cloud service
- interaction sequences among consumers, developers, and providers to implement the use cases
- data artifacts exchanged in the interaction sequences

The following aspects of clouds are specifically out of the scope of this document:

- The architecture addresses management function only; it does not address how general business applications use business services exposed by a cloud, or the applications deployed into a cloud.
- The architecture does not address how to *build* management function in a cloud. Instead, it focuses on the management *interfaces* to the cloud. Providers are free to implement the services behind these interfaces in any way.

2 References

Cloud Security Alliance, *Security Guidance for Critical Areas of Focus in Cloud Computing 2.1,* http://www.cloudsecurityalliance.org/csaguide.pdf

DMTF DSP0243, *Open Virtualization Format Specification 1.1*, http://www.dmtf.org/standards/published_documents/DSP0243_1.1.pdf

DMTF DSP1001, *Management Profile Specification Usage Guide 1.0*, http://www.dmtf.org/standards/published_documents/DSP1001_1.0.pdf

DMTF DSP-IS0101, *Interoperable Clouds, A White Paper from the Open Cloud Standards Incubator 1.0*, http://www.dmtf.org/standards/published_documents/DSP-IS0101_1.0.pdf

DMTF DSP-IS0103, *Use Cases and Interactions for Managing Clouds, A White Paper from the Open Cloud Standards Incubator 1.0,* http://www.dmtf.org/standards/published_documents/DSP-IS0103_1.0.pdf

ITIL® V3 Glossary, *Glossary of Terms, Definitions, and Acronyms*, v3.1.24, 11 May 2007, http://www.best-management-practice.com/gempdf/ITIL_Glossary_V3_1_24.pdf

National Institute of Standards and Technology, *The NIST Definition of Cloud Computing*, http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc

OASIS, *ebXML Messaging Services Version 3.0*, http://www.oasis-open.org/committees/download.php/24618/ebms_core-3.0-spec-cs-02.pdf

OASIS, *Web Services Make Connection (WS-MakeConnection) Version 1.1*, http://docs.oasis-open.org/ws-rx/wsmc/200702/wsmc-1.1-spec-os.html

Storage Networking Industry Association, *Cloud Data Management Interface Version 1.0g,* http://www.snia.org/tech_activities/publicreview/Cloud_Data_Management_Interface_Draft1.0.pdf

3 Terms and Definitions

For the purposes of this document, the following terms and definitions apply.

3.1

client

runs proprietary cloud software stacks side-by-side with DMTF standards-based protocol software stacks

3.2

cloud service

a publicly available service or a private service that is used within an enterprise

3.3

cloud service consumer

- approves business/financial expenditures for consumed services
- maintains accounts for used service instances
- requests service instances and changes to service instances (typically on behalf of the consumer business manager)
- provides access to services for service users

See Figure 2.

3.4

cloud service developer

designs, implements, and maintains service templates (see Figure 2)

3.5

cloud service provider

an organization that supplies cloud services to one or more internal or external consumers (see Figure 2)

3.6

configuration management database

CMDB

as used in the IT Infrastructure Library® context, a repository of information that represents authorized configuration of all the components of an information system

3.7

data artifacts

as used in this document, the control and status elements exchanged across the provider interface using the infrastructure

3.8

deployment

the process of creating a service instance in a reserved or prepared environment

NOTE: This may be more suitable for software or configuration deployment to existing running services.

3.9

infrastructure

as used in this document, the means by which control and status elements are exchanged between the cloud service provider and the cloud service consumer

The elements so exchanged are referred to as data artifacts.

3.10

Infrastructure as a Service

IaaS

the capability provided to the consumer to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications

The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly limited control of select networking components (for example, host firewalls). (Source: NIST)

3.11

notification

a signal from the cloud service provider to the cloud service consumer that a condition exists that requires attention

A notification can be either polled or sent asynchronously.

3.12

profiles

a specification that defines the CIM model and associated behavior for a management domain (for example, Server Virtualization) (Source: DSP1001)

The CIM model includes the CIM classes, associations, indications, methods, and properties. The management domain is a set of related management tasks.

3.13

provider interface

the interface through which cloud service consumers may access and monitor their contracted services

The interface covers SLO negotiation and measurement, service access, service monitoring, and billing. This interface is also the interface through which a cloud service developer interacts with a cloud service provider to create a service template that could be added to the service catalog. Service offerings, which contain the service template, are configured by providers. Cloud service consumers can then select the service offerings for their use. See Figure 2.

3.14

provisioning

the process of selecting, reserving, or creating an instance of a service offering

Service offerings are selected from the service provider's service catalog and are then provisioned into service instances.

Provisioning is also the process of selecting or reserving service resources from available pools, assembling them together, and configuring them based on a specific request in the contract, in order to fulfill the contract.

For example, a server instance can be created from a template; assigned CPU, memory, storage, and network resources; and configured for the consumer to satisfy the contract requirements (apply patches, adjust security, configure firewall, and so on).

3.15

security manager

responsible for managing the credentials and authentication processes as they relate to the operations across the provider interface

Security requirements for the cloud include user authentication, identity and access management, data protection, multi-tenancy resource isolation, monitoring and auditing for compliance, incident response, user and customer privacy, and the underlying portability and interoperability of security components.

3.16

service catalog

a database of information about the cloud services offered by a service provider

The service catalog includes a variety of information about the services, including description of the services, the types of services, cost, supported SLAs, and who can view or use the services. More generally, the service catalog contains information about services through their entire lifecycle. It contains service templates (created by developers), service offerings (created by providers), and deployed service instances.

3.17

service contract

an agreement between the cloud service provider and cloud service consumer to state the terms of service usage by the cloud service consumer

3.18

service entity

the representation of an identifiable logical element that serves and satisfies a list of operations

For example, a server (both physical and virtual) that contains an OS stack is a service entity that supports the prescribed operations defined on the particular OS. In addition, a service entity can contain a composite of other service entities or elements, as long as the group of these entities and elements satisfies, collectively, the prescribed operations.

3.19

service instance

the instantiation of a service request

3.20

service offering

a service template combined with service-level agreements, constraints, costs, billing information and other data necessary to offer the service described in the template to a consumer

3.21

service request

a request by a consumer to instantiate a service offering. Requests require service contracts, which may be created prior to or simultaneously with service requests

3.22

service template

a collection of items (machine images, connectivity definitions, storage, and so on) that are stored in the service catalog and can be provisioned at the cloud service provider (see Figure 3)

3.23

service-level agreement

SLA

in the context of cloud service providers, a negotiated, legally binding contract between the cloud service provider and cloud service consumer

The SLA includes agreements about services and responsibilities for service availability, performance, and billing. The SLA must be structured such that it can be propagated to all involved cloud service providers. The consumer contracts services from one cloud service provider who could be a broker, federator, or non-broker/non-federator, and that SLA is used as the basis for the broker or federator to programmatically contract for services from other cloud service providers.

3.24

service-level objective

SLO

a unique, distinct, and measurable aspect of an SLA

All parties of the SLA must agree that sets of SLOs and their measurement, correlation, and reporting will represent the compliance or non-compliance of unique stipulations within the SLA. The output of one or more correlated SLOs must be deterministic and supported by audit logs that allow interested parties to prove the output of an SLO or the correlation of the output of several SLOs.

3.25

service manager

responsible for managing the service instance and service topology

The service manager provides facilities to an administrator to create virtual machine instances and services. The service manager also provides mechanisms for creation, monitoring, control, and reporting of services.

3.26

strong authentication

authentication that requires more than the customary UserID and password. For example, biometric (fingerprint), certificate (X.509), smartcard, extra questions (for example, mother's maiden name) are used in addition to UserID and password to elevate the claim that the authentication is of high quality.

3.27

virtual image

virtual Image is an element (often as part of a package using the Open Virtualization Format), that encapsulates a workload consisting of all the code necessary to run the workload together with the metadata that is necessary to configure the environment in which to run it

3.28

workload portability

provides the capability for the cloud service consumer to create a service package and then provision that package in different cloud service providers without substantial modifications

4 Symbols and Abbreviated Terms

4.1
API
application programming interface

4.2
CMDB
Configuration Management Database

4.3
CMIWG
Cloud Management Interface Working Group

4.4
CMMWG
Cloud Management Model Working Group

4.5
CSP
cloud service provider

4.6
CSPI
cloud service provider interface

4.7
DoS
denial of service

4.8
DPI
deep packet inspection

4.9
HTTP
Hypertext Transfer Protocol

4.10
IaaS
Infrastructure as a Service

4.11
IDS
intrusion-detection system

4.12
MEP
Message Exchange Pattern

4.13
MOF
Managed Object Format

4.14
NIC
Network Interface Card

4.15
NIST
National Institute of Standards and Technology

4.16
OVF
Open Virtualization Format

4.17
PaaS
Platform as a Service

4.18
QoS
quality of service

4.19
RBAC
role-based access control

4.20
RDF
Resource Description Framework

4.21
REST
Representational State Transfer

4.22
RPC
Remote Procedure Call

4.23
SIP
Semantic Interaction Pattern

4.24
SLA
service-level agreement

4.25
SLO
service-level objective

4.26
SNIA
Storage Networking Industry Association

4.27

SOAP

Simple Object Access Protocol

4.28

SSL

Secure Sockets Layer

4.29

TCP

Transmission Control Protocol

4.30

TLS

Transport Layer Security

4.31

UDP

User Datagram Protocol

4.32

URI

Uniform Resource Identifier

5 Introduction

This document follows the definitions of cloud computing from work by the National Institute of Standards and Technology (*NIST Definition of Cloud Computing*, version 15*)*. In part, NIST defines cloud computing as "… a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (for example, networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."

NIST defines four cloud deployment models:

- public clouds (cloud infrastructure made available to the general public or a large industry group)
- private clouds (cloud infrastructure operated solely for an organization)
- community clouds (cloud infrastructure shared by several organizations)
- hybrid clouds (cloud infrastructure that combines two or more clouds)

The environment under consideration by the Open Cloud Standards Incubator includes all of these deployment models. The main focus of the Incubator is management aspects of Infrastructure as a Service (IaaS), with some common characteristics that might be applicable to other service stacks like the PaaS. These aspects include service-level agreements (SLAs), service-level objectives (SLOs), quality of service (QoS), workload portability, automated provisioning, accounting, and billing.

The fundamental IaaS capability made available to cloud consumers is a cloud service. Examples of services are computing systems, storage capacity, and networks that meet specified security and performance constraints. Examples of cloud service consumers are enterprise datacenters, small businesses, and other clouds.

Many existing and emerging standards will be relevant in cloud computing. Some of these, such as security-related standards, apply generally to distributed computing environments. Others apply directly to virtualization technologies, which are currently considered one of the important building blocks in cloud implementations. (The dynamic infrastructure enabled by technologies such as virtualization aligns well with the dynamic on-demand nature of clouds.) Examples of standards include SLA management and compliance, federated identities and authentication, and cloud interoperability and portability.

Figure 1 shows the scope of the Open Cloud Standards Incubator and the benefits of extending management standards.

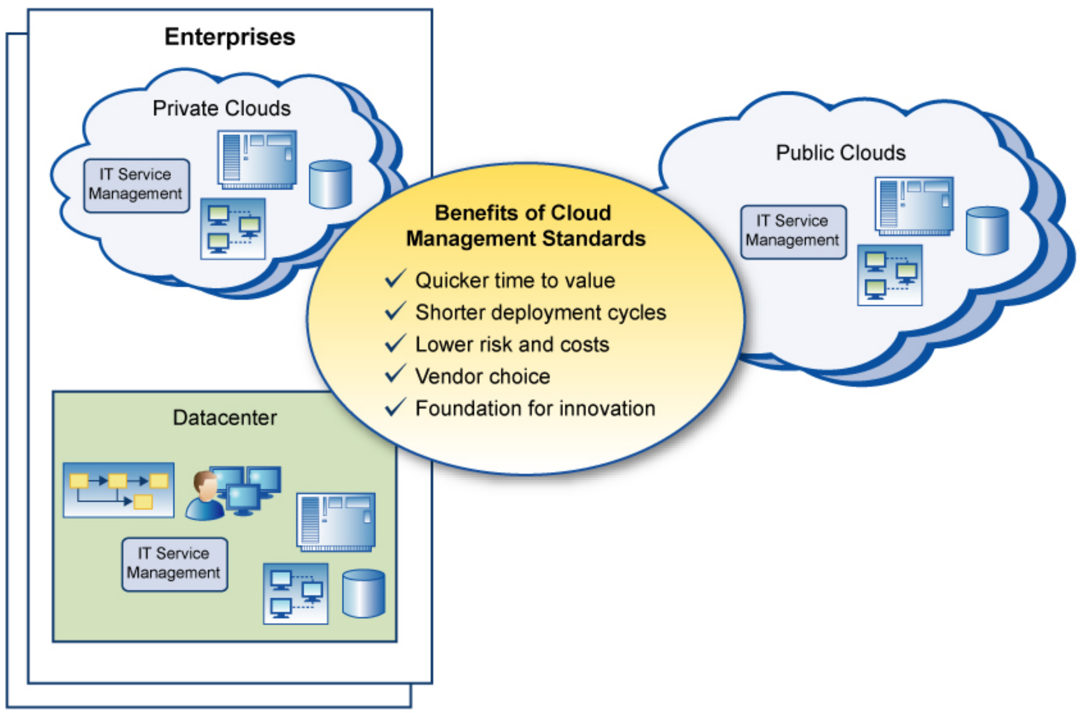


Figure 1 – Scope and Benefits of DMTF Open Cloud Standards Incubator

The Open Cloud Standards Incubator addresses the following aspects of the lifecycle of a cloud service:

- description of the cloud service in a template
- deployment of the cloud service into a cloud
- offering of the service to consumers
- consumer entrance into contracts for the offering
- provider operation and management of instances of the service
- removal of the service offering

When practical, existing standards (or extensions to them) will be integrated into the recommended solution. Examples of standardization areas include resource management protocols, data artifacts, packaging formats, identity protocols, key management protocols, audit formats, compliance formats, and security mechanisms to enable interoperability.

6 Reference Architecture Overview

The lifecycle narrative contained in this document cites example functional interfaces that cloud consumers need to establish with the cloud service provider. This section provides a cloud service reference architecture (Figure 2) that describes key components such as actors, interfaces, data artifacts, and profiles with an indication of interrelationships among these components. This section contains a high-level discussion of the architecture, and section 9 contains more details about an expansion of the architecture. The discussion herein provides a functional description that is non-normative and profitable for study in determining architectural elements that a cloud service provider should make available.

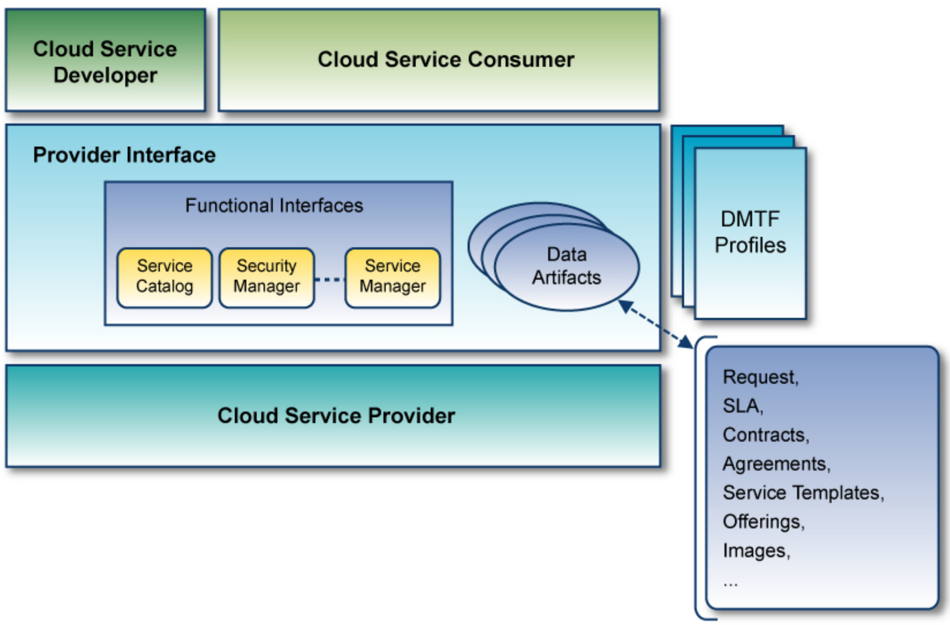


Figure 2 – Cloud Service Reference Architecture

6.1 Service Lifecycle and Use Cases

Figure 3 shows the six lifecycle states of a typical cloud service with the use cases that are most relevant to each state.

- **Template:** A developer defines the service in a template that describes the content of and interfaces to a service.
- **Offering:** A provider applies constraints, costs, and policies to a template to create an offering available for request by a consumer.
- **Contract:** A consumer and provider enter into a contract for services, including agreements on costs, SLAs, SLOs, and specific configuration options.
- **Provision Service:** A provider deploys (or modifies) a service instance per the contract with the consumer.
- **Runtime Maintenance:** A provider manages a deployed service and all its resources, including monitoring resources and notifying the consumer of key situations.
- **End of Service:** A provider halts a service instance, including reclaiming resources for redeployment to support other service.

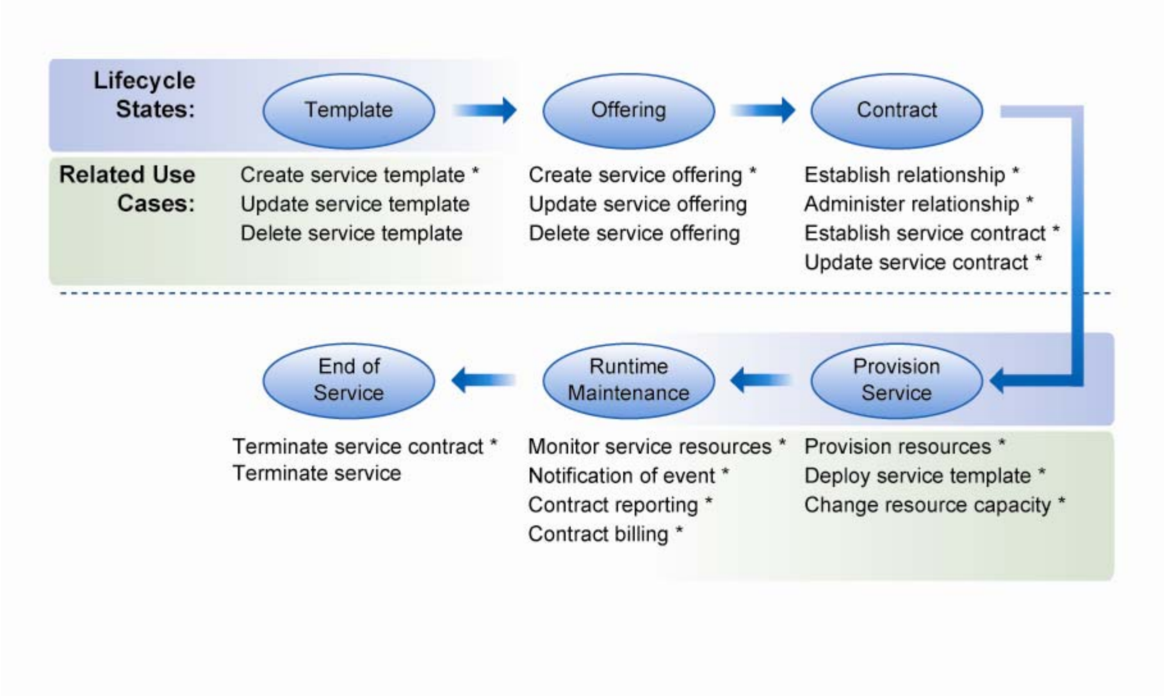


Figure 3 – Cloud Service Lifecycle States and Use Cases

The use cases with asterisks in Figure 3 are described in detail in the *Use Cases and Interactions for Managing Clouds* white paper (DSP-IS0103).

6.2 Resource Models

Any programmatic API has an underlying resource model, whether implicit or explicit. In the IT management domain, the practice has long been to make resource models explicit and clearly separated from the protocols used to manipulate model elements. The DMTF CIM model is an example of such a protocol-independent model, for which several access protocols have been defined, supporting various interaction patterns. For large domains, such as cloud computing, there is typically not just one resource model but a series of related models. The boundaries between them are established based on considerations of reusability and complementarity. Some entities (such as *customer*, *provider*, *service*, and *policy*) are applicable independently of the type of cloud resources provided, while others (such as *host* and *disk image*) describe a specific type of cloud resources. These type-specific model entities are grouped based on the likelihood of being offered and consumed simultaneously. For example, a *host* and a *volume* are typically considered part of the same domain of cloud resources (typically called IaaS — Infrastructure as a Service). Application data storage resources (such as a SQL engine or other structured data store) would typically be grouped as part of another resource domain. In addition to deciding how to group model entities, the other key design decision is the choice of a meta-model (for example, MOF or RDF) in which the model is represented.

After the meta-model and models have been determined, the way the resource model is made accessible through the management interface represents the next set of design decisions. These decisions depend on how model-centric the protocol is — whether the protocol granularity corresponds exactly to the resource model granularity (resources are individually addressable and the logical protocol endpoints correspond to the model entities) or whether the model is used to describe and document the operations in the interface but the operations can take place at a different level of granularity. These addressability/granularity decisions apply to data retrieval, data setting (if direct access is permitted), and invocation of operations. Other decision points include the interaction patterns and security requirements and are described elsewhere in this document.

6.2.1 Service Offering Model

To manage cloud resources of various types, a provider/consumer resource model must exist. That model describes the way in which an offering is presented and consumed. It can be abstracted from the specific type of cloud resources offered. The service offering model defines entities for the service consumer and the service provider. Service templates are used to describe in a general form what a provider can offer, and a specific provider turns a service template into a specific offering. A catalog or other mechanism is used to query and retrieve service templates and offerings. The consumer can request a service instance of a service offering. Resource model elements define the lifecycle of a request. The actual service delivery is modeled through domain-specific model entities, such as those in the IaaS model (see 6.2.3).

6.2.2 Identity Model

Although cloud computing brings new use cases and threats, it does not by itself require a new identity model. Existing enterprise identity models are appropriate but should be integrated, as discussed later in this document. What may change is the amount of auditing kept and the precision of the user definitions (individual user and organization). However, the introduction of a multi-tenant model as a cloud service provider business accelerator may require more consideration. The intermingling of identity and attribute information of multiple cloud service provider consumers will require careful data modeling to prevent accidental disclosure.

6.2.3 IaaS Model

The IaaS domain is usually defined to include resources of the following types: server, whether virtual or physical; storage; network connections; and IP addresses, public or private, exposed by the network interfaces. In addition, the resource model for the IaaS layer may include resources that represent additional features of the infrastructure, such as network features (for example, load balancing), templates (for deployment), and snapshots. Data storage elements (for example, volume) have quality-of-service elements of their own (for example, deduplication, encryption, backup), but these are usually captured in a separate resource model (for data and storage) outside of the scope of this work.

6.3 Interaction Patterns

An analysis of various cloud management use cases revealed that each use case could be decomposed into a combination of interaction patterns. An interaction pattern consists of a sequence of messages. For any interaction pattern, the specific messages may vary from use case to use case, but the messages have similar characteristics at an architectural level, particularly the protocol and security considerations.

Figure 4 shows the interaction patterns, grouped in four broad categories.

- **Identify:** A person or entity that interacts with the cloud service provider establishes their identity and receives appropriate credentials, such as a session token. An identity token may also be obtained through an external identity provider that has a trust relationship with the cloud service provider. Operations and data are made accessible to the connection authenticated by the credentials or identity token.

- **Administer:** These patterns work with the data that describe offerings, users, and other administrative metadata information needed for interactions with the cloud service provider. For example, an administrator or operator may browse a list of available offerings to select one, to update its metadata to configure it for a particular purpose, and to retrieve details about how to access instances of a service that is part of the offering.

- **Deploy and Update:** These patterns (there are actually two types of Negotiate/Provision Resources) are used when selecting services and resources, and then making them into services. Included are any needed negotiations about the amount and type of resource, operations to provision services including the infrastructure that supports them, and tracking the status of what may be long-running operations.
- **Steady State:** These patterns are used after services and resources have been provisioned and are in use. They include client-initiated requests such as a report request and notifications from a provider about situations that are of interest to a consumer and that may require remediation actions.

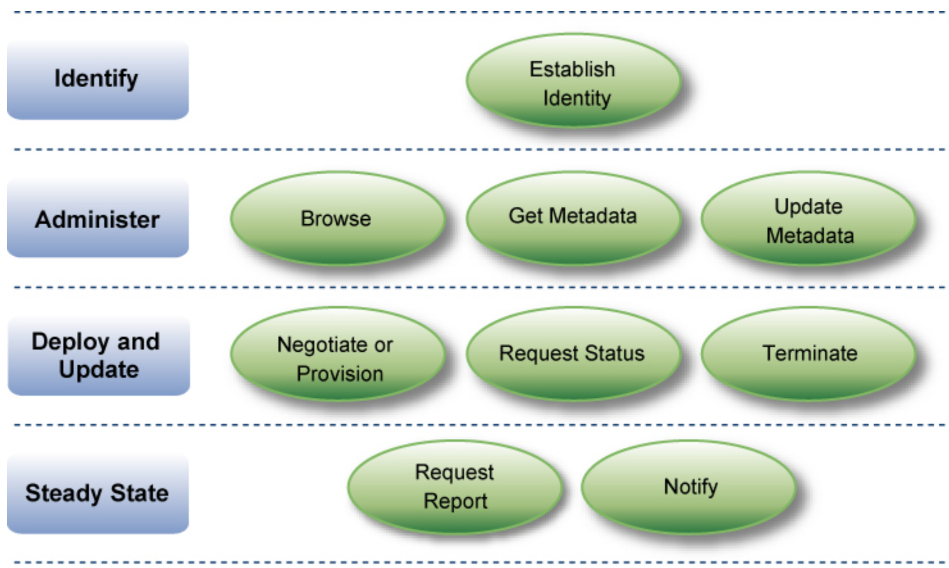


Figure 4 – Interaction Patterns

Figure 5 shows an example of how interaction patterns can be combined to represent a use case. Provisioning a service follows the following sequence:

1) Establish the identity of the accessing entity.
2) Browse available offerings.
3) Submit and execute a request to provision a service, including supporting resources. The request may require negotiation before the consumer and provider agree to an acceptable request.
4) The status of a possibly long-running provisioning process may be requested while in process.
5) The consumer gets metadata about the provisioned service, such as the network addresses to access service endpoints and resources.

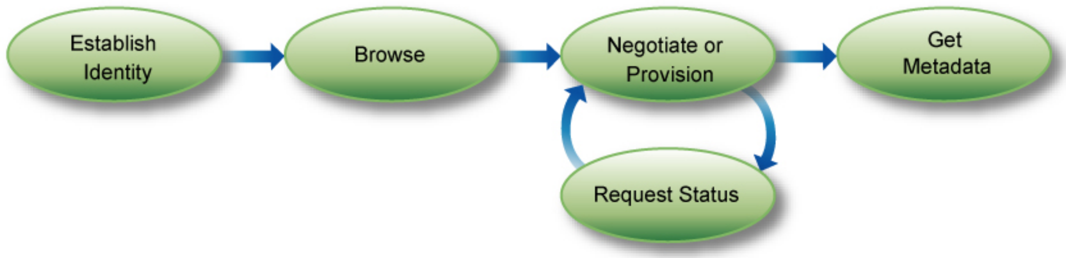


Figure 5 – Example of Interaction Patterns for a Provision Service Use Case

6.4 Security Architecture

In general, we are applying appropriate security primitives to the interfaces, the services, and the objects in the control and management planes. Because this work is focused on management, the security of the data plane (for example, the security of the run-time interactions between the users and the applications running on the IaaS platform) is out of the scope of the provider interface. Figure 6 shows the security context, the flow of information through the cloud service provider interface, and the objects secured.

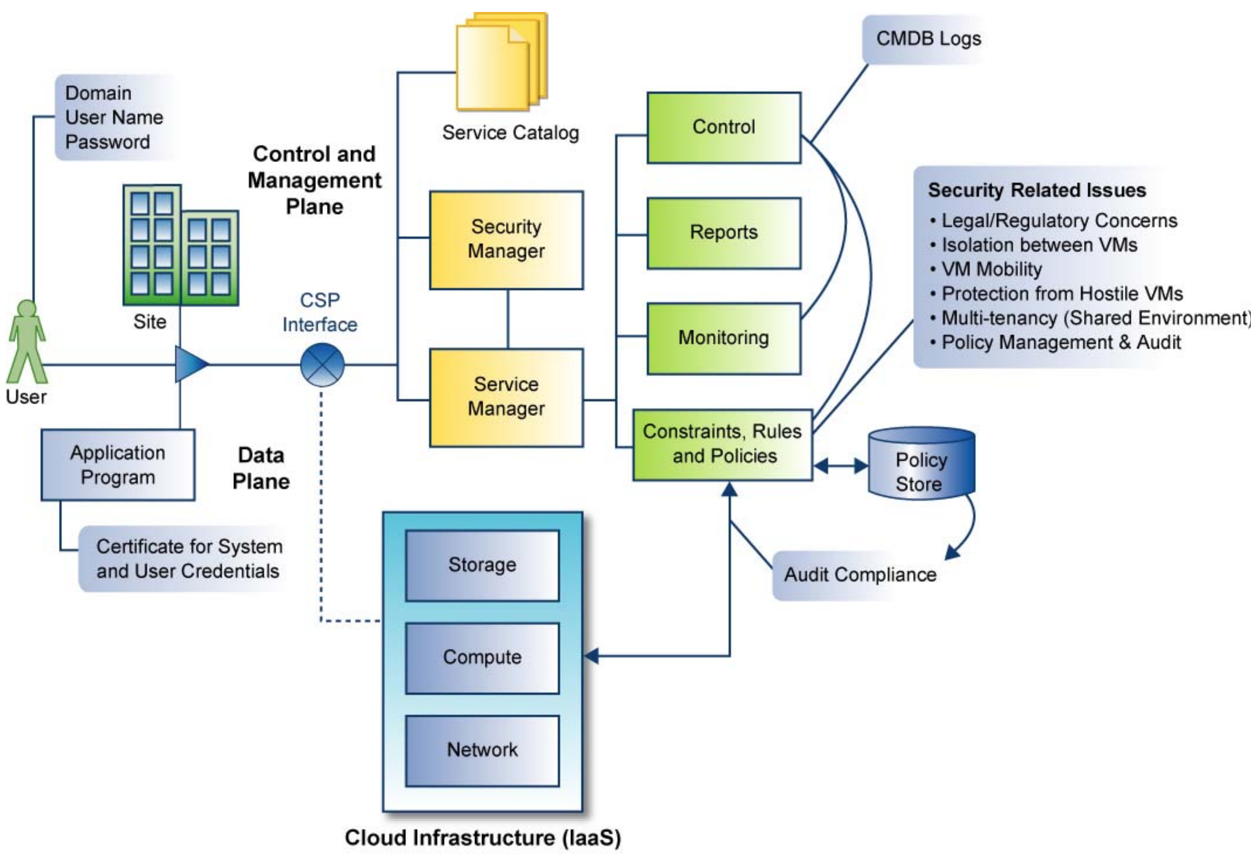


Figure 6 – Security Context

The cloud service provider (CSP) interface provides access to the logical endpoints, including the security manager, service manager, and the service catalog. These endpoints provide the various services to interact with service entities (such as VMs, volumes, networks, and composite applications), get audit reports, and perform a host of other activities required to fulfill and maintain a cloud infrastructure.

The major categories of objects that are managed by the service manager are control, monitor, and report objects, access to which may be controlled by role-based access control (RBAC).

The Constraints, Rules, and Policies objects are consumed by the cloud infrastructure, and the function of the provider interface is to manage the content of these policy-related elements. The cloud infrastructure (IaaS) is a “black box” to the provider interface, and how the policies are implemented is left to the cloud service provider. But the fact that the various constraints, rules, and policies are being implemented is verified by the audit events.

The two categories of actors who interact with the CSP interface are human users and application programs (such as management, automatic provisioning, billing, or audit applications). The human user might also be interacting through a portal interface, usually using a web browser. The portal interface will be developed using the cloud service provider interfaces. Both actors would be authenticated at the CSP interface by the security manager or present an identity token to the security manager.

Traditionally, the human user uses a user name and password as credentials for authentication; however, stronger mechanisms (identity federation and assertion provisioning) should be used. Commonly, an application program uses certificates. However, it is deemed insecure to embed user names and passwords in application programs. In this case as well, tokenized identity can be profitably used to provide a higher standard of security.

These are common examples, but they are not prescriptive. Humans might use authenticator tokens, for example, or applications might use Kerberos tickets. Appropriate mechanisms may vary in different environments. Trust relationships may be employed to strengthen the authentication and authorization mechanisms. Our intention is not to declare an exhaustive list of acceptable authentication mechanisms, but to establish a good starting point.

6.5 Role of Rules, Policies, and Constraints

Policies and constraints are expected to play a major role in the management of clouds. Given the focus on the interfaces for managing clouds rather than the internal details of a cloud implementation, a useful abstraction is to define the desired capabilities of the cloud using policies and constraints. These are interpreted to be *what* the cloud service provider should offer rather than *how* it offers it.

There will be several types of policies, such as service-level agreements (SLAs), service-level objectives (SLOs), deployment constraints (for example, those in OVF, now and proposed), data residency constraints, auditability constraints, security constraints, and so on. After a consumer or developer and a provider agree to a set of constraints, these constraints will govern how each (especially the provider) acts. Figure 7 shows a model of the flow of constraints and rules.

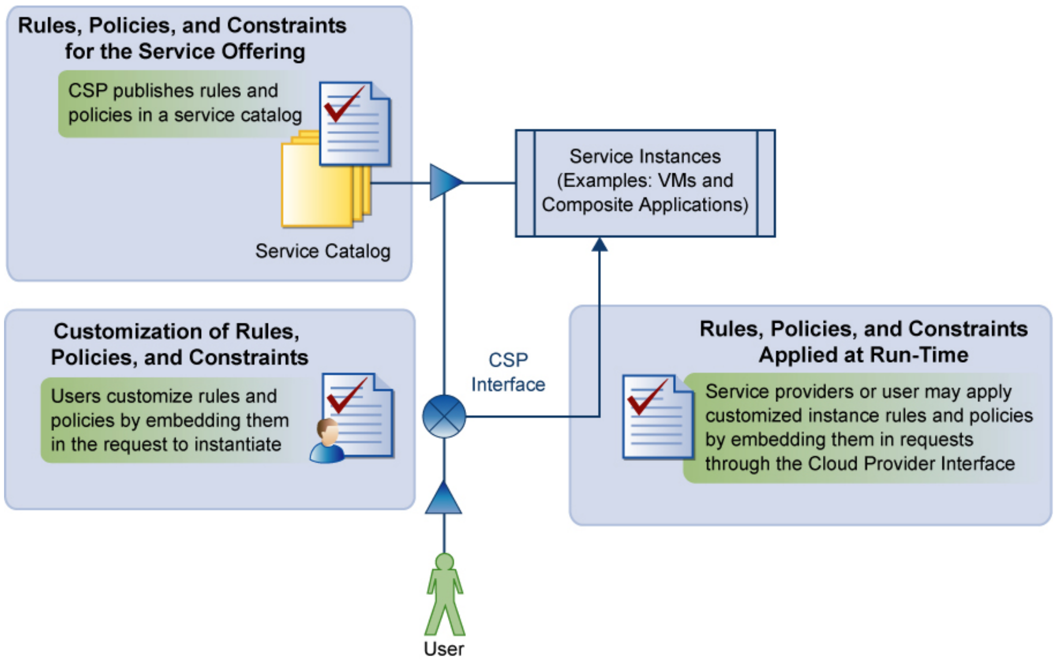


Figure 7 – Constraints Flow

The cloud service provider publishes various relevant constraints, rules, and policies and may include them as a part of the service offering in the service catalog. These become part of the instance when a service entity (for example, a VM or a composite application) is instantiated from the template.

The user can customize the constraints, policies, and rules as a part of the service request to instantiate a service. Users might not be able to customize some policies and constraints from the service provider. The instantiate request will include sections to add and modify the constraints bundle. Then the instance will have an aggregate of the constraints, rules, and policies. The modified constraints have to be applied to the instance throughout its lifecycle (for example, suspend/resume or migrate).

After a service instance (for example, a VM or a composite application) is running, the CSP or user may request to change the policy set, within limits. It may not always be appropriate or feasible for a CSP to change the policies applied to a live service entity, which might impact the characteristics that were agreed upon before instantiation.

Examples of constraints are security policies. A number of different security policies can be applied to offerings in the service catalog or instances. A few simple examples follow:

- **Access control**. A service instance may be accessible only to a subset of cloud users (for example, users belonging to an enterprise or users with appropriate authorization levels). In addition, policies may specify which users are entitled to modify an instance and where and when they can deploy it.
- **Network security policies**. Such policies may specify how a service instance in a template may be connected with other service instances or resources outside the cloud. These could take the form of traffic filtering (for instance, firewall) rules and requirements for configuring traffic inspection for intrusion detection and prevention.
- **"Scope" of the security policies**. The offering may specify where instances may run; for example, they may be constrained within a particular cloud service provider's infrastructure, enterprise zone, or geographic region.

7 Requirements

This section describes the high-level requirements that a cloud service provider should make available as a part of the cloud service offering. The list should not be considered exhaustive but rather indicative of the types of functionality that should exist in a well-rounded and exemplary cloud service provider offering.

7.1 Protocol Stack

The following requirements are related to the protocol stack:

- The interface should support all Message Exchange Patterns (MEPs) relevant to this domain. This includes one-way MEP (push and pull) and two-way MEP (synchronous and asynchronous) over a request-response transport like HTTP.
- Programmatic access to various cloud provider services should be provided using industry-standard mechanisms such as REST, SOAP, and so on.
- Payloads may vary dramatically in size from a few bytes to many gigabytes, and may flow between a consumer or developer and a provider in either direction. Appropriate choices must be made based on the situation, and the protocol should enable any of these choices.
- The stack should be able to carry additional contextual information such as authentication information.
- The stack should adapt to client capability, such as availability of client address, payload size, processing duration, and throughput, and use the most appropriate message pattern.
- The stack should support client- or server-initiated cancellations and restart of a large transfer.

7.2 Resource Model

The following requirements are related to the resource model:

- The resource model should be separated from the interfaces and mechanisms. Separating the resource model from the protocol allows the consumer (and provider) applications to be isolated from the protocol. As long as the applications' features are implemented based on the resource model (entity types, corresponding data, associations between them, operations), they can be mapped to any protocol over which the management interactions take place. This separation also allows new entities, from contiguous domains, to be added to the resource model without needing to change any part of the protocol design and implementation.
- The resource model should support instantiating and addressing a large number (100,000) of artifacts.
- Any resources exposed at the provider interface may need to be individually addressable, at least for purposes of identification, even if a resource does not support direct operations. The access to various resources should have appropriate authentication (for example, discoverable resources might need to expose metadata without authentication).
- Many actors may need access to resources. They will have varying entitlements, and the model must support authorizations at all levels. Policies such as role-based access controls (RBACs) should authorize access to and control all the resources, with visibility rules enforceable based on context. Examples of the resources and capabilities whose access should be controlled include the ability to create and manipulate aggregates (like virtual datacenter, virtual application stack elements like VMs, network and storage, images, templates, offerings and so on), the ability to monitor resources, and the ability to ask for and receive audit information about these resources.
- The metadata endpoints need not be same as the resource endpoints. For example, an audit report might be a database rather than the virtual datacenter resource, or a power-on-VM interface might

not need to connect directly to a VM but rather to a resource representation of the VM (for example, a message queue).

- If versioning is supported, the resource model should have the capability to create and read versions of all the persistent resources, particularly those that are accessed by multiple actors and are mutable.
- Providers must be able to expose resource metadata that may be needed by clients, such as the geographical location in which data is stored.

7.3 Adoptability

The following requirements are related to adoptability:

- The service offerings should integrate with commonly found IT mechanisms, such as authentication and networking.
- The service offerings should use easily and widely understood programming models and programmer skills.
- The service offerings should be able to federate with other clouds, especially those that support the same standards-based interfaces.
- Standard syntax, open APIs, and open standards should be used whenever possible.

7.4 Internationalization

Given that clouds are easily accessible world-wide, the interface should impose no restrictions on the location and language of users, managed resources, and data. While a provider may implement restrictions, the interface should not constrain the provider's choices.

7.5 Rules, Constraints, and Policies

The following requirements are related to rules, constraints, and policies:

- The cloud service provider should allow the definition of policy per consumer that will define how service entities (virtual machines, network conductivity, storage, and so on) are to be treated by the cloud service provider on behalf of the consumer.
- Whenever services are deployed by the consumer in the cloud service provider infrastructure, the policies and rules imposed by the cloud service provider must be augmented by the policy of rules specified by the consumer before making the images operational in the cloud service provider’s infrastructure. If there are any conflicts concerning policy specifications between the cloud service provider and the cloud service consumer, those conflicts should be communicated to the cloud service consumer for resolution.
- The system shall have capabilities to declaratively specify the various relevant constraints through exchange of rules and policies and have the ability to verify the implementations of the said constraints through audit reports. The various security threats will be mitigated by specifying policies and then inspecting auditing information. How these are implemented is beyond the scope of the provider interface.
- The rules and constraints should include:
  - elasticity rules and constraints for compliance in terms of geographical constraints
  - adjacency rules regarding data and processing adjacency
  - performance affinity rules between VMs, applications, and so on
  - anti-affinity rules for PCI compliance

- In the case of a virtualized infrastructure, there should be mechanisms to capture the hypervisor security capability as well as to push policies and rules to reflect the VM security primitives. The service catalog and offering should reflect these rules and constraints.
- The interface should be capable of multi-tenant security specification of items including resource and data isolation, network isolation with security of virtualized networks, and so on.

7.6 Cloud Management Interface Security

The following requirements are related to cloud management interface security:

- Access to the cloud service provider interface endpoints and controlled resources should be secure through standardized techniques. (Examples of endpoint security for cloud service providers include providing protection for REST-based URLs, URL routers, and so on.)
- Programmatic and manual access to cloud provider services and content flow should be protected by using industry-standard protection mechanisms such as SSL, TLS, and so on.
- The cloud service provider should assure that all access or changes to cloud services, resources, and data produce auditable records regardless of success or failure, and that these audit records can be compiled into a consistent audit trail that can correlate to end user- or service-initiated actions. Audit records should include clear indication of any delegations of identity or authorizations.
- In cases where anonymous access is required, anonymous access should be allowed only through the security manager. These cases should be recorded as exceptions and logged in the audit database with clear descriptions of reasons for the exceptions and actions taken by the security manager.
- The cloud service provider should provide services that establish trust relationships between itself and customers or other service providers using standardized techniques; these standard mechanisms include the exchange of certificates, cryptographic materials (for example, keys), and root identities, as well as security policies that can be used to establish subsequent trust relationships and policies.
- The cloud service provider should provide identity services that permit customers to provision and manage identities that can be authenticated and authorized to access cloud services using standardized mechanisms. These services should consider the need to provision identities either manually or by automated means and the need to synchronize or collaborate with external identity provider services.
- The cloud service provider should allow its customers to control and manage the cryptographic materials and methods used to protect its applications and data, whether in motion or at rest, when entering, exiting, or residing within the cloud service provider's infrastructure. The minimum functionality would allow an administrator to enter this information manually, with more functionality allowing secure synchronization.
- Cloud service providers should provide industry-standardized entry points into their identity provider service. Such entry points allow for consumers to establish appropriate trust relationships with a cloud service provider.
- The cloud service provider should provide services that can authenticate the identity of customers or services and produce security information that can be consumed by cloud services to enforce security policy and access control.
- Identity functionality provided by the cloud service provider may allow for the delegation of identity and authorization information using standardized mechanisms such as authentication tokens. Any representation of identity and authentication information, either direct or delegated, should be secured and provided for. Delegation gives an account holder the ability to grant access to someone who does not have an account on the service by giving them time-limited access to a specific resource. Part of delegation is also auditing and keeping track of usage from these delegated

accesses. Further capabilities may include the ability to create an account for these accesses so that the delegated client can be billed for their usage.

7.7 Data and Storage

The following requirements are related to data and storage:

- The cloud service provider should provide customer transparency regarding how data integrity is maintained throughout the lifecycle of the data.
- Encryption and key management should use industry and government standards. The cloud service provider should provide role management and separation of duties for data access controls. Encryption and key management should be able to handle data isolation for multi-tenant storage and separation of customer data from operational data from the service provider.
- Sensitive customer data should be encrypted in transit over the cloud service provider's internal network, in addition to being encrypted at rest. The transport access to the provider interface may be secured by TLS/SSL with a server certificate, and for increased security, mutual authentication at the TLS layer may be optionally implemented, based on client certificates.
- In accordance with regulatory requirements, the cloud service provider should provide appropriate notification to data owners when data can be (or has been) seized by a third party (for example, a supply chain or the government) for content discovery.
  - Whether data can be seized by a third party could be part of the definition of the provider's offering, and no notification might be meaningful there.
  - When data is seized, the decision to notify the owner of the data should be based on the provider's governance.
- Data backup and recovery schemes for the cloud must be in place and effective in order to prevent data loss, unwanted data overwrite, and destruction.
- Geographic data location should be exposed to the cloud consumer where appropriate. Knowing where the cloud service provider will host the data and implement required measures ensures compliance with local laws that restrict the cross-border flow of data.
- Data retention and secure destruction capabilities should be provided. The service provider should be able to completely and effectively locate data in the cloud to be removed and provide a level of assurance and evidence regarding data destruction.

7.8 Logs, Event Management, Incident Response, and Notification

The following requirements are related to logs, event management, incident response, and notification:

- Cloud service providers should provide incident and alert information to the customer by using defined interfaces.
- Cloud service providers should maintain the confidentiality of customer incident data.
- The event management interface should have rich semantics with identifiable data sources (application logs, firewall logs, IDS logs, and so on) so that the cloud consumers can make inferences about their systems. The events and information should enable the cloud service consumer for the verification of billing, the providing of audit events, verification of regulatory compliance, the monitoring of operational characteristics, and so on.
- During the operation of service entities (for example, VMs or other services) in the cloud, the cloud service provider should be monitoring the service entities to log events concerning operations, billing, compliance, and so on. The cloud service provider should make this information available to the cloud service consumer in accordance with the SLA in real-time, batch mode, or both.

- The event interface should also be versatile enough to address incident notification, incident response, containment, and remediation.
- There should be error handling mechanisms, with error messages that are descriptive.
- Operation logs should be available in accordance with the SLA that defines the log persistence duration.
- Appropriate log preservation and integrity mechanisms, such as digital signature and storage devices with write-once media, should be employed.

7.9 Audit, Legal, and Compliance Monitoring

The following requirements are related to audit, legal, and compliance monitoring:

- Cloud service providers should include metrics and controls that customers can leverage to implement their information risk management requirements.
- Cloud service providers should provide evidence on how customer security requirements are being met.
- Cloud service providers should provide the interfaces to perform an external audit, where appropriate.
- Cloud service providers should comply with appropriate auditing standards, such as SAS70, as required by the business models and industry regulations.

7.10 Security Considerations for Virtualization Technology

The following requirements are related to security considerations for virtualization technology:

- Administrative access and control of virtualized operating systems should integrate with strong authentication, identity management, RBAC (with separation of duty — SoD), as well as logging and integrity monitoring tools.
- The cloud service provider should expose capabilities for segregating the VMs and creating security zones by type of usage (for example, desktop versus server), production stage (for example, development, production, and testing), location of the VM, and sensitivity of data.
- The reporting mechanism through the event management interface should provide the information necessary to show data isolation and separation-of-duty in a multi-tenant environment.
- The cloud service provider should allow for the definition of virtual machines and application services. The definition of such virtual machine images and services must be protected so that, if required by the consumer, the images are protected from other consumers' view and reference.

7.11 Portability and Interoperability for Secure Migration

The following requirements are related to portability and interoperability for secure migration:

- Service entities (for example, VMs) should be able to migrate across organizational and ownership boundaries (for example, between an enterprise and a service provider's IaaS infrastructure).
- In the case of a virtualized infrastructure, VM migration should address secure deprovisioning (removal of the VM image after it is ported to a different location or service provider) and partial migration (cloud burst: secure integration between old and new locations and service providers).
- Service providers should provide assurance on the consistency of control effectiveness, management, monitoring, and reporting interfaces and their integration across old and new locations and providers.

- If storage migration capabilities are provided, the service provider should have verified functionalities for secure data transfer including encryption, access control, key management, decommissioning of storage devices, and destruction of data after migration.

8 Protocol Examples

A discussion on cloud infrastructure needs to include rendering of the identified uses cases into commonly used protocols. During the analysis work undertaken to identify requirements from an infrastructure standpoint, different approaches to analysis were explored. A set of criteria roughly divided into the following different categories was used to analyze every use case:

- interaction characteristics
- resource requirements
- deployment requirements
- security requirements

During the analysis work for use case interactions, it was observed that a simpler set of interaction patterns were repeating in different combinations of all the use cases. It was further observed that a set of nine smaller patterns combined in various ways could render every use case that had been identified. These building blocks were termed Semantic Interaction Patterns (SIPs). As an example, Status Request is involved in many use cases and hence is considered a separate interaction pattern. Some others include Resource Negotiation and Change Notification, which get used in various use cases. The SIPs were then used in the infrastructure analysis based on the preceding categories of criteria. This section focuses on the patterns of message exchange and provides examples of how to render these patterns into commonly observed protocol styles. Using HTTP as the transport protocol, from among various styles of protocol rendering, two popular styles in IT management domain are considered: Remote Procedure Call (RPC) and Representational State Transfer (REST).

Some distinguishing features of each of these styles are as follows:

- RPC style assumes that each resource class provides a specific interface for manipulating the resource. Two variants can be considered: RPC over HTTP, and SOAP RPC. Both use an XML payload for method invocation and use the HTTP POST method to deliver it.
- REST style uses a hypertext driven paradigm with protocols such as HTTP with standard methods GET, POST, PUT, DELETE, and others as the interfaces to a resource. Each resource is identifiable with a URI.

8.1 Message Exchange Patterns

Two message exchange patterns, or MEPs, can be used to render a semantic interaction between a cloud user and cloud service: a one-way MEP, and a two-way MEP. Furthermore, to reflect various connectivity constraints or user profiles, it is useful to define “execution modes” for these MEPs that can be associated with specific requirements or environment constraints. This section illustrates a few execution modes as applicable to one-way and two-way MEPs. The execution modes pertain to the two actions “push” and “pull” that the sender and the receiver may take in order to complete the interaction. It should be noted that *sender* and *receiver* are generic terms and that the entity sending back the response may not be the same addressable entity that received the request.

The protocol options show a few variants in using HTTP. A protocol solution has several aspects: RPC versus Document styles, RPC versus REST, plain HTTP with REST-style versus SOAP-based solutions. The focus here is on REST versus RPC. Some SOAP-based solutions that do not rely on RPC (for example, document/literal SOAP style) are not described here.

8.1.1 One-Way MEP

Execution modes for one-way MEP, as illustrated in Figure 8, are as follows:

- The **One-Way/Push MEP**, or “push” mode, where the sender takes the initiative of sending the business message to the receiver (for example, over an HTTP request). This mode assumes that the receiver has an IP address. An example would be a change notification interaction where the notified party — the user — is addressable.
  - Typical steps by a client and service for a one-way/push MEP using RPC:
    1) The client (here the sender) issues an HTTP POST request with an RPC payload that contains the object instance ID, method call, and parameters.
    2) The service sends an HTTP response — a status code that indicates success or failure. No RPC payload is expected.
  - Typical steps by a client and service for a one-way/push MEP using REST:
    1) The client (here the sender) issues an HTTP command (PUT, POST, or DELETE) to a URI that represents the particular resource.
    2) The service responds with an HTTP success or failure response.

  See Annex A for protocol examples using HTTP.

- The **One-Way/Pull MEP**, or “pull” mode, where the receiver takes the initiative of pulling the business message from the sender (for example, by initiating an HTTP request that contains a pull signal for a particular queue or mailbox). The response is sent over the HTTP response. This mode allows receivers that do not have an address to receive business messages.

  An example would be a change notification interaction in which the notified party — the user — is not addressable and must pull notifications that are obtained on the protocol back-channel (for example, HTTP response).

  In the pull mode, the receiver might periodically send several pull signals before a successful MEP is complete (for instance, including a business message on the response leg). These pull signals are typically scheduled automatically within the underlying messaging layer (for example, using WS-MakeConnection [*Web Services Make Connection (WS-MakeConnection) Version 1.1*], issuing an HTTP GET, or using a PullRequest signal in *OASIS ebXML Messaging Services Version 3.0*. Such pull signals contain only the absolute minimal information necessary to pull (for example, a queue ID, a channel ID) and no other business information.

  Here it is assumed that the client can provide a client side URI prior to or as a part of the call that is issued. In other pull variants, a queue ID or mailbox ID is provided to the client for use in all its subsequent pulls from the service.
  - Typical steps by a client and service for a one-way/pull MEP using RPC:
    1) Optionally, the client receives a notification with an RPC payload on the URL provided with an event ID.
    2) The client issues an HTTP GET or POST with an event ID (or a queue ID) as a parameter to get the details.
    3) The responding party sends back a payload about resource information that corresponds to the event ID.
  - Typical steps by a client and service for a one-way/pull MEP using REST:
    1) Optionally, the client receives a notification on the URI of the event.
    2) The client issues an HTTP GET of the event’s URI.
    3) The responding party sends back a payload about the resource identified by the URI.

See Annex A for protocol examples using HTTP.

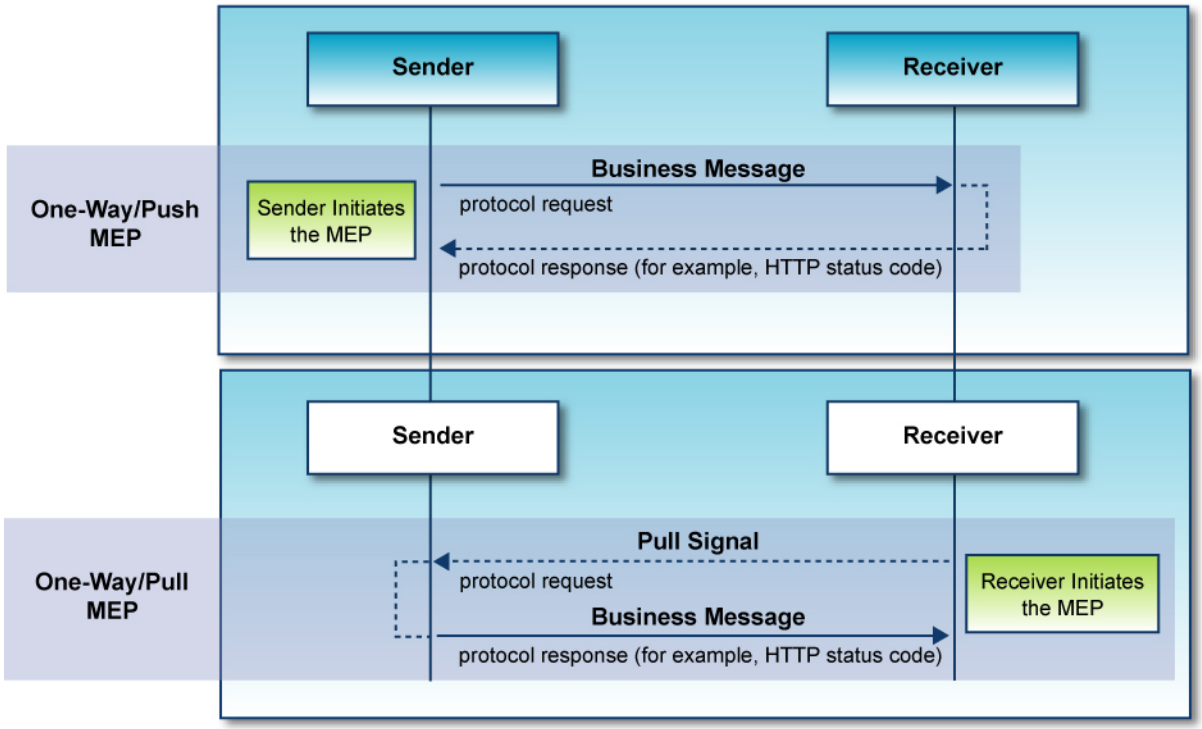


Figure 8 – One-Way MEP Execution Modes

8.1.2 Two-Way MEP

Execution modes for two-way MEP are as follows:

- In the **Two-Way/Sync MEP**, or "sync" mode, the sender takes the initiative of sending the business request to the receiver and gets the business response over the same connection (in case of a request-response transport such as HTTP). This mode does not require the sender to be addressable. An example would be requesting status interaction, assuming that the generation of the status information is quick and can accommodate a single HTTP request-response exchange. Figure 9 shows an illustration.

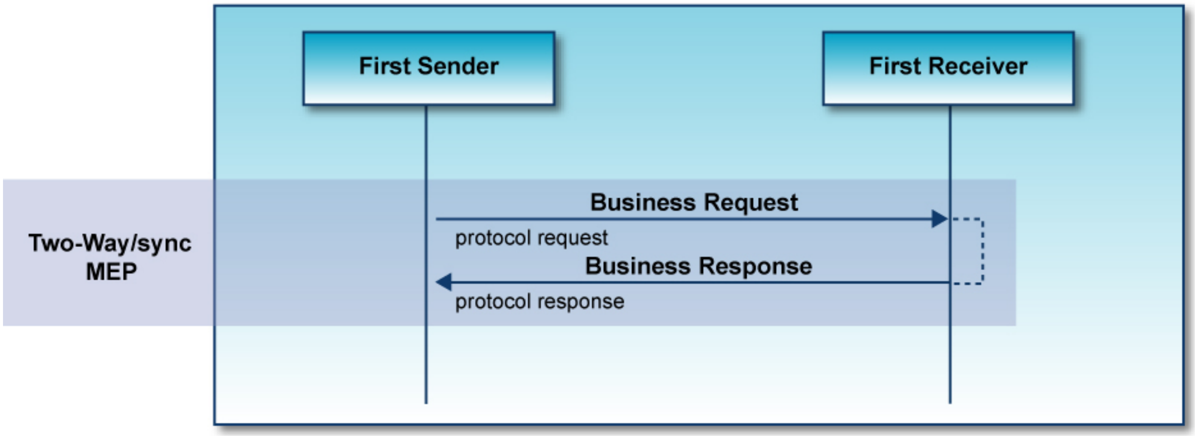


Figure 9 – Two-Way/Sync MEP Execution Mode

– Typical steps by a client and service for a two-way/sync MEP using RPC:

(This is quite similar to the one-way/push MEP example. The primary difference is that this execution mode contains an RPC response payload.)

1) The client issues an HTTP POST request with an RPC payload that contains request data such as the object instance ID, the method call, and parameters.

2) The client receives an HTTP response with the RPC payload if the method succeeds, or an HTTP error code failure if the method fails.

– Typical steps by a client and service for a two-way/sync MEP using REST:

1) The client issues an HTTP command (GET, PUT, POST, or DELETE) to a URI that represents the particular resource. The client expects to receive a business response (which is different from a one-way/push MEP).

2) The client receives an HTTP response with a payload that represents resource data if the method succeeds.

See Annex A for protocol examples using HTTP.

- The **Two-Way/Push-and-Push MEP** is an asynchronous mode, where the business request and the business response are sent over different connections (for example, both pushed over a separate HTTP connection). Both sender and receiver are supposed to be addressable. This mode is useful when the response requires some computation time. An example would be a simple resource-negotiation interaction, in which the cloud service may take some time to allocate requested resources, beyond the delay tolerated for a single HTTP request-response. In that case, the confirmation is sent over a callback. Figure 10 shows an illustration.

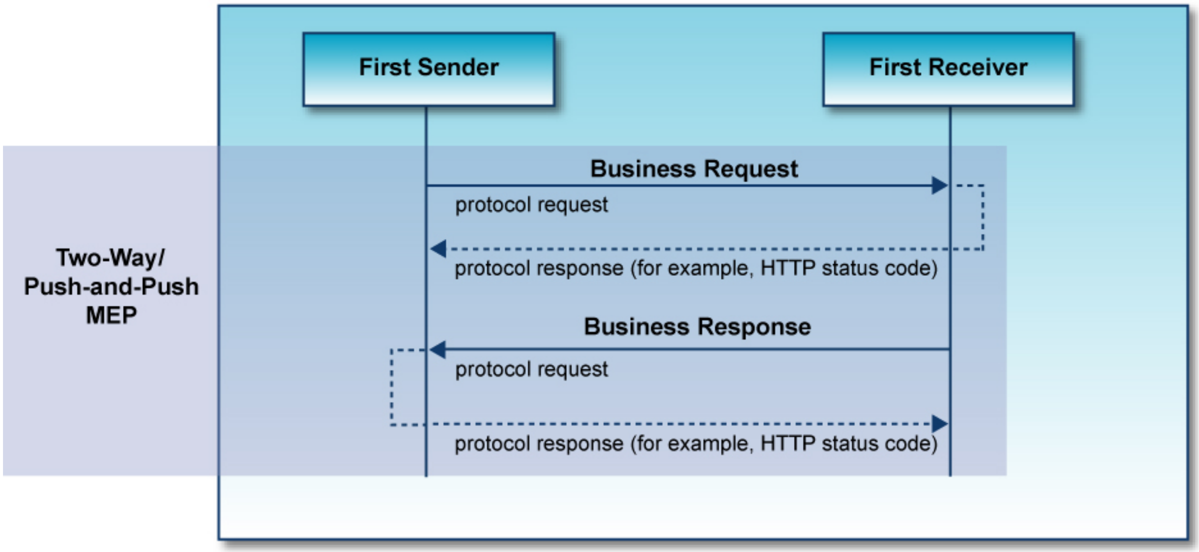


Figure 10 – Two-Way/Push-and-Push MEP Execution Mode

- Typical steps by a client and service for a two-way/push-and-push MEP using RPC:
  1) The client issues an HTTP POST request with an RPC payload that contains request data including the client's addressable endpoint.
  2) Optionally, the service responds with an identifier that represents the work that is proceeding in response to the request.
  3) Sometime later, the service issues a call-back (for example, through HTTP POST) on the client URL with the payload that contains the response to the initial Method Call.
- Typical steps by a client and service for a two-way/push-and-push MEP using REST:
  1) The client issues an HTTP command to the URI that identifies the resource.
  2) Optionally, the service responds with a URI from which the client can obtain status information.
  3) Sometime later, the service issues a call-back (generally through HTTP POST) on the URI that the client registered.

See Annex A for protocol examples using HTTP.

- The **Two-Way/Push-and-Pull MEP** is an asynchronous mode, where the sender takes the initiative of both exchanges: pushing the business request and pulling the business response. The advantages over the "push-and-push" mode are that the sender is not required to have an addressable endpoint and can decide when it is ready to transfer the response. An example would be a simple resource-negotiation interaction, in which the cloud user is not addressable, and needs to pull the confirmation message later. Figure 11 shows an illustration.

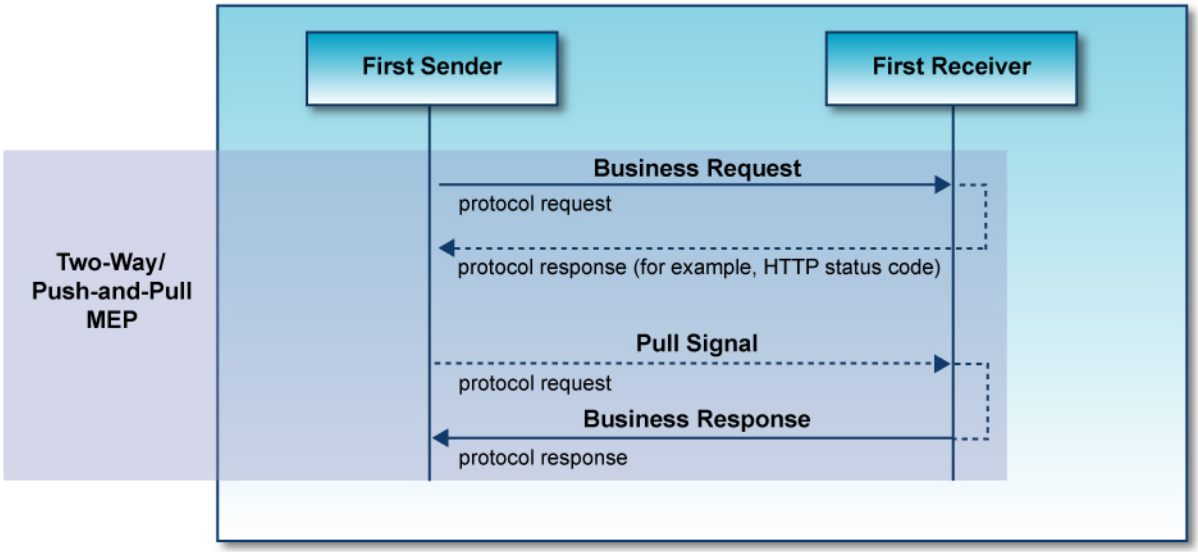


Figure 11 – Two-Way/Push-and-Pull MEP Execution Mode

– Typical steps by a client and service for a two-way/push-and-pull MEP using RPC:

1) The client (here "First Sender") issues an HTTP POST request with a payload that contains the object instance ID, the method call, and parameters.
2) Optionally, the service (here "First Receiver") responds with a payload that contains an identifier to be used by the client in a subsequent query, unless the ID is already agreed upon.
3) The client issues an HTTP POST request with a payload that contains the ID received in the response to the initial request.
4) The service sends back the business RPC payload over the HTTP response. It sends back an empty payload if no message is available for pulling.

– Typical steps by a client and service for a two-way/push-and-pull MEP using REST:

1) The client (here "First Sender") issues an HTTP request to the URI that identifies the resource.
2) Optionally, the service (here "First Receiver") responds with a payload that contains a URI to be used by the client in a subsequent request.
3) The client issues an HTTP GET request to the URI.
4) The service sends back the business payload (for example, a notification) over the HTTP response, or the service may simply return an HTTP respond code without any payload.

See Annex A for protocol examples using HTTP.

8.2 Infrastructural Aspects Affecting the Choice of MEP and Its Execution Mode

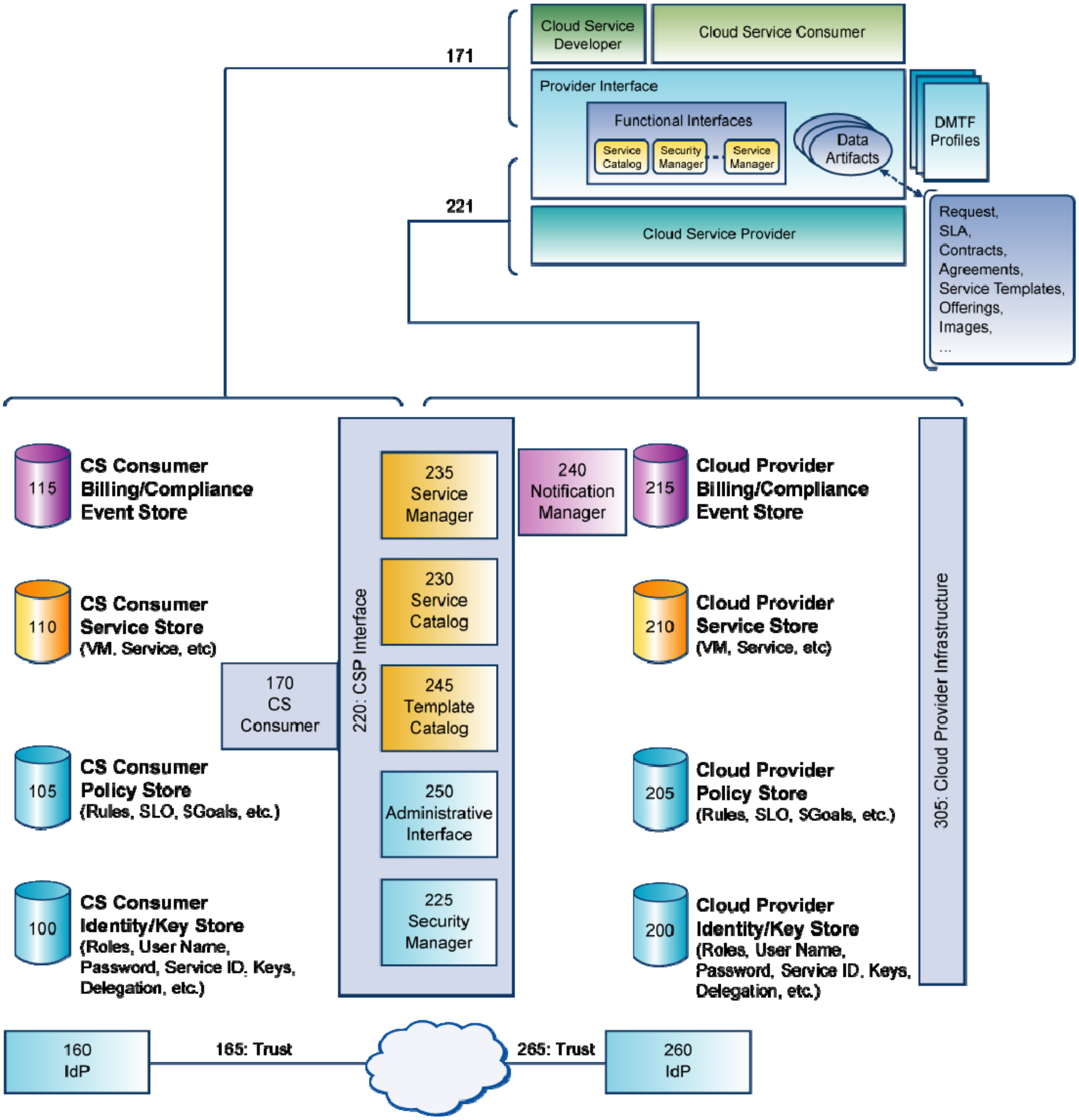
Some infrastructural aspects have more of a bearing on the choice of the MEP than others.

- The payload size informs whether an interaction is likely to be short or long, and correspondingly whether a synchronous or an asynchronous pattern of interaction is preferable.
- The amount of time that is likely to be taken by the backend in the processing affects the choice of execution mode. The time that it takes to execute an operation may be completely independent of the payload that it takes to execute it. A long delay may prohibit use of a two-way/sync MEP because it may exceed the timing tolerated for an HTTP request-response. However, if the response time is constrained to be short (for example, by SLA), then a two-way/sync MEP can be used.
- Client addressability affects the choice of execution mode. Only if the client is addressable by the cloud can the two-way/push-and-push MEP (for instance, with “callback”) or the one-way/push MEP directed to the client be used. If the client is *not* addressable, it will have to pull all asynchronous responses to its requests.
- Finally, the ability to browse or enumerate a large collection also has a bearing on which MEPs to use for browse operations. This is a variation of the first bullet, though there may be different processing time characteristics because of namespace enumeration operations.

9 Cloud Ecosystem

This section expands on the reference architecture described earlier in this document and rationalizes the architecture with data artifacts described in the *Use Cases and Interactions for Managing Clouds* white paper (DSP-IS0103). Familiarity with DSP-IS0103 is required to fully understand this section’s contents. By understanding this section, the reader can expect to come to an understanding of the architectural interactions and functional capabilities that would be needed by an offering from a cloud service provider. The information contained here should be considered referential only and non-normative.

Figure 12 is provided to help the reader understand the relationship of the architecture presented in *Interoperable Clouds* (DSP-IS0101) and the more detailed architectural diagram and discussion presented in Figure 13. In the area labeled 221, Figure 12 shows that the provider interface offered by the cloud service provider is represented by elements labeled 225–250, and the cloud service provider is represented by the areas labeled 200–215, 240, and 305. The area labeled 171 shows that the cloud service consumer is represented by the areas labeled 100 through 170. The identity provider relationships are shown in the areas labeled 160, 165, 265, and 260. The function of the identity provider service (IdP) at 160 and 260 together with the trust relationship at 165 and 265 is shown to help the reader make the transition to the more detailed diagram in Figure 13.



1099

1100 **Figure 12 – High-Level Architecture**

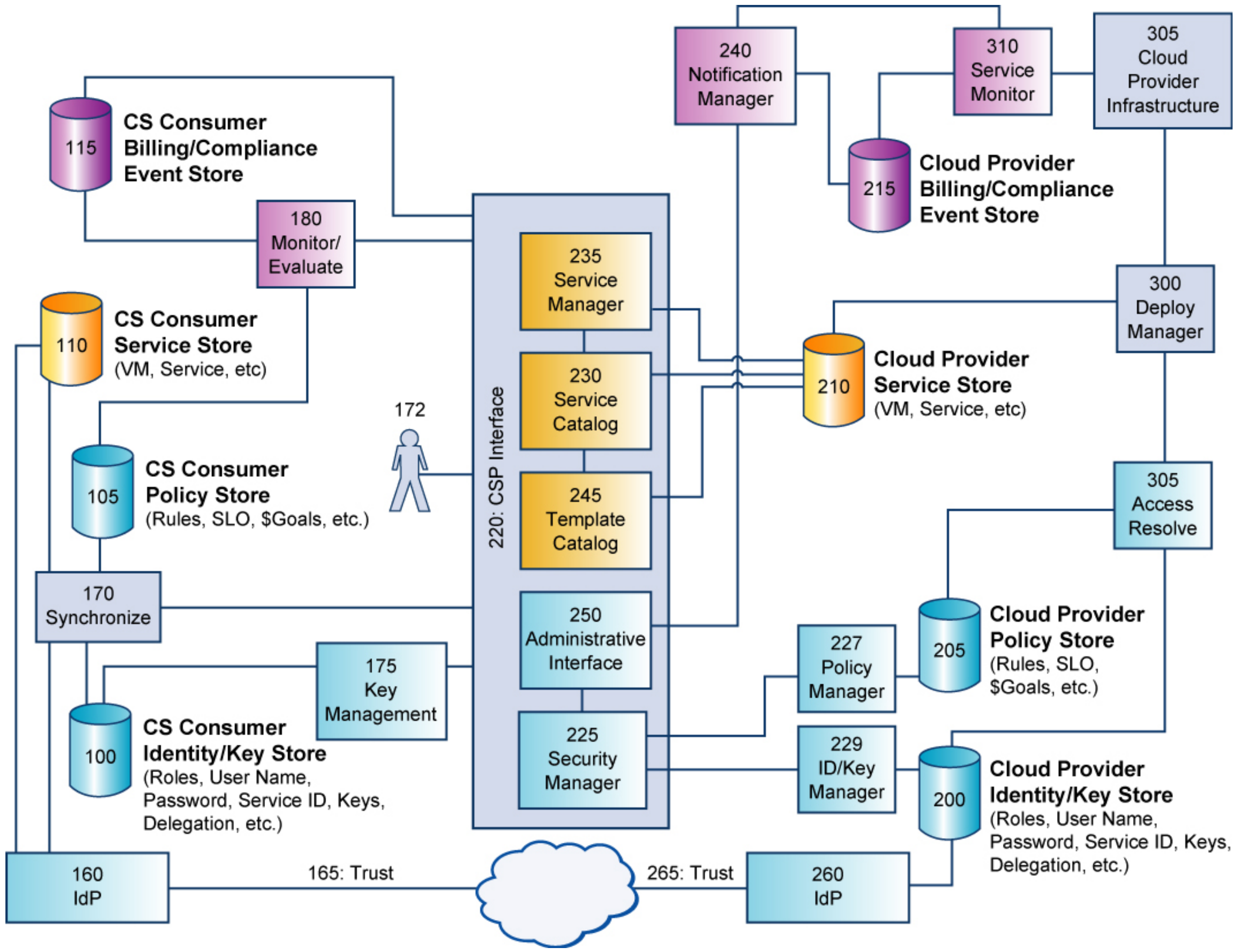


Figure 13 – Expanded Architecture

The discussion of Figure 13 begins by referencing the data artifacts that are managed by the interfaces 225, 230, 235, 245, and 250, along with their supporting data stores that are represented by 200, 205, 210, 215, 100, 105, 110, and 115. The interfaces are responsible for managing the lifecycle of the data artifacts described in the *Use Cases and Interactions for Managing Clouds* white paper (DSP-IS0103).

9.1 Architectural Impact on Data Artifacts

The Service Manager [235] manages the following data artifacts described in DSP-IS0103:

- service instance
- service topology
- service topology item
- service topology relationship

The Service Catalog [230] manages the following data artifacts described in DSP-IS0103:

- service offering
- service contract

- service request

The Template Catalog [245] manages the following data artifact described in DSP-IS0103:

- service template

The Security Manager [225] manages the following data artifacts described in DSP-IS0103:

- UserIdentityInformation
- CredentialsToken

The Administrative Interface [250] manages the following data artifacts described in DSP-IS0103:

- service reporting information
- service event log
- service notification
- All other data artifacts not mentioned above

9.2 Cloud Service Provider Identity/Key Store [200]

The cloud service provider identity/key store [200] provides the architectural abstraction for the cloud service provider to store and provide local identities, roles, cryptographic keys, delegation specifications, and so on. The data artifacts in the cloud service provider identity/key store [200] can be provided by the cloud service provider but are more commonly provided by a cloud consumer administrator [172], which manages the user names, passwords, roles, and so on for personnel associated with the consumer's business to use cloud resources. Using an administrator to provision the cloud service provider identity/key store [200] is labor intensive and has been found to be less than desirable by cloud consumers in today's market.

Another source of data artifacts in the cloud service provider identity/key store [200] is a synchronize function at 170 sponsored by the cloud consumer. This function synchronizes user names, passwords, roles, delegation specifications, and so on from an identity/key store on the consumer's premises [100]. This mechanism provides for much more rapid provisioning and deprovisioning of authorized cloud users and role and delegation information. However, unless the synchronize function at 170 can provide a password other than the password used by personnel to authenticate to the consumer's business and a single sign-on mechanism, the user name and password that allows access to the consumer's internal data systems is exposed. The alternative is to provide a different user name and password at the cloud service provider identity/key store [200], but this has its own issues with personnel remembering the user name and password and those credentials being exposed through "sticky notes" on the monitor.

For consumer businesses that have access to an identity provider [160], a trust relationship [165, 265] can be established with the cloud service provider's identity provider [260], and provisioning and deprovisioning of user names, passwords, and so on can be performed from attributes contained within the identity assertion produced by the identity provider [160]. This method still suffers from the possibility of exposing sensitive credentials at the cloud service provider identity/key store [200] or requiring multiple credentials for the user (although, the identity provider [160] can broker the use of multiple user names and passwords and therefore lift the burden from the user).

The most secure mechanism for providing consumer identity to the cloud service provider is through the identity provider [260], but rather than populating the cloud service provider identity/key store [200] with user names and passwords, the identity token from the identity provider [160] is used directly during access resolution at the cloud service provider infrastructure [305] without ever storing any user credential information in the cloud service provider identity/key store [200]. An alternative mechanism is to store the identity assertions in the cloud service provider identity/key store [200] and use those in lieu of user names and passwords as long as the identity assertion remains valid (for example, does not expire).

In the case of using the synchronize function [170] and cloud consumer administrator [172], the cloud service provider will need to provide the functions of an administrative interface [250], which acts as the interface or API for marshaling information through the security manager [225] to the ID/key manager [229], which is responsible for keeping the cloud service provider identity/key store [200] up to date. These functions (together with all other functions shown in Figure 13) are architectural only and represent functions to be performed rather than implementation mechanisms.

Another function affecting the cloud service provider identity/key store [200] is the maintenance of cryptographic keys on behalf of the consumer. These keys can be provided by either an administrator [172] or a key management mechanism [175]. In either case, the administrative interface [250] provides the functions necessary to marshal the key information to the cloud service provider identity/key store [200] through the security manager [225]. Storing cryptographic materials at the cloud service provider identity/key store [200] risks the possibility of cryptographic material exposure to the outside world. As with identity, key material can also be passed through identity tokens, which can provide cryptographic material without storing it locally at the cloud service provider's premises (for example, 160, 165, 260, 265).

To illustrate security policies in the cloud interface, Figure B-1 in Annex B describes security policies from the point of view of the organization. Table 1 provides a cross reference of items from Figure B-1 in Annex B to Figure 13:

Table 1 – Cross-Mapping of Elements in Figure B-1 with Elements in Figure 13

Annex B Policy Model (Figure B-1)	References to Figure 13
Items 120,121, 122, and 123	Items 110, 210, and 245
Item 124	Items 230 and 235
Item 126	Item 105
Item 127	Item 205
Item 125	Items 105 and 205
Item 255	Item 305
Item 310	Item 310
Item 315	Item 215
Items 320 and 325	Item 180

9.3 Cloud Service Provider Policy Store [205]

The cloud service provider policy store [205] provides a repository for policies that govern the use of the cloud service provider's infrastructure [305], service-level objectives (SLOs), monetary goals involving negotiated charges with the consumer ($Goal), firewall settings for consumer deployments, and so on. The cloud service provider policy store [205] is maintained by the policy manager [227], which, functionally, is associated with the security manager [225]. Commonly, the cloud service provider specifies its policies and that allows consumer administrators [172] to select the policies that they will use and provide some limited modification of those policies. Manual maintenance of consumer policies at the cloud service provider through the cloud service provider policy store [205] is labor intensive and is not scalable. This situation argues for a consumer policy store [105] wherein the consumer manages the policies, rules, and so on that are synchronized with the cloud service provider's policy store [205] (these policies are generally maintained by the cloud consumer as a part of their CMDB). Synchronizing policy from the consumer sites to the cloud service provider site is especially advantageous when the consumer is hosting a private cloud and wants to be able to use the same policies for private and public cloud deployments. Such a synchronization of policies requires arbitration between policy specifications at the cloud service provider's policy store [205] and policy specifications at the consumer policy store [105] provided by 170, 250, 225, and 227.

9.4 Cloud Service Provider Service Store [210]

The cloud service provider service store [210] provides the cloud storage for virtual machines and services. Services are defined in this paper as a collection of virtual machines that have been configured to provide a specific service. For example, an e-mail system may comprise an SMTP Gateway, IMAP4 server, POP3 server, storage, DNS server, and so on. The service may contain rules for load balancing and load shedding so that the service reacts appropriately to traffic decreases, increases, and imbalances.

Traditionally, the administrator [172] uses the service catalog [230] or template catalog [245] to discover what services and virtual machines are already configured in the cloud service provider service store [210]. Configuration of a virtual machine or service is performed through the service manager [235], which also provides facilities for the administrator [172] to define new virtual machines and services.

Today's use of private clouds makes it advantageous to allow the consumer's business to define its own services and virtual machines that could be used in both a private cloud and multiple public clouds. This functionality is depicted in the architecture as the CS Consumer Service Store [110]. The synchronization function shown at 170 provides the mechanisms for synchronizing the consumer service store [110] with provider service store [210] through functions and services provided by the service manager [235]. Synchronization can both add and remove virtual machines and services, and as this happens the service catalog [230] is updated. The service catalog [230] and the service manager [235] are both protected by identity so that virtual machines and services are accessible only by the consumer audiences for which they are intended (whether public or restricted to a membership list). The architecture also depicts a relationship between the consumer service store [110] and the identity provider [160]. As identity and policy-applied-by-identity become increasingly important to regulatory compliance and other audit concerns, it will be important for virtual machines and services to be protected by both digital signatures and digital identities. These digital identities, and possibly the cryptographic materials, will be provided by the identity provider [160]. Because of the trust relationship [165, 265], virtual machines and services thus tagged with identity are recognizable by the cloud service provider's infrastructure (specifically by the deploy manager [300]) so that policy and access can be regulated by the identity and tampering can be recognized.

9.5 Cloud Service Provider Billing/Compliance Event Store [215]

A critical facility in the architecture supporting cloud service provider functionality is the cloud service provider Billing/Compliance Event Store [215]. As virtual machines and services operate within the cloud service provider infrastructure [305], it is critical that monitoring take place at the service monitor [310] to populate the cloud service provider billing/compliance event store [215] with the appropriate billing events and regulatory compliance events. These events are used by the cloud service provider to provide billing materials at the end of the month, provide audit materials for the customer to verify the charges from the cloud service provider, and provide an audit trail concerning regulatory compliance.

Consuming businesses are asking for ways to track their use of cloud resources and to watch for audit events from cloud processes and resources in real-time. This is shown in the architecture at CS Consumer Billing/Compliance Event Store [115]. This event store is provided either through a pull or push model with the billing and compliance events monitored by the cloud service provider through the notification manager [240], and is managed through the administrative interface [250]. The notification manager [240] can also provide real-time event flow to the monitor/evaluate function [180], which can allow the consumer's business to monitor cloud evaluation and capture possible regulatory compliance deviations in real-time. The monitor/evaluate function [180] can also provide for forensic evaluations from the data contained within the event store [115]. In all of the evaluations by the monitor/evaluate function [180], the determination of whether events may be causing compliance deviation or exceeding cost goals is determined by the policy stored in the consumer policy store [105]. Likewise, a service-level objective

(SLO) can be used to verify that the service-level agreement (SLA) to which the consumer and the cloud service provider agreed is being honored.

9.6 Cloud Service Provider Deployment [300]

Finally, the architecture depicts deploying virtual machines and services into the cloud service provider infrastructure [305]. This is performed by first resolving access issues [305] and then matching the access request with the services by the deploy manager [300]. It is the responsibility of the deploy manager [300] to ensure that services and virtual machine utilization are managed according to policy and identity. The policy includes both the internal policy by the cloud service provider and policies provided by the consumer. Services are deployed into the cloud service provider infrastructure [305] and finally deprovisioned from the cloud service provider infrastructure according to either a manual request by the administrator [172] or some procedural or policy specification through the administrative interface [250].

10 Conclusion and Next Steps

The current phase of the Open Cloud Standards Incubator is concluded. The work of the incubator has resulted in three documents outlining the use cases, data artifacts, and architectural considerations for IaaS clouds. This output will be handed over to the appropriate DMTF working groups and alliance partners so that they can develop standards while the Open Cloud Standards Incubator considers other potential areas to tackle.

10.1 Standards Development Steps

To take the output of the incubator and produce DMTF standard specifications, one or more working groups are needed to model the domain and design the details of the infrastructure and the interfaces. Deliberations in the incubator resulted in the request to create two working groups:

- the Cloud Management Model Working Group (CMMWG) under the Platform Management Subcommittee)
- the Cloud Management Interface Working Group (CMIWG) under the Infrastructure Subcommittee)

After the DMTF Board approves the appropriate working groups, their charters will be posted at http://www.dmtf.org/about/committees/.

10.2 Important Extensions to Be Considered

This section describes further work that needs to be done to improve the flexibility, interoperability, and ease of acquisition of a DMTF cloud. The work that is remaining is itemized in this section, and it is expected that the two working groups will address these work items in some way as part of their deliberations.

10.2.1 Interface Extensibility

The incubator's interface is "narrower" than available cloud interfaces. For example, what will the consumer do if it needs both? Use them side by side? Have a DMTF interface that is extendable so more detailed interfaces can be exposed by an implementation that is under the umbrella of the DMTF interface?

10.2.2 Template Extensibility

It may be necessary to provide the means of adding new types of entities to catalogs, and composing existing entities in a catalog into another entity. For example, if a consumer wants to add a firewall in front of an existing service, how is it represented? The interface needs to have features such as copying an existing template and modifying parts of it, or downloading a template into a metadata file (for example, OVF) on the client end, modifying it, and uploading it back to the cloud.

10.2.3 Management Interface Granularity

The cloud interfaces provided by the implementations in the industry broadly fall into three groups:

- an administrative group responsible for aspects like the overall relationship, service contract, and catalog management
- a workload management group that deals with workloads, involving service instances and their lifecycle
- a resource management group that is responsible for management of individual artifacts such as a VM or logical server, NICs, and storage volumes

The incubator interface represents the administrative group and, to some extent, the workload management group. It does not have any artifacts to represent the resource management group.

10.2.4 Security Model Granularity

The security model may need further clarification such as mapping the main security categories (identity and access management, data protection, security event management, and software/platform/ infrastructure [compliance] assurance) to the cloud reference architecture. Furthermore, it may be necessary to define specific interfaces for cloud users to have the option of more granular visibility of and control over how their security policies and requirements are met by the service providers.

10.3 Integration with Alliance Partner Frameworks

The Open Cloud Standards Incubator looks to alliance partners to provide domain-specific implementations and expects the DMTF cloud implementations to interoperate with them. An example is a storage cloud standard being worked on by SNIA. Activities need to be undertaken to provide guidance on connecting the DMTF cloud interfaces and those created by alliance partners such as SNIA. Work may be needed both from a resource-model-compatibility perspective as well as from a protocol perspective.

11 Future of the Cloud Incubator

With regard to the Open Cloud Standards Incubator, two areas of further work have been identified: cloud federation and Platform as a Service (PaaS). It is recommended that the incubator charter be extended to do further work in these areas.

Annex A
(informative)

Message Exchange Protocol Examples

This annex provides examples of how various Message Exchange Patterns (MEPs) introduced in 9.1 can be rendered using XML-RPC and REST-oriented mechanisms.

A.1 One-Way/Push MEP: RPC Example

A.1.1 Client Request: Set User Age

```
POST /MyWebService.Contos.com HTTP/1.1
   Content-Type: text/xml
      Content-length:
      <?xml version="1.0"?>
         <methodCall>
            <methodName>Users.SetUserAge</methodName>
            <params>
               <param>
                  <name><string>Brian Young</string></name>
               </param>
               <param>
                  <value><i4>23</i4></value>
               </param>
            </params>
         </methodCall>
```

A.1.2 Service Response

```
   HTTP/1.1 200 OK
   Host: contoso.com
   Content-Type: text/xml
   Content-length:0
```

A.2 One-Way/Push MEP: REST Example

A.2.1 Client Request: Set User Age

```
   PUT /uri_of_brain_young
   Host: contoso.com
   Content-Type : application/vnd.cloud.com.User+xml
   Content-Length: xxxx
   <xml>
      <user>
         <age value="23"/>
      </user>
   </xml>
```

A.2.2 Service Response

```
HTTP/1.1 200 OK
Host: contoso.com
Content-Type: text/xml
Content-length:0
```

A.3 One-Way/Pull MEP: RPC Example

The one-way/pull MEP is different from a request-response (two-way) synchronous MEP in the sense that the “pulled” message is not the result of a query with a specific selector that may vary for each MEP execution; it is more akin to reading a queue of messages. Typically, a pull message only knows about a channel or queue ID (the equivalent of a mailbox ID). A variant may rely on a notified “event ID” that the client must use later only once.

In order to pull, the client must have prior knowledge of this event ID or channel ID. The event ID or channel ID could be provided dynamically by the service calling a particular client method. In these examples, it is assumed that the client can provide a client-side URL prior to or as a part of the call that is issued.

A.3.1 Service Initiated Notification

```
<!--- In this variant, an “event ID” is initially notified to the client by the
service. The event ID is to be used for subsequent pulling. ---!>

   POST /MyAddress.Client.com HTTP/1.1
   Content-Type: text/xml
   Content-length:

   <?xml version=”1.0”>
      <methodCall>
         <methodName>notifyEvent</methodName>
         <params>
            <param><eventID><i4>1234</i4></eventID></param>
            <param>
               <eventDescription><string>”disk low”</string></eventDescription>
            </param>
         </params>
      </methodCall>
```

A.3.2 Client Response

```
HTTP/1.1 200 OK
Host: client.com
Content-Type: text/xml
Content-length:0
```

A.3.3 Client Request: Pull Event

```
<!--- Pull the Event Request using a one-way/pull method. ---!>
   POST /MyWebService.Contoso.com HTTP/1.1
   Content-Type: text/xml
   Content-length:
```

```

   <?xml version=”1.0”?>
   <methodCall><methodName>eventGetInfo</methodName></methodCall>
      <params>
         <param>
            <eventID><i4>123456</i4></eventID>
         </param>
      </params>
   </methodCall>
<!--- Pull the Event Response ---!>
```

A.3.4 Service Response

```
   HTTP/1.1 200 OK
   Host: contoso.com
   Content-Type: text/xml
   Content-length:

   <?xml version=”1.0”?>
   <methodResponse>
         <params>
            <param>
               <maxDisk><kilobytes>512000000</kilobytes></maxDisk>
            </param>
            <param>
               <usedDisk><kilobytes>501000000</kilobytes></usedDisk>
            </param>
         </params>
   </methodResponse>
```

A.4 One-Way Pull MEP: REST Example

A.4.1 Service Initiated Notification

```
   POST /MyAddress.Client.com/Events
   Content-Type : application/vnd.cloud.com.Event+xml
      <event>
         <eventURI>/Events/1234</eventURI>
         <□ventide>1234</□ventide>
         <eventMessage>disk low</eventMessage>
      </event>
```

A.4.2 Client Response

```
   HTTP/1.1 200 OK
   Host:Client.com
   Content-length:0
```

A.4.3 Client Request

```
   GET /MyWebService.Contoso.com/Events/1234
   Accept :application/vnd.cloud.com/Event+xml
```

A.4.4 Service Response

```
HTTP/1.1 200 OK
Host: contoso.com
Content-Type : application/vnd.cloud.com/Event+xml
Content-Length:xxx

<?xml version="1.0"?>
   <event>
      <□ventide>1234</□ventide>
      <eventMessage>disk low</eventMessage>
      <eventDetail>
         For Volumn abc: 501000000 out of 512000000 are used
      </eventDetail>
   </event>
```

A.5 Two-Way/Sync MEP: RPC Example

A.5.1 Client Request: Enumerate Users with Filter

```
POST /MyWebService.Contoso.com HTTP/1.1
Content-Type: text/xml
Content-length:
<?xml version="1.0"?>
   <methodCall>
      <methodName>Users.EnumerateUsers</methodName>
         <params>
            <param>
               <filter><string>*B*</string>
            </param>
         </params>
   </methodCall>
```

A.5.2 Service Response

```
HTTP/1.1 200 OK
Host: contoso.com
Content-Type: text/xml
Content-length:

<?xml version="1.0"?>
   <methodResponse>
      <params>
         <param>
            <name><string>Betty White</string></name>
         </param>
         <param>
            <name><string>Brian Young</string></name>
         </param>
      </params>
   </methodCall>
```

A.6 Two-Way/Sync MEP: REST Example

A.6.1 Client Request: Enumerate Users with Filter

```
GET /Users?name=[*B*]
Host:MyWebService.Contoso.com
Accept:application/vnd.cloud.com/Users+xml
```

A.6.2 Service Response

```
HTTP/1.1 200 OK
Host: contoso.com
Content-Type : applicationapplication/vnd.cloud.com/Users+xml
Content-Length:

<?xml version=”1.0”?>
   <users>
      <user>
         <name>Betty White</name>
      </user>
      <user>
         <name>Brian Young</name>
      </user>
   </users>
```

A.7 Two-Way/Push-and-Push MEP: RPC Example

```
<!----- Issue API call to format volume ----!>
```

A.7.1 Client Request: Format Volume

```
POST /MyWebService.Contoso.com HTTP/1.1
<?xml version=”1.0”>
   <methodCall>
      <methodName>formatVolume</methodName>
      <params>
         <param>
            <volumeID><string>”some volume”</string></volumeID>
         </param>
         <param>
            <callbackURL><string>”http://MyAddress.Client.com”</callbackURL>
         <param>
      </params>
   </methodCall>
```

A.7.2 Service Initial Response

```
HTTP/1.1 200 OK
Host: contoso.com
Content-Type:text/xml
Content-Length:

```

```
<?xml version=”1.0”?>
   <params>
      <param>
         <jobID><i4>1234</i4></jobID>
      </param>
      <param>
         <eventDescription>
            <string>”volume format Initiated”</string>
         </eventDescription>
      </param>
   </params>
```

A.7.3 Service Initiated Progress Notification

```
POST /MyAddress.Client.com
Content-Type: text/xml
Content-Length:

<?xml version=”1.0”?>
   <methodResponse>
      <params>
         <param><jobID><i4>1234</i4></jobID></param>
         <param><maxDisk><kilobytes>512000000</kilobytes></maxDisk></param>
         <param><usedDisk><kilobytes>0</kilobytes></usedDisk></param>
      </params>
   </methodResponse>
```

A.7.4 Client Response to the Service

```
HTTP/1.1 200
Host: contoso.com
Content-Length:0
```

A.8 Two-Way/Push-and-Push MEP: REST Example

A.8.1 Client Request: Format Volume

The client has already registered the callback URI before sending the request.

```
PUT /Volumes/MyVolumeuri
Host: MyWebService.Contoso.com
Content-Type: application/vnd.cloud.com.Volume+xml
Content-Length: xxxx

<xml>
   <volume>
      <format>desired_format</format>
   </volume>
</xml>
```

A.8.2 Service Initial Response

```
HTTP/1.1 200 OK
Host:Contoso.com
Content-type: text/xml
Content-length:0
```

A.8.3 Service Initiated Progress Notification

```
POST /MyAddress.client.com/
Content-type : applicationapplication/vnd.cloud.com.Event+xml
Content-length :
< ?xml version=”1.0” ?>
<xml>
   <event>
      <resource
       uri=”/Volumes/MyVolumeuri”
       type=”application/vnd.cloud.com.Volume+xml”
       status=”READY”/>
      <detail>Reformatting the volume from aaa to bbb</detail>
   </event>
</xml>
```

A.8.4 Client Response

```
HTTP/1.1 200 OK
Host: Client.com
Content-length:0
```

The client can then use the resource URI returned in the event to GET the details of the volume.

A.9 Two-Way/Push-and-Pull MEP: RPC Example

A.9.1 Client Request: Format Volume

```
<!----- Issue API call to format volume ----!>

POST /MyWebService.Contoso.com HTTP/1.1
Content-Type: text/xml
Content-length:

<?xml version=”1.0”>
   <methodCall>
      <methodName>formatVolume</methodName>
      <params>
         <param><volumeID><string>”MyVolumeID”</string></volumeID></param>
      </params>
   </methodCall>
```

A.9.2 Service Response

```
HTTP/1.1 200 OK
Host:Contoso.com
Content-Type: text/xml
Content-Length:
<!--- receive jobID to poll --!>

<?xml version=”1.0”?>
   <methodResponse>
      <params>
         <param><jobID><i4>1234</i4></jobID></param>
      </params>
   </methodResponse>
```

A.9.3 Client Request: Check Job Status

```
POST /MyWebService.Contoso.com HTTP/1.1
Content-Type: text/xml
Content-length:
<!—PULL with jobID as the parameter ---!>

<?xml version=”1.0”?>
   <methodCall>
      <methodName>pollJob</methodName>
      <params>
         <param><jobID><i4>1234</i4></jobID></param>
      </params>
   </methodCall>
```

A.9.4 Service Response

```
<!— get back progress information ---!>
HTTP/1.1 200 OK
Host: contoso.com
Content-Type: text/xml
Content-Length:

<!xml version=”1.0”?>
<methodResponse>
   <params>
      <param><jobID><i4>1234</i4></jobID></param>
      <params<percentComplete><i4>74</i4></param>
   </params>
</methodResponse>
```

A.10 Two-Way/Push-and-Pull MEP: REST Example

A.10.1 Client Request: Format Volume

```
PUT /Volumes/MyVolumeuri
Host: MyWebService.Contoso.com
Content-Type : application/vnd.cloud.com.Volume+xml
Accept : application/vnd.cloud.com.Volumn+xml, application/vnd.cloud.com.Event+xml
Content-Length: xxxx

<xml>
   <volume>
      <format>desired_format</format>
   </volume>
</xml>
```

A.10.2 Service Response

```
HTTP/1.1 200 OK
Content-type : application/vnd.cloud.com.Event+xml
Location: /Events/ThisEventUri
Content-Length:xxxx

<?xml version=”1.0”?>
   <xml>
      <event uri=”/Events/ThisEventUri”>
         <resource
          uri=”/Volumes/MyVolumeuri”
          type=”application/vnd.cloud.com.Volume+xml”
          status=”REFORMATTING” percentage=”0”/>
         <detail>reformatting volume from aaa to bbb</detail>
      </event>
   </xml>
```

A.10.3 Client Request

```
<!---------- Poll for the status of the job ---!>

   GET /Events/ThisEventUri
   Host: MyWebService.Contoso.com
   Accept : application/vnd.cloud.com.Event+xml,
```

A.10.4 Service Response

```
http/1.1 200 OK
Content- type: application/vnd.cloud.com.Event+xml
Location: /Events/ThisEventUri
Content-Length:xxxx
<?xml version=”1.0”?>
   <xml>
      <event uri=”/Events/ThisEventUri”>
```

```
            <resource
             uri=”/Volumes/MyVolumeuri”
             type=”application/vnd.cloud.com.Volume+xml”
             status=”REFORMATTING” percentage=”74”/>
            <detail>reformatting volume from aaa to bbb</detail>
        </event>
    </xml>
```

Annex B
(informative)

Policy Discussion

This annex describes what a policy model for clouds might look like. It is not intended to be prescriptive, but rather exemplary.

B.1 Policy Model

The policy model could be represented by Figure B-1.

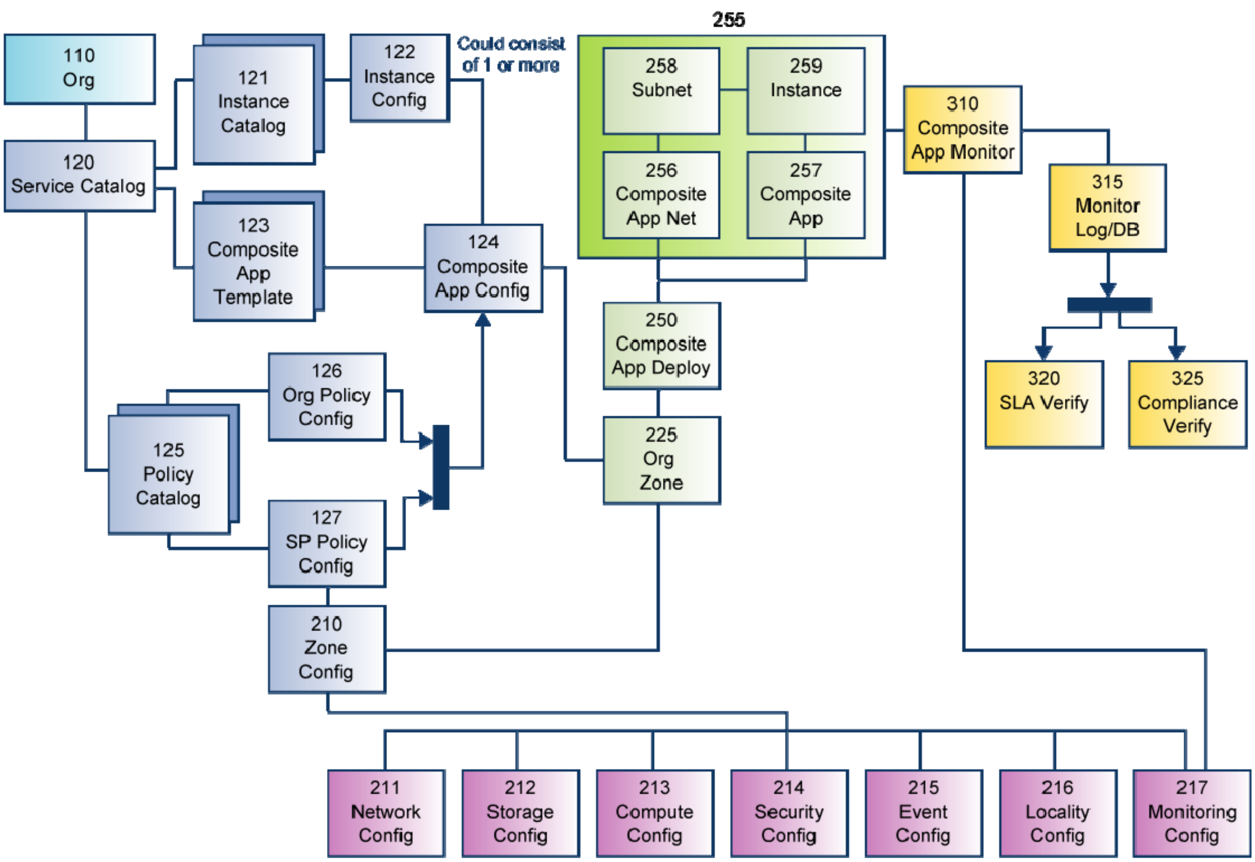


Figure B-1 – Policy Model

B.1.1 Policy Model from the Organization's Perspective

Consider an organization [110] that wants to use the cloud at a service provider. The organization [110] is constrained by its contract with the service provider to only the services in the service provider's catalog that the organization approves (a part of the contract between the organization and the service provider). The service provider's catalog consists of an instance catalog [121], a composite application template

[123], and a policy catalog [125]. The instance catalog [121] consists of a collection of the individual instances [122], and creates a VM repository.

The composite application template [123] consists of the collection of instances and templates defined by the organization for use. These templates could be uploaded by an API, created out of individual instance groupings, deployed in the service provider's cloud, or uploaded manually.

The policy catalog [125] consists of the organization's policy configurations [126] and the service provider's policy configurations [127]. The service provider's policy configurations [127] consist of the service provider policies that govern the use and deployment of the infrastructure of the IaaS (for example, network [211], storage [212], compute [213], security [214], event [215], locality [216], and monitoring [217] configurations).

The organization now constructs a repository of services at the service catalog [120], which comprises instance specifications (for example, pointers to instances [122]) and configuration and operational stipulations (for example, SLO specifications). The organization previously specified policy [126] that defines the organization's governance expectations concerning services when they become operational. The collection of 120, 121, 123, and 126 can be considered part of the organization's CMDB.

B.1.2 Policy Model from the Service Provider's Perspective

The service provider specifies the operational characteristics and configurations of each of the zones that compose the overall IaaS cloud. This is represented by zone configuration [210], which consists of a collection of zone configurations comprising network configuration [211], storage configuration [212], compute configuration [213], security perimeter configuration [214], event configuration [215], locality configuration [216], and monitoring configuration [217]. Each of these provides detailed hardware and interconnect configurations necessary to specify the infrastructure policy enforcement configurations.

The service provider's policy configuration [127] will provide the governance for the collection of zone configurations [210] (because there are multiple zones, there will be multiple policies governing the zones and zone interactions). The collection at 210 and 127 can be considered part of the service provider's CMDB.

The composite application configuration process [124] uses information at 123, 126, 127, and 210 to provide a coherent definition of a composite application's zone configuration [225]. The composite application configuration process [124] must provide a normalization of the policies of 126 and 127 so that the expectations of both the organization [110] and the service provider are enforced. If these policy expectations cannot be realized, the specific composite application configuration process [124] will not be created, and an exception will be asserted to both the organization [110] and the service provider.

The resulting composite application's zone configuration [225] can be deployed [250] into one or more service provider's infrastructures [255]. These infrastructures include 256, 257, 258, and 259. Note that "media" is implied but not specified in Figure B-1.

The resulting deployment is monitored [310], providing a log [315]. Processes to verify the SLA through SLOs [320] and operational compliance to regulatory agencies [325] are provided. SLA verification [320] and compliance verification [325] may be operational in the organization's data center, in the cloud service provider, or a combination of both.

An example of the production of a coherent definition of a deployable composite application configuration could be as follows:

1) The service provider has noted in the policy configuration [127] that all 10 GB network interconnects are available only to platinum customers.

2) The organization under consideration is a customer that has purchased a standard performance package from the service provider. The standard performance package specifies that only 100 MB and 1 GB network interconnects are available to the organization.

3) When the composite application configuration is generated, all 10 GB network interconnects are removed from any of the zone configurations that are available to the organization. In this way, although the organization is participating in a zone that has 10 GB network conductivity, that conductivity is not available to the organization because any 10 GB network interface cards are not available to instances being deployed by the organization.

4) If the organization upgrades its customer status to that of a platinum customer, the 10 GB network interconnects are not culled from the zone configuration, which allows any composite applications to use the higher bandwidth without changes to the application templates, instances, and so on.

5) Likewise, if the contract period for being a platinum customer expires and the customer reverts to standard status, the enhanced network bandwidth is not available.

6) The organization may have specified in its organization policy [126] that certain services must be connected to 10 GB networks. If the customer type is that of "standard," a conflict exists between the policy of the service provider and the policy of the organization. This conflict is noted to both the service provider and the organization for resolution.

7) Because policy between the organization and the service provider is being normalized during composite application configuration [124] and verified during composite application deployment [250], capabilities can be added and subtracted from any given deployment because of policy resolution based on configuration (for example, customer type).

B.1.3 Policy Examples

Table B-1 – lists examples of policy types grouped by area.

Table B-1 – Policy Examples

Policy Area	Examples
User and identity	Identity: Federation Authentication Authorization
Operational attributes	Access control User profiles Key/Certificate management Patching Perimeter Virtualization security Monitoring, incident response, and notification Secure workload migration
Legal and compliance	Electronic discovery Data jurisdiction Data retention and destruction Data location Privacy
Network and connectivity	Perimeter firewall template (to enable organizations to specify the network perimeter firewall rules): • Action: Allow/Deny • Protocol TCP/UDP/IP • Source IP • Destination IP • Port • Log

Policy Area	Examples
	Layer 2 ACLs template (to enable organizations to specify the network layer ACL rules): • Action: Allow/Deny • Protocol TCP/UDP/IP • Source IP • Destination IP • Port
	IDS/IPS template(to enable organizations to specify network IDS/IPS signatures): • Action detect/prevent • Signatures • Learning period
	Network isolation (to provide physical and logical separation of network resources)
	Transport security: TLS, VPN
Storage	The storage rules, constraints, and policies apply to all forms of storage, including block, file, queue, database, and key store. Some examples are: • Zones (to provide logical separation of organization storage) • Data classification (to provide organizations with data classifications of *public* or *confidential*) • Access/encryption (to allow organizations to request access restrictions or encryption of the data at rest) • Isolation • Location • Retention (to allow organizations to define retention intervals) • Backup and recovery (to allow organizations to specify policy for data backup and recovery)
Compute resources	Affinity Composite application constraints Antivirus Patch management Priority Availability (for example, against DoS attacks) Location and placement Isolation for multi-tenancy Platform (HW/SW) integrity and assurance Scope of policies (where they are applicable)
Security	DPI firewall rules (Relation to Network) allow customers to translate their tier-to-tier firewall rules to the service provider server tier firewall. • Action: Allow/Deny • Protocol: TCP, UDP, ICMP, IP • Source IP • Destination IP • Port • Log • Threat management (Relation to Network) • Log management (Relation to Audit) • Legal/regulatory controls • Isolation (Related to Affinity) • Audit

Policy Area	Examples
Alarms	Thresholds Ticketing
Monitoring	Events Alerts Thresholds
Location	Specific location (related to legal/regulatory) Availability

Annex C
(informative)

Change Log

Version	Date	Description
1.0.0	2010-06-18	Released as DMTF Informational