SLA Agreement, Negotiation, Execution and Monitoring using OCCI

Thijs Metsch (Platform), Victor Bayon (Intel), Andy Edmonds (Intel), Alexander Papaspyrou (TUDO).

Introduction

This document describes how the Open Cloud Computing Interface (OCCI) can be extended to support mechanisms for Monitoring and SLA agreement negotiation. It is a first step towards creating extensions to the current version of the OCCI specification (Version 1.1) which enable OCCI-based Services to offer these features. This work is influenced by WS-Agreement & WS-Negotiation as well as several monitoring frameworks.

This specification contains both details on SLAs and monitoring since these two aspects are closely related. It should be noted that—although related—the two specifications can exist independently. Ideally, where a system implements the OCCI notion of an SLA, it should also realise the exposure of a monitoring API with this specification's contained description of monitoring.

After describing the terminology, this document will introduce the negotiation of agreements, followed by the description of a Monitoring extension for OCCI.

Terminology

Throughout the document, we will use the following terms:

- **Service Level Agreement** (SLA) - An agreement which defines a dynamically-established and dynamically managed automated contract/agreement between between a service provider and a customer. The representation of this SLA is machine readable.
- **Service Level Objective** (SLO) - a technical clause that specify a non-functional guarantee in legally bounding document (SLA), specifying the terms and conditions of service provisioning.
- **Agreement** - A set of SLOs (reflected in a resource instances) on which the client and service provider 'agree' (Part of an SLA).
- **Negotiation** - The process of creating an agreement. This process can be part- or fully-automated.
- **Template** - An agreement template is a representation used by the agreement responder (normally the service provider) to advertise the types of service requests or

offers a service provider is (possibly) willing to accept/provide. A template can be thought of the serivce providers *invitation to offers*.
- **Offer** - what the client requests of the provider through an instance of a template.

Negotiation and Provisioning of Agreements

The following sections deal with the negotiation of agreements and therefore also the creation of the SLAs. The process of reaching an agreement - signing an SLA - and managing resources under this agreement can be described by following this workflow:

1. **Negotiation** phase - The customer retrieves the Templates. After populating a suitable template with required values the customer can Negotiate with the Service Provider.
2. **Agreement** phase - The service provider can decide whether to accept the filled out template (the offer) or not. It is also possible to provide a counter-offer to the customer.
3. **Execution** phase - When the agreement has been accepted the Agreement is in place and (newly) created resource can be marked as falling under and associated with the reached agreement.
4. **Monitoring & Notification** phase - Finally once the resources are provisioned they can be monitored. Upon violation of the SLAs a Notification mechanism should be used to notify the customer.

Proposal to support SLA Negotiation & Provisioning in OCCI

OCCI provides means for defining types through Categories which define Kinds, Actions and Mixins. The following Kind and Mixin definitions are considered useful for this Agreement Negotiation proposal.

Agreement Kind Definition

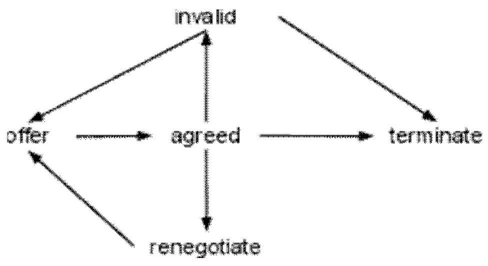
This later will represent the kind ‘Agreement’ through a resource instance.

Term	agreement
Scheme	http://schemas.ogf.org/occi/sla#
Title	An agreement
Related	http://schemas.ogf.org/occi//core#resource
Actions	agree, terminate
Attributes	

Attribute	**Type**	**Multiplicity**	**Mutability**	**Description**
state	enum	1	Immutable	The state the agreement is

				in
hash	string	1	Immutable	a hash to verification reasons filled after agree

The following states for the **agreement phase** are defined for the agreement instance:



- *offered* - not agreed upon, agreed upon by the ‘server side’
- *agreed* - agreement in agree state
- *invalid* - agreement became invalid
- *terminated* - agreement got terminated
- *renegotiate* - agreement needs to renegotiate (initialized by the ‘server side’)

The `end`, `updated` and `create` date can be added in as an `audit`-Mixin (See below).

AgreementLink Kind Definition

This link will associate resource instances with the agreement. Here the Service provider needs to ensure that certain criteria are met by the resource (e.g. Can it fall under the agreement, does it have the right monitoring mix-ins applied etc.)

Term	agreement_link
Scheme	http://schemas.ogf.org/occi/sla#
Title	An link between an agreement and resource instances
Related	http://schemas.ogf.org/occi/core#link
Actions	N/A
Attributes	

Attribute	**Type**	**Multiplicity**	**Mutability**	**Description**

The, `updated` and `create` date can be added in as an audit-Mixin (See below).

Templates Mixin Definitions

OCCI's Template mixin can be used to define Templates as they are used in this document. The following table shows an template.

Term	template
Scheme	http://schemas.ogf.org/occi/sla#
Title	OCCI SLA Template Mixin
Related	None
Actions	None
Attributes	None

A template can be associated with other types of Mixins, for example an OS template. A simple example of a provider-specific SLA template is shown below:

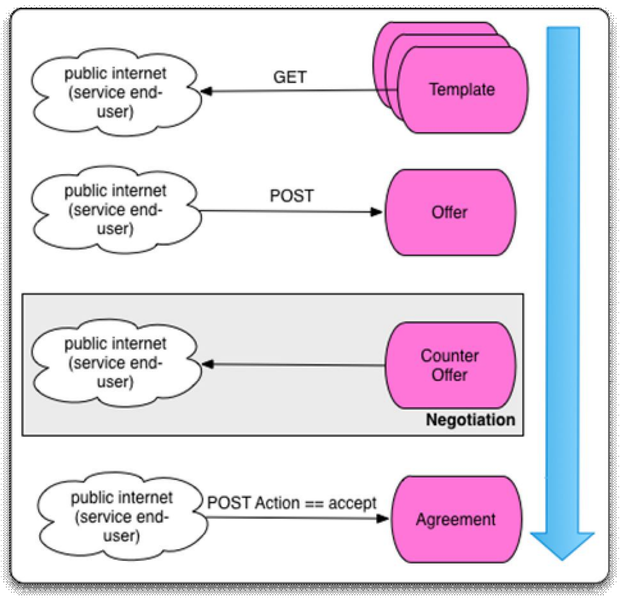
Term	gold
Scheme	http://www.provider.com/infrastucture/templates#
Title	An example template
Related	http://schemas.ogf.org/occi/sla#agreement_template
Actions	N/A
Attributes[1]	

Attribute	**Type**	**Multiplicity**	**Mutability**	**Description**
occi.compute.cores	Integer	1	Immutable	Number of CPUs - ***default 8***
occi.compute.memory	float	1	Immutable	8.0

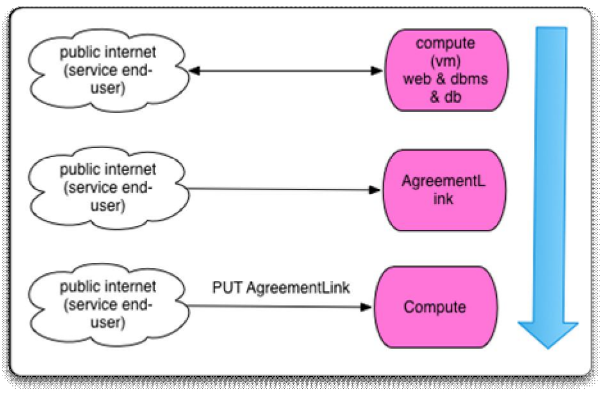
[1] These attributes are defined and set by the provider.

SLA Negotiation & Provisioning in OCCI

Based on those 3 Categories the following scenario can be realized:


public internet (service end-user)
GET
Template
public internet (service end-user)
POST
Offer
public internet (service end-user)
Counter Offer
Negotiation
public internet (service end-user)
POST Action == accept
Agreement

1. The customer (from now on referred to as User) requests a if the Agreement kind, available Templates (for SLOs and Metrics) and the Agreement link are available in the service provider's QI. The user can use all the query interface capabilities such as filtering to accomplish this job.
    a. HTTP GET /.well-known/org/ogf/occi/
2. The User tries creates an instance of the kind Agreement with the help of the provided templates. This is known as the offer. If the service provider agrees with the offer it'll return a 200/201 OK and create the Agreement. If the service provider chooses to reject Bad Request is returned. If the service provider wants to make a counter-offer 302 is used (This is part of the negotiation phase). A Location with a temporary agreement should be created
    a. HTTP POST /agreement/ or HTTP PUT /agreement/123 (with Category agreement;scheme... and templates as Mixin definitions if applicable)
3. When the user finally wants to agree upon the agreement he needs to trigger the `agree` Action. The state of the agreement will change according the life cycle. Now resources can be put under this agreement.
    a. HTTP POST /agreement/123?action=agree


public internet (service end-user)
compute (vm) web & dbms & db
public internet (service end-user)
AgreementLink
public internet (service end-user)
PUT AgreementLink
Compute

4. Existing or newly created resources can now be bound to the agreement using the AgreeementLink.
    a. HTTP POST /agreement/ (with Category of the resource and a link description)
    b. HTTP POST /agreement/link/ (with Kind of the AgreementLink and source and target attributes)
    c. HTTP POST /agreement/123 (with the Link definition)

An Audit-Mixin

This mixin allows for date stamping of certain operations related to SLA establishment, updates and termination.

Term	date_audit
Scheme	http://schemas.ogf.org/occi/audit#
Title	Adds the date-time stamping to an associated instance
Related	None
Actions	None
Attributes	

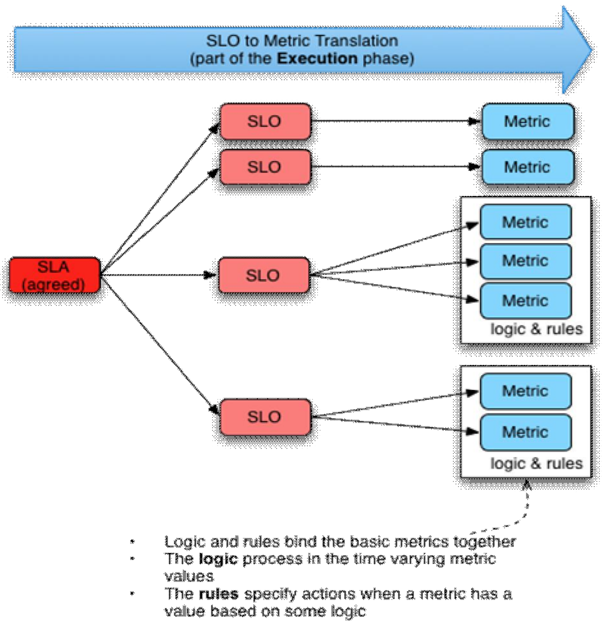
Attribute	**Type**	**Multiplicity**	**Mutability**	**Description**
end	string	0..1	Mutable	ISO8601
created	string	1	Immutable	ISO8601
updated	string	0..1	Immutable	ISO8601

Instance Monitoring

Once a service consumer has running instances with a provider, the next thing that this consumer should be able to do is monitoring those instances to proactively manage potential problems. As such—from a point of view of manageability—monitoring is essential. The approach taken in this monitoring extension is to keep things as simple as is possible and stay compatible with the Core model in that the defined Mixin can be validly applied to all instances of Entity.

Note on Monitoring and SLAs

Before continuing into the specification of monitoring, it is valuable to discuss how monitoring and SLAs are related. In order for a provider to offer SLAs (a set of SLOs), he needs to be monitoring his service instance offerings with respect to certain basic (e.g. network receive per second) and/or calculated metrics (e.g. uptime). Such metrics, expressed as SLOs, then form the basis of service offerings that the provider is willing to guarantee. Note that, these do not necessarily resemble the service provider's internal representation of those SLOs. When a service consumer accepts an SLA for which the provider asserts he can satisfy the conditions, and the consumer creates service instances based on that Agreement, the service provider MUST setup the required monitoring on those instances so that the related metrics be collected and verified (through logic and rules) by internal SLA management systems of the provider. Those metrics SHOULD be exposed to the consumer, and—as a basic minimum the SLOs within the agreed SLA—SHOULD be offered for inspection by the consumer. The consumer COULD access these SLOs as metrics using the OCCI monitoring specification.


SLO to Metric Translation
(part of the **Execution** phase)
SLA (agreed)
SLO
SLO
SLO
SLO
Metric
Metric
Metric
Metric
Metric
logic & rules
Metric
Metric
logic & rules
• Logic and rules bind the basic metrics together
• The **logic** process in the time varying metric values
• The **rules** specify actions when a metric has a value based on some logic

Representing Metric

A metric is represented by an instance of OCCI Mixin. A metric has a set of associated attributes. It should be kept in mind that a Metric is in effect an Attribute of the resource being monitored and so their formulation is very much the same as what is seen with X-OCCI-Attribute.

Metric Mixin Definition

Term	metric
Scheme	http://schemas.ogf.org/occi/monitoring#
Title	A metric mixin
Related	None
Actions	
Attributes	

Attribute	**Type**	**Multiplicity**	**Mutability**	**Description**
value	string	1	Immutable	Based on unit, infer the value’s type.
timestamp	string	1	Immutable	ISO8601
samplerate	float	0..1	Mutable	hertz?
resolution	string	0..1	Mutable	SI prefix
unit	string	1	Immutable	if the metric being monitored is an OCCI attribute (e.g. occi.compute.memory) then the designated unit is used. Otherwise, use provider defined appropriate unit.

Example Metric Mixins

Compute Metrics

Compute Metrics Available in prototype:

- `cpu.user, cpu.sys, cpu.wait, cpu.lavg1, cpu.intsec, cpu.ctxsec,`
- `mem.tot, mem.buf, mem.used, mem.free, mem.cached, mem.swap`

These all MUST have their `rel` attribute set to “`http://schemas.ogf.org/occi/monitoring#metric`”

Title	**Term**	**Scheme**
cpu.user	user	http://iolanes.eu/occi/infrastructure/metric/compute/cpu#
cpu.sys	sys	http://iolanes.eu/occi/infrastructure/metric/compute/cpu#

cpu.wait	wait	http://iolanes.eu/occi/infrastructure/metric/compute/cpu#
cpu.lavg1	lavg1	http://iolanes.eu/occi/infrastructure/metric/compute/cpu#
cpu.intsec	intsec	http://iolanes.eu/occi/infrastructure/metric/compute/cpu#
cpu.ctxsec	ctxsec	http://iolanes.eu/occi/infrastructure/metric/compute/cpu#
mem.tot	tot	http://iolanes.eu/occi/infrastructure/metric/compute/memory#
mem.buf	buf	http://iolanes.eu/occi/infrastructure/metric/compute/memory#
mem.used	used	http://iolanes.eu/occi/infrastructure/metric/compute/memory#
mem.free	free	http://iolanes.eu/occi/infrastructure/metric/compute/memory#
mem.cached	cached	http://iolanes.eu/occi/infrastructure/metric/compute/memory#
mem.swap	swap	http://iolanes.eu/occi/infrastructure/metric/compute/memory#

Network Metrics

Network Metrics Available in prototype:

- `net.rxkbtot, net.txkbtot`

Title	Term	Scheme
net.rxkbtot	rxtot	http://iolanes.eu/occi/infrastructure/metric/network#
net.txkbtot	txtot	http://iolanes.eu/occi/infrastructure/metric/network#

Storage Metrics

Storage Metrics Available in the prototype:

- `dsk.readtot, dsk.writetot, dsk.readkbtot,dsk.writekbtot`

Title	Term	Scheme
dsk.readtot	readtot	http://iolanes.eu/occi/infrastructure/metric/storage#
dsk.writetot	writetot	http://iolanes.eu/occi/infrastructure/metric/storage#
dsk.readkbtot	readkbtot	http://iolanes.eu/occi/infrastructure/metric/storage#
dsk.writekbtot	writekbtot	http://iolanes.eu/occi/infrastructure/metric/storage#

Monitoring Behaviour

Discovery of Metrics

Sample Query Interface Rendering of Two Metrics offered by the provider

```
> GET /-/ HTTP/1.1

...
< Category: metric; scheme=”http://iolanes.eu/occi/infrastructure#”; class=”mixin”; title=”The
metric mixin”; attributes=”timestamp{immutable} samplerate resolution unit”
< Category: rxtot; scheme=”http://iolanes.eu/occi/infrastructure/metric/network#”; class=”mixin”;
title=”net.rxkbtot”; attributes=”timestamp{immutable} samplerate resolution unit”;
rel=”http://iolanes.eu/occi/infrastructure#metric”; location=”/metric/network/rxtot”
< Category: user; scheme=”http://iolanes.eu/occi/infrastructure/metric/compute/cpu#”;
class=”mixin”; title=”cpu.user”; attributes=”timestamp{immutable} samplerate resolution unit”;
rel=”http://iolanes.eu/occi/infrastructure#metric”; location=”/metric/compute/cpu/user”
…
```

Activate Specific Metrics

Add a mixin

```
> PUT /$RESOURCE/123-123-123

> category: rxtot; scheme='http://iolanes.eu/occi/infrastructure/metric/network#'; class='mixin'
> category: user; scheme='http://iolanes.eu/occi/infrastructure/metric/compute/cpu#';
class='mixin'
```

Update (configure) specific Resource Metric Parameters

By saying update, we mean to update the attributes of the metric and not the values that they represent. Only those metric attributes that are signalled as mutable can be modified.

Full Update

> PUT /$RESOURCE/123-123-123

Partial Update

> POST /$RESOURCE/123-123-123

Deactivate specific Resource Metric

PUT the resource representation without those metrics to be deactivated.

```
> PUT /$RESOURCE/123-123-123

< category: occi.compute.memory; scheme='http://iolanes.eu/infra/metrics#'; class='mixin'
< category: occi.compute.speed; scheme='http://iolanes.eu/infra/metrics#'; class='mixin'
```

Retrieve a Instantaneous Metric Value

```
> GET /service/123-123?attribute=occi.service.messages.throughput
< 204 OK HTTP/1.1
…
< x-occi-attribute: occi.service.messages.throughput=2.0
```

alternative:

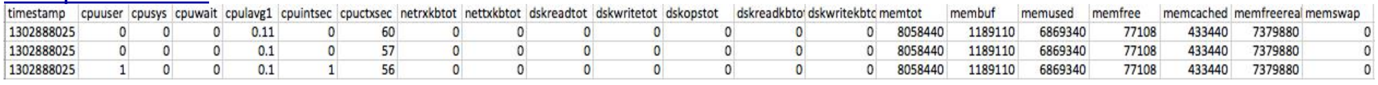
```
> GET /compute/123-123-123
> Category: occi.service.messages.throughput; scheme='http://iolanes.eu/infra/metrics#';
class='mixin'

< x-occi-attribute: occi.service.messages.throughput = 2.0
```

Retreive all Instantaneous Metric Values

```
> GET /service/123-123
< 200 OK HTTP/1.1
< Category: occi.service.messages.throughput; scheme='http://iolanes.eu/infra/metrics#';
class='mixin'
<
< x-occi-attribute: occi.compute.memory = 2.0
< x-occi-attribute: occi.compute.cores = 2
< x-occi-attribute: occi.service.messages.throughput = 2.0
```

Response Content as CSV

Content-type: text/csv
Sample data file



timestamp	cpuuser	cpusys	cpuwait	cpulavg1	cpuintsec	cpuctxsec	netrxkbtot	nettxkbtot	dskreadtot	dskwritetot	dskopstot	dskreadkbto	dskwritekbtc	memtot	membuf	memused	memfree	memcached	memfreereal	memswap
1302888025	0	0	0	0.11	0	60	0	0	0	0	0	0	0	8058440	1189110	6869340	77108	433440	7379880	0
1302888025	0	0	0	0.1	0	57	0	0	0	0	0	0	0	8058440	1189110	6869340	77108	433440	7379880	0
1302888025	1	0	0	0.1	1	56	0	0	0	0	0	0	0	8058440	1189110	6869340	77108	433440	7379880	0

Monitoring Push Notification

- What about the use of streamed HTTP traffic e.g. how node.js does some streaming operations for output?
  - Can a client can signal their preference?

Proposed changes to the OCCI specification

- Alter the Query Interface so default values can be defined for templating
  - add default value and include enums
- Add Notification mechanisms (look at SNIA's DDMI)
  - Also to chunked transfer encoding