# **Reference Architecture for SLA Management on a Microservices Environment**

## Context

Microservices is a growing pattern to build services in the cloud. In an infrastructure agnostic scenario where different PaaS (Platform as a Service) offers different capabilities and impose design choices like usage of black-boxed load balancers.

The following proposed architecture is a first proposal to allow SLA definition, measure, usage and enforcement in a Microservices ecosystem. It is proposed as a minimal impact addition to allow execution in multiple environments like IaaS, PaaS, Docker, etc.

## Basic Microservice Architecture

Basic architecture for a Microservice includes:

1. An optional load-balancer to handle and distribute the traffic. (Required if more than one nodes are used).
2. N nodes providing the service itself.
3. Optional storage system (cluster or standalone) providing a shared persistence layer.



**Note:** Traditionally, in a PaaS system the (1) Load Balancer is provided as a black-blox. Limited parametrization is allowed if any.

## SLA Enforcement and metrics

Therefore, if we want to instrument the service to allow:

1. SLA verification to allow/deny service or adjust QoS
2. SLA measure

we can implement it as the following architecture:



Where **SLA Service** and **Metric Store** are per-se Microservices as described in section 2 (with optional load-balancing functionality when needed).

Interceptors for SLA checks and data collectors on the domain microservice can be written as filters/plugins to enhance or decorate the domain services.

### API for SLA Service

**GET /deliverService** Query service over the SLA Service to indicate if service can be executed or not based on the current SLA Status.

Parameters

**?s=<serviceName>** Service Route. Sample: s=/bird/get

**&u=<userId>** UserId. Sample: u=123456789asf

Service Authentication: Authentication Basic: *appId:apiKey* pair

**Calculation:**

Based on request host (caller) information, service will be identified. Based on user-id provided, current plan will be recovered. Current SLA for this user then is calculated and responded.

**Response:**

Format: *application/json*

Positive response:

200 OK

Content-Type: application/json

{

"acept": true,

serviceProperties{

"quotaLimit": 100,

"quotaUsed": 80,

"quotaTimePeriodSec": 60, //60 secs

}

serviceConfiguration {

codingAlgorithm : “FAST”

bitrate= XXXX

MaxOptimizationTime : 100 (sec)

}

requiredMetrics[

// implicit

“requestSize”, “responseSize”,”responseTime” // domian-independent (preimplmented)

“optimizacionTime”, “ProblemSize”, //domain-specific (plugin neccessary

]

}

Negative response:

200 OK

Content-Type: application/json

{

"acept": false,

"reason": "Quota limit exceed.",

"quotaLimit": 100,

"quotaUsed": 100,

"quotaTimePeriodSec": 60,

"awaitBeforeRetrySec": 451 //Operation not allowed in the following 451 secs

}

Based on the response accept value the service will allow service execution or will deny it using a **403 Forbidden** Error.

If denegation reason is quota, then it will use **429 Too Many Requests**

Adding Quota information and reason if available as metadata into the client response to notify clients the denegation of service.

### API for Metric Store

**POST /measure** Sends one or more measures to a Metric Store Service.

Service Authentication: Authentication Basic: *appId:apiKey* pair

Sample body: Array of batch measures identifying the events and measures per event and source instance information provided (only once).

{

"src" : {

"host" : "node1234",

"env" : "qa",

"cluster" : "cl1.acme.com"

},

"measures"

Scope: /papamocas/qa/n1

Metrics:[{

"service": "/birds/get",

(Optional: Complete request)

"t": "2016-01-12T12:57:37.345Z",

"ellapsedMs": 350,

"cpu": 20.5,

"result": "200",

"userId": "13456789aadfc"

}{

//Extra measures (batch blocks. Block-size tuneable for performance vs real time information)

}]

}

## Additional infrastructure for SLA Life-cycle Management

Governify provides a reference implementations of tooling for SLA management to simplify the following operations:

* SLA definition
* Agreements definition
* Plans definitions
* SLA Status
* Metrics Store
* SLA Dashboard



Additionally, Microservices can resolve Authentication (AuthN) and Authorization (AuthZ) locally or externalized to a third party services.

## Dimensions to take in account when storing metrics

A single data metric store can be used to store several metrics. The following dimensions has been identified as relevant for indexing the data for later aggregation or comparison:

1. Microservice (logical name. Sample: *Papamoscas Service*)
   1. Environment (devel, qa, prod, etc.)
      1. Cluster (logical microservice deployed running)
         1. Instance (node name. Sample node12345)
2. User (Account)
   1. Group
   2. Department
   3. Organization
3. SLA Plan

Get qa.api.com/birds/1

* Nodo 1

Post datastore.governify.com/metrics

Scope : “….”

Get datastore.governify.com/agreements/

/papamoscas/USERID/availabilty

{

Scope = “papamosces/asdasd/asdasd/”

}

Get datastore.governify.com/agreements/

/papamoscas/scope/responseTime

deliverService

{{SERVICIO}}{{USER-API-KEY}}/

Each scenario have a different Scope modes (even multi-dimensional scope)

Different ScopeGenerator /Builder

Predefined:

Basic (Unidimensional based on URL: /domains) for clusters nodes, …