Cloud Computing

Part 4: Research

University of Padova
Department of Mathematics

July 14, 2014

Summary

1. Active Cloud Computing research fronts
2. Elastic Scalability, PaaS and SOA
3. Monitoring and Control
4. Private and Hybrid Cloud
5. Multitenant data and app architecture

Research fronts - Open challenges

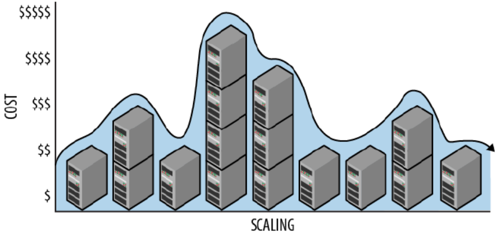
- PaaS, SOA, Elastic Scalability ⇒ role of PaaS, SOA applied to PaaS
- Monitoring and Control SPI stack ⇒ get indicators, analyze and forecast, act
- Private and Hybrid Cloud ⇒ hw consolidation, bursting, federation
- Multitenancy data and apps ⇒ cloud-enabled application arch, economy of scale

*Approach: set objectives ⇒ get results !!
Agile method and measurable progress to solve enterprises use cases

- theory = comprehension
  - review literature: books, articles, documentation, tutorials
  - survey of tech and and comparison on paper of spec & features
- practice = application + experience
  - technical knowledge & solution convenience
  - use open-source technologies and tools to build solutions (PoC)

Elastic Scalability and the PaaS layer - Business concerns

- Rapid elasticity, or Elastic Scalability, allows the cloud provider to dynamically adapt service capacity according to the use profile determined by customers
- an ISV supplies a service at the SaaS level, normally via a web application. As part of that, it needs the infrastructure to support the service that is selling
- the provider wants to avoid over-provisioning of resources so to take advantage of the infrastructure in the most cost-effective way (dynamic compromise)
- The PaaS can help



Elastic Scalability and the PaaS layer - PaaS role

- PaaS = capability to develop, deploy, and orchestrate onto the cloud
- analogy between PaaS and traditional OS ⇒ "cloud OS"

- the PaaS layer and its role is not yet completely understood
  - technology concerns
  - deployment framework model
  - elastic scalability
  - service orchestration

- PaaS can be very powerful
  - provisioning responsibilities can be better apportioned
  - SaaS provider can escape from lock-in relations with the particular infrastructure used
  - elastic scalability of services can be implemented openly

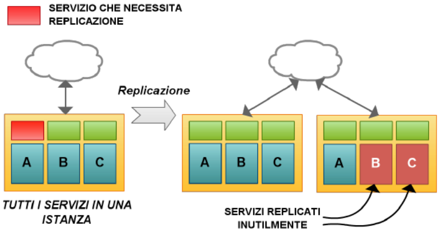
Service-orientation and SOA - Overview

- Service-orientation = architectural paradigm/approach that adopts the concept of services as the main building blocks of application and system development
  ⇒ core reference model for cloud computing systems
- SOA = Service Oriented Architecture
- Its main principles are:
  - standardized contracts
  - loose coupling
  - abstraction
  - reusability
  - autonomy
  - statelessness
  - discoverability
  - composability

Granularity matters! ⇒ SOA has the right granularity for business processes distributed over a landscape of existing and new heterogeneous systems that are under the control of different owners

Service-orientation and SOA - Cacco's remark

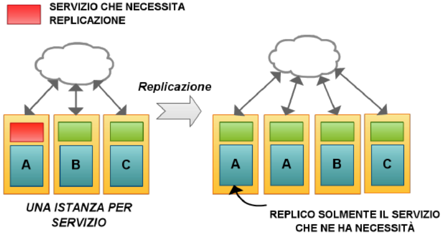
Replicate only the components that need it!



All services within a unique instance

- to scale a service I have to replicate the instance that provides all services

$\Rightarrow$ monolithic = waste



One service within each instance

- to scale a service I can replicate the instance that provides that service

Service-orientation and SOA - Jolie

- Jolie is a service-oriented language born in Bologna (SOC research scope)
- built on a strong mathematical foundation (SOCK)
- based on Java, provides an intuitive and easy to use C-like syntax to deal with the implementation of architectures made of services
- service as a first-class citizen

Example: minimal multi-threading server (service) with Jolie:

SOURCE

```
/* server.ol */
execution { concurrent }
interface TwiceInterface {
    RequestResponse:  twice( int )( int )
}
inputPort TwiceServiceIn {
    Location:  "socket://localhost:8000"
    Protocol:  sodep
    Interfaces:  TwiceInterface
}
main {
    twice( number )( result ) {
        result = number * 2
    }
}
```

EXECUTION

```
$ jolie server.ol
```

SOURCE

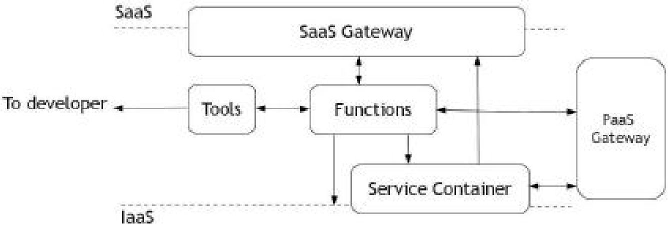
```
/* client.ol */
include "console.iol"
interface TwiceInterface {
    RequestResponse:  twice( int )( int )
}
outputPort TwiceServiceOut {
    Location:  "socket://localhost:8000"
    Protocol:  sodep
    Interfaces:  TwiceInterface
}
main {
    twice@TwiceServiceOut( 5 )( response );
    println@Console( response )()
}
```

EXECUTION

```
$ jolie client.ol
10
$
```

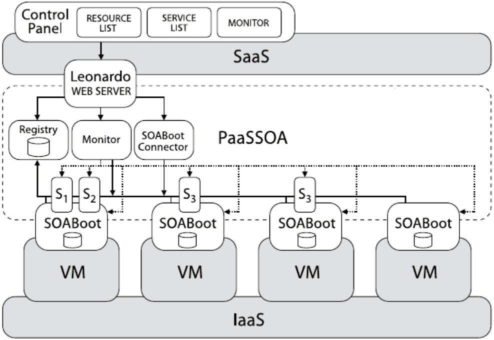
PaaSSOA - Inception

- Jolie is the enabling technology of PaaSSOA ⇒ deploying SOA principles could greatly facilitate build an open cloud platform
- the primary purpose of the PaaSSOA project is to explore the role of a SOA PaaS layer in the Cloud SPI stack (it began in Bologna)
- open PaaS framework ⇒ understanding interaction protocols between the SPI stack layers
- Vision ⇒ prototype framework to control the deployment and the execution of Jolie services, where the latter are the building blocks of cloud applications



Prior PaaSSOA - Architecture & features

- PaaSSOA resource model provides base abstractions for Jolie services at two levels
  - virtualization layer = capabilities for deploying and executing VM images
  - SOABoot services = container service of PaaSSOA, one for each VM
- PaaSSOA can manage two types of resources
  - VM hosts = virtualized environments controlled by SOABoots
  - Jolie services = services deployed and executed within SOABoots



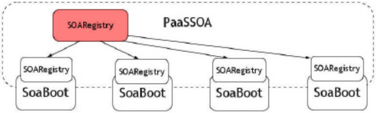
Prior PaaSSOA - Web Dashboard - Registry List

PaaSSOA control Panel

See services
See registries
See runtime monitor

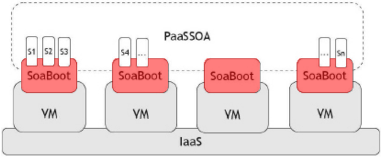
Domain	Protocol	Location	Actions
passoa/soaboot/1.1.1.1/	sodep	socket://localhost:10000	Get Services

- SOARegistry ⇒ keep track of resources
- federated architecture
- 2nd level registry:
  - store Jolie services descriptions
  - promote res names to the upper level
- 1st level registry:
  - centralize requests
  - forward request to 2nd level registry, which has the resource

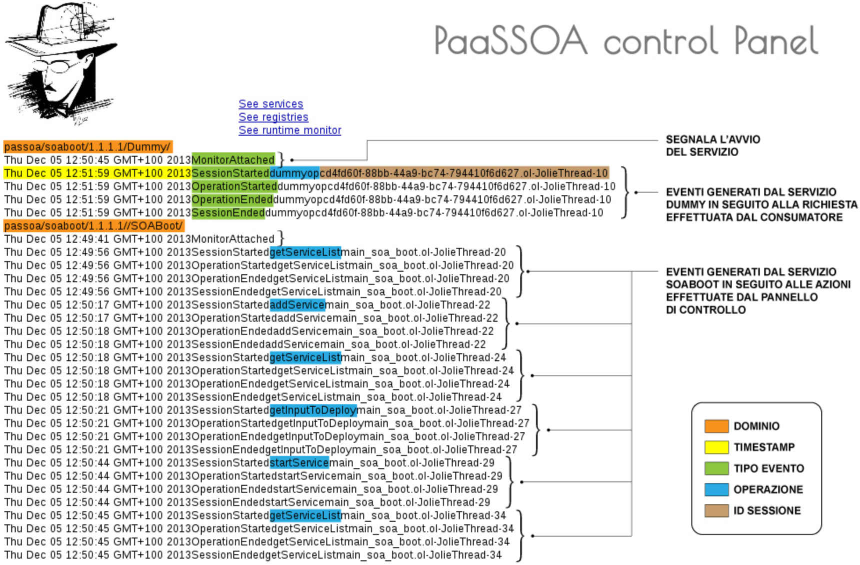


Prior PaaSSOA - Web Dashboard - Service List

- SOABoot = Jolie services container
- list of services contained by each SOABoot
- actions on services:
  - deploy, start, stop, undeploy, details
  - make available a subset of "ports"



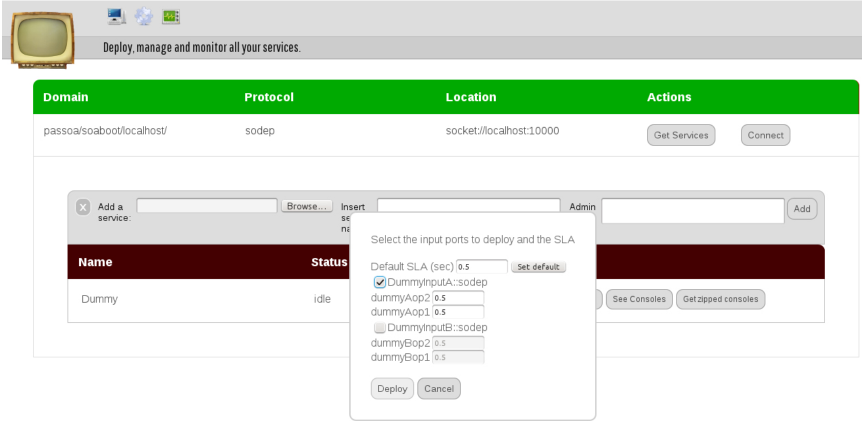
Prior PaaSSOA - Web Dashboard - Service Monitor



Prior PaaSSOA - Lack of elastic scalability

- the idea of PaaSSOA seems promising but... it lacked the elastic scalability characteristic to demonstrate its usefulness as a cloud PaaS layer
- an entire master thesis dedicated on the theme of elastic scalability and runtime enforcement of Service Level Agreements (by Zuccato A.)
- challenges:
  - SLA definition
  - SLA enforcement
  - events collection
  - events interpretation
  - actions to perform
  - test
  - paradigm shift
  - new technologies

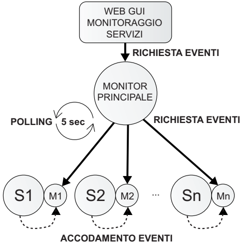
Zuccato's PaaSSOA - Enaling SLA input



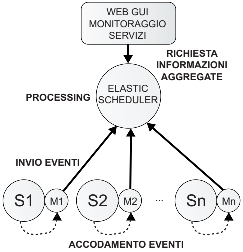
- SLA = limit on service response time
- SLA input enabled on the web dashboard
- service start requires SLA to be specified

Zuccato's PaaSSOA - Changed monitoring modality

- before ⇒ pull mode:
  - events gathered and queued by Jolie interpreter
  - events request performed by main monitor
  - reply by 2nd level monitor empties internal queues
  - events gathered through polling ⇒ disadvantageous



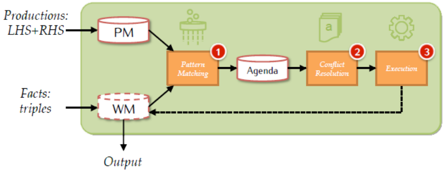
- after ⇒ push mode:
  - event generation remains unchanged
  - Elastic Scheduler receives events
  - 2nd level mon send events automatically upon queuing
  - enable "real-time" events processing



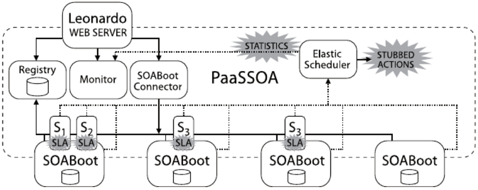
Changes impact also the SOABoot service

Zuccato's PaaSSOA - Drools inference engine integration

- Drools working:
  - Production Memory ⇒ Rules
  - Working Memory ⇒ Facts
  - Facts assertion in WM ⇒ pattern matching with LHS
  - Agenda ⇒ schedule rules execution
  - side effect produced by RHS execution ⇒ forward chaining



- Elastic scheduler integration:
  - ES core service in PaaSSOA
  - Java Drools embedded in a Jolie wrapper (Elastic Scheduler)
  - definition of declarative rules
  - stats and control on services
  - input ⇒ services events
  - output ⇒ stubbed actions



Zuccato's PaaSSOA - Declarative logic implementation

- Events aggregation:
  - Events and Facts = Java objects
  - objects type: Service, Session, Operation
  - rules to match the start event correlated with the received end event
  - based on timestamp attribute
  - calculate and store statistics

```
rule "updateOperation"
salience 1
when
    $e1: PaaSSOAMonitorEvent( $dom: domain, $sid: SessionId,
                              $opname: OperationName,
                              type matches "OperationStarted",
                              $ts: timestamp )
    $e2: PaaSSOAMonitorEvent( domain == $dom,
                              SessionId == $sid,
                              OperationName == $opname,
                              type matches "OperationEnded",
                              timestamp >= $ts )
    $s: Service( name == $dom, $maddr: email_address,
                 $isb: is_soaboot )
    $op: Operation( name == $opname, service == $dom,
                    $sla: response_time_threshold,
                    $rst: stats_reset )
    $js: DroolsJavaService()
then
    long duration = $e2.getTimestamp() - $ts;
```

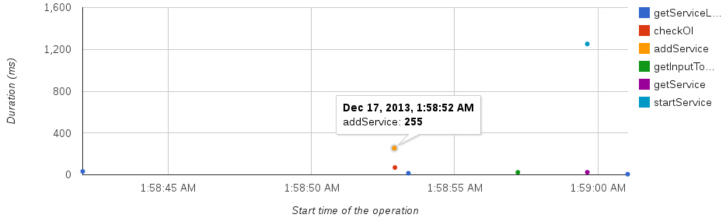
- Enablement of elastic capacity:
  - Drools Fusion ⇒ temporal relationship between events (Allen operators)
  - rule replicates service when the end event doesn't arrive within the SLA threshold
  - replication is only stubbed (test toolkit)

```
rule "breakSLA"
salience 1
when
    $e1: PaaSSOAMonitorEvent( $dom: domain, $sid: SessionId,
                              $opname: OperationName,
                              type matches "OperationStarted" )
    not PaaSSOAMonitorEvent( domain == $dom,
                              SessionId == $sid,
                              OperationName == $opname,
                              type matches "OperationEnded",
                              this after [0ms,500ms] $e1 )
    $op: Operation( name == $opname, service == $dom,
                    $sla: response_time_threshold )
    not ActionEvent( domain == $dom )
    $js: DroolsJavaService()
then
    $js.actionOnPaaSSOA( $dom, "AddInstance" );
```

Zuccato's PaaSSOA - Service Monitor redesign

MONITOR: see monitor details of your services.

Domain	Statistics	Actions
passoa/soaboot/localhost/	Service type: SOABoot service Started on: Tue Dec 17 01:58:26 CET 2013 Memory used by JVM on start: 17.1 MB Total number of sessions executed: 8 Total number of requests executed: 8 Total duration of sessions: 1.688 sec Total duration of operations: 1.68 sec Total memory used by sessions: 8.52 MB Total memory used by operations: 8.16 MB SESSION: getServiceList | Sessions: 3 Duration: (avg 19, max 35, min 8) ms | Memory: (avg 1.1, max 1.45, min 0.77) MB OPERATION: getServiceList | Requests: 3 Duration: (avg 18, max 33, min 6) ms | Memory: (avg 1.1, max 1.45, min 0.77) MB ...	Hide Services



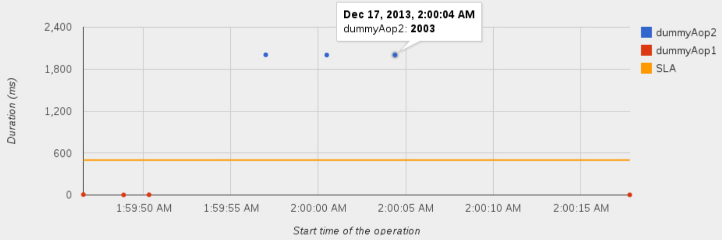
Zuccato's PaaSSOA - Service Monitor redesign

passoa/soaboot/localhost
/Dummy

Service Type: Deployed service
Admin email: zukzuc@gmail.com
Started on: Tue Dec 17 01:59:01 CET 2013
Memory used by JVM on start: 1.21 MB
Started on: Tue Dec 17 01:59:01 CET 2013
Memory used by JVM on start: 1.21 MB
Total number of sessions executed: 7
Total number of requests executed: 7
Total duration of sessions: 6.029 sec
Total duration of operations: 6.022 sec
Total memory used by sessions: 1.51 MB
Total memory used by operations: 1.41 MB
Total number of SLA violations: 3
Total duration of SLA violations: 4.51 sec

SESSION: dummyAop2 | Sessions: 3
Duration: (avg 2004, max 2007, min 2002) ms | Memory: (avg 0.36, max 0.37, min 0.33) MB
OPERATION: dummyAop2 | Requests: 3 | SLA violations: 3
Duration: (avg 2003, max 2005, min 2002) ms | Memory: (avg 0.36, max 0.37, min 0.33) MB | SLA: (limit 0.5, over 4.51) sec

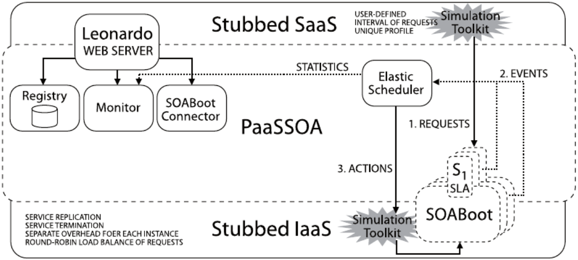
...



Zuccato's PaaSSOA - ES Test Toolkit

Ad-hoc toolkit implemented to test the ES to overcome current limitations

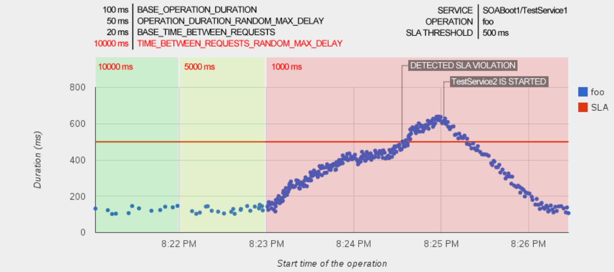
- service simulation
  ⇒ event generation
- use profile
  ⇒ sequential ops
- 1 iter = 1 simulated op
- 4 parameters
  - 2 for op duration
  - 2 for time between ops
- enable overload
- enable effect of actions
  - service replication
  - service termination
- enable load-balancing



Zuccato's PaaSSOA - ES Test (upward scalability)

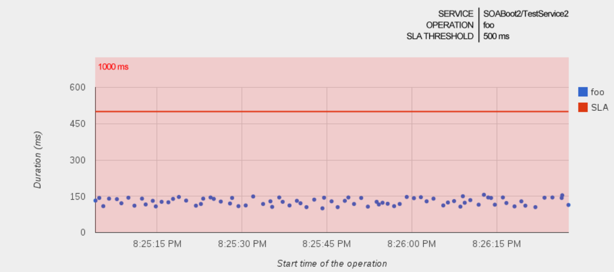
TestService1

- increased frequency ⇒ overload
- detected violation ⇒ replication
- new instance start delay



TestService2 (replicated instance)

- new SOABoot
- new service instance
- load-balancer comes into play



Zuccato's PaaSSOA - ES Test (downward scalability)

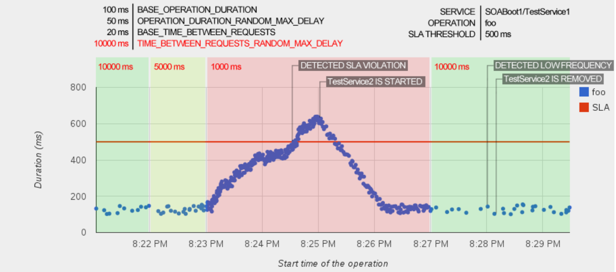
more difficult!

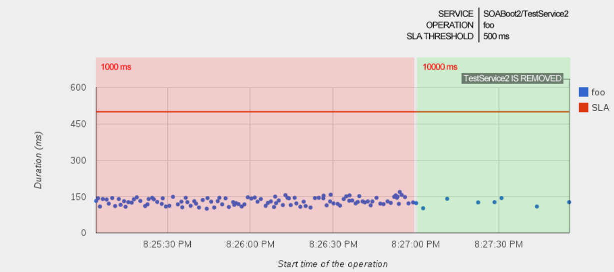
TestService1

- diminished frequency
  ⇒ terminate instances
- optimization
  ⇒ reduce cost!
- instance termination delay

TestService2 (replicated instance)

- instance termination
- SOABoot termination
- effect of rebalancing
- can lead to trashing (now one less cannot be enough)





PaaSSOA post Zuccato - Hurdles and new challenges

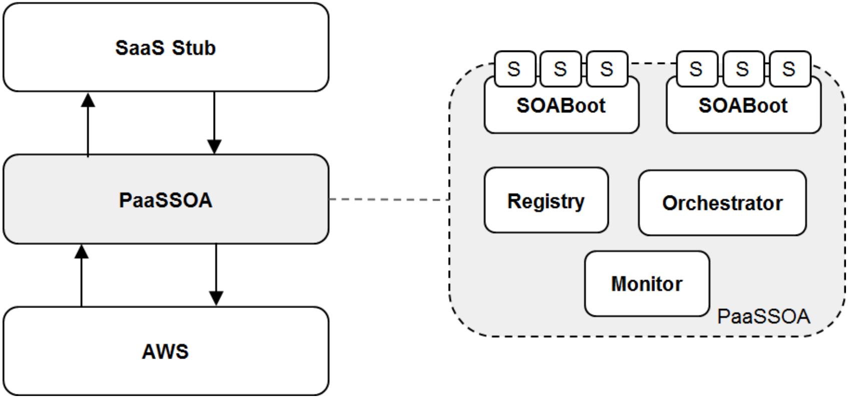
- hurdles in PaaSSOA:
  - no real services to test
  - no real IaaS (tested in a local environment)
  - no implemented PaaSSOA replication and migration feats to manage services

  ⇒ thus the Elastic Scalability test was a mere simulation !!
- another entire master thesis dedicated on the theme of elastic scalability (by Baraldo V.)

⇒ this time the research involves the integration of PaaSSOA with a real IaaS

- with a partial redesign, PaaSSOA can reside on a real state-of-the-art IaaS such AWS
- abstraction layer that helps to decouple logic service from real service instances

Baraldo's PaaSSOA - High-level architecture

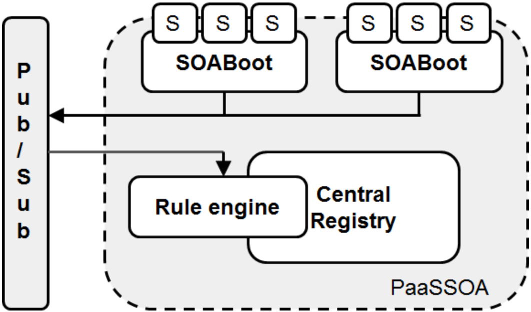


Baraldo's PaaSSOA - Key components

- A SOABoot is a PaaS representation of a VM provided by the IaaS
  - Every SOABoot can contain multiple services
- The registry keeps trace of deployed services and SOABoots mapping
- The monitor collects and present data received from SOABoots
- The orchestrator helps to manage services (automatically and manually)

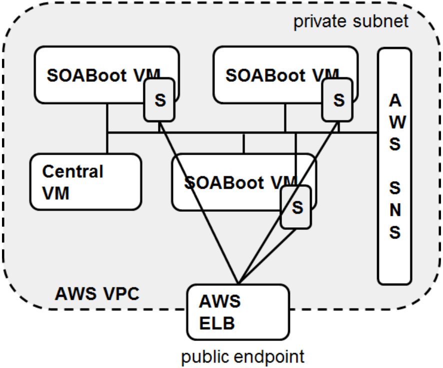
Baraldo's PaaSSOA - Publish/subscribe

- We need to obtain loose coupling
- A publish/subcribe broker is a possible solution
- It helps to obtain scalability and elasticity



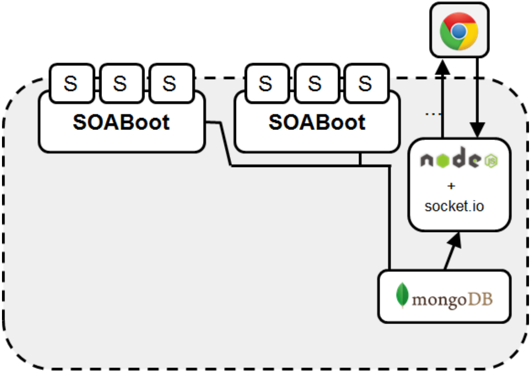
Baraldo's PaaSSOA - AWS Architecture

- Environment boundaries
  - VPC
  - Private subnets (scalable resource group)
- Load balancing
  - AWS ELB (2-levels dispaching)



Baraldo's PaaSSOA - Dashboard and monitor

- A new dashboard that becomes a simple orchestrator
- A scalable logging system where every entity can push log events
- Information could be available as soon as possible
  - Full-push architecture



Baraldo's PaaSSOA - Current status and Future Prospects

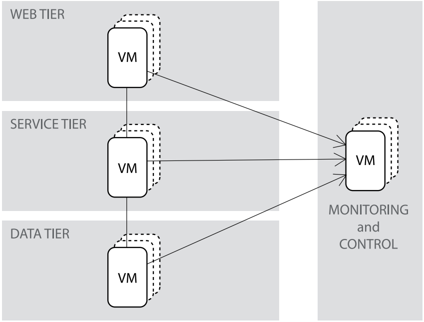
- This is a prototype: it shows the potential of the SOA principles injected in a real cloud stack
- It needs reengineering based on the experimental results
  - What are the key system components?
  - What components should be decomposed?
- Outlook:
  - Improve the elasticity mechanisms
  - Improve SOA components' implementation
  - Explore other technology stacks

Monitoring and Control - Overview

- Cloud application monitoring problem ⇒ 3 dimensions:
  - physical (hosts)
  - logical (services)
  - time
- Need to consider two sides:
  - events collection side (input)
  - actions execution side (output)
- Then we need to put logic in-between ⇒ how
  - collection side ⇒ event processors = collect, parse, filter, transform, transfer events
  - actions execution side ⇒ configuration tools = streamline the task of configuring & maintaining servers
  - in-between current state or knowledge ⇒ production rule system = execute productions in order to achieve some goal for the system

Monitoring and Control - Architecture

- VMs on each tier are equipped differently:
  - web tier: web servers, application servers (front-ends)
  - service tier: interpreters and compilers, services (back-ends)
  - data tier: DBMS relational, NoSQL, storage volumes (back-ends)
- monitoring and control tier is the "brain"
- the first step is to equip each VM with an active agent
  - gathering of system and service performance
  - fixed footprint ⇒ round robin DB
  - agents send statistics and events to monitoring nodes (async)



Monitoring and Control - Current status

- gathering basic and general system performance statistics with Collectd
- graph visualization with Graphite

Monitoring and Control - Next step

- test a minimal application simulating the presence of users
  - differentiate VMs, equip VMs with the real tools
  - specialize configuration of plugin to match tier
  - automate the provisioning of new VMs

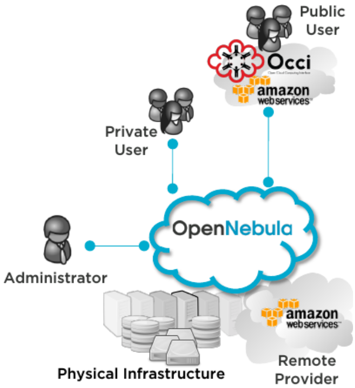
Private and Hybrid cloud - Overview

- investigate the use of open-source cloud managers to build private and hybrid cloud
- challenges of IaaS clouds:
  - How do I provision a new VM?
    Image Management & Context
  - Where do I store the disks?
    Storage
  - How do I set up networking for a multitier service?
    Network & VLANs
  - Where do I put my web server VM?
    Monitoring & Scheduling
  - How do I manage any hypervisor?
    Virtualization
  - Who has access to the Cloud's resources?
    User & Role Management
  - How do I manage my distributed infrastructure
    Interfaces & APIs?

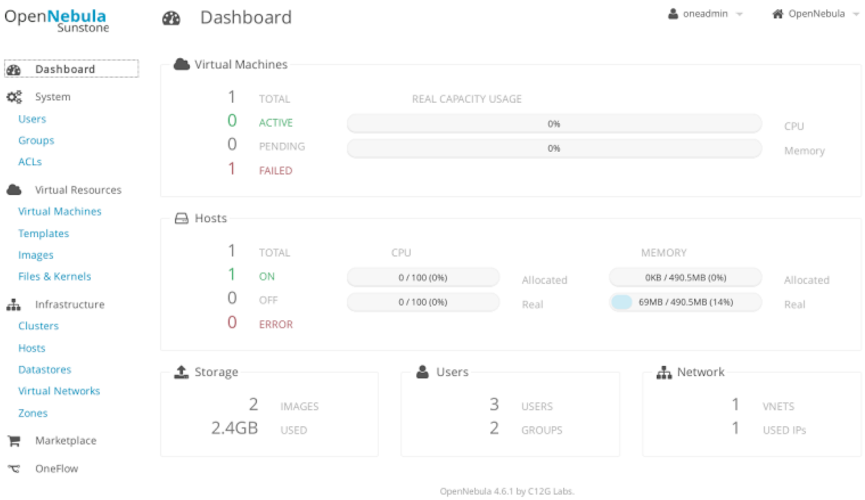
Private and Hybrid cloud - OpenNebula

OpenNebula = uniform management layer that orchestrate multiple technologies

- Data Center Virtualization Manager
  - Open-source Apache license
  - Interoperable, based on standards
  - Adaptable
- Private Clouds ⇒ virtualize your on-premise infrastructure
- Pubic Clouds ⇒ expose standard cloud interfaces
- Hybrid Clouds ⇒ extend your private cloud with resources from a remote cloud provider
- Ready for end-users
  - Advanced user management
  - CLI and Web Interface

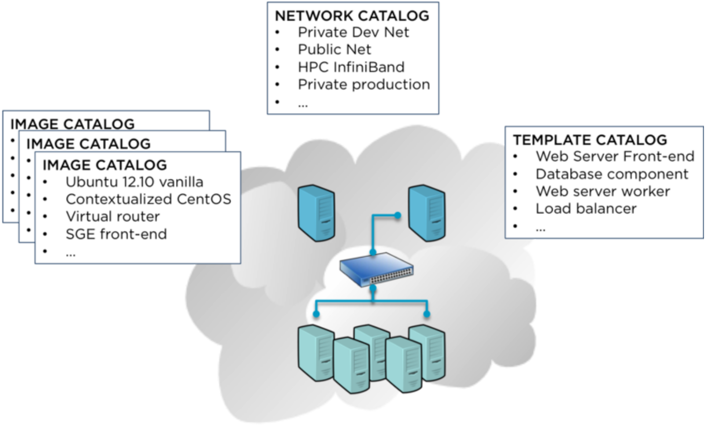


OpenNebula - Quick tour of main features



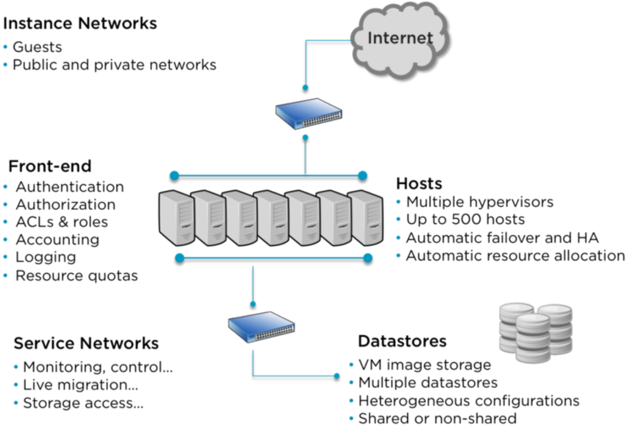
OpenNebula - What does it offer to Cloud Consumers?

- Image Catalogs
- Network Catalogs
- VM Template Catalog
- Virtual Resource Control and Monitoring
- Multi-tier Cloud Applicaion Control and Monitoring



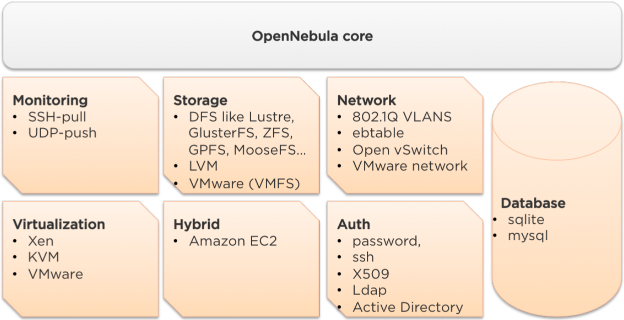
OpenNebula - What does it offer to Cloud Operators?

- Users and Groups
- Virtualization
- Hosts
- Monitoring
- Accounting
- Networking
- Storage
- Security
- High Availability
- Clusters
- Multiple Zones
- VDCs
- Cloud Bursting
- App market



OpenNebula - What does it offer to Cloud Builders?

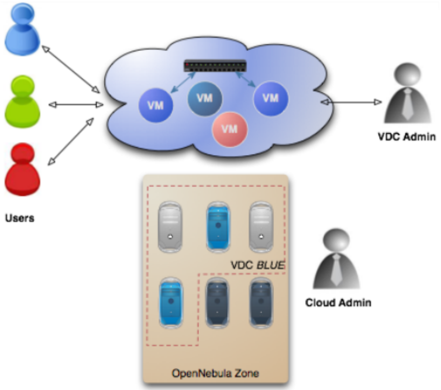
- User Management
- Virtualization
- Monitoring
- Networking
- Storage
- Databases
- Cloud Bursting



OpenNebula - Current status and Use Cases

Virtual private Cloud Computing

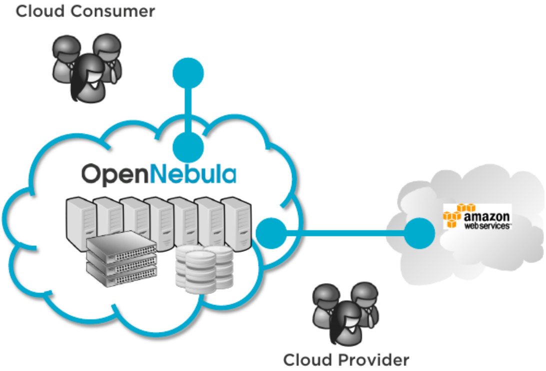
- Typical scenario in large organizations and cloud providers
- On-demand provision of fully-configurable and isolated Virtual Data Center (VCD) with full control and capacity to administer its users and resources



OpenNebula - Current status and Use Cases

Hybrid Cloud Computing

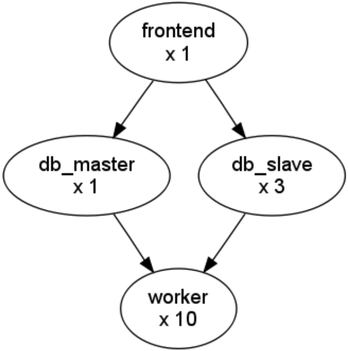
- Extension of the local private infrastructure with resources from remote clouds
- Cloud bursting to meet peak or fluctuating demands



OpenNebula - Current status and Use Cases

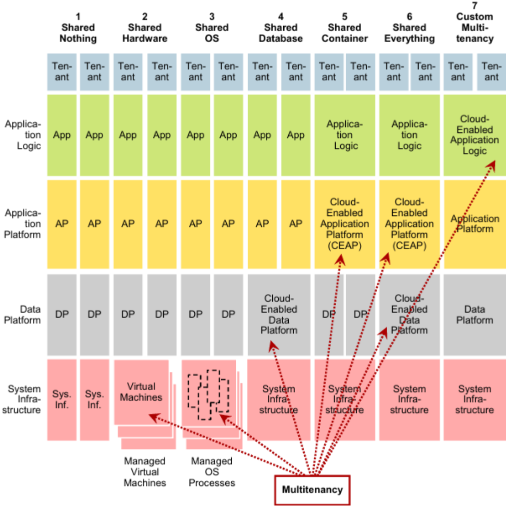
Managing Multi-tier Applications

- define, execute and manage multi-tiered applications
- interconnected Virtual Machines (roles, cardinality)
- each group of Virtual Machines is deployed and managed as a single entity



Multitenant data and app - Overview

Investigate how to design and to build multitenant data and apps



Focus on:

- 3 - Shared OS
  ⇒ with Docker
- 4 - Shared Database
  ⇒ separate schema for each tenant
  ⇒ with PostgreSQL

Multitenant data and app - Current status

A solution for multitenancy data $\Rightarrow$ migrate the schema and the data of a tenant across VMs on data tier

- identical schema for each tenant
- logical separation
  $\Rightarrow$ more physical integration = more difficult to ensure logical isolation
- performance management
  $\Rightarrow$ common startegy is to deploy resource-hungry tenants alongside tenants with low resource demands

Building a very minimal application that use the underlying data tier
$\Rightarrow$ moving schemas around

Multitenant data and app - Next steps

Add pieces to the stack ⇒ complement the PoC with the monitoring solution (collectd + graphite).

We will need also the logic !!

- The ultimate goal is to achieve a prototype cloud platform:
  - continuously monitoring
  - reorganizing (moving tenants around)
  - horizontally scaling service instances

  ⇒ it's all done transparently

References

- Cloud Architecture Patterns. Wilder - Chapter 4
- Master Thesis - GeoServer nel Cloud. Un caso di studio sulle modifiche architetturali nel passaggio a piattaforma Cloud. Cacco F.
- Master Thesis. PaaSSOA, a support system for the specification and runtime verification of Service Level Agreement in the Cloud. Zuccato A.
- PaaSSOA: An Open PaaS Architecture for Service Oriented Applications. Guidi, C., Anedda, P., Vardanega, T.
- OpenNebula documentation `http://docs.opennebula.org/4.6/`
- Gartner Reference Model for Elasticity and Multitenancy. Natis `https://www.gartner.com/doc/2058722/gartner-reference-model-elasticity-multitenancy`