RFP 133 Cloud Computing

Proposed Final Draft

40 Pages

Abstract

An investigation into the possibility for standard OSGi Service APIs and new features for the framework that can be useful for using OSGi in the context of cloud/grid/utility computing. The unique aspects of cloud computing require services for awareness and control of the environment as well as more control over the instantiation of new machines and new processes. Many of these functions are also required for computing in large data centers.

Copyright © OSGi Alliance 2011.

All company, brand and product names contained within this document may be trademarks that are the sole property of the respective owners.

The above notice must be included on all copies of this document that are made.

1 Document Information

Table of Contents

1 Document Information **2**
Table of Contents 2
Terminology and Document Conventions 5
Revision History 5

2 Introduction **8**

3 Application Domain **8**
Introduction 8
Advantages 10
Disadvantages 11
Scalable Application Architectures? 13
Terminology & Concepts 15
Assumptions 22

4 Applicability of OSGi? **23**
Relevance of OSGi to Cloud? 23
Existing OSGi Standards 25

5 Problem Description **26**
Scale & Volatility 26
Isolation 26
Configuration 26
Resource Capabilities 27
Resource Discovery 27
Resource Provisioning 27
Node Hosting Behaviours 28
Starting an OSGi Framework 28
Service Discovery 29
System Provisioning 29
System Validation 30
Monitoring 30
Logging & Audit 31
Post Mortem/Debugging 31

6 Use Cases **32**
Purpose 32
Use Case - The Customer 32
Use Case - The Cloud Provider 32
Use Case - Agility 32
Use Case – LockIn 33
Use Case – Hybrid 33
Mixing private and public clouds 33

Development.......... 34

7 Requirements.......... 35

Management.......... 35

Metadata.......... 35

Event notifications.......... 36

Instances.......... 36

Various.......... 36

8 Deferred Requirements.......... 37

9 Appendices 38

Cloud Providers.......... 38

Amazon.......... 38

Google Apps.......... 38

Rackspace.......... 38

Microsoft Windows Azure.......... 39

IBM 39

Private Cloud Solutions 39

Red Hat.......... 39

Current best practices of virtual-machine-based management and deployment 40

10 Document Support.......... 41

References.......... 41

Author's Address.......... 41

End of Document.......... 42

Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in .

`Source code is shown in this typeface.`

Revision History

The last named individual in this history is currently responsible for this document.

Revision	Date	Comments
0.1	07/04/10	Peter Kriens, aQute: Initial
0.2	13/07/10	Richard Nicholson, Paremus: Initial Recut
0.3	27/07/10	Richard Nicholson, Paremus: On-going Major reworking – stopped at 4.7.2
0.4	01/09/10	Richard Nicholson, Paremus: Reworked up to & including Section 5
0.5	16/09/10	Richard Nicholson, Paremus: Updated comments on framework personalisation, capabilities / requirements and Use Cases
0.6	10/09/10	Mark Little, Red Hat David Bosschaert, Red Hat Additions to use-cases and requirements.
0.7	29/10/10	David Bosschaert, Red Hat Expanded cloud services section.
0.8	15/11/10	Richard Nicholson, Paremus: Major re-work of sections 3,4,5,7 & inclusion of diagrams.
0.9	28/01/11	Jan S, Rellermeyer Major re-work.
0.9.1	14/02/11	Jan S, Rellermeyer Minor editorial changes in response to the Berlin F2F discussion.
0.9.2	20/03/11	David Bosschaert Put images that went missing back in.
0.9.3	23/03/11	Mark Little Added text about academic research and EU funded efforts.

Revision	Date	Comments
0.9.4	23/03/11	David Bosschaert Added references to Cloud efforts in Java EE 7 and Java SE 8
0.10.	May 2011	David Bosschaert Incorporated feedback from Jclouds, revamped the requirements section.
0.11	16/05/11	David Bosschaert Incorporate feedback from the requirements review call. Added Cloud Domain description.
0.12	04/06/11	Some restructuring and general clean-up.
0.13	23/06/11	Accepted all changes and prepare for vote.

2 Introduction

The OSGi Alliance held a joint workshop March 25, 2010 in the Santa Clara Hyatt. The goal of the workshop was to find out what role OSGi could play in the Cloud.
Obviously OSGi is very well suited for remote deployment, a characteristic that seems extremely applicable for cloud based applications.

The workshop discussed a number of important topics and potential requirements. This RFP was created as a working document to further investigate this area and provide a basis for a subsequent set of Cloud related RFCs to be pursued by the Enterprise Expert Group.

3 Application Domain

Introduction

Cloud Computing is a major IT trend; possibly qualifying as a real paradigm shift whose eventual impact may exceed that of the move to client server computing in the 1980's. *Cloud Computing* service are already extensively used to provide cost effective third party hosting for some categories of traditional application; differing only in the granularity of utilisation and subsequent payment from the previous utility compute service providers. However, for applications architected in the appropriate manner, *Cloud Computing* also offers the promise of massive 'just-in-time' resource elasticity.

The National Institute of Standards and Technology (NIST) (see http://csrc.nist.gov/groups/SNS/cloud-computing/index.html) define *Cloud Computing* as:

> "*Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.*"

The NIST go on to describe the essential characteristics that collectively define Cloud Computing. Note that NIST definitions for IaaS, PaaS 7 SaaS have been submitted to ISO for consideration.

There is perhaps no better demonstration of the importance of the *Cloud Computing* than by the number of standards bodies and industry collaborations that are involved; see (http://cloud-standards.org/wiki/index.php?title=Main_Page) and attempt to map these standardisation activities to the Cloud service model in the following presentation http://cloud-standards.org/wiki/index.php?title=Cloud_standards_overview.

Distributed Management Task Force's (DMTF) recently formed Cloud Management WG (CMWG) are focused on resource management aspects of Infrastructure as a Service (a.k.a IaaS – see next section) including constraint and policies, SLAs, QoS along with modelling considerations for managing utilisation, provisioning, monitoring/reporting, auditing and cloud service lifecycle. It is also stated that the CMWG will need to work closely with the OVF workgroup on the evolution of that model as middleware and applications are brought into the composite image, including the aspects of personalisation and policy controls for those elements within an OVF package. With respect to OVF, DMTF have received initial contribution from VMware. Meanwhile Fujitsu, and recently Oracle, have contributed REST based API's via which a customer might create an appropriate set of infrastructure resources. For an overview of DMTF cloud related activities see; http://www.dmtf.org/about/cloud-incubator/DSP_IS0101_1.0.0.pdf. Further open source REST API's for cloud virtual machine provisioning include JCloud (http://code.google.com/p/jclouds/) & Red Hat backed deltaCloud project (see – http://deltacloud.org/).

In addition to these standards activities, several attempts have been made by commercial led ventures to define reference architectures or open-source communities. The most recent of these being the http://www.openstack.org/ consortium formed by Rackspace and NASA, whose stated aim it to build an Infrastructure as a Service (IaaS) framework capable of support 1,000,000 physical server nodes and 60,000,000 VM instances. In contrast, OW2's Open Source Cloud initiative has more emphasis of model driven artefact deployment (virtual machine and other artefacts – via JASMINe) and resource optimisation (Entropy & ProActive).

There has also been substantial academic research, including the often cited “Above the Clouds: A Berkeley View of Cloud Computing” (http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf), as well as work on Cloud Security issues by MIT and others (http://portal.acm.org/citation.cfm?id=1653687). Many government research organisations are committing research and development effort into the Cloud, either purely through universities, or with the collaboration of businesses: e.g. the Cloud-TM effort (http://www.cloudtm.eu/) is a multi-million dollar, 3 year program.

From a storage perspective Cloud providers have already implemented 1st generation persistence solutions such as S3 (Amazon) & BlobStore (Rackspace) and Blob Service (Microsoft Azure). Meanwhile Storage Network Industry Association (SNIA) are working on storage discovery and management standards; see http://www.snia.org/forums/csi/knowledge/tech_activities/standards/curr_standards/cdmi/CDMI_SNIA_Architecture_v1.0.pdf).

A valuable *reality check* is provided by the following article http://www.sector.ca/Presentations09/Cloudifornication-SecTor.pdf which highlights a number of common misconceptions and risks concerning *public Cloud Computing.*

Finally, the JCP is preparing for Cloud support in future versions of JavaSE and JavaEE. JSR 342 (http://jcp.org/en/jsr/detail?id=342), the JavaEE 7 JSR, makes cloud its main theme, mentioning PaaS support, multi-tenancy and elasticity. To properly support multi-tenancy within the VM some form of modularity and isolation is required and OSGi modularity is available for JavaSE today.

Advantages

Traditional enterprise applications are built around a number of static coarse grained software components. Such Systems can be large and distributed in nature: for example enterprise Compute Grid solutions routinely consist of thousands of processing nodes. However, such solutions are static functional silos. Hence, one significant difference between cloud computing and more traditional computing is not necessarily 'scale' but the dynamic nature of these resources; the fact that the quantity of available compute resource may change over time.

Cloud Computing is often cited (some perceived) has having the following advantages:

1. No up-front investment in hardware and licenses; computing cost is based on actual consumption (pay-as-you-go). This is business enabler as it enables company start-ups to provide innovative services without investing in data centers that can scale when the start-up is successful.

2. Applications can be scaled based on demand by load balancing across available instances; this a cost effective to handle highly volatile customer load. Note – this is only true for correctly architected applications.

3. Cloud based applications leverage many instances and so have inherent failover capabilities, so masking failures from users of those Services. Note – this is only true for correctly architected applications.

4. Cloud providers have multiple data centers located in different geographic locations. For example, Amazon provides datacenters spread around the US as well as in Ireland and Asia. Location can provide affinity to the data and user for performance but it can also provide for complying with legal requirements, for example certain privacy laws put requirements on the location of the server.

Disadvantages

Meanwhile, cited disadvantages (some perceived), include:

1. Robustness: Despite industry mythology – as recently demonstrated by Amazon - current mainstream Cloud implementations are no more robust than traditional well maintained Enterprise environments. (see http://blog.rightscale.com/2011/04/25/amazon-ec2-outage-summary-and-lessons-learned/) & http://www.theregister.co.uk/2011/04/21/amazon_web_services_outages_spans_zones/)

2. Latency: Network latency is limited by the speed of light; a hard limit. When substituting local computing resources with cloud solutions, the latency generally increases. Furthermore, there is also a tradeoff to be made regarding the placement of computing resources within the cloud. Co-locating instances in the same data center minimizes latency between cloud instances but increases risk that a failure through external influences affects all instances. However, such latency / locality trade-offs apply to all distributed environments, not just Cloud.

3. Throughput: Network throughput is most likely limited; both internal within the Cloud and connectivity to the Cloud. Again, this is not just an issue for Cloud, but any distributed environment as high performance non-blocking networks cost a premium.

4. Contention: Despite the marketing claims, Cloud resources are not infinite and compute contention does occur; see http://www.webmetrics.com/landingpage/bitcurrentcloud2/The_Performance_of_Clouds_Complete.pdf

5. Data Services. The classic relational database has problems with scaling 'horizontally' due to ACID behaviour and the potential volatility of underlying compute resources. To address this, some PaaS solutions offer scalable alternatives with different Consistency, Availability, Partitioning (CAP) trade-offs; i.e. Amazon's Simple DB. For traditional applications, however, this means that cloud data services cannot be considered drop-in replacements for relational database engines. In fact, many cloud data services essentially require applications to be developed around the data service and its consistency model and require pervasive changes to existing code.

6. Trust: A Cloud provider is independent of the cloud user and therefore cannot be completely trusted to act in the interest of the user. Cloud providers have full access to all the data and processing in the applications. Cloud applications must therefore consider privacy and security issues. Even though a cloud provider could be trusted, cloud providers must follow the jurisdiction that they are based in which might be different than the jurisdiction of the user. A specific example are the EU privacy laws that are much stricter than US laws.

7. Security: Applications are frequently co-located upon the same physical hardware. Isolation between applications is usually enforced by the virtual machine, virtual network and virtual storage infrastructure. For some customers these levels of isolation are not sufficient.

8. Remoteness: The remoteness of a deployed application requires some re-think with respect its management. For example, for scalable applications, how might one deal with the potential of run-away scaling; especially in the event that external management connectivity is unavailable?

9. Lockin: From a developer perspective a **major concern** is implicit “runtime” lock-in. As soon as PaaS specific services are used, the application becomes locked into that vendor's Cloud runtime. Even with regard to IaaS, the “inertia” of the data can lead to an implicit lock-in since the moving of large quantities of data between cloud providers leads to significant costs.

10. Complexity: Traditional datacenters attempt to maintain static unchanging environments, within which static applications are deployed once and run unchanged. The complexities that ensue through a failure of these assumptions, are born by the operational staff. In contrast Clouds provide a dynamic environments, where resource may come and go. Whilst this poses an immediate and significant challenge for traditional applications; e.g. distributed transaction / ACID based applications that assume a fixed / unchanging resource landscape, the resultant Cloud optimised replacements will from an operational perspective be much simpler to manage. For further comments of the issue of Clouds & Complexity see: http://www.slideshare.net/mfrancis/complexity-components-clouds-paremus

11. Application Architecture: 'Cloud' is not a magic cure-all. In-correctly architected applications cannot be scaled on demand. For this class of application a public Cloud Provider is simply a remote Utility compute provider that offers a minimal Service SLA.

Scalable Application Architectures?

Whilst Clouds *in principle* provide resource elasticity, applications must be architected in an appropriate fashion to make use of this opportunity. Not 'just' stateless compute; elastic applications may also require an elastic data-service layer, elastic group caching and elastic middleware messaging capabilities.

Looking at each of these in-turn:

1. **Data Services:** Traditional ACID transactional relational database was not designed to run across a large and possibly/probably volatile sets of compute resources. This, together with the realisation that not all data is relational in nature, has spawned a diverse eco-system of cloud centric, data-service layer solutions; these collectively referred to as NoSQL (Not Only SQL): examples include Hadoop, Voldermort, Cassandra & Amazon's Simple DB.

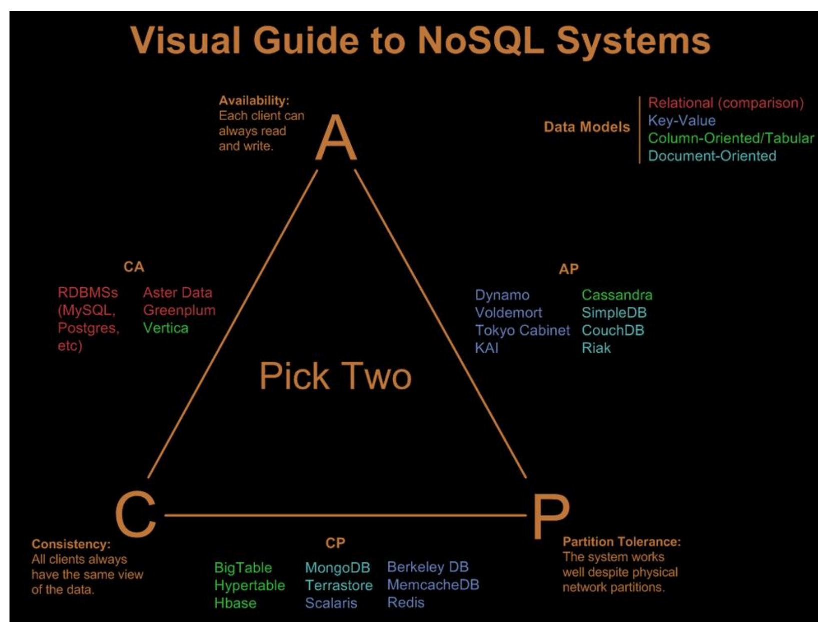


Figure 1: A Visual Guide to NoSQL Systems

(Source: http://blog.nahurst.com/visual-guide-to-nosql-systems)

   Rather than enforcing strict ACID behaviour these solutions make different Consistency / Availability trade-offs to cope with resource partitioning and various data-sharding and replication strategies to leverage the horizontal scaling capabilities of Cloud environments.

2. **Shared State Management:** Dependent upon the required functional behaviour of a *Process Group*, access to shared state between *Process Group* members may be required. A typical example may be sharing session state across a group of web servers. The business SLA / CAP trade-off should dictate the type of mechanism used to achieve this.

3. **Middleware – Coordination & Contention:**

   A broker based middleware service (e.g. JMS / AMQP)

   - Removes the concern about co-ordination from the clients; this by virtue of middleware service being a singleton, or less-frequently by sophisticated internal co-ordination mechanisms between members in a load-balancing cluster of brokers.

   - By providing durability, a message broker service allows publishers and consumers to be stateless. However, this is achieved at the cost of transferring the burden of data persistence and recovery to the brokers – so coupling these to the environment's underlying persistence mechanisms.

   The alternative, P2P broker-less messaging alternatives (i.e. JXTA, JGroups or OMG/DDS), provide a more naturally elastic / scale-out architecture. However the same issue of persistence and the coupling to the environment's underlying persistence mechanisms still exists.

Terminology & Concepts

Where terms are defined by National Institute of Standards and Technology (NIST), those same terms and interpretations will be used here. NIST defined terms include:

- Infrastructure as a Service (IaaS): The capability provided to the consumer is 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 (e.g., host firewalls).

- Platform as a Service (PaaS): The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly application hosting environment configurations.

- Software as a Service (SaaS): The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through a thin client interface such as a web browser (e.g., web-based email). The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.

- Private cloud: The cloud infrastructure is operated solely for an organization. It may be managed by the organization or a third party and may exist on premise or off premise.

- Public cloud: The cloud infrastructure is made available to the general public or a large industry group and is owned by an organization selling cloud services.

- Hybrid cloud: The cloud infrastructure is a composition of two or more clouds (private, community, or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load-balancing between clouds).

- Resource Pool/ing: The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter). Examples of resources include storage, processing, memory, network bandwidth, and virtual machines.

- Rapid Elasticity (cloud bursting): Capabilities can be rapidly and elastically provisioned, in some cases automatically, to quickly scale out and rapidly released to quickly scale in. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be purchased in any quantity at any time.

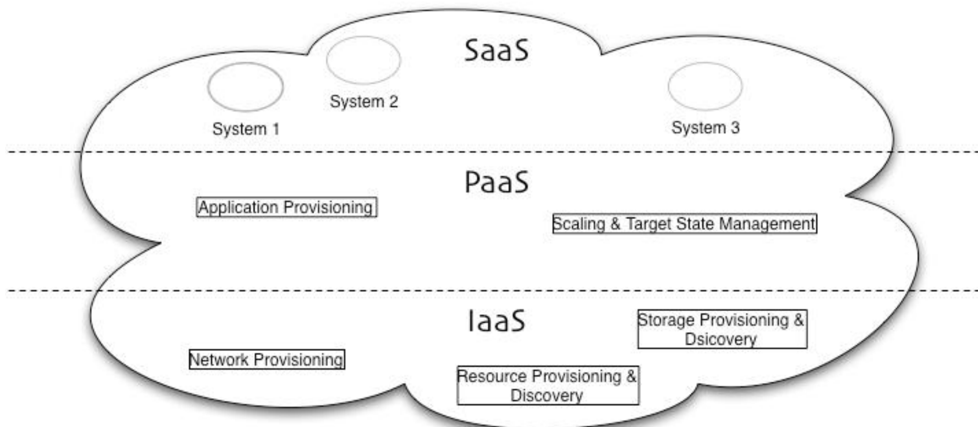


Figure 2: Cloud Morphology
(Shades of grey w.r.t what constitutes a IaaS or PaaS services)

The following terminology and structural hierarchy are also used extensively throughout this document.

- *Resource / Node*: A member or a resource pool – may be a physical or virtual machine.
- *Resource Provisioner*: Each *Resource Provisioner* is responsible for ongoing maintenance of the *Resource Pools* under its control:
  - Ensuring sufficient number of *Resources* in each *Resource Pool.*
  - Ensuring each *Resource Pool* population has the required characteristics.

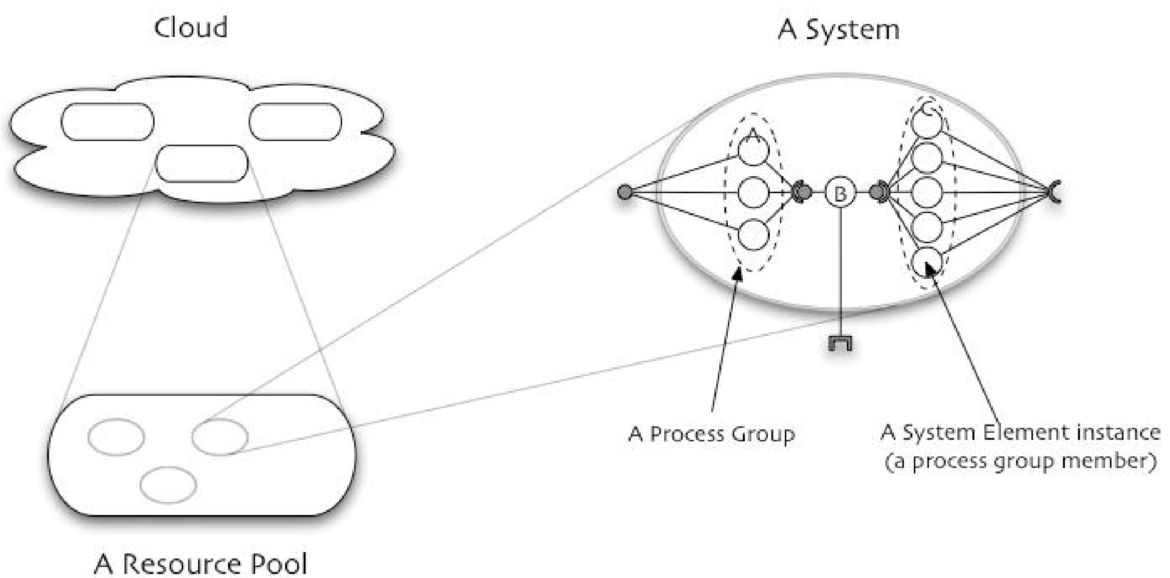


Figure 3: Structural Hierarchy

- *System*: The entity which is deployed to the runtime and subsequently managed. Alternatively referred to as an '*Assembly*' or a 'C*omposite Application*'.
- *System Description:* Describes a *System* in-terms of one or more '*System Elements*'. An instantiated *System* must comprise of the minimum number of instances of each non-optional *System Element*.

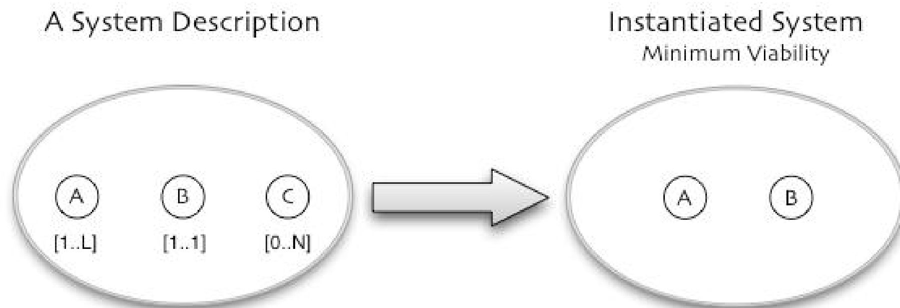


Figure 4: System: Model and Instantiated form
(System description comprises of 3 System Elements A,B,C. Minimum instantiated form is a System comprised of an instance of A & B)

System Elements referenced in a *System Description* may be backed by different underlying technologies; e.g. '*A'* may be an OSGi Blueprint based assembly, '*B'* a Java JEE artefact.

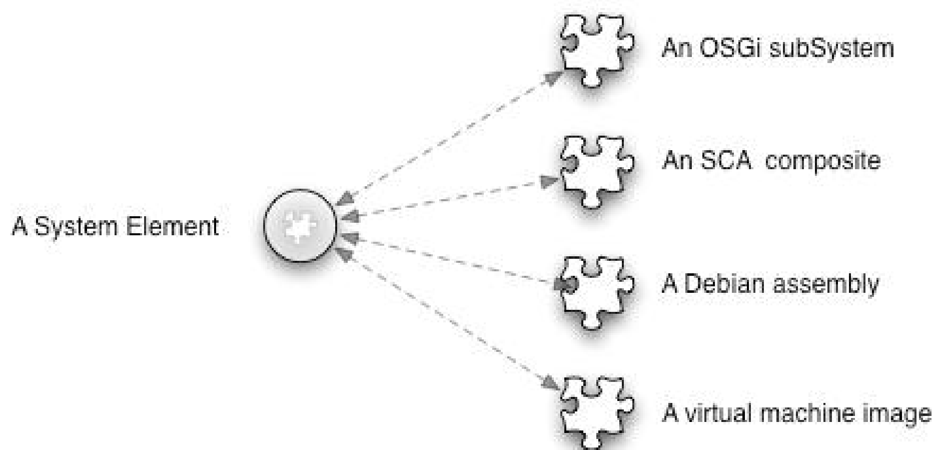


Figure 5: Possible Types of System Element

- *Process Group* (a.k.a 'Scale-Out' group): If a *System Element* is instantiated multiple times (e.g. N instances of System Element C), then the collection may be referred to as *Process Group*; e.g. *Process Group C*.

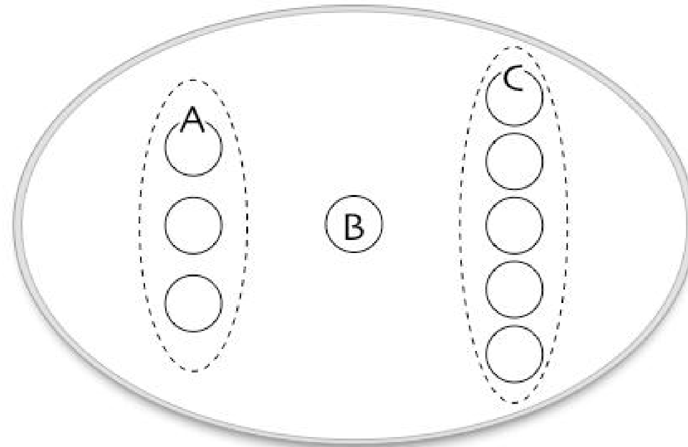


Figure 6: Process Groups

(In runtime multiple instances of System Element A & C may be created)

It is the **architecture** of the *System* which dictates whether a *System Element* is *optional*, *a singleton*, or may be instantiated multiple times so creating *a Process Group.* Dependant up System implementation; *Process Groups* may expand or contract across available resources in response to changing load, environmental conditions or manual management commands.

- *Resource Domain:* All the resources which, at each point in time, support one or more *System Element* instances of a running *System*. For example; for an OSGi based *System,* a *System's Resource Domain* would comprise of all resources hosting OSGi frameworks within which *OSGi assemblies* for the given *System* run.

- *Resource Topology*: At each point in time the network location of the resources that participate in a specified *Resource Domain*. For example; a local subnet *Resource Topology* would be needed for a *System* that requires minimal latency between *System Element instances.*

- *Hosting Policy:* Individual resources, or resource pools, may have hosting policies which control participation within the runtime environment.

  Example policies might include:

  - Only host named *System Elements* from named *Systems*.
  - The inverse – host any *System Element* other than those in named *System / System Element* list.

- *Resource Contract:* Metadata associated optionally associated with each *System Element* which dictates which type of resource should be considered for hosting instances of a *System Element.* A *resource contract* might reference:
    - Static Attributes – Characteristics Intrinsic to the resource (i.e. CPU, amount of physical memory), or acquired characteristics (i.e. rack location, cost centre)
    - Dynamic Attributes – Current CPU load, free memory, currently hosted *System Elements.*
    - Analysis of the resource contract should also include any metadata embedded within the *System Element:* i.e. in the case on an OSGi subSystem*,* requirements embedded in bundle manifests.
- *System Provisioner*: The System Provisioner (a PaaS service) is responsible for the ongoing mapping of *System Elements* to the underlying resource in a manner dictated by any associated *Resource Contract*. The System Provisioner is responsible for:
    - Deploying *System Elements* instances to nodes with correct static characteristics – e.g. to maintain the required *Resource Domain* characteristics and *Topology.*
    - Resource optimisation behaviours; including reacting to dynamic characteristics of a node, releasing *less-fit* resource in preference for resource which are more *fit-for-purpose*. The most extreme case being re-deployment on resource failure.
    - Ongoing expansion / contraction of *Process Groups* (i.e. and so the size of the *Resource Domain*) in response to
        - overall User load on System,
        - resource availability & relative System priority,
        - or external operational commands.
    - Ensuring physical co-location if *System Elements* have some form of '*affinity'* with each other. Or ensuring '*avoidance'* if the named *System Elements* should not be deployed to the same physical resource.

- *Services and References*: When instantiated, a *System Element* may export one or more *Services* and may have one or more *References*. *References* may be to internal Services, or to external Services: environmental / *PaaS Services,* public *Services* from other co-hosted *Systems,* or even *off-Cloud* Services.

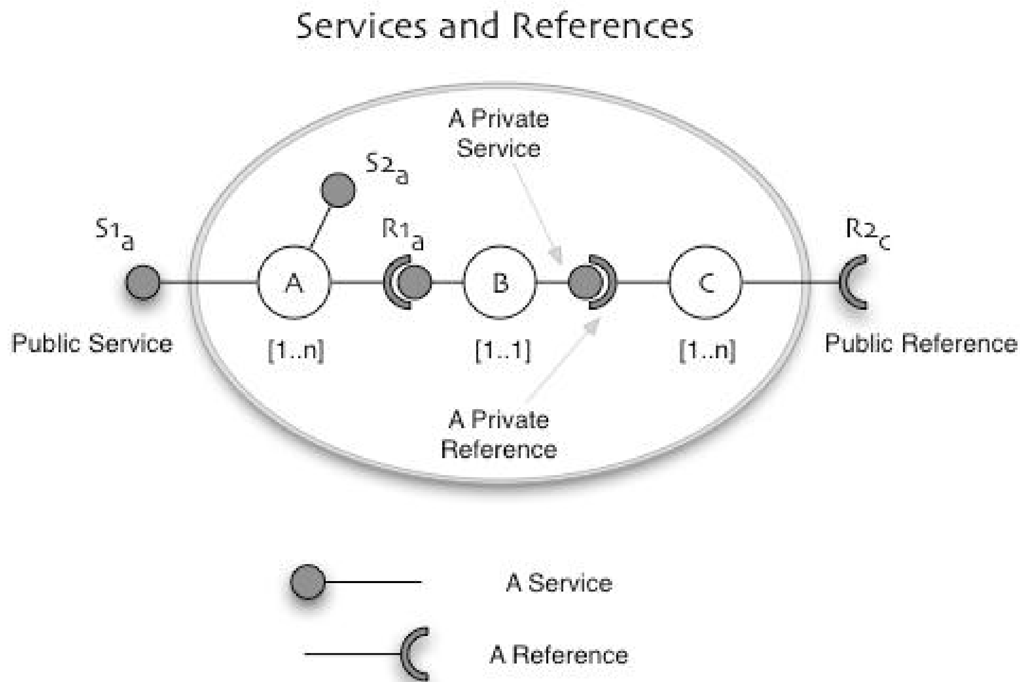


Figure 7: Service and References (Public and Private)

- *Service & Reference Scopes*:
    - *Private Services*: Service consumed by other *System Element* instances within the same System; either members of the same *Process Group*, or members of other *Process Groups* within the same *System.*
    - *Public Services*: A Public service may be consumed by an entity external to the System.
    - *References* may be *Private* or *Public: A Reference* may be to a third party *Service* co-hosted on the same Cloud, or a *Service* offered by the underlying Cloud runtime; e.g. in the above examples, a PaaS *Messaging Service* or an IaaS *Storage Service*.

  For OSGi based *System Elements* that leverage the *OSGi Remote Admin Specification (RSA) specification;* distributed *scopes* will be enforced by the *Topology Manager* and / or *Discovery Filters.* Meanwhile, local scopes within the JVM will be determined by the System Element's *subSystem type*; i.e. by whether the *System Element is* an *Application*, *Composite* or *Feature* subSystem.

- *Runtime Service Dependencies*: A particular type of environmental Service dependency.

  A 'middleware' service may be provided by an instance of a middleware *System Element.* If so, the '*Requirements*' of the selected middleware *System Element* must be consistent with the '*Capabilities*' of the *Storage Service* offered by the intended Cloud runtime (see Figure 7).

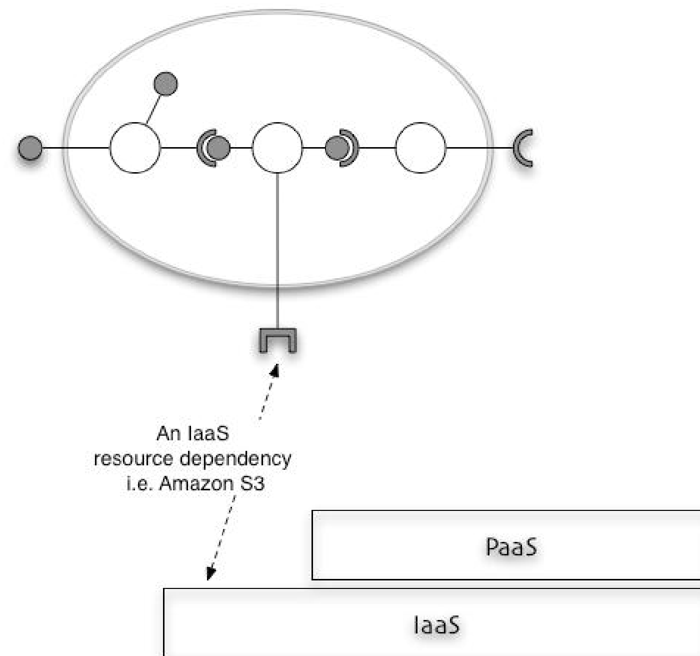


Figure 8: Middleware as an integral component of the System

Alternatively, the *System* may choose to have an external *Service* dependency on a PaaS provided middleware service. Here the *System* has a '*Requirement'* for a Cloud runtime which offers the required middleware Service '*Capability*' (see Figure 9).

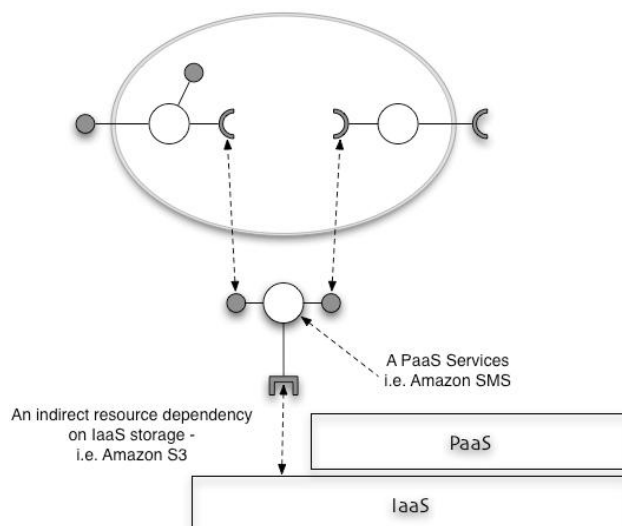


Figure 9: A dependency on a PaaS Service

Assumptions

1. The IaaS layer may / may not / be centered around hardware virtualization. Alternatives include physical machines and / or bare metal Java JVM's.

2. The environment is multi-tenant; i.e. multiple concurrent *Systems.*

3. Based on local hosting policy, individual resources may / may not / be multi-tenant; i.e. – able to host multiple System Element instances from the same or different Systems.

4. SaaS, PaaS & IaaS providers may be independent of each other.

5. Resource provisioning – addition / removal of resources – is on-going. The environment is dynamic, with new resources being deployed and existing resources failing or being withdrawn.

6. Resource and Service discovery is – as a result of the elasticity of the underlying platform – a continuous process.

7. System provisioning – deployment / re-deployment of *System Elements* – is on-going

8. Management of Systems, including configuration / re-configuration interactions with *Process Groups* needs to be asynchronous and idempotent.

9. Different Systems may have different runtime management constraints. One System may require a change of a property to be consistently applied to all *System Element* instances before its Services are again available (i.e. Consistency is selected over Availability). Meanwhile for a second System, a rolling upgrade of properties across all instances may be preferred (i.e. Availability selected over Consistency). Note that both Systems run within the same Cloud environment. (NOTE: This is just an example of the Consistency, Availability, Partitioning (CAP) trade-offs, which are usually only discussed w.r.t. No-SQL solutions).

10. A System may be comprised of a number of *System Elements* each of which may be backed by a different implementation:

    I. May / May Not be OSGi based.

    II. If OSGi based - May / May Not implement ConfigAdmin.

    III. If OSGi based - May / May Not support RS / RSA (Remote Service Admin)

4 Applicability of OSGi?

Relevance of OSGi to Cloud?

Modularity as in OSGi solves a major issue with architecting elastic applications for the cloud. As previously motivated, cloud resources are inherently dynamic in their nature and undergo frequent changes either due to explicit management operations (adding and removing resources) or due to their volatility (sharing effects, failures). Therefore, it is not a reasonable assumption that monolithic software incapable of dynamic adaptation can effectively run in such an environment. In fact, the compositional approach of modular software---traditionally applied to a single runtime system---is key to building scalable and dependable systems across a variable set of machines in the cloud. [3].

Software modules in their most generic form combine the following properties:

- Declaratively self-contained: A module is self-contained with regard to its own content and its declared dependencies.
- Encapsulated: a module exposes its content solely through well-defined interfaces. Interface here does not strictly mandate an interface of the programming language but can be a declarative interface in the module system that is not visible to the language.
- Decomposed: modules are created by segregating a larger problem into smaller subproblems so that a module ideally only deals with a single, not further separable concern and the content of the module is highly cohesive.
- Composable: modules are created for reuse in different applications and can be composed into new applications. Declared dependencies and declared interfaces put constraints on the validity of compositions. Ideally, the degree of coupling between modules is low so that composition is facilitated and not prohibited.
- Substitutable: two modules (or sets of modules) providing the same interfaces can be exchanged for one another.
- Localized behavior: modules are designed to behave locally, i.e., the effect of the code is restricted to the content of the module or its declared dependencies. A module should not make any assumptions about its dependents other than the ones expressed through the declared dependencies.

OSGi particularly facilitates composition and localized behavior by separating tightly-coupled and loosely-coupled interaction between modules into two different concepts: package dependencies and services. The latter are not only potential boundaries of distribution when composing applications running across many machines by introducing communication channels, the substitutability also permits the adaptation of the deployment at runtime, e.g., for load balancing or failover.

OSGi addresses the following operational & runtime challenges:

Sized just right:

It is envisaged that software artefacts would be available in one or more in-Cloud OSGi bundle repositories. As instances of *System Elements* are deployed to create a *System*; participating nodes would resolve the dependencies, and if not already cached locally, download required bundles from these OBR repositories. Only the required software components are downloaded and installed. This assembly process is asynchronous and completed across a large population of nodes in seconds.

Contrast this with deploying a opaque software blob (two to three orders of magnitude larger), consisting of an operating system and a traditional monolithic stack of software components.

Hence, relative to the current generation of imaged based Cloud solutions, it is suggested that OSGi enabled Clouds will place <u>significantly less stress</u> on the underlying IaaS infrastructure.

Rapid Re-deployment and Upgrades

Application upgrades and roll-back using virtual machine images typically take 10's of minutes. Not only does a completely new virtual machine image need to be deployed, but the required patched virtual machine image must first be created.

In contrast, to upgrade a running System in an OSGi enabled Cloud, one would release updated bundles to the appropriate in-Cloud bundle repositories and load the update System description. Affected *System Elements would* re-deployed to affect nodes by the System provisioner, the nodes re-resolving and downloading the updated bundles from the *in-Cloud* bundle repositories. <u>Via such mechanisms application upgrades and roll-backs are achieved in seconds.</u>

From here it is short step to *reactive runtimes* where System's are able to modify themselves in response to pre-defined environmental changes, user behaviours or load.

Avoiding Runtime Lock-In:

As a *System* is self-describing, one might *validated* it against potential OSGi Cloud runtimes before it is deployed.

- *Artefact Dependencies:* Are the required artefacts for my *System* in the in-Cloud bundle repositories? If not, load missing artefacts.

- *Service Dependencies:* Can the target runtime meet the System's *public Service dependencies (References).* If not, these might need to be provided by the System itself; i.e. as an internal private Service implementation.

- *Resource Dependencies:* Are the required types of resource available? As a result of running the *System* what data has been persisted to the Cloud?

By understanding these forms of **<u>runtime dependency</u>** one can more easily manage / avoid Cloud runtime lock-in.

Dependencies, dependencies, dependencies:

All the above boil down to dependency management.

If module and context / environmental dependencies are fully addressed; (see http://www.kirkk.com/modularity/2009/12/chapter-6-realizing-reuse/) then *re-use* and *ease-of-use* can be achieved and application maintenance costs radically reduced.

Existing OSGi Standards

The following OSGi standards and EEG activities are relevant to Cloud.

- RFC 152 – Subsystems: It is assumed that an OSGi *System Element* would correspond to an OSGi RFC-152 subSystem.
- RFC 112 – OBR: Bundle Repository
- RFC 154 – Bundle Requirements & Capabilities
- Remote Service Admin – See R4.2 Enterprise Spec – Only relevant if the *System Element* is an OSGi subSystem.
- Configuration Admin – See R4.2 Compendium – Only relevant for *System Element*s that implement OSGi Config Admin.

5 Problem Description

Scale & Volatility

To meet the challenges of scale and resource volatility, solutions must be:

- Asynchronous: Specifically monitoring, management, provisioning & configuration. needs to happen outside of the critical path of running applications to not adversely affect their performance and scalability.
- Idempotent: A management interaction can be re-issued several times if necessary, and be ignored by parties that have already enacted the request.
- Scale-Free: Services must scale and partition along well defined boundaries as the size of the runtime environment increases from 1 to 10^6 nodes (see http://www.theregister.co.uk/2010/07/19/nasa_rackspace_openstack/).

Isolation

It is assumed that isolation for OSGi *System-Elements* co-located within the same multi-JVM/OSGi framework will be provided by the OSGi subsystem mechanism.

However System level isolation is also required. Specifically, the internal constituents of "System A" should not have visibility of internal / private services, events or message flows between instances of System Elements in System B.

Configuration

- Service scopes need to be appropriately configured; i.e. a service may be a fine-grained service; only of concern to other dependent *System Element* instances within the same *System.* Or may-be a coarse grained Business Services provided by a System to external entities?

  NOTE: A coherent approach to service scoping is required which spans from Services from different subSystems but co-located within the same JVM through to remote services on the network. Hence Service scope configuration involves both the subSystem and RSA OSGi standards.

- Not all *System Elements* may be OSGi ConfigAdmin compliant. How are such artefacts configured in an OSGi centric cloud environment? Or aren't they support?
- *System Elements* may have configurations that are interlinked; i.e. the configuration of a deployed System-Element instance dependent upon the IP address and port dynamically allocated to another dynamically deployed component.
- *System Elements* may need be be configured / re-configured / w.r.t. their local runtime environment.
- Ongoing configuration behaviour (consistency v.s. availability trade-offs) should be a *System / SLA* concern – not restricted / dictated / by the underlying PaaS.

Resource Capabilities

Virtual machine image centric solutions present a homogenous resource landscape; the OS / Application typically deployed to a “small” / “medium” / “large” virtual machine. Also, as long as one is in the correct geographic region, data may be accessed from any node.

It is however unlikely that enterprise users will want to *homogenate* naturally *heterogeneous* data-centre resources in such a fashion. This is wasteful of both network bandwidth and CPU processing power. Rather, the enterprise will want to, *as simply as possibly*, place applications on the most suitable physical hardware and as close to the physical location of the data as possible. This requirement does not always agree with the transparency created by virtualization. Similar arguments apply as once made against the degree of virtualization applied by traditional operating systems to large-scale services [4].

For this reason a System provisioner needs to be able to query the physical reality of the IaaS platform for current resource information; including number of resources with capabilities that match a System's requirements; including current locations of persisted data upon which the System is dependant.

Ideally both local and public Cloud resource information should be presented in a normalised fashion: Iaas / PaaS services aggregating the resource characteristics of the members in each resource pool creating a summary or characteristics of the '*population*'.

However, there is no consensus on how such resources are described. Hence how can characteristics derived from each Cloud vendor be mapped to subSystem/bundle requirements? It is suggested that mechanisms (a set of resource adapters) are required to map key/value resource strings from each supported Cloud environment for comparison against RFC 154 requirements.

Resource Discovery

Each IaaS platform requires some form of manual or automatic mechanism that enables new compute or storage resource to be discovered and assimilated. However, these resource discovery and assimilation mechanisms are assumed to be out of scope of this RFP.

However, a generic mechanism via which a local Service can discover previously persisted state is required. This might take the form of an OSGi storage service proxy which abstracts the implementation details of specific host PaaS/IaaS; i.e. the details of Amazon S3, Rackspace BlobStore and SNIA storage discovery mechanisms.

Resource Provisioning

A resource provisioner is responsible for maintaining the appropriate population for a specified resource pool:

- The size of population may vary of time
- The population characteristics may related to physical resource types, or metadata concerning organisation or location.

Discoverable resources may be:

- A physical machine with a traditional OS with some form of advertisement / discovery protocol.
- A hypervisor service with some form of advertisement / discovery protocol.
- A bare metal JVM with some form of advertisement / discovery protocol.

A resource must not only advertise its presence, but also appropriate capabilities metadata. Only trusted resources are allowed to join a resource pool.

It is assumed that these resource provisioning is out of scope of this RFP. However as discussed in the previous section one needs to be able to map between the advertised resource capabilities in each resource pool and System requirements.

Node Hosting Behaviours

Nodes are not passive entities but are required to interact with their environment. Potential behaviours include:

- Advertising their static and dynamic capabilities and currently installed *System Elements.*
- Ability to narrow availability to a named *System* or *System Element*.

Starting an OSGi Framework

Some consideration is needed with respect to bootstrapping the OSGi framework on each participating node.

When a resource joins a resource pool either:

- A pre-installed OS / JVM / OSGi framework image is started (virtual machine image) approach, or a new node is created in a copy-on-write fashion from an existing running node.
- An OSGi framework, and potentially JVM, are dynamically loaded.

If the former, it is the cloned image of an OSGi framework which is started. RFC 151 states that an OSGi framework UUID is reset each time the framework is initialised. Assuming quiesced virtual machine images are distributed (**<u>NOT</u>** a snapshot of a running OSGi framework), then each instances upon startup should initialise the framework instance. Also, if a UUID is unset both Remote Service Admin implementations will set a unique UUID for advertised services.

If the OSGi framework was created from a snapshot or as a clone of running OSGi assemblies, it has to be ensured that the UUID is updated accordingly so that its uniqueness across a cloud deployment is not violated.

Service Discovery

A service discovery mechanisms are need for the following:

- To discover services provided by the host PaaS upon which the System is dependant.
- To discover internal system services, exposed by one group of system element instances, consumed by other parts of the same system.

The OSGi RSA standard provides the basis for a flexible cloud discovery solution; allowing custom discovery mechanisms for host PaaS environment for *public* Services and References, and potentially separate discovery mechanism for a System's own internal *private* Services.

System Provisioning

A System provisioner is responsible for:

- The provisioning of the initial System across appropriate / available resources in a resource pool. Resource capabilities that the System provisioner may need to consider include:
    - Persistence – Access to previously persisted state.
    - Capabilities – Correct physical (OS type, CPU type) and/or organisation/location characteristics (data-centre location, rack location).
    - Affinity – *System Element* should be deployed to a resource that is already hosting a specified *System Element; i.e.* physical colocation is required.
    - Aversion – To be considered as a suitable host - a resource should not be already hosting a specified *System Element; i.e.* colocation is to be avoided.
- Subsequently increase or decrease the size of Process Groups to ensure System meets its SLA.
- Re-deploying members of Process Groups (instances of System Elements) in response to either:
    - Resource failure, or
    - Discovery of a resource with greater 'fitness'; this defined by SLA.

The System provisioning / bundle resolver process must ensure that all element instances are assembled in an *identical fashion*; specifically that transitive dependences met in the same manner across all instances; this avoiding the need to clone virtual machine instances. Furthermore, it needs to support a mechanism for applying bundle life-cycle operations consistently across multiple framework instances where required.

System Validation

System validation is clearly related to System provisioning. However one may wish to first validate whether a target runtime is actually capable of supporting the System, before releasing the System into the runtime whereupon the System Provisioner instantiates it.

Specifically:

- Bundle Dependencies: Does the target Cloud environment have the required artefacts in its OBR repository to satisfy transitive dependencies?
- Service Dependencies: Are the public Services referenced by the System provided by the PaaS, or other Systems currently running upon it?
- Resource Dependencies: Does the resource population have the required *capabilities* to meet the systems *environmental requirements*?
- State Dependencies: If persisted state is required by a *System Element*, is this actually available?

Conversely; knowing these **environmental dependencies**, provides the cloud user with information concerning the degree to which a system is coupled to any particular Cloud runtime.

Monitoring

Monitoring is fundamental consideration for any automated / reactive environment. Monitoring should be scoped / aggregated / with respect to the structural entities; i.e.

- Each *System Element* instances will have access to monitor information concerning its own local environment.
- Monitoring metrics need to be aggregated at *Process Group* and *System* levels.
- Monitoring metrics also need to be aggregated for each *Resource Pool.*
- System Provisioner will be a consumer of aggregated *Process Group / System* metrics
- Resource pool manager will also be a consumer of aggregated *Resource Pool* metrics.

Logging & Audit

Logging and audit must preserve runtime context .i.e. Which *instance* of which *System Element* of which *System* threw the particular exception. At the time of the exception, which node was acting as host? At the time of the exception, were any other artifacts co-hosted?

Adaptive local behaviours also require access to appropriate logging; i.e. a node may be required to watch installed assembly for certain types of exception, upon which locally behaviours may be triggered:

- destroys the local environment. Local '*Crash Only*' behaviour – global recovery (see – http://dslab.epfl.ch/pubs/crashonly/).
- Or cause a forced stop and restart – local '*Recovery Only Computing*' (ROC) behaviour (see – http://www.armandofox.com/geek/past-projects/recovery-oriented-computing-roc/).

Post Mortem/Debugging

It will be necessary to provide tools to debug a large could systems. The following services can be foreseen:

1. Snapshot of a OSGi framework instance including installed artefacts.
2. Automatic snapshotting triggered by certain error conditions.

6 Use Cases

Purpose

The following Use Cases highlight the potential of an OSGi value proposition, both from the perspective of customer and service provider, above and beyond what is presently achievable with current Cloud solutions.

Use Case - The Customer

Fast Financial Services (FFS)' – is a small boutique financial services organisation that develop and use their own in house systems.

Having a small team of business focused developers – and needing to frequently re-factor their applications for different customer requirements and trading strategies – FFS decided to build a modular OSGi based system. The first iteration consisting of several hand crafted nodes in a traditional cloud.

Business is good and FFS need to determine how to scale the processing power per customer, and concurrently support multiple customers – all whilst maintaining agility and availability. Rolling VM images for each functional change becomes an major agility impediment.

Use Case - The Cloud Provider

'***Composite Cloud Corp***' (CCC) are a next generation Public Cloud provider. In addition to providing an IaaS, CCC also offer an OSGi enabled PaaS and a libraries service for premium customers; the library service comprises of managed OBR repositories with certified open-source and commercial OSGi components as well as a private OBR repository for each customer.

FFS are attracted by CCC service offering. Not only does this provide flexible / scalable runtime, BUT the cloud provider also maintains certified infrastructure components that FFS require, but have no wish to manage. FFS refractor their Systems to leverage CCC managed repositories and load their proprietary business components into their private OBR cloud repository along with the relevant System.

CCC PaaS layer deploys and manages FFS composite system – resolving and loading all required components from the relevant OBR repositories.

Use Case - Agility

One of FFS key differentiators is the speed at which they can customise their services for

clients – either custom updates for existing users or meet the requirements of new customer.

FFS systems comprise of a number of customisable functional units, some of which are required to horizontally scale in response to Market volatility. With a traditional Cloud vendor FFS would need to generate new VM images for each required change – including all unchanged software – and re-release the entire System to the Cloud. A somewhat laborious process when changes occur every day.

Being OSGi based – FFS simply develop the required customisations / updates, load the updated bundles into their private Cloud OBR repository and then either instantiate a new System, or roll forwards an existing running System. The PaaS platform with advanced provisioning and OBR resolver capabilities does the rest.

In addition to fast upgrades - roll back to a previous “good” System is also near instantaneous.

Use Case – LockIn

FFS is successful and as a result acquired by a large financial Services organisation – 'Big Bank'. Initially Big Bank have excess data centre space and decide to host FFS Systems in house.

Because FFS Systems are self-describing composite Systems, with software artefact , service and environmental dependencies defined, migration is relatively simple. Big Bank load all required infrastructure software components into their own OBR repository and deploy the FFS business Systems on an internal OSGi enabled private Cloud.

Use Case – Hybrid

In due course the FFS division of Big Bank are so successful that significant “burst” capability is required to cope with volatile demand. As the System Elements in each System explicitly state environmental and service dependencies it is simple to identify the stateless asynchronous elements that are idea for hosting on a third party Cloud such as that initially provided by 'CCC'.

As the FFS system dependencies can be simply validated against each external Cloud vendors PaaS platform – the FFS team can calculate effort v.s. cost associated with using competitive Cloud vendor services. The agility and flexibility provided by CCC OSGi PaaS platform and managed OBR repositories wins the day and FFS reselect CCC. Further systems from Big Bank big migration to OSGi so of which – including Dev and UAT being hosted by CCC.

Mixing private and public clouds

A medium-sized enterprise is using a private cloud to provide its IT services such as the company website, forums and wikis, developer resources such as SCM repositories and other systems such as HR and Finance support systems.

Because of a power failure part of the internal cloud becomes temporarily unavailable. The company doesn't want its employees to be hindered by this technical issue and quickly spins up a few images providing the affected systems in a public cloud so that its employees can continue to work productively.

Development

A company develops Facebook applications which are deployed in a public cloud. The applications tend to be very popular so the company has selected a public cloud provider that provides great performance, bandwidth and latency.

However the cloud provider is on the pricey side so the company doesn't want to deploy Facebook applications to it that are still in development. They are looking for a way to deploy these applications locally on the developer's machine under an environment that matches the public cloud closely.

7 Requirements

Management

- MAN0001 – The solution SHOULD define APIs to interact with a variety of different cloud platforms to manage the resource pool by, e.g., adding new nodes or removing nodes, where nodes represent OSGi Frameworks.
- MAN0002 – The solution SHOULD define APIs that allow querying of deployed Systems (Composite Applications), System Elements (Subsystems), Bundles and Services on an OSGi Framework in the Cloud.
- MAN0003 – The deployment of bundles SHOULD facilitate consistent behaviour across multiple framework instances. It SHOULD be possible to receive feedback on the deployment status.
- MAN0004 – The solution MUST provide APIs to discover available OSGi Frameworks in a Resource Domain.

Metadata

- MD0001 – The solution MUST define APIs that allow querying of capabilities and other metadata of OSGi Frameworks in the Cloud. This information SHOULD at least include the following:
    - Framework GUID
- MD0002 – An OSGi bundle MUST be able to add proprietary capabilities to the metadata exposed by its OSGi Framework in the Cloud.
- MD0003 – The solution MUST provide information about the environment, system, and the capabilities and properties of the platform underlying an OSGi Framework in the Cloud. This information SHOULD include at least the following static capabilities:
    - location
    - IP address
    - cpu architecture
    - cpu capacity
    - Total memory

And the following dynamic capabilities

  - cpu load factor
  - Available Memory

- MD0004 – The solution MUST allow provider-specific capabilities to be added regarding the underlying platform.
- MD0005 – The solution SHOULD define APIs for the application or a sysadmin to assess the current costs of an existing deployment and estimate the potential cost of an alternative deployment.
- MD0007 – The solution MUST allow querying the available OSGi Frameworks in a Cloud Domain based on the metadata and capabilities these OSGi Frameworks expose.

Event notifications

- EN0001 – The solution SHOULD define a notification system which must be compatible with the EventAdmin framework. Notification events should include container up / container down and container environment/capability changes.
- EN0002 – The event notifications SHOULD also include remote events for bundle started/stopped, service registered/unregistered, etc. This should be extensible by the PaaS provider and also by the PaaS end/user (developer) and their bundles, such that any of those are able to define events.

Instances

- INS0001 – The solution MUST NOT violate the uniqueness of the RFC 151 framework UUIDs of the corresponding OSGi framework instances.

Various

- VAR0001 - Solution MUST allow running an application in separate scopes as defined RFC 152.
- VAR0002 – Migrating OSGi bundles and application to the solution SHOULD be non-intrusive.
- VAR0003 – It MUST be possible to run OSGi bundles and applications developed for the solution outside of a Cloud environment.
- VAR0004 – The solution MUST NOT prevent running multiple OSGi Frameworks, each running in its own JVM, on a single node.

8 Deferred Requirements

Requirements listed in this section will not be the primary focus of the initial design. They may be addressed as the need arises or are otherwise addressed in subsequent revisions of the technical design.

- SVC0005 – The solution should define service APIs that encapsulate cloud-provided file storage.
- SVC0006 – The solution should define service APIs that encapsulate cloud-provided NoSQL type database storage.
- SVC0007 – The solution should define service APIs that encapsulate cloud-provided data grids.
- SVC0008 – The solution should define service APIs that encapsulate cloud-provided messaging systems.
- INS0003 – The solution should allow the saving and restoring of the system bundle and user bundles to instantiate them in another frameworks. Makes it easy to migrate/fork.
- INS0004 – The system should provide a broker service that knows about latency, cost, location, etc. - A complex large scale system will require making many trade offs, requiring a broker abstraction

9 Appendices

Cloud Providers

There are a number of active cloud providers in the market.

Amazon

Amazon is likely the most active and innovative provider in the cloud market. With Amazon Web Services they provide a suite of tools that address different aspects of Cloud based computing:

1. EC2 – Elastic Compute Cloud is a model where instances of pre-specified images can be created on the fly.

2. S3 – Simple Storage Service. A highly scalable storage service

3. EBS – Elastic Block Storage.

4. RDS – Relational Storage. An instance of MySQL

5. SQS – A highly scalable queuing service

6. SNS – A notification service

7. Simple DB – A simple but highly scalable database

8. Cloudwatch – A monitoring service

Google Apps

Google Apps provide a much more limited, web based, model of cloud computing with their App Engine. Their model is based on a very restrictive sandbox accessible from the web. Threads are for example not supported.

Rackspace

Rackspace provides www.rackspacecloud.com. Rackspace provides similar functions to Amazon, it is possible to programmatically launch new instances based on pre-defined images.

1. Cloud Servers – Create/Delete instances

2. Cloud Sites – Run your code on a Rackspace managed cluster

3. Cloud Files – Storage service

Microsoft Windows Azure

Microsoft provides a Window server based cloud platform with the following functions:

1. Windows Azure Platform – Windows Azure™ is a cloud services operating system that serves as the development, service hosting and service management environment for the Windows Azure platform. Windows Azure provides developers with on-demand compute and storage to host, scale, and manage web applications on the internet through Microsoft® datacenters.

2. SQL Azure – A cloud-based relational database service built on SQL Server® technologies. It provides a highly available, scalable, multi-tenant database service hosted by Microsoft in the cloud.

3. AppFabric - Windows Azure platform AppFabric helps developers connect applications and services in the cloud or on-premises. This includes applications running on Windows Azure, Windows Server and a number of other platforms including Java, Ruby, PHP and others. It provides a Service Bus for connectivity across network and organizational boundaries, and Access Control for federated authorization as a service.

IBM

IBM provides a Cloud computing environment but it is not clear if this has an API or is provided as a general computing service. http://www.ibm.com/ibm/cloud/smart_business/

Private Cloud Solutions

VM based (IaaS) open-source solutions including Eucalyptus (http://www.eucalyptus.com/) & Open Nebulae (http://www.opennebula.org/about:about). Meanwhile, the Paremus Service Fabric is an example of a reactive OSGi based (PaaS) solution.

Red Hat

JBoss Cloud Foundations (http://www.redhat.com/solutions/cloud/) provides Cloud Computing management independent of the underlying IaaS layer. This is realized through the cloud independent REST APIs provided by Apache DeltaCloud (http://incubator.apache.org/deltacloud/). Red Hat provides users with graphical tools that allow them to create virtual machines independent of cloud provider and to move virtual machines from one cloud to another, for instance from a private cloud to a EC2. In addition, the tooling supports the specification and application of SLAs and Policies independent of the underlying cloud infrastructure.

Red Hat Platform-as-a-Service (PaaS) is a solution that can be deployed or offered through a public or private cloud to build, deploy, and manage applications across their lifecycle. Based on JBoss Enterprise Middleware and other Red Hat technologies, Red Hat PaaS enables Cloud and SaaS providers, Global 5000 enterprises, and Government organizations to develop applications, seamlessly integrate with existing heterogeneous environments, and deploy them to a cloud infrastructure more rapidly, with greater flexibility, and at a lower cost than traditional application development and deployment paradigms. (http://www.jboss.com/solutions/PaaS/).

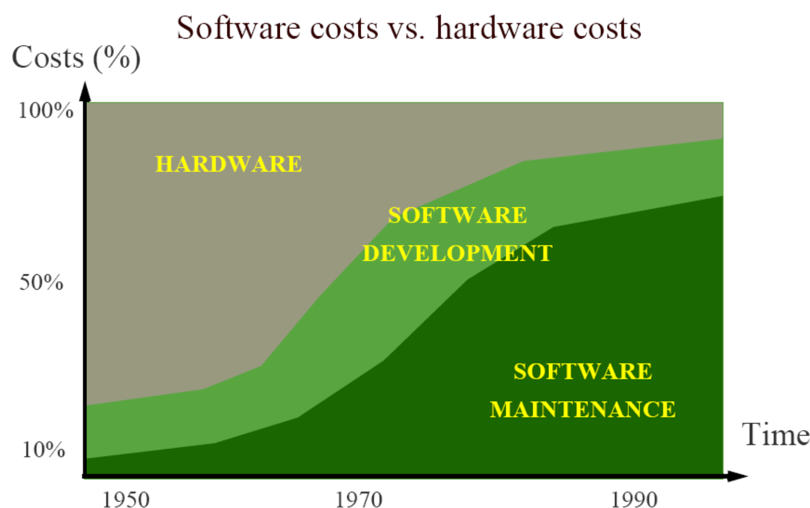
Current best practices of virtual-machine-based management and deployment

To deploy an application one first creates a set of virtual machine images loaded

with all relevant software artefacts.

- In order to reduce the number of virtual machine images that need managing,vendors recommend that software is consolidated as far as possible; see summary in '*Building a Dynamic Infrastructure with IBM Power Systems: A Close Look at Private Cloud TCO*' - http://www.computerweekly.com/DowntimePDF/pdf/IBMEZWhitepaper.pdf.
- To apply subsequent changes one must re-cut a new virtual image with the required changes, un-deploy the current image and deploy the newly modified version.
- If the patched virtual machine image is responsible for hosting many services – then many runtime services are impacted by this minor change.
- However, if one attempts to have one service per virtual machine image – service interruption is reduced (only the changed service is affected), but at the expense of virtual machine sprawl, increased management costs and runtime complexity.
- In both cases instead of a small incremental change, the complete software stack must be redeployed across the cloud environment; consuming unnecessary bandwidth: deployment times of 30 minutes considered rapid.
- Finally with a virtual machine based deployment model, Operations tend to be concerned with managing virtual machine images their deployment and updates, and not the software elements of the business systems. The mapping between these virtual machines and the business systems they represents needs to be reconstituted by the management infrastructure.

Compute Cloud vendors seek to optimize resource utilisation. Virtual machine based *Compute Cloud* solutions have nothing to say about the internal structure of the hosted applications and this is seen as a positive advantage!. Hence application maintainability issues remain unaddressed and are simply carry forwards into the Cloud environment and quite probably compounded.



10 Document Support

References

[1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.

[2]. Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0

[3]. J.S. Rellermeyer, M. Duller, and G. Alonso: Engineering the Cloud from Software Modules. In: ICSE-Cloud 2009

[4]. M. Welsh and D. Culler: Virtualization Considered Harmful: OS Design Directions for Well-Conditioned Services. In: HotOS 2001

Author’s Address

Name	Peter Kriens
Company	aQute SARL
Address	9c, Avenue St. Drezery
Voice	33467542167
e-mail	Peter.Kriens@aQute.biz

Name	Richard Nicholson
Company	Paremus
Address	107/111 Fleet Street London
Voice	+44 207 936 9098
e-mail	Richard.nicholson@paremus.com

Name	Mark Little
Company	Red Hat
Address	

Voice	
e-mail	mlittle@redhat.com

Name	David Bosschaert
Company	Red Hat
Address	6700 Cork Airport Business Park Kinsale Road Cork Ireland
Voice	+353 21 230 3400
e-mail	david@redhat.com

Name	Jan S. Rellermeyer
Address	ETH Zuerich Department of Computer Science CAB E 78 Universitaetstrasse 6 8092 Zuerich, Switzerland
Voice	+41 44 632 30 38
e-mail	rellermeyer@inf.ethz.ch

End of Document