Cloud Application Management for Platforms Version 1.1

Committee Specification Draft 01

09 January 2013

Specification URIs

This version:

http://docs.oasis-open.org/camp/camp-spec/v1.1/csd01/camp-spec-v1.1-csd01.pdf (Authoritative)
http://docs.oasis-open.org/camp/camp-spec/v1.1/csd01/camp-spec-v1.1-csd01.html
http://docs.oasis-open.org/camp/camp-spec/v1.1/csd01/camp-spec-v1.1-csd01.doc

Previous version:

N/A

Latest version:

http://docs.oasis-open.org/camp/camp-spec/v1.1/camp-spec-v1.1.pdf (Authoritative)
http://docs.oasis-open.org/camp/camp-spec/v1.1/camp-spec-v1.1.html
http://docs.oasis-open.org/camp/camp-spec/v1.1/camp-spec-v1.1.doc

Technical Committee:

OASIS Cloud Application Management for Platforms (CAMP) TC

Chair:

Martin Chapman (martin.chapman@oracle.com), Oracle

Editors:

Jacques Durand (jdurand@us.fujitsu.com), Fujitsu Limited
Anish Karmarkar (Anish.Karmarkar@oracle.com), Oracle
Adrian Otto (adrian.otto@rackspace.com), Rackspace Hosting, Inc.

Related work:

This specification replaces or supersedes:

- Cloud Application Management for Platforms, Version 1.0, August 29, 2012. http://cloudspecs.org/CAMP/CAMP_v1-0.pdf

Abstract:

This document defines the artifacts and APIs that need to be offered by a Platform as a Service (PaaS) cloud to manage the building, running, administration, monitoring and patching of applications in the cloud. Its purpose is to enable interoperability among self-service interfaces to PaaS clouds by defining artifacts and formats that can be used with any conforming cloud and enable independent vendors to create tools and services that interact with any conforming cloud using the defined interfaces. Cloud vendors can use these interfaces to develop new PaaS offerings that will interact with independently developed tools and components.

Status:

This document was last revised or approved by the OASIS Cloud Application Management for Platforms (CAMP) TC on the above date. The level of approval is also listed above. Check the “Latest version” location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee’s email list. Others should send comments to the Technical Committee by using the

"Send A Comment" button on the Technical Committee's web page at http://www.oasis-open.org/committees/camp/.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (http://www.oasis-open.org/committees/camp/ipr.php).

Citation format:

When referencing this specification the following citation format should be used:

[CAMP-v1.1]

Cloud Application Management for Platforms Version 1.1. 09 January 2013. OASIS Committee Specification Draft 01. http://docs.oasis-open.org/camp/camp-spec/v1.1/csd01/camp-spec-v1.1-csd01.html.

Notices

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The name "OASIS" is a trademark of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see http://www.oasis-open.org/policies-guidelines/trademark for above guidance.

Table of Contents

1 Introduction

1.1 Overview

Platform as a Service (PaaS) is a term that refers to a type of cloud computing in which the service provider offers customers/consumers access to one or more instances of a running application computing platform or application service stack. NIST defines PaaS [SP800-145] as a "service model" with the following characteristics:

> 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.

There are multiple commercial PaaS offerings in existence using languages such as Java, Python and Ruby and frameworks such as Spring and Rails. Although these offerings differ in such aspects as programming languages, application frameworks, etc., there are inherent similarities in the way they manage the lifecycle of the applications that are targeted for, and deployed upon them. *The core proposition of this specification is that these similarities can be leveraged to produce a generic application and platform management API that is language, framework, and platform neutral.*

For PaaS consumers this management API would have the following benefits:

- "Portability between clouds" is emerging as one of the primary concerns of cloud computing. By standardizing the management API for the use cases around deploying, stopping, starting, and updating applications, this specification increases consumers' ability to port their applications between PaaS offerings.
- It is likely that implementations of this specification will appear as plugins for application development environments (ADEs) and application management systems. Past experience has shown that, over time, such generic implementations are likely to receive more attention and be of higher quality than the implementations written for solitary, proprietary application management interfaces.

For PaaS providers this management API would have the following benefits:

- Because the strength and features of a PaaS offering's application management API are unlikely to be perceived as key differentiators from other PaaS offerings, the existence of a consensus management API allows providers to leverage the experience and insight of the specification's contributors and invest their design resources in other, more valuable areas.
- By increasing the portability of applications between PaaS offerings, this management API helps "grow the pie" of the PaaS marketplace by addressing one of the key pain points for PaaS consumers.

1.2 Purpose

This document defines the artifacts and APIs that need to be offered by a Platform as a Service (PaaS) cloud to manage the building, running, administration, monitoring and patching of applications in the cloud. Its purpose is to enable interoperability among self-service interfaces to PaaS clouds by defining artifacts and formats that can be used with any conforming cloud and enable independent vendors to create tools and services that interact with any conforming cloud using the defined interfaces. Cloud vendors can use these interfaces to develop new PaaS offerings that will interact with independently developed tools and components.

The following is a non-exhaustive list of the use cases which are supported by this specification. Details of these use cases can be found in Appendix C.

- Building and packaging an application in a local Application Development Environment (ADE)

- Building an application in an ADE running in the cloud
- Importing a Platform Deployment Package into the cloud
- Uploading application artifacts into the cloud
- Run, stop, suspend, snapshot, and patch an application

1.3 Example (Non-normative)

This example illustrates a scenario in which the application administrator wants to run and monitor an application. It assumes that the application package was previously made available to the platform, either because it was uploaded to the platform or developed directly on the platform.

The administrator starts by deploying the application package to the platform. This is done by sending an HTTP POST request to the platform entry point URL as shown below, where "/myPaaS" is the entry point and "/myPaas/pkgs/1" is the location of the application package.

```
POST /myPaaS HTTP/1.1
Host: example.org
Content-Type: application/json
Content-Length: ...

{
  "pdp_uri": "/myPaaS/pkgs/1"
}
```

On receiving such a request the platform deploys the application package and creates a new resource "/myPaas/templates/1" that represents the deployed application. The response from the platform is show below.

```
HTTP/1.1 201 Created
Location: http://example.org/myPaaS/templates/1
Content-Type: ...
Content-Length: ...

...
```

Once the application is deployed, the administrator starts the application by sending an HTTP POST request to the resource that represents the deployed application, which was obtained in the previous step ("/myPaaS/templates/1").

```
POST /myPaaS/templates/1 HTTP/1.1
Host: example.org
```

On successful start the platform creates a new resource representing the running application and provides the URL of that resource "/myPaaS/apps/1" in the response as show below.

```
HTTP/1.1 201 Created
Location: http://example.org/myPaaS/apps/1
Content-Type: ...
Content-Length: ...

...
```

The administrator can now monitor the running application by sending an HTTP GET request to the resource that represents the running application, which was obtained in the previous step ("/myPaas/apps/1").

```
GET /myPaaS/apps/1 HTTP/1.1
Host: example.org
```

The response contains the JSON representation of the running application as shown below.

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: ...

{
  "uri": "http://example.org/myPaaS/apps/1",
  "name": "Hello Cloud App",
  "type": "assembly",
  "description": "Hello Cloud Application Running in a PaaS Env",
  "created": "2012-06-04T20:47Z",
  "definition": "...",
  "applicationComponents": [
    {
      "href": "/myPaaS/apps/1/acs/1"
    },
    {
      "href": "myPaaS/apps/1/acs/2"
    }
  ],
  "platformComponents": [
    {
      "href": "/myPaaS/pcs/1"
    },
    {
      "href": "myPaaS/pcs/2"
    }
  ],
  "assemblyTemplate": "/myPaaS/templates/1",
  "resourceState": {
    "state": "RUNNING"
    }
}
```

1.4 Non-Goals

The specification of functional interfaces specific to services provided by individual components (see Application Components and Platform Components, below) is out of scope for this document. This is because such interfaces may be quite diverse and differ significantly from platform to platform.

1.5 PaaS Roles

There are many roles that can be defined for a PaaS environment. For the purposes of this specification we identify four roles:

Application Developer: The person that builds and tests an application and presents the developed artifacts for deployment.

Application Administrator: The person that deploys applications and manages the application throughout its life-cycle.

Together these two roles make up the consumers of the management API described in this specification. This specification is intended mainly for Application Administrators, though it does constraint the artifacts that an Application Developer presents for deployment.

Platform Administrator: The person that manages the platform. This specification describes some of the functions of a Platform Administrator, though most of the functions of this role are outside its scope.

Application End-User: A user of an application deployed on the platform. The interactions of the Application end-user and the application are outside the scope of this specification.

1.6 Terminology

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in **[RFC2119]**.

Only the lower case versions of these keywords are used in this document in accordance with RFC 2119 in order to conform to ISO document requirements.

All text except examples, unless otherwise labeled, is normative. All examples are non-normative.

1.7 Normative References

[RFC2119]	Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels”, BCP 14, RFC 2119, March 1997. http://www.ietf.org/rfc/rfc2119.txt.
[RFC2616]	R. Fielding et al, “Hypertext Transfer Protcol – HTTP/1.1”, IETF RFC 2616, June 1999. http://www.w3.org/Protocols/rfc2616/rfc2616.txt
[RFC2617]	J. Franks et al, “HTTP Authentication: Basic and Digest Access Authentication”, IETF RFC 2617, June 1999. http://www.ietf.org/rfc/rfc2617.txt
[RFC 3986]	T.Berners-Lee et al, “Uniform Resource Identifiers (URI): Generic Syntax”, IETF RFC 3986, August 1998. http://www.ietf.org/rfc/rfc3986.txt
[RFC4346]	T. Dierks, E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.2”, IETF RFC 4346, April 2006. http://www.ietf.org/rfc/rfc4346.txt
[RFC4627]	D. Crockford, “The application/json Media Type for JavaScript Object Notation (JSON)”, IETF RFC 4627, July 2006. http://www.ietf.org/rfc/rfc4627.txt
[RFC5246]	T. Dierks, E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.2”, IETF RFC 5246, August 2008. http://www.ietf.org/rfc/rfc5246.txt
[ISO 8601:20044]	International Organization for Standardization, Geneva, Switzerland, “Data elements and interchange formats -- Information interchange - - Representation of dates and times”, March 2008. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=40874

1.8 Non-Normative References

[SP800-145]	Peter Mell, Timothy Grance, “The NIST Definition of Cloud Computing”, Special Publication 800-145, September 2011. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf
[OVF]	Distributed Management Task Force, “Open Virtualization Format Specification”, DSP0243, 12 January 2010. http://www.dmtf.org/sites/default/files/standards/documents/DSP0243_1.1.0.pdf
[Git]	The Software Freedom Conservancy, “Git, the fast version control system”, March 2012. http://git-scm.com/

2 Architecture

2.1 Management Model

This document specifies the self-service management API that a Platform as a Service offering presents to the consumer of the platform. The API is typically the interface into a platform implementation layer that controls the deployment of applications and their use of the platform.

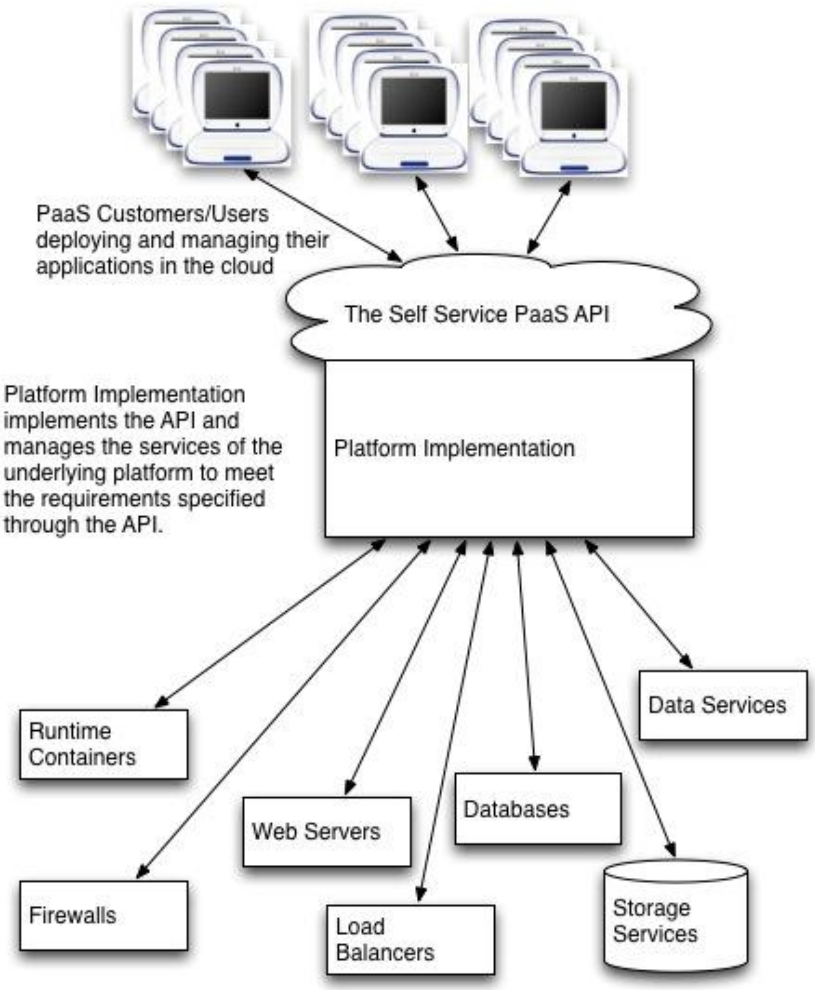
PaaS Customers/Users deploying and managing their applications in the cloud
The Self Service PaaS API
Platform Implementation implements the API and manages the services of the underlying platform to meet the requirements specified through the API.
Platform Implementation
Runtime Containers
Firewalls
Web Servers
Load Balancers
Databases
Data Services
Storage Services

Figure 1 - Typical PaaS Architecture

The figure above shows a typical architecture of a Platform as a Service cloud. The platform implementation is a management client of the underlying resources that transforms (through policies) the application requirements expressed by the Application Administrator into provisioning and other operations on those resources. The Platform Administrator manages the underlying hardware, storage, networks, and software services that make up the platform through existing administrative interfaces. Thus the Application Administrator is able to concentrate on their application and it’s deployment environment rather than having to be a systems administrator, database administrator and middleware administrator as well (as is the case with IaaS).

The goal of the management interface is to provide the PaaS consumer with a model that is as simple as possible, and yet still provides the essential elements that give them control over the deployment, execution, administration and metering of their application and it's deployment environment.

2.1.1 PaaS Resource Model

The model of resources manipulated through the interface, in combination with the protocol used to remotely accomplish this, constitutes the self-service PaaS management API. The model contains resource types corresponding to the artifacts discussed earlier.

2.1.1.1 Platform

The Platform resource is the primary view of the platform and what is running on it. The Platform references deployed applications (as Assembly Templates) as well as running applications (as Assemblies) and enables discovery of the PaaS offering in terms of Platform Components and their capabilities. The Platform also determines the scope of access for sharing amongst multiple applications.

2.1.1.2 Assemblies

The Assembly Template resource represents a deployed application and its dependencies on the platform and other application components.

The Assembly resource represents a running application. Operations on an Assembly affect the components and elements of that application.

2.1.1.3 Components

There are two kinds of components, application components and platform components, each of which may exist in either template or instantiated forms.

The Application Component Template resource represents a discrete configuration of a deployed application component. The attributes of this resource represent the configuration values that will be applied to the component upon instantiation.

The Application Component resource represents an instantiated instance of an application component.

The Platform Component Template resource represents a discrete configuration of a platform component. The attributes of this resource represent the configuration values that will be applied to the component upon instantiation.

The Platform Component resource represents an instantiated instance of a platform component in use by an application. In addition to the configuration metadata (derived from the corresponding Platform Component Template on instantiation), the attributes of this resource also include metering information for this component. Such information is typically used in generating the consumer's bill.

2.1.1.4 Capabilities and Requirements

Like Templates, Capability resources represent the configuration of instantiatable Components (Application Components or Platform Components). Unlike Templates, which delineate discrete configurations, Capabilities specify ranges of configuration values.

Requirement resources are created by the Application Developer or Application Administrator to express an application's dependency on a component that is capable of satisfying a certain set of requirements. For example, an application component may depend upon a messaging service that supports a certain version of an AMQP API, can accept messages of up to 2MB in size, and which provides a persistent message store.

The process of matching Requirements with Capabilities is referred to as “requirement resolution”.

2.1.2 Resource Relationships

A Platform provides a set of Platform Components that may be used by the invoking applications. Examples of Platform Components include a servlet container, a web server, an LDAP store, and a database instance. The implementation and operation of Platform Components is managed by the Platform Administrator. Platforms may also provide higher-level business components such as a business rules manager to gain competitive advantage and developer loyalty.

An application is composed of a set of Application Components that depend on one or more Platform Components. Examples of Application Components include Ruby gems, Java libraries, and PHP modules. Application Components may also include non-code artifacts such as datasets and collections of identity information.

Application Components may also interact with other Application Components. Thus, an Application Component has two different sets of dependencies. It depends on the Components provided by the platform, and depends on services provided by other Application Components. Such Application Components may be on the same platform or may reside at some other location. The Assembly resource is used to aggregate the management of these components as shown below:

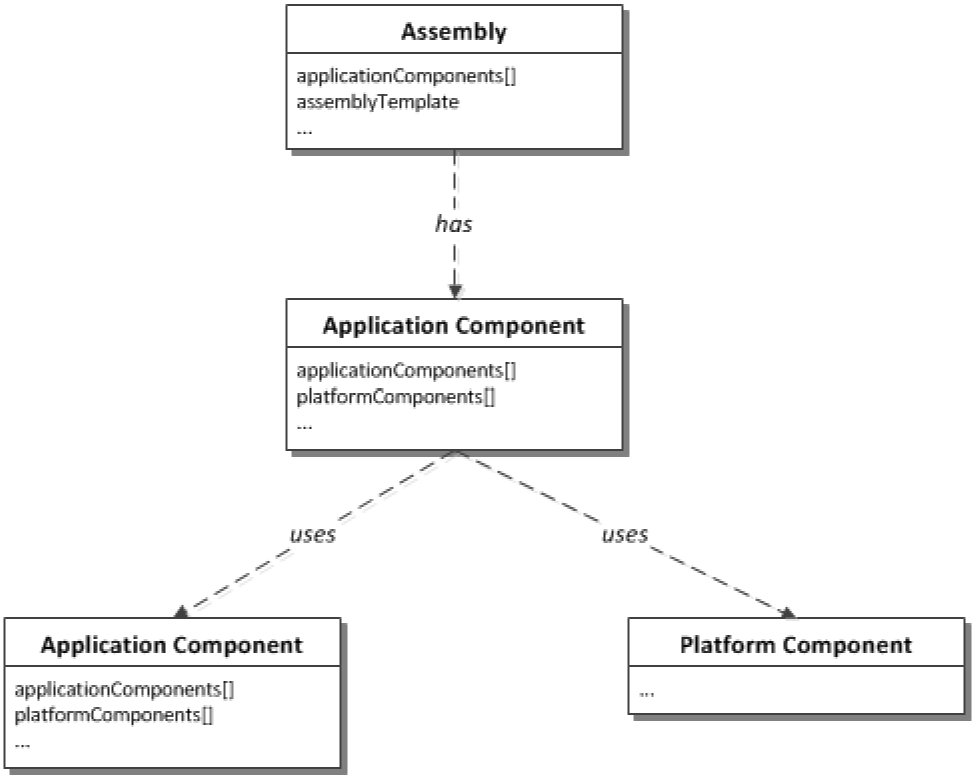

Figure 2 - Runtime Management Resources

Platform Components have a set of Platform Component Capabilities that an application can choose from to meet its requirements. Applications can tailor Platform Component Requirements (a refinement or narrowing of the configuration ranges in the Capabilities) to meet their needs based on the range of parameters expressed in the Platform Component Capability.

The relationships of an Assembly Template to Application Component Templates and Platform Component Templates, or their Requirements are shown below:

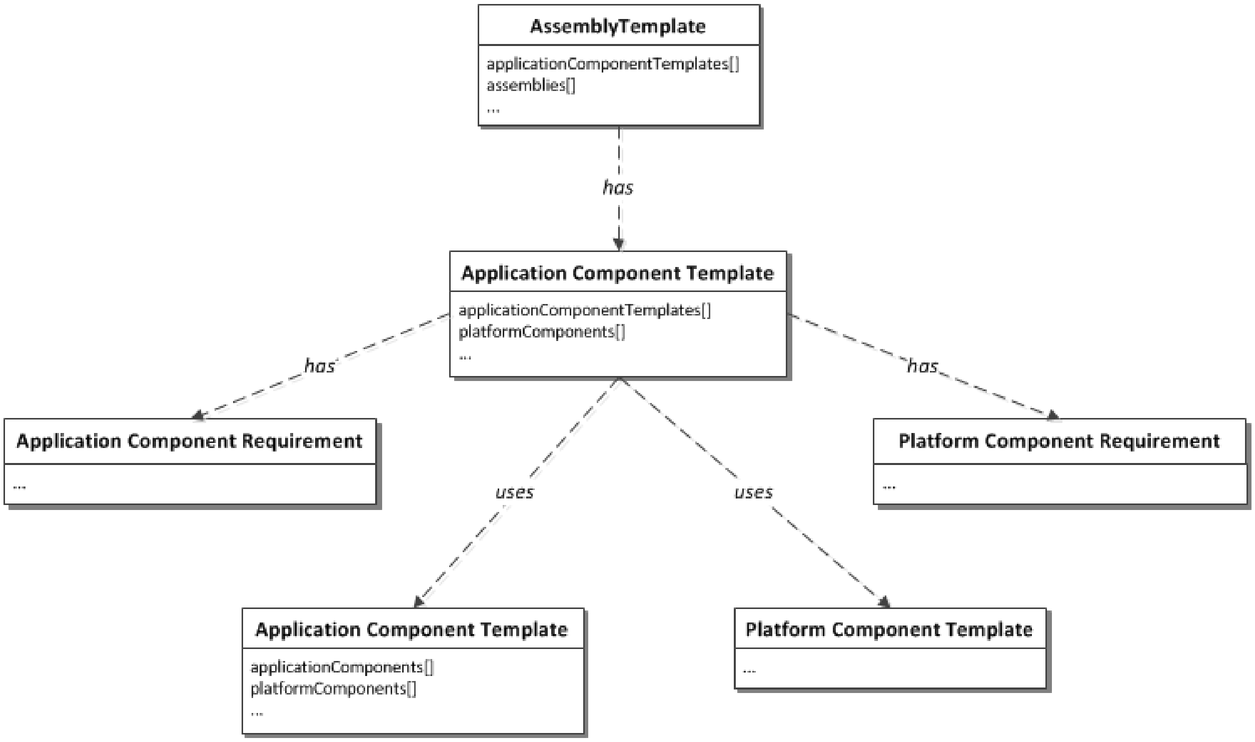
AssemblyTemplate
applicationComponentTemplates[]
assemblies[]
...
has
Application Component Template
applicationComponentTemplates[]
platformComponents[]
...
has
has
Application Component Requirement
...
Platform Component Requirement
...
uses
uses
Application Component Template
applicationComponents[]
platformComponents[]
...
Platform Component Template
...

Figure 3 - Template Management Resources

An Application Component can express the exact configuration of its dependency on other Components using one of the Component Template resources (either an Application Component Template or Platform Component Template). Alternatively, it can express a range of configuration values that are acceptable for that dependency by using one of the Component Requirement resources (either an Application Component Requirement or Platform Component Requirement). This might be done, for example, in an ADE when the existing Component Templates are not known. During the deployment, these Requirements are matched with Capabilities that have attribute values that fall within the ranges specified by the Requirements.

An Application Component Template cannot be instantiated unless all of its dependencies are satisfied. An Assembly Template cannot be instantiated until all of its Application Component Templates are successfully instantiated.

When instantiated, an Assembly Template results in an Assembly. The Assembly resource references the Assembly Template it was instantiated from, although attributes of the Assembly can deviate from the original template. An Assembly can be snapshot at a point in time and the resulting snapshot can be used to create a new Assembly Template.

A Deployment Plan is packaging management meta-data that includes a serialized copy of the Assembly Template resource and all dependent management resources such as Application Component Templates. Deployment Plans are an essential part of a Platform Deployment Package (PDP).

A Platform Deployment Package is an archive of executable images, dependency descriptions and metadata (management resources) that can be used to move an Application and its Components from Platform to Platform, or between an Application Development Environment and a Platform. It can be thought of as a serialized form of an Assembly Template and all of its dependent management resources.

2.1.3 Management Model Diagrams

The figures in this section list attributes for various management resources. The attributes listed here are not exhaustive. For a comprehensive list of attributes for resources see Section 5.

2.1.3.1 Basic Platform Resources

The CAMP model includes the resources below when no Assemblies have been created.

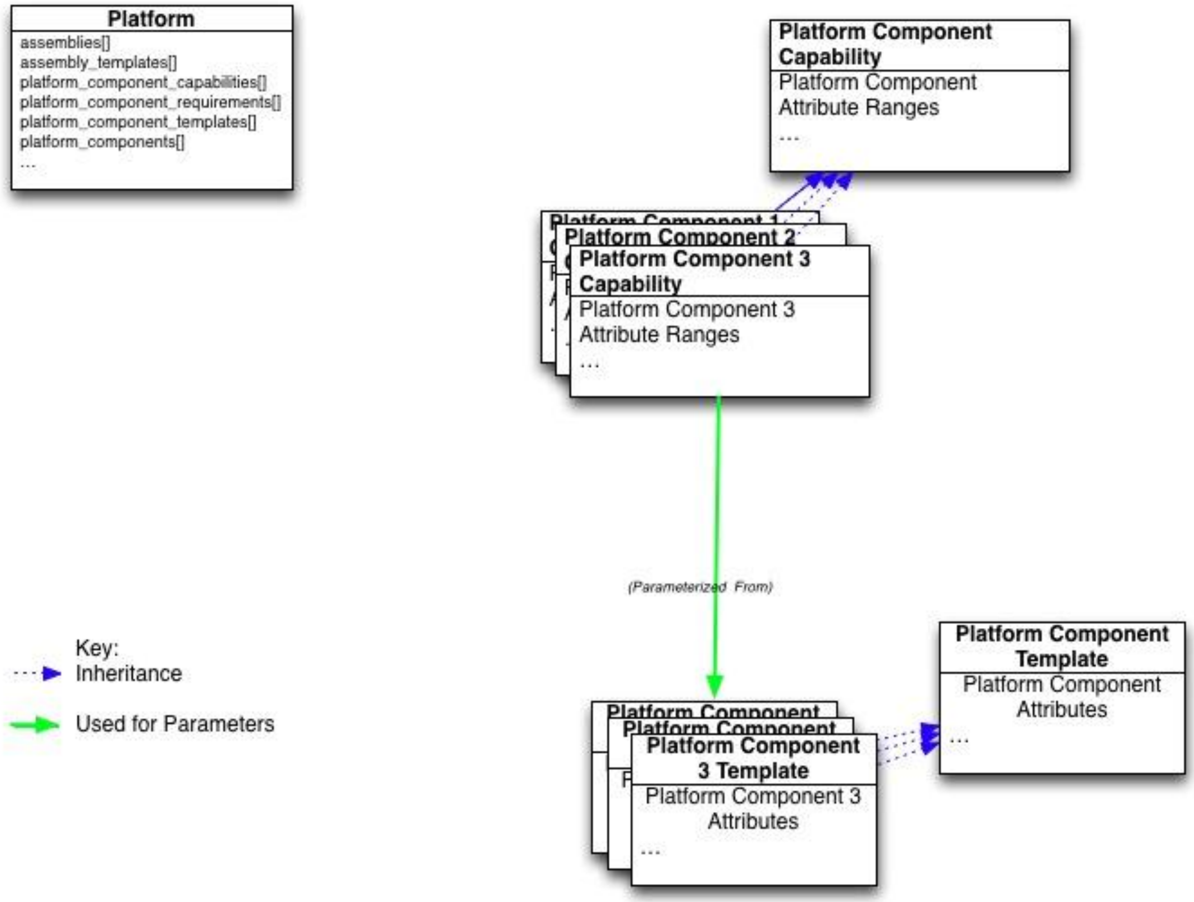
Platform
assemblies[]
assembly_templates[]
platform_component_capabilities[]
platform_component_requirements[]
platform_component_templates[]
platform_components[]
...
Platform Component Capability
Platform Component Attribute Ranges
...
Platform Component 1
Platform Component 2
Platform Component 3 Capability
Platform Component 3 Attribute Ranges
...
(Parameterized From)
Key:
Inheritance
Used for Parameters
Platform Component Template
Platform Component Attributes
...
Platform Component
Platform Component
Platform Component 3 Template
Platform Component 3 Attributes
...

Figure 4 - Basic Platform Resources

An empty Platform will have a number of resources visible through the API when the Application Administrator first accesses a new account. The Platform resource is used to find the other resources in this diagram. The various Platform Component Capabilities allow for discovery of all the platform services that are available with value ranges for each service’s attributes. The various Platform Component Templates may represent pools of previously configured platform resources and represent this configuration as specific values for each of the attributes. These values are chosen from the range of values in the corresponding Platform Component Capability.

2.1.3.2 Resources Loaded

After loading a PDP, the model may appear as follows:

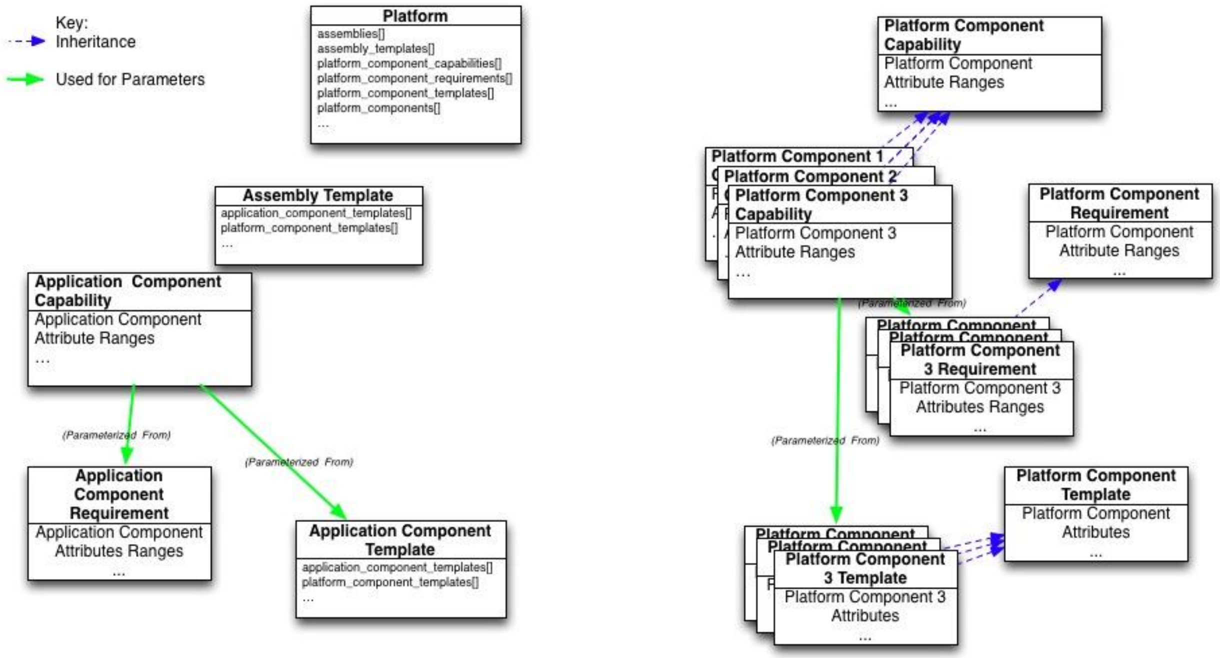
Key:
Inheritance
Used for Parameters
Platform
assemblies[]
assembly_templates[]
platform_component_capabilities[]
platform_component_requirements[]
platform_component_templates[]
platform_components[]
...
Assembly Template
application_component_templates[]
platform_component_templates[]
...
Application Component Capability
Application Component Attribute Ranges
...
(Parameterized From)
(Parameterized From)
Application Component Requirement
Application Component Attributes Ranges
...
Application Component Template
application_component_templates[]
platform_component_templates[]
...
Platform Component Capability
Platform Component Attribute Ranges
...
Platform Component 1
Platform Component 2
Platform Component 3 Capability
Platform Component 3 Attribute Ranges
...
Platform Component Requirement
Platform Component Attribute Ranges
...
(Parameterized From)
Platform Component
Platform Component
Platform Component 3 Requirement
Platform Component 3 Attributes Ranges
...
(Parameterized From)
Platform Component Template
Platform Component Attributes
...
Platform Component
Platform Component
Platform Component 3 Template
Platform Component 3 Attributes
...

Figure 5 - Loaded Resources

An Assembly Template now is available with one or more Application Component Templates, Application Component Requirements and Platform Component Requirements. The Application Component Template resources represent actual code that was loaded with the deployment plan. The Application Component Requirement resources were used to find Application Component Templates (pre-existing software libraries for example) that were not part of the deployment plan but are required for the application. The Application Component Capability resources show the range of configurable attributes that the loaded Application Components can take on, when reused by future Application Components.

The Platform Component Requirement resources were used to find Platform Component Templates (pre-configured platform resources) that were part of the platform and required by the loaded Application Components, but perhaps unknown at the time the deployment plan was created (in an ADE for example).

2.1.3.3 Instantiated Resources

To start an application utilizing all of the previously configured Application Components and Platform Components, an Assembly is created using the Assembly Template. This also causes the Application Components and Platform Components corresponding to their respective templates to be created as shown below.

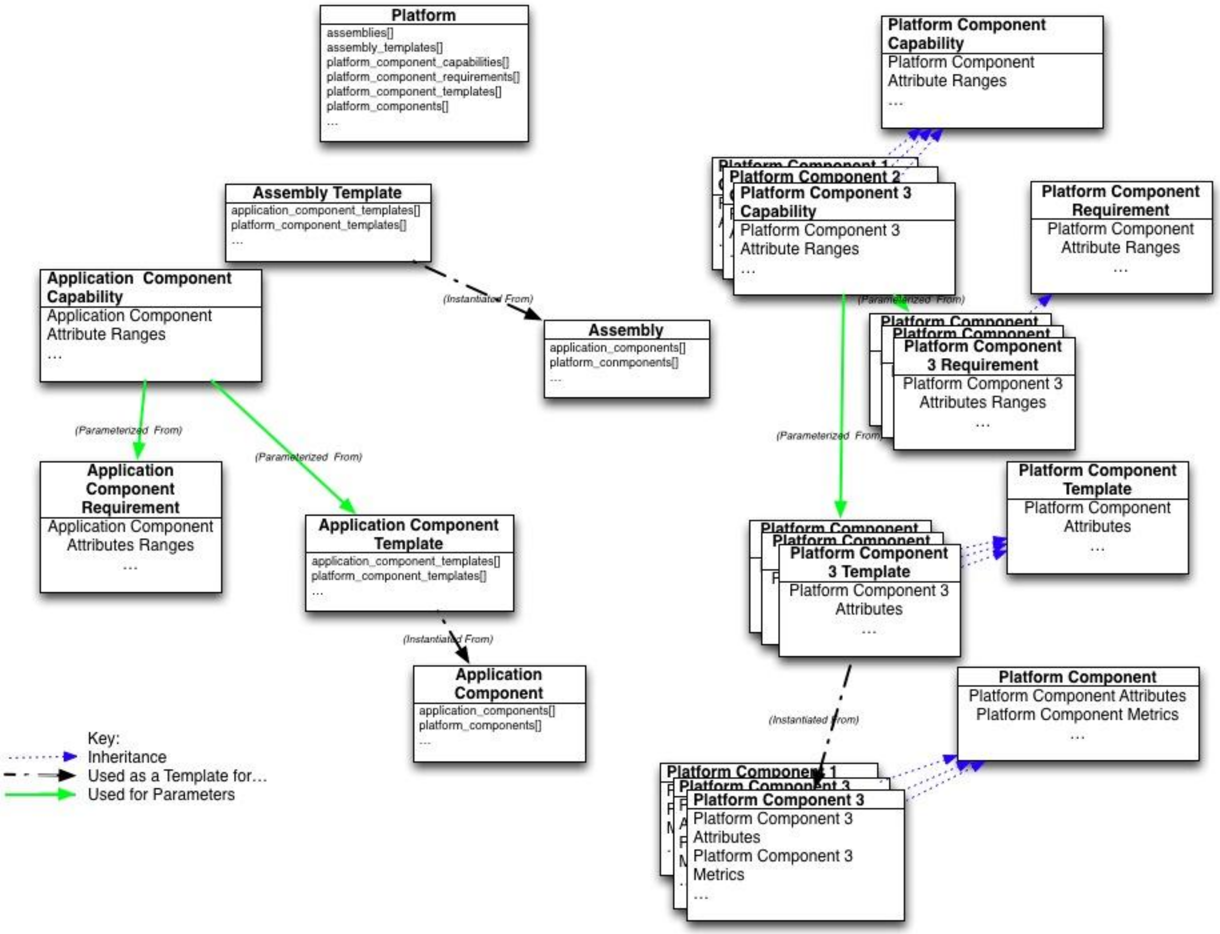
Platform
assemblies[]
assembly_templates[]
platform_component_capabilities[]
platform_component_requirements[]
platform_component_templates[]
platform_components[]
…
Assembly Template
application_component_templates[]
platform_component_templates[]
…
(Instantiated From)
Assembly
application_components[]
platform_conmponents[]
…
Application Component Capability
Application Component Attribute Ranges
…
(Parameterized From)
(Parameterized From)
Application Component Requirement
Application Component Attributes Ranges
…
Application Component Template
application_component_templates[]
platform_component_templates[]
…
(Instantiated From)
Application Component
application_components[]
platform_components[]
…
Key:
Inheritance
Used as a Template for…
Used for Parameters
Platform Component Capability
Platform Component Attribute Ranges
…
Platform Component 1
Platform Component 2
Platform Component 3 Capability
Platform Component 3 Attribute Ranges
…
Platform Component Requirement
Platform Component Attribute Ranges
…
(Parameterized From)
Platform Component
Platform Component
Platform Component 3 Requirement
Platform Component 3 Attributes Ranges
…
(Parameterized From)
Platform Component Template
Platform Component Attributes
…
Platform Component
Platform Component
Platform Component 3 Template
Platform Component 3 Attributes
…
(Instantiated From)
Platform Component
Platform Component Attributes
Platform Component Metrics
…
Platform Component 1
Platform Component 3
Platform Component 3
Platform Component 3 Attributes
Platform Component 3 Metrics
…

Figure 6 - Instantiated Resources

To manage the operation of the application, the Application Administrator interacts with the Assembly resource and the related Application Component and Platform Component resources.

The traversal of the resources in the model can be accomplished by following arrays of URIs from each resource to the other resources it depends on as shown by the red arrows in the figure below:

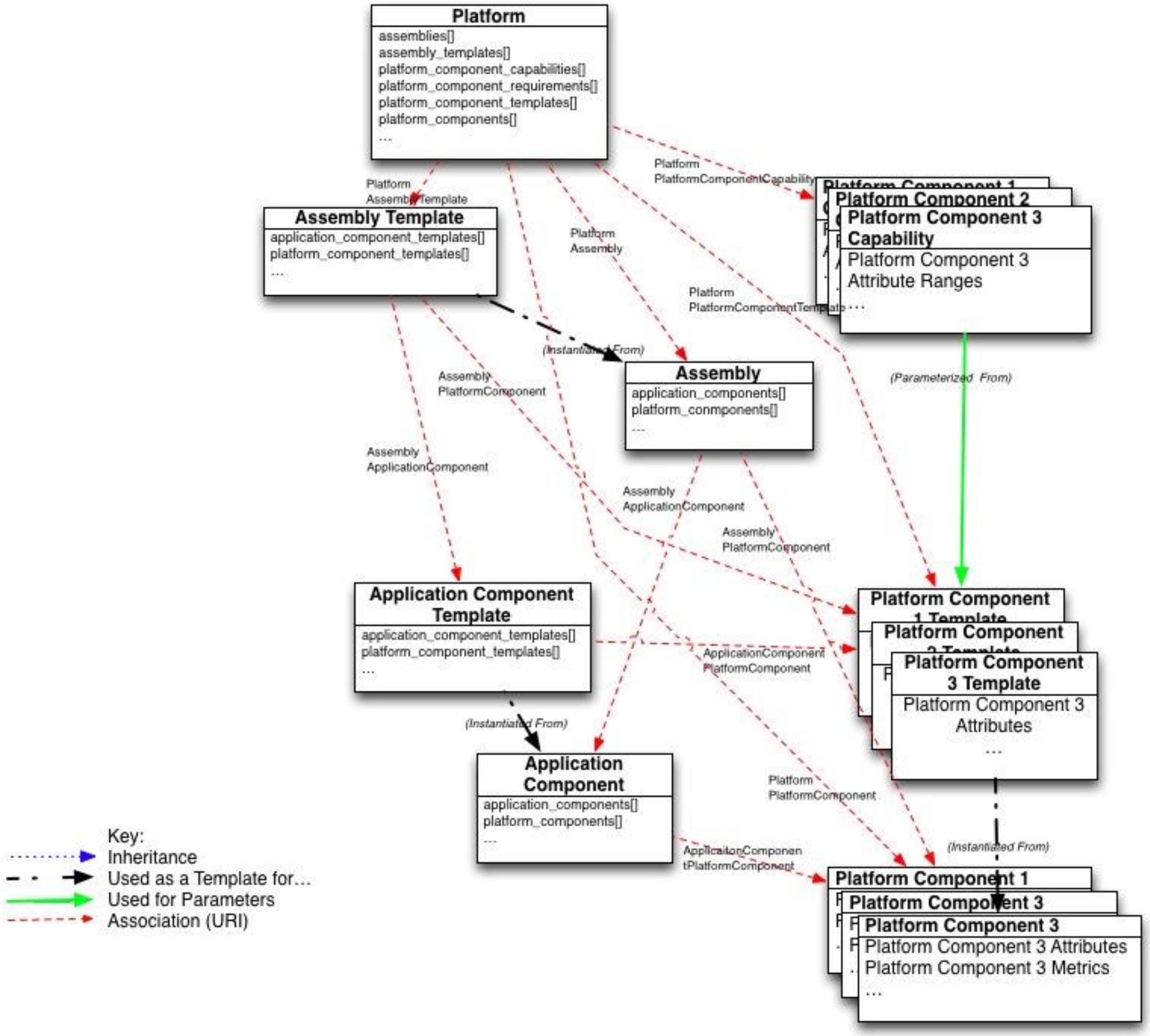
Platform
assemblies[]
assembly_templates[]
platform_component_capabilities[]
platform_component_requirements[]
platform_component_templates[]
platform_components[]
…
Platform AssemblyTemplate
Assembly Template
application_component_templates[]
platform_component_templates[]
…
Platform Assembly
Platform PlatformComponentCapability
Platform Component 1
Platform Component 2
Platform Component 3 Capability
Platform Component 3 Attribute Ranges
…
Platform PlatformComponentTemplate
(Instantiated From)
Assembly
application_components[]
platform_conmponents[]
…
(Parameterized From)
Assembly PlatformComponent
Assembly ApplicationComponent
Assembly ApplicationComponent
Assembly PlatformComponent
Application Component Template
application_component_templates[]
platform_component_templates[]
…
Platform Component 1 Template
Platform Component 2 Template
Platform Component 3 Template
Platform Component 3 Attributes
…
ApplicationComponent PlatformComponent
(Instantiated From)
Application Component
application_components[]
platform_components[]
…
Platform PlatformComponent
ApplicaitonComponentPlatformComponent
(Instantiated From)
Platform Component 1
Platform Component 3
Platform Component 3
Platform Component 3 Attributes
Platform Component 3 Metrics
…
Key:
Inheritance
Used as a Template for…
Used for Parameters
Association (URI)

Figure 7 - Traversing Resources

2.2 Importing a Platform Deployment Package

A Platform Deployment Package may be imported into the platform and as a result the following takes place:

1. One or more Assembly Templates are created with URIs to the other elements and with any binary executables installed onto the platform.
2. One or more Application Component Templates are created with URIs to the other elements and with and binary executables installed onto the platform.
3. Zero or more Platform Component Requirements are created, representing the requirements on the underlying platform components. Connections may be made to existing Platform Component Templates whose attribute values fall within the ranges specified by the requirements.
4. Zero or more Application Component Requirements are created, representing the requirements on previously installed Application Components. Connections may be made to existing Application Component Templates whose attribute values fall within the ranges specified by the requirements.

2.3 Exporting a Platform Deployment Package

A Platform Deployment Package may be exported from the platform by an operation on an Assembly or an Assembly Template. The Assembly (Assembly Template), dependent Application Components (Application Component Templates) and Platform Components (Platform Component Templates) and associated Capabilities Requirements, if any, as well as all related images are then used to create the Deployment Plan and Platform Deployment Package. A URI to the Platform Deployment Package is returned as a result.

3 Application Lifecycle

The following state diagram depicts lifecycle of an Application from the perspective of the PaaS Management API.

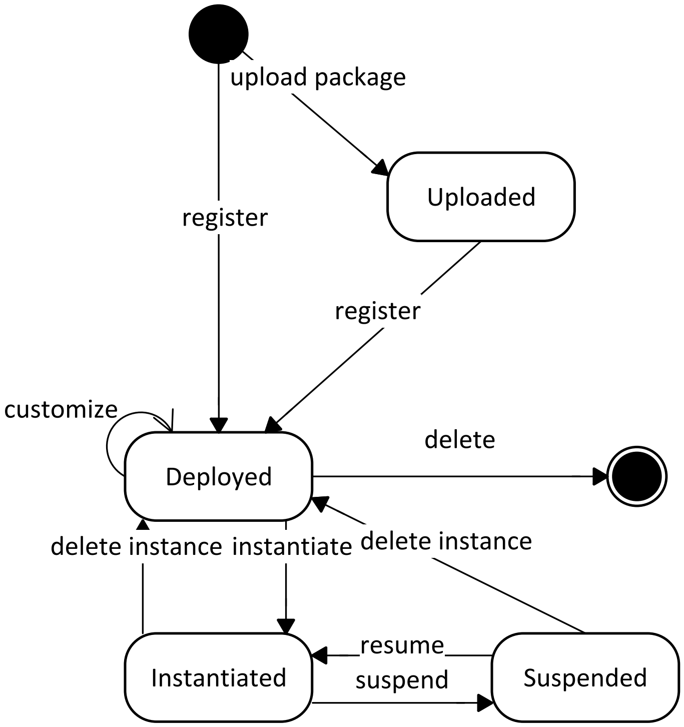

Figure 8 - Application Lifecycle

The following sections describe this diagram. It is important to note:

- The states and transitions shown in this diagram are abstract and do not necessarily correspond in a one-to-one fashion with any specific information or activities maintained or performed by either the Consumer or the Provider.
- The entities that define “the Application” vary between states. For example, in the “Deployed” state, the Application is represented by the graph of entities rooted in an Assembly Template. In in the “Instantiated” state, the Application is represented by the graph of entities rooted in an Assembly.
- Platform-specific environments might add to or extend this lifecycle to accommodate the needs of a particular framework or language. Such extensions are outside the scope of this specification.

3.1 Uploaded

While the development of Applications and their components is outside the scope of this specification, nonetheless it is recognized that such development may occur either within the PaaS environment itself, or externally (i.e. on the Application Developer’s local machine). In the latter case, the executable images, and Deployment Plan, and other metadata that make up the Application shall be imported into the platform in the form of a PDP. Because the PDP represents a significant interoperability point, the Uploaded state exists to depict the *upload package* transition. During this transition the PDP is un-archived, validated, and parsed (not necessarily in that order). Invalid PDPs will cause the *upload* *package* transition to fail. Once uploaded, the Application/PDP may be persisted in some fashion outside the scope of this specification. The invocation of the *register* transition may occur either automatically or via some unspecified action on the part of the Application Administrator.

3.2 Deployed

The Application enters the Deployed state via one of the *register* transitions. During these transitions the Deployment Plan is parsed and validated and the corresponding Assembly Template, Application Component Templates, Application Component Requirements, etc. are created as addressable entities in the Platform. The *register* transition also encompasses the process of dependency resolution in which Platform Component Requirements and Application Component Requirements are matched (respectively) against candidate Platform Component Templates and Application Component Templates. Note this process may require interaction with the Consumer. If the required dependencies are not resolved or if the metadata is incomplete, the register transition shall not happen.

Once deployed, an Application may be customized by the Application Administrator. This is represented by the *customize* transition. Customization may include changes to configuration parameters, modifying dependencies, etc. Application Administrators may elect to copy or create new Assembly Templates – effectively creating a new Application.

Deployed Applications may be removed as represented by the *delete* transition. Deleting an Application removes its Assembly Template, Application Component Template, Platform Component Requirement, and Application Component Requirement entities.

3.3 Instantiated

The Instantiated state represents a running Application. The management entity for an Application in this state is the Assembly. The Assembly and its graph of dependent Application Components and Platform Components are created upon entry to this state via the *instantiate* transition and removed during the *delete instance* transition.

3.4 Suspended

The optional Suspended state represents an Application that has been suspended. During this state the Application is unavailable to its clients and does not process any requests. This specification does not specify whether the suspension is done at full scale with all its existing resources allocated, or scaled back with minimum resources necessary to preserve its runtime context, or something in between. An implementation can specify scale at which suspension is done or allow clients to specify the scale of suspension via extensions. The scale at which the suspension is done affects resource utilization and therefore cost during the suspension phase. It also affects the time it takes to resume processing client requests at roughly the same rate as before suspension.

Note the *delete instance* transition has the same effect as the *delete instance* transition for an Instantiated Application.

4 Platform Deployment Package

The Platform Deployment Package (PDP) ensures portability across platforms. It can be created by a platform to export to another platform, which then imports it. It can also be created by an Application Development Environment running locally or deployed as Software as a Service in the cloud.

4.1 PDP Package Structure

A PDP consists of the following files:

- One Deployment Plan file with extension .dp
- Zero or one manifest file with the extension .mf
- Zero or one certificate with the extension .cert
- Zero or more language specific bundles
- Zero or more additional resource files

The manifest file contains a list of all the other files in the package and may contain file-related metadata such as checksums. The certificate is used to sign the package manifest. The PDP may be delivered as a single tar file consisting of the above items.

4.2 Deployment Plan

The Deployment Plan is an XML file that contains the templates, dependencies and platform component configurations that enables the platform to create the environment and instantiate all the components in order to launch the application. Attributes in the deployment plan are used by the platform to create the Assembly Template, Component Templates and Component Dependencies.

4.3 Deployment Plan Schema

This section will be completed when the resource encoding in section 4 is complete. The intent is to reuse that encoding in the Deployment Plan Schema such that implementations have a ready capability to generate Platform Deployment Plans from their internal representations.

5 Resources

The following sections describe the resources defined by this specification.

5.1 Common Types

Resource attributes are defined using the following types:

5.1.1 Boolean

As defined by JSON [RFC4627], a token having a literal value of either `true` or `false`.

5.1.2 String

A UNICODE string as defined by JSON [RFC4627].

5.1.3 URI

A String (see above) that conforms to the syntax defined in RFC 3986 [RFC3986].

5.1.4 Timestamp

A String (see above) that conforms to the syntax defined in ISO 8601 [ISO 8601:20044] with the restriction that time stamps shall be expressed in UTC (Coordinated Universal Time), with a special UTC designator ("Z").

5.1.5 Link

The management model defined in this specification involves resource entity attribute values that link to other resource entities. For example, one of the Platform resource entity attribute values points to Assembly Templates. The “Link” type defined here is used for such attribute values.

Link has one or more attributes. The attribute href whose value is a URL MUST be present. Other attributes, not defined in this specification, MAY also be present. A JSON example is shown below:

```
"LinkName": {"href": "http://example.org/paas1/assemblies/1"}
```

5.1.6 ResourceState

This type is used to denote the state of the resource within the lifecycle defined for that resource.

The following table is the data model of this type.

Table 1 – ResourceState Data Model

Field	Type	Occurs	Description
state	String	1	Current state of the resource. This is a label containing lifecycle state (e.g. INITIATED, CREATING, CREATED, DESTROYING, DESTROYED, READY). When there are vendor extensions, the vendor shall publish and document their semantics
progress	Integer between 0 and 100	0..1	Indicates the progress made as an approximate percentage. Not all state labels assign semantic meaning to this field

messages	Message[]	0..1	Include the message data model instances to denote noteworthy communications

5.2 Attribute Constraints

Resource attributes are constrained along a number of axes. These are:

5.2.1 Cardinality

This constraint indicates the cardinality of the attribute.

5.2.2 Mutability

This constraint indicates the mutability of the attribute's value(s). "immutable" indicates that the value of the attribute, once set, shall not change for the lifetime of the resource. "mutable" indicates that the value of the attribute may change due to the actions or activity of either the provider or the user.

5.2.3 Writeable

This constraint indicates the ability of a user to update the value of the attribute. It is only relevant for mutable attributes. "false" indicates that the value(s) of the attribute can only be changed by the provider. "true" indicates that users may change the value(s) of the attribute.

5.3 Common Resource Attributes

All the resources in this specification contain the following common attributes:

5.3.1 uri

Type: URI

Cardinality: 1

Mutability: immutable

This attribute expresses the URI of the resource.

5.3.2 name

Type: String

Cardinality: 1

Mutability: mutable

Writeable: true

This attribute expresses the human-readable name of the resource.

5.3.3 description

Type: String

Cardinality: 0..1

Mutability: mutable

Writeable: true

This attribute expresses the human-readable description of the resource.

5.3.4 created

Type: Timestamp

Cardinality: 1

Mutability: immutable

This attribute expresses the time which the resource was created.

5.3.5 tags

Type: String[]

Cardinality: 0..1

Mutability: mutable

Writeable: true

This attribute is an array of String values that may be assigned by the provider or the user. These values can be used for keywording and terms-of-interest.

5.3.6 type

Type: String

Cardinality: 1

Mutability: immutable

Writeable: false

This attribute expresses the CAMP resource type. Every CAMP resource type defined in this specification specifies the required value for this attribute.

5.4 Error Response Message

Successful requests will generally return an HTTP status code of 200 (OK), 201 (Created), 202 (Accepted), or 204 (No Content), to indicate that the requested action has been successfully performed or submitted. In addition, they might include a response message body containing a representation of the requested information. However, it is possible for a number of things to go wrong. The various underlying causes are described (as discussed in the previous section) by various HTTP status codes in the range 400-499 (for client side errors) or 500-599 (for server side problems).

If a response is returned with an error status code (400-499 or 500-599), where appropriate, the server is encouraged to include a response message body containing an *ErrorMessage* object (as defined below). The text values of such messages might be used, for example, to communicate with a human user of the client side application.

The list of messages included in a single error response is encapsulated in the ErrorMessage data model.

Table 2 – ErrorMessage Data Model

Field	Type	Occurs	Description
message	Message	0..n	Zero or more message data for each individual message.

An individual message contains the following fields:

Table 3 - Message Data Model

Field	Type	Occurs	Description
code	String	0..1	Symbolic error code identifying the type of error reported by this message
field	String	0..1	Name of the field from the request data model that this message is associated with

hint	String	0..1	Localized text further describing the nature of the problem, possibly including potential workarounds that the client could try
text	String	1	Localized text describing the nature of the problem reported by this message
severity	String	0..1	Label indicating the severity of the error condition represented by this message Vendor shall publish the enumerators that are associated with this field and their semantics
stackTrace	String	0..1	Vendor specific stack trace associated with this message
source	String	0..1	Symbolic identifier of the implementation component that triggered this message
message-id	String	0..1	A unique string that identifies this particular message
profile	URI	0..1	A reference to the standard URI to indicate the meaning of this message

The *profile* attribute indicates the semantic meaning of the message which clients may handle automatically. Messages with the same profile shall adhere to the semantic requirements of that profile, but the payload (*hint*, *text*, *severity*, *stackTrace*) may be different. In other words, given a profile, clients processing the message should be able to subsequently interact with the providers in a consistent manner across.

Each provider may extend the profile to include specific scenarios and use cases.

The information captured in the *message* data element should be complementary to the HTTP status code, and COULD provide more detailed information. However, it shall not contradict the HTTP status code that is returned with the request.

The following table outlines the common profiles that would accompany this specification.

Table 4 - Common Message Profiles

Profile	**Description**
/msg/unknown	Unknown error and information given is descriptive in nature
/msg/security	Security issues
/msg/security/authentication	An authentication error
/msg/access	Access violation error
/msg/allocation	Allocation related issues
/msg/allocation/insufficient	Insufficient resource to satisfy the request
/msg/infrastructure	Infrastructure related issues
/msg/infrastructure/maintenance	The request cannot be immediately responded due to the infrastructure being in maintenance status

5.5 Extensibility to the resource model

To support returning selective attributes, the following shall be observed by the service provider:

- The attribute name of a resource shall contain alphanumeric characters with “_” and “-“. Thus, [a-zA-Z0-9_\-]

5.6 Platform

For a user, a Platform represents the user’s starting view of all accessible resources and deployed entities. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "platform",
  "description": String,
  "created": Timestamp,
  "tags": [
    String, +
  ], ?
  "specificationVersion": String[],
  "implementationVersion": String, ?
  "assemblyTemplates": [
    {"href": URI}, +
  ], ?
  "assemblies": [
    {"href": URI}, +
  ], ?
  "platformComponentTemplates": [
    {"href": URI}, +
  ], ?
  "platformComponentCapabilities": [
    {"href": URI}, +
  ], ?
  "platformComponents": [
    {"href": URI}, +
  ], ?
  "resourceState": {
    "state": String,
    "progress": String, ?
    "messages": [
      { Message }, +
    ] ?
  }
}
```

The Platform resource contains the following attributes:

5.6.1 specificationVersion

Type: String[]

Cardinality: 1

Mutability: immutable

This attribute lists the version(s) of this specification this instance the Platform supports. See section 6.8 for details on concurrent version support.

5.6.2 implementationVersion

Type: String

Cardinality: 0..1

Mutability: immutable

This attribute expresses the vendor specific version of the server implementation.

5.6.3 assemblyTemplates

Type: Link[]

Cardinality: 0..1

Mutability: mutable

Writeable: false

This attribute is an array of Links to the AssemblyTemplates that are accessible to the user.

5.6.4 assemblies

Type: Link[]

Cardinality: 0..1

Mutability: mutable

Writeable: false

This attribute is an array of Links to the Assembly resources that are accessible to the user.

5.6.5 platformComponentTemplates

Type: Link[]

Cardinality: 0..1

Mutability: mutable

Writeable: false

This attribute is an array of Links to the PlatformComponentRequirement resources that are accessible to the user.

5.6.6 platformComponentCapabilities

Type: Link[]

Cardinality: 0..1

Mutability: mutable

Writeable: false

This attribute is an array of Links to the PlatformComponentCapability resources that are accessible to the user.

5.6.7 platformComponents

Type: Link[]

Cardinality: 0..1

Mutability: mutable

Writeable: false

This attribute is an array of Links to the PlatformComponentCapability resources that are accessible to the user.

5.6.8 resourceState

Type: ResourceState

Cardinality: 1

Mutability: mutable

Writeable: false

This attribute expresses state of the resource within the lifecycle.

5.7 AssemblyTemplate

For an Application Administrator, an Assembly Template represents the definition of the deployable Application. Application Administrators can create an Assembly Instance by specifying the URI of a Assembly Template as a field in a deployment request. The platform shall instantiate the Application Components and their configurations as specified in the definition of the Assembly Template. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "assemblyTemplate",
  "description": String,
  "created": Timestamp,
  "tags": [
    String, +
  ], ?
  "definition": String,
    "applicationComponentTemplates": [
    {"href": URI}, +
  ], ?
}
```

The AssemblyTemplate resource contains the following attributes:

5.7.1 definition

Type: String

Cardinality: 1

Mutability: mutable

Writeable: true

This attribute contains the definition of the AssemblyTemplate represented in some format, such as XML, that contains the metadata necessary for the platform to deploy the Application.

5.7.2 applicationComponentTemplates

Type: Link[]

Cardinality: 0..*

Mutability: mutable

Writeable: true

This attribute is an array of Links to the ApplicationComponentTemplate resources that are part of this AssemblyTemplate. An AssemblyTemplate MAY have zero references to ApplicationComponentTemplate resources, but conformant CAMP server implementations MUST NOT instantiate an Assembly using an AssemblyTemplate that does not reference at least one ApplicationComponentTemplate.

5.8 ApplicationComponentTemplate

For an Application Administrator, an Application Component Template represents the definition of the deployable Component. Application Administrators can create an Application Component by specifying the URI of an Application Component Template (with referenced Application Component Templates) as a field in a deployment request to create a standalone service. The cloud shall instantiate the Components and their configurations as specified in the definition of the Application Component Template. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "applicationComponentTemplate",
  "description": String,
  "created": Timestamp,
  "tags": [
    String, +
  ], ?
  "definition": String,
  "applicationComponentTemplates" : [
    {"href": URI}, +
  ], ?
  "platformComponentTemplates" : [
    {"href": URI}, +
  ], ?
  "applicationComponentRequirements" : [
    {"href": URI}, +
  ], ?
  "platformComponentRequirements" : [
    {"href": URI}, +
  ], ?
}
```

The ApplicationComponentTemplate resource contains the following attributes:

5.8.1 definition

Type: String

Cardinality: 1

Mutability: mutable

Writeable: true

This attribute contains the definition of the ApplicationComponentTemplate represented in some format, such as XML, that contains the metadata necessary for the platform to deploy the Application.

5.8.2 applicationComponentTemplates

Type: Link[]

Cardinality: 0..1

Mutability: mutable

Writeable: true

This attribute is an array of Links to ApplicationComponentTemplate resources. Each of these resources is a template for an Application Component that will be used by the Application Component that is instantiated from this template.

5.8.3 platformComponentTemplates

Type: Link[]

Cardinality: 0..1

Mutability: mutable

Writeable: true

This attribute is an array of Links to PlatformComponentTemplate resources. Each of these resources is a template for a Platform Component that will be used by the Application Component that is instantiated from this template.

5.8.4 applicationComponentRequirements

Type: Link[]

Cardinality: 0..1

Mutability: mutable

Writeable: true

This attribute is an array of Links to ApplicationComponentRequirement resources.

5.8.5 platformComponentRequirements

Type: Link[]

Cardinality: 0..1

Mutability: mutable

Writeable: true

This attribute is an array of Links to PlatformComponentRequirement resources.

5.9 ApplicationComponentRequirement

For an Application Administrator, an Application Component Requirement represents an Application Component's requirement on another Application Component and its required range of component capabilities. Each Application Component Requirement implements this class and adds its' own Attribute Ranges to the list below. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "applicationComponentRequirement",
  "description": String,
  "created": Timestamp,
  "tags": [
    String, +
  ], ?
  "definition": String
}
```

The ApplicationComponentRequirement resource contains the following attributes:

5.9.1 definition

Type: String

Cardinality: 1

Mutability: mutable

Writeable: true

This attribute contains the definition of the ApplicationComponentRequirement represented in some format, such as XML, that contains the metadata necessary for the platform to deploy the Application.

5.10 ApplicationComponentCapability

For an Application Administrator, an Application Component Capability represents the definition of an Applicaton Component's range of component capabilities. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "applicationComponentCapability",
  "description": String,
  "created": Timestamp,
  "tags": [
    String, +
  ], ?
  "definition": String
}
```

Each Application Component Capability implements this class and adds its' own Attributes to the list below.

5.10.1 definition

Type: String

Cardinality: 1

Mutability: mutable

Writeable: true

This attribute contains the definition of the ApplicationComponentCapability represented in some format, such as XML, that contains the metadata necessary for the platform to deploy the Application.

5.11 PlatformComponentTemplate

For an Application Administrator, a Platform Component Template represents the desired configuration of a Platform Component with specific values for the component capabilities. The specified value for each component attribute shall be in the range defined in the corresponding Platform Component Capability. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "platformComponentTemplate",
  "description": String,
  "created": Timestamp,
  "tags": [
    String, +
  ], ?
  "definition": String
}
```

Each Platform Component implements this class and adds its own Component Attributes to the list below.

5.11.1 definition

Type: String

Cardinality: 1

Mutability: mutable

Writeable: true

This attribute contains the definition of the PlatformComponentTemplate represented in some format, such as XML, that contains the metadata necessary for the platform to deploy the Application.

5.12 PlatformComponentRequirement

For an Application Administrator, a Platform Component Requirement represents an Application Component's requirement on a Platform Component and its required range of component capabilities. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "platformComponentRequirement",
  "description": String,
  "created": Timestamp,
  "tags": [
    String, +
  ], ?
  "definition": String
}
```

Each Platform Component Requirement implements this class and adds its own Attribute Ranges to the list below.

5.12.1 definition

Type: String

Cardinality: 1

Mutability: mutable

Writeable: true

This attribute contains the definition of the PlatformComponentRequirement represented in some format, such as XML, that contains the metadata necessary for the platform to deploy the Application.

5.13 PlatformComponentCapability

For an Application Administrator, a Platform Component Capability represents the definition of Platform Component and its range of component capabilities. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "platformComponentCapability",
  "description": String,
  "created": Timestamp,
  "tags": [
    String, +
  ], ?
  "definition": String
}
```

Each Platform Component Capability implements this class and adds its' own Attributes to the list below.

5.13.1 definition

Type: String

Cardinality: 1

Mutability: mutable

Writeable: true

This attribute contains the definition of the PlatformComponentCapability represented in some format, such as XML, that contains the metadata necessary for the platform to deploy the Application.

5.14 Assembly

For an Application Administrator, an Assembly represents the definition of the deployed Application at runtime. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "assembly",
  "description": String,
  "created": Timestamp,
  "tags": [
    String, +
  ], ?
  "definition": String,
  "applicationComponents": [
    {"href": URI}, +
  ], ?
  "assemblyTemplate": {"href": URI}, ?
  "resourceState": {
    "state": String,
    "progress": String, ?
    "messages": [
      { Message }, +
    ] ?
  }
}
```

The Assembly resource contains the following attributes:

5.14.1 definition

Type: String

Cardinality: 1

Mutability: immutable

This attribute contains the definition of the Assembly represented in some format, such as XML, that contains the metadata necessary for the platform to deploy the Application.

5.14.2 applicationComponents

Type: Link[]

Cardinality: 1..*

Mutability: immutable

This attribute is an array of Links to the ApplicationComponent resources that are part of this Assembly. An Assembly resource MUST have at least one reference to an ApplicationComponent resource.

5.14.3 assemblyTemplate

Type: Link

Cardinality: 1

Mutability: immutable

This attribute is a Link to the AssemblyTemplate resource from which this Assembly was created.

5.14.4 resourceState

Type: ResourceState

Cardinality: 1

Mutability: mutable

Writeable: false

This attribute expresses state of the resource within the lifecycle.

5.15 ApplicationComponent

For an Application Administrator, an Application Component represents the management of the deployed Component at runtime. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "applicationComponent",
  "description": String,
  "created": Timestamp,
  "tags": [
    String, +
  ], ?
  "definition": String,
  "applicationComponents": [
    {"href": URI} +
  ], ?
  "platformComponents": [
    {"href": URI}, +
  ], ?
  "resourceState": {
    "state": String,
    "progress": String, ?
    "messages": [
      { Message }, +
    ] ?
  }
}
```

The ApplicationComponent resource contains the following attributes:

5.15.1 definition

Type: String

Cardinality: 1

Mutability: immutable

This attribute contains the definition of the ApplicationComponent represented in some format, such as XML, that contains the metadata necessary for the platform to deploy the Application.

5.15.2 applicationComponents

Type: Link[]

Cardinality: 0..*

Mutability: immutable

This attribute is an array of Links to the ApplicationComponent resources that this ApplicationComponent depends on.

5.15.3 platformComponents

Type: Link[]

Cardinality: 0..*

Mutability: immutable

This attribute is an array of Links to the PlatformComponent resources that this ApplicationComponent depends on.

resourceState

5.15.4 Type: ResourceState

Cardinality: 1

Mutability: mutable

Writeable: false

This attribute expresses state of the resource within the lifecycle.

5.16 PlatformComponent

For an Application Administrator, a Platform Component represents the runtime instance of a platform component and its configuration of component attributes as well as metrics associated with those attributes. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "platformComponent",
  "description": String,
  "created": Timestamp,
  "tags": [
    String, +
  ], ?
  "definition": String,
  "externalManagementResource": URI, ?
  "resourceState": {
    "state": String,
    "progress": String, ?
    "messages": [
      { Message }, +
    ] ?
  }
}
```

Each Platform Component implements this class and adds its own attributes and to the list below.

5.16.1 definition

Type: String

Cardinality: 1

Mutability: immutable

This attribute contains the definition of the PlatformComponent represented in some format, such as XML, that contains the metadata necessary for the platform to deploy the Application.

5.16.2 externalManagementResource

Type: URI

Cardinality: 0..1

Mutability: immutable

A URI to an external management interface to manage this platform component (such as an IaaS API to manage the virtual machines that support this component). This is platform dependent and requires external documentation to understand its meaning.

5.16.3 resourceState

Type: ResourceState

Cardinality: 1

Mutability: mutable

Writeable: false

This attribute expresses state of the resource within the lifecycle.

6 Protocol

6.1 Transport Protocol

All of The Platform APIs are based on the Hypertext Transfer Protocol, version 1.1 [RFC2616]. Each request will be authenticated using HTTP Basic Authentication [RFC2617] unless otherwise noted. Therefore, requests sent from clients across unsecured networks should use the HTTPS protocol. TLS 1.1 [RFC4346] shall be implemented by the provider and TLS 1.2 [RFC5246] is strongly encouraged. When TLS is implemented, the following cipher suites shall be supported to ensure a minimum level of security and interoperability between implementations:

- TLS_RSA_WITH_AES_128_CBC_SHA (mandatory for TLS 1.1/1.2)
- TLS_RSA_WITH_AES_256_CBC_SHA256 (addresses 112-bit security strength requirements)
- TLS_RSA_WITH_NULL_SHA (for TLS without encryption)

6.2 URI Space

The resources in the system are identified by URIs. To begin operations, a client shall know the URI for a resource. Dereferencing the URI will yield a representation of the resource containing resource attributes and links to associated resources.

Clients shall not make assumptions about the layout of the URIs or the structure of the URIs of the resources.

6.3 Media Types

In this specification, resource representations, request bodies, and error response messages are encoded in JSON, as specified in [RFC4627]. The media-type associated with CAMP JSON resource and error response message representation is "application/json".

The Platform shall provide representations of all resources available in JSON, but other formats are allowed. This specification doesn't define any schemata or aid in validating an XML encoding.

The Platform shall accept requests from clients encoded in JSON.

If alternative formats are supported they shall be supported uniformly for all resources.

6.4 Request Headers

This API does not impose any requirements on clients' use of HTTP headers. All PUT requests that update a resource SHOULD contain the *If-Match* header field with a single entity tag value. If the *If-Match* header field value in the request does not match the one on the server-side, the server MUST send back a '412 Precondition Failed' status code.

6.5 Request Parameters

In order to address a subset of attributes in a resource, the client can use request parameters in GET and PUT requests to formulate the following

Table 5 – Request Parameters

Format	Description	Example
?SelectAttr=attr1,attr2,…	Comma (",") separated attribute names of the resource to return. If an attribute is not part of the resource, an HTTP 4XX status code shall be	Assembly132?SelectAttr=name,description,status Would access only "name", "description", "status" attributes of Assembly132.

	returned.	

The “SelectAttr” query parameter may appear more than once (separated by an “&”). The client shall URL encode the request parameters.

When one or more request parameters are specified for a PUT request, a client shall not include attributes in the request entity body that are not specified in the request parameter. Upon receiving such a request the server shall respond with a 400 status code.

6.6 Response Headers

Responses returned by the Platform makes standard use of HTTP headers. All HTTP responses that return representation of a resource SHOULD use strong *ETag* response header field indicating the current value of the entity tag for the resource.

6.7 HTTP Status Codes

The API returns standard HTTP response codes.

6.8 Concurrent API versions

Multiple API versions may be offered concurrently to allow backward compatibility for clients. To discover available versions a client issues a GET request for the *<platform-url>/api/versions* resource and may select one of the listed versions. Each available API version is assigned a namespace identified by a URL.

The response to a GET request for *<platform-url>/api/versions* will include a list of available API versions and the URL where each version may be accessed. The list of available versions shall be expressed using the common *Link* type with an additional name *version* added to the link and its corresponding value.

An example of client requests to the platform are as follows:

```
GET / => {…, "links": […, "/api/versions", …], …}

GET /api/versions => [
  {"version" : "1.0", "href": "http://paas.example.org/"},
  {"version" : "1.1", "href": "http://api.example.org/foo"},
]
```

Only versions defined by a CAMP specification can be listed. Additional features not defined in this specification can be implemented using the extensibility features discussed in this specification.

6.9 Registering a PDP

As indicated in Section 3, registering a package moves it to the deployed state. To register a PDP, a client sends a POST HTTP request to the Platform URL. The entity body of the request contains the URI that identifies the PDP that is being registered. If the URI that identifies the PDP is a relative URI, its base URI is the Platform URI. The JSON serialization of the HTTP request entity body is:

```
{"pdp_uri" : "<uri-of-the-pdp>"}
```

Where, the *pdp_uri* points to the URI of the PDP to be registered. The JSON object MAY contain additional name-value pairs that are not defined in this specification. On successful registration of the PDP, the server creates an AssemblyTemplate resource and sends a 201 Created HTTP status code with the *Location* header in the HTTP response. The *Location* header points to the newly created AssemblyTemplate resource. The server also updates the *application_templates* attribute of the Platform resource to include a reference to the newly created assembly template.

An example HTTP request-response is as follows:

```
POST /<platform-url> HTTP/1.1
Host: example.org
Content-Type: application/json
Content-Length: ...

{"pdp_uri": "/paas/pdp/1"}

HTTP/1.1 201 Created
Location: http://example.org/paas/asm_template/1
Content-Type: ...
Content-Length: ...

...
```

6.10 Instantiating an Application

Once the application is in the deployed state, a client can instantiate the application by sending a POST HTTP request to the corresponding AssemblyTemplate URL. The entity body of the request can be empty. Interpretation of a non-empty entity body of the request is implementation-dependent. On success the server creates an Assembly resource and sends a 201 Created HTTP status code with the *Location* header in the HTTP response. The *Location* header points to the newly created Assembly resource. The server also updates the *AssemblyInstances* attribute of the Platform resource to include a reference to the newly created assembly.

An example HTTP request-response is as follows:

```
POST /paas/asm_template/1 HTTP/1.1
Host: example.org

HTTP/1.1 201 Created
Location: http://example.org/paas/assembly/1
Content-Type: ...
Content-Length: ...

...
```

6.11 Suspending and Resuming an Application

To suspend, or resume an application, a client sends a POST HTTP request to the assembly resource URL. The entity body of the request contains the value of the new state for the application. The JSON serialization of the HTTP request entity body is:

```
{"new_state" : "<new-state-value>"}
```

Where, *new_state* specifies the new desired value for the application state. This specification defines two such values: "suspend", and "resume," whose semantics are as defined in Section 3. An implementation MAY have additional state values that it allows. The JSON object MAY contain additional name-value pairs that are not defined in this specification. An implementation can define additional state values or name-value pairs, to allow clients to specify the scale at which the application is suspended and resumed.

An example HTTP request-response is as follows:

```
POST /<assembly-resource-url> HTTP/1.1
Host: example.org
Content-Type: application/json
Content-Length: ...

{"new_state": "suspend"}

HTTP/1.1 200 OK
```

6.12 Deleting an Application Instance and a Deployed Application

To delete an application instance (an assembly), a client sends a DELETE HTTP request to the assembly resource URL. Similarly, to delete a deployed application, a client sends a DELETE HTTP request to the assembly template URL.

7 Conformance

TBD

Appendix A. Acknowledgments

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Participants:

[Participant Name, Affiliation | Individual Member]
[Participant Name, Affiliation | Individual Member]

Appendix B. Glossary

application – a set of components that act together to provide useful functions and are typically exposed as a service to Application end-users. An application is represented by different resources (e.g. Assembly Template, Assembly) throughout its lifecycle.

Application Component – a collection of code and/or, resources (optionally accompanied by metadata) that either provides a set of related services or functionality or contains a set of related information. Codeexamples include Ruby gems, Java libraries, and PHP modules. Examples of resources include data sets, identity sets (i.e. collections of user account and attribute information), and collections of graphical images.

Application Component Capability – a management resource that represents an Application Component's capabilities.

Application Component Requirement – a management resource that represents a requirement on an Application Component, expressed with attributes that may have value ranges.

Application Component Template - a management resource that represents an unrealized Application Component and includes a reference to the executable code as well as metadata for configuring the Application Component and referencing its platform and other components.

Application Development Environment (ADE) – a developer tool used to create an application (can be an offline tool installed locally or part of the platform offering itself).

Assembly – a management resource that represents a running application.

Assembly Template - a management resource that represents an unrealized Assembly and includes a reference to the Application Component Templates used within the Application as well as metadata for configuring the Application Component and referencing its platform and other components.

Deploy – the step of creating one or more management resources on the platform. Deployment can be done through the API for individual management resources (i.e via a POST to a URI), or can be done as part of the import of a Platform Deployment Package.

Deployment Plan - packaging management meta-data that includes a serialized copy of the Assembly Template resource and all dependent management resources such as Application Component Templates. Deployment Plans are an essential part of a Platform Deployment Package.

Platform – The collection of management resources that constitute the consumer visible view of the Platform as a Service offering. The Platform management resource is an aggregation and discovery point for all the Applications and their dependencies currently deployed and running.

Platform as a Service (PaaS) - A type of cloud computing in which the service provider offers customers/consumers access to one or more instances of a running application computing platform or application service stack.

Platform Component – a management resource that represents an application's use of a realized and running Platform Component.

Platform Component Capability – a management resource that represents a Platform Component's capabilities.

Platform Component Requirement – a management resource that represents a requirement on a Platform Component, expressed with attributes that may have value ranges.

Platform Component Template - a management resource that represents an unrealized Platform Component and includes references to metadata for configuring an instance of that Platform Component.

Platform Deployment Package (PDP) - an archive of executable images, dependency descriptions and metadata (Management Resources serialized into a Deployment Plan) that can be used to move an Application and its Components from Platform to Platform, or between an Application Development Environment and a Platform (e.g. a storefront application with component binaries, database images and all the configurations needed to install and run).

Appendix C. Use Cases (Non-Normative)

The following is non-exhaustive list of detailed use cases addressed by this specification.

C.1 Building the Application

C.1.1 Build Application and Package

The Application Developer selects a local Application Development Environment (ADE) and uses it to build an application. The ADE *may* be aware of the Platform Components that are available in the target platform for the application and *may* provide the developer with assistance in consuming these Components. Once built, the Components that make up the application are packaged along with any necessary metadata as part of a deployment package. The output of the ADE is a Platform Deployment Package (PDP) with bundled Component executables (including language specific packages such as .war files) and Metadata included in the deployment plan (including the requirements for Platform Component elements). This is shown in the Figure below.

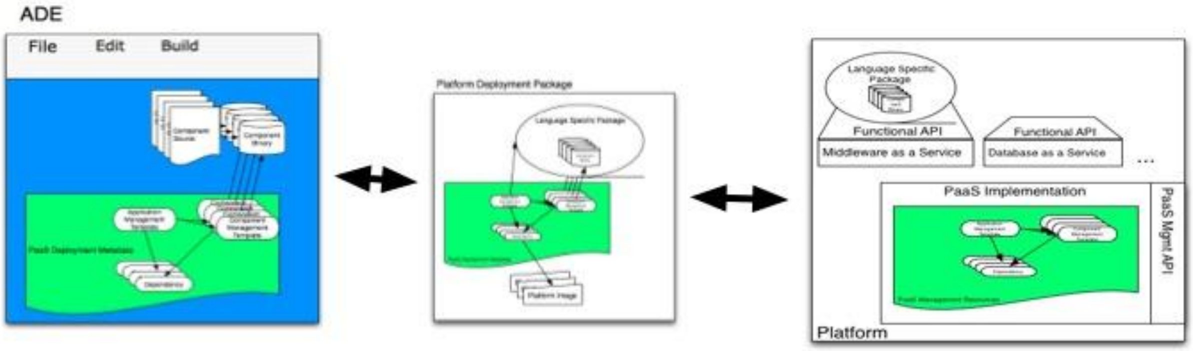

Figure 9 - Platform Deployment

If the platform is using an IaaS infrastructure, there will be additional packages for the deployment of Platform Components virtual machines referenced by the metadata in the PDP as shown below:

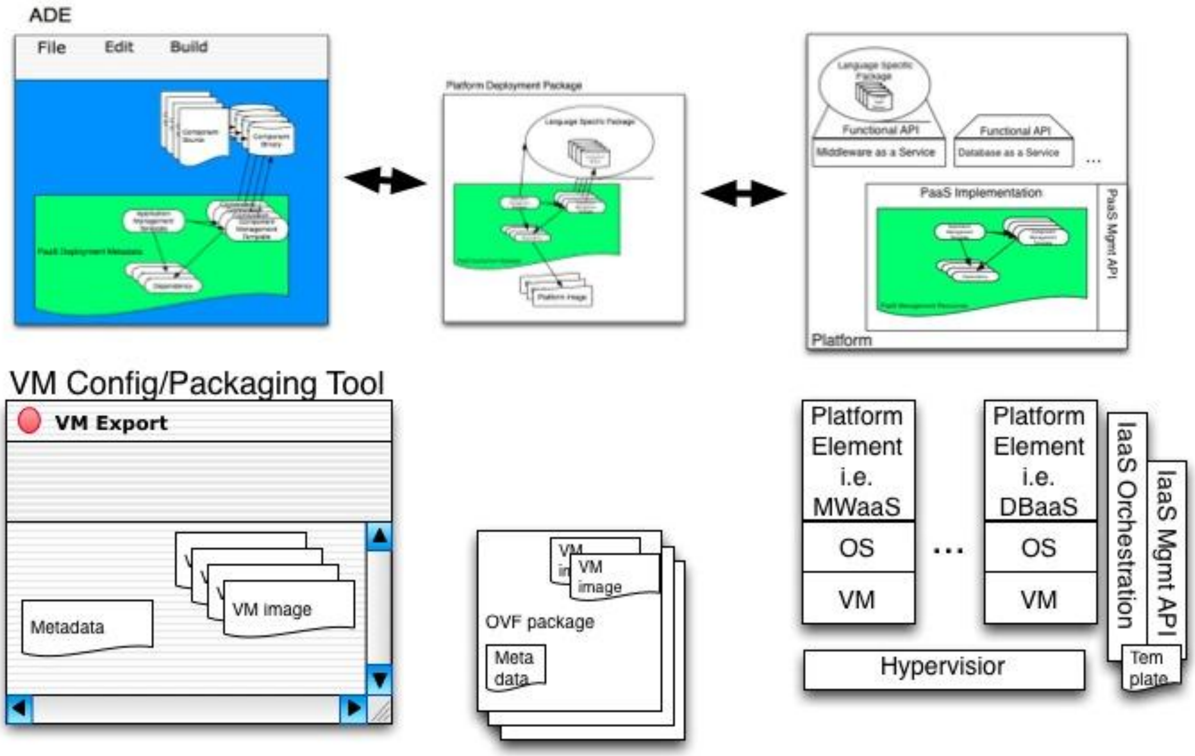

Figure 10 - Platform Deployment with IaaS

This shows the configuration of Platform Components in a virtual machine configuration and packaging tool, which outputs OVF [OVF] packages (one for each Platform Component). These OVF packages are then deployed in an Infrastructure as a Service environment, creating the platform. The PaaS Management API will include extension points for the Platform Components that allow the administrator to invoke the IaaS management API to manage the Platform Components.

C.1.2 Build Application in the Cloud and Optionally Package

This use case is the same as that described in the previous section except for the following:

- The ADE is itself an application that may be deployed on the Platform (see SaaS).
- The ADE is aware of the Platform Components that are available on the Platform for the Application.
- If the target platform for the application is “the same” as the platform that hosts the ADE, the packaging, registering and instantiation process may be simplified.

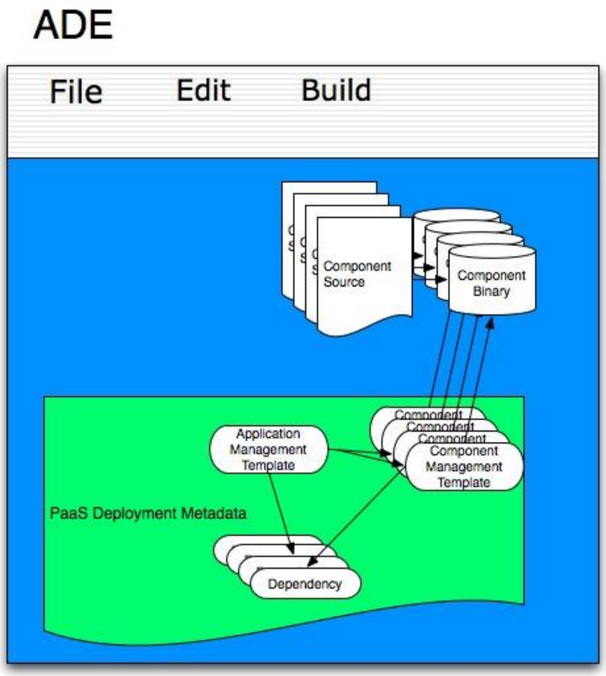

Figure 11 - ADE Development to a Platform

C.2 Deploying and Managing the Application

To deploy an application the Application Administrator shall have an account with the platform provider and access to the required resources. Assume that Assembly Templates, Application Component Templates and Platform Component Templates have already been created in the ADE and serialized into a Platform Deployment Package as shown below:

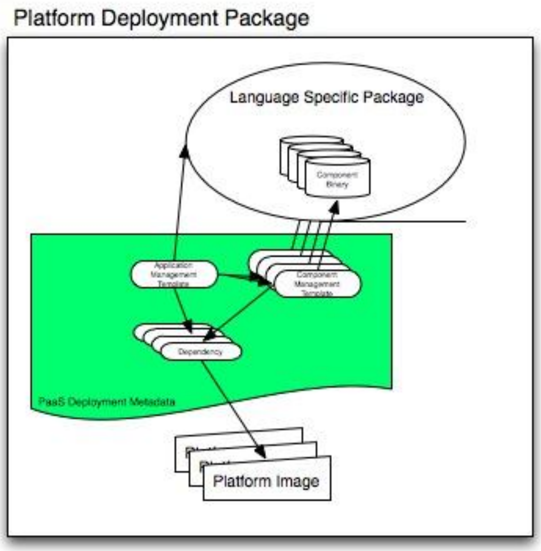
Platform Deployment Package
Language Specific Package
Component Binary
Application Management Template
Component Management Template
Dependency
PaaS Deployment Metadata
Platform Image

Figure 12 - Platform Deployment Package (PDP)

C.2.1 Import Platform Deployment Package

This allows the Application Administrator to import a Platform Deployment Package and that creates the corresponding Management Resources on the Platform. The Platform Deployment Package is unpacked and Assembly Templates, Application Component Templates and their Platform Component Templates are realized in the API as Management Resources and an Assembly Instance can then be created that is ready to run. Any resources that the Components depend on are created as well. The inverse export operation creates a Platform Deployment Package from a given set of Management Resources. The imported Management Resources are shown in the Figure below:

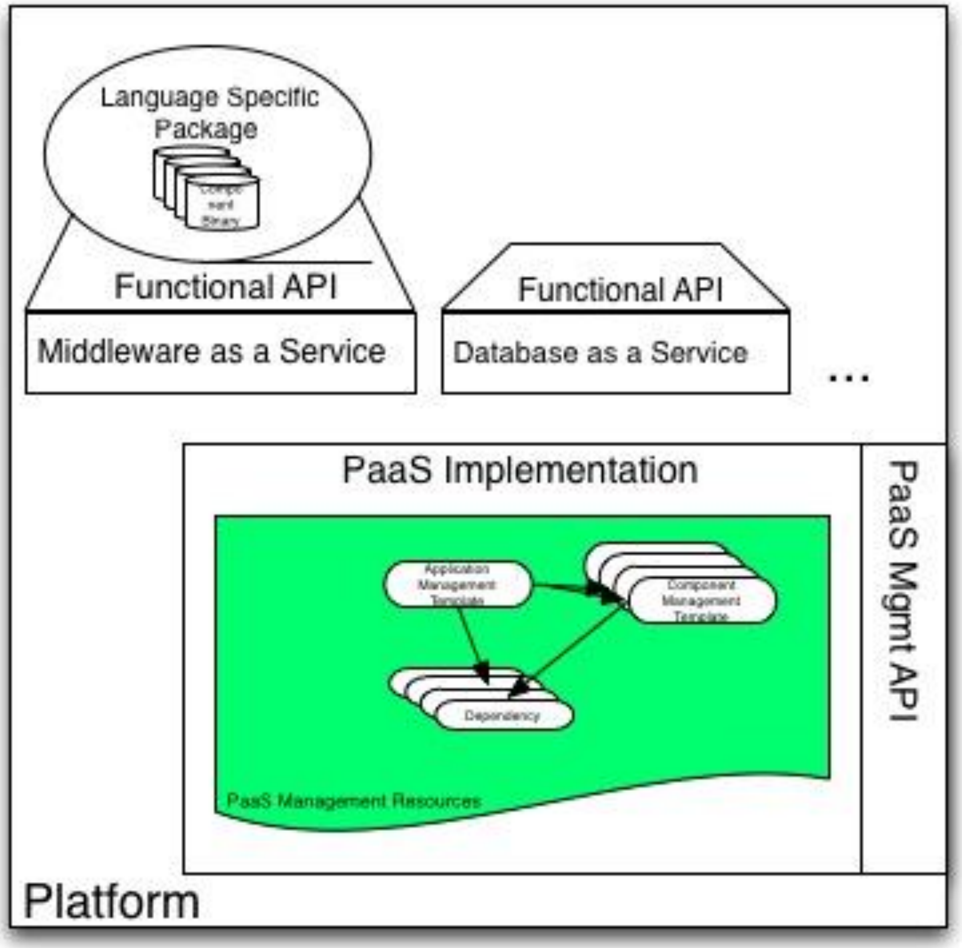

Figure 13 - Imported Platform Deployment Package

C.2.2 Upload Application

Instead of importing a Platform Deployment Package, the PaaS Management API also supports uploading of application files, be it code, configuration, or data files, in full or in part. The typical use case is that of “pushing” incremental updates (“diffs”) as one would when using a version control system such as Git [Git]. However, this upload mechanism is not limited to incremental uploads. It may also be used to upload applications in their entirety.

C.2.3 Run/Stop/Suspend/Snapshot

This allows the Application Administrator to run an Assembly or individual Application Components, stop them, put them into suspension or snapshot them. Assemblies in suspension can be restarted and Assemblies can be restarted from snapshots.

C.2.4 Patch an Application Component Template

Replaces one or more Application Component Templates within an Assembly Template with, usually, updated versions. These updated templates will be used the next time an Assembly Instance is created from the Assembly Template or when already created Assemblies take up the change.

C.2.5 Patch a Created, Deployed or Running Application

This operation replaces one or more Application Component Templates within the Assembly Template associated in a deployed or running Assembly. The updated templates are used the next time the Assembly or an Application Component within the Assembly Instance is deployed.

Appendix D. Example Database Platform Component (Non-Normative)

One important Platform Component that can be provided by many platforms is a database. The following sections illustrate how database components could be provided by extending the model defined in this specification. The material in these sections is non-normative.

D.1 Model

A Database Platform Component provides four sub-classed resources as shown in the below diagram:

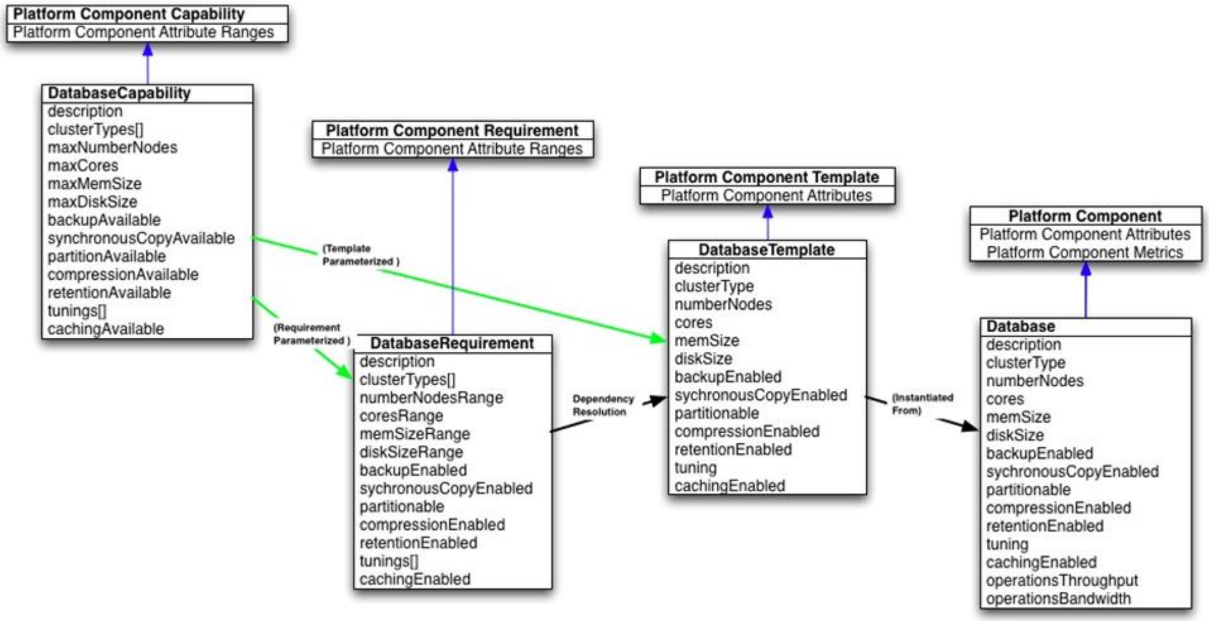

Figure 14 - Database Platform Component Model

D.2 DatabaseCapability

For an Application Administrator, a Database Capability represents the definition of a database platform component and its range of capabilities. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "databaseCapability",
  "description": String,
  "created": Timestamp,
  "tags": [
    String, +
  ], ?
  "definition": String,
  "clusterTypes": [
    String, +
  ],
  "maxNumberNodes": String,
  "maxCores": String,
  "maxMemSize": String,
  "maxDiskSize": String,
  "backupAvailable": Boolean,
  "synchronousCopyAvailable": Boolean,
  "partitionAvailable": Boolean,
  "compressionAvailable": Boolean,
  "retentionAvailable": Boolean,
  "tunings": [
    String, +
  ],
  "cachingAvailable": Boolean,
}
```

Each type of DatabasePlatformComponent implements this class and populates the attributes in the list below.

D.2.1 definition

Type: String

Cardinality: 1

Mutability: mutable

Writeable: false

This attribute contains the definition of the DatabaseCapability represented in some format, such as XML, that contains the metadata necessary for the platform to deploy the Application.

D.2.2 clusterTypes

Type: String[]

Cardinality: 1

Mutability: mutable

Writeable: false

An array of supported cluster types. Values include: “active” and “passive”.

D.2.3 maxNumberNodes

Type: String

Cardinality: 1

Mutability: mutable

Writeable: false

Expresses the maximum number of supported nodes for scaling purposes.

D.2.4 maxCores

Type: String

Cardinality: 1

Mutability: mutable

Writeable: false

Expresses the maximum number of supported cores for scaling purposes.

D.2.5 maxMemSize

Type: String

Cardinality: 1

Mutability: mutable

Writeable: false

Expresses the maximum size of supported memory for an instance.

D.2.6 maxDiskSize

Type: String

Cardinality: 1

Mutability: mutable

Writeable: false

Expresses the limit to the size of a disk for an instance.

D.2.7 backupAvailable

Type: Boolean

Cardinality: 1

Mutability: mutable

Writeable: false

true if backup can be enabled.

D.2.8 synchronousCopyAvailable

Type: Boolean

Cardinality: 1

Mutability: mutable

Writeable: false

true if synchronous copy can be enabled

D.2.9 partitionAvailable

Type: Boolean

Cardinality: 1

Mutability: mutable

Writeable: false

true if partitioning can be enabled

D.2.10 compressionAvailable

Type: Boolean

Cardinality: 1

Mutability: mutable

Writeable: false

true if compression can be enabled

D.2.11 retentionAvailable

Type: Boolean

Cardinality: 1

Mutability: mutable

Writeable: false

true if retention can be enabled

D.2.12 tunings

Type: String[]

Cardinality: 1

Mutability: mutable

Writeable: false

An array of supported tuning types. Values include: “OLAP”, “DataMining”, and “Spatial”

D.2.13 cachingAvailable

Type: Boolean

Cardinality: 1

Mutability: mutable

Writeable: false

true if caching can be enabled

D.3 DatabaseRequirement

For an Application Administrator, a Database Requirement represents an Application Component’s requirements on a database platform component and its required range of capabilities. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "databaseRequirement",
  "description": String,
  "created": Timestamp,
  "tags": [
    String, +
  ], ?
  "definition": String,
  "clusterTypes": [
    String, +
  ],
  "numberNodesRange": String,
  "coreRange": String,
  "memSizeRange": String,
  "diskSizeRange": String,
  "backupEnabled": Boolean,
  "synchronousCopyEnabled": Boolean,
  "partitionEnabled": Boolean,
  "compressionEnabled": Boolean,
  "retentionEnabled": Boolean,
  "tunings": [
    String, +
  ],
  "cachingEnabled": Boolean
}
```

Each type of DatabaseRequirement implements this class and populates the attributes in the list below.

D.3.1 definition

Type: String

Cardinality: 1

Mutability: mutable

Writeable: true

This attribute contains the definition of the DatabaseRequirement represented in some format, such as XML, that contains the metadata necessary for the platform to deploy the Application.

D.3.2 clusterTypes

Type: String[]

Cardinality: 1

Mutability: mutable

Writeable: true

An array of required cluster types. Values include: “active” and “passive”.

D.3.3 numberNodesRange

Type: String

Cardinality: 1

Mutability: mutable

Writeable: true

Expresses the range of the number of nodes for scaling purposes

D.3.4 coreRange

Type: String

Cardinality: 1

Mutability: mutable

Writeable: true

Expresses the required range of number of cores.

D.3.5 memSizeRange

Type: String

Cardinality: 1

Mutability: mutable

Writeable: true

Expresses the range of required memory sizes.

D.3.6 diskSizeRange

Type: String

Cardinality: 1

Mutability: mutable

Writeable: true

Expresses the range if disk sizes required.

D.3.7 backupEnabled

Type: Boolean

Cardinality: 1

Mutability: mutable

Writeable: true

true if backup is required.

D.3.8 synchronousCopyEnabled

Type: Boolean

Cardinality: 1

Mutability: mutable

Writeable: false

true if synchronous copy is required.

D.3.9 partitionEnabled

Type: Boolean

Cardinality: 1

Mutability: mutable

Writeable: true

true if partitioning is required.

D.3.10 compressionEnabled

Type: Boolean

Cardinality: 1

Mutability: mutable

Writeable: true

true if compression is required.

D.3.11 retentionEnabled

Type: Boolean

Cardinality: 1

Mutability: mutable

Writeable: true

true if retention is required.

D.3.12 tunings

Type: String[]

Cardinality: 1

Mutability: mutable

Writeable: true

An array of required tuning types. Values include: “OLAP”, “DataMining” and “Spatial”.

D.3.13 cachingEnabled

Type: Boolean

Cardinality: 1

Mutability: mutable

Writeable: true

true if caching is required.

D.4 DatabaseTemplate

A Database Template represents the desired configuration of a Database Platform Component with specific values for the component capabilities. The specified value for each component attribute shall be in the range defined in the corresponding Database Capability. Some PaaS offerings might only offer a fixed number of Database Template instances that cannot be modified (read-only, no new instances) indicating pre-tuned and configured pools of database resources from which these draw. Other PaaS offerings may allow newly created Database Template instances with values in any combination. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "databaseTemplate",
  "description": String,
  "created": Timestamp,
  "tags": [
    String, +
  ], ?
  "definition": String,
  "clusterType": String,
  "numberNodes": String,
  "cores": String,
  "memSize": String,
  "diskSize": String,
  "backupEnabled": Boolean,
  "synchronousCopyEnabled": Boolean,
  "partitionEnabled": Boolean,
  "compressionEnabled": Boolean,
  "retentionEnabled": Boolean,
  "tuning": String,
  "cachingEnabled": Boolean
}
```

Each type of DatabaseTemplate implements this class and populates the attributes in the list below.

D.4.1 definition

Type: String

Cardinality: 1

Mutability: mutable

Writeable: false

This attribute contains the definition of the DatabaseTemplate represented in some format, such as XML, that contains the metadata necessary for the platform to deploy the Application.

D.4.2 clusterType

Type: String

Cardinality: 1

Mutability: mutable

Writeable: false

The desired cluster type. Values include: “active” and “passive”

D.4.3 numberNodes

Type: String

Cardinality: 1

Mutability: mutable

Writeable: false

The desired number of nodes for an instance

D.4.4 cores

Type: String

Cardinality: 1

Mutability: mutable

Writeable: false

The desired number of cores for an instance

D.4.5 memSize

Type: String

Cardinality: 1

Mutability: mutable

Writeable: false

The desired size of memory for an instance

D.4.6 diskSize

Type: String

Cardinality: 1

Mutability: mutable

Writeable: false

The desired size of a disk for an instance.

D.4.7 backupEnabled

Type: Boolean

Cardinality: 1

Mutability: mutable

Writeable: false

true if backup is desired for an instance.

D.4.8 synchronousCopyEnabled

Type: Boolean

Cardinality: 1

Mutability: mutable

Writeable: false

true if synchronous copy is desired for an instance.

D.4.9 partitionEnabled

Type: Boolean

Cardinality: 1

Mutability: mutable

Writeable: false

true if partitioning is desired for an instance

D.4.10 compressionEnabled

Type: Boolean

Cardinality: 1

Mutability: mutable

Writeable: false

true if compression is desired for an instance

D.4.11 retentionEnabled

Type: Boolean

Cardinality: 1

Mutability: mutable

Writeable: false

true if retention is desired for an instance

D.4.12 tuning

Type: String

Cardinality: 1

Mutability: mutable

Writeable: false

The desired tuning types. Values include: “OLAP”, “DataMining” and “Spatial”

D.4.13 cachingEnabled

Type: Boolean

Cardinality: 1

Mutability: mutable

Writeable: false

true if caching is desired for an instance

D.5 Database

A Database represents the runtime instance of a Database Template and its configuration of component attributes as well as metrics associated with those attributes. Each Application’s use of a Database is represented by an instance of this resource. This resource has the following, general representation:

```
{
  "uri": URI,
  "name": String,
  "type": "database",
  "description": String,
  "created": Timestamp,
  "tags": [
    String, +
  ], ?
  "definition": String,
  "externalManagementResource": String,
  "clusterType": String,
  "numberNodes": String,
  "cores": String,
  "memSize": String,
  "diskSize": String,
  "backupEnabled": Boolean,
  "synchronousCopyEnabled": Boolean,
  "partitionEnabled": Boolean,
  "compressionEnabled": Boolean,
  "retentionEnabled": Boolean,
  "tuning": String,
  "cachingEnabled": Boolean,
  "operationsThroughput": String,
  "operationsBandwidth": String,
  "resourceState": {
    "state": String,
    "progress": String, ?
    "messages": [
      { Message }, +
    ] ?
  }
}
```

Each type of Database implements this class and populates the attributes in the list below.

D.5.1 definition

Type: String

Cardinality: 1

Mutability: immutable

This attribute contains the definition of the Database represented in some format, such as XML, that contains the metadata necessary for the platform to deploy the Application.

D.5.2 externalManagementResource

Type: URI

Cardinality: 0..1

Mutability: immutable

A URI to an external management interface to manage this platform component (such as an IaaS API to manage the virtual machines that make up this database). This is platform dependent and requires external documentation to understand its meaning.

D.5.3 clusterType

Type: String

Cardinality: 1

Mutability: immutable

The actual cluster type. Values include: “active” and “passive”

D.5.4 numberNodes

Type: String

Cardinality: 1

Mutability: immutable

The actual number of nodes for an instance.

D.5.5 cores

Type: String

Cardinality: 1

Mutability: immutable

The actual number of cores for an instance.

D.5.6 memSize

Type: String

Cardinality: 1

Mutability: immutable

The actual size of memory for an instance.

D.5.7 diskSize

Type: String

Cardinality: 1

Mutability: immutable

The actual size of a disk for an instance.

D.5.8 backupEnabled

Type: Boolean

Cardinality: 1

Mutability: immutable

true if backup is enabled for this instance.

D.5.9 synchronousCopyEnabled

Type: Boolean

Cardinality: 1

Mutability: immutable

true if synchronous copy is enabled for this instance.

D.5.10 partitionEnabled

Type: Boolean

Cardinality: 1

Mutability: immutable

true if partitioning is enabled for this instance.

D.5.11 compressionEnabled

Type: Boolean

Cardinality: 1

Mutability: immutable

true if compression is enabled for this instance.

D.5.12 retentionEnabled

Type: Boolean

Cardinality: 1

Mutability: immutable

true if retention is enabled for this instance.

D.5.13 tuning

Type: String

Cardinality: 1

Mutability: immutable

The actual tuning type of the database instance. Values include: “OLAP”, “DataMining” and “Spatial”

D.5.14 cachingEnabled

Type: Boolean

Cardinality: 1

Mutability: immutable

true if caching is enabled for this instance.

D.5.15 operationsThroughput

Type: String

Cardinality: 1

Mutability: mutable

Writeable: false

The billable operations per second.

D.5.16 operationsBandwidth

Type: String

Cardinality: 1

Mutability: mutable

Writeable: false

The billable MegaBytes per second.

D.5.17 resourceState

Type: ResourceState

Cardinality: 1

Mutability: mutable

Writeable: false

This attribute expresses state of the database instance within its lifecycle.

Appendix E. Implementation Considerations (Non-Normative)

E.1 Types of Platform Deployments

Platforms can be built with resources that provide isolation at different levels. For example, each Platform Component (e.g. application server, database) can sit atop a stack of virtual machine, operating system, and component, as seen below:

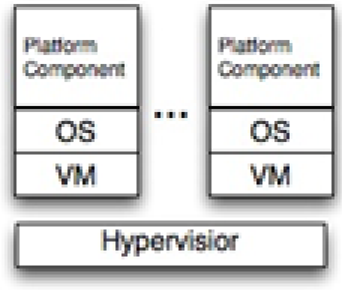

Figure 15 - Platform Component Virtual Machine Isolation

The point of isolation in the above is the virtual machine.

The level at which isolation is provided has profound effects upon the way in which Platform Components are managed and secured. In the above example, each consumer is provided with a dedicated VM, OS, and Platform Component. It is likely (though not certain) that, in the above model, the Consumer will perform both the Application Administrator and the Platform Administrator roles - managing not only their application but also the underlying infrastructure including, potentially, the operating system.

Platform Components can also be deployed without a hypervisor, and instead use multiple Platform Components running on a single OS as shown below:

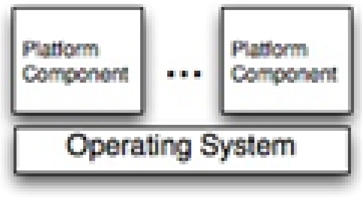

Figure 16 - Platform Component Operating System Isolation

The point of isolation in the above is the operating system.

Although the consumer may still perform both the Application Administrator and some portions of the Platform Administrator roles, for obvious security reasons consumers cannot be allowed control of the operating system as a single OS can be shared by multiple consumers.

Lastly, the Platform Component container itself can offer virtualize instances of the Platform Component to each consumer and isolate multiple consumers from each other as shown below:

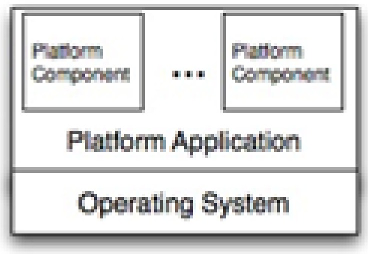

Figure 17 - Platform Component Container Isolation

The point of isolation in the above is the Platform Component container.

In the above case the consumer can only act as an Application Administrator. The tasks of configuring and managing the Platform Component container (tasks assigned to the Platform Administrator) can only be performed by the cloud provider.

This PaaS Management API is agnostic with respect to the above deployment models as it deals only with the management of an Application's dependencies on various Platform Components and not with the lifecycle of the Platform Components themselves. The API may be extended to include the management of Platform Component containers, Platform Components, operating systems, and other infrastructural elements.

E.2 Scaling

The PaaS Management API assumes that applications are organized architecturally in tiers, some of which are hosted on the platform, while others may be external to the platform (and hence unmanaged). Each tier may consist of one or more components. For example, the “application tier” of a PHP application may be implemented by a mod_php component combined with an Apache component. Components in turn can be either provided by the platform or by the application itself.

Conceptually, scaling operations affect individual application tiers. In practice however, since tiers are composed of Application and Platform Components, this means they affect Components. In addition, scaling operations are also specific to each Component and may be subject to explicit or implicit restrictions and policies imposed by the Component on both, scaling operations and grouping. For instance, a Component such as a database may be inherently non-replicable. Conversely, a Component such as PHP module may need to be in the same group as its hosting Apache instance because it needs to be dynamically loaded into the latter--even though it can scale out without limit.

Individual Components may define events and policies for implementing auto-scaling. For instance, a multithreaded webserver component may trigger a scale-up if the number of idle worker threads drops below a given threshold. Or a MongoDB component may impose a policy that scaling may only happen in increments of 3 nodes.

Scaling operations may also be performed directly on the Application Component's Platform Component resources. This is known as manual scaling and generally results in a new reservation being created in the platform's auto-scaling setup. Other forms of scaling may require changes to how the Assembly Template uses Application Component Templates and Platform Component Templates. The actual scaling architecture for a Platform Component is outside the scope of this specification.

Appendix F. Revision History

Revision	Date	Editor	Changes Made
1	2012-12-04	Anish Karmarkar	Applied OASIS template to version 1.0
2	2012-12-18	Anish Karmarkar	Included resolutions for issues 7, 12, 13, 15, 24, 33.