Red Hat Reference Architecture Series

Red Hat CloudForms

Architectural Overview

Steve Reichard, RHCE
Principal Software Engineer

Vinny Valdez, RHCA
Principal Software Engineer

Version 1.0
May 2011

redhat.

1801 Varsity Drive™
Raleigh NC 27606-2072 USA
Phone: +1 919 754 3700
Phone: 888 733 4281
Fax: +1 919 754 3701
PO Box 13588
Research Triangle Park NC 27709 USA

The GPG fingerprint of the security@redhat.com key is:
CA 20 86 86 2B D6 9D FC 65 F6 EC C4 21 91 80 CD DB 42 A6 0E

Send feedback to us at refarch-feedback@redhat.com

Table of Contents

1 Executive Summary

Cloud computing is quickly becoming the platform of choice for users and businesses that want to reduce operating expenses and be able to scale resources rapidly. Eased automation, flexibility, mobility, resiliency, and redundancy are several other advantages of moving resources to the cloud.

Even though cloud computing is in the early stages, there are different types of cloud solutions available to businesses today. On-premise private clouds allow businesses to take advantage of cloud technologies while remaining on a private network. Public clouds allow businesses to make resources available to external consumers. Hybrid clouds allow the best of both public and private cloud computing models.

In this paper the concepts that comprise an Infrastructure as a Service (IaaS) Cloud are discussed first at a high-level conceptual view, then broken down into actual products, an example application deployed and each step of this use case broken down. The reader concludes with complete knowledge of a Red Hat CloudForms, how to deploy applications, and how Red Hat is uniquely positioned to be the authoritative interface of all Private, Hybrid, Community, and Public Clouds.

2 Red Hat Cloud Strategy

Red Hat's cloud vision is unlike that of any other IT vendor. We recognize that your IT infrastructure is - and will continue to be - composed of pieces from many different hardware and software vendors. We let you use and manage these diverse assets as one cloud, enabling cloud to be an evolution, not a revolution or a monolithic stack locked to the technology roadmap and business practices of a single vendor.

When you choose Red Hat for your cloud, you get:

- The most comprehensive solutions for clouds - both private and public.
- Consistent enterprise-class environments that bridge the physical and virtual world, inside the data center and public clouds.
- Strategic flexibility without lock-in.
- Better infrastructure, designed specifically for multi-tenant clouds.
- Industry-leading ecosystem that makes cloud usable, accessible, and safe.

In a market full of hype, Red Hat makes the cloud real and compelling. Today.

Infrastructure-as-a-Service (IaaS) is about delivering infrastructure—which is to say resources like compute, storage, and networking - to users. Many organizations are getting into cloud computing by building an on-premise IaaS cloud. They may want to keep the option to bridge from private to public clouds, a.k.a. hybrid clouds, open. But they are often concerned about using public clouds for important business applications, whether because of specific regulatory or audit issues or just because they are wary of adding a new element of potential risk to their IT governance.

The IaaS term is widely used. Dig deeper though, and you find that not all IaaS solutions are created equal. For example, the typical IaaS manages the cloud but does not manage the life-cycle of applications running in the cloud - even though the cloud should be in support of the application and not the other way around.

Furthermore, this typical IaaS makes the naive assumption that organizations are looking to start over with a brand new infrastructure as they move into cloud computing. Nothing could be further from the truth. Organizations want to join the cloud computing revolution, but they want to do it in an evolutionary way that leverages and extends their existing infrastructure and maintains portability across different technology stacks and providers.

Red Hat CloudForms is different. Like others, it allows organizations to build and manage their own IaaS cloud for internal consumption. But it does far more. It integrates with existing products and technologies, including physical servers and virtualization platforms from other vendors, to provide the easiest on-ramp to an on-premise cloud. It manages applications throughout their life-cycle rather than just the virtual machine containers in which they sit.

In short, Red Hat CloudForms is Infrastructure-as-a-Service done right.

Previously, Red Hat has shown that Red Hat Cloud Foundations provided the necessary technologies needed for the cloud infrastructure. CloudForms is the next generation of technologies which builds upon Red Hat Cloud Foundations to provide a complete IaaS cloud solution.

CloudForms provides the IaaS infrastructure through:

- Application Lifecycle Management
- Compute Resource Management
- Infrastructure Services

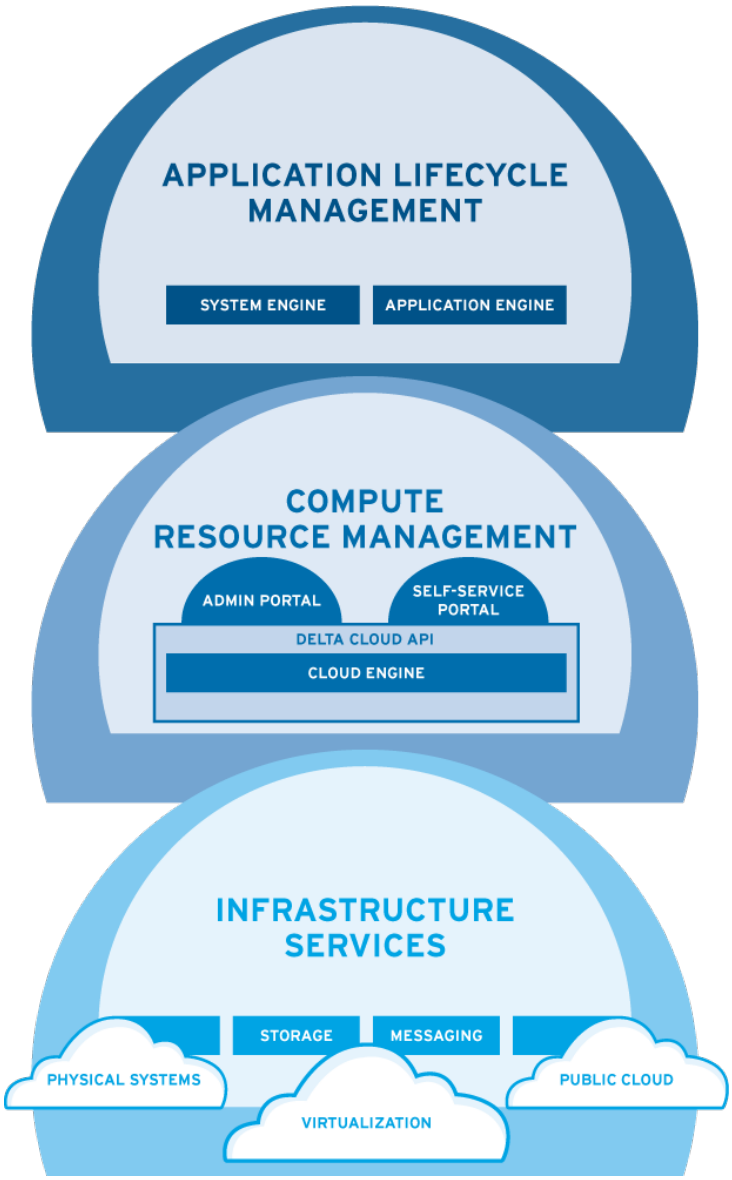

Illustration 2-1: Red Hat CloudForms

CloudForms is delivering technologies in the following areas:

- CloudForms Cloud Engine
- CloudForms Application Engine
- CloudForms System Engine
- CloudForms Cloud Services

2.1 CloudForms Cloud Engine

CloudForms Cloud Engine is responsible for all cloud resource management. It enables creating cloud resources, managing policies and work-flows around those resources, and governing access and permissions for the resources. Quotas, quality-of-service, and security policies are also under administrator control. End-users can then provision resources through a self-service web interface subject to policy constraints.

The CloudForms Cloud Engine provides functionality in the following areas:

- Cloud Interface
- Application Lifecycle Management

2.2 CloudForms Application Engine

The CloudForms Application Engine provides template-based management of applications. One or more templates can then be aggregated or associated and given the operational parameters and configurations needed to boot, initialize, and provide the defined services. Application Engine therefore explicitly handles applications that span multiple virtual machines, a common occurrence.

The CloudForms Application Engine provides functionality in the following areas:

- Application Description Generation
- Image Lifecycle Management

2.3 CloudForms System Engine

CloudForms System Engine operationally manages running systems across physical, virtual, and cloud environments. It provides continuous compliance of content and configurations (as well as Red Hat entitlements) consistent with the definitions used by Application Engine. It builds on top of Application Engine's functionality by monitoring and updating while systems are running on an ongoing basis. System Engine also works in concert with Application Engine by supplying content that it can use to build images and deploy.

The CloudForms System Engine provides functionality in the area of Content Provision Management.

2.4 CloudForms Cloud Services

CloudForms Cloud Services provide the consistent functionality across varied cloud environments for a wide variety of service such as storage, availability, etc..

3 Red Hat Cloud Solution Architecture

In this section the cloud definitions as currently defined by NIST are provided, the mapping of Red Hat CloudForms to the definitions are proposed, and a high level look that the Red Hat CloudForms Solution Architecture is described.

3.1 The Cloud as viewed by NIST

NIST[1] (National Institute of Standards and Technology) has produced several documents that supply definitions and provide common terminology for the cloud paradigm that are reiterated in the remainder of this section.

- NIST Definition of Cloud Computing[2]
- NIST Cloud Computing Reference Architecture, v1.0[3]

3.1.1 Definition of Cloud Computing

Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models.

3.1.2 Essential Characteristics

On-demand self-service:

A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service's provider. Broad network access capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, laptops, and PDAs).

Resource pooling:

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

Rapid elasticity:

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

Measured Service:

Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported providing transparency for both the provider and consumer of the utilized service.

3.1.3 Service Models

Cloud Infrastructure as a Service (IaaS)

The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly limited control of select networking components (e.g., host firewalls).

Cloud Platform as a Service (PaaS)

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

Cloud Software as a Service (SaaS)

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

3.1.4 Deployment Models

Private cloud:

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

Community cloud:

The cloud infrastructure is shared by several organizations and supports a specific community that has shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be managed by the organizations or a third party and may exist on premise or off premise.

Public cloud:

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

Hybrid cloud:

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

3.1.5 Cloud Actors

The following are some of the Cloud Actors from the NIST Cloud Model.

Cloud Consumer

Person or organization that maintains a business relationship with, and uses service from, Cloud Providers.

Cloud consumers are categorized into three groups, based on their different application/usage scenarios as listed in **Table 3-1: Cloud Consumer Activities**.

Consumer Type	Major Activities
IaaS	Creates/installs, manages and monitors services for IT infrastructure operations.
PaaS	Develops, tests, deploys and manages applications in a cloud environment.
SaaS	Uses application/service for business process operations

Table 3-1: Cloud Consumer Activities

Cloud Provider

Person, organization or entity responsible for making a service available to Cloud Consumers.

The providers perform different tasks for different service types, which are listed in **Table 3-2: Cloud Provider Activities**.

Provider Type	Major Activities
IaaS	Provisions and manages the physical processing, storage, networking and the hosting environment and cloud infrastructure for IaaS consumers.
PaaS	Provisions and manages cloud infrastructure and middleware for the platform consumers; provides development, deployment and administration tools to platform consumers.
SaaS	Installs, manages, maintains and supports the software application on a cloud infrastructure.

Table 3-2: Cloud Provider Activities

The activities of cloud providers can be grouped into the following perspectives: Service Deployment, Service Orchestration, Cloud Service Management, Security, and Privacy.

Service Deployment refers to the cloud infrastructure operation as related to the deployment models: Private cloud, Community cloud, Public cloud, Hybrid cloud.

Service Orchestration refers to the arrangement, coordination and management of cloud infrastructure to provide different cloud services to meet IT and business requirements. The three conceptual layers of a generalized cloud environment: Service Layer, Resource Abstraction and Control Layer, and Physical Resource Layer.

As depicted in the following illustration, Cloud Service Management includes all the service-related functions that are necessary for the management and operations of those services required by or proposed to Cloud Consumers. A cloud provider performs the following functions to support cloud service management: Business Support, Provisioning/Configuration, and Portability/Interoperability.

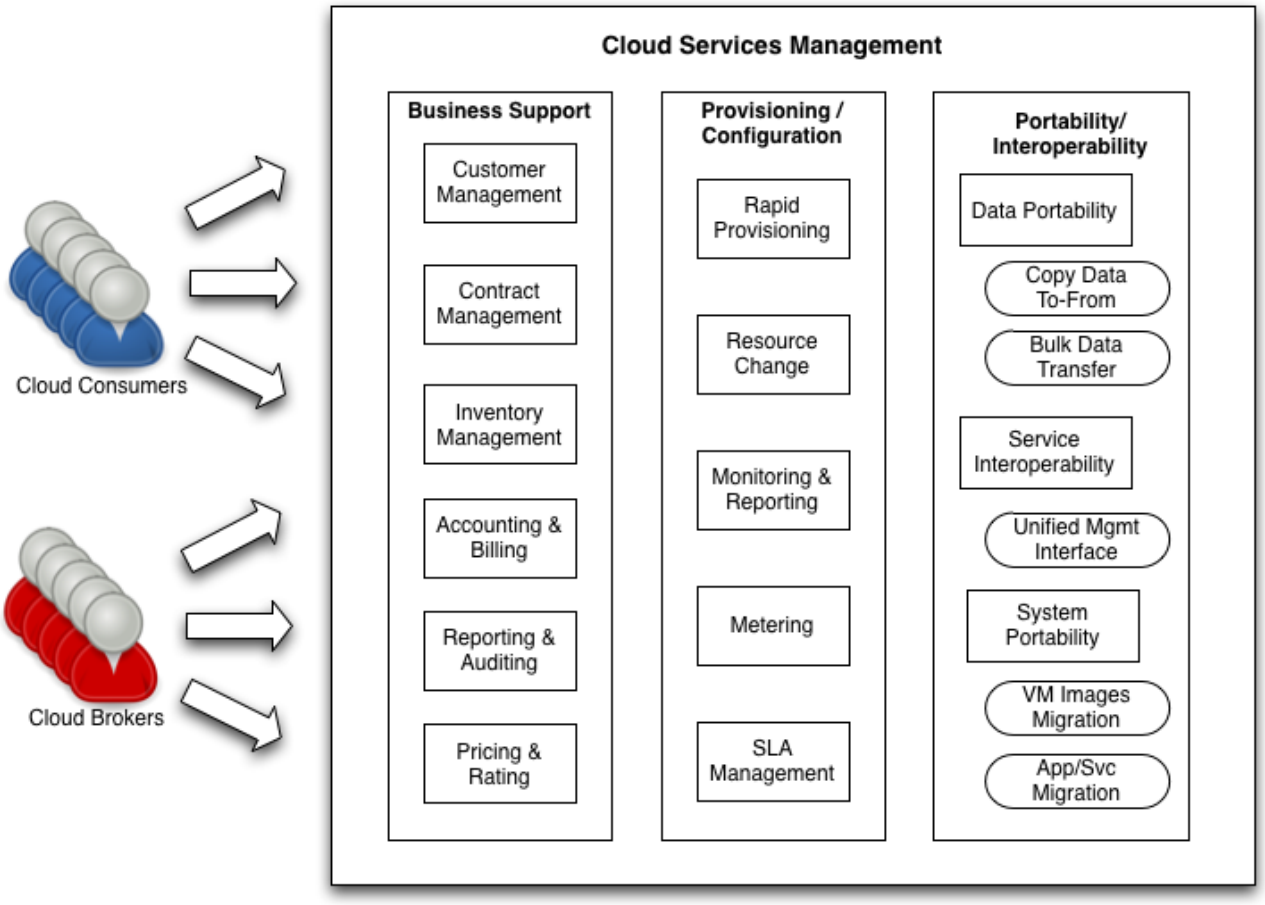

Illustration 3-1: Cloud Providers – Cloud Service Management

The following aspects of Security must be managed in the cloud: Authentication and Authorization, Availability, Confidentiality, Identity Management, Integrity, Security Monitoring & Incident Response, and Security Policy Management.

The goal of Privacy in the cloud is to protect the assured, proper, and consistent collection, processing, communication, use and disposition of personal information (PI) and personally identifiable information (PII) in the cloud.

Cloud Broker

An entity that manages the use, performance and delivery of cloud services, and negotiates relationships between Cloud Providers and Cloud Consumers.

Three major services provided by Cloud Brokers:

Service Intermediation:

A cloud broker enhances a given service by improving some specific capability and provides the value-added service to Cloud Consumers.

Service Aggregation:

A cloud broker combines and integrates multiple services into one or more new services. The broker will provide data integration and ensure the secure data movement between Cloud Consumer and multiple cloud providers.

Service Arbitrage:

Service Arbitrage is similar to service aggregation, with the difference in that the services being aggregated are not fixed. Service arbitrage allows flexible and opportunistic choices for the broker. For example, the cloud broker can use a credit-scoring service and select the best score from multiple scoring agencies.

3.2 Red Hat CloudForms and the NIST model

Red Hat CloudForms does not fit as a single actor in the NIST model. By itself, Red Hat CloudForms is not a NIST defined Cloud Provider. Where a NIST defined Cloud Provider provides the underlying hosting environment such as virtual machines, Red Hat CloudForms does not. Rather, it extends the Cloud Provider's Cloud Service Management support and facilitates Service Deployment and Service Orchestration. The illustration below shows the standard NIST Cloud Provider without Red Hat CloudForms.

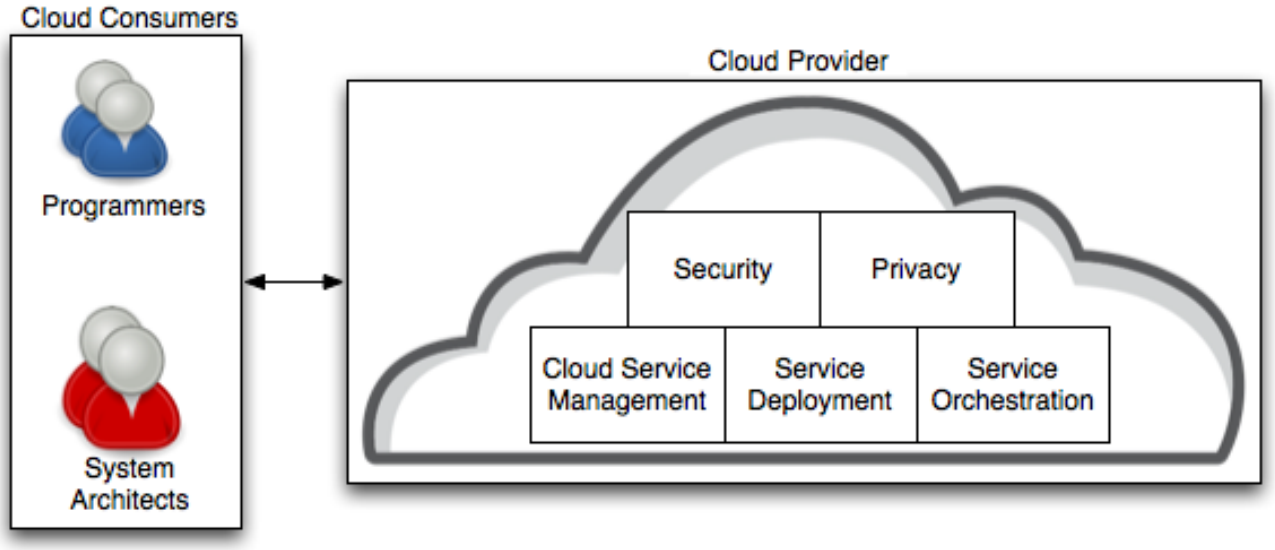

Illustration 3-2: NIST Cloud Provider

Red Hat CloudForms also provides much more functionality than a NIST defined Cloud Broker. A Cloud Broker merely redirects the Cloud Consumer to existing cloud providers as pictured here.

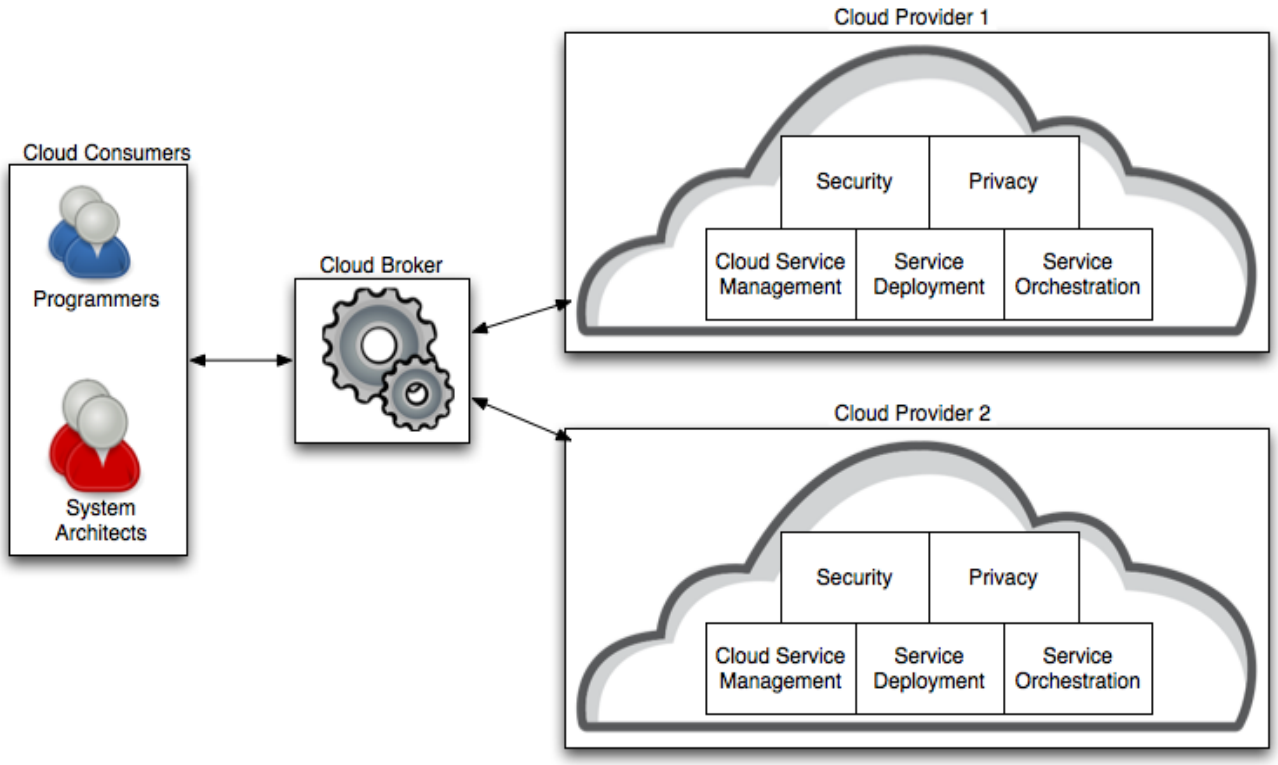

Illustration 3-3: NIST Cloud Broker

Red Hat CloudForms, however, extends a Red Hat Certified Cloud Provider's features, especially those related to Cloud Service Management. In particular,the portability/interoperability functionality is increased with the features that are inherent in Red Hat CloudForms, and further facilitate all requests from the Cloud Consumers. Other areas may also see increased functionality and benefit from Red Hat CloudForms' abstraction being able to provide a single multipurpose interface. The following illustration represents Red Hat CloudForms extending a Cloud Provider's functionality.

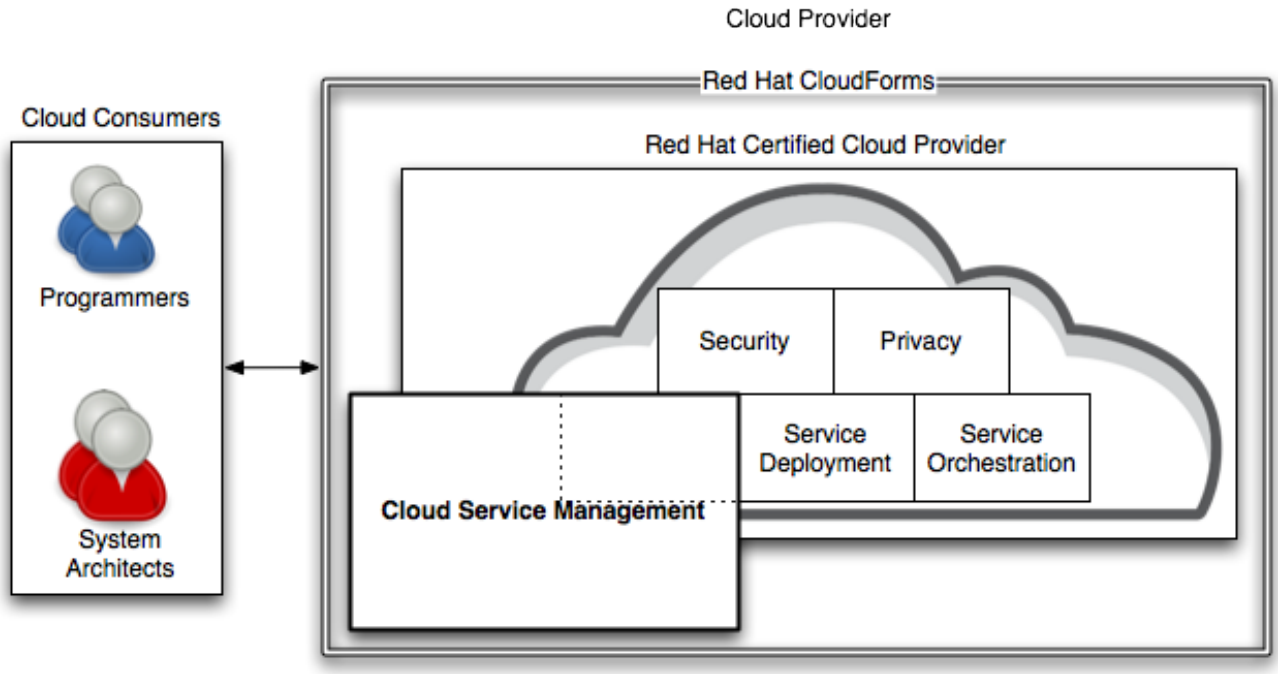

Illustration 3-4: Red Hat CloudForms Extends Certified Clouds

When combined with a virtualization environment, grid deployment, or bare-metal farm, missing essential cloud characteristics are provided by Red Hat CloudForms. The hosted environment is transformed into a functional cloud provider by the sharing of the Cloud Service Management functionality between the hosting environment and Red Hat CloudForms, as portrayed below**.**

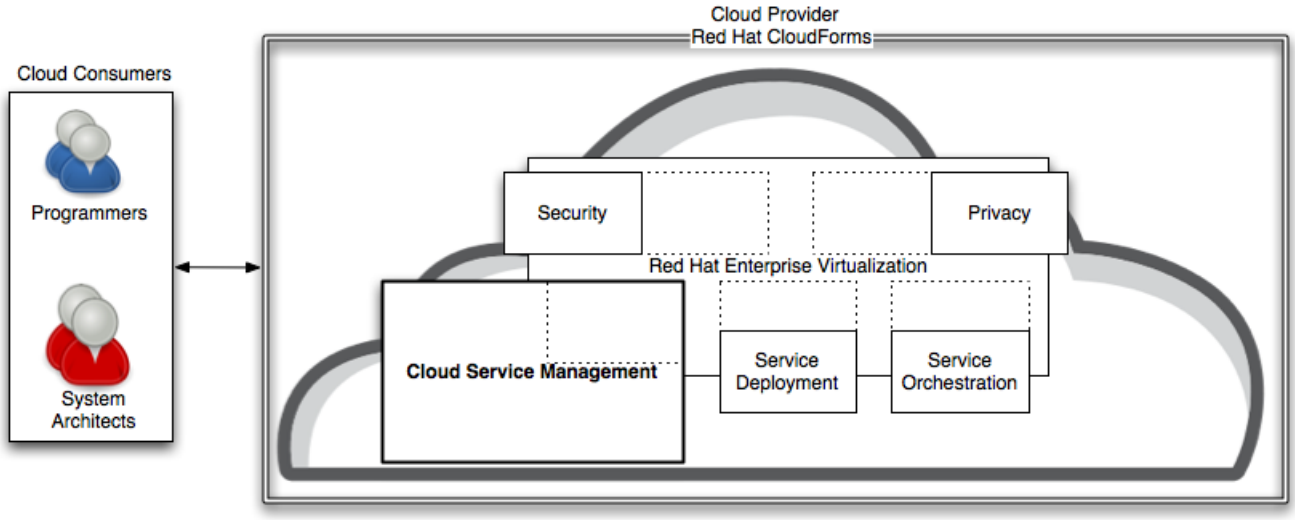

Illustration 3-5: Red Hat CloudForms & Hosted Environment Cloud

Red Hat CloudForms abstraction capabilities allow it to perform more than the functionality of a Service Aggregation Broker, by providing consistent features, content, and services across supported environments. The ability to control deployments into any certified cloud provider results in consistent cross-cloud views of content. The next illustration displays how Red Hat CloudForms makes this possible.

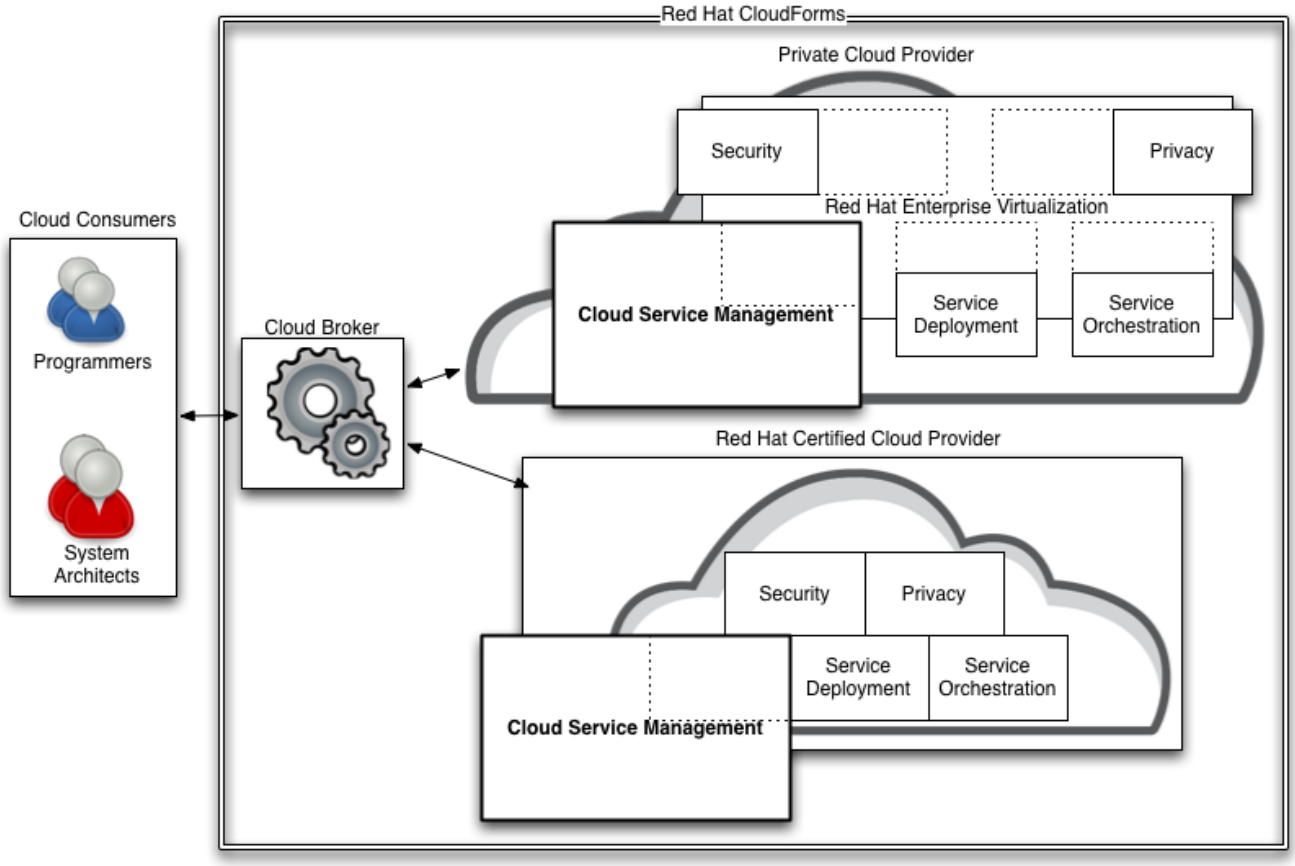

Illustration 3-6: Red Hat CloudForms Provides Multi-Cloud Interoperability

3.3 High Level Functional Areas

The high level functional areas of Red Hat CloudForms are:

- Cloud Interface
- Content Provision Management
- Application Description Generation
- Image Lifecycle Management
- Application Lifecycle Management
- Cloud Services (optional)

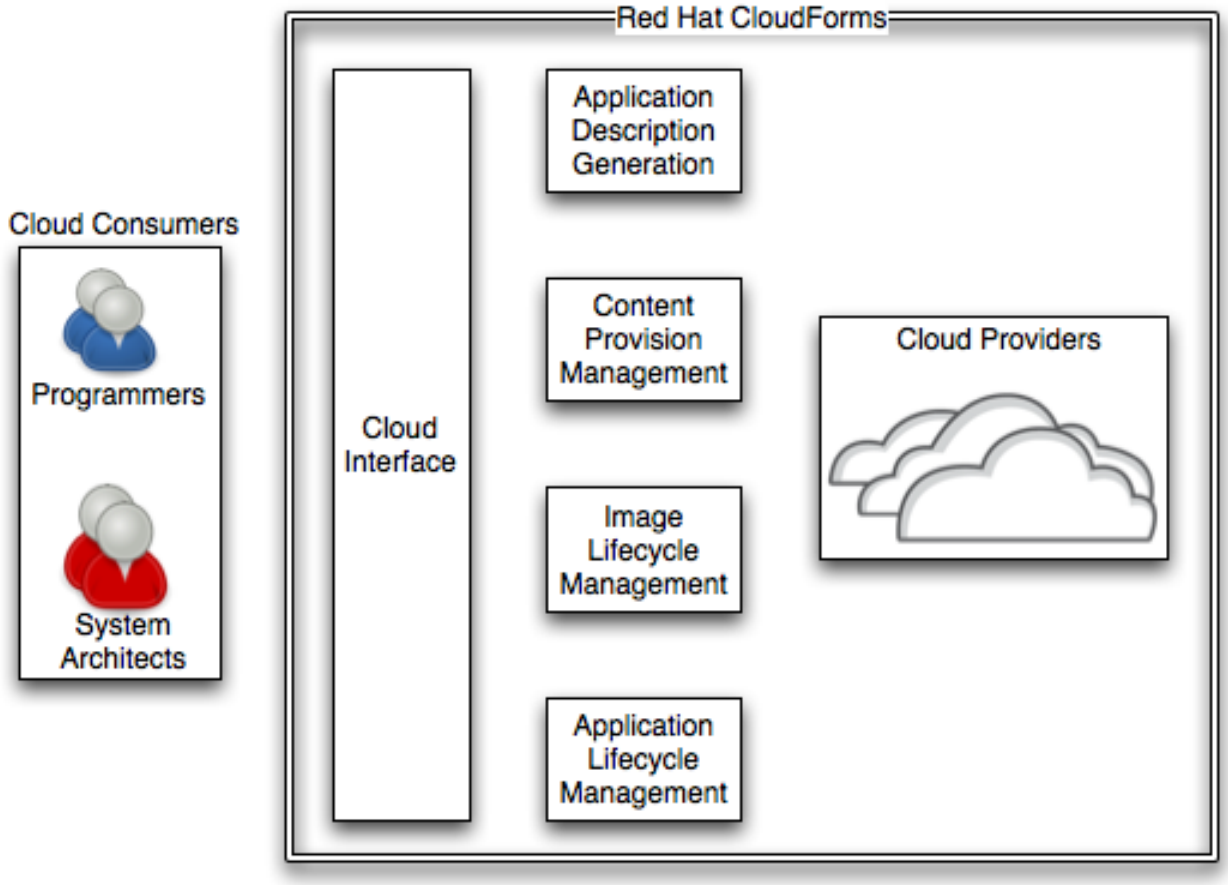

Illustration 3-7: Functional Overview

The *Cloud Interface* provides the Cloud Consumer a central point of interaction for defining, deploying, reporting, and managing numerous cloud applications on potentially many public and private cloud providers. The cloud interface is multi-tenant and provides multiple level administration capabilities.

Content Provision Management supplies content (as well as Red Hat entitlements) to other functional areas and provides configuration compliance and software modifications for running instances.

Application Description Generation allows the Cloud Consumer to define their entire application deployment, which is stored in XML format. This XML is used to build and configure the application in various cloud provider environments.

Image Lifecycle Management controls the creation and management of the images used in deploying the Cloud Consumer's application. *Image Lifecycle Management* uses the XML definitions to create the images required and propagate create images to the various targeted Cloud Providers.

Application Lifecycle Management is used to control and monitor the state of Cloud Consumer applications. This functionality includes resource management, quota enforcement, policy enforcement, application instantiation, configuration controller, etc.

Cloud Services are add-ons to a cloud deployment that ensures consistent functionality at various cloud providers. The following is a list of functional areas some planned services provide:

- Archival Storage
- Replicated Reliable File Systems
- Messaging
- Cloud ID Management
- Availability Monitoring/High Availability

3.3.1 Cloud Interface

When a Cloud Consumer engages Red Hat CloudForms, the *Cloud Interface* is the primary point of interaction that the Cloud Consumer uses to initiate activities, from administration duties, gathering reports on various resources, to defining and controlling an application deployment into a cloud. The following illustration summarizes these functions.

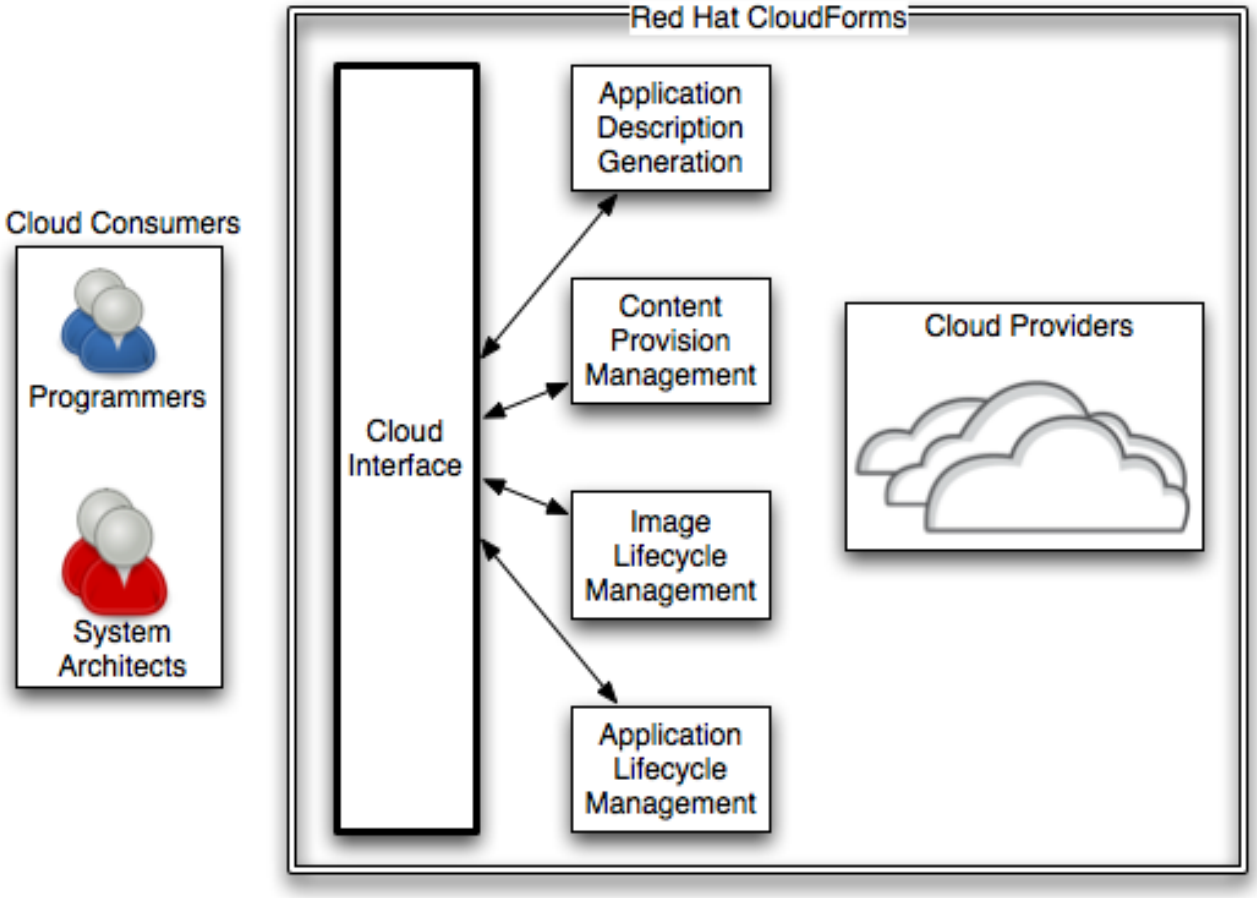

Illustration 3-8: Cloud Interface

3.3.2 Content Provision Management

Content Provision Management provides software to the other functional areas, manages software repositories (from standard content sources such as Red Hat Network, uploaded self-supplied collections, ISOs, etc), and applies configuration compliance and software modifications for running instances. The illustration below depicts its interaction with the other functional areas.

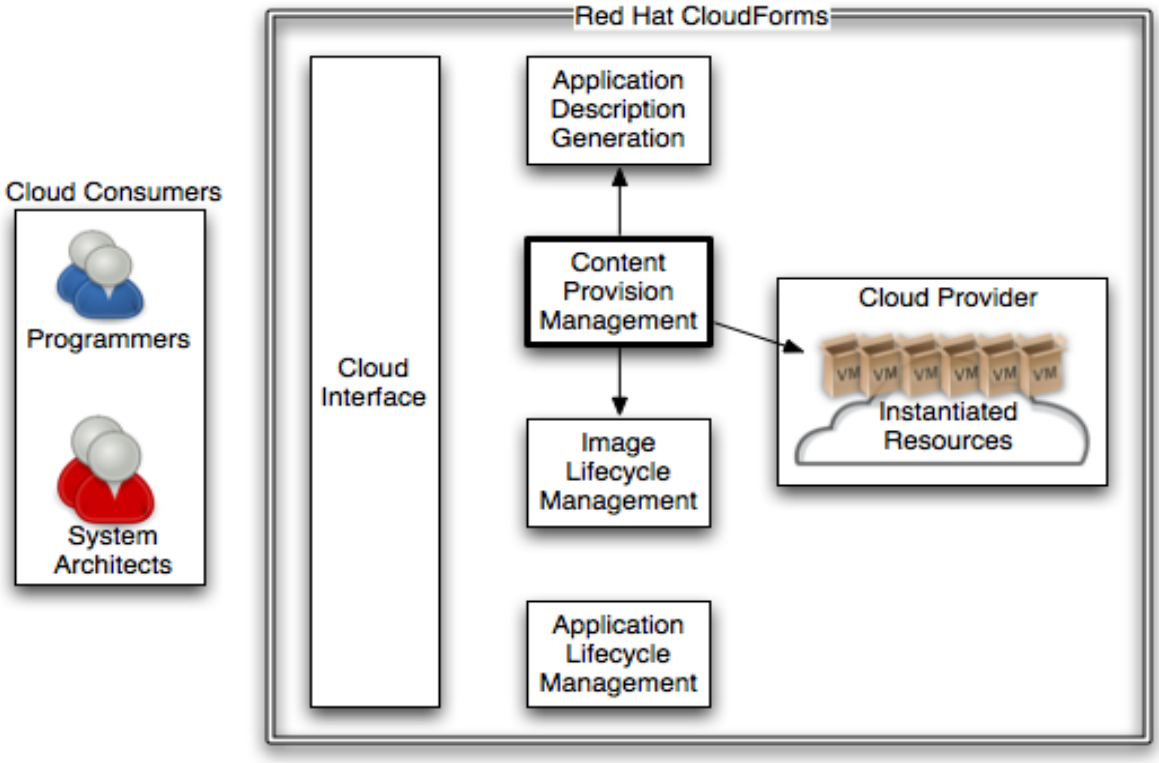

Illustration 3-9: Content Provision Management

3.3.3 Application Description Generation

The Cloud Consumer defines their application deployment as a set of systems configured with collections of software and configuration data required to accomplish the assigned task. The *Application Description Generation* outputs this definition as XML, as depicted below.

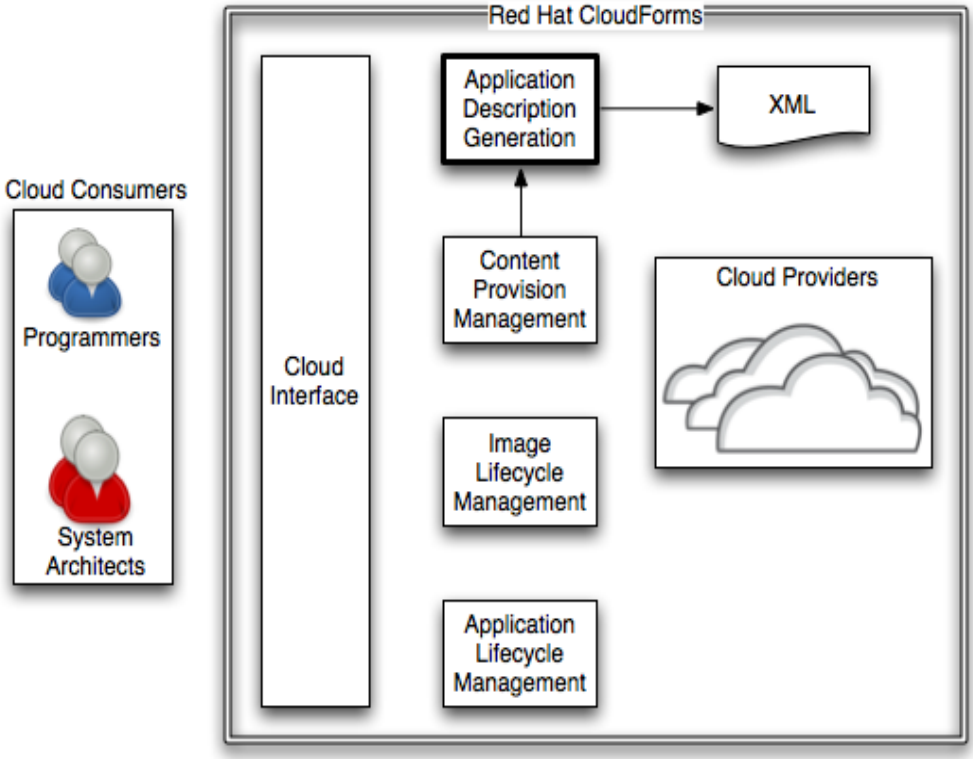

Illustration 3-10: Application Description Generation

3.3.4 Image Lifecycle Management

The Cloud Consumer can decide to stage the software (in the form of disk images) or *Image Lifecycle Management* has the ability for force a late staging when required. Either way, Image Lifecycle Management is responsible to affirm the disk images are available at the Cloud Provider. Image Lifecycle Management tracks images and may use one image as the source for another or build the image from scratch. Once the image is available, Image Lifecycle Management is responsible for the availability of this disk image at the Cloud Provider, as shown below**.**

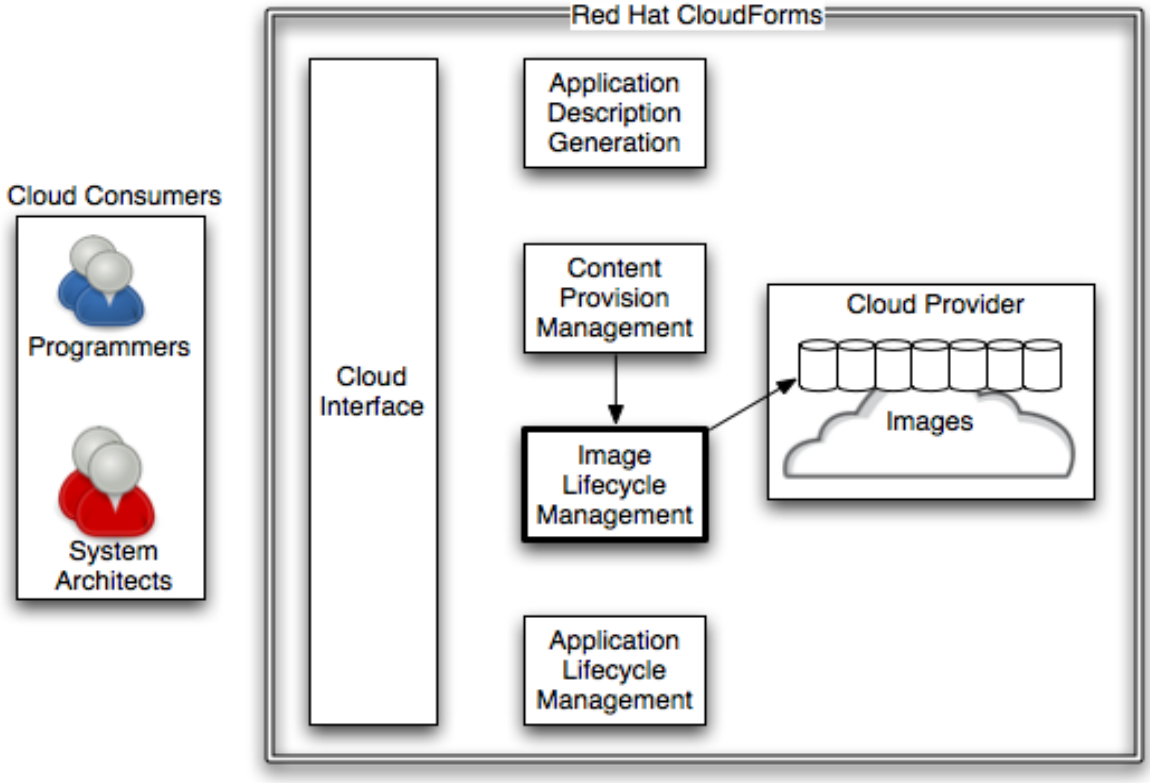

Illustration 3-11: Lifecycle Management

While using an existing image as the source for another image may help to limit proliferation of images, if a system uses a single disk image that contains all the software needed for that system, the potential for re-use is limited. However, if software is layered as separate disk images, e.g. OS, database, and Java environment, any of these individual layers/disk images has much greater potential re-use value. Using this concept of stratifying software proves to be more effectual. For this concept to work, the ability to link the separate disk images to function as a single functional image is required. This is accomplished as part of the post-boot configuration.

3.3.5 Application Lifecycle Management

When the Cloud Consumer decides to deploy their application, the cloud interface is used to instruct *Application Lifecycle Management* to carry out this activity. This is accomplished by using a internal resource manger to identify a Cloud Provider that matches the policies, quota, accessibility and availability that the Cloud Consumer requests. Then Application Lifecycle Management instantiates each system with the software desired, applying local and intra-deployment configurations, activating required cloud services, providing secure access to the systems, and monitoring the deployment. Application Lifecycle Management performs other actions such as shutting down the deployment, etc, displayed below.

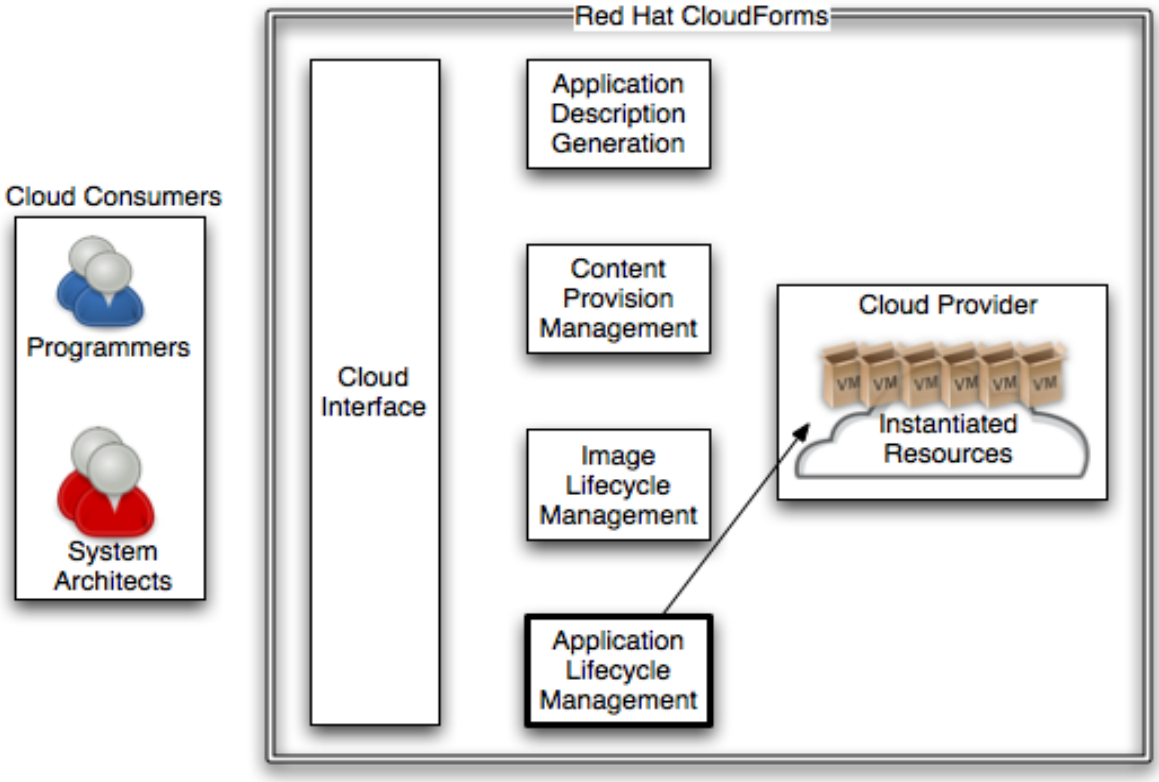

Illustration 3-12: Application Lifecycle Management

3.3.6 Functional Area Summary

The illustration below summarizes the high level conceptual solution when Red Hat CloudForms abstracts various public Cloud Providers or by CloudForms extending various virtualization, Grid, or farm environments. Red Hat CloudForms operations are segregated into the following functional areas:

- Cloud Interface
- Content Provision Management
- Application Description Generation
- Image Lifecycle Management
- Application Lifecycle Management

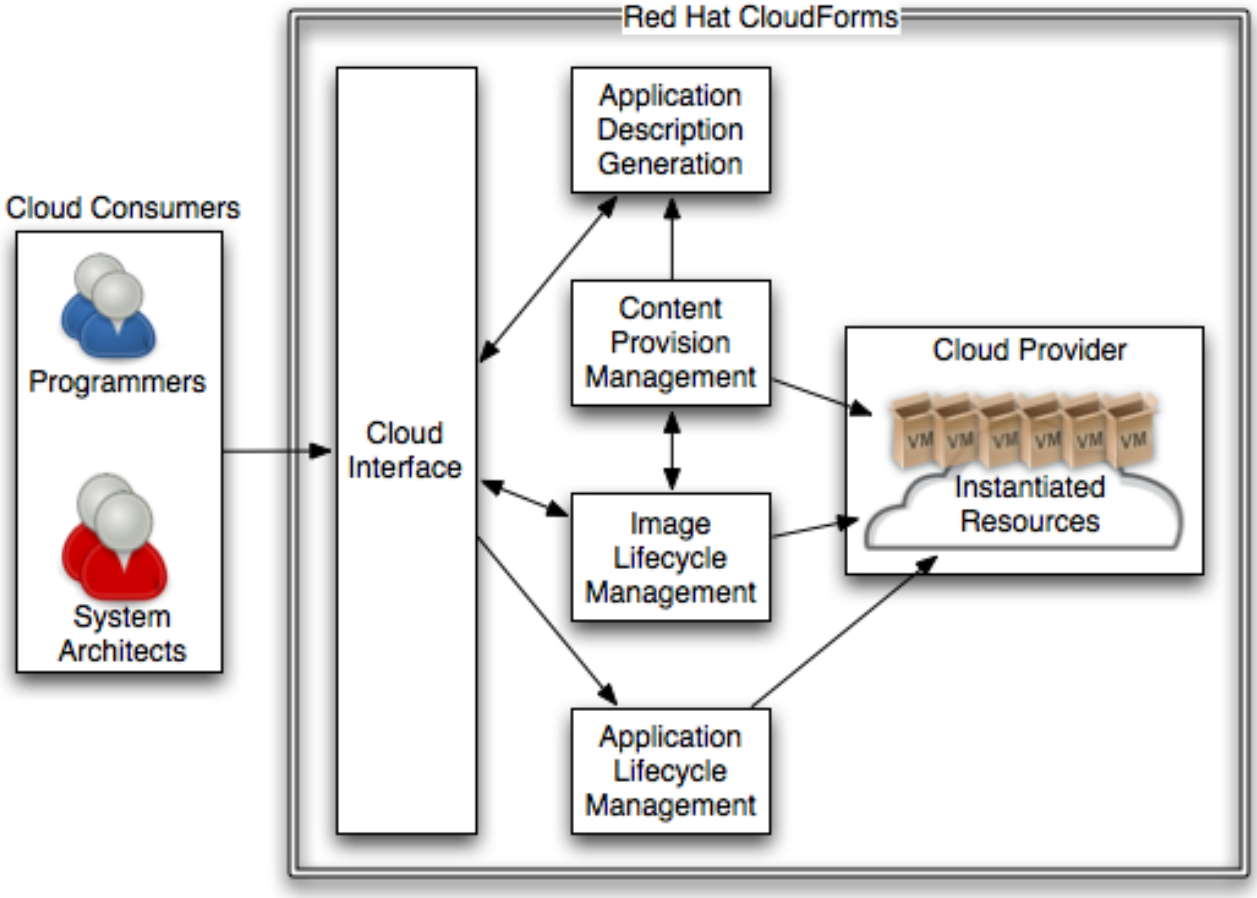

Illustration 3-13: Expanded Functional Overview

4 Red Hat CloudForms Components

The previous section described the architecture in terms of the functional areas. The actual implementation performs the functionality as a set of products. This section identifies the components of Red Hat CloudForms and associates the architectural function with the corresponding components. The Aeolus Project[4] is the umbrella project for many pieces of the cloud software. While depicted below is the mapping of functional areas to the higher level projects, the following sections provide greater detail.

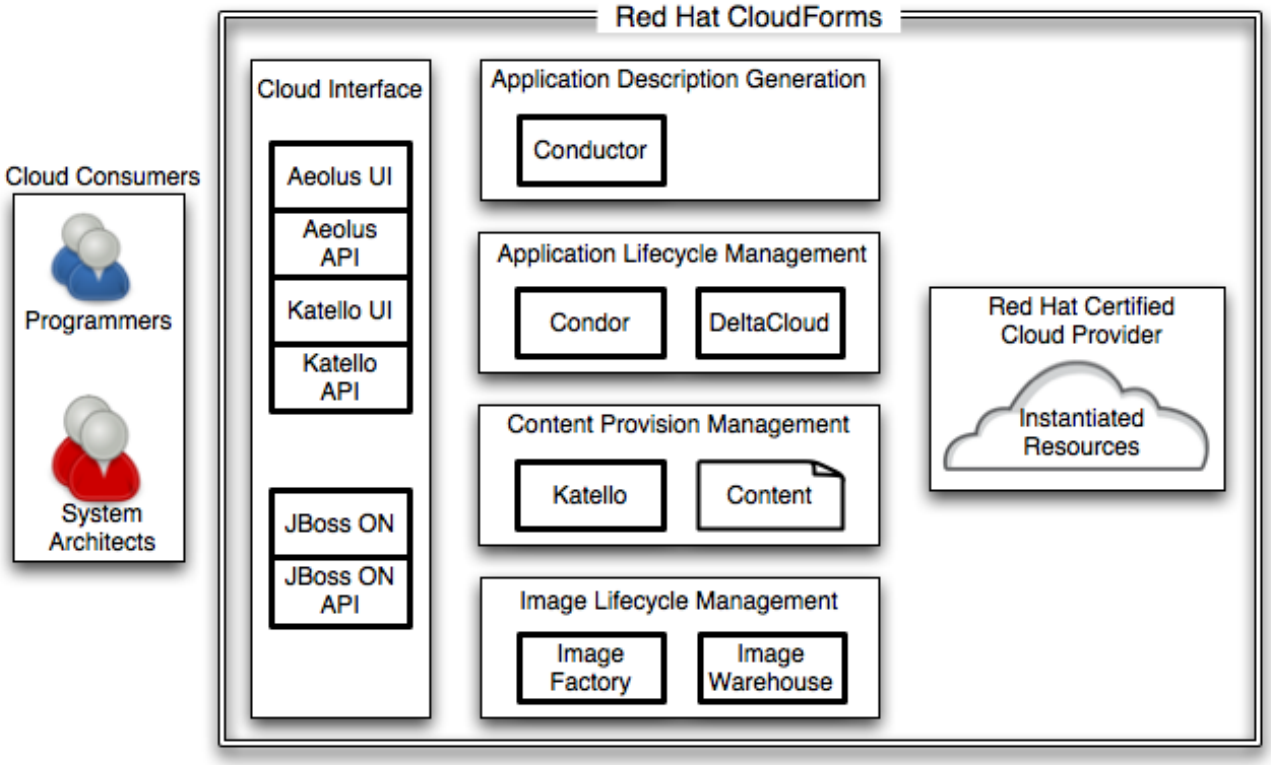

Illustration 4-1: Architectural Component Mapping

4.1 Cloud Interface

The *Cloud Interface* provides the primary user interface for Red Hat CloudForms activities. An API is also available as an alternative access method. The Cloud Interface functionality is supplied by the *Aeolus UI* as represented below.

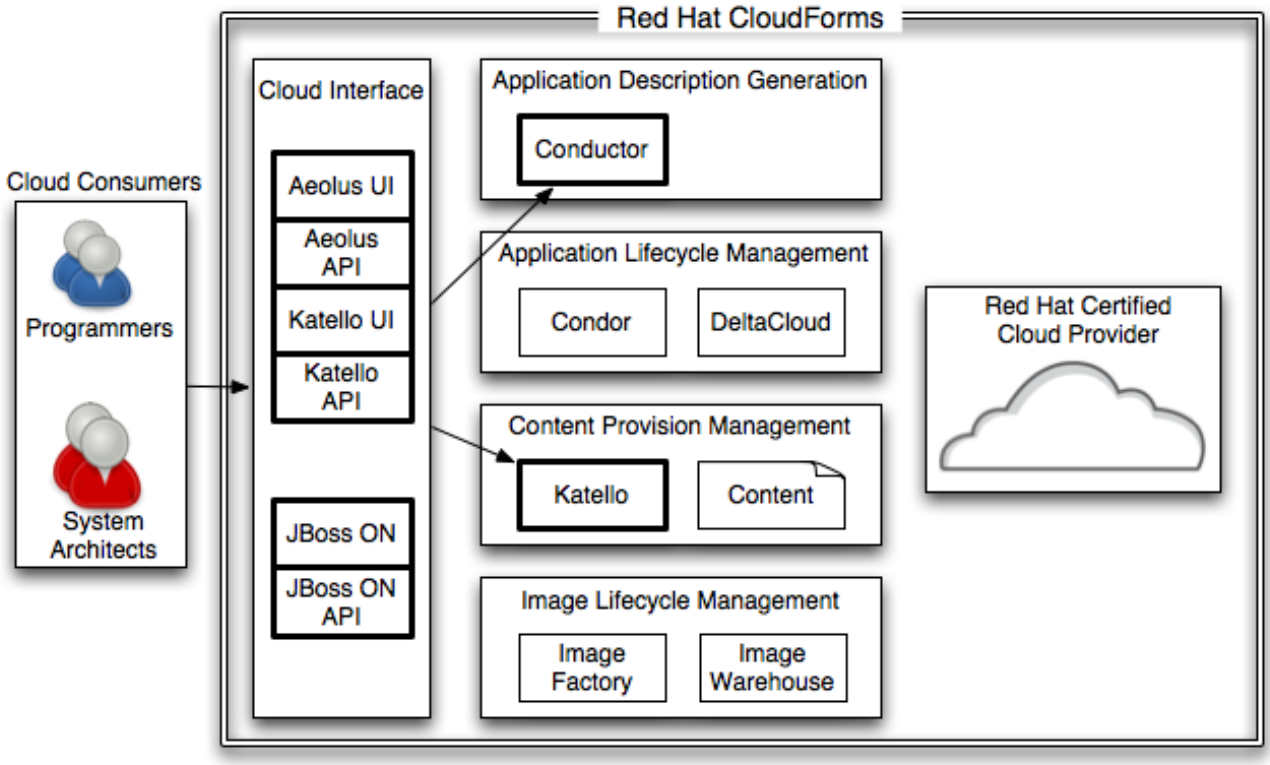

Illustration 4-2: Cloud Interface Components

The cloud interface provides a centralized management interface for Cloud Consumers, whether they be administrators or developers, to interact with disparate cloud providers. Using the web-based interface, a Cloud Consumer can log on and perform certain actions based on the rights associated with their account. The goal of the cloud interface is to abstract the back end cloud provider from the Cloud Consumer, regardless of whether the Cloud Consumer desires to utilize EC2, Rackspace, a Red Hat Enterprise Virtualization infrastructure, or other approved cloud infrastructure for their application. The illustration below shows how the cloud interface provides access to *Resource Management, Image Management, Administration, Reporting,* and *Accounting.*

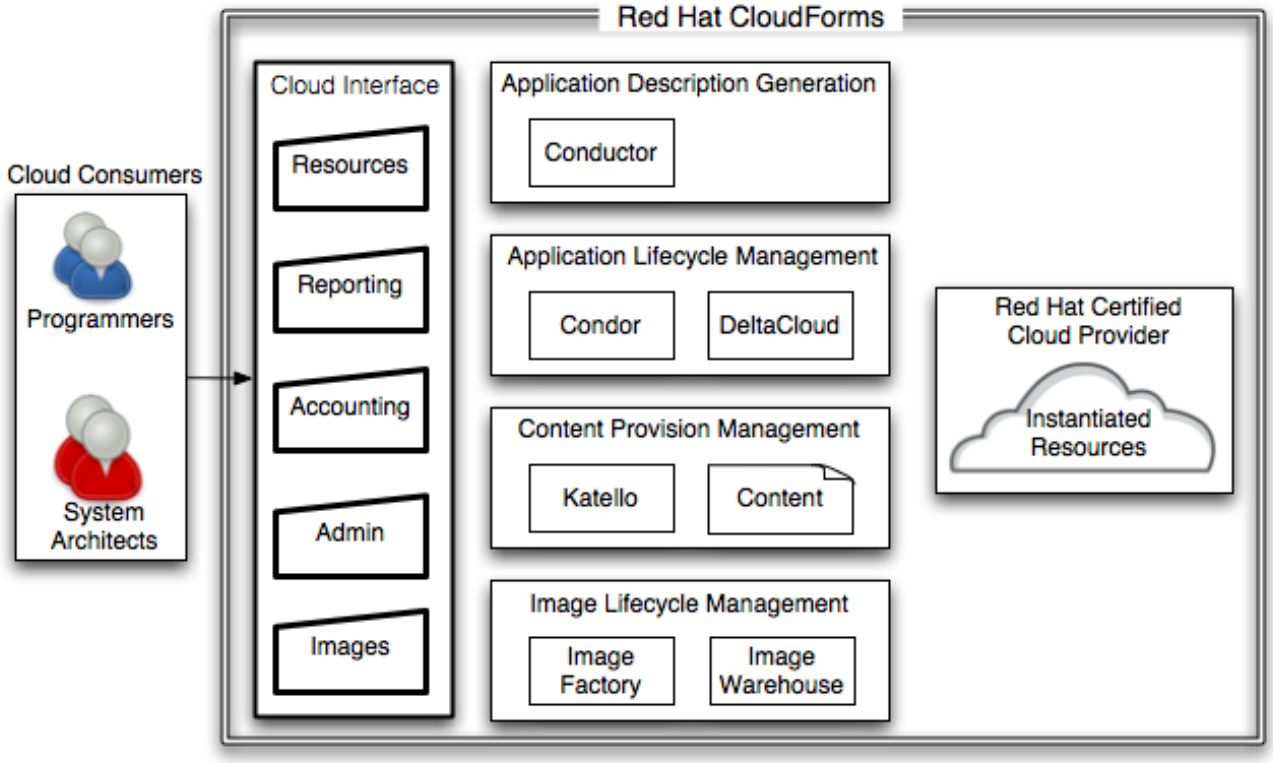

Illustration 4-3: Cloud Interface Functional View

With *Resource Management,* the authorized Cloud Consumer is able to manage the active resources such as listed in **Table 4-1: Resources**. Within each of these categories, attributes like properties and permissions can me modified as needed.

Resource	Description
User account	A Red Hat CloudForms Cloud Consumer; allows access and controls permissions / roles.
Quota	Implements limits on instances or disk usages, can be associated with a user account, cloud provider account, pool, or pool family.
Cloud Provider account	The account that allows access to a specific cloud provider. This account can be associated with multiple pools.
Pool	A grouping of cloud providers as specified by the Cloud providers accounts assigned to for the pool's use.
Pool family	A grouping of pools by user defined semantics– e.g. dev, test. A pool can only be assigned to on pool family.
Instances	These are systems that are running in cloud provider.
Deployment	Groups of instances that are related by being defined as part of the same application deployment.

Table 4-1: Resources

With the *Image Management* interface, the Cloud Consumer is able to create, modify and delete items relating to the definition of application deployment, e.g., systems, disk images, configuration settings, etc. This allows the user to create the framework for publishing applications and instances.

The interface also provides *Administration* capabilities which allows control over account roles and permissions. Providers can be managed as well as hardware profiles.

One other key attribute of the cloud interface is the ability to track *Reporting* and *Accounting* details from the cloud providers and local resources which are being utilized. Items such as the number of instances running and the corresponding charges becomes more and more important as applications are scaled.

4.2 Content Provision Management

Content Provision Management functionality is provided by the Katello component which provides the collection of software and software feeds utilized by Image Factory when building and modifying images. This content can come from a variety of resources. While Red Hat Network is the premium supplier, other methods include repositories, ISOs, or software collections - whether these are Red Hat, Red Hat Partners, other OEMs, or custom supplied.

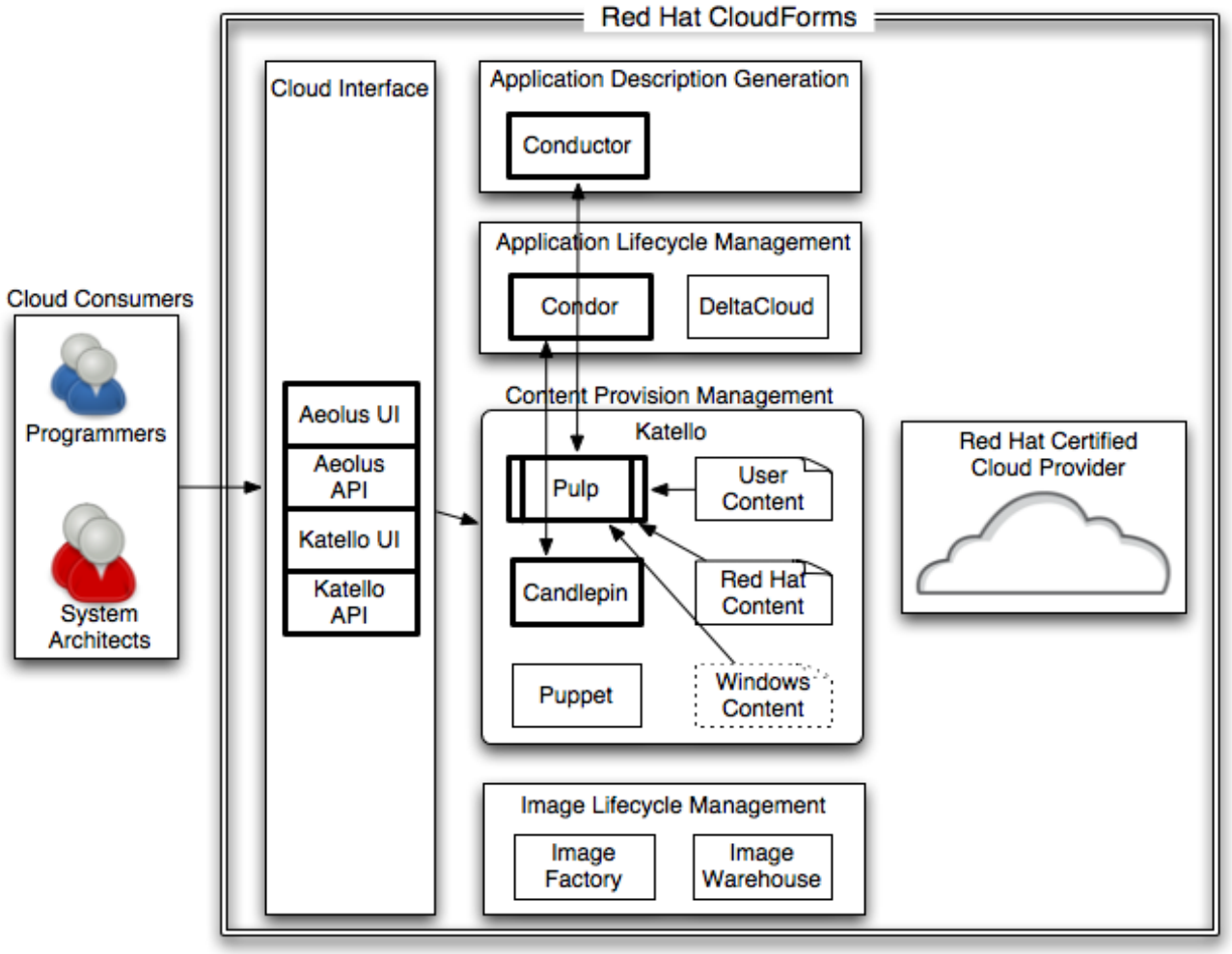

Illustration 4-4: Content Provision Management

4.3 Application Description Generation

Application Description Generation is the set of functionality that allows the Cloud Consumer to create a recipe describing an application that they desire to deploy. As a recipe lists the ingredients and the instructions of how to combine the ingredients. The generated application description identifies the systems and software along with configuration data used in the process of combining all elements. The functionality of Application Description Generation is mostly provided by the Conductor. **Table 4-2: Definitions** defines terms relevant to this component.

Term	Explanation
Image or Disk Image	The contents of a mountable disk; the contents of a mount-point
Template	Description of a disk image with any meta-data required to create an image; the described image may be bootable or non-bootable; a non-bootable image is used to provide a distinct software layer, such as a database
Assembly	Definition of a single instance containing one or more templates and meta-data related to service configurations; since this defines an instance, one constituent template must be described as a bootable image; all configuration actions are performed post-boot
Service Configuration	Optional attribute of the assembly which describes the service or services that the assembly provides to requires; this information is used to configure and tie the assemblies of a deployment together at launch
Deployable	Application deployment definition, contains one or more assemblies and meta-data configuration; this configuration specializes a deployment by qualifying it for a specific targeted infrastructure
CDL	Content Description Language; XML format language used for Template, Assembly and Deployable (TAD) definitions

Table 4-2: Definitions

The application deployments must be described in a structured format that includes the bootable operating system, any software requirements, configuration provided or required, and any specific targeting information to instantiate the application. This is accomplished by the Cloud Consumer interacting with the Conductor via the Cloud Interface (1), as pictured below. The deployable is defined in CDL (3) by describing the templates, assemblies, services, configuration data, and targeting data that compose the entire application deployment. The source of the software options of content is provided by the *pulp* instance in Content Provision Management (2). Targeting information allows the generic description to be specialized for a specific deployment. Most commonly, this information specifies the compute requirements (vCPUs, memory, disk space) needed for an instance.

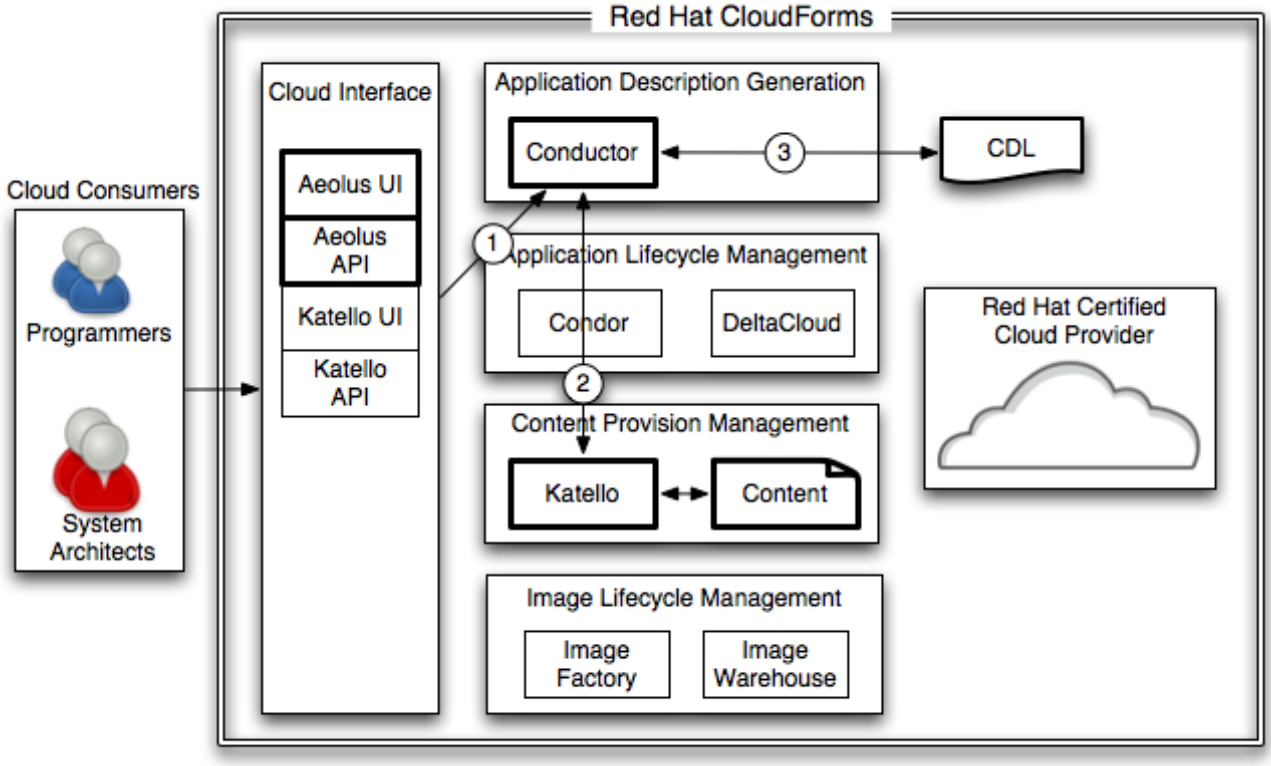

Illustration 4-5: Application Description Generation

The following abstracted sample provides the general concepts of a CDL layout. A Deployable is made of one or more Assemblies with configuration data. Each Assembly is made of one or more Templates and configuration. Each Template lists the software and configuration data. An Assembly optionally indicates the services that it defines or requires. Some of the configuration data may be defined a the time of instance launch.

```
Begin Deployable Definition
  ...
  Begin Assembly Definition
    ...
    Begin Template Definition
      ...
      Software list
    End Template Definition
  End Assembly Definition
End Deployable Definition
```

4.4 Image Lifecycle Management

Image Lifecycle Management is the functionality that creates, stores, and maintains the images and descriptions, which supplies this content to the supported cloud providers. Image Lifecycle Management functionality is performed by multiple component products:

- Conductor
- Image Factory
- Image Warehouse

Relevant terms used in this component section are listed in **Table 4-3: Definitions**.

Term	Explanation
ICICLE	Image Content and Intended Configuration LanguagE; While the CDL describes the definition of Templates, Assemblies, and Deployables, the ICICLE provides listing of detailed software revisions and configuration parameters of the created entities. While CDL can be thought of as the shopping list, the ICICLE is the itemized receipt.

Table 4-3: Definitions

One of the many functions of the Conductor is to initiate and co-ordinate Image Lifecycle Manager activities.

Image Factory is the component that is responsible for building all cloud images. The image description is supplied from the XML/CDL generated from Application Description Generation (Conductor). The content comes from Content Provision Management. Additional software that is needed to support the cloud operations is also added to the image.

Image Warehouse tracks all images and is responsible for staging the images at the appropriate cloud provider.

The images that are defined by the cloud consumer, are built with Image Factory and stored in Image Warehouse. When a Cloud Provider provides an image as the source for the desired image, Image Factory is not called upon. The provided image is specified in the definition. However, Image Warehouse still stores the meta-data which allows the proper image formation and assembly. In addition, in all cases Image Warehouse is also responsible for all staging of images.

4.4.1 Image Lifecycle - Standard

The standard Image Lifecycle is explicitly initiated by the Cloud Consumer after defining templates using the cloud interface and requesting a build, or indirectly by the consumer by requesting an instance launch of a deployable that contains templates that have not had corresponding images built (1), as shown in the diagram in this section.

After Conductor is requested to initiate a build (2), it transmits a message containing the CDL and the target Cloud Provider to Image Factory which is placed onto a queue (3). When resources are available, Image Factory starts a build using the input removed from the queue. If the build is from scratch, a Just Enough Operation System (JeOS) - which is a minimal OS - is initially created. Instead of building from scratch, Image Factory can use a pre-existing disk image. Next, the JeOS or pre-existing disk image is modified with the addition or removal of software until the CDL is satisfied (4). The format of the disk image may require manipulation to ensure compatibility with the target Cloud Provider. Once the disk image is complete, Image Factory creates an ICICLE from the image, listing the specific versions of the software content and configuration parameters (5).

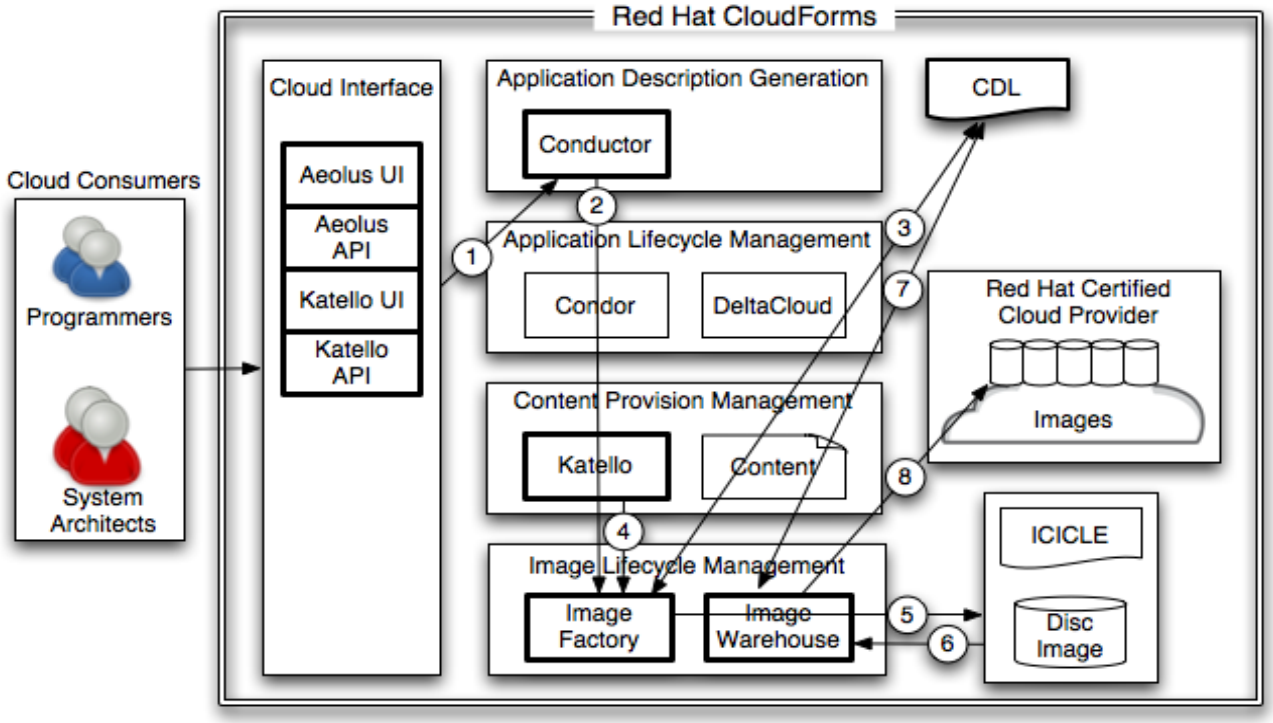

Illustration 4-6: Image Lifecycle Management - Standard

The completed disk image and ICICLE are transmitted to Image Warehouse (6). Image Warehouse stores the disk image, ICICLE, and source CDL for potential future use (7). It also assures the disk image is available in the target cloud (8). For example, if the target cloud provider is a Red Hat CloudForms cloud using Red Hat Enterprise Virtualization, the image is migrated into RHEV's import domain and instantiated. If the target is Amazon EC2, the image is bundled in S3 and registered in the appropriate region, allowing access for the specified Cloud Provider user.

4.4.2 Image Lifecycle – Snapshot

After an instance had been deployed, it may be modified in the environment by user directed software updates provided by the management capabilities of Content Provision Management. These updates create a discrepancy between the image stored in Image Warehouse and the running instance. Since the Cloud Consumer performed the updates, they may also desire to update the Image Warehouse ICICLE and disk image. The Cloud Consumer may also desire to leave the original images, since multiple deployments could have been started and not all should be updated.

The Cloud Consumer updates the image definitions (1) by informing Conductor (2) to initiate the activity. Katello updates definitions initiated by the Cloud Consumer and sends Image Factory the listing of changes that have been applied to a previously instantiated image (3). Image Factory creates the updated ICICLE and disk images (4) which are stored in Image Warehouse (5). Image Warehouse optionally pushes the disk images to the appropriate cloud providers (6). This workflow is depicted in the following illustration.

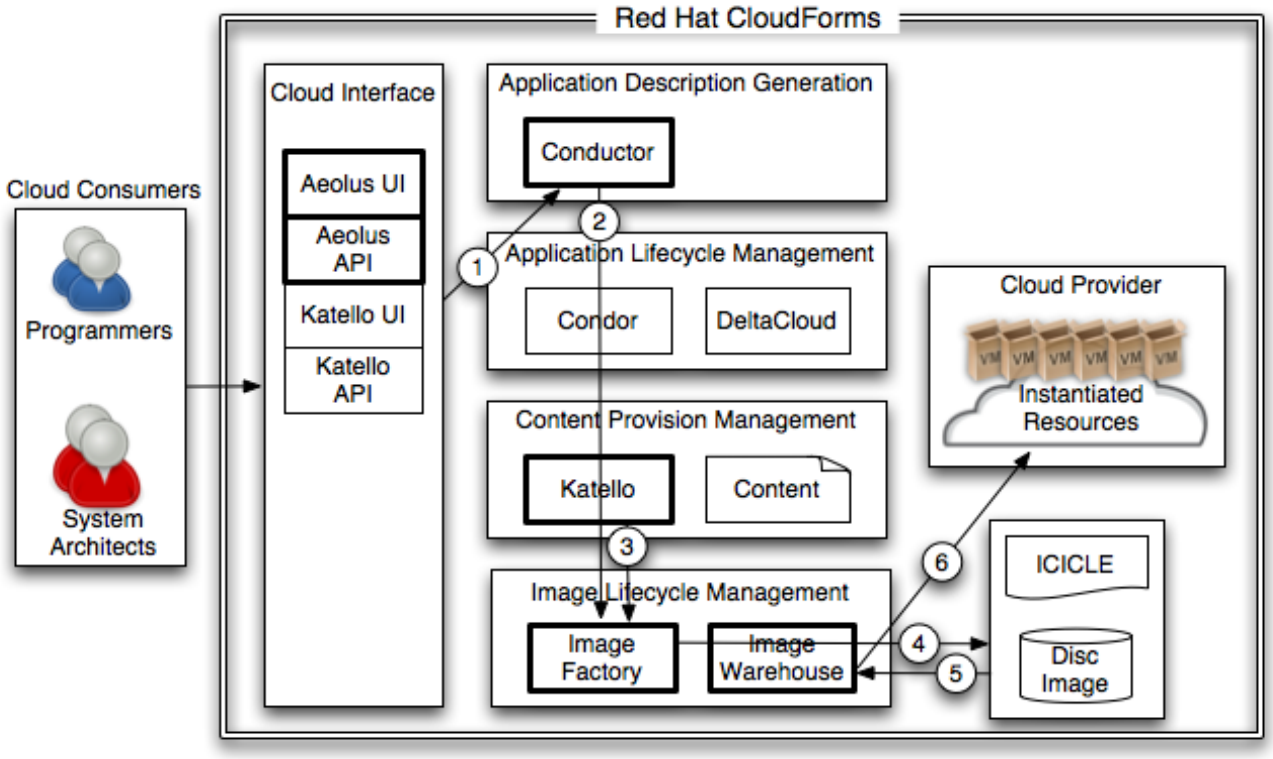

Illustration 4-7: Image Lifecycle Management - Snapshots

4.4.3 Image Lifecycle – Katello Import

The Cloud Consumer may initiate (1) an image build from a Katello template (2), allowing the Katello template to specify the content details of an image instead of the Cloud Consumer. The remaining process , steps (3) through (8), flows as in the standard case (refer to **Image Lifecycle - Standard**), except Conductor is initiated by Katello. This process is portrayed below.

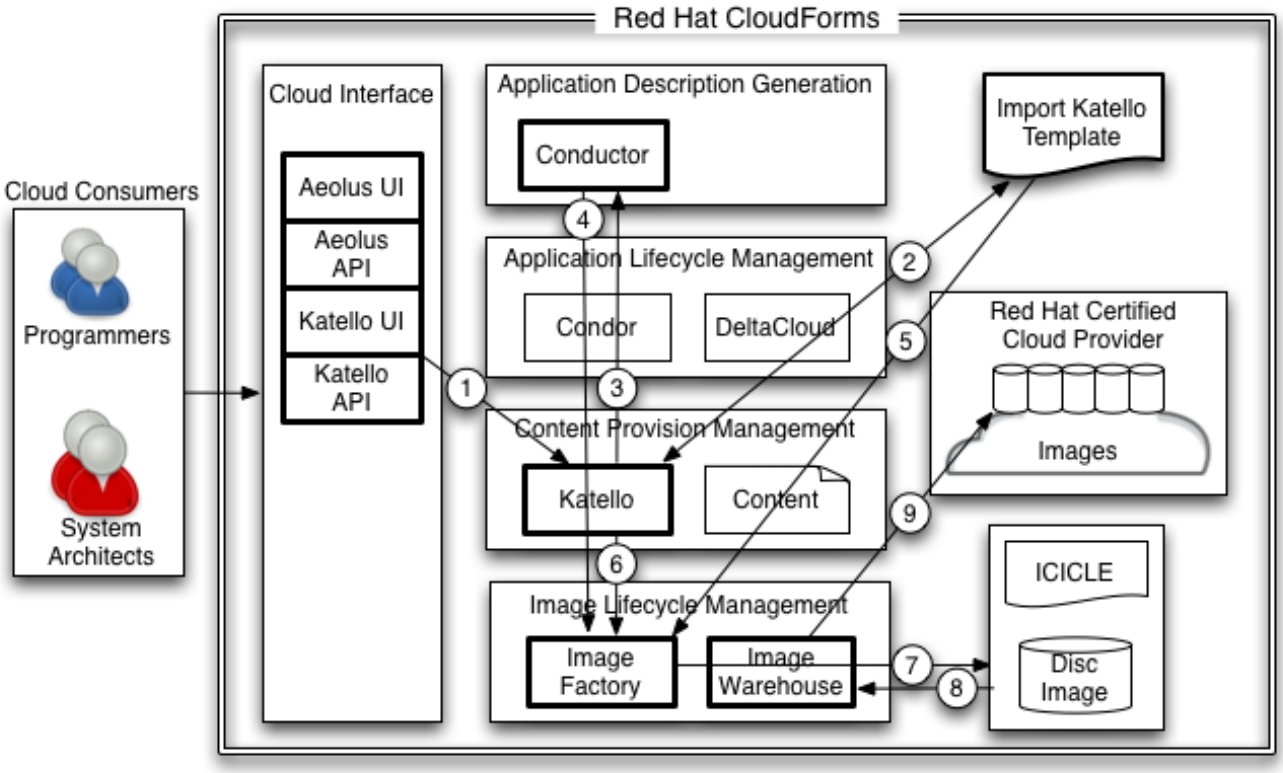

Illustration 4-8: Katello Template Import

4.5 Application Lifecycle Management

The functionality of Application Lifecycle Management allows the Cloud Consumer to control the state of instances in the cloud, whether launching, stopping, monitoring, etc. The functionality of Application Lifecycle Management is provided by several products, specifically:

- Conductor
- Condor
- Audrey
- Deltacloud
- Image Warehouse

Table 4-4: Definitions defines relevant terms used in this section.

Term	Explanation
UUID	Universally Unique Identifier; an identifier unique for each instance
Post-boot Configuration	Once an instance is initially launched, activities are performed to apply configuration and parameter settings, add additional software or disk images, provide data to other systems for configuration, and prepare instances for cloud management.

Table 4-4: Definitions

One of the many functions of the Conductor is to initiate and co-ordinate all Application Lifecycle Management activities. The Conductor is also the maintainer of the resource data that is used in the workflow which determines the best suitable launch environment.

Condor provides the functionality of a resource manager. In addition to scheduling the cloud instances, it ensures that resources are available and enforces quota and policy. Condor controls the state of cloud instances, whether launching or destroying. If Condor sees that an instance is no longer operating, it restarts the instance based upon policy settings.

Communication with different cloud providers is controlled through the Deltacloud driver. The Deltacloud driver creates an abstraction layer between the consumer and third party clouds. This model allows Red Hat CloudForms to function with various Cloud Providers without requiring all components to be written for the specific Cloud Provider.

Audrey is a set of tools that performs post-boot configuration of cloud instances. The list of functionality it provides includes applying local and intra-deployment configurations, activating required cloud services, providing secure access to the systems, and monitoring the deployment.

Image Warehouse stores the descriptions of the instances. These descriptions are supplied to the configuration server for completion of any required actions. The process of launching an instance is pictured as follows.

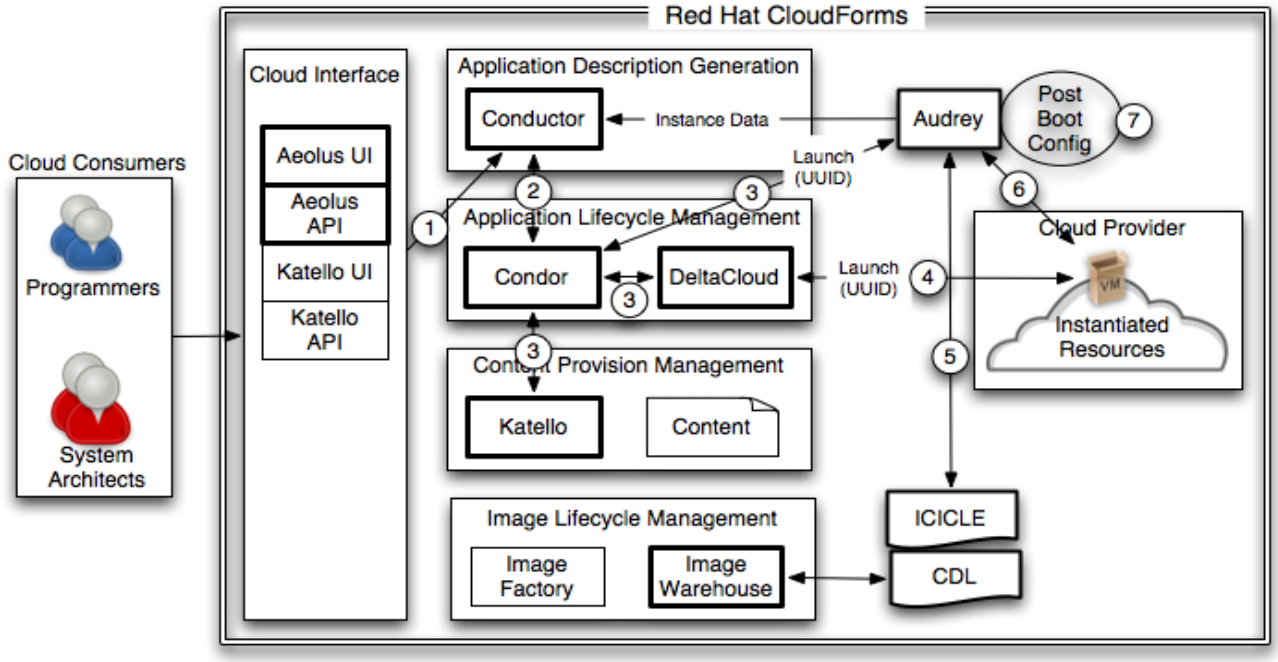

Illustration 4-9: Application Lifecycle Instance Launch

The Cloud Consumer initiates a instance launch using the cloud interface (1). Conductor starts the instance launch by submitting a request for the instance with Condor (2). Condor confirms that the request does not violate policy or quota, and matches the request to a cloud provider. Once matched, Condor sends out two parallel messages (3) with the UUID to be used for the instance and other configuration data. One is to the Deltacloud driver to start the instance (4). The other message is to Audrey to configure the instance with the provided UUID. Condor also contacts Katello to resolve entitlements for the launched instances (3). Audrey contacts the Image Warehouse to retrieve the CDL and ICICLE data and pass on the configuration requests and UUID on to the configuration server (5). Once the configuration server discovers the running instance with the matching UUID (6), it controls the post-boot configuration process (7). Included in the post boot process is establishing trusted identity and credentials. Once the instance has completed this process, the configuration server passes instance data to Conductor.

Once the instance boots up, the post configuration takes place. This sets any IP addresses, start services, etc. for the image so it is ready to run. In the case of the load-balancer a pool of IP addresses must be provided and policies must be set. For the web servers, they must be configured with a default gateway and the web services must be started.

4.6 Cloud Services

Section 4.3 Application Description Generation broached the subject that assemblies may indicate what services they provide or the services they require. Using a service that the Cloud Consumer defined allows the interaction of multiple instances to become part of the deployable's recipe. However, there may be services that a deployment may use that are not provided in the cloud users deployable definition. *Cloud Services* are add-ons to cloud deployments that ensure consistent functionality among various cloud providers. The following is a list of functional areas that some planned cloud services may provide:

- Monitoring
- Managing
- Messaging
- Archival Storage
- Replication File System Storage
- Cloud Id Management
- High Availability

Cloud Services are special services that a user does not need to define, as a consistent definition is provided. Cloud Services are added to a deployable's defintion when the Cloud Consumer indicates they wish to include the service. The instance may be spawned as part of their deployable, or the cloud provider may have a dedicated instance available in the cloud which provides the service to multiple tenants.

4.6.1 Monitoring

Red Hat uses the Matahari[5] infrastructure to allow monitoring and controlling agents on cloud instances. The agents provided allow the starting of applications and provide the monitoring used in High Availability.

For JBoss Enterprise Middleware content, a management agent (JBoss ON) is installed via a managed service definition. This mechanism can be used for any additional managed containers used in deployments.

4.6.2 Managing

As with non-cloud environments, Red Hat provides Management capabilities both for Red Hat Enterprise Linux and JBoss Enterprise Middleware.

For Red Hat Enterprise Linux, the Cloud Consumer's application can be updated in place when deployed using Katello. This causes the deployed images to fall out of compliance with the definitions and saved images stored in the Image Warehouse. When a running instance deviates from the stored image, it is identified as divergent. The Cloud Consumer can decide to leave this as the status quo, or reconcile the instances and saved images. To make the images consistent, Katello sends Image Factory the list of changes it has applied to the instance. Image Factory generates updated CDL, images, and ICICLEs for the deployment. The updated image or images are pushed to Image Warehouse which the pushes to the cloud provider.

For JBoss Enterprise Middleware, the Cloud Consumer can deploy their application using the basic provided infrastructure, or use a JBoss Operations Network (JBoss ON) server. In the example in this paper, a JBoss ON server allows the Cloud Consumer to fully monitor, control, and update the JBoss Enterprise application.

4.6.3 Messaging

Not only does the infrastructure use AMQP in the form of MRG[6] for communications internally, but Red Hat will deploy the Messaging component of Red Hat Enterprise MRG as the CloudForms' Messaging Service.

MRG Messaging key features include:

- AMQP support
- Flexible messaging paradigms
- Multi-language client support
- High performance
- Transient and durable messaging
- Federation
- Transactions
- Security
- Queue semantics
- XML support
- Distributed management console

4.6.4 Archival Storage

Archival Storage, sometimes referred to as blob storage, is provided as a method for reading and writing large objects. The mechanism used for image storage in Image Warehouse is also available to the Cloud Consumer for their own data. Operations such as whole file “get” and “put” are performed via HTTP. The implementation is based on Project Hail[7] or CloudFiles with modifications such as a distributed database for tags and metadata. The data is stored is a shard format, meaning the data is distributed using horizontal partitioning. Data is written to one place, however, replication can be used to distribute reads. Replication is policy driven and can be based on object context, site, tags, security, etc.. The Image Warehouse daemon provides the ability for a caching or replication process to push copies to existing Archival Storage such as Amazon's S3, RackSpace's Cloud files, Azure, Google Storage or to another warehouse instance. If replication is between warehouse instances, when a request for an object that has not been copied to the child node is received, the object is 'pulled' on demand.

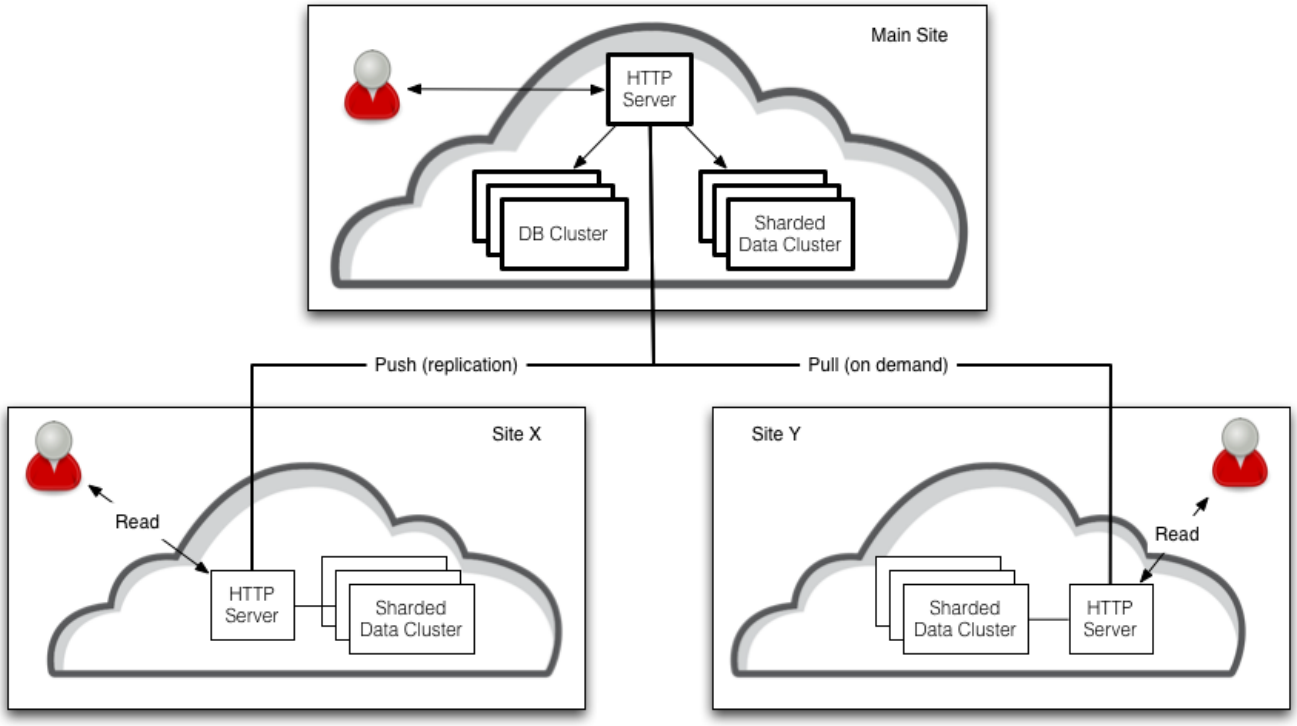

Illustration 4-10: Archival Storage

4.6.5 Replicated File System Storage

CloudFS provides a distributed shared file system for cloud use with POSIX semantics. CloudFS can be included in CDL definitions to provide storage for statefull and hybrid cloud deployments. In addition, this file system is suitable for deployment by a cloud provider as a permanent, shared service. CloudFS is based on GlusterFS and adds the following capabilities:

- Stronger authentication and authorization
- Encryption (AES-128/AES-256), both on the wire and on disk
- Multi-tenancy (isolating tenants' namespaces from one another)
- Quota and accounting support
- Multi-site replication

All of these features can be implemented in a modular way, so that deployments can utilize only those deemed necessary or appropriate for their specific situation.

4.6.6 Cloud Id Management

Cloud Id Management's main goal is to transparently integrate with the existing identities and identity management systems present in the enterprise. Identity management in Red Hat CloudForms is accomplished through Red Hat Enterprise Identity (IPA)[8] project is based on the open source FreeIPA project[9]. FreeIPA is an authentication and authorization framework for large-scale Linux and Unix deployments. It integrates servers for Kerberos, LDAP, DNS, and X509 Certificates into a secure, reliable, and scalable identity management solution.

In a common scenario, when a Cloud Consumer instantiates a resource, there are several entities to be considered:

Entity	Description	Notes
User	Cloud Consumer	In contemporary deployments authentication is predominantly through Active Directory Domain Services (ADDS).
Machine	System which the Cloud Consumer uses to access the CloudForms environment	Cloud Consumer's workstation which may or may not be connected to an authenticating agent.
CloudForms	CloudForms infrastructure	Uses Red Hat CloudForms internal secure authentication.
Instance	Cloud Consumer's deployment/instances	May use a separate or an existing domain.

Table 4-5: Identity Domains

Red Hat CloudForms takes advantage of an internal IPA instance or enables proxy authentication to the external ADDS server(s) in case a cloud consumer has multiple domains. CloudForms addresses the use case of enterprise Single Sign-On (SSO) allowing the Cloud Consumer credentials acquired by logging into their workstation to be respected by the Cloud Interface, thus the CloudForms infrastructure. Additional functionality allows the Cloud Consumer's identity to be respected across different identity domains, thus the Cloud consumer will be able to directly access launched instances.

Using System Security Services Daemon (SSSD)[10] cross kerberos trust functionality between IPA and ADDS can be established. The fact that multiple identity domains must be considered creates a complex matrix of use cases - most of which CloudForms supports. However, there are limitations that CloudForms might not be able to address in the near future as listed below.

Use Case	Status
Joining a Windows machine into the IPA domain	It is not possible to make a Windows machine be a part of IPA domain since it has proprietary protocols that IPA does not plan to support in the near term future
Changing the way Windows machine joins domain	It is the same problem but the client side solution. CloudForms do not plan to provide a client software for the Windows workstation to join an IPA domain, however, a solution might be provided by the Red Hat partners in the future.

Table 4-6: Use Cases Limitations

4.6.7 High Availability

The optional High Availability cloud service has the goal to deliver maximum application service availability for a collection of deployments. This is achieved by the detection or recovery of failures in any of the following components of a deployment:

1. Monitored Applications
2. Individual Instances of Deployments
3. Cluster Services
4. Entire Deployments

Recovery from a detected failure may require terminations of components of the deployment. The restarting of components is controlled by either Matahari agents or Condor. The following illustration depicts a basic High Availability Configuration.

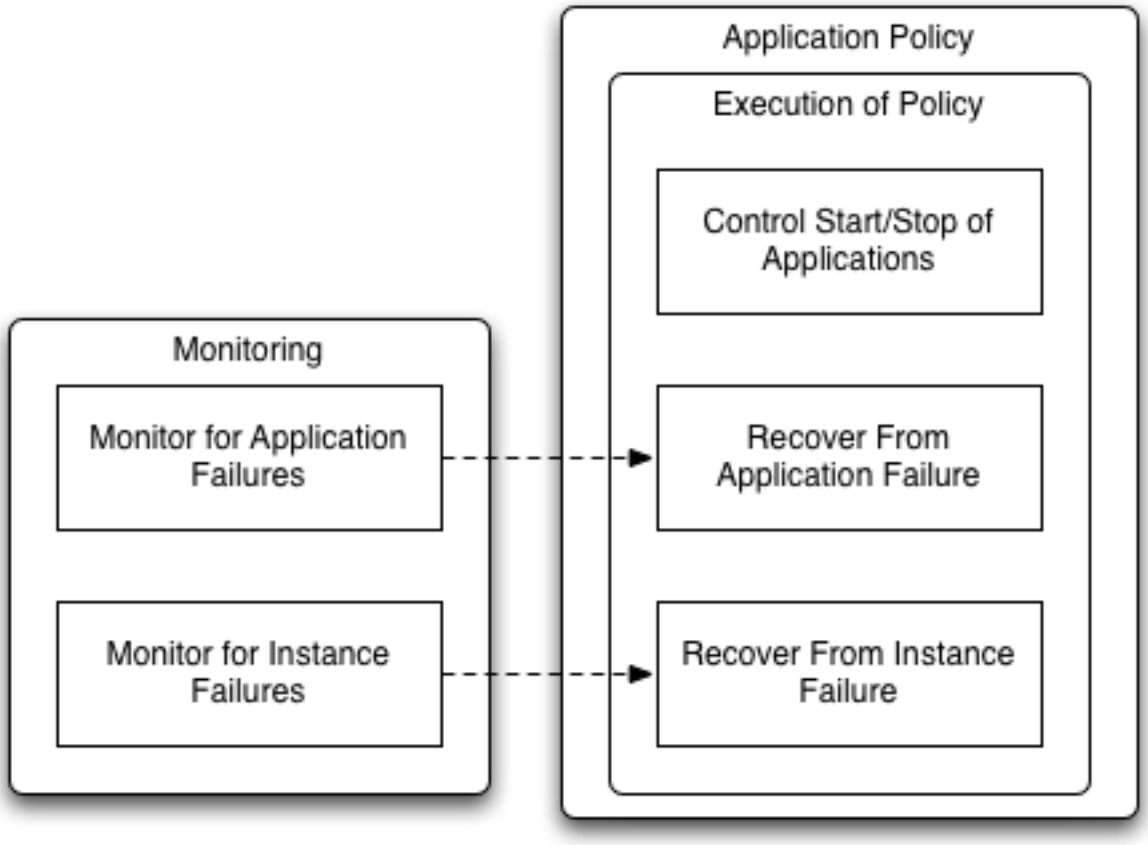

Illustration 4-11: High Availability Standard Cloud Policy Engine

The High Availability Service also has the ability to escalate failures as determined by the *Cloud Policy Engine,* as shown below. The purpose of escalating failures allows a repetitive lower level failure to be recovered using a higher level recovery. For example, if an application fails 10 times in 30 minutes, the Cloud Consumer may wish to escalate the application failure into an instance failure. The Cloud Policy Engine is implemented using upstream Pacemaker[11] services.

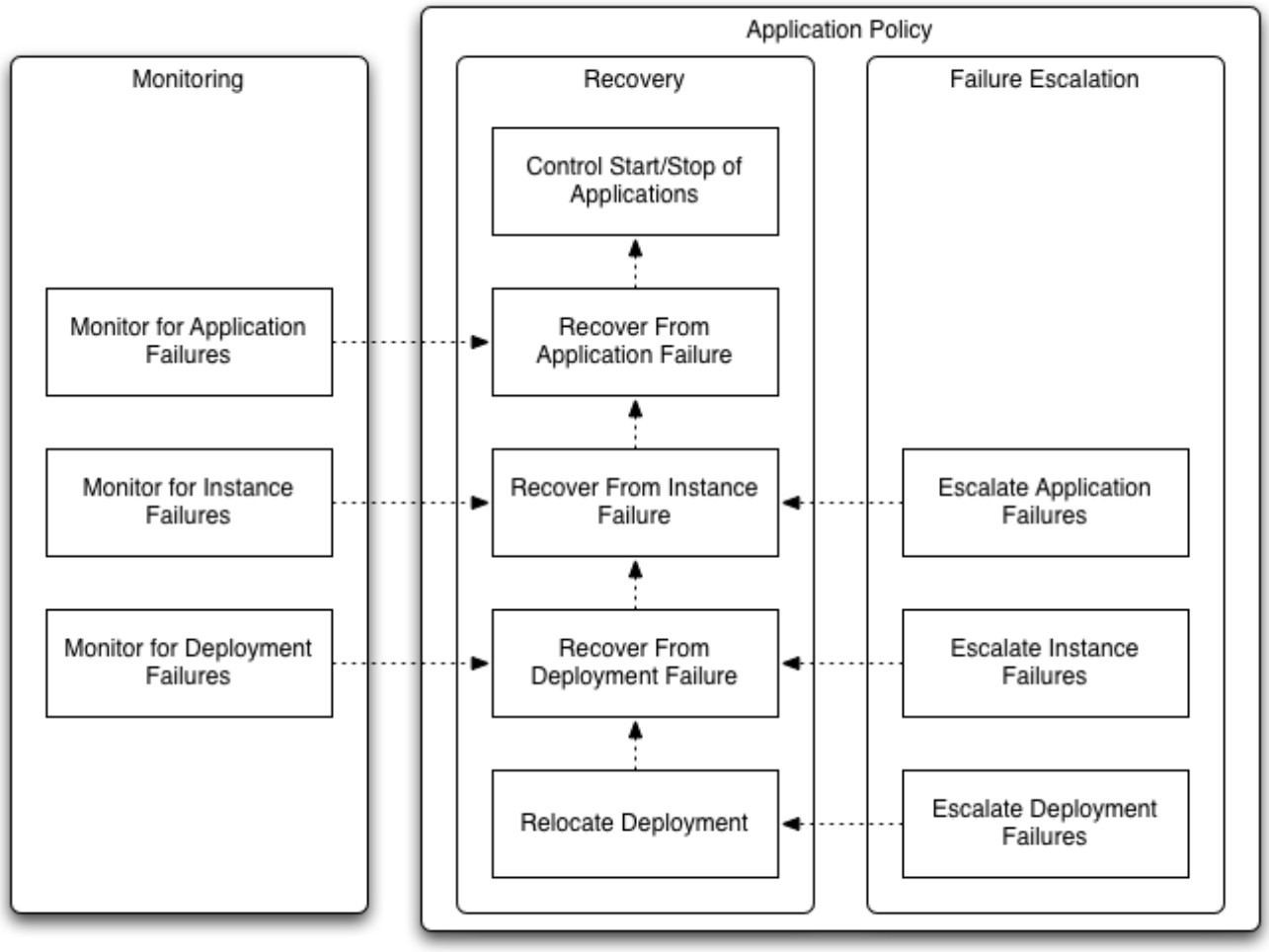

Illustration 4-12: High Availability Advanced Cloud Policy Engine

5 High Level Architectural Example

This section takes a high level approach in demonstrating the process of defining an application into a Red Hat CloudForms environment. For this discussion, the activities to support the Cloud Consumer have been assumed, in other words, the focus is on how the Cloud Consumer implements their application not the configuration of the infrastructure.

This Cloud Consumer would like to present a managed, highly-available web retail presence which offers digital products such as ringtones, apps, e-books, music, etc. The specific cloud provider that hosts the retail presence is not a priority. The requirements are such that the Cloud Consumer can access their store while controlling the entire life cycle management of the application, e.g. define, deploy, update, scale, manage (migrate, snapshot/backup), and tear-down.

The *Overview* section follows the process of planning and designing the retail store. The *Define* section provides the specific definitions that would be implemented.

5.1 Overview

Whether or not one is using a cloud, virtualization, or bare metal, a retail presence requires planning and design. This type of application uses a multi-tier model to allow for scalability and availability. These tiers consist of a web tier, application tier, and database tier, as shown the diagram that follows.

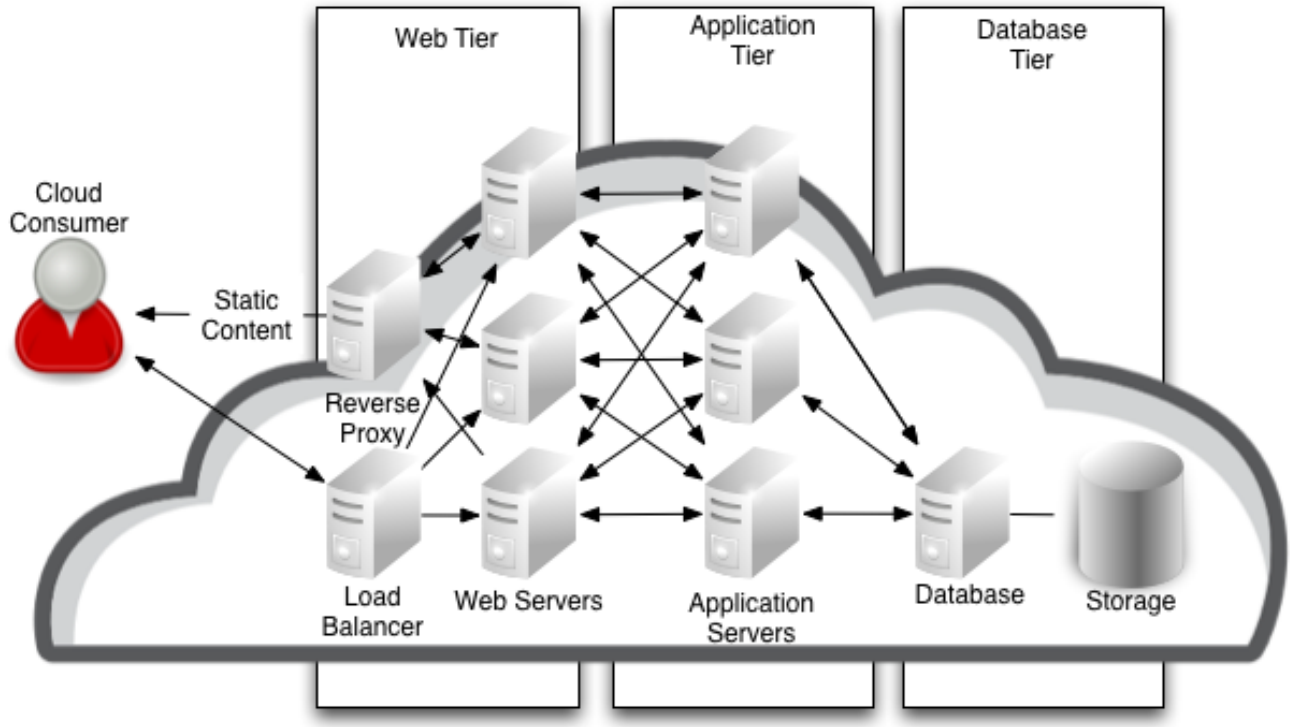

Illustration 5-1: Application Tiers

Web servers are used to provide the front end. A reverse proxy is used to cache static content, while dynamic content is generated by a customized Java Enterprise Edition (EE) application.

Java EE middleware provides a database driver which allows for the connectivity between the web front end and the database back end in an abstracted fashion.

While a solution for the retail presence could be hosted on a single system, this solution uses multiple web servers to provide scaling and active/active availability. Hardware or software based load balancing may be used to spread the requests across the participating web servers, but this implementation uses a software-based load balancer. The middleware is hosted on the same systems as the web-server, utilizing clustering to maintain availability and consistency. The load balancer, reverse proxy, and database are each separate systems.

For high availability of the load balancer, reverse proxy, and database server, the active instance is monitored. Upon a disruption in the service, a replacement server is instantiated. For the database server, this requires the storage to be highly available, highly reliable, and persistent. This storage must be able to be disassociated with the old instance and associated with the new instance. A Cloud Service that provides a Cloud FileSystem is used.

The process of directing network traffic to the site may require a Virtual Private Network configuration. If the deployment is open to the internet, a DNS update is possible.

To summarize, **Table 5-1: Store Components** lists all active systems by functionality planned for the initial deployment.

Instance Name	Instance Count	Role
load-balancer01	1	Distribute Incoming Requests
reverse-proxy01	1	Serve static content quickly
database01	1	Store data for application
user-app-store01-03	3	Host user applications (webserver, middleware, clusterized JEE instance with JDBC instances

Table 5-1: Store Components

The following diagram represents the entire application deployment.

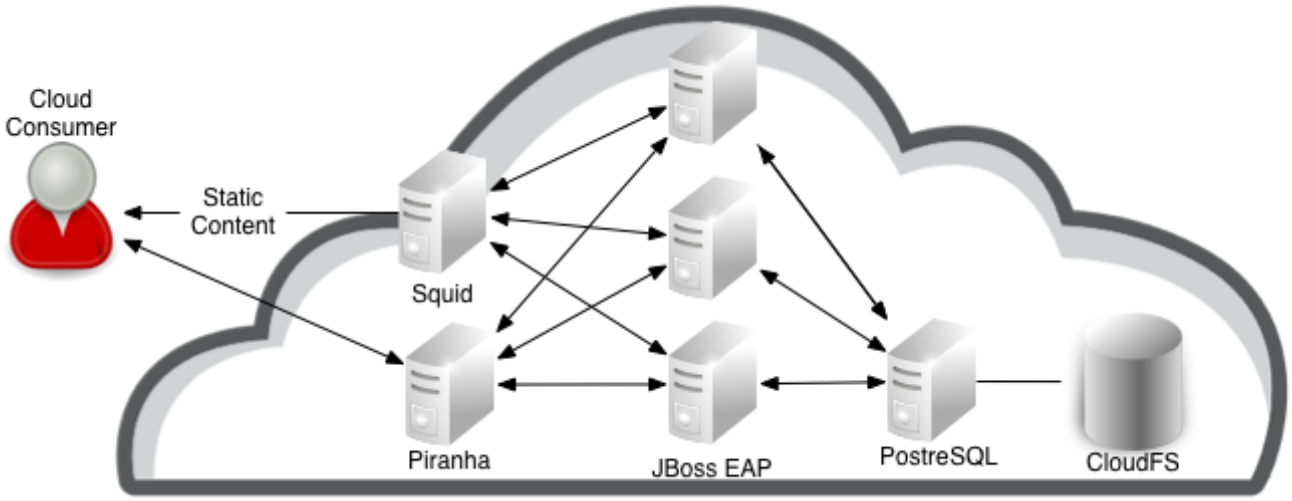

Illustration 5-2: Retail Web Store Application Deployment

Not all the systems have the same compute, memory, or IO requirements. For example, a load balancer may have minimal requirements in regards to storage space, however, the storage for a database is more critical.

5.2 Defining Application Deployment

This section defines the application in terms of the Templates, Assemblies, and the Deployable. The Cloud Consumer has the option to supply configuration and/or customization parameters including a script for the various components.

5.2.1 Define Templates

As described in detail in Table 4-2: Definitions, a Template is a recipe of what software should be in a disk image. This description is the list of software contained in the disk image, along with metadata identifying the supported Base OS. The Base OS template is provided by either a Cloud Provider's definition or from a Katello definition.

All Templates are defined as required for the final application stack as detailed below.

Template	Requirements	Boot
rhel6_base	rhel-x86_64-server-6	Y
lb	rhel-x86_64-server-lb-6	N
rproxy	squid	N
db	PostgreSQL	N
app_server	JBoss Enterprise Application Platform	N
app_store	User supplied application bundle	N

Table 5-2: Templates

5.2.2 Define Assemblies

An Assembly is a list of Templates, one of which must describe a bootable image. Assemblies also describe the service configurations that are provided and required by the assembly. Each assembly that indicates it requires management result in JBoss ON and Katello participating in managing the instance. The Assemblies used for this solution are detailed in the following table.

Assembly Name	Templates Included	Services Provided	Services Required
load-balancer	rhel6_base, lb	load-balancer	web-ip (accepts multiples), content-management
reverse-proxy	rhel6_base, rproxy	reverse-proxy	content-management
database	rhel6_base, db	database	cloud-storage (provided by CloudFS), content-management
app-store	rhel6_base, db, app_server, app_store app_server	web-ip	database, reverse-proxy, load-balancer, content-management, JON-management

Table 5-3: Assemblies

5.2.3 Define Deployable

Now the complete solution stack can be defined as a Deployable, which is composed of Assemblies and additional meta-data. When instantiated, each Assembly is created according to specified parameters. **Table 5-4: Deployable** lists all relevant components.

Instance Name	Assemblies Included	Instance Count	Targeting Data
load-balancer01	load-balancer	1	Small instance size
reverse-proxy01	reverse-proxy	1	Medium instance size
database01	database	1	Large instance size
user-app-store01-03	app-store	3	Medium instance size

Table 5-4: Deployable

6 Detailed Architectural Workflows

This section describes the major flow of activity that Red Hat CloudForms performs when a Cloud Consumer initiates Red Hat CloudForms actions as described in **High Level Architectural Example**. The following workflow shows the high-lever overview for this process.

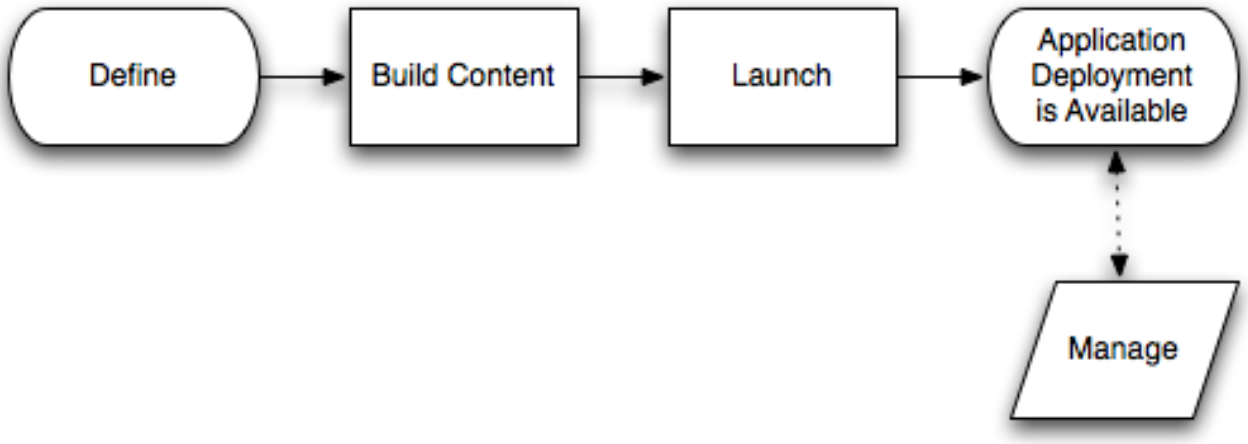

Illustration 6-1: High-level Instance Workflow

6.1 Functionality Mapping

In the previous sections functionality was described as performed by Application Description Generation, Application Lifecycle Management, Content Provision Management, Image Lifecycle Management, and Cloud Interface. In this section the actual product components from Red Hat CloudForms are used. The following illustration maps functionality to CloudForms products. Cloud Interface remains abstracted as it represents an interface to each component, and Cloud Services are invoked as needed.

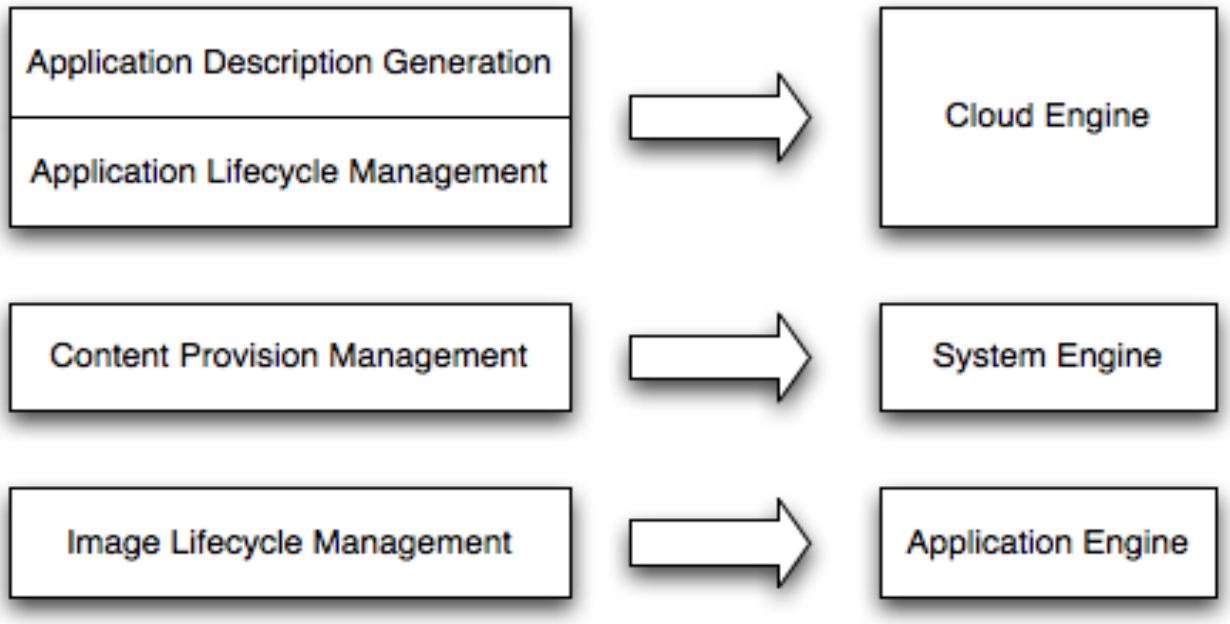

Illustration 6-2: Functionality to Product Mapping

The Product Classification maps in the following way into the architecture as represented in the following diagram.

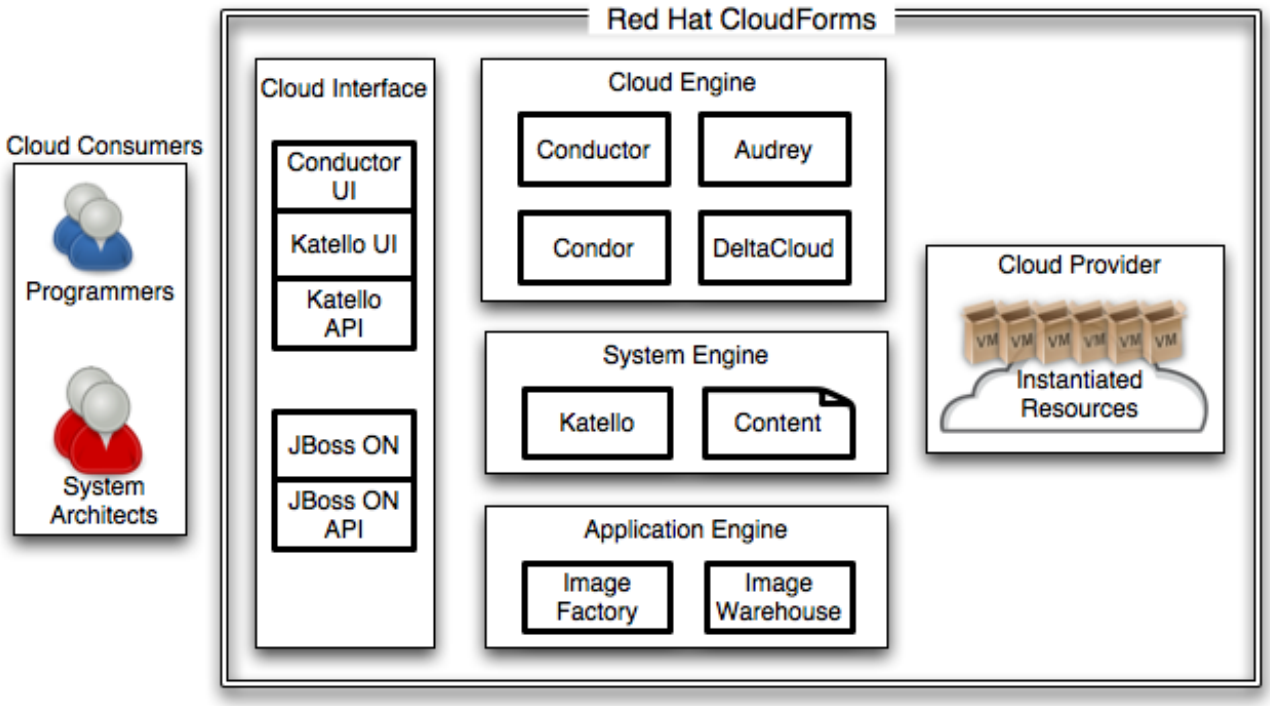

Illustration 6-3: Architectural Overview

6.2 Assumptions

The assumptions which follow are either fairly straightforward actions, or actions that are described in more detail in a future Reference Architecture.

Assumed Activities:

- All users have been created with required permissions to perform the activities attempted
- A Pool/Pool Family has been established with the account access required for the constituent cloud providers
- All infrastructure and support functions have been performed e.g., Red Hat CloudForms has been installed and configured

6.3 Define

The section **High Level Architectural Example** provides the content details of defining the Templates, Assemblies, and Deployable to specify the on-line store application.

6.3.1 Templates

Base OS

The base Operating System is chosen from a list of available pre-configured Red Hat Enterprise Linux 6 images. In this case these are Amazon EC2 Machine Images (AMI) but may be provided by the Cloud Provider. The process is shown in the following diagram which contains sequential numbers that correlate to the numbered steps that follow.

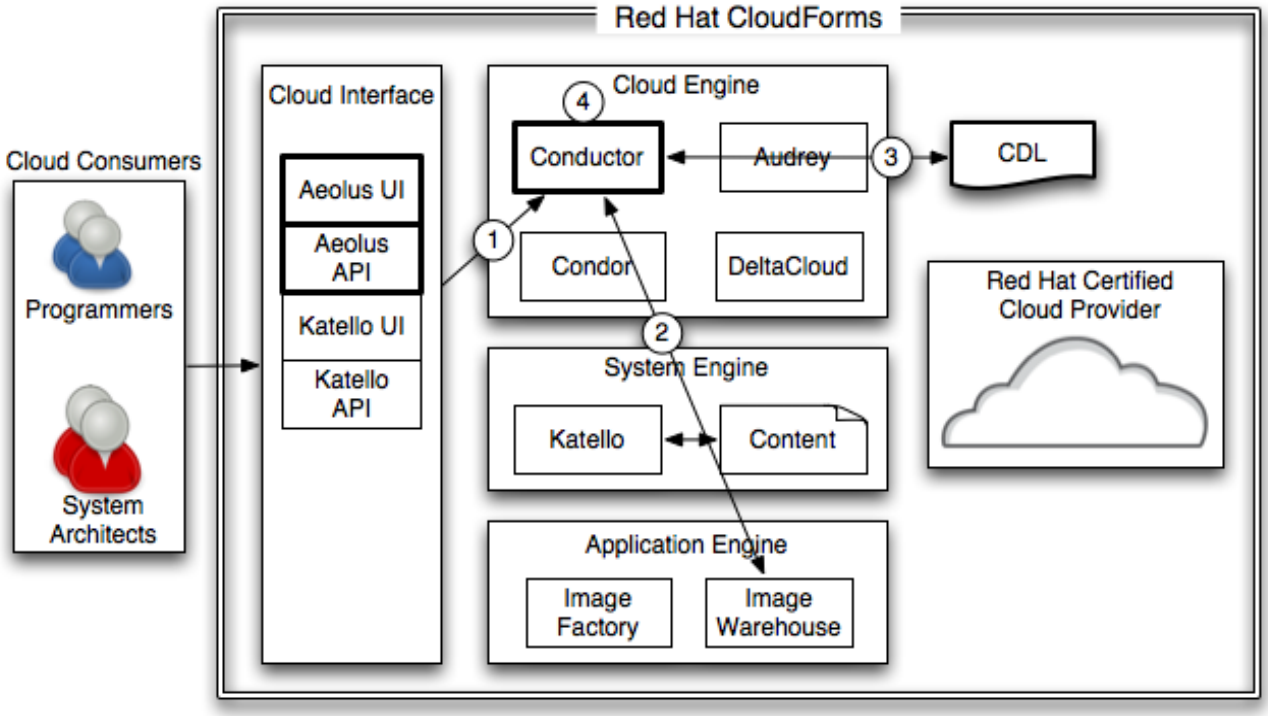

Illustration 6-4: Create AMI Template

1. Cloud Consumer specifies the creation of a new template using Red Hat Enterprise Linux 6 base OS for Amazon Web Services (AWS)
2. Conductor contacts Image Warehouse to retrieve the list of available AMIs
3. Conductor generates Template CDL based on Cloud Consumer input
4. Conductor saves Template CDL to local DB under Cloud Consumer account

Template for Non-Boot Image

The process outlined below for the load balancer Template should be repeated for each of the remaining Templates, as depicted in the illustration that follows.

- Piranha load balancer
- Squid reverse proxy
- PostgreSQL database
- JBoss Enterprise Application Platform
- Cloud consumer uploaded application

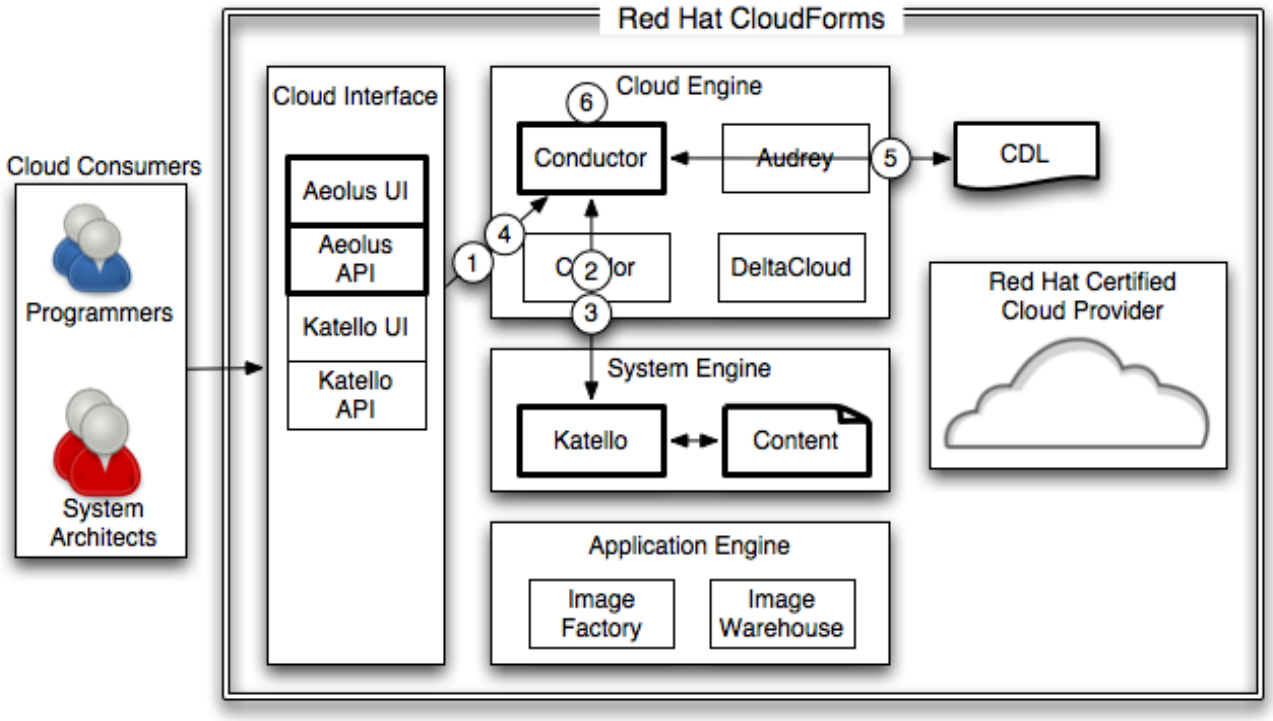

Illustration 6-5: Define Template

1. Cloud Consumer specifies the creation of a new template based on Red Hat Enterprise Linux 6
2. Conductor contacts Katello to obtain list of related available software
3. Katello provides a list of packages/software groups available
4. Cloud consumer selects “load-balancer” package group (refer to **Table 5-2: Templates**)
5. Conductor generates Template CDL based on user input
6. Conductor saves Template CDL (to local DB under Cloud Consumer account)

6.3.2 Assemblies

The process outlined below for the Load-balancer Assembly should be repeated for each of the remaining Assemblies. The process to create each Assembly is shown below.

- Reverse proxy
- Database
- App Store

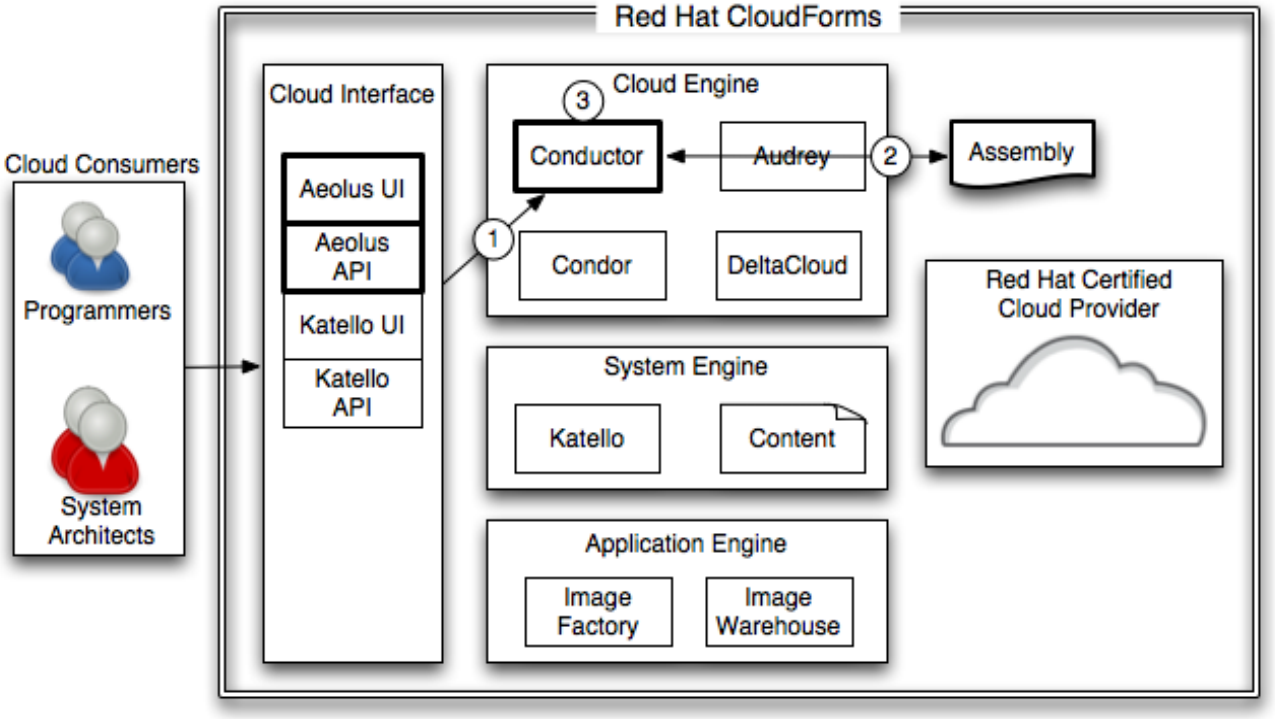

Illustration 6-6: Define Assembly

1. Cloud Consumer specifies new Load-balancer Assembly with Red Hat Enterprise Linux6 Base and Piranha as the Templates (refer to **Table 5-3: Assemblies**)
   a) Includes Red Hat Enterprise Linux 6 Base Template
   b) Includes Load Balance Template
   c) Identifies that it provides load balance Service
   d) Identifies that it requires one or more Web IP addresses
   e) Specifies it requires management
2. Conductor generates Assembly based on Cloud Consumer input
3. Conductor saves Red Hat Enterprise Linux 6 Web Server Assembly CDL to DB under Cloud Consumer account

6.3.3 Deployable

In this step the Cloud Consumer defines the overall Deployable which consists of the previously created Assemblies. The workflow is pictured next.

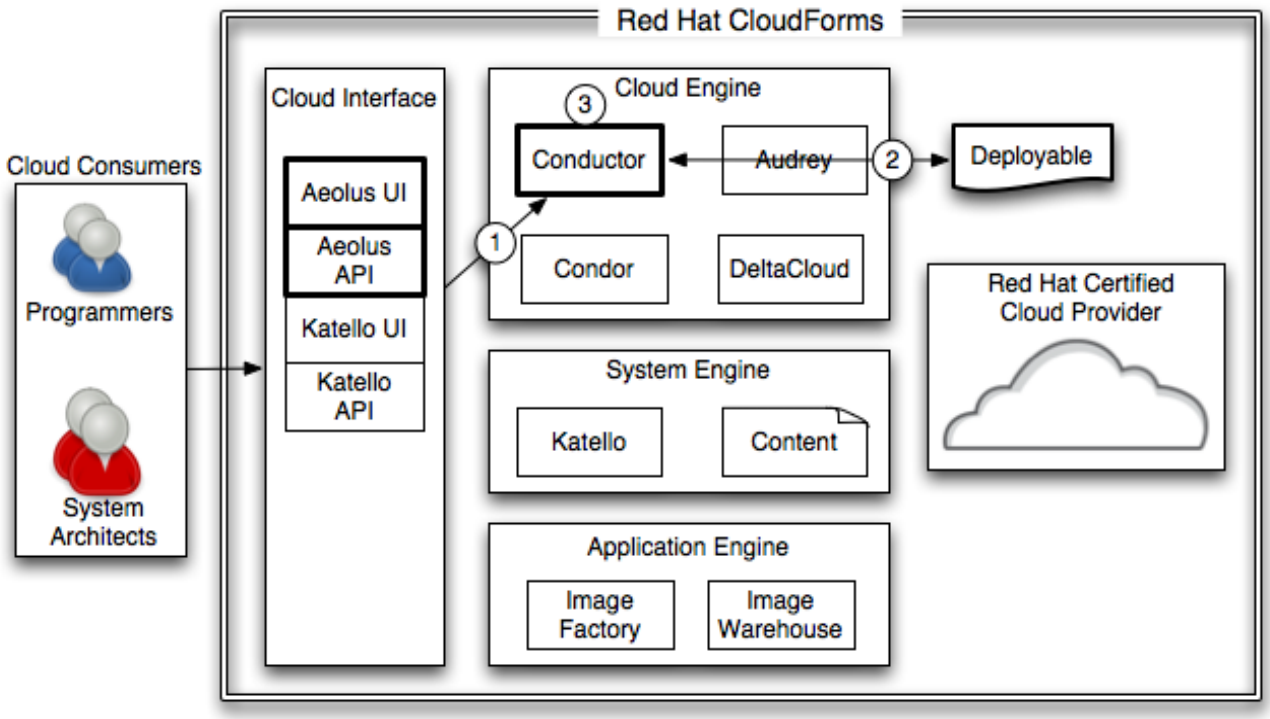

Illustration 6-7: Define Deployable

1. Cloud Consumer specifies new Deployable
   a) One instance of load-balancer Assembly with name load-balancer01 and size small
   b) One instance of reverse-proxy Assembly with name reverse-proxy01 and size medium
   c) One instance of database Assembly with name database01 of size large
   d) Three instances of app-store with name user-app-store01-03 of size medium
2. Conductor generates Red Hat Enterprise Linux 6 Web Server Deployable based on Cloud Consumer input
3. Conductor saves the Red Hat Enterprise Linux 6 Web Server Deployable CDL to DB under Cloud Consumer account

6.4 Deploy

The Cloud Consumer has planned and defined application deployment, inputting the definitions in Red Hat CloudForms. This section details the process of making the application live.

6.4.1 Build

The build process is described in the following diagram and explains how a Template recipe is made into a disk image. In our example, the Base OS is not built but provided by an Amazon EC2 AMI.

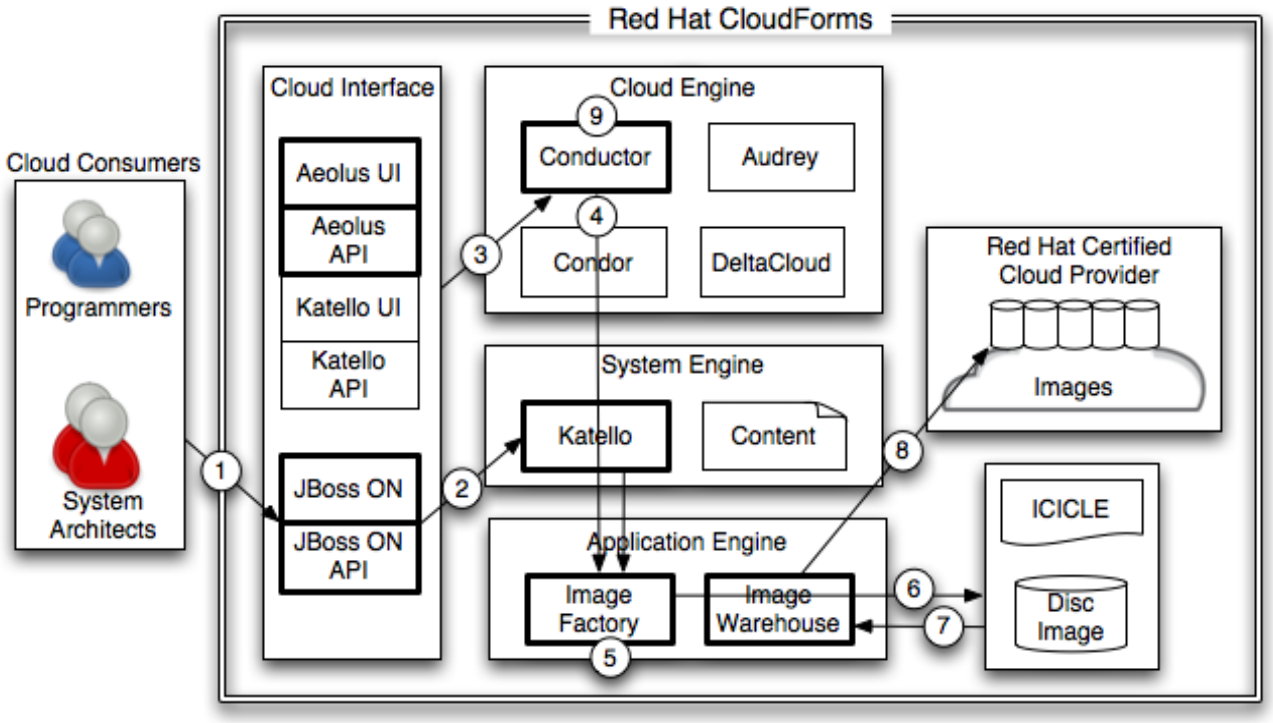

Illustration 6-8: Image Build

1. Cloud Consumer uploads JBoss bundle to JBoss ON
2. JBoss ON pushes to Katello
3. Cloud Consumer initiates build action from Conductor
4. Conductor sends message to Image Factory to build image
5. Image Factory receives request to build image

6. Image Factory calls build process
    a) creates a temporary VM
    b) uses Katello as source to build minimal VM
    c) manipulates minimal VM to allow temporary access
    d) installs remaining requested packages/software
    e) installs software and updates configuration required to support cloud environment, including any software needed for Management
    f) generates ICICLE
    g) undoes manipulation from c) that allowed temporary access
7. Image Factory pushes image, ICICLE, template to Image Warehouse
8. Image Warehouse/Image Factory prepares image for Cloud Provider and publishes
    a) Image Warehouse updates its DB
    b) Image Warehouse tells Image Factory image is ready at Cloud Provider
    c) Image Factor tells Conductor image is ready at Cloud Provider
9. Conductor updates DB and Cloud Interface

For non-bootable images a VPATH[12] install is done and the relevant directory structures are build into a disk image. In addition, the meta-data required to mount and link the disk image in much the same way alternatives are managed is recorded.

6.4.2 Instantiate

The steps to launch the on-line store deployable are pictured next. This is a detailed process which details many steps.

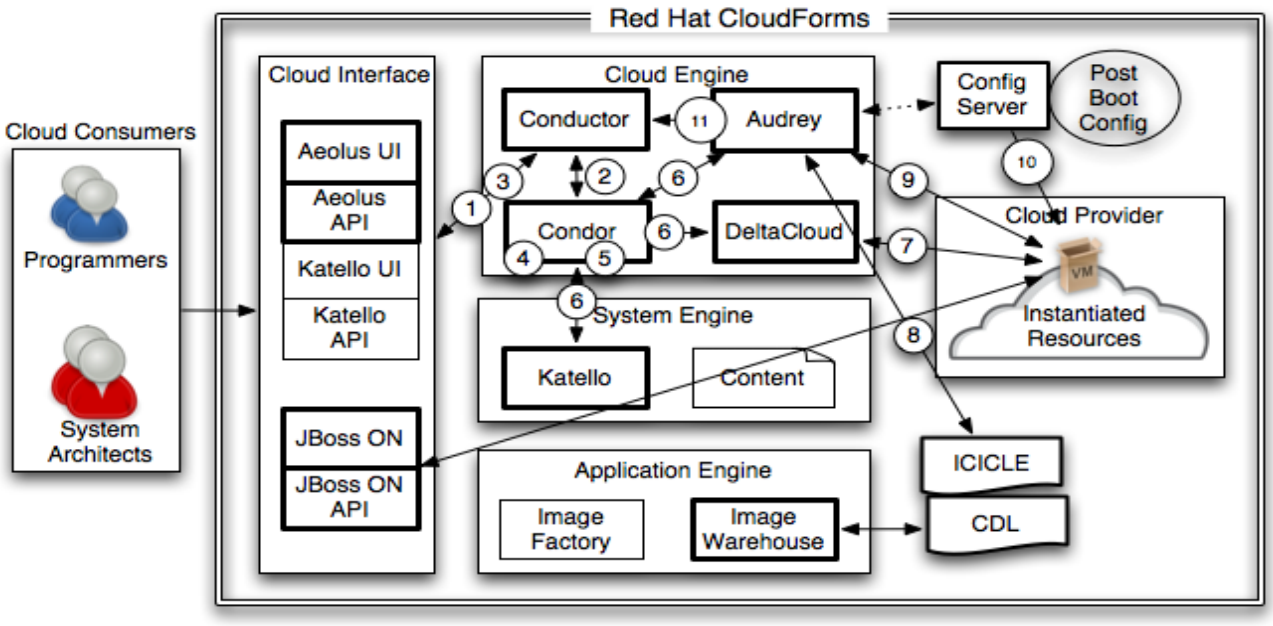

Illustration 6-9: Deployable Launch Process

1. Cloud Consumer indicates the start of a deployable in a particular pool.
2. Conductor creates a condor request to start all six instances.
   - load-balancer01
   - reverse-proxy01
   - database01
   - user-app-store01, user-app-store02, user-app-store03.
3. Cloud Consumer is prompted for any missing parameters that are required.
4. Condor accepts and queues request.

5. Condor attempts to match request using process outline in the workflow below. Condor begins by interrogating Conductor to find available cloud provider.

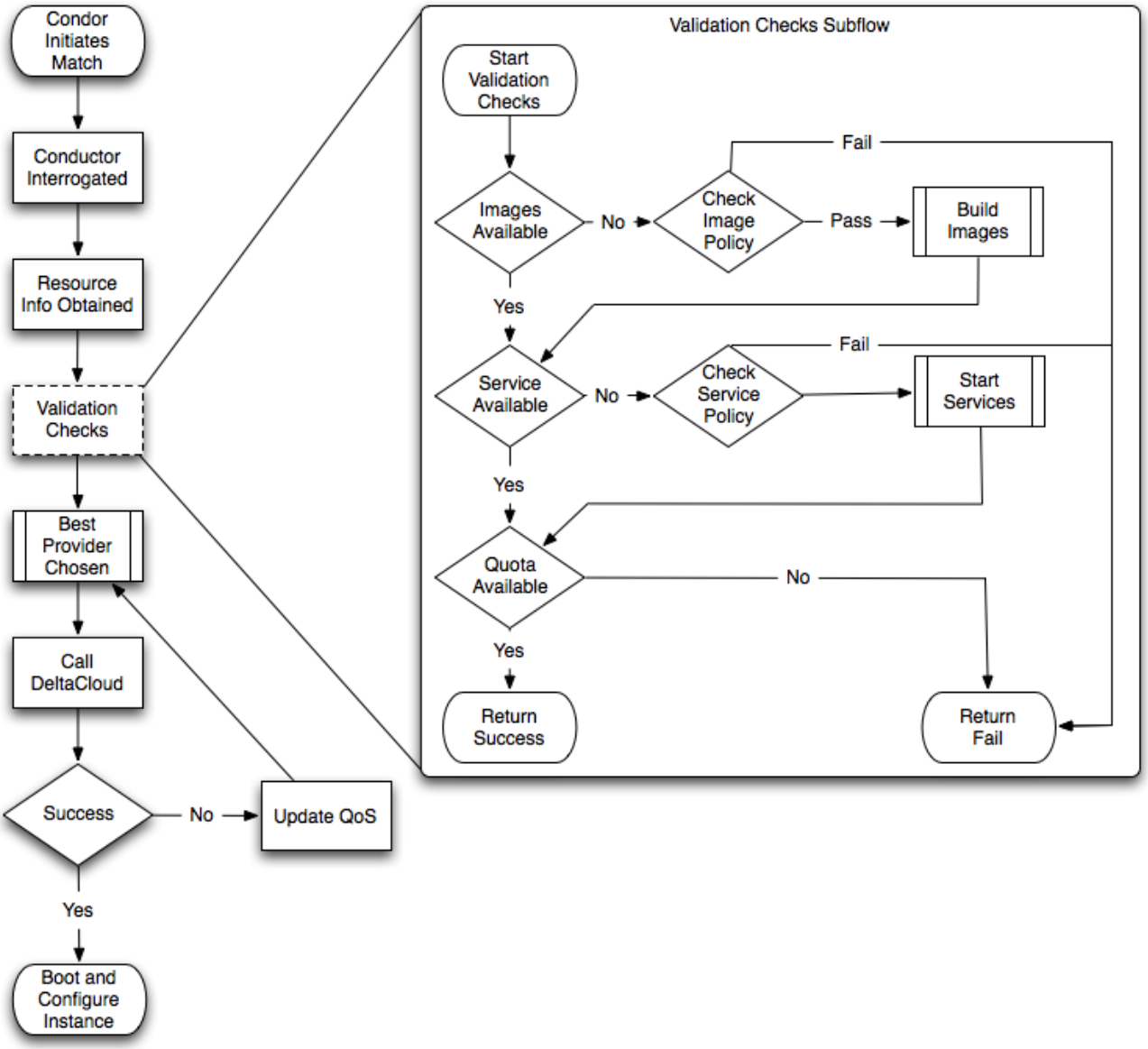

Illustration 6-10: Condor Resource Matching

6. If a match is successful, Condor informs Deltacloud to start instances and informs Audrey to configure instances, providing one time credentials, UUID, and other host identity information for each image. Condor also communicates with Katello's Candlepin[13] to reserve entitlements for each instance. This requires three actions.

7. Deltacloud receives launch requests from condor and initiates instances.

8. Audrey requests CDL and ICICLE for each instance.

9. As each instance is launched, a temporary secure connection is established using data collected from the Cloud Consumer.
    - Using the temporary secure connection, long-term identity and credentials are used for authenticated connections.
    - Required VPN connections are established.
10. Each instance provides its UUID to the Audrey configuration server, starting any remaining configuration(s). The following actions are performed, however, not necessarily in the order provided.
    - All instances
        - standard configuration including basic server and cloud specific details are retrieved from configuration server
        - standard configuration scripts are applied
        - non-bootable images are mounted and integrated
    - Load-balancer01 instance
        - waits for web-IPs configuration script from configuration server
        - applies web-IP configuration scripts
        - provides load balancer parameters to configuration server
        - informs configuration server that configuration is complete
    - reverse-proxy01 instance
        - provides reverse proxy parameters to configuration server
        - informs configuration server that configuration is complete
    - database01 instance
        - waits for CloudFS & database configuration scripts from configuration server
        - applies CloudFS configuration script
        - applies database configuration script
        - provides database server parameters to configuration server
        - informs configuration server that configuration is complete

- user-app-store01-03 instances
  - each instance starts JBoss agents
  - agents connect with JBoss ON
  - each instance provides web-IP parameters to configuration server
  - each instance waits for app-store configuration scripts which contain the database, reverse-proxy, and load-balancer parameters
  - each instance applies app-store configuration script
  - informs configuration server that configuration is complete
- CloudFS Service
  - waits for CloudFS parameters from configuration server
  - applies CloudFS configuration
  - informs configuration server that service is ready
- Audrey's configuration server
  - waits for each instance to provide parameters, which it used to generate coordinated application configuration, and is sent in the form of scripts to each instance
  - waits for configuration to complete from each instance, then prepares and transmits instance data to Conductor

11. Audrey Configuration Server transmits instance data to Conductor.

6.5 Manage

This section provides insight to the actions that can be performed on a deployment after launch. The following activities are addressed:

- Updating
- Maintaining/Suspending
- Scaling
- Migrating
- Reporting
- Business Continuity
- Eliminating

6.5.1 Updating

There are multiple targets for updating in a deployment. The most common would be errata and software updates. Others include updating the user provided application which may require additional software, cloud service updates or new offerings, or changes in the deployable's definition.

These updates can happen by three different methods controlled by user policy.

- live update – Katello/JBoss ON update running deployable
- restart of deployment – definition is updated, then redeployed
- hybrid – live update followed by an updated CDL which pushes and requires a restart

6.5.2 Maintaining/Suspending

The Cloud Consumer may desire to temporarily have their application stop processing so that modifications can be made, then allow processing to resume. This process follows the steps below:

- stop all instances of the deployment
- retain snapshot from all instances
- perform maintenance/modification
- continue instances from snapshot

6.5.3 Scaling

The Cloud Consumer may find that they wish to scale up or down their running deployment. The options available include the following:

- update deployable definition to include more or larger instances, then restart entire deployment
- using the same deployable definition, start more deployments
- update deployable definition, apply changes and condor starts/stops appropriately
- automatically increase or decrease number of instances in deployable based on capacity measurement as monitored with Matahari

6.5.4 Migrating

Once a deployment has been ruled unstable, the existing deployment is stopped and a deployment using the same definition is started at a different cloud provider.

6.5.5 Reporting

The categories of reporting relating to a running deployment include:

- application/instance/deployment status
- resource usage reports
- application specific reports (matahari agent dependent)

6.5.6 Business Continuity

Whether the Cloud Consumer is using a cloud or not, the idea of guaranteeing that data is not lost is a priority. While the methods have not been resolved as to using live snapshots, back up and archival software, or data replication, each of the following is possible.

- Point in Time backup of image and data storage – restorable to previous location
- Migration of data from one place to another
- Backup of data to a remote location, restorable to a different location

6.5.7 Eliminating

When the Cloud Consumer determines that a deployment is no longer needed, any images at the cloud provider can be discarded. The responsibility to confirm any required data has been replicated to a location that allows at-will access is up to the Cloud Consumer.

7 Architectural Operational Flexibility

The example that was previously detailed in this paper was one possible method of implementing a Cloud Consumer's need for an online store. Assuming no changes in the requirements, this section discusses alternative considerations and possibilities to accomplishing this goal. Additional considerations for cloud deployments not covered in the example are also explored.

7.1 Security, Multi-tenancy, Service Proxy

Red Hat CloudForms provides the capability for multiple Cloud Consumers to securely share a cloud provider account or to simultaneously and securely access multiple clouds as a single Cloud Consumer. Access to a Red Hat Certified Cloud Provider Public Clouds may require the use of proxies.

7.2 Alternative Deployments

The example in this paper provided one deployable definition. There are a multitude of variations including the following, but not limited to:

- using a cloud based on local virtualization (opposed to EC2)
- defining and building a base OS image
- using existing images to build new images
- do not stratify the software layers, i.e. define a single assembly per instance that has all the needed software for that instance
- have images be pulled when needed (opposed to pre-placement)
- have assemblies execute in separate clouds

8 Conclusion

In moving to the cloud or building new opportunities using a cloud infrastructure, the ownership, control, cost visibility, and decisions are moving to the domain expert ('owner' of the application). Red Hat CloudForms cloud infrastructure allows for better operational efficiency and lower total cost of ownership (TCO) for the creation and life-cycle of cloud application by enabling the domain expert.

This paper provided a high-level overview of Red Hat's new CloudForms technologies. As part of this overview several concepts were covered, such as a review of the NIST definition standards, Red Hat's cloud strategy and a description, example and workflow of a CloudForms deployment.

The key takeaway from this Reference Architecture is that Red Hat is providing the technologies to make your cloud infrastructure flexible – and flexibility means choice. This unique offering enables you to take advantage of disparate cloud providers without the overhead of having to customize the images for each environment. By providing a single user interface that interacts with technologies such as Conductor, Image Factory, Condor, etc., Red Hat is lowering the barriers to using the new cloud paradigm. If your enterprise developer knows that they can write to one API – DeltaCloud API and then be able to take advantage of multiple cloud providers, they are more likely to embrace the technology.

The following diagram depicts an over view of the Red Hat CloudForms architecture.

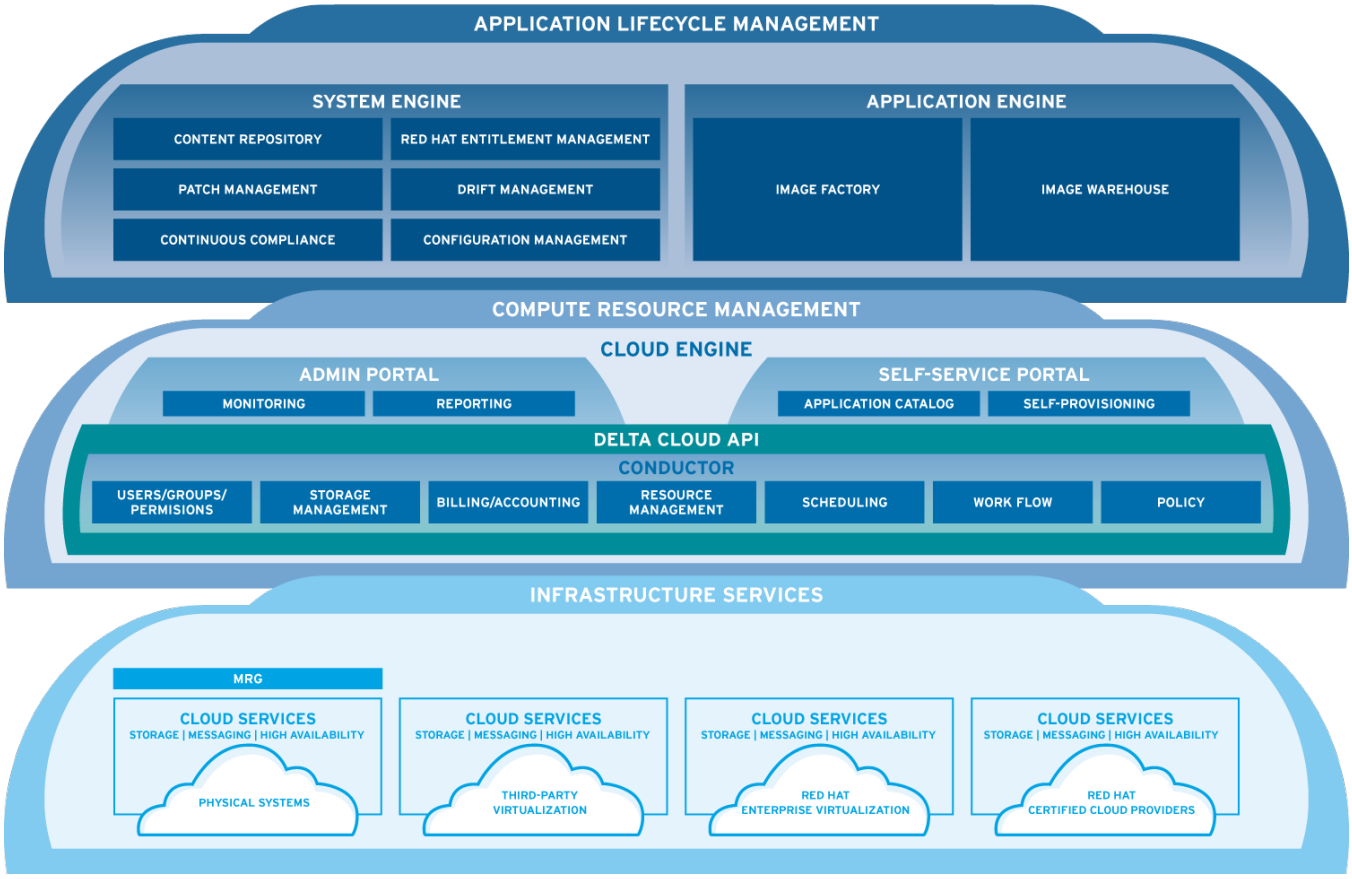

Illustration 8-1: Red Hat CloudForms Architectural Overview

Appendix A: Contributors

We would like to thank the following individuals for their time and patience as we collaborated on this process. This document would not have been possible without their many contributions.

Contributor	Title	Contribution
Vijay Trehan	Director of Solutions Architectures	Content, Diagrams, Reviews
Carl Trieloff	Technical Director, Software Engineering	Vision, Content, Reviews
John Dunning	Manager, Software Engineering	Content, Reviews
Hugh Brock	Manager, Software Engineering	Content, Reviews
Chris Lalancette	Senior Software Engineer	Diagrams, Content
Scott Collier, RHCA	Principal Software Engineer	Content, Diagrams, Reviews
Bryan Kearney	Manager, Software Engineering	Content, Reviews
Charles Crouch	Manager, Software Engineering	Content, Reviews
Jeffery Darcy	Principal Software Engineer	Content, Diagrams
Dmitri Pal	Manager, Software Engineering	Content
Steven Dake	Principal Software Engineer	Content, Diagrams
Gordon Haff	Senior Product Marketing Manager	Content, Diagrams
Brett Thurber, RHCA	Senior Software Engineer	Reviews
John Herr, RHCA	Senior Software Engineer	Reviews

Table A: Contributors

1 Appendix B: References

1 http://csrc.nist.gov/groups/SNS/cloud-computing/

2 http://collaborate.nist.gov/twiki-cloud-computing/pub/CloudComputing/Documents/Draft-SP-800-145_cloud-definition.pdf

3 http://collaborate.nist.gov/twiki-cloud-computing/pub/CloudComputing/ReferenceArchitectureTaxonomy/NIST_CC_Reference_Architecture_v1_March_30_2011.pdf

4 http://www.aeolusproject.org

5 https://github.com/matahari/matahari/wiki

6 http://www.redhat.com/mrg/messaging/

7 http://hail.wiki.kernel.org/

8 http://www.redhat.com/identity_management/

9 http://freeipa.org/

10 http://fedoraproject.org/wiki/Features/SSSD

11 http://www.clusterlabs.org/wiki/Pacemaker

12 http://www.gnu.org/s/hello/manual/automake/VPATH-Builds.html

13 https://fedorahosted.org/candlepin/wiki/Overview