



# **Reference Architectures 2017 Deploying CloudForms at Scale**

---

Peter McGowan



# Reference Architectures 2017 Deploying CloudForms at Scale

---

Peter McGowan  
pemcg@redhat.com

## Legal Notice

Copyright © 2017 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## Abstract

The purpose of this document is to provide guidelines and considerations for deploying Red Hat CloudForms 4.x to manage large-scale clouds or virtual infrastructures

## Table of Contents

<b>COMMENTS AND FEEDBACK</b>	<b>4</b>
<b>CHAPTER 1. INTRODUCTION</b>	<b>5</b>
1.1. ACKNOWLEDGEMENTS	6
<b>CHAPTER 2. ARCHITECTURE</b>	<b>7</b>
2.1. APPLIANCES	7
2.2. DATABASE	8
2.3. APPLICATION	8
2.4. PROVIDERS	8
2.5. SERVER ROLES	9
2.6. WORKERS	15
2.7. MESSAGES	19
2.8. SUMMARY OF ROLES, WORKERS AND MESSAGES	24
<b>CHAPTER 3. REGION AND ZONES</b>	<b>27</b>
3.1. REGIONS	28
3.2. ZONES	34
<b>CHAPTER 4. DATABASE SIZING AND OPTIMIZATION</b>	<b>38</b>
4.1. SIZING THE DATABASE APPLIANCE	38
4.2. SIZING THE DATABASE PARTITION BEFORE INSTALLATION	38
4.3. INSTALLATION	39
4.4. MAINTAINING PERFORMANCE	41
4.5. RESIZING THE DATABASE DIRECTORY AFTER INSTALLATION	42
<b>CHAPTER 5. INVENTORY REFRESH</b>	<b>44</b>
5.1. REFRESH OVERVIEW	44
5.2. CHALLENGES OF SCALE	45
5.3. MONITORING REFRESH PERFORMANCE	45
5.4. IDENTIFYING REFRESH PROBLEMS	47
5.5. TUNING REFRESH	48
<b>CHAPTER 6. CAPACITY &amp; UTILIZATION</b>	<b>52</b>
6.1. COMPONENT PARTS	52
6.2. DATA RETENTION	55
6.3. CHALLENGES OF SCALE	55
6.4. MONITORING CAPACITY & UTILIZATION PERFORMANCE	55
6.5. IDENTIFYING CAPACITY AND UTILIZATION PROBLEMS	56
6.6. RECOVERING FROM CAPACITY AND UTILIZATION PROBLEMS	61
6.7. TUNING CAPACITY AND UTILIZATION	62
<b>CHAPTER 7. AUTOMATE</b>	<b>64</b>
7.1. AUTOMATION ENGINE	64
7.2. CHALLENGES OF SCALE	65
7.3. IDENTIFYING AUTOMATE PROBLEMS	66
7.4. TUNING AUTOMATE	67
<b>CHAPTER 8. VM AND INSTANCE PROVISIONING</b>	<b>71</b>
8.1. STATE MACHINES	71
8.2. CHALLENGES OF SCALE	72
8.3. TUNING PROVISIONING	74
<b>CHAPTER 9. EVENT HANDLING</b>	<b>76</b>

9.1. EVENT PROCESSING WORKFLOW	76
9.2. GENERIC EVENTS	78
9.3. EVENT STORMS	79
9.4. TUNING EVENT HANDLING	80
9.5. SCALING OUT	83
<b>CHAPTER 10. SMARTSTATE ANALYSIS</b>	<b>84</b>
10.1. PROVIDER-SPECIFIC CONSIDERATIONS	84
10.2. MONITORING SMARTSTATE ANALYSIS	86
10.3. CHALLENGES OF SCALE	87
10.4. TUNING SMARTSTATE ANALYSIS	89
<b>CHAPTER 11. WEB USER INTERFACE</b>	<b>91</b>
11.1. SCALING WORKERS	91
11.2. SCALING APPLIANCES	91
<b>CHAPTER 12. MONITORING</b>	<b>93</b>
12.1. DATABASE APPLIANCE	93
12.2. CFME 'WORKER' APPLIANCES	93
12.3. ALERTS	95
12.4. CONSOLIDATED LOGGING	97
<b>CHAPTER 13. DESIGN SCENARIO</b>	<b>98</b>
13.1. ENVIRONMENT TO BE MANAGED	98
13.2. DESIGN PROCESS	99
13.3. INITIAL DEPLOYMENT	104
13.4. PROVISIONING WORKFLOW	104
<b>CHAPTER 14. CONCLUSION</b>	<b>106</b>
<b>APPENDIX A. DATABASE APPLIANCE CPU COUNT</b>	<b>107</b>
<b>APPENDIX B. CONTRIBUTORS</b>	<b>108</b>
<b>APPENDIX C. REVISION HISTORY</b>	<b>109</b>



## COMMENTS AND FEEDBACK

In the spirit of open source, we invite anyone to provide feedback and comments on any reference architecture. Although we review our papers internally, sometimes issues or typographical errors are encountered. Feedback allows us to not only improve the quality of the papers we produce, but allows the reader to provide their thoughts on potential improvements and topic expansion to the papers. Feedback on the papers can be provided by emailing [refarch-feedback@redhat.com](mailto:refarch-feedback@redhat.com). Please refer to the title within the email.



# CHAPTER 1. INTRODUCTION

This document discusses the challenges of deploying CloudForms *at scale* to manage large virtual infrastructures or clouds. The term "at scale" in this case infers several thousand managed virtual machines, instances, templates, clusters, hosts, datastores, containers or pods.

Unfortunately there is no magic formula to describe the various maximum sizes, the number of CloudForms appliances and workers that will be required, nor the number of regions or placement of zones. The diverse nature of the many types of provider and their various workload characteristics makes generalization difficult, and at best misleading.

Experience shows that the most effective way to deploy CloudForms in large environments is to start with a minimal set of features enabled, and go through an iterative process of monitoring, tuning and expanding. Understanding the architecture of the product and the various components is an essential part of this process. Although by default a CloudForms Management Engine (CFME) appliance is tuned for relatively small environments, the product *is* scalable to manage many thousands of virtual machines, instances or containers. Achieving this level of scale however generally requires some customization of the core components for the specific environment being managed. This might include increasing the virtual machine resources such as vCPUs and memory, or tuning the CFME workers; their numbers, placement, or memory thresholds for example.

This guide seeks to explain the architecture of CloudForms, and expose the inner workings of the core components. Several 'rules of thumb' such as guidelines for CFME appliance to VM ratios are offered, along with the rationale behind the numbers, and when they can be adjusted. The principal source of monitoring and tuning data is the *evm.log* file, and many examples of log lines for various workers and strings to search for have been included, along with sample scripts to extract real-time timings for activities such as EMS refresh.

The document is divided into three sections, as follows:

## Part I - Architecture and Design

- ✳ *Architecture* discusses the principal architectural components that influence scaling: appliances, server roles, workers and messages.
- ✳ *Regions and Zones* discusses the considerations and options for region and zone design.
- ✳ *Database Sizing and Optimization* presents some guidelines for sizing and optimizing the PostgreSQL database for larger-scale operations.

## Part II - Component Scaling

- ✳ *Inventory Refresh* discusses the mechanism of extracting and saving the inventory of objects - VMs, hosts or containers for example - from an external management system.
- ✳ *Capacity and Utilization* explains how the three types of C&U worker interact to extract and process performance metrics from an external management system.
- ✳ *Automate* describes the challenges of scaling Ruby-based automate workflows, and how to optimize automation methods for larger environments.
- ✳ *Provisioning* focuses on virtual machine and instance provisioning, and the problems that sometimes need to be addressed when complex automation workflows interact with external enterprise tools.
- ✳ *Event Handling* describes the three workers that combine to process events from external management systems, and how to scale them.

- ✦ *SmartState Analysis* takes a look at some of the tuning options available to scale SmartState Analysis in larger environments.
- ✦ *Web User Interface* discusses how to scale WebUI appliances behind load balancers.
- ✦ *Monitoring* describes some of the in-built monitoring capabilities, and how to setup alerts to warn of problems such as workers being killed.

### Part III - Design Scenario

- ✦ *Region Design Scenario* takes the reader through a realistic design scenario for a large single region comprising several provider types.

## 1.1. ACKNOWLEDGEMENTS

The author would particularly like to thank Tom Hennessy and Bill Helgeson of Red Hat for their patience, knowledge and advice when preparing this document.

## CHAPTER 2. ARCHITECTURE

In order to understand how to deploy CloudForms at scale, it is important to understand the architectural components that affect the design and deployment decisions. These principal components are described in this chapter.

### 2.1. APPLIANCES

To simplify installation, the Red Hat CloudForms product is distributed as a self-contained virtual machine template, which when cloned becomes a CloudForms Management Engine (CFME) *appliance*. Each release of the Red Hat CloudForms product since v2.0 has had a corresponding CloudForms Management Engine release, although the version numbers are not the same (for historical reasons). The following table summarizes the relative CFME and CloudForms product versions.

**Table 2.1. Summary of the relative CFME and CloudForms product versions**

CloudForms Management Engine version	CloudForms (Product) version
5.1	2.0
5.2	3.0
5.3	3.1
5.4	3.2
5.5	4.0
5.6	4.1
5.7	4.2
5.8	4.5

A CFME 5.8 (CloudForms 4.5) appliance runs Red Hat Enterprise Linux 7.3, with PostgreSQL 9.5, Rails 5.0.2, the CloudForms evmservd service, and all associated Ruby gems installed. A new addition with **CFME 5.8 is the Embedded Ansible 3.1 automation manager**, also packaged with the appliance.

The self-contained nature of appliances makes them ideal for horizontally scaling a CloudForms deployment to handle the increased load that larger clouds or virtual infrastructures present.

Appliances are downloadable as images or templates in formats suitable for VMware, Red Hat Virtualization, OpenStack, Amazon EC2, Microsoft's System Center Virtual Machine Manager or Azure cloud, and Google Compute Engine. The most recent versions can also be installed as an OpenShift 3.x container image (although this platform is a technology preview).

## 2.2. DATABASE

A CloudForms region stores all of its data in a PostgreSQL database. This is known as the *Virtual Management Database* or *VMDB*, although the terms "database" and "VMDB" are often used interchangeably. The database can be internal and integral with an appliance running several other roles (typical for smaller CloudForms deployments), but for larger CloudForms deployments it is typically a dedicated database server or cluster configured for high availability and disaster recovery.

## 2.3. APPLICATION

CloudForms is a Ruby on Rails application. The main *miq\_server.rb* Rails application is supported by a number of worker processes that perform the various interactions with managed systems, or collect and analyse data.

## 2.4. PROVIDERS

CloudForms manages each cloud, container or virtual environment using modular subcomponents called providers. Each provider contains the classes and modules required to connect to and manage its specific target platform, and this provider specialization enables common functionality to be abstracted by provider type or class. CloudForms acts as a "manager of managers", and in keeping with this concept, providers communicate with their respective underlying cloud or infrastructure platform manager (such as vCenter server or RHV-M) using the native APIs published for the platform manager. A provider's platform manager is referred to as an *External Management System* or *EMS*.



### Note

Although the terms *provider* and *external management system (EMS)* are often used interchangeably, there is an important distinction. The *provider* is the CloudForms component, whereas the *EMS* is the managed entity that the provider connects to, such as the VMware vCenter

Providers are broadly divided into categories, and in CloudForms 4.5 these are Cloud, Infrastructure, Container, Configuration Management, Automation, Network, Middleware and Storage.<sup>[1]</sup>

### 2.4.1. Provider Namespaces

Many provider components are named according to a name-spacing schema that follows the style of:

```
ManageIQ::Providers::<ProviderName>::<ProviderCategory>
```

Some examples of this are as follows:

- ✚ `ManagelQ::Providers::EmbeddedAnsible::AutomationManager`
- ✚ `ManagelQ::Providers::OpenshiftEnterprise::ContainerManager`
- ✚ `ManagelQ::Providers::Openstack::CloudManager`
- ✚ `ManagelQ::Providers::Openstack::InfraManager`
- ✚ `ManagelQ::Providers::Azure::NetworkManager`
- ✚ `ManagelQ::Providers::StorageManager::CinderManager`
- ✚ `ManagelQ::Providers::Vmware::InfraManager`

## 2.5. SERVER ROLES

A CloudForms Management Engine 5.8 appliance can be configured to run up to 19 different server roles. These are enabled or disabled in the server **Configuration** section of the WebUI (see [Figure 2.1, “Server Roles”](#)).

**Figure 2.1. Server Roles**

Server Roles	<input checked="" type="checkbox"/> On	Automation Engine
	<input checked="" type="checkbox"/> On	Capacity & Utilization Coordinator
	<input checked="" type="checkbox"/> On	Capacity & Utilization Data Collector
	<input checked="" type="checkbox"/> On	Capacity & Utilization Data Processor
	<input type="checkbox"/> Off	Database Operations
	<input checked="" type="checkbox"/> On	Embedded Ansible
	<input checked="" type="checkbox"/> On	Event Monitor
	<input type="checkbox"/> Off	Git Repositories Owner
	<input checked="" type="checkbox"/> On	Notifier
	<input checked="" type="checkbox"/> On	Provider Inventory
	<input checked="" type="checkbox"/> On	Provider Operations
	<input type="checkbox"/> Off	RHN Mirror
	<input type="checkbox"/> Off	Reporting
	<input checked="" type="checkbox"/> On	Scheduler
	<input checked="" type="checkbox"/> On	SmartProxy
	<input checked="" type="checkbox"/> On	SmartState Analysis
	<input checked="" type="checkbox"/> On	User Interface
	<input checked="" type="checkbox"/> On	Web Services
	<input checked="" type="checkbox"/> On	Websocket

Server roles are implemented by worker processes (see [Section 2.6, “Workers”](#)), many of which receive work instructions from messages (see [Section 2.7, “Messages”](#)).

### 2.5.1. Automation Engine

The *Automation Engine* role enables a CFME appliance to process queued automation tasks<sup>[2]</sup>. There **should be at least one CFME appliance with this role set in each zone**. The role does not have a dedicated worker, automate tasks are processed by either a *MiqGenericWorker* or a *MiqPriorityWorker*, depending on message priority.



#### Note

The Automation Engine also handles the processing of events through the *automate event switchboard*

### 2.5.2. Capacity and Utilization

Capacity and utilization (C&U) metrics processing is a relatively resource-intensive operation, and there are three roles associated with its operation.

- ✦ The *Capacity & Utilization Coordinator* role acts as a scheduler for the collection of C&U data in a zone, and queues work for the Capacity and Utilization Data Collector. If more than one CFME appliance in a zone has this role enabled, only one will be active at a time. This role does not have a dedicated worker, **the C&U Coordinator tasks are processed by either a *MiqGenericWorker* or a *MiqPriorityWorker*, depending on message priority.**
- ✦ The *Capacity & Utilization Data Collector* performs the actual collection of C&U data. This role has a dedicated worker, and there is no limit to the number of concurrent workers in a zone. Enabling this role starts the provider-specific data collector workers for any providers in the appliance's zone. For example a CFME appliance in a zone configured with a Red Hat Virtualization provider would contain one or more *ManageIQ::Providers::Redhat::InfraManager::MetricsCollectorWorker* processes if the C&U Data Collector server role was enabled.
- ✦ The *Capacity & Utilization Data Processor* processes all of the data collected, allowing CloudForms to create charts, display utilization statistics, etc.. This role has a dedicated worker called the *MiqEmsMetricsProcessorWorker*, and there is no limit to the number of concurrent workers in a zone.



#### Note

The Capacity & Utilization roles are described in more detail in [Chapter 6, Capacity & Utilization](#)

### 2.5.3. Database Operations

The *Database Operations* role enables a CFME appliance to run certain database maintenance tasks such as purging old metrics. This role does not have a dedicated worker, the database operations tasks are processed by a *MiqGenericWorker*.

### 2.5.4. Embedded Ansible

The *Embedded Ansible* role enables the use of the built-in Ansible automation manager, which allows Ansible playbooks to be run from service catalogs, or from control actions and alerts. If more than one CFME appliance in a region has this role enabled, only one will be active at a time. This role has a dedicated worker called the *EmbeddedAnsibleWorker*, but enabling the role also starts the following event catcher and refresh workers:

✱ *ManageIQ::Providers::EmbeddedAnsible::AutomationManager::EventCatcher*

✱ *ManageIQ::Providers::EmbeddedAnsible::AutomationManager::RefreshWorker*



#### Note

Enabling the Embedded Ansible role adds approximately 2GBytes to the memory requirements of a CFME appliance

### 2.5.5. Event Monitor

The *Event Monitor* role is responsible for detecting and processing provider events such as a VM starting or stopping, a cloud instance being created, or a hypervisor rebooting. Enabling the role starts at least 2 workers; one or more provider-specific, and one common event handler.

The provider-specific event catcher maintains a connection to a provider's event source (such as the Google Cloud Pub/Sub API for Google Compute Engine) and detects or 'catches' events and passes them to the common event handler. An event catcher worker is started for each provider in the appliance's zone; a zone containing a VMware provider would contain a *ManageIQ::Providers::Vmware::InfraManager::EventCatcher* worker, for example.

Some cloud providers automatically add several types of manager, and these might each have an event catcher worker. To illustrate this, enabling the event monitor role on an appliance in an OpenStack Cloud provider zone would start the following event catcher workers:

✱ *ManageIQ::Providers::Openstack::CloudManager::EventCatcher*

✱ *ManageIQ::Providers::Openstack::NetworkManager::EventCatcher*

✱ *ManageIQ::Providers::StorageManager::CinderManager::EventCatcher*

The event handler worker, called *MiqEventHandler*, is responsible for feeding the events from all event catchers in the zone into the automation engine's event switchboard for processing.

There should be at least one CFME appliance with the event monitor role set in any zone containing a provider, however if more than one CFME appliance in a zone has this role, only one will be active at a time.



#### Note

The event catcher and event handler workers are described in more detail in [Chapter 9, Event Handling](#)

### 2.5.6. Git Repositories Owner

A CFME appliance with the *Git Repositories Owner* role enabled is responsible for synchronising git repository data from a git source such as Github or Gitlab, and making it available to other appliances in the region that have the automation engine role set. The git repository data is copied

to `/var/www/miq/vmdb/data/git_repos/<git_profile_name>/<git_repo_name>` on the CFME appliance. This role does not have a dedicated worker.

### 2.5.7. Notifier

The *Notifier* role should be enabled if CloudForms is required to forward SNMP traps to a monitoring system, or to send e-mails. These might be initiated by an automate method or from a control policy, for example.

If more than one CFME appliance in a region has this role enabled, only one will be active at a time. This role does not have a dedicated worker, notifications are processed by either a *MiqGenericWorker* or a *MiqPriorityWorker*, depending on message priority.

### 2.5.8. Provider Inventory

The *Provider Inventory* role is responsible for refreshing provider inventory data for all provider objects such as virtual machines, hosts, clusters, tenants, or orchestration templates. It is also responsible for capturing datastore file lists. If more than one CFME appliance in a zone has this role enabled, only one will be active at a time.

Setting this role starts the provider-specific refresh workers for any providers in the appliance's zone; a zone containing a RHV provider would contain a *ManageIQ::Providers::Redhat::InfraManager::RefreshWorker* worker, for example.

VMware providers add an additional *MiqEmsRefreshCoreWorker*, while cloud providers that use several types of manager add a worker per manager. For example enabling the Provider Inventory role on an appliance in an Azure provider zone would start the following Refresh workers:

➤ *ManageIQ::Providers::Azure::CloudManager::RefreshWorker*

➤ *ManageIQ::Providers::Azure::NetworkManager::RefreshWorker*



#### Note

Provider Inventory refresh workers are described in more detail in [Chapter 5, Inventory Refresh](#)

### 2.5.9. Provider Operations

A CFME appliance with the *Provider Operations* role performs certain managed object operations such as stop, start, suspend, shutdown guest, clone, reconfigure, etc., to provider objects such as VMs. These operations might be initiated from the WebUI, from Automate, or from a REST call. It also handles some storage-specific operations such as creating cloud volume snapshots. The role does not have a dedicated worker, provider operations tasks are processed by either a *MiqGenericWorker* or a *MiqPriorityWorker*, depending on message priority. There is no limit to the number of concurrent workers handling this role in a zone.



**Note**

The Provider Operations role is often required in zones that don't necessarily contain providers.

For example, enabling the Provider Operations role in a WebUI zone can improve performance by reducing the number of individual EMS connections required for user-initiated VM operations, in favour of a single brokered connection. The Provider Operations role is also required in any zone that may run service-initiated VM provisioning operations.

**2.5.10. RHN Mirror**

A CFME appliance with the *RHN Mirror* role acts as a repository server for the latest CloudForms Management Engine RPM packages. It also configures other CFME appliances within the same region to point to itself for updates. This provides a low bandwidth method to update environments with multiple appliances. The role does not have a dedicated worker.

**2.5.11. Reporting**

The *Reporting* role allows a CFME appliance to generate reports. There should be at least one CFME appliance with this role in any zone in which reports are automatically scheduled or manually requested/queued.<sup>[3]</sup> (such as from a WebUI zone).

Enabling this server role starts one or more *MiqReportingWorker* workers.

**2.5.12. Scheduler**

The *Scheduler* sends messages to start all scheduled activities such as report generation, database backups, or to retire VMs or services. One server in each region must be assigned this role or scheduled CloudForms events will not occur. Enabling this server role starts the *MiqScheduleWorker* worker.

**Note**

Each CFME appliance also has a schedule worker running but this only handles local appliance task scheduling.

The Scheduler role is for region-specific scheduling and is only active on one appliance per region.

**2.5.13. SmartProxy**

Enabling the *SmartProxy* role turns on the embedded SmartProxy on the CFME appliance. The embedded SmartProxy can analyse virtual machines that are registered to a host and templates that are associated with a provider. Enabling this role starts three *MiqSmartProxyWorker* workers.

**2.5.14. SmartState Analysis**

The *SmartState Analysis* role controls which CFME appliances can control SmartState Analyses and process the data from the analysis. There should be at least one of these in each zone that contains a provider. This role does not have a dedicated worker. SmartState tasks are processed by either a *MiqGenericWorker* or a *MiqPriorityWorker*, depending on message priority.



#### Note

The SmartProxy and SmartState Analysis roles are described in more detail in [Chapter 10, SmartState Analysis](#)

### 2.5.15. User Interface

This role enables access to a CFME appliance using the Red Hat CloudForms *Operations* WebUI console. More than one CFME appliance can have this role in a zone (the default behaviour is to have this role enabled on all appliances). Enabling this server role starts one or more *MiqUiWorker* workers.



#### Note

The use of multiple WebUI appliances in conjunction with load balancers is described in more detail in [Chapter 11, Web User Interface](#)

### 2.5.16. Web Services

This role enables the RESTful Web service API on a CFME appliance. More than one CFME appliance can have this role in a zone. Enabling this server role starts one or more *MiqWebServiceWorker* workers.



#### Note

The Web Services role is required by the Self-Service User Interface (SSUI). Both the User Interface and Web Services roles must be enabled on a CFME appliance to enable logins to the Operations WebUI

### 2.5.17. Websocket

This role enables a CFME appliance to be used as a websocket proxy for the VNC and SPICE HTML5 remote access consoles. It is also used by the WebUI notification service. Enabling this server role starts one or more *MiqWebsocketWorker* workers.

### 2.5.18. Server Role Zone Affinity

Many server roles - or more accurately their worker processes - have an affinity to the zone with which the hosting CFME appliance is associated. For example messages intended for zone "A" will generally not be processed by worker processes in zone "B".

The following server roles have zone affinity:

- ✳ C&U Metrics Coordinator

- ✱ C&U Metrics Collector
- ✱ C&U Metrics Processor
- ✱ Event Monitor
- ✱ Git Repositories Owner
- ✱ Provider Inventory
- ✱ Provider Operations
- ✱ SmartProxy
- ✱ SmartState Analysis



### Note

Some server roles such as Automation Engine have optional zone affinity. If an automate message specifies the zone to be run in, the task will only be processed in that zone. If an automate message doesn't specify the zone, the task can run anywhere.

## 2.6. WORKERS
















As can be seen, many of the server roles start worker processes. The currently running worker processes on a CFME appliance can be viewed using the following commands in a root bash shell on an appliance:

```
vmdb
bin/rake evm:status
```

The same information can also be seen in the **Workers** tab of the **Configuration** → **Diagnostics** page (see [Figure 2.2, “Worker Processes”](#)).

**Figure 2.2. Worker Processes**

Summary **Workers** Collect Logs Utilization Timelines

	Name	Status	PID	SPID	URI / Queue Name
	C&U Metrics Collector for Openstack	started	7847	24744	openstack
	C&U Metrics Collector for Openstack Network	started	7855	24745	openstack_network
	C&U Metrics Collector for OpenstackInfra	started	7860	24746	openstack_infra
	C&U Metrics Processor	started	7879	24748	ems_metrics_processor
	C&U Metrics Processor	started	7868	24747	ems_metrics_processor
	Event Handler	started	7964	24757	ems
	Event Monitor for Providers: Openstack director	started	7955	24756	ems_7
	Event Monitor for Providers: Openstack director Network Manager	started	7943	24755	ems_8
	Event Monitor for Providers: OpenStack Stage Cinder Manager	started	8992	24931	ems_1
	Generic Worker	started	2982	41579	generic
	Generic Worker	started	2974	41578	generic
	Priority Worker	started	2990	41580	generic
	Priority Worker	started	2998	41581	generic
	Refresh Worker for Providers: Openstack director	started	7915	24752	ems_7
	Refresh Worker for Providers: Openstack director Network Manager	started	7905	24751	ems_8

### Note

CFME 5.8 has provided a new command that allows the currently running worker processes on the local server *and* remote servers can be seen, ordered by server and zone:

```
vmdb
bin/rake evm:status_full
```

In addition to the workers started by enabling a server role, each appliance has by default four workers that handle more generic tasks: two *MiqGenericWorkers* and two *MiqPriorityWorkers*. The *MiqPriorityWorkers* handle the processing of the highest priority messages (priority 20) in the *generic* message queue (see [Section 2.7, “Messages”](#)).

Generic and Priority workers process tasks for the following server roles:

- ✎ Automate
- ✎ C&U Coordinator
- ✎ Database Operations
- ✎ Notifier
- ✎ Provider Operations
- ✎ SmartState Analysis

## 2.6.1. Worker Validation

Monitoring the health status of workers becomes important as a CloudForms installation is scaled. A server thread called `validate_worker` checks that workers are alive (they have recently issued a 'heartbeat' ping.<sup>[4]</sup>), and are within their time limits and memory thresholds. Some workers such as Refresh and SmartProxy workers have a maximum lifetime of 2 hours to restrict their resource consumption.<sup>[5]</sup> If this time limit is exceeded, the `validate_worker` thread will instruct the worker to exit at the end of its current message processing, and spawn a new replacement.

The following `evm.log` line shows an example of the normal timeout processing for a RefreshWorker:

```
INFO -- : MIQ(MiqServer#validate_worker) Worker ↵
[ManageIQ::Providers::Vmware::InfraManager::RefreshWorker] ↵
with ID: [1000000258651], PID: [17949], ↵
GUID: [77362eba-c179-11e6-aaa4-00505695be62] uptime has reached ↵
the interval of 7200 seconds, requesting worker to exit
```

The following log line shows an example of an abnormal exit request for a `MiqEmsMetricsProcessorWorker` that has exceeded its memory threshold (see [Section 2.6.2.1, “Worker Memory Thresholds”](#)):

```
WARN -- : MIQ(MiqServer#validate_worker) Worker
[MiqEmsMetricsProcessorWorker] ↵
with ID: [1000000259290], PID: [15553], ↵
GUID: [40698326-c18a-11e6-aaa4-00505695be62] process memory usage
[598032000] ↵
exceeded limit [419430400], requesting worker to exit
```

### Tip

The actions of `validate_worker` can be examined in `evm.log` by using the following command:

```
grep 'MiqServer#validate_worker' evm.log
```

Use this command to check for workers exceeding their memory allocation.

## 2.6.2. Tuning Workers

It is often a requirement to tune the number of per-appliance workers and their memory thresholds when CloudForms is deployed to manage larger clouds or virtual infrastructures.

### 2.6.2.1. Worker Memory Thresholds

Each worker type is given an out-of-the-box initial memory threshold. The default values have been chosen to perform well with an 'average' workload, but these sometimes need to be increased, depending on the runtime requirements of the specific CloudForms installation.

### 2.6.2.2. Adjusting Worker Settings

The count and maximum memory thresholds for most worker types can be tuned from the CloudForms WebUI, in the **Workers** tab of the **Configuration** → **Settings** page for each appliance (see [Figure 2.3, “Worker Settings”](#)).

**Figure 2.3. Worker Settings**

Server	Authentication	Workers	Custom Logos	Advanced
<div>Generic Workers</div> <div>Count <input type="text" value="2"/></div> <div>Memory threshold <input type="text" value="800 MB"/></div>				
<div>Priority Workers</div> <div>Count <input type="text" value="2"/></div> <div>Memory threshold <input type="text" value="800 MB"/></div>				
<div>C &amp; U Data Collectors</div> <div>Count <input type="text" value="2"/></div> <div>Memory threshold <input type="text" value="200 MB"/></div>				
<div>C &amp; U Data Processors</div> <div>Count <input type="text" value="2"/></div> <div>Memory threshold <input type="text" value="600 MB"/></div>				
<div>Event Monitor</div> <div>Memory threshold <input type="text" value="2 GB"/></div>				
<div>Refresh</div> <div>Memory threshold <input type="text" value="2 GB"/></div>				
<div>Connection Broker</div> <div>Memory threshold <input type="text" value="2 GB"/></div>				
<div>VM Analysis Collectors</div> <div>Count <input type="text" value="3"/></div> <div>Memory threshold <input type="text" value="600 MB"/></div>				
<div>UI Worker</div> <div>Count <input type="text" value="1"/></div>				
<div>Websocket Workers</div> <div>Count <input type="text" value="1"/></div>				
<div>Reporting Workers</div> <div>Count <input type="text" value="2"/></div> <div>Memory threshold <input type="text" value="400 MB"/></div>				
<div>Web Service Workers</div> <div>Count <input type="text" value="1"/></div> <div>Memory threshold <input type="text" value="1 GB"/></div>				

For other workers not listed in this page, the memory threshold settings can be tuned (with caution) in the **Configuration** → **Advanced** settings by directly editing the YAML, for example:

```
:workers:
  :worker_base:
  ...
  :ui_worker:
    :connection_pool_size: 8
    :memory_threshold: 1.gigabytes
    :nice_delta: 1
    :count: 1
```

### 2.6.3. Worker Task Allocation

Tasks are dispatched to the various workers in one of three ways:

1. From a scheduled timer. Some tasks are completely synchronous and predictable, and these are dispatched from a timer. The Schedule worker executes in this way.

2. From an asynchronous event. Some tasks are asynchronous but require immediate handling to maintain overall system responsiveness, or to ensure that data is not lost. The following workers poll or listen for such events:
  - » Event Catcher workers
  - » WebUI workers
  - » Web Services (REST API) workers
  - » Web Socket workers
3. From a message. Asynchronous tasks that are not time-critical are dispatched to workers using a message queue. The following list shows "queue workers" that receive work from queued messages:
  - » Generic workers
  - » Priority workers
  - » Metrics Collector workers
  - » Metrics Processor workers
  - » Refresh workers
  - » Event Handler workers
  - » SmartProxy workers
  - » Reporting workers

Many of the queued messages are created by workers dispatching work to other workers. For example, the Schedule worker will queue a message for the SmartProxy workers to initiate a SmartState Analysis. An Event Catcher worker will queue a message for an Event Handler worker to process the event. This will in turn queue a message for a Priority worker to process the event through the automate event switchboard.

### Tip

Queue workers process messages in a serial fashion. A worker processes one and only one message at a time.

## 2.7. MESSAGES

The queue workers receive work instructions from messages, delivered via a VMDB table called *miq\_queue*, and modelled by the Rails class **MiqQueue**. Each queue worker queries the *miq\_queue* table to look for work for any of its roles. If a message is claimed by a worker, the message state is changed from "ready" to "dequeue" and the worker starts processing the message. When the message processing has completed the message state is updated to indicate "ok", "error" or "timeout". Messages that have completed processing are purged on a regular basis.

### 2.7.1. Message Prefetch

To improve the performance of the messaging system, each CFME appliance prefetches a batch of messages into its local memcache. When a worker looks for work by searching for a "ready" state

message, it calls an `MiqQueue` method `get_message_via_drb` that transparently searches the prefetched message copies in the memcache. If a suitable message is found, the message's state in the VMDB `miq_queue` table is changed to "dequeue", and the message is processed by the worker.

## 2.7.2. Message Fields

A message contains a number of fields. The useful ones to be aware of for troubleshooting purposes are described below.

### 2.7.2.1. Ident

Each message has an `Ident` field that specifies the worker type that the message is intended for. Messages with an `Ident` field of 'generic' can be processed by either `MiqGenericWorkers` or `MiqPriorityWorkers`, depending on message priority.

### 2.7.2.2. Role

The message also has a `Role` field that specifies the server role that the message is intended for. Some workers - the Generic and Priority workers for example - process the messages for several server roles such as Automation Engine or Provider Operations. Workers are aware of the active server roles on their CFME appliance, and only dequeue messages for the enabled server roles.

### 2.7.2.3. Priority

Messages each have a `Priority` field such that lower priority messages for the same worker role are processed before higher priority messages (1 = highest, 200 = lowest). For example, priority 90 messages are processed before priority 100 messages regardless of the order in which they were created. The default message priority is 100, but tasks that are considered of greater importance are queued using messages with lower priority numbers. These message priorities are generally hard-coded and not customizable.

### 2.7.2.4. Zone

Each message has a `Zone` field that specifies the zone that the receiving worker should be a member of in order to dequeue the message. Some messages are created with the zone field empty, which means that the message can be dequeued and processed by the `Ident` worker type in any zone.

### 2.7.2.5. Server

Messages have a `Server` field, which is only used if the message is intended to be processed by a particular CFME appliance. If used, the field specifies the GUID of the target CFME appliance.

### 2.7.2.6. Timeout

Each message has a `Timeout` field. If the dispatching worker has not completed the message task in the time specified by the timeout, the worker will be terminated and a new worker spawned in its place.

### 2.7.2.7. State



The messages have a *State* field that describes the current processing status of the message (see below).

### 2.7.3. Tracing Messages in evm.log

Message processing is so critical to the overall performance of a CloudForms installation, that understanding how to follow messages in *evm.log* is an important skill to master when scaling CloudForms. There are generally four stages of message processing that can be followed in the log file. For this example a message will be traced that instructs the Automation Engine (role "automate" in queue "generic") to run the method **AutomationTask.execute** on automation task ID 7829.

#### 2.7.3.1. Stage 1 - Adding a message to the queue.

A worker (or other Rails process) adds a message to the queue by calling **MiqQueue.put**, passing all associated arguments that the receiving worker needs to process the task. For this example the message should be processed in zone 'RHV', and has a timeout of 600 seconds (automation tasks typically have a 10 minute time period in which to run). The message priority is 100, indicating that a Generic worker rather than Priority worker should process the message (both workers monitor the "generic" queue). The line from *evm.log* is as follows:

```
... INFO -- : Q-task_id([automation_request_6298]) MIQ(MiqQueue.put) ↵
Message id: [32425368], ↵
id: [], ↵
Zone: [RHV], ↵
Role: [automate], ↵
Server: [], ↵
Ident: [generic], ↵
Target id: [], ↵
Instance id: [7829], ↵
Task id: [automation_task_7829], ↵
Command: [AutomationTask.execute], ↵
Timeout: [600], ↵
Priority: [100], ↵
State: [ready], ↵
Deliver On: [], ↵
Data: [], ↵
Args: []
```

#### 2.7.3.2. Stage 2 - Retrieving a message from the queue.

A Generic worker calls **get\_message\_via\_drb** to dequeue the next available message. This method searches the prefetched message queue in the memcache for the next available message with a state of "ready". The new message with ID 32425368 is found, so its state is changed to "dequeue" in the VMDB *miq\_queue* table, and the message is dispatched to the worker. The line from *evm.log* is as follows:

```
... INFO -- : MIQ(MiqGenericWorker::Runner#get_message_via_drb) ↵
Message id: [32425368], ↵
MiqWorker id: [260305], ↵
Zone: [RHV], ↵
Role: [automate], ↵
Server: [], ↵
Ident: [generic], ↵
Target id: [], ↵
```

```
Instance id: [7829], ↵
Task id: [automation_task_7829], ↵
Command: [AutomationTask.execute], ↵
Timeout: [600], ↵
Priority: [100], ↵
State: [dequeue], ↵
Deliver On: [], ↵
Data: [], ↵
Args: [], ↵
Dequeued in: [6.698342458] seconds
```

### Tip

The "Dequeued in" value is particularly useful to monitor when scaling CloudForms as this shows the length of time that the message was in the queue before being processed. Although most messages are dequeued within a small number of seconds, a large value does not necessarily indicate a problem. Some messages are queued with a 'Deliver On' time which may be many minutes or hours in the future. The message will not be dequeued until the 'Deliver On' time has expired.

An example of this can be seen in the message to schedule a C&U hourly rollup, as follows:

```
... State: [dequeue], Deliver On: [2017-04-27 09:00:00 UTC], ↵
Data: [], Args: ["2017-04-27T08:00:00Z", "hourly"], ↵
Dequeued in: [2430.509191336] seconds
```

#### 2.7.3.3. Stage 3 - Delivering the message to the worker.

The `MiqQueue` class's **deliver** method writes to *evm.log* to indicate that the message is being delivered to a worker, and starts the timeout clock for its processing. The line from *evm.log* is as follows:

```
... INFO -- : Q-task_id([automation_task_7829]) ↵
MIQ(MiqQueue#deliver) Message id: [32425368], Delivering...
```

#### 2.7.3.4. Stage 4 - Message delivered and work is complete.

Once the worker has finished processing the task associated with the message, the `MiqQueue` class's **delivered** method writes to *evm.log* to indicate that message processing is complete. The line from *evm.log* is as follows:

```
... INFO -- : Q-task_id([automation_task_7829]) ↵
MIQ(MiqQueue#delivered) ↵
Message id: [32425368], ↵
State: [ok], ↵
Delivered in [23.469068759] seconds
```

### Tip

The "Delivered in" value is particularly useful to monitor when scaling CloudForms as this shows the time that the worker spent processing the task associated with the message.

### 2.7.4. Monitoring Message Queue Status

The overall performance of any multi-appliance CloudForms installation is largely dependant on the timely processing of messages. Fortunately the internal **log\_system\_status** method writes the queue states to *evm.log* every 5 minutes, and this information can be used to assess message throughput.

To find the numbers of messages currently being processed (in state "dequeue") in each zone, use the following bash command:

```
grep 'count for state=\["dequeue"\]' evm.log

... Q-task_id([log_status]) MIQ(MiqServer.log_system_status) ↵
[EVM Server (2768)] MiqQueue count for state=["dequeue"] ↵
by zone and role: {"RHV"=>{nil=>1, "automate"=>1, ↵
"ems_metrics_coordinator"=>1, "ems_metrics_collector"=>2, ↵
"ems_metrics_processor"=>2, "smartproxy"=>1, "smartstate"=>2}, ↵
nil=>{"database_owner"=>1}}
```

#### Tip

Messages that appear to be in state 'dequeue' for longer than their timeout period were probably 'in-flight' when the worker process running them died or was terminated.

To find the numbers of messages in state "error" in each zone, use the following bash command:

```
grep 'count for state=\["error"\]' evm.log

... Q-task_id([log_status]) MIQ(MiqServer.log_system_status) ↵
[EVM Server (2768)] MiqQueue count for state=["error"] ↵
by zone and role: {"RHV"=>{nil=>36}, "default"=>{nil=>16}, ↵
"UI Zone"=>{nil=>35}}
```

To find the numbers of messages in state "ready" that are waiting to be dequeued in each zone, use the following bash command:

```
grep 'count for state=\["ready"\]' evm.log

... Q-task_id([log_status]) MIQ(MiqServer.log_system_status) ↵
[EVM Server (2768)] \ MiqQueue count for state=["ready"] ↵
by zone and role: {"UI Zone"=>{"smartstate"=>15, "smartproxy"=>2, ↵
nil=>4}, "default"=>{"automate"=>2, nil=>21, "smartstate"=>1, ↵
"smartproxy"=>1}, "RHV"=>{"automate"=>6, "ems_inventory"=>1, ↵
nil=>19, "smartstate"=>2, "ems_metrics_processor"=>1259, ↵
"ems_metrics_collector"=>641}}
```

**Tip**

The count for "ready" state elements in the MiqQueue table should not be greater than twice the number of managed objects (e.g. hosts, VMs, storages) in the region. A higher number than this is a good indication that the worker count should be increased, or further CFME appliances deployed to handle the additional workload.

## 2.8. SUMMARY OF ROLES, WORKERS AND MESSAGES

The following table summarises the server roles, the workers performing the role tasks, the 'Role' field within the messages handled by those workers, and the maximum number of concurrent instances of the role per region or zone.

Role	Worker	Message 'Role'	Maximum Concurrent Workers
Automation Engine	Generic or Priority	automate	unlimited/ region
C&U Coordinator	Generic or Priority	ems_metrics_coordinator	one/zone
C&U Data Collector	provider-specific MetricsCollectorWorker	ems_metrics_collector	unlimited/ zone
C&U Data Processor	MiqEmsMetricsProcessorWorker	ems_metrics_processor	unlimited/ zone
Database Operations	Generic or Priority	database_owner	unlimited/ region
Embedded Ansible	EmbeddedAnsibleWorker	N/A	one/ region
Event Monitor	MiqEventHandler & provider-specific EventCatchers	event	one/zone & one/ provider/ zone
Git Repositories Owner	N/A	N/A	one/zone

Role	Worker	Message 'Role'	Maximum Concurrent Workers
Notifier	Generic or Priority	notifier	one/ region
Provider Inventory	provider-specific RefreshWorker	ems_inventory	one/ provider/ zone
Provider Operations	Generic or Priority	ems_operations	unlimited/ zone
RHN Mirror	N/A	N/A	unlimited/ region
Reporting	MiqReportingWorker	reporting	unlimited/ region
Scheduler	MiqScheduleWorker	N/A	one/ region
SmartProxy	MiqSmartProxyWorker	smartproxy	unlimited/ zone
SmartState Analysis	Generic or Priority	smartstate	unlimited/ zone
User Interface	MiqUiWorker	N/A	unlimited/ region
Web Services	MiqWebServiceWorker	N/A	unlimited/ region
Web Socket	MiqWebsocketWorker	N/A	unlimited/ region

[1] The full list of supported providers and their capabilities is included in the CloudForms Support Matrix document. The most recent Support Matrix document is here: [https://access.redhat.com/documentation/en-us/red\\_hat\\_cloudforms/4.2/html/support\\_matrix/](https://access.redhat.com/documentation/en-us/red_hat_cloudforms/4.2/html/support_matrix/)

[2] Not all automation tasks are queued. The automate methods that populate dynamic dialog elements, for example, are run immediately on the CFME appliance running the WebUI session, regardless of whether it has the *Automation Engine* role enabled

[3] See also [https://bugzilla.redhat.com/show\\_bug.cgi?id=1422943](https://bugzilla.redhat.com/show_bug.cgi?id=1422943)

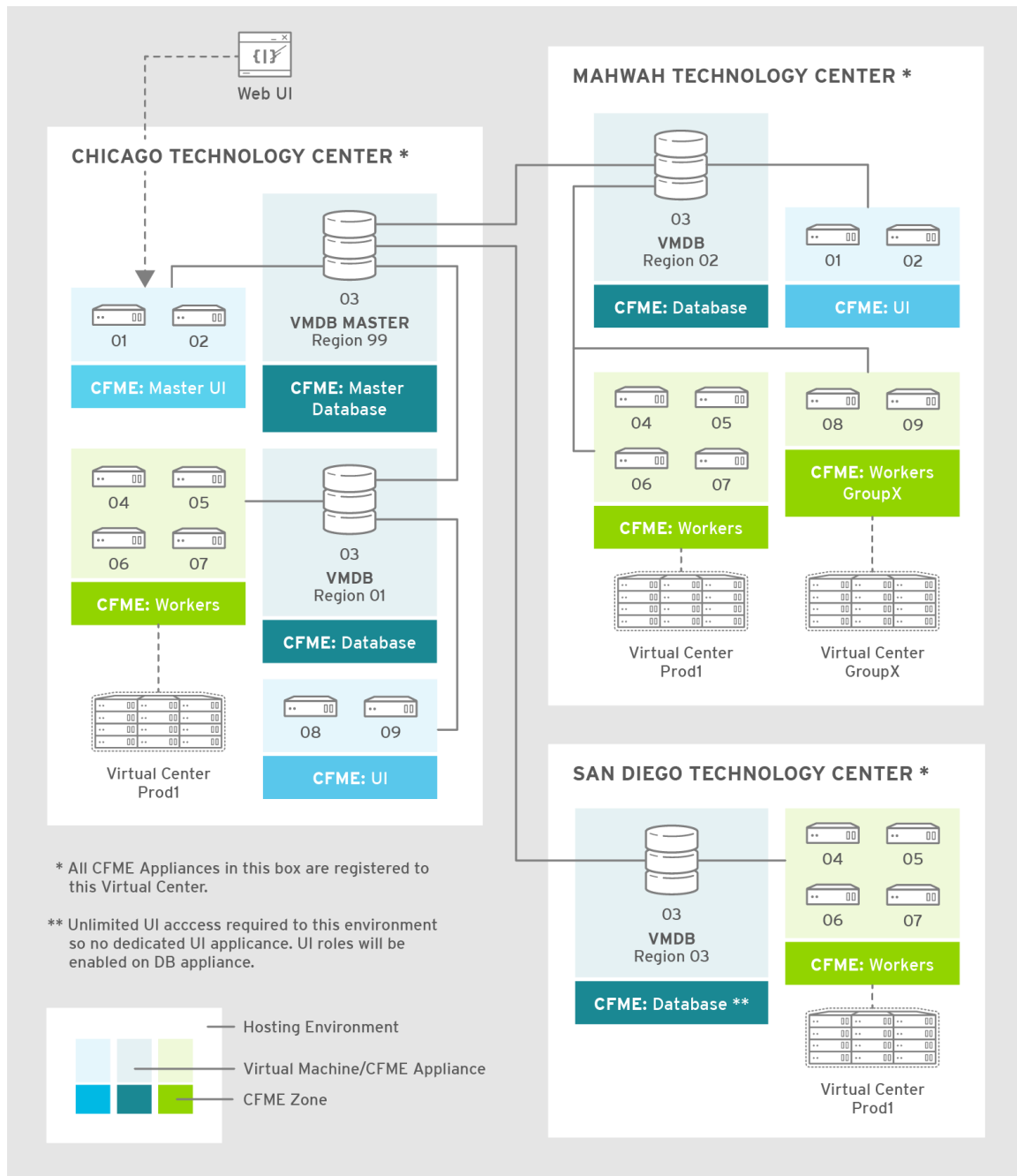
[4] Worker processes issue a heartbeat ping every 10 seconds

[5] The time limit for Refresh workers sometimes needs to be increased in very large environments where a full refresh can take longer than 2 hours

## CHAPTER 3. REGION AND ZONES

When planning a large CloudForms implementation, consideration must be given to the number and size of regions required, and the layout of zones within those regions. [6]. [Figure 3.1, “Regions and Zones”](#) shows an example of multiple regions working together in a Red Hat CloudForms environment.

**Figure 3.1. Regions and Zones**



In this example the Corporate HQ is located in Chicago Technology Center. This location contains a global "Master" region, and a sub-region that manages the worker appliances. The Mahwah technology center contains a single sub-region that manages two zones. Likewise, the San Diego Technology Center contains a single sub-region managing one zone.

This section describes some of the considerations when designing regions and zones, and presents some guidelines and suggestions for implementation.

## 3.1. REGIONS

A region is a self-contained collection of CloudForms Management Engine (CFME) appliances. Each region has a database - the VMDB - and one or more appliances running the `evmservd` service with an associated set of configured worker processes. **Regions are often used for organisational or geographical separation of resources**, and the choice of region count, location and size is often based on both operational and technical factors.

### 3.1.1. Region Size

All CFME appliances in a region access the same PostgreSQL database, and so the I/O and CPU performance of the database server is a significant factor in determining the maximum size to which a region can grow (in terms of numbers of managed objects) whilst maintaining acceptable performance.

#### 3.1.1.1. Database Load Factors

The VMDB database load is determined by many factors including:

- ✧ **The number of managed objects (VMs, hosts, datastores, etc.) in the region**
- ✧ **The number and type of providers added to the region (for example providers such as VMware or RHV have more out-of-the-box events that can be detected and processed).**
- ✧ The overall "busyness" of the external management systems (such as vCenters), which determines the rate at which events are received and processed, and thus the rate at which inventory refreshes are requested and loaded.<sup>[7]</sup> (see [Chapter 5, Inventory Refresh](#))
- ✧ The frequency of "event storms" (see [Chapter 9, Event Handling](#)) from the external management systems
- ✧ **Whether or not Capacity and Utilization (C&U) metric collection is enabled for the region**
  - Whether this is for all clusters and datastores or a subset of each
  - The frequency of collection
- ✧ Whether or not SmartState Analysis is enabled for the region
  - **The frequency of collection**
  - **The amount of data collected in the SmartState Analysis profile**
- ✧ The complexity of reports and widgets, and frequency of generation
- ✧ The frequency of VM lifecycle operations
  - Provisioning
  - Retirement
- ✧ The frequency of running automate requests and tasks, including service requests
- ✧ The number of control or compliance policies in use



- ✧ The number of concurrent users accessing the "classic" WebUI, especially displaying large numbers of objects such as VMs
- ✧ The frequency and load profile of connections to the RESTful API (including the Self-Service UI)
- ✧ The number of CFME appliances (more accurately, worker processes) in the region

### 3.1.1.2. Sizing Estimation

It is very difficult to define a representative load for simulation purposes, due to the many permutations of workload factors. Some analysis has been made of existing large CloudForms installations however, and it has been observed that for an "average" mix of the workload factors listed above, an optimally **tuned and maintained PostgreSQL server should be able to handle the load from managing up to 5000 VMware objects (VMs, hosts, clusters and datastores, for example)**. Larger regions than this are possible if the overall database workload is lighter - typically the case for the cloud and container providers - but as with any large database system, performance should be carefully monitored.

**Table 3.1, "Guidelines for Maximum Region Size"** **provides suggested guidelines for the maximum number of objects (VMs, instances, images, templates, clusters, hosts, datastores, pods or containers, for example) in a region containing an active provider.** Regions with several provider types (for example both VMware and Amazon EC2) will have a practical maximum somewhere between the limits suggested for each provider.

**Table 3.1. Guidelines for Maximum Region Size**

Provider	Guideline Number of Objects in Region
VMware	5000
RHV	5000
OpenStack	7500
OpenShift	10000
Microsoft SCVMM	10000
Microsoft Azure	10000
Amazon EC2	10000

Provider	Guideline Number of Objects in Region
Google Compute Engine	10000

It should be noted that these numbers are approximate and are only suitable for planning and design purposes. The absolute practical maximum size for a region will depend on acceptable performance criteria, database server capability, and the factors listed in [Section 3.1.1.1, “Database Load Factors”](#).

When planning regions it is often useful to under-size a region rather than over-size. It is usually easier to add capacity to a smaller region that is performing well, than it is to split an under-performing large single region into multiple regions.



#### Note

A 'global' region is generally capable of handling considerably more objects as it has no active providers of its own, and has a lower database load.

### 3.1.2. Number of CFME Appliances in a Region

When sizing a region, some thought needs to be given to the number of CloudForms worker processes that are likely to be needed to handle the expected workload, and hence the number of CFME appliances. The workload will depend on the capabilities of the providers that will be configured, and the CloudForms features that are likely to be used.

Two of the most resource-intensive tasks are those performed by the C&U Data collector and Data Processor workers, particularly where there is a limited time window for the collection of realtime data as there is with VMware or OpenStack providers (see [Chapter 6, Capacity & Utilization](#)). It has been established through testing that one C&U Data Collector worker can retrieve and store the metrics from approximately 150 VMware VMs or OpenStack instances in the rolling 60 minute time window that realtime metrics are retained for. As an out-of-the-box CFME appliance is configured with 2 C&U Data Collector workers, it should be able to handle the collection of realtime metrics for 300 VMs. If the number of workers is increased to 4, the appliance could handle the collection of realtime metrics for 600 VMs, although the increased CPU and memory load may adversely affect other processing taking place on the appliance.

Using the 1:300 ratio of CFME appliances to VMs is a convenient starting point for scaling the number of CFME appliances required for a region containing VMware, RHV or OpenStack providers. For other provider types this ratio is often increased to 1:400.

[Table 3.2, “Objects per CFME Appliance Guidelines”](#) provides suggested guideline ratios for each of the provider types. It should be noted that these numbers are approximate and are only suitable for planning and design purposes. The final numbers of CFME appliances required for a region or zone can only be determined from analysis of the specific region workload, and the performance of existing CFME appliances.

**Table 3.2. Objects per CFME Appliance Guidelines**

Provider	Guideline Number of Objects/CFME Appliance
VMware	300 (VMs)
RHV	300 (VMs)
OpenStack	300 (instances)
Microsoft SCVMM	400 (VMs)
Microsoft Azure	400 (instances)
Amazon EC2	400 (instances)
Google Compute Engine	400 (instances)
OpenShift	400 (pods &/or containers)

### 3.1.3. Region Design

There are a number of considerations for region design and layout, but the most important are the anticipated number of managed objects (discussed above), and the location of the infrastructure components being managed, or the public cloud endpoints.

#### 3.1.3.1. Centrally Located Infrastructure

With a single, centrally located small or medium sized virtual infrastructure or cloud, the selection of region design is simpler. A single region is usually the most suitable option, with high availability and fault tolerance built into the design.



#### Note

Large virtual infrastructures can often be split between several regions using multiple sets of provider credentials that have a restricted span-of-control within the entire enterprise.

#### 3.1.3.2. Distributed Infrastructure

With a distributed or large infrastructure the most obvious choice of region design might seem to be to allocate a region to each distributed location, however there are a number of advantages to both single and multi-region implementations for distributed infrastructures.

### 3.1.3.2.1. Wide Area Network Factors

Network latency between CFME appliances and the database plays a big factor in overall CloudForms "system" responsiveness. There are two utilities supplied on each CFME appliance that can check the latency to a database. The first, `db_ping`, is designed to check intra-region latency between an existing appliance and its own regional database. The second, `db_ping_remote`, is designed to check inter-region latency, and so requires external PostgreSQL server details and credentials. They are run as follows:

```
vmdb
tools/db_ping.rb
0.358361 ms
1.058845 ms
0.996966 ms
1.029908 ms
1.048192 ms
```

Average: 0.898454 ms

```
tools/db_ping_remote.rb 10.3.0.22 5432 root vmdb_production
```

Enter the password for database user root on host 10.3.0.22

Password:

```
0.874407 ms
0.984994 ms
1.040376 ms
1.119602 ms
1.031609 ms
```

Average: 1.010198 ms

#### Note

On CFME versions prior to 5.8, these tools should be prefixed by **bin/rails runner**, for example:

```
bin/rails runner tools/db_ping.rb
bin/rails runner tools/db_ping_remote.rb
```

The architecture of CloudForms assumes LAN-speed latency ( $\approx 1$  ms) between CFME appliances and the database for optimal performance. As latency increases, so overall system responsiveness decreases.

Typical symptoms of a high latency connection are as follows:

- ✧ WebUI operations appear to be slow, especially viewing screens that display a large number of objects such as VMs
- ✧ Database-intensive actions such as complex report or widget generation take longer to run
- ✧ CFME appliance restarts are slower since the startup seeding acquires an exclusive lock.
- ✧ Worker tasks such as EMS refresh or C&U metrics collection that load data into the VMDB run more slowly
  - Longer EMS refreshes may have a detrimental effect on other operations such as VM

provisioning.<sup>[8]</sup>

- Metrics collection might not keep up with the EMS's realtime statistics retention period.<sup>[9]</sup>

When considering deploying a CloudForms region spanning a WAN, it is important to establish acceptable performance criteria for the installation. Although in general a higher latency will result in slower but error-free performance, it has been observed that a latency of 5ms can cause the VMDB update transaction from an EMS refresh to timeout in very large regions. A latency as high as 42 ms can cause failures in database seeding operations.<sup>[10]</sup>

### 3.1.3.2.2. Single Region

Where WAN latency is deemed acceptable, the advantages of deploying a single region to manage all objects in a distributed infrastructure are as follows:

- ✧ Simplified appliance upgrade procedures (no multiple regions or global region upgrade coordination issues)
- ✧ Simplified disaster recovery when there is only one database to manage
- ✧ Simpler architectural design, and therefore more straightforward operational procedures and documentation
- ✧ Easier to manage the deployment of customisations such as automate code, policies, or reports (there is a single point of import)

### 3.1.3.2.3. Multi-Region

The advantages of deploying multiple regions to manage the objects in a distributed infrastructure are as follows:

- ✧ Operational resiliency; no single point of failure to cause outage to the entire CloudForms managed environment
- ✧ Continuous database maintenance runs faster in a smaller database
- ✧ Database reorganisations (backup & restore) run faster and don't take offline an entire CloudForms installation
- ✧ More intuitive alignment between CloudForms WebUI view, and physical and virtual infrastructure
- ✧ Reduced dependence on wide-area networking to maintain CloudForms performance
- ✧ Region isolation (for performance)
  - Infrastructure issues such as event storms that might adversely affect the local region database will not impact any other region
  - Customisations can be tested in a development or test region before deploying to a production environment

## 3.1.4. Connecting Regions

As illustrated in Figure 3.1, “Regions and Zones” regions can be linked in such a way that several subordinate regions replicate their object data to a single *global* region. The global region has no providers of its own, and is typically used for enterprise-wide reporting as it has visibility of all objects. A new feature introduced with CloudForms 4.2 allows some management operations to be

performed directly from the global region, utilising a RESTful API connection to the correct child region to perform the action. These operations include the following:

- ✧ Virtual machine provisioning
- ✧ Service provisioning
- ✧ Virtual machine power operations
- ✧ Virtual machine retirement
- ✧ Virtual machine reconfiguration

### 3.1.5. Region Numbering

Regions have associated with them a region number that is allocated when the VMDB appliance is first initialised. When several regions are linked in a global/subregion hierarchy, all of the region numbers must be unique. Region numbers can be up to three digits long, and the region number is encoded into the leading digits of every object ID in the region. For example the following 3 message IDs are from different regions:

- ✧ Message id: [1000000933021] (region 1)
- ✧ Message id: [9900023878436] (region 99)
- ✧ Message id: [398451] (region 0)

Global regions are often allocated a higher region number (99 is frequently used) to distinguish them from subordinate regions whose numbers often start with 0 and increase as regions are added. There is no technical restriction on region number allocation in a connected multi-region CloudForms deployment, other than uniqueness.

### 3.1.6. Region Summary and Recommendations

The following guidelines can be used when designing a region topology:

- ✧ Beware of over-sizing regions. Several slightly smaller interconnected regions will generally perform better than a single very large region
- ✧ Network latency from CFME appliances to the VMDB within the region should be close to LAN speed
- ✧ Database performance is critical to the overall performance of the region
- ✧ All CFME appliances in a region should be NTP synchronized to the same time source
- ✧ Identify all external management system (EMS) host or hypervisor instances where steady-state or peak utilization > 50%, and avoid these hosts for placement of CFME appliances, especially the VMDB appliance.

## 3.2. ZONES

Zones are a way of logically subdividing the resources and worker processing within a region. They perform a number of useful functions, particularly for larger CloudForms installations.

### 3.2.1. Zone Advantages

The following sections describe some of the advantages of implementing zones within a CloudForms region.

### 3.2.1.1. Provider Isolation

Zones are a convenient way of isolating providers. Each provider has a number of workers associated with it that run on any appliance running the Provider Inventory and Event Monitor roles. These include:

- ✧ One Refresh worker
- ✧ Two or more Metrics Collector workers
- ✧ One Event Catcher
- ✧ For VMware:
  - One Core Refresh worker
  - One Vim Broker

Some types of cloud provider add several sub-provider types, each having their own Event Catchers and/or Refresh workers, and some also having Metrics Collector workers. For example adding a single OpenStack Cloud provider will add the following workers to each appliance with the Provider Inventory and Event Monitor roles:

- ✧ `ManagelQ::Providers::Openstack::CloudManager::EventCatcher`
- ✧ `ManagelQ::Providers::Openstack::CloudManager::MetricsCollectorWorker (x 2)`
- ✧ `ManagelQ::Providers::Openstack::CloudManager::RefreshWorker`
- ✧ `ManagelQ::Providers::Openstack::NetworkManager::EventCatcher`
- ✧ `ManagelQ::Providers::Openstack::NetworkManager::MetricsCollectorWorker (x 2)`
- ✧ `ManagelQ::Providers::Openstack::NetworkManager::RefreshWorker`
- ✧ `ManagelQ::Providers::StorageManager::CinderManager::EventCatcher`
- ✧ `ManagelQ::Providers::StorageManager::CinderManager::RefreshWorker`
- ✧ `ManagelQ::Providers::StorageManager::SwiftManager::RefreshWorker`

In addition to these provider-specific workers, the two roles add a further two worker types that handle the events and process the metrics for all providers in the zone:

- ✧ One Event Handler
- ✧ Two or more Metrics Processor workers

Each worker has a minimum startup cost of approximately 250-300MB, and the memory demands of each may vary depending on the number of managed objects for each provider. Having one provider per zone reduces the memory footprint of the workers running on the CFME appliances in the zone, and allows for dedicated per-provider Event Handler and Metrics Processor workers. This prevents an event surge from one provider from adversely affecting the handling of events from another provider, for example.

### 3.2.1.2. Appliance Maintenance

Shutting down or restarting a CFME appliance in a zone because of upgrade or update is less disruptive if only a single provider is affected.

### 3.2.1.3. Provider-Specific Appliance Tuning

Zones allow for more predictable and provider-instance-specific sizing of CFME appliances and appliance settings based on the requirement of individual providers. For example small VMware providers can have significantly different resourcing requirements to very large VMware providers, especially for C&U collection and processing.

### 3.2.1.4. VMDB Isolation

If the VMDB is running on a CFME appliance (as opposed to a dedicated PostgreSQL appliance), putting the VMDB appliance in its own zone is a convenient way to isolate the appliance from non database-related activities.

### 3.2.1.5. Logical Association of Resources

A zone is a natural and intuitive way of associating a provider with a corresponding set of physical or logical resources, either in the same or remote location. For example there might be a requirement to open firewall ports to enable access to a particular provider's EMS on a restricted or remote network. Isolating the specific CFME appliances to their own zone simplifies this task.



#### Note

Not all worker processes are zone-aware. Some workers process messages originating from or relevant to the entire region

### 3.2.1.6. Improved and Simplified Diagnostics Gathering

Specifying a log depot per zone in **Configuration** → **Settings** allows log collection to be initiated for all appliances in the zone, in a single action. When requested, each appliance in the zone is notified to generate and deposit the specified logs into the zone-specific depot.

## 3.2.2. Zone Summary and Recommendations

The following guidelines can be used when designing a zone topology:

- ✧ Use a separate zone per provider instance (rather than provider type)
- ✧ Never span a zone across physical boundaries or locations
- ✧ Use a minimum of two appliances per zone for resiliency of zone-aware workers and processes
- ✧ Isolate the VMDB appliance in its own zone (unless it is a standalone PostgreSQL server)
- ✧ At least one CFME appliance in each zone should have the 'Automate Engine' role enabled, to process zone-specific events
- ✧ At least once CFME appliance in each zone should have the 'Provider Operations' role enabled to ensure that the service provision request tasks are processed correctly



- ✱ Isolating the CFME appliances that general users interact with (running the User Interface and Web Services workers) into their own zone can allow for additional security measure to be taken to protect these servers
  - At least one CFME appliance in a WebUI zone should have the 'Reporting' role enabled to ensure that reports interactively scheduled by users are correctly processed (see [Section 2.5.11, “Reporting”](#) for more details)

---

[6] Regions and zones are described in the CloudForms "Deployment Planning Guide"

[https://access.redhat.com/documentation/en-us/red\\_hat\\_cloudforms/4.2/html/deployment\\_planning\\_guide/](https://access.redhat.com/documentation/en-us/red_hat_cloudforms/4.2/html/deployment_planning_guide/)

[7] With VMware providers relatively minor changes such as VM and Host property updates are detected by the Vim Broker and also cause EMS refreshes to be scheduled

[8] discussed in [Chapter 8, VM and Instance Provisioning](#)

[9] discussed in [Chapter 6, Capacity & Utilization](#)

[10] See [https://bugzilla.redhat.com/show\\_bug.cgi?id=1422671](https://bugzilla.redhat.com/show_bug.cgi?id=1422671)

## CHAPTER 4. DATABASE SIZING AND OPTIMIZATION

As discussed in [Chapter 2, \*Architecture\*](#) CloudForms 4.5 uses a PostgreSQL 9.5 database as its VMDB back-end, and the performance of this database is critical to the overall smooth-running of the CloudForms installation. This section discusses techniques to ensure that the database is installed and configured correctly, and optimally tuned.

### 4.1. SIZING THE DATABASE APPLIANCE

The database server or appliance should be sized according to the anticipated scale of the CloudForms deployment. A minimum of 8 GBytes memory is recommended for most installations, but the required number of vCPUs varies with the number of worker processes accessing the database. An initial estimate can be made based on the anticipated idle load of the CFME appliances in the region.

Some earlier investigation into the database load generated by idle CFME appliances is published in [Appendix A, \*Database Appliance CPU Count\*](#). To determine the number of CFME appliances required for a region, the total anticipated number of managed objects (VMs, hosts, clusters, datastores etc.) should first be established. Using the appliance to VM ratios suggested in [Table 3.2, “Objects per CFME Appliance Guidelines”](#), the CFME appliance count can be estimated. For example a Red Hat Virtualization virtual infrastructure containing 3000 managed objects would need 10 CFME appliances in default configuration to handle a typical workload.

It can be seen from the table in [Appendix A, \*Database Appliance CPU Count\*](#) that a region containing 10 idle CFME appliances would need a database server with 4 vCPUs to maintain CPU utilisation at under 25%. Adding the provider(s) and enabling the various server roles on the appliances will increase the database load, and so CPU, memory and I/O utilisation must be monitored. A CPU utilisation of 75-80% should be considered a maximum for the database server, and an indication that any of the following should be increased:

- » Real memory
- » vCPUs
- » The value of `shared_buffers` (see below)

### 4.2. SIZING THE DATABASE PARTITION BEFORE INSTALLATION

The disk used for the database partition should be presented to the database appliance from the fastest available block storage. Sparsely allocated, thin provisioned or file-backed storage such as NFS are not recommended for optimum performance.

A rough estimate of database size in GBytes can be made by approximating the number of managed VMs, hosts and storage devices in the region over a one and two year period.<sup>[11]</sup>

The following guidelines can be used:

After 1 year:

Database size (GBytes) =

$(\text{VM Count} * 0.035) + (\text{Host count} * 0.0002) + (\text{Storage Count} * 0.001)$

After 2 years:

Database size (GBytes) =

$(\text{VM Count} * 0.055) + (\text{Host count} * 0.0002) + (\text{Storage Count} * 0.0015)$

As an example, given an installation that is projected to manage 1500 VMs, with 20 hypervisors and 25 storage domains, the estimated database size would be:

$(1500 * 0.035) + (20 * 0.0002) + (25 * 0.001) = \mathbf{52.57 \text{ GBytes}}$  after 1 year

$(1500 * 0.055) + (20 * 0.0002) + (25 * 0.0015) = \mathbf{82.58 \text{ GBytes}}$  after 2 years

## 4.3. INSTALLATION

The first CFME appliance in a region can be configured as a database appliance with or without Rails. The database is created using **appliance\_console** by selecting the following option:

### 5) Configure Database

After creating a new encryption key and selecting to create an internal database, the following question is asked:

Should this appliance run as a standalone database server?

#### NOTE:

- \* The CFME application will not be running.
- \* This is required when using highly available database deployments.
- \* CAUTION: This is not reversible.

Selecting 'Y' to this option configures the appliance without Rails or the **evmserved** application, and allows the server to be configured optimally as a pure PostgreSQL server. This configuration also allows for PostgreSQL high availability to be configured at any time in the future using the following **appliance\_console** option:

### 6) Configure Database Replication

#### Note

Although configuring a CFME appliance as a dedicated database instance will result in optimum database performance, the absence of Rails and the **evmserved** service and workers means that the server will not appear in the CloudForms WebUI as a CFME appliance in the region.

If it is known that PostgreSQL high availability will never be required (and at the expense of some loss of memory and CPU utilisation), the answer 'N' can be given to the question **Should this appliance run as a standalone database server?**. In this case the VMDB appliance will be configured to run Rails and the **evmserved** service as normal, and will appear as a CloudForms appliance in the region.

It is recommended that this type of VMDB appliance be isolated in its own dedicated zone, and that any unnecessary server roles are disabled.

### 4.3.1. Configuring PostgreSQL

The PostgreSQL configuration file on a CFME appliance is `/var/opt/rh/rh-postgresql95/lib/pgsql/data/postgresql.conf`. Some of the values in this file have been defined based on the assumption of a small CloudForms installation. For larger installations - particularly when using a dedicated database instance - these values should be changed.

#### 4.3.1.1. Shared Buffers

The most important tuning parameter to set is the value for **shared\_buffers**. The default value from the configuration file is as follows:

```
shared_buffers = 128MB      # MIQ Value SHARED CONFIGURATION
#shared_buffers = 1GB      # MIQ Value DEDICATED CONFIGURATION
```

For a dedicated PostgreSQL server this should be set to 25% of the real memory on the database appliance, but not more than a maximum of 4GB. This allows many more of the dense index pages and actively accessed tables to reside in memory, significantly reducing physical I/O and improving overall performance.

#### 4.3.1.2. Max Connections

Each worker process on a CFME appliance in a region opens a connection to the database. There are typically between 20 and 30 worker sessions per appliance in default configuration, and so a region with 20 CFME appliances will open approximately 500 connections. The default value for **max\_connections** in the configuration file is as follows:

```
max_connections = 1000      # MIQ Value;
#max_connections = 100
```

Although this default value allows for 1000 connections, under certain extreme load conditions CFME appliance workers can be killed but their sessions not terminated. In this situation the number of connections can rise above the expected value.

#### Tip

The number of open connections to the database can be seen using the following psql command:

```
SELECT datname,application_name,client_addr FROM pg_stat_activity;
```

The number of outbound database connections from a CFME appliance can be seen using the following bash command:

```
netstat -tp | grep postgres
```

It may be necessary to increase the value for **max\_connections** if the default number is being exceeded.

#### 4.3.1.3. Log Directory

By default the block device used for the database partition is used for the **PGDATA** directories and files, and also the `postgresql.log` log file (this is the text log file, not the database write-ahead log). Moving the log file to a separate partition allows the **PGDATA** block device to be used exclusively for database I/O, which can improve performance. The default value for **log\_directory** in the

configuration file is as follows:

```
#log_directory = 'pg_log'      # directory where log files are written,
                                # can be absolute or relative to PGDATA
```

This value creates the log file as `/var/opt/rh/rh-postgresql95/lib/pgsql/data/pg_log/postgresql.log`. To use the default CFME log directory for the log file, change this line to be:

```
log_directory = '/var/www/miq/vmdb/log'
```

#### 4.3.1.4. Huge Pages

For VMDB appliances configured as dedicated database instances, some performance gain can be achieved by creating sufficient kernel huge pages for PostgreSQL and the configured `shared_buffers` region. The following bash commands allocate 600 huge pages (1.2 GBytes):

```
sysctl -w vm.nr_hugepages=600
echo "vm.nr_hugepages=600" >> /etc/sysctl.d/rh-postgresql95.conf
```

The default setting for PostgreSQL 9.5 is to use huge pages if they are available, and so no further PostgreSQL configuration is necessary.

## 4.4. MAINTAINING PERFORMANCE

Several of the database tables benefit greatly from regular vacuuming and frequent re-indexing, and database maintenance scripts can be added to cron to perform these functions.<sup>[12]</sup>

On a CFME 5.8 appliance these scripts can be installed using the following **appliance\_console** option:

```
7) Configure Database Maintenance
```

The scripts perform hourly reindexing of the following tables:

- ✧ metrics\_00 to metrics\_23 (one per hour)
- ✧ miq\_queue
- ✧ miq\_workers

The scripts perform weekly or monthly vacuuming of the following tables:

- ✧ vms
- ✧ binary\_blob\_parts
- ✧ binary\_blobs
- ✧ customization\_specs
- ✧ firewall\_rules
- ✧ hosts
- ✧ storages

- ✱ miq\_schedules
- ✱ event\_logs
- ✱ policy\_events
- ✱ snapshots
- ✱ jobs
- ✱ networks
- ✱ miq\_queue
- ✱ miq\_request\_tasks
- ✱ miq\_workers
- ✱ miq\_servers
- ✱ miq\_searches
- ✱ miq\_scsi\_luns
- ✱ miq\_scsi\_targets
- ✱ storage\_files
- ✱ taggings
- ✱ vim\_performance\_states

## 4.5. RESIZING THE DATABASE DIRECTORY AFTER INSTALLATION

It is sometimes the case that a managed virtual infrastructure or cloud grows at a faster rate than anticipated. As a result the CloudForms database mount point may need expanding from its initial size to allow the database to grow further.

The database mount point `/var/opt/rh/rh-postgresql95/lib/pgsql` is a logical volume formatted as XFS. A new disk can be presented to the database appliance and added to LVM to allow the filesystem to grow.



### Note

Some virtual or cloud infrastructures don't support the 'hot' adding of a new disk to a virtual machine that is powered on. It may be necessary to stop the **evmserv** service on all CFME appliances in the region, and shut down the VMDB appliance before adding the new disk.

The following steps illustrate the procedure to add an additional 10 GBytes of storage (a new disk `/dev/vdd`) to the database mount point:

```
# label the new disk
parted /dev/vdd mklabel msdos

# partition the disk
parted /dev/vdd mkpart primary 2048s 100%
```

```

# create an LVM physical volume
pvcreate /dev/vdd1
Physical volume "/dev/vdd1" successfully created.

# add the new physical volume to the vg_pg volume group
vgextend vg_pg /dev/vdd1
Volume group "vg_pg" successfully extended

# determine the number of free extents in the volume group
vgdisplay vg_pg
--- Volume group ---
VG Name                vg_pg
System ID
...
VG Size                19.99 GiB
PE Size                4.00 MiB
Total PE               5118
Alloc PE / Size        2559 / 10.00 GiB
Free PE / Size         2559 / 10.00 GiB
VG UUID                IjKZmo-retr-qJ9f-WCdG-gzrc-jbl3-i52mUn

# extend the logical volume by the number of free extents
lvextend -l +2559 /dev/vg_pg/lv_pg
Size of logical volume vg_pg/lv_pg changed from 10.00 GiB ↵
(2559 extents) to 19.99 GiB (5118 extents).
Logical volume vg_pg/lv_pg successfully resized.

# grow the filesystem to fill the logical volume
xfs_growfs /var/opt/rh/rh-postgresql95/lib/pgsql
meta-data=/dev/mapper/vg_pg-lv_pg isize=256    ...
      =                               sectsz=512    ...
      =                               crc=0        ...
data      =                               bsize=4096    ...
      =                               sunit=0      ...
naming    =version 2                       bsize=4096    ...
log       =internal                       bsize=4096    ...
      =                               sectsz=512    ...
realtime  =none                           extsz=4096    ...
data blocks changed from 2620416 to 5240832

```

[11] These sizing estimates have been generated from real-world VMDB usage statistics gathered from earlier versions of ManageIQ/CloudForms, managing virtual infrastructures such as VMware. To date insufficient data has been gathered for comparable sizing estimates of CloudForms installations that primarily manage OpenShift Container Platforms

[12] See <https://access.redhat.com/solutions/1419333> (Continuous Maintenance for CloudForms Management Engine VMDB to maintain Responsiveness)

## CHAPTER 5. INVENTORY REFRESH

One of the biggest factors that affects the perceived performance of a large CloudForms installation is the time taken to update the provider inventory in the VMDB. This is known as an EMS refresh. There are two types of EMS refresh: a full refresh, where all objects are returned from the provider; and a targeted refresh, where only the details of requested components such as specific VMs or hosts are fetched and processed. In CloudForms Management Engine 5.8 only the VMware and Red Hat Virtualization providers are capable of supporting targeted refreshes; all other providers perform a full refresh.

### Note

A new type of refresh architecture called *graph refresh* is currently in development, and has been implemented for the Amazon S3 and Ansible providers in CFME 5.8. Graph refresh improves EMS refresh performance by up to 6 times.

The timings mentioned in this chapter are based on the pre-graph refresh architecture that existed in CFME 5.7

### 5.1. REFRESH OVERVIEW

Whenever CloudForms is notified of a change related to a managed object, a message is queued either for a refresh of that object (where targeted is supported), or a full EMS refresh. There is never more than one EMS refresh operation in progress for each provider at any one time, with at most one further refresh queued.

If a new refresh is called for, the `miq_queue` table is first examined to see if a refresh message already exists in the "ready" state for the intended EMS. If no such message already exists, a new one is created. If a message already exists and it is for a full refresh, the new request is ignored, but if the new refresh is targeted and an existing targeted message is found, the new request is merged into the existing message payload, and the message is re-queued. The addition of further targets to a "ready" queued message can happen several times until the message is dequeued.

This action can be observed in `evm.log`. In the following example an EMS refresh was initially queued for a VM with ID 1167. The following log line shows the initial `MiqQueue.put` operation:

```
... INFO -- : MIQ(MiqQueue.put) Message id: [32170091], id: [], ↵
Zone: [VMware], Role: [ems_inventory], Server: [], Ident: [ems_2], ↵
Target id: [], Instance id: [], Task id: [], ↵
Command: [EmsRefresh.refresh], Timeout: [7200], Priority: [100], ↵
State: [ready], Deliver On: [], Data: [], ↵
Args: [[["ManageIQ::Providers::Vmware::InfraManager::Vm", 1167]]]
```

Before this message has been dequeued and processed however, a further EMS refresh request is made for another VM, this time with ID 1241. The following log line shows the `MiqQueue.put_or_update` operation, where the queued message 32170091 is updated with the addition of a second VM in the "Args" field:

```
... INFO -- : MIQ(MiqQueue.put_or_update) Message id: [32170091], ↵
id: [], Zone: [VMware], Role: [ems_inventory], Server: [], ↵
Ident: [ems_2], Target id: [], Instance id: [], Task id: [], ↵
Command: [EmsRefresh.refresh], Timeout: [7200], Priority: [100], ↵
```



```
State: [ready], Deliver On: [], Data: [], ↵
Args: [[["ManageIQ::Providers::Vmware::InfraManager::Vm", 1167], ↵
["ManageIQ::Providers::Vmware::InfraManager::Vm", 1241]]], Requested
```

## 5.2. CHALLENGES OF SCALE

As might be expected, the more managed objects in a virtual infrastructure or cloud, the longer a full refresh takes to complete. The refresh time has a knock-on effect for the process or workflow that initiated the refresh. In some cases this is inconvenient but not critical, such as a delay in seeing a VM's power status change for its WebUI tile icon when it powers on. In other cases - such as provisioning a new VM - a very long EMS refresh may cause the triggering workflow to timeout and exit with an error condition.

## 5.3. MONITORING REFRESH PERFORMANCE

An EMS refresh operation has two significant phases that each contribute to the overall performance:

- ✧ Extracting and parsing the data from the EMS
  - Network latency to the EMS
  - Time waiting for the EMS to process the request and return data
  - CPU cycles parsing the returned data
- ✧ Updating the inventory in the VMDB
  - Network latency to the database
  - Database appliance CPU, memory and I/O resources

Fortunately the line printed to *evm.log* at the completion of the operation contains detailed timings of each stage of the operation, and these can be used to determine bottlenecks.<sup>[13]</sup> A typical log line is as follows:

```
... INFO -- :
MIQ(ManageIQ::Providers::Vmware::InfraManager::Refresher#refresh) ↵
EMS: [CLOUD], id: [1000000000001] Refreshing targets for EMS...Complete -
↵
Timings {:server_dequeue=>0.006215572357177734, ↵
: get_ems_data=>1.1113097667694092, ↵
: get_vc_data=>46.28569030761719, ↵
: filter_vc_data=>0.025593042373657227, ↵
: get_vc_data_host_scsi=>11.575390100479126, ↵
: collect_inventory_for_targets=>59.012681007385254, ↵
: parse_vc_data=>0.15207147598266602, ↵
: parse_targeted_inventory=>0.15630817413330078, ↵
: db_save_inventory=>65.91589498519897, ↵
: save_inventory=>65.9160327911377, ↵
: ems_refresh=>125.0889003276825}
```

The actual realtime values displayed vary with provider type. All providers report one or more of the following timings:

- ✧ **:ems\_refresh** (total time to perform the refresh)

- ✧ **:collect\_inventory\_for\_targets** (for VMware and RHV providers this is the time to extract data from the EMS)
- ✧ **:parse\_targeted\_inventory** (for VMware and RHV providers this is the time to parse the inventory. For all other providers this is the time taken to extract and parse data from the EMS)
- ✧ **:save\_inventory** (time saving the inventory into the VMDB)

VMware providers additionally report one or more of the following sub-timings:

- ✧ **:collect\_inventory\_for\_targets**
  - **:get\_ems\_data** (time retrieving EMS information)
  - **:get\_vc\_data** (time retrieving vCenter inventory such as VMs, dvportgroups, hosts, clusters etc.)
  - **:filter\_vc\_data** (time filtering vCenter inventory)
  - **:get\_vc\_data\_ems\_customization\_spec** (time retrieving customization spec inventory)
  - **:get\_vc\_data\_host\_scsi** (time retrieving storage device inventory)
- ✧ **:parse\_targeted\_inventory**
  - **:parse\_vc\_data** (time parsing vCenter inventory)
- ✧ **:save\_inventory**
  - **:db\_save\_inventory** (time saving the inventory to the VMDB)

RHV providers additionally report one or more of the following sub-timings:

- ✧ **:collect\_inventory\_for\_targets**
  - **:fetch\_host\_data** (retrieval time when targeted refresh is to a host)
  - **:fetch\_vm\_data** (retrieval time when targeted refresh is to a VM)
  - **:fetch\_all** (retrieval time for any other refresh)
- ✧ **:parse\_targeted\_inventory**
  - **:parse\_inventory**

'Legacy' providers additionally report the following timing:

- ✧ **:parse\_legacy\_inventory**

Performing the required calculation.<sup>[14]</sup> on the log line shown above reveals the following performance values:

```
Refresh timings:
get_ems_data:                0.032891 seconds
get_vc_data:                 3.063675 seconds
filter_vc_data:              0.000959 seconds
get_vc_data_host_scsi:       1.047531 seconds
collect_inventory_for_targets: 4.146032 seconds
parse_vc_data:               0.010229 seconds
```

```

parse_targeted_inventory:      0.010285 seconds
db_save_inventory:            2.471521 seconds
save_inventory:               2.471530 seconds
ems_refresh:                  6.628097 seconds

```

This shows that the two significant time components to this operation were extracting and parsing the inventory from vCenter (4.146 seconds), and loading the data into the database (2.472 seconds).

## 5.4. IDENTIFYING REFRESH PROBLEMS

Refresh problems are best identified by establishing baseline timings when the managed EMS is least busy. To determine the relative EMS collection and database load times, the `:collect_inventory_for_targets` and `:db_save_inventory` timing counters from *evm.log* can be plotted. For this example the `cfme-log-parsing/ems_refresh_timings.rb` script is used, as follows:

```

ruby ~/git/cfme-log-parsing/ems_refresh_timings.rb ↵
-i evm.log -o ems_refresh_timings.out

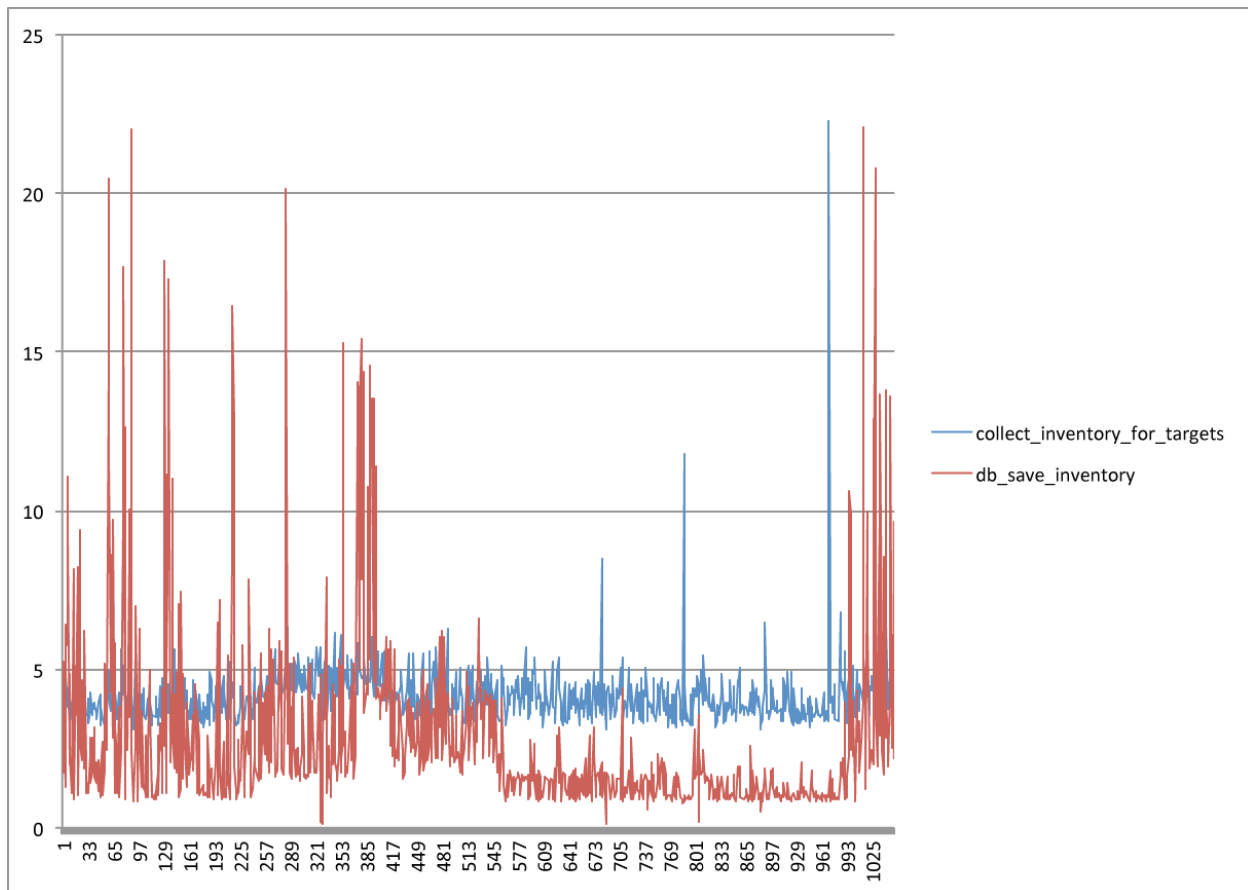
grep -A 13 "Vm: 1$" ems_refresh_timings.out | ↵
grep collect_inventory_for_targets | ↵
awk '{print $2}' > collect_inventory_for_targets.txt

grep -A 13 "Vm: 1$" ems_refresh_timings.out | ↵
grep db_save_inventory | ↵
awk '{print $2}' > db_save_inventory.txt

```

The contents of the two text files can then be plotted, as shown in [Figure 5.1, “Single VM EMS Refresh Component Timings, 24 Hour Period”](#).

**Figure 5.1. Single VM EMS Refresh Component Timings, 24 Hour Period**



A significant increase or wide variation in data extraction times from this baseline can indicate that the EMS is experiencing high load and not responding quickly to API requests.

Some variation in database load times throughout a 24 hour period is expected, but sustained periods of long load times can indicate that the database is overloaded.

## 5.5. TUNING REFRESH

There is little CloudForms tuning that can be done to improve the data extraction time of a refresh. If the extraction times vary significantly throughout the day then some investigation into the performance of the EMS itself may be warranted.

If database load times are high, then CPU, memory and I/O load on the database appliance should be investigated and if necessary tuned. The *top\_output.log* and *vmstat\_output.log* files in */var/www/miq/vmdb/log* on the database appliance can be used to correlate the times of high CPU and memory demand against the long database load times.

### 5.5.1. Configuration

The `:ems_refresh` section of the **Configuration** → **Advanced** settings is listed as follows:

```
:ems_refresh:
  :capture_vm_created_on_date: false
  :ec2:
    :get_private_images: true
    :get_shared_images: true
    :get_public_images: false
```

```

:public_images_filters:
- :name: image-type
  :values:
  - machine
:ignore_terminated_instances: true
:ansible_tower_configuration:
  :refresh_interval: 15.minutes
:foreman_configuration:
  :refresh_interval: 15.minutes
:foreman_provisioning:
  :refresh_interval: 1.hour
:full_refresh_threshold: 100
:hawkular:
  :refresh_interval: 15.minutes
:kubernetes:
  :refresh_interval: 15.minutes
:openshift:
  :refresh_interval: 15.minutes
:openshift_enterprise:
  :refresh_interval: 15.minutes
:raise_vm_snapshot_complete_if_created_within: 15.minutes
:refresh_interval: 24.hours
:scvmm:
  :refresh_interval: 15.minutes
:vmware_cloud:
  :get_public_images: false

```

#### 5.5.1.1. Refresh Interval

The **:refresh\_interval** defines a base frequency that a full refresh will be performed for a provider. The default value is 24 hours, although as can be seen this is overridden for several providers.

Refresh workers also however have a **Configuration** → **Advanced** setting called **:restart\_interval** which by default is set as **2.hours** (see [Section 2.6.1, “Worker Validation”](#)). Unless a provider connection broker is being used, each time a new refresh worker starts it queues a messages for itself to perform an initial full refresh. The following line from *evm.log* illustrates this behaviour:

```

... INFO -- : MIQ(ManageIQ::Providers::Redhat::InfraManager::
RefreshWorker::Runner#do_before_work_loop) EMS [rhvm] as [admin]
Queueing initial refresh for EMS

```



#### Note

Currently only the VMware provider uses a connection broker, called the **VIM Broker**

The net result is that even though a provider may have a **:refresh\_interval** setting of 24 hours, in practice a full refresh is often performed at the frequency of the worker's **:restart\_interval** value.

#### 5.5.1.2. Refresh Threshold

Although targeted refreshes are generally considerably faster than full refreshes, there is a break-even point after which a full refresh becomes more efficient to perform than many tens or hundreds of merged targeted requests. This point unfortunately varies between different CloudForms installations, and is dependant on the provider EMS type and API responsiveness, VMDB database I/O and CPU performance, and the number of managed objects within each provider.

There is a **Configuration** → **Advanced** setting called **:full\_refresh\_threshold**. This specifies the maximum number of concurrent targeted refreshes that should be attempted before being replaced by a single full request, by any provider in the region.

The default **:full\_refresh\_threshold** value is 100 and is global (provider-independent), however the value can be modified or overridden by provider type if required. For example to override the setting for all RHV providers in the region, the following lines could be added to the **:ems\_refresh** section:

```
:rhevms:
  :full_refresh_threshold: 200
```

If the **:full\_refresh\_threshold** value is triggered, there will be a corresponding "Escalating" line written to *evm.log*, for example:

```
... MIQ(ManageIQ::Providers::Vmware::InfraManager::Refresher# ↵
preprocess_targets) Escalating to full refresh for EMS: [vCenter6], ↵
id: [10000000000002].
```

Such escalations can happen if too many events are received in a short period of time (section [Chapter 9, Event Handling](#) discusses blacklisting events).

#### 5.5.1.2.1. Calculating a Suitable Refresh Threshold

Finding the correct value for the refresh threshold for each CloudForms installation is important. The duration of the refresh process should be as short as possible for several reasons, including the following:

1. New VM instances are not recognised until an EMS refresh completes. This can have an adverse impact on other related activities such as VM provisioning.
2. A new EMS refresh operation cannot start until any prior refreshes have completed. If an existing (long) refresh has just missed the creation of a new object but is still in progress, a further refresh may be needed to capture the new object.

The optimum value for the refresh threshold can only be found by examining the actual refresh times encountered for each provider. Having multiple providers of the same type in the same region can complicate this process, and if the optimal thresholds for each provider are found to be very different it may be worth splitting providers between regions.

For example a CloudForms installation managing a single VMware provider with approximately 800 VMs was examined to find the optimum refresh threshold. The *evm.log* file for the CFME appliance with the Provider Inventory role was examined over a period of several days.

It was discovered that the average time for a targeted EMS refresh for a single VM was approximately 9 seconds, and that this increased by roughly 3 seconds for each additional VM added to the targeted refresh list.

Over the same time period the average time for a full EMS refresh was approximately 225 seconds. A more suitable **full\_refresh\_threshold** for this particular installation would therefore be:

---

$$(225 - 6) / 3 = 73$$

---

[13] Unfortunately the timings are often incorrect until [https://bugzilla.redhat.com/show\\_bug.cgi?id=1424716](https://bugzilla.redhat.com/show_bug.cgi?id=1424716) is fixed. The correct times can usually be calculated by subtracting the previous counter values from the current

[14] Example scripts to perform the calculations are available from <https://github.com/RHsyseng/cfme-log-parsing>

## CHAPTER 6. CAPACITY & UTILIZATION

The processing of Capacity & Utilization (C&U) data is both resource intensive, and with some providers also time critical as real-time counters are only stored for a short period of time. These two factors must be carefully considered and monitored when deploying CloudForms to manage large virtual infrastructures or clouds.



### Note

C&U processing is often referred to as *metrics processing*

### 6.1. COMPONENT PARTS

As discussed in [Chapter 2, Architecture](#), there are three CFME appliance roles connected with C&U processing:

- ✧ Capacity & Utilization Coordinator
- ✧ Capacity & Utilization Data Collector
- ✧ Capacity & Utilization Data Processor

#### 6.1.1. C&U Coordination

Every 3 minutes a message is queued for the C&U Coordinator to begin processing<sup>[15]</sup>. The Coordinator schedules the data collections for all of the managed objects in the VMDB, and queues messages for the C&U Data Collector to retrieve metrics for any objects for which a collection is due. The default time interval between collections (the *capture threshold*) is 10 minutes, but for VMs, Clusters and Hosts this is expanded to 50 minutes, and for storage, 60 minutes.

The capture thresholds are defined in the **:performance** section of the **Configuration** → **Advanced** settings, as follows:

```
:performance:
  :capture_threshold:
    :default: 10.minutes
    :ems_cluster: 50.minutes
    :host: 50.minutes
    :storage: 60.minutes
    :vm: 50.minutes
  :capture_threshold_with_alerts:
    :default: 1.minutes
    :host: 20.minutes
    :vm: 20.minutes
```

If a control alert is defined for an object type, the shorter capture thresholds defined under **:capture\_threshold\_with\_alerts** are used to ensure a faster response.

The message created by the C&U Coordinator specifies a counter type to retrieve ("realtime" or "hourly"), and an optional time range to collect data for.

#### 6.1.2. Data Collection



The data collection phase of C&U processing is split into two parts: capture, and initial processing and storage, both performed by the C&U Data Collector.

### 6.1.2.1. Capture

Upon dequeue of a new message the Data Collector makes a connection to the provider's metrics source API to retrieve the data for the object and time range specified in the message.

The following table shows the metrics sources for the supported providers.

**Table 6.1. Metrics Sources**

Provider	Metrics source
VMware	vCenter Server statistics
Red Hat Virtualization	Data Warehouse database (default: ovirt_engine_history)
OpenStack CloudManager (OSP 6-9)	Ceilometer
OpenStack CloudManager (OSP 10+)	Gnocchi
OpenStack InfraManager (Director)	Ceilometer
Amazon	Amazon CloudWatch
Azure	Azure Monitor
Google	Google Cloud Monitoring API (superseded by Stackdriver)
OpenShift	Hawkular

A successful capture is written to *evm.log*, as follows.<sup>[16]</sup>:

```
... MIQ(ManageIQ::Providers::Vmware::InfraManager::Vm#perf_capture) ↵
[realtime] Capture for ↵
ManageIQ::Providers::Vmware::InfraManager::Vm name: [VP23911], ↵
id: [1000000000789]...Complete - Timings: ↵
```

```
{:capture_state=>0.08141517639160156, ↵
:vim_connect=>0.06982016563415527, ↵
:capture_intervals=>0.014894962310791016, ↵
:capture_counters=>0.20465683937072754, ↵
:build_query_params=>0.0009250640869140625, ↵
:num_vim_queries=>1, ↵
:vim_execute_time=>0.2935605049133301, ↵
:perf_processing=>0.10299563407897949, ↵
:num_vim_trips=>1, ↵
:total_time=>0.7732744216918945}
```

### 6.1.2.2. Initial Processing & Storage

The realtime data retrieved from the metrics source is stored in the VMDB in the *metrics* table, and in one of 24 sub-tables called *metrics\_00* to *metrics\_23* (based on the timestamp, each table corresponds to an hour). Dividing the records between sub-tables simplifies some of the data processing tasks. Once the data is stored, the Data Collector queues messages to the Data Processor to perform the hourly, daily and parental rollups.

The successful completion of this initial processing stage can be seen in *evm.log*, as follows:

```
... MIQ(ManageIQ::Providers::Vmware::InfraManager::Vm#perf_process) ↵
[realtime] Processing for ↵
ManageIQ::Providers::Vmware::InfraManager::Vm name: [VR11357], ↵
id: [10000000000043], ↵
for range [2017-01-25T05:59:00Z - 2017-01-25T06:50:20Z]... ↵
Complete - Timings: ↵
{:process_counter_values=>0.019364118576049805, ↵
:db_find_prev_perfs=>0.015059232711791992, ↵
:process_perfs=>0.2053236961364746, ↵
:process_perfs_db=>1.8572983741760254, ↵
:total_time=>2.1722793579101562}
```

### 6.1.3. Data Processing

The C&U Data Processors periodically perform the task of 'rolling up' the realtime data. Rollups are performed hourly and daily, and counters for more granular objects such as virtual machines are aggregated into the counters for their parent objects. For example for a virtual infrastructure such as VMware or Red Hat Virtualization, the parent rollup process would include the following objects:

```
VM {hourly,daily} → Host {realtime,hourly,daily} → Cluster {hourly,daily} → Provider {hourly,daily} →
Region {hourly,daily} → Enterprise
```

Rollup data is stored in the *metrics\_rollups* table and in one of 12 sub-tables called *metric\_rollups\_01* to *metric\_rollups\_12* (each table corresponds to a month).

Additional analysis is performed on the hourly rollup data to identify bottlenecks, calculate chargeback metrics, and determine normal operating range and right-size recommendations. The completion of a successful rollup is written to *evm.log*, as follows:

```
... INFO -- : MIQ(ManageIQ::Providers::Vmware::InfraManager::Vm# ↵
perf_rollup) [hourly] Rollup for ManageIQ::Providers::Vmware:: ↵
InfraManager::Vm name: [ranj001], id: [100000000000752] for time: ↵
[2016-12-13T02:00:00Z]...Complete - Timings: ↵
{:server_dequeue=>0.0035326480865478516, ↵
```

```
:db_find_prev_perf=>3.514737129211426, ↵
:rollup_perfs=>27.559985399246216, ↵
:db_update_perf=>7.901974678039551, ↵
:process_perfs_tag=>1.1872785091400146, ↵
:process_bottleneck=>2.1828694343566895, ↵
:total_time=>54.16198229789734}
```

## 6.2. DATA RETENTION

Capacity and Utilization data is not retained indefinitely in the VMDB. By default hourly and daily rollup data is kept for 6 months after which it is purged, and realtime data samples are purged after 4 hours. These retention periods for C&U data are defined in the **:performance** section of the **Configuration** → **Advanced** settings, as follows:

```
:performance:
...
:history:
...
:keep_daily_performances: 6.months
:keep_hourly_performances: 6.months
:keep_realtime_performances: 4.hours
```

## 6.3. CHALLENGES OF SCALE

The challenges of scale for capacity & utilization are related to the time constraints involved when collecting and processing the data for several thousand objects in fixed time periods, for example:

- ✎ Retrieving realtime counters before they are deleted from the EMS
- ✎ Rolling up the realtime counters before the records are purged from the VMDB
- ✎ Inter-worker message timeout

When capacity & utilization is not collecting and processing the data consistently, other CloudForms capabilities that depend on the metrics - such as chargeback or rightsizing - become unreliable.

The challenges are addressed by adding concurrency - scaling out both the data collection and processing workers - and by keeping each step in the process as short as possible to maximise throughput.

## 6.4. MONITORING CAPACITY & UTILIZATION PERFORMANCE

As with EMS refresh, C&U data collection has two significant phases that each contribute to the overall performance:

- ✎ Extracting and parsing the metrics from the EMS
  - Network latency to the EMS
  - Time waiting for the EMS to process the capture and return data
  - CPU cycles performing initial processing
- ✎ Storing the data into the VMDB

- Network latency to the database
- Database appliance CPU, memory and I/O resources

The line printed to *evm.log* at the completion of each stage of the operation contains detailed timings, and these can be used to determine bottlenecks. The typical log lines for VMware C&U capture and initial processing can be parsed using a script such as `perf_process_timings.rb`.<sup>[17]</sup>, for example:

```
Capture timings:
  build_query_params:          0.000940 seconds
  vim_connect:                1.396388 seconds
  capture_state:              0.038595 seconds
  capture_intervals:          0.715417 seconds
  capture_counters:           1.585664 seconds
  vim_execute_time:           2.039972 seconds
  perf_processing:            0.044047 seconds
  num_vim_queries:            1.000000
  num_vim_trips:              1.000000
Process timings:
  process_counter_values:      0.043278 seconds
  db_find_prev_perfs:          0.010970 seconds
  process_perfs:               0.540629 seconds
  process_perfs_db:            3.387275 seconds
```

C&U data processing is purely a CPU and database-intensive activity. The rollup timings can be extracted from *evm.log* in a similar manner

```
Rollup timings:
  db_find_prev_perf:          0.014738
  rollup_perfs:               0.193929
  db_update_perf:             0.059067
  process_perfs_tag:           0.000054
  process_bottleneck:         0.005456
  total_time:                  0.372196
```

## 6.5. IDENTIFYING CAPACITY AND UTILIZATION PROBLEMS

The detailed information written to *evm.log* can be used to identify problems with capacity and utilization

### 6.5.1. Coordinator

With a very large number of managed objects the C&U Coordinator becomes unable to create and queue all of the required **perf\_capture\_realtime** messages within its own message timeout period of 600 seconds. An indeterminate number of managed objects will have no collections scheduled for that time interval. An extraction of lines from *evm.log* that illustrates the problem is as follows:

```
... INFO -- : MIQ(MiqGenericWorker::Runner#get_message_via_drb) ↵
Message id: [10000221979280], MiqWorker id: [10000001075231], ↵
Zone: [OCP], Role: [ems_metrics_coordinator], Server: [], ↵
Ident: [generic], Target id: [], Instance id: [], Task id: [], ↵
Command: [Metric::Capture.perf_capture_timer], Timeout: [600], ↵
```

```

Priority: [20], State: [dequeue], Deliver On: [], Data: [], ↵
Args: [], Dequeued in: [2.425676767] seconds

... INFO -- : MIQ(Metric::Capture.perf_capture_timer) Queueing ↵
performance capture...

... INFO -- : MIQ(MiqQueue.put) Message id: [10000221979391], ↵
id: [], Zone: [OCP], Role: [ems_metrics_collector], Server: [], ↵
Ident: [openshift_enterprise], Target id: [], ↵
Instance id: [10000000000113], Task id: [], ↵
Command: [ManageIQ::Providers::Kubernetes::ContainerManager:: ↵
ContainerNode.perf_capture_realtime], Timeout: [600], ↵
Priority: [100], State: [ready], Deliver On: [], Data: [], ↵
Args: [2017-03-23 20:59:00 UTC, 2017-03-24 18:33:23 UTC]

...

... INFO -- : MIQ(MiqQueue.put) Message id: [10000221990773], ↵
id: [], Zone: [OCP], Role: [ems_metrics_collector], Server: [], ↵
Ident: [openshift_enterprise], Target id: [], ↵
Instance id: [100000000032703], Task id: [], ↵
Command: [ManageIQ::Providers::Kubernetes::ContainerManager:: ↵
ContainerGroup.perf_capture_realtime], Timeout: [600], ↵
Priority: [100], State: [ready], Deliver On: [], Data: [], ↵
Args: [2017-03-24 18:10:20 UTC, 2017-03-24 18:43:15 UTC]

... ERROR -- : MIQ(MiqQueue#deliver) Message id: [10000221979280], ↵
timed out after 600.002976954 seconds. Timeout threshold [600]

```

Such problems can be detected by looking for message timeouts in the log using a command such as the following:

```
egrep "Message id: \[\\d+\\], timed out after" evm.log
```

Any lines matched by this search can be traced back using the PID field in the log line to determine the operation that was in process when the message timeout occurred.

### 6.5.2. Data Collection

Some providers keep realtime performance data for a limited time period, and if not retrieved in that time period, it is lost. For example VMware ESXi servers sample performance counter instances for themselves and the virtual machines running on them every 20 seconds, and maintain 180 realtime instance data points for a rolling 60 minute period. Similarly the OpenStack Gnocchi 'low' and 'high' archive policies on OSP 10+ only retain the finest granularity collection points for one hour (although this is configurable). There is therefore a 60 minute window during which performance information for each VMware or OpenStack element must be collected. If the performance data samples are not collected before that rolling 60 minutes is up, the data is lost.

The C&U Coordinator schedules a new VM, host or cluster realtime performance collection 50 minutes after the last data sample was collected for that object. This allows up to 10 minutes for the message to be dequeued and processed, before the realtime metrics are captured. In a large VMware or OpenStack environment the messages for the C&U Data Collectors can take longer than 10 minutes to be dequeued, meaning that some realtime data samples are lost. As the environment grows (more VMs) the problem slowly becomes worse.

There are several types of log line written to *evm.log* that can indicate C&U data collection problems.

### 6.5.2.1. Messages Still Queued from Last C&U Coordinator Run

Before the C&U Coordinator starts queueing new messages it calls an internal method **perf\_capture\_health\_check** that prints the number of capture messages still queued from previous C&U Coordinator schedules. If the C&U Data Collectors are keeping pace with the rate of message additions, there should be approximately 0 messages remaining in the queue when the C&U Coordinator runs. If the C&U Data Collectors are not dequeuing and processing messages quickly enough there will be a backlog.

Searching for the string "perf\_capture\_health\_check" on the CFME appliance with the active C&U Coordinator role will show the state of the queue before the C&U Coordinator adds further messages, and any backlog will be visible.

```
... INFO -- : MIQ(Metric::Capture.perf_capture_health_check) ↵
520 "realtime" captures on the queue for zone [VMware Zone] - ↵
oldest: [2016-12-13T07:14:44Z], recent: [2016-12-13T08:02:32Z]
... INFO -- : MIQ(Metric::Capture.perf_capture_health_check) ↵
77 "hourly" captures on the queue for zone [VMware Zone] - ↵
oldest: [2016-12-13T08:02:15Z], recent: [2016-12-13T08:02:17Z]
... INFO -- : MIQ(Metric::Capture.perf_capture_health_check) ↵
0 "historical" captures on the queue for zone [VMware Zone]
```

### 6.5.2.2. Long Dequeue Times

Searching for the string "MetricsCollectorWorker::Runner#get\_message\_via\_drb" will show the log lines printed when the C&U Data Collector messages are dequeued. **A "Dequeued in" value higher than 600 seconds is likely to result in lost realtime data for VMware or OpenStack providers.**

```
... INFO -- : MIQ(ManageIQ::Providers::Vmware::InfraManager:: ↵
MetricsCollectorWorker::Runner#get_message_via_drb) ↵
Message id: [1000032258093], MiqWorker id: [1000000120960], ↵
Zone: [VMware], Role: [ems_metrics_collector], Server: [], ↵
Ident: [vmware], Target id: [], Instance id: [10000000000060], ↵
Task id: [], Command: [ManageIQ::Providers::Vmware::InfraManager:: ↵
Vm.perf_capture_realtime], Timeout: [600], Priority: [100], ↵
State: [dequeue], Deliver On: [], Data: [], Args: [], ↵
Dequeued in: [789.95923544] seconds
```

### 6.5.2.3. Missing Data Samples - Data Collection

Searching for the string "expected to get data" can reveal whether requested data sample points were not available for retrieval from the EMS, as follows:

```
... WARN -- : MIQ(ManageIQ::Providers::Vmware::InfraManager::HostEsx ↵
#perf_capture) [realtime] For ManageIQ::Providers::Vmware:: ↵
InfraManager::HostEsx name: [esx04], id: [10000000000023], ↵
expected to get data as of [2016-12-13T01:20:00Z], ↵
but got data as of [2016-12-13T02:00:20Z].
```

### 6.5.2.4. Missing Data Samples - Data Loading

Searching for the string "performance rows...Complete" reveals the number of performance rows that were successfully processed and loaded into the VMDB, as follows:

-

```
... INFO -- : MIQ(ManageIQ::Providers::Vmware::InfraManager::Vm#
perf_process) [realtime] Processing 138 performance rows...Complete
- Added 138 / Updated 0
```

For VMware this should be less than 180 per collection interval (180 points is the maximum retained for an hour). The presence of a number of lines with a value of 180 usually indicates that some realtime data samples have been lost.

### 6.5.2.5. Unresponsive Provider

In some cases the CloudForms processes are working as expected, but the provider EMS is overloaded and not responding to API requests. To determine the relative EMS connection and query times for a VMware provider, the `:vim_connect` and `:vim_execute_time` timing counters from `evm.log` can be plotted. For this example the `perf_process_timings.rb` script can be used, as follows:

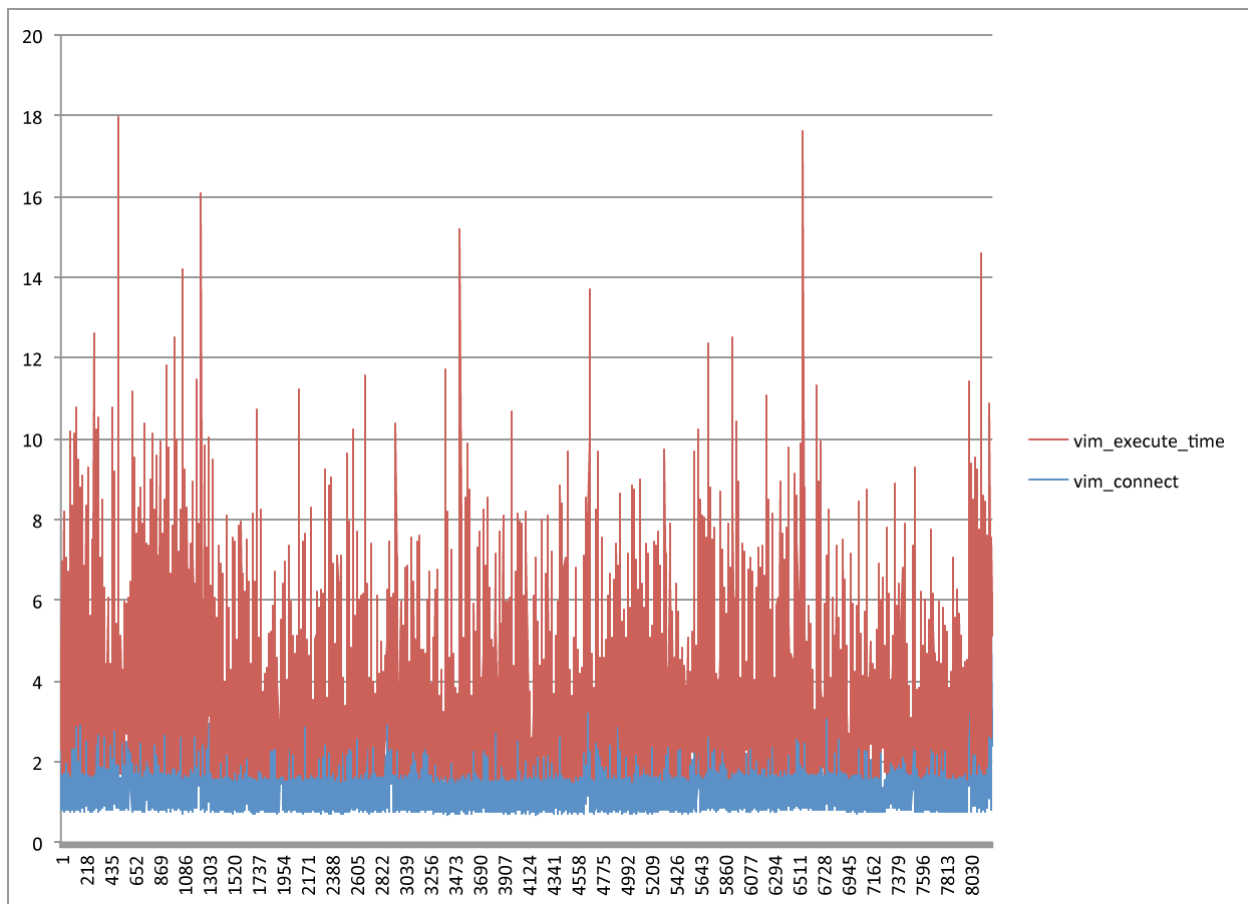
```
ruby ~/git/cfme-log-parsing/perf_process_timings.rb
-i evm.log -o perf_process_timings.out

egrep -A 22 "Worker PID:\s+10563" perf_process_timings.out |
grep vim_connect | awk '{print $2}' > vim_connect_times.txt

egrep -A 22 "Worker PID:\s+10563" perf_process_timings.out |
grep vim_execute_time | awk '{print $2}' > vim_execute_times.txt
```

The contents of the two text files can then be plotted, as shown in [Figure 6.1, “VMware Provider C&U Connect and Execute Timings, Single Worker, 24 Hour Period”](#).

**Figure 6.1. VMware Provider C&U Connect and Execute Timings, Single Worker, 24 Hour Period**



In this example the stacked lines show a consistent connect time, and an execute time that is slightly fluctuating but still within acceptable bounds for reliable data collection.

### 6.5.3. Data Processing

The rollup and associated bottleneck and performance processing of the C&U data is less time sensitive, **although must still be completed in the 4 hour realtime performance data retention period.**

With a very large number of managed objects and insufficient worker processes, the time taken to process the realtime data can exceed the 4 hour period, meaning that that data is lost. The time taken to process the hourly rollups can exceed an hour, and the rollup process never keeps up with the rate of messages.

The count of messages queued for processing by the Data Processor can be extracted from *evm.log*, as follows:

```
grep 'count for state=\["ready"\]' evm.log |
egrep -o "\"ems_metrics_processor\"=>[[:digit:]]+\"

"ems_metrics_processor"=>16612
"ems_metrics_processor"=>16494
"ems_metrics_processor"=>12073
"ems_metrics_processor"=>12448
"ems_metrics_processor"=>13015
...
```

**The "Dequeued in" and "Delivered in" times for messages processed by the MiqEmsMetricsProcessorWorkers can be used as guidelines for overall throughput, for example:**



```

... INFO -- : MIQ(MiqEmsMetricsProcessorWorker::Runner#
get_message_via_drb) Message id: [1000032171247], MiqWorker id:
[10000000253077], Zone: [VMware], Role: [ems_metrics_processor],
Server: [], Ident: [ems_metrics_processor], Target id: [],
Instance id: [1000000001228], Task id: [],
Command: [ManageIQ::Providers::Vmware::InfraManager::Vm.perf_rollup],
Timeout: [1800], Priority: [100], State: [dequeue],
Deliver On: [2016-12-13 03:00:00 UTC], Data: [],
Args: ["2016-12-13T02:00:00Z", "hourly"],
Dequeued in: [243.967960013] seconds

... INFO -- : MIQ(MiqQueue#delivered) Message id: [1000032171247],
State: [ok],
Delivered in [0.202901147] seconds

```

When C&U is operating correctly, for each time-profile instance there should be one daily record and at least 24 hourly records for each powered-on VM. There should also be at most 5 of the metrics\_## tables that contain more than zero records.

The following SQL query can be used to confirm that the records are being processed correctly:

```

select resource_id, date_trunc('day',timestamp) as collect_date,
resource_type, capture_interval_name, count(*)
from metric_rollups
where resource_type like '%Vm%'
group by resource_id, collect_date, resource_type, capture_interval_name
order by resource_id, collect_date, resource_type, capture_interval_name,
count
;
..._id | collect_date          | resource_type | capture_int... | count
-----+-----+-----+-----+-----
-
...
    4 | 2017-03-17 00:00:00 | VmOrTemplate | daily          |      1
    4 | 2017-03-17 00:00:00 | VmOrTemplate | hourly         |     24
    4 | 2017-03-18 00:00:00 | VmOrTemplate | daily          |      1
    4 | 2017-03-18 00:00:00 | VmOrTemplate | hourly         |     24
    4 | 2017-03-19 00:00:00 | VmOrTemplate | daily          |      1
    4 | 2017-03-19 00:00:00 | VmOrTemplate | hourly         |     24
    4 | 2017-03-20 00:00:00 | VmOrTemplate | daily          |      1
    4 | 2017-03-20 00:00:00 | VmOrTemplate | hourly         |     24
...

```

## 6.6. RECOVERING FROM CAPACITY AND UTILIZATION PROBLEMS

If C&U realtime data is not collected it is generally lost. Some historical information is retrievable using C&U gap collection (see [Figure 6.2, “C&U Gap Collection”](#)), but this is of a lower granularity than the realtime metrics that are usually collected. **Gap collection is fully supported with VMware providers, but also works in a more limited capacity with some other providers such as OpenShift.**

**Figure 6.2. C&U Gap Collection**

## Diagnostics Zone "Default Zone" (current)

Roles by Servers

Servers by Roles

Servers

Collect Logs

C &amp; U Gap Collection

## Collection Options

Timezone	(GMT+00:00) UTC
Start Date	03/14/2017
End Date	03/31/2017

Note: Gap Collection is only available for VMware vSphere Infrastructures

## 6.7. TUNING CAPACITY AND UTILIZATION

Tuning capacity and utilization generally involves ensuring that the VMDB is running optimally, and adding workers and CFME appliances to scale out the processing capability.

### 6.7.1. Scheduling

Messages for the `ems_metrics_coordinator` (C&U coordinator) server role are processed by a Generic or Priority worker. These workers also process automation messages, which are often long-running. For larger CloudForms installations it can be beneficial to separate the C&U Coordinator and Automation Engine server roles onto different CFME appliances.

### 6.7.2. Data Collection

The `metrics_00` to `metrics_23` VMDB tables have a high rate of insertions and deletions, and benefit from regular reindexing. The database maintenance scripts that can be installed from `appliance_console` run a `/usr/bin/hourly_reindex_metrics_tables` script that reindexes one of the tables every hour.

If realtime data samples are regularly being lost, there are two remedial measures that can be taken.

#### 6.7.2.1. Increasing the Number of Data Collectors

The default number of C&U Data Collector workers per appliance is 2. This can be increased to a maximum of 9, although consideration should be given to the additional CPU and memory requirements that an increased number of workers will place on an appliance. It may be more appropriate to add further appliances and scale horizontally.

For larger CloudForms installations it can be beneficial to separate the C&U Data Collector and Automation Engine server roles onto different CFME appliances, as both are resource intensive. Very large CloudForms installations (managing several thousand objects) may benefit from dedicated CFME appliances in the provider zones exclusively running the C&U data collector role.

#### 6.7.2.2. Reducing the Collection Interval

The collection interval can be reduced from 50 minutes to a smaller value (for example 20-30 minutes) allowing more time for collection scheduling and for queuing wait time. The delay or "capture threshold" is defined in the **:performance** section of the **Configuration** → **Advanced** settings, as follows:

```
:performance:
  :capture_threshold:
    :ems_cluster: 50.minutes
    :host: 50.minutes
    :storage: 60.minutes
    :vm: 50.minutes
```

Reducing the collection interval places a higher overall load on both the EMS and CloudForms appliances, so this option should be considered with caution.

### 6.7.3. Data Processing

If C&U data processing is taking too long to process the rollups for all objects, the number of C&U Data Processor workers can be increased from the default of 2 up to a maximum of 4 per appliance.

As before, consideration should be given to the additional CPU and memory requirements that an increased number of workers will place on an appliance. Adding further CFME appliances to the zone may be more appropriate.

For larger CloudForms installations it can be beneficial to separate the C&U Data Processor and Automation Engine server roles onto different CFME appliances, as both are resource intensive. CloudForms installations managing several thousand objects may benefit from dedicated CFME appliances in the provider zones exclusively running the C&U Data Processor role.

---

[15] The default value is 3 minutes, but this can be changed in 'Advanced' settings

[16] As with the EMS collection timings, the C&U timings are sometimes incorrect until [https://bugzilla.redhat.com/show\\_bug.cgi?id=1424716](https://bugzilla.redhat.com/show_bug.cgi?id=1424716) is fixed. When incorrect the correct times can be calculated by subtracting the previous counter values from the current

[17] From <https://github.com/RHsyseng/cfme-log-parsing>

## CHAPTER 7. AUTOMATE

Automate is an important component of CloudForms that performs many tasks, such as:

- ✧ Event processing
- ✧ Service provisioning and retirement
- ✧ VM and instance provisioning and retirement

### 7.1. AUTOMATION ENGINE

Automate requests and tasks are processed by an **automation engine running in either a Generic or Priority worker**. Priority workers dequeue high priority (priority 20) messages, which include the following types of automate task:

- ✧ Automate instances started from a custom button in the WebUI
- ✧ Processing events through the automate event switchboard

Generic workers dequeue all priorities of messages for the server roles that they manage. These include the less time-sensitive automation jobs such as service requests and provisioning workflows, which are typically queued at a priority of 100.

An automate message's "Args" list is passed to the automation engine which instantiates the requested automate entry point instance (usually under */System/Process* in the automate datastore). The Generic worker does not process any further messages until the automate instance, its children, and any associated methods have completed, or a state machine method exits with **\$evm.root['ae\_result'] = 'retry'**.

This process can be observed in *evm.log* by following the message (Message id: [1038885]) that initiated a virtual machine provisioning operation to a Red Hat Virtualization provider. The message is dequeued and passed to the automation engine, as follows:

```
... INFO -- : MIQ(MiqGenericWorker::Runner#get_message_via_drb) ↵
Message id: [1038885], MiqWorker id: [3758], Zone: [default], ↵
Role: [automate], Server: [], Ident: [generic], Target id: [], ↵
Instance id: [], Task id: [miq_provision_147], ↵
Command: [MiqAeEngine.deliver], Timeout: [600], Priority: [100], ↵
State: [dequeue], Deliver On: [], Data: [], ↵
Args: [{:object_type=>"ManageIQ::Providers::Redhat::InfraManager:: ↵
Provision", :object_id=>147, :attrs=>{"request"=>"vm_provision"}, ↵
:instance_name=>"AUTOMATION", :user_id=>1, :miq_group_id=>2, ↵
:tenant_id=>1}], Dequeued in: [3.50249721] seconds

... INFO -- : Q-task_id([miq_provision_147]) MIQ(MiqAeEngine.deliver) ↵
Delivering {"request"=>"vm_provision"} for object ↵
[ManageIQ::Providers::Redhat::InfraManager::Provision.147] ↵
with state [] to Automate
```

The automation engine processes the first 9 states in the state machine, but does not complete the processing of the message until the `CheckProvisioned` method exits with **ae\_result="retry"**. The automation engine is seen re-queueing a new message for delivery in 60 seconds, and the current message is flagged as "Delivered" after 50.708623233 seconds of processing time, as follows:

■

```

... INFO -- : Q-task_id([miq_provision_147]) MIQ(MiqAeEngine.deliver) ↵
Requeuing :object_type=>"ManageIQ::Providers::Redhat::InfraManager:: ↵
Provision", :object_id=>147, :attrs=>{"request"=>"vm_provision"}, ↵
:instance_name=>"AUTOMATION", :user_id=>1, :miq_group_id=>2, ↵
:tenant_id=>1, :state=>"CheckProvisioned", :ae_fsm_started=>nil, ↵
:ae_state_started=>"2017-03-22 13:05:34 UTC", :ae_state_retries=>1, ↵
:ae_state_previous=>"---\n\n"/Bit63/Infrastructure/VM/ ↵
Provisioning/StateMachines/VMPvision_vm/template\":"\n ae_state: ↵
CheckProvisioned\n ae_state_retries: 1\n ae_state_started: ↵
2017-03-22 13:05:34 UTC\n"} for object ↵
[ManageIQ::Providers::Redhat::InfraManager::Provision.147] with state ↵
[CheckProvisioned] to Automate for delivery in [60] seconds

... INFO -- : Q-task_id([miq_provision_147]) ↵
MIQ(ManageIQ::Providers::Redhat::InfraManager::Provision# ↵
after_ae_delivery) ae_result="retry"

... INFO -- : Q-task_id([miq_provision_147]) MIQ(MiqQueue#delivered) ↵
Message id: [1038885], State: [ok], Delivered in [50.708623233] seconds

```

The retry allows the Generic worker to dequeue and process the next message.

## 7.2. CHALLENGES OF SCALE

There are several challenges of scale for automate.

### 7.2.1. Zone-Related Considerations

There are some zone-related implications that should be considered when running automation tasks in a large multi-zone CloudForms deployment.

#### 7.2.1.1. Automation Engine Enablement

All zones should have at least one CFME appliance with the Automation Engine role set, to process zone-related events. Even zones that have no providers associated with them will generate events such as `evm_worker_stop`, or `evm_worker_memory_exceeded`. If no appliance in a zone has the Automation Engine role enabled, the messages for these events will accumulate in the `miq_queue` table and never be removed.

#### 7.2.1.2. Services

Most automate messages specify the zone that the automate action should be performed in, however an exception to this is found in the messages that initiate service template provision requests and tasks. These messages don't specify a zone and can therefore run on any appliance in the region that has the Automation Engine role enabled. This can lead to unexpected behaviour.

For example if employing services to provision virtual machines using the `/ManageIQ/Service/Provisioning/StateMachines/ServiceProvision_Template` class, the messages are zone independent as described. The message to execute the first child service template provision task however is queued for a worker with the `ems_operations` role, but in the same zone as the CFME appliance creating the message. If there are no CFME appliances in the zone with the Provider Operations role enabled, the task will not be dequeued and processed, and the service provision will appear to hang.

Similarly services that have a catalog item type of "Generic" can run in any zone that has the Automation Engine role enabled. This might not be the intended action if the workflow is intended to run in a specific zone, such as in an environment where network access rules or firewall restrictions might be appliance-specific.

### 7.2.1.3. RESTful API

When an automation request is submitted via the API, the automate task is queued by default on the same appliance that the web service is running on. This will be dequeued to run by any appliance with the Automation Engine role set *in the same zone*. If the WebUI/web service appliances have been placed in their own zone, this may not necessarily be the desired behaviour.

## 7.3. IDENTIFYING AUTOMATE PROBLEMS

There are several problems that can be seen when running automation workflows in large-scale CloudForms deployments.

### 7.3.1. Requests Not Starting

By default each CFME appliance has 2 Priority and 2 Generic workers. If both of the Generic workers are busy processing long-running automate tasks or high priority messages (such as from an event storm), no further priority 100 automate messages will be dequeued and processed until either of the workers completes their current task. This is often observed in larger deployments when service or automation requests appear to remain in a "Pending" state for a long time.

### 7.3.2. Long Running Tasks Timing Out

The default message timeout for automate messages is 600 seconds, which means that the combined execution times of all automate methods that share a common `$evm.root` object must be less than 10 minutes. If this time limit is exceeded the automate method will be deemed "non responsive" and terminated, and the Generic worker running the automation will exit and be re-spawned. This timer is only reset if a state machine method exits with `$evm.root['ae_result'] = 'retry'`.

The timeout mechanism can be observed in `evm.log`, as follows:

```
... ERROR -- : <AutomationEngine> Terminating non responsive method ↵
with pid 29188

... ERROR -- : <AutomationEngine> <AEMethod test> The following error ↵
occurred during method evaluation:

... ERROR -- : <AutomationEngine> <AEMethod test> SignalException: ↵
SIGTERM

... ERROR -- : MIQ(MiqQueue#deliver) Message id: [1054092], timed ↵
out after 600.03190583 seconds. Timeout threshold [600]

... INFO -- : MIQ(MiqQueue#delivered) Message id: [1054092], ↵
State: [timeout], Delivered in [600.047235602] seconds
```

```
... ERROR -- : MIQ(MiqGenericWorker::Runner) ID [3758] PID [3149] ↵
GUID [d8bbe584-0e0f-11e7-a1a8-001a4aa0151a] ↵
Exiting worker due to timeout error Worker exiting.
```

### 7.3.3. State Machine Retries Exceeded

If the number of retries attempted by a state machine state reaches the limit defined in the class schema, an error will be logged to *evm.log*, as follows:

```
... ERROR -- : Q-task_id([automation_task_13921]) State=<pre4> running ↵
raised exception: <number of retries <6> exceeded maximum of <5>>
```

## 7.4. TUNING AUTOMATE

Automate can be tuned for scale in several ways. The first is to add concurrency to the workers processing automate requests and tasks, so that more operations can be run at the same time.

Individual Ruby-based automate workflows can be made more reliable by adopting efficient automate coding techniques where possible to reduce the overall execution time.

### 7.4.1. Increasing Concurrency

The number of Priority workers per CFME appliance can be increased up to a maximum of 4, and Generic workers up to a maximum of 9. This will increase the concurrency at which automate messages can be processed, however worker count should only be increased after consideration of the additional CPU and memory requirements that an increased number of workers will place on an appliance.

For larger CloudForms installations it can be beneficial to separate any of the Capacity and Utilization, and the Automation Engine server roles onto different CFME appliances, as both are resource intensive. In very large CloudForms installations it can be beneficial to have dedicated appliances per zone with the Automation Engine role enabled, each with the maximum numbers of Generic and Priority workers.

### 7.4.2. Reducing Execution Time

There are two useful techniques that can be used to help keep the overall execution time of custom Ruby-based automation workflows within the 10 minute timeout period. The first is to use state machines as much as possible to model workflows, and to include **CheckCompleted** states after any asynchronous and potentially long-running operation. The **CheckCompleted** state methods check for completion of the prior state, and issue an **ae\_result="retry"** if the operation is incomplete.

The second is to use **\$evm.execute('create\_automation\_request',...)** rather than **\$evm.instantiate** to execute long-running instances. Using **\$evm.instantiate** to start another instance from a currently running method will execute the called instance synchronously. The calling method will wait until the instantiated instance completes before continuing. If the instantiated method integrates with an external system for example, this delay might be significant, and contributes towards the total message processing time.

The use of these two techniques can be illustrated with the following example. In this case a call is made using `$evm.instantiate` to run an instance `update_cmdb` that updates the IP address for a virtual machine in an external CMDB, but the external API call to the CMDB sometimes takes several minutes to complete. The existing in-line call is as follows:

```
$evm.instantiate("/Integration/Methods/update_cmdb?name=dbsrv01&
  ip=10.1.2.3")
```

To run the `update_cmdb` instance asynchronously, the call can be rewritten to run as a new automation request, for example:

```
options = {}
options[:namespace] = 'Integration'
options[:class_name] = 'Methods'
options[:instance_name] = 'update_cmdb'
options[:user_id] = $evm.root['user'].id
options[:attrs] = {
  'name' => 'dbsrv01',
  'ip'   => '10.1.2.3'
}
auto_approve = true

update_cmdb_request = $evm.execute('create_automation_request',
  options, 'admin', auto_approve)
```

If the calling method does not need to wait for the completion of `update_cmdb` then processing can continue, and minimal delay has been incurred. If `update_cmdb` should complete before the main processing can continue, the request ID can be saved, and a 'CheckCompleted' state added to the state machine, as follows:

```
update_cmdb_request = $evm.execute('create_automation_request',
  options, 'admin', auto_approve)
$evm.set_state_var(:request_id, update_cmdb_request.id)
$evm.root['ae_result'] = 'ok'
exit MIQ_OK
```

The following state in the state machine would be `check_cmdb_request`, containing code similar to the following:

```
update_cmdb_request =
  $evm.vmdb(:miq_request, $evm.get_state_var(:request_id))
case update_cmdb_request.state
when "pending", "active"
  $evm.log(:info, "Request still active, waiting for 30 seconds...")
  $evm.root['ae_retry_interval'] = '30.seconds'
  $evm.root['ae_result'] = 'retry'
when "finished"
  $evm.log(:info, "Request complete!")
  $evm.root['ae_result'] = 'ok'
else
  $evm.log(:warn, "Unexpected request status")
  $evm.root['ae_result'] = 'error'
end
exit MIQ_OK
```



Sometimes the called method needs to pass data back to the caller, and this can be returned via the request object's options hash. The called method **update\_cmdb** can retrieve its own request object and use the **set\_option** method to encode a key/value pair (where the value is a JSON-encoded hash) as follows:

```
request = $evm.root['automation_task'].automation_request
request.set_option(:return, JSON.generate({:status => 'success',
                                           :cmdb_return => 'update successful'}))
```

The options hash can be read from the request object by the caller using the **get\_option** method, as follows:

```
update_cmdb_request =
  $evm.vmdb(:miq_request, $evm.get_state_var(:request_id))
returned_data = update_cmdb_request.get_option(:return)
```

Executing long-running tasks asynchronously in this way using a state machine retry loop to check for completion, is an efficient way of reducing overall processing time, and increasing concurrency and throughput of automate operations.

### 7.4.3. Overcoming Default Zone Behaviour

The default behaviour of services and API requests with regard to zones may not necessarily be suitable for all cases.

#### 7.4.3.1. Services

If services are to be used to provision virtual machines, at least one CFME appliance with the Provider Operations role should be enabled in each zone.

As mentioned in [Section 7.2.1, “Zone-Related Considerations”](#), services that have a catalog item type of “Generic” might run in any zone that has a CFME appliance with the Automation Engine server role enabled. If this is not desired behaviour, a workaround is for the service catalog item provisioning entry point to run a simple method that re-launches the service provisioning state machine from a **\$evm.execute('create\_automation\_request',...)** call. This allows the target zone to be specified as the **:miq\_zone** option, for example:

```
attrs = {}
attrs['dialog_stack_name'] = $evm.root['dialog_stack_name']
attrs['dialog_password']   = $evm.root['dialog_password']
options = {}
options[:namespace]        = 'Service/Provisioning/StateMachines'
options[:class_name]       = 'ServiceProvision_Template'
options[:instance_name]    = 'create_stack'
options[:user_id]          = $evm.vmdb(:user).find_by_userid('admin').id
options[:miq_zone]         = 'Generic'
options[:attrs]            = attrs
auto_approve               = true
$evm.execute('create_automation_request', options, 'admin', ↵
  auto_approve)
```

#### 7.4.3.2. RESTful API

Automation requests submitted via RESTful API can be run in a specific zone if required. The zone name can be specified using the **:miq\_zone** parameter to the automation request, as follows:

```
:requester => {  
  :auto_approve => true  
},  
:parameters => {  
  :miq_zone => 'Zone Name'  
}
```

## CHAPTER 8. VM AND INSTANCE PROVISIONING

Although the provisioning workflows for virtual machines and instances are run by the automation engine, there are several provisioning-specific factors that should be considered when deploying CloudForms at scale.

### 8.1. STATE MACHINES






























The VM provisioning process is one of the most complex automate workflows supplied out-of-the-box with CloudForms. The workflow consists of two nested state machines, the VM provision state machine in the automate datastore, and a provider-specific *internal* state machine.

#### 8.1.1. VM Provision State Machine

The default automate datastore state machine has the fields shown in [Figure 8.1, “VM Provision State Machine”](#):

**Figure 8.1. VM Provision State Machine**

Fields

Name	Value
  CustomizeRequest	/Cloud/VM/Provisioning/StateMachines/Methods/CustomizeRequest#\$/#miq_provision.source.vendor}
  AcquireIPAddress	
  AcquireMACAddress	
  RegisterDNS	
  RegisterCMDB	
  RegisterAD	
  Placement	/Cloud/VM/Provisioning/Placement/default#\$/#miq_provision.source.vendor}
  PreProvision	/Cloud/VM/Provisioning/StateMachines/Methods/PreProvision#\$/#miq_provision.source.vendor}
  Provision	/Cloud/VM/Provisioning/StateMachines/Methods/Provision
  CheckProvisioned	/Cloud/VM/Provisioning/StateMachines/Methods/CheckProvisioned
  PostProvision	/Cloud/VM/Provisioning/StateMachines/Methods/PostProvision#\$/#miq_provision.source.vendor}
  RegisterDHCP	
  ActivateCMDB	
  EmailOwner	/Cloud/VM/Provisioning/Email/MiqProvision_Complete?event=vm_provisioned
  Finished	/System/CommonMethods/StateMachineMethods/vm_provision_finished

As can be seen, many of the fields are empty "placeholder" states such as **AcquireIPAddress** that can be used to extend the functionality of the state machine and integrate the workflow with the wider enterprise.

#### 8.1.2. Internal State Machine

The internal state machine is a nested state machine that is launched asynchronously at the **Provision** state of the VM provision state machine. The subsequent **CheckProvisioned** state of the VM provision state machine performs a check-and-retry loop until the internal state machine completes.

Internal provision state machines are provider-specific, and are not exposed to automate (they are not designed to be user-customizable). They perform the granular steps of creating the virtual machine; communicating with the EMS using its native API, and customizing the VM using the parameters defined in the provisioning options hash. A typical set of internal state machine steps to provision a VMware virtual machine are as follows:

- ✳ Determine placement
- ✳ Start VMware clone from template
- ✳ Poll for clone completion
  - When complete issue an EMS refresh on the host
- ✳ Poll the VMDB for the new object to appear
- ✳ Customize the VM
  - Reconfigure hardware if necessary
- ✳ Autostart the VM
- ✳ Run post-create tasks
  - Set description
  - Set ownership
  - Set retirement
  - Set genealogy
  - Set miq\_custom\_attributes
  - Set ems\_custom\_attributes
  - Connect to service
- ✳ Mark as completed
- ✳ Finish

The final state of the internal state machine marks the provision task object as having a state of *provisioned*. The outer VM provision state machine **CheckProvisioned** state polls for this status, and continues to its own **PostProvision** state when detected.

## 8.2. CHALLENGES OF SCALE

The VM or instance provisioning workflow contains several operations that are external to CloudForms, but contribute to overall provisioning time.

- ✳ Interactions with the external management system
  - EMS API calls from the internal state machine - cloning the template or adding a disk for example
  - EMS refresh to retrieve details of the new VM
- ✳ Interactions with and time consumed by external provisioning components such as PXE/Kickstart servers

- ✳ Interactions with other enterprise systems such as Active Directory, IPAM or a CMDB
- ✳ Post-provisioning time consumed by initialization scripts such as cloud-init or sysprep (particularly where this includes a software update of the new virtual machine)

With larger enterprises the number of interactions - and inherent workflow delays - often increases, and CloudForms sometimes needs tuning to cater for this.

### 8.2.1. State Machine Timeouts

As mentioned in [Chapter 7, Automate](#), the message to initiate a VM provisioning workflow has a timeout value of 600 seconds. The VM provision state machine therefore has a maximum time of 10 minutes to execute down to the first retry stage, which is **CheckProvisioned**.





































#### 8.2.1.1. External Integration

In larger CloudForms deployments it is common to add enterprise integration to the VM provisioning workflow. Custom instances are often added to the placeholder fields such as **AcquireIPAddress** to retrieve an IP address from a corporate IP Address Management (IPAM) solution, for example. If the methods run by these stages take minutes to run under high load, the state machine may timeout before the **CheckProvisioned** state is reached.

To reduce this possibility the VM provision state machine can be expanded to include check-and-retry states after the custom methods, such as the **CheckIPAddressAcquired** state in [Figure 8.2, “Modified VM Provision State Machine”](#).

**Figure 8.2. Modified VM Provision State Machine**

Fields

Name	Value
  CustomizeRequest	/Infrastructure/VM/Provisioning/StateMachines/Methods/CustomizeRequest#\$/#miq_provision.source.vendor}
  AcquireIPAddress	/Integration/IPAM/Methods/GetIPAddress
  CheckIPAddressAcquired	/Integration/IPAM/Methods/WaitForIPAddress
  AcquireMACAddress	
  RegisterDNS	
  RegisterCMDB	
  RegisterAD	
  Placement	/Infrastructure/VM/Provisioning/Placement/default#\$/#miq_provision.source.vendor}
  PreProvision	/Infrastructure/VM/Provisioning/StateMachines/Methods/PreProvision#\$/#miq_provision.source.vendor}
  Provision	/Infrastructure/VM/Provisioning/StateMachines/Methods/Provision
  CheckProvisioned	/Infrastructure/VM/Provisioning/StateMachines/Methods/CheckProvisioned
  PostProvision	/Infrastructure/VM/Provisioning/StateMachines/Methods/PostProvision#\$/#miq_provision.source.vendor}
  WalkObjects	
  RegisterDHCP	
  ActivateCMDB	
  RegisterSatellite	/Integration/Satellite/AnsibleMethods/register_satellite
  EmailOwner	/Infrastructure/VM/Provisioning/Email/MiqProvision_Complete?event=vm_provisioned
  Finished	/System/CommonMethods/StateMachineMethods/vm_provision_finished

#### 8.2.1.2. Placement

The `/Infrastructure/VM/Provisioning/Placement` namespace in the *RedHat* automate domain includes 3 additional placement methods:

- ✎ `redhat_best_placement_with_scope`
- ✎ `vmware_best_fit_with_scope`
- ✎ `vmware_best_fit_with_tags`

These methods perform additional processing to search for an optimum cluster, host and datastore on which to place the new VM, based on tags or criteria such as most free space, or lowest current CPU utilization. With a large virtual infrastructure containing many hosts and datastores, the real-time checking of these placement permutations can take a long time, and occasionally cause the state machine to timeout.

The placement methods are designed to be user-editable so that alternative criteria can be selected. If the placement methods are taking too long they may need to be edited to simplify the placement criteria.

### 8.2.1.3. CheckProvisioned

The **CheckProvisioned** state of the VM provision state machine executes a check-and-retry loop until the provisioning task object shows a **state** of 'provisioned' or 'error'. At this point the newly provisioned VM is powered on, and is represented by an object in the CloudForms VMDB. The maximum retries for the **CheckProvisioned** state is set at 100, and the default retry interval (set in the `check_provisioned` method) is as follows:

```
$evm.root['ae_retry_interval'] = '1.minute'
```

When managing very large cloud environments or virtual infrastructures under high load, it can sometimes take longer than 100 minutes for the provisioning steps, related event handling, and EMS refresh to complete. Delays can be caused by many factors, including the following:

- ✎ Many other automation messages are queued at the same priority ahead of the provider message for the VM create event
- ✎ The message queue is filled with event messages from a provider in the region that is experiencing an event storm
- ✎ A prior full refresh is still active
- ✎ The provider does not support targeted refresh

The effect of such delays can be minimized by increasing the number of retries in the VM provision state machine for the **CheckProvisioned** state, or by editing the `check_provisioned` method to increase the retry interval.

## 8.3. TUNING PROVISIONING

As can be seen, many of the provisioning related problems of scale are related to external factors. Although some fine tuning of timeouts and method optimization can be performed, reliability cannot necessarily be improved by scaling out CloudForms (for example adding CFME appliances, or increasing worker counts).

### 8.3.1. Incubation Region

It can sometimes be beneficial in large virtual environments to **create a separate provisioning or incubation CloudForms region** that manages a small sub-set of the overall infrastructure. This can be used to provision new virtual machines, which can then be migrated to the production data centers or clusters once they are patched and ready for use.

## CHAPTER 9. EVENT HANDLING

The timely processing of external and internal events is important to the overall smooth running of a CloudForms installation. This section discusses the event handling process and how it can be tuned for scale.

### 9.1. EVENT PROCESSING WORKFLOW

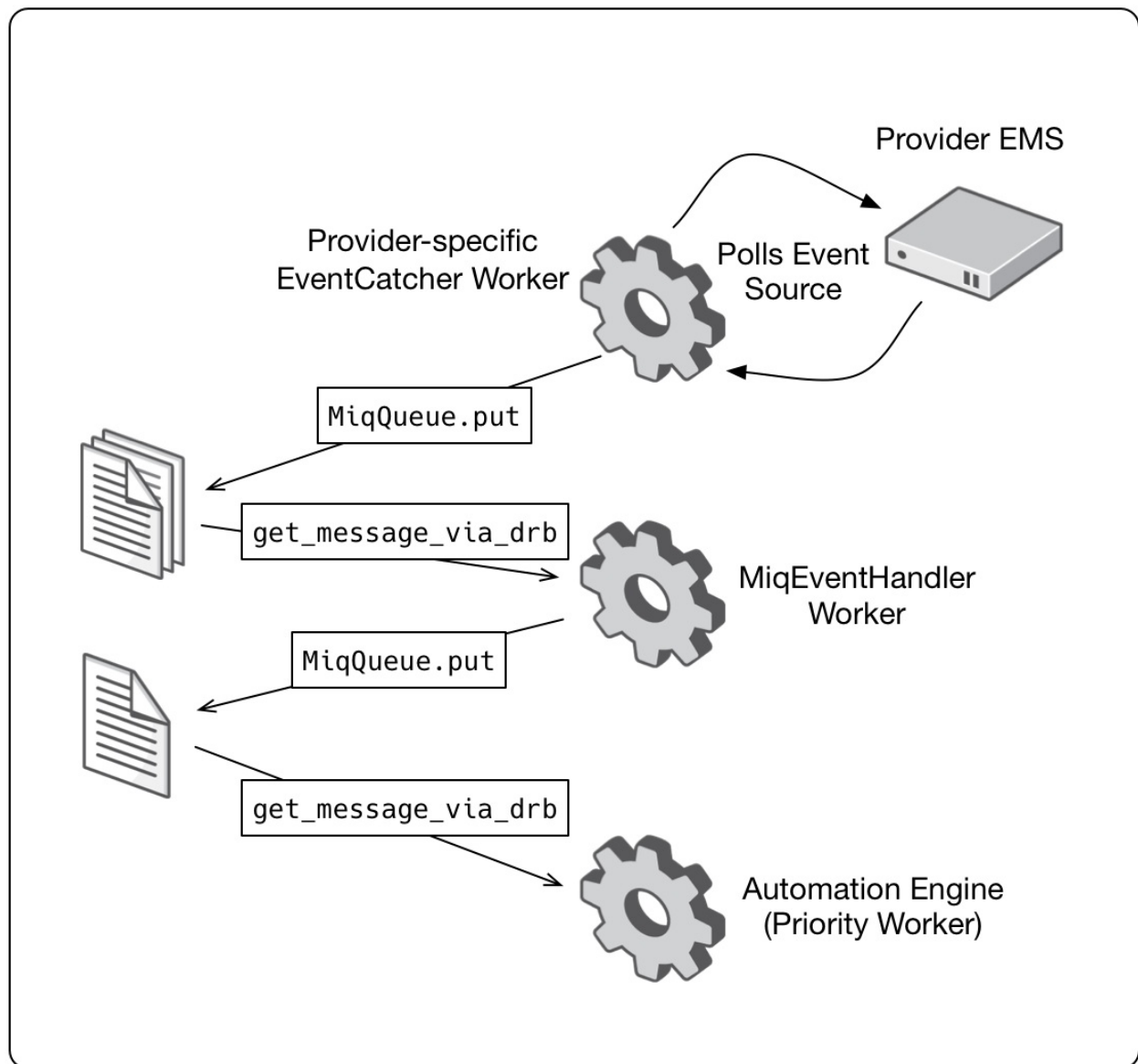
The event processing workflow involves 3 different workers, as follows:

1. A provider-specific event catcher polls the EMS event source for new events using an API call such as <https://rhevm/api/events?from=54316> (see [Section 9.1.1, “Event Catcher Polling Frequency”](#) for the frequency of this polling). For each new event caught a message is queued for the event handler
2. The generic *MiqEventHandler* worker dequeues the message, and creates an *EmsEvent* EventStream object. Any EMS-specific references such as `:vm⇒{:id⇒"4e7b66b7-080d-4593-b670-3d6259e47a0f"}` are translated into the equivalent CloudForms object ID such as `"VmOrTemplate::vm"⇒10000000000023`, and a new high priority message is queued for automate
3. A Priority worker dequeues the message and processes it through the automate event switchboard using the EventStream object created by the *MiqEventHandler*. Processing the event may involve several *event handler* automate instances that perform actions such as:
  - ✳ Process any control policies associated with the event
  - ✳ Process any alarms associated with the event
  - ✳ Initiate any further operations that are required after the event, such as triggering an EMS refresh

The event workflow is illustrated in [Figure 9.1, “Event Processing Workflow”](#)

**Figure 9.1. Event Processing Workflow**





### 9.1.1. Event Catcher Polling Frequency

The polling frequency of each of the provider-specific event catchers is defined in the **:event\_catcher** section of the **Configuration → Advanced** settings. The default settings for CloudForms Management Engine 5.8 are as follows:

```

:event_catcher:
  :poll: 1.seconds
:event_catcher_ansible_tower:
  :poll: 20.seconds
:event_catcher_embedded_ansible:
  :poll: 20.seconds
:event_catcher_redhat:
  :poll: 15.seconds
:event_catcher_openstack:
  :poll: 15.seconds
:event_catcher_openstack_infra:
  :poll: 15.seconds
:event_catcher_openstack_network:
  :poll: 15.seconds

```

```

:event_catcher_hawkular:
  :poll: 10.seconds
:event_catcher_hawkular_datawarehouse:
  :poll: 1.minute
:event_catcher_google:
  :poll: 15.seconds
:event_catcher_kubernetes:
  :poll: 1.seconds
:event_catcher_lenovo:
  :poll: 4.minutes
:event_catcher_openshift:
  :poll: 1.seconds
:event_catcher_cinder:
  :poll: 10.seconds
:event_catcher_swift:
  :poll: 10.seconds
:event_catcher_amazon:
  :poll: 15.seconds
:event_catcher_azure:
  :poll: 15.seconds
:event_catcher_vmware:
  :poll: 1.seconds
:event_catcher_vmware_cloud:
  :poll: 15.seconds

```

## 9.2. GENERIC EVENTS

Some external management systems implement generic event types that are issued under a variety of conditions. They are often used by third-party software vendors as a means to add their own specific events to those of the native EMS. Generic events often have a sub-type associated with them to indicate a more specific event source.

### 9.2.1. EventEx

VMware vCenter management systems use an event type called *EventEx* as a catch-all event. Several VMware components issue EventEx events with a subtype to record state changes, problems, and recovery from problems. They appear as **[EventEx] - [subtype]**, for example:

- ✱ [EventEx]-[com.vmware.vc.VmDiskConsolidatedEvent]
- ✱ [EventEx]-[com.vmware.vim.eam.task.scanForUnknownAgentVmsCompleted]
- ✱ [EventEx]-[com.vmware.vim.eam.task.scanForUnknownAgentVmsInitiated]
- ✱ [EventEx]-[esx.problem.scsi.device.io.latency.high]
- ✱ [EventEx]-[esx.problem.vmfs.heartbeat.recovered]
- ✱ [EventEx]-[esx.problem.vmfs.heartbeat.timedout]
- ✱ [EventEx]-[vprob.storage.connectivity.lost]
- ✱ [EventEx]-[vprob.vmfs.heartbeat.recovered]
- ✱ [EventEx]-[vprob.vmfs.heartbeat.timedout]

## 9.3. EVENT STORMS

Event storms are very large bursts of events emitted by a provider's EMS. They can be caused by several types of warning or failure condition, including storage or adapter problems, or host capacity, swap space usage or other host thresholds being crossed. When a component is failing intermittently the storm is often made worse by events indicating the transition between problem and non-problem state, for example:

```
[----] I, [2017-01-25T03:23:04.998138 #374:66b14c] ... caught event ↵
[EventEx]-[esx.clear.scsi.device.io.latency.improved] chainId [427657]
[----] I, [2017-01-25T03:23:04.998233 #374:66b14c] ... caught event ↵
[EventEx]-[esx.problem.scsi.device.io.latency.high] chainId [427658]
[----] I, [2017-01-25T03:23:04.998289 #374:66b14c] ... caught event ↵
[EventEx]-[esx.clear.scsi.device.io.latency.improved] chainId [427659]
[----] I, [2017-01-25T03:23:04.998340 #374:66b14c] ... caught event ↵
[EventEx]-[esx.clear.scsi.device.io.latency.improved] chainId [427660]
[----] I, [2017-01-25T03:23:04.998389 #374:66b14c] ... caught event ↵
[EventEx]-[esx.problem.scsi.device.io.latency.high] chainId [427661]
[----] I, [2017-01-25T03:23:04.998435 #374:66b14c] ... caught event ↵
[EventEx]-[esx.problem.scsi.device.io.latency.high] chainId [427662]
[----] I, [2017-01-25T03:23:04.998482 #374:66b14c] ... caught event ↵
[EventEx]-[esx.clear.scsi.device.io.latency.improved] chainId [427663]
[----] I, [2017-01-25T03:23:04.998542 #374:66b14c] ... caught event ↵
[EventEx]-[esx.clear.scsi.device.io.latency.improved] chainId [427664]
```



### Note

The log snippet above is from a production CloudForms installation. Note that many events are received within the same millisecond - typical of an event storm

Event storms are highly detrimental to the overall performance of a CloudForms region for many reasons, including the following:

- ✦ All *MiqEventHandler* workers in a zone can be overwhelmed processing messages from one provider, to the detriment of other providers in that zone
- ✦ The many hundreds of thousands (up to tens of millions) of unprocessed high-priority messages in the `miq_queue` table consume all Generic and Priority workers in the zone
- ✦ The number of messages in the `miq_queue` table affects the performance of `get_message_via_drb` for all queue workers in the entire region

In some cases the problems are temporary and clear themselves after the event message emission stops and the CFME appliances can process the messages already queued for processing. In other cases the sheer volume of event messages can result in appliances which still appear to be running, but where the CFME services - including the WebUI - are unresponsive.

### 9.3.1. Handling and Recovering from Event Storms

Until the cause of the event storm is identified and corrected, the quickest way to restore any operation for the CloudForms environment is to prevent the continued growth of the `miq_queue` table. The simplest techniques are to blacklist the event(s) causing the storm (see Section 9.4.1, "Blacklisting Events"), or to disable the event monitor role on all CFME appliance in the provider's zone.



## Note

Disabling the event monitor will disable both the event catcher and event processor workers, so queued messages in the `miq_queue` table will not be processed. If there are multiple providers in the zone, event catching and handling for these providers may also become inactive.

In critical situations with many hundreds of thousands to millions of queued messages, it may be necessary to selectively delete message instances from the `miq_queue` table. Since the overwhelming number of messages expected to be in this table will be of type 'event', the following SQL statement can be used to remove all such instances from the `miq_queue` table:

```
delete from miq_queue where role = 'event' and class_name = 'EmsEvent';
```

Before running this query the following points should be noted:

- ✳ The only response from this query is a count of the number of messages removed
- ✳ The query only deletes messages where the role is 'event' and should not touch any other messages that have been queued
- ✳ Even though one single specific event may be responsible for 99+% of the instances, any non-problem event messages will also be deleted.

## 9.4. TUNING EVENT HANDLING

There are several measures that can be taken to tune event handling for scale, including filtering the events that are to be processed or ignored.

### 9.4.1. Blacklisting Events

Some provider events occur relatively frequently, but are either uninteresting to CloudForms, or processing them would consume excessive resources (such as those typically associated with event storms). Events such as these can be skipped or *blacklisted*. The event catchers write a list of blacklisted events to `evm.log` when they start, for example:

```
... MIQ(ManageIQ::Providers::Redhat::InfraManager::EventCatcher::
Runner#after_initialize) EMS [rhevm.bit63.net] as [cfme@internal]
Event Catcher skipping the following events:
... INFO -- : - UNASSIGNED
... INFO -- : - USER_REMOVE_VG
... INFO -- : - USER_REMOVE_VG_FAILED
... INFO -- : - USER_VDC_LOGIN
... INFO -- : - USER_VDC_LOGIN_FAILED
... INFO -- : - USER_VDC_LOGOUT
```

These events are defined in the `blacklisted_events` table in the VMDB. The default rows in the table are as follows:

```
vmdb_production=# select event_name,provider_model
from blacklisted_events;
          event_name          | provider_model
-----+-----
```

```
--
storageAccounts_listKeys_BeginRequest | ...Azure::CloudManager
storageAccounts_listKeys_EndRequest | ...Azure::CloudManager
identity.authenticate | ...Openstack::CloudManager
scheduler.run_instance.start | ...Openstack::CloudManager
scheduler.run_instance.scheduled | ...Openstack::CloudManager
scheduler.run_instance.end | ...Openstack::CloudManager
ConfigurationSnapshotDeliveryCompleted | ...Amazon::CloudManager
ConfigurationSnapshotDeliveryStarted | ...Amazon::CloudManager
ConfigurationSnapshotDeliveryFailed | ...Amazon::CloudManager
UNASSIGNED | ...Redhat::InfraManager
USER_REMOVE_VG | ...Redhat::InfraManager
USER_REMOVE_VG_FAILED | ...Redhat::InfraManager
USER_VDC_LOGIN | ...Redhat::InfraManager
USER_VDC_LOGOUT | ...Redhat::InfraManager
USER_VDC_LOGIN_FAILED | ...Redhat::InfraManager
AlarmActionTriggeredEvent | ...Vmware::InfraManager
AlarmCreatedEvent | ...Vmware::InfraManager
AlarmEmailCompletedEvent | ...Vmware::InfraManager
AlarmEmailFailedEvent | ...Vmware::InfraManager
AlarmReconfiguredEvent | ...Vmware::InfraManager
AlarmRemovedEvent | ...Vmware::InfraManager
AlarmScriptCompleteEvent | ...Vmware::InfraManager
AlarmScriptFailedEvent | ...Vmware::InfraManager
AlarmSnmpCompletedEvent | ...Vmware::InfraManager
AlarmSnmpFailedEvent | ...Vmware::InfraManager
AlarmStatusChangedEvent | ...Vmware::InfraManager
AlreadyAuthenticatedSessionEvent | ...Vmware::InfraManager
EventEx | ...Vmware::InfraManager
UserLoginSessionEvent | ...Vmware::InfraManager
UserLogoutSessionEvent | ...Vmware::InfraManager
identity.authenticate | ...Openstack::InfraManager
scheduler.run_instance.start | ...Openstack::NetworkManager
scheduler.run_instance.scheduled | ...Openstack::NetworkManager
scheduler.run_instance.end | ...Openstack::NetworkManager
ConfigurationSnapshotDeliveryCompleted | ...Amazon::NetworkManager
ConfigurationSnapshotDeliveryStarted | ...Amazon::NetworkManager
ConfigurationSnapshotDeliveryFailed | ...Amazon::NetworkManager
(37 rows)
```

If processing of any of the events in the `blacklisted_events` table *is* required, the *enabled* field can be set to false and the provider-specific event catcher restarted.

An EMS can also report some minor object property changes as events, even though these are not modelled in the CloudForms VMDB. For VMware providers such event types can be added to the "Vim Broker Exclude List" so that they can be discarded without processing. The exclude list is found under **:broker\_notify\_properties** in the **Configuration → Advanced** settings, as follows:

```
:broker_notify_properties:
:exclude:
:HostSystem:
- config.consoleReservation
- config.dateTimeInfo
- config.network
- config.service
```

```

- summary
- summary.overallStatus
- summary.runtime.bootTime
- summary.runtime.healthSystemRuntime.systemHealthInfo. ↵
  numericSensorInfo
:VirtualMachine:
- config.locationId
- config.memoryAllocation.overheadLimit
- config.npivWorldWideNameType
- guest.disk
- guest.guestFamily
- guest.guestFullName
- guest.guestId
- guest.ipStack
- guest.net
- guest.screen
- guest.screen.height
- guest.screen.width
- guest.toolsRunningStatus
- guest.toolsStatus
- resourceConfig
- summary
- summary.guest.guestFullName
- summary.guest.guestId
- summary.guest.toolsRunningStatus
- summary.overallStatus
- summary.runtime.bootTime
- summary.runtime.memoryOverhead
- summary.runtime.numMksConnections
- summary.storage
- summary.storage.committed
- summary.storage.unshared

```

### 9.4.2. Flood Monitoring

CloudForms recently introduced the concept of flood monitoring for the provider-specific event catchers. This stops provider events from being queued when too many duplicates are received in a short time. **By default an event is considered as flooding if it is received 30 times in one minute.**

Flood monitoring is a generic concept for event processing, but requires the appropriate supporting methods to be added to each provider. **As of CloudForms Management Engine 5.8 only the VMware provider supports this functionality.**

### 9.4.3. Event Catcher Configuration

The **:event\_catcher** section is one of the largest of the **Configuration → Advanced** settings, and it defines the configuration of each type of event catcher. For example the following extract shows the settings for the *ManageIQ::Providers::Openstack::InfraManager::EventCatcher* worker:

```

:event_catcher:
...
:event_catcher_openstack:
  :poll: 15.seconds
  :topics:
    :nova: notifications.*

```

```

        :cinder: notifications.*
        :glance: notifications.*
        :heat: notifications.*
:duration: 10.seconds
:capacity: 50
:amqp_port: 5672
:amqp_heartbeat: 30
:amqp_recovery_attempts: 4
:ceilometer:
    :event_types_regex: "\\A(?!firewall|floatingip|gateway|
net|port|router|subnet|security_group|vpn)"
...

```

The configuration settings rarely need to be changed from their defaults.

## 9.5. SCALING OUT

The event processing workflow can be quite resource-intensive. CloudForms installations managing several thousand objects may benefit from dedicated CFME appliances exclusively running the provider-specific *EventCatcher* workers and *MiqEventHandler* worker in any zone containing providers.

## CHAPTER 10. SMARTSTATE ANALYSIS

SmartState Analysis allows CloudForms to perform a deep introspection of virtual machines, containers and hosts to discover their contents. The technology is agentless, and does not require virtual machines to be powered on.



### Note

SmartState Analysis is alternatively known as "fleecing"

SmartState Analysis uses two server roles. The first - SmartState Analysis - is performed by a Generic or Priority worker, depending on message priority. The second server role - SmartProxy - enables the embedded or *coresident*.<sup>[18]</sup> *MiqSmartProxyWorker* processes. These workers perform the following sequence of steps to scan each virtual machine:

- ✦ Create a snapshot of the VM
- ✦ Analyze the snapshot:
  - Mount the VM's disks
  - Analyze the content
  - Unmount the VM's disks
- ✦ Remove the snapshot
- ✦ Upload the metadata to temporary storage in the VMDB

The SmartState Analysis role calls a component named the *JobProxyDispatcher* to determine the most suitable SmartProxy server to run the VM scan. Once the scan has completed the SmartState Analysis worker saves the scan metadata to the VM's model, and creates drift history data by comparing the most recent scan with previous results.

### 10.1. PROVIDER-SPECIFIC CONSIDERATIONS

There are several provider-specific considerations to be aware of when configuring SmartState Analysis.

#### 10.1.1. VMware

The *MiqSmartProxyWorker* processes scan VMware virtual machines using the VixDiskLib API functionality provided by the VMware Virtual Disk Development Kit (VDDK). Any CFME appliance in the provider's zone that is running the SmartProxy role must therefore have the VDDK installed.<sup>[19]</sup>

##### 10.1.1.1. Authentication

The VDDK requires an authenticated connection to be made to the ESXi host running the VM. For the authentication to succeed the credentials for each ESXi hypervisor must be defined against the host properties in the CloudForms WebUI. The credentials should use either *root*, or a VMware account with the following role permissions:

- ✦ Datastore



- Browse Datastore
- Low level file operations
- ✱ Global
  - Diagnostics
  - Licenses
- ✱ Host
  - Configuration
    - Advanced Settings
- ✱ Virtual Machine
  - Provisioning
    - Allow read-only disk access
  - Snapshot Management
    - Create snapshot
    - Remove snapshot

#### 10.1.1.1.1. Authentication via vCenter

If it is not possible to add credentials for the ESXi hosts, virtual machine scanning can still be performed using an authentication token provided by the vCenter.

The CloudForms **Configuration** → **Advanced** settings contain a section entitled **:coresident\_miqproxy** that has a value **:scan\_via\_host**. By default this is set to **true**, but changing the value to **false** and restarting the *MiqSmartProxyWorker* processes enables vCenter authentication for VM scans.



#### Note

The name **:scan\_via\_host** is slightly misleading. Setting this value to **false** only enables VDDK authentication via the vCenter. The actual scan is still performed by the SmartProxy server connecting directly to the ESXi host using port 902.

```
:coresident_miqproxy:
...
:scan_via_host: false
```

#### 10.1.2. Red Hat Virtualization

For SmartState Analysis of Red Hat Virtualization (RHV) virtual machines to complete successfully, the CFME appliances running the SmartProxy server roles must be in the same RHV datacenter as the VM being scanned. The storage domains must also be accessible to the SmartProxy appliances. Fibre channel or iSCSI storage domains should be presented to each SmartProxy appliance as shareable direct LUNs. NFS datastores must be mountable by each SmartProxy appliance, which may mean adding secondary network interfaces to the CFME appliances,

connected to the storage network.

The *management engine relationship* must also be set for each CFME appliance. This enables the VM SmartState Analysis job to determine the datacenter where the CFME appliance is running and thus to identify which storage it has access to.

### 10.1.3. OpenStack

CloudForms is capable of performing a SmartState Analysis of both Overcloud images, and Undercloud Nova compute nodes.

#### 10.1.3.1. Overcloud

CloudForms is able to perform a SmartState Analysis of Glance-backed OpenStack images. In order to scan a running instance, an image snapshot is taken and copied to the CFME appliance to be scanned (SmartState Analysis requires byte-level offset/length access to images which cannot be performed remotely using the current OpenStack APIs).

To ensure that this storage area is large enough to receive large image snapshots, any CFME appliance in an OpenStack zone with the SmartProxy role enabled should have its temporary storage area extended using the following **appliance\_console** option:

##### 10) Extend Temporary Storage

This option will format an unpartitioned disk attached to the CFME appliance and mount it as `/var/www/miq_tmp`

#### 10.1.3.2. OpenStack Platform Director (Undercloud)

CloudForms is able to perform a SmartState Analysis of OpenStack Platform Director Nova compute nodes. To allow the smart proxy to connect to the Nova hosts, the RSA key pair private key for the hosts should be added to the provider details. The heat-admin user is typically used for host connection.

### 10.1.4. OpenShift

SmartState scanning of OpenShift containers is performed by an *image\_inspector* pod that is pulled from the Red Hat registry as required. The image inspector dynamically downloads and uses the latest OpenScap definition/rules file from Red Hat before scanning.<sup>[20]</sup>

With CloudForms 4.5 the registry and repository are configurable in **Configuration → Advanced** settings, as follows:

```
:ems_kubernetes:
...
:image_inspector_registry: registry.access.redhat.com
:image_inspector_repository: openshift3/image-inspector
```

## 10.2. MONITORING SMARTSTATE ANALYSIS

The total time for each VM scan can be determined from the time duration between the "request\_vm\_scan" and corresponding "vm\_scan\_complete" events being processed through automate, as follows:

```

... INFO -- : MIQ(MiqAeEngine.deliver) Delivering ↵
{:event_type=>"request_vm_scan", "VmOrTemplate::vm"=>39, :vm_id=>39, ↵
:host=>nil, "MiqEvent::miq_event"=>20690, :miq_event_id=>20690, ↵
"EventStream::event_stream"=>20690, :event_stream_id=>20690} ↵
for object [ManageIQ::Providers::Redhat::InfraManager::Vm.39] ↵
with state [] to Automate

...

... INFO -- : MIQ(MiqAeEngine.deliver) Delivering ↵
{:event_type=>"vm_scan_complete", "VmOrTemplate::vm"=>39, :vm_id=>39, ↵
:host=>nil, "MiqEvent::miq_event"=>20692, :miq_event_id=>20692, ↵
"EventStream::event_stream"=>20692, :event_stream_id=>20692} ↵
for object [ManageIQ::Providers::Redhat::InfraManager::Vm.39] ↵
with state [] to Automate

```

This time includes the scan pre-processing by the Generic worker, the handoff by the *JobProxyDispatcher* to the appropriate SmartProxy appliance, and the subsequent scan and data process and upload times.

More granular timings are logged to *evm.log* and these can be examined if required to determine the source of bottlenecks. For example the time taken for the *MiqSmartProxyWorker* process to extract each part of the profile is logged, and can be extracted using the following bash command:

```
grep 'information ran for' evm.log
```

```

... Scanning [vmconfig] information ran for [0.156029053] seconds.
... Scanning [accounts] information ran for [0.139248768] seconds.
... Scanning [software] information ran for [4.357743037] seconds.
... Scanning [services] information ran for [3.767868137] seconds.
... Scanning [system] information ran for [0.305050798] seconds.
... Scanning [profiles] information ran for [0.003027426] seconds.

```

## 10.3. CHALLENGES OF SCALE

SmartState Analysis is a relatively time-consuming operation per virtual machine. Many of the problems associated with scaling SmartState Analysis are related to performing many hundreds or thousands of analyses in a limited time window.

Periodic scans of a complete VM inventory should be scheduled with a frequency that allows each scan to complete before the next is scheduled. For small installations this is sometimes daily, but larger scale installations often schedule these on a weekly or monthly basis. Control policies can be used to perform initial scans when VMs are first provisioned, so that SmartState data is available for new VMs before a scheduled analysis has been run.

### 10.3.1. Virtual Machines Running Stateful Applications

A virtual machine SmartState Analysis is always performed on a temporary snapshot of the VM. The snapshot is taken using the native means exposed by the EMS, however most snapshotting technology does not take into account the requirements of any application running in the virtual machine. **Taking a virtual machine snapshot can have unintended and unexpected consequences for some applications that maintain state data such as Microsoft Exchange Server.**<sup>[21]</sup>

Virtual machines running such applications must not be snapshotted, and should therefore be

excluded from SmartState Analysis.

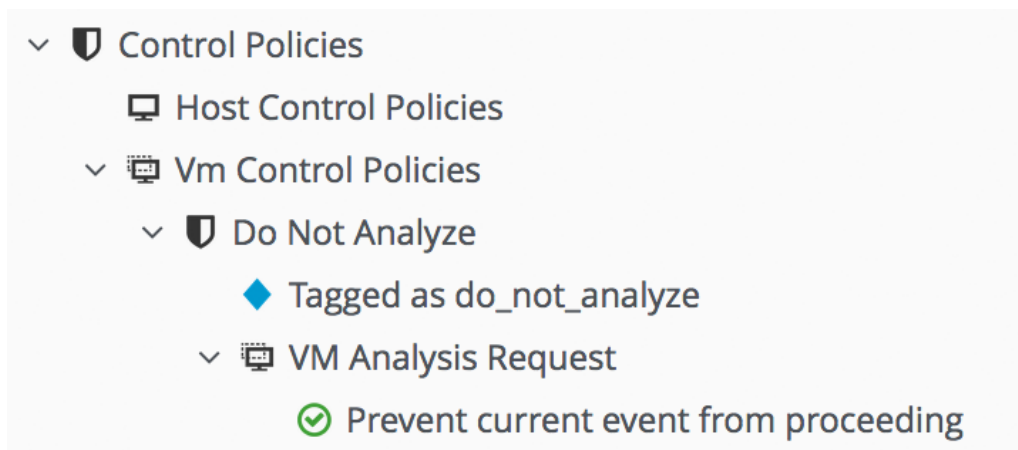


#### Note

A SmartState Analysis of the CloudForms VMDB appliance should never be performed

A control policy can be created to prevent SmartState Analysis from running on any VM tagged with "exclusions/do\_not\_analyze", as shown in [Figure 10.1, "Control Policy to Block SmartState Analysis"](#).

**Figure 10.1. Control Policy to Block SmartState Analysis**



Virtual machines running stateful workloads can be tagged accordingly to prevent the snapshot from being taken.

### 10.3.2. Identifying SmartState Analysis Problems

Problems with SmartState Analysis are logged to *evm.log*, and can be identified using the following bash command:

```
grep 'VmScan#process_abort' evm.log
```

Many of the most common errors are caused as a result of scaling parts of the infrastructure - hosts or CFME appliances - and forgetting to update the provider-specific considerations for SmartState Analysis.

#### 10.3.2.1. No active SmartProxies found

If the *JobProxyDispatcher* cannot find a suitable SmartProxy to scan a virtual machine, the error "No active SmartProxies found to analyze this VM" is logged. In VMware environments this is often caused by failing to install the VDDK on a new CFME appliance that has been configured with the SmartProxy server role.

```
... MIQ(VmScan#process_abort) job aborting, No eligible proxies for VM
:[NFS_PROD] odrsrv001/odrsrv001.vmx] - [No active SmartProxies found
to analyze this VM], aborting job [8064001a-e2ea-11e6-9140-005056b19b0f].
```

### 10.3.2.2. Provide credentials

If a new VMware ESXi hosts's credentials have been omitted from the CloudForms WebUI (or a host's credentials changed), the error "Provide credentials for this VM's Host to perform SmartState Analysis" will be logged if a scan is attempted of a virtual machine running on that host.

```
... MIQ(VmScan#process_abort) job aborting, No eligible proxies for VM
:[[[FCP_MID] osdweb01/osdweb01.vmx] - [Provide credentials for this VM's
Host to perform SmartState Analysis], aborting job
[d2e08e70-c26b-11e6-aaa4-00505695be62].
```

### 10.3.2.3. Unable to mount filesystem

If a CFME appliance running the SmartProxy server role does not have access to the storage network of a RHV provider, an attempted scan of a virtual machine on an NFS storage domain will timeout.

```
... MIQ(VmScan#process_abort) job aborting, Unable to mount filesystem.
Reason:[mount.nfs: Connection timed out
```

## 10.4. TUNING SMARTSTATE ANALYSIS

SmartState Analysis settings are stored in the **:coresident\_miqproxy** section of the **Configuration → Advanced settings**, as follows:

```
:coresident_miqproxy:
  :concurrent_per_ems: 1
  :concurrent_per_host: 1
  :scan_via_host: true
  :use_vim_broker: true
  :use_vim_broker_ems: true
```

The default value of **:concurrent\_per\_host** is 1, which limits the number of concurrent VM scans that can be carried out to any particular host. This can be increased - with caution - to allow several scans to run concurrently.

### 10.4.1. Increasing the Number of SmartProxy Workers

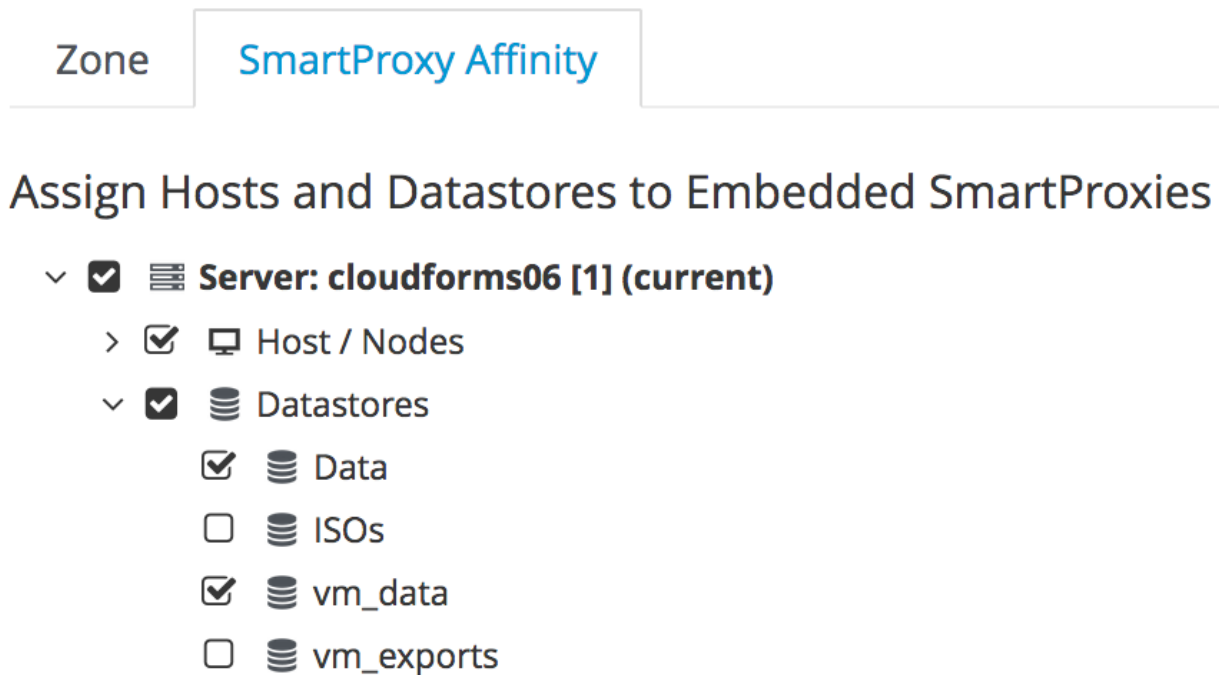
The default number of "VM Analysis Collector" (*MiqSmartProxyWorker*) workers per appliance is 3. This can be increased to a maximum of 5, although consideration should be given to the additional CPU and memory requirements that an increased number of workers will place on an appliance. It may be more appropriate to add further appliances and scale horizontally.

CloudForms installations managing several thousand objects may benefit from dedicated CFME appliances in the provider zones exclusively running the SmartState Analysis and SmartProxy roles.

### 10.4.2. SmartProxy Affinity

Hosts and datastores can be 'pinned' to specific embedded SmartProxy servers using the **SmartProxy Affinity** setting in the **Configuration → Settings → Zones** area of the WebUI, as shown in [Figure 10.2, "SmartProxy Affinity"](#):

Figure 10.2. SmartProxy Affinity



This can help ensure that only the most optimally placed or suitably configured CFME appliances are used for SmartState Analysis scans.

[18] Earlier versions of CloudForms and ManageIQ supported *external* Smart Proxies running on Windows servers or VMware ESX hosts. These are no longer required and so have been removed from the product

[19] The procedure to install the VDDK is described in the following Red Hat Knowledge Base article: <https://access.redhat.com/articles/2078103>

[20] Enabling proxy access for the openshift3/image-inspector is described in the following Red Hat Knowledge Base article: <https://access.redhat.com/solutions/2915411>

[21] Further information can be found in the following Microsoft Technet article: [https://technet.microsoft.com/en-us/library/jj126252%28v=exchg.141%29.aspx?f=255&MSPPErr=-2147217396#BKMK\\_ExchangeStor](https://technet.microsoft.com/en-us/library/jj126252%28v=exchg.141%29.aspx?f=255&MSPPErr=-2147217396#BKMK_ExchangeStor)

## CHAPTER 11. WEB USER INTERFACE

Scaling a CloudForms installation usually implies that many users will be accessing the WebUI components. It is therefore prudent to scale the WebUI capability along with the CFME infrastructure components and workers to ensure that responsiveness and connection reliability are maintained.

The "Operations" or "Classic" WebUI (as opposed to the Self-Service UI) uses an Apache web server as a reverse proxy front-end to a Puma application server. Each instance of a *MiqUiWorker* worker is a Puma process.

### 11.1. SCALING WORKERS

Most but not all of the UI transactions are written to be asynchronous, but a very few of them are either still synchronous or perform other processing. This can sometimes cause the *MiqUiWorker* process to appear unresponsive to other user sessions. An example of this can be seen when executing a long-running automate task from simulation in one browser window. Other browser sessions connected to the same *MiqUiWorker* process may appear hung until the simulation has completed.

A solution to this is to increase the number of WebUI workers. The default number of UI workers per CFME appliance is 1, but this can be increased to a maximum of 9, although consideration should be given to additional CPU and memory requirements that an increased number of workers will place on an appliance (the maximum memory threshold for a UI worker is 1GByte).

#### Tip

WebUI transactions initiated by each *MiqUiWorker* process are written into the *production.log* file. This is often a useful source of information when troubleshooting WebUI problems.

### 11.2. SCALING APPLIANCES

To allow for a degree of fault-tolerance in a large CloudForms installation, it is common to deploy several dedicated WebUI CFME appliances in their own zone for general user session use. Each of the CFME appliances should be configured with a minimal set of server roles, for example:

- ✎ Automation Engine (to process zone events)
- ✎ Provider Operations (if VM provisioning services are used)
- ✎ Reporting (if logged-on users will be running their own reports)
- ✎ User Interface
- ✎ Web Services
- ✎ Websocket

#### 11.2.1. Load Balancers

Multiple CFME appliances in a WebUI zone are often placed behind a load balancer. The load balancer should be configured with sticky sessions enabled, which will force it to send requests to the same UI worker during a session.

The load balancer should also be configured to test for connectivity using the CloudForms ping response page at [https://cfme\\_appliance/ping](https://cfme_appliance/ping). The expected reply from the appliance is the text string “pong”. Using this URL is preferable to the standard login URL as it does not establish a connection to the database.

By default the CloudForms UI workers store session data in the local appliance’s memcache. When operating behind a load balancer the UI workers should be configured to store session data in the database. This prevents a user from having to re-login if the load balancer redirects them to an alternative server if their original UI worker is unresponsive.

The location of the session store is defined in the **Configuration** → **Advanced** settings. The default value for **session\_store** is as follows:

```
:server:
...
:session_store: cache
```

This should be changed to:

```
:session_store: sql
```



## CHAPTER 12. MONITORING

Monitoring of the various components described in this document is essential for maintaining optimum performance of a large CloudForms installation.

As mentioned in [Chapter 1, Introduction](#) the key to deploying CloudForms at scale is to monitor and tune at each stage of the scaling process. Once confidence has been established that the installation is working optimally at restricted scale, the scope of deployment can be enlarged and the CFME appliances tuned as required to handle the additional workload.

The VMDB and CFME worker appliances within a region have different monitoring requirements, as described below.

### 12.1. DATABASE APPLIANCE

The database appliance can become a performance bottleneck for the CloudForms region if it is not performing optimally. The following items should be regularly monitored:

- ✧ VMDB disk space utilization - monitor and forecast when 80% of filesystem will become filled. Track actual disk consumption versus expected consumption
- ✧ CPU utilization. A steady state utilization approaching 80% may indicate that VMDB appliance scaling or region redesign is required
- ✧ Memory utilization, especially swap usage
  - Increase appliance memory if swapping is occurring
- ✧ I/O throughput - use the `sysstat` or `iotop` tools to monitor I/O utilization, throughput, and I/O wait state processing
- ✧ Monitor the `miq_queue` table
  - Number of entries
    - Check for signs of event storm: messages with `role = 'event'` and `class_name = 'EmsEvent'`
  - Number of messages in a "ready" state
- ✧ Check that the maximum number of configured connections is not exceeded
- ✧ Ensure that the database maintenance scripts run regularly

### 12.2. CFME 'WORKER' APPLIANCES

Operational limits for non-VMDB or "worker" appliances are usually established on a per-appliance basis, and depend on the enabled server roles and number of worker processes. The following items are typically monitored:

#### 12.2.1. General Appliance

- ✧ CPU utilization
- ✧ Memory utilization, especially swap usage

- Increase appliance memory if swapping is occurring
- ✎ Check for message timeouts

### 12.2.2. Workers

- ✎ Review rates and reasons for worker process restarts
  - Increase allocated memory if workers are exceeding memory thresholds
- ✎ Validate that the primary/secondary roles for workers in zones and region are as expected, and force a role failover if necessary

#### 12.2.2.1. Provider Refresh

- ✎ Review EMS refresh activity, especially full refresh rates
  - How many full refreshes per day?
  - How long does a refresh take by provider instance?
    - Data extraction component
    - Database load component
  - Are refresh times consistent throughout the day?
    - What is causing periodic slowdowns?
  - Are certain property changes triggering too many refreshes?
- ✎ Validate the `:full_refresh_threshold` value

#### 12.2.2.2. Capacity & Utilization

- ✎ Are any realtime metrics being lost?
  - Long message dequeue times
  - Missing data samples
- ✎ How long does metric collection take?
  - Data extraction component
  - Database load component
- ✎ Are rollups completing in time?
  - Confirm expected daily and hourly records for each VM
- ✎ Validate the numbers of Data Collector and Data Processor workers

#### 12.2.2.3. Automate

- ✎ Are any requests staying in a "pending" state for a long time?
  - Validate the number of Generic workers

- ✎ Check for state machine retries or timeouts exceeded
- ✎ Monitor provisioning failures
  - Timeouts?
  - Internal or external factors?

#### 12.2.2.4. Event Handling

- ✎ Monitor the utilization of CFME appliances with the Event Monitor role enabled
- ✎ Validate the memory allocated to Event Monitor workers

#### 12.2.2.5. SmartState Analysis

- ✎ Monitor utilization of CFME appliances with the SmartProxy role enabled when scheduled scans are running
- ✎ Review scan failures or aborts
- ✎ Validate the number of SmartProxy workers

#### 12.2.2.6. Reporting


















- ✎ Monitor utilization of appliances with Reporting role enabled when periodic reports are running.
- ✎ Validate the number of Reporting workers

### 12.3. ALERTS

Some self-protection policies are available out-of-the-box in the form of control alerts. [Figure 12.1, “EVM Self-Monitoring Alerts”](#) shows the alert types that are available. Each is configurable to send an email, an SNMP trap, or run an automate instance.

**Figure 12.1. EVM Self-Monitoring Alerts**

## Alerts

	EVM Server Database Backup Insufficient Space
	EVM Server Exceeded Memory Limit
	EVM Server High App Disk Usage
	EVM Server High DB Disk Usage
	EVM Server High Log Disk Usage
	EVM Server High System Disk Usage
	EVM Server is Master
	EVM Server Not Responding
	EVM Server Start
	EVM Server Stop
	EVM Worker Exceeded Memory Limit
	EVM Worker Exceeded Uptime Limit
	EVM Worker Exit File
	EVM Worker Killed
	EVM Worker Not Responding
	EVM Worker Started
	EVM Worker Stopped



### Note

EVM Worker Started and EVM Worker Stopped events are normal occurrences and should not be considered cause for alarm

An email sent by one of these alerts will have a subject such as:

**Alert Triggered: EVM Worker Killed, for (MIQSERVER) cfmesrv06.**

The email body will contain text such as the following:

```
Alert 'EVM Worker Killed', triggered
```

```
Event: Alert condition met
```

```
Entity: (MiqServer) cfmesrv06
```

To determine more information - such as the actual worker type that was killed - it may be necessary to search *evm.log* on the appliance mentioned.

## 12.4. CONSOLIDATED LOGGING

The distributed nature of the worker/message architecture means that it is often impossible to predict which CFME appliance will run a particular action. This can add to the troubleshooting challenge of examining log files, as the correct appliance hosting the relevant log file must first be located.

Although there is no out-of-the-box consolidated logging architecture for CloudForms at the time of writing, it is possible to add **CloudForms logs as a source to an ELK/EFK stack**. This can bring a number of benefits, and greatly simplifies the task of log searching in a CloudForms deployment comprising many CFME appliances.

## CHAPTER 13. DESIGN SCENARIO

This chapter discusses a hypothetical region and zone design for a new CloudForms installation, based on the topics discussed in this guide.

### 13.1. ENVIRONMENT TO BE MANAGED

CloudForms is to be installed to manage the virtualization and cloud environments used by the Engineering and R&D departments of a large organization. These environments comprise a traditional virtual infrastructure, public and private IaaS clouds, and a container-based PaaS.

The organization also has a centrally-managed VMware 6.0 environment that hosts many enterprise-wide services such as email, file & print, collaboration, and the Microsoft Active Directory infrastructure. This will not be managed by CloudForms, although it is available to host CFME appliances if required.

#### 13.1.1. Virtual Infrastructure

Red Hat Virtualization 4.0 is installed as the Engineering/R&D virtual infrastructure. It currently comprises 2 clusters, 20 hosts, 10 storage domains and approximately 500 virtual machines. The number of VMs is not expected to grow significantly over the next two years.

#### 13.1.2. Private IaaS Cloud

A Red Hat OpenStack Platform 10 private IaaS cloud is installed. This contains approximately 900 images and instances spread between 50 tenant/projects, and also hosts the OpenShift PaaS. An OpenStack Director manages the Undercloud, comprising 42 Nova compute nodes.

The number of Overcloud instances is forecast to grow by approximately 400 per year over the next two years, giving a projected total of around 1900 managed objects.

#### 13.1.3. Public Clouds

A recently acquired subsidiary uses Amazon EC2 for cloud workloads. There are approximately 250 EC2 instances used by two accounts (separate access key IDs), but this number is expected to gradually reduce over the next two years as work is migrated to the OpenStack IaaS. Ansible playbooks are frequently used to configure Amazon EC2 cloud components such as Elastic Load Balancers.

#### 13.1.4. PaaS Cloud

A Red Hat OpenShift Container Platform 3.4 PaaS is installed, hosted in OpenStack, and currently comprising approximately 100 nodes, 750 pods and 1000 containers. This number is expected to rise to 300 nodes, 2000 pods and 3500 containers over the next two years.

#### 13.1.5. Network Factors

All in-house networking components are split between two campus datacenters. There is LAN-speed (<1MSec) latency between all points on this network. For security isolation the Engineering/R&D RHV, OpenStack and OpenShift environments are on separate vLANs, with only limited connectivity to the 'Enterprise' network.

Additional firewall routes into and out of the Enterprise network are possible, but require security change approval.

User workstations are connected to a 'Desktops' network, which has very limited access to servers in the Enterprise or Engineering/R&D networks. User who wish to access the CloudForms environment must connect to WebUI servers accessible from this Desktops network.

### 13.1.6. Enterprise Integration Points

The Enterprise network hosts common components such as a Configuration Management Database (CMDB) and an IPAM solution, however strict security policies are in place that restrict access to these components from non-Enterprise networks.

Virtual machines provisioned into the Engineering/R&D RHV and OpenStack networks may require registration with one or more of these enterprise tools.

### 13.1.7. Required CloudForms Functionality

The following capabilities of CloudForms are required:

- ✎ Inventory/Insight of all VMs, instances pods, containers and infrastructure components such as hosts and storage domains
- ✎ Rightsizing recommendations for cloud instances
- ✎ Reporting
- ✎ SmartState Analysis of RHV VMs and OpenShift containers
- ✎ Capacity and Utilization metrics for RHV and Amazon EC2
- ✎ Service catalog-based provisioning of VMs into RHV and instances into OpenStack.

The rightsizing calculation process uses metrics gathered by C&U, so this must also be enabled for cloud providers.

## 13.2. DESIGN PROCESS

The design process usually starts with sizing the region. How many VMs and containers will be managed in total, projected for the next 1-2 years? For this design scenario the projected number of objects to be managed over the next 2 years is shown in [Table 13.1, "Provider Object Numbers - 2 Year Projection"](#)

**Table 13.1. Provider Object Numbers - 2 Year Projection**

Provider	Number of objects
RHV	600
OpenStack	1900

Provider	Number of objects
OpenShift	5800
Amazon EC2	200
Total	8500

Based on the maximum suggested region sizes shown in [Table 3.1, “Guidelines for Maximum Region Size”](#), it can be estimated that a single region will be required, although this region will be large and require careful database tuning.

### 13.2.1. Network Latency

Latency from worker appliance to VMDB should be LAN speed, around 1ms or less. This will dictate where the VMDB appliance should be situated, and also the optimum location of the worker CFME appliances. For this design network latency is good, so the VMDB server should be placed in the most centrally accessible location.

### 13.2.2. VMDB Server

The optimum VMDB server for this design will be a CFME appliance configured as a standalone PostgreSQL server. Although database high availability (HA) has not been specified as an initial requirement, installing a standalone database appliance allows for HA to be configured in future if required.

The database server will be installed in the Enterprise network, hosted by the VMware 6.0 virtual infrastructure. The estimated size of the database after two years, based on the formula presented in [Chapter 4, Database Sizing and Optimization](#) is approximately 468 GBytes. To allow for unexpected growth and a margin of uncertainty, a 750 GByte disk will be presented from a datastore backed by fast FC SAN storage, and used as the database volume.

The database server will have 8 GBytes memory, and a PostgreSQL **shared\_buffers region of 2 GBytes. A 2 GByte hugepage region will be created for PostgreSQL to use.**

The planned zone design contains 13 CFME appliances. Referring to the table in [Appendix A, Database Appliance CPU Count](#) shows that the database server will need 6 vCPUs to maintain an idle CPU load under 20%.

The database maintenance scripts will be enabled on the VMDB server.

### 13.2.3. Zones

A zone should be created per provider (unless the EMS only manages a small number of systems; around 100 VMs or so). **There should be a minimum of 2 CFME appliances per zone for resilience, and zones should not span networks.**

For this design scenario the following zones are proposed.



### 13.2.4. WebUI Zone

A WebUI zone will be created that contains 2 CFME appliances, each running the following server roles:

- ✳ Automation Engine (to process zone events)
- ✳ Provider Operations (because VM provisioning services are used)
- ✳ Reporting (if logged-on users will be running their own reports)
- ✳ User Interface
- ✳ Web Services
- ✳ Websocket

The CFME appliances in this zone will be hosted by the enterprise VMware 6.0 environment, in a vLAN accessible from user workstations. User access to them will be via a hardware load-balancer and common Fully-Qualified Domain Name.

### 13.2.5. Management Zone

A Management zone will be created that contains 2 CFME appliances, each running the following server roles:

- ✳ Automation Engine
- ✳ Provider Operations
- ✳ Reporting (for scheduled reports)
- ✳ Database Operations
- ✳ Notifier
- ✳ Scheduler
- ✳ Git Repositories Owner
- ✳ User Interface
- ✳ Web Services
- ✳ Websocket

The CFME appliances in this zone will be hosted by the enterprise VMware 6.0 environment. The zone will not contain any providers, but automate workflows that interact with the CMDB and IPAM solutions will run in this zone.

### 13.2.6. RHV Zone

The RHV zone will contain approximately 600 managed objects. The table [Table 3.2, “Objects per CFME Appliance Guidelines”](#) suggests that 2 appliances should be sufficient, each running the following server roles:

- ✳ Automation Engine
- ✳ 3 x C&U roles

- ✳ Provider Inventory
- ✳ Provider Operations
- ✳ Event Monitor
- ✳ SmartProxy
- ✳ SmartState Analysis
- ✳ Git Repositories Owner
- ✳ User Interface
- ✳ Web Services
- ✳ Websocket

The CFME appliances in this zone will be hosted by the RHV environment, and so firewall ports must be opened to allow these appliances to connect to the VMDB server in the Enterprise network. The RHV provider will be in this zone.

### 13.2.7. OpenStack zone

The OpenStack zone will initially contain approximately 900 instances (for example instances, images, tenants, or networks), increasing to 1700 in two years time. The table [Table 3.2, “Objects per CFME Appliance Guidelines”](#) suggests that 3 appliances should be sufficient initially, each running the following server roles:

- ✳ Automation Engine
- ✳ 3 x C&U roles
- ✳ Provider Inventory
- ✳ Provider Operations
- ✳ Event Monitor
- ✳ Git Repositories Owner
- ✳ User Interface
- ✳ Web Services
- ✳ Websocket

The CFME appliances in this zone will be hosted by the OpenStack environment, and so firewall ports must be opened and routes created to allow these appliances to connect to the VMDB server in the Enterprise network, and to the OpenStack Director. Both OpenStack Cloud and Infrastructure Manager (Undercloud) providers will be in this zone.

Further appliances will need to be added to this zone as the number of managed objects increases.

### 13.2.8. OpenShift Zone

The OpenShift zone will contain approximately 800 managed objects. The table [Table 3.2, “Objects per CFME Appliance Guidelines”](#) suggests that 2 appliances should be sufficient initially, each running the following server roles:

- ✧ Automation Engine
- ✧ 3 x C&U roles
- ✧ Provider Inventory
- ✧ Provider Operations
- ✧ Event Monitor
- ✧ SmartProxy
- ✧ SmartState Analysis
- ✧ Git Repositories Owner
- ✧ User Interface
- ✧ Web Services
- ✧ Websocket

The CFME appliances in this zone will also be hosted by the OpenStack environment, and so firewall ports must be opened and routes created to allow these appliances to connect to the VMDB server in the Enterprise network, and to the OpenShift master. The OpenShift provider will be in this zone.

Further appliances will need to be added to this zone as the number of managed objects increases.

### 13.2.9. Amazon EC2 Zone

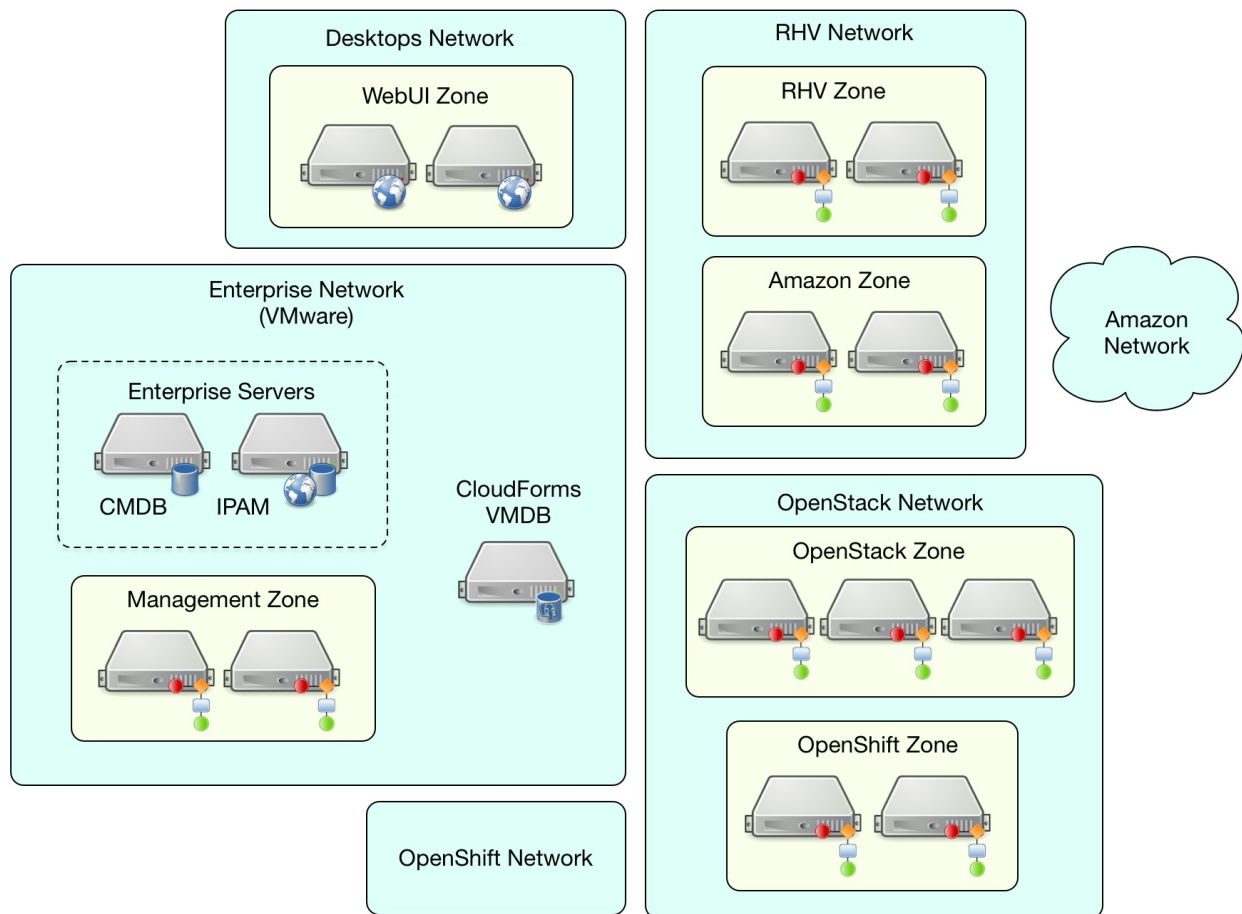
The Amazon zone will contain approximately 250 managed objects. The table [Table 3.2, “Objects per CFME Appliance Guidelines”](#) suggests that 1 appliance should be sufficient, however for resilience and load balancing 2 will be installed, each running the following server roles:

- ✧ Automation Engine
- ✧ 3 x C&U roles
- ✧ Embedded Ansible
- ✧ Provider Inventory
- ✧ Provider Operations
- ✧ Event Monitor
- ✧ Git Repositories Owner
- ✧ User Interface
- ✧ Web Services
- ✧ Websocket

The CFME appliances in this zone will be hosted on a separate vLAN in the RHV environment, and so firewall ports must be opened to allow these appliances to connect to the VMDB server in the Enterprise network, and to the Amazon EC2 network. The Embedded Ansible role will be enabled on these CFME appliances so that Ansible playbooks can be run from service catalogs. The Amazon EC2 providers for both accounts will be in this zone.

The proposed zone design is shown in [Figure 13.1, “Networks and Zones”](#).

**Figure 13.1. Networks and Zones**



### 13.3. INITIAL DEPLOYMENT

The initial deployment and configuration of CFME appliances will be made without enabling the C&U or SmartState Analysis roles on any server. This allows the baseline VMDB database server load to be established over a period of several days from purely EMS refresh activity, and allow an initial **RHV :full\_refresh\_threshold** to be calculated.

Once the initial performance baselines have been established (and any associated tuning performed), the remaining roles can be enabled. Ongoing monitoring at this stage is important, as this will help fine-tune the number and configuration of worker processes, CFME appliance vCPU and memory sizes, and database configuration parameters.

### 13.4. PROVISIONING WORKFLOW

The VM provisioning workflow (which will run in an automation engine in one of the provider zones) will require the services of the CMDB and IPAM servers that are only accessible from the Enterprise network. The workflow can be customised to use the techniques discussed in [Chapter 7, Automate](#) and [Chapter 8, VM and Instance Provisioning](#) to launch new child automation requests using `$vm.execute(create_automation_request,...)` at each of the **AcquireIPAddress** and **RegisterCMDB** states of the VM provision state machine.

The `:miq_zone` option for `create_automation_request` will specify the Management zone as the target zone in which to run the request. Newly inserted states **CheckIPAddressAcquired** and **CheckCMDBUpdated** will use check-and retry logic to determine completion of the child requests.

## CHAPTER 14. CONCLUSION

As can be seen from the previous chapters, the architecture of CloudForms is inherently scalable.

- ✦ The role/worker/message model allows server roles to be distributed throughout CFME appliances in a region.
- ✦ The appliance model allows for both horizontal and vertical scaling
  - The number of worker processes can be increased on each CFME appliance (scaling out workers)
  - The appliance vCPU count and memory can be increased (scaling up each appliance)
  - Additional CFME appliances can be added to a region (scaling out appliances)
- ✦ The zone model allows containment of provider-specific workers, appliances and workflow processing
- ✦ The region model allows many regions to be grouped together under a single master region

The unique performance and load characteristics of individual virtual infrastructure, container or cloud platforms, and the many permutations of provider mean that there is no "magic formula" for tuning. Deploying CloudForms at scale involves careful monitoring and tuning of the various components; detecting low memory or high CPU conditions for workers and appliances, or identifying the conditions that trigger message timeouts, for example.

The scaling process is made easier by starting with a minimal set of server roles enabled to support the configured providers; inventory and event handling, for example. Once the CloudForms installation is optimally tuned for average and peak EMS load, performance baselines can be established and used as a reference. Additional features such as capacity & utilization metrics collection, SmartState Analysis, provisioning, and automate workflows can then be enabled as required, with performance being monitored and compared against the baselines, and appliances and workers tuned at each step.

Before this can be done however, an understanding of the components and how they fit together is necessary. The architectural and troubleshooting descriptions in this guide are presented as a means to further this understanding.

## APPENDIX A. DATABASE APPLIANCE CPU COUNT

The following table shows the anticipated CPU load on the VMDB appliance for a varying number of idle CFME appliances in a region. An average number of 20 worker processes per CFME appliance is assumed, where each worker process creates a single PostgreSQL session. The CPU consumed per idle PostgreSQL session is approximately 0.00435%.

Figure A.1. Database Server CPU Count

	Database Appliance vCPU Count											
	2	4	6	8	10	12	14	16	18	20	22	24
1	4.35%	2.18%	1.45%	1.09%	0.87%	0.73%	0.62%	0.54%	0.48%	0.44%	0.40%	0.36%
2	8.70%	4.35%	2.90%	2.18%	1.74%	1.45%	1.24%	1.09%	0.97%	0.87%	0.79%	0.73%
3	13.05%	6.53%	4.35%	3.26%	2.61%	2.18%	1.86%	1.63%	1.45%	1.31%	1.19%	1.09%
4	17.40%	8.70%	5.80%	4.35%	3.48%	2.90%	2.49%	2.18%	1.93%	1.74%	1.58%	1.45%
5	21.75%	10.88%	7.25%	5.44%	4.35%	3.63%	3.11%	2.72%	2.42%	2.18%	1.98%	1.81%
6	26.10%	13.05%	8.70%	6.53%	5.22%	4.35%	3.73%	3.26%	2.90%	2.61%	2.37%	2.18%
7	30.45%	15.23%	10.15%	7.61%	6.09%	5.08%	4.35%	3.81%	3.38%	3.05%	2.77%	2.54%
8	34.80%	17.40%	11.60%	8.70%	6.96%	5.80%	4.97%	4.35%	3.87%	3.48%	3.16%	2.90%
9	39.15%	19.58%	13.05%	9.79%	7.83%	6.53%	5.59%	4.89%	4.35%	3.92%	3.56%	3.26%
10	43.50%	21.75%	14.50%	10.88%	8.70%	7.25%	6.21%	5.44%	4.83%	4.35%	3.95%	3.63%
11	47.85%	23.93%	15.95%	11.96%	9.57%	7.98%	6.84%	5.98%	5.32%	4.79%	4.35%	3.99%
12	52.20%	26.10%	17.40%	13.05%	10.44%	8.70%	7.46%	6.53%	5.80%	5.22%	4.75%	4.35%
13	56.55%	28.28%	18.85%	14.14%	11.31%	9.43%	8.08%	7.07%	6.28%	5.66%	5.14%	4.71%
14	60.90%	30.45%	20.30%	15.23%	12.18%	10.15%	8.70%	7.61%	6.77%	6.09%	5.54%	5.08%
15	65.25%	32.63%	21.75%	16.31%	13.05%	10.88%	9.32%	8.16%	7.25%	6.53%	5.93%	5.44%
16	69.60%	34.80%	23.20%	17.40%	13.92%	11.60%	9.94%	8.70%	7.73%	6.96%	6.33%	5.80%
17	73.95%	36.98%	24.65%	18.49%	14.79%	12.33%	10.56%	9.24%	8.22%	7.40%	6.72%	6.16%
18	78.30%	39.15%	26.10%	19.58%	15.66%	13.05%	11.19%	9.79%	8.70%	7.83%	7.12%	6.53%
19	82.65%	41.33%	27.55%	20.66%	16.53%	13.78%	11.81%	10.33%	9.18%	8.27%	7.51%	6.89%
20	87.00%	43.50%	29.00%	21.75%	17.40%	14.50%	12.43%	10.88%	9.67%	8.70%	7.91%	7.25%
21	91.35%	45.68%	30.45%	22.84%	18.27%	15.23%	13.05%	11.42%	10.15%	9.14%	8.30%	7.61%
22	95.70%	47.85%	31.90%	23.93%	19.14%	15.95%	13.67%	11.96%	10.63%	9.57%	8.70%	7.98%
23	100.05%	50.03%	33.35%	25.01%	20.01%	16.68%	14.29%	12.51%	11.12%	10.01%	9.10%	8.34%
24	104.40%	52.20%	34.80%	26.10%	20.88%	17.40%	14.91%	13.05%	11.60%	10.44%	9.49%	8.70%
25	108.75%	54.38%	36.25%	27.19%	21.75%	18.13%	15.54%	13.59%	12.08%	10.88%	9.89%	9.06%
26	113.10%	56.55%	37.70%	28.28%	22.62%	18.85%	16.16%	14.14%	12.57%	11.31%	10.28%	9.43%
27	117.45%	58.73%	39.15%	29.36%	23.49%	19.58%	16.78%	14.68%	13.05%	11.75%	10.68%	9.79%
28	121.80%	60.90%	40.60%	30.45%	24.36%	20.30%	17.40%	15.23%	13.53%	12.18%	11.07%	10.15%
29	126.15%	63.08%	42.05%	31.54%	25.23%	21.03%	18.02%	15.77%	14.02%	12.62%	11.47%	10.51%
30	130.50%	65.25%	43.50%	32.63%	26.10%	21.75%	18.64%	16.31%	14.50%	13.05%	11.86%	10.88%
31	134.85%	67.43%	44.95%	33.71%	26.97%	22.48%	19.26%	16.86%	14.98%	13.49%	12.26%	11.24%
32	139.20%	69.60%	46.40%	34.80%	27.84%	23.20%	19.89%	17.40%	15.47%	13.92%	12.65%	11.60%
33	143.55%	71.78%	47.85%	35.89%	28.71%	23.93%	20.51%	17.94%	15.95%	14.36%	13.05%	11.96%
34	147.90%	73.95%	49.30%	36.98%	29.58%	24.65%	21.13%	18.49%	16.43%	14.79%	13.45%	12.33%
35	152.25%	76.13%	50.75%	38.06%	30.45%	25.38%	21.75%	19.03%	16.92%	15.23%	13.84%	12.69%
36	156.60%	78.30%	52.20%	39.15%	31.32%	26.10%	22.37%	19.58%	17.40%	15.66%	14.24%	13.05%
37	160.95%	80.48%	53.65%	40.24%	32.19%	26.83%	22.99%	20.12%	17.88%	16.10%	14.63%	13.41%
38	165.30%	82.65%	55.10%	41.33%	33.06%	27.55%	23.61%	20.66%	18.37%	16.53%	15.03%	13.78%
39	169.65%	84.83%	56.55%	42.41%	33.93%	28.28%	24.24%	21.21%	18.85%	16.97%	15.42%	14.14%
40	174.00%	87.00%	58.00%	43.50%	34.80%	29.00%	24.86%	21.75%	19.33%	17.40%	15.82%	14.50%
41	178.35%	89.18%	59.45%	44.59%	35.67%	29.73%	25.48%	22.29%	19.82%	17.84%	16.21%	14.86%
42	182.70%	91.35%	60.90%	45.68%	36.54%	30.45%	26.10%	22.84%	20.30%	18.27%	16.61%	15.23%
43	187.05%	93.53%	62.35%	46.76%	37.41%	31.18%	26.72%	23.38%	20.78%	18.71%	17.00%	15.59%
44	191.40%	95.70%	63.80%	47.85%	38.28%	31.90%	27.34%	23.93%	21.27%	19.14%	17.40%	15.95%
45	195.75%	97.88%	65.25%	48.94%	39.15%	32.63%	27.96%	24.47%	21.75%	19.58%	17.80%	16.31%
46	200.10%	100.05%	66.70%	50.03%	40.02%	33.35%	28.59%	25.01%	22.23%	20.01%	18.19%	16.68%
47	204.45%	102.23%	68.15%	51.11%	40.89%	34.08%	29.21%	25.56%	22.72%	20.45%	18.59%	17.04%
48	208.80%	104.40%	69.60%	52.20%	41.76%	34.80%	29.83%	26.10%	23.20%	20.88%	18.98%	17.40%
49	213.15%	106.58%	71.05%	53.29%	42.63%	35.53%	30.45%	26.64%	23.68%	21.32%	19.38%	17.76%
50	217.50%	108.75%	72.50%	54.38%	43.50%	36.25%	31.07%	27.19%	24.17%	21.75%	19.77%	18.13%

## APPENDIX B. CONTRIBUTORS

Contributor	Title	Contribution
Peter McGowan	Principal Software Engineer	Author
Tom Hennessy	Principal Software Engineer	Content, Review
Bill Helgeson	Principal Domain Architect	Content
Brett Thurber	Engineering Manager	Review
Christian Jung	Senior Specialist Solution Architect	Review
Chandler Wilkerson	Senior Software Engineer	Review



## APPENDIX C. REVISION HISTORY

---

Revision 1.1-2	2017-05-25	PM
----------------	------------	----