Reference Architectures 2018

Managing Large-Scale OpenShift Installations with CloudForms 4.6

Last Updated: 2018-06-29

Reference Architectures 2018 Managing Large-Scale OpenShift Installations with CloudForms 4.6

Peter McGowan
pemcg@redhat.com

Legal Notice

Copyright © 2018 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 4.6 to manage large-scale OpenShift Container Platform installations

Table of Contents

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. Please refer to the title within the email.

CHAPTER 1. INTRODUCTION

This document discusses how best to deploy CloudForms 4.6 to manage large OpenShift Container Platform (OCP) installations. The term "large" in this case infers several thousand managed nodes, projects, services, containers or pods, in possibly several OCP clusters

Experience shows that the most effective way to deploy CloudForms in large OpenShift Container Platform 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 or CloudForms OpenShift project is tuned for relatively small environments, the product *is* scalable to manage many thousands of managed pods 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 or StatefulSet 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, particularly with reference to managing large OpenShift providers. Several 'rules of thumb' such as guidelines for CFME appliance to pod/container 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 - pods, nodes or containers for example - from an OpenShift Container Platform cluster.
- *Capacity and Utilization* explains how the three types of C&U worker interact to extract and process performance metrics from an OpenShift Container Platform cluster.
- *Kubernetes Event Handling* describes the three workers that combine to process events from an OpenShift Container Platform cluster, and how to scale them.
- *SmartState Analysis and OpenSCAP Compliance Checking* decsribes the SmartState Analysis process for container images, and takes a look at some of the tuning options available to scale SmartState Analysis in larger environments.
- *Monitoring* describes some of the in-built monitoring capabilities for both OpenShift Container Platform and CloudForms.
- *Web User Interface* discusses how to scale WebUI appliances behind load balancers.

Part III - Design Scenario

- *Design Scenario* describes a typical region, zone and CFME appliance layout for a large multi-cluster OpenShift Container Platform installation.

CHAPTER 2. CLOUDFORMS ARCHITECTURE

In order to understand how best to install and configure CloudForms to manage large-scale OpenShift Container Platform deployments, 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. DEPLOYMENT METHOD

CloudForms 4.6 can be installed as one or more virtual machines, or in a "podified" form, running in pods and containers natively in OpenShift Container Platform 3.7 or 3.9.

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
5.9	4.6

CloudForms Management Engine 5.9 (CloudForms 4.6) runs Red Hat Enterprise Linux 7.4, with PostgreSQL 9.5, Rails 5.0.6, the CloudForms evmserverd service, and all associated Ruby gems installed.

2.1.1. Appliance-Based Installation

To simplify installation, the Red Hat CloudForms product is available as a self-contained virtual machine template, which when cloned becomes a CloudForms Management Engine (CFME) *appliance*.

The self-contained nature of appliances makes them ideal for horizontally scaling a VM-based 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. If using the appliance version of CloudForms to manage OpenShift Container Platform, it is often convenient to run the CFME virtual machines in the virtual infrastructure or cloud hosting the OpenShift Container Platform nodes.

2.1.2. Podified Installation

CloudForms can also be installed in a podified form running natively in OpenShift Container Platform 3.7 or 3.9. This method of deployment uses 4-6 pods depending on scale and configuration, running the following applications:

- postgresql (optional)
- apache
- memcached
- cloudforms (including Web/UI workers) managed as a scalable StatefulSet
- cloudforms-backend (no Web/UI workers) managed as a scalable StatefulSet (optional)
- ansible

For optimum database performance or if a highly available database configuration is required, an external PostgreSQL database server (or CFME appliance) is recommended. In this case the PostgreSQL pod is not required.

NOTE

Each replica of the cloudforms and cloudforms-backend StatefulSet requires a 5GiB Persistent Volume

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 podified or 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 Kubernetes or a vCenter server) 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 OpenShift Container Platform cluster.

Providers are broadly divided into categories, and in CloudForms 4.6 these are Cloud, Infrastructure, Container, Configuration Management, Automation, Network and Storage.[1]

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>
```

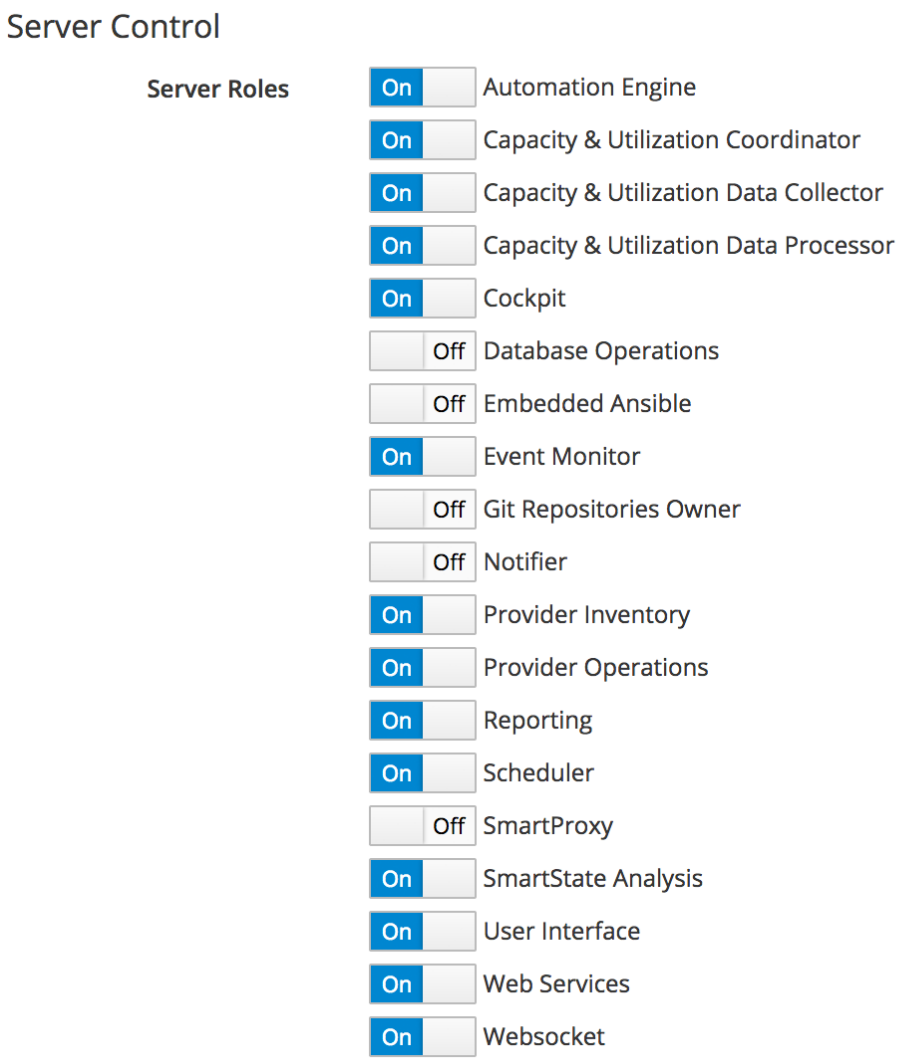
Some examples of this are as follows:

- ManageIQ::Providers::EmbeddedAnsible::AutomationManager
- ManageIQ::Providers::Openshift::ContainerManager
- ManageIQ::Providers::Kubernetes::ContainerManager::Container

2.5. SERVER ROLES

A CloudForms Management Engine 5.9 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 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. [2]. 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*

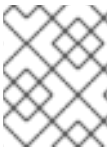
2.5.2. Capacity and Utilization

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 containing an OpenShift Container Platform provider would contain one or more *ManageIQ::Providers::Openshift::ContainerManager::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. Cockpit

The *Cockpit* role starts the cockpit webservice worker on an appliance. This allows users to connect to cockpit on a managed VM or host using their existing CloudForms WebUI session.

This role has a dedicated *MiqCockpitWsWorker* worker.

2.5.4. Database Operations

The *Database Operations* role enables a CFME appliance to run certain database maintenance tasks such as purging old metrics. On CloudForms Management Engine 5.9.2 and later appliances this role also runs the database maintenance scripts. This role does not have a dedicated worker, the database operations tasks are processed by a *MiqGenericWorker*.

2.5.5. Embedded Ansible

The *Embedded Ansible* role enables the use of the built-in "Ansible Inside" automation manager, which allows Ansible playbooks to be run from service catalogs, from control actions and alerts, or as Automate methods. 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

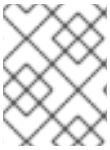
2.5.6. Event Monitor

The *Event Monitor* role is responsible for detecting and processing provider events such as a pod starting or stopping, or a node 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 (Kubernetes in the case of OpenShift Container Platform) 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 an OpenShift Container Platform provider would contain a *ManageIQ::Providers::Openshift::ContainerManager::EventCatcher* worker and an optional *ManageIQ::Providers::Openshift::MonitoringManager::EventCatcher*, for example.

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 7, *Kubernetes Event Handling*

2.5.7. 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.

2.5.8. Notifier

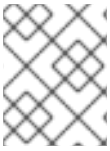
The *Notifier* role should be enabled if CloudForms is required to forward SNMP traps to a monitoring system, or to send e-mails. These might be initiated by an automate method or from a control policy, for example.

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.

2.5.9. Provider Inventory

The *Provider Inventory* role is responsible for refreshing inventory data for all provider objects such as pods, containers, projects, or nodes. 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 an OpenShift Container Platform provider would contain a *ManageIQ::Providers::Openshift::ContainerManager::RefreshWorker* worker, for example.

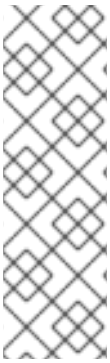


NOTE

Provider Inventory refresh workers are described in more detail in Chapter 5, *Inventory Refresh*

2.5.10. 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 process service template provisioning requests.

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.[3] (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.

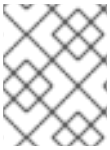
The Scheduler role is for region-specific scheduling and is only active on one appliance per region.

2.5.13. SmartProxy

Enabling the *SmartProxy* role turns on the embedded SmartProxy on the CFME appliance, which analyses the results of a SmartState Analysis on a container image. 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.

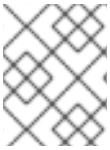


NOTE

The SmartProxy and SmartState Analysis roles are described in more detail in Chapter 8, *SmartState Analysis and OpenSCAP Compliance Checking*

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 10, *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.

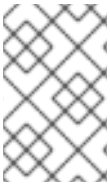
2.5.18. Server Role Zone Affinity

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:

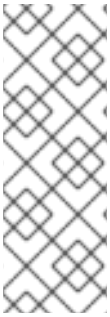
```
vmdb
bin/rake evm:status
```

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 OpenShift	started	33388	38066	openshift
	C&U Metrics Collector for OpenShift	started	33396	38067	openshift
	C&U Metrics Processor	started	1972	40550	ems_metrics_processor
	C&U Metrics Processor	started	1964	40549	ems_metrics_processor
	Event Handler	started	1985	40558	ems
	Event Monitor for Providers: e2e OpenShift	started	33413	38071	ems_1000000000013
	Event Monitor for Providers: e2e OpenShift Origin	started	34031	38360	ems_1000000000014
	Generic Worker	started	1846	40500	generic
	Generic Worker	started	1854	40503	generic
	Priority Worker	started	1870	40501	generic
	Priority Worker	started	1862	40502	generic
	Refresh Worker for Providers: e2e OpenShift	started	7554	58657	ems_1000000000013
	Refresh Worker for Providers: e2e OpenShift Origin	started	7931	58949	ems_1000000000014



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.[4]), 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.[5]. 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::Openshift::ContainerManager::RefreshWorker] ⏎
with ID: [10000001113926], PID: [7858], ⏎
GUID: [76ffdf4c-1f96-11e7-8750-0050568712f3] 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.

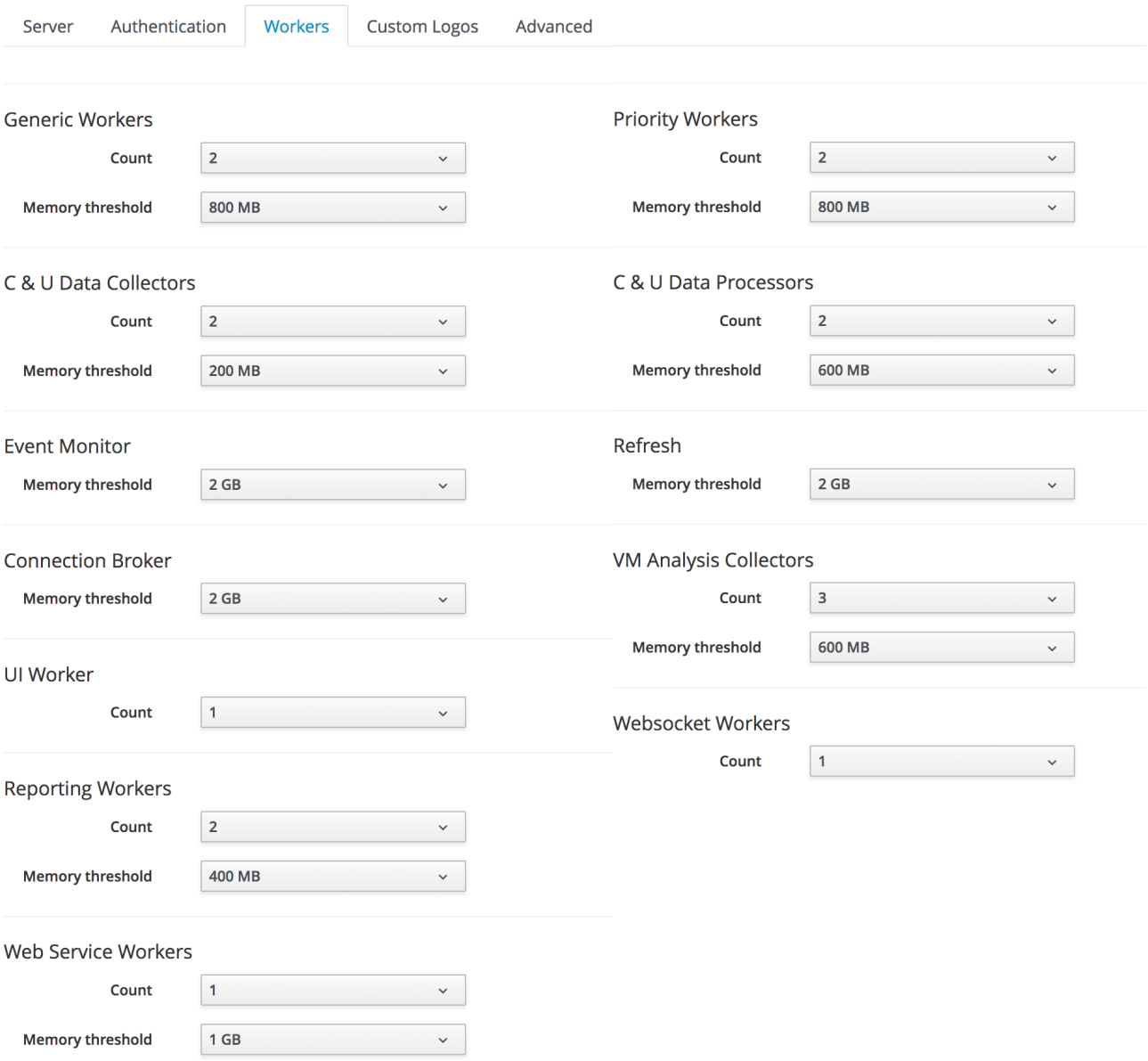
2.6.2.1. Worker Memory Thresholds

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



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
```

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, 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 per region
C&U Coordinator	Generic or Priority	ems_metrics_coordinator	one per zone
C&U Data Collector	provider-specific MetricsCollectorWorker	ems_metrics_collector	unlimited per zone
C&U Data Processor	MiqEmsMetricsProcessorWorker	ems_metrics_processor	unlimited per zone
Cockpit	MiqCockpitWsWorker	N/A	one per zone
Database Operations	Generic or Priority	database_owner	unlimited per region
Embedded Ansible	EmbeddedAnsibleWorker	N/A	one per region
Event Monitor	MiqEventHandler & provider-specific EventCatchers	event	one per zone & one per provider per zone
Git Repositories Owner	N/A	N/A	one per zone
Notifier	Generic or Priority	notifier	one per region
Provider Inventory	provider-specific RefreshWorker	ems_inventory	one per provider per zone
Provider Operations	Generic or Priority	ems_operations	unlimited per zone
Reporting	MiqReportingWorker	reporting	unlimited per region
Scheduler	MiqScheduleWorker	N/A	one per region
SmartProxy	MiqSmartProxyWorker	smartproxy	unlimited per zone

Role	Worker	Message 'Role'	Maximum Concurrent Workers
SmartState Analysis	Generic or Priority	smartstate	unlimited per zone
User Interface	MiqUiWorker	N/A	unlimited per region
Web Services	MiqWebServiceWorker	N/A	unlimited per region
Web Socket	MiqWebsocketWorker	N/A	unlimited per 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/4.6/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

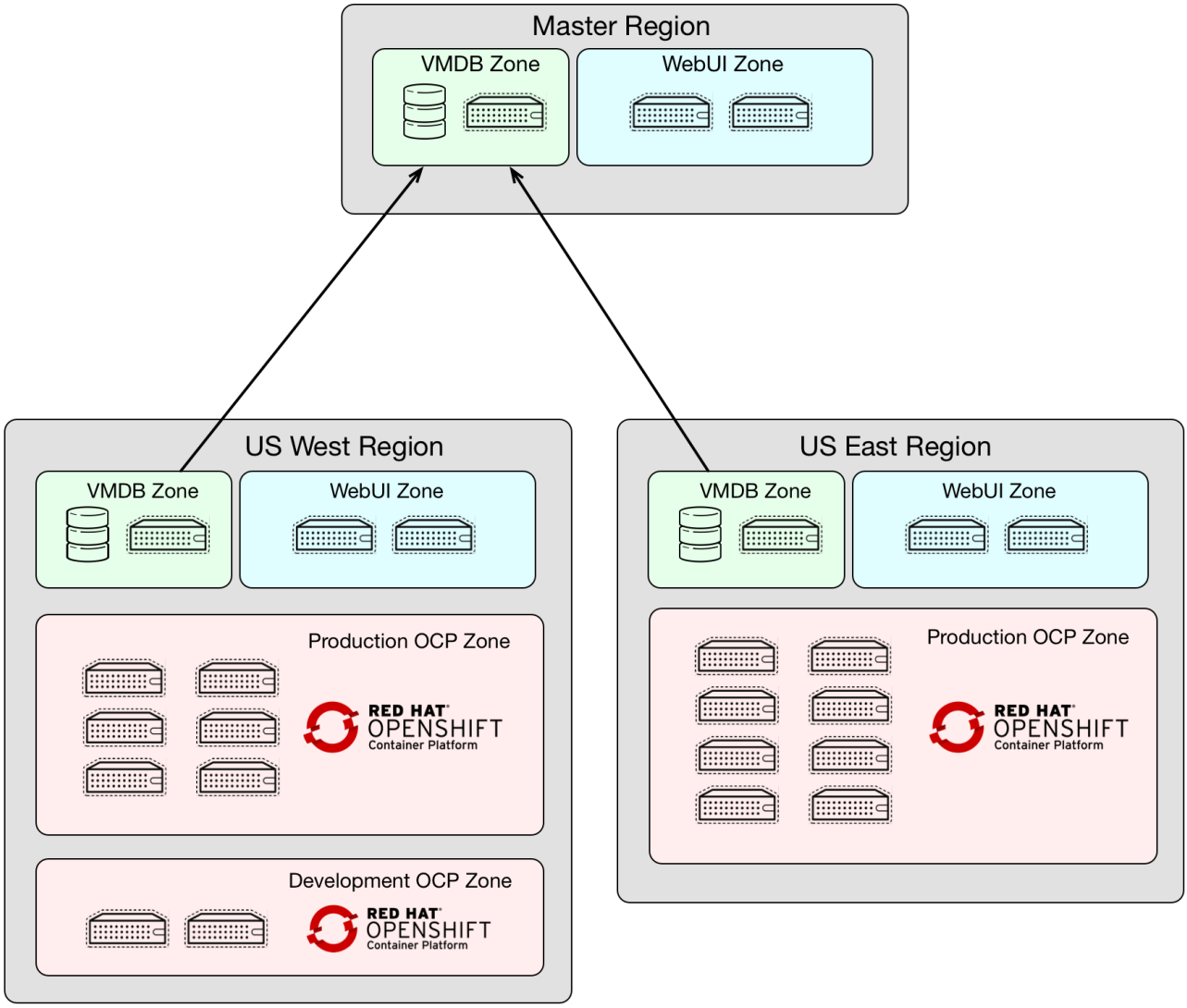
[4] Worker processes issue a heartbeat ping every 10 seconds

[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 there are two geographical "subordinate" regions containing providers (US West and US East), and one master region. Each of the subordinate regions has its own VMDB database, managed in its own dedicated VMDB zone. The two subordinate region VMDBs are replicated to the master region VMDB.

Each region has a dedicated WebUI zone containing two CFME appliances (load-balanced), that users local to the region connect to for interactive management. The two subordinate regions each have one or more provider-specific zones, containing the CFME appliances that manage the workload for their respective OpenShift Container Platform providers.

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 *evmserverd* 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 or pods 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 total number of managed objects (for example images, nodes, routes, port configs, projects, services, pods, containers, etc.) in the region
- The number of running or "active" objects generating events (typically nodes, pods and containers)
- The number of OpenShift Container Platform providers/clusters added to the region
- The overall "busyness" of the OpenShift Container Platform cluster, which determines the rate at which Kubernetes events are received and processed, and thus the rate at which inventory refreshes are requested and loaded
- Whether or not Capacity and Utilization (C&U) metric collection is enabled for the region
  - The frequency of collection
- Whether or not SmartState Analysis and OpenSCAP scanning is enabled for the region
  - The frequency of analysis
- The complexity of reports and widgets, and frequency of generation
- 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 containers
- 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

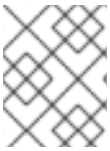
3.1.1.2. Sizing Estimation

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 20000 active or running objects (for example nodes, pods and containers). Larger regions than this are possible if the overall database workload is lighter, but as with any large database system, performance should be carefully monitored.

When sizing a region, some thought must be also 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 (or *cloudforms-backend* StatefulSet replicas) and active database connections.

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.

3.1.1.3. Scaling Workers in a Podified CloudForms Region

An initial deployment of the podified distribution of CloudForms 4.6 starts a single *cloudforms* StatefulSet, containing all workers. This is the functional equivalent of a single CFME appliance. The numbers of workers in this StatefulSet replica can be increased up to their limits, after which there will be a requirement to add further pods to scale out the numbers of worker processes (the equivalent of adding further CFME appliances). For this the *cloudforms-backend* StatefulSet should be used, which contains no externally accessible Web or UI components.

3.1.2. 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 components being managed.

3.1.2.1. Centrally Located Infrastructure

With a single, centrally located small or medium sized OpenShift Container Platform cluster, 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.

3.1.2.2. Distributed Infrastructure

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

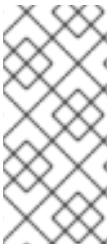
3.1.2.2.1. Wide Area Network Factors - Intra-Region

Network latency between CFME appliances and the database *within a region* plays a big factor in overall CloudForms "system" responsiveness. There is a utility, db_ping, supplied on each CFME appliance that can check the latency between an existing appliance and its own regional database. It is 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
```



NOTE

On CFME versions prior to 5.8, this tool should be prefixed by **`bin/rails runner`**, for example:

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

The architecture of CloudForms assumes LAN-speed latency (≈ 1 ms) between CFME appliances and their regional 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 detecting short-lived pods.
    - Metrics collection might not keep up with Prometheus metrics retention time.[7]

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.[8]

3.1.2.2.2. Wide Area Network Factors - Inter-Region

Network latency between subordinate and master regions is less critical as database replication occurs asynchronously. Latencies of 100 ms have been tested and shown to present no performance problems.

A second utility, db_ping_remote, is designed to check inter-region latency. It requires external PostgreSQL server details and credentials, and is run as follows:

```
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:
10.874407 ms
```

```
10.984994 ms
11.040376 ms
11.119602 ms
11.031609 ms

Average: 11.010198 ms
```

3.1.2.2.3. 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)

3.1.2.2.4. 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

3.1.3. 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 such as service provisioning to be performed directly from the global region, utilising a RESTful API connection to the correct child region to perform the action.

3.1.4. 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.

3.1.5. 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 or pods 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 (i.e. OpenShift Container Platform clusters). 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

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 or pods in the zone, and allows for dedicated per-provider Event Handler and Metrics Processor workers. The prevents an event surge from one OpenShift Container Platform cluster from adversely affecting the handling of events from another cluster, for example.

3.2.1.2. Appliance Maintenance

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

3.2.1.3. Cluster-Specific Appliance Tuning

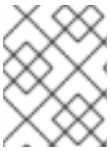
Zones allow for more predictable and provider-instance-specific sizing of CFME appliances and appliance settings based on the requirement of individual OpenShift Container Platform clusters. For example small clusters can have significantly different resourcing requirements to very large clusters, especially 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 or external 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

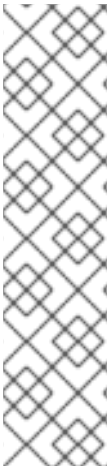
3.2.1.6. Improved and Simplified Diagnostics Gathering

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. Number of CFME Appliances or Pods in a Zone

One of CloudForms' most resource-intensive tasks is metrics collection, performed by the C&U Data Collector workers. It has been established through testing that a single CFME 5.9.2 C&U Data Collector worker can retrieve and process the Hawkular metrics from approximately 1500 objects (750 pods, each with 1 container) in the default 50 minute capture_threshold period.

This 1:1500 guideline ratio is a convenient starting point for scaling the number of CFME appliance/pods required for a zone containing an OpenShift Container Platform provider. For example a default CFME appliance with 2 C&U Data Collector workers can manage approximately 3000 pods and/or containers. If the number of C&U Data Collector workers is increased to 4, the appliance should be able to manage approximately 6000 pods and/or containers (e.g. 3000 pods, each with 1 container; or 2000 pods, each with 2 containers).



NOTE

The limiting factor in determining C&U data collection performance can often be the configuration of Hawkular itself, or the resources given to the nodes running the *openshift-infra* project components. For example the number of CloudForms C&U Data Collector workers can be increased linearly by scaling out worker processes and CFME appliances, but each concurrent C&U Data Collector worker process contributes to the workload of the Hawkular and Cassandra pods, and therefore CPU, I/O and memory load on the nodes.

It has been observed that if the nodes hosting the *openshift-infra* project components are already highly utilised, adding further C&U Data Collector workers will have an adverse rather than beneficial affect on C&U collection performance.

3.2.3. Zone Summary and Recommendations

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

- Use a separate zone per OpenShift Container Platform cluster
- 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 or pod in each zone should have the 'Automate Engine' role enabled, to process zone-specific events
- At least once CFME appliance or pod 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)

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/

[7] discussed in Chapter 6, *Capacity & Utilization*

[8] See https://bugzilla.redhat.com/show_bug.cgi?id=1422671

CHAPTER 4. DATABASE SIZING AND OPTIMIZATION

As discussed in Chapter 2, *CloudForms Architecture* CloudForms 4.6 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. The degree of customisation and type of tuning depends on whether CloudForms is running as a Virtual Machine external to the OpenShift Container Platform cluster, or podified within OpenShift Container Platform itself.

4.1. POSTGRESQL RUNNING IN A VIRTUAL MACHINE

The following sections decsribe the tuning that can be performed if CloudForms is running as a Virtual Machine external to the OpenShift Container Platform cluster.

4.1.1. 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 active or "running" nodes, pods and containers should first be established. Using the appliance to OCP object ratios suggested in Chapter 3, *Region and Zones*, the CFME appliance count can be estimated. For example an OpenShift Container Platform cluster containing 6000 active nodes, pods and containers would need approximately 2 CFME appliances in default configuration to handle a typical workload. If a WebUI zone is to be used, this should typically contain a further 2 CFME appliances.

It can be seen from the table in Appendix A, *Database Appliance CPU Count* that a region containing 4 idle CFME appliances would need a database server with 2 vCPUs to maintain CPU utilisation at under 20%. 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)

4.1.2. Configuring the Database Partition

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.

4.1.3. 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 **evmserverd** 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 **evmserverd** 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 **evmserverd** 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.

4.1.3.1. Configuring PostgreSQL

The PostgreSQL configuration file on a newly installed CFME 5.9 appliance is */var/opt/rh/rh-postgresql95/lib/pgsql/data/postgresql.conf*. This file contains no CloudForms-specific settings, however it makes reference to a CloudForms/ManageIQ-specific include directory with the line:

```
include_dir = '/etc/manageiq/postgresql.conf.d'
```

All CloudForms/ManageIQ-specific customisations are read from files in the directory. The file */etc/manageiq/postgresql.conf.d/01_miq_overrides.conf* contains the default CloudForms-specific PostgreSQL settings, however as this file is potentially overwritten on a **yum update**, any user-custom settings should be added to a new file in the include directory such as */etc/manageiq/postgresql.conf.d/02_user_overrides.conf*.

NOTE

CFME appliances upgraded from a version prior to 5.9 will retain all PostgreSQL configuration in the main */var/opt/rh/rh-postgresql95/lib/pgsql/data/postgresql.conf* file as before. This will not make reference to the include directory, and any of the parameters changes described below should be made to this file.

4.1.3.1.1. Shared Buffers

The most important tuning parameter to set in a user-custom overrides file is the value for **`shared_buffers`**. A default value for **`shared_buffers`** is set automatically using **`ALTER SYSTEM`** depending on whether the database server is a standalone "dedicated" database appliance or a "shared" appliance that also runs *evmserverd*. For a standalone database server this is 1GB; for a shared database server this is 128MB.

For a dedicated PostgreSQL server in a large region the value of **`shared_buffers`** 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.

To change the value for **`shared_buffers`** the existing default value should be reset using the **`ALTER SYSTEM RESET shared_buffers`** command from psql as the postgres user, for example:

```
# su - postgres
Last login: Fri Jun 29 09:18:07 EDT 2018 on pts/0
-bash-4.2$ psql -d vmdb_production
psql (9.5.9)
Type "help" for help.

vmdb_production=# ALTER SYSTEM RESET shared_buffers;
ALTER SYSTEM
vmdb_production=# \q
```

Now the updated value can be added to a new */etc/manageiq/postgresql.conf.d/02_user_overrides.conf* file, as follows:

```
shared_buffers = 2GB        # 25% of physical memory
```

4.1.3.1.2. 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.

4.1.3.1.3. 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*. The following commands can be used to setup an alternative directory for the log file such as */var/log/pg_log*.

```
mkdir -p /var/log/pg_log
chown postgres:postgres /var/log/pg_log
```

Change the **log_directory** line in */var/opt/rh/rh-postgresql95/lib/pgsql/data/postgresql.conf*:

```
log_directory = '/var/log/pg_log'
```

Restart PostgreSQL:

```
systemctl restart rh-postgresql95-postgresql.service
```

The file */etc/logrotate.d/miq_logs.conf* should be edited to reflect the new PostgreSQL log directory so that logs will be rotated correctly:

```
/var/www/miq/vmdb/log/*.log /var/www/miq/vmdb/log/apache/*.log ⏎
        /var/log/pg_log/*.log /var/log/tower/*.log {
  daily
  dateext
  missingok
  rotate 14
  notifempty
  compress
  copytruncate
```

```
  prerotate
    source /etc/default/evm; /bin/sh ⏎
       ${APPLIANCE_SOURCE_DIRECTORY}/logrotate_free_space_check.sh $1
  endscript
  lastaction
    /sbin/service httpd reload > /dev/null 2>&1 || true
  endscript
}
```

4.1.3.1.4. 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-postgresql96.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.

4.1.4. 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.[9]

On a CFME 5.9.0 & 5.9.1 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
- event_streams

NOTE

From CFME 5.9.2 onwards these scripts are automatically run by the appliance with the *database maintenance* role configured. No manual configuration from **appliance_console** is necessary.

4.1.5. 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 **`evmserverd`** 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
```

4.2. POSTGRESQL RUNNING IN A CONTAINER

The following sections describe the tuning that can be performed if CloudForms is to run podified in OpenShift Container Platform.

4.2.1. Sizing and Configuring the Database Pod

The PostgreSQL pod should be sized according to the anticipated scale of the CloudForms deployment. The following install-time parameters can be overridden if required:

Table 4.1. Install-time PostgreSQL Parameters

Parameter	Default Value	Description
POSTGRESQL_CPU_REQ	500m	Minimum amount of CPU time the PostgreSQL container will need (expressed in millicores)
POSTGRESQL_MEM_REQ	4Gi	Minimum amount of memory the PostgreSQL container will need
POSTGRESQL_MEM_LIMIT	8Gi	Maximum amount of memory the PostgreSQL container can consume
DATABASE_VOLUME_CAPACITY	15Gi	Volume space available for database
POSTGRESQL_MAX_CONNECTIONS	1000	PostgreSQL maximum number of database connections allowed
POSTGRESQL_SHARED_BUFFERS	1GB	Amount of memory dedicated for PostgreSQL shared memory buffers

For example if deploying CloudForms to manage larger OpenShift Container Platform installations it may be desirable to increase the size of database pod and the database volume, and this can be done using the following parameters for the **oc new-app** command:

```
oc new-app --template=cloudforms \
-p
DATABASE_VOLUME_CAPACITY=100Gi,POSTGRESQL_MEM_LIMIT=12Gi,POSTGRESQL_SHARED
_BUFFERS=3Gi
```

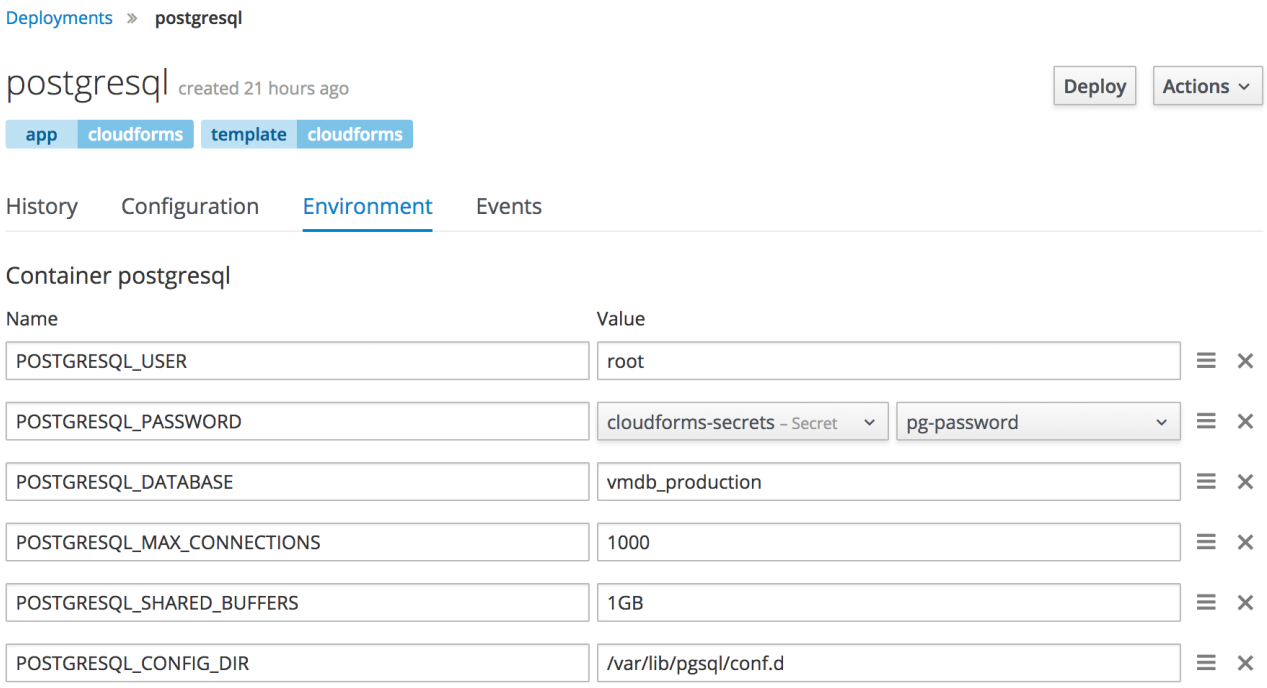
4.2.1.1. Sizing the Database Persistent Volume Before Installation

The disk used for the database persistent volume 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.

4.2.1.2. Configuring PostgreSQL

The configuration settings **shared_buffers** and **max_connections** described in Section 4.1.3.1, “Configuring PostgreSQL” are also relevant for the podified installation of PostgreSQL. Rather than directly editing the PostgreSQL configuration file however, these should either be defined as template parameters (see Table 4.1, “Install-time PostgreSQL Parameters”) when the application is installed, or changed after installation by editing the environment variables for the **postgresql** deployment (see Figure 4.1, “postgresql Deployment Environment Variables”).

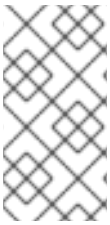
Figure 4.1. postgresql Deployment Environment Variables



4.2.2. Resizing the Database Persistent Volume After Installation

It is sometimes the case that an OpenShift Container Platform cluster grows at a faster rate than anticipated. As a result the CloudForms database persistent volume 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 **`evmserverd`** 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
```

[9] 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 5.9 the OpenShift Container Manager provider always performs a full refresh.

5.1. REFRESH OVERVIEW

Whenever CloudForms is notified of a change related to a managed object, a message is queued for 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.

5.2. CHALLENGES OF SCALE

As might be expected, the more managed objects in an OpenShift Container Platform cluster, the longer a full refresh takes to complete. Tests on an OpenShift Container Platform cluster containing 3000 pods (each with 1 container) and 3000 images have shown that a full refresh of a cluster of this size takes between 50 and 70 seconds, depending on master node and VMDB database activity.

The refresh time has a knock-on effect for the process or workflow that initiated the refresh. In most cases this is inconvenient but not critical, such as a delay in seeing a container's status change in the WebUI. In other cases however - particularly when using an automate workflow - a very long EMS refresh may cause the triggering workflow to timeout and exit with an error condition.

5.3. MONITORING REFRESH PERFORMANCE

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

- Extracting and parsing the data from OpenShift Container Platform
  - Network latency to the OpenShift Container Platform Master node
  - Time waiting for the Master node 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. A typical log line is as follows:

```
  ...
MIQ(ManageIQ::Providers::Openshift::ContainerManager::Refresher#refresh) ⏎
```

```
 EMS: [OpenShift], id: [1000000000004] Refreshing targets for
EMS...Complete ⏎
 - Timings { ⏎
 :collect_inventory_for_targets=>4.851187705993652, ⏎
 :parse_targeted_inventory=>4.859905481338501, ⏎
 :save_inventory=>5.751120328903198, ⏎
 :manager_refresh_post_processing=>1.8835067749023438e-05, ⏎
 :ems_refresh=>15.463248014450073}
```

The timing values are described as follows:

- **:collect_inventory_for_targets** - the time taken to extract the inventory from the OpenShift Container Platform master node.
- **:parse_targeted_inventory** - the time taken to parse the inventory data
- **:save_inventory** - the time taken to save or update the inventory into the VMDB
- **:manager_refresh_post_processing** - the time taken to batch process some post-processing actions
- **:ems_refresh** - the total time to perform the refresh

Extracting the timings[10] from the log line shown above reveals the following performance values:

```
Refresh timings:
  collect_inventory_for_targets:       4.851188 seconds
  parse_targeted_inventory:            4.859905 seconds
  save_inventory:                      5.751120 seconds
  manager_refresh_post_processing:     0.000019 seconds
  ems_refresh:                         15.463248 seconds
```

This shows that two of the significant time components to this operation were extracting and parsing the inventory from the OpenShift Container Platform Master node (4.85 seconds), and loading the data into the database (5.75 seconds).

5.4. IDENTIFYING REFRESH PROBLEMS

Refresh problems are best identified by establishing baseline timings when the managed OpenShift Container Platform cluster is least busy. To determine the relative data collection and database load times, the ':collect_inventory_for_targets' and ':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 collect_inventory_for_targets ems_refresh_timings.out | ⏎
awk '{print $2}' > parse_targeted_inventory.txt

grep save_inventory ems_refresh_timings.out | ⏎
awk '{print $2}' > save_inventory.txt
```

A significant increase or wide variation in data extraction times from this baseline can indicate that the master node 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:
...
  :openshift:
    :refresh_interval: 15.minutes
    :inventory_object_refresh: true
    :inventory_collections:
      :saver_strategy: :batch
    :get_container_images: true
    :store_unused_images: true
```

Most of the values are not intended to be user-customisable, but the following two settings can be changed to **false** if required to improve performance.

5.5.1.1. Get Container Images

The **:get_container_images** value defines whether or not container images should be included in the inventory collection. If an OpenShift Container Platform installation has many thousands of container images the inventory refresh can take a long time, and performance can be improved by not collecting inventory data on the images.

5.5.1.2. Store Unused Images

The **:store_unused_images** value defines whether to save metadata - for example labels - on all container images (the default), or only for "used" images that are referenced by active pods.

---

[10] Example scripts to extract the timings from evm.log are available from https://github.com/RHsyseng/cfme-log-parsing

CHAPTER 6. CAPACITY & UTILIZATION

CloudForms 4.6 can retrieve Capacity & Utililization (C&U) metrics from either Hawkular or Prometheus endpoints in an OpenShift Container Platform environment. Metrics are retrieved for nodes, pods and containers.

NOTE

C&U metrics collection of an OpenShift Container Platform cluster should not be deployed/enabled on CloudForms Management Engine (CFME) 5.9.0 or 5.9.1 if Hawkular is used as the OpenShift Container Platform metrics endpoint.

C&U metrics collection from Hawkular can be enabled on CFME 5.9.2 or later, but the **`:hawkular_force_legacy:`** value should be set to **`true`** in **Configuration → Advanced** settings, i.e.

```
:queue_worker_base:
  :ems_metrics_collector_worker:
  ...
    :ems_metrics_collector_worker_kubernetes:
    ...
      :hawkular_force_legacy: true
```

It should also be noted that Prometheus is supplied as a Technology Preview feature in OpenShift Container Platform 3.7 and 3.9, and that interaction with Prometheus for metrics collection is also a CloudForms 4.6 Technology Preview feature.

6.1. COMPONENT PARTS

As discussed in Chapter 2, *CloudForms 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[11]. The Coordinator schedules the data collections for the VMDB objects that support metrics collection, and queues messages for the C&U Data Collector to retrieve metrics for any objects for which a collection is due. For an OpenShift Container Platform provider, metrics are retrieved for nodes, pods and containers.

The time interval between collections is defined by the **`:capture_threshold`** setting in 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
    :container: 50.minutes
    :container_group: 50.minutes
    :container_node: 50.minutes
  :capture_threshold_with_alerts:
    :default: 1.minutes
    :host: 20.minutes
    :vm: 20.minutes
```

The intention is that no metric is ever older than the **`:capture_threshold`** value for its object type. The default capture_threshold for OpenShift Container Platform pods, containers & nodes is 50 minutes. On average therefore messages for approximately 6% (100 / (50/3)) of the total number of these types of managed objects are created every 3 minutes.

As an example, if CloudForms is managing an OpenShift Container Platform cluster containing 1000 pods, 1000 containers and 20 nodes, there will be approximately 2020/16.666 or 121 messages created every 3 minutes. In practice the number varies slightly at each C&U Coordinator run, and for this example might vary between approximately 110 and 130 as greater or fewer objects fall outside of the **`:capture_threshold`** timeout value.

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 Hawkular or Prometheus (depending on configuration setting for the provider) to retrieve the data for the object and time range specified in the message.

A successful capture is written to *evm.log*, as follows:

```
Capture for
ManageIQ::Providers::Kubernetes::ContainerManager::ContainerNode ⏎
name: [master1.cloud.example.com], ⏎
id: [1000000000019], ⏎
start_time: [2018-01-23 08:04:00 UTC]...Complete ⏎
- Timings: { ⏎
:capture_state=>0.014913082122802734, ⏎
:collect_data=>1.4922049045562744, ⏎
:total_time=>1.8736591339111328}
```

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:

```
Processing for
ManageIQ::Providers::Kubernetes::ContainerManager::ContainerNode ⏎
name: [master1.cloud.example.com], ⏎
id: [1000000000019], ⏎
for range [2018-01-23 08:03:00 UTC - 2018-01-23 09:42:00 UTC]...Complete ⏎
- Timings: { ⏎
:process_counter_values=>0.0027306079864501953, ⏎
:get_attributes=>0.001895904541015625, ⏎
:db_find_prev_perfs=>0.008209705352783203, ⏎
:preload_vim_performance_state_for_ts=>0.0017957687377929688, ⏎
:process_perfs=>0.01605057716369629, ⏎
:process_build_ics=>0.003462076187133789, ⏎
:process_perfs_db=>0.11580920219421387, ⏎
:total_time=>0.16924238204956055}
```

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 containers are aggregated into the counters for their parent objects. For example for an OpenShift Container Platform provider the parent rollup process would include the following objects:

- Active & recently deleted Pods {hourly,daily} → Project
- Active Pods {hourly,daily} → Replicator
- Active Pods {hourly,daily} → Service
- Node {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:

```
Rollup for
ManageIQ::Providers::Kubernetes::ContainerManager::ContainerNode ⏎
name: [master1.cloud.example.com], ⏎
id: [1000000000019] ⏎
for time: [2018-01-23T09:00:00Z]...Complete ⏎
- Timings: { ⏎
:db_find_prev_perf=>0.0028629302978515625, ⏎
:rollup_perfs=>0.018657922744750977, ⏎
:db_update_perf=>0.005361080169677734, ⏎
:process_bottleneck=>0.0017886161804199219, ⏎
:total_time=>0.037772417068481445}
```

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 Hawkular or Prometheus
  - Network latency to the Hawkular or Prometheus pod
  - Time waiting for Hawkular or Prometheus to process the request 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 C&U capture and initial processing can be parsed using a script such as perf_process_timings.rb.[12], for example:

```
Capture timings:
  capture_state:                        0.018490 seconds
  collect_data:                         1.988341 seconds
  total_time:                           2.043008 seconds
```

```
Process timings:
  process_counter_values:                0.002075 seconds
  get_attributes:                        0.000989 seconds
  db_find_prev_perfs:                    0.009578 seconds
  preload_vim_performance_state_for_ts: 0.002601 seconds
  process_perfs:                         0.014115 seconds
  process_build_ics:                     0.002263 seconds
  process_perfs_db:                      0.049842 seconds
  total_time:                            0.120220 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:                   0.017392 seconds
  rollup_perfs:                        0.149081 seconds
  db_update_perf:                      0.072613 seconds
  process_operating_ranges:            0.081201 seconds
  total_time:                          0.320613 seconds
```

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: [10000000032703], 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

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 [OCP 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 [OCP 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 [OCP 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. Long dequeue times (over 600 seconds) indicate that the number of C&U Data Collectors should be increased.

```
... MIQ(ManageIQ::Providers::Openshift::ContainerManager:: ⏎
MetricsCollectorWorker::Runner#get_message_via_drb) ⏎
Message id: [1476318], MiqWorker id: [2165], Zone: [default], ⏎
Role: [ems_metrics_collector], Server: [], Ident: [openshift], ⏎
Target id: [], Instance id: [1475], Task id: [], Command:
[ManageIQ::Providers::Kubernetes::ContainerManager:: ⏎
Container.perf_capture_realtime], Timeout: [600], Priority: [100], ⏎
State: [dequeue], Deliver On: [], Data: [], Args: [], ⏎
Dequeued in: [1576.125461466] seconds
```

6.5.2.3. Missing Data Samples

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

```
... WARN -- : MIQ(ManageIQ::Providers::Kubernetes::ContainerManager:: ⏎
ContainerGroup#perf_capture) [realtime] For ⏎
ManageIQ::Providers::Kubernetes::ContainerManager::ContainerGroup ⏎
name: [jenkins-1], id: [2174], start_time: [2017-08-11 00:00:00 UTC], ⏎
expected to get data as of [2017-08-11T00:00:20Z], but got data as  ⏎
of [2017-08-11 14:28:20 UTC]
```

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:

```
... MIQ(MiqEmsMetricsProcessorWorker::Runner#get_message_via_drb) ⏎
Message id: [1000003602714], MiqWorker id: [1000000003176], Zone:
[default], ⏎
Role: [ems_metrics_processor], Server: [], Ident: [ems_metrics_processor],
⏎
Target id: [], Instance id: [1000000000141], Task id: [], ⏎
```

```
Command: [ContainerService.perf_rollup], Timeout: [1800], Priority: [100],
⏎
State: [dequeue], Deliver On: [2018-02-15 10:00:00 UTC], Data: [],  ⏎
Args: ["2018-02-15T09:00:00Z", "hourly"], Dequeued in: [1.982437188]
seconds

... INFO -- : MIQ(MiqQueue#delivered) Message id: [1000003602714], ⏎
State: [ok], Delivered in [0.042801554] 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 active OpenShift Container Platform entity such as service, pod or container. There should also be at most 5 of the metrics_## tables that contain more than zero records.

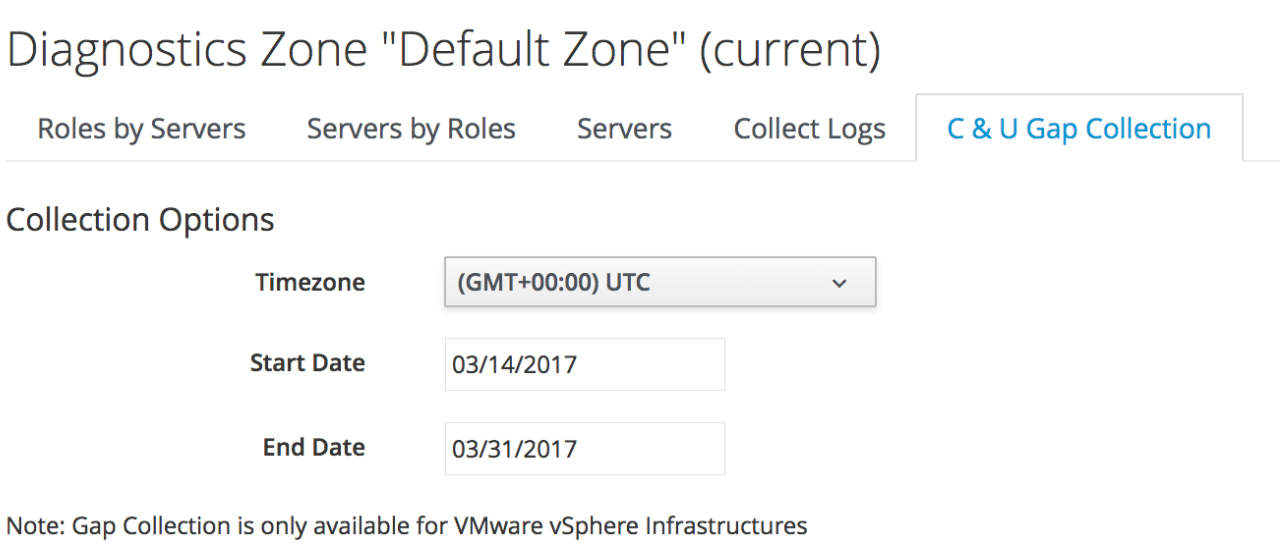
The following SQL query can be used to confirm that the records are being processed correctly:

```
select resource_id, date_trunc('day',timestamp) as collect_date, ⏎
resource_type, capture_interval_name, count(*)
from metric_rollups
where resource_type like '%Container%'
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
-----+---------------------+---------------------+----------------+----
---
...
  89 | 2018-01-26 00:00:00 | ContainerService    | daily          |     1
  89 | 2018-01-26 00:00:00 | ContainerService    | hourly         |    24
  89 | 2018-01-27 00:00:00 | ContainerReplicator | daily          |     1
  89 | 2018-01-27 00:00:00 | ContainerReplicator | hourly         |    24
...
2050 | 2018-02-01 00:00:00 | ContainerGroup      | daily          |     1
2050 | 2018-02-01 00:00:00 | ContainerGroup      | hourly         |    24
2050 | 2018-02-02 00:00:00 | ContainerGroup      | daily          |     1
2050 | 2018-02-02 00:00:00 | ContainerGroup      | hourly         |    24
...
2332 | 2018-01-29 00:00:00 | Container           | daily          |     1
2332 | 2018-01-29 00:00:00 | Container           | hourly         |    24
2332 | 2018-01-30 00:00:00 | Container           | daily          |     1
2332 | 2018-01-30 00:00:00 | Container           | 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 6.1, “C&U Gap Collection”), but this is of a lower granularity than the realtime metrics that are usually collected. Although gap collection is intended for use with VMware providers, it also works in a more limited capacity with the OpenShift Container Platform provider.

Figure 6.1. C&U Gap Collection



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 run a `/usr/bin/hourly_reindex_metrics_tables` script that reindexes one of the tables every hour.

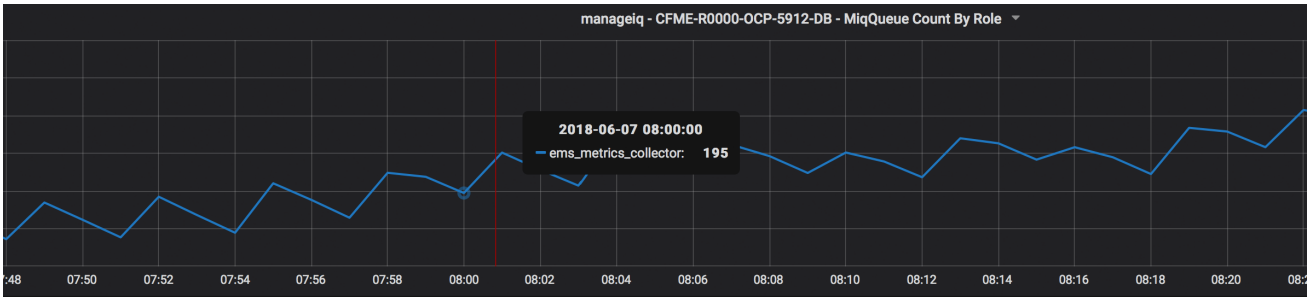
6.7.2.1. Increasing the Number of Data Collectors

Ideally each new batch of ems_metrics_collector messages should be fully processed in the 3 minute time window before the next batch of messages is created. If the ems_metrics_collector message queue length were plotted against time, it should look like a saw tooth with a sudden rise in the number of queued messages every 3 minutes, followed by a gradual decline down to zero as the data collectors dequeue and process the messages (see Figure 6.2, “ems_metrics_collector Message Queue Length Stable over Time”)

Figure 6.2. ems_metrics_collector Message Queue Length Stable over Time

If the number of C&U Data Collector workers can't keep up with the rate that messages are added to the queue, the queue length will just rise indefinitely (see Figure 6.2, "ems_metrics_collector Message Queue Length Stable over Time")

Figure 6.3. ems_metrics_collector Message Queue Length Increasing over Time



If the ems_metrics_collector message queue length is steadily increasing, the number of C&U Data Collector workers should be increased. The default number of workers per appliance or pod is 2, but this can be increased up to a maximum of 9. Consideration should be given to the additional CPU and memory requirements that an increased number of workers will place on the appliance or pod. It may be more appropriate to add further appliances or *cloudforms-backend* StatefulSet replicas 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 many thousands of objects) may benefit from dedicated CFME appliances in the provider zones exclusively running the C&U data collector role.

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 or "cloudforms-backend" StatefulSet replicas 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 or pods, as both are resource intensive. CloudForms installations managing several thousand objects may benefit from dedicated CFME appliances or "cloudforms-backend" StatefulSet replicas in the provider zones exclusively running the C&U Data Processor role.

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

[12] From https://github.com/RHsyseng/cfme-log-parsing

CHAPTER 7. KUBERNETES EVENT HANDLING

If a CloudForms installation is to effectively manage an external management system such as an OpenShift Container Platform cluster, it must be able to efficiently detect and process any events from the EMS. This section discusses the process of handling events from Kubernetes.

7.1. EVENT PROCESSING WORKFLOW

The event processing workflow involves 3 different worker types, as follows:

1. The cluster-specific ManageIQ::Providers::Openshift::ContainerManager::EventCatcher worker polls Kubernetes for new events using an API call (see Section 7.3, "Event Catcher Configuration" 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 Kubernetes-specific references are translated into the equivalent CloudForms object ID, 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:

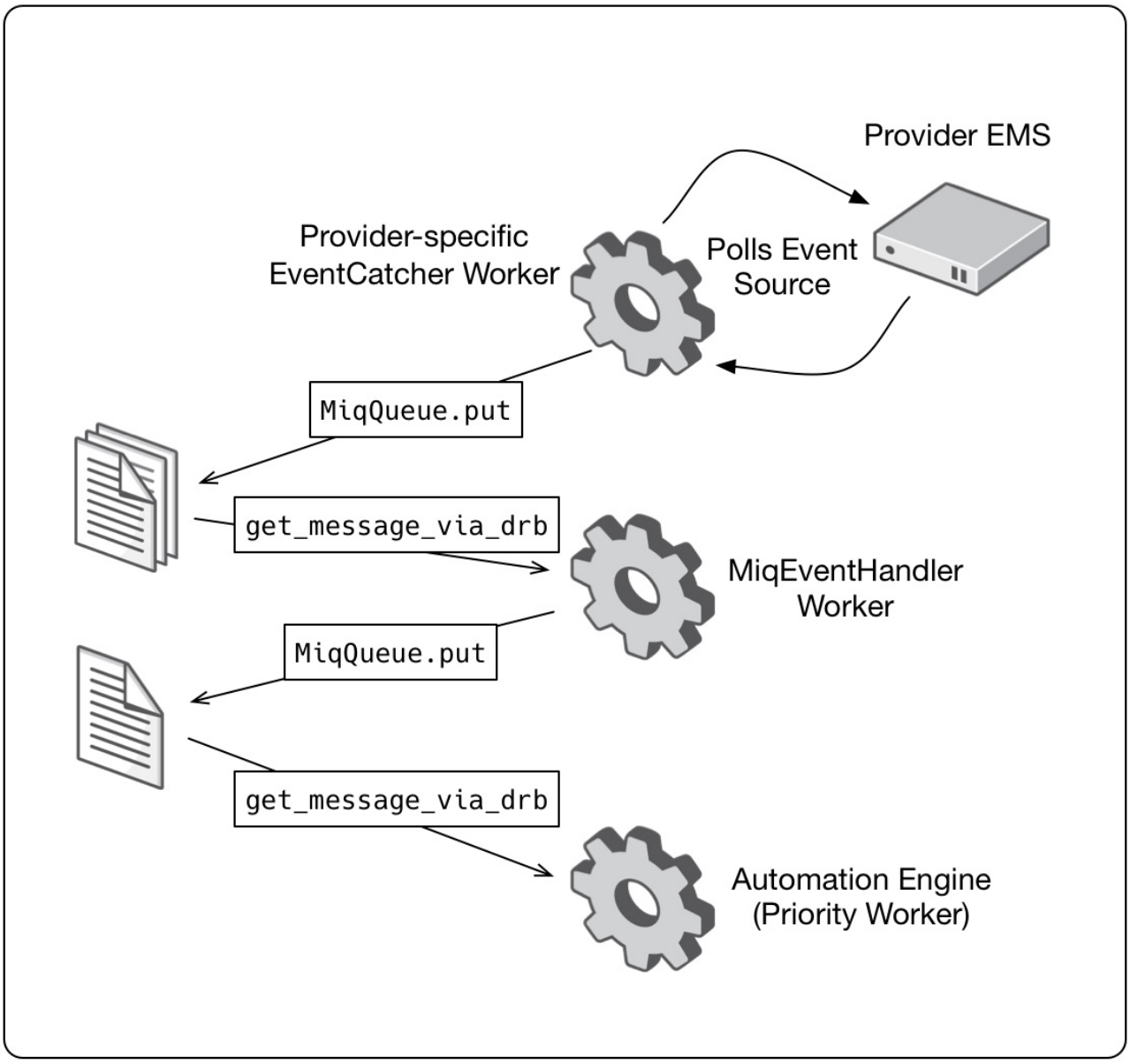
    - Triggering a provider refresh

    - Initiate any further CloudForms-specific operations that are required after the event, such as raising internal *Policy Events*

    - Perform any custom event-related processing (see Section 7.4, "Extending Event Handling Using Automate")

The event workflow is illustrated in Figure 7.1, "Event Processing Workflow"

Figure 7.1. Event Processing Workflow



7.2. EVENT TYPES

CloudForms 4.6 detects and processes the following event types from Kubernetes.

7.2.1. Node-related events

- NODE_FAILEDMOUNT
- NODE_INVALIDDISKCAPACITY
- NODE_NODENOTREADY
- NODE_NODENOTSCHEDULABLE
- NODE_NODEREADY
- NODE_NODESCHEDULABLE

- NODE_REBOOTED

7.2.2. Pod-related events

- POD_CREATED
- POD_DEADLINEEXCEEDED
- POD_FAILED
- POD_FAILEDSCHEDULING
- POD_FAILEDVALIDATION
- POD_HOSTPORTCONFLICT
- POD_INSUFFICIENTFREECPU
- POD_INSUFFICIENTFREEMEMORY
- POD_KILLING
- POD_NODESELECTORMISMATCHING
- POD_OUTOFDISK
- POD_SCHEDULED
- POD_STARTED
- POD_STOPPED
- POD_UNHEALTHY

7.2.3. Replicator-related events

- REPLICATOR_FAILEDCREATE
- REPLICATOR_SUCCESSFULCREATE

7.3. 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. The following extract shows the settings for the *ManageIQ::Providers::Openshift::ContainerManager::EventCatcher* worker, and the default settings for all EventCatcher workers:

```
    :event_catcher:
...
      :event_catcher_openshift:
        :poll: 1.seconds
...
      :defaults:
        :flooding_events_per_minute: 30
        :flooding_monitor_enabled: false
        :ems_event_page_size: 100
```

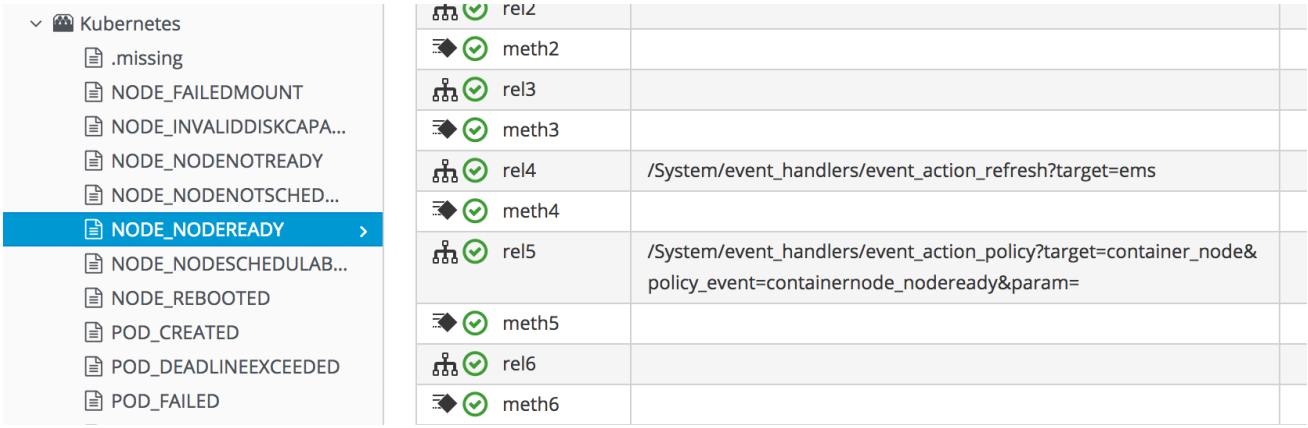
```
        :ems_event_thread_shutdown_timeout: 10.seconds
        :memory_threshold: 2.gigabytes
        :nice_delta: 1
        :poll: 1.seconds
```

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

7.4. EXTENDING EVENT HANDLING USING AUTOMATE

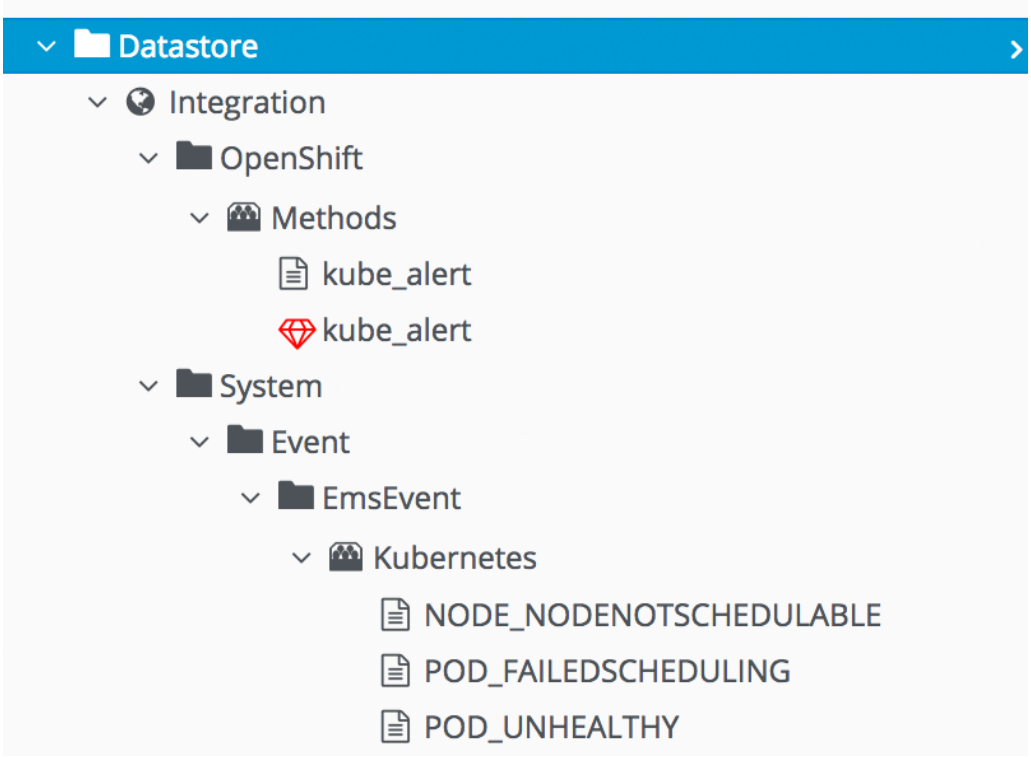
The automate instances that handle the processing of Kubentes events through the event switchboard are located under the */System/Event/EmsEvent/Kubernetes* class in the *ManageIQ* domain of the automate datastore. A typical instance calls two event handler methods via relationships, the first to trigger a provider refresh, and the second to raise a CloudForms *Policy Event* (see Figure 7.2, "Kubernetes Event Instance")

Figure 7.2. Kubernetes Event Instance



This default processing can easily be extended by calling one or more further relationships to perform any custom event processing that might be required, for example sending a warning email, or forwarding the event details via REST API to a separate monitoring tool. Figure 7.3, "Custom Event Handling Automate Domain" shows the creation of a new *Integration* automate domain, into which three of the event-handling instances have been copied.

Figure 7.3. Custom Event Handling Automate Domain



These three event instances have each been modified to call the */OpenShift/Methods/kube_alert* instance from their **rel6** relationship. The *kube_alert* instance in this example defines two attributes: **to_email_address** and **from_email_address**, and then calls the *kube_alert* method.

7.4.1. Automate Method

The following *kube_alert* automate method is a simple example illustrating how the event details could be emailed to a named recipient (defined in the instance schema, and retrieved using **`$evm.object['to_email_address']`**). The event-specific parameters such as pod, container or node are contained in the **`$evm.root['event_stream']`** object.

```
cluster    = $evm.vmdb(:ems, $evm.root['event_stream'].ems_id).name

project    = $evm.root['event_stream'].container_namespace || "N/A"
pod        = $evm.root['event_stream'].container_group_name || "N/A"
container  = $evm.root['event_stream'].container_name || "N/A"
event_type = $evm.root['event_stream'].event_type
message    = $evm.root['event_stream'].message

to      = $evm.object['to_email_address']
from    = $evm.object['from_email_address']
subject = "#{event_type} event received from cluster #{cluster}"

body = "A #{event_type} event was received from cluster #{cluster}<br>
<br>"
body += "Project: #{project}<br>"
body += "Pod: #{pod}<br>"
body += "Container: #{container}<br>"
```

```
body += "Message: #{message}"

$evm.execute('send_email', to, from, subject, body)
```

7.4.1.1. Example Emails

For a node-related event the **event_stream** object does not contain any project, pod or container entries. An example email from the method would have the following message body:

```
A NODE_NODENOTSCHEDULABLE event was received from cluster OpenShift Prod

Project: N/A
Pod: N/A
Container: N/A
Message: Node node2.cloud.example.com status is now: NodeNotSchedulable
```

For a pod-related event the **event_stream** object does contain the related project, pod and container values. An example email from the method would have the following message body:

```
A POD_UNHEALTHY event was received from cluster OpenShift Prod

Project: default
Pod: registry-console-1-wcwx4
Container: registry-console
Message: Readiness probe failed: Get http://10.1.2.3:9090/ping: ⏎
dial tcp 10.1.2.3:9090: getsockopt: connection refused
```

These examples show how CloudForms can behave as a Kubernetes event broker, and forward event-specific details to a further monitoring, alerting or ticketing system via email, SNMP trap or API call.

7.5. SCALING OUT

The event processing workflow can be quite resource-intensive when handling events from several possibly large OpenShift Container Platform clusters. CloudForms installations managing several thousand objects may benefit from dedicated CFME appliances exclusively running the *ManageIQ::Providers::Openshift::ContainerManager::EventCatcher* workers and *MiqEventHandler* worker in any zone containing an OpenShift Container Platform provider.

CHAPTER 8. SMARTSTATE ANALYSIS AND OPENSCAP COMPLIANCE CHECKING

SmartState Analysis allows CloudForms to perform a deep introspection of OpenShift Container Platform container images to discover their contents and (for RHEL-based images) determine their OpenSCAP compliance.

NOTE

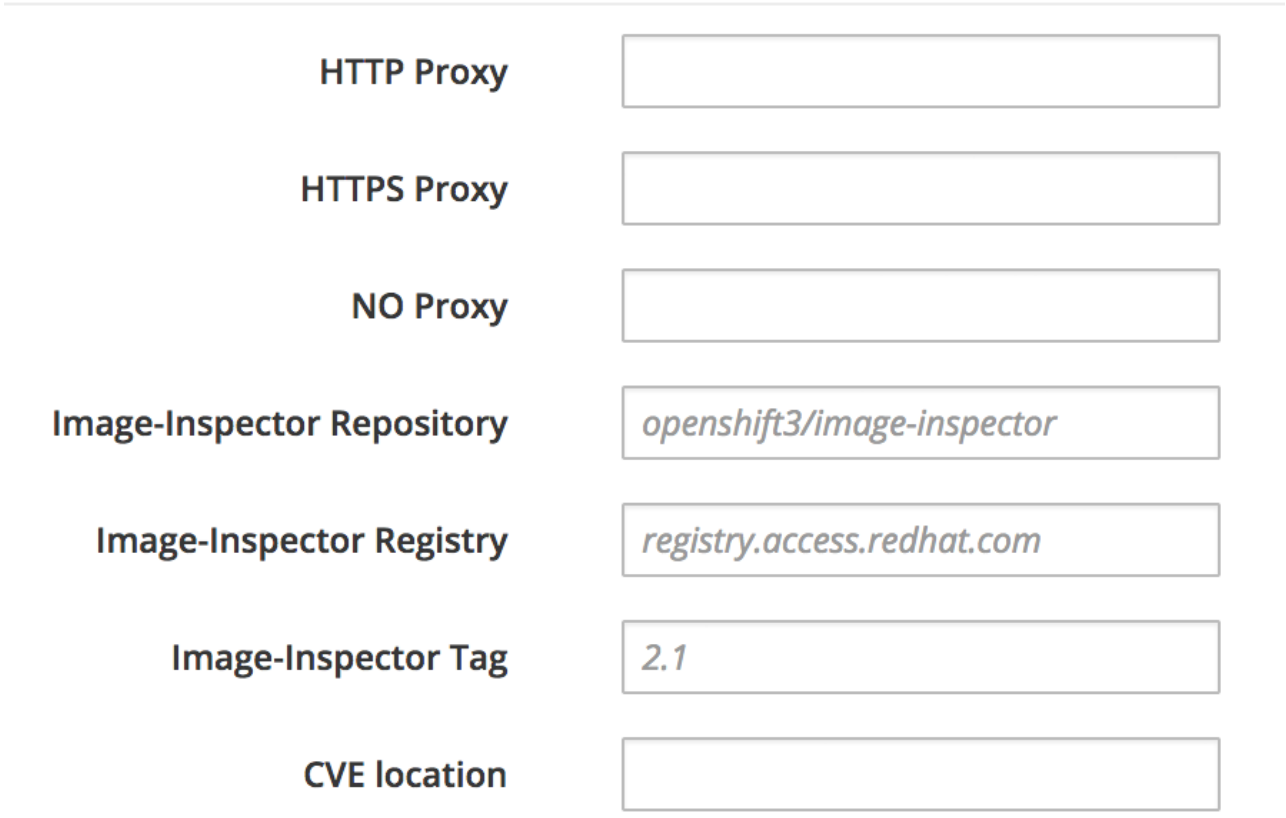
SmartState Analysis is alternatively known as "fleecing"

SmartState Analysis of OpenShift Container Platform images uses the functionality of two CloudForms server roles, and an image inspector pod that runs in OpenShift. The first server role - SmartState Analysis - is performed by a Generic or Priority worker, depending on message priority. The second server role - SmartProxy - starts the embedded or *coresident*[13] *MiqSmartProxyWorker* processes.

SmartState scanning of OpenShift Container Platform images is performed by an *image-inspector* pod that is pulled from a registry as required. The image-inspector dynamically downloads and uses the latest CVE definition/rules file from Red Hat before scanning RHEL-based images.

In CloudForms 4.6 the image-inspector repository, registry and version tag are optionally configurable in the provider's **Advanced Settings** page. The proxy settings and CVE location that the image-inspector pod uses to download the rules file is also configurable here (see Figure 8.1, "Provider Advanced Settings")

Figure 8.1. Provider Advanced Settings



8.1. SMARTSTATE ANALYSIS STEPS

Performing a SmartState Analysis of a container image involves the following steps:

1. The SmartState Analysis worker calls the OpenShift Container Platform API to launch an image-inspector pod, and injects a command such as:

```
"/usr/bin/image-inspector",
"--chroot",
"--image=<registry>/<repository>/<image_name>@sha256:...",
"--scan-type=openscap",
"--serve=0.0.0.0:8080",
"--dockercfg=/var/run/secrets/kubernetes.io/ ⏎
inspector-admin-secret-inspector-admin-dockercfg-42mjt/.dockercfg"
```

The --image switch tells the image-inspector which image to scan. The SmartState Analysis worker then waits for the resulting *manageiq-img-scan-<id>* pod to complete.

2. The image-inspector fetches the target (**`--image`**) image and extracts it to a temporary directory such as **`/var/tmp/image-inspector-084630810`**.

3. If the scanned image is based on RHEL 5, 6,or 7, the image-inspector container retrieves the version-specific OpenSCAP CVE file such as *com.redhat.rhsa-RHEL7.ds.xml.bz2* from the location specified in the advanced configuration (see Figure 8.1, “Provider Advanced Settings”). If not overridden the default location for the CVE file is *www.redhat.com/security/data/metrics/ds/*.

4. For a RHEL-based image the image-inspector container performs a CVE scan of the contents of the temporary directory and writes the results to another temporary directory such as **`/var/tmp/image-inspector-scan-results-689112497`**.

5. The image-inspector container serves the metadata and OpenSCAP results over WebDAV, on the socket specified by the **`--serve`** argument, for example **`webdav://0.0.0.0:8080/api/v1/content/`**.

6. The SmartState Analysis worker queues a message for the SmartProxy worker to scan the metadata with a command line argument list similar to the following:

```
args=["<repository>/<image_name>", ⏎
"--- ⏎
:pod_namespace: management-infra ⏎
:pod_name: manageiq-img-scan-6db1d ⏎
:pod_port: 8080 ⏎
:guest_os: Linux\n"], ⏎
method_name="scan_metadata", ⏎
vm_guid="<registry>/<repository>/<image_name>", ⏎
category="system,software", ⏎
taskid="<task_id_assigned_to_scan>", ⏎
target_id=<container_image_id>, ⏎
target_type="ContainerImage"
```

7. The SmartProxy worker connects to the WebDAV URI and retrieves the extracted pod package list, and if available the OpenSCAP scan results. It stores these in the VMDB with the data model for the container image.

8. The SmartState Analysis worker calls the OpenShift Container Platform API to delete the image-inspector pod.

9. If the *OpenSCAP profile* compliance policy profile has been added to the OpenShift Container Platform provider, the image's compliance is evaluated. If the image is deemed non-compliant it is tagged with the following annotations:

```
images.openshift.io/deny-execution=true
security.manageiq.org/failed-policy=openscap policy
```

8.2. MONITORING SMARTSTATE ANALYSIS

The total time for each image scan can be determined from the time duration between the "request_containerimage_scan" and corresponding "containerimage_scan_complete" events being processed through automate, as follows:

```
... INFO -- : MIQ(MiqAeEngine.deliver) Delivering ⏎
{:event_type=>:request_containerimage_scan, ⏎
"MiqEvent::miq_event"=>1000001206765, :miq_event_id=>1000001206765, ⏎
"EventStream::event_stream"=>1000001206765, ⏎
:event_stream_id=>1000001206765} for object ⏎
[ContainerImage.1000000000671] with state [] to Automate

...

... INFO -- : MIQ(MiqAeEngine.deliver) Delivering ⏎
{:event_type=>"containerimage_scan_complete", ⏎
"MiqEvent::miq_event"=>1000001206774, :miq_event_id=>1000001206774, ⏎
"EventStream::event_stream"=>1000001206774, ⏎
:event_stream_id=>1000001206774} for object ⏎
[ContainerImage.1000000000671] with state [] to Automate
```

This time includes the entire scan sequence of tasks, including image-inspector pod launch, and the time for it to pull the target container image from the registry.

8.3. CHALLENGES OF SCALE

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

8.3.1. Identifying SmartState Analysis Problems

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

```
grep ':abort_job' evm.log
```

8.3.1.1. Permissions-Related Problems

The SmartState-related activities that run in OpenShift Container Platform are performed as the **`system:serviceaccount:management-infra:management-admin`** account. If this account doesn't exist or is setup incorrectly then SmartState Analysis will not complete successfully.

8.3.1.1.1. Failing to Launch the image-inspector Pod

If the account does not have permission to create pods in the *management-infra* project, an error similar to the following will be seen in *evm.log*:

```
pod creation for [management-infra/manageiq-img-scan-16d48] failed: ⏎
[HTTP status code 403, User "system:serviceaccount:management-
infra:management-admin" ⏎
cannot create pods in project "management-infra"]
```

8.3.1.1.2. Failing to annotate the image

If the account does not have permission to annotate images in the container image's project, an error similar to the following will be seen in *evm.log*:

```
Container Image "cotd/master": Error during 'Check Compliance': ⏎
User "system:serviceaccount:management-infra:management-admin" cannot
"patch" "images" ⏎
with name
"sha256:57978b86281fdb4a09869a3aa879200ed1d5004b295e7421c2f603829e96ac73"
⏎
in project ""
```

8.3.1.2. Failing to Download the image-inspector Container Image

The image-inspector container image is downloaded from the *registry.access.redhat.com* registry by default. If the OpenShift Container Platform cluster nodes do not have connectivity to this registry then the pod will fail to deploy. An error similar to the following will be seen in *evm.log*:

```
:event_type=>"CONTAINER_FAILED", :source=>"KUBERNETES", :timestamp=>"2017-
12-14T16:18:45Z", ⏎
:message=>"Failed to pull image
\"registry.access.redhat.com/openshift3/image-inspector:2.1\"
```

An alternative registry can be defined in the advanced settings for the provider (see Figure 8.1, "Provider Advanced Settings")

8.3.1.3. Failing to Download the OpenSCAP CVE file

By default the OpenSCAP CVE file is downloaded from *https://www.redhat.com/security/data/metrics/ds/*. If the image-inspector pod does not have direct Internet connectivity, the download will timeout and an error similar to the following will be seen in *evm.log*:

```
job finished, Unable to run OpenSCAP: Unable to retreive the CVE file: ⏎
Could not download file https://www.redhat.com/security/data/metrics/ds/ ⏎
com.redhat.rhsa-RHEL7.ds.xml.bz2 ⏎
Get https://www.redhat.com/security/data/metrics/ds/com.redhat.rhsa-
RHEL7.ds.xml.bz2: ⏎
dial tcp 23.214.47.223:443: i/o timeout
```

A proxy server and/or an alternative location for the CVE file can be defined in the advanced settings for the provider (see Figure 8.1, "Provider Advanced Settings")

8.3.1.4. Non-RHEL Images

If the container image is based on a non-RHEL operating system, the image-inspector container writes the following message:

```
Unable to run OpenSCAP: Unable to get RHEL distribution number: could not
find RHEL dist
```

8.4. TUNING SMARTSTATE ANALYSIS

SmartState Analysis settings are stored in the **`:container_scanning`** section of the **Configuration → Advanced** settings, as follows:

```
:container_scanning:
  :scanning_job_timeout: 20.minutes
  :concurrent_per_ems: 3
```

The default value of **`:concurrent_per_ems`** is 3, which limits the number of concurrent container scans that can be carried out to any particular OpenShift Container Platform provider. This can be increased - with caution - to allow more scans to run concurrently.

8.4.1. Increasing the Number of SmartProxy Workers

The default number of "VM Analysis Collector" (*MiqSmartProxyWorker*) workers per CFME appliance/pod 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 or pods in the provider zones exclusively running the SmartState Analysis and SmartProxy roles.

---

[13] 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

CHAPTER 9. MONITORING

CloudForms 4.6 is capable of monitoring the health of an OpenShift Container Platform cluster by receiving alerts created by Prometheus, but if CloudForms is to accurately and reliably provide inventory, utilization metrics, reporting and automation, then its health should also be monitored to ensure reliable operation.

9.1. MONITORING OF OPENSHIFT CONTAINER PLATFORM

CloudForms 4.6 can receive and process alerts generated by Prometheus in an OpenShift Container Platform 3.7 and later cluster.

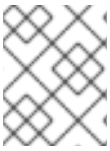
NOTE

It should be noted that Prometheus is supplied as a Technology Preview feature in OpenShift Container Platform 3.7 and 3.9, and that interaction with Prometheus monitoring is also a CloudForms 4.6 Technology Preview feature.

9.1.1. OpenShift Node Exporter

To be able to generate Prometheus alerts on node performance (such as CPU utilization), the node-exporter pod must run on each node. The following instructions describe how to create a new project for the node-exporter and install it onto each node:

```
oc adm new-project openshift-metrics-node-exporter ⏎
   --node-selector='zone=default'
oc project openshift-metrics-node-exporter
oc create  ⏎
   -f https://raw.githubusercontent.com/openshift/origin/master/ ⏎
      examples/prometheus/node-exporter.yaml ⏎
   -n openshift-metrics-node-exporter
oc adm policy add-scc-to-user -z prometheus-node-exporter ⏎
   -n openshift-metrics-node-exporter hostaccess
```



NOTE

The label supplied with the **--node-selector** switch should be applied to all nodes in the cluster

9.1.2. Defining Prometheus Alerts

Alerts can be defined in Prometheus by editing the **prometheus** ConfigMap in the **openshift-metrics** project (as long as Prometheus has been installed in OpenShift Container Platform)

```
oc project openshift-metrics
oc edit cm prometheus
```

Prometheus alerts to be consumed by CloudForms are defined with a "target" of either a node or a provider (the *miqTarget* annotation). Alert rules are defined within the context of a named rule group, for example the following alert in the *custom_rules* group will trigger if an OpenShift Container Platform node is down:

```
prometheus.rules: |
```

```
groups:
- name: custom-rules
  interval: 30s # defaults to global interval
  rules:
  - alert: "Node Down"
    expr: up{job="kubernetes-nodes"} == 0
    annotations:
      miqTarget: "ContainerNode"
      url: "https://www.example.com/node_down_fixing_instructions"
      description: "Node {{ $labels.instance }} is down"
    labels:
      severity: "error"
```

Each rule has a number of parameters, as follows:

- **alert:** - the name of the alert as seen in the Prometheus and CloudForms Monitoring WebUI consoles
- **expr:** - the promQL expression to be evaluated to determine whether the alert should fire. [14]

9.1.2.1. Annotations

An alert can have several annotations, although the following three should always be defined:

- **miqTarget:** this should be "ContainerNode" or "ExtManagementSystem", and determines whether the alert should be associated with an individual node, or the entire OpenShift Container Platform provider.
- **url:** a URL to a web page containing a "Standard Operating Procedure" (SOP) describing how to fix the problem.
- **description:** a more complete description of the alert. The description text can include substitution variables that can insert label values defined with the alert (for example **`{{ $labels.<label_name> }}`**), or the value of a counter (for example **`{{ $value }}`**).

9.1.2.2. Labels

An alert can have several labels, although the following should always be defined:

- **severity:** the severity level of the alert within CloudForms. Valid levels are "error", "warning" or "info".

9.1.2.3. Triggering Prometheus to Reload its Configuration

Once the ConfigMap has been edited, the prometheus process in the Prometheus container must be triggered with a SIGHUP to read the configuration changes. This can be done using the following commands:

```
oc rsh -c prometheus prometheus-0 bash
bash-4.2$ kill -SIGHUP 1
bash-4.2$ exit
```

9.1.2.4. Example Alerts

The following alert definition is useful to test whether the alerting functionality has been configured correctly. It should raise an alert if any of the OpenShift Container Platform nodes are up:

```
- alert: "Node up" # helpful for testing
  expr: up{job="kubernetes-nodes"} == 1
  annotations:
    miqTarget: "ContainerNode"
    url: "https://www.example.com/fixing_instructions"
    description: "ContainerNode {{ $labels.instance }} is up"
  labels:
    severity: "info"
```

The following alert definition should raise an alert if the number of pods running on any node exceeds the threshold defined in the promQL expression:

```
- alert: "Too Many Pods"
  expr: sum(kubelet_running_pod_count) > 50
  annotations:
    miqTarget: "ExtManagementSystem"
    url: "https://www.example.com/too_many_pods_fixing_instructions"
    description: "Too many running pods"
  labels:
    severity: "error"
```

The following alert definition should raise an alert if the number of authenticated logins to the OpenShift Container Platform console exceeds the threshold defined in the promQL expression:

```
- alert: "Too Many Requests"
  expr: rate(authenticated_user_requests[2m]) > 20
  annotations:
    miqTarget: "ExtManagementSystem"
    url: "https://www.example.com/too_many_requests_fixing_instructions"
    description: "Too many authenticated login requests"
  labels:
    severity: "warning"
```

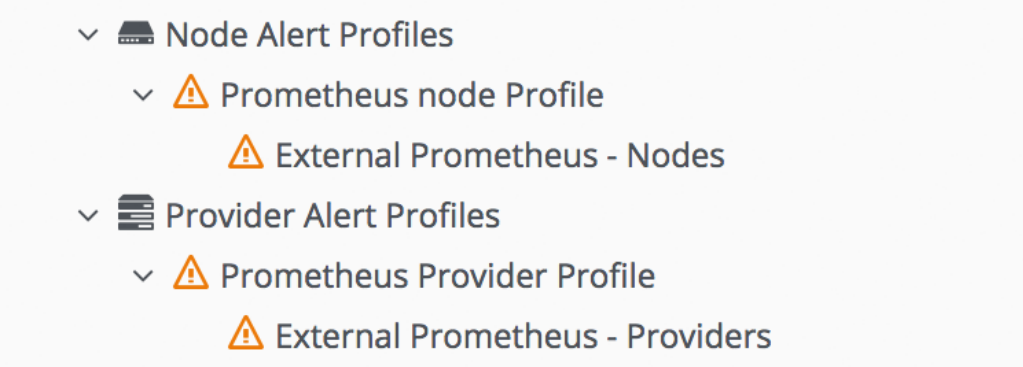
The following alert definition uses metrics returned by the node-exporter, and should raise an alert if node CPU utilization exceeds the threshold defined in the promQL expression:

```
- alert: "Node CPU Usage"
  expr: (100 - (avg by (instance) ⏎
        (irate(node_cpu{app="prometheus-node-exporter", mode="idle"}[5m]))
⏎
        * 100)) > 70
  for: 30s
  annotations:
    miqTarget: "ExtManagementSystem"
    url: "https://www.example.com/high_node_cpu_fixing_instructions"
    description: "{{ $labels.instance }}: CPU usage is above 70% ⏎
          (current value is: {{ $value }})"
  labels:
    severity: "warning"
```

9.1.3. Alert Profiles

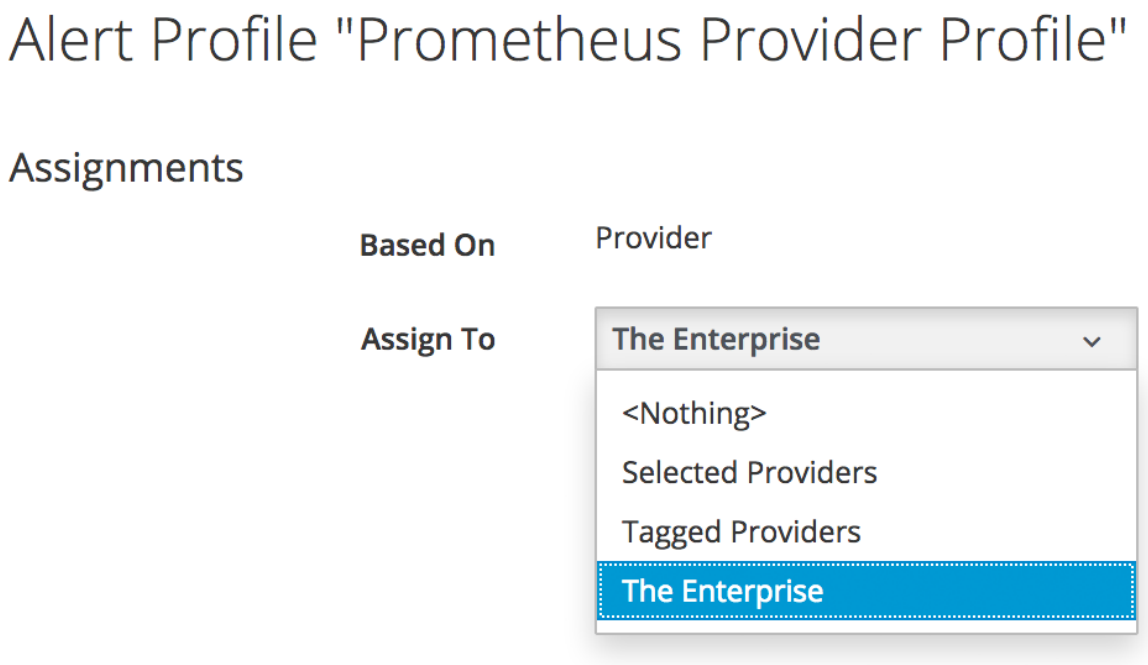
To enable CloudForms to consume and process the Prometheus alerts, a pre-defined Control Alert Profile must be assigned for each of the two alert target types (see Figure 9.1, “Prometheus Alert Profiles”).

Figure 9.1. Prometheus Alert Profiles



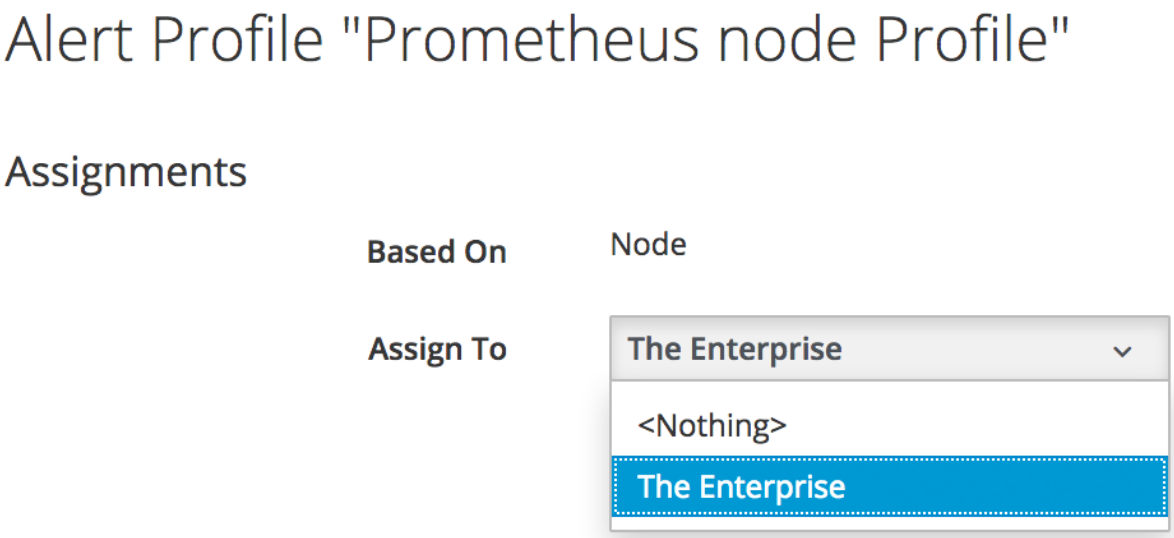
The "Prometheus Provider Profile" can be assigned to selected, tagged or all OpenShift Container Platform providers (see Figure 9.2, “Prometheus Provider Profile Assignments”).

Figure 9.2. Prometheus Provider Profile Assignments



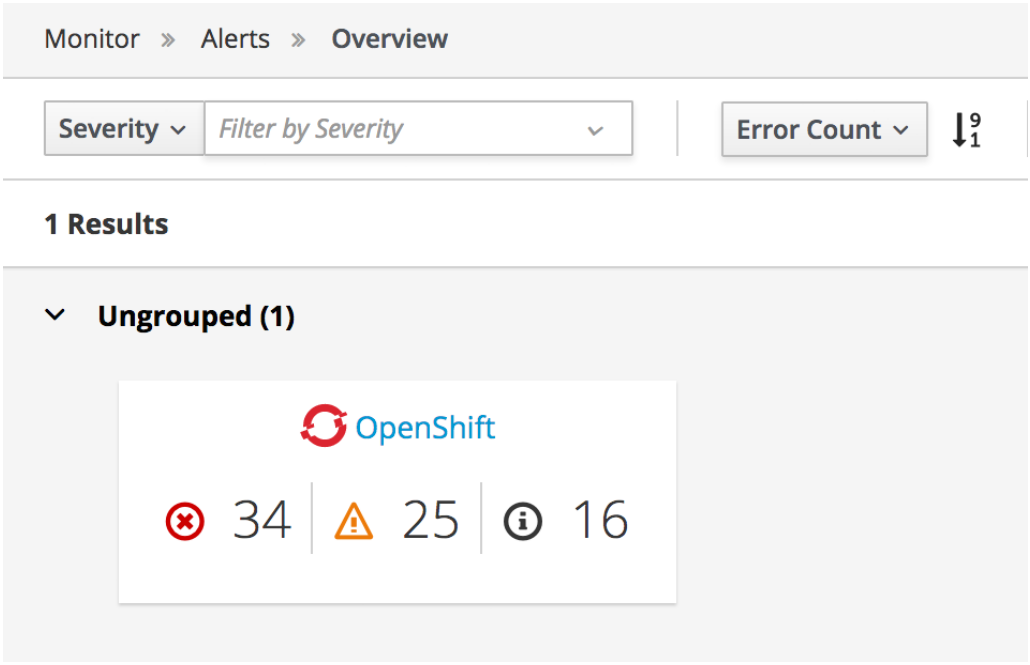
The "Prometheus Node Profile" can be assigned to all OpenShift Container Platform nodes (see Figure 9.3, “Prometheus Node Profile Assignments”).

Figure 9.3. Prometheus Node Profile Assignments



Once the alert profiles have been assigned, any new Prometheus alerts will be visible in the CloudForms **Monitor → Alerts** WebUI page (see Figure 9.4, "Prometheus Alerts Visible in CloudForms")

Figure 9.4. Prometheus Alerts Visible in CloudForms



9.2. MONITORING OF CLOUDFORMS

A large CloudForms deployment has many interacting components: several appliances or pods, many worker processes, a message queue and a PostreSQL database.

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

9.2.1. Database Appliance or Pod

The PostgreSQL database 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. Increase pod memory
- 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

9.2.2. CFME 'Worker' Appliances or Pods

Operational limits for "worker" appliances are usually established on a per-appliance basis, and depend on the enabled server roles and number of worker processes. Resource limits for the "cloudforms-backend" StatefulSet replicas must be defined in the YAML definition for the StatefulSet, and so should be set at the maximum level that any of the replicas will require.

The following items should typically be monitored:

9.2.2.1. General Appliance/Pod

- CPU and memory utilization
- Check for message timeouts

9.2.2.2. 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

9.2.2.2.1. Provider Refresh

- Review EMS refresh activity
  - How many refreshes per hour?
  - How long does a refresh take per OpenShift Container Platform cluster?
    - 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?

9.2.2.2.2. 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 container, pod, etc.
- Validate the numbers of Data Collector and Data Processor workers

9.2.2.2.3. 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

9.2.2.2.4. Event Handling

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

9.2.2.2.5. SmartState Analysis

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

9.2.2.2.6. Reporting

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

9.2.3. Control Alerts

Some self-protection policies are available out-of-the-box in the form of control alerts. The following **CFME Operation** alert types are available:

9.2.3.1. Server Alerts

- EVM Server Database Backup Insufficient Space
- EVM Server Exceeded Memory Limit
- EVM Server High /boot Disk Usage
- EVM Server High /home Disk Usage
- EVM Server High /tmp Disk Usage
- EVM Server High /var Disk Usage
- EVM Server High /var/log Disk Usage
- EVM Server High /var/log/audit Disk Usage
- EVM Server High DB Disk Usage
- EVM Server High System Disk Usage
- EVM Server High Temp Storage Disk Usage
- EVM Server Not Responding
- EVM Server Start
- EVM Server Stop
- EVM Server is Master

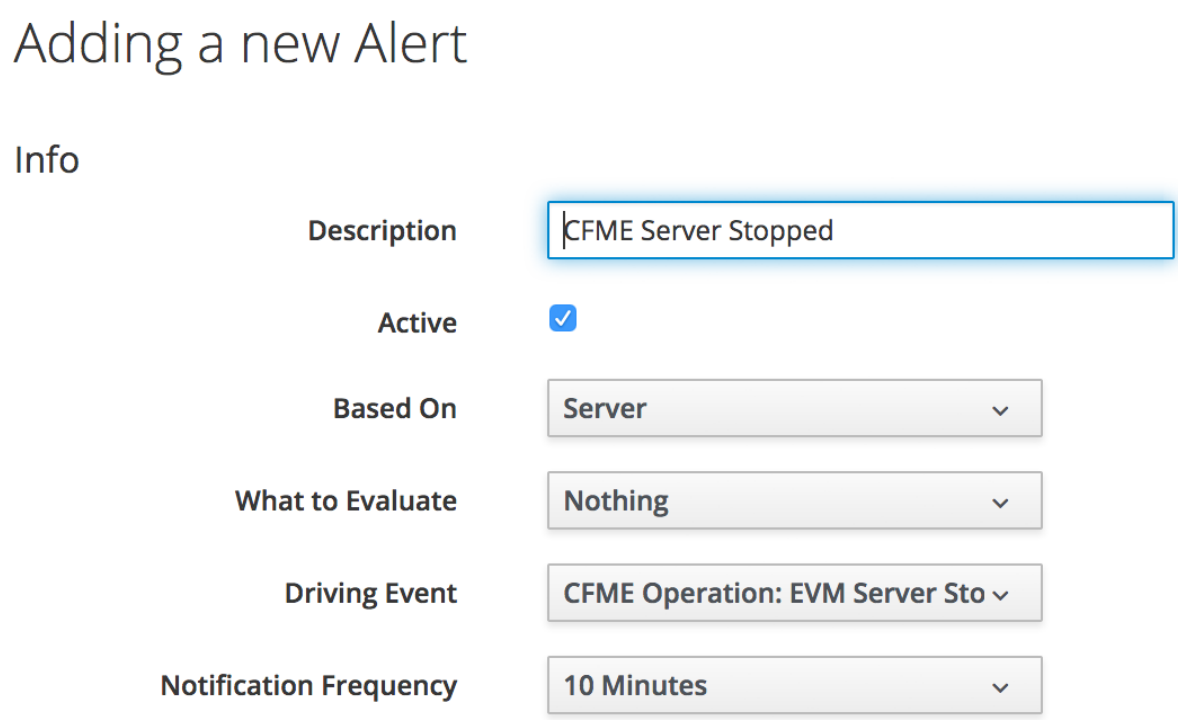
9.2.3.2. Worker Alerts

- 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

Each alert type is configurable to send an email, an SNMP trap, or run an automate instance (see Figure 9.5, “Defining a "Server Stopped" Alert”).

Figure 9.5. Defining a "Server Stopped" Alert



NOTE

EVM Worker Exceeded Uptime Limit, 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.

9.3. 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.

For the podified deployment of CloudForms the logs are written in JSON format to STDOUT in the containers, so Fluentd running on the OpenShift Container Platform nodes will forward them to Elasticsearch automatically (if EFK has been deployed in the cluster).

Although there is no out-of-the-box consolidated logging architecture for the VM appliance version of CloudForms 4.6, it is possible to add CloudForms logs as a source to an external 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.

---

[14] Prometheus alerting rules are described here:
https://prometheus.io/docs/prometheus/latest/configuration/alerting_rules/

CHAPTER 10. 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.

10.1. 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 or "cloudforms" StatefulSet replica is 1, but this can be increased to a maximum of 9. When increasing workers, consideration should be given to the additional CPU and memory requirements placed on the appliance or pod (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.

10.2. 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 or "cloudforms" StatefulSet replicas in their own zone for general user session use. Each of the CFME appliances or StatefulSet replica should be configured with a minimal set of server roles, for example:

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

10.2.1. Load Balancers

Multiple CFME appliances or StatefulSet replicas 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. 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, or the 'memcached' pod. For VM-based CFME appliances 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
```

The session store can be safely left as **`cache`** for the podified deployment of CloudForms as all pods share a single memcache.

CHAPTER 11. DESIGN SCENARIO

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

11.1. ENVIRONMENT TO BE MANAGED

CloudForms is to be installed to manage six OpenShift Container Platform clusters within a large organization. These clusters represent the development, test, and production application environments in each of two geographic regions - Europe and North America.

11.1.1. Virtual Infrastructure

The organization's virtual infrastructure in each geographic region is a Red Hat Virtualization (RHV) 4.2 installation. The OpenShift Container Platform nodes are virtual machines in these RHV environments.

11.1.2. Network Factors

Each geographical site has a single datacenter. There is LAN-speed (<1ms) latency between all points within these datacenters, and 25ms latency between datacenters.

11.1.3. Required CloudForms Functionality

The following capabilities of CloudForms are required:

- Inventory/Insight of all OpenShift Container Platform components such as projects, pods, containers and nodes
- SmartState Analysis and OpenSCAP scanning of container images
- Capacity and Utilization statistics from Hawkular
- Reporting, both regionally and globally

Management of the RHV infrastructure is not required.

11.2. DESIGN PROCESS

The design process usually starts with sizing the region. How many nodes, pods 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 11.1, "Provider Object Numbers - 2 Year Projection"

Table 11.1. Provider Object Numbers - 2 Year Projection

OpenShift cluster	Nodes	Pods	Containers	Images
Europe - Dev	10	500	500	5000
Europe - Test	10	2000	2000	4000

OpenShift cluster	Nodes	Pods	Containers	Images
Europe - Prod	50	10000	10000	20000
North America - Dev	10	600	1000	6000
North America - Test	10	1750	1750	3500
North America - Prod	40	7500	7500	15000

Based on the maximum suggested region sizes shown in Section 3.1.1.2, "Sizing Estimation", it can be estimated that four subordinate regions will be required, each reporting into a single master region. The regions will be as follows:

- **Production (US) Region** managing the US Production OpenShift Container Platform cluster
- **Dev/Test (US) Region** managing the US Development and Test OpenShift Container Platform clusters
- **Production (EMEA) Region** managing the Europe Production OpenShift Container Platform cluster
- **Dev/Test (EMEA) Region** managing the Europe Development and Test OpenShift Container Platform clusters
- **Master Region**

11.2.1. Network Latency

Latency from worker appliance to VMDB should be LAN speed, around 1ms or less. This will dictate where the VMDB servers should be situated, and also the optimum location of the worker CFME appliances. For this design network latency is good, so the VMDB servers and CFME appliances can be placed anywhere in the same datacenter as their managed provider OpenShift Container Platform cluster.

11.2.2. VMDB Servers

The optimum VMDB server for this design will be a CFME 5.9.2 appliance configured as a standalone PostgreSQL server. Although database high availability (HA) has not been specified as an initial requirement, installing a standalone database VM appliance allows for HA to be configured in future if required.

The database servers will be installed in the Red Hat Virtualization virtual infrastructure in their respective data centers. A 500 GByte disk - to be used as the database volume - will be presented to each subordinate region database server from a datastore backed by iSCSI storage. A 1 TByte iSCSI disk will be presented to the master region database server for use as the database volume.

The database servers will each 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 two largest regions by numbers of managed objects will be the Europe and North America Production regions, which will contain 6 & 5 CFME appliances respectively (including WebUI zone appliances). Referring to the table in Appendix A, *Database Appliance CPU Count* shows that the

database servers for these regions will need 4 vCPUs to maintain an idle CPU load under 20%. The database servers for the smaller subordinate regions will need 2 vCPUs. Although the overall processing load for a master region is generally less than for a region managing active providers, the database server for the master region will be given 4 vCPUs.

11.2.3. Zones

A zone should be created per provider (OpenShift Container Platform cluster). There should be a minimum of 2 CFME appliances per zone for resilience, and zones should not span networks. The CFME appliances in each zone will be hosted by the RHV virtual infrastructure in the appropriate data center.

For this design scenario the zones listed in Table 11.2, “Regions and Zones” are proposed.

Table 11.2. Regions and Zones

Region	Region ID	Zones
Production (EMEA) Region	1	WebUI Zone, Production OCP Zone
Dev/Test (EMEA) Region	2	WebUI Zone, Development OCP Zone, Test OCP Zone
Production (US) Region	3	WebUI Zone, Production OCP Zone
Dev/Test (US) Region	4	WebUI Zone, Development OCP Zone, Test OCP Zone
Master	99	WebUI/Reporting Zone

11.2.3.1. WebUI Zones

A WebUI zone containing 2 CFME appliances will be created in each region, each appliance running the following server roles:

- Automation Engine (to process zone events)
- 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 RHV virtual infrastructure in the appropriate data center, in a vLAN accessible from user workstations. User access to them will be via a hardware load-balancer and common Fully-Qualified Domain Name.

11.2.3.2. OpenShift Container Platform Zones

The OCP zones will contain the provider-specific ("worker") CFME appliances. The section Section 3.2.2, “Number of CFME Appliances or Pods in a Zone” suggests a scaling factor of 1 C&U Data Collector worker for every 1500 nodes, pods and/or containers. Scaling to 4 C&U Data Collector

workers per CFME appliance allows for 6000 active objects per CFME appliance, but there should be a minimum of 2 appliances per zone. Table 11.3, “Active Nodes/Pods/Containers” gives the proposed allocation of CFME appliances per zone.

Table 11.3. Active Nodes/Pods/Containers

Zone	Nodes/Pods/Containers	CFME appliances required
Europe - Dev	1010	2
Europe - Test	4010	2
Europe - Prod	20050	4
North America - Dev	1610	2
North America - Test	3510	2
North America - Prod	15040	3

Each of the CFME appliances in these zones should run 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 these zones will also be hosted by the RHV virtual infrastructure in the appropriate data center. The worker count for the C & U Data Collectors should be increased to 4 on each CFME appliance; all other worker counts shoud be left at their defaults (pending further in-operation worker tuning). The `:hawkular_force_legacy` parameter should be set to `true` in the **Advanced** settings on each CFME appliance.

The OpenShift providers will be moved into these zones. Further appliances may need to be added to these zones if the number of managed objects increases.

11.2.3.3. Master Region WebUI/Reporting Zone

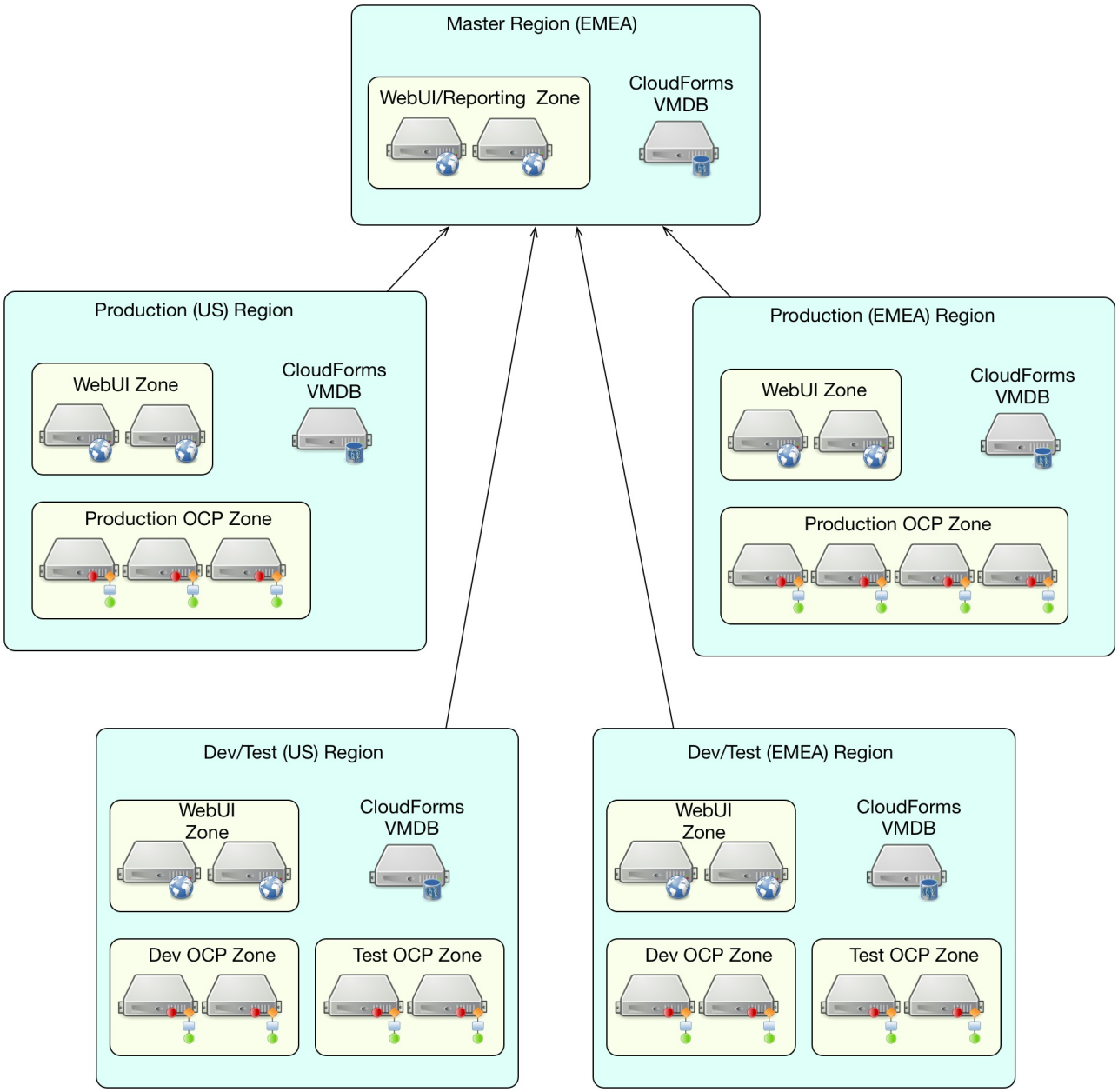
A WebUI/Reporting zone containing 2 CFME appliances will be created in the master region, each appliance running the following server roles:

- Automation Engine (to process zone events)
- 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 RHV virtual infrastructure in the Europe data center, in a vLAN accessible from user workstations. User access to them will be via a hardware load-balancer and common Fully-Qualified Domain Name.

The proposed zone design is shown in Figure 11.1, “Regions and Zones”.

Figure 11.1. Regions and Zones



CHAPTER 12. CONCLUSION

As can be seen from the previous chapters, the architecture of CloudForms is scalable to manage very large OpenShift Container Platform clusters.

- 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

CloudForms can be deployed as a VM appliance outside of OpenShift Container Platform, or running podified within the managed OpenShift Container Platform cluster itself.

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

Number of CFME Appliances in Region	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
Brett Thurber	Engineering Manager	Review
Jason Frey	ManageIQ/CloudForms Chief Architect	Review

APPENDIX C. REVISION HISTORY

Revision 1.0-0 **June, 2018** **Peter McGowan**

- **First release**