



EMC® Watch4net®

Version 6.3u2

Load Balancer Administration Guide

P/N: pending

Rev : 46184

EMC²

Copyright ©2013 - 2014 EMC Corporation. All rights reserved. Published in the USA.

Published April, 2014

EMC believes the information in this publication is accurate as of its publication date. The information is subject to change without notice.

The information in this publication is provided as is. EMC Corporation makes no representations or warranties of any kind with respect to the information in this publication, and specifically disclaims implied warranties of merchantability or fitness for a particular purpose. Use, copying, and distribution of any EMC software described in this publication requires an applicable software license.

EMC², EMC, and the EMC logo are registered trademarks or trademarks of EMC Corporation in the United States and other countries. All other trademarks used herein are the property of their respective owners.

For the most up-to-date regulatory document for your product line, go to EMC Online Support (<https://support.emc.com>).

Contents

Preface	iii
Purpose	iii
Intended Audience	iii
Prerequisites	iii
Document Organization	iv
Where to get help	v
1 Load Balancer Architecture	1
2 Collector Manager Configuration	3
3 Load Balancer Configuration	5
1 Arbiter Example	6
2 Configuration Tags Explanation	7
2.1 Other configuration options	8
3 Balancer Connector Example	9
4 Configuration Tags Explanation	9
4.1 Other configuration options	10
4 Decision Managers	11
1 LoadFactor	11
2 RoundRobin	11
5 Balancing list reconstruction	13
1 Reconstruction Script	13
2 Reconstruction Example	14
3 Configuration Tags Explanation	14

Preface

Purpose

The *Load Balancer* is a collecting component for *APG* that allows variables from one or multiple sources to be distributed across multiple end points (I.E. Backends, Socket collector, ...) with the guarantee that once a variable has been distributed, it will stay on the same end point.

Intended Audience

This guide is intended for advanced *APG* Administrators that have a good knowledge of the *APG* collecting framework.

Prerequisites

The *Load Balancer* can only be used within an instance of *Collector Manager*. Thus, to be able to use this collecting component, you must have a working *Collector Manager* ready.

The *Load Balancer* is designed to integrate into the standard *APG* collecting/backend structure. Therefore, components such as filters, connectors and the backend can be used along with the *Load Balancer* to achieve the desired results.

Document Organization

1. <i>Load Balancer Architecture</i>	Architecture of the <i>Load Balancer</i> .
2. <i>Collector Manager Configuration</i>	How to configure the <i>Collector Manager</i> to include a <i>Load Balancer</i> .
3. <i>Load Balancer Configuration</i>	How to configure the <i>Load Balancer</i> itself.
4. <i>Decision Managers</i>	Explanation of the available Decision Managers.
5. <i>Balancing list reconstruction</i>	Balancing list reconstruction

Where to get help

EMC support, product, and licensing information can be obtained as follows:

PRODUCT INFORMATION - For documentation, release notes, software updates, or information about EMC products, go to EMC Online Support at:

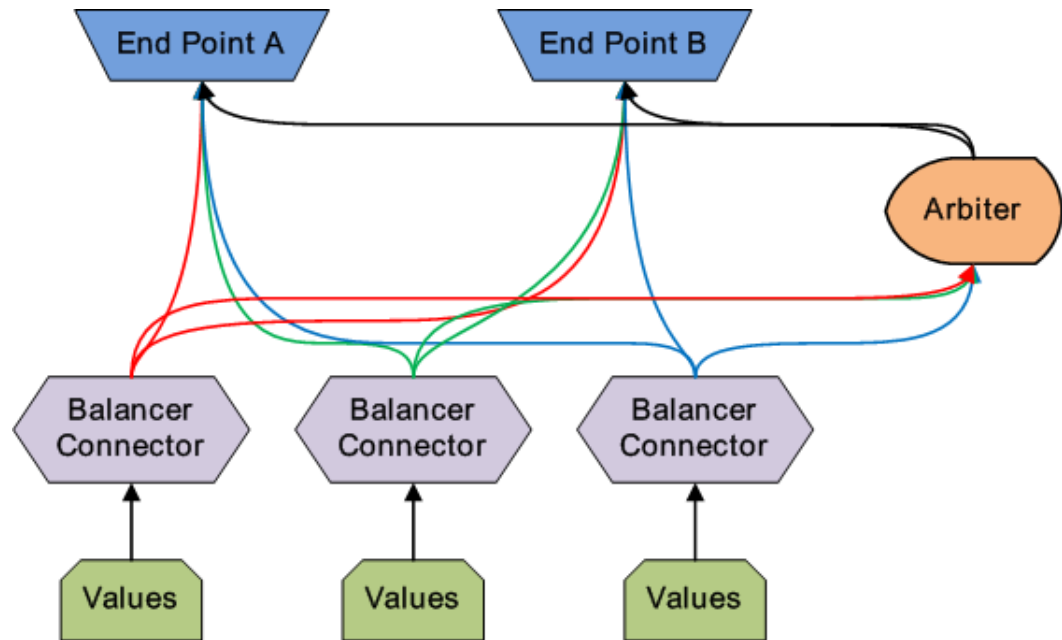
<https://support.emc.com>

TECHNICAL SUPPORT - Go to EMC Online Support and click Service Center. You will see several options for contacting EMC Technical Support. Note that to open a service request, you must have a valid support agreement. Contact your EMC sales representative for details about obtaining a valid support agreement or with questions about your account.

1

Load Balancer Architecture

The *Load Balancer* requires two components to work correctly, the *Arbiter* and *Balancer Connector*. The *Arbiter* runs in its own *Collector Manager* and is the main component responsible to balance any unknown variables and spread the connecting configuration and updated balancing list to all the connected *Balancer Connector*. The *Balancer Connector* replaces the standard connector of any *Collector Manager* and asks the connected *Arbiter* for the up to date balancing list while sending the new variables to the *Arbiter*.



2

***Collector Manager* Configuration**

To use a *Load Balancer* instance, please follow this *Collector Manager* configuration file example:

```
<connectors>
...
  <!-- This is the Load Balancer Arbiter -->
  <connector enabled="true" name="Arbiter" type="Load-Balancer" config="Load-Balancer/Default/
conf/arbiter.xml" />
...
</connectors>
```

From the *Collector Manager* point of view, an *Arbiter* and an *Balancer Connector* is the same thing. The *Load Balancer* configuration will determine which behavior will be chosen by the component.

3

Load Balancer Configuration

1 Arbiter Example

The following example is the default *Load Balancer* configuration for the *Arbiter*:

```
<?xml version="1.0" encoding="UTF-8"?>
<balancing-configuration ...>

  <persistence>
    <file-persistence folder="data" />
  </persistence>

  <arbiter decision-manager="LoadFactor">
    <balancing-properties property="device" allow-missing-property="false" />

    <end-points name="Backend1">
      <load-factor-service validate-ssl="true">
        <username>admin</username>
        <password>changeme</password>
        <url>https://localhost:48443/Backends/APG-Backend/Default?wsdl</url>
      </load-factor-service>
      <filter-configurations>
        <location>../../FailOver-Filter/Default</location>
        <configurations>conf/failover-backend-1.xml</configurations>
      </filter-configurations>
      <connector-configuration type="Socket-Connector">
        <location>../../Collector-Manager/Default</location>
        <configurations>conf/socket-backend-1.xml</configurations>
      </connector-configuration>
    </end-points>

    <end-points name="Backend2">
      <load-factor-service validate-ssl="true">
        <username>admin</username>
        <password>changeme</password>
        <url>https://localhost:48443/Backends/APG-Backend/INSTANCE?wsdl</url>
      </load-factor-service>
      <filter-configurations>
        <location>../../FailOver-Filter/Default</location>
        <configurations>conf/failover-backend-2.xml</configurations>
      </filter-configurations>
      <connector-configuration type="Socket-Connector">
        <location>../../Collector-Manager/Default</location>
        <configurations>conf/socket-backend-2.xml</configurations>
      </connector-configuration>
    </end-points>
  </arbiter>
</balancing-configuration>
```

```
</connector-configuration>
</end-points>
</arbiter>
</balancing-configuration>
```

2 Configuration Tags Explanation

- **persistence**: The persistence tag defines what kind of persistence will be applied to the balancing lists

The available persistence are:

- **file-persistence**: Specify that the balancing lists will be kept in a **folder** relative to the instance of the *Load Balancer*.
- **no-persistence**: Specify that the balancing lists will be kept only in the runtime memory and flushed after a restart. This method must only be used when the destination doesn't keep an history like the Alerting Backends.
- **arbiter**: The arbiter tag defines that the current *Load Balancer* will act as an *Arbiter*. The optional **decision-manager** attribute defines how the *Arbiter* will take the balancing decision. The available managers are: LoadFactor (default) or RoundRobin.
 - **balancing-properties**: (1 to n) Specify the balancing properties that will be used to create unique keys. A variable that doesn't contains a property that isn't allowed to be missing will be discarded.
 - **end-points**: (0 to n) Defines the end points with unique **names** that will received the balanced variables.
 - * **load-factor-service**: If the decision manager is LoadFactor, the end point should define the location of its load factor service. This tag won't have any effect with any other decision managers. The instance name of the WebService URL will depends on the instance name of the actual Backend on the remote host.
 - * **filter-configurations**: (0 to n) Set the internal end point collect chain filtering order. It's not recommended to do data enrichment at this point but a FailOver Filter is a good idea. A component should define the **type** and must provide the **location** of the actual installed component relative to the *Load Balancer*. The given **configurations** are the configure files local to the *Load Balancer* needed by the component. The first entry is the actual configuration file while the next entries are needed when the primary configuration refer to other configuration files in order to send them also.

- * **connector-configuration:** (0 or 1) Specify how the variables will be sent to the end point and is configured like a filter. The *Arbiter* is not forced to have an internal collect chain if for example the server manager the *Arbiter* doesn't have access to the end point servers. In this case, the *Arbiter* can only create the balancing lists and it will be up to the *Balancer Connector* to send the variables.

2.1 Other configuration options

```
<balancing-properties property="source" allow-missing-property="false" />
<balancing-properties property="propertyA" allow-missing-property="false" />
<balancing-properties property="device" allow-missing-property="false" />

<restricted-on property="source">
  <when value="VMWareCollector">
    <restricted-on property="propertyA">
      <when value="property_value_1">
        <end-points name="End_Point_A"/>
      </when>
      <when value="property_value_2">
        <end-points name="End_Point_B"/>
      </when>
      <otherwise>
        <end-points name="End_Point_A"/>
      </otherwise>
    </restricted-on>
  </when>
  <when value="TextCollector">
    <end-points name="End_Point_B" />
  </when>
  <otherwise>
    <end-points name="End_Point_C" />
  </otherwise>
</restricted-on>

<end-points ...>
```

It's possible to actually control where the variables will be sent by adding to restrictions to one or more balancing properties. The properties and end point names must be defined in the rest of the configuration and since the *Load Balancer* is not meant to filter out variables, the **otherwise** tag is mandatory.

3 Balancer Connector Example

The following example is the default *Load Balancer* configuration for the *Balancer Connector*:

```
<?xml version="1.0" encoding="UTF-8"?>
<balancing-configuration ...>

  <persistence>
    <file-persistence folder="data" />
  </persistence>

  <balancer-connector>
    <arbiter-balancing-service validate-ssl="true">
      <username>admin</username>
      <password>changeme</password>
      <url>https://localhost:48443/Collecting/Collector-Manager/Default/Arbiter/?wsdl</url>
    </arbiter-balancing-service>

    <update-check-time>60s</update-check-time>

    <arbiter-connector>
      <filter-configurations>
        <location>../../FailOver-Filter/Default</location>
        <configurations>conf/failover-arbiter.xml</configurations>
      </filter-configurations>
      <connector-configuration type="Socket-Connector">
        <location>../../Collector-Manager/Default</location>
        <configurations>conf/socket-arbiter.xml</configurations>
      </connector-configuration>
    </arbiter-connector>
  </balancer-connector>
</balancing-configuration>
```

4 Configuration Tags Explanation

- **persistence:** The persistence tag defines what kind of persistence will be applied for the received balancing list. Behave just like the *Arbiter*.
- **balancer-connector:** The balancer-connector tag defines that the current *Load Balancer* will act as a *Balancer Connector*.

- **arbiter-balancing-service**: Specify the connection properties to the *Arbiter* balancing service.
- **update-check-time**: Specify how much time between two update requests for the newest configurations and newest balancing lists from the *Arbiter* (Default is 60 seconds).
- **arbiter-connector**: Internal collect chain for the *Arbiter*. Behave just like the internal collecting chain of an end point.

4.1 Other configuration options

```
</arbiter-connector>

<end-points name="Backend1" >
  <load-factor-service validate-ssl="true">
    <username>admin</username>
    <password>changeme</password>
    <url>https://localhost:48443/Backends/APG-Backend/Default?wsdl</url>
  </load-factor-service>
  <filter-configurations>
    <location>../../FailOver-Filter/Default</location>
    <configurations>conf/failover-backend-1.xml</configurations>
  </filter-configurations>
  <connector-configuration type="Socket-Connector">
    <location>../../Collector-Manager/Default</location>
    <configurations>conf/socket-backend-1.xml</configurations>
  </connector-configuration>
</end-points>

</balancer-connector>
```

It's possible to define local end points configuration (mandatory if the *Arbiter* doesn't have connectors configuration to send). The name of the end points must match one of the end points of the *Arbiter*. A configuration for an unknown end point will be discarded.

4

Decision Managers

When using a configuration of an *Arbiter*, the Decision Manager will be responsible to choose which end point will receive each new key. The available managers are:

1 LoadFactor

The LoadFactor Decision Manager will prioritize the selection of the end points with the lowest load factor returned by their configured load-factor-service. It's important to note that the *Arbiter* will never block any new key event if all the end points have load factor higher or equal to 1. Furthermore, the distribution algorithm has been designed to be performant for the selection in order to handle correctly a big group of new keys. This algorithm will not prevent the most loaded end point to receive new keys but will do so in a ratio really small compared to the less loaded end points.

2 RoundRobin

The RoundRobin Decision Manager will simply loop over the configured end points when choosing a destination for each new key without any regards to the limitation of the chosen end point.

Balancing list reconstruction

If an error is detected when the persistence is loaded or if a change is detected in the balancing property lists, a reconstruction will be asked. A reconstruction should also be launched if an operation is made on a database (split, mass deletion, ...). A reconstruction will go fetch all the possible unique keys in the end points databases in order to recreate the balancing list and still guarantee the data will remain where it should be. Before launching a reconstruction, stop all the *Balancer Connector*, wipe their local persistence files and stop the *Arbiter*.

1 Reconstruction Script

In order to launch a reconstruction, use the script *load-balancer-reconstruction* available in the *bin* folder of the installed *Load Balancer*. The script expect the *Arbiter* configuration (typically *conf/arbiter.xml*) and the reconstruction configuration (typically *conf/reconstruction.xml*).

2 Reconstruction Example

The following example is the default *Load Balancer* configuration for the reconstruction of the balancing lists:

```
<?xml version="1.0" encoding="UTF-8"?>
<reconstruction ...>
  <!-- Optional sources filtering -->
  <!--
  <sources></sources>
  -->

  <end-points name="Backend1">
    <datasource username="apg" password="watch4net" driver="com.mysql.jdbc.Driver" url="
jdbc:mysql://localhost:53306/apg" />
  </end-points>

  <end-points name="Backend2">
    <datasource username="apg" password="watch4net" driver="com.mysql.jdbc.Driver" url="
jdbc:mysql://localhost:53306/apg" />
  </end-points>
</reconstruction>
```

3 Configuration Tags Explanation

- **sources:** (0 to n) If an end point is not dedicated to the data coming for the *Load Balancer*. It's a good idea to filter on the sources that are going through the *Load Balancer*.
- **end-points:** Must have one entry per end point defines in the running *Arbiter* configuration. The **datasource** contains the information needed for the database connection.